UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

GOOGLE LLC,
Petitioner,

v.

VIRTAMOVE, CORP.,
Patent Owner.

_____

Case No. IPR2025-00487
Patent No. 7,519,814

_____

DECLARATION OF SAMRAT BHATTACHARJEE, PH.D.

Google Exhibit 1003
Google v. VirtaMove

TABLE OF CONTENTS (CONDENSED)

I. PERSONAL AND PROFESSIONAL BACKGROUND 1

II. MATERIALS REVIEWED AND CONSIDERED 6

A. Public Accessibility of Non-Patent Prior Art Exhibits 6

III. MY UNDERSTANDING OF PATENT LAW 6

A. Anticipation 8

B. Obviousness 9

IV. UNPATENTABILITY GROUNDS 12

V. THE '814 PATENT 13

A. Background and Specification 13

B. Person of Ordinary Skill in the Art ("POSA") 16

C. Prosecution History 17

D. Challenged Claims 18

E. The Independent Claims Are Not Supported By Provisional Application No. 60/502,619 19

VI. CLAIM INTERPRETATION 20

A. Terms with Agreed Upon Constructions 20

1. Preamble of Claim 1 20

2. "*container*" (claims 1, 2, 4, 6, 9, 10, 13, 14) 20

3. "*a kernel a set of associated local system files* [sic]" (claim 1) 21

4. "*processor*" (claim 1) 21

5. "*servers*" (claim 1) 21

6. "*within a container*" (claim 1) 21

7. Terms whose antecedent refer to the same entity 22

B. Terms with Disputed Constructions 22

1. "*operating system*" (claims 1, 10) 22

2. "*kernel*" / "*operating system kernel*" (claim 1) 23

3. "*disparate computing environments*" (claim 1) 23

4. "*service*" (claims 1, 14) 24

5. "*at least some of the different operating systems*" (claim 1) 24

"*at least some of the plurality of different operating systems*" (claim 1)24

6. "*memory accessible to at least some of the servers*" (claim 1).....25

7. "*local kernel residing permanently on one of the servers*" (claim 1) 25

8. "*secure containers of application software*" (claim 1) 25

9. "*an operating system's root file system*" (claim 1) 25

VII. Application of Prior Art to Challenged Claims 27

A. GROUND 1: Claims 1-4, 7-11, 14, and 16-30 would have been obvious to POSAs over the Blaser-Calder combination. 27

1. Blaser (EX1005).....27

2. Calder (EX1006) 30

3. The Blaser-Calder Combination.....37

4. Mapping of Blaser-Calder to Claims 1-4, 7-11, 14, and 16-30 58

a. Claim 1 58

b. Claim 2: "[C]laim 1, wherein each container has an execution file associated therewith for starting the one or more applications." 152

c. Claim 3: "[C]laim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed." 156

d. Claim 4: "[C]laim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step [[1A]]. 159

e. Claim 7: "[C]laim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container." 161

f. Claim 8: "[C]laim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers." 162

g. Claim 9: "[C]laim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed."....166

h. Claim 10: "[C]laim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof." 178

i. Claim 11: "[C]laim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location."....181

j. Claim 14 182

k. Claim 16 194

l. Claim 17 195

m. Claim 18: "[C]laim 17 wherein the target server and the graphical user interface are at remote locations."....203

n. Claim 19: "[C]laim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server." 208

o. Claim 20: "[C]laim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server." 210

p. Claim 21: "[C]laim 20 further comprising starting a distributed software application." 213

q. Claim 22: "[C]laims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server." 216

r. Claim 23: "[C]laim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server." 219

s. Claim 24: "[C]laim 17 further comprising creating a user account for the service."....222

t. Claim 25: "[C]laim 17, further comprising the step of installing files specific to the selected application on the selected server." 224

u. Claim 26: "[C]laim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed." 224

v. Claim 27....226

w. Claim 28: "[C]laim 27 further comprising separating icon representing the service from the icon representing the server." 231

x. Claim 29: "[C]laim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service."....232

y. Claim 30: "[C]laim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation." 233

B. GROUND 2: Claims 5-6, 12-13, 15, and 31-34 would have been obvious to POSAs over the Blaser-Calder-Schmidt combination. 234

1. Schmidt-449 (EX1007) 234

2. The Blaser-Calder-Schmidt Combination....236

3. Mapping of Blaser-Calder-Schmidt to Claims 5-6, 12-13, 15, and 31-34....238

a. Claim 5: "[C]laim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container." 238

b. Claim 6: "[C]laim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address." 239

c. Claim 12: "[C]laim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1." 243

d. Claim 13: "[C]laim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application." 244

e. Claim 15 253

f. Claim 31 255

g. Claim 32: "[C]laim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources." 264

h. Claim 33: "[C]laim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address [sic]." 266

i. Claim 34 267

VIII. APPENDIX: CLAIM LISTING 1

TABLE OF CONTENTS (FULL)

I. PERSONAL AND PROFESSIONAL BACKGROUND 1

II. MATERIALS REVIEWED AND CONSIDERED 6

A. Public Accessibility of Non-Patent Prior Art Exhibits 6

III. MY UNDERSTANDING OF PATENT LAW 6

A. Anticipation 8

B. Obviousness 9

IV. UNPATENTABILITY GROUNDS 12

V. THE '814 PATENT 13

A. Background and Specification 13

B. Person of Ordinary Skill in the Art ("POSA") 16

C. Prosecution History 17

D. Challenged Claims 18

E. The Independent Claims Are Not Supported By Provisional Application No. 60/502,619 19

VI. CLAIM INTERPRETATION 20

A. Terms with Agreed Upon Constructions 20

1. Preamble of Claim 1 20

2. "*container*" (claims 1, 2, 4, 6, 9, 10, 13, 14) 20

3. "*a kernel a set of associated local system files* [sic]" (claim 1) 21

4. "*processor*" (claim 1) 21

5. "*servers*" (claim 1) 21

6. "*within a container*" (claim 1) 21

7. Terms whose antecedent refer to the same entity 22

B. Terms with Disputed Constructions 22

1. "*operating system*" (claims 1, 10) 22

2. "*kernel*" / "*operating system kernel*" (claim 1) 23

3. "*disparate computing environments*" (claim 1) 23

4. "*service*" (claims 1, 14) 24

5. "*at least some of the different operating systems*" (claim 1) 24

"*at least some of the plurality of different operating systems*" (claim 1)24

6. "*memory accessible to at least some of the servers*" (claim 1).....25

7. "*local kernel residing permanently on one of the servers*" (claim 1) 25

8. "*secure containers of application software*" (claim 1) 25

9. "*an operating system's root file system*" (claim 1) 25

VII. Application of Prior Art to Challenged Claims 27

A. GROUND 1: Claims 1-4, 7-11, 14, and 16-30 would have been obvious to POSAs over the Blaser-Calder combination. 27

1. Blaser (EX1005).....27

2. Calder (EX1006) 30

3. The Blaser-Calder Combination.....37

4. Mapping of Blaser-Calder to Claims 1-4, 7-11, 14, and 16-30 58

a. Claim 1 58

i. [1PREA].....59

(1) "1. In a system having a plurality of servers with operating systems that differ".....59

(2) "operating in disparate computing environments".....63

(3) "wherein each server includes a processor and an operating system" 66

(4) "[OS] including a kernel [and] a set of associated local system files compatible with the processor" 69

ii. [1PREB].....79

(1) "a method of providing at least some of the servers in the system with secure, executable, applications" 79

(2) "[applications] related to a service" 81

(3) "wherein the applications are executed in a secure environment" 83

(4) "wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service" 85

iii. [1A] 95

(1) "storing in memory accessible to at least some of the servers a plurality of secure containers of application software" 95

a) Blaser-Calder's layers (containers) are "stor[ed] in memory accessible to…the servers" 95

b) Blaser-Calder's layers are "containers of application software" 99

c) Blaser-Calder's layers are "secure containers of application software" 118

(2) "each container comprising one or more of the executable applications" 124

(3) "and a set of associated system files required to execute the one or more applications" 127

(4) "for use with a local kernel residing permanently on one of the servers" 130

iv. [1B] "wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems," 133

v. [1C] "the containers of application software excluding a kernel," 135

vi. [1D] "wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server," 138

vii. [1E] "wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server," 140

viii. [1F] “and wherein the application software cannot be shared between the plurality of secure containers of application software,” 144

ix. [1G] “and wherein each of the containers has a unique root file system that is different from an operating system’s root file system.” 146

b. Claim 2: “[C]laim 1, wherein each container has an execution file associated therewith for starting the one or more applications.” 152

c. Claim 3: “[C]laim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.” 156

d. Claim 4: “[C]laim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step [[1A]]. 159

e. Claim 7: “[C]laim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.” 161

f. Claim 8: “[C]laim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.” 162

g. Claim 9: “[C]laim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”.....166

h. Claim 10: “[C]laim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.” 178

i. Claim 11: “[C]laim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.”.....181

j. Claim 14 182
   i. [14PRE] "[C]laim 1 comprising the step of creating containers prior to said step of storing containers in memory" 182
   ii. [14A] "wherein containers are created by: a) running an instance of a service on a server;" 183
   iii. [14B] "b) determining which files are being used; and," 187
   iv. [14C] "c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files." 191
k. Claim 16 194
   i. [16A] "[C]laim 1 comprising the step of creating containers prior to said step of storing containers in memor," 194
   ii. [16B] "wherein a step of creating containers includes: using a skeleton set of system files as a container starting point and installing applications into that set of files." 194
l. Claim 17 195
   i. [17A] "[C]laim 1 further comprising installing a service on a target server selected from one of the plurality of servers," 195
   ii. [17B] "wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server." 197
m. Claim 18: "[C]laim 17 wherein the target server and the graphical user interface are at remote locations." 203
n. Claim 19: "[C]laim 18, wherein the graphical user interface is installed on a computing platform, and wherein

the computing platform is a different computing platform than the target server." 208

o. Claim 20: "[C]laim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server." 210

p. Claim 21: "[C]laim 20 further comprising starting a distributed software application." 213

q. Claim 22: "[C]laims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server." 216

r. Claim 23: "[C]laim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server." 219

s. Claim 24: "[C]laim 17 further comprising creating a user account for the service." 222

t. Claim 25: "[C]laim 17, further comprising the step of installing files specific to the selected application on the selected server." 224

u. Claim 26: "[C]laim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed." 224

v. Claim 27 226

i. [27A] "[C]laim 1, further comprising de-installing a service from a server, comprising:" 226

ii. [27B] "displaying the icon representing the service; [and] displaying the icon representing the server on which the service is installed; and" 226

iii. [27C] "utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed." 229

w. Claim 28: "[C]laim 27 further comprising separating icon representing the service from the icon representing the server." 231

x. Claim 29: "[C]laim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service." 232

y. Claim 30: "[C]laim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation." 233

B. GROUND 2: Claims 5-6, 12-13, 15, and 31-34 would have been obvious to POSAs over the Blaser-Calder-Schmidt combination. 234

1. Schmidt-449 (EX1007) 234

2. The Blaser-Calder-Schmidt Combination 236

3. Mapping of Blaser-Calder-Schmidt to Claims 5-6, 12-13, 15, and 31-34 238

a. Claim 5: "[C]laim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container." 238

b. Claim 6: "[C]laim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address." 239

c. Claim 12: "[C]laim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1." 243

d. Claim 13: "[C]laim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application." 244

e. Claim 15 253

i. [15A] “[C]laim 14 comprising the steps of: assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;” 253

ii. [15B] “installing the container on a server; and,” 254

iii. [15C] “testing the applications and files within the container.” 254

f. Claim 31 255

i. [31PRE] “A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:” 255

ii. [31A] “a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, ” 256

iii. [31B] “each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers,” 257

iv. [31C] “each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address;” 258

v. [31D] “wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes,” 259

vi. [31E] “and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system,” 259

vii. [31F] “each container having its own execution file associated therewith for starting one or more applications,” 260

viii. [31G] “in operation, each container utilizing a kernel resident on the server and” 260

ix. [31H] “wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,” 261

x. [31I] “a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.” 262

g. Claim 32: “[C]laim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.” 264

h. Claim 33: “[C]laim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address [sic].” 266

i. Claim 34 267

i. [34A] “[C]laim 31, wherein the run time module performs: monitoring resource usage of applications executing;” 267

ii. [34B] “intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;” 267

iii. [34C] “comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.” 269

VIII. APPENDIX: CLAIM LISTING 1

APPENDIX LISTING OF EXHIBITS

Exhibit	Description
1001	U.S. Patent No. 7,519,814 ("'814 patent")
1002	Prosecution History of U.S. Patent No. 7,519,814
1005	U.S. Patent No. 7,117,495 ("Blaser")
1006	U.S. Patent Application Publication No. 20020066022 ("Calder")
1007	U.S. Patent No. 6,931,449 ("Schmidt-449")
1008	U.S. Patent Application Publication No. 20020095479 ("Schmidt-479")
1009	U.S. Patent Application Publication No. 20020133529 ("Schmidt-529")
1010	U.S. Patent Application Publication No. 20020124072 ("Tormasov")
1011	U.S. Patent Application Publication No. 20010009425
1012	U.S. Patent Application Publication No. 20010018708
1013	U.S. Patent Application Publication No. 20010047472
1014	U.S. Patent Application Publication No. 20010056572
1015	U.S. Patent Application Publication No. 20010029605
1016	U.S. Patent Application Publication No. 20020023158
1017	U.S. Patent Application Publication No. 20020052727
1018	U.S. Patent Application Publication No. 20020143795
1019	U.S. Patent Application Publication No. 20020156612
1020	U.S. Patent Application Publication No. 20020156877
1021	U.S. Patent Application Publication No. 20020174215
1022	U.S. Patent Application Publication No. 20020188718
1023	U.S. Patent Application Publication No. 20020194394
1024	U.S. Patent Application Publication No. 20020194488
1025	U.S. Patent Application Publication No. 20030009408
1026	U.S. Patent Application Publication No. 20030014381
1027	U.S. Patent Application Publication No. 20030014466
1028	U.S. Patent Application Publication No. 20030018717
1029	U.S. Patent Application Publication No. 20030023839
1030	U.S. Patent Application Publication No. 20030041173
1031	U.S. Patent Application Publication No. 20030093688
1032	U.S. Patent Application Publication No. 20030097464
1033	U.S. Patent No. 6,467,052
1034	U.S. Patent Application Publication No. 20030140179
1035	U.S. Patent Application Publication No. 20030154221
1036	U.S. Patent No. 7,159,184
1037	U.S. Patent No. 7,103,745
1038	U.S. Patent No. 6,263,440

Exhibit	Description
1039	U.S. Patent Application Publication No. 20040190534
1040	U.S. Patent No. 7,356,771
1041	U.S. Patent No. 5,903,753
1042	U.S. Patent No. 4,685,125
1043	U.S. Patent No. 5,406,644
1044	U.S. Patent No. 5,421,009
1045	U.S. Patent No. 5,568,630
1046	U.S. Patent No. 5,640,562
1047	U.S. Patent No. 5,657,221
1048	U.S. Patent No. 5,675,831
1049	U.S. Patent No. 5,742,829
1050	U.S. Patent No. 4,742,450
1051	U.S. Patent No. 5,761,669
1052	U.S. Patent No. 5,784,555
1053	U.S. Patent No. 5,835,765
1054	U.S. Patent No. 6,044,465
1055	U.S. Patent No. 6,122,744
1056	U.S. Patent No. 6,195,650
1057	U.S. Patent No. 6,321,323
1058	U.S. Patent No. 6,363,409
1059	U.S. Patent No. 6,453,470
1060	U.S. Patent No. 6,567,767
1061	U.S. Patent No. 6,732,359
1062	U.S. Patent No. 6,874,148
1063	U.S. Patent No. 6,985,937
1064	U.S. Patent No. 7,140,015
1065	U.S. Patent No. 7,185,192
1066	U.S. Patent No. 7,454,440
1067	U.S. Patent No. 8,209,680
1068	U.S. Patent No. 5,926,636
1069	U.S. Patent No. 5,218,530
1070	U.S. Patent No. 6,134,593
1071	U.S. Patent No. 6,918,038
1072	U.S. Patent Application Publication No. 20040034623
1073	U.S. Patent No. 7,461,148
1074	U.S. Patent No. 7,136,800
1075	PCT Application Publication No. WO2002056156

Exhibit	Description
1076	Kamp et al., "Jails: Confining the omnipotent root," Proceedings of the 2nd International SANE Conference (Vol. 43, p. 116) (2000)
1077	Prevelakis et al., "Sandboxing Applications," USENIX Annual Technical Conference, FREENIX Track (pp. 119-126) (2001)
1078	Plaintiff VirtaMove Corp.'s Supplemental Preliminary Disclosure of Asserted Claims and Infringement Contentions, in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Sept. 06, 2024)
1079	Chart re: '814 Patent accompanying Plaintiff VirtaMove Corp.'s Supplemental Preliminary Disclosure of Asserted Claims and Infringement Contentions, in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Sept. 6, 2024)
1080	First Amended Complaint for Patent Infringement Against Google LLC in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (May 21, 2024)
1083	U.S. Patent Application Publication No. 20020095500 ("Schmidt-500")
1084	U.S. Patent Application Publication No. 20050169073
1085	U.S. Provisional Application No. 60/533,388
1086	U.S. Patent No. 5,412,808
1088	Microsoft's Computer Dictionary (5th ed. 2002) (excerpt)
1089	U.S. Patent No. 6,381,742
1090	U.S. Patent Application Publication No. 20030035430
1091	U.S. Patent No. 6,477,624
1092	U.S. Patent No. 5,931,947
1093	U.S. Patent No. 6,148,335
1094	U.S. Patent No. 6,944,790
1095	U.S. Patent No. 6,014,135
1096	U.S. Patent No. 6,282,660
1097	U.S. Patent Application Publication No. 20040153709
1098	U.S. Patent Application Publication No. 20050050389
1099	U.S. Patent Application Publication No. 20040225952
1100	Google LLC's Proposed Claim Terms for Construction in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1101	Plaintiff's Disclosure of Proposed Claim Constructions in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 1, 2024)
1102	Google LLC's Opening Claim Construction Brief in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Oct. 24, 2024)

Exhibit	Description
1103	Plaintiff's Responsive Claim Construction Brief in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Nov. 12, 2024)
1104	Google LLC's Reply Claim Construction Brief in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Nov. 26, 2024)
1105	Plaintiff's Sur-Reply Claim Construction Brief in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 13, 2024)
1106	Joint Claim Construction Statement in *VirtaMove, Corp. v. Google LLC*, 7:24-cv-00033-DC-DTG (W.D. Tex.) (Dec. 18, 2024)
1107	U.S. Provisional Application No. 60/502,619
1108	U.S. Patent Application Publication No. 20050188268
1109	U.S. Patent Application Publication No. 20020140743
1110	U.S. Patent Application Publication No. 20020143906 ("Tormasov-906")
1111	U.S. Patent No. 7,076,633 ("Tormasov-633")
1112	U.S. Patent Application Publication No. 20060089950 ("Tormasov-950")
1113	U.S. Patent No. 7,222,132 ("Tormasov-132")
1114	U.S. Patent Application Publication No. 20020147815 ("Tormasov-815")
1115	U.S. Patent No. 7,209,973 ("Tormasov-973")
1116	U.S. Patent Application Publication No. 20020138629 ("Schmidt-629")
1117	U.S. Patent No. 6,944,860 ("Schmidt-860")
1118	U.S. Patent Application Publication No. 20020174265 ("Schmidt-265")
1119	U.S. Patent Application Publication No. 20020116659 ("Tormasov-659")
1120	Bach, Maurice. 1987. Design of the Unix Operating System by Marice J Bach, 1 edition (Feb. 27, 1987) Prentice Hall; ISBN: 0132017997.
1121	Crowley, Charles. 1997. Operating Systems: a design-oriented approach. Irwin. 1997. ISBN 0-256-15151-2.
1122	Eckel, George and Chris Hare. 1995. Building a Linux Internet Server, Chris Hare", G. Eckel, New Riders Publishing; ISBN: 1562055259.
1127	U.S. Patent No. 6,854,009
1128	U.S. Patent No. 7,080,356
1129	U.S. Patent No. 5,870,539
1130	U.S. Patent No. 6,883,028
1131	U.S. Patent No. 7,209,959
1132	U.S. Patent Application Publication No. 20020065835

Exhibit	Description
1133	The Wayback Machine archive of USENIX Association – Home Page (2001-11-14)
1134	The Wayback Machine archive of USENIX – Publications – Proceedings (2001-11-15)
1135	The Wayback Machine archive of USENIX – 2001 USENIX Annual Technical Conference (2001-11-11)
1136	The Wayback Machine archive of USENIX – 2001 USENIX Annual Technical Conference (2001-11-11)
1137	The Wayback Machine archive of 2001 FREENIX Track Technical Program – Abstract (2002-01-12)
1138	The Wayback Machine archive of "Sandboxing Applications" (2003-03-29)
1139	The Wayback Machine archive of "Sandboxing Applications" (2004-01-19)
1140	The Wayback Machine archive of PDF of "Sandboxing Applications" (2004-01-19)
1141	The Wayback Machine archive of NLUUG: UNIX User Group – The Netherlands (2001-07-22)
1142	The Wayback Machine archive of NLUUG: Previous Events (2001-08-03)
1143	The Wayback Machine archive of Second International SANE Conference (2001-08-12)
1144	The Wayback Machine archive of Second International SANE Conference (2001-06-20)
1145	The Wayback Machine archive of "Jails: Confining the omnipotent root." (2000-09-02)
1146	The Wayback Machine archive of Linux Journal (1999-05-08)

I, Samrat Bhattacharjee, Ph.D., declare:

I have been retained by Wolf, Greenfield & Sacks, P.C., counsel for Petitioner Google LLC, to assess claims 1-34 (the "challenged claims") of U.S. Patent No. 7,519,814 ("the '814 patent") (EX1001). I am being compensated for my time at my standard rate of $700.00 per hour, plus actual expenses. My compensation is not dependent in any way upon the outcome of the inter partes review of the '814 patent.

I. PERSONAL AND PROFESSIONAL BACKGROUND

[0001] My qualifications for forming the opinions set forth in this declaration are summarized below and explained in more detail in my current curriculum vitae, provided as EX1004. EX1004 also includes a list of my publications.

[0002] I am a tenured professor in the Computer Science Department at the University of Maryland. I am also an Affiliate Professor with the Department of Electrical and Computer Engineering at the University of Maryland. I have been an appointed faculty member at the University of Maryland since 1999.

[0003] I received a Ph.D. in computer science from the Georgia Institute of Technology in 1999. Before that I received a B.S. degree in Mathematics and Computer Science with Highest Distinction (Summa Cum Laude) and was recognized as the Outstanding Department Major at Georgia College and State University, Milledgeville, Georgia in 1994.

[0004] I have been involved in the research and development of computing systems for over two decades. I perform research and I teach in the areas of computer software and the management and deployment of server applications, such as computer networks, distributed systems, computer security, cryptography, and operating systems, among others. A major theme of my research has been the development of technology to improve the performance and security of data transfers and data storage on networks that deploy and manage servers and server applications, like the Internet. My research has examined problems ranging from network and operating system support audio and video streaming, large scale Internet video distribution, the design of novel network architectures for better and secure content distribution and storage, efficiency of content distribution over wireless networks, and so on.

[0005] For example, starting in the late 1990s, the focus of my research was the development of network and operating system technology to design a new network architecture that allowed for more efficient data transfer. Part of my Ph.D. research was to develop new and more efficient architectures for video delivery on the Internet, and I have published papers on this architecture during my graduate studies. I continued to work on the management and deployment of server applications for video delivery and security and storage of digital content for server applications as a faculty member, and have published various papers in

these areas. During 2007, I was a visiting researcher at AT&T Labs, and one of the projects I focused on was a video content delivery platform using server applications. This work resulted in both publications and a granted US patent (US 8,752,100 B2). Among other research, I have authored papers and developed research software for public-key infrastructure KeyChains and a cryptographic library for "chit"-based security, which all involve the management and deployment of server applications.

[0006] Along with working on research problems related computer software and the management and deployment of server applications, I routinely taught this material in both graduate and undergraduate courses. A number of my papers in these domains are with students at the University of Maryland, many of whom started in these areas after taking my classes.

[0007] Before the earliest alleged priority date of the '814 patent I had had extensive experience working with various Operating Systems, including at the kernel level. My experience included kernel modifications to both Linux and various flavors of SunOS (including Solaris). For example, I developed an OS for networking applications, and also performed research on various performance improvements for networking applications within the kernel. I also taught the undergraduate OS course at Maryland, and started a reading group to introduce students, including undergraduates to OS research and code for the management

and deployment of server applications. Inspired partly by this experience, a graduate student developed a teaching OS from scratch, which I and others than modified to teach undergraduates about OSs, including the management and deployment of server applications.

[0008] As a result of that experience, I also co-authored a paper on how to teach low-level systems/OS to undergraduates. Evolutions of this teaching OS is still used to teach undergraduate OS at Maryland.

[0009] For over 25 years, I have developed professional and academic experience in the field of computer software and the management and deployment of server applications. For example, I have authored or co-authored over 100 articles in peer-reviewed journals, conference proceedings, and workshops relating computer software and the management and deployment of server applications. Over the years I have received several fellowships, prizes, and awards for teaching excellence and technical papers, including an Alfred P. Sloan Jr. Fellowship in 2004 and the National Science Foundation CAREER award in 2001. I have also received an award from the SIGCOMM foundation for a "Test of Time" paper (that is given to an influential paper published ten years ago). My research work has been supported by multiple grants from the US National Science Foundation, and the Department of Defense. I have also started a Joint Ph.D. program with the University of Maryland and the Max Planck Society in Germany, and co-founded

the annual Cornell, Maryland, Max Planck Research School that provides research exposure to about 80 students from across the world during a week-long school.

[0010] I have served on numerous paper and proposal review panels and participated on program committees for ACM/IEEE groups, technical journals, and conferences in the field of computer software and the management and deployment of server applications. I have created research computer software for all of my projects, much of which is available online for others to build on and involves the management and deployment of server applications.

[0011] I am a named inventor on four U.S. Patents. These patents are generally related to computer software and the management and deployment of server applications.

[0012] I have served as an expert witness and technical consultant in litigation and Inter Partes Review matters concerning computer software and the management and deployment of server applications, including videoconferencing, data conferencing and screen sharing, voice over IP (VoIP) telephony, multimedia networking, distributed systems, operating systems, computer networks, and datacenter networking, among others. I have testified in several trials and depositions as an expert witness.

II. MATERIALS REVIEWED AND CONSIDERED

[0013] My findings, as explained below, are based on my years of education, research, experience, and background in the field of computer software and the management and deployment of server applications, as well as my investigation and study of relevant materials for this declaration. When developing the opinions set forth in this declaration, I assumed the perspective of a person having ordinary skill in the art, as set forth in Section V.B below. In forming my opinions, I have studied and considered the materials identified in the table in the Appendix Listing of Exhibits listed above.

A. Public Accessibility of Non-Patent Prior Art Exhibits

[0014] My opinions regarding public accessibility of the following exhibits are discussed in my Appendix Non-Patent Literature attached hereto in the sections of that Appendix summarized in the table below:

Exhibit	Appendix Section
EX1076	§II.A
EX1077	§II.B
EX1088	§II.C
EX1120	§II.D
EX1121	§II.E
EX1122	§II.F

III. MY UNDERSTANDING OF PATENT LAW

[0015] In developing my opinions, I discussed various relevant legal principles with Petitioner's attorneys. I understood these principles when they

were explained to me and have relied upon such legal principles, as explained to me, in the course of forming the opinions set forth in this declaration. My understanding in this respect is as follows:

[0016] I understand that "inter partes review" (IPR) is a proceeding before the United States Patent & Trademark Office for evaluating the patentability of an issued patent's claims based on prior-art patents and printed publications.

[0017] I understand that, in this proceeding, Petitioner has the burden of proving that the challenged claims of the '814 patent are unpatentable by a preponderance of the evidence. I understand that "preponderance of the evidence" means that a fact or conclusion is more likely true than not true.

[0018] I understand that, in IPR proceedings, claim terms in a patent are given their ordinary and customary meaning as understood by a person of ordinary skill in the art ("POSA") in the context of the entire patent and the prosecution history pertaining to the patent. If the specification or prosecution history provides a special definition for a claim term that differs from the meaning the term would otherwise possess, that special definition controls. If a claim element is expressed as a "means" for performing a specified function, I understand that it covers the corresponding structure described in the specification and equivalents of the described structure. I have applied these standards in preparing the opinions in this declaration.

[0019] I understand that determining whether a particular patent or printed publication constitutes prior art to a challenged patent claim can require determining the effective filing date (also known as the priority date) to which the challenged claim is entitled. I understand that for a patent claim to be entitled to the benefit of the filing date of an earlier application to which the patent claims priority, the earlier application must have described the claimed invention in sufficient detail to convey with reasonable clarity to the POSA that the inventor had possession of the claimed invention as of the earlier application's filing date. I understand that a disclosure that merely renders the claimed invention obvious is not sufficient written description for the claim to be entitled to the benefit of the filing date of the application containing that disclosure.

[0020] I understand that for an invention claimed in a patent to be patentable, it must be, among other things, new (novel—i.e., not anticipated) and not obvious from the prior art. My understanding of these two legal standards is set forth below.

A. Anticipation

[0021] I understand that, for a patent claim to be "anticipated" by the prior art (and therefore not novel), each and every limitation of the claim must be found, expressly or inherently, in a single prior-art reference. I understand that a claim limitation is disclosed for the purpose of anticipation if a POSA would have

understood the reference to disclose the limitation based on inferences that a POSA would reasonably be expected to draw from the explicit teachings in the reference when read in light of the POSA's knowledge and experience.

[0022] I understand that a claim limitation is inherent in a prior art reference if that limitation is necessarily present when practicing the teachings of the reference, regardless of whether a person of ordinary skill recognized the presence of that limitation in the prior art.

B. Obviousness

[0023] I understand that a patent claim may be unpatentable if it would have been obvious in view of a single prior-art reference or a combination of prior-art references.

[0024] I understand that a patent claim is obvious if the differences between the subject matter of the claim and the prior art are such that the subject matter as a whole would have been obvious to a person of ordinary skill in the relevant field at the time the invention was made. Specifically, I understand that the obviousness question involves a consideration of:

- the scope and content of the prior art;
- the differences between the prior art and the claims at issue;
- the knowledge of a person of ordinary skill in the pertinent art; and

- if present, objective factors indicative of non-obviousness, sometimes referred to as "secondary considerations." To my knowledge, the Patent Owner has not asserted any such secondary considerations with respect to the '814 patent.

[0025] I understand that in order for a claimed invention to be considered obvious, a POSA must have had a reason for combining teachings from multiple prior-art references (or for altering a single prior-art reference, in the case of obviousness in view of a single reference) in the fashion proposed.

[0026] I further understand that in determining whether a prior-art reference would have been combined with other prior art or with other information within the knowledge of a POSA, the following are examples of approaches and rationales that may be considered:

- combining prior-art elements according to known methods to yield predictable results;
- simple substitution of one known element for another to obtain predictable results;
- use of a known technique to improve similar devices in the same way;
- applying a known technique to a known device ready for improvement to yield predictable results;

- applying a technique or approach that would have been "obvious to try," *i.e.*, choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success.
- known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art;
- some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior-art reference or to combine prior-art reference teachings to arrive at the claimed invention. I understand that this teaching, suggestion or motivation may come from a prior-art reference or from the knowledge or common sense of one of ordinary skill in the art.

[0027] I understand that a universal motivation known in a particular field to improve technology can provide a motivation to combine prior art references even absent any hint or suggestion in the references themselves. I also understand that obviousness is determined in light of all the facts and that a given course of action often has simultaneous advantages and disadvantages, and that this does not necessarily obvious motivation to combine teachings from multiple references.

[0028] I understand that for a single reference or a combination of references to render the claimed invention obvious, a POSA must have been able to arrive at the claimed invention by modifying, implementing, or combining the teachings of the applied references.

IV. UNPATENTABILITY GROUNDS

[0029] The table below identifies the references, applicable claims, and basis for each ground of unpatentability.

Ground Number and Reference(s)		Claims	Basis
1	Blaser (EX005); Calder (EX1006)	1-4, 7-11, 14, 16-30	Obviousness
2	Blaser (EX1005); Calder (EX1006); Schmidt-449 (EX1007)	5-6, 12-13, 15, 31-34	Obviousness

[0030] The '814 patent's earliest-alleged priority date on the patent itself is 2003-09-15. *See* EX1001, code (60). I understand that Blaser (filed 2003-06-11), Schmidt-449 (filed 2001-03-22), and Calder (published 2002-05-30) qualify as prior art to the '814 patent.[1]

[1] I understand that VirtaMove has alleged that the alleged invention of the '814 patent was conceived sometime in September 2002. For purposes of this declaration, I have been asked to assume that the earliest prior date for the '814 patent is the earliest priority date alleged on the face of the patent, which is

[0031] None of these references were of record during prosecution of the ’814 patent. *See* EX1002; EX1001, code (56).

V. THE ’814 PATENT

A. Background and Specification

[0032] Computer Operating Systems (“OSs”) typically support various system files (like configuration files and libraries) used by the applications installed to a computer. Configuration files provide information about, for example, the computer’s IP address or user-defined settings (e.g., the available and/or selected font(s), font size(s)). Libraries provide pre-existing code for common functions (e.g., opening files) that any application can use. *See, e.g.*, EX1086, 2:62-3:11 (“System calls look like ordinary function calls in C programs, and libraries map these function calls to the primitives needed to enter the

---

September 15, 2003 – the filing date of provisional application No. 60/502,619. *See* EX1001, code (60). However, as I discuss below (§V.E), it is my opinion that the September 15, 2003 provisional application does not provide written description support for either independent claim of the ’814 patent. Nonetheless, the prior art that forms the Grounds I discuss below (Blaser, Calder, and Schmidt-449) are prior art to the ’814 patent even if any claim were entitled to claim priority to the September 15, 2003 provisional application.

operating system in a well-known manner"; referring to conventional Unix-based "system calls, such as open (to open a file for reading or writing), close, read, write").

[0033] Typically, multiple applications are installed on each computer. However, POSAs knew that problems can sometimes arise when multiple applications are installed on the same computer. For example, the '814 patent says an application may replace a shared library with a new version of the library but another application may have been designed in a way such that the application requires (or works better with) the previous, replaced version of the shared library. Another known problem with having multiple applications installed and executing on one computer is that one application may over-consume shared computing "resources," like the processor or "network port numbers," and prevent other applications from executing properly. EX1001, 1:31-36.

[0034] The patent says one known solution to these problems is to "separate[]" applications by installing sets of applications on "individual computer system[s]" (EX1001, 1:27-30), but that can quickly become costly by having to purchase, install, and manage individual computers for each set of applications. An alternative prior-art solution is "Virtual Machine [VM] technology," where one computer can run multiple VMs, each of which simulates a different machine with its own hardware, OS, etc. EX1001, 1:51-61. The patent alleges, however, that

VMs "imposes significant performance overhead," like storage for multiples copies of entire OSs on one machine. EX1001, 1:62-65. The patent says another known technique separated "application[s] from the underlying" OS but allegedly did "not isolate applications into distinct environments." EX1001, 2:4-12.

[0035] As illustrated in the '814 patent's Figure 2 (which I have color-annotated below), the '814 patent's alleged invention is directed to a software "container" having copies of system files for applications, so those applications can be used in different operating environments.

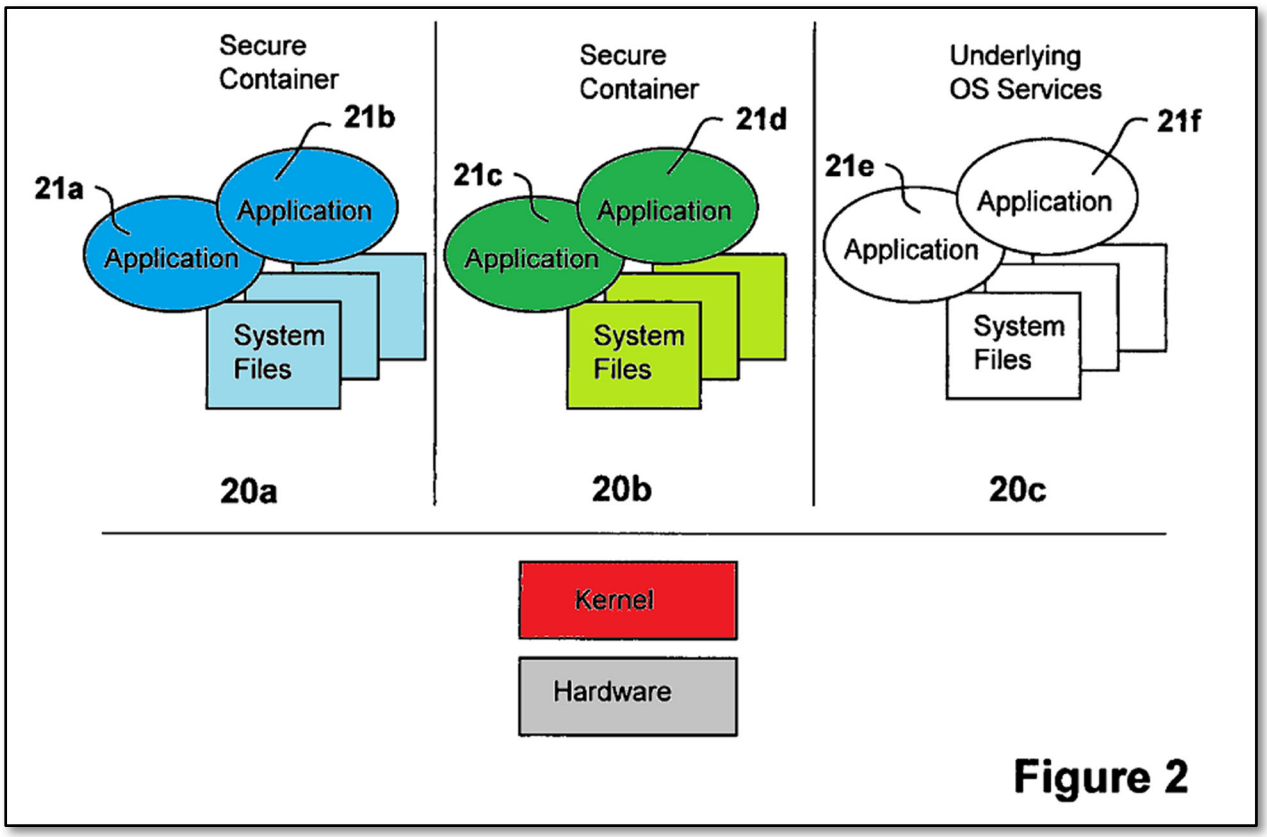


Figure 2

As shown above, "a single server" can execute sets of "[a]pplications" (blue and green) that "are segregated" into respective "secure containers" 20a-20b. EX1001,

7:4-15. Each container has respective system files (light blue and light green) that cannot be overwritten by another container. EX1001, 7:10-12, 9:53-57. Unlike VMs, containers do not comprise an entire OS within each container or simulate having different machines for each container. Rather, containers share a computer's hardware (grey) ***and*** OS kernel (red). EX1001, FIG. 2, 1:56-61. (The kernel is core of an OS that controls access to the underlying computer's resources (e.g., hardware).) Each container is allegedly unlike prior-art Application Virtualization techniques because each container can have a unique identity (e.g., unique IP address). EX1001, 2:10-12, 3:55-57.

B. Person of Ordinary Skill in the Art ("POSA")

[0036] I have been informed and understand that for purposes of assessing whether prior-art references disclose every element of a patent claim (thus "anticipating" the claim) and/or would have rendered the claim obvious, the patent and the prior-art references must be assessed from the perspective of a person having ordinary skill in the art ("POSA") to which the patent is related, based on the understanding of that person at the time of the patent claim's priority date. I have been informed and understand that a POSA is presumed to be aware of all pertinent prior art and the conventional wisdom in the art, and is a person of ordinary creativity. I have applied this standard throughout my declaration.

[0037] A POSA at the time of the '814 patent's earliest claimed priority date (i.e., in the September-October 2003 timeframe) would have had at least a bachelor's degree in computer science, computer engineering, or a related field, with three years of academic and/or industry experience in the area of "management and deployment of server applications." EX1001, 1:15-16. More education may substitute for less experience.

[0038] I am submitting a declaration in IPR2025-00488 that demonstrates that the challenged claims of the '814 patent would have also been obvious even if the '814 patent were entitled to claim priority to a date earlier than the earliest-alleged priority date on the face of the '814 patent (i.e., September 15, 2003) that is also demonstrated to be earlier than the prior art dates for the Blaser or Calder references.

C. Prosecution History

[0039] The original application's claims were rejected as anticipated by U.S. Patent 6,381,742 ("Forbes"). EX1002, 238-271. The applicants responded by adding Limitations [1E]-[1G] and the word "remain" to Limitation [1D]. EX1002, 287-288, 291; *see infra* §VIII (Claim Listing). The applicants also argued, among other arguments, that an example of different OSs occurs when one server has "one container…running an application…compatible with Solaris 8" (a Unix-based OS) and "another container" "running an application…compatible

with Solaris 10." EX1002, 300-301. Solaris 8 and 10 are different versions of the Solaris OS (i.e., they're in the same OS-family, similar to Microsoft Windows 95, 98, XP, etc.). Thus, the inventors' discussion during prosecution establishes that in the context of the '814 patent, different OS, includes OSs that differ by being different versions of the same OS-family, as opposed to different OS only being form different OS-families (e.g., a Windows-based vs. a Unix-based OSs). The Examiner subsequently allowed the claims because the prior art allegedly failed to disclose containers of application software that had unique file systems and that allowed different versions of an OS to execute on the "same system/server environment." EX1002, 341-344.

D. Challenged Claims

[0040] Claims 1-34 are reproduced in the Claim Listing (*infra* §VIII), with labeled limitations (e.g., [**1PREA**]-[**1G**]).

[0041] The '814 patent's two independent claims (1 and 31) each recite a system having secure containers of application programs comprising associated system files for executing a container's applications. The containers do not include an OS kernel, but instead all containers on server's use that server's underlying OS kernel.

[0042] Each dependent claims recites further functionality that was routine and conventional before the '814 patent's earliest claimed priority date—

like using execution files to start applications and pre-allocating computer resources to applications.

E. The Independent Claims Are Not Supported By Provisional Application No. 60/502,619

[0043] I have been asked to consider whether Provisional Application No. 60/502,619 (EX1107; the "'619 application") provides written description support for the independent claims of the '814 patent. (I have not been asked to consider whether '619 application provides written description support for any other claim or to consider whether Provisional Application No. 60/512,103 provides written description support for any claim of the '814 patent.)

[0044] In my opinion, the '619 application does **not** provide written description support for the independent claims of the '814 patent. The '619 application is directed to "Drag & Drop Application Management." EX1107, 1. The '619 application discusses an alleged invention that "relates to software that manages the operation of a data center." EX1107, 1. As the '619 application itself says, the alleged invention discussed in that application is about "installing a software application on a computing platform" using a "software application icon." EX1107, 1. That is, the '619 application is about methods and systems for installing and de-installing applications. However, the '619 application is not about containers of application software. Notably, the '619 application does not use the term "*container*" at all and does not otherwise describe or illustrate

anything that meets the parties' agreed upon construction of a *container* (*infra* §VI.A.2). For at least this reason, it is my opinion that the '619 application does not provide written description support for any independent claim of the '814 patent.

VI. CLAIM INTERPRETATION

[0045] I understand that determining whether a claimed invention is novel and/or non-obvious requires comparing the prior art to the claim. I understand that in some scenarios, an express construction defining the outer bounds of one or more claim terms may be appropriate during an IPR proceeding—for example, where the prior art would meet a challenged claim under one proposed construction but not another construction.

[0046] I have been informed that the parties have agreed to constructions of certain terms and dispute the construction of other terms. Below is my understand of the parties' current positions on claim constructions:

A. Terms with Agreed Upon Constructions

1. Preamble of Claim 1

[0047] The parties agree that the preamble of claim 1 is limiting. EX1106, 6.

2. "*container*" (claims 1, 2, 4, 6, 9, 10, 13, 14)

[0048] The parties agree that the construction of "*container*" should be:

> An aggregate of files required to successfully execute a set of software applications on a computing platform. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.

EX1106, 6.

3. "*a kernel a set of associated local system files* [sic]" (claim 1)

[0049] The parties agree that an "*and*" should be inserted between *"a kernel*" and "*a set of associated local system files.*" EX1106, 6.

4. "*processor*" (claim 1)

[0050] The parties agree that the construction of "*processor*" should be: "physical computer processor." EX1106, 6.

5. "*servers*" (claim 1)

[0051] The parties agree that the construction of "*servers*" should be: "physical servers." EX1106, 6.

[0052] I note that VirtaMove further argues that the scope of "*servers*" should include "a physical server running virtual machine software." *See* EX1105, 4.

6. "*within a container*" (claim 1)

[0053] The parties agree that the construction of "*within a container*" should be: "associated with a container." EX1106, 6.

7. Terms whose antecedent refer to the same entity

[0054] The parties agree that the following terms and their respective antecedent should refer to the same entity. *See* EX 1106, 6-7.

