

RESTRICTED CONFIDENTIAL SOURCE CODE

UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

VIRTAMOVE, CORP.,

Plaintiff,

v.

HEWLETT PACKARD ENTERPRISE
COMPANY,

Defendant.

§ Case No. 2:24-cv-00093-JRG

§ (Lead Case)

§

§

§ **JURY TRIAL DEMANDED**

§

§

§

§

§

§

VIRTAMOVE, CORP.,

Plaintiff,

v.

INTERNATIONAL BUSINESS
MACHINES CORP.,

Defendant.

§ Case No. 2:24-cv-00064-JRG

§ (Member Case)

§

§

§ **JURY TRIAL DEMANDED**

§

§

§

§

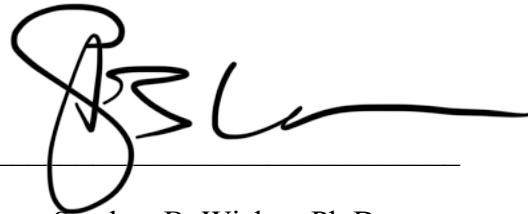
§

§

§

OPENING EXPERT REPORT OF STEPHEN B. WICKER
REGARDING U.S. PATENT NOS. 7,519,814 AND 7,784,058

DATE: June 23, 2025

A handwritten signature in black ink, appearing to be 'S. B. Wicker', written over a horizontal line.

Stephen B. Wicker, Ph.D.

I. INTRODUCTION

1. My name is Stephen B. Wicker. I am a Professor Emeritus of Electrical and Computer Engineering at Cornell University in Ithaca, New York. I am over the age of eighteen and am competent to make this declaration. I have personal knowledge, or have developed knowledge, of the technologies and issues addressed in this report based upon my education, training, and/or experience, of the matters set forth within.

2. I have been retained by Defendant International Business Machines, Corp. Defendant has asked me to provide my expert opinions on technical subjects relating to U.S. Patent No. 7,519,814 (the "'814 Patent") and U.S. Patent No. 7,784,058 (the "'058 Patent") (collectively, the "Asserted Patents"), and in particular claims 1, 2, 6, 9, and 10 of the '814 Patent and claims 1, 2, 3, 4, and 18 of the '058 Patent (collectively, the "Asserted Claims").

3. I provide my opinions on areas including: the technical background related to the technologies purported claimed in the Asserted Patents, whether the Asserted Patents and the Asserted Claims are anticipated or rendered obvious by the prior art or otherwise fail to meet the legal standards relevant to patentability and validity, the materiality of certain prior art references to the Asserted Patents, and VirtaMove's purported development, disclosure, and use of the claimed inventions.

4. My opinions and conclusions and the bases for my opinions and conclusions, as well as the matters on which I may testify at trial in this case, are set forth below.

II. BACKGROUND AND QUALIFICATIONS, PREVIOUS TESTIMONY, COMPENSATION, AND MATERIALS AND INFORMATION CONSIDERED

5. Below I have summarized my educational background, career history, publications, and other relevant qualifications. My full curriculum vitae is attached as Exhibit A to this report.

26. A chart of my opinions on the invalidity of the Asserted Patents in view of prior art under 35 U.S.C. Section 102 and/or 103 is below.¹

The '814 Patent is Invalid in View of:	The '058 Patent is Invalid in View of:
Solaris Zones; Solaris Zones in view of McMillan	Alpine
Thinstall/ThinApp	Deianov; Deianov in view of Levine
Schaefer in view of Gaines and/or Tormasov	FreeBSD 4.0; FreeBSD 4.0 in view of Alpine; FreeBSD in view of Levine
Linux VServer (“Gelinas”)	Virtuozzo; Virtuozzo in view of Alpine
Zap (“Osman”)	Eggert; Eggert in view of Levine
Virtuozzo	User-mode Linux; User-mode Linux in view of Alpine
McMillan in view of Gaines and/or Tormasov	Solaris Zones; Solaris Zones in view of Alpine

IV. LEGAL STANDARDS

27. I am not an attorney. For the purposes of this report, I have been informed about and understand certain aspects of the law that are relevant to my analysis, opinions, and conclusions.

¹ Although this chart identifies the primary references relied on for my opinions concerning invalidity opinions under § 102 and § 103, my analysis further applies the knowledge of a person of ordinary skill in the art. This knowledge is evidence and supported by the discussion and references included in the section of my report entitled “Background of the Technology and Prior Art.” Accordingly, while I provide this chart of references for clarity, it is not intended to be interpreted to exclude other references that I discuss below and that inform the understanding of one of skill in the art or otherwise exclude any other opinions or analysis stated in my report.

isolating untrusted workloads, consolidating multi-tenant hosting, and enforcing least-privilege boundaries across diverse UNIX services.

D. Solaris Zones/Tucker

192. Solaris Zones is Sun Microsystems's operating-system virtualization technology that first appeared as "N1 Grid Containers" during Solaris 10 development and was subsequently productized as "Zones." By virtualizing core OS services inside a single Solaris kernel, Zones allows multiple, isolated, secure, and independently administered application environments to coexist on the same physical or logical server. ORACLE_VM_PA_00001214 at 229; ORACLE_VM_PA_00006499 at 516. The technology delivers containerization without the performance overhead of multiple kernels, thereby combining the manageability of separate systems with the efficiency of a shared kernel. *Id.* at 522.

193. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

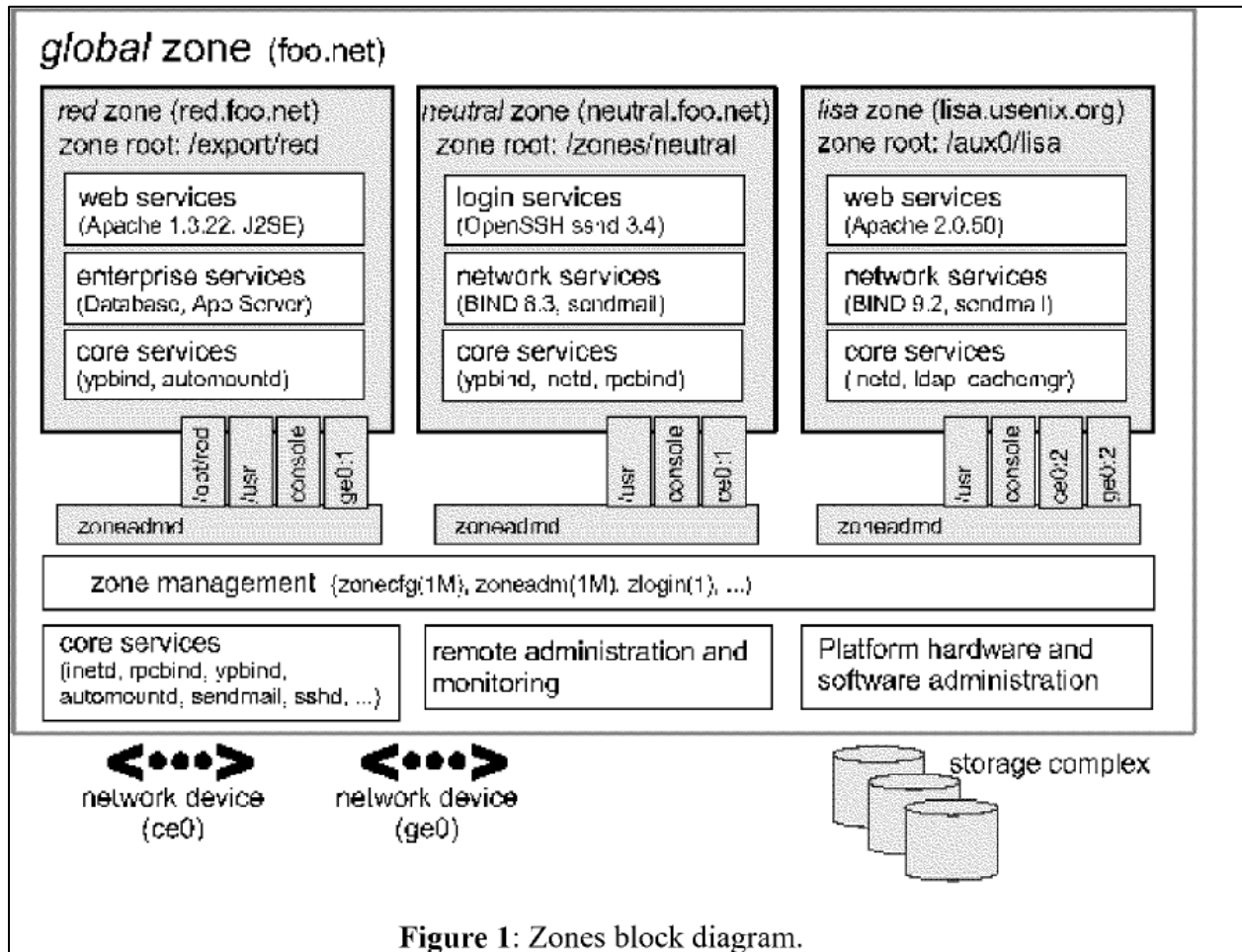
[REDACTED] Reporting on Zones at the time noted how it "allow[ed] multiple applications to run in isolation from each other on the same physical hardware." IBM_VM_000020309.

194. By February 2004, Sun announced Zones in the Solaris Express 2/04 release, describing it as "a software partitioning technology used to virtualize operating-system services and provide an isolated and secure environment for running applications."

ORACLE_VM_PA_00001214 at 229. Detailed information on the functionality of Solaris Zones was deposited with the USPTO on May 3, 2003. ORACLE_VM_PA_00005455 (Tucker Provisional) at -5456. U.S. Patent No. 7,437,556 (the “Tucker”) claims priority to this provisional application, and I understand is entitled to its priority date. This patent incorporates the entirety of the provisional application by reference. *See* ORACLE_VM_PA_00006003 at -6020. The ’556 Patent and the Tucker it incorporates by reference show how Solaris Zones system functioned before the priority dates of the Asserted Patents. I have also reviewed the source code for Solaris Zones, and have confirmed that the copyright dates reflect dates of 2003 or 2004, [REDACTED]
[REDACTED]

195. I understand that a patent properly claims priority to a provisional application when the provisional application contains a written description of the invention claimed in the patent. *See, e.g., Dynamic Drinkware, LLC v. Nat’l Graphics, Inc.*, 800 F.3d 1375 at 1378, 1831-82 (Fed. Cir. 2015). The Tucker Provisional application fully supports the invention claimed in Tucker, including claim 1 of Tucker. Indeed, Tucker Provisional describes in depth the operations of Solaris Zones, including global zones, non-global zones, and where isolated processes run within each non-global zone. *See* ORACLE_VM_PA_00005455 (Tucker Provisional). This is precisely what claim 1 of Tucker is directed at: the establishment of a global zone, the establishment of non-global zones, and the concurrent execution of processes in different non-global zones simultaneously. *See* ORACLE_VM_PA_00006003 at -6028. Thus, Tucker is entitled to Tucker Provisional’s priority date.

196. Depicted in the figure below, the architecture of Solaris Zones is composed of several components that work together to provide secure, isolated, and manageable virtual environments within a single Solaris instance.



ORACLE_VM_PA_00000001 at 004.

197. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED] [REDACTED]

[REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] The virtualized root ensures that applications cannot see outside a zone, providing privacy, security, and failure isolation. IBM_VM_000016471 at 517. [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

199. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

200. Taken together, these functionalities and features of Solaris Zones mirrors the characteristics of modern containers, such as namespace isolation, resource partitioning, security containment, and lightweight deployment. *See, e.g.,* IBM_VM_000016517. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

201. Zones further supports the use of shared libraries as a mechanism for providing system services to zones. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED] Named inventor of the Asserted Patents, Donn Rochette, had studied the Solaris Zones technology and confirmed that Zones one could copy or modify configuration and shared objects into the zone while retaining the originals in the global file system, thereby enabling per-zone behavior. D. Rochette Dep. Tr. at 84:4-85:25.

E. User-mode Linux

202. User-mode Linux (UML) is a thin virtualization of a Linux kernel that executes as a regular, unprivileged process on a Linux host. IBM_VM_000012915 at 917. Early versions were available in 1991, with version 2.4.0 being released by January 2000. IBM_VM_000012939. In contrast to “full” virtualization platforms, which must trap, interpret, and emulate privileged CPU instructions and legacy I/O buses, UML uses the host kernel itself to perform the syscall interception needed to keep guest user-space processes confined to the UML kernel process, enabling the guest to run without special CPU extensions and without elevated permissions. *Id.* at 918. By running in user mode, UML confers unique robustness and observability advantages, such that “[i]f a User Mode Linux kernel crashes, your host kernel is still fine...” *Id.* at 917. When used for virtual hosting, providers can segment commodity servers into isolated UML guests, permitting customers to run standard Linux distributions without risk to neighboring tenants. IBM_VM_000013014.

203. UML employs two operating modes, historically called TT mode and SKAS mode. In legacy TT mode a dedicated “tracing thread” ptraces each guest thread, delivering signals to effect context switches. This offered portability but incurred performance costs by requiring four context switches per guest syscall. To address those costs, the project introduced SKAS

(“Separate Kernel Address Space”) mode, in which “the UML kernel [] run[s] in an entirely different host address space from its processes,” thereby removing the guest kernel image from every process and “eliminating the signal delivery that used to happen for every UML system call.” IBM_VM_000013000. SKAS nearly doubled kernel build throughput and reduced host load averages by up to an order of magnitude when many UML instances run concurrently. *Id.* at 001-002. Crucially, SKAS also “solves the security and honeypot fingerprinting problems by making the UML kernel totally inaccessible to UML processes.” IBM_VM_000013002.

204. UML supplies a comprehensive suite of virtual devices whose backing resources reside in the host. Whereas block devices (“ubd”), for example, are ordinarily mapped to host files that carry guest filesystems, UML supports copy-on-write layering so that “[a]ny changed blocks are stored in the private COW file, while reads come from either device,” enabling disk-space savings and fast rollback for test scenarios. IBM_VM_000012915 at 932. Network devices may be transported over TAP/TUN, multicast, vector I/O, GRE, L2TPv3, or user-level switch daemons, with configurable queue depths and checksum offload options. *Id.* at 919. For direct host-file access, the “hostfs” virtual filesystem allows mounting arbitrary host directories into the guest. *Id.* at 933.

205. Physical memory inside UML is simply a region of the host address space mapped into the guest kernel. Developers may emulate device memory regions by exporting “iomem” areas backed by host files, which a driver can locate via the `find_iomem()` API. IBM_VM_000013032.

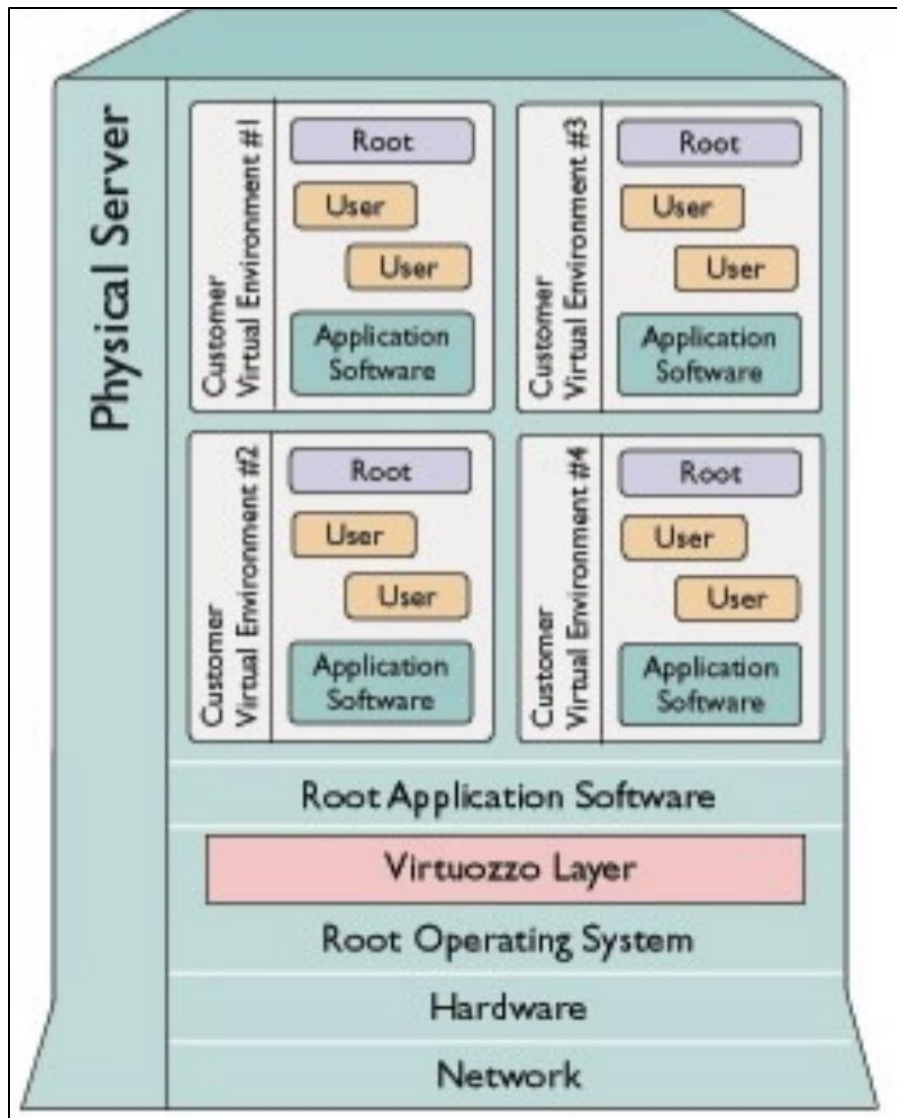
F. Virtuozzo

206. Virtuozzo, originally released by SWsoft, traces its lineage to mainframe logical-partition (“LPAR”) concepts developed in the 1970s and 1980s, implementing the

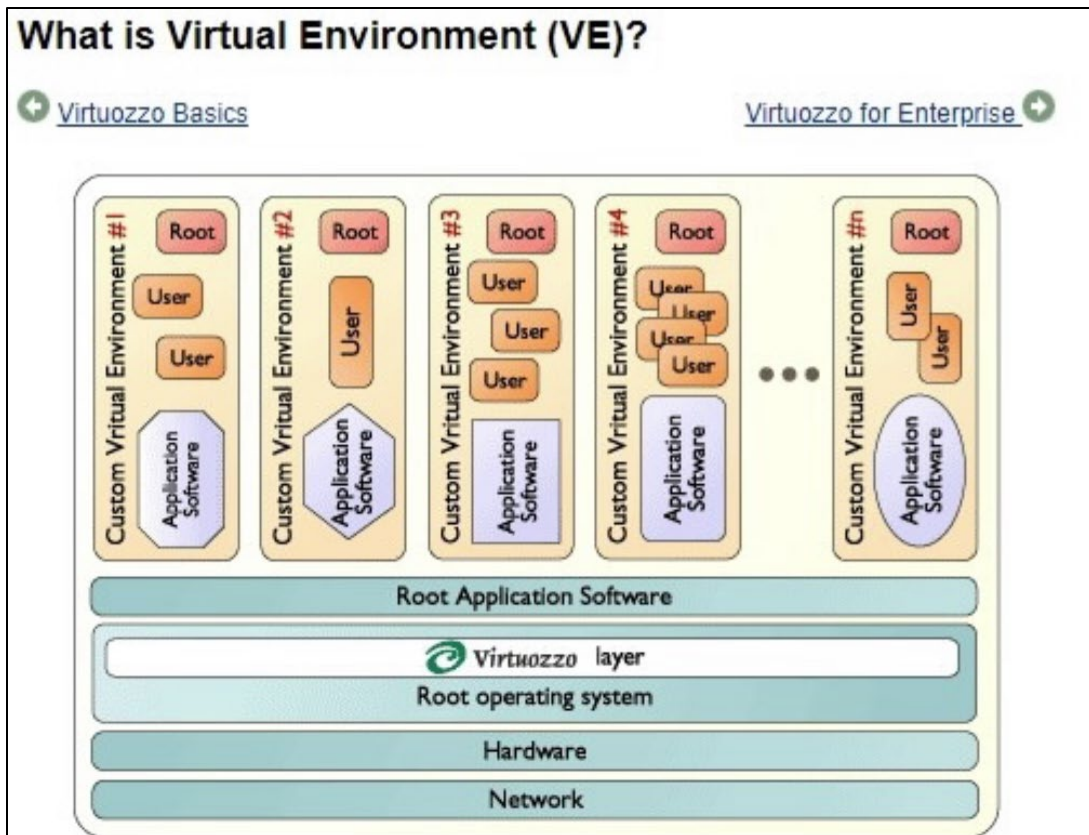
“[m]ainframe-like partitioning and manageability on commodity hardware and operating system” platforms. IBM_VM_000012876. Commercial availability of the product began in approximately 2001 for Linux servers, was extended to Solaris and FreeBSD, and by 2004 had become the “world’s most popular platform for Virtual Private Servers (VPS).” IBM_VM_000012889. It provides a “Virtual Environment (VE)” model that can “run multiple applications on the same server with advanced resource scheduling.” *Id.* By 2004, hosting service providers began deploying thousands of VEs on standard Intel hardware, IBM_VM_000012876, demonstrating carrier-grade scalability.

