

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Liberty Mutual Insurance Company, Liberty Mutual Technology Group, Inc.,
Liberty Mutual Holding Company Inc., Liberty Mutual Group Inc., Liberty Mutual
Plano LLC, Comparison Insurance Agency, LLC, Ironshore Holdings (U.S.) Inc.,

and

Comerica Incorporated

Petitioners

v.

Intellectual Ventures II LLC

Patent Owner

IPR2025-00202
U.S. Patent No. 8,332,844

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 8,332,844**

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	GROUND FOR STANDING.....	2
III.	IDENTIFICATION OF CHALLENGE	2
IV.	THE 844 PATENT	3
	A. Overview	3
	B. Prosecution History	8
	1. The 477 Application.....	9
	2. The 622 Application.....	12
	C. Priority Date	13
	D. POSA14	
V.	CLAIM CONSTRUCTION	14
VI.	GROUND 1A: Claims 1-13 are obvious over <i>Sapuntzakis</i> combined with <i>Holzmann</i>	15
	A. Overview	15
	1. <i>Sapuntzakis</i> (EX1005).....	15
	2. <i>Holzmann</i> (EX1007)	19
	B. Motivation To Combine	20
	C. Detailed Analysis.....	22
VII.	GROUND 1B: Claims 14-27 are obvious over <i>Sapuntzakis</i> combined with <i>Federwisch</i>	37
	A. Overview	37
	1. <i>Sapuntzakis</i> (EX1005).....	37
	2. <i>Federwisch</i> (EX1008)	38
	B. Motivation To Combine	38
	C. Detailed Analysis.....	40
VIII.	GROUND 2A: Claims 14-13 are obvious over <i>Birse</i> combined with <i>Holzmann</i>	50
	A. Overview	50

1.	<i>Birse</i> (EX1006)	50
2.	<i>Holzmann</i> (EX1007)	52
B.	Motivation To Combine	53
C.	Detailed Analysis.....	54
IX.	GROUND 2B: Claims 14-27 are obvious over <i>Birse</i> combined with <i>Federwisch</i>	70
A.	Overview	70
1.	<i>Birse</i> (EX1006)	70
2.	<i>Federwisch</i> (EX1009)	70
B.	Motivation To Combine	70
C.	Detailed Analysis.....	71
X.	DISCRETIONARY DENIAL IS NOT APPROPRIATE	82
A.	35 U.S.C. § 325(d).....	82
B.	35 U.S.C § 314(a).....	83
XI.	MANDATORY NOTICES	85
A.	Real Party-in-Interest	85
B.	Related Matters.....	85
C.	Lead and Backup Counsel and Service Information.....	86
XII.	CONCLUSION.....	87

EXHIBIT LIST

Exhibit	Description	Short
1001	U.S. Patent No. 8,332,844 B1, entitled “Root Image Caching and Indexing for Block-Level Distributed Application Management,” issued 12/11/2012 to Kulkarni, et al.	844 Patent
1002	File History for U.S. Patent No. 8,332,844	844 FH
1003	Expert Declaration of Erez Zadok, Ph.D.	Decl.
1004	Erez Zadok, Ph.D. Curriculum Vitae	Zadok CV
1005	Sapuntzakis, et al. “Optimizing the Migration of Virtual Computers,” Association for Computing Machinery, Special Interest Group on Operating Systems, Operating Systems Review, Proceedings of the 5 th Symposium on Operating Systems Design and Implementation, Volume 36, Issue S1, published 12/31/2002	<i>Sapuntzakis</i>
1006	U.S. Patent No. 7,089,300 B1, entitled “Method and Apparatus for Administering the Operating System of a Net-Booted Environment,” filed on 10/18/1999, and issued 8/8/2006 to Birse, et al.	<i>Birse</i>
1007	U.S. Patent Application Publication No. 2005/0283597 A1, entitled “System and Method for Booting Multiple Servers from a Single Operating System Image,” filed on 6/22/2004, and published on 12/22/2005 (matured into U.S. Patent No. 8,601,100, issued 12/3/2013 to Holzmann).	<i>Holzmann</i>
1008	U.S. Patent Application Publication No. 2003/0182313 A1, entitled “System and Method for Determining Changes in Two Snapshots and for Transmitting Changes to Destination Snapshot,” filed on 3/19/2002, and published on 9/25/2003 (matured into U.S. Patent No. 6,993,539, issued 1/31/2006 to Federwisch, et al.).	<i>Federwisch</i>

Exhibit	Description	Short
1009	File History for U.S. Patent Application No. 11/026,622 (now abandoned)	622 FH
1010	File History for U.S. Patent Application No. 11/395,816 (now U.S. Patent No. 7,721,282)	816 FH
1011	Microsoft Press Computer Dictionary, 3rd ed., 1997 (excerpts)	Microsoft Computer Dictionary
1012	Abraham Silberschatz, et al., Operating Systems Concepts, 4th Edition, 1994, Addison-Wesley Publishing (excerpts)	Silberschatz
1013	Dave Hitz et al., “File System Design for an NFS File Server Appliance,” Network Appliance, USENIX 1994	Hitz
1014	Hugo Patterson et al., “SnapMirror®: File System Based Asynchronous Mirroring for Disaster Recovery,” Network Appliance, USENIX 2002	Patterson I
1015	U.S. Patent No. 6,668,264 B1, entitled “Resynchronization of a Target Volume with a Source Volume,” filed on 04/03/2001, and issued 12/23/2003 to Patterson, et al.	Patterson II
1016	U.S. Patent No. 7,334,095 B1, entitled “Writeable Clone of Read-Only Volume,” filed on 04/30/2004, and issued 2/19/2008 to Fair, et al.	Fair
1017	Internet Small Computer Systems Interface (iSCSI), RFC 3720, IETF, April 2004, https://www.rfc-editor.org/rfc/pdf/rfc3720.txt.pdf	iSCSI
1018	<i>Intellectual Ventures v. Liberty Mutual</i> , No. 23-cv-525 (E.D. Tex.) – Infringement Contentions	Liberty IC

Exhibit	Description	Short
1019	<i>Intellectual Ventures v. Comerica</i> , No. 23-cv-524 (E.D. Tex.) – Infringement Contentions	Comerica IC

Index of Challenged Claims

Claims
[1.0] A system for providing data to a plurality of compute nodes, the system comprising:
[1.1] a first storage unit configured to store blocks of a root image of said compute nodes;
[1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.
[2.0] The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.
[3.0] The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.
[4.0] The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.
[5.0] The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.
[6.0] The system as recited in claim 1 wherein said first storage unit is read-only.

Claims
[7.0] A method for providing data to a plurality of compute nodes, comprising:
[7.1] storing blocks of a root image of said compute nodes on a first storage unit;
[7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.
[8.0] The method as recited in claim 7 further comprising:
[8.1] receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and
[8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory.
[9.0] The method as recited in claim 8 further comprising: updating said cache memory based on said read request.
[10.0] The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises:
[10.1] caching said second portion in said cache memory; and
[10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.
[11.0] The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

Claims
[12.0] The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.
[13.0] The method as recited in claim 7 wherein said first storage unit is read-only.
[14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:
[14.1] a first storage unit configured to store blocks of a root image of said compute nodes;
[14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[14.3] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.
[15.0] The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.
[16.0] The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments

Claims
by merging the blocks of said root image with the blocks of respective leaf images.
[17.0] The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.
[18.0] The system as recited in claim 14 wherein said first storage unit is read-only.
[19.0] A method for indexing file systems for a plurality of compute nodes, comprising:
[19.1] storing blocks of a root image of said compute nodes on a first storage unit;
[19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;
[19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;
[19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and
[19.5] providing said indexing results to the others of said compute nodes.
[20.0] The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.
[21.0] The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

Claims
[22.0] The method as recited in claim 19 wherein said first storage unit is read-only.
[23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:
[23.1] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;
[23.2] index said root image portion; and
[23.3] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.
[24.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.
[25.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.
[26.0] The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.
[27.0] The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

I. INTRODUCTION

Liberty Mutual Insurance Company, Liberty Mutual Technology Group, Inc., Liberty Mutual Holding Company Inc., Liberty Mutual Group Inc., Liberty Mutual Plano LLC, Comparion Insurance Agency, LLC, Ironshore Holdings (U.S.) Inc. (collectively, “LM”), and Comerica Incorporated (“Comerica”) (collectively, “Petitioners”) petition for *inter partes* review (“IPR”) of claims 1-27 (all claims) (the “challenged claims”) of U.S. Patent No. 8,332,844, entitled “Root Image Caching and Indexing for Block-Level Distributed Application Management” (“the 844 patent,” EX1001).¹

The 844 patent describes technology for “root image caching and indexing for block-level distributed application management.” EX1001, 2:36-38. However, the references to caching and indexing were merely added to differentiate the 844 patent from two prior parent applications (that mention neither indexing or caching, and to which the 844 patent improperly claims priority). Instead, the 844 patent disclosure is largely directed to systems/methods for distributing an application environment to a compute node using a “root/leaf” system of application environment storage (“storing blocks of a root image on a first storage unit and storing blocks of leaf

¹ The Office is authorized to charge fees to Deposit Account 50-3725.

images on respective second storage units”) – to speed creation of and access to boot images. *Id.* 2:38-40, 51-55. As the prosecution history makes clear, the asserted novelty of the challenged claims was limiting the leaf images to “additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image.” *Id.* 2:39-42. However, that architecture was well-known before December 30, 2004, and certainly before February 21, 2007 (the 844 patent’s earliest possible priority date).

Petitioners, supported by the declaration of Dr. Erez Zadok (who has over 30 years of experience), demonstrate that all 27 claims are unpatentable.

II. GROUNDS FOR STANDING

Petitioners certify the 844 patent is available for IPR, and they are not barred or estopped from requesting IPR of the challenged claims on the grounds below.

III. IDENTIFICATION OF CHALLENGE

Petitioners request IPR and cancellation of all claims (claims 1-27) of the 844 patent on the following grounds:

Ground	Challenged Claims	Basis for Rejection
1A	1-13	§103: <i>Sapuntzakis</i> and <i>Holzmann</i> (U.S. 2005/0283597)
1B	14-27	§103: <i>Sapuntzakis</i> and <i>Federwisch</i> (U.S. 2003/0182313)
2A	1-13	§103: <i>Birse</i> (U.S. 7,089,300) and <i>Holzmann</i>

Ground	Challenged Claims	Basis for Rejection
2B	14-27	§103: <i>Birse</i> and <i>Federwisch</i>

The 844 patent is **not** entitled to a 12/30/2004 or 3/30/2006 priority date, but every prior art reference in the above grounds precedes those dates.

Reference	Prior Art Date	Status
<i>Sapuntzakis</i> (EX1005)	Published: 12/31/2002	§ 102(b)
<i>Birse</i> (EX1006)	Filed: 10/19/1999 Issued: 8/8/2006	§ 102(e)
<i>Holzmann</i> (EX1007)	Filed: 6/22/2004 Published: 12/22/2005 Issued: 12/3/2013 (US 8,601,100)	§ 102(b) § 102(e)
<i>Federwisch</i> (EX1008)	Filed: 3/19/2002 Published: 9/25/2003 Issued: 1/31/2006 (US 6,993,539)	§ 102(b) § 102(e)

IV. THE 844 PATENT

A. Overview

The 844 patent initially describes clustered computing involving multiple compute nodes with Operating System (“OS”) software image management, where nodes are activated by diskless booting protocols or remote software installation to local storage, and where a boot image is required for each node in the cluster – and states that storing a boot image for each server results in wasted storage, and copying the entire master image at boot results in long boot times. EX1001, 1:31-67. The

Background also discusses branching-store file systems where a read-only “root” image is accessible by all nodes, and changes made by a node to the root image are stored in a “leaf” image unique to that node – with a filter operating between the nodes and the OS file system(s) that merges the leaf images with the root image and delivers the result to the appropriate node (which the node views as its own cohesive instance of the application environment). *Id.* 2:13-24.

