

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. 2026-00371

U.S. Patent No. 12,143,425

**PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 12,143,425**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. MANDATORY NOTICES AND FEES	2
A. Real Party In Interest (37 C.F.R. § 42.8(b)(1)).....	2
B. Related Matters (37 C.F.R. § 42.8(b)(2))	2
1. Related Patent Office Proceedings	2
2. Related Litigation	2
C. Lead and Backup Counsel and Service Information (37 C.F.R. §§ 42.8(b)(3)-(4)).....	3
D. Payment of Fees (37 C.F.R. § 42.15(a)).....	4
E. Requirements For <i>Inter Partes</i> Review (37 C.F.R. §§ 42.101(A)- (C), 42.104(A), AND 42.108).....	5
III. STATEMENT OF RELIEF REQUESTED AND IDENTIFICATION OF CHALLENGE (37 C.F.R. § 42.104(B)).....	5
IV. PERSON OF ORDINARY SKILL IN THE ART	5
V. BACKGROUND	6
A. '425 patent.....	6
1. Earliest Priority Date.....	6
2. Specification	6
3. Prosecution History.....	7
VI. CLAIM CONSTRUCTION	8
VII. GROUNDS.....	15
A. Ground 1: Barsness.....	15
1. Summary of How Barsness Meets the Claims.....	16
2. Claim 1	19
(a) 1[pre] – A system comprising:	19
(b) 1[a] – a distributed computing network comprising a first plurality of computer systems, a second	

	plurality of computer systems, and a third plurality of computer systems, ...	19
(c)	1[b] – wherein a first computer system of the first plurality of computer systems is configured to execute first software instructions that:	26
(d)	1[b][i] – receive a first stream of data, ...	28
(e)	1[b][ii] – process first portions of the first stream of data by applying a first transformation pipeline to the first portions of the first stream of data to generate first pipeline output messages, and.....	29
(f)	1[b][iii] – transmit the first pipeline output messages;.....	33
(g)	1[c] – wherein a second computer system of the first plurality of computer systems is configured to execute second software instructions that:.....	34
(h)	1[c][i] – process first portions of the first pipeline output messages by applying a second transformation pipeline to the first portions of the first pipeline output messages;	35
(i)	1[d] – wherein a third computer system of the second plurality of computer systems comprises a memory that stores a first data, ...	37
(j)	1[d][i] – wherein the first data represents a portion of a distributed computational graph, ...	42
(k)	1[d][ii] – and wherein the portion of the distributed computational graph describes a flow of data from the first transformation pipeline to an input of the second transformation pipeline;	46
(l)	1[e] – wherein the third computer system is configured to execute third software instructions that:.....	48
(m)	1[e][i] – process the first data to determine information about at least one of the first transformation pipeline and the second transformation pipeline, ...	48

- (n) 1[e][ii] – monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions in accordance with the determined information, and 51
- (o) 1[f] – in response to the monitoring, cause a fourth computer system of the third plurality of computer systems to execute fourth software instructions that perform at least one of: 54
- (p) 1[f][i] – processing second portions of the first stream of data by applying the first transformation pipeline to the second portions of the first stream of data, or.....56
- 1[f][ii] – processing second portions of the first pipeline output messages by applying a second transformation pipeline to the second portions of the first pipeline output messages, ...57
- (q) 1[g] – wherein the first computer system and the third computer system are distinct, and.....59
- (r) 1[h] – wherein the third computer system and the fourth computer system are distinct.....60
- 3. Claim 2 – The system of claim 1, wherein the third software instructions monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions to identify execution that has become unresponsive. 60
- 4. Claim 3 – The system of claim 1, wherein the first computer system and the second computer system are distinct. 61
- 5. Claim 4 62
 - (a) 4[pre] – The system of claim 1,.....62
 - 4[a] – wherein the first computer system and the second computer system each further comprise one or more memories that each store a respective second data, ...62
 - (b) 4[b] – wherein the respective second data represents at least a respective portion of the distributed computational graph,.....64

(c)	4[c] – and wherein the first software instructions transmit the first pipeline output messages according to the flow of the distributed computational graph.....	65
6.	Claim 5 – The system of claim 1, wherein the third software instructions are further configured to monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions, to identify a transformation pipeline that causes at least one of the first computer system and the second computer system to transmit information that is outside established data stability boundaries.....	67
7.	Claim 6 – The system of claim 5,.....	67
(a)	6[a] – wherein the third computer system of the second plurality of computer systems is further configured to execute fifth software instructions that, upon identifying transmitted information that is outside established data stability boundaries:	67
6[b]	– determine changes to at least one of the first transformation pipeline and the second transformation pipeline in response to the identifying,.....	68
6[c]	– and cause at least one of the first computer system and the second computer system to execute sixth software instructions that modify a transformation pipeline in response to the determined changes.	68
B.	Ground 2: Claims 1-6 Are Obvious Over Barsness in View of Chakradhar	69
1.	Chakradhar (EX1005).....	70
2.	Combining Barsness and Chakradhar.....	71
3.	Claim 1	74
(a)	1[a]	74
(b)	1[d][i]	75
4.	Claim 4	76

C.	Ground 3: Claims 1-6 Are Obvious Over Barsness in View of Siripurapu	77
1.	Siripurapu (EX1007).....	77
2.	Combining Barsness and Siripurapu	79
3.	Claims 1-6	82
	(a) 1[e] through 1[h].....	82
	(b) Claim 2	83
D.	Ground 4: Claims 1-6 Are Obvious Over Barsness in View of Chakradhar and Siripurapu	84
VIII.	NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS	84
IX.	CONCLUSION	84

EXHIBIT LIST

Ex.	Description
1001	U.S. Patent No. 12,143,425 (“the ’425 patent”)
1002	Prosecution History for U.S. Patent No. 12,143,425
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

LIST OF CHALLENGED CLAIMS

Claim	Limitation
1[pre]	A system comprising:
1[a]	a distributed computing network comprising a first plurality of computer systems, a second plurality of computer systems, and a third plurality of computer systems,
1[b]	wherein a first computer system of the first plurality of computer systems is configured to execute first software instructions that:
1[b][i]	receive a first stream of data,
1[b][ii]	process first portions of the first stream of data by applying a first transformation pipeline to the first portions of the first stream of data to generate first pipeline output messages, and
1[b][iii]	transmit the first pipeline output messages;
1[c]	wherein a second computer system of the first plurality of computer systems is configured to execute second software instructions that:
1[c][i]	process first portions of the first pipeline output messages by applying a second transformation pipeline to the first portions of the first pipeline output messages;
1[d]	wherein a third computer system of the second plurality of computer systems comprises a memory that stores a first data,
1[d][i]	wherein the first data represents a portion of a distributed computational graph,
1[d][ii]	and wherein the portion of the distributed computational graph describes a flow of data from the first transformation pipeline to an input of the second transformation pipeline;
1[e]	wherein the third computer system is configured to execute third software instructions that:

Claim	Limitation
1[e][i]	process the first data to determine information about at least one of the first transformation pipeline and the second transformation pipeline,
1[e][ii]	monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions in accordance with the determined information, and
1[f]	in response to the monitoring, cause a fourth computer system of the third plurality of computer systems to execute fourth software instructions that perform at least one of:
1[f][i]	processing second portions of the first stream of data by applying the first transformation pipeline to the second portions of the first stream of data, or
1[f][ii]	processing second portions of the first pipeline output messages by applying a second transformation pipeline to the second portions of the first pipeline output messages,
1[g]	wherein the first computer system and the third computer system are distinct, and
1[h]	wherein the third computer system and the fourth computer system are distinct.
2	The system of claim 1, wherein the third software instructions monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions to identify execution that has become unresponsive.
3	The system of claim 1, wherein the first computer system and the second computer system are distinct.

Claim	Limitation
4[pre]	The system of claim 1 ,
4[a]	wherein the first computer system and the second computer system each further comprise one or more memories that each store a respective second data,
4[b]	wherein the respective second data represents at least a respective portion of the distributed computational graph,
4[c]	and wherein the first software instructions transmit the first pipeline output messages according to the flow of the distributed computational graph.
5	The system of claim 1 , wherein the third software instructions are further configured to monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions, to identify a transformation pipeline that causes at least one of the first computer system and the second computer system to transmit information that is outside established data stability boundaries.
6	The system of claim 5 ,
6[a]	wherein the third computer system of the second plurality of computer systems is further configured to execute fifth software instructions that, upon identifying transmitted information that is outside established data stability boundaries:
6[b]	determine changes to at least one of the first transformation pipeline and the second transformation pipeline in response to the identifying,
6[c]	and cause at least one of the first computer system and the second computer system to execute sixth software instructions

Claim	Limitation
	that modify a transformation pipeline in response to the determined changes.

I. INTRODUCTION

U.S. Patent No. 12,143,425 (“the ’425 patent”) was filed on June 21, 2024 and lists an earliest priority date of October 28, 2015. The claims of the ’425 patent relate to processing data streams using a distributed computing cluster. It employs multiple computer systems in a pipeline to apply transformations to those streams, with one of the computer systems monitoring the processing to determine if additional computer systems should be added. But the notion of monitoring the processing of data streams in a pipeline is well-known in the art. For example, Barsness (EX1004), which published more than three years before to the earliest priority date recited on the ’425 patent and was not of record during prosecution, discloses a distributed computing architecture for processing stream data. Processing is carried out by a multi-branch pipeline of “processing elements” (PEs) distributed among different compute nodes. It also describes monitoring the PEs and scaling this architecture, e.g., by duplicating nodes, to ensure processing rates remain within a desired range.

Barsness alone renders all challenged claims obvious. Indeed, it teaches many claim elements in more detail than does the ’425 patent itself. For example, Barsness teaches dynamically duplicating processing elements from one compute node to an additional compute node, to maintain desired processing rates. In contrast, the ’425 patent describes only generally, without detail, that “actions can be implemented to

resolve the issue” of slow processing. EX1001, 16:54-58. This petition relies on Barness as a single-reference obviousness ground, as well as alternative grounds combining Barness with other secondary references.

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

The '425 patent is in the same family as U.S. Patent No. 12,143,424 ('424 patent). It shares a common specification with the '425 patent. 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.

2. Related Litigation

Patent Owner is currently asserting the '425 patent against Petitioner in the following litigation:

Name	Number	Court	Filed
<i>Qomplx LLC v. Palo Alto Networks, Inc.</i>	2:25-cv-00913	EDTX	Aug. 28, 2025

Patent Owner is currently asserting the related U.S. Patent No. 12,143,424 in the following litigations:

Name	Number	Court	Filed
<i>Qomplx LLC v. Palo Alto Networks, Inc.</i>	2:25-cv-00913	EDTX	Aug. 28, 2025
<i>Qomplx LLC v. Microsoft Corporation</i>	1:25-cv-01383	WDTX	Aug. 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 10014 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 '425 patent is available for *inter partes* review, that Petitioner is not barred or estopped from challenging the '425 claims on the grounds identified herein, and that the prohibitions of 35 U.S.C. §§ 315(a)-(b) are inapplicable.

III. STATEMENT OF RELIEF REQUESTED AND IDENTIFICATION OF CHALLENGE (37 C.F.R. § 42.104(B))

Petitioner respectfully requests *inter partes* review of claims 1-6 (“Challenged Claims”) on the following grounds:

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

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, ¶¶39-41.

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, ¶42.

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

V. BACKGROUND

A. ’425 patent

1. Earliest Priority Date

The ’425 patent issued November 12, 2024, from an application filed July 21, 2024. EX1001, 1. It alleges priority via a string of continuation and continuation-in-part applications to October 28, 2015. *Id.* Petitioner does not concede entitlement to a date of October 28, 2015, but for purposes of this Petition treats that as the effective filing date.

2. Specification

The ’425 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, 16:43-58. The ’425 patent does not, however, further describe these issue-resolving “actions” or how to implement them.

The ’425 patent provides examples of prior art transformation pipelines (FIGS. 6, 12) and other transformation pipelines (FIGS. 7-9, 13-15). EX1001, 17:13-19:25, 21:33-22:59 (describing transformation pipelines). However, the ’425 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, 13:52-64. EX1003, ¶¶36-38.

3. Prosecution History

The first Office Action granted a request for prioritized examination. EX1002, 420. Thereafter, the applicant and Examiner agreed to an examiner’s amendment, and the Examiner issued a notice of allowability. EX1002, 422, 501-505. The Reasons for Allowance concluded that the prior art of record did not disclose the amended claim elements, namely:

in response to the monitoring, cause a fourth computer system of the third plurality of computer systems to execute fourth software instructions that perform at least one of: processing second portions of the first stream of data by applying the first transformation pipeline to the second portions of the first stream of data, or processing

second portions of the first pipeline output messages by applying a second transformation pipeline to the second portions of the first pipeline output messages, wherein the first computer system and the third computer system are distinct, and wherein the third computer system and the fourth computer system are distinct.

EX1002, 506-09; *id.*, 509-513 (discussing prior art of record).

VI. CLAIM CONSTRUCTION

As an initial matter, the claim terms should be given their plain and ordinary meaning under *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–13 (Fed. Cir. 2005).

The '425 patent claims the same priority as, was filed the same day as, and shares the same specification as the '424 patent, which is the subject of a petition for *inter partes* review filed by Microsoft Corporation (IPR2026-00298), which Petitioner has moved to join in IPR2026-00363. The '424 patent has also been asserted against Microsoft in *Qomplx LLC v. Microsoft Corporation*, 1:25-cv-01383 (W.D. Tex. Aug. 28, 2025). Because the claims of the '424 patent contain many of the same elements and claim terms as the '425 patent, and to maintain consistency in its position, Petitioner proposes the same constructions for common claim terms that Microsoft proposed in IPR2026-00298 and the district court litigation, and that Petitioner proposed in IPR2026-00363. Patent Owner has argued in the district court

case against Microsoft that these terms need not be construed and should be given their plain meaning. However, Patent Owner’s construction is broader and inclusive of Petitioner’s construction. 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 ’425 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, 9:33-37 (emphasis added). The specification does not elsewhere describe a “***distributed computational graph***.” See, e.g., *id.*, 9:37-52. 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. Consistently, the ’425 patent defines “graph” with reference to nodes and

edges (*id.*, 9:7-25) and “directed” graph as a graph with edges directed “from one node to another in one direction” (*id.*, 9:25-33).

Petitioner’s construction is consistent with the ’425 patent’s embodiments. For example, Figure 7 (reproduced below), depicts a “distributed computational graph 700.” EX1001, 17:53-57. Each “transformation node” corresponds to a “function” that is performed on a stream of input data and that outputs an “output message” to the next transformation node. *Id.*, 17:60-62. Each transformation corresponds to a node: “Transformation Node 1” 710, “Transformation Node 2” 720, up to “Transformation Node *n*” 750. *Id.*, 17:54-18:36. And each edge (715, 725, 735, 745) between transformation nodes corresponds to an output message. *Id.*, 17:60-62. These edges include arrows indicating the direction of data flow, showing this is a directed graph. The same is true of Figures 8-9 and 13-15. EX1001, 18:37-19:19, 19:20-20:8, 21:50-22:59.

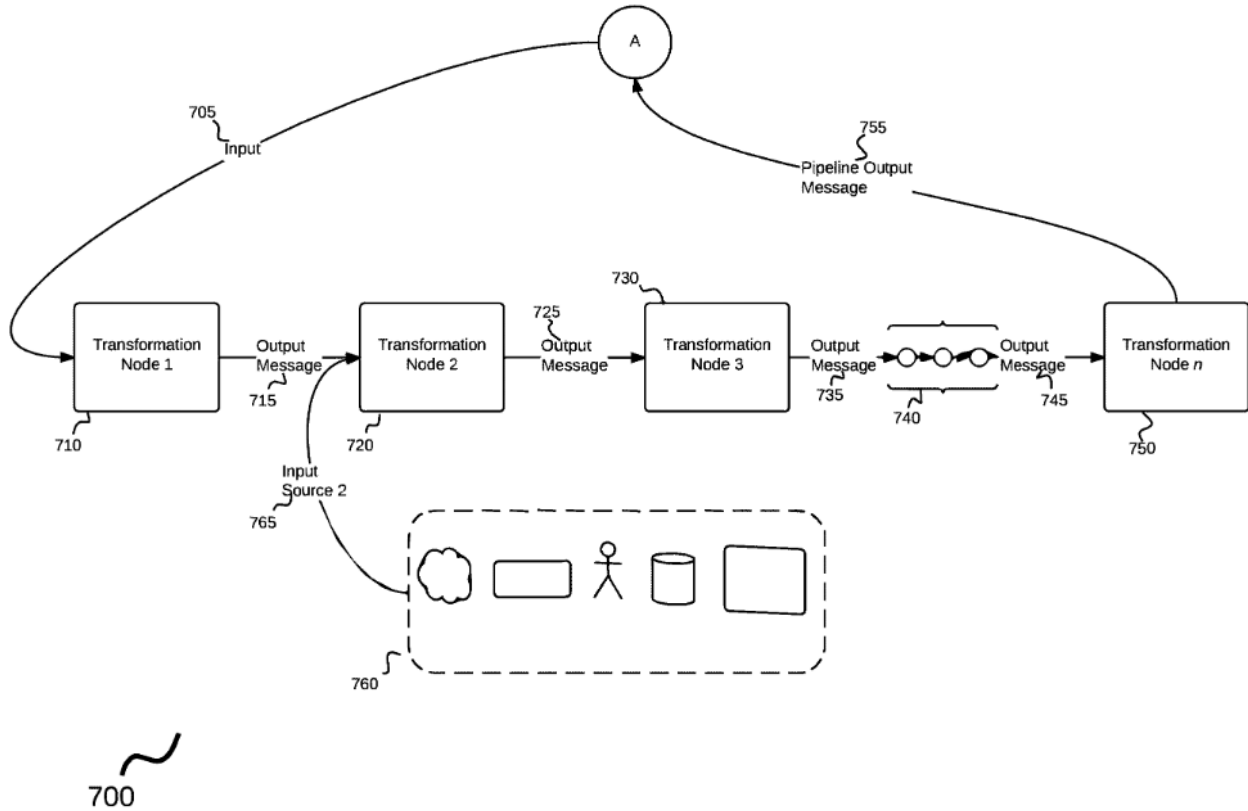


Fig. 7

Patent Owner has disputed Petitioner’s construction in the parallel district court proceeding against Microsoft. EX1019, 16-20. While Patent Owner only proposes that the term be given its “[p]lain and ordinary meaning,” Patent Owner states 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,” with no “requirement[] about what precisely the graph’s nodes and edges represent.” *Id.*, 16. In other words, Patent Owner argues that the term does not require any of (1) a directed graph, (2) that the nodes represent transformations, and (3) that the edges represent the output messages of a

transformation. While Petitioner’s construction is correct, Patent Owner’s construction is broader and inclusive of Petitioner’s construction. Thus, even under Patent Owner’s broader construction, the prior art meets this limitation—at the very least, any graph that meets Petitioner’s narrower construction would also meet Patent Owner’s broader 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 ’425 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, 9:53-56. Nothing in the rest of the patent or the prosecution history alters this definition or suggests it does not apply to the challenged claims. Thus, patentee has at least acted as its own lexicographer. *Thorner v. Sony Computer Entertainment America LLC*, 669 F.3d 1362, 1365-66 (Fed. Cir. 2012).