- "*a processor*" / "*the processor*" (claim 1)
- "*a plurality of servers*" / "*the servers*" (claim 1)
- "*one of the servers*" / "*the server*" (claim 1)
- "*operating systems*" / "*the operating systems*" (claims 1, 10)
- "*a plurality of different operating systems*" / "*the plurality of different operating systems*" (claim 1)
- "*one or more of the executable applications*" / "*the one or more of the executable applications*" (claim 1)
- "*a plurality of secure containers of application software*" / "*the containers of application software"* / "*the plurality of secure containers of application software*" / "*the containers*" (claims 1, 6, 9)

B. Terms with Disputed Constructions

1. "*operating system*" (claims 1, 10)

Google's Construction	VirtaMove's Construction
"The software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices." (EX1102, 4)	"No construction necessary; plain and ordinary meaning." (EX1103, 7) VirtaMove further argues that the parties do not "dispute [] the scope of" this term. EX1105, 4.

2. "*kernel*" / "*operating system kernel*" (claim 1)

Google's Construction	VirtaMove's Construction
"The core of an operating system—the portion of the system that manages memory, files, and peripheral devices; maintains the time and date; launches applications; and allocates system resources." (EX1102, 4)	"No construction necessary; plain and ordinary meaning." (EX1103, 7) VirtaMove further argues that the parties do not "dispute [] the scope of" this term. EX1105, 4.

3. "*disparate computing environments*" (claim 1)

Google's Construction	VirtaMove's Construction
Indefinite	"Environments run by standalone or unrelated computers" (EX1103, 8)

[0055] The '814 patent provides a definition of the term "[d]isparate computing environments" at EX1001, 2:17-19: "[e]nvironments where computers are stand-alone or where there are plural computers and where they are unrelated." My understanding of Google's basis for arguing the claimed "*disparate computing environments*" is indefinite is that "the specification provides no objective boundaries of 'relatedness'" for the computers running the environments. EX1102, 6-7. However, my understanding is that Google has not argued that it would be unclear to POSAs what it means to have "stand-alone" computers.

[0056] As I discuss further below (§VII.A.4.a.i.(2)), it would have been obvious to implement the Blaser-Calder combination (*infra* §VII.A.3) to

execute its environments on multiple, standalone computers and thus meet Limitation [**1PREA**]. In other words, even though the full scope of *disparate computing environments* is unclear because the claim language read in light of the specification does not provide a sufficiently clear boundary, the Blaser-Calder combination would have been well-within that boundary.

[0057] Thus, I understand that the Board can find that the Blaser-Calder combination render claim 1 unpatentable without having to resolve whether the term *disparate computing environments* renders the claim indefinite.

4. "*service*" (claims 1, 14)

Google's Construction	**VirtaMove's Construction**
"specialized, software-based functionality provided by network servers and comprised of one or more applications" (EX1102, 7)	"No construction necessary; plain and ordinary meaning." (EX1103, 9)

5. "*at least some of the different operating systems*" (claim 1)

"*at least some of the plurality of different operating systems*" (claim 1)

Google's Construction	**VirtaMove's Construction**
"at least two or more of the different operating systems" / "at least two or more of the plurality of different operating systems" (EX1102, 11)	"No construction necessary; plain and ordinary meaning." (EX1103, 12)

6. "*memory accessible to at least some of the servers*" (claim 1)

Google's Construction	VirtaMove's Construction
"memory that at least two or more of the servers can read from or write to" (EX1102, 11)	"memory that at least some of the servers can read from or write to" (EX1103, 12)

7. "*local kernel residing permanently on one of the servers*" (claim 1)

Google's Construction	VirtaMove's Construction
"local kernel in one of the server's memory that is not lost when power is removed from it" (EX1106, 4; EX1102, 12)	"No construction necessary; plain and ordinary meaning." (EX1106, 4; EX1103, 13)

8. "*secure containers of application software*" (claim 1)

Google's Construction	VirtaMove's Construction
"environments where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets" (EX1102, 13)	"[c]ontainers where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets" (EX1103, 14)

9. "*an operating system's root file system*" (claim 1)

Google's Construction	VirtaMove's Construction
Indefinite (EX1102, 14)	"No construction necessary; plain and ordinary meaning." (EX1103, 15) Further arguing that "[t]he point of the claim is that each claimed container will have its own root file system rather

	than sharing a root file system with *any* operating system's root file system. If a container shared its root file system with even one operating system, then it would not be unique." EX1105, 8 (emphasis original)

[0058] This term is part of limitation [**1G**], which states "*wherein each of the containers has a unique root file system that is different from* ***an operating system's root file system***." In the related district court litigation, Google argues this term is indefinite because "the claim language does not indicate to *which* 'operating system' th[e] comparison should be made" to determine whether "each of the containers [] unique root file system" is "*different from an operating system's root file system*." EX1102, 14.

[0059] As I discuss further below (§VII.A.4.a.ix, [**1G**]), Blaser-Calder's application layers (*containers*) each have a unique root file system that is different from each other and different from the root file system of every server in the world. Thus, Blaser-Calder would have met limitation [**1G**] as to **any** OS's root file system in the Blaser-Calder world. In other words, even though the full scope of limitation [**1G**] is unclear because the claim language read in light of the specification does not provide a sufficiently clear boundary, the Blaser-Calder combinations would have been well-within that boundary.

[0060] Thus, I understand that the Board can find that the Blaser-Calder combination renders claim 1 unpatentable without having to resolve whether the term *an operating system's root file system* renders the claim indefinite.

* * *

[0061] As I discuss further below (§VII), the Blaser-Calder combination renders obvious challenged claims 1-4, 7-11, 14, and 16-30 under both parties' claim constructions and the Blaser-Calder-Schmidt combination renders obvious challenged claims 5-6, 12-13, 15, 31-34 under both parties' claim constructions. For all other claim terms, I have applied the term's plain and ordinary meaning as it would have been understood by POSAs at the time of the earliest alleged priority date (i.e., September-October 2003 timeframe), which would have been the same in the September-October 2002 timeframe.

VII. APPLICATION OF PRIOR ART TO CHALLENGED CLAIMS

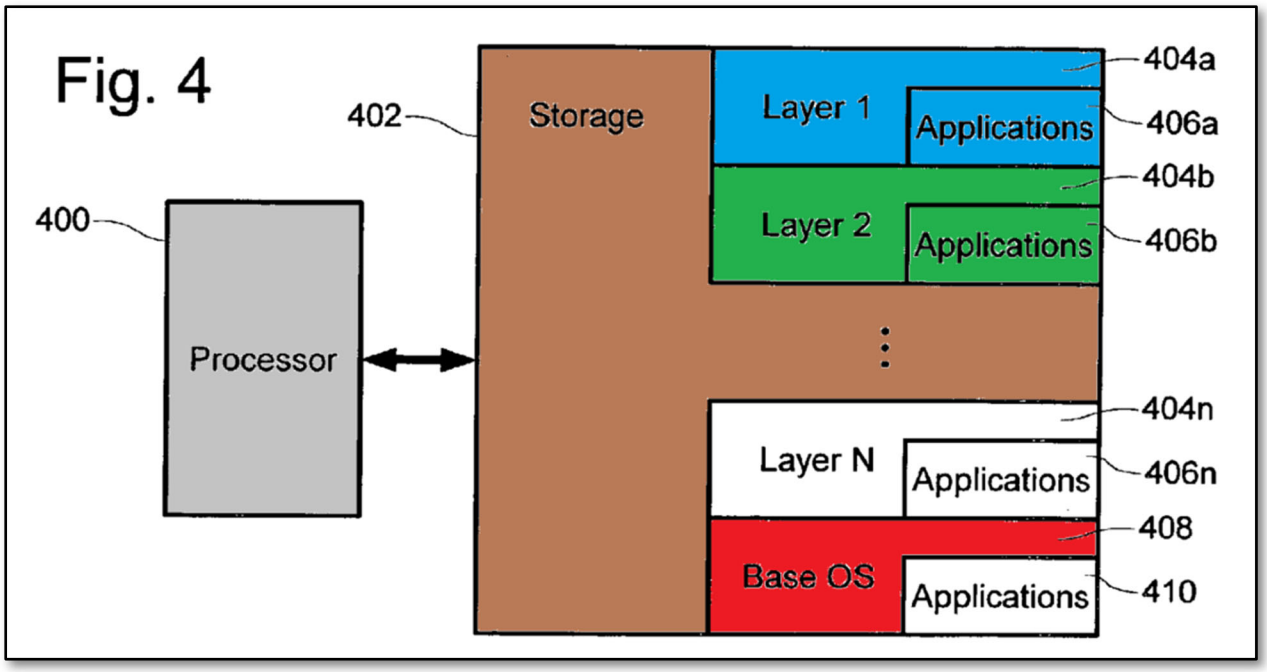
A. GROUND 1: Claims 1-4, 7-11, 14, and 16-30 would have been obvious to POSAs over the Blaser-Calder combination.

1. Blaser (EX1005)

[0062] Blaser is an issued U.S. Patent that was filed on June 6, 2003 and issued on October 3, 2006. Blaser further claims priority to a provisional application filed on June 12, 2002. *See* Blaser, code (60). Blaser generally

"relate[s] to methods of capturing software packages using layered computing systems and software package products produced by those methods." Blaser, Abstract. As I discuss further below, the computing devices on which Blaser installs application layers include server devices. Blaser, 3:16-27. Thus, Blaser is in the same field of endeavor as the '814 patent. *See* EX1001, 1:12-16 ("FIELD OF THE INVENTION: The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.").

[0063] Like the '814 patent (*supra* §V.A), Blaser recognizes that "computing system have been susceptible to application conflicts" due to conflicting "shared libraries and system configuration" (e.g., conflicts when applications require "different versions" of a single shared library). Blaser, 1:14-31. Blaser proposes a software-based solution to that potential problem in installing multiple applications to one computer. Blaser's solution is to install "an application layer" that is "a group of files in combination with any associated application configuration stored to operating system files" (Blaser, 3:44-46), an example of which is illustrated in Figure 4 (which I annotated below) (Blaser, 4:51-59).



[0064] The computing device (e.g., a "server[]") (Blaser, 3:16-27) has "processor 400" (gray) and "storage devices 402" (brown) (Blaser, 4:51-59), which store "base operating system 408" (red) and several "layers 404a-n" – e.g., Layer 1 (blue) and Layer 2 (green) – each "having [respective] applications 406a-n." Each layer also comprises its own set of "libraries" and configuration files. Blaser, 3:42-58, 4:7-27, 5:18-32, 6:62-7:23. Blaser's layers of "application files and directories are shadowed or overlaid over the regular operating system file system." Blaser, 5:20-26. Thus, the layers do not include an OS, but instead share and use the same Base OS (red) (and its corresponding kernel), which "forms a platform with which applications can be run." Blaser, 4:7-26.

[0065] My opinion that Blaser's OS comprises a kernel is supported by EX1046, which discloses (as POSAs knew) that the "kernel is the core of the

operating system." EX1046, 1:25-26. I note too that VirtaMove's infringement contentions argue that an OS "by definition, includes a kernel." EX1103, 12. VirtaMove's argument is consistent with my opinion that POASs understood that the conventional OSs that Blaser used would have included a kernel.

[0066] Each of Blaser's layers can be "managed as a unit for the purposes of exporting, importing." Blaser, 3:61-4:2. Thus, "vendors can provide 'pre-installed' applications as layers on CD-ROM or other media" and applications can be transferred "between similar computing systems" "by moving the layer containing the application." Blaser, 7:12-21. In other words, vendors can pre-package applications into layers and store them to a CD-ROM from which the layer and applications within layers can be installed to multiple computers/servers.

[0067] Blaser says that by "isolating application files and configuration in a layer," Blaser's technique "provid[es] a barrier between applications which may use conflicting configuration or library files." Blaser, 3:51-56. Thus, "[a] layer [] provides a convenient container to limit access to an application." Blaser, 7:21-23.

2. Calder (EX1006)

[0068] Calder is a U.S. Patent Application Publication that was filed on November 29, 2000 and published on May 30, 2002. Calder generally "relates to distributed computing, and more particularly, relates to secure peer-to-peer

Internet or enterprise distributed computing. The invention also relates to the secure execution of an application on a client computer." Calder, [0002]. As I discuss further below, the computing devices on which Calder stores and installs application packages include server devices. Calder, [0007], [0073]-[0078]. Thus, Calder is in the same field of endeavor as the '814 patent. *See* EX1001, 1:14-16 ("FIELD OF THE INVENTION: The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.").

[0069] Calder teaches techniques for modifying "an application package to be executed safely, securely, and transparently on a remote machine." Calder, [0073]. Calder teaches that conventional software "[p]ackages" that are used install applications to computing systems typically include the application's binaries ("also called application program," which typically include, e.g., executable files to start and run the application) and system files, like libraries ("DLL[s]") and configuration files. Calder, [0076], [0082]. As I discuss further below, Calder's invention takes conventional packages for installing applications and outputs modified versions of the files in those conventional packages (e.g., "modified binaries [], modified libraries [], modified configuration files []," Calder, [0082]).

[0070] My opinion that a conventional software package for installing an application typically includes, among other files, executables and libraries is further corroborated by EX1014, which discloses:

> **Typically**, **one software package includes all the files which are normally required for a local setup procedure and which correspond to the application being considered**. Those installation files clearly depends upon the type and the complexity of the particular application for which an installation is required. **Such installation files, including the Dynamic Link Libraries (DLL)** and all the subsequent files which are to be copied on the hard disk drive of PC client 3, for instance, are well known from the skilled man and will not, for that reason, be developed with more details. Typically, it is sufficient to observe that those files include all the files which are normally involved **in a local setup procedure and that the particular executable file—the setup.exe—which causes the launching of the installation procedure**, has to support an unattended mode, which is that which is being executed when the user types the "-s" switch on the command line (unattended or silent setup).
>
> In addition to the installation files required for a standard local setup procedure, the software package located on server 2 further includes an **additional description file, hereinafter referred to as package.ini file**. Package.ini file may take the form of a text file and contains the description of the installation files which are involved in the setup procedure. **It particularly includes the precise list of the installation files required for a local setup procedure**, plus an additional line

carrying the command which is required for starting the local setup procedure.

Considering the example of the Microsoft Office™ software package which is marketed by Microsoft™ Corp., server 2 is arranged to store **the standard Microsoft installation files**. In addition server 2 includes a package.ini description file which defines the list of those files and further comprises an additional line to run the silent setup procedure, i.e. the following line: "setup.exe -s".

EX1014, [0026]-[0028]. *See also* Blaser, 1:14-19 (noting that "[w]hen an application is installed to an OS, a number of globally accessible files are often placed to the computing system, including for example shared libraries and system configuration").

[0071] As Calder notes (and POSAs knew), one common function provided by libraries is to make system calls, which are requests to use a service or functionality provide by the underlying OS's kernel.[2] Some examples of

[2] My opinion that system calls are often made using one of an OS's libraries is corroborated by EX1120, which discloses that in a Unix-based OS, for example, "[s]ystem calls look like ordinary function calls in C programs, and libraries map these function calls to the primitives needed to enter the operating

conventional functions that are implemented via conventional system calls, are functions to open a file in the file system and use the computer's network connection. However, POSAs knew that different OS often have different system calls and/or syntaxes for executing system calls, in particular OS is different OS families.[3] Thus, an application package intended for one OS may not work with another OS.

[0072] Calder's solution to permitting an application made for one OS to execute on a different OSs having a potentially different set of system calls is to modify an "application package" by adding "system libraries…that translate" "system calls" intended for one OS (e.g., "Windows 2000") to work with a computer having a different OS, including an OS in a different OS-family (e.g., "Linux"). Calder, [0101]. For example, as I discuss further below, Calder teaches that "the application package is modified using a preprocessing module which,

---

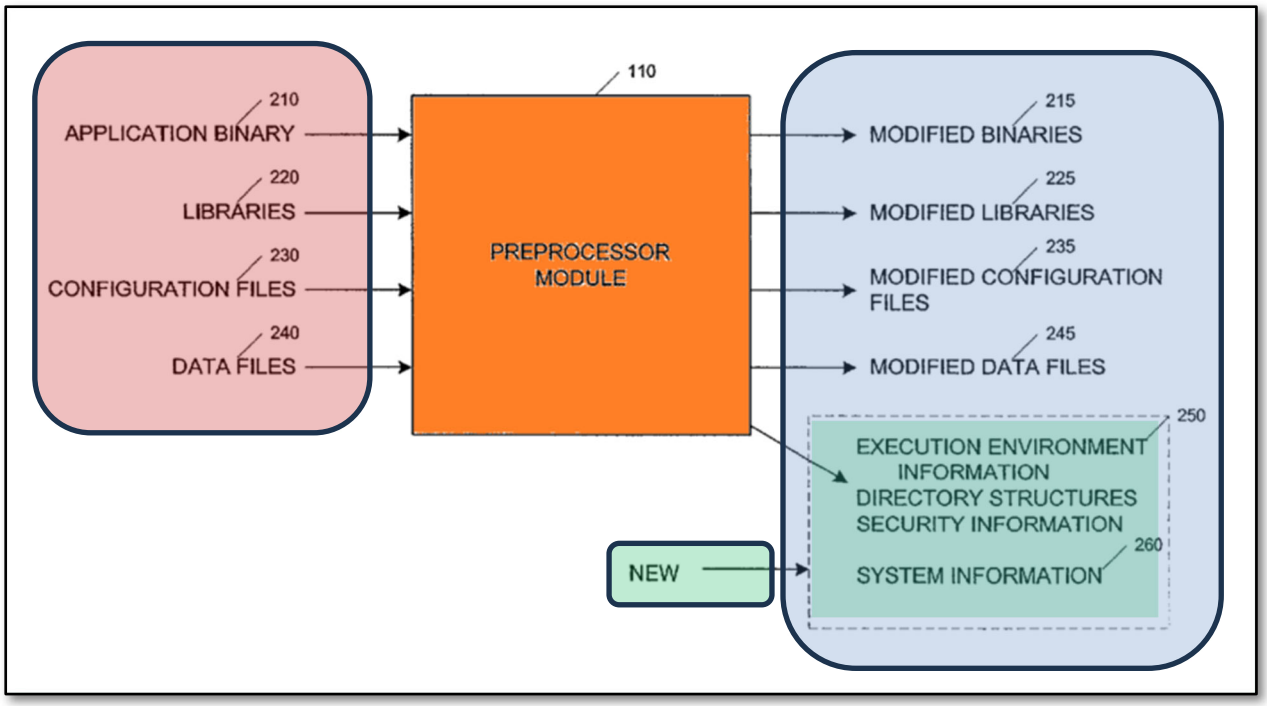
system….Programs frequently use other libraries such as the standard I/O library to provide a more sophisticated use of the system calls." EX1120, 19-20.

[3] My opinion that OS in different families often used different system calls is corroborated by EX1097, which states (as POSAs knew) that "[d]ifferent operating systems provide different mechanisms for implementing system calls." EX1097, [0082].

among other things, modifies the binaries of applications in the application package such that an interception module is loaded when the binary is executed." Calder, [0073].

[0073] As shown in Calder's Figure 2 (which I have colored below), Calder's software-implemented "preprocessing module [orange]" processes an application package's original files (pink) to create modified files (blue) (e.g., modified libraries and configuration files) that "allow for the application 405 to execute and communicate on any non-native platforms" (i.e., to execute on an OS that is different from the application's "native" (originally intended) OS). Calder, [0087]-[0088], [0101].[4]

[4] My opinion that POSAs understood Calder's non-native platform to be a computer executing an OS that is different from the application's native OS is corroborated by EX1054, which states (as POSAs knew) that a "'non-native' server is [] a server…**running an operating system that is <u>different</u> than**" "the client['s]" OS. EX1054, 4:37-41.



[0074] Calder discloses that "[i]n one embodiment, system resources are controlled by using the virtual layer 415 to intercept part or all of the application programming interface (API) routines that utilize these resources." Calder, [0086]. The virtual layer includes "an interception module" that "intercept[s]" "[p]art or all of the system call made by the application." Calder, [0087]. "By intercepting these interfaces, the interception modules can prevent a user of the client computer 140 from accessing the contents of the application package 115 and/or the application from improperly modifying or accessing data from the client computer." Calder, [0088].

3. The Blaser-Calder Combination

[0075] As I discuss below, POSAs had several reasons to install Blaser's layers in a system having multiple servers running different OSs, with each layer being executable on more than one different OS in the system, based on Calder's teachings to modify an application package so that the application(s) in the package could be executed on native **and** any non-native OS.

[0076] As I discussed above (§VII.A.1), Blaser's "layering system" provides for "transfer of applications between similar computing systems." Blaser 3:16-24; Blaser, 8:58-61 ("files of a layer [can] be moved from one Windows system to another"). Blaser's "computing systems" include "servers" (Blaser, 3:20-24) and Blaser says layers can be installed to various "Windows operating system[s]" – "Windows 95, 98, NT, 2000, and XP" (Blaser, 13:43-45) and "operating systems other than Windows" (Blaser, 3:33-36).

[0077] As I discussed above (§V.C), during prosecution, the '814 patent's applicants told the Patent Office that different OSs in the same OS-family (e.g., different versions of OSs with the same tradename, like Unix-based OSs "Solaris 8" and "Solaris 10") are OSs that differ. *See* EX1002, 300-301. By that same reasoning, Windows 95 and Windows 98 would be different OSs, different Linux distributions would be different OSs, etc.

[0078] POSAs knew that applications native to one OS (e.g., Windows 95) could often execute on another OS in the same family (e.g., Windows 98). In other words, POSAs knew that it would have been conventional to have applications that were executable on different OSs, at least under the inventors' argument that different versions of OSs in the same OS-family are different OS. My opinion that POSAs knew that applications native to one OS could often execute on another OS in the same family is corroborated by EX1017. EX1017 discloses that it was known that "[p]eople want to run desired applications in their chosen platform in the ideal world" but it can sometimes be "very difficult to run an application in a different host platform that it wasn't written for." EX1017, [0019].

[0079] EX1017 further notes as background information that "[e]ach platform has its corresponding body of native applications that are designed to run under it." EX1017, [0020]. Thus, "[w]hen a new generation of platform is released, software developers generally upgrade their products to run under the new generation platform." EX1017, [0020]. EX1017 provides as additional background information that "OS developers **wish to make their products backwards compatible**" so that "**older generations of applications may run on the latest generation of the OS** (and thus the latest generation of platform)." EX1017, [0021]. In other words, product developers sought to develop

applications and OS so that "non-native applications can run under a native platform (including the new OS)." EX1017, [0021].

[0080] My opinion that POSAs knew that applications native to one OS (e.g., Solaris 8) could often execute on another OS in the same family (e.g., Solaris 10) is further corroborated by EX1128. EX1128 discloses that "it would be desirable to provide a tool for application software developers to assess the compatibility of their software programs with existing versions of operating systems on which the software may execute." EX1128, 1:51-57. EX1128 further discloses "[a]spects of" the reference's invention, in context of "certifying binary compatibility between application software and the Sun Solaris ™ operating system":

> Aspects of the present invention will be described with reference to certifying binary compatibility between application software and the Sun Solaris ™ operating system, or more particularly between application software and the Sun Solaris™ Application Binary Interface (ABI). It will be appreciated, however, that the present invention is not so limited.
>
> **The term binary compatibility refers to maintaining compatibility between a binary (i.e., compiled) version of a first software module, e.g., an application module, and a second software module, e.g., an operating system.** The Solaris™ operating system complies with an ABI, which is a set of supported run-time interfaces available for an application to use. Among other things, the Solaris™ ABI defines the

interfaces provided by the system libraries, the interfaces provided by the kernel system calls, the locations and formats of various system files and directories, and the input and output syntax and semantics of system utilities. **An application that is binary compatible with the ABI of a particular version of an operating system <u>should be compatible with any other version of the operating system that maintains the same ABI</u>.**

The Solaris™ ABI changes <u>infrequently</u>, and only with significant advance notice.

EX1128, 3:7-30. Thus, EX1128 corroborates my opinion that it was conventional in the prior art for applications native to one OS (e.g., one particular version of the Solaris OS) to be compatible with other versions in the same OS family (e.g., another version of the Solaris OS that had a binary compatible ABI, which, as EX1128 recognized, "changes infrequently" between versions).

[0081] My opinion that POSAs knew that applications native to one OS were typically executable on another OS in the same family is further corroborated by EX1050. EX1050 discloses that:

The manner in which UNIX implements System Calls, particularly to storage, is in many respects quite advantageous to system performance. In UNIX, t**he System Call is the interface between UNIX and an application program**. **A System Call by the application program requests the "kernel" portion of the UNIX operating system to perform one particular task or service on behalf of the operating**

> **system**. T**he "kernel" portion of UNIX includes approximately 60 System Calls which are not changed between different hardware systems, and are the standard interface to UNIX**. Other programs in UNIX adopt the kernel to the particular hardware environment.

EX1050, 2:25-38. As EX1050 states, all Unix distributions would have "approximately 60 System Calls which are **not changed** between different hardware systems, and are the **standard interface** to UNIX." Thus, POSAs would reasonably expect that software applications that relied on only standard UNIX system calls would for (at least) that reason be compatible with multiple versions of a Unix-based OS. POSAs would have reasonably expected the same to have been true regarding different versions of OSs within other OS families.

[0082] Thus, EX1017, EX1128, and EX1050 corroborate my opinion that POSAs would have reasonably expected that Blaser's applications would have been compatible with multiple, different OSs (at least) under VirtaMove's argument during prosecution of the '814 patent that OSs that differ include OSs in the same OS-family (e.g., an application originally intended for Windows 95 then, and now, can also run on a computer running Windows 98).

[0083] Blaser also says "multiple versions of a software product are [often] installed" in a network because "an older version…supports a function desired but not supported in a newer version" or because a "software product tester[]" wants to "verify" "multiple development versions." Blaser, 13:29-41.

POSAs understood that testing different "development versions" of software includes (or typically would have included) testing **different** versions of the same application designed for **different** OS, as that was the most common situation where an older function (for an older version of an application native to one OS) would have no longer been supported by a new version of the application (because the newer version was native to a new version of an OS in the same OS family and, for various potential reasons (e.g., it was not widely popular), the old function was not continued forward for the new OS). Thus, Blaser itself motivates (if not itself teaches) operating layers in systems having multiple servers having different OSs, with each layer being executable on at least some of the different OSs in the system.

[0084] As I discuss below, POSAs had further motivation to make Blaser's layers executable on multiple, different OSs in a system in light of Blaser's additional disclosures and POSAs' background knowledge that I discuss below.

[0085] Blaser discusses an exemplary system operating within "a company ha[ving] 500 computers" and having different "layers defined for different types of users." Blaser, 13:1-8. For example, there is a "secretarial layer [that] contains…secretarial applications" (e.g., "word processing") and an "engineering layer [that] contains…engineering related applications" (e.g., "CAD

tools"). Blaser, 13:8-15. POSAs knew that large corporate systems like the one Blaser discusses often had "computers spread across a wide geographical area that are networked together," with groups of users "interconnected" to "one or more server computers" that "**run[] different network operating systems**." EX1049, 1:12-14, 1:41-52. That is another reason why POSAs would have had reason to implement each of Blaser's layers so that they were each compatible with OSs that differ.

[0086] POSAs also had reason to make each of Blaser's application layers compatible with multiple, different OSs (e.g., one layer that would have been compatible with both a server running a Windows-based OS and another server running a Linux-based OS) and deploy those layers in a system having a plurality of servers running different OSs. That is because POSAs knew as part of their background knowledge that "[i]t is **important…that an application be usable with several different operating systems**." EX1058, 1:14-21. Similarly, POSAs also knew as part of their background knowledge that "[s]oftware developers often strive to tailor or 'port' their applications to a variety of computing platforms," but "system-dependent variables, **such as…operating system** [i.e., operating system differences], make porting a difficult task." EX1057, 1:10-14. For example, POSAs knew "it is desirable to have the 32-bit applications operate and function correctly on a 64-bit platform":

> The capability of running 32-bit applications on the 64-bit platform **is highly <u>desirable</u> for easing the transition from the popular 32-bit platform to the new 64-bit platform**. Not only may this ease the transition, it may **reduce the burden on software developers to immediately port their software to the 64-bit platform**. It also **gives the new 64-bit platform an existing library of software** (e.g., the old 32-bit applications). Furthermore**, it is <u>desirable</u> to have 32-bit and 64-bit applications interoperate**.

EX1017, [0029]-[0030].

[0087] Moreover, as EX1049 corroborates, POSAs knew as part of their background knowledge that "in a distributed system of a large corporation having many end users, there are usually a number of computers **with <u>different</u> operating systems**." EX1049, 1:41-58. As EX1051 similarly corroborates, POSAs also knew that "[i]n larger networks, it is common…for the servers to be running **<u>different</u> network operating systems**." EX1051, 2:35-37. And, as EX1020 corroborates, in systems for large networks, POSA knew as part of their background knowledge that it would have been common and beneficial to have "software…centrally hosted by a Server and used by a number of Clients." EX1020, [0004]-[0005].

[0088] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further

corroborated by EX1054. EX1054 states that "[i]n the prior art, a user of a Windows NT client [] is typically authenticated by a Windows NT server." EX1054, 2:48-54; 4:48-50. EX1054 further describes a "server system" that "run[s] any of a number of computer operating systems," including various "[w]ell-known server operating systems like "Novell Netware, IBM OS/2 Warp Server, IBM AS/400®, Microsoft Windows NT, and many variations of OSF UNIX." EX1054, 1:14-35. "The server system is accessed by the client system for specific functions," including "execution of applications, and storage of and access to user information." EX1054, 1:27-35. As I discussed above, in a typical large network in the prior art, there would have been client computers running a variety of different OS. Thus, in light of that background knowledge, POSAs knew it would have been conventional for the client computers in a large network to access servers running corresponding OS. For example, Windows-based clients would often access Windows-NT-based servers to execute Windows-based applications; OS/2-based clients would access OS/2-based server to execute OS/2-based applications; etc.

[0089] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1066. EX1066 states that it was known that "[t]here exist

many **legacy server applications that have provided and continue to provide valuable information processing servers** to organizations worldwide. Many of these legacy server applications, which are the result of many years of development, contain, store or access many years (and sometimes decades) worth of data." EX1066, 1:17-37. Thus, in light of that background knowledge, POSAs would have had reason to make legacy applications executable on different OS, such as more-recent OSs that had additional features, such as improved security, speed, and the ability to support more-recent applications. Thus, for example, POSAs would have wanted to deploy servers that ran state-of-the-art OSs, while having the ability to execute and server legacy applications designed for older OSs.

[0090] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1068. EX1068 states that "[a] current trend in computing is **to interconnect a variety of computer architectures and computer operating systems in a single network** 100." EX1068, 1:19-27. For example, a network that "includes a variety of servers, i.e., a Unix server 150, a Netware server 160, an OS/2 server 170, and a Windows NT server 180," which is "referred to as heterogeneous networks" and can "include local area networks." EX1068, 1:19-27. Thus, EX1068 further corroborates my opinion that it would have been

conventional in the prior art to have networks with a plurality of servers, where the servers executed different OSs.

[0091] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1071. EX1071 notes as background information that "companies having large networked facilities **can internally have thousands of networked computers accessible by numerous content servers** on a single network. **Each of the content servers can be running any of various operating systems** as can the computers with which the servers are communicating." EX1071, 2:4-14. EX1071 further notes as background information that POASs knew that in those large systems "in the case of networked software, **each server has an individual copy of networked software** for a subset of the users." EX1071, 2:4-14. Thus, EX1071 further corroborates my opinion that it would have been conventional to have large networks that deployed a plurality of servers running different OSs, where the servers had copies of networked software for subsets of users. POSAs understood that the subsets of users were typically grouped based on location and/or role in the organization (e.g., engineering department, finance department, word-processing department, etc.).

[0092] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1016. EX1016 discloses as background information:

> Many businesses and other enterprises **use a variety of computer systems that are specially adapted for certain purposes**. For example, **a manufacturing division of an enterprise may use one kind of computer system specifically designed** to handle manufacturing data **while the sales division of the same enterprise may use another kind of system** for sales information. The **engineering division of the enterprise may use an entirely different computer system** as well. Using different computer systems for different divisions of an enterprise makes sense because **each kind of computer system will provide certain strengths that suit that division**.

EX1016, [0002]. Thus, EX1016 further corroborates my opinion that POSAs knew it would have been typical to have large networks (e.g., large corporate networks like the ones Blaser discusses) with a variety of computer systems, specialized for different divisions. POSAs further knew that the systems for different divisions often beneficially used different OSs. For example, a Unix- / Linux-based OS for the engineering division and a Windows-based OS for the secretarial division. At the same time, as I discussed above/below with corroboration, POSAs would have understood that certain applications would have

beneficially been available across all divisions, regardless of the OS being run on the division's server(s) (e.g., e-mail applications, applications unique to the organization, etc.).

[0093] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1053. EX1053 discloses as background information that UNIX was "[o]riginally designed for software development, computer-aided design (CAD), and other **technical applications run by engineers**." EX1053, 1:36-41. Thus, EX1053 further corroborates my opinion discussed above that in large organizations different divisions would have had need for applications (and corresponding application servers) designed for particular OSs (such as Unix-based OSs for an engineering division).

[0094] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1049. EX1049 discloses as background information:

> The client computers within a group of computers are typically used by end users. **End users often have preferences regarding what operating system runs on their computer**. Therefore, in a distributed

system of a large corporation having many end users, **there are usually a number of computers with different operating systems**.

EX1049, 1:53-58. Thus, EX1049 corroborates my opinion that it would have been typical in a large network to have client computers running a variety of different OSs because different users had different preferred OSs (e.g., Windows-based versus Mac-based). As I also discussed above with corroboration, it would have also been typical and conventional to have application servers running OSs matched to the OSs on the client devices (e.g., a Windows-based server to serve Windows-based clients).

[0095] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1012. EX1012 discloses as background information:

> **Modern data processing systems are often implemented in "client-server" configurations in which a number of "client" computers**, typically personal computers or workstations, **communicate with one or more "server" computers and request such server computers to perform processing tasks**.
>
> In conventional client-server configurations, **each client computer is required to be compatible in some manner with each server computer so that the client and server computers can understand one another**. In some environments, this requirement is easily met, as

each client and server computer is of the same platform type, e.g., UNIX, Windows, Macintosh OS. In other environments, **special translation protocols are implemented to permit a particular client platform type to communicate with a particular server platform type**.

As client-server systems become larger and serve entire enterprises, such as large corporations or institutions, conventional techniques for facilitating communication between and among client and server computers become more difficult to use, while **the need for processing that is seamlessly distributed among the systems' constituent client and server computers generally increases**.

For example, if a user of a particular client machine **creates a particular report application program** that provides a printed report of general value throughout the organization, **it would be extremely valuable for other users, perhaps working with clients and servers of different platform types, to be able to use the report application program**.

EX1012, [0002]-[0005]. Thus, EX1012 further corroborates my opinions that it would have been typical and conventional in large organizations such as the ones discussed in Blaser (1) to have a client-server model where the clients and servers ran a variety of different OSs; (2) to have servers running OSs matched to the client OSs and to support serving applications between clients and servers running different OSs; (3) to have applications where it would be beneficial for the same

application program to be available across the organization, regardless of what platform (e.g., OSs) the client devices were running.

[0096] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1093. EX1093 discloses as background information:

> The present invention relates to the monitoring of server performance in a network like the Internet. System administrators, as well as information technology (IT) executives, generally like to know the productivity of their systems, and want to accurately forecast future needs for upgrading or expanding existing systems. Capital expenditures must be budgeted according to changes in server resource requirements, based on key server operating parameters such as CPU utilization, memory availability, I/O usage, and permanent storage (disk) capacity. This objective has become very difficult to achieve, due to extremely **complex environments of hundreds or thousands of networked servers running on many different platforms, including UNIX and its variations (such as AIX, HP-UX, System V, and Mach), Windows NT, OS/2, NetWare, etc. (UNIX is a trademark of UNIX System Laboratories; AIX is a trademark of International Business Machines Corp (IBM); HP-UX is a trademark of Hewlett-Packard Co.; System V is a trademark of American Telephone & Telegraph Co.; Mach is a trademark of NeXT Computers; Windows NT is a trademark of**

Microsoft Corp.; OS/2 is a trademark of IBM; and NetWare is a trademark of Novell Corp.). In addition to these platforms, some servers are managed by specialized systems, such as SAP (a suite of bundled business software including an accounts receivable program, a materials management program, a scheduling program, etc.).

EX1093, 1:66-2:25.

[0097] My opinion that POSAs would have had motivation to make each of Blaser's application layers compatible with multiple, different OSs and deploy them in a system with a plurality of servers running different OSs is further corroborated by EX1127. EX1127 discloses a computer accessing application programs from a remote "server farm" having a plurality of servers running different OSs (including OSs in different OS families):

> Referring again to FIG. 1A, appliance 132 may include a mother board and sound card, a video card, ports to plug in a keyboard and a monitor as well as some other digital equipment, but need not have a hard drive installed in it. Upon startup, appliance 132 automatically calls up the server farm 100 through either a telephone wire 112 or a cable modem, a digital subscriber line (DSL) or a network cable 111 on either a wide area network (WAN) or a local area network (LAN) to one of the remote servers in the server farm 100. The remote server acts as an application host and allows the user of the appliance 132 to choose from a number of operating systems offered by the remote server. **For example, the user can choose to operate in Windows NT, Windows**

> **95, Windows 98, or Linux**. Once the automatic download of the OS is completed, the user is then able to choose from a number of applications. The applications may be the same applications that are commonly locally installed. The applications can be downloaded when they are to be used, without incurring large delays. The downloaded software is stored in random access memory (RAM), at least until the user logs out of the system. User data are stored on one of the severs of the server farm 100. No results have to be stored locally. Because all software is downloaded from the network, and all work is remotely saved in the server farm 100, the appliance 132 need not have a hard drive. The appliance can thus be less expensive than a conventional computer.

EX1127, 4:39-65.

[0098] Thus, as I have demonstrated above with multiple pieces of corroborating evidence, to the extent Blaser does not disclose using layers in a system having multiple servers running different OSs (and it does for the reasons I discussed above), POSAs would have also had motivation based on their background knowledge to implement Blaser's layers in a system having multiple servers running disparate OSs (e.g., a typical, conventional, large corporate network), where each application layer could be executed on each of those different OSs in the system. *See also* EX1109, [0002]-[0003] (disclosing as background information that "an enterprise network" can have "more than one

thousand networked server computers" that "may be remote from one another," such as on different continents).

[0099] As I discussed above (§VII.A.2), Calder's teaches modifying an "application package" (e.g., its libraries and configuration files) so "the application" can "execute and communicate on any non-native platforms," like allowing native Windows applications to work on a server running a Unix-based OS. Calder, [0101]. As discussed above (§VII.A.1), Blaser's layers include similar system files for installing applications, and Blaser's application layers can be used with various Windows- and non-Windows-based OSs (i.e., OSs that overlap and/or are similar to the conventional OSs that Calder says can be used with Calder's technique). Even if a Blaser layer were limited to one OS, POSAs would have recognized that Calder's teachings could have been used to make any suitable application package (including Blaser's application layers) executable on multiple, different OSs (including in different OS-families), which POSAs would have had motivation to do, for the reasons I discussed above.

[0100] In this Blaser-Calder combination (which I may also refer to as "Blaser-Calder"), the system would have comprised multiple servers, at least some of which would run different, conventional prior art OSs from each other (including OSs in different OS families), for the reasons I discussed above. For example, one or more servers would run Windows 95, one or more servers would

run Windows 98, one or more servers would run Windows 2000, one or more servers would run Unix-based OSs, one or more servers would run Linux-based OSs, etc.

[0101] Additionally, in the Blaser-Calder combination, each layer for use in the above multi-server system would have comprised, *inter alia*, the files that Blaser discloses as being included in layers (*see supra* §VII.A.1) – for example, application(s) and system files required to execute the applications on (at least) on servers running OSs that are native to the application(s) (which, as I discussed above in this section POSAs would have reasonable expected to also be executable on OSs in the same OS family as the native OS). In addition, based on Calder's teachings (*see supra* §VII.A.2), each layer would have also comprised modified system-file copies (e.g., modified libraries and configuration files) that were required to allow the layer (and the application(s) within the layer) to also execute on servers running OSs that are non-native to the layer's native OS (e.g., OSs in different OS families as the native OS). For example, a native Windows 2000 layer would have "system libraries [] added to" the layer that would execute software to "translate Windows 2000 system calls to Linux system calls" (Calder, [0101]), and vice versa.

[0102] POSAs would have recognized that it would have been a matter of design choice and tradeoffs (e.g., in simplicity of management versus cost of

testing and development) whether each Blaser-Calder layer would have been made executable on every OSs in the system, other whether to make some of the layers executable on just native OSs and OSs in the same family, and other layers executable on both native and non-native OSs (including OSs in different OS families).

[0103] In my opinion, POSAs would have reasonably expected success in arriving at the Blaser-Calder combination that I discussed above (and discuss in further detail below in the Grounds 2 where necessary to understand my opinion of how the combination would have met the challenged claims §VII.A.4). That is because Blaser's and Calder's techniques are each software-based techniques that can be applied together for use on the same conventional prior art OSs, and thus it would have required only a POSAs's ordinary skill to adapt those teachings (as necessary) for use in the Blaser-Calder combination that I discussed above.

[0104] For example, Blaser discloses that its invention can be implemented in "a **conventional** personal computer or workstation," or "other computing systems," such as "servers," having **conventional** computer components like "a CPU, memory, display," etc. Blaser, 3:15-24. Calder similarly discloses embodiments of its technique implemented in conventional servers and client computers. Calder, [0077]-[0082].

[0105] As I also discussed above, Blaser expressly discloses using application layers in large corporate networks (Blaser, 13:1-15) and Blaser further says layers can be used with conventional OSs, like servers running Windows-based OSs and servers running "operating systems other than Windows" (Blaser, 3:28-41). Calder's technique similarly allows one application package (i.e., the same application package) to be executable on a computer/server regardless of whether it was running a Windows- or non-Windows-based OS. Calder, [0101].

[0106] Thus, it would have required only a POSA's ordinary programming skill to make a Blaser layer executable on any of multiple servers running different OSs (including in different OS families) based on Calder's teachings.

4. Mapping of Blaser-Calder to Claims 1-4, 7-11, 14, and 16-30

[0107] As explained in the sections below (§§VII.A.4.a-y), in my opinion a POSA would have found the Blaser-Calder combination to have met all limitations of claims 1-4, 7-11, 14, and 16-30 of the '814 patent, and therefore would have rendered those claims obvious to POSAs.

a. Claim 1

[0108] The parties agree that claim 1's preamble is limiting. EX1106, 6. As I discuss below (§§VII.A.4.a.i-ix), the Blaser-Calder combination meets the preamble. I have divided claim 1's preamble into two parts addressed in separate

sections below: the preamble's "system" portion, which I have designated limitation [**1PREA**] (§VII.A.4.a.i), and the preamble's "method" portion, which I have designated limitation [**1PREB**] (§VII.A.4.a.ii).

i. [1PREA]

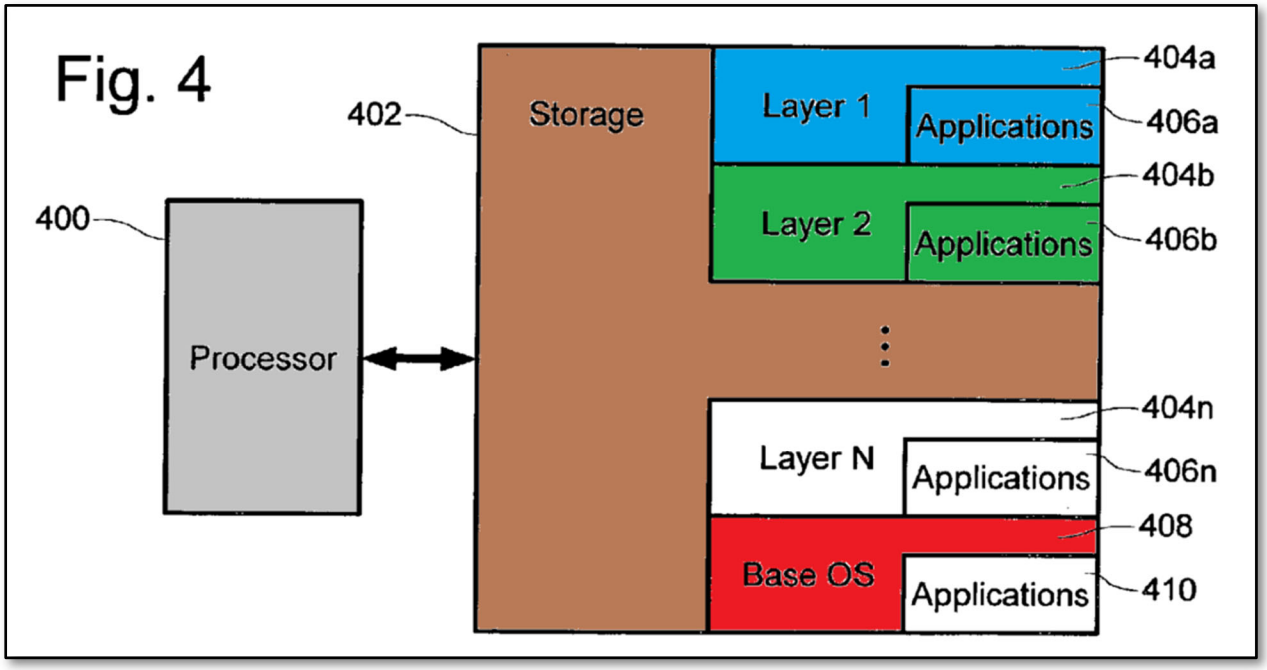
[0109] For convenience, I have further divided limitation [**1PREA**] into four sub-elements. For the reasons discussed below (§§VII.A.4.a.i.(1)-(4)), it is my opinion that Blaser-Calder meets [**1PREA**] under both parties' constructions.

