UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

DOCKER, INC.,

Petitioners,

v.

INTELLECTUAL VENTURES II LLC,

Patent Owner.

Case No. IPR2025-00840
U.S. Patent No. 8,332,844

PATENT OWNER PRELIMINARY RESPONSE

TABLE OF CONTENTS

PATENT OWNER'S EXHIBIT LIST

Ex. 2001	Petition in IPR2025-00931 filed by American Airlines and Southwest Airlines
Ex. 2002	Reexamination No. 90/019,938 filed by Unified Patents
Ex. 2003	Invalidity Contentions of American Airlines in District Court
Ex. 2004	S. Munoz, *"The History of Docker's Climb In The Container Management Market,"* published on TechTarget.com, June 25, 2019
Ex. 2005	J. Kronchmaiski, *Developing with Docker*, published November, 2016 by Packet Publishing.
Ex. 2006	A. Shaw, "Docker, Container Images and Standards," published on Medium.com, August 1, 2016
Ex. 2007	Posting on Docker.com by Docket CEO Scott Johnston dated February 1, 2022
Ex. 2008	Order Granting Reexamination Request in case no. 90/019,938 filed by Unified Patents
Ex. 2009	Defendant Southwest Airlines Co.'s Preliminary Invalidity Contentions in *Intellectual Ventures I LLC et al. v. Southwest Airlines*, Co., 7:24-cv-277 (WDTX)
Ex. 2010	Declaration of Erez Zadok in IPR2025-00931
Ex. 2011	Abstract of Title of '844 Patent
Ex. 2012	Evidence of Use of Docker By American Airlines
Ex. 2013	CV of Dr. Richard Newman
Ex. 2014	Declaration of Dr. Richard Newman

I. INTRODUCTION

Institution should be denied. Petitioner has not carried its burden under 35 U.S.C. § 314(a) to demonstrate a reasonable likelihood of prevailing on any challenged claim of U.S. Patent No. 8,332,844 ("the '844 patent"). Each of the four asserted grounds relies on prior art that fails to disclose, teach, or suggest the '844 patent's core innovations: a demand-driven cache expressly configured to store root-image blocks previously accessed by a compute node, and a compute-node-to-compute-node "index once/share" model that allows later nodes to reuse the work of an earlier node.

Ground 1, based on Menage, fails because Menage never discloses a root-image block cache. Its only concrete caching disclosure is a usage-map metadata cache, and its generic aside that shared storage "may be cached" fails to teach all of the limitations that the claims require. Petitioner's fallback inherency argument is legally insufficient and factually unsupported.

Ground 2, which combines Menage with Murphy, fares no better. Murphy describes a pre-boot, one-to-one client↔server repair loop in which each client indexes its own files and receives replacement files back from the server. Nothing in Murphy discloses, or even suggests, one compute node indexing a shared root image and providing those indexing results to another compute node. The proposed combination of Menage and Murphy would require discarding the principles of

operation of each reference, and Petitioner offers no credible rationale or reasonable expectation of success.

Ground 3, based on Birse and Rothman, fails because neither reference teaches the claimed cache configured to store blocks of a root image previously accessed by a compute node. Rothman describes pre-boot, network-edge caching of entire boot images at routers or switches, while Birse describes per-client shadow volumes that incorporate unchanged content. Neither reference discloses the claimed cross-node, demand-driven caching keyed to prior access, and combining them would require substantial redesign.

Ground 4, based on Birse and Murphy, is deficient for the same reasons. Birse has no indexing concept at all, and Murphy's "index" is a per-client pre-boot file listing used by a server to repair that same client. Nothing in either reference teaches the claimed sharing of indexing results of a shared root image from one computer node to another, nor would a skilled artisan have been motivated to retrofit them to do so.

Because the Petition fails to demonstrate that any challenged claim would have been obvious in light of the asserted references, and because the dependent claims add further narrowing features that are nowhere addressed, institution should be denied.

II. SUMMARY OF THE '844 PATENT

A. Background

The '844 Patent is directed to "clustered computing" where, typically hundreds of "compute nodes may be clustered together to share the load of serving a high-traffic website." EX1001, 1:33-38. Such systems typically "centralize data management" using "a globally accessible file system." *Id*., 1:38-40. In such systems, storage of "a boot image" containing the kernel and, in some cases, application software to be run on the compute node, was required for each compute node in the cluster. *Id*., 1:45-49.

Low cluster "bring-up time" is a concern in clustered computing. *Id*., 1:49-50. Typically, the cluster stores a master boot image from which it may "either pre-create clones" or create them "on the fly" which "involves copying the entire contents of the master image, which are typically in the range of 5-15 GB" resulting in "a large bring-up time." *Id*., 1:50-58. "Pre-creating a boot image for each server" reduces cluster bring-up time but results "in wasted disk space." *Id*., 1:61-63. Both methods suffer from the problem that updating "the boot image(s) for the cluster is cumbersome" because it involves "updating a number of copies of the boot image." *Id*., 1:64-67.

The '844 Patent also observed that, once compute nodes have booted, they often engage in redundant activities especially in cases where the cluster includes

multiple compute nodes "each running the same operating system and using substantially similar hardware." *Id*., 2:1-5. The '844 Patent also observed that such "compute nodes will generally need to access much of the same data (e.g., drivers, library files, etc.)," and while each of the compute nodes will index their file systems, the index results will only vary slightly. *Id*., 2:5-11.

The '844 Patent explained that in a "branching store file system, a read-only base image (or "root" image) of the application environment is created" which "is accessible by all compute nodes in the cluster" and "[c]hanges made by a compute node to the root image are stored in a 'leaf' image unique to that compute node." *Id*., 2:13-18. A filter operating between the compute nodes and the file systems "merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node," thereby permitting the computer to run "its own unique and cohesive instance of the application environment." *Id*., 2:18-24. The '844 Patent observed that, while this approach allows for creation of boot images on the fly with less bring-up time, "a separate version of the system must be created for each unique operating system because data is stored at the file system level (i.e., on a "per file basis"). Thus, migrating a computing cluster from one operating system to another is much more complicated than simply installing a new root image containing the new OS." *Id*., 2:24-31.

B. Summary And Advantages Of Disclosed Embodiments

The '844 Patent addressed the shortcomings associated with creating and storing a unique operating system for each compute node at the file system level through the use of "root image caching and indexing for block-level distributed application management." *Id*., 2:35-37 (In the disclosed system, blocks of a root image are stored on a first storage unit and blocks of leaf images are stored on respective second storage units (e.g., one for each compute node). *Id*., 2:39-40. The leaf images are reserved for "additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image." *Id*., 2:40-43. In the disclosed system, "blocks of the root image that have been accessed by at least one compute node" are cached to accelerate access time, and "indexing results pertaining to the root image" generated from one compute node are provided to the other compute nodes" in order to minimize the duplication of activities across compute nodes. *Id*., 2:43-48.

The disclosed system provided a combination of at least *five* important advantages over prior systems.

First, the disclosed system was "operating system-independent." *Id*., 2:50-51.

Second, the system provided for the "creation of boot images on the fly without significantly diminishing bring-up time." *Id*., 2:50-56. In the disclosed

system, "creating a new boot image does not require copying the contents of the root image" but instead involves only registering a new union block device (or UBD) with the system, which occurs very quickly. *Id*., 2:56-59.

Third, in the disclosed system, "[b]ring up time, and access time in general" are "further improved by caching commonly accessed portions of the root image." *Id*., 2:58-61.

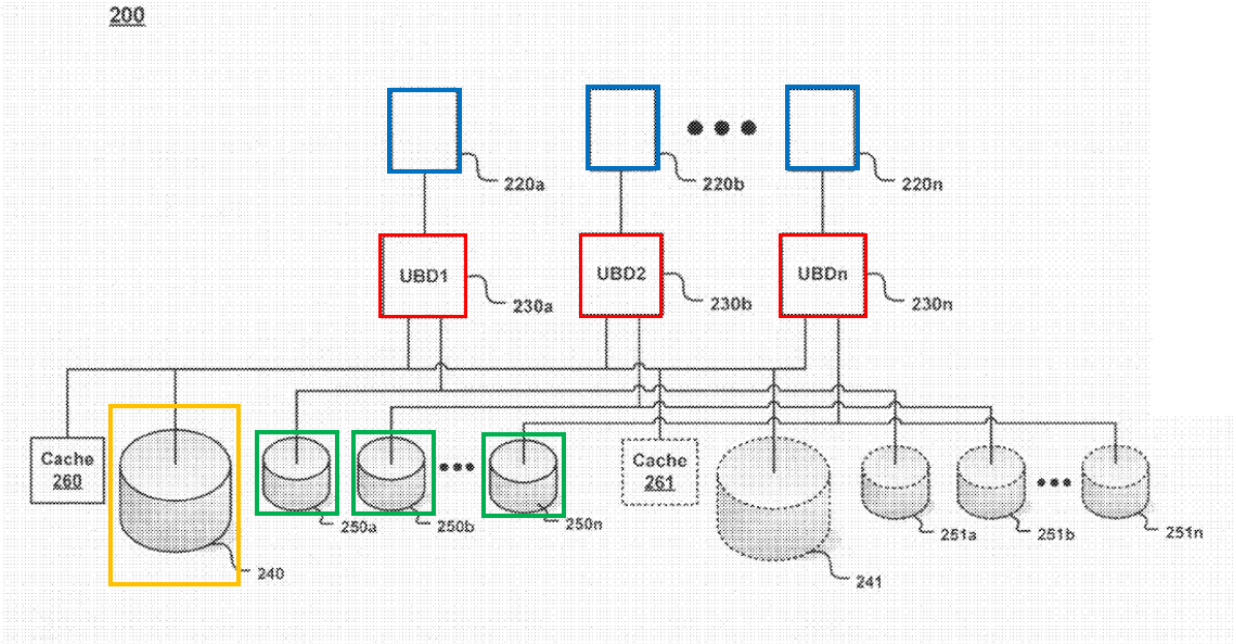
Fourth, in the disclosed system, "updating the boot image for the entire cluster simply involves updating the root image." *Id*., 2:61-63.

Fifth, "because of the commonality of the root image and the fact that its contents are not directly changed, certain operations performed on the root image (e.g., indexing) only need to be performed once by one compute node" and "the results of that operation can be shared with the other compute nodes in the cluster, thus saving the other compute nodes valuable time and resources." *Id*., 2:63-3:2.

C. Detailed Description

Figure 2 of the '844 Patent (reproduced below) depicts an exemplary system "for root image caching and indexing in a block-level distributed application environment." *Id*., 5:12-14. Compute nodes 220*a-n* [blue boxes] are coupled to first storage unit 240 [orange box]. *Id*., 5:19-20. In addition, each of the compute nodes 220*a-n* [blue boxes] is coupled to "a corresponding second storage unit 250*a-*

n [green boxes] though a corresponding union block device (UBD) 230*a-n* [red boxes].” *Id*., 5:20-22.



Using the techniques explained next, the system results in it appearing to each of the compute nodes 220 *a-n* “that they have access to their own version of a distributed application environment. However, a separate and complete boot image is not created and stored for each compute node 220 *a-n*.” *Id*., 5:22-26.

Storage unit 240 [orange box] stores blocks of a root image of an application environment that are “initially common to the compute nodes 220 *a-n*. The root image is not changed by compute nodes 220 *a-n*” (e.g., the compute nodes 220 *a-n* may have read-only access to the first storage unit 240.) *Id*., 5:27-32. “[E]ach compute node 220*a-n* [blue boxes] has a corresponding second storage unit 250 *a-n* for storing a leaf image [green boxes].” *Id*., 5:33-34. “The leaf image may contain

blocks of new data, blocks of changed data, or other blocks of data unique to the individual compute node. The leaf image may also contain a block modification log" that "will describe the changes made by the respective compute node 220 *a-n* [blue boxes] to its instance of the application environment." *Id*., 5:40-45. "Thus, when a compute node (e.g., node 220 *a*) makes changes involving the root image, modifications are made to that compute node's leaf image (e.g., leaf image stored on second storage device 250 *a*). With respect to changes to the root image, only the specific blocks that are changed are stored in the leaf image." *Id*., 5:45-51.

"A compute node 220*a-n* [blue boxes] mounts its instantiation of the application environment via its respective UBD [or union block device] 230 *a-n* [red boxes]." *Id*., 5:59-60. "UBDs 230 *a-n* are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 220 *a-n*. ... Because UBDs 230 *a-n* operate *below the file system, they are concerned merely with the blocks of data themselves, rather than files they form*. As a result, system 200 is completely file system, and thus operating system, independent." *Id*., 5:61-6:3 (emphasis added). "UBDs 230*a-n* [red boxes] determine what leaf image (from the appropriate second storage unit 250 [green boxes]) is needed for portions of the application environment that their respective compute nodes 220*a-n* [blue boxes] have changed. UBDs 230*a-n* also locate the portions of

the application environment that are not changed by their respective compute nodes 220 *a-n*. These portions may reside in the root image." *Id*., 6:3-10.

"UBDs 230*a-n* create a new leaf image on a respective second storage unit 250*a-n* when their respective compute nodes 220*a-n* make changes to their instantiations of the application environment." *Id*., 6:13-16. "UBDs 230 *a-n* may also modify the leaf image in response to their respective compute node's access to its instance of the application environment. For example, upon receiving a write request from their respective compute nodes for a sector X, the UBDs 230 *a-n* will create an appropriate persistent mapping for sector X and then write sector X onto their respective second storage units 250 *a-n*, where sector X can then be modified." *Id*., 6:20-26.

This disclosed system also "provides for caching of portions of the root image. Because several compute nodes in a cluster may often access the same data (e.g., same drivers, same library files, etc.) on the root image, tremendous speed improvements can be realized by caching such data in cache 260." *Id*., 6:38-43. In "a first boot, the cache is empty. The first compute node to boot (e.g., compute node 220 *a*) will therefore load its data directly from the pertinent disks (e.g., first storage unit 240 and second storage unit 250*a*). While the first compute node is loading its data, its reads are intercepted and cached on a per-block basis in cache 260. Thereafter, when a second compute node (e.g., compute node 220*b*)

goes to boot, any data required by the second compute node that was previously accessed by the first compute node can be served out of the cache 260 rather than the first storage unit." *Id*., 6:53-63. "As new data is accessed on the root image that is not currently stored in the cache 260, the cache 260 may thereafter be updated with the new data." *Id*., 7:6-8.

The disclosed system further "provides for indexing the root image." *Id*., 7:18-18. "In a traditional clustered computing situation, each compute node would independently index its file system. To the extent that each compute node in a cluster has similar operating environments to the others, indexing performed on common data is therefore redundant." *Id*., 7:21-25. Since, in the disclosed system, "the data stored in the root image is by definition common to all the compute nodes 220*a*-*n*, one compute node (e.g., compute node 220*a*) can index the contents of the root image and then provide the indexing results to the other compute nodes (e.g., compute nodes 220 *b*-*n*). Thereafter, the compute nodes 220 *a*-*n* need only be concerned with indexing their respective leaf image, which are relatively small compared to the root image." *Id*., 7:26-34.

III. CLAIM CONSTRUCTION

Consistent with Patent Owner's positions in the related district court litigations, the Board should apply plain and ordinary meaning to all challenged claim terms under the Phillips standard. The governing regulation provides that, in

inter partes review, "a claim of a patent … shall be construed using the same claim construction standard that would be used to construe the claim in a civil action under 35 U.S.C. 282(b)," i.e., the standard articulated in *Phillips v. AWH Corp*., 415 F.3d 1303 (Fed. Cir. 2005) (en banc). See 37 C.F.R. § 42.100(b). Under Phillips, claim terms are given their ordinary and customary meaning as understood by a person of ordinary skill in the art at the time of the invention, in the context of the intrinsic record. 415 F.3d at 1312–14. Extrinsic evidence "is less significant than the intrinsic record" and may not contradict it. Id. at 1317–19; see also *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582–84 (Fed. Cir. 1996).

Applying these principles, the terms used in the challenged claims (including, for example, "root image," "leaf image," "cache configured to cache blocks of said root image," and "indexing results") should be afforded their plain and ordinary meaning in view of the specification and prosecution history. To the extent Petitioner advances extra-textual constructions that broaden or narrow the claims beyond what a skilled artisan would understand from the intrinsic record, those proposals should be rejected as inconsistent with Phillips and its progeny. See *Phillips*, 415 F.3d at 1312–14; *Vitronics*, 90 F.3d at 1582–84.

Accordingly, and in alignment with Patent Owner's positions in the parallel district court actions, the Board should resolve this IPR under the plain-and-

ordinary-meaning standard without adopting any special constructions beyond what is necessary to address a concrete dispute.

IV. PRIOR ART RELIED ON IN THE PETITION

A. Menage (EX1004)

Menage describes a virtualization approach designed for multiple virtual private servers running on a single physical host. Rather than operating at the block-layer, Menage implements its design at the system-call level, where a wrapper (element 111) intercepts file system calls and consults a per-template usage map (element 306). That usage map determines whether a given read should be directed to a shared storage unit (302) containing baseline data or to a private storage unit (304) holding modifications for that virtual process. Writes are always redirected to the private storage unit, and the usage map is updated to reflect the change, while reads are routed to the private storage if the usage map indicates valid data is present, or otherwise to the shared storage. The system also allows templates consisting of multiple private storage sets, with usage maps (306) consulted in priority order to resolve requests.

Where Menage discusses caching, it does not disclose caching of shared-image data at all. Its concrete disclosure is limited to caching a copy of the usage map for faster lookups, i.e., metadata rather than data blocks. The only additional mention is a passing remark that the shared storage units "may be easily cached" in

main memory or memory integrated with the storage device. This statement is purely generic; it does not identify what would be cached, when or how a cache would be populated, how cache hits would be recognized and served, or what eviction rules might apply. Menage therefore does not teach a demand-driven cache of root-image data whose contents are keyed to previous access by a compute node.

B. Murphy (EX1005)

Murphy (EX1005) discloses a pre-boot, server-orchestrated maintenance workflow in which each client gathers information about its own local storage and interacts one-to-one with a server. The boot sequence includes downloading a preloader and scripts from a boot server and executing a maintenance "boot image" before the client's normal OS loads (see Figs. 4A–4B). In this context, Murphy's "file management" program creates an "index" which is simply a directory listing for the client's local files and compares that index to a reference image stored on the server; discrepancies prompt the server to deliver replacement or updated files back to that same client so its installation can be repaired prior to boot. The message flow is explicitly closed-loop: as illustrated in Figure 6, the client sends "Entries to Server" (message 616) and receives "Results" (message 620) from the server, with the server performing the comparison against its image and returning any updates only to the originating client. Murphy's claims mirror that architecture; for example, they recite providing an attribute determination program that prepares "an index of

files present on said client computer," receiving that index at the server, and using it to select instructions to provide updated files from the server to the client before boot. EX1005, 22:4-29. Other claims likewise describe verifying a client's file system against an index file and performing the integrity check on the server or the client, all within the same server↔client model. EX1005, 15:1-15.

C. Birse (EX1006)

Birse describes a net-booted network computer (NC) architecture in which clients boot a shared operating system image from an NC server and then operate with reads and writes redirected over the network by a client-side block device driver. The boot process is centrally administered: the server supplies the client with boot parameters and transfers a boot image; after RAM-boot, the client mounts remote system and application volumes, and administration tasks can replace the shared images on the server so that subsequent boots pick up the new configuration. This model is presented as a remedy for the complexity of maintaining per-machine operating systems and emphasizes centralized image management rather than per-node, block-level storage composition.

At the heart of Birse's runtime organization is a SplitOS with three volumes hosted on the server: a read-only core system volume (922), a read-write user system volume (924), and, for each connected client, a client-specific shadow system volume (930). The shadow "shadows the user system volume 924 by storing

modifications that are made to the user system volume 924," while reads may be served from the core, user, or shadow volumes and writes are directed only to the shadow. EX1006, Col. 14:50-52. The client's block device driver implements this redirection transparently so that, from the user's perspective, modifications appear to be written to the user system volume. This is a read/write-redirection scheme between server-side volumes, not a union device that composes a shared root image with per-node deltas below the file system level.

Birse discloses two distinct shadow embodiments, neither of which fits the negative leaf-image limitation recited in the claims of the '844 Patent. In one embodiment, the shadow is a complete copy of the system volume with any user-modifications incorporated, necessarily including unchanged system data. In the other, the shadow is sparse and "written in band increments" (e.g., 128K bands) tracked by a band table; because storage occurs at band granularity, unchanged blocks within a modified band are still present, defeating block-precise exclusion of unchanged content as claimed in the '844 Patent. EX1006, 15:21-32. These structures are shown and described with Fig. 10 (complete copy) and Fig. 11 (banded sparse shadow).

The read and write flows confirm this. On a write, the client's file system generates a request that the block device driver redirects to the user's shadow volume on the NC server; on a read, the block device driver (or server-side logic) determines

whether the requested portion has been modified, and retrieves from the system volumes if unchanged or from the shadow if changed. EX1006, 11:55-67. Birse also contemplates alternatives in which the server keeps separate full copies of the operating system per user, underscoring that the design is comfortable duplicating unchanged content when administratively convenient. None of these flows discloses a cache of blocks of a shared root image or any union block device that creates a per-node file system by block-level merging of a shared root image with a node's delta (e.g., stored in a leaf image).

Birse's administrative model further highlights its focus: a single server-resident master read-only OS image can be replaced to upgrade the fleet; the system supports net-booting and maintains non-persistent client-specific data separately from persistent user data that is restored at the next login. These teachings support centralized provisioning and per-client state management, not cluster-level block-layer composition, root-block caching across nodes, or index-once/share mechanisms.

D. Rothman (EX1007)

Rothman discloses techniques to improve network-boot efficiency by caching operating-system boot images at network devices—such as switches, routers, bridges, or gateway servers—located near the client. On an initial boot, the client downloads the OS boot image from a PXE boot server; in conjunction with

that transfer, the image is cached at a proximate network device. On subsequent boots, the client obtains the OS boot image from the caching device rather than the boot server, thereby reducing boot latency. This flow is presented in the overview and in the Figure 2 sequence, which shows the first power-on obtaining the image from the boot server and caching it at a network device, followed by later boots retrieving the image directly from the caching device. Rothman implements caching by intercepting PXE/TFTP traffic and copying packets for the boot image (and, optionally, the Network Bootstrap Program (NBP)), reassembling those packets into files, and storing the files on a local store at the network device. In other embodiments, the PXE server pushes directives over an in-band or out-of-band management channel instructing a specific network device to cache and then serve the identified image to a particular client. The claims and message-flow diagrams detail these modes, including intercepting download requests destined for the PXE server, forwarding cached files to the client, and reassembling packet copies into cached files at the device.

Rothman also provides version and validity control for cached images. The PXE server maintains an image checksum table (or a replicated “boot image verification table”) and coordinates updates so that caching devices replace outdated images with valid ones after a checksum verification exchange. When a client requests an image, the caching device can consult its checksum for the cached file,

ask the PXE server to verify it, and either serve the cached file if valid or download and cache the updated image if invalid. The reference describes a representative network-device architecture with fast-path packet filtering and slow-path processing that includes a file system, cache-management logic, an "image map" of cached files, and a local non-volatile store. The image map is updated when files are cached; filtering logic and services (e.g., M/TFTP/TCP) support detecting, caching, and serving images from the device.

Rothman's caching is pre-boot and file-level, carried out at network devices by intercepting or directing PXE/TFTP transfers, maintaining checksums, and pushing or serving whole image files to the same requesting client in subsequent boots. It does not teach a demand-driven cache of root-image data keyed to prior access by a compute node.

V. GROUND 1 SHOULD BE DENIED: MENAGE DOES NOT TEACH THE CLAIMED ROOT-IMAGE CACHE, AND INHERENCY CANNOT SUPPLY IT

In Ground 1, Petitioner has not shown that Menage discloses the functionally limiting cache recited in Claims 1-14. The '844 patent requires a cache of root-image data blocks keyed to prior access by a compute node. Menage's concrete disclosure is a cached usage map, which is metadata for bookkeeping, and its lone statement that shared storage "may be cached" is a generic, optional aside that lacks any object, trigger, hit handling, or eviction semantics. Nor can inherency bridge this gap:

Menage can be practiced exactly as disclosed without any prior-access root-image block cache, and ordinary OS page caching is, at most, an implementation choice that is transient and per host, not the claimed cache which stores blocks of a shared root image previously accessed by a compute node. Under the Federal Circuit's standards for inherency in obviousness, the limitation must be necessarily present, not merely possible. See, e.g., *Par Pharm., Inc. v. TWi Pharm., Inc.*, 773 F.3d 1186, 1194–95 (Fed. Cir. 2014); *Southwire Co. v. Cerro Wire LLC*, 870 F.3d 1306, 1310–11 (Fed. Cir. 2017), and *Millennium Pharms., Inc. v. Sandoz, Inc.*, 862 F.3d 1356, 1367 (Fed. Cir. 2017).

On this record, Petitioner has not met that burden, so Ground 1 fails to establish a reasonable likelihood of prevailing under § 103.

A. Menage's Passing Reference to "Caching" Usage Maps Is Not the Claimed Root-Image Block Cache and Cannot Meet the '844 Patent's Specific Cache Requirements

Petitioner's reliance on Menage's single sentence that "the shared storage units 302 may be easily cached for high speed access in the computer memory 102 or in a memory integrated with the storage device 204" (EX1004, 10:4-6) does not meet the '844 patent's functionally limiting cache requirement which stores blocks of a shared root image previously accessed by a compute node, and, critically, is different in kind—not degree. Claim 1 of the '844 patent requires "a cache configured to cache blocks of said root image previously accessed by at least one of

said compute nodes." The claim language is implemented in a demand-driven manner, "caching blocks of the root image that have been accessed by a compute node," These features are all spelled out in the claims and flowcharts and are tied to the root image blocks actually read by one node and then subsequently utilized again.

By contrast, Menage's "may be cached" remark is a generic, optional performance aside that never identifies what data is cached, when it is cached, or how it is populated, served, or evicted. Where Menage does provide concrete caching detail, it only describes caching a copy of the usage map—bookkeeping metadata that indicates which private units contain valid data—to provide faster lookups for its system-call framework. EX1004, 8:54-57. Menage does not disclose a cache of shared-image data blocks, much less a cache whose contents are keyed to prior access by another compute node. The relevant portions of Menage expressly teach: (i) a system-call wrapper intercepts reads/writes and consults/updates a usage map to choose between shared versus private storage; (ii) "a copy of the usage map 306 is cached … to provide high-speed access to the usage map"; and (iii) only in passing, that shared storage "may be cached," without elaboration or any instruction about demand-driven population.

As Dr. Newman explains:

> Technically, a usage map cache and a root-image data-block cache serve different purposes, operate over different objects, and exhibit different behaviors:

> Object of caching: Menage's cacheable object is the usage map (metadata); the '844 cache is the content itself—"blocks of the root image."
>
> Population trigger: In the '844 system, population is demand-driven by reads—blocks are cached because a node accessed them (step 322) and the cache is updated "based on the read request" (step 345). Menage's remarks do not teach demand-driven population of shared-image data blocks keyed to prior access.
>
> Serve-on-hit semantics: The '844 flow requires serving requested data from the cache when present (step 328). Menage's metadata cache can speed up decisions (which unit to read) but does not itself serve the data content.
>
> EX2014 73-76.

Therefore, Menage's concrete caching discussion is about caching a copy of the usage map—the metadata that its system-call framework consults to decide whether a given access should go to a shared storage unit or a private unit. In other words, Menage accelerates lookups into its bookkeeping structure (the usage map 306) that indicates which private units contain valid, overridden data; the map is consulted and updated by a wrapper that intercepts reads and writes. By contrast, Menage's passing remark that shared storage units "may be cached" supplies no specifics: it does not identify what data granularity is cached (block, page, file, entire storage), when population occurs (on read demand, prefetch, or lazily), how hits are recognized and served, or which eviction policy governs the cache.

Accordingly, Petitioners reliance on Menage's isolated generic statement that shared storage "may be cached" cannot substitute for the '844 patent's specific and functionally limiting cache requirements. Menage's actual disclosure centers on caching a usage map, which is metadata used to resolve read and write routing, not on demand-driven caching of root-image data blocks. The '844 patent, by contrast, requires a cache expressly tied to prior node access of root-image blocks, populated and updated in response to reads. Because Menage never discloses or suggests these critical, explicitly claimed features, its generic aside about caching is qualitatively different from the claimed root-image block cache, and thus does not anticipate or render obvious the challenged claims.

B. Petitioner Cannot Rely on Inherency: Menage Does Not Necessarily Disclose the Claimed Root-Image Cache and Fails the Federal Circuit's High Standard

Petitioner cannot fill the cache-limitation gap with inherency. Under controlling law, inherency "must be carefully circumscribed in the context of obviousness," and "the mere fact that a certain thing may result from a given set of circumstances is not sufficient to establish inherency." *Par Pharm.,*, 773 F.3d 1186, 1194–95. The Federal Circuit has "emphasized that the limitation at issue necessarily must be present in order to be inherently disclosed by the reference," *Southwire Co.*, 870 F.3d 1306, 1310–11, and has reiterated that a party must meet a "high standard" to rely on inherency for obviousness. *Millennium Pharms.,*, 862 F.3d 1356, 1367.

Those standards are not met here because Menage can be—and, as disclosed, actually is—practiced entirely without caching root image blocks because they have been accessed.

Menage resolves reads and writes by intercepting system calls and consulting a usage map to decide between shared and private units; its architecture neither requires nor even describes a prior access conditioned data block cache. The core mechanisms are the system call wrapper, the private/shared unit scheme, and the usage map bookkeeping that indicates whether a private unit contains valid data. Those features are laid out across Menage's description of the "system call wrapper 111," the write to private/read from private if valid flow, and the usage map modules (e.g., Figs. 4–6), none of which depend on or necessitate a cross workload cache keyed to "previous access." EX1004, 9:1-27. In fact, where Menage *does* provide concrete "caching" detail, it is only to cache a copy of the usage map to speed metadata lookups—explicitly a shortcut for map access—not a cache of previously accessed *shared-image data blocks*. The only mention of caching data in Menage is a single, generic sentence that "the shared storage units 302 may be easily cached for high speed access" in memory (EX1004, 2-7), which says nothing about *what* is cached, *when* it is cached, or *how* entries are inserted, served, or evicted—much less that the cache reflects "previous access" by a compute node. Because Menage's disclosed system works by usage-map–driven selection of shared vs. private units at

the system-call layer, it can be fully implemented with no prior-access data-block cache at all. It follows that the cache of previously accessed data blocks in the '844 Patent is not the "necessary" or "inevitable" result of Menage; inherency therefore does not apply. *Par Pharm.*, 773 F.3d at 1194–95; *Southwire*, 870 F.3d at 1310–11; *Millennium*, 862 F.3d at 1367.

To the extent Petitioner suggests that ordinary OS page caching would make Menage "inherently" practice the limitation, that argument still fails both legally and factually. Legally, a feature that *might* occur in some implementations is insufficient; inherency demands that the limitation be necessarily present in the reference. *Par Pharm.*, 773 F.3d at 1194–95; *Southwire*, 870 F.3d at 1310–11. Factually, Menage is a single host VPS system; any OS page cache is transient and per host, not a cross node cache that stores root image data blocks previously accessed by one node to accelerate another. Menage's affirmative "caching" disclosure is the cached usage map; its "may be cached" sentence does not require demand driven population and nothing ties it to "previously accessed" blocks of a root image across independent compute nodes. By contrast, the '844 patent repeatedly teaches the cross node behavior: during first boot the cache is empty; the first node's reads are intercepted and cached on a per block basis; a second node is then served blocks "previously accessed" by the first; and cache entries are admitted/evicted according to recency/threshold logic (Fig. 4).

Therefore, Petitioner cannot meet the "high standard" for inherency in obviousness because Menage neither necessarily contains nor motivates the '844 patent's specific, demand driven cache that stores previously accessed root image blocks, that serves later reads from cache.

C. Claim 7 and the Dependent Claims Are Patentable For The Same Reasons As Claim 1

The Petition's challenge to independent claim 7 in Ground 1 fails for the same reasons as claim 1. It recites the same operative limitations in method form: receiving and using blocks of a shared root image; maintaining per-node leaf images that contain only new or changed blocks and exclude unchanged root blocks; and employing a cache configured to cache blocks of the root image previously accessed so later accesses benefit from prior reads. Petitioner's mapping for claim 7 is the same as for claim 1 and relies on the same Menage disclosure. As explained for claim 1, Menage does not teach the claimed cache, which must be keyed to prior access to a shared root image in a multi-node root/leaf architecture, nor does it satisfy the negative limitation on the per-node leaf images. Menage's discussion of caching concerns metadata or generic performance remarks rather than a cache "of the root image," and its system-call interception model does not disclose the claimed block-level merging context. For the reasons already set out for claim 1, the record does not show that Menage discloses or renders obvious the method steps of claim 7 that

implement the same architecture and caching behavior. Ground 1 therefore fails as to claim 7.

The same conclusion follows for the dependent claims. Each dependent claim incorporates the full limitations of its base claim, so Petitioner's failure to establish the base limitations is dispositive. At a minimum, claims depending from claim 1 (claims 2–6) and those depending from claim 7 (claims 8–13) are patentable by virtue of that dependency. In addition, several dependent claims add further narrowing features that Menage does not teach, including providing requested portions from the cache when present (claim 8), updating the cache based on read requests (claim 9), caching uncached portions and removing least-recently-accessed data when a threshold would be exceeded (claim 10), merging blocks of the root and leaf images to create cohesive environments (claim 11), performing the merging at an operational level between the file systems and the storage units and the cache memory (claim 12), and requiring a read-only root storage unit (claim 13). Petitioner does not substantively map these added limitations to Menage, and they remain missing for the same reasons the core cache and root/leaf requirements are missing. On this record, Petition has failed to establish a likelihood of success in Ground 1 for claim 7 and all claims depending from claims 1 or 7.

VI. GROUND 2 FAILS; MENAGE AND MURPHY DO NOT DISCLOSE COMPUTE-NODE-TO-COMPUTE-NODE ROOT-IMAGE INDEXING AND PROVIDE NO MOTIVATION OR REASONABLE EXPECTATION OF SUCCESS TO COMBINE

In Ground 2, Petitioner asserts that claims 14–27 would have been obvious in view of Menage and Murphy. That contention fails because the references do not disclose the core functionality that claim 14 expressly requires, and the proposed combination lacks any articulated rationale or reasonable expectation of success. The challenged claims turn on a system in which one compute node indexes a shared root image and provides those results to another compute node so later nodes can avoid duplicating work. Section A explains why Murphy's client–server integrity-check model, which operates in a closed loop between a single client and a server, does not disclose or suggest compute-node-to-compute-node indexing and sharing of a common root image, and why Menage adds nothing to cure this deficiency. Section B addresses motivation to combine, showing that the two references address different problems at different layers, that the proposed graft would upend each reference's principle of operation, and that a person of ordinary skill would not reasonably expect success in fusing them into the root/leaf, index-once-share-across-nodes architecture of the '844 patent.

A. Murphy does not teach claim 14's "compute-node-to-compute-node indexing of a shared root image—and combining it with Menage does not sure the defect.

Claim 14 of the '844 patent recites, in pertinent part, a system in which "at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes." The specification makes the same point in multiple places and shows exactly how it works: one node indexes the cluster-wide root image, stores or shares those indexing results (e.g., on a shared storage unit), and the other nodes consume those results so they need only index their leaf images. Figure 5 operationalizes this flow (index root at step 520; store results at step 530; provide results to another compute node at step 540; later index/re-index only the leaf at steps 550/560), and the narrative around Figure 2 explains that this cross-node sharing is enabled precisely because the root image is common for all nodes.

Murphy discloses something fundamentally different. Its "index" is built per client in a pre-boot maintenance context, and it is used to verify and repair that same client's local files. The server instructs each client to prepare its own directory/index information for that client's storage device and to send those entries back to the server; the server then compares the client's index against an image on the server and returns "results" to that same client (e.g., replacement files or a status code). Murphy's own description of Figure 6 tracks this one-to-one flow: message 616

carries the client's index to the server, the server performs the comparison, and message 620 returns the results to that client. EX1005, 15:25-57. There is no disclosure of forwarding a client's index to any other client, no suggestion that a client indexes a shared root image for the benefit of others, and no mechanism by which a second client could rely on a first client's index to avoid performing its own indexing.

Murphy's claims and narrative confirm the point. The "attribute determination" or file-integrity routines are expressly configured to prepare "an index of files present on said client computer," transmit that index to the server, and then obtain updated files from the server—again, a closed loop between server and the originating client. EX1005, 16:49-17-8. That is the opposite direction of the '844 patent's compute-node → compute-node sharing of root-image indexing results.

Murphy's data model is also incompatible with the '844 claim language. Murphy's "image file" on the server is merely a reference copy used for file-level repair; each client still indexes and repairs its own local operating-system image. There is no root/leaf arrangement in which a single root image is mounted and shared across many compute nodes, with per-node leaf deltas layered on top with only the leaf needing subsequent indexing. By contrast, the '844 patent's architecture (Figure 2 and accompanying text) requires exactly that: a cluster-wide root image stored on a first unit, per-node leaf images on second units, and union-block devices

that merge them and enable cross-node reuse of the root index. That is why claim 14 (and the associated methods/flowcharts) require one node to index the root image and provide those results to other nodes—the functional payoff is that later nodes need not re-index the shared root at all.

Because Murphy never contemplates (i) a compute node indexing a shared, read-only root image that is common to multiple nodes, (ii) transmitting that index from one node to another node, or (iii) any mechanism by which a second node uses the first node's indexing results to skip its own root-image indexing, it cannot supply the expressly recited limitations of claim 14. The only dataflow Murphy actually teaches is client $A \rightarrow$ server $\rightarrow$client A (Figure 6), not the client $A \rightarrow$ client B (compute-node-to-compute-node) flow expressly required by the '844 patent.

Nor does Menage fill the gap. Menage operates with a per-template usage-map and system-call interception to decide between reading shared versus private storage units; when Menage mentions "caching," it is as a generic optional performance aside, not as a cross-node sharing of indexing results of a root image. Nothing in Menage teaches compute-node-to-compute-node transmission of a root-image index.

B. A POSITA would not have been motivated to combine Menage with Murphy, and any such rationale would be impermissible hindsight.

Obviousness requires "some articulated reasoning with some rational underpinning" for the proposed combination; conclusory assertions or generic appeals to "common sense" are not enough. *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 418 (2007); *In re Magnum Oil Tools Int'l, Ltd.*, 829 F.3d 1364, 1380–81 (Fed. Cir. 2016) (petitioner bears the burden to articulate a reasoned motivation to combine, not ipse dixit); *Metalcraft of Mayville, Inc. v. The Toro Co.*, 848 F.3d 1358, 1366 (Fed. Cir. 2017) (rationale must be supported by a "rational underpinning"); *Arendi S.A.R.L. v. Apple Inc.*, 832 F.3d 1355, 1363 (Fed. Cir. 2016) (cautioning against using "common sense" to supply a missing, non-peripheral limitation).

1. The references solve different problems in different settings, so a POSITA would not look to Murphy to modify Menage.

Menage addresses VPS file system virtualization on a single host, using a system call wrapper and a per template usage map to decide reads/writes between shared units and private units (see Menage, Figs. 4–6 and associated text describing the "system call wrapper 111," "usage map checking module," and "private storage units 304" tied to "shared storage units 302"). The specification even notes the caching aside and contemplates that virtualization might alternatively occur at a lower level such as a virtual disk device, but does not implement a block device

union; the core disclosure is call interception and usage map semantics. By contrast, Murphy is a pre boot, server managed maintenance system where each client builds an index/directory of its own local drive and transmits that index to the server for comparison against a server held image; the server then returns "results" to that same client (see Figure 6: arrow 616 "Entries to Server," arrow 620 "Results," and the accompanying text at col. 15 describing comparing the client's index "to files actually in place" and delivering replacements). The fields and problem statements do not align: Menage virtualizes storage for multi-tenant VPS isolation; Murphy monitors and repairs standalone clients via a server centric workflow. Under the analogous art test, Murphy is not "reasonably pertinent" to Menage's problem because it addresses client maintenance, not template/usage map virtualization or cross node sharing of root image indexing.

2. The proposed modification would change the prior art's principles of operation and render the systems unsatisfactory for their intended purposes.

Murphy's one-to-one client↔server pre-boot flow is explicit and closed-loop: client prepares index → server compares → server returns results to that client (see EX1005 Fig. 6). Reengineering Murphy to support compute node→compute node sharing of root image indexing results (as the '844 claims require) would invert Murphy's server mediated model and graft in a peer to peer distribution layer and a shared, read only root image that Murphy never contemplates; Murphy has every

client individually index and repair its own image. Likewise, pushing Murphy into Menage would require replacing Menage's system call/usage map framework with the '844 patent's block level, union device context that merges a shared root image with per node leaf blocks—an architectural shift Menage does not teach, beyond a passing acknowledgment that virtualization could occur at a lower level (Menage never implements a union block device, nor any cross node indexing or sharing). Federal Circuit law cautions against combinations that would "render the prior art device inoperable for its intended purpose" or alter its "principle of operation." *In re Gordon*, 733 F.2d 900, 902 (Fed. Cir. 1984); *DePuy Spine, Inc. v. Medtronic Sofamor Danek, Inc.*, 567 F.3d 1314, 1327 (Fed. Cir. 2009). That is precisely what happens here: the proposed graft would discard Murphy's central server orchestrated repair paradigm and Menage's usage map/system call design in favor of a block device union with node to node reuse, which neither reference suggests.

3. Even if a POSITA tried, there is no reasonable expectation of success; the stack-level mismatch creates unpredictable engineering hurdles.

Menage's logic operates above the block layer by intercepting system calls and consulting a usage map to decide read from shared vs. read from private (Figs. 4–6), while Murphy's index and repair occurs pre boot, outside the normal OS, exchanging file level indices with a remote server (Fig. 6; col. 15–16). Bridging these would demand: (i) introducing a shared, read only root image accessible across

compute nodes; (ii) implementing a union block device to merge root and leaf below the filesystem; and (iii) devising cross node distribution and consumption of root image indexing results so subsequent nodes can skip root indexing—none of these are taught or suggested in either reference. The law requires that a proposed combination offer a reasonable expectation of success; where, as here, the record shows a layer mismatch (system call metadata vs. pre boot file maintenance) and missing mechanisms (no union block device, no node to node sharing, no root/leaf model), such an expectation is absent.

C. Ground 2 fails to demonstrate a reasonable likelihood of success for Claims 19 and 23 (and their dependents) or the same reasons as Claim 14

Independent claims 19 (method) and 23 (CRM) are patentable over the relied on references for the same reasons as claim 14. Each restates the same operative limitations in method or computer-readable medium form: a compute node indexes a shared, read-only root image; those indexing results are provided to other compute nodes for reuse; subsequent activity focuses on the per-node leaf image rather than re-indexing the shared root image; and the architecture is the same root/leaf system created at the block layer beneath the file system. The Petition's Ground 2 maps Menage and Murphy to claims 19 and 23 the same way it maps them to claim 14. As explained for claim 14, Menage has no indexing at all and says nothing about node-to-node provision of indexing results. Murphy's "index" is a client's own file

list sent to a server in a pre-boot repair workflow, with results returned to the same client, not to a different client. Nothing in Menage and Murphy teaches or suggests the claimed compute-node indexing of a shared root image followed by providing those results to another compute node, nor the leaf-only re-index and merge model the patent discloses. For the reasons already set out for claim 14, Ground 2 does not show that Menage and Murphy discloses or renders obvious the method steps of claim 19 or the instructions of claim 23 that implement the same cross-node indexing and reuse.

The same conclusion follows for the dependent claims. Each dependent claim incorporates the full limitations of its base claim, so Petitioner's failure to establish the base limitations is dispositive. At a minimum, claims depending from claim 19 (for example, claims 20–22) and those depending from claim 23 (for example, claims 24–27) are patentable by virtue of that dependency. Moreover, several dependent claims add narrowing features that the combination likewise does not supply, such as storing the root-image indexing results on shared storage accessible to other compute nodes, re-indexing only the leaf and merging with previously stored root results on re-index, and implementing the claimed functionality in the same block-level root/leaf architecture that excludes unchanged root blocks from the leaf images. Because Menage and Murphy do not teach the cross-node index once/share requirement or the underlying root/leaf union context, the added dependent

limitations remain unmet. On this record, Ground 2 fails to present a reasonable likelihood of success as to independent claims 19 and 23 and all claims depending from them.

VII. GROUND 3 FAILS: BIRSE AND ROTHMAN DO NOT TEACH THE CLAIMED "PREVIOUSLY-ACCESSED" ROOT-IMAGE CACHE, AND THEIR COMBINATION WOULD REQUIRE REDESIGN THAT STILL DOES NOT YIELD THE CLAIMED INVENTION

In Ground 3, Petitioner argues that claims 1–13 would have been obvious in view of Birse and Rothman. That argument is flawed because the references do not disclose the specific cache that the claims require, and combining them would not create it. The challenged claims center on a cache that is populated with blocks of a shared root image accessed by a compute node. Neither Birse nor Rothman teaches this functionality. Rothman describes network devices caching entire boot images in connection with PXE/TFTP traffic, while Birse discusses operating system administration using client-specific shadows, but neither reference addresses demand-driven caching of root-image blocks based on prior node access.

The references also operate in fundamentally different contexts that do not fit together. Rothman's caches work at the network edge in pre-boot scenarios, whereas Birse's framework manages client volumes at runtime through redirection between a shared system volume and client shadows. Integrating the two would require redesigning each system and altering their principles of operation, which the law

makes clear cannot serve as the basis for obviousness. On this record, Ground 3 does not demonstrate that claims 1–13 would have been obvious.

A. Birse and Rothman never teach the claimed "previously-accessed" root-image cache.

Claim 1 requires "a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes." In the operational sequence disclosed in the specification, a first node boots and its reads of the root image populate the cache; when a second node later requests the same data, "that portion" is served directly from the cache; the cache is then updated in response to the read and trimmed by removing the least-recently-accessed data if needed.

Rothman addresses a different problem with a different mechanism. It improves network boot efficiency by caching entire boot images at network devices (switches/routers/gateway servers) along the path between a PXE boot server and clients. EX1007, Abstract. Caching is initiated by packet interception and "message indicia," or by a server directive over a management channel that instructs the network device what to cache and when to push it; version control is maintained via checksum verification and a centrally managed "boot image verification table." EX1007, 13: 44-46. None of this caching is based on, or populated by, a compute node's read access to a shared root image.

Birse, for its part, is about administering the operating system of a net-booted environment—provisioning and maintaining client OS images through a boot server and related components. EX1006, 2:52-3-13. It does not disclose a cache whose contents are populated because a different compute node previously accessed portions of a shared root image, nor does it disclose serving later reads from such a cache or updating that cache based on read requests. As Dr. Newman explains "Birse's figures and description (e.g., the boot-server/NC-client arrangement and OS administration focus) underscore that Birse is an OS management framework, not a demand-driven root-image cache." EX2014 89.

Taken together, the references still miss the claim's functional heart: a cache whose population is conditioned on prior access by a compute node to a shared root image. That is precisely what the '844 patent teaches and it is absent from both Rothman's network-device caching and Birse's OS-administration flow.

B. The references are architecturally incompatible; combining them would change their principle of operation and still would not yield the claimed system.

Rothman is about pre-boot, network device caching of boot images by intercepting PXE/TFTP message flows and storing (and later serving) copies of those images from switches/routers/bridges or gateway servers proximate to clients. Its flow is two phase: on the first boot, the image is cached at the network device; on subsequent boots, the client "obtains the OS boot image from the caching device

(e.g., switch, router, bridge, gateway server)" rather than from the boot server. (FIG. 2 and text.) Rothman details implementations in which the switch runs M/TFTP/TCP services and pushes or serves the cached image when it detects or is directed to satisfy PXE/TFTP requests; it also describes "intelligent" switches that intercept those request messages and deliver the cached image locally. (EX1007, FIGS. 1b–1d; 3b–3d; 4–6).

These two architectures do not fit together. Birse's operating point is run time request resolution with a client specific overlay; Rothman's is pre-boot content distribution from the network edge. Birse depends on correct per-client disambiguation at read time between the shared system volume and that client's shadow volume; it is explicit that "some mechanism is needed to direct reads and writes to the appropriate volume," which Birse implements either in the client block device driver or in centralized server logic. Rothman's caches at switches/routers/gateways have no visibility into Birse's per-client state (i.e., which portions a particular client has modified) and are not in the data path Birse uses to perform that determination. They merely store and serve copies of boot images in response to PXE/TFTP traffic before the operating system mounts Birse's system volumes.

Attempting to "insert" Rothman's network caches into Birse would either (i) divorce the per-client read routing from the only components that know how to do

it (the block device driver and/or server logic), or (ii) force a redesign in which edge devices acquire Birse specific knowledge of each client's shadow usage and participate in the read/write redirection logic. The first option risks serving stale or wrong content by ignoring the client's shadow state, directly undercutting Birse's stability and isolation goals; the second option is nowhere suggested in either reference and would require reimplementing Birse's redirection semantics at unmanaged network devices designed only to cache and serve pre-boot image files. That would render Birse "unsatisfactory for its intended purpose*." In re Gordon*, 733 F.2d 900, 902 (Fed. Cir. 1984).

For example, Birse's system is intended to work by ensuring that (a) writes always land in that client's shadow, and (b) reads are arbitrated at run time between a shared, read-only volume and the client's shadow, so the client always sees its own modifications without corrupting the shared base. Replacing that with Rothman's pre-boot caching at switches/routers would either (i) ignore the client's shadow and serve shared bytes that are wrong for that client, or (ii) require network devices to assume Birse-specific COW knowledge and run-time routing—neither taught nor suggested—which changes how Birse works and risks making it unsatisfactory for its intended purpose.

Legally, this is not a mere "design choice." Under *KSR*, there must be "a reason to combine" that fits the technology, supported by "articulated reasoning with

some rational underpinning." *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006); *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 418 (2007). Here, Petitioner offers no reason why a POSITA would graft a pre-boot, network edge image cache onto a run-time, server-centric split-volume system whose correctness hinges on per-client copy on write semantics. Even if one were to try, the combination would change the principle of operation of Birse (and of Rothman) by relocating Birse's per-client read/write arbitration to devices that neither reference equips to perform it—precisely the kind of major reengineering the Federal Circuit disfavors. See, e.g., *Allied Erecting & Dismantling Co. v. Genesis Attachments, LLC*, 825 F.3d 1373, 1381–82 (Fed. Cir. 2016) (rejecting combinations that require substantial redesign undermining the prior art's functionality); *In re Ratti*, 270 F.2d 810, 813–14 (CCPA 1959) (impermissible when proposed modification "changes the basic principles under which the device was intended to operate").

Nor can "common sense" bridge the gap. The Federal Circuit has cautioned that common sense cannot supply a missing, technologically specific limitation or reallocate core functionality without evidence. *Arendi S.A.R.L. v. Apple Inc.*, 832 F.3d 1355, 1363–64 (Fed. Cir. 2016). On this record, there is no disclosure in Rothman of any run time participation in Birse's read routing, no awareness of client specific overlays, and no suggestion in Birse to push its redirection logic to network switches/routers. Birse's detailed flows (see EX1006, FIGS. 4–5) and SplitOS

depiction (see EX1006, FIG. 9) instead confirm that the client/NC server pair—not a third party switch—performs the copy on write redirection and read arbitration.

Finally, even if a POSITA were to attempt the combination, the result still would not yield the claimed system—which requires, among other things, a cache based on prior access to a shared root image. Rothman's practice of caching and re-serving static boot images at network devices for the same client on subsequent boots is not a Birse compatible, demand driven caching of root image content influenced by earlier accesses in a cluster. Birse, meanwhile, does not re-serve content based on a prior client's reads; its mechanism is per-client isolation through shadows.

C. Claim 7 and the Dependent Claims Are Patentable For The Same Reasons As Claim 1

In connection with ground 3, independent claim 7 stands for the same reasons as claim 1. It recites the same operative limitations in method form, including using a shared root image, maintaining per-node leaf images that include only additional or changed blocks and exclude unchanged root blocks, and employing a cache configured to cache blocks of the root image previously accessed so later accesses benefit from prior reads. Petitioner's Ground 3 maps Brise and Rothman to claim 7 in the same way it maps them to claim 1, but those references still do not teach the claimed cache in the required root/leaf architecture. Brise's split-OS with per-client shadows, routes, reads, and writes based on per-client modification state and does

not disclose a cache of the root image populated by prior access requests. Rothman's pre-boot, network-device caching of boot images is not a cache "of the root image" operating in the storage path alongside per-node leaf images, is not populated by compute-node reads of the shared root, and does not provide the read-driven update behavior the claims require. For the reasons set forth above for claim 1, the Brise/Rothman combination likewise fails to disclose or render obvious the method steps of claim 7 that implement the same architecture and caching behavior. Ground 3 therefore does not demonstrate a reasonable likelihood of prevailing on claim 7.

The same conclusion follows for the dependent claims. Each dependent claim incorporates the full set of limitations from its respective base claim, so Petitioner's failure to show the base limitations are taught or suggested is dispositive. At a minimum, claims depending from claim 1 (claims 2–6) and those depending from claim 7 (claims 8–13) remain patentable by virtue of that dependency. In addition, several dependent claims add further narrowing features that the combination does not supply, including providing requested portions from the cache when present (claim 8), updating the cache based on read requests (claim 9), caching uncached portions and removing least-recently-accessed data when a threshold would be exceeded (claim 10), merging blocks of the root and leaf images to create cohesive environments (claim 11), performing that merging at an operational level between the file systems and the storage units and the cache memory (claim 12), and requiring

a read-only root storage unit (claim 13). Because those added features presuppose the same missing root/leaf cache architecture and read-path behavior, they are not met for the same reasons. On this record, Ground 3 fails to establish a reasonable likelihood of success as to claim 7 and all claims depending from claims 1 or 7.

VIII. GROUND 4 FAILS: BIRSE AND MURPHY DO NOT DISCLOSE COMPUTE-NODE-TO-COMPUTE-NODE ROOT-IMAGE INDEXING, AND THEIR COMBINATION WOULD REQUIRE REDESIGN THAT ALTERS THEIR PRINCIPLES OF OPERATION

In Ground 4, Petitioner contends that claims 14–27 are obvious in view of Birse and Murphy. That theory fails because the two references do not disclose the peer-to-peer indexing and sharing of a common root image required by the claims, and their combination would require a fundamental redesign that alters each reference's principle of operation. The challenged claims call for one compute node to index a shared, read-only root image and then provide those results to another compute node, enabling later-booting nodes to reuse that work rather than re-indexing the same root data. Birse, however, has no concept of indexing at all; its architecture resolves reads and writes at runtime by directing each client between a shared system volume and its own shadow volume. Murphy, by contrast, describes a pre-boot, server-orchestrated integrity check in which each client builds its own index of its local files, sends it to the server, and receives repair results back—never forwarding results to another client or enabling cross-node reuse.

Even if combined, these disparate disclosures do not meet the client-to-client handoff recited in claims 14, 19, and 23. Birse would need to be retrofitted with an indexing stage it does not contemplate, while Murphy would need to be re-engineered from a server-centric one-to-one workflow into a compute-node-to-compute-node model of sharing results. Such a graft would not only depart from the teachings of both references but also undermine their intended purposes: Birse's client-specific shadowing for isolation and Murphy's server-based repair paradigm. The arguments that follow explain why neither Birse nor Murphy discloses the required indexing and sharing, and why the proposed combination represents impermissible hindsight rather than a predictable improvement supported by the record.

A. Neither Birse Nor Murphy Discloses or Suggests Compute-Node-to-Compute-Node Indexing of a Shared Root Image

Claims 14, 19, and 23 require that a compute node index a common root image and then provide those indexing results to at least one other compute node—in practice, this enables later-booting nodes to reuse the first node's work instead of re-indexing the same root data.

Birse contains no indexing—let alone cross-node indexing. Birse's design is copy-on-write over a server-hosted "system volume" with per-client "shadow" volumes. Reads and writes are resolved by a client block-device driver (or server

logic) that decides, for that client, whether to hit the shared system volume or the client's shadow; Birse never teaches any "indexing" operation or any sharing of "indexing results" between clients. (See, e.g., the write-redirection and read-resolution flows and the description of the per-client shadow volume).

Murphy is strictly one-to-one (server↔client) maintenance—not client-to-client sharing of indexing results. Murphy's "index" is a client-specific list used to verify/repair that same client's installation during a pre-boot workflow. The client collects entries and sends them to the server ("Entries to Server" 616); the server compares the entries against a golden image and returns "Results" 620 to that same client (Fig. 6; col. 15). There is no disclosure of forwarding an index prepared by client A to client B, and no suggestion that one client's results spare another client from creating its own index.

Even in combination, Petitioner's theory never gets to the claimed client-to-client handoff. Birse has no indexing at all, and Murphy's "indexing" is a per-client, pre-boot hygiene step whose data flow is client A→server→client A, not compute-node A → compute-node B. Nothing in Birse suggests adding an indexing stage, and nothing in Murphy suggests repurposing a client's index so other clients can reuse it. To satisfy claims 14, 19, and 23, the combination would need a compute node (not the Birse server) to index the shared root, persist those results on shared storage, and provide them to other compute nodes—the exact sequence the '844

patent teaches (Fig. 3B, steps 352–356; Fig. 5, steps 520/530/540). The cited portions of Birse and Murphy do not disclose or suggest any of these client-to-client features.

Nor can Murphy's server simply be relabeled as the "compute node" of claim 14. The claims and the '844 disclosure center the indexing action on a compute node that functions in concert with other compute nodes (e.g., in a cluster) and then provides results to other compute nodes; the figures show results being stored on shared storage and made available to peers, not a server conducting its own analysis and returning repairs to the originating client. (See Fig. 3B, steps 352/354/356; Fig. 5, steps 520/530/540.) Murphy's server-processed workflow is the opposite, both directionally and functionally.

B. Birse and Murphy are architecturally mismatched; the proposed modification would change their principles of operation and would be "unsatisfactory for its intended purpose."

Birse operates by mounting server-side "system volumes" and using a client block-device driver to perform copy-on-write into a *client-specific* shadow volume; reads are resolved at run time by determining whether the requested portion was modified by *that client* (see OS WRITE/READ processing in Figs. 4–5; "shadow system volume 930"; and the boot/mount sequence).

Murphy's core mechanism is different in kind: a *pre-boot* PXE/TFTP workflow where a server pushes a preloader and scripts, the client builds a *local file*

list/index, returns it to the server, and the server repairs/updates that *same client* before chain-booting its local OS (Fig. 4A; Fig. 6; text describing file administration program 412).

Transplanting Murphy's pre-boot, *per-client* file-integrity index into Birse would require (i) having a client, *before* its normal Birse boot/mount path, mount and index Birse's *server-hosted, read-only* OS volume; (ii) redesigning the data flow so that the client then *forwards* that index to *other* clients; and (iii) ensuring coherence with Birse's client-specific shadowing. Neither reference teaches or suggests such a redesign, and the change would undercut Birse's per-client copy-on-write resolution model by introducing a shared, cross-client pre-boot state that the Birse block-device path never consults. That is precisely the sort of alteration that would render the primary reference "unsatisfactory for its intended purpose." *In re Gordon*, 733 F.2d 900, 902 (Fed. Cir. 1984). The Petition also lacks the required "articulated reasoning with some rational underpinning" to make these steps, *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006), as refined by *KSR* (obviousness demands a reason to combine that fits the technology and yields a predictable result). Given the gulf between Birse's run-time, block-device read/write redirection and Murphy's pre-boot per-client maintenance index, there is no persuasive rationale or reasonable expectation of success for the claimed peer-to-peer, root-image indexing/share scheme.

C. Claims 19 and 23 (and their dependents) are patentable for the same reasons as Claim 14

In connection with Ground 4, independent claims 19 (method) and 23 (CRM) stand or fall with claim 14. Each recites the same operative limitations in method or computer-readable-medium form: a compute node indexes a shared, read-only root image and provides those indexing results to another compute node for reuse; subsequent indexing focuses on the per-node leaf rather than re-indexing the shared root; and the functionality operates in the same root/leaf, block-level architecture claimed in 14. Petitioner's Ground 4 maps Birse and Murphy to claims 19 and 23 exactly as it maps them to claim 14. But—as with claim 14—Birse discloses no "indexing" at all, much less node-to-node sharing of root-image indexing results, and Murphy's "index" is a pre-boot, server-mediated file list used to repair the same client, not a compute-node-to-compute-node provision of root-image indexing results that lets a second node avoid duplicative root indexing. For the reasons set forth above for claim 14, the Birse/Murphy combination likewise fails to disclose or render obvious the method steps of claim 19 or the instructions of claim 23 that implement the same index-once/share-across-nodes behavior. Ground 4 therefore does not demonstrate a reasonable likelihood of prevailing on claims 19 and 23.

The same conclusion follows for the dependent claims. Each dependent claim incorporates the full limitations of its base claim, so Petitioner's failure to establish

the base "index-once/share" requirement is dispositive. At a minimum, claims depending from claim 19 (e.g., claims 20–22) and those depending from claim 23 (e.g., claims 24–27) are patentable by virtue of that dependency. Moreover, several dependent claims add narrowing features that the combination still does not supply, including storing the root-image indexing results on shared storage accessible by other nodes, re-indexing only the leaf and merging with previously stored root results on re-index, and performing these steps within the same block-level root/leaf architecture. Because Birse and Murphy do not teach the core cross-node index once/share model or the claimed distribution and reuse of root-image indexing results, the added dependent limitations remain unmet. On this record, Ground 4 fails to demonstrate a reasonable likelihood of success as to independent claims 19 and 23 and all claims depending therefrom.

IX. CONCLUSION

Petitioner has not met its burden to demonstrate a reasonable likelihood of prevailing on any challenged claim. Each asserted ground fails for the same fundamental reason: the cited art does not disclose or suggest the specific root/leaf and cache/index architecture claimed in the '844 patent.

Ground 1 rests on Menage, but Menage never teaches a demand-driven cache of root-image blocks whose contents are based on prior access by a compute node. Its disclosure is limited to caching metadata (a usage map) and a generic aside about

shared storage "being cached," neither of which meets the functionally limiting cache requirement. Petitioner's inherency theory fares no better under controlling Federal Circuit precedent.

Ground 2 combines Menage with Murphy, but Murphy discloses only a pre-boot, server-mediated file-repair workflow in which each client indexes its own files and receives repairs back from the server. That one-to-one client↔server loop is the opposite of the claims' compute-node-to-compute-node index once/share model, and Menage contributes nothing to fill the gap. The combination also lacks any articulated rationale that fits the technology, and would require altering both references' principles of operation.

Ground 3 relies on Birse and Rothman, but neither reference discloses a cache configured to store root-image blocks previously accessed by a compute node. Birse implements per-client shadows that include unchanged content, while Rothman caches whole boot images at network devices. Neither reference teaches or suggests the claimed cross-node, demand-driven caching tied to prior reads of a shared root. Their architectures are incompatible, and combining them would undermine their intended purposes.

Ground 4 pairs Birse with Murphy, but that combination fails for the same reasons as Ground 2. Birse has no concept of indexing, while Murphy's indexing is limited to a client's own file system and is returned to the same client via the server.

Nothing in either reference discloses one compute node indexing a shared root image and providing those results to another compute node, nor would a skilled artisan have been motivated to retrofit them to do so.

The dependent claims are not obvious for the same reasons as their respective independent claims, and many add further narrowing features—such as explicit cache-management policies, read-only root images, or explicit leaf-only re-indexing—that the Petition does not even attempt to map to the prior art. On this record, Petitioner has not established a reasonable likelihood of prevailing on any claim, and the Board should deny institution.

Respectfully submitted,

Dated: August 29, 2025

By: /Brandon R. Theiss /
Brandon R. Theiss
Registration No. 70,507

Counsel for Patent Owner
Intellectual Ventures II LLC

CERTIFICATE OF COMPLIANCE WITH WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), I certify that this PATENT OWNER'S PRELIMINARY RESPONSE complies with the type-volume limits set forth in the Memorandum and 37 C.F.R. § 42.24 because it contains 11,902 words, excluding the parts of the Brief that are exempted by 37 C.F.R. § 42.24(a), according to the word processing system used to prepare this Brief.

Respectfully submitted,

Dated: August 29, 2025

By: /Brandon R. Theiss /
Brandon R. Theiss
Registration No. 70,507

Counsel for Patent Owner
Intellectual Ventures II LLC

CERTIFICATE OF SERVICE

The undersigned certifies that pursuant to 37 C.F.R. § 42.6(e), a copy of the foregoing PATENT OWNER PRELIMINARY RESPONSE was served via email (as consented to by counsel) on August 29, 2025 to lead and backup counsel of record for Petitioner as follows:

Brian M. Buroker (Reg. No. 39,125)
Gibson, Dunn & Crutcher LLP
1700 M St. NW
Washington, DC 20036
Phone: (202) 955-8500
Fax: (202) 467-0539
Email: bburoker@gibsondunn.com

Nathan Curtis (Reg. No. 70,471)
Gibson, Dunn & Crutcher LLP
2001 Ross Avenue, Suite 2100
Dallas, TX 75201
Phone: (214) 698-3423
Fax: (214) 571-2961
Email: ncurtis@gibsondunn.com

GDC-DockerIPR@gibsondunn.com

Dated: August 29, 2025

By: /Brandon R. Theiss /
Brandon R. Theiss
Registration No. 70,507

Counsel for Patent Owner
Intellectual Ventures II LLC