In the parallel district court proceeding against Microsoft, Patent Owner argues that Petitioner’s construction of this term *is* the plain meaning of the term and that Petitioner’s construction would “likely cause jury confusion.” EX1019, 22-23. Regardless, there is no dispute that “transformation” means “a function performed

on zero or more streams of input data which results in a single stream of output,” whether that is because it is the term’s plain meaning or because Patent Owner acted as its own lexicographer. 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”</i>¹	/	“A progression of functions which each perform a transformation on a data stream.”
--	---	---

The specification section titled “Discussion of the State of the Art” describes “data pipelines” as “a progression of functions which each perform some action or transformation on a data stream.” EX1001, 4:31-34. Although this passage refers to “data pipelines” rather than “transformation pipelines,” it remains instructive because, first, the claimed ***“transformation pipelines”*** operate on data. *Id.*, cl. 1 (a “first stream of data”). Second, 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.

¹ While the term “pipeline” is used in the claims without the modifier “transformation,” it always refers to a “transformation pipeline.” *See, e.g.*, EX1001, cl. 1 (“a ***first transformation pipeline*** ... to generate ***first pipeline*** output messages”).

Further supporting Petitioner’s construction, the specification’s “Definition” section explains: “In the invention, transformations are *placed into pipelines* such that the output of one transformation may serve as an input for another. These pipelines can consist of two or more transformations with the number of transformations limited only by the resources of the system.” EX1001, 9:64-10:2. This, and related passages describing “*transformation pipelines*” all show that this term refers to a progression of functions which each perform a transformation on a data stream. *E.g., id.*, 10:2-10:9, 17:54-60 (describing FIG. 7 “transformation pipeline 700” as a progression of transformation node functions performed on data stream). Thus, a POSITA would understand that the term “transformation pipeline” within the context of the ’425 patent means “[a] progression of functions which each perform a transformation on a data stream.”

In the parallel district court proceeding against Microsoft, Patent Owner argues that this term does not need to be construed and should be given its plain meaning. EX1019, 20-22. Patent Owner does not propose any particular plain meaning but argues that “[t]he specification is replete with examples of transformation pipelines.” EX1019, 21 (citing to the ’424 patent’s equivalent of EX1001, 17:13-20:8, 21:33-22:59, Figs. 6-9, 12-15). All of these “examples of transformation pipelines” meet Petitioner’s construction. For example, as explained above, 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.*, 17:60-64. As shown in Figure 7, each transformation node also transforms the input message into an output message that is sent to a downstream transformation node. 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 ’425 patent are directed to transforming a data stream using a pipeline of computers and monitoring the processing. 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.

Monitoring is performed by a *second plurality of computer systems*. The monitoring system (a *third computer system*) stores a description of the processing flow as a *distributed computing graph*, then monitors the transformation of the data, such as determining whether additional computing is needed. The monitoring can then cause another computer system (a *fourth computer system* of a *third plurality of computer systems*) 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 *second plurality*): monitors the transformation
- *Fourth computer system* (of the *third plurality*): in response to monitoring, performs the same transformation being 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, ¶47. Barsness qualifies as prior art under Sections 102(a)(1) and 102(a)(2).

Barsness discloses the transformation of data using pipelines, and the monitoring of the transformation. Specifically, Barsness discloses transformation of 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, ¶¶48-51. As shown in the annotated Figure 1B below, each compute node is a *computer system* and the PEs are a *transformation pipeline*.

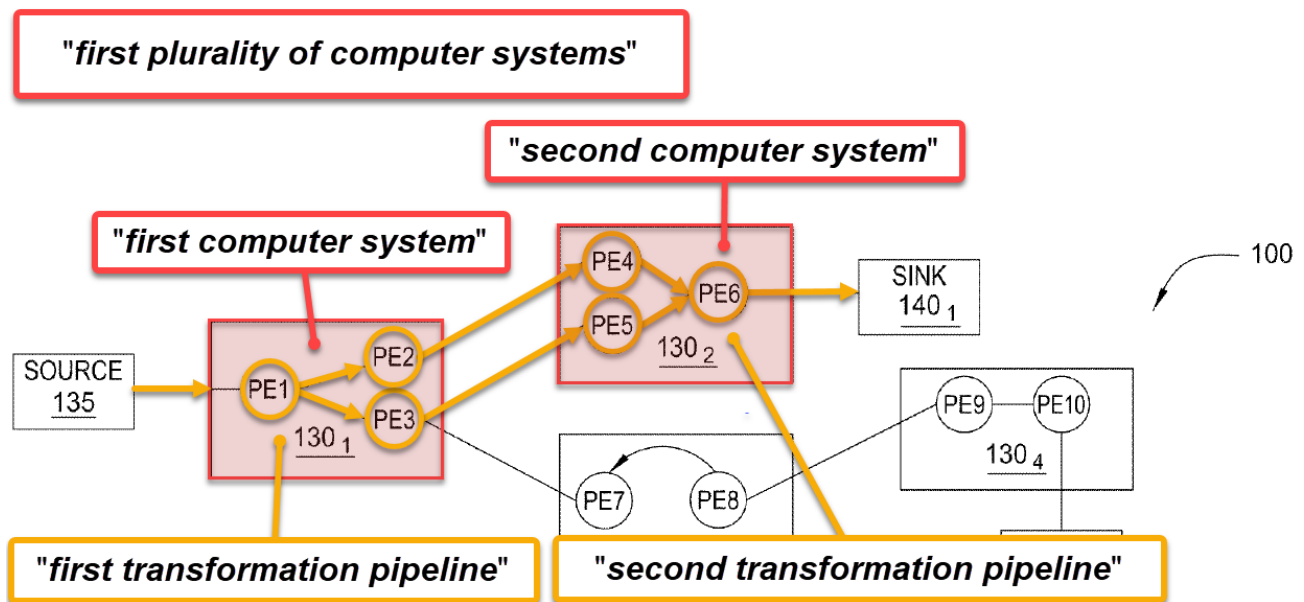
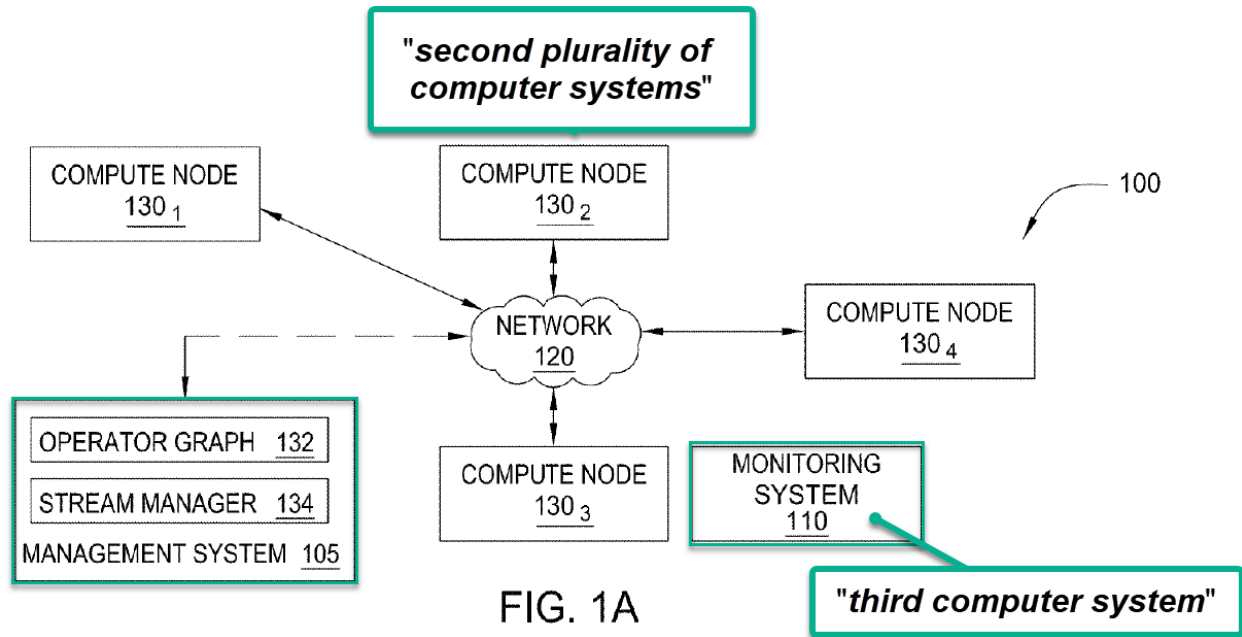


FIG. 1B

Barsness also discloses computer systems for monitoring the transformation. Barsness discloses that its “monitoring system 110” (*third computer system*) stores the operator graph. EX1004, 3:45-50.



EX1004, Fig. 1A (annotations added).

The “monitoring system 110” uses the 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 “spawning new compute nodes, [and] duplicating a compute node” (*fourth computer system*). EX1004, 14:51-56; EX1003, ¶52.

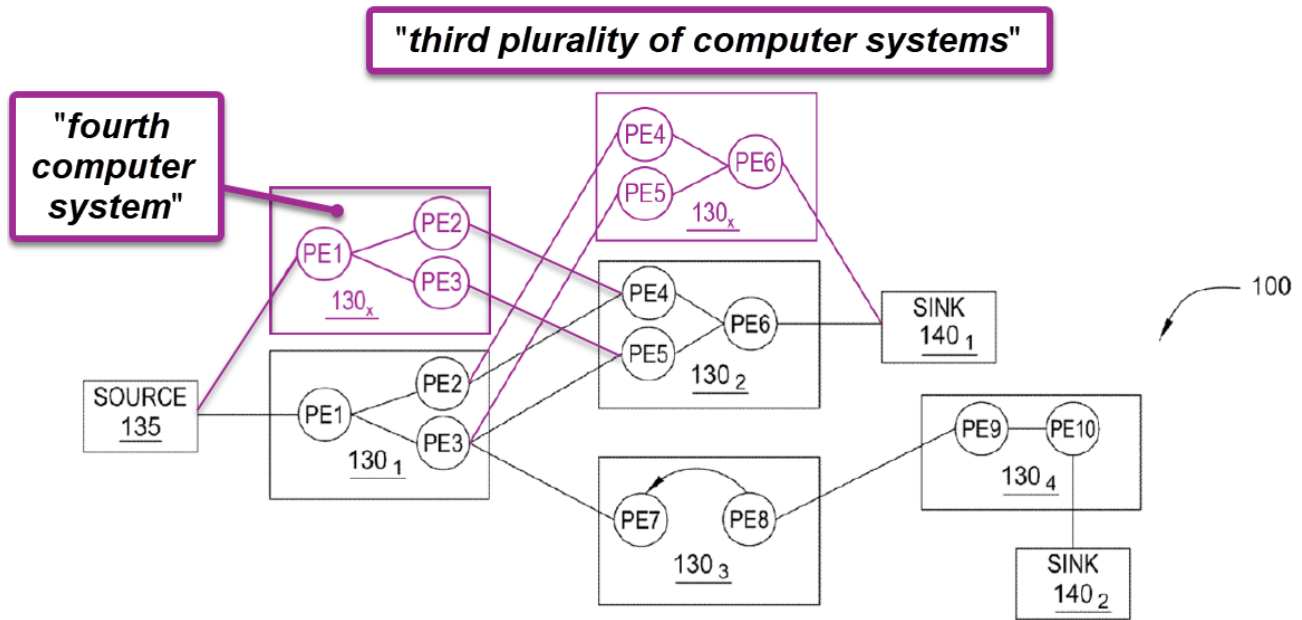


FIG. 1B

EX1004, FIG. 1B (annotations added).

2. Claim 1

(a) 1[pre] – A system comprising:

To the extent the preamble is limiting, Barsness discloses a **system**, specifically, “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, ¶¶65-67.

(b) 1[a] – a distributed computing network comprising a first plurality of computer systems, a second plurality of computer systems, and a third plurality of computer systems,

Barsness discloses a “computing infrastructure 100” (*distributed computing network*) that includes “a monitoring system 110,” “a management system 105 and a plurality of compute nodes 130₁₋₄, each connected to a communications network 120.” EX1004, 7:15-26, 6:43-48. FIG. 1A shows an exemplary computing infrastructure and its distributed computing elements:

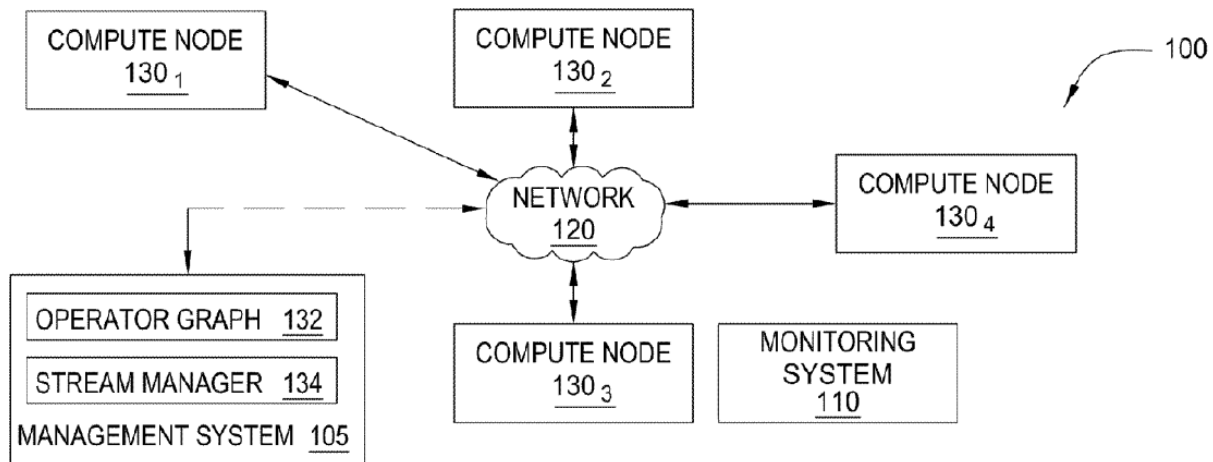


FIG. 1A

EX1004, FIG. 1A; EX1003, ¶¶68-69.

Each compute node is “a distributed computing node.” EX1004, 2:35-37. 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. Barsness further describes a “stream manager 134” which “control[s] a stream application *running* on the compute nodes,” such as by “moving processing elements (PEs) from one compute node 130 to another.” *Id.*, 7:27-37. Thus, the computing infrastructure—with its “plurality of compute nodes” and “management” and “monitoring” systems distributed throughout

network 120—is a *distributed computing network* because the elements communicate over the network to perform coordinated processing. EX1003, ¶70.

“first plurality of computer systems,”: Barsness discloses that its computing infrastructure includes “a plurality of compute nodes 130₁₋₄.” EX1004, 6:45-49; EX1003, ¶71. FIGS. 1A and 1B show the compute nodes:

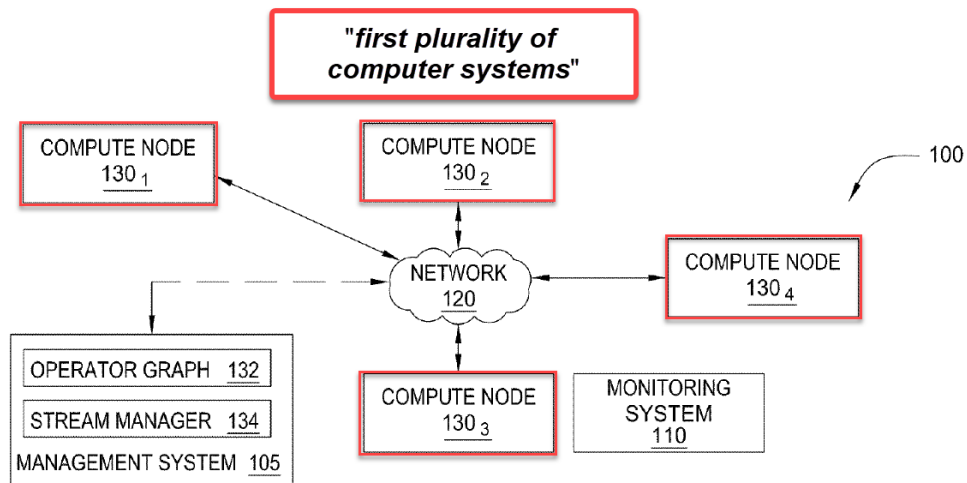


FIG. 1A

EX1004, FIG. 1A (annotations added); EX1003, ¶71.

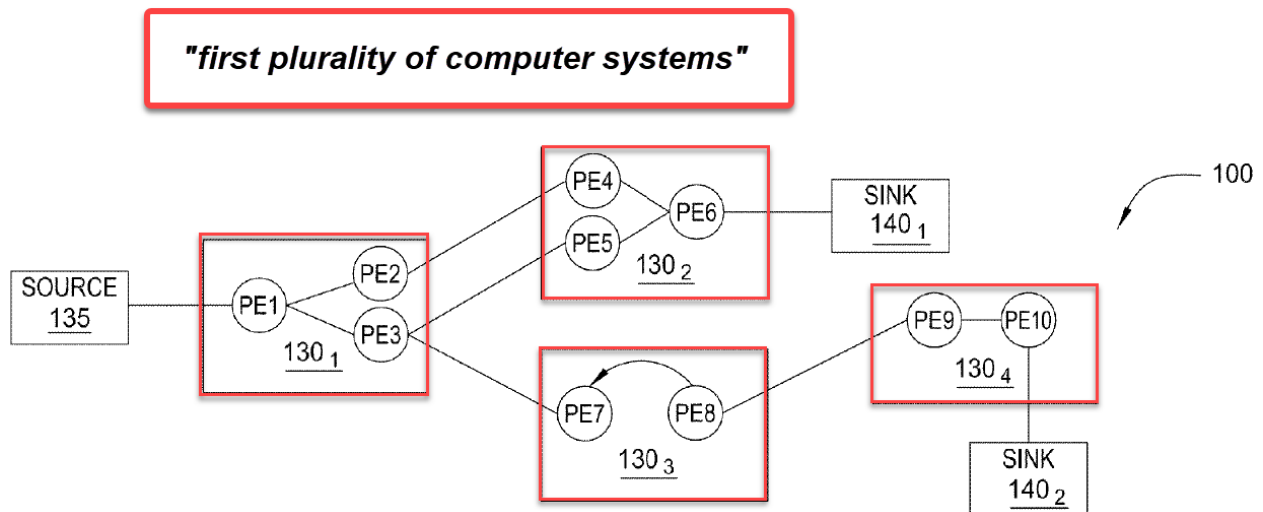


FIG. 1B

EX1004, FIG. 1B (annotations added); EX1003, ¶71.

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; EX1003, ¶72.