The 844 patent’s system, as shown in Fig. 1, includes a general purpose computing system environment, such as compute node **100**, which includes at least one processing unit **102** and memory **104**). *Id.* 4:27-36. Nodes may also have additional features, including additional removable **108** and/or non-removable **110** storage, communications connections **112**, input devices **114**, and output devices **116**. *Id.* 4:36-5:10. Memory **104**, removable storage **108**, and non-removable storage **110** are all computer storage media, which includes “any...medium which can be used to store the desired information and which can be accessed by compute node **100**.” *Id.* 4:46-55.

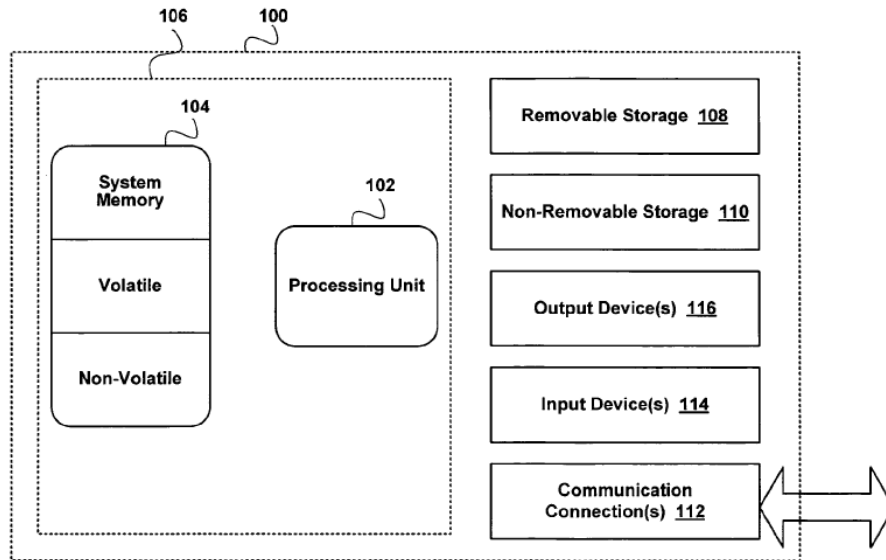
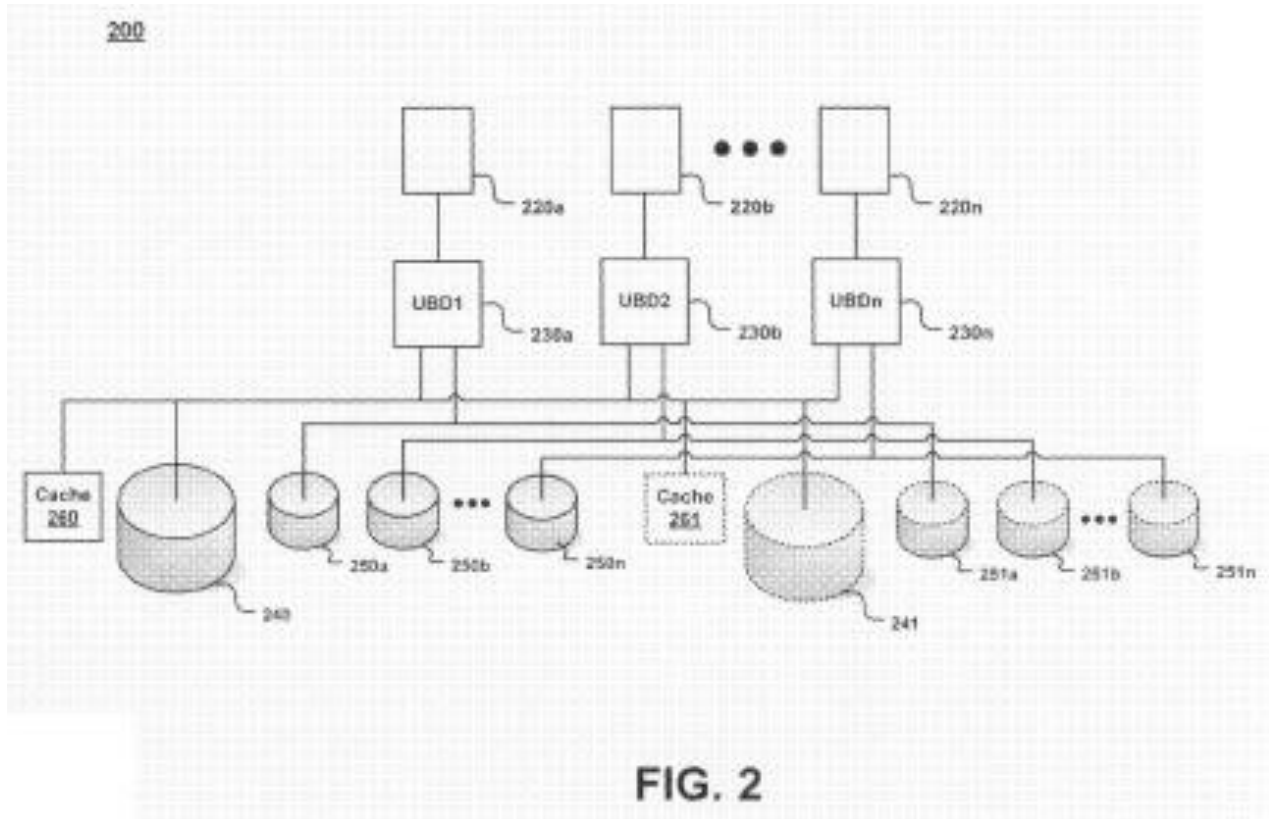


FIG. 1

844 Patent, Figure 1

Fig. 2 diagrams system 200 (*e.g.*, a multi-computer system) having a number of compute nodes 220a-n coupled to a first storage unit 240 and a corresponding second storage unit 250a-n through a corresponding union block device (“UBD”) 230a-n. *Id.* 5:12-22. To the compute nodes 220a-n, it appears that they have access to their own version of the application environment. However, a separate boot image is not created and stored for each node. *Id.* 5:22-2



844 Patent, Figure 2

Instead, a first storage unit **240** stores blocks of the root image. The root image contains data initially common to the compute nodes **220a-n**, and is not changed by them. *Id.* 5:27-32. Each compute node **220a-n** also has access to a corresponding second storage unit **250a-n** for storing a leaf image, which in the challenged claims includes “only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image...” or similar language. *Id.* 5:33-34, 10:61-65, 11:32-35, 12:4-8, 12:47-50. The first storage unit **240** and second storage units **250a-n** may each be contained

on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof. *Id.* 5:34-38.

A compute node **220a-n** mounts its instantiation of the application environment via its respective UBD **230a-n** which, in one embodiment, are “effectively low level drivers that operate as an interface between the first and second storage devices and the file system of each compute node **220a-n**.” *Id.* 5:59-64. UBDs **230a-n** determine what “leaf image” (from the appropriate second storage unit **250**) is needed for portions of the image that their respective compute nodes **220a-n** have changed, and locate the portions of the image not changed by their respective node (which may reside in the root image or intermediate images comprising versions of the root image, not depicted in Fig. 2). *Id.* 6:4-13. UBDs **230a-n** also create a new leaf image on a respective second storage unit **250a-n** when their respective compute nodes **220a-n** make changes to their instantiations of the image (and may also modify the leaf image in response to their respective compute node’s access to its instance of the image). *Id.* 6:13-22.

The blocks of the root image, leaf image, and cache (discussed below) are merged “on the fly” to create the node’s image. *Id.* 8:13-15. From the node’s perspective, the created image appears as one cohesive image even though a separate

and complete image is not actually stored for the node. *Id.* 8:18-55. This saves space without significantly impacting load times. *See Id.* 2:51-57.

The 844 patent also claims to further improve speed by caching portions of the root image in a cache **260** (as several nodes in a cluster may often access the same root image data). *Id.* 2:59-61, 6:38-43. This may shorten times to load blocks by as much as 90%. *Id.* 6:43-46. Because cache capacity is typically limited, older data can be removed (as the amount approaches a threshold) to make way for new data by techniques “known in the art.” *Id.* 7:1-16.

The 844 patent also provides an “added benefit” by which one compute node can index the contents of the (shared) root image and provide the results to the other compute nodes (*e.g.*, by storing results on a shared storage unit or providing the results directly). *Id.* 7:23-31. *See also*, Decl. ¶157-203.

B. Prosecution History

Application No. 11/709,477 (“the 477 Application”), which matured into the 844 patent, was a continuation-in-part of Application No. 11/395,816 (now Patent No. 7,721,282) (“the 816 Application”), which was a continuation-in-part of Application No. 11/026,622 (now abandoned) (“the 622 Application”).

1. The 477 Application

The 477 Application's five independent claims were originally open-ended concerning the content of leaf images – reciting they comprised “additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image” or similar language. EX1002, 25-30. Claims 14-27 were rejected (1/5/2010) as anticipated by Publication No. 2003/0126242 A1 (“Chang”). Claims 1-13 (all others) were rejected as obvious over Chang and Patent No. 6,101,576 (“Kobayashi”). Kobayashi was cited only for caches; and the Examiner found it would have been obvious to a person of ordinary skill in the art (“POSA”) to have combined the “detailed image/data cache” of Kobayashi with the “server/client image assess and remotely stored boot/OS system” of Chang “for the benefit of increasing the speed of data transfers from the host to client in data streaming or transfer operations as frequently accessed data is kept in [cache] memory...[versus] slower disk sources.” *Id.* 53-64.

In response (4/5/2010), Applicants amended the independent claims to all include a “wherein” limitation of the general form:

7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:
storing blocks of a root image of said compute nodes on a first storage unit;
storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

Id. 73-78. Applicants stated this overcame the rejections. *Id.* 82-88. The findings regarding Kobayashi's cache and motivation to combine were not contested. Nor were any dependent limitations raised. *Id.*

The amended claims were rejected (5/25/2010) on the same grounds (with additional citations to Chang addressing the "wherein" amendments). *Id.* 102-25.

Applicants then further amended the independent claims (8/31/2010) to all include the term "only" in the following manner:

7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:
storing blocks of a root image of said compute nodes on a first storage unit;
storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

Id. 166-71. Applicants stated these amendments overcame the rejections because the “client image copies” in Chang (leaf images) include “blocks of the root image that are unchanged by a compute node (*e.g.*, the common OS and application blocks for storing the base boot image).” *Id.* 176-82. Again, Applicants never contested Kobayashi’s cache or motivation to combine. Nor were any dependent limitations raised.² *Id.*

The third Office Action (3/12/2012) never mentioned Chang.³ Instead, the Examiner rejected claims 19-27 as anticipated by Patent No. 7,870,106 (“Nguyen”), claims 14-18 as obvious over Nguyen, and claims 1-13 as obvious over Nguyen and Kobayashi. Again, Kobayashi was only cited for caches, and the Examiner found it would have been obvious to a POSA to have combined the “detailed image/data

² Applicants also stated a UBD is “a driver,” that “drivers serving as interfaces between [OS] file systems...and associated storage devices are well known in the art;” and that a POSA “would understand how a driver may serve as [such] an interface.” *Id.* 172-73.

³ Applicants’ prior arguments were found “moot,” as they did not apply to the current references. *Id.* 206.

cache” teachings of Kobayashi with the “server/client image assess and remotely stored boot/OS system” teachings of Nguyen. *Id.* 192-206.

On 7/11/2012, Applicants traversed these rejections, stating Nguyen was not prior art under § 103(a) given alleged common ownership with the pending application. *Id.* 224, 300. Applicants also amended several claims, including substitution of “including” for “comprising” in two independent claims. *Id.* 239-45. Again, Applicants did not contest findings regarding the cache of Kobayashi or the motivation to combine. Nor did they raise any dependent limitations. *Id.* 221-25.

Thus, the sole purported point of novelty for the 844 patent claims is the limitation, in each independent claim, that the leaf images only comprise/include “additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image” (or corresponding language in claim 23).

2. The 622 Application

During prosecution of the “grandparent” 622 Application, a different Examiner repeatedly rejected all proposed claims as anticipated (4/13/2009) by Chang and obvious (8/24/2009, 3/15/2010, 8/13/2010, 12/6/2010) over Chang and Patent No. 6,751,658 (“Haun”). EX1009, 50-63, 92-109, 144-167, 203-25, 259-85. Indeed, Applicants (who differed, in part, from the 477 Applicants) ultimately

abandoned the 622 Application with no claims ever allowed. *Id.* 294-95. This, despite the same attorney (at the same time) offering similar amendments/arguments during prosecution of both the 622 and 477 Applications (including addition of the word “only” to several claims).⁴ *Id.* 183-99.