207. Virtuozzo was publicly available at least by January 10, 2002. *See* IBM_VM_000012862.

208. The components of the Virtuozzo system and deployment of VEs are depicted in the figures below.



IBM_VM_000012884.



IBM_VM_000012878.

209. Virtuozzo inserts a lightweight kernel extension into the host OS, which operates to ensure isolation between different containers. The VE is a full-featured OS box that is multiplied on the host, wherein each of the VEs include: independent process tables, user and group IDs, root privileges, as well as dedicated IP addresses, routing tables, firewall rules, memory and bandwidth guarantees, and other critical system resources. IBM_VM_000018171 at 172; IBM_VM_000012884. The system further provides “mainframe-like level of cluster resources utilization, monitoring and control on standard, inexpensive hardware” *Id.* A “sophisticated versioning database for all system files, libraries, and applications” permits mass deployment and single-command updates “to thousands of VEs.” IBM_VM_000012876. Application templates use shared, read-only libraries stored once on disk and mapped into each VE, dramatically reducing storage footprint. *Id.* Moreover, Virtuozzo can “[d]rag and drop a

live site or hundreds of running applications to another physical server ... for planned system maintenance” with “only a few seconds of planned downtime.” *Id.* A graphical and CLI toolset enables central administration of “hundreds of partitions across multiple physical servers.” *Id.* And through copy-on-write techniques, multiple VEs map the same read-only library pages; only modifications are stored privately, yielding high consolidation ratios. IBM_VM_000133698 at 699 (describing how Virtuozzo can “create and run hundreds of containers on a standard production physical server”); IBM_VM_000012889 (describing how a single server can host “over 1,000 VEs”).

210. Virtuozzo’s containerization model differs fundamentally from hardware virtualization. Rather than emulate devices, Virtuozzo leverages namespace and cgroup-style isolation to multiplex a single running kernel. Each containerized process “appears to have its own processes, users, files and provides full root access,” IBM_VM_000018171, maintaining distinct IP stacks with “port numbers, tables, filtering and routing rules” and establishing isolation and protection from cross-tenant interference. IBM_VM_000012878; IBM_VM_000012884 at 885. These mechanisms collectively enable service-provider multitenancy and enterprise consolidation while maintaining near-native performance. IBM_VM_000012886 at 888. Use cases of Virtuozzo’s technology include dev/test sandboxing, high-availability clusters, and educational environments where “each student gets its own virtual server with root access.” *Id.* at 886.

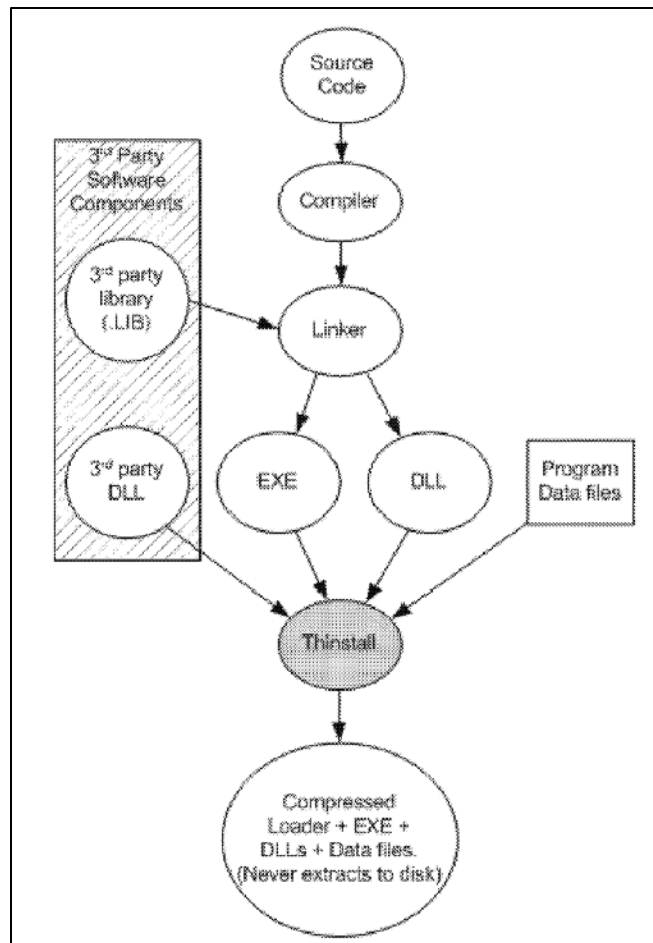
G. Thinstall/ThinApp

211. Thinstall is an application-level virtualization platform that repackages a complete software stack—including the primary executable, dependent DLL/OCX components, ancillary utilities, data files and even registry information—into a single, self-contained, compressed executable that runs directly from disk or network locations without traditional

installation steps. IBM_VM_000011520; IBM_VM_000011536. It provides a robust, portable and security-hardened containerization layer that simplifies distribution, enhances compatibility across heterogeneous Windows estates, and preserves performance while shielding intellectual property.

212. Thinstall was publicly available at least by July 30, 2003. *See* IBM_VM_000012628.

213. The system operates by embedding a lightweight “Virtual Operating System” (VOS), typically 100-300 KB on disk and ~1 MB in RAM for each process, inside every packaged application, allowing the program to execute in an isolated sandbox that transparently merges virtual and physical resources. IBM_VM_000011536; IBM_VM_000011532. Depictions of the Thinstall components used to create its compressed executable and how it interfaces with Windows are provided below.



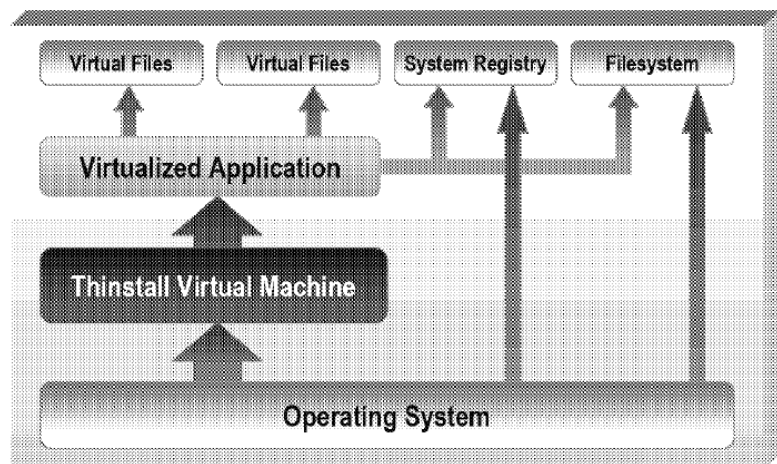
IBM_VM_000011527.

The *Thinstall Virtual Machine*, only 100-300k in size on disk, consists of 3 core technologies, the Virtual Operating System (VOS), Virtual File system (VFS) , and Virtual Registry (VREG). *Thinstall Studio* packages the VOS, VFS, and VREG along with the application EXE, DLLs, data files, and registry keys into a single compressed EXE. When the application is running, virtual files and registry keys will override system files and registry keys where they might collide, otherwise the application can access any system files and registry keys it normally could.

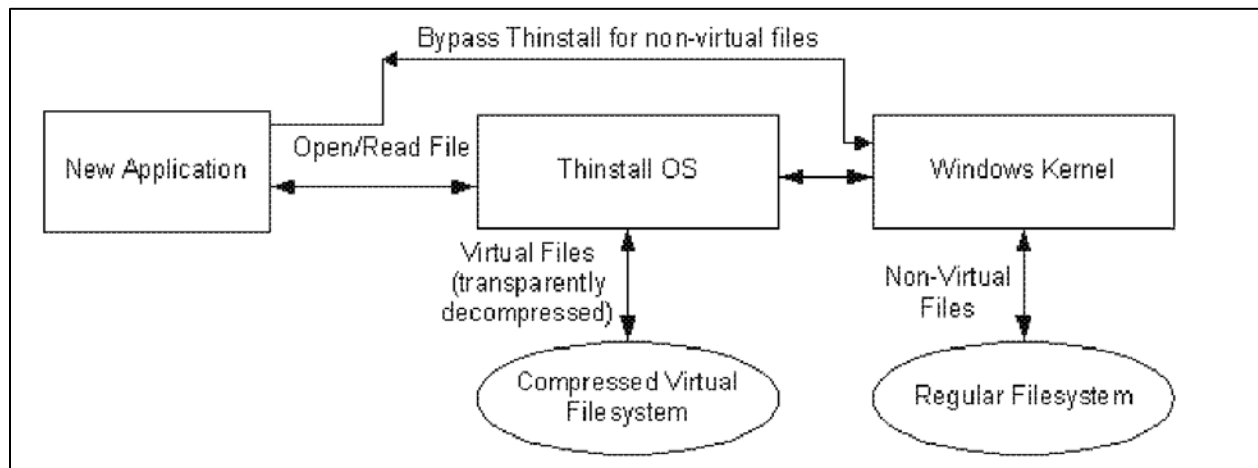
The Virtual Operating System (VOS)

The VOS allows the application to load and run COM, DLL, and EXE files directly from the created package file, bypassing Windows normal loading procedures for DLL loading and COM object creation.

Thinstall's Virtual Machine VM is responsible for loading virtualized applications into memory and managing the virtualized file system and registry.



IBM_VM_000011536 at 538.



IBM_VM_000011506 at 508.

214. At runtime, the VOS exposes three core subsystems: a Virtual File System (VFS) that presents archived files as if they existed on the local drive, a Virtual Registry (VREG) that super-imposes application-specific keys over the machine hive, and a process/DLL loader that intercepts more than 300 Win32 APIs to redirect file, registry and module operations back into the virtual layers. IBM_VM_000011536; IBM_VM_000013192. Because the VFS streams compressed blocks on demand, most file interactions incur minimal I/O while avoiding any permanent footprint; optional cache locations can be encrypted and purged on exit or uninstall. IBM_VM_000011506; IBM_VM_000012630. The VREG allows COM and ActiveX components to function without system registration and prevents clashes with globally installed versions, thereby eliminating “DLL Hell” on all Windows releases from 95 through XP and Server 2003. IBM_VM_000011506; IBM_VM_000011536.

215. Thinstall’s container executes entirely in user space, requires no kernel drivers, and can run under Guest or restricted accounts. IBM_VM_000011532; IBM_VM_000011536. Multiple virtualized applications on the same machine maintain isolated dependency chains, enabling concurrent side-by-side use of conflicting runtimes such as different .NET, Java or

MDAC versions without code changes. IBM_VM_000011532. The technology also supports application streaming: individual files or DLLs may be mapped to HTTP/HTTPS URLs and downloaded the first time they are referenced. IBM_VM_000011506.

216. Performance overhead in Thinstall is intentionally lightweight. Because the container reads compressed data in large sequential blocks, disk seek counts drop sharply and launch times often improve; Microsoft Office, for example, loads twice as fast when packaged. *Id.* CPU execution remains native, with no emulation layer, so compute-bound operations run at full speed; only virtualized file and registry calls incur a minor indirection, and empirical profiling shows a fixed memory delta of roughly 700 KB per process plus 64 KB per concurrently open virtual file. IBM_VM_000013191; IBM_VM_000011536. Where the host is under extreme memory pressure, Thinstall's private pages may be swapped to disk rather than discarded, yet this impact is mitigated because up to 90 % of DLL pages are modified by relocation anyway. IBM_VM_000013191.

217. Security features include automatic PE-header stripping, optional encryption of cached files, process anti-debug hooks, and the ability to encapsulate Java bytecode or .NET assemblies so that only a small stub is visible to external dump utilities, greatly complicating reverse-engineering attempts. IBM_VM_000011506; IBM_VM_000012630. Additional commercial controls—trial-period enforcement, hardware-locked licenses, serial-number APIs and anti-tamper heuristics—can be embedded directly in the stub without altering application source. *Id.*

218. From a deployment perspective, Thinstall adds a GUI or silent one-click installer when desired, but even that installer never fully extracts program files; it merely copies the packed executable and optional shortcuts, writes uninstall metadata, and exits—achieving installation

times 100-times faster than conventional setups and reducing disk usage by 50 % because the payload remains compressed. *Id.* Uninstall is equally trivial, deleting the single EXE and any cached artifacts under the user-defined directory. IBM_VM_000012630.

219. Finally, Thinstall can be used to complement rather machine-level hypervisors such as VMware or Virtual PC: where those solutions virtualize entire operating systems at considerable storage and licensing cost, Thinstall delivers per-application isolation with negligible disk overhead and no requirement for additional OS images. IBM_VM_000011532.

H. VServer (“Gélinas”)

220. Linux-VServer is a kernel-level containerization technology that “creates many independent containers under a common Linux kernel” and presents each container “just like a separate host” to applications and their users. IBM_VM_000181672. Conceived in the early 2000s by Jacques Gélinas of Solucorp, the project was publicly announced as an enhancement to Linux 2.4.11/2.4.18 that “allow[ed] several independent virtual servers running on the same box (sharing the same kernel as well).” IBM_VM_000016393. Gélinas introduced two new system calls—*new_s_context* and *set_ipv4root*—together with an expanded capability ceiling, thereby delivering an operating-system-level virtualization mechanism that required no emulation layer and imposed “negligible overhead compared to a vanilla kernel.” IBM_VM_000181672 673.

221. Linux-VServer was publicly available at least by April 4, 2002. *See* ORACLE_VM_PA_00001491 at -1518.

222. From its inception, the design objectives were security isolation, resource efficiency, and administrative flexibility. A virtual server (“vserver”) is isolated along five axes: filesystem, processes, networking, super-user capabilities, and System V IPC. The project documentation explains that a vserver “is trapped into a sub-directory of the main server and

can't escape" via the traditional *chroot()* call; it "can only see the processes in the same security context"; it "is assigned a host name and an IP number" and may bind only to that address; its root user has "less privileges than the normal Linux root user"; and its SysV IPC resources are keyed to the security context. ORACLE_VM_PA_00001491 at 492-493.

223. Operationally, Linux-VServer employs context isolation rather than full hardware virtualization. The kernel tags every process, node, socket and IPC object with a context identifier. IBM_VM_000181672. When a process attempts to enumerate or interact with resources, kernel filters compare context IDs and silently hide objects that belong to a different container. *Id.* Because the host kernel remains single and unmodified for guests, execution cost is effectively native: benchmark work cited in the same paper found "in the worst case a 4 percent overhead" and in most cases "performance ... is indistinguishable" from an unvirtualized system. *Id.* at 675, 677.

224. The functional components that realize this architecture include:

- **Kernel patch set (≈17 KLOC)** – adds context tags, capability ceilings, the *new_s_context* and *set_ipv4root* system calls, per-context limits, scheduler filters, and network-stack hooks (*Id.* at 673).
- **User-space utilities** – *chcontext* to enter a security context, *chbind* to pin a process to a specific IP, *reducecap* to lower capability ceilings, and wrapper commands (*vps*, *vpstree*, *vserver*, etc.) that simplify management (ORACLE_VM_PA_00001491 at 503-504).
- **Filesystem unification tools** – *unify* hard-links identical files across guests while marking them immutable, shrinking a 2 GB reference installation to "38 megs" per guest and achieving dramatic memory-sharing benefits (*Id.* at 500; IBM_VM_000181672 at 678).
- **Per-context resource accounting** – token-bucket CPU scheduler delivering "fair-share, work-conserving, or hard limit" reservations (IBM_VM_000181672 at p. 4) and storage/memory quotas tied to security context IDs (ORACLE_VM_PA_00003709 at 727).
- **Administration scripts** – boot-time scripts to start all vservers, per-vserver hooks for pre-start/post-stop actions, and *vreboot/vhalt* helpers that let a container reboot itself without affecting the host (ORACLE_VM_PA_00001491 at 504).

The system's capability to combine strong isolation with native performance made it attractive for web hosting centers, research testbeds and embedded gateways. Production deployments cited include PlanetLab, where "a large number of VPSs can be robustly hosted on a single machine ... it is not uncommon ... to pack 200 VPSes onto a single machine"

(IBM_VM_000181672), and the OLPC gateway servers, where low-power hardware required “a resource efficient solution like Linux-VServer rather than Xen”
Id.

225. In summary, Linux-VServer originated as a lightweight patch set that inserted context isolation primitives into the Linux 2.4 kernel, and it matured into a comprehensive container platform whose efficiency and security properties were—by contemporaneous measurements—comparable to running directly on bare metal. *Id.*; *Id.* Its design demonstrates that by constraining processes with irreversible system-call transitions and augmenting the kernel with per-context resource governance, a single Linux instance can provide multiple, mutually isolated virtual servers that administrators may manage independently yet operate at full native speed.

I. US Patent No. 6,529,985 (“Deianov”)

226. US Patent No. 6,529,985 (“Deianov”) was filed on February 4, 2000, and issued on March 4, 2003. IBM_VM_000012597 (Deianov). It is assigned to Ensim Corporation, a provider of provisioning and management technology for cloud-based applications. Deianov describes a system and method for the selective interception of system calls made by specific processes in multitasking operating systems, providing a flexible, secure, and stable method for selectively intercepting system calls in multitasking operating systems.

227. The invention enables certain processes to have their system calls intercepted and handled by custom “system call wrappers,” while other processes continue to use the default system calls. Unlike previous methods, which intercepted system calls for all processes indiscriminately, Deianov allows interception to be limited to specific processes, providing greater flexibility and security. IBM_VM_000012597 (Deianov) at 602-03. This selective interception enhances security, stability, and ease of development, particularly in environments where customization of operating system functionality is required, such as server computers.

IBM_VM_000012597 (Deianov) at 601-02. Moreover, by allowing system call wrappers to execute in user space rather than operating system address space, the invention described in Deianov reduces the risk that errors in wrapper code will affect the entire system, thus improving stability and making development and debugging safer and easier. IBM_VM_000012597 (Deianov) at-602-03. A depiction of one embodiment of Deinov's architecture is shown in Figure 1, below:

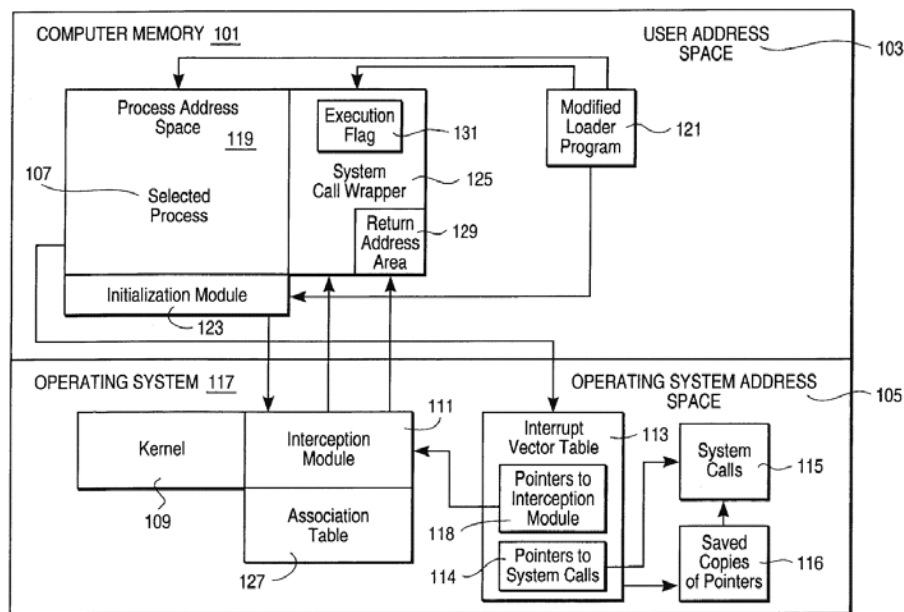


FIG. 1

IBM_VM_000012597 (Deianov) at Fig. 1.