(1) "1. In a system having a plurality of servers with operating systems that differ"

[0110] As I discussed above (§VI.A.5), the parties have jointly construed "*servers*" as "physical servers" (EX1106, 6) and VirtaMove further argues that the claim scope includes "a physical server running virtual machine software" (EX1105, 4).

[0111] Blaser's "computing systems" include "***servers***" (i.e., plural servers, not just one server). Blaser, 3:16-24. As shown in Blaser's Figure 4 (which I have colored below), Blaser uses physical (not merely software) servers (e.g., they are hardware-based server computers that include physical components, like a processor and storage devices). Thus, Blaser computing devices meet the parties' joint construction that "*servers*" means "physical servers." EX1106, 6; *see also* EX1100, 3; EX1105, 4-7; EX1101, 1; EX1001, FIG. 2 (illustrating example

computing devices that are similar to Blaser's Figure 4 and have computer "hardware").



[0112] Blaser's "layering system" provides for "transfer of applications between similar computing systems [plural]." Blaser, 7:6-23; Blaser, 8:55-62 ("files of a layer [can] be moved from one Windows system to another"). As I noted above, Blaser's "computing systems" include "***servers***" (i.e., plural servers). Blaser, 3:16-24. Thus, because Blaser layering system can be moved between different computing systems in Blaser's overall system, and Blaser says each of those computing systems can be *servers*, Blaser teaches deploying layers in a *system having a plurality of servers*.

[0113] Blaser's "[e]xample systems" execute various "Windows operating system[s]" – "Windows 95, 98, NT, 2000, and XP" (Blaser, 13:42-51),

which, as I explained above (§V.C), are prior art OSs that differ according to the statements made by the inventors of the '814 patent during its prosecution history because they are different versions of Windows-based OSs. In addition, Blaser discloses implementing layers in computing devices executing "operating systems other than Windows" (Blaser, 3:33-36), which are OS in different OS families from Windows (and in a conventional implementation, would have been different OS families from each other, e.g., Unix, Mac, OS/2, and/or Linux).

[0114] EX1088's definition of "*operating system*" – which is Google's construction (*see* EX1106, 4; EX1100, 2) – is "software that controls the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices" – and identifies "Windows 98, Windows NT…" as examples. EX1088, 378. POSAs understood that the conventional OSs that Blaser uses (e.g., Windows-based OSs) are software that control the allocation of hardware resources such as memory, CPU time, disk space, and peripheral devices, and thus meet Google's construction of "*operating system*" (EX1106, 4; EX1102, 4) and VirtaMove's "plain and ordinary meaning" construction and apparent view of the scope of the claimed "*operating system*" (*see* EX1106, 4; EX1103, 7-8 – not disputing that Google's construction is, at least, a valid exemplary definition of an operating system within the scope of the claims; EX1105, 4 (alleging that the parties do not "dispute [] the scope of" this term)).

[0115] My opinion regarding a POSA's understanding of a conventional prior art OS is corroborated by EX1088, 378, which, as I noted above, provides a definition of "*operating system*" that is consistent with a POSA's understanding of a conventional OS.

[0116] My opinion in that regard is further corroborated by Calder. Calder notes that "system resources" are typically accessed via "a system interface" to the computing device's OS. Calder, [0083]. Thus, Calder demonstrates that in a conventional computer, the computer's "resources" are controlled via the OS (which is why requests for resources are made via an interface to the OS).

[0117] My opinion that the conventional prior art OSs that Blaser and Calder say can be used with their techniques meet each party's construction of "*operating system*" is further corroborated by EX1121. EX1121 discloses that an "***operating system*** is the layer of software that nearly every application software package expects to be present" (EX1121, 2), to provide "services" that "are invoked with a special 'trap' or 'system call' instruction" (EX1121, 3). EX1121 further states that "[t]he functions of an operating system" include being "a resource managed" that "manages the hardware resources of the computer system," such as the "*Processor*," "*Memory*," "*Input/output (I/O) controllers*," "*Disk devices*," and "*Other devices*." EX1121, 4.

[0118] The ’814 patent discloses that exemplary OSs include “Linux, Windows and Unix systems” and that “the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used.” EX1001, 7:34-38.

[0119] Thus, the OSs in Blaser (and Blaser-Calder) meet Google’s construction of “operating system” **and** the patent’s exemplary OSs. EX1100, 2.

[0120] Thus, Blaser (and Blaser-Calder) meets a *system having a plurality of servers with operating systems that differ*.

[0121] Moreover, as I discussed above (§VII.A.3), POSAs would have had reason to implement Blaser-Calder in *a system having a plurality of servers with operating systems that differ*, under both parties’ constructions, including servers with OSs in different OS-families. For example, as I discussed above with corroborating evidence (§VII.A.3 (citing, e.g., Calder, [0083]; EX1121, 2-4)), a large corporate network having at least one server running a Windows-based OS (e.g., Windows-2000) and at least one other server running a non-Windows-based OS (e.g., a Unix- and/or Linux-based OS).

(2) “operating in disparate computing environments”

[0122] VirtaMove construes “*disparate computing environments*” as “[e]nvironments run by standalone or unrelated computers,” based on the “definition[]” in the specification. EX1103, 8; EX1001, 2:16-19. The ’814 patent

defines "disparate computing environments" as "[e]nvironments where computers are stand-alone or where there are plural computers and where they are unrelated" (EX1001, 2:17-19), but does not explain what the patent means to have "stand-alone" or "unrelated" computers.

[0123] As I discussed above (§VI.B.3), Google argues in the District Court litigation that this claim term is indefinite because it is unclear to POSAs what it means to have "unrelated" computers in context of the entire claim language. However, as I also discussed above (VI.B.3), Patent Owner construes the term as "[e]nvironments run by standalone ***or*** unrelated computers." EX1106, 4; EX1103, 8. While Google explains that it is unclear what "unrelated" means (EX1102, 5-7; §VI.B.3), in my opinion, as I discuss below, Blaser-Calder's plurality of servers (and client computers) would have been "stand alone" computing devices, which I understand is sufficient for the Board to find that Blaser-Calder meets *operating in disparate computing environments* even though it is unclear to POSAs what it means to have "unrelated" servers in context of the claim.

[0124] As I discussed above (§VII.A.4.a.i.(1)), Blaser-Calder's plural servers are separate computer machines and thus are stand-alone computers, which meets *operating in disparate computing environments*, even though it is unclear what makes computers "unrelated" in the context of claim 1. For example,

Blaser's FIG. 4 illustrates that each "computing device" (a server in Blaser-Calder's system) is a stand-alone computer that "includes a processor 400," attached to various hardware "such as memory, input devices or output devices" and "one or more storage devices." Blaser, 4:50-59.

[0125] Moreover, Patent Owner has argued that "standalone" "indicate[s] the ability of computers to operate independently of each other." EX1103, 9. The servers in Blaser-Calder would have (or in an obvious implementation thereof would have) operated independently of each other (e.g., because one server does not require the support of another server in order to execute applications on the server). Thus, Blaser-Calder's servers also meet *operating in disparate computing environments* under VirtaMove's apparent claim scope.

[0126] Furthermore, during prosecution, the examiner found that "plurality of servers…operating in disparate computing environments" was met by EX1010's disclosure of a "network environment" (EX1002, 239-240) that had a server and client (EX1089, FIG. 2, 6:28-35). The applicants did not disagree that a system like that taught *disparate computing environments*. As I discussed above (§VII.A.1-3), Blaser, Calder, and the Blaser-Calder combination include systems with servers and client computers operating in a network environment. For example, in the Blaser-Calder combination, end-user client computers can access

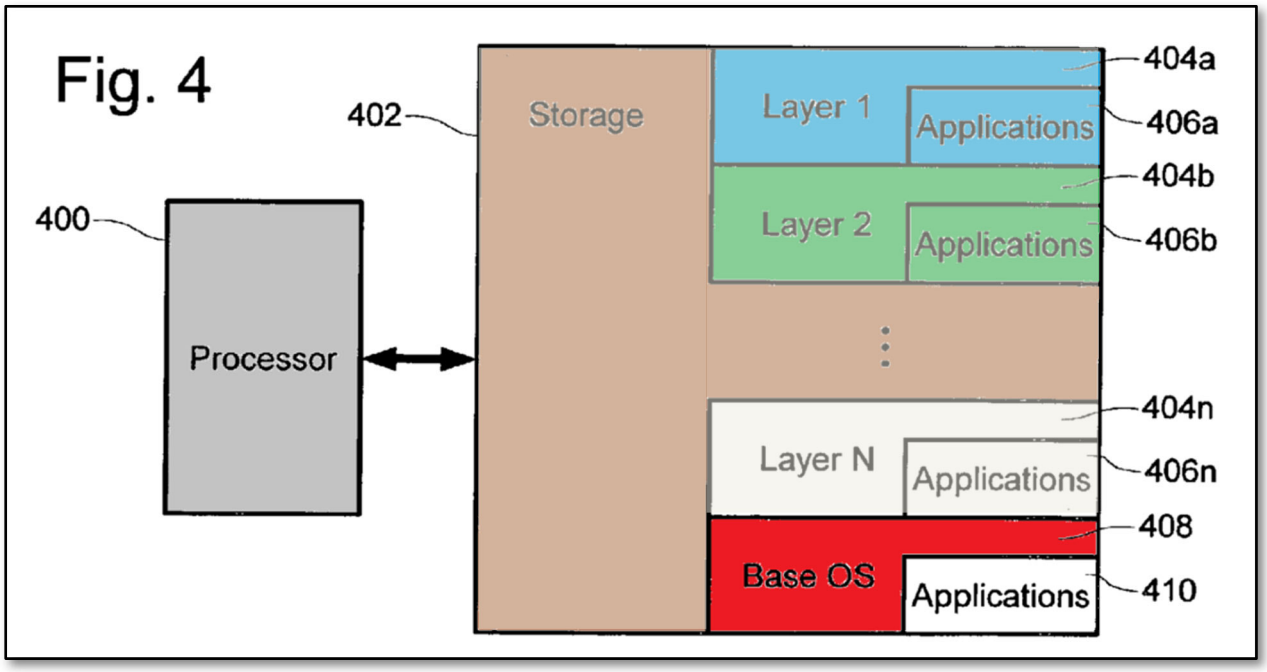
servers that provide Blaser-Calder layers (and the servers providing layers are or had reason to be clients of other servers, e.g., servers that manage and provide layers to the server computers that interact with end-user client computers). *See, e.g.*, Blaser, 7:59-62 (disclosing that "[s]ome systems provide a **multi-user environment** providing a facility for an administrator to designate layers accessible to individual users," which teaches or at least suggests and motivates multiple users using their personal client devices to share one of the server computers in the system), 12:55-65 (noting that the computing devices can communicate with "an auxiliary storage device," which conventionally would have been implemented as a server); Calder, [0007] (noting that one implementation of a system for use with Calder's techniques is a system with "a network, and **a client computer operably connected to the server computer** via the network, wherein the client computer receives the modified application from the server computer").

[0127] Thus, Blaser-Calder also meet *operating in disparate computing environments* under a claim scope found by the Patent Examiner that VirtaMove did not disagree with, as I explained above, even though it is unclear what makes computers "unrelated" in the context of claim 1.

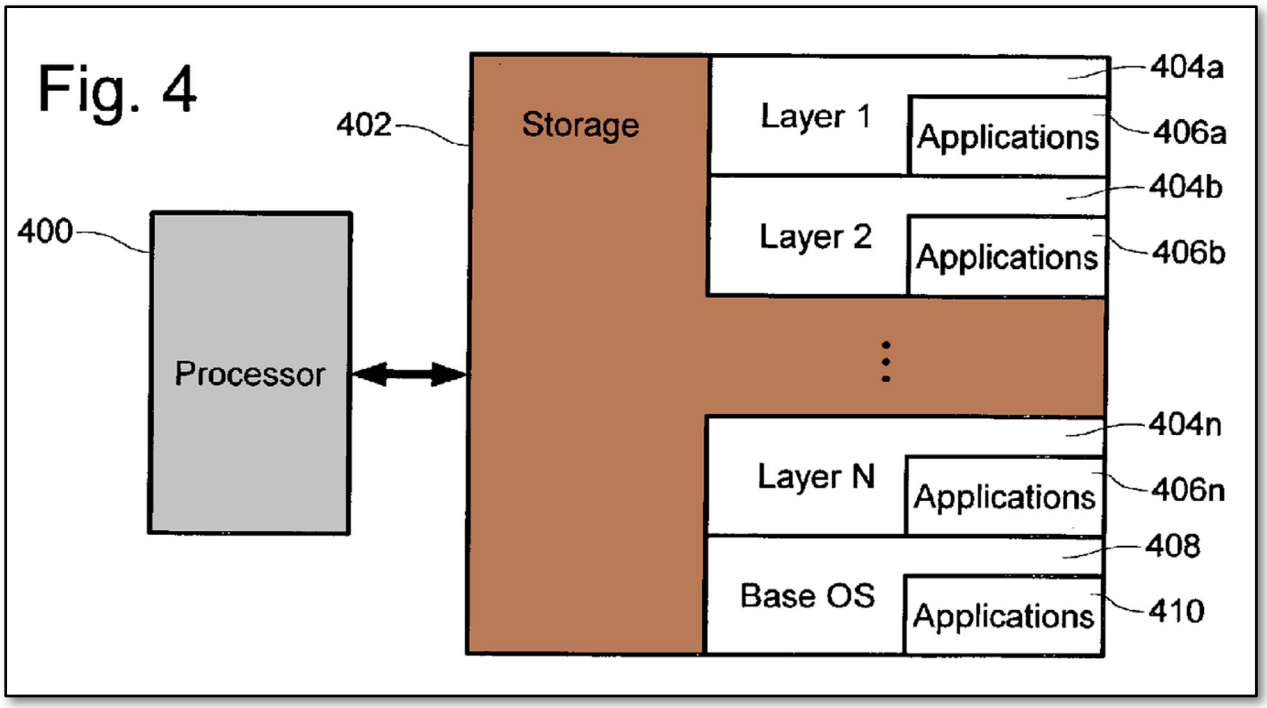
(3) "wherein each server includes a processor and an operating system"

[0128] As illustrated in Blaser's Figure 4 (which I have colored below), each "computing device" – which would have been *servers* in the Blaser-

Calder combination (*supra* §VII.A.3) – "includes a ***processor*** [gray]" and "a base ***operating system*** [red]." Blaser, 4:51-59. *See also* Calder, [0083] (a computer's "standard architecture" has an "operating system").



[0129] Blaser's example server computer "is a conventional" computer "having a CPU." Blaser, 3:15-27. POSAs understood the conventional processor/CPU for a conventional computer is a physical device, not merely software, and thus meets the parties' joint construction of *processor* as meaning a "physical computer processor." EX1106, 6. Blaser says its CPU may "have peripheral **devices** attached" to it (Blaser, 4:50-59) – for example the Storage device 402 that I have color-annotated below – which reinforces that the processor/CPU is a physical computer processor device.



[0130] As I discussed above (§VII.A.4.a.i.(1)), POSAs knew that the conventional prior art OSs like those that would have been used in Blaser-Calder are software that control the allocation of hardware resources such as memory, CPU time, disk space, and peripheral devices. *See* EX1088, 378 (providing a definition of an "*operating system*" that matches Google's construction is consistent with a POSA's understanding of a conventional OS). Thus, the conventional prior art OSs that would have been used in Blaser-Calder meet each party's construction of *operating system* (EX1106, 4), as I discussed above (§VII.A.4.a.i.(1)).

(4) "[OS] including a kernel [and][5] a set of associated local system files compatible with the processor"

[0131] The '814 patent defines "*system files*" as "files provided within an operating system and which are available to applications as shared libraries and configuration files." EX1001, 2:52-54, 2:55-3:19. In other words, the patent recognizes that *system files* like shared libraries and configuration files were not only known in the prior art but were often included as *local system files* with conventional prior art OSs.[6] Indeed, the patent provides in its "**BACKGROUND**" section an "example" of "shared libraries, supplied by the [Linux Apache] OS distribution, which are '*system' files*" and examples of "configuration files, also provided with the OS distribution." EX1001, 2:55-3:13; 3:14-19 (admitting that "these shared library files and configuration files form *system files* provided by the operating system" and "[t]here may be any number of other files included as *system files*," including "for example, to support maintenance activities or to start

[5] My understanding is that the parties agree "and" should be inserted. *See* EX1106, 6.

[6] Whether any OS available today includes *system files* is not something I have been asked to analyze and it would require me at least to study the documentation and operation of any particular OS.

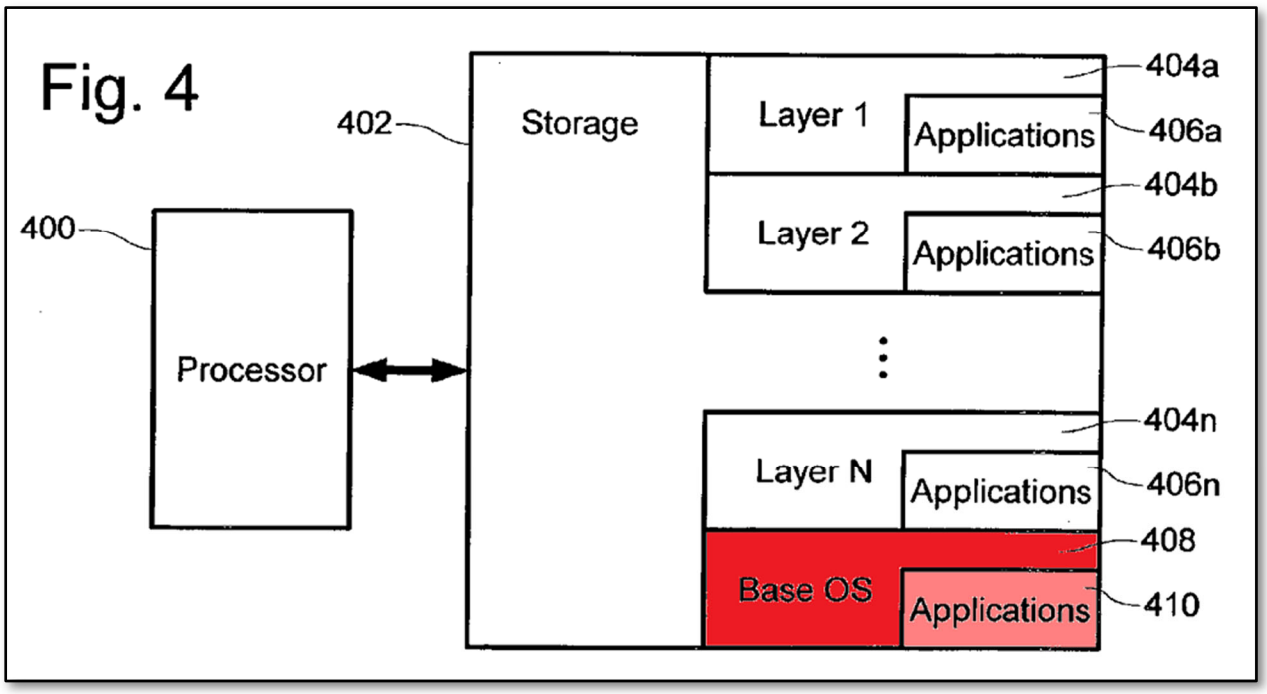
other network services to be associated with a container"); 10:39-50 (discussing making containers using "a copy of the system files provided with a given operating system (OS) distribution"; i.e., using "the OS provided system files before an application is installed").

[0132] In Blaser, a layer is "overlaid over the regular operating system file system" that exists in the server's Base OS. Blaser, 5:18-26. Blaser's "regular operating system files" include the same conventional system files that the '814 patent identifies as example of system files: "**[s]hared libraries** (such as DLLs)" **and** "**system accessible *configuration*** (such as registry entries)." Blaser, 5:20-32. POSAs understood that those example ***configuration*** setting (like registry entries) are (or in a conventional implementation would have been) stored in a corresponding configuration file (e.g. a registry configuration file). *See also* Blaser, 3:42-58 (disclosing that "an application layer" includes "a group of **files** in combination with any associated application **configuration** stored to **operating *system files***"; disclosing further that layers "provide a barrier between applications" to allow "conflicting **configuration or library files**").

[0133] My opinion that configuration settings, like registry entries, were conventionally stored in one or more corresponding files in a conventional prior art OS's local file system is corroborated by EX1108. EX1108 discloses as background information that "[u]sers can tailor the operation of [various

applications] applications to suit their needs by specifying various **configuration parameters**." EX1108, [0003]. EX1108 further discloses as background information that the "values of the **configuration parameters can be stored in application-specific configuration *files*** such as **UNIX resource files** or in a central registry such as the **Windows® registry file**." EX1108, [0003].

[0134] As shown in Blaser's Figure 4 (which I have colored below), some applications (labelled 410; light red) are installed to the Base OS (red) rather than a layer, and thus execute outside of the layers.



POSAs thus understood that those applications use local *system files* that are *associated* with the Base OS that are separate from the *system files* that are inside the layers. I note that Blaser having a server with layers and applications that are associated with the Base OS and executing outside Blaser's layering architecture is

consistent with the '814 patent's disclosure that "applications executing within a secure container **can co-exist with applications that are <u>not associated with a container</u>, but which are <u>associated with the underlying operating system</u>**." EX1001, 7:12-15; *see also* EX1079, 28 (VirtaMove arguing: "[I]n the Accused Instrumentalities the associated system files provide at least some of the same functionalities as the associated local system files. The host/node's associated local system files remain resident on the host/node, **for example for use by system processes or applications outside the container environment**."). *See also infra* §VII.A.4.a.iii.(1).b) (discussing how Blaser-Calder's layers meet the claimed *containers*).

[0135] Thus, in Blaser, the server's Base OS has and uses (or in an obvious implementation using a conventional prior art OS would have had) *local system files* and as I discuss below a corresponding *kernel*.

[0136] EX1088's definition of "*kernel*" – which is Google's construction (EX1106, 4) is "[t]he core of an operating system—the portion of the system that manages memory, files, and peripheral devices; maintains the time and date; launches applications; and allocates system resources." EX1088, 300. As I explain below, each conventional prior art OS that would have been used in Blaser-Calder would have each included a conventional prior art OS *kernel* that

would have met Google's construction and VirtaMove's "plain and ordinary meaning" construction of "*kernel*." EX1106, 4.

[0137] POSAs knew that the conventional prior art OSs that Blaser supports – Windows-based OSs ([0085]) and "operating systems other than Windows" ([0023]) – include a *kernel* and *local system files* (or would have included a *kernel* and *local system files* in a conventional implementation thereof). I note that VirtaMove's infringement contentions argue that an OS "by definition, includes a kernel." EX1103, 12. VirtaMove's argument is consistent with my opinion that POSAs understood that the conventional prior art OSs that Blaser's uses include (or in an obvious implementation would have included) a *kernel*.

[0138] As I noted above, the '814 patent itself appears to recognize that *local system files* were known in the prior art and were conventionally "supplied by the OS distribution" of a conventional prior art OS (e.g., a prior art Linux-based OS). EX1001, 2:55-3:19. Indeed, the '814 patent itself does not purport to have invented the use and/or inclusion of *a kernel and set of local system files compatible with the processor*, but rather relies on pre-existing *kernels* and *sets of local system files compatible with* a server's *processor*. In any event, my opinion that the conventional OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is corroborated the prior art I discuss below.

[0139] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised a *kernel and local system files*) is corroborated by EX1029. EX1029 discloses that the "minimum…OS components **include[] a *kernel***, hardware drivers, **and *system files***." EX1029, [0027].

[0140] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1023. EX1023 discloses that Windows-based OSs have "***system files*** which form the ***kernel*** part of the operating system." EX1023, [0023]. EX1023 further notes that these OSs have "customized **configuration files** [examples of *system files* in the '814 patent]" and "***system files*** of the operating system." EX1023, [0027].

[0141] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1011. EX1011 discloses that "[e]very windows application is registered in a **windows *kernel*** with an application name." EX1011, [0032].

[0142] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1067.

EX1067 discloses that conventional "**operating *system files***" include "[t]he operating system ***kernel***, which forms the operating system's core program." EX1067, 9:12-20.

[0143] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1064. EX1064 discloses:

> **A traditional operating system is logically layered and divided into two main portions: the *kernel*** and user programs.[7] The kernel interfaces with and controls the hardware, and also provides the user programs with a set of abstract system services called system calls. The kernel runs at a kernel level, where it can execute privileged operations and allows the kernel to have full control over the hardware as well as user level programs. This centralization provides an environment where all programs share the underlying hardware in a coordinated fashion.

[7] Though EX1064 calls the "two main portions" "the kernel [or 'kernel level'] and user **programs**," POSAs conventionally refer to the two main portions of a conventional OS as "kernel **mode**" and "user **mode**." E.g., 1034, [0041] ("Kernel mode is a privileged processor mode that allows system code to have direct access to hardware resources."), [0047] ("A user mode is a mode of the host microprocessor that does not allow code to directly access hardware resources.").

EX1064, 1:53-62.

[0144] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1062. EX1062 discloses that "[m]any computing platforms **support standard shared libraries**. For example, the **Windows platform supports Dynamic Link Libraries (DLLs)**, the **Unix platform supports Shared Libraries**, and the **Macintosh platform supports Code Fragments**." EX1062, 3:37-40. As I discussed above, libraries are examples of *system files* identified in the '814 patent. Thus, EX1062 corroborates my opinion that conventional OSs like those discussed in Blaser (and Calder) had *local system files* (e.g., shared libraries).

[0145] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1094. EX1094 discusses some of the "standard directories that will be found in many UNIX [OS]," which include:

> /bin This directory contains the basic system commands.
>
> **/etc** This directory contains **system configuration files** and programs used for administrating the system.
>
> **/lib** This directory contains the **system libraries**.

/tmp This directory is used to store temporary files.

/usr/bin This directory contains the commands that are not stored in /bin.

/usr/man This directory contains manual pages for programs

/usr/local This directory contains local programs that were installed by the system administrator (sysadmin) and were not included with the original system. In particular, /usr/local/bin contains local command files (binaries), and /usr/local/man contains local manual pages.

/home The actual directory location varies from system to system, but somewhere on the system will be a location where all of the users' home directories are located.

EX1094, 3:34-63. The files conventionally found in the /etc and /lib directories of a conventional Unix OS file system are examples of *system files* identified in the '814 patent. *See* EX1001, 2:55-3:13 (identifies example "'system' files" in /lib and /etc directories).

[0146] My opinion that the conventional prior art OSs that Blaser discusses for its servers comprise (or in a conventional implementation would have comprised) a *kernel and local system files* is further corroborated by EX1046, which discloses (as POSAs knew) that the "***kernel*** is the core of the operating system." EX1046, 1:25-26.

[0147] Thus, each conventional prior art OS kernel that would have been used in each Blaser-Calder server meets Google's construction and VirtaMove's "plain and ordinary meaning" construction of "*kernel*." EX1106, 4.

[0148] Because each server's processor in Blaser would be able to use the Base OS's *kernel and set of associated local system files* discussed above, the kernel and system files are compatible with the server's processor. Otherwise, if the kernel and *local system files* were not compatible with the server's processor, the processor would not be able to use the *kernel* and *local system files*.

[0149] My opinion that the conventional *kernel* and *set of associated local system files* for the conventional OSs that Blaser discusses for its servers are (or in a conventional implementation would have been) compatible with the server's processor is further corroborated by EX1067. EX1067 discloses that "for Windows 2000 to boot correctly on a computer…**the active kernel must be compatible with the computer's processor** configuration." EX1067, 35:48-54

[0150] My opinion that the conventional *kernel* and *set of associated local system files* for the conventional OSs that Blaser discusses for its servers are (or in a conventional implementation would have been) compatible with the server's processor is further corroborated by EX1045. EX1045 discloses that "kernel programs tend not to be compatible from processor to processor." EX1045, 16:60-62. Similarly, EX1043 discloses that "computer programs written

for" a processor "are frequently unsuitable for use with computers based on a different processor." EX1043, 1:17-20. In other words, EX1043 and EX1045 corroborate that because the processors in Blaser's servers **are** able to run the corresponding Base OS installed to the server (and its corresponding kernel and local system files) POSAs understood that each processor is compatible with that kernel and set of associated local system files.

* * *

[0151] Thus, for the reasons I discussed above (§§VII.A.4.a.i.(1)-(4)), Blaser-Calder meets [**1PREA**].

ii. [1PREB]

[0152] For convenience, I have further divided limitation [**1PREB**] into four sub-elements. For the reasons discussed below (§§VII.A.4.a.ii.(1)-(4)), it is my opinion that Blaser-Calder meets [**1PREB**] under both parties' constructions.

(1) "a method of providing at least some of the servers in the system with secure, executable, applications"

[0153] Blaser-Calder includes Blaser's method, which "provide[s] an installation for an ***application***" that "is [] installed to the destination system [which are Blaser-Calder's servers], but isolated in a layer." Blaser, 10:25-42; *see also* Calder, [0076] ("application package" includes "an application binary (also called application program)"). Blaser's layering method provides "***Secure Applications***"

and discloses that, when in Blaser's layers, "[a]pplications can be protected from unauthorized access." Blaser, 11:20-38. For example, an application "isolated in a layer" "is protected from corruption from other applications or meddling." Blaser, 10:37-42. *See also* Calder, [0073] (which teaches that its technique, which is adapted for use in Blaser-Calder's servers, modifies "application package to be executed **safely [and]** ***securely***"), [0076] (providing additional security by "prevent[ing] the application program from accessing certain files and directories on the client machine"). Thus, the applications in Blaser-Calder provided to the servers would have been *secure applications*.

[0154] Blaser discloses that an "***application*** 200 is running [i.e., ***executing***] on a layered computing system." Blaser, 4:27-30, col. 20 page 17 ("FSLActivate" function identifies "***Applications*** in the layer that are specified to be **run** on system startup…are started."). Calder similarly teaches that its "application package" will be "***executed***" (Calder, [0073]) so that the servers that install a package can "***execut[e]*** **an** ***application***" that is within the package (Calder, [0083]).

[0155] Thus, the applications in Blaser-Calder provided to the servers would have been *executable applications*.

[0156] Thus, Blaser-Calder's method would have *provided at least some of the servers in the system with secure, executable, applications*, as claimed.

(2) "[applications] related to a service"

[0157] As I discussed above (§VI.B.4), Google construes ***service*** to be "specialized, software-based functionality provided by network servers and comprised of one or more applications" (EX1102, 7); VirtaMove contends that "[n]o construction [is] necessary" and that the "plain and ordinary meaning" applies (EX1103, 9).

[0158] The '814 patent discusses conventional website-related applications related to corresponding services. For example, the patent says website-related "applications such as Apache, MySql and PHP" can be placed "in a single container for supporting a single ***service***, such as a **web based** human resource or customer management type of *service*." EX1001, 4:26-31. The patent identifies additional conventional "***services***" like "CRM (Customer Relation Management) tools, Accounting, and Inventory." EX1001, 7:18-21.

[0159] Blaser similarly discloses application "layers defined for different types of users, for example, secretaries, engineers and accountants." Blaser, 13:1-15. For example, "[t]he secretarial layer contains…secretarial applications" and "[t]he accounting layer includes accounting software." Blaser, 13:1-15.

[0160] Each layer in Blaser provides a conventional ***service*** because POSAs understood the above applications for different users are specialized,

software-based functionality provided by Blaser-Calder's network servers (e.g., specialized engineering software for the engineering layer; specialized accounting software for the accounting layer; etc.). Thus, each layer in Blaser is comprised of one or more applications (e.g., secretarial, engineering, and/or accounting applications) (EX1088, 475) that provide different, specialized, software-based functionality for different types of users (secretaries vs. engineers vs. accountants). Those applications are also like the patent's example services discussed above (e.g., applications for an Accounting service). EX1001, 7:18-21. EX1100, 3. Thus, each of the *services* that would have been provided by a Blaser-Calder layer's applications are *related to a service* under Google's construction and under VirtaMove's "plain and ordinary meaning" construction of "*service*." EX1106, 4.

[0161] To the extent VirtaMove's "plain and ordinary meaning" construction permits the claimed *services* to include services accessed locally, POSAs knew that Blaser's example services (e.g., engineering services for an engineering layer) could also be accessed locally from a Blaser-Calder having that service installed thereon.

[0162] Thus, each layer's applications in Blaser-Calder *are related to a service* (e.g., secretarial service, engineering service, accounting service, etc.) under each party's constructions.

(3) "wherein the applications are executed in a secure environment"

[0163] Blaser's layers provide "***Secure*** Applications" and says "[a]pplications can be **protected from unauthorized access**." Blaser, 11:20-38. Layers "**protect** application files from viewing and copying" because "[a]ny [executing] application of the computing system desiring to open files within the layer" must supply "an authentication key or token." Blaser, 11:21-38. Layers "may additionally be **encrypted**" (Blaser, 11:31-34) and "authenticat[e] a user before enabling a layer" (Blaser, 11:59-62).

[0164] Based on Calder, POSAs would have also had motivation to modify Blaser-Calder's application packages installed to Blaser-Calder layers to "protect the contents of the project from improper tampering" and "protect the non-secure machine from improper tampering by the project." Calder, [0006].

[0165] For example, teaches to modify an "***application*** package" so it is "***executed*** safely, ***securely***, and transparently on a remote machine." Calder, [0073]. Calder discloses providing that protection by "intercepting" "system calls made by the application" to restrict "access [to] approved files" in the file system, thereby protecting the servers (and their file system) and "the contents of the application package" from corruption by each other. Calder, [0087]. Calder also teaches to use "***security*** information" in "application ***package***" that "protect[s]

both the client computer [], as well as the contents of the application" from corruption from each other. Calder, [0100].

[0166] Thus, based on Calder, POSAs would have also had motivation to implement at least some layers to have their system calls intercepted in the manner Calder describes so as to further *secure* the *environments* that Blaser-Calder's layers provided for *executing applications*, to protect the applications from improper tampering and protect the server on which the layers were installed from corruption by the applications.

[0167] POASs would have reasonably expected success in doing so because, as I discussed above (§VII.A.2), Calder's techniques are based on software executable in the same conventional OSs that would have been used in Blaser-Calder, and thus it would have required a POSA's ordinary programming skill to incorporate in Blaser-Calder's layers software to intercept system calls, as taught by Calder, to protect the server and application package.

[0168] For either of the above-discussed reasons, Blaser-Calder's layers would have provided a *secure environment* for executing applications.

[0169] Thus, applications executing Blaser-Calder's layers would have been applications that *are executed in a secure environment*. *See also infra* §VII.A.4.a.iii.(1).c) (where I demonstrate that Blaser-Calder layers are each a *secure application container*).

(4) "wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service"

[0170] The '814 patent discloses an example where "**[a]n icon is an object** supported by a GUI [graphical user interface]" that "**[n]ormally an icon associates an image with an *object*** [e.g., a small graphic corresponding to an application] that can be operated on through a GUI." EX1001, 14:21-23. For example, the patent says that "the GUI supports the ability to **select an *object*, move it and initiate an operation**" – for example, "copy[ing] and transfer[ring] of files" – "when the ***object*** is placed on a specific destination." EX1001, 14:4-8. Thus, examples of *objects* in the '814 patent are conventional icons associated with applications or with files that can be moved around a conventional GUI and *executed* on by clicking the icon or dragging-and-dropping the icon (e.g., moving the icon of an application or file to a GUI image of a directory can copy or move the corresponding file / application file(s) to that directory).

[0171] Additional examples of "***tasks***" in the '814 patent that use icon *objects* include "[i]nstall[ing] application specific files" and "[s]tarting a first application." EX1001, 15:35-56.

[0172] Thus, the '814 patent discloses conventional uses of application icons in prior art GUI-based OS (e.g., clicking an icon object to activate an application's executable object). The same would have been true of two or more

of the conventional prior OSs that would have been used on two or more of the servers in the Blaser-Calder system (*see supra* §VII.A.3). That is, the conventional prior art OSs that Blaser-Calder would have used in its plurality of servers (e.g., Linux- and Windows-based OSs) used and/or beneficially supported installation of GUIs, where applications would each include an object (icons, application executables, and other application files) that would have been executable by the different OSs in the system.

[0173] My opinion that the conventional prior art OSs that Blaser discusses included GUIs where applications were associated with one or more icon objects that were executable by the OS for performing a task related to a service (e.g., an icon for an accounting program related to performing a task related to an accounting service) regard is further corroborated by EX1047 discloses as background information:

> In graphical user interfaces employing **an *object*-oriented programming paradigm, *application* programs are typically represented to a user by an icon** …. ***Execution* of an *application* program is initiated by selecting its corresponding icon**, ….

EX1047, 1:21-67.

[0174] My opinion that the conventional prior art OSs that Blaser discusses included GUIs where applications were associated with one or more icon objects that were executable by the OS for performing a task related to a service

(e.g., an icon for an accounting program related to performing a task related to an accounting service) regard is further corroborated by EX1059. EX1059 discloses as background information:

> Hardware changes force the development of separate software executables (e.g., **software *objects***) for the different hardware components. **A software *object* is a single software *executable*.** Conventionally, each hardware component type requires its own object or executable.

EX1059, 1:34-38.

[0175] My opinion that the conventional prior art OSs that Blaser discusses included GUIs where applications were associated with one or more icon objects that were executable by the OS for performing a task related to a service (e.g., an icon for an accounting program related to performing a task related to an accounting service) regard is further corroborated by EX1047. EX1047 discloses as background information:

> Execution of an application program is initiated by selecting its corresponding icon, most often using a pointing device such as a mouse. **When an application program is selected, a message is sent to the corresponding application program *object*, indicating that the application program object is to invoke certain of its methods.** For example, if a word processing program is selected, the methods contained within **the application program *object* may include starting [i.e., *executing*] the word processing program**.

EX1047, 1:43-51.

[0176] Blaser's exemplary prior art OSs which would have been used in Blaser-Calder (e.g., Windows-based OSs and Linux-based OSs), are conventional GUI-based OSs and associate executable applications with icons ("objects"), where the icons are conventionally included with the application. The icon objects were conventionally operated upon (e.g., clicked) by a user to cause the OS to execute the corresponding application associated with the icon. Blaser, 3:28-36. Blaser also says there are "layers that represent different environments" and each include their own "icons and menus," which are conventional objects executable by the OS that is executing the corresponding application. Blaser, 13:25-28.

[0177] Blaser's exemplary application objects (e.g., icons) are *executable for performing a task related to the service*—e.g., the icon for a word-processing application would be for performing a task related to the secretarial service; the icon for a CAD (computer automated design) application would be for performing a task related to the engineering service. Blaser, 13:1-15. More generally, Blaser discloses that "suppose layer A contained **B.EXE** [an executable file]. When **B.EXE** [another executable file] executes, it results in process C running." Blaser, 15:16-37. In GUI-based OSs like those used in Blaser-Calder, the conventional way to execute B.EXE (which is an executable file) is via its

corresponding icon (object). Thus, each application's .EXE file discussed above is an executable object included with the applications.

[0178] Moreover, POSAs knew that each .EXE file contains *object* code that is *executable for performing a task related to the service* provided by the corresponding application associated with the .EXE file. Thus, if VirtaMove contends that an object executable is *an object executable by at least some of the different operating systems for performing a task related to the service* the .EXE files in Blaser-Calder meets the portion of limitation [**1PRE**] discussed in this section of my declaration.

[0179] Furthermore, POSAs knew that when using a conventional OS "[a] *program* is a static ***object*** that can exist in a file" and "[a] *process* is a dynamic ***object*** that is a program in ***execution***." EX1121, 43 (italics of "program" and "process" original). As I discussed above, Blaser-Calder's application layers (*containers*) each include one or more application programs for executing one or more services (e.g., engineering programs for executing engineering-related services). Thus, the applications in each Blaser-Calder layer are each associated with programs that exist as processes when executing and, therefore, each application is *each an object executable by at least some of the different operating systems for performing a task related to the service* associated with that application program.

[0180] My opinion that Blaser-Calder's servers included (or had reason to include) *applications* that *each include an object executable by at least of the different OS for performing a task related to the service* is further corroborated by EX1047. EX1047 discloses as background information that an OS (GUI's "**[e]xecution of an application program is initiated by selecting its corresponding icon**." EX1047, 1:39-45. EX1059 discloses as background information that another common "**software *object* is a single software *executable***." EX1059, 1:34-39.

[0181] My opinion that Blaser-Calder's servers included (or had reason to include) *applications* that *each include an object executable by at least of the different OS for performing a task related to the service* is further corroborated by EX1095. EX1095 discloses:

> ***Objects* such as documents, applications, and OS components are represented by icons.** While the primary means of interfacing with these types of systems are the menus, there are limited visual facilities that allow operation of the system purely via manipulation of the objects themselves. For example, Apple's Macintosh System 7 and MicroSoft's [sic] Windows 95 **allows the user to delete files by moving the icons representative of the files to a delete function *object* called "Trash" or the "Recycle Bin"**.

EX1095, 8:4-13.

[0182] My opinion that Blaser-Calder's servers included (or had reason to include) *applications* that *each include an object executable by at least of the different OS for performing a task related to the service* is further corroborated by EX1096, which states:

> Typically, **an application is initiated** [i.e. *executed*] when a user, using the mouse 70, moves a pointer, or cursor, (not shown) onto the **application's icon and double clicks a button on the mouse** 70.

EX1096, 4:7-10.

[0183] As I discussed above in this section, exemplary "tasks" in the '814 patent include "[i]nstall[ing] application specific files" and "[s]tart[ing] a first application." EX1001, 15:52-56. As I also discussed above in this section with corroboration, installing and starting applications using icons were the types of conventional types of tasks performed by and within Blaser-Calder's layers.

[0184] Moreover, the "process [] running" in Blaser in response to executing an application (Blaser, 15:35-37) is itself a *task related to the service* because POSAs understood that "[a] 'task' is also a process," which itself "may be several processes." EX1030, [0002]. Thus, POSAs understood that whenever a program would have been executed in Blaser-Calder, the operating system would have created a new task or process for the program being executed.

[0185] My opinion that Blaser's running processes are each a *task related to the* corresponding *service* (e.g., a process associated with a Word-processing application is a task related to the secretarial service) is further corroborated by EX1042. EX1042 discloses that:

> [I]t is often useful to structure the program code executing within a process into so-called 'tasks.' **A task is a software entity having an associated body of program code which is executed whenever program control within the process is vested in that task**.

EX1042, 1:5-20.

[0186] My opinion that Blaser's running processes are each a *task related to the* corresponding *service* (e.g., a process associated with a Word-processing application is a task related to the secretarial service) is further corroborated by EX1061. EX1061 discloses:

> Typically, a computer system/computer has an operating system, and **a computer application is instantiated** on the computer **in the form of one or more application processes** running in a work space managed by the operating system.

EX1061, 1:13-16.

[0187] My opinion that Blaser's running processes are each a *task related to the* corresponding *service* (e.g., a process associated with a Word-

processing application is a task related to the secretarial service) is further corroborated by EX1064. EX1064 discloses:

> When a user requests **a program to be executed, a new process is created to encompass its execution**. The process is a combination of the program plus the current state of its execution that normally includes the values of all variables, as well as the conditions of the hardware (the program counter, registers and condition code, among others and the contents of the address space). The process exists within the system until it terminates, either by itself as designed, by the kernel, or by the request of the user.

EX1064, 2:12-23.

[0188] My opinion that Blaser's running processes are each a *task related to the* corresponding *service* (e.g., a process associated with a Word-processing application is a task related to the secretarial service) is further corroborated by EX1052. EX1052 discloses an "**application program** is the set of software that **performs a task desired by the user**, using computer resources made available through the operating system." EX1052, 4:28-31.

[0189] Another conventional service beneficially provided in a conventional networked environment like Blaser-Calder's, would have been a conventional network-attached database that is run on the server-side, to support application executed and/or configured by a client device (which conventionally was often itself a server).

[0190] As I discussed above (§VII.A.3; *see also infra* §VII.A.4.a.iii.(1).b), in Blaser-Calder each layer is configured with system files (e.g., libraries and configuration files. *See* Blaser, 5:18-32, Calder, [0082]. Those system files permit the application's object(s) that I discussed above "to execute and communicate on any non-native [or native] platforms" (i.e., different OSs in the Blaser-Calder system. Calder, [0101].

[0191] As I discussed above (§VII.A.3), Google construes *at least some of the different operating systems* as "at least two or more of the different operating systems" (EX1106, 4; EX1102, 11); VirtaMove contends that "[n]o construction is necessary" and that the "plain and ordinary meaning" applies (EX1106, 4; EX1103, 12).

[0192] As I discussed above (§VII.A.3), in Blaser-Calder, there would have been a plurality of servers having OSs that differ (i.e., at least two or more different OS), thus meeting Google's construction of *at least some of the different operating systems*. *See* EX1106, 4. As I discussed above in this section, Blaser-Calder is also consistent with the patent's example disclosure of *applications each including an object executable by at least some of the different operating systems for performing a task related to the service*, and thus meets VirtaMove's "plain and ordinary meaning" construction. *See* EX1106, 4.

[0193] Thus, in Blaser-Calder *the applications each include an object executable by at least some of the different operating systems for performing a task related to the service*, as claimed.

