

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

PALO ALTO NETWORKS, INC.
Petitioners,

v.

QOMPLX LLC,
Patent Owner.

Case No. IPR2026-00432

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

**DECLARATION OF PAUL S. MIN, PH.D.
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 12,143,424**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION AND ENGAGEMENT	4
II. BACKGROUND AND QUALIFICATIONS	4
III. LEGAL STANDARDS	8
A. Claim Construction.....	9
B. Anticipation Under 35 U.S.C. § 102	9
C. Obviousness Under 35 U.S.C. § 103.....	10
IV. LEVEL OF SKILL IN THE ART, AND STATE OF THE ART	13
A. Person Of Ordinary Skill In The Art.....	13
B. State Of The Art	16
1. Distributed, Highly Scalable, Stream Processing Engines Were Commonly Used For Real-Time Data Processing	16
2. Stream Processing Engines Typically Used Directed Graphs To Organize And Manage Processing Data Streams.....	22
V. THE '424 PATENT	25
A. The Claims' Earliest Effective Filing Date	25
B. The '424 Patent's Specification	25
VI. CLAIM CONSTRUCTION	27
VII. GROUND 1: CLAIMS 4, 6, 8 AND 12 ARE OBVIOUS OVER BARSNESS	28
A. Barsness (EX1004).....	28
B. Claim 1	34
1. [1.1] - A distributed computing cluster comprising:	34
2. [1.2] - a first plurality of computer systems,.....	37
3. [1.2.a] - wherein each respective computer system of the first plurality of computer systems comprises a memory that stores a respective first data, ...	44
4. [1.2.b] - wherein the respective first data represents a respective portion of a distributed computational graph, ...	50

5.	[1.2.c] - and wherein the distributed computational graph describes a flow of output data of a first transformation pipeline to an input of a second transformation pipeline,.....	66
6.	[1.3] - wherein a first computer system of the first plurality of computer systems is configured to:	73
7.	[1.3.a] - receive a first stream of input data from a first input feed,.....	75
8.	[1.3.c] - process the respective first data stored in the memory of the first computer system to determine information about the second transformation pipeline, ...	88
9.	[1.3.d] - and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined information,.....	99
10.	[1.4] - wherein the second computer system is configured to:.....	101
11.	[1.4.a] - receive the first pipeline output messages, ...	103
12.	[1.4.b] - and process the first pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the first pipeline output messages to generate second pipeline output messages, ...	104
13.	[1.5] - wherein the first and second computer systems are distinct; and.....	108
14.	[1.6] - a second plurality of computer systems;.....	110
15.	Element [1.7] - wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause a fourth computer system of the second plurality of computer systems to execute software instructions that apply at least one of the first transformation pipeline and the second transformation pipeline.....	115
C.	Claim 2 - The distributed computing cluster of claim 1, wherein at least one of the first transformation pipeline and the second transformation pipeline is non-linear.	128

D.	Claim 3 - The distributed computing cluster of claim 2, wherein the third computer system of the first plurality of computer systems is configured to: identify a fifth computer system of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the first transformation pipeline to any stream of input data, and cause the fifth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data.	131
E.	Claim 4	145
1.	[4.1] - The distributed computing cluster of claim 3, wherein the third computer system is further configured to execute software instructions that:.....	145
2.	[4.2] - identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph, ...	148
3.	[4.2.a] - wherein the respective first data includes information about the first transformation pipeline,.....	152
4.	[4.3] - and store in the identified one or more memories information identifying the fifth computer system.	153
F.	Claim 5 - The distributed computing cluster of claim 2, wherein the third computer system of the first plurality of computer systems is configured to: identify a fifth computer system of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the second transformation pipeline to any pipeline output message, and cause the fifth computer system to process third pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the third pipeline output messages.....	156
G.	Claim 6	160
1.	[6.1] - The distributed computing cluster of claim 5, wherein the third computer system is further configured to execute software instructions that:.....	160

2.	[6.2] - identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph, ...	160
3.	[6.2.a] - wherein the respective first data includes information about the second transformation pipeline, ...	160
4.	[6.3] - and store in the identified one or more memories information identifying the fifth computer system.	161
H.	Claim 8 - The distributed computing cluster of claim 1, wherein the first computer system is further configured to transmit the first pipeline output messages to another computer system of the second plurality of computer systems.	161
I.	Claim 9	164
1.	[9.1] - A distributed computing cluster comprising:	164
2.	[9.2] - a first plurality of computer systems, ...	164
3.	Element [9.3] - wherein each respective computer system of the first plurality of computer systems comprises a respective memory that stores a respective first data that: describes at least a respective portion of a configuration of a first transformation pipeline, and identifies at least a second transformation pipeline, ...	166
4.	Element [9.4] - wherein a first computer system of the first plurality of computer systems is configured to: receive a first stream of input data from a first input feed, process the first stream of input data substantially in real time by executing software instructions that apply the first transformation pipeline to the first stream of input data to generate first pipeline output messages, process the respective first data to determine an identification of the second transformation pipeline, and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined identification, ...	170
5.	Element [9.5] - wherein the second computer system is configured to: receive the first pipeline output messages, and process the first pipeline output messages substantially in real time by executing software	

	instructions that apply the second transformation pipeline to the first pipeline output messages,.....	172
6.	Element [9.6] - wherein the first and second computer systems are distinct; and	173
7.	Element [9.7] - a second plurality of computer systems;	173
8.	Element [9.8] - wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause the second plurality of computer systems to execute at least one of the first transformation pipeline and the second transformation pipeline.....	174
J.	Claim 10 - The distributed computing cluster of claim 9, wherein one or more of the first transformation pipeline and the second transformation pipeline is non-linear.	174
K.	Claim 11 - The distributed computing cluster of claim 10, wherein the third computer system of the first plurality of computer systems is configured to: identify a fourth computer system of the first plurality of computer systems, wherein the fourth computer system is not configured to apply the first transformation pipeline to any stream of input data, and cause the fourth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data.	175
L.	Claim 12 - The distributed computing cluster of claim 11, wherein the third computer system is further configured to execute software instructions that: identify one or more of the respective memories that stores respective data that describes a respective portion of a configuration of the first transformation pipeline, and store in the identified one or more memories information identifying the fourth computer system.	176
VIII.	ROUND 2: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF CHAKRADHAR.....	176
A.	Chakradhar (EX1005)	177
B.	Chakradhar Is Supported By The Chakradhar Provisional.....	179
C.	The Chakradhar Provisional Supports Claim 1 Of Chakradhar.....	179

D.	Combining Barsness And Chakradhar	181
E.	Claims 4, 6, 8, and 12	186
IX.	GROUND 3: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF SIRIPURAPU	192
A.	Siripurapu (EX1007)	193
B.	Combining Barsness And Siripurapu	196
C.	Claims 4, 6, 8, and 12	202
X.	GROUND 4: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF CHAKRADHAR AND SIRIPURAPU	203
XI.	CONCLUSION	205

TABLE OF EXHIBITS LIST

Ex.	Description
1001	U.S. Patent No. 12,143,424 (“the ’424 patent”)
1002	Prosecution History for U.S. Patent No. 12,143,424
1003	Intentionally Omitted
1004	U.S. Patent No. 8,732,300 (“Barsness”)
1005	U.S. Patent Application Pub. No. 2016/0269247 A1 (“Chakradhar”)
1006	U.S. Provisional Application No. 62/132,686 (“Chakradhar Provisional”)
1007	U.S. Patent Application Pub. No. 2014/0059185 A1 (“Siripurapu”)
1008	Thomas Heinze, et al., Cloud-based Data Stream Processing, Proc. of the 8th ACM Int’l Conf. on Distributed Event-Based Sys., 238-245 (May 2014), https://di.acm.org/doi/10.1145/2611286.2611309 (“Heinze”)
1009	Ashkon Soroudi, et al., High Availability on a Distributed Real Time Processing System, Electrical Engineering and Computer Sciences,

	University of California at Berkeley (May 15, 2015), http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-135.html (“Soroudi”)
1010	Naresh Kumar Reddy Rapolu, Dynamic Re-Optimization Techniques for Stream Processing Engines and Object Stores (2015) (Ph.D. dissertation, Purdue University) (“Rapolu”)
1011	Ankit Toshniwal, et al., Storm @Twitter, SIGMOD ’14: Proc. of the 2014 ACM SIGMOD Int’l Conf. on Mgmt. of Data, 147-156 (2014), http://dx.doi.org/10.1145/2588555.2595641 (“Toshniwal”)
1012	Leonardo Neumeyer, et al., S4: Distributed Stream Computing Platform, IEEE Int’l Conf. on Data Mining Workshops, 170-177 (2010), https://ieeexplore.ieee.org/document/5693297 (“Neumeyer”)
1013	Michael Duller, et al., Virtualizing Stream Processing, Middleware (2011) (“Duller”)
1014	Scott Schneider, et al., Tutorial: Stream Processing Optimizations, DEBS ’13 (2013) (“Schneider”)

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

I, Paul S. Min, do hereby declare as follows:

I. INTRODUCTION AND ENGAGEMENT

1. I, Paul S. Min, Ph.D., submit this declaration to state my opinions on the matter described below.

2. I have been retained by Petitioner Palo Alto Networks, Inc. (“Petitioner”), as an independent expert in this proceeding before the United States Patent and Trademark Office. Although I am being compensated at my usual and customary rate of \$500 per hour, no part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I understand that this proceeding involves U.S. Patent No. 12,143,424 (the “’424 patent”), and I have been asked to consider the validity of certain claims of the ’424 patent based on certain prior art references. I have also been asked to consider the state of the art and prior art available as of October 28, 2015. Based on the prior art discussed in this declaration, it is my opinion that claims 4, 6, 8, and 12 of the ’424 patent are unpatentable for the reasons provided below.

II. BACKGROUND AND QUALIFICATIONS

4. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience, and specifically, in the field of distributed systems. My curriculum vitae (“CV”) is submitted as Exhibit 1018.

5. I received a B.S. degree in Electrical Engineering in 1982, an M.S. degree in Electrical Engineering in 1984, and a Ph.D. degree in Electrical Engineering in 1987 from the University of Michigan in Ann Arbor. I received several academic honors, including my B.S. degree with honors, a best graduate student award and a best teaching assistant award during my M.S. study, and a best paper award from a major international conference for reporting results from my Ph.D. thesis. After receiving my Ph.D., I worked at Bellcore in New Jersey from August 1987 until August 1990. At Bellcore, I was responsible for evolving the public switched telephone network (POTS) into a multi-services voice and data network that incorporated packet switches, optical technologies, and wireless technologies.

6. In September 1990, I joined the faculty at Washington University in St. Louis. In July 1996, I was promoted to an Associate Professor of Electrical Engineering with tenure. I am currently a Senior Professor at Washington University of the Electrical and Systems Engineering. I have also served as the Chair of the Graduate Curriculum (2000-2002) and the Chair of the Undergraduate Curriculum (2011-2014) for the Electrical and Systems Engineering department. At Washington University, I have conducted research in communication, computing, and related electronic hardware and software. My research group has pioneered a new paradigm for designing electronic circuits that can alleviate the speed and performance

mismatch against optical technology. I have received several grants from the U.S. Federal Agencies, including the National Science Foundation and the Defense Advanced Research Project Agency, and numerous contracts from companies and organizations around the world. Specifically related to the technology in this matter, I have researched various computing paradigms including pipeline architecture, distributed computing platform, peer-to-peer multi-processor architecture. In addition, I have developed various monitoring/inspection techniques for the extremely high-speed Internet traffic I have an extensive background and experience in each of these technologies.

7. As a faculty member at Washington University, I have taught a number of courses in electronics, communication, and computing at both the undergraduate and graduate levels. For example, I have taught communication theory (Washington University ESE 471), transmission and multiplexing (Washington University ESE 571), and signaling and control of communication networks (Washington University ESE 572). I have supervised numerous graduate students, many of whom received a doctoral degree under my guidance. A number of doctoral theses that I have supervised relate specifically to multi-processor distributed computing technology.

8. In addition to my responsibilities as a university faculty member, I have founded two companies. In May 1997, I founded MinMax Technologies, Inc., a fabless semiconductor company that developed switch fabric integrated circuit chips

for the Internet. In March 1999, I founded Erlang Technology, Inc., a fabless semiconductor company that focused on the design and development of integrated circuit chips and software for the Internet. One of Erlang's products received a best product of the year award in 2004 from a major trade journal for the electronics industry.

9. Outside my own start-up companies, I have also served in various technology and business advisor roles for other companies and organizations around the world. I was the main technical author for one of two winning proposals to the Korean government for CDMA wireless service licenses (1996). I was responsible for designing a commercial scale IS-95 CDMA cellular network, which I understand to be one of the earliest such networks deployed in the world. I worked with numerous engineers and scientists around the world to implement this commercial-scale cellular network before IS-95 CDMA was widely accepted. This provided me with extensive insight into various components of CDMA technology, which by and large are used in WCDMA network. I have also been involved in a semiconductor company that specializes in semiconductor memories such as flash EEPROMs as a board member and as a technical advisor (2007-2011).

10. I am a named inventor on eleven U.S. patents. I have extensively published technical papers in international conferences and journals, technical memoranda and reports, and given a number of seminars and invited talks. Many of

these papers are specifically within the context of the distributed computing technology. I have organized several international conferences and served as an international journal editor.

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

III. LEGAL STANDARDS

12. In forming my opinions and considering the subject matter of the '424 patent and its claims in light of the prior art, I am relying on certain legal principles that counsel in this case explained to me. My understanding of these concepts is summarized below.

13. I understand that the claims define the invention. I also understand that an unpatentability analysis is a two-step process. First, the claims of the patent are construed to determine their meaning and scope. Second, after the claims are construed, the content of the prior art is compared to the construed claims.

14. I understand that a claimed invention is only patentable when it is new, useful, and non-obvious in light of the "prior art." That is, the invention, as defined

by the claims of the patent, must not be anticipated or rendered obvious by the prior art.

A. **Claim Construction**

15. I understand that the United States Patent and Trademark Office interprets claim terms in an *inter-partes* review proceeding under the same claim construction standard that is used in a United States federal court. I understand that under this standard, the meaning of claim terms is considered from the viewpoint of one of ordinary skill in the art at the time of the alleged invention.

16. I understand that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand, however, that claim terms are generally not limited by the embodiments described in the specification.

17. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I have been informed that this other evidence is often less significant and less reliable than the claims, specification, and prosecution history.

B. **Anticipation Under 35 U.S.C. § 102**

18. I understand that under 35 U.S.C. § 102, a patent claim is invalid if its subject matter was patented or described in a printed publication before the effective filing date of the claimed invention. I have been told that this is referred to as invalidity by anticipation. I have been told that a patent claim is anticipated under § 102 if a single prior art reference discloses all limitations of the claimed invention.

C. **Obviousness Under 35 U.S.C. § 103**

19. I understand that a patent claim is invalid as obvious if the claimed invention would have been obvious to a person of ordinary skill in the art (“POSITA”) at the time the claimed invention was made. This means that even if all of the elements of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the field who knew about all the prior art would have come up with the claimed invention. I understand that in an obviousness determination, the person of ordinary skill in the art is presumed to have knowledge of all material prior art. I understand that whether a claim is obvious is based upon the determination of several factual issues. I understand that obviousness is a determination of law based on underlying determinations of fact. I understand that these factual determinations include the scope and content of the prior art, the level of ordinary skill in the art, the differences between the claimed invention and the prior art, and secondary considerations of non-obviousness.

20. In considering obviousness, I understand that one must determine the scope and content of the prior art. I understand that, in order to be considered as prior art to a patent being considered, a prior art reference must be reasonably related to the claimed invention of that patent. A reference is reasonably related if it is in the same field as the claimed invention or is from another field to which a person of ordinary skill in the art would look to solve a known problem.

21. I understand that one must determine what differences, if any, existed between the claimed invention and the prior art.

22. I understand that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was independently known in the prior art. In evaluating whether such a claim would have been obvious, one may consider whether a reason has been identified that would have prompted a person of ordinary skill in the art to combine the elements or concepts from the prior art in the same way as in the claimed invention. There is no single way to define the line between true inventiveness on the one hand (which is patentable) and the application of common sense and ordinary skill to solve a problem on the other hand (which is not patentable). For example, market forces or other design incentives may be what precipitated a change, rather than true inventiveness.

23. I understand that whether a prior art reference renders a patent claim unpatentable as obvious is determined from the perspective of a person of ordinary

skill in the art at the time of the alleged invention. I have been told that there is no requirement that the prior art contain an express suggestion to combine known elements to achieve the claimed invention, but a suggestion to combine known elements to achieve the claimed invention may come from the prior art, as filtered through the knowledge of one skilled in the art. In addition, I have been told that the inferences and creative steps a person of ordinary skill in the art would employ are also relevant to the determination of obviousness.

24. I understand that there is no rigid rule that a reference or combination of references must contain a “teaching, suggestion, or motivation” to combine references. But I also understand that the “teaching, suggestion, or motivation” test can be a useful guide in establishing a rationale for combining elements of the prior art. I have been told that this test poses the question as to whether there is an express or implied teaching, suggestion, or motivation to combine prior art elements in a way that realizes the claimed invention, and that it seeks to counter impermissible hindsight analysis.

25. I understand that one may consider, e.g., whether (1) the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness; (2) there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; (3) the claimed innovation applies a known technique

that had been used to improve a similar device or method in a similar way; (4) the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art; (5) the invention merely substituted one known element for another known element in order to obtain predictable results; (6) the invention merely applies a known technique to a known device, method, or product to yield predictable results; or (7) known work in the field may have prompted variations of use of the same inventions in the same or different fields due to market forces or design incentives that would have been predictable to a person of ordinary skill in the art. I understand that any assertion of secondary considerations of non-obviousness must be accompanied by a nexus between the merits of the invention and the evidence offered.

IV. LEVEL OF SKILL IN THE ART, AND STATE OF THE ART

A. Person Of Ordinary Skill In The Art

26. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶26) states:

I understand that in evaluating obviousness of the '424 patent claims, the content of a patent or printed publication prior art should be interpreted the way a person of ordinary skill in the art would have interpreted the prior art as of the effective filing date of the challenged patent claims. I assume that the effective filing date is the earliest filing date of an alleged related application, namely October 28, 2015. My understanding of a person of ordinary skill in the art would not materially differ if the effective filing date were the date the application

resulting in the '424 patent was filed, namely July 21, 2024, at least because the knowledge and level of skill of a person of ordinary skill in the art in 2024 would encompass the knowledge and skill of such person in 2015.

I agree with Dr. Crovella because I likewise understand that, when obviousness is assessed, a prior-art patent or printed publication is read as a person of ordinary skill in the art would have read it on the effective filing date of the challenged claims. I adopt October 28, 2015, the earliest filing date of an alleged related application, as that effective filing date. Even if the effective filing date were instead July 21, 2024, the date on which the application that issued as the '424 patent was filed, my understanding of how a person of ordinary skill would read the prior art would be the same, because a person of ordinary skill in 2024 already possessed the full measure of knowledge and skill that such a person had in 2015.

27. I understand that factors that may be considered in determining the level of ordinary skill in the art at the time of the effective filing date of the challenged patent include: (1) the educational level of the inventor; (2) the types of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field.

28. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶28) states:

A person of ordinary skill in the art (“POSITA”) in July 2024 or October 2015 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, e.g., a Master’s or Ph.D., would compensate for less work experience and vice versa.

I agree with Dr. Crovella because it is my opinion that a person of ordinary skill in the art (a “POSITA”), whether in July 2024 or October 2015, would have held a bachelor’s degree in computer science, computer engineering, electrical engineering, or a comparable field, together with two to four years of experience designing, building, or otherwise working with distributed computing systems. In my opinion, additional education, such as a Master’s or a Ph.D., would offset a shorter period of work experience, just as greater work experience would offset less formal education.

29. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶29) states:

A POSITA would have been familiar with common, then-existing technologies for distributed computing and cloud-based stream processing systems, as well as issues surrounding the design, implementation, and maintenance of such technologies. The state of the art below reflects certain technologies within the knowledge and skill of a POSITA.

I agree with Dr. Crovella because it is my opinion that a POSITA would have been familiar with the common technologies then available for distributed computing and cloud-based stream processing systems, as well as with the challenges of designing,

implementing, and maintaining those technologies. The state-of-the-art discussion that follows sets out particular technologies that fell within the knowledge and skill of a POSITA.

B. State Of The Art

**1. Distributed, Highly Scalable, Stream Processing Engines
Were Commonly Used For Real-Time Data Processing**

30. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶30)

states:

By 2015, distributed stream processing engines were already well-established tools for real-time analysis of continuous data streams. Contemporary literature such as Heinze explained that cloud-based data stream processing systems were designed for high-throughput, low-latency processing of unbounded streams, and that newer "third generation" systems were specifically built to be "highly scalable and robust towards faults." EX1008, 238. Heinze explained that cloud-based streaming engines were "designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads." EX1008, 240. These systems provided parallel processing of data streams and distributed multiple data streams / queries across a cluster. EX1008, 240. These "elastic data stream processing systems" could "dynamically vary the number of processing nodes within the system based on the workload." EX1008, 241; *id.*, 238, 240 ("Elasticity describes the behavior of a system to dynamically scale in or out based on a changing workload.").

I agree with Dr. Crovella because, in my opinion, distributed stream processing engines were, by 2015, well-established tools for analyzing continuous data streams in real time. As the contemporaneous literature confirms, Heinze described cloud-based data stream processing systems as built for high-throughput, low-latency

processing of unbounded streams, and explained that a newer “third generation” of such systems was designed to be “highly scalable and robust towards faults.” EX1008, 238. Heinze further described cloud-based streaming engines as “designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads.” EX1008, 240. Such systems processed data streams in parallel and distributed multiple data streams and queries across a cluster. EX1008, 240. These “elastic data stream processing systems” could “dynamically vary the number of processing nodes within the system based on the workload.” EX1008, 241; *id.*, 238, 240 (“Elasticity describes the behavior of a system to dynamically scale in or out based on a changing workload.”). In my opinion, these capabilities, taken together, show that real-time stream processing had matured well before the relevant timeframe. The ability to scale across many nodes, to distribute streams and queries in parallel, to adjust capacity elastically as workloads changed, and to continue operating despite faults reflects mature, well-established technology rather than a nascent or purely experimental capability. A POSITA would therefore have understood these distributed, cloud-based stream processing systems, along with the design considerations Heinze identifies, to be part of the ordinary toolkit available for building and operating real-time data processing systems as of the effective filing date.

31. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶31)

states:

Other references confirm that these characteristics were standard features of practical stream processing systems. As early as 2010, there was already “a clear need for highly scalable stream computing solutions that can operate at high data rates and process massive amounts of data.” EX1012, 170. Neumeyer described S4 in 2010 as a “distributed, scalable” platform that “allow[ed] programmers to easily develop applications for processing continuous unbounded streams of data,” and explained that S4's design was intended for “low latency” operation in “large clusters built with commodity hardware.” EX1012, 170. Neumeyer further explained that production environments required “scalability (ability to add more servers to increase throughput with minimal effort) and high availability (ability to achieve continuous operation with no human intervention in the presence of system failures).” EX1012, 170. Stream processing engines such as Storm were designed to be scalable, resilient, extensible, efficient, and easy to administer. EX1011, 147. Toshniwal described Storm as a “real-time fault-tolerant and distributed stream data processing system” used at Twitter “at scale, and in real-time,” and explained that Storm was specifically designed so operators could “add or remove nodes from the Storm cluster without disrupting existing data flows,” while continuing to operate on “large clusters.” EX1011, 147.

I agree with Dr. Crovella because, in my opinion, these same characteristics recur across other contemporaneous references, which confirms that they were standard features of practical stream processing systems rather than isolated design choices. By 2010, the literature already recognized a broadly felt industry need, namely “a clear need for highly scalable stream computing solutions that can operate at high data rates and process massive amounts of data.” EX1012, 170. Neumeyer described the S4 platform that year as “distributed, scalable” and as one that “allow[ed]

programmers to easily develop applications for processing continuous unbounded streams of data,” with a design aimed at “low latency” operation in “large clusters built with commodity hardware.” EX1012, 170. In my opinion, S4 is one representative example of an approach that was already common in the field. Neumeyer tied those goals to the demands of production, explaining that such environments required “scalability (ability to add more servers to increase throughput with minimal effort) and high availability (ability to achieve continuous operation with no human intervention in the presence of system failures).” EX1012, 170. It is my opinion that scalability and high availability of this kind were understood as baseline expectations for production stream processing systems, not aspirational targets. The same priorities are reflected in Apache Storm, which was designed to be scalable, resilient, extensible, efficient, and easy to administer. EX1011, 147. Toshniwal described Storm as a “real-time fault-tolerant and distributed stream data processing system” that Twitter ran “at scale, and in real-time,” and explained that Storm let operators “add or remove nodes from the Storm cluster without disrupting existing data flows” while continuing to run on “large clusters.” EX1011, 147. In my opinion, the consistency of these descriptions across independently developed systems is significant. S4 and Apache Storm are two representative examples, among others available at the time, of platforms built around the same objectives of horizontal scalability, low-latency processing of

unbounded streams, and continuous fault-tolerant operation on commodity clusters.

That convergence indicates that these features were not the aspirations of any single project but the accepted design norms for practical stream processing systems generally, and a POSITA would have recognized scalability, high availability, and fault tolerance as expected, well-understood properties of such systems as of the effective filing date.

32. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶32)

states:

Other literature reflecting the knowledge of a POSITA recognized scalability through parallelization and scale-out as a core requirement of streaming systems. Schneider explained that handling "large volumes of live data in short periods of time" made "high throughput processing" a "critical requirement," which in turn required taking advantage of "multiple cores and/or host machines to achieve scale." EX1014, 249; *see also* EX1011, 1 ("Many modern data processing environments require processing complex computation on streaming data in real-time."); EX1009, 4 ("The term -real-time processing in this context means that the system is able to respond within seconds or sub-second to requests, while 'distributed' means that it is running on multiple computers."). Duller similarly explained that a stream-processing application could be "scale[d] out" across nodes and that the tested application "linearly scale[d] up the load ... with the number of nodes." EX1013, 283.

I agree with Dr. Crovella because, in my opinion, still other literature within the knowledge of a POSITA treated scalability through parallelization and scale-out as a core requirement of streaming systems, not an optional enhancement. Schneider explained that handling "large volumes of live data in short periods of time" made

“high throughput processing” a “critical requirement,” which in turn required taking advantage of “multiple cores and/or host machines to achieve scale.” EX1014, 249; *see also* EX1011, 1 (“Many modern data processing environments require processing complex computation on streaming data in real-time.”); EX1009, 4 (“The term -real-time processing in this context means that the system is able to respond within seconds or sub-second to requests, while ‘distributed’ means that it is running on multiple computers.”). The recognized answer to rising data volumes was to spread the work across cores and machines, which is the essence of horizontal scale-out. Duller made the same point in concrete terms, explaining that a stream-processing application could be “scale[d] out” across nodes and that the tested application “linearly scale[d] up the load ... with the number of nodes.” EX1013, 283. In my opinion, Duller’s report of near-linear scaling is particularly telling, because it shows that adding nodes translated directly into added capacity, which is the behavior a POSITA would have expected from a well-designed distributed streaming system. Taken together, these sources reflect a settled understanding that streaming systems met throughput demands by parallelizing work and scaling out across a cluster, and a POSITA would have regarded parallelization and scale-out as ordinary, expected means of achieving scale as of the effective filing date.

33. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶33) states:

By 2015, various “distributed stream processing engines,” such as Aurora, Borealis, TimeStream, and open-source systems such as S4, Samza, and Storm were available “[t]o process continuous streams of data at low latency.” EX1010, 2, 12; EX1011, 147.

I agree with Dr. Crovella because, in my opinion, by 2015 a variety of “distributed stream processing engines” was available and in practical use. Systems such as Aurora, Borealis, and TimeStream, together with open-source systems such as S4, Samza, and Storm, were available “[t]o process continuous streams of data at low latency.” EX1010, 2, 12; EX1011, 147. It is my opinion that the number and variety of these engines confirms that low-latency distributed stream processing was a mature and widely adopted capability rather than a specialized or emerging one, and a POSITA would have been aware of these engines and the common approach they embodied as of the effective filing date.

**2. Stream Processing Engines Typically Used Directed Graphs
To Organize And Manage Processing Data Streams**

34. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶34)

states:

It was well known by 2015 to use directed graphs to represent and manage the flow of data through streaming applications: “Stream engines code the application workflow as a directed acyclic graph (DAG) of operators, referred to as a topology.” EX1010, 2; *see also* EX1013, 271-273 (describing a “data processing model” comprising a “graph” of operators). Further, “[t]uples are processed in a pipelined fashion as they traverse through the topology.” EX1010, 2. “A topology is a directed graph where the vertices represent computation and the edges represent the data flow between the computation components.”

EX1011, 148 (“The basic Storm data processing architecture consists of *streams of tuples* flowing through *topologies*.”).

I agree with Dr. Crovella because, in my opinion, the use of directed graphs to represent and manage the flow of data through streaming applications was well known by 2015. As the literature put it, “Stream engines code the application workflow as a directed acyclic graph (DAG) of operators, referred to as a topology.” EX1010, 2; *see also* EX1013, 271-273 (describing a “data processing model” comprising a “graph” of operators). Within that model, “[t]uples are processed in a pipelined fashion as they traverse through the topology.” EX1010, 2. And, as applied in a concrete system, “A topology is a directed graph where the vertices represent computation and the edges represent the data flow between the computation components.” EX1011, 148 (“The basic Storm data processing architecture consists of *streams of tuples* flowing through *topologies*.”). It is my opinion that this graph-based topology model was the conventional way of organizing streaming computation, with operators at the vertices, data flowing along the edges, and tuples moving through the graph in pipelined fashion. A POSITA would have understood a streaming application to be structured as such a directed graph of operators as of the effective filing date.

35. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶35) states:

Schneider explained that a “stream processing application is organized as a *graph*, formed by a set of *operators* connected to each other by *streams*,” where a “stream is a series of *data items*.” EX1014, 250.

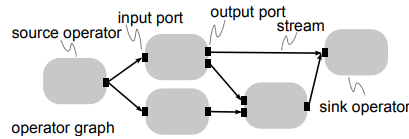


Figure 1: Basic concepts related to streaming applications.

EX1014, 250. Schneider explains how the operators of the graph are connected via input ports and output ports. *Id.* Further, source operators perform “edge adaptation to receive data from an external source,” while sink operators perform “edge adaptation to deliver data” to an external sink. *Id.* Andrade, an IBM patent, similarly explains:

A data-flow graph can include a set of processing elements (PEs) connected by streams, wherein each stream carries a series of stream data objects SDOs.) The PEs implement data stream analytics and are basic execution units that are distributed over the compute nodes. The PEs also communicate with each other via input and output ports, connected by streams.

EX1015, 5:31-41.

I agree with Dr. Crovella because, in my opinion, this graph-based structure was the standard way of describing streaming applications, and the references bear that out. Schneider explained that a “stream processing application is organized as a *graph*, formed by a set of *operators* connected to each other by *streams*,” where a “stream is a series of *data items*.” EX1014, 250. Schneider further explained that the operators of the graph are connected through input ports and output ports, that source operators perform “edge adaptation to receive data from an external source,” and that sink operators perform “edge adaptation to deliver data” to an external sink. *Id.*

Andrade, an IBM patent, described the same architecture, explaining that a data-flow graph comprises a set of processing elements connected by streams, where each stream carries a series of data objects, and that those processing elements are distributed over the compute nodes and communicate with one another through input and output ports. EX1015, 5:31-41. It is my opinion that these sources describe one and the same model: operators or processing elements at the vertices, streams along the edges carrying data items between them, and ports as the defined connection points. A POSITA would have recognized this operator-and-stream graph as the ordinary and expected way to structure a distributed streaming application as of the effective filing date.

V. THE '424 PATENT

A. The Claims' Earliest Effective Filing Date

36. For this declaration, I have been asked to assume that the claims' earliest possible effective filing date is October 28, 2015. That said, my opinions herein would not be materially different if the effective filing date was July 21, 2024.

B. The '424 Patent's Specification

37. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶37) states:

The '424 patent describes the "analysis of very large data sets using distributed computational graph tools which allow for transformation of data through both linear and non-linear transformation pipelines." EX1001, 1:32-35. Among other things, it describes determining when

aspects of the streaming data analysis “are processing slowly or may have hung ... so that actions can be implemented to resolve the issue.” EX1001, 16:41-56. The ’424 patent does not, however, further describe these issue-resolving “actions” or how to implement them.

I agree with Dr. Crovella because, in my opinion, the ’424 patent describes its subject matter at a high level while leaving key implementation details unexplained. The specification states that the invention concerns 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. In the context of streaming analysis, the patent describes a system sanity and retrain module that monitors the pipeline and detects when data searches or transformations “are processing slowly or may have hung ... so that actions can be implemented to resolve the issue.” EX1001, 16:41-56. It is my opinion, however, that the ’424 patent stops at detection. It identifies that corrective “actions” are to be taken once a slow or hung condition is found, but it does not describe what those “actions” are, the mechanism for carrying them out, or any example of their implementation. In my opinion, a POSITA would not have understood, from the specification, how to implement the issue-resolving actions the patent refers to.

38. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶38) states:

The ’424 patent provides examples of prior art transformation pipelines (EX1001, FIGS. 6, 12) and other transformation pipelines (EX1001, FIGS. 7-9, 13-15). EX1001, 17:11-19:17, 21:25-22:51 (describing

transformation pipelines). However, the '424 patent provides no disclosure of how to implement these transformation pipelines using the general hardware architecture shown in FIG. 3, with “any number of clients and/or servers.” EX1001, 13:50-62.

I agree with Dr. Crovella because, in my opinion, the '424 patent illustrates its transformation pipelines and its hardware architecture separately, without explaining how one is implemented on the other. The specification presents prior art transformation pipelines (EX1001, FIGS. 6, 12) and other transformation pipelines (EX1001, FIGS. 7-9, 13-15), described in the accompanying text as arrangements of transformation nodes that pass output messages from one node to the next. EX1001, 17:11-19:17, 21:25-22:51 (describing transformation pipelines). Separately, the patent depicts its computing environment in FIG. 3 as a general distributed network having “any number of clients and/or servers.” EX1001, 13:50-62. It is my opinion that these disclosures remain disconnected. The '424 patent shows what the pipelines look like as functional or process diagrams and shows a generic client-server architecture, but it does not describe how to implement the transformation pipelines on that general hardware, such as how the transformation nodes are assigned to, distributed across, or coordinated among the clients and servers of FIG. 3. In my opinion, a POSITA would not have understood, from the specification, how to build these transformation pipelines on the general architecture the patent depicts.

VI. CLAIM CONSTRUCTION

39. For this declaration, I apply the plain and ordinary meaning of all claim terms, in addition to certain proposed constructions as noted below. I reserve the right to offer my opinions on any constructions or construction issues presented by Patent Owner.

40. I have been asked to apply the following proposed constructions:

“distributed computational graph” (claims 1, 4): “a directed graph with each transformation comprising a node and the output messages between transformations comprising edges.” EX1001, 9:31-35; *see also id.*, 9:4 (“Definitions”), 9:35-50, 17:52-53; FIGS. 6-9 & 12-15.

“transformation” (claims 1-6, 9-12): “a function performed on zero or more streams of input data which results in a single stream of output.” EX1001, 9:51-54.

“transformation pipeline” / “pipeline” (throughout the ’424 claims): “a progression of functions which each perform a transformation on a data stream.” EX1001, 4:31-34, 9:62-67.

VII. GROUND 1: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS

41. As explained below, claims 4, 6, 8, and 12 are obvious over Barsness.

A. Barsness (EX1004)

42. Barsness is a U.S. patent that issued May 20, 2014, from an application filed January 10, 2011. EX1004, Cover.

43. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶43)

states:

Barsness discloses a distributed stream-processing computing cluster comprising a plurality of compute nodes executing processing elements ("PEs") that apply transformations to streaming data and transmit output data to downstream PEs according to an operator graph. Barsness explains that "[i]n a stream data processing application, operators are connected to one another such that data flows from one processing element (PE) to the next" and that "[s]calability is achieved by distributing an application across nodes." EX1004, 3:16-22.

I agree with Dr. Crovella because, in my opinion, Barsness discloses a distributed stream-processing computing cluster in which a plurality of compute nodes execute processing elements ("PEs") that apply transformations to streaming data and pass their output to downstream PEs arranged as an operator graph. As Barsness explains, "[i]n a stream data processing application, operators are connected to one another such that data flows from one processing element (PE) to the next," and "[s]calability is achieved by distributing an application across nodes." EX1004, 3:16-22. It is my opinion that Barsness thus describes the same operator-graph structure discussed above, with the PEs serving as the nodes that perform the transformations and the connections between them carrying the streaming data from one PE to the next. In my opinion, Barsness also ties that structure directly to scalability, explaining that distributing the application across nodes is what allows the system to scale. A POSITA would have understood Barsness to disclose a distributed, operator-graph-based stream processing system of the kind that was well understood in the art.

44. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶44)

states:

Barsness further describes a computing infrastructure including a management system and multiple compute nodes connected via a network, where “the operator graph ... represents a stream application beginning from one or more source processing elements (PEs) through to one or more sink PEs.” EX1004, 6:43-56. Barsness depicts these data streams generally flowing left to right, e.g., FIG. 1B shows data flowing through PEs (1-10) on respective compute nodes (130₁₋₄).

(Figure omitted. EX1004, FIG. 1B (red annotations added))

I agree with Dr. Crovella because, in my opinion, Barsness describes a computing infrastructure made up of a management system, monitoring system, and multiple compute nodes connected over a network, in which the operator graph organizes the flow of data through the application. As Barsness explains, “the operator graph ... represents a stream application beginning from one or more source processing elements (PEs) through to one or more sink PEs.” EX1004, 6:43-56. It is my opinion that this confirms the directed, source-to-sink structure of the Barsness system, in which data enters at one or more source PEs and moves through intermediate PEs to one or more sink PEs. Barsness illustrates this arrangement in FIG. 1B, which depicts data flowing generally from left to right through processing elements PE1 through PE10 distributed across compute nodes 130₁ through 130₄. *Id.*, FIG. 1B. A POSITA would have understood Barsness to arrange its processing elements as a

directed operator graph spanning multiple compute nodes, with data flowing from source to sink.

45. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶45) states:

As illustrated in FIG. 1B of Barsness, upstream processing elements collectively form a first transformation pipeline (e.g., 130₁) whose output is provided as input to downstream processing elements forming a second transformation pipeline (e.g., 130₂, 130₃, 130₄). Barsness describes that "data elements flow into a source PE of a stream application and are processed by that PE," and that output data elements subsequently flow to downstream PEs. EX1004, 6:49-56; 7:52-67.

I agree with Dr. Crovella because, in my opinion, FIG. 1B of Barsness shows the processing elements grouped into successive transformation pipelines. The upstream processing elements collectively form a first transformation pipeline (e.g., 130₁), and the output of that pipeline is provided as input to the downstream processing elements that form a second transformation pipeline (e.g., 130₂, 130₃, 130₄). Consistent with that arrangement, Barsness describes that "data elements flow into a source PE of a stream application and are processed by that PE," and that the output data elements then flow to the downstream PEs. EX1004, 6:49-56; 7:52-67. It is my opinion that Barsness therefore discloses not a single monolithic pipeline but a progression of connected transformation pipelines, in which the output of one pipeline serves as the input to the next. A POSITA would have understood the processing elements in Barsness to be organized into successive transformation

pipelines distributed across compute nodes, with data passing from the source pipeline to the downstream pipelines.

46. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶46) states:

Barsness teaches that routing of pipeline output messages is governed by topology information and performed at the operator level. Each compute node is a computer system comprising a processor, network interface, and memory. EX1004, 8:30-35. The memory stores processing elements and operators configured to process incoming data and to "emit data to other operators ... and to other PEs in the stream application as described by the operator graph." EX1004, 8:66-9:20; 9:4-9. Thus, Barsness requires that operators locally determine where to send output messages based on topology-related information.

I agree with Dr. Crovella because, in my opinion, Barsness routes each pipeline's output messages using topology information, and it does so at the level of the individual operators. Barsness describes each compute node as a computer system that includes a processor, a network interface, and memory. EX1004, 8:30-35. That memory holds the processing elements and operators, which Barsness configures to process incoming data and to "emit data to other operators ... and to other PEs in the stream application as described by the operator graph." EX1004, 8:66-9:20; 9:4-9. It is my opinion that this operator-graph-directed emission is what governs routing in Barsness, because each operator emits its output to the specific downstream operators and PEs identified by the operator graph, so the routing decision is made locally, at the operator, on the basis of the topology the graph describes. A POSITA

would have understood Barsness to route pipeline output messages at the operator level according to the topology defined by the operator graph.

47. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶47) states:

Barsness further emphasizes the importance of dynamic reconfiguration in stream-processing systems. For example, Barsness teaches that a monitoring application may "increase the priority of the PE in question or even move the PE to separate machines to gain more resource," and that processing elements may be moved or replicated to improve performance and fault tolerance. EX1004, 3:56-60. Barsness also explains that the operator graph and compute nodes may be reconfigured by a user interface and management system. EX1004, 7:3-6; 10:28-39.

I agree with Dr. Crovella because, in my opinion, Barsness treats dynamic reconfiguration as an important feature of stream-processing systems. As one example, Barsness teaches that a monitoring application may "increase the priority of the PE in question or even move the PE to separate machines to gain more resource," and, more generally, that processing elements may be moved or replicated to improve performance and fault tolerance. EX1004, 3:56-60. Barsness also discloses computer systems for managing and monitoring the compute nodes of the streaming application. Barsness discloses that "management system 105" and "monitoring system 110" each store and maintain the operator graph in memory. EX1004, 3:45-50, 10:12-14, 10:49-51; 11:26-27, 11:53-55. Specifically, Barsness explains that the operator graph and the compute nodes themselves may be

reconfigured through a user interface, the monitoring system, and a management system. EX1004, 7:3-6; 10:28-39. For example, “monitoring system 110” uses the operator graph to monitor the performance of the compute nodes by collecting “tuple processing data” and calculating “current tuple processing rates” to compare with “historical tuple processing rates.” EX1004, 13:4-7. If the current processing rates fail to meet the historical rates, the monitoring system can issue “corrective actions” including instructing the management system 105 to “spawn[] new compute nodes, [and] duplicat[e] a compute node.” EX1004, 14:51-56; *see also* EX1004, 10:14-39, 11:27-37. It is my opinion that these disclosures show that the Barsness system is not static, because it is designed so that the placement of processing elements, and the structure of the operator graph itself, can be changed at runtime in response to load, resource needs, or failures. A POSITA would have understood Barsness to support dynamic reconfiguration of its processing elements and operator graph to improve performance and fault tolerance.

B. **Claim 1**

1. **[1.1] - A distributed computing cluster comprising:**

48. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶48)

states:

Barsness satisfies the preamble. Barsness discloses a “computing infrastructure 100” that includes “a management system 105 and a plurality of compute nodes 130₁₋₄, each connected to a communications network 120.” EX1004, 6:43-48. This computing infrastructure is a

distributed computing cluster because these separate computing elements communicate over the network to perform coordinated processing. FIG. 1A shows an example computing infrastructure and its distributed computing elements:

(Figure omitted. EX1004, FIG. 1A)

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

I agree with Dr. Crovella because, in my opinion, to the extent the preamble is limiting Barsness discloses the *distributed computing cluster* recited in the preamble. Barsness discloses “a stream data processing application” with “operators [] connected to one another such that data flows from one processing element (PE) to the next” and “distributing [the] application across nodes by creating many small executable pieces of code (operators), as well as replicating processing elements on multiple nodes and load balancing among them.” EX1004, 3:16-22; *see also* EX1004, 6:49-56, 7:52-8:2. Barsness describes a “computing infrastructure 100” that includes “a management system 105 and a plurality of compute nodes 130₁₋₄, each connected to a communications network 120.” EX1004, 6:43-48. Additionally, “[a] monitoring system 110 may also be included in the computing infrastructure 100.” EX1004, 7:15-18. It is my opinion that this infrastructure is a *distributed computing cluster*, because its separate computing elements communicate over

network 120 to perform coordinated processing, as FIG. 1A illustrates. Barsness confirms this by describing each compute node as “a distributed computing node,” EX1004, 2:35-37, and by explaining that the monitoring system 110 “may be a cluster of a plurality of computing resources, which may be distributed in the network 120,” EX1004, 7:19-21. In my opinion, the Barsness computing infrastructure, with its plurality of compute nodes and its management and monitoring systems distributed throughout network 120, is therefore a ***distributed computing cluster***. A POSITA would have understood Barsness to disclose a ***distributed computing cluster*** in which separate compute nodes are coordinated and communicating over a network.

49. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶49) states:

A cluster was a well-known concept that referred to a group of machines working together. *E.g.*, EX1005, [0056] (executing Storm on a “cluster”); EX1007, [0349] (describing “hardware components including clusters of commodity nodes connected through commodity networking means,” and the “MapReduce software framework supports distributed computing on large data sets on clusters of computers”), [0438]; EX1008, 241 (“multiple processing nodes within a cluster, which allows a distributed execution”); EX1009, 34 (discussing running “Storm on a cluster of 8 virtual machines, each having 4 cores, 8GB of memory, and running CentOS 6”); EX1001, 10:32-35 (contrasting systems operating on “single machine” with systems “operating on a cluster of machines as is known in the art”). As explained in the following sections, the Barsness computing infrastructure comprises each of the recited features.

I agree with Dr. Crovella because, in my opinion, a cluster was a well-understood concept that referred to a group of machines working together and communicating over a network to perform coordinated operations such as data processing, as the contemporaneous references confirm. *E.g.*, EX1005, [0056] (executing Storm on a “cluster”); EX1007, [0349] (describing “hardware components including clusters of commodity nodes connected through commodity networking means,” and the “MapReduce software framework supports distributed computing on large data sets on clusters of computers”), [0438]; EX1008, 241 (“multiple processing nodes within a cluster, which allows a distributed execution”); EX1009, 34 (discussing running “Storm on a cluster of 8 virtual machines, each having 4 cores, 8GB of memory, and running CentOS 6”); EX1001, 10:32-35 (contrasting systems operating on “single machine” with systems “operating on a cluster of machines as is known in the art”). It is my opinion that these sources, which span academic and commercial systems as well as the ’424 patent itself, show that a POSITA would have recognized a cluster as an ordinary term for a group of networked machines operating together. As further explained in the sections that follow, the Barsness computing infrastructure includes each of the recited features.

2. **[1.2] - a first plurality of computer systems,**

50. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶50) states:

Barsness satisfies this element. Barsness discloses that its computing infrastructure includes *a first plurality of computer systems*, e.g., management system 105 and “a plurality of compute nodes 130₁₋₄.” EX1004, 6:45-49. A first plurality of computer systems is shown in FIG. 1A of Barsness:

(Figure omitted. EX1004, FIG. 1A (highlighting added))

I agree with Dr. Crovella because, in my opinion, Barsness discloses a *first plurality of computer systems*. Its computing infrastructure *a first plurality of computer systems*, e.g., management system 105, monitoring system 110, and at least compute node 130₁₋₂. EX1004, 6:45-49. and FIG. 1A depicts that *first plurality of computer systems*:

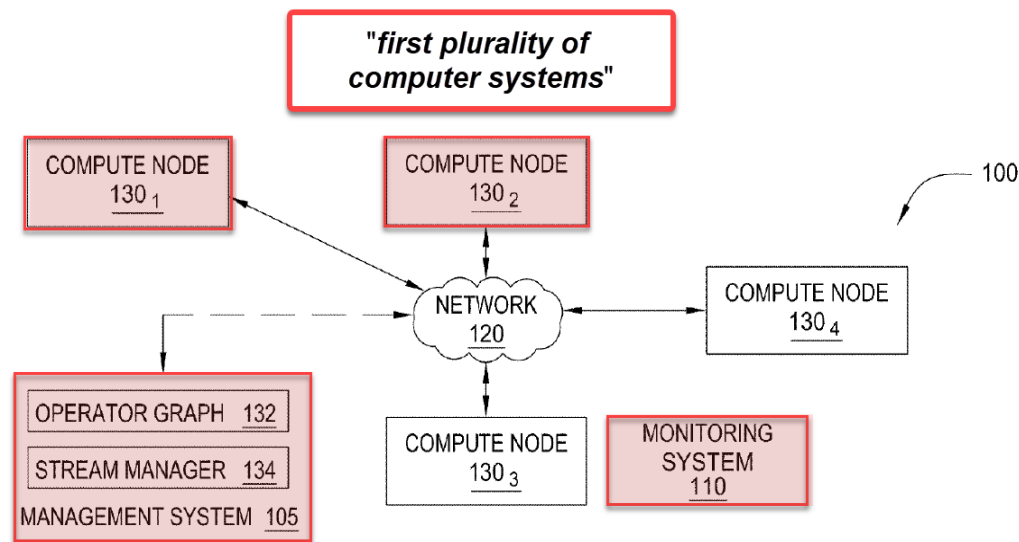


FIG. 1A

EX1004, FIG. 1A (annotations added). It is my opinion that these separate machines, the management and monitoring systems together with the several compute nodes, make up a *first plurality of computer systems* within the meaning of the claim. A POSITA would have understood Barsness to disclose a *first plurality of computer*

systems, namely its management system, monitoring system, and its several compute nodes.

51. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶51) states:

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; *see also* EX1001, 14:63-15:23 ('424 patent describing exemplary “computer system” including basic elements such as “CPU” and “memory”). Barsness FIG. 2 shows details of a compute node:

(Figure omitted. EX1004, FIG. 2)

Id., 8:53-65 (further describing the components of the compute node). Barsness discloses that “[t]he compute node 130 may be a physical host or a virtual machine,” where “[a] virtual machine 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).

I agree with Dr. Crovella because, in my opinion, each compute node in Barsness is a *computer system*. Barsness describes each compute node as including “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; *see also* EX1001, 14:63-15:23 (the '424 patent describing an exemplary “computer system” with the same basic elements, such as a “CPU” and “memory”). FIG. 2 of Barsness shows these components.

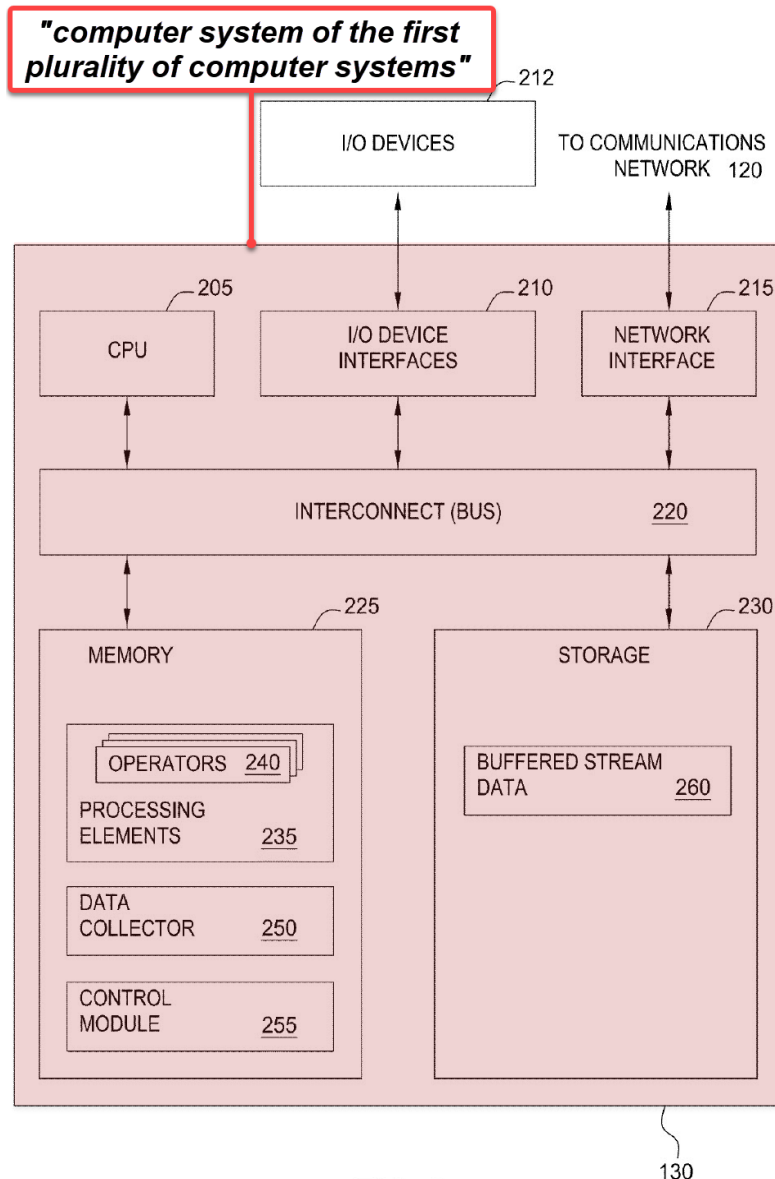


FIG. 2

EX1004, FIG. 2; *id.*, 8:53-65. Barsness further explains that “[t]he compute node 130 may be a physical host or a virtual machine,” and that “[a] virtual machine provides a software abstraction of computing resources of a physical host computer” and “runs as an independent computer system.” EX1004, 8:37-44. It is my opinion that a compute node containing a processor, network interface, memory, and storage is a **computer system**, whether it is implemented as a physical host or as a virtual

40

machine. A POSITA would have understood each Barsness compute node to be a *computer system*.

52. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶52) states:

The management system 105 is also a *computer system* because it includes a "central processing unit (CPU), a network interface 315, an interconnect 320, a memory 325, and storage 330." EX1004, 9:53-59. Barsness FIG. 3 shows details of a management system:

(Figure omitted. EX1004, FIG. 3)

Id., 9:60-10:64 (further describing the management system); *see also* EX1001, 14:63-15:23 ('424 patent describing an exemplary "computer system").

I agree with Dr. Crovella because, in my opinion, the management system 105 is likewise a *computer system*. Barsness describes it as including a "central processing unit (CPU), a network interface 315, an interconnect 320, a memory 325, and storage 330." EX1004, 9:53-59. FIG. 3 of Barsness shows those components:

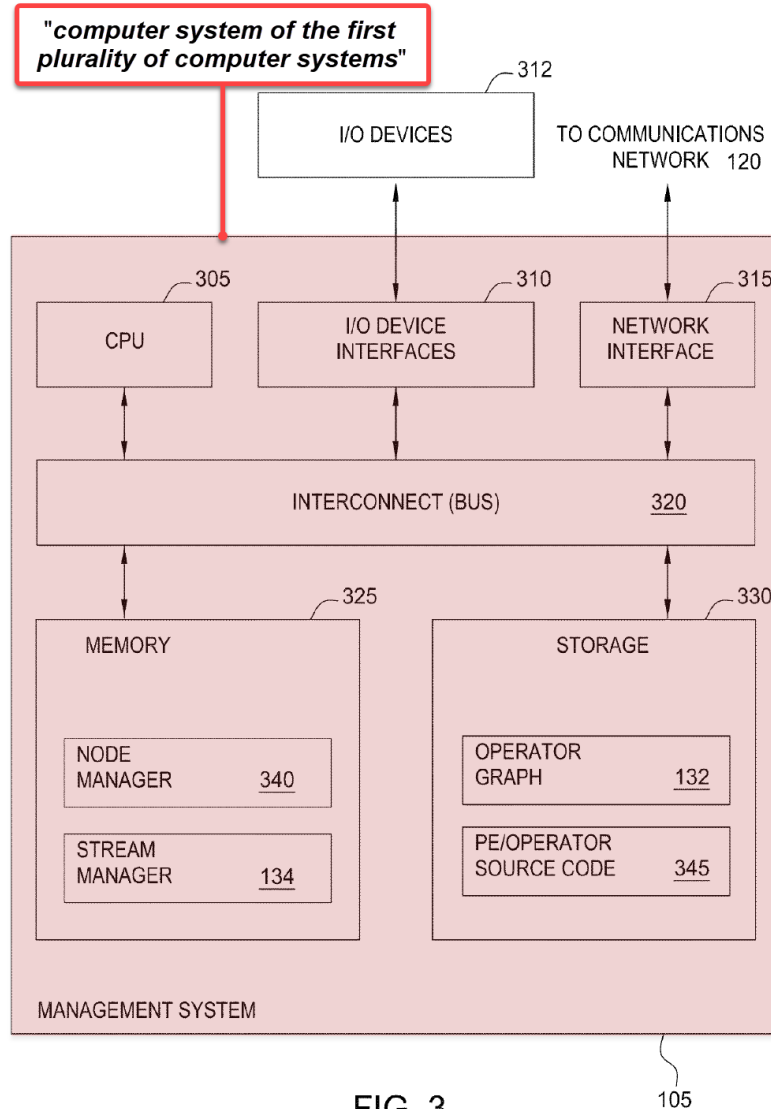
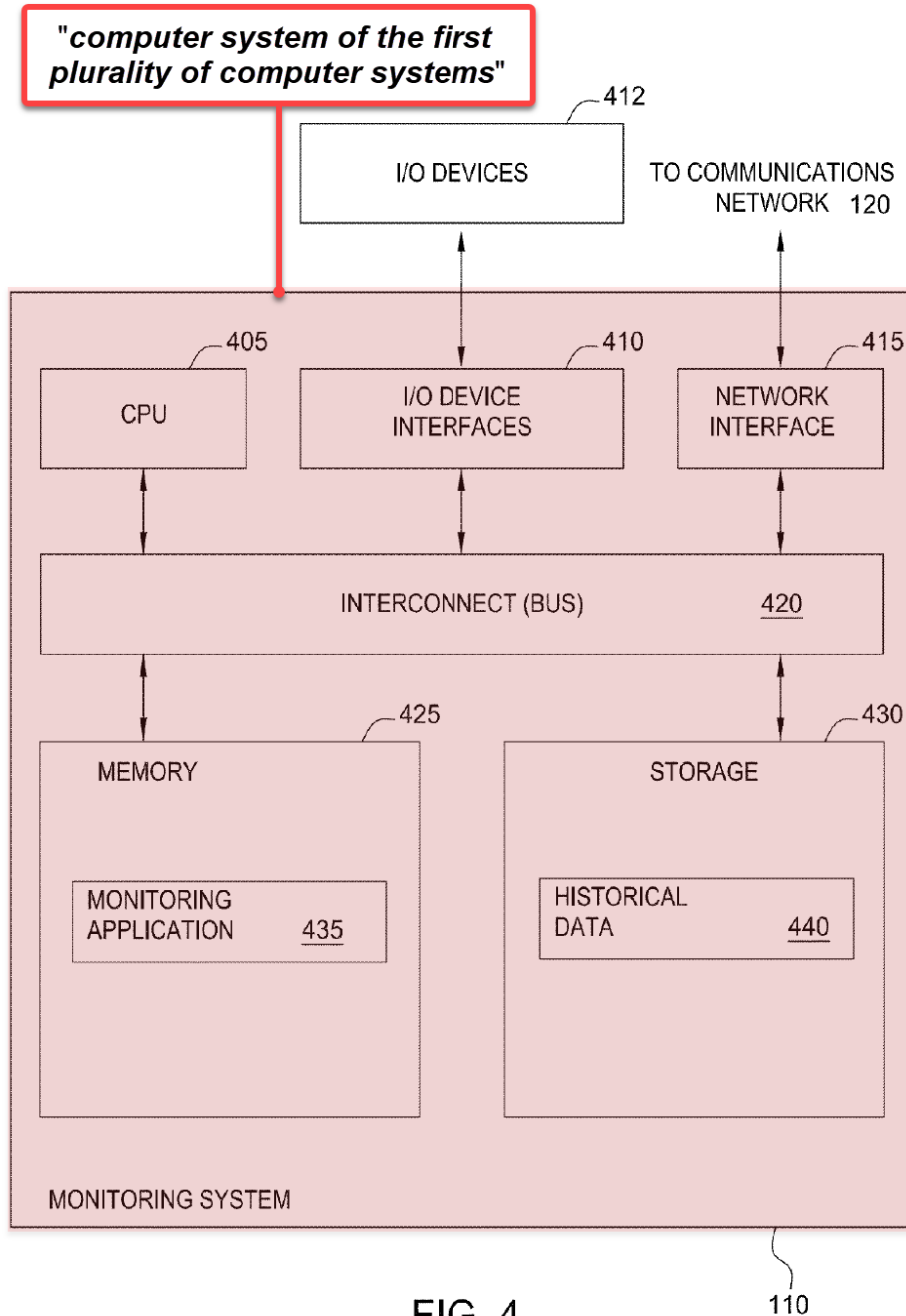


FIG. 3

EX1004, FIG. 3; *id.*, 9:60-10:64; *see also* EX1001, 14:63-15:23 (the '424 patent describing an exemplary "computer system"). It is my opinion that the management system, like each compute node, has the processor, network interface, interconnect, memory, and storage of a **computer system**. A POSITA would have understood the Barsness management system to be a **computer system** as well. Similarly, the Barsness monitoring system 110 is also a **computer system** as seen in FIG. 4 below:



EX1004, FIG. 4 (annotations added). Additionally, monitoring system 110 “may be a cluster of a plurality of computing resources.” EX1004, 7:17-21.

3. **[1.2.a] - wherein each respective computer system of the first plurality of computer systems comprises a memory that stores a respective first data,**

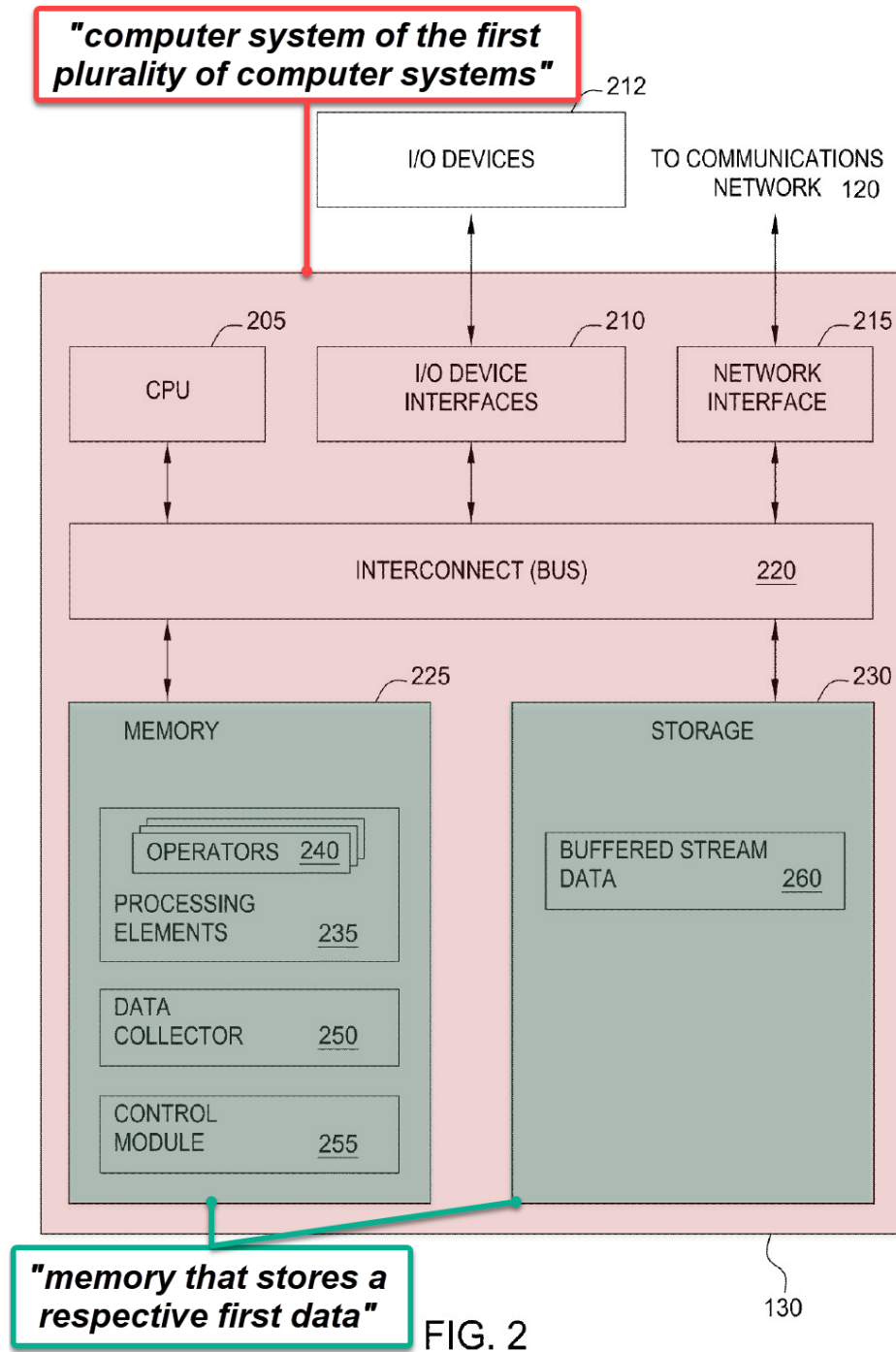
53. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶53)

states:

Barsness satisfies this element. Each compute node (*respective computer system*) of the plurality of compute nodes (*the first plurality of computer systems*) in Barsness comprises "memory 225" (*a memory*) that "store[s]" data such as programming instructions and "application data" (*stores a respective first data*). EX1004, 8:53-55. The management system 105, also a *respective computer system of the first plurality of computer systems*, also comprises "memory 325" (*a memory*) that stores data, e.g., "programming instructions and application data." EX1004, 9:62-67. Each also includes a "storage" 230/330 that stores data, e.g., buffered stream data and operator graph 132. EX1004, 8:61-65, 10:3-11. The memory and storage (each alone and collectively, *a memory*) of a compute node and management system are depicted in FIGS. 2 and 3 of Barsness, as discussed below.

I agree with Dr. Crovella because, in my opinion, each *respective computer system* of the *first plurality of computer systems* in Barsness comprises a *memory* that *stores a respective first data*. Barsness discloses that compute nodes 130₁₋₄, management system 105, and monitoring system 110 each stores operator graph 132 in memory. Each compute node 130 (*respective computer system*) comprises "memory 225," (*a memory*) which "store[s]" data such as programming instructions and "application data" (*stores a respective first data*). EX1004, 8:53-55. The management system 105, also a *respective computer system of the first plurality of*

computer systems, likewise comprises “memory 325” (*a memory*) that stores data, including “programming instructions and application data.” EX1004, 9:62-67. Monitoring system 110’s (*respective computer system*) “memory 425” stores programming instructions and application data (*a memory*). EX1004, 11:7-11. Each also includes a “storage” (230 and 330 and 430) that stores data, such as buffered stream data and the operator graph 132. EX1004, 8:61-65, 10:3-11, 11:14-18. It is my opinion that this memory and storage, whether considered alone or together, provide a *memory* that *stores a respective first data* in each such *computer system*, as FIGS. 2, 3, and 4 of Barsness show and as discussed below. A POSITA would have understood each *respective computer system* in Barsness to include a *memory* that *stores a respective first data*.



EX1004, FIG. 2 (annotations added).

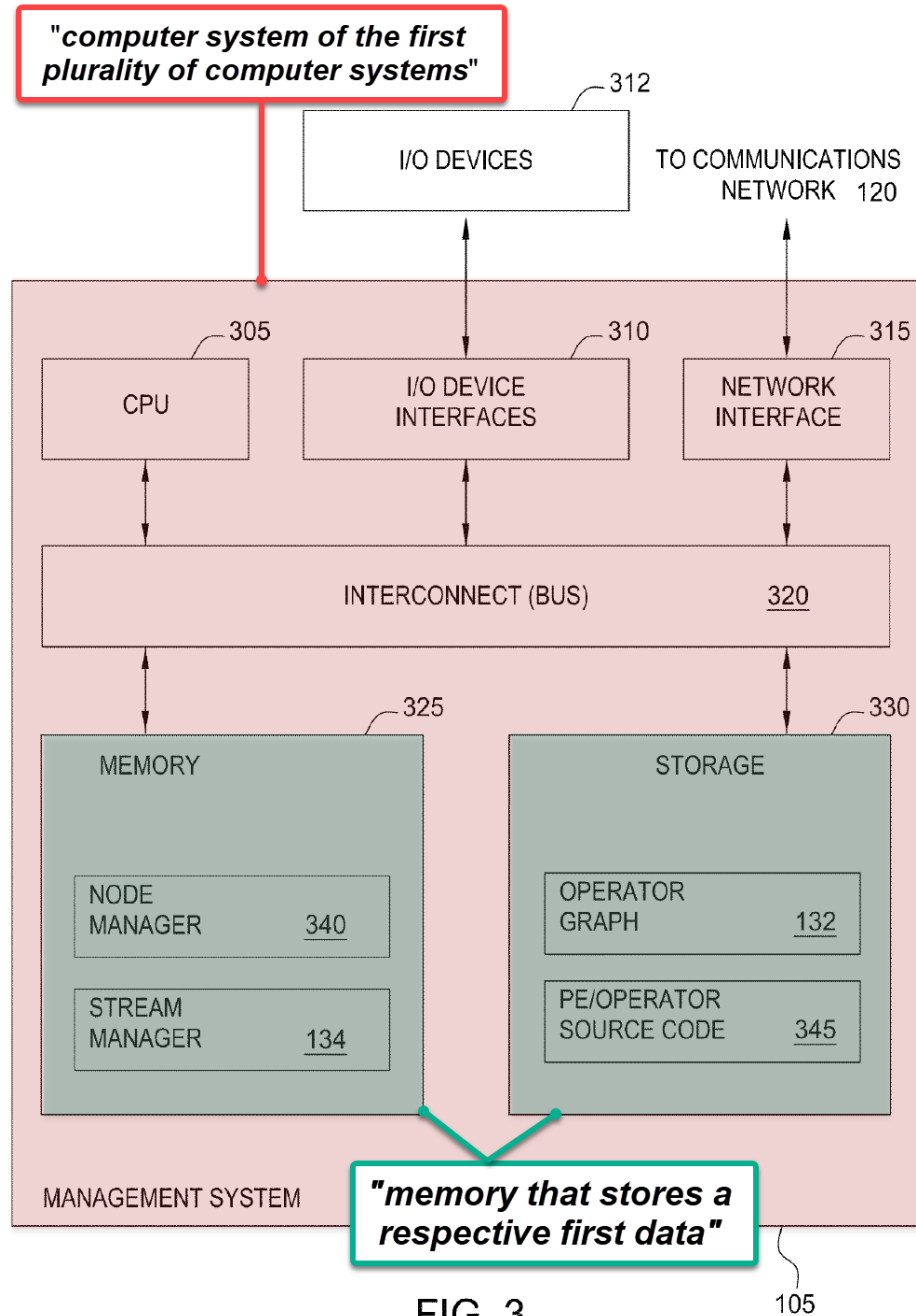
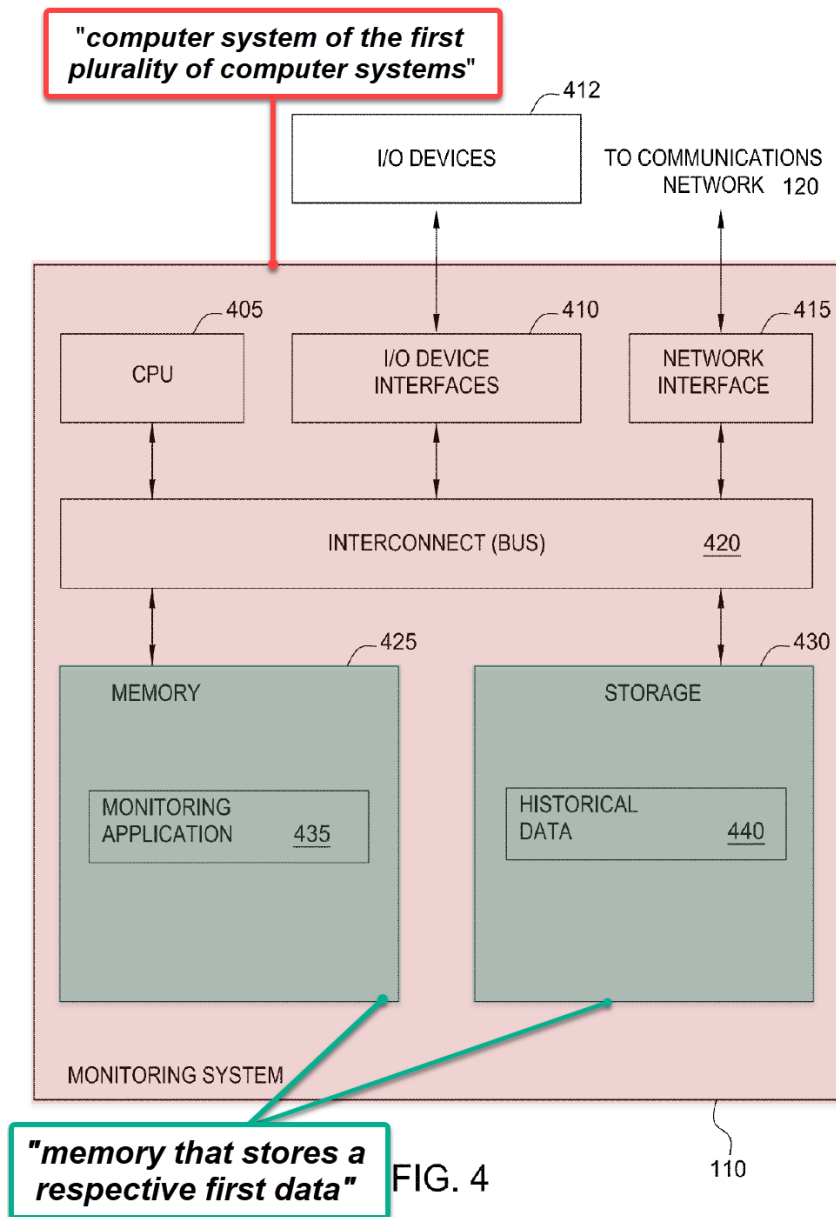


FIG. 3

EX1004, FIG. 3 (annotations added).



EX1004, FIG. 4 (annotations added).

54. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶54) states:

The "memory 225 is generally included to be representative of a random access memory," while "[s]torage 230, such as a hard disk drive, solid state device (SSD), or flash memory storage drive, may store non-volatile data." EX1004, 8:61-65; *id.*, 10:3-11 (similarly for

the management system). The memory and storage of a compute node are depicted in FIG. 2 of Barsness:

(Figure omitted. EX1004, FIG. 2 (highlighting added))

I agree with Dr. Crovella because, in my opinion, the memory and storage that Barsness describes are conventional forms of the claimed *memory*. Barsness explains that “memory 225 is generally included to be representative of a random access memory,” while “[s]torage 230, such as a hard disk drive, solid state device (SSD), or flash memory storage drive, may store non-volatile data.” EX1004, 8:61-65; *id.*, 10:3-11 (the same being true for the management system). FIG. 2 of Barsness depicts the memory and storage of a compute node. It is my opinion that both the random access memory and the non-volatile storage are forms of *memory* capable of storing the respective first data. A POSITA would have understood these components to provide the claimed *memory*.

55. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶55) states:

The management system 105 (*respective computer system*) comprises memory 325 and storage 330 that stores, among other things, an operator graph 132 (*a respective first data*), as shown in FIG. 3 of Barsness:

(Figure omitted. EX1004, FIG. 3 (highlighting added)) *Id.*, 10:12-64.

I agree with Dr. Crovella because, in my opinion, the management system 105, which is itself a *respective computer system*, comprises memory 325 and storage 330 that store, among other things, an operator graph 132, which is a *respective first*

data. FIG. 3 of Barsness shows this. EX1004, FIG. 3; *id.*, 10:12-64. It is my opinion that the operator graph 132 stored in the management system's memory and storage is exactly the kind of *respective first data* the claim requires. A POSITA would have understood the management system to store a *respective first data*, including the operator graph 132, in its *memory*. Similarly, monitoring system 110's (*respective computer system*) memory 425 and storage 430 also stores operator graph 132. EX1004, 11:7-11, 11:27-49; *see also* EX1004, 11:53-55 ("The GUI 510 may configured to depict the operator graph 132 of a selected stream application.").

