

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In the *Ex Parte* Reexamination of

Intellectual Ventures I LLC

Reexam Control No.: 90/015,337

Confirmation No.: 6365

Filed: February 27, 2003

Patent Number: 7,257,582

Issued: December 11, 2012

For: LOAD BALANCING WITH SHARED
DATA

Group: 3992

Examiner: ANGELA M LIE

Our File:

IVIPR- RE90015337

Date: February 17, 2026

RESPONSE TO OFFICE ACTION IN EX PARTE REEXAMINATION

Mail Stop *Ex Parte* Reexam
Central Reexamination Unit - EFS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

This Response is being timely filed in response to the December 17, 2025

Non-Final Office Action.

TABLE OF CONTENTS

PATENT OWNER’S EXHIBIT LIST	vii
I. INTRODUCTION	1
II. THE ‘582 PATENT	4
A. Specification.....	5
B. Claims	8
C. Inter Partes Review	10
III. ABADI DECLARATION	14
IV. MATERIALS CITED IN THE RE-EXAMINATION	16
A. Iyers.....	16
B. Reeve.....	18
C. Weissman Declaration (EX1003)	23
1. The declaration largely mirrors the Request and adopts counsel-provided legal framing rather than providing independent technical analysis	23
2. The declaration supplies missing limitations through ipse dixit “gap filling,” not disclosure-based analysis	24
3. The declaration’s claim 5 analysis illustrates conclusory “label substitution” rather than teaching of the claimed output model	25
4. The declaration’s treatment of claims 8–11 further confirms it is not supplying disclosure-based proof of dependent limitations	25
5. The declaration does not provide a legally sufficient motivation to combine Iyer with Reeve’s GRAB strategy	26
V. GROUND 1: CLAIMS 1-14 ARE REJECTED UNDER PRE-AIA 35 U.S.C. 103(a) AS BEING UNPATENTABLE OVER IYER IN VIEW OF REEVE.....	28
A. Iyer teaches one partition per processor, not “more partitions than processors,” undermining the Office Action’s “repeat step (c)” theory	31

1.	In Iyer’s sorting flow, partitions are sized and counted to match the number of processors (no over-subscription)	32
2.	The Office Action’s inference from Reeve does not transfer to Iyer; adopting GRAB would require re-architecting Iyer	35
3.	The ‘582 Patent underscores the claimed over-subscription + repeated allocation model that is absent from Iyer	38
4.	The “only two processors are needed” passage is about boundary determination, not about processors repeatedly processing additional unprocessed partitions	41
B.	The Office Action lacks an articulated motivation to combine Iyer with Reeve’s GRAB strategy (beyond a conclusory “efficiency” assertion)	43
1.	“Improve efficiency” is not a substitute for explaining the concrete problem being solved in Iyer, and why GRAB would be selected to solve it.....	44
2.	Implementing GRAB in Iyer is not a minor “efficiency tweak”; it requires changing Iyer’s fundamental work-division model.....	47
3.	The ‘582 Patent highlights why the claimed “repeat” mechanism is not a routine scheduling detail, but an end-to-end system design.....	48
4.	The Office Action also does not address the incompatibilities created by this redesign	49
C.	Reeve Is Not Analogous Art Under Sanofi-Aventis and Cannot Supply the Missing FCFS “Repeat” Limitation or Support the Iyer–Reeve Combination	52
1.	Different field of endeavor: Reeve is about compile-time tiling of an iteration space, not run-time partitioning and processing of a preexisting input file.....	53
2.	Not reasonably pertinent: a POSITA solving the ‘582 Patent’s run-time data-partition allocation problem would not look to compile-time iteration-space tiling.....	54
3.	The Office Action lacks a legally sufficient reason to combine Iyer with Reeve’s GRAB strategy	56

D.	Dependent Claims	59
1.	Iyer + Reeve combination does not teach “an accumulation of output records from said subtasks in an arbitrary order.”	60
2.	Iyer + Reeve combination does not teach (or suggest) the claimed “report-creating process.”	62
3.	Iyer + Reeve combination does not teach (or suggest) the claimed “computer-executable process includes a database query.”	64
E.	Ground 1 Fails to Establish a Prima Facie Case: Iyer Teaches One-Partition-Per-Processor, Reeve Is Non-Analogous, and Dependent Claims Remain Unmet	66
VI.	GROUND 2: CLAIMS 1-14 ARE REJECTED UNDER PRE-AIA 35 U.S.C. 103(a) AS BEING UNPATENTABLE OVER REEVE	68
A.	Reeve does not teach the core “file-partition” limitations of claim 1 (and the Office Action’s mapping relies on an over-abstract characterization of Reeve).....	70
1.	Different object being partitioned: iteration space tiles vs partitions of a preexisting file’s records (file allocation + record boundaries)	70
2.	Different partitioning target: the ‘582 Patent divides the data, while Reeve divides the task.....	72
3.	Different meaning of “First Come First Serve (FCFS)”: FCFS tile-to-thread assignment vs FCFS acquisition of another unprocessed/unallocated file partition after completion	73
a)	Reeve’s FCFS is a scheduling rule for tiles in an iteration space, assigned to threads	74
b)	Claim 1’s FCFS is a repeated allocation mechanism over a pool of unprocessed or unallocated data partitions.....	75
c)	The predicate conditions for claim 1’s FCFS “repeat” limitation are not shown in Reeve.....	76

d)	Reeve’s “tiles” are task fragments, while the ‘582 partitions are data fragments	77
4.	Output generation in Reeve is not the “subtask outputs combined into an output” required by claim 1, as described in the ‘582 Patent	78
B.	The Office Action’s Reeve-alone mapping is structurally flawed because it substitutes generic “parallelism” for the specific file-partition load-balancing architecture of claim 1 ...	80
C.	The Reeve-alone rejection fails to establish a prima facie case because Reeve is non-analogous art, and non-analogous references are outside the “scope and content of the prior art” for an obviousness analysis.....	85
1.	<i>Circuit Check</i> confirms the pertinent question is not whether a concept is generally known, but whether the POSITA would look to that “particular art” to solve “the particular problem at hand”	85
2.	The particular problem addressed by the ‘582 Patent is run-time allocation and processing of data partitions of a preexisting input file, not compile-time tiling of an iteration space.....	86
3.	Under <i>Circuit Check</i> , Reeve is not “reasonably pertinent” because it would not logically commend itself to a POSITA seeking to solve the ‘582 Patent’s run-time data-partition allocation problem	88
4.	Because the Office Action’s Reeve-alone ground depends on treating Reeve as within the scope of the pertinent prior art, a non-analogous Reeve cannot support a prima facie case of obviousness.....	89
D.	Dependent Claims	90
1.	Reeve does not teach the claimed “merging said subtask outputs”	90
2.	Reeve does not teach the claimed “accumulation of output records from said subtasks in an arbitrary order.”	93
3.	Reeve does not teach the claimed “report-creating process.”	95

4. Reeve does not teach the claimed “computer-executable process includes a database query.”	97
E. Ground 2 Fails to Establish a Prima Facie Case: Reeve’s Task-Tiling Framework Does Not Disclose the ‘582 Patent’s Data-Partition Allocation Architecture, and Reeve Is Non-Analogous Art	99
VII. CONCLUSION.....	102

PATENT OWNER'S EXHIBIT LIST

EX2001	Declaration of Dr. Daniel Abadi
EX2002	CV of Dr. Daniel Abadi

I. INTRODUCTION

This Response is submitted in the above-identified ex parte reexamination to address the Non-Final Office Action rejecting claims 1–14 of U.S. Patent No. 7,257,582 (the “‘582 Patent”) under pre-AIA 35 U.S.C. § 103(a). The ‘582 Patent is directed to a specific, implementation-constraining load-distribution technique for accelerating a computer-executable process that would otherwise read a preexisting input file sequentially and process its records one-by-one. The invention achieves this acceleration by first determining file allocation and logically partitioning the input file into well-defined data partitions, typically expressible as consecutive byte-address or track-address ranges, and doing so without reading the file contents during the partitioning step. Only after the split step completes are subtasks activated on a plurality of processors, where each subtask reads and processes allocated file partitions and repeatedly allocates another partition that has not yet been allocated, thereby implementing run-time load balancing on a first-come/first-served basis and ensuring all partitions are processed. This architecture depends on the distinction between the preexisting input file, the logical partitions of that file, and the subtasks that execute on those partitions, as well as the required sequencing that partitioning occurs before subtask execution and that subtasks operate by repeatedly claiming unallocated data partitions until completion.

The Office Action presents two asserted obviousness grounds. Ground 1 rejects claims 1–14 as allegedly obvious over US Patent No. 5,179,699 by Iyer (“Iyer”) in view of US Patent No. 5,535,393 by Reeve (“Reeve”). Ground 2 separately rejects claims 1–14 as allegedly obvious over Reeve alone. Although the Office Action characterizes the references at a high level as “partitioning” and “load balancing,” neither ground supplies a legally and factually adequate mapping to the claim language as actually written, and neither ground supplies an articulated rationale that bridges the substantial architectural gaps between the claimed file-partition allocation system and the cited art.

Ground 1 fails for multiple independent reasons. The Office Action’s theory depends on importing Reeve’s “GRAB” scheduling concept into Iyer to supply claim 1’s requirement that each subtask processor “thereafter repeat” processing “each with another unprocessed partition on a first-come/first-served basis.” But Iyer teaches the opposite work-division model. Iyer’s sorting embodiment is built around defining lists and later partitions so that work is divided “as equally as possible” across processors, using a one-to-one allocation approach that does not contemplate a pool of unallocated file partitions repeatedly claimed by processors after completing earlier work. The “repeat step (c)” limitation in claim 1 presupposes the existence of more partitions than processors and a corresponding mechanism for repeatedly allocating another unprocessed partition, which the Office Action

concedes is not explicitly taught by Iyer. The Office Action attempts to bridge this gap with a generalized “improve efficiency” rationale, but that rationale never identifies a concrete deficiency in Iyer that GRAB is selected to solve, and it never explains—consistent with Iyer’s disclosures—how or why a POSITA would redesign Iyer’s core one-to-one allocation architecture into an over-subscription, pool-based, repeated-claiming regime. In addition, Reeve is not analogous art to the particular problem addressed by the ‘582 Patent and therefore cannot properly serve as the gap-filler for the missing limitations. Finally, even assuming *arguendo* that independent claim 1 could be stretched to fit the Office Action’s mapping, dependent claims 5, 10, and 11 recite additional, narrowing limitations that are not taught or suggested by the cited art and are not established by the conclusory assertions relied upon by Requester.

Ground 2 likewise fails at the threshold and on the merits. Reeve is directed to iteration-space tiling and scheduling of computational tiles to threads. It does not teach the claim 1 framework of automatically determining file allocation and logically subdividing records of a preexisting input file into data partitions (without reading the file contents), distributing descriptions of all such partitions for allocation by subtasks, and implementing run-time load balancing via repeated allocation of another unprocessed (i.e., unallocated) file partition on a first-come/first-served basis until all partitions are processed. The Office Action’s Reeve-

only mapping relies on an over-abstract characterization that treats “tiles” of an iteration space as if they were partitions of a preexisting input file’s records, and treats Reeve’s first-come/first-served assignment of tiles to threads as if it were the claimed mechanism for repeated allocation of unallocated file partitions. Those are not semantic differences; they reflect fundamentally different partitioning targets (data versus computation) and fundamentally different allocation architectures. For the same reasons, and independently, Reeve is non-analogous to the ‘582 Patent’s specific file-partition allocation problem and cannot supply the missing limitations.

Accordingly, as set forth below, the Office Action has not established a prima facie case of obviousness for either Ground 1 or Ground 2. Patent Owner respectfully requests withdrawal of the rejections of claims 1–14.

II. THE ‘582 PATENT

The ‘582 Patent is directed to a specific load-distribution technique for accelerating a computer-executable process that would otherwise read a preexisting input file sequentially and process its records one-by-one. ‘582, 1:62–2:2. The invention achieves this acceleration by first logically partitioning the input file into well-defined data partitions, typically expressible as consecutive byte-address or track-address ranges, and doing so without reading the file contents during the partitioning step. ‘582 Patent, 3:36–59. Only after the split step completes are subtasks activated on a plurality of processors, where each subtask reads and

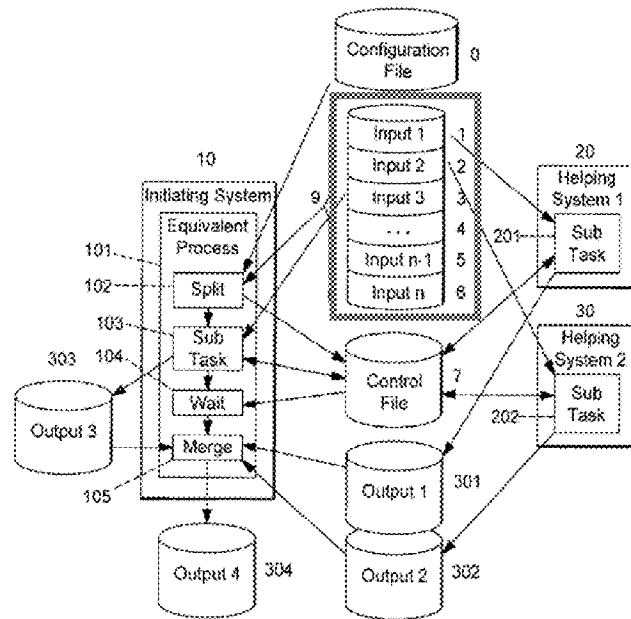
processes allocated file partitions and repeatedly allocates another partition that has not yet been allocated, thereby implementing run-time load balancing on a first-come/first-served basis and ensuring all partitions are processed. ‘582 Patent, 3:64–4:3. This architecture depends on the distinctions between the preexisting input file, the logical partitions of that file, and the subtasks that execute on those partitions, as well as the required sequencing that partitioning occurs before subtask execution and that subtasks operate by repeatedly claiming unallocated data partitions.

A. Specification

The Summary of the ‘582 Patent describes “a method which distributes the load of a process that normally reads an input file sequentially and processes its records one by one between a plurality of potentially heterogeneous processors through the logical partition of the input and the activation of a plurality of sub tasks in said plurality of processors, each said sub task processing the partitions defined by said logical partition in a first come first serve basis.” ‘582 Patent, 1:62-2:2. As explained in the specification and claims of the patent, the “input file” and the “sub tasks” are different. More specifically, the input file includes records that are logically partitioned, while the sub tasks process the logical partitions.

An example of the logical partitioning of input files is depicted in Figure 1 of the ‘582 Patent, reproduced below with colored annotations added by Patent Owner. In Figure 1, “[i]nput files 9 are, thus, logically partitioned into n logical partitions

where n is a number the split step 102 has decided upon. In FIG. 1, these logical partitions [red box] are numbered as 1 to 6.” ‘582 Patent, 3:34-35.



The specification explains that, in the content of the ‘582 Patent, a logical partition is a well-defined part of the input file:

A logical partition in this context is a well-defined part of the input or output. A very simple way to define a partition, which in many cases is the most efficient, would be to define the partitions as consecutive ranges on the input or output, ranging from one relative byte address to another relative byte address or from one relative track address to another. ‘582 Patent, 3:36-41 (emphasis added).

Note that the logical partition does not rely on actual reading of the file. The actual reading of the file is reserved to the subtasks which read the partitions allocated to them. See, e.g., Claim 1, step (c) (“with each subtask reading and processing the respective partition.”) “A splitting function may be

allocating offset 0 to offset 100000 to a first partition, offset 100001 to 200000 to the second partition, offset 200001 to 300000 to the third partition and 300001 to the end of the file to a fourth partition.” ‘582 Patent, 3:55-59.

Thus, in the parlance of the ‘582 Patent, the logical partitions correspond to well-defined portions of an input file, and the logical partitions are different from the sub tasks which, *inter alia*, read and process the logical partitions.

The specification also explains that the activation of the sub tasks occurs only after the logical partitioning of the input file (referred to in the ‘582 Patent as “the split step”) is finished. In this regard, the specification provides:

When the split step terminates, the various *Sub Tasks* (in this case: 103, 202,203) can be activated. This activation can be initiated by the split step 102 itself or by an external scheduler. *Id.*, 3:64-67 (emphasis added).

As explained more fully below, the requirement that the logical partitioning of the input file into partitions occur before any of the subtasks are activated is expressly reflected in the claims of the ‘582 Patent.

After being activated, each sub task is provided with a description of the subdivisions of the input file (reflecting the logical partitions) and uses such information “to allocate for itself and then process, an input partition that has not been allocated yet.” *Id.*, 3:67-4:3.

After the sub tasks have terminated, a merge step merges “the outputs of the various Sub Tasks into the result output file or files.” *Id.*, 4:40-61.

B. Claims

Claim 1 of the ‘582 Patent is the only independent claim, and recites as follows:

1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:
 - (a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;
 - (b) distributing descriptions of all of said partitions to each of a plurality of subtask processors
 - (c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and;
 - (d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and
 - (e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.

Several aspects of the plain and ordinary meaning of claim 1 are worthy of note.

