UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

ADVANCED MICRO DEVICES, INC. and PENSANDO SYSTEMS, INC.,

Petitioners,

v.

CONCURRENT VENTURES, LLC and XTREAMEDGE, INC.,

Patent Owners.

_____

Case IPR2025-00478

U.S. Patent 8,924,596

_____

DECLARATION OF DR. PAUL S. MIN IN SUPPORT OF PATENT OWNERS' RESPONSE

TABLE OF CONTENTS

I. SCOPE AND SUMMARY OF OPINIONS....6

II. MATERIALS CONSIDERED....7

III. QUALIFICATIONS AND PROFESSIONAL BACKGROUND....8

IV. UNDERSTANDING OF LEGAL PRINCIPLES....12

A. Obviousness....13

B. Claim Construction....15

V. PRIORITY DATE OF THE '596 PATENT....15

VI. LEVEL OF ORDINARY SKILL....16

VII. TECHNICAL BACKGROUND....17

VIII. SUMMARY OF THE '596 PATENT....21

A. Overview of the Patent....21

B. Prosecution History....27

C. District Court Allegations....28

IX. CLAIM CONSTRUCTION....28

X. THE ASSERTED PRIOR ART....29

A. U.S. Patent No. 8,559,439 ("Dongare") (EX1005)....29

B. U.S. Patent No. 8,051,227 ("Gewirtz") (EX1006)....34

C. U.S. Patent No. 7,398,528 ("Khawand") (EX1008)....36

D. U.S. Patent No. 5,805,930 ("Rosenthal") (EX1009)....38

E. U.S. Patent No. 8,028,284 ("Ohkawa") (EX1010)....39

XI. PETITIONERS AND DR. ALPERT DO NOT SHOW THAT ANY CHALLENGED CLAIM WOULD HAVE BEEN OBVIOUS....40

A. Petitioners Fail to Prove their Combination of Dongare and Gewirtz (Ground 1 and 2) Discloses or Renders Obvious Claims 1, 2, 5–8, 11–14, 17, and 18.....40

1. Claim 1.....40

a. Dongare and Gewirtz Did Not Render Obvious the Second Processing Element Accessing and Utilizing the Reservation Register, as Required by Limitations 1[e], 1[g], 1[h], and 1[i]. 41

b. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Sending Notifications to a First Processing Element to Issue Commands, as Required by Limitations 1[i] and 1[j]. 52

c. Dongare Does Render Obvious an "Input Queue Implemented in Hardware" As Required By Claims 1[a].....56

2. Independent Claims 7 and 13.....60

a. Limitations 7[a], 7[e]–7[i] and 13[a], 13[e]–13[i] Fail for the same reasons as equivalent in claim 1. 60

b. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Receiving a Command from the First Processing Element and Issuing a Command to the Input Queue on Behalf of the First Processing Element, as Required by Limitations 7[g]/[j] and 13[g]/[j].....60

c. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Receiving a Response Corresponding to Said Command from Said Output Queue, as Required by Limitation 13[k]. 65

3. Dependant claims.....68

a. Dongare and Gewirtz Did Not Render Obvious the Second Processing Element Receiving the Stored

Value from the Reservation Register as Required by Claims 5, 11, and 17. 68

b. Dongare and Gewirtz Did Not Render Obvious a Third Processing Element Notifying the Output Queue to Issue the Response as Required by Claims 6, 12, and 18. 72

4. Ground 2 Does Not Address The Problems In Ground 1. 74

B. Petitioners Fail to Prove their Combination of Khawand, Rosenthal, Ohkawa, and Matsunami (Grounds 3, 4, and 5) Discloses or Renders Obvious Claims 1–4, 6–10, 12–16, 18. 74

1. Claim 1 76

a. The References Teach Away From Petitioners' Proposed Combination 76

(1) Rosenthal Teaches Away From Petitioners' Proposed Combination for Limitations 1[e]–1[j]. 76

(2) Rosenthal Teaches Away From Petitioners' Proposed Combination for Limitations 1[e]–1[j]. 78

b. In Any Event, Petitioners' Proposed Combination Does Not Render Obvious Claim 1. 81

(1) Limitations 1[i]–1[j] Are Not Obvious in Light of Khawand and Rosenthal. 82

(2) Limitations 1[c] and 1[j] Are Not Obvious in Light of Khawand and Rosenthal 86

2. Claims 7 and 13 90

a. Limitations 7[a], 7[e]–7[i], and 7[k], and 13[a], 13[e]–13[i] Fail for the same reasons as equivalent in claim 1. 90

b. Khawand and Rosenthal Did Not Disclose or Render Obvious a Second Processing Element Receiving a Command from the First Processing Element and Issuing a Command to the Input Queue on Behalf of the First Processing Element, as Required by Limitations 7[g]/[j] and 13[g]/[j].....90

c. Khawand and Rosenthal Did Not Disclose or Render Obvious the Second Processing Element Receiving a Response Corresponding to a Command from the Output Queue, as Required by Limitations 13[k]. 93

3. Dependant claims.....96

a. Khawand and Rosenthal Did Not Disclose or Render Obvious Claims 6 and 12. 96

b. Petitioners Do Not Even Allege Claim 18 Would Be Obvious Over Khawand and Rosenthal. 97

4. Ground 4 Does Not Address The Problems In Ground 3. 97

5. Ground 5 Does Not Address The Problems In Ground 3. 98

XII. SIGNATURE.....99

I. SCOPE AND SUMMARY OF OPINIONS

1. I, Dr. Paul S. Min, am making this declaration in support of Concurrent Ventures, LLC and XtreamEdge, Inc.'s Patent Owners' Response in the matter of IPR2025-00486 before the Patent Trial and Appeal Board (PTAB) of the United States Patent and Trademark Office, which involves the Petition for Inter Partes Review ("IPR") of U.S. Patent No. 8,924,596 (the "'596 patent") filed by Advanced Micro Devices, Inc. and Pensando Systems, Inc. ("Petitioners").

2. As explained in detail below, I am qualified to offer opinions from the perspective of a person of ordinary skill in the art ("POSITA") at the time of the invention and can render an objective opinion regarding the issues raised in this IPR.

3. I have been asked to analyze the Petition (Paper 1) and Dr. Alpert's declaration (EX1003) in support thereof, as well as the '596 patent and its file history, and all identified prior art. I have been asked to provide my opinions about certain obviousness theories and opinions in Dr. Alpert's declaration. I have also been asked to provide opinions about how a POSITA would have understood and interpreted the '596 patent, including its claims; how a POSITA would have understood and interpreted the Petition's alleged prior art references; and a POSITA's knowledge of the state of the art as it relates to addressing these obviousness theories and opinions. My opinions and conclusions are set forth in this

Declaration. I understand that this Declaration is being submitted in connection with Patent Owners' Response to the Petition.

4. For purposes of this Declaration, I was asked to consider claims 1–18 of the '596 patent, which I understand are the challenged claims. *See* Pet., 16. I understand that claims 1, 7, and 13 are the independent claims.

5. This Declaration and its exhibits, in addition to my education, training, and experience in technology fields pertinent to the '596 patent, form the basis of my opinion.

6. I am not offering any conclusions as to the ultimate determinations that I understand the Patent Trial and Appeal Board ("Board") will make in this proceeding. I am solely providing my opinions on the technical interpretations, characterizations, theories, and opinions that I understand are involved in this proceeding.

7. I may offer additional opinions based on my further review of the materials in this case, including opinions and/or testimony of other expert declarants.

II. MATERIALS CONSIDERED

8. In forming the opinions in this Declaration, I reviewed the Petition (Paper 1), Dr. Alpert's declaration (EX1003), the materials considered by Dr. Alpert (EX1003, Appx. 3), the prior art and supporting materials cited by Petitioners, the

’596 patent, and the ’596 patent’s file history. I have also cited various other materials throughout my declaration that I considered as part of my analysis.

III. QUALIFICATIONS AND PROFESSIONAL BACKGROUND

9. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience. My curriculum vitae (“CV”) is submitted as Exhibit 2021.

10. I received a B.S. degree in Electrical Engineering in 1982, an M.S. degree in Electrical Engineering in 1984, and a Ph.D. degree in Electrical Engineering in 1987 from the University of Michigan in Ann Arbor. I received several academic honors, including my B.S. degree with honors, a best graduate student award and a best teaching assistant award during my M.S. study, and a best paper award from a major international conference for reporting results from my Ph.D. thesis.

11. After receiving my Ph.D., I worked at Bellcore in New Jersey from August 1987 until August 1990. At Bellcore, I was responsible for evolving the public switched telephone network (POTS) into a multi-services voice and data network that incorporated packet switches, optical technologies, and wireless technologies.

12. In September 1990, I joined the faculty at Washington University in St. Louis. In July 1996, I was promoted to an Associate Professor of Electrical

Engineering with tenure. I am currently a Senior Professor at Washington University of Electrical and Systems Engineering. I have also served as the Chair of the Graduate Curriculum (2000-2002) and the Chair of the Undergraduate Curriculum (2011-2014) for the Electrical and Systems Engineering department.

13. At Washington University, I have conducted research in communication, computing, and related electronic hardware and software. My research group has pioneered a new paradigm for designing electronic circuits that can alleviate the speed and performance mismatch against optical technology. I have received several grants from the U.S. Federal Agencies, including the National Science Foundation and the Defense Advanced Research Project Agency, and numerous contracts from companies and organizations around the world. As part of my research activities, I have designed and implemented a number of System-on-Chip (SoC) devices, which incorporated multiple processor-cores that communicate and cooperate with others.

14. As a faculty member at Washington University, I have taught a number of courses in electronics, communication, and computing at both the undergraduate and graduate levels. For example, I have taught communication theory (Washington University ESE 471), transmission and multiplexing (Washington University ESE 571), and signaling and control of communication networks (Washington University ESE 572).

15. I have supervised numerous graduate students, many of whom received a doctoral degree under my guidance. A number of doctoral theses that I have supervised relate specifically to the computing and communication technologies related to this proceeding. My students and I have published a number of peer-reviewed articles in these topics.

16. In addition to my responsibilities as a university faculty member, I have founded two companies. In May 1997, I founded MinMax Technologies, Inc., a fabless semiconductor company that developed switch fabric integrated circuit chips for the Internet. In March 1999, I founded Erlang Technology, Inc., a fabless semiconductor company that focused on the design and development of integrated circuit chips and software for the Internet. One of Erlang's products received a best product of the year award in 2004 from a major trade journal for the electronics industry. As part of my company activities, I have prototyped numerous chassis systems, which included multiple modular cards performing various tasks (e.g., switching card, memory card, input/output card, power supply card, etc.). In doing so, I became intimately familiar with various module standards such as CompactPCI and ATCA (Advanced Telecom Computing Architecture).

17. I developed and implemented a number of semiconductor integrated circuit chips. These integrated circuit chips include switch fabrics, network processors, protocol processor high speed interfaces, protocol transposers,

controllers, content search engines, look up table processors, and memory cells and arrays.

18. I am a named inventor on eleven U.S. patents, many of which are directly related to computing and communication technologies. I have extensively published technical papers in international conferences and journals, technical memoranda and reports, and given a number of seminars and invited talks. I have organized several international conferences and served as an international journal editor.

19. I am a member of and have been actively involved in a number of professional organizations. For example, I have served as the Chair of the Saint Louis Section of the IEEE with more than 3,000 members (2014), and a member of the Eta Kappa Nu Honor Society for electrical engineers. I have also been an Ambassador of the McDonnell International Scholars Academy (2007-2013).

20. I was a member of the Presidential Business Advisors Committee (to President George W. Bush) and was named 2002 Businessman of the Year by the Wall Street Journal for the State of Missouri, for my entrepreneurial efforts.

21. I have received multiple awards for my publications and contributions to technical communication systems. In 1988, I received the Best Paper Award at 18th ISATA Award of Technical Excellence in Florence, Italy. I was also a Rockwell Fellow at the University of Michigan in Ann Arbor in 1988 and 1989. I received a

Research Initiation Award in 1993 from the Defense Advance Research Project Agency (DARPA). I received the Best Paper Award at MOBILITY 2011 in Barcelona, Spain. In my nearly 40 years of professional experience, I have acquired significant knowledge about telecommunications, computer, and electronics industry standards and standard setting organizations such as ANSI, IEEE, ITU, IETF, Bellcore, and 3GPP.

22. Additional details of my education and work experience, awards and honors, and publications that may be relevant to the opinions I have formed are set forth in my curriculum vitae, as well as a list of cases in which I have testified as an expert at trial or by deposition. *See* EX2022.

IV. UNDERSTANDING OF LEGAL PRINCIPLES

23. At the outset, I want to clarify that all of my analysis and opinions in this declaration are from the perspective of a POSITA at the time of invention. To the extent that portions of this declaration do not expressly include language indicating that the analysis is from the perspective of a POSITA at the time of invention, it should be understood that my analysis and opinions are from the perspective of a person of ordinary skill in the art at the time of invention. For example, my opinion that a reference does not disclose or teach a claim limitation should be understood to mean that, in my opinion, a POSITA at the time of invention would have interpreted the reference as not disclosing the limitation. Likewise, my

opinions regarding obviousness are from the perspective of a POSITA at the time of the invention.

24. I do not offer opinions of law. The opinions I am expressing in this Declaration involve the application of my knowledge and experience to the evaluation of the challenged patent and certain art with respect to the challenged. The following paragraphs express my understanding of legal principles related to my analysis.

A. Obviousness

25. I understand that a claim is unpatentable under 35 U.S.C. § 103 if the differences between the claimed subject matter and the prior art are such that the claimed subject matter as a whole would have been obvious to a person of ordinary skill in the art at the time of the invention. I also understand that an obviousness analysis is based on the scope and content of the prior art, the differences between the claimed subject matter and the prior art, the level of ordinary skill in the pertinent art at the time of the invention, and objective indicia of nonobviousness, such as commercial success, industry praise, unexpected results, copying, and other such factors.

26. It is my understanding that, to find a claimed invention obvious in light of a combination of prior-art references, a person of ordinary skill in the art at the time of the invention must have had reason to combine the teachings of the prior-art

references to achieve the claimed invention, and that the person of ordinary skill must have had a reasonable expectation of succeeding in combining the teachings to arrive at the claimed invention. I further understand that a reason to combine or expectation of success is undermined if the combination or modification of the technology disclosed in the prior art would interfere with an objective of the technology disclosed in the prior art or if the prior art teaches away from making the combination or modification. I understand that a prior art reference is said to teach away from the claimed invention if a person of ordinary skill in the art at the time of the invention, upon reading the reference, would be discouraged from following the path set out in the reference, or would be led in a direction divergent from the path that was taken in the claim.

27. I understand that an assessment of what a reference discloses or teaches, for purposes of an obviousness analysis, must be conducted from the perspective of a person of ordinary skill in the art at the time of the invention. In other words, a reference discloses or teaches a claim limitation only if a person of ordinary skill in the art, at the time of the invention, would have interpreted the reference as disclosing the claim limitation. I understand that reliance on common sense, common wisdom, and common knowledge to inform an obviousness analysis must be explained with sufficient reasoning. I understand that assessment of obviousness

from the perspective of someone skilled in the art before the invention was made, without the benefit of hindsight.

28. I understand that it is not permissible to use hindsight after viewing the claimed invention to determine the question of obviousness or to rely on the teachings of the claimed invention in determining whether a POSITA would find the invention obvious.

B. Claim Construction

29. I understand that claims and their terms are construed, that is, interpreted, in accordance with their ordinary and customary meaning as understood by a person of ordinary skill in the art at the time of the invention when read in the context of the patent's specification and prosecution history. I further understand that if the patent's specification reveals a special definition given to a claim term by the inventor, then the inventor's special definition governs, even if it differs from the term's ordinary meaning.

30. I understand that there are certain terms for construction that are at issue in this proceeding. I have discussed these terms and my understanding of them further below.

V. PRIORITY DATE OF THE '596 PATENT

31. Dr. Alpert states that the time of the invention is December 6, 2013, which is the filing date of the '596 patent. For the purpose of my opinions in this

Declaration, I have been asked to assume the '596 patent is entitled to that priority date. Therefore, when I refer to a POSITA at the time of the invention in my Declaration, I am referring to a person of ordinary skill in the art as of December 6, 2018.

VI. LEVEL OF ORDINARY SKILL

32. Dr. Alpert set forth a level of ordinary skill as follows:

> I came to the conclusion that a person of ordinary skill in the field of art of the '596 patent would have been a person with (1) a bachelor's degree in electrical engineering, computer engineering, or a comparable major, and (2) approximately two to three years of experience in the field. Additional experience can substitute for level of education and vice-versa. A person with less education but more relevant practical experience may also meet this standard.

EX1003, ¶56.

33. I understand that the following factors may be considered in determining the appropriate level of skill in the art: (1) the educational level of the inventor; (2) type of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field. I understand that the person of ordinary skill in the art is not an actual individual, but instead is a construct for a hypothetical person who is presumed to be aware of all the pertinent prior art. I have addressed relevant information regarding these factors and applied this understanding below.

34. As summarized above with regards to my qualifications and experience, I possessed at least the level of a POSITA at the time of the invention of the ’596 patent because I had a Ph.D., M.S., and B.S. in Electrical Engineering. EX2021. I am currently a Senior Professor in the Department of Electrical Systems Engineering at Washington University, St. Louis, and have worked in the field of electrical engineering and data processing, streaming architecture, and data networks for more than thirty years. *Id.*

VII. TECHNICAL BACKGROUND

35. At the time of the ’596 patent, high-performance computing systems frequently employ specialized hardware modules, like network controllers, that execute commands issued by one or more processing elements. These commands were typically placed into hardware-implemented command queues such as first-in, first-out (“FIFO”) buffers for processing by execution hardware. In latency-sensitive systems, command queues are often intentionally shallow to minimize the delay between command issuance and execution, making efficient queue management critical to system performance.