Barsness FIG. 2 shows details of a compute node:

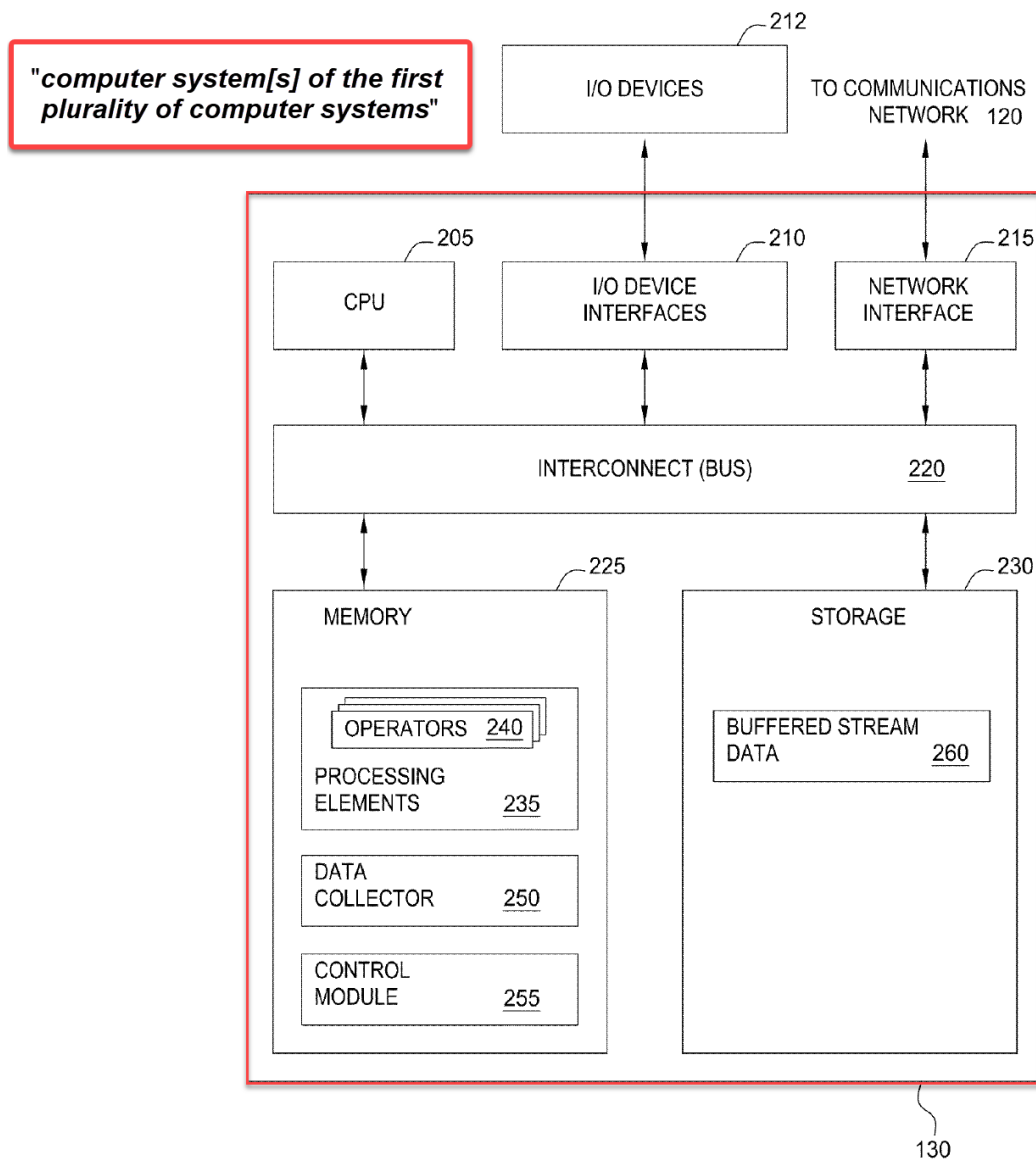


FIG. 2

EX1004, FIG. 2 (highlighting 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, ¶73.

“*second plurality of computer systems*,”: Barsness discloses that its computing infrastructure includes *a second plurality of computer systems*, e.g., “management system 105” and “monitoring system 110.” EX1004, 6:45-49; *id.*, 7:15-17; EX1003, ¶74. FIG. 1A shows a second plurality of computer systems (green):

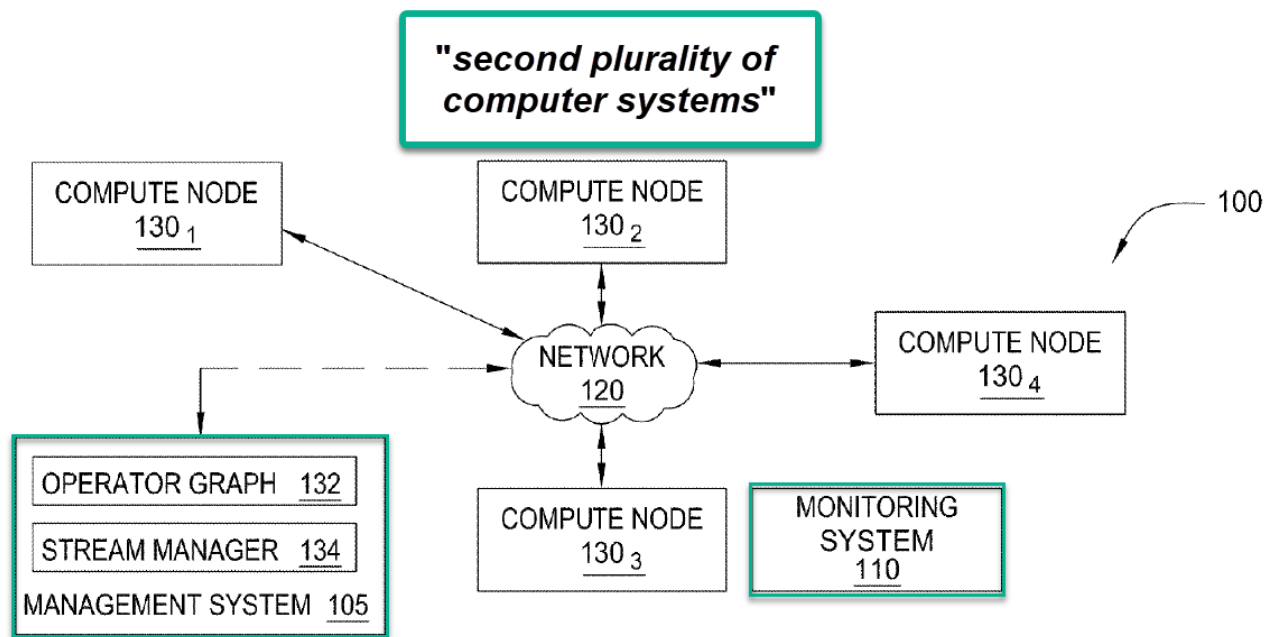


FIG. 1A

EX1004, FIG. 1A (highlighting added); EX1003, ¶74.

The Barsness management system 105 is 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.*, 9:51-10:64; EX1003, ¶75.

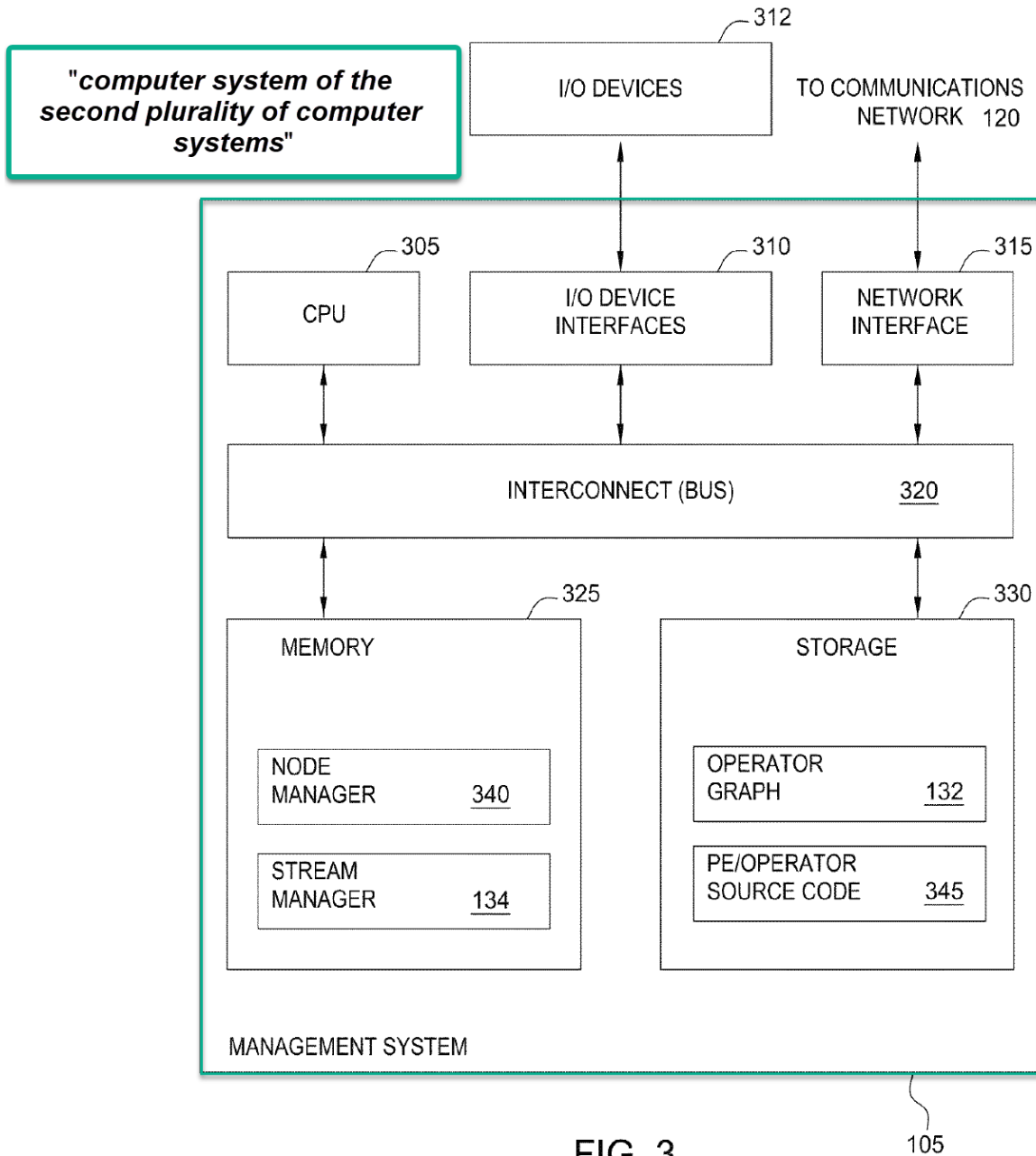
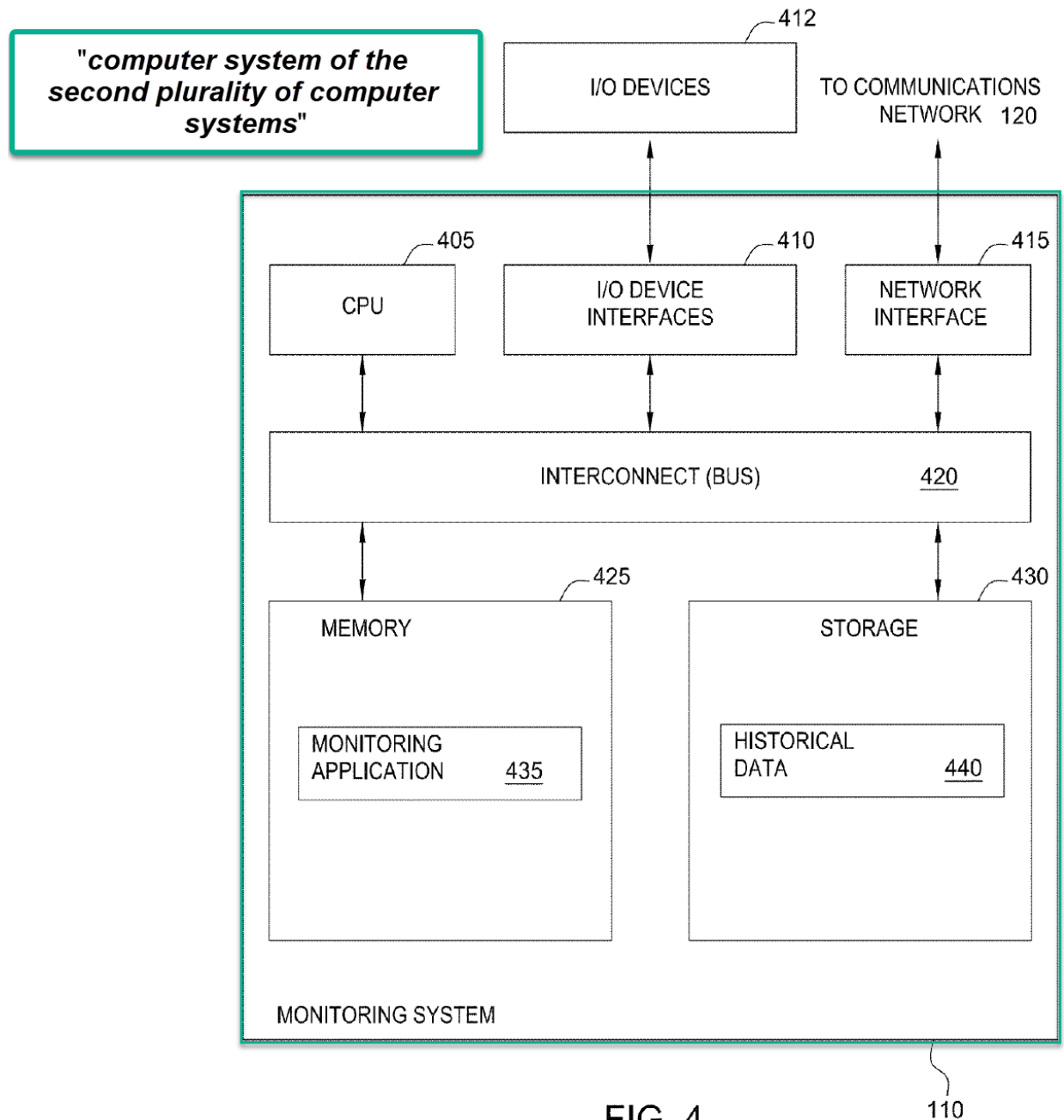


FIG. 3

EX1004, FIG. 3 (highlighting added).

The Barsness monitoring system 110 is also a ***computer system*** as seen in FIG. 4. EX1004; EX1003, ¶76. Additionally, monitoring system 110 “may be a cluster of a plurality of computing resources.” EX1004, 7:17-21.



EX1004, FIG. 4 (highlighting added).

“third plurality of computer systems”: As discussed regarding limitation 1[f], 1[f][i], and 1[f][ii], Barsness discloses that monitoring system 110 is configured to add additional compute nodes 130_x (**a third plurality of computer systems**) to improve performance of the stream application. Barsness thus contemplates additional, unconfigured compute nodes (**computer systems**) that are available to be configured and assigned by monitoring system 110 on demand. These additional compute nodes comprise **a third plurality**. EX1003, ¶¶77-78.

- (c) 1[b] – wherein a first computer system of the first plurality of computer systems is configured to execute first software instructions that:

Compute node 130₁ (**first computer system**) is depicted in FIG. 1B as part of the **first plurality of computer systems**:

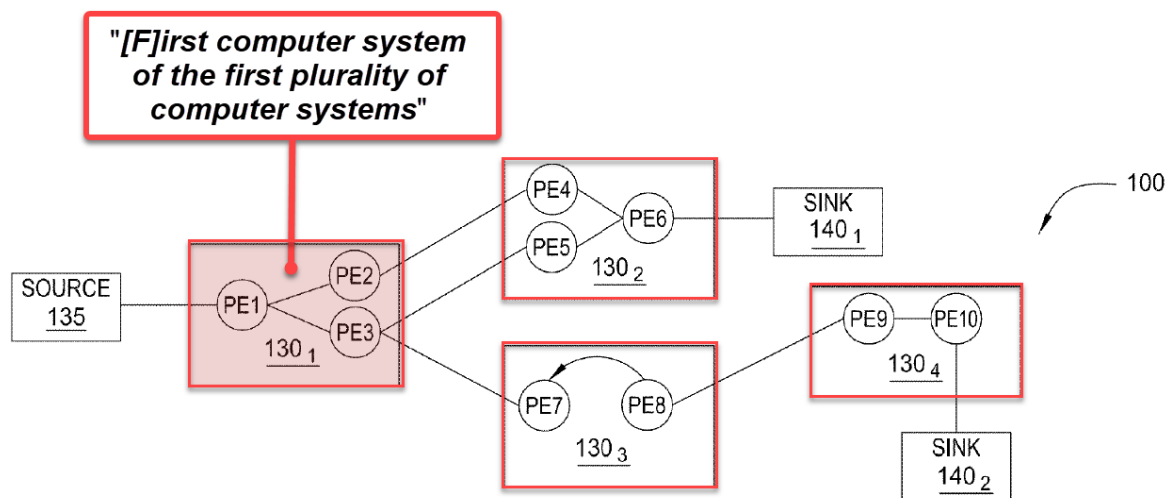


FIG. 1B

EX1004, FIG. 1B (annotations added); *see also id.*, FIG. 2, 7:54-55, 8:30-34; EX1003, ¶79.

“configured to execute first software instructions”: Barsness teaches that each compute node’s CPU executes the “executable code” for its PEs. EX1003, ¶¶80-81. The CPU “retrieves and executes programming instructions stored in the memory 225,” and the memory stores one or more PEs. EX1004, 8:53-55, 8:66-9:14; *see also id.*, 7:13-15. Each PE performs “data analytic functions or operations” according to the operators 240 of the PE. EX1004, 6:65-7:3; *id.*, 7:40-44. Each operator is a “small chunk of executable code configured to process data flowing into a processing element.” EX1004, 9:4-9; *id.*, 3:19-22.