Further, while the 622 and 477 Applications were co-pending, the 477 Application Examiner (unlike the 622 Application Examiner) never relied on Haun. EX1002. Indeed, the former mentioned Haun only once (1/5/2010), and then only to state that Haun (and another reference) were prior art of record and were “not relied upon but is considered pertinent to applicant’s disclosure.” *Id.* 64.

C. Priority Date

The 477 Application was filed on February 21, 2007, and claimed priority to the 816 Application (filed 3/30/2006), which claimed priority to the 622 Application (filed 12/20/2004). However, none of the 844 patent claims are entitled to either priority date, as the 816 and 622 Applications **never** mention caching or indexing –

⁴ For example, ‘622 Application claim 1 was amended (on 6/15/2010) to recite a “plurality of second file collections separate from the root file system, each comprising only modified portions of said distributed applications for respective ones of said plurality of compute nodes...” *Id.* 184.

at least one of which is expressly recited in each challenged claim. EX1001, 10:55-14:17; compare EX1001 6:37-7:44, Figs. 2, 3A, 3B, 7 with EX1009 Figs. 1, 7 and EX1010 Figs. 1, 3. See *Studiengesellschaft Kohle v. Shell Oil Co.*, 112 F.3d 1561, 1564 (Fed. Cir. 1997). Applicants even admitted that “indexing and indexing results are not discussed anywhere within Nguyen” (which matured from a continuation-in-part of the 622 Application). EX1002, 223.

D. POSA

A POSA for the 844 patent, at the time of the claimed invention, would have had: (1) a bachelor’s degree in computer science, computer engineering, or an equivalent degree; and (2) at least two years of experience working in the field of networked data storage, file systems, and operating systems. Additional education could substitute for less experience and *vice versa*. Decl. ¶34-36.

Petitioner’s independent expert, Dr. Zadok, was qualified as a POSA on the earliest possible priority date, and remains qualified to testify about what a POSA would have understood. *Id.* 36; EX1004.

V. CLAIM CONSTRUCTION

Claim terms are construed according to the *Phillips* standard in an IPR. *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005); 37 C.F.R. §42.100. The analysis in Sections VI-IX establishes that the challenged claims are obvious under

any reasonable interpretation under the *Phillips* standard, and thus Petitioners submit the Board need not resolve any express constructions.⁵ *See, e.g., Toyota Motor Corp. v. Cellport Systems, Inc.*, IPR2015-00633, Paper 11 at 16 (08/14/2015) (citing *Wellman, Inc. v. Eastman Chemical Co.*, 642 F.3d 1355, 1361 (Fed. Cir. 2011)) (claim construction only required where necessary to resolve dispute). Should any subsequent discussion of claim interpretation arise here or in the parallel district court proceeding, Petitioners reserve the right to propose constructions and respond to any construction advanced by the Patent Owner, the Board, and/or the Court.

VI. GROUND 1A: Claims 1-13 are obvious over *Sapuntzakis* combined with *Holzmann*

A. Overview

1. *Sapuntzakis* (EX1005)

Sapuntzakis concerns system architecture for quickly moving the state of a running computer (including its OS, applications and user files on disk, and all data in memory) – called a “capsule” – across a network to another machine. EX1005, 2. This is achieved by reducing the amount of data sent over the network through use

⁵ Patent Owner very broadly reads the claims asserted in the litigation against commonly and widely used open-source software known as Docker/Kubernetes “containers.” *See* EX1018-EX1019.

several mechanisms. Of interest here is the use of “copy-on-write” (“COW”) disk images that track just the updates to the disk-storage component of capsules. *Id.*

Computers and storage act as caches of capsules. As users travel (commute), the system can move their capsules to computers close to them, giving them a consistent environment. *Id.* 3. As capsules are mostly derived from other capsules, and thus differences between related capsules are small relative to their total size, disks in those capsules can be stored efficiently by creating a hierarchy where each child capsule can be viewed as inheriting from its parent – with the differences in disk state between them captured in a separate COW virtual disk. *Id.* 6.

At the hierarchy base is a root disk, which is a complete capsule disk. For example, administrators can install and maintain the same software set on multiple machines by creating one (inactive) capsule and distributing it to all machines. *Id.* 6. This approach shares similarities with disk imaging, but the disclosed system allows multiple capsules to exist on the same machine. *Id.* 5

All other nodes of the hierarchy represent a COW disk. Each path of COW disks originating from the root in the capsule hierarchy represents a capsule disk. Figure 1 shows an example of a capsule hierarchy (for a university) where the root capsule contains all software available to students, departments may extend the basic

capsules with department-specific software, and student files are stored on networked storage servers. *Id.* 6.

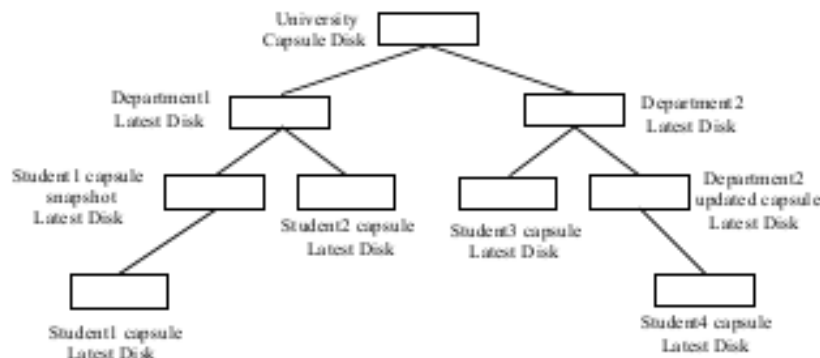


Figure 1: An example capsule hierarchy.

Sapuntzakis, Figure 1

Each COW disk is implemented as a bitmap file and a sequence of extent files (a sequence of blocks of the COW disk). The bitmap file contains one bit for each 16KB block on the disk, indicating whether the block is present in the COW disk. Writes to a capsule disk are performed by writing the data to the latest COW disk and updating its bitmap file. Reads involve searching the latest COW disk and its ancestor disks in turn until the required block is found. *Id.* 6. Because the root COW disk contains a copy of all blocks, the search is guaranteed to terminate. *Id.* 6-7.

Figure 2 shows an example capsule disk and the chain of COW disks that comprise the capsule disk. Notably, none of the COW disks (from “COW Disk 1” to the “Latest COW Disk”) contain any unchanged blocks of the root disk. Nor is

the COW disk hierarchy visible to the user, as they see a normal flat disk as illustrated in the figure. *Id.* 7

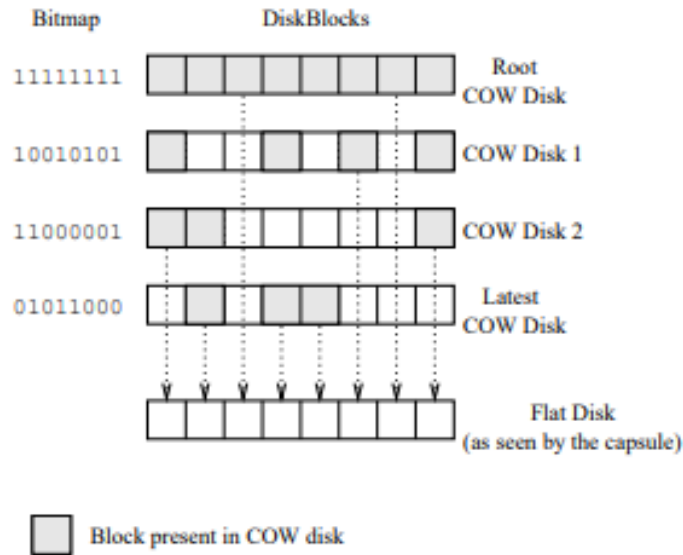


Figure 2: An example capsule disk.

Sapuntzakis, Figure 2

To further reduce the capsule start-up time, the COW disks are read page-by-page on demand, rather than being pre-fetched. Demand paging is useful because COW disks can be extremely large; and, during a typical user session, the working set of disk blocks needed is a small fraction of the total blocks on the disk. *Id.* 7

The implementation of the system, including demand paging, is shown in Figure 3. Each COW disk can be local or remote, with each remote COW disk having a corresponding local shadow COW disk that contains all locally cached blocks of the remote COW disk. Because the latest COW disk is local, all writes are local.

Reads can be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW. *Id.* 7. *See also*, Decl. ¶224-42.

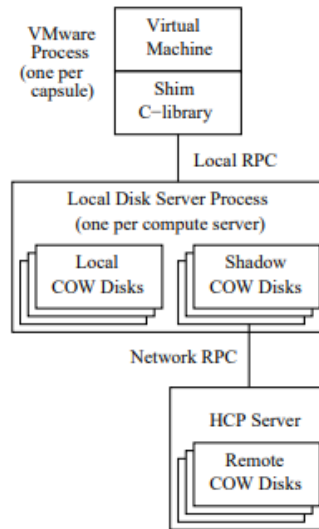


Figure 3: Implementation of capsule disks and demand paging.

Sapuntzakis, Figure 3

2. *Holzmann* (EX1007)

Holzmann is directed to a system and method for booting multiple servers or other external devices from a storage area network using a single OS image. EX1007 [0017]. In Figure 1, a solid state disk system 101 is disclosed having an internal architecture of component devices combined to achieve its purposes. *Id.* [0017-20]

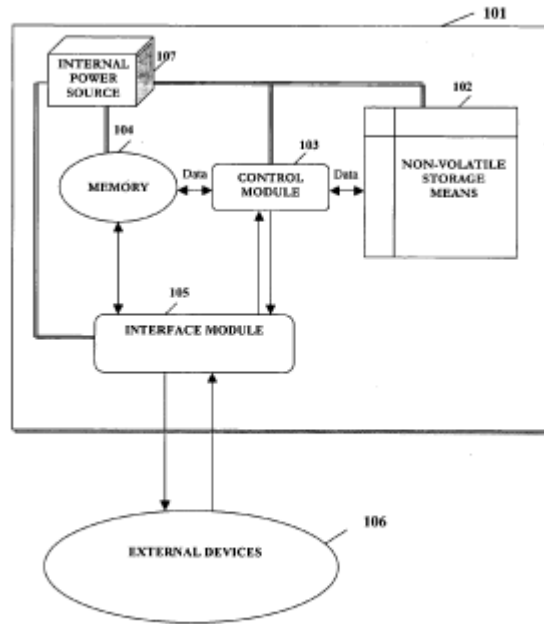


Fig. 1

Holzmann, Figure 1

When a boot request is received from an external device, the control module **103** allocates cache space in the volatile cache memory portion **104** of the disk. The cache is used in the typical manner to optimize future access to data once it has demonstrated some likelihood of being further referenced. As additional servers or other network resources are booted, the cache is used to speed access to the most commonly used data. *Id.* Abstract, [0021], [0024]. *See also*, Decl. ¶243-56.

B. Motivation To Combine

The Supreme Court in *KSR International Co. v. Teleflex Inc.*, 550 U.S. 398 (2007) adopted a flexible approach to evaluating whether a patent claiming a combination of elements of prior art is obvious. Under *KSR* and as detailed in the

Zadok Declaration (EX1003), a POSA would have been motivated to combine *Sapuntzakis* and *Holzmann*. Both are concerned with the management of whole OS images being shared by users on a network, particularly efficient storage/transfer of such images. Both disclose caching blocks of the root/boot disk images to improve efficiency. Further, a POSA would have been motivated to modify *Sapuntzakis* with *Holzmann*'s caching techniques, including because, while *Sapuntzakis* discloses caching, a POSA would have known there are many different caching techniques with associated benefits and drawbacks (several disclosed in *Holzmann*) – and *Holzmann* details how to cache data for each OS (*e.g.*, *Sapuntzakis*'s virtual machine). Moreover, caching data from comparatively slow storage devices in faster devices (including solid-state memory) was a well-known technique commonly used for decades before the 844 patent. Modifying *Sapuntzakis* with *Holzmann*'s caching teachings would have been simple to a POSA, and would not have required extensive experimentation or imposed undue burden – especially when both disclose caching. For the same reasons, the combination would have yielded predictable results (as caching is well-understood to improve performance and resource utilization). A POSA would also have a high expectation of success, as *Holzmann* merely describes caching in greater detail than *Sapuntzakis*. *See* Decl. ¶¶331-50.