36. As processing demands increased, system architects began distributing the task of command determination across multiple processing elements, rather than relying on a single processor to perform all command logic. In such architectures, different processing elements may perform distinct but interrelated functions. For

example, one processing element may act as a primary controller responsible for issuing commands into the hardware queue and receiving responses, while one or more additional processing elements assist by performing computationally intensive analysis, background scheduling, or decision-making related to which commands should be issued and when. These processing elements may communicate through shared memory, inter-processor links, or other coordination mechanisms. Most importantly, they operate concurrently and independently when determining command issuance.

37. This division of labor among multiple processing elements introduces a synchronization problem. While the processing elements may be cooperative, they are distinct execution units that may simultaneously determine that a command should be issued. Because all commands ultimately target a shared hardware command queue, the system must ensure that the queue is not overfilled and command issuance is orderly. The synchronization requirement is not merely between software threads, but between independent processing elements that each participate in a shared command-generation workflow.

38. In the prior art, coordination among multiple processing elements was typically achieved using software-based synchronization techniques. A shared variable stored in memory would track the amount of available space in the command queue, and each processing element would be required to acquire

exclusive access (*e.g.*, using mutexes, semaphores, or similar constructs) before checking and updating the variable. This approach forced multiple processing elements to serialize access to shared memory, consuming processing cycles that could otherwise be used for computational work. As the number of processing elements increased, the overhead associated with this software arbitration became more pronounced.

39. Software-based synchronization also complicated the relationship between command determination and command issuance. Because reservation of queue space and issuance of commands were separated in time and mediated by software, a processing element that determined a command should be issued could not reliably ensure that space would still be available by the time the command was actually enqueued. This problem was exacerbated in systems where different processing elements played different roles. For example, systems where one processing element made the decision to issue a command, but another processing element was responsible for placing the command into the hardware queue. In such cases, additional coordination was required, further increasing latency.

40. One novel architectural solution was to relocate the synchronization burden from software into hardware, specifically to support coordination among multiple processing elements. By using a shared hardware resource (*e.g.*, reservation register) that maintains a value indicative of available space in a

hardware command queue and is directly accessible by more than one processing element or processor, each processing element involved in command determination or issuance can independently access this reservation register as part of its role in the distributed workflow.

41. In the broader technical field, multi-processor and multi-engine systems were also developed to improve throughput by distributing work across multiple processing engines operating in parallel. These systems focused on assigning work units like packets to available engines and then restoring ordering after processing. The emphasis in such architecture was on load balancing and output reordering, rather than coordinated command issuance into a shared hardware queue. Processing engines were treated as independent execution resources, with system-level logic responsible for assigning work based on availability and later reconciling results to preserve ordering semantics.

42. Within this class of systems, the relationship between different processors or processing elements was typically mediated by centralized scheduling or tagging mechanisms instead of a defined cooperative interaction between a first processing element responsible for queue access and one assisting in command determination. Such systems describe multiple engines working in parallel, but they do not address architectures in which one processing element determines whether a command should be issued, another issues the command into a shared hardware

queue, and both rely on a common hardware-based reservation mechanism to synchronize access to limited queue space.

43. As a result, the technical field generally treated command ordering, engine scheduling, and output reordering as distinct concerns. The technical field did not assume these systems contained a hardware-level coordination mechanism governing how multiple processing elements jointly participate in command reservation, issuance, and execution against a shared queue resource.

VIII. SUMMARY OF THE ’596 PATENT

A. Overview of the Patent

44. I reviewed the ’596 patent in preparing this declaration. The inventors of the ’596 patent identified a problem in the art of computer processing relating to “load distribution.” EX1001, 1:9–10. The ’596 patent addresses the problem of coordinating multiple processing elements that collectively generate commands for a hardware module without overfilling the module’s command queue. *Id.* The inventors of the ’596 patent discovered that it was preferable for that role to be performed by a different processing element, distinct from both the executing and issuing processors so the determination could be performed quickly and efficiently. *Id.*, 5:3-7:58.

45. “In the prior art, the processing load (of a processing element) to determine what command to issue is relatively high.” *Id.* at 1:15–16. These high

processing loads could be addressed by "partially or fully offload[ing] to a multiplicity of other processing elements (e.g., processors, processor cores, or custom processing circuitry)," but this created a new problem with "synchronization . . . amongst the multiplicity of processing elements." *Id.* at 1:17–25.

46. Prior art systems attempted to address the synchronization problem by "using software techniques, such as semaphores and mutexes to a shared available space count variable in memory." *Id.* at 1:25–27. These software techniques, however, required "additional processing time" and thus were "non-optimal," particularly because "there may be a need to keep the time from command issue to execution minimized." *Id.* at 1:27–32. Software techniques that operate on a shared "available space" variable necessarily require overhead by the first processor, meaning that the first processor has less memory to accomplish other processing tasks. *Id.*

47. To address this problem in the art, the inventors of the '596 patent developed "a system for dividing and synchronizing a processing task across a plurality of processing elements" that utilized input and output queues and a reservation register implemented in hardware, among other claim elements. *Id.* at 2:13–14. The invention uses a hardware reservation register that tracks available queue space and automatically decrements itself when read by a processing element. *Id.* at 3:43–47. The hardware reservation register guarantees mutual exclusion

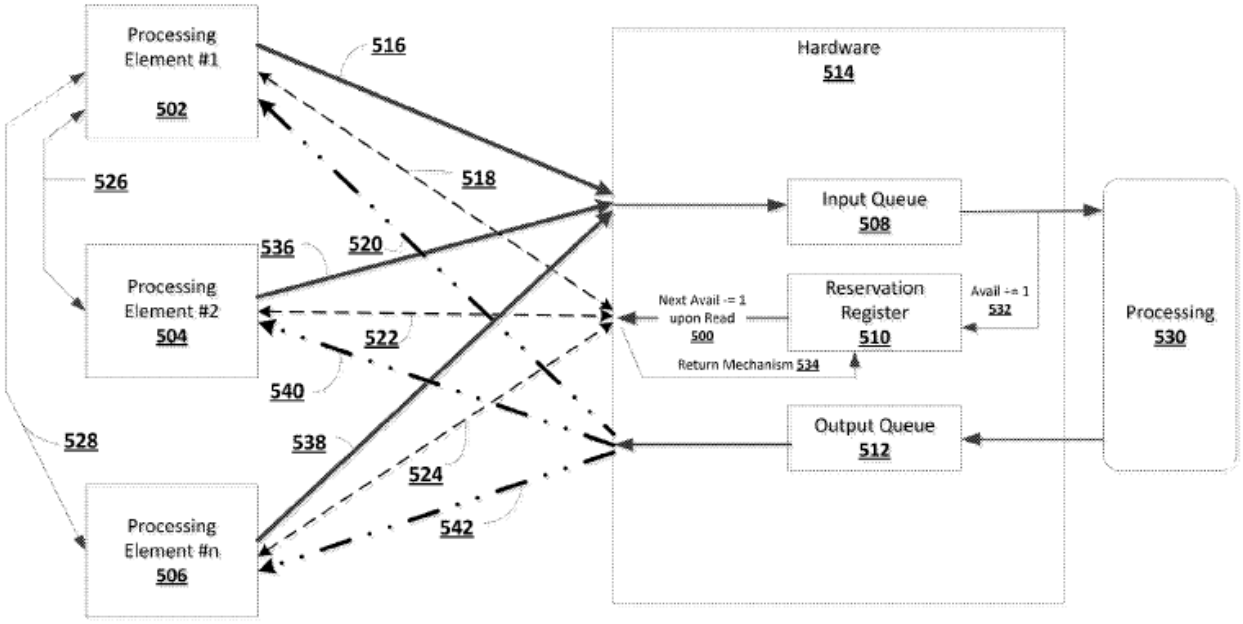
without software locks because a processor need only read the reservation register to determine if there is sufficient space to add a command to a queue. *Id.* at 3:59–62, 3:23–29.

48. The inventors of the ’596 patent recognized the importance of using multiple processors in this data flow system. The invention requires at least two processing elements that share access to the reservation register and the input/output queues because “the processing load (of a processing element) to determine what command to issue is relatively high).” EX1001, 1:15–16. The ’596 patent describes that distributing the determination workload across multiple processes is not desirable because of the processing load placed on that processor, so the inventors presented a solution where the processing load is placed on a separate processing element:

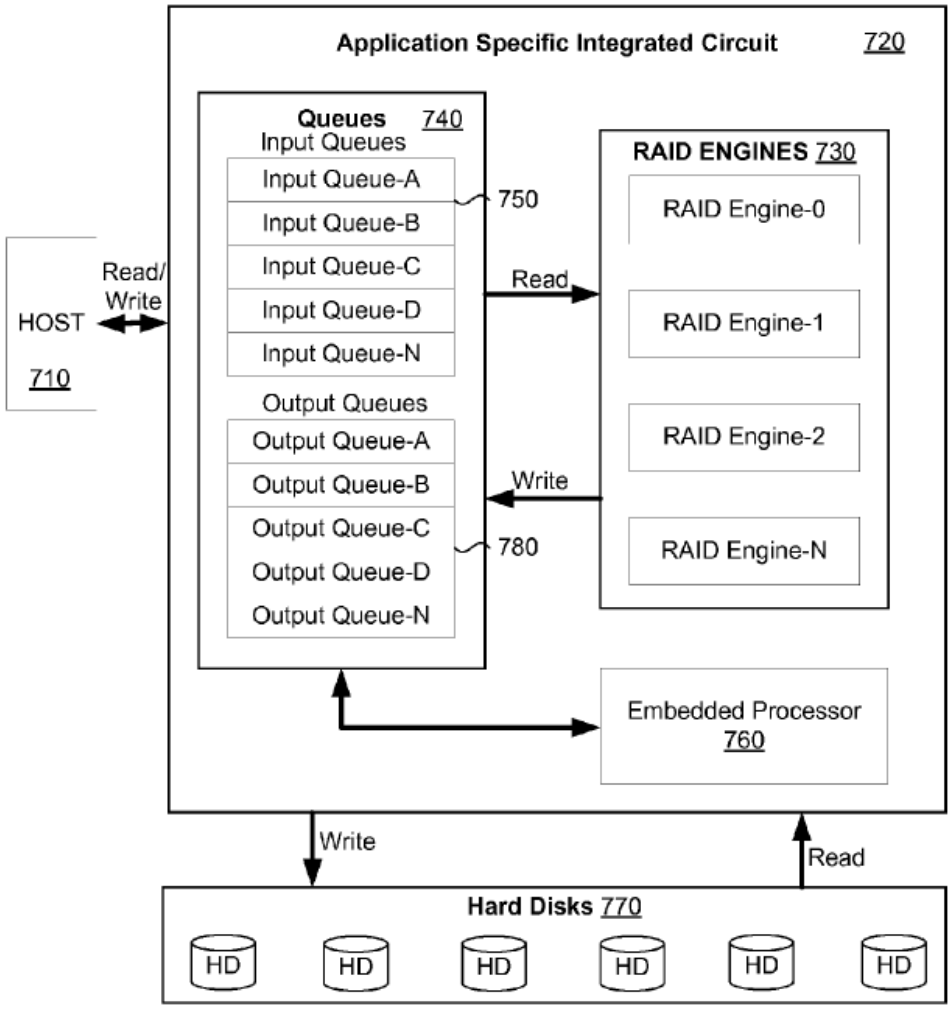
> In one embodiment there is a single processing element connected to the module’s command queue. It is desired to have a single point of responsibility for this hardware module (such as a control interface to a storage processing element) to issue the commands and deal with the responses from the module. However, the processing load to determine all or a subset of commands to issue may be high. This processing load is placed on a separate processing element. The two processing elements communicate in some efficient way (such as through shared memory, etc.).

Id. at 4:21–29. Pairing the reservation register with a multi-processor system where the first processor offloads queue management to the second processor solves problems in the prior art with memory and latency.

49. This can be seen in Figure 5 of the '596 patent. It depicts the relationship between a first processing element, second processing element, and hardware reservation register tracking the space available in an input queue:



EX1001, FIG. 5. Unlike in the prior art, the first processing element is in direct communication with the second processing element, and the second processing element is the processor responsible for checking the available space in the reservation register. *Id.* This sharply contrasts with the architecture in the prior art (*e.g.*, Dongare) where the first processing element alone is forced to manage the input queue because the second processing element (*e.g.*, Dongare's Embedded Processor) instead sits between the input queues and downstream processing engines which do not support input queue management.



EX1005, FIG. 7.

50. Claim 1 recites a hardware system that divides responsibility for issuing and coordinating commands between a first processing element and at least one second processing element, with synchronization enforced through a shared reservation register. EX1001, cl. 1. In claim 1, the first processing element is the only processor that adds commands to the hardware input queue and receives responses from the hardware output queue. *Id.* The second processing element does not issue commands into the input queue; rather, it monitors queue availability by

reading a reservation register implemented in hardware and notifying the first processing element when it is safe to issue a command. *Id.* In this way, claim 1 teaches a cooperative relationship between two processing elements where the first processing element offloads queue-state management to the second processing element, with synchronization enforced in hardware through the shared reservation register accessible to both processors. *Id*

51. Claims 7 and 13 are similar to Claim 1, but each have a key difference that cause Petitioners' arguments to fail for additional reasons. Claims 7 and 13 both expand on the cooperative processor framework of claim 1 by shifting more responsibility to the second processing element. EX1001, cl. 7, 13. For claim 7, the first processing element still serves as the originator of commands and recipient of responses, but the second processing element issues the commands into the hardware input queue "on behalf of" the first processing element. *Id.*, cl. 7. For claim 13, the second processing element still issues the commands "on behalf of" the first processing element, but the second processing element is also given access to the output queues so that it (instead of the first processing element) receives the response from the output queue. *Id.*, cl. 13. This further distributes decision-making, issuance, and execution across distinct processors while maintaining synchronization via the reservation register. *Id.*

52. To summarize, one of the key insights of the ’596 patent is that it is beneficial to offload the decision-making for when a first processor should issue a command to a distinct second processor. Thus, in the ’596 patent, responsibilities are split: the first processor is responsible for issuing the command, but it is the second processor that is responsible for deciding when that command should be issued.

53. Each claim of the ’596 patent requires a second processor, communicatively linked to a first processor, which reads a hardware reservation register and either: (1) signals the first processor that it may safely write a command into the input queue; or (2) receives the command from the first processor and writes it into the input queue on behalf of the first processor when it is safe. *Id.*, 4:26–27. This novel relationship between the first processor, second processor, and reservation register offloads responsibility from the first processor, allowing it to allocate its memory to more demanding computational tasks. This is advantageous in complex multiprocessor systems where the first processor has limited memory.

B. Prosecution History

54. Dr. Alpert discusses that a European counterpart to the ’596, EP14867963, was allegedly rejected because a Supplementary Search Report identified Gewirtz, with the EU Search Opinion noting that “[i]t is common general

knowledge of the skilled person [] that an output queue [] is an implementation choice for buffering request results from a storage device." EX1003, ¶44.

55. In my opinion, the cited portion of the prosecution history is not relevant to this proceeding because there is no dispute that output queues in general existed before the priority date of the '596 patent. Instead, as explained below, output queues were not used in the particular configurations claimed by the '596 patent before that date.

C. District Court Allegations

56. Dr. Alpert repeats certain infringement allegations from the related district court litigation, but he does not tether these claims to his opinion on the prior art. Based on my reading, I disagree with the limited scope of his summary, namely his summarizing hundreds of pages of infringement contentions into no more than five paragraphs.

IX. CLAIM CONSTRUCTION

57. In my opinion, a POSITA would have interpreted each claim term of the '596 patent, when read in the context of the specification and prosecution history, to have their plain and ordinary meaning. Except for the specific instance discussed below where I have applied a construction proposed by Petitioner for purposes of evaluating the Petition, I have applied the plain meaning of the claims in forming my opinions.

58. I understand that Petitioners have proposed a construction for "input/output queue implemented in hardware" in district court. I analyzed the Petitioners' proposed construction below and disagree that an "input/output queue implemented in hardware" must mean an input/output queue implemented in physical circuits without software intervention. A POSITA would understand that, even where an input/output queues is implemented in hardware, software may be used to configure the logic implemented in the hardware, thus Petitioners' proposed construction is too restrictive and introduces ambiguity on how the input/output queues is configured.

59. However, for purposes of analyzing the Petition, I have applied Petitioners' proposed construction because Petitioners' arguments fail regardless of whether this proposed construction is adopted. Where discussed below, the prior art does not teach input/output queues implemented in physical circuits with or without software intervention.

X. THE ASSERTED PRIOR ART

A. U.S. Patent No. 8,559,439 ("Dongare") (EX1005)

60. I have reviewed the Dongare reference.

61. Dongare is U.S. Patent No. 8,559,439 entitled "Method and Apparatus for Queue Ordering in a Multi-Engine Processing System." Dongare purports to

disclose "[a] method and apparatus for queue-ordering commands in multi-engines, multi-queues and/or multi-flows environment." EX1005, Abstract.

62. Dongare teaches a buffer-based system in which "input queues," described as "sequentially ordered commands or segments to be processed by the plurality of engines, are "received and stored in input queue buffer[s]." EX1005, 5:3–12. Petitioners use Dongare as the primary reference used in Grounds 1 and 2, namely for the "second processing element" limitation present in each independent claim.

63. Dongare describes a multi-engine processing system where an on-chip processing element preserves the order of commands or packet segments as they are processed by RAID engines. EX1005, Abstract. Dongare identifies a problem in the prior art inherent to multi-engine processing systems where multiple engines are used to increase the throughput of a device or network system. *Id.* at 1:25–32.

64. When multiple engines process commands or packet segments from the same input queue, it is beneficial to process the commands or packet segments out-of-order based on the availability of the processing engine. *Id.* at 5:62–65. But because segments from the same input stream may be dispatched to different engines, the resulting output may be disordered relative to the input. *Id.* at 1:25–32.