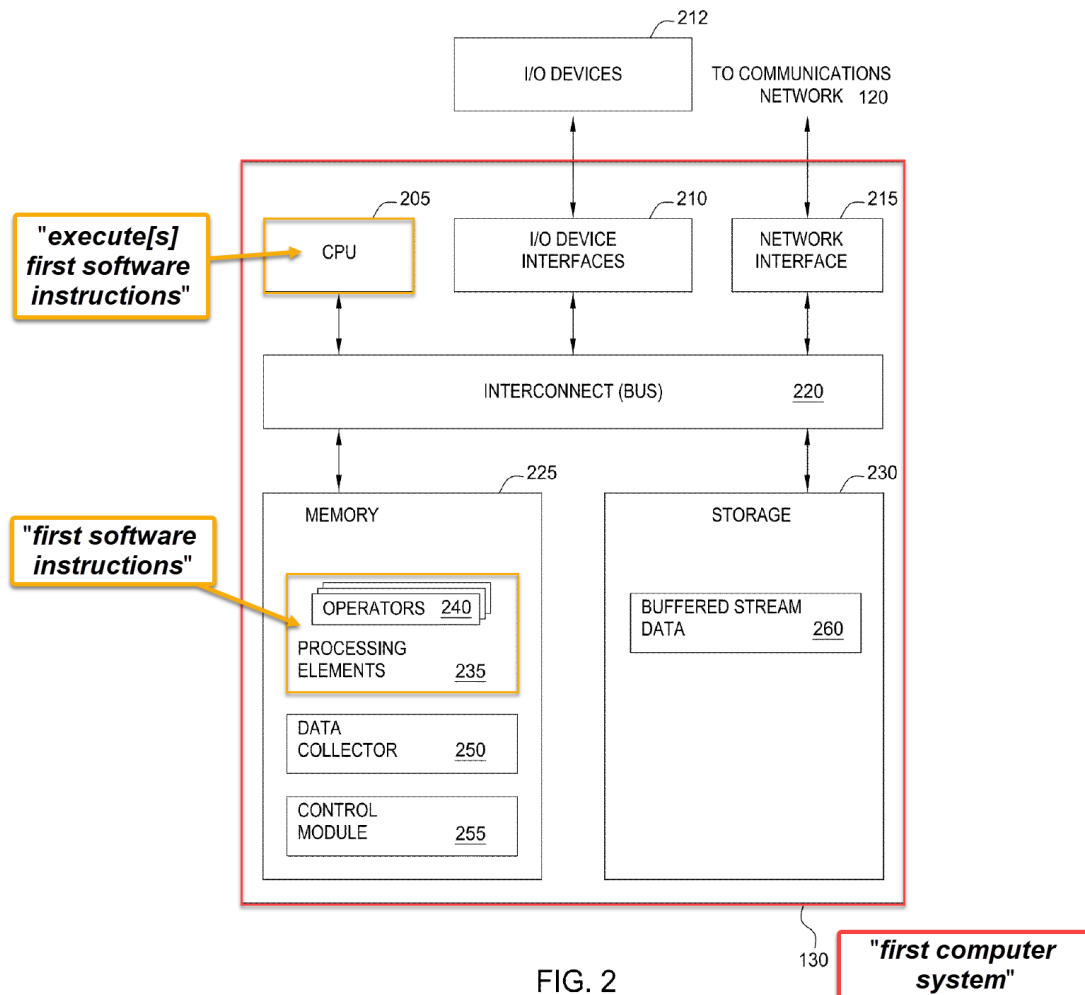


FIG. 2

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

(d) 1[b][i] – receive a first stream of data,

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):

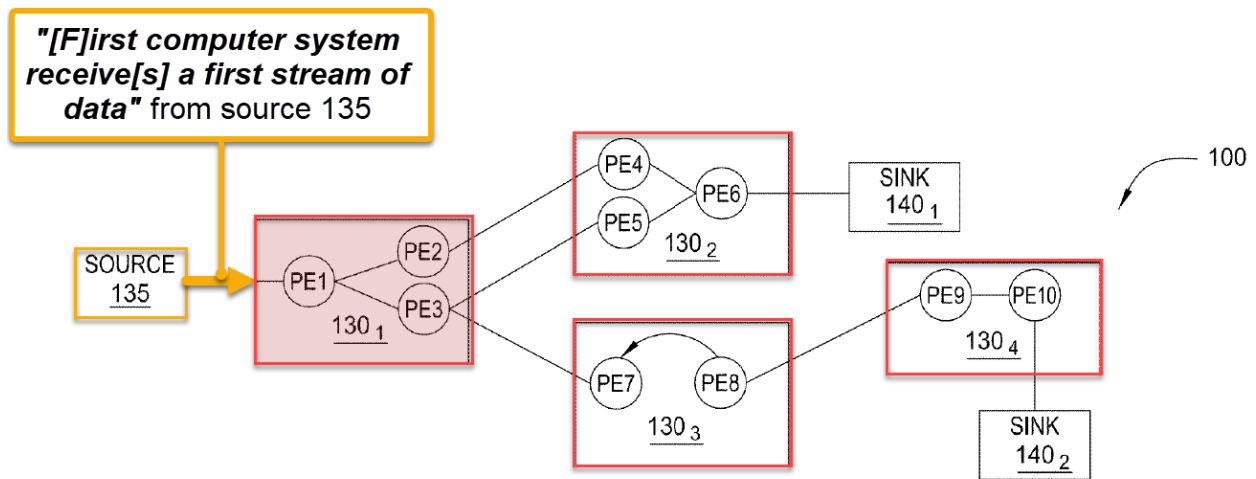
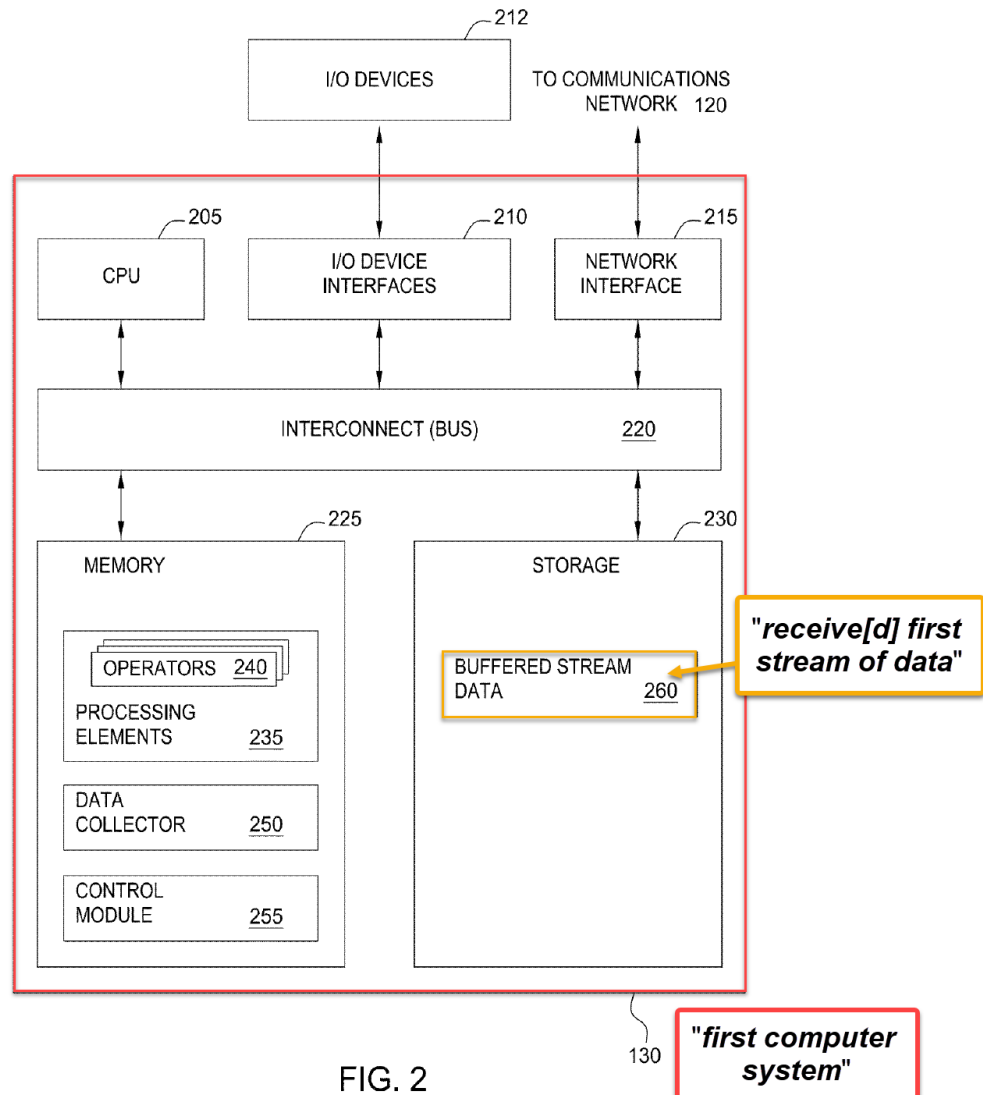


FIG. 1B

EX1004, FIG. 1B (annotations added), 6:49-56; EX1003, ¶82.

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, ¶83.

- (e) **1[b][ii] – process first portions of the first stream of data by applying a first transformation pipeline to the first portions of the first stream of data to generate first pipeline output messages, and**

Barsness discloses compute node 130₁'s PEs process the data elements received from source 135, as seen in FIG. 1B:

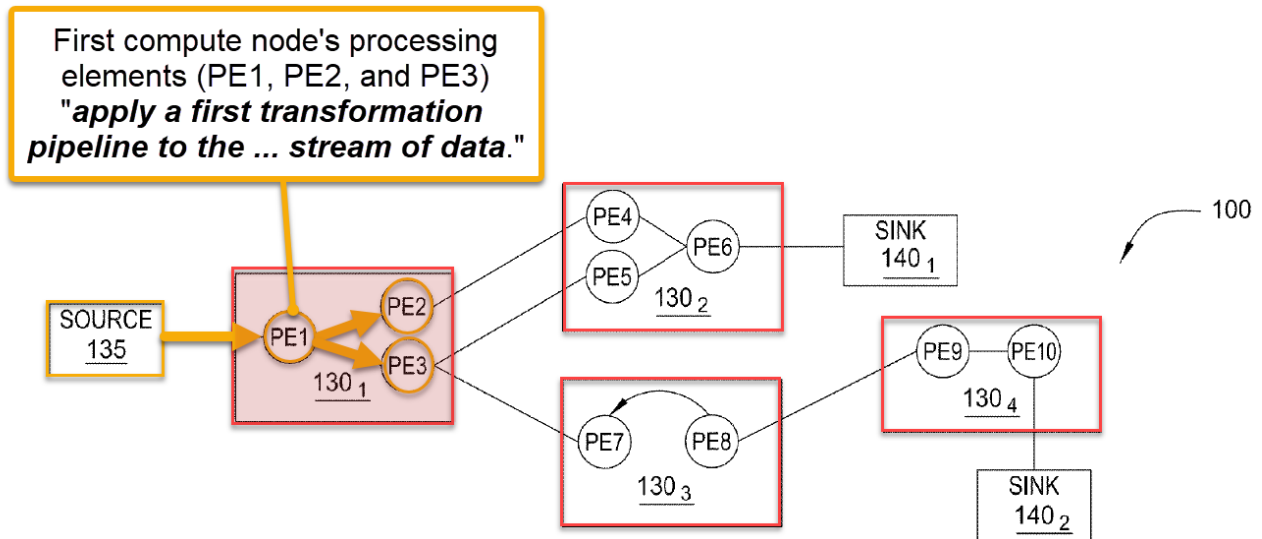


FIG. 1B

EX1004, FIG. 1B (annotations added); EX1003, ¶84.

“process first portions of the first stream of data”: As discussed regarding limitation 1[b][i], compute node 130₁’s PE1 receives a series of “data elements” from source 135 (*first stream of data*). EX1003, ¶85.

“by applying a first transformation pipeline to the first portions of the first stream of data”: 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 (“Data elements flow into a source PE of a stream application and are processed by that PE.”), 6:57-60 (“processing elements ... emit an N-tuple of data elements into the stream”). Then PE2 and PE3

process the received tuples and emit additional tuples. EX1004, 7:58-59, 7:64-67; EX1003, ¶86. 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.

Further, the series of PEs for a given compute node is a ***transformation pipeline***, as construed *supra* Section VI, because those PEs form “a progression of functions which each perform a transformation on a data stream” passing through the compute node. Barsness teaches that “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 function on the data via its one or more operators and emits the transformed data to the next PE in the graph. EX1004, 6:57-7:3 (“[E]ach processing element may be configured to carry out any form of data processing functions on a received N-tuple”); *id.*, 6:55-56, 3:16-22, 9:4-7. 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”). In Barsness’s Figure 1B example, “PE1 may split data elements received in a tuple and pass some data elements to PE2 [and]

others to PE3,” *i.e. transform the data stream. Id.*, 7:56-57. Thus, the series of PEs within a given compute node, including compute node 130₁, comprises *a progression of functions that each perform some action or transformation on the data stream. EX1003, ¶87.*

In addition, as discussed above, PO’s “plain and ordinary meaning” of the term includes the specification’s examples of transformation pipelines that are likewise *a progression of functions that each perform some action or transformation on the data stream. See, e.g., EX1001, 17:54-18:36, FIG. 7.* Thus, 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, ¶88.

For the “***process[ing]***” performed by each PE, Barsness teaches that “each processing element may be configured to carry out any form of data processing functions on a received N-tuple, including, ... data analytic functions or operations.” EX1004, 6:65-7:3. Each PE includes a “collection of operators” comprised of “executable code configured to process data flowing into a processing element.” EX1004, 9:3-9. Barsness further teaches that the PEs may “***transform*** incoming data

tuples by either adding or removing one or more data attributes before emitting outgoing data tuples to other PEs.” EX1004, 8:11-15; EX1003, ¶89.

“*to generate first pipeline output messages*”: The tuples emitted by PE2 and PE3 are the *first pipeline output messages* resulting from PE1, PE2, and PE3 (*the first transformation pipeline*):

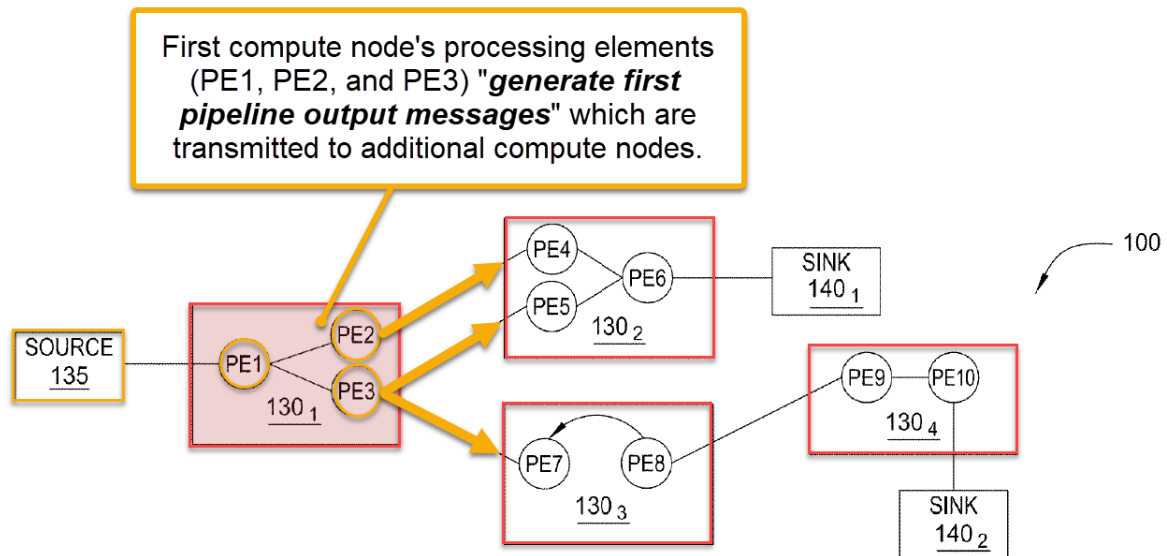


FIG. 1B

EX1004, FIG. 1B (highlighting added); *id.*, 6:62-65, 7:52-67; EX1004, 9:4-9; EX1003, ¶90. Barsness explains that “processing elements ... emit an N-tuple of data elements into the stream.” EX1004, 6:57-60.

(f) 1[b][iii] – transmit the first pipeline output messages;

Barsness discloses compute node 130₁ *transmit[s]* data tuples (*i.e. first pipeline output messages*) from PE2 and PE3 to compute node 130₂ and compute node 130₃. EX1004, 7:58-59 (“Data that flows to PE2 results in tuples emitted to PE4 on compute node 130₂”), 7:64-67 (“This example operator graph also shows

data tuples flowing from PE3 to PE7 on compute node 130₃”); EX1003, ¶91. FIG.

1B reflects the transmission with the edges from compute node 130₁’s PE2 and PE3 to compute node 130₂’s PE4 and PE5 and to compute node 130₃’s PE7.

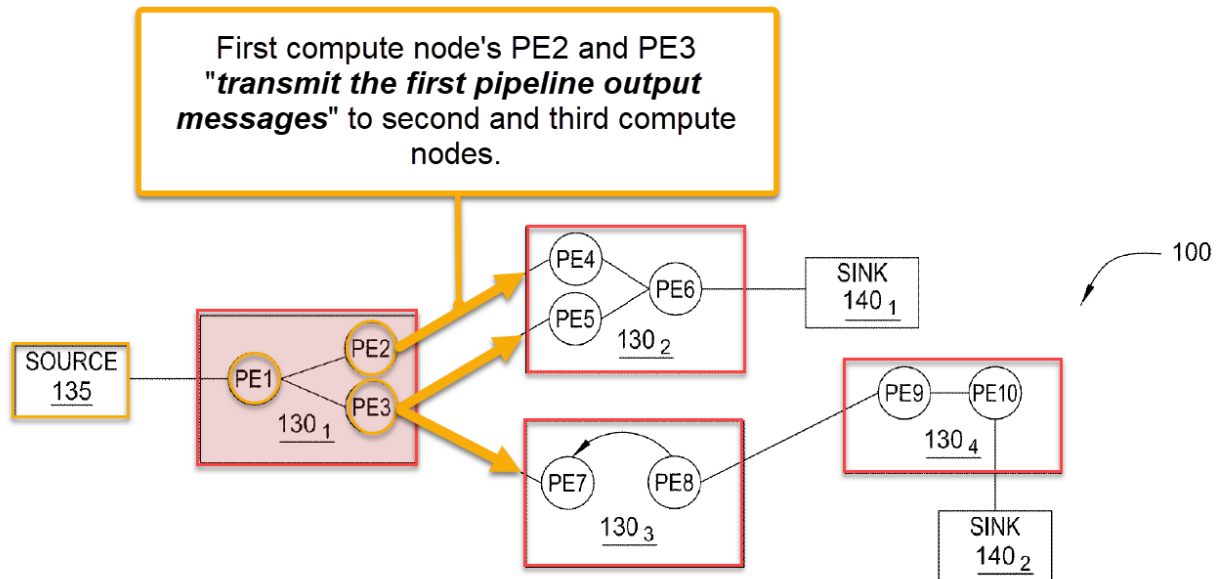


FIG. 1B

EX1004, FIG. 1B (highlighting added); EX1003, ¶91.