228. As shown in the above embodiment, computer memory is divided into user address space (103), where user processes execute, and OS address space (105), where the OS kernel and related modules reside. Selected Process (107) represents a user-level process executing in user address space, and can be comprised of multiple processes, each with its own wrapper and initialization module. IBM_VM_000012597 (Deianov) at 604. Interception module (111) can be dynamically loaded into the OS, and is responsible for determining, at runtime, whether a system call made by a process should be intercepted and redirected to a wrapper, or passed

through to the default system call. IBM_VM_000012597 (Deianov) at 604. The interception module (111) maintains an association table (127) that maps selected processes to their corresponding system call wrappers, enabling selective interception. Interrupt Vector Table (113) contains pointers to system call routines that include pointers to the interception module and pointers to system calls, copies of which are saved. IBM_VM_000012597 (Deianov) at - 604. The modified loader program (121) is used to load selected processes, including an initialization module (123) for registering the process and its wrapper with the interception module and a system call wrapper (125) to be executed for selected processes. IBM_VM_000012597 (Deianov) at 604. System call wrappers are alternative object code routines that execute instead of the default system call. IBM_VM_000012597 (Deianov) at 602. The return address area (127) stores the address to which control should return after the wrapper executes.

229. Operationally, the system described by Deianov uses an interception module loaded into the operating system for selective interceptions of system calls. IBM_VM_000012597 (Deianov) at 603. Pointers in the interrupt vector table to system calls that are to be intercepted are replaced with pointers to this interception module. IBM_VM_000012597 (Deianov) at 597. When a process makes a system call, the interception module determines whether the process is one of the selected processes and, if so, executes the associated system call wrapper. IBM_VM_000012597 (Deianov) at 597, 600-03. System call wrappers are alternative object code routines that execute instead of the default system call. IBM_VM_000012597 (Deianov) at 602. These wrappers can be loaded into the process address space of selected processes, into user address space (but not tied to a specific process), or into operating system address space, depending on the embodiment. IBM_VM_000012597 (Deianov) at 597, 601, 603. This

association is managed by a modified loader program, which loads the process, the system call wrapper, and an initialization module into the process address space. The initialization module registers the wrapper with the interception module, which maintains an association table mapping processes to their wrappers. IBM_VM_000012597 (Deianov) at 601, 603-04.

J. Eggert

230. The paper entitled “File Systems in User Space” was authored by Eggert and Parker and published as part of the USENIX Winter 1993 Conference between January 25-29, 1993. IBM_VM_000012090 (“Eggert”). Eggert introduces a then-novel approach to file system extensibility by allowing users to define and manipulate file system behavior entirely in user space, without requiring kernel modifications or root privileges, thereby democratizing file system extension. IBM_VM_000012090 (Eggert) at 90-91. The technology described by Eggert was implemented on SunOS 4.1.x. IBM_VM_000012090 (Eggert) at 98.

231. As Eggert explains, then existing methods for extending or modifying file systems to user programs “suffer two major drawbacks: they require kernel modifications or root privileges, and they are too complicated to be given to ordinary users,” thereby impeding widespread adoption. IBM_VM_000012090 (Eggert) at 90. For example, adding new naming services or supporting shell-like expressions in path names typically involved duplicative and ad hoc modifications across many applications, as the kernel and file system interfaces did not natively support such features. IBM_VM_000012090 (Eggert) at 90-91. Other approaches, such as stackable file systems or file system interface mapping, either required expert knowledge, root permissions, or kernel changes, and were not designed for end-user extensibility. IBM_VM_000012095 (Eggert) at 96.

232. The technology proposed by Eggert leverages the rise of dynamic linking to interpose a layer of user code between applications and the system call interface, enabling users to alter

or extend file system behavior systematically and safely in user space. IBM_VM_000012090 (Eggert) at 90-91, 97. This is achieved through the development of two prototype systems, IFSO and IFS1, which allow users to define new file naming rules and behaviors using shell-like expressions and user-defined naming escapes, without writing C code or modifying the kernel. IBM_VM_000012090 (Eggert) at 90-93. These systems support a wide range of extensions, such as on-demand file generation, versioning, remote storage, encryption, and archive navigation, by interpreting pathnames as expressions that can be evaluated in context. IBM_VM_000012090 (Eggert) at 91-95.

233. The core mechanisms enabling this user-space extensibility described in Eggert includes the use of shared libraries and dynamic linking. By manipulating environment variables, users can instruct the dynamic linker to preload a specially crafted library (such as IFS1) before the standard C library is loaded, causing the system to intercept and potentially modify system calls. IBM_VM_000012090 (Eggert) at 97. This interception is achieved without kernel changes: the preload library wraps system calls like open and stat, and only intervenes when a pathname does not exist or matches certain patterns (e.g., contains shell expressions or special escape characters). IBM_VM_000012090 (Eggert) at 97. The approach is lightweight, adding minimal overhead in the common case, and is transparent to unmodified applications, as all dynamically linked programs automatically adopt the new file system semantics when the preload library is active. IBM_VM_000012090 (Eggert) at 97.

K. Levine (“Linkers and Loaders”)

234. The text “Linkers and Loaders” was written by John Levine, bears a copyright date of 2000, and was published by at least 2001. IBM_VM_000013250 (“Levine”). In my opinion, it is particularly reflective of what a person of ordinary skill would have understood about the construction, operation, and use of linking and loading mechanisms, including with respect to

the topics of loading and overlays (Chapter 8), shared libraries (Chapter 9), and dynamic linking and loading (Chapter 10).

235. Levine explains how a shared library of code and/or data can be mapped into the address space of multiple processes, allowing a single physical copy of instructions to be shared. IBM_VM_000013250 (Levine) at 449, 463. For example, shared objects can be assigned fixed virtual address ranges, so all processes can map them at the same location, avoiding load-time relocation. IBM_VM_000013250 (Levine) at 449. Administrators can maintain a global table to allocate address space among libraries. *Id.*

236. Levine further demonstrates how shared-object technology enables operating systems and runtimes to be modular and extensible. IBM_VM_000013250 (Levine) at 199, 216, 223. For example, Levine describes how functional overlays can be loaded using mechanisms such as LD_PRELOAD to insert custom libraries ahead of system components. IBM_VM_000013250 (Levine) at 199, 216. As explained, pre-loaded libraries allow different programs to use different implementations of the same API, supporting staged upgrades, A/B testing, or compatibility shims. IBM_VM_000013250 (Levine) at 199. Such servicing can be offloaded from the kernel space to the user space by, for example, mapping shared objects (e.g., vDSO) into every process, exposing high-frequency system calls without kernel/user mode switches. IBM_VM_000013250 (Levine) at 223. Levine further describes various dynamic linking and function loading mechanisms. IBM_VM_000013250 (Levine) at 213-23.

L. U.S. Patent Application Publication 2002/0174215 (“Schaefer”)

237. U.S. Patent Application Publication 2002/0174215, entitled “Operating System Abstraction and Protection Layer,” published on November 21, 2002, and invented by Stuart Schaefer, describes a client-side virtualization framework—termed the “Operating System

Guard”—that inserts a transparent software layer between any executing application and the underlying operating system. ORACLE_VM_PA_00006280. at Abstract. The guard constructs a self-contained “virtual environment” in which the application perceives a fully installed operating-system configuration even though the host system itself remains untouched. *Id.* The application explains that the overarching goal of the invention is to deliver, isolate, and manage applications on-the-fly, without the conflicts or administrative overhead typical of conventional installations. *Id.* at [0003].

238. The Guard operates by trapping or “hooking” system-level calls and servicing those calls within its own data structures unless policy dictates that they be passed through to the real operating system. *Id.* at [0005]. For registry activity, this produces a two-stage model: keys required by the application can be supplied from a virtual registry held entirely by the Guard; any attempted modification is likewise captured and stored virtually, leaving the physical registry pristine. *Id.* at [0027].

239. As described, file operations are redirected through a virtual file-system manager. Prior to first launch, the Guard can preload a map of files. *Id.* at [0040]. Then, at runtime, create/open requests are seamlessly pointed to the designated virtual counterparts. *Id.* at [0040]. Read-only mirroring, copy-on-write, and intelligent redirection enable the application to believe it is working against local disk locations while the host disk is never altered (or, where alteration is unavoidable, only modified in Guard-controlled staging areas). *Id.* at [0041]. Redirected files can be renamed or relocated to avoid path collisions and conflicts, and differential changes are persisted for future sessions. *Id.* at [0041].

240. The '215 application further describes how shared libraries (DLLs on Windows, .so files on UNIX, INITs on Macintosh) are also virtualized. *Id.* at Abstract. The Guard maintains

version rules for each application; during a LoadLibrary event it can (i) rewrite search paths, (ii) dynamically edit “known[] DLL” registry keys, (iii) substitute symbolic links, or (iv) supply the correct binary directly from its own store so that the appropriate version is loaded without overwriting a system-wide copy. *Id.* at [0035]. Special-case frameworks such as OLE, ODBC, or MDAC are supported by ensuring that the correct version is loaded privately yet still shares memory or inter-process data with any co-resident components that require the same build. *Id.* at [0037].

241. Device drivers, fonts, and environment variables are brought under similar control. *Id.* at [0044]-[0045]. Where a driver can be loaded dynamically, the Guard loads and later unloads it on the application’s behalf; if static installation is required, the Guard schedules installation/removal at the next reboot and records the action for automatic cleanup. *Id.* at [0044]-[0045]. Fonts are copied into and registered from a virtual store, then deregistered and deleted (or redirected) when no longer required. *Id.* at [0052]. Environment variables needed by the application are injected into the process space at launch without altering system-wide settings. *Id.* at [0054].

242. Internally, the Guard is organized into several cooperating subsystems: (i) a Core that tracks each application and its context; (ii) a Configuration Manager governing registry and INI virtualization; (iii) a File Manager implementing the virtual file system; (iv) a Shared Object Manager handling DLL/so versioning; (v) Device and Font Managers; (vi) a Process Manager that groups processes into “application bundles” so multiple processes (or multiple user sessions) can share or segregate virtual resources as required; (vii) a Loader that serializes, imports, exports, and stages virtual environments; and (viii) a Recovery Manager maintaining

a transaction log so that any change, even one in progress during a system crash, can be rolled back, guaranteeing no residual impact on the host operating system. *Id.* at [0022]-[0024].

243. The Guard supports fully private contexts (each application instance sees its own virtual registry, files, libraries, etc.) or controlled shared contexts (e.g., letting Word and Excel share Office-wide settings). *Id.* at [0019]-[0020]. Administrators or wrapper tools can assemble these contexts automatically. *Id.* at [0020]. During an installation capture or monitored first run, the Guard records every registry, file, driver, and font interaction, then converts the observations into configuration files. *Id.* at [0061]. Rules-based post-processing lets administrators fine-tune which elements remain private, which are merged with others, and which are locked by policy. *Id.* at [0062].

244. Collectively, the disclosed technology enables “pseudo-installations” that appear complete to the application, allow simultaneous execution of multiple versions or instances, isolate users from one another, and guarantee automated rollback of all system modifications, thereby improving reliability, security, and ease of software deployment. *Id.* at [0007].

M. U.S. Patent No. 5,961,582 (“Gaines”)

245. “Distributed and Portable Execution Environment,” U.S. Patent No. 5,961,582 (“Gaines”), issued on October 5, 1999, was invented by R. Stockton Gaines, and was assigned to Acorn Technologies, Inc. Gaines discloses an expansive architecture that allows users, whether local or remote, to run programs on heterogeneous computers through a virtual operating system layer that abstracts the underlying hardware and native operating systems. This layer enables seamless, secure, and portable execution of applications across networks.

246. At its core, the invention overlays a host virtual operating system (“VOS”) on a conventional host computer. The VOS sits between user-level programs and the native operating system (“first operating system”) that controls the processor, memory, file system,

and peripheral devices. Through a dedicated virtual application interface, the VOS exposes a standardized, machine-independent set of resources—process control, a virtual file system, inter-process communications, device interfaces, GUI services, and database services—so that programs written once can execute unaltered on a multitude of underlying platforms. Each VOS instance carries a unique identifier and a self-verification checksum so that tampering can be detected at load time. Usage is metered based on the VOS’s charging record reflecting resource consumption or value-added services rendered (e.g., advertising, brokerage transactions).

247. Various filters disclosed in Gaines ensure that virtualization is both comprehensive and secure:

- **Process-control filter** – Implements a virtual process model. It mirrors creation, scheduling, and termination of “virtual processes” to real host processes while enforcing resource quotas (e.g., CPU time, memory) and security policies. Inter-process communication primitives are provided, letting virtual applications converse whether they reside on the same host or different hosts.
- **File-system filter** – Presents a hierarchical virtual file system that can be mapped selectively onto the host’s real file system. Some virtual files correspond directly to actual files, but others are synthesized or redirected, enabling transparent resource sharing without revealing sensitive host paths. The embodiment leverages concepts akin to the Prospero file system.
- **GUI filter** – Supplies uniform graphical user-interface services. Input and output requests are funneled through virtual devices (keyboard, mouse, display) and translated into host-specific calls, allowing applications to provide familiar windows and graphics regardless of the local windowing environment.
- **Database filter** – Grants controlled SQL-type access to relational databases, mediating every command, restricting visibility to authorized schema elements, and translating permissible requests into operations on the host file system or external data sources.

248. As Gaines explains, these filters collectively act as guardians: every call from an application is vetted against a permissions set defined by the host administrator. The administrator can confine programs to a safe “sandbox,” preventing unauthorized file access, network misuse, or excessive resource consumption.

249. Gaines further explains how execution of applications is language-agnostic yet interpreter-friendly. The language engine strips or redirects native file and process operations, compelling the program to invoke VOS primitives instead. Consequently, the same script can run unchanged on PCs, UNIX workstations, or mainframes.

250. To extend these capabilities beyond a single host, the patent describes a distributed execution environment. Multiple hosts, each running its own VOS, are networked. A user at a client machine launches a user interface—optionally a graphical shell or an integration into web tools like Mosaic—to migrate a “transferable program” to a chosen server host. Selection may depend on live metrics (bandwidth, cost, CPU load) stored in a local database of hosts. The server-side VOS verifies the program, applies Kerberos-style authentication, and enforces constraints before execution. The program can then spawn additional transferable programs on other hosts, forming a mobile, self-propagating application that cooperatively processes data or searches for resources across the network.

N. U.S. Patent No. 7,673,308 (“McMillan”)

251. U.S. Patent No. 7,673,308, titled “Virtual OS computing environment” (“’308 Patent” or “McMillan”) was invented by John McMillan and Gary Chirhart, was issued on March 2, 2010, and was assigned to Symantec Corporation. IBM_VM_000011947. The application resulting in the ’308 Patent (Appl. No. 10/716,337) describes the same invention and was filed on November 18, 2003, which was published as US 2005/0076326 on April 7, 2005. IBM_VM_000013963. The patent claims priority to provisional application 60/427,339 filed on November 18, 2002.

252. I understand that a patent properly claims priority to a provisional application when the provisional application contains a written description of the invention claimed in the patent. *See, e.g., Dynamic Drinkware, LLC v. Nat’l Graphics, Inc.*, 800 F.3d 1375 at 1378, 1831-82

(Fed. Cir. 2015). McMillan claims priority to Provisional Application No. 60/427,339 (the “’339 Application”). McMillan’s specification is almost the same as the specification of the ’339 Application, except that McMillan adds sections which have the headers “Reference to Related Application,” “Field of the Invention,” and “Background of the Invention.” However, none of these sections add anything substantive to the invention disclosed in the ’339 Application. *Compare* IBM_VM_000011947 *with* Provisional Application No. 60/427,339. Reference to Related Application is totally non-substantive, Field of the Invention is merely a categorization of the technology, and Background of the Invention adds discussion of the state of the art at the time of the invention. The specification of McMillan, which is shared with the ’339 Application, fully supports at least claim 1 of McMillan. *See, e.g.,* IBM_VM_000011947. Thus, McMillan is entitled to the ’339 Application’s priority date.

253. McMillan describes a method and system for creating multiple, semi-independent virtual operating system (OS) environments within a single base OS. Each virtual OS environment operates as if it were an independent OS for the applications running within it, but all environments coexist on the same underlying OS. As described, the system allows for the creation of several virtual OS environments within a single base OS. IBM_VM_000011947 (McMillan) at 952. Each environment has its own independent file system and registry, separate from the base OS and from each other. IBM_VM_000011947 (McMillan) at 952. This ensures that changes made in one environment do not affect the main OS or any other environment. Each environment can contain a group of installed applications that run independently of those in other environments or the base OS. IBM_VM_000011947 (McMillan) at 952. Each virtual OS environment is isolated such that changes to configuration, file system, or registry information are made only within that

environment and do not affect the base OS or other virtual environments. IBM_VM_000011947 (McMillan) at 951-52. This isolation is achieved by redirecting all relevant API calls and maintaining separate copies of the file system and registry for each environment. IBM_VM_000011947 (McMillan) at 952.

254. Since all virtual OS environment data is contained within isolated sections of the base OS file system and registry, McMillan explains that entire environments can be easily saved, stored, and loaded. IBM_VM_000011947 (McMillan) at 953. This enables the creation of pre-configured or "clean" virtual OS environments that can be reused or distributed across multiple computers. Moreover, by maintaining separate copies of built-in OS applications within each virtual OS environment, the system allows multiple versions of such applications to be supported and run simultaneously in different environments. IBM_VM_000011947 (McMillan) at 953.

255. As described, the claimed system determines which environment an application should run in based on the location of the application's executable file. IBM_VM_000011947 (McMillan) at 953. The base OS maintains a list of directories associated with specific virtual OS environments. IBM_VM_000011947 (McMillan) at 953. Applications executed from these directories, or their subdirectories, are redirected to the corresponding virtual OS environment. IBM_VM_000011947 (McMillan) at 953. Shared directories can be specified to allow certain files to be accessed by both the base OS and virtual environments, facilitating tasks like software installation.

256. McMillan further describes how embodiments of the claimed invention use injected dynamic link libraries (DLLs) to redirect application calls that would normally access the base OS file system or registry. IBM_VM_000011947 (McMillan) at 952. When an application is

executed, a DLL is injected into its process. IBM_VM_000011947 (McMillan) at 952. This DLL determines whether the application should run under the base OS or a virtual OS environment. IBM_VM_000011947 (McMillan) at 952. If redirection is needed, the DLL scans the application's function import table and redirects all file system and registry API calls to functions within the injected DLL. IBM_VM_000011947 (McMillan) at 952. These functions modify the parameters of the API calls to point to the appropriate virtual OS environment locations before passing them to the real OS APIs. IBM_VM_000011947 (McMillan) at 952. The DLL also modifies returned information to convert it back from the virtual OS environment to the application's expected format.

O. US Patent Application Publication 2002/0124072 (“Tormasov”)

257. US Patent Application Publication 2002/0124072 (“Tormasov”), entitled “Virtual Computing Environment”, was published on September 5, 2002. IBM_VM_000012590 (Tormasov). The '072 Publication later issued as U.S. Patent No. 7,009,948, assigned to SWsoft Holdings, provider of the Virtuozzo system. The named inventors of the '072 Publication include Alexander Tormasov and Dennis Lunev, who developed the Virtuozzo technology and later open sourced it under the name OpenVZ.

258. The invention describes a system and method for creating multiple virtual computing environments managed by a single operating system kernel. Each virtual environment provides users with the experience of having their own independent system, but without the need for hardware emulation or dedicated physical resources. The invention described in Tormasov distinguishes itself from earlier solutions such as hardware partitioning or software emulators (e.g., VMware), which require either multiple operating system kernels or full hardware emulation. IBM_VM_000012590 (Tormasov) at 594. The described invention instead uses a single kernel and namespace isolation. IBM_VM_000012590 (Tormasov) at 594. This

303. The patent also details a distributed architecture involving servers, gateways, and endpoints, where each component can be dynamically assigned and reconfigured, and where failover and load balancing are managed automatically to maintain service continuity IBM_VM_000022821 (Benfield) at -836-39. The system includes mechanisms for detecting component failures through polling and communication tests, starting new instances as needed, and performing self-correction by terminating duplicate or failed components. IBM_VM_000022821 (Benfield) at -841-44.

