

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAP AMERICA, INC.,
Petitioner,

v.

VALTRUS INNOVATIONS LTD. and KEY PATENT INNOVATIONS LTD.
Patent Owners.

IPR2025-00417
U.S. Patent No. 6,823,409

**PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 6,823,409**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	11
II. COMPLIANCE WITH IPR REQUIREMENTS.....	11
III. CHALLENGED CLAIMS	14
IV. THE '409 PATENT.....	14
A. Effective Filing Date	14
A. Person of Ordinary Skill in the Art	15
B. Overview of the '409 Patent.....	15
V. CLAIM CONSTRUCTION	17
VI. PRINCIPAL PRIOR ART.....	18
A. Summary of Khare	18
B. Summary of Kirkman.....	20
VII. CLAIMS 1-17 ARE UNPATENTABLE	21
A. Ground 1: Claims 1-4, 7-12, 15-17 Are Obvious Based On Khare	21
1. Claim 1	21
2. Claim 2.....	41
3. Claim 3	44
4. Claim 4.....	45
5. Claim 7.....	48
6. Claim 8.....	52
7. Claim 9.....	55
8. Claim 10.....	58
9. Claim 11	59
10. Claim 12.....	60
11. Claim 15.....	60
12. Claim 16.....	60

13.	Claim 17	60
B.	Ground 2: Claims 5-6 and 13-14 Are Obvious Based On Khare and Kirkman	61
1.	Motivation to Combine	61
2.	Claim 5	67
2.	Claim 6	72
1.	Claim 13	73
2.	Claim 14	74
VIII.	DISCRETIONARY DENIAL IS NOT APPROPRIATE	74
A.	The Board Should Not Deny Institution Under 35 U.S.C. § 325(d)	74
B.	The Board Should Not Deny Institution Under 35 U.S.C. § 314(a)	74
IX.	CONCLUSION	78

TABLE OF ABBREVIATIONS AND CONVENTIONS

Petitioner	SAP America, Inc.
Patent Owners	Valtrus Innovations Ltd. (“Valtrus”) and Key Patent innovations Ltd. (“KPI”)
’409 Patent	U.S. Patent No. 6,823,409 (the patent being challenged herein)
Challenged Claims	Claims 1-17
POSITA	Person of Ordinary Skill in the Art

CLAIM LISTING

Claim 1	
1.0	A computer system comprising:
1.1	a host controller;
1.2	a first processor bus coupled between the host controller and a first processor;
1.3	a second processor bus coupled between the host controller and a second processor,
1.4	a random access memory (RAM) coupled to the host controller via a memory bus, the RAM comprising a portion of static memory and a portion of dynamic memory; and
1.5	a coherency control module operably coupled to the first processor bus and the second processor bus and comprising:
1.6	a request module configured to receive requests from the first processor bus and the second processor bus and to maintain proper ordering of the requests from each processor bus,
1.7	an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests currently being processed and to prevent simultaneous multiple accesses to a single address in the static portion of the RAM; and
1.8	a static RAM interface module configured to access an address look-up table corresponding to data stored in the static portion of the RAM.
Claim 2	
2.0	The computer system, as set forth in claim 1, comprising an input/output (I/O) bus coupled between the host controller and a plurality of I/O devices, wherein the I/O bus is operably coupled to the coherency control module and wherein the request module is configured to receive requests from the I/O bus.

Claim 3	
3.0	The computer system, as set forth in claim 1, wherein the coherency control module is located within the host controller.
Claim 4	
4.0	The computer system, as set forth in claim 1, wherein the coherency control module comprises a plurality of list structures corresponding to the first processor bus and the second processor bus.
Claim 5	
5.0	The computer system, as set forth in claim 4, wherein the list structures comprise two-dimensional doubly-linked lists with head and tail pointers.
Claim 6	
6.0	The computer system, as set forth in claim 5, wherein the list structures comprise dependency links.
Claim 7	
7.0	The computer system, as set forth in claim 4, wherein the request module comprises a plurality of bypass paths configured to provide a direct request path from the first and second processor buses to the static RAM interface module, thereby bypassing the list structures.
Claim 8	
8.0	The computer system, as set forth in claim 1, wherein the active snoop queue module comprises a plurality of active snoop queues, each active snoop queue configured to maintain a list of requests currently being processed and to function independently with respect to each other.

Claim 9	
9.0	The computer system, as set for the in claim 1, wherein the static RAM interface module comprises a plurality of static RAM interfaces, each static RAM interface corresponding to a segment of the static portion of the RAM.
Claim 10	
10.0	A coherency control module configured to control access to cache memory in a computer system, the coherency control module comprising:
10.1	a request module configured to receive requests from a plurality of buses in a computer system and to maintain proper ordering of the requests from each bus,
10.2	an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests from all of the buses currently being processed and to prevent multiple accesses to a single address in the cache memory simultaneously; and
10.3	a static RAM interface module configured to access an address look-up table corresponding to data stored in the cache memory.
Claim 11	
11.0	The coherency control module, as set forth in claim 10, wherein the plurality of buses comprise processor buses.
Claim 12	
12.0	The coherency control module, as set forth in claim 10, wherein the coherency control module comprises a plurality of list structures corresponding to the first process or bus and the second processor bus.

Claim 13	
13.0	The coherency control module, as set forth in claim 12, wherein the list structures comprise two-dimensional doubly-linked lists with head and tail pointers.
Claim 14	
14.0	The coherency control module, as set forth in claim 13, wherein the list structures further comprise dependency links.
Claim 15	
15.0	The coherency control module, as set forth in claim 12, wherein the request module comprises a plurality of bypass paths configured to provide a direct request path from the first and second processor buses to the static RAM interface module, thereby bypassing the list structures.
Claim 16	
16.0	The coherency control module, as set forth in claim 10, wherein the active snoop queue module comprises a plurality of active snoop queues, each active snoop queue configured to maintain a list of requests currently being processed and to function independently with respect to each other.
Claim 17	
17.0	The coherency control module, as set forth in claim 10, wherein the static RAM interface module comprises a plurality of static RAM interfaces, each static RAM interface corresponding to a segment of the static portion of the RAM.

PETITIONER’S EXHIBIT LIST

No.	Description
1001	U.S. Patent No. 6,823,409
1002	U.S. Patent No. 6,487,643 (“Khare”)
1003	SAP’s Invalidity Contentions (Cover Pleading) in <i>Valtrus Innovations Ltd. v. SAP America, Inc. et al</i> , 2:24-cv-00021-JRG (E.D. Tex.)
1004	U.S. Patent No. 6,581,063 (“Kirkman”)
1005	Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation by Director Katherine K. Vidal
1006	Declaration of Dr. Bruce Jacob
1007	<i>Curriculum Vitae</i> of Dr. Bruce Jacob
1008	File History of U.S. Patent No. 6,823,409
1009	First Amended Docket Control Order entered in <i>Valtrus Innovations Ltd. v. SAP Am. Inc.</i> , No. 2:24-cv-21, ECF 65 (E.D. Tex. Jan. 15, 2024)
1010	SAP America, Inc. and SAP SE’s Motion to Dismiss for Lack of Standing Pursuant to Rule 12(b)(1) and Rule 12(b)(6)
1011	SAP America, Inc. and SAP SE’s Motion to Reconsider Partial Denial of SAP’s Motion to Dismiss and Grant of Valtrus’s Motion for Leave to Amend Complaint (ECF 83)
1012	Valtrus and KPI’s Complaint for Patent Infringement and Jury Demand, filed in <i>Valtrus Innovations Ltd. v. SAP Am, Inc.</i> , No. 2:24-cv-533, ECF 1 (E.D. Tex. July 12, 2024)
1013	Valtrus and KPI’s Statement Regarding <i>Valtrus II</i> (Case No. 2:24-CV-533-JRG), filed in <i>Valtrus Innovations Ltd. v. SAP Am, Inc.</i> , No. 2:24-cv-21, ECF 85 (E.D. Tex. Nov. 15, 2024)
1014	John L. Hennessy and David A. Patterson, COMPUTER ARCHITECTURE: A QUANTITATIVE APPROACH (5th ed. 2012)
1015	Bruce Jacob, Spencer W. Ng, and David T. Wang, MEMORY SYSTEMS: CACHE, DRAM, DISK (2008)

1016	Daniel J. Sorin, Mark D. Hill, David A. Wood, A PRIMER ON MEMORY CONSISTENCY AND CACHE COHERENCY (2011)
1017	Kunle Olukotun, et al., <u>The Case for a Single-Chip Multiprocessor</u> , STANFORD UNIVERSITY: COMPUTER SYSTEMS LABORATORY (1996)
1018	Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun, <u>A Single-Chip Multiprocessor</u> , STANFORD UNIVERSITY (Sept. 1997)
1019	Alan Jay Smith, <u>Cache Memories</u> , 14 COMPUTING SURVEYS 3 (Sept. 1982)
1020	Thomas Plum, RELIABLE DATA STRUCTURES IN C (1985)

I. INTRODUCTION

Petitioner submits this Petition for *Inter Partes* Review of claims 1-17 of U.S. Patent No. 6,823,409 (Ex-1001), assigned to Patent Owners. Petitioner respectfully submits that the Challenged Claims of the '409 Patent are unpatentable under 35 U.S.C. § 103 in view of the prior art references discussed herein. Accordingly, it is respectfully requested that the Board institute IPR of the '409 Patent.

II. COMPLIANCE WITH IPR REQUIREMENTS

A. Grounds for Standing

As required by 37 C.F.R. § 42.104(a), Petitioner certifies that the '409 Patent “is available for inter partes review and that the Petitioner is not barred or estopped from requesting an inter partes review challenging the patent claims on the grounds identified in the petition.” Neither Petitioner nor any real party in interest has filed a civil action challenging the validity of a claim of the '409 Patent. This petition is being filed no more than one year after the date on which Petitioner or real parties in interest or privies of Petitioner were served with a complaint alleging infringement of the '409 Patent.

B. Mandatory Notices

Real Party-in-Interest: Petitioner identifies SAP America, Inc. and SAP SE (collectively, “SAP”) as real parties-in-interest.

Related Matters: No previous IPR or post-grant proceeding has been filed challenging the '409 Patent. Valtrus has asserted the '409 Patent in litigation against SAP in *Valtrus Innovations Ltd. v. SAP America, Inc. et al*, 2:24-cv-00021-JRG (E.D. Tex.), Patent Owners have asserted the '409 Patent in litigation against SAP in *Valtrus Innovations Ltd. v. SAP America, Inc. et al*, 2:24-cv-00533 (E.D. Tex.), and SAP filed a declaratory judgment action of non-infringement of the '409 Patent against Patent Owners and Patent Platform Services, LLC in *SAP America, Inc. v. Valtrus Innovations Limited et al*, 1:24-cv-00756-GBW (D. Del.).

Lead and Back-up Counsel:

Lead Counsel	Back-up Counsel
Louis L. Campbell (Reg. # 59,963) WINSTON & STRAWN LLP 255 Shoreline Drive, Suite 520 Redwood City, CA 94065 Tel: (650) 858-6500 LLcampbell@winston.com	Scott M. Border (Reg. # 77,744) WINSTON & STRAWN LLP 1901 L Street NW Washington, DC 20036 Tel: (202) 282-5000 SBorder@winston.com
	Samuel W. Riebe (Reg. # 78,044) WINSTON & STRAWN LLP 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500 SRiebe@winston.com
	Michael Bittner (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500

	<u>MBittner@winston.com</u>
	Matt R. McCullough (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP <i>Pro hac vice</i> to be submitted Winston & Strawn LLP 255 Shoreline Drive, Suite 520 Redwood City, CA 94065 Tel: (650) 858-6500 <u>MRMcCullough@winston.com</u>
	William Logan (<i>Pro hac vice</i> to be submitted) <i>Pro hac vice</i> to be submitted WINSTON & STRAWN LLP 800 Capitol Street, Suite 2400 Houston, TX 77002 Tel: (713) 651-2600 <u>WLogan@winston.com</u>
	Patrick Clark (<i>Pro hac vice</i> to be submitted) WINSTON & STRAWN LLP <i>Pro hac vice</i> to be submitted 2121 N. Pearl Street, Suite 900 Dallas, TX 75201 Tel: (214) 453-6500 <u>PClark@winston.com</u>

Service Information: Service information is provided in the designation of counsel above. Petitioner consents to service of all documents via electronic mail at IPR-SAP-Val@winston.com.

C. Fee Authorization

The Office is authorized to charge all fees due in connection with this proceeding to Deposit Account No. 50-1814.

III. CHALLENGED CLAIMS

Under 37 C.F.R. §§ 42.104(b) and 42.22, Petitioner requests that the Board institute this IPR on the Challenged Claims of the '409 Patent and cancel those claims as unpatentable on the following grounds:

Ground	Claims	Basis for Unpatentability
1	1-4, 7-12, 15-17	Obviousness under 35 U.S.C. § 103 by <u>Khare</u>
2	5-6, 13-14	Obviousness under 35 U.S.C. § 103 over <u>Khare</u> in view of <u>Kirkman</u>

As discussed below, Khare and Kirkman qualify as prior art and are properly relied upon for showing unpatentability of the '409 Patent.

IV. THE '409 PATENT

A. Effective Filing Date

September 28, 2001, is the effective filing date because this is the filing date of the application that issues as the '409 Patent, and it does not claim priority to any other application. In parallel litigation, Patent Owners have asserted a conception date of May 25, 2001. While Petitioner does not waive its right to dispute any assertion that the claims are entitled to an earlier invention date in the parallel

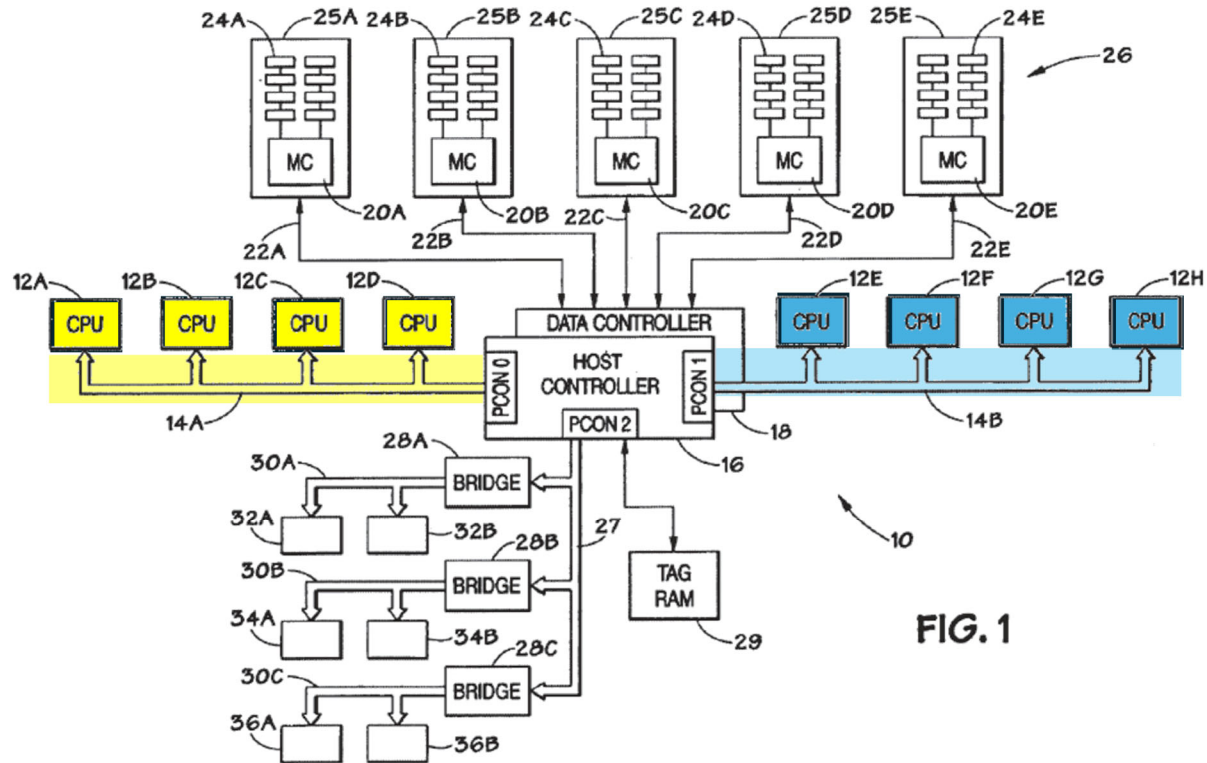
litigation, the dispute is immaterial to this petition because all prior art references asserted here qualify as prior art to either date.

B. Person of Ordinary Skill in the Art

A POSITA in September 2001 or May 2001 would have had an understanding of multiprocessor design and/or multicore microprocessor design, as well as multiprocessor cache design. Ex-1006, ¶ 12. A POSITA would have had a bachelor's degree in computer science, electrical engineering, or computer engineering, and two years of relevant experience in memory-systems design, and related fields. Ex-1006, ¶ 12. Lack of professional experience can be remedied by additional education, and vice versa. Ex-1006, ¶ 12.

C. Overview of the '409 Patent

The '409 Patent describes a mechanism for “filtering processor cache snoops in a multi-processor-bus system.” Ex-1001, 1:9-12. As shown in Figure 1, the '409 Patent provides an exemplary multi-processor-bus computer system, which utilizes a split bus configuration comprised of processors 12A-12D coupled to a first bus 14A and processors 12E-12H coupled to a second bus 14B. Ex-1001, 4:6-12.



Ex-1001, Fig. 1 (annotated). Buses 14A and 14B are coupled to a chip set containing host controller 16 and data controller 18 (together, “host/data controller”). Ex-1001, 4:20-21. The host/data controller is coupled to five memory controllers 20A-20E via individual bus segments, and the memory controllers are further coupled to main memory 24A-24E. Ex-1001, 4:25-29. The host/data controller is coupled to one or more bridges 28A-28C via an input/output (I/O) bus 27. Ex-1001, 4:45-46. “The opposite side of each bridge 28A-28C is coupled to a respective bus 30A-30C, and a plurality of peripheral devices 32A and 32B, 34A and 34B, and 36A and 36B may be coupled to the respective buses 30A, 30B, and 30C.” Ex-1001, 4:47-50. Host/data controller contains processor controller PCON0–PCON2, which connects

the buses external to the host controller 16 (i.e., processor bus 14A and 14B and I/O bus 27). Ex-1001, 5:5-11. The host controller also contains a TCON module 40, which acts “as a snoop filter to minimize the number of clock cycles required to obtain cache state information and to determine the proper snoop response on the processor or I/O buses 14A, 14B, and 27.” Ex-1001, 5:16-20. The TCON module contains several components responsible for maintaining cache coherence—(1) the request module 42 “insures memory coherency through proper cycle order, arbitrates access to the memory among the processor and I/O buses 14a, 14B, and 27, and optimally utilizes the tag look up bandwidth” (Ex-1001, 5:32-36); (2) an active snoop queue 44 “which may include one or more buffers which contain the indices of all requests which are currently active in the TCON module 40” (Ex-1001, 6:3-6); (3) “[t]he tag RAM 29 is a separate memory storage device from main memory 26 that keeps track of all cachelines in use by the CPUs 12A-12H” (Ex-1001, 6:26-28); and (4) the “response module 50 receives the current tag state information for a cycle being processed by the ASQ 44” (Ex-1001, 6:66-67).