C. Detailed Analysis

Claims 1-13 are obvious as stated herein, the table below, and the Zadok Declaration (EX1003, ¶¶309-412). Notably, *Sapuntzakis* discloses the claimed root/leaf architecture, including where leaf images only contain additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image. *Holzmann* discloses any additional caching limitations to the extent, if any, not already taught by *Sapuntzakis* and/or obvious to a POSA.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
[7.0] A method for providing data to a plurality of compute nodes, comprising:	<i>Sapuntzakis</i> discloses a method for providing data (capsules) to a plurality of compute nodes (user computers): “Computers and storage in the Collective system act as caches of capsules. As users travel, the Collective can move their capsules to computers close to them, giving users a consistent environment. Capsules could be moved with users as they commute between home and work. Capsules can be duplicated, distributed to many different machines, and updated like any other data; this can form the basis for administering a group of computers.” <i>Sapuntzakis</i> at 2.
[7.1] storing blocks of a root image of said compute nodes on a first storage unit;	<i>Sapuntzakis</i> discloses storing blocks of a root image (root disk) of said compute nodes (computers) on a first storage unit (remote COW disks on HCP server of Figure 3): “At the root of the hierarchy is a <i>root disk</i>, which is a complete capsule disk.” <i>Sapuntzakis</i> at 5 “Figure 1 shows an example of a capsule hierarchy illustrating how it may be used in a university. The root

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	capsule contains all the software available to all students.” <i>Sapuntzakis</i> at 5 <pre> graph TD A[University Capsule Disk] --> B[Department1 Latest Disk] A --> C[Department2 Latest Disk] B --> D[Student1 capsule snapshot Latest Disk] B --> E[Student2 capsule Latest Disk] C --> F[Student3 capsule Latest Disk] C --> G[Department2 updated capsule Latest Disk] D --> H[Student1 capsule Latest Disk] G --> I[Student4 capsule Latest Disk] </pre> Figure 1: An example capsule hierarchy. <pre> graph TD subgraph VMware_Process [VMware Process one per capsule] VM[Virtual Machine] Shim[Shim C-library] end subgraph Local_Disk_Server_Process [Local Disk Server Process one per compute server] Local_COW[Local COW Disks] Shadow_COW[Shadow COW Disks] end subgraph HCP_Server [HCP Server] Remote_COW[Remote COW Disks] end VMware_Process -- Local RPC --> Local_Disk_Server_Process Shadow_COW -- Network RPC --> HCP_Server </pre> Figure 3: Implementation of capsule disks and demand paging. A POSA would have understood that the remote COW Disk images would have been stored on a first storage unit of the server. Decl. ¶312.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
[7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and	<i>Sapuntzakis</i> discloses storing leaf images (the local COW disk images of Figure 3) for respective compute nodes on respective second storage units (attached either to the HPC server or the client computer), said leaf images including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written): “We can store the disks in these capsules efficiently by creating a hierarchy, where each child capsule could be viewed as inheriting from the parent capsule with the differences in disk state between parent and child captured in a separate copy-on-write (COW) virtual disk.” <i>Sapuntzakis</i> at 5. The local COW disk images holding the leaf images would have been stored on respective second storage units attached to the (local) client computers: “Writes to a capsule disk are performed by writing the data to the latest COW disk and updating its bitmap file. Reads involve searching the latest COW disk and its ancestor disks in turn until the required block is found. Since the root COW disk contains a copy of all the blocks, the search is guaranteed to terminate.” <i>Sapuntzakis</i> at 5–6 “Since the latest COW disk is always local, all writes are local.” <i>Sapuntzakis</i> at 6 A POSA would have understood that, because writes to the local (latest) COW disk are described to occur only in these circumstances, the local COW disk will including

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	only blocks according to the limitations of this term and no others. Decl. ¶316
[7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.	<i>Sapuntzakis</i> and <i>Holzmann</i>, disclose caching blocks of said root image (the remote COW disks) that have been accessed by at least one of said compute nodes in a cache memory (the shadow COW disks): <u>Sapuntzakis:</u> “Each COW disk can either be local or remote. Each remote COW disk has a corresponding local <i>shadow COW</i> disk which contains all the locally cached blocks of the remote COW disk.” <i>Sapuntzakis</i> at 6 “Reads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW.” <i>Sapuntzakis</i> at 6 “First, the memory in most systems caches disk blocks.” <i>Sapuntzakis</i> at 7. <u>Holzmann:</u> “...[T]he invention [of <i>Holzmann</i>] is directed to a system and method for booting multiple servers using a single operating system image located on a solid state disk in a storage area network.” <i>Holzmann</i> [0002].

(While referred to as servers in Holzmann, a POSA would have understood these to correspond to the compute nodes of the 844 Patent and the NC clients of Birse) Decl. ¶322.

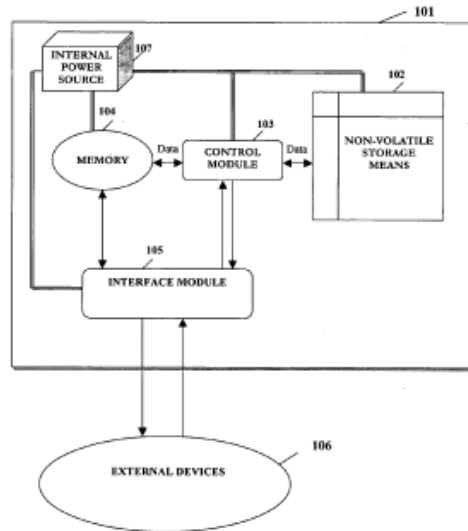


Fig. 1

“FIG. 1 is a block diagram illustrating the component parts of the invented system and the functions of each component part. The invented system comprises a solid state disk system **101** having a storage means **102**, a control module **103**, a memory module **104**, an interface module **105** that communicates with external devices **106**, and an internal power source **107**.” *Id.* [0017].

“The memory module **104** comprises at least one direct-access memory module for holding data currently or recently used by external devices **106**. The memory module **104** is more quickly accessible, and performs read and write processing functions more quickly, than non-volatile or disk memory, such as storage means **102**. The memory module **104** preferably comprises at least one random-access memory (RAM) module.” *Id.* [0020].

A POSA would have recognized this description as describing a cache memory. Decl. ¶325.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	“Upon receiving a boot request, the control module 103 loads the required portion of the operating system image (the portion loaded typically depends on the operating system and its boot process) into memory module 104.... Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically...” <i>Id.</i> [0021]. “The external device 106 requests operating system data through the interface module 105....[T]he control module 103 monitors the request for operating system data, and determines whether the particular operating system data block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that data block has not been overwritten...the requested data block is provided from the cache...” <i>Id.</i> [0024]. “If a second server or other external device sends a boot request to control module 103, the module allocates additional space in the cache for a second cache 402....If additional servers require boot operations, additional cache space 403 can [be] allocated for those servers. By allocating separate cache space for each server or other network resource, the inventive system prevents corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction.” <i>Id.</i> [0033]. A POSA would have understood that, when the cache of Holzmann is used to cache blocks of a read-only shared OS image like Birse 660 there is no need to create separate

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	caches for each client because the read-only image is not subject to corruption. Decl. ¶329. “If the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501.” <i>Id.</i> [0034-35].
[8.0] The method as recited in claim 7 further comprising:	<i>See</i> [7]. ⁶
[8.1] receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and	<i>Sapuntzakis</i> and <i>Holzmann</i> disclose receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory (the shadow COW disks): <u>Sapuntzakis:</u> “Reads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW.” <i>Sapuntzakis</i> at 6 <u>Holzmann:</u> “[T]he <u>control module</u> 103 monitors the request for operating system data, and determines whether the...data block requested...has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and...has not been

⁶ Unless stated otherwise, all citations of this form in this and other chart are to other rows in the same chart.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	overwritten....[it] is provided from the cache...” <i>Holzmann</i> [0024]. See also Decl ¶354.
[8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory.	<i>Sapuntzakis</i> and <i>Holzmann</i> disclose providing said first portion of said data to said at least one of said compute nodes from said cache memory (the shadow COW disks): <u>Sapuntzakis:</u> “Reads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW.” <i>Sapuntzakis</i> at 6. A POSA would have understood from this passage that if the requested blocks <i>are</i> cached, they are served from the cache. Decl. ¶358. <u>Holzmann:</u> See [8.1].
[9.0] The method as recited in claim 8 further comprising: updating said cache memory based on said read request.	See [8.0]-[8.2]. <i>Sapuntzakis</i> and <i>Holzmann</i> further discloses updating said cache memory (the shadow COW disks) based on said read request: <u>Sapuntzakis:</u> “Reads, on the other hand, could either be local or remote. In the case of a remote read....If the block is not cached locally, it is fetched remotely and added to the shadow COW.” <i>Sapuntzakis</i> at 6. <u>Holzmann:</u> “If the requested data has not been loaded into the cache 501, the control module 103 copies the data from ... memory portion 400 to the cache 501.” <i>Holzmann</i>, [0035].

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	<i>See also</i> , Decl. ¶360-63.
[10.0] The method as recited in claim 9	<i>See</i> [9.0].
[10.1] wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and	<i>Sapuntzakis</i> and <i>Holzmann</i> disclose processing a request wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory: <i>See</i> Decl. ¶368 (“[This] describes nothing more than common caching behavior (updating a cache based on reading a new item, called a “cache admission” policy)”). <i>See also</i> Decl. ¶ 371. <u>Sapuntzakis:</u> <i>See</i> [9.0]. <u>Holzmann:</u> <i>See</i> [8.1], [9.0].
[10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.	<i>Sapuntzakis</i> and <i>Holzmann</i> disclose removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value: <u>Sapuntzakis:</u> “On a miss, the cache adds the entry to the bucket, possibly evicting an existing entry. ... When adding an entry to the cache, the hash cache chooses a fraction of the entries at random from the bucket and replaces the entry with the lowest use count” <i>Sapuntzakis</i> at 8. <u>Holzmann:</u>

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	“It will be understood by those skilled in the art that still more cache configurations can be used in accordance with the invented system. <i>Holzmann</i> [0038]. A POSA would have understood this limitation describes a standard “cache eviction” policy. Decl. ¶372-76.
[11.0] The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.	See [7]. <i>Sapuntzakis</i> further discloses merging the blocks of said root image (the root COW disk) with the blocks of respective leaf images (the local COW disk and ancestors between it and the root) to create cohesive respective application environments: “To reduce the start-up time of a capsule, the COW disks corresponding to a capsule disk are read page-by-page on demand, rather than being pre-fetched.” <i>Sapuntzakis</i> at 6. “Reads involve searching the latest COW disk [the leaf image] and its ancestor disks in turn until the required block is found.” <i>Sapuntzakis</i> at 5-6. Figure 2: An example capsule disk. “Note that the COW disk hierarchy is not visible to the VMM, or to the OS and applications inside the

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	capsule; all of them see a normal flat disk as illustrated in the figure.” <i>Sapuntzakis</i> at 6. A POSA would have understood this to mean that the COW disk hierarchy is also invisible to the client, which perceives, instead, a single cohesive storage image corresponding to the 844 Patent’s merged root and leaf blocks. Decl. ¶382.
[12.0] The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.	See [11.0]. <i>Sapuntzakis</i> and <i>Holzmann</i> disclose said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit (the root COW disks), respective second storage units (the local COW disks), and said cache memory (the shadow COW disks). “The COW disk implementation interfaces with GSX Server through a shim library....The shim library intercepts GSX Server’s I/O requests to disk image files...and redirects them to a local disk server.” <i>Sapuntzakis</i> at 6. “The local disk server translates these requests to COW disk requests, and executes the I/O operations against the COW disks.” <i>Sapuntzakis</i> at 6. “The implementation of the capsule disk system, including demand paging, is shown in Figure 3. The shim library intercepts all of VMware’s accesses to plain disks and forwards them to a disk server on the local machine. The disk server performs a translation from a plain disk access to the corresponding access on the COW disks of the capsule.” <i>Sapuntzakis</i> at 6. A POSA would have understood the references to the COW disks in these quotes refer to all three of: the root COW disks, the local (leaf) COW disks, and the shadow (cache) COW disks. Decl. ¶388.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
[13.0] The method as recited in claim 7 wherein said first storage unit is read-only.	See [7]. <i>Sapuntzakis</i> further discloses the first storage unit (storing root disk images and possibly certain intermediate COW disks) being read-only. “At the root of the hierarchy is a root disk, which is a complete capsule disk. All other nodes represent a COW disk. Each path of COW disks originating from the root in the capsule hierarchy represents a capsule disk; the COW disk at the end of the path for a capsule disk is its latest disk. We cannot directly run a capsule whose latest disk is not a leaf of the hierarchy. We must first derive a new child capsule by adding a new child disk to the latest disk and all updates are made to the new disk. Thus, once capsules have children, they become immutable; this property simplifies the caching of capsules.” <i>Sapuntzakis</i> at 6 (describing Figure 1). A POSA would have understood that capsules with children are immutable and cannot be run directly because they are read-only. Decl. ¶392.
[1.0] A system for providing data to a plurality of compute nodes, the system comprising:	See [7.0].
[1.1] a first storage unit configured to store blocks of a root image of said compute nodes;	See [7.1].
[1.2] a plurality of second storage	See [7.2].