* * *

[0194] Thus, for the reasons I discussed above (§§VII.A.4.a.ii.(1)-(4)), Blaser-Calder meets [**1PREB**], and thus meets claim 1's preamble.

iii. [1A]

[0195] For convenience, I have divided limitation [**1A**] into four sub-elements. For the reasons discussed below (§§VII.A.4.a.iii.(1)-(4)), it is my opinion that Blaser-Calder meets limitation [**1A**] under both parties' constructions.

(1) "storing in memory accessible to at least some of the servers a plurality of secure containers of application software"

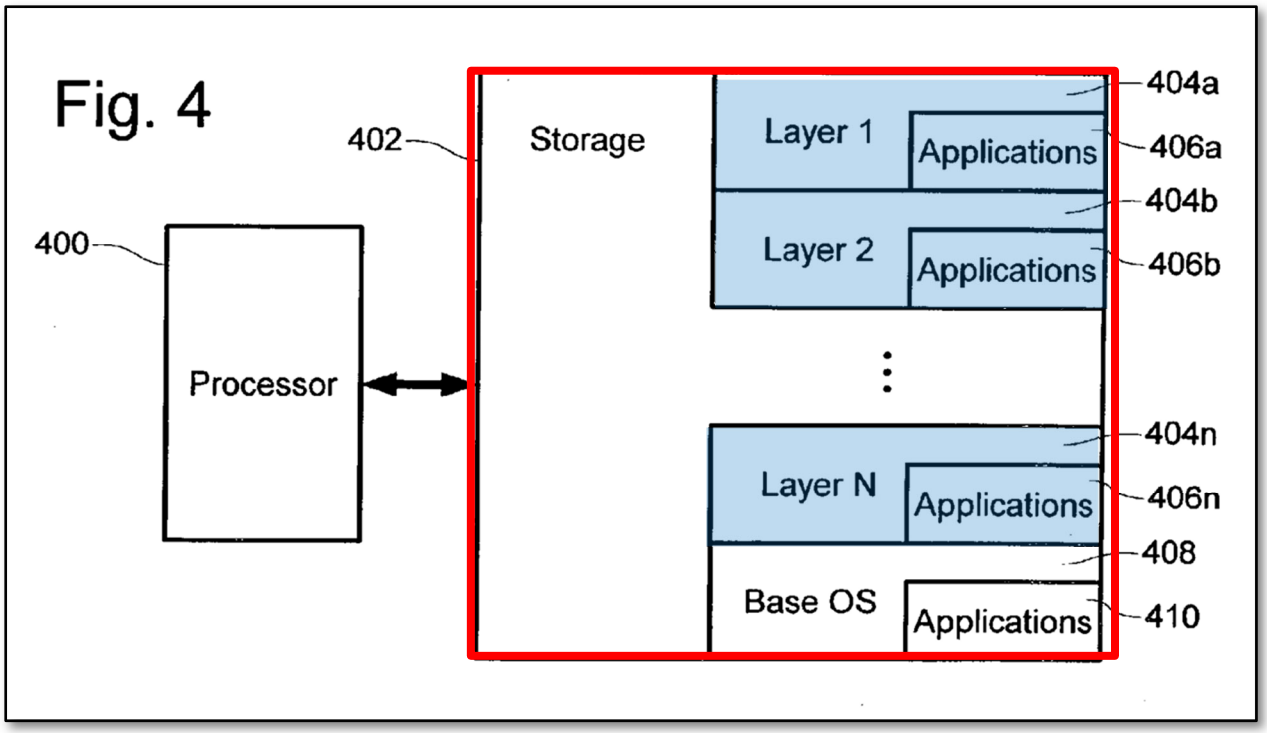
[0196] For convenience, I have further divided limitation [**1A**] into three sub-elements. For the reasons discussed below (§§VII.A.4.a.iii.(1).a)-c)), it is my opinion that Blaser-Calder meets limitation [**1A**] under both parties' constructions.

a) Blaser-Calder's layers (containers) are "stor[ed] in memory accessible to…the servers"

[0197] Below in §VII.A.4.a.iii.(1).b), I demonstrate that Blaser-Calder's layers meet the claimed *containers*. In this section, I demonstrate that

Blaser-Calder's layers would have been *stored in memory accessible to the* Blaser-Calder servers.

[0198] As illustrated in Blaser's Figure 4 (which I have colored below), Blaser's layers (e.g., Layers 1-N, highlighted in blue) are stored on the *memory* of storage device 402 (red), which is *accessible* to that computing device (a *server* in Blaser-Calder) containing processor 400. For example, Blaser discloses that "layers 404a-n are also contained on storage 402" (4:51-59) and, as I discussed above, Blaser's computing devices include "server" (3:16-27). Blaser further discloses that "a layer is a higher order storage unit." Blaser, 3:59-4:6. Blaser's layers can be "stored on separate disk partitions or remote file systems" *accessible* to the *server(s)* (6:62-7:23), with "each layer spanning one or more fixed disks" *accessible* to the *server*(s) retrieving and installing the layer (7:36-47).



[0199] The above meets VirtaMove's construction because that construction only requires the claimed *memory* to be *accessible* to one *server*. *See* EX1103, 12; EX1106, 4. Thus, if VirtaMove's constructions including *storing containers in memory accessible to the servers* using local memory of the server executing the *container*, then Blaser-Calder's layers (*containers*) meets that too because Blaser discloses that application layers can also be *stored in* local *memory* of the *server* executing the layer (and its applications), as I demonstrated above. Blaser, 9:32-34 (disclosing that "[a]ll of the capture data [in a layer] is held separate from the regular OS either **locally** or remotely in a data file, hard disk partition, or some other container"), 12:55-67 ("A layering system might also be used to provide a 'bottomless' virtual storage device. In that system[,] an auxiliary

storage device is presented by layering system software as being available on a main **local storage device** (e.g., C:). The auxiliary storage device might be **an additional <u>local</u> fixed disk**, network drive or file system, or other storage resource.").

[0200] Google's construction of "*memory accessible to at least some of the servers* is "memory that at least two or more of the servers can read from or write to." EX1106, 4; *supra* §VI.6. Based on Blaser, Blaser-Calder meets Google's construction. Blaser discloses that layers "may be ***stored*** to a network server and **retrieved** and ***stored*** as required to any computer of a computer farm" (e.g., a collection of Blaser-Calder *servers*). Blaser, 12:35-44. POSAs understood that when a layer is ***stored*** to that network server the layers are written to memory at that network server or memory that is accessible to that network server and that when a layer is **retrieved** the layer is being **read** from that memory.

[0201] Calder similarly teaches a server (server system) that "act[s] as the master control center" that would have been *accessible to* the network's "***server[s]***.'' Calder, [0078]-[0079]. POSAs would have had reason to implement a similar master control center in Blaser-Calder (in Blaser's network server or another conventional physical server) for the benefits Calder identified: "for all of the data processing, data transmissions, security information, and results

processing" as application packages are transferred to server devices in the system. Calder, [0078].

[0202] Thus, Blaser-Calder meets Google's construction because it would have been *storing* layers (containers) *on memory accessible to the servers* (e.g., the memory of the network server's storage device(s) from which layers are stored to and retrieved from). *See also infra* §VII.A.4.a.iii.(1).c) (where I demonstrate that Blaser-Calder's layers would have been "*secure containers of application software*").

b) Blaser-Calder's layers are "containers of application software"

[0203] Blaser explicitly says each layer "provides a convenient ***container*** to limit access to an application." Blaser, 7:21-22. Blaser further discloses that "captured data [for a layer] is held separate from the regular OS either locally or remotely in a data file, hard disk partition, or some other ***container***." Blaser, 9:31-33. Thus, Blaser itself recognizes that its layers are ***containers*** of software.

[0204] The '814 patent (at 2:23-42) provides a definition of "container" that is discussed below with labels [**1**]-[**6**] added. However, as I discussed above (§VI.A.2), the parties have agreed to a joint construction of the claimed *container* that is the combination of part [**1**] and the first sentence of part

[**3**] that I discuss below. *See* EX1106, 6.[8] For completeness, I demonstrate how Blaser-Calder meets the patent's full definition of "*container*" (i.e., parts [**1**]-[**6**]) in light of both parties' constructions, and thus also meets the parties' joint construction of *container* (i.e., part [**1**] and the first sentence of part [**3**] below).

> ***[1]: "An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container."*** *EX1001, 2:23-25; see also EX1106, 6.*

[0205] In Blaser, each layer includes *a set of software applications* (e.g., accounting applications, secretarial applications, or engineering applications) and "the persistent **files** and configuration **required to operate [*execute*] an[y] application [*software*]**" in the layer. Blaser, 7:35-41, 4:27-29 ("application 200 is **running** [*executing*] on a layered computing system"). Based on Calder's teachings, as I discussed above (§VII.A.3), each Blaser-Calder layer includes modified "application binar[ies]," "libraries," and "configuration files" that are *required to successfully execute a set of software applications* in the layer *on a computing platform*, even if it uses a non-native OS. Calder, [0082], [0102].

---

[8] I note that the parties have each offered a competing construction of the related term "*secure containers of application software*" (*see supra* §VI.B.8), which I address below in §VII.A.4.a.iii.(1).c).

[0206] I note too that in its supplemental preliminary infringement contentions, VirtaMove identifies as a *container* that meets the claims "[a] Docker image [that] is built up from a series of **layers**," which "are stacked on top of each other." EX1079, 15-16. Thus, at least under VirtaMove's view of the claim scope, Blaser's embodiments that stack a series of layers on top of each (*see, e.g.*, Blaser, FIG. 2, 4:27-50) are not inconsistent with the claimed *containers*. (I note, however, that Blaser also discloses embodiments where "applications are stored **individually** in layers, [such that] interactions between application files may no longer occur…as each application 'sees' only it's [sic] own installed libraries, followed by libraries in the base operating system" (i.e., layers are not stacked on top of each other, but are separately layered on top of the Base OS). Blaser, 6:62-7:1.)

[0207] Thus, Blaser-Calder's layers meet part [**1**] of the '814 patent's "container" definition, which is included in the parties' joint construction of *container*. *See* EX1106, 6.

> ***[2]****: "A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers." EX1001, 2:25-29.*

[0208] Blaser's "application layer may be defined to be a **group of files in combination with any associated application configuration stored** to operating system files" (Blaser, 3:42-46), and thus each layer *is not a physical container but a* [software-based] *grouping of associated files*. Blaser discloses that each "layer may be ***stored*** to a network server and retrieved and ***stored*** as required to any computer" (Blaser, 12:35-44), where "the subject application is then installed to the destination system, but isolated in a layer" (Blaser, 10:37-42). Thus, each layer *may be stored in a plurality of different locations that is accessible to, and for execution on,* each destination *server*. *See also* Calder, [0075], [0078], [0081] (teaching that application packages – *grouping of associated files* – "may reside within the memory of a server" or "a dedicated preprocessing computer" which can be retrieved for *execution on servers*).

[0209] Thus, Blaser-Calder's layers meet **part** [**2**] of the '814 patent's "***container***" definition.

> ***[3]**: "**Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers**. The term 'within a container', used within this specification, is [sic] to mean 'associated with a container.'" EX1001, 2:29-34; see also EX1106, 6.*

[0210] Blaser "isolat[es] application files and configuration in a layer" "to provide a barrier between applications which may use conflicting configuration or library files." Blaser, 3:51-56. Applications are "stored individually in layers" so "interactions between application files may no longer occur." Blaser, 6:62-7:1 (noting that applications in layers have their "own installed libraries"). Each layer has its own "read-writable portion containing additions, deletions and modifications to the original installation," where "each read-writable portion [can be] protected from access by other users." Blaser, 7:48-58. Moreover, "layers may be provided **accessible only** to an individual user," where "a writable layer is provided for each user, providing data protection and isolation between users." Blaser, 7:59-67. Thus, Blaser discloses that one implementation option includes read/write files associated with one user's layer that cannot be shared with another

user's layer.[9] Another mechanism by which Blaser prevents *read/write files within a* layer (a *container* in Blaser-Calder) from *being shared with other* layers is by providing each layer its own unique root directory, as discussed below (§VII.A.ix). Blaser, 29:11-28.

[0211] Even if read/write files were sharable across layers, POSAs would have had motivation to prevent sharing to protect read/write files within Blaser-Calder layers from corruption/interference by another layer. For example, Calder discloses "protecting the contents of the application package 115 from user access" ([0087]) to beneficially prevent "improper[] modifying or accessing data" ([0088]). POSAs would have similarly wanted to prevent one Blaser-Calder layer from being able to modify read/write files in (1) the Base OS of a server and (2) in another layer so that each layer could exist on its own and not damage the Base OS and not damage and/or be damaged by operations performed in another layer. (For example, to prevent an engineering layer from modifying the local system files on of the Base OS and to prevent the engineering layer from potentially corrupting an accounting layer and vice versa.)

[9] The parties agree that "within a container" in the joint construction of "*container*" means "associated with a container." EX1106, 6; EX1001, 2:29-34.

[0212] A POSA's motivation (and reasonable expectation of success in doing so) would have been reinforced by their background knowledge that techniques for preventing read/write files in one logical area of a computer system storage from modifying read/write files in another logical area of computer system storage were well-known and conventional. An example of which is the well-known chroot (change-root) functionality in Unix-based OSs (and similar functionality for other OS families), which existed precisely to provide the benefit of protect one user of a computer system (or one application / set of applications) from potentially corrupting the system files for another user / application. Because these features were well-known in the prior art, POSAs would have had a reasonable expectation of success in incorporating into Blaser-Calder using only ordinary skill because Blaser-Calder used conventional OSs that were already capable of supporting that functionality.

[0213] My opinion that POSAs would have wanted to provide different layers different views of a file system and that functionality like chroot were well-known in the prior art and used to prevent read/write files in one directory / storage area from being access from application / users assigned to a different directory / storage area is corroborated by EX1027. EX1027 discloses:

> As used herein, "chroot" refers to a command for changing the active root directory. That is, chroot is a command that executes to change a user's active root directory. For example, in a typical Unix system, the

root directory is indicated as "/". Thus, the command "cd/"[10] will normally execute to change to the root directory of the filesystem. The chroot command may be executed to change a user's active root directory. For instance, the command "chroot/home" **changes a user's active root directory to "/home"**. Thereafter, executing the command "cd/" results in changing to the "/home" directory rather than "/", as "/home" is now the user's active root directory. As a further example, the chroot command may be utilized to specify a particular compartment as the active root directory. For instance, the command "chroot/compt/FOO" changes the active root directory to compartment FOO. Accordingly, as the "cd/" executes to change to the active root directory (which may be changed by the chroot command)**, the chroot command may be utilized as a further security mechanism. For instance, the chroot command may be utilized to effectively prevent a user (or a process) from escaping the active root directory specified by the chroot command.** That is, for example, **a user (or process) may not escape the active root directory specified by the chroot command** (e.g., compartment FOO) to the actual root directory ("/") of the system. Once the active directory of a user (or process) has been changed via the chroot command, it is said to have been chroot-ed. Accordingly, via kernel controls, **the ability of a process to transition to root from within a compartment is removed** in this

[10] I note that, conventionally, there should be a space between "cd" and "/" in the UNIX system, but POSAs would have no problem understanding what EX1027 is disclosing.

exemplary implementation so that **the chroot cannot be escaped**. **This exemplary implementation provides the ability to make at least selected files within chroot immutable.**

EX1027, [0042].

[0214] My opinion that POSAs would have wanted to provide different layers different views of a file system and that functionality like chroot were well-known in the prior art and used to prevent read/write files in one directory / storage area from being access from application / users assigned to a different directory / storage area is corroborated by EX1035. EX1035 discloses:

> Many file systems present a global view of data accessible to all processes running on a machine with access to those file systems. **There are situations, however, where it is advantageous to provide different processes with different views of a file system**. **Machines with multiple network interfaces, machines with multiple disparate network views, and machines running multi-level trusted systems where different users get access to different parts of the system, are all such situations**.
>
> **Conventional systems may implement differing file system views**, for example, by utilizing file and folder permissions or access control lists. For example, in a UNIX operating system**, the chroot ( ) system call limits a file system view to a certain subset of files. Other systems enable a user to move parts of a globally valid directory (e.g., folder) tree for a private view**. **Yet other systems enable a user**

to create a new directory tree with a private view of processes by mounting file systems over a network.

EX1035, [0002]-[0003].

[0215] My opinion that POSAs would have wanted to provide different layers different views of a file system and that functionality like chroot were well-known in the prior art and used to prevent read/write files in one directory / storage area from being access from application / users assigned to a different directory / storage area is corroborated by EX1072. EX1072 discloses:

Chroot Environment

In one embodiment, the local machine is running the **Unix™ operating system**. **A user interacts with the system through an interface running from a chroot (or change root) directory. All actions initiated by the interface also run from the chroot directory. The chroot directory (or environment) is an alternate root directory that appears as the root directory for a given process or set of processes. Thus, a malicious user is unable to tamper with the local machine's system files in the main directory because the user is confined to the directory tree below the chroot directory (i.e., the user only has access to files stored in the directory tree below the chroot directory).**

FIG. 6 illustrates the process of user interaction with a computer system through a local machine running a Unix operating system in accordance with one embodiment of the present invention. At block 600, a user

requests access to the computer system at the local machine. At block 610, the chroot environment (or directory) is made available to the user. **The chroot environment is a restricted run-time environment; processes initiated in the chroot environment and all processes spawned thereby do not have access to anything outside the chroot environment**.

At block 620, a user context is dynamically created at the local machine for the user. At block 630, a user interface is initiated in the chroot environment. **Since the user interface is initiated in the chroot environment, the user will not be able to tamper with the local machine's system files, which are maintained outside the chroot environment**. However, the system files contained in the chroot environment are used by the user and remain at risk for user tampering.

In one embodiment, all users of a local machine share a single chroot environment. **Protection within the chroot environment is provided (only) by the standard Unix user semantics. Use of the chroot environment provides greater protection for the system resources outside of the chroot environment**.

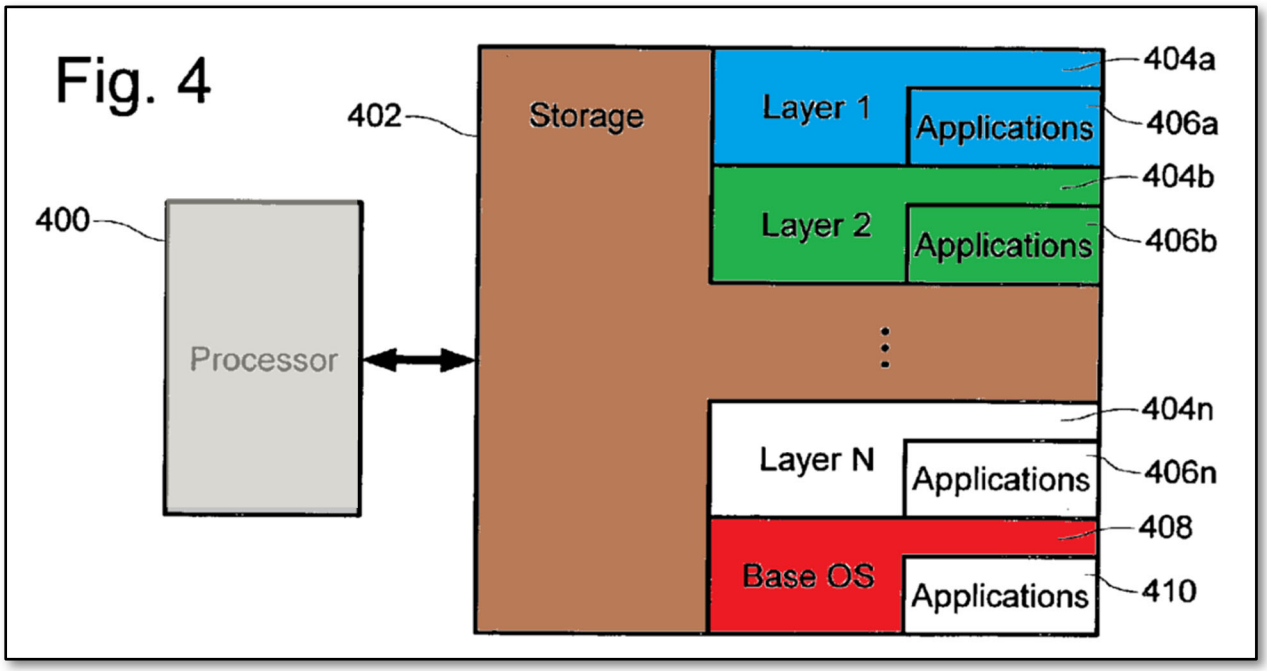
EX1072, [0049]-[0053].

[0216] Thus, Blaser-Calder's layers meet part [**3**] of the '814 patent's "container" definition, and therefore meets part [**2**] of the parties' joint construction of the claimed *container*. *See also supra* (where I demonstrate that

Blaser-Calder meets part [**1**] of the '814 patent's definition of "container," which includes part [**1**] of the parties' joint construction of the claimed "*container*").

> ***[4]****: "A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel[.]" EX1001, 2:34-37.*

[0217] As illustrated in Blaser's Figure 4 (which I have colored below), "*applications* [are] stored in a layer." Blaser, 4:51-59, 5:26-32.



See also Blaser, 4:58-59 ("**A number of layers 404a-n** are also contained on storage 402, **each having applications 406a-n**"). For example, different layers for different services / types of users include respective applications:

> Multiple layers can advantageously be used. In one layer hierarchy, one layer represents a user type and a writable layer contains the user's changes. For example, a company has 500 computers. These computers all have the same base software installed, which may be only the OS. The company then has layers defined for different types of users, for example secretaries, engineers and accountants. The **secretarial layer contains** word processing, spreadsheet, and other **secretarial applications**. The **engineering layer contains** software development tools, CAD tools, and other **engineering related applications**. The **accounting layer includes accounting software**. In addition, each user may have a personal layer, which contains an individual's changes on top of the type layer and base system.

Blaser, 13:2-15.

[0218] Based on Blaser's teachings, each Blaser-Calder layer stores the associated "files and directory structure of the application's installation," including system files like "[s]hared libraries (such as DLLs), [and] system accessible configuration (such as registry entries)." Blaser, 5:18-26.

[0219] Similarly, based on Calder's teachings, each Blaser-Calder layer comprises *application programs* and *associated system files* (e.g., libraries **and** configuration files) *for use in executing* the applications on native and non-native OSs, as I discussed above (§VII.A.3). *See* Calder, [0082]-[0083], [0101].

[0220] When an application in a layer "executes, it results in [a] **process** [] running." Blaser, 15:27-45. Calder similarly discloses that when

applications execute there are "**requests for process creation** and termination." Calder, [0122]-[0131].[11] *See also* EX1001, 16:11-15 ("programs/processes").

[11] I note that Calder at [0106] through [0131] refers to "a virtual machine (VM) communication thread," which is "used to provide a communications conduit between the application to the application manage [] and to control the operation of [sic] application." Calder, [0106]. POSAs would understand that Calder's use of the term "virtual machine (VM)" in this context is **not** referring to the prior art virtual machine techniques where multiple, independent OSs are installed on one machine, each VM's OS having its own kernel. A POSA would understand that the point of Calder's technique is that without a non-native kernel in an application package or installed to a non-native machine the application package can be made to allow the application to execute on non-native OSs. As I discussed above (§VII.A.2), Calder does that by translating native OS's system calls to the non-native OS's system calls, thereby permitting the application to execute on a host server using the server's non-native OS and non-native kernel, without having to rely on the native OS's system calls being interpreted by a native OS kernel. *See also, e.g.*, EX1121, 4 (noting that even a conventional, prior art single-kernel OS can sometimes be said to be "an implementor of virtual computers," by which a

[0221] POSAs also knew as part of their background knowledge that "[t]ypically,…a computer application is instantiated…in the form of one or more application processes…." EX1061, 2:28-33. Similarly, EX1064 corroborates that POSAs knew that "[w]hen a user requests a program to be executed, a new process is created to encompass its execution." EX1064, 2:12-13.

[0222] As Blaser's Figure 4 (which I annotated above) shows, layers are outside and "overlaid over the regular operating system," which is the Base OS and its kernel. Blaser, 5:18-59. The "base operating system [] forms a platform with which applications can be run and files can be accessed in file systems." Blaser, 4:7-26. Blaser's layers contain "a set of file system and registry changes" for use in executing the application's processes. Blaser, 4:60-5:2. Blaser discloses that when layers are "captured" (i.e., when layers are being created for future use), "changes made by the installation procedure do not affect the base system" and "captured data is held separate from the regular OS." Blaser, 9:5-34. Subsequent installation of layers result in changes being recorded to the layer but "not to the underlying file systems and OS resources." Blaser, 12:8-20.

POSA may "mean" the OS is "an implementation of the functionality of a computer in software").

[0223] Thus, layers do not comprise and cannot affect the Base OS's kernel.

[0224] I note that the fact that users in Blaser-Calder can make registry changes in not inconsistent with my opinion that Blaser-Calder layers do not comprise the Base OS's kernel, because the registry is not part of the Windows kernel. This is further corroborated by EX1024, which discloses that (as POSAs knew) users can "access[] and edit[]" the Windows registry in a prior art Windows-based OS. EX1024, [0024]-[0025]. By allowing each layer to edit a registry, POSAs understood that that editable part of a registry file exists outside of the shared kernel (otherwise a registry change in one layer would affect another layer, which would be contrary to Blaser's disclosure that "changes are recorded to a layer and **not to the underlying file systems and OS resources**" and even if user "pass[]" "viruses to the computers" it only affects the layer, not the computer or any other layers, Blaser, 12:8-20).

[0225] Moreover, in Blaser, "each layer is a separate and individual entity within **the** host OS" (Blaser, 5:18-32), which reinforces that each computing device has only one ("the") OS and associated kernel.

[0226] Furthermore, in Blaser-Calder, the layer's system files are modified so the application(s) in the layer can "execute and communicate on any non-native platforms." Calder, [0101]. For example, "system libraries are

added…that translate Windows 2000 system calls to Linux system calls." Calder, [0101]. "[W]hen a program runs under the Windows operating system," for example, "it accesses **the** operating system via the Windows API," with "[o]nly the WIN32 API calls [] allowed to access **the** operating system." Calder, [0096]. POSAs knew that the WIN32 API is how a 32-bit Windows's kernel is accessed. *See* EX1048, 6:7-8 ("**Win32 kernel**")). Thus, rather than *comprise a kernel*, Blaser-Calder's layers would not have (and would not require) a kernel because, based on Calder's teachings, the applications in a layer can execute on different OSs by translating (modifying) system calls to be understandable by the Base OS's kernel.

[0227] Thus, Blaser-Calder's layers meet part [**4**] of the '814 patent's "container" definition.

> *[**5**]: "[E]ach container has its own execution file associated therewith for starting one or more applications." EX1001, 2:38-39.*

[0228] Blaser discloses a "FSLLIB32.DLL runtime library," which "provides an API [application program interface] that may be used by other applications to manage the layered system," "includ[ing] functions to" "**activate** [i.e., execute] and deactivate layers" and "enable and disable layers." Blaser, 20:19-38.

[0229] Blaser discloses that the function "FSLActivate (PTCHAR fslName)" "[v]alidates the fslName against defined layers" – which are the layers available in the system – and "[c]ommunicates with FSLX driver…to activate the layer" or group of layers identified by "fslNAME." *See* Blaser, col. 20 page 17 (table's first function); *see also* ("FSLAddLayerToGroup (PTCHAR fslName, PTCHAR, groupName": "Verifies that both the specified layer [fslName] and group [groupName] are defined.").

[0230] "Applications in the layer [fslName] that are specified to be run on system startup…are started." *See* Blaser, col. 20, page 17 (table's first listed function).

[0231] Thus, FSLActivate is an execution file that starts fslName (a layer/group of layers available to the system), which, in turn, executes execution file(s) for starting the startup application(s) within the identified layer(s).

[0232] POSAs also had reason for FSLActivate to be executable using an execution file linked to an object, like an icon, to provide a user-friendly way to manage (e.g., install, activate, etc.) layers. For example, the engineering layer would be executed via an engineering-layer icon. POSAs would have reasonably expected success because Blaser's "API [] may be used by other applications to manage the layered system" (Blaser, 20:19-38) and thus it would have required only ordinary skill to use the API (application program interface) to associate an

icon object with a layer. Moreover, it was conventional for execution files to be associated with executable objects like icons, as I discussed above with support from corroborating evidence (*supra* §VII.A.4.a.ii.(4)).

[0233] Thus, Blaser-Calder's layers meet **part** [**5**] of the '814 patent's "container" definition.

> *[**6**]: "In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications." EX1001, 2:39-42.*

[0234] In Blaser, the "base operating system 110 forms a platform with which applications can be run [i.e., execute]" in layers. Blaser, 4:7-10, FIGs. 1-4. When applications request access to the Base OS, "the layering system software" "determine[s] whether the accesses should be permitted to continue to the base operating system []." Blaser, 4:10-26. As I discussed above (§VII.A.3), the conventional prior art OSs that would have been resident on Blaser-Calder's servers (e.g., Windows-based, Unix-based, etc.) each have a kernel that is utilized to execute applications to obtain resources/services that are needed to execute applications (e.g., open files, use the server's hardware), which are accessed via the Base OS's kernel. *See* Calder, [0150] (applications can "execute an interrupt call on the operating system kernel of a client computer").

[0235] My opinion that conventional system calls are used by applications to perform core functionality that are part of the OS kernel is corroborated by EX1097, which discloses:

> System Call—**Application programs interact with operating systems by System Calls**. A System Call occurs when **an application program invokes a function that is implemented by its operating system, for example, open or read a file or get the current time**.

EX1097, [0051]. The above-discussed functionality are examples of functionality provided by the OS's kernel. *See* EX1064, 1:53-62 (noting that in a "traditional operating system" the "kernel" "provides the user programs with a set of abstract system services called system calls"). Thus, EX1097 corroborates my opinion that applications within layers making conventional system calls from their respective layers is an example of *utilizing a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications*.

[0236] Thus, Blaser-Calder's layers meet part [**6**] of the '814 patent's "container" definition.

c) Blaser-Calder's layers are "secure containers of application software"

[0237] Google's construction of "*secure containers of application software*" (EX1106, 5) is consistent with the '814 patent's definition of "*secure* application *container*": "environments where each application set appears to have

individual control of some critical system resources and/**or** where data within each application set is insulated from effects of other application sets." *See also* EX1001, 2:43-48; 7:38-40. VirtaMove's construction is similar to Google's construction but replaces "an environment" with "containers." EX1106, 5. The '814 patent further discloses that an "application set" is "a set of one or more software applications that support a specific service." EX1001, 7:39-41.

[0238] Thus, a *container* is a *secure container of application software* under the parties' construction if it either (or both) (i) is an environment/container where each application set appears to have individual control of some critical system resources or (ii) data within each application set is insulated from effects of other application sets.

[0239] As I discuss below, Blaser-Calder's layers meet (i) and (ii) above. Below I start by demonstrating how Blaser-Calder's layers meet prong (ii) of each party's construction of *secure containers of application software* before turning to prong (i).

[0240] "**data within each application set is insulated from effects of other application sets**": As I discussed above (§VII.A.1), Blaser discloses an example of a "secretarial layer contain[ing]…secretarial applications." Blaser, 13:1-15. Those applications are an "application set" because they are "a set of one or more software applications that support a specific service" (*e.g.*, a secretarial

service). EX1001, 7:39-41. Similar application sets exist in other layers for other services (e.g., engineering layer comprising engineering applications for an engineer service). Blaser, 13:1-15.

[0241] As I discussed above (§VII.4.a.iii.(1).b), "container" definition part [**3**]), each layer's application set is insulated from effects of application sets in other layers because applications are "**isolated** in a layer" and "**protected from corruption** from other applications or meddling." Blaser, 10:25-42. For example, "application layers…**are isolated** from other applications on [the] computer," which "provide[s] a **barrier** between applications." Blaser, 3:42-58. When "applications are stored individually in layers, interactions between application files may no longer occur…." Blaser, 6:62-7:23. Blaser's layering technique thus provides "***Secure* Applications**" and "[a]pplications can be **protected** from unauthorized access through the…layered system." Blaser, 11:20-38. That Blaser-Calder's layers are ***secure*** is reinforced by Calder's teaching to modify application packages to prevent corruption from other applications. Calder, [0079], [0087], [0100]. *See also supra* §VII.A.4.a.ii.(3) (demonstrating that Blaser-Calder's layers are *secure environments*).

[0242] Thus, application sets in Blaser-Calder's layers (and the application set it includes) are executed in environments that are "insulated from effects of other application sets" and thus Calder-Blaser's layers meet each party's

construction of "*secure containers of application software*." EX1106, 5; *see also* EX1001, 2:43-48.

[0243] "**each application set appears to have individual control of some critical system resources**": As I explain below, Blaser-Calder layers also meet prong (i) of each party's construction of a "*secure containers of application software*" because, as I explain below, the layers would have been "environments where each application set appears to have individual control of some critical system resources." EX1106, 5.

[0244] Exemplary "resource[s]" in the '814 patent "include CPU time, memory, file activity and network bandwidth." EX1001, 10:15-17. Blaser says that because changes made by user applications in a layer are only "recorded to a layer and not to the underlying file systems and OS resources" users can "**use the computer in an unrestricted fashion**." Blaser, 12:8-20. Layers (and the application executing therein) appear to be "unrestricted" within the layers, which is an example of the layers *appearing to have control of some critical system resources* (like the memory and hard disk that they are accessing). However, that access to *critical system resources* is only an appearance of control over those resources because when a layer is deleted, any changes made within/by a layer are "wiped and the computer is restored to its base state. Blaser, 12:8-20. In other

words, the changes recorded within the layer do not have actual control over critical system resources, just an appearance of control of those resources.

[0245] Similarly, as I discussed above (§VII.A.4.a.iii.(1).b) and further below (§VII.A.4.a.ix), layers cannot access the host server's root directory but, because of the changed root ("chroot") functionality, appear to have control over the entire file system. Blaser, 12:8-20; Calder, [0138], [0241]. The chroot-ing of each layer's access to the file system reinforces that layers *appear to have control of* the entire file system (a *critical system resource*) but lack actual control over that *critical system resource*.

[0246] Moreover, as I discussed above (§VII.A.4.a.iii.(1).b)) with corroboration like EX1072, [0049]-[0053], POSAs would have had reason to use in Blaser-Calder prior art techniques to prevent one layer from accessing and/or editing the Base OS and critical system resources therein.

[0247] Based on Blaser's teachings, Blaser-Calder's layers (which include those Blaser teachings) also "**intercept key file system and registry calls** and manipulate the results **to create the appearance** that virtual files and registry setting contained in the layer definition exist in the real file system and real registry. Blaser, 14:60-15:3.

[0248] The key file system and registry calls – which a Blaser-Calder layer appears to have control of – are and/or manage critical system resources. My

opinion that files like the registry files are a critical system resource (or are at least used to access a critical system resource) is corroborated by EX1026, which discloses that an example of "non-file system resources" include the "registry." EX1026, [0004].

[0249] Thus, each Blaser-Calder layer appears to have individual control of some critical system resources, which is another way Blaser-Calder layers meet each party's construction of "*secure containers of application software*." EX1106, 5.

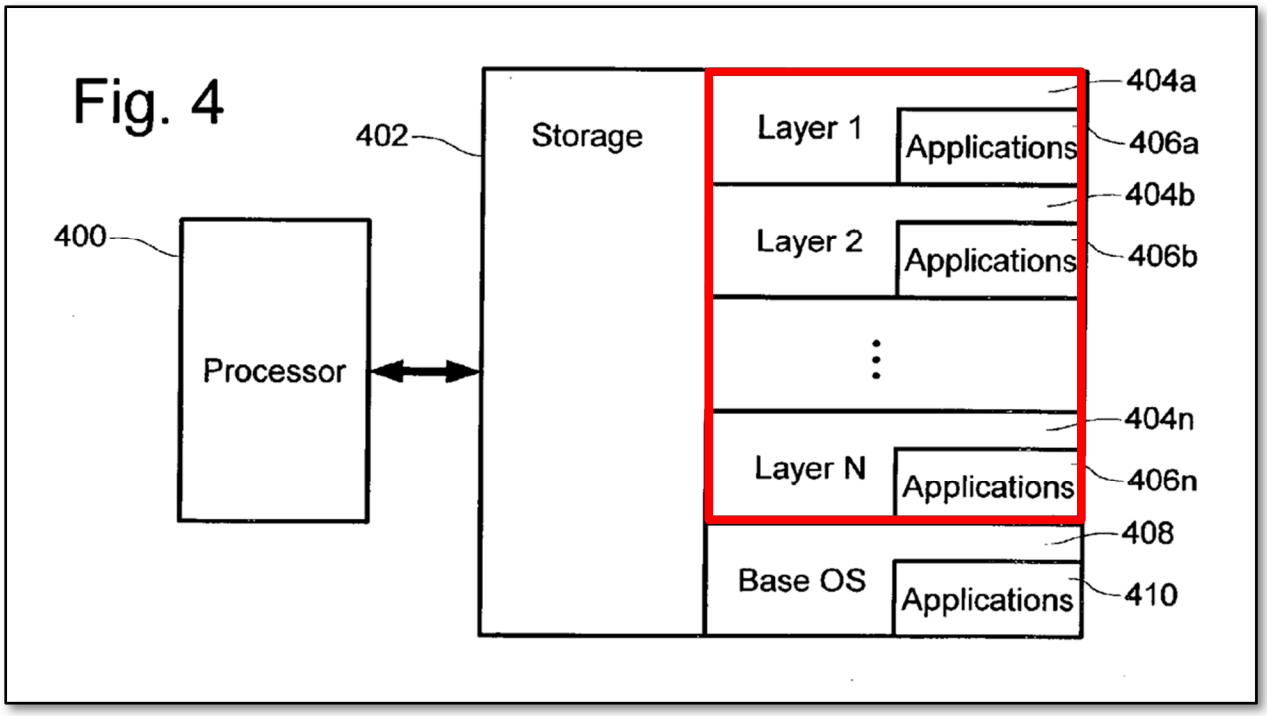
[0250] Additional, as I discussed above (§VII.A.3), Blaser-Calder would have used Calder's technique to modify application packages to be executable on both native- and non-native OSs. In doing so, as Calder explains, Calder's technique would have made an application's "**resource** requests virtualized" to give the **appearance** of having the ability to "**control**[] **resources**," including "memory usage," "network bandwidth," "and disk usage. *See also* Calder, [0126]. Thus, the resources that actually would have been used by the application in a Blaser-Calder layer would have been "transparen[t]" to and "hidden from the application." Calder, [0130].

[0251] That is another reason why each Blaser-Calder layer (and the application(s) executing therein) would have appeared to have individual control of some critical system resources, and thus is another by which Blaser-Calder's layers

meet each party's construction of "*secure containers of application software*." EX1106, 5.

(2) "each container comprising one or more of the executable applications"

[0252] As I discussed above (§VII.A.4.a.iii.(1).b)), *container* definition part [**4**]), Blaser-Calder's layers "each hav[e] applications" (Blaser, 4:51-59), which Blaser says are "running" (i.e., are *executable*) in the layer (Blaser, 4:27-29). *See also* Blaser, 5:18-32 ("application stored in a layer"), 15:6-36 ("For example, suppose layer A contained B.EXE [i.e., an executable file for an executable application]. When B.EXE ***executes***, it results in process C running."). This is illustrated, for example, in Blaser's Figure 4 (which I have annotated below), which shows various applications (406a-406n) within corresponding layers (404a-404n):



For example, "[t]he **secretarial layer contains** word processing, spreadsheet, and other **secretarial applications**" and "[t]he **engineering layer contains** software development tools, CAD tools, and other **engineering related applications**," etc. Blaser, 13:1-15.

[0253] As I noted above, Blaser discloses that when an application in a layer "***executes***, it results in [a] process [] running." Blaser, 15:6-36. Calder similarly discloses that when applications *execute* there are "**requests for process creating** and termination." Calder, [0122]-[0131]. *See also* EX1001, 16:11-15 ("programs/processes"). POSAs knew as part of their background knowledge that "[t]ypically,…a computer ***application* is instantiated** [i.e., starts *executing*]…in the form of one or more ***application* processes**…." EX1061, 1:13-16. Similarly,

EX1064 corroborates that POSAs knew that "**[w]hen a user requests a program to be executed, a new process is created** to encompass its execution." EX1064, 2:12-13. Thus, by Blaser and Calder disclosing that there are processes running that correspond to applications it reinforces to POSAs that the applications in the Blaser-Calder layers (*containers*) would have been *executable applications*.

[0254] Each layer also stores the associated "files and directory structure of the application's installation," including *system files* like "[s]hared libraries (such as DLLs), [**and**] system accessible configuration (such as registry entries)." Blaser, 5:18-32.

[0255] Thus, each Blaser-Calder layer would have comprised *application programs* and *associated system files* (e.g., libraries **and** configuration files) *for use in executing* the applications.

[0256] Additionally, as I discussed above (§VII.A.3), based on Calder's teachings, each Blaser-Calder layer would have comprised *application programs* and additional *associated system files* (e.g., modified libraries **and** modified configuration files) *for use in executing the applications* on both native and non-native OSs. *See* Calder, [0082]-[0083], [0101].

[0257] Thus, *each* Blaser-Calder layer (*container*) *comprises one or more of the executable applications*, as claimed.

(3) "and a set of associated system files required to execute the one or more applications"

[0258] The '814 patent defines "system files" as "files provided within an operating system and which are available to applications as **shared libraries and configuration files**." EX1001, 2:52-54. Thus, the '814 patent recognizes (as POSAs knew) that conventional OSs in the prior art provided (or at least typically provided) *system files*.

[0259] In Blaser, each "layer may be defined to be a set of file system and registry changes" (Blaser, 4:60-5:2), which includes "the ***files*** and directory structure of the ***application's*** installation" (Blaser, 5:19-32). These "***files*** and directories" are "shadowed or overlaid over the regular operating system file system." Blaser, 5:19-32. The files copied into layers include "**shared libraries (DLLs)**" (Blaser, 6:62-7:23) **and** "persistent ***files* and configuration *required to* operate *an application***" (Blaser, 7:36-47) **and** copies of "registry changes" (Blaser, 4:60-5:2). *See supra* §VII.A.4.a.i.(4) (where I demonstrate with corroboration that registry changes would conventionally be made to registry files, which are configuration files and thus are examples of *system files* under the patent's definition of *system files*).

[0260] Thus, based on Blaser's teachings, Blaser-Calder's layers comprise *system files* like shared libraries **and** configuration files, and thus meet the '814 patent's definition of "*system files*." *See* EX1001, 2:55-3:19 (exemplary

system files include libraries and configuration files); Blaser, 9:18-34 (layers "capture" "all of the application[']s shared DLLs, registry entries, and .ini files"), 10:50-59 ("software installation of an application layer" is recorded to the layer's "readable-only portion" so the user cannot "damage the application installation"). These system files (shared libraries, like DLLs, and configuration files, like registry files) that would have been in each Blaser-Calder layer are *a set of associated system files required to execute the one or more applications* in the layer, as claimed.

[0261] Moreover, as I discussed above (§VII.A.3), based on Calder's teachings, Blaser-Calder's layers also include modified *system files* (e.g., libraries and configuration files) that are *required to execute the one or more applications* in the layer on native and non-native OS. Calder, [0077], [0082], [0094], [0101]. For example, each application in a Blaser-Calder layer would have had "system libraries" that would have been "added…that translate Windows 2000 system calls to Linux system calls." Calder, [0101].

[0262] POSAs also knew as part of their background knowledge that when executing applications in a conventional OS (like those that would have been used in Blaser-Calder's system) the OS would have looked for, e.g., a configuration file (a *system file*) associated with the application that is required to

execute the application. My opinion in that regard is corroborated by EX1061, which states:

> Typically, a computer system/computer has an operating system, and **a computer application is instantiated** on the computer **in the form of one or more application processes** running in a work space managed by the operating system.

EX1061, 1:13-16.

> Typically, **when an operator commands the operating system 14 of a computer 12 to instantiate an application 16,** such operating system 14 performs a number of procedures. In one of such procedures, **the operating system 14 locates a configuration file 22 for the application** 16 in the memory 13, and **then processes the located configuration file 22**. In particular, the configuration file includes a list of the application processes 18 that are to be run as part of instantiating the application 16.

EX1061, 2:53-58. Thus, a POSA's background knowledge further corroborates that when an application executed in a Blaser-Calder layer the OS would have looked for the application's configuration file (an example of a *system file*) in the layer, in order to execute the application according to the configuration file (e.g., use the application's preferred, or user-configured preferred, font types and sizes for the application).

(4) "for use with a local kernel residing permanently on one of the servers"

[0263] As I discussed above (§VI.B.7), Google construes *local kernel residing permanently on one of the servers* as "local kernel in one of the server's memory that is not lost when power is removed from it" (EX1106, 4); VirtaMove argues that "[n]o construction is necessary" and that the "plain and ordinary meaning" should apply (EX1106, 4).

[0264] In Blaser, the "base operating system [] forms a platform with which applications can be run [i.e., execute]" in layers. Blaser, 4:7-27, FIGs. 1-4. When applications request access to the Base OS, "the layering system software" "determine[s] whether the accesses should be permitted to continue to the base operating system [], or should be redirected" to the file system within the layer. Blaser, 4:7-27. As I discussed above (§VII.A.3), the conventional prior art OSs that would have been *resident on* Blaser-Calder's *servers* (e.g., Windows-based, Unix-based, etc.) each have or had reason to have a *local kernel* that is exclusively *used* to obtain resources/services provided by the kernel, where those resources are resources that are needed *to* execute applications (*e.g.*, open files, use the underlying hardware). *See* Calder, [0150] (corroborating that POSAs knew applications can "execute an interrupt call on the operating system kernel of a client computer").