304. Additionally, the disclosed technology provides for the synchronization and validation of topology and resource status databases after failover events, ensuring that network management applications always operate on accurate and up-to-date information. IBM_VM_000022821 (Benfield) at -842-44. The patent emphasizes the importance of scalability, fault tolerance, and the ability to manage heterogeneous and geographically dispersed environments, making it suitable for large enterprises and service providers (IBM_VM_000022821 (Benfield) at -834, -835, -836).

X. THE ASSERTED CLAIMS OF THE ASSERTED PATENTS ARE INVALID UNDER 35 U.S.C. § 102 AND/OR § 103

A. U.S. Patent No. 7,519,814

i. Solaris Zones and/or Tucker

305. I incorporate by reference the background of Solaris Zones/Tucker in Section IX.D. into my analysis below. I also note that Tucker alone discloses and/or renders obvious the asserted claims of the '814 patent, for the reasons discussed below. Additionally, Solaris Zones as a product, which implements the concepts taught in the Tucker Provisional, further discloses and/or renders obvious the asserted claims of the '814 patent.

a. Claim 1

306. Claim 1 of the '814 Patents states:

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;

wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

307. In my opinion, for the reasons discussed below, Solaris Zones discloses and/or renders obvious claim 1.

1[Pre.a]: “In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor,”

308. As detailed above in the section of my report regarding claim constructions, I understand the parties have agreed the preamble is limiting. I further understand the parties have agreed that “processor” means “physical computer processor” and “a processor” and “the processor” refer to the same entity. I understand that the parties dispute the construction for “disparate computing environments” where IBM proposes “environments where computers

are stand-alone or where there are multiple computers and where they are unrelated” and VirtaMove proposes “environments run by standalone computers” or alternatively “environments on standalone computers or on computers that are unrelated.” Dkt. 124, Appx. A, C. I also understand the parties dispute the construction for “system files” where IBM proposes “files within an operating system and which are available to applications as shared libraries and configuration files” and VirtaMove proposes plain and ordinary meaning. *Id.* In my opinion, Solaris Zones discloses or renders obvious 1[Pre.a] under either parties’ constructions.

309. **First**, using VirtaMove’s own application of the claims in its infringement contentions, Solaris Zones renders obvious “[i]n a system having a plurality of servers with operating systems that differ, operating in disparate computing environments.” Solaris Zones can be implemented on servers running Solaris 10 OS, and specifically teaches “the same application environment can be maintained on different physical machines.” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5468.

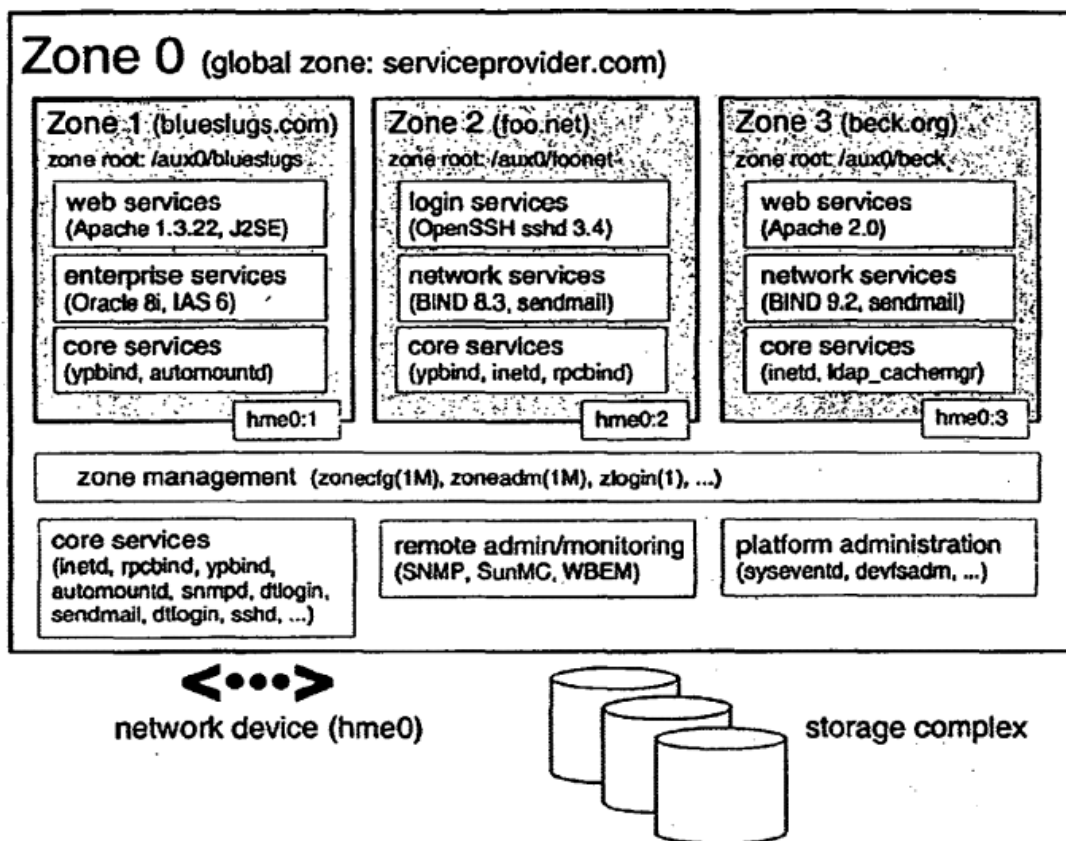


Figure 1.1: Server Consolidation Using Zones

Id. at -5469. Indeed, Solaris Zones also teaches that there is a system's "primary IP address and host name," and thus, a POSITA would understand that Solaris Zones can run on separate computers with differing IP addresses and host names. *Id.* at -5468.

310. Thus, at least under VirtaMove's interpretation, a POSITA would have found it obvious that Solaris Zones discloses or renders obvious a system having a plurality of servers with operating systems that differ, in which a system is comprised of multiple computers using version(s) of Solaris 10 operating system. Additionally, a POSITA would have understood that the plurality of servers are disparate computing environments comprising of "environments where computers are stand-alone or where there are multiple computers and where they are unrelated." Alternatively, under VirtaMove's interpretation, disparate

computing environments are “environments run by standalone computers” or alternatively “environments on standalone computers or on computers that are unrelated.”

311. **Second**, Solaris Zones renders obvious “wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor.” A POSITA would understand that Solaris Zones (running on Solaris OS) includes a processor and an OS. *See Id.* (discussing “different physical machines” zones can run on), -5537 (discussing non-global zones needing to access files delivered as part of Solaris OS), -5473 (“Zones are based on the basic idea of jails, but extend the concept to provide a comprehensive facility that is integrated with core operating system services.”).

1[Pre.b]: “a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment”

312. Solaris Zones discloses or renders obvious “a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment.”

313. **First**, Solaris Zones teaches “a method of providing at least some of the servers in the system with secure, executable, applications related to a service.” Indeed, Solaris Zones teaches that “Zones provide a means of virtualizing operating system services, allowing one or more processes to run in isolation from other activity on the system” in which “one or more applications can run” by “provid[ing] an abstraction layer that separates applications from physical attributes of the machine on which they are deployed...” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467; *see also id.* (“Zones allow the deployment of multiple applications on the same machine, even where those applications operate in different trust domains...”)(“For example, multiple applications running in different

zones (but on the same system)....”). Indeed, as illustrated in Figure 1.1 below, “Zones 1, 2 and 3 are each running a disjoint workload” which “demonstrates that different *versions of the same application may be run without negative consequence in different zones*”:

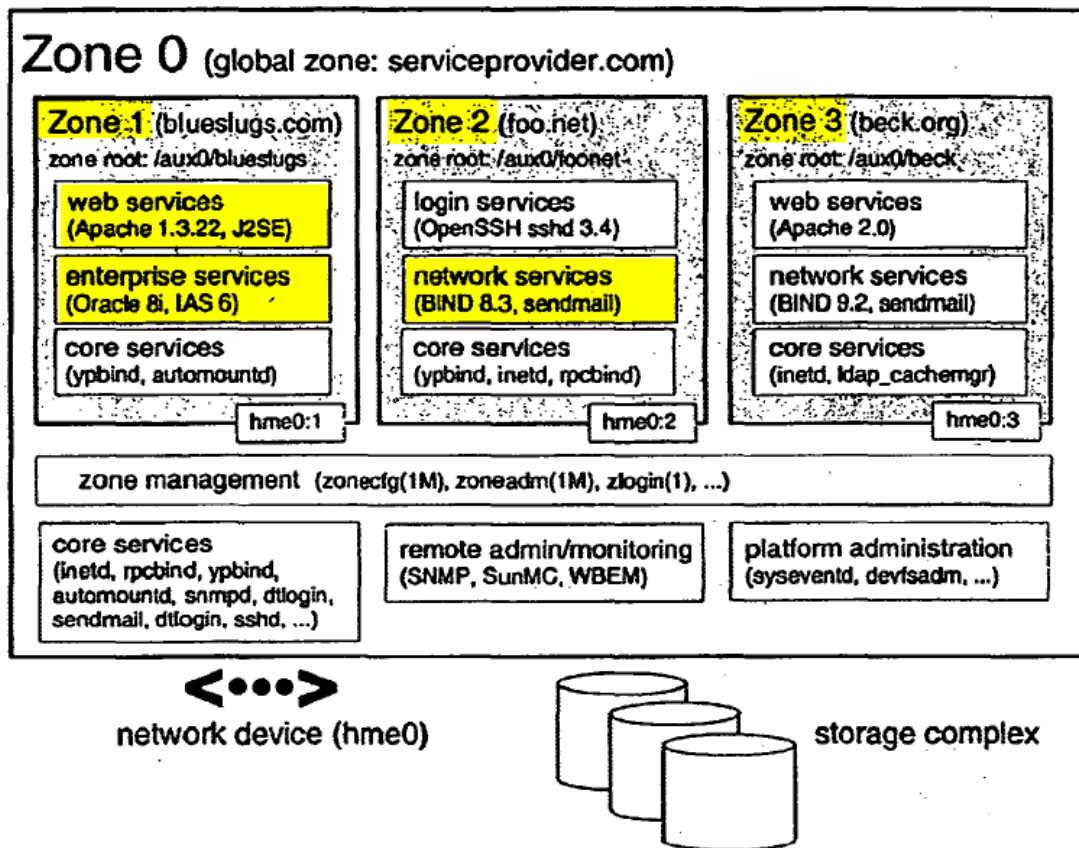


Figure 1.1: Server Consolidation Using Zones

Id. at -5468–5469 (emphasis added). As illustrated above, these applications are related to various services including “web services,” “network services,” “enterprise services,” among others. *Id.* A POSITA would also understand that each zone provides a secure, executable application related to these services. *See id.* at -5468 (“avoid changing the environment in which *applications are executing*”) (emphasis added), -5473 (“labels can be used to created isolated for containers *for applications that execute on the same system.*”) (emphasis added).

314. *Second*, Solaris Zones teaches “wherein the applications are executed in a secure environment.” Indeed, Solaris Zones expressly teaches that “Zones provide a means of virtualizing operating system services, allowing one or more processes to run in isolation from other activity on the system” and “[t]his isolation prevents processes running within a given zone from monitoring or affecting processes running in other zones.” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467. Solaris Zones further teaches that “[a] zone is a ‘sandbox’ within which *one or more applications can run without affecting or interacting* with the rest of the system.” *Id.* (emphasis added) Solaris Zones further explains the isolation of each Zone below:

1. Isolation

- The kernel exports a number of distinct objects that can be associated with a particular zone. These include processes, file system mounts, network interfaces, and System V IPC objects.
- No zone (other than the global zone) should be able to access objects belonging to another zone (including the global zone), either to control or modify those objects in some way, or to simply monitor or read them.
- As much as possible, processes in a non-global zone should not be able to interfere with the execution of processes in other zones in the system. Although it is unlikely that this project can prevent all possible active denial-of-service attacks by privileged processes in non-global zones, accidental and relatively trivial interference should be prevented.
- Appropriately privileged processes in the global zone can access objects associated with other zones. As much as possible, it should be possible to administratively and programmatically identify the zone with which each such object is associated.
- Cross-zone communication may occur over the network (which is actually looped back inside IP, as with any traffic routed between logical interfaces in the same system), but not through other mechanisms without the participation of the global zone.

Id. at -5470; *see also id.* at -5468 (“Because each zone is confined to its subtree of the file system hierarchy, the workloads cannot access each other’s on-disk data.”).

315. My review of the Solaris source code further confirms that the applications running within a zone do not interfere with others or escape their root filesystem. [REDACTED]

316. Thus, Solaris Zones discloses that the applications executed in each zone are executed in a secure environment.

1[Pre.c]: “wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:”

317. Solaris Zones discloses or renders obvious “wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising.”

318. As previously discussed and as illustrated below, Solaris Zones teaches that there are a variety of applications that can be run in zones, which include web services (Apache, J2SE), enterprise services (Oracle 8i), network services (BIND 8.3, sendmail), among others:

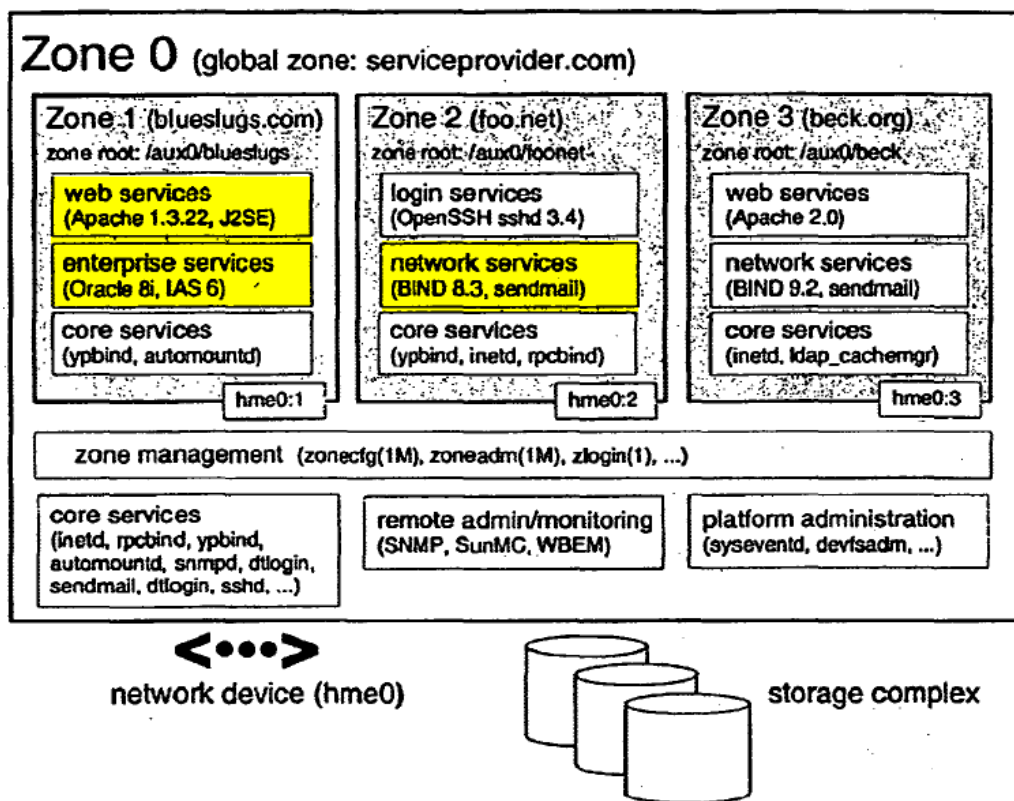


Figure 1.1: Server Consolidation Using Zones

ORACLE_VM_PA_00005455 (Tucker Provisional) at -5469 (emphasis added). A POSITA would understand that these applications each include an object executable by the OS for performing a task related to a service. For example, Oracle enterprise services Oracle 8i includes OTT, which are “OTT (executable name OTT.EXE) is used to create C-struct representations of Abstract Data Types that have been created and stored in an Oracle database. To take advantage of objects, run OTT against the database, and a header file is generated that includes the C-structs.”²

319. Thus, Solaris Zones discloses or renders this limitation obvious.

1[a]: “storing in memory accessible to at least some of the servers a plurality of secure containers of application

² https://docs.oracle.com/cd/A69464_01/nt_815/server.815/a68694/Output/ch2.htm

software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;”

320. I understand the parties agreed that “secure container / secure container of application software” means “[a] container environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other applications sets,” wherein “each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.” Dkt. 124, Appx A. I understand the parties dispute the construction for “system files” where IBM proposes “files provided with an operating system and which are available to applications as shared libraries and configuration files” and VirtaMove proposes plain and ordinary meaning. *Id.*, Appx. C. In my opinion, Solaris Zones discloses or renders obvious “storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers” under either parties’ construction.

321. **First**, Solaris Zones teaches “storing in memory accessible to at least some of the servers a plurality of secure containers of application software.” For example, Solaris Zones teaches that Zones must be configured and installed on a given system. *See, e.g.*, ORACLE_VM_PA_00005455 (Tucker Provisional) at -5475 (“A freshly installed Solaris system will not have any (non-global) zones. The first step toward bringing up a zone is to create a configuration which specifies various parameters for that zone. Once a configuration is complete, it can be installed onto the system, then it can be booted, logged into, and

administered.”). An illustration of the configuration and installation state model is shown below:

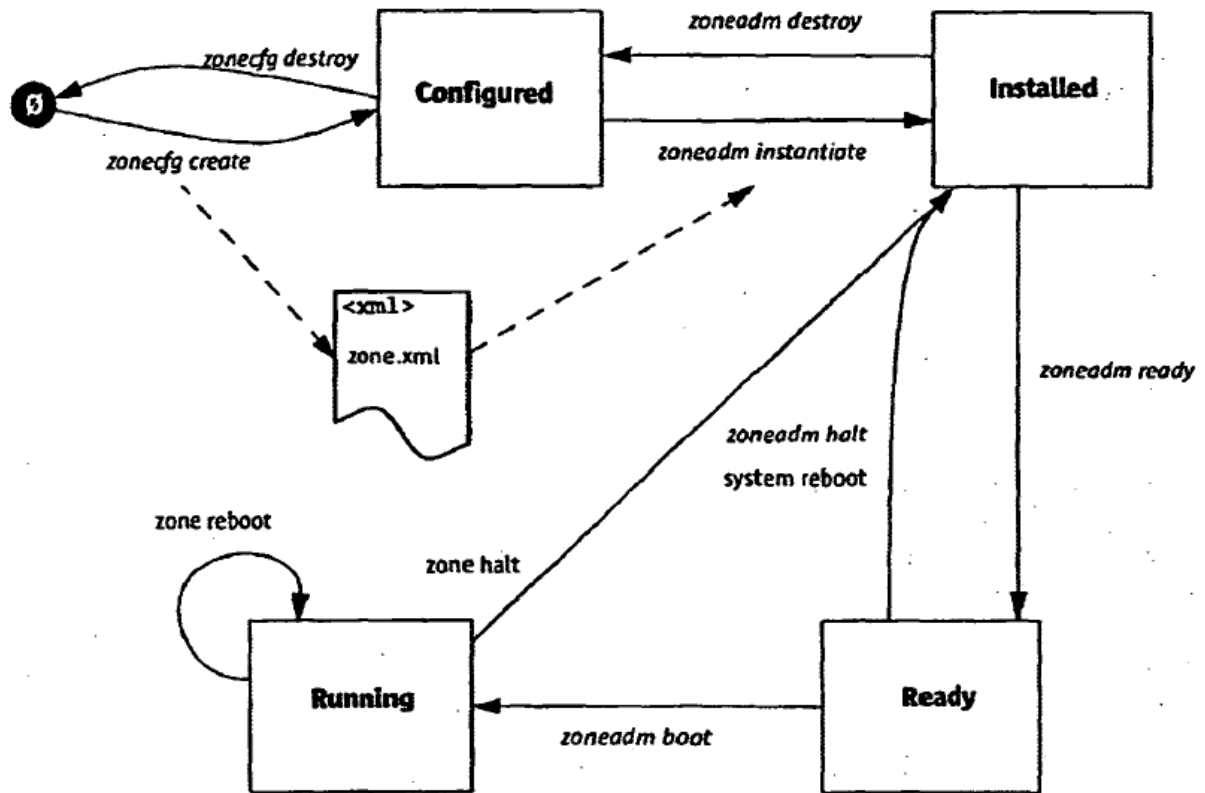


Figure 3.1: Zone State Model

Id. at -5476. Additionally, Solaris Zones teaches that “[t]here also needs to be a repository to pull down packages so that the zone’s file system can be populated” such as a “distribution media like a DVD or a net-install image.” *Id.* at -5540; *see also id.* at -5537-5538 (teaching Solaris packaging system as way of installing Zones).