V. CLAIM CONSTRUCTION

Claims are given their “ordinary and customary meaning” as understood by a POSITA and the prosecution history pertaining to the patent. 37 C.F.R. § 42.100(b). Because a POSITA would find the challenged claims unpatentable under any interpretation consistent with their plain and ordinary meaning in the context of the

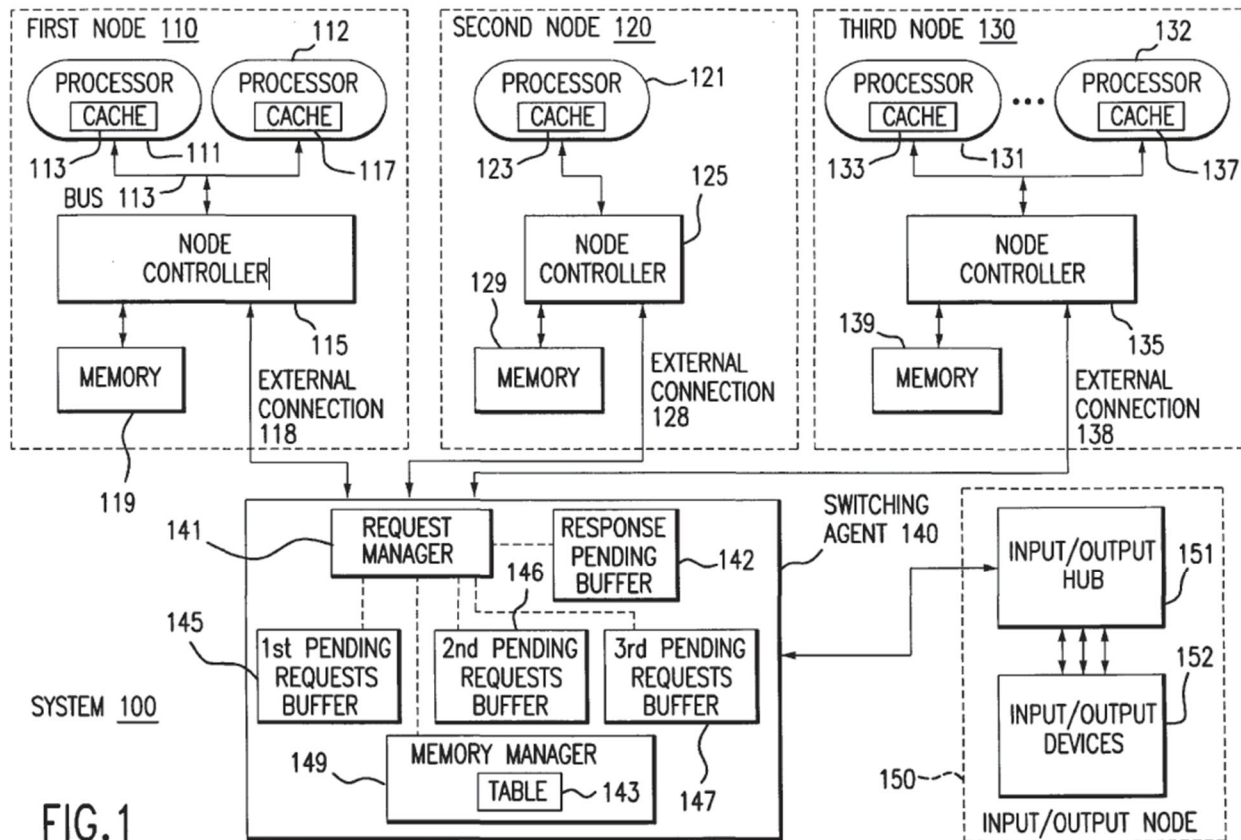
'409 Patent, the Board need not expressly construe the claim terms. *See Vivid Techs., Inc. v. Am. Sci. & Eng'g. Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999).

VI. PRINCIPAL PRIOR ART

A. Summary of Khare

U.S. Patent No. 6,487,643 to Khare (Ex-1002) was filed on September 29, 2000, and issued on November 26, 2002. Khare qualifies as prior art to the '409 Patent at least under pre-AIA 35 U.S.C. § 102(e).

Khare relates to “a method and apparatus for managing memory related requests in a multi-node architecture such that there is no starvation of requests from a subset of nodes and every request from a node gets a chance to eventually complete.” Ex-1002, 1:8-12. Khare teaches a multi-node system, and each node contains several components, connected via a plurality of buses or signals, including one or more processors, memories, or input/output devices. Ex-1002, 2:29-32. As shown in Figure 1, the system disclosed by Khare contains nodes 110, 120, and 130 and an input/output node 150. Ex-1002, 3:25-27.



Ex-1002, Fig. 1. The nodes are coupled to a switching agent via external connections 118, 128, and 138 and such coupling encompasses “a direct connection, an indirect connection, an indirect communication, etc.” Ex-1002, 3:27-36. Node 110, 120, and 130 each contain processors, a node controller, and external memory such that the processors and external memory are connected to a node controller. Ex-1002, 3:37-39. The external memory may be random access memory (RAM), and the caches within each processor may be Level 2 (L2) cache comprised of static random access memory. Ex-1002, 3:42-48. “[S]witching agent 140 may be a routing switch for routing messages within system 100.” Ex-1002, 3:66-7. Switching agent 140 contains request manager 141, response pending buffer 142, first pending requests

buffer 145, second pending requests buffer 146, third pending requests buffer 147, and memory manager 149. Ex-1002, 4:1-4. Request manager 141 can manage requests that are received by switching agent 140 from nodes in system 100. Ex-1002, 4:7-9. Also, the response pending buffer 142 may store information about requests that have not been sent out and have not yet been received, and pending requests buffers may store requests that have been received by switching agent 140 but have not yet been processed by switching agent 140. Ex-1002, 4:13-23.

B. Summary of Kirkman

U.S. Patent No. 6,581,063 to Kirkman (Ex-1004) was filed on June 15, 2000, and issued on June 17, 2003. Kirkman qualifies as prior art to the '409 Patent at least under pre-AIA 35 U.S.C. § 102(e).

Kirkman relates to a method and apparatus for maintaining a linked list where “[p]referably, the linked list is doubly-linked, and list maintenance is accomplished using object-oriented programming techniques.” Ex-1004, Abstract. Kirkman describes methods for implementing “inspectors” and “mutators” where “[a]n inspector object traverses the list without changing it; while a mutator alters the list, and may traverse it as well” and “[b]oth inspectors and mutators have associated view objects, which contain the pointers to the actual list elements.” Ex-1004, Abstract.

VII. CLAIMS 1-17 ARE UNPATENTABLE**A. Ground 1: Claims 1-4, 7-12, 15-17 Are Obvious Based On Khare****1. Claim 1****i. [1.0] *A computer system comprising:***

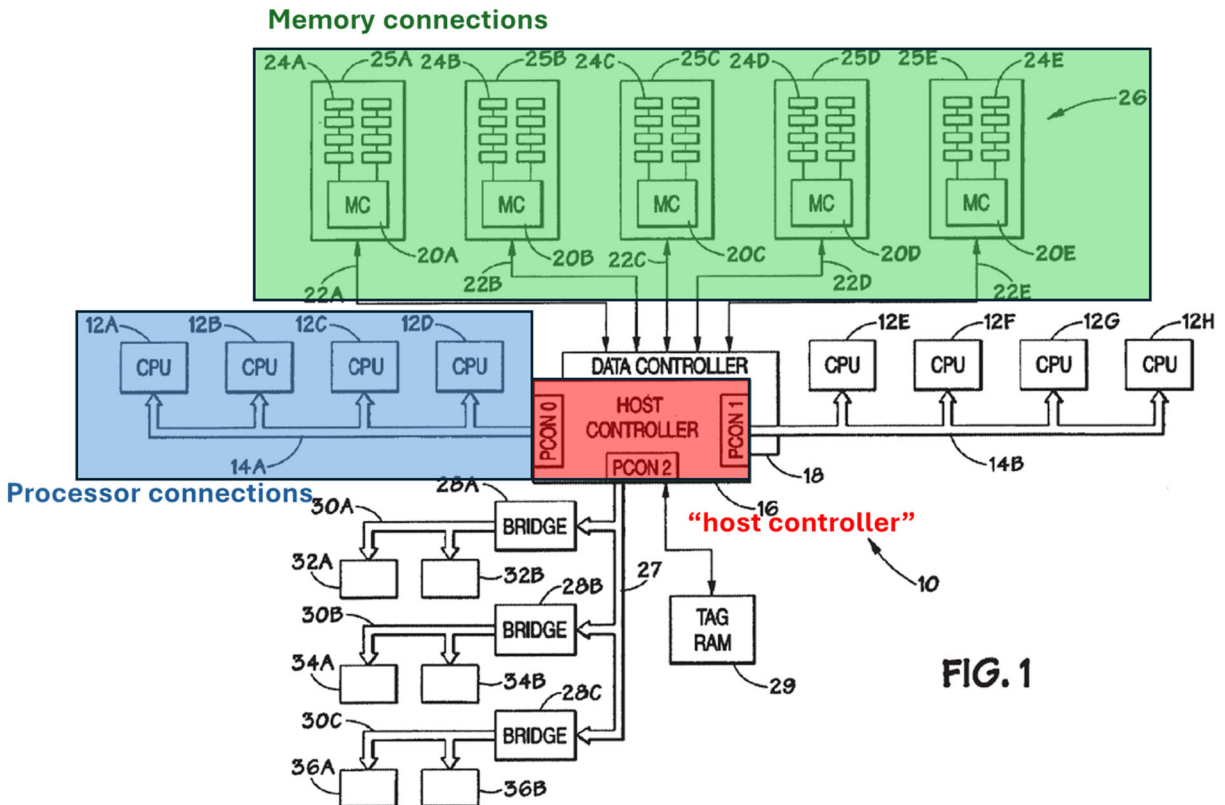
As depicted in Figure 1, Khare teaches “**a computer system** that includes processors, memory devices, and input/output devices.” Ex-1002, 2:23-25¹; Ex-1006, ¶¶ 71-72.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 70.

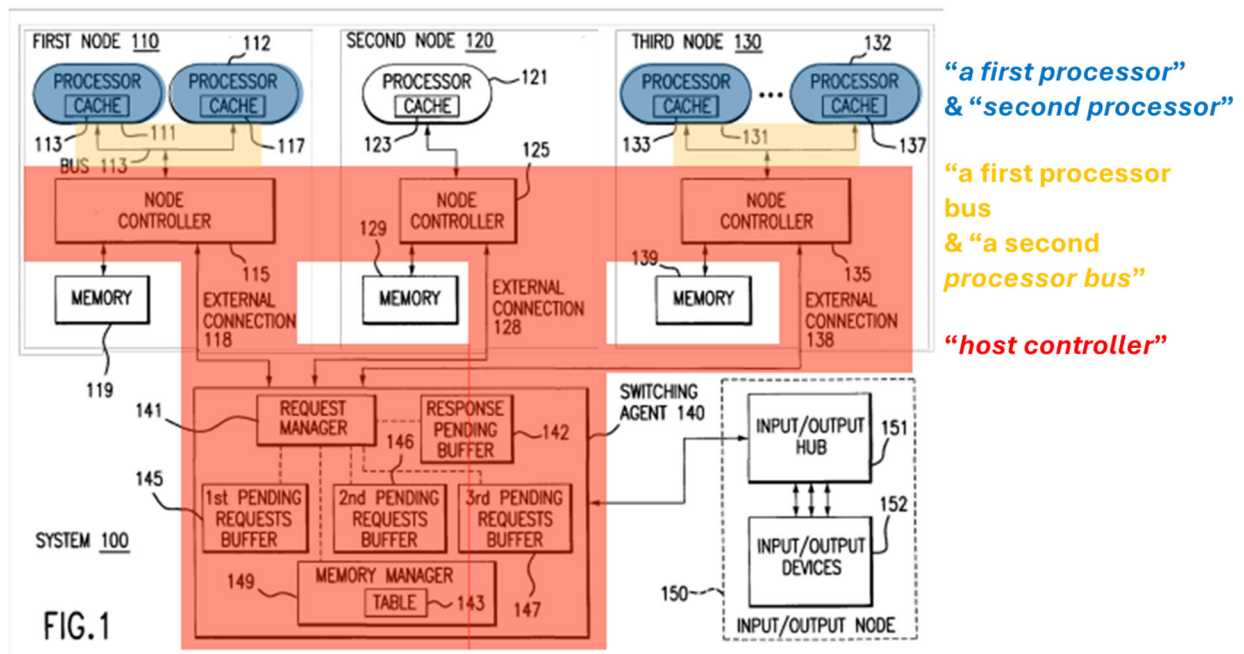
ii. [1.1] *a host controller;*

The '409 Patent describes the “*host controller*” as being a hardware component that is connected to memory through memory controllers and processors through buses. Ex-1001, 4:20-52; Ex-1001, Fig. 1.

¹ Emphasis added in bold throughout unless otherwise stated.



Ex-1001, Fig. 1 (annotated).



Ex-1002, Fig. 1 (annotated).

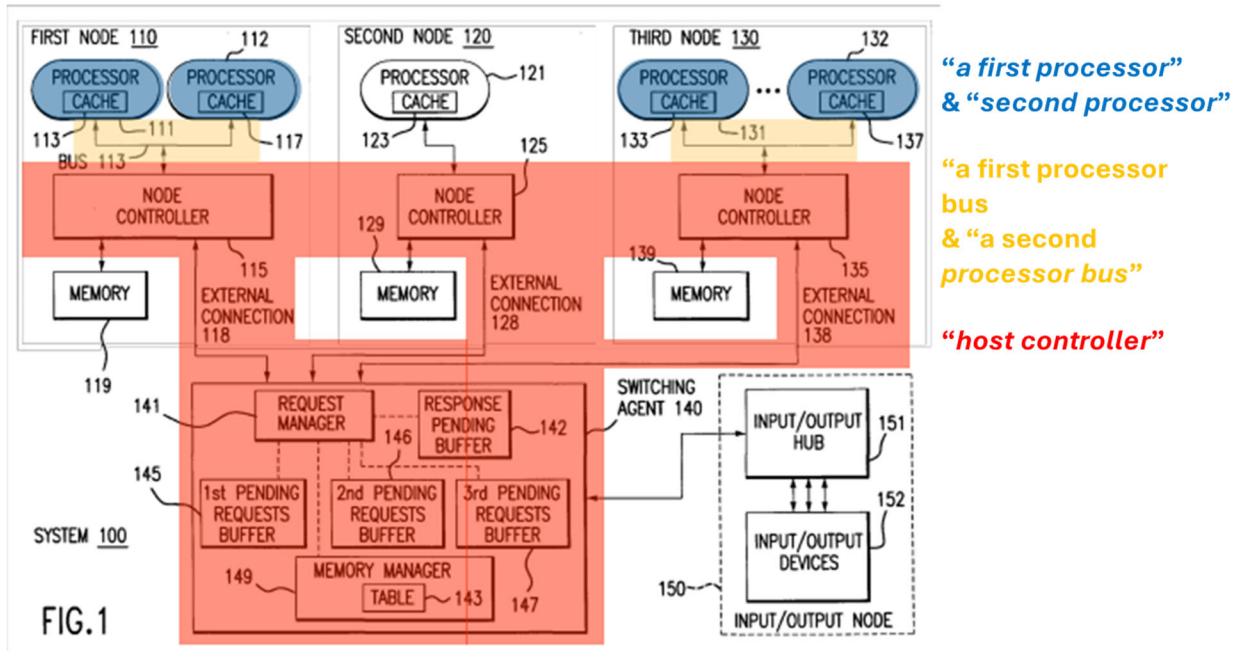
Figure 1 shows that Khare teaches a system arranged as nodes. Ex-1002, 2:29-32. Each node contains multiple processors, a node controller, and memory. Ex-1002, 3:37-49. In particular, “first node 110, includes processor 111, processors 112, and node controller 115, which are coupled to each other by bus 113” (Ex-1002, 3:38-40), and a “memory 119 which is coupled to node controller” (Ex-1002, 3:44-45). Ex-1002, 3:29-30 (“The term ‘coupled’ encompasses a direct connection, an indirect connection, an indirect communication, etc.”). Second and third nodes also contain processors and memory connected to the node controller. Ex-1002, 3:50-56. In Khare, the processors and memory are each connected through the node controller which is “a *host controller*.” Ex-1002, 3:37-49; Ex-1006, ¶¶ 76-77.

Accordingly, Khare teaches this claim limitation. Ex-1006, ¶ 74.

- iii. ***[1.2] a first processor bus coupled between the host controller and a first processor; [1.3] a second processor bus coupled between the host controller and second processor***

Khare teaches a system with several nodes, each containing processors, a node controller, and memory. Ex-1002, 3:37-49. The processors may also be connected to the node controller through one or more buses. Ex-1002, 2:29-32 (“The components within a node may be connected to other components in that node through **one or more busses** or signals.”). Khare teaches this architecture for the first node as well as a similar architecture for the second and third nodes. Ex-1002, (“First node 110 includes processor 111, processor 112, and node controller 115,

which are coupled to each other by bus 113.”), 3:50-51 (“Similarly, second node 120 contains a processor 121 and node controller 125 which are coupled to each other.”); 3:53-55 (“Third node 130 contains a processor 131, processor 132, and node controller 135 that are coupled to each other.”); Ex-1006, ¶ 75.