65. To address this, the on-chip processing element assigns each command or packet segment an "identity tag" that encodes its stream ID and original sequence

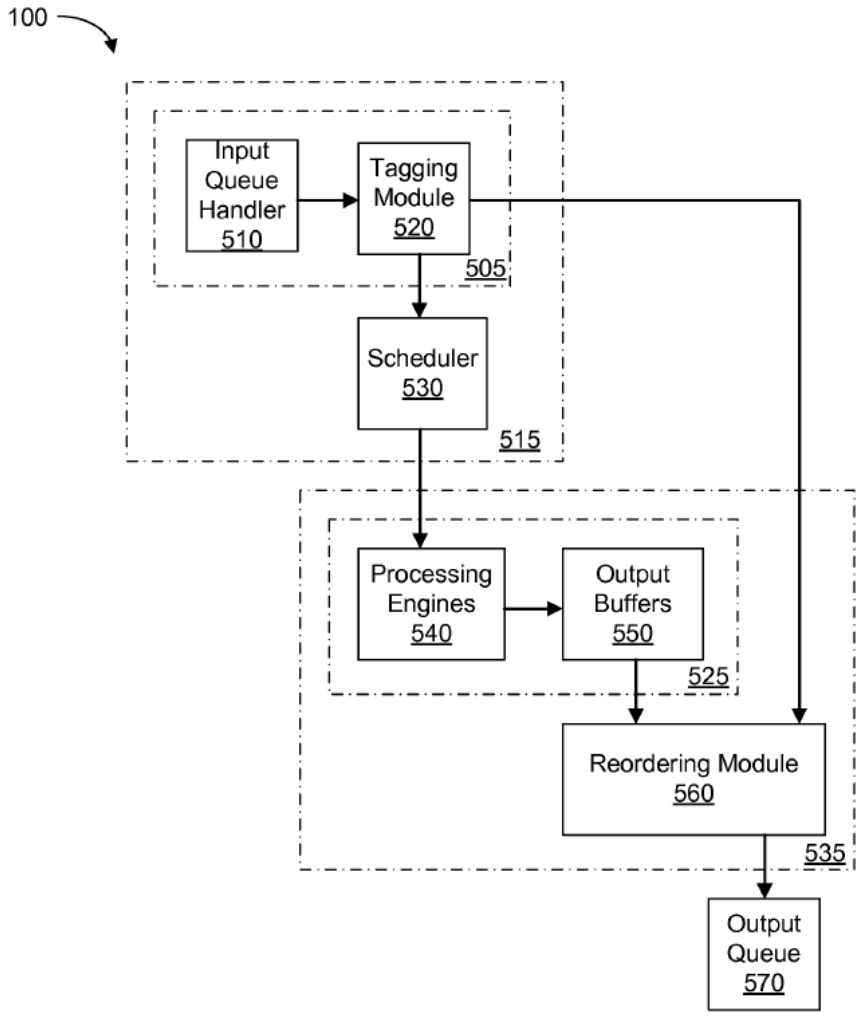
position, which is used by a reordering module after execution to reconstruct the original sequence. *Id.* at 5:42–53.

66. Dongare is about the optimization of command and packets segment processing by downstream processing engines—not ensuring that input queues remain available and uncorrupted. The only reference to the input queue in Dongare is discussion of storing the commands and packet segments in an input buffer after they are already placed in the input queue by the first processor and before they are sent to the downstream processing engine for execution. EX1005, 5:8–12.

67. Dongare makes clear that it concerns the relationship between commands in an input queue rather than anything related to how those commands enter the input queue in the first place. *Id.* at 5:27–41. Dongare distinguishes itself from prior art by enabling out-of-order assignment and execution of commands without the reordering penalties inherent to older systems. *Id.* at 4:49–62. Dongare describes prior approaches that required either matching sequence numbers at write-out time, which would cause engines to idle when earlier segments were delayed, or maintaining complex reassembly buffers, expiration timers, and retransmission logic. *See, e.g.*, 3:13–40.

68. Dongare sought to improve upon the problem of disordered commands requiring reassembly after execution. It does not address offloading queue management from a first processor to conserve processing power or ensure that input

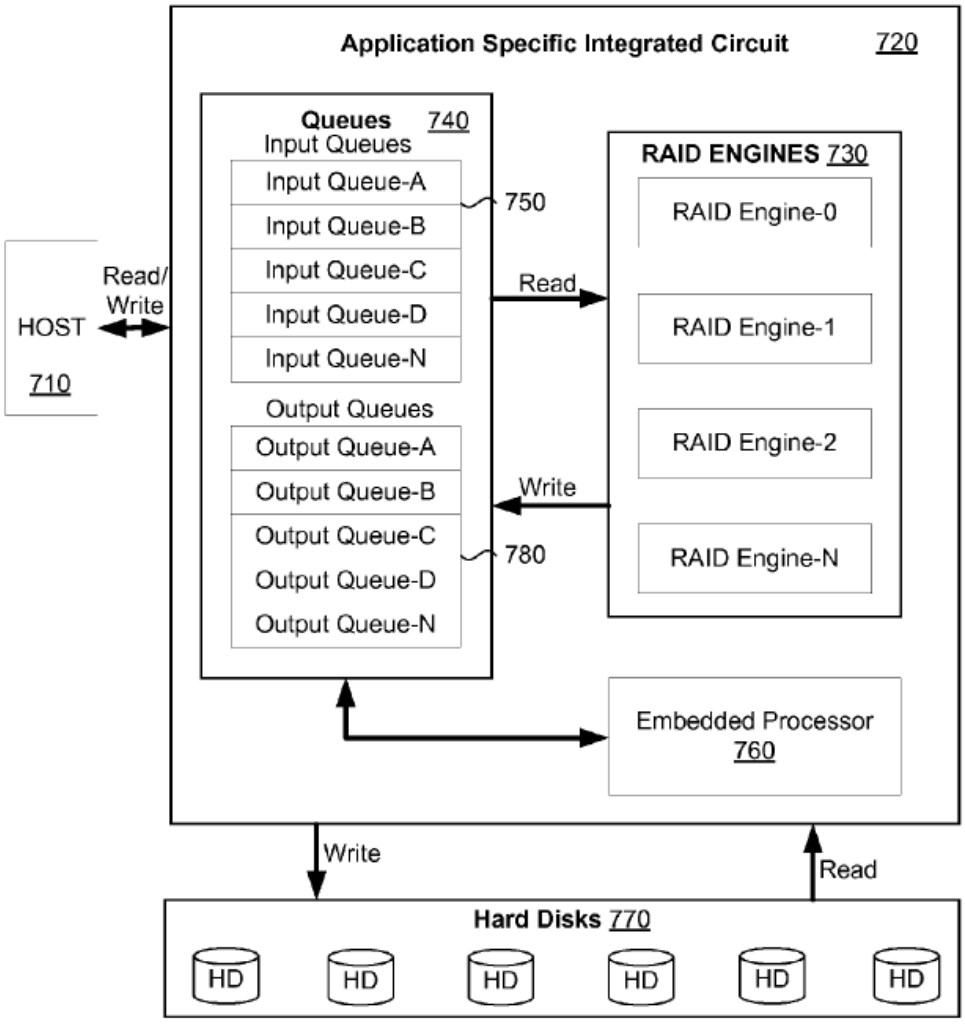
queues remain available by atomic counting. Measures like a semaphore or mutex are not discussed because it would not solve the problem identified by the reference, nor does the reference disclose a processor capable of managing a space variable in software or hardware. Dongare merely discloses a system architecture including an input queue handler, tagging module, scheduler, bank of parallel processing engines, dedicate per-engine output buffers, and reordering module:



EX1005, FIG. 5, 10:50–11:13. System 100 covers the entirety of the disclosed system, and Figure 5 illustrates how the system is only concerned with what occurs

after commands or packet segments are added to an input queue. *Id.* (showing the pathway after being picked up by the input queue handler).

69. Figure 7 of Dongare also underscores the narrow application of the invention. EP 760 (i.e., the on-chip processor responsible for tagging the commands and input segments) sits downstream from the input queues inside the ASIC to aid in handling, tagging, and scheduling:



EX1005, FIG. 7. Dongare does not disclose a system where Host 710 and EP 760 are in direct communication (except indirectly via post-processing confirmations

placed in the output queue) because the reference proposes a system where Host 710 writes commands without regard to available space in the input queue. Dongare's Embedded Processor is only concerned with internal engine coordination and ordered output reconstruction.

B. U.S. Patent No. 8,051,227 ("Gewirtz") (EX1006)

70. I have reviewed the Gewirtz reference.

71. Gewirtz is entitled "Programmable Queue Structures for Multiprocessors." EX1006, Abstract. Petitioners rely on Gewirtz primarily for its teachings regarding a "Queue State Register." Pet. 19–27. Gewirtz purports to disclose a processing system where a host processor manages input queues using a queue-state register which tracks the available space in an input queue. EX1006, Abstract. Petitioners use Gewirtz as a secondary reference in Grounds 1 and 2, namely for the "reservation register" limitation required in each independent claim.

72. Gewirtz describes a processing system where a credit counter is used for flow control. EX1006, 2:59–62. The counter tracks the number of free entries and returns a pass/fail result to an agent attempting to reserve space before writing a command or packet segment. *Id.* at 4:61–67. When the agent reads the credit counter, the credit counter decrements the count on a successful reservation and increments when an entry is executed by an engine. *Id.* at 5:3–6.

73. The credit counter does not schedule, orchestrate, or manage queue operations on behalf of the agent, rather the agent (*i.e.*, the host processor core running the software process) decides whether to perform or retry the operation. *Id.* at 4:56–61. The credit counter simply provides an atomic "check-and-update upon successful read" and leaves the remaining logic up to the host processor. *Id.*

74. Gewirtz does not teach a system where queue-state management is offloaded from a host processor to a second processor. Rather than offloading queue-state management from a host processor with the goal of reducing its processing load, Gewirtz teaches a system where the processor which reads the queue-state register is the same processor that writes the command into the input queue. EX1006, 4:56–61.

75. Gewirtz provides that all agents which access the queues are software processes executed by processor cores of a network processor. *Id.* at 1:39–44. The queue-state register merely contains logic which can help the host processor evaluate queue fullness and make accept/reject decisions while leaving the software processes responsible for deciding whether to write or retry writing the commands. *Id.*

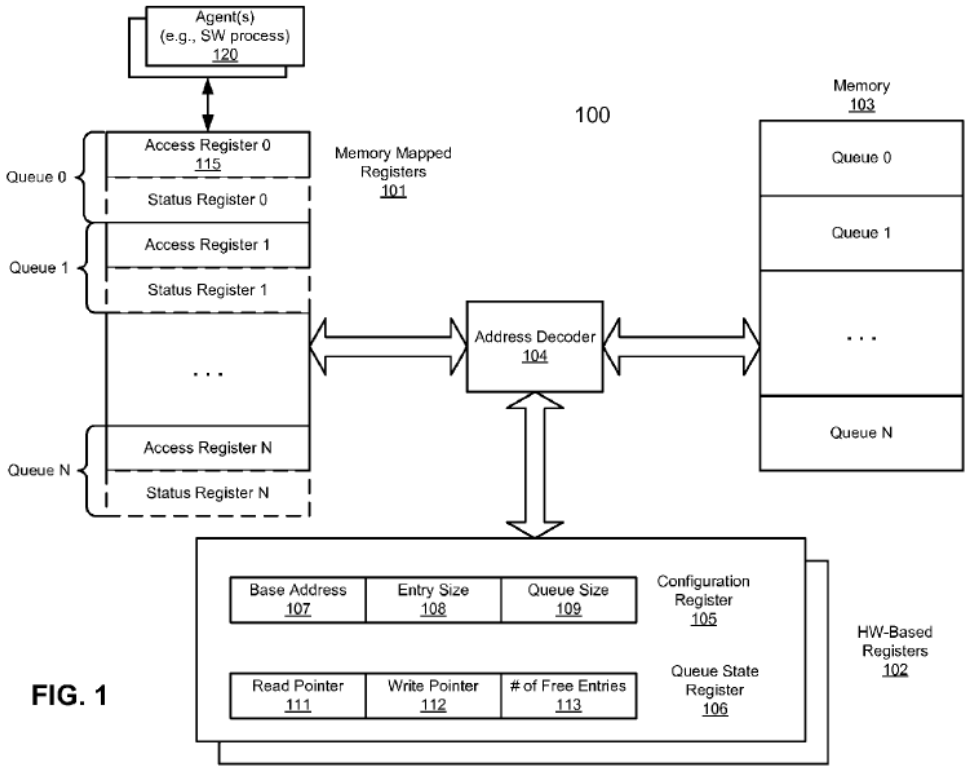


FIG. 1

Id. at FIG. 2 (reflecting that the reference does not teach a load balancing system, as reflected by the Agent(s) 120 block of the diagram). Gewirtz does not solve the problem identified in the '596 patent where the processing load to determine all or a subset of commands to issue is too burdensome on the host processor. EX1001, 4:20–29.

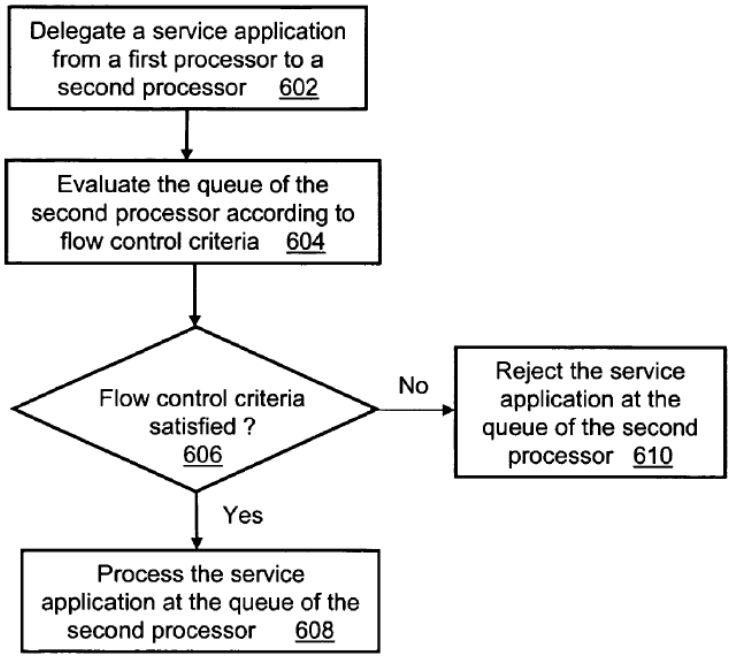
C. U.S. Patent No. 7,398,528 ("Khawand") (EX1008)

76. I have reviewed the Khawand reference.

77. Khawand is entitled "Method And System For Efficient Multiprocessor Processing In A Mobile Wireless Communication Device." Petitioners contend that Khawand teaches "a multiprocessor architecture that improves efficiency using delegation, queues, and flow control." Pet., 44.

78. Khawand describes a processing system in which "Processor 1" is set to process a service application or task (*i.e.*, command) and instead delegates the entire a service application or task to "Processor 2" for execution. EX1008, 1:35–48, FIG. 3. Processor 1 initiates this delegation on its own when its processing efficiency begins to suffer and Processor 2 receives the command in its input queue. *Id.*, 3:62–4:5. However, if Processor 2 is also unable to execute the command due to lack of available memory, it rejects the command and sends a notification through an "Inter-Processor Manager" ("IPM 308) back to Processor 1 to "suspend" any more delegation. *Id.*, 4:60–67. Processor 2 may then send a "resume" notification through IPM 308 if space becomes available. *Id.*, 5:18–29.

79. In my opinion, Khawand's model for synchronizing multiple processors differs substantially from the '596 patent, where multiple processing elements jointly participate in issuing commands to a shared hardware queue using a reservation register. In Khawand, the purported 2PE (IPM 308) does not mediate the issuing of commands to an input queue ***before*** those commands are issued. EX1008, 4:6–14.



Id., FIG. 6. Instead, Processor 1 issues commands to an input queue on its own, and if Processor 2 is unable to process those commands, it may drop them and send back a response to Processor 1 to suspend any subsequent delegation. *Id.*

80. I understand that Khawand is the primary reference for Grounds 3, 4, and 5.

D. U.S. Patent No. 5,805,930 ("Rosenthal") (EX1009)

81. I have reviewed the Rosenthal reference.

82. Rosenthal is entitled "System for FIFO Informing the Availability of Stages to Store Commands Which Include Data and Virtual Address Sent Directly From Application Programs." Petitioners rely on Rosenthal as a secondary reference in Grounds 3 and 4, principally for its teachings regarding FIFO methodologies. Pet. 57-60. Rosenthal was before the Patent Office during prosecution of the '596 patent.

83. Rosenthal describes an input/output architecture centered around a FIFO buffer, where application programs write commands directly into a dedicated FIFO associated with an I/O control unit. EX1009, Abstract. Before doing so, the CPU (which is the only processor involved in the system) reads a "flow-control register" (which Petitioners map to the RR) indicating how much space remains in the FIFO. *Id*, 2:11–26. This register reports capacity, but does not reserve space, and there is no coordination between multiple processing elements to issue the command. *Id.*, 2:26–28. Rosenthal's system is fundamentally single processor (*i.e.*, CPU), where flow control is enforced by checking a register before writing to the queue. *Id.*, 2:28–33.

84. I understand that Rosenthal is the secondary reference for Grounds 3, 4, and 5.

E. U.S. Patent No. 8,028,284 ("Ohkawa") (EX1010)

85. I have reviewed the Ohkawa reference.

86. Ohkawa is entitled "Data Processing System And Program For Transferring Data."

87. I understand that Petitioners rely on Ohkawa solely to supply the "output queue implemented in hardware" limitation in Ground 4. Pet. 74–79.

XI. PETITIONERS AND DR. ALPERT DO NOT SHOW THAT ANY CHALLENGED CLAIM WOULD HAVE BEEN OBVIOUS

A. Petitioners Fail to Prove their Combination of Dongare and Gewirtz (Ground 1 and 2) Discloses or Renders Obvious Claims 1, 2, 5–8, 11–14, 17, and 18.