322. Thus, a POSITA would understand that Solaris Zones renders obvious storing in memory accessible to the computers running Zones “a plurality of secure containers of application software” in which through configuration and installation, each non-global zone is a secure container of application software. These non-global zones are secure containers as

“Zones provide a means of virtualizing operating system services, allowing one or more processes to run in isolation from other activity on the system” and “[t]his isolation prevents processes running within a given zone from monitoring or affecting processes running in other zones.” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467. Solaris Zones further teaches that “[a] zone is a ‘sandbox’ within which *one or more applications can run without affecting or interacting* with the rest of the system.” *Id.* (emphasis added)

323. *Second*, Solaris Zones teaches “each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications.” As previously discussed, Solaris Zones teaches that each non-global zone comprises executable applications related to a service. *See supra* 1[Pre.b]-[Pre.c]. Solaris Zones also explains that each zone includes a set of associated system files required to execute the applications. For example, Solaris Zones teaches the restriction of files and file systems within a given Zone (secure container):

File Systems

Virtualization of storage in a zone is achieved via a restricted root, similar to the chroot(2) environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root. Unlike chroot, a zone will not be escapable; once a process enters a zone, it and all of its children will be restricted to that zone and associated root.

The loopback file system (lofs) provides a useful tool for constructing a file system namespace for a zone. This can be used to mount segments of a file system in multiple places within the namespace; for example, /usr could also be mounted underneath a zone root. The use of lofs in installing a zone is further described in Chapter 12.

ORACLE_VM_PA_00005455 (Tucker Provisional) at -5503. These system files also include configuration files. *See id.* (7.1 Configuration); *see also id.* at -5468 (“Finally, each zone is provided a portion of the file system hierarchy. Because each zone is confined to its subtree of the file system hierarchy, the workloads cannot access each other's on-disk data.”). As

illustrated below, “[e]ach zone may have various file systems which should be mounted when the zone transitions from the INSTALLED state to the READY state”:

3.2.1 Configuration Data

At present, `zonecfg(1M)` uses a project-private XML file to store its configuration data (see Appendix B for the Document Type Definition); later, a Greenline repository or other more general directory-based back-end is envisioned. These data consist of various *resources* some of which have *properties*. The resource types are below; see the man page for the resource type syntax and the details on property types.

zone name Each zone has a unique name; see the next section for details on the allowed syntax for zone names.

zone id Each zone has a unique non-negative integer by which the kernel keeps track of it. Zero is for the global zone; all other numbers less than 100 are reserved.

root path Each zone has a path to its root directory relative to the global zone’s root directory. At installation time, this directory will be required to have the same ownership and permissions as the root directory in the global zone: owned by root, mode 755. Unprivileged users in the global zone will be prevented from traversing a non-global zone’s file system hierarchy; see Section 7.4 for further discussion of such issues.

file systems Each zone may have various file systems which should be mounted when the zone transitions from the INSTALLED state to the READY state.

network interfaces Each zone may have network interfaces which should be plumbed when the zone transitions from the INSTALLED state to the READY state.

devices Each zone may have devices which should be configured when the zone transitions from the INSTALLED state to the READY state.

Id. at -5477 (emphasis added).

324. **Third**, Solaris Zones teaches “for use with a local kernel residing permanently on one of the servers.” Solaris Zones teaches that the container (non-global zones) are for use with the local kernel of the underlying OS. *See, e.g.,* ORACLE_VM_PA_00005455 (Tucker Provisional) at -5540 (“the user-level software in each zone ***must be compatible with the kernel software installed in the global zone.***”) (emphasis added). Solaris Zones also teaches that “the [underlying OS] kernel [] create[s] the zone.” *Id.* at -5476; *see also id.* at -5470 (“The kernel exports a number of distinct objects that can be associated with a particular-zone. These

include processes, file system mounts, network interfaces, and System V IPC objects.”), -5477 (“Each zone has a unique non-negative integer by which the kernel keeps track of it.”).

325. My review of the Solaris source code further confirms that the kernel has functions, including: `zone_create()`, `zone_destroy()`, `zone_getattr()`, `zone_enter()`, `zone_list()`, `zone_shutdown()`, among others. See `on10_52/on10_52/usr/src/uts/common/os/zone.c` and `on10_52/on10_52/usr/src/uts/common/sys/zone.h`.

326. Thus, Solaris Zones discloses that the containers of application software (non-global zones) are for use, and executed by, a local kernel residing permanently on one of the servers.

1[b]: “wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,”

327. I understand the parties dispute the construction for “system files” where IBM proposes “files provided with an operating system and which are available to applications as shared libraries and configuration files” and VirtaMove proposes plain and ordinary meaning. Dkt. 124, Appx. C. In my opinion, Solaris Zones discloses or renders obvious “wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems” under either parties’ construction.

328. As previously discussed, Solaris Zones teaches that the non-global zones (containers) are created and managed by the underlying local kernel of the OS. See *supra* 1[a]. Thus, a POSITA would further understand that the set of associated system files installed with each zone are compatible with the underlying local kernel of the OS.

1[c]: “the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,”

329. I understand the parties agreed that “within a container” means “associated with a container.” Dkt. 124, Appx. A. In my opinion, Solaris Zones teaches “the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.”

330. **First**, Solaris Zones teaches “the containers of application software excluding a kernel.” As previously discussed, “the user-level software in each zone *must be compatible with the kernel software installed in the global zone.*” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5540 (emphasis added). Solaris Zones also teaches that “the [underlying OS] kernel [] create[s] the zone.” *Id.* at -5476; *see also id.* at -5470 (“The kernel exports a number of distinct objects that can be associated with a particular-zone. These include processes, file system mounts, network interfaces, and System V IPC objects.”), -5477 (“Each zone has a unique non-negative integer by which the kernel keeps track of it.”), -5488 (“Kernel messages are often associated with physical hardware and other aspects of a system that are not relevant within a zone; hence, most kernel messages will be sent to the global zone.”), -5530 (“Projects running in different zones with the same project id are considered distinct by the kernel....”). Thus, a POSITA would understand that each non-global zone (secure containers of application software) exclude a kernel, as the kernel creates and configures all of the zones on a given computer.

331. **Second**, Solaris Zones teaches “wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.” As previously discussed, Solaris Zones also explains that each zone includes a set of associated system files required to execute the applications, which

a POSITA would understand are used in place of the associated local system files that remain resident on the server. For example, Solaris Zones teaches the restriction of files and file systems within a given Zone (secure container):

File Systems

Virtualization of storage in a zone is achieved via a restricted root, similar to the `chroot(2)` environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root. Unlike `chroot`, a zone will not be escapable; once a process enters a zone, it and all of its children will be restricted to that zone and associated root.

The loopback file system (`lofs`) provides a useful tool for constructing a file system namespace for a zone. This can be used to mount segments of a file system in multiple places within the namespace; for example, `/usr` could also be mounted underneath a zone root. The use of `lofs` in installing a zone is further described in Chapter 12.

ORACLE_VM_PA_00005455 (Tucker Provisional) at -5503. These system files also include configuration files. *See id.* (7.1 Configuration); *see also id.* at -5468 (“Finally, each zone is provided a portion of the file system hierarchy. Because each zone is confined to its subtree of the file system hierarchy, the workloads cannot access each other's on-disk data.”). As illustrated below, “[e]ach zone may have various file systems which should be mounted when the zone transitions from the INSTALLED state to the READY state”:

3.2.1 Configuration Data

At present, `zonecfg(1M)` uses a project-private XML file to store its configuration data (see Appendix B for the Document Type Definition); later, a Greenline repository or other more general directory-based back-end is envisioned. These data consist of various *resources* some of which have *properties*. The resource types are below; see the man page for the resource type syntax and the details on property types.

zone name Each zone has a unique name; see the next section for details on the allowed syntax for zone names.

zone id Each zone has a unique non-negative integer by which the kernel keeps track of it. Zero is for the global zone; all other numbers less than 100 are reserved.

root path Each zone has a path to its root directory relative to the global zone's root directory. At installation time, this directory will be required to have the same ownership and permissions as the root directory in the global zone: owned by root, mode 755. Unprivileged users in the global zone will be prevented from traversing a non-global zone's file system hierarchy; see Section 7.4 for further discussion of such issues.

file systems Each zone may have various file systems which should be mounted when the zone transitions from the `INSTALLED` state to the `READY` state.

network interfaces Each zone may have network interfaces which should be plumbed when the zone transitions from the `INSTALLED` state to the `READY` state.

devices Each zone may have devices which should be configured when the zone transitions from the `INSTALLED` state to the `READY` state.

Id. at -5477 (emphasis added).

332. Additionally, because Solaris Zones teaches that “different versions of the same application may be run without negative consequences in different zones,” a POSITA would understand that each container (non-global zone) utilizes the associated system files within a container stored in memory” to execute a given application. *Id.* at -5468.

1[d]: “wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,”

333. I understand the parties dispute the construction for “system files” where IBM proposes “files within an operating system and which are available to applications as shared libraries and configuration files” and VirtaMove proposes plain and ordinary meaning. Dkt. 124, Appx.

C. In my opinion, Solaris Zones teaches “wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,” under either parties’ construction.

334. For example, Solaris Zones teaches that “different versions of the same application may be run without negative consequences in different zones...” ORACLE_VM_PA_00005455 at -5468. Additionally, Solaris Zones discloses certain modified system files, including autofs in which “[e]ach zone runs its own copy of automounted”:

7.3 File System-Specific Issues

There are certain security restrictions on mounting certain file systems from within a zone, while others exhibit special behavior when mounted in a zone. The list of modified file systems, the issues surrounding them, and how we plan to deal with these issues are summarized below.

autofs Each zone runs its own copy of automountd, with the auto maps and timeouts under the zone administrator’s control. Since the zone’s file system namespace is really only a subset of that of the global zone, the global zone could possibly create auto maps that reference the non-global zone, or traverse into non-global zones and attempt to trigger mounts. This would cause a number of complications, which we circumvent by saying that triggering a mount in another zone (i.e., crossing an autofs mount point for a non-global zone from the global zone) is disallowed, and that the lookup request fails. Note that such situations cannot arise without the participation of the global zone.

Id. at -5504. Thus, in my opinion, a POSITA would have found it obvious that certain configuration files or libraries for the different versions of the same application would be copies or modified copies of the underlying system files.

1[e]: “and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system’s root file system.”

335. I understand the parties agreed that “secure container / secure container of application software” means “[a] container environment where each application set appears to have

individual control of some critical system resources and/or where data within each application set is insulated from effects of other applications sets,” wherein “each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.” In my opinion, Solaris Zones teaches “and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.”

336. **First**, Solaris Zones teaches “and wherein the application software cannot be shared between the plurality of secure containers of application software.” As previously discussed, Solaris Zones teaches that “Zones provide a means of virtualizing operating system services, *allowing one or more processes to run in isolation from other activity on the system*” in which “one or more applications can run” by “provid[ing] an abstraction layer that separates applications from physical attributes of the machine on which they are deployed...” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467; *see also id.* (“Zones allow the deployment of multiple applications on the same machine, even where those applications operate in different trust domains...”)(“For example, multiple applications running in different zones (but on the same system)...”). Indeed, as illustrated in Figure 1.1 below, “Zones 1, 2 and 3 are each running a disjoint workload” which “demonstrates that different *versions of the same application may be run without negative consequence in different zones*”:

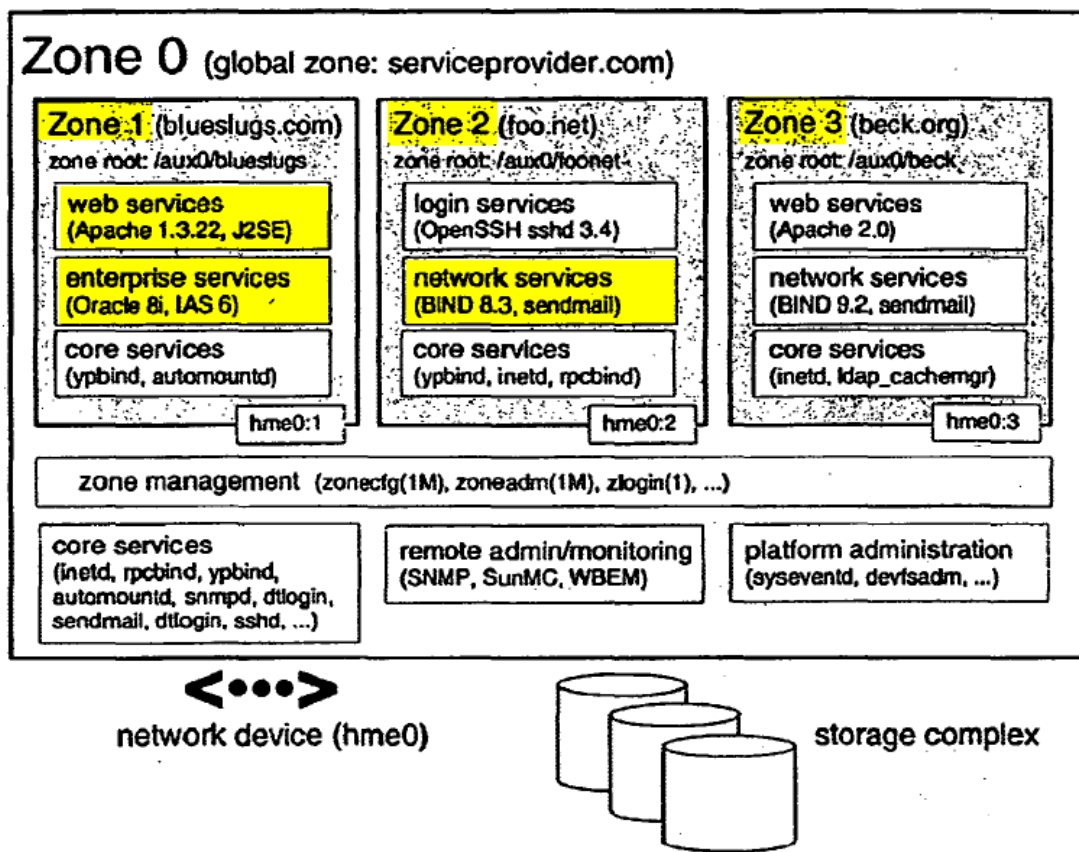


Figure 1.1: Server Consolidation Using Zones

Id. at -5468–5469; *see also id.* at -5503 (“Virtualization of storage in a zone is achieved via a restricted root, similar to the chroot (2) environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root.”).

337. Indeed, as described below, processes within non-global zones (containers) cannot affect other containers:

Process Model

As described in Chapter 5, processes within one zone (other than the global zone) must not be able to affect the activity of processes running within another zone. This also extends to visibility; processes within one (non-global) zone should not even be able to see processes outside that zone.¹ This can be enforced by restricting the process id space exposed through /proc accesses and process-specific system calls (kill(2), priocntl(2), etc.). If the calling process is running within a non-global zone, it will only be able to see or affect processes running within the same zone; applying the operations to any other process ids will return an error.

Id. at -5499.

338. Thus, a POSITA would understand that each non-global zone (container) is secure and isolated in which the application software cannot be shared between the plurality of secure containers of application software.

339. *Second*, Solaris Zones teaches “and wherein each of the containers has a unique root file system that is different from an operating system's root file system.”

340. For example, Solaris Zones teaches that each zone is installed under its root path:

3.3.2 Installing Zones

Next, the administrator verifies that the zone can be installed safely, and installs the zone, which results in the zone being installed under its root path on the global zone's file system:

```
# zoneadm -z nebbiolo-z1 verify
# zoneadm -z nebbiolo-z1 install
# zoneadm -u info
```

ZID	ZONENAME	STATE	ROOT
0	global	running	/
100	nebbiolo-z1	installed	/aux0/my-zone

ORACLE_VM_PA_00005455 (Tucker Provisional) at -5479; *see also id.* at -5537 (“zones require a separate root file systems for isolation.”). Solaris Zones further teaches that each zone is achieved via a restricted root:

Virtualization of storage in a zone is achieved via a restricted root, similar to the chroot (2) environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root. Unlike chroot, a zone will not be escapable; once a process enters a zone, it and all of its children will be restricted to that zone and associated root.

Id. at -5503.

341. My review of the Solaris source code further confirms each zone has its own filesystem with a root directory. [REDACTED]

[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]

[REDACTED]

[REDACTED]

342. Thus, Solaris Zones discloses that each container has a unique root file system that is different from an operating system's root file system.

b. Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”

343. In my opinion, Solaris Zones discloses and/or renders obvious “wherein each container has an execution file associated therewith for starting the one or more applications.” For example, Solaris Zones teaches that once zones are configured and installed, the processes (i.e., applications) can be booted:

3.3.3 Booting Zones (external view)

Next, the administrator boots the zone; this results in kernel data structures being set up, file systems being mounted, network interfaces being plumbed, devices being configured and processes being started:

```
# zoneadm -z nebbiolo-z1 boot
# zoneadm -v info
```

ZID	ZONENAME	STATE	ROOT
0	global	running	/
100	nebbiolo-z1	running	/aux0/my-zone

Refer to `zoneadm(1M)` for further details.

ORACLE_VM_PA_00005455 (Tucker Provisional) at -5479; *see also id.* at -5475 (“Once a configuration is complete, it can be installed onto the system, then it can be booted, logged into, and administered.”). Solaris Zones also teaches that “[a]pplications that are installed into a zone and deliver start/stop scripts into `/etc/init.d` and one or more of the `/etc/rc?.d` directories will be started when the zone is booted.” *Id.* at -5543.

344. Thus, a POSITA would have found it obvious that in creating and installing the non-global zones (containers) to execute applications, each container would also have an execution file associated therewith for starting the one or more applications.

- c. **Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”**

345. In my opinion, Solaris Zones discloses and/or renders obvious “comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”

346. For example, Solaris Zones expressly teaches that each Zone has distinct IP addresses. *See* ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467 (“multiple applications running in different zones (but on the same system) can bind to the same network port using

the *distinct IP addresses associated with each zone.*”), -5510 (“Each zone which requires network connectivity will have one or more dedicated IP addresses.”), -5513 (“Since each zone has one or more unique IP addresses...”). Solaris Zones also teaches certain identification information for each zone by combining MAC address with a zoneid. *See id.* at -5512. Each Zone (container) has its own name and ID:

3.2.2 Zone Names and IDs

Each zone, including the global zone, is assigned a zone name. The rules for a valid zone name are similar to those for host names: they must start with an alpha-numeric character, and all remaining characters must consist of alpha-numerics plus “.” and “-”. The name “global” is reserved, as is anything beginning with “SUNW”.

Each zone name is mapped to a unique numeric zone identifier (or zoneid); this id uniquely identifies the zone on a given system. IDs 0-99 are reserved for use by Sun.

The global zone always has the name “global” and is always mapped to id 0. Zone ids and zone names are under the control of the global administrator, and are generally of little interest within non-global zones. To distinguish between zones, however, the zonename(1) command can be useful.

It is worth noting that each zone also has a node name (i.e. returned by uname -n in the zone). The node name for each zone is under the control of the zone administrator, and generally of little interest to other zones. The node name is completely independent of the zone name.

Id. at -5478.

zone id Each zone has a unique non-negative integer by which the kernel keeps track of it. Zero is for the global zone; all other numbers less than 100 are reserved.

See id. at -5477.

347. My review of the Solaris source code further confirms each zone can have their own IP addresses. [REDACTED]

348. Thus, a POSITA would understand that each non-global zone (container) is assigned a unique identity, which includes an IP address.