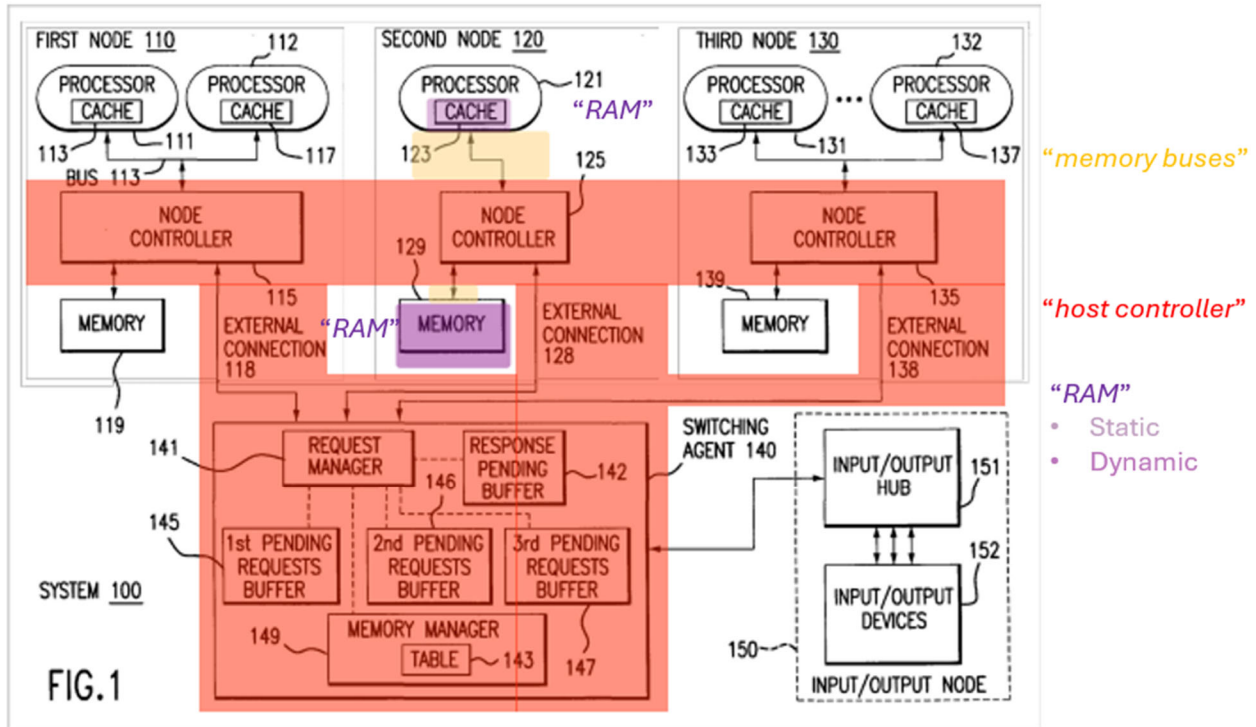


Ex-1002, Fig. 1 (annotated). Figure 1 shows that in first node 110, processors 111 and 112 (“a first processor”) are coupled to the “host controller” via bus 113 (“a first processor bus”) (“a first processor bus coupled between the host controller and a first processor”). Ex-1002, Fig. 1; Ex-1006, ¶¶ 74-77. Figure 1 also shows that in third node 130, processors 131 and 132 (“a second processor”) are coupled to the “host controller” via a bus (“a second processor bus”) (“a second processor bus coupled between the host controller and a second processor”). Ex-1002, Fig. 1; Ex-1006, ¶ 77.

Accordingly, Khare teaches these limitations. Ex-1006, ¶ 77.

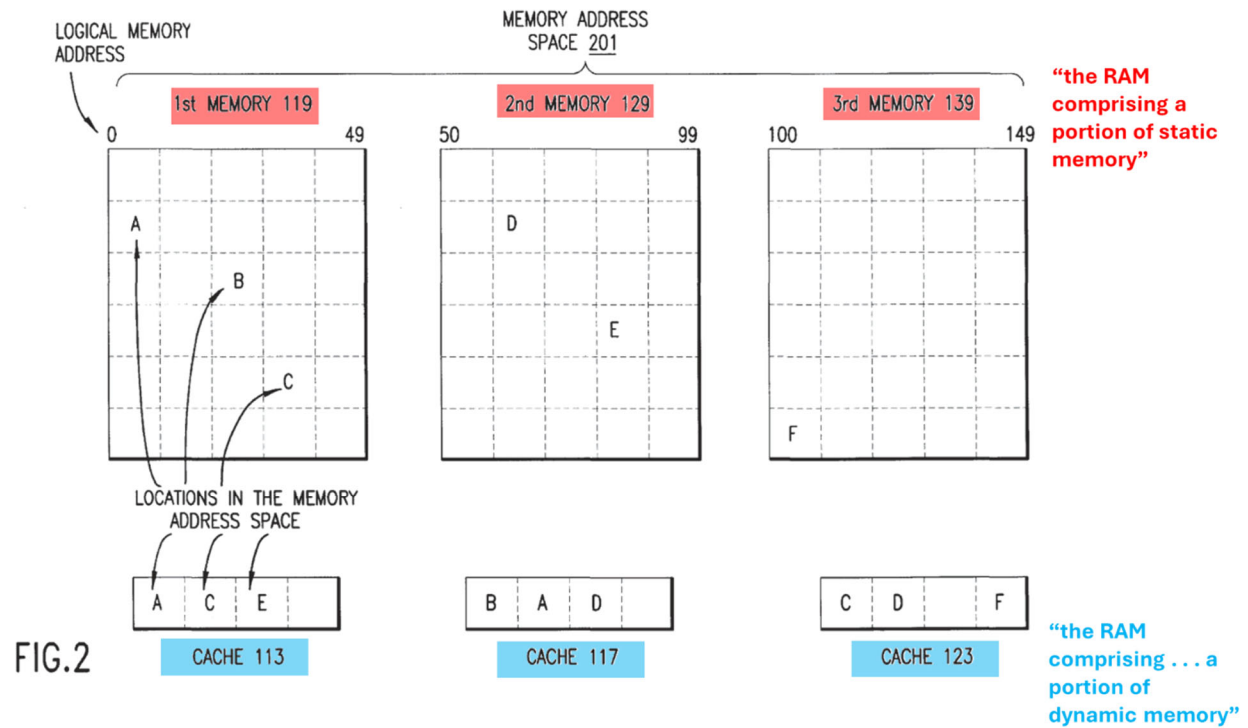
- iv. ***[1.4] a random access memory (RAM) coupled to the host controller via a memory bus, the RAM comprising a portion of static memory and a portion of dynamic memory; and***

Khare explains that “the set of data currently being used by a microprocessor may be copied from a system memory device such as a dynamic random access memory (DRAM) into a relatively smaller but faster cache memory device such as a static random access memory (SRAM).” Ex-1002, 1:26-30. Indeed, Khare explains that the cache memory within each processor “may be Level 2 (L2) cache memories that are comprised of static random access memory” (“*the RAM comprising a portion of static memory*”). Ex-1002, 3:46-48. Khare also explains that memory 119, 129, and 139 depicted in each node may be composed of dynamic random access memory (DRAM) (“*the RAM comprising . . . a portion of dynamic memory*”). Ex-1002, Fig. 1; Ex-1002, 3:44-46 (“Memory 119 may be a Random Access Memory (RAM).”); Ex-1002, 1:26-30 (“In many computer systems, the set of data currently being used by a microprocessor may be copied from a system memory device such as a dynamic random access memory (DRAM).”); Ex-1006, ¶ 78.



Ex-1002, Fig. 1 (annotated).

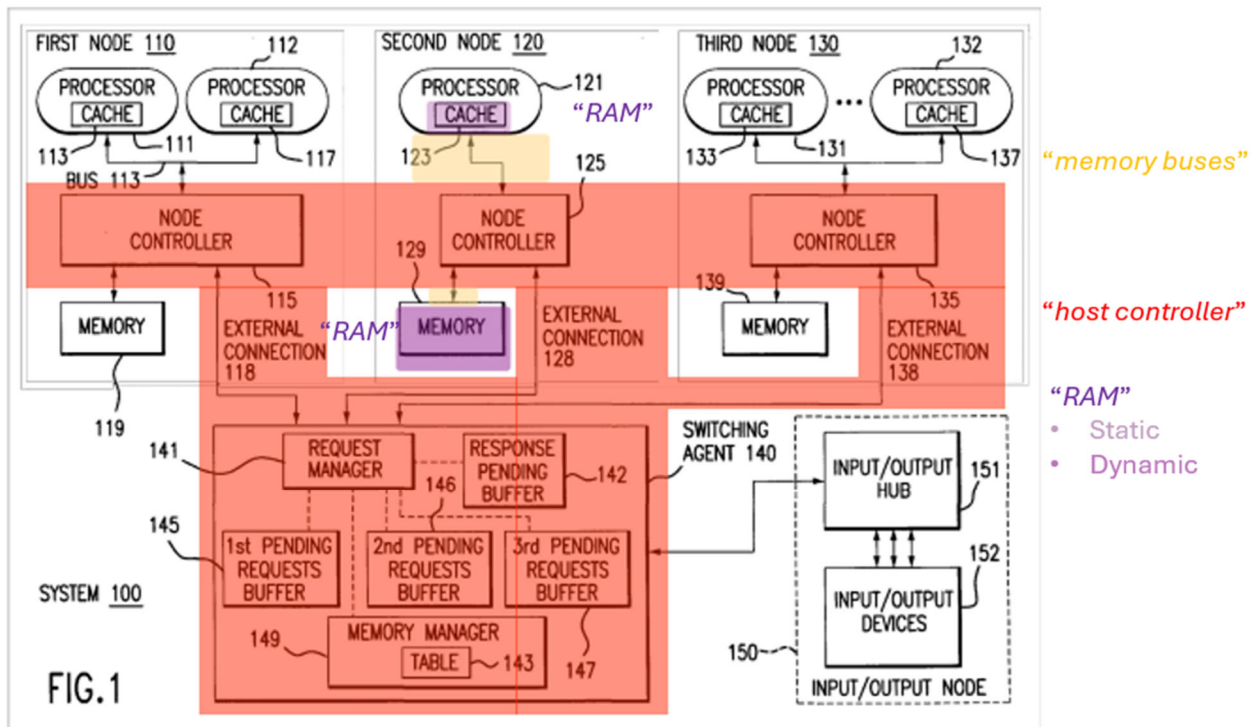
Figure 2 depicts a memory address space, which contains, for example, the memories 119, 129, and 139 of each node, which a POSITA would have understood to be implemented in DRAM, and the internal caches 113, 117, and 123 of each processor, which a POSITA would have understood to be implemented in SRAM. Ex-1002, 5:36-39, 5:54-55; Ex-1006, ¶ 80. Indeed, Khare states, “The information in any particular location in the memory address space 201 may be cached in one or more of the caches in the system.” Ex-1002, 5:63-65; Ex-1006, ¶ 81.



Ex-1002, Fig. 2 (annotated). Khare explains that the memory address space can be a “shared” address space “because any node in the multi-node system may address any location in address space 201.” Ex-1002, 5:48-50. Figure 2 also shows certain locations designated as A, B, C, D, E, and F storing data and instructions, which may be cached in one or more of the caches in the system. Ex-1002, 5:60-65. For example, “a copy of the information in the location designated A in first memory 119 is cached in caches 113, 117, and 123,” and “a copy of the information in the location designated B in first memory 119 is cached in cache 117.” Ex-1002, 5:66-6:2; Ex-1006, ¶ 80.

Memory 119, 129, and 139 and the cache memory internal to each processor are coupled to the node controller via a bus. Ex-1002, 2:29-32 (“Each node may

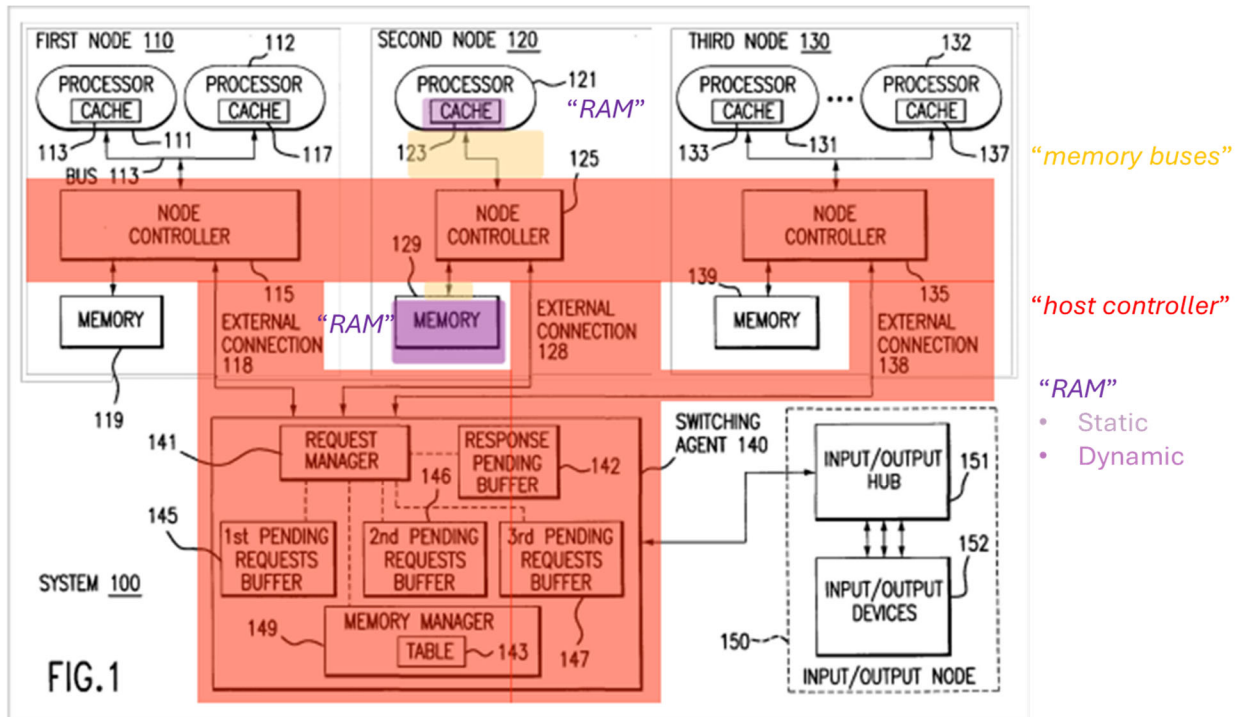
contain one or more processors, memories, or input/output devices. The components within a node may be connected to other components in that node through one or more busses or signals.”). Khare also explains that each node contains memory coupled to the node controller, and each processor may contain cache memory. Ex-1002, 3:42-46; Ex-1006, ¶ 78.



Ex-1002, Fig. 1 (annotated).

The '409 patent uses only “*memory bus*” in the claim, and not the specification. Ex-1001, cl.1. Based on the embodiments described in the specification, the functionality of a “*memory bus*” would entail coupling the “*host controller*” to the “*RAM*,” which is described in the embodiment as being the memory cartridges connected to the “*host controller*” through “individual bus

annotated figure below. Ex-1006, ¶¶ 78, 82.



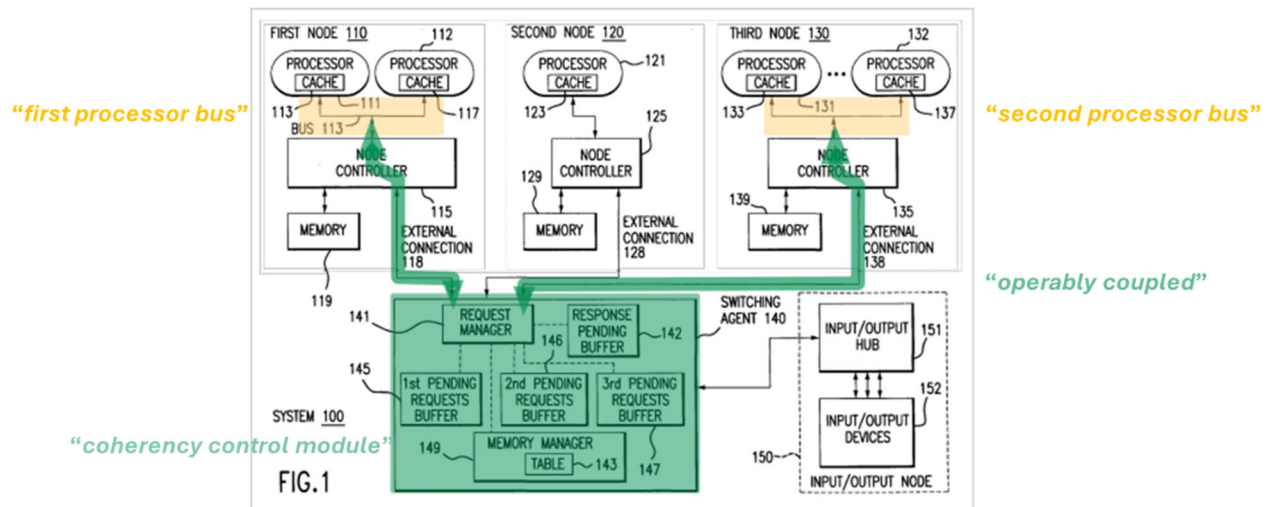
Ex-1001, Fig. 1 (annotated).

Accordingly, Khare's bus connections between the memory (119, 129, 139) (“*RAM*”) and the node controller (“*the host controller*”) would be a “*memory bus*” as shown in the annotated figure above. Ex-1006, ¶ 82. Further, the processor bus that connects the cache memory on each node to the node controller would also be “*memory buses*” based on the embodiment’s descriptions. Ex-1006, ¶ 82.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 78.

v. ***[1.5] a coherency control module operably coupled to the first processor bus and the second processor bus and compromising:***

As discussed in limitations [1.2] and [1.3], Khare teaches that the bus 113 of the first node coupling processors 111 and 112 is “*the first processor bus*” and the bus coupling processors 131 and 132 of the second node is “*the second processor bus*.” Ex-1002, Fig. 2; Ex-1006, ¶ 77. Khare explains that “the processors in nodes 110, 120 and 130 each have one or more caches (e.g., Level 1 and Level 2 caches), and these caches are kept coherent using the switching agent 140,” which indicates that the processors are operably coupled to switching agent 140 (“*a coherency control module*”) over the buses (“*the first processor bus and the second processor bus*”) connecting the processors of the first node and third node (“*a coherency control module operably coupled to the first processor bus and the second processor bus*”). Ex-1006, ¶ 85 (quoting Ex-1002, 5:21-24); Ex-1002, 3:28-29 (“The term ‘coupled’ encompasses a direct connection, an indirect connection, an indirect communication, etc.”).



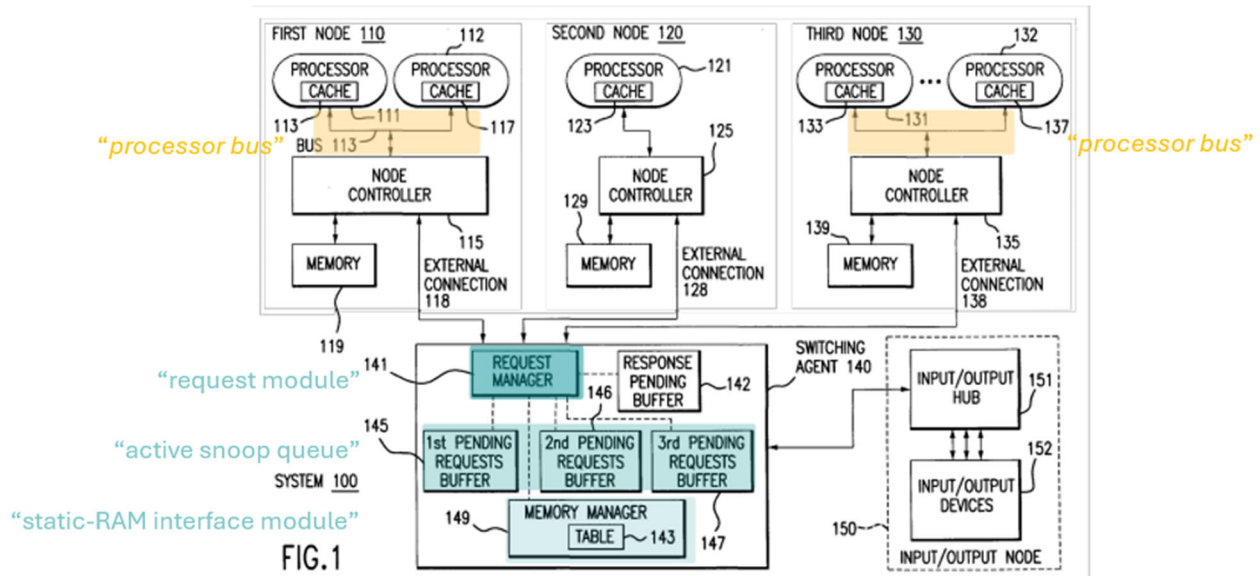
Ex-1002, Fig. 1 (annotated); Ex-1006, ¶ 83.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 83.