1. Claim 1

88. In my opinion, Grounds 1 and 2 fail to disclose claim 1 for three independent reasons.

89. As background, the ’596 patents teaches that it is beneficial to offload the decision-making for when a first processor should issue a command to a distinct second processor so the first processor may allocate its memory to more demanding tasks. It splits responsibilities among these processors: the 1PE is responsible for issuing the command, but it is the 2PE that is responsible for deciding when that command should be issued.

90. Accordingly, each claim of the ’596 patent requires the 2PE, communicatively linked to the 1PE, which reads a hardware RR and either: (1) signals the 1PE that it may safely write a command into the input queue (claim 1); or (2) receives the command from the 1PE and writes it into the input queue on behalf of the 1PE when there is room to do so (claims 7 and 13). *Id.*, 4:26–27.

91. First, Dongare and Gewirtz at most disclose two processors (Dongare) and a reservation register (Gewirtz)—there is nothing in Dongare or Gewirtz to

suggest it is the alleged "second processor" that uses the reservation register to determine when there is space in the input queue.

92. Second, there is no teaching or suggestion in either Dongare or Gewirtz that it is the alleged "second processor" that notifies the "first processor" to issue a command. The key innovation of the '596 patent—that different processors are responsible for issuing the command (the first processor) and for deciding when that command should be issued (the second processor)—is wholly absent from Petitioners' references.

93. Third, neither reference discloses that the alleged "input queue" is implemented in hardware.

a. Dongare and Gewirtz Did Not Render Obvious the Second Processing Element Accessing and Utilizing the Reservation Register, as Required by Limitations 1[e], 1[g], 1[h], and 1[i].

94. In my opinion, Dongare and Gewirtz do not disclose or suggest (alone or in combination) a "second processing element" that accesses the reservation register (claim elements 1[e] and 1[g]), reads the stored value in the reservation register indicative of available space in the input queue (claim element 1[g]), and determines when that value indicates available space in the input queue for the first processing element to issue a command (claim element 1[h]). EX1001, cl. 1.

95. I understand that Petitioners' theory is: (1) Dongare's "EP 760" is the claimed "second processing element," Pet., 28, 30–31; (2) Gewirtz's "Queue State

Register 106" is the claimed "reservation register," Pet., 19–20; and (3) it would have been obvious for it to be EP 760 that accesses and utilizes Queue State Register 106 to perform the input queue offloading steps.

96. Grounds 1 and 2 for claim 1 thus require that it would be obvious for it to be the second processor that accesses the reservation register, reads the stored value, and determines when there is available space in the input queue for the first processing element to issue a command. I disagree that the prior art presents any suggestion or teaching of these limitations.

97. There is nothing in Dongare that suggests that EP 760 would perform the input queue offloading steps. Dongare contains limited discussion on EP 760, referencing it only three times. Dongare's entire disclosure regarding EP 760 is below:

- "The input queues 750 are read by the **embedded processor 760** and commands are created for RAID engines 730 to process data. Newly generated commands are loaded to input queues 750 dedicated to RAID engines 730." EX1005, 14:41–46.

- "The **embedded processor 760** monitors output queues 780 for completion of RAID commands in the same order as was loaded in input queues 750." EX1005, 14:49–51.

- "After completing the Host command, the **embedded processor 760** will signal the Host 710 that the command is complete by writing into the outbound queue 780." EX1005, 14:52–54.

98. In my opinion, a POSITA would understand that EP 760 accesses commands that have already been added to an input queue and appends an identity

tag to the commands within the input stream in the order they were added. After the commands are tagged, EP 760 assigns them to processing engines irrespective of the order they were added to the input queue. EP 760's role is simply to access commands that have already been written to the input queue and portion overly long commands into smaller command segments and append an identity tag to the command segments that identifies their order in the queue (*i.e.*, their sequence).

99. This is Dongare's claimed invention—processing engines downstream of a first processor execute commands without regard to the order they were placed in their input queue, and then after they produce tagged output segments, place them in an output buffer. EP 760 reorders the tagged output segments within the output buffer and then routes them to their destination via output streams.

100. A POSITA would understand that EP 760 performs the following functions: (1) portions long commands into smaller command segments; (2) tags commands and command segments for reordering after processing; and (3) reorders the commands after they have been processed. I understand that Dr. Russ agrees with this summary. EX2022, 104:12–16 (stressing that EP 760 only operates on commands that Host 710 has already added to an input queue).

101. EP 760 does not determine whether there is space in the queue, or that it has any role regarding issuing commands to the queues at all. EP 760 does not have any role before the command is issued at all. Its first role is to tag queues that

have already been issued, which can inherently only be done after they are added to input queues by Host 710—without first determining whether there is space at the destination. EX1005, 10:50–11:13, FIG. 3. The only role of EP 760 is to enable RAID engines to process commands out of order by tagging their correct order and reordering them after processing. *Id.*, 10:50–11:13.

102. Dongare does not suggest that the EP should determine when there is space in the input queue Each of Petitioners arguments to the contrary does not withstand muster. Petitioners make a number of arguments about why a POSITA would be motivated to use Gewirtz's Queue Status Register with Dongare generally, but these arguments highlight that the claims do not just require a reservation register, but that it is the second processing element using the reservation register for the first processing element to issue commands.

103. First, they argue that it would be obvious for the "reservation register [to be] accessible by both the first and second processing elements" because "Gewirtz makes the register accessible by software agents running on 'multiple processor cores'" and "Dongare's host and embedded processors are the processors in Ground 1." Pet., 22 ("A POSITA would have recognized the value in having this information be visible to multiple processors for synchronization purposes." *See id.*; Ex-1003, ¶113.") Pet., 22.

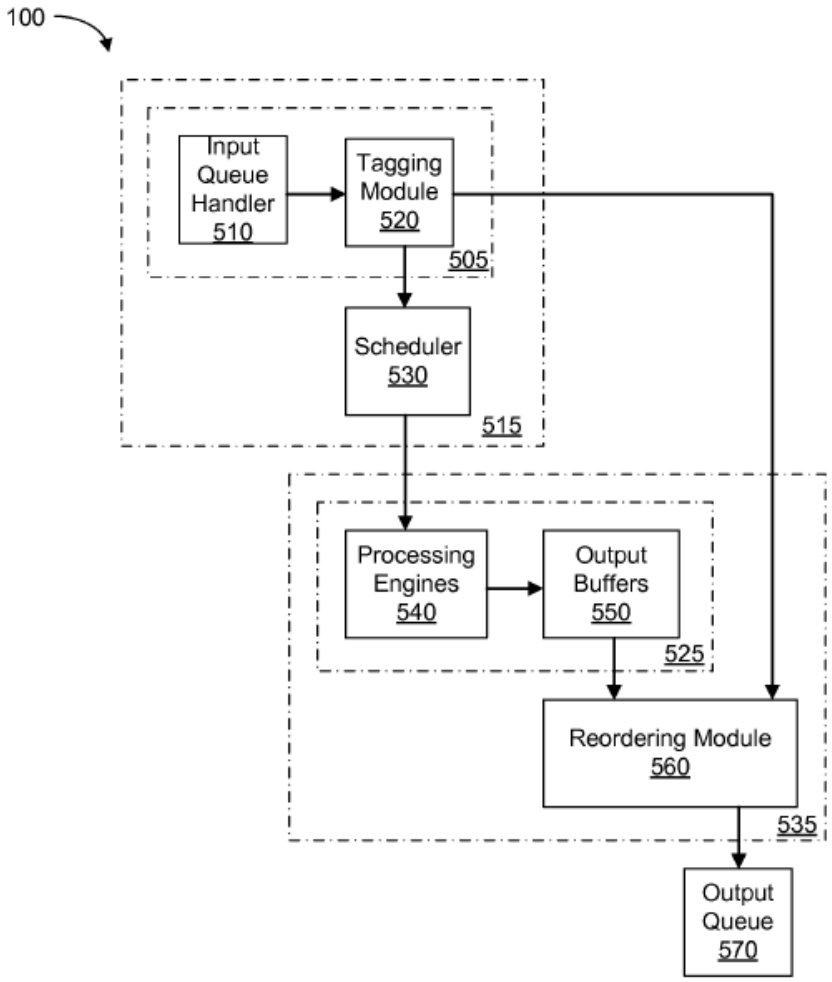
104. In my opinion, EP 760 is only responsible for tagging and reordering commands. There is no reason for it to access the reservation register to perform this role, as the commands have already been added to input queues at this point. Petitioners cite Gewirtz, but a POSITA would understand that it says nothing about why EP 760 would have access, as Gewirtz does not teach a configuration with a second processing element.

105. Further, Dr. Russ's does not explain any reason why EP 760 would have access. Petitioners at most show that EP 760 could have access to the reservation register, not that it does all of the steps which separate the '596 patent from the prior art.

106. Second, they argue that it would be obvious for EP 760 to perform the input queue offload steps because, allegedly, "Dongare teaches that the embedded processor manages the input queues, e.g., through queue handling/scheduling, but leaves implementation details to the POSITA." Pet., 19, 28-29 ("Dongare's embedded processor manages the input queues. One part of queue management is tracking the amount of free space in the queue" (internal citations omitted)), 30-31 ("Dongare's embedded processor manages the input queue . . . .").

107. In my opinion, this is simply not true. Dongare does not disclose that EP 760 manages anything to do with input queues or managing the input queues, except that it may tag commands that have already been added to the input queues

and reorder commands that already reside within the output buffer. For example, Dongare contains Input Queue Handler 510 and Scheduler 530, but these blocks reflect the limited role of assigning tagged input segments to any one of a plurality of processing engines. These two blocks are the "earliest" portion of Dongare' system—that is, they are the closest to when commands are written to the input queue. But even these blocks only perform the limited role of assigning tagged input segments to any one of a plurality of processing engines. EX1005, 10:56–57.



EX1005, FIG. 5, 10:65–11:2. They have no role in determining when commands are issued to the input queue because, again, commands have already been added to the input queue by the time Input Queue Handler 510 "receive[s] a plurality of input streams." *Id.* Petitioners repeatedly overstate the role of EP 760 by suggesting it was placed on-chip "for precisely this type of management function," Pet., 31, but Dongare makes clear that its role is entirely different.

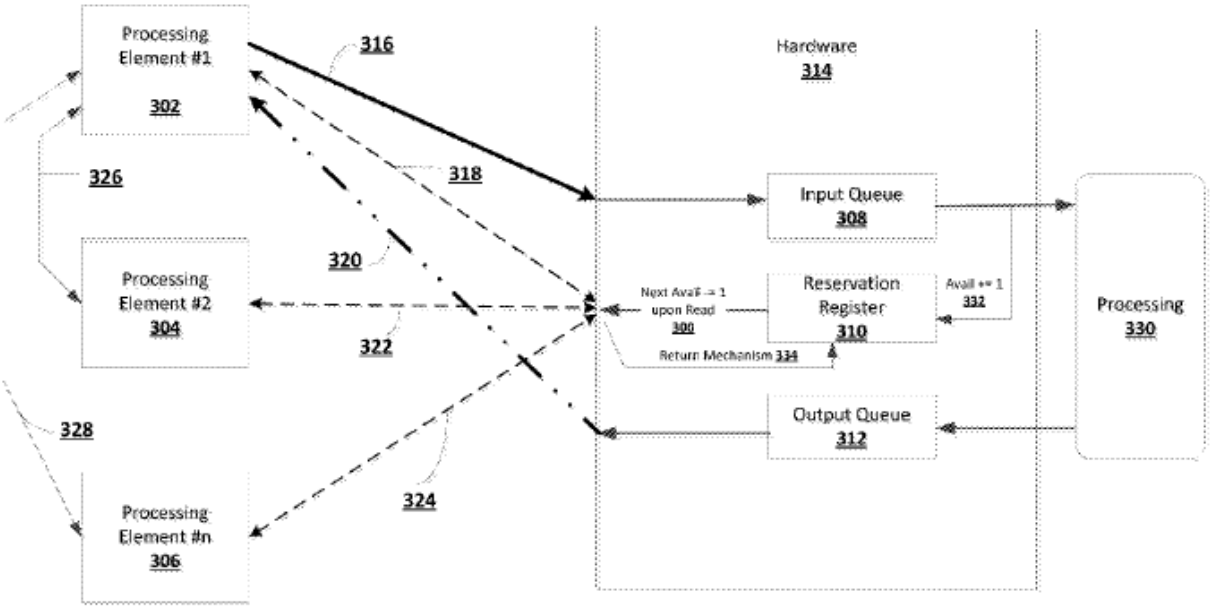
108. Petitioners' do not cite anything that supports their position. For example, Petitioners cite Dongare at 10:50–11:30, but this passage discusses that "[t]the input queue handler 510 is configured ***to receive*** a plurality of input streams," not that EP 760 manages the input queues on the front-end before commands are input by the host processor. Pet., 28.

109. Petitioners also cite Dongare at 14:30–55, but this passage likewise discusses that "***[t]he Host 710 writes commands*** for the RAID controller ***into the input queues 750***" and EP 760 only interacts with input queues that are full of commands. *Id.*

110. Even Petitioners' expert, Dr. Russ, agreed that this passage in Dongare means that the commands are written into the input queue ***by Host 710*** before EP 760 even access the input queues. EX2022, 100:2–101–7. In sum, there is no support in Dongare for Petitioners' core assertion that the EP 760 is in charge of queue

management such that it would have any role in tracking the amount of free space in the queue.

111. At best, Host 710 and EP 760 are in communication via the output queue. Pet., 31–33. But the '596 patent requires that EP 760 communicate with Host 710 before it issues commands to the input queue so it can offload queue-state management. Thus, any theory relying on output queues is backwards. EP 760 can add completion notifications into an output queue directed back to Host 710, but only after the commands have been executed. EX1005, 14:30–35. This is in contrast to the '596 patent, which depicts inter-processor link 326 between the first and second processors, and describes that "[t]he two processing elements communicate in some efficient way (such as through shared memory, etc.)." EX1001, 4:25–29.



EX1001, FIG. 3.

112. Third, Petitioners argue that "[i]t was obvious and natural for Dongare's embedded processor to perform a coordination role and read the reservation register due to its proximity to the queues and the register (everything is on-chip, avoiding cross-chip communication delays), in contrast to the host processor. Therefore, it was obvious and more efficient for Dongare's embedded processor to monitor the queue states via the reservation register and notify the host processor when there is space available for additional commands." Pet., 29–31.

113. In my opinion, there is no support for this argument. Petitioners are wrong that it would have been beneficial for EP 760 to perform the queue-state management. To start, EP 760 managing the queue-state does not make sense, since EP 760 reads the commands that were already written into the input queues. Moreover, in the '596 patent, the benefits of offloading to a second processing element include reducing latency and contention and improving synchronization. Shifting this responsibility to EP 760 would not further these benefits because EP 760 plays the central component in the RAID Controller ASIC 720, coordinating the input commands to be processed by RAID engine 730 and triggering output to be recognized by the host 710. These functions performed by EP 760 is highly time-sensitive and thus EP 760 is already burdened with the control and coordination of the RAID engines and reordering output segments. If EP 760 were to also take on the role of queue-state management, which is unnecessary since it is the host 710

that puts the commands into the input queue 740, certain time-sensitive processing that needs be performed by EP 760 may be delayed and latency could increase dramatically. Assuming the queue state management role by EP 760 can become bottlenecks on both the input and output sides, exactly what the '596 patent sought to avoid. It would also increase software overhead, introduce a scalability problem, and undermine the hardware-based synchronization advantages of the reservation register, without yielding a performance benefit.

114. Regardless, Petitioners are wrong that it would have been obviously beneficial for EP 760 to perform the queue-state management. In Dongare, if Host 710 performed the Input Queue Determination Process, there would be two off-chip communications: reading the RR and, when appropriate, writing the command to the input queue. If EP 760 performed the Input Queue Determination Process, there would still be two off-chip communications ***and also*** one additional on-chip communication: reading the RR (on-chip); when appropriate, notifying Host 710 to issue the command (off-chip); and Host 710 writing the command to the input queue (off-chip). Thus, using EP 760 to perform the Input Queue Determination Process in the context of Dongare would involve more communications (one on-chip communication and two off-chip communications versus just two off-chip communications) and be slower.

115. In fact, this hypothetical highlights the '596 patent's non-obviousness. For example, applying the '596 patent to Dongare would require ***more*** communications than prior art systems. In the context of Dongare, the obvious option is for the ***1PE*** to determine when to issue a command, since it is the processor that is writing that command. By adding a 2PE that has to determine when to issue a command and then also notify the Host to write that command, the '596 patent adds additional steps to the process, which inherently takes more time. But the inventors of the '596 patent realized there were unexpected benefits in reducing latency that made this sacrifice worthwhile. This unexpected benefit demonstrates that non-obvious nature of the '596 patent's solution.

116. A POSITA would not be surprised that Dongare and Gewirtz do not suggest a second processor performing the input queue offloading steps, as Dongare and the '596 patent are directed at different problems. Dongare's system for optimizing the performance of RAID engines (*i.e.*, the "third processing elements" in the '596 patent) is substantially different than the '596 patent's goal of offloading queue-state management from the first processor. *Compare* EX1001, 1:15–16 ("In the prior art, the processing load (of a processing element) to determine what command to issue is relatively high.") *with* EX1005, 1:25–32 ("Additional processing is then required to restore the order of the segments of the output stream

relative to the input stream resulting in a compromise of the efficiencies gained by multi-processing.").

117. The '596 patent's second processing element acts as an intermediary between the first processor and input queue so the first processor can make a simple input request without expending resources on queue-state management or overflowing the input queue when space is unavailable. EX1001, 1:38–59, 4:20–27. Dongare, on the other hand, discloses an entirely different tagging and ordering workflow where the second processing has no front-end communication with the first processor. EX1005, 1:14–17.

b. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Sending Notifications to a First Processing Element to Issue Commands, as Required by Limitations 1[i] and 1[j].