[0265] My opinion that one conventional use of conventional system calls used (e.g., by applications) is to request functionality provided by the OS's kernel is corroborated by EX1097. EX1097 discloses the following regarding a "System Call," and describes various functionality conventionally provided by an OS kernel (e.g., opening or reading a file):

> System Call—**Application programs interact with operating systems by System Calls**. A System Call occurs when **an application program invokes a function that is implemented by its operating system, for example, open or read a file or get the current time**.

EX1097, [0051]. *See also supra* §VII.A.4.a.iii.(1).b) (container definition part [**6**]).

[0266] POSAs understood that each Blaser-Calder's server's *local kernel resides permanently* in memory of *the server* (Blaser, 4:51-59, FIG. 4) (or had reason to reside permanently in the server) because the kernel is not lost (or conventionally would not be lost) when power is removed from the server, and thus is (or conventionally would have been) stored in non-volatile memory (e.g., ROM, a hard disk, etc.), which meets Google's construction of a "*local kernel residing permanently on one of the servers*." EX1106, 4; EX1102, 12.

[0267] My opinion that Blaser-Calder's local kernel would not have been lost (or had reason not to be lost) when power is removed from the

corresponding server is corroborated by EX1091. EX1091 discloses as background information:

> In general, the ability to store and access data is critical for computers. For example, **when turned on**, a computer (e.g., a personal computer ("PC")) accesses, and **prepares (or "boots") the operating system from its local persistent storage device (e.g., "hard disk"**). Once the booting is finished, the contents of the hard disk are accessible and available to the user. The contents of the hard disk (also referred to as the hard disk's "disk image" or "data image") define the user's personalized environment: **the operating system (such as Windows 98 SR-2, Linux, etc.)**, the software applications (such as word processors, spreadsheet programs, web browsers, etc.), the data files (such as documents, spreadsheets, images, or cookies), and any additional customization (such as whether a particular web browser, such as Netscape Navigator or Internet Explorer, is automatically launched when an HTML file is accessed).

EX1091, 1:23-38. Thus, EX1091 corroborates that in a conventional computing device, the OS (and its associated local kernel) are stored in a persistent storage device (i.e., a device whose contents are not lost when power is removed from the server) so that it can be loaded whenever the computer is turned on from a powered-off state.

[0268] To the extent VirtaMove's "plain and ordinary meaning" construction of *local kernel residing permanently on one of the servers* (EX1106,

4) includes (at least) at least a kernel that is not lost when power is removed from the server, then Blaser-Calder also meets this limitation under VirtaMove's construction. *See also* EX1103, 13. Moreover, as I discussed above, the Blaser-Calder layers would have used conventional *local kernels* for conventional OSs (e.g., Windows- and Unix-based OSs), which are the same types of conventional OSs discussed in the '814 patent (e.g., EX1001, 7:33-35). Thus, to the extent VirtaMove's construction includes the same conventional OSs and *local kernels* that the patent discloses, that is another reason why Blaser-Calder meets this limitation under VirtaMove's construction.

[0269] Thus, Blaser-Calder's layers are *for use with a local kernel residing permanently on one of the servers*, as claimed, under both parties' constructions.

iv. [1B] "wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,"

[0270] As I discussed above (§VI.B.5), Google construes *at least some of the plurality of different operating systems* as "at least two or more of the plurality of different operating systems" (EX1102, 11); VirtaMove argues that "[n]o construction is necessary" and that the "plain and ordinary meaning" applies (EX1103, 12).

[0271] As I discussed above (§VII.A.4.a.iii.(3), [**1A**]), each Blaser-Calder layer (*container*) comprises a *set of associated system files*, like shared libraries (e.g., DLLs), configuration files (e.g., registry files), **and** system files for execution on the plurality of different OSs in the system. Blaser, 4:60-5:2; Calder, [0077], [0082], [0094], [0101].

[0272] Because a server's Base OS (and its *local kernel*) is the "platform" for the layers installed to that server (Blaser, 4:7-27), each layer's *system files are compatible with a local kernel of* that server's Base OS.

[0273] As I discussed above (§VII.A.3; §VII.A.4.a.i.(1), [**1PREA**]), each Blaser-Calder's layer is compatible with at least two of *the plurality of different OSs* in the system; and thus applications in the layers would have been *compatible with* at least two of *the plurality of different OSs* (and their respective local kernel) in the system as well because otherwise the layers would not have been executable on those servers. Calder, [0101] (copied system files are modified so the application can also run on "any non-native [OS] platform" and thus utilize non-native kernels).

[0274] My opinion that in the conventional OSs that would have been used as the Base OS in the Blaser-Calder would have had *local kernel* is corroborated by EX1086, 2:53-55, which discloses that "the operating system is commonly called the system kernel [], or just the kernel." My opinion is further

corroborated by EX1064, which notes that in a "traditional operating system" the "kernel" "provides the user programs with a set of abstract system services called system calls." EX1064, 1:53-62.

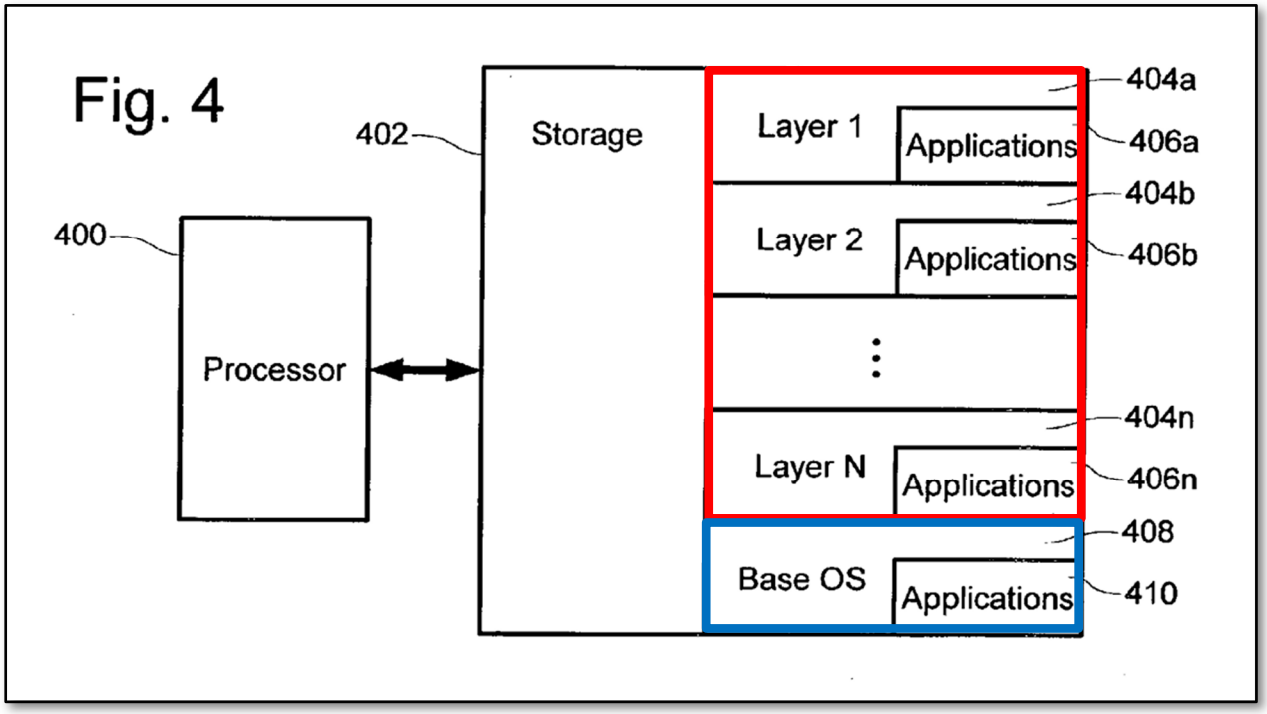
[0275] Thus, Blaser-Calder meets limitation [**1B**] under Google's construction of *at least some of the different operating systems*. EX1106, 4. To the extent VirtaMove's "plain and ordinary meaning" construction of *at least some of the plurality of different operating systems* (EX1106, 4) also includes at least two or more of the plurality of different operating systems in the system, then Blaser-Calder also meets this limitation under VirtaMove's construction.

v. [1C] "the containers of application software excluding a kernel,"

[0276] As discussed above (§VII.A.4.a.iii.(1).b)), [**1A**]) for the patent's "container" definition **part [4]**) and reiterate below, Blaser-Calder's layers (which are *the containers of application software*) *exclude a kernel*.

[0277] Blaser's layers "each hav[e] applications" and contain "a set of file system and registry changes" for use in executing the applications. Blaser, 4:51-63, 13:1-15 ("layer…contains…applications"). As Blaser's Figure 4 shows (which I have colored below), layers and their ***application software*** (which are surrounded by the red box) are outside and "overlaid on top of an operating system" (the Base OS and its *kernel*, which are surrounded by the blue box). Blaser, 2:50-58. On each server, the "base operating system [] forms a platform

with which ***applications*** can be run and files can be accessed in file systems." Blaser, 4:7-27.



[0278] When a layer is initially "captured" (i.e., when the layer is being created for future use), "changes made by the installation procedure **do not affect the base system**" because "captured data is **held separate from the regular OS**." Blaser, 9:5-34. After a layer is subsequently installed to that or another computer (e.g., server) in the system, changes are recorded to the installed layer but "**not to the** underlying file systems and **OS** resources." Blaser, 12:8-20.

[0279] Thus, Blaser-Calder's layers *exclude* and cannot affect the Base OS's kernel.

[0280] I note again that the fact that layers include a "set of file system and registry changes" (Blaser, 4:60-63) is not inconsistent with my opinion that POSAs understood layers exist outside the Base OS's kernel because, as I discussed above (§VII.A.4.a.iii.(1).b)), registry changes do not affect the Base OS and, generally, do not affect other layers (unless another layer is explicitly made a sub-layer of another layer, which is an optional configuration separate from and/or in addition to Blaser's disclosure of having completely independent layers).

[0281] Moreover, based on Blaser's teachings, "each layer is a separate and individual entity within **the** host OS" (Blaser, 5:18-32), reinforcing that each computing device (server in Blaser-Calder) has only one OS whose single local kernel is shared by all layers installed on the server device.

[0282] Furthermore, based on Calder's teachings, a Blaser-Calder layer, as I discussed above (§VII.A.3), would have also included "system libraries [that] are added…that translate Windows 2000 system calls to Linux system calls." Calder, [0101]. Thus, "when a program [would have] run[] under the Windows operating system," for example, "it [would have] accesse[d] the operating system via the Windows API," with "[o]nly the WIN32 API calls [] allowed to access the operating system." Calder, [0096]. POSAs knew that the WIN32 API is how applications access a 32-bit Windows's kernel (other conventional OSs had similar APIs or libraries for accessing the kernel). My opinion in that regard is

corroborated by EX1048, which refers to the "Win32 kernel." EX1048, 6:7-8. Thus, when implemented in an alternative conventional prior art OS, a corresponding API to a corresponding kernel would have been used. Because each layer must access the OS's kernel via the API (or some similar functionality for accessing a kernel), it reinforces to POSAs that the layer *excludes* the *kernel* (and must instead access the underlying OS kernel for access to the services and resources that the kernel provides / mediates).

[0283] Thus, Blaser-Calder meets limitation [**1C**].

vi. [1D] "wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,"

[0284] The '814 patent says "[t]he term '*within a container*'" "mean[s] 'associated with a container.'" *See* EX1001, 2:32-34. The parties agree that the term "within a container" means "associated with a container" when used in their joint construction of "*container*." *See* EX1106, 6.

[0285] As I discussed above (§VII.A.4.a.iii.(3), [**1A**]), the *associated system files within* (i.e., the *system files* associated with) each Blaser-Calder layer (*container*) would have been *stored in memory* and would have included "captures" (e.g., copies) of "all of the applications shared DLLs, registry entries, and .ini files" that are needed to execute the application(s) within the layer.

Blaser, 9:18-34. The "[a]pplication files that would be placed on file systems managed by the OS are also redirected into the layer." Blaser, 9:18-34. "All of the captured data is held [i.e., *stored*] separate from the regular OS either locally or remotely in a data file, hard disk partition, or some other container." Blaser, 9:18-34.

[0286] Similarly, based on Calder, Blaser-Calder's layers include modified system files – e.g., "libraries" and "configuration files" (Calder, [0082]) – that *are utilized* by each layer's application(s) to allow the applications to execute on both native **and** non-native OSs (e.g., the same application layer stored in memory can be installed to and execute on different OS families, like a Windows-**and** Linux-based OS).

[0287] As I discussed above (§§VII.A.4.a.iii.(3)-(4), [**1A**]), the Base OS *that remains resident on* a Blaser-Calder *server* comprises *associated local system files*, like shared libraries and configuration files, that are used, for example, to execute applications that are installed outside of (i.e., not within) a layer. *See* Blaser, FIG. 4, 3:42-58, 4:51-59 (applications 410 associated with Base OS 408 rather than a layer). But as I also discussed above (§VII.A.4.a.iii.(1).b)), §VII.A.4.a.iii.(3)), the *associated system files within* a layer are *utilized in place of* those *local system files* when an application in a layer is being executed (e.g., the

server will look for and execute the application based on a configuration file that is present in the layer and associated with the copy of the application in the layer).

[0288] Thus, *system files within* Blaser-Calder layers (containers) would have been *utilized in place of the associated local system files that* would have *remained resident on the server*, and Blaser-Calder thereby meets limitation [**1D**].

vii. [1E] "wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,"

[0289] Blaser's "layers may be stacked on top of each other, with **the real file system at the bottom of the stack**." Blaser, 8:6-18. The real file system at the bottom of the stack is the Base OS's file system, which includes the Base OS's *associated local system files that remain resident on the server*, which I discussed above (§VII.A.4.a.i.(4) and §VII.A.4.a.vi). Blaser further discloses that "files of the same name and location" can "exist in multiple layers, or in the base file system," and "rules can be provided whereby the layered system can determine which file to present to an application." Blaser, 8:6-18. POSAs understood that as disclosing that a layer's *system files are utilized in place of system files that remain resident on the server* in its base OS. For example, when a layer and the "real file system at the bottom of the stack" have system files with the same name and

location, the layer's system files are utilized (which is why files of the same name and location can exist in multiple layers and the Base OS).

[0290] The *system files* in each Blaser-Calder layer are created "through a 'capture' operation" that "intercept[s] operations" involving "files and configuration such as a registry," such that "changes made by the installation procedure **do not affect the base system** but are rather recorded to the new layer." Blaser, 9:5-17. "[A]ll operations by any application to **create**, ***modify*** or delete ***files*** are entered into [the] layer," so "all of the applications shared DLLs, registry entries, and .ini files [which are examples of system files] that would be directed to the Windows [local] system directories **become trapped in the capture layer**." Blaser, 9:18-34. Blaser also uses a "file system ***copy***" to create layers (Blaser, 9:64-67), which are *copies* of the *system files* that are within the file system that is used to create ("capture") a layer. Thus, Blaser teaches that the *system files* in each layer are *copies* and/or *modified copies* of the *associated local system files that remain resident on the server* (i.e., the *local system files* that are not affected when a layer is installed).

[0291] Furthermore, Blaser says "if applications are stored individually in layers" (e.g., each application has its own layer) "each application 'sees' only it's [sic] own installed libraries first, followed by libraries in the base operating system." Blaser, 6:62-7:1. Blaser further discloses that each layer has a "'read-

writable' portion[]…containing additions, deletions and ***modifications*** to the original installation" of "files and configuration as originally installed" that would be installed by an application when the application is installed to a computing device. Blaser, 7:48-58. Thus, the *local system files* within each layer (e.g., libraries **and** configuration files) would have included *copies* of *associated local system files* (e.g., *copies* of the same libraries and configuration that are typically included with a prior art OS and with application installation packages) **and**, once *copied* into a layer, Blaser discloses that the *system files* within the layer can be *modified*. For example, as I noted above with corroboration (§VII.A.3), application installation packages typically include copies or modified copies (e.g., newer versions) of system files that were often provided by the OS (e.g., shared libraries). Application packages often included and installed these system files in case the copy that was already present in the computing was corrupted and/or was an older copy that did not include new functionality that the application preferred and/or needed to use.

[0292] There would have also been some program installed to Blaser's Base OS **and** at least one (if not all) layers, and thus the *local system files* used by those programs would have been *copies* of each and each layer would have been permitted to *modify* those copies. For example, Blaser says that layers permit each server to "provide a multi-user environment providing a facility for an

administrator to designate layers accessible to individual layers" and "to automatically enable layers on user login and disable layers after a user has logged off." Blaser, 7:59-67. Thus, Blaser is teaching (or at least suggesting) that each computer has a *copy* of the server's login shell program (and thus the *system files* associated with the login shell).

[0293] Moreover, as I discussed above (§VII.A.3), based on Calder's teachings, Blaser-Calder's layers would have included *modified system files*, such as "***modified*** libraries 225, [**and**] ***modified*** configuration files 235" that would have "contain[ed] specific details about the type of system that the application package 115 is to be run on," in which information in these system files "may be completely reformatted to run on a specified operating system." Calder, [0082]. "When an application package 115 is run for the first time, or each time it starts to run, it ***copies*** specific information **from the existing system registry to the [application package's] virtual database**." Calder, [0121].

[0294] As I discussed above (§VII.A.2), Calder's "interception module" that is installed with an application package "prevent[s] a user of the client computer 140 from accessing the contents of the application package 115 and/or the application from improperly modifying or accessing data from the client computer." Calder, [0088]. This is because the interception module redirects

requests to use system files to the package's modified system files, rather than to the (unmodified) *local system files that remain resident on the server*.

[0295] Thus, in Blaser-Calder's system, the *associated system files utilized in place of the associated local system files* would have been *copies or modified copies of the associated local system files that* would have *remained resident on the server*, and Blaser-Calder meets limitation [**1E**].

viii. [1F] "and wherein the application software cannot be shared between the plurality of secure containers of application software,"

[0296] Blaser-Calder's "application layers" (containers) "**are isolated from other applications** on a computer" by "provid[ing] **a barrier between applications** which may use conflicting configuration or library files." Blaser, 3:42-58. Deleting layers does not disturb other layers (Blaser, 3:42-58), which reinforces that each layer's applications are not shared.

[0297] "[E]ach layer is **a separate and individual entity** within the host OS" (Blaser, 5:18-32) and applications in layers are "**protected from corruption** from other ***applications*** or meddling" (Blaser, 10:25-42). For example, for "***applications***…stored individually in layers, **interactions** between application files **may no longer occur** due to conflicting shared libraries (DLLs), as each ***application*** 'sees' only it's [sic] own installed libraries first, followed by libraries in the base operating system." Blaser, 6:62-7:23. "**A second *application***

installed to the computer…**cannot view the *application* layer files** because it does not possess the correct authentication key." Blaser, 11:39-58. That disclosure further reinforces that a layer's (*container's*) *application software cannot be shared between the plurality of secure containers of application software*, because otherwise there would be interactions between layers and deleting a layer would potentially disturb applications in other layers (e.g., if a read/write file in the deleted layer were being shared with an application in another layer that needed that file).

[0298] Moreover, as discussed further below (§VII.A.4.a.ix), each layer has its own, unique root directory to which applications are installed, which further teaches or at least suggests and motivates (and thus would have rendered obvious to POSAs) that applications cannot be shared across layers. POSAs also had reason to prevent sharing application software between layers to protect applications installed in layers. POSAs knew, for example, that the chroot functionality was often used to create a so-called "sandbox" environment, where applications in the sandbox have "access only to the constrained environment and **cannot corrupt software applications outside it**, i.e. beyond the sandbox boundary." EX1075, 2:9-15. As Calder, a "sandbox directory" technique will include "all of the files for the application packages." Calder, [0135]. As I discuss further below (§VII.A.4.a.ix), each sandbox directory has a unique "virtual" root

directory that appears to be the host's root directory, which teaches or at least suggests that applications cannot be shared across sandbox directories.

[0299] Thus, Blaser-Calder meets *the application software* within a layer (*container*) *cannot be shared between the plurality of secure containers of application software*, and meets limitation [**1F**].

ix. [1G] "and wherein each of the containers has a unique root file system that is different from an operating system's root file system."

[0300] In Blaser, each layer's file system is "overlaid on the normal file system." Blaser, 29:11-28. An "fslrdr directory" is created, and "[u]nder the fslrdr directory[,] **directories that correspond to *each* layer are maintained**." Blaser, 29:11-28. For example, if the layer named "TEST" has "the directory 'C:\fslrdr\TEST\c\XYZCorp,'" (i.e., the directory "XYZCorp") the layer's application(s) do not know that the directory "XYZCorp" is actually a subfolder of the file path "fslrdr\TEST\c" because "the directory 'c:\XYZCorp' **appears** on the C: drive to all applications running under that layer." Blaser, 29:11-28. That is, to the layer and its applications, the folder "XYZCorp" appears to be in the server's file system as C:\XYZCorp (i.e., it appears to be a subfolder of the *root file system* C:\), but in reality the location of the XYZCorp folder (and all of its sub-folders / sub-directories) in the real file system is under the layer's *different root file system* (C:\fsldr\Test\). (My opinion that the "C:\" is the *OS's root file system* is

corroborated by Calder at [0241], which refers to the "***root*** directory C:". Other conventional prior art OSs would have had corresponding *root* directories for the OS.)

[0301] Thus, based on Blaser, each layer in Blaser-Calder would have had a *unique root file system* (e.g., "C:\fslrdr\TEST") *that is different from* the server's *operating system's root file system* – the *root file system* under "c:\," meeting VirtaMove's interpretation of [1G]. That is because, in the (currently stayed) district court litigation against Google, VirtaMove has alleged that "[r]ealistically if a container on a particular server does not share a root file system with the OS *on that server*, it will not share a root file system with ***any*** operating system (in the entire world).") (all emphasis original).

[0302] As I discussed above (§VI.B.9), in the District Court litigation, Google explains that limitation [**1G**] makes the claim indefinite because limitation [**1G**] recites "*different from **an** operating system*" without indicating **which** OS the *container*'s *root file system* must be "*different from*." EX1102, 14. I understand that the Board does not need to resolve that dispute because, as I explain below, it would have been obvious to POSAs that some modifications users may make to their own layer's *unique root file system* would make that *file system different from* the *root file system* of **every** OS in the world, which would meet limitation [**1G**]

regardless of the uncertain boundaries of what OS (or set of OSs) the claim is referring to

[0303] Blaser, for example, says "[w]hile in a user session, **changes** are recorded to that user's layers," with "the state of the user's files and configuration being maintained between sessions." Blaser, 12:27-31. Blaser also says that "[a]pplications specific to a user might be **installed** to a user layer providing an easy way to control what applications are available to a particular." Blaser, 12:31-34. Thus, as users make changes to their layers (e.g., changing a configuration file, that configures, for example, the default fonts, font sizes, application startup order, icons, etc.) and/or users install applications to their layers (which typically would include installing various *system files*, like shared libraries, to the layer) users will be making changes to the layer;s local *file system* that, in turn, makes the layer's *root file system* a *unique file system* that is *different from* the *root file system* of every other OS's *root file system* in the world.

[0304] In addition, if Blaser does not permit users to define the name of their layers, POSAs would have had motivation to permit users to name their layers, which would have been a typical, conventional functionality for systems, like Blaser-Calder's system, that allows different users to login to servers that provide each user their own computing environment. For the same reason, it would have been obvious that each user would have given (or been required to

give) their layers *unique* names (or been automatically assigned *unique* names) that would have been different from the name of every other layer in the world (similar to how it was, and still is, conventional for people to user their email addresses – a globally unique identifier – as their login name for accessing remote servers, like websites). Doing that would have been another obvious implementation for the *root file system* of each layer – which would be based on the layer's name as I discussed above – that would have made each layer's *root file system* different from the *root file system* of every OS in the world.

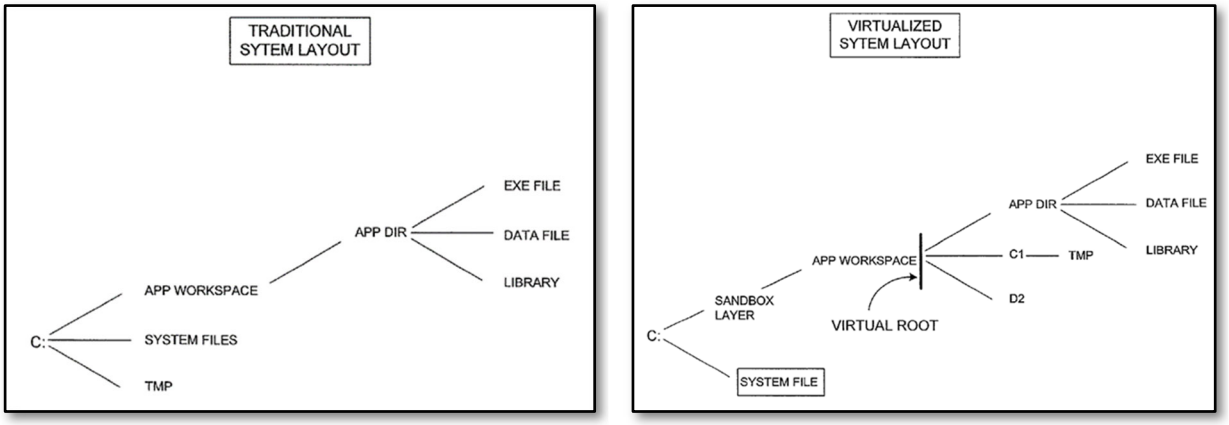
[0305] Additionally, in the district court litigation, VirtaMove alleges that a container assigned to a "**chroot**" file system – as in Blaser-Calder (*see* §VII.A.4.a.viii) – meets limitation [**1G**]. EX1079, 40-51 (emphasis original). For example, VirtaMove's contentions quote a document that says, in part, "[t]he original purpose of the cgroup, **chroot**, and namespace facilities in the kernel was to protect applications from noisy, nosey, and messy neighbors." EX1079, 40 (emphasis original). VirtaMove appears to have bolded "chroot" to emphasize that that is the functionality that VirtaMove alleges meets limitation [**1G**].

[0306] Thus, under VirtaMove's apparent claim scope in the district court litigation, each layer in Blaser-Calder would have *had a unique root file system* (e.g., "'C:\fslrdr\TEST\c") *that is different from an operating system's root file system* (i.e., the actual root file system that is designated as being under "c:\").

(My opinion that the "C:\" is the *OS's root file system* is corroborated by Calder at [0241], which refers to the "***root*** directory C:".)

[0307] Finally, to the extent not taught or rendered obvious by Blaser alone, POSAs would have also had motivation and a reasonable expectation of success in implementing each Blaser-Calder's layer's *file system* in a chroot environment, based on Calder's teachings discussed below.

[0308] Calder's Figure 46 (below left) shows a "traditional system layout"; Figure 47 (below right) shows a "virtualized system layout":



In the traditional layout (above left), an application can "access a DOS prompt for the root directory C:," and see "three folders located within the root directory": "APP WORKSPACE," "SYSTEM FILES," and "TMP." Calder, [0241]. In Calder's "virtualized system layout," inside the "sandbox directory" is "a **virtual *root* directory** [that] provides the directory structure." Calder, [0241]. When Calder-modified application "request[]" "to access[] the subdirectory 'C:\TMP,'

the interception module would rename the file to its corresponding location in the sandbox directory C:\SANDBOX_LAYER\AP[P]_WORKSPACE\ TMP," which is "done transparently to the application." Calder, [0241].

[0309] POSAs would have found it obvious based on Calder's teachings to sandbox Blaser-Calder's layers in the way Calder describes because Calder explains that its techniques for modifying and executing the modified applications provide a way to beneficially "safely" and "securely" "execute[]" the applications "on a remote machine." Calder, [0073].

[0310] Thus, POSA would have had motivation to implement Blaser-Calder's layers to use Calder's modified system calls to sandbox the applications in a Blaser-Calder layer using a chroot environment, which would have resulted in the application(s) in each layer thinking it would have access to the OS's *root file system* ("C:\") but, in fact, would have only had access to a *unique root* directory on the machine – "C:\SANDBOX_LAYER_AP[P]_WORKSPACE."

[0311] Thus, that is another reason why it would have been obvious to implement Blaser-Calder to meet limitation [**1G**] under VirtaMove's apparent claim scope (*see* EX1079, 40-51).

* * *

[0312] Thus, for the reasons I discussed above (§§VII.A.4.a.i-ix), in my opinion, POSAs would have found claim 1 to have been obvious over the Blaser-Calder combination under each party's constructions.

b. Claim 2: "[C]laim 1, wherein each container has an execution file associated therewith for starting the one or more applications."

[0313] Blaser-Calder meets Claim 2 for the same reason I discussed above (§VII.A.4.a.iii.(1).b)) and reiterate below that Blaser-Calder meets part [**5**] of the patent's definition of a "*container*," which says that "*each container has its own execution file associated therewith for starting one or more applications*." EX1001, 2:38-39.

[0314] As I discussed above (§VII.A.4.a.ii.(4), §VII.A.4.a.iii.(2)), each layer includes .EXE files (***execution files***) for ***applications*** like "word processors." Blaser, 3:42-58; *see also* Blaser, 15:16-37 ("[S]uppose layer A contained B.EXE. When B.EXE ***executes***, it results in process C running."). POSAs knew that "[p]rograms with the .EXE suffix are written in…***executable*** format…." EX1069, 11:35-39. Calder, [0077] ("prepares a software package for ***execution*** on…computers"). Moreover, based on Blaser (and as would have been typical in conventional GUI-based OSs) each Blaser-Calder layer would have had "icons," where each icon is *associated with* the layer and is itself an *execution file for starting the one or more applications* or the icon is *associated with execution file*

for starting the one or more applications in the layer. Blaser, 13:25-28, 15:6-36. *See* §VII.A.4.a.ii.(4) ([**1PREB**]).

[0315] POSAs further knew that a conventional icon is an icon associated with *an execution file* for *starting the one or more applications*, as claimed. My opinion in that regard is corroborated by the evidence I cited above (§VII.A.4.a.ii.(4); citing EX1047, 1:21-67; EX1059, 1:36-38) and further corroborated by EX1070, which discloses that an "**icon represents an *executable file*** for the…***application***." EX1070, 4:50-56.

[0316] Thus, that is one way by which Blaser-Calder meets claim 2.

[0317] Additionally, Blaser's "FSLLIB32.DLL runtime library" "provides an API [application program interface] that may be used by other applications to manage the layered system," "includ[ing] functions to" "**activate** and deactivate layers" and "**enable** and disable layers." Blaser, 20:19-36. The function "FSLActivate (PTCHAR fslName)" "[v]alidates the fslName against defined layers" – i.e., the defined layers being the layers available in the system – and "[c]ommunicates with FSLX driver…to **activate** the layer[/layer group]" identified by "fslNAME." *See* Blaser, col. 20 (table's first function); *see also* ("FSLAddLayerToGroup (PTCHAR fslName, PTCHAR, groupName": "Verifies that both the specified layer [fslName] and group [groupName] are defined.").

"Applications in the layer [identified by fslName] that are specified to be run on system startup…are **started**." *See* Blaser, col. 20 (table's first listed function).

[0318] Blaser's FSLActivate would conventionally be executed using an *execution file* (e.g., Blaser's runtime library discussed above or another conventional *execution file*), which starts the layer/group of layers identified in the variable fslName, which, in turn, causes execution of *execution file(s)* in the layer for starting startup *application(s)* within the identified layer(s), as claimed.

[0319] I further note that Patent Owner alleges claim 2 is met by an "image configuration" that "includes information such as application arguments, environments, etc." EX1079, 51-55. Blaser similarly discloses a similar "layer definition" file that "include[s] layer properties and settings" (Blaser, 4:60-5:2), which POSAs understood would be interpreted by (or in a conventional implementation would have been interpreted by) an *execution file associated* with the layer *for starting applications*.

[0320] My opinion in that regard is corroborated by EX1084, which is another published patent application by the Blaser inventors that was filed on June 11, 2003 and claims priority to a provisional application (EX1085) filed on June 12, 2002.

[0321] EX1084 discusses "Portable Storage Devices Containing Layers," like Blaser's layers, stored on, for example, "Flash memory devices" or

"CD-ROM or CD-RW media." EX1084, [0145]-[0146]. The portable media device "includes software components for **automatically installing layered system drivers** contained in driver package 1006 to a Windows-type architecture." EX1084, [0147]. When portable media with layers is recognized by a computing device, the computer "attempt[s] to detect an 'autorun' configuration element, for example an **executable file** with a particular name or a script or other configuration file **specifying instructions to execute**." EX1084, [0148]. "The purpose of the installation application is to **install layered system drivers to the host operating system and to enable layers** located on the media device." EX1084, [0149].

[0322] "The installation application may then proceed to **enable the layers** stored to the portable device." EX1084, [0150]. "[T]he installation application may **activate all layers** stored to the device." EX1084, [0150]. For example, consistent with Blaser's disclosures discussed above regarding the FSLActivate function, "**the specified default application is executed or started**." EX1084, [0150]. "Alternatively, the executable started in step 1518 might include a window, icon or other widget to signal termination of the use of the portable media device and layers stored thereon." EX1084, [0151]. "Other auto-loading layered devices may be constructed for other operating systems, or for several operating systems for the same device, by following the principles given above." EX1084, [0153].

[0323] Thus, in my opinion, POSAs would have found claim 2 to have been obvious over the Blaser-Calder combination in any of the ways I discussed above.

c. Claim 3: "[C]laim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed."

[0324] As I discussed above (§VII.A.4.b, claim 2), Blaser provides the function "FSLActivate" that "activate[s] the layer" identified in an *execution file*'s code (e.g., "a runtime library," Blaser, 13:43-57) that executes FSLActivate and "[a]pplications in the layer that are specified to be run on **system startup** (in **win.ini**, **registry**, **startup folder**, etc.) are started." Blaser, col. 20, page 17 (table's first function).

[0325] The file "Win.ini" (which is a configuration file that typically includes registry keys for application startup), the referenced "startup folder," and the registry configuration for "startup" would each conventionally *include instructions related to an order in which executable applications will be executed*.

[0326] My opinion that, based on their background information, POSAs would have known it would have been typical for the conventional OSs that Blaser discusses to have an *execution file* that *includes instructions related to an order in which executable applications within will be executed* regard is further corroborated by EX1041. EX1041 discloses as background information:

> **Various application programs and system components** can also read **their own configuration and initialization files when they start up**. For example, the **WINDOWS™ program** developed and sold by the Microsoft Corporation, Redmond, Washington is a well-known graphic interface program which **has its own initialization files**. **When this program starts, it reads five initialization files**, some of which contain parameters that specify the appearance of a user interface, **a list of application programs to start automatically**, and the size and location of a swap file.

EX1041, 2:8-17.

[0327] My opinion in that, based on their background information, POSAs would have known it would have been typical and conventional to have an *execution file* that *includes instructions related to an order in which executable applications within will be executed* regard is further corroborated by EX1027. EX1027 discloses what it terms "compartments" that had "**startup and shutdown scripts** [that] are responsible for inserting rules, creating routing-tables, mounting filesystems…and other per-service initialization steps." EX1027, [0070]-[0073].

[0328] My opinion in that, based on their background information, POSAs would have known it would have been typical and conventional to have an *execution file* that *includes instructions related to an order in which executable applications within will be executed* regard is further corroborated by EX1013. EX1013 discloses as background information:

> **The selection of what applications to load and the order in which the applications are loaded by the operating system** ("the start-up sequence") is determined by **examining one or more start-up data structures maintained by the operating system**.

EX1013, [0003].

[0329] My opinion in that, based on their background information, POSAs would have known it would have been typical and conventional for the conventional to have an *execution file* that *includes instructions related to an order in which executable applications within will be executed* regard is further corroborated by EX1021 discloses:

> For example, **many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality**. **The application must know about the installation or presence of Word and be able to tap into its functions**.

EX1021, [0020]. Thus, in order for a program to use Word for the Mail Merge and/or document creation functionality, Word must be activated before the merge could occur and thus POSAs would have had reason to include an execution file that executed Word before executing any application that relied on Word for supporting mail-merge functionality.

[0330] Thus, in my opinion, POSAs would have found claim 3 to have been obvious over the Blaser-Calder combination.

d. Claim 4: "[C]laim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step [[1A]].

[0331] Blaser discloses that "bundling of an application and user files into a layer **provides a package** that may be…transported conveniently" to "provide 'pre-installed' applications as layers on CD-ROM or other media." Blaser, 6:62-7:23. Calder's teachings reinforce that Blaser-Calder's layers would have contained *pre-identified* applications and associated system files (Calder, [0082]) for transmission to and installation on computers (Calder, [0073]), which would have been stored one or more networked servers from which other servers in the system would have obtained the layers (as an alternative to installing via a physical storage device).

[0332] Because the applications and files within (i.e., associated with) a layer are stored in a transportable package – the one or more application packages that are installed to each Blaser-Calder layer and each layer itself – that is in existence **before** installation of the layer and its applications to a particular computing device (server), the layer's *applications and system files required for association with the one or more containers* would have been *pre-identified prior to said storing step*.

[0333] My opinion in that regard is corroborated by EX1084, which is another published patent application by the Blaser inventors that was filed on

December 30, 2004 and claims priority to a provisional application (EX1085) filed on December 30, 2003.

[0334] EX1084 discusses "Portable Storage Devices Containing Layers," like Blaser's layers, stored on, for example, "Flash memory devices" or "CD-ROM or CD-RW media." EX1084, [0145]-[0146]. The portable media device "includes software components for **automatically installing layered system drivers** contained in driver package 1006 to a Windows-type architecture." EX1084, [0147]. When portable media with layers is recognized by a computing device, the computer "attempt[s] to detect an 'autorun' configuration element, for example an **executable file** with a particular name or a script or other configuration file **specifying instructions to execute**." EX1084, [0148]. "The purpose of the installation application is to **install layered system drivers to the host operating system and to enable layers** located on the media device." EX1084, [0149].

[0335] "The installation application may then proceed to **enable the layers stored to the portable device**." EX1084, [0150]. Thus, the layers (and applications and system files within the layer) were pre-identified and prior to being stored to the portable media device and associated with the layer(s) on the storage device.

[0336] Thus, in my opinion, POSAs would have found claim 4 to have been obvious over the Blaser-Calder combination.

e. Claim 7: "[C]laim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container."

[0337] The '814 patent mentions the term "***mount points***" but does not explain what it means or provide an example of a mount point. EX1001, 11:17-18. POSAs understood that, conventionally, "[a] mount point simply allows a drive volume to be mounted at some point within an existing directory tree structure." EX1056, [0008]. For example, EX1018 discloses that:

> **"Mounting" a file system means taking a file system and mapping it to an existing directory in the file tree, called the mount point**. Once a file system is mounted at a given mount point, the file tree of that file system is accessed as if it is contained in the directory serving as the mount point. **Mount points are typically empty directories**, as the contents of a directory serving as a mount point will be inaccessible while the new file system is mounted there.

EX1018, [0044].

[0338] Blaser says "[w]hen [a] layer becomes **mounted** (or enabled), those application files and directories are shadowed or overlaid over the regular operating system file system." Blaser, 5:18-32. "[A]n fslrdr directory at the root of the volume" (the host's root) "contains file system information for each of the defined layers." Blaser, 29:11-28.

[0339] As I discussed above (§VII.A.4.a.ix), the example layer "TEST" is associated with the mount point "C:\fslrdr\TEST\" (Blaser, 29:11-28), which is identified in a modified system file – *e.g.*, the registry (configuration) "key called fslrdr," which keeps "all registry information contained in each layer." Blaser, 14:43-59. *See also* Calder, FIG. 47, [0135] ("sandbox directory" with mount point "C:\SANDBOX_LAYER\AP[P]_WORKSPACE\").

[0340] Thus, Blaser-Calder layer meets *modifying at least some of the system files to define container specific mount points associated with the container*.

[0341] Thus, in my opinion, POSAs would have found claim 7 to have been obvious over the Blaser-Calder combination.

f. Claim 8: "[C]laim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers."

[0342] Blaser discloses that each "layer may be **stored to a network server [*a computer system*] and *retrieved and stored*** as required to any computer." Blaser, 12:36-44. As I discussed above (§§VII.A.4.a.iii.(1).b)-c), §§VII.A.4.a.iii.(2)-(3)), each layer is a *secure container* comprising *one or more applications and associated system files*. Thus, Blaser's network server from which layers (and the one or more *applications and associated systems files* stored within each of those layers) *are retrieved from* is *a computer system having a plurality of secure containers* stored thereon.

[0343] Thus, in my opinion, POSAs would have found claim 8 to have been obvious over the Blaser-Calder combination.

[0344] Additionally, like Blaser, Calder discloses "at least **one server [*a computer system*] that transmits application packages to the member computers**." Calder, [0075]. That server is a computer system having a plurality of application packages, which comprise *applications and associated system files* that allow the application to be executed on native as well as non-native OS, as I discussed above (§VII.A.2, §VII.A.4.a.i). Thus, Calder would have also motivated (and provided a reasonable expectation of success in) having Blaser-Calder's servers retrieve their layers *from a computing system having a plurality of secure containers* (e.g., a remote server repository for Blaser-Calder layers).

[0345] As I discussed above (§VII.A.1), Blaser's example corporation embodiment uses different layers for different types of users (e.g., an engineering layer for the engineering department and an accounting layer for an accounting department). Blaser, 13:1-23. In my opinion, POSAs would have found it efficient and conventional (and thus obvious) for those different layers installed across a large corporation to be retrievable from a centralized computer system.

[0346] My opinion that it would have been efficient and conventional to retrieve applications (and thus Blaser-Calder's application layers) from a centralized computer system is further corroborated by EX1084. As I discussed

above (§VII.A.4.b (claim 2), §VII.A.4.d (claim 4)), EX1084 is a published patent application by the same Blaser inventors. EX1084 discloses:

> When first directed to server 1200 browser displays a first screen as depicted in FIG. 11A. In this example, **server 1200 stores layers related to the business of "Acme. Inc.", which are made available to persons to the internal Acme local area network**.

EX1084, [0162]-[0166]. Thus, EX1084 discloses an example of Blaser's layers being stored on a server system for a corporation, where the layers are available to other computers in the system (which, in Blaser-Calder, would include Blaser-Calder's servers).

[0347] My opinion that it POSAs would have recognized that it would have been efficient and conventional to retrieve applications (and thus Blaser-Calder's application layers) from a centralized computer system is further corroborated by EX1014. EX1014 discloses as background information a prior art feature in UNIX-based OSs called "TELNET," which "is based on human interface over a communication protocol, allowing remote operation of a machine in a ***console*** mode, as if the user was operating the machine locally." EX1014, [0005]. Using a feature like TELNET, "[t]he IT administrator" can "have a full control over the installation procedures from his own ***console*** or computer, wherever the remote physical situation of the PC client." EX1014, [0006]. EX1014 further disclose that using a console like that provides "a very simple and

effective way for controlling the setup procedure of **a software package**, **stored on a server**, and which are installed within a remote client computer, elsewhere in the LAN." EX1014, [0015].

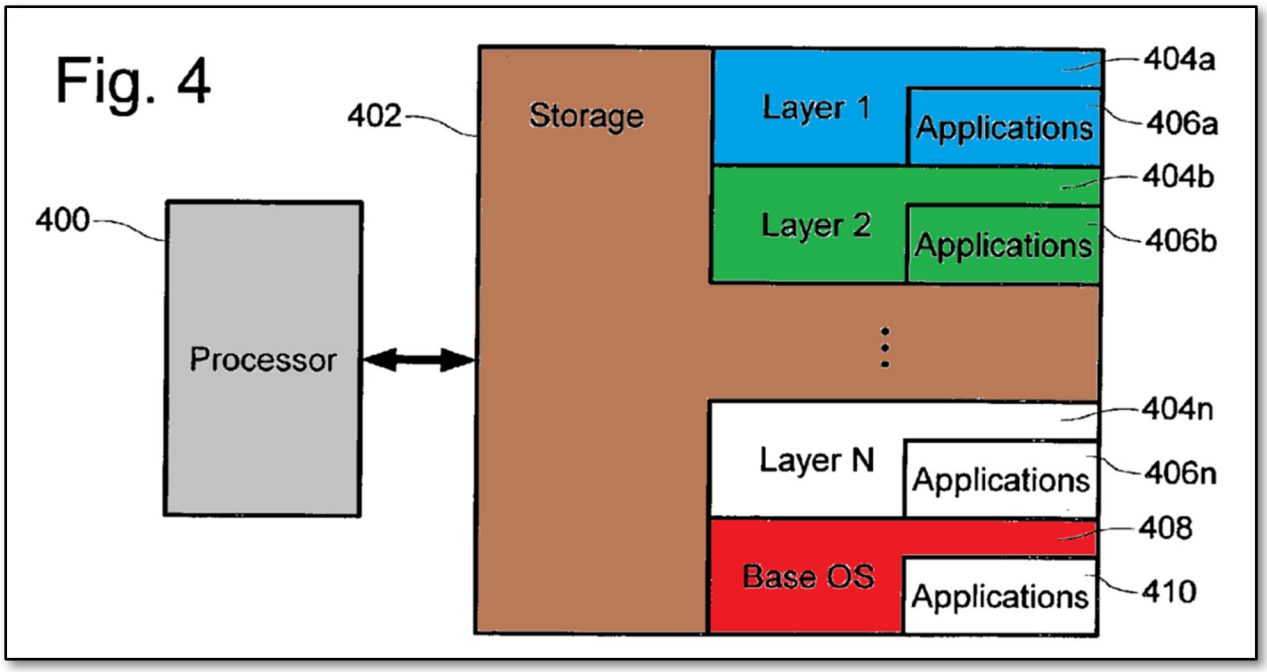
[0348] My opinion that would have recognized that it would have been efficient and conventional to retrieve applications (and thus Blaser-Calder's application layers) from a centralized computer system is further corroborated by EX1015. EX1015 discloses as background information:

> [M]ore and more individuals are **acquiring software by downloading it from remote server computers** connected to the client computers through the Internet. Additionally, **companies and organizations are distributing software** to their users **across their local area networks**.