- vi. ***[1.6] a request module configured to receive requests from the first processor bus and the second processor bus and to maintain proper ordering of the requests from each processor bus;***

As discussed in limitations [1.2] and [1.3], Khare teaches that bus 113 of the first node coupling processors 111 and 112 is “the first processor bus” and the bus coupling processors 131 and 132 of the third node is “the second processor bus.” Ex-1002, Fig. 2; Ex-1006, ¶ 77.

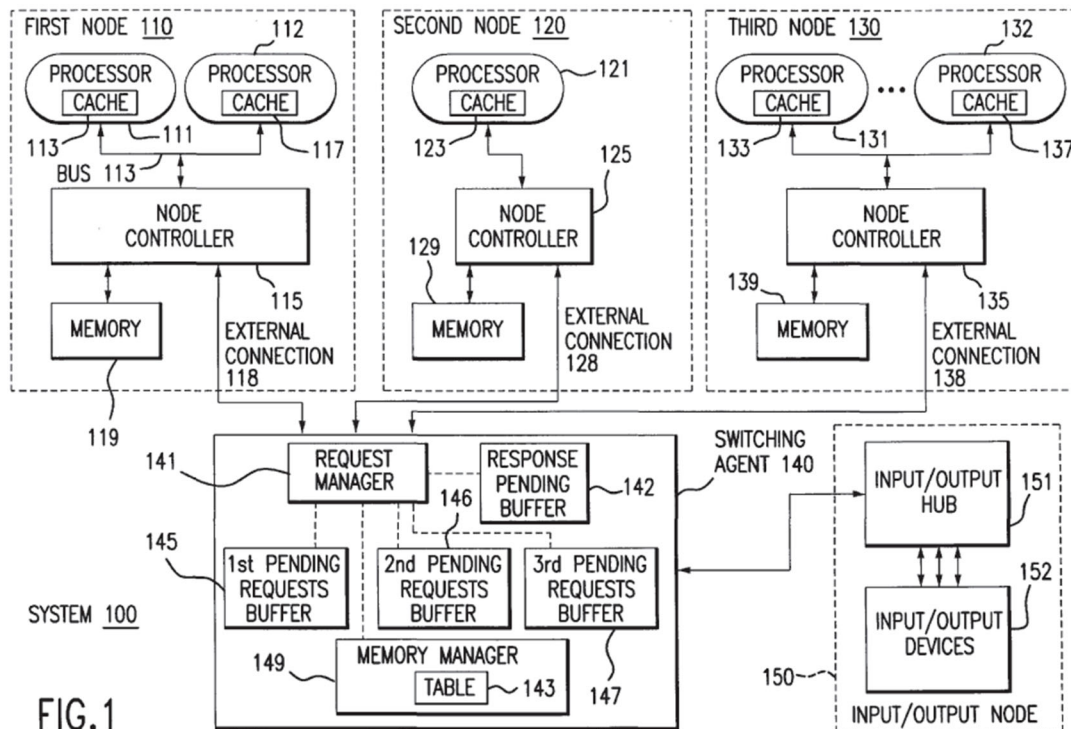
As depicted in Figure 1, Khare explains the “switching agent 140 contain[ing] a request manager 141, response pending buffer 142, first pending requests buffer 145, second pending requests buffer 146, third pending requests buffer 147, and memory manager 149.” Ex-1002, 4:1-4.



Ex-1002, Fig. 1 (annotated).

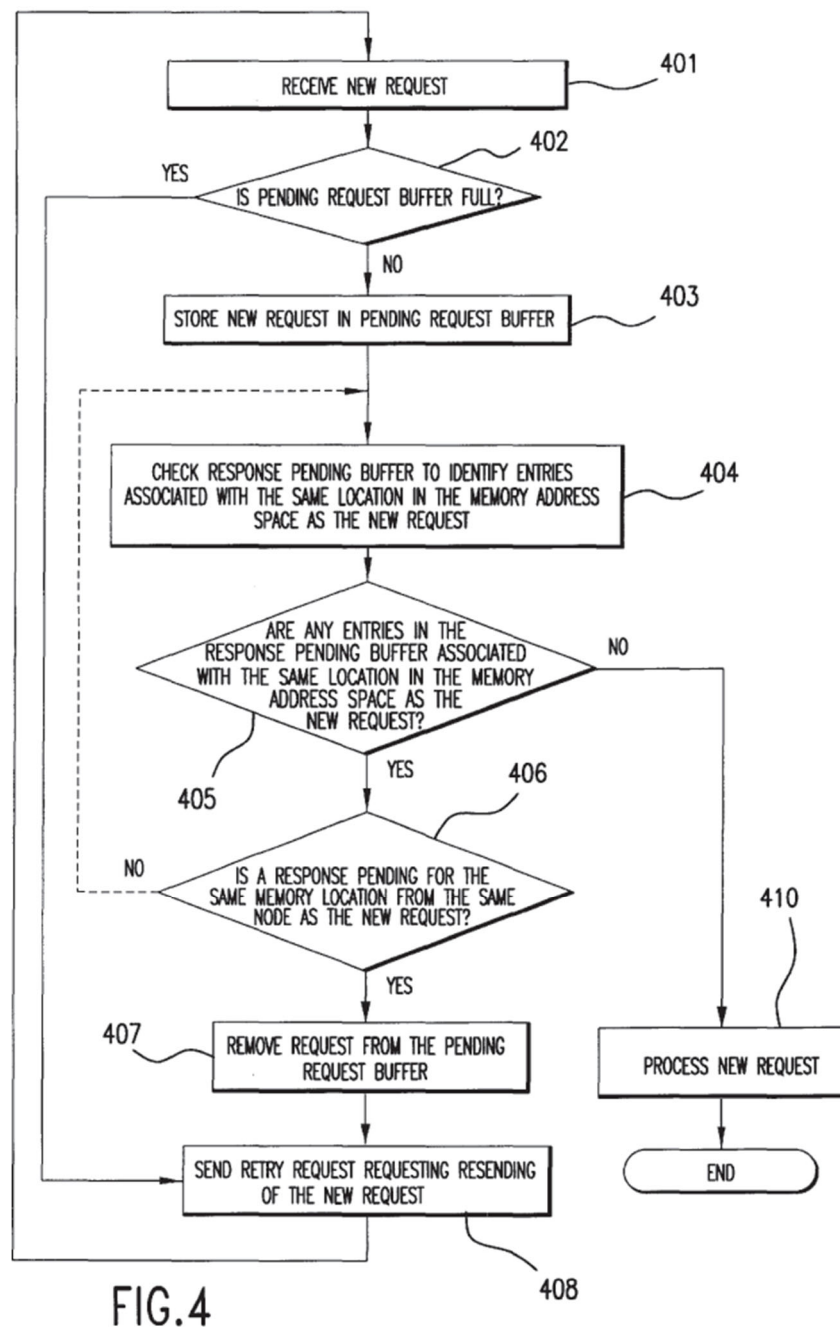
Request manager 141 (“a request module”) (containing circuits that are wired to perform operations, or program instructions, such as firmware that performs operations) “may be adapted to manage requests that are received by switching agent 140 from nodes [(composed of processors)] in system 100, or are to be sent to nodes by switching agent 140, in order to prevent livelock and starvation according to embodiments of the present invention.” Ex-1002, 4:5-12; Ex-1005, ¶ 89. The ordering of these requests is managed by request manager “to prevent livelock and starvation.” Ex-1002, 4:5-12; Ex-1002, 3:8-16 (explaining that the disclosed invention of Khare was designed to prevent starvation or livelock conditions where some requests are stuck in a perpetual loop always receiving a retry response from the switching agent). Pending requests buffers 145, 146, and 147 are utilized as a temporary repository of data where requests received but not yet processed by

switching agent 140 are stored. Ex-1002, 4:13-23. Similarly, a response pending buffer 147 is used to “store information about requests that have been sent out but for which a response has not yet been received (i.e., a response is pending)” (“*a request module configured to receive requests from the first processor bus and the second processor bus*”). Ex-1002, 4:17-20; Ex-1006, ¶ 89.



Ex-1002, Fig. 1.

Figure 4 demonstrates the order in which the requests stored in the buffers are processed.



Ex-1002, Fig. 4. When a new request is received, and a corresponding request buffer is not full, the new request is stored in a pending request buffer. Ex-1002, Fig. 4; Ex-1005, ¶ 94 (citing 6:10-8:65). The response pending buffer is then checked to identify entries associated with the same location in the memory address space as

the new request stored in a pending request buffer. Ex-1002, Fig. 4; Ex-1005, ¶ 94 (citing 6:10-8:65). The new request is then processed if there are no entries in the response pending buffer associated with the same location in the memory address space as the new request (*“to maintain proper ordering of the requests from each processor bus”*). Ex-1002, Fig. 4; Ex-1005, ¶ 94 (citing 6:10-8:65). These requests that a node may send and receive are a “memory read request, memory write request, cache snoop request, cache flush request, memory update request, cache line replacement request, input/output port read request, and input/output port write request.” Ex-1002, 5:5-10. Khare explicitly teaches that requests can also have ordering information, and the request manager maintains proper ordering of the requests from each processor bus (*“a request module configured to receive requests from the first processor bus and the second processor bus and to maintain proper ordering of the requests from each processor bus”*). Ex-1002, 5:10-12 (“Requests may contain fields such as . . . ordering semantics.”); Ex-1006, ¶¶ 92-93.

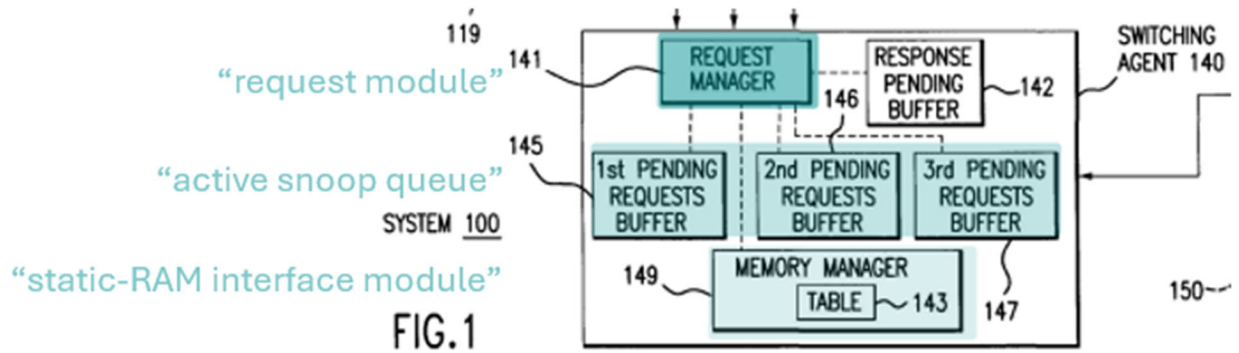
Accordingly, Khare teaches this limitation. Ex-1006, ¶ 86.

- vii. ***[1.7] an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests currently being processed and to prevent simultaneous multiple accesses to a single address in the static portion of the RAM; and***

Khare explains the “switching agent 140 contain[ing] a request manager 141, response pending buffer 142, first pending request buffer 145, second pending

request buffer 146, third pending request buffer 147, and memory manager 149.”
Ex-1002, 4:1-4. As discussed in limitation [1.6] request manager 141 is “*the request module*.” Ex.-1006, ¶¶ 86, 89.

Based on the ’409 Patent specification, “*an active snoop queue (ASQ)*” is “used to prevent multiple accesses to the same index simultaneously” by “includ[ing] one or more buffers which contain the indices of all requests which are currently active[.]” Ex-1001, 6:3-24. Khare teaches that first pending requests buffer 145, second pending requests buffer 146, and third pending requests buffer 147 correspond to “*an active snoop queue (ASQ) module*” as they “may be used to store requests that have been received by switching agent 140 but have not yet been processed by switching agent 140.” Ex-1006, ¶¶ 88, 90 (citing Ex-1002, 4:20-23); Ex-1002, 6:22-25 (“Second pending requests buffer 146 and third pending requests buffer 147 may contain the same structure as first pending requests buffer 145.”). Indeed, Figure 1 shows that the request manager (“*request module*”) is coupled to first, second, and third pending requests buffers (“*active snoop queue (ASQ) module*”) (“*an active snoop queue (ASQ) module is coupled to the request module*”). Ex-1006, ¶ 90.



Ex-1002, Fig. 1 (annotated).

Figure 4 shows a process by which requests in the memory address space 201 are managed in a multi-mode system. Ex-1002, 6:41-43. Khare explains that “the switching agent selects an entry from the pending requests buffer for processing which entry is not associated with the same location in the memory address space as any of the entries in the response pending buffer.” Ex-1002, 8:40-44; Ex-1006, ¶ 95. Khare also explains that “if a request from first pending request buffer 145 cannot enter the response pending buffer because the response pending buffer is processing another request to the same memory location, then another request from the first pending buffer 145 must be given a chance to check and enter the response pending buffer. The request that is stored in the entry selected from the pending requests buffer may then be processed as discussed above” (“*ASQ . . . configured to maintain a list of requests currently being processed*”). Ex-1002, 8:51-58; Ex-1006, ¶ 95.

Khare explains that as part of the process for receiving a request, the switching device 140 checks the response pending buffer (“*active snoop queue*”) “to identify

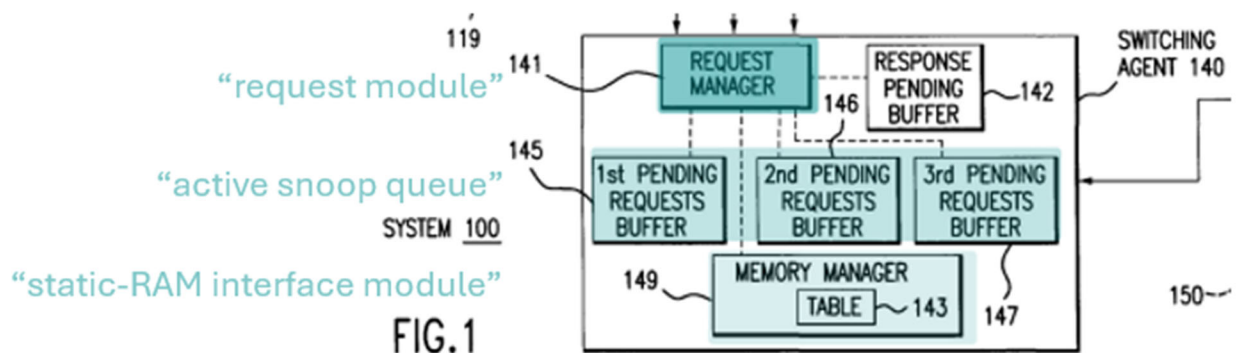
entries associated with the same location in the memory address space as the new request (404-405).” Ex-1002, 7:12-16. Khare explains that if an entry in the response pending buffer is identified as being associated with the same location as the new request (“*simultaneous multiple accesses to a single address*”), the switching agent checks two things: (1) whether the entry is associated with a request that was sent to the same node as the new request, and (2) whether a response to that request has not been received. Ex-1002, 7:16-23. Khare further explains that “[i]f the identified entry has an outstanding request to the node from which the new request was received, then the request is removed from the Pending Buffer and a retry response is sent to that node requesting resending of the new request (407-408).” Ex-1002, 7:23-27. Khare classifies this type of request retry as a “conflict induced retry,” where a conflicting entry to the same memory location in the response pending buffer causes the request to be rejected from the switching agent (“*prevent simultaneous multiple accesses*”). Ex-1002, 7:36-39. The switching agent 140 may then “receive a resent version of the new request, and the received resent version may be processed according to the same method.” Ex-1002, 7:40-42. By sending this retry message, Khare blocks the processor from making a conflicting multiple access to the same address (“*ASQ . . . configured . . . to prevent simultaneous multiple accesses to a single address*”). Ex-1002, 7:39-42; Ex-1006, ¶¶ 94-95. As depicted in Figure 2, Khare teaches that the memory being assessed in prevention of

simultaneous multiple access can be the cache memory (113, 123, 133, 117, 137) on each processor or the memory (119, 129, 139) connected to each node. Ex-1002, Fig. 2, 5:60-6:2. As discussed in limitation [1.4], the cache memory being assessed is “the static portion of the RAM.” Ex-1002, 3:46-48; Ex-1006, ¶ 78.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 86.

viii. [1.8] a static RAM interface module configured to access an address look-up table corresponding to data stored in the static portion of the RAM.

As depicted in Figure 1, Khare explains the “switching agent 140 contain[ing] a request manager 141, response pending buffer 142, first pending requests buffer 145, second pending requests buffer 146, third pending requests buffer 147, and memory manager 149.” Ex-1002, 4:1-4; Ex-1006, ¶ 87.



Ex-1002, Fig. 1 (annotated).

The '409 Patent refers to the “static RAM interface module” as a “tag RAM interface module” in the specification. Ex-1001, Fig. 2, 6:25-32 (“The tag RAM interface module is optimized to make efficient use of the bandwidth in the SRAM”). Further, the specification describes the “tag RAM” as being “a separate memory

storage device from main memory that keeps track of all cachelines in use by the CPUs” and “provides a look-up table for the cache state in cache memory.” Ex-1001, 6:26-30.

Khare provides a memory manager 149 that can access the SRAM and assist in efficiently processing requests by “identifying nodes that contain a copy in their private cache memory of information stored at a location in a memory address space that is associated with the request, sending requests that are based on the request being processed to each of said identified nodes, and storing an entry in the response pending buffer for each request sent.” Ex-1002, 4:42-47; Ex-1006, ¶ 87. Accordingly, the memory manager 149 is the “*static RAM interface module*.” Ex-1006, ¶ 88.