118. In my opinion, Petitioners do not show that Dongare and Gewirtz disclosed a system where the "first processing element receives notification from said at least one second processing element regarding issuing the command." EX1001, cl. 1. Neither reference even hints at using separate processors to: (1) issue commands (the first processor) and (2) notify the first processor when commands should be issued (the second processor). This division of labor is fundamental to claim 1 and is wholly lacking from the references

119. As I explained above, EP 760: (1) portions long commands into smaller command segments; (2) tags commands and command segments for reordering after

processing; and (3) reorders the commands after they have been processed. It does not read input queues to determine if space is available, schedule commands to be issued into the input queue, or notify the host processor that space is available.

120. Dongare does not disclose a second processing element that notifies a first processing element whether there is space in the input queue to issue a command. In fact, Dongare discloses that the Host 710 makes this decision on its own. EX1005, 14:36-42.

121. A POSITA would observe a disconnect between Dongare and Gewirtz and the ’596 patent when elements 1[i] and 1[j] are viewed in combination. Element 1[j] requires that the first processing element issues the command after it receives the notification from the second processing element. That makes sense as a matter of logic because the point of the ’596 patent is that the first processing element does not expend its finite processing power deciding when to issue a command. But in Dongare, Host 710 (the first processing element) issues the command before the EP does anything.

122. Thus, not only is there no suggestion that the second processing element notifies the first processing element when to issue a command, adding that role for EP 760 makes no sense. EP 760 has no role before the command is issued—that is fundamentally different from what the claim requires (the second processing element notifying the first processing element to issue the command before the first

processing element issues the command). EP 760 has a completely different role than the claimed second processing element, nor would it have been obvious or beneficial to task EP 760 with the responsibilities of the second processing element.

123. A POSITA would observe that Dongare teaches away from claim 1. EX1005, FIG. 7. The claims require that the second processing element be the one to determine when there is space in the queue for a new command and notify the first processing element. But there is no mechanism in Dongare for EP 760 to notify the first processor. Host 710 connects to the ASIC via a PCIe bus, but EP 760 does not use this PCIe bus to connect to Host 710.

124. Gewirtz teaches a register that sends "queue status interrupts" about input queue capacity, but there is no suggestion that EP 760 could pass these interrupts to Host 710. Pet., 32–33. In fact, Dongare indicates the ***exact opposite*** as the only interrupts that reach Host 710 are via ***output queues***; EP 760 may "signal the Host 710 that the command is complete by writing into the outbound queue 780," however, it does not write a similar signal in the input queue as it is only configured to optimize the RAID engines by segment tagging. EX1005, 14:52–55.

125. Even though Gewirtz's flow control register can generate "interrupts" to a programmable target based on queue status, a POSITA would not find it obvious for EP 760 to be the programmable target to use those interrupts to notify Host 710 whether to issue a command, simply because it sits on the ASIC. Even if it would be

obvious for it to receive Gewirtz's queue status interrupts (which it is not), EP 760 does not communicate with Host 710, nor does it manage input queue capacity. EP 760 would have no use for Gewirtz's queue status interrupts because they would not support its tagging and reordering of commands for the RAID engines.

126. Also, even if Defendants' proposed configuration were obvious, I do not believe it meets the claim language. Petitioners allege that "when an input queue is full—for example, when the queue is full and waiting on RAID Engines to complete pending commands—the embedded processor notifies the host after the input queue drains below a predetermined threshold. The host processor can then issue commands to the input queue." Pet., 33 (citations and footnotes omitted). But the claims require that the second processor notify the first processor when there is "available space in said input queue for said first processing element to issue a command" and then "notif[ying] said first processing element to issue said command to said input queue." EX1001, cl. 1. A POSITA would not understand that simply alerting the first processing element that there is some "predetermined" amount of space in the input queue—without knowing what "said command" is and thus whether there is room available for that particular command—meets claim 1.

127. I do not agree that Petitioners' generically alleging that "[t]he prior art provides ample motivation for one processor to notify another when space is available" supports that it would be obvious for EP 760 to conduct queue-state

management. Pet., 33. For example, Petitioners cite unasserted prior art, U.S. Patent No. 5,204,954, but the prior art does not provide ample motivation to reconfigure mismatched processors for entirely new roles. Pet., 33 (citing EX1011, 10:3–26, 11:52–54). To be clear, a processor being proximally close to another processor is not enough to show that it would be obvious to offload from one process to the other.

128. Petitioners' argument relies on misstating the purpose of EP 760. They argue that Gewirtz's queue status interrupts would "enable[] Dongare's embedded processor to perform its queue-management and scheduling role," but EP 760 has no role in input queue management Pet., 32–33. Host 710—not EP 760—"writes commands for the RAID controller into the input queues 750." Pet., 34 (citing EX1005, 14:41–42, 14:46–49; EX1003 at ¶¶145–46). The PCIe bus between Host 710 and the ASIC is not for communication between EP 760 and Host 710. EX1005, 14:30–35. Host 710 uses the PCIe bus to write commands directly to the input queue without communicating with EP 760.

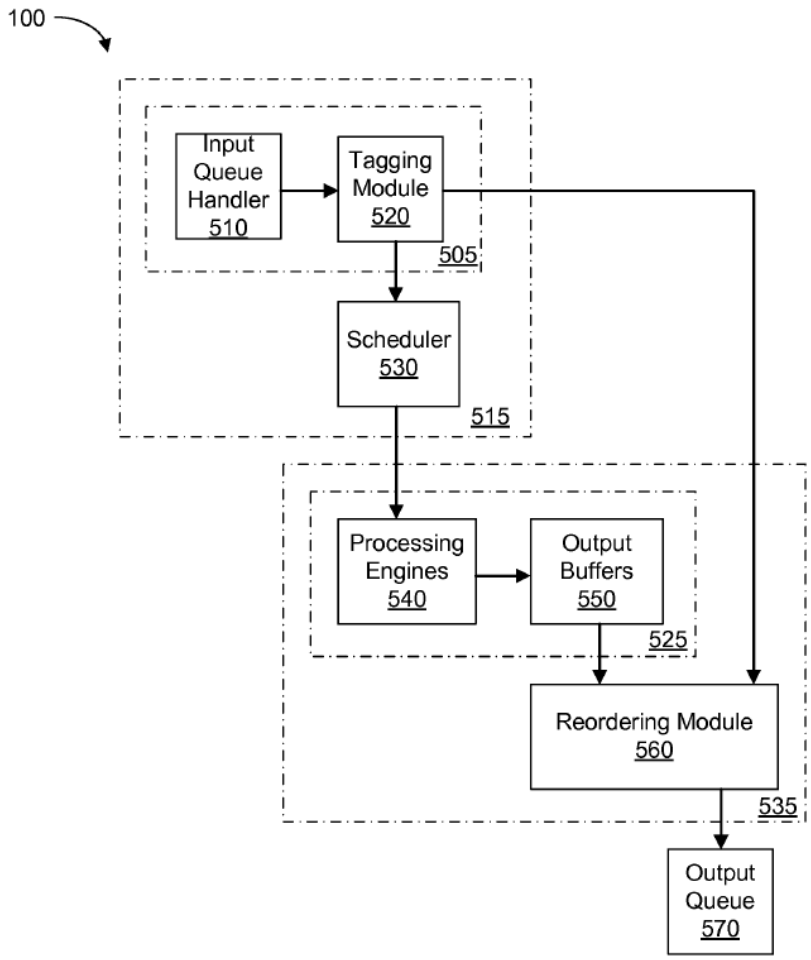
c. Dongare Does Render Obvious an "Input Queue Implemented in Hardware" As Required By Claims 1[a].

129. In my opinion, Petitioners have not shown that Dongare discloses "an input queue implemented in hardware." EX1001, cl. 1, 7 and 13.

130. I observe that Petitioners identify a single sentence in Dongare to support their argument: "[i]t is noted that the elements of the multi-engine processing

system 100 may be implemented as a software routine, hardware circuit, firmware, or a combination thereof." Pet., 15–16 (citing EX1005, 11:14–16). The remainder of their evidence only shows that Dongare uses input queues generally, without specifying whether they are in software or hardware. *See, e.g.*, Pet., 14–15 ("'FIG. 1 illustrates a multi-engine processing system 100 capable of handling a plurality of input queues[.]' Ex-1005, 5:3-36, 14:39-42 ('The queues 740 in the controller are classified as input queues 750 from the Host[.]'); Ex-1003, ¶¶84-87.").

131. A POSITA would understand that Dongare is silent on whether its input queues are in hardware. None of the evidence cited by Petitioners says anything about input queues. Dongare provides that "elements of the multi-engine processing system 100 may be implemented as a software routine, hardware circuit, firmware, or a combination thereof," but Dongare's multi-engine processing system 100 does not include the input queues. Pet., 15–16 (citing EX1005, 11:14–16) (emphasis added). Instead, "[m]ulti-engine processing system 100" only comprises "an input queue handler 510; a tagging module 520; a scheduler 530; a plurality of processing engines 540; a plurality of output buffers 550; a reordering module 560; and an output queue 570." EX1005, 10:51–56.



Id. at FIG. 5 (showing that multi-engine processing system 100 does not encompass the actual input queues). Each of these hardware components is separate from the input queues flowing into the system—Input Queue Handler 510 and Tagging Module 520 are only responsible for tagging packets ***after*** they are placed in an input queue. EX1005, 10:56–62.

132. A POSITA reading the disclosure in Dongare would understand that the input queues 110 are implemented in software. This is because, as described in FIG. 3 of Dongare, the received input segments are tagged (by Tagging Module 520

shown in FIG. 5), wherein the "identity tag includes information indicating the input stream and the relative order within the input stream of each of the plurality of input segments." (Id. at 7:48-50.) If the input queues 100 were implemented as hardware queues, then the ordering information would not be necessary since the segments from the same input stream would be taken out of the input queues in sequence of their receipts and the engines in the multi-engine 130 would process them in sequence preserving the original sequence of the segments. The mere fact that a system includes hardware circuits elsewhere does not mean that it would be obvious for its queues to be implemented in hardware. A POSITA would assume that Dongare's queues are software queues because, first Dongare does not discuss the available space in the queues and second software queue provide more flexible utility to processing packets out-of-sequence for later reordering, by appending identity tags to the segments. Moreover, software queues essentially do not have fixed sizes as the size can increase or decrease using the head and tail pointers, while hardware queues have fixed sizes. Because the reference does not disclose any information about the size of its queues, a POSITA would not understand that Dongare's queues are hardware.

133. The read/write arrow between Host 710 and the Queues 740 is only an indication that the host can enqueue read/write commands to the input queues—it does not indicate that the queues are in hardware. In fact, Figure 7 would look

identical if the queues were implemented in software. The fact that an input queue is on-chip is not indicative that it is in hardware. ASICs may contain physical modules that contain queues created in software using head and tail pointers. Even in cases where a first processor is connected to a chip with a physical PCIe bus, the input queue may still reside on software within the chip because chips include their own CPUs, like ARM cores, which house the input queues in software.

2. Independent Claims 7 and 13

a. Limitations 7[a], 7[e]–7[i] and 13[a], 13[e]–13[i] Fail for the same reasons as equivalent in claim 1.

134. In my opinion, as shown above, Dongare and Gewirtz do not anticipate or render obvious limitations 1[a], 1[e], 1[g]–[j]. For the equivalent limitations in claims 7 and 13, Petitioners simply cite back to the portions of the Petition concerning claim 1 without any additional evidence, analysis, or argument. Pet., 37–38. Thus, Petitioners fail to show that elements 7[a], 7[e], 7[h]–7[i] and 13[a], 13[e], 13[h]–13[i] would be obvious for the same reasons discussed above.

b. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Receiving a Command from the First Processing Element and Issuing a Command to the Input Queue on Behalf of the First Processing Element, as Required by Limitations 7[g]/[j] and 13[g]/[j].

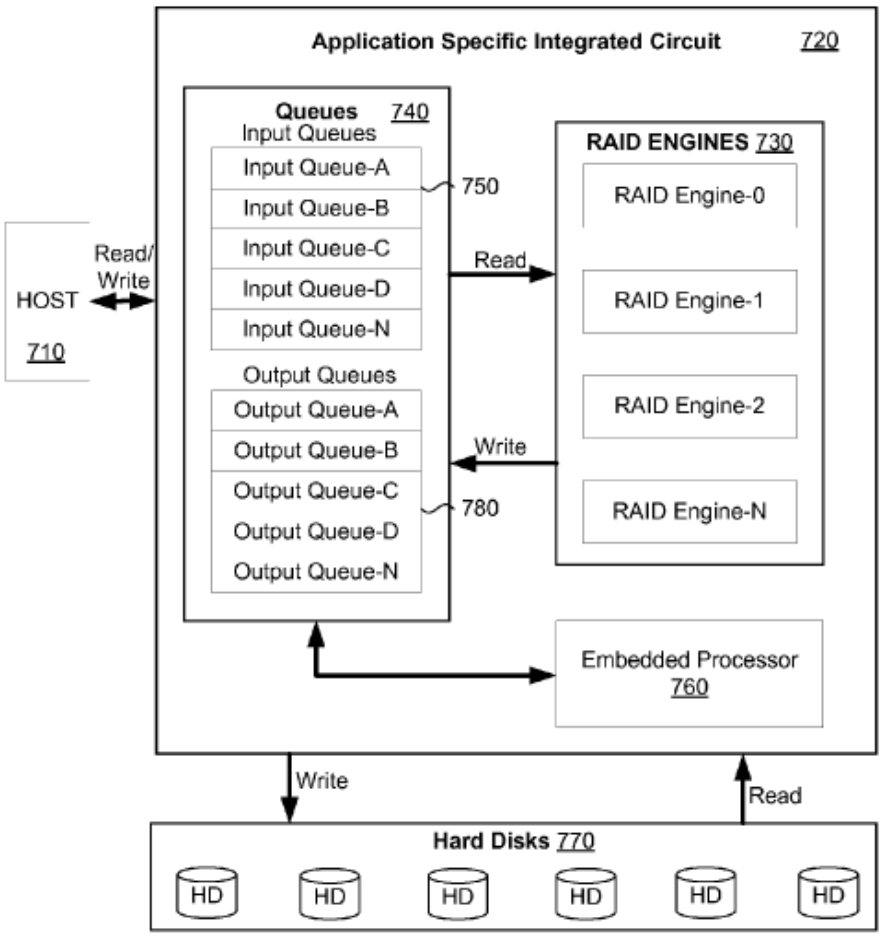
135. In my opinion, Petitioners do not show that the first processing element sends the command to the second processing element, which then issues the command "on behalf of" it once it determines that space is available in the input

queue. As explained above, Petitioners' alleged "second processing element"—EP 760—only comes into the picture after the command has been issued to the input queue. Petitioners' hypothetical configuration in which EP 760 receives a command from the first processing element and then issues that command on behalf of the first processing element is not disclosed anywhere in the references.

136. Petitioners allege that "Host 710 writes commands for the RAID controller," and EP 760 "receives [it] . . ., tags it for scheduling purposes, and then writes (issues) the tagged command to the input queue on behalf of the host, for execution by one or more RAID engines." Pet., 38–39 (citing EX1005, 14:41–44, FIG. 7). I disagree with these arguments for multiple reasons.

137. First, the second processing element does not "receive[] a command from said first processing element." Instead, Dongare discloses that Host 710 writes a command to the input queue and then EP 760 can read the commands ***inside*** the input queue to add tags to those commands. At most, then, the first processing element is sending commands to the input queue—not to EP 760.

138. This is shown in particular by Figure 3. Dongare discloses that Host 710 only accesses the ASIC 720 via PCIe bus, which it uses to write commands into the queues. Dongare only discloses Host 710 and EP 760 communicating via the output queue. There is no way for Host 710 to send a command to EP 760 consistent with Dongare:



EX1005, FIG. 7.

139. Second, EP 760 does not "issue[] said command to said input queue on behalf of said first processing element." EX1001, cl. 7, 13. Petitioners rely on the following disclosure from Dongare: "The input queues 750 are read by the embedded processor 760 and commands are created for RAID engines 730 to process data. Newly generated commands are loaded to input queues 750 dedicated to RAID engines 730." Pet., 38. But Dr. Russ even testified that this passage only discloses EP 760 tagging and potentially dividing the "existing commands," not creating and issuing new ones:

Q. Does the embedded processor create commands?

A. The words on the page are commands are created for the RAID engines, and so I acknowledge the words on the page.

I think read in context, ***it's clear that [the Embedded Processor] is simply tagging, and potentially dividing the existing command***, because the alternative would be really, really weird for a disk controller."

