

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re *Inter Partes* Review of:)
U.S. Patent No. 9,146,777)
Issued: Sep. 29, 2015)
Application No.: 13/750,696)
Filing Date: Jan. 25, 2013)

For: **Parallel Processing With Solidarity Cells By Proactively Retrieving From
A Task Pool A Matching Task For The Solidarity Cell To Process**

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 9,146,777**

TABLE OF CONTENTS

I.	Introduction.....	1
II.	Identification of Challenges (37 C.F.R. § 42.104(b))	1
III.	Background.....	2
	A. State of the Art	2
	B. The '777 Patent (Ex. 1001)	4
	C. Prosecution History	5
	D. Person of Ordinary Skill in the Art	6
	E. Claim Construction	6
	1. Solidarity Cell	7
	2. Thread	7
IV.	Ground 1: Claims 1-10 and 12-14 are unpatentable over Lurie and Ethernet Specification.....	8
	A. Overview of the Prior Art.....	9
	1. Lurie (Ex. 1007).....	9
	2. Ethernet Specification (Ex. 1009).....	11
	3. Andrews (Ex. 1011)	12
	B. Independent Claim 14	13
	1. Preamble.....	14
	2. Elements 14[a]-[b]: CPU and task pool.....	14
	3. Elements 14[c.1], [d.1], [g]: solidarity cells and their agents.....	17
	4. Elements 14[c.2], [d.2], [h].....	20
	a. Element 14[h.1]: dispatch agent to task pool	24

b.	Element 14[h.2]: search the task pool	27
c.	Element 14[h.3]: return to cell.....	31
5.	Element 14[e]: matching task in task pool.....	32
6.	Element 14[f]: each cell sends agent when not processing a task.....	32
C.	Independent claim 1	38
1.	Preamble.....	38
2.	Elements 1[a]-[b] and [d]: CPU populates the task pool.....	38
3.	Element 1[e]: tasks have a task descriptor.....	41
4.	Elements 1[c.1]-[c.2] and [f]-[g]: first solidarity cell with agent retrieves task.....	42
D.	Dependent claims 2-10, 12-13	44
1.	Claim 2: thread completion notification	44
2.	Claim 3: task types.....	45
3.	Claims 4-7: matching task ready to be processed.....	45
4.	Claim 8: task descriptor data storage location	46
5.	Claim 9: task descriptor is a data structure	47
6.	Claim 10: task pool in physical memory	48
7.	Claim 12: second solidary cell and agent	48
8.	Claim 13: send agent when no task to process	48
V.	Ground 2: Claim 11 is unpatentable over Lurie, Ethernet Specification, and Lippett.....	49
A.	Overview of Lippett (Ex. 1031).....	49
B.	Claim 11: task pool in a dedicated hardware block	49
VI.	Secondary Considerations	50

VII.	The Board Should Reach the Merits of This Petition	50
VIII.	Mandatory Notices under 37 C.F.R. § 42.8.....	51
A.	Real Parties-in-Interest	51
B.	Related Matters.....	52
C.	Grounds for Standing	53
D.	Lead and Backup Counsel and Service Information.....	53
E.	Fee for <i>Inter Partes</i> Review	54
IX.	Conclusion	54

TABLE OF AUTHORITIES

Page(s)

CASES

<i>KSR Int’l Co. v. Teleflex Inc.</i> , 550 U.S. 398 (2007).....	36
<i>Uber Techs., Inc. v. X One, Inc.</i> , 957 F.3d 1334 (Fed. Cir. 2020)	36

REGULATIONS

37 C.F.R.	
§ 42.8.....	51
§ 42.8(b)(3)	53
§ 42.8(b)(4)	53
§ 42.10(a)	53
§ 42.10(b)	54
§ 42.15(a)	54
§ 42.102(b).....	<i>passim</i>
§ 42.103.....	1, 2, 8
§ 42.104(b).....	1