56. As I further explain in the next section, respective "data" stored in each respective memory of Barsness includes a *respective first data*. For example, the memory of each compute node, management system 105, and monitoring system 110 includes the code and configuration of processing elements of the operator graph (*first data*). EX1004, 8:66-9:23 (describing the "executable code" of processing element operators on compute nodes), 10:51-53 (describing "processing element and operator source code 345" as stored on the management system), 11:52-12:8 (describing the monitoring system displaying the configuration of processing elements running across compute nodes).

4. **[1.2.b] - wherein the respective first data represents a respective portion of a distributed computational graph,**

57. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶57)

states:

Barsness satisfies this element. Regarding a **distributed computational graph**, Barsness discloses an “operator graph” (*distributed computational graph*) that, for example, “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, FIGS. 1A-1B; *id.*, 6:49-53 (“[T]he operator graph 132 represents a stream application beginning from of one or more source processing elements (PEs) through to one or more sink PEs.”), 10:49-51 (“[T]he operator graph 132 maintains the connectivity configuration of a plurality of compute nodes and processing elements in a streaming application.”).

I agree with Dr. Crovella because, in my opinion, the operator graph of Barsness is the claimed ***distributed computational graph***. Compute nodes 130, management system 105, and monitoring system 110 each store the operator graph 132 that describes the streaming application's compute nodes, their PEs, and the data flowing between those PEs and between compute nodes. Barsness discloses an “operator graph” that “represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data tuples through the stream application.” EX1004, 1:43-46; *id.*, 2:12-15, 2:59-62, 6:43-56, FIGS. 1A-1B. Barsness further explains that “[t]he operator graph 132 represents a stream application beginning from of one or more source processing elements (PEs) through to one or more sink PEs,” EX1004, 6:49-53, and that “[t]he operator graph 132 maintains the connectivity configuration of a plurality of compute nodes and

processing elements in a streaming application,” *id.*, 10:49-51. It is my opinion that this operator graph, which identifies the processing elements and the links among them and governs the flow of data tuples through the application, is a ***distributed computational graph***. A POSITA would have understood the Barsness operator graph to be a ***distributed computational graph***.

58. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶58) states:

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

(Figure omitted. EX1004, FIG. 1B (green and blue annotations added))

Processing elements are grouped into respective compute nodes 130 as shown. Each processing element includes an operator comprising executable code to perform an operation on data input to the processing element. EX1004, 3:16-35, 6:57-7:3 (explaining how “each processing element may be configured to carry out any form of data processing functions on a received N-tuple,” which can include any “data analytic functions or operations”), 9:3-8, FIG. 2.

I agree with Dr. Crovella because, in my opinion, FIG. 1B of Barsness depicts an example operator graph in which the processing elements are the graph nodes and the links between processing elements are the graph edges:

¹I am using the terms “graph nodes” and “graph edges” to distinguish these aspects of the operator graph from the “compute nodes” disclosed by Barsness.

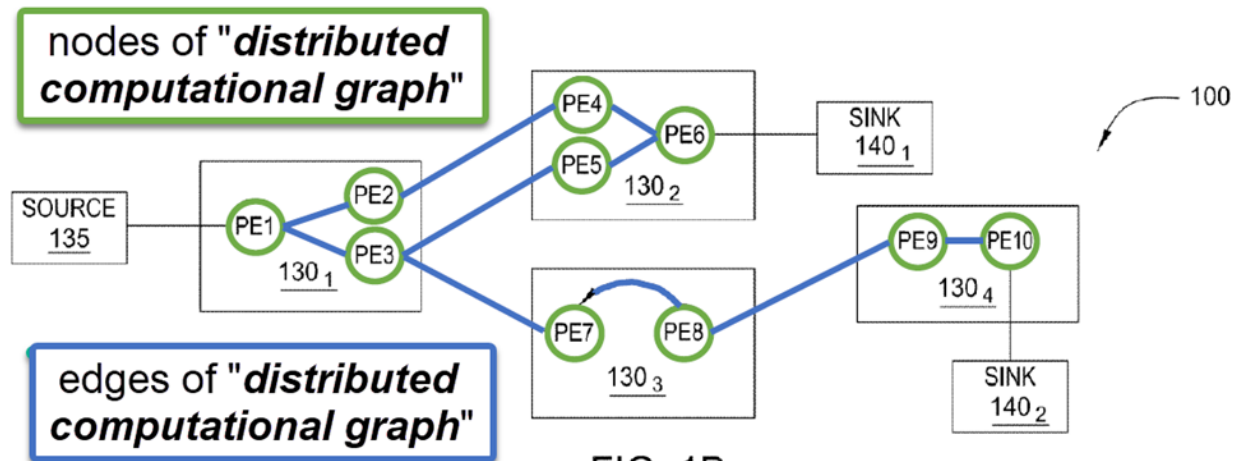


FIG. 1B

EX1004, FIG. 1B (annotations added). Processing elements are grouped into respective compute nodes 130₁₋₄, and each processing element includes an operator comprising executable code that performs an operation on the data input to that processing element. EX1004, 3:16-35, 6:57-7:3 (explaining that “each processing element may be configured to carry out any form of data processing functions on a received N-tuple,” which can include any “data analytic functions or operations”), 9:3-8, FIG. 2. It is my opinion that this node-and-edge structure is what one expects of a graph, with the processing elements serving as the nodes and the data links serving as the edges. A POSITA would have understood the Barsness operator graph to be a graph whose nodes are the processing elements and whose edges are the links between them.

59. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶59) states:

As noted above in paragraph 40, I have been asked to also construe “distributed computational graph” as a “directed graph with each transformation comprising a node and the output messages between transformations comprising edges.” The operator graph of Barsness meets this construction because each graph node corresponds to a processing element (PE) that performs a transformation on data in the stream and each graph edge corresponds to the output from a given graph node (transformation) to the next graph node in the operator graph. More precisely:

- The operator graph is *directed* because the data stream flows in one direction from the source graph node to the sink graph node(s), with each graph edge indicating the flow of data output from one PE graph node to the next PE graph node. EX1004, 1:43-46, 2:13-15, 2:59-62, 6:43-56, FIG. 1B; *see also* EX1001, 9:27-31 (’424 patent explaining: “Under circumstances when a relationship flows from one node to another in one direction, ... the order of the endpoints is significant. *Graphs with such edges are designated as ‘directed.’*”) (emphasis added). The arrow from PE8 back to PE7 depicted in Figure 1B confirms that the operator graph otherwise depicts the flow of data in a direction from left to right, from the source to the sink(s), in the figure.
- Each PE (graph node) comprises 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.” *See* paragraph 40. 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 (emphasis added). To that end, each PE includes an operator comprising “a small chunk of executable code configured to process data flowing into a processing element”. *Id.*, 9:3-9; *id.*, 3:16-22. The function performed by each PE results in a single stream of output emitted “to other PEs in the stream application.” *Id.*, 9:4-9; *see also id.*, 9:43-50; 7:55-56 (“Source PE1 emits tuples received by PE2 and PE3.”); *compare with* EX1001, 18:37-42 (describing node sending same “output stream to two transformational pipelines”). Barsness describes,

alternatively, that a PE “may split data elements,” thus confirming that absent such a split Barsness expects a single output stream. EX1004, 7:56-57. In that alternative example where the stream is split such that a portion of the stream is sent to one downstream PE while the remainder is sent to another downstream PE, together this bifurcated stream still comprises a single data stream output by a PE.

- The edges of the operator graph are “links between processing elements, corresponding to a flow of data tuples through the stream application,” and thus comprise the output messages between processing elements / operators, where the data stream emitted from a processing element comprises output messages from that processing element. EX1004, 2:12-15. A message is a unit of information passed between two elements, and each “tuple” emitted by a processing element is a message, or an “output message” because it is data output by the processing element. *E.g.*, EX1004, 6:57-65 (explaining that input and output data can be “in formats other than an N-tuple (e.g., the processing elements could exchange data marked up as XML documents)” and that “[o]f course, the N-tuple received by a processing element need not be the same N-tuple sent downstream”); *see also* EX1005, [0122] (describing “output messages” from an operator); EX1015, 12:11-14 (another IBM patent explaining that “streaming data items from one PE to another involves *message transmission*”). The ’424 patent does not provide any meaningful explanation of “output messages,” but instead uses the term to refer to data emitted from a transformation node. *E.g.*, EX1001, 17:17-19 (an “output message” is the “transformed” output of a transformation node).

The operator graph of Barsness is further described below in paragraphs 64-68.

I agree with Dr. Crovella because, in my opinion, the operator graph of Barsness satisfies the construction of ***distributed computational graph***. Similar, I have been

asked also to construe *distributed computational graph* as “a directed graph with each transformation comprising a node and the output messages between transformations comprising edges.” The operator graph of Barsness meets this construction because each graph node corresponds to a processing element that performs a transformation on data in the stream, and each graph edge corresponds to the output from a given graph node (a transformation) to the next graph node in the operator graph. Within the Barsness operator graph, each *graph node* is a processing element; Barsness uses the term “compute node” to refer to the computer systems that execute or run the processing elements. Specifically:

- “*directed graph*”: Operator graph 132 is *directed* because the data stream flows through the PEs (*nodes*) from source 135 to sinks 140₁₋₂. Each *edge* indicates the flow of data from one PE node to the next PE node. EX1004, 1:43-46, 2:12-15, 2:59-62, 6:43-56, FIG. 1B; *see also* EX1001, 9:27-31 (describing directed graphs); For example, Barsness explains that data flows from PE1 to PE2 and PE3, and from PE2 and PE3 to PE4 and PE5, and then from PE4 and PE5 to PE6. EX1004, 7:52-61.
- “*transformation comprising a node*”: Each processing element (*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.” Each processing element 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 processing element 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 processing element’s function *transforms* the input. *Id.*, 6:60-62 (“[T]he N-tuple received by a processing element need not be the same N-tuple sent downstream”); 8:12-15 (“PEs may be configured to *transform* incoming data tuples by either adding or removing one or more data attributes before emitting outgoing data tuples to other PEs”) (emphasis added). The function performed by each processing element results in a single stream of output emitted “to other PEs in the stream application.” *Id.*, 9:4-9; *see also id.*, 9:43-50; 7:55-56 (“Source PE1 emits tuples received by PE2 and PE3.”); *compare with* EX1001, 18:37-42 (describing node sending same “output stream to two transformational pipelines”). Barsness describes, alternatively, that a processing element “may split data elements” (EX1004, 7:56-57), which together still comprises a single output stream.

- “*output messages between transformations comprising edges*”: The edges of operator graph 132 are “links between processing elements,

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

It is my opinion that the Barsness operator graph is thus a directed graph whose nodes are transformations and whose edges are the output messages between them, and it accordingly meets the construction of *distributed computational graph*. I describe the operator graph of Barsness further below.

60. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶60) states:

Under the plain and ordinary meaning of distributed computational graph, the operator graph is a *distributed computational graph* because, as I just explained, 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 PEs.

I agree with Dr. Crovella because, in my opinion, under its plain and ordinary meaning the operator graph is also a *distributed computational graph*. It describes the distributed computations performed on a data stream, including the distribution of processing elements across different compute nodes and the flow of data between

those processing elements. A POSITA would have understood the Barsness operator graph to be a *distributed computational graph* under its plain and ordinary meaning.

61. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶61)

states:

Within each compute node, memory stores **the respective first data [that] represents a respective portion of a distributed computational graph**. Barsness teaches that each compute node stores in its memory data corresponding to processing elements 235 and operators 240 associated with that compute node. For example, “memory 225 includes one or more processing elements (PEs) 235.” EX1004, 8:66-9:1; *id.*, FIG. 2. As each PE is a node in the operator graph, in storing a given compute node's respective PEs, each compute node memory thus stores data representing a respective portion of the operator graph. Barsness further explains that: “PEs 235 includes a collection of operators 240. Each operator 240 may provide a small chunk of executable code configured to process data flowing into a processing element.” *Id.*, 9:3-12; *see also id.*, 9:13-33 (alternative manners of distributing operators and PEs across compute nodes); *id.*, 3:19-22 (“Scalability is achieved by distributing an 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.”), 2:3-4 (“The compute nodes are configured to execute processing elements of a stream application.”), 1:39-41 (“stream application composed from a plurality of processing elements executing on one or more compute nodes”), 1:55-57 (same). Source code for the processing element and operator is loaded by the management system into the compute node to setup the processing elements and operators. EX1004, 10:51-56 (“The processing element and operator source code 345 is loaded to setup the processing elements 235 and the operators 240 in the compute nodes 130.”). The operator graph of FIG. 1B shows how first compute node 130₁ includes PE1, PE2, and PE3, and thus stores respective first data representing a respective portion of the operator graph. EX1004, FIG. 1B. Similarly, the other compute nodes 130₂₋₄ store respective processing elements as depicted in the operator graph.

I agree with Dr. Crovella because, in my opinion, within each compute node the memory stores the *respective first data* that *represents a respective portion of a distributed computational graph*. Barsness teaches that each compute node stores in its memory the data for the processing elements 235 and operators 240 associated with that compute node. For example, “memory 225 includes one or more processing elements (PEs) 235.” EX1004, 8:66-9:1; *id.*, FIG. 2. Because each processing element is a node in the operator graph, in storing a compute node’s own processing elements, each compute node’s memory stores data representing a *respective portion* of the operator graph. Barsness further explains that “PEs 235 includes a collection of operators 240,” and that “[e]ach operator 240 may provide a small chunk of executable code configured to process data flowing into a processing element.” *Id.*, 9:3-12; *see also id.*, 9:13-33; *id.*, 3:19-22 (“Scalability is achieved by distributing an 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.”), 2:3-4, 1:39-41, 1:55-57. The management system loads the source code for the processing elements and operators into the compute node to set them up. EX1004, 10:51-56. As FIG. 1B shows, the first compute node 130₁ includes PE1, PE2, and PE3, so it stores *respective first data* representing a *respective portion* of the operator graph, and the other compute nodes 130₂₋₄ likewise store their respective processing elements as depicted in the operator graph. EX1004,

FIG. 1B. A POSITA would have understood each compute node's memory to store a *respective first data* that *represents a respective portion of a distributed computational graph*.

62. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶62) states:

The processing elements and operators stored in the memory 225 of the compute nodes are at least a part of the "programming instructions and application data" (*respective first data*) stored in the memory 225 of the compute nodes (*the first plurality of computer systems*) because the processing elements and operators comprise the stream application. EX1004, 8:53-55, 1:39-41, 2:3-4. Specifically, the operators stored in the memory of a compute node are *respective first data* because the operators are "executable pieces of code" that are each "configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132." EX1004, 3:19-22, 9:4-9. Within a given compute node's memory, the data for the PEs and operators *represents a respective portion of a distributed computational graph* because those PEs and related operators correspond to at least that portion of the Barsness operator graph (*distributed computational graph*) associated with the respective compute node. *E.g.*, EX1004, 2:12-15 ("The operator graph represents the plurality of processing elements, and links between processing elements"), 10:51-56 ("The processing element and operator source code 345 is loaded to setup the processing elements 235 and the operators 240 in the compute nodes 130."). Within the operator graph, each processing element (and its related operator) corresponds to a graph node of the operator graph. Thus, the data representing the PEs / operators (*respective first data*), loaded and stored in the memory of each compute node, *represents a respective portion of an operator graph (distributed computational graph)*.

I agree with Dr. Crovella because, in my opinion, the processing elements and operators stored in the memory 225 of the compute nodes are at least part of the

“programming instructions and application data,” that is, the *respective first data*, stored in the memory 225 of the compute nodes, which are the *first plurality of computer systems*, because the processing elements and operators make up the stream application. EX1004, 8:53-55, 1:39-41, 2:3-4. In particular, the operators stored in a compute node’s memory are *respective first data* because they are “executable pieces of code” each “configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132.” EX1004, 3:19-22, 9:4-9. Within a given compute node’s memory, the data for the PEs and operators *represents a respective portion of a distributed computational graph* because those processing elements and their operators correspond to at least that portion of the streaming application’s operator graph, the *distributed computational graph*, associated with the respective compute node. *E.g.*, EX1004, 2:12-15 (“The operator graph represents the plurality of processing elements, and links between processing elements”), 10:51-56. Within the operator graph, each processing element and its operator corresponds to a graph node. It is my opinion that the data representing the PEs and operators, loaded and stored in each compute node’s memory, is therefore a *respective first data* that *c*. A POSITA would have understood each compute node’s stored processing elements and operators to represent a *respective portion of a distributed computational graph*. In my opinion,

the data stored in the memory of each compute node represents a respective portion of the operator graph (*distributed computational graph*). Barsness discloses that the management system loads “processing element and operator source code 345 ... to setup the processing elements 235 and the operators 240 in the compute nodes 130.” EX1004, 10:51-56. Each operator is “a small chunk of executable code configured to process data flowing into a processing element ... and to emit data to other operators 240 in a PE and to other PEs in the stream application *as described by the operator graph 132*.” EX1004, 9:4-9 (emphasis added). The stored operator code and its associated routing information encode both the transformation performed at the corresponding graph node and the outgoing links from that node, thus, a respective portion of the operator graph. A POSITA would understand this stored data to represent that portion of the graph. Further, a processing element, as stored in compute-node memory, is itself data: executable operator code together with the routing and connection information the node uses to send its output to the correct downstream processing elements. As I explain below in connection with elements [1.3.c] and [1.3.d], each compute node stores connection information, such as the destination TCP/IP socket (destination IP address and destination port) for its downstream processing elements. EX1004, 3:16-18, 7:44-47, 9:9-12. A POSITA would have known this from both the Barsness reference and because it was established knowledge at the time that routing messages within distributed systems

often used some form of connection data. EX1017, 1:43-54; EX1016, 8:56-62 (stream connection data “represents the connections between PEs on compute node 130 ... as well as connections to other compute nodes 130 with upstream and[/]or downstream PEs”). That stored connection data represents the edges of the operator graph incident to the node, and the stored operator code represents the corresponding graph node. Data that encodes the nodes and edges of a portion of a graph is, in my opinion, data that “represents a respective portion of” that graph within the meaning of the claim. This stored data—the “programming instructions and application data” residing in memory 225 (EX1004, 8:53-55)—is a part of the *respective first data* I identified in connection with element [1.2.a]; the processing elements, operators, their connection data, and their configuration data are “application data.” EX1004, 8:53-55, 8:66-9:1, 1:39-41, 2:3-4. Additionally, the executable code for the operators of each processing element are “programming instructions” (*first data*) and representative of that processing element (*a respective portion of a distributed computational graph*).

63. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶63) states:

The *respective portion of a distributed computational graph* stored in the memory (memory 325 and storage 330) of the management system 105 includes the full operator graph 132 (*respective first data*), which allows the management system 105 “to control a stream application running on the compute nodes 130₁₋₄ as well as change the structure of the operator graph 132.” EX1004, 7:27-29; *id.*, 6:48-56. By storing the

operator graph 132 in the management system 105, the node manager 340 and stream manager 134 of the management system 105 can “mov[e] processing elements (PEs) from one compute node 130 to another, e.g., to manage the processing loads of the compute nodes 130,” and otherwise build and configure the operator graph. EX1004, 7:27-37; *id.*, 10:12-64 (describing management of the operator graph by the management system).

I agree with Dr. Crovella because, in my opinion, the *respective portion of a distributed computational graph* stored in the memory of the management system 105, namely memory 325 and storage 330, includes the full operator graph 132, which is a *respective first data*. That full operator graph allows the management system 105 “to control a stream application running on the compute nodes 130₁₋₄ as well as change the structure of the operator graph 132.” EX1004, 7:27-29; *id.*, 6:48-56. By storing the operator graph 132, the node manager 340 and stream manager 134 of the management system 105 can “mov[e] processing elements (PEs) from one compute node 130 to another, e.g., to manage the processing loads of the compute nodes 130,” and otherwise build and configure the operator graph. EX1004, 7:27-37; *id.*, 10:12-64. It is my opinion that, by storing the full operator graph 132, the management system’s memory stores a *respective first data* that *represents a respective portion of a distributed computational graph* (indeed, the whole of it). A POSITA would have understood the management system’s memory to store a *respective first data* representing a *respective portion of a distributed computational graph*. Similarly, the monitoring system 110 (*respective computer*

system) also includes memory (memory 425 and storage 430) storing monitoring application 435 (*first data*) including operator graph 132 (*respective first data that represents a respective portion of a distributed computational graph*). EX1004, 3:45-50 (describing monitoring system 110's "graphical representation of the operator graph"). 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. A POSITA would have understood the monitoring system's memory to store a *respective first data* representing a *respective portion of a distributed computational graph*.

5. [1.2.c] - and wherein the distributed computational graph describes a flow of output data of a first transformation pipeline to an input of a second transformation pipeline,

64. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶64) states:

Barsness satisfies this element. The operator graph of Barsness is a distributed computational graph, as I explained above in paragraphs 57-59. The operator graph describes the flow of data through several compute nodes, each containing a group of processing elements (PEs)

that process / transform the data flowing through them. *E.g.*, EX1004, 8:11-15 (“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, as per the configuration of target PEs.”); *see also id.*, 7:52-54.

I agree with Dr. Crovella because, in my opinion, the operator graph of Barsness, which is a ***distributed computational graph*** as I explained above, describes the flow of data through several compute nodes, each of which contains a group of processing elements that process and transform the data flowing through them. *E.g.*, EX1004, 8:11-15 (“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, as per the configuration of target PEs.”); *see also id.*, 7:52-54. A POSITA would have understood the Barsness operator graph to describe the flow of data through the processing elements distributed across the compute nodes.

65. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶65) states:

Further, the group of PEs for a given compute node taught by Barsness satisfies a *transformation pipeline*, as construed above in paragraph 40, because those PEs form “a progression of functions which each perform a transformation on the data stream” passing through the compute node. In the Barsness stream data processing application, “operators are connected to one another such 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 operation on the data via its operator and emits the transformed data to the next PE(s) in the graph. EX1004, 6:57-7:3; *id.*, 6:55-56 (“Data elements flow into a source PE of a stream application and are processed by that PE.”), 3:16-22 (describing operators), 9:4-7

(explaining how PEs on a compute node include “a collection of operators” that “provide a small chunk of executable code configured to process data flowing into a processing element”); *see also* paragraph 59. Barsness teaches that each PE may have “only one” active operator. EX1004, 9:9-23. The group of PEs within a given compute node thus comprise a progression of functions that each perform some action or transformation on the data stream.

I agree with Dr. Crovella because, in my opinion, the group of processing elements for a given compute node is a ***transformation pipeline*** as construed in paragraph 40, because those processing elements form “a progression of functions which each perform a transformation on a data stream” passing through the compute node. In the Barsness stream data processing application, “operators are connected to one another such 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 operation on the data through its operator and emits the transformed data to the next PE or PEs in the graph. EX1004, 6:57-7:3; *id.*, 6:55-56 (“Data elements flow into a source PE of a stream application and are processed by that PE.”), 3:16-22, 9:4-7 (each PE on a compute node includes “a collection of operators” that “provide a small chunk of executable code configured to process data flowing into a processing element”). Barsness teaches that each PE may have “only one” active operator. EX1004, 9:9-23. It is my opinion that the group of PEs within a given compute node is therefore a progression of functions that each perform a transformation on the data stream, which is a ***transformation pipeline***. A

POSITA would have understood the group of PEs in a compute node to form a *transformation pipeline*.

66. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶66) states:

For example, the operator graph depicted in Barsness FIG. 1B "begins at a source PE 135 (labeled as PE1) and ends at sink PEs 140₁₋₂ (labeled as PE6 and PE10)." EX1004, 7:52-54. As shown in FIG. 1B, each compute node (boxes 130₁₋₄) contains a group of PEs that form a *transformation pipeline*. Here, the progression of transformations performed by PE1, PE2, and PE3 in the compute node 130₁ is a *first transformation pipeline*. A pipeline can include branching and iterative (cyclical) configurations. *E.g.*, EX1001, 9:67-10:7.

I agree with Dr. Crovella because, in my opinion, FIG. 1B of Barsness illustrates these pipelines. The operator graph "begins at a source PE 135 (labeled as PE1) and ends at sink PEs 140₁₋₂ (labeled as PE6 and PE10)." EX1004, 7:52-54. As FIG. 1B shows, each compute node (boxes 130₁₋₄) contains a group of processing elements that form a *transformation pipeline*. The progression of transformations performed by PE1, PE2, and PE3 in compute node 130₁ is a *first transformation pipeline*. A pipeline may include branching and iterative, or cyclical, configurations. *E.g.*, EX1001, 9:67-10:7. A POSITA would have understood each compute node's group of processing elements in FIG. 1B to form a *transformation pipeline*.

67. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶67) states:

Accordingly, as shown in annotated FIG. 1B below, the transformations performed by processing elements PE1, PE2, and PE3 form a first transformation pipeline, the transformations performed by processing elements PE4, PE5, and PE6 form a second transformation pipeline, and the operator graph describes a flow of output data from the first transformation pipeline to inputs (PE4 and PE5) of the second transformation pipeline, as shown by the graph edges between PE2-PE4 and PE3-PE5 respectively:

(Figure omitted. EX1004, FIG. 1B (highlighting and red annotations added))

Id., 7:52-67 (describing the flow of data tuples through the nodes and processing elements of the example operator graph).

I agree with Dr. Crovella because, in my opinion, as the annotated FIG. 1B shows, the transformations performed by processing elements PE1, PE2, and PE3 form a ***first transformation pipeline***, the transformations performed by processing elements PE4, PE5, and PE6 form a ***second transformation pipeline***, and the operator graph describes a ***flow of output data*** from the ***first transformation pipeline*** to the inputs (PE4 and PE5) of the ***second transformation pipeline***, as shown by the graph edges between PE2 and PE4 and between PE3 and PE5. EX1004, FIG. 1B; *id.*, 7:52-67.

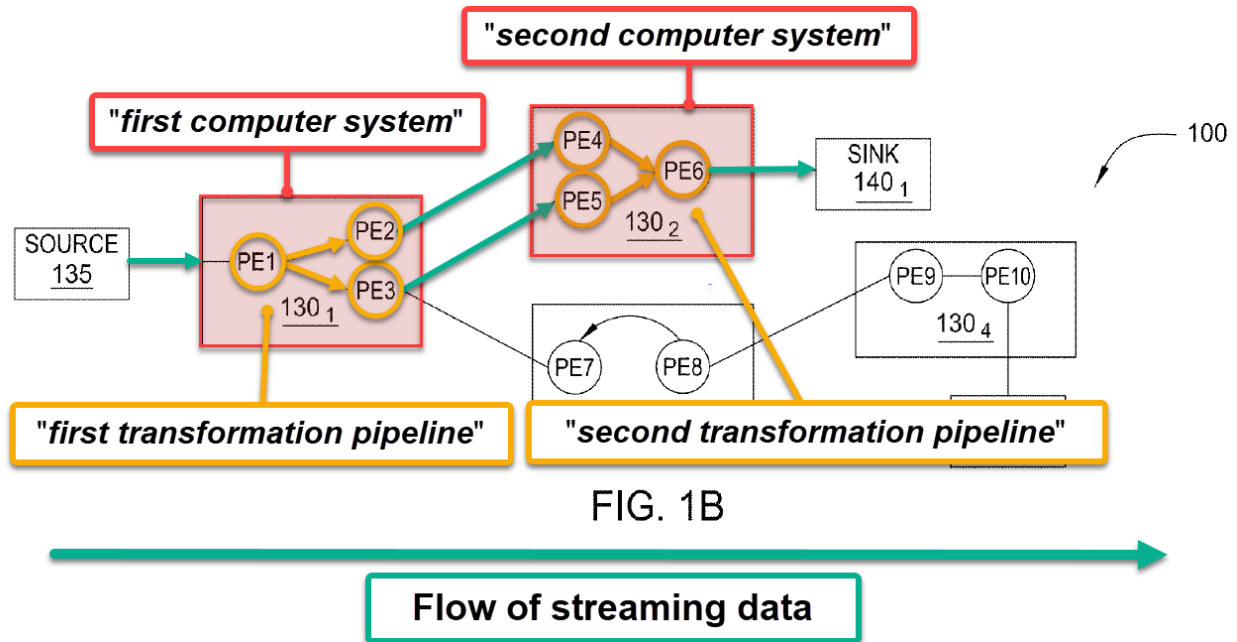


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 7:52-67 (describing the flow of data tuples through the nodes and PEs of the operator graph). Compute node 130₁'s PE1, PE2, and PE3 each apply operators to data elements flowing from PE1 to PE2 and PE3 (**first transformation pipeline**). PE1 processes the received data elements and “emits tuples received by PE2 and PE3.” EX1004, 7:52-57; *see also id.*, 6:55-56, 6:57-60. Then PE2 and PE3 process the received tuples and emit additional tuples. EX1004, 7:58-59, 7:64-67. 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. The **second transformation pipeline** comprises the data processing operations of PE4, PE5, and PE6. It is my opinion that Barsness thus describes a **flow of output data** from a **first transformation pipeline** to an input of a **second transformation pipeline**, as the

claim requires. A POSITA would have understood the Barsness operator graph to describe a *flow of output data* from a *first transformation pipeline* to an input of a *second transformation pipeline*.

68. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶68) states:

Other combinations of processing elements can be considered pipelines as well. For example, the combinations of processing elements PE1-PE2 and PE1-PE3 are each a transformation pipeline for the same reason PE1-PE2-PE3 is a transformation pipeline. Similarly, PE4-PE5 and PE4-PE6 are transformation pipelines for the same reason PE4-PE5-PE6 is a transformation pipeline. These examples show Barsness operator graph implementations that describe data streams flowing through a series of interconnected transformation pipelines. The processing elements PE7 and PE8 in the third compute node 130₃, where data flows from PE7 to PE8 and then loops back to PE7 (e.g., to perform iterative or cyclical processing before passing to PE9), are also a (second) transformation pipeline. EX1004, 7:64-67. The processing elements PE9 and PE10 in the fourth compute node 130₄ are also a transformation pipeline.

I agree with Dr. Crovella because, in my opinion, other combinations of processing elements are *transformation pipelines* as well. For example, the combinations PE1-PE2 and PE1-PE3 are each a *transformation pipeline* for the same reason PE1-PE2-PE3 is, and PE4-PE5 and PE4-PE6 are *transformation pipelines* for the same reason PE4-PE5-PE6 is. These examples show Barsness operator-graph implementations that describe data streams flowing through a series of interconnected *transformation pipelines*. The processing elements PE7 and PE8 in the third compute node 130₃, where data flows from PE7 to PE8 and then loops back to PE7, for example to

perform iterative or cyclical processing before passing to PE9, are also a *transformation pipeline*. EX1004, 7:64-67. The processing elements PE9 and PE10 in the fourth compute node 130₄ are also a *transformation pipeline*. A POSITA would have understood these combinations of processing elements to form *transformation pipelines*.

69. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶69) states:

Under the plain and ordinary meaning of transformation pipeline, the group of PEs for a given compute node is a *transformation pipeline* because they form a pipeline of transformations, where each PE performs a transformation and the group of PEs forms a series of transformations on a data stream. EX1004, 6:57-7:3; *id.*, 6:55-56, 3:16-22, 9:4-7.

I agree with Dr. Crovella because, in my opinion, under its plain and ordinary meaning the group of PEs for a given compute node is also a *transformation pipeline*, because they form a pipeline of transformations in which each PE performs a transformation and the group of PEs forms a series of transformations on a data stream. EX1004, 6:57-7:3; *id.*, 6:55-56, 3:16-22, 9:4-7. A POSITA would have understood the group of PEs for a given compute node to be a *transformation pipeline* under its plain and ordinary meaning.

6. [1.3] - wherein a first computer system of the first plurality of computer systems is configured to:

70. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶70)

states:

Barsness satisfies this element. For example, the first plurality of computer systems includes compute node 130₁, which is *a first computer system of the first plurality of computer systems*. EX1004, 7:54-55, 8:30-34. The first compute node (*first computer system*) within this plurality of systems is depicted in FIG. 1B of Barsness:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

See also id., FIG. 2 (depicting a compute node), 8:30-34 (explaining that “the compute node 130 includes, without limitation, a central processing unit (CPU) 205, a network interface 215, an interconnect 220, a memory 225, and storage 230”).

I agree with Dr. Crovella because, in my opinion, the *first plurality of computer systems* includes compute node 130₁, which is a *first computer system* of the *first plurality of computer systems*. EX1004, 7:54-55, 8:30-34. FIG. 1B of Barsness depicts that first compute node:

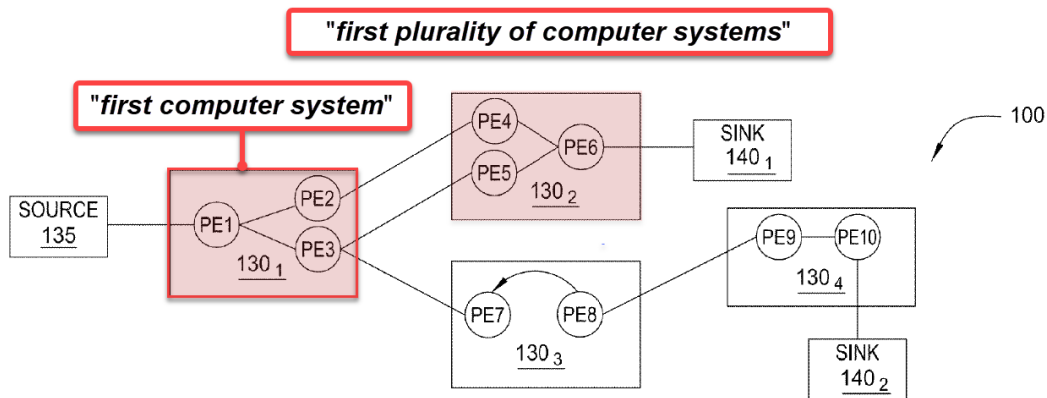


FIG. 1B

EX1004, FIG. 1B; *see also id.*, FIG. 2 (depicting a compute node), 8:30-34 (explaining that “the compute node 130 includes, without limitation, a central

processing unit (CPU) 205, a network interface 215, an interconnect 220, a memory 225, and storage 230”). A POSITA would have understood compute node 130₁ to be a *first computer system* of the *first plurality of computer systems*.

7. [1.3.a] - receive a first stream of input data from a first input feed,

71. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶71) states:

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

(Figure omitted. EX1004, FIG. 2 (highlighting added))

I agree with Dr. Crovella because, in my opinion, Barsness discloses receiving a *first stream of input data from a first input feed*. Barsness discloses “[d]ata elements flow[ing] into a source PE of a stream application,” which is a *first stream of input data*. EX1004, 6:49-56; *id.*, 7:52-54. Barsness also discloses “a data source for the stream application,” which is the *first input feed*, and “a storage space for data flowing into the compute node.” EX1004, 9:43-46. FIG. 2 shows the storage holding the “[b]uffered stream data 260.”

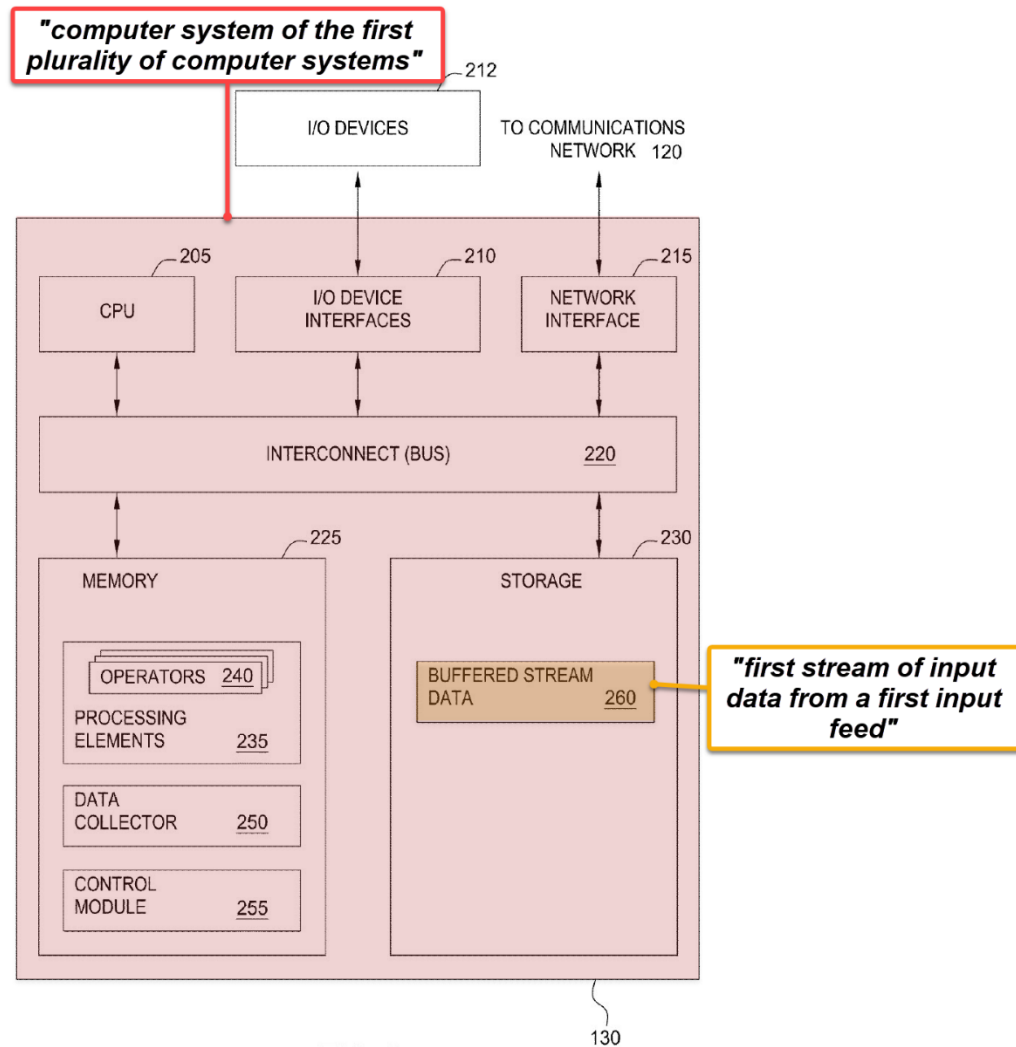


FIG. 2

EX1004, FIG. 2 (annotations added); see also EX1004, 1:22-24, 3:16-18, 9:43-50, 10:20-26. A POSITA would have understood Barsness to receive a *first stream of input data* from a *first input feed*.

72. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶72) states:

Barsness discloses that the "buffered stream data may include data tuples waiting to be processed by one of the PEs 235." EX1004, 9:43-47; *id.*, 9:47-50. A POSITA would have understood that the "buffered stream data" is the data elements received as an incoming stream from

an upstream source, because Barsness repeatedly refers to “streaming data” or “stream data” being processed by processing elements of a stream application. *E.g.*, EX1004, 10:40-48 (“The stream manager 134 may be used to monitor the *data stream* as it flows through the processing elements in a stream application.”) (emphasis added); *id.*, 1:22-24, 3:16-18, 9:43-50, 10:20-26.

I agree with Dr. Crovella because, in my opinion, the “buffered stream data” in Barsness is the incoming stream of data received from an upstream source. Barsness discloses that the “buffered stream data may include data tuples waiting to be processed by one of the PEs 235.” EX1004, 9:43-47; *id.*, 9:47-50. It is my opinion that a POSITA would have understood the “buffered stream data” to be the data elements received as an incoming stream from an upstream source, because Barsness repeatedly refers to “streaming data” or “stream data” being processed by the processing elements of a stream application. *E.g.*, EX1004, 10:40-48 (“The stream manager 134 may be used to monitor the data stream as it flows through the processing elements in a stream application.”); *id.*, 1:22-24, 3:16-18, 9:43-50, 10:20-26. A POSITA would have understood the buffered stream data to be the ***first stream of input data*** received by the compute node.

73. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶73) states:

Further, the first compute node 130₁ (*first computer system*) includes “a source PE 135 (labeled as PE1)” (EX1004, 7:52-54) that *receives* the “[d]ata elements flow[ing] into a source PE of a stream application,” as depicted in FIG. 1B:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Id., 6:49-56.

I agree with Dr. Crovella because, in my opinion, the first compute node 130₁, which is a **first computer system**, includes “a source PE 135 (labeled as PE1)” that receives the “[d]ata elements flow[ing] into a source PE of a stream application,” as FIG. 1B depicts. EX1004, 7:52-54; *id.*, 6:49-56. 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):

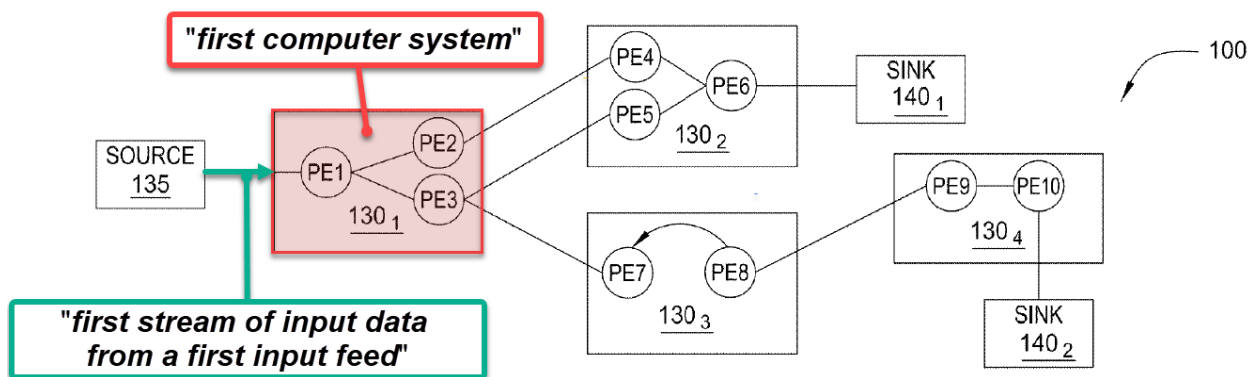


FIG. 1B

EX1004, FIG. 1B (annotations added), 6:49-56. A POSITA would have understood the first compute node to **receive** the *first stream of input data* at source PE 135.

8. [1.3.b] - process the first stream of input data substantially in real time by executing software instructions that apply the

**first transformation pipeline to the first stream of input data
to generate first pipeline output messages,**

74. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶74)

states:

Barsness satisfies this element. As I explained for elements 1.2.b through 1.2.c, each compute node processes the source streaming data using transformation pipelines comprising processing elements. *See* paragraphs 57-68. For example, the first compute node 130₁ (*first computer system*) of Barsness *processes the first stream of input data* from the source 135 *by applying a first transformation pipeline* via a plurality of processing elements, with processing elements PE1, PE2, and PE3 forming a first transformation pipeline applied to the data elements received from the first stream of input data at PE1:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

I agree with Dr. Crovella because, in my opinion, and as I explained for elements 1.2.b through 1.2.c, each compute node processes the source streaming data using transformation pipelines made up of processing elements. For example, the first compute node 130₁, which is a *first computer system*, processes the *first stream of input data* from the source 135 by applying a *first transformation pipeline* through a plurality of processing elements, with processing elements PE1, PE2, and PE3 forming a *first transformation pipeline* applied to the data elements received from

the *first stream of input data* at PE1, as FIG. 1B shows:

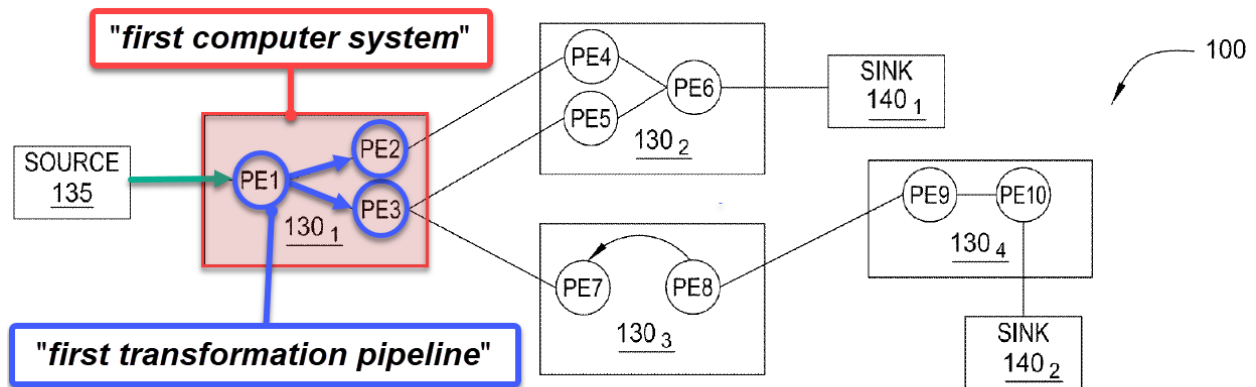


FIG. 1B

EX1004, FIG. 1B (annotations added). A POSITA would have understood the first compute node to apply a *first transformation pipeline* to the *first stream of input data*.

75. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶75)

states:

Regarding **executing software instructions that apply the first transformation pipeline**, Barsness teaches that a CPU of a given compute node / computing system, including compute node 130₁, “retrieves and executes programming instructions stored in the memory 225,” the memory storing one or more processing elements, and each processing element including an operator. EX1004, 8:53-54, 8:66-9:14; *see also id.*, 7:13-15 (“load programming instructions and configurations to be executed by the compute nodes 130₁₋₄”). Each processing element performs “data analytic functions or operations” according to the operator provided at the processing element. EX1004, 6:65-7:3; *id.*, 7:40-44 (“a processing element may be executed as an independently running process”). Further, each operator is a “small chunk of executable code configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132.” EX1004, 9:4-9; *id.*, 3:19-22 (“small

executable pieces of code (operators)”). The operators of processing elements in a compute node are executed by a CPU of the compute node, as shown in FIG. 2 of Barsness:

(Figure omitted. EX1004, FIG. 2 (highlighting added))

As the processing elements and operators in the compute node form the first transformation pipeline as described by the operator graph, executing the software instructions comprising the operators thus applies the first transformation pipeline to the incoming data stream.

I agree with Dr. Crovella because, in my opinion, the first compute node applies the first transformation pipeline by *executing software instructions*. Barsness teaches that a CPU of a given compute node, including compute node 130₁, “retrieves and executes programming instructions stored in the memory 225,” where the memory stores one or more processing elements and each processing element includes an operator. EX1004, 8:53-54, 8:66-9:14; *see also id.*, 7:13-15 (“load programming instructions and configurations to be executed by the compute nodes 130₁₋₄”). Each processing element performs “data analytic functions or operations” according to its operator. EX1004, 6:65-7:3; *id.*, 7:40-44 (“a processing element may be executed as an independently running process”). Each operator is a “small chunk of executable code configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132.” EX1004, 9:4-9; *id.*, 3:19-22. FIG. 2 shows that the operators of the processing elements in a compute node are executed by the compute node’s CPU. Barsness FIG. 2 shows this:

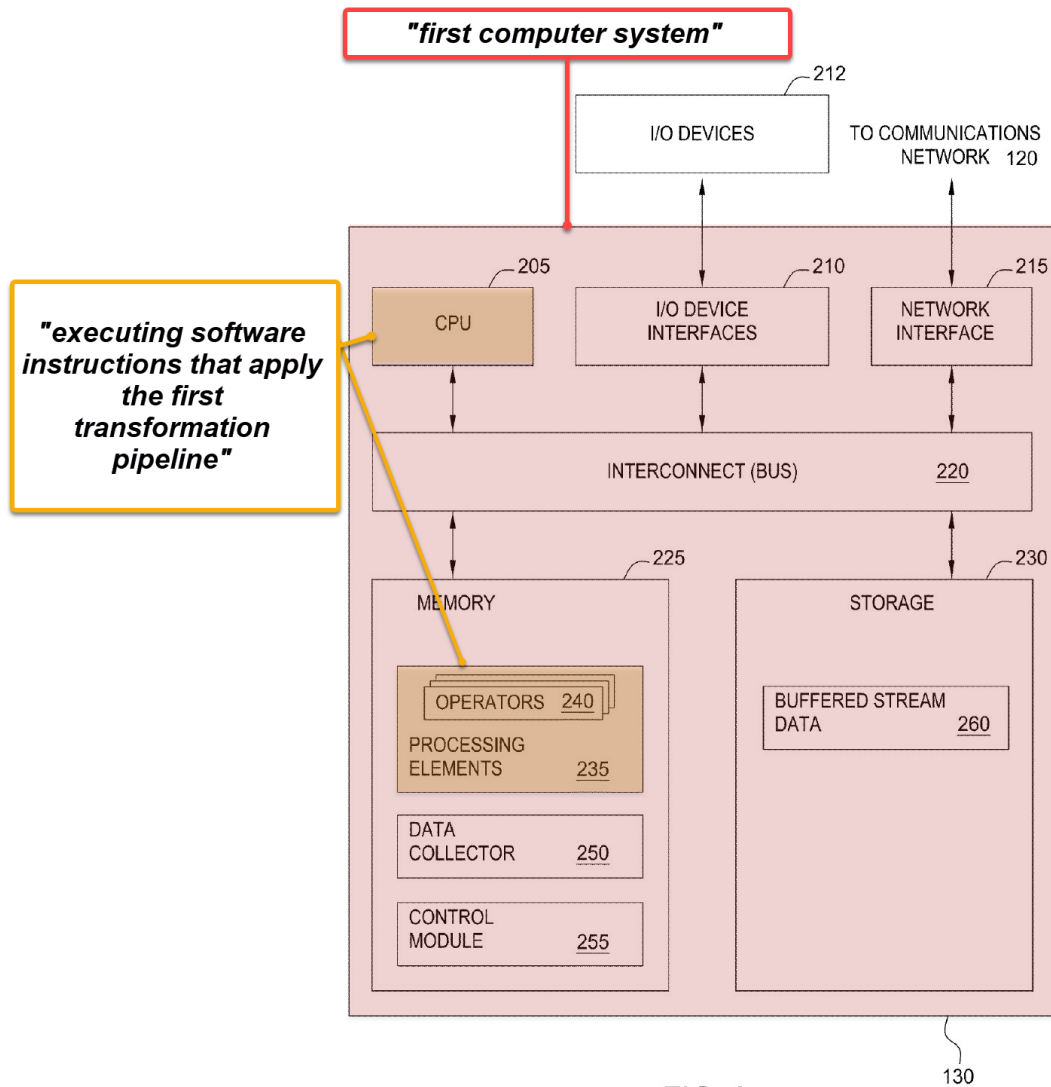


FIG. 2

EX1004, FIG. 2 (annotations added). It is my opinion that, because the processing elements and operators in the compute node form the *first transformation pipeline* as described by the operator graph, executing the software instructions that make up the operators applies the *first transformation pipeline* to the incoming data stream. A POSITA would have understood the compute node to *apply the first transformation pipeline* by *executing software instructions*.

76. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶76)

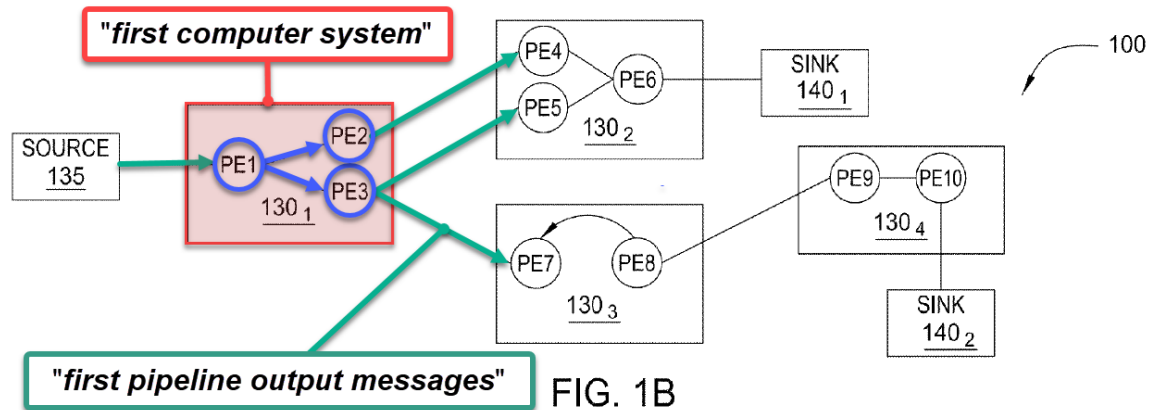
states:

Regarding **first pipeline output messages**, Barsness discloses that the “processing elements ... *emit an N-tuple of data elements into the stream* (except for a sink PE where the stream terminates).” EX1004, 6:57-60 (emphasis added). The output data emitted by the processing elements PE2 and PE3 are *first pipeline output messages* because they are the resulting output of the progression of transformations applied to the input data by the processing elements PE1, PE2, and PE3 (*the first transformation pipeline*):

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Id., 6:62-65 (PEs “receive or emit data in formats other than an N-tuple”), 7:52-67 (describing the flow of data tuples through the nodes and processing elements of the example operator graph); *see also* paragraphs 64-68 (element 1.2.c).

I agree with Dr. Crovella because, in my opinion, the output of the first transformation pipeline is the ***first pipeline output messages***. Barsness discloses that the “processing elements ... emit an N-tuple of data elements into the stream (except for a sink PE where the stream terminates).” EX1004, 6:57-60. The output data emitted by processing elements PE2 and PE3 are ***first pipeline output messages*** because they are the result of the progression of transformations applied to the input data by processing elements PE1, PE2, and PE3, that is, the ***first transformation pipeline***.



EX1004, FIG. 1B (annotations added); *id.*, 6:62-65 (PEs “receive or emit data in formats other than an N-tuple”), 7:52-67. A POSITA would have understood the output emitted by the *first transformation pipeline* to be the *first pipeline output messages*.

77. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶77) states:

Barsness teaches processing the data stream in real-time. EX1004, 2:52-3:15 (describing “real-time monitoring”). Barsness describes its “stream data processing application” system in the context of providing improvements to “real-time data processing,” i.e., “when the incoming data need to be processed in real-time,” and “without losing the real-time characteristics of the data processing.” EX1004, 2:58-61, 3:3-6, 3:16-18. Barsness further describes “processing logic to apply to inbound data records while they are ‘in flight,’ with the results available in milliseconds.” EX1004, 1:24-27.

I agree with Dr. Crovella because, in my opinion, Barsness teaches that the stream application processes the data stream *substantially in real time*. EX1004, 2:52-3:15 (describing “real-time monitoring”). Barsness describes its “stream data processing application” in the context of improvements to “real-time data processing,” that is,

“when the incoming data need to be processed in real-time,” and “without losing the real-time characteristics of the data processing.” EX1004, 2:58-61, 3:3-6, 3:16-18. Barsness further describes “processing logic to apply to inbound data records while they are ‘in flight,’ with the results available in milliseconds.” EX1004, 1:24-27. A POSITA would have understood Barsness to process the incoming data stream *substantially in real time*.

78. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶78) states:

The architecture of Barsness confirms that it teaches real-time data processing. For example, Barsness explains that database systems are slow for the needs of real-time data processing. EX1004, 1:15-19 (“For example, the rate at which a database system can receive and store incoming data can limit how much data can be processed or otherwise evaluated, which, in turn, limits the utility of database applications configured to process large amounts of data in real-time.”). In contrast, Barsness teaches an architecture configured such that input data streams proceed directly to processing, rather than first being stored in a database, with the data stream provided directly to source PE 135 of the first compute node 130₁. EX1004, 7:52-56, FIG. 1B.

I agree with Dr. Crovella because, in my opinion, the architecture of Barsness confirms that it processes data *substantially in real time*. Barsness explains that database systems are too slow for the needs of real-time data processing. EX1004, 1:15-19 (“[T]he rate at which a database system can receive and store incoming data can limit how much data can be processed or otherwise evaluated, which, in turn, limits the utility of database applications configured to process large amounts of data

in real-time.”). By contrast, Barsness teaches an architecture in which input data streams proceed directly to processing, rather than first being stored in a database, with the data stream provided directly to source PE 135 of the first compute node 130₁. EX1004, 7:52-56, FIG. 1B. A POSITA would have understood this architecture to process the data stream *substantially in real time*.

79. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶79) states:

Further, the monitoring application of Barsness monitors “current tuple processing rates at a selected PE” so that “the monitoring application 435 may act and reconfigure the processing modules of the stream application” when the throughput of a particular processing element slows down, to maintain real-time processing. EX1004, 12:28-36; *id.*, 3:61-4:2 (describing cloning a PE “that may be running too slowly to process data tuples through the stream”), 12:40-13:3 (discussing monitoring processing rates of a selected processing element, with example of “receiving 100 tuples per second at the input but only producing 30 tuples at the output” potentially indicating “processing issues at the selected processing element” where the element historically processed 70 tuples per second), 13:12-14 (disclosing “employ[ing] corrective measures automatically when a discrepancy in tuple processing rates is found”), 13:24-29 (describing adding an operator to maintain minimum processing speed), 14:30-39 (describing “corrective measures” taken to maintain the tuple processing rate, which “include adjusting the run-time priority of a particular processing job in the stream application to bring the current tuple processing rates within the pre-configured range”), FIG. 6 (at 612, “employing corrective measures in the stream application according to pre-set configurations”). These dynamic pipeline adjustments, which are used to ensure that the data stream is processed with sufficient throughput, further show that the compute nodes, including the first compute node 130₁ (the first computer system), process the input data stream substantially in real time. EX1004, 3:3-6.

I agree with Dr. Crovella because, in my opinion, the monitoring application of Barsness monitors the “current tuple processing rates at a selected PE” so that “the monitoring application 435 may act and reconfigure the processing modules of the stream application” when a particular processing element slows down, in order to maintain real-time processing. EX1004, 12:28-36; *id.*, 3:61-4:2 (cloning a PE “that may be running too slowly to process data tuples through the stream”), 12:40-13:3 (monitoring the processing rates of a selected processing element, for example “receiving 100 tuples per second at the input but only producing 30 tuples at the output” as “processing issues at the selected processing element” where it historically processed 70 tuples per second), 13:12-14 (“employ[ing] corrective measures automatically when a discrepancy in tuple processing rates is found”), 13:24-29 (adding an operator to maintain a minimum processing speed), 14:30-39 (“corrective measures” that “include adjusting the run-time priority of a particular processing job in the stream application to bring the current tuple processing rates within the pre-configured range”), FIG. 6 (at 612, “employing corrective measures in the stream application according to pre-set configurations”). It is my opinion that these dynamic pipeline adjustments, which ensure that the data stream is processed with sufficient throughput, confirm that the compute nodes, including the first compute node 130₁, process the input data stream *substantially in real time*.

EX1004, 3:3-6. A POSITA would have understood the compute nodes of Barsness to process the input data stream *substantially in real time*.

9. [1.3.c] - process the respective first data stored in the memory of the first computer system to determine information about the second transformation pipeline,

80. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶80)

states:

Barsness satisfies this element. As I explained above in paragraphs 53-63 (elements 1.2.a-1.2.b), Barsness discloses storing first data representing a respective portion of an operator graph (*distributed computational graph*) in the memory of each respective compute node (each a *computer system*), including the first compute node 130₁ (*the first computer system*).

I agree with Dr. Crovella because, in my opinion, and as I explained above for elements 1.2.a and 1.2.b, Barsness discloses storing the *respective first data* representing a respective portion of an operator graph, the *distributed computational graph*, in the memory of each respective compute node, each of which is a computer system, including the first compute node 130₁, the *first computer system*. A POSITA would have understood each compute node to store this *respective first data* in its memory.

81. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶81)

states:

For example, Barsness teaches that each operator is “executable code configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132.” EX1004, 9:4-9, FIG. 2 (showing operator in memory of compute node). Similarly, Barsness describes a PE as passing data elements on to subsequent PEs. *Id.*, 7:56-58 (“PE1 may ... pass some data elements to PE2 others to PE3”); *see also id.*, 8:14-15 (describing PEs “configured to” transform data before “emitting outgoing data tuples to other PEs, as per the configuration of target PEs.”). Barsness further describes a PE in one compute node emitting data to a next PE on another compute node. *Id.*, 7:58-59 (“Data that flows to PE2 results in tuples emitted to PE4 on compute node 130₂.”); *id.*, 7:60-61 (“data tuples flowing from PE3 to PE5”). The operator graph “maintains the *connectivity configuration* of a plurality of compute nodes and processing elements in a streaming application.” EX1004, 10:49-51 (emphasis added); *id.*, 2:12-15 (operator graph represents the “links between processing elements, corresponding to a flow of data tuples through the stream application”), 3:45-50.

I agree with Dr. Crovella because, in my opinion, Barsness teaches that each operator is “executable code configured to process data flowing into a processing element (e.g., PEs 235) and to emit data to other operators 240 in a PE and to other PEs in the stream application as described by the operator graph 132.” EX1004, 9:4-9, FIG. 2 (showing the operator in the memory of a compute node). Barsness likewise describes a PE passing data elements on to subsequent PEs. *Id.*, 7:56-58 (“PE1 may ... pass some data elements to PE2 others to PE3”); *see also id.*, 8:14-15 (PEs “configured to” transform data before “emitting outgoing data tuples to other PEs, as per the configuration of target PEs”). Barsness further describes a PE in one compute node emitting data to a PE on another compute node. *Id.*, 7:58-59 (“Data

that flows to PE2 results in tuples emitted to PE4 on compute node 130₂"); *id.*, 7:60-61 ("data tuples flowing from PE3 to PE5"). The operator graph "maintains the connectivity configuration of a plurality of compute nodes and processing elements in a streaming application." EX1004, 10:49-51; *id.*, 2:12-15, 3:45-50. A POSITA would have understood the operator graph to identify, for each processing element, the downstream processing elements to which it emits its output.

82. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶82) states:

Barsness describes PEs using TCP/IP sockets to direct output data to the next PE(s): "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)." EX1004, 3:16-18; *id.*, 3:24-28. Barsness provides a specific example where "inter-process communication" between two processing elements "occur[s] using a network socket (e.g., a TCP/IP socket)." EX1004, 7:44-47. POSITAs knew that TCP/IP communications included a port (socket) number and IP address to allow precise communication among devices, as referenced by Barsness. *E.g.*, EX1004, 9:9-12 ("configured to receive data traffic at a[n] incoming port of the PE"), 11:42-45 ("each input and output port of each operators 240 in a processing element (PE)"). POSITAs further knew that this involved each device (here, each PE) storing TCP/IP socket information for its target PE(s), in order to transmit output data to those PE(s).

I agree with Dr. Crovella because, in my opinion, Barsness describes processing elements using TCP/IP sockets to direct their output data to the next processing elements: "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)." EX1004, 3:16-18;

id., 3:24-28. Barsness gives a specific example in which “inter-process communication” between two processing elements “occur[s] using a network socket (e.g., a TCP/IP socket).” EX1004, 7:44-47. A POSITA would have known that TCP/IP communications include a port (socket) number and an IP address to allow precise communication among devices, as Barsness references. *E.g.*, EX1004, 9:9-12 (“configured to receive data traffic at a[n] incoming port of the PE”), 11:42-45 (“each input and output port of each operators 240 in a processing element (PE)”). A POSITA would further have known that this involves each device, here each PE, storing the TCP/IP socket information for its target PEs in order to transmit output data to those PEs.

83. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶83) states:

Bahadur reflects background knowledge regarding TCP/IP. Bahadur describes implementing inter-process communication (IPC) with the Transmission Control Protocol (TCP) operating above the Internet Protocol (IP) network layer protocol, and explains that “[w]hen a TCP connection is established between two applications, TCP/IP provides a number of services for the packet communications between the two applications,” such as “reorder[ing] packets received out of order, automatically transmit[ting] lost packets in a stream, prevent[ing] packet duplication, check[ing] for transmission errors, and implement[ing] flow control procedures.” EX1017, 1:22-42. Bahadur explains that inter-process communication between two applications using a TCP/IP connection involves “specification of a ‘socket’ that comprises the IP address associated with the remote processor and the port number associated with the application running on that processor.” EX1017, 1:43-50. Further, “[w]hen a packet arrives at the TCP layer on a processor, the specific TCP connection is identified by the

combination of a source socket (the source IP address and source port) and the destination socket (the destination IP address and destination port).” EX1017, 1:50-54. Setting up a network socket between two applications using a TCP/IP socket allows transmission of packets between the applications, where a TCP connection is established and maintained between a port bound to one application and a port bound to the other application. EX1017, 1:55-2:26, FIG. 1. Given this background knowledge, POSITAs would have understood from the Barsness disclosure of processing elements communicating with each other “using a network socket (e.g., a TCP/IP socket)” (EX1004, 7:44-47) as disclosure that the compute nodes store such connection information (e.g., IP address and port number) to connect processing elements with each other.

I agree with Dr. Crovella because, in my opinion, Bahadur reflects the background knowledge regarding TCP/IP. Bahadur describes implementing inter-process communication (IPC) with the Transmission Control Protocol (TCP) operating above the Internet Protocol (IP) network-layer protocol, and explains that “[w]hen a TCP connection is established between two applications, TCP/IP provides a number of services for the packet communications between the two applications,” such as “reorder[ing] packets received out of order, automatically transmit[ting] lost packets in a stream, prevent[ing] packet duplication, check[ing] for transmission errors, and implement[ing] flow control procedures.” EX1017, 1:22-42. Bahadur explains that inter-process communication between two applications over a TCP/IP connection involves “specification of a ‘socket’ that comprises the IP address associated with the remote processor and the port number associated with the application running on that processor.” EX1017, 1:43-50. And “[w]hen a packet arrives at the TCP layer on

a processor, the specific TCP connection is identified by the combination of a source socket (the source IP address and source port) and the destination socket (the destination IP address and destination port).” EX1017, 1:50-54. Setting up a network socket between two applications allows packets to be transmitted between them, with a TCP connection established and maintained between a port bound to one application and a port bound to the other. EX1017, 1:55-2:26, FIG. 1. It is my opinion that, given this background knowledge, a POSITA would have understood the Barsness disclosure of processing elements communicating “using a network socket (e.g., a TCP/IP socket)” (EX1004, 7:44-47) to disclose that the compute nodes store such connection information, such as the IP address and port number, to connect the processing elements with one another.

84. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶84) states:

As further confirmation, Santosuosso (EX1016, another IBM patent) with similar disclosure of the operator graph and a management system for processing streaming data expressly teaches each compute node storing stream connection data. EX1016, FIG. 2. Specifically: “The stream connection data 255 represents the connections between PEs on compute node 130 (e.g., a TCP/IP socket connection between two separate PEs 235), as well as connections to other compute nodes 130 with upstream and or downstream PEs in the stream application, also via TCP/IP sockets (or other inter-process data communication mechanisms).” EX1016, 8:56-62. This confirms that a POSITA would have understood that Barsness, through its disclosure of using TCP/IP for communication between processing elements, discloses each compute node storing TCP/IP socket connection information to enable

communication between PEs (including between PEs in different compute nodes).

I agree with Dr. Crovella because, in my opinion, Santosuosso (EX1016), another IBM patent with a similar disclosure of the operator graph and a management system for processing streaming data, further confirms this by expressly teaching each compute node storing stream connection data. EX1016, FIG. 2. Specifically, “[t]he stream connection data 255 represents the connections between PEs on compute node 130 (e.g., a TCP/IP socket connection between two separate PEs 235), as well as connections to other compute nodes 130 with upstream and or downstream PEs in the stream application, also via TCP/IP sockets (or other inter-process data communication mechanisms).” EX1016, 8:56-62. It is my opinion that this confirms that a POSITA would have understood Barsness, through its disclosure of using TCP/IP for communication between processing elements, to disclose each compute node storing TCP/IP socket connection information to enable communication between PEs, including PEs in different compute nodes.

85. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶85) states:

Thus, Barsness discloses the application data stored in the first compute node 130₁ (*the respective first data stored in the memory of the first computer system*) includes an identification of the specific “incoming port” for downstream PE(s) (*information about the second transformation pipeline*), and the first compute node 130₁ (*first computer system*) processes this application data to obtain the port

number for routing output data to the downstream PE(s). EX1004, 9:9-12.

I agree with Dr. Crovella because, in my opinion, Barsness discloses that the application data stored in the first compute node 130₁, the *respective first data stored in the memory of the first computer system*, includes an identification of the specific “incoming port” for the downstream processing elements, which is *information about the second transformation pipeline*, and that the first compute node 130₁, the *first computer system*, processes this application data to obtain the port number for routing its output data to the downstream processing elements. EX1004, 9:9-12. A POSITA would have understood the first compute node to *process the respective first data stored in the memory of the first computer system to determine information about the second transformation pipeline*.

86. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶86) states:

For example, in the example operator graph disclosed by Barsness, the processing elements PE2 and PE3 of the first compute element 130₁ / first transformation pipeline emit output data to processing elements PE4 and PE5 of the second compute element 130₂ / second transformation pipeline:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Id., 7:52-67 (describing the flow of data tuples through the nodes and processing elements of the example operator graph). In order for the processing elements PE2 and PE3 of the first transformation pipeline to direct the output data to the processing elements PE4 and PE5 of the second transformation pipeline, the respective operator of each processing element PE2 and PE3 stored in the memory of the first

computer system, includes information identifying the processing elements PE4 and PE5 as a destination (*information about the second transformation pipeline*). For example, using TCP/IP, first compute node 130₁ stores socket information including the “source socket (the source IP address and source port) and the destination socket (the destination IP address and destination port)” (EX1017, 1:43-54), which would include the destination socket information for the downstream PEs PE4 and PE5.

I agree with Dr. Crovella because, in my opinion, in the example operator graph of Barsness the processing elements PE2 and PE3 of the first compute node 130₁, the *first transformation pipeline*, emit their output data to processing elements PE4 and PE5 of the second compute node 130₂, the *second transformation pipeline*, as FIG. 1B shows.

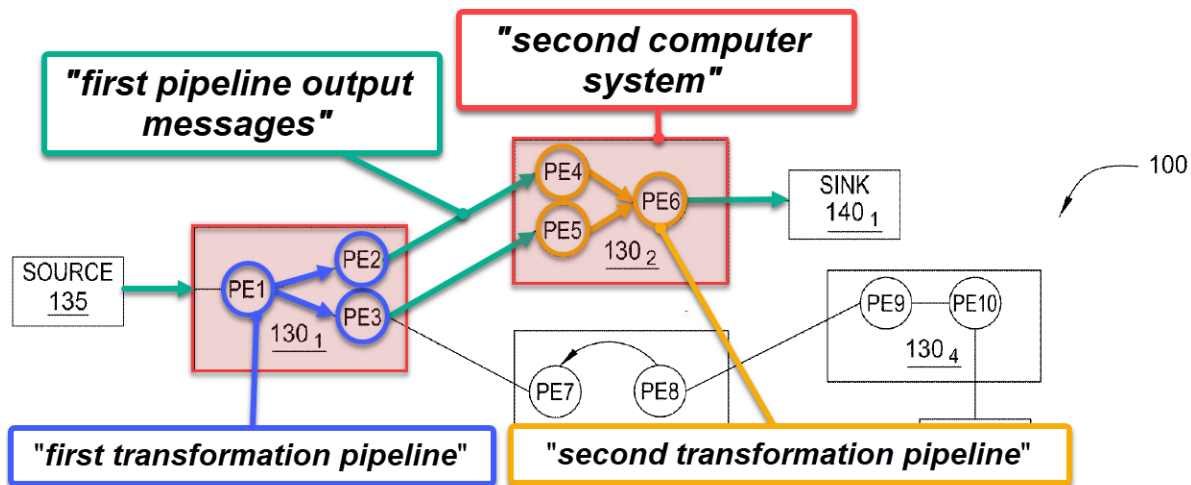


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 7:52-67. It is my opinion that, for PE2 and PE3 of the *first transformation pipeline* to direct their output data to PE4 and PE5 of the *second transformation pipeline*, the respective operator of each of PE2 and PE3 stored in the memory of the *first computer system* includes information

identifying PE4 and PE5 as a destination, which is *information about the second transformation pipeline*. Using TCP/IP, for example, the first compute node 130₁ stores socket information including the “source socket (the source IP address and source port) and the destination socket (the destination IP address and destination port),” EX1017, 1:43-54, which would include the destination socket information for the downstream processing elements PE4 and PE5. A POSITA would have understood the first computer system to store this *information about the second transformation pipeline*.

87. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶87) states:

The first compute node thus executes the operator of a given processing element at the end of the first transformation pipeline (e.g., PEs 2 and 3 in example operator graph of Barsness FIG. 1B) (*processes the respective first data stored in the memory of the first computer system*) to determine the destination of its output tuples (*to determine information about the second transformation pipeline*).

I agree with Dr. Crovella because, in my opinion, the first compute node executes the operator of a processing element at the end of the *first transformation pipeline*, for example PE2 and PE3 in the example operator graph of Barsness FIG. 1B, thereby *processing the respective first data stored in the memory of the first computer system* to determine the destination of its output tuples, that is, *to determine information about the second transformation pipeline*. A POSITA would have understood the first computer system to *process the respective first data*

stored in the memory of the first computer system to determine information about the second transformation pipeline.

88. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶88) states:

A POSITA would thus understand that the first PE is configured with (and therefore knows, at least logically) which downstream PE(s)/port(s) are the targets for its emitted data. At the very least a POSITA implementing Barsness would have been motivated to configure the operator / executable code of each PE with information identifying the downstream PE(s) to which the given operator should emit its output data, such that the PE emitted its transformed data to the appropriate subsequent PE(s) in the stream application. A POSITA would have been motivated to implement the PEs in such a way so that they could emit the data to destination(s) without having to consult the operator graph stored separately on the management system. Having this information on each PE would allow for rapid transmission of outputs to the next downstream node.

I agree with Dr. Crovella because, in my opinion, a POSITA would have understood that the first processing element is configured with, and therefore knows, at least logically, which downstream processing elements and ports are the targets for its emitted data. At the very least, a POSITA implementing Barsness would have been motivated to configure the operator, or executable code, of each processing element with information identifying the downstream processing elements to which that operator should emit its output data, so that the processing element emits its transformed data to the appropriate subsequent processing elements in the stream application. A POSITA would have been motivated to implement the processing

elements in this way so that they could emit their data to the correct destinations without having to consult the operator graph stored separately on the management system, because keeping this information on each processing element would allow for rapid transmission of outputs to the next downstream node.

10. **[1.3.d] - and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined information,**

89. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶89)

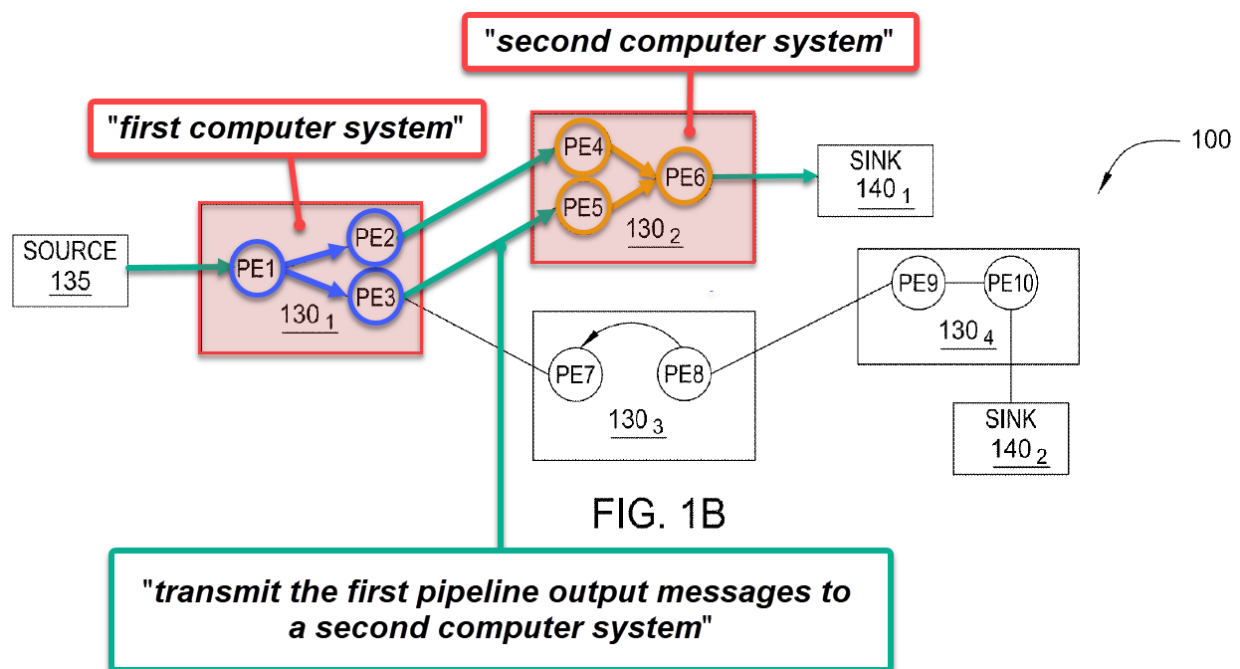
states:

Barsness satisfies this element. Barsness discloses a given computing node transmitting outgoing data to a second computing system of the first plurality of computer systems per the configuration represented by the operator graph. EX1004, 8:11-15. In the example provided by Barsness, the processing elements PE2 and PE3 in the first compute node 130₁ emit/transmit data (*transmit the first pipeline output messages*) to PEs 4 and 5 of the second compute system 130₂ (*a second computer system of the first plurality of computer systems*):

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Supra paragraphs 64-68. In FIG. 1B, this transmission is reflected in each of the edges between PE2-PE4 and PE3-PE5. As discussed above in paragraphs 80-88, the operators of Barsness identify the destination of their output data, and thus the first compute node (*first computer system*) transmits the output of the first compute node (*the first pipeline output messages*) to the second computer system according to the destination(s) specified in the operator (*the determined information*). EX1004, 3:16-18 (inter-PE communication via TCP/IP socket), 7:44-47.