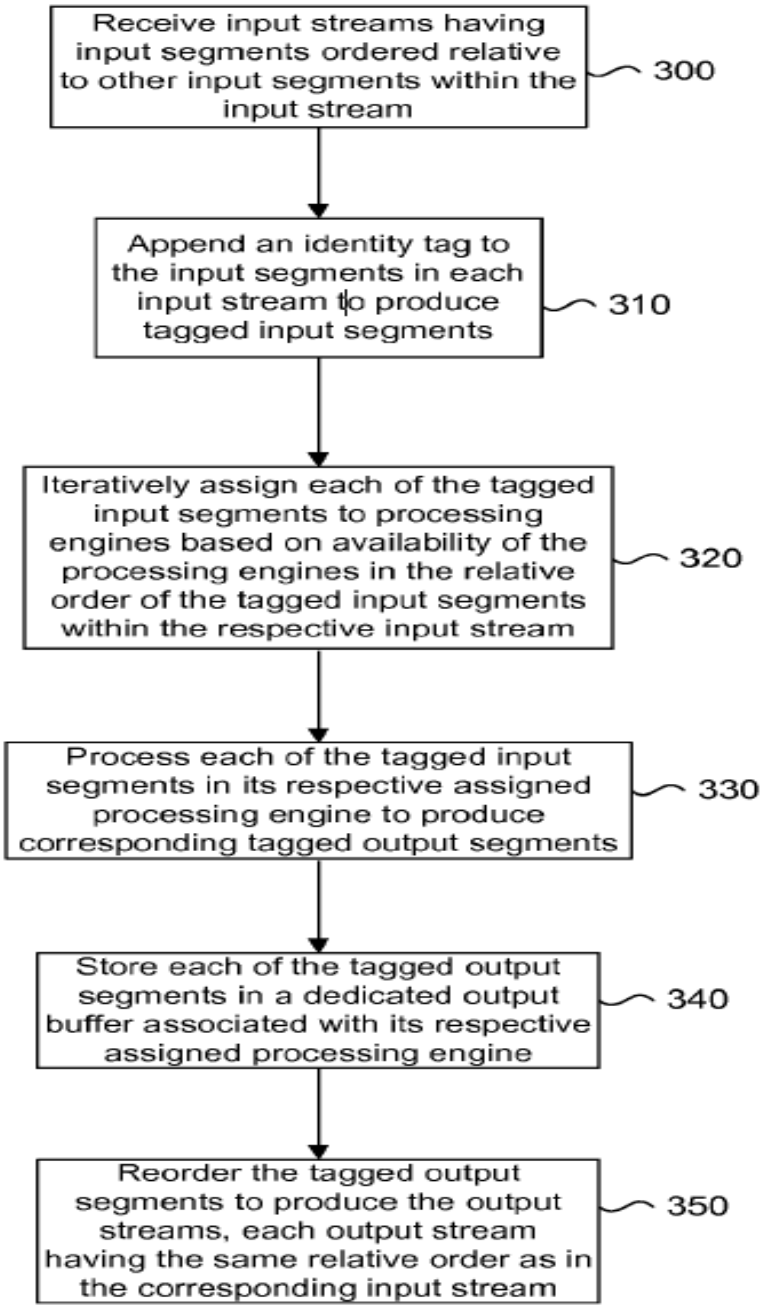
EX2022, 108:5-13 (emphasis added). When asked what this passage means, Dr. Russ testified that in Dongare:

> ***[T]he Host is writing a command, and it gets put into the input queue, and the embedded processor performs processing on that command***. As we described earlier, for example, it performs a tagging operation, which is important for making sure commands are issued in the correct order. And that processor can also divide a command into portions. ***And so what the words on the page say is 'newly generated commands,' and of course it can't mean something like completely different***. That just wouldn't make any sense if you're writing files to a hard drive. ***What it actually means is it's the commands that have been divided into portions and tagged.***

Id. at 104:5–105:8 (emphasis added). In other words, Dongare discloses the complete opposite of claims 7 and 13: Host 710 writes the command into the input queue, not EP 760.

140. In my opinion, Petitioners' arguments are internally inconsistent. When summarizing the workflow in claim 7[j], Petitioners admit that "[t]he input queues are read by the embedded processor 760," meaning that at the point the commands

are received by the second processing element, they have already been added to the input queue by the first processing element. Pet. 38–39 (emphasis added). Figure 3 illustrates this as it describes how input segments are received by EP 760 only after they have been added to the input stream.



EX1005, FIG. 3. This contradicts the structure of the '596 patent, where the second processing element assists the first processing element with adding the commands to the input queue. EX1001, 6:36–40.

141. Third, Dongare does not disclose that EP 760 "issues said command to said input queue on behalf of said first processing element" after "determining [] said read value indicates available space in said input queue to issue said command." Even if EP 760's tagging and portioning of commands was considered to be issuing a command on behalf of the first processing element (it is not), it is nonsensical to say that it would occur after EP 760 reads the value to determine whether there is space available in the input queue. In Petitioners' hypothetical, the command is already in the queue from the Host. Even if EP 760 re-issues the command to a different input queue for the RAID engines, there must already be space in the queue for the command, because it was already written there by Host 710.

c. Dongare and Gewirtz Did Not Render Obvious a Second Processing Element Receiving a Response Corresponding to Said Command from Said Output Queue, as Required by Limitation 13[k].

142. In my opinion, Petitioners do not show that EP 760 receives a response corresponding to Host 710's command from the RAID engines. EX1001, cl. 13. EP 760 continually monitors output buffers—it does not receive a response from any output queue. EX1005, 14:49–51. EP 760 only tags and reorder commands, so it would not receive a response to a question it never asked. *Id.*

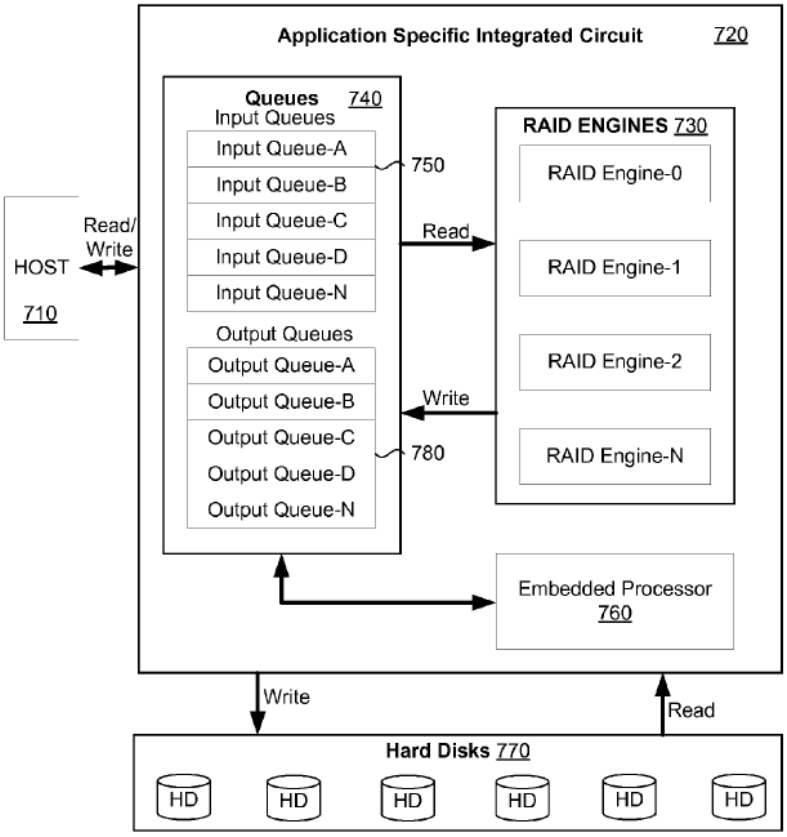
143. EP 760 "comprises a sequencer controller configured to repeatedly scan the plurality of dedicated output buffers to identify tagged output segments according to their relative order within each input stream to generate the plurality of output streams." EX1005, 7:65–8:3 (emphasis added). EP 760 is not notified by the output queues after a task is finished, but rather each tagged output from the RAID engines is stored "in a temporary write out buffer . . . as it is identified; and writing out the output stream having the same relative order of output segments as in the corresponding input stream." *Id.*, 8:4–8.

144. Thus, in my opinion, Dongare's output streams are not written until after the commands are reordered by EP 760, so there is no output queue for EP 760 to receive a response before it begins its work. *Id.*, 8:9–13.

145. In my opinion, Dongare is clear that "[a]fter processing of commands, each engine writes the processed commands in its dedicated command output buffer" where they "wait for write out until [] selected by the sequencer." EX1005, 12:30–33. EP 760—containing the sequencers—will then "start scanning expected sequence numbers for each queue to each element in the respective command output buffer" where a command may be "selected by the sequencer for write out." *Id.*, 12:50–55.

146. EP 760 can "signal the Host 710 that the command is complete by writing into the outbound queue 780," but this is only ***after*** EP 760 scanned the output buffer for output of the RAID engines. *Id.*, 14:52–54.

147. There is no evidence that EP 760 receives a response from an output queue, nor do they contend that it would be obvious for it to include this functionality. Pet., 40. Figure 7 does not depict the system sending responses to a second processing element, only that EP 760 connects to module 740 (which Dongare does not otherwise define):



Id., FIG. 7. In fact, a POSITA would not expect EP 760 to receive a response corresponding to the command because it did not add the command to the input queues on behalf of another processing element.

3. Dependant claims

a. Dongare and Gewirtz Did Not Render Obvious the Second Processing Element Receiving the Stored Value from the Reservation Register as Required by Claims 5, 11, and 17.

148. In my opinion, Gewirtz does not teach that "the stored value" is returned to the second processing element.

149. Petitioners provide almost no argument on these claims; they simply allege that Gewirtz discloses these claims because it teaches that "'[t]he available reservation count is maintained by hardware: it is decremented when a successful reservation read request is processed.'" Pet., 36 (citing EX1006, 4:56–5:6).

150. A POSITA would not understand Gewirtz to return a numerical value of available space in the queue to the second processing element—it only teaches a system where the reservation register returns a pass/fail value to the host processor:

> The queue hardware can return a pass/fail value to the agent in response to the read, depending on whether there is a reservation available, which may be specified via a reserved field (not shown) of state register 106. When an agent receives a pass value, it can perform a write to one of the data access registers. If it receives a fail value, it must attempt to obtain a reservation again

EX1004, 4:59–67; 4:26–29 ("Non-blocking reads may return a fail value to the requesting agent, whereas blocking reads may be queued in the queue element storage space (e.g., memory 103) in the order that they are received."). This evidence is uncontroverted. Petitioners only contend that Gewirtz teaches that the space counted is stored in the Queue State Register and decremented upon a successful read request. Pet., 36; EX1003, ¶154–57 (only discussing the decrement logic, not that the value is sent to another processor).

151. In my opinion, there is no evidence that Dongare or Gewirtz teach this limitation. Petitioners discuss elsewhere that "Gewirtz's Queue Status Register comprises a 'reservation field' for 'specify[ing] the number of entries available for writing into the queue'" and "before entries are written to the queue, a processor 'first reads the reservation register to see if an entry is available,'" but this does not show that the value of available space is returned to the second processing element. Pet., 30 (citing EX1006, 4:12–13, 4:56–64; EX1003, ¶¶133–34.

152. Gewirtz is not designed to offload processing tasks from the first processing element. In fact, Gewirtz states that it does not return the value of available space: an attempted reservation that results in a failure will require the agent to try again, rather than scheduling when more space is available. EX1006, 4:61–67.

153. In my opinion, it is important for the second processing element in the ’596 patent to obtain the stored value upon a read request because: (1) this is how the second processing element determines there is enough space to add a command to the queue; and (2) it allows the second processing element to make policy decisions and conduct the “peek” operation described in the specification.

154. The second processing element determines whether there is space available in the queue by comparing the value of available space to the size of the command from the first processing element. If the 2PE did not receive the value of available space, it would necessarily require “attempting” to issue the command only to observe that it failed—this is the exact opposite of the ’596 patent, where commands are not issued unless the processor confirms that there is available space and the command will not overfill and corrupt the queue. Further, by returning the stored value to the second processing element instead of a simple pass/fail result, the ’596 patent enables advanced scheduling and load-balancing behavior.

155. The inventors of the ’596 patent invented a system where the second processing element acts as an intermediary between the first processing element and input queue and all queue-state management functions are offloaded from the first processor. EX1001, 2:5–6 (explaining that the second processing element “reads the stored value,” not simply receives a confirmation from the reservation register). Part of acting as an intermediary in the disclosed embodiments is having full insight into

the available space in the reservation register. *Id.* The specification describes how the second processing element can use this value:

> In the event there is insufficient space to satisfy the request the reservation register may be configured to reserve however much space is available or to not reserve any space at all. At times there may exist a need to test if a reservation would succeed without actually making a reservation, a "peek" at the reservation register. This peek may be implemented via an alternate, mirrored, reservation register that mirrors the value of the reservation register that can be read by a processing element, but does not actually make a reservation, whereby the alternate register does not automatically modify the available free space. Any number of processing elements may read these registers (hardware synchronizes these reads internally to ensure they do not overlap using standard arbitration for multiple processing elements). Thus there is a guarantee of not "over subscribing" the command queue.

Id. at 3:49–64.

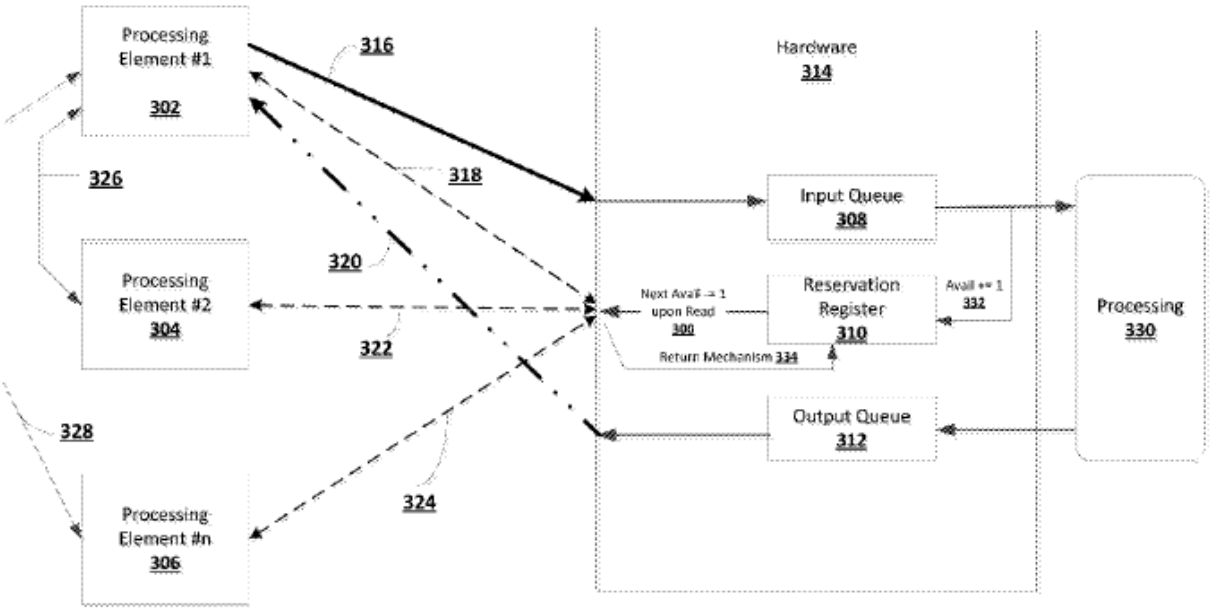
156. If the reservation register only provided a pass/fail result in response to a reservation request, the second processing element would not be able to "reserve however much space is available" depending on the size or importance of its request. *Id.* at 3:49–52.

157. Further, if the reservation register is only capable of returning a pass/fail result to the second processing element—as is the case for Gewirtz—there would be no way to "peek" at the available space. *Id.* at 3:52–64.

b. Dongare and Gewirtz Did Not Render Obvious a Third Processing Element Notifying the Output Queue to Issue the Response as Required by Claims 6, 12, and 18.

158. In my opinion, Petitioners do not show that a third processing element that notifies the output queue to issue said response. Dongare instead assigns this task to the second processing element. EX1005, 14:52–54.

159. A POSITA would understand that Dongare teaches a different relationship between the first, second, and third processing elements than the '596 patent. The '596 patent discloses a third processing element which executes the commands from the input queue and then writes a confirmation back to the first processing element via an output queue. EX1001, 7:51–44.



Id., FIG. 3. Claims 6, 12, and 18 expressly require that it is the third processing element that "notifies said output queue to issue said response" to the first processing element. EX1001, cl. 6, 12, and 18; 5:56–59.

160. In contrast, Dongare discloses that RAID engines (*i.e.*, the third processing element) process the commands, but EP 760 (*i.e.*, the second processing element) "monitors output queues" from the RAID engines and "will signal the Host 710 that the command is complete by writing into the outbound queue." EX1005, 14:49–54. Thus, the RAID engines write processed commands into output buffers, which are only accessible to EP 760, and then EP 760 notifies the first processing element of the completion via an output queue. *Id.*, Abstract ("Processed commands are stored in dedicated command output buffer associated with each engine temporarily. The processed commands are re-ordered while writing out.").

161. Petitioners and Dr. Russ do not explain these differing relationships or why a POSITA would see these different relationships and find the '596 patent's configuration obvious.

162. Dongare teaches a system where EP 760 monitors the output queues because the invention is only a tagging system used to optimize downstream processing. EX1005, 11:9–11. EP 760 notifies the host processor via an output queue because its purpose is to reorder the completed commands after processing before they reach their next destination. *Id.* 11:9–11.

163. It would not make sense for the RAID engines (*i.e.*, the third processing element) to notify Host 710 via an output queue because the processing is not complete until after EP 760 conducts the reordering. Dongare does not disclose any output queues connected to the RAID engines at all, as the processed commands go directly into an output buffer where they sit until a different processing element moves them along.

4. Ground 2 Does Not Address The Problems In Ground 1.

164. In my opinion, Matsunami does not address the deficiencies in Ground 2. In Ground 2, Petitioners rely on Matsunami only for the limitations recited in claims 3–4, 9–10, 15–16. Petitioners do not rely on Matsunami to teach or suggest any of the limitations challenged above.

B. Petitioners Fail to Prove their Combination of Khawand, Rosenthal, Ohkawa, and Matsunami (Grounds 3, 4, and 5) Discloses or Renders Obvious Claims 1–4, 6–10, 12–16, 18.

165. In my opinion, Khawand, Rosenthal, Ohkawa, and Matsunami also did not disclose or render obvious claims 1–4, 6–10, 12–16, 18 of the '596 patent. Specifically, Khawand and Rosenthal are architecturally incompatible and neither teaches anything resembling the claimed architecture.

166. A POSITA would combine Khawand and Rosenthal because they are architecturally incompatible. Rosenthal states that its queue-state management must be done by the same processor that writes the command into the input queue (*i.e.*,

the 1PE). However, Petitioners contend that Khawand's purported queue-state management is conducted by IPM 308, which they map to the 2PE. Rosenthal prescribes that it could not be combined with Khawand, and its single-processor system teaches away from the multi-processor architecture of the '596 patent.