EX1015, [0004].

[0349] Thus, in my opinion, POSAs would have found claim 8 to have been obvious over the Blaser-Calder combination based on a POSA's background knowledge.

[0350] Alternatively, as I discussed above (§VII.A.1, §VII.A.4.a.i.(1)), Blaser's Figure 4 (copied below with color added) illustrates one *computer system* (a server) having stored thereon layers 1-N and ***applications*** within each of those layers (which include *associated system files*). Blaser, 4:51-59.



Thus, when a client computer accesses, e.g., layer 1 on that *computer system*, the client computer is *retrieving from* that *computer system* (a server) one or more *applications and associated system files* (in layer 1), where that *computer system has a plurality of secure containers* (layers 2-N) stored thereon. That is another way claim 8 would have been obvious over the Blaser-Calder combination.

g. Claim 9: "[C]laim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed."

[0351] Calder teaches to modify an application package to control" use of computer resources, like "**memory** usage," "**network bandwidth** used," and "**disk usage**." Calder, [0125]-[0126]. The "grant[ed] allocation" of resources is

"predicated" (i.e., pre-assigned) using "heuristics" (e.g., "amount of virtual memory currently being consumed"). Calder, [0126].

[0352] Thus, Calder teaches allocating to applications *hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation*, which in Blaser-Calder would have been *associated with at least some of the containers prior to the applications within the containers being executed*.

[0353] Moreover, conventional prior art OSs like those that would have been used in Blaser-Calder (e.g., Unix-based OS) included commands for allocation *server hardware resources prior to executing applications* (which require *hardware resources*). For example, conventional prior art Unix-based OS include the command "quota," which "implements the policy for **restricting the disk capacity** used by each user." EX1132, [0009]-[0011] ("the UNIX quota implements the policy for restricting the disk capacity used by each user").

[0354] In my opinion, POSAs would have had motivation to apply Calder's resource-allocation teachings (and other conventional resource-allocation functionality) to Blaser-Calder's layers and allocate resources "prior to the applications within the containers being executed" because, as I discussed above (§VII.A.4.a.ii.(1), §VII.A.4.iii.(1).c)), layers are isolated and unaware of competition for hardware resources and POSAs would have had motivation to

ensure layers had sufficient resources to execute their applications. *See* Blaser, 3:42-58.

[0355] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation* and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority application / layers is corroborated by EX1036. EX1036 discloses that:

> For computer systems operating multiple workloads, **it is <u>desirable to specify how computer system resources</u>, such as central processing unit (CPU) resources <u>are allocated</u> to the workloads**. As used herein, a 'workload' refers to any process or group of processes that accesses computer system resources.

EX1036, 1:16-26.

[0356] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network*

bandwidth, and disk allocation and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority application / layers is corroborated by EX1063.[12] EX1063 discloses that:

> Different customers have different virtual server needs. For example, a company A providing large quantities of data and information to its employees and customers will want to ensure that its virtual servers are

[12] I note that EX1063 refers to "a virtual server," however POSAs would understand that EX1063 is not referring to a virtual machine (or virtual machine-like) technology, as the physical server in EX1063 is only executing a single operating system and single operating system kernel, and there is no indication or suggestion that there are multiple OS kernels executing on the physical server executing virtual servers. *See, e.g.*, EX1063, 4:41-46 (discussing that "the physical host to request the correct quality of service from **the operating system kernel** of the physical host"). But even if EX1063's virtual servers on one physical server utilized separate OS kernels, EX1063's disclosure quoted below would still corroborate my opinion that it was known to track a physical server's resources and associate them to different applications/environments executed by the same physical server.

always available to perform a large number of tasks. Company A may be willing to pay a premium for a guaranteed high quality of service, with high server availability and large amounts of processing power always on-call. By contrast, a small individual B who merely uses his virtual server for back-up file storage space has very different quality of service requirements. Customer B needs (and wishes to pay for) only a limited amount of storage space to be available on an intermittent basis.

When servicing the needs of multiple customers having different needs, it is desirable to provide a virtual server that is dynamic, not static, in its allocation of resources. A customer's virtual server is typically assigned a fixed level of resources, corresponding to either a fixed percentage of the capacity of a particular physical host…. However, customers may be unable to anticipate the exact amount of resources they will require, and a static assignment of a particular resource allocation limit may not allow the virtual server system to adapt to changing customer needs.

Instead of requiring customers to select a static level of resources, **a better resource allocation model is structured along the lines of electricity pricing—a customer receives what he needs, and he pays for what he receives**. Referring back to a previous example, small customer B may initially request a very low level of resources. However, should his new home business suddenly expand, he may quickly bump up against the limit of the server resources he originally requested. In this case, it would be preferable if customer B's virtual

server resources were able to automatically, dynamically adjust to his increased resource needs.

Thus it is desirable to provide a system and method for a virtual server capable of providing quality of service guarantees for a customer, which is also capable of adjusting the quality of service based upon changing customer demand. **It is desirable for such a system to dynamically adjust the physical host resources allocated to a virtual server**.

SUMMARY OF THE INVENTION

The present invention dynamically adjusts the quality of service guarantees for virtual servers based upon the resource demands experienced by the virtual servers. Virtual servers having individual quality of service guarantees are distributed among a group of physical hosts. Each physical host's resources are allocated among the physical host's resident virtual servers. The resources allocated to a particular virtual server may be dynamically adjusted in response to changing virtual server resource needs.

Occasionally, a physical host executing a virtual server may not have additional resources to allocate to a virtual server requiring increased resources. In this instance, a virtual server hosted by the overloaded physical host is transferred to another physical host with sufficient resources.

In one embodiment, **a dynamic resource configuration module monitors resource denials received by virtual servers and**

determines if a virtual server is overloaded based upon the resource denials. A resource denial may refer to any request by the virtual server that cannot be immediately serviced, such as a denial of a request to create a file or a network packet delay. If the resource denials received by a particular virtual server exceed a pre-specified limit, the virtual server is considered overloaded and a request is made for additional resources.

The resource usage of the physical hosts within the system is monitored. A load-balancing function is performed to select the appropriate physical host when a virtual server transfer becomes necessary. A virtual server is transferred between physical hosts with minimal impact upon the operation of the virtual server.

EX1063, 1:52-2:62.

[0357] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation* and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority application / layers is corroborated by EX1027. EX1027 discloses an example of

"error-prevention checks that may be performed by the compartment management utilities," which "**include checking for compartment name duplication and checking that sufficient memory is available**." EX1027, [0176].

[0358] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation* and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority application / layers is corroborated by EX1037. EX1037 discloses an example where:

> [E]ach user application 170 can be controlled through **explicit allocation of resources** owned by the protection domain, by the partition operating system 160, **to prevent the applications from affecting the operation of the entire system.**

EX1037, 6:29-38.

[0359] EX1076 discloses that "[a]nother area of great interest" for its technique – which uses so-called "jails" to isolate applications, similar to Blaser-Calder's isolation of applications in layers – "is the ability to **limit the impact of**

one jail on the CPU resources available for other[s]," which "would require that the jail of a process play a rule in its scheduling parameters." EX1076, 13.

[0360] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation* and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority application / layers is corroborated by EX1083. EX1083 discloses:

> Resource Aggregation Units
>
> **To <u>manage the resources of a group of processes</u> in the aggregate**, the group of processes must be transformed into a single unit. The group of processes are transformed into such a unit when a compute capsule is created. Then, the compute capsule is promoted to the status of a "first-class object" within the operating system. A first-class object can have its resources managed, and security policies for its member processes can be implemented. This enables the system to treat groups of processes as single units for accounting and resource management decisions, which has three important effects. First, **resources available anywhere in the system can be directly assigned to a particular capsule**. This may require migrating a capsule between machines,

thereby **dynamically load-balancing the system**. Second, **users may demand a guaranteed level of resources or performance isolation from others. This can be achieved by assigning fixed shares of resources to capsules**. Third, **capsules maybe subjected to policy constraints, such as restrictions or quotas on their resource allocations**, or restrictions on network access, certain file system operations, or resource or server access.

FIG. 1 is a flowchart which describes the **allocation of system resources to compute capsules** according to an embodiment of the present invention. At step 100, a compute capsule is promoted to first-class object status. At step 110, it is determined what resources need to be assigned to the compute capsule. **Resources include memory use, processor attention, or the use of a device, for instance**. **Once the resources that need to be assigned are determined, then it is determined at step 120 if the capsule must migrate to a new machine to receive the needed resources**. If it does, the capsule migrates at step 130, and the required resources are assigned at step 140. If the capsule does not need to migrate (i.e., the needed resources are available on the current machine), then the resources are assigned at step 140.

Resource Aggregation

Once a resource aggregation unit, such as a capsule, is created, various resource management operations can be performed on the unit. FIG. 2 is a flowchart showing the process of aggregating resources to compute capsules according to an embodiment of the present

invention. At step 200, a compute capsule is created. Then, at step 210, the compute capsule is promoted to first class object status. Thereafter, some form of resource management operation is performed on the compute capsule at step 220.

Some examples of the types of resource management operations that can be performed on compute capsules are shown in FIGS. 3-6. FIG. 3 is a flowchart showing how **a guaranteed share of resources can be assigned to a compute capsule according to an embodiment of the present invention**. At step 300, a compute capsule is created. Then, at step 310, the compute capsule is promoted to first class object status. Thereafter, **a guaranteed share of the resources is assigned to the compute capsule at step 320. Resources include, for instance, memory, processor attention, and a plurality of other types of resources as well**.

EX1083, [0035]-[0040].

[0361] My opinion that, based on their background knowledge, POSAs would have found it typical and conventional (and thus would have been motivated with a reasonable expectation of success) to track *server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation* and that POSAs would have had motivation to *associate* those resources *with at least some of the containers prior to the applications within the containers being executed* to ensure that each layer's applications had sufficient resources and/or to allocate resources to higher-priority

application / layers is corroborated by EX1094. EX1094 discloses an example where:

> **The UNIX operating system keeps track of all programs running in the system and allocates resources, such as disks, memory, and printer queues, as required. UNIX allocates resources so that, ideally, each program receives a fair share of resources to execute properly.** UNIX doles out resources using two methods: scheduling priority and system semaphores. **Each program is assigned a priority level. Higher priority tasks (like reading and writing to the disk) are performed more regularly**. User programs may have their priority adjusted dynamically, upwards or downwards, depending on their activity and the available system resources. System semaphores are used by the operating system to control system resources. A program can be assigned a resource by getting a semaphore by making a system call to the operating system. When the resource is no longer needed, the semaphore is returned to the operating system, which can then allocate it to another program.

EX1094, 2:5-21.

[0362] POSAs would have had a reasonable expectation of success because, as the corroboration I cite above demonstrates, pre-allocating resources to applications/processes was conventional and typical functionality for conventional, prior art OSs by the early-2000s. Thus, pre-allocating resources to layers in Blaser-Calder would have required only a POSA's ordinary skill, as it would have

required conventional computer-programming skill to *associate server information with at least some of the* layers (*containers*) *prior to the applications within the* layers (*containers*) *being executed*.

[0363] Thus, in my opinion, POSAs would have found claim 9 to have been obvious over the Blaser-Calder combination.

h. Claim 10: "[C]laim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof."

[0364] Applications in Blaser's layers "are **isolated from other applications** on a computer" "to **provide a barrier between applications** which may use conflicting configuration or library files." Blaser, 3:42-58. "[E]ach layer is a separate and individual entity within the host OS." Blaser, 5:18-32. Additionally, "layers may be provided **accessible only to** an individual user," which "provid[es] data protection and isolation between users" (Blaser, 7:59-67) and thus isolates the system files and applications in another user's layer. Thus, an application "isolated in a layer" "is **protected** from corruption **from other applications** or meddling." Blaser, 10:25-42; *see also* Blaser, 5:3-17 (layers "prevent[] accidental data modification, loss, or meddling" between applications).

[0365] Moreover, as I discussed above (§VII.A.4.a.ix [**1G**]), each layer has a unique root file system and thus the layer's *system files* (which are in the

layer's root or a subfolder/subdirectory thereof) *cannot* be *accessed* by applications in other layers. Blaser, 29:11-28.

[0366] When Blaser creates ("captures") a layer, "all operations by any application to create, modify or delete files are entered into a layer" and "[a]pplication files that would be placed on file systems managed by the OS are also redirected to into the layer," so that "[a]ll of the captured data is **held separate from the regular OS**." Blaser, 9:18-34. Requests to access local *system files* are redirected into the layer because a layer's application *has no access to system files within the operating system*, as claimed.

[0367] Accordingly, applications executed within Blaser-Calder's layers *have no access to system files or applications in other* layers (*containers*) *or to system files within the operating system during execution thereof*, and thus meets claim 10.

[0368] POSAs also had motivation, based on their background knowledge, to prevent applications in layers from accessing *system files within the operating system during execution thereof* to protect the Base OS's system files from being corrupted. For example, EX1015 discloses one known technique for doing so is assigning a "'namespace,'" which was a conventional way to "isolate software packages from one another even if the same component is used by multiple packages" ([0080]) and so "any files and components installed in that

namespace are visible to the application while files and components installed in other namespaces are not" ([0086]).

[0369] I note that VirtaMove's infringement contentions identify "namespaces" as "isolat[ing] processes in an environment." EX1079, 59.

[0370] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to ensure that applications (such as applications in Blaser-Calder's layers) *have no access to system files or applications in other containers or to system files within the operating system during execution thereof* is further corroborated by EX1076. EX1076 discloses "tak[ing] advantage of the **existing chroot(2) behaviour to limit access to the file system name-space** for jailed processes" so that applications/processes in a "jail" are "bound to a particular file system root" and "files outside of the jail file system root are protected." EX1076, 5.

[0371] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to ensure that applications (such as applications in Blaser-Calder's layers) *have no access to system files or applications in other containers or to system files within the operating system during execution thereof* is further corroborated by EX1077. EX1077 discloses a technique for "Sandboxing Applications" so "applications are **run with the chroot operation that limits their access to the FMAC managed filesystem**" and thus

"will have access only to special sanitized versions of the system files." EX1077, 120-21 (explaining that its "File Monitoring and Access Control (FMAC) tool," "implements a filesystem that mirrors the system's existing file structure," similar to Blaser's layer system).

[0372] Thus, in my opinion, POSAs would have found claim 10 to have been obvious over the Blaser-Calder combination.

i. Claim 11: "[C]laim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location."

[0373] Though claim 11 refers to *descriptors of containers*, the '814 patent's specification does not describe a *descriptor of* a *container*.

[0374] Blaser-Calder's layers (*containers*) *include* "files located to fixed disks and ***network*** **drives**" (Blaser, 7:36-47), which are *files stored in network file storage.* Blaser similarly discloses that a "layer may be ***stored*** to a ***network*** server and retrieved and ***stored*** as required." Blaser, 12:36-44; *see also* Blaser, 12:55-67 ("The **auxiliary *storage* device** might be an additional local fixed disk, ***network*** **drive** or file system, or other ***storage*** resource."); Calder, [0139] ("files can be stored remotely on separate machines, other than a client computer" that installs the application).

[0375] Blaser's "manager application" can manage *parameters forming descriptors of* layers (*containers*), such as "view[ing] information about a

layer" and "edit[ing] certain layer information." Blaser, 8:19-41, 20:19-38. That information is (or in an obvious implementation would be) *stored in a separate location* (e.g., to facilitate management by a system administrator).

[0376] POSAs would have also had reason to store *parameters forming descriptors of containers stored in a separate location*, to facilitate retrieval, management, and storage of the layers that would have been stored on remote (i.e., separate) network locations, for the reason discussed above. Some obvious parameters that POSAs would have used would have been parameters that identified the name, type, applications within, and storage location of available layers.

[0377] Thus, in my opinion, POSAs would have found claim 11 to have been obvious over the Blaser-Calder combination.

j. Claim 14

i. [14PRE] "[C]laim 1 comprising the step of creating containers prior to said step of storing containers in memory"

[0378] As I discussed above (§VII.A.4.d, claim 4), Blaser's "layer provides **a package** that may be…transported conveniently" so "application vendors can provide **'pre-installed' applications as layers** on CD-ROM or other media." Blaser, 6:62-7:23. A "packaged" layer is thus created *prior to said step of storing containers in memory* accessible to the destination server computer, as

claimed. As I also discussed above (§VII.A.4.a.iii.(1).a)), POSAs would have had motivation (with a reasonable expectation of success) to store pre-generated layers (and application(s) therein) on a layer-repository server from which other servers in Blaser-Calder would have retrieved layers for installation local to the servers (or to another network-based storage device for purposes of storing installed layers). *See also* Calder, [0073] (pre-packaged application "transferred" "from a server to the client"); EX1084, [0145]-[0153].

[0379] Thus, POSAs would have found limitation [**14PRE**] to have been obvious over the Blaser-Calder combination.

ii. [14A] "wherein containers are created by: a) running an instance of a service on a server;"

[0380] Blaser discloses that files in a layer are identified by "captur[ing]" how an application's installation process uses/changes the file system and system files. Blaser, 9:5-39, 10:1-24. Capturing begins by "execut[ing] an application installation program." Blaser, 9:18-34. Executing the application's installation program to create layers meets *running an instance of a service* provided by the application.

[0381] My opinion that executing a Blaser-Calder application in a layer is *running an instance of an* associated *service on a server* is corroborated by EX1040. EX1040 discloses:

> For a suite of ***applications*** developed for a common software platform (e.g., an application server platform), each ***application*** within the suite **typically includes a group of *services*** (e.g., content management ***services***, workflow services, publishing ***services***, search and query services) **that are common amongst the *applications* in the suite**.
>
> As the software platform does not allow **the *services* of one *application* to be shared by another**, these ***services*** must be coded for and included in each ***application*** written for the software platform, even though **multiple *applications* use common *services***.

EX1040, 1:14-25.

[0382] My opinion that executing a Blaser-Calder application in a layer is *running an instance of an* associated *service on a server* is corroborated by EX1099. EX1099 discloses:

> In general, **an application could be viewed as a collection of distributed services**. For example, Hotmail would be an application comprised of multiple services where each service performs a different function.

EX1099, [0104].

[0383] POSAs also had motivation to create layers by *running an instance of a service on a server*. For example, to create the accounting layers, POSAs would have had reason to run programs for an accounting service (Blaser, 13:1-23), which, as I explain below (§VII.A.4.j.iii, [**14B**]) would have identified

system files required to execute the engineering programs associated with that service. Additionally, POSAs would have wanted to create layers by *running an instance of a service on a server* in order to test the layers (and the applications therein) to ensure their compatibility with each server's hardware configuration and OS / OS-kernel, for the plurality of severs in the system.

[0384] My opinion that it would have been conventional and beneficial to test an application on a target computing platform (like a server) *prior to* storing the application package in long-term storage device memory (e.g., a DVD for distribution to servers) or in a memory accessible to individual servers executing the layer is further corroborated by EX1129. EX1129 discloses:

> Producers of software products **generally test their products extensively to ensure proper operation of the products on several different various operating systems and computer models**.

EX1129, 1:11-20.

[0385] My opinion that it would have been conventional and beneficial to test an application on a target computing platform (like a server) *prior to* storing the application package in long-term storage device memory (e.g., a DVD for distribution to servers) or in a memory accessible to individual servers executing the layer is further corroborated by EX1031. EX1031 discloses:

> Pre Validate Hardware Configuration of Network Element (5):

There may be dependencies between the software modules to install and the hardware configuration of the network element (2) to upgrade. E.g. the software may require that a special I/O processor is installed or a higher stepping of the main processor.

Software/hardware dependencies should be checked prior to initiating any further upgrade activities, since they will ultimately force the upgrade to backtrack. A step of the recipe is designed to check these software/hardware dependencies for the software modules included in the package distributed.

The check is performed to assure that the new software is loadable in the current hardware configuration. The software configuration states the HW compatibility window for the delivered software. The compatibility window will be compared with the installed HW revision.

The check function will return OK/NOK status. **If the check fails, the recipe will not continue unless the operator overrides the result**. The operator is warned by the system that overruling of the negative result may lead to traffic disturbances.

EX1031, [0064]-[0068].

[0386] My opinion that it would have been conventional and beneficial to test an application on a target computing platform (like a server) *prior to* storing the application package in long-term storage device memory (e.g., a DVD for distribution to servers) or in a memory accessible to individual servers executing the layer is further corroborated by EX1020. EX1020 discloses:

> For instance, administrators who manage the installation of software on a large number of computer systems **often must keep track of each computer system's hardware configuration <u>to ensure that the hardware configuration is adequate to support the software that is to be installed</u>.**

EX1020, [0007].

[0387] Thus, POSAs would have found limitation [**14A**] to have been obvious over the Blaser-Calder combination.

iii. [14B] "b) determining which files are being used; and,"

[0388] Blaser discloses that "[d]uring the install, all of the applications shared DLLs, registry entries, and .ini files that would be directed to the Windows system directories become trapped in the capture layer" and "[a]pplication files that would be placed on file systems managed by the OS are also redirected into the layer." Blaser, 9:18-34, 10:1-24. Doing that is teaching that Blaser *determines which files are being used* by the application, as claimed, or at least would have suggested to POSAs to perform a step of *determining which files are being* used by the application(s) in the layer.

[0389] Alternatively, as I discussed above (§VII.A.4.j.ii, [**14A**]), POSAs would have had additional reasons to create layers by first *running (e.g.,*

using) an instance of a service on a server to *determine* the system files *being used* and thus needed to execute the layer's application(s).

[0390] My opinion that it was known to isolate applications (like Blaser-Calder does with its layers) by having copies or modified copies of the *system files* required to run the applications is corroborated by EX1077, which discloses a technique for "Sandboxing Applications" in a chroot environment – much like Blaser and Calder provide different roots and the isolation provided by Blaser-Calder layers – by "[r]unning the application" to identify required *system files* based on what *system files are being used* by the application:

> [O]ur approach for sandboxing applications is based on the following steps:
>
> 1. **Run the application** with known benign input to create access logs **specifying its file access behavior.**
>
> 2. Use the logs to create **an access list** that can be morphed into a typical chroot environment (like the one used by ftpd). The user can augment this list based on the particular requirements of the application.
>
> 3. Create the sandbox as a chroot environment based on the access list.
>
> 4. Run the application with untrusted input in the sandboxed environment thus denying it access to unauthorized files.
>
> …

To support the approach we outlined above, we designed the File Monitoring and Access Control (FMAC) tool. … Initially **applications are run** with FMAC in passive mode to create the access log (step 1 of our approach).

…

We ran Netscape Communicator version 4.75 with the FMAC system in passive mode and we extracted, using FMAC in passive mode, **the file hierarchy shown in Figure 1**.

We separated the files into two categories, ***system* files** that are common to all installations (e.g. the /Netscape hierarchy) and *user* files that are personal files accessible by the user. The former category is typically read-only and consists of files that are not owned by the user. These are the files that will be handled by the access control part of FMAC as we will see later on:

```
/bin/
      /sh
  /dev/
      ...
  /etc/
      /group
      /kerberosIV
              /krb.conf
      /localtime
      /login.conf
      /networks
      /pwd.db
      /resolv.conf
      /spwd.db
  /usr/
      /X11R6
          /bin
          /lib
              /X11
                  /XKeysymDB
                  /app-defaults
                  /locale
                      ...
      /bin
          /su
      /lib
          ...
      /libexec
          /ld.so
      /local
          /netscape
                  /netscape
      /share
          /zoneinfo
                 ...
  /users
      /bob
          ...
  /var/
      /X11
          /app-defaults
      /db
          /kvm_bsd.db
      /mail
              /bob
      /run
          /ld.so.hints
```

Figure 1: Files required by the Netscape Communicator sandbox

…

If we do not wish to use the FMAC tool for the active phase, we will **need to create an actual file hierarchy by copying all the files included in Figure 1 to another part of the filesystem**. We would then run the chroot command as in the earlier example.

This approach relies only on the standard system tools for the active phase. **It also allows the sandbox environment to be moved to other similar platforms with little or no customization**. Thus, in a company environment we **can create the Netscape sandbox in one machine and then copy it to the machines of all the other employees**.

EX1077, 120-122 (italicized emphasis original).

[0391] Thus, POSAs would have found limitation [**14B**] to have been obvious over the Blaser-Calder combination.

iv. [14C] "c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files."

[0392] Blaser's captured files and applications are "trapped in the captured layer" and "redirected into the layer" rather than "placed on file systems managed by the OS." Blaser, 9:18-34. "[T]he captured data is **held separate from** the regular OS" (e.g., "in a data file, hard disk partition, or some other

container"). Blaser, 9:18-34. As I discussed above (§VII.A.4.a.vii, [**1E**]), each layer's system files are copies of system files that remain in the Base OS. *See also* EX1077, 120-122 (which I excerpted and annotated above).

[0393] Thus, Blaser's capturing process includes *copying applications and associated system files to memory without overwriting the associated system files*, as claimed.

[0394] Pre-existing or subsequent installation of an application would have *provided a second instance of the applications and associated system files*, as claimed. *See* Blaser, 6:62-7:23 ("Different versions of an application may be stored as layers on a single computer."), 13:29-41 ("**multiple versions of a software product are installed on a computer**, each isolated in a layer").

[0395] Moreover, POSAs would have known, based on their background knowledge, that *copying applications and associated system files to memory without overwriting the associated system files* on the server would have beneficially allowed a "sandbox [chroot'd] environment" (as in Blaser-Calder, *see supra* §VII.A.3) "to be moved to other similar platforms with little or no customization." EX1077, 122. EX1077 says that, for example, "in a company environment [they] can create the Netscape sandbox in one machine and then **copy** it to the machines of all the other employees" (i.e., the *associated local system files* from which these copies are made *remain resident on the* server). EX1077, 122.

EX1077, notes, "[h]owever, copying all these files is wasteful and more importantly, we will need to keep track of all the duplicates and update them every time we upgrade the application, or the operating system." EX1077, 122. In other words, POSAs understood that a straight-copying approach to creating sandboxed/chroot'd environments by copying *system files* into the environment may not have always been the most efficient technique, but, as EX1077 discloses, it was a known technique, with benefits over an alternative, like a copy-on-write approach. For example, a straight copying approach would have been simpler on the front-end (by simply copying rather than observing and copying on edited versions) and, as EX1077 states, straight copying allows for moving the environment (and the application and system files therein) to another machine "with little or no customization," consistent with Blaser's goal of being able to move application layers to different computers in a multi-server organization, as needed (Blaser, 13:1-23), and store "'pre-installed' applications as layers on CD-ROM or other media" (Blaser, 7:18-23).

[0396] Thus, POSAs would have found limitation [**14C**] to have been obvious over the Blaser-Calder combination.

* * *

[0397] Thus, for the reasons discussed above (§§VII.A.4.j.i-iv), in my opinion, POSAs would have found claim 14 to have been obvious over the Blaser-Calder combination.

k. Claim 16

i. [16A] "[C]laim 1 comprising the step of creating containers prior to said step of storing containers in memor,"

[0398] Blaser-Calder meet limitation [**16A**] for the same reason it meets limitation [**14A**]. *Supra* §VII.A.4.j.ii.

ii. [16B] "wherein a step of creating containers includes: using a skeleton set of system files as a container starting point and installing applications into that set of files."

[0399] The '814 patent describes a "skeleton" file system as "the OS provided system files before an application is installed." EX1001, 10:41-47.

[0400] In Blaser's "'capture' operation" for creating layers (containers), "all of the applications shared DLLs, registry entries, and .ini files that would be directed to the Windows system directories become trapped in the capture layer" and "[a]pplication files that would be placed on file systems managed by the OS are also redirected into the layer." Blaser, 9:18-34.

[0401] Blaser's layers address problems with computing systems becoming "susceptible to application conflicts with the host operating system (OS) and other applications." Blaser, 1:14-31. To prevent those conflicts, POSAs

would have had motivation, based on their background knowledge, to use *a skeleton set of system files* to serve *as a container starting point* when "capturing" applications so the skeleton files would be *installed into that set of files* to avoid creating conflicts with the base OS's local copy of those system files.

[0402] My opinion in that regard is corroborated by EX1077. EX1077 discloses a technique to reduce security risks by sandboxing applications with "all the files needed for executing the application" by "copying all the files included in Figure 1" of EX1077 – which were application system files – "to another part of the filesystem." EX1077, 119-122.

[0403] Thus, POSAs would have found limitation [**16B**] to have been obvious over the Blaser-Calder combination.

* * *

[0404] Thus, for the reasons discussed above (§§VII.A.4.i-ii), in my opinion, POSAs would have found claim 16 to have been obvious over the Blaser-Calder combination.

l. Claim 17

i. [17A] "[C]laim 1 further comprising installing a service on a target server selected from one of the plurality of servers,"

[0405] Blaser-Calder's servers (*supra* §VII.A.3) install different layers for different user types, like "install[ing] the accounting type layer" (with accounting-service applications) when "a computer is to be transferred from

engineering to accounting." Blaser, 13:16-23. Thus, Blaser-Calder meets *installing a service* [e.g., the accounting service] *on a target server*.

[0406] POSAs also had motivation, based on their background knowledge, to *install a service on a target server* so that service was available to the server's client computers (*e.g.*, accounting-group server providing accounting services). POSAs would have also had a reasonable expectation of success because installing services on a server was conventional by the early-2000s.

[0407] My opinion that it would have been conventional and beneficial to install applications and related services on a server by selecting the *target server* for the installation is corroborated by EX1014. EX1014 disclose an example where:

> The IT manager is therefore given **a very simple and effective way for controlling the setup procedure of a software package, stored on a server, and which are installed within a remote client computer, elsewhere in the LAN**.

EX1014, [0015].

[0408] My opinion that it would have been conventional and beneficial to install applications and related services on a server by selecting the *target server* for the installation is corroborated by EX1020. EX1020 disclose:

> In FIG. 3, **from the deployment console 110 the user requests from the IPS 120 a list of all the images in the image database and a list**

> **of all installation engines known to the IPS**. The installation engines known to the IPS are maintained in a database that is accessible to the IPS. **The information stored in the database that is associated with a particular installation engine includes, but is not limited to, the hostname and the network address, such as IP address, of the intelligent installation server at the installation engine**. In one embodiment of the invention, the database is created by manually entering the information for each installation engine into the database.

EX1020, [0069]-[0078].

[0409] Thus, POSAs would have found limitation [**17A**] to have been obvious over the Blaser-Calder combination.

ii. [17B] "wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server."

[0410] Blaser teaches that "[f]or ease of configuring and managing a layering system, a manager application may be provided" that "permits an administrator or user to control the presentation of applications and data on a system" by providing "facilities for importing and exporting layers" between systems. Blaser, 8:19-41.

[0411] POSAs would have had reason to implement Blaser's manager application with a GUI and *associate a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server*, as claimed, because of the well-known convenience of using GUIs (e.g., support for drag-and-drop operations using icons).

[0412] My opinion that it was part of a POSA's background knowledge to use a GUI having icons associated with a service (e.g., an application for a service) and icons associated with a computing device for hosting the applications and executing the service to cause an application to be distributed to and installed to the server is further corroborated by EX1014. EX1014 discloses a GUI showing the different "clients comprised within the network" and "the list of the different software packages which are present and available on server [] and the distribution of those between the different PC clients." EX1014, [0030]. EX1014 further discloses using the GUI to install to computers a "software package" residing on a server "[b]y dragging an icon corresponding to one software, and dropping it onto the visual icon representative of one PC client." EX1014, [0031].

[0413] My opinion that it was part of a POSA's background knowledge to use a GUI having icons associated with a service (e.g., an application

for a service) and icons associated with a computing device for hosting the applications and executing the service to cause an application to be distributed to and installed to the server is further corroborated by EX1047. EX1047 discloses:

> **In graphical user interfaces employing an object-oriented programming paradigm, application programs are typically represented to a user by an icon displayed within a window on a computer screen, one icon for each application program that can be run. Execution of an application program is initiated by selecting its corresponding icon**, most often using a pointing device such as a mouse. When an application program is selected, a message is sent to the corresponding application program object, indicating that the application program object is to invoke certain of its methods. For example, if a word processing program is selected, the methods contained within the application program object may include starting the word processing program. The user may also "drag" icons from one area of the screen to another, or from one window to another using the mouse. The user may even "drag" one icon representative of an application object and "drop" that icon on top of another. This "object-object" interaction will result in a combination of application objects. For example, if a word processing document icon is dropped upon a word processing program icon, the object-object interaction results in starting the word processing program and causing that program to open the word processing document. This is possible because both the word processing program and the word processing document have been represented as compatible application program objects. **Hence, the icons in the object-oriented programming paradigm allow the user**

to graphically control various computer system elements and the interrelationships between computer system elements.

EX1047, 1:39-67.

[0414] My opinion that it was part of a POSA's background knowledge to use a GUI having icons associated with a service (e.g., an application for a service) and icons associated with a computing device for hosting the applications and executing the service to cause an application to be distributed to and installed to the server is further corroborated by EX1020. EX1020 discloses:

> **The deployment console provides a user interface to the overall software duplication and installation process**. From the deployment console, **a user can select a source computer system, create an image of that source computer system, deploy that image to an installation engine, select a plurality of target computer systems associated with the installation engine and direct the installation engine to install a copy of the software system, represented in the deployed image, onto each of the selected target computer systems**.
>
> …
>
> The deployment console is a software interface that interfaces with the IPS. **This software interface may be a Graphical User Interface (GUI), such as Win32, a web browser, or other form of interface**. In the case of a GUI, the interface runs on a computer system and preferably connects over a network to the IPS using a socket interface. In the case of a browser, the browser runs on a computer system and

preferably acts as a web client that connects over a network to a web server that is part of the IPS.

EX1020, [0024], [0046].

[0415] My opinion that it was part of a POSA's background knowledge to use a GUI having icons associated with a service (e.g., an application for a service) and icons associated with a computing device for hosting the applications and executing the service to cause an application to be distributed to and installed to the server is further corroborated by EX1109. EX1109 discloses:

> **Generally described, computer networks can include a collection of a number of computers, such as server computer systems**, to perform various functions. In one application, a large computer network can include more than one thousand networked server computers. Such a computer network is generally referred to as an enterprise network.
>
> In some typical enterprise networks, several of the server computer systems may be remote from one another. For example, a corporation may maintain an enterprise network in which a set of server computer systems are located in North America, a set of server computer systems are located in Asia and another set of server computer systems are located in Europe. In one aspect, a system administrator can manage each individual server computer by utilizing an attached display and keyboard/mouse. **However, as will be easily understood by one skilled in the relevant art, managing an enterprise network by**

individually manipulating each server is extremely inefficient, if not impossible. For example, it would be very difficult for a single system administrator to implement a simultaneous change requiring a manipulation of each server's input devices on all multiple server computers within an enterprise network, **especially if the server computers were remote from one another**.

To mitigate the need for individually manipulating each individual server computer, some enterprise networks utilize a centralized control to remotely manage each individual server in the enterprise network. For example, some enterprise networks utilize a graphical control in which individual servers are managed by manipulating a graphical user interface ("GUI"). In such an embodiment, each server is represented as individual graphical icons within the GUI. Accordingly, the individual server icon may be selected by the system administrator to display various server properties and/or to allow the system administrator to mange [sic] server from the centralized control.

While a common user interface facilitates centralized control of a group of servers, the conventional approach remains deficient for situations in which a common action is to be implemented in a number of servers within the enterprise network. For example, a system administrator may wish to initialize one or more capacity planning measures in each of the database servers within an enterprise network. In the conventional centralized control GUI, the system administrator would then have to individually select each of the database servers and then issue and activate the measurement through the centralized user interface.

> Additionally, if the system administrator would want to slightly modify the measurement or repeat the same set of capacity planning measurement at a different time, the entire initialization process may have to be repeated for each of the servers. Accordingly, this deficiency is further accentuated for larger scale enterprise networks.

EX1109, [0002]-[0005].

[0416] POSAs would have reasonably expected success because, as demonstrated above, using GUIs to configure servers was typical and conventional and Blaser's "layer manager application" "permit[s] control and configuration of the layer[]…through a management API and library" (Blaser, 4:7-26), which would have facilitated adding a GUI for installing application layers.

[0417] Thus, POSAs would have found limitation [**17B**] to have been obvious over the Blaser-Calder combination.

* * *

[0418] Thus, for the reasons discussed above (§§VII.A.4.l.i-ii), in my opinion, POSAs would have found claim 17 to have been obvious over the Blaser-Calder combination.

m. Claim 18: "[C]laim 17 wherein the target server and the graphical user interface are at remote locations."

[0419] In my opinion, POSAs would have had reason to allow, and a reasonable expectation of success in allowing, system administrators at remote locations (e.g., corporate/IT headquarters) to configure target servers (*e.g.*, in

different cities) in the Blaser-Calder combination using a GUI. POSAs would have recognized that doing so would have beneficially allowed system administrators to configure and maintain servers remotely, rather than require administrators to present at each target server, which could be located in different cities, states, or countries in a large organization like the large corporation that Blaser considers (*see supra* §VII.A.3, §VII.A.4.a.i.(1)).

[0420] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to configure and install applications on target computing devices using a GUI at a remote location is corroborated by EX1014. EX1014 discloses using a GUI showing the different "clients [computers] comprised with the network" and "the list of the different software packages which are present and available on server [] and the distribution of those between the different PC clients. EX1014, [0030].

[0421] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to configure and install applications on target computing devices using a GUI at a remote location is corroborated by EX1020. EX1020 discloses:

> **The deployment console provides a user interface to the overall software duplication and installation process**. From the deployment console, **a user can select a source computer system, create an image of that source computer system, deploy that image to an installation**

engine, select a plurality of target computer systems associated with the installation engine and direct the installation engine to install a copy of the software system, represented in the deployed image, onto each of the selected target computer systems.

…

The deployment console is a software interface that interfaces with the IPS. This software interface may be a Graphical User Interface (GUI), such as Win32, a web browser, or other form of interface. In the case of a GUI, the interface runs on a computer system and preferably connects over a network to the IPS using a socket interface. In the case of a browser, the browser runs on a computer system and preferably acts as a web client that connects over a network to a web server that is part of the IPS.

…

One benefit of the present invention is that it enables the duplication of a software system onto a number of target computer systems without having to manually install each piece of software that comprises the software system on each target computer system.

EX1020, [0024], [0046], [0108].

[0422] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to configure and install

applications on target computing devices using a GUI at a remote location is corroborated by EX1109. EX1109 discloses:

> **Generally described, computer networks can include a collection of a number of computers, such as server computer systems**, to perform various functions. In one application, a large computer network can include more than one thousand networked server computers. Such a computer network is generally referred to as an enterprise network.
>
> In some typical enterprise networks, **several of the server computer systems may be remote from one another**. For example, a corporation may maintain an enterprise network in which a set of server computer systems are located in North America, a set of server computer systems are located in Asia and another set of server computer systems are located in Europe. In one aspect, a system administrator can manage each individual server computer by utilizing an attached display and keyboard/mouse. **However, as will be easily understood by one skilled in the relevant art, managing an enterprise network by individually manipulating each server is extremely inefficient, if not impossible**. For example, it would be very difficult for a single system administrator to implement a simultaneous change requiring a manipulation of each server's input devices on all multiple server computers within an enterprise network, **especially if the server computers were remote from one another**.
>
> **To mitigate the need for individually manipulating each individual server computer, some enterprise networks utilize a centralized**

control to remotely manage each individual server in the enterprise network. For example, some enterprise networks utilize a graphical control in which individual servers are managed by manipulating a graphical user interface ("GUI"). In such an embodiment, each server is represented as individual graphical icons within the GUI. Accordingly, the individual server icon may be selected by the system administrator to display various server properties and/or to allow the system administrator to mange [sic] server from the centralized control.

While a common user interface facilitates centralized control of a group of servers, the conventional approach remains deficient for situations in which a common action is to be implemented in a number of servers within the enterprise network. For example, a system administrator may wish to initialize one or more capacity planning measures in each of the database servers within an enterprise network. In the conventional centralized control GUI, the system administrator would then have to individually select each of the database servers and then issue and activate the measurement through the centralized user interface….

EX1109, [0002]-[0005].

[0423] Thus, in my opinion, POSAs would have found claim 18 to have been obvious over the Blaser-Calder combination.

n. Claim 19: "[C]laim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server."

[0424] The '814 patent says "[a] computer system with a single instance of a fully functional operating system installed is referred to as a **computing platform**." EX1001, 2:20-22

[0425] If a *different computing platform* includes a different physical computer, then Blaser-Calder meets claim 19 in the same way it meets claim 18, as I discussed (§VII.A.4.m).

[0426] If *different computing platform* means the platform and target server have different OSs and/or hardware, POSAs would have had reason to do so because large organizations typically had multiple OS-families and/or OS-versions on their servers, which often had different hardware configurations. *See supra* §VII.A.3 (where I discuss POSAs reason to have servers with different OSs in the Blaser-Calder system.

[0427] My opinion that POSAs would have found it conventional and beneficial based on their background knowledge to configure and install applications on target computing devices using a GUI installed on a computing platform that is a different computing platform from the target server is corroborated by EX1020. EX1020 discloses:

> **The deployment console provides a user interface to the overall software duplication and installation process**. From the deployment console, **a user can select a source computer system, create an image of that source computer system, deploy that image to an installation engine, select a plurality of target computer systems associated with the installation engine and direct the installation engine to install a copy of the software system, represented in the deployed image, onto each of the selected target computer systems**.
>
> …
>
> The deployment console is a software interface that interfaces with the IPS. **This software interface may be a Graphical User Interface (GUI), such as Win32, a web browser, or other form of interface**. In the case of a GUI, the interface runs on a computer system and preferably connects over a network to the IPS using a socket interface. In the case of a browser, the browser runs on a computer system and preferably acts as a web client that connects over a network to a web server that is part of the IPS.

EX1020, [0024], [0046]. Because the GUI discussed in EX1020 can be a web-browser-based GUI, POSAs understood that the OS running on the computing device with the installation GUI can be a different OS than the target computing device.

[0428] Thus, in my opinion, POSAs would have found claim 19 to have been obvious over the Blaser-Calder combination.

o. Claim 20: "[C]laim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server."

[0429] In my opinion, POSAs would have had reason for limitation [**17B**]'s *associating* step – associating a unique icon representing a service with an unique icon representing a server – to include *relatively moving the unique icon representing the service to the unique icon representing a server* because that would have been a conventional, convenient drag-and-drop type method of using a GUI.

[0430] My opinion that, based on their background knowledge, POSAs would have known that typical and conventional GUI functionality included drag-and-drop operations to install software to a computer based on *relatively moving* one icon to another icon is further corroborated by EX1014. EX1014 discloses using a GUI to install on computers a "software package" that is residing on a server "[b]y dragging an icon corresponding to one software, and dropping it onto the visual icon representative of one PC client." EX1014, [0031].

[0431] My opinion that, based on their background knowledge, POSAs would have known that typical and conventional GUI functionality included drag-and-drop operations to install software to a computer based on *relatively moving* one icon to another icon is further corroborated by EX1047. EX1047 discloses:

In graphical user interfaces employing an object-oriented programming paradigm, application programs are typically represented to a user by an icon displayed within a window on a computer screen, one icon for each application program that can be run. Execution of an application program is initiated by selecting its corresponding icon, most often using a pointing device such as a mouse. When an application program is selected, a message is sent to the corresponding application program object, indicating that the application program object is to invoke certain of its methods. For example, if a word processing program is selected, the methods contained within the application program object may include starting the word processing program. The user may also "drag" icons from one area of the screen to another, or from one window to another using the mouse. **The user may even "drag" one icon representative of an application object and "drop" that icon on top of another. This "object-object" interaction will result in a combination of application objects.** For example, if a word processing document icon is dropped upon a word processing program icon, the object-object interaction results in starting the word processing program and causing that program to open the word processing document. This is possible because both the word processing program and the word processing document have been represented as compatible application program objects. **Hence, the icons in the object-oriented programming paradigm allow the user to graphically control various computer system elements and the interrelationships between computer system elements**.

EX1047, 1:39-67.

[0432] My opinion that, based on their background knowledge, POSAs would have known that typical and conventional GUI functionality included drag-and-drop operations to install software to a computer based on *relatively moving* one icon to another icon is further corroborated by EX1109. EX1109 discloses:

> **To complete the initiation, the user manipulates the desired action icon with a graphical interface device, such as a mouse, for the purpose of depositing it on the selected server icon. Specifically, the user utilizes a "drag and drop" technique in which the action icon is selected, moved across the screen display until it overlaps the desired server group and then deselected**. FIG. 8 is exemplary of the screen display 120 of FIG. 7 illustrating the graphical manipulation of the action icon 126 to a selected server grouping icon. **Drag and drop techniques are well known in the relevant art and will not be discussed in further detail. One skilled in the relevant art will appreciate that alternative methods for selecting a server icon may be practiced with the present invention, including the selection of a server icon and drag and dropping it on the selected action icon**.