I agree with Dr. Crovella because, in my opinion, Barsness discloses a given computing node that transmits its outgoing data to a *second computer system of the first plurality of computer systems* in accordance with the configuration represented by the operator graph. EX1004, 8:11-15. In the Barsness example, processing elements PE2 and PE3 in the first compute node 130₁ emit, or *transmit*, their data, that is, they *transmit the first pipeline output messages* to PE4 and PE5 of the second compute node 130₂, which is a *second computer system of the first plurality of computer systems*.



EX1004, FIG. 1B (annotations added); *see also* EX1004, 7:58-59 ("Data that flows to PE2 results in tuples emitted to PE4 on compute node 130₂"), 7:64-67. In FIG. 1B, this transmission is reflected in the edges between PE2 and PE4 and between PE3 and PE5. As I discussed above, the operators of Barsness identify the

destination of their output data, so the first compute node, the first computer system, transmits its output, the *first pipeline output messages*, to the *second computer system* according to the destinations specified in the operator, which is the *determined information*. EX1004, 3:16-18 (inter-PE communication via TCP/IP socket), 7:44-47. A POSITA would have understood the first computer system to *transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined information*.

11. [1.4] - wherein the second computer system is configured to:

90. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶90) states:

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

(Figure omitted. EX1004, FIG. 1B (highlighting added))

The following sections explain why Barsness teaches configuring this compute node in the manner recited by claim 1. The Barsness third compute node 130₃ is an alternative “second computer system” with a real-time transformation pipeline of PE7 and PE8 that transmits output messages to PE9, for the same reasons explained below for compute node 130₂. For example, the pipeline comprising the processing elements PE7 and PE8 may alternatively comprise the second transformation pipeline because they are downstream from the processing elements of the first compute node 130₁ comprising the first transformation pipeline and process the output of that pipeline (i.e., the output of PE3, first pipeline output messages). Further, the processing element PE8 of the third compute node 130₃ transmits its output (pipeline output messages) to PE9 of the fourth compute node 130₄.

I agree with Dr. Crovella because, in my opinion, Barsness discloses a second compute node 130₂, which is a *second computer system*, as FIG. 1B shows.

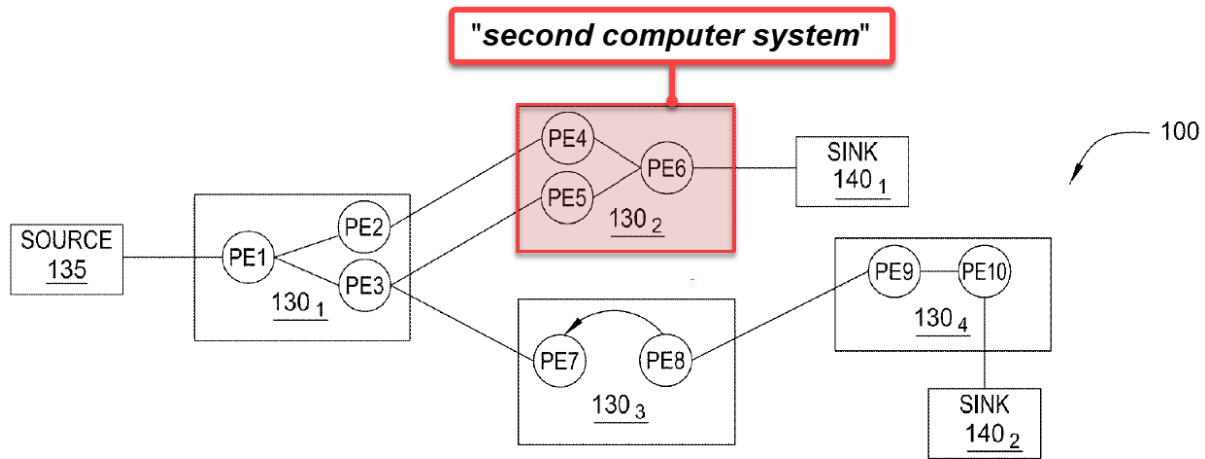


FIG. 1B

EX1004, FIG. 1B (annotations added). The following sections explain why Barsness teaches configuring this compute node in the manner claim 1 recites. The Barsness third compute node 130₃ is an alternative *second computer system*, with a real-time transformation pipeline of PE7 and PE8 that transmits output messages to PE9, for the same reasons explained below for compute node 130₂. For example, the pipeline comprising processing elements PE7 and PE8 may alternatively be the second transformation pipeline, because they are downstream from the processing elements of the first compute node 130₁ that comprise the first transformation pipeline and process the output of that pipeline, that is, the output of PE3, the first pipeline output messages. Further, processing element PE8 of the third compute node 130₃ transmits its output, the pipeline output messages, to PE9 of the fourth compute node 130₄. A

POSITA would have understood Barsness to disclose a *second computer system*, including the alternative provided by the third compute node.

12. [1.4.a] - receive the first pipeline output messages,

91. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶91)

states:

Barsness satisfies this element. Barsness discloses the second compute node 130₂ (*the second computer system*) receiving the data emitted by PEs of the first compute node 130₁ (*the first pipeline output messages*). EX1004, 7:58-61, FIG. 1B. For example, as discussed above in paragraphs 64-68, the operator graph of Barsness shows a flow of output data (*the first pipeline output messages*) from the first transformation pipeline to inputs (processing elements PE4 and PE5) of the second transformation pipeline in the second compute node 130₂ depicted in FIG. 1B:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Barsness further describes “[d]ata that flows to PE2 results in tuples emitted to PE4 on compute node 130₂,” and similarly “data tuples flowing from PE3 to PE5,” where PE5 is also on compute node 130₂. EX1004, 7:58-61. Barsness describes the processing elements transmitting the messages “using a network socket (e.g., a TCP/IP socket).” EX1004, 7:44-47.

I agree with Dr. Crovella because, in my opinion, Barsness discloses the second compute node 130₂, the *second computer system*, receiving the data emitted by the PEs of the first compute node 130₁, that is, the *first pipeline output messages*. EX1004, 7:58-61, FIG. 1B. As I discussed above, the operator graph of Barsness shows a flow of output data, the *first pipeline output messages*, from the first

transformation pipeline to the inputs, processing elements PE4 and PE5, of the second transformation pipeline in the second compute node 130₂, as FIG. 1B depicts.

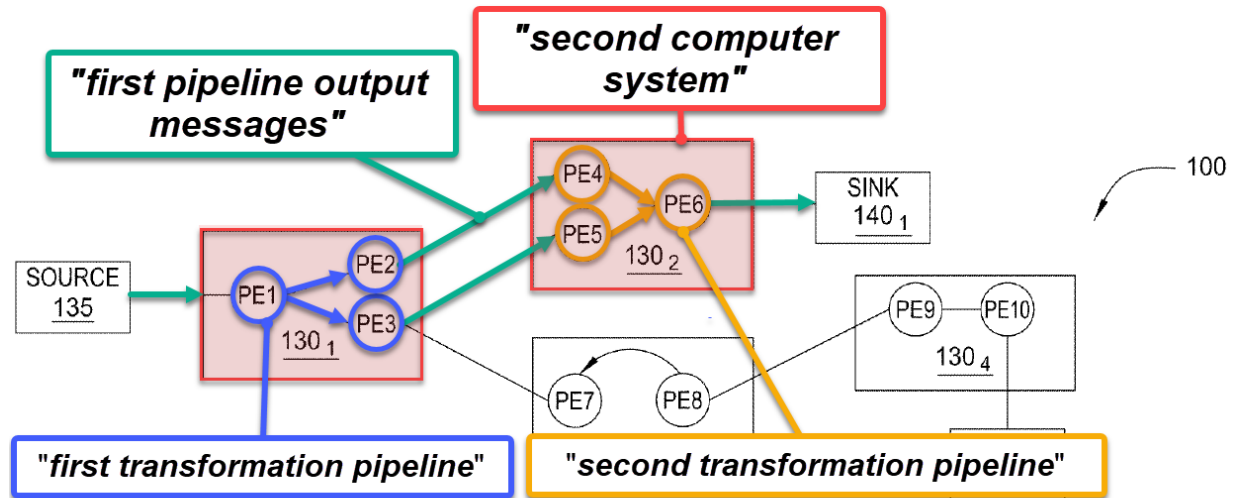


FIG. 1B

EX1004, FIG. 1B (annotations added). Barsness further describes “[d]ata that flows to PE2 results in tuples emitted to PE4 on compute node 130₂,” and “data tuples flowing from PE3 to PE5,” where PE5 is also on compute node 130₂. EX1004, 7:58-61. Barsness describes the processing elements transmitting these messages “using a network socket (e.g., a TCP/IP socket).” EX1004, 7:44-47. A POSITA would have understood the second computer system to *receive the first pipeline output messages*.

13. [1.4.b] - and process the first pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the first

**pipeline output messages to generate second pipeline output
messages,**

92. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶92)

states:

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

(Figure omitted. EX1004, FIG. 1B (highlighting added))

See also id., 7:52-67. Barsness further teaches configuring sink PEs to send data to other applications (*generate second pipeline output messages*). *Id.*, 8:4-6 (“Sinks 140 may also be configured to send data tuples to other applications in the computing infrastructure 100.”). POSITAs knew how to output data from sink PEs using edge adaptation, as I discussed above in the state of the art. *See* paragraph 35. Further, a POSITA would have been motivated with a reasonable expectation of success to implement a given sink PE to generate output messages to other applications that used the data at least because Barsness teaches doing so. Thus, in the FIG. 1B example, PE6 (and thus the *second transformation pipeline*) generates output data that are *second pipeline output messages* because PE6 is the last PE in the second transformation pipeline, and so its output messages are output messages of the second transformation pipeline. And as I mentioned above, compute node 130₃, an alternative “second computing system” with transformation pipeline PE7 and PE8, also generates output messages to PE9 of compute node 130₄.

I agree with Dr. Crovella because, in my opinion, in FIG. 1B the second compute node 130₂, the *second computer system*, processes the output data from the first compute node 130₁, the *first pipeline output messages*, by *executing software*

instructions, the operators or executable code, associated with processing elements PE4, PE5, and PE6, the *second transformation pipeline*, to generate output data, the *second pipeline output messages*.

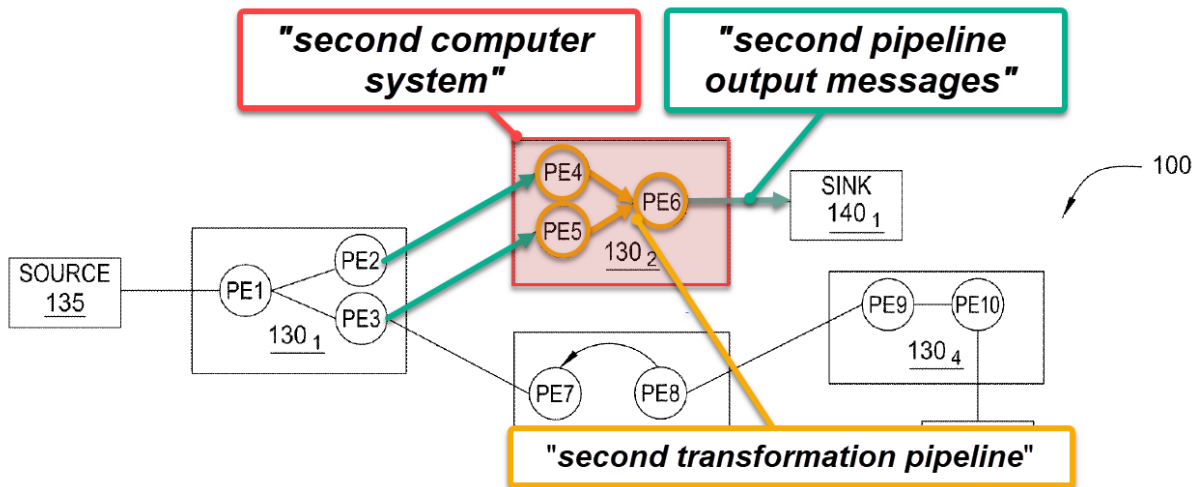


FIG. 1B

EX1004, FIG. 1B (annotations added); *see also id.*, 7:52-67. Barsness further teaches configuring sink PEs to send data to other applications, thereby generating *second pipeline output messages*. *Id.*, 8:4-6 ("Sinks 140 may also be configured to send data tuples to other applications in the computing infrastructure 100."). A POSITA would have known how to output data from sink PEs using edge adaptation, as I discussed above in the state of the art. A POSITA would further have been motivated, with a reasonable expectation of success, to implement a given sink processing element to generate output messages to other applications that used the data, at least because Barsness teaches doing so. In the FIG. 1B example, PE6, and thus the *second transformation pipeline*, generates output data that are *second*

pipeline output messages, because PE6 is the last processing element in the *second transformation pipeline*, so its output messages are the output messages of that pipeline. And as I noted above, compute node 130₃, an alternative second computer system with the transformation pipeline of PE7 and PE8, also generates output messages to PE9 of compute node 130₄. A POSITA would have understood the second computer system to *execute software instructions that apply the second transformation pipeline to the first pipeline output messages to generate second pipeline output messages*.

93. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶93) states:

The PEs within a given compute node apply a transformation pipeline by processing the stream of input data. EX1004, 8:11-15 ("In one or more embodiments, 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, as per the configuration of target PEs"). This occurs in real time, as explained with respect to the PEs of compute node 130₁ (*first transformation pipeline*) and Element 1.3.b. See paragraphs 74-79.

I agree with Dr. Crovella because, in my opinion, the PEs within a given compute node apply a transformation pipeline by processing the stream of input data. EX1004, 8:11-15 ("In one or more embodiments, 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, as per the configuration of target PEs"). This processing occurs *substantially in real time*, as I explained with

respect to the PEs of compute node 130₁, the first transformation pipeline, and element 1.3.b. A POSITA would have understood the second computer system to *process the first pipeline output messages substantially in real time.*

14. [1.5] - wherein the first and second computer systems are **distinct; and**

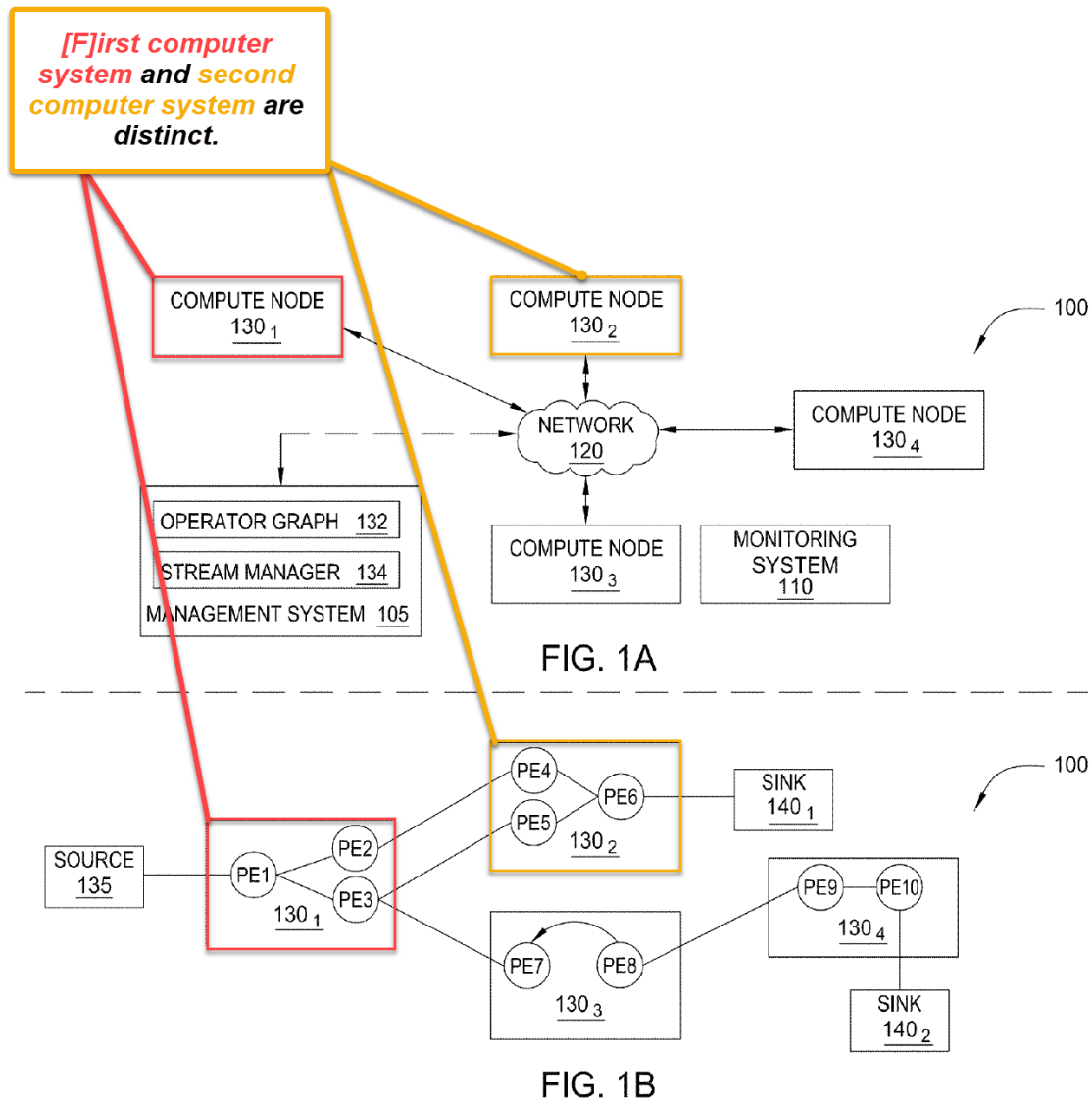
94. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶94) states:

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

(Figure omitted. EX1004, FIG. 1B (highlighting added))

When implemented as a virtual machine, each compute node “runs as an independent computer system.” EX1004, 8:43-44.

I agree with Dr. Crovella because, in my opinion, the compute nodes of Barsness, including the first compute node 130₁ and the second compute node 130₂, are shown as *distinct* computing systems, as is the third compute node 130₃.



EX1004, FIG. 1A, 1B (annotations added); *see also id.*, 7:52-67, 8:4-6. When implemented as a virtual machine, each compute node “runs as an independent computer system.” EX1004, 8:43-44; FIG. 1B. A POSITA would have understood the *first and second computer systems* of Barsness to be *distinct*.

95. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶95) states:

Barsness further teaches moving PEs to “separate machines,” further confirming that the compute nodes are distinct machines. EX1004, 3:56-60 (“In response, the monitoring application could then increase the priority of the PE in question or even move the PE to separate machines to gain more resource, assuming that the user responds favorably to prompts suggesting such changes to increase performance.”).

I agree with Dr. Crovella because, in my opinion, Barsness further teaches moving PEs to “separate machines,” which confirms that the compute nodes are *distinct* machines. EX1004, 3:56-60 (“In response, the monitoring application could then increase the priority of the PE in question or even move the PE to separate machines to gain more resource, assuming that the user responds favorably to prompts suggesting such changes to increase performance.”). A POSITA would have understood this teaching to confirm that the *first and second computer systems* are *distinct*.

15. [1.6] - a second plurality of computer systems;

96. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶96) states:

Barsness satisfies this element. Barsness discloses automatically adding additional compute nodes to the compute nodes 130₁₋₄ to improve the processing of the stream application. EX1004, 14:44-56; *id.*, 14:26-39, 13:12-14, FIG. 6. POSITAs would thus understand that the system includes additional, unconfigured compute nodes available for use. Such available compute nodes, on their own or together with one or more of the compute nodes 130₁₋₄ executing the stream application, are *a second plurality of computer systems*. The “second plurality of computer systems” is arbitrary, such that any group of two or more compute nodes satisfies it, including combinations of available /

duplicated compute nodes and one or more of the compute nodes 130₁₋₄, multiple duplicated compute nodes, a duplicated compute node and an unprovisioned / available compute node, and so on.

I agree with Dr. Crovella because, in my opinion, Barsness discloses automatically adding additional compute nodes to the compute nodes 130₁₋₄ to improve the processing of the stream application. EX1004, 14:44-56; *id.*, 14:26-39, 13:12-14, FIG. 6. A POSITA would thus have understood that the system includes additional, unconfigured compute nodes available for use. Those available compute nodes, on their own or together with one or more of the compute nodes 130₁₋₄ executing the stream application, are a *second plurality of computer systems*. Under the claims and '424 patent's specification, the *second plurality of computer systems*, so that any group of two or more compute nodes satisfies it, including combinations of available or duplicated compute nodes with one or more of the compute nodes 130₁₋₄, multiple duplicated compute nodes, a duplicated compute node together with an unprovisioned or available compute node, and so on.

97. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶97) states:

Barsness discloses implementing its stream data processing application using a "cloud computing infrastructure," where "the nodes used to create a stream application may be virtual machines hosted by a cloud service provider." EX1004, 6:16-42. This infrastructure includes a "shared pool of configurable computing resources that can be rapidly provisioned and released." EX1004, 6:23-27. The infrastructure thus provides virtual computing resources, including "complete virtualized computing systems." EX1004, 6:26-31; *id.*, 6:32-40. Thus, as Barsness

teaches that certain “virtual machines hosted by a cloud service provider” would serve as the compute nodes (EX1004, 6:40-42), POSITAs would understand that other hosted virtual machines remained available for use, even if not currently executing one or more processing elements. Two or more of these virtual machines that are available but not currently assigned to execute make up a *second plurality of computer systems*. EX1004, 6:24-26. Similarly, one or more of such virtual machines together with one or more of the in-use virtual machine (e.g., compute nodes 130₁₋₄ executing the processing elements of the stream application) would also make up a *second plurality of computer systems*.

I agree with Dr. Crovella because, in my opinion, Barsness discloses implementing its stream data processing application using a “cloud computing infrastructure,” in which “the nodes used to create a stream application may be virtual machines hosted by a cloud service provider.” EX1004, 6:16-42. This infrastructure includes a “shared pool of configurable computing resources that can be rapidly provisioned and released.” EX1004, 6:23-27. The infrastructure thus provides virtual computing resources, including “complete virtualized computing systems.” EX1004, 6:26-31; *id.*, 6:32-40. Because Barsness teaches that certain “virtual machines hosted by a cloud service provider” serve as the compute nodes, EX1004, 6:40-42, a POSITA would have understood that other hosted virtual machines remained available for use, even if not currently executing one or more processing elements. Two or more of these virtual machines that are available but not currently assigned to execute make up a *second plurality of computer systems*. EX1004, 6:24-26. Likewise, one or more of those virtual machines together with one or more of the in-use virtual

machines, such as compute nodes 130₁₋₄ executing the processing elements of the stream application, would also make up a *second plurality of computer systems*.

98. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶98) states:

Specifically, Barsness discloses that each compute node "may be a physical host or a virtual machine," where "a physical host may be configured to host a plurality of virtual machines, which may share hardware resources of the physical host" and where each "virtual machine runs as an *independent computer system*." EX1004, 8:37-44 (emphasis added). Barsness further discloses that the "processing nodes and the management server may be hosted in a virtualized computing environment to provide well known advantages of virtualized computing infrastructure including another *layer of scalability* and fault tolerance." EX1004, 8:48-52 (emphasis added). Through this "scalability and fault tolerance" of the virtualized computing infrastructure, the plurality of compute nodes taught by Barsness thus includes as many "independent computer system[s]" as needed for executing stream applications, including a *second plurality of computer systems*. EX1004, 8:43-44, 8:48-52. For example, Barsness discloses "spawn[ing] more process nodes 130" as needed. EX1004, 10:28-34; *id.*, 14:51-56 ("spawning new compute nodes" and "duplicating a compute node"), 8:45-48 ("spawn more virtual machines on the fly").

I agree with Dr. Crovella because, in my opinion, Barsness discloses that each compute node "may be a physical host or a virtual machine," where "a physical host may be configured to host a plurality of virtual machines, which may share hardware resources of the physical host," and where each "virtual machine runs as an independent computer system." EX1004, 8:37-44. Barsness further discloses that the "processing nodes and the management server may be hosted in a virtualized computing environment to provide well known advantages of virtualized computing

infrastructure including another layer of scalability and fault tolerance.” EX1004, 8:48-52. It is my opinion that, through this “scalability and fault tolerance” of the virtualized computing infrastructure, the plurality of compute nodes taught by Barsness includes as many “independent computer system[s]” as are needed for executing stream applications, including a *second plurality of computer systems*. EX1004, 8:43-44, 8:48-52. For example, Barsness discloses “spawn[ing] more process nodes 130” as needed. EX1004, 10:28-34; *id.*, 14:51-56 (“spawning new compute nodes” and “duplicating a compute node”), 8:45-48 (“spawn more virtual machines on the fly”). A POSITA would have understood Barsness to disclose a *second plurality of computer systems*.

99. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶99) states:

While I am referring to the virtualized cloud-computing embodiment disclosed by Barsness, Barsness also discloses that each compute node 130 may be a “physical host,” so the teachings in Barsness regarding adding compute nodes 130 apply equally for a physical embodiment. EX1004, 8:30-39.

I agree with Dr. Crovella because, in my opinion, although I am referring to the virtualized cloud-computing embodiment disclosed by Barsness, Barsness also discloses that each compute node 130 may be a “physical host,” so its teachings regarding adding compute nodes 130 apply equally to a physical embodiment. EX1004, 8:30-39. A POSITA would have understood the Barsness teachings

regarding a *second plurality of computer systems* to apply equally to the physical-host embodiment.

16. **Element [1.7] - wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause a fourth computer system of the second plurality of computer systems to execute software instructions that apply at least one of the first transformation pipeline and the second transformation pipeline.**

100. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶100)

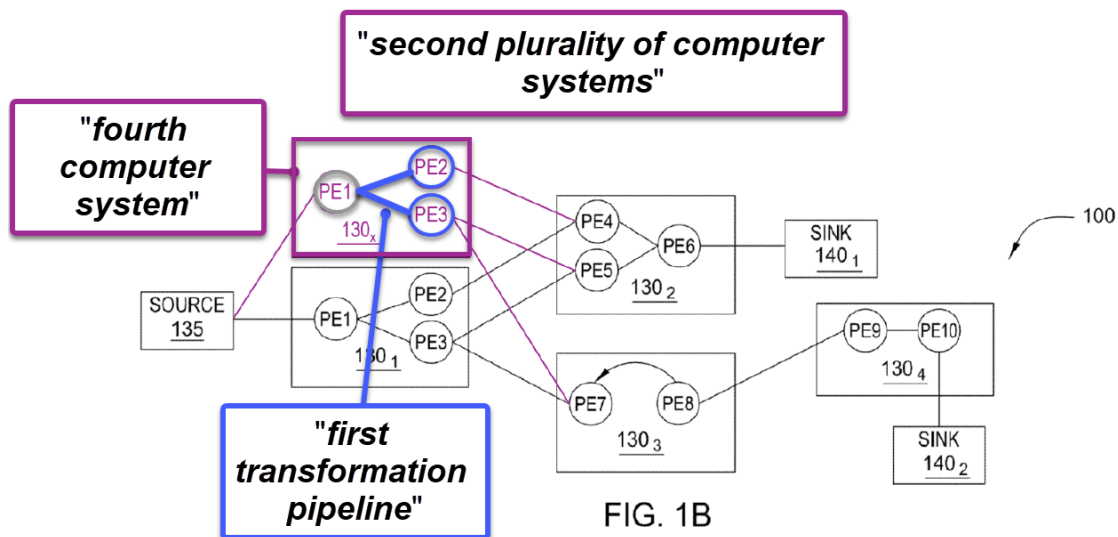
states:

Barsness satisfies this element. As explained in the following paragraphs, Barsness discloses the management system 105 (*third computer system*) configured to duplicate or migrate an existing compute node to an additional compute node (*fourth computer system*), thereby causing that additional compute node to apply the transformation pipeline of the duplicated or migrated compute node (*first / second transformation pipeline*). For example, with reference to FIG. 1B, Barsness teaches a management system configured to address performance issues detected in compute node 130₁ by deploying an additional compute node 130_x (*fourth computer system of the second plurality of computer systems*) that executes software instructions applying the same PEs of compute node 130₁ (*first transformation pipeline*):

(Figure omitted. EX1004, FIG. 1B (purple annotations added))

I agree with Dr. Crovella because, in my opinion, and as explained in the following paragraphs, Barsness discloses the management system 105, the *third computer*

system, configured to duplicate or migrate an existing compute node to an additional compute node, the **fourth computer system**, thereby causing that additional compute node to apply the transformation pipeline of the duplicated or migrated compute node, the **first** or **second transformation pipeline**. For example, with reference to FIG. 1B, Barsness teaches a management system configured to address performance issues detected in compute node 130₁ by deploying an additional compute node 130_x, a **fourth computer system of the second plurality of computer systems**, that executes software instructions applying the same PEs as compute node 130₁, the **first transformation pipeline**.



EX1004, FIG. 1B (annotations added). A POSITA would have understood Barsness to disclose a **third computer system** configured to cause a **fourth computer system** to apply the **first** or **second transformation pipeline**.

101. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶101)

states:

Specifically, Barsness discloses its management system being configured to deploy additional computer systems to increase performance. For example, the management system will provision or “spawn more process nodes 130 to process certain type of data [to address] a detection of performance issues with a computer node 130.” EX1004, 10:28-34. And it will “move processing elements 235 from one computer node 130 to another to improve the data processing performance of the stream application.” *Id.*, 10:34-39. Relatedly, Barsness teaches increasing data throughput by using processing elements “in parallel to process [sic] same type of data.” *Id.*, 10:26-28. Barsness provides further description of how the management system duplicates or moves compute nodes to maintain expected performance levels. For example, if the “processing rate is above or below” a pre-determined level, the system takes “corrective actions” that include “spawning new compute nodes” and “duplicating a compute node.” 14:30-57; *see also id.*, FIG. 6, claim 2 (“adjusting the computing resources”), claim 5 (“cloning one of the processing elements” based on “processing state”).

I agree with Dr. Crovella because, in my opinion, Barsness discloses its management system being configured to deploy additional computer systems to increase performance. For example, the management system will provision or “spawn more process nodes 130 to process certain type of data [to address] a detection of performance issues with a computer node 130.” EX1004, 10:28-34. And it will “move processing elements 235 from one computer node 130 to another to improve the data processing performance of the stream application.” *Id.*, 10:34-39. Barsness also teaches increasing data throughput by using processing elements “in parallel to process [sic] same type of data.” *Id.*, 10:26-28. Barsness further describes how the

management system duplicates or moves compute nodes to maintain expected performance levels: if the “processing rate is above or below” a pre-determined level, the system takes “corrective actions” that include “spawning new compute nodes” and “duplicating a compute node.” EX1004, 14:30-57; *see also id.*, FIG. 6, claim 2 (“adjusting the computing resources”), claim 5 (“cloning one of the processing elements” based on “processing state”). A POSITA would have understood the management system to *execute software instructions* that deploy additional computer systems.

102. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶102) states:

The management system 105 runs a node manager 340, as depicted in FIG. 3 of Barsness, “to configure the compute nodes,” for example as part of “assigning or reassigning” processing elements “on different compute nodes.” EX1004, 10:14-18; *id.*, 10:18-28 (describing the node manager “configuring each processing element and operator”); *see also id.*, 9:60-11:10 (describing management system CPU “configured to retrieve and execute programming instructions stored in the memory”). Regarding the virtual machine implementation of its system, Barsness describes the ability to “*spawn more virtual machines on the fly*” and thus provide “well known advantages” “including another layer of scalability.” EX1004, 8:43-52 (emphasis added); *id.*, 7:45-48.

I agree with Dr. Crovella because, in my opinion, the management system 105 runs a node manager 340, as FIG. 3 of Barsness depicts, “to configure the compute nodes,” for example as part of “assigning or reassigning” processing elements “on different compute nodes.” EX1004, 10:14-18; *id.*, 10:18-28 (the node manager

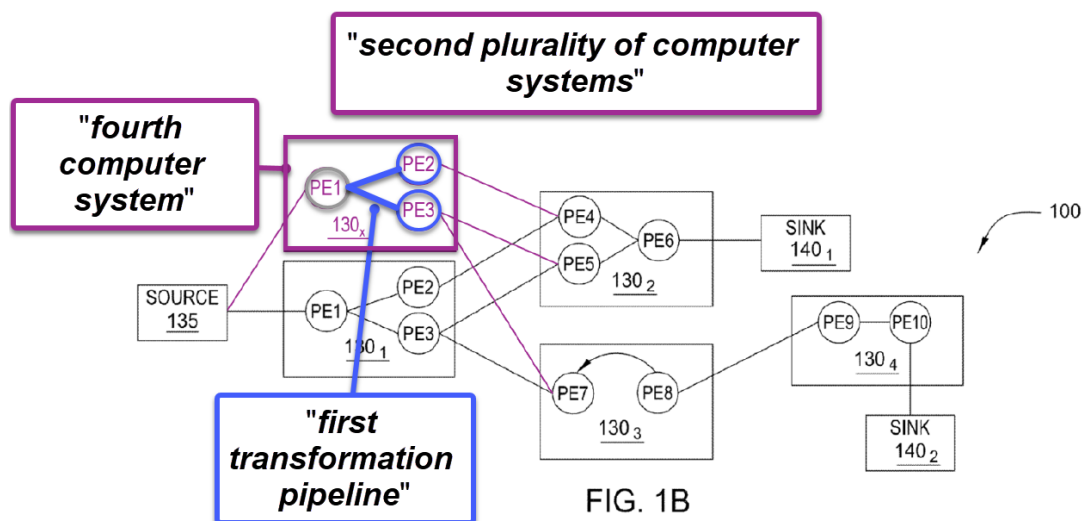
“configuring each processing element and operator”); *see also id.*, 9:60-11:10 (the management system CPU “configured to retrieve and execute programming instructions stored in the memory”). As to the virtual-machine implementation, Barsness describes the ability to “spawn more virtual machines on the fly” and thereby provide “well known advantages” “including another layer of scalability.” EX1004, 8:43-52; *id.*, 7:45-48. A POSITA would have understood the management system to *execute software instructions* that configure and deploy compute nodes through the node manager.

103. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶103) states:

Applying these teachings in the context of the computing infrastructure shown in FIG. 1B, the management system is configured to provision a new compute node 130_x (*fourth computer system*) with processing elements PE1, PE2, and PE3 (*first transformation pipeline*) to address any “performance issues” that arise with compute node 130₁. EX1004, 10:30-34; *see also id.*, 10:34-39 (describing movement of “processing elements” among compute nodes “to improve the data processing performance of the stream application”). By processing the data in “parallel ... to increase data throughput” (EX1004, 10:26-28), this “duplicating a compute node” helps ensure an appropriate “processing rate” for the stream application (EX1004, 14:54-56, 14:30-39). The following annotated FIG. 1B shows duplicated / provisioned compute node 130_x (*fourth computer system*) configured with the processing elements of the first compute node 130₁ (*first transformation pipeline*), as taught by Barsness:

(Figure omitted. EX1004, FIG. 1B (purple annotations added))

I agree with Dr. Crovella because, in my opinion, applying these teachings to the computing infrastructure shown in FIG. 1B, the management system is configured to provision a new compute node 130_x, a *fourth computer system*, with processing elements PE1, PE2, and PE3, the *first transformation pipeline*, to address any “performance issues” that arise with compute node 130₁. EX1004, 10:30-34; *see also id.*, 10:34-39 (movement of “processing elements” among compute nodes “to improve the data processing performance of the stream application”). By processing the data in “parallel ... to increase data throughput,” EX1004, 10:26-28, this “duplicating a compute node” helps ensure an appropriate “processing rate” for the stream application. EX1004, 14:54-56, 14:30-39. As the annotated FIG. 1B depicts, the duplicated or provisioned compute node 130_x, the *fourth computer system*, is configured with the processing elements of the first compute node 130₁, the *first transformation pipeline*.



EX1004, FIG. 1B (annotations added). A POSITA would have understood the management system to cause the *fourth computer system* to apply the *first transformation pipeline*.

104. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶104) states:

Similar detection of performance issues for compute node 130₂ would lead the management system to duplicate that node, with duplicated / provisioned compute node 130_x (*a fourth computer system*) configured with the processing elements of the second compute node 130₂ (*the second transformation pipeline*). For example, if the monitoring system detects performance issues with compute node 130₂, the management system 105 provisions a new compute node 130_x (*fourth computer system*) with processing elements PE4, PE5, and PE6 (*second transformation pipeline*) migrated / duplicated thereto "to improve the data processing performance of the stream application." EX1004, 10:34-39; *id.*, 10:28-34, 14:51-56 (describing automatically "duplicating a compute node"). Such a duplicated / provisioned compute node 130_x (*a fourth computer system*) configured with the processing elements of the second compute node 130₂ (*the second transformation pipeline*) is shown in this annotated FIG. 1B:

(Figure omitted. EX1004, FIG. 1B (purple annotations added))

I agree with Dr. Crovella because, in my opinion, a similar detection of performance issues for compute node 130₂ would lead the management system to duplicate that node, with the duplicated or provisioned compute node 130_x, a *fourth computer system*, configured with the processing elements of the second compute node 130₂, the *second transformation pipeline*. For example, if the monitoring system detects performance issues with compute node 130₂, the management system 105 provisions

a new compute node 130_x, a *fourth computer system*, with processing elements PE4, PE5, and PE6, the *second transformation pipeline*, migrated or duplicated to it “to improve the data processing performance of the stream application.” EX1004, 10:34-39; *id.*, 10:28-34, 14:51-56 (automatically “duplicating a compute node”). As the annotated FIG. 1B shows, that duplicated or provisioned compute node 130_x is configured with the processing elements of the second compute node 130₂, the *second transformation pipeline*.

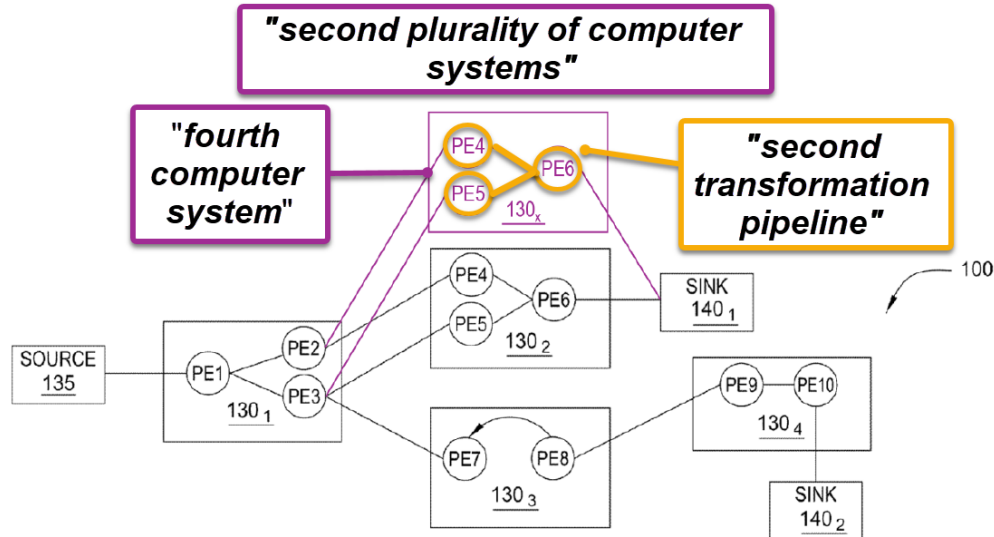


FIG. 1B

EX1004, FIG. 1B (annotations added). A POSITA would have understood the management system to cause the *fourth computer system* to apply the *second transformation pipeline*.

105. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶105) states:

Further, while Barsness discloses “spawning new compute nodes” and “duplicating a compute node” generally (EX1004, 14:51-56), a POSITA would have found it obvious to duplicate the compute nodes 130₁ and 130₂ in particular because Barsness discloses its management system being configured to duplicate any compute node in the context of corrective measures for improving data processing throughput. EX1004, 10:26-39, 14:26-56. My analysis for claims 3 and 5 (below in Sections IX.C and IX.D) provides further examples of how the management system duplicates / spawns compute nodes to process data streams with the stream application.

I agree with Dr. Crovella because, in my opinion, although 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 in particular, because Barsness discloses its management system being configured to duplicate any compute node as part of corrective measures for improving data-processing throughput. EX1004, 10:26-39, 14:26-56. My analysis for claims 3 and 5, below in Sections VII.D and VII.F, provides further examples of how the management system duplicates or spawns compute nodes to process data streams with the stream application.

106. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶106-107) states:

Each of these newly spawned, duplicate compute nodes executes software instructions that apply at least one of the first transformation pipeline and the second transformation pipeline, as carried out by the respective processing elements. *See* paragraphs 74-79. Each duplicate compute node is of a second plurality of computer systems, for example a second plurality comprising the duplicate compute node along with the compute nodes of the first plurality of computer systems. As I noted

above, the recited “second plurality of computer systems” in claim 1 is arbitrary because nothing says this second plurality is limited to only computer systems not found in the first plurality of computing systems or even that it is distinct from the first plurality. Even if it did, Barsness contemplates multiple additional compute nodes that are used as resources for provisioning nodes in order to meet system performance requirements, as explained above. These additional nodes also form a second plurality of computer systems, which includes duplicate compute nodes that perform the first transformation pipeline, the second transformation pipeline, or both.

Finally, the management system 105 is a *computer system of the first plurality of computer systems* as discussed above in paragraphs 50-52. The management system is a computer system including a CPU 305 that “retrieve[s] and *execute[s] programming instructions* stored in the memory 325 and storage 330,” including a node manager 340. EX1004, 9:51-62 (emphasis added), 10:12-13. A detailed view of the management system, including the node manager as mentioned above, is shown in FIG. 3 of Barsness:

(Figure omitted. EX1004, FIG. 3 (highlighting added))

The management system executes the node manager “to configure the compute nodes 130 with respect to loading programming instructions and related configurations, assigning or reassigning processing elements on different compute nodes 130, etc.” EX1004, 10:14-18; *id.*, 10:18-28 (describing the node manager “configuring each processing element and operator”).

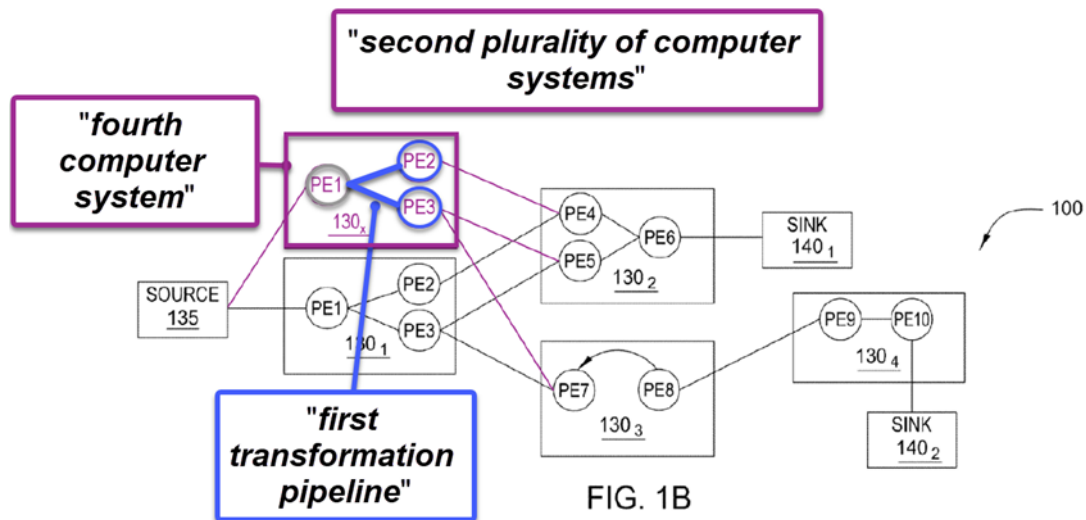
I agree with Dr. Crovella because, in my opinion, each of these newly spawned duplicate compute nodes executes software instructions that apply at least one of the first transformation pipeline and the second transformation pipeline, as carried out by the respective processing elements. Each duplicate compute node is one of a second plurality of computer systems, for example a second plurality comprising the duplicate compute node together with the compute nodes of the first plurality of

computer systems. As I noted above, the recited *second plurality of computer systems* in claim 1 requires at most only distinction from computer systems of the first plurality. Even if it were, Barsness contemplates multiple additional compute nodes used as resources for provisioning nodes to meet system performance requirements, as I explained above. Those additional nodes also form a *second plurality of computer systems*, which includes duplicate compute nodes that perform the *first transformation pipeline*, the *second transformation pipeline*, or both. A POSITA would have understood each duplicate compute node to be a *fourth computer system* of the *second plurality of computer systems*. I further agree with Dr. Crovella because, in my opinion, the management system 105 is a *computer system of the first plurality of computer systems*, as I discussed above. The management system is a computer system including a CPU 305 that “retrieve[s] and execute[s] programming instructions stored in the memory 325 and storage 330,” including a node manager 340. EX1004, 9:51-62, 10:12-13. FIG. 3 of Barsness provides a detailed view of the management system, including the node manager. The management system executes the node manager “to configure the compute nodes 130 with respect to loading programming instructions and related configurations, assigning or reassigning processing elements on different compute nodes 130, etc.” EX1004, 10:14-18; *id.*, 10:18-28 (the node manager “configuring each processing element and operator”). A POSITA would have understood the

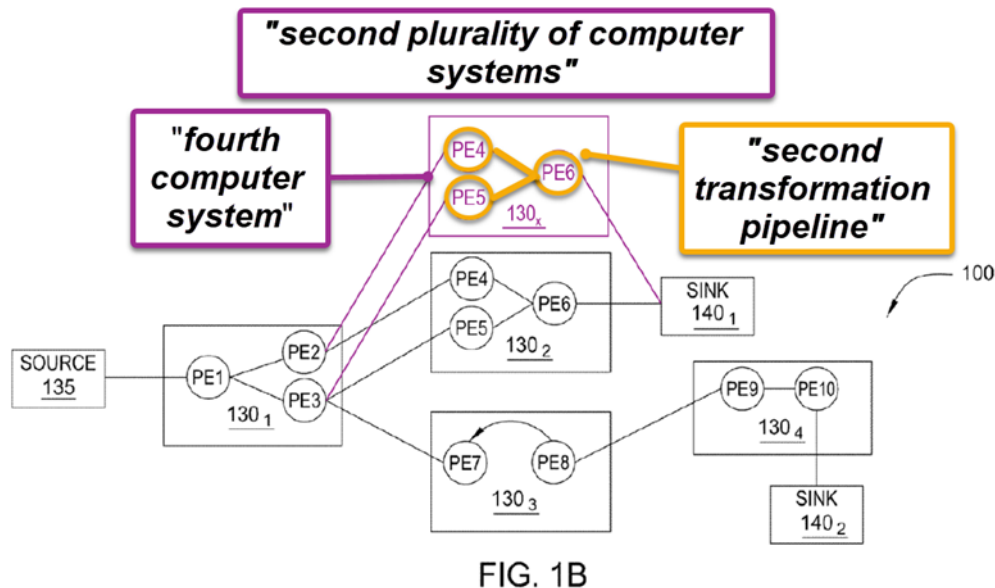
management system to be the *third computer system* of the *first plurality of computer systems* that executes software instructions causing a *fourth computer system* to apply a transformation pipeline.

107. **Alternatively**, Barsness teaches monitoring system 110 (*third computer system*) *cause[s]* the provisioning of compute node 130_x (*fourth computer system*) by “instruct[ing] the management system 105 to spawn more process nodes 130 to process certain type of data when the monitoring system 110 determines a need based on a detection of performance issues with a computer node 130 or the processing elements 235 in a computer node 130.” EX1004, 10:28-34; *see also id.*, 10:34-39. Barsness also teaches monitoring system 110 configuring additional compute nodes. EX1004, 11:37-41 (“The monitoring application may ... configure the compute nodes 130 with respect to assigning or reassigning processing elements on different compute nodes 130, spawning new processing elements 235, operators 240, etc.”). Thus, when monitoring system 110 detects performance issues for compute node 130₁ or 130₂, the monitoring system 110 *cause[s]* management system 105 to provision compute node 130_x and configure it with the processing elements of compute node 130₁ (*first transformation pipeline*) or 130₂ (*second transformation pipeline*). EX1004, 10:28-39; *see also id.*, 12:45-13:3, 13:12-14, 14:51-56. The preceding explanation regarding the management system’s spawning and duplicating additional compute nodes applies here as well with the monitoring

system instructing the management system. Thus, compute node 130_x processes the same type of stream data in parallel, “increas[ing] data throughput” (*id.*, 10:18-28) and maintaining the desired “processing rate” for those PEs (*id.*, 14:54-56, 14:30-39), either in parallel with compute node 130₁ or compute node 130₂ as seen in the figures below:



EX1004, FIG. 1B (annotations added).



EX1004, FIG. 1B (annotations added). A POSITA would have found it obvious to duplicate compute nodes 130₁ and 130₂ because Barsness discloses monitoring system 110 being configured to interact with management system 105 to duplicate any compute node 130 in the context of corrective measures for improving the data processing rates measured by monitoring system 110. EX1004, 10:26-39, 14:26-56. 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). As discussed above, monitoring system 110 is a *computer system of the first plurality of computer systems*. *Supra* Section VII.A.2.b.

- C. **Claim 2 - The distributed computing cluster of claim 1, wherein at least one of the first transformation pipeline and the second transformation pipeline is non-linear.**

108. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶108) states:

Barsness renders this claim obvious. Barsness renders the distributed computing cluster of claim 1 obvious, as discussed above in Section IX.A (claim 1). Barsness discloses non-linear transformation pipelines. For example, as shown in FIG. 1B of Barsness, both the first transformation pipeline in the first compute node 130₁ and the second

transformation pipeline in the second compute node 130₂ exhibit branching:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Here, the operator graph shows efferent (fan-out) branching from PE1 to PE2 and PE3 and afferent (fan-in) branching from PE4 and PE5 to PE6. EX1004, 7:55-61, FIG. 1B. The operator graph also shows a third transformation pipeline comprising the processing elements PE7 and PE8 in the third compute node 130₃, where “data tuples flowing to PE8 [] loop[] back to PE7,” a cyclical flow of data. EX1004, 7:64-67, FIG. 1B. The fourth transformation pipeline comprising the processing elements PE9 and PE10 on the fourth compute node 130₄ is, by contrast, a linear transformation pipeline because the data flows linearly from PE9 to PE10.

I agree with Dr. Crovella because, in my opinion, Barsness renders this claim obvious. Barsness renders the *distributed computing cluster of claim 1* obvious, as I discussed above in Section VII.B for claim 1. Barsness also discloses *non-linear* transformation pipelines. For example, as FIG. 1B of Barsness shows, both the *first transformation pipeline* in the first compute node 130₁ and the *second transformation pipeline* in the second compute node 130₂ exhibit branching.

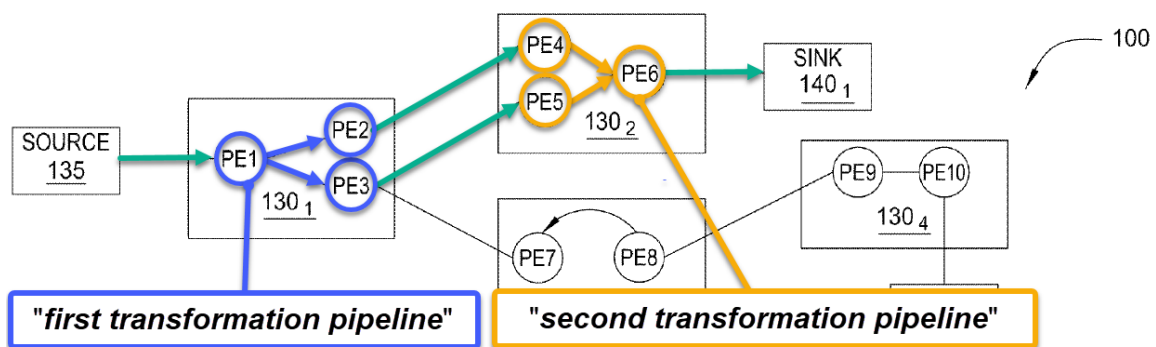


FIG. 1B

EX1004, FIG. 1B (annotations added). The operator graph shows efferent, or fan-out, branching from PE1 to PE2 and PE3, and afferent, or fan-in, branching from PE4 and PE5 to PE6. EX1004, 7:55-61, FIG. 1B. The operator graph also shows a third transformation pipeline comprising processing elements PE7 and PE8 in the third compute node 130₃, where “data tuples flowing to PE8 [] loop[] back to PE7,” a cyclical flow of data. EX1004, 7:64-67, FIG. 1B. By contrast, the fourth transformation pipeline comprising processing elements PE9 and PE10 on the fourth compute node 130₄ is a linear transformation pipeline, because the data flows linearly from PE9 to PE10. A POSITA would have understood at least one of the *first transformation pipeline* and the *second transformation pipeline* in Barsness to be *non-linear*.

109. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶109) states:

Thus, at least one of the first transformation pipeline and the second transformation pipeline in an obvious implementation of Barsness is non-linear. Regarding non-linear transformation pipelines, the ’424 patent indicates that pipelines configured with an “afferent branch,” an “efferent branch,” or a “cyclical” configuration are non-linear. EX1001, 10:4-7, FIGS. 13-15. As reflected in FIG. 13 of the ’424 patent and its corresponding disclosure, an “afferent branch” configuration refers to output from each of two nodes flowing into the input of a third node, e.g., a fan-in arrangement. EX1001, FIG. 13, 21:42-60. Similarly, an “efferent branch” refers to a single node outputting to the inputs of a second and third node, e.g., a fan-out arrangement. EX1001, FIG. 14, 22:1-21. A “cyclical” configuration refers to the output of a node flowing back to a preceding node. EX1001, FIG. 15, 22:28-46.

I agree with Dr. Crovella because, in my opinion, at least one of the *first transformation pipeline* and the *second transformation pipeline* in an obvious implementation of Barsness is *non-linear*. As to what makes a transformation pipeline non-linear, the '424 patent indicates that pipelines configured with an “afferent branch,” an “efferent branch,” or a “cyclical” configuration are non-linear. EX1001, 10:4-7, FIGS. 13-15. As FIG. 13 of the '424 patent and its corresponding disclosure reflect, an “afferent branch” configuration refers to output from each of two nodes flowing into the input of a third node, for example a fan-in arrangement. EX1001, FIG. 13, 21:42-60. An “efferent branch” likewise refers to a single node outputting to the inputs of a second and a third node, for example a fan-out arrangement. EX1001, FIG. 14, 22:1-21. And a “cyclical” configuration refers to the output of a node flowing back to a preceding node. EX1001, FIG. 15, 22:28-46. A POSITA would have understood the branching and cyclical pipelines in Barsness to be *non-linear* under these descriptions.

- D. **Claim 3 - The distributed computing cluster of claim 2, wherein the third computer system of the first plurality of computer systems is configured to: identify a fifth computer system of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the first transformation pipeline to any stream of input data, and cause the fifth computer system to process a**

second stream of input data received from a second input feed
substantially in real time by executing software instructions that
apply the first transformation pipeline to the second stream of
input data.

110. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶110)

states:

Barsness renders claim 3 obvious. Barsness renders obvious the distributed computing cluster of claim 2, as discussed above in Section IX.B. Further, Barsness discloses a management system (*the third computer system of the first plurality of computer systems*) that identifies additional compute nodes and provisions those compute nodes to, for example, duplicate existing compute nodes and “to improve the data processing performance of the stream application.” EX1004, 10:34-39; *id.*, 10:30-34; *id.*, 8:45-48, FIG. 6, 14:30-56; *supra* paragraphs 100-107 (element 1.7). As explained in the following paragraphs, the Barsness teachings on node provisioning, scalability, and multiple input sources render obvious a management system configured to satisfy claim 3.

I agree with Dr. Crovella because, in my opinion, Barsness renders claim 3 obvious.

Barsness renders obvious the distributed computing cluster of claim 2, as I discussed above. Barsness further discloses a management system, the *third computer system of the first plurality of computer systems*, that identifies additional compute nodes and provisions them, for example, to duplicate existing compute nodes and “to improve the data processing performance of the stream application.” EX1004, 10:34-39; *id.*, 10:30-34, 8:45-48, FIG. 6, 14:30-56; *supra* paragraphs 100-107 (element 1.7). It is my opinion that the Barsness teachings on node provisioning,

scalability, and multiple input sources render obvious a management system configured to satisfy claim 3.

111. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶111) states:

Barsness also teaches multiple input data streams, including a second stream of input data. For example, Barsness describes its operator graph representing "a stream application beginning from *one or more source processing elements* (PEs)..." and being "configurable to add or remove one or more PEs based on the data processing needs and scalability." EX1004, 6:49-55 (emphasis added); *id.*, 2:52 (discussing "data sources"), 10:36-28 (multiple PEs "used in parallel to process same type of data to increase data throughput").

I agree with Dr. Crovella because, in my opinion, Barsness also teaches multiple input data streams, including a *second stream of input data*. For example, Barsness describes its operator graph as representing "a stream application beginning from one or more source processing elements (PEs)" and as being "configurable to add or remove one or more PEs based on the data processing needs and scalability." EX1004, 6:49-55; *id.*, 2:52 ("data sources"), 10:36-28 (multiple PEs "used in parallel to process same type of data to increase data throughput"). A POSITA would have understood Barsness to accommodate a *second stream of input data*.

112. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶112) states:

Regarding scalability, Barsness teaches addressing increased "data processing needs" (which arise with an additional input stream) by provisioning additional compute nodes. EX1004, 6:49-55; *id.*, 3:61-63

(“clone a given PEs” if “running too slowly to process data tuples through the stream”); *id.*, 6:49-55. Barsness teaches achieving scalability “by distributing an application across nodes ... as well as replicating processing elements on multiple nodes and load balancing among them.” EX1004, 3:19-22; *id.*, 6:16-42 (cloud computing infrastructure enables the “provision of scalable computing resources as a service over a network”). Similarly, Barsness teaches addressing processing needs by having the management system “spawn more process nodes ... to process certain type of data” (EX1004, 10:28-31), and it expressly lists “spawning new compute nodes, duplicating a compute node, [and] spawning new processing elements” as example corrective measures (EX1004, 14:55-56). Collectively, these and other passages teach configuring the management system such that, when an additional input source stream is introduced, it duplicates/spawns existing compute nodes in order to scale and handle the additional stream and associated processing demand.

I agree with Dr. Crovella because, in my opinion, and as to scalability, Barsness teaches addressing increased “data processing needs,” which arise with an additional input stream, by provisioning additional compute nodes. EX1004, 6:49-55; *id.*, 3:61-63 (“clone a given PEs” if “running too slowly to process data tuples through the stream”), 6:49-55. Barsness teaches achieving scalability “by distributing an application across nodes ... as well as replicating processing elements on multiple nodes and load balancing among them.” EX1004, 3:19-22; *id.*, 6:16-42 (cloud computing infrastructure enables the “provision of scalable computing resources as a service over a network”). Barsness likewise teaches addressing processing needs by having the management system “spawn more process nodes ... to process certain type of data,” EX1004, 10:28-31, and it expressly lists “spawning new compute nodes, duplicating a compute node, [and] spawning new processing elements” as

example corrective measures, EX1004, 14:55-56. It is my opinion that, taken together, these passages teach configuring the management system so that, when an additional input source stream is introduced, it duplicates or spawns existing compute nodes to scale and handle the additional stream and its processing demand.

113. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶113) states:

A POSITA would also have reasonably expected success with such an implementation because Barsness explains how to do so, as compute nodes may be "a physical host or a virtual machine" (EX1004, 8:37-39), virtualized deployments may "spawn more virtual machines on the fly" (EX1004, 8:47-48), and compute nodes may be provisioned by loading "programming instructions and configurations to be executed by the compute nodes" (EX1004, 7:13-15); *see also* EX1004, 10:12-42 (describing how system configures and sets up "processing elements and operators in the compute nodes 130"). Barsness further explains, in the context of its FIG. 1B example, that the management system "stream manager" is configured to "control the stream application by inserting, removing, or otherwise modifying the processing elements (or what data-tuples flow to the processing elements) running on the compute nodes 130₁₋₄." EX1004, 7:33-37; *see also id.*, 7:29-33.

I agree with Dr. Crovella because, in my opinion, a POSITA would also have reasonably expected success with such an implementation, because Barsness explains how to do so: compute nodes may be "a physical host or a virtual machine," EX1004, 8:37-39, virtualized deployments may "spawn more virtual machines on the fly," EX1004, 8:47-48, and compute nodes may be provisioned by loading "programming instructions and configurations to be executed by the compute nodes," EX1004, 7:13-15; *see also* EX1004, 10:12-42 (how the system configures

and sets up “processing elements and operators in the compute nodes 130”). Barsness further explains that the management system “stream manager” is configured to “control the stream application by inserting, removing, or otherwise modifying the processing elements (or what data-tuples flow to the processing elements) running on the compute nodes 130₁₋₄.” EX1004, 7:33-37; *see also id.*, 7:29-33. A POSITA would have reasonably expected success in configuring the management system in this way.

114. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶114) states:

Barsness therefore expressly teaches the operator graph having multiple input streams and scaling the operator graph as needed to handle incoming data. *E.g.*, EX1004, 6:16-42; *id.*, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42. These contemplated additional input source streams of the same data type (e.g., due to increased volume or data coming from another location), predictably increase incoming data volume and/or the number of concurrent inputs the system must process. POSITAs would expect scenarios where the stream application needs to process multiple input streams, and thus scales accordingly, especially when using a “cloud computing infrastructure,” as taught by Barsness. EX1004, 6:16-42. For example, Heinze, an Association for Computing Machinery tutorial on scaling mechanisms in cloud-based stream processing systems, is a background reference reflecting the state of the art, and explains that “[c]loud-based data streaming engines, in particular, are designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads.” EX1008, 240-241. Heinze further discusses example data streams that are handled by data streaming engines, such as social network data streams, that “require systems to elastically scale and to support diverse workloads” because there may be “significant spikes” in data volume and data streams, including additional streams originating from around

the world. EX1008, 238-241; *see also* EX1007, [0243] (adding nodes to handle “additional data streams”); *see below* Section XI.

I agree with Dr. Crovella because, in my opinion, Barsness therefore expressly teaches the operator graph having multiple input streams and scaling the operator graph as needed to handle incoming data. *E.g.*, EX1004, 6:16-42; *id.*, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42. These contemplated additional input source streams of the same data type, whether from increased volume or from data coming from another location, predictably increase the incoming data volume, the number of concurrent inputs the system must process, or both. A POSITA would have expected scenarios in which the stream application needs to process multiple input streams and scales accordingly, especially when using a “cloud computing infrastructure” as taught by Barsness. EX1004, 6:16-42. For example, Heinze, an Association for Computing Machinery tutorial on scaling mechanisms in cloud-based stream processing systems, is a background reference reflecting the state of the art, and explains that “[c]loud-based data streaming engines, in particular, are designed to dynamically scale to hundreds of computing nodes and automatically cope with varying workloads.” EX1008, 240-241. Heinze further discusses example data streams handled by data streaming engines, such as social network data streams, that “require systems to elastically scale and to support diverse workloads” because there may be “significant spikes” in data volume and data streams, including additional

streams originating from around the world. EX1008, 238-241; *see also* EX1007, [0243] (adding nodes to handle “additional data streams”); *see below* Section XI.

115. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶115) states:

Thus, Barsness contemplates multiple input streams and input nodes, and teaches scaling its streaming application to handle processing needs associated with additional input streams. Given these teachings, a POSITA would have been motivated with a reasonable expectation of success to configure a Barsness management system to, upon addition of a new input data stream: provision and configure an available compute node with processing elements for the first transformation pipeline (*identify a fifth computer systems of the first plurality of computer systems, wherein the fifth computer system is not configured to apply the first transformation pipeline to any stream of input data*), and assign that duplicated provisioned node to process the second data stream with the first transformation pipeline (*cause the fifth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data*). *E.g.*, EX1004, 7:27-37, 10:12-64, 11:30-34, 14:51-56.

I agree with Dr. Crovella because, in my opinion, Barsness contemplates multiple input streams and input nodes and teaches scaling its streaming application to handle the processing needs associated with additional input streams. Given these teachings, a POSITA would have been motivated, with a reasonable expectation of success, to configure a Barsness management system so that, upon the addition of a new input data stream, it would provision and configure an available compute node with processing elements for the first transformation pipeline, that is, *identify a fifth computer system of the first plurality of computer systems, wherein the fifth*

computer system is not configured to apply the first transformation pipeline to any stream of input data, and would assign that provisioned node to process the second data stream with the first transformation pipeline, that is, *cause the fifth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data*. E.g., EX1004, 7:27-37, 10:12-64, 11:30-34, 14:51-56.

116. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶116) states:

Barsness discloses identifying a fifth computer system of the first plurality of computer systems because the management system (the third computer system) manages the compute nodes and thus knows what compute nodes are available and not currently executing a transformation pipeline. EX1004, 10:12-64. To the extent that Barsness does not expressly disclose the management system identifying a fifth computer system, a POSITA would have been motivated with a reasonable expectation of success to implement the management system to do so in order to “manage the processing loads of the compute nodes 130,” as taught by Barsness. EX1004, 7:27-33. For example, the management system is responsible for “assigning or reassigning processing elements on different compute nodes.” EX1004, 11:30-34; *id.*, 14:51-56 (describing “automatic duplication of an operator” and automatically “spawning new processing elements”). Specifically, the management system is “configured for moving processing elements (PEs) from one compute node 130 to another, e.g., to manage the processing loads of the compute nodes 130 in the computing infrastructure 100.” EX1004, 7:29-33. The management system may also “control the stream application by inserting, removing, or otherwise modifying the processing elements (or what data-tuples flow to the processing elements) running on the compute nodes 130₁₋₄.” EX1004, 7:33-37. I note that Barsness teaches the monitoring system

performing a number of automatic corrective actions (EX1004, 14:51-56) but also explains that the monitoring system accomplishes those actions (e.g., spawning new compute nodes, duplicating a compute node) by communicating with the management system. *E.g.*, EX1004, 14:30-35; 12:64-67 (the monitoring system interacts with the management system “to determine if rearranging some processing elements on computer nodes 130 may remedy the issue”). Given that the management system handles management and configuration of the compute nodes, the actions taken to dynamically adjust / scale the processing is performed by the management system, rather than the monitoring system.

I agree with Dr. Crovella because, in my opinion, Barsness discloses *identifying a fifth computer system of the first plurality of computer systems*, because the management system, the *third computer system*, manages the compute nodes and therefore knows which compute nodes are available and not currently executing a transformation pipeline. EX1004, 10:12-64. To the extent Barsness does not expressly disclose the management system identifying a *fifth computer system*, a POSITA would have been motivated, with a reasonable expectation of success, to implement the management system to do so in order to “manage the processing loads of the compute nodes 130,” as Barsness teaches. EX1004, 7:27-33. For example, the management system is responsible for “assigning or reassigning processing elements on different compute nodes.” EX1004, 11:30-34; *id.*, 14:51-56 (“automatic duplication of an operator” and automatically “spawning new processing elements”). Specifically, the management system is “configured for moving processing elements (PEs) from one compute node 130 to another, e.g., to manage the processing loads

of the compute nodes 130 in the computing infrastructure 100.” EX1004, 7:29-33.

The management system may also “control the stream application by inserting, removing, or otherwise modifying the processing elements (or what data-tuples flow to the processing elements) running on the compute nodes 130₁₋₄.” EX1004, 7:33-37. Although Barsness teaches the monitoring system performing a number of automatic corrective actions, EX1004, 14:51-56, it also explains that the monitoring system accomplishes those actions, such as spawning new compute nodes and duplicating a compute node, by communicating with the management system. *E.g.*, EX1004, 14:30-35; 12:64-67 (the monitoring system interacts with the management system “to determine if rearranging some processing elements on computer nodes 130 may remedy the issue”). Because the management system handles the management and configuration of the compute nodes, it is the management system, rather than the monitoring system, that performs the actions taken to dynamically adjust or scale the processing.

117. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶117) states:

The POSITA would have been motivated to implement the computing infrastructure in this way to beneficially provide system scaling, as needed (e.g., by duplicating the stream application by duplicating the compute nodes 130₁₋₄ executing the stream application), for an additional data stream because this would handle data processing needs, as taught by Barsness. The POSITA would further be motivated to simplify state management of the data stream, because duplicating the stream application allows the new data stream to be processed in real-

time without imposing additional complexity and processing load on the compute nodes 130₁₋₄. The fact that a POSITA would have easily and predictably implemented such a system is confirmed by the absence of any description in the '424 patent of any computer systems causing another computer system to apply an existing transformation pipeline to a second stream of input data. Further, a POSITA would have reasonably expected success with such an implementation because Barsness explains how to do so, with its teachings on provisioning compute nodes. *E.g.*, EX1004, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42.

I agree with Dr. Crovella because, in my opinion, a POSITA would have been motivated to implement the computing infrastructure in this way to provide system scaling as needed, for example by duplicating the stream application through duplicating the compute nodes 130₁₋₄ executing it, for an additional data stream, because this would handle the data processing needs Barsness teaches. A POSITA would further have been motivated to simplify the state management of the data stream, because duplicating the stream application allows the new data stream to be processed in real time without imposing additional complexity and processing load on the compute nodes 130₁₋₄. The absence of any description in the '424 patent of any computer system causing another computer system to apply an existing transformation pipeline to a second stream of input data confirms that a POSITA would have easily and predictably implemented such a system. And a POSITA would have reasonably expected success with such an implementation, because Barsness explains how to do so through its teachings on provisioning compute nodes. *E.g.*, EX1004, 7:13-15, 7:29-37, 8:37-39, 8:47-48, 10:12-42.

118. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶118)

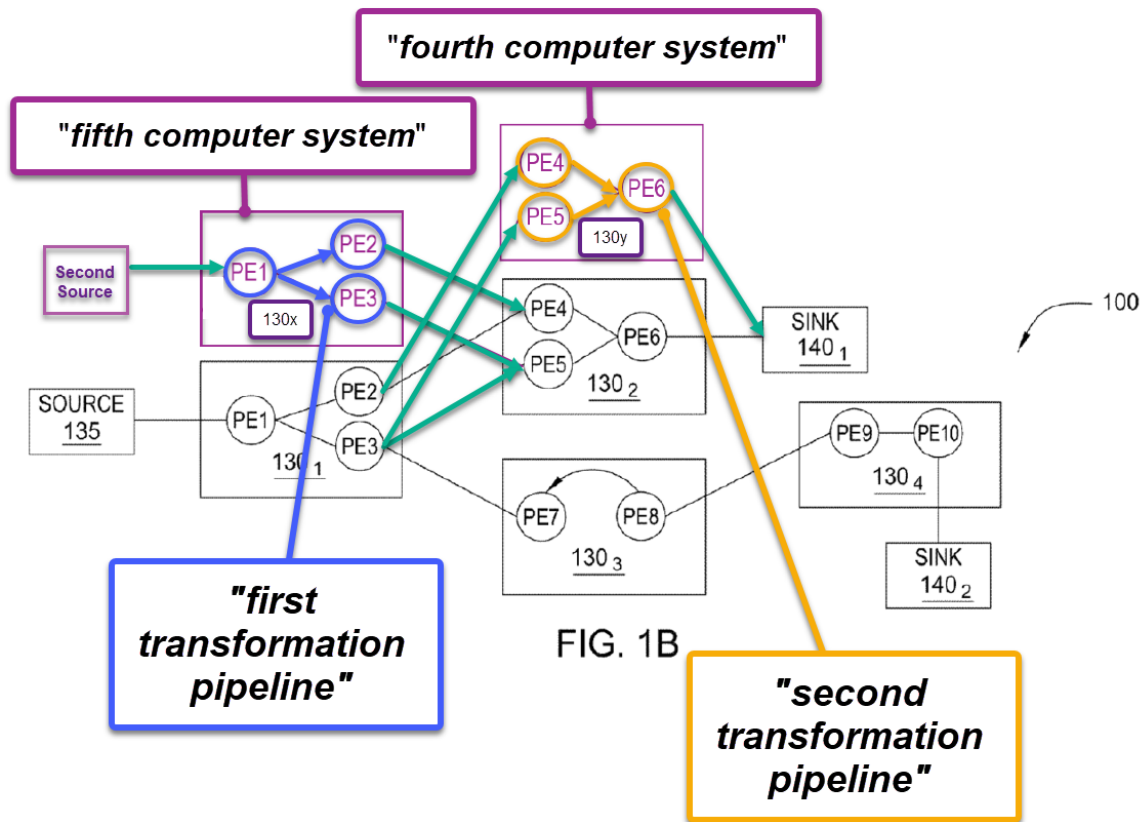
states:

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

(Figure omitted. EX1004, FIG. 1B (purple annotations added))

Though not depicted, a POSITA would have been motivated with a reasonable expectation of success to configure the management system to scale the entire streaming application, i.e., duplicate all compute nodes, to handle the increased processing needs of the second source. *E.g., id.*, 7:29-37. This would result in duplication of third and fourth compute nodes 130₃ and 130₄ with PEs 7-10. These newly spawned compute nodes are a second plurality of computer systems, including compute node 130_x (*fifth computer system*) and compute node 130_y. Any two or more of these newly spawned compute nodes are a second plurality of computer systems, including compute node 130_x (*fifth computer system*) and compute node 130_y.

I agree with Dr. Crovella because, in my opinion, the annotated Barsness FIG. 1B shows an example of the management system scaling compute nodes to handle a new input stream, with a provisioned compute node 130_x, a ***fifth computer system***, configured to apply processing elements PE1, PE2, and PE3, the ***first transformation pipeline***, to a second source of data, a ***second stream of input data received from a second input feed***:



EX1004, FIG. 1B (annotations added). Although not depicted, a POSITA would have been motivated, with a reasonable expectation of success, to configure the management system to scale the entire streaming application, that is, to duplicate all compute nodes, to handle the increased processing needs of the second source. *E.g.*, *id.*, 7:29-37. This would duplicate the third and fourth compute nodes 130₃ and 130₄ with PE7 through PE10. These newly spawned compute nodes are a *second plurality of computer systems*, including compute node 130_x, the *fifth computer system*, and compute node 130_y. Any two or more of these newly spawned compute nodes are a *second plurality of computer systems*, including compute node 130_x, the *fifth computer system*, and compute node 130_y. A POSITA would have understood the

management system to *cause the fifth computer system to process a second stream of input data received from a second input feed* by applying the *first transformation pipeline*.