167. Khawand and Rosenthal do not teach or suggest the novel architecture of the '596 patent. Khawand discloses a system for delegating a command from one processor to another, but it does not disclose that a 2PE manages a shared input queue on behalf of a 1PE. Although a 2PE can route suspend/resume commands between processors, there is no concept of issuing a command "on behalf of" another processor, nor a shared hardware RR accessible by multiple processors or output queues returning results to the originating processor. Rosenthal, meanwhile, fails because its flow-control register (what Petitioners map to the RR) must be accessed by the same processor that writes data into an input queue (*i.e.*, the 1PE). There is no 2PE reading the register, no offloaded decision-making, and no issuing of commands on behalf of another processor.

1. Claim 1

a. The References Teach Away From Petitioners' Proposed Combination

(1) Rosenthal Teaches Away From Petitioners' Proposed Combination for Limitations 1[e]–1[j].

168. As a preliminary matter, it is my opinion that the Petition fails to show that a POSITA would combine Khawand and Rosenthal. Rosenthal explicitly teaches away from Petitioners' proposed combination because it states that processers like IPM 308 cannot access Rosenthal's register

169. Petitioners theorize that a POSITA would have applied Rosenthal's teachings to Khawand because Khawand's IPM 308 evaluates "flow control criteria" and Rosenthal's Flow Control Register 45 "indicates the amount of space remaining in the queue." Pet., 57. But while the Petition identifies IPM 308 as the 2PE responsible for conducting the queue-state management, Rosenthal teaches that the queue-state management ***must*** be done by the ***same*** processor that writes or executes the command (*i.e.*, not the 2PE).

170. Rosenthal describes a system in which a CPU (a type of processor) can run multiple different applications programs, each of which has an associated input buffer. EX1009, 2:18–33. Rosenthal explains that the CPU running the application program can access a "flow control register" to determine whether there is sufficient space in the input buffer for new commands. *Id.*, 16:33–49. If there is, the application

program running on the CPU can then write new commands to the input buffer. *Id.*, Abstract, 2:21–26

171. Rosenthal specifies that its version of a RR is only for systems where the RR would not be accessible by Khawand's IPM 308. Rosenthal states that access to input/output devices is limited to the operating system of the CPU, and that ***only*** the CPU can access and read from the input/output devices, including the "flow control register." EX1009, 1:10–28, 16:44–46 ("This it does by having the ***central processing unit*** read the value in the flow control register 45 associated with the FIFO buffer 39 being utilized." (emphasis added)). Rosenthal repeatedly states that the "***flow control register [] must be read by the processing unit running the application register***." *Id.*, Abstract, 2:21–26 (emphasis added). When discussing reads and writes to the input queues, Rosenthal states that the purpose of the RR is to improve the functioning of "the central processing unit 21." *Id.*, 4:6–11. In other words, Rosenthal requires that the processor writing commands (the CPU) is also the only processor that can read from the flow control register.

172. Thus, a POSITA would not be motivated to combine Rosenthal with Khawand because Rosenthal specifies that Khawand's IPM 308 could not be the processing unit accessing the RR. In this sense, Rosenthal also teaches away from the '596 patent because it mandates a single-processor system.

173. Because the alleged motivation to combine does not address that the two references are incompatible and teach away from the ’596 patent, there is no evidence that a POSITA would have reasonably expected that adopting Rosenthal’s flow control as part of Khawand’s load balancing would result in a system that meets any claim limitation.

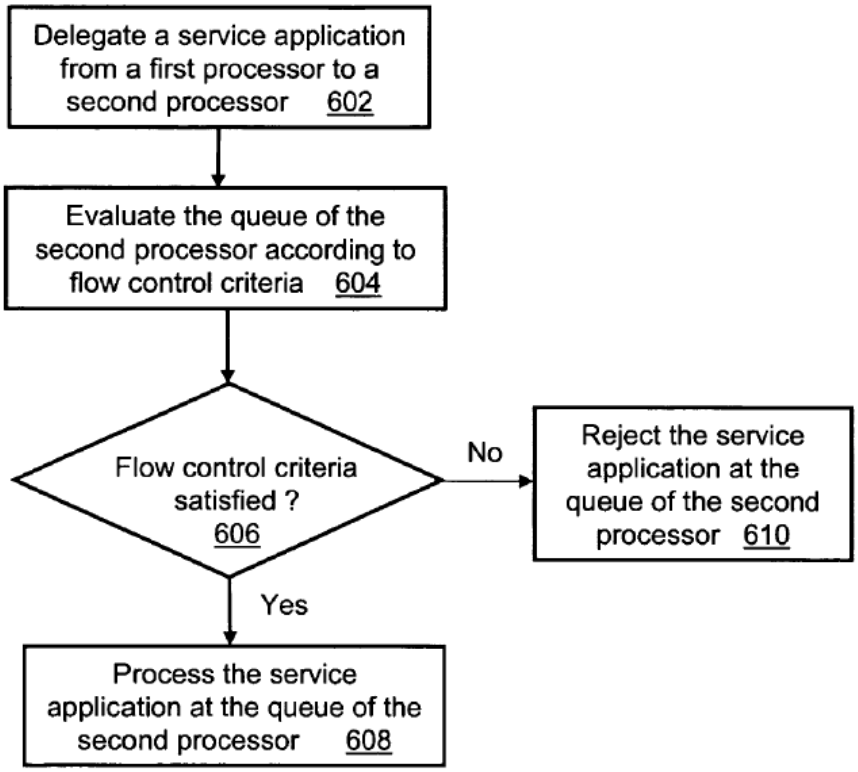
(2) Rosenthal Teaches Away From Petitioners’ Proposed Combination for Limitations 1[e]–1[j].

174. In my opinion, even if a POSITA would have been motivated to combine Khawand and Rosenthal in general, a POSITA would not have been motivated to combine them as Petitioners propose for limitation 1[h] because Khawand teaches away from the combination.

175. Khawand discloses its own, “divergent” approach that is different than that described in Rosenthal and claimed in the ’596 patent. Khawand thus teaches away from Petitioners’ combination and the ’596 patent.

176. Khawand’s first step is Step 602, in which the “first processor” (the alleged 1PE) “delegate[s] a service application from a first processor to a second processor [the alleged 3PE].” *Id.*, Fig. 6, 3:65–4:2. Dr. Russ explained that this means that “the first processor writ[es] [the application] to the queue of the second processor.” EX2022, 139:11–21. After Khawand’s Processor 1 writes the application to the queue of Khawand’s Processor 2, IPM 308 evaluates Processor

2's queue and determines whether the flow control criteria is satisfied (in Steps 604 and 606).



EX1008, FIG. 6.

177. Khawand teaches that its "flow control criteria" is based on the "capacity threshold," which "indicates the capacity of the queue 404 for processing service applications." *Id.*, 4:15-18. When the queue is over the "capacity threshold," the service application is rejected and a "suspend command" is issued so that no more service applications can be put into the queue; when the queue falls below the "capacity threshold," a "resume command" is issued, and more service applications

can be placed in the queue.[1] *Id.*, 4:18–67. Khawand discloses that the capacity threshold can be static or variable and describes a complex method for determining a variable threshold. *Id.*, 4:28–59. But for both static and dynamic capacity thresholds, IPM 308 simply determines whether the queue is above or below the current capacity threshold.

178. Khawand affirmatively discloses a "determination" that is based solely on: (1) how full the queue is; and (2) what the capacity threshold is. Notably, this method of "determining" does ***not*** account for any particular command and thus does ***not*** determine if there is enough space available in the queue for that particular command, as required by Petitioners' combination and the '596 patent.

179. Khawand's approach is fundamentally different than that taken by the '596 patent. In Khawand, IPM 308 only determines whether there is sufficient space *in general* to re-open the queue (based on the capacity threshold). Indeed, much of Khawand is concerned with how best to calculate a capacity threshold, and that calculation is only necessary because of Khawand's alternative approach.

180. There is no need for a "capacity threshold" in the '596 patent because either there is enough room in the queue for a particular command or there is not.

[1] Although Khawand refers to "suspend ***commands***" and "resume ***commands***," Petitioners do not map these concepts to the claimed "commands" issued to the input queue. To avoid confusion, this Response always refers to these terms with quotation marks: "suspend command" and "resume command."

But in Khawand, the IPM 308 is "determining" whether there is enough room available in the queue to safely reopen the queue for new applications in general, without reference to any command. Thus, Khawand is concerned with calculating how much availability a queue must have in general for it to safely resume allowing new service applications (the "capacity threshold"). And this alternative determining is not a peripheral teaching; it is central to Khawand and is even explicitly required by each claim. *See, e.g.*, EX1008, cl. 1.

181. Khawand's alternative, capacity threshold determining represents "a direction divergent from the path that was taken by the applicant" that teaches away from the '596 patent

b. In Any Event, Petitioners' Proposed Combination Does Not Render Obvious Claim 1.

182. In my opinion, even if a POSITA would be motivated to combine Khawand and Rosenthal—despite their vastly different architectures and purposes—Petitioners do not address that such a combination does not render obvious the '596 patent.

183. First, neither reference teaches or suggests a second processing element notifying a first processing element to issue a command.

184. Second, neither reference teaches or suggests that the first processing element accesses an "output queue" and "receives a response corresponding to said command from said output queue."

(1) Limitations 1[i]–1[j] Are Not Obvious in Light of Khawand and Rosenthal.

185. In my opinion, Petitioners' combination of Khawand and Rosenthal does not render obvious a second processing element notifying a first processing element to issue a command.

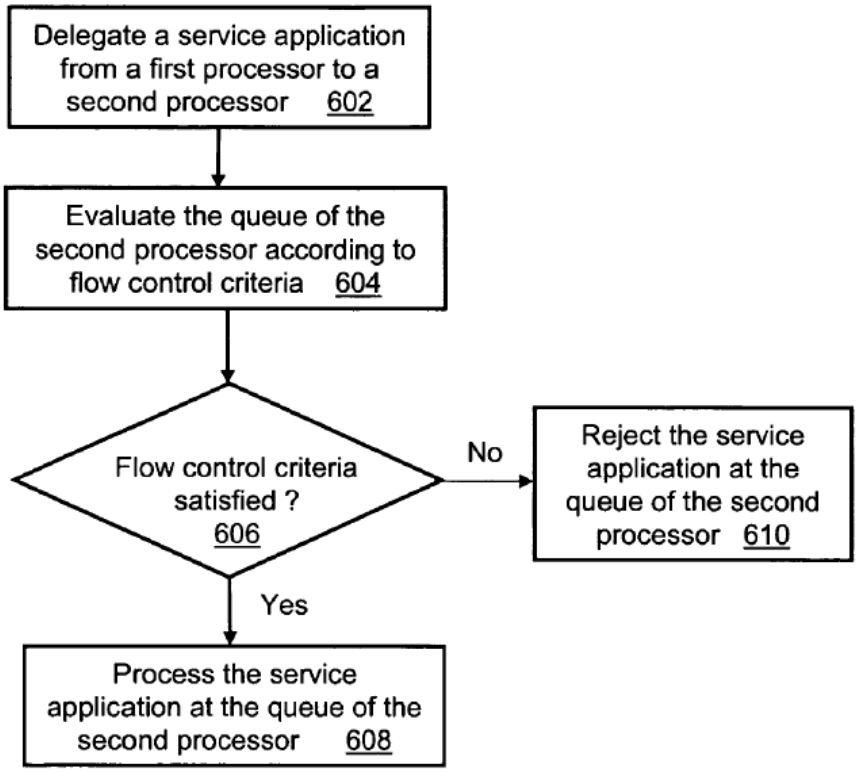
186. I understand that Petitioners allege that "Khawand teaches, or at least suggests, that [IPM 308] notifies the first processor to issue the command to the input queue, when the queue is no longer full and is available to receive service applications." Pet., 64. In particular, Petitioners argue that "Khawand's Processor 2 [the claimed "third processing element"] sends a suspend command to [IPM 308] [the claimed "second processing element"] when its queue is full, indicating that the associated queue is not available" and "[w]hen the input queue becomes available again, [IPM 308] notifies the first processor of the availability through a resume command." *Id.* at 63. Petitioners contend that this "resume command" is what "notifies" the 1PE to issue the claimed "command."

187. Based on my analysis, a "resume command" does not "notif[y] said first processing element to issue said command to said input queue" and Khawand's Processor 1 does not issue "said command." As discussed above, IPM 308 does not notify Processor 1 to issue a particular command, but instead issues a "resume command" that is a traffic signal that allows Processor 1 to resume writing to the input queue.

188. The differences between a "resume command" and the claimed "notif[ying]" are manifest. For example, Khawand does not disclose that IPM 308 has any knowledge of a particular command and thus IPM 308's "resume command" cannot notify 1PE to issue "said command." For another example, Processor 1 might have no commands pending, in which case it will not issue any commands after the "resume command." The fact that Processor 1 may not issue any commands after the "resume command" further demonstrates that IPM 308's "resume command" does not "notif[y]" Processor 1 to issue a command. These issues reflect a fundamental disconnect between Khawand and the '596 patent: IPM 308's "resume command" only informs Processor 1 that it is allowed to issue any commands it wants, it does not "notif[y] said first processing element to issue" a command. Accordingly, Processor 1 does not "issue[] said command," as there is no "said command." In sum, nothing in either reference suggests that IPM 308 notifies Processor 1 to issue "said command."

189. Khawand also makes clear that IPM 308 is incapable of notifying Processor 1 to issue "said command" because it only evaluates the queue occurs after Processor 1 has already written the command. As shown in Figure 6, Khawand's Processor 1 writes the application to the queue of Khawand's Processor 2 as the first step of the process (Step 602)—before the IPM 308 has done anything, and certainly

without first receiving approval from IPM 308 that there is space available in the queue:



EX1008, FIG. 6.

190. It is only after Khawand's Processor 1 writes the application to the queue of Khawand's Processor 2 that IPM 308 "evaluates the queue" and determines whether the flow control is satisfied. That means that, at most, IPM 308 can drop a command from an input queue. That is vastly different than determining whether to notify Processor 1 to issue the command in the first place.

191. This distinction is important because Khawand's configuration undermines a key objective of the '596 patent. By introducing the 2PE that manages

input queues on behalf of the 1PE using a hardware RR, the ’596 patent ensures that input queues never overflow with commands exceeding the available space.

192. The inventors of the ’596 patent designed a mechanism to prevent the 1PE from issuing a command larger than the remaining space in the input queue. Without this safeguard, such a command could be dropped or corrupt other commands in the queue. The ’596 patent achieves this by measuring each command against the actual remaining space in the input queue before issuance.

193. Khawand’s flow-control mechanism only operates at a level of generalized queue availability. Processor 2 evaluates whether its queue is above or below a certain ***threshold*** and then issues a suspend or resume signal to IPM 308. But this suspend or resume signal is not based on the size requirement of the specific command that Processor 1 seeks to delegate—the signal only conveys whether the queue is above or below its defined threshold at a given time. Nor does it convey that space for the next service application has been reserved, as there is nothing to prevent the next service applications from immediately exceeding the capacity if it is larger than Processor 2 expects.

194. In fact, Khawand expressly tolerates this imprecision by allowing Processor 2 to reject a service application after delegated, confirming that its design is retrospective. Dr. Russ could not opine on what it meant for a service application

to be "rejected," EX2022, 144:4–5, but a POSITA would understand it to mean the command is dropped, which would never happen in the '596 patent system.

195. Even if IPM 308 could read Flow Control Register 45 to determine the exact amount of free space in a queue, Khawand does not teach communicating this information back to Processor 1. There is no teaching that Processor 1 or IPM 308 may compare queue capacity to the size of the pending service application. Petitioners contend that IPM 308 could access Flow Control Register 45 and read the number of available spaces in the associated FIFO buffer, Pet., 63, but this value is useless without also knowing the size of the pending service application.

196. Petitioners allege that IPM 308 "sends a suspend command to the first processor when the input queue is full, followed by a resume command when space becomes available," but without having insight into the size of the pending service application, IPM 308 is not able to notify Processor 1 whether there is space to issue "said command."

(2) Limitations 1[c] and 1[j] Are Not Obvious in Light of Khawand and Rosenthal.

197. In my opinion, Petitioners also have not shown that the 1PE has access to an "output queue" and "receives a response corresponding to said command from said output queue." EX1001, cl. 1.

198. Petitioners map Khawand's Processor 1 to the claimed 1PE and Processor 2 to the claimed third processing element. Pet., 65. Khawand discusses

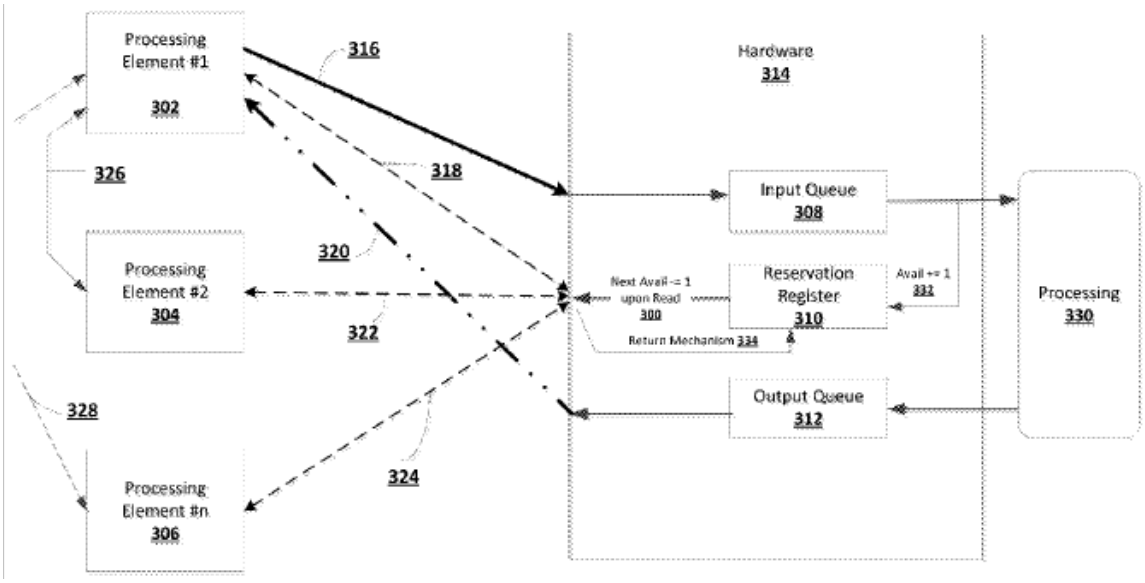
that Processor 1 may delegate a service application to Processor 2, which resolves the services application as an output. EX1008, 1:49–61. However, Processor 2 does not send this output ***back*** to the 1PE—when Processor 1 delegates a service application, it cedes all ownership and responsibility for it, contrary to the architecture of the '596 patent.