Exhibit List

Ex.	Description
1001	U.S. Patent No. 9,146,777 B2 (the “777 Patent”)
1002	File History for U.S. Patent No. 9,146,777 (“777 FH”)
1003	Declaration of David Lowenthal, Ph.D. (“Lowenthal Decl.”)
1004	<i>Curriculum Vitae</i> of David Lowenthal, Ph.D.
1005	Declaration of Ingrid Hsieh-Yee, Ph.D. (“Hsieh-Yee Decl.”)
1006	<i>Curriculum Vitae</i> of Ingrid Hsieh-Yee, Ph.D.
1007	U.S. Patent Application Publication No. 2007/0124363 A1 (“Lurie”)
1008	U.S. Patent Application Publication No. 2005/0132380 A1 (“Chow”)
1009	Digital Equip. Corp., Intel Corp. & Xerox Corp., <i>The Ethernet – A Local Area Network, Data Link Layer and Physical Layer Specifications, Version 1.0</i> (Sept. 30, 1980) (“Ethernet Specification”)
1010	RESERVED
1011	Gregory R. Andrews, FOUNDATIONS OF MULTITHREADED, PARALLEL, AND DISTRIBUTED PROGRAMMING (2000) (excerpts) (“Andrews”)
1012	Jyoti Batheja & Manish Parashar, <i>Adaptive Cluster Computing Using JavaSpaces</i> , PROC. 2001 IEEE INT’L CONF. CLUSTER COMPUTING (Oct. 8, 2001) (“Batheja”)
1013	U.S. Patent Application Publication No. 2007/0074207 A1 (“Bates”)
1014	Swarm Technology, LLC’s First Request for Production of Documents to Defendants Amazon.com, Inc. and Amazon Web Services, Inc., <i>Swarm Tech., LLC v. Amazon.com, Inc.</i> , No. 2:21-cv-00438 (D. Ariz. May 25, 2021) (“Swarm RFPs”)
1015	RESERVED

Ex.	Description
1016	Steve Tang, <i>Advantages of Ethernet vs. SONET/SDH</i> (TC Communications White Paper (rev. 2021) (“Tang”))
1017	Matt Hamblen, <i>Q&A: Ethernet’s Creator Describes Its Past, Future</i> , COMPUTERWORLD (May 16, 2003, 12:00 AM PST), https://www.computerworld.com/article/2570092/q-a--ethernet-s-creator-describes-its-past--future.html (“Hamblen”)
1018	Information Scis. Inst., Univ. of S. Cal., <i>Internet Protocol – DARPA Internet Program Protocol Specification</i> , RFC 791 (Jon Postel ed., Sept. 1981) (“Internet Protocol”)
1019	J. Postel, <i>User Datagram Protocol</i> , RFC 768, Info. Scis. Inst. (Aug. 28, 1980) (“UDP Specification”)
1020	RESERVED
1021	Petition for <i>Inter Partes</i> Review of U.S. Patent No. 9,146,777, <i>Juniper Networks, Inc. v. Swarm Tech. LLC</i> , IPR2021-01317, Paper 1 (P.T.A.B. July 26, 2021) (“Juniper Petition”)
1022	RESERVED
1023	John D. Day & Hubert Zimmermann, <i>The OSI Reference Model</i> , 71 PROC. IEEE 1334 (Dec. 1983) (“Day”)
1024	Douglas Meyer & George Zobrist, <i>TCP/IP Versus OSI: The Battle of the Network Standards</i> , 9 IEEE Potentials 16 (Feb. 1990) (“Meyer”)
1025 – 1026	RESERVED
1027	Hugh C. Lauer & Roger M. Needham, <i>On the Duality of Operating System Structures</i> , PROC. 2D INT’L SYMP. IN OPERATING SYS. REV., IRIA (Oct. 1978), reprinted in 13 OPERATING SYS. REV. 3 (Apr. 2, 1979) (“Lauer”)
1028	Nicholas John Carriero, Jr., <i>Implementation of Tuple Space Machines</i> , RSCH. REP. YALEU/DCS/RR-567 (Dec. 1987) (“Carriero”)
1029	U.S. Patent No. 9,852,004 B2 (the “’004 Patent”)