Claims	Ground 1A <i>Sapuntzakis (EX1005)/Holzmann (EX1007)</i>
units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and	
[1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.	See [7.3].

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
[2.0] The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.	See [1]. <i>Sapuntzakis</i> and <i>Holzmann</i> further discloses said cache being configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value. <u>Sapuntzakis:</u> “On a miss, the cache adds the entry to the bucket, possibly evicting an existing entry. Each entry contains a use count that the cache increments on every hit. When adding an entry to the cache, the hash cache chooses a fraction of the entries at random from the bucket and replaces the entry with the lowest use count; this evicts the least used and hopefully least useful entry of the group.” <i>Spuntzakis</i> at 8. <u>Holzmann:</u> See [9.0], [10.2]. A POSA would have understood this limitation describes a standard “cache eviction” policy. Decl. ¶¶399-403.
[3.0] The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.	See [1]. <i>Sapuntzakis</i> and <i>Holzmann</i> further disclose said first storage unit, said second storage units, and said cache being contained within a single storage appliance. A POSA would have understood that, as to <i>Sapuntzakis</i>, in settings where communication speed was not the main factor (<i>i.e.</i> for students ,and faculty working on campus on the fast internal network), the “local” COW disks (the leaf images) could be stored on the same device as the root images. Decl. ¶405. The same is true of the cache “shadow” COW disks. <i>Sapuntzakis</i> recognizes that “[w]e expect to find identical blocks of data between disk images and memories, even

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
	across different users’ capsules [’s caches].” <i>Sapuntzakis</i> at 8. Thus, POSA would know that caching these on a centralized appliance would be advantageous. Decl. ¶406. This would also improve the cost/benefit of the use of high speed solid state, persistent disk-centered memory caches (of the sort present in certain embodiments of <i>Holzmann</i>). Decl. ¶407-08.
[4.0] The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of	See [1], [12.0]. A POSA would have understood that such system components, taken together, embody the Union Block Device of this claim. Decl. ¶410.

Claims	Ground 1A <i>Sapuntzakis</i> (EX1005)/ <i>Holzmann</i> (EX1007)
respective leaf images.	
[5.0] The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.	See [4.0], [12.0].
[6.0] The system as recited in claim 1 wherein said first storage unit is read-only.	See [1], [13.0].

VII. GROUND 1B: Claims 14-27 are obvious over *Sapuntzakis* combined with *Federwisch*

A. Overview

1. *Sapuntzakis* (EX1005)

See Section VI(A)(1).

2. ***Federwisch* (EX1008)**

Federwisch discloses mirroring changes between source and destination snapshots by comparing two snapshots and identifying differences in “the logical file block index of each snapshot.” The system compares the indexes to file block numbers between two snapshots to identify changed data. The “scanner searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier....Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed...” EX1008, Abstract, [0071-72].

Each snapshot (the “leaf image”) has its own index of file blocks. *Id.* [0109]. The system performs re-indexing when updating destination snapshots’ index maps. *Id.* [0134, 0137]. It uses NVRAM and buffer caches for both snapshots and active file systems. *Id.* [0048, 0053, 0058, 0070].

Data is shared (“exported” over a network), thus “allowing source and destination systems of different vendors to share the information.” *Id.* [0015, 0086, 0120]. *See also*, Decl. ¶257-73.

B. Motivation To Combine

Under *KSR* and as detailed in the Zadok Declaration (EX1003), a POSA would have been motivated to combine *Sapuntzakis* and *Federwisch*. Both operate at and recognize disk blocks at the block layer (while also recognizing that file

systems interoperate and work atop the block layer), disclose/use CoW snapshots (a POSA would have known CoW techniques enable snapshots), and disclose caching blocks of the root/boot disk images. Further, a POSA would have been motivated to modify *Sapuntzakis* with *Federwisch*'s indexing – including because, while *Sapuntzakis* discloses that data is indexed, it does so only in connection with a hash table that indexes cached entries for faster access. A POSA building a *Sapuntzakis* system, and knowing indexing of data items was a well-known technique, would have been motivated to seek additional details on indexing storage systems, files, and caches thereof (supplied by *Federwisch*). A POSA would also have known that indexing of data is an age-old technique long utilized in computer systems. Modifying *Sapuntzakis* to include *Federwisch*'s indexing would have been simple to a POSA, and would not have required extensive experimentation or imposed undue burden – including because *Sapuntzakis* already indexed data objects and a POSA would not have expected *Sapuntzakis* to provide details of such a basic concept. For the same reasons, the combination would have yielded predictable results, as indexes are well understood to structure and locate data objects efficiently, and a POSA would have had a high expectation of success, because indexing techniques were well understood for decades beforehand. See Decl. ¶¶429-51.

C. Detailed Analysis

Claims 14-27 are obvious as stated herein, the table below, and the Zadok Declaration (EX1003, ¶413-491). Again, *Sapuntzakis* discloses the claimed root/leaf architecture. *Federwisch* discloses any additional indexing limitations to the extent, if any, not already taught by *Sapuntzakis* and/or obvious to a POSA.

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
[14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:	See [7.0] of Ground 1A (henceforth “G1A-”), above.
[14.1] a first storage unit configured to store blocks of a root image of said compute nodes;	See G1A-[7.1].
[14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and	See G1A-[7.2].

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and	
[14.3] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by	<i>Sapuntzakis</i> and <i>Federwisch</i> further disclose at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes. <i>Sapuntzakis</i> discloses that data is indexed. Decl. ¶424. <i>Federwisch</i> discloses a “scanner” which serves to index an image. Decl. ¶417-20, 426-28, <i>Federwisch</i> Abstract, [0053], [0071], [0072]. A POSA would have understood that the scanner could be implemented on any compute node and used to index the root image and provide the index results to other compute nodes. Decl. ¶419. <i>Sapuntzakis</i> discloses the remaining limitations of this term. See G1A-[4.0].

Claims	Ground 1B <i>Sapuntzakis (EX1005)/Federwisch (EX1008)</i>
merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.	
[15.0] The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.	<i>See [14], G1A-[3.0].</i>
[16.0] The system as recited in claim 14 further comprising: a plurality of union block devices configured to	<i>See [14], G1A-[4.0].</i>

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.	
[17.0] The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit,	See [16.0], G1A-[5.0].

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
respective second storage units, and said cache.	
[18.0] The system as recited in claim 14 wherein said first storage unit is read-only.	<i>See</i> [14], G1A-[13.0].
[19.0] A method for indexing file systems for a plurality of compute nodes, comprising:	<i>See</i> [14.0].
[19.1] storing blocks of a root image of said compute nodes on a first storage unit;	<i>See</i> [14.1].
[19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute	<i>See</i> [14.2].

Claims	Ground 1B <i>Sapuntzakis (EX1005)/Federwisch (EX1008)</i>
nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;	
[19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;	<i>See [11.0].</i>
[19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and	<i>See [14.3].</i>
[19.5] providing said indexing results to the others	<i>See [14.3].</i>

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
of said compute nodes.	
[20.0] The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.	<i>See</i> [19].<i>Sapuntzakis</i> and <i>Federwisch</i> further disclose storing said indexing results on a shared storage unit. <i>See</i> [14.3]. <i>Federwisch</i> discloses sharing indexes in a system-agnostic format, and the ability to share indexes among devices. Decl. ¶466-67, <i>Federwisch</i> [0086], [0015], [00120]. A POSA would have understood it would be advantageous to store said indexing results on a shared storage unit. Decl. ¶463.
[21.0] The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.	<i>See</i> [19], G1A-[12.0].
[22.0] The method as recited in claim 19 wherein said first storage unit is read-only.	<i>See</i> [19], G1A-[13.0].
[23.0] A computer-readable storage medium having instructions stored	<i>See</i> [19].

Claims	Ground 1B <i>Sapuntzakis (EX1005)/Federwisch (EX1008)</i>
thereon that, in response to execution by at least one computing device, cause the at least one computing device to:	
[23.1] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file	See [19.2], [19.3].

Claims	Ground 1B <i>Sapuntzakis (EX1005)/Federwisch (EX1008)</i>
system is the result of merging said root image portion and said leaf image portion together at the block-level;	
[23.2] index said root image portion; and	<i>See [19.4].</i>
[23.3] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.	<i>See [19.5], [14.3].</i>
[24.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by	<i>See [23], [20.0].</i>

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
said second compute node.	
[25.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.	<i>See</i> [23]. <i>Sapuntzakis</i> and <i>Federwisch</i> further disclose the instructions further causing the at least one computing device to index said leaf image portion. A POSA would have understood that it is desirable to index only changed portions of target rather than re-indexing the entire target each time. Decl. ¶476. <i>Sapuntzakis</i> discloses root and leaf images represented at the block level, wherein the leaf images store changed blocks of the root image. <i>See</i> G1A-[7.2]-[7.3]. <i>Federwisch</i> discloses an indexing system operating at the block level capable of identifying and indexing changed blocks. ¶480-82, <i>Federwisch</i> [0071], [0072]. A POSA would have understood to combine these disclosures to index individual leaf images. Decl. ¶483.
[26.0] The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image	<i>See</i> [25.0]. <i>Sapuntzakis</i> and <i>Federwisch</i> further disclose the instructions further causing the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion. <i>See</i> [14.3]. <i>Federwisch</i> discloses re-indexing of incremental snapshots (the leaf images). Decl. ¶487, <i>Federwisch</i> [0137]. A POSA would have understood that it would be desirable to merge the (unchanging) index of the root image with the

Claims	Ground 1B <i>Sapuntzakis</i> (EX1005)/ <i>Federwisch</i> (EX1008)
portion with said results of said indexing of said root image portion.	result of re-indexing a leaf image to form an up-to-date overall image. Decl. ¶485.
[27.0] The computer-readable storage medium of claim 23 wherein said root image portion is read-only.	See [23], G1A-[13.0].

VIII. GROUND 2A: Claims 14-13 are obvious over *Birse* combined with *Holzmann*

A. Overview

1. *Birse* (EX1006)

Birse relates to the provision of a OS in a net-booted environment. EX1006, 1:18-21. According to one embodiment, a server in a network computer (“NC”) system maintains a copy of the OS that cannot be corrupted by ordinary users of the NC system. Additionally, the NC system may preserve user customizations (*e.g.*, preferences) by maintaining individual, user, storage areas. When an NC client boots from the network and accesses a stored copy of the OS from an NC server, the user’s preferences are dynamically merged with the system environment provided to the

NC client. *Id.* 2:56-65. Advantageously, because of these features, a user may login to any NC client on the network and have the same user experience. *Id.* 2:65-3:3.

The NC system includes an NC client **150** (*e.g.*, personal computer) coupled to an NC server **170** via data communication line **140**. *Id.* 4:65-5:8. NC server **170** can include one or more system volumes **174** (including a protected, read-only, master copy of the OS software), one or more application volumes **176**, a user registry **178**, non-persistent client-specific data **184** (temporary data storage), and persistent user data **186** (long-term storage for user information, *e.g.*, preferences, browser bookmarks, other desktop environment customizations). *Id.* 5:9-29, 6:43-48.