EX1109, [0048].

[0433] Thus, in my opinion, POSAs would have found claim 20 to have been obvious over the Blaser-Calder combination.

p. Claim 21: "[C]laim 20 further comprising starting a distributed software application."

[0434] Claim 21 refers to a *distributed software application*. A conventional *distributed software application* is an application that is "spread over several computers that are connected via a network." EX1025, [0038]. POSAs understood that Blaser's layers support including a *distributed software application* in a layer (and Blaser teaches or at least motivates *distributed software applications*) because a layer can "span[] one or more fixed disks," including those on "network drives." Blaser, 7:35-46.

[0435] Blaser-Calder would have also supported installing and *starting* various network-based applications (e.g., web-server applications) that would have been conventionally implemented as *distributed software applications*. My opinion in that regard is corroborated by EX1034, which discloses existing "Distributed Computing Frameworks or Architectures" for ***distributed software applications***:

> Exemplary ***Distributed*** **Computing Frameworks or Architectures**
>
> **Various *distributed* computing frameworks have been and are being developed in light of the convergence of personal computing and the Internet**. Individuals and business users alike are provided with a seamlessly interoperable and **Web-enabled interface for *applications* and computing devices, making computing activities increasingly Web browser or network-oriented**.

> For example, **MICROSOFT®'s .Net platform includes servers, building-block services, such as Web-based data storage and downloadable device software**. Generally speaking, the .Net platform provides (1) the ability to make the entire range of computing devices work together and to have user information automatically updated and synchronized on all of them, (2) increased interactive capability for Web sites, enabled by greater use of XML rather than HTML, (3) online services that feature customized access and delivery of products and services to the user from a central starting point for the management of various applications, such as e-mail, for example, or software, such as Office .Net, (4) centralized data storage, which will increase efficiency and ease of access to information, as well as synchronization of information among users and devices, (5) the ability to integrate various communications media, such as e-mail, faxes, and telephones, (6) for developers, the ability to create reusable modules, thereby increasing productivity and reducing the number of programming errors and (7) many other cross-platform integration features as well.

EX1034, [0074]-[0076].

[0436] In my opinion, POSAs would have had motivation to include a *distributed software application* in Blaser-Calder because it was a common, beneficial implementation for applications in computer systems like Blaser-Calder's with multiple servers and client computers. Calder itself "relates to **distributed computing**" ([0002]), with files "stored remotely **on separate machines**," where calls to a local "kernel to perform [a] file operation" are

"communicated over the network 130 to another computer or the server 120" ([0139]). Based on Calder's teachings, POSAs would have had motivation to include starting a *distributed software application* in a Blaser-Calder layer.

[0437] My opinion that POSAs knew that *starting a distributed software application* was typical, conventional, and beneficial is further corroborated by EX1025. EX1025 discloses

> **"Distributed" is a description of a software system architecture that locates components of a software application on separate, but networked computers.** This is in contrast to having all the components of a software application residing on the same computer. **A distributed software application is spread over several computers that are connected via a network**.

EX1025, [0038].

[0438] My opinion that POSAs knew that *starting a distributed software application* was typical, conventional, and beneficial is further corroborated by EX1028. EX1028 discloses that "**[i]t is common** modern software practice **to develop *software applications* which are *distributed***." EX1028, [0005].

[0439] My opinion that POSAs knew that *starting a distributed software application* was typical, conventional, and beneficial is further corroborated by EX1014. EX1014 disclose installing a "software package [that]

may be located in different locations or shared resources within the NT domain." EX1014, [0046].

[0440] Thus, in my opinion, POSAs would have found claim 21 to have been obvious over the Blaser-Calder combination.

q. Claim 22: "[C]laims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server."

[0441] Claim 22 refers to *updating a console*. The '814 patent refers to using a console but does not explain what it means by "console" or illustrate an example of a console. Thus, POSAs understood the patent to be relying on a conventional, prior art console. POSA knew that "**[c]omputing devices typically include a *console*** that is used **to control the computing device** manually…and provide communications…between an operator and the…operating system." EX1022, [0004].

[0442] EX1022's ***console*** teaches (or at least suggests) a so-called graphical console, which is also commonly referred to as a dashboard, that conventional *console*s for conventional server OSs would have had *resident on the server* and *updated* when a *service* was installed, activated, and/or updated. My opinion in that regard is further corroborated by EX1032, which discloses an example of a "***console*** application," which "allows network administrators to

configure network services engines [] network services switches [] and, thus, operation of web services network []":

> The ***console*** **application facilitates configuration of all data required for routing, security and user management**. Network services switches 56 depend on instances of network services engine 52, 54 for routing table and other associated data. Network services switches also depend on engines 52, 54 to provide functionality allowing for maintenance, configuration and support of web services network 40. For example, embodiments of network services switch 56 do not store routing table data in persistent memory, but depend on network services engine 52, 54 to provide this information on demand and/or by push updates, as required. Accordingly, configuration of network services switch 56 entails providing a computer network address of a network services engine 54 with which it communicates for routing table and other configuration data

EX1032, [0025].

> **The *console* application provides an interface facilitating access to functionality for network administrators and other users of the web services network** enabled by the present invention. For example, console application 90 provides the user interface through which web services are configured, registered, and discovered, by which access to those services may be controlled, by which network services engines 54 and switches 56 are configured, and by which the user rights of those participating in the network may be administered. In one embodiment,

the *console* application further provides access to service activity reporting capabilities and customer service functionality.

Console application 90 presents the user interface by which authenticated end-users interact with and/or configure the web services network of the present invention. In one embodiment, ***console*** application 90 is a web-based application **executed on a server** or other computing device operably connected to the computer network environment. In another embodiment, the ***console*** application may reside on a desktop computer 22 associated with a network administrator. As discussed above, the functionality allowing for maintenance, configuration, support and other administrative tasks associated with web services network 40 is, in one embodiment, available as web services residing on one or more instances of network services engines 52 and/or 54. In one embodiment, a set of SNMP interfaces allow access to such administrative functionality without the use of ***console*** application 90. To implement an end-user command, ***console*** application 90 generates an appropriate service action request and transmits it to a network services engine 52 or 54. ***Console*** application 90 is further operative to allow for configuration of all nodes associated with a web services network concurrently or individually.

EX1032, [0032]-[0033].

[0443] Thus, POSAs would have had motivation (with a reasonable expectation of success) to use a conventional *console* in Blaser-Calder so the

Blaser-Calder system could *update a console on the selected target server with information indicating that the service is resident on the selected target server*, as claimed, in order to configure the server's console to recognize and properly execute the service installed and executing thereon.

[0444] Thus, in my opinion, POSAs would have found claim 22 to have been obvious over the Blaser-Calder combination.

r. Claim 23: "[C]laim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server."

[0445] In my opinion, POSAs would have had motivation for Blaser-Calder to *test to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server*. POSAs would have wanted to test the target server before distribution and installation of software to the server so as to not waste time and resources and potentially damage servers by attempting to install layers that were incompatible with the target server and/or to target servers that did not have sufficient resources (e.g., memory, storage space, etc.) to support the layer applications.

[0446] My opinion that POSAs would have found it conventional and beneficial to test a target computing platform (like a server) *prior to distributing*

and installing the application software on the computing platform is further corroborated by EX1044. EX1044 discloses remote-installation of software programs to a computing device but first "determin[ing] whether the hardware of the remote computer system is capable of supporting the remote installation" and "whether sufficient disk space exists on the remote computer system for the installation." EX1044, 2:25-32.

[0447] My opinion that POSAs would have found it conventional and beneficial to test a target computing platform (like a server) *prior to distributing and installing* the application software on the computing platform is further corroborated by EX1031. EX1031 discloses:

> Pre Validate Hardware Configuration of Network Element (5):
>
> **There may be dependencies between the software modules to install and the hardware configuration of the network element (2) to upgrade**. E.g. the software may require that a special I/O processor is installed or a higher stepping of the main processor.
>
> **Software/hardware dependencies <u>should be checked prior to initiating</u> any further upgrade activities, since they will ultimately force the upgrade to backtrack**. A step of the recipe is designed to check these software/hardware dependencies for the software modules included in the package distributed.
>
> **The <u>check is performed to assure that the new software is loadable</u> in the current hardware configuration**. The software configuration

states the HW compatibility window for the delivered software. The compatibility window will be compared with the installed HW revision.

The check function will return OK/NOK status. **If the check fails, the recipe will not continue unless the operator overrides the result**. The operator is warned by the system that overruling of the negative result may lead to traffic disturbances.

EX1031, [0064]-[0068].

[0448] My opinion that POSAs would have found it conventional and beneficial to test a target computing platform (like a server) *prior to distributing and installing* the application software on the computing platform is further corroborated by EX1020. EX1020 discloses:

For instance, administrators who manage the installation of software on a large number of computer systems **often must keep track of each computer system's hardware configuration <u>to ensure that the hardware configuration is adequate to support the software that is to be installed</u>.**

EX1020, [0007].

[0449] Thus, in my opinion, POSAs would have found claim 23 to have been obvious over the Blaser-Calder combination.

s. Claim 24: "[C]laim 17 further comprising creating a user account for the service."

[0450] Blaser discloses that "[s]ome systems provide a multi-user environment providing a facility for **an administrator to designate layers accessible to individual *users*** and another facility to automatically **enable layers on user login and disable layers after a *user* has logged off**," to "provide layers accessible to all users or some users" and "[o]ther layers…accessible only to an individual user." Blaser, 7:59-67. Blaser also discloses that for "Windows operating systems," for example, there will be "variable values [that] may be supplied at runtime, for example, a 'CURRENTUSER' variable," that "is set to a ***user's* login name** while that ***user*** is logged in." Blaser, 8:54-65. The "CURRENTUSER variable is to provide a layered file reference for a global file that appears in each user's profile directory." Blaser, 8:65-67.

[0451] To facilitate those system-administration tasks that Blaser describes, POSAs would have had reason to *create a user account for the service* that the user may access. Doing so would have been a typical, conventional system-administrator function. By disclosing that each user has a "login name" (Blaser, 8:54-65), Blaser is teaching (or at least suggesting to) POSAs that a *user account* (the login name, or an account associated with the login name) has been *created* for the user to access the one or more *services* being provided on the server to which the user is logging into.

[0452] My opinion that POSAs would have found it to be typical, conventional functionality for system-administrators to have the ability to *create a user account for* access to a *service* offered by a computing device is corroborated by EX1032. EX1032 discloses:

> **The console application also facilitates creation and management of user accounts**. Each user account has associated therewith a set of access privileges to the functionality available over the console application or other administrative tool. **An IT administrator may have privileges to create and modify user accounts and entities. An administrative end-user associated with a subscriber may also have the ability to modify or create user accounts within the subscriber realm.** An administrative end-user associated with a publisher may have access privileges to register or delete services or actions within the publisher's realm. In one embodiment, a user account further includes access privilege information relating to specific services or actions available to the corresponding end-user. **The console application supports similar capabilities** available with respect to entities: searching, browsing, **creation of new accounts**, editing existing accounts, and deletion of obsolete accounts.

EX1032, [0041]-[0042].

[0453] Thus, in my opinion, POSAs would have found claim 24 to have been obvious over the Blaser-Calder combination.

t. Claim 25: "[C]laim 17, further comprising the step of installing files specific to the selected application on the selected server."

[0454] In Blaser, "***install[ing]***…**an *application*** can be as simple as enabling…a containing layer." Blaser, 7:24-35. That *selected application* (and *files specific to that selected application* that are within the layer) are thereby *installed on the selected* computing device (which is a *server* in Blaser-Calder).

[0455] Thus, in my opinion, POSAs would have found claim 25 to have been obvious over the Blaser-Calder combination.

u. Claim 26: "[C]laim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed."

[0456] As I discussed above (§VII.A.4.s, claim 24), Blaser's "administrator [] designate[s] layers accessible to individual users" (Blaser, 7:59-67), which POSAs understood would include (or conventionally would have included) *setting the file access permissions to allow an individual user to access the one of the applications to be distributed* and installed in the layer.

[0457] My opinion that POSAs would have found it be typical in conventional computing systems to *set file access permissions to allow a user to access the one of the applications to be distributed* is corroborated by EX1065. EX1065 discloses:

> **Most conventional computing systems operate using an operating system that provides certain basic access control mechanisms.**

> **Using such conventional access control mechanisms, a computer systems manager or administrator (a person responsible for maintaining the computer systems) can configure, for example, an operating system such as Unix in a computer system to control how that computer system allows users to access various directories and files maintained under control of the operating system**.
>
> In an operating system such as Unix, **access control is frequently provided via access control lists (ACLs) and/or by file and directory permission settings. ACLs are lists of users with associated permissions (e.g., read, write) that a systems manager can create and associate with a file and/or a directory in a filesystem**. For instance, an ACL might list all users for a file that have read access to the file, write access to the file, and execute access to the file. W**hen a user requests access to a particular file, the operating system checks the ACL associated with that file to determine if the user requesting access is listed for the type of access that is requested (e.g., read, write or execute)**. If the user is listed for the requested access, the operating system allows the requested access. Otherwise, a protection violation occurs and the operating system denies user access as requested.

EX1065, 1:33-56.

[0458] Thus, in my opinion, POSAs would have found claim 26 to have been obvious over the Blaser-Calder combination.

v. Claim 27

i. [27A] "[C]laim 1, further comprising de-installing a service from a server, comprising:"

[0459] Blaser provides for "***de-installation* of an application**" by "disabling a containing layer." Blaser, 7:24-35. By disabling the layer, Blaser will also *de-install the service* provided by the layer's application. For example, "[i]f a computer is to be transferred from engineering to accounting, the administrator [first] **removes** the engineering type layer" (thereby de-installing engineering service(s) from the computer). Blaser, 13:16-23. Thus, based on Blaser's teachings, POSAs would have had motivation for Blaser-Calder to include functionality for *de-installing a service from a server*.

[0460] Thus, POSAs would have found limitation [**27A**] to have been obvious over Blaser-Calder.

ii. [27B] "displaying the icon representing the service; [and] displaying the icon representing the server on which the service is installed; and"

[0461] As I discussed above (§VII.A.4.l.ii, [**17B**]), POSAs would have had motivation to install a service by dragging and dropping an icon for the service into an icon/folder representing the target server, using conventional GUI-based functionality. POSAs similarly had reason to *display the icon representing the service* and *display the icon representing the server on which the service is*

installed to provide a user-friendly way to identify which services are installed on each Blaser-Calder server.

[0462] My opinion that POSAs would have known as part of their background knowledge that typical, conventional GUI functionality included drag-and-drop operations to install software to a computer based on *relatively moving* one icon to another icon is further corroborated by EX1047. EX1047 discloses:

[0463] EX1047 discloses:

> **In graphical user interfaces employing an object-oriented programming paradigm, application programs are typically represented to a user by an icon displayed within a window on a computer screen, one icon for each application program that can be run. Execution of an application program is initiated by selecting its corresponding icon**, most often using a pointing device such as a mouse. When an application program is selected, a message is sent to the corresponding application program object, indicating that the application program object is to invoke certain of its methods. For example, if a word processing program is selected, the methods contained within the application program object may include starting the word processing program. **The user may also "drag" icons from one area of the screen to another, or from one window to another using the mouse. The user may even "drag" one icon representative of an application object and "drop" that icon on top of another**. This "object-object" interaction will result in a combination of application objects. For example, if a word processing document icon

> is dropped upon a word processing program icon, the object-object interaction results in starting the word processing program and causing that program to open the word processing document. This is possible because both the word processing program and the word processing document have been represented as compatible application program objects. **Hence, the icons in the object-oriented programming paradigm allow the user to graphically control various computer system elements and the interrelationships between computer system elements.**

EX1047, 1:39-67.

[0464] My opinion that POSAs would have known as part of their background knowledge that typical, conventional GUI functionality included drag-and-drop operations to install software to a computer based on *relatively moving* one icon to another icon is further corroborated by EX1109. EX1109 discloses:

> **To complete the initiation, the user manipulates the desired action icon with a graphical interface device, such as a mouse, for the purpose of depositing it on the selected server icon. Specifically, the user utilizes a "drag and drop" technique in which the action icon is selected, moved across the screen display until it overlaps the desired server group and then deselected**. FIG. 8 is exemplary of the screen display 120 of FIG. 7 illustrating the graphical manipulation of the action icon 126 to a selected server grouping icon. **Drag and drop techniques are well known in the relevant art and will not be discussed in further detail. One skilled in the relevant art will**

> **appreciate that alternative methods for selecting a server icon may be practiced with the present invention, including the selection of a server icon and drag and dropping it on the selected action icon**.

EX1109, [0048].

[0465] Thus, POSAs would have found limitation [**27B**] to have been obvious over Blaser-Calder.

iii. [27C] "utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed."

[0466] In my opinion, POSAs would have had motivation for Blaser-Calder to *utilize the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed* as a user-friendly way to de-install services from target servers, which was a conventional, user-friendly system-administrator feature.

[0467] My opinion that POSAs would have found it typical and conventional to utilize icons in a GUI-based OS to initiated de-installation of application programs (and corresponding services) is further corroborated by EX1095. EX1095 discloses:

> **Objects such as documents, applications, and OS components are represented by icons.** While the primary means of interfacing with these types of systems are the menus, there are limited visual facilities

that allow operation of the system purely via manipulation of the objects themselves. For example, **Apple's Macintosh System 7 and MicroSoft's [sic] Windows 95 allows the user to delete files by moving the icons representative of the files to a delete function object called "Trash" or the "Recycle Bin"**.

EX1095, 8:4-13.

[0468] My opinion that POSAs would have found it typical and conventional to utilize icons in a GUI-based OS to initiated de-installation of application programs (and corresponding services) is further corroborated by EX1109. EX1109 discloses:

> **To complete the initiation, the user manipulates the desired action icon with a graphical interface device, such as a mouse, for the purpose of depositing it on the selected server icon. Specifically, the user utilizes a "drag and drop" technique in which the action icon is selected, moved across the screen display until it overlaps the desired server group and then deselected**. FIG. 8 is exemplary of the screen display 120 of FIG. 7 illustrating the graphical manipulation of the action icon 126 to a selected server grouping icon. **Drag and drop techniques are well known in the relevant art and will not be discussed in further detail. One skilled in the relevant art will appreciate that alternative methods for selecting a server icon may be practiced with the present invention, including the selection of a server icon and drag and dropping it on the selected action icon**.

EX1109, [0048].

[0469] Thus, POSAs would have found limitation [**27C**] to have been obvious over Blaser-Calder.

* * *

[0470] Thus, for the reasons discussed above (§§VII.A.4.v.i-iii), it is my opinion that POSAs would have found claim 27 to have been obvious over Blaser-Calder.

w. Claim 28: "[C]laim 27 further comprising separating icon representing the service from the icon representing the server."

[0471] As discussed above (§§VII.A.4.v.ii-iii, [**27B**]-[**27C**]), POSAs would have had reason to *separate* the *icon representing the service from the icon representing the server* for de-installing services from Blaser-Calder's servers. POSAs would have had reason to allow for *separating* the server and service icons as a user-friendly way to de-install the service, using a conventional drag-and-drop operations that was a typical, conventional feature of priori art GUI-based system administration. *See* §§VII.A.4.v.ii-iii, [**27B**]-[**27C**] (citing corroboration).

[0472] My opinion that POSAs would have found it typical and conventional in a prior art GUI-based OS to de-install applications by manipulating icons is further corroborated by EX1109. EX1109 discloses:

> To complete the initiation, the user manipulates the desired action icon with a graphical interface device, such as a mouse, for the purpose of depositing it on the selected server icon. Specifically, the user utilizes

> a "drag and drop" technique in which the action icon is selected, moved across the screen display until it overlaps the desired server group and then deselected. FIG. 8 is exemplary of the screen display 120 of FIG. 7 illustrating the graphical manipulation of the action icon 126 to a selected server grouping icon. **Drag and drop techniques <u>are well known</u> in the relevant art** and will not be discussed in further detail. **One skilled in the relevant art will appreciate that alternative methods for selecting a server icon may be practiced with the present invention, including the selection of a server icon and drag and dropping it on the selected action icon**.

EX1109, [0048].

[0473] Thus, in my opinion, POSAs would have found claim 28 to have been obvious over the Blaser-Calder combination.

x. Claim 29: "[C]laim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service."

[0474] As I discussed above with corroboration (§VII.A.4.r, claim 23), POSAs would have had motivation to test whether the selected server is a va*lid platform* for installing *the service*. For similar reasons, POSAs would have had motivation (with a reasonable expectation of success) to *test whether the selected server is a valid computing platform for de-installation of the service*, as claimed, to prevent corrupting the server by improper de-installation of an application layer and so as not to waste time attempting to de-install an application layer that should

or would not have been installed or was not installed despite the attempt to de-install the service.

[0475] Thus, in my opinion, POSAs would have found claim 29 to have been obvious over the Blaser-Calder combination.

y. Claim 30: "[C]laim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation."

[0476] Blaser says it was known to "record[] a series of changes (or 'diffs') to a buffer" so "a system can be restored back to a point in time by reverse application of the diffs to the file system." Blaser, 2:12-25. Blaser also discloses that as layers use application(s) "it may be desired to record changes to the virtual file system into the writable portion of a layer." Blaser, 10:50-59. Blaser further discloses that "[w]hile in a user session, **changes are recorded to that user's layer**" so the layer can be "**retained for future use**." Blaser, 12:21-35. For example, "**a user layer may be stored to a network server and retrieved and stored as required**." Blaser, 12:36-44. POSAs understood from the above disclosure that Blaser is disclosing or at least suggesting to POSAs *copying data file changes specific to the service*, as claimed.

[0477] Moreover, as a service is executed, the service will at times change local files and thereby create a *copy* of those *data file changes specific to*

the service, which are *copied back to a storage medium from which the data file changes originated prior to installation*.

[0478] In my opinion, POSAs would have had motivation (with a reasonable expectation of success) to store those changes *back to a storage medium from which the data file changes originated prior to installation*, as claimed, to allow any data file changes to be retrievable for later user.

[0479] Thus, in my opinion, POSAs would have found claim 30 to have been obvious over the Blaser-Calder combination.

B. <u>GROUND 2:</u> Claims 5-6, 12-13, 15, and 31-34 would have been obvious to POSAs over the Blaser-Calder-Schmidt combination.

1. Schmidt-449 (EX1007)

[0480] Schmidt-449 is an issued U.S. Patent that was filed on March 22, 2001 and issued on August 16, 2005. Schmidt-449 "relates primarily to the field of computer networks, and in particular to migrating open network connections." Schmidt-449, 1:7-9. As I discuss further below, Schmidt-449 discloses a capsule technique that various services can conveniently be installed on one or more servers. Schmidt-449's capsule technique, which I discuss further below, "may be embodied in any form of computer program product. A computer program product comprises a medium configured to store or transport computer readable code." Schmidt-449, 11:35-39. Thus, Schmidt-449 is in the same field of

endeavor as the ’814 patent. *See* EX1001, 1:14-16 (“FIELD OF THE INVENTION: The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.”).

[0481] Schmidt-449 says “[m]oving (or migrating) [] programs between servers is desirable” for several reasons (e.g., when a “remote computer” “becomes busy or is off-line for repair or upgrades” or to access a “different machine” in a different location)). Schmidt-449, 2:15-28. However, when a “user migrates to another machine the IP address changes” and so “all packets received prior to migration [are] lost” and “any packets remaining in the transfer will not reach to user.” Schmidt-449, 2:39-54.

[0482] Schmidt-449’s solution to the above-identified issue of how to migrate applications to different machines, including those with different IP addresses, are software “capsules ha[ving] a unique locator, such as an IP address” that allow the capsule (and its applications) to “be moved to a different machine having potentially a different operating system or on [a] different network,” while “maintain[ing] the open network connections [the capsule] had” before migrating. Schmidt-449, 2:57-65. Each capsule “comprises one or more processes and their associated system environment,” including “state information relating to exactly what the processes are doing,” including system files like “configuration settings,

working directories and files, assigned resources, open devices, [and] installed software." Schmidt-449, 4:14-26.

2. The Blaser-Calder-Schmidt Combination

[0483] As I discuss below, in my opinion, POSAs would have had motivation to assign unique IDs like Schmidt-449's capsule IDs (*supra* §VII.B.1) to Blaser-Calder layers and would have had a reasonable expectation of success in doing so.

[0484] Blaser's layers can be "accessible to all users or some users" or "only to an individual user" (Blaser, 7:59-67) and "defined for different types of users, for example, secretaries, engineers and accountants" (Blaser, 13:1-15). One server in Blaser can have "multiple layers…active at the same time." Blaser, 15:27-36. Additionally, Blaser discloses using layers for "software product testers" that "verify software functionality against multiple development versions." Blaser, 13:29-41. Like Schmidt-449's disclosure of moving capsules of applications between computers (§VII.B.1), Blaser discloses moves layers between different computers (6:62-7:23). As I discussed above, Blaser-Calder's system (§VII.A.3), includes several servers operating different OSs, providing application layers for different services (e.g., an engineering layer for engineering services).

[0485] As I discussed above (§VII.B.1), Schmidt-449 teaches a technique for moving applications between different computers by assigning each

capsule of applications "a unique locator" so the capsule (and its application(s)) "may be moved to a different machine having potentially a different operating system or on [a] different network" (and thus often a different server) "and maintain the open network connections [e.g., the applications/processes] had prior to the migration." Schmidt-449, 2:57-65.

[0486] Thus, POASs would have recognized that assigning (and thus would have had motivation to assign) Blaser-Calder's layers unique locators like the locators Schmidt-449 assigns to its capsules (e.g., IP addresses) would have beneficially facilitated (i) exchanging data between applications and/or users isolated in different Blaser-Calder layers, (ii) moving layers to different machines (including machines running a different OS as the prior machine) and/or on different networks (e.g., networks in different locales) (e.g., for purposes of load-balancing among available servers and/or server maintenance), and/or (iii) testing/running software application layers that beneficially provided unique IP addresses when installed on one physical machine (e.g., so different users and/or companies could rent access to the same physical server that could host their respective websites (more cheaply and/or effectively than they could on their own), with each website being assigned its own, respective IP addresses, which was typical for different websites, especially those for different users or companies).

[0487] In the resulting combination (which I will refer to as the "Blaser-Calder-Schmidt" combination), each Blaser-Calder layer would have been assigned conventional, unique locators (e.g., an IP address) based on Schmidt-449's disclosure discussed above.

[0488] In my opinion, POSAs would have reasonable expected success in the combination because (1) unique locators like IP addresses were conventional identifiers, (2) Schmidt-449's capsules (like applications in Blaser-Calder's layers) are OS-independent, and (3) Schmidt-449 explicitly says its techniques are "implement[able] in any type of computer system or programming or processing environment" (Schmidt-449, 11:43-46).

3. Mapping of Blaser-Calder-Schmidt to Claims 5-6, 12-13, 15, and 31-34

[0489] As explained in the sections below (§§VII.B.3.a-i), in my opinion a POSA would have found the Blaser-Calder-Schmidt combination to have met all limitations of claims 1 5-6, 12-13, 15, and 31-34 of the '814 patent, and therefore would have rendered those claims obvious to POSAs.

a. Claim 5: "[C]laim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container."

[0490] In Blaser-Calder-Schmidt (*supra* §VII.B.2), based on Schmidt-449's teachings (*supra* §VII.B.1), each application layer (*container*) has *a*

container specific identify assigned to that *particular container* (e.g., an IP address). Schmidt-449, 2:57-65.

[0491] In my opinion, POSAs would have had motivation (with a reasonable expectation of success) to *modify at least some of the associated system files in plural containers to provide an association* to those unique locators so a layer's application could identify/communicate with an application isolated in another layer (*e.g.*, to exchange data over IP-based networks).

[0492] Thus, in my opinion, POSAs would have found claim 5 to have been obvious over the Blaser-Calder-Schmidt combination.

b. Claim 6: "[C]laim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address."

[0493] In Blaser-Calder-Schmidt (*supra* §VII.B.2), *each of* the *plurality of* layers (*containers*) is *assigned a unique associated identity* like Schmidt-449's exemplary "IP address" (Schmidt-449, 2:57-65).

[0494] Additionally, in my opinion, POSAs would have also had motivation (with a reasonable expectation of success) to use other conventional unique locators – like a "host name" and "MAC address" – to facilitate communication between layers using communication protocols that relied on those identifies (e.g., a MAC address for Ethernet networks; hostname for remote access).

[0495] My opinion that POSAs, based on their background knowledge, would have had motivation to assign each layer a unique a *host name* – a conventional, unique identifier in the prior art – is further corroborated by EX1039. EX1039 discloses as background information that a "***hostname* is a *unique* name by which a computing device may be *identified*** on a network…[and thereby] simplify access to computing devices." EX1039, [0002].

[0496] My opinion that POSAs, based on their background knowledge, would have had motivation to assign each layer a unique a *MAC address*– a conventional, unique identifier in the prior art – is further corroborated by EX1019. EX1019 discloses "virtualization of local area networks (LANs) or the virtualization of I/O storage" so that:

> processing resources may be deployed rapidly and easily through software via configuration commands, e.g., from an administrator, rather than through physically providing servers, cabling network and storage connections, providing power to each server and so forth.

EX1019, [0023]. EX1019 further discloses "**assign[ing] virtual *MAC addresses* to each processor node**" so "processing resources may be deployed rapidly and easily through software via configuration commands." EX1019, [0012], [0023].

[0497] My opinion that POSAs, based on their background knowledge, would have had motivation to assign each layer a unique *IP address* and/or *host*

name – each, a conventional, unique identifier in the prior art for web servers – is further corroborated by EX1130. EX1130 discloses as background information:

> **Every host and router on the Internet has an *IP address***. The Domain Name System (DNS) is often used to map ***host names*** to these ***IP addresses***. By way of example, a client typically sends a DNS query to a DNS server which includes a ***host name*** and an indication that an ***IP address*** is requested. The DNS server then returns an ***IP address*** **associated with the *host name***. It would therefore be desirable if a DNS server were designed to accomplish load distribution compatible with the Internet and DNS.

EX1130, 2:6-14.

[0498] My opinion that POSAs, based on their background knowledge, would have had motivation to assign each layer a unique *IP address* and/or *host name* – each, a conventional, unique identifier in the prior art for web servers – is further corroborated by EX1131. EX1131 discloses as background information:

> In general, all the machines on the Internet can be categorized into two types: servers and clients. Typically, machines that provide services (like Web servers, FTP servers, Email servers, etc.) are servers. Servers are loaded with the appropriate software in order to allow them to perform their intended services. Machines that request information from servers are typically called clients. **In order to differentiate between machines on the network, each machine is given a unique address called an *IP address***.

The IP address is a thirty-two bit number that is normally expressed as 4 octets in a dotted decimal number (e.g., 192.168.1.101). Each of the octets can have values between 0 and 255 (2^8 possibilities per octet). **When a client connects to the Internet, the client is assigned an *IP address*** through their Internet Service Provider (ISP) for the duration of the connection. Conversely, the IP addresses of servers are relatively static, and do not change very often.

Because it is difficult for clients to remember IP addresses, and because IP addresses need to change, most servers on the Internet possess domain names (e.g., "www.whoknowz.com") to help users reach their intended servers without remembering strings of numbers. **Name servers, used in the domain name system (DNS), map the human-readable names [e.g., host names] into *IP addresses* to help clients reach their destinations**. When a client enters a domain name, the browser (via a resolver) extracts the domain name and passes it to a name server, which will return the correct ***IP address*** **to the associated site**. The Domain Name System is comprised of a distributed database and name servers that access that database.

EX1131, 1:27-56.

[0499] Thus, in my opinion, POSAs would have found claim 6 to have been obvious over the Blaser-Calder-Schmidt combination.

c. Claim 12: "[C]laim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1."

[0500] Though claim 12 refers to *merging* and *parameters*, the '814 patent's specification does not use the term "*merging*" and does not explain what it means by "*parameters*" or provide examples thereof.

[0501] As I noted above (§VII.A.4.f, claim 8), Blaser's layer (*container*) files can be retrieved from storage of a "network server." Blaser, 12:36-44. The retrieved layer "provide files and directories overlaid on the existing fixed disk file systems" (Blaser, 12:36-44), which merges the files with the existing file system.

[0502] In my opinion, POSAs also had reason to merge retrieved files *with the parameters* discussed above for claim 11 (§VII.A.4.i), *to affect* claim 1's step of storing (§VII.A.4.a.iii.(1), [**1A**]). For example, POSAs would have had reason to have the *parameters forming descriptors of containers* (§VII.A.4.i, claim 11) include information about where each layer (container) should be stored, the layer's name, the layer's mount point, etc. Those parameters – e.g., the layer's name, mount point, and storage device storing the layer – would have been conventional parameters that would have been *merged* with *the files stored in network storage to affect the step of storing in claim 1*. For example, a conventional, convenient *parameter* that would have been obvious to use would

have been a *parameter* identifying the storage location of the layer, which would have been used to *affect the step of claim 1's storing*, by identifying the storage device on which the layer would be stored and/or retrieved from.

[0503] Thus, in my opinion, POSAs would have found claim 12 to have been obvious over the Blaser-Calder-Schmidt combination.

d. Claim 13: "[C]laim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application."

[0504] As I discussed above (§VII.A.1), Blaser's exemplary OSs include prior art Windows-based OS and conventional prior art non-Windows OSs (e.g., Unix-based OSs). Blaser, 13:43-45. POSAs knew that those prior art OSs typically and conventionally included tools for measuring/tracking *processes related to an application* being executed by those OSs, including *tracking statistics for resource allocation and for monitoring the status of the application(s)*.

[0505] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art

OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1033.

EX1033 discloses:

> **There exists known performance monitoring software in various forms. Among them is software known as PerfMon software, which is commercially available from Microsoft Corporation. PerfMon software is a utility which, among other things, can provide an indication of the utilization of the computer's central processor unit (CPU) and memory unit**. PerfMon software operates by sampling. That is, it **tracks continuous data by monitoring a machine and looking at its behavior. It can track the free space on a disk, monitor network usage, and so on**, but it cannot gather event-based information, such as what function was most recently started.
>
> **There also exist known tools called profilers. These look at a single executing software application and try to understand its performance**. They do this either by monitoring the program (in a similar way to PerfMon software), or else they hook into the program they are monitoring and generate "events" each time a program subcomponent (function) commences or completes.
>
> …
>
> **The performance analysis component analyzes the collected data and creates a call tree by pairing events (e.g. Call and Return) and ordering them using temporal ordering and heuristics**. The result is a presentation of the call tree in a Gantt style view with any Perfmon (or other dynamic) data displayed adjacent to or overlying the displayed call tree."

…

> FIG. 19 illustrates this concept. Here, for example, the user made a time selection in the performance view window 556 (representing PerfMon data) over a period of time where CPU behavior was in question. The animated application model or process diagram 552 highlights the entities/processes involved in the selection. **The event log window 554 highlights all events in the specified time range, part of which represent a call tree**.

EX1033, 2:62-3:18, 34:30-39, 36:53-60.

[0506] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1055. EX1055 discloses:

> The determination may also use, for example, **a Windows® application program called SYSMON.EXE** which accesses **the Performance Registry of Windows® and allows monitoring of system performance and resources.**

EX1055, 5:35-39.

[0507] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1060. EX1060 discloses:

> The number of concurrent clients was gradually increased while the scripts were cycled through the various applications. Thus **multiple test runs were conducted and additional sets of <u>Performance Monitor (PERFMON) log files</u> were produced to verify reproducibility**.

EX1060, 5:57-61.

[0508] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1094. EX1094 discloses:

> The UNIX operating system keeps track of all programs running in the system and allocates resources, such as disks, memory, and printer

queues, as required. UNIX allocates resources so that, ideally, each program receives a fair share of resources to execute properly. UNIX doles out resources using two methods: scheduling priority and system semaphores. Each program is assigned a priority level. Higher priority tasks (like reading and writing to the disk) are performed more regularly. User programs may have their priority adjusted dynamically, upwards or downwards, depending on their activity and the available system resources. System semaphores are used by the operating system to control system resources. A program can be assigned a resource by getting a semaphore by making a system call to the operating system. When the resource is no longer needed, the semaphore is returned to the operating system, which can then allocate it to another program.

…

UNIX **maintains** a great deal of information about the files that it manages. For each file, the file system **keeps track** of the file's size, location, ownership, security, type, creation time, modification time, and access time. All of this information **is maintained automatically by the file system** as the files are created and used. UNIX file systems reside on mass storage devices such as disk drives and disk arrays. UNIX organizes a disk into a sequence of blocks. These blocks are usually either 512 or 2048 bytes long. The contents of a file are stored in one or more blocks which may be widely scattered on the disk

EX1094, 2:5-21, 4:20-30.

[0509] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1120. EX1120 discloses that one system call in the Unix operating system was called "times," which:

> returns the elapsed real time in clock ticks from an arbitrary fixed time in the recent past, and fills *tbuf* with account information: [such as] CPU time spend in user mode[.]

EX1120, 450 (emphasis original). Thus, *tbuf* would include a stored history of clock ticks in various modes, which is a *tracking statistic* of when applications in user mode where consuming CPU resources. Based on that teaching, POSAs would have wanted to *associate with* the Blaser-Calder layers (*containers*) *a stored history of when processes related to applications within the containers* were consuming the CPU's clock cycles (the CPU's *resources*), for *tracking-statistic* purposes (tracking how many clock cycles each layer's applications were requesting), which POSAs, in turn, had motivation to use for *resource allocation* (e.g., to ensure each application layer would be installed to a server with sufficient clock cycles available to execute the application(s) in that layer).

[0510] My opinion that, based on their background knowledge, POSAs would have known that functionality for *storing* the *history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application* was conventional and beneficially provided by and/or typically used with the same conventional prior art OS used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) is corroborated by EX1121. EX1121 discloses a definition of "process descriptor" for generic, conventional OSs as:

> A process descriptor is a data structure in an operating system that contains all the information the operating system needs to **record about the process**. This information includes:…how much processor time and **other resources** the process has used….

EX1121, G-14. Thus, EX1121 corroborates my opinion that it would have been conventional for prior art OSs like those used in Blaser-Calder (*supra* §VII.A.4.a.i.(1)) to maintain *a stored history of when processes related to applications are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application*.

[0511] In my opinion, POSAs would have had motivation (and a reasonable expectation of success) to use those conventional performance-monitoring and tracking tools in Blaser-Calder-Schmidt to track the history of processes in layers for the same reason POSAs had to measure process history in

conventional prior art computer systems. For example, POSAs would have had motivation to measure and maintain a history of computing resources so as to determine how to allocate computing resources and make decisions on the purchase of future equipment, maintenance, and support staff.

[0512] My opinion that POSAs would had motivation to beneficially track and maintain a history of computing resources based on conventional, prior art techniques is further corroborated by EX1093. EX1093 discloses:

> The present invention relates to the monitoring of server performance in a network like the Internet. **System administrators, as well as information technology (IT) executives, generally like to know the productivity of their systems, and want to accurately forecast future needs for upgrading or expanding existing systems. Capital expenditures must be budgeted according to changes in server resource requirements, based on key server operating parameters such as CPU utilization, memory availability, I/O usage, and permanent storage (disk) capacity**. This objective has become very difficult to achieve, due to extremely complex environments of hundreds or thousands of networked servers running on many different platforms, including UNIX and its variations (such as AIX, HP-UX, System V, and Mach), Windows NT, OS/2, NetWare, etc. (UNIX is a trademark of UNIX System Laboratories; AIX is a trademark of International Business Machines Corp (IBM); HP-UX is a trademark of Hewlett-Packard Co.; System V is a trademark of American Telephone & Telegraph Co.; Mach is a trademark of NeXT Computers;

Windows NT is a trademark of Microsoft Corp.; OS/2 is a trademark of IBM; and NetWare is a trademark of Novell Corp.). In addition to these platforms, some servers are managed by specialized systems, such as SAP (a suite of bundled business software including an accounts receivable program, a materials management program, a scheduling program, etc.).

…

The foregoing objects are achieved in a method of **monitoring a computer network, generally comprising the steps of collecting resource data from a plurality of network nodes, analyzing the resource data to generate historical performance data,** and reporting the historical performance data to another network node. In one implementation, the plurality of network nodes are servers operating on different platforms, and the collecting step includes the step of gathering resource data using separate programs having different application programming interfaces for the respective platforms. The analyzing step can generate daily, weekly, and monthly historical performance data, based on a variety of resources including **CPU utilization, memory availability, I/O usage, and permanent storage capacity.** The report may be constructed using a plurality of documents related by hypertext links. The hypertext links can be color-coded in response to at least one performance parameter in the historical performance data surpassing an associated threshold. An action list can also be created in response to at least one performance parameter in the historical performance data surpassing an associated threshold.

EX1093, 1:66-3:6.

[0513] My opinion that POSAs would had motivation to beneficially track and maintain a history of computing resources based on conventional, prior art techniques is further corroborated by EX1121. EX1121 discloses a definition of "efficiency" for OSs as:

> Efficiency is a goal of a resource manager. An efficient use of resources is one where the least amount of the resource is used to accomplish the stated goal. Operating systems worry about efficient use of processor time and memory space.

EX1121, G-5. Thus, EX1121 corroborates my opinion that POSAs would have had motivation to monitor resources in Blaser-Calder-Schmidt to develop algorithms for efficient use of resources.

[0514] Thus, in my opinion, POSAs would have found claim 13 to have been obvious over the Blaser-Calder-Schmidt combination.

e. Claim 15

i. [15A] "[C]laim 14 comprising the steps of: assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;"

[0515] Blaser-Calder-Schmidt meets [**15A**] for the same reason I discussed above that Blaser-Calder meets claim 14 (§VII.A.4.j) and Blaser-Calder-Schmidt meets claim 6 (§VII.B.3.b). Thus, in my opinion, POSAs would have

found limitation [**15A**] to have been obvious over the Blaser-Calder-Schmidt combination.

ii. [15B] "installing the container on a server; and,"

[0516] Blaser discloses "'pre-installed' applications" provided as "layers on CD-ROM or other media." Blaser, 6:62-7:23. As I discussed above (§VII.A.4.f, claim 8), POSAs also had reason to store pre-generated layers (and one or more applications installed therein) on one or more layer-repository servers and associated storage device(s). Those pre-installed / pre-generated layers are for *installing the container* layer *on a* Blaser-Calder server (*supra* §VII.A.4.a.i.(1)) in the Blaser-Calder-Schmidt combination. Thus, in my opinion, POSAs would have found limitation [**15B**] to have been obvious over the Blaser-Calder-Schmidt combination.

iii. [15C] "testing the applications and files within the container."

[0517] Blaser's "'pre-installed' applications" on CD/media are "**pre-tested**." Blaser, 6:62-7:23. Blaser discloses that "software product **testers**" can use layers to 'verify software functionality against multiple development versions." Blaser, 13:29-41. Thus, Blaser discloses *testing the applications and files within the* layer (*container*).

[0518] My opinion that POSAs would have had motivation, based on their background knowledge, to test the *applications and files within* each Blaser-Calder layer (which would have been implemented in various server devices running different OSs) is also corroborated by EX1129. EX1129 discloses as background information that:

> Producers of software products **generally test their products extensively to ensure proper operation of the products on several different various operating systems and computer models**.

EX1129, 1:11-20.

[0519] Thus, in my opinion, POSAs would have found limitation [**15C**] to have been obvious over the Blaser-Calder-Schmidt combination.

* * *

[0520] Thus, for the reasons discussed above (§§VII.B.3.e.i-iii), in my opinion, POSAs would have found claim 15 to have been obvious over the Blaser-Calder-Schmidt combination.

f. Claim 31

i. [31PRE] "A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:"

[0521] As I discussed above (§VII.A.3, §VII.A.4.a.i), Blaser-Calder would have provided a system for installing layers (containers) of applications to a "computing system" comprising, *inter alia*, a plurality of servers running different

OSs. As I also discussed above (§VII.A.4.a.ii.(4), §VII.A.4.a.iii.(1).b) ("container" definition part [**4**]), a layer's applications are *for performing a plurality of tasks* – e.g., an engineering layer for performing engineering tasks – that *comprise a plurality of processes*. Blaser, 13:1-23.

[0522] Thus, in my opinion, POSAs would have found limitation [**31PRE**] to have been obvious over the Blaser-Calder-Schmidt combination for the reasons I discussed above that Blaser-Calder renders obvious limitations [**1PREA**]-[**1PREB**] (§§VII.A.4.i-ii) and [**1A**] (§VII.A.4.a.iii).

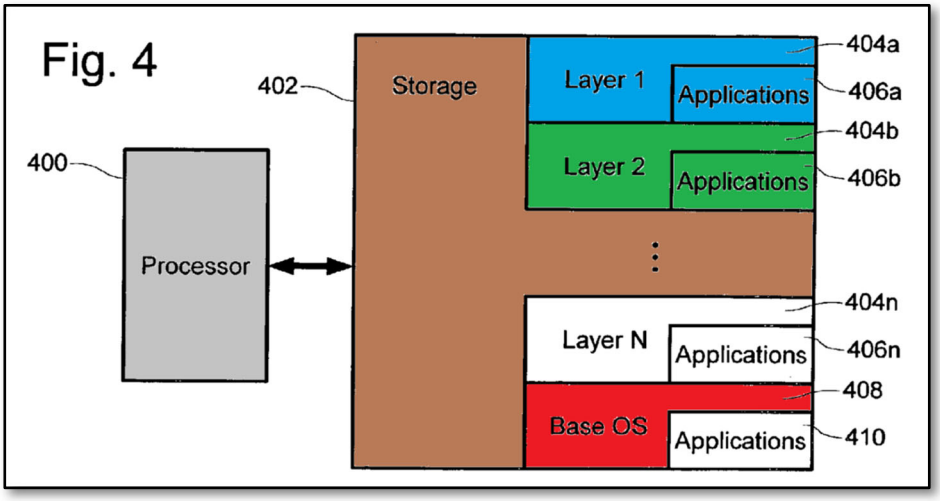
ii. [31A] "a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, "

[0523] For the same reason I discussed above that Blader-Calder meets [**1A**] (§VII.A.4.a.iii), the layers in Blaser-Calder-Schmidt's system are *secure containers of associated files accessible to, and for execution on, one or more servers*.