Ex.	Description
1030	U.S. Patent No. 10,592,275 B2 (the “’275 Patent”)
1031	U.S. Patent Application Publication No. 2007/0220517 A1 (“Lippett”)
1032	Decision Denying Institution of <i>Inter Partes</i> Review, <i>Juniper Networks, Inc. v. Swarm Tech. LLC</i> , IPR2021-01317, Paper 15 (P.T.A.B. Feb. 1, 2022) (“Juniper Decision”)
1033	U.S. Patent 9,146,777 Claim Chart, <i>Swarm Tech. LLC v. Amazon.com Inc.</i> , No. 2:21-cv-00438 (D. Ariz. Aug. 26, 2022), ECF No. 102-7 (“’777 Claim Chart”)
1034	Order, <i>Swarm Tech. LLC v. Amazon.com Inc.</i> , No. 2:21-cv-00438 (D. Ariz. Sept. 20, 2022), ECF No. 111 (“Second Stay Order”)
1035 – 1050	RESERVED
1051	The American Heritage Dictionary (5th ed. 2012) (excerpts) (“American Heritage Dictionary”)
1052	IEEE 100: The Authoritative Dictionary of IEEE Standards Terms (2000) (excerpts) (“IEEE Dictionary”)
1053 – 1056	RESERVED
1057	<i>MATLAB – The Language of Technical Computing: Computation, Visualization, Programming</i> , LANGUAGE REFERENCE MANUAL, VERSION 5 (1997) (excerpts) (“MATLAB Reference”)
1058	Kurt Mehlhorn & Peter Sanders, <i>Hash Tables and Associative Arrays</i> , Algorithms and Data Structures: The Basic Toolbox 81 (2008) (excerpt) (“Mehlhorn”)
1059	RESERVED
1060	<i>The OSI Model – What It Is; Why It Matters; Why It Doesn’t Matter</i> , printed from https://www.tech.faq.com/osi-model.html (last visited Dec. 13, 2022)

Ex.	Description
1061	<i>OSI Model: 7 Layers Explained in Computer Network – Javatpoint</i> , https://www.javatpoint.com/osi-model (last visited Dec. 13, 2022)

I. Introduction

Amazon.com, Inc. and Amazon Web Services, Inc. (collectively “Petitioner” or “Amazon”) request *inter partes* review of claims 1-14 of U.S. Patent No. 9,146,777 (the “’777 Patent”) (Ex. 1001), assigned to Swarm Technology LLC (“Patent Owner” or “Swarm”).

The ’777 patent relates to “processing information in parallel us[ing] autonomous computer processing cells to perform tasks needed by a central processing unit.” ’777 Patent Abstract. The claims generally recite a parallel processing system comprising a CPU that populates a task pool with tasks, and multiple “solidarity cells” (i.e., worker processors) that proactively retrieve and process tasks from the task pool using an agent, generate resulting data, and update the task pool to reflect task completion. *See, e.g.*, ’777 Patent cls. 1, 14.

None of these elements is new, nor is their claimed arrangement. Parallel processing systems with multiple processing cells that retrieve and complete tasks without communicating with the central CPU were well known in the prior art before the filing date of the ’777 patent. Accordingly, Amazon asks the Board to institute review and find all claims unpatentable.

II. Identification of Challenges (37 C.F.R. § 42.104(b))

- Claims 1-10 and 12-14 would have been obvious under § 103 over Lurie (Ex. 1007) and Ethernet Specification (Ex. 1009).

- Claim 11 would have been obvious under § 103 over Lurie, Ethernet Specification, and Lippett (Ex. 1031).

