

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

PALO ALTO NETWORKS, INC.

Petitioner,

v.

QOMPLX LLC,

Patent Owner.

Inter Partes Review No. 2026-00432

U.S. Patent No. 12,143,424

**PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 12,143,424**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. MANDATORY NOTICES AND FEES	1
A. Real Party In Interest (37 C.F.R. § 42.8(b)(1))	1
B. Related Matters (37 C.F.R. § 42.8(b)(2)).....	1
1. Related Patent Office Proceedings.....	1
2. Related Litigation.....	2
C. Lead and Backup Counsel and Service Information (37 C.F.R. §§ 42.8(b)(3)-(4))	2
D. Payment of Fees (37 C.F.R. § 42.15(a))	3
E. Requirements For <i>Inter Partes</i> Review (37 C.F.R. §§ 42.101(A)- (C), 42.104(A), AND 42.108)	4
III. GROUNDS	4
IV. PERSON OF ORDINARY SKILL IN THE ART	4
V. BACKGROUND	5
A. '424 patent	5
1. Specification.....	5
2. Prosecution History.....	6
VI. CLAIM CONSTRUCTION	6
VII. GROUNDS	11
A. Ground 1: Barsness	11
1. Summary of How Barsness Meets the Claims	12
2. Claim 1	16
(a) [1.1].....	16
(b) [1.2].....	18
(c) [1.2.a].....	23
(d) [1.2.b].....	27
(e) [1.2.c].....	33

(f)	[1.3].....	36
(g)	[1.3.a].....	36
(h)	[1.3.b].....	39
(i)	[1.3.c].....	43
(j)	[1.3.d].....	46
(k)	[1.4].....	47
(l)	[1.4.a].....	48
(m)	[1.4.b].....	49
(n)	[1.5].....	50
(o)	[1.6].....	52
(p)	[1.7].....	53
3.	Claim 2.....	61
4.	Claim 3.....	62
5.	Claim 4.....	67
(a)	[4.1].....	67
(b)	[4.2].....	68
(c)	[4.2.a].....	70
(d)	[4.3].....	71
6.	Claim 5.....	73
7.	Claim 6.....	75
8.	Claim 8.....	75
9.	Claim 9.....	78
(a)	[9.1].....	78
(b)	[9.2].....	78
(c)	[9.3].....	78
(d)	[9.4].....	81
(e)	[9.5].....	81
(f)	[9.6].....	81
(g)	[9.7].....	81

(h) [9.8].....	81
10. Claim 10.....	82
11. Claim 11	82
12. Claim 12.....	82
B. Ground 2: Claims 4, 6, 8, 12 Are Obvious Over Barsness in View of Chakradhar	82
1. Chakradhar (EX1005).....	82
2. Combining Barsness And Chakradhar.....	84
3. Claims 4, 6, 8, 12	87
C. Ground 3: Claims 4, 6, 8, and 12 Are Obvious Over Barsness in View of Siripurapu	90
1. Siripurapu (EX1007).....	90
2. Combining Barsness And Siripurapu	94
3. Claims 4, 6, 8, 12	96
D. Ground 4: Claims 4, 6, 8, and 12 Are Obvious Over Barsness in View of Chakradhar and Siripurapu	97
VIII. NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS.....	98
IX. CONCLUSION.....	98

EXHIBIT LIST

Ex.	Description
1001	U.S. Patent No. 12,143,424 (“the ’424 patent”)
1002	Prosecution History for U.S. Patent No. 12,143,424
1003	Declaration of Dr. Paul Min
1004	U.S. Patent No. 8,732,300 (“Barsness”)
1005	U.S. Patent Application Pub. No. 2016/0269247 A1 (“Chakradhar”)
1006	U.S. Provisional Application No. 62/132,686 (“Chakradhar Provisional”)
1007	U.S. Patent Application Pub. No. 2014/0059185 A1 (“Siripurapu”)
1008	Thomas Heinze, et al., Cloud-based Data Stream Processing, Proc. of the 8th ACM Int’l Conf. on Distributed Event-Based Sys., 238- 245 (May 2014), https://di.acm.org/doi/10.1145/2611286.2611309 (“Heinze”)

1009	Ashkon Soroudi, et al., High Availability on a Distributed Real Time Processing System, Electrical Engineering and Computer Sciences, University of California at Berkeley (May 15, 2015), http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-135.html (“Soroudi”)
1010	Naresh Kumar Reddy Rapolu, Dynamic Re-Optimization Techniques for Stream Processing Engines and Object Stores (2015) (Ph.D. dissertation, Purdue University) (“Rapolu”)
1011	Ankit Toshniwal, et al., Storm @Twitter, SIGMOD ’14: Proc. of the 2014 ACM SIGMOD Int’l Conf. on Mgmt. of Data, 147-156 (2014), http://dx.doi.org/10.1145/2588555.2595641 (“Toshniwal”)
1012	Leonardo Neumeyer, et al., S4: Distributed Stream Computing Platform, IEEE Int’l Conf. on Data Mining Workshops, 170-177 (2010), https://ieeexplore.ieee.org/document/5693297 (“Neumeyer”)
1013	Michael Duller, et al., Virtualizing Stream Processing, Middleware (2011) (“Duller”)

1014	Scott Schneider, et al., Tutorial: Stream Processing Optimizations, DEBS '13 (2013) (“Schneider”)
1015	U.S. Patent No. 8,407,360 (“Andrade”)
1016	U.S. Patent No. 8,560,527 (“Santosuosso”)
1017	U.S. Patent No. 8,190,960 (“Bahadur”)
1018	<i>Curriculum Vitae</i> of Dr. Paul Min
1019	Plaintiff Qomplx LLC’s Responsive Claim Construction Brief, (Dkt. 57), <i>Qomplx LLC v. Microsoft Corporation</i> , Case No. 1:25-cv-01383-ADA
1020	Declaration of Mark Crovella, Ex. 1003, IPR-2026-00298 (July 21, 2024)

LIST OF CHALLENGED CLAIMS

Claim	Limitation
[1.1]	A distributed computing cluster comprising:
[1.2]	a first plurality of computer systems,
[1.2.a]	wherein each respective computer system of the first plurality of computer systems comprises a memory that stores a respective first data,
[1.2.b]	wherein the respective first data represents a respective portion of a distributed computational graph,
[1.2.c]	and wherein the distributed computational graph describes a flow of output data of a first transformation pipeline to an input of a second transformation pipeline,
[1.3]	wherein a first computer system of the first plurality of computer systems is configured to:
[1.3.a]	receive a first stream of input data from a first input feed,
[1.3.b]	process the first stream of input data substantially in real time by executing software instructions that apply the first

Claim	Limitation
	transformation pipeline to the first stream of input data to generate first pipeline output messages,
[1.3.c]	process the respective first data stored in the memory of the first computer system to determine information about the second transformation pipeline,
[1.3.d]	and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined information,
[1.4]	wherein the second computer system is configured to:
[1.4.a]	receive the first pipeline output messages,
[1.4.b]	and process the first pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the first pipeline output messages to generate second pipeline output messages,
[1.5]	wherein the first and second computer systems are distinct; and
[1.6]	a second plurality of computer systems;

Claim	Limitation
[1.7]	wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause a fourth computer system of the second plurality of computer systems to execute software instructions that apply at least one of the first transformation pipeline and the second transformation pipeline.
[2]	The distributed computing cluster of claim 1, wherein at least one of the first transformation pipeline and the second transformation pipeline is non-linear.
[3]	The distributed computing cluster of claim 2, wherein the third computer system of the first plurality of computer systems is configured to: identify a fifth computer system of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the first transformation pipeline to any stream of input

Claim	Limitation
	data, and cause the fifth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data.
[4.1]	The distributed computing cluster of claim 3, wherein the third computer system is further configured to execute software instructions that:
[4.2]	identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph,
[4.2.a]	wherein the respective first data includes information about the first transformation pipeline,
[4.3]	and store in the identified one or more memories information identifying the fifth computer system.

Claim	Limitation
[5]	The distributed computing cluster of claim 2, wherein the third computer system of the first plurality of computer systems is configured to: identify a fifth computer system of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the second transformation pipeline to any pipeline output message, and cause the fifth computer system to process third pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the third pipeline output messages.
[6.1]	The distributed computing cluster of claim 5, wherein the third computer system is further configured to execute software instructions that:

Claim	Limitation
[6.2]	identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph,
[6.2.a]	wherein the respective first data includes information about the second transformation pipeline,
[6.3]	and store in the identified one or more memories information identifying the fifth computer system.
8	The distributed computing cluster of claim 1, wherein the first computer system is further configured to transmit the first pipeline output messages to another computer system of the second plurality of computer systems.
[9.1]	A distributed computing cluster comprising:
[9.2]	a first plurality of computer systems,

Claim	Limitation
[9.3]	wherein each respective computer system of the first plurality of computer systems comprises a respective memory that stores a respective first data that: describes at least a respective portion of a configuration of a first transformation pipeline, and identifies at least a second transformation pipeline,
[9.4]	wherein a first computer system of the first plurality of computer systems is configured to: receive a first stream of input data from a first input feed, process the first stream of input data substantially in real time by executing software instructions that apply the first transformation pipeline to the first stream of input data to generate first pipeline output messages, process the respective first data to determine an identification of the second transformation pipeline,

Claim	Limitation
	and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined identification,
[9.5]	wherein the second computer system is configured to: receive the first pipeline output messages, and process the first pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the first pipeline output messages,
[9.6]	wherein the first and second computer systems are distinct; and
[9.7]	a second plurality of computer systems;

Claim	Limitation
[9.8]	wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause the second plurality of computer systems to execute at least one of the first transformation pipeline and the second transformation pipeline.
[10]	The distributed computing cluster of claim 9, wherein one or more of the first transformation pipeline and the second transformation pipeline is non-linear.
[11]	The distributed computing cluster of claim 10, wherein the third computer system of the first plurality of computer systems is configured to: identify a fourth computer system of the first plurality of computer systems, wherein the fourth computer system is not

Claim	Limitation
	configured to apply the first transformation pipeline to any stream of input data, and cause the fourth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data.
[12]	The distributed computing cluster of claim 11, wherein the third computer system is further configured to execute software instructions that: identify one or more of the respective memories that stores respective data that describes a respective portion of a configuration of the first transformation pipeline, and store in the identified one or more memories information identifying the fourth computer system.

I. INTRODUCTION

The challenged claims relate to processing data streams using a distributed computing cluster with a plurality of computer systems that apply transformations to one or more streams of data. But well before the earliest alleged priority date, these claimed concepts were used and common in the art. For example, prior art Barsness (EX1004), which was not of record during prosecution, discloses a distributed computing architecture for processing stream data using transformations carried out by processing elements distributed among different compute nodes. It also describes scaling this architecture, e.g., by duplicating nodes, to ensure proper processing speeds.

Barsness teaches dynamically duplicating processing elements from one compute node to an additional compute node, to maintain desired processing capacity. In contrast, the '424 patent describes only generally, without detail, that “actions can be implemented to resolve the issue” of slow processing. EX1001, 16:52-56.

II. MANDATORY NOTICES AND FEES

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

The real party in interest for Petitioner is Palo Alto Networks, Inc.

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

1. Related Patent Office Proceedings

On March 18, 2026, Microsoft Corporation filed a petition for *inter partes* review (IPR2026-00298) challenging the '424 patent's claims 1-3, 5, 7, 9-11, and 13. On May 15, 2026, Petitioner filed a petition for *inter partes* review (IPR2026-00363) challenging the same claims of the '424 patent, which is substantively identical to Microsoft's petition in IPR2026-00298, and a motion for joinder.

The '424 patent is in the same family as U.S. Patent No. 12,143,425 ('425 patent). It shares a common specification with the '425 patent. On May 26, 2026 Petitioner file a petition for inter partes review (IPR2026-00371) challenging all claims of the '425 patent.

2. Related Litigation

Patent Owner is currently asserting the '424 patent against Petitioner in *Qomplx LLC v. Palo Alto Networks, Inc.*, Case No. 2-25-cv-00913 (E.D. Tex.), filed August 28, 2025. Patent Owner has also asserted the '424 patent against Microsoft in *Qomplx LLC v. Microsoft Corporation*, Case No. 1-25-cv-01383 (W.D. Tex.), filed August 28, 2025.

C. Lead and Backup Counsel and Service Information (37 C.F.R. §§ 42.8(b)(3)-(4))

Petitioner provides the following counsel and service information. Petitioner consents to electronic service the email addresses listed in the table below. Pursuant to 37 C.F.R. § 42.10(b), Powers of Attorney accompany this Petition.

<u>Lead Counsel</u>	<u>Back-Up Counsel</u>
James M. Glass (Reg. No. 46,729) jimglass@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 295 5 th Avenue, 9th Floor New York, NY 10016 Tel: (212) 849-7000 Fax: (212) 849-7100	Quincy Lu (Reg. No. 76,954) quincylu@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 1109 First Avenue, Suite 210 Seattle, WA 98101 Tel: (206) 905-7000 Fax: (206) 905-7100 Brian Biddinger (<i>pro hac vice</i> forthcoming) brianbiddinger@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 295 5 th Avenue, 9 th Floor New York, NY 10016 Tel: (212) 849-7000 Fax: (212) 849-7100 Jeremy Thomas (<i>pro hac vice</i> forthcoming) jeremythomas@quinnemanuel.com QUINN EMANUEL URQUHART & SULLIVAN, LLP 50 California Street, 22 nd Floor San Francisco, California 94111 Tel: (415) 875-6600 Fax: (415) 875-6700

D. Payment of Fees (37 C.F.R. § 42.15(a))

The undersigned authorizes the Office to charge the fee required for this
Petition (and any additional fees) to Deposit Account No. 50-5708.

E. Requirements For *Inter Partes* Review (37 C.F.R. §§ 42.101(A)-(C), 42.104(A), AND 42.108)

Petitioner certifies that the '424 patent is available for *inter partes* review, that Petitioner is not barred or estopped from challenging the '424 claims on the grounds identified herein, and that the prohibitions of 35 U.S.C. §§ 315(a)-(b) are inapplicable.

III. GROUNDS

Ground	Basis	Reference(s)	Claims
1	§103	Barsness	4, 6, 8, 12
2	§103	Barsness and Chakradhar	4, 6, 8, 12
3	§103	Barsness and Siripurapu	4, 6, 8, 12
4	§103	Barsness, Chakradhar, and Siripurapu	4, 6, 8, 12

IV. PERSON OF ORDINARY SKILL IN THE ART

The person of ordinary skill in the art (“POSITA”) would have had a bachelor’s degree in computer science, computer engineering, electrical engineering, or an equivalent degree, with two to four years’ experience designing, implementing, or otherwise working with distributed computing systems. More education would compensate for less work experience and vice versa. EX1003, ¶¶26-28.

A POSITA would have been familiar with common, then-existing technologies for scalable, distributed, computing and stream processing systems, as

well as issues surrounding the design, implementation, and maintenance of such technologies. EX1003, ¶29.

Petitioner's expert, Dr. Paul Min, Ph.D., exceeded this level by the priority date.

V. BACKGROUND

A. '424 patent

1. Specification

The '424 patent describes the “analysis of very large data sets using distributed computational graph tools which allow for transformation of data through both linear and non-linear transformation pipelines.” EX1001, 3:32-35. Among other things, it describes determining when aspects of the streaming data analysis “are processing slowly or may have hung ... so that actions can be implemented to resolve the issue.” EX1001, 16:41-56. The '424 patent does not, however, further describe these issue-resolving “actions” or how to implement them. EX1003, ¶37.

The '424 patent provides examples of prior art transformation pipelines (FIGS. 6, 12) and other transformation pipelines (FIGS. 7-9, 13-15). EX1001, 17:11-19:18, 21:25-22:51 (describing transformation pipelines). However, the '424 patent provides no disclosure of how to implement these transformation pipelines using the general hardware architecture shown in FIG. 3, with “any number of clients and/or servers.” EX1001, 13:50-62. EX1003, ¶38.

2. Prosecution History

The first Office action granted a request for prioritized examination. EX1002, 424/539. The applicant and Examiner agreed to enter an Examiner-proposed amendment “to add the substance of claim 9 into the independent claims,” which recited the “third computer system” to cause a “fourth computer system” to execute some of the pipeline instructions. EX1002, 433/539, 435/539, 438/539, EX1002, 517-526/539.

VI. CLAIM CONSTRUCTION

Unless otherwise noted below, Petitioner applies the plain and ordinary meaning of all claim terms as per *Phillips*.

Petitioner proposes the same constructions for common claim terms proposed in IPR2026-00298, IPR2026-00363, IPR2026-00371 and the district court litigation against Microsoft. Patent Owner has argued in the district court that these terms should be given their plain meaning. However, Patent Owner’s constructions are broader and inclusive of Petitioner’s constructions. Accordingly, the prior art meets the limitations under either Petitioner’s construction or Patent Owner’s construction, and it is not necessary for the Board to resolve the parties’ dispute.

<i>“distributed computational graph”</i>	“A directed graph with each transformation comprising a node and
---	--

	the output messages between transformations comprising edges.”
--	---

Under its “Definitions,” the ’424 patent explains: “In the *distributed computational graph* system, transformations within transformation pipeline are represented as directed graph with each transformation comprising a node and the output messages between transformations comprising edges.” EX1001, 9:31-35 (emphasis added). The specification does not elsewhere describe a “*distributed computational graph*.” See, e.g., *id.*, 9:35-50. This passage’s reference to “transformation pipeline[s]” matches how “*distributed computational graph*” is used in the claims—as something that describes “a flow of data from the first transformation pipeline to an input of the second transformation pipeline.” See, e.g., claim 1.

This is consistent with, for example, Figure 7 (reproduced below), which depicts a “transformation pipeline 700” of a “distributed computational graph.” EX1001, 17:52-53. Each transformation corresponds to a node, and each directed edge corresponds to an output message. *Id.*, 17:53-18:32. The same is true of Figures 8-9 and 13-15. EX1001, 18:37-19:13, 19:14-67, 21:42-22:51.

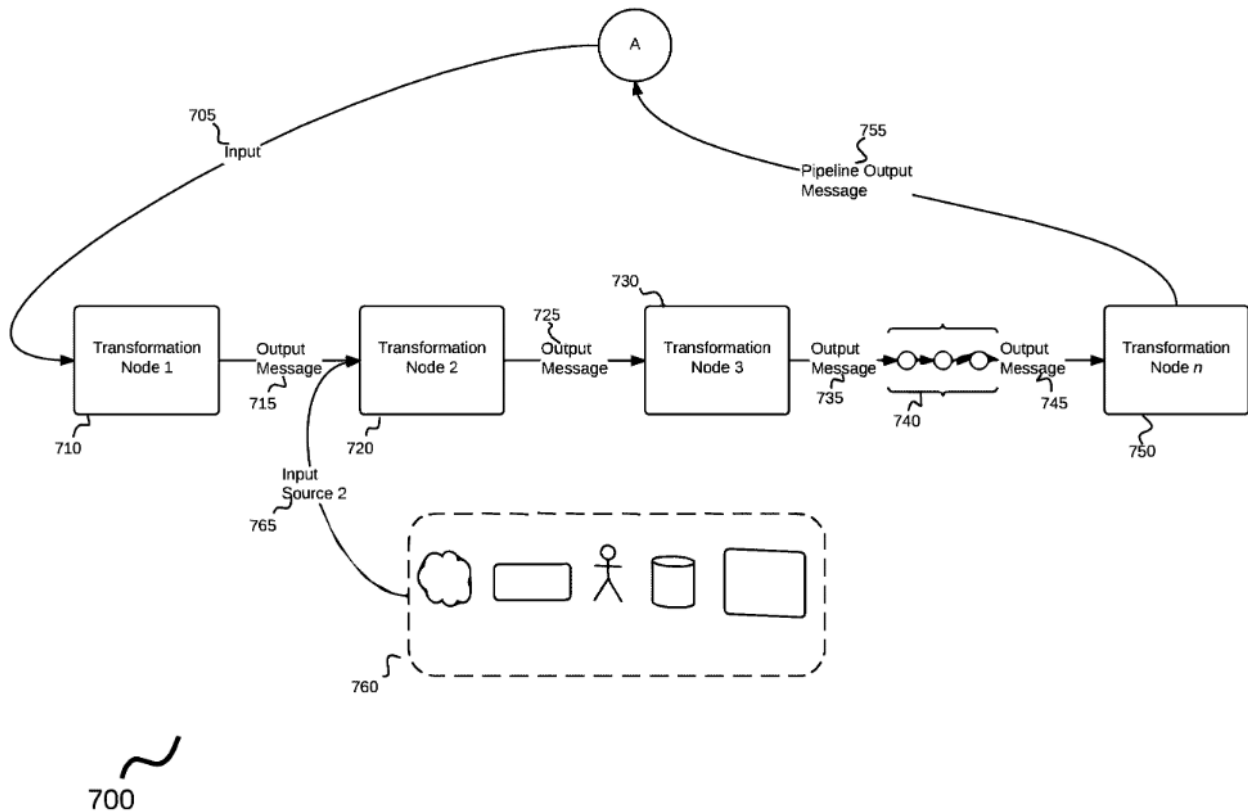


Fig. 7

In the district court, Patent Owner proposes “[p]lain and ordinary meaning.” EX1019, 16-20. Patent Owner agrees that the term refers to a graph constituting nodes and edges, that “describes a flow of output data of a first transformation pipeline to an input of a second transformation pipeline,” but has no “requirement[] about what precisely the graph’s nodes and edges represent.” *Id.*, 16. Patent Owner’s construction is broader and inclusive of Petitioner’s construction. Therefore, the Board need not resolve the parties’ dispute for this term.

<i>“transformation”</i>	“A function performed on zero or more streams of input data which results in a single stream of output.”
--------------------------------	--

The ’424 patent’s “Definitions” section expressly defines ***“transformation,”*** stating that ***“[a]s used herein,”*** a transformation “is a function performed on zero or more streams of input data which results in a single stream of output.” EX1001, 9:51-54; *Thorner*, 669 F.3d 1362, 1365-66 (Fed. Cir. 2012).

In district court, Patent Owner argues that Petitioner’s construction *is* the plain meaning of the term. EX1019, 22-23. Thus, there is no dispute as to the term’s meaning, whether because of plain meaning or lexicography. Accordingly, this petition applies Petitioner’s construction, and the Board need not resolve the parties’ dispute for this term.

<i>“transformation pipeline”</i> <i>“pipeline”¹</i>	/ “A progression of functions which each perform a transformation on a data stream.”
--	--

The specification’s “Discussion of the State of the Art” describes “data pipelines” as “a progression of functions which each perform some action or

¹ The term “pipeline” always has an antecedent basis to a “transformation pipeline.”

transformation on a data stream.” EX1001, 4:31-34. Although this passage refers to “data pipelines” rather than “transformation pipelines,” because the passage describes the pipeline performing a progression of “some action or *transformation* on a data stream.” Thus, a “***transformation pipeline***” is plainly a “data pipeline” that performs transformations. Further supporting Petitioner’s construction, the specification’s “Definition” section explains: “In the invention, transformations are ***placed into pipelines***.” EX1001, 9:62-67; *see also e.g., id.*, 9:67-10:7, 17:52-66.

In district court, Patent Owner proposes plain meaning without construction. EX1019, 20-22. Patent Owner argues that “[t]he specification is replete with examples of transformation pipelines.” EX1019, 21 (citing to the ’424 patent’s equivalent of EX1001, 17:11-19:67, 21:25-22:51, Figs. 6-9, 12-15). All of these “examples of transformation pipelines” meet Petitioner’s construction. For example, the specification describes Figure 7’s “transformation pipeline” as having a “streaming input from a data filter” (*i.e., streaming data*) that inputs into a “first transformation node.” EX1001, 17:58-60. The first “[t]ransformation node’s ***function*** is performed on input data stream and ***transformed*** output message 715 is sent to transformation node 2.” *Id.*, 17:56-58. Thus, in this example as well as the other examples Patent Owner provides, the transformation pipeline is a “progression of functions which each perform a transformation on a data stream.”

Accordingly, Patent Owner's construction is broader than and inclusive of Petitioner's construction. Petitioner will therefore apply Petitioner's construction to the analysis below. Because the prior art meets the limitation under Petitioner's narrower construction, it also meets the limitation under Patent Owner's broader construction. The Board need not resolve the parties' dispute for this term.

VII. GROUNDS

A. Ground 1: Barsness

The claims of the '424 patent are directed to transforming a data stream using a pipeline of computers. The transformation is performed by a *first plurality of computer systems*. The *first computer system* transforms the data stream using a *transformation pipeline*, then transmits the transformed data to a *second computer system* which further transforms the data stream using a different *transformation pipeline*. As discussed above, the parties agree that a "transformation pipeline" includes a progression of functions that each transform the data.

A *third computer system* then causes a *fourth computer system* (of a *second plurality of computer systems*) to also perform some of the transformation processing. Additionally, the *third computer system* identifies a *fifth computer system* (of the *first plurality of computer systems*) and causes it to also perform some of the transformation processing.

To summarize the various claimed computer systems:

- *First* and *second computer systems* (of the *first plurality*): performs transformation using *transformation pipelines*,
- *Third computer system* (of the *first plurality*): causes the *fourth computer system* (of the *second plurality*) to perform one of the *transformations pipelines* performed by the *first* or *second computer systems*, and
- *Third computer system* (of the *first plurality*) causes the *fifth computer system* (of the *first plurality*) to perform one of the *transformation pipelines* performed by the *first* or *second computer systems*

1. Summary of How Barsness Meets the Claims

Barsness is a U.S. patent that issued May 20, 2014, from an application filed January 10, 2011. EX1004, 1; EX1003, ¶42. Barsness qualifies as prior art under Sections 102(a)(1) and 102(a)(2).

Barsness discloses an architecture of hardware and software for streaming applications that process streaming data through pipelines of transformations. Barsness also discloses systems for managing and monitoring the compute nodes that perform the transformations. Specifically, Barsness discloses the transformation of streaming data using multiple compute nodes, each of which has a pipeline of processing elements (“PEs”). Each PE applies a transformation to incoming data and transmits the output data to downstream PEs in accordance with an operator graph.

EX1004, 3:16-22 (“In a stream data processing application, operators are connected to one another such that data flows from one processing element (PE) to the next (e.g., over a TCP/IP socket).”); EX1003, ¶¶43-46. As shown in the annotated Figure 1B below, each compute node is a *computer system* and the PEs form a *transformation pipeline*.

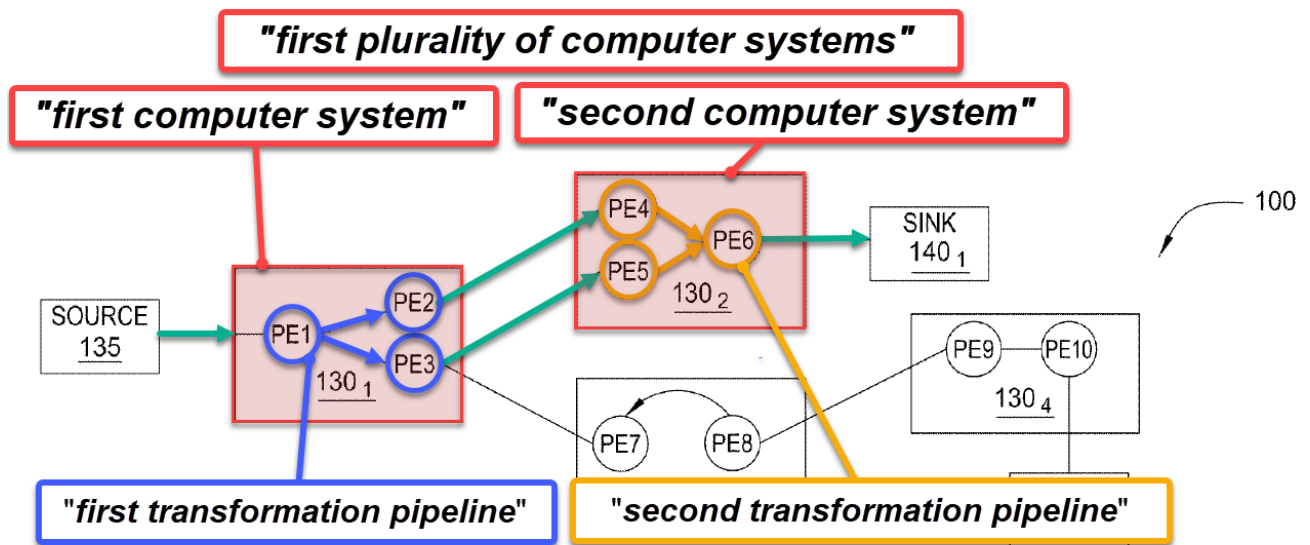
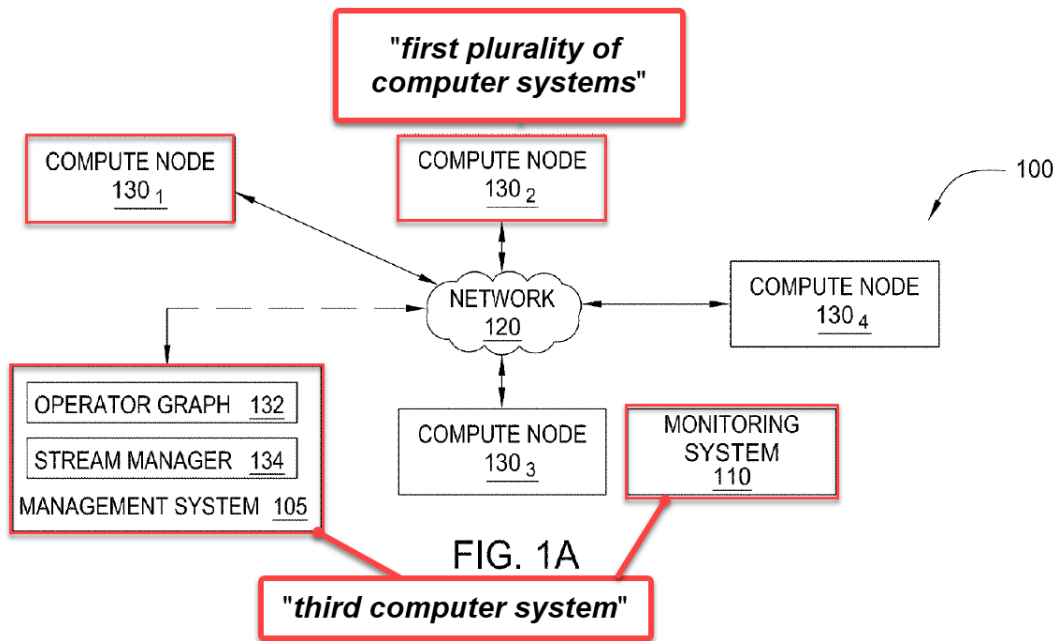


FIG. 1B

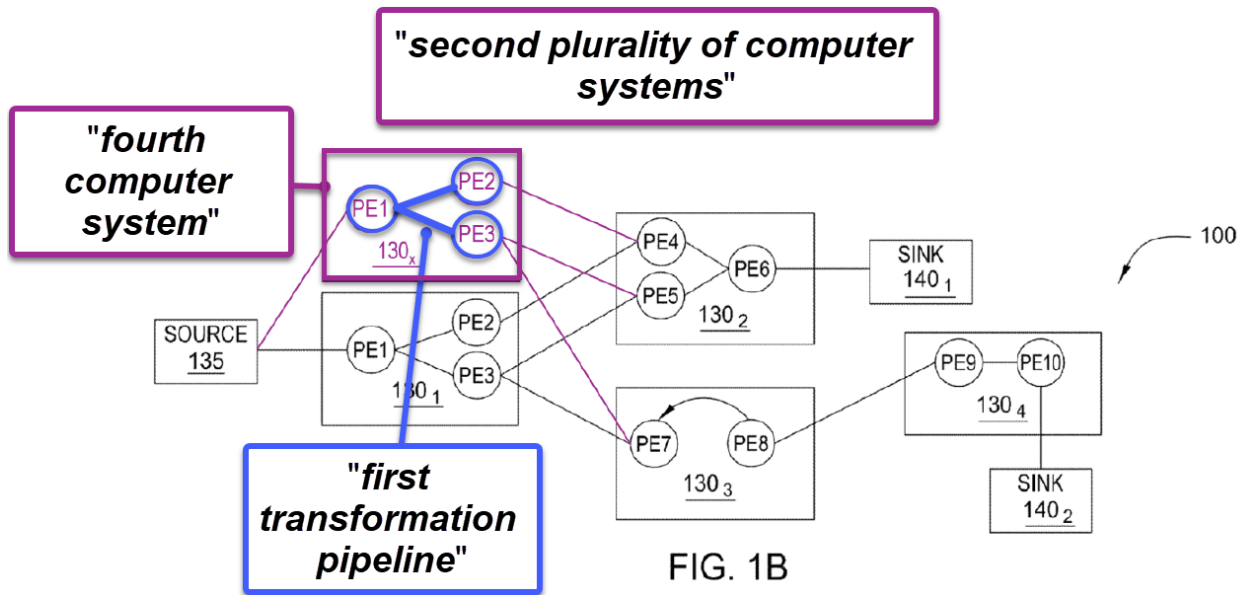
EX1004, FIG. 1B (annotations added).

Barsness discloses computer systems for managing and monitoring the compute nodes of the streaming application. Barsness discloses that “management system 105” and “monitoring system 110” each store and maintain the operator graph in memory. EX1004, 3:45-50, 10:12-14, 10:49-51; 11:26-27, 11:53-55.

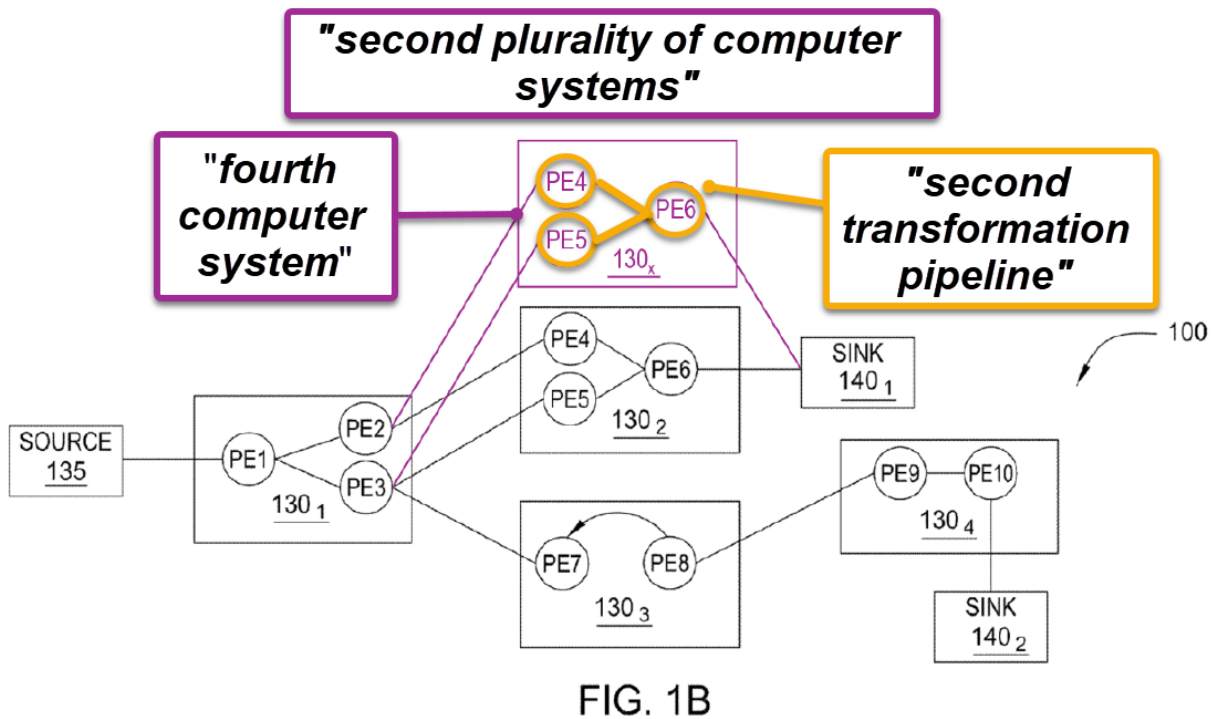


EX1004, Fig. 1A (annotations added).

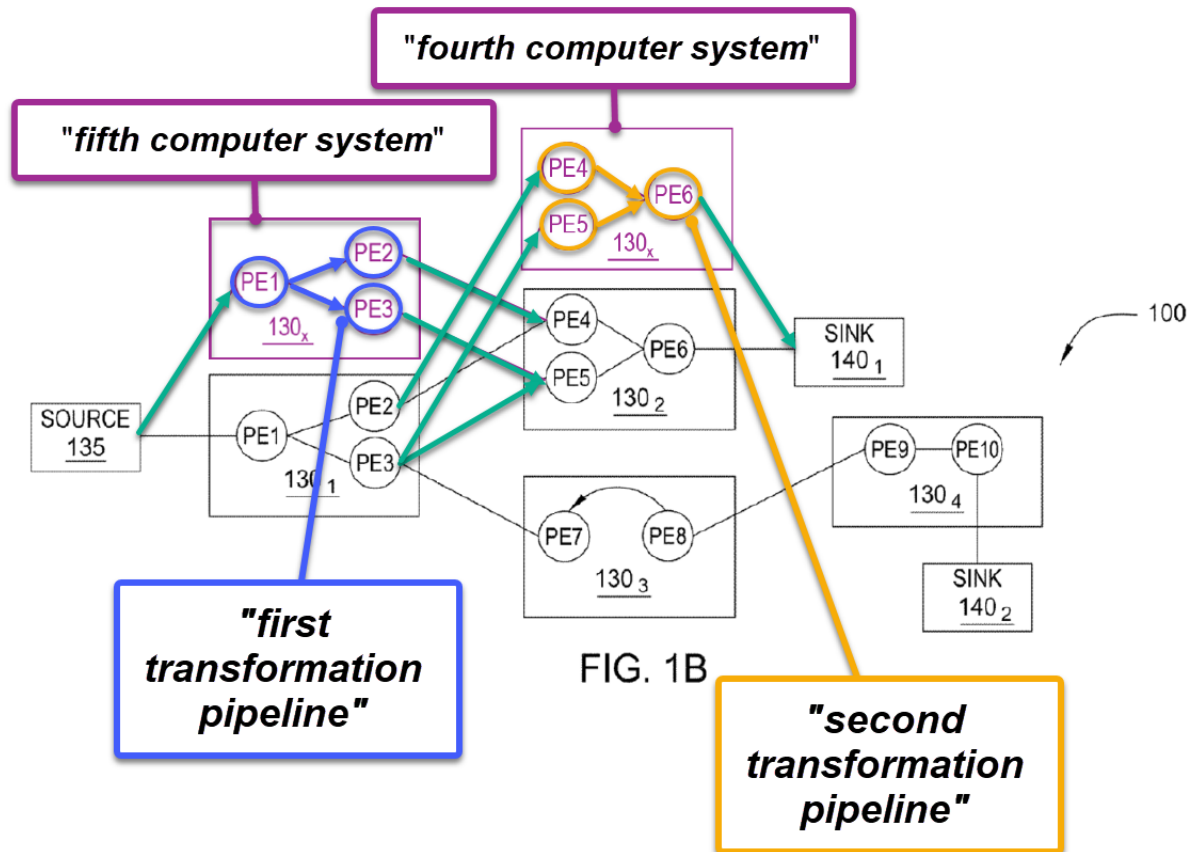
Monitoring system 110 uses the operator graph to monitor the performance of the compute nodes by collecting “tuple processing data” and calculating “current tuple processing rates” to compare with “historical tuple processing rates.” EX1004, 13:4-7. If the current processing rates fail to meet the historical rates, the monitoring system can issue “corrective actions” including instructing the management system 105 to “spawn[] new compute nodes, [and] duplicat[e] a compute node” (*fourth computer system* and/or *fifth computer system*). EX1004, 14:51-56; EX1003, ¶47, *see also* EX1004, 10:14-39, 11:27-37. Monitoring system 110 instructs management system 105 to configure the newly spawned or duplicated computer node 130_x to perform the *first transformation pipeline*. EX1004, 10:26-39.



EX1004, FIG. 1B (annotations added). Alternatively, the newly spawned or duplicated computer node 130_x performs the *second transformation pipeline*.



EX1004, FIG. 1B (annotations added). Alternatively, newly spawned fourth and fifth computer systems perform the first and second transformation pipelines.



EX1004, FIG. 1B (annotations added).

2. Claim 1

(a) [1.1]

To the extent the preamble is limiting, Barsness discloses “a stream data processing application” with “operators [] connected to one another such that data flows from one processing element (PE) to the next” and “distributing [the] application across nodes by creating many small executable pieces of code (operators), as well as replicating processing elements on multiple nodes and load balancing among them.” EX1004, 3:16-22; EX1003, ¶¶48-49; *see also* EX1004, 6:49-56, 7:52-8:2.

Barsness discloses a “computing infrastructure 100” (*distributed computing cluster*) that includes “a management system 105 and a plurality of compute nodes 130₁₋₄, each connected to a communications network 120.” EX1004, 6:43-48. Additionally, “[a] monitoring system 110 may also be included in the computing infrastructure 100.” EX1004, 7:15-18. FIG. 1A shows an exemplary computing infrastructure and its distributed computing elements:

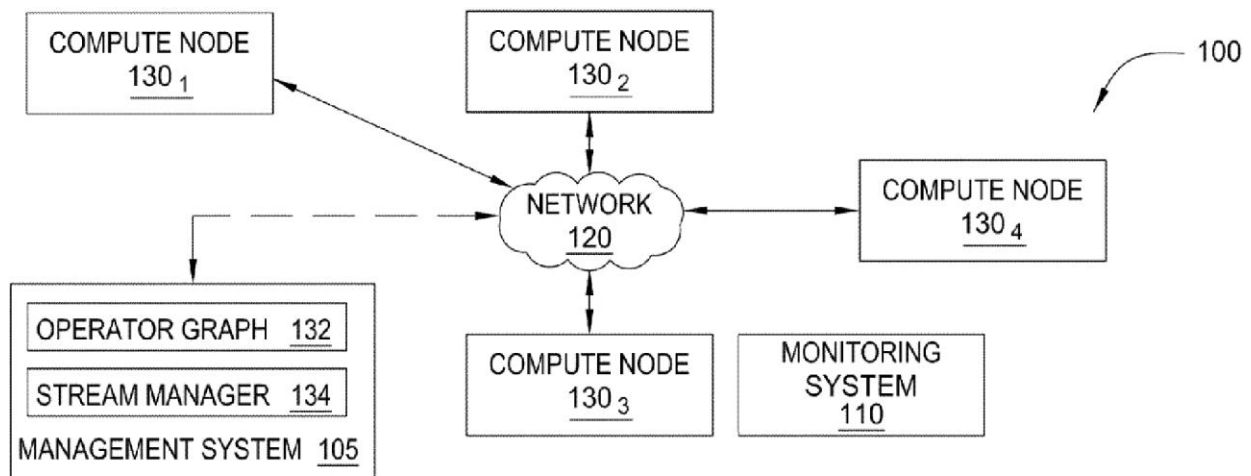


FIG. 1A

EX1004, FIG. 1A; EX1003, ¶48.

Each compute node is “a *distributed* computing node.” EX1004, 2:35-37 (emphasis added). Barsness also discloses that a monitoring system 110 of the computing infrastructure “may be a *cluster of a plurality of computing resources*, which may be *distributed* in the network 120.” EX1004, 7:17-21 (emphasis added). Thus, the computing infrastructure—with its “plurality of compute nodes” and “management” and “monitoring” systems distributed throughout network 120—is a *distributed computing cluster* because the elements communicate over the network

to perform coordinated processing. EX1003, ¶49. As explained in the following sections, the Barsness computing infrastructure comprises each of the recited features.

(b) [1.2]

Barsness satisfies this element. Barsness discloses that its computing infrastructure includes *a first plurality of computer systems*, e.g., management system 105, monitoring system 110, and at least compute nodes 130₁₋₂. EX1004, 6:45-49. FIG. 1A shows *a first plurality of computer systems*:

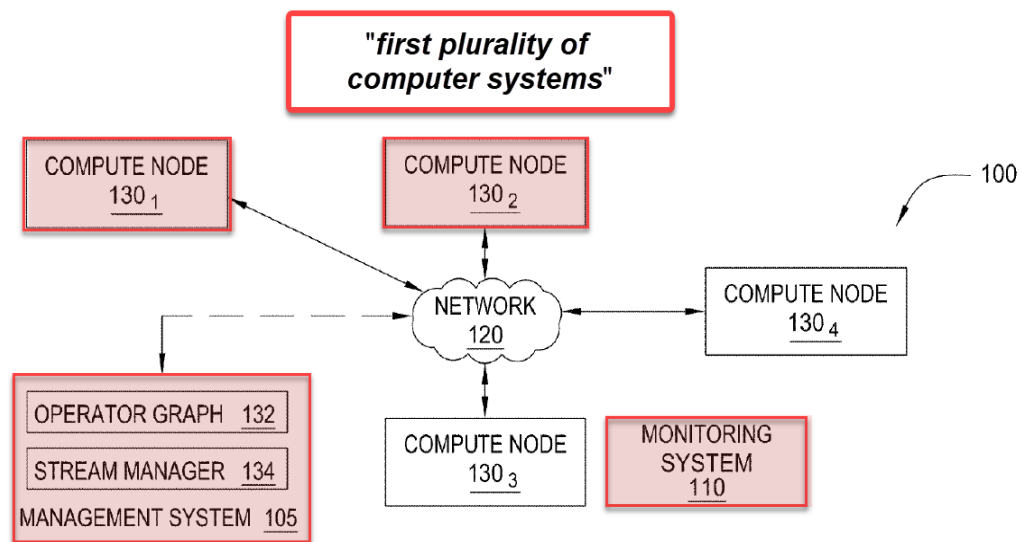


FIG. 1A

EX1004, FIG. 1A (annotations added); EX1003, ¶50.

"first plurality of computer systems"

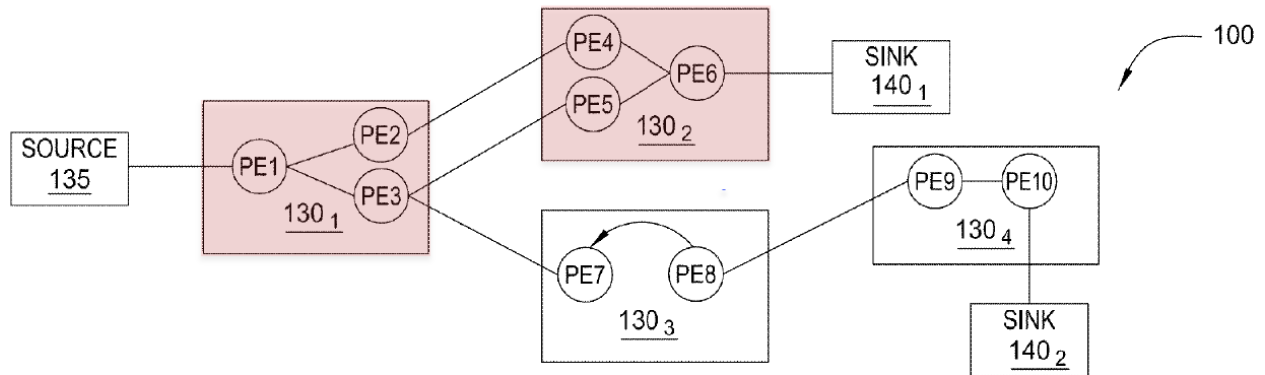


FIG. 1B

EX1004, FIG. 1B (annotations added).

Each compute node is a ***computer system*** as it includes “a central processing unit (CPU) 205, a network interface 215, an interconnect 220, a memory 225, and storage 230,” along with “an I/O devices interface 210 used to connect I/O devices 212).” EX1004, 8:30-39; *compare with* EX1001, 14:63-15:23. Barsness FIG. 2 shows details of a compute node:

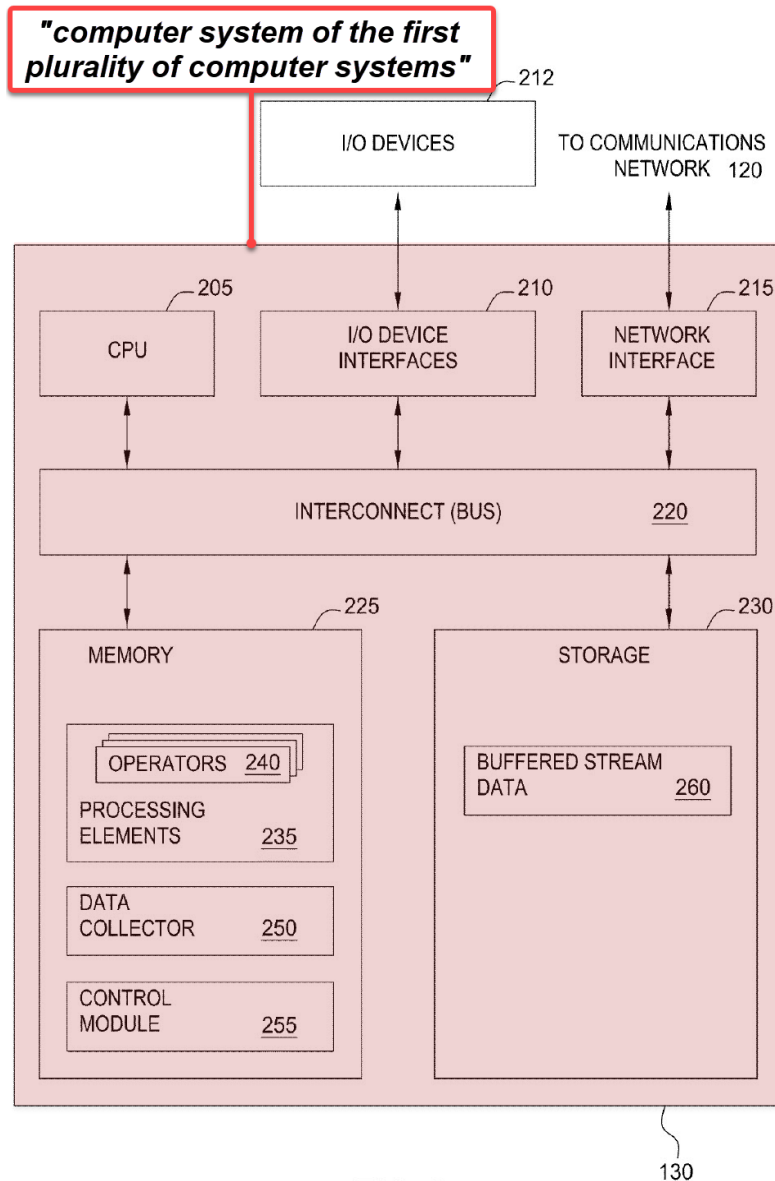


FIG. 2

EX1004, FIG. 2 (annotations added); *id.*, 8:53-65. Barsness discloses that “[t]he compute node 130 may be a physical host or a virtual machine,” that “provides a software abstraction of computing resources of a physical host computer” and “runs as an independent *computer system*.” EX1004, 8:37-44 (emphasis added); EX1003, ¶51.

The Barsness management system 105 is also a ***computer system*** because it includes a “central processing unit (CPU) 305, a network interface 315, an interconnect 320, a memory 325, and storage 330.” EX1004, 9:53-59; *see also id.*, FIG. 3; *id.*, 9:60-10:64; EX1003, ¶52.

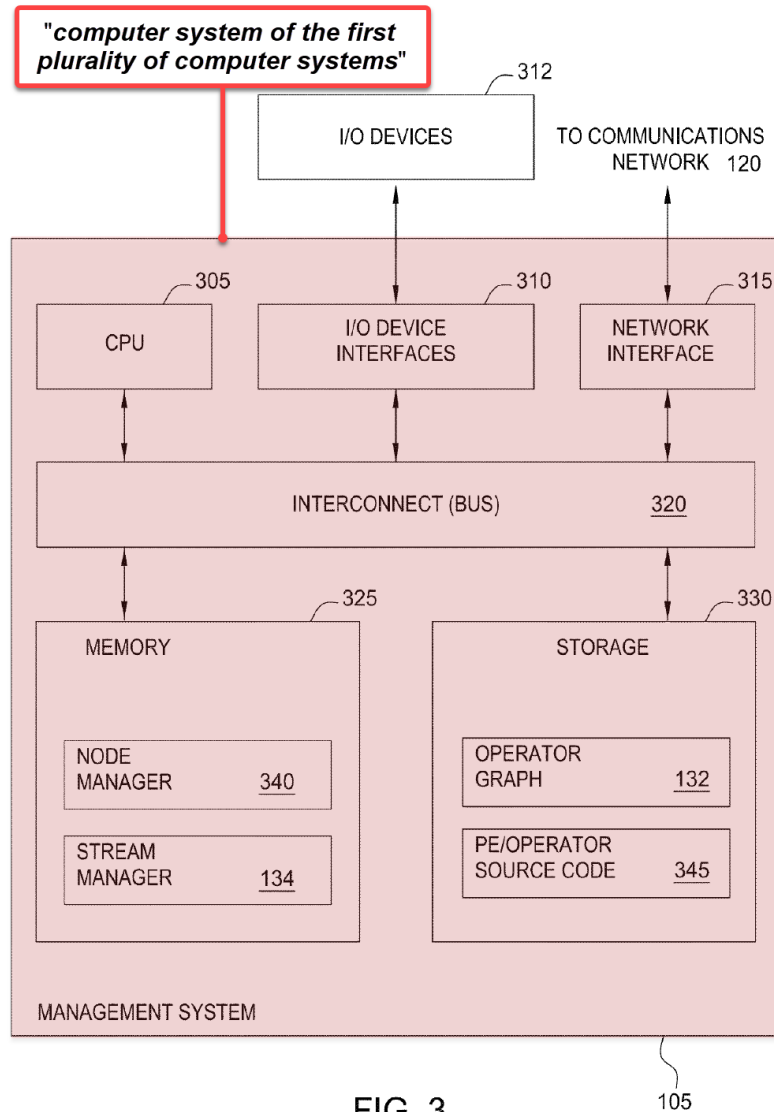


FIG. 3

EX1004, FIG. 3 (annotations added).

The Barsness monitoring system 110 is also a ***computer system*** as seen in FIG. 4. EX1004; EX1003, ¶52. Additionally, monitoring system 110 “may be a cluster of a plurality of computing resources.” EX1004, 7:17-21.

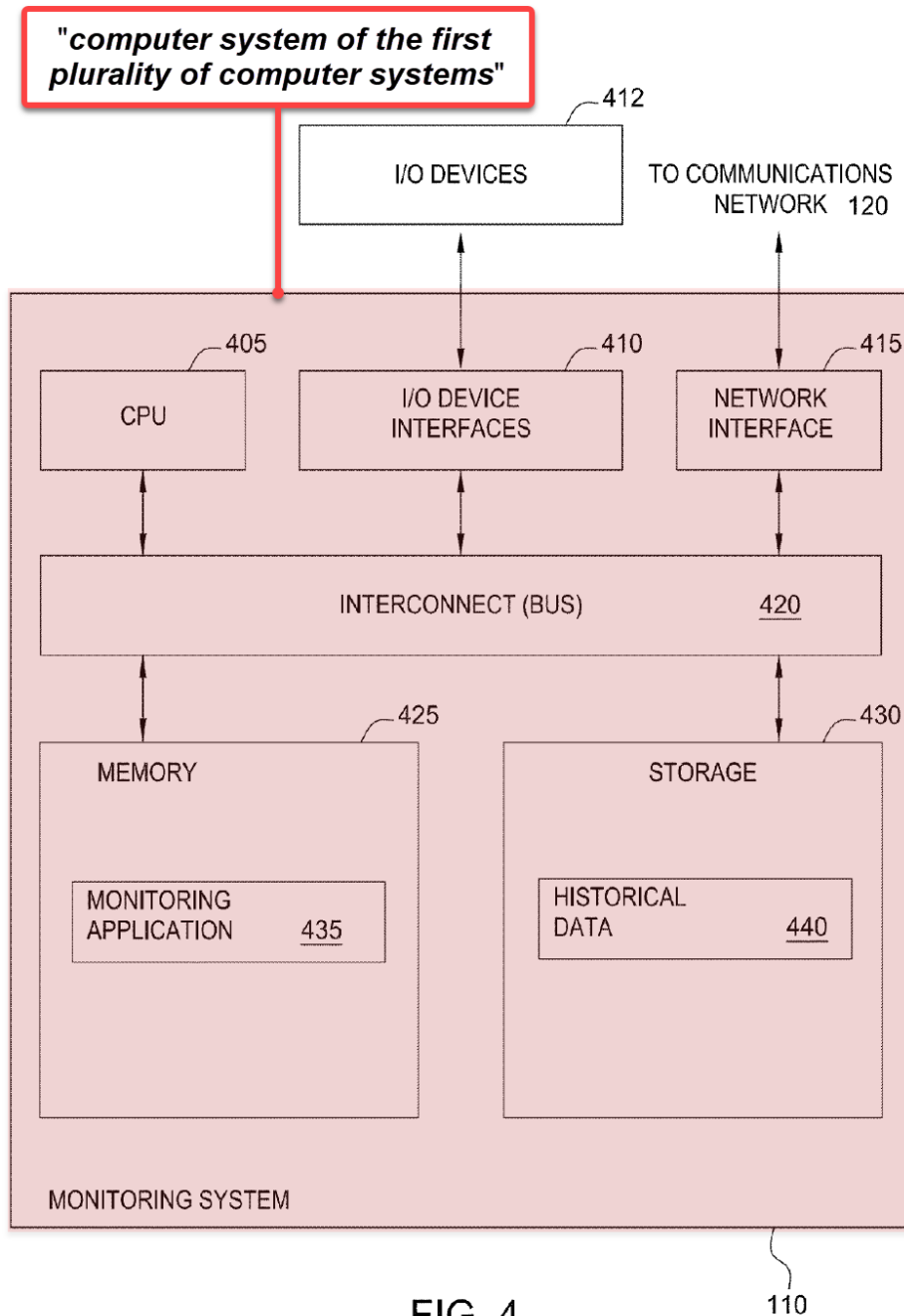
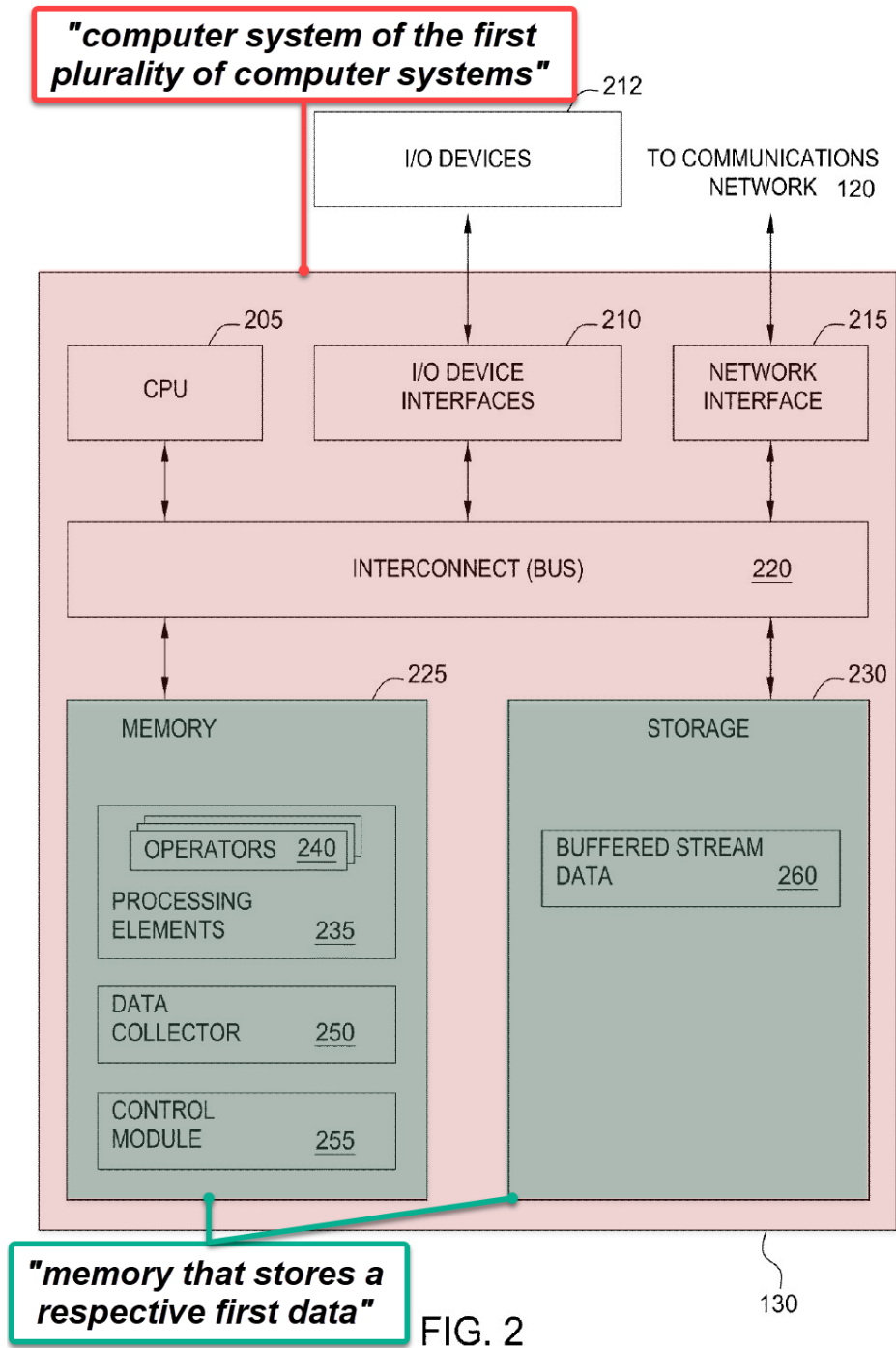


FIG. 4

EX1004, FIG. 4 (annotations added).

(c) [1.2.a]

Barsness satisfies this element. Barsness discloses that compute nodes 130₁₋₄, management system 105, and monitoring system 110 each stores operator graph 132 in memory. EX1003 ¶¶53-56. Each compute node (*respective computer system*) of the plurality of compute nodes (*the first plurality of computer systems*) comprises “memory 225” (*a memory*) that “store[s]” data, e.g., programming instructions and “application data” (*stores a respective first data*). EX1004, 8:53-55. Management system 105, also a *respective computer system of the first plurality of computer systems*, also comprises “memory 325” (*a memory*) that stores data, e.g., “programming instructions and application data.” EX1004, 9:62-67; EX1003, ¶53. Monitoring system 110’s (*respective computer system*) “memory 425” stores programming instructions and application data (*a memory*). EX1004, 11:7-11; EX1003, ¶53-56. Each computer system also includes “storage” 230/330/430 that stores data, e.g., buffered stream data 260 and operator graph 132. EX1004, 8:61-65, 9:43-50; 10:3-11; 11:14-18. The memory and storage (each alone and collectively, *a memory*) of a compute node, management system 105, and monitoring system 110 are depicted, respectively, in FIGS. 2-4 of Barsness:



EX1004, FIG. 2 (annotations added).

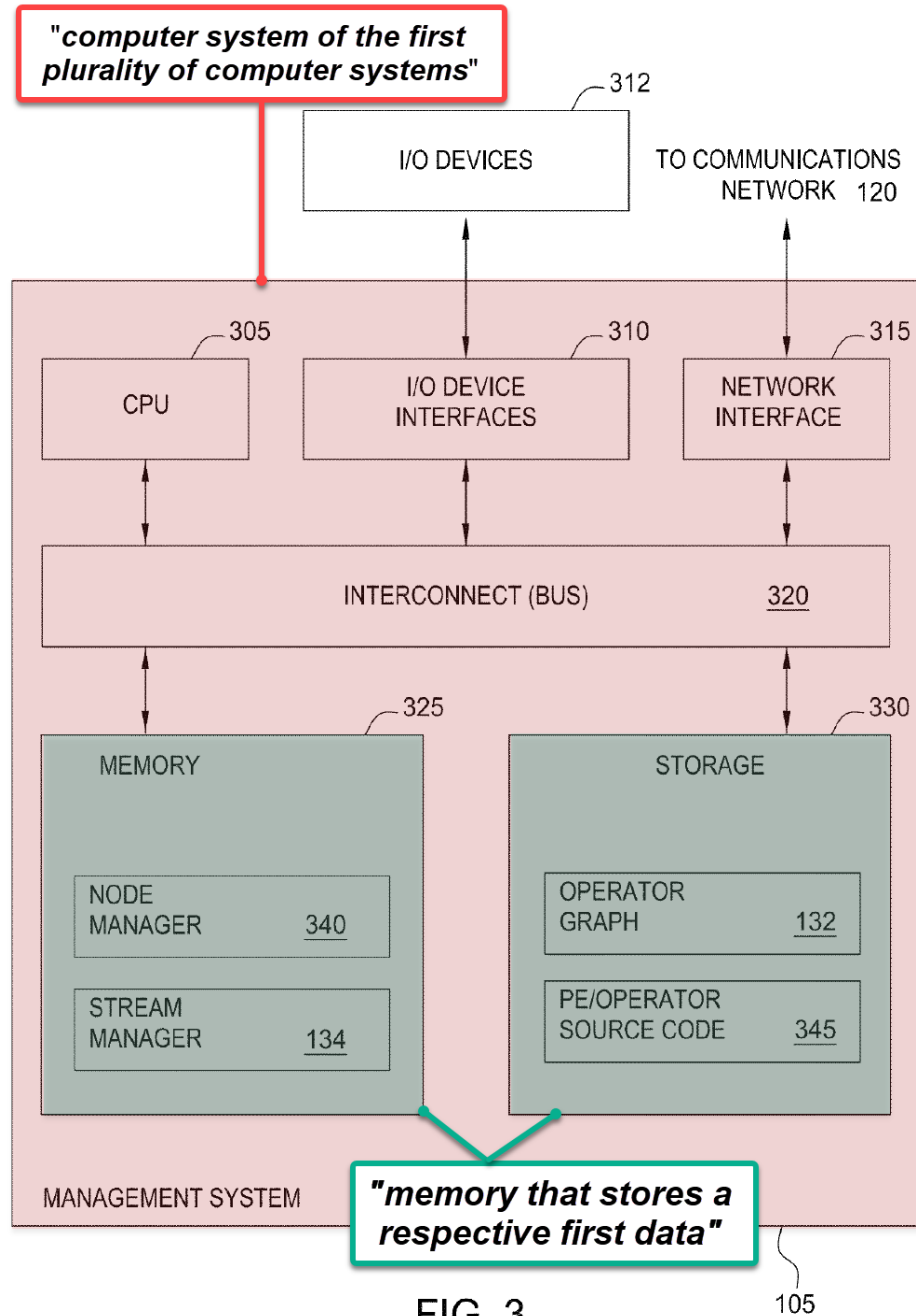
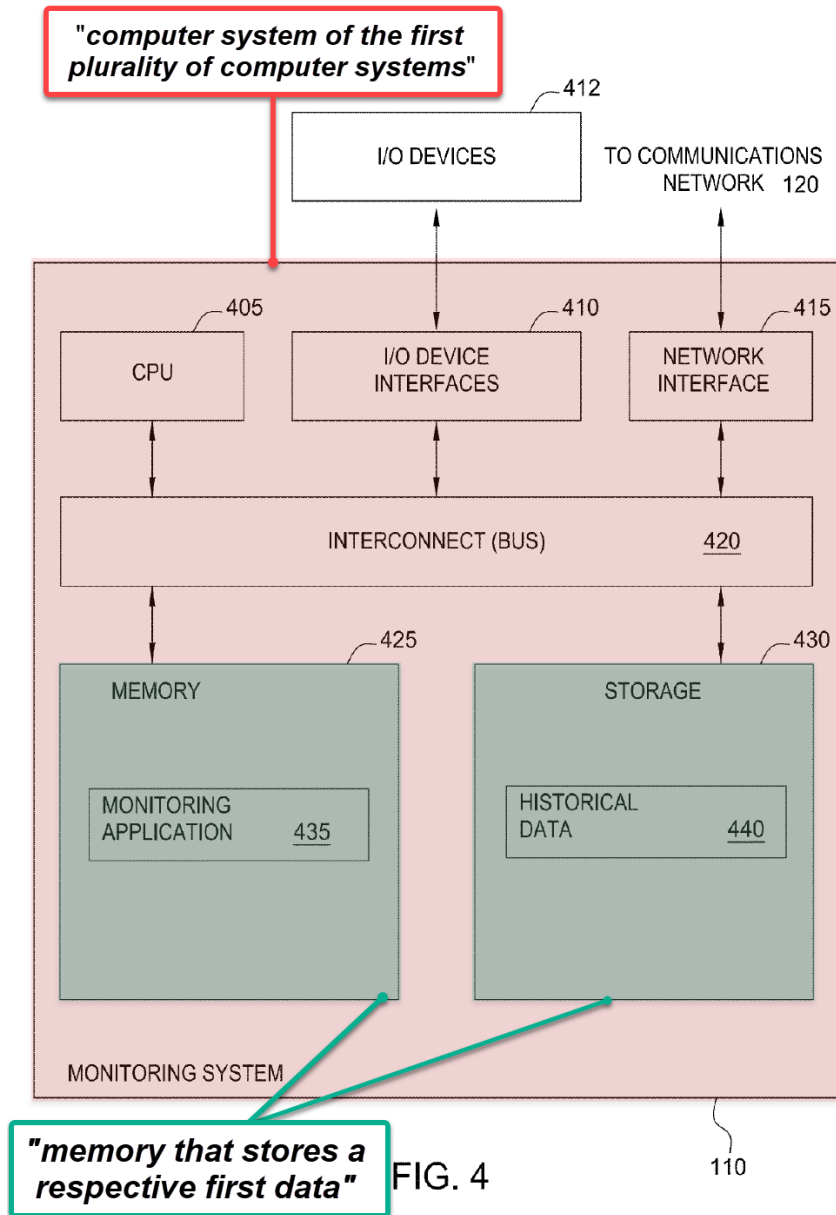


FIG. 3

EX1004, FIG. 3 (annotations added).



EX1004, FIG. 4 (annotations added).

Respective “data” stored in each respective memory of Barsness includes a *respective first data*. *Infra* Section VII.A.2.d (element [1.2.b]); EX1003, ¶56. For example, the memory of each compute node, management system 105, and monitoring system 110 includes the code and configuration of processing elements of the operator graph (*first data*). EX1004, 8:66-9:23 (describing the “executable

code” of processing element operators on compute nodes), 10:51-53 (describing “processing element and operator source code 345” as stored on the management system), 11:52-12:8 (describing the monitoring system displaying the configuration of processing elements running across compute nodes).

(d) [1.2.b]

Barsness satisfies this element. Compute nodes 130, management system 105, and monitoring system 110 each store the operator graph 132 that describes the streaming application’s compute nodes, their PEs, and the data flowing between those PEs and between compute nodes. Regarding a *distributed computational graph*, Specifically, Barsness discloses an “operator graph” (*distributed computational graph*) that “represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data tuples through the stream application.” EX1004, 1:43-46; *id.*, 2:12-15, 2:59-62, 6:43-56, 10:49-51, FIGS. 1A- 1B; *Infra* Section VII.A.2.e; EX1003, ¶57.

FIG. 1B depicts an example operator graph, where the PEs are **graph nodes** and the links between PEs are **graph edges**:

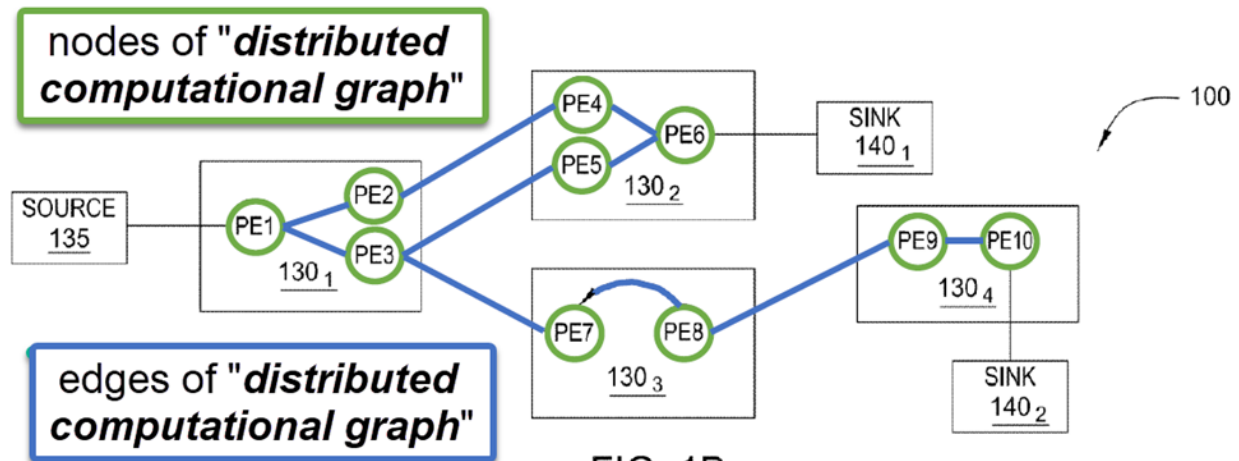


FIG. 1B

EX1004, FIG. 1B (annotations added). PEs are grouped into respective compute nodes 130_{1-4} as shown. Each PE includes an operator comprising executable code to perform an operation on data input to the PE. EX1004, 3:16-35, 6:57-7:3, 9:3-8, FIG. 2; EX1003, ¶58.

As described above regarding claim construction, *distributed computational graph* means “[a] directed graph with each transformation comprising a node and the output messages between transformations comprising edges.” *Supra* VI. Operator graph 132 meets this construction because: each *node*² corresponds to a PE that performs a *transformation* on data in the stream, and each *edge* corresponds to the output from one PE to another PE.

² Within the Barsness operator graph, each *graph node* is a PE; Barsness uses the term “compute node” to refer to the computer systems that execute or run the PEs.

- “*directed graph*”: Operator graph 132 is *directed* because the data stream flows through the PEs (*nodes*) from source 135 to sinks 140₁₋₂. Each *edge* indicates the flow of data from one PE node to the next PE node. EX1004, 1:43-46, 2:12-15, 2:59-62, 6:43-56, FIG. 1B; *see also* EX1001, 9:27-31 (describing directed graphs); EX1003, ¶59. For example, Barsness explains that data flows from PE1 to PE2 and PE3, and from PE2 and PE3 to PE4 and PE5, and then from PE4 and PE5 to PE6. EX1004, 7:52-61.
- “*transformation comprising a node*”: Each PE (*node*) corresponds to a *transformation* of data, i.e., “a function performed on zero or more streams of input data which results in a single stream of output” (*supra* VI). Each PE is configured to “carry out any form of data processing functions” including “data analytic functions or operations” on data received through the data stream. EX1004, 6:49-7:3; *id.*, 9:3-6. To that end, each PE includes a collection of operators with each operator comprising “a small chunk of executable code configured to process data flowing into a processing element.” *Id.*, 9:3-9; 3:16-22. Barsness teaches that each PE’s function *transforms* the input. *Id.*, 6:60-62 (“[T]he N-tuple received by a processing element need not be the same N-tuple sent downstream”); 8:12-15 (“PEs may be configured to

transform incoming data tuples by either adding or removing one or more data attributes before emitting outgoing data tuples to other PEs”) (emphasis added). The function performed by each PE results in a single stream of output emitted “to other PEs in the stream application.” *Id.*, 9:4-9; *see also id.*, 9:43-50; 7:55-56 (“Source PE1 emits tuples received by PE2 and PE3.”); *compare with* EX1001, 18:37-42 (describing node sending same “output stream to two transformational pipelines”). Barsness describes, alternatively, that a PE “may split data elements” (EX1004, 7:56-57), which together still comprises a single output stream. EX1003, ¶59.

- “*output messages between transformations comprising edges*”: The edges of operator graph 132 are “links between processing elements, corresponding to a flow of data tuples through the stream application,” and thus comprise output messages between PEs, where the data stream emitted from a PE comprises output messages from that PE. EX1004, 2:12-15; *id.*, 6:57-65; *see also* EX1005, [0122]; EX1015, 12:11-14. The ’424 patent provides no meaningful explanation of “output messages” but uses that term to refer to data flowing from a transformation node. *E.g.*, EX1001, 17:17-19; EX1003, ¶59.

See also infra Section VII.A.2.e (explaining Barsness operator graph).

Under PO's "plain and ordinary meaning," the Barsness operator graph is a ***distributed computational graph*** because it is a graph that describes the distributed computations performed on a data stream, including the distribution of PEs across different compute nodes and the flow of data between the PEs. EX1003, ¶60.

Within each compute node, memory stores ***the respective first data [that] represents a respective portion of a distributed computational graph***. For example, "memory 225 includes one or more processing elements (PEs) 235." EX1004, 8:66-9:1; *id.*, FIG. 2. As each PE is a node in the operator graph, in storing a given compute node's respective PEs including their code (i.e. "programming instructions") and configuration (i.e. "application data"), each compute node memory thus stores data representing a respective portion of the operator graph (***a respective portion of a distributed computational graph***). EX1004, 8:53-55, 8:66-67, 9:1-33 (describing PEs on compute nodes); *id.*, 2:3-4, 1:39-41, 1:55-57; 3:19-22; 10:51-56. For example, the operator graph of FIG. 1B shows how first compute node 130₁ includes PE1, PE2, and PE3, and thus stores respective first data representing a respective portion of the operator graph; similarly, the other compute nodes 130₂₋₄ store respective PEs as depicted in the operator graph. EX1004, FIG. 1B; EX1003, ¶¶61-62.

The management system 105 (*respective computer system*) also includes memory (memory 325 and storage 330) storing the operator graph 132 (*respective first data that represents a respective portion of a distributed computational graph*). The management system's operator graph includes the code and configuration for processing elements. EX1004, 10:51-53. This allows the management system 105 “to control a stream application running on the compute nodes 130₁₋₄ as well as change the structure of the operator graph 132.” EX1004, 7:27-29; *id.*, 6:48-56. Storing the operator graph 132 in the memory of management system 105 allows node manager 340 and stream manager 134 to build and configure the operator graph. EX1004, 7:27-37; *id.*, 10:12-64 (describing management of the operator graph by the management system); EX1003, ¶63.

The monitoring system 110 (*respective computer system*) also includes memory (memory 425 and storage 430) storing monitoring application 435 (*first data*) including operator graph 132 (*respective first data that represents a respective portion of a distributed computational graph*). EX1004, 3:45-50 (describing monitoring system 110's “graphical representation of the operator graph”), 11:52-12:8 (describing the monitoring system's operator graph as including the configuration of processing elements). Specifically, Barsness teaches that monitoring application 435 includes a graphical user interface (GUI) 510 “configured to depict the operator graph 132 of a selected stream application”

including “the interconnected computer nodes 130 and processing elements (PEs) as shown in FIG. 1B.” EX1004, 11:53-61. GUI 510 includes “displaying of the operator graph 132” and its “PEs (and operators) and their interconnections.” *Id.* 14:5-6, 14:1-5; *see also id.*, 11:65-12:5, 14:11-12; EX1003, ¶63.

(e) [1.2.c]

Barsness satisfies this element. The Barsness operator graph (*distributed computational graph*) describes the flow of data through several compute nodes, each containing a group of PEs that process / transform the data flowing through them. *E.g.*, EX1004, 8:11-15; *see also id.*, 7:52-54; *supra* Section VII.A.2.d (explaining, in second bullet point, the transformations performed at each PE); EX1003, ¶64.

The group of PEs for a given compute node is a *transformation pipeline*, as construed above, because those PEs form “a progression of functions which each perform a transformation on a data stream” passing through the compute node. *Supra* Section VI (providing construction). In Barsness, “data flows from one processing element (PE) to the next.” *E.g.*, EX1004, 3:16-18; *see also id.*, 3:22-28. As the data flows from one PE to the next, each PE performs a data processing operation on the data via its operator and emits the transformed data to the next PE(s) in the graph. EX1004, 6:57-7:3; *id.*, 6:55-56, 3:16-22, 9:4-7. Barsness teaches that each PE may have “only one” active operator. EX1004, 9:9-23. The group of PEs within a given

compute node thus comprises a progression of functions that each perform some action or transformation on the data stream. EX1003, ¶65.

The operator graph depicted in Barsness FIG. 1B “begins at a source PE 135 (labeled as PE1) and ends at sink PEs 140₁₋₂ (labeled as PE6 and PE10).” EX1004, 7:52-54. As shown in FIG. 1B, each compute node (boxes 130₁₋₄) contains a group of PEs that form a *transformation pipeline*. For example, the progression of transformations performed by PE1, PE2, and PE3 in the compute node 130₁ is a *first transformation pipeline*.³ EX1003, ¶66. Compute node 130₁’s PE1, PE2, and PE3 each apply operators to data elements flowing from PE1 to PE2 and PE3 (*first transformation pipeline*). PE1 processes the received data elements and “emits tuples received by PE2 and PE3.” EX1004, 7:52-57; *see also id.*, 6:55-56, 6:57-60. Then PE2 and PE3 process the received tuples and emit additional tuples. EX1004, 7:58-59, 7:64-67; EX1003, ¶67. The *first transformation pipeline* comprises:

- (1) PE1 processing of data elements into tuples,
- (2) the receipt of those tuples by PE2 and PE3, and
- (3) the processing of those tuples by PE2 and PE3.

³ A pipeline can include branching and iterative configurations. EX1001, 9:67-10:7; *infra* Section VII.3 (claim 2).

The transformations performed by PE4, PE5, and PE6 form a ***second transformation pipeline***.

Thus, the operator graph describes a flow of output data from the ***first transformation pipeline*** to inputs (PE4 and PE5) of the ***second transformation pipeline***, as shown by the graph edges between PE2-PE4 and PE3-PE5 respectively:

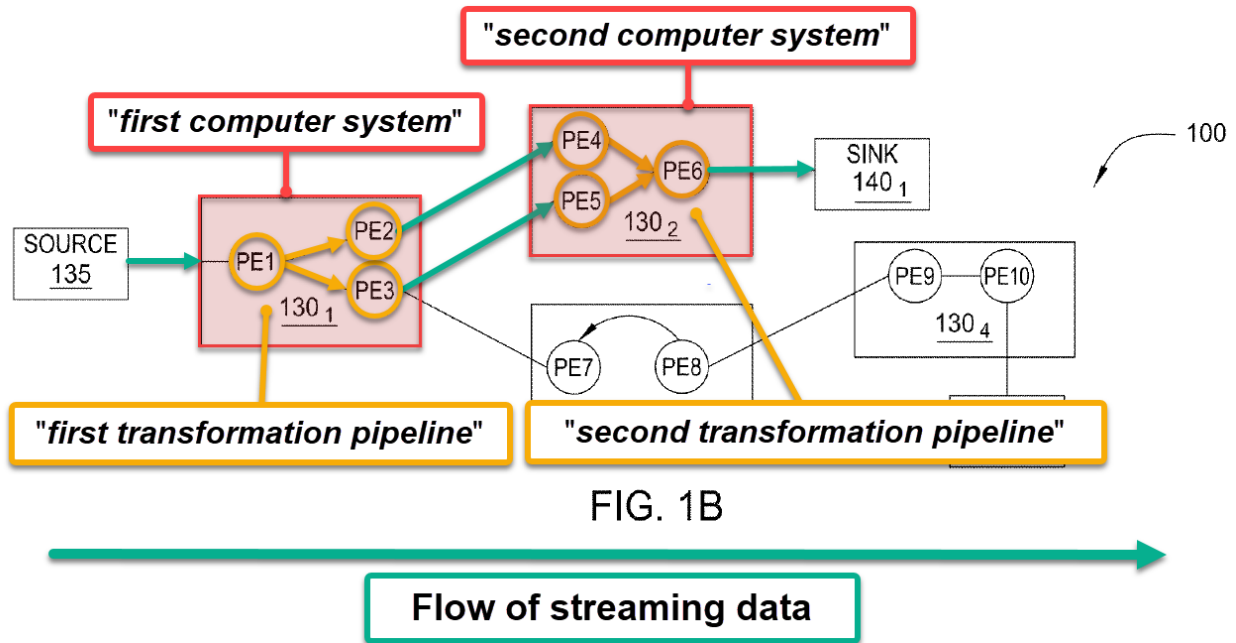


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 7:52-67 (describing the flow of data tuples through the nodes and PEs of the operator graph). The combinations of PEs PE1-PE2 and PE1-PE3 are each a transformation pipeline for the same reasons PE1-PE2-PE3 is a transformation pipeline; similarly, PE4-PE5 and PE4-PE6 are transformation pipelines for the same reason PE4-PE5-PE6 is a transformation pipeline. EX1003, ¶¶67-68. These examples show Barsness operator graph

implementations that describe data streams flowing through a series of interconnected transformation pipelines. *Id.*

Under PO's "plain and ordinary meaning," the group of PEs associated with a given compute node—including each group mentioned in the preceding paragraph—forms a ***transformation pipeline*** as each PE performs a ***transformation*** and the group of PEs forms a pipeline or series of transformations on the data stream. EX1004, 6:57-7:3; *id.*, 6:55-56, 3:16-22, 9:4-7; EX1003, ¶69.

(f) [1.3]

Barsness satisfies this element. For example, the ***first plurality of computer systems*** (*supra* Section VII.A.2.b) includes first compute node 130₁ (***first computer system***) within this plurality of systems, as depicted in Barsness FIG. 1B:

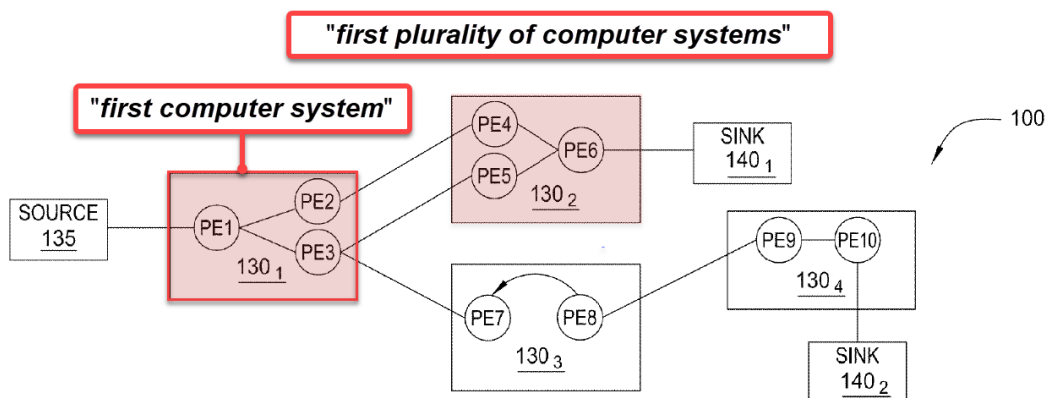


FIG. 1B

EX1004, FIG. 1B (highlighting added); *see also id.*, FIG. 2, 7:54-55, 8:30-34; EX1003, ¶70. The following sections explain how Barsness teaches configuring this and other first computer systems in the manner required by claim 1.

(g) [1.3.a]

Barsness satisfies this element. Barsness discloses “[d]ata elements flow[ing] into a source PE of a stream application” (*a first stream of input data from a first input feed*). EX1004, 6:49-56; *id.*, 7:52-54, 10:40-48 (“data stream”). Barsness further discloses “a data source for the stream application” (*first input feed*) and “a storage space for data flowing into the compute node.” EX1004, 9:43-46. This is shown in FIG. 2, where storage contains “[b]uffered stream data 260”:

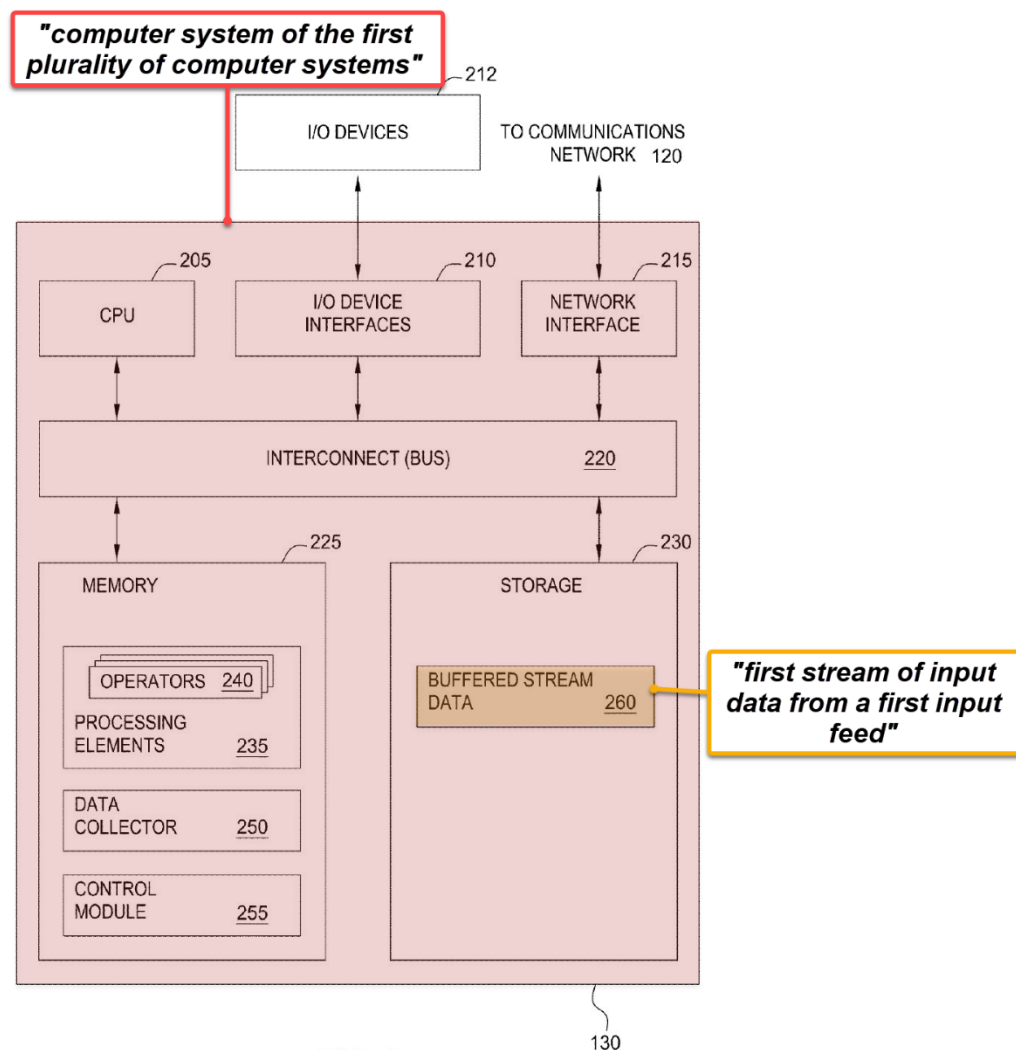


FIG. 2

EX1004, FIG. 2 (annotations added); see also EX1004, 1:22-24, 3:16-18, 9:43-50, 10:20-26; EX1003, ¶¶71-72.

Barsness discloses that compute node 130₁ receives “data elements” from the stream source 135 (*a first stream of data*). EX1004, 7:52-57; *id.*, 6:55-56; *see also* 6:57-65. FIG. 1B shows compute node 130₁ with “source PE 135 (labeled as PE1)” (EX1004, 7:52-54) *receiv[ing]* the “[d]ata elements flow[ing] into a source PE of a stream application” (EX1004, 6:55-56):

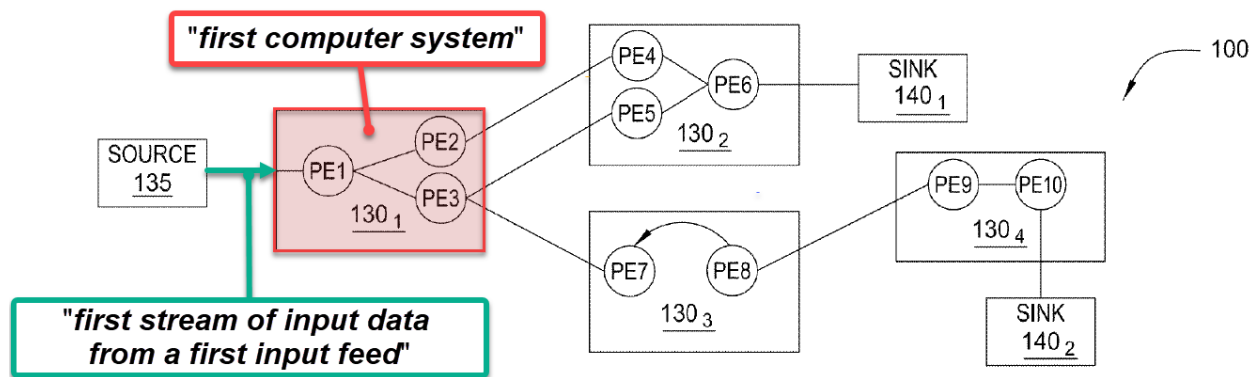
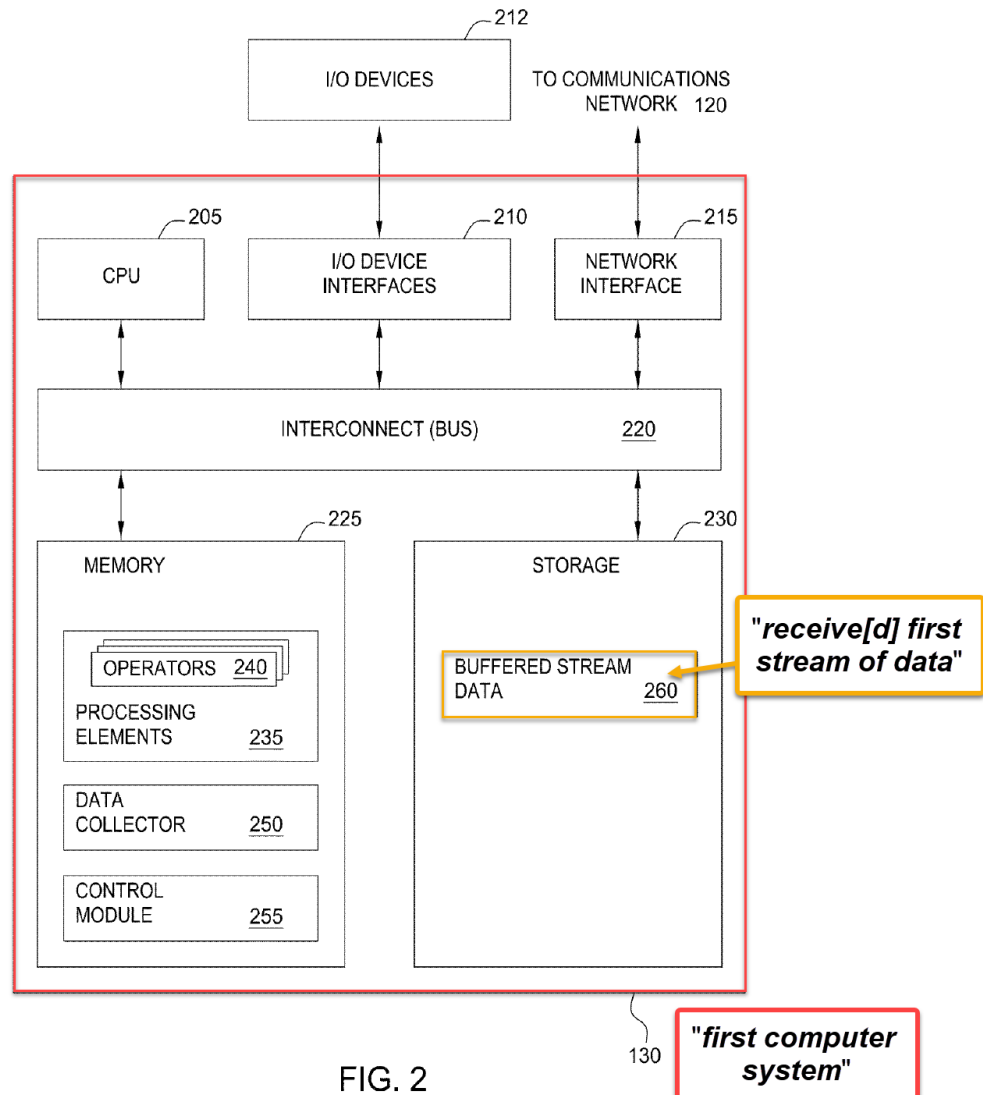


FIG. 1B

EX1004, FIG. 1B (annotations added), 6:49-56; EX1003, ¶73. Barsness further discloses each compute node includes a “[b]uffered stream data 260” that “represents a storage space for data flowing into the compute node,” as seen in FIG. 2 below. EX1004, 9:43-50.



EX1004, FIG. 2 (highlighting added); *see also* EX1004, 1:22-24, 3:16-18, 9:43-50, 10:20-26; EX1003, ¶73.

(h) [1.3.b]

Barsness satisfies this element. As explained for elements [1.2.b]-[1.2.c], each compute node processes the source streaming data using transformation pipelines made up of PEs. *Supra* Sections VII.A.2.c-d. For example, the FIG. 1B first compute node 130₁ (***first computer system***) includes PEs PE1, PE2, and PE3 forming a ***first***

transformation pipeline applied to the data elements received from the first stream of input data at PE1 (*process the first stream of input data by applying a first transformation pipeline*):

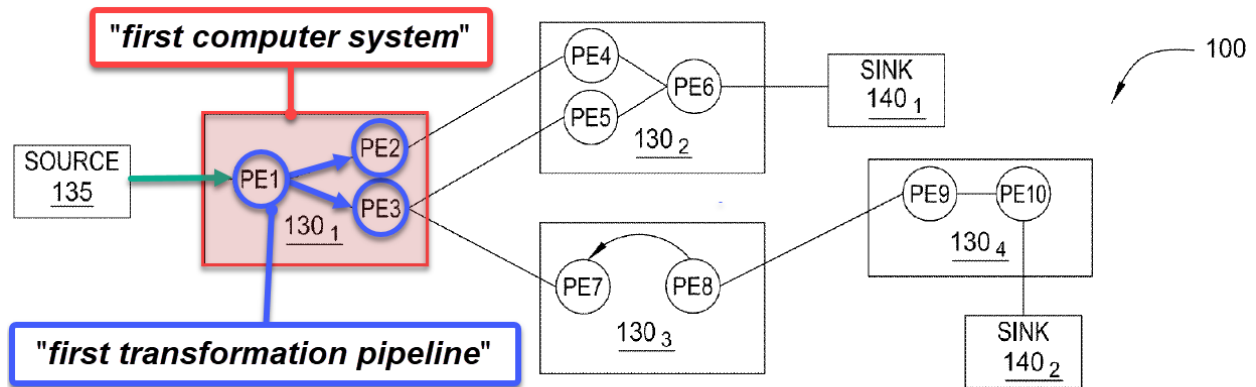


FIG. 1B

EX1004, FIG. 1B (annotations added); EX1003, ¶74.

“process the first stream of input data substantially in real time”: Barsness teaches that the stream application processes data in real-time. *E.g.*, EX1004, 2:52-3:15 (describing “real-time monitoring”). Barsness describes its “stream data processing application” system in the context of providing improvements to “real-time data processing,” i.e., “when the incoming data need to be processed in real-time,” and “without losing the real-time characteristics of the data processing.” EX1004, 2:58-61, 3:3-6, 3:16-18. Barsness further describes “processing logic to apply to inbound data records while they are ‘in flight,’ with the results available in milliseconds.” EX1004, 1:24-27; EX1003, ¶77; *see also* EX1001, 15:54-61.

The Barsness architecture further confirms real-time data processing, as input data streams proceed directly to processing (e.g., directly to source PE 135 of the first compute node 130₁), rather than first being stored in a database. EX1004, 7:52-56, FIG. 1B; *see also id.*, 1:15-19; EX1003, ¶78.

Further, Barsness teaches monitoring “processing rates at a selected PE” and—as needed when a particular processing element slows—“reconfigure[ing] the processing modules of the stream application” such as by duplicating the compute node. EX1004, 12:28-36; *id.*, 3:61-4:2, 12:40-13:3, 13:12-14, 13:24-29, 14:30-39, FIG. 6. These dynamic pipeline adjustments ensure the data stream is processed with sufficient speed, further showing that the Barsness compute nodes *process the input data stream substantially in real time*. EX1004, 3:3-6; EX1003, ¶79.

“by executing software instructions that apply the first transformation pipeline to the first stream of input data”: Barsness teaches that each compute node’s CPU executes “executable code” associated with operators of the PEs for that compute node (*transformation pipeline*). For example, the CPU of a given compute node / computing system (including compute node 130₁) “retrieves and executes programming instructions stored in the memory 225,” the memory storing one or more processing elements, and each processing element including an operator. EX1004, 8:53-54, 8:66-9:14; *see also id.*, 7:13-15. Each processing element performs “data analytic functions or operations” according to the operator provided

at the processing element. EX1004, 6:65-7:3; *id.*, 7:40-44. And each operator is a “small chunk of executable code configured to process data flowing into a processing element (e.g., PEs 235) ... as described by the operator graph 132.” EX1004, 9:4-9; *id.*, 3:19-22. Barsness FIG. 2 shows this:

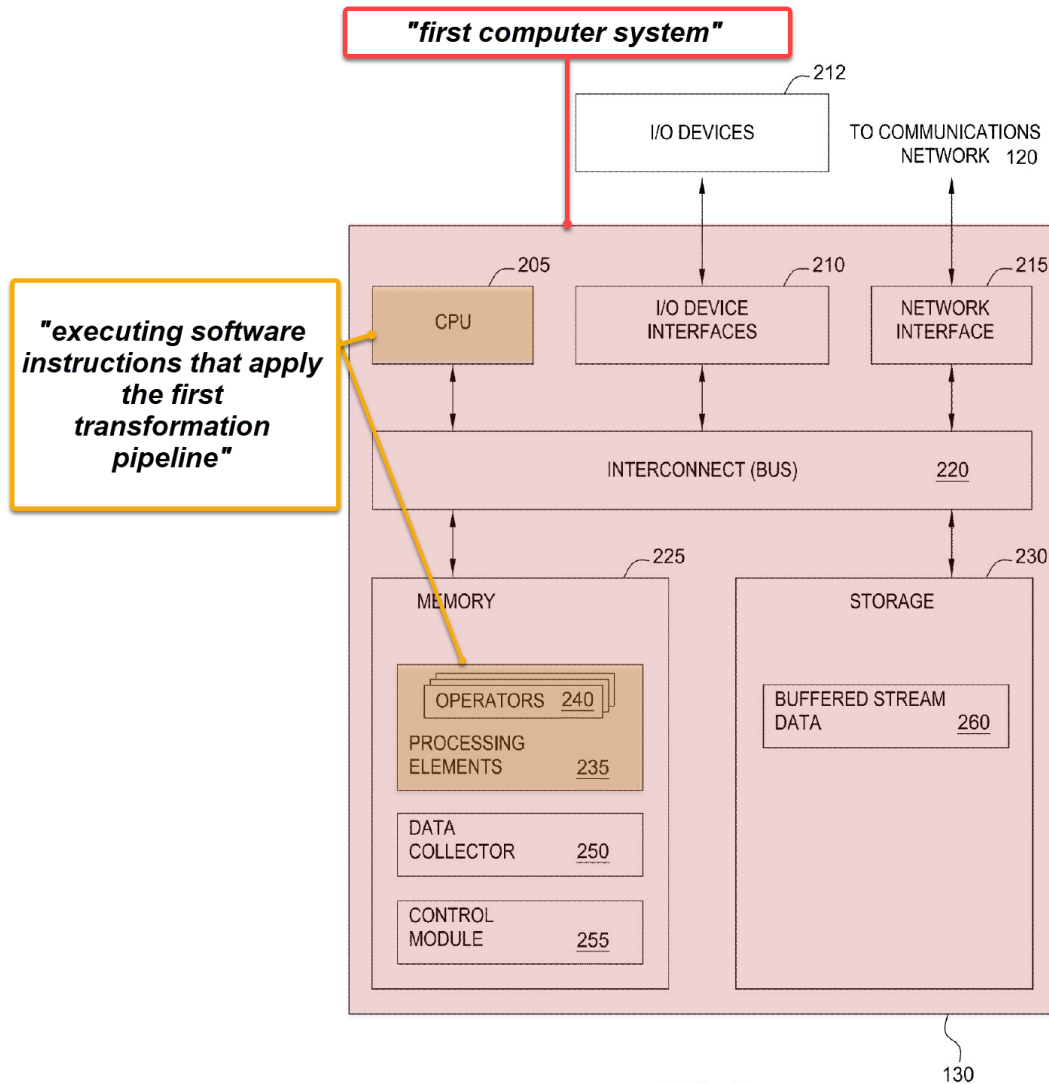
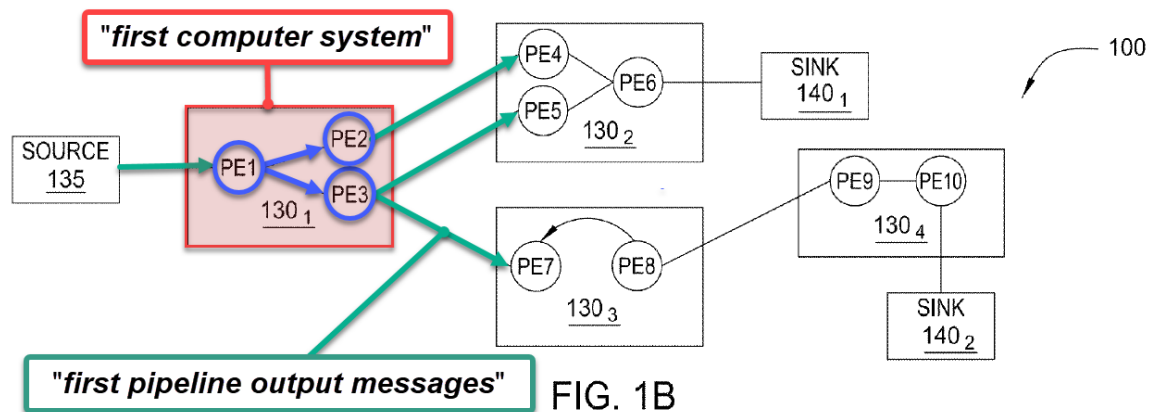


FIG. 2

EX1004, FIG. 2 (annotations added); EX1003, ¶75.

“to generate first pipeline output messages”: Barsness discloses that the “processing elements ... emit an *N*-tuple of data elements into the stream (except for

a sink PE where the stream terminates).” EX1004, 6:57-60 (emphasis added). The output data emitted by processing elements PE2 and PE3 are *first pipeline output messages* because they are the resulting output of the progression of transformations applied to the input data by the processing elements PE1, PE2, and PE3 (*the first transformation pipeline*):



EX1004, FIG. 1B (annotations added); *id.*, 6:62-65, 7:52-67; EX1004, 9:4-9; *supra* Section VII.A.2.e; EX1003, ¶76.

(i) [1.3.c]

Barsness satisfies this element. As explained (*supra* Sections VII.A.2.c-VII.A.2.e), Barsness discloses storing first data representing a respective portion of an operator graph (*distributed computational graph*) in the memory of each respective compute node (each a *computer system*), including compute node 130₁ (*the first computer system*). EX1003, ¶80. That stored data includes operator “executable code.” EX1004, 9:4-9, FIG. 2 (“operators” in compute node memory).

Barsness teaches that each PE executes this operator code, which specifies where output data should be sent next in the stream application, such as “other PEs in the stream application.” EX1004, 9:4-9. Relatedly, Barsness describes PEs passing data elements on to subsequent PEs (*id.*, 7:56-58, 8:14-15) and a PE on one compute node emitting data to a next PE on another compute node (*id.*, 7:58-59; *id.*, 7:60-61, 10:49-51, 2:12-15, 3:45-50). EX1003, ¶81. Thus, when the first compute node executes the operator code associated with the last processing element in the first transformation pipeline, it processes stored data that identifies the downstream destination for that output—namely, the next processing element or processing elements in the second transformation pipeline. In this way, the first compute node processes stored data to determine information about the second transformation pipeline and transmits output accordingly. EX1003, ¶85.

To further illustrate, in the example operator graph of FIG. 1B, processing elements PE2 and PE3 of first compute node 130₁ (**a *first transformation pipeline***) emit output data messages to processing elements PE4 and PE5 of the second compute node 130₂ (***second transformation pipeline***):

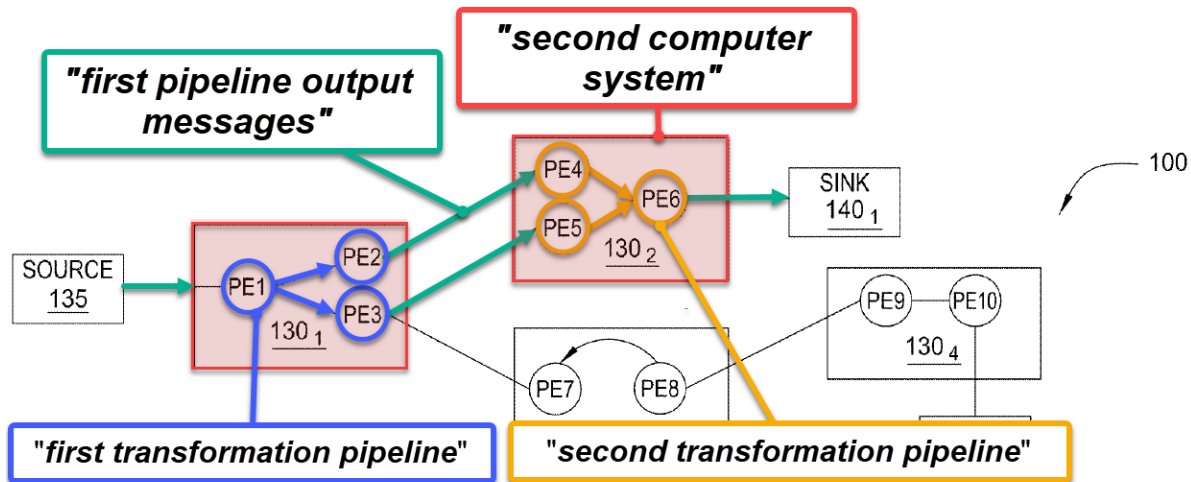


FIG. 1B

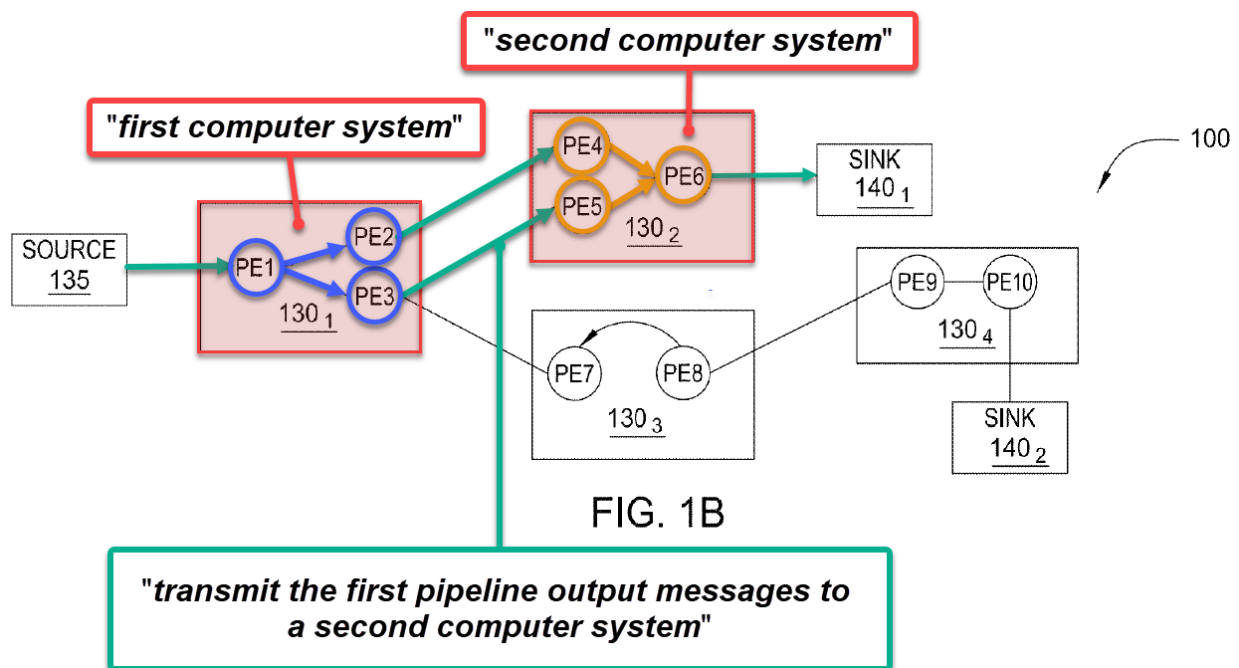
EX1004, FIG. 1B (annotations added); *id.*, 7:52-67. Applying Barsness's teachings, output data is directed to PE4 and PE5 via the respective operators of PE2 and PE3. Processing of those operators by compute node 130₁ (*process the respective first data stored in the memory of the first computer system*), determines destinations PE4 and PE5 (*information about the second transformation pipeline*). EX1003, ¶86.

Barsness describes an exemplary way of directing data between nodes: TCP/IP sockets, with each PE operator having an "input and output port." EX1004, 3:16-18; *id.*, 3:24-28, 7:44-47, 9:9-12, 11:42-45; *see also* EX1017, 1:22-2:26, FIG. 1; EX1016, 8:56-62, FIG. 2; EX1003, ¶¶82-84. Processing the data stored in a given first compute node would determine information about a given second transformation pipeline, i.e., the destination port for the first PE(s) in that second pipeline. EX1003, ¶87.

Thus, a POSITA would have implemented the Barsness operator/executable code of each PE to provide information identifying the downstream PE(s) to which the given operator should emit its output data. EX1003, ¶88.

(j) [1.3.d]

Barsness satisfies this element. Barsness discloses a given first compute node transmitting outgoing data to a second computing system of the first plurality of computer systems according to the information determined by executing the operator of the first compute node. *Supra* Section VII.A.2.e; EX1004, 8:11-15. In the example provided by Barsness, PEs 2 and 3 in the first compute node 130₁ emit/transmit data (*transmit the first pipeline output messages*) to PEs 4 and 5 of compute node 130₂ (*a second computer system of the first plurality of computer systems*):



EX1004, FIG. 1B (annotations added); *supra* Section VII.A.2.e; *see also* EX1004, 7:58-59 (“Data that flows to PE2 results in tuples emitted to PE4 on compute node 130₂”), 7:64-67. In FIG. 1B, this transmission is reflected in each of the edges between PE2-PE4 and PE3-PE5. As discussed *supra* Section VII.A.2.c-VII.A.2.e, the operators of Barsness identify the destination of their output data, and thus the first compute node (*first computer system*) transmits the output of the first compute node (*first pipeline output messages*) to the second computer system according to the destination(s) specified in the operator (*the determined information*). EX1004, 3:16-18, 7:44-47; EX1003, ¶89.

(k) [1.4]

Barsness satisfies this element. Barsness discloses a second compute node 130₂ (*second computer system*) as shown in FIG. 1B:

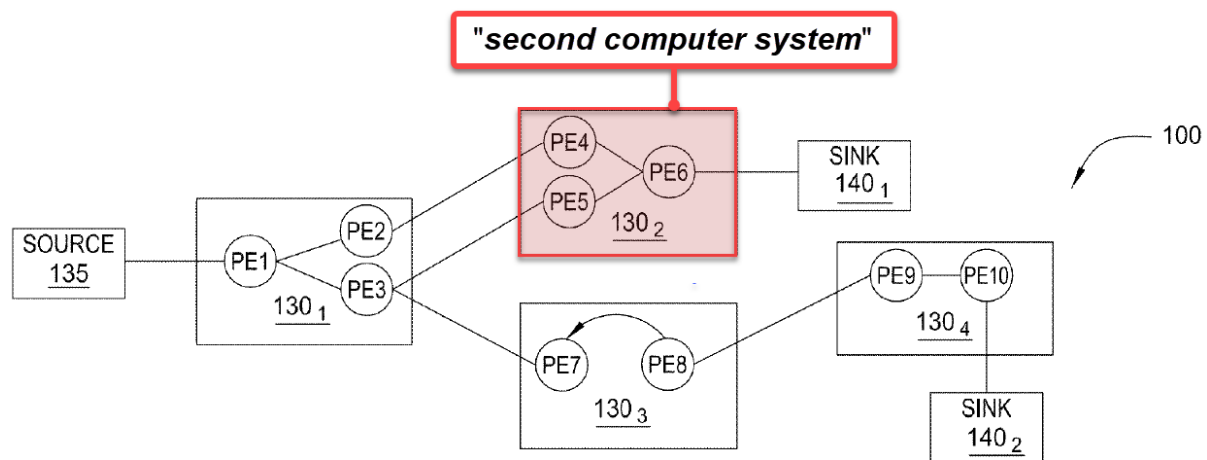


FIG. 1B

EX1004, FIG. 1B (annotations added). Alternatively, the Barsness third compute node 130₃ is an alternative “*second computer system*” with a real-time

transformation pipeline of PE7 and PE8 that transmits output messages to PE9, for the same reasons explained below for compute node 130₂. EX1003, ¶90.

(I) [1.4.a]

Barsness satisfies this element. Barsness discloses the second compute node 130₂ (*second computer system*) *receiv[ing]* the data emitted by PEs of the first compute node 130₁ (*first pipeline output messages*). EX1004, 7:58-61, FIG. 1B. For example, as discussed *supra* Section VII.A.2.e), the operator graph of Barsness shows a flow of output data (*first pipeline output messages*) from the *first transformation pipeline* to inputs (PE4 and PE5) of the *second transformation pipeline* in the compute node 130₂ depicted in FIG. 1B:

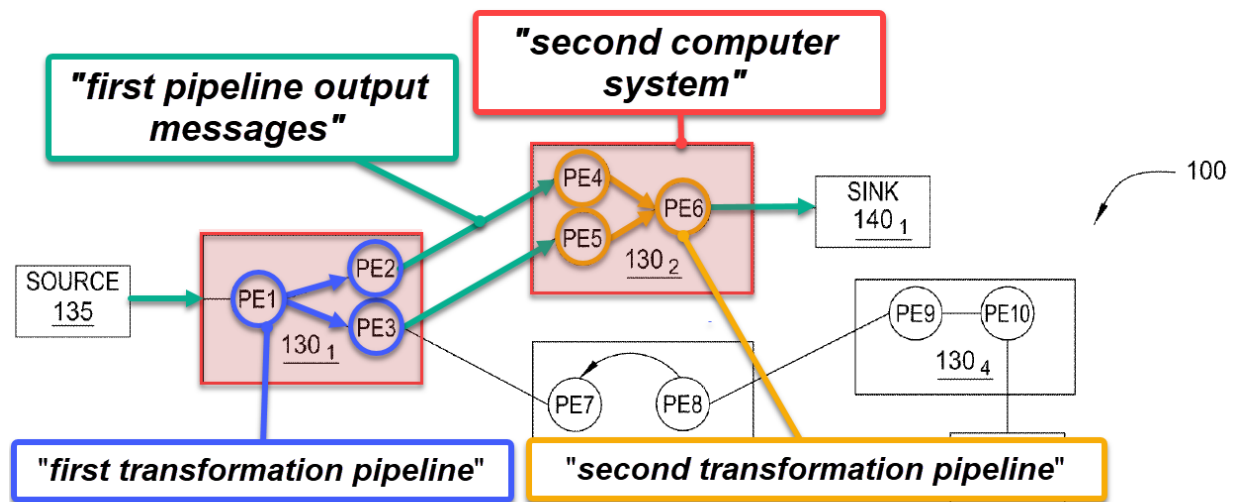


FIG. 1B

EX1004, FIG. 1B (annotations added). Barsness further describes “[d]ata that flows to PE2 results in tuples emitted to PE4 on compute node 130₂,” and similarly “data tuples flowing from PE3 to PE5,” where PE5 is also on compute node 130₂. EX1004,

7:58-61. Barsness provides examples of PEs transmitting messages “using a network socket (e.g., a TCP/IP socket).” EX1004, 7:44-47; EX1003, ¶91.

(m) [1.4.b]

Barsness satisfies this element. For example, in FIG. 1B, the second compute node 130₂ (*second computer system*) processes the output data from the first compute node 130₁ (*first pipeline output messages*) by executing the operators / executable code (*executing software instructions*) associated with PEs PE4, PE5, and PE6 (*second transformation pipeline*) to generate output data (*second pipeline output messages*):

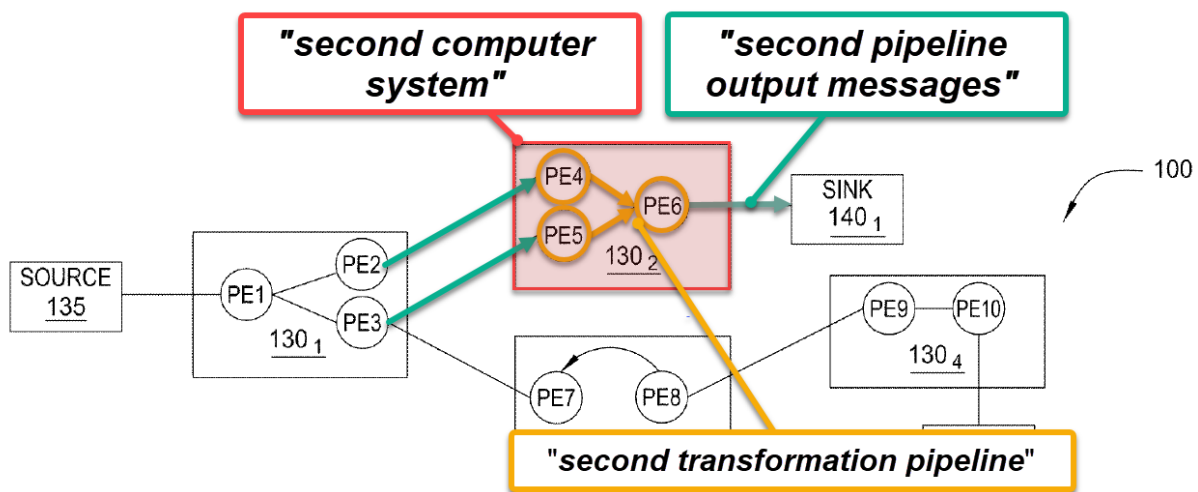


FIG. 1B

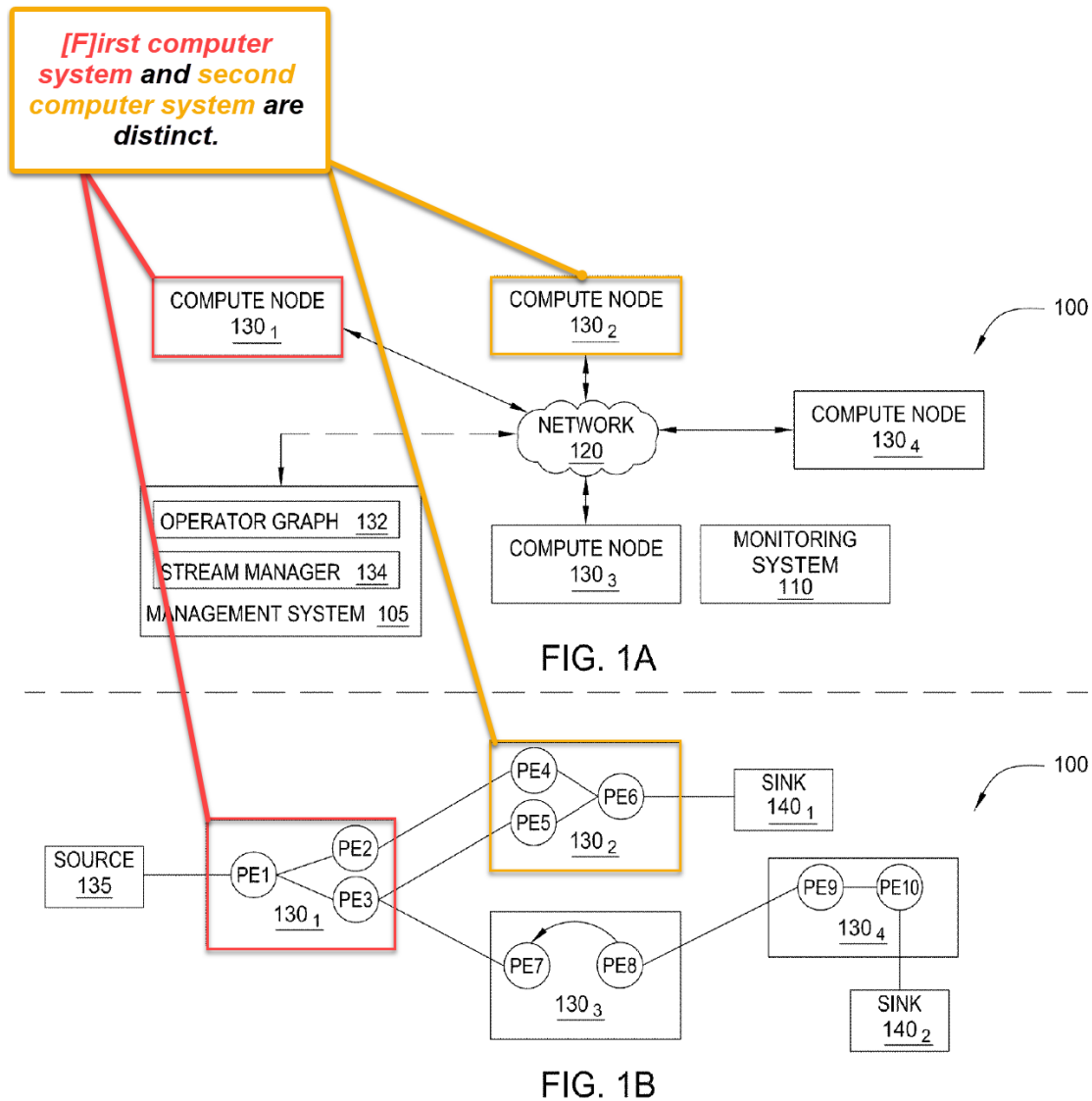
EX1004, FIG. 1B (highlighting added); *see also id.*, 7:52-67, 8:4-6. Barsness discloses PE4, PE5, and PE6 (*second transformation pipeline*) processing data tuples and Barsness discloses that each PE “may be configured to transform incoming data tuples.” EX1004, 7:58-61; *id.*, 8:11-15. Thus, in the FIG. 1B example,

PE6 (and thus the *second transformation pipeline*) generates output data that are *second pipeline output messages* because PE6 is the last PE in the second transformation pipeline, and so its output messages are output messages of the second pipeline. EX1003, ¶92. And compute node 130₃, an alternative “second computing system” with transformation pipeline PE7 and PE8, also generates output messages to PE9 of compute node 130₄. *Id.*

The PEs within a given compute node apply a transformation pipeline by processing the stream of input data. EX1004, 8:11-15. This occurs *in real time*, as explained with respect to the PEs of compute node 130₁ (*first transformation pipeline*) and limitation [1.3.b]. *Supra* Section VII.A.3.h; EX1003, ¶93.

(n) [1.5]

Barsness satisfies this element. The compute nodes of Barsness, including the compute node 130₁ (*first computer system*) and compute node 130₂ (*second computer system*) are shown as *distinct* computing systems (as is the third compute node 130₃):



EX1004, FIG. 1A, 1B (annotations added); *see also id.*, 7:52-67, 8:4-6. Thus, in the FIG. 1B example, PE6 (and thus the *second transformation pipeline*) generates output data that are *second pipeline output messages* because PE6 is the last PE in the EX1004, FIG. 1B (annotations added). When implemented as a virtual machine, each compute node “runs as an independent computer system.” EX1004, 8:43-44; EX1003, ¶94. Barsness further teaches moving PEs to “separate machines,” further

confirming that the compute nodes are distinct machines. EX1004, 3:56-60; EX1003, ¶95.

(o) [1.6]

Barsness satisfies this element. Barsness discloses that its management system is configured to add additional compute nodes to improve performance of the stream application. EX1004, 14:44-56; *id.*, 14:26-39, 13:12-14, FIG. 6. Barsness thus contemplates additional, unconfigured compute nodes / computer systems that are available to be configured and assigned by the management system when needed. These available or added compute nodes (or both) make up *a second plurality*.⁴ EX1003, ¶96.

This is reinforced by the Barsness teachings on a “cloud computing infrastructure” where “the nodes used to create a stream application may be virtual machines.” EX1004, 6:16-42. This infrastructure includes a “shared pool of configurable computing resources” that “can be rapidly provisioned and released.” EX1004, 6:23-27. These additional virtual machines are available to be provisioned

⁴ The claim recites “*second plurality of computer systems*” without further restriction, suggesting that any group of two or more computer systems satisfies it. EX1003, ¶96.

as compute nodes when needed and provide another example of *a second plurality of computer systems*. EX1004, 6:24-26; EX1003, ¶97.

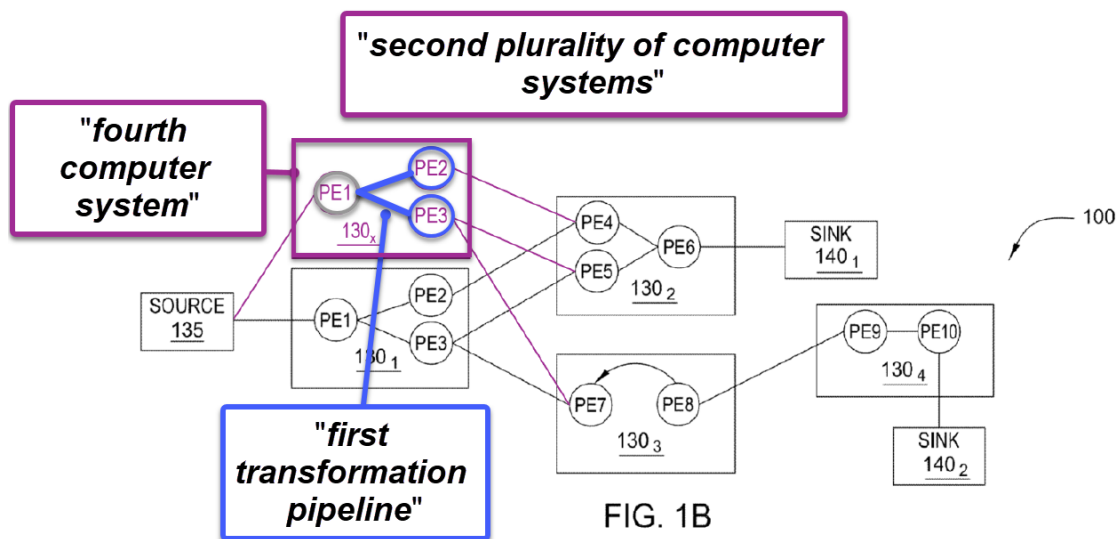
Specifically, Barsness discloses that each compute node “may be a physical host or a virtual machine” and that each “virtual machine runs as an *independent computer system*.” EX1004, 8:37-44 (emphasis added). Barsness further teaches that the “processing nodes and the management server may be hosted in a virtualized computing environment to provide ... another *layer of scalability* and fault tolerance.” EX1004, 8:48-52 (emphasis added). Relatedly, Barsness discloses “spawn[ing] more process nodes 130” as needed. EX1004, 10:28-34; *id.*, 14:51-56, 8:45-48. These teachings confirm that Barsness contemplates an infrastructure with additional independent computer systems beyond the initially assigned compute nodes, and these additional systems make up *a second plurality of computer systems*. EX1004, 8:43-44, 8:48-52. EX1003, ¶98.

Barsness also discloses that each compute node 130 may be a “physical host,” so the teachings regarding adding compute nodes 130 apply equally for a physical embodiment. EX1004, 8:30-39; EX1003, ¶99.

(p) [1.7]

Barsness satisfies this element. Barsness teaches the management system 105 (*third computer system*) configured to provision and configure an additional compute node (*fourth computer system*) and assign to it PEs executed on another

compute node, thereby causing that additional compute node to execute software instructions applying at least one of the claimed transformation pipelines (*first* or *second transformation pipeline* or both). For example, if the monitoring system 110 detects performance issues associated with compute node 130₁, Barsness teaches that management system 105 provisions an additional compute node (e.g., compute node 130_x) and assigns it the same processing elements used to implement *the first transformation pipeline*, such that the new compute node begins executing that pipeline as well:



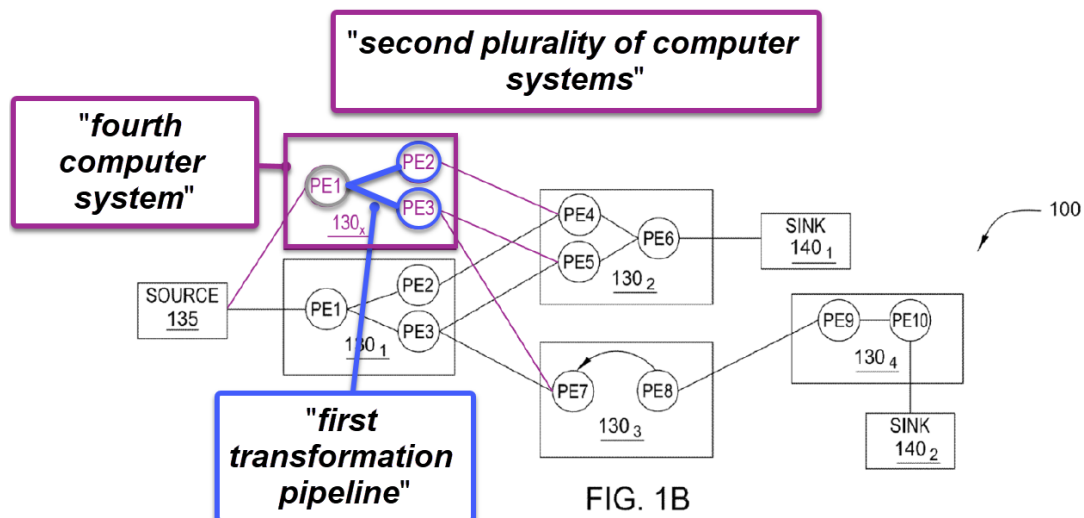
EX1004, FIG. 1B (annotations added). The management system thus duplicates or migrates the PEs from an existing compute node (e.g., 130₁) onto a newly provisioned compute node (e.g., 130_x). EX1003, ¶100.

The management system runs a node manager “to configure the compute nodes,” e.g., as part of “assigning or reassigning” PEs “on different compute nodes.” EX1004, 10:14-18; *id.*, 10:18-28; *see also id.*, 9:60-11:10. Regarding virtual

machine implementation of its system, Barsness similarly describes the ability to “spawn more virtual machines on the fly” and thus provide “well known advantages” “including another layer of scalability and fault tolerance.” *Id.*, 8:43-52 (emphasis added); *see also id.*, 7:45-48; EX1003, ¶102.

Barsness specifically teaches that its management system responds to performance issues by provisioning additional compute nodes and assigning or reassigning processing elements among those nodes. For example, the management system will “spawn more process nodes 130” in response to compute node performance issues (EX1004, 10:28-34) and “move processing elements 235 from one computer node 130 to another to improve the data processing performance of the stream application” (*id.*, 10:34-39). Barsness further teaches using processing elements “in parallel” to process the same type of data (*id.*, 10:26-28), which is exactly the kind of parallel execution that results when an added compute node is assigned the same processing elements as an existing node. Barsness further explains that these corrective actions include duplicating compute-node functionality and spawning new compute nodes to maintain expected performance levels. *Id.*, 14:30-57. For example, if the “processing rate is above or below” a pre-determined level, the system takes “corrective actions” that include “spawning new compute nodes” and “duplicating a compute node.” *Id.*; *see also id.*, FIG. 6, 15:38-41 (claim 2), 15:49-57 (claim 5); EX1003, ¶101.

Applied to its Figure 1B example, Barsness teaches its management system will provision a new compute node 130_x (*fourth computer system*) with processing elements PE1, PE2, and PE3 (*first transformation pipeline*) to address any “performance issues” that arise with compute node 130₁. EX1004, 10:30-34; *see also id.*, 10:34-39. The duplicate/added node then processes the same type of stream data in parallel, “increas[ing] data throughput” (*id.*, 10:26-28) and maintaining the desired “processing rate” for the stream application (*id.*, 14:54-56, 14:30-39). The following annotated FIG. 1B shows duplicated / provisioned compute node 130_x (*fourth computer system*) configured with the processing elements of the first compute node 130₁ (*first transformation pipeline*), as taught by Barsness:



EX1004, FIG. 1B (annotations added); EX1003, ¶103.

The same Barsness teachings apply to other transformation pipelines, such as the *second transformation pipeline*: if performance issues arise for compute node 130₂, the management system will provision another compute node (compute node

130_x) and configure it with the processing elements that implement the second transformation pipeline (of the second compute node 130₂). EX1004, 10:34-39; *id.*, 10:28-34, 14:51-56 (describing automatically “duplicating a compute node”). Such a provisioned compute node 130_x (*a fourth computer system*) configured with the processing elements of the second compute node 130₂ (*the second transformation pipeline*) is shown in this annotated FIG. 1B:

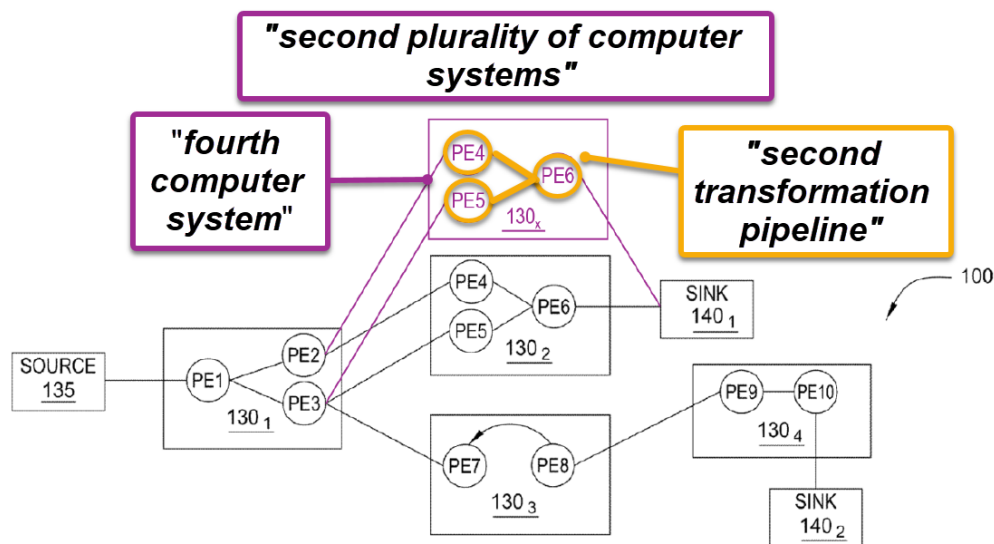


FIG. 1B

EX1004, FIG. 1B (annotations added); EX1003, ¶104. Ground 3, below, further reinforces this implementation of Barsness with Siripurapu’s express teachings on adding computer nodes to handle additional processing needs. *Infra* Section VII.C.

Further, while Barsness discloses “spawning new compute nodes” and “duplicating a compute node” generally (EX1004, 14:51-56), a POSITA would have found it obvious to duplicate the compute nodes 130₁ and 130₂ in particular because Barsness discloses its management system being configured to duplicate any

compute node in the context of corrective measures for improving data processing throughput. EX1004, 10:26-39, 14:26-56; EX1003, ¶105. The analysis for claims 3 through 6 (*infra* Sections VII.A.4-VII.A.7) provides further examples, further evidencing the obviousness of claim 1.

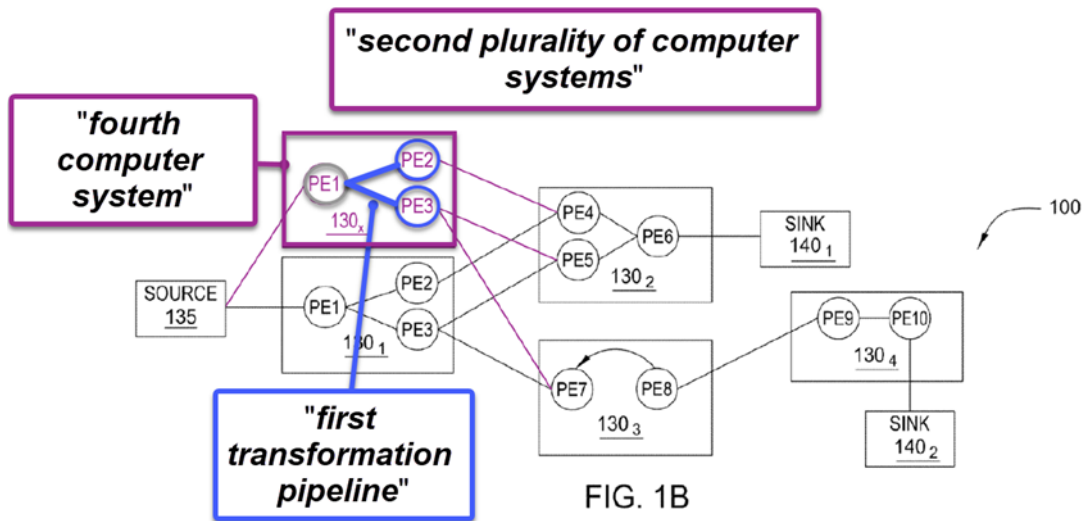
Each of these newly spawned, duplicate compute nodes executes software instructions that *apply at least one of the first transformation pipeline and the second transformation pipeline*, as carried out by the respective PEs. *Supra* Section VII.A.2.e. Each such added compute node is part of the claimed second plurality—i.e., the set of additional computer systems available to be provisioned and assigned by the management system beyond the compute nodes already executing the recited pipelines. *Supra* Section VII.A.2.o. EX1003, ¶106.

Finally, the management system is a *computer system of the first plurality of computer systems* as discussed *supra* Section VII.A.2.b. EX1003, ¶106.

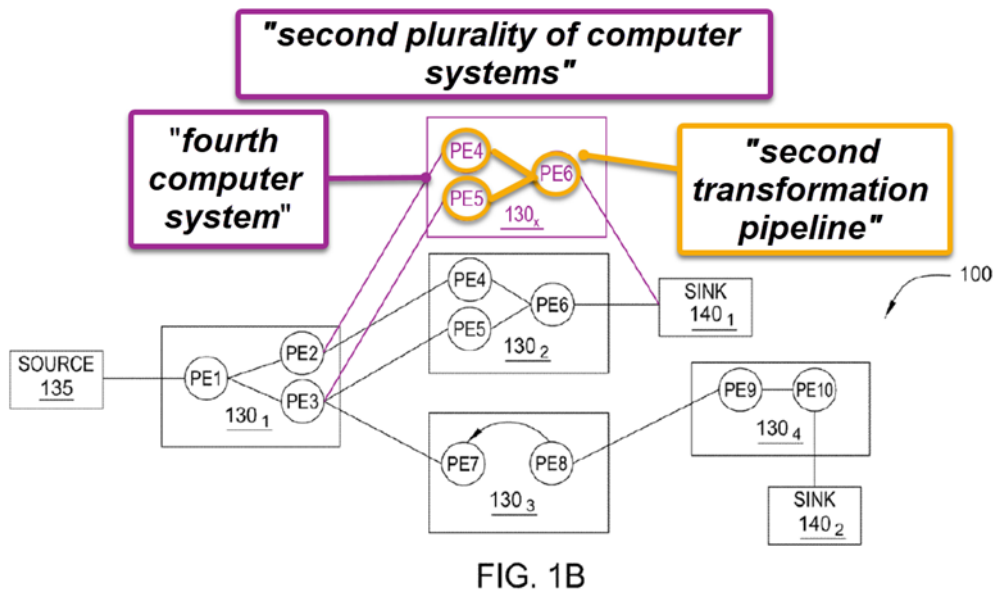
Alternatively, Barsness teaches monitoring system 110 (*third computer system*) *cause[s]* the provisioning of compute node 130_x (*fourth computer system*) by “instruct[ing] the management system 105 to spawn more process nodes 130 to process certain type of data when the monitoring system 110 determines a need based on a detection of performance issues with a computer node 130 or the processing elements 235 in a computer node 130.” EX1004, 10:28-34; *see also id.*, 10:34-39. Barsness also teaches monitoring system 110 configuring additional

compute nodes. EX1004, 11:37-41 (“The monitoring application may ... configure the compute nodes 130 with respect to assigning or reassigning processing elements on different compute nodes 130, spawning new processing elements 235, operators 240, etc.”).

Thus, if monitoring system 110 detects performance issues for compute node 130₁ or 130₂, the monitoring system 110 *cause[s]* management system 105 to provision compute node 130_x and configures it with the PEs of compute node 130₁ (*first transformation pipeline*) or 130₂ (*second transformation pipeline*). EX1004, 10:28-39; *see also id.*, 12:45-13:3, 13:12-14, 14:51-56. The preceding explanation regarding the management system’s spawning and duplicating additional compute nodes applies here as well with the monitoring system instructing the management system. Thus, compute node 130_x processes the same type of stream data in parallel, “increas[ing] data throughput” (*id.*, 10:18-28) and maintaining the desired “processing rate” for those PEs (*id.*, 14:54-56, 14:30-39), either in parallel with compute node 130₁ or compute node 130₂ as seen in the figures below:



EX1004, FIG. 1B (annotations added); EX1003, ¶107.



EX1004, FIG. 1B (annotations added); EX1003, ¶107.

A POSITA would have found it obvious to duplicate compute nodes 130₁ and 130₂ because Barsness discloses monitoring system 110 being configured to interact with management system 105 to duplicate any compute node 130 in the context of corrective measures for improving the data processing rates measured by monitoring

system 110. EX1004, 10:26-39, 14:26-56; EX1003, ¶107. Barsness explains that data throughput is determined by “tuple processing rate,” *i.e.* how many tuples from the data stream are processed per second. EX1004, 12:40-54. Thus, in order to increase the tuple processing rate, the duplicated compute nodes (*i.e.* 130_x) process different tuples within the incoming data stream than the compute nodes they duplicated (*i.e.* 130₁ and 130₂). EX1003, ¶107.

As discussed above, monitoring system 110 is a ***computer system of the first plurality of computer systems***. *Supra* Section VII.A.2.b. EX1003, ¶107.

3. Claim 2

Barsness renders this claim obvious. Barsness discloses non-linear transformation pipelines, including in FIG. 1B, where branching occurs in both the ***first transformation pipeline*** in the first compute node 130₁ and the ***second transformation pipeline*** in the second compute node 130₂:

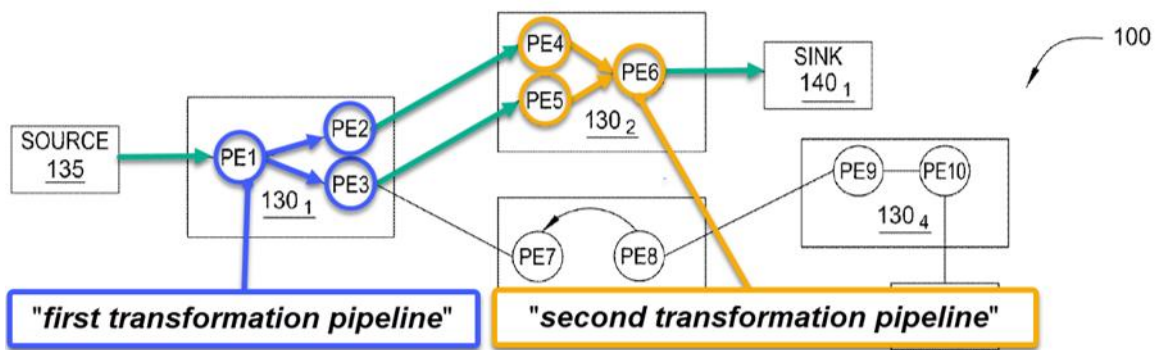


FIG. 1B

EX1004, FIG. 1B (annotations added). Here, the operator graph shows efferent (fan-out) branching from PE1 to PE2 and PE3 and afferent (fan-in) branching from PE4

and PE5 to PE6. EX1004, 7:55-61, FIG. 1B. The operator graph also shows a third transformation pipeline with processing elements PE7 and PE8 in compute node 1303, with a non-linear “looping” or cyclical flow of data. EX1004, 7:64-67, FIG. 1B; *supra* Section VII.A.2 (claim 1). EX1003, ¶108.

Barsness thus satisfies claim 2, as it teaches use of “afferent branch,” “efferent branch,” and “cyclical” transformation pipelines, and the ’424 patent itself identifies these types of pipelines as non-linear. EX1001, 10:4-7, FIGS. 13-14; EX1003, ¶109.

4. Claim 3

Barsness renders claim 3 obvious. As explained above for element 1.7 (*supra* Section VII.A.2.p), Barsness discloses a management system (***third computer system***) that identifies additional unconfigured compute nodes and provisions those compute nodes to, for example, duplicate existing compute nodes. EX1004, 10:34-39; *id.*, 10:30-34; *id.*, 8:43-52, FIG. 6, 14:30-56. As explained in the following paragraphs, the Barsness teachings on node provisioning, scalability, and multiple input sources render obvious a management system configured to satisfy claim 3. EX1003, ¶110.

Barsness teaches multiple input data streams and scaling the system as needed to handle incoming data. For example, its operator graph represents “a stream application beginning from *one or more* source processing elements (PEs)” and is “configurable to *add or remove one or more PEs* based on the data processing needs

and *scalability*.” EX1004, 6:49-55 (emphasis added); *id.*, 2:52 (“data sources”).

Barsness further teaches parallel processing of the “same type of data to increase data throughput.” EX1004, 10:26-28; EX1003, ¶111. These contemplated additional input source streams of the same data type originating from different input feeds (e.g., from different locations), predictably increase the number of concurrent inputs the system must process. *E.g.*, EX1008,⁵ 238-241 (discussing example data streams handled by stream processing engines); EX1003, ¶114.

The management system of Barsness addresses such increased “data processing needs” (which arise with an additional input stream) by identifying available resources and provisioning additional compute nodes. EX1004, 6:49-55; *id.*, 3:19-22, 3:61-63, 6:16-42, 8:45-52; 10:26-34, 11:30-34, 14:51-56. Further, POSITAs would expect scenarios where the stream application handles multiple input data streams (e.g., with the same type of data, originating from different locations) and scales accordingly, especially when using a “cloud computing infrastructure,” as taught by Barsness. EX1004, 6:16-42; *id.*, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42; *see also* EX1008, 238-241 (“Cloud-based data streaming engines, in particular, are designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads.”); EX1007, [0243]; EX1003,

⁵ Petitioner relies on EX1008 to show the state of the art.

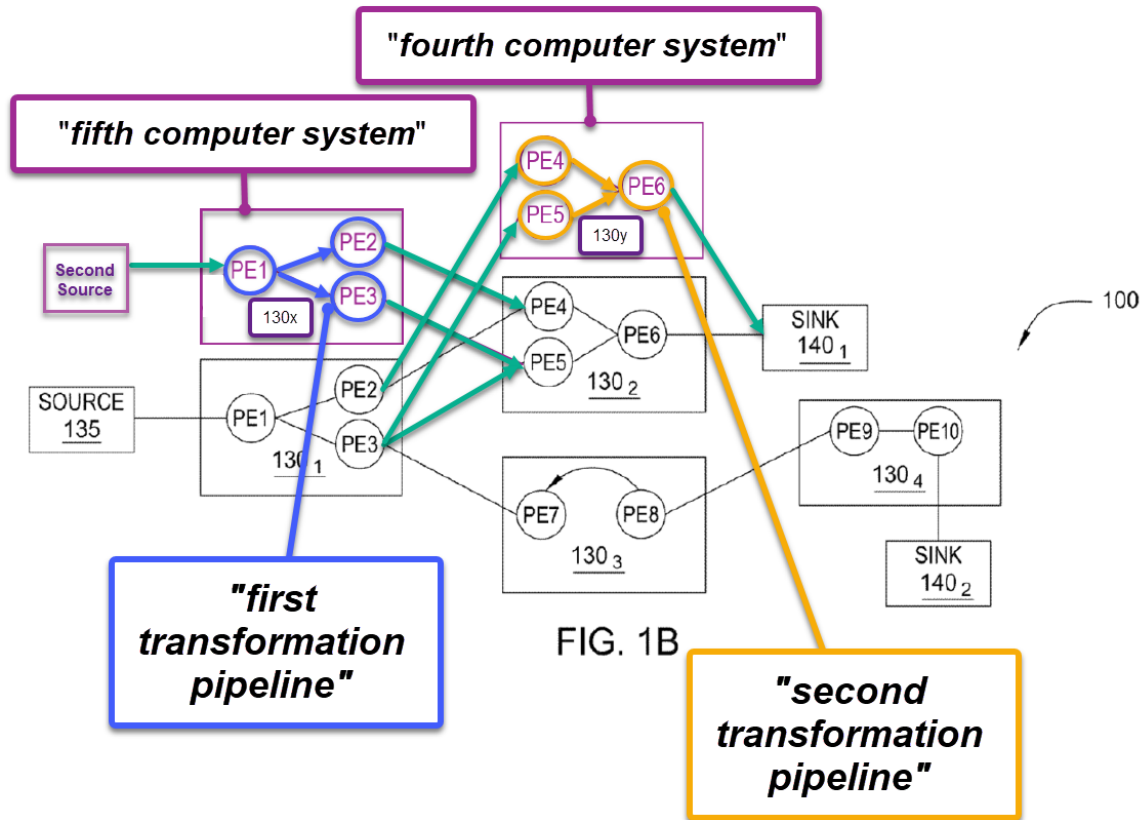
¶¶112-114. To the extent PO argues otherwise, Ground 3 further relies on Siripurapu, which expressly teaches scaling to handle processing additional real-time data streams and increased data flow. *Infra* Section VII.C.

Barsness thus teaches operator graphs with multiple input data streams and scaling the system as needed to handle incoming data. A POSITA therefore would have understood Barsness to teach, or at least render obvious, that when a second input stream is introduced, the management system 105 would provision and configure an available compute node with processing elements for the first transformation pipeline (*identify a fifth computer system*), and assign that provisioned node to process the second data stream (*cause the fifth computer system to process a second stream*). The management system 105 (*third computer system*) performs these actions because it manages the compute nodes and thus knows what compute nodes are available, and handles configuring compute nodes to execute PEs. *E.g.*, EX1004, 7:27-37, 10:12-64, 11:30-34, 14:51-56; EX1003, ¶¶114-116. For the same reasons discussed for the first compute node, the provisioned compute node processes the second data stream in real-time by executing software instructions that apply the *first transformation pipeline* to the second data stream. *Supra* Section VII.A.2.h (discussing applying *first transformation pipeline*).

A POSITA would have been motivated to implement the computing architecture in this way to beneficially provide system scaling, as needed, to handle

data processing needs, as taught by Barsness, as well as to simplify state management of a data stream, where a duplicated stream application allows the new data stream to be processed in real-time without the additional complexity of a single compute node managing processing of multiple data streams. EX1003, ¶117. That POSITAs would have easily and predictably implemented such a system is confirmed by the '424 patent's own failure to describe a computer system causing another computer system to apply an existing transformation pipeline to a second stream of input data. *In re Fox*, 471 F.2d 1405, 1407 (C.C.P.A. 1973). Further, a POSITA would have reasonably expected success with such an implementation because Barsness explains how to do so, with its teachings on provisioning compute nodes. *E.g.*, EX1004, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42; EX1003, ¶117.

The following annotated Barsness FIG. 1B shows an example of the management system 105 scaling compute nodes to handle a new input stream, with compute node 130_x (***fifth computer system***) configured to apply the processing elements PE1, PE2, and PE3 (***first transformation pipeline***) to a second source (***a second stream of input data received from a second input feed***):



EX1004, FIG. 1B (annotations added). Though not depicted, a POSITA would have also been motivated with a reasonable expectation of success to configure the management system to scale the entire streaming application, i.e., duplicate all compute nodes, to handle the increased processing needs of the second source. *E.g.*, *id.*, 7:29-37. This would result in duplication of third and fourth compute nodes 130₃ and 130₄ with PEs 7-10. For the purposes of claim 3, **the first plurality of computer systems** comprises, in addition to the management system 105 and the compute nodes 130₁₋₂ as described *supra* Section VII.A.2.b, these additional compute nodes, including compute node 130_x (**fifth computer system**) and compute node 130_y, provisioned for implementing an additional instance of the stream application. The

analysis of elements 1.2 discussed *supra* Section VII.A.2.b applies to these additional compute nodes 130_x and 130_y for the same reason it applies to compute nodes 130₁₋₄. EX1003, ¶118.

5. Claim 4

(a) [4.1]

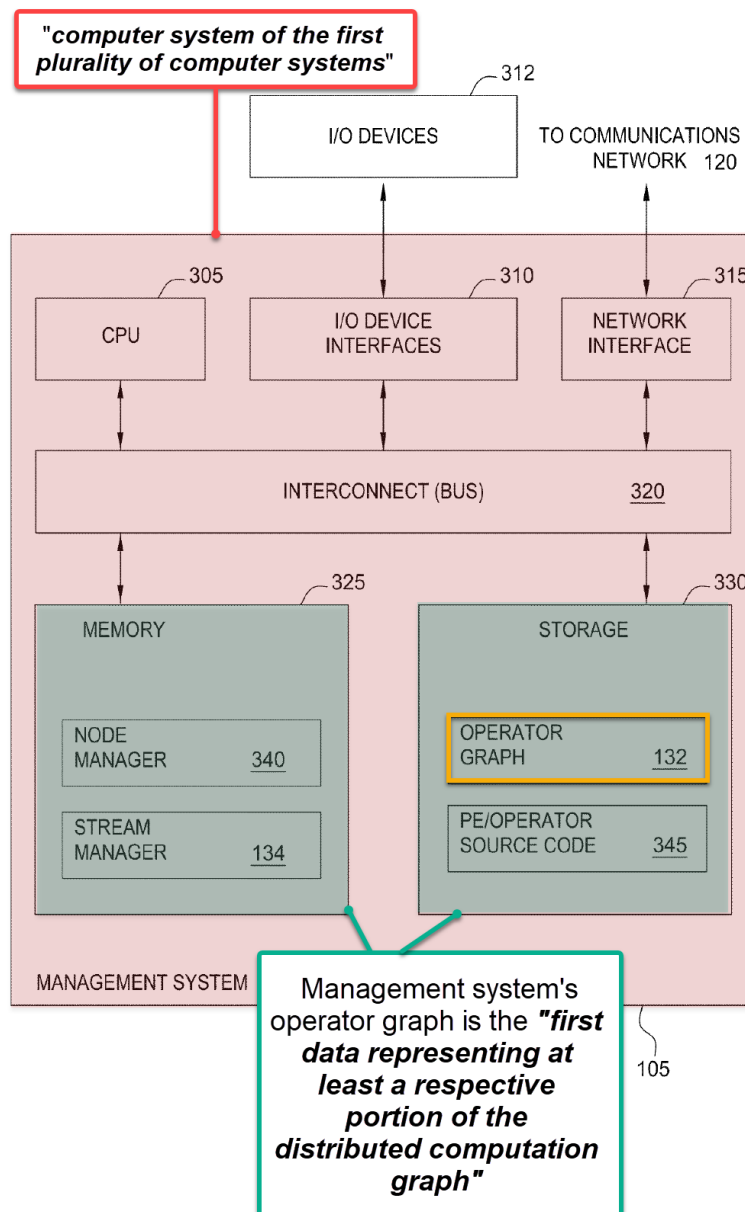
The management system (*third computer system*) stores in memory the operator graph of the streaming application. The operator graph represents the *distributed computational graph* of a streaming application and includes information regarding the compute nodes, processing elements, and operators. Thus, the management system's operator graph includes information representing the compute nodes, processing elements, and operators of the *first transformation pipeline*. The management system's node manager and stream manager both maintain the operator graph by adding, removing, or otherwise editing it with up-to-date information regarding the streaming application. When the management system spawns or duplicates new compute nodes, those additional compute nodes and their processing elements and operators are added to the operator graph. EX1003, ¶119.

The management system (*third computer system*) includes a node manager 340 and a stream manager 134 (*execute software instructions*) that “configure the compute nodes 130,” and “monitor the data stream as it flows through the processing

elements in a stream application.” EX1004, 10:12-20, 10:40-48, EX1003, ¶120; *see also supra* Section VII.A.2.p (regarding limitation [1.7]).

(b) [4.2]

Management system 105 (*third computer system*) stores in *memory* an operator graph 132 (*first data representing at least a respective portion of the distributed computational graph*). EX1004, 9:62-67, 10:3-11.



EX1004, FIG. 3 (annotations added); *id.*, 10:12-64; EX1003, ¶121.

Barsness discloses that the operator graph (*distributed computational graph*) “represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data tuples through the stream application.” EX1004, 1:43-46; *id.*, 2:12-15, 2:59-62, 6:43-56, 10:49-51, FIGS. 1A- 1B; *Supra* Section VII.A.2.d; EX1003, ¶122. FIG. 1B depicts an example operator graph, where the processing elements are graph nodes and the links between processing elements are graph edges:

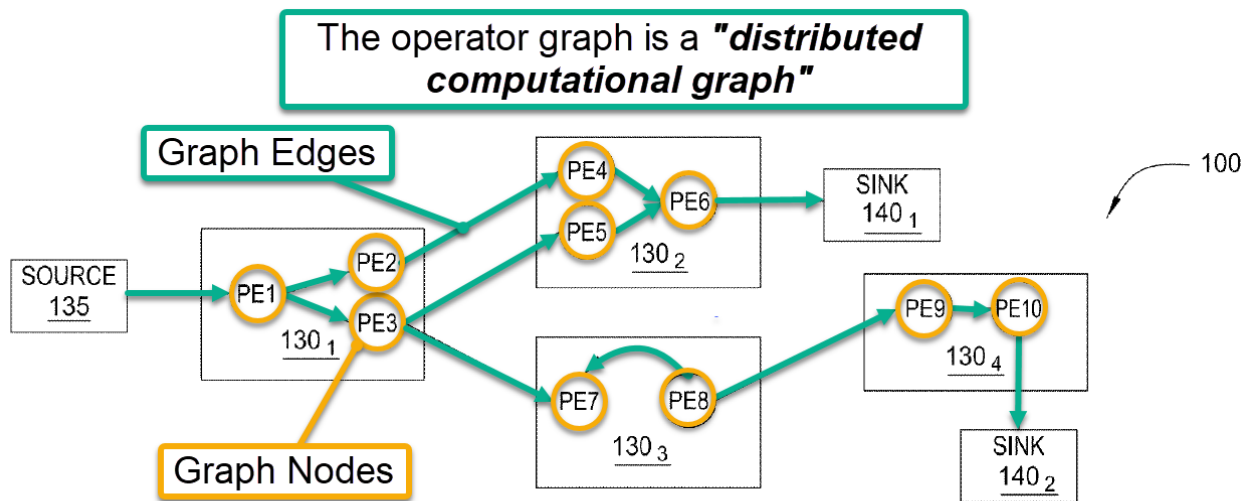


FIG. 1B

EX1004, FIG. 1B (annotations added).

As explained above regarding limitation [1.2.a], the operator graph of Barsness is a *distributed computational graph* under Petitioner’s construction or PO’s “plain and ordinary meaning.” *Supra* Section VII.A.2.e.

Barsness discloses that the stream manager 134 “control[s] a stream application running on the compute nodes 130₁₋₄” and “change[s] the structure of the operator graph 132.” EX1004, 7:27-37. Thus, the management system’s stream manager 134 uses the operator graph stored in memory on the management system to “control the stream application by inserting, removing, or otherwise modifying the processing elements” (*identify one or more of the respective memories*). EX1004, 7:33-37. Additionally, Barsness discloses the node manager 340 “setting up one or more processing elements and operators in the compute nodes 130.” EX1004, 10:18-20. Thus, both the stream manager 134 and node manager 340 *store ... data* of the operator graph. EX1003, ¶122.

(c) [4.2.a]

As explained above regarding limitation [1.2.c], the management system’s operator graph describes the flow of data through several compute nodes, each containing a group of PEs that process and transform the data flowing through them. EX1004, 10:49-51; *e.g.*, EX1004, 7:52-8:21; *supra* Section VII.A.2.e. Thus, as highlighted in Fig. 1B below, the operator graph describes the compute nodes and PEs for the *first transformation pipeline*.

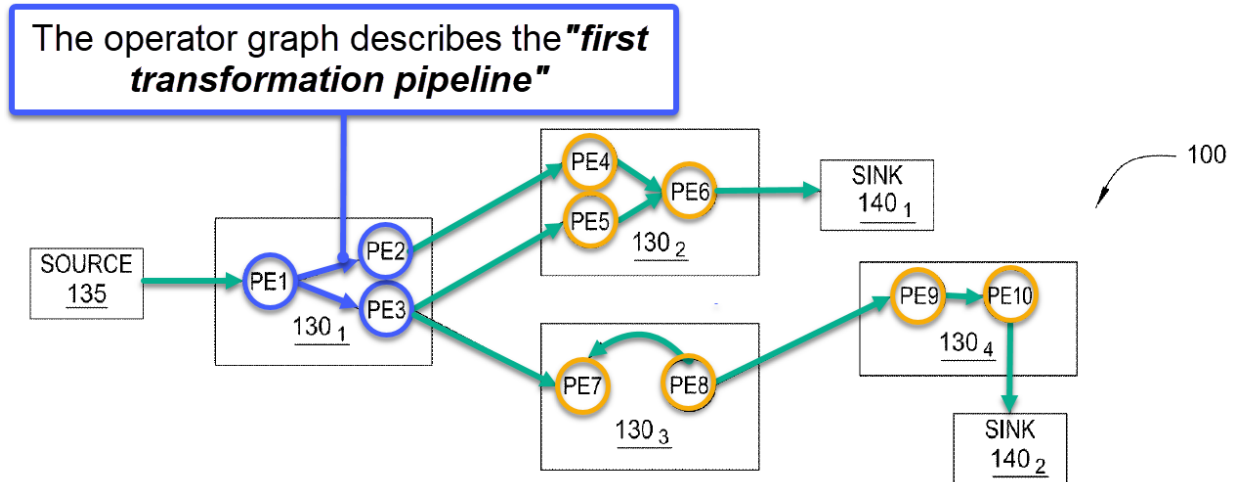


FIG. 1B

EX1004, FIG. 1B (annotations added); *see also id.*, 7:52-8:21. EX1003, ¶123.

(d) [4.3]

Barsness discloses that the management system's in-memory operator graph includes information for each compute node and its PEs including the newly spawned or duplicated compute nodes 130_x and 130_y. EX1004, 10:49-51, 10:57-64. EX1003, ¶124.

As explained above regarding limitations [1.7] and claim 3, the management system spawns or duplicates compute nodes (e.g. compute node 130_x) to perform parallel processing of data flowing through the streaming application. Barsness discloses that the operator graph "maintains the connectivity configuration of a plurality of compute nodes and processing elements in a streaming application." EX1004, 10:49-51. Thus, newly spawned or duplicated compute nodes, e.g. 130_x and 130_y, (*fifth computer system*) are *identified* in the operator graph. Thus, when

the management system spawns or duplicates compute nodes, the stream manager 134 and/or node manager 340 “change[s] the structure of the operator graph 132.” EX1004, 7:27-29, 10:12-20.

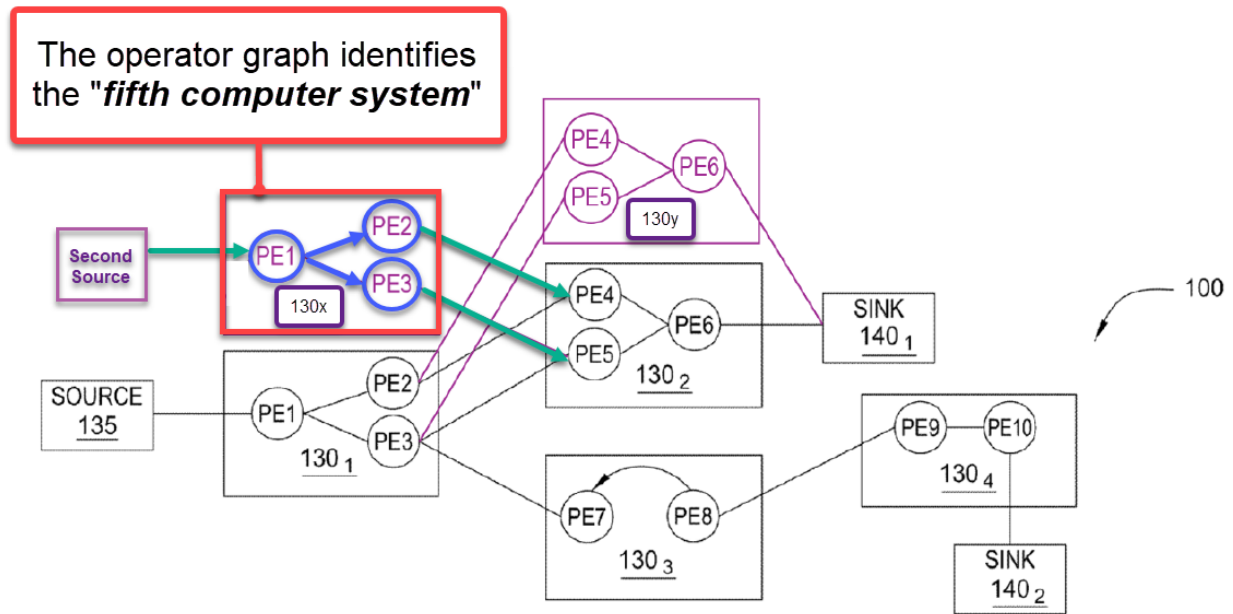


FIG. 1B

EX1004, FIG. 1B (annotations added).

A POSITA would have recognized that the operator graph is updated to maintain information identifying the *fifth computer system*. EX1003, ¶125. The operator graph allows the management system 105 “to control a stream application running on the compute nodes 130₁₋₄ as well as change the structure of the operator graph 132.” EX1004, 7:27-29; *id.*, 6:48-56. Storing the operator graph 132 in the memory of management system 105 allows node manager 340 and stream manager 134 to build and configure the operator graph. EX1004, 7:27-37; *id.*, 10:12-64 (describing management of the operator graph by the management system);

EX1003, ¶125. Since the node manager 340 and stream manager 134 each require a complete operator graph to perform their functions, a POSITA would have known that the *fifth computer system* would be added to the operator graph when the compute node is added to the stream application.

6. Claim 5

Barsness renders obvious claim 5. As explained above for element 1.7 and claim 3 (*supra* Sections VII.A.2.p, VII.A.4), Barsness discloses a management system (*third computer system of the first plurality*) that manages the creation of new compute nodes and the moving / replication of processing elements among compute nodes “to improve the data processing performance of the stream application.” EX1004, 10:34-39; *id.*, 10:30-34; *id.*, 8:45-48; EX1003, ¶126.

As discussed above, a POSITA would have found it obvious to configure the Barsness management system to duplicate a streaming application’s compute nodes to accommodate a second stream of input data. *Supra* Section VII.A.2.p. This would result, for example, in compute node 130_x, applying the first transformation pipeline to a second stream of input data, and outputting its data messages (*third pipeline output messages*) to duplicated compute node 130_y (*fifth computer system*). *Id.* Compute node 130_y, in turn, applies the processing elements PE4, PE5, and PE6 (*second transformation pipeline*) to the output data from PE2 and PE3 of the compute node 130_x (*process the third pipeline output messages substantially in real*

time by executing software instructions that apply the second transformation pipeline to the third pipeline output messages) in order to process data from the

second source with the duplicated stream application, as shown in this annotated

FIG. 1B:

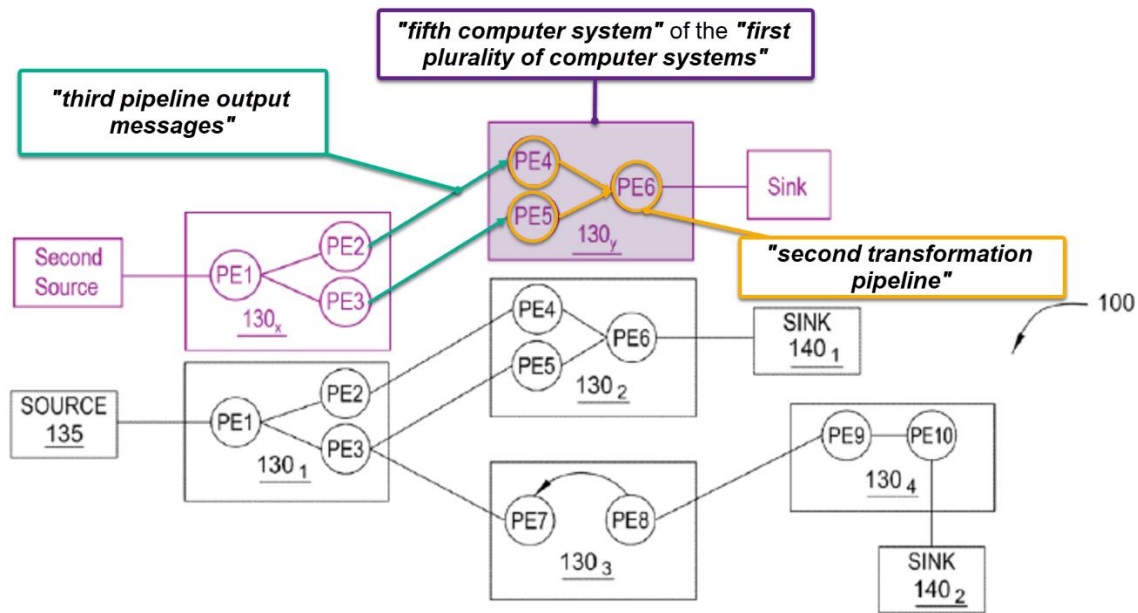


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 3:19-22; *supra* Section VII.A.2.p.

Alternatively, viewing third compute node 130₃ as the ***second transformation pipeline***, newly-spawned compute node 130_y duplicates processing elements PE7

and PE8 and thus duplicates the ***second transformation pipeline***. While only PE1

through PE6 are depicted in this example, a POSITA would have found it obvious

to duplicate the transformation pipelines of the compute nodes 130₃ and 130₄ as well

to complete the stream application as configured in the operator graph. EX1003,

¶127.

Barsness teaches—and renders obvious—configuring its management system (*third computer system*) to “identify” a *fifth computer system of the first plurality of computer systems* for the same reasons explained for claim 3. *Supra* Section VII.A.4; EX1003, ¶128.

7. Claim 6

Barsness satisfies this claim. As explained above regarding limitations [1.2]-[1.2.c], claim 3, and claim 4, the management system maintains the operator graph in its memory, the operator graph represents the *distributed computational graph*, the operator graph *includes information about* the compute nodes, processing elements, and operators of *the second transformation pipeline*, and the operator graph stores identifying information for newly spawned or duplicated compute nodes, e.g. compute node 130_x or 130_y (*fifth computer system*). EX1003, ¶129.

8. Claim 8

Barsness satisfies this claim. Barsness teaches that compute node 130₁ transmits data tuples to compute node 130₂ and compute node 130₃ (*another*

computer system of the second plurality of computer systems) as seen in Figure 1B.⁶

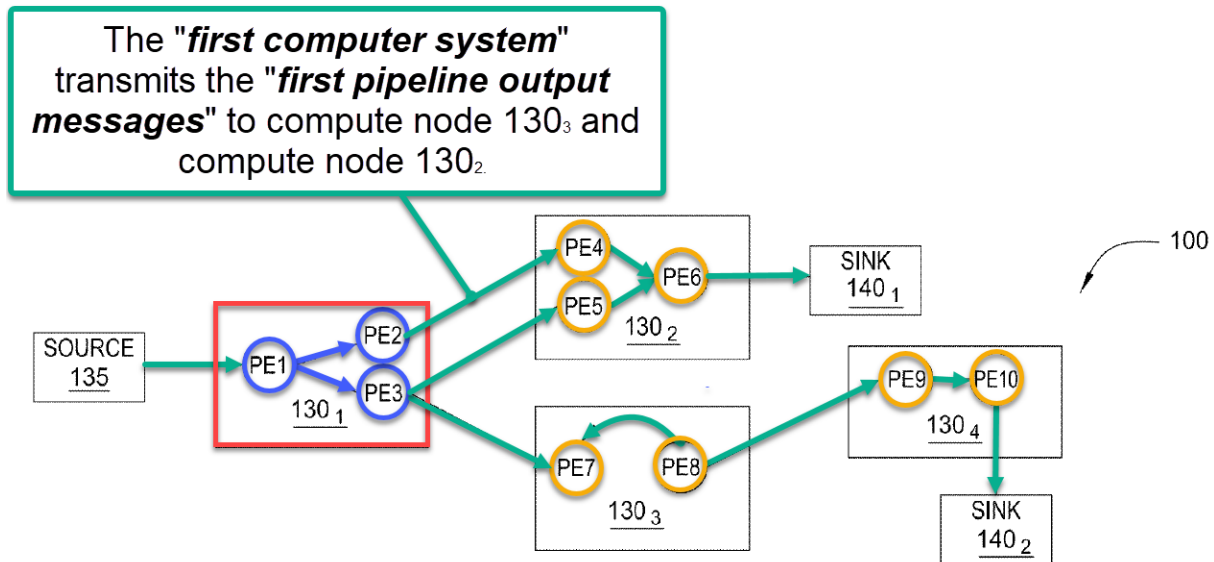


FIG. 1B

EX1004, FIG. 1B (annotations added); EX1003, ¶130.

Alternatively, it would have been obvious to a POSITA to provision and configure several additional compute nodes to meet the processing performance needs. EX1003, ¶131. As discussed regarding [1.7], Barsness teaches that monitoring system 110 detects performance issues and instructs the management system 105 to provision and configure additional compute nodes and assign PEs so

⁶ As explained above, the claim recites “*second plurality of computer systems*” without further restriction, suggesting that any group of two or more computer systems satisfies it including compute node 130₃. EX1003, ¶130.

that the additional compute nodes apply one of the claimed transformation pipelines to improve tuple processing rates. As discussed above, Barsness discloses management system 105 provisioning additional compute nodes such as in the annotated Figure 1B below where 130_x resolves performance issues detected on compute node 130_1 . See [1.7] and [3.a]. By provisioning more than one additional compute node to resolve performance issues identified on compute node 130_1 , each compute node 130_x and 130_y receives *the first pipeline output messages* from compute node 130_1 (*first computer system*). Furthermore, as discussed above, Barsness teaches management system 105 provisioning the additional compute nodes 130_x and 130_y from the *first plurality of computer systems* (see [1.7]’s *fourth computer system*) or from the *second plurality of computer systems* (see [3.a]’s *fifth computer system*).

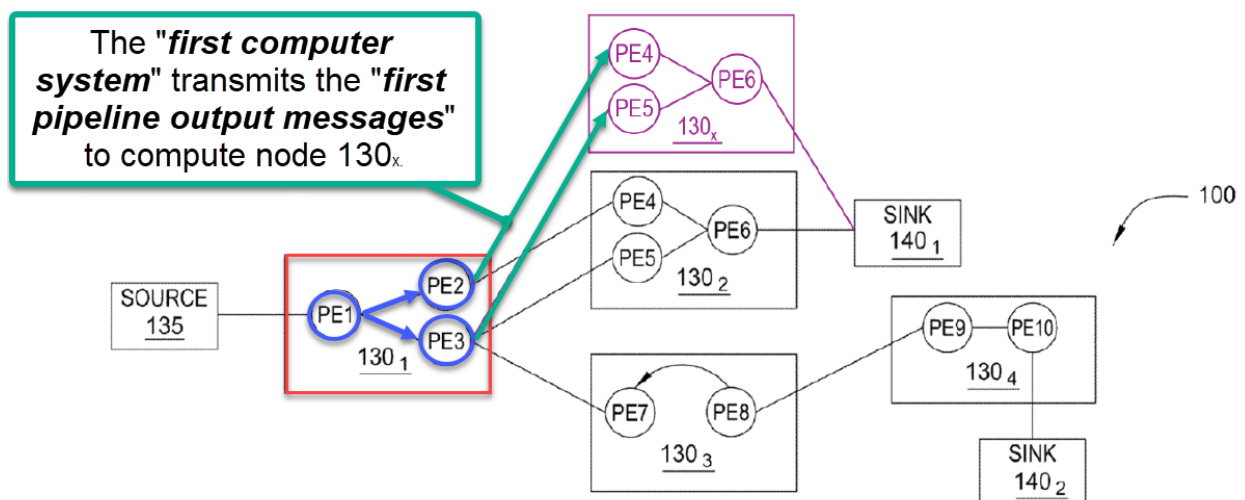


FIG. 1B

EX1004, FIG. 1B (annotations added). EX1003, ¶131.

9. Claim 9

(a) [9.1]

Supra Section VII.A.2.a (element 1.1); EX1003, ¶132.

(b) [9.2]

Supra Section VII.A.2.b (element 1.2). For example, with regard to Barsness FIG. 1A-B, *a first plurality of computer systems* comprises at least management system 105, monitoring system 110, and compute nodes 130₁ and 130₂. EX1004, 6:45-49.

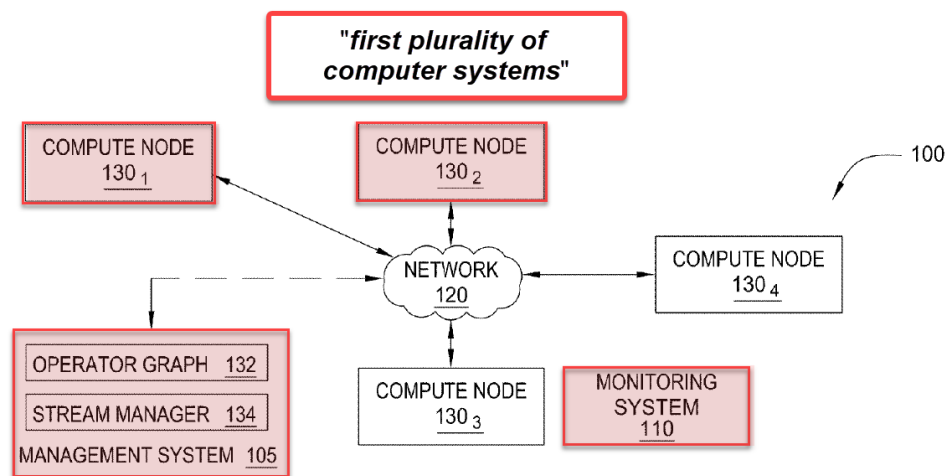


FIG. 1A

EX1004, FIG. 1A (annotations added). EX1003, ¶133.

(c) [9.3]

Barsness satisfies this element because each of the management system 105, monitoring system 110, and compute nodes 130 (*each respective computer system of the first plurality of computer systems*) comprises memory (*respective memory*) that stores respective programming instructions, application data, and other data

(*respective first data*). *Supra* Section VII.A.2.c (element 1.2.a); EX1003, ¶134. That data meets the additional recited claim language. *Supra* Section VII.A.2.d-e (elements 1.2.b-1.2.c). For example, the Barsness management system stores an operator graph that describes *a configuration of a first transformation pipeline* because the operator graph represents a stream application that flows from source PEs to sink PEs, and shows the interconnection of PEs of a compute node applying a *first transformation pipeline*, as shown in FIG. 1B:

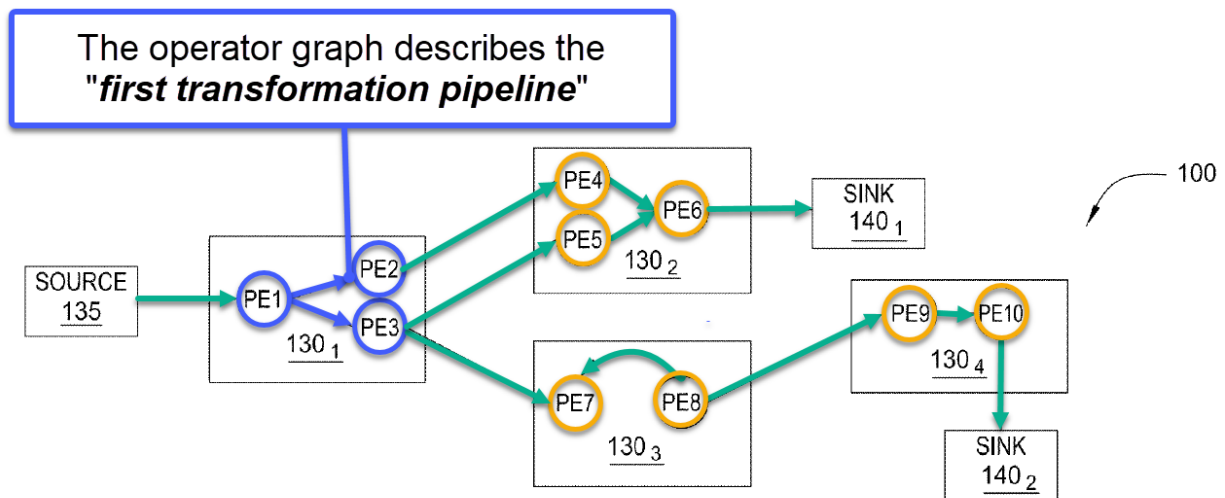


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 6:50-56, 7:52-67; *supra* Section VII.A.2.e. In the context of the FIG. 1B example, the management system memory stores operator graph 132 that includes data that *describes at least a respective portion of a configuration of a first transformation pipeline* because it describes the configuration of PE1, PE2, and PE3 (*first transformation pipeline*). EX1004, 10:1214, 10:48-56; *id.*, FIGS. 1B, 3; EX1003, ¶135.

Each compute node of the plurality (e.g., 130₁ and 130₂) stores “programming instructions,” “application data,” buffered stream data, and other data (*respective first data*) in its respective memory, including processing elements as designated by the operator graph. EX1004, 8:53-59, 8:66-9:33; FIG. 2. This “application data” (*respective first data*) specifies the connection to adjacent compute nodes (e.g., between compute node 130₁ and compute node 130₂), e.g., as the source and destination information of a TCP/IP socket, to transmit data between processing elements. EX1004, 9:4-9; EX1017, 1:22-2:26, FIG. 1; EX1016, 8:56-62. Each compute node thus stores data *describing at least a portion of a configuration of the transformation pipelines* that it receives data from and emits data to. EX1003, ¶136. The data stored on each compute node (e.g., compute node 130₁) also *describes at least a portion of a configuration of the transformation pipeline* on that particular compute node. *Supra* Section VII.A.2.d-VII.A.2.e. For example, compute node 130₁ stores the processing elements PE1, PE2, and PE3 comprising the *first transformation pipeline*. EX1003, ¶136.

Thus, in the FIG. 1B example, at least first compute node 130₁ and the compute nodes it emits data to (130₂ and 130₃) store in memory data that *describes at least a portion of a configuration of the first transformation pipeline*. EX1003, ¶137. Similarly, compute node 130₂, (*second computer system*) with processing elements and operators that perform the *second transformation pipeline* (*supra*

Sections VII.A.2.k-VII.A.2.m), has memory storing data that *identifies at least a second transformation pipeline* (the pipeline of compute node 130₂).

(d) [9.4]

Supra Section VII.A.2.f (element 1.3). The second “process ...” and the “transmit ...” aspects are largely similar to corresponding claim 1 language, and are met for the same reasons. Namely, the Barsness FIG. 1B first compute node 130₁ (*first computer system*) is configured to process data stored in its memory (*the respective first data*) to identify the destination compute node PEs (*second transformation pipeline*) and to transmit its output message stream to that compute node (*transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined identification*). *Supra* Section VII.B.3; EX1003, ¶138.

(e) [9.5]

Supra Sections VII.A.2.k (element 1.4), VII.A.2.l (element 1.4.a), VII.A.2.m (element 1.4.b); EX1003, ¶139.

(f) [9.6]

Supra Section VII.A.2.n (element 1.5); EX1003, ¶140.

(g) [9.7]

Supra Section VII.A.2.o (element 1.6); EX1003, ¶141.

(h) [9.8]

Barsness satisfies this element, as the fourth compute node is part of the second plurality of computer systems. *Supra* Section VII.A.2.p (element 1.7); EX1003, ¶142. Additionally, for reasons explained above, the management system would, when needed, provision additional compute nodes implementing the first and second transformation pipelines, respectively (e.g., by duplicating both 130₁ and 130₂) and these additional nodes form the second plurality of compute nodes that execute at least one of the first and second transformation pipelines. *Supra* Section VII.A.2.p.

10. Claim 10

Supra Sections VII.A.9 (claim 9), VII.A.3 (claim 2); EX1003, ¶143.

11. Claim 11

Supra Sections VII.A.10 (claim 10), VII.A.4 (claim 3); EX1003, ¶144.

12. Claim 12

Supra Sections VII.A.9-11 (claim 9-11), VII.A.5 (claim 4), VII.A.7 (claim 6); EX1003, ¶145.

B. Ground 2: Claims 4, 6, 8, 12 Are Obvious Over Barsness in View of Chakradhar

Claim 1 (and therefore claims 4, 6, 8, and 12) is further obvious in view of Chakradhar, which expressly teaches using route-maps and explains the benefits of doing so. EX1003, ¶146.

1. Chakradhar (EX1005)

Chakradhar is a U.S. patent application publication that published September 15, 2016, from an application filed March 14, 2016, and alleges priority to a provisional application filed on March 13, 2015 (“Chakradhar Provisional”). EX1005, Cover. Chakradhar is prior art under at least Section 102 (a)(2) because the effective filing date of Chakradhar is the filing date of the provisional application, March 13, 2015. EX1003, ¶147.

Chakradhar is directed to the same field of distributed stream processing and provides an explicit mechanism for encoding and using topology information at the operator level. Chakradhar teaches that “[e]ach operator maintains a route-map that specifies the routes on which output messages should be sent after processing incoming messages,” and provides examples in which route-map entries identify downstream operators. EX1005, [0122]. Chakradhar further explains that route-maps encode “tuple-routing information and also the topology link structure.” EX1005, [0054]; EX1003, ¶148.

Chakradhar expressly ties the use of route-maps to concrete technical benefits. In particular, Chakradhar teaches that, “[b]y applying novel heuristics on the topology performance (feedback) metrics, ... the controller determines efficient route-maps,” and that these route-maps are “atomically injected into multiple operators, on-the-fly, using a light-weight, fault-tolerant protocol, for fast topology re-optimization.” EX1005, [0054]. Chakradhar further explains that operators use

these injected route-maps to control routing behavior during execution, enabling topology changes without interrupting stream processing. EX1005, [0054], [0122]; EX1003, ¶149.

Chakradhar is entitled to at least the March 13, 2015, filing date of the Chakradhar Provisional (EX1006) because Chakradhar lists the same inventors “named in the provisional application,” was “filed not later than 12 months after the date on which the provisional application was filed,” and “contain[s] a specific reference to the provisional application,” thus satisfying the requirements of 35 U.S.C. § 119(e)(1). *See* EX1005, Cover; EX1006, 1/22. Further, the Chakradhar Provisional discloses, *inter alia*, operators maintaining route-maps that include tuple-routing information and topology structure for directing output messages, i.e., the same technique disclosed in Chakradhar, as reflected in the support for Chakradhar claim 1 in the Chakradhar provisional. EX1006, 6/22, 9/22; EX1003, ¶¶150-51. As further demonstrated by citations to the Chakradhar Provisional supporting Chakradhar’s relied-upon disclosure in the mapping of Barsness and Chakradhar to the challenged claims (*infra* Sections VII.B.3), the Chakradhar Provisional supports all relied-upon disclosure in Chakradhar. EX1003, ¶¶150-51.

2. Combining Barsness And Chakradhar

As discussed below, a POSITA would have been motivated, with a reasonable expectation of success, to combine the computing infrastructure for real-time stream

processing applications taught by Barsness with the route-map teachings of Chakradhar to beneficially achieve “accelerated processing of stream processing engines” and “address[] bottlenecks in stream engines.” EX1005, [0006]; EX1006, 4/22; EX1003, ¶152.

A POSITA implementing a system like that taught in Barsness would have been motivated to incorporate Chakradhar’s route-map mechanism. Barsness already requires operators to emit output to downstream PEs according to an operator graph, but does not prescribe a specific internal data structure for locally storing and updating routing information at each operator. EX1004, 9:4-9. Chakradhar addresses this known design need by explicitly teaching operator-local storage and processing of routing data structures representing at least portions of the distributed topology. EX1005, [0122]; EX1006, 9/22. A POSITA would have recognized that Chakradhar’s route-maps provide a natural and efficient way to implement the routing behavior required by Barsness. EX1003, ¶153.

Moreover, the benefits identified by Chakradhar directly align with the goals and challenges described in Barsness. Chakradhar’s ability to update routing behavior “on-the-fly” and “atomically” across operators supports fast reconfiguration, scalability, and fault tolerance, which Barsness identifies as important for stream-processing systems that must rebalance load or move processing elements between machines. EX1004, 3:56-60, 10:28-39; EX1005,

[0054]; EX1006, 5/22. A POSITA would therefore have expected that incorporating Chakradhar's route-maps into Barsness would complement and improve the scalability, adaptability, and robustness of the computing system taught by Barsness. EX1004, 3:3-6 (describing "desired results with regards to scalability, efficiency, and real-time data processing"), 3:13-15 (describing "desired results where fault detections need to be fixed in real-time"), 3:19-22 (describing scalability); EX1003, ¶154.

Applying Chakradhar's route-maps to Barsness would have allowed each compute node to store respective data representing a portion of a distributed computational graph and to process that data locally to determine information identifying a downstream transformation pipeline and routing destination. Because Chakradhar's route-maps are expressly designed for distributed operator-based stream-processing systems, a POSITA would have had a reasonable expectation of success in making this modification. EX1005, [0043], [0056]; EX1006, 4/22, 6-7/22. The combination would have predictably reduced overhead for managing scalable stream applications and enabled efficient, consistent routing updates with minimal disruption to in-flight data streams. EX1005, [0054]-[0055]; EX1006, 5/22; EX1003, ¶155.

This obvious combination of Barsness and Chakradhar would include the features of the cloud-based cluster for distributed stream data processing

applications taught by Barsness, as described *supra* Section VII.A, with the operators storing and maintaining route maps “which encode tuple-routing information and also the topology link structure” as taught by Chakradhar. EX1005, [0054]; *id.*, [0009]-[0010], [0053], [0122]; EX1006, 5/22, 9/22. So implemented, the Barsness-Chakradhar combination renders all challenged claims obvious. EX1003, ¶156.

3. Claims 4, 6, 8, 12

Claim 1 is obvious in view of the Barsness-Chakradhar combination. Regarding a **distributed computing cluster** ([1.1]), Chakradhar discloses “[s]tream processing engines” such as “Storm,” which “is a distributed, real-time computation system” that “makes it possible to process unbounded streams of data.” EX1005, [0042], [0056]; *id.*, FIG. 13 (depicting a “cloud-based stream processing system” with multiple incoming data streams), [0134]; EX1006, 4/22. Chakradhar explains that “[o]n a storm *cluster*, topologies are executed and process streams of tuples (i.e., data).” EX1005, [0056] (emphasis added); *see also id.*, [0046], [0069] (“route-maps are installed in a consistent manner on a running cluster”), [0098], [0148] (evaluation of techniques “conducted on a 30-node cluster”); EX1006, 4/22, 6-7/22, 11/22. These disclosures complement the distributed computing architecture of Barsness, and thus, in combination, the Barsness-Chakradhar combination includes a ***distributed computing cluster***. EX1003, ¶157.

Regarding **element [1.2.b]**, just as the operator graph of Barsness “represents a stream application” (EX1004, 6:49-53), Chakradhar explains that “stream engines code an application workflow as a *directed* acyclic graph (DAG) of operators, referred to as a topology.” EX1005, [0043] (emphasis added); EX1006, 4/22. Chakradhar further discloses that stream data is “processed in a *pipelined* fashion as they traverse through the topology.” EX1005, [0043] (emphasis added); EX1006, 4/22. Chakradhar explains that “[t]he logic for *realtime* applications is packaged in a topology,” and that “[e]ach topology is represented by a graph including spouts (which produce tuples) and bolts (which transform tuples).” EX1005, [0056] (emphasis added); *id.*, [0106]; EX1006, 5/22, 8/22; EX1003, ¶158.

Chakradhar further discloses that “[e]very operator chooses a destination operator for its outgoing tuples based on route-maps.” EX1005, [0069]; *id.*, [0070] (describing “tuple-routing” between operators); EX1006, 5-6/22, 14/22. Further, “[r]oute maps contain information on the type and proportion of traffic for each destination operator.” EX1005, [0069]; EX1006, 6/22. Further, Chakradhar discloses that “[e]ach operator maintains a route-map that specifies the routes on which output messages should be sent after processing incoming messages.” EX1005, [0122]; EX1006, 9/22. The route-maps taught by Chakradhar therefore “represent[] topology link structure.” EX1005, [0053]; EX1006, 5/22. When route-maps are updated, they are “injected” into the operators “for fast topology re-

optimization.” EX1005, [0010], [0054]; EX1006, 5/22, 14/22. These route-maps are *first data* as recited in claim 1. EX1003, ¶159.

Further, in combination, each compute node of Barsness-Chakradhar stores an operator (executable code corresponding to the task assigned to that processing element) as taught by Barsness (EX1004, 9:4-9) and a route-map specifying where “output messages should be sent after processing incoming messages” as taught by Chakradhar (EX1005, [0122]; EX1006, 9/22). Together, this data stored at the first compute node is *respective first data representing a respective portion of a distributed computational graph*. EX1003, ¶160.

Regarding **elements [1.3.c] and [1.3.d]**, Chakradhar teaches that “[e]ach operator maintains a route-map that specifies the routes on which *output messages* should be sent after processing incoming messages.” EX1005, [0122] (emphasis added); EX1006, 9/22. In combination, an operator and its route-map are stored in the memory of the first computer system of Barsness-Chakradhar, which enables the processing element / operator to emit / send its output messages to the appropriate downstream PE as described by the operator graph. *E.g.*, EX1004, FIG. 2 (showing operator in memory of compute node). Specifically, such a route-map “encode[s] tuple-routing information and also the topology link structure,” as taught by Chakradhar. EX1005, [0010], [0054]; *id.*, [0043]; EX1006, 5/22, 9/22. In this way, “[e]very operator chooses a destination operator for its outgoing tuples based on

route-maps.” EX1005, [0069]; EX1006, 6/22. The first compute node, when executing the operator of a given PE at the end of the first transformation pipeline (e.g., PEs 2 and 3 in example operator graph of Barsness FIG. 1B), thus processes the route-map (*the respective first data stored in the memory of the first computer system*) to determine the destination of its output message (*to determine information about the second transformation pipeline*). The operators of Barsness-Chakradhar use route-maps to identify the destination of their output messages, and thus the first compute node (*first computer system*) transmits the output of the first compute node (*the first pipeline output messages*) to the second computer system according to the destination(s) specified in the route-map(s) (*the determined information*). EX1003, ¶161.

Thus, claim 1 is obvious in view of the Barsness-Chakradhar combination. Claims 4, 6, 8, and 12 are obvious for the same reasons set forth in Ground 1. *Supra* Section VII.A; EX1003, ¶162.

C. Ground 3: Claims 4, 6, 8, and 12 Are Obvious Over Barsness in View of Siripurapu

To the extent PO argues that Barsness provides insufficient detail on its scaling and multiple data streams, the claims are further obvious in view of Siripurapu, which expressly teaches adding computer nodes to handle processing additional real-time data streams and increased data flow. EX1003, ¶163.

1. Siripurapu (EX1007)

Siripurapu is a U.S. patent application publication that published February 27, 2014, and alleges priority to a provisional application filed November 18, 2010. EX1007, Cover. Siripurapu is prior art under at least Sections 102(a)(1-2). EX1003, ¶164.

Like Barsness, Siripurapu relates to processing real-time data streams with a scalable, “distributed, elastic architecture.” EX1007, [0060], [0453]. Siripurapu expressly teaches handling “additional data streams” and “increased data stream flows” through scaling, “wherein one or more additional worker nodes may be added ... to accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” *Id.*, [0453]; *see also id.*, [0359]-[0363]. “At any point in time, there are usually several Streams: some of them external (e.g., Twitter, Facebook).” EX1007, [0178]; *id.*, [0439]-[0440] (describing various “[i]nput real-time data streams” from different sources); EX1003, ¶165.

Siripurapu also teaches providing “fault-tolerance,” e.g., by assigning tasks from one worker node “to a different worker node.” EX1007, [0454]. Siripurapu also discloses re-assigning streams among compute nodes to address “hot spots in the flow of incoming stream events.” EX1007, [0470]; EX1003, ¶166.

For example, Figure 36 depicts how the system handles multiple “real-time data streams from one or more external applications 3602” through a conductor 3612 that manages the real-time processing of stream data by providing “instructions to

designate worker nodes for performing ... operations.” EX1007, [0444], [0446]. The conductor monitors “performance” and “perform[s] load balancing by reassigning tasks to the worker nodes.” *Id.*, [0448]-[0449]; *see also id.*, [0481]; EX1003, ¶167.

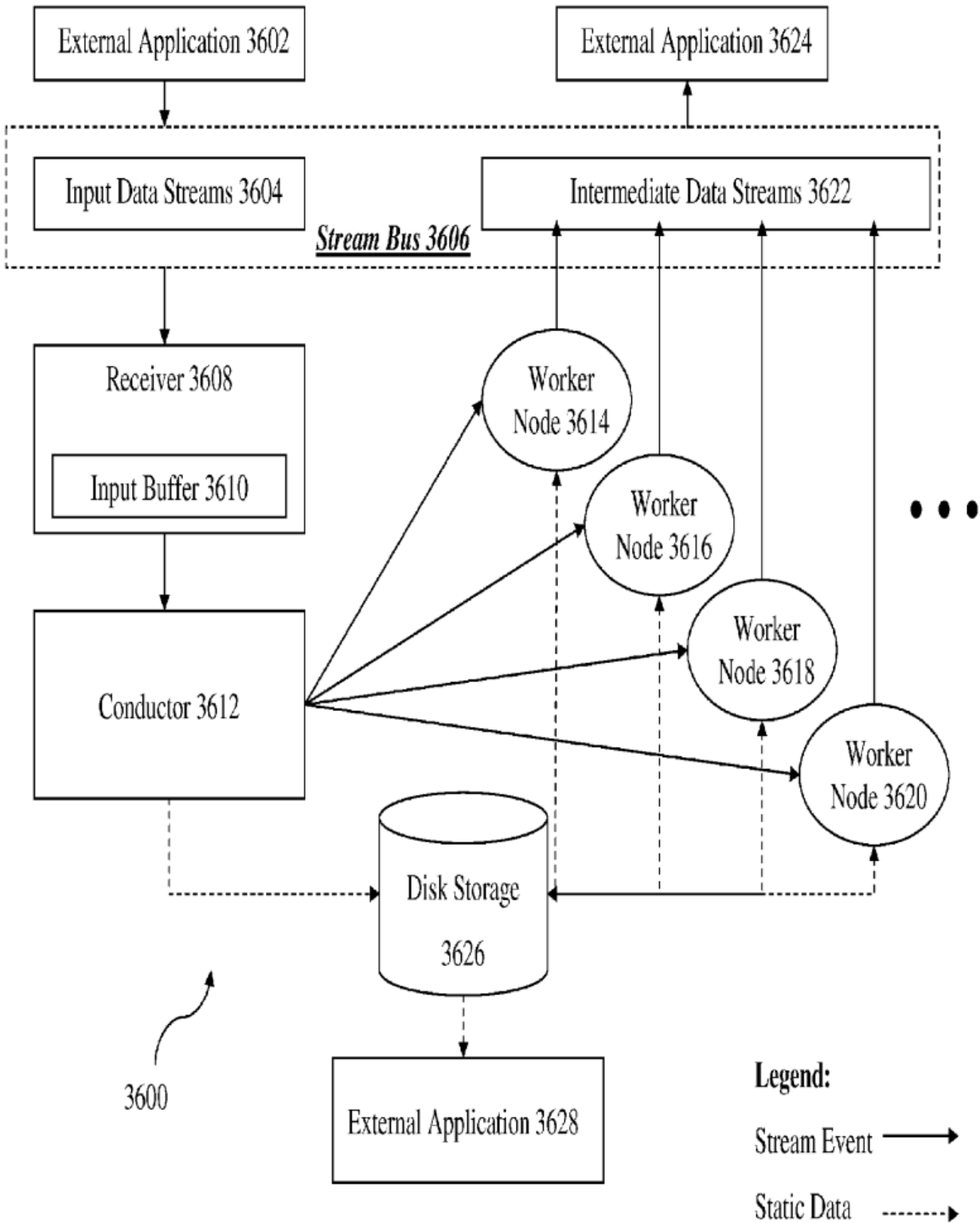


Figure 36

EX1007, FIG. 36; EX1003, ¶167.

2. Combining Barsness And Siripurapu

A POSITA would have been motivated, with a reasonable expectation of success, to implement the real-time stream processing applications taught by Barsness with the scalability teachings of Siripurapu to beneficially “accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” EX1007, [0453]; EX1003, ¶168.

Barsness expressly teaches scaling a distributed stream-processing application, e.g., to address data processing requirements. EX1004, 3:19-22, 6:53-55. Siripurapu teaches the same type of scaling in the same stream processing context, and provides further details on how and when to scale, e.g., by dynamically adding “one or more additional worker nodes” “to accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” EX1007, [0453]. A POSITA would have recognized Siripurapu’s teachings as directly aligning with—and easily integrated with—the scalable infrastructure of Barsness, which distributes PEs across compute nodes and can add/remove PEs for “data processing needs and scalability.” EX1004, 6:53-55; EX1003, ¶169.

The scalability teachings of Siripurapu reinforce the teachings in Barsness regarding “replicating processing elements on multiple nodes and load balancing

among them” (EX1004, 3:19-22) and spawning/provisioning more compute nodes as needed (EX1004, 6:19-42, 10:26-39). Like the management system of Barsness, the conductor of Siripurapu “perform[s] load balancing by reassigning tasks to the worker nodes” (EX1007, [0448]) and “assign[s] stream processing operations to the new worker nodes” provisioned to “accommodate increased computational requirements, additional data streams, increased data stream flows, and the like” (EX1007, [0453]); EX1003, ¶170.

Siripurapu further teaches details on receiving “one or more real-time data streams” and distributing them to worker nodes executing operators. EX1007, [0439]; *id.*, [0440]-[0460]. Incorporating these teachings into Barsness would have ensured the ability to dynamically handle additional input data streams and increased data stream flows, especially given “spikiness in the flow of stream events in the input data streams” where “a particular topic or link may suddenly become ‘hot’ or popular in an input data stream, resulting in a sudden surge.” *Id.*, [0352]; *see also id.*, [0002]; EX1003, ¶171.

Thus, to the extent Barsness alone did not teach doing so, a POSITA would have been motivated in view of Siripurapu to configure the Barsness infrastructure to scale the stream-processing application by adding additional, duplicate nodes to handle additional data streams from other applications. EX1003, ¶¶169-73. The two teachings are complementary, further reinforcing that a POSITA would have

naturally combined them to create an infrastructure adaptable to processing needs and the dynamic number of data streams. EX1003, ¶170.

Barsness already discloses a scalable cloud-based computing infrastructure, where compute nodes can be dynamically provisioned/duplicated to handle data processing needs, and thus a POSITA would have reasonably expected to succeed combining its teachings with those of Siripurapu. EX1004, 6:16-42; *id.*, 3:19-22; EX1003, ¶172.

This obvious combination of Barsness and Siripurapu includes the features of the cloud-based cluster for distributed stream data processing applications taught by Barsness, including scalability through the duplication/provisioning of nodes (*supra* Section VII.A) combined with Siripurapu's teachings on scaling and adding nodes to accommodate additional data streams. EX1007, [0453]. This Barsness-Siripurapu combination renders all challenged claims obvious. EX1003, ¶173.

3. Claims 4, 6, 8, 12

Claim 8 is obvious in view of the Barsness-Siripurapu combination, implementing the management system of Barsness to provision additional compute node(s) to handle increased data volume / processing needs at a given compute node/transformation pipeline, as taught by Barsness and Siripurapu. *Supra* Sections VII.A, VII.C.1-VII.C.2; EX1007, [0453]; *id.*, [0352] (explaining that teachings “are able to react in a fast and efficient manner to surges in input data streams, and to

balance computational loads relating to the ‘hot spots’ so that the data structures and nodes are still able to operate efficiently and generate output in short response times”); EX1003, ¶174.

Claims 4, 6, and 12 are further obvious in view of Barsness-Siripurapu, which includes Siripurapu’s express teachings on provisioning additional compute node(s) to handle processing of one or more additional input data streams from different sources. EX1007, [0453], [0352]; *supra* Sections VII.C.1-VII.C.2; EX1003, ¶174.

D. Ground 4: Claims 4, 6, 8, and 12 Are Obvious Over Barsness in View of Chakradhar and Siripurapu

For the same reasons a POSITA would have combined Barsness with Chakradhar and Siripurapu individually, a POSITA would have applied those teachings together when implementing Barsness. Those teachings relate to different aspects, are fully compatible with and complementary to one another—Chakradhar’s relates to storing graph information on nodes to ensure proper and efficient downstream transmission of transformed data, and Siripurapu provides details on scalability of the system to handle different workload needs.

Thus, a POSITA would have been motivated with a reasonable expectation of success to implement both simultaneously within the Barsness system. EX1003, ¶175.

VIII. NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS

Petitioner is not aware of any evidence of objective indicia of non-obviousness having a nexus to the challenged claims. EX1003, ¶176-180.

IX. CONCLUSION

Petitioner requests institution for claims 4, 6, 8, and 12 on Grounds 1-4 specified in this petition.

Date: July 20, 2026

Respectfully submitted,

By: /s/ James M. Glass
James M. Glass

Counsel for Petitioner

**CERTIFICATE OF COMPLIANCE WITH
TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS,
AND TYPE STYLE REQUIREMENTS**

1. This Petition complies with the type-volume limitation of 14,000 words, comprising 13,950 words, as counted using the Microsoft Word software that was used to prepare this paper, excluding the parts exempted by 37 C.F.R. § 42.24(a).

2. This Petition complies with the general format requirements of 37 C.F.R. § 42.6(a) and has been prepared using Microsoft® Word 2016 in 14-point Times New Roman.

Date: July 20, 2026

By: /s/ James M. Glass
James M. Glass
QUINN EMANUEL URQUHART
& SULLIVAN, LLP
295 5th Ave., 9th Floor
New York, NY 10016
Tel: (212) 849-7000
Fax: (212) 849-7100

Counsel for Petitioner

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on July 20, 2026, true and correct copies of the foregoing document and supporting materials were served in its entirety on the Patent Owner at the following address of record as listed on PAIR via First-Class Mail:

NK Patent Law
4101 Lake Boone Trail
Suite 218
Raleigh, NC 27607

Official Correspondence Address

Courtesy copies were also sent via electronic mail to Patent Owner's counsel of record in the related district court proceeding

aberge@irell.com
cwen@irell.com
dslusarczyk@irell.com
iwashburn@irell.com
claire@millerfairhenry.com

Date: July 20, 2026

By: /s/ James M. Glass
James Glass
QUINN EMANUEL URQUHART
& SULLIVAN, LLP
295 5th Ave., 9th Floor
New York, NY 10016
Tel: (212) 849-7000
Fax: (212) 849-7100

Counsel for Petitioner