- (g) 1[c] – wherein a second computer system of the first plurality of computer systems is configured to execute second software instructions that:

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

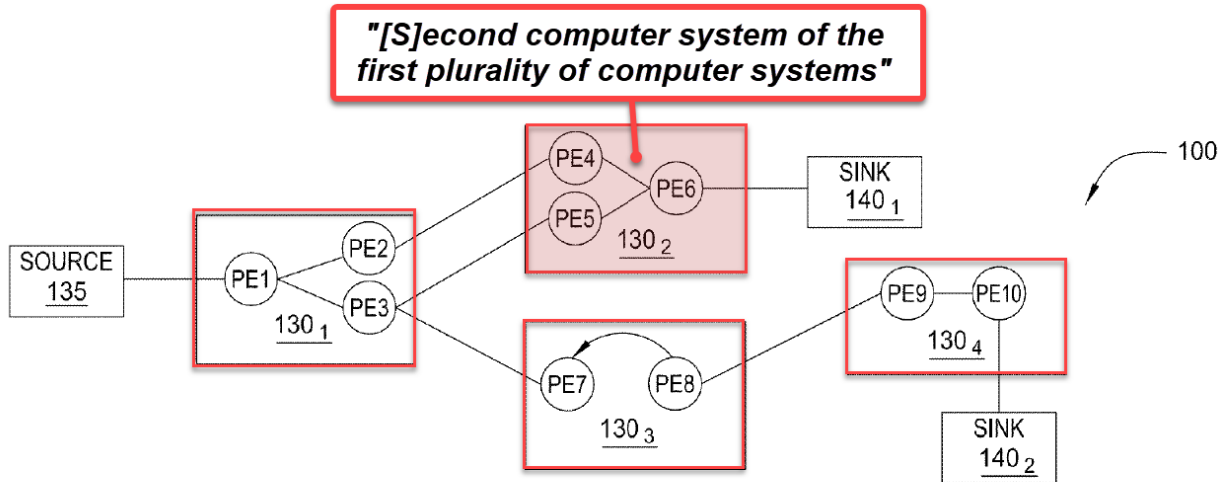


FIG. 1B

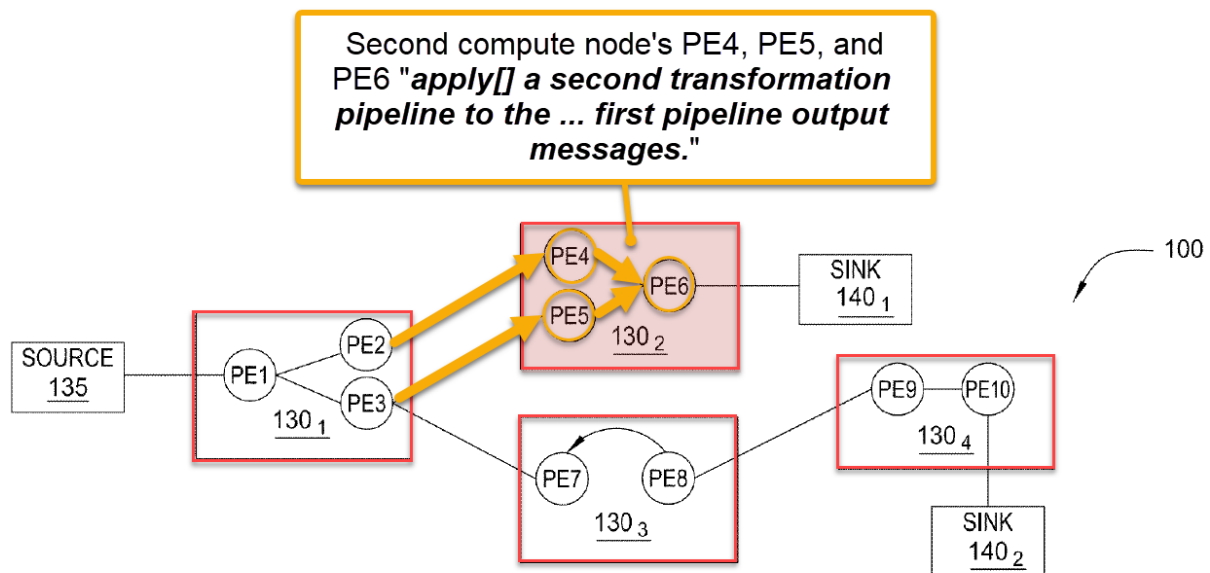
EX1004, FIG. 1B (highlighting added). As discussed regarding limitation 1[a], compute node 130₂ is one of the *first plurality of computer systems* 130₁₋₄. EX1003, ¶92.

Compute node 130₂'s CPU is *configured to execute second software instructions* through the operators of PE4, PE5, and PE6. As discussed regarding limitation 1[b], compute nodes 130₁₋₄ each contain PEs and execute code (*software instructions*) based on each of its PE's operators. EX1003, ¶93.

- (h) 1[c][i] – process first portions of the first pipeline output messages by applying a second transformation pipeline to the first portions of the first pipeline output messages;

Barsness discloses that the second compute node 130₂ (*second computer system*) processes the data tuples received from the first compute node 130₁ (*first pipeline output messages*) with PE4, PE5, and PE6 (*second transformation pipeline*). EX1003, ¶¶94-96.

“first portions of the first pipeline output messages”: Barsness discloses that the second compute node 130₂ receives the data tuples (**first pipeline output messages**) emitted by PE2 and PE3 of the first compute node 130₁. EX1004, 7:58-61. FIG. 1B’s operator graph shows the flow of output data tuples from compute node 130₁’s PE1, PE2, and PE3 (**first transformation pipeline**) to PE4, PE5, and PE6 (**second transformation pipeline**) of compute node 130₂:



EX1004, FIG. 1B (annotations added); EX1004, 7:58-61, EX1004, 7:44-47; EX1003, ¶95.

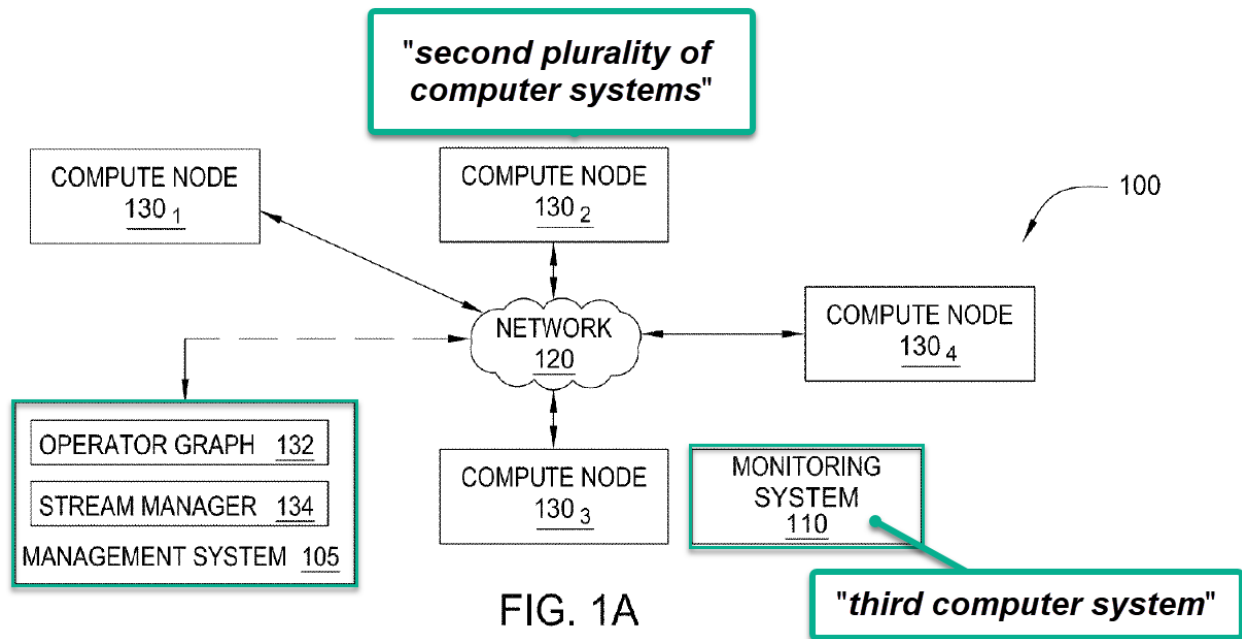
“second transformation pipeline”: Barsness discloses that PE4, PE5, and PE6 process the data tuples received from compute node 130₁. EX1004, 7:59-64 (“And data tuples emitted by PE4 flow to sink PE6 140₁. Similarly, data tuples flowing from PE3 to PE5 also reach sink PE6 140₁. Thus, in addition to being a sink for this

example operator graph, PE6 could be configured to perform a join operation for combining tuples received from PE4 and PE5.”). As discussed regarding limitation 1[b][ii], PEs comprise a “collection of operators ... configured to process data flowing into a processing element” which “carry out any form of data processing functions on a received N-tuple.” EX1004, 9:3-9, 6:65-7:3, 8:11-15. As discussed regarding limitation 1[b][ii], the series of PE4, PE5, and PE6 within a given compute node, including compute node 130₂, comprises a *transformation pipeline* because, under Petitioner’s construction, it is *a progression of functions that each perform some action or transformation on the data stream*, which also meets Patent Owner’s understanding of the plain meaning of the term. EX1003, ¶¶96-97.

- (i) **1[d] – wherein a third computer system of the second plurality of computer systems comprises a memory that stores a first data,**

“a third computer system of the second plurality of computer systems”:

Barsness teaches the monitoring system 110 (*third computer system*) “to monitor data processing components of the stream application.” EX1004, 7:15-17; EX1003, ¶¶98-99. As discussed regarding limitation 1[a], the *second plurality of computer systems* includes “management system 105” and “monitoring system 110.”



EX1004, Fig. 1A (annotations added).

Barsness discloses that monitoring system 110 is “a single computing system, [or] the monitoring system 110 may be a cluster of a plurality of computing resources, which may be distributed in the network 120.” EX1004, 7:17-21; *see also* 10:65-11:37, FIG. 4, EX1003, ¶¶99-100.

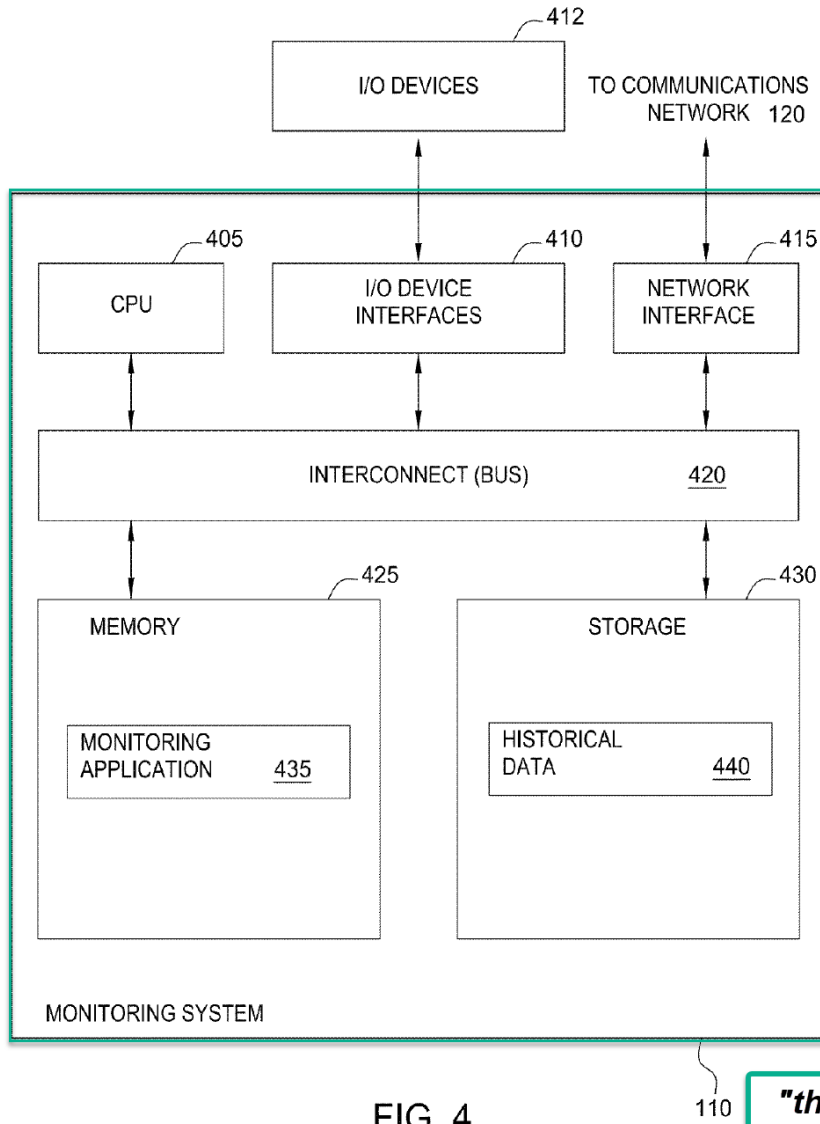
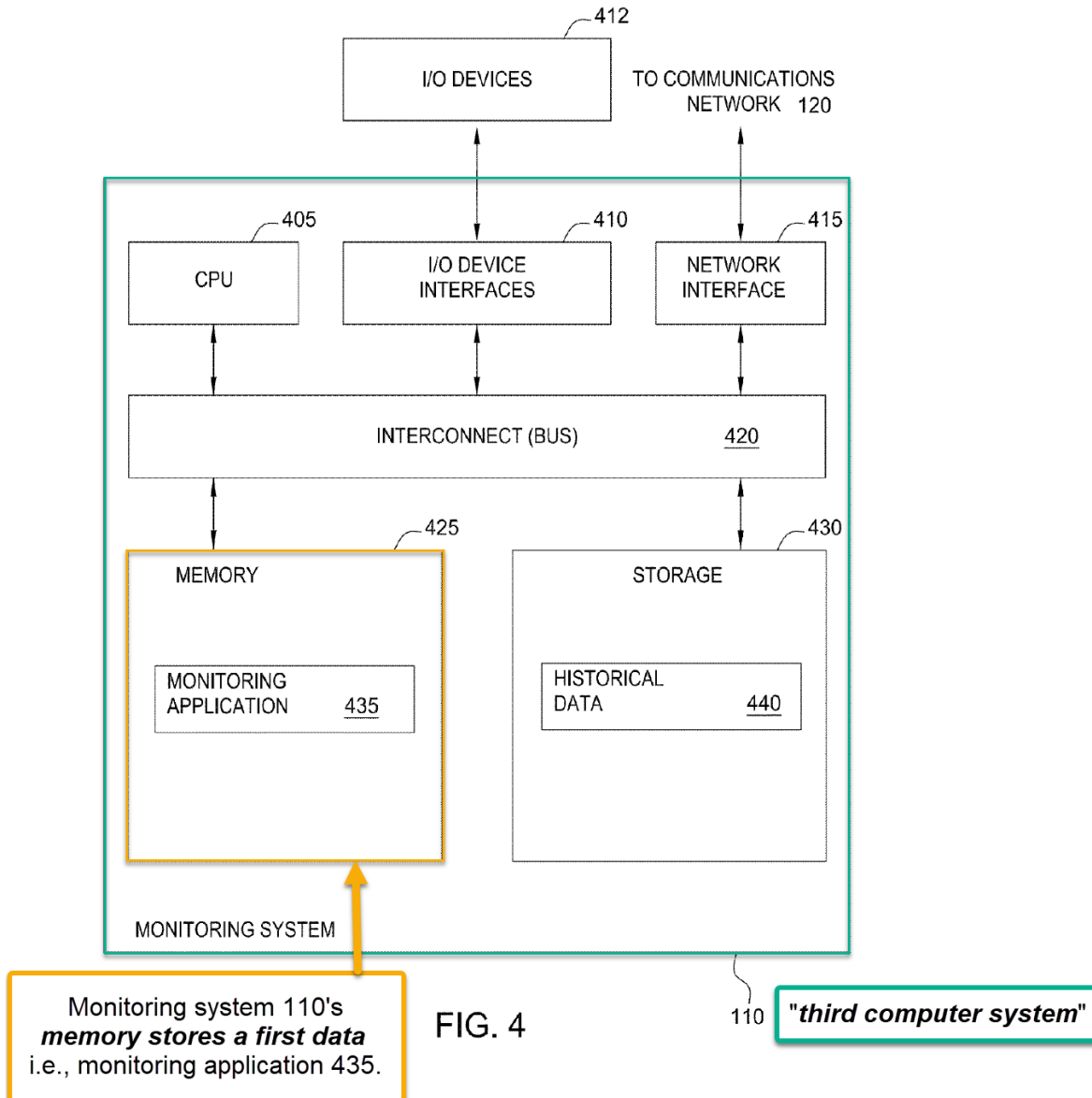


FIG. 4

EX1004, FIG. 4 (highlighting added).

“a memory that stores a first data”: Monitoring system 110’s “memory 425” stores programming instructions and application data (***a memory***). EX1004, 11:7-11; EX1003, ¶101. Barsness discloses that memory 425 “stores a monitoring application 435” (***first data***) that “may be used to monitor performance of processing

elements (PEs) based on user interactions with the user interface.” EX1004, 11:26-30, FIG. 4; EX1003, ¶101.



Barsness discloses that monitoring application 435 (**first data**) includes operator graph 132. EX1004, 3:45-50 (describing monitoring system 110’s “graphical representation of the operator graph”). 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 (“the user may be looking at the operator graph 132 of running code.”); EX1003, ¶102.

GUI 510’s depiction of operator graph 132 includes for each PE: (1) “tuple processing data,” (2) “performance metrics,” and (3) “configurations.” EX1004, 14:1-25, 11:50-12:12; EX1003, ¶103.

Tuple Processing Data: Barsness discloses that monitoring application 435 (*first data*) includes “tuple processing data” collected from each PE. Barsness teaches that compute nodes 130 “collect[] tuple processing data from each processing element 235, including operators 240.” EX1004, 13:32-34. Then, “[c]ompute nodes 130 ... transfer this collected data at pre-configured intervals to the monitoring system 110, which persistently stores the collected data.” EX1004, 13:34-37, *see also* EX1004, 12:13-21. Monitoring system 110 stores the data received from each compute node in *memory*: “the monitoring system 110 may cache the collected data in memory 425.” EX1004, 13:43-45, 13:66-67; *see also* EX1004, 9:30-33; EX1003, ¶104-105.

Barsness teaches that GUI 510 presents the “tuple processing data” as part of operator graph 132. EX1004, 11:65-12:11 (“The graphical representation of the operator graph may present ... logs ... [that] typically include pre-configured data related to the data processing of data tuples at the selected PE.”); *see also id.*, 14:21-25 (“the granularity of the collected data ... may be automatically increased for the selected operator or processing element being visualized”); EX1003, ¶106.

Performance Metrics: Monitoring application 435 (*first data*) calculates performance metrics based on the “tuple processing data” for each PE such as “average tuple rates ... e.g., average tuple rate in the last one hour at the input of PE1.” EX1004, 13:64-66; *id.*, 11:65-12:3. The performance metrics are stored in *memory* 425. EX1004, 13:66-67. Barsness teaches that GUI 510 presents the “performance metrics” as part of operator graph 132. EX1004, 14:10-25, 11:65-12:3 (“The graphical representation of the operator graph may present ... performance metrics associated with the processing elements”); EX1003, ¶107.

Configuration: Barsness discloses that monitoring application 435 (*first data*) includes the “configurations of PEs” as part of GUI 510. EX1004, 12:5-8. A POSITA would have recognized that including the “configurations of PEs” in GUI 510 requires storing those configurations in *memory*. EX1003, ¶108.

- (j) 1[d][i] – wherein the first data represents a portion of a distributed computational graph,

Barsness discloses that monitoring application 435 (*first data*) includes data *represent[ing]* operator graph 132 (*distributed computational graph*). EX1003, ¶109.

“*first data*”: As discussed regarding limitation 1[d], monitoring application 435 includes operator graph 132 combined with each PE’s: (1) “tuple processing data,” (2) “performance metrics,” and (3) “configurations.” EX1003, ¶110.