E. **Claim 4**

1. **[4.1] - The distributed computing cluster of claim 3, wherein the third computer system is further configured to execute software instructions that:**

119. In my opinion, Barsness satisfies this claim. In the Barsness architecture, the management system is the *third computer system*, and it serves as the central point at which the structure of the entire streaming application is recorded and kept. The management system stores in its memory the operator graph of the streaming application. EX1004, 6:48-49 (“Also, the management system 105 includes an operator graph 132 and a stream manager 134.”); *id.*, 10:12-13 (“As shown, the memory 325 stores a node manager 340 and a stream manager 134.”). That operator graph is the data structure through which Barsness expresses the application’s *distributed computational graph*. EX1004, 6:50-51 (“the operator graph 132 represents a stream application”). The operator graph is not a mere abstraction; it enumerates the concrete constituents of the running application, namely its compute nodes, the processing elements hosted on those nodes, and the operators executed by those processing elements. EX1004, 1:43-46 (“The operator

graph represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data tuples through the stream application.”). Because the operator graph enumerates each of these constituents, it necessarily contains the information that represents the compute nodes, processing elements, and operators that together form the *first transformation pipeline*. A POSITA would have understood that the value of this operator graph lies in it accurately reflecting the current state of the streaming application. The management system does not build the graph once and then abandon it; it maintains the operator graph so that the graph continuously reflects the compute nodes, processing elements, and operators of the streaming application as they exist at any given moment. EX1004, 7:27-29 (“The stream manager 134 may be configured to ... change the structure of the operator graph 132.”). That currency matters because the Barsness application is dynamic, with its topology changing as the system responds to shifting load and to node failures. When the management system spawns or duplicates a new compute node, it therefore does not leave the graph stale; Barsness adds that new compute node, together with the processing elements and operators it hosts, to the operator graph, so that the graph remains an accurate and authoritative description of the application as actually deployed. EX1004, 14:51-56 (“These complex operations may include automatic adjustment of an operator within compute nodes 130, automatic fusion or separation of operators, automatic duplication of an operator,

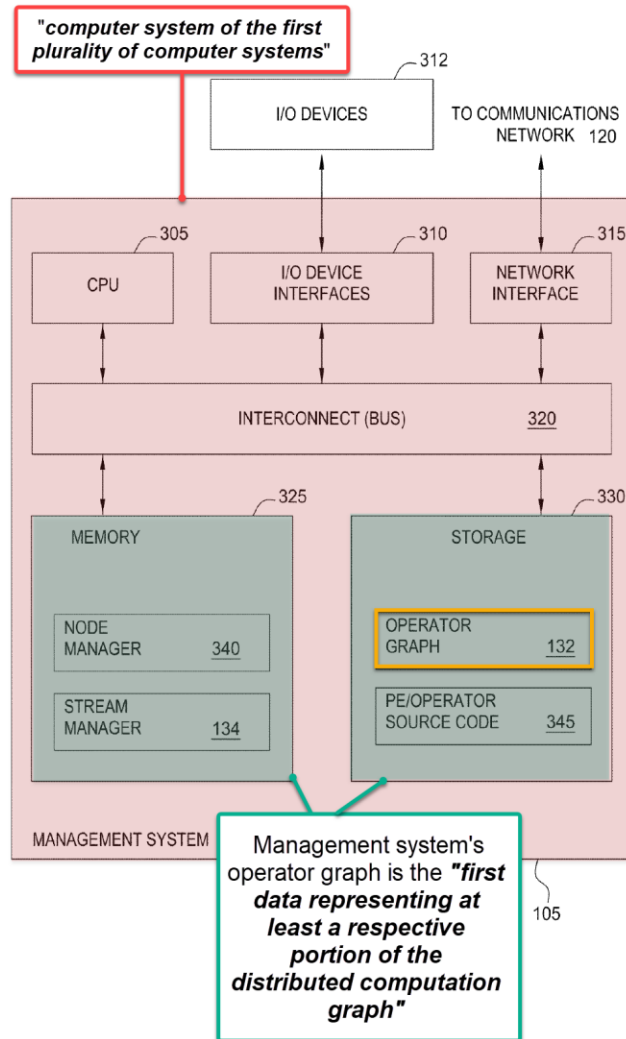
spawning new compute nodes, duplicating a compute node, spawning new processing elements, etc.”).

120. The management system carries out this work through two cooperating components, and it is those components that *execute software instructions* within the meaning of the claim. The management system includes a node manager 340 and a stream manager 134. EX1004, 10:12-13 (“As shown, the memory 325 stores a node manager 340 and a stream manager 134.”). Those components “configure the compute nodes 130” and “monitor the data stream as it flows through the processing elements in a stream application.” EX1004, 10:14-18 (“The node manager 340 may be used to configure the compute nodes 130 with respect to loading programming instructions and related configurations, assigning or reassigning processing elements on different compute nodes 130, etc.”); *id.*, 10:40-42 (“The stream manager 134 may be used to monitor the data stream as it flows through the processing elements in a stream application.”). Configuring the compute nodes is the mechanism by which the management system establishes and adjusts where and how the processing elements of the application execute, and monitoring the data stream as it flows through those processing elements is how the management system observes the application in operation and detects the conditions that prompt it to update the operator graph or to provision additional compute nodes. Taken together, these operations confirm that the management system, as the *third computer system*,

executes software instructions that govern how the application's transformation pipelines run.

2. **[4.2] - identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph,**

121. In my opinion, the management system 105 (*third computer system*) stores in its memory an operator graph 132, and that operator graph 132 is the *first data representing at least a respective portion of the distributed computational graph*. Barsness describes the management system 105 as a computing system having a CPU 305, a memory 325, and storage 330, and it identifies the operator graph 132 among the contents that the management system stores. EX1004, 9:62-67, 10:3-11; *id.*, 10:12-13 (“As shown, the memory 325 stores a node manager 340 and a stream manager 134. The storage 330 also includes the operator graph 132.”). The internal structure of the management system 105, and the place of the operator graph 132 within it, is depicted in the annotated FIG. 3 below.



EX1004, FIG. 3 (annotations added); *id.*, 10:12-64, *see also* EX1004, 9:62-67, 10:3-11. In my opinion, Barsness discloses that the operator graph “represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data tuples through the stream application” (***representing at least a respective portion of the distributed computational graph***). EX1004, 1:43-46; *id.*, 2:12-15 (“The operator graph represents the plurality of processing elements, and links between processing elements, corresponding to a flow of data

tuples through the stream application.”), 6:49-53 (“As described in greater detail below, the operator graph 132 represents a stream application beginning from of one or more source processing elements (PEs) through to one or more sink PEs.”), 10:49-51 (“As noted above, the operator graph 132 maintains the connectivity configuration plurality of compute nodes and processing elements in a streaming application.”), FIGS. 1A-1B. In other words, the operator graph models the streaming application as a graph whose nodes are the processing elements and whose edges are the links that carry data tuples between them. FIG. 1B depicts an example operator graph, in which the processing elements are the graph nodes and the links between processing elements are the graph edges, as shown below:

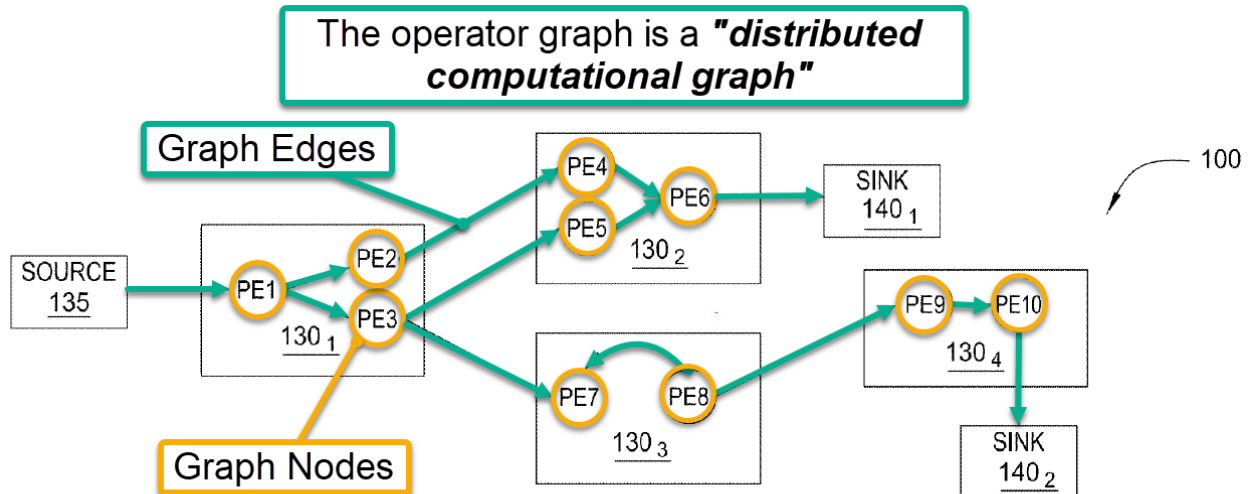


FIG. 1B

EX1004, FIG. 1B (annotations added). As I explained above in connection with limitation [1.2.a], the operator graph of Barsness is a ***distributed computational***

graph under either Petitioner’s proposed construction or the Patent Owner’s “plain and ordinary meaning.”

122. Barsness discloses that the management system’s stream manager 134 uses the operator graph stored in the management system’s memory both to control the running stream application and to “change the structure of the operator graph 132.” EX1004, 7:27-37. In particular, the stream manager is used to “control the stream application by inserting, removing, or otherwise modifying processing elements” running on the compute nodes, an operation that requires the management system to identify one or more of the respective memories in which those processing elements and their data reside. EX1004, 7:33-38. Thus, the management system’s stream manager 134 uses the operator graph stored in memory on the management system to “control the stream application by inserting, removing, or otherwise modifying the processing elements” (*identify one or more of the respective memories*). EX1004, 7:33-37. A POSITA would have recognized that “inserting, removing, or otherwise modifying the processing elements” would have required the stream manager to identify and process the existing operator graph. A POSITA would have also recognized that managing the streaming application with these modifications would have required updating the management system’s stored operator graph to maintain its accuracy in reflecting the streaming application. Additionally, Barsness discloses the node manager 340 “setting up one or more

processing elements and operators in the compute nodes 130.” EX1004, 10:18-20.

Similarly, a POSITA would have understood that the node manager required identifying and processing the management system’s stored operator graph in order to perform those operations and to update the operator graph to maintain it as an accurate reflection of the streaming application. Thus, both the stream manager 134 and node manager 340 *identify* and *store* the operator graph that on the management system as part of their management operations including updating the operator graph with information regarding the compute nodes, processing elements, and operators of the streaming application.

3. [4.2.a] - wherein the respective first data includes
information about the first transformation pipeline,

123. In my opinion, the *respective first data* includes information about the *first transformation pipeline*. As I explained above in connection with limitation 1.2.c, the operator graph describes the flow of data through several compute nodes, each of which hosts a group of processing elements that process and transform the data flowing through them. EX1004, 7:52-8:21; *see e.g.*, EX1004, 8:11-15 (“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, as per the configuration of target PEs.”). Because the operator graph records those compute nodes and their processing elements, it necessarily describes the compute nodes and

processing elements that make up the *first transformation pipeline*, as highlighted in the annotated FIG. 1B below.

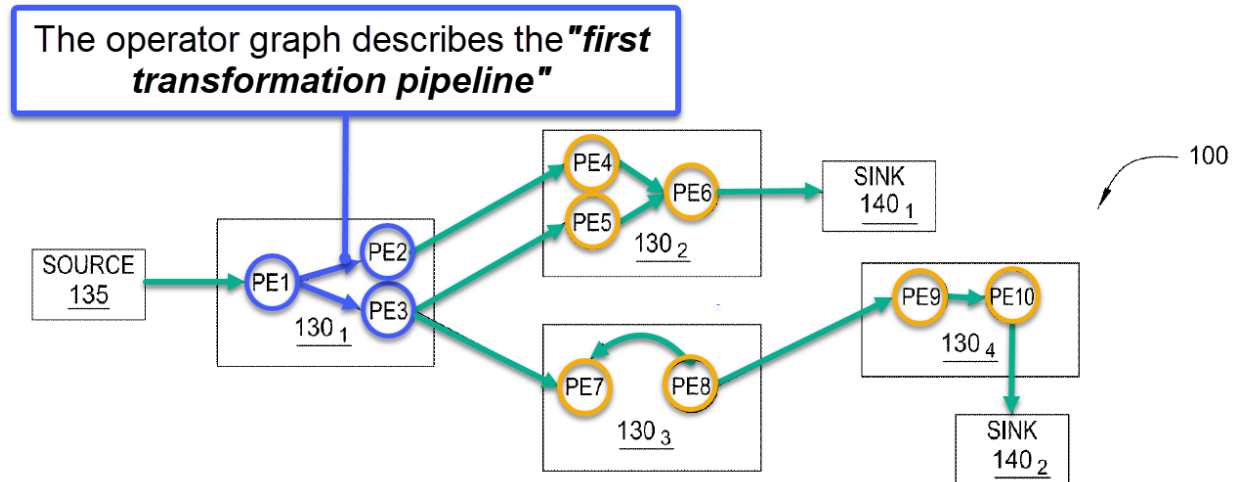


FIG. 1B

EX1004, FIG. 1B (annotations added); *see also id.*, 7:52-8:21. As explained above regarding limitations 1.2.c and 1.3.a-1.3.d, the compute node 130₁'s processing elements comprise the *first transformation pipeline*. Thus, the management system's operator graph, as explained above regarding 4.2 and 4.2.a, includes *information about the first transformation pipeline*.

4. [4.3] - and store in the identified one or more memories information identifying the fifth computer system.

124. In my opinion, the management system's in-memory operator graph includes information for each compute node and its processing elements, including the newly spawned or duplicated compute nodes 130_x and 130_y. EX1004, 10:49-51, 10:57-64. As I explained above in connection with limitation [1.7] and claim 3, the

management system spawns or duplicates compute nodes, such as compute node 130_x, to perform parallel processing of the data flowing through the streaming application. Barsness discloses that the operator graph “maintains the connectivity configuration of a plurality of compute nodes and processing elements in a streaming application.” EX1004, 10:49-51. It follows that the newly spawned or duplicated compute nodes, discussed above regarding limitation 1.7, such as 130_x and 130_y, (*fifth computer system*), are identified and information regarding those compute nodes is included as a part of the operator graph. Indeed, when the management system spawns or duplicates a compute node, the stream manager 134 or node manager 340 acts to “change the structure of the operator graph 132.” EX1004, 7:27-29, 10:49-51 (“the operator graph 132 maintains the connectivity configuration plurality of compute nodes and processing elements in a streaming application.”), 10:42-44 (“The stream manager 134 may also maintain logging and audit trails of the streaming data as the streaming data is processed by the compute nodes 130.”), 10:14-20 (“The node manager 340 may be used to configure the compute nodes 130 with respect to loading programming instructions and related configurations, assigning or reassigning processing elements on different compute nodes 130, etc. The node manager 340 may also be used for setting up one or more processing elements and operators in the compute nodes 130.”).

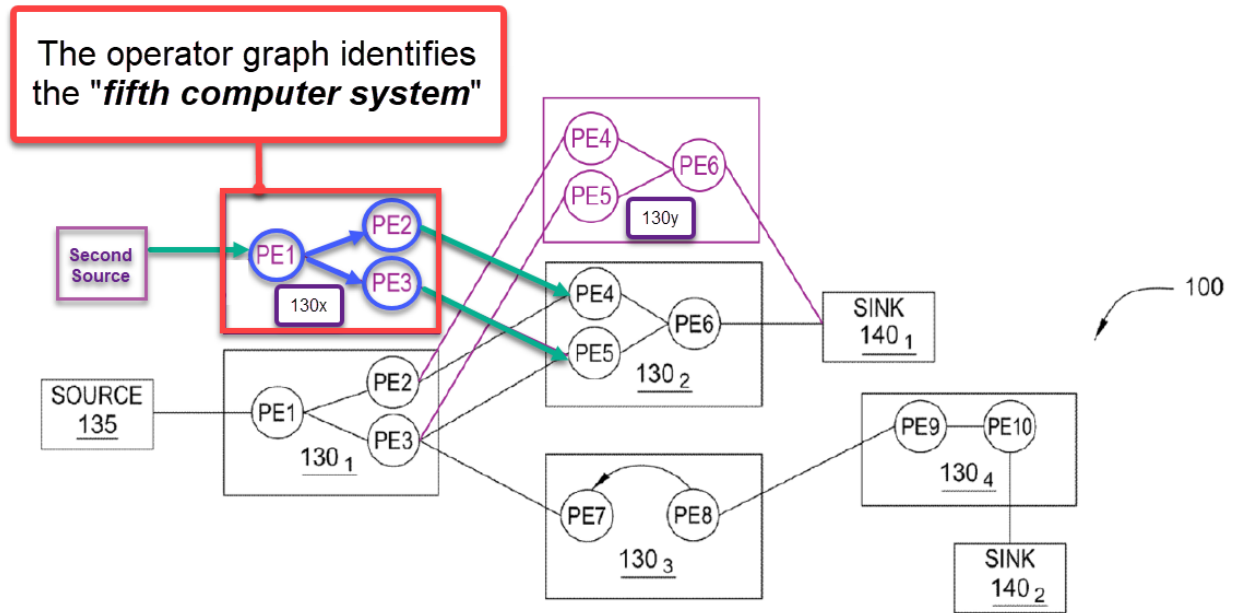


FIG. 1B

EX1004, FIG. 1B (annotations added).

125. In my opinion, a POSITA would have recognized that the operator graph is updated to include the information identifying the *fifth computer system*. The operator graph allows the management system 105 “to control a stream application running on the compute nodes 130₁₋₄ as well as change the structure of the operator graph 132.” EX1004, 7:27-29; *id.*, 6:48-56. Storing the operator graph 132 in the memory of the management system 105 allows the node manager 340 and the stream manager 134 to read and configure the operator graph, its representation of the compute nodes, processing elements, and operators, and perform configurations of those represented compute nodes, processing elements, and operators. EX1004, 7:27-37; *id.*, 10:12-64. Because the node manager 340 and the stream manager 134 each require a complete operator graph to perform their

155

operations over the compute nodes, processing elements, and operators of the stream application, a POSITA would have understood that the *fifth computer system* is added to the operator graph when the corresponding compute node is added to the stream application.

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

126. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶119)

states:

Barsness renders this claim obvious. Barsness renders obvious the distributed computing cluster of claim 2, as discussed above in Section IX.B. Further, Barsness discloses a management system (*the third computer system of the first plurality of computer systems*) that manages the creation of new compute nodes and the moving / replication of processing elements among compute nodes "to improve the data processing performance of the stream application." EX1004, 10:34-39; *id.*, 10:30-34; *id.*, 8:45-48; *supra* Sections IX.A.7 (element 1.7) and IX.C (claim 3).

I agree with Dr. Crovella because, in my opinion, Barsness renders this claim obvious. Barsness renders obvious the distributed computing cluster of claim 2, as I discussed above. Barsness further discloses a management system, the *third computer system of the first plurality of computer systems*, that manages the creation of new compute nodes and the moving or replication of processing elements among compute nodes “to improve the data processing performance of the stream application.” EX1004, 10:34-39; *id.*, 10:30-34, 8:45-48; *supra* regarding element 1.7 and claim 3.

127. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶120) states:

As discussed above in Section IX.C (claim 3), a POSITA would have found it obvious to implement the management system of Barsness to cause a compute node, such as the compute node 130_x, to apply the first transformation pipeline to a second stream of input data. In that obvious implementation of Barsness, the output of this duplicated compute node 130_x applying the first transformation pipeline comprises *third pipeline output messages*. Further, for the same reasons discussed above for claim 3, a POSITA would have been motivated to implement the management system 105 to cause the compute node 130_y (*fifth computer system*) to apply the processing elements PE4, PE5, and PE6 (*second transformation pipeline*) to the output data from PE2 and PE3 of the compute node 130_x (*process the third pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the third pipeline output messages*) in order to process data from the second source with the duplicated stream application, as shown in this annotated FIG. 1B:

(Figure omitted. EX1004, FIG. 1B (purple annotations added))

Id., 3:19-22 (describing achieving “[s]calability” of the stream application by “replicating processing elements on multiple nodes and load balancing among them”). In the alternative example where the third compute node 130₃ hosts the second transformation pipeline, the compute node 130_y may duplicate the processing elements PE7 and PE8 and thus duplicate the second transformation pipeline. Further, while only the processing elements PE1 through PE6 are depicted in this example, a POSITA would have found it obvious to duplicate the transformation pipelines of the compute nodes 130₃ and 130₄ as well to complete the stream application as configured in the operator graph.

I agree with Dr. Crovella because, in my opinion, and as I discussed above regarding claim 3, a POSITA would have found it obvious to implement the management system of Barsness to cause a compute node, such as compute node 130_x, to apply the first transformation pipeline to a second stream of input data. In that obvious implementation of Barsness, the output of the duplicated compute node 130_x applying the first transformation pipeline is the ***third pipeline output messages***. For the same reasons discussed above for claim 3, a POSITA would have been motivated to implement the management system 105 to cause compute node 130_y, a ***fifth computer system***, to apply processing elements PE4, PE5, and PE6, the ***second transformation pipeline***, to the output data from PE2 and PE3 of compute node 130_x, that is, to ***process the third pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the third pipeline output messages***, in order to process data from the second source with the duplicated stream application, as the annotated FIG. 1B shows.

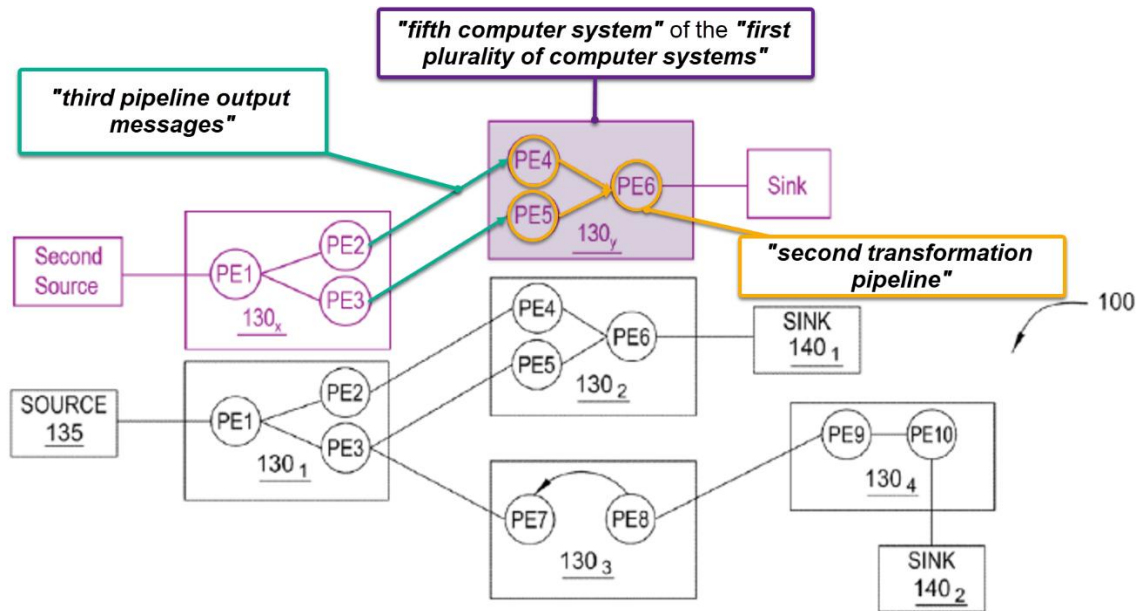


FIG. 1B

EX1004, FIG. 1B (annotations added); *id.*, 3:19-22 (achieving “[s]calability” of the stream application by “replicating processing elements on multiple nodes and load balancing among them”). In the alternative example where the third compute node 130₃ hosts the second transformation pipeline, compute node 130_y may duplicate processing elements PE7 and PE8 and thus duplicate the second transformation pipeline. And while only processing elements PE1 through PE6 are depicted in this example, a POSITA would have found it obvious to duplicate the transformation pipelines of compute nodes 130₃ and 130₄ as well, to complete the stream application as configured in the operator graph. A POSITA would have understood the management system to *cause the fifth computer system to process the third pipeline output messages* by applying the *second transformation pipeline*.

128. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶121)

states:

Barsness teaches, or at least renders obvious, configuring its management system (*the third computer system*) to “identify” a *fifth computer system of the first plurality of computer systems* for the same reasons explained for claim 3. *Supra* Section IX.C.

I agree with Dr. Crovella because, in my opinion, Barsness teaches, or at least renders obvious, configuring its management system, the *third computer system*, to *identify a fifth computer system of the first plurality of computer systems* for the same reasons explained for claim 3. A POSITA would have understood the management system to *identify a fifth computer system of the first plurality of computer systems* that is *not configured to apply the second transformation pipeline to any pipeline output message*.

G. **Claim 6**

1. [6.1] - The distributed computing cluster of claim 5, wherein the third computer system is further configured to execute software instructions that:
2. [6.2] - identify one or more of the respective memories that store respective first data representing at least a respective portion of the distributed computational graph,
3. [6.2.a] - wherein the respective first data includes information about the second transformation pipeline,

4. **[6.3] - and store in the identified one or more memories
information identifying the fifth computer system.**

129. In my opinion, Barsness satisfies this claim. As I explained above in connection with limitations [1.2] through [1.2.c], claim 3, and claim 4, the management system maintains the operator graph in its memory; the operator graph represents the *distributed computational graph*; the operator graph includes information about the compute nodes, processing elements, and operators of the *second transformation pipeline*; the stream manager and node manager use the operator graph to manage the stream application's compute nodes, processing elements, and operators such as when spawning or duplicating compute nodes; the stream manager and node manager read and update the operator graph as part of those management operations to keep the operator graph as accurately reflecting the stream application; and, thus, the operator graph stores identifying information for newly spawned or duplicated compute nodes, such as compute node 130_x or 130_y, which are the *fifth computer system*.

- H. **Claim 8 - The distributed computing cluster of claim 1, wherein the first computer system is further configured to transmit the first pipeline output messages to another computer system of the second plurality of computer systems.**

130. In my opinion, Barsness satisfies this claim. Barsness teaches that compute node 130₁ transmits data tuples to compute node 130₂ and compute node 130₃, which is *another computer system of the second plurality of computer systems*, as shown in the annotated FIG. 1B below.

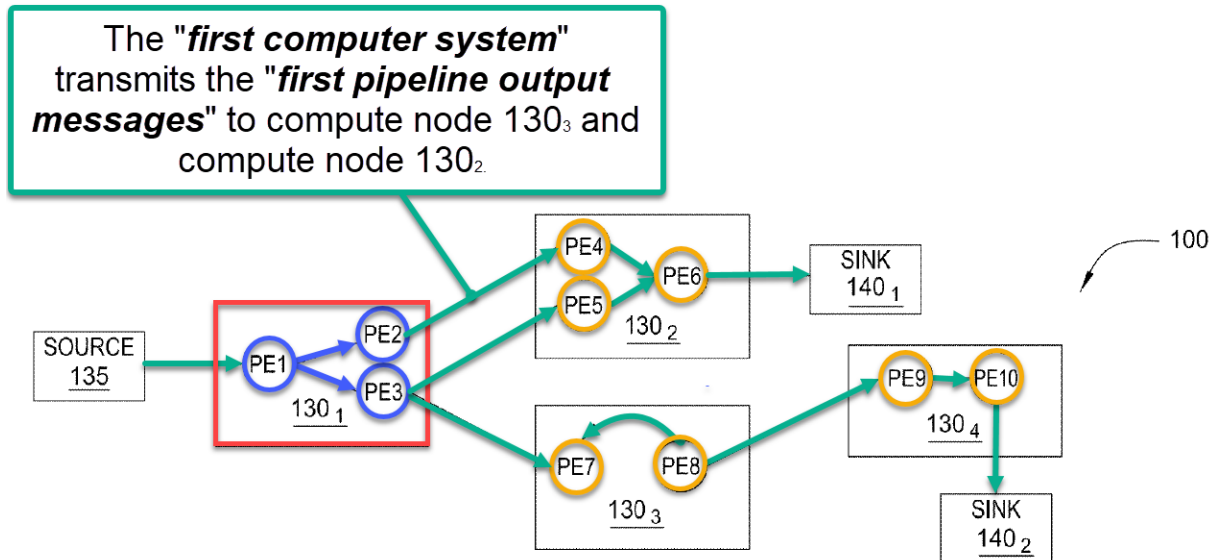
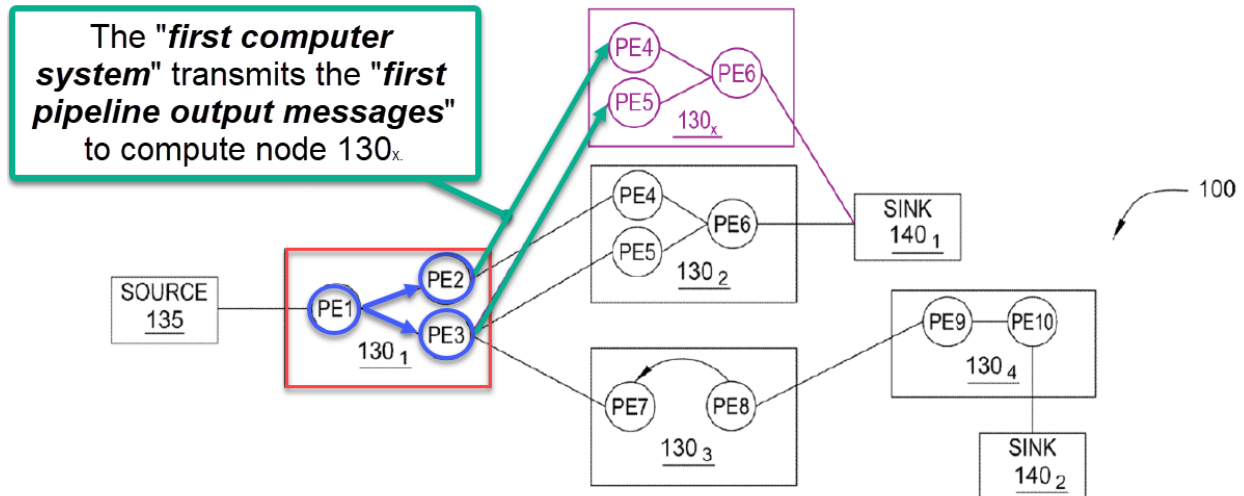


FIG. 1B

EX1004, FIG. 1B (annotations added). As I explained above, the claim recites a *second plurality of computer systems* without further restriction, which indicates that any group of two or more computer systems satisfies the limitation, including compute node 130₃.

131. Alternatively, in my opinion, Barsness renders this claim obvious. It would have been obvious to a POSITA to provision and configure several additional compute nodes to meet the processing performance needs of the application. As I discussed in connection with limitation [1.7], Barsness teaches that the monitoring

system 110 detects performance issues and instructs the management system 105 to provision and configure additional compute nodes and to assign processing elements to them, so that those additional compute nodes apply one of the claimed transformation pipelines and thereby improve tuple processing rates. As discussed above, Barsness discloses the management system 105 provisioning additional compute nodes, such as compute nodes 130_x in the annotated FIG. 1B below, to resolve performance issues detected on compute node 130_1 . See [1.7] and [3.a]. By provisioning more than one additional compute node to resolve the performance issues identified on compute node 130_1 , each of compute nodes 130_x and 130_y receives the *first pipeline output messages* from compute node 130_1 , which is the *first computer system*. Barsness further teaches that the management system 105 may provision the additional compute nodes 130_x and 130_y from the first plurality of computer systems, e.g. [1.7]'s fourth computer system, or from the second plurality of computer systems, e.g. [3.a]'s fifth computer system.



EX1004, FIG. 1B (annotations added).

I. **Claim 9**

1. **[9.1] - A distributed computing cluster comprising:**

132. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶127)

states:

Barsness satisfies the preamble because Barsness discloses a distributed computing cluster. *Supra* Section IX.A.1 (element 1.1).

I agree with Dr. Crovella because, in my opinion, to the extent the preamble is limiting, Barsness satisfies the preamble because Barsness discloses a ***distributed computing cluster***. *Supra* Section VII.B.1 (element 1.1). A POSITA would have understood Barsness to disclose ***a distributed computing cluster*** as recited in the preamble of claim 9.

2. **[9.2] - a first plurality of computer systems,**

133. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶128)

states:

Barsness satisfies this element. *Supra* Section IX.A.2 (element 1.2). In the context of claim 9, Barsness discloses *a first plurality of computer systems* comprising at least a management system 105 and compute nodes 130₁ and 130₂. EX1004, 6:45-49. This first plurality of computer systems is shown in FIG. 1A of Barsness:

(Figure omitted. EX1004, FIG. 1A (highlighting added))

Further, although not shown in FIG. 1A, the first plurality of computer systems further comprises compute nodes spawned by the management system 105 from the pool of computing resources. *Supra* Section IX.A.6 (element 1.6).

I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element. *Supra* Section VII.B.2 (element 1.2). In the context of claim 9, Barsness discloses *a first plurality of computer systems* comprising at least a management system 105, monitoring system 110, and compute nodes 130₁ and 130₂. EX1004, 6:45-49. As FIG. 1A of Barsness shows, this *first plurality of computer systems* includes those systems. EX1004, FIG. 1A. Further, although not shown in FIG. 1A, the *first plurality of computer systems* may also comprise compute nodes spawned by the management system 105 from the pool of computing resources. *Supra* Section VII.B.15 (element 1.6).

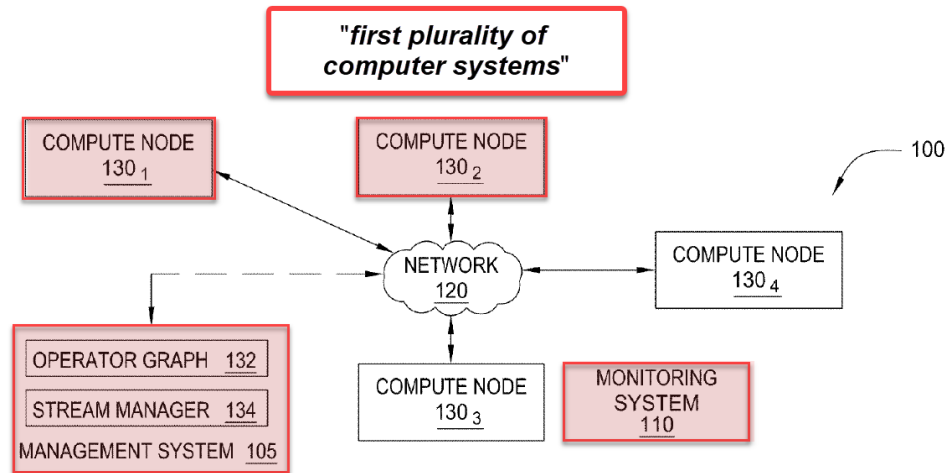


FIG. 1A

EX1004, FIG. 1A (annotations added).

3. **Element [9.3] - wherein each respective computer system of the first plurality of computer systems comprises a respective memory that stores a respective first data that: describes at least a respective portion of a configuration of a first transformation pipeline, and identifies at least a second transformation pipeline,**

134. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶129)

states:

Barsness satisfies this element because each of the management system 105 and compute nodes 130 (*each respective computer system of the first plurality of computer systems*) comprises a memory (*respective memory*) that stores respective programming instructions and application data (*respective first data*). *Supra* Section IX.A.2.a) (element 1.2.a). That data meets the additional recited claim language here. *Supra* Section IX.A.2.b) (element 1.2.b). For example, the operator graph of Barsness describes *a configuration of a first transformation pipeline* because the operator graph represents a stream

application that flows from source PEs to sink PEs, and shows the interconnection of PEs of a compute node applying a first transformation pipeline, as shown in FIG. 1B:

(Figure omitted. EX1004, FIG. 1B (highlighting added))

Id., 6:50-56, 7:52-67 (describing the flow of data tuples through the nodes and processing elements of the example operator graph). In the context of the FIG. 1B example, the management system memory stores operator graph 132 that includes data that *describes at least a respective portion of a configuration of a first transformation pipeline* because it describes the configuration of PE1, PE2, and PE3 (*first transformation pipeline*). EX1004, 10:12-14, 10:48-56; *id.*, FIGS. 1B, 3.

135. I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element because each of the management system 105, monitoring system 110, and compute nodes 130, *each respective computer system of the first plurality of computer systems*, comprises a memory, a *respective memory*, that stores respective programming instructions and application data, the *respective first data*. *Supra* Section VII.B.3 (element 1.2.a). That data meets the additional recited claim language here. *Supra* Section VII.B.4-5 (elements 1.2.b-1.2.c). For example, the operator graph of Barsness describes a configuration of a *first transformation pipeline*, because the operator graph represents a stream application that flows from source processing elements to sink processing elements and shows the interconnection of the processing elements of a compute node applying a *first transformation pipeline*. EX1004, FIG. 1B; *id.*, 6:50-56, 7:52-67. In the FIG. 1B example, the management system memory stores operator graph 132, which includes

data that *describes at least a respective portion of a configuration of a first transformation pipeline* because it describes the configuration of PE1, PE2, and PE3, the *first transformation pipeline*.