III. Background

A. State of the Art

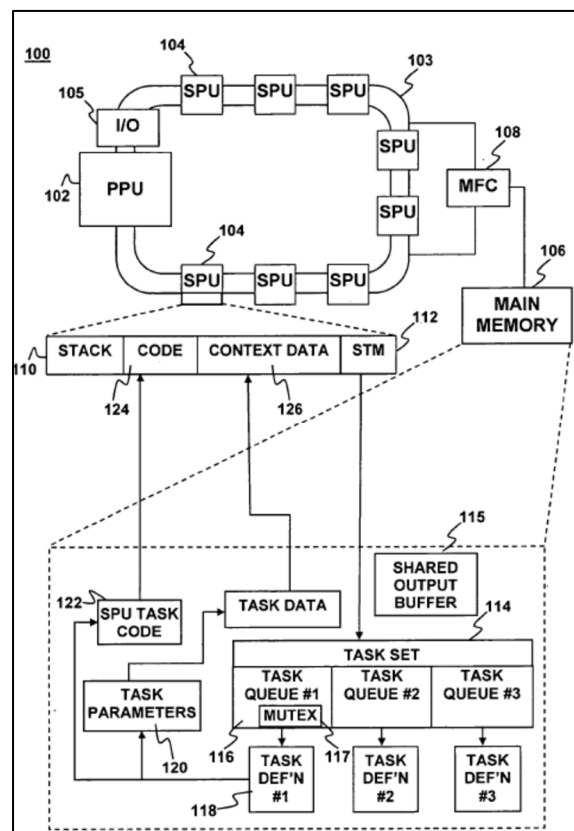
Since the 1960s, parallel processing systems have used multiple processors to accomplish tasks that were otherwise too time-consuming, complex, or specialized for a single processor. Andrews (Ex. 1011) at v-vii, 2-3, 31-32¹; Lowenthal Decl. (Ex. 1003) ¶39. The simplest ones use a central task scheduler that assigns tasks to individual processors. Lowenthal Decl. ¶40. In such architectures, however, the central unit often became a bottleneck. *Id.* Thus, there was a recognized need in the art to relieve the central unit while still coordinating task distribution among processors, and minimizing the time processors spent idle, waiting for a task, or flooded with too many tasks. *Id.*

Later systems used a parallel-processing architecture, known as a “bag-of-tasks,” whereby each worker process searches for, retrieves, and completes tasks from a shared task “bag,” or pool. Andrews 132-39; Lowenthal Decl. ¶41. When executing tasks, “bag-of-tasks” workers may produce smaller tasks that are deposited in the pool for further processing. Andrews 134-35; Ex. 1028 at 44-47;

¹ Citations to Andrews reflect its internal pagination.

Lowenthal Decl. ¶¶42-43. “Bag-of-tasks” systems communicate via shared memory or by passing messages between processors. Andrews 2; Ex. 1027 at 4; Lowenthal Decl. ¶44. In message-passing systems, information such as an index, name, or function pointer can be “passed by reference” to a data structure through messages. Ex. 1027 at 5-8; Lowenthal Decl. ¶¶44-46.

For example, Bates used a power processing unit (“PPU”) that populated its “task queue” while synergistic processing units (“SPUs”) retrieved tasks from the queue for execution:



Ex. 1013 Fig. 1, [0032]-[0041]; Lowenthal Decl. ¶47. Processors retrieved tasks over a (wired or wireless) network using data frames or network packets (“agents”

in the '777 patent) using common networking communication protocols.

Lowenthal Decl. ¶¶48-49; Andrews 31-33.

B. The '777 Patent (Ex. 1001)

The '777 patent's disclosure is significantly different from that of its children, the '004 and '275 patents challenged in earlier-filed IPR2022-00283 and IPR2022-00633. Indeed, it has four fewer figures and seven fewer columns of disclosure. *Compare* '777 Patent with, e.g., '275 Patent (Ex. 1030). But like the '275 and '004 patents, the '777 patent generally describes a parallel processing architecture using a central processing unit ("CPU"), a "task pool" populated by the CPU, and other processors called "solidarity cells" that retrieve tasks from the task pool and perform them. '777 Patent Abstract, Fig. 1, 1:17-2:63; Lowenthal Decl. ¶¶24-25.

In the '777 patent, a solidarity cell is "proactive, in that it obtains a task to perform by sending its agent to the task pool when the solidarity cell has no processing to perform." '777 Patent 2:8-10; *see id.* at 5:9-21 (same). The "agent is a data frame comprising[] a source address, a destination address, and a payload ... that comprises a list of functions the first solidarity cell is configured to perform." *Id.* cl. 1, 5:30-60. On its return from the task pool, the agent's payload is a "descriptor of the matching task" from the task pool, which may be in the form of "a memory location" or a "full descriptor data structure." *Id.*; Lowenthal Decl. ¶26.