“*distributed computational graph*”: Operator graph 132 (*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; EX1003, ¶111.

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

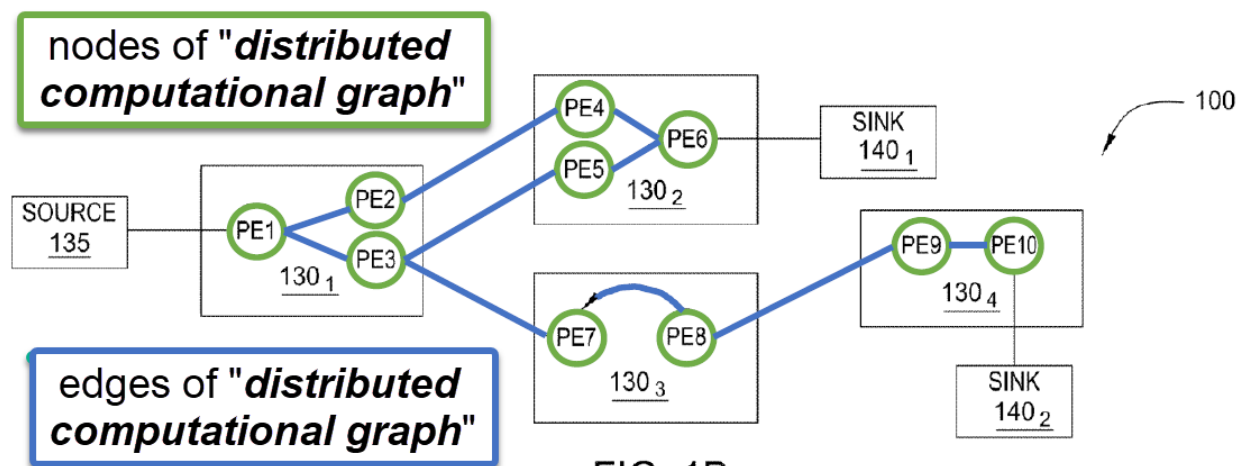


FIG. 1B

EX1004, FIG. 1B (green and blue annotations added). PEs are grouped into respective compute nodes 130₁₋₄. Each PE includes one or more operators 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, ¶112-113.

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.

- “*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, 9:29-33 (describing directed graphs); EX1003, ¶114. For example, Barsness explains that data flows from PE1 to PE2 and PE3,

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.

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”). 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, 18:43-48 (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, ¶114.

- “*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 ’425 patent provides no meaningful explanation of “output messages” but uses that term to refer to data flowing from a transformation node. *E.g.*, EX1001, 17:19-21; EX1003, ¶114,

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, ¶115.

- (k) 1[d][ii] – and wherein the portion of the distributed computational graph describes a flow of data from the first transformation pipeline to an input of the second transformation pipeline;

“distributed computational graph describes a flow of data”: As discussed regarding limitation 1[d], monitoring application 435 includes operator graph 132 combined with each PE’s: (1) “tuple processing data,” (2) “performance metrics,” and (3) “configurations” (***distributed computational graph***). Combined, this data ***describes a flow of data*** through each PE among compute nodes 130. EX1004, 13:30-37, 14:1-12; EX1003, ¶¶116-117. For example, the “tuple processing data” and “performance metrics” associated with each PE ***describe [the] flow of data*** between PEs, e.g. “average tuple rate in the last one hour at the input of PE1.” EX1004, 13:63-66; *see also id.* 13:45-55 (“tuple rates at each inputs and outputs of each of the operators 240 of processing elements (PEs of FIG. 1B).”). EX1003, ¶¶116-118.

“from the first transformation pipeline to an input of the second transformation pipeline”: As discussed regarding limitation 1[b][ii], the progression of transformations performed by PE1, PE2, and PE3 in compute node 130₁ is the ***first transformation pipeline***. EX1003, ¶¶66. And as discussed regarding limitation 1[c][i], the progression of transformations performed by PE4, PE5, and PE6 in compute node 130₂ is the ***second transformation pipeline***. EX1003, ¶¶119.

Accordingly, monitoring application 435’s operator graph 132 describes the ***flow of data*** between those transformation pipelines with the tuple processing data and performance metrics. FIG. 1B’s operator graph represents the ***flow of data*** with the edges between PE2-PE4 and PE3-PE5 respectively:

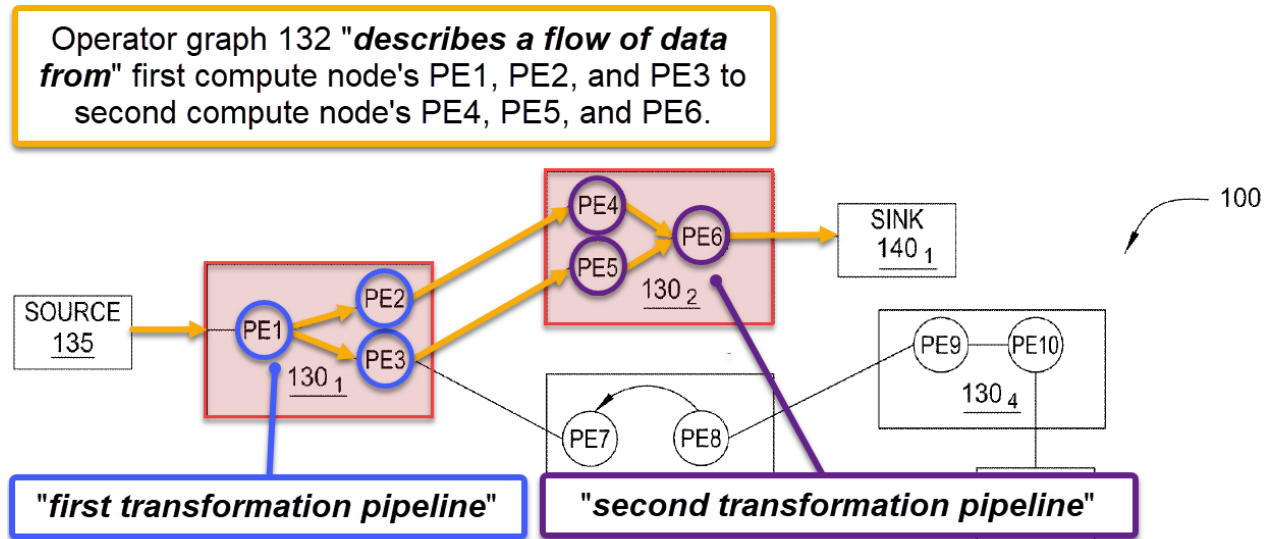


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 7:52-67 (describing the flow of data tuples through the nodes and PEs of the operator graph). EX1003, ¶¶119-120.

- (l) 1[e] – wherein the third computer system is configured to execute third software instructions that:

Barsness teaches that monitoring system 110's (*third computer system*) CPU "is configured to retrieve and execute programming instructions stored in the memory 425" and monitoring system 110's "memory 425 stores a monitoring application 435" (*configured to execute third software instructions*). EX1004, 11:7-11, 11:26-27; EX1003, ¶121.

- (m) 1[e][i] – process the first data to determine information about at least one of the first transformation pipeline and the second transformation pipeline,

Barsness discloses *processing* with monitoring application 435 *to determine information about* PE1, PE2, and PE3 on compute node 130₁ (*the first transformation pipeline*) and PE4 and PE5 on compute node 130₂ (*the second transformation pipeline*). Barsness discloses *processing the first data to determine information* under three alternatives: **Identifying Compute Nodes, Configuring Processing Elements, and Calculating Performance Metrics**. EX1003, ¶122.

Identifying Compute Nodes: Monitoring system 110 *process[es]* monitoring application 435's operator graph 132 to *determine* the compute nodes 130 for each PE (*information*). Barsness teaches that monitoring system 110 collects data from compute nodes 130 and arranges the data according to the associated PE. EX1004, 11:38-49, 13:45-47, 13:56-67. To collect data from each compute node 130, "monitoring system 110 is configured to communicate with compute nodes 130₁₋₄." EX1004, 7:21-22; *see also id.*, 7:17-21, 11:18-20, 6:45-48, FIGS. 1B, 2, and 4. Monitoring system 110 "interact[s] with the data collector 250 [of each compute node 130] either continuously or at pre-configured intervals to retrieve data." EX1004, 9:29-32. A POSITA would have recognized that communicating and interacting with compute nodes 130₁₋₄ to collect this data requires *process[ing]* monitoring application 435's operator graph 132 to identify the PEs associated with each compute node 130 from which to collect the data. The identification of the compute nodes 130 is determined for each PE (*determining information*) including

compute node 130₁'s PE1, PE2, and PE3 and compute node 130₂ PE4 and PE5 (*about at least one of the first transformation pipeline and the second transformation pipeline*). EX1003, ¶123.

Configuring Processing Elements: Monitoring system 110 *process/es/* monitoring application 435's "configuration" of PEs to then "configure[]" compute nodes 130 to collect the data accordingly." EX1004, 13:49-55. As discussed regarding limitation 1[d], monitoring application 435 includes configuration for each compute node 130 and PE. For example, Barsness discloses that "the monitoring application could turn on additional tracing in the log file or set of log files that are connected to a given PE. Likewise the monitoring application could remove tracing (or step down a logging level)." EX1004, 4:25-30; *see also id.* 12:3-12. Monitoring system 110 *process/es/* the configurations of PEs to *determine* the "granularity of the collected data ... based on occurrences of pre-configured events" (*information*). EX1004, 13:53-55. The granularity of the collection data is *determined* for each PE (*determining information*) including compute node 130₁'s PE1, PE2, and PE3 and compute node 130₂ PE4 and PE5 (*about at least one of the first transformation pipeline and the second transformation pipeline*). EX1003, ¶124.

Calculating Performance Metrics: Monitoring system 110 *process/es/* monitoring application 435's "tuple processing data" (*first data*) to calculate "performance metrics" associated with each PE (*determining information*).

Barsness discloses that monitoring system 110 collects “tuple processing data” for each PE. EX1004, 13:32-36. Then, monitoring system 110 “formats and transforms the collected data according to pre-set configuration” to calculate, for example, “average tuple rates within a pre-configured time interval[.]” EX1004, 13:56-67. Monitoring system 110 uses these performance metrics to compare the current “tuple processing rates” against “historical tuple processing rates.” EX1004, 12:40-54. The performance metrics are determined for each PE (*determining information*) including compute node 130₁’s PE1, PE2, and PE3 and compute node 130₂ PE4 and PE5 (*about at least one of the first transformation pipeline and the second transformation pipeline*). EX1003, ¶125.

- (n) 1[e][ii] – monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions in accordance with the determined information, and

Barsness discloses monitoring system 110 *monitor[ing]* the execution of each PE on compute nodes 130. EX1004, 7:15-17; *see also, id.*, 9:29-33, 11:27-30. Specifically, monitoring system 110 continuously collects “tuple processing data” from each of the compute nodes 130 as part of a method for “monitoring a stream application.” EX1004, 13:32-34, 14:7-9; *see also id.*, EX1004, 9:23-27, 12:13-21. The collected data includes “tuple rates at each inputs and outputs of each of the operators 240 of processing elements.” EX1004, 13:32-47. Monitoring system 110’s GUI 510 “continuously monitor[s] the current tuple processing rates at various

processing elements and compare[s] the current tuple processing rates with the historical tuple processing rates.” EX1004, 13:4-7. Therefore, monitoring system 110 *monitor[s]* “tuple processing rates” based on the inputs and outputs of each PE. EX1003, ¶126.

“the execution of at least one of the first software instructions and the second software instructions”: Monitoring system 110 *monitor[s]* each PE’s execution (*first software instructions and the second software instructions*) through “tuple processing rates.” EX1004, 12:41-45. Barsness explains that a processing element could be “receiving 100 tuples per second at the input but only producing 30 tuples at the output,” which if “the selected processing element in the last one hour was processing on an average 70 tuple, the current processing of 30 tuples per second may be an indication of performance issues.” EX1004, 12:45-54. Thus, the monitoring system monitors the rate of the PEs’ *execution of the first and second software instructions*. As discussed regarding limitation 1[b], compute node 130₁’s CPU executes the code for PE1, PE2, and PE3 (*first software instructions*). As discussed regarding limitation 1[c], compute node 130₂’s CPU executes the code for PE4 and PE5 (*second software instructions*). The “tuple processing rates” measure the rate of processing (*execution of ... software instructions*) of each of those PEs in terms of data tuples. EX1003, ¶127.

Identifying Compute Node - “*in accordance with the determined information*”: The monitoring of “tuple processing rates” is *in accordance with* the PEs identified from monitoring application 435’s operator graph 132 (*determined information*). Monitoring system 110 collects the tuple processing rates for each PE from compute nodes 130. EX1004, 13:45-47. Therefore, the “tuple processing rates” collected from each compute node 130 *accord[]* with a PE monitoring system 110’s operator graph 132. EX1003, ¶128.

Configuring Processing Elements - “*in accordance with the determined information*”: The monitoring of “tuple processing rates” is *in accordance with* the configuration of each compute node 130 *determined* from monitoring system 110’s PE configurations (*information*). EX1004, 13:49-55 (“[A] user configures the monitoring system 110 with regards to the type of data to be collected. The monitoring system 110 configures compute nodes 130 to collect the data accordingly.”), 14:21-25; EX1003, ¶129.

Calculating Performance Metrics - “*in accordance with the determined information*”: The monitoring of “tuple processing rates” is *in accordance with* the performance metrics calculated for each PE (*determined information*). Monitoring system 110 includes historical data 440 retrieved from each compute node 130’s PEs. EX1004, 11:37-49, 12:13-21. This historical data 440 is analyzed along with the “current tuple processing rates” to identify any issues in the tuple processing

rates for a PE. 12:37-50 (“the data analysis module 530 may compare these rates with the historical processing rates (or historical processing averages) to determine if there is any issue with the selected processing element”); *see also id.* 13:4-12, 14:26-35; EX1003, ¶130.

- (o) **1[f] – in response to the monitoring, cause a fourth computer system of the third plurality of computer systems to execute fourth software instructions that perform at least one of:**

Barsness teaches monitoring system 110 (*third computer system*) is configured to provision and configure additional compute nodes 130_x (*fourth computer system of the third plurality of computer systems*) to the stream application when it identifies discrepancies in the monitored PE’s tuple processing rates compared to its historical tuple processing rates (*in response to the monitoring*). EX1003, ¶131.

“*in response to the monitoring*”: Barsness teaches that while monitoring the tuple processing rates of each PE, “monitoring system 110 determines if the current tuple processing rate is outside a pre-configured range based on the historical tuple processing rate averages.” EX1004, 14:25-29. “[I]f the current tuple processing rate is above or below the pre-configured tuple processing rate, the monitoring system 110 may take pre-configured corrective actions.” EX1004, 14:31-35; *see also id.* EX1004, 12:37-13:3, 13:4-14, 13:22-29. Monitoring system 110’s “corrective actions” includes “spawning new compute nodes, [and] duplicating a compute

node.” EX1004, 14:51-56; *see also id.*, 10:28-39, 11:30-34, 12:24-28, 12:54-67, *cls.*

2, 5. Monitoring system 110 interacts with management system 105 to perform these corrective actions. EX1004, 10:29-34; EX1003, ¶132-33.

“cause a fourth computer system of the third plurality of computer systems to execute fourth software instructions”: Monitoring system 110 causes new compute nodes 130_x (***third plurality of computer systems***) to “spawn” to provide additional computing resources to the stream application, *i.e.* ***execute fourth software instructions***. EX1004, 10:28-34. Each of the spawned compute nodes 130_x is a ***fourth computer system***. Barsness teaches using processing elements “in parallel” to process the same type of data (*id.*, 10:26-28), which results when a new compute node is assigned the same processing elements as an existing compute node. Barsness further explains that these corrective actions maintain expected performance levels. *Id.*, 14:30-57; EX1003, ¶134-136.

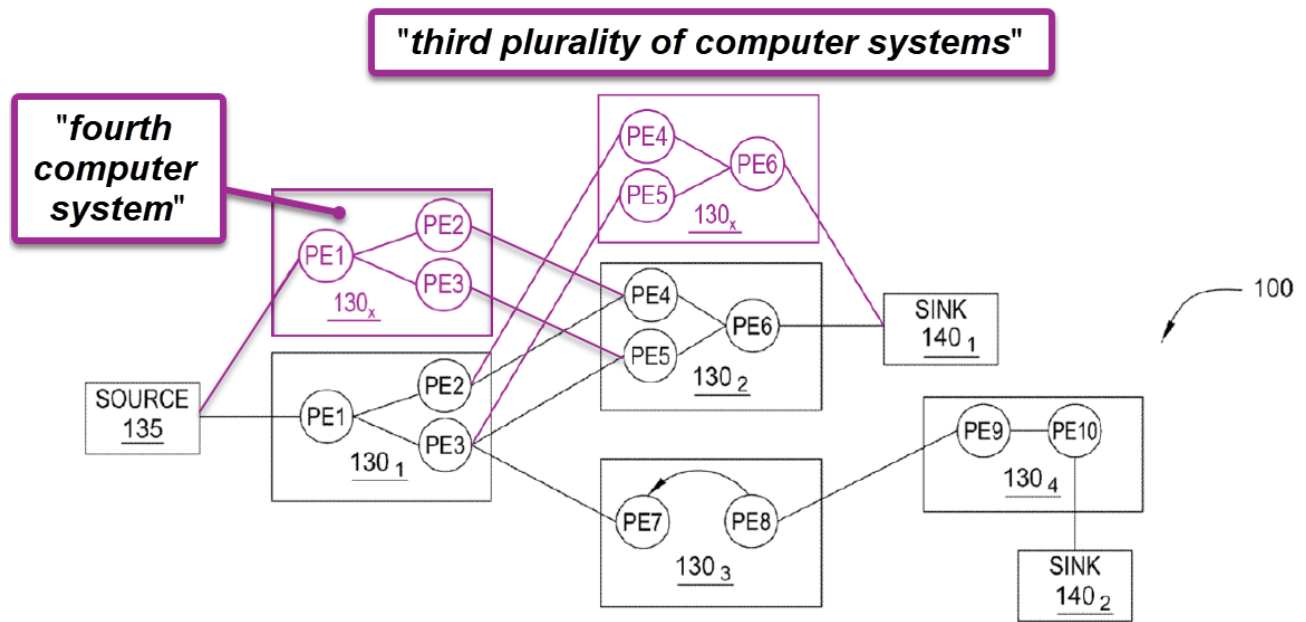


FIG. 1B

EX1004, FIG. 1B (annotations added).

These “new” or “duplicat[e]” compute nodes 130_x comprise a *third plurality of computer systems*. Barsness discloses that compute nodes 130 may be virtual machines allowing the system to “spawn more virtual machines on the fly.” EX1004, 8:30-52. Further, Barsness teaches the use of “cloud computing infrastructure” for “the provision of scalable computing resources” including “the nodes used to create a stream application may be virtual machines hosted by a cloud service provider.” EX1004, 6:16-42. A POSITA would have recognized these new compute nodes as a *third plurality of computer systems*. EX1003, ¶137.