199. Khawand does not discuss output queues, nor do Petitioners contend that it does. Instead, they contend that it would nonetheless be obvious that Processor 1 receives a response corresponding to the service application it delegated. Pet., 65. Dr. Russ provides a conclusory opinion that "the purpose of delegation is for another processor to process a service application, [so] it would have been obvious to include an output queue to return processed data from that service application" and "output queues had been used for that purpose since at least the 1990s." EX1003, ¶293.

200. However, in my opinion, Petitioners misstate "the purpose of delegation." It would not be obvious for Processor 1 to receive a response corresponding to the delegated service application exactly for the reason that Dr. Russ opined: the purpose of delegation is for ***another processor*** to process a service application. Khawand teaches a system for load balancing across functional processors, so when Processor 1 delegates a service application, it cedes all responsibility for it and leaves any processing and output up to Processor 2.

201. The purpose of an output queue is to deliver results or acknowledgements back to the processing element where the command originated so that processing element may either confirm that the command was processed or conduct further processing with the results of the command. But in Khawand, Processor 1 is a functional processor tasked with executing the service application ***on behalf of another processor***. Service applications do not originate at Processor 1, thus it has no use for the results of the service application and would not require a response before processing other service applications.

202. This is a different relationship than between processors in the '596 patent. The '596 patent teaches a system for offloading tasks from the 1PE to a third processing element, but then ***receiving those results*** back via an output queue:



EX1001, FIG. 3. Processing 330 (*i.e.*, the third processing element) executes the command in the '596 patent workflow, but the result is sent back to Processing

Element #1 which initiated the command in the first place. *Id.* If Processor 330 were to delegate the processing to a different functional processor, as is the case in Khawand, then Processor 330 would not receive the results via an output queue—the 1PE would.

203. Khawand, on the other hand, only balances service applications among execution engines (*e.g.*, instances of Processing 330) to address a problem in the prior art where system have "poor processing efficiency" when there is more than one processor capable of executing a service application. EX1008, 1:25–28. Khawand teaches that Processor 1 (*i.e.*, the purported 1PE) and Processor 2 (*i.e.*, the purported third processing element) are both capable of executing service applications, but Processor 1 may delegate a service application if it is "running out of capacity for a given task." *Id.*, 4:2–5. But unlike in the '596 patent, Khawand's delegation among functional processors is effectively a "handoff" instead of a "feedback loop."

204. Processor 2 would not provide its results back to Processor 1 because it is not part of a closed-loop communication system. Sending a response via an output queue from Processor 2 to Processor 1 would not further any goal of Khawand's system. A POSITA would view this as an extra step and understand that Processor 1 would have no use for the response.

2. Claims 7 and 13

a. Limitations 7[a], 7[e]–7[i], and 7[k], and 13[a], 13[e]–13[i] Fail for the same reasons as equivalent in claim 1.

205. Khawand and Rosenthal do not anticipate or render obvious limitations 1[e]–[j]. For the equivalent limitations in claims 7 and 13, Petitioners simply cite back to the portions of the Petition concerning claim 1 without any additional evidence, analysis, or argument. Pet., 37–38. For this reason, in my opinion, Petitioners fail to show that elements 7[e]–7[i] and 7[k], and 13[e]–13[i] would be obvious for the same reasons discussed above.

b. Khawand and Rosenthal Did Not Disclose or Render Obvious a Second Processing Element Receiving a Command from the First Processing Element and Issuing a Command to the Input Queue on Behalf of the First Processing Element, as Required by Limitations 7[g]/[j] and 13[g]/[j].

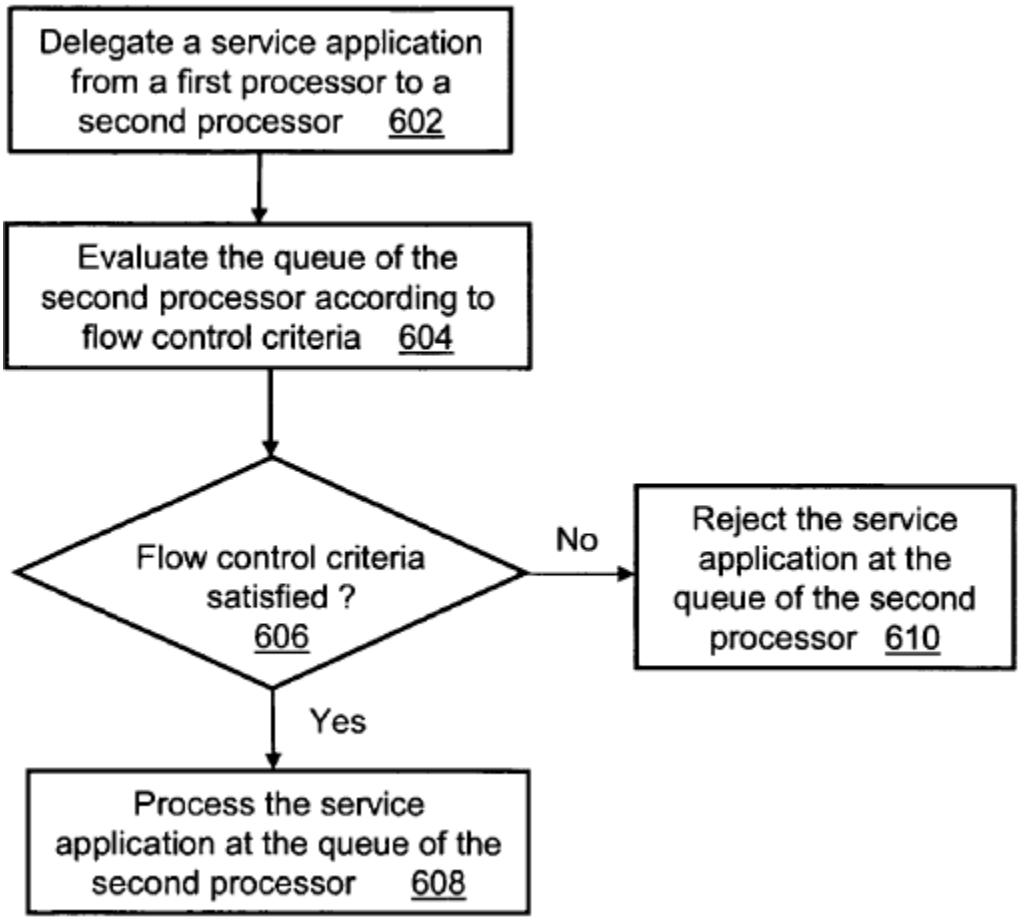
206. In my opinion, Petitioners do not demonstrate that Khawand and Rosenthal teach or suggest limitations 7[g]/[j] and 13[g]/[j] which require that the 2PE "receives a command from said first processing element" and "issues said command to said input queue ***on behalf of*** said 1PE." EX1001, cl. 7, 13 (emphasis added). There is no evidence that IPM 308 can accept or submit commands on behalf of the 1PE.

207. Petitioners admit that "[b]efore Khawand's Inter-Processor Manager submits the command from the first processor, it must first receive the command

from the first processor." Pet., 68. However, Khawand does not disclose IPM 308 "receiving the command" anywhere in the reference.

208. IPM 308 only acts as a flow control device that routes resume/suspend commands between execution engines. EX1008, 4:60–67. Khawand teaches that when a processor "is no longer available to process service applications," the processor submits a suspend command to IPM 308, "which in turn synchronizes the first processor 304 according to the suspend command." *Id.* When space becomes available, "the second processor 306 submits a resume command to the inter-processor manager 308 indicating the corresponding one of the group of processing elements is available to process service applications, which in turn synchronizes the first processor 304 according to the resume command." *Id.*, 5:21–26.

209. Processor 1 issues commands directly to Processor 2. EX1008, 3:66–4:2. Petitioners cite Figure 6 as evidence that IPM 308 purportedly issues the command from Processor 1 to the input queue of Processor 2, but Khawand explains that it is the complete opposite. Pet., 69.



Id., FIG. 6. At step 602, "the first and second processors 304–306 of the mobile device 106 are programmed to delegate a service application from the first processor 304 to a queue 404 of the second processor 306." *Id.* 3:66–4:2.

210. Khawand thus makes clear that it is the 1PE which delegates the service application to the 2PE, not an intermediary processor on its behalf. EX1008, 5:42–45 ("Referring back to step 602, ***the first processor*** 304 can be programmed to request an update regarding the capacity of the queue 404 of the second processor 306 ***before delegating*** the service application thereto.") (emphasis added).

211. Dr. Russ does not provide any unique opinion on the limitations, instead reciting the same unsupported argument from the Petition.

c. Khawand and Rosenthal Did Not Disclose or Render Obvious the Second Processing Element Receiving a Response Corresponding to a Command from the Output Queue, as Required by Limitations 13[k].

212. In my opinion, Petitioners do not demonstrate that it would be obvious for Khawand's IPM 308 to perform limitation 13[k], which requires that the 2PE "receives a response corresponding to said command from said output queue." EX1001, cl. 13. In other words, the 2PE that issued the command "on behalf of" the 1PE must receive a response back from the output queue corresponding to the executing of the issued command. *Id.*

213. The only purpose of IPM 308 is to signal processing engines whether to suspend or resume commands. As discussed above, IPM 308 does not offload commands on behalf of another processing element, thus there is ***no return data*** for IPM 308 to receive from the output queue.

214. Khawand is explicit that IPM 308 only plays a centralized role in load-balancing across service applications (*i.e.*, the third processing elements in the '596 patent system, not the 1PE before it issues a command):

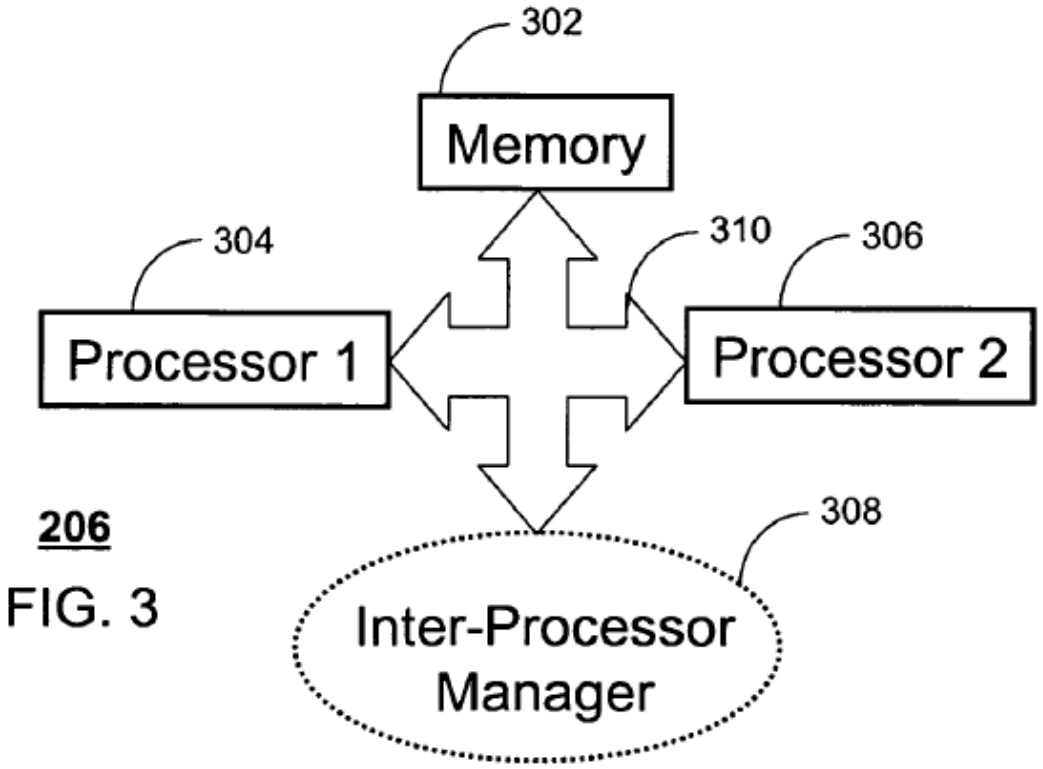


FIG. 3

EX1008, FIG. 3. Processor 1 and Processor 2 are functional processors resolving tasks that have already been added to an input queue before IPM 308 seeks to conduct load balancing. *Id.*, 4:2–5. Commands flow into the system from the 1PE to these execution engines and then a confirmation is sent back ***to the originating unit*** via an output queue once completed.

215. IPM 308 merely works downstream to balance which execution engine has the most capacity to resolve each task, so it is not designed to receive confirmation that the tasks have been executed. EX1008, 4:6–14. This would instead go back to the originating unit (*i.e.*, CPU); sending the confirmation to IPM 308 would add an unnecessary, burdensome step in the system because IPM 308 would have no use for the result. *Id.*

216. Petitioners contend that Khawand suggests IPM 308 "would also be able to access the corresponding output queues" because it "directly requests updates from each input queue." Pet., 72. But this ignores that IPM 308 only acts as a flow-control mediator between the processing elements and has no role in the system once the tasks have been executed. EX1008, 4:6–14, 5:57–63. Khawand describes that IPM 308 may act as a centralized hub for the processors within the system to track whether each other processor has capacity to load-balance a task. *Id.*

217. Petitioners effectively argue that it would be obvious for IPM 308 to take on the new, undisclosed role of receiving updates from output queues, which it would then pass via different output queues back to the host processor. Pet., 72. Even if it were technically possible to redesign Khawand to give IPM 308 access to any undisclosed output queues, there is no teaching or suggestion that a POSITA would do this because it would only harm the efficiency of the disclosed system by adding new, inconsistent roles to IPM 308.

218. In fact, I observe that Petitioners undercut this argument in their discussion of claims 6, 12, and 18. Each of these claims requires that "at least one third processing element executes said command and notifies said output queue to issue said response." EX1001, cl. 6, 12, 18. Petitioners contend that when the third processing element executes the commands, it provides its results ***back to the first processor*** (*i.e.*, "the point of delegation")—not to IPM 308. Pet., 66. Even more

damaging to their theory, Petitioners go on to allege that it would have been obvious to a POSITA that the output queue notifies the ***first processor*** of the result through a "ready" or "queue not empty" flag. Pet., 66 ("Accordingly, it was obvious for Processor 2 to provide the results from the delegated command to the output queue, and then notify the output queue ***to provide the response back to the first processor***") (emphasis added). It makes sense that the 1PE alone would be notified by the output queue because it is the processor which initially issued the service application. EX1008, 3:62–4:5.

219. Thus, Petitioners appear to agree that it would not be obvious for IPM 308 to receive the results back from the output queue. Pet., 66. Khawand and Rosenthal did not disclose limitation 13[k] because IPM 308 only mediates the load balancing between execution engines.

3. Dependant claims

a. Khawand and Rosenthal Did Not Disclose or Render Obvious Claims 6 and 12.

220. Petitioners contend that Khawand discloses that "at least one third processing element executes said command and notifies said output queue to issue said response" because Processor 2 purportedly "provides its results back" to Processor 1 after executing the service application. Pet., 66. Petitioners do not provide additional support for this argument beyond what they discussed for claim 1[j]. Pet., 66 (citing argument on limitation 1[j]). Thus, their argument fails for the

same reasons as above: Khawand does not disclose that Processor 2 sends results back to Processor 1 because Processor 1 ceded all responsibility when it delegated to Processor 2. Because Khawand does not teach disclose or suggest any output queues sending a response to Processor 1, it also does not teach that this response comes from the "third processing element."

b. Petitioners Do Not Even Allege Claim 18 Would Be Obvious Over Khawand and Rosenthal.

221. In my opinion, Petitioners fail to actually allege claim 18 is unpatentable. In particular, Petitioners argue that "[a]ccordingly, it was obvious for Processor 2 to provide the results from the delegated command to the output queue, and then notify the output queue to provide the response back to the first processor." Pet., 65–66. As explained above, that is incorrect. But even if it were correct, that does not address claim 18, which requires that the response be sent back to ***IPM 308*** (the alleged 2PE)—not the "IPM 308. Petitioners fail to provide any argument, analysis or evidence—or to even assert—that claim 18 would be obvious. *Id.* The Petition thus fails to prove claim 18 is unpatentable.

4. Ground 4 Does Not Address The Problems In Ground 3.

222. In my opinion, Ohkawa does not address the deficiencies in Ground 3. Petitioners present an alternative theory using the Ohkawa reference to demonstrate that hardware output queues were obvious by 2013. Pet., 72. Petitioners rely on

Ohkawa simply to disclose "output queues," and do not rely on Ohkawa to teach or suggest any of the limitations challenged above.

5. Ground 5 Does Not Address The Problems In Ground 3.

223. In my opinion, Matsunami thus does not address the deficiencies in Ground 3. In ground 5, Petitioners rely on Matsunami only for the limitations recited in claims 3–4, 9–10, 15–16. Petitioners do not rely on Matsunami to teach or suggest any of the limitations challenged above.

XII. SIGNATURE

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on this 5th day of January, 2026.

Dr. Paul S. Min