

Filed: January 30, 2025

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMAZON.COM, INC.,
Petitioner,

v.

VIRTAMOVE, CORP.,
Patent Owner.

IPR2025-00561
U.S. Patent No. 7,784,058

**PETITION FOR *INTER PARTES* REVIEW
OF CLAIMS 1-16 AND 18 OF U.S. PATENT NO. 7,784,058**

TABLE OF CONTENTS

INTRODUCTION -----	1
I. BACKGROUND-----	2
A. Shared Libraries Were Well Known -----	2
1. Many Common Operating Systems Supported Shared Libraries Dynamically Linked to Applications -----	3
2. Levine Disclosed Shared Libraries and Dynamic Linking -----	5
B. Storing “Critical System Elements” (“CSEs”) in Shared Libraries Was Known-----	6
1. Ely Disclosed Networking in a Shared Library-----	6
2. Thekkath Disclosed Networking in a Shared Library -----	8
3. Eggert Disclosed File Systems in a Shared Library -----	10
II. THE ’058 PATENT-----	10
A. Overview -----	10
B. Prosecution History -----	13
III. STATEMENT OF RELIEF REQUESTED -----	13
A. Grounds -----	13
B. The References Are Prior Art-----	14
1. Ely is Prior Art -----	14
2. Eggert is Prior Art -----	15
3. Levine is Prior Art -----	16
4. Thekkath is Prior Art -----	16

C.	The Asserted References Are Analogous Art-----	16
IV.	LEVEL OF ORDINARY SKILL -----	17
V.	CLAIM CONSTRUCTION -----	17
VI.	GROUND OF UNPATENTABILITY -----	19
A.	Ground 1: Ely and Levine -----	19
1.	Claim 1 -----	21
a.	Preamble-----	22
b.	Processor -----	23
c.	Operating System with Kernel and OSCSEs -----	23
d.	Shared Library with SLCSEs-----	25
i.	SLCSEs are Functional Replicas of OSCSEs -----	26
ii.	SLCSEs Form a Part of the Application-----	27
e.	CSE Provided to First and Second Applications-----	29
i.	First Instance of SLCSE Provided to First Application-----	29
ii.	Unique Instance of Corresponding CSE Provided to Second Application-----	32
f.	Two Instances of SLCSE Run Simultaneously -----	33
i.	SLCSE Related to a Predetermined Function -----	33
ii.	Second Instance of SLCSE-----	33
2.	Claim 2 -----	34
3.	Claim 3 -----	34

4.	Claim 4 -----	35
5.	Claim 5 -----	35
6.	Claim 6 -----	37
	a. Kernel Provides Notification of an Event -----	37
	b. Event is Asynchronous and Outside of the Application -----	37
7.	Claim 9 -----	38
8.	Claim 10 -----	39
9.	Claim 11 -----	40
10.	Claim 12 -----	41
11.	Claim 13 -----	41
12.	Claim 14 -----	42
13.	Claim 16 -----	43
14.	Claim 18 -----	43
B.	Ground 2: Ely, Levine, and Thekkath -----	44
	1. Claim 5 -----	46
	2. Claim 6 -----	48
	a. Kernel Provides Notification of an Event -----	48
	b. Event is Asynchronous and Outside of the Application -----	48
	3. Claim 7 -----	49
	4. Claim 8 -----	49
	5. Claim 15 -----	50

a.	Kernel Module Enables Data Exchange -----	51
b.	Data Exchange Uses Virtual Memory Mapping ----	51
C.	Ground 3: Eggert and Levine -----	52
1.	Claim 1 -----	54
a.	Preamble-----	54
b.	Processor -----	54
c.	Operating System with Kernel and OSCSEs -----	55
d.	Shared Library with SLCSEs-----	55
i.	SLCSEs are Functional Replicas of OSCSEs-----	56
ii.	SLCSEs Form a Part of the Application-----	57
e.	CSE Provided to First and Second Applications ----	58
i.	First Instance of SLCSE Provided to First Application -----	58
ii.	Unique Instance of Corresponding CSE Provided to Second Application -----	59
f.	Two Instances of SLCSE Run Simultaneously -----	59
i.	SLCSE Related to a Predetermined Function -----	59
ii.	Second Instance of SLCSE-----	60
2.	Claim 2 -----	60
3.	Claim 3 -----	61
4.	Claim 4 -----	61
5.	Claim 9 -----	62

6.	Claim 10 -----	63
7.	Claim 11 -----	63
8.	Claim 12 -----	64
9.	Claim 16 -----	65
10.	Claim 18 -----	65
VII.	NO SECONDARY CONSIDERATIONS OF NONOBVIOUSNESS -----	65
VIII.	NO BASIS FOR DISCRETIONARY DENIAL UNDER §314(A) -----	66
IX.	NO BASIS FOR DISCRETIONARY DENIAL UNDER §325(D) -----	67
X.	MANDATORY NOTICES -----	68
A.	Real Parties-In-Interest (37 C.F.R. §42.8(b)(1))-----	68
B.	Related Matters (37 C.F.R. §42.8(b)(2)) -----	69
C.	Lead and Backup Counsel (37 C.F.R. §42.8(b)(3)) -----	69
D.	Service Information (37 C.F.R. §42.8(b)(4)) -----	70
E.	Payment of Fees (37 C.F.R. §42.103) -----	70
F.	Grounds for Standing (37 C.F.R. §42.104(a)) -----	71
	CONCLUSION -----	71
	CERTIFICATE OF COMPLIANCE -----	72

TABLE OF AUTHORITIES

Cases:	Page(s):
<i>Apple v. Fintiv</i> , IPR2020-00019, Paper 11 (P.T.A.B. Mar. 20, 2020)	66
<i>In re GPAC Inc.</i> , 57 F.3d 1573 (Fed. Cir. 1995)	17
<i>Intel Corp. v. Qualcomm Inc.</i> , 21 F.4th 801 (Fed. Cir. 2021)	18, 19
<i>Jazz Pharms., Inc. v. Amneal Pharms., LLC</i> , 895 F.3d 1347 (Fed. Cir. 2018)	15
<i>Juniper Networks, Inc. v. Monarch Networking Solutions LLC</i> , IPR2024-00175, 2024 WL 2054958 (PTAB May 8, 2024)	68
<i>Leapfrog Enters. v. Fisher-Price, Inc.</i> , 485 F.3d 1157 (Fed. Cir. 2007)	65
<i>M & K Holdings, Inc. v. Samsung Elecs. Co.</i> , 985 F.3d 1380-83 (Fed. Cir. 2021)	15
<i>Mueller Systems, LLC v. Rein Tech, Inc.</i> , IPR2020-00098, 2020 WL 2478514 (May 12, 2020)	68
<i>Newell Cos. v. Kenney Mfg. Co.</i> , 864 F.2d 757 (Fed. Cir. 1988)	65
<i>Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co. Ltd.</i> , 868 F.3d 1013 (Fed. Cir. 2017)	17
<i>Samsung Electronics Co., Ltd. v. Maxell, Ltd.</i> , IPR2024-00907, 2024 WL 5079910 (PTAB Dec. 11, 2024)	68
<i>Sotera Wireless, Inc. v. Masimo Corp.</i> , IPR2020-01019, Paper 12 (P.T.A.B. Dec. 1, 2020)	66
<i>Unwired Planet, LLC v. Google Inc.</i> , 841 F.3d 995 (Fed. Cir. 2016)	16

<i>Vivid Techs., Inc. v. Am. Sci. & Eng’g, Inc.</i> , 200 F.3d 795 (Fed. Cir. 1999)	17
--	----

Statutes and Rules:

35 U.S.C. §102	14, 15, 16
35 U.S.C. §103	13
35 U.S.C. §314	66, 67
35 U.S.C. §325	67, 68
37 C.F.R. §42.8	69, 70
37 C.F.R. §42.10	70
37 C.F.R. §42.15	70
37 C.F.R. §42.103	70
37 C.F.R. §42.104	71

Miscellaneous:

Katherine K. Vidal, <i>Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation</i> (June 21, 2022)	66
--	----

TABLE OF EXHIBITS

Exhibit No.	Description
1001	U.S. Patent No. 7,784,058 (“the ’058 patent”)
1002	Declaration of Dr. Darrell Long
1003	David Ely et al., <i>Alpine: A User-Level Infrastructure for Network Protocol Development</i> , Proc. 3rd USENIX Symposium on Internet Technologies and Systems (USITS ’01) (2001) (“Ely”)
1004	Paul R. Eggert & D. Stott Parker, <i>File Systems in User Space</i> , Proc. USENIX Winter 1993 Conference (1993) (“Eggert”)
1005	John R. Levine, <i>Linkers & Loaders</i> (2000) (“Levine”)
1006	Chandramohan A. Thekkath, et al., <i>Implementing Network Protocols at User Level</i> , 23 ACM SIGCOMM Computer Comm. Rev., 64-73 (1993) (“Thekkath”)
1007	David Ely et al., <i>Alpine: A User-Level Infrastructure for Network Protocol Development</i> , 3rd USENIX Symposium on Internet Technologies and Systems (USITS ’01) (“Ely Presentation”)
1008	Excerpts from the ’058 patent’s file history
1009	Arindam Banerji et al., <i>Protected Shared Libraries – a New Approach to Modularity and Sharing</i> , Proc. USENIX Annual Technical Conf. (USENIX 1997) 59-75 (“Banerji”)
1010	Excerpts of UNIX System V Release 3 Programmer’s Guide
1011	Excerpts of SunOS 4.1 Release Manual
1012	Excerpts of Solaris and SunOS System Interfaces Guide

Exhibit No.	Description
1013	Excerpts of Sun Linker and Libraries Guide
1014	Linux Release Notes
1015	FreeBSD 3.3 Linker Information
1016	Excerpts of Windows API Guide Reference Volume 3 (1992)
1017	Scheduling order from <i>VirtaMove, Corp. v. Amazon.com, Inc. et al.</i> , No. 7:24-cv-30-ADA-DTG (W.D. Tex.) (the “Litigation”)
1018	Order cancelling <i>Markman</i> hearing in the Litigation
1019	Order granting transfer in the Litigation
1020	Arnold, J. Q., “Shared Libraries on UNIX System V”, Summer Conference Proceedings, Atlanta 1986, USENIX Association, 1986.
1021	FreeBSD 3.0 Release Notes
1022	FreeBSD 3.3 Release Notes
1023	FreeBSD System Calls Manual – “select(2)”
1024	Curriculum Vitae of Dr. Darrell Long
1025	Declaration of Sylvia D. Hall-Ellis
1026	Wayback Machine – USITS ’01 Home Page
1027	Amazon’s Opening Claim Construction Brief in the Litigation
1028	Wayback Machine – USITS ’01 Alpine Abstract

Amazon.com, Inc. v. VirtaMove, Corp.
IPR Petition – U.S. Patent No. 7,784,058

Exhibit No.	Description
1029	Australian Unix User Group Newsletter, Vol. 8, No. 5 excerpts
1030	Declaration of Rachel J. Watters
1031	<i>Ex Parte</i> Reexamination Request
1032	Order Granting Request for <i>Ex Parte</i> Reexamination

Petitioner Amazon.com, Inc. (“Petitioner” or “Amazon”) requests *inter partes* review of claims 1-16 and 18 of U.S. Patent No. 7,784,058 (“the ’058 patent”), which VirtaMove, Corp. (“Patent Owner” or “PO”) purportedly owns.

INTRODUCTION

The ’058 patent claims computing systems in which functionality provided within a computer’s operating system (“OS”) is replicated in another location on the same computer. The patent’s purported novelty lies in the location where the replicated functionality is stored: a “shared library.” Although such libraries were well known, the patent asserts that shared libraries in the prior art did not replicate functionality that was *also* present in an operating system on the same computer.

Contrary to the patent’s assertion, replicating OS functionality in a shared library was not new. Prior art disclosing this use of shared libraries was published at least a decade before the patent’s priority date. The prior art grounds presented in this Petition disclose replicating two types of OS functionality in shared libraries: (1) network functionality, and (2) file-system functionality. Both of these functionalities are also used as example embodiments in the ’058 patent.

The ’058 patent’s Examiner never considered the prior art presented in this Petition. Because the prior art demonstrates that the ’058 patent contributed nothing new to the public store of knowledge, the patentee has no right to exclude the public from practicing the patent claims.

I. BACKGROUND

A. Shared Libraries Were Well Known.

Shared libraries allow software applications to share common functionality without wasting memory. (Ex. 1002 ¶40.) Instead of each application having its own copy of computer code for common functionality, the applications (1) link to a shared library that contains that functionality and (2) use a single copy of the library's contents whenever possible, as explained below. (Ex. 1020, 395.)

Different parts of a shared library may be shared in different ways. One part of a shared library contains executable code that every linked application can access. (Ex. 1009, 60-61; Ex. 1002 ¶41.) Because this code does not change from one application to another, every application shares the same code. (*Id.*) Sharing this code avoids duplication and saves space in the computer's memory. (Ex. 1020, 395.) For example, if multiple applications need to connect to the Internet, all of the applications can link to a single shared library that contains code for establishing such connections. (Ex. 1002 ¶41.) Similarly, the patent admits that it was "common practice" for all applications that access files on a computer to use shared code for doing so. (Ex. 1001 at 3:32-37 ("All applications currently share code for common services, such services include operations such as ... open a file, write to the file, read from the file, etc.").)