First, the preamble separately introduces an “input file” and “subtasks,” indicating that the two are different. The body of the claim goes on to explain the differences between the two. More specifically, step (a) of the claim recites that the input file includes records that are logically partitioned, while step (c) recites that the subtasks process the logical partitions. Moreover, in the parlance of the ‘582 Patent, the logical partitions correspond to “well-defined” portions of the preexisting input file such as “ranging from one relative byte address to another.” ‘582, 3:36-41.

Second, Claim 1 specifies that an input file is initially subdivided into a sufficient number of partitions to support two rounds of processing on the partitions that resulted from the initial subdividing, i.e., the processing of some partitions in step (c) (the first round) and the further processing of unprocessed partitions in step (d) (the second round). More specifically, step (a) of claim 1 specifies that the input file is initially subdivided into "a plurality of partitions," then step (c) recites "simultaneously executing at least a respective one of the subtasks ... in each of at least some of said processors on a respective one of the partitions," and step (d) requires the repetition of step (c) "in at least some of the subtask processors each with another unprocessed partition." Thus, the claim specifies that the input file is initially subdivided into a sufficient number of partitions to support two rounds of processing on the partitions.

C. Inter Partes Review

The claims 1-14 of the ‘582 Patent were recently challenged by American Airlines, Inc. and Southwest Airlines Co. in IPR2025-00785. The Patent Owner filed a Preliminary Response, and the Board ultimately denied institution, concluding that Petitioner “has not demonstrated a reasonable likelihood of success in proving that any one of the Challenged Claims is unpatentable,” and therefore denying institution under 35 U.S.C. § 314.

With respect to the scope of Claim 1, the Board agreed that “[i]n the ‘582 Patent, an “input file” and “subtasks” are different,” and that “dividing an aggregate tasks into subtasks” does not correspond to “logically subdividing records of said input file in a plurality of partitions” as recited in claim 1. (ID, p. 20) (emphasis in original.) The Board reiterated this important distinction later in the decision, reasoning as follows:

Patent Owner submits that the preamble of claim 1 “separately introduces a ‘preexisting input file’ and ‘subtasks,’ indicating that the two are different,” and that “[t]he body of the claim goes on to explain the differences between the two.” Prelim. Resp. 28. Patent Owner further submits that the Partitioning Limitation “recites that the input file includes records that are logically partitioned, while Element (c) the subtasks process the logical partitions,” and that “the logical partitions correspond to ‘well-defined’ portions of the preexisting input file such as ‘ranging from one relative byte address to another.’” Id. at 28–29 (citing Ex. 1001, 3:36–41). With that context, Patent Owner persuasively argues (and we agree for the same reasons):

The Petition fails to offer any explanation whatsoever why, in the context of the claim language which clearly distinguishes between a file and subtasks, Reiffin's division of tasks and subtasks equates to the logical the [sic] partitioning of the "preexisting input file" as recited in [the Partitioning Limitation]. Moreover, the specification of the '582 Patent makes clear that the file partitions resulting from [the Partitioning Limitation] correspond to "well-defined" portions of the preexisting input file such as "ranging from one relative byte address to another." Ex. 1001, 3:36-41. Completely absent from the Petition's purported mapping of Reiffin to [the Partitioning Limitation] is any explanation as to how Reiffin's subtasks correspond to portions of a preexisting input file such as "ranging from one relative byte address to another." (ID, p. 21-22, quoting POPR, at 30) (emphasis in original.)

* * *

Rather than addressing the language of [the Partitioning Limitation] which expressly refers to logically subdividing a preexisting input file, the Petition simply points to Kurowski's division of "a very large computing problem" into tasks. The Petition fails to offer any explanation whatsoever why, in the context of the claim language which clearly distinguishes between a file and subtasks, Kurowski's division of "a very large computing problem" into tasks equates to the logical the [sic] partitioning of the "preexisting input file" as recited in [the Partitioning Limitation]. (ID, p. 28-29, quoting POPR, at 43) (emphasis in original.)

In denying institution in IPR2025-00785, the Board's analysis frames the '582 Patent at the correct level of technical specificity and, in doing so, rejects the type of over-abstract "parallelism" characterization that underlies the present Reeve-based

theories. The Board summarized the ‘582 Patent as addressing acceleration of a process that “reads an input file sequentially and processes its records one by one” by using “logical partition of the input” and activating subtasks, where “each sub task process[es] the partitions defined by said logical partition” on a first-come/first-served basis. (ID, p. 4.) That description is important because it ties the claimed “partitioning” to data partitions of a preexisting input file, not merely to dividing execution into generic work units. (ID, p. 4.)

The Board then described the operative architecture in concrete, data-centric terms. It explained that the input files are “logically partitioned into n logical partitions,” and that such logical partitions may be defined as “consecutive ranges on the input” such as relative byte-address or track-address ranges. (ID, p. 4; ID, p. 5.) Critically, the Board emphasized that “the logical partition does not rely on actual reading of the file” and that “[t]he actual reading of the file is reserved to the subtasks which read the partitions allocated to them.” (ID, p. 5.) That feature is central to the ‘582 Patent’s claimed invention and sharply distinguishes it from technologies that divide computation over an iteration space without reference to file allocation or file-range partition descriptors. (ID, p. 5.)

The Board further summarized the load-balancing mechanism recited in claim 1 in a way that tracks Patent Owner’s core position in this reexamination. After the split step terminates and the subtasks are activated, each subtask is provided a

description of the subdivisions of the input file and uses that information “to allocate for itself and then process, an input partition that has not been allocated yet.” (ID, p. 5.) The Board reiterated that, in this architecture, “each subtask read[s] and process[es] the partitions on a first come first serve basis.” (ID, p. 5.) This is not merely “FCFS ordering” in the abstract. It is a work-queue style, application-level allocation mechanism over a pool of unallocated data partitions, where “unprocessed” has concrete meaning tied to allocation state and the operative behavior is the repeated claiming of partitions “that ha[ve] not been allocated yet.” (ID, p. 5.)

The Board’s decision is also persuasive because it underscores the burden and evidentiary rigor required when a petitioner asserts unpatentability based on prior art mappings. The Board emphasized that “the petitioner has the burden from the onset to show with particularity why the patent it challenges is unpatentable,” and that this burden “never shifts to Patent Owner.” (ID, p. 7.) That principle is directly relevant here because the Reeve-based theories in the Office Action similarly depend on substituting iteration-space tiling and tile-to-thread scheduling concepts for the ‘582 Patent’s data-partition allocation model and then treating those substitutions as if they were self-evident. The institution denial provides a reasoned, adversarially tested articulation of what the ‘582 Patent actually claims and how the claim language operates in context, and it reinforces that any unpatentability showing must

address the specific run-time allocation of unallocated file partitions and the file-centric partition-definition constraints, rather than relying on generalized notions of “parallelism” or conclusory equivalences.

III. ABADI DECLARATION

Patent Owner submits herewith the Declaration of Dr. Daniel Abadi (EX2001) in support of this Response. Dr. Abadi is a Professor of Computer Science at the University of Maryland and a widely recognized expert in high-performance, parallel, and distributed data processing systems, including large-scale file and database workloads, with extensive academic and industry experience (including commercialization and start-up experience). (EX2001, ¶¶4–6). Dr. Abadi sets forth a POSITA definition consistent with the ‘582 Patent’s technology and explains that he is applying the Phillips claim construction standard as the governing framework in this reexamination. (EX2001, ¶¶21–24).

Dr. Abadi’s opinions directly rebut the Office Action’s central mapping assumptions and the Requester’s expert narrative. First, Dr. Abadi concludes that Iyer’s disclosed sorting flow is built around a one-processor-to-one-partition model that deliberately sizes and counts work partitions to match processor count, thereby avoiding load imbalance by construction, and therefore does not disclose the over-subscription predicate that makes claim 1 step (d) operative. (EX2001, ¶¶81-92). Second, he explains that the Office Action’s proposal to import Reeve’s GRAB

strategy into Iyer is not a plug-in “efficiency” tweak, but would require a substantial redesign of Iyer into a pool-based repeated-claiming regime, introducing technical incompatibilities and unaddressed engineering constraints concerning work-description distribution, allocation coordination, and preservation of merge/concatenation correctness. (EX2001, ¶¶93–97). Third, Dr. Abadi opines that claim 1 step (a) imposes a specific engineering constraint requiring file-allocation-based partitioning of file records, including the ‘582 Patent’s explicit requirement that logical partitions are well-defined, e.g., file-range partitions (e.g., byte/track ranges) subdivided during partition definition, and that neither Iyer nor Reeve teaches this element. (EX2001, ¶¶115–123).

With respect to the Reeve-only ground, Dr. Abadi explains that Reeve is a compiler/runtime iteration-space tiling reference that partitions execution into tiles defined by index/bounds and schedules tiles to threads, and that this task-tiling paradigm is technically distinct from the ‘582 Patent’s run-time allocation and processing of data partitions of a preexisting input file. (EX2001, ¶¶78-79.) He further opines that, viewed from a POSITA perspective, Reeve and the ‘582 Patent address different problems and operate at different levels of the computing stack, supporting Patent Owner’s position that Reeve is not a proper basis for the Office Action’s asserted mappings. (EX2001, ¶¶80).

Finally, Dr. Abadi addresses the dependent claims and concludes that, even assuming *arguendo* the Examiner’s independent-claim mapping, the Office Action does not identify teachings in the applied references that satisfy the additional narrowing limitations of dependent claims (including the claim 5 “arbitrary order” output requirement, the claim 10 “report-creating process” requirement as distinct from post-run telemetry, and the claim 11 “database query” requirement as distinct from generic storage access), and that the Reeve-only ground likewise fails as applied to dependent claims 3, 5, 10, and 11. (EX2001, ¶¶138–158).

IV. MATERIALS CITED IN THE RE-EXAMINATION

A. Iyers

Iyer is directed to improving the performance of a sort by parallelizing sorting and merge operations across multiple processors in a multiprocessor system. Iyer’s disclosed approach is premised on dividing the work so that each available processor is assigned a corresponding portion of the work, rather than creating a pool of many partitions to be dynamically claimed. In particular, Iyer explains that the goal is to minimize sorting time by “dividing the work as equally as possible between the multiple processors,” and it teaches that the “first step is to divide the list into multiple equal sized lists,” where “[t]he number of such lists should be equal to the number of processors available.” (Iyer, 4:21–33). In other words, Iyer teaches a one-to-one allocation between processors and list segments which are then processed in

a single round, and fails to disclose the claimed sequence of operations where the input file is initially subdivided into a sufficient number of partitions to later support two rounds of processing. See, e.g., Office Action, 4 (acknowledging that Iyer fails to “teach repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis.”)

Iyer then reinforces that one-to-one allocation principle at the subsequent partitioning and merge stage. After the sublists are sorted, Iyer describes dividing the lists into partitions for merging and states: “There should be one partition of elements for each available processor.” (Iyer, 4:36–40). Consistent with that structure, Iyer’s merge flow assigns each processor responsibility for merging the lists within its partition. Iyer explains that “each of the P processors merges the lists in its partition to form P sorted lists,” and then “the lists are then simply strung together to form the final sorted lists,” reflecting an ordered merge-and-concatenation output model. (Iyer, 3:45–49). Iyer further emphasizes that partitions can be selected to be nearly the same size so that “no processor load imbalance occurs,” again reflecting Iyer’s design choice to pre-balance work via one partition per processor rather than to rely on dynamic reassignment. (Iyer, 2:45–49).

Iyer also discusses the determination of “partition boundaries” so that processors can identify the portions of each sorted list that belong in a given partition. In the described approach, processors determine partition boundaries and

share partition-related information so that each processor can merge the correct portions of the sorted lists for its assigned partition. (Iyer, 3:33–37; Iyer, 3:45–49). Iyer’s disclosure includes synchronization behavior in its processing flow, including “waiting” behavior in the described implementation, which appears in the context of coordinating the parallel steps rather than describing any particular output-ordering requirement. (Iyer, 11:23–30).

Iyer further describes a cleanup routine executed at program completion (“pr_end”), which is executed once when the program finishes and, in addition to general cleanup, produces statistics and reports about the execution. (Iyer, 8:17–19). Finally, Iyer describes a multiprocessor hardware environment in which processors access main storage through an I/O bus and data may be stored in direct access storage devices (DASD) coupled to the bus, describing general system architecture rather than any database-query semantics. (Iyer, 4:11–16).

B. Reeve

Reeve is directed to a fundamentally different technology than the claimed invention. Rather than allocating and processing partitions of a preexisting input file at run time, Reeve concerns compile-time generation and runtime execution of tiled code for parallelizing an iterative computation over an iteration space. Reeve describes a “Compilation System” that generates a “tiled sequence of instructions” representing an iterative sequence, where a “tile” is defined as a portion of an

“iteration space,” and a “Runtime Library” that manages execution of those tiled instruction sequences. (Reeve, 1:63–2:13). The unit of decomposition in Reeve is therefore a task fragment, namely a tile of a loop’s iteration space, not a data fragment, such as a partition of file records defined by file allocation.

As noted above, claim 1 of the ‘582 Patent separately introduces an “input file” and “subtasks,” indicating that the two are different. More specifically, step (a) of the claim recites that the input file includes records that are logically partitioned, while step (c) recites that the subtasks process the logical partitions. In that context, Reeve partitions instructions in a computer program into subtasks, as opposed to dividing data into logical partitions that are processed by subtasks, as claimed.

Reeve’s interface and control structures reinforce that it is an iteration-space tiling system rather than a file-partition allocation system. Reeve describes a user or compiler directive for defining an “AFFINITY REGION (<index and bounds>)” with parameters including tilesize and selectable strategies “strategy={GRAB, MOD, WAVE, SLICE},” which specifies tiling over an index and bounds-defined region of an iteration space. (Reeve, 32:10–16). Reeve also explains that the Runtime Library influences resource allocation for a “parallel construct” by choosing the number of threads, potentially “more or less threads than available processors,” while delegating much of load balancing to the operating system scheduler. (Reeve, 31:7–13). These disclosures are characteristic of a shared-

memory, thread-based runtime for executing tiled loop constructs, not a control-file-driven mechanism for allocating partitions of a file to subtasks running on helping systems.

Reeve's GRAB strategy likewise operates in this iteration-space scheduling context. Reeve explains that GRAB "divides the iteration space such that there are more tiles than available processors" and issues tiles on a first-come first-serve basis. (Reeve, 4:1–7.) Reeve later reiterates that "tiles are assigned to threads with a pure first come, first serve algorithm." (Reeve, 39:3–6). In describing GRAB's behavior, Reeve emphasizes that the runtime "looks only at the iteration space, and not at the data space," underscoring that Reeve's scheduling decisions are made over an iteration space, not over partitions of a preexisting data object such as a file. (Reeve, 39:15–17). Reeve further discusses affinity in iteration-space terms, such as allowing a thread to access the same portion of the iteration space for each tile family. (Reeve, 40:1–4).

As a result, Reeve, at most, addresses how to divide the execution of a computer program, not how to divide the underlying data being processed. By contrast, the claimed invention divides the data. Claim 1 is anchored to "file allocation" and "logically subdividing records in the preexisting input file ... into partitions," and the load balancing model is implemented by repeatedly allocating an unallocated input partition until all data partitions have been processed. The '582

Patent describes partitions as well-defined parts of the input, such as consecutive byte-address or track-address ranges of a preexisting input file, and it emphasizes that the number of logical input partitions can exceed the number of helping systems so that subtasks repeatedly allocate an input partition “that has not been allocated yet.” (‘582 Patent, 3:35–41, 3:47–50 and 4:1–4). That is a data-centric work-queue model over partitions of the input file itself.

Dr. Abadi provides a helpful analogy for the structural difference between the ‘582 Patent’s claimed architecture and Reeve’s disclosure can be illustrated with a simple analogy.

The 582 Patent is analogous to starting with a preexisting physical object, such as a loaf of bread that already exists on the counter, and accelerating the task by slicing that loaf into many slices based on high level information about the loaf (i.e. it is a foot long). A coordinator can prepare a list of slices without eating or inspecting the bread, but only using the knowledge that it was originally a foot long. For example, it can specify that it will be divided into 24 half-inch slides. Multiple helpers then repeatedly take the next unclaimed slice from the 24 slice pool, , further cut it into smaller pieces, dip each piece into French toast egg batter, fry the pieces, and place the finished pieces into their own trays. When all slices have been processed, the finished pieces from all trays are combined into one basket that is equivalent to the result that would have been produced if one person had made the French toast out of the entire loaf alone. This reflects the 582 Patent’s file centric architecture, in which a preexisting input file is subdivided into file range partitions, those partitions are allocated to subtasks from a pool of unallocated partitions, and the outputs of the

subtasks are later combined to form an output equivalent to the original process.

Reeve is analogous to a different type of parallelization. Instead of starting with an already existing loaf and dividing it into slices, one starts with a recipe and divides the cooking procedure into steps, such as “cut slices into pieces”, “dip pieces into French toast egg batter,” “fry the pieces”, and “make the result look nice with powdered sugar and chocolate drizzle”. The steps have dependencies (e.g. a particular piece can only be fried after it has been dipped into batter), but can still be parallelized in a pipeline fashion --- e.g. one person dips the next piece while the other person fries the previous piece. In short, this model partitions the procedure or task, not a preexisting physical object. This is analogous to Reeve’s disclosure, which partitions an iteration space into tiles defined by index and bounds and tile size directives and schedules those tiles to threads under a strategy such as GRAB. EX2001, ¶¶182-183

The analogy illustrates why the Office Action’s Reeve alone mapping is structurally flawed. Claim 1 requires dividing an existing data object, the preexisting input file, into partitions tied to file allocation and offsets, dynamically allocating those partitions among workers, and then combining subtask outputs into a final output equivalent to the original process. Reeve instead describes dividing a computation into tiles of an iteration space and scheduling those tiles to threads during execution of a parallel construct. The fact that both approaches can be described as “parallel” does not make them equivalent. They operate on different

objects, use different partition definition mechanisms, and produce results through different execution and output models.