[0524] Thus, in my opinion, POSAs would have found limitation [**31A**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason Blaser-Calder renders obvious limitation [**1A**].

iii. [31B] "each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers,"

[0525] As I discussed above (§§VII.A.4.a.iii.(1).b)-c), [**1A**]), application sets in Calder-Blaser's layers (*containers*) are insulated from effects of other application sets (i.e., they are *mutually exclusive of* each *other*). Similarly, Blaser-Calder-Schmidt's "application layers…**are isolated** from other applications on [the] computer" because layers "provide a barrier between applications." Blaser, 3:42-58, FIG. 4 (annotated below) (illustrating distinct layers). Blaser further discloses that when "applications are stored individually in layers, **interactions between application files may no longer occur** due to conflicting shared libraries (DLLs)." Blaser, 6:62-7:23. Thus, Blaser-Calder-Schmidt's layers (*containers*) are *mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers*.



[0526] As I also discussed above (§VII.A.4.a.ix, [**1G**]), each layer has its own unique root directory that is mutually exclusive of the root directories of other layers. *See* Blaser, 29:11-28; Calder, [0138], [0241]. The layer's read/write files would be located under the layer's mutually exclusive root and thus *cannot be shared with other* layers (*containers*).

[0527] Thus, in my opinion, POSAs would have found limitation [**31B**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder meets limitation [**1A**] (specifically, part [**2**] of the parties' construction of *container*, *supra* §VII.A.4.a.iii.(1).b)).

iv. [31C] "each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address;"

[0528] Blaser-Calder-Schmidt meets [**31C**] for the same reason it meets Claim 6 (*supra* §VII.B.3.b).

[0529] Thus, in my opinion, POSAs would have found limitation [**31C**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious claim 6.

v. [31D] "wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes,"

[0530] Blaser-Calder-Schmidt meets [**31D**] for the same reason Blaser-Calder meets part [**4**] of the patent's definition of a "*container*." *Supra* §VII.A.4.a.iii.(1).b) ([**1A**]).

[0531] Thus, in my opinion, POSAs would have found limitation [**31D**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious limitation [**1A**] (specifically, part [**4**] of the patent's definition of a *container*). *Supra* §VII.A.4.a.iii.(1).b) ([**1A**]).

vi. [31E] "and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system,"

[0532] Blaser-Calder's layers have "*associated system files for use in executing the one or more processes*" for the reasons discussed above (§VII.A.4.a.iii.(1).b), §§VII.A.4.a.iii.(2)-(3) ([**1A**])). For the same reason Blaser-Calder meets [**1D**]-[**1E**] (*supra* §§VII.A.4.a.vi-vii), Blaser-Calder-Schmidt meets "*the associated system files [in a layer] are files that are copies of files or modified copies of files that remain as part of the operating system*."

[0533] Thus, in my opinion, POSAs would have found limitation [**31E**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious limitations [**1A**], [**1D**]-[**1E**].

vii. [31F] "each container having its own execution file associated therewith for starting one or more applications,"

[0534] Blaser-Calder-Schmidt meets [**31D**] for the same reason Blaser-Calder meets part [**5**] of the patent's definition of a "container" (*supra* §VII.A.4.a.iii.(1).b), [**1A**])) and claim 2 (*supra* §VII.A.4.b). Thus, in my opinion, POSAs would have found limitation [**31F**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious limitation [**1A**] (specifically, part [5] of the patent's definition of a *container*) and claim 2.

viii. [31G] "in operation, each container utilizing a kernel resident on the server and"

[0535] When "*in operation*," each Blaser-Calder layer (*container*) *utilizes a kernel resident on the server* for the reasons discussed above that each layer is *for use with a local kernel residing permanently on one of the servers* (§VII.A.4.a.iii.(1).b), § VII.A.4.a.iii.(4), [**1A**]). Thus, in my opinion, POSAs would have found limitation [**31G**] to have been obvious over the Blaser-Calder-

Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious limitation [**1A**].

ix. [31H] "wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,"

[0536] Each Blaser-Calder layer (container) *uses a kernel in an underlying operation system in which it is running* ([**31G**], [**1A**]) and *is absent its own kernel* for the same reason Blaser-Calder's layer meets part [**4**] of the patent's definition of a "container" (§VII.A.4.a.iii.(1).b)) and limitation [**1C**] (§VII.A.4.a.v). Because a Blaser-Calder-Schmidt layer (*container*) would not have included a kernel, the layer *exclusively uses* the Base OS's kernel of the server where the layer is installed, for the reasons I discussed above (§VII.A.4.a.iii.(1).b), §VII.A.4.a.v).

[0537] Thus, in my opinion, POSAs would have found limitation [**31H**] to have been obvious over the Blaser-Calder-Schmidt combination for the same reason I discussed above that Blaser-Calder renders obvious limitation [**1A**] (specifically, part [4] of the patent's definition of a *container*) and limitation [**1C**].

x. [31I] "a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications."

[0538] Blaser discloses that for Windows-based OSs, "a layering system is formed by adding several files to the stock Windows operating system," "including a runtime library FSLLIB32.DLL," "and an FSLX driver which is either an FSLX.VXD driver (for 95/98/ME based platforms) or an FSLX.SYS driver (for NT based platforms)." Blaser, 13:43-57. "[T]he FSLLIB32.DLL runtime library provides an API…to manage the layered system and communicate with the FSLX driver, and further provides system management function implementations." Blaser, 20:19-36.

[0539] The FSLX driver "operate[s] to intercept key file system and registry calls and manipulate the results to create the appearance that virtual files and registry settings contained in the layer definitions exist in the real file system and real registry." Blaser, 14:60-15:1. For example, "[w]hen requests come that access virtual files or virtual registry settings, these requests are redirected by the FSLX driver to the proper locations in the layer." Blaser, 14:60-15:1.

[0540] Thus, in my opinion, POSAs would understand Blaser's FSLLIB32.DLL "runtime library" that uses the FSLX driver meets a *run time module for monitoring system calls from applications associated one or more* layers (*containers*). As discussed above, that functionality is implemented by

"intercept[ing] key file system and registry calls" and is *for providing control of the one or more applications*, for example, by "redirect[ing] [access requests] to the proper locations in the layer." Blaser, 20:19-36.

[0541] That is one way by which Blaser-Calder-Schmidt would have rendered obvious limitation [**31I**].

[0542] Moreover, Blaser-Calder-Schmidt incorporates Calder's teaching of translating system calls for using applications that use those system calls on native and non-native OSs. Calder, [0101]; *supra* §VII.A.3, §VII.A.4.a.i.(1). As Calder teaches, some run-time software module (e.g., an "interception module") will *monitor system calls from the applications* in order to translate them to the system call format / syntax required by the Base OS on the server, and thus *provide control of the one or more applications*. Calder, [0086]-[0087]. That is another way by which Blaser-Calder-Schmidt would have rendered obvious limitation [**31I**].

[0543] Thus, in my opinion, POSAs would have found limitation [**31I**] to have been obvious over the Blaser-Calder-Schmidt combination.

* * *

[0544] Thus, for the reasons discussed above (§§VII.B.3.f.i-x), it is my opinion that POSAs would have found claim 31 to have been obvious over the Blaser-Calder-Schmidt combination.

g. Claim 32: "[C]laim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources."

[0545] As I discussed above (§VII.A.4.g, claim 9), POSAs would have had reason to pre-allocate system resources to Blaser-Calder's layers (and for the same reason Blaser-Calder-Schmidt's layers). POSAs similarly had reason, and would have found it obvious (with a reasonable expectation of success), to include *a scheduler comprising values related to an allotted time in which processes within a* layer (*container*) *may utilize predetermined resources*, as a conventional way to assign a server's resources to its applications.

[0546] My opinion that it would have been conventional to allocate utilization of predetermined computing resources based on an allotted time is further corroborated by EX1120. EX1120 discloses that a conventional system call in a Unix-based OS is utime, which "sets the access and modification times of the specified file according to the value of ***times*** [a parameters provided to the utime system call]." EX1120, 452 (italics original).

[0547] My opinion that it would have been conventional to allocate utilization of predetermined computing resources based on an allotted time is further corroborated by EX1034. EX1034 discloses:

> Beginning with the first time-sharing system in the mid-1960s, operating systems (OSs) have implemented numerous methods of

> allowing multiple applications to share computational resources of a computer without knowledge of one another. By **allocating small 'time slices' to each application, and interrupting when a 'time slice' has expired**, a computer can present each application with the illusion that it is running alone on the computer.

EX1034, [0003].

[0548] My opinion that it would have been conventional to allocate utilization of predetermined computing resources based on an allotted time is further corroborated by EX1094. EX1094 discloses:

> The UNIX operating system keeps track of all programs running in the system and allocates resources, such as disks, memory, and printer queues, as required. UNIX allocates resources so that, ideally, each program receives a fair share of resources to execute properly. **UNIX doles out resources using two methods: <u>scheduling priority</u>** and system semaphores. Each program is assigned a priority level. Higher priority tasks (like reading and writing to the disk) are performed more regularly. User programs may have their priority adjusted dynamically, upwards or downwards, depending on their activity and the available system resources.… When the resource is no longer needed, the semaphore is returned to the operating system, which can then allocate it to another program.

EX1094, 2:5-21.

[0549] Thus, in my opinion, POSAs would have found claim 32 to have been obvious over the Blaser-Calder-Schmidt combination.

h. Claim 33: "[C]laim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address [sic]."

[0550] As I discussed above (§VII.B.3.g, claim 32), Blaser-Calder-Schmidt would have rendered obvious a *run time module* that *intercepts system calls* from reaching the host's kernel and redirects them into the layer. Blaser, 14:60-15:3; Calder, [0139]. As I discussed above (§VII.B.3.b, claim 6), POSAs would have had reason to assign each layer (*container*) *at least one of the IP address, the host name and the Mac_Address*.

[0551] POSAs would have known that a conventional way to implement those conventional, unique identifiers would have been for the *run time module* that *intercepts system calls from any of the plurality of containers* to replace the host's identifier with the container's identifier (or vice-versa) to ensure the layers *can run independently of one another without contention*, as claimed, as Blaser teaches (as I discussed above, §VII.A.4.a.ii.(1), §VII.A.4.a.ii.(4), §VII.A.4.a.iii.(1).b), §VII.A.4.a.ix, §VII.A.4.h). For example, if a layer made a

system call to request the server's IP address, Blaser-Calder-Schmidt's run-time module would (or in an obvious implementation thereof would) have provided the layer's unique IP address in response to that request.

[0552] Thus, in my opinion, POSAs would have found claim 33 to have been obvious over the Blaser-Calder-Schmidt combination.

i. Claim 34

i. [34A] "[C]laim 31, wherein the run time module performs: monitoring resource usage of applications executing;"

[0553] Blaser-Calder-Schmidt meets [**34A**] for the same reasons I discussed above that POSAs would have had motivation to allocate resources to Blaser-Calder's layers (*containers*) (§VII.A.4.g, claim 9) and track/monitor *resource usage of applications executing* in layers (§VII.B.3.d, claim 13). Thus, POSAs would have found [**34A**] to have been obvious over the Blaser-Calder-Schmidt combination.

ii. [34B] "intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;"

[0554] POSAs knew that applications typically run in *user mode* and thus need to make *system calls to kernel mode from user mode to kernel mode* (e.g., to access the server's hardware). My opinion in that regard is corroborated by EX1120, which discloses:

> The execution of user processes on UNIX systems is divided into two levels: **user** and **kernel**. When a process executes a **system call**, the *execution mode* of the process changes from ***user mode* to *kernel mode***: the operating system executes and attempts to service the user request, returning an error code if it fails.

EX1120, 15 (italicized emphases original).

[0555] As I discussed above (§VII.A.3, §VII.A.4.iii.(4)), applications in Blaser-Calder-Schmidt's layers share a *kernel*, and *system calls* made by applications in a layer are either (i) intercepted and redirected into the layer (e.g., if a requested system file is within the layer) or (ii) passed to the Base OS's kernel (e.g., if server hardware is required and/or a system file is not present in the layer). Blaser, 14:60-15:3; Calder, [0139].

[0556] Calder, for example, discloses that its techniques for translating system calls (which would have been used in Blaser-Calder) operates via an "***interception*** module [that] ***intercepts system calls*** that are made by the ***application*** program" to the *kernel*, which are (or conventionally would have been) *made by the respective application from user mode to kernel mode*. Calder, [0076]; *see also* Calder, [0074]-[0076].

[0557] Thus, in my opinion, POSAs would have found limitation [**34B**] to have been obvious over the Blaser-Calder-Schmidt combination.

iii. [34C] "comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits."

[0558] In my opinion, POSAs would have had reason (and it would have been obvious with a reasonable expectation of success) to *compare the monitored resource usage of the at least one respective application with the resource limits* in Blaser-Calder-Schmidt that I discussed above (§VII.B.3.g, claim 32) and only *forward the system calls to a kernel on the basis of the comparison*. POSAs would have wanted to do so to ensure that the allocated resource limits would not be violated if the system call were passed to the kernel without first comparing the monitored resources to resource limits. As I discussed above with corroboration (§VII.A.4.g, claim 9), the conventional OSs that would have been used in Blaser-Calder-Schmidt (e.g., Windows-, Unix-, and Linux-based OS) included functionality that *monitored resource usage* allocated to a user (and thus to the *applications* executed by the user) and that only *forward system calls* by the user's applications *to the kernel based on comparing the monitored resource and the resource limits* of the user. EX1132, [0009]-[0011] (noting that Unix includes the command "quota," which "implements the policy for **restricting the disk capacity** used by each user"). Thus, POSAs would have found limitation [**34C**] to have been obvious over the Blaser-Calder-Schmidt combination.

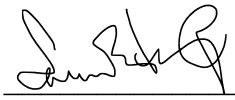
* * *

[0559] Thus, for the reasons discussed above (§§VII.B.3.i.i-iii), it is my opinion that POSAs would have found claim 34 to have been obvious over the Blaser-Calder-Schmidt combination.

I declare that all statements made herein of my own knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

I declare under penalty of perjury that the foregoing is true and correct.

Dated: Jan 31, 2025



Samrat Bhattacharjee

VIII. APPENDIX: CLAIM LISTING

The following claim listing assigns element labels (*e.g.*, [**1PREA**], [**1PREB**], [**1A**], etc.) to certain claims for convenience of reference.

Claim 1
[**1PREA**] In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor,
[**1PREB**] a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:
[**1A**] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;
[**1B**] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,
[**1C**] the containers of application software excluding a kernel,
[**1D**] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,
[**1E**] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,
[**1F**] and wherein the application software cannot be shared between the plurality of secure containers of application software,
[**1G**] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

Claim 2
A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.
Claim 3
A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.
Claim 4
A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.
Claim 5
A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.
Claim 6
A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.
Claim 7
A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.
Claim 8
A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.
Claim 9
A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

Claim 10
A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.
Claim 11
A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.
Claim 12
A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.
Claim 13
A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.
Claim 14
[**14PRE**] A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory,
[**14A**] wherein containers are created by: a) running an instance of a service on a server;
[**14B**] b) determining which files are being used; and,
[**14C**] c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.
Claim 15
A method as defined in claim 14 comprising the steps of: assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation; installing the container on a server; and, testing the applications and files within the container.

Claim 16
[**16A**] A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory,
[**16B**] wherein a step of creating containers includes: using a skeleton set of system files as a container starting point and installing applications into that set of files.
Claim 17
[**17A**] A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers,
[**17B**] wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.
Claim 18
A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.
Claim 19
A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.
Claim 20
A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.
Claim 21
A method as defined in claim 20 further comprising starting a distributed software application.
Claim 22
A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

Claim 23
A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.
Claim 24
A method according to claim 17 further comprising creating a user account for the service.
Claim 25
A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.
Claim 26
A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.
Claim 27
[**27A**] A method as defined in claim 1, further comprising de-installing a service from a server, comprising: displaying the icon representing the service;
[**27B**] displaying the icon representing the server on which the service is installed; and
[**27C**] utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.
Claim 28
A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.
Claim 29
A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.
Claim 30
A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

Claim 31
[**31PRE**] A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:
[**31A**] a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers,
[**31B**] each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers,
[**31C**] each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC_address;
[**31D**] wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes,
[**31E**] and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system,
[**31F**] each container having its own execution file associated therewith for starting one or more applications,
[**31F**] each container having its own execution file associated therewith for starting one or more applications,
[**31H**] wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
[**31I**] a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.
Claim 32
A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

Claim 33
A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the MAC_Address.
Claim 34
[**34A**] A computing system as defined in claim 31, wherein the run time module performs: monitoring resource usage of applications executing;
[**34B**] intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;
[**34C**] comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

APPENDIX – NON-PATENT LITERATURE

APPENDIX TABLE OF CONTENTS

I. OVERVIEW OF SEVERAL RESOURCES....3

A. Internet Archive....3

B. ACM and ACM Digital Library....5

II. OPINIONS REGARDING PUBLIC ACCESSIBILITY....6

A. Public Accessibility of the Kamp Article (EX1076)....6

B. Public Accessibility of the Prevelakis Article (EX1077)....10

C. Public Accessibility of Microsoft's Computer Dictionary 5th Edition (EX1088)....17

D. Public Accessibility of the Bach Book (EX1120)....19

E. Public Accessibility of the Crowley Book (EX1121)....21

F. Public Accessibility of the Eckel Book (EX1122)....23

1. I understand from counsel that a printed publication qualifies as publicly accessible as of the date it was disseminated or otherwise made available such that any member of the public interested in the subject matter could locate it through the exercise of reasonable diligence.

2. While I understand that the determination of public accessibility under the foregoing standard rests on a case-by-case analysis of the facts particular to an individual publication, I also understand that a printed publication is rendered "publicly accessible" if it is cataloged and indexed by a library such that a member of the public interested in the subject matter a could locate it exercising reasonable diligence (i.e., I understand that cataloging and indexing by a library in a manner that permits a member of the public interested in the subject matter to locate the publication is sufficient, though there are other ways that a printed publication may qualify as publicly accessible, such as being indexed and catalogued in an online database). One manner of sufficient indexing is indexing according to subject matter category. I understand that the cataloging and indexing by a single library of a single instance of a particular printed publication is sufficient, even if the single library is in a foreign country. I also understand that the cataloging and indexing of information that would guide a person interested in the relevant subject matter to the printed publication, such as the cataloging and indexing of an abstract

for the printed publication, is sufficient to render the printed publication publicly accessible.

3. I understand that a document may also be publicly accessible if it is catalogued and indexed in an online digital database such that a person interested in the relevant subject matter could locate the document in the database exercising reasonable diligence. For example, the ACM Digital Library is an example of a digital database that is well-regarded among the academic community and considered to be accurate and trustworthy source of information regarding the scientific and academic publications catalogued in the database. As I discuss in Appendix-§I.B below, I have personally used that and similar databases and, in my opinion, POSAs would have been aware of this database (and similar databases like IEEEXplore) and would have used it to search for and retrieve publications in the art of management and deployment of server applications and related fields at the time of the alleged invention.

4. I understand that routine business practices, such as general library cataloging and indexing practices, can be used to establish an approximate date on which a printed publication became publicly accessible. I further understand that the date on which digital databases, like the ACM Digital Library, indicate a document was published and/or added to the database can be used to establish a date on which the document became publicly accessible.

5. In my opinion, POSAs would have been engaged in research, learning through study and practice in their field, and possibly through formal instruction, the bibliographic resources relevant to their research. In my opinion, in the early-2000s POSAs would have had access to a vast array of long-established print and online resources providing indexing information, abstracts, and full-text services for publications in the field of computing.

6. In my opinion, POSAs would have been able to locate the material discussed in this declaration as well as any additional documents I discuss in the sections below on their own or with the assistance of a research librarian, with relative ease using the tools and resources described in the sections below.

I. OVERVIEW OF SEVERAL RESOURCES

A. Internet Archive

7. Exhibits 1133-1145 were captured by the Internet Archive at the indicated dates and made browsable by the Internet Archive's Wayback Machine. The Internet Archive's Wayback Machine is well known to researchers and the general public as a service that provides archives of webpages as they existed on dates in the past. I am personally familiar with the Internet Archive and its Wayback Machine from my own use of the service in my work and research. It is well known that the Internet Archive collects and archives captures of webpages using web crawlers, which are software programs that surf the Web, visiting

websites and automatically storing copies of the webpages and other files represented by the URLs that the crawler visits. Crawlers are also used by Internet search engines (e.g., Google, Bing, etc.) to access webpages to be provided as search results, as is also well known to researchers and the general public. When the Internet Archive's crawlers visit a URL on a particular date, they store a copy of that webpage or file at the URL as it existed on that date. Users of the Wayback Machine can later access that archived capture of the URL to view and interact with the webpage or retrieve a file as it existed on that past date.

8. It is well known to users of the Internet Archive's Wayback Machine, and I am personally aware from my own use of the Wayback Machine, that the date on which an archived webpage was captured by the Internet Archive's crawlers can be observed in the URL that the Internet Archive assigns to the archived capture (what the Internet Archive calls an "extended URL"). When an archived capture of a webpage is viewed through the Wayback Machine on the Internet Archive's website (http://web.archive.org), the URL of the archived capture is http://web.archive.org/web, followed by numbers indicating the year, month, day, and time at which the webpage was captured by the Internet Archive's crawlers, followed by the original URL of the archived webpage. For example, a Wayback Machine URL of http://web.archive.org/web/20111231171057/http://www.uspto.gov:80/ is the

Internet Archive's archived capture of the USPTO's homepage (http://www.uspto.gov:80/) captured on December 31, 2011 (2011/12/31), at 17:10:57 (5:10 p.m. and 57 seconds).

B. ACM and ACM Digital Library

9. The Association for Computing Machinery (ACM) is "the world's largest educational and scientific computing society, delivers resources that advance computing as a science and a profession."[1] According to the ACM's official homepage, there are "more than 100,000 members" of the ACM.[2] The ACM's website links to a digital database (currently called the ACM Digital Library), which is "the world's most comprehensive database of full-text articles and bibliographic literature covering computing and information technology. This renowned repository includes the complete collection of ACM publications plus an extended bibliographic database of core works in computing from scholarly publishers."[3] I have personally used the ACM Digital Library to retrieve copies of

[1] https://www.acm.org/about-acm (last visited January 16, 2025).

[2] https://www.acm.org/about-acm/about-the-acm-organization (last visited January 16, 2025).

[3] https://www.acm.org/publications/digital-library (January 16, 2025); *see also* https://dl.acm.org/ (January 16, 2025).

articles or presentations published in journals in the field of computing and information technology and in my opinion POSA at the time of the alleged invention would have been aware of the ACM and the ACM Digital Library and could have retrieved an article or presentation published by the ACM or publishers indexed by the ACM Digital Library by exercising reasonable diligence (e.g., by obtaining a copy of an article from a library, by being a member of or subscriber of the publication, or by navigating to the ACM Digital Library).[4]

II. OPINIONS REGARDING PUBLIC ACCESSIBILITY

A. Public Accessibility of the Kamp Article (EX1076)

10. EX1076 (which I refer to herein as "Kamp") is titled "Jails: Confining the omnipotent root." Based on the evidence I discuss below, it is my opinion that Kamp is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

11. According to the face of EX1076, Kamp was published as part of the "The FreeBSD Project." According to a footnote on the bottom of EX1076, Kamp

[4] *See, e.g.*, August 15, 2000 Wayback Machine archive of the ACM Digital Library, at https://web.archive.org/web/20000815053844/http://www.acm.org/dl/.

"was sponsored by http://www.servetheweb.com/ and donated to the FreeBSD Project for inclusion in the FreeBSD OS. FreeBSD 4.0-RELEASE was the first release including this code." According to the Semantic Scholar database,[5] Kamp was "Published 2000."[6]

12. As I discuss below, a copy of Kamp also appears to have been publicly accessible from the NLUUG Unix User Group – The Netherlands' website more than one year before the earliest possible priority date of the Asserted Patent. I understand from counsel that a POSA may use other publicly accessible information as "research aids" as a roadmap to locating additional prior art in the same or related field.

[5] According to the Semantic Scholars website, "Semantic Scholar provides free, AI-driven search and discovery tools, and open resources for the global research community." *See* https://www.semanticscholar.org/about (last checked January 16, 2025).

[6] https://www.semanticscholar.org/paper/Jails%3A-confining-the-omnipotent-root-Kamp-Watson/a659692ed5031c5410668d644a60d7806bb2efbb?year%5B0%5D=2001&year%5B1%5D=2003&sort=relevance&page=2 (last checked January 16, 2025).

13. For example, the NLUUG's website (as archived by the Internet Archive on June 4, 2002) states as follows:

> The NLUUG (UNIX User Group - The Netherlands) is the association of professional UNIX users in the Netherlands.
>
> Since the late seventies, the NLUUG has brought together the community of UNIX systems administrators, programmers, researchers and IP network professionals.
>
> The primary goal of the NLUUG is to extend the application of, and knowledge about, UNIX and related open systems.[7]

14. In my opinion, POSAs in the field of the challenged patent would have had reason to navigate to the NLUUG's website, to find information about publications or, in some cases, copies of publications relating to the field of the challenged patent.

15. EX1141 is the Internet Archive's archived capture of the NLUUG's homepage, http://www.nluug.nl/, as it existed on July 22, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20010722200036/http://www.nluug.nl/. In the middle of the NLUUG's homepage, under the heading "Conferences," there is text with hyperlinks stating "More information about upcoming and previous events can be

[7] https://web.archive.org/web/**20020604031**649/http://www.nluug.nl/folder/index-uk.html (last checked January 16, 2025) (emphasis added to emphasize the archival date of this website by the Internet Archive).

found in the calender [sic] of events." Clicking on the "previous" events navigates to http://www.nluug.nl/events/indexOld.html. EX1142 is the Internet Archive's archived capture of that webpage as it existed on August 3, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20010803092814/http://www.nluug.nl/events/indexOld.html.

16. As shown on EX1142, there is a hyperlink to the "Sane 2000, 'Second International SANE Conference', May 22-25, 2000 Maastricht The Netherlands,'" reachable by clicking on the text "SANE 2000." Clicking on that text navigates to http://www.nluug.nl/events/sane2000/index.html. EX1143 is the Internet Archive's archived capture of that webpage as it existed on August 12, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20010812213235/http://www.nluug.nl/events/sane2000/index.html.

17. As shown on EX1143, there is a hyperlink to "SANE 2000 papers,'" reachable by clicking on that text. Clicking on that text navigates to http://www.nluug.nl/events/sane2000/papers.html. EX1144 is the Internet Archive's archived capture of that webpage as it existed on June 20, 2001, as reflected in the Internet Archive's extended URL:

https://web.archive.org/web/20010620043826/http://www.nluug.nl/events/sane2000/papers.html.

18. As shown on EX1144, there is a hyperlink to "Jails: Confining the omnipotent root (*PDF*), by Poul-Henning Kamp,'" reachable by clicking on the hyperlink text. Clicking on that text navigates to http://www.nluug.nl/events/sane2000/papers/kamp.pdf, which is a copy of the "Jails: Confining the omnipotent root" article in a PDF form. EX1145 is a screen shot of a portion of the Internet Archive's archived capture of that PDF as it existed on September 2, 2000, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20000902205700/http://www.nluug.nl/events/sane2000/papers/kamp.pdf.

19. Kamp (EX1076) is a copy of the PDF retrieved from that link.

20. Based on the above evidence, I conclude that Kamp would have been publicly accessible, via (at least) the NLUUG's website, to a POSA exercising reasonable diligence by September 2, 2000, or at least by August 12, 2001.

B. Public Accessibility of the Prevelakis Article (EX1077)

21. EX1077 (which I refer to herein as "Prevelakis") is titled "Sandboxing Applications." Based on the evidence I discuss below, it is my opinion that Prevelakis is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the

face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

22. According to the face of EX1077, Prevelakis was published in the June 25-30, 2001 issue of the conference proceedings of the *FREENIX Track 2001 USENIX Annual Technical Conference*, that occurred in Boston, Massachusetts, USA. Page 3 of EX1077's PDF bears the text "© 2001 by The USENIX Association," indicating it was published and copyrighted by The USENIX Association in 2001 and thus publicly accessible by the year 2001. EX1077's Page 2 bears what appears to be a stamp from a library, indicating the date when EX1077 was received by that library (August 2001).

23. Additionally, the ACM Digital Library (*see supra* I.B) includes an entry for an article bearing the same title as Prevelakis, published in the same proceedings of the USENIX Annual Technical Conference discussed above, at the same pages of the proceedings indicated in EX1077 (i.e., pages 119 to 126).[8] The ACM Digital Library indicates that that article was "Published: 25 June 2001."[9]

[8] *See* https://dl.acm.org/doi/10.5555/647054.715767 (last checked January 16, 2025).

[9] *Id.*

24. Based on the above evidence, I conclude that Prevelakis would have been publicly accessible to a POSA exercising reasonable diligence at least by the end of 2001.

25. As I discuss below, a copy of Prevelakis also appears to have been publicly accessible from the USENIX Association's website more than one year before the earliest possible priority date of the Asserted Patent. I understand from counsel that a POSA may use other publicly accessible information as "research aids" as a roadmap to locating additional prior art in the same or related field.

26. For example, as the USENIX Association's website states:

> The USENIX Association is a 501(c)(3) nonprofit organization, dedicated to supporting the advanced computing systems communities and furthering the reach of innovative research. **We are known for organizing conferences and publishing research**, but our greatest strength lies in building communities in computing systems. We represent the interests of our communities in a variety of ways, including our professional affiliation with the Computing Research Association.
>
> **Since 1975, USENIX has brought together the community of engineers, researchers, and practitioners** working on the cutting edge of the computing world. **USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on the development of all aspects of computing systems**.[10]

[10] *See* https://www.usenix.org/about (last checked January 16, 2025).

27. In my opinion, POSAs in the field of the challenged patent would have had reason to navigate to the USENIX website, to find information about publications or, in some cases, copies of publications relating to the field of the challenged patent.

28. EX1133 is the Internet Archive's archived capture of the USENIX Association's homepage, http://www.usenix.org, as it existed on November 14, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20011114235021/http://www.usenix.org:80/. On the left of USENIX Association's homepage, there is a set of "SHORTCUTS." Selecting "Conference Proceedings" navigates to USENIX's "Proceeding" webpage at http://www.usenix.org/publications/library/proceedings/. EX1134 is the Internet Archive's archived capture of that webpage as it existed on November 15, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20011115072346/http://www.usenix.org/publications/library/proceedings/.

29. As shown on EX1134, there is a hyperlink to the "2001 USENIX Annual Technical Conference, Boston, MA," reachable by clicking on that text. Clicking on that text navigates to http://www.usenix.org/publications/library/proceedings/usenix01/, which provides information about the 2001 USENIX Annual Technical Conference. EX1135 is

the Internet Archive's archived capture of that webpage as it existed on November 15, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20011115173912/http://www.usenix.org/publications/library/proceedings/usenix01/.

30. As shown on EX1135, there is a hyperlink to the "Conference Proceedings," including a hyperlink to "FREENIX Track Refereed Papers," reachable by clicking on that text. Clicking on that text navigates to http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/technical.html, which provides a list of the "REFEREED PAPERS" for the "FREENIX Track: 2001 USENIX Annual Technical Conference." EX1136 is the Internet Archive's archived capture of that webpage as it existed on November 18, 2001, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20011118162406/http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/technical.html.

31. As shown on EX1136, there is a hyperlink to a paper from "Friday, June 29, 2001" titled "Sandboxing Applications," by "Vassilis Prevelakis, University of Pennsylvania, and Diomidis Spinellis, Athens University," reachable by clicking on that text. Clicking on that text navigates to http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/prevelakis.html, which provides an "Abstract" of that article. EX1137 is the Internet

Archive's archived capture of that webpage as it existed on January 12, 2002, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20020112043538/http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/prevelakis.html.

32. I note that page 6 of the PDF of EX1077 similarly lists Prevelakis under the heading "Friday, June 29," with the same title and authors (and associated universities) as shown in EX1137. *See also* EX1077 at 119. I further note that the text of the "Abstract" in EX1137 matches the text of the "Abstract" on page 119 of Prevelakis in EX1077 (i.e., the first page of the article).

33. As shown on EX1137, there are hyperlinks to "View the full text of this paper in HTML form, and PDF form," each of which is reachable by clicking on the text that says "HTML" and "PDF," respectively.

34. Clicking on the "HTML" text shown on EX1137 navigates to http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/full_papers/prevelakis/prevelakis_html/index.html, which provides the text of the "Sandboxing Applications" article in an HTML website. EX1138 is the Internet Archive's archived capture of that webpage as it existed on March 29, 2003, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20030329030611/http://www.usenix.org/publications/

library/proceedings/usenix01/freenix01/full_papers/prevelakis/prevelakis_html/index.html. The bottom of EX1138 states as follows (italic original):

> ***This paper was originally published*** *in the Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Masssachusetts [sic], USA*
>
> ***Last changed: 21 June 2001*** *bleu [sic]*

35. I have compared the text of EX1138 with the copy of Prevelakis in EX1077. Except for some differences in formatting, the text and figures of EX1138 and EX1077 appear to be identical.

36. Clicking on the "PDF" text shown on EX1137 navigates to http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/full_papers/prevelakis/prevelakis.pdf, which a copy of the "Sandboxing Applications" article in aa PDF form. EX1139 is a screen shot of a portion of the Internet Archive's archived capture of that PDF as it existed on January 19, 2004, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/20040119150721/http://www.usenix.org/publications/library/proceedings/usenix01/freenix01/full_papers/prevelakis/prevelakis.pdf.

37. EX1140 is a copy of the PDF retrieved from that link. The cover page of EX1140 states that it was published in connection with the same "Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference," in "Boston, Massachusetts, USA" on "June 25-30, 2001" that is indicated on the

cover of EX1077. Like EX1077, page 1 of EX1140 bear the text "© 2001 by The USENIX Association," indicating it was published and copyrighted by The USENIX Association in 2001 and thus publicly accessible by the year 2001.

38. I have compared the text of EX1140 with the copy of Prevelakis in EX1077. Except for some differences in formatting on the cover page of EX1140 and the footers at the bottom of EX1077, the text and figures of EX1138 and EX1077 appear to be identical.

39. Based on the above evidence, I conclude that Prevelakis would have also been publicly accessible, via the USENIX Association's website, to a POSA exercising reasonable diligence by January 12, 2002, or at least by March 29, 2003.

C. Public Accessibility of Microsoft's Computer Dictionary 5th Edition (EX1088)

40. EX1088 (which I refer to herein as "Microsoft Dictionary") includes excerpts from a book titled "Microsoft Computer Dictionary Fifth Edition." *See* EX1088, pdf. page 1. Based on the evidence I discuss below, it is my opinion that Bach is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

41. Page 2 of EX1088's PDF bears the text "Copyright © 2002 by Microsoft Corporation" and that it was "PUBLISHED BY Microsoft Press A

Division of Microsoft Corporation," indicating it was copyrighted and published by Microsoft in 2002 and thus publicly accessible by the year 2002.

42. Page 2 of EX1088's PDF further includes an "ISBN" number, which I understand "uniquely identifies your book, and facilitates the sale of your book to bookstores (physical and digital) and libraries."[11]

43. According to the U.S. Library of Congress's online catalog, a book bearing the same name, publisher, and ISBN number as EX1088 was "Published/Created" "c2002."[12]

44. According to the ACM Digital Library, a book bearing the same name and publisher as EX1088 was "Published: 01 May 2002."[13]

---

[11] *See* https://www.isbn.org/ (last checked January 16, 2025).

[12] https://catalog.loc.gov/vwebv/search?searchArg1=0735614954&argType1=all&searchCode1=KNUM&searchType=2&combine2=and&searchArg2=&argType2=all&searchCode2=GKEY&combine3=and&searchArg3=&argType3=all&searchCode3=GKEY&year=1525-2025&fromYear=&toYear=&location=all&place=all&type=all&language=all&recCount=25 (last checked January 16, 2025).

[13] https://dl.acm.org/doi/10.5555/515434 (last checked January 16, 2025).

45. Based on the above evidence, I conclude that the Microsoft Computer Dictionary would have also been publicly accessible, via (at least) its publisher Microsoft Press or a library (e.g., the Library of Congress), to a POSA exercising reasonable diligence at least by May 2002.

D. Public Accessibility of the Bach Book (EX1120)

46. EX1120 (which I refer to herein as "Bach") includes excerpts from a book titled "THE DESIGN OF THE UNIX® OPERATING SYSTEM," by Maurice J. Back. *See* EX1120, pdf. page 2. Based on the evidence I discuss below, it is my opinion that Bach is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

47. Page 3 of EX1120's PDF bears the text "Copyright © 1986 by Bell Telephone Laboratories, Incorporated" and that it was "Published by Prentice-Hall, Inc.," indicating it was copyrighted by Prentice-Hall, Inc. and published in 1986 and thus publicly accessible by the year 1986. Page 3 of EX1120's PDF further states that "[t]he Publisher offers discounts of this book when ordered in bulk quantities," and provides contact information for obtaining a copy of the book.

48. Page 3 of EX1120's PDF further includes an "ISBN" number, which I understand "uniquely identifies your book, and facilitates the sale of your book to bookstores (physical and digital) and libraries."[14]

49. According to the U.S. Library of Congress's online catalog, a book bearing the same name, author, publisher, and ISBN as EX1120 was "Published/Created" "c1986."[15]

50. I further note that EX1010 (Tormasov) – a U.S. Patent Application Publication that published on September 5, 2002 – includes a "BIBLIOGRAPHY" that cites the following, which has the same author, title, publisher, and ISBN number as shown in EX1120 and on the Library of Congress's online catalog. *See* EX1010, [0033].

51. I additional note that EX1086 – which is an issued U.S. Patent that issued on May 2, 1995 – discloses that "[t]he general description of the well-known operation of a UNIX operating system" discussed in EX1086 "is derived from Chapter 2 of the book entitled 'The Design of the UNIX Operating System'

[14] *See* https://www.isbn.org/ (last checked January 16, 2025).

[15] https://catalog.loc.gov/vwebv/holdingsInfo?searchId=22140&recPointer=0&recCount=25&searchType=0&bibId=1941862 (last checked January 16, 2025).

by Maurice J. Bach (EX1086, 2:58-61), which is the same title and author listed on the face of EX1120.

Based on the above evidence, I conclude that Bach would have also been publicly accessible, via (at least) its publisher Prentice-Hall or a library (e.g., the Library of Congress), to a POSA exercising reasonable diligence by 1987, or at least by September 5, 2002.

E. Public Accessibility of the Crowley Book (EX1121)

52. EX1121 (which I refer to herein as "Crowley") includes excerpts from a book titled "OPERATING SYSTEMS *A Design-Oriented Approach*," by Charles Crowley (University of New Mexico). *See* EX1121, pdf. page 1. Based on the evidence I discuss below, it is my opinion that Crowley is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

53. Page 1 of EX1121's PDF includes text of what appears to be a library stamp from the "J. ROBERT VAN PELT LIBRARY" "MICHIGAN TECHNOLOGY UNIVERSITY" "HOUGHTON, MICHIGAN."

54. Page 2 of EX1121's PDF bears the text "© Richard D. Irwin, a Times Mirror Higher Education Group, Inc., company, 1997" indicating it was published

and copyrighted by the Richard D. Irwin company in 1997 and thus publicly accessible by the year 1997. Page 2 of EX1121's PDF further includes an "ISBN" number, which I understand "uniquely identifies your book, and facilitates the sale of your book to bookstores (physical and digital) and libraries."[16]

55. Page 3 of EX1121's PDF bears a stamp with the date "APR 18 2002."

56. According to the U.S. Library of Congress's online catalog, a book bearing the same name, author, publisher, and ISBN number as EX1121 was "Published/Created" "1997."[17]

57. I further note that EX1010 (Tormasov) – a U.S. Patent Application Publication that published on September 5, 2002 – includes a "BIBLIOGRAPHY" that cites the following, which has the same author, title, publisher, and ISBN

---

[16] *See* https://www.isbn.org/ (last checked January 16, 2025).

[17] https://catalog.loc.gov/vwebv/search?searchArg=crowley+operating+systems++a+design-oriented+approach&searchCode=GKEY%5E*&searchType=1&limitTo=none&fromYear=&toYear=&limitTo=LOCA%3Dall&limitTo=PLAC%3Dall&limitTo=TYPE%3Dall&limitTo=LANG%3Dall&recCount=25 (last checked January 16, 2025).

number discussed above for EX1120 and on the Library of Congress's online catalog. *See* EX1010, [0034].

58. Based on the above evidence, I conclude that Bach would have also been publicly accessible, via (at least) its publisher Irwin or a library (e.g., the Library of Congress), to a POSA exercising reasonable diligence by 1997, or at least by September 5, 2002.

F. Public Accessibility of the Eckel Book (EX1122)

59. EX1122 (which I refer to herein as "Eckel") includes excerpts from a book titled "Building a Linux Internet Server," by George Eckel & Chris Hare. *See* EX1122, pdf. page 1. Based on the evidence I discuss below, it is my opinion that Eckel is prior art to the challenged patent (at least) because it was publicly accessible more than one year before the earliest priority date cited on the face of the challenged patent (September 15, 2002) by being disseminated or otherwise made available to POSAs exercising reasonable diligence.

60. Page 2 of EX1122's PDF (native page ii) bears the text "Copyright © 1995 by New Riders Publishing" indicating it was published and copyrighted by New Riders Publishing in 1995 and thus publicly accessible by the year 1995. Page 2 of EX1122's PDF (native page ii) further includes an "ISBN" number,

which I understand "uniquely identifies your book, and facilitates the sale of your book to bookstores (physical and digital) and libraries."[18]

61. I further note that EX1010 (Tormasov) – a U.S. Patent Application Publication that published on September 5, 2002 – includes a "BIBLIOGRAPHY" that cites the following, which has the same author, title, publisher, and ISBN number as shown in EX1120 and on the Library of Congress's online catalog. *See* EX1010, [0035].

62. As I discuss below, an article describing Eckel also appears to have been publicly accessible from the Linux Journal's website more than one year before the earliest possible priority date of the Asserted Patent. I understand from counsel that a POSA may use other publicly accessible information as "research aids" as a roadmap to locating additional prior art in the same or related field.

63. For example, the Linux Journal's website (as archived by the Internet Archive on May 8, 1999) describes itself as "the Premier Linux Magazine."[19] In my opinion, POSAs in the field of the challenged patent would have had reason to

[18] *See* https://www.isbn.org/ (last checked January 16, 2025).

[19] *See* https://web.archive.org/web/19990508005124/http://www.ssc.com/lj/ (last checked January 16, 2025).

navigate to the Linux Journal's website, to find information about publications or, in some cases, copies of publications relating to the field of the challenged patent.

64. Page 1 of EX1146 is the Internet Archive's archived capture of the Linux Journal's homepage, http://www.ssc.com/lj/, as it existed on May 8, 1999, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/19990508005124/http://www.ssc.com/lj/. On the left of Linux Journal's homepage, there is hyperlink text to "Previous Issues." Clicking that navigates to a webpage at http://www.ssc.com/lj/mags.html.

65. Pages 2-4 of EX1146 is the Internet Archive's archived capture of that webpage as it existed on April 18, 1999, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/19990418013008/http://www.ssc.com/lj/mags.html.

66. As shown on page 3 of EX1146, there is a hyperlink to "Issue #23, March 1996 (Focus: Linux Distributions Compared)," reachable by clicking on the "Issue #23, March 1996" text. Clicking on that text navigates to http://www.ssc.com/lj/issue23/index.html, which provides the "Table of Contents -- Linux Journal #23 -- March 1996." Pages 5-6 of EX1146 is the Internet Archive's archived capture of that webpage as it existed on April 24, 1999, as reflected in the Internet Archive's extended URL:

https://web.archive.org/web/19990424144757/http://www.ssc.com/lj/issue23/index.html.

67. On page 6 of EX1146, there is a hyperlink to "Book Review: Building a Linux Internet Server," reachable by clicking on the "Book Review:" text. Clicking on that text navigates to http://linuxjournal.com/issue23/blisrev.html.

68. Pages 7-8 of EX1146 is the Internet Archive's archived capture of that webpage as it existed on December 2, 1998, as reflected in the Internet Archive's extended URL: https://web.archive.org/web/19981202103832/http://linuxjournal.com/issue23/blisrev.html. Pages 7-8 of EX1146 are a "Book Review" of a book having the same title, authors, publisher, and ISBN number discussed above for EX1122.

69. Based on the above evidence, I conclude that Eckel would have also been publicly accessible, via (at least) its publisher New Riders Publishing, to a POSA exercising reasonable diligence by May 8, 1999, or at least by September 5, 2002 (the date of Tormasov's publication).