- d. **Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”**

349. Solaris Zones discloses and/or renders obvious “wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”

350. For example, Solaris Zones teaches that Zones have “resource controls” in which “[e]ach zone may have resource controls which should be enabled when the zone transitions from the INSTALLED state to the READY state.” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5478.

351. Additionally, Solaris Zones teaches certain resource management features:

10.1 Solaris Resource Management Interactions

The resource management features in Solaris and their interactions with zones are summarized below:

10.1.1 Accounting

10.1.1.1 System V Accounting

The traditional accounting system uses a fixed record size, and can not be extended to differentiate between a process running in the global zone and a non-global zone. We plan to modify the system such that accounting records generated in any (including the global) zone only contain records pertinent to the zone.

10.1.1.2 Extended Accounting

The extended accounting subsystem is virtualized to permit different accounting settings and files on a per-zone basis for process- and task-based accounting. Since exact records are extensible, they can be tagged with a zone id (EXD_PROC_ZONEID and EXD_TASK_ZONEID, for processes and tasks, respectively), allowing the global administrator to determine resource consumption on a per-zone basis. Accounting records are written to the global zone's accounting files as well as their per-zone counterparts. The EXD_TASK_HOSTNAME, EXD_PROC_HOSTNAME, and EXD_HOSTNAME records contain the uname -n value for the zone in which the process/task executed¹, rather than the global zone's node name. Flow accounting is covered in Section 8.4.

Id. at -5530. Specifically, as further described below, Solaris Zones teaches certain “zone-wide limits applicable resources, limiting the total resource usage of all process entities within a zone” in which a “global administrator will be able to specify these limits in the zonecfg configuration file”:

10.1.2 Projects, Resource Controls and the Fair Share Scheduler

Projects are abstracted such that different zones may use separate project(4) databases, each with their own set of defined projects and resource controls. Projects running in different zones with the same project id are considered distinct by the kernel, eliminating the possibility of cross-zone interference, and allowing projects running in different zones (with the same project id) to have different resource controls.

In order to prevent processes in a zone from monopolizing the system, we will introduce zone-wide limits applicable resources, limiting the total resource usage of all process entities within a zone (regardless of project). The global administrator will be able to specify these limits in the zonecfg configuration file². Privileged zone-wide rctl's can only be set by superuser in the global zone.

Id.

352. My review of the Solaris source code further confirms that Solaris Zones allows for certain resource pools, including what portion of the available CPU time a given set of processes can use. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

353. Thus, a POSITA would have found it obvious that server information related to hardware resource usage can include either CPU memory, network bandwidth, and/or disk allocation for each of the containers (non-global zones) to manage resource controls before a given application is started.

- e. **Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.”**

354. I understand the parties dispute the construction for “system files” where IBM proposes “files within an operating system and which are available to applications as shared libraries and configuration files” and VirtaMove proposes plain and ordinary meaning. In my opinion, Solaris Zones discloses and/or renders obvious “wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof,” under either parties’ construction.

355. As previously discussed, Solaris Zones teaches that “Zones provide a means of virtualizing operating system services, *allowing one or more processes to run in isolation from other activity on the system*” in which “one or more applications can run” by “provid[ing] an abstraction layer that separates applications from physical attributes of the machine on which they are deployed...” ORACLE_VM_PA_00005455 (Tucker Provisional) at -5467; *see also id.* (“Zones allow the deployment of multiple applications on the same machine, even where those applications operate in different trust domains...” (“For example, multiple applications running in different zones (but on the same system)....”). Indeed, as illustrated in Figure 1.1 below, “Zones 1, 2 and 3 are each running a disjoint workload” which “demonstrates that different *versions of the same application may be run without negative consequence in different zones*”:

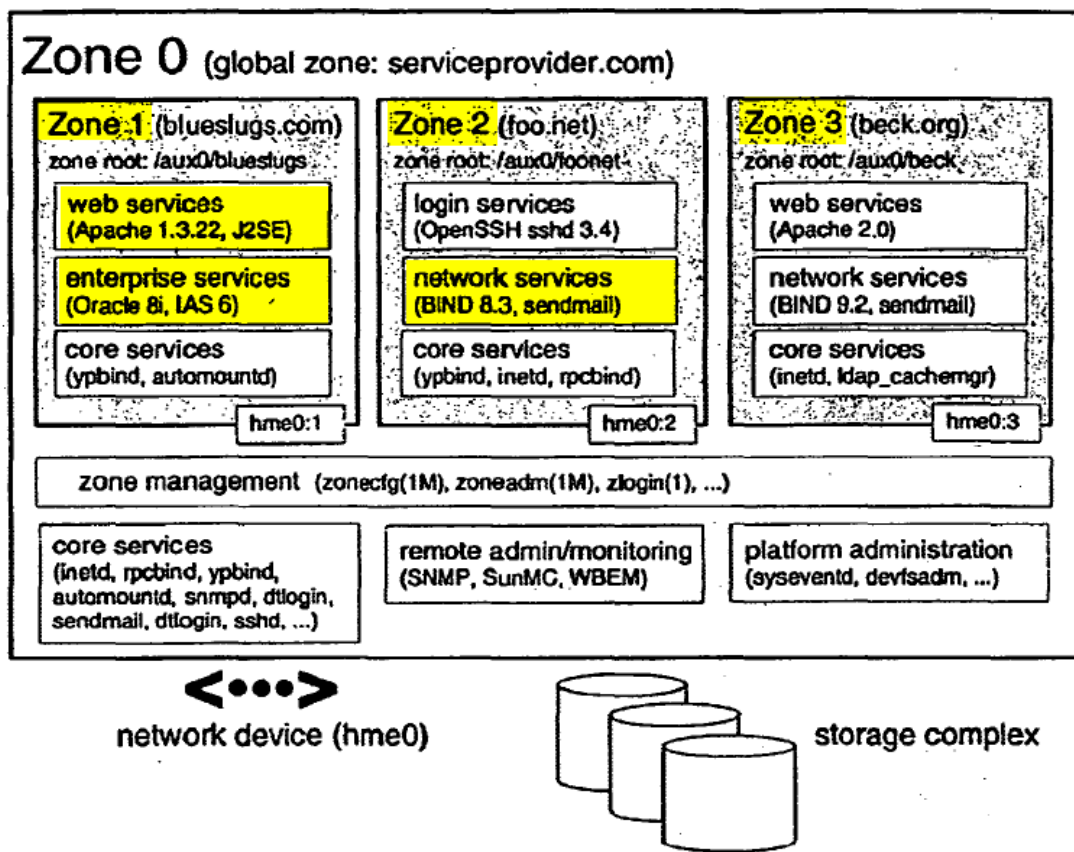


Figure 1.1: Server Consolidation Using Zones

Id. at -5468–5469; *see also id.* at -5503 (“Virtualization of storage in a zone is achieved via a restricted root, similar to the chroot (2) environment at the file system level. Processes running within a zone will be limited to files and file systems that can be accessed from the restricted root.”).

356. Indeed, as described below, processes within non-global zones (containers) cannot affect other containers:

Process Model

As described in Chapter 5, processes within one zone (other than the global zone) must not be able to affect the activity of processes running within another zone. This also extends to visibility; processes within one (non-global) zone should not even be able to see processes outside that zone.¹ This can be enforced by restricting the process id space exposed through /proc accesses and process-specific system calls (kill(2), priocntl(2), etc.). If the calling process is running within a non-global zone, it will only be able to see or affect processes running within the same zone; applying the operations to any other process ids will return an error.

Id. at -5499. Thus, Solaris Zones discloses that wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

ii. Solaris Zones in view of McMillan

357. I incorporate by reference the background of Solaris Zones in Section IX.D. and McMillan in Section IX.N. into my analysis below. I also incorporate by reference my analysis of Solaris Zones and/or Tucker as described in Section X.A.i. above.

a. Claim 1

1[Pre.a]: “In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor,”

358. To the extent Solaris Zones and/or Tucker alone does not disclose or render obvious 1[Pre.a], Solaris Zones in view of McMillan does. For example, McMillan teaches virtual OS environments (containers like Solaris’s Zones) that can be stored in a central location to be used by multiple computers. *See* McMillan, 4:57-65 (“Since all of the virtual OS environment

information is isolated inside of the base OS file system and registry, it becomes easy to save, store, and load entire virtual OS environments. It is therefore convenient to create a virtual OS environment that contains a set of applications configured in a specific way (or a “clean” virtual OS environment that contains no applications yet) *and store it in a separate or central location to be used by multiple computers or the same computer at different times.*”). In my opinion, a POSITA would have been motivated and found it obvious to combine McMillan’s teachings of storing virtual OS environments (containers) with Solaris’s teachings such that each container (non-global zone) of Solaris could be stored in a separate or central location to be used by *multiple computers*. A POSITA would have been motivated to increase the versatility of Solaris zones with the teachings of McMillan. Additionally, to the extent not expressly disclosed, a POSITA would have been motivated and found it obvious to use McMillan’s teachings of a separate and central location to store the containers (non-global zones) that could be used by multiple computers (servers) that have different operating systems. For example, at least under VirtaMove’s interpretation, a POSITA would have found it obvious that the containers (non-global zones) could be stored and accessible to multiple computers running different versions of Solaris operating system; or alternatively different computers running different types of operating systems (MacOS and Solaris).

359. Therefore, in my opinion, Solaris Zones in view of McMillan renders 1[Pre.a] obvious.

1[Pre.b]: “a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment”

1[Pre.c]: “wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:”

1[a]: “storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;”

360. I incorporate by reference my discussion of Solaris Zones. *See supra* X.A.i.a.1[a].

361. Additionally, as I discuss in 1[Pre.a], McMillan teaches virtual OS environments (containers like Solaris’s Zones) that can be stored in a central location to be used by multiple computers. *See* McMillan, 4:57-65. As I previously discussed, a POSITA would have found it obvious to combine McMillan’s teachings of storing the containers (non-global zones) in a separate or central location, such as in memory that would be accessible to the multiple computers (servers). *See supra* 1[Pre.a].

1[b]: “wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,”

1[c]: “the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,”

1[d]: “wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,”

362. I incorporate by reference my discussion of Solaris Zones. *See supra* X.A.i.a.1[d].

363. Additionally, to the extent Solaris Zone alone does not disclose or render obvious 1[d], Solaris Zones in view of McMillan does. For example, McMillan explicitly teaches that the virtual OS environments (containers) utilize copies of the underlying base OS file system and registry. *See* McMillan, 4:21-32 (“Because all of the file system and registry calls are

redirected, a copy of the base OS file system and registry must be created in the virtual OS environment file system and registry before any applications may be executed in the virtual OS environment. Once this is done, any built-in OS applications that are run by applications within the virtual OS environment (such as Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser, all from Microsoft Corporation) will be executed from the virtual OS environment copy of that application. This will allow multiple versions of built-in OS applications to be supported at the same time in different virtual OS environments.”), Figs. 2-4. In my opinion, a POSITA would have found it obvious to combine McMillan’s teachings of utilizing copies of the base OS file system and registry and implement these teachings for Solaris’s non-global zones. This would have been a simple substitution of a known method to achieve a predictable result.

1[e]: “and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system’s root file system.”

- b. Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”**
- c. Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”**
- d. Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”**
- e. Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or**

applications in other containers or to system files within the operating system during execution thereof.”

iii. Thinstall/ThinApp

364. I incorporate by reference the background of Thinstall/ThinApp in Section IX.G. into my analysis below.

a. Claim 1

365. Claim 1 of the '814 Patents states:

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;

wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

366. In my opinion, for the reasons discussed below, Thinstall/ThinApp discloses and/or renders obvious claim 1.

1[Pre.a]: “In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor,”

software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”

f. Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”

996. In my opinion, Claims 2, 3, 4, and 18 are rendered obvious by Deianov in view of Levine for the same reasons previously discussed in my analysis of Deianov alone, except that here the shared library in which the SLCSEs are stored is the shared library as taught by Levine.

iv. FreeBSD 4.0

997. I incorporate by reference the background of FreeBSD 4.0 in Section IX.C. into my analysis below.

a. Claim 1

998. Claim 1 of the '058 Patent recites:

A computing system for executing a plurality of software applications comprising:

a) a processor;

b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

999. In my opinion, for the reasons discussed below, FreeBSD 4.0 discloses and/or renders obvious claim 1.

1[Pre]: “A computing system for executing a plurality of software applications comprising:”

1000. I understand that the parties agree that the preamble to the '058 Patent is limiting. Dkt. 124, Appx. A. In my opinion, FreeBSD 4.0 teaches a computing system for executing a plurality of software applications.

1001. I have reviewed evidence that FreeBSD 4.0 was run on a computing system prior to the '058 Patent. For example, in a review of FreeBSD 4.0, Clifford Smith wrote on July 11, 2000 that he set up “a fully functioning FreeBSD setup, painless and virtually identical to the setup I used in the original article – right here.” IBM_VM_000197149. Mr. Smith links to an article which discusses his setup on which he used FreeBSD 4.0. In it, he states that the “fully functioning FreeBSD setup” was run on a “new AMD K6-III/400 with a 20 gig IBM Deskstar and 256 megs o’ ram.” IBM_VM_000197152. I understand that an AMD K6-III/400 with a 20 gigabyte IBM Deskstar and 256 megabytes of RAM is a computing system. Therefore, FreeBSD 4.0 was run on a computing system prior to the '058 Patent.

1002. *First*, the FreeBSD 4.0 Handbook explains that “FreeBSD is a 4.4BSD-Lite2 based operating system for the Intel architecture (x86) and DEC Alpha based systems.” IBM_VM_000013076; IBM_VM_000013082 (“The FreeBSD Ports collection allows you to compile and install a very wide range of applications with minimum amount of effort.”).

1003. Further, the FreeBSD 4.0 Handbook teaches that FreeBSD 4.0 “runs on *computer systems* based on the Intel architecture (x86), and also the DEC Alpha architecture,” and a POSITA would understand the terms “Intel architecture (x86)” and “DEC Alpha architecture” to be referring to CPU (or hardware) architectures. IBM_VM_000013077 (emphasis added); *see also* IBM_VM_000013094 (describing different CPU architectures); IBM_VM_000190432 (“FreeBSD’s main goal is to run on the hardware people are most likely

to have. Today, this means the Intel x86-compatible systems (386, 486, Pentium I through IV, Celeron, and AMD”). Indeed, FreeBSD 4.0 “can be [and was] used as a powerful desktop OS.” See, e.g., IBM_VM_000190432 at -457-59 (FreeBSD “runs quite well on modern computers.”), (“Who Should Use FreeBSD”), (“FreeBSD as Your Desktop”), (“I even wrote this book on my FreeBSD laptop, using an open-source word processor (Emacs) and a business suite called StarOffice. I also use FreeBSD to watch MPEG video ... This is a fairly exhaustive sample of desktop tasks.”).

1004. **Second**, the FreeBSD 4.0 Handbook explains that FreeBSD offers several features which indicate that FreeBSD 4.0 discloses the execution of software applications on computer systems, including “[p]reemptive multitasking with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users,” “memory protection [to] ensure that applications (or users) cannot interfere with each other[,]” “binary compatibility with many programs built for Linux, SCO, SVR4, BSDI and NetBSD[,]” “thousands of ready-to-run applications [which] are available from the FreeBSD ports and packages collection[,]” and “thousands of additional and easy-to-port applications available on the Internet.” IBM_VM_000013077; see also IBM_VM_000013082 (“The FreeBSD Ports collection allows you to compile and install a very wide range of applications with minimum amount of effort.”).

1005. Indeed, as the FreeBSD 4.0 Handbook further explains, “FreeBSD allows many applications to be run locally, if desired, thus relieving the burden on a central server[,]” and “[t]he applications to which FreeBSD can be put are truly limited only by your own imagination.” *Id.*

1006. Therefore, based on the foregoing, it is my opinion that claim limitation 1[pre] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[a]: “a processor;”

1007. In my opinion, FreeBSD 4.0 teaches a processor. I understand that the parties have agreed that “a processor” means “physical computer processor.” Dkt. 124, Appx. A.

1008. I have reviewed evidence that FreeBSD 4.0 was run on a computing system having a physical computer processor prior to the '058 Patent. As I discussed above, Mr. Smith wrote in an article that he ran FreeBSD 4.0 on a computing system prior to the '058 Patent. Mr. Smith wrote that he ran FreeBSD 4.0 on an “AMD K6-III/400.” IBM_VM_000197149; IBM_VM_000197152. This refers to the AMD K6-III 400 MHz physical computer processor manufactured by AMD. This is a processor, and therefore, Mr. Smith ran FreeBSD 4.0 on a physical computer processor prior to the '058 Patent.

1009. Moreover, a POSITA would have understood and found it obvious FreeBSD 4.0 would run on a computing system having a physical computer processor. *See, e.g.*, IBM_VM_000013077 (FreeBSD “runs on computer systems based on the Intel architecture (x86), and also the DEC Alpha architecture.”), (“With FreeBSD, you can easily start out small with an inexpensive 386 class PC and upgrade all the way up to a quad-processor Xeon with RAID storage as your enterprise grows.”); IBM_VM_000013083 (“FreeBSD runs on a wide variety of ISA, VLB, EISA, and PCI bus based PCs, ranging from the 386SX to Pentium class machines[.]”). I understand the parties agreed that “processor” means a “physical computer processor.” Indeed, the FreeBSD 4.0 Handbook explains that FreeBSD 4.0 offers “SMP support for machines with multiple CPUs[.]” IBM_VM_000013077; *see also* IBM_VM_000013091 (“The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”). A POSITA

would understand the term “CPU” to refer to a “central processing unit” or “central processor,” which is the primary processor in a given computer. *See* IBM_VM_000023016 (“CPU is often used interchangeably with *processor* and *microprocessor*.”); *Id.* (defining “CPU”). This is a piece of physical hardware on the computing system, meaning it is a physical computer processor.

1010. Further, the FreeBSD 4.0 Handbook explains that FreeBSD “runs on computer systems based on the Intel architecture (x86), and also the DEC Alpha architecture,” and a POSITA would understand the terms “Intel architecture (x86)” and “DEC Alpha architecture” to be referring to CPU architectures. IBM_VM_000013077; *see also* IBM_VM_000013094 (describing different CPU architectures); IBM_VM_000190432 (“FreeBSD’s main goal is to run on the hardware people are most likely to have. Today, this means the Intel x86-compatible systems (386, 486, Pentium I through IV, Celeron, and AMD)”); (“FreeBSD also supports the Alpha processor.”); (“Your brand of processor is really irrelevant to FreeBSD; FreeBSD won’t care if you’re running an Intel, AMD, IBM, or Cyrix CPU.”).

1011. Therefore, based on the foregoing, it is my opinion that Claim 1[a] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[b]: “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,”

1012. In my opinion, FreeBSD 4.0 teaches “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.” I understand that VirtaMove has argued that a “critical system element” means “[a]ny service or any part of a service, ‘normally’ supplied by an operating system, that is critical to the operation of a software application.” Dkt. 157, Ex. A. I also understand that the

parties have agreed that “kernel mode” means “[t]he context in which the kernel portion of an operating system executes. Application code cannot run in kernel mode.” Dkt. 124, Appx. A. 1013. **First**, as explained above, “FreeBSD is a 4.4BSD-Lite2 based *operating system* for the Intel architecture (x86) and DEC Alpha based systems.” IBM_VM_000013076 (emphasis added); *see also* IBM_VM_000013077 (“**FreeBSD is a state-of-the-art operating system** based on 4.4BSD-Lite2.”) (emphasis added).

1014. **Second**, FreeBSD 4.0 includes an “operating system kernel having OS critical system elements (OSCEs) for running in kernel mode using said processor.” *See* IBM_VM_000190432 at pages 25, 66, 70-96 (“Your system should have fast access to its root filesystem (/), which contains the **kernel** and just enough utilities and program to boot the computer into its most basic running status[.]”) (emphasis added), (“This minimal boot ... loads the **kernel**”) (emphasis added), (“**Chapter 4: Kernel Games**”) (emphasis added); *see also* IBM_VM_000013119 (“[T]he kernel is one process that must always be present in memory.”); IBM_VM_000013093 (“The kernel is then started, which will then probe for devices and initialize them for use.”); IBM_VM_000013115; IBM_VM_000013118; IBM_VM_000013120 (“7.3. Building and Installing a Custom Kernel”); IBM_VM_000013094 (“7.4. The Configuration File”).