Accordingly, even if Reeve is described at a high level as “partitioning,” it is partitioning the execution domain, namely which iterations are executed together and when. (Reeve, 1:63–2:13, 4:1–7 and 32:10–16). It does not teach dividing a preexisting input file into data partitions, distributing data-partition descriptors, and repeatedly allocating unallocated data partitions until completion, which is the architecture recited in claim 1 and described in the ‘582 Patent.

C. Weissman Declaration (EX1003)

Requester relies on the Weissman declaration (EX1003) as the principal evidentiary support for the asserted grounds. The Office should afford the declaration little weight because it largely tracks the Request’s attorney argument, repeatedly adopts counsel-provided legal framing, fills material evidentiary gaps with conclusory assertions, and, critically, does not provide an articulated motivation to combine with a rational underpinning for the Iyer plus Reeve theory.

1. The declaration largely mirrors the Request and adopts counsel-provided legal framing rather than providing independent technical analysis

The declaration’s “analogous art” discussion illustrates the problem. Dr. Weissman states he has “been informed by counsel” of the legal test for analogous art and then applies that test by characterizing the field at a very high level as

“multiprocessor systems.” (Weissman Decl. ¶53). That framing mirrors the Request’s abstraction of the challenged invention into generic “multiprocessor load balancing,” rather than analyzing the ‘582 Patent’s run-time data-partition allocation architecture. (Weissman Decl. ¶53–¶54). By adopting this counsel-driven field definition, the declaration does not provide the technology-tethered analysis needed to show why Reeve’s iteration-space tiling would logically commend itself to a POSITA confronting the ‘582 Patent’s file-partition allocation problem. (Weissman Decl. ¶53–¶54.)

2. The declaration supplies missing limitations through ipse dixit “gap filling,” not disclosure-based analysis

The declaration’s treatment of claim 1’s “thereafter repeating ... each with another unprocessed partition on a first-come/first-served basis” limitation is emblematic. Dr. Weissman asserts that a POSITA “would have modified Iyer’s processors 1–P to implement Reeve’s GRAB strategy to improve efficiency of Iyer’s sort method,” without identifying any teaching in Iyer suggesting the prerequisite redesign that GRAB presupposes (an over-subscribed pool of unprocessed partitions with repeated claiming). (Weissman Decl. ¶82). Instead, the declaration attempts to manufacture the need for GRAB by pointing to “idle and waiting” behavior in Iyer and speculating that waiting is due to “different computing capacities.” (Weissman Decl. ¶83–¶84.) That is not disclosure-based reasoning; it is a post hoc narrative that

supplies the missing mechanism by inference and speculation rather than by what the references actually teach. (Weissman Decl. ¶83–¶84).

3. The declaration’s claim 5 analysis illustrates conclusory “label substitution” rather than teaching of the claimed output model

For claim 5 (“accumulation of output records ... in an arbitrary order”), the declaration asserts that Iyer’s output is “in an arbitrary order” because processors “finished at different times,” citing Iyer’s “waiting” statements. (Weissman Decl. ¶108–¶110). That reasoning substitutes completion timing for an output-ordering rule and fills a missing limitation by conclusory equivalence rather than disclosure. (Weissman Decl. ¶109–¶110).

4. The declaration’s treatment of claims 8–11 further confirms it is not supplying disclosure-based proof of dependent limitations

For claims 8–11 (including “report-creating process” and “database query”), Dr. Weissman opines that these limitations add “no further meaningful limitations,” characterizing them as “preamble labeling.” (Weissman Decl. ¶115–¶116). That position underscores that the declaration is not attempting to identify specific teachings in Iyer or Reeve of report creation or database query execution; it instead seeks to deprive those dependent limitations of patentable weight. (Weissman Decl. ¶115–¶116).

5. The declaration does not provide a legally sufficient motivation to combine Iyer with Reeve's GRAB strategy

Even apart from analogous-art defects, the Weissman declaration does not supply the required articulated reasoning with a rational underpinning for combining Iyer and Reeve. The declaration's stated motivation is that a POSITA would implement GRAB in Iyer "to improve efficiency," and it supports that conclusion by speculating that Iyer's "waiting" indicates that some processors are idle because they have "different computing capacities." (Weissman Decl. ¶¶82–¶84). That alleged motivation is deficient for several reasons.

First, "improve efficiency" is a generic aspiration, not a reasoned explanation of what specific inefficiency exists in Iyer's disclosed approach and why GRAB, in particular, would have been selected as the solution. (Weissman Decl. ¶82). Iyer's disclosed sorting embodiment is structured around pre-balancing work by dividing the list into a number of sublists equal to the number of processors and then defining one merge partition per processor, precisely to avoid load imbalance. (See, e.g., Iyer, 4:25–33; Iyer, 4:36–40). A declaration that simply asserts "efficiency" without engaging with Iyer's stated design principle does not supply a rational underpinning for the specific modification proposed.

Second, the declaration does not bridge the structural gap between Iyer's one-to-one allocation model and the architecture GRAB presupposes. GRAB is

meaningful only where there is a pool of multiple unprocessed work units and processors repeatedly acquire additional units after finishing prior work. Yet the declaration never explains why a POSITA would deliberately depart from Iyer’s one-partition-per-processor model by over-partitioning the work, maintaining a global availability state, and enabling repeated allocation of “another unprocessed partition” on an FCFS basis. (Weissman Decl. ¶¶82–¶84). Instead, it assumes those changes as if they were plug-in scheduling tweaks, which is precisely the kind of hindsight-driven gap filling that does not satisfy the motivation-to-combine requirement.

Third, the declaration’s “different computing capacities” story is not grounded in Iyer. It is speculation used to justify importing GRAB. (Weissman Decl. ¶¶83–¶84). A POSITA cannot be presumed to redesign Iyer based on an assumed problem that the reference itself does not identify, particularly where Iyer teaches selecting partitions to avoid load imbalance. (See, e.g., Iyer, 2:45–49). The declaration therefore provides, at most, an after-the-fact rationale that selects GRAB because it happens to resemble the missing claim language, rather than because the prior art would have motivated the modification.

V. GROUND 1: CLAIMS 1-14 ARE REJECTED UNDER PRE-AIA 35 U.S.C. 103(a) AS BEING UNPATENTABLE OVER IYER IN VIEW OF REEVE

The Office Action rejects claims 1–14 under pre-AIA 35 U.S.C. § 103(a) over Iyer in view of Reeve. (Office Action, p. 4). The Office Action’s theory is that Iyer allegedly supplies the basic parallel-processing framework of claim 1, while Reeve is relied upon to supply the missing requirement that subtask processors “thereafter repeat” processing “each with another unprocessed partition on a first-come/first-served basis [FCFS],” through Reeve’s GRAB strategy. (Office Action, p. 4; Office Action, p. 5). Critically, the Office Action expressly acknowledges that Iyer “does not explicitly teach” this FCFS “repeat” limitation and thus treats Reeve as the necessary gap-filler. (Office Action, p. 4). The Office Action further asserts, in a single sentence, that it would have been obvious to “modify Iyer’s processors 1–P to implement Reeve’s GRAB strategy to improve efficiency of Iyer’s sort method.” (Office Action, p. 5).

As shown below, that rejection fails for multiple independent reasons. At the outset, the Office Action’s mapping of Iyer relies on treating Iyer’s parallel sort/merge scheme for lists as if it were the ‘582 Patent’s run-time, data-partition allocation architecture for a preexisting input file, even though the ‘582 Patent’s load balancing is implemented through over-subscription of data partitions and repeated allocation of partitions “that ha[ve] not been allocated yet.” (‘582, 3:47–50; ‘582,

4:1–4). More fundamentally, the Office Action’s premise for importing GRAB is incompatible with what Iyer actually teaches. Iyer’s disclosed embodiment repeatedly instructs that the number of work units should match the number of available processors, including dividing the list into a number of sublists “equal to the number of processors available” and creating “one partition of elements for each available processor,” precisely so that “no processor load imbalance occurs.” (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40). That one-partition-per-processor design eliminates the pool of unprocessed partitions that must exist for claim 1’s “thereafter repeating ... each with another unprocessed partition” limitation to be operative. (Office Action, p. 4). The Office Action’s suggestion that step (c) “may be repeated” because Reeve has “more tiles than processors” therefore does not transfer to Iyer’s architecture without a substantial redesign that Iyer neither discloses nor suggests. (Office Action, p. 5; Iyer, 4:25–33; Iyer, 4:36–40).

The Office Action’s stated motivation to combine underscores the deficiency. The only articulated rationale is a generalized “improve efficiency” statement. (Office Action, p. 5). But “efficiency” is not an explanation of what specific problem remains in Iyer after it has already achieved “no processor load imbalance,” nor does it explain why a POSITA would abandon Iyer’s pre-balanced, one-to-one partitioning model and instead redesign the system into the over-subscribed, pool-

based work-allocation regime that GRAB presupposes. (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40).

Further, Reeve is not analogous art to the ‘582 Patent’s particular problem and thus cannot properly serve as the Office Action’s gap-filler. The ‘582 Patent addresses a run-time, data-centric scheduling problem in which the system defines logical partitions of a preexisting input file and repeatedly allocates unallocated data partitions among helping systems to keep resources utilized. (‘582, 1:46–55; ‘582, 3:47–50; ‘582, 4:1–4). Reeve, by contrast, is directed to compile-time generation and runtime scheduling of tiled execution of an iterative sequence over an iteration space, where tiles are portions of an iteration space and are issued to threads under GRAB on a first-come/first-served basis. (Reeve, 1:63–2:13; Reeve, 4:1–7; Reeve, 39:3–6). Under the Federal Circuit’s analogous-art framework, the relevant inquiry must be tethered to the challenged patent’s field and problem, and a reference outside that scope cannot supply a missing limitation via an after-the-fact “efficiency” assertion. *Sanofi-Aventis Deutschland GmbH v. Mylan Pharms. Inc.*, 66 F.4th 1373, 1380–82 (Fed. Cir. 2023); (Office Action, p. 4; Office Action, p. 5).

Finally, even assuming *arguendo* that the Office Action had adequately mapped claim 1, dependent claims 5, 10, and 11 recite additional narrowing limitations that the proposed combination does not teach or suggest, including claim 5’s “accumulation of output records ... in an arbitrary order,” claim 10’s requirement

that the executed process is itself report-creating (not merely emitting post-run telemetry), and claim 11's requirement that the executed process includes a database query (not merely generic disk access). (Office Action, p. 6; Office Action, p. 7). For all of these reasons, the Office Action has not carried its burden under § 103(a), and Ground 1 should be withdrawn.

A. Iyer teaches one partition per processor, not “more partitions than processors,” undermining the Office Action’s “repeat step (c)” theory

The Office Action relies on Reeve's GRAB strategy to supply the limitation in claim 1 requiring that subtask processors “thereafter repeat” processing “each with another unprocessed partition on a first-come/first-served basis,” and it asserts that because Reeve has “more tiles (partitions) than processors,” step (c) “may be repeated.” (Office Action, p. 4; Office Action, p. 5). But that “repeatability” premise is not supported by Iyer's disclosed sorting embodiment that the Office Action uses for the remaining limitations of claim 1. To the contrary, Iyer's architecture is expressly designed around matching the number of work units to the number of available processors, thereby eliminating the unprocessed-partition pool that would be necessary for “repeating step (c) ... with another unprocessed partition” in the claim sense. (Iyer, 4:23–33).

1. In Iyer's sorting flow, partitions are sized and counted to match the number of processors (no over-subscription)

Iyer explains that when sorting a list using multiple processors, “the goal is to perform the sort in the shortest period of time,” which is achieved “by dividing the work as equally as possible between the multiple processors.” (Iyer, 4:21–24). Consistent with that goal, Iyer teaches that the “first step is to divide the list into multiple equal sized lists,” and critically, “[t]he number of such lists should be equal to the number of processors available.” (Iyer, 4:25–33). After those lists are sorted, Iyer again adopts a one-to-one mapping: “The second step in the sorting process is to divide the lists into partitions,” and “There should be one partition of elements for each available processor.” (Iyer, 4:36–40).

That disclosure is fundamentally inconsistent with the Office Action's reliance on a “more partitions than processors” scheduling premise to supply claim 1's “thereafter repeating” limitation. Under Iyer's express approach, each processor is assigned its work from a set of partitions purposefully defined to match the number of processors, not from a larger pool of unprocessed partitions that processors repeatedly claim after completing initial work. (Iyer, 4:25–33; Iyer, 4:36–40). Indeed, Iyer describes the merge phase as one in which “each of the P processors merges the lists in its partition,” followed by the lists being “simply strung together to form the final sorted lists,” emphasizing that “each partition is nearly the same

size so no processor load imbalance occurs.” (Iyer, 3:44–50; Iyer, 2:45–49). In other words, Iyer’s solution is static equalization (avoid imbalance up front), not dynamic reallocation (address imbalance during execution).

Dr. Abadi provides the helpful analogy of multiple chefs preparing a set of tasks to illustrate the distinction between Iyer’s approach and a dynamic “repeat/claim from a pool” approach of the ‘582 Patent.

Under Iyer’s approach, the head chef first divides the total prep work into exactly as many plates as there are chefs, with each plate intentionally sized to be about the same effort. Each chef is assigned one plate and completes that plate. When a chef finishes, the chef’s assigned work is complete because there are not additional unassigned plates waiting to be claimed. This corresponds to Iyer’s express teaching that, in the first step, “the list [is divided] into multiple equal sized lists,” and “[t]he number of such lists should be equal to the number of processors available,” followed by “one partition of elements for each available processor.” (Iyer, 4:25–33; Iyer, 4:36–40.)

By contrast, a dynamic “repeat/claim” scheduling model is like a kitchen that intentionally creates many more work items than chefs, such as a rail of numerous small order tickets. Each chef takes one ticket, completes it, and then returns to the rail to take the next available ticket. Faster chefs naturally complete more tickets, and slower chefs complete fewer, so the kitchen balances workload during execution through repeated claiming of additional unassigned tasks. In High Performance Computing (HPC) terms, that is a self-scheduling or work-queue model characterized by over-subscription and repeated task acquisition.

Iyer does not describe the “order ticket rail” model. Instead, Iyer’s text aligns with the “one plate per chef” model because it

directs that the number of initial lists should match the number of processors, and that there should be one partition for each processor, with the explicit goal of avoiding load imbalance by making partitions “nearly the same size so no processor load imbalance occurs.” (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40.) In other words, Iyer’s solution is to avoid imbalance up front through static equalization, not to address imbalance by repeatedly allocating additional unprocessed work items after a processor completes its initial assignment.

EX2001, ¶¶89-91

Accordingly, Iyer affirmatively teaches away from the over-subscription premise that must exist for claim 1’s step (d) to be operative. Claim 1 requires that, after the initial simultaneous execution in step (c), at least some subtask processors “thereafter” repeat step (c) “each with another unprocessed partition” on a first-come/first-served basis, which presupposes the existence of additional unprocessed partitions available for allocation after a first round of processing partitions. Iyer’s disclosed sorting flow eliminates that predicate condition by expressly directing that the number of work units be matched to the number of available processors, first by dividing the list into a number of sublists equal to the number of processors and then by defining “one partition of elements for each available processor,” with the stated objective that partitions be nearly the same size so “no processor load imbalance occurs.” (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40). As Dr. Abadi’s kitchen analogy illustrates, Iyer is a “one plate per chef” system, not an “order ticket rail” system in which processors repeatedly claim additional unassigned work from a

pool. Because Iyer does not disclose, and in fact rejects, the multi-round processing architecture required by the ‘582 Patent’s FCFS repetition model, the Office Action’s reliance on Reeve’s “more tiles than processors” concept does not cure Iyer’s deficiency without fundamentally departing from Iyer.

2. The Office Action’s inference from Reeve does not transfer to Iyer; adopting GRAB would require re-architecting Iyer

The Office Action acknowledges that “Iyer ... does not explicitly teach repeating step (c) ... each with another unprocessed partition on a first-come/first-served basis,” and then turns to Reeve for that feature. (Office Action, p. 4). The Office Action further reasons that because Reeve teaches “more tiles (partitions) than processors,” “the execution of step (c) may be repeated,” and concludes that it would have been obvious to “modify Iyer’s processors 1-P to implement Reeve’s GRAB strategy to improve efficiency of Iyer’s sort method.” (Office Action, p. 5).

That reasoning rests on a predicate that Iyer does not disclose: the existence of more partitions than processors and a corresponding pool of unprocessed work that processors repeatedly “grab” after completing previously assigned work. Iyer instead teaches choosing the number of lists and partitions to equal the number of processors available, precisely to balance load and avoid idle time. (Iyer, 4:25–33; Iyer, 4:36–40; Iyer, 2:45–49). Importing GRAB into Iyer is therefore not a simple “efficiency” tweak; it would require departing from Iyer’s stated design and re-

engineering Iyer into a different work-distribution regime (i.e., intentionally defining more partitions than processors, maintaining a shared pool, and enabling repeated allocation of “another unprocessed partition”). (Office Action, p. 5; Iyer, 4:25–33; Iyer, 4:36–40).

As Dr. Abadi explains “the Office Action’s inference, drawn from Reeve, that “more tiles than processors” implies that execution “may be repeated” via GRAB does not transfer to Iyer without changing Iyer’s foundational work-distribution regime. Stated differently, adopting GRAB in Iyer would not be a simple “efficiency” tweak to Iyer’s existing parallel sort flow; it would require a redesign with several interdependent changes.” EX2001, ¶97

As *KSR* explains, “a patent composed of several elements is not proved obvious merely by demonstrating that each of its elements was, independently, known in the prior art,” and it remains “important to identify a reason that would have prompted a person of ordinary skill in the relevant field to combine the elements in the way the claimed new invention does.” *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 418–19 (2007). Conclusory expert testimony cannot supply that missing reasoning. *I/P Engine, Inc. v. AOL Inc.*, 576 F. App’x 982, 999 (Fed. Cir. 2014). Nor may Petitioners use the challenged claims as a “template” to piece together disparate teachings from the prior art. *In re Fritch*, 972 F.2d 1260, 1266 (Fed. Cir. 1992) (citing *In re Gorman*, 933 F.2d 982, 987 (Fed. Cir. 1991)); see also

Metalcraft of Mayville, Inc. v. The Toro Co., 848 F.3d 1358, 1367 (Fed. Cir. 2017) (without explanation of how/why references would be combined to arrive at the claimed invention, the analysis devolves into hindsight). And even where a modification is technically possible, “the mere fact that the prior art may be modified in the manner suggested . . . does not make the modification obvious unless the prior art suggested the desirability of the modification.” *Fritch*, 972 F.2d at 1266 (citing *In re Gordon*, 733 F.2d 900, 902 (Fed. Cir. 1984)). Obviousness therefore requires more than identifying separate references that, in hindsight, can be mapped to separate limitations; it requires evidence that a POSITA “would have selected and combined those prior art elements in the normal course of research and development to yield the claimed invention.” *Unigene Labs., Inc. v. Apotex, Inc.*, 655 F.3d 1352, 1360 (Fed. Cir. 2011).

Accordingly, the Office Action’s reliance on Reeve’s “more tiles than processors” concept does not supply claim 1’s FCFS “repeat” limitation in the context of Iyer’s disclosed system. Iyer affirmatively teaches a one-to-one work-division regime in which the number of lists and merge partitions is selected to match the number of available processors in order to avoid load imbalance, not an over-subscribed pool of unprocessed partitions that processors repeatedly claim after completing prior work. (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40). As Dr. Abadi explains, the inference the Office Action draws from Reeve does not “transfer to

Iyer” without changing Iyer’s foundational work-distribution regime, and adopting GRAB would require a redesign with several interdependent changes rather than a simple plug-in optimization. (EX2001, ¶97]). The Office Action’s conclusory “improve efficiency” rationale does not identify a concrete problem in Iyer that would motivate that redesign, does not explain how the redesign would be implemented while preserving Iyer’s merge/concatenation correctness and performance objectives, and therefore does not provide the articulated reasoning with rational underpinning required by *KSR* and its progeny.

3. The ‘582 Patent underscores the claimed over-subscription + repeated allocation model that is absent from Iyer

The claimed invention expressly contemplates and leverages the concept of multiple rounds of processing of partitions. For example, the ‘582 Patent explains that “the number of logical input partitions will largely exceed the number of helping systems,” and that each subtask “repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (‘582 Patent, 3:47–50 and 4:1–4). That is the claimed load-balancing model: multiple processing rounds including allocation of unallocated partitions after the first round.

Iyer’s sorting embodiment discloses the opposite approach: equalize the work by defining the number of lists (and later partitions) to match the number of processors, such that “no processor load imbalance occurs.” (Iyer, 4:25–33, 4:36–40

and 2:45–49). Because Iyer does a predicate that makes “thereafter repeating ... with another unprocessed partition” operative, the Office Action’s reliance on Reeve’s “more tiles than processors” premise does not cure Iyer’s acknowledged deficiency without fundamentally departing from Iyer’s described architecture.

Instead, as Dr. Abadi explains, “because Iyer does not disclose the over-subscription condition and corresponding pool of unallocated partitions that the ’582 Patent identifies as central to its load-balancing operation, the Office Action’s reliance on Reeve’s “more tiles than processors” premise does not cure Iyer’s acknowledged deficiency without departing from Iyer’s described architecture and redesigning Iyer into a different scheduling regime.” EX2001, ¶109

The Federal Circuit has expressly explained “[o]bviousness may not be established using hindsight or in view of the teachings or suggestions of the inventor.” *Para-Ordnance Mfg. v. SGS Importers Int’l*, 73 F.3d 1085, 1087 (Fed. Cir. 1995) (citing *W.L. Gore & Assocs., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1551, 1553 (Fed. Cir. 1983)). “The mere fact that the prior art may be modified in the manner suggested . . . does not make the modification obvious unless the prior art suggested the desirability of the modification.” *In re Fritch*, 972 F.2d 1260, 1266 (Fed. Cir. 1992) (citing *In re Gordon*, 733 F.2d 900, 902, 221 USPQ 1125, 1127 (Fed. Cir. 1984)). “It is impermissible to use the claimed invention as an instruction manual or ‘template’ to piece together the teachings of the prior art so that the

claimed invention is rendered obvious.” *Fritch*, 972 F.2 at 1266 (citing *In re Gorman*, 933 F.2d 982, 987 (Fed. Cir. 1991)).

Accordingly, the ‘582 Patent’s disclosure confirms that the claimed FCFS “repeat” mechanism is not an incidental scheduling preference but an architecture that depends on multiple process rounds including allocation of partitions subsequent to the first round that have “not been allocated yet.” (‘582 Patent, 3:47–50 and 4:1–4). Iyer’s disclosed sorting embodiment is built on the opposite premise, namely selecting the number of lists and merge partitions to match the number of processors so that “no processor load imbalance occurs,” leaving no pool of unallocated work units for processors to claim in a subsequent processing round. (Iyer, 2:45–49, 4:25–33 and 4:36–40). As Dr. Abadi explains, because Iyer does not disclose an over-subscription condition or a pool of unallocated partitions, the Office Action’s reliance on Reeve’s “more tiles than processors” premise cannot cure Iyer’s acknowledged deficiency without departing from Iyer’s described architecture and redesigning Iyer into a different scheduling regime. That type of after-the-fact reconstruction is precisely what the Federal Circuit has prohibited: obviousness cannot be established through hindsight or by using the claimed invention as a template to piece together disparate teachings from the prior art where the prior art itself does not suggest the desirability of the modification.

4. The “only two processors are needed” passage is about boundary determination, not about processors repeatedly processing additional unprocessed partitions

To the extent that the Examiner were to allege that Iyer contemplates scenarios where “only two processors” are used even “when three processors are available,” implying an imbalance or spare capacity that could lead to “repetition,” that new theory would also be lacking. Iyer’s disclosure makes clear that this “only two processors” discussion concerns determining partition boundaries, not executing merges (or any other step-(c)-type processing) on a succession of “unprocessed partitions.” (Iyer, 4:60–63).

Specifically, Iyer explains that “only two partition boundaries need to be determined” and therefore “it only requires two processors to perform the partitioning when three processors are available to perform the entire sort.” (Iyer, 4:60–63). That is a statement about the boundary-definition task (i.e., identifying the cut points that define partitions), not a statement that processors iteratively “grab” additional merge partitions after finishing prior work. Indeed, immediately after boundary determination, Iyer describes the merge phase as one in which processors merge the lists within their assigned partitions and then the merged partitions are concatenated. (Iyer, 3:44–50).

Even accepting that boundary determination can be performed by fewer than all available processors, that does not disclose a system where processors repeatedly

execute step (c) “each with another unprocessed partition” on a FCFS basis. (Office Action, p. 4). As Dr. Abadi explains, “A boundary computation that does not fully occupy all processors is a common preprocessing characteristic in parallel algorithms that use domain decomposition and does not establish the existence of more partitions than processors, a shared pool of unprocessed partitions remaining after initial assignments, or repeated allocation of additional unprocessed partitions to processors after completing prior work.” EX2001, ¶113

Accordingly, the “only two processors are needed” passage cannot supply, and does not suggest, claim 1’s step (d) requirement that subtask processors “thereafter” repeat step (c) “each with another unprocessed partition” on a first-come/first-served basis. Iyer’s statement is limited to the mechanics of determining partition boundaries in a particular scenario, and it says nothing about a pool of unprocessed partitions or any mechanism by which, after completion of initially assigned processing work, processors continue operations on the preexisting input file by processing “another unprocessed partition” FCFS. (Iyer, 4:60–63). Nor does it alter Iyer’s overall processing flow, which proceeds from boundary determination to a merge phase in which processors merge within their assigned partitions and the results are concatenated, reflecting a static assignment model rather than repeated task acquisition. (Iyer, 3:44–50). Therefore, even if the Examiner were to invoke Iyer’s boundary-determination passage, it would not cure the Office Action’s failure

to show the claimed FCFS “repeat/allocate another unprocessed partition” architecture. (Office Action, p. 4).

B. The Office Action lacks an articulated motivation to combine Iyer with Reeve’s GRAB strategy (beyond a conclusory “efficiency” assertion)

The Office Action concedes that Iyer does not disclose claim 1’s requirement that processors “thereafter repeat” processing “each with another unprocessed partition on a first-come/first-served basis,” and therefore relies on Reeve’s GRAB strategy to supply that limitation. (Office Action, p. 4). The Office Action then asserts that it would have been obvious to “modify Iyer’s processors 1–P to implement Reeve’s GRAB strategy to improve efficiency of Iyer’s sort method.” (Office Action, p. 5). That stated rationale is insufficient because it does not explain how or why a POSITA would restructure Iyer’s disclosed workflow—one that is expressly designed around a one-to-one partition/processor mapping to avoid load imbalance—into a GRAB-style over-subscribed scheduling regime premised on a shared pool of unprocessed work.

Instead, as Dr. Abadi explains “In my opinion, a POSITA would not have been motivated to combine Iyer’s parallel sorting and merging technique with Reeve’s iteration space tiling and GRAB scheduling in the manner suggested, because the two references address different computational problems, operate on different

decomposition targets, and embody materially different execution models.”

EX2001, ¶124

1. “Improve efficiency” is not a substitute for explaining the concrete problem being solved in Iyer, and why GRAB would be selected to solve it

Iyer’s stated goal is to minimize sorting time by dividing work “as equally as possible” across multiple processors. (Iyer, 4:21–24). Consistent with that goal, Iyer instructs that the number of initial divided lists “should be equal to the number of processors available,” and then again that “there should be one partition of elements for each available processor.” (Iyer, 4:25–33 and 4:36–40). Iyer further emphasizes that, because each partition is “nearly the same size,” “no processor load imbalance occurs.” (Iyer, 2:45–49).

Given those disclosures, the Office Action’s “efficiency” rationale never identifies a deficiency in Iyer that GRAB purportedly remedies. In fact, Iyer’s express design objective is to avoid the very scenario that makes GRAB useful (variable-cost work units and idle processors that must “steal” remaining work). (Iyer, 2:45–49, 4:25–33 and 4:36–40). Without explaining what inefficiency remains in Iyer’s own design (after it has already achieved “no processor load imbalance”), the asserted motivation to import GRAB is, at most, a generalized statement that “different scheduling might be efficient,” which is not enough to justify a specific combination. (Office Action, p. 5).

Dr. Abadi further explains:

“...a POSITA seeking to implement or improve Iyer’s sorting flow would not look to Reeve for the specific feature the Office Action attributes to the combination, namely repeated processing of “another unprocessed partition” after completion of prior work. The reason is that Iyer’s core design objective is to eliminate the need for dynamic rescheduling by selecting the number and size of partitions to avoid processor load imbalance. (Iyer, 2:45–49; Iyer, 4:25–33; Iyer, 4:36–40.) Introducing Reeve’s GRAB mechanism would undermine Iyer’s express design choice and would require deliberately departing from Iyer’s one to one partitioning scheme, including creating more work units than processors and introducing shared state to manage repeated allocation. A POSITA would recognize that this is not a routine “efficiency improvement” within Iyer’s framework, but instead a shift to a different class of scheduling regime, with different correctness and coordination considerations.” EX2001, ¶128

Accordingly, the Office Action’s generic “improve efficiency” rationale does not supply the articulated motivation to combine required for obviousness, particularly where Iyer’s own disclosures affirmatively counsel against the premise that makes Reeve’s GRAB strategy relevant. Iyer expressly seeks to eliminate load imbalance by construction, instructing that the number of sublists and partitions should match the number of available processors and emphasizing that partitions are made “nearly the same size so no processor load imbalance occurs.” (Iyer, 2:45–49, 4:21–24, 4:25–33 and 4:36–40). As Dr. Abadi explains, a POSITA attempting to implement or improve Iyer would not look to Reeve for repeated “claim another

unprocessed partition” behavior, because doing so would require abandoning Iyer’s one-to-one partitioning scheme and introducing shared state and coordination to support repeated allocation, which is not a routine “efficiency improvement” within Iyer’s framework but a shift to a different class of scheduling regime with different coordination and correctness considerations.

That conclusion is consistent with the Federal Circuit’s motivation-to-combine jurisprudence, including *Arctic Cat*, which explains that even where some evidence could be marshaled in favor of combining elements, the analysis must account for record evidence and engineering realities that temper or negate the purported motivation, including reasons a skilled artisan would be discouraged from pursuing the asserted combination and reasons the combination would not yield predictable, expected results. *Arctic Cat Inc. v. Bombardier Recreational Prods. Inc.*, 876 F.3d 1350, 1360–64 (Fed. Cir. 2017). Here, Iyer itself supplies that tempering evidence: it teaches a design specifically structured to avoid imbalance through static equalization, leaving no identified deficiency that GRAB would remedy, and the Office Action offers no explanation of how or why a POSITA would intentionally depart from Iyer’s stated design objective to adopt an over-subscribed, pool-based repeated-claiming regime. (Iyer, 2:45–49, 4:21–24, 4:25–33 and 4:36–40; Office Action, p. 5). On this record, the asserted “efficiency” rationale is, at most, an after-the-fact label that does not bridge the gap between Iyer’s disclosed architecture and

the materially different scheduling paradigm the Office Action seeks to import from Reeve.

2. Implementing GRAB in Iyer is not a minor “efficiency tweak”; it requires changing Iyer’s fundamental work-division model

To satisfy claim 1’s “thereafter repeating ... each with another unprocessed partition” limitation (which the Office Action admits is missing from Iyer), the modified system must include, at a minimum: (i) the existence of a pool of unprocessed partitions from the preexisting input file subsequent to a first round of processing partitions of the preexisting input file, and (ii) a mechanism for processors, subsequent to the first round, to claim unprocessed partitions on a First Come First Serve (FCFS) basis. (Office Action, p. 4; Office Action, p. 5.)

But Iyer’s disclosed sorting embodiment is explicitly built around not having that architecture. Iyer teaches choosing the number of lists and partitions to equal the number of processors available, precisely to pre-balance the workload and avoid idle time. (Iyer, 4:25–33, 4:36–40 and 2:45–49). That is not an incidental detail—it is the stated organizing principle of Iyer’s parallel sorting approach. (Iyer, 4:21–24).

Specifically, Dr. Abadi explains:

Introducing Reeve’s GRAB mechanism would undermine Iyer’s express design choice and would require deliberately departing from Iyer’s one to one partitioning scheme, including creating more work units than processors and introducing shared state to manage repeated allocation. A POSITA would

recognize that this is not a routine “efficiency improvement” within Iyer’s framework, but instead a shift to a different class of scheduling regime, with different correctness and coordination considerations. EX2001, ¶128

Therefore, “modifying Iyer to implement GRAB” is not a matter of swapping out one scheduling rule for another; it requires re-architecting Iyer from a static, one-to-one assignment model (partitions = processors) into a model in which a pool of unprocessed partitions from the preexisting input file exist subsequent to a first round of processing partitions of the preexisting input file and a mechanism is introduced for processors, subsequent to the first round, to claim unprocessed partitions on a First Come First Serve (FCFS) basis.. (Iyer, 4:25–33 and 4:36–40; Office Action, p. 4-5). The Office Action does not explain why a POSITA would take that step given that Iyer already teaches that its approach avoids processor load imbalance. (Iyer, 2:45–49).

3. The ‘582 Patent highlights why the claimed “repeat” mechanism is not a routine scheduling detail, but an end-to-end system design

The ‘582 Patent expressly contemplates over-subscription condition where “the number of logical input partitions will largely exceed the number of helping systems.” (‘582 Patent, 3:47–50). It then describes the operative consequence: each subtask “repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (‘582 Patent, 4:1–4). Similarly, the “repeat step (c)”

requires a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file.

That is exactly what is absent from Iyer's disclosed sorting embodiment, which instead directs that partitions are defined so that "no processor load imbalance occurs." (Iyer, 2:45–49). The Office Action does not bridge this architectural gap; it simply asserts that adopting GRAB would "improve efficiency." (Office Action, p. 5). But the claimed "repeat/allocate another unprocessed partition" behavior is inconsistent with Iyer's static one-to-one partitioning design. ('582 Patent, 3:47–50 and 4:1–4; Iyer, 4:25–33 and 4:36–40).

4. The Office Action also does not address the incompatibilities created by this redesign

The Office Action states, in conclusory fashion, that it would have been obvious to "modify Iyer's processors 1 to P to implement Reeve's GRAB strategy to improve efficiency of Iyer's sort method." (Office Action, p. 5). As Dr. Abadi explains from a High Performance Computer (HPC) engineering perspective, once one departs from Iyer's disclosed one-to-one partitioning model and restructures Iyer into a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from

the input file, additional technical incompatibilities and design constraints arise that the Office Action does not address. (EX2001, ¶96).

First, a multi-round processing regime requires that processors have access to a representation of the available work units so that they can obtain an unprocessed partition subsequent to an initial round of processing of partitions from the input file. In practice, this requires generating and distributing “descriptions of all partitions” in a form that can be queried or claimed by processors, and it requires coordination so that each partition is claimed once. The Office Action does not explain how such partition descriptions and coordination would be implemented in Iyer, which instead teaches dividing the list into equal sized lists and selecting the number of lists and partitions to match the number of processors. (EX2001, ¶96; Iyer, 4:25–33 and, 4:36–40).

Second, converting Iyer’s merge execution into a regime where allocation/processing of unallocated partitions occurs subsequent to an initial round of processing of partitions from the input file introduces correctness questions for Iyer’s merge and concatenation logic that the Office Action does not address. Iyer describes a merge phase in which “each of the P processors merges the lists in its partition,” followed by the merged lists being “simply strung together to form the final sorted lists.” (EX2001, ¶112; Iyer, 3:44–50). That description presumes predetermined partition assignments and a known concatenation order sufficient to

yield a correct globally sorted output. If allocation/processing of unallocated partitions instead occurs subsequent to an initial round of processing of partitions from the input file, the system must specify how the outputs are to be organized and concatenated to preserve correctness, including maintaining the proper ordering relationships among merged partitions. The Office Action does not identify where Iyer or Reeve teaches these integration details, and it does not explain how Iyer's correctness invariants would be preserved in a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file. (EX2001, ¶128).

Third, introducing over-subscription and dynamic claiming is in tension with Iyer's stated design objective of dividing work "as equally as possible" to complete sorting in the shortest time. (EX2001, ¶90; Iyer, 4:21–24). Iyer achieves that objective by selecting the number of lists and partitions to match the number of processors and by making each partition nearly the same size so that "no processor load imbalance occurs." (EX2001, ¶85-86; Iyer, 2:45–49, 4:25–33 and 4:36–40). If the system shifts to a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file, the design goal shifts from static equalization to runtime balancing, which introduces overhead such as synchronization, coordination, and potentially increased merge complexity, and no

longer preserves Iyer’s stated premise that balance is achieved by constructing partitions that are nearly the same size. The Office Action does not address these tradeoffs or explain why a POSITA would expect the redesigned system to maintain Iyer’s stated equal-division goal.

As Dr. Abadi concludes, these unaddressed issues illustrate that the proposed modification is not a straightforward optimization of Iyer. It is a substantial redesign that raises multiple engineering questions concerning work-description distribution, allocation coordination, and preservation of correctness and performance objectives, none of which are addressed by the Office Action’s conclusory “improve efficiency” rationale.

C. Reeve Is Not Analogous Art Under Sanofi-Aventis and Cannot Supply the Missing FCFS “Repeat” Limitation or Support the Iyer-Reeve Combination

Under the Federal Circuit’s analogous-art framework, a reference qualifies as analogous only if it is either in the same field of endeavor or “reasonably pertinent to the particular problem with which the inventor is involved.” *Sanofi-Aventis Deutschland GmbH v. Mylan Pharms. Inc.*, 66 F.4th 1373 (Fed. Cir. 2023). The Court emphasized that the inquiry must be tethered to the challenged patent’s field and problem, and it rejected an approach that defines the “problem” at an impermissibly abstract level or derives it from the prior art rather than the challenged patent. *Id.*; see *id.* at 1380–82.

Here, the Office Action relies on Reeve’s GRAB strategy to supply claim 1’s requirement that processors “thereafter repeat” processing “each with another unprocessed partition on a first-come/first-served basis,” after expressly acknowledging that Iyer does not teach that limitation. (Office Action, p. 4-5.)

1. Different field of endeavor: Reeve is about compile-time tiling of an iteration space, not run-time partitioning and processing of a preexisting input file

The ‘582 Patent is directed to run-time processing of data—specifically, accelerating a process that “normally reads an input file sequentially and processes its records one by one” by distributing the work among helping systems that process file partitions. (‘582 Patent, 1:46–55). The ‘582 Patent’s load balancing is explicitly implemented as a run-time allocation mechanism over data partitions: “Normally, the number of logical input partitions will largely exceed the number of helping systems,” and “Each such Sub Task then repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (‘582 Patent, 3:47–50 and 4:1–4).

Reeve, by contrast, is directed to compilation and code generation prior to execution. Reeve’s disclosure centers on a “Compilation System” and generating a “tiled sequence of instructions” that represents an iterative sequence, where a “tile” is defined as a portion of an “iteration space.” (Reeve, 1:1–8 and 1:63–2:13). Reeve’s GRAB strategy is likewise rooted in iteration-space tiling: it “divides the iteration

space such that there are more tiles than available processors” and issues tiles “on a first-come first-serve basis.” (Reeve, 4:1–7). Reeve later reiterates the same point in scheduling terms: “In the grab strategy, tiles are assigned to threads with a pure first come, first serve algorithm.” (Reeve, 39:3–6).

This is a fundamental field mismatch. The ‘582 Patent is addressing a run-time, data-centric scheduling problem—how to allocate and process partitions of a preexisting input file at execution time, including repeated allocation of unallocated partitions. (‘582 Patent, 1:46–55, 3:47–50 and 4:1–4). Reeve is addressing a compile-time/program-structure problem—how to compile and arrange loop-based code into a tiled instruction sequence over an iteration space and then schedule iteration-space tiles to threads. (Reeve, 1:63–2:13, 4:1–7 and 39:3–6).

As Dr. Abadi expressly states: “in my opinion, the ‘582 Patent and Reeve are directed to different fields of endeavor, and Reeve would not have been a natural reference a POSITA would consult when addressing the specific shared file partition allocation and repeated allocation load balancing techniques described and claimed in the 582 Patent.” EX2001, ¶80

2. Not reasonably pertinent: a POSITA solving the ‘582 Patent’s run-time data-partition allocation problem would not look to compile-time iteration-space tiling

Sanofi-Aventis cautions that the “problem” cannot be defined so broadly that it collapses the analogous-art inquiry into “anything about parallelism.” *Sanofi-*

Aventis, 66 F.4th at 1380–82. The relevant problem here is the one the ‘582 Patent actually addresses: run-time balancing of a file-processing workload by defining logical input partitions and repeatedly allocating unallocated partitions among helping systems to keep resources utilized proportional to capacity. (‘582 Patent, 3:47–50; ‘582, 4:1–4).

Reeve does not address that run-time data-partition allocation problem. Reeve’s FIRST COME FIRST SERVE (FCFS) “repeatability” arises because there are more iteration-space tiles than processors and the runtime issues tiles to requesting processors/threads First Come First Serve (FCFS). (Reeve, 4:1–7 and 39:3–6). That does not make Reeve reasonably pertinent to the ‘582 Patent’s control-file-driven, run-time allocation of data partitions of a preexisting input file—particularly where the ‘582 Patent’s repeated allocation is expressly framed as allocating an “input partition that has not been allocated yet.” (‘582 Patent, 4:1–4).

As Dr. Abadi explains:

Because the two references solve different problems using different abstractions, a POSITA would not naturally treat them as belonging to the same field of endeavor. The fact that both references use generic terms such as partition, tile, or processor does not place them in the same field, because the technical meaning and use of those concepts differs. In the ‘582 Patent, a partition is a well defined range of a file’s records described by byte or track offsets and allocated to subtasks without reading file contents during partition definition. In Reeve, a tile is a subset of an iteration space defined by index bounds and tile size and scheduled for execution as part of a loop based

computation. (582, 3:36–41; 582, 3:47–53; Reeve, 32:10–16; Reeve, 39:15–17). EX2001, ¶79

Therefore, a POSITA confronting the ‘582 Patent’s run-time scheduling problem would look to work-queue and partition-allocation techniques in data/file processing environments, not to compiler transformations that generate tiled instruction sequences for loop iteration spaces. That is exactly the kind of “abstract problem” reframing that *Sanofi-Aventis* rejects.

3. The Office Action lacks a legally sufficient reason to combine Iyer with Reeve’s GRAB strategy

The Office Action concedes that Iyer “does not explicitly teach” the claim 1 limitation requiring “repeating step (c) ... each with another unprocessed partition on a first-come/first-served basis,” and it therefore turns to Reeve’s GRAB strategy to supply that missing feature. (Office Action, p. 4). It then states—without more—that it would have been obvious to “modify Iyer’s processors 1–P to implement Reeve’s GRAB strategy to improve efficiency of Iyer’s sort method.” (Office Action, p. 5). That stated reason to combine is deficient for at least three independent reasons.

First, the asserted motivation is conclusory. “Improve efficiency” is a generic aspiration, not an explanation of what specific inefficiency exists in Iyer’s disclosed system and why GRAB, in particular, would have been selected by a POSITA as the

solution. The Office Action does not identify a concrete problem in Iyer’s parallel sort/merge flow that remains unresolved by Iyer’s own design principles, nor does it explain why a POSITA would depart from Iyer’s approach to workload division to adopt Reeve’s different scheduling paradigm. (Office Action, p. 5).

Second, the Office Action does not grapple with the fact that implementing GRAB in Iyer is not a plug-in change, but a fundamental redesign of Iyer’s workload model. The “repeat step (c) ... with another unprocessed partition” limitation is not satisfied merely because a system uses multiple processors; it presupposes that there exists a pool of multiple unprocessed partitions and that processors can acquire “another unprocessed partition” on a first-come/first-served basis. (Office Action, p. 4). Reeve’s GRAB strategy fits that premise because Reeve intentionally creates “more tiles (partitions) than processors” and issues tiles to requesting processors First Come First Serve (FCFS). (Office Action, p. 5). Iyer’s disclosed sorting embodiment, by contrast, is structured around pre-balancing work, not dynamically reassigning it; it does not disclose the over-subscription and shared-pool architecture that makes GRAB meaningful. The Office Action never explains why a POSITA would restructure Iyer into an over-subscribed, pool-based work-queue regime—creating additional partitions beyond the number of processors, maintaining global availability state, enabling repeated allocation—merely to achieve a generalized “efficiency” benefit. (Office Action, p. 5). Stated differently, the rationale does not

“bridge the gap” between the system Iyer actually teaches and the system GRAB presupposes.

Third, the Office Action’s proffered rationale depends on importing a feature from a reference that is not reasonably selected for Iyer’s context. Under *Sanofi-Aventis Deutschland GmbH v. Mylan Pharms. Inc.*, the scope of relevant prior art is bounded by the analogous-art inquiry; a reference outside that scope does not logically commend itself to a POSITA confronting the challenged patent’s problem and therefore cannot supply a missing limitation via an after-the-fact “efficiency” assertion. *Sanofi-Aventis*, 66 F.4th 1373, 1380–82. Here, Reeve’s GRAB strategy arises from a compile-time iteration-space tiling and runtime thread-scheduling context, not from the run-time data partition allocation context addressed by the ‘582 Patent. As a result, even if GRAB can be described abstractly as “first-come/first-served,” the Office Action must still articulate why a POSITA working in Iyer’s parallel sort/merge framework would have been motivated to consult and adopt a compiler/runtime tiling scheduling policy, and why doing so would have been expected to improve Iyer’s system in a predictable way. The Office Action does not do that; it instead relies on the label “efficiency” as a substitute for a reasoned explanation. (Office Action, p. 5).

For these reasons, the Office Action has not provided an articulated, non-hindsight reason to combine Iyer with Reeve’s GRAB strategy to meet claim 1’s

“thereafter repeating ... with another unprocessed partition” First Come First Serve (FCFS) limitation.

D. Dependent Claims

In addition to being allowable for the reasons set forth above with respect to independent claim 1, dependent claims 5, 10, and 11 are independently patentable for at least the additional limitations they introduce. Each of these claims narrows the scope of claim 1 by reciting specific output and process-type requirements that further constrain the claimed run-time, partition-based processing architecture. The Office Action does not carry its prima facie burden to show that the proposed Iyer plus Reeve combination teaches or suggests these additional limitations. As explained below, even assuming arguendo that the Examiner’s mapping of claim 1 were sufficient, the rejection still fails because the cited references do not disclose (i) claim 5’s requirement that the step (e) output is an accumulation of output records in an arbitrary order, (ii) claim 10’s requirement that the computer-executable process is itself a report-creating process (as opposed to merely emitting post-run diagnostics), or (iii) claim 11’s requirement that the computer-executable process includes a database query (as opposed to generic disk access). Accordingly, claims 5, 10, and 11 are patentable both by virtue of their dependency from patentable claim 1 and for these additional, independent reasons.

1. Iyer + Reeve combination does not teach “an accumulation of output records from said subtasks in an arbitrary order.”

For claim 5, the Office Action asserts that Iyer teaches “an accumulation of output records from said subtasks in an arbitrary order” because “waiting” in Iyer purportedly “confirms” that processors complete their subtasks “independently and at varying times,” which the Office Action equates to “arbitrary order.” (Office Action, p. 6 (citing Iyer, 11:23–30)). That showing does not meet the prima facie burden for at least three independent reasons.

First, the limitation in claim 5 is directed to the ordering characteristic of the output in step (e), not the relative timing of when subtasks finish. Claim 5 requires that “the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.” The Office Action’s reasoning instead treats “waiting” and “varying times” as a proxy for “arbitrary order.” (Office Action, p. 6; Iyer, 11:23–30). But completion time is not output order. Parallel tasks can complete in different sequences while still producing an output that is assembled in a defined order, especially where a subsequent step collects results according to a prescribed ordering rule. “Asynchronous completion” is therefore not a disclosure that the output itself is an accumulation of records in “an arbitrary order.” (Office Action, p. 6; Iyer, 11:23–30).

Second, Iyer’s cited “waiting” is a synchronization and dependency mechanism, not an output-format teaching. The Office Action relies on the “waiting” discussion at Iyer, 11:23–30, to infer that processors operate independently. (Office Action, p. 6; Iyer, 11:23–30). But “wait” in a parallel algorithm is typically a barrier that ensures correctness, such as ensuring prerequisite work has completed or ensuring required information has been received before proceeding. That is exactly the opposite of an “arbitrary order” output disclosure. A waiting step indicates that certain operations cannot proceed until other operations complete, which reflects structured coordination, not an instruction that output records may be accumulated without order. (Office Action, p. 6; Iyer, 11:23–30).

Third, the Office Action’s “waiting implies arbitrary order” inference cannot be reconciled with Iyer’s own output formation teaching. Iyer’s sorting embodiment is expressly directed to producing a correctly sorted final output, and it describes combining processor outputs in a defined way: “each of the P processors merges the lists in its partition to form P sorted lists” and then “the lists are then simply strung together to form the final sorted lists.” (Iyer, 3:45–49). That is an order-constrained output formation rule. “Stringing together” to form the “final sorted lists” presupposes a prescribed concatenation order that preserves global sortedness. (Iyer, 3:45–49). An “accumulation ... in an arbitrary order” is inconsistent with the ordered

merge and ordered concatenation required to yield the “final sorted lists” that Iyer teaches. (Iyer, 3:45–49).

Accordingly, even accepting the Office Action’s premise that Iyer includes “waiting” and that processors may complete intermediate subtasks at different times, that does not establish that the step (e) output is “an accumulation of output records ... in an arbitrary order” as claim 5 requires.

2. Iyer + Reeve combination does not teach (or suggest) the claimed “report-creating process.”

Claim 10 requires: “the method defined in claim 1 wherein said computer-executable process is a report-creating process.” This limitation is directed to the nature of the computer-executable process being executed under the claim 1 framework (the process that is partitioned into subtasks and whose outputs are combined in step (e)). It is not satisfied by showing that a program includes a post-run routine that prints logs, statistics, or diagnostic reports about how the program executed.

The Office Action maps claim 10 to Reeve’s disclosure that “pr_end is executed once when the program finished. In addition to general cleanup, it produces statistics and reports.” (Office Action, 6 citing Reeve 30:13-14). But that passage does not teach that the “computer-executable process” is a “report-creating process” within the meaning of claim 10. It teaches only that, after completing the actual work

of the program, a termination routine performs cleanup and emits execution metrics and reports. Those are reports about the execution of the application, not reports that constitute the purpose or output of the application itself.

That distinction is legally and technically material. Many applications, including sorting utilities, compilers, and distributed jobs, generate end-of-run statistics, logs, and summaries. That does not convert the underlying application into a “report-creating process.” Under the Office Action’s reasoning, virtually any program that logs runtime statistics would satisfy claim 10, effectively reading the “report-creating process” limitation out of the claim. Claim 10 requires that the job being executed, partitioned, and combined under claim 1 is itself a report-creating process, meaning a process whose substantive output is a report. Reeve’s cited “pr_end” disclosure does not teach that. It describes a cleanup and telemetry step executed once the program has finished doing its actual work.

Accordingly, the Examiner has not identified a teaching in Reeve (or in the proposed Iyer + Reeve combination) that the claimed computer-executable process is a report-creating process, as opposed to merely having a post-execution routine that produces diagnostic statistics and reports about the run. The prima facie case therefore fails for claim 10.

3. Iyer + Reeve combination does not teach (or suggest) the claimed “computer-executable process includes a database query.”

Claim 11 requires that “said computer-executable process includes a database query.” This is not a trivial label. A “database query,” under its ordinary meaning, connotes execution of a query operation against a database structure (for example, retrieving records in response to a query statement or predicates), not merely reading data from storage or moving data over an I/O bus.

The Office Action’s entire claim 11 showing rests on a single citation: “With respect to claim 11, Iyer teaches the method defined in claim 1 wherein said computer-executable process includes a database query (4:11–16).” (Office Action, p. 7). The cited passage in Iyer, however, does not describe any query operation. It describes system hardware architecture, namely a multiprocessor system in which processors access main storage through an I/O bus, and data may be stored in direct access storage devices (DASD) coupled to the bus. (Iyer, 4:11–16). That disclosure is about the existence of storage devices and connectivity. It does not disclose querying a database, issuing a query, evaluating query predicates, accessing records through a database management system, or otherwise performing a “database query” as claim 11 requires.

In effect, the Office Action appears to treat the presence of DASD and storage access as a proxy for a “database query.” But that is not what claim 11 says. Almost

any computer process involves storage access somewhere in the system, and equating “data stored on disk” with a “database query” would read the word “database” and the word “query” out of the claim. The prima facie burden requires the Examiner to identify where the reference teaches the specific limitation as claimed, not to rely on a generalized observation that data can reside on storage. (Office Action, p. 7; Iyer, 4:11–16).

This deficiency is particularly consequential here because Iyer is directed to parallel sorting and merge operations, not to database retrieval operations. To the extent the Examiner is asserting that database querying is implicit because data is stored on DASD, that is an unsupported leap. A disk-resident list is not necessarily a “database,” and reading list elements from DASD is not necessarily a “database query.” Without an identified disclosure of an actual query operation, the Office Action has not met its burden to show that claim 11’s narrowing limitation is taught by Iyer. (Office Action, p. 7; Iyer, 4:11–16).

For these reasons, the Office Action’s citation to Iyer, 4:11–16 is insufficient to establish the claim 11 “database query” limitation, and the Iyer plus Reeve rejection fails at least on this point for claim 11.

E. Ground 1 Fails to Establish a Prima Facie Case: Iyer Teaches One-Partition-Per-Processor, Reeve Is Non-Analogous, and Dependent Claims Remain Unmet

For the reasons set forth above, the Office Action has not established a prima facie case that claims 1 through 14 would have been obvious over Iyer in view of Reeve. The Office Action's theory hinges on using Reeve's GRAB strategy to supply the missing claim 1 requirement that processors "thereafter repeat" processing "each with another unprocessed partition" on a first-come/first-served basis, yet Iyer's disclosed sorting embodiment teaches the opposite workload model, namely selecting the number of lists and merge partitions to match the number of available processors so that "no processor load imbalance occurs." (Iyer, 2:45–49, 4:25–33 and 4:36–40). The Office Action's premise that step (c) "may be repeated" because Reeve has "more tiles than processors" does not transfer to Iyer's one-partition-per-processor architecture, and importing GRAB would require re-architecting Iyer into an over-subscribed, pool-based work-allocation regime that Iyer neither discloses nor suggests. (Office Action, p. 4-5; Iyer, 4:25–33 and 4:36–40). The Office Action also offers only a conclusory "improve efficiency" rationale for this substantial redesign, without identifying any concrete deficiency in Iyer's approach that GRAB would remedy, and without bridging the gap between Iyer's stated design principle of static pre-balancing and the claimed invention's subsequent allocation of

unallocated partitions. (Office Action, p. 5; Iyer, 2:45–49; ‘582 Patent, 3:47–50 and 4:1–4).

Independently, Reeve is not analogous art to the ‘582 Patent’s run-time, data-partition allocation problem and therefore cannot permissibly be used as the gap-filler for the missing FCFS “repeat” limitation. The ‘582 Patent addresses run-time allocation and processing of data partitions of a preexisting input file, including repeated allocation of partitions “that ha[ve] not been allocated yet,” whereas Reeve addresses compile-time iteration-space tiling and runtime scheduling of iteration-space tiles to threads. (‘582 Patent, 1:46–55, 3:47–50 and 4:1–4; Reeve, 1:63–2:13, 4:1–7 and 39:3–6). Under *Sanofi-Aventis*, Reeve’s iteration-space tiling framework does not logically commend itself to a POSITA confronting the ‘582 Patent’s particular problem, and it cannot supply the missing limitation through an after-the-fact “efficiency” assertion. *Sanofi-Aventis*, 66 F.4th 1373, 1380–82; (Office Action, p. 4-5).

Finally, even assuming *arguendo* that the Office Action had adequately mapped independent claim 1, the rejection still fails as to dependent claims 5, 10, and 11 because the Office Action does not show that the cited references teach the additional narrowing limitations those claims introduce. The Office Action’s claim 5 “arbitrary order” theory improperly equates synchronization and completion timing with an output-formation rule, despite Iyer’s express ordered merge-and-

concatenation teaching for producing the final sorted output. (Office Action, p. 6; Iyer, 11:23–30 and 3:45–49). The Office Action’s claim 10 mapping relies on post-execution telemetry reports about how a program ran, not that the executed computer-executable process is itself report-creating. (Iyer, 8:17–19). And the Office Action’s claim 11 mapping improperly equates generic storage architecture (DASD and I/O bus access) with a “database query,” without identifying any disclosure of query execution against a database. (Office Action, p. 7; Iyer, 4:11–16). Accordingly, claims 1 through 14 remain patentable, and Ground 1 should be withdrawn.

VI. GROUND 2: CLAIMS 1-14 ARE REJECTED UNDER PRE-AIA 35 U.S.C. 103(a) AS BEING UNPATENTABLE OVER REEVE

The Office Action separately rejects claims 1–14 as obvious over Reeve alone, asserting that Reeve discloses the claim 1 framework, including a “preexisting input file,” “logical subdivision ... without reading,” distribution of partition descriptions, First Come First Serve (FCFS) repetition, and an output combining subtask outputs. (Office Action, pp. 7–8). This rejection rests on an over-abstract characterization of Reeve that substitutes Reeve’s compile-time iteration-space tiling and runtime tile-to-thread scheduling paradigm for the ‘582 Patent’s run-time, data-partition allocation architecture directed to processing records of a preexisting input file. (‘582 Patent, 3:30–33, 3:35–41, 3:47–50 and 4:1–4; Reeve, 31:7–13, 32:10–16 and 39:4–6). As shown below, when the claim language is given its ordinary meaning in context under *Phillips*, Reeve does not teach (i) the core file-allocation and file-

record partitioning limitations of claim 1, (ii) the claim’s FCFS “repeat ... with another unprocessed partition” mechanism as used in the ‘582 Patent, or (iii) the claimed output-generation model in which subtask outputs are produced and then combined into an output equivalent to the original process. (Office Action, pp. 7–8; ‘582 Patent, 4:5–9 and 4:57–60; Reeve, 39:15–18).

In addition, the Reeve-only ground fails at the threshold because Reeve is non-analogous art to the particular problem addressed by the ‘582 Patent. The Federal Circuit has confirmed that a reference must be analogous to be considered within the prior art for an obviousness analysis, and the analogous-art inquiry must be tethered to the inventor’s particular problem rather than generalized notions of “parallelism.” *Circuit Check, Inc. v. QXQ Inc.*, 795 F.3d 1331, 1335, 1338 (Fed. Cir. 2015). Reeve’s focus on compile-time tiling and scheduling of iteration-space tiles would not logically commend itself to a POSITA seeking to solve the ‘582 Patent’s run-time data-partition allocation problem for file-processing workloads. (‘582 Patent, 1:46–55, 3:47–50 and 4:1–4; Reeve, 1:63–2:13 and 32:10–16). Finally, even assuming arguendo that claim 1 could be met, dependent claims 3, 5, 10, and 11 recite additional limitations that the Office Action does not show are taught or suggested by Reeve.

A. Reeve does not teach the core “file-partition” limitations of claim 1 (and the Office Action’s mapping relies on an over-abstract characterization of Reeve)

The Office Action asserts that Reeve discloses claim 1’s “preexisting input file,” “logical subdivision ... without reading,” distribution of “descriptions of all ... partitions,” First Come First Serve (FCFS) repetition, and an “output combining all of the subtask outputs.” (Office Action, pp. 7–8). That mapping conflates the claim’s file-partition/job-processing architecture with Reeve’s compiler/runtime tiling and tile-to-thread scheduling framework. When the claim terms are given their ordinary meaning in context (*Phillips*), Reeve does not disclose the required file-partition mechanisms.

1. Different object being partitioned: iteration space tiles vs partitions of a preexisting file’s records (file allocation + record boundaries)

Claim 1 is anchored to data-domain partitioning: it requires “automatically determining ... file allocation” and “logically subdividing records in the preexisting input file ... into partitions.” (Office Action, pp. 7–8). The ‘582 Patent confirms that the claimed partitions are file-defined ranges derived from OS-level file information, not abstract compute regions. The split step uses OS information about “size and location of the input file” to create a “control file ... specifying a logical partition of the input,” and it explains that a “logical partition ... is a well-defined part of the input.” (‘582 Patent, 3:30–33 and 3:35–36). The ‘582 Patent then gives a concrete

definition of partitions: a “very simple way” is to define partitions as “consecutive ranges on the input ... ranging from one relative byte address to another ... or from one relative track address to another.” (‘582 Patent, 3:37–41).

Reeve’s “tiles,” by contrast, are compute-domain subregions of an iteration space—not partitions of a file. Reeve’s directive format makes that explicit: it defines an “AFFINITY REGION (<index and bounds>)” with “tilesize” and a selectable “strategy={GRAB, MOD, WAVE, SLICE}.” (Reeve, 32:10–16). That is an index/bounds-driven decomposition used to execute a parallel construct; it is not “file allocation” nor a “logical subdivision of records in a preexisting input file” into partitions defined by file offsets/record boundaries. (Office Action, pp. 7–8; Reeve, 32:10–16; ‘582 Patent, 3:35–41).

As Dr. Abadi explains:

This is not a semantic distinction. It goes to the mechanism and the object being partitioned. Claim 1 requires partitions of a preexisting file’s records based on file allocation information. Reeve describes tiling of an iteration space based on indices, bounds, and a tile size for scheduling computation. (Reeve, 32:10–16; 582, 3:30–33; 582, 3:37–41.) A POSITA would understand these as different kinds of partitioning with different inputs, different artifacts, and different runtime consequences.
EX2001, ¶162

2. Different partitioning target: the ‘582 Patent divides the data, while Reeve divides the task

A key architectural difference is what gets partitioned. The ‘582 Patent partitions the data set, meaning a “preexisting input file,” into logical partitions of records, which are data ranges. The specification explains that the split step uses OS information about the “size and location of the input file” to create a “control file ... specifying a logical partition of the input,” and it describes partitions as “consecutive ranges on the input” defined by relative byte or track addresses. (‘582 Patent, 3:30–33 and 3:37–41). In other words, the unit of parallelism is a slice of the input data.

Reeve, by contrast, partitions instructions. It breaks a parallel construct into tiles of an iteration space and then schedules those tiles to threads. Reeve’s directive syntax is explicit that tiling is defined by an “AFFINITY REGION (<index and bounds>)” with a “tilesize” and a chosen strategy such as “GRAB,” meaning the partitioning is over an index and bounds defined iteration space, not over a file’s record ranges. (Reeve, 32:10–16). Reeve’s runtime description likewise frames the system as influencing execution of a parallel construct by choosing the number of threads and relying on the OS scheduler for load balancing, which again reinforces that Reeve divides and schedules instructions, not file data. (Reeve, 31:7–13).

This data versus task distinction matters for the Office Action’s Reeve-alone mapping. Claim 1’s architecture assumes processing is performed on partitions of

the input file's data and that completion is achieved by covering all unprocessed data partitions; the '582 Patent even contemplates over subscription of data partitions relative to helping systems to enable dynamic allocation. ('582 Patent, 3:47–50 and 4:1–4). Reeve's tiling, however, is a mechanism for decomposing and scheduling computational iterations, that is, tiles, among threads during execution of a parallel program construct, not for dividing a preexisting file's records into partitions and ensuring that all data partitions are processed. (Reeve, 31:7–13; Reeve, 32:10–16; Office Action, pp. 7–8).

3. Different meaning of “First Come First Serve (FCFS)”: FCFS tile-to-thread assignment vs FCFS acquisition of another unprocessed/unallocated file partition after completion

The Office Action relies on Reeve's GRAB strategy to satisfy claim 1's requirement that subtask processors “thereafter” repeat processing “each with another unprocessed partition on a first-come/first-served basis.” (Office Action, pp. 7–8). But the term “first-come/first-served” is being used in two materially different ways. Reeve uses FCFS as a local tile assignment policy within a thread-based runtime tiling framework, whereas the '582 Patent uses FCFS as an application-level work-claiming mechanism that repeatedly allocates unallocated file-data partitions among subtasks until all partitions are processed.

a) Reeve's FCFS is a scheduling rule for tiles in an iteration space, assigned to threads

Reeve's FCFS statement appears in its description of the GRAB strategy: "In the grab strategy, tiles are assigned to threads with a pure first come, first serve algorithm." (Reeve, 39:4–6). In context, that statement presupposes Reeve's tiling model, in which the relevant "work items" are tiles and the relevant "workers" are threads. Reeve's runtime explanation reinforces this context by describing that "Runtime Library 66" influences resource allocation for a "parallel construct" by choosing the number of "threads," including "more or less threads than available processors," and that much of load balancing is delegated to the OS scheduler. (Reeve, 31:7–13). So, Reeve's FCFS is a rule about the order in which tiles are handed out to threads during execution of a tiled parallel construct, within a runtime and OS scheduling environment. (Reeve, 31:7–13; Reeve, 39:4–6).

That type of FCFS is not inherently a mechanism for repeated allocation of "another unprocessed partition" of a preexisting input file as recited in claim 1. As Dr. Abadi explains, "Reeve's FCFS tile assignment to threads within an iteration space tiling framework does not teach or suggest a pool of unallocated file data partitions that are repeatedly allocated to independent subtasks operating on partitions of a preexisting input file." EX2001, ¶170. Therefore, Reeve is simply a policy for assigning pre-defined compute tiles to threads as they become available

not the “thereafter repeating” limitation in the file partition sense used by the ‘582 Patent.

b) Claim 1’s FCFS is a repeated allocation mechanism over a pool of unprocessed or unallocated data partitions

By contrast, claim 1’s FCFS limitation has an explicit temporal and operational structure. It is not merely “FCFS ordering.” It requires that the system “thereafter” repeats processing “each with another unprocessed partition” on an FCFS basis. (Office Action, pp. 7–8). The ‘582 Patent explains the intended operating model and the predicate conditions that make this limitation meaningful. Specifically, the ‘582 Patent teaches that “the number of logical input partitions will largely exceed the number of helping systems.” (’582 Patent, 3:47–50). It then describes the operative mechanism: “Each such Sub Task then repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (’582, 4:1–4).

Two features are critical here. First, FCFS is tied to a pool of partitions that have “not been allocated yet,” meaning the system must track allocation status and support claiming of unallocated partitions. (’582, 4:1–4). Second, FCFS is tied to repetition after completion of prior work, because partitions outnumber helping systems and subtasks repeatedly allocate and process successive partitions until none remain. (’582 Patent, 3:47–50 and 4:1–4). In short, FCFS in the ‘582 Patent is an

application-level allocation policy that governs which subtask gets the next unallocated data partition when it requests additional work after finishing its prior partition. (‘582, 4:1–4).

c) The predicate conditions for claim 1’s FCFS “repeat” limitation are not shown in Reeve

The Office Action’s reliance on Reeve’s FCFS tile assignment does not establish the predicate conditions the ‘582 Patent associates with claim 1’s FCFS repetition. The ‘582 Patent’s FCFS repetition depends on a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file (‘582 Patent, 3:47–50 and 4:1–4). Reeve’s disclosure, in contrast, describes FCFS tile assignment to threads within a runtime tiling framework and discusses load balancing via thread scheduling and OS scheduling. (Reeve, 31:7–13 and 39:4–6). Even if tiles can be numerous, Reeve does not disclose the same type of “not allocated yet” partition pool and repeated application-level allocation to independent subtasks operating on partitions of file data, as described in the ‘582 Patent. (‘582 Patent, 4:1–4).

Accordingly, the mere presence of the words “first come, first served” in Reeve does not mean Reeve teaches claim 1’s “thereafter repeating ... each with another unprocessed partition” limitation. The claim requires FCFS in a specific context, namely repeated allocation of unallocated data partitions among subtasks

until completion, which is the load balancing mechanism the ‘582 Patent actually describes.

d) Reeve’s “tiles” are task fragments, while the ‘582 partitions are data fragments

Reeve’s FCFS is about handing out tiles, which are fragments of a computation. A “tile” in Reeve is defined by an “AFFINITY REGION (<index and bounds>)” and a “tilesize,” meaning the tile is a bounded subregion of an iteration space that represents a chunk of a task. (Reeve, 32:10–16). Reeve then applies FCFS to that work decomposition by assigning those tiles to threads: “tiles are assigned to threads with a pure first come, first serve algorithm.” (Reeve, 39:4–6). In other words, in Reeve, the thing being queued and assigned is a portion of a task, and the recipient is a thread executing a parallel construct. (Reeve, 31:7–13 and 39:4–6).

Claim 1 of the ‘582 Patent differentiates between “a preexisting input file” and “subtasks,” and the Board has made clear that a mapping that conflates these distinct constructs fails to meet the claim language. The ‘582 Patent’s FCFS is about allocating data partitions of a preexisting input file, which are fragments of the input data itself. The ‘582 Patent defines partitions as well-defined parts of a “preexisting input file,” for example consecutive file ranges specified by relative byte addresses or relative track addresses. (‘582 Patent, 3:35–41). Those partitions are not “pieces of a loop” as disclosed in Reeve (which at best correspond to subtasks). The

claimed partitions are instead pieces of a file that already exists, and each subtask processes one data partition at a time. The load balancing mechanism depends on the fact that “the number of logical input partitions will largely exceed the number of helping systems,” so when a subtask finishes, it “repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (‘582 Patent, 3:47–50; ‘582, 4:1–4). In other words, in the ‘582 Patent, the thing being queued and claimed is an unallocated segment of a preexisting input file, and the recipient is a subtask running on a helping system that will read and process that specific file range. (‘582 Patent, 3:35–41 and 4:1–4).

Accordingly, Reeve partitions the task (rather than the input file) by splitting an iteration space into tiles and scheduling those tiles to threads under a runtime policy. (Reeve, 31:7–13, 32:10–16 and 39:4–6). The ‘582 Patent partitions the data by splitting a preexisting input file into file-range partitions and repeatedly allocating unallocated data partitions to subtasks until all data partitions have been processed, which is the context in which FCFS is used in claim 1.

4. Output generation in Reeve is not the “subtask outputs combined into an output” required by claim 1, as described in the ‘582 Patent

Claim 1 requires “generating an output ... combining all of the subtask outputs.” (Office Action, pp. 7–8). In the ‘582 Patent, that limitation has a concrete file-processing meaning. Each Sub Task operates on a partition of a preexisting input

file and produces its own output, and the system then performs further processing to create an output equivalent to the original process. For example, the ‘582 Patent explains that “the output of the various Sub Tasks has to be further processed to create an output that is equivalent to the one expected from the original process,” and that this is why “each Sub Task creates its output on dedicated output files.” (‘582 Patent, 4:5–9). The ‘582 Patent gives the sort example where Sub Task 201 creates Output1, Sub Task 202 creates Output2, and Sub Task 103 creates Output3. (‘582 Patent, 4:7–9). The specification then describes a Merge Step that is scheduled after the Sub Tasks complete, and states that the Merge Step’s function is “to merge the outputs of the various Sub Tasks into the result output file or files.” (‘582 Patent, 3:14–20 and 4:57–60).

That is materially different from Reeve’s disclosure. Reeve is directed to runtime scheduling of tiles to threads during execution of a parallel construct, not a pipeline in which each worker produces a discrete subtask output that is later merged into a result output file. Reeve describes GRAB as a rule where “tiles are assigned to threads with a pure first come, first serve algorithm.” (Reeve, 39:4–6). Reeve’s discussion is framed around distributing computation over an iteration space and maintaining thread affinity, and it makes clear that the runtime “looks only at the iteration space, and not at the data space.” (Reeve, 39:15–18). In that setting, the “outputs” of the work are typically the results of executing code over portions of an

iteration space, often manifested as updates to shared program state, not as separate per-thread output files that must be merged to produce a result output file. (Reeve, 39:15–18).

Accordingly, the Office Action’s reliance on Reeve to satisfy claim 1’s output limitation overlooks the claim’s file-processing context and the ‘582 Patent’s description of how the output is generated. The ‘582 Patent’s “output combining all of the subtask outputs” is implemented by Sub Tasks producing dedicated output files and a subsequent merge that combines those outputs into a result output file or files. (‘582 Patent, 4:5–9 and 4:57–60). Reeve’s tile-to-thread scheduling and iteration-space execution does not disclose that discrete “subtask output” and “merge into result output file” model required by claim 1.

B. The Office Action’s Reeve-alone mapping is structurally flawed because it substitutes generic “parallelism” for the specific file-partition load-balancing architecture of claim 1

The Office Action’s Reeve-alone rejection depends on treating Reeve as if it were directed to the same kind of partitioning and load balancing claimed in the ‘582 Patent. (Office Action, pp. 7–8). But claim 1, read in context under *Phillips*, is not directed to “parallel execution” in the abstract. Claim 1 recites a specific architecture for accelerating a linear file-processing job by dividing a preexisting input file into partitions and dynamically allocating those partitions among subtasks, then producing a combined output equivalent to the original process. The ‘582 Patent

describes this architecture in concrete, file-centric terms that are not present in Reeve.

First, the claim is anchored to data partitioning of a preexisting input file. The ‘582 Patent explains that the split step uses OS information about the “size and location of the input file” to create a “control file ... specifying a logical partition of the input.” (‘582 Patent, 3:30–33). It further explains that a “logical partition ... is a well-defined part of the input,” and provides a “very simple way” to define partitions as “consecutive ranges on the input,” such as from one relative byte address to another or from one relative track address to another. (‘582 Patent, 3:35–41). This is not a generic “divide the work” concept. It is a particular mechanism for subdividing an existing data object, the input file, into file-range partitions.

Second, the claim requires load balancing by repeated allocation of unallocated file partitions. The ‘582 Patent expressly contemplates over-subscription: “the number of logical input partitions will largely exceed the number of helping systems.” (‘582 Patent, 3:47–50). It then describes the operative load-balancing mechanism: “Each such Sub Task then repeatedly tries to allocate for itself and then process, an input partition that has not been allocated yet.” (‘582 Patent, 4:1–4). This is an application-level allocation scheme over a pool of unallocated partitions. It is not merely that “multiple units exist” or that “processors may take turns.” The system must support a partition description framework that involves

allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file, until all partitions are processed. (‘582 Patent, 3:47–50 and 4:1–4).

Third, claim 1’s output limitation is similarly grounded in this file-partition pipeline. The ‘582 Patent explains that the outputs of the Sub Tasks must be “further processed to create an output that is equivalent to the one expected from the original process,” and that this is why each Sub Task can create output on dedicated output files, followed by a merge that “merge[s] the outputs of the various Sub Tasks into the result output file or files.” (‘582 Patent, 4:5–9 and 4:57–60). Thus, the claim’s “output combining all of the subtask outputs” corresponds to a specific workflow involving partition-level outputs and combining or merging into the final result.

Reeve is structurally different. Reeve describes a compiler/runtime framework for executing “parallel constructs” using threads, with the runtime influencing resource allocation by “choosing the number of threads” and delegating load balancing to the OS scheduler. (Reeve, 31:7–13). Reeve’s tiling is defined through an “AFFINITY REGION (<index and bounds>)” with “tilesize” and a chosen strategy (including GRAB), which is an iteration-space decomposition for scheduling computation. (Reeve, 32:10–16). GRAB then assigns “tiles” to “threads” FCFS. (Reeve, 39:4–6). These are mechanisms for dividing and scheduling a computation over an index/bounds-defined iteration space, not mechanisms for

partitioning a preexisting input file into file-range partitions derived from file allocation metadata and repeatedly allocating unallocated data partitions among subtasks on helping systems.

Dr. Abadi explains the structural mismatch between the '582 Patent and Reeve using a “bread and French toast” analogy. (see §IV. B). The '582 Patent is likened to starting with a preexisting physical object, such as a loaf of bread already on the counter, and accelerating the work by slicing that loaf into many slices using high-level information about the loaf, such as its length, without consuming or inspecting the bread itself. A coordinator can define a pool of slices, and multiple helpers then repeatedly claim the next unclaimed slice, process it, and place the resulting outputs into separate trays. Once all slices are processed, the outputs are combined into a final basket equivalent to what one person would have produced by processing the whole loaf end-to-end. This mirrors the '582 Patent's file-centric architecture: an existing input file is subdivided into file-range partitions based on allocation and offsets, partitions are dynamically allocated from a pool of unallocated partitions to subtasks, and the subtask outputs are later combined into an output equivalent to the original process.

Reeve, by contrast, is likened to parallelizing a recipe rather than slicing an existing loaf. Instead of dividing a preexisting object, one divides the cooking procedure into steps with dependencies, and parallelizes execution in a pipeline

fashion (e.g., one person dips pieces while another fries). This corresponds to Reeve's approach of partitioning an iteration space into tiles defined by index/bounds and tile-size directives, then scheduling those tiles to threads under a strategy such as GRAB. (EX2001, ¶¶182–183.) The analogy underscores why the Office Action's Reeve-alone mapping is structurally flawed: claim 1 requires partitioning an existing data object (the preexisting input file) into partitions tied to file allocation and offsets, dynamically allocating those partitions among workers, and combining subtask outputs into a final output, whereas Reeve partitions execution of a computation into iteration-space tiles and schedules those tiles during execution. (EX2001, ¶¶182–183.)

Accordingly, the Office Action's Reeve-alone mapping is flawed at the architectural level. It treats Reeve's tile-to-thread scheduling, which partitions a computation, as if it disclosed the '582 Patent's file-partition allocation system, which partitions a preexisting input file's data and repeatedly allocates unallocated partitions until completion, producing subtask outputs that are combined into a final output equivalent to the original process.

C. The Reeve-alone rejection fails to establish a prima facie case because Reeve is non-analogous art, and non-analogous references are outside the “scope and content of the prior art” for an obviousness analysis

The Office Action rejects claims 1–14 as obvious over Reeve alone. (Office Action, pp. 7–8). But the Federal Circuit has repeatedly treated the analogous-art inquiry as a threshold predicate to any prima facie obviousness case. In *Circuit Check, Inc. v. QXQ Inc.*, the Court explained that “[t]o be considered within the prior art for purposes of the obviousness analysis, a reference must be analogous.” *Circuit Check, Inc. v. QXQ Inc.*, 795 F.3d 1331, 1335 (Fed. Cir. 2015). The Court further reiterated the two familiar tests: prior art is analogous only if it is “from the same field of endeavor” or “reasonably pertinent to the particular problem the inventor is trying to solve.” *Id.* at 1335. Because a non-analogous reference is not “within the scope of the prior art” for the § 103 inquiry, reliance on a non-analogous reference cannot, as a matter of law, supply the foundation for a prima facie case.

1. *Circuit Check* confirms the pertinent question is not whether a concept is generally known, but whether the POSITA would look to that “particular art” to solve “the particular problem at hand”

Circuit Check is especially helpful here because it rejects the kind of impermissible over-generalization that often drives an analogous-art error. The Court emphasized that the question is not whether a technique is within “common knowledge” or would be understood by “any layman.” Rather, “the question is

whether an inventor would look to this particular art to solve the particular problem at hand.” *Id.* at 1338. The Court put the point bluntly: just because something is within common knowledge “does not mean” it is analogous art, and an accused infringer cannot “transform all systems and methods within the common knowledge into analogous prior art simply by stating that anyone would have known of such a system or method.” *Id.* at 1338. The governing inquiry is whether the reference “logically would have commended itself to an inventor’s attention in considering his problem.” *Id.* at 1338.

Applied to this reexamination, the Office Action’s Reeve-alone mapping effectively defines the “problem” at too high a level of abstraction, namely “parallel processing” and “first-come/first-served repetition.” (Office Action, pp. 7–8). *Circuit Check* teaches that this is the wrong framing. The inquiry must be anchored to the particular problem the ‘582 Patent addresses, and then ask whether Reeve is the kind of reference that would logically commend itself to a POSITA confronting that specific problem.

2. The particular problem addressed by the ‘582 Patent is run-time allocation and processing of data partitions of a preexisting input file, not compile-time tiling of an iteration space

The Office Action characterizes Reeve as teaching a method “effecting on a preexisting input file” and maps Reeve’s “automatic tiling” and GRAB strategy to

the claim's file-partition steps. (Office Action, pp. 7–8). Respectfully, that reasoning conflates “a preexisting input file” with “subtasks,” as such terms are used in claim 1 of the ‘582 Patent. But the ‘582 Patent's field and problem are data-centric and run-time: accelerating a file-processing job that “normally reads an input file sequentially and processes its records one by one” by partitioning the data and allocating those data partitions among helping systems during execution. (‘582 Patent, 1:46–55). The ‘582 Patent then describes the load-balancing mechanism as a run-time allocation scheme over data partitions, where the number of data partitions exceeds the number of helping systems and each subtask repeatedly allocates and processes an input partition that has not yet been allocated. (‘582 Patent, 3:47–50 and 4:1–4).

Reeve, by contrast, is directed to compile-time and runtime support for tiled execution of code over an iteration space. The tiles in Reeve are task fragments defined by an “AFFINITY REGION (<index and bounds>)” and “tilesize,” and the GRAB policy assigns those tiles to threads FCFS. (Reeve, 32:10–16 and 39:3–6). That is a task-decomposition and scheduling framework for executing a parallel construct over an iteration space, not a data-partition allocation system for a preexisting input file's record partitions. (Reeve, 32:10–16 and 39:3–6).

In other words, the ‘582 Patent divides and allocates data partitions (file ranges / record subsets) at run time, whereas Reeve divides and schedules task tiles

(iteration-space fragments) to threads. Even if both can be described loosely as “parallel processing,” *Circuit Check* makes clear that this type of broad abstraction cannot substitute for the required “particular problem at hand” analysis.

3. Under *Circuit Check*, Reeve is not “reasonably pertinent” because it would not logically commend itself to a POSITA seeking to solve the ‘582 Patent’s run-time data-partition allocation problem

Under *Circuit Check*, when a reference is outside the inventor’s field of endeavor, it can qualify as analogous only if it is “reasonably pertinent,” meaning it “logically would have commended itself to an inventor’s attention in considering his problem.” *Circuit Check*, 795 F.3d 1331, 1338. The Court upheld a finding of non-analogousness where the evidence supported that a POSITA would not have looked to the “disputed prior art” when trying to solve the specific marking problem, even though the underlying concepts were familiar and widely understood. *Id.* at 1337–38.

That reasoning applies here. A POSITA attempting to solve the ‘582 Patent’s run-time problem, namely, how to keep helping systems utilized by repeatedly allocating unallocated file-data partitions until completion, would not logically consult compiler transformations and runtime tile-to-thread scheduling for loop iteration spaces. (‘582 Patent, 3:47–50 and 4:1–4). Reeve’s GRAB policy, even if described as FCFS, is disclosed as part of a tiling strategy over an iteration space

and therefore does not logically commend itself as a solution to the ‘582 Patent’s specific data-partition allocation problem. (Reeve, 39:3–6).

Circuit Check also forecloses any attempt to rescue Reeve’s analogousness by arguing that the concepts are “simple,” “familiar,” or generally known. The Court explicitly rejected the “any layman” style reasoning and held that the proper inquiry is not general knowledge but whether a POSITA would look to the particular art to solve the particular problem. *Circuit Check*, 795 F.3d 1331, 1338. Thus, even if Reeve’s FCFS scheduling could be characterized as a familiar concept, that does not make Reeve analogous to the ‘582 Patent’s field and problem.

4. Because the Office Action’s Reeve-alone ground depends on treating Reeve as within the scope of the pertinent prior art, a non-analogous Reeve cannot support a prima facie case of obviousness

The Office Action’s Reeve-alone rejection fails to establish that Reeve is properly within the “scope and content of the prior art” for the challenged claims. (Office Action, pp. 7–8). Under *Circuit Check*, a reference must be analogous to be considered within the prior art for the obviousness analysis. *Circuit Check*, 795 F.3d at 1335. If Reeve is not analogous to the ‘582 Patent’s field and particular problem, then Reeve is outside the permissible universe of prior art for this § 103 determination and cannot, as a matter of law, be used as the sole foundation for a prima facie case of obviousness. *Id.*

Accordingly, the Reeve-alone rejection should be withdrawn because it rests on an impermissible premise: that a compiler/runtime tiling reference directed to iteration-space tiles is relevant to a run-time file-data partition allocation invention merely because both involve “parallelism” and FCFS at a high level.

D. Dependent Claims

In addition to the deficiencies in the Office Action’s Reeve-only mapping of independent claim 1, the Reeve-only ground independently fails as applied to dependent claims 3, 5, 10, and 11. Each of these claims adds a further, distinct limitation that narrows the scope of claim 1, and the Examiner bears the burden of identifying where Reeve teaches or suggests those additional requirements. (Office Action, p. 9). As set forth below, the Office Action does not meet that burden. Reeve’s disclosures concern compile-time tiling and runtime scheduling of tiles to threads over an iteration space, and the Office Action’s reliance on high-level “combined results” language, execution-statistics reporting, and generalized capability statements does not establish the specific merging, record-accumulation, report-creation, or database-query limitations recited in claims 3, 5, 10, and 11.

1. Reeve does not teach the claimed “merging said subtask outputs”

Claim 3 depends from claim 1 and adds a further, distinct requirement: “further comprising the step of merging said subtask outputs in step (e).” This additional limitation is not satisfied by a generic statement that results are

“combined.” It requires a merge operation applied to subtask outputs as part of the step (e) output generation.

The Office Action maps claim 3 to Reeve by asserting that “the results from processed tiles are combined (i.e. merged)” and cites Reeve, 2:25–27, 8:9–11, and 19:38–42. (Office Action, p. 9). That mapping does not establish a prima facie case for at least three reasons.

First, the Office Action’s mapping conflates “combined” with “merged.” While claim 1 recites “combining all of the subtask outputs,” claim 3 further requires that the combining be performed “by merging said subtask outputs.” Clearly, “combining” (in independent claim 1) and “merging” (in dependent claim 3) have different meanings, otherwise claim 3 would not further narrow claim 1. The Office Action fails to acknowledge this distinction. Rather than identifying any disclosure in Reeve of a merge step performed on subtask outputs, the Office Action relies on Reeve’s cited statements that “results ... are combined.” (Reeve, 2:25–27, 8:9–11 and 19:38–42). A “combine results” concept does not require the claimed “merging” step, which connotes a specific operation that takes discrete outputs produced by subtasks and merges them into a unified output as part of step (e). Claim 3 requires more than “combining” which is already recited in independent claim 1.

Second, Reeve’s “tiles” and “threads” are not the “subtasks” and “subtask outputs” recited in claim 3, and the Office Action’s mapping does not bridge that

structural mismatch. The Reeve-only ground treats “tiles” as partitions and “threads” as subtask processors. (Office Action, p. 9). Even under that mapping, Reeve’s tile execution is not described as producing discrete “subtask outputs” that are subsequently merged. Instead, Reeve describes a runtime tiling/scheduling framework where tiles are executed as parts of a parallel construct, and “results” reflect completion of that construct. (Reeve, 2:25–27 and 19:38–42). Many tiled/threaded computations do not generate separable per-thread output artifacts at all; they produce intermediate updates to shared state during execution. The Office Action does not identify any disclosure in Reeve that the work units corresponding to “subtasks” emit discrete “subtask outputs” that are later merged.

Third, claim 3 requires a further step of merging subtask outputs “in step (e),” and the Office Action’s citations do not show an additional step beyond whatever the Office Action already relies on to satisfy claim 1’s step (e) “output combining all of the subtask outputs.” (Office Action, p. 9). In other words, even if Reeve’s “combined results” language were sufficient to meet claim 1’s step (e) combination requirement (which Patent Owner disputes), it does not automatically satisfy claim 3’s additional requirement of a separate “merging” step. The Office Action does not explain what, specifically, constitutes the distinct “merging” step in Reeve, as opposed to simply labeling the same “combined results” concept as “merge.” (Office Action, p. 9; Reeve, 2:25–27, 8:9–11 and 19:38–42).

Accordingly, the Examiner has not met the prima facie burden of showing that Reeve teaches claim 3's additional limitation requiring "the step of merging said subtask outputs" in step (e). The Reeve-only rejection of claim 3 should therefore be withdrawn.

2. Reeve does not teach the claimed "accumulation of output records from said subtasks in an arbitrary order."

Claim 5 depends from claim 1 and further requires that "the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order." This is a narrowing, output-formation limitation directed to the structure and ordering characteristics of the step (e) output, not merely to the fact that multiple work units are executed in parallel. To establish a prima facie case of obviousness based on Reeve alone, the Office Action must identify where Reeve teaches (or at least suggests) an output that is (i) composed of "output records" produced by "subtasks," and (ii) accumulated into the step (e) output "in an arbitrary order." (Office Action, p. 9).

The Office Action does not make that showing. Reeve is directed to tiling an iteration space and scheduling tiles to threads for execution of a parallel construct, and its disclosures describe how tiles are issued and executed, not how an output is formed as an accumulation of record outputs. (Reeve, 39:3–6). In that framework, the "results" of execution are the results of executing code over portions of an

iteration space, often realized as updates to program state, not “output records” generated by subtasks and accumulated into a record-based output. (Reeve, 39:15–17). The Office Action’s Reeve-only mapping fails because it does not identify any disclosure in Reeve that subtasks produce “output records” that are then accumulated into the step (e) output as records. (Office Action, p. 9).

Nor does Reeve disclose the further requirement that the step (e) output is accumulated “in an arbitrary order.” Reeve’s GRAB discussion concerns the order in which tiles are assigned to threads, stating that “tiles are assigned to threads with a pure first come, first serve algorithm.” (Reeve, 39:4–6). Even if tile issuance order were relevant to claim 1’s repetition limitation, it is not a disclosure that the output itself is an accumulation of output records “in an arbitrary order.” Claim 5 speaks to the ordering characteristic of the output produced in step (e), whereas Reeve’s FCFS language speaks to the scheduling policy for issuing work units. (Reeve, 39:4–6). A system can schedule work FCFS and still produce an ordered output without disclosing any record-level accumulation where ordering is immaterial or arbitrary.

To the extent the Office Action relies on Reeve’s generalized statements that “results” are “combined” after tile execution to support claim 5, that is likewise insufficient. (Office Action, p. 9). “Combining results” in Reeve is a high-level description of obtaining the intended overall computation after executing tiles; it does not describe an “accumulation of output records” and it does not teach that any

such accumulation occurs “in an arbitrary order.” (Reeve, 19:38–42 and 39:15–17). Claim 5 is an additional narrowing limitation beyond claim 1’s requirement of generating an output combining subtask outputs, and it cannot be satisfied by simply equating generic “combined results” with “accumulation of output records ... in an arbitrary order” without identifying a corresponding disclosure in Reeve. (Office Action, p. 9).

Accordingly, because Reeve’s tiled execution framework does not disclose subtasks producing “output records,” does not disclose forming the step (e) output by “accumulation” of such records, and does not disclose that the accumulated output is “in an arbitrary order,” the Examiner has not met the prima facie burden of showing that claim 5 is obvious over Reeve alone.

3. Reeve does not teach the claimed “report-creating process.”

Claim 10 depends from claim 1 and further requires that “said computer-executable process is a report-creating process.” This limitation is directed to the nature of the computer-executable process that is executed under the claim 1 framework, meaning the process that is partitioned into subtasks and whose outputs are combined in step (e). Claim 10 is not satisfied by showing that the system generates administrative or diagnostic reports about how a process ran. To establish a prima facie case of obviousness of claim 10 under Reeve alone, the Office Action must identify where Reeve teaches that the process being executed is itself report-

creating, not merely that the runtime can output statistics or reporting about execution. (Office Action, p. 9).

The Office Action's mapping does not meet that burden because the cited Reeve passage concerns reporting statistics about the execution of a process, not report creation as the substantive purpose of the executed process. (Office Action, p. 9 (citing Reeve, 30:13–14)). Reporting statistics on execution is qualitatively different from a “report-creating process” as recited in claim 10. Many non-reporting applications, including numerical kernels, sorting routines, and parallel constructs, can emit performance statistics, logs, or summaries after execution. That does not convert the underlying application into a report-creating process. In claim terms, telemetry about the run is not the “computer-executable process” being characterized; it is ancillary output about execution. (Reeve, 30:13–14).

This distinction is especially important in view of what Reeve is actually about. Reeve's disclosure is centered on compile-time tiling and runtime scheduling of tiles to threads for executing a parallel construct over an iteration space. (Reeve, 39:3–6). Reeve's technical discussion addresses how tiles are issued and executed and how load balancing is handled at runtime, not how an application generates a substantive report as its output. (Reeve, 39:3–6). A statement about reporting execution statistics therefore does not teach that the underlying computer-executable

process being partitioned and executed is a report-creating process, as claim 10 requires. (Office Action, p. 9; Reeve, 30:13–14).

Accordingly, because the Office Action relies on a Reeve passage that relates to reporting statistics about execution rather than disclosing that the executed computer-executable process is a report-creating process, the Examiner has not met the prima facie burden for claim 10 under the Reeve-only rejection. (Office Action, p. 9; Reeve, 30:13–14).

4. Reeve does not teach the claimed “computer-executable process includes a database query.”

Claim 11 depends from claim 1 and further requires that “said computer-executable process includes a database query.” This limitation is directed to the nature of the computer-executable process executed under the claim 1 framework, meaning the process that is partitioned into subtasks whose outputs are combined in step (e). Claim 11 is not satisfied by showing only that a parallel execution framework can execute code that accesses memory or reads data. To establish a prima facie case of obviousness of claim 11 based on Reeve alone, the Office Action must identify where Reeve teaches that the executed process includes a database query, or at least provide a reasoned basis for concluding that Reeve’s disclosed process includes querying a database. (Office Action, p. 9).

The Office Action does not make that showing. Reeve is directed to compile-time tiling and runtime scheduling for executing a parallel construct over an iteration space, with tiles assigned to threads according to strategies such as GRAB. (Reeve, 39:3–6). Reeve’s disclosures focus on how to generate and schedule tiled instruction sequences and how to allocate tiles to threads for load balancing. (Reeve, 39:3–6). They do not describe database structures, query statements, query predicates, query execution against a database management system, or any retrieval of records responsive to a database query. In fact, Reeve’s discussion is framed at the level of iteration space and execution scheduling, not at the level of application semantics such as “database queries.” (Reeve, 39:15–17).

To the extent the Office Action relies on Reeve’s general statements that the framework can be used with different types of processes, those statements do not disclose that the process being executed includes a database query. A generic capability statement, without a disclosure of a database query operation, is insufficient to meet claim 11 which requires more than the existence of data or memory access; it requires a database query as part of the executed process. The Office Action does not point to any Reeve passage teaching that feature. (Office Action, p. 9).

Accordingly, because Reeve does not disclose a database query and the Office Action does not identify any specific teaching or suggestion in Reeve that the

executed process includes a database query, the Examiner has not met the prima facie burden necessary to establish obviousness for claim 11 under the Reeve-only rejection.

E. Ground 2 Fails to Establish a Prima Facie Case: Reeve’s Task-Tiling Framework Does Not Disclose the ‘582 Patent’s Data-Partition Allocation Architecture, and Reeve Is Non-Analogous Art

For the reasons set forth above, the Office Action has not established a prima facie case that claims 1 through 14 would have been obvious over Reeve alone. The Reeve-only rejection depends on substituting Reeve’s compile-time iteration-space tiling and runtime tile-to-thread scheduling framework for the ‘582 Patent’s runtime, data-partition allocation architecture, but that substitution is not supported by Reeve’s disclosures. (Office Action, pp. 7–8; Reeve, 31:7–13, 32:10–16 and 39:4–6; ‘582 Patent, 3:30–33, 3:35–41, 3:47–50 and 4:1–4). Reeve’s “tiles” are bounded regions of an iteration space defined by index and bounds and issued to threads under scheduling strategies such as GRAB; they are task fragments of an iterative computation. (Reeve, 32:10–16 and 39:4–6). The claims, by contrast, require automatically determining file allocation and logically subdividing records in a preexisting input file into data partitions, with load balancing implemented by repeated allocation of an input partition “that has not been allocated yet.” (‘582 Patent, 3:35–41, 3:47–50 and 4:1–4). Reeve’s FCFS concept likewise operates in a different context, namely tile issuance to threads, and does not disclose the predicate

conditions and work-queue style allocation mechanism required by claim 1's "thereafter repeating ... each with another unprocessed partition" limitation. (Reeve, 39:4–6; '582 Patent, 3:47–50 and 4:1–4; Office Action, pp. 7–8).

The Office Action's mapping also fails with respect to claim 1's output generation requirement as described in the '582 Patent. The '582 Patent explains that each Sub Task produces output that must be further processed to create an output equivalent to the original process, including dedicated output files and a merge step that "merge[s] the outputs of the various Sub Tasks into the result output file or files." ('582 Patent, 4:5–9 and 4:57–60). Reeve's tiled execution of a parallel construct over an iteration space, where the runtime "looks only at the iteration space, and not at the data space," does not disclose that file-output pipeline or the generation and combination of discrete subtask output records into a result output file. (Reeve, 39:15–18; '582 Patent, 4:5–9 and 4:57–60; Office Action, pp. 7–8).

Independently, the Reeve-only rejection fails as a matter of law because Reeve is non-analogous art to the '582 Patent's particular problem and therefore cannot serve as the sole basis for a prima facie case of obviousness. *Circuit Check* confirms that a reference must be analogous to be considered within the prior art for an obviousness analysis, and the relevant inquiry must be tethered to the particular problem the inventor was trying to solve rather than a high-level abstraction such as "parallelism." *Circuit Check*, 795 F.3d 1331, 1335, 1338; (Office Action, pp. 7–8).

Here, the ‘582 Patent addresses run-time balancing of file-processing workloads through repeated allocation of unallocated file partitions, whereas Reeve addresses compile-time tiling and scheduling of iteration-space tiles. (‘582 Patent, 1:46–55, 3:47–50 and 4:1–4; Reeve, 1:63–2:13 and 32:10–16). Reeve therefore would not logically commend itself to a POSITA confronting the ‘582 Patent’s run-time data-partition allocation problem.

Finally, even if Reeve could be treated as relevant to claim 1 (which Patent Owner disputes), the Reeve-only ground independently fails as applied to dependent claims 3, 5, 10, and 11 because the Office Action has not identified where Reeve teaches the additional narrowing limitations those claims recite. (Office Action, p. 9). Reeve’s generalized statements that results are “combined” do not disclose claim 3’s distinct “merging” step for subtask outputs. (Office Action, p. 9; Reeve, 2:25–27, 8:9–11 and 19:38–42). Reeve does not disclose claim 5’s “accumulation of output records ... in an arbitrary order,” because Reeve’s tiled execution does not generate record outputs accumulated into a record-based output. (Office Action, p. 9; Reeve, 39:4–6 and 39:15–17). Reeve’s reporting of execution statistics does not disclose that the executed process itself is a “report-creating process” as required by claim 10. (Office Action, p. 9; Reeve, 30:13–14). And Reeve does not disclose that the executed process includes a “database query” as required by claim 11. (Office

Action, p. 9; Reeve, 39:15–17). Accordingly, claims 1 through 14 remain patentable, and Ground 2 should be withdrawn.

VII. CONCLUSION

For the reasons set forth above, Patent Owner respectfully submits that the Office Action has not established a prima facie case of unpatentability under either asserted ground, and that claims 1–14 remain patentable.

With respect to Ground 1 (Iyer in view of Reeve), the Office Action concedes that Iyer “does not explicitly teach” claim 1’s requirement that subtask processors “thereafter” repeat processing “each with another unprocessed partition” on a first-come/first-served basis and therefore relies on Reeve’s GRAB strategy as the necessary gap-filler. (Office Action, p. 4- 5). As shown above, that theory fails because Iyer affirmatively teaches a one-to-one work-division model in which the number of lists and partitions is selected to match the number of available processors so that “no processor load imbalance occurs,” and it does not disclose the a partition description framework that involves allocation/processing of unallocated partitions subsequent to an initial round of processing of partitions from the input file as the claims require. (Iyer, 2:45–49, 4:25–33 and 4:36–40; ‘582 Patent, 3:47–50 and 4:1–4). The Office Action’s only articulated motivation to combine is a generalized “improve efficiency” assertion, which does not identify any concrete deficiency in Iyer that GRAB would remedy and does not provide a reasoned, non-hindsight basis

to restructure Iyer into a pool-based repeated-claiming regime. (Office Action, p. 5). In addition, Reeve is not analogous art to the ‘582 Patent’s run-time, data-partition allocation problem and therefore cannot permissibly be used to supply the missing limitation or to support the proposed combination. *Sanofi-Aventis*, 66 F.4th 1373, 1380–82. Finally, even assuming *arguendo* that claim 1 were met, the Office Action does not show that Iyer and Reeve teach the additional narrowing limitations of dependent claims 5, 10, and 11. (Office Action, p. 6-7).

With respect to Ground 2 (Reeve alone), the Office Action’s mapping rests on an over-abstract characterization of Reeve that substitutes iteration-space tiling and tile-to-thread scheduling for claim 1’s file-allocation-based logical subdivision of records in a preexisting input file, the required sequencing, and the work-queue style repeated allocation of unallocated partitions. (Office Action, pp. 7–8; ‘582 Patent, 3:36–59 and 3:64–4:3; Reeve, 31:7–13, 32:10–16 and 39:4–6). Reeve is also non-analogous art to the particular problem addressed by the ‘582 Patent and therefore cannot serve as the sole basis for a *prima facie* case of obviousness. *Circuit Check*, 795 F.3d 1331, 1335, 1338. The Reeve-only ground independently fails as applied to dependent claims 3, 5, 10, and 11 because the Office Action does not identify where Reeve teaches the additional narrowing limitations that those claims recite. (Office Action, p. 9).

Patent Owner therefore respectfully requests withdrawal of the rejections under Grounds 1 and 2 and confirmation of the patentability of claims 1–14.

Respectfully submitted,

Dated: February 17, 2026

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

*Counsel for Patent Owner
Intellectual Ventures I LLC*