The '777 patent does not purport to have invented any novel hardware. The claimed CPU “may be any CPU conventionally used to satisfy the computing needs of the environment in which the system 10 is implemented,” “now known or later developed,” ranging from a personal computer to a supercomputer. '777 Patent 2:64-3:24. Likewise, “[a] solidarity cell may be a general- or special-purpose processor” that “may have the same or different instruction set architecture and microarchitecture compared to the [claimed] CPU.” *Id.* 1:62-63; Lowenthal Decl. ¶34.

C. Prosecution History

The '777 patent issued from the first application filed in this patent family, but it has a significantly different specification from the '275 and '004 patents addressed in Amazon's other IPRs.

The '777 patent claims were allowed after pending dependent claims 14 and 19 were incorporated into the independent claims via an Examiner's amendment. *See* Ex. 1002 at 33-36, 67, 104, 172-173. Juniper filed an IPR challenging the claims with “Leong” as the primary reference, and the Board denied institution for failure to identify separate structures corresponding to the claimed “central processing unit” and “first solidarity cell.” *See* Ex. 1032 at 6, 12-14, 16-17.

The primary reference here, Lurie (Ex. 1007) was not considered alone or in combination with other art during prosecution or Juniper’s IPR. *See generally* Ex. 1002.

D. Person of Ordinary Skill in the Art

A person of ordinary skill in the art (“POSITA”) at the time of the earliest claimed priority date (January 2013) would have had a bachelor’s degree in computer science, computer engineering, or a related field, and about five years of industry or academic research experience in parallel or distributed computing, system architecture, or an equivalent. Lowenthal Decl. ¶¶27-28. A post-graduate degree or additional formal education can compensate for less experience, and vice-versa. *Id.*

E. Claim Construction

The prior art relied on in this Petition discloses the subject matter of the challenged claims under any reasonable construction, including their plain meaning.² The terms “solidarity cell” and “thread” are not recited in the claims of the previously challenged ’004 and ’275 patents (see IPR2022-00283 and IPR2022-00633), and are discussed further here.

² Petitioner reserves the right to argue alternative constructions in other proceedings, and where such a defense is available, that the claims are indefinite.

1. Solidarity Cell

The claims recite “solidarity cell[s],” and explain what they do: a solidarity cell “dispatch[es]” its agent to the task pool to “proactively retrieve, from the task pool” a “task for the solidarity cell to process.” *See, e.g.*, ’777 Patent cls. 1, 14.

A solidarity cell is *any* processor or computer that performs the recited features. As the ’777 patent explains, “[a] solidarity cell may be a general- or special-purpose processor” that “may have the same or different instruction set architecture and microarchitecture compared to the [claimed] CPU.” ’777 Patent 1:62-65, 3:30-45. Indeed, the previously challenged ’004 and ’275 patents simply recite “co-processor[s]” that fill similar roles. *See* Exs. 1029, 1030 (’004 and ’275 patents). Thus, the claimed “solidarity cells” are merely any processor or computer programmed to perform the recited functions. Lowenthal Decl. ¶¶32-35.

2. Thread

Claim 1 recites “dividing the [large computing] requirement into one or more threads” and that “each thread comprising one or more tasks.” *See* ’777 Patent cl. 1; *see also id.* cl. 2 (notifying the CPU when “the tasks of a thread are completed”).

In the context of the ’777 patent claims, a “thread” is a computational job that is at least a portion of a large computing requirement, and is itself composed

of one or more “tasks.” *See id.* cl. 1 (“dividing the [large computing] requirement into one or more threads, ... each thread comprising one or more tasks”).

As the ’777 patent specification explains, “[a] task thread 21 represents a computational task that [is] a component of the larger requirement imposed on the CPU 11.... A thread 21 is composed of one or more tasks 22.” *Id.* 4:22-28. The patent further explains that “solidarity cells may execute the task threads sequentially or in parallel, and independently or collaboratively, depending on recipes provided by the CPU.” *Id.* 2:13-18. Thus, using the ’777 patent’s jargon, “large computing