- (p) 1[f][i] – processing second portions of the first stream of data by applying the first transformation pipeline to the second portions of the first stream of data, or

1[f][ii] – processing second portions of the first pipeline output messages by applying a second transformation pipeline to the second portions of the first pipeline output messages,

Barsness teaches monitoring system 110 *cause[s]* the provisioning of 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. Compute node 130_x processes the same type of stream data in parallel, “increas[ing] data throughput” (*id.*, 10:26-28) and maintaining the desired “processing rate” for those PEs (*id.*, 14:54-56, 14:30-39). The following annotated FIG. 1B shows compute node 130_x (*fourth computer system*) configured with the processing elements of the first compute node 130₁ (*first transformation pipeline*):

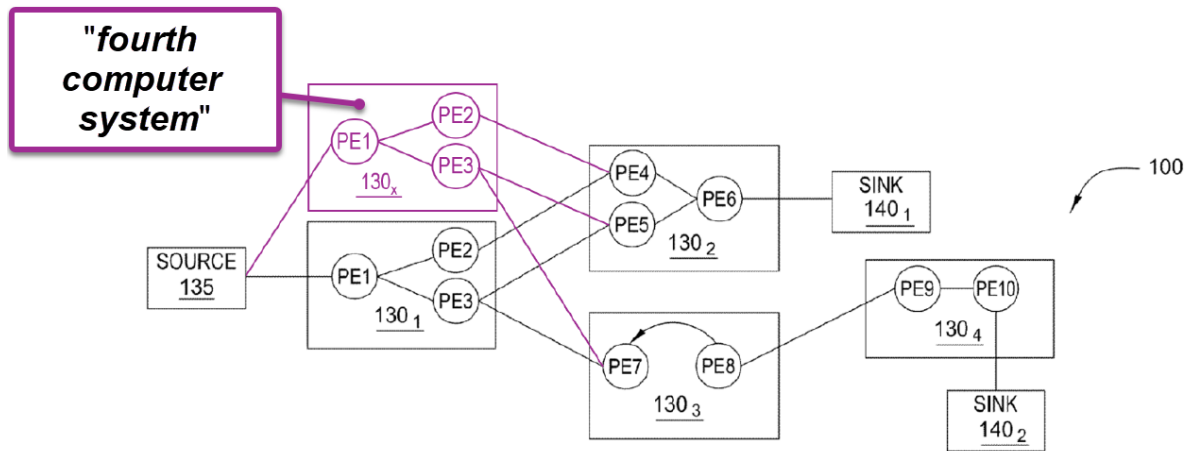


FIG. 1B

EX1004, FIG. 1B (purple annotations added); EX1003, ¶138.

The same Barsness teachings apply to the *second transformation pipeline*: if performance issues arise for compute node 130₂, the monitoring system 110 *cause[s]* the provision compute node 130_x and configures it with the processing elements of compute node 130₂ (*second transformation pipeline*). EX1004, 10:34-39; *id.*, 10:28-34, 14:51-56. As shown in annotated FIG. 1B:

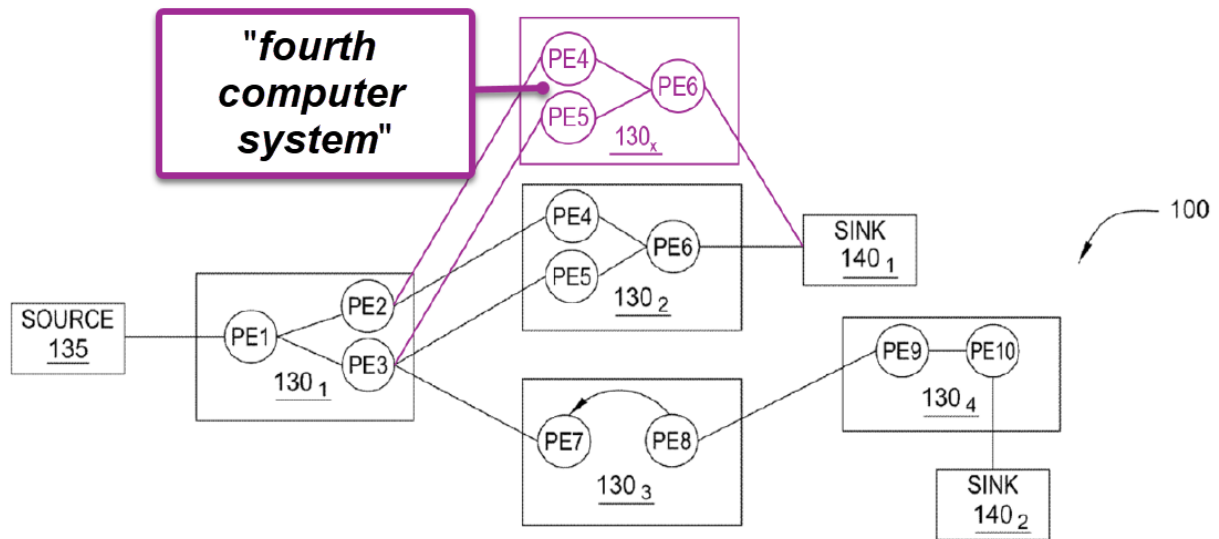


FIG. 1B

EX1004, FIG. 1B (purple annotations added); EX1003, ¶139. 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 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 data processing rates. EX1004, 10:26-39, 14:26-56; EX1003, ¶140.

Barsness teaches that the new, duplicated compute nodes perform the same processing as the computing nodes they duplicate. As explained in limitations 1[b][ii] and 1[c][i], the first compute node 130₁ processes *the first stream of data by applying the first transformation* pipeline and the second compute node 130₂ processes *the first pipeline output message by applying a second transformation pipeline*. Barsness teaches that the new, duplicated compute nodes process a different, *second portion* of the respective data, because they process the “same type of data to **increase data throughput.**” EX1004, 10:26-28 (emphasis added). 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, ¶141.

(q) **1[g] – wherein the first computer system and the third computer system are distinct, and**

Barsness discloses that compute nodes 130 and monitoring system 110 are *distinct computer system[s]*. As discussed regarding limitations 1[a] and 1[b], Barsness discloses these computer systems as parts of distinct pluralities of computer systems. Barsness describes compute nodes 130 as physical or virtual machines with CPU 205, network interface 215, memory 225, and storage 230. EX1004, 8:30-65, FIG. 2. Barsness describes monitoring system 110 as comprising CPU 405, network

interface 415, memory 425, and storage 430. 10:65-11:25, FIG. 4. Further, Barsness discloses that “monitoring system 110 is configured to communicate with compute nodes 130₁₋₄.” EX1004, 7:21-22; EX1003, ¶142.

(r) 1[h] – wherein the third computer system and the fourth computer system are distinct.

As explained regarding limitation 1[f], Barsness discloses *the fourth computer system* through monitoring system 110 (*third computer system*) “spawning new compute nodes, duplicating a compute node” where the new/duplicated compute node 130 is the *fourth computer system*. EX1004, 14:40-56. Barsness makes the same *distinction* between compute nodes 130 and monitoring system 110 as discussed above regarding limitation 1[g]. EX1003, ¶143.

3. Claim 2 – The system of claim 1, wherein the third software instructions monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions to identify execution that has become unresponsive.

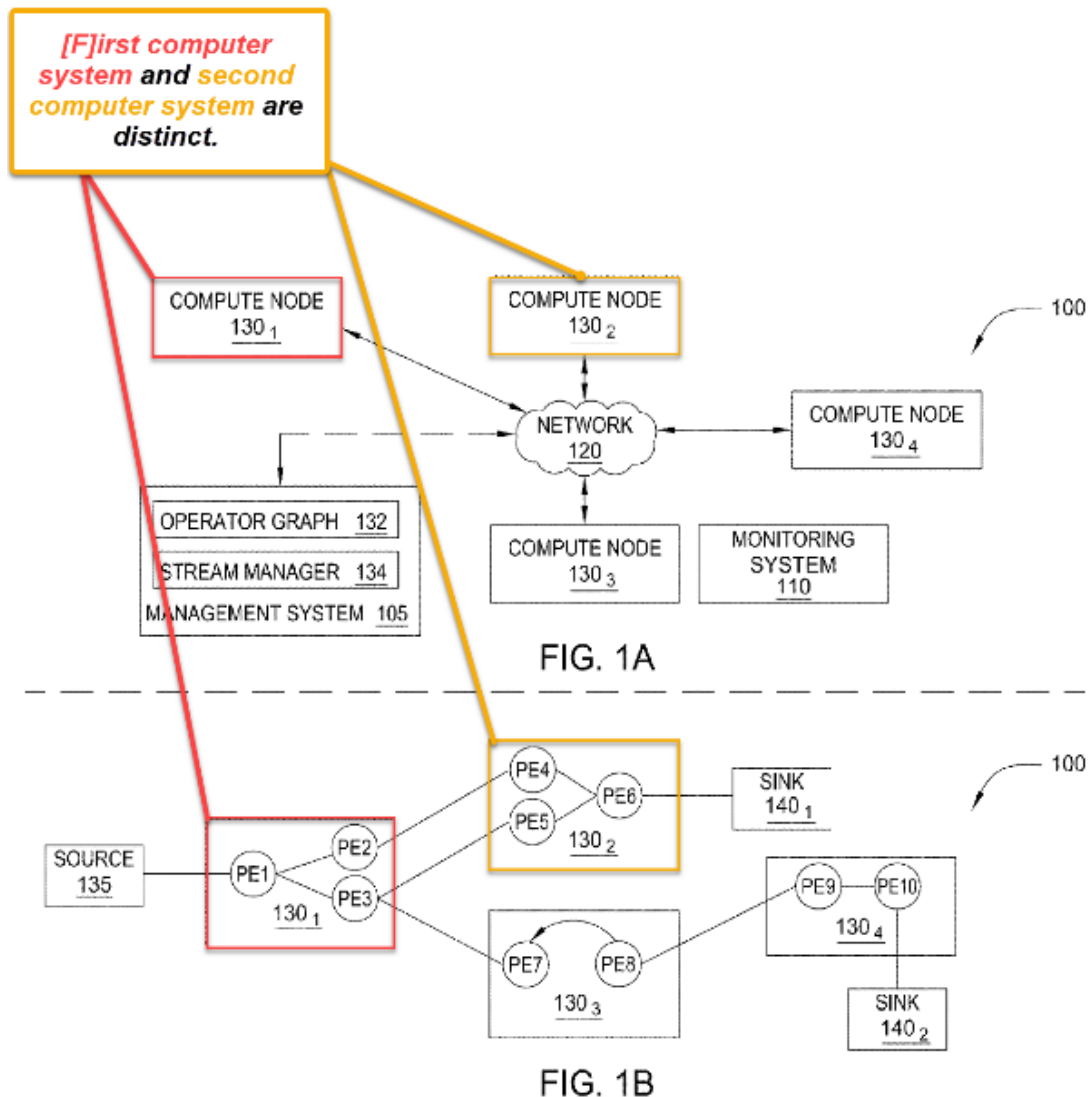
As discussed regarding limitation 1[e], Barsness teaches that monitoring system 110 monitors the data tuple rates for PEs. A POSITA would have recognized that data tuple rates of zero may *identify execution that has become unresponsive*. Barsness teaches that its disclosure addresses known challenges with both “monitoring and fault detection.” EX1004, 3:7-15; EX1003, ¶144, 146.

Alternatively, Barsness teaches that “processing errors ... may also be a factor on which the monitoring application 435 may act and reconfigure the processing

modules of the stream application.” EX1004, 12:31-36. A POSITA would have recognized that processing errors would *identify execution that has become unresponsive*. Additionally, Barsness discloses “tracing in log files” as one form of the monitoring which a POSITA would have recognized as *identify[ing] execution that has become unresponsive*. EX1003, ¶145-146.

4. Claim 3 – The system of claim 1, wherein the first computer system and the second computer system are distinct.

Barsness discloses that compute node 130₁ (*first computer system*) and compute node 130₂ (*second computer system*) are *distinct*. As described regarding limitation 1[], Barsness teaches a “computing infrastructure 100” that includes “a plurality of compute nodes 130₁₋₄, each connected to a communications network 120.” EX1004, 6:43-48. This plurality of computer systems includes distinct computer systems for compute nodes 130₁ (*see discussion of first computer system for limitation 1[b]*) and 130₂ (*see discussion of second computer system for limitation 1[c]*)). FIG. 1A shows an exemplary computing infrastructure and its plurality of compute nodes 130:



EX1004, FIG. 1A; EX1003, ¶147.

5. Claim 4

(a) 4[pre] – The system of claim 1,

4[a] – wherein the first computer system and the second computer system each further comprise one or more memories that each store a respective second data,

Barsness teaches that compute node 130₁ (*first computer system*) and 130₂ (*second computer system*) each comprise “memory 225” (*memories*) that “store[s]”

data, e.g., programming instructions and “application data” (*stores a respective second data*). EX1004, 8:53-55. Each computer system also includes “storage” 230 that stores data, e.g., buffered stream data 260 and operator graph 132. EX1004, 8:61-65, 9:43-50, 10:3-11. The memory and storage (each alone and collectively, *memor[y]*) of a compute node 130 are depicted, in FIG. 2:

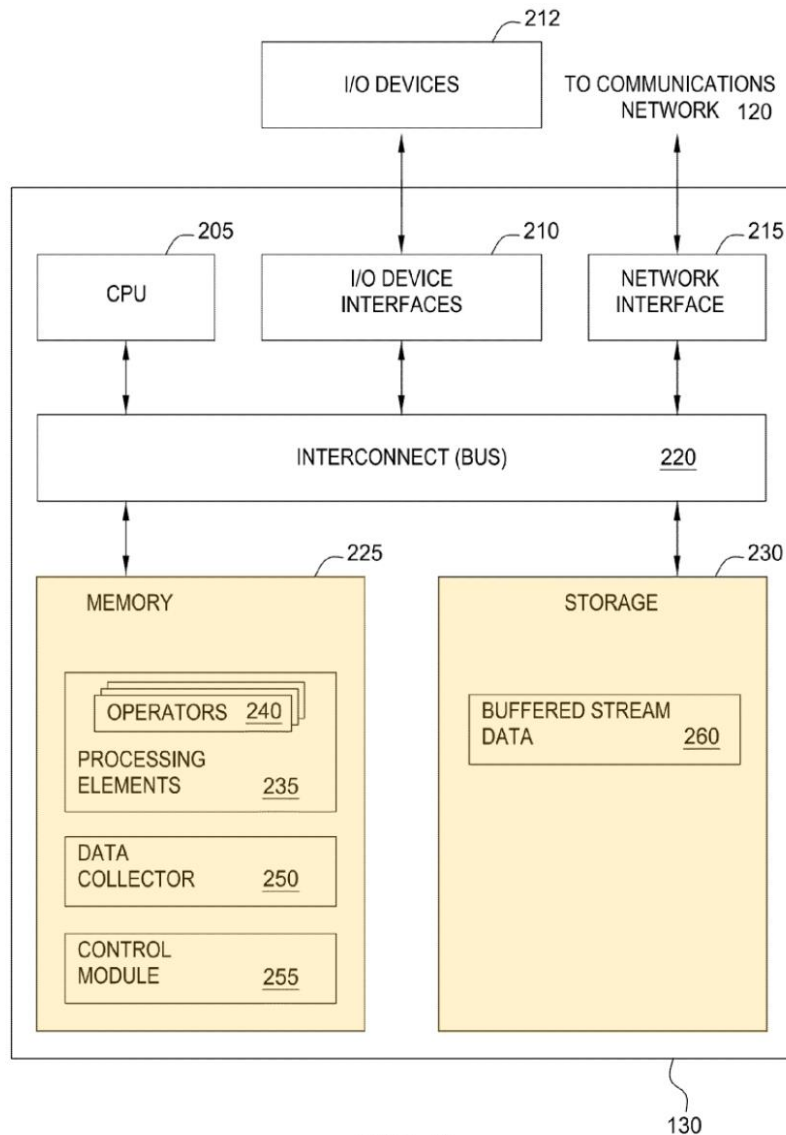


FIG. 2

EX1004, FIG. 2 (highlighting added); EX1003, ¶148.

Respective “data” stored in each respective memory of Barsness includes a *respective second data*. EX1003, ¶149.

(b) 4[b] – wherein the respective second data represents at least a respective portion of the distributed computational graph,

Barsness discloses that each compute node 130 stores at least the portion of operator graph 132, specifically the portion of the operator graph 132 comprising the PEs configured to run on the compute node 130. Within each compute node 130, memory 225 stores PEs 235 (*respective second data*). EX1004, 8:66-9:1; *id.*, FIG. 2. As each PE is a node in operator graph 132, in storing a given compute node’s respective PEs, each compute node memory thus stores data *represent[ing] ... a respective portion* of operator graph 132. EX1004, 7:52-67; *id.*, 1:39-41, 1:55-57, 2:3-4, 9:1-33 (describing PEs on compute nodes); 3:19-22; 10:49-56. For example, operator graph 132 of FIG. 1B shows how first compute node 130₁ includes PE1, PE2, and PE3, and thus stores data representing a respective portion of the operator graph; similarly, the other compute nodes 130₂₋₄ store respective processing elements as depicted in operator graph 132. EX1004, FIG. 1B; EX1003, ¶150.

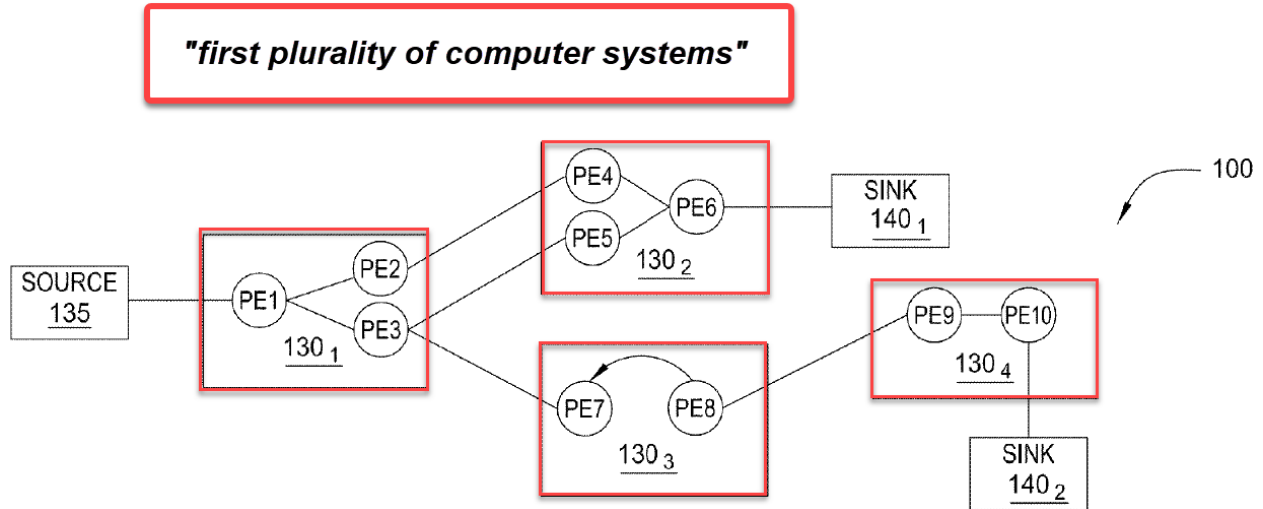


FIG. 1B

As discussed regarding limitation 1[d][i], operator graph 132 *represents ... the distributed computation graph*. EX1003, ¶151.