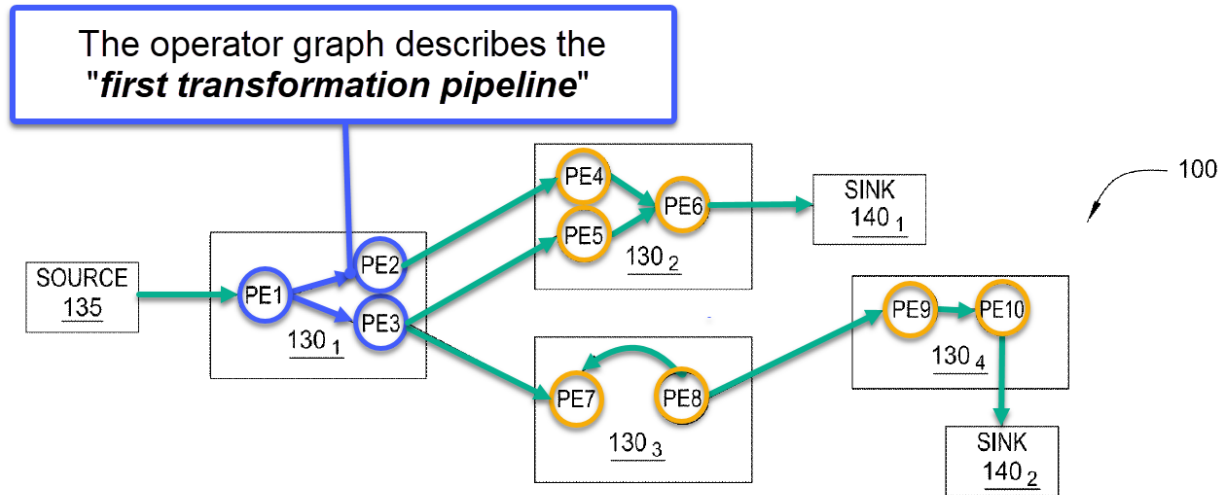


FIG. 1B

EX1004, 10:12-14, 10:48-56; *id.*, FIGS. 1B (annotations added), 3.

136. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶130)

states:

Each compute node of the first plurality (e.g., 130₁ and 130₂) stores “programming instructions,” “application data,” buffered stream data, and other data (*respective first data*) in its respective memory, including processing elements as designated by the operator graph. EX1004, 8:53-59, 8:66-9:33; FIG. 2. This “application data” (*respective first data*) specifies the connection to adjacent compute nodes (e.g., between compute node 130₁ and compute node 130₂), e.g., as the source and destination information of a TCP/IP socket, to transmit data between processing elements “in the stream application as described by the operator graph 132.” EX1004, 9:4-9; EX1017, 1:22-2:26, FIG. 1; EX1016, 8:56-62. Each compute node thus stores data *describing at least a portion of a configuration of the transformation pipelines* that it receives data from and emits data to. The data stored on each compute node (e.g., compute node 130₁) also *describes at least a portion of a*

configuration of the transformation pipeline on that particular compute node. *Supra* Section IX.A.2.b). For example, compute node 130₁ stores the processing elements PE1, PE2, and PE3 comprising the first transformation pipeline.

I agree with Dr. Crovella because, in my opinion, each compute node of the first plurality, such as 130₁ and 130₂, stores “programming instructions,” “application data,” buffered stream data, and other data, the *respective first data*, in its *respective memory*, including processing elements as designated by the operator graph. EX1004, 8:53-59, 8:66-9:33; FIG. 2. This “application data,” the *respective first data*, specifies the connection to adjacent compute nodes, such as between compute node 130₁ and compute node 130₂, for example as the source and destination information of a TCP/IP socket, to transmit data between processing elements “in the stream application as described by the operator graph 132.” EX1004, 9:4-9; EX1017, 1:22-2:26, FIG. 1; EX1016, 8:56-62. Each compute node thus stores data *describing at least a portion of a configuration of the transformation pipelines* from which it receives data and to which it emits data. The data stored on each compute node, such as compute node 130₁, also *describes at least a portion of a configuration of the transformation pipeline* on that particular compute node. *Supra* Section VII.B.5 (element 1.2.c). For example, compute node 130₁ stores processing elements PE1, PE2, and PE3, which comprise the *first transformation pipeline*.

137. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶131)

states:

Thus, in the FIG. 1B example, at least first compute node 130₁ and the compute nodes it emits data to (130₂ and 130₃) store in memory data that *describes at least a portion of a configuration of the first transformation pipeline*. Similarly, if compute node 130₂ is viewed as the second transformation pipeline (*supra* Sections IX.A.3.d) and IX.A.4), then at least that compute node (130₂) and the compute node from which it receives messages (130₁) each has memory storing data that *identifies at least a second transformation pipeline* (the pipeline of compute node 130₂). If compute node 130₃ is viewed as the second transformation pipeline, then it and 130₁ satisfy this element. *Supra* Sections IX.A.2.b) and IX.A.2.c).

I agree with Dr. Crovella because, in my opinion, in the FIG. 1B example, at least the first compute node 130₁ and the compute nodes to which it emits data, 130₂ and 130₃, store in memory data that *describes at least a respective portion of a configuration of a first transformation pipeline*. Similarly, if compute node 130₂ is viewed as the *second transformation pipeline*, *supra* Sections VII.B.11-13 (elements 1.4, 1.4.a, and 1.4.b), then at least that compute node, 130₂, and the compute node from which it receives messages, 130₁, each has memory storing data that *identifies at least a second transformation pipeline*, the pipeline of compute node 130₂.

4. **Element [9.4] - wherein a first computer system of the first plurality of computer systems is configured to: receive a first stream of input data from a first input feed, process the first**

stream of input data substantially in real time by executing software instructions that apply the first transformation pipeline to the first stream of input data to generate first pipeline output messages, process the respective first data to determine an identification of the second transformation pipeline, and transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined identification,

138. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶132)

states:

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

I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element.

Supra Section VII.B.6-10 (elements 1.3-1.3.d). The second “process” and the “transmit” aspects are largely similar to the corresponding claim 1 language and are met for the same reasons. In the Barsness FIG. 1B example, the first compute node

130₁, the *first computer system*, is configured to process the data stored in its memory, *the respective first data*, to identify the destination compute node processing elements, the *second transformation pipeline*, and to transmit its output message stream to that compute node, that is, to *transmit the first pipeline output messages to a second computer system of the first plurality of computer systems in accordance with the determined identification*. *Supra* Section VII.B.10.

5. **Element [9.5] - wherein the second computer system is configured to: receive the first pipeline output messages, and process the first pipeline output messages substantially in real time by executing software instructions that apply the second transformation pipeline to the first pipeline output messages,**

139. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶133) states:

Barsness satisfies these elements. *Supra* Sections IX.A.4 (element 1.4), IX.A.4.a) (element 1.4.a), IX.A.4.b) (element 1.4.b).

I agree with Dr. Crovella because, in my opinion, Barsness satisfies these elements. *Supra* Sections VII.B.11-13 (elements 1.4-1.4.b). A POSITA would have understood the second computer system in Barsness to *receive the first pipeline output messages* and to *process the first pipeline output messages* by applying the

second transformation pipeline, for the same reasons given for elements 1.4, 1.4.a, and 1.4.b.

6. **Element [9.6] - wherein the first and second computer systems are distinct; and**

140. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶134) states:

Barsness satisfies this element. *Supra* Section IX.A.6 (element 1.5).

I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element. *Supra* Section VII.B.14 (element 1.5). A POSITA would have understood that, in Barsness, *the first and second computer systems are distinct*, for the reasons given for element 1.5.

7. **Element [9.7] - a second plurality of computer systems;**

141. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶135) states:

Barsness satisfies this element. *Supra* Section IX.A.6 (element 1.6).

I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element. *Supra* Section VII.B.15 (element 1.6). A POSITA would have understood Barsness to disclose *a second plurality of computer systems*, for the reasons given for element 1.6.

8. **Element [9.8] - wherein a third computer system of the first plurality of computer systems is configured to execute software instructions that cause the second plurality of computer systems to execute at least one of the first transformation pipeline and the second transformation pipeline.**

142. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶136)

states:

Barsness satisfies this element. *Supra* Section IX.A.7 (element 1.7). The fourth compute node discussed above with regard to element 1.7 is part of the second plurality of computer systems.

I agree with Dr. Crovella because, in my opinion, Barsness satisfies this element. *Supra* Section VII.B.16 (element 1.7). The fourth compute node discussed above in connection with element 1.7 is part of the *second plurality of computer systems*. A POSITA would have understood Barsness to disclose a *third computer system* configured to cause the *second plurality of computer systems* to execute at least one of the first and second transformation pipelines, for the reasons given for element 1.7.

- J. **Claim 10 - The distributed computing cluster of claim 9, wherein one or more of the first transformation pipeline and the second transformation pipeline is non-linear.**

143. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶137)

states:

Barsness renders this claim obvious. Barsness renders claim 9 obvious. *Supra* Section IX.F (claim 9). Further, Barsness discloses non-linear transformation pipelines. *Supra* Section IX.B (claim 2).

I agree with Dr. Crovella because, in my opinion, Barsness renders this claim obvious. Barsness renders claim 9 obvious. *Supra* Section VII.I (claim 9). Further, Barsness discloses ***non-linear*** transformation pipelines. *Supra* Section VII.C (claim 2). A POSITA would have understood at least one of the first and second transformation pipelines in Barsness to be ***non-linear***, for the reasons given for claim 2.

K. **Claim 11 - The distributed computing cluster of claim 10, wherein the third computer system of the first plurality of computer systems is configured to: identify a fourth computer system of the first plurality of computer systems, wherein the fourth computer system is not configured to apply the first transformation pipeline to any stream of input data, and cause the fourth computer system to process a second stream of input data received from a second input feed substantially in real time by executing software instructions that apply the first transformation pipeline to the second stream of input data.**

144. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶138)

states:

Barsness renders this claim obvious for the same reasons discussed with regard to claim 3. *Supra* Sections IX.G (claim 10), IX.C (claim 3).

I agree with Dr. Crovella because, in my opinion, Barsness renders this claim obvious for the same reasons discussed for claim 3. *Supra* Sections VII.J (claim 10), VII.D (claim 3). A POSITA would have understood Barsness to render claim 11 obvious for the same reasons it renders claim 3 obvious.

L. **Claim 12 - The distributed computing cluster of claim 11, wherein the third computer system is further configured to execute software instructions that: identify one or more of the respective memories that stores respective data that describes a respective portion of a configuration of the first transformation pipeline, and store in the identified one or more memories information identifying the fourth computer system.**

145. In my opinion, Barsness renders claim 12 obvious for the same reasons as discussed for claims 4 and 6. *Supra* Sections VII.I-K (claim 9-11), VII.E (claim 4), VII.G (claim 6).

VIII. ROUND 2: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF CHAKRADHAR

146. As explained above, Barsness alone renders claims 4, 6, 8, and 12 obvious. *Supra* Section VII. These claims are further obvious in view of Chakradhar, which expressly teaches using route-maps and explains the benefits of doing so.

A. **Chakradhar (EX1005)**

147. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶141) states:

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 assigned to NEC Laboratories America, Inc.

I agree with Dr. Crovella because Chakradhar is a U.S. patent application publication that published September 15, 2016, from an application filed March 14, 2016, and it claims priority to a provisional application filed March 13, 2015 (the "Chakradhar Provisional"). EX1005, Cover. Chakradhar is assigned to NEC Laboratories America, Inc.

148. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶142) states:

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].

I agree with Dr. Crovella because, in my opinion, 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 it gives 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].

149. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶143) states:

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].

I agree with Dr. Crovella because, in my opinion, 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, which enables topology changes without interrupting stream processing. EX1005, [0054], [0122].

B. Chakradhar Is Supported By The Chakradhar Provisional

150. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶144)

states:

As I note below, the Chakradhar Provisional supports (1) at least one claim of Chakradhar, and (2) the portions of Chakradhar relied upon in this declaration.

I agree with Dr. Crovella because, in my opinion, and as I explain below, the Chakradhar Provisional supports (1) at least one claim of Chakradhar and (2) the portions of Chakradhar relied upon in this declaration.

C. The Chakradhar Provisional Supports Claim 1 Of Chakradhar

151. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶145)

states:

The chart below indicates disclosure in the Chakradhar Provisional that provided written description support for each element of claim 1 of Chakradhar.

I agree with Dr. Crovella because, in my opinion, the Chakradhar Provisional provides written description support for each element of claim 1 of Chakradhar. The chart below indicates the relevant disclosure.

Claim 1 of Chakradhar (EX1005)	Written Description Support in Chakradhar Provisional (EX1006)
A stream processing acceleration method employing network-aware routing, said method comprising the computer implemented steps of:	EX1006, 4/22: “Stream processing applications...” ... “Stream processing engines...used to process continuous streams of data...” <i>Id.</i> , 5/22: “...a per-topology controller, which uses novel methods and protocols for dynamic, network-aware routing and on-the-fly topology modifications.”... “These new route-maps are atomically injected into multiple operators, on-the-fly, using a light-weight, fault-tolerant protocol, for fast topology re-optimization.”
<i>Id.</i> , 13/22: “...the proposed MWD approach outperforms the random binary tree by a significant margin (more than two-fold speedup), for different model sizes.”	
<i>Id.</i> , 14/22: “...we focus on heuristics and protocols for fast topology re-optimization, to maintain high stream-throughput at all times.”	
<i>Id.</i> , 14/22: “...high-throughout stream processing...” representing topology link structures using route-maps	<i>Id.</i> , 5/22: “(i) representing topology link structure using route-maps” wherein said route-maps include tuple-routing information and topology structure encoded therein; <i>Id.</i> , 5/22: “...the controller determines efficient route-maps, which encode tuple-routing information and also the topology link structure.” applying, on-the-fly, the route maps into multiple operators using

Claim 1 of Chakradhar (EX1005)	Written Description Support in Chakradhar Provisional (EX1006)
	a topology route-map update method. <i>Id.</i> , 5/22: “...(iii) a light-weight, fault-tolerant protocol for atomic route-map update.” ... “These new route-maps are atomically injected into multiple operators, on-the-fly, using a light-weight, fault-tolerant protocol, for fast topology re-optimization.”
Claim 1 of Chakradhar (EX1005)	Written Description Support in Chakradhar Provisional (EX1006)
<i>Id.</i> , 10/22: “...all operators start using the new route-maps for the same batch.”	
b) The Chakradhar Provisional Supports <u>Relied-Upon Subject Matter Of Chakradhar</u>	
As demonstrated above, the Chakradhar Provisional discloses operators maintaining route-maps that include tuple-routing information and topology structure for directing output messages, i.e., the same technique disclosed in Chakradhar. In the discussion below, I also include citations to the Chakradhar Provisional supporting Chakradhar’s relied-upon disclosure in the mapping of Barsness and Chakradhar to the challenged claims, which demonstrates that the Chakradhar Provisional supports all the relied upon-disclosure in Chakradhar.	

D. Combining Barsness And Chakradhar

152. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶147)

states:

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.

I agree with Dr. Crovella because, in my opinion, and 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.

153. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶148) states:

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.

I agree with Dr. Crovella because, in my opinion, a POSITA implementing a system like the one 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 it 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 that represent 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 that Barsness requires.

154. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶149) states:

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).

I agree with Dr. Crovella because, in my opinion, 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 the fast reconfiguration, scalability, and fault tolerance that 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).

155. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶150) states:

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

I agree with Dr. Crovella because, in my opinion, applying Chakradhar's route-maps to Barsness would have allowed each compute node to store respective data

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

156. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶151) states:

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.

I agree with Dr. Crovella because, in my opinion, this 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.

E. **Claims 4, 6, 8, and 12**

157. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶152) states:

As discussed above in Section IX.A (claim 1), claim 1 is obvious in view of Barsness alone. Claim 1 is further obvious in view of the Barsness-Chakradhar combination. Regarding a **distributed computing cluster** (element [1.1]), Chakradhar discloses “[s]tream processing engines” such as “Storm,” which “is a distributed, real-time computation system” that “makes it possible to process unbounded streams of data.” EX1005, [0042], [0056]; *id.*, FIG. 13 (depicting a “cloud-based stream processing system” with multiple incoming data streams), [0134]; EX1006, 4/22. Chakradhar explains that “[o]n a storm *cluster*, topologies are executed and process streams of tuples (i.e., data).” EX1005, [0056] (emphasis added); *see also id.*, [0046] (“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. These disclosures complement the distributed computing architecture of Barsness, and thus, in combination, the Barsness-Chakradhar combination includes a *distributed computing cluster*.

I agree with Dr. Crovella because, in my opinion, as discussed above in Section VII.B (claim 1), claim 1 is obvious in view of Barsness alone, and claim 1 is further obvious in view of the Barsness-Chakradhar combination. As to a **distributed computing cluster** (element [1.1]), Chakradhar discloses “[s]tream processing

engines” such as “Storm,” which “is a distributed, real-time computation system” that “makes it possible to process unbounded streams of data.” EX1005, [0042], [0056]; *id.*, FIG. 13, [0134]; EX1006, 4/22. Chakradhar explains that, “[o]n a storm cluster, topologies are executed and process streams of tuples,” EX1005, [0056], and that cluster resources are orchestrated by a global cluster scheduler, that route-maps are installed on a running cluster, and that its techniques were evaluated on a 30-node cluster. EX1005, [0046], [0069], [0098], [0148]; EX1006, 4/22, 6-7/22, 11/22. These disclosures complement the distributed computing architecture of Barsness, so the Barsness-Chakradhar combination includes a ***distributed computing cluster***.

158. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶153) states:

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

I agree with Dr. Crovella because, in my opinion, as to a ***distributed computational graph*** (element [1.2.b]), just as the operator graph of Barsness “represents a stream application,” EX1004, 6:49-53, Chakradhar explains that “stream engines code an application workflow as a directed acyclic graph (DAG) of operators, referred to as a topology.” EX1005, [0043]; EX1006, 4/22. Chakradhar further discloses that stream data is “processed in a pipelined fashion as they traverse through the topology.” EX1005, [0043]; EX1006, 4/22. Chakradhar also 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]; *id.*, [0106]; EX1006, 5/22, 8/22. A POSITA would have understood these topologies to be a ***distributed computational graph***.

159. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶154) states:

Regarding **the respective first data [that] represents a respective portion of a distributed computational graph** (element [1.2.b]), Chakradhar 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.

I agree with Dr. Crovella because, in my opinion, as to *the respective first data [that] represents a respective portion of a distributed computational graph* (element [1.2.b]), Chakradhar discloses that “[e]very operator chooses a destination operator for its outgoing tuples based on route-maps.” EX1005, [0069]; *id.*, [0070]; 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. And 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, and, when updated, are “injected” into the operators “for fast topology re-optimization.” EX1005, [0010], [0054]; EX1006, 5/22, 14/22.

160. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶155) states:

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

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

161. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶156) states:

Regarding the first computer system processing the respective first data to determine information about the second transformation pipeline (element [1.3.c]) and transmitting the first pipeline output messages to a second computer system in accordance with the determined information (element [1.3.d]), Chakradhar teaches that “[e]ach operator maintains a route-map that specifies the routes on which *output messages* should be sent after processing incoming messages.” EX1005, [0122] (emphasis added); EX1006, 9/22. In combination, an operator and its route-map are stored in the memory of the first computer system of Barsness-Chakradhar, which enables the processing element / operator to emit / send its output messages to the appropriate downstream PE as described by the operator graph. *E.g.*, EX1004, FIG. 2 (showing operator in memory of compute node). Specifically, such a route-map “encode[s] tuple-routing information and also the topology link structure,” as taught by Chakradhar. EX1005, [0010], [0054]; *id.*, [0043]; EX1006, 5/22, 9/22. In this way, “[e]very operator chooses a destination operator for its outgoing tuples based on route-maps.” EX1005, [0069]; EX1006, 6/22. The first compute node, when executing the operator of a given processing element at the end of the first transformation pipeline (e.g., PEs 2 and 3 in example operator graph of Barsness FIG. 1B), thus processes the route-map (*the respective first data stored in the memory of the first computer system*)

to determine the destination of its output message (*to determine information about the second transformation pipeline*). The operators of Barsness-Chakradhar use route-maps to identify the destination of their output messages, and thus the first compute node (*first computer system*) transmits the output of the first compute node (*the first pipeline output messages*) to the second computer system according to the destination(s) specified in the route-map(s) (*the determined information*).

I agree with Dr. Crovella because, in my opinion, as to the first computer system processing the respective first data to determine information about the second transformation pipeline (element [1.3.c]) and transmitting the first pipeline output messages to a second computer system in accordance with the determined information (element [1.3.d]), Chakradhar teaches that “[e]ach operator maintains a route-map that specifies the routes on which output messages should be sent after processing incoming messages.” EX1005, [0122]; EX1006, 9/22. In combination, an operator and its route-map are stored in the memory of the first computer system of Barsness-Chakradhar, which enables the processing element or operator to send its output messages to the appropriate downstream processing element as described by the operator graph. E.g., EX1004, FIG. 2. Specifically, such a route-map “encode[s] tuple-routing information and also the topology link structure,” as taught by Chakradhar. EX1005, [0010], [0054]; id., [0043]; EX1006, 5/22, 9/22. In this way, “[e]very operator chooses a destination operator for its outgoing tuples based on route-maps.” EX1005, [0069]; EX1006, 6/22. The first compute node, when executing the operator of a given processing element at the end of the first

transformation pipeline, such as PE2 and PE3 in the example operator graph of Barsness FIG. 1B, thus processes the route-map, the respective first data stored in the memory of the first computer system, to determine information about the second transformation pipeline. The operators of Barsness-Chakradhar use route-maps to identify the destination of their output messages, and so the first compute node, the first computer system, transmits the first pipeline output messages to a second computer system in accordance with the determined information.

162. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶157) states:

Thus, claim 1 is obvious in view of the Barsness-Chakradhar combination. Claims 2, 3, 5, 7, 9-11, and 13 are obvious for the same reasons set forth above and in Ground 1. *Supra* Section IX.

I agree with Dr. Crovella because, in my opinion, claim 1 is obvious in view of the Barsness-Chakradhar combination. Further, Claims 4, 6, 8, and 12 are obvious for the same reasons set forth above and in Ground 1.

IX. GROUND 3: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF SIRIPURAPU

163. As explained above, Barsness alone renders claims 4, 6, 8, and 12 obvious. *Supra* Section VII. These claims are further obvious in view of Siripurapu, which expressly teaches adding computer nodes to handle processing additional real-time data streams and increased data flow.

A. **Siripurapu (EX1007)**

164. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶159)

states:

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.

I agree with Dr. Crovella because Siripurapu is a U.S. patent application publication that published February 27, 2014, and it claims priority to a provisional application filed November 18, 2010. EX1007, Cover.

165. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶160)

states:

Siripurapu is directed to the same field of processing real-time data streams with a “distributed, elastic architecture” and provides an explicit mechanism for handling “additional data streams” and “increased data stream flows.” EX1007, [0060], [0453]; *id.*, [0359]-[0363]. Specifically, Siripurapu teaches that the “distributed implementation provides scalability, 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]. Siripurapu explains that “[a]t any point in time, there are usually several Streams: some of them external (e.g., Twitter, Facebook) and others that are generated as intermediate values and outputs by the application.” EX1007, [0178]; *id.*, [0439]-[0440] (describing various “[i]nput real-time data streams”).

I agree with Dr. Crovella because, in my opinion, Siripurapu is directed to the same field of processing real-time data streams with a “distributed, elastic architecture” and provides an explicit mechanism for handling “additional data streams” and

“increased data stream flows.” EX1007, [0060], [0453]; *id.*, [0359]-[0363]. Specifically, Siripurapu teaches that the “distributed implementation provides scalability, 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]. Siripurapu also explains that “[a]t any point in time, there are usually several Streams: some of them external (e.g., Twitter, Facebook) and others that are generated as intermediate values and outputs by the application.” EX1007, [0178]; *id.*, [0439]-[0440] (describing various “[i]nput real-time data streams”).

166. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶161) states:

Further, “[t]he distributed implementation also provides fault-tolerance so that there is no single point of failure, wherein a task at a failed worker node may be assigned to a different worker node without affecting the entire distributed system.” EX1007, [0454]. Siripurapu also discloses assigning a stream to another compute node if the computational load of a compute node is too high, for example due to “hot spots in the flow of incoming stream events.” EX1007, [0470].

I agree with Dr. Crovella because, in my opinion, Siripurapu further teaches that “[t]he distributed implementation also provides fault-tolerance so that there is no single point of failure, wherein a task at a failed worker node may be assigned to a different worker node without affecting the entire distributed system.” EX1007, [0454]. Siripurapu also discloses assigning a stream to another compute node when

the computational load of a compute node is too high, for example due to “hot spots in the flow of incoming stream events.” EX1007, [0470].

167. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶162) states:

As depicted in Figure 36, multiple “real-time data streams” arrive at a stream bus and are distributed to worker nodes by a conductor that manages the real-time processing of stream data by “act[ing] as a task scheduling mechanism to provide instructions to designate worker nodes for performing ... operations.” EX1007, [0444], [0446]; *id.*, [0448]-[0449] (explaining how the conductor monitors “performance of the worker nodes” and “perform[s] load balancing by reassigning tasks to the worker nodes”). For example, “[t]ask scheduling may be performed by the conductor in a manner to balance the loads among the worker nodes, prevent ‘hot spots’ from arising at any worker node and to reduce the response time required by each worker node to generate an output.” EX1007, [0481].

(Figure omitted. EX1007, FIG. 36)

I agree with Dr. Crovella because, in my opinion, as depicted in Figure 36, multiple “real-time data streams” arrive at a stream bus and are distributed to worker nodes by a conductor that manages the real-time processing of stream data by “act[ing] as a task scheduling mechanism to provide instructions to designate worker nodes for performing ... operations.” EX1007, [0444], [0446]; *id.*, [0448]-[0449] (explaining that the conductor monitors “performance of the worker nodes” and “perform[s] load balancing by reassigning tasks to the worker nodes”). For example, “[t]ask scheduling may be performed by the conductor in a manner to balance the loads among the worker nodes, prevent ‘hot spots’ from arising at any worker node and to

reduce the response time required by each worker node to generate an output.”

EX1007, [0481].

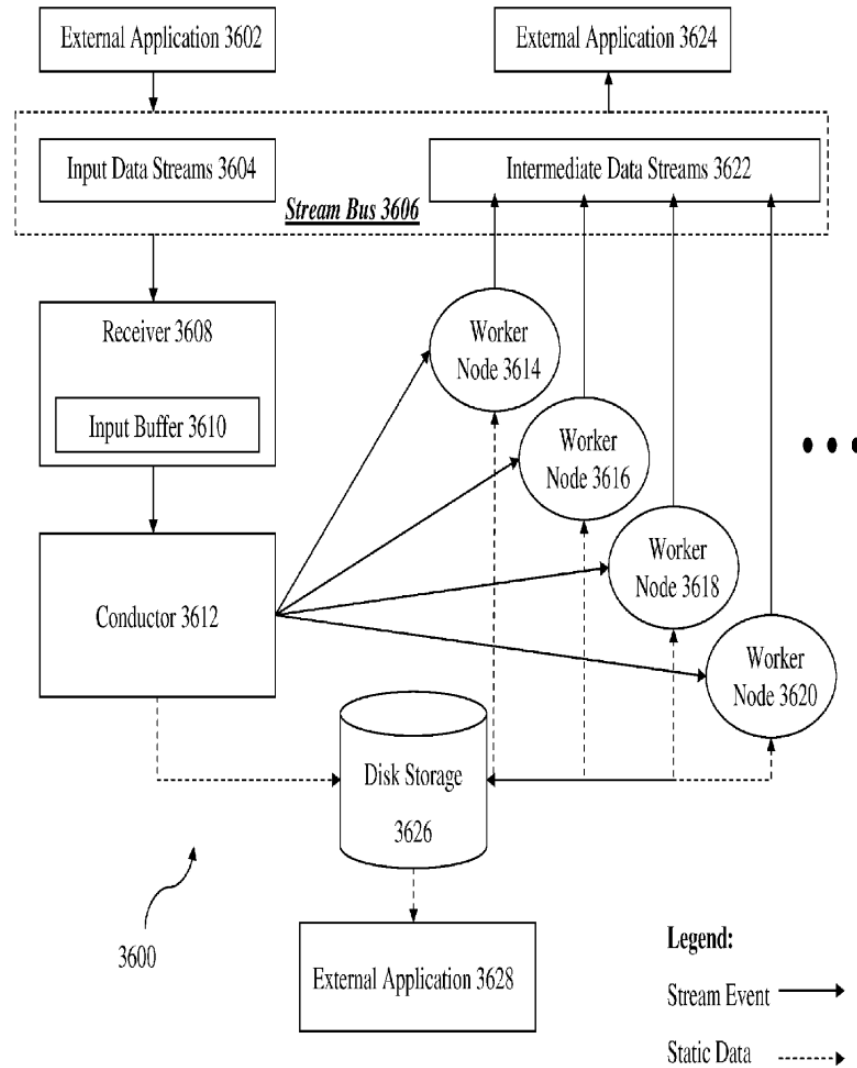


Figure 36

EX1007, FIG. 36.

B. Combining Barsness And Siripurapu

168. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶163)

states:

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 scalability teachings of Siripurapu to beneficially “accommodate increased computational requirements, additional data streams, increased data stream flows, and the like. EX1007, [0453].

I agree with Dr. Crovella because, in my opinion, 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 scalability teachings of Siripurapu to beneficially “accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” EX1007, [0453].

169. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶164)

states:

Barsness is expressly concerned with scaling a stream-processing application across multiple compute nodes and generally describes the flexibility and adaptability of the distributed implementation to address data processing requirements. EX1004, 3:19-22 (“Scalability is achieved by distributing an application across nodes ...”), 6:53-55 (PEs can be added or removed from operator graph “based on the data processing needs and scalability”). Siripurapu addresses the same practical scaling problem in a stream processing context, and provides a complementary, implementation-level rationale for scaling capacity when workload grows, where “one or more additional worker nodes” are dynamically added “to accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” EX1007, [0453]. A POSITA reading Barsness and Siripurapu together would understand Siripurapu’s additional worker nodes for additional data streams as directly aligning with the scalable infrastructure of Barsness that distributes execution of operators across

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

I agree with Dr. Crovella because, in my opinion, Barsness is expressly concerned with scaling a stream-processing application across multiple compute nodes and generally describes the flexibility and adaptability of the distributed implementation in addressing data processing requirements. EX1004, 3:19-22 (“Scalability is achieved by distributing an application across nodes ...”), 6:53-55 (processing elements can be added or removed from the operator graph “based on the data processing needs and scalability”). Siripurapu addresses the same practical scaling problem in a stream-processing context and provides a complementary, implementation-level rationale for scaling capacity when the workload grows, where “one or more additional worker nodes” are dynamically added “to accommodate increased computational requirements, additional data streams, increased data stream flows, and the like.” EX1007, [0453]. A POSITA reading Barsness and Siripurapu together would have understood Siripurapu’s additional worker nodes for additional data streams to align directly with the scalable infrastructure of Barsness, which distributes execution of operators across compute nodes and can add or remove processing elements for “data processing needs and scalability.” EX1004, 6:53-55.

170. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶165) states:

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

I agree with Dr. Crovella because, in my opinion, 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 or provisioning more compute nodes as needed (EX1004, 6:19-42, 10:26-39). Like the management system of Barsness, the conductor of Siripurapu “perform[s] load balancing by reassigning tasks to the worker nodes” (EX1007, [0448]) and “assign[s] stream processing operations to the new worker nodes” provisioned to “accommodate increased computational requirements, additional data streams, increased data stream flows, and the like” (EX1007, [0453]).

171. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶166) states:

Further, Siripurapu describes how a “distributed computational system” for providing real-time stream processing handles receiving “one or more real-time data streams” and distributing them to worker nodes executing operators. EX1007, [0439]; *id.*, [0440]-[0460] (further describing the elements of the distributed computational system). A POSITA implementing Barsness thus would have been motivated to combine the teachings of Siripurapu for handling additional input data

streams and increased data stream flows, to thus ensure the scalability of the stream application when handling “the sheer volume of content available and the speed with which new content is made available.” EX1007, [0002]; *id.*, [0352] (explaining “spikiness in the flow of stream events in the input data streams” where “a particular topic or link may suddenly become ‘hot’ or popular in an input data stream, resulting in a sudden surge ...”).

I agree with Dr. Crovella because, in my opinion, Siripurapu describes how a “distributed computational system” for providing real-time stream processing handles receiving “one or more real-time data streams” and distributing them to worker nodes executing operators. EX1007, [0439]; *id.*, [0440]-[0460]. A POSITA implementing Barsness thus would have been motivated to combine the teachings of Siripurapu for handling additional input data streams and increased data stream flows, so as to ensure the scalability of the stream application when handling “the sheer volume of content available and the speed with which new content is made available.” EX1007, [0002]; *id.*, [0352] (explaining “spikiness in the flow of stream events in the input data streams” where “a particular topic or link may suddenly become ‘hot’ or popular in an input data stream, resulting in a sudden surge ...”).

172. Dr. Mark Crovella’s declaration in IPR2026-00298 (Ex. 1003, ¶167) states:

Further, as Barsness already discloses a scalable cloud-based computing infrastructure, where compute nodes can be dynamically provisioned to handle data processing needs, a POSITA would have reasonably expected to succeed combining the teachings of Siripurapu with Barsness. EX1004, 6:16-42; *id.*, 3:19-22.

I agree with Dr. Crovella because, in my opinion, because Barsness already discloses a scalable cloud-based computing infrastructure in which compute nodes can be dynamically provisioned to handle data processing needs, a POSITA would have reasonably expected to succeed in combining the teachings of Siripurapu with Barsness. EX1004, 6:16-42; *id.*, 3:19-22.

173. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶168) states:

This obvious combination of Barsness and Siripurapu includes the features of the cloud-based cluster for distributed stream data processing applications taught by Barsness, as described *supra* Section IX, with the stream application scaling via the addition of compute nodes to accommodate additional data streams as taught by Siripurapu. EX1007, [0453]. These additional compute nodes would include the additional compute nodes I depicted in the annotated FIG. 1B of Barsness above in Sections IX.A.7, IX.C, and IX.D. So implemented, the Barsness-Siripurapu combination renders all challenged claims obvious.

I agree with Dr. Crovella because, in my opinion, this combination of Barsness and Siripurapu includes the features of the cloud-based cluster for distributed stream data processing applications taught by Barsness, as described *supra* Section VII, with the stream application scaling through the addition of compute nodes to accommodate additional data streams as taught by Siripurapu. EX1007, [0453]. These additional compute nodes would include the additional compute nodes I depicted in the annotated FIG. 1B of Barsness above regarding claim 1. So implemented, the Barsness-Siripurapu combination renders all challenged claims obvious.

C. **Claims 4, 6, 8, and 12**

174. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶169)

states:

As discussed above in Section IX.A (claim 1), claim 1 is obvious in view of Barsness alone. Claims 1, 2, 7, 9, 10, and 13 are further obvious in view of the Barsness-Siripurapu combination, as a POSITA would have found it obvious to implement the management system of Barsness to provision additional compute node(s) to handle increased data volume / processing needs at a given compute node/transformation pipeline, as taught by Barsness and Siripurapu. EX1007, [0453];]; *id.*, [0352] (explaining that teachings “are able to react in a fast and efficient manner to surges in input data streams, and to balance computational loads relating to the ‘hot spots’ so that the data structures and nodes are still able to operate efficiently and generate output in short response times”); *supra* Sections IX.A.7 (element 1.7), IX.B (claim 2), IX.E (claim 7), IX.F (claim 9), IX.G (claim 10), IX.I (claim 13).

I agree with Dr. Crovella because, in my opinion, as discussed above claim 1 is obvious in view of Barsness alone. *Supra* Section VII.B. Claims 4, 6, 8, and 12 are further obvious in view of the Barsness-Siripurapu combination, because a POSITA would have found it obvious to implement the management system of Barsness to provision additional compute nodes to handle increased data volume or processing needs at a given compute node or transformation pipeline, as taught by Barsness and Siripurapu. EX1007, [0453]; *id.*, [0352] (explaining that the teachings “are able to react in a fast and efficient manner to surges in input data streams, and to balance computational loads relating to the ‘hot spots’ so that the data structures and nodes are still able to operate efficiently and generate output in short response times”);

supra Sections VII.B.16 (element 1.7), VII.C (claim 2), VII.I (claim 9), VII.J (claim 10).

X. GROUND 4: CLAIMS 4, 6, 8, AND 12 ARE OBVIOUS OVER BARSNESS IN VIEW OF CHAKRADHAR AND SIRIPURAPU

175. Dr. Mark Crovella's declaration in IPR2026-00298 (Ex. 1003, ¶171)

states:

As explained above, Barsness alone renders claims 1-3, 5, 7, 9-11, and 13 obvious. *Supra* Section IX. 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 X, XI), 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 relate 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.

I agree with Dr. Crovella, and, in my opinion, as explained above, Barsness alone renders claims 4, 6, 8, and 12 obvious. *Supra* Section VII. 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 and IX), a POSITA would have applied those teachings together when implementing Barsness. Those teachings relate to different aspects and are fully compatible with,

and complementary to, one another: Chakradhar's teachings relate to storing graph information on nodes to ensure the proper and efficient downstream transmission of transformed data, and Siripurapu provides details on the scalability of the system to handle different workload needs. It is my opinion that a POSITA would have been motivated, with a reasonable expectation of success, to implement both simultaneously within the Barsness system. The challenged claims are therefore further obvious in view of a combination of Barsness, Chakradhar, and Siripurapu.

XI. SECONDARY CONSIDERATIONS OF NON-OBVIOUSNESS

176. I am not aware of any secondary considerations, or so-called "objective indicia of non-obviousness" for the challenged claims.

177. Secondary considerations could include things like commercial success of a product due to the merits of the claimed invention; a long felt need for the solution provided by the claimed invention; unsuccessful attempts by others to find the solution provided by the claimed invention; copying of the claimed invention by others; unexpected and superior results from the claimed invention; acceptance by others of the claimed invention as shown by praise from others in the field or from the licensing of the claimed invention; teaching away from the conventional wisdom in the art at the time of the invention; independent invention of the claimed invention by others before or at about the same time as the named inventor thought of it; and other evidence tending to show obviousness.

178. As discussed throughout, the claimed features reflect predictable results of combining known elements according to their known functions, and thus, there are no unexpected and superior results from the claimed invention.

179. Further, even if Patent Owner were to assert that secondary considerations exist, given the strong reasons that the challenged claims are obvious in light of the prior art, including the motivations to combine the references set forth above, I do not believe any such secondary considerations would rise to the level of overcoming the invalidity opinions I have expressed.

180. I understand that Patent Owner may address the issue of secondary considerations more fully in the future. I reserve the right to respond to any specific bases for such secondary considerations that Patent Owner may identify.

XII. CONCLUSION

181. For the reasons set forth above, I believe claims 4, 6, 8 and 12 of the '424 patent are unpatentable in view of the prior art. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

182. I declare that all statements made herein of my knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: July 20, 2026

By: 
Paul S. Min, Ph.D.