As shown in Figure 1, memory manager 149 includes a table 143 in a memory that is used to determine which nodes a request should be sent (“*a static RAM interface module configured to access an address look-up table*”). Ex-1002, 4:48-50; Ex-1006, ¶ 88. Khare states that “[p]rocessing of such requests may include identifying nodes that contain a copy [corresponding to data stored] in their private cache memory [the static RAM caches in the processors—i.e., the claimed ‘*static portion of the RAM*’] of information stored at a location in a memory address space that is associated with the request [corresponding to data stored].” Ex-1002, 4:40-43. Khare teaches that this is done in an “*address look-up table*.” Ex-1006, ¶ 91. In

particular, Khare explains that by “identifying nodes that contains a copy . . . of information stored at a location in a memory address space that is associated with the request,” this is done by memory manager 149, which “may include a table 143 in a memory that is used to determine to which nodes a request should be sent.” Ex-1002, 4:40-43, 4:47-49; Ex-1006, ¶ 91. Indeed, “memory manager 149 [(“static RAM interface module”)] may determine from table 143 [(“an address look-up table”)] that memory location A is cached in both cache 113 (in first node 110) and cache 117 (in second node 120)” (“configured to access an address look-up table corresponding to data stored in the static portion of the RAM”). Ex-1002, 6:13-15.

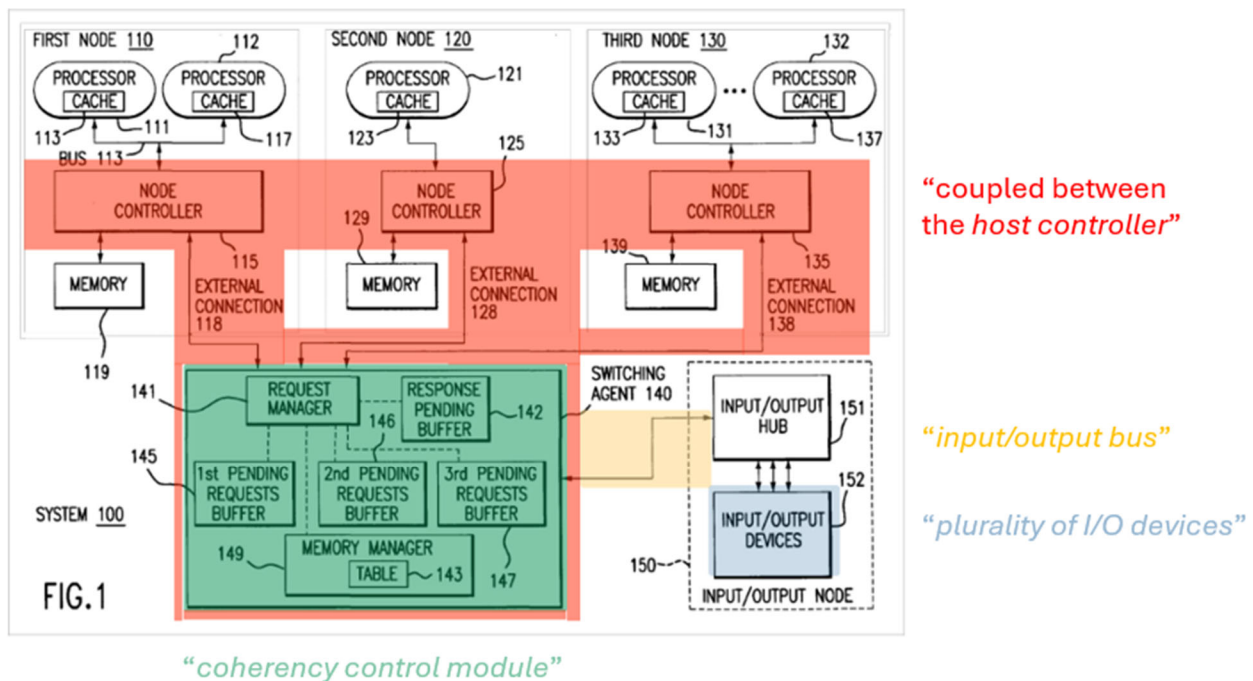
Accordingly, Khare teaches this limitation. Ex-1006, ¶ 86.

2. Claim 2

- i. ***[2.0] The computer system, as set forth in claim 1, comprising an input/output (I/O) bus coupled between the host controller and a plurality of I/O devices, wherein the I/O bus is operably coupled to the coherency control module and wherein the request module is configured to receive requests from the I/O bus.***

As discussed in limitation [1.0], Khare teaches “a computer system that includes processors, memory devices, and input/output devices” (“computer system”; “a plurality of I/O devices”). Ex-1002, 2:23-23. Khare’s Figure 1 shows input/output devices, located within the input/output node, which contain a plurality of buses, connecting the input/output devices to other components within the system (“input/output (I/O) bus”). Ex-1002, 2:23-25 (“Fig. 1 shows a system 100 which is

a computer system that includes processors, memory devices, and input/output devices.”); Ex-1002, 2:30-32 (“The components within a node may be connected to other components in that node through one or more busses.”); Ex-1002, 2:32-35 (“[E]ach node in system 100 has an external connection that may be used by the components within that node to communicate with components in other nodes.”); Ex-1002, 4:51-61; Ex-1006, ¶ 101.



Ex-1002, Fig. 1 (annotated); Ex-1006, ¶ 100.

Here, Figure 1 demonstrates that input/output devices are coupled via several buses (“input/output (I/O) bus”) to an input/output hub, containing bus interface logic, which is itself coupled to the switching agent (“the coherency control module”). Ex-1002, Fig. 1; Ex-1002, 4:50-59. “Coupled” involves “a direct connection, an indirect connection, an indirect communication, etc.” Ex-1002,

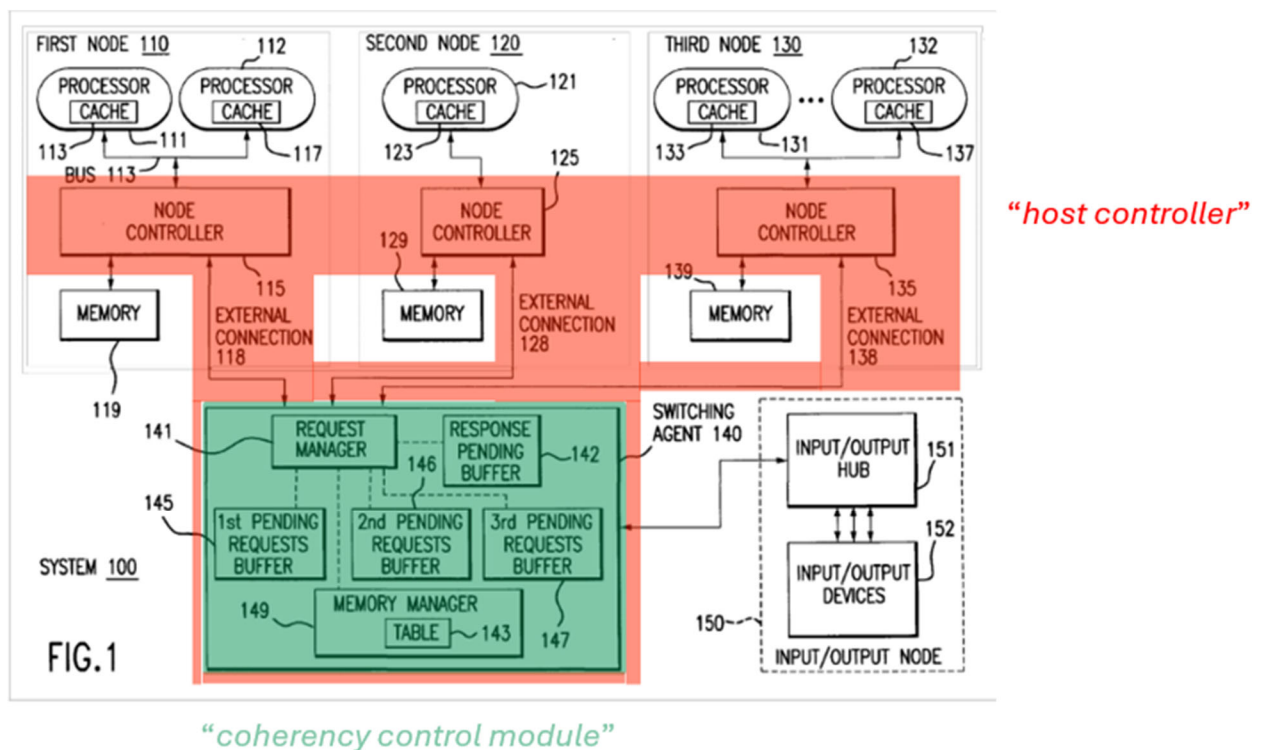
3:28–29. Indeed, Figure 1 depicts system 100, which “includes a first node 110, a second node 120, a third node 130, and an input/output node 150,” each of which are coupled to switching agent 140 (“*wherein the I/O bus is operably coupled to the coherency control module*”). Ex-1002, 3:24-27. And as discussed in limitation [1.6], the switching agent contains a request manager, managing requests received by switching agent 140 from nodes in system 100, including the input/output node, coupled together with input/output buses and containing input/output devices (“*wherein the request module is configured to receive requests from the I/O bus*”). Ex-1002, 3:66-4:12. Switching agent 140 coupled to node controller of the first, second, and third nodes (“*host controller*”), as discussed in limitation [1.1], is connected to the input/output node 150 via an input/output bus, which “contains an input/output hub 151 that is coupled to one or more input/output devices 152” (“*an input/output (I/O) bus coupled between the host controller and a plurality of I/O devices*”). Ex-1002, 4:50-52; Ex-1002, 3:37-49; Ex-1002, 3:50-55; Ex-1002, 3:28-29; Ex-1006, ¶ 102.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 103.

3. Claim 3

- i. [3.0] *The computer system, as set forth in claim 1, wherein the coherency control module is located within the host controller.*

As discussed in limitation [1.1], Khare teaches that switching agent 140 coupled to node controller of the first, second, and third nodes comprise the “*host controller*.” Ex.-1006, ¶¶ 76-77. As discussed in limitation [1.5], Khare teaches that switching agent 140 is “*the cache coherency control module*.” Ex-1006, ¶ 85. As demonstrated by Figure 1, “*the cache coherency module*” is located within “*the host controller*”:



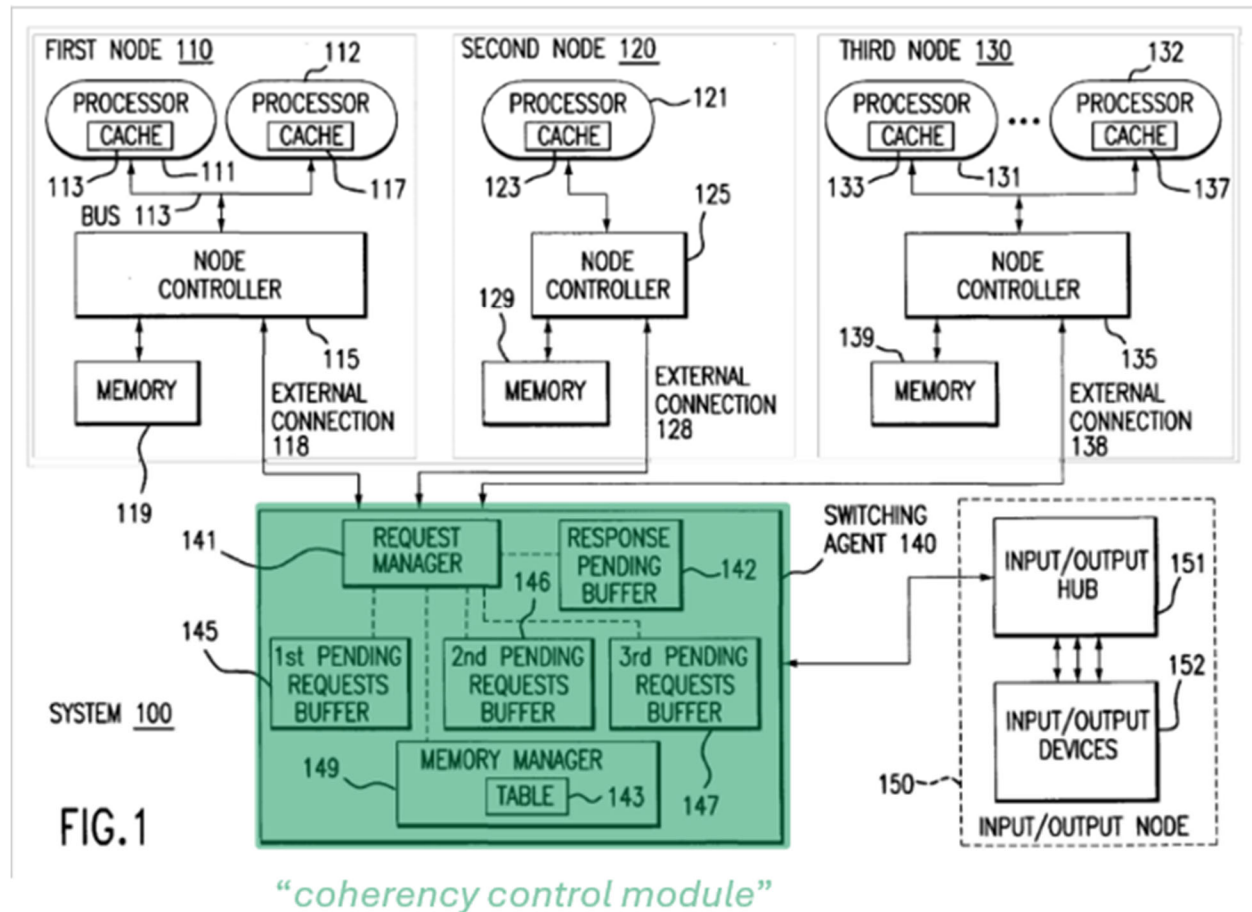
Ex-1002, Fig. 1 (annotated); Ex-1006, ¶¶ 107-08.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 109.

4. Claim 4

- i. ***[4.0] The computer system, as set forth in claim 1, wherein the coherency control module comprises a plurality of list structures corresponding to the first processor bus and the second processor bus.***

As discussed in limitation [1.0], Khare teaches “*the computer system.*” Ex-1006, ¶¶ 70-72. As discussed in limitation [1.5], Khare teaches that switching agent 140 (“*a coherency control module*”) contains a request manager 141, response pending buffer 142, first pending requests buffer 145, second pending requests buffer 146, third pending requests buffer 147, and memory manager 149.” Ex-1002, 4:1-4; Ex-1006, ¶ 77. As discussed in limitations [1.2] and [1.3], bus 113 coupling processors 111 and 112 in the first node is “*the first processor bus*” and the bus coupling processors 131 and 132 in the third node is “*the second processor bus.*” Ex-1006, ¶ 77.



Ex-1002, Fig. 1 (annotated); Ex-1006, ¶ 113.

The '409 Patent describes “*list structures*” as being necessary to “maintain the order of the requests in the READ and WRITE buffers.” Ex-1001, 8:16-18. The '409 Patent further describes that the “*list structures*” are used to “ensure that proper cycle order is maintained for each type of transaction.” Ex-1001, 8:29-30.

Khare teaches that the pending requests buffers (145, 146, 147) contained in the switching agent 140 (“*coherency control module*”) “may be used to store requests that have been received by switching agent 140 but have not yet been processed by switching agent 140.” Ex-1002, 4:20-23. Khare further explains that the buffers

store requests that “may contain fields such as . . . ordering semantics” and accordingly teaches the use of at least “*list structures*” in the pending requests buffers. Ex-1002, 5:11-13. Indeed, “first pending requests buffer 145 contains requests sent by first node 110, second pending requests buffer 146 contains requests sent by second node 120, and third pending requests buffer 147 contains requests sent by third node 130.” Ex-1002, 4:24-28; Ex-1005, ¶ 115. First pending requests buffer 145 and third pending requests buffer 147 therefore constitute “*a plurality of list structures*” as bus 113 of the first node (“*the first processor bus*”) corresponds and is coupled to the first pending requests buffer 145 and the processor bus of the third node (“*the second processor bus*”) corresponds and is coupled to the third pending requests buffer 147 (“*a plurality of list structures corresponding to the first processor bus and the second processor bus*”). Ex-1006, ¶ 115. And Khare explains that “switching agent 140 may have more or less pending request buffers” (“*coherency control module comprises a plurality of list structures*”). Ex-1002, 4:28-29; Ex-1006, ¶¶ 113, 115. Khare also confirms that a list of requests currently being processed within first pending requests buffers 145 and 147 as Khare states that “[a]s shown in FIG. 3, first pending requests buffer 145 may contain entries 1 to n and response pending buffer 142 may contain entries to m,” which is an additional teaching of list structures. Ex-1006, ¶ 117 (quoting Ex-1002, 6:25-27).

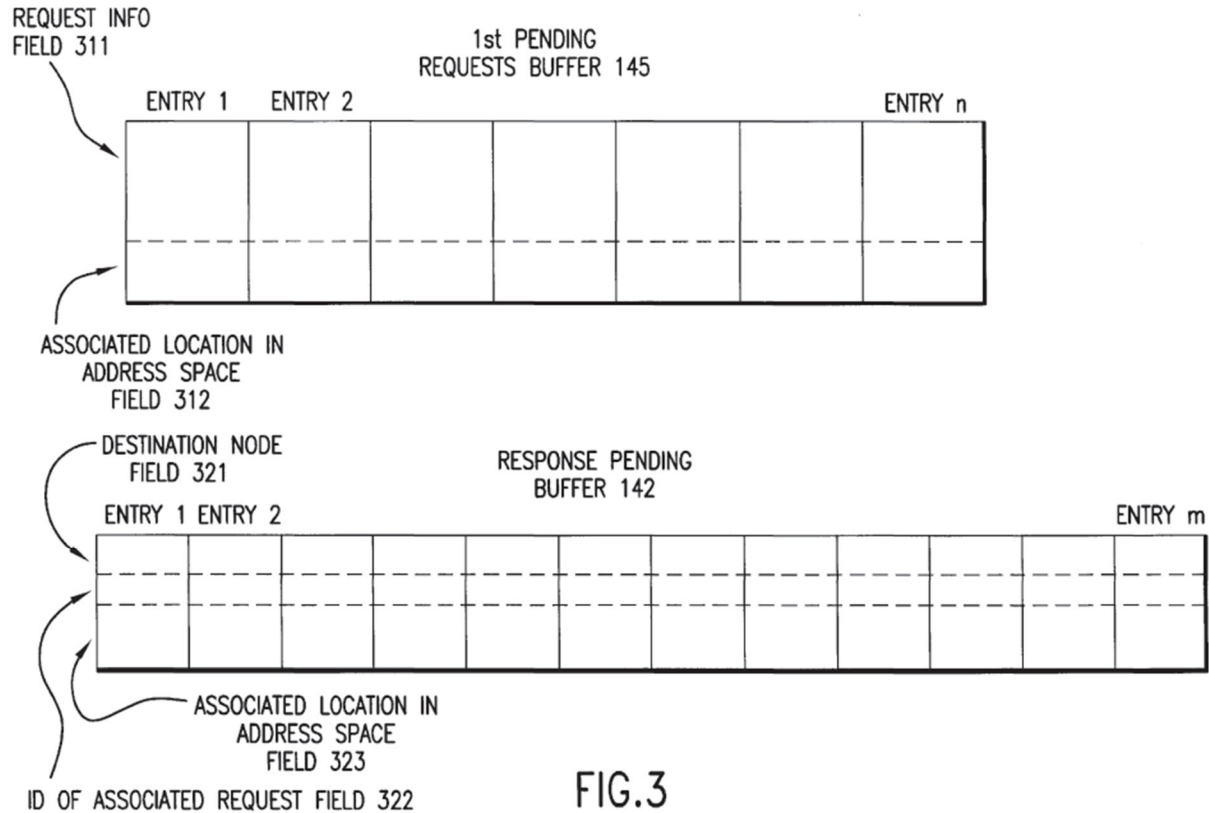


FIG. 3

Ex-1002, Fig. 3; Ex-1006, ¶ 116.

Accordingly, Khare would have rendered this limitation obvious. Ex-1006, ¶ 118.

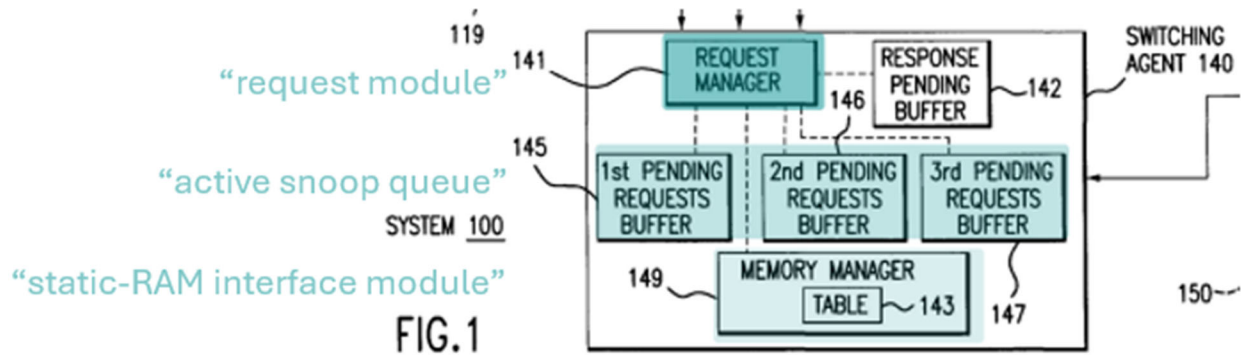
5. Claim 7

- i. *[7.0] The computer system, as set forth in claim 4, wherein the request module comprises a plurality of bypass paths configured to provide a direct request path from the first and second processor buses to the static RAM interface module, thereby bypassing the list structures.*

Khare explains switching agent 140, containing “a request manager 141, response pending buffer 142, first pending requests buffer 145, second pending

requests buffer 146, third pending requests buffer 147, and memory manager 149.”

Ex-1002, 4:1-4.



Ex-1002, Fig. 1 (annotated).

As discussed in limitations [1.2] and [1.3], bus 113 coupling processors 111 and 112 in the first node is “*the first processor bus*” and the bus coupling processors 131 and 132 in the third node is “*the second processor bus.*” Ex-1006, ¶ 77. As discussed in limitation [1.6], request manager 141 is “*the request module,*” which is located within switching agent 140, and “may be adapted to manage requests that are received by switching agent 140 from nodes [(composed of processors)] in system 100, or are to be sent to nodes by switching agent 140, in order to prevent livelock and starvation according to embodiments of the present invention.” Ex-1002, 4:5-12; Ex-1006, ¶ 89. As discussed in limitation [1.8], memory manager 149 (“*the static RAM interface module*”), also located in switching agent 140, includes a table 143 in a memory that is used to determine which nodes a request should be sent. Ex-1002, 4:48-50; Ex-1006, ¶¶ 87, 122. Further, the pending requests buffers

145, 146, and 147 (“*the list structures*”) are located in the switching agent 140 and connect (118, 128, 138 respectively) the processor buses to the memory manager 149 (“*static RAM interface module*”) through the request manager 141 (“*request module*”). See Ex-1002, Fig. 1. The pending request buffers “store requests that have been received by switching agent 140 but have not yet been processed by switching agent 140” and as discussed in limitation [4.0] maintain the order in which the requests are received. Ex-1002, 3:66-4:49; Ex-1006, ¶ 115.

Figure 1 shows pending requests buffers 145, 146, and 147 (“*the list structures*”) are directly connected to request manager 141 (“*request module*”), not memory manager 149 (“*the static RAM interface module*”). Ex-1002, Fig. 1; Ex-1006, ¶ 124. Khare explains that “[r]equest manager 141 may be adapted to manage requests that are received by switching agent 140 from nodes in system 100, or are to be sent to nodes by switching agent 140.” Ex-1002, 4:7-9; Ex-1006, ¶ 123. Khare also explains that “[m]emory manager 149 may be adapted to process requests that are received by switching agent 140.” Ex-1002, 4:38-40; Ex-1006, ¶ 123. Here, request manager 141 (“*request module*”) may handle requests received from the processor buses or memory manager 149 (“*static RAM interface module*”) can be configured to process those requests directly. Ex-1006, ¶ 123.

Khare further explains that if, for example, “switching agent 140 received from third node 130 a request to snoop the memory location designated as A (in FIG.

2), memory manager 149 may determine from table 143 that memory location A is cached in both cache 113 (in first node 110) and cache 117 (in second node 120). Memory manager 149 may then cause snoop requests that are associated with location A to be sent to first node 110 and second node 120” (“*plurality of bypass paths*”). Ex-1002, 6:10-16. Here, Khare teaches that memory manager 149 directly processes requests that are received by switching agent 140 including from bus 113 coupling processors 111 and 112 in the first node is “*the first processor bus*” and the bus coupling processors 131 and 132 in the third node is “*the second processor bus.*” Ex-1002, 3:66-4:49; Ex-1006, ¶ 127. For requests not processed immediately by memory manager 149, they are instead placed into pending requests buffers 145, 146, and 147 (“*list structures*”)—“In one embodiment, a pending request buffer is used to store new requests if the memory manager 149 (“*static RAM interface module*”) is not ready to process the new request (e.g., if it is busy processing another request). In this embodiment, processing a new request may include determining if the memory manager is ready to process the request and, if not, storing the request in a pending request buffer.” Ex-1002, 8:58-65; Ex-1006, ¶ 126. This functionality of request manager (“*request module*”) “*comprises*” a “*plurality of bypass paths configured to provide a direct request path*” for requests from bus 113 coupling processors 111 and 112 in the first node (“*the first processor bus*”) and for the bus coupling processors 131 and 132 in the third node (“*the second processor bus*”) to

be processed by memory manager 149 “*static RAM interface module*,” “*thereby bypassing*” pending requests buffers 145, 146, and 147 (“*the list structures*”). Ex-1006, ¶ 126-27.

Accordingly, Khare teaches this claim element. Ex-1006, ¶ 128.

6. Claim 8

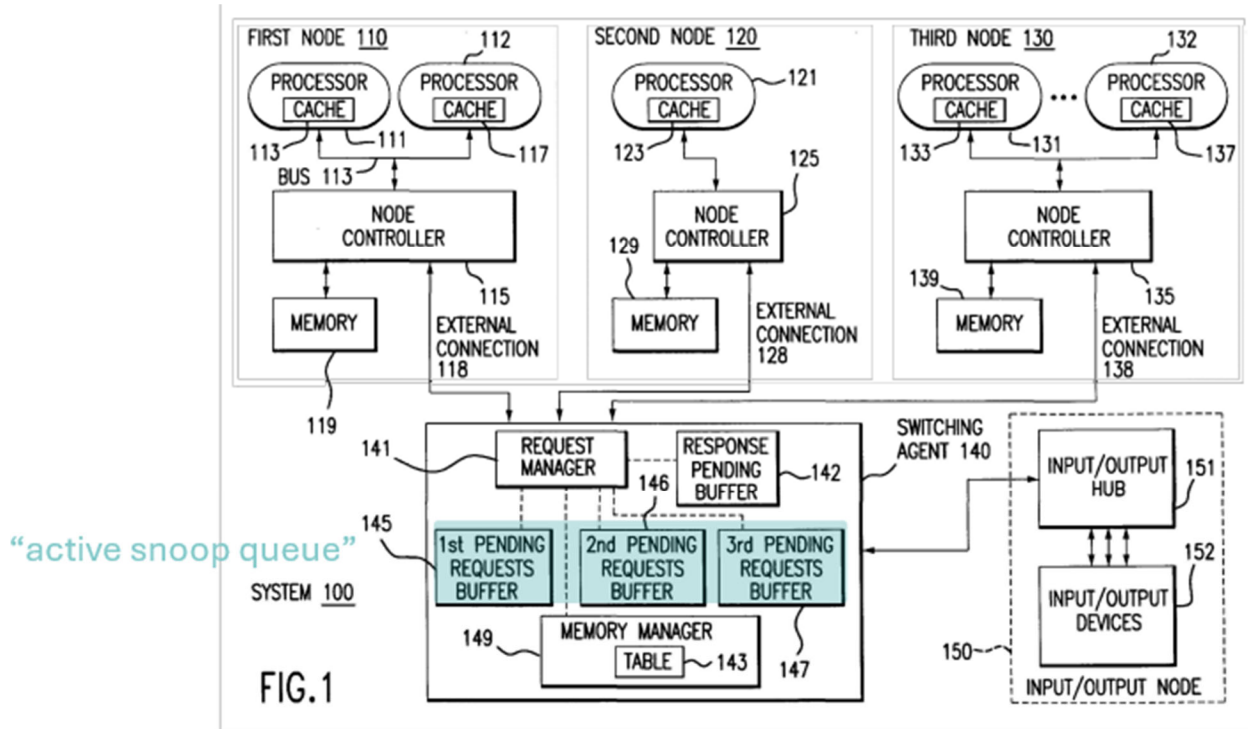
- i. ***[8.0] The computer system, as set forth in claim 1, wherein the active snoop queue module comprises a plurality of active snoop queues, each active snoop queue configured to maintain a list of requests currently being processed and to function independently with respect to each other.***

As discussed in limitation [1.7], Khare explains that the switching agent 141 (“*coherency control module*”) contains pending requests buffers 145, 146, 147 (“*active snoop queue module*”) which maintain a list of requests currently being processed (“*maintain a list of requests currently being processed*”). Ex-1002, 3:66-4:49; Ex-1006, ¶¶ 132-34.

Khare also teaches that the pending requests buffers 145, 146, 147 are a “*plurality of active snoop queues*.” Ex-1006, ¶¶ 135-37. Khare states that “[s]witching agent 140 may process requests as follows. If switching agent 140 receives from third node 130 a **request to snoop** the memory location designated as A (in FIG. 2), memory manager 149 may determine from table 143 that memory location A is cached in both cache 113 (in first node 110) and cache 117 (in second node 120). Memory manager 149 may then cause snoop requests that are associated

with location A to be sent to first node 110 and second node 120.” Ex-1002, 6:10-16. Khare also explains that “[f]or example, first node 110 may have sent a request to read from a location in the memory address space or a **request to snoop** for a location in the memory address space. Prior to receiving the new request, switching agent 140 may have sent one or more requests to nodes in the system. During the process of sending requests for each memory location, switching agent 140 may have created a new entry associated with the memory location in a response pending buffer.” Ex-1002, 6:49-57. Khare further explains that “[p]rocessing the request may include identifying nodes that contain a copy of the location associated with the request in their private cache memory, **sending snoop requests** that are based on the new request to each of the identified nodes (503), and updating the entry in the response pending buffer to indicate that a response is pending from each of the identified nodes (504).” Ex-1002, 8:8-14. Khare then explains that “if a new request to read from a location that is contained in memory 129 is received from first node 110, a memory manger in the switching device may check a table and determine that a request to read from that location should be sent to second node 120. **A snoop request** to that location may be sent to the second node 120, and this request may be based on the received new request. In addition, the entry associated with the request stored in the response pending buffer 142 may be updated to include a destination

node (second node 120), an ID of the associated request, and the location in the memory address space that is associated with the request.” Ex-1002, 8:15-26.



Ex-1002, Fig. 1 (annotated).

Khare teaches that the pending requests buffers operate independently with respect to each other: “In the embodiment shown in FIG. 1, first pending requests buffer 145 contains requests sent by first node 110, second pending requests buffer 146 contains requests sent by second node 120, and third pending requests buffer 147 contains requests sent by third node 130” (“*function independently with respect to each other*”). Ex-1002, 4:23-28. Ex-1006, ¶¶ 132, 134.

Accordingly, Khare would have rendered this limitation obvious. Ex-1006, ¶ 138.

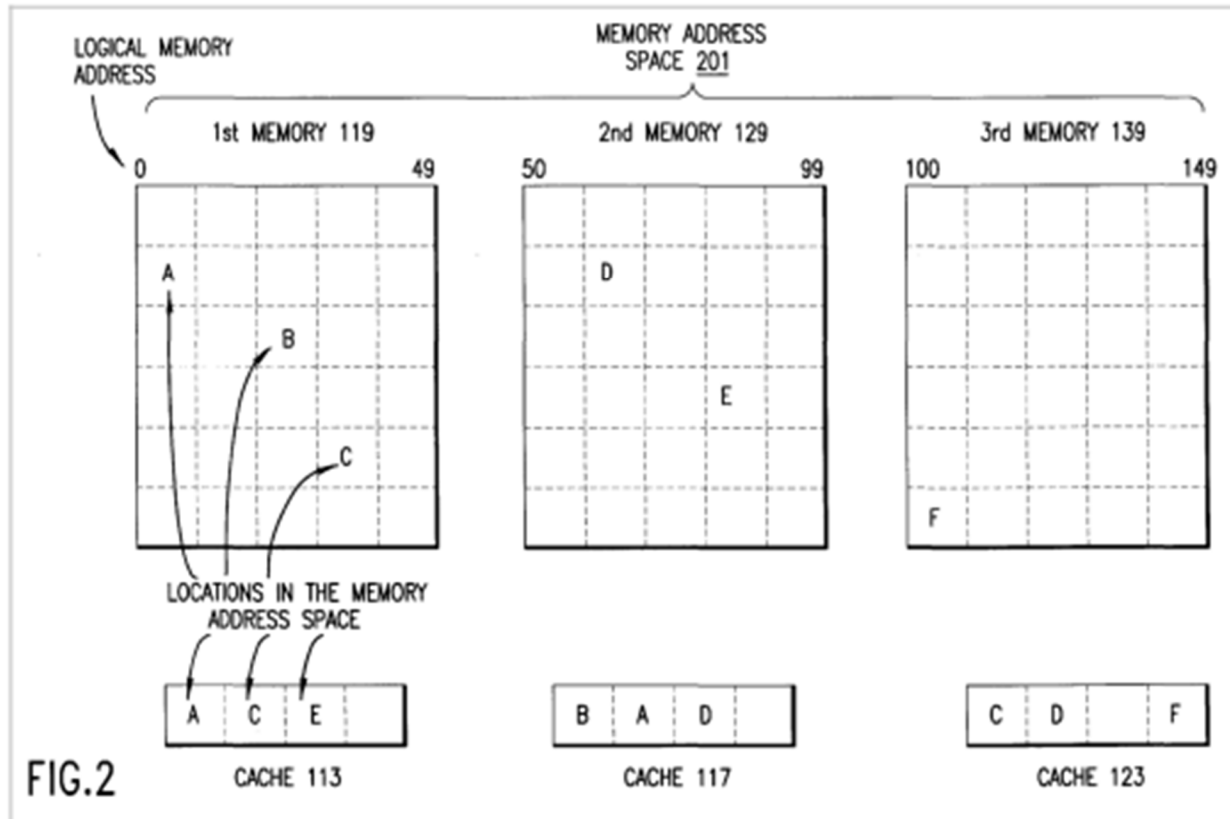
7. Claim 9

- i. ***[9.0] The computer system, as set for the in claim 1, wherein the static RAM interface module comprises a plurality of static RAM interfaces, each static RAM interface corresponding to a segment of the static portion of the RAM.***

As discussed in limitation [1.8], Khare teaches that memory manager 149 is “*the static RAM interface module.*” Ex-1006, ¶ 88. Khare explains that memory manager 149 (“*a static RAM interface module*”) may process requests received by switching agent 140 by (1) identifying nodes that contain a copy in their private cache memory of information stored at a location in a memory address space associated with the request; (2) sending requests based on the request being processed to each of the identified nodes; and (3) storing an entry in the response pending buffer for each request sent. Ex-1002, 4:40-49.

As discussed in limitation [1.4], Khare explains that the cache memory within each processor “may be Level 2 (L2) cache memories that are comprised of static random access memory.” Ex-1002, 3:46-48. Khare also explains that memory 119, 129, and 139 depicted in each node may be composed of dynamic random access memory (DRAM). Ex-1002, Fig. 1; Ex-1002, 3:44-46. Khare explains that memories 119, 129, and 139 are DRAM main memories and their contents are copied into SRAM caches 113, 117, and 123—“The information in any particular location in the memory address space 201 may be cached in one or more of the

caches in the system . . .” See Ex-1002, 5:63-6:9; Ex-1006, ¶¶ 143-44. Khare also states that Figure 2 shows caches 113, 117, and 123, and “the set of data currently being used by a microprocessor may be copied from a system memory device such as a dynamic random access memory (DRAM) into a relatively smaller but faster cache memory device such as a static random access memory (SRAM).” Ex-1002, 5:54-55, 1:26-30. Khare also explains that memory manager 149 (“*static RMA interface module*”) maintains a plurality of entries that represent coherence information for all data cached in caches 113, 117, and 123 (“*a plurality of static RAM interfaces*”), and the coherence information corresponds to all of the cached contents in the system, which includes all data corresponding to the segment of the static portion of the RAM. Ex-1002, 6:10-17; Ex-1006, ¶¶ 145-46. Figure 2 shows that each of the external memory blocks 119, 129, and 130 are copied to SRAM caches 113, 117, and 123, and coherency is maintained by the memory manager 149 (“*the static RAM interface module*”) resulting in “*the static RAM interface module comprises a plurality of static RAM interfaces.*” Ex-1006, ¶¶ 142-146; Ex-1002, 5:63-6:9.



Ex-1002, Fig. 2; Ex-1006, ¶ 143.

Khare states that “[f]igure 2 also shows certain locations designated as A, B, C, D, E, and F storing data and instructions, which may be cached in one or more of the caches in the system.” Ex-1002, 5:60-65. For example, “a copy of the information in the location designated A in first memory 119 is cached in caches 113, 117, and 123,” and “a copy of the information in the location designated B in first memory 119 is cached in cache 117” (“each static RAM interface corresponding to a segment of the static portion of the RAM”). Ex-1002, 5:66-6:2; Ex-1006, ¶¶ 143, 145-46.

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 147.

8. Claim 10

- i. ***[10.0] A coherency control module configured to control access to cache memory in a computer system, the coherency control module comprising:***

Khare teaches this limitation. See claim [1.5]. As discussed in limitation [1.0], Khare explains “a computer system that includes processors, memory devices, and input/output devices” (“*a computer system*”). Ex-1002, 2:23-25; Ex-1006, ¶¶ 151-52. Within the computer system, Khare teaches the switching agent 140 (“*coherency control module*”), which is responsible for maintaining coherence among one or more caches, which are contained within the processors of each node (“*a coherency control module configured to control access to cache memory in a computer system*”). Ex-1002, 5:21-24; Ex-1006, ¶ 156.

Accordingly, Khare teaches this limitation. Ex-1006, ¶¶ 148-67.

- ii. ***[10.1] a request module configured to receive requests from a plurality of buses in a computer system and to maintain proper ordering of the requests from each bus;***

Khare teaches this limitation. See claim [1.6]; Ex-1006, ¶¶ 148-67. Further, the bus 113 of the first node coupling processors 111 and 112 and the bus coupling processors 131 and 132 of the third node are “*a plurality of buses in a computer system*” for the same reasons described in limitation [1.6]. Ex-1006, ¶ 157.

- iii. ***[10.2] an active snoop queue (ASQ) module coupled to the request module and configured to maintain a list of requests from all of the buses currently being processed and to prevent multiple accesses to a single address in the cache memory simultaneously; and***

Khare teaches this limitation. See claim [1.7]; Ex-1006, ¶¶ 148-67.

- iv. ***[10.3] a static RAM interface module configured to access an address look-up table corresponding to data stored in the cache memory.***

Khare teaches this limitation. See claim [1.8]; Ex-1006, ¶¶ 148-67.

9. Claim 11

- i. ***[11.0] The coherency control module, as set forth in claim 10, wherein the plurality of buses comprise processor buses.***

As discussed in limitation [10.0], Khare teaches the switching agent 140 is the “coherency control module.” Ex-1006, ¶ 156. Also, as discussed in limitation [1.6],], Khare teaches that bus 113 of the first node coupling processors 111 and 112 is “the first processor bus” and the bus coupling processors 131 and 132 of the third node is “the second processor bus.” Ex-1002, Fig. 2; Ex-1006, ¶ 77. These buses are “a plurality of buses” which connect processors within the first and third nodes “comprise processor buses.” Ex.-1002, 3:27-29 (“First node 110, includes processor 111, processor 112, and node controller 115, which are coupled to each other by bus 113.”); Ex-1002, 3:50-65; Ex-1006, ¶ 173.

Accordingly, Khare teaches this limitation. Ex-1006, ¶¶ 168-75.

10. Claim 12

- i. *[12.0] The coherency control module, as set forth in claim 10, wherein the coherency control module comprises a plurality of list structures corresponding to the first processor bus and the second processor bus.*

Khare teaches this limitation. See claim [4]; Ex-1006, ¶¶ 176-84.

11. Claim 15

- i. *[15.0] The coherency control module, as set forth in claim 12, wherein the request module comprises a plurality of bypass paths configured to provide a direct request path from the first and second processor buses to the static RAM interface module, thereby bypassing the list structures.*

Khare teaches this limitation. See claim [7]; Ex-1006, ¶¶ 185-94.

12. Claim 16

- i. *[16.0] The coherency control module, as set forth in claim 10, wherein the active snoop queue module comprises a plurality of active snoop queues, each active snoop queue configured to maintain a list of requests currently being processed and to function independently with respect to each other.*

Khare would have rendered this limitation obvious. See claim [8]; Ex-1006, ¶¶ 195-204.

13. Claim 17

- i. *[17.0] The coherency control module, as set forth in claim 10, wherein the static RAM interface module comprises a plurality of static RAM interfaces, each static RAM interface corresponding to a segment of the static portion of the RAM.*

Khare teaches this limitation. See claim [9]; Ex-1006, ¶¶ 205-13.

B. Ground 2: Claims 5-6 and 13-14 Are Obvious Based On Khare and Kirkman

1. Motivation to Combine

Khare combined with Kirkman teaches “*list structures*” comprising of “*two-dimensional doubly-linked lists with head and tail pointers.*” Ex-1001, Cls. 5 & 13.

It would have been obvious to include a known element (Kirkman’s doubly linked list data structure) into another mechanism (Khare’s cache-coherence engine, using data structures) that yield predictable results (the list structure of Khare using a two-dimensional version of the commonplace doubly-linked list described in Kirkman) for the reasons explained below. Ex-1002; Ex-1004; Ex-1006, ¶ 237.

As discussed above, the ’409 Patent describes “*list structures*” as being necessary to “maintain the order of the requests in the READ and WRITE buffers.” Ex-1001, 8:16-18. The ’409 Patent further describes that the “*list structures*” are used to “insure that proper cycle order is maintained for each type of transaction” and are generally “designed as two dimensional doubly-linked lists.” Ex-1001, 8:28-30.

Khare teaches a method of managing inter-processor requests in a multi-processor system and describes that this is commonly accomplished through hardware structure implementations. Ex-1006, ¶ 215. However, Khare discloses that at least the switching agent 140 managing the pending requests buffers 145, 146, 147 and the memory manager 149, processing request received by switching agent

140, could be implemented in software. Ex-1002, 4:13-34, 35-39; Ex-1006, ¶¶ 216, 217-218. Accordingly, a POSITA would understand that Khare presents a system in which one or more components could be lists implemented as data structures in software. Ex-1006, ¶ 219. Further, as discussed above, Khare teaches that the pending requests buffers (145, 146, 147) contained in the switching agent 140 “may be used to store requests that have been received by switching agent 140 but have not yet been processed by switching agent 140.” Ex-1002, 4:20-23. Khare further explains that the buffers store requests that “may contain fields such as . . . ordering semantics” and accordingly teaches the use of “*list structures*” in the pending requests buffers. Ex-1002, 5:11-13.

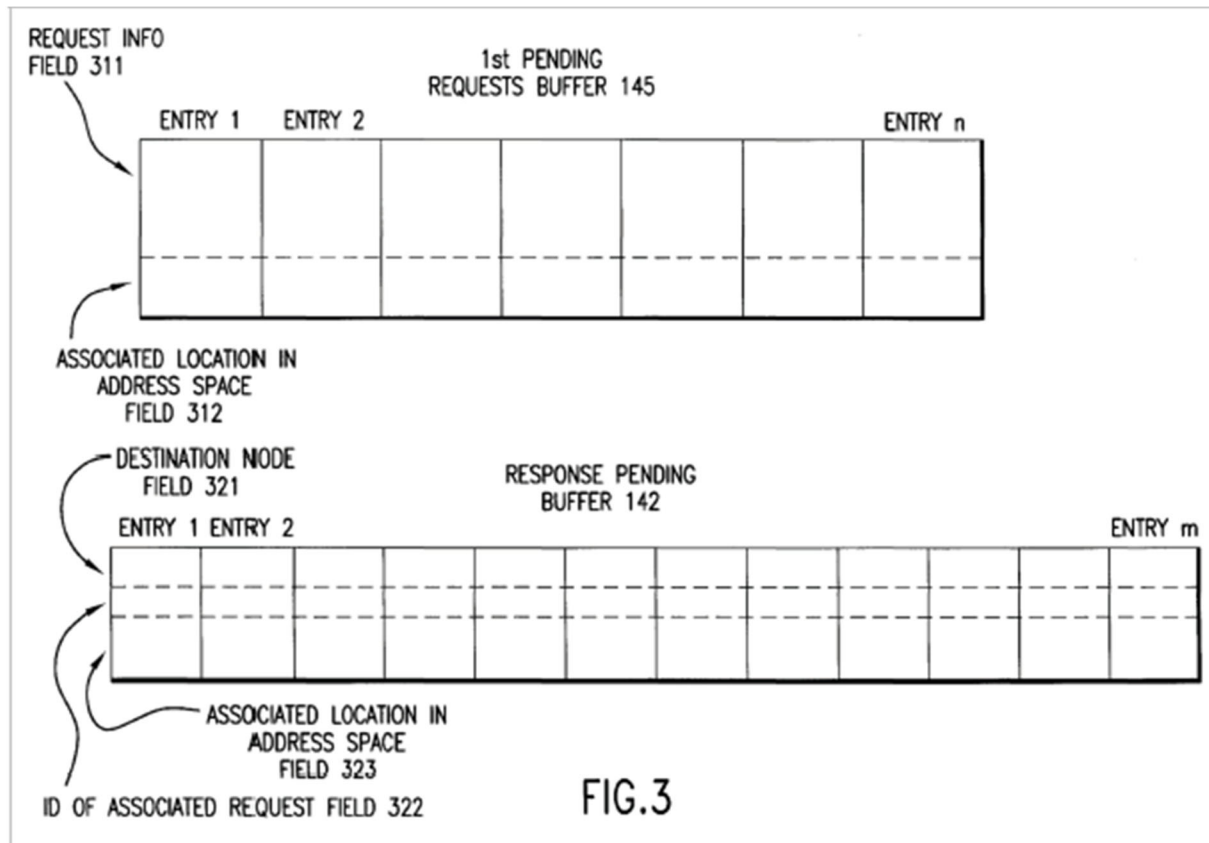
Khare also explains that the switching agent 140 may select one of the requests for processing from a pending requests buffer, and “the selection algorithm is fair among all the pending request buffers such that the requests from each pending buffer gets a fair chance to be processed and enter the response pending buffer.” Ex-1002, 8:40-47. Indeed, Khare explains that the “selection algorithm is also fair among requests within a pending requests buffer such that each request in a pending buffer gets a fair chance to check the response pending buffer and enter the response pending buffer.” Ex-1002, 8:47-51. As shown in Khare’s Figure 3 below, Khare demonstrates cross-list dependencies through the pending requests buffers 145, 146, 147 and response pending buffer 142 containing entries that reference the associated

memory address that corresponds to their respective memories. Ex-1002 6:17-8:65, 4:35-49, 4:66-5:13; Ex-1006, ¶¶ 248-49. Further, Khare teaches that entries in the buffer are removed from the buffer as part of the request-processing and administrative steps. Ex-1006, ¶ 250.

Khare describes the operations that can be performed on a software-based pending request buffer. Ex-1006, ¶ 219; Ex-1002, 6:10-8:65. Khare describes the system processing requests as they arrive and not necessarily in the order in which their entries were initially stored in the buffer. Ex-1006, ¶ 220; Ex-1002, 6:10-8:65. A POSITA interested in implementing this component in software would have been motivated to seek out a software data structure that implements a list of data entries and allows for searching for data within the list, inserting new entries into the list, and removing of entries from the list. Ex-1006, ¶ 221.

Kirkman teaches that a common data structure for a POSITA at the time was the linked list and the preferred embodiment would have been the doubly-linked list. Ex-1004, 1:44-46, 49-53 (“Software designers are well aware of the need . . . to arrange data in a logical manner . . . Although almost any custom data structure may be created, several common types of data structures are used over and over again by software designers. One of the most ubiquitous is the linked list, which may be singly linked or doubly linked.”), 2:7-9. Kirkman specifically discloses a software data structure that implements a list of data entries and allows for searching for data

within the list, inserting new entries into the list, and removing of entries from the list. Ex-1006, ¶¶ 222-23; Ex-1004, 6:16-24. One embodiment of this system is exemplified in Figure 3:



Ex-1004, Fig. 3. Kirkman further teaches in Figure 11 an example of intersecting lined lists that produce an orthogonal, “*two-dimensional*” form, due to the data in the lists stretching in two or more dimensions. Ex-1006, ¶ 227; Ex-1004, Figs. 11a, 11b, 44:25-45:39.

Providing further perspective on Kirkman, Plum is a similar reference, entitled “Reliable Data Structures in C.” Ex-1006, ¶ 228; Ex-1020. Indeed, the data

structure discussed in Plum is similar to those described in Kirkman, providing additional insight into the use of doubly linked lists. Ex-1020; Ex-1006, ¶ 230.

Plum, discussing double-ended queues, states that double-ended queues are also known as doubly linked lists, contain a head and tail pointer, allow for both forward and backward traversal to search through the list for matching data entries, a new node can be inserted at any point in the list, and a node can easily be removed from any point in the list. Ex-1006, ¶ 229; Ex-1020. The ease of insertion and removal is due to the requiring only that the pointers to adjacent nodes be updated. Ex-1006, ¶ 229; Ex-1020.

A POSITA would have been motivated to combine the teachings of Khare and Kirkman to use the doubly-linked list described in Kirkman to enable efficient list removal for requests that have already been processed. Ex-1006, ¶ 231. Indeed, a POSITA would have known that it is much harder to remove entries from the middle of a list when implemented as a singly-linked list. Ex-1006, ¶ 234. Specifically, a POSITA would have understood that Kirkman's teachings would optimize the cross-list searching being performed by switching agent 140 through eliminating the need for linear list searching. Ex-1006, ¶ 235. This optimization is provided by implementing a doubly linked list resembling Kirkman's disclosure of intersecting linked lists producing an orthogonal form. Ex-1006, ¶¶ 235-36.

A POSITA would further recognize that doubly-linked lists were an obvious and well-known structure for storing lists of information, as taught in Kirkman, and therefore, would be an obvious way of implementing Khare's request buffers. Ex-1006, ¶¶ 251-52. For example, Khare's data structures, depicted in Figure 3, and associated description indicate how a doubly-linked list would have been used by a POSITA. Ex-1002, 6:17-8:65; Ex-1006, ¶ 233. Further, as early as the 1980s and 1990s, Plum demonstrates that doubly-linked lists were popular data structures for handling data sets requiring searching for data within the set, inserting new entries into the set, and removing of entries from the set. Ex-1006, ¶ 228; Ex-1020.

Both Khare and Kirkman are analogous art to the '409 Patent. Khare and Kirkman are both directed to a similar field of endeavor of the '409 Patent—efficient processing of data structures in systems using lists. Ex-1002, 1:5-15; Ex-1004, 2:33-49; *see* Ex-1001, 1:9-12; Ex-1006, ¶¶ 220-22.

Second, both Khare and Kirkman are reasonably pertinent to a problem described by the '409 Patent—efficiency of data lookup. *See* Ex-1001, 3:21-34. Indeed, Khare describes the importance of efficient data lookup to avoid “livelock” circumstances in memory units. Ex-1002, 1:42-62. And Kirkman describes the need for making improvements to commonly used software structures, in particular linked lists, which are more efficient by being which is “less disruptive of tasks executing in other processors.” Ex-1004, 1:39-43. 2:48-49; Ex-1006, ¶ 222-23.

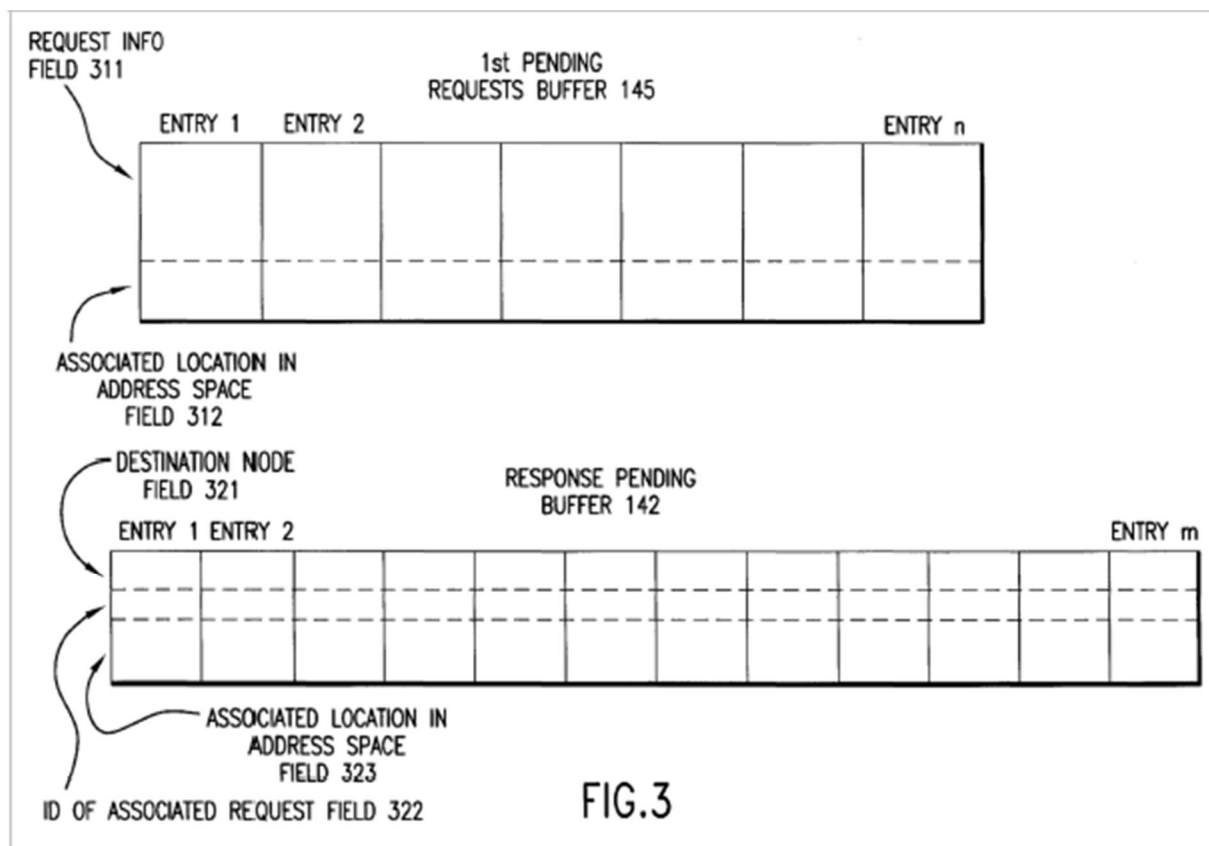
A POSITA would reasonably expect to succeed in using a two-dimensional version of Kirkman's doubly-linked list as the “*list structures*” in Khare since the system is already designed to implement list structures of which the doubly-linked list is one of the most common types for a POSITA. Ex-1004, 1:44-46, 49-53; Ex-1006, ¶¶ 231, 233. Further, a POSITA would have expected to succeed due to Kirkman's disclosure of intersecting linked lists that produce an orthogonal form resembling the cross-list pointers described in Khare. Ex-1006, ¶¶ 233, 236-238, 252.

2. Claim 5

- i. ***[5.0] The computer system, as set forth in claim 4, wherein the list structures comprise two-dimensional doubly-linked lists with head and tail pointers.***

Specifically, it would have been obvious to include Kirkman's techniques relating to two-dimensional doubly linked lists in the system of Khare. As discussed in claim 4, Khare recognizes the necessity of list structures and the proper ordering of requests to be processed and resolved across processors in the system. Ex-1002, 5:10-12 (“Requests may contain fields such as a packet type, destination ID, request type, source ID, transaction address, request length, stream ID, and **ordering semantics**.”). Khare explains that “switching agent 140 may have more or less pending request buffers.” Ex-1002, 4:28-29. For example, Khare teaches that switching agent 140 contains response pending buffer 142, first pending requests

buffer 145, second pending requests buffer 146, and third pending requests buffer 147. Ex-1002, 4:13-17. Pending requests buffers 145, 146, and 147 (“*list structures*”) can store requests that have been received by switching agent 140 but have not yet been processed thereby maintaining an order of requests. Ex-1002, 4:20-23; *see also* Ex-1002, 5:10-12 (“Requests may contain fields such as . . . ordering semantics.”). “Buffers 142, 145, 146, and 147 may be stored in a single memory device or in some combination of multiple memory devices in switching agent 140.” Ex-1002, 4:31-34; Ex-1006, ¶ 216.



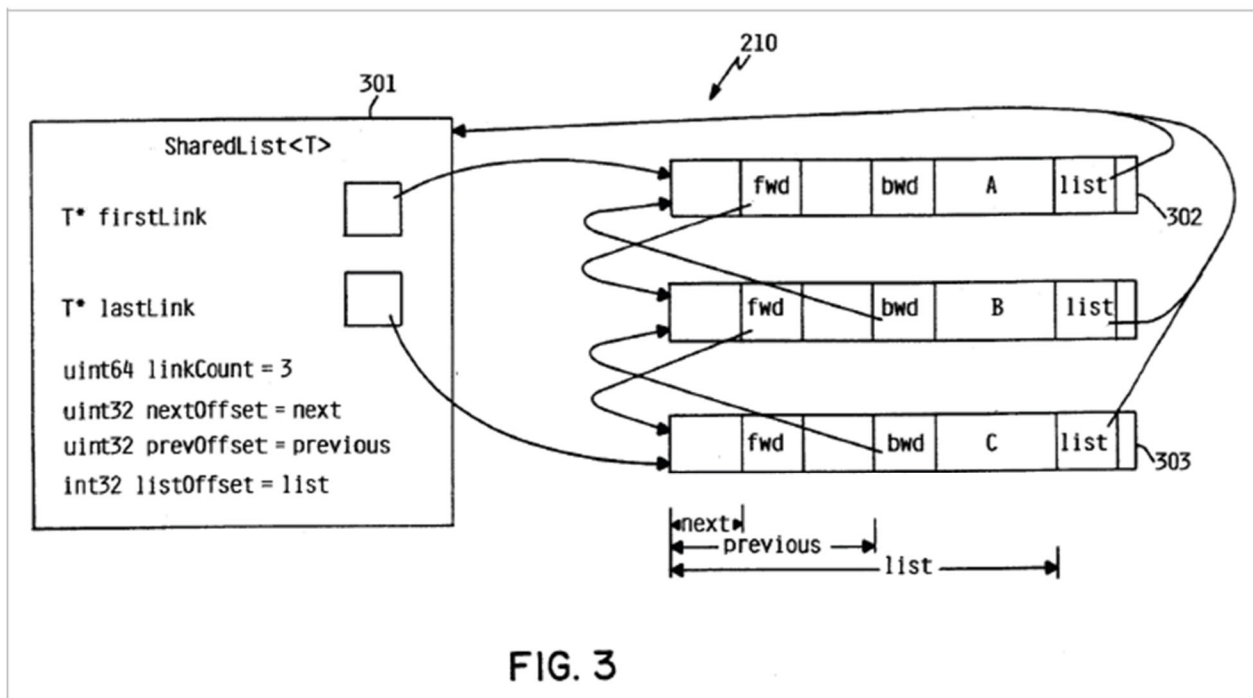
Ex-1002, Fig. 3.

As shown in Figure 3, Khare demonstrates cross-list dependencies through the pending requests buffers 145, 146, 147 and response pending buffer 142 containing entries that reference the associated memory address that corresponds to their respective memories. Ex-1002 6:17-8:65, 4:35-49, 4:66-5:13; Ex-1006, ¶¶ 248-49. Further, Khare teaches that entries in the buffer are removed from the buffer as part of the request-processing and administrative steps. Ex-1006, ¶ 250. A POSITA would have found it obvious to use doubly linked lists to accomplish this implementation. Ex-1006, ¶¶ 250-251. And a POSITA would have looked to Kirkman for further details on implementing a doubly-linked list. Ex-1006, ¶¶ 251-52.

Here, a POSITA would further recognize that doubly linked lists was an obvious and well-known structure for storing lists of information, as taught in Kirkman, and therefore, would be an obvious way of implementing Khare's request buffers. Ex-1006, ¶¶ 251-52. For example, Kirkman teaches that a common data structure for a POSITA at the time was the linked list and the preferred embodiment would have been the doubly-linked list. Ex-1004, 1:44-46, 49-53 ("Software designers are well aware of the need . . . to arrange data in a logical manner . . . Although almost any custom data structure may be created, several common types of data structures are used over and over again by software designers. One of the

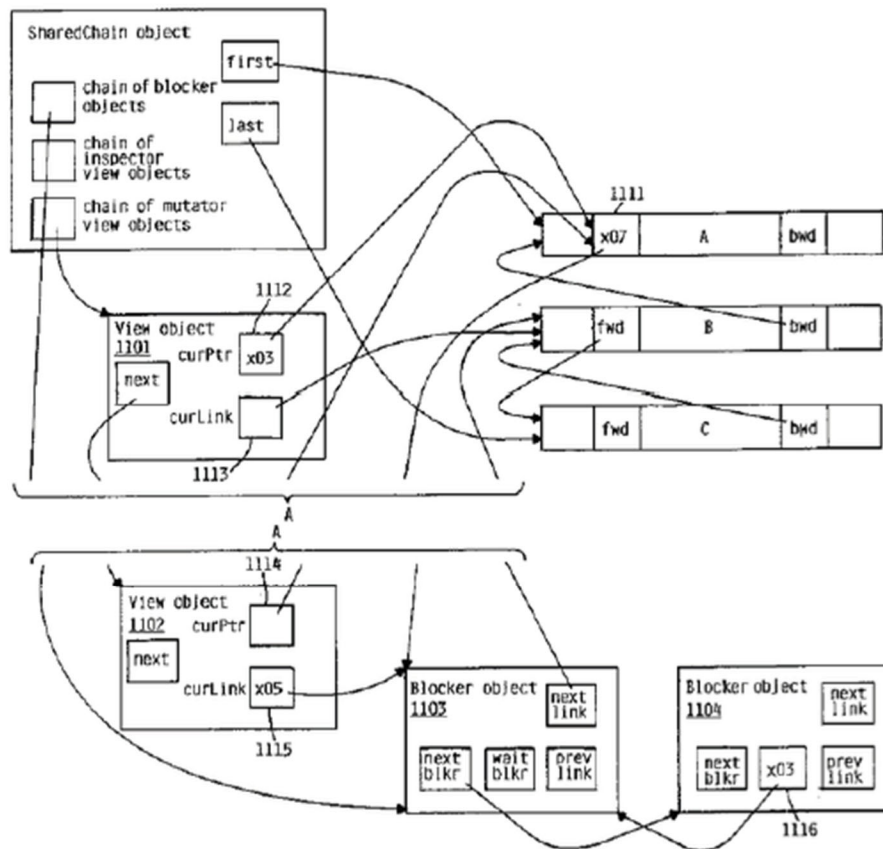
most ubiquitous is the linked list, which may be singly linked or doubly linked.”); 2:7-9.

The 409 Patent’s “*head and tail pointers*” are entries in a doubly-linked list that contain only the referential pointers without any associated data objects. *See* Ex-1001 at 12:4-17. Kirkman teaches that within an embodiment of the linked list “a single element . . . may be a dummy which contains nothing in addition to the reference. The reference is typically a pointer to the memory address of the next list element” Ex-1004, 1:56-64. Kirkman’s dummy element when at the front of a linked list teaches a “*head pointer*” and a “*tail pointer*” when at the back of a linked list. Ex-1006, ¶¶ 242-44.



Ex-1004, Fig. 3, 6:16-24 (“The SharedList<T> object 301 *contains pointers to the first client link object 302 and last client link object 303 in the list*, and a count of the number of links in the list. It also *contains offsets of the forward and backward pointers within each link.*”) (emphasis added); Ex-1006, ¶¶ 243-44.

Kirkman also teaches an orthogonal structure (“two-dimensional”) of doubly linked lists.



Ex-1004, Figs 11b, 11a, 44:25-45:39 (accompanying description of the figures in the specification); Ex-1006, ¶¶ 245-47. A POSITA would have found it obvious that Kirkman’s disclosure of intersecting linked lists that produce an orthogonal form

(“*two-dimensional*”) resemble the cross-list pointers described in Khare. Ex-1006, ¶ 252.

Accordingly, Khare in view of Kirkman would have rendered this limitation obvious to a POSITA. Ex-1006, ¶ 253.

2. Claim 6

- i. ***[6.0] The computer system, as set forth in claim 5, wherein the list structures comprise dependency links.***

As discussed in limitation [1.7], Khare teaches that if a pending request buffer entry is available to process the new request, the new request is stored in the pending request buffer. Ex-1002, 7:10-12.

And as discussed for claim 5, Khare demonstrates cross list dependencies (“*dependency links*”) between the pending requests buffers and the pending response buffers, which are the “*list structures*” of claim 5. Ex-1004, Fig. 3, 6:17-7:47; Ex-1006, ¶¶ 256-58.

Khare further demonstrates that the “*list structures comprise dependency links*” by teaching that “[e]ach entry in response pending buffer 142 may contain . . . an ID of associated request field 322,” meaning that each entry in the Response Pending Buffer can point to the corresponding entry in the appropriate Pending Requests Buffer that generated the pending response entry. Ex-1004 6:17-7:47; Ex-1006, ¶ 258. These cross-list dependencies are used by switching agent 140 to keep the lists consistent and the cache coherent. Ex-1006, ¶ 258; Ex-1004,

4:40-46 (“Processing of such requests may include identifying nodes that contain a copy in their private cache memory of information stored at a location in a memory address space that is associated with the request, sending requests that are based on the request being processed to each of said identified nodes, and storing an entry in the response pending buffer for each request sent.”); Ex-1004, 5:29-32 (“A snoop request may be generated when a processor needs other processors in the system to look in their own caches to see if a particular line is present in their cache.”); Ex-1004, 7:12-16 (“Once a Pending Request Buffer entry is allocated to the new request, the switching device 140 may check the response pending buffer to identify entries associated with the same location in the memory address space as the new request (404-405).”).

Accordingly, Khare teaches this limitation. Ex-1006, ¶ 259.

1. Claim 13

- i. ***[13.0] The coherency control module, as set forth in claim 12, wherein the list structures comprise two-dimensional doubly-linked lists with head and tail pointers.***

Khare in view of Kirkman teaches this limitation. See claim [5]; Ex-1006, ¶¶ 260-74.

2. Claim 14

- i. *[14.0] The coherency control module, as set forth in claim 13, wherein the list structures further comprise dependency links.*

Khare teaches this limitation. See claim [6]; Ex-1006, ¶¶ 275-80.

VIII. DISCRETIONARY DENIAL IS NOT APPROPRIATE

A. The Board Should Not Deny Institution Under 35 U.S.C. § 325(d)

Denial under § 325(d) is not warranted because the grounds advanced are neither cumulative nor redundant to the prosecution of the '409 Patent, as Khare and Kirkman were not cited or considered by the Examiner during prosecution of the '409 Patent. The references in this Petition were neither relied upon during prosecution nor are cumulative of art relied upon during prosecution. Further the '409 Patent has not been previously challenged in an IPR. Further, the Examiner did not have the benefit of Dr. Jacob's detailed expert testimony and the other evidence of record.

Thus, the Board should not exercise discretion to deny institution.

B. The Board Should Not Deny Institution Under 35 U.S.C. § 314(a)

The merits of Petitioner's arguments are compelling and the evidence in support is substantial. That "alone demonstrates that the PTAB should not discretionarily deny institution under *Fintiv*." Ex-1005 at 4–5. But in any event, the six *Fintiv* factors do not justify denying institution.

Factor 1 (stay) is **neutral**. SAP has not yet moved to stay pending this IPR.

Factor 2 (proximity to trial) is **neutral or weighs against denial**. In the originally filed litigation by Patent Owner Valtrus (*Valtrus I*), the trial date is not currently set (although jury selection is set for September 22, 2025). Ex-1009, 1. However, the Board should not rely on the current schedule in *Valtrus I* because Patent Owner Valtrus filed suit alone (without the other named Patent Owner, Key Patent Innovations) and thus did not have Article III Standing. Ex-1010; Ex-1011. Petitioner filed a Motion to Dismiss for Lack of Standing in *Valtrus I*, reconsideration of which remains pending and casts serious doubt on whether *Valtrus I* will actually proceed along the current schedule. Indeed, in response to Petitioner's Motion to Dismiss for Lack of Standing, Patent Owners essentially concede that Valtrus alone lacked standing because Patent Owners together filed a second, duplicative lawsuit (*Valtrus II*) in an effort to rectify the Article III standing deficiencies in *Valtrus I*. Ex-1012; Ex-1013. This conduct confirms that Patent Owners themselves doubt that the trial schedule in *Valtrus I* will hold, further supporting that the Board should not rely on that schedule at all. Additionally, the other pending litigations involving the challenged patent (*Valtrus II* and a declaratory judgment action filed by Petitioner) do not have any trial date currently set, meaning there is no reliable trial date for the Board to consider in relation to this Petition.

Even if the Board assumes the *Valtrus I* trial date applies, this factor is neutral because there will be other claims not litigated at trial that are covered by this

Petition. Therefore, a final written decision will issue first at least on the claims not litigated at trial but covered by this Petition.

Factor 3 (investment in parallel proceedings) **weighs against denial**. Importantly, claims 3, 6, 11, and 14 are not at issue in the related litigations. *Samsung Bioepis Co. Ltd v. Regeneron Pharmaceuticals, Inc.*, IPR2023-00739, Paper 9 at 57 (Oct. 20, 2023) (“[B]ecause certain of the challenged claims, . . . are not expressly at issue in the district court litigation, *Fintiv* factor 3 favors institution.”). The related litigations are in their early stages of discovery, motions to transfer and dismiss have been filed, and the investment by the parties has been relatively minimal. All litigations involving the challenged patent are at an early stage, with major deadlines several months away. Ex-1009. Patent Owners may argue that the parties have invested substantial resources on procedural motions, but the resources expended have been directed toward resolving procedural missteps perpetuated by Patent Owner Valtrus’s choice to file alone (without Article III standing) and Patent Owners’ later attempts to fix that with procedurally improper duplicative litigation. These procedural disputes (caused by Patent Owners’ actions) should not be held against Petitioner under this factor.

Factor 4 (overlap) **weighs against denial** because Petitioner hereby stipulates that, if the IPR is instituted, it will not pursue the IPR grounds in the district court. Indeed, is Kirkman are not at issue in the District Court Litigation. Ex-1003. Thus,

“[i]nstituting trial here serves overall system efficiency and integrity goals by not duplicating efforts and by resolving materially different patentability issues.” *Apple, Inc. v. SEVEN Networks, LLC*, IPR2020-00156, Paper 10 at 19 (June 15, 2020); *see also Sand Revolution II, LLC v. Continental Intermodal Group-Trucking LLC*, IPR2019-01393, Paper 24 at 12 (June 16, 2020). In addition, Petitioner challenge more claims than Patent Owners will ultimately assert at trial. Patent Owner Valtrus agreed to narrow to 30 total asserted claims by April 30, 2025, and a maximum of six asserted claims per asserted patent. Ex-1009, 1. Thus, Patent Owners will be limited to asserting up to six of the nine asserted claims of the ’409 Patent, merely a subset of the claims challenged in this Petition. *See, e.g., Apple*, No. IPR2020-00156, Paper 10, at 17, 21-22.

Factor 6 (other circumstances) **weigh against denial**. Here, the merits of the Petition are particularly compelling—for example, Khare and Kirkman are strikingly similar disclosures to that of the ’409 Patent, are directed at the same problem and proposes the same solutions as the ’409 Patent, and teach nearly identical architecture, including several nearly identical figures. The evidence presented, if unrebutted at trial, would plainly lead to a conclusion that one or more claims are unpatentable by a preponderance of the evidence.

Although Factor 5 (parties) may weigh slightly in favor of denial because the parties are the same as in district court, under the “holistic view” of the remaining

factors and whether integrity of the system and efficiency is best served, institution is favored. *Samsung Elecs. Co. Ltd. v. Dynamics Inc.*, IPR2020-00505, Paper 11 at 15 (Aug. 12, 2020).

IX. CONCLUSION

Claims 1-17 are unpatentable for the reasons demonstrated above.

Dated: January 14, 2025

Respectfully submitted,

/ Louis L. Campbell /

Louis L. Campbell (Lead Counsel)

llcampbell@winston.com

USPTO Reg. No. 59,963

Winston & Strawn LLP

255 Shoreline Dr., Ste. 520

Redwood City, CA 94065

T: (650) 858-6500, F: (650) 858-6550

Lead Counsel for Petitioner

CERTIFICATE OF COMPLIANCE

This Petition complies with the word count limits set forth in 37 C.F.R. § 42.24(a)(1)(i), because this Petition contains 11,775 words, excluding the parts of the Petition exempted by 37 C.F.R. § 42.24(a)(1), determined using the word count provided by Microsoft Word, which was used to prepare this Petition.

Dated: January 14, 2025

Respectfully submitted,

/ Louis L. Campbell /

Louis L. Campbell (Lead Counsel)

llcampbell@winston.com

USPTO Reg. No. 59,963

Winston & Strawn LLP

255 Shoreline Dr., Ste. 520

Redwood City, CA 94065

T: (650) 858-6500, F: (650) 858-6550

Lead Counsel for Petitioner

CERTIFICATE OF SERVICE

Under 37 C.F.R. §§ 42.6(e) and 42.105(a), this is to certify that on January 14, 2025, I caused to be served a true and correct copy of the foregoing “**PETITION *INTER PARTES* REVIEW OF U.S. PATENT NO. 6,823,409**” and accompanying exhibits by Federal Express priority overnight mail on the following:

Botos Churchill IP Law LLP
430 Mountain Avenue
Suite 401
New Providence, NJ 07974

With a courtesy copy of this Petition and supporting material was also served on litigation counsel for Patent Owners via email:

Matt Berkowitz
mberkowitz@reichmanjorgensen.com

Patrick R. Colsher
pcolsher@reichmanjorgensen.com

Aaron Morris
amorris@reichmanjorgensen.com

Navid Cyrus Bayar
nbayar@reichmanjorgensen.com

Reichman Jorgensen Lehman & Feldberg LLP
100 Marine Parkway
Suite 300
Redwood Shores, CA 94065

Khue Van Hoang
khoang@reichmanjorgensen.com
Reichman Jorgensen Lehman & Feldberg LLP
400 Madison Avenue
Suite 14d
New York, NY 1001

Philip Ekle

peklem@reichmanjorgensen.com

Savannah Harman Carnes

scarnes@reichmanjorgensen.com

Reichman Jorgensen Lehman & Feldberg LLP

1909 K Street, NW

Suite 800

Washington, DC 20006

/ Louis L. Campbell /

Louis L. Campbell (Lead Counsel)

llcampbell@winston.com

USPTO Reg. No. 59,963

Winston & Strawn LLP

255 Shoreline Dr., Ste. 520

Redwood City, CA 94065

T: (650) 858-6500, F: (650) 858-6550

Lead Counsel for Petitioner