- (c) 4[c] – and wherein the first software instructions transmit the first pipeline output messages according to the flow of the distributed computational graph.

Barsness discloses that compute node 130₁ *transmit[s]* PE2 and PE3's output data to compute node 130₂ *according* to the *flow* described in operator graph 132 (*distributed computational graph*). EX1004, 3:16-31, 6:43-56, 7:38-8:29; EX1003, ¶152.

As discussed regarding limitation 1[b][ii], compute node 130₁'s PE1, PE2, and PE3 apply the first transformation pipeline which results in the *first pipeline output messages*.

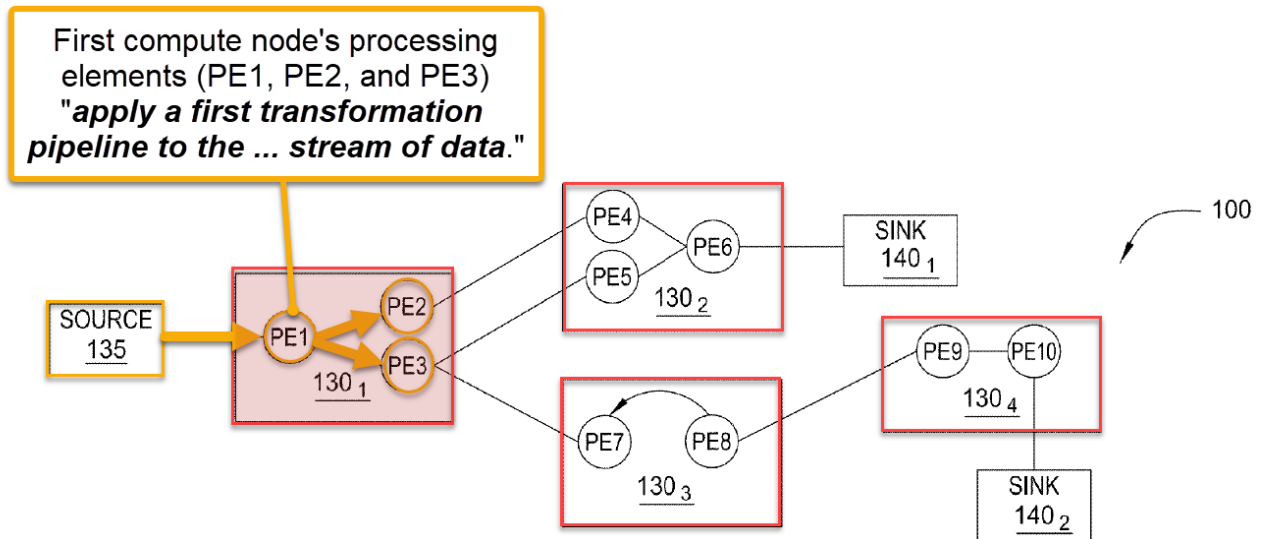


FIG. 1B

EX1004, FIG. 1B (highlighting added); EX1003, ¶153.

As discussed regarding limitation 1[b][iii], PE2 and PE3 emit data tuples (*first pipeline output messages*) that compute node 130₁ *transmit[s]* to compute node 130₂'s PE4 and PE5.

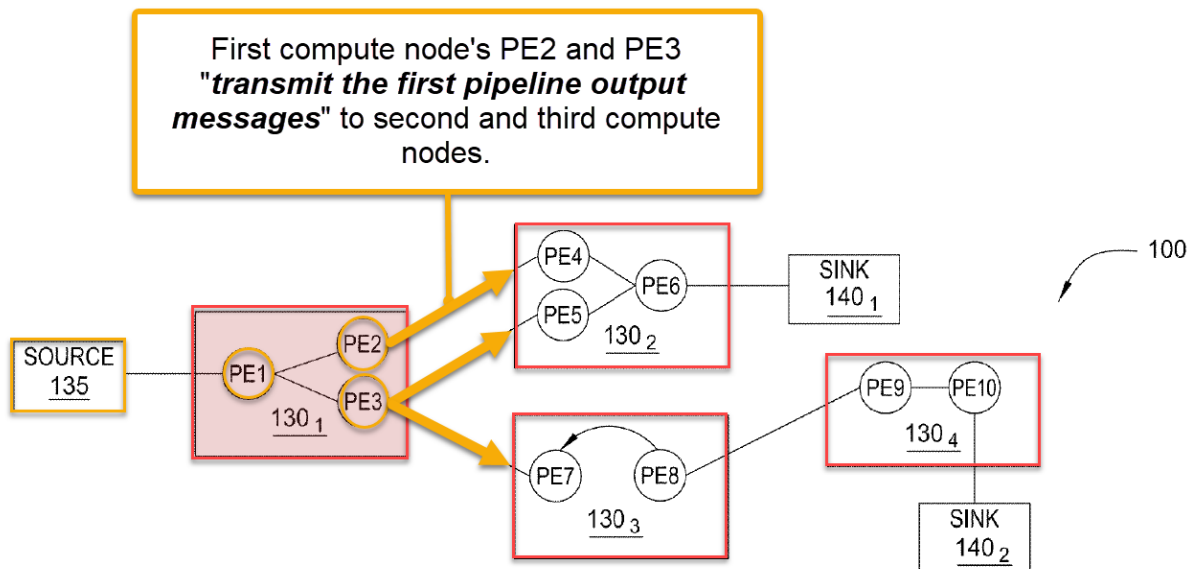


FIG. 1B

EX1004, FIG. 1B (highlighting added); EX1003, ¶154.

As discussed regarding limitation 1[b], Barsness teaches that each compute node's CPU executes "executable code" (*software instructions*) associated with operators of the PEs for that compute node. For example, the CPU of 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; EX1003, ¶155.

6. **Claim 5 – The system of claim 1, wherein the third software instructions are further configured to monitor at least a portion of the execution of at least one of the first software instructions and the second software instructions, to identify a transformation pipeline that causes at least one of the first computer system and the second computer system to transmit information that is outside established data stability boundaries.**

As discussed regarding limitation 1[f], Barsness discloses that monitoring system 110 monitors the tuple processing rates which measure the input and output of PEs to identify when the rates are "outside a pre-configured range based on the historical tuple processing rate averages" (*data stability boundaries*). EX1004, 14:25-29; EX1003, ¶156.

7. **Claim 6 – The system of claim 5,**
 - (a) **6[a] – wherein the third computer system of the second plurality of computer systems is further configured to execute fifth software instructions that,**

upon identifying transmitted information that is outside established data stability boundaries:

6[b] – determine changes to at least one of the first transformation pipeline and the second transformation pipeline in response to the identifying,

6[c] – and cause at least one of the first computer system and the second computer system to execute sixth software instructions that modify a transformation pipeline in response to the determined changes.

As discussed above regarding limitation 1[f] and claim 5, Barsness discloses that monitoring system 110 (*third computer system*) *identifies* when the tuple processing rates are “outside a pre-configured range” (*data stability boundaries*). EX1004, 14:25-29. Monitoring system 110’s changes to the PEs and operators of compute node 130₁ or 130₂ are the *fifth software instructions* and are distinct operations from the spawning or duplicating compute nodes discussed regarding limitation 1[f]. EX1003, ¶157.

Barsness teaches that monitoring system 110 *determines* the corrective actions (*changes ... in response to the identifying*) to resolve performance issues of PEs identified by monitoring the associated tuple processing rates. Specifically, Barsness discloses that monitoring system 110 “may interact with the selected processing element to determine if adding more operators 240 ... [or] adding more processing elements ... [or] rearranging some processing elements on compute nodes 130 may remedy the issue.” EX1004, 12:54-13:3; EX1003, ¶158.

Barsness further discloses performing those “corrective actions” for the relevant PEs including: “automatic adjustment of an operator within compute nodes 130, automatic fusion or separation of operators, automatic duplication of an operator, ... spawning new processing elements,” “add one operator in the processing element in question,” or “reconfigure the processing modules.” EX1004, 14:35-39, 14:51-56, 13:24-29, 12:31-36. These changes are implemented by compute node 130₁ or 130₂’s control module 255 (*sixth software instructions*). EX1004, 9:34-42 (“control module 255 may be included in the compute node 130 to enable the monitoring module 110 to configure the computer node 130 based on the occurrence and detection of pre-configured conditions in the computer node 130” including adding “more processing elements 235 or operators 240”); EX1003, ¶159.

B. Ground 2: Claims 1-6 Are Obvious Over Barsness in View of Chakradhar

As explained above, Barsness alone renders claims 1-6 obvious. *Supra* Section VII.A. These claims are further obvious in view of Chakradhar, which further details how the storage and use of a claimed *distributed computational graph* was known in the art. Specifically, Chakradhar describes that each node of a distributed computing system maintains a route-map that describes the route for processing a message. Chakradhar explains the benefits of using route-maps, and as explained below, a POSITA would be motivated to combine them with Barsness. EX1003, ¶160.

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.

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, ¶¶53-54.

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, ¶55.

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, ¶¶56-57. 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-VII.B.4), the Chakradhar Provisional supports all relied-upon disclosure in Chakradhar. EX1003, ¶¶57-58.

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, ¶161.

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, ¶162.

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, ¶163.

Applying Chakradhar's route-maps to Barsness would have allowed each compute node and the monitoring system 110 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, ¶164.

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, 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, ¶165.

3. Claim 1

As discussed *supra* Section VII.A, claim 1 is obvious in view of Barsness alone. Claim 1 is further obvious in view of the Barsness-Chakradhar combination. EX1003, ¶166.

(a) 1[a]

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” (*distributed computing network*). 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]; *see also id.*, [0046] (“The orchestration of cluster resources among these systems is handled by a global cluster scheduler such as Mesos.”),

[0069] (“route-maps are installed in a consistent manner on a running cluster”), [0098] (“nodes in a cluster”), [0148] (evaluation of techniques “conducted on a 30-node cluster”); EX1006, 4/22, 6-7/22, 11/22. The clusters disclosed by Chakradhar are comprised of distributed computer systems communicating over a network (*distributed computing network*). EX1005, [0069]. Therefore, the Barsness-Chakradhar combination includes a *distributed computing network*. EX1003, ¶167.

(b) 1[d][i]

Chakradhar explains that “stream engines code an application workflow as a *directed* acyclic graph (DAG) of operators, referred to as a *topology*” (*distributed computational graph*). 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) (“As noted—with Storm—a topology is a graph whose nodes are operators (spouts and bolts) and edges are virtual connections among operators.”); *id.*, [0106] (describing “a complete directed graph, where nodes denote machines hosting learner operators and edges represent potential overlay links”); EX1006, 5/22, 8/22; EX1003, ¶168.

4. Claim 4

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 *second data* as recited in claim 4. EX1003, ¶169.

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 second data represents at least a respective portion of a distributed computational graph*. EX1003, ¶170.

C. Ground 3: Claims 1-6 Are Obvious Over Barsness in View of Siripurapu

As explained above, Barsness alone renders claims 1-6 obvious. *Supra* Section VII.A. These claims are further obvious in view of Siripurapu, which provides further details of how the claimed monitoring and provision of additional computer systems was known in the art. Specifically, Siripurapu describes monitoring data flows within a distributed computing system and adding computer nodes to accommodate increased computational requirements, additional data streams, and increased data stream flows. EX1003, ¶171.

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).

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, ¶¶59-60.

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, ¶61. Siripurapu also teaches monitoring worker nodes to detect failures. EX1007, [0509].

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, ¶62.

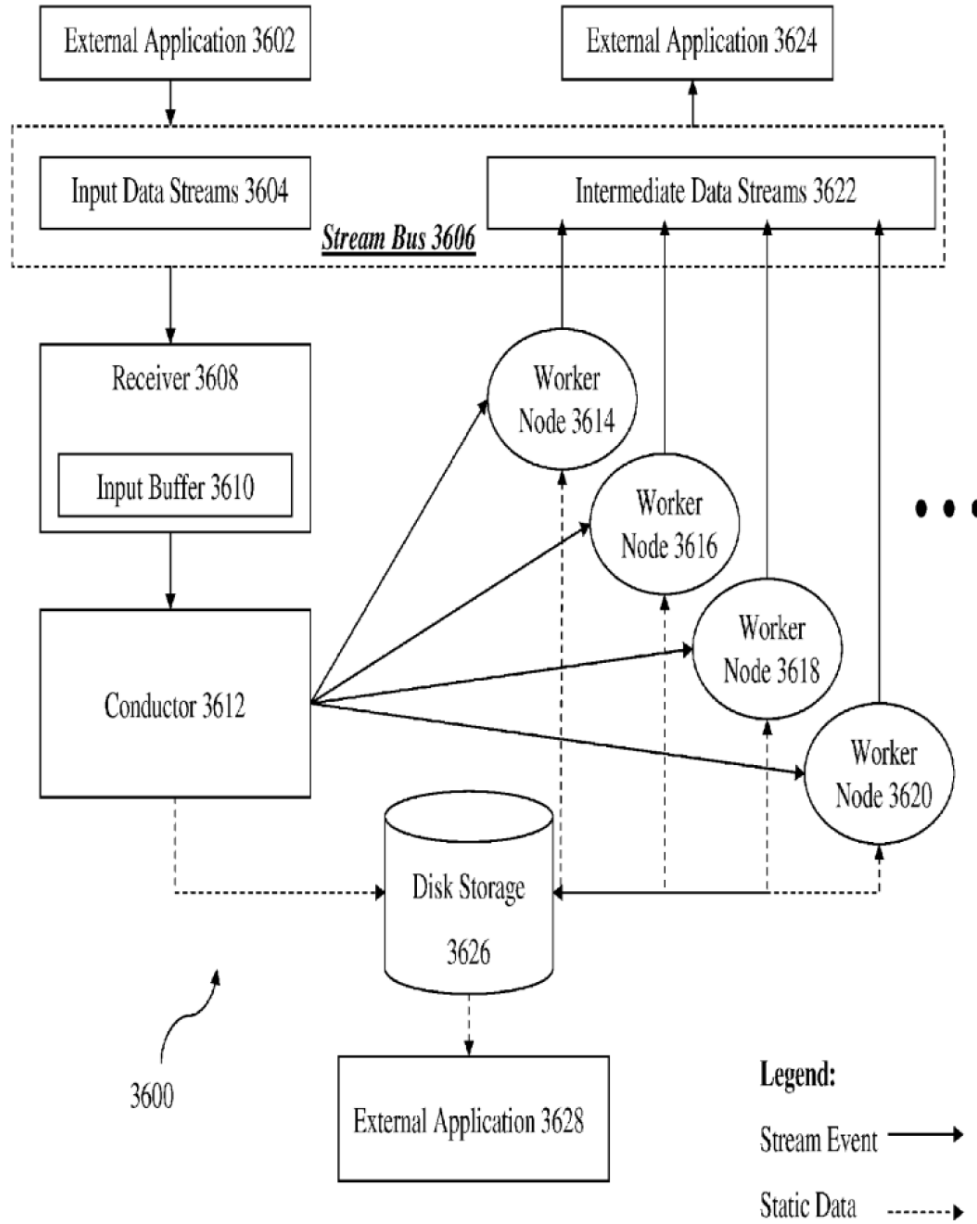


Figure 36

EX1007, FIG. 36; EX1003, ¶62.

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, ¶172.

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 processing elements across compute nodes and can add/remove processing elements for “data processing needs and scalability.” EX1004, 6:53-55; EX1003, ¶173.

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 based on monitoring the compute nodes (EX1004, 6:19-42, 10:26-39). Like the monitoring system 110 of Barsness, the conductor of Siripurapu “may have visibility into the performance of the worker nodes and may perform 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, ¶174. These teachings would motivate a POSITA to combine the references, as they would increase efficiency and processing rates. *Id.*

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 based monitoring those nodes. EX1003, ¶¶173-175. The two teachings are complementary, as Barsness describes adding additional compute nodes, and Siripurapu provides further details on adding additional computing resources. This further reinforces that a POSITA would have naturally combined them to create an infrastructure capable of monitoring the performance metrics of a streaming application and dynamically adapting the computing resources to meet the desired performance of the system. EX1003, ¶175.

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, ¶176.

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) combined with Siripurapu’s teachings on scaling and adding nodes to accommodate additional data streams based on monitoring the load of those nodes. EX1007, [0453], [0496]. This Barsness-Siripurapu combination renders all challenged claims obvious. EX1003, ¶177.

3. Claims 1-6

(a) 1[e] through 1[h]

As discussed *supra* Section VII.A, claims 1-6 are obvious in view of Barsness alone. Claims 1-6 are further obvious in view of the Barsness-Siripurapu combination, as a POSITA would have found it obvious to implement the monitoring system 110 of Barsness to cause the provisioning of additional compute node(s) to handle increased data volume / processing needs at a given compute node/transformation pipeline based on monitoring of the compute nodes, as taught by Barsness and Siripurapu. EX1007, [0453]; *id.*, [0448] (explaining “[t]he conductor 3612 may have visibility into the performance of the worker nodes and

may perform load balancing by reassigning tasks to the worker nodes.”); EX1003, ¶178.

Further, Siripurapu teaches that “[t]he conductor 3612 may maintain metadata about the available worker nodes in the system including ... the computational specifications of the worker nodes, the current status of each worker node (e.g., whether it is operational, is in a failed state, etc.), the current processing load at each worker node” (*first data*). EX1007, [0449]. Siripurapu teaches that the conductor may poll the worker nodes to determine each worker node’s “load data.” EX1007, [0470], [0496], [0509]. A POSITA would have found it obvious to implement the *processing* of the current status of worker nodes (*first data*) of Siripurapu in the monitoring system 110 of Barsness to then *monitor* the processing load at each compute node 130 if the node is operational. EX1003, ¶179.

(b) Claim 2

Siripurapu teaches that the conductor “may poll the worker nodes at pre-determined intervals of time for load data and to detect failures.” EX1007, [0509]; *see also id.*, [0449]. A POSITA would have found it obvious to implement the failure detection state of worker nodes from Siripurapu in the monitoring system 110 of Barsness to *identify execution that has become unresponsive*. EX1003, ¶180.

D. Ground 4: Claims 1-6 Are Obvious Over Barsness in View of Chakradhar and Siripurapu

As explained above, Barsness alone renders claims 1-6 obvious. *Supra* Section VII.A. For the same reasons a POSITA would have combined the real-time stream processing system of Barsness with the teachings of Chakradhar and Siripurapu individually (*supra* Sections VIII.B.2, VII.C.2), 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. The challenged claims are further obvious in view of a combination of Barsness, Chakradhar, and Siripurapu. EX1003, ¶181.

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, ¶182-186.

IX. CONCLUSION

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

Date: May 26, 2026

Respectfully submitted,

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

Counsel for Petitioners

**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 12,783 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: May 26, 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 Petitioners

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on May 26, 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 Priority Mail Express® or Express 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: May 26, 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 Petitioners