Shared libraries also contain data that is accessed by the shared code. (Ex. 1009, 60-61; Ex. 1010, 8-28; Ex. 1002 ¶42.) Each linked application gets its own copy of the data from the shared library. (*Id.*; Ex. 1013, 100.) Each application's copy of the data is private—i.e., it is unique to that application and can be accessed only by code running in the context of that application. (Ex. 1005, 169-172.) For example, multiple applications may share the same library code for connecting to the Internet, but each application may need to access a different web site. (Ex. 1002 ¶42.) Although all applications share the same library *code*, each application uses its own private copy of the library *data* to keep track of the unique Internet address it is accessing. (*Id.*) Giving each application its own private copy of the data allows the applications to share code without interfering with other applications' differing uses of the shared code. (*Id.*)

1. Many Common Operating Systems Supported Shared Libraries Dynamically Linked to Applications.

Popular operating systems such as UNIX included shared libraries by the 1980s. (Ex. 1002 ¶43; Ex. 1010, 3-24 (“Beginning with Release 3.0, shared libraries are supported.”), Ch. 8 (explaining how to build and use shared libraries).) Shared libraries were quickly adopted by other operating systems in the late 1980s and early 1990s. (Ex. 1002 ¶43; Ex. 1011, 25 (shared libraries supported in SunOS 4.0 and 4.1 by 1990); Ex. 1012, 2; Ex. 1013, 3, 11-12; Ex. 1014; Ex. 1005, 4.) Thus, shared

libraries were widely used well before the '058 patent's earliest claimed priority date in 2003. (Ex. 1002 ¶43; Ex. 1009, 60 (“Most commercial operating systems support shared libraries in one form or another ...”).) The patent admits this. (Ex. 1001, 7:3-5 (“what is commonly done is to provide an application library in shared code space”).) PO also admitted this during prosecution. (Ex. 1008, 21 (“It is fully acknowledged that [prior art from 1995] describe[s] shared libraries”).)

Shared libraries can be linked with applications using “static” linking, performed when an application is first compiled and installed, or “dynamic” linking, which delays the linking process until an application starts running. (Ex. 1005, 188; Ex. 1012, 2-3; Ex. 1010, G-7 (“Dynamic linking refers to the ability to resolve symbolic references at run time.”), G-18 (“Static linking refers to the requirement that symbolic references be resolved before run time.”); Ex. 1013, 4.) Dynamic linking was introduced to UNIX in the late 1980s and was quickly adopted by other popular operating systems such as SunOS, Solaris, and Windows. (Ex. 1012, 109; Ex. 1013, 3; Ex. 1016, 1.) Thus, many operating systems supported dynamic linking of shared libraries well before the '058 patent's earliest claimed priority date. (Ex. 1002 ¶44.)

2. Levine Disclosed Shared Libraries and Dynamic Linking.

A textbook called *Linkers and Loaders*, authored by John R. Levine and published in 2000 (“Levine”), explains that most operating systems included shared libraries and dynamic linking by the time it published. (Ex. 1005, xi, 4, 187-88, 206 (Linux and Unix used “ELF” format for shared libraries), 217-18 (Microsoft Windows used “DLL” format); Ex. 1002 ¶45.) Levine explains how shared libraries were commonly created and linked to applications using either static or dynamic linking. (Ex. 1005, xiii-xiv, Ch. 9 (shared libraries and static linking), Ch. 10 (dynamic linking).)

Levine explains that most applications “usually turn out to share a lot of common code;” therefore, if “all the programs that use a library can share a single copy of it,” significantly less storage space is needed. (Ex. 1005, 4; Ex. 1002 ¶46.) Although different shared library formats may vary slightly, the process of linking them to applications is “nearly the same.” (Ex. 1005, 190; Ex. 1002 ¶46.) When an application program that is linked to shared libraries starts running, “startup code finds those [linked] libraries and maps them into the program’s address space before the program starts.” (Ex. 1005, 188; Ex. 1002 ¶46.) The program’s address space is the part of the computer’s memory that is allotted to that program. (Ex. 1002 ¶46.)

B. Storing “Critical System Elements” (“CSEs”) in Shared Libraries Was Known.

The '058 patent coins the phrase “Critical System Elements” to describe OS functionality replicated in the shared library. (Ex. 1001, 6:6-10.) Two examples of CSEs are “Network services including TCP/IP” and “File System services.” (*Id.*, 6:11-28.) The prior art disclosed replicating both examples in shared libraries.

1. Ely Disclosed Networking in a Shared Library.

In 2001, David Ely and other researchers at the University of Washington published a paper titled “Alpine: A User-Level Infrastructure for Network Protocol Development.” (Ex. 1003 (“Ely”).) They presented their paper at the 3rd USENIX Symposium on Internet Technologies and Systems. (Ex. 1007.) The paper explains that traditional networking stacks were difficult for developers to work with because they were part of a “kernel,” which is the core of an operating system. (Ex. 1003, 171.) Any changes to a traditional networking stack required rebuilding the entire kernel. (*Id.*) To overcome this inconvenience, Ely presented the “Alpine” system, which copies the networking stack from an operating system’s kernel into a shared library that is dynamically linked with applications. (*Id.*, 171, 177; Ex. 1002 ¶48.)

Figures 1 and 2 of Ely show, respectively, a “traditional” stack where “all network processing is performed in the kernel” and Alpine’s library where “unmodified networking code is placed:”

Figure 1

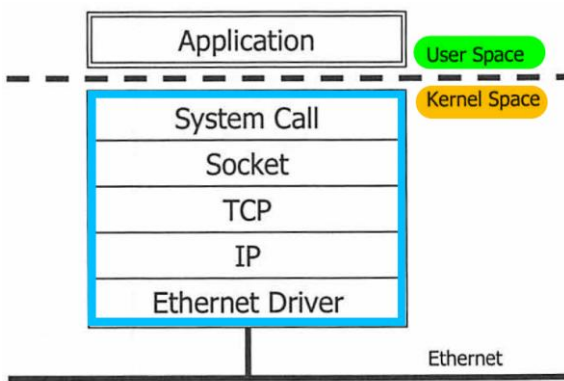
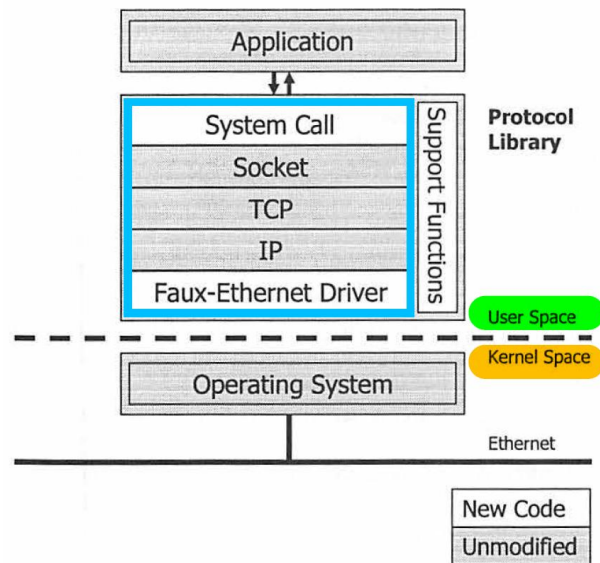


Figure 2



(Ex. 1003, Figs. 1, 2; Ex. 1002 ¶49.)¹ As these figures show, the traditional stack implements **networking** in **kernel space**. (Ex. 1003, Fig. 1.) But Alpine's stack moves the **networking** to a library in **user space**. (*Id.*, Fig. 2.) Kernel space is the part of the computer's memory in which the kernel resides. (Ex. 1002 ¶49.) User space is the part of memory in which applications and shared libraries reside. (*Id.*)

¹ Figures in this Petition may be colored and annotated for ease of reference. Additionally, some figures in this Petition come from online copies of the references because figures in the online copies provide better clarity in some cases than the scanned paper copies. (See Ex. 1025 ¶33, n.18; ¶40, n.23 (describing online sources).) The content of the online copies and paper copies is the same.

Alpine was implemented in a “dynamic library,” which was a well-known type of shared library widely used in popular operating systems. (Ex. 1003, 177; *supra* §I.A.1; Ex. 1002 ¶50.) Alpine’s library used “dynamic linking” to integrate with pre-existing applications. (Ex. 1003, 177.)

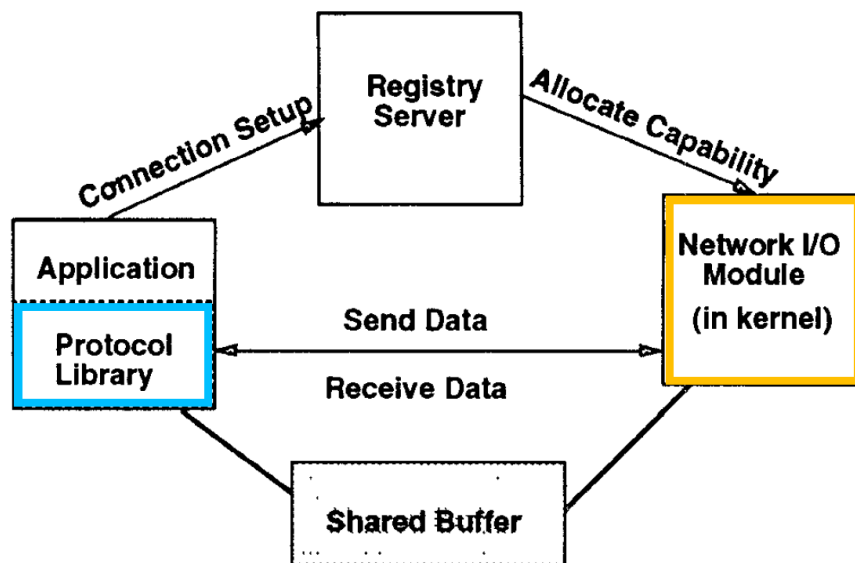
Alpine was “fully implemented in the FreeBSD 3.3 operating system.” (*Id.*, 178.) “However, very little of [Alpine’s] code [was] specific to this version of FreeBSD, and most of it [was] portable to any Unix environment.” (*Id.*) Thus, Ely explained that Alpine broadly “support[ed] a FreeBSD networking stack on top of a Unix operating system.” (*Id.*, 171.)

Presentation slides from the conference where Ely was first presented further confirm that Alpine was implemented as a shared library. The presentation shows that applications linked to Alpine using the command “LD_PRELOAD=libAlpine.so.” (Ex. 1007, 8; *id.* at 3, 6-7; Ex. 1002 ¶52.) The “.so” extension of the libAlpine file shows that it was a shared library. (Ex. 1015, line 326 (“a ‘.so’ extension indicates a shared library.”); Ex. 1002 ¶52.)

2. Thekkath Disclosed Networking in a Shared Library.

In 1993, Chandramohan Thekkath and other researchers published a conference paper titled “Implementing Network Protocols at User Level.” (Ex. 1006 (“Thekkath”).) Thekkath, like Ely, described research conducted at the University of Washington’s Department of Computer Science and Engineering. (*Id.*; Ex. 1003.)

Thekkath implemented networking protocols outside of the kernel to allow for “ease of prototyping, debugging, and maintenance.” (Ex. 1006, 64.) Thekkath’s system allowed for the “co-existence of multiple protocols” that could “exploit application-specific knowledge,” unlike the traditional networking stack, which was generalized for the entire OS. (*Id.*; Ex. 1002 ¶54.) Similar to Ely, Thekkath used a shared library to store its networking protocols outside the kernel. (Ex. 1006, 64, 67, 68; Ex. 1002 ¶54.) This system, with the “protocol library” linked to an application, is shown in Figure 2:



(Ex. 1006, Fig. 2.) Because the “protocol library [] is linked into the application” and “[t]he library contains the code that implements the communication protocol,” applications can directly access several different networking protocols stored in the library. (*Id.*, 67, 68; Ex. 1002 ¶54.)

Because the protocol library is in user space rather than the kernel, it does not have direct access to networking hardware. (Ex. 1002 ¶55.) Rather, the protocol library exchanges data with a “Network I/O Module” in the kernel. (Ex. 1006, 69.) This kernel module acts as an interface between the protocol library and networking devices. (*Id.*; Ex. 1002 ¶55.)

3. Eggert Disclosed File Systems in a Shared Library.

Also in 1993, Paul Eggert and another researcher published “File Systems in User Space” and presented it at the USENIX Winter Conference. (Ex. 1004.) This paper disclosed “two user-extensible file systems,” collectively called “IFS,” that extended a computer’s default file system with added capabilities. (*Id.*, 229.) These file systems were implemented with libraries to which applications could dynamically link. (*Id.*, 229, 235-237; Ex. 1002 ¶56.) When an application was dynamically linked to the IFS shared library, that library intercepted the application’s calls to the operating system, allowing the application to access files in new ways. (Ex. 1004, 231-32; Ex. 1002 ¶56.)

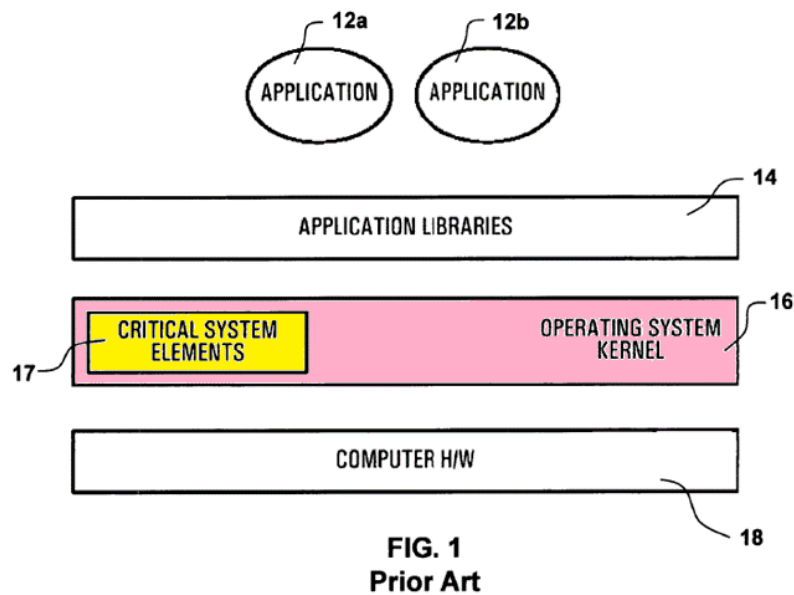
II. THE ’058 PATENT

A. Overview

The ’058 patent discloses shared libraries containing so-called “critical system elements” (CSEs), which replicate functionality from an OS kernel. (Ex. 1001, 1:58-64.) “The term replica used [in the patent] is meant to denote a CSE having similar

attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system[.]” (*Id.*, 1:66-2:3.)