1015. Further, the FreeBSD 4.0 Handbook explains that the features of FreeBSD 4.0 include “[p]reemptive multitasking with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users. ... [and] [m]ultiuser facilities which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system ... and that individual resource limits can be placed on users or groups

of users, protecting critical system resources from over-use.” IBM_VM_000013077; *see also* IBM_VM_000013091 (“The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”). In my opinion, a POSITA would further understand and find obvious that FreeBSD 4.0 has an operating system kernel because the kernel is the part of the operating system that controls interactions between a computer’s hardware and the software applications executing on the computer. *See, e.g.,* IBM_VM_000190432 at 517 (“The kernel is the interface between the hardware and the software.”).

1016. Further, the FreeBSD 4.0 operating system kernel includes “OS critical system elements (OSCEs) for running kernel mode.” Indeed, “[t]he kernel provides interfaces to programs that *need* access to the hardware.” *Id.* (emphasis added). And whereas files stored in the /kernel and /modules directories are part of the kernel, “every file you see outside of /kernel and /modules is not part of the kernel; these files and programs as a group are called the ‘user-land.’” *Id.* Files and data that are part of the kernel provide facilities and functionalities (*i.e.,* services) that are “critical to the operation of software applications.” *See, e.g., id.* at 517, 525-526 (“[T]hese programs and data use the kernel facilities.”), (“If all possible functions were compiled into the kernel, the kernel would be much larger than it is.”), (“When you customize your kernel ... you can also include new functionality in it, or support for new hardware.”). Further, in my opinion, a POSITA would have understood and found it obvious that files that are part of /kernel and /modules are part of “kernel mode” and files that are outside of /kernel and /modules are part of “user mode.” *See, e.g., id.* Further, in my opinion, a POSITA would have understood and found it obvious that the FreeBSD 4.0 operating system kernel includes OSCEs for running in kernel mode because FreeBSD 4.0 discloses the use of

“system calls,” which a POSITA would understand indicates the presence of kernel functions that an application requires to operate. *See, e.g.,* IBM_VM_000190432 at 704 (discussing “system calls”).

1017. Further, as explained above, FreeBSD 4.0 includes a processor, which a POSITA would further understand to be required for any operating system to run on any given computing system. *See supra* Claim 1[a]; *see also* IBM_VM_000013077; IBM_VM_000013094 (describing different CPU architectures); IBM_VM_000190432 (“FreeBSD’s main goal is to run on the hardware people are most likely to have. Today, this means the Intel x86-compatible systems (386, 486, Pentium I through IV, Celeron, and AMD). FreeBSD also supports the Alpha processor.”), (“Your brand of processor is really irrelevant to FreeBSD; FreeBSD won’t care if you’re running an Intel, AMD, IBM, or Cyrix CPU.”) IBM_VM_000190432 at 463.

1018. Therefore, based on the foregoing, it is my opinion that Claim 1[b] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[c]: “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and”

1019. In my opinion, FreeBSD 4.0 teaches “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.” I understand that the parties have agreed that “user mode” means “the context in which applications execute.” Dkt. 124, Appx. A.

1020. **First**, FreeBSD 4.0 includes “a shared library having shared library critical system elements (SLCSEs) stored therein.” The ’058 Patent defines a shared library “an application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an

address space that is accessible to multiple applications,” wherein an “application library” is “a collection of functions in an archive format that is combined with an application to export system elements.” I understand VirtaMove argues “shared library” means “an application library whose code space is shared among all user mode applications.” FreeBSD’s shared library meets both of these constructions. Absolute BSD explains that “The basic idea behind a shared library is quite straightforward: It’s a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for *all programs* to use, and they are designed to be reused by *as many different programs as possible*.” See IBM_VM_000190432 (emphasis added). Through this, it is clear that FreeBSD 4.0’s shared libraries are application libraries whose code space is shared among all user mode applications, as user mode is the context in which applications execute. Moreover, the code space is necessarily different from that occupied by the kernel and its associated files, as applications cannot execute in kernel mode. Because FreeBSD 4.0’s shared libraries are shared among all user mode applications, the files therein are in an address space accessible to multiple applications. Because shared libraries in FreeBSD 4.0 provide services and functions to software applications, a POSITA would have understood and found it obvious that these shared libraries to have “shared library critical elements (SLCSEs) stored therein.”

1021. Further, in VirtaMove’s Infringement Contentions, VirtaMove identifies the GNU C Library, commonly known as glibc, as an example of a shared library having shared library critical systems (SLCSEs) stored therein, as claimed by the ’058 Patent. See 06/05/2024 Infringement Contentions at PDF pgs. 72, 85. Applying VirtaMove’s own application of the claim language for purposes of infringement, FreeBSD 4.0 discloses this claim limitation, as it includes such shared libraries, such as libc, which is a C standard library like glibc, with the

latter being used with Linux OS distributions. *See, e.g.*, IBM_VM_000190432 at pages 250-264; IBM_VM_000017667 (discussing different C library implementations, including “BSD libc” and “GNU C Library (glibc).”); IBM_VM_000013056 at -064 (indicating that “a complete FreeBSD installation” includes “copies of all relevant system binaries, data files, and its own /etc directory.”)

1022. As depicted below, libc is available to software applications running on FreeBSD 4.0. *See, e.g.*, IBM_VM_000190432. A POSITA would have understood and found it obvious that libc is a shared library, since the file name “.so” refers to “shared objects.”

```
# ldconfig -r
/var/run/ld-elf.so.hints:
  search directories:
/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/lib
0:-lcom_err.2 => /usr/lib/libcom_err.so.2
1:-lscrypt.2 => /usr/lib/libscrypt.so.2
2:-lcrypt.2 => /usr/lib/libcrypt.so.2
...
```

```
# ldd /usr/local/bin/emacs
/usr/local/bin/emacs:
  libXaw.so.6 => /usr/X11R6/lib/libXaw.so.6 (0x28159000)
  libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x2818e000)
  libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x2819f000)
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x281e2000)
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x281ea000)
  libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x281fe000)
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x28209000)
  libutil.so.3 => /usr/lib/libutil.so.3 (0x282a2000)
  libm.so.2 => /usr/lib/libm.so.2 (0x282ab000)
  libc.so.4 => /usr/lib/libc.so.4 (0x282c6000)
  libXThrStub.so.6 => /usr/X11R6/lib/libXThrStub.so.6 (0x28361000)
#
```

1023. **Second**, in FreeBSD 4.0, the “shared library having shared library critical system elements (SLCSEs) stored therein [is] for use by the plurality of software applications in user mode.” For example, FreeBSD 4.0 can store shared libraries in a /usr directory (*i.e.*, “user mode”) which a POSITA would understand to be distinct from the /kernel and /modules directory (*i.e.*, “kernel mode”). *See, e.g.*, IBM_VM_000190432 at 517 (“The actual kernel is

a file on disk: /kernel. Kernel modules—the kernel code that can be loaded and unloaded after boot—lives in /modules. ... *Every file you see outside of /kernel and /modules is not part of the kernel; these files and programs as a group are called the user-land, meaning they're meant for users.*") (emphasis added).

1024. FreeBSD 4.0 has a shared library (e.g., libc) with SLCSEs stored therein for use by “the plurality of software applications in user mode.” See, e.g., IBM_VM_000190432 575 (“Once the system is shut down to single-user mode, *you can install the userland programs*[.]”). These software applications can search the /usr directory, for example, in order to locate shared libraries for particular services and function for use in user mode. See, e.g., *id.* at 697-714.

```
.....
# ldconfig -r
/var/run/ld-elf.so.hints:
  search directories:
/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/lib
0:-lcom_err.2 => /usr/lib/libcom_err.so.2
1:-lscrypt.2 => /usr/lib/libscrypt.so.2
2:-lcrypt.2 => /usr/lib/libcrypt.so.2
...
.....
```

```
.....
# ldd /usr/local/bin/emacs
/usr/local/bin/emacs:
  libXaw.so.6 => /usr/X11R6/lib/libXaw.so.6 (0x28159000)
  libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x2818e000)
  libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x2819f000)
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x281e2000)
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x281ea000)
  libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x281fe000)
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x28209000)
  libutil.so.3 => /usr/lib/libutil.so.3 (0x282a2000)
  libm.so.2 => /usr/lib/libm.so.2 (0x282ab000)
  libc.so.4 => /usr/lib/libc.so.4 (0x282c6000)
  libXThrStub.so.6 => /usr/X11R6/lib/libXThrStub.so.6 (0x28361000)
#
.....
```

1025. Therefore, based on the foregoing, it is my opinion that Claim 1[c] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[c][i]: “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are

accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,”

1026. In my opinion, FreeBSD 4.0 teaches “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.”

1027. *First*, in VirtaMove’s Infringement Contentions, VirtaMove identifies the GNU C Library, commonly known as glibc, as an example of a shared library having shared library critical systems (SLCSEs) stored therein. *See* 06/05/2024 Infringement Contentions at PDF pgs. 72, 85. VirtaMove alleges that critical system elements stored in glibc, a system library, are “functional replicas” of critical system elements stored in the operating system kernel. *See id.* at 85. Applying VirtaMove’s own application of the claim language, FreeBSD 4.0 discloses this claim limitation, as it includes shared libraries, such as libc, which is a C standard library such as glibc, with the latter being used with Linux OS distributions. *See, e.g.,* IBM_VM_000190432 at pages 250-64; IBM_VM_000017667 (discussing different C library implementations, including “BSD libc” and “GNU C Library (glibc).”).

1028. As depicted below, libc is available to software applications running on FreeBSD 4.0. *See, e.g.,* IBM_VM_000190432. That is, for example, software applications can access shared libraries stored in any given location of the /usr directory. A POSITA would have understood and found it obvious that libc is a shared library, since the file name “so” refers to “shared objects.”

```

# ldconfig -r
/var/run/ld-elf.so.hints:
    search directories:
/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/lib
0:-lcom_err.2 => /usr/lib/libcom_err.so.2
1:-lscrypt.2 => /usr/lib/libscrypt.so.2
2:-lcrypt.2 => /usr/lib/libcrypt.so.2
...

```

```

# ldd /usr/local/bin/emacs
/usr/local/bin/emacs:
    libXaw.so.6 => /usr/X11R6/lib/libXaw.so.6 (0x28159000)
    libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x2818e000)
    libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x2819f000)
    libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x281e2000)
    libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x281ea000)
    libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x281fe000)
    libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x28209000)
    libutil.so.3 => /usr/lib/libutil.so.3 (0x282a2000)
    libm.so.2 => /usr/lib/libm.so.2 (0x282ab000)
    libc.so.4 => /usr/lib/libc.so.4 (0x282c6000)
    libXThrStub.so.6 => /usr/X11R6/lib/libXThrStub.so.6 (0x28361000)
#

```

1029. Therefore, FreeBSD teaches “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications.”

1030. Further, I understand that VirtaMove has argued in claim construction that “functional replica” means “substantial functional equivalents or replacements of kernel functions.” FreeBSD 4.0 allows for the install—and addition—of shared libraries. *See, e.g., IBM_VM_000190432* at 701-702, 709. A POSITA would have understood and found it obvious to load shared libraries comprising kernel functions (*i.e.*, OSCSEs) such that those shared libraries store SLCSEs that are “functional replicas”—or “substantial functional equivalents” or “replacements”—of OSCSEs. Therefore, under VirtaMove’s proposed claim construction, FreeBSD teaches “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications.”

1031. **Second**, I understand that VirtaMove has argued in claim construction that the claim term “forms a part of the one or more plurality of software applications” should be afforded its plain and ordinary meaning. Without the benefit of having VirtaMove position regarding the plain and ordinary meaning of this term, I refer to the ’058 Patent explaining that “CSEs stored in the shared library ... form a part of at least some of the software applications by being linked thereto,” meaning that SLCSEs form a part of applications by being linked thereto. *See* ’058 Patent, 2:13-15.

1032. FreeBSD 4.0 uses shared libraries, such as *libc*, which software applications dynamically link to. *See, e.g.*, IBM_VM_000190432 at 702. For example, FreeBSD 4.0 supports two binary types: *aout* and *ELF*. *See, e.g., id.* When an ELF binary executes and requires shared libraries, “the system calls *rtld(1)*, the ‘run-time **linker**.’ *Rtld* examines binaries as they’re loaded, determines which shared libraries they need, and loads those libraries.” *See, e.g., id.* Further, a POSITA would also understand and find obvious that *libc* is a “dynamic library” such that software applications dynamically link to *libc*. *See, e.g.*, IBM_VM_000014587 (“Dynamic libraries are generally named with major and minor versions, like *libc.so.1.1*.”).

1033. Therefore, FreeBSD teaches “when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.”

1034. Further, I understand that IBM has argued in claim construction that the claim term “forms a part of the one or more of the plurality of software applications” means “literally form a part of the application such that it resides in the same address space as application code, in contrast to a proxy that is exclusive of the application.” As explained above, *libc* is a “dynamic

library” such that software applications dynamically link to libc. A POSITA would understand when software applications dynamically link to a shared library (*e.g.*, libc in FreeBSD 4.0), the shared library code is memory mapped into each application’s virtual address space such that the shared library (and the SLCSEs stored therein) “literally form[s] a part of the application such that it resides in the same address space as application code.” *See, e.g.*, IBM_VM_000015301; *also above* re Levin’s “Linkers and Loaders” disclosures.

1035. Therefore, based on the foregoing, it is my opinion that Claim 1[c][i] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[c][ii]: “wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and”

1036. In my opinion, FreeBSD 4.0 teaches “wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.” I understand that the parties have agreed that “instance” to mean “a single, specific occurrence or copy.” Dkt. 124, Appx. A.

1037. *First*, FreeBSD 4.0 allows for a “[copy] of a SLCSE provided to at least a first of the plurality of software application from the shared library [to be] run in a context of said at least

first of the plurality of software applications without being shared with other of the plurality of software applications.” For example, FreeBSD 4.0 offers a feature called “jails,” which “imprisons a process and all future de[s]cendants.” IBM_VM_000017754. As further discussed below, a POSITA would have understood that each “jail” environment is the “context” of the application.

1038. Each “jailed environment” has its own root file system or directory separate from the host file system or directory. *See, e.g., id*; IBM_VM_000013056 at -059-60 (“FreeBSD environment (processes, file system, network resources) ... When a jail is created, it is bound to a particular file system root. ... each jail is provided with its exclusive file system root.”); IBM_VM_000017712; IBM_VM_000190432. Each “jail” environment (and its contents) are run in user mode. *See, e.g., IBM_VM_000013056* (“The jail environments must also be custom-configured. This consists of building and installing a miniature version of the FreeBSD file system tree off of a subdirectory in the host environment, usually */usr/jail* ... with a subdirectory per jail.”) (emphasis added).

1039. While building your first jail, you can run the following command to build a complete copy of FreeBSD and install it in the */jail1* directory. *See, e.g., IBM_VM_000190432*. This includes system libraries, including shared libraries like *libc*. *See supra* Claim Limitation 1[c]; *see also IBM_VM_000013056* at -064 (“While the *jail(2)* call could be used in a number of ways, the expected configuration creates a complete FreeBSD installation for each jail. ***This includes copies of all relevant system binaries, data files, and its own /etc. directory.***”) (emphasis added).

1040. For every subsequent jail, rather than to build the FreeBSD again, the copy of FreeBSD in the first jails directory can be installed again, which “will copy a complete set of FreeBSD

userland programs into the [subsequent] jail[s].” *See, e.g., id.* In other words, a copy of FreeBSD userland programs can be installed in the /jail1 directory, which includes system libraries, including shared libraries like libc. *See supra* Claim Limitation 1[c]. For the second and third “jails,” the libc (and the SLCSEs stored therein) that was previously built and stored in the /usr/obj directory is copied into /jail2 and /jail3 directories such that a “copy” of an SLCSE is provided to at least the plurality of software applications running in /jail2 from the shared library in the /jail1 directory.

The `make buildworld` command first uses the source code to build the tools necessary to build the system compiler, after which it builds the compiler and the associated libraries. Finally, it uses the new tools, compiler, and libraries to build all the software included in a core FreeBSD install. (It does not install these components, however, but puts them under `/usr/obj` for later use.)

Client Setup

Setting up a jail is straightforward, though you will need a FreeBSD source tree (see Chapter 6). For example, say you want to build a jail on the partition /jail1. (Jails can be in directories as well, but putting them on separate partitions gives you a quick-and-dirty method of controlling their size. Other admins will just keep an eye on their users, and raise their rates for disk hogs.) To begin, go to your FreeBSD source tree (generally under /usr/src). For your first jail, run this command:

```
# make world DESTDIR=/jail1
```

This command will build a complete copy of FreeBSD and install it in the directory /jail1.

For all subsequent jails, you don't have to build all the binaries; you can install the ones you built the first time by just running this command:

```
# make installworld DESTDIR=/jail1
```

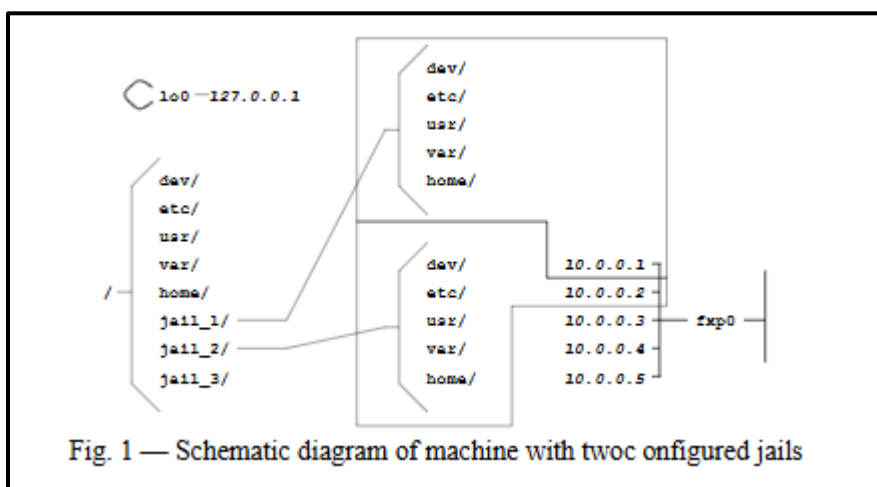
This will copy a complete set of FreeBSD userland programs into the jail.

See, e.g., IBM_VM_000190432.

1041. Because a FreeBSD jail may be based on a first template jail, including its resources including libc, the SLCSEs in each jail are instances of the same shared library which the

original jail has an instance of. *See id.* This first jail is based on the system's resources. Therefore, each libc in a jail is an instance of the same shared library from that first template jail stored on the computing system. As the examiner explained in the final rejection of Claim 1 of the '058 Patent in the *ex parte* reexamination of the patent, "the CSEs in the cited prior art do not stop being part of the shared library just because a particular component instance is isolated in a jail. The jail components are still elements of the shared library of which they are copies." IBM_VM_000197154 at 18. Therefore, the instances of a SLCSE provided to the applications in each jail from the jail's libc instance are instances of "the shared library" from Limitation 1[c] as stored in a template, as the libc components in each jail are "still elements of the shared library of which they are copies." *See id.*

1042. Alternatively, a POSITA would have understood and found it obvious that shared libraries, such as libc, residing in the /usr/lib directory of the host file system can be copied into the file systems for each "jail." *See, e.g.,* IBM_VM_000013056 at -064 ("This includes copies of all relevant system binaries, data files, and its own /etc directory.").



See, e.g., IBM_VM_000013056; IBM_VM_000017754.

1043. Further, through the use of FreeBSD 4.0's "jails," a "[copy] of a SLCSE" provided to at least a first of the plurality of software applications from the shared library *is run in a context of said at least first of the plurality of software applications without being shared* with other of the plurality of software applications." (emphasis added). For example, "[a]ll processes that are running in the jail are restricted to the jail environment, and the kernel does not give them access to any information not in their jail. The filesystem in the jail does not know about files or filesystems outside the jail. ... no program or process in the jail knows about anything outside the jail, and cannot read or access anything outside the jail." IBM_VM_000190432; *see also* IBM_VM_000013056 at 057 ("Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition."), at -061 ("Processes within the jail will find that they are unable to interact or even verify the existence of processes outside the jail – processes within the jail are prevented from delivering signals to processes outside the jail, as well as connecting to those processes with debuggers, or even see them in the sysctl or process file system monitoring mechanisms."). Further, due to the abovementioned process isolation conferred by FreeBSD 4.0's "jails," the plurality of software applications running in a "jail" has its own "unique [copy] of a corresponding critical system element for performing same function" relative to the other plurality of software applications running in a different "jail." *See, e.g., id;* IBM_VM_000017712; IBM_VM_000013056; IBM_VM_000190432.