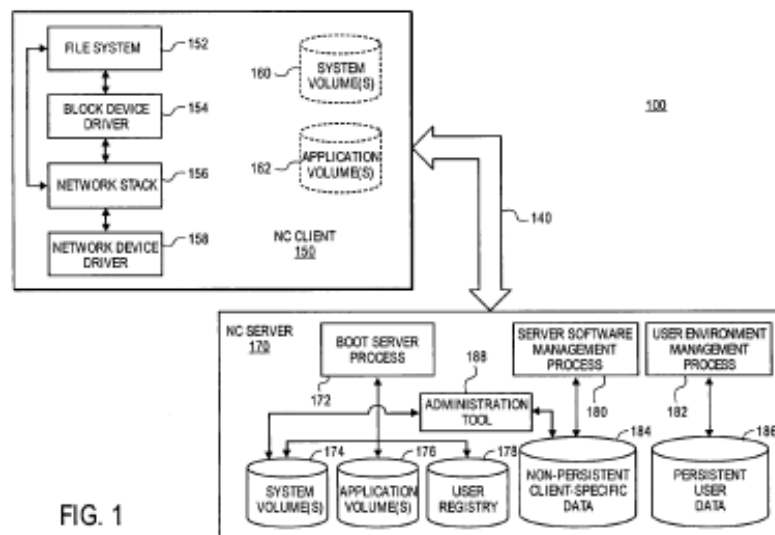


FIG. 1

Birse, Figure 1

The NC server also includes a boot server process **172** (managing access to the system volumes **174** and the application volumes **176**), a server software process **180** (managing access to the non-persistent client-specific data **184**), and a user environment process **182** (maintaining the persistent user data **186**). *Id.* 5:46-6:34.

The NC client also includes a file system **152** (logical structures and software routines to control access to and from local and remote storage media), a block device driver **154** (servicing file system read/write requests to the system volume **160** or shadow volumes, discussed below), a network stack **156** (used to access the data communication link **140**), and a network device driver **158**. *Id.* 7:6-60.

When an NC client boots from the network and accesses the operating system from an NC server, the user's preferences are dynamically merged with the system environment provided to the NC client. *Id.* 2:62-65. In one embodiment, one or more shadow volumes is created from the NC client **150** and stored with the non-persistent client-specific data **184** for the purpose of storing user preferences and/or changes to the operating system. *Id.* 7:41-46. *See also*, Decl. ¶274-305.

2. *Holzmann* (EX1007)

See Section VI(A)(2).

B. Motivation To Combine

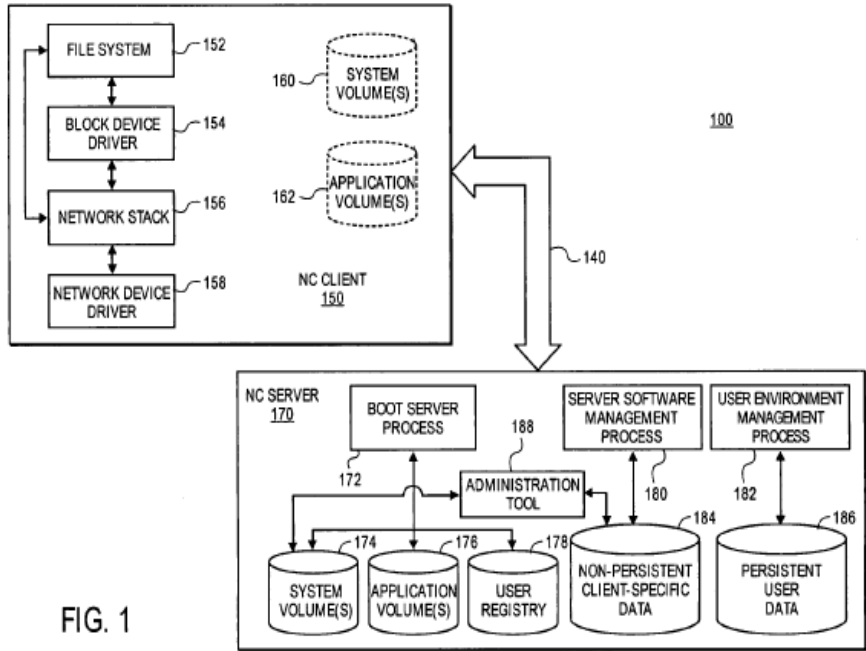
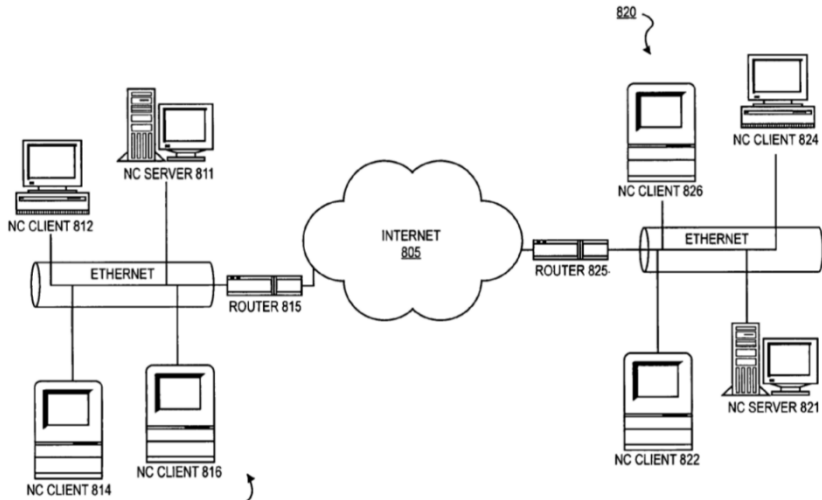
Under *KSR* and as detailed in the Zadok Declaration (EX1003), a POSA would have been motivated to combine *Birse* and *Holzmann*. Both are concerned with efficient management of whole OS images and booting systems over a network. While *Birse* discloses the presence of cache as a feature on some embodiments of the system's server and client computers, it lacks detail how caching may be applied to its boot images; while *Holtzman* discloses details of caches for each system/server being booted, as well as different cache configurations. Further, a POSA would have been motivated to modify *Birse* with *Holzmann*'s caching techniques including because, while *Birse* discloses typical caching, *Holzmann* offers details of how to cache data for the OS distribution purpose (*e.g.*, *Birse*'s boot images). Moreover, caching data from comparatively slow storage devices in faster devices (including solid-state memory) was a well-known technique commonly used for decades before the 844 patent. Modifying *Birse* with *Holzmann*'s caching teachings would have been simple to a POSA, and would not have required extensive experimentation or imposed undue burden – especially when both disclose caching. For the same reasons, the combination would have yielded predictable results (as caching is well-understood to improve performance and resource utilization). A POSA would also

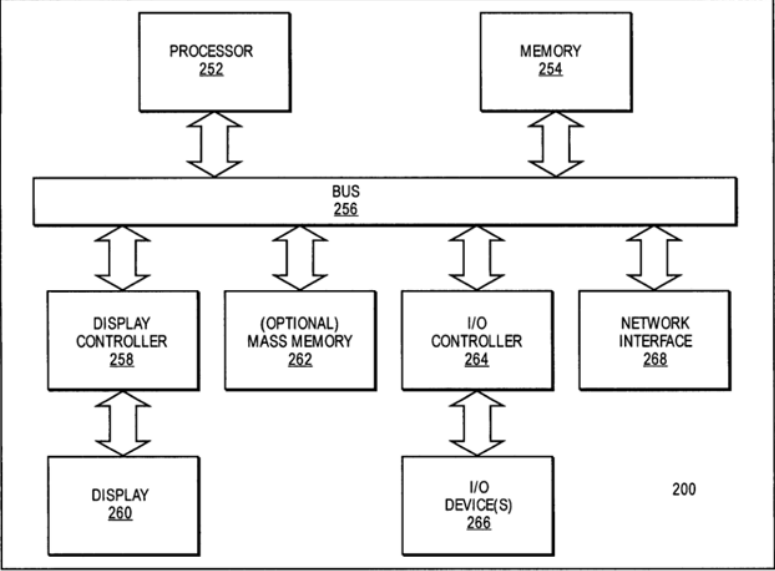
have a high expectation of success, as *Holzmann* merely describes caching in greater detail than *Birse*. Decl. ¶¶510-28.

C. Detailed Analysis

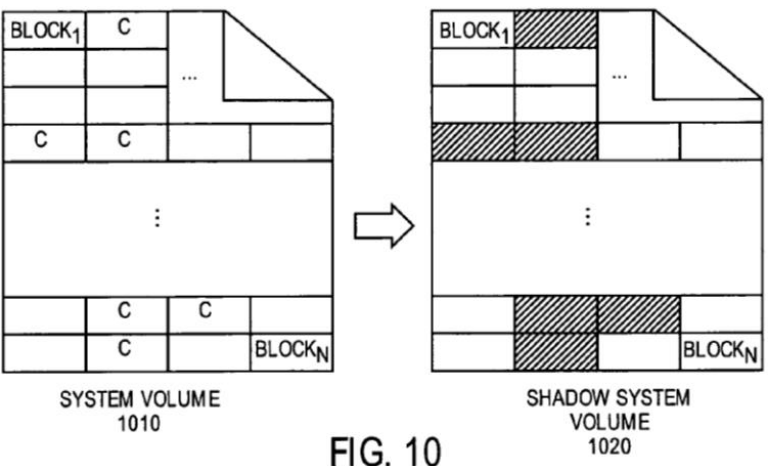
Claims 1-13 are obvious as stated herein, the table below, and the Zadok Declaration (EX1003, ¶¶492-565). Notably, *Birse* discloses the claimed root/leaf architecture where leaf images only contain additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image. *Holzmann* discloses any additional caching limitations to the extent, if any, not already taught by *Birse* and/or obvious to a POSA.

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
[7.0] A method for providing data to a plurality of compute nodes, comprising:	<i>Birse</i> discloses a method for providing data (from NC server 170) to a plurality of compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826): “...[T]he NC clients [150] boot over the network by accessing a stored copy of the operating system from the NC server [170]. Additionally, when NC clients want to run an application, they access it from the NC server.” EX1006, 4:56-60. <i>See also</i> 2:51-3:14 (alternative embodiments providing data from NC server to NC clients to upgrade operating system, upgrade applications).

Claims	Ground 2A Birse (EX1006)/Holzmann (EX1007)
	 FIG. 1 “FIG. 8 is a simplified block diagram illustrating two NC systems 810 and 820 coupled via the Internet 805.” <i>Id.</i> 14:8-9.  FIG. 8

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	Further, NC server 170 and NC clients 150 are illustrated by exemplary digital processing system 200 of FIG. 2. As described at 7:63-8:35, they disclose all essential elements of a compute node. See 4:27-57.  FIG. 2
[7.1] storing blocks of a root image of said compute nodes on a first storage unit;	<i>Birse</i> discloses storing blocks of a root image (system volumes 174, 660, 922, 924) of said compute nodes (multiple NC clients 150) on a first storage unit (one or more hard drives 610, logically separate in NC folders/directories 630, 640): “The NC server 170 includes one or more system volumes 174....The system volumes 174 include a protected, read-only, master copy of the operating system software.” <i>Id.</i> 5:10-14. “[T]he NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 The core system volume 922 preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	system volume 922 is to provide all the system components that are mandatory for system operation.... The user system volume 924 contains all the user-configurable system components, including preferences and all the system additions installed by application software, such as application-installed extensions and libraries.” <i>Id.</i> 14:27-34, 41-44. “[T]he system volumes 174...may be independently distributed across multiple server computer systems.” <i>Id.</i> 6:59-63. “FIG. 6 [below] is a simplified block diagram illustrating the layout of an exemplary hierarchical directory structure that may be used by an NC server [170] according to one embodiment....[A] hard drive 610 of the NC server 170...includes an NC folder (directory) 620. Within the NC folder 620 are an NC admin folder 630 and an NC shared data folder 640.” <i>Id.</i> 12:30-38. <i>Birse</i> discloses that the file 660, storing the shared OS image (within the folder 640), is a first storage unit storing blocks of a root image. “The shared operating system image 660 is a read-write version of the read-only master operating system image 650 and may be mounted by the NC client 150 as part of step 395.” <i>Id.</i> 12:44–47. The shared system volume 660 (a.k.a. system volume 174, 922, 924, 1010) is stored as blocks as shown in Fig. 10:

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	 FIG. 10
[7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by	<i>Birse</i> discloses storing leaf images (shadow volumes/images 930, 695) for respective compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826) on respective second storage units (NC client folders 690 #1...n), said leaf images including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written): “In this example, the NC clients are numbered from 1 to N and the clients folder 680 includes a folder for each NC client. For example, an NC #1 folder 690 is an area for storing client-specific information, such as a shadow image 695, corresponding to NC client #1.” <i>Id.</i> 12:54–59 (discussing FIG. 6, in [7.1] above). “The block device driver 154 services file system read and write requests to the system volumes 160. In one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver ... [but is] configured to service the read and

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
respective compute nodes; and	write requests of the file system 152 by accessing the NC server 170. As will be discussed further below, in one embodiment, one or more shadow volumes [695], may be created for the NC client 150 and stored with the non-persistent client-specific data 184 for the purpose of storing user preferences and/or changes to the operating system. Therefore, some mechanism is needed to direct reads and writes to the appropriate volume, e.g., the shadow volumes or the system volumes 174. According to one embodiment, the block device driver 154 has knowledge of the separate shadow volumes and system volumes 174 and redirects read and write requests to the appropriate volume. Alternatively, server-side logic may perform the redirection locally.” <i>Id.</i> 7:35-53; see <i>also</i> 11:30-32. “FIG. 4 is a flow diagram illustrating how NC client [150] write requests directed to the system volumes 160 are handled... “...[A]t step 410, the file system 152 generates a write request directed to the system volumes 160. The write request is received by the block device driver 154 and redirected by the block device driver 154 at step 420 by translating it into a write request directed to the user’s shadow volume on the NC server 170. At step 430, the NC server 170 stores information associated with the write request in the shadow volume associated with the NC client 150. In this manner, the file system 152 need not be aware that attempted modifications to the system volumes 160 are recorded instead in a client-specific shadow volume residing on the NC server 170.” <i>Id.</i> 10:63-65, 11:6-17.

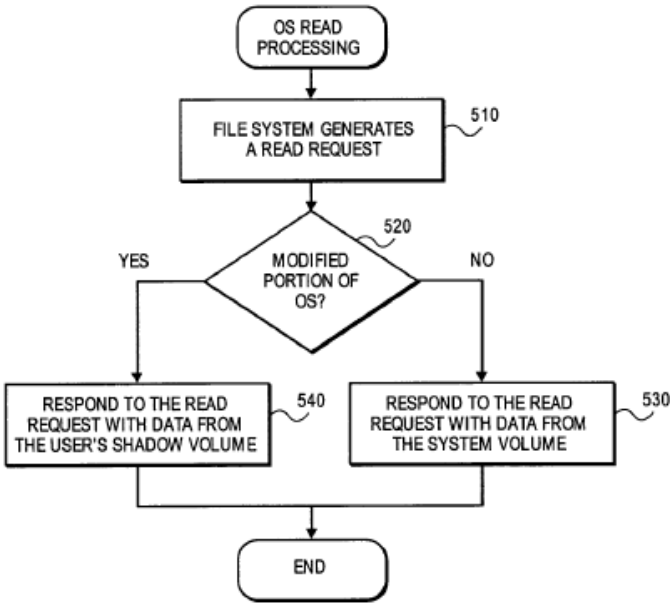
Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	<pre> graph TD A([OS WRITE PROCESSING]) --> B[FILE SYSTEM GENERATES A WRITE REQUEST 410] B --> C[BLOCK DEVICE DRIVER REDIRECTS THE WRITE REQUEST TO THE USER'S SHADOW VOLUME ON THE NC SERVER 420] C --> D[NC SERVER STORES INFORMATION ASSOCIATED WITH THE WRITE REQUEST IN THE USER'S SHADOW VOLUME 430] D --> E([END]) </pre> FIG. 4 “[In one embodiment,] the shadow volume contains only those portions of the operating system that have been modified by the user rather than a complete copy of the operating system as modified.” <i>Id.</i> 11:49-52. In another embodiment, the shared OS image 660 is divided into two parts: “[t]he NC server 170 also creates a shadow system volume 930 for each connected NC client. The shadow system volume 930 shadows the user system volume 924 by storing modifications that are made to the user system volume 924.” <i>Id.</i> 14:49-53. “...[T]he shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a ‘copy-on-write’ storage scheme in which modifications (writes) directed to the user system

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	volume 930 are instead copied to the shadow system volume 930. ” <i>Id.</i> 14:56-61.
[7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.	<i>Birse</i> and <i>Holzmann</i> disclose caching blocks of said root image (system volumes 174, 660, 922, 924) that have been accessed by at least one of said compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826) in a cache memory: <u>Holzmann:</u> <i>See Holzmann</i> portion of [7.3] of Ground 1A (henceforth “G1A-”), above. It would have been obvious to a POSA to combine the cache system of <i>Holzmann</i> with the system of <i>Birse</i> to improve the performance of <i>Birse</i>. <i>See</i> VII(B), above.
[8.0] The method as recited in claim 7 further comprising:	<i>See</i> [7].
[8.1] receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and	<i>Birse</i> and <i>Holzmann</i> disclose receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory: <u>Holzmann:</u> <i>See Holzmann</i> portion of G1A-[8.1]. It would have been obvious to a POSA to combine the cache system of <i>Holzmann</i> with the system of <i>Birse</i> to improve the performance of <i>Birse</i>. <i>See</i> VII(B), above.
[8.2] providing said first portion of said data to said at least one of said compute nodes	<i>Birse</i> and <i>Holzmann</i> disclose providing said first portion of said data to said at least one of said compute nodes from said cache memory: <u>Holzmann:</u> <i>See Holzmann</i> portion of G1A-[8.2].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
from said cache memory.	It would have been obvious to a POSA to combine the cache system of <i>Holzmann</i> with the system of <i>Birse</i> to improve the performance of <i>Birse</i> . See VII(B), above.
[9.0] The method as recited in claim 8 further comprising: updating said cache memory based on said read request.	See [8]. See <i>Holzmann</i> portion of G1A-[9.0].
[10.0] The method as recited in claim 9	See [9.0].
[10.1] wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and	See <i>Holzmann</i> portion of G1A-[10.1].
[10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.	See <i>Holzmann</i> portion of G1A-[10.2].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
[11.0] The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.	See [7]. <i>Birse</i> discloses merging the blocks of said root image (blocks of the operating system software stored on system volumes 174, 660, 922) with the blocks of respective leaf images (695) to create cohesive respective application environments (dynamically merged system environment): “In one embodiment, the shadow system volume 930 is not visible to the user. Consequently, the user system volume 924 will appear to be the location where all writes are directed. For example, if a user modifies a preference, that preference will appear to be written to the preference file located in the preferences folder on the user system volume 924.” 14:61-67. “When an NC client [150] boots from the network and accesses the operating system from an NC server [170], the user’s preferences are dynamically merged with the system environment provided to the NC client [150]. Advantageously, since, the user’s desktop preferences and other customized settings are all preserved from session to session and supplied to the NC client [150] as it boots from the network, the user may login to any NC client [150] on the network and have the same user experience.” 9:2-10. “Referring now to Fig. 5, NC client [150] read request processing will now be described according to one embodiment of the present invention....At step 510, the file system 152 generates a read request directed to the system volumes 160. At step 520, the read request is received by the block device driver 154 and a determination is made whether the read request specifies a portion of the operating system that has been modified by the user or whether the read request specifies a portion of the operating system that

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	remains unchanged. If the read request corresponds to a portion of the operating system that has not been modified by the user, the processing continues with step 530. However, if the read request corresponds to a portion of the operating system that has been previously modified by the user, then processing continues with step 540. At step 530, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the system volumes 174. At step 540, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the user's shadow volume. In this manner, the file system 152 need not be aware of the particular mechanism used to coordinate user modifications with the original operating system as contained in the system volumes 174.” 11:39-41, 11:55-12:10.

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	 <pre> graph TD A([OS READ PROCESSING]) --> B[FILE SYSTEM GENERATES A READ REQUEST 510] B --> C{MODIFIED PORTION OF OS? 520} C -- YES --> D[RESPOND TO THE READ REQUEST WITH DATA FROM THE USER'S SHADOW VOLUME 540] C -- NO --> E[RESPOND TO THE READ REQUEST WITH DATA FROM THE SYSTEM VOLUME 530] D --> F([END]) E --> F </pre> FIG. 5
[12.0] The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.	<i>Birse</i> in view of <i>Holzmann</i> discloses the method as recited in claim 11, and, in particular, that the 544 (which serve as the union block devices) work below the client file system, and, hence, at an operational level. <i>See</i> [11.0]. It would have been obvious to a POSA to combine <i>Birse</i> with <i>Holzmann</i> to extend the function of the block device drivers (154), implementing the algorithm of Figure 5, to incorporate blocks stored in the cache, as well. Decl. ¶545.
[13.0] The method as recited in claim	<i>See</i> [7].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
7 wherein said first storage unit is read-only.	<i>Birse</i> further discloses the first storage unit being read-only: “The system volumes 174 include a protected, read-only, master copy of the operating system software.” <i>Birse</i> at 5:13-14.
[1.0] A system for providing data to a plurality of compute nodes, the system comprising:	<i>See</i> [7.0].
[1.1] a first storage unit configured to store blocks of a root image of said compute nodes;	<i>See</i> [7.1].
[1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of	<i>See</i> [7.2].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and	
[1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.	<i>See</i> [7.3].
[2.0] The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.	<i>See</i> [1.0]-[1.3]. <i>See Holzmann</i> portion of G1A-[2.0]. <i>See</i> Decl. ¶¶552-56.
[3.0] The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.	<i>See</i> [1]. <i>Birse</i> discloses the first storage unit (holding the core system volumes and the user system volume), and the second storage units (holding the shadow system volumes) contained within a single storage appliance: “FIG. 6 is a simplified block diagram illustrating the layout of an exemplary hierarchical directory structure

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
	that may be used by an NC server according to one embodiment of the present invention. According to the embodiment depicted, a hard drive 610 of the NC server 170 or other mass storage device associated with the NC server 170 includes an NC folder (directory) 620.” <i>Birse</i> at 12:30–37 FIG. 6 <i>Holzmann</i> discloses a cache mechanism for server drives. <i>Holzmann</i> [0021], Decl. ¶560. It would have been obvious to a POSA to implement the cache system of <i>Holzmann</i> on the storage device holding the first and second storage units. Decl. ¶561.
[4.0] The system as recited in claim 1	See [1], [12.0].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.	A POSA would have understood that such system components, taken together, embody the Union Block Device of this claim. Decl. ¶563.
[5.0] The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute	See [4.0], [12.0].

Claims	Ground 2A <i>Birse</i> (EX1006)/ <i>Holzmann</i> (EX1007)
nodes and said first storage unit, respective second storage units, and said cache.	
[6.0] The system as recited in claim 1 wherein said first storage unit is read-only.	See [1],[13.0].

IX. GROUND 2B: Claims 14-27 are obvious over *Birse* combined with *Federwisch*

A. Overview

1. *Birse* (EX1006)

See Section VIII(A)(1).

2. *Federwisch* (EX1009)

See Section VII(A)(2).

B. Motivation To Combine

Under *KSR* and as detailed in the Zadok Declaration (EX1003), a POSA would have been motivated to combine *Birse* and *Federwisch*. Both operate at and recognize disk blocks at the block layer (while also recognizing that file systems interoperate and work atop the block layer), disclose/use CoW snapshots (a POSA would have known CoW techniques enable snapshots), and disclose caching blocks

of the root/boot disk images. Further, a POSA would have been motivated to modify *Birse* with *Federwisch*'s indexing – including because, while *Birse* suggests that the boot images are indexed and organized through bootfile selection and a registry, a POSA building a *Birse* system, and knowing that indexing of data items was a well-known technique, would have been motivated to seek additional details on indexing storage systems, files, and caches thereof (supplied by *Federwisch*). A POSA would also have known that indexing of data is an age-old technique long utilized in computer systems. Modifying *Birse* to include *Federwisch*'s indexing would have been simple to a POSA, and would not have required extensive experimentation or imposed undue burden – including because *Birse* likely already supported the use indexing tools and a POSA would not have expected *Birse* to provide details of such a basic concept. For the same reasons, the combination would have yielded predictable results, as indexes are well understood to structure and locate data objects efficiently, and a POSA would have had a high expectation of success, because indexing techniques were well understood for decades beforehand. *See* Decl. ¶¶581-605.

C. Detailed Analysis

Claims 14-27 are obvious as stated herein, the table below, and the Zadok Declaration (EX1003, ¶¶566-646). Again, *Birse* discloses the claimed root/leaf

architecture. *Federwisch* discloses any additional indexing limitations to the extent, if any, not already taught by *Birse* and/or obvious to a POSA.

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
[14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:	See [7.0] of Ground 2A (henceforth “G2A-”), above.
[14.1] a first storage unit configured to store blocks of a root image of said compute nodes;	See G2A-[7.1].
[14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf	See G2A-[7.2].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and	
[14.3] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of	<i>Birse</i> and <i>Federwisch</i> further disclose at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes. <i>See Federwisch</i> portion of Ground 1B (henceforth “G1B-”) [14.3]. <i>Birse</i> discloses the remaining limitations of this term. <i>See</i> Ground 2A (henceforth “G2A-”) [4.0]. <i>See also</i>, Decl. ¶569-81.

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.	
[15.0] The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.	<i>See</i> [14], G2A-[3.0].
[16.0] The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second	<i>See</i> [14], G2A-[4.0].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.	
[17.0] The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.	See [16.0], G2A-[5.0].

Claims	Ground 2B <i>Birse</i> (EX1006)/<i>Federwisch</i> (EX1008)
[18.0] The system as recited in claim 14 wherein said first storage unit is read-only.	<i>See</i> [14], G2A-[13.0].
[19.0] A method for indexing file systems for a plurality of compute nodes, comprising:	<i>See</i> [14.0].
[19.1] storing blocks of a root image of said compute nodes on a first storage unit;	<i>See</i> [14.1].
[19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf	<i>See</i> [14.2].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;	
[19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;	<i>See</i> [11.0].
[19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and	<i>See</i> [14.3].
[19.5] providing said indexing results to the others of said compute nodes.	<i>See</i> [14.3].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
[20.0] The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.	<i>See</i> [19], G1B-[20.0]. <i>See also</i> , Decl. ¶¶616-22.
[21.0] The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.	<i>See</i> [19], G2A-[12.0].
[22.0] The method as recited in claim 19 wherein said first storage unit is read-only.	<i>See</i> [19], G2A-[13.0].
[23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device,	<i>See</i> [19.0].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
cause the at least one computing device to:	
[23.1] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image	<i>See</i> [19.2], [19.3].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
portion together at the block-level;	
[23.2] index said root image portion; and	<i>See</i> [19.4].
[23.3] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.	<i>See</i> [19.5], [14.3].
[24.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.	<i>See</i> [23], [20.0].

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
[25.0] The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.	<i>See</i> [23]. <i>Sapuntzakis</i> and <i>Federwisch</i> further disclose the instructions further causing the at least one computing device to index said leaf image portion. A POSA would have understood that it is desirable to index only changed portions of target rather than re-indexing the entire target each time. Decl. ¶631. <i>Birse</i> discloses root and leaf images represented at the block level, wherein the leaf images store changed blocks of the root image. <i>See</i> G2A-[7.2]-[7.3]. <i>Federwisch</i> discloses an indexing system operating at the block level capable of identifying and indexing changed blocks. Decl. ¶480-82, <i>Federwisch</i> [0071], [0072]. A POSA would have understood to combine these disclosures to index individual leaf images. Decl. ¶638.
[26.0] The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said	<i>See</i> [25.0], G1B-[26.0]. <i>See also</i>, Decl. ¶639-45.

Claims	Ground 2B <i>Birse</i> (EX1006)/ <i>Federwisch</i> (EX1008)
indexing of said root image portion.	
[27.0] The computer-readable storage medium of claim 23 wherein said root image portion is read-only.	See [23], G2A-[13.0].

X. DISCRETIONARY DENIAL IS NOT APPROPRIATE

A. 35 U.S.C. § 325(d)

None of the references in the above-grounds were substantively addressed or even cited during prosecution of the 844 patent, and nothing suggests their teachings relied on herein – much less the combination of those teachings as informed by a POSA’s knowledge and expert testimony—were considered by the Examiner. *Mylan Pharmaceuticals Inc., v. Regeneron Pharmaceuticals, Inc.*, IPR2022-01226, Paper 22 at 27-29 (PTAB Jan. 11, 2023).

While Haun (which shares a common specification with *Birse*) was briefly referenced by the Examiner during prosecution of the 844 patent, it was never relied upon. Indeed, despite being “pertinent” to Applicants’ disclosure, it was never mentioned again (even after Nguyen was discarded). This, at the same time a different Examiner, assigned the “grandparent” 622 Application, repeatedly and

successfully cited Haun to support rejection of all similar claims (resulting in abandonment).

B. 35 U.S.C § 314(a)

Fintiv and the USPTO's 6/21/2022 interim guidelines favor institution. Factor 1 is at worst neutral. No stay request has been filed (although Petitioners may seek a stay). *Sand Revolution II LLC v. Continental Intermodal Group - Trucking LLC*, IPR2019-01393, Paper 24, 7 (PTAB June 16, 2020) (informative).

Factors 2 and 3 favor institution. The court cases against Petitioners (filed 11/15/2023) are in early stages. The *Markman* process has not begun, fact discovery remains open, no merits fact depositions have occurred, and expert discovery has not commenced. In 10/2024, the trial date was extended to 9/22/2025; but on 11/5/2024 (after trial was moved back) Patent Owner filed amended Complaints against Liberty Mutual⁷ and Comerica, and finally served source code infringement contentions (which themselves were delayed by upwards of 135 days). Without contentions, Petitioners could not know the full details of Patent Owner's theories. In *Sand Revolution*, the court "conducted a *Markman* hearing and entered a related

⁷ Liberty Mutual has not answered, and a pending motion (filed 1/2024) seeks to dismiss the entire Complaint for improper venue.

Order.” However, Factor 3 weighed “only marginally, if at all, in favor of exercising discretion to deny institution.” *Id.* at 10. Here, the parallel proceeding is at a substantially earlier phase.

Factor 4 also strongly favors institution. Only claims 1 and 7 have been asserted against Petitioners. This Petition thus largely addresses issues that will not be litigated in court (25 of 27 challenged claims). Moreover, while the parties are the same in both proceedings, this is “far from an unusual circumstance,” and does not warrant denial under Factor 5. *Sand Revolution*, Paper 24 (instituting IPR).

Factor 6 is only addressed if the others support denial, which they do not. *CommScope Techs. LLC v. Dali Wireless, Inc.*, IPR2022-01242, Paper 23 (Feb. 27, 2023) (precedential).

This Petition presents a compelling unpatentability challenge. The prior art, none of which was relied upon by the Examiner, rebuts the Patent Owner’s arguments during prosecution that persuaded the Examiner to allow the claims. Accordingly, the Board should not deny institution under *Fintiv* and its guidelines.

XI. MANDATORY NOTICES

Pursuant to 37 C.F.R. § 42.8(b), Petitioners state:

A. Real Party-in-Interest

The real parties-in-interest are Liberty Mutual Insurance Company, Liberty Mutual Technology Group, Inc., Liberty Mutual Holding Company Inc., Liberty Mutual Group Inc., Liberty Mutual Plano LLC, Comparion Insurance Agency, LLC, and Ironshore Holdings (U.S.) Inc. (“Liberty Mutual”) and Comerica Incorporated (“Comerica”).

B. Related Matters

Patent Owner Intellectual Ventures II LLC (along with Intellectual Ventures I LLC), has asserted the 844 patent (and three other unrelated patents) in:

- *Intellectual Ventures I LLC et al. v. Liberty Mutual Holding Company Inc., et al.*, No. 23-cv-525 (E.D. Tex. filed November 15, 2023)
- *Intellectual Ventures I LLC et al. v. Comerica Incorporated*, No. 23-cv-524 (E.D. Tex. filed November 15, 2023)
- *Intellectual Ventures I LLC et al. v. JP Morgan Chase Bank, National Association et al.*, No. 23-cv-523 (E.D. Tex. filed November 15, 2023)

These cases were consolidated, with the latter Case No. 23-cv-523 as the lead case.

The 844 patent is also asserted in the following cases:

- *Intellectual Ventures I LLC et al., v. American Airlines, Inc.*,
No. 24-cv-980 (E.D. Tex. filed November 2, 2024)
- *Intellectual Ventures I LLC et al., v. Southwest Airlines, Co.*,
No. 24-cv-277 (W.D. Tex. filed November 2, 2024).

The 080 patent is also at issue in the following declaratory judgment case:

- *Assurant, Inc. v. Intellectual Ventures I LLC, et al.*, No. 24-cv-344
(D. Del. filed March 15, 2024)

The action against JP Morgan has been dismissed with prejudice. The actions against Petitioners and the other cases remain pending. There has been no ruling on claim construction in any case.

Petitioners are not aware of any other civil actions or proceedings filed in connection with the 844 patent.

C. Lead and Backup Counsel and Service Information

Lead Counsel	Back-up Counsel
Joshua D. Curry josh.curry@lewisbrisbois.com LEWIS BRISBOIS BISGAARD SMITH LLP 600 Peachtree St. NE, Suite 4700 Atlanta, GA 30308 Ph: 404.348.8585 Fax: 404.467.8845 Reg No. 65,478	François O. Ecclesiaste francois.ecclesiaste@lewisbrisbois.com LEWIS BRISBOIS BISGAARD SMITH LLP Wells Fargo Center 90 S 7th Street, Suite 2800 Minneapolis, MN 55402 Ph: 612.428.5048 Fax: 612.428.5001 Reg No. 75,836

Petitioners consent to service by email to the counsel above.

XII. CONCLUSION

For at least the foregoing reasons, Petitioners request IPR be instituted and claims 1-27 (all claims) of the 844 Patent be cancelled.

Respectfully submitted,

/Joshua D. Curry/

Joshua D. Curry – Lead Counsel
Registration No. 65,478
LEWIS BRISBOIS BISGAARD SMITH LLP
600 Peachtree St. NE, Suite 4700
Atlanta, GA 30308
Ph: 404.348.8585
Fax: 404.467.8845
josh.curry@lewisbrisbois.com

François O. Ecclesiaste
Registration Reg No. 75,836
LEWIS BRISBOIS BISGAARD SMITH LLP
Wells Fargo Center
90 S 7th Street, Suite 2800
Minneapolis, MN 55402
Ph: 612.428.5048
Fax: 612.428.5001
francois.ecclesiaste@lewisbrisbois.com

*Counsel for Liberty Mutual Insurance
Company, Liberty Mutual Technology
Group, Inc., Liberty Mutual Holding
Company Inc., Liberty Mutual Group Inc.,
Liberty Mutual Plano LLC, Comparison
Insurance Agency, LLC, Ironshore
Holdings (U.S.) Inc.*

-and-

*Counsel for Petitioner Comerica
Incorporated*

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.6(e) and 42.105(b) and agreement of the parties, the undersigned certifies that on the date indicated below the foregoing document with any attachments was served by email on counsel for the Patent Owner as indicated below.

Brandon R. Theiss
Volpe Koenig
30 South 17th Street, 18th Floor
Philadelphia, PA 19103
btheiss@vklaw.com

Jonathan K. Waldrop
Kasowitz Benson Torres LLP
333 Twin Dolphin Drive, Suite 200
Redwood Shores, CA 94065
jwaldrop@kasowitz.com
IntellectualVentures@kasowitz.com

Date: 11/20/2024

/Joshua D. Curry/_____

CERTIFICATION OF WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), Petitioners certify that the foregoing IPR petition contains 13,878 words, excluding the portions as permitted by § 42.24(a).

Date: 11/20/2024

/Joshua D. Curry/_____