Figure 1 shows a “conventional architecture where critical system elements execute in kernel mode”:



(Ex. 1001, Fig. 1; *id.*, 6:62-63.)

An “important distinction between [the ’058 patent] and [some] prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application.” (*Id.*, 1:46-48.) CSEs “are replicated through the use of shared libraries” and “replicated CSEs are then able to run in the context of a software application” like any other shared library. (*Id.*, 5:22-34.)

“By way of example; a Linux based platform provides a TCP/IP [i.e., networking protocol] stack in the Linux kernel. [The ’058 patent] provides a TCP/IP

stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD [TCP]/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library.” (*Id.*, 5:41-49; *see also* Ex. 1002 ¶60 (providing background on TCP/IP).)

Figure 4 of the patent shows “how critical system elements exist in the same context as an application.” (Ex. 1001, 5:11-12.) In this figure, “the CSE shared library, 47, provides an implementation of a critical system element.” (*Id.*, 8:23-27.)

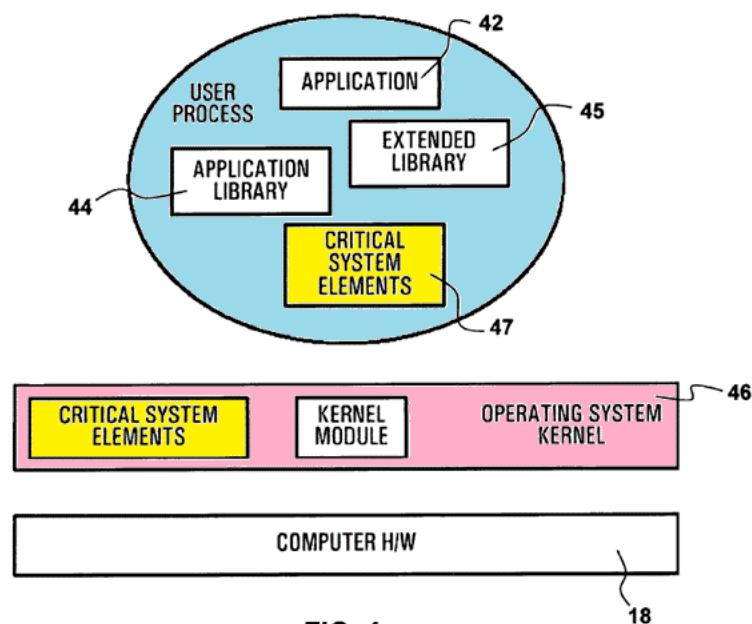


FIG. 4

(*Id.*, Fig. 4.)

B. Prosecution History

The applicant admitted during prosecution that (1) shared libraries were known and (2) running CSEs outside of the kernel was known. (Ex. 1008, 21-22.) But the applicant argued that CSEs *in shared libraries* that *replicate CSEs in the kernel* were novel and non-obvious. (*Id.*) The applicant amended the claims to capture these distinctions. (*Id.*, 15 (2/18/2009 response at 2).) The amended claims required that CSEs in a shared library (“SLCSEs”) are “replicas” of CSEs in the OS kernel (“OSCSEs”). (*Id.*) The Examiner ultimately allowed the patent without considering any of the references raised in this Petition. (See Ex. 1001 at 1 (References Cited).)

III. STATEMENT OF RELIEF REQUESTED

A. Grounds

The Board should cancel the claims under 35 U.S.C. §103 as follows:

Ground	Claims Challenged	References
1	1-6, 9-14, 16, 18	Ely and Levine
2	5-8, 15	Ely, Levine, and Thekkath
3	1-4, 9-12, 16, 18	Eggert and Levine

B. The References Are Prior Art.

The '058 patent's earliest claimed priority date is September 22, 2003. (Ex. 1001.) Petitioner does not concede the claims are entitled to that priority date, and reserves its right to show they are not so entitled. Regardless, the references herein are prior art under pre-AIA 35 U.S.C. §102 for the reasons set forth below.

1. Ely is Prior Art.

Ely (Ex. 1003) was publicly available as part of the published proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01), which was held on March 26-28, 2001. (See Ex. 1025 ¶¶32-33.) Ely was catalogued and indexed at the University of Texas-Austin and University of California-Santa Cruz Libraries no later than May 22, 2001. (See *id.* ¶¶32-38, 52.) Accordingly, Ely was publicly available more than one year before the earliest claimed priority date of the '058 patent and is prior art under at least §102(b).

A slide presentation about Ely (Ex. 1007, "Ely Presentation") was publicly available more than one year before the '058 patent's earliest claimed priority date because it was delivered at the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01). (Ex. 1028.) This conference was held March 26-28, 2001 and was chaired by professors from the University of Washington, Stanford University, Massachusetts Institute of Technology, the University of California - Berkeley, Rice University, and executives from technology companies including Akamai

Technologies and Fast Forward Networks. (Ex. 1026.) Additionally, Ely Presentation was posted on the USENIX website no later than Jan. 4, 2002. (See Ex. 1028 (link to download Ely Presentation on a webpage that was “[l]ast changed: 4 Jan. 2002”).) Thus, Ely Presentation was publicly accessible by live presentation as of March 28, 2001, and by online publication no later than January 4, 2002. See *M & K Holdings, Inc. v. Samsung Elecs. Co.*, 985 F.3d 1380-83, (Fed. Cir. 2021) (finding references publicly accessible when discussed at company meetings and made available on the company’s website); *Jazz Pharms., Inc. v. Amneal Pharms., LLC*, 895 F.3d 1347, 1355-60 (Fed. Cir. 2018) (finding materials to be publicly accessible when presented at an FDA meeting and made available on the FDA’s website). Accordingly, Ely Presentation is prior art under at least §§102(a) and (b).

2. Eggert is Prior Art.

Eggert (Ex. 1004) was publicly available as part of the published proceedings of the Winter 1993 USENIX Conference, which was held on January 25-29, 1993. (See Ex. 1025 ¶¶46.) Eggert was catalogued and indexed at the University of California-Los Angeles Southern Regional Library Facility on October 7, 1993 and at the OCLC Corporate Library in Dublin, Ohio, on February 24, 1993. (See *id.* ¶¶46-51, 52.) Accordingly, Eggert was publicly available more than one year before the earliest claimed priority date of the ’058 patent and is prior art under at least §102(b).

3. Levine is Prior Art.

Levine (Ex. 1005) was catalogued and indexed at the Kurt F. Wendt Library at the University of Wisconsin-Madison on February 29, 2000. (Ex. 1030.) Accordingly, Levine was publicly available more than one year before the earliest claimed priority date of the '058 patent and is prior art under at least §102(b).

4. Thekkath is Prior Art.

Thekkath (Ex. 1006) was publicly available as part of the published proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM '93), which was held on September 13-17, 1993. (See Ex. 1025 ¶¶39.) Thekkath was catalogued and indexed at the University of Wisconsin-Madison and Oklahoma State University Libraries no later than January 7, 1994. (See *id.* ¶¶39-45, 52.) Accordingly, Thekkath was publicly available more than one year before the earliest claimed priority date of the '058 patent and is prior art under at least §102(b).

C. The Asserted References Are Analogous Art.

The references are analogous art because they are from the same field as the '058 patent. (Ex. 1002 ¶27; Ex. 1001, 1:15-17 (patent “relates to a computing system, and to an architecture that affects and extends [the] services exported through application libraries”)); *Unwired Planet, LLC v. Google Inc.*, 841 F.3d 995, 1000 (Fed. Cir. 2016). The references are also pertinent to problems the inventor was

focused on, e.g., extending operating system services using shared libraries. (Ex. 1002 ¶27)

IV. LEVEL OF ORDINARY SKILL

Based on the relevant factors, *In re GPAC Inc.*, 57 F.3d 1573, 1579 (Fed. Cir. 1995), a person of ordinary skill in the art (“POSITA”) would have had a minimum of a bachelor’s degree in computer science, computer engineering, software engineering, or a similar field, and approximately two years of related work experience. (Ex. 1002 ¶37.) Additional work experience could substitute for formal education and additional formal education could substitute for work experience. (*Id.*)

V. CLAIM CONSTRUCTION

On January 26, 2024, PO sued Petitioner for infringement of the ’058 patent in *VirtaMove, Corp. v. Amazon.com, Inc.*, No. 7:24-cv-00030-ADA-DTG (W.D. Tex.) (“the Litigation”). Although the parties proposed constructions for certain terms pertaining to issues in the Litigation, no claim terms require construction to resolve the obviousness challenges here. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co. Ltd.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017); *Vivid Techs., Inc. v. Am. Sci. & Eng’g, Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999).

Petitioner has asserted in the Litigation that two limitations in claim 1 of the ’058 patent—“critical system elements” and “functional replicas”—are indefinite.

(Ex. 1027 at 7, 10.) The indefiniteness of these limitations does not preclude a patentability determination here because the claim limitations clearly cover the prior art, even though the limitations’ outer boundaries are uncertain. *See Intel Corp. v. Qualcomm Inc.*, 21 F.4th 801, 812-13 (Fed. Cir. 2021) (“The indefiniteness of a limitation ... precludes a patentability determination only when the indefiniteness renders it logically impossible for the Board to reach such a decision.”)

For “critical system elements,” the full scope of the limitation is unclear but one of the ’058 patent’s dependent claims recites two specific examples that are expressly within the scope: “services related to ... network protocol processes” and “services related to ... the management of files.” (Ex. 1001, claim 12; Ex. 1002 ¶65.) The prior art relied on here discloses both examples. Specifically, Ely and Thekkath disclose critical system elements related to network protocol processes. (*Supra*, §§ I.B.1, I.B.2.) Eggert discloses critical system elements related to the management of files. (*Supra*, § I.B.3.) Thus, whatever the boundaries of “critical system elements” may be, the prior art is squarely within those boundaries.

Similarly, for “functional replicas,” the full scope of the limitation is unclear but example embodiments in the specification confirm that the prior art here is within the scope. The example embodiments make clear that examples of “functional replicas” include an alternate implementation of the TCP/IP networking protocol and an extension of a computer’s file system. (Ex. 1001, 3:16-30, 5:35-53, 6:11-15,

claim 12.) The prior art here discloses both of these examples. (*Supra*, §§ I.B.1, I.B.2, I.B.3.) Thus, whatever the boundaries of “functional replicas” may be, the prior art is squarely within those boundaries.

Accordingly, no terms require construction to resolve the obviousness challenges in this Petition. *Intel*, 21 F.4th at 812-13.

VI. GROUNDS OF UNPATENTABILITY

A. Ground 1: Ely and Levine

Ely discloses that its Alpine library was implemented in the “FreeBSD 3.3” operating system and could be easily transferred to a “Linux” operating system. (Ex. 1003 at 178, 182.) These operating systems both used the “ELF” format for shared libraries. (Ex. 1005 at 62 (“ELF has been adopted by the popular freeware Linux and by BSD variants of UNIX as well”), 206; Ex. 1002 ¶68.) Specifically, FreeBSD began using ELF in version 3.0 and continued to do so in version 3.3. (Ex. 1021 §1.3; Ex. 1022 §4.) Levine explains how shared libraries were implemented using ELF. (Ex. 1005 at 190.) Thus, a POSITA would understand that the Alpine library disclosed in Ely, which ran on FreeBSD 3.3, was an ELF shared library that worked in accordance with Levine’s explanation of such libraries. (Ex. 1002 ¶68.)

Regardless, even if Ely did not inherently disclose the ELF shared libraries that Levine describes, combining these references would have been obvious. A POSITA would have been motivated to combine the teachings of Ely and Levine to

dynamically link the Alpine shared library in the manner disclosed in Levine for several reasons. (Ex. 1002 ¶69.)

First, Ely contains an express teaching, suggestion, or motivation because it calls for implementing Alpine as a dynamically linked library and Levine discloses how dynamically linked libraries were implemented. (*Supra*, §I.A.1; *supra*, §I.B.1.)

Second, Ely sought to achieve shortened development times and easier debugging of networking protocols by using a user-space library. (Ex. 1003, 171; Ex. 1002 ¶71.) This disclosure would have motivated a POSITA to look to the dynamically linked shared libraries in Levine, which discloses that “[d]ynamically linked shared libraries are easier to update than statically linked shared libraries.” (Ex. 1005, 205; Ex. 1002 ¶71.)

Third, because dynamically linked shared libraries were widely known and supported, using them in place of any other type of library would represent nothing more than a simple substitution of one known element (Levine’s dynamically linked library) for another (Ely’s dynamically linked library) to obtain predictable results (dynamically linking a library with applications). (Ex. 1005, 205, 206, 217; Ex. 1002 ¶72.) Moreover, the combination represents using a known technique (Levine’s shared library execution) to improve a similar device and method (a computer executing Ely’s Alpine library) in the same way. (*Id.*) The combination also applies a known technique (Levine’s shared library execution) to a known device and

method (a computer executing Ely’s Alpine library) that is ready for improvement and yields predictable results. (*Id.*)

Fourth, a POSITA would have been motivated to use the ELF shared library format described in Levine to implement Ely’s Alpine library because ELF was one of a finite number of identified, predictable solutions for dynamically linking a library to an application. (Ex. 1002 ¶73; *see also* Ex. 1005, 206 (ELF solution), 217 (DLL solution).)

Finally, Ely discloses using a shared library that is dynamically linked to applications, giving a POSITA a reasonable expectation of success in doing so using the conventional techniques in Levine. (Ex. 1003, 177 (“Alpine exploits dynamic linking” to load “Alpine’s dynamic library before any other dynamic librar[ies]”); Ex. 1002 ¶74.) Additionally, Levine discloses that dynamically linked shared libraries were widely supported and already available in most operating systems, further strengthening the POSITA’s reasonable expectation of success. (Ex. 1005, 206 (“by the late 1990s [dynamic linking] had become the standard”), 217; Ex. 1002 ¶74.)

Ely and Levine disclose each limitation of claims 1-6, 9-14, 16, 18 and thus those claims would have been obvious to a POSITA. (Ex. 1002 ¶75.)

1. Claim 1

Claim 1 recites a computing system comprising (a) a processor, (b) an OS with OSCSEs for running in kernel mode, and (c) a shared library containing

SLCSEs for use by software applications in user mode. (Ex. 1001 at 10:51-59.) The claim further recites several features of the SLCSEs—requiring that (i) some are “functional replicas” of the OSCSEs, (ii) an application runs its own instance of an SLCSE that is not shared with other applications, and (iii) two instances of the same SLCSE are provided to two different applications. (*Id.* at 10:60-11:14.) Ely discloses each of these limitations, expressly or inherently. The limitations that are inherent in Ely are conventional features of dynamically linked shared libraries, which have been widely used in popular operating systems since the 1980s. Levine expressly discloses these conventional dynamic-linking features on which Ely relies.

a. Preamble

The preamble of claim 1 recites “[a] computing system for executing a plurality of software applications.” Ely discloses the preamble. (Ex. 1002 ¶77.) Specifically, Ely discloses that the “Alpine” library runs “on top of a Unix operating system.” (Ex. 1003, 171.) A POSITA would have understood that Unix was a conventional computer operating system for executing a plurality of software applications. (Ex. 1002 ¶77.) Moreover, Ely discloses that “Alpine interfaces with [software] applications” (plural) and was implemented on a computing system. (Ex. 1003, 176; *see also id.* at 178 (disclosing implementation on a “test machine”), Fig.

7 (showing the execution of applications on a computer running Alpine), 180 (“Alpine running in a graphical debugger” with “a telnet application,” and a “mail program and web browser.”) Thus, Ely discloses this limitation. (Ex. 1002 ¶77.)

b. Processor

Claim 1 recites “a processor.” Ely discloses the Alpine runs on a system with a “CPU,” which a POSITA would understand is an abbreviation for central processing unit—a type of processor. (Ex. 1003, 175 (describing how Alpine optimizes “CPU utilization”); Ex. 1002 ¶78.) Ely also discloses a computing system as explained above (*supra*, §VI.A.1.a), which a POSITA would have understood to include a processor as a necessary component. (Ex. 1002 ¶78.) Thus, Ely discloses this limitation. (*Id.*)

c. Operating System with Kernel and OSCSEs

Claim 1 next recites “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.” The ’058 patent admits that this limitation was well known in the prior art. (Ex. 1001, Fig. 1 (“Prior Art” system with OS and kernel 16, and OSCSEs 17), 6:62-7:22 (discussing “conventional architecture where critical system elements execute in kernel mode”).)

Ely also discloses this limitation. (Ex. 1002 ¶80.) Specifically, Ely discloses a “networking stack” that runs in the “kernel” of “traditional operating systems.”

(Ex. 1003, 171.) The services in this networking stack are CSEs. (*Supra*, § V (identifying examples of “critical system elements” in the ’058 patent); Ex. 1001, 3:17-30, 5:35-53, 6:11-15 (“[n]etwork services” are CSEs); *id.*, claim 12.) The traditional kernel-based networking stack is shown in Ely’s Figure 1:

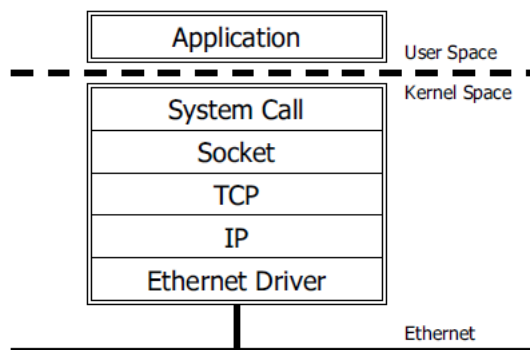


Figure 1: In a traditional network stack, applications interface with the network through a set of system calls (e.g., *socket*, *bind*, *send*), and all network processing is performed in the kernel.

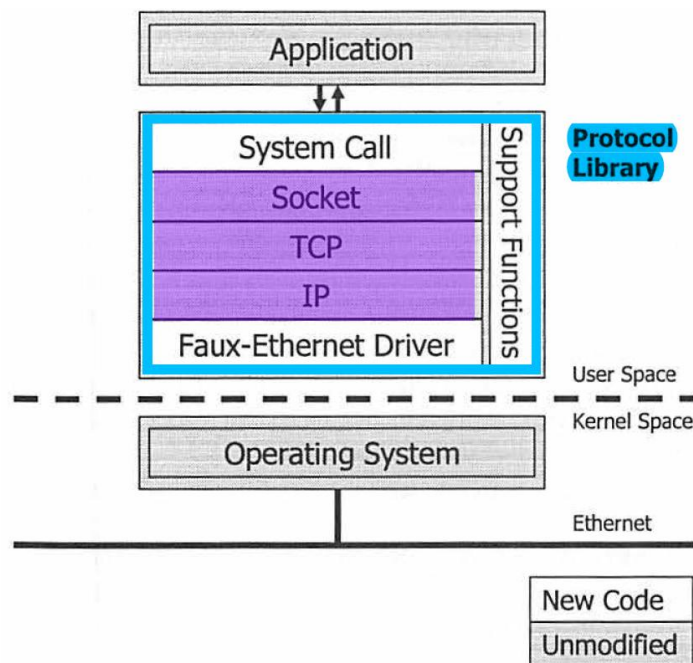
(Ex. 1003, Fig. 1.) In this traditional stack, “all network processing is performed in the kernel.” (*Id.*)

Although Ely’s Alpine library adds another networking stack outside the kernel, the traditional kernel-based networking stack remains in place. (Ex. 1003, 172 (“*No kernel changes.*”), 175 (“Alpine provides a service that is also provided by the kernel”), 178.) Thus, Ely discloses claim 1’s OSCSEs for running in kernel mode. (Ex. 1002 ¶81.)

d. Shared Library with SLCSEs

Claim 1 next recites “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.” Ely discloses this limitation. (Ex. 1002 ¶82.)

Ely discloses placing a networking stack in a “user-level library” called the “[Alpine library](#).” (Ex. 1003, 176, Fig. 2.) This is shown in Figure 2:



(*Id.*, Fig. 2.) The Alpine library includes the “[socket layer and the TCP/IP layer](#)” and is “responsible for sending and receiving packets and maintaining state about connections.” (*Id.*, 173.)

Ely explains that the Alpine library provides networking functionality “also provided by the kernel.” (Ex. 1003, 175; Ex. 1002 ¶84.) Ely thus discloses that both OSCSEs (e.g., elements of a network stack in the kernel) and SLCSEs (e.g., elements

of the Alpine library, also providing a network stack but outside of the kernel) are present in the same system. (Ex. 1003, 176; Ex. 1002 ¶84.)

Ely further discloses that the Alpine library is a “dynamic library,” which was a well-known type of shared library. (Ex. 1003, 177; *supra* §I.A; Ex. 1002 ¶85.) Ely Presentation contains a slide further confirming that Alpine was implemented as a shared library. The slide explains that applications link with Alpine using the command “LD_PRELOAD=libAlpine.so.” (Ex. 1007, 8; *id.* at 3, 6-7; Ex. 1002 ¶85.) The “.so” extension of this “libAlpine” file shows that the file is a shared library. (Ex. 1015 (FreeBSD 3.3 Linker Information Page), line 326 (“By convention, a ‘.so’ extension indicates a shared library.”); Ex. 1002 ¶85.) Thus, Ely discloses a shared library containing SLCSEs for use by applications in user mode. (Ex. 1002 ¶85.)

i. SLCSEs are Functional Replicas of OSCSEs

Claim 1 further recites “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs.” The ’058 patent explains that replicas may be exact replicas or slightly differing replicas. (Ex. 1001, 8:27-40; *see also id.*, 5:20-5:34, 9:52-56; Ex. 1002 ¶86.) Ely discloses both exact replicas and differing replicas. (Ex. 1002 ¶86.)

The exact replicas in Ely include an “unmodified socket layer and the TCP/IP layers” that are “responsible for sending and receiving packets and maintaining state

about connections,” as well as “the same networking API as the kernel.” (Ex. 1003, 173; Ex. 1002 ¶87.) The differing replicas include SLCSEs “that emulate their counterparts in the kernel,” including SLCSEs “dealing with memory allocation,” implementations of kernel “software interrupt functions,” initialization functions, and functions the “networking stack uses ... to handle synchronous events and ... asynchronous events.” (Ex. 1003, 174-75; Ex. 1002 ¶87.) Ely therefore discloses SLCSEs that are functional replicas of OSCSEs. (Ex. 1002 ¶87.)

ii. SLCSEs Form a Part of the Application

Claim 1 further recites that some of the SLCSEs “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.” Ely inherently discloses this limitation. (Ex. 1002 ¶88.) Even if it were not inherent in Ely, it would have been obvious to a POSITA in view of Ely, or alternatively in view of Ely and Levine. (*Id.* ¶88.)

Ely discloses the use of dynamic linking to load the Alpine library. (Ex. 1003, 177; Ex. 1002 ¶89.) The ’058 patent admits that dynamic linking of a shared library of SLCSEs (e.g., the Alpine library) results in the SLCSEs forming a part of each application to which the library is linked. (Ex. 1001, 2:14-15 (SLCSEs “form a part

of at least some of the software applications by being linked thereto”).) Even without this admission, a POSITA would have known dynamic linking worked this way and thus would have found the limitation obvious in view of Ely. (Ex. 1002 ¶89.)

Additionally, the limitation would have been obvious in further view of Levine. (Ex. 1002 ¶90.) Levine explains how shared libraries worked with dynamic linking. A dynamic linker “finds the names of the libraries required by [an application]” and maps the libraries’ contents into the application. (Ex. 1005, 211-12.)² Once a shared library has been mapped into an application, its contents (e.g., executable code and data) are loaded into the application’s address space. (*Id.*, 212-13, 190 (“Each shared library occupies a fixed piece of address space in each program in which it is used.”); Ex. 1002 ¶90.)

The dynamic linking process causes the shared library to form part of the application as described in the ’058 patent and admitted by the applicant during prosecution. (Ex. 1001, 2:12-15; Ex. 1008, 91 (“In other words, the SLCSEs literally form part of the application. SLCSEs reside in the same address space as application

² Levine uses the word “program” rather than “application,” but the words are synonymous. (Ex. 1002 ¶90 n.2.) Indeed, the ’058 patent uses “application” as a shorthand for “application program.” (See, e.g., Ex. 1001, 1:21-28 (using “application programs” and “applications” interchangeably).)

code ...”); Ex. 1002 ¶91.) Thus, Ely’s disclosure of dynamic linking inherently satisfies this limitation, renders the limitation obvious in view of a POSITA’s knowledge, and renders the limitation obvious in view of Levine’s explanation of how dynamic linking worked. (Ex. 1002 ¶91.)

e. CSE Provided to First and Second Applications

i. First Instance of SLCSE Provided to First Application

Claim 1 recites “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.” As discussed above, Ely discloses a shared library containing SLCSEs. (*Supra* §VI.A.1.d.) When an application accesses a shared library, an instance of any shared object used by the application (e.g., an instance of a SLCSE in the Alpine library) runs in the context of the linked application. (*Supra* § I.A; Ex. 1002 ¶92; *see also* Ex. 1008, 23 (“It should be noted that a SLCSE in the context of an application is, for example, TCP IP under the request/control of an application.”).)

Although library code is shared between applications, each *instance* of a library object runs “without being shared” because each application gets a unique copy of private data. (*Supra* § I.A; Ex. 1009, 60-61; Ex. 1013, 100-109; Ex. 1010,

8-10, 8-13, 8-29; Ex. 1002 ¶93.) The '058 patent confirms this understanding of how conventional shared libraries worked and were used in the alleged invention:

In accordance with this invention, the code [is shared] by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services *Each application has its own unique data space.* This indivisible data space ensures that CSEs are unique to an application[.] ... Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. *In this manner CSEs are not shared among applications even though the code is shared.*

(Ex. 1001, 3:30-45 (emphasis added).) Accordingly, a POSITA would have understood that Ely discloses an instance of a SLCSE (e.g., a networking element in the Alpine library) running in the context of an application without that instance being shared. (Ex. 1002 ¶93.)

This limitation also would have been obvious in view of Levine, which explains how shared libraries were implemented in conventional operating systems. (Ex. 1002 ¶94.) Levine discloses that shared libraries are separated into segments, as shown in Figure 9.2 below:

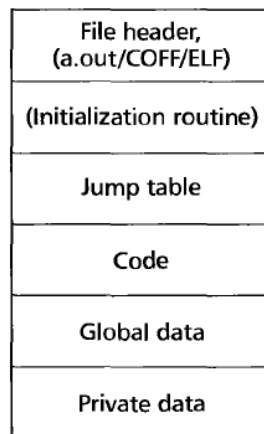


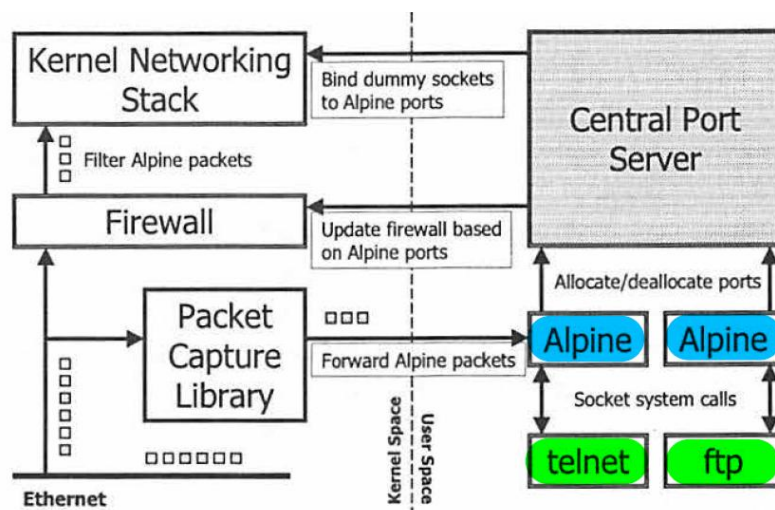
FIGURE 9.2 • Structure of typical shared library.

(Ex. 1005, Fig. 9.2.) These segments are mapped to the address space of each application that accesses the shared library so that the code is executed in each application's own address space. (Ex. 1005, 39, 212-13, 220; Ex. 1002 ¶94.) Unique private data segments are provided to each application that links to a shared library. (Ex. 1005, 38-39 (private writable data is mapped “copy-on-write”), 40 (data pages of a shared library mapped copy-on-write are not shared), 169-172 (each application has its own copy of private data); Ex. 1002 ¶94.)

The '058 patent uses shared libraries in the same way Levine describes. (Ex. 1001, 3:30-45 (“In this manner CSEs are not shared among applications even though the code is shared”).) Accordingly, Ely discloses this limitation and renders it obvious in view of Levine. (Ex. 1002 ¶95.)

**ii. Unique Instance of Corresponding CSE
Provided to Second Application**

Claim 1 recites “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.” As discussed above, each application that links to a dynamic library gets a unique instance of any shared object it uses from the library. (*Supra* § VI.A.1.e.i.) So, when a second application links to the Alpine library, it receives a unique instance of a CSE for performing the same function (e.g., a networking function) as the first application. (Ex. 1002 ¶196.) This scenario is depicted in Ely’s Figure 3, which shows two applications (“telnet” and “ftp”) both using [Alpine](#):



(Ex. 1003, 176 (highlighting added).) Thus, Ely discloses this limitation and renders the limitation obvious in view of Levine. (Ex. 1002 ¶196.)

f. Two Instances of SLCSE Run Simultaneously

i. SLCSE Related to a Predetermined Function

Claim 1 further recites “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.” Ely discloses providing SLCSEs related to predetermined functions—for example the functions required to implement a networking stack. (*Supra* § VI.A.1.d.) Any networking service that an application accesses via Alpine would provide a first instance of an SLCSE to a first application. (*E.g.*, Ex. 1003, 180 (describing “TCP” service that “telnet” application accesses via Alpine), Ex. 1002 ¶97; *see also supra* §VI.A.1.e.i (Ely discloses instances of SLCSEs).) Thus, Ely discloses this limitation. (Ex. 1002 ¶97.)

ii. Second Instance of SLCSE

Claim 1 further recites “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.” As discussed above, when a second application links to the Alpine library, a SLCSE for performing a same function (e.g., a networking function) is provided to that second application for running a second instance of the SLCSE. (*Supra*, § VI.A.1.e.ii.) And both applications run Alpine’s networking stack simultaneously as shown in Figure 3 of Ely, for example. (Ex. 1003 at 176 (“telnet” and “ftp” applications).) Thus, Ely discloses this limitation or

renders it obvious in view of Levine’s explanation of shared libraries. (Ex. 1002 ¶¶98.) Accordingly, claim 1 would have been obvious a POSITA. (*Id.* ¶¶76-98.)

2. Claim 2

Claim 2 depends from claim 1 and further recites “wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.” As discussed above, Ely discloses that multiple instances of an SLCSE run simultaneously within the operating system—e.g., when more than one application uses the Alpine library on one computer. (*Supra* §VI.A.1.f.) Thus, Ely discloses claim 2 or at least renders it obvious in view of Levine. (Ex. 1002 ¶¶99.)

3. Claim 3

Claim 3 depends from claim 1 and further recites “wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.” Ely discloses this limitation. (Ex. 1002 ¶¶100.)

Ely describes the use of networking SLCSEs to “provide[] a service that is also provided by the kernel.” (Ex. 1003, 175.) Ely’s system is implemented “on an unmodified legacy operating system” that retains the kernel networking stack. (*Id.*, 172, Fig. 1 (showing a “traditional network stack” in the operating system kernel); *supra*, §VI.A.1.c; Ex. 1002 ¶¶101.) Thus, Ely discloses OSCSEs (e.g., networking

elements in the kernel) corresponding to and capable of performing the same functions as SLCSEs (e.g., networking elements in the Alpine library). (*Id.*) Accordingly, Ely discloses claim 3 or at least renders it obvious in view of Levine. (Ex. 1002 ¶101.)

4. Claim 4

Claim 4 depends from claim 1 and further recites “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.” Ely discloses this additional limitation. (Ex. 1002 ¶102.)

As discussed above, the individual instance of an SLCSE that is provided to an application is not shared. (*Supra* §VI.A.1.e.i.) Thus, the application has “exclusive use thereof” as claim 4 requires. (Ex. 1002 ¶103.)

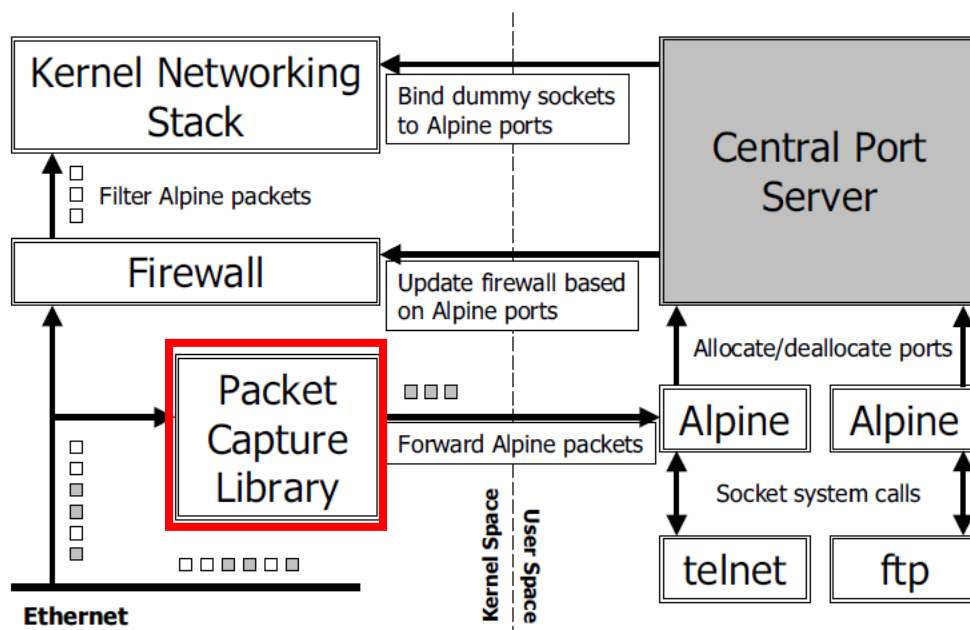
Ely discloses that in some circumstances, its networking SLCSE “must make a *select* call into the kernel” of the operating system. (Ex. 1003, 177; Ex. 1002 ¶104.) The *select* call is a system call in FreeBSD 3.3 and similar operating systems. (Ex. 1002 ¶104; Ex. 1023.) Thus, Ely discloses claim 4 or at least renders it obvious in view of Levine. (Ex. 1002 ¶104.)

5. Claim 5

Claim 5 depends from claim 1 and further recites “wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an

SLCSE in the context of an application program and a device driver.” The ’058 patent defines a “kernel module” as a “set of functions that reside and execute in kernel mode as extensions to the operating system kernel.” (Ex. 1001, 6:56-57.) The patent admits that kernel modules were “common” in conventional systems. (*Id.*, 6:57-59; Ex. 1002 ¶105.) The patent discloses a kernel module that allows CSEs in user mode to communicate with devices operating in kernel mode. (Ex. 1001, 9:60-64, 10:35-43.)

Ely discloses claim 5’s kernel module. (Ex. 1002 ¶106.) Specifically, Ely discloses a kernel-mode “Packet Capture Library” that serves as an interface between an ethernet device driver and Alpine’s SLCSEs:



(Ex. 1003, Fig. 3; Ex.1002 ¶106.) The packet capture library is “adapted to serve as an interface” as claim 5 requires because it receives packets from the ethernet driver

and forwards them to Alpine’s networking stack. (Ex. 1003, Fig. 3; *id.* at 173-74; Ex. 1002 ¶106.) Thus, Ely discloses claim 5 or at least renders it obvious in view of Levine. (Ex. 1002 ¶106.)

6. Claim 6

Claim 6 depends from claim 5 and further recites two “wherein” clauses.

a. Kernel Provides Notification of an Event

Claim 6 recites: “wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program[.]” Ely discloses that its kernel module (the packet capture library) provides notification of an event (the arrival of incoming packets) to an SLCSE (in Alpine’s networking stack) running in the context of an application program. (Ex. 1002 ¶108.) Specifically, Alpine uses an “interrupt handler” to respond to incoming packets. (Ex. 1003, 174.) The interrupt handler polls “libpcap” (the packet capture library) to see whether a packet has arrived, and if so, sends the packet to Alpine’s networking stack for processing. (*Id.*)

b. Event is Asynchronous and Outside of the Application

Claim 6 further recites: “wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.” Ely discloses this limitation because receiving packets from the network is an asynchronous event and information about a packet’s arrival comes from the kernel—outside

the application. (Ex. 1003, 174 (“packets arrive asynchronously” and the “kernel receives an interrupt whenever a packet arrives”) (emphasis omitted); Ex. 1002 ¶109.) Thus, Ely discloses claim 6 or at least renders it obvious in view of Levine. (Ex. 1002 ¶109.)

Ely also discloses an alternative design whereby, rather than polling libpcap, the authors “could have modified the kernel to forward [packet-arrival interrupts] to Alpine using signals.” (Ex. 1003, 174; Ex. 1002 ¶110.) The authors did not implement this alternative design in Alpine but they disclosed it nonetheless. (Ex. 1003, 174; Ex. 1002 ¶110.) For this second, independent reason, Ely discloses claim 6 or at least renders it obvious in view of Levine. (Ex. 1002 ¶110.)

7. Claim 9

Claim 9 depends from claim 2 and further recites “wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.” Ely discloses claim 9’s additional limitation. (Ex. 1002 ¶111.)

As described in the ’058 patent, a “function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead” and a function overlay “is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded.” (Ex. 1001, 8:62-9:8.) Ely discloses exactly this:

[B]y loading Alpine’s dynamic library before any other dynamic library, its networking stack is used instead of the kernel’s. ... The LD_PRELOAD environment variable allows the Alpine dynamic library to be loaded before any other library, which implies that Alpine’s networking stack will be used instead of the kernel’s.

(Ex. 1003, 177; *see also id.*, 178 (“To integrate with unmodified applications, Alpine ... exploit[s] properties of the linker, which allows Alpine to override any system call or library routine[.]”); Ex. 1002 ¶112.) Thus, Ely discloses claim 9 or at least renders it obvious in view of Levine. (Ex. 1002 ¶112.)

8. Claim 10

Claim 10 depends from claim 2 and further recites “wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.” Ely discloses this limitation. (Ex. 1002 ¶113.) Specifically, Ely discloses that “by linking [an application] with the Alpine library before other libraries, Alpine’s networking stack is used instead of the kernel’s.” (Ex. 1003, 177.) Alpine uses “dynamic linking” to accomplish this; the Alpine library is “dynamically linked” with each application that uses it. (Ex. 1003, 177.) Dynamic linking provides an application with unique access to a unique instance of a shared library’s contents (e.g., networking SLCSEs in Alpine). (*Supra* §VI.A.1.e, §I.A; Ex. 1005,

169-172; Ex. 1002 ¶114.) Thus, Ely discloses claim 10 or at least renders it obvious in view of Levine. (Ex. 1002 ¶114.)

9. Claim 11

Claim 11 depends from claim 2 and further recites “wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.” Ely discloses claim 11’s additional limitation. (Ex. 1002 ¶115.)

In conventional operating systems, user-mode processes rely on kernel services for device access, interrupt delivery, and virtual memory mapping because user-mode processes are not permitted to perform these functions themselves. (Ex. 1002 ¶116.) Ely discloses that Alpine was implemented on such conventional operating systems. (Ex. 1003, 171 (“Alpine supports a FreeBSD networking stack on top of a Unix operating system.”), 178 (“Alpine has been fully implemented in the FreeBSD 3.3 operating system.”); Ex. 1002 ¶116.) Thus, a POSITA would understand that Ely’s user-mode SLCSEs relied on kernel services for the recited functions. (Ex. 1002 ¶116.)

Specifically, Ely’s SLCSEs relied on kernel services for access to networking devices when packets were received over the network, as discussed above. (*Supra* §VI.A.5; Ex. 1002 ¶117.) Ely’s SLCSEs also utilized kernel services for interrupt delivery—for example, when the CPU switched between processes. (Ex. 1002

¶117.) Additionally, because the kernel controls virtual memory, Ely’s SLCSEs relied on kernel services for virtual memory mapping when, for example, they were mapped into an application’s address space during the dynamic linking process. (*Supra* §I.A; Ex. 1002 ¶117.) Thus, Ely discloses claim 11 or at least renders it obvious in view of Levine. (Ex. 1002 ¶117.)

10. Claim 12

Claim 12 depends from claim 1 and further recites “wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.” Ely discloses SLCSEs implementing a user-space networking stack including networking protocol processes. (*Supra* §VI.A.1.d; Ex. 1003, 171 (Alpine “moves network protocols into a user-level library”); Ex. 1002 ¶118.) Thus, Ely discloses claim 12 or at least renders it obvious in view of Levine. (Ex. 1002 ¶118.)

11. Claim 13

Claim 13 depends from claim 10 and further recites “wherein some SLCSEs are modified for a particular one of the plurality of software applications.” Ely discloses that the Alpine user-level networking stack “can be examined and changed just as any other part of the application source code” and that “Alpine provides per-application specialization.” (Ex. 1003, 179, 172; Ex. 1002 ¶119.) Thus, Ely discloses claim 13 or at least renders it obvious in view of Levine. (Ex. 1002 ¶119.)

12. Claim 14

Claim 14 depends from claim 13 and further recites “wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.” Ely discloses that its Alpine networking stack (which contains the SLCSEs and runs in user mode) can be application-specific. (*Supra* §VI.A.11.) Further the Alpine networking stack coexists with an unmodified kernel, which retains the kernel networking stack. (Ex. 1003, 171 (conventional “networking stack resides in the kernel”), 172 (design constraint required “[n]o kernel changes”) (emphasis omitted), 180 (“telnet” application using Alpine networking stack while “web browser” used kernel networking stack).) The kernel networking stack includes CSEs. (*Supra*, §VI.A.1.c.)

The CSEs in the kernel networking stack are platform-specific. According to the '058 patent, a “[c]ompute platform” is the “combination of computer hardware and a single instance of an operating system” (OS). (Ex. 1001, 6:29-30.) Thus, a component is platform-specific if it is (1) hardware-specific or (2) OS-specific. (Ex. 1002 ¶120; *see* Ex. 1001, 7:1-7 (distinguishing “platform independent” libraries.) The CSEs in Ely’s kernel networking stack are platform-specific in both ways. First, the kernel stack is hardware-specific because it includes an “Ethernet Driver,” which controls network interface hardware. (Ex. 1003, 172 (Fig. 1); Ex. 1002 ¶120.) Second, the kernel stack comes from FreeBSD 3.3. (Ex. 1003, 178-79.) This BSD

networking stack is OS-specific because it differs from networking stacks in other operating systems such as Linux. (Ex. 1002 ¶120; *see* Ex. 1001, 5:41-47 (distinguishing BSD stack from networking stack used by “Linux platform”).) Thus, Ely discloses claim 14 or at least renders it obvious in view of Levine. (Ex. 1002 ¶120.)

13. Claim 16

Claim 16 depends from claim 1 and further recites “wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.” Ely and Levine render obvious claim 16 for reasons explained above in connection with claim 1. (*Supra* §VI.A.1.d.ii; Ex. 1002 ¶121.)

14. Claim 18

Claim 18 depends from claim 2 and further recites “wherein SLCSEs are not copies of OSCSEs.” Ely discloses this limitation. A primary purpose of the Alpine library is to support development of “protocol modifications or new network protocols.” (Ex. 1003, 171.) These modifications and new protocols are necessarily different than the *unmodified* network protocols in the operating system kernel. (Ex. 1002 ¶122.) Thus, Ely discloses that the SLCSEs in Alpine (modified networking services) are not copies of the OSCSEs in the kernel (unmodified networking services). (*Id.*; *see also* Ex. 1003, 172 (“Alpine provides per-application specialization”), 179 (“Alpine running TCP Daytona,” which is “a set of modifications” to

TCP networking); Ex. 1002 ¶122.) Thus, Ely discloses claim 18, or at least renders it obvious in view of Levine. (Ex. 1002 ¶122.)

B. Ground 2: Ely, Levine, and Thekkath

The claims addressed in this Ground depend (directly or indirectly) from claim 1. As discussed above, Ely and/or Levine disclose each limitation of claim 1. (*Supra* §VI.A.1.) Like Ely, Thekkath discloses a networking “protocol library” linked to an application in user space. (Ex. 1006, 67.) Thekkath further discloses modifying an operating system’s kernel to include a “Network I/O Module,” which facilitates communication between the user-level protocol library and the network. (*Supra*, §I.B.2.)

A POSITA would have been motivated to combine the teachings of Thekkath and Ely (with or without Levine). (Ex. 1002 ¶124.) Specifically, a POSITA would have been motivated to replace Ely’s packet capture approach (which passes network packets to the user-level networking library) with Thekkath’s network I/O module (which performs the same function more efficiently) for the reasons set forth below. (*Id.*)

First, Ely states that its “approach is similar to that proposed in [Thekkath],” and cites Thekkath. (Ex. 1003, 176, 183 (citation “[29]”); Ex. 1002 ¶125.)

Second, Ely and Thekkath address the same problems of slow development and debugging of networking protocols implemented in the kernel. (*E.g.*, Ex. 1003, 171; Ex. 1006, 64; Ex. 1002 ¶126.)

Third, replacing Ely’s packet capture with Thekkath’s network I/O module would represent nothing more than a simple substitution of one known element (Thekkath’s module) for another (Ely’s packet capture) to obtain predictable results (accessing the network from a user-level protocol library). (Ex. 1002 ¶127.)

Fourth, Ely expressly recognizes the potential benefits of modifying the kernel to facilitate user-level networking, and Thekkath discloses how to make such modifications. (Ex. 1002 ¶128.) Ely avoided kernel modifications only because of a self-imposed “design constraint.” (Ex. 1003, 174.) The purpose of this constraint was to make deployment of the Alpine library easier for users. (*Id.*, 172.) But Ely also disclosed that relaxing this constraint and allowing kernel modification could lead to higher performance. (*See id.* (“Since our design constraints include requiring no kernel ... modifications, Alpine cannot achieve the high-performance of [alternative] systems”).) Based on this disclosure, a POSITA interested in maximizing performance would have been motivated to modify the kernel instead of relying on Ely’s lower-performance packet-capture approach. (Ex. 1002 ¶128.) Thekkath discloses how to implement such a kernel modification, in the form of its Network I/O

module. (Ex. 1006, 67-69, Ex. 1002 ¶128.) Thekkath also describes how its disclosed implementation improved performance. (Ex. 1006, 70 (showing that Thekkath’s implementation achieved higher throughput than the earlier “Mach 3.0/UX” user-level networking implementation); Ex. 1002 ¶128.)

Fifth, the combination represents using a known technique (signaling from a kernel module as in Thekkath) to improve a similar device and method (Ely’s user-level networking library) in the same way. (Ex. 1002 ¶129.) The combination also applies a known technique (signaling from a kernel module as in Thekkath) to a known device and method (Ely’s user-level networking library) that is ready for improvement and yields predictable results. (*Id.*)

Finally, because Ely uses an approach similar to Thekkath’s approach to solve the same problems, a POSITA would have had a reasonable expectation of success in making the combination. (Ex. 1003, 176; Ex. 1002 ¶130.)

The combination of Thekkath and Ely (with or without Levine) renders obvious claims 5-8 and 15 as explained below. (Ex. 1002 ¶¶131.)

1. Claim 5

Claim 5 depends from claim 1 and further recites “wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.” Thekkath in combination with Ely discloses this limitation. (Ex. 1002 ¶132.) Thekkath discloses

that its protocol library “is linked into [each] application” that uses it and “contains the code that implements the communication protocol,” but cannot access device drivers such as the network device drivers directly. (Ex. 1006, 67.) Instead, the protocol library relies on a network I/O module, which sends and receives data from the protocol library and functions “to provide efficient and protected access to the network by the libraries.” (*Id.*, 67-69, Fig. 2.) Thekkath’s network I/O module “is co-located with the network device driver” in the kernel. (Ex. 1006, 67; Ex. 1002 ¶132.) These components are shown in Figure 2 of Thekkath:

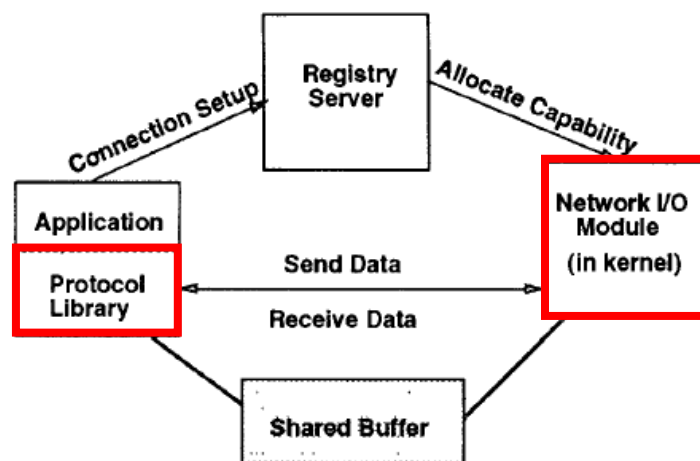


Figure 2: Structure of the Protocol Implementation

(Ex. 1006, Fig. 2 (annotated); Ex. 1002 ¶132.)

As explained above, a POSITA would have been motivated to combine Thekkath’s network I/O module with Ely’s Alpine library, as a replacement for Ely’s packet capture approach. (*Supra*, p. 44) In this combination, Thekkath’s module would serve as an interface to the networking SLCSEs in Ely’s Alpine library. (*Id.*;

Ex. 1002 ¶133.) These SLCSEs run in the context of an application program as explained in connection with claim 1. (*Supra*, §VI.A.1.e.i.) Thus, claim 5 would have been obvious in view of Thekkath and Ely. (Ex. 1002 ¶133.)

2. Claim 6

Claim 6 depends from claim 5 and recites two additional “wherein” clauses.

a. Kernel Provides Notification of an Event

Claim 6 recites: “wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program[.]” Thekkath discloses that when the kernel’s network I/O module receives a packet, “a notification is sent to the application library via a lightweight semaphore.” (Ex. 1006, 69; *see also id.*, 67; Ex. 1002 ¶135.) The application library contains SLCSEs and runs in the context of an application program as explained above in connection with Ely. (*Supra*, §VI.A.1.e.i.)

b. Event is Asynchronous and Outside of the Application

Claim 6 further recites: “wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.” The event that triggers the notification in Thekkath is the receipt of a network packet. (Ex. 1006, 68-69.) Receipt of a network packet is asynchronous. (Ex. 1003, 174-75 (“processing of incoming packets is asynchronous”).) Thekkath’s system requires information about this asynchronous event to be passed to the networking

stack (SLCSEs) from outside of the application. (Ex. 1006, 68-69 (“[i]ncoming packets from the network are moved into the shared [memory] region” where the application-level network protocols can access them).) Thus, claim 6 would have been obvious in view of Thekkath and Ely. (Ex. 1002 ¶136.)

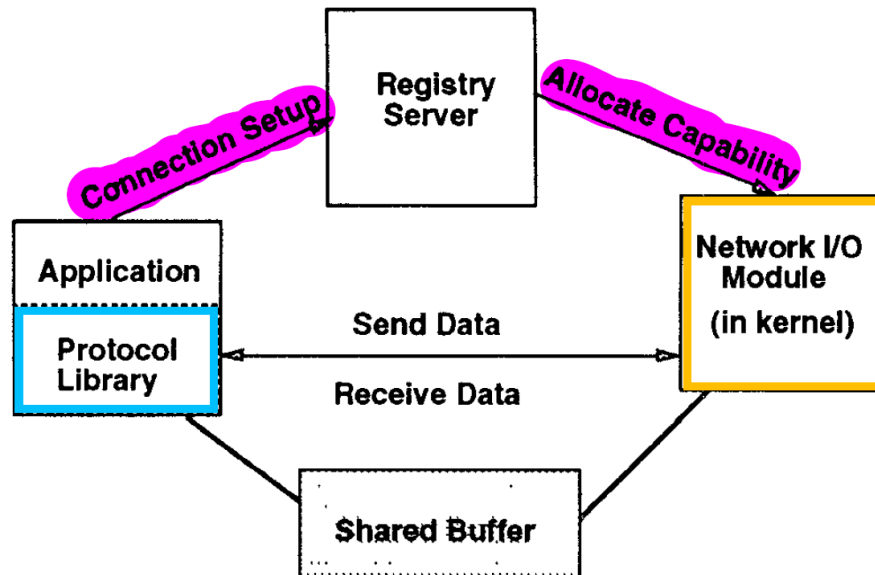
3. Claim 7

Claim 7 depends from claim 6 and further recites “wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.” Again, the ’058 patent defines an up call mechanism as a “means by which a service in kernel mode executes a function in a user mode application.” (Ex. 1001, 6:60-61.) Thekkath discloses the use of an upcall mechanism in the form of a “semaphore” to provide “network packet arrival notification” from the kernel to the user-level network stack. (Ex. 1006, 68.) This notification causes the user-level network stack (running in the context of a user-mode application) to perform the function of responding to the packet. (*Id.*; Ex. 1002 ¶137.) Thekkath expressly describes this user-level code as being “upcalled.” (Ex. 1006, 68.) Thus, claim 7 would have been obvious in view of Thekkath and Ely. (Ex. 1002 ¶137.)

4. Claim 8

Claim 8 depends from claim 7 and further recites “wherein the upcall mechanism in operation, executes instructions from an SLCSE resident in user mode space,

in kernel mode.” Thekkath discloses claim 8’s additional limitation. Specifically, Figure 2 of Thekkath shows that networking SLCSEs in the **protocol library** provide “**connection setup**” instructions to a registry server, which then configures the **kernel’s network I/O module**:



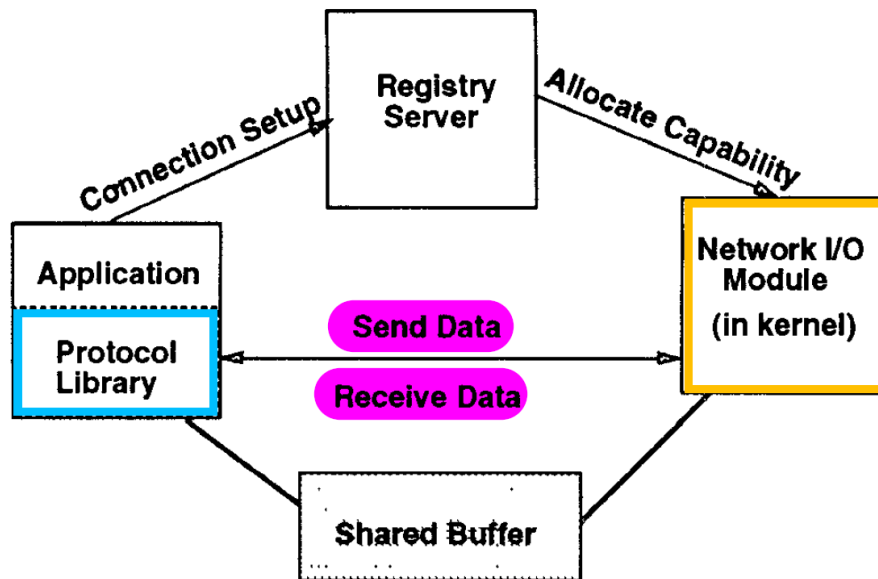
(Ex. 1006, Fig. 2; *id.* at 69; Ex. 1002 ¶138.) Because the network I/O module is in the kernel, a POSITA would understand that it executes the configuration instructions it receives in kernel mode. (Ex. 1006, 69 (library cannot access incoming packets until it provides connection-setup instructions that are used to configure the network I/O module); *see also id.* at Fig. 2, 67-68; Ex. 1002 ¶138.) Thus, claim 8 would have been obvious in view of Thekkath and Ely. (Ex. 1002 ¶138.)

5. Claim 15

Claim 15 depends from claim 5 and recites two additional “wherein” clauses.

a. Kernel Module Enables Data Exchange

Claim 15 recites: “wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode[.]” Thekkath discloses that the **kernel’s network I/O module** and the user-space SLCSEs in the **protocol library** can **send and receive data** to and from each other:



(Ex. 1006, Fig. 2; Ex. 1002 ¶140.) Thekkath’s network I/O module is “located with the in-kernel network device driver” and is used “to provide efficient and protected access to the network” from the user-mode library. (Ex. 1006, 69; Ex. 1002 ¶140.)

b. Data Exchange Uses Virtual Memory Mapping

Claim 15 further recites: “wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSE in user mode.” Thekkath discloses that data is exchanged back and forth using a

“virtual memory region in the library [that] is mapped shared with the particular I/O module for the network device that the connection is using.” (Ex. 1006, 68; *id.* (“This shared memory region is used to convey data between the protocol and the network device.”); Ex. 1002 ¶141.) Thus, claim 15 would have been obvious to a POSITA in view of Ely and Thekkath (with or without Levine). (Ex. 1002 ¶141.)

C. Ground 3: Eggert and Levine

Eggert discloses a dynamically linked library that extends a computer’s default file system with added capabilities. (*Supra*, §I.B.3.) A POSITA would have understood that Eggert’s dynamically linked library inherently worked in accordance with Levine’s disclosure of how such libraries were conventionally implemented. (Ex. 1002 ¶142.) Indeed, Eggert specifically discloses that its library was “implemented atop SunOS 4.1.x.” (Ex. 1004, 237.) As Levine explains, the “SunOS” operating system “introduced dynamic shared libraries to UNIX in the late 1980s.” (Ex. 1005, 206.) Because Eggert was published in 1993, it used a version of SunOS that supported such dynamically linked libraries. (Ex. 1002 ¶142; Ex. 1029, 37 (SunOS 4 release announcement describing dynamically linked shared libraries).)

Regardless, even if Eggert did not inherently use the sort of shared libraries that Levine describes, combining these references would have been obvious. A

POSITA would have been motivated to implement Eggert’s dynamically linked library in accordance with Levine’s disclosure for several reasons. (Ex. 1002 ¶143.)

First, Eggert expressly teaches linking its library to applications using dynamic linking technology that was widely available. (Ex. 1004, 229 (“The recent rise of dynamic linking provides a new way for users to develop their own file systems”), 235 (“any dynamically linked program automatically incorporates IFS1”).) Based on this teaching, a POSITA would have looked to Levine for an explanation of how such linking was conventionally implemented. (Ex. 1002 ¶144.)

Second, Eggert disclosed a goal of making the work of “[c]hanging file systems” and “writing [] new file system[s]” simpler than previous approaches. (Ex. 1004, 229; Ex. 1002 ¶145.) This disclosure would have motivated a POSITA to look to Levine, which discloses that its “[d]ynamically linked shared libraries are easier to update than statically linked shared libraries.” (Ex. 1005, 205; Ex. 1002 ¶145.)

Third, because the conventional shared libraries in Levine were widely known and supported, using them in place of any other type of library would represent nothing more than a simple substitution of one known element (Levine’s conventional shared library) for another (any other type of library that Eggert could have used) to obtain predictable results (dynamic linking of the shared library). (Ex. 1005, 205,

206, 217; Ex. 1002 ¶146.) Moreover, the combination uses a known technique (dynamic linking as in Levine) to improve a similar device and method (a computer running Eggert’s library) in the same way. (*Id.*) The combination also applies a known technique (dynamic linking as in Levine) to a known device and method (a computer running Eggert’s library) that is ready for improvement and yields predictable results. (*Id.*)

Finally, because Levine discloses that conventional shared libraries were widely supported and already available in common operating systems, a POSITA would have had a reasonable expectation of success in using them to implement Eggert’s IFS1 library. (Ex. 1005, 206, 217; Ex. 1002 ¶147.)

Eggert and Levine disclose each limitation of claims 1-4, 9-12, 16, and 18 and render those claims obvious to a POSITA. (Ex. 1002 ¶148.)

1. Claim 1

a. Preamble

The preamble recites “[a] computing system for executing a plurality of software applications.” Eggert discloses a file system extension used by applications running on a computing system. (Ex. 1004, 229, 237; Ex. 1002 ¶149.)

b. Processor

Claim 1 recites “a processor.” Eggert discloses a system that utilizes “cycles” on “client processors,” and therefore discloses this limitation. (Ex. 1004, 235; Ex.

1002 ¶150.) In addition, a POSITA would have understood Eggert to implicitly disclose a processor as an essential component of conventional computing systems. (Ex. 1002 ¶150.)

c. Operating System with Kernel and OSCSEs

Claim 1 next recites “an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.” The patent admits this limitation was well known. (*Supra*, §VI.A.1.c.)

Additionally, Eggert discloses that its file system extensions are “implemented atop SunOS 4.1.x,” a Unix-based operating system that includes a kernel. (Ex. 1004, 237; Ex. 1002 ¶152.) Eggert’s file system extensions improve on the traditional file system, located in the kernel. (Ex. 1004, 229 (“Changing [existing] file systems requires kernel modifications or root privileges”).) The kernel’s file system implementation contains OSCSEs. (Ex. 1001, 6:6-8 (defining CSEs), 6:11-22 (CSEs include file system implementations); Ex. 1002 ¶152.) Thus, Eggert discloses this limitation. (Ex. 1002 ¶152.)

d. Shared Library with SLCSEs

Claim 1 recites “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.” Eggert discloses this limitation. (Ex. 1002 ¶153.)

Eggert discloses “user-extensible file systems” that extend the kernel’s conventional file system. (Ex. 1004, 229; Ex. 1002 ¶154.) One such file system, IFS1, extends the conventional file system to permit more flexible pathnames. (Ex. 1004, 229; Ex. 1002 ¶154.) These file system extensions fit the patent’s definition of CSEs. (Ex. 1001, 6:11-15 (“CSEs include: ... File System services that offer extensions to those supplied by the OS[.]”); Ex. 1002 ¶154.)

Eggert further discloses that these CSEs are implemented using a dynamically linked shared library that “works entirely in user space.” (Ex. 1004, 236 (IFS uses an approach that is “automatically adopted by all dynamically linked programs”), 235 (“any dynamically linked program automatically incorporates IFS1”), 237 (“dynamic linker” is used “to preload IFS1”); Ex. 1002 ¶155.) Thus, Eggert discloses this limitation. (Ex. 1002 ¶155.)

i. SLCSEs are Functional Replicas of OSCSEs

Claim 1 recites “wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs.” The ’058 patent defines the term “replica” to mean “a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS).” (Ex. 1001, 1:66-2:1.) The file system extensions in Eggert’s library are functional replicas of the kernel’s file system because they perform a similar function—accessing files using a pathname. (Ex. 1004,

231 (conventional file systems support “[e]xtensional pathnames” and Eggert extends this by also supporting “intensional pathnames, which include shell-evaluable expressions”), 233; Ex. 1002 ¶156.) Because Eggert’s SLCSEs provide similar functionality to the kernel’s conventional OSCSEs (but are not an exact copy), Eggert’s SLCSEs are “functional replicas” of the kernel’s OSCSEs. (Ex. 1001, 8:27-37; Ex. 1002 ¶156.) Thus, Eggert discloses this limitation. (Ex. 1002 ¶156.)

ii. SLCSEs Form a Part of the Application

Claim 1 further recites that some of the SLCSEs “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.” Eggert discloses the use of dynamic linking to provide applications with access to the IFS1 library. (Ex. 1004, 229, 235, 237; Ex. 1002 ¶157.) Because Eggert’s file system SLCSEs are dynamically linked, they form part of the applications to which they are linked. (See Ex. 1001, 2:14-15 (SLCSEs “form a part of at least some of the software applications *by being linked thereto*”) (emphasis added).)

To the extent that Eggert does not disclose this limitation, it would have been obvious for the same reasons discussed above in Ground 1. (*Supra* §VI.A.1.d.ii; Ex. 1002 ¶158.) Accordingly, Eggert and Levine disclose, or at least suggest, this limitation. (Ex. 1002 ¶158.)

e. CSE Provided to First and Second Applications

i. First Instance of SLCSE Provided to First Application

Claim 1 recites “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.” As discussed above, Eggert discloses a dynamically linked library containing SLCSEs. (*Supra*, §VI.C.1.d.) When an application accesses a dynamically linked library, an instance of each shared object used by the linked application (e.g., an instance of each SLCSE in Eggert’s IFS1 library) runs in the context of that application. (*Supra* §I.A; *see also* Ex. 1004, 233-34 (describing “Sample IFS Applications”); Ex. 1002 ¶159.) Although library code is shared between applications, each *instance* of a library object runs “without being shared” because each application gets a unique copy of private data for that object. (*Supra* §I.A; Ex. 1001, 3:30-45 (explaining how conventional shared libraries worked and were used in the alleged invention).) Accordingly, Eggert discloses this limitation. (Ex. 1002 ¶159.)

This limitation also would have been obvious in view of Levine, which explains how shared libraries were implemented in conventional operating systems. (*Supra* §VI.A.1.e.i; Ex. 1002 ¶160.)

**ii. Unique Instance of Corresponding CSE
Provided to Second Application**

Claim 1 recites “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.” As discussed above, each application that links to a dynamic library gets a unique instance of any shared object it uses from the library. (*Supra* §VI.C.1.e.i.) So, when a second application links to Eggert’s IFS1 library, it receives a unique instance of a CSE for performing the same function (e.g., accessing files) as the first application. (Ex. 1002 ¶161; Ex. 1004, 234 (two applications, “diff” and “grep,” each using IFS to access files.) Thus, Eggert discloses this limitation or at least renders it obvious in view of Levine. (Ex. 1002 ¶161.)

f. Two Instances of SLCSE Run Simultaneously

**i. SLCSE Related to a Predetermined
Function**

Claim 1 further recites “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.” Eggert discloses providing SLCSEs related to predetermined functions—for example the functions required to implement file-system extensions. (*Supra* §VI.C.1.d.) Any file-system extension that an application accesses via IFS would provide a first instance of an SLCSE to a first application. (*E.g.*, Ex.

1004, 234 (file-versioning extension that “diff” application accesses via IFS); Ex. 1002 ¶162; *see also supra* §VI.C.1.e.i (explaining how Eggert discloses instances of SLCSEs).) Thus, Eggert discloses this limitation. (Ex. 1002 ¶162.)

ii. Second Instance of SLCSE

Claim 1 further recites “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.” As discussed above, when a second application links to the IFS library, a SLCSE for performing a same function (e.g., a file-system function) is provided to that second application for running a second instance of the SLCSE. (*Supra*, § VI.C.1.e.ii.) A POSITA would recognize that running two applications at the same time was very common in operating systems like SunOS, which Eggert used. (Ex. 1002 ¶163; Ex. 1004, 237.) When two applications for accessing the file system were used at the same time, the two instances of the SLCSE would run simultaneously. (Ex. 1002 ¶163.)

Thus, claim 1 would have been obvious to a POSITA in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶164.)

2. Claim 2

Claim 2 depends from claim 1 and further recites “wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.” As discussed above regarding claim 1, multiple instances of

Eggert’s SLCSEs run simultaneously when a shared library is dynamically linked to multiple applications that are running at the same time. (*Supra* §VI.C.1.f.ii; Ex. 1002 ¶165.) A POSITA would have understood that running multiple applications that access a file system at the same time was very common. (Ex. 1002 ¶165.) Thus, claim 2 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶165.)

3. Claim 3

Claim 3 depends from claim 1 and further recites “wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.” Eggert discloses that SLCSEs for evaluating pathnames are implemented in a shared library while OSCSEs for performing the same function—evaluating pathnames—remain in the kernel. (Ex. 1004, 233 (pathnames are evaluated first in the kernel and then in IFS1), 236 (“This approach requires no kernel modifications[.]”); Ex. 1002 ¶166.) Thus, claim 3 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶166.)

4. Claim 4

Claim 4 depends from claim 1 and further recites “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.” Eggert discloses that a file system extension in IFS1 intercepts a system call, determines

a new pathname, and then “retries the system call with the new pathname.” (Ex. 1004, 233, 236 (the user-space file system “operates as a layer between the actual system calls and the virtual [system] call”); Ex. 1002 ¶167.) Eggert thus discloses that its SLCSEs use system calls to access whatever kernel services the linked application requires using the pathnames generated by the SLCSEs. (Ex. 1002 ¶167.) Accordingly, claim 4 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶168.)

5. Claim 9

Claim 9 depends from claim 2 and further recites “wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.” The ’058 patent explains that a “function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead.” (Ex. 1001, 8:62-64.) Function overlays may be used “to intercept software application accesses to operating system services.” (*Id.*, 8:65-67.) “The overlay is accomplished by [providing] a library to be preloaded before other libraries are loaded.” (*Id.*, 9:1-8.)

Eggert discloses that “a user can tell the dynamic linker to preload IFS1 before linking the standard C library[.]” (Ex. 1004, 237.) This preloading allows “redefinition of the basic system call interface in [the standard C library].” (*Id.*, 236; Ex.

1002 ¶170.) As a result, IFS1 can intercept system calls from applications and provide extensions that change how the system calls are handled by the operating system. (Ex. 1004, 233, 236, 237; Ex. 1002 ¶170.) Thus, claim 9 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶170.)

6. Claim 10

Claim 10 depends from claim 2 and further recites “wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.” Eggert discloses the use of dynamic linking which, as discussed above, links a shared library with an application as the application is loaded and provides each linked application with unique access to a unique instance of the shared library’s contents (e.g., SLCSEs). (*Supra* §I.A; Ex. 1004, 229, 235-37; Ex. 1002 ¶171.) Thus, claim 10 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶171.)

7. Claim 11

Claim 11 depends from claim 2 and further recites “wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.” Eggert discloses and/or renders obvious this limitation. (Ex. 1002 ¶172.) Eggert’s file system extension is implemented

on the Unix-based SunOS operating system. (Ex. 1004, 237; Ex. 1002 ¶172.) As with other conventional operating systems, user-mode processes in SunOS rely on kernel services for device access, interrupt delivery, and virtual memory mapping. (*Supra* §VI.A.9; Ex. 1002 ¶172.)

Eggert’s SLCSEs rely on kernel services for device access, for example, when using system calls to access files stored on disk. (Ex. 1004, 233; Ex. 1002 ¶173.) Eggert’s SLCSEs also utilize kernel services for interrupt delivery and virtual memory mapping, like all user processes in conventional Unix-based operating systems. (Ex. 1002 ¶173.) For example, Eggert’s SLCSEs rely on kernel services for interrupt delivery when the CPU switches between processes and for virtual memory mapping when SLCSEs are mapped into each application’s address space during the dynamic linking process. (*Supra* §VI.A.9; Ex. 1002 ¶173.) Thus, claim 11 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶173.)

8. Claim 12

Claim 12 depends from claim 1 and further recites “wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.” Eggert discloses SLCSEs related to management of files. (*Supra*, §VI.C.1.d; Ex. 1004, 230; Ex. 1002 ¶174.) Thus, claim 12 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶174.)

9. Claim 16

Claim 16 depends from claim 1 and further recites “wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.” As discussed above, Eggert discloses dynamically linking SLCSEs with applications, which causes SLCSEs to form a part of those applications. (*Supra*, §VI.C.1.d, §I.A; Ex. 1002 ¶175.) Thus, claim 16 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶175.)

10. Claim 18

Claim 18 depends from claim 2 and further recites “wherein SLCSEs are not copies of OSCSEs.” As discussed above, Eggert discloses SLCSEs that extend the functionality of the kernel’s file system rather than copying it. (*Supra* §VI.C.1.d; Ex. 1002 ¶176.) Thus, claim 18 would have been obvious in view of Eggert by itself, or in combination with Levine. (Ex. 1002 ¶176.)

VII. NO SECONDARY CONSIDERATIONS OF NONOBVIOUSNESS

Secondary considerations should be considered but do not control the obviousness conclusion. *Newell Cos. v. Kenney Mfg. Co.*, 864 F.2d 757, 768 (Fed. Cir. 1988). Where, as here, a strong *prima facie* obviousness showing exists, any evidence of secondary considerations may not dislodge the obviousness conclusion. *Leapfrog Enters. v. Fisher-Price, Inc.*, 485 F.3d 1157, 1162 (Fed. Cir. 2007). Petitioner is aware of no evidence supporting a claim for secondary considerations.

VIII. NO BASIS FOR DISCRETIONARY DENIAL UNDER §314(A)

Efficiency, fairness, and the merits support institution despite the existence of co-pending litigation. *Apple v. Fintiv*, IPR2020-00019, Paper 11 (P.T.A.B. Mar. 20, 2020) (“*Fintiv*”). Petitioner hereby stipulates that, should this Petition be instituted, Petitioner will not pursue in the Litigation any ground of invalidity raised, or that could reasonably have been raised, herein. This stipulation suffices to avoid exercise of the Board’s discretion under 35 U.S.C. §314(a). *Sotera Wireless, Inc. v. Masimo Corp.*, IPR2020-01019, Paper 12 (P.T.A.B. Dec. 1, 2020); Katherine K. Vidal, *Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation* (June 21, 2022) (“Director Memo”).

The remaining *Fintiv* factors also support institution. The Litigation has not progressed far. The parties completed venue discovery but discovery on the merits opened less than three weeks ago. (Ex. 1017 at 2.) Since merits discovery opened, the parties have not taken any depositions or served any discovery responses. Also, the district court has not construed the claims. (Ex. 1018.) Although a trial date was set, that date will be vacated because the district court granted Petitioner’s motion for transfer to a more convenient district. (Ex. 1019.) The district court also stayed all deadlines while the transfer is finalized. (*Id.* at 6.) No schedule or trial date has been set in the transferee district.

Additionally, the merits of this Petition are compelling, as demonstrated above. See Director Memo. Further, denying institution would negate Congress’s intent in providing a 1-year period to file petitions and would encourage forum shopping as patent owners look to shield their patents from PTAB scrutiny by seeking judges with aggressive case schedules.

Accordingly, the Board should decline to exercise discretion under §314(a).

IX. NO BASIS FOR DISCRETIONARY DENIAL UNDER §325(D)

The grounds here rely on Ely, Eggert, Thekkath, and/or Levine, none of which the Office has considered. Nor has the Office considered “substantially the same prior art or arguments,” 35 U.S.C. §325(d). In prosecution, PO argued that the references the Examiner applied did not teach shared libraries that replicate CSEs from an operating system. (*Supra*, § II.B.) However, the references here clearly disclose this. (*Supra*, §VI.A.1.d.i (Ely), §VI.C.1.d.i (Eggert).)

The ’058 patent is also involved in *ex parte* reexamination (“EPR”). (Ex. 1032.) None of the references in this Petition are at issue in the EPR. Further, the three-reference combination in the EPR (Lucas, Blott, and Kamp) uses references that lack key disclosure present in each of the Ely and Eggert references here. For example, Ely and Eggert disclose CSEs in dynamically linked libraries. (*Supra*, §VI.A.1.d, §VI.C.1.d.) The EPR relies on Blott for its disclosure of CSEs in librar-

ies, but Blott never discloses dynamic linking. (See Ex. 1031, 21-22.) The disclosure in Eggert and Ely of dynamic linking is material to multiple claim limitations. (E.g., *supra*, §VI.A.1.d, §VI.A.1.d.ii, §VI.A.1.e.ii, §VI.C.1.d, §VI.C.1.d.ii, §VI.C.1.e.ii.) Thus, the prior art here is materially different from the EPR. See *Samsung Electronics Co., Ltd. v. Maxell, Ltd.*, IPR2024-00907, 2024 WL 5079910, at *5-6 (PTAB Dec. 11, 2024) (instituting IPR despite previous reexamination).

Additionally, the EPR is limited to only two claims (1 and 12). The other fifteen claims here are not at issue in the EPR. Further, no rejection and no allowance has been issued in the EPR. See *Juniper Networks, Inc. v. Monarch Networking Solutions LLC*, IPR2024-00175, 2024 WL 2054958, at *11-12 (PTAB May 8, 2024); *Mueller Systems, LLC v. Rein Tech, Inc.*, IPR2020-00098, 2020 WL 2478514, at *5 (May 12, 2020) (instituting IPR when EPR was incomplete). Thus, the Board should not deny institution under §325(d).

X. MANDATORY NOTICES

A. Real Parties-In-Interest (37 C.F.R. §42.8(b)(1))

Amazon.com, Inc., Amazon.com Services LLC, and Amazon Web Services, Inc. are the real parties-in-interest.

B. Related Matters (37 C.F.R. §42.8(b)(2))

The '058 patent is the subject of *ex parte* reexamination under Control No. 90/019,676. To the best knowledge of Petitioner, the '058 patent is or has been involved in the following district court litigations:

Name	Number	Court	Filed
<i>VirtaMove, Corp. v. Amazon.com, Inc.</i>	7:24-cv-00030	W.D. Tex.	Jan. 26, 2024
<i>VirtaMove, Corp. v. Google LLC</i>	7:24-cv-00033 3:25-cv-00860	W.D. Tex. N.D. Cal.	Jan. 31, 2024
<i>VirtaMove, Corp. v. Int'l Business Machines Corp.</i>	2:24-cv-00064	E.D. Tex.	Jan. 31, 2024
<i>VirtaMove, Corp. v. Hewlett Packard Enterprise Co.</i>	2:24-cv-00093	E.D. Tex.	Feb. 9, 2024
<i>Red Hat, Inc. v. VirtaMove, Corp.</i>	5:24-cv-04740	N.D. Cal.	Aug. 5, 2024
<i>VirtaMove, Corp. v. Microsoft Corporation</i>	7:24-cv-00338	W.D. Tex.	Dec. 20, 2024
<i>VirtaMove, Corp. v. Oracle Corporation</i>	7:24-cv-00339	W.D. Tex.	Dec. 20, 2024

C. Lead and Backup Counsel (37 C.F.R. §42.8(b)(3))

Amazon provides the following designation of counsel:

Lead Counsel	Backup Counsel
Colin B. Heideman (Reg. No. 61,513) 2cbh@knobbe.com BoxSEAZN2L2102LP2@knobbe.com <u>Postal and Hand-Delivery Address:</u> Knobbe, Martens, Olson & Bear, LLP 925 4th Ave., Ste. 2500 Seattle, WA 98104	Joseph R. Re (Reg. No. 31,291) 2jrr@knobbe.com Jeremy A. Anapol (Reg. No. 75,686) 2jaa@knobbe.com <u>Postal and Hand-Delivery Address:</u> Knobbe, Martens, Olson & Bear, LLP 2040 Main Street, 14th Floor

Telephone: (206) 405-2000 Facsimile: (206) 405-2001	Irvine, CA 92614 Telephone: (949) 760-0404 Facsimile: (949) 760-9502 Nathan D. Reeves (Reg. No. 77,806) 2ndr@knobbe.com Logan P. Young (Reg. No. 79,294) 2lpy@knobbe.com <u>Postal and Hand-Delivery Address:</u> Knobbe, Martens, Olson & Bear, LLP 925 4th Ave., Ste. 2500 Seattle, WA 98104 Telephone: (206) 405-2000 Facsimile: (206) 405-2001
--	--

Pursuant to 37 C.F.R. §42.10(b), a Power of Attorney accompanies this Petition. The above identified lead and backup counsel are registered practitioners associated with Customer No. 20,995 listed in that Power of Attorney.

D. Service Information (37 C.F.R. §42.8(b)(4))

Service information for counsel is provided above. Amazon consents to service by email at: BoxSEAZN2L2102LP2@knobbe.com.

E. Payment of Fees (37 C.F.R. §42.103)

The fee set forth in 37 C.F.R. §42.15(a) for this petition has been paid. Any additional fees due in connection with this petition may be charged to Deposit Account 11-1410.

F. Grounds for Standing (37 C.F.R. §42.104(a))

Amazon certifies that the '058 patent is available for IPR and that Amazon is not barred or estopped from requesting IPR. This petition is being filed within one year of service of the Complaint in the Litigation.

CONCLUSION

Petitioner respectfully requests cancellation of the challenged claims.

Respectfully submitted,

KNOBBE, MARTENS, OLSON & BEAR, LLP

January 30, 2025

/Colin B. Heideman/

Colin B. Heideman (Reg. No. 61,513)

Joseph R. Re (Reg. No. 31,291)

Jeremy A. Anapol (Reg. No. 75,686)

Nathan D. Reeves (Reg. No. 77,806)

Logan P. Young (Reg. No. 79,294)

Counsel for Petitioner,
AMAZON.COM, INC.

CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. §42.24(d), the undersigned certifies that foregoing **PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 7,784,058**, exclusive of the parts exempted as provided in 37 C.F.R. §42.24(a), contains 13,977 words and therefore complies with the type-volume limitations of 37 C.F.R. §42.24(a).

Respectfully submitted,

KNOBBE, MARTENS, OLSON & BEAR, LLP

January 30, 2025

/Colin B. Heideman/

Colin B. Heideman (Reg. No. 61,513)

Joseph R. Re (Reg. No. 31,291)

Jeremy A. Anapol (Reg. No. 75,686)

Nathan D. Reeves (Reg. No. 77,806)

Logan P. Young (Reg. No. 79,294)

Counsel for Petitioner,
AMAZON.COM, INC.

CERTIFICATE OF SERVICE

I hereby certify that a true and correct copy of the foregoing **PETITION FOR INTER PARTES REVIEW OF CLAIMS 1-16 and 18 OF U.S. PATENT NO. 7,784,058** and **EXHIBITS 1001-1032** are being served on January 30, 2025, via Federal Express overnight mail on counsel of record for U.S. Patent No. 7,784,058 as addressed below:

Allen, Dyer, Doppelt + Gilchrist, PA
1135 East State Road 434, Suite 3001
Winter Springs, FL 32708

A courtesy copy is also being served on counsel for the patent holder in the co-pending litigation *VirtaMove Corp. v. Amazon.com, Inc.*, No. 7:24-cv-00030 (W.D. Tex.):

Reza Mirzaie
rmirzaie@raklaw.com
Marc A. Fenster
mfenster@raklaw.com
Neil A. Rubin
nrubin@raklaw.com
Amy E. Hayden
ahayden@raklaw.com
Christian W. Conkle
cconkle@raklaw.com
Jonathan Ma
jma@raklaw.com

Qi (Peter) Tong
ptong@raklaw.com

RUSS AUGUST & KABAT
4925 Greenville Ave, Suite 200
Dallas, TX 75206

RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025

KNOBBE, MARTENS, OLSON & BEAR, LLP

January 30, 2025

/Colin B. Heideman/

Colin B. Heideman (Reg. No. 61,513)

Joseph R. Re (Reg. No. 31,291)

Jeremy A. Anapol (Reg. No. 75,686)

Nathan D. Reeves (Reg. No. 77,806)

Logan P. Young (Reg. No. 79,294)

Counsel for Petitioner,
AMAZON.COM, INC.