1044. **Second**, to the extent that the term "instance" refers to a "occurrence," FreeBSD 4.0 allows for a "[occurrence] of a SLCSE" provided to at least a first of the plurality of software application from the shared library [to be] run in a context of said at least first of the plurality

of software applications without being shared with other of the plurality of software applications.”

1045. As explained above, libc is a “dynamic library” such that software applications dynamically link to libc. A POSITA would understand when software applications dynamically link to a shared library (*e.g.*, libc in FreeBSD 4.0), the shared library code is memory mapped into each application’s virtual address space. *See supra* Claim Limitation 1[c][i]; *see also* IBM_VM_000015301 at 49-52, 203, 239-40..

1046. A POSITA would further understand that each application’s virtual address space is the “context” for the application.

1047. Thus, while the software applications share the same physical copy of the shared library, the shared library code is mapped into each application’s address space such that an application is provided with its own “[occurrence] of a SLCSE” that is not shared with other applications. *See, e.g.*, IBM_VM_000015301 at 49-52, 203, 239-40.

1048. Further, because each application receives its own “[occurrence] of an SLCSE” (*i.e.*, memory-mapped page), each application has its own unique “[occurrence] of an SLCSE.” In other words, an SLCSE is memory mapped into one application’s virtual address space, and that same SLCSE—that is, the SLCSE corresponding to the memory mapped page (*i.e.*, “occurrence”)—is memory mapped into a different application’s virtual address space.

1049. Therefore, in my opinion, FreeBSD 4.0 teaches “wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the

plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.”

1050. Therefore, based on the foregoing, it is my opinion that Claim 1[c][ii] is disclosed and/or rendered obvious by FreeBSD 4.0.

1[c][iii]: “wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.”

1051. In my opinion, FreeBSD 4.0 teaches “wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.”

1052. *First*, shared libraries, such as libc, provide predetermined services and functions to software applications. *See supra* Claim Limitation 1[c]; IBM_VM_000017667 (discussing different C library implementations, including “BSD libc” and “GNU C Library (glibc).” Further, as discussed above, a SLCSE is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE. *See supra* Claim Limitation 1[c][ii]. For example, software applications memory map their own unique instances (*i.e.*, a first and second “occurrence”) of any given SLCSE in a shared library (*e.g.*, libc) into their virtual address spaces respectively, such that an SLCSE with a predetermined function can be provided to both a first software application and a second software application for running an instance of that SLCSE related to

a predetermined function. Further, the first software application and second software application, by virtue of having their own respective memory mapped pages, are able to run a first and second instance of any given SLCSE simultaneously despite sharing the same shared library file (*e.g.*, *libc*). *See, e.g.*, IBM_VM_000015301 at 49-52, 203, 239-40..

1053. As another example, as discussed above, each FreeBSD 4.0 “jail” can have its own unique instances (*i.e.*, a first and second “copy”) of a shared library (*e.g.*, *libc*)—and thus “unique instances” of SLCSEs store therein—such that an SLCSE with a predetermined function can be provided to the software applications in two different “jails” for running an instance of that SLCSE related to a predetermined function. *See supra* Claim Limitation 1[c][ii]. Further, a POSITA would have understood and found it obvious that the software applications running in two different “jails,” by virtue of having their own respective copy of the shared library file (*e.g.*, *libc*), are able to run a first and second instance of any given SLCSE simultaneously. Indeed, each FreeBSD 4.0’s “jail” has its own respective file systems, which run independently of one another, having no access nor visibility to files and processes in other “jails.” *See, e.g.*, IBM_VM_000013056 at 057 (“Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition.”).

1054. Therefore, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

1055. Therefore, based on the foregoing, it is my opinion that Claim 1[c][iii] is disclosed and/or rendered obvious by FreeBSD 4.0.

b. Claim 2: “A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.”

1056. In my opinion, FreeBSD 4.0 discloses and/or renders obvious Claim 2.

1057. As discussed above, FreeBSD 4.0 discloses and/or renders obvious Claim 1.

1058. As discussed previously, multiple instances of an SLCSE stored in the system library shared library are available simultaneously to the plurality of applications running on the computing system. *See supra* Claim Limitation 1[c][iii].

1059. First, FreeBSD 4.0 discloses that a first software application and second software application, by virtue of having their own respective memory mapped pages, are able to run a first and second instance of any given SLCSE simultaneously despite sharing the same shared library file (*e.g.*, *libc*). *See supra* Claim Limitation 1[c][iii]. Indeed, multiple processes (*i.e.*, applications) are able to run simultaneously on a computing platform using shared libraries on FreeBSD 4.0 because the purpose of shared libraries was to provide “instances” of critical system elements to each application while reducing the risk of the applications conflicting with one another. *See, e.g.*, IBM_VM_000015301 at 360 (“In nearly every system that handles multiple programs simultaneously, each program has a separate set of page tables, giving each program a logically separate address space. ... If a single program or single program library is in use in more than one address space, the system can save a great deal of memory if all of the address spaces share a single physical copy of the program or library. ... When several different programs use a shared library, the linker needs to mark [] each program so that when each starts, the library is mapped into the program’s address space.”).

1060. Second, the software applications running in two different “jails,” by virtue of having their own respective copy of the shared library file (*e.g.*, libc), are able to run a first and second instance of any given SLCSE simultaneously. Multiple “jails,” which each comprise applications and an “instance of an SLCSE” (*i.e.*, different instances of the same system library protocol found in different Virtual Environments), may run simultaneously on the computing system running FreeBSD 4.0. *See, e.g.*, IBM_VM_000190432 at -629 (“Managing Jails”).

1061. Therefore, based on the foregoing, it is my opinion that Claim 2 is disclosed and/or rendered obvious by FreeBSD 4.0.

c. Claim 3: “A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.”

1062. In my opinion, FreeBSD 4.0 discloses and/or renders obvious Claim 3.

1063. As discussed above, FreeBSD 4.0 discloses and/or renders obvious Claim 1.

1064. For example, FreeBSD 4.0 discloses OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel. *See supra* Claim Limitation 1[c][i] (“Functional Replicas”).

1065. As discussed previously, in VirtaMove’s Infringement Contentions, VirtaMove identifies the GNU C Library, commonly known as glibc, as an example of a shared library having shared library critical systems (SLCSEs) stored therein. *See* 06/05/2024 Infringement Contentions at PDF pgs. 72, 85. VirtaMove alleges that critical system elements stored in glibc, a system library, are “functional replicas” of critical system elements stored in the operating system kernel. *Id.* at 85. A POSITA would have understood and found it obvious that FreeBSD 4.0 uses shared libraries, such as libc, which is a C standard library such as glibc, with the latter being used with Linux OS distributions. *See, e.g.*, IBM_VM_000190432 at 250-

64 (covering Chapter 11); IBM_VM_000017667 (discussing different C library implementations, including “BSD libc” and “GNU C Library (glibc).”)

1066. As depicted below, libc is available to software applications running on FreeBSD 4.0. *See, e.g.,* IBM_VM_000190432. That is, for example, software applications can access shared libraries stored in any given location of the /usr directory. A POSITA would have understood and found it obvious that libc is a shared library, since the file name “so” refers to “shared objects.”

```
# ldconfig -r
/var/run/ld-elf.so.hints:
  search directories:
/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/li
0:-lcom_err.2 => /usr/lib/libcom_err.so.2
1:-lscrypt.2 => /usr/lib/libscrypt.so.2
2:-lcrypt.2 => /usr/lib/libcrypt.so.2
...
```

```
# ldd /usr/local/bin/emacs
/usr/local/bin/emacs:
  libXaw.so.6 => /usr/X11R6/lib/libXaw.so.6 (0x28159000)
  libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x2818e000)
  libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x2819f000)
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x281e2000)
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x281ea000)
  libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x281fe000)
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x28209000)
  libutil.so.3 => /usr/lib/libutil.so.3 (0x282a2000)
  libm.so.2 => /usr/lib/libm.so.2 (0x282ab000)
  libc.so.4 => /usr/lib/libc.so.4 (0x282c6000)
  libXThrStub.so.6 => /usr/X11R6/lib/libXThrStub.so.6 (0x28361000)
#
```

1067. A POSITA would have further understood and found it obvious that the OSCSEs corresponding to and capable of performing the same function as SLCSEs (which are “functional replicas” of OSCSEs) remain in the operating system kernel because, rather than to make system calls directly to the kernel for specific services or functions (*i.e.*, OSCSEs), applications make library calls to a shared library such as libc for specific services or functions (*i.e.*, SLCSEs), and the shared library then makes system calls to the operating system kernel

to retrieve and provide the requested services or functions. If the OSCSEs do not remain in the operating system kernel, SLCSE would not be able to provide the requested services or functions to an application.

1068. Therefore, based on the foregoing, it is my opinion that Claim 3 is disclosed and/or rendered obvious by FreeBSD 4.0.

d. Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”

1069. In my opinion, FreeBSD 4.0 discloses and/or renders obvious Claim 4.

1070. As discussed above, FreeBSD 4.0 discloses and/or renders obvious Claim 1.

1071. For example, in FreeBSD 4.0, one or more SLCSEs can be provided to one of the plurality of software applications having exclusive use thereof. *See supra* Claim Limitation 1[c]-[c][iii]. Further, as explained above, libc is a shared library having SLCSEs stored therein. *See supra* Claim Limitation 1[c]. A POSITA would have understood and found it obvious that libc includes SLCSEs that wrap system calls (*i.e.*, system call wrappers), which are used to access services in the operating system kernel (*i.e.*, kernel functions). *See, e.g.*, IBM_VM_000015301.

1072. Therefore, based on the foregoing, it is my opinion that Claim 4 is disclosed and/or rendered obvious by FreeBSD 4.0

e. Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”

1073. In my opinion, FreeBSD 4.0 discloses and/or renders obvious claim 18.

1074. As discussed above, FreeBSD 4.0 discloses and/or renders obvious claim 2.

1075. **First**, as discussed previously, a POSITA would have understood and found it obvious that, rather than to make system calls directly to the kernel for specific services or functions

(i.e., OSCSEs), applications make library calls to a shared library such as libc for specific services or functions (i.e., SLCSEs). The shared library then makes system calls to the operating system kernel to retrieve and provide the requested services or functions. If the OSCSEs do not remain in the operating system kernel, SLCSE would not be able to provide the requested services or functions to an application. A POSITA would have understood and found it obvious that if SLCSEs are “copies” of OSCSEs, then both the SLCSEs and OSCSEs could exist independently of one another, since a POSITA would understand the term “copies” to mean that the SLCSEs and OSCSEs are “identical.”

1076. **Second**, a POSITA would understand the term “copies” to mean that the OSCSEs on any given server are copied into a shared library in user-mode to become SLCSEs thereafter. A POSITA would thus understand and find it obvious that SLCSEs stored in a shared library such as libc are not “copies” of OSCSEs because shared libraries such as libc (and SLCSEs stored therein) are installed with OSCSEs as part of the installation of a FreeBSD 4.0 operating system distribution, which includes shared libraries and kernel services.

1077. Therefore, based on the foregoing, it is my opinion that claim 18 is disclosed and/or rendered obvious by FreeBSD 4.0.

v. FreeBSD 4.0 in view of Alpine

1078. I incorporate by reference the background of FreeBSD 4.0 in Section IX.C. and Alpine in Section IX. B. into my analysis below. I also incorporate by reference my analysis of FreeBSD 4.0 as described in Section X.B.iv. and Alpine as described in Section X.B.i. above.

a. Motivation to Combine

1079. To the extent VirtaMove argues that SLCSEs which are functional replicas of OSCSEs are not disclosed by FreeBSD, a POSITA would have been motivated to include this functionality in FreeBSD as taught by Alpine.

f. Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”

1104. In my opinion, FreeBSD in view of Alpine renders obvious claim 4.

1105. As discussed above, FreeBSD in view of Alpine renders obvious claim 2.

1106. In my opinion, the additional limitation of Claim 2 is rendered obvious by FreeBSD 4.0 in view of Alpine for the same reasons as I discussed in my analysis of Alpine. In use in FreeBSD 4.0's system, the same remains true as they are still in user mode with changes from the kernel networking stack to enable running in user mode. *See id.* Moreover, Alpine is designed for users to make modifications to the networking protocols in the Alpine shared library. *See IBM_VM_000197146.* Once these modifications are made, as is this purpose of Alpine, there will be further changes made to the SLCSEs such that they have further differences from the kernel OSCSEs.

vi. FreeBSD 4.0 in view of Levine

1107. I incorporate by reference the background of FreeBSD 4.0 in Section IX.C. and Levine in Section IX.K. into my analysis below. I also incorporate by reference my analysis of FreeBSD 4.0 as described in Section X.B.iv. above.

a. Motivation to Combine

1108. To the extent VirtaMove argues that multiple instances of library components running simultaneously is not disclosed or rendered obvious by FreeBSD, a POSITA would have been motivated to include this functionality in FreeBSD as taught by Levine.

1109. Absolute BSD discloses that FreeBSD is a UNIX-like operating system available to users freely over the internet. *IBM_VM_000190432* at 452. FreeBSD has strengths including portability, power, simplified software management, an optimized upgrade process, and a sophisticated file system. *Id.* at 457-458. Moreover, FreeBSD utilizes ELF shared libraries,

which Absolute BSD explains are “a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for all programs to use, and they are designed to be reused by as many different programs as possible.” *Id.* at 699. There is not, however, a disclosure of how specifically these ELF shared libraries interact with applications with regards to simultaneous access to ELF shared libraries or isolation of shared library components accessed by FreeBSD’s applications. Given the express goal of FreeBSD to provide its shared library files to as many application as possible, a POSITA would have been motivated to look to references such as Levine which explain how ELF libraries interact with applications, including how simultaneously running applications may utilize their own unique instances of the shared library’s components simultaneously, thus maximizing the number of applications that can utilize the shared library. Should those shared library components not be available simultaneously, more shared libraries would be required on the system, taking up valuable storage. Further, should the instances of the shared library not be unique to applications, conflicts might occur.

1110. As previously discussed in my analysis of Deianov in view of Levine, Levine teaches the use of a dynamic shared libraries for storage of files such as SLCSEs in an address space available to the plurality of applications. Levine discloses dynamically linked shared libraries, called ELF libraries, which have files stored therein for simultaneous use by applications. *See id.* These applications each have use of unique instance of each SLCSE stored in that shared library, and that instance is not shared amongst with other applications. *See id.*

1111. A POSITA would have found it obvious to incorporate Levine’s teachings regarding the implementation of dynamic shared libraries into FreeBSD to implement shared libraries which may be simultaneously utilized by a plurality of applications, which would be of great

benefit to FreeBSD's applications which require shared library resources. Indeed, Absolute BSD explains that in FreeBSD, the purpose of shared libraries is "to be reused by as many different programs as possible." IBM_VM_000190432, at 699. A POSITA would have understood that if shared libraries were not simultaneously available to a plurality of applications at the same time, this would go against FreeBSD's goal of having its shared libraries be reused by as many programs (applications) at once. Moreover, providing isolation of the instances of these shared libraries which are provided to each application would maximize the number of applications that could utilize the shared library resources. Absolute BSD actually discloses that FreeBSD uses ELF libraries as taught by Levine. IBM_VM_000190432 at 700. Given FreeBSD's use of ELF libraries and FreeBSD's goal of providing as many applications as possible with the same shared library resources, as I discussed in my analysis of Deianov in view of Levine, a POSITA would have been motivated to look to a reference like Levine which explains how ELF libraries may be utilized to provide shared library components in ELF libraries to a wide number of applications. Moreover, a POSITA would have been motivated to look to references like Levine for teachings regarding how to isolate the instances of shared library code provided to applications such that they prevent conflicts from occurring between the applications simultaneously accessing the same shared library files. *See id.*

1112. FreeBSD and Levine share the same ELF shared library architecture. *See* IBM_VM_000190432 at 700. Given this synergy between FreeBSD's and Levine's shared libraries, a POSITA would have been motivated to incorporate Levine's teachings of ELF libraries which provide multiple, private simultaneously running shared library components in user mode to improve FreeBSD's ELF shared libraries to be simultaneously available to a

plurality of applications. Rather than requiring modifications to FreeBSD's shared libraries, incorporating the teachings of Levine would merely instruct a POSITA on how to implement FreeBSD's pre-existing ELF libraries. This would amount to applying a known technique to a known system ready for improvement to yield predictable results. For these same reasons, modification of FreeBSD to include multiple simultaneously running shared library components would also be combining prior art elements according to known methods to yield predictable results and use of a known technique to improve similar devices in the same way.

1113. Incorporating Levine's teachings regarding dynamic linked libraries would also have been obvious to try. FreeBSD discloses that its system utilizes ELF dynamic shared libraries. Given Levine's teaching of the benefit of simultaneously providing shared library components to applications, simultaneously available shared library components would have been obvious to try in FreeBSD for the reasons previously discussed. A POSITA would have had a reasonable expectation of success in doing so given that implementation of these teachings involves no changes to the operating system itself, simply teaching how ELF shared libraries may be utilized in practice. Accordingly, no changes to the operating system or hardware would be required either.

b. Claim 1

1114. In my opinion, limitations 1[pre] – 1[c] are rendered obvious by FreeBSD in view of Levine for the same reasons previously discussed in my analysis of FreeBSD alone.

1115. Further, in my opinion, FreeBSD in view of Levine renders obvious “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.”

1116. As I previously discussed, FreeBSD alone discloses or renders obvious this limitation. FreeBSD in view of Levine renders obvious for the same reason that some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications for the same reasons previously discussed in my analysis of FreeBSD alone. However, as I discussed in the Motivation to Combine section above, a POSITA would have been motivated to incorporate Levine's shared library teachings into the FreeBSD system. As I discussed in my analysis of Deianov in view of Levine, Levine teaches that when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications. For the same reasons as discussed there, FreeBSD, including its dynamic shared library, in view of Levine renders obvious this limitation.

1117. Further, in my opinion, limitations 1[cii] – 1[c][iii] are rendered obvious by FreeBSD in view of Levine for the same reasons previously discussed in my analysis of FreeBSD alone and Deianov in view of Levine. In my analysis of Deianov in view of Levine, I explained Levine's teachings of shared library improvements which allow for unique instances of SLCSEs to be provided to applications simultaneously, without sharing said instances with other applications. In implementing these teachings, FreeBSD and Levine's shared library improvements still perform the same functions they do individually. FreeBSD utilizes the shared library teachings of Levine regarding unique and isolated instances of SLCSEs from FreeBSD's shared library being provided to a plurality of applications simultaneously. *See* Motivation to Combine. Therefore, in incorporating Levine's shared library teachings into FreeBSD, FreeBSD in view of Levine renders obvious "an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context

of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function,” and “a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.”

- c. Claim 2: “A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.”**

1118. In my opinion, FreeBSD discloses or renders obvious claim 2.

1119. As discussed above, FreeBSD discloses or renders obvious claim 1.

1120. In my opinion, FreeBSD in view of Levine renders obvious this additional limitation for the same reasons discussed in my analysis of FreeBSD alone. Further, as I discussed in my analysis of Deianov in view of Levine and in the Motivation to Combine Section, Levine teaches multiple instances of an SLCSE stored in a shared library which run simultaneously. Moreover, a POSITA would have understood that the applications running in FreeBSD in view of Levine’s system run within the operating system, as that is the system on which they are implemented. Therefore, FreeBSD in view of Levine renders obvious that in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

- d. **Claim 3: “A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.”**
- e. **Claim 4: “A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.”**
- f. **Claim 18: “A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.”**

1121. In my opinion, Claims 3, 4, and 18 are rendered obvious by FreeBSD in view of Levine for the same reasons previously discussed in my analysis of FreeBSD alone.

vii. Virtuozzo

1122. I incorporate by reference the background of Virtuozzo in Section IX.F. into my analysis below.

a. Claim 1

1123. Claim 1 of the '058 Patent recites:

A computing system for executing a plurality of software applications comprising:

- a) a processor;*
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,*
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and*
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,*
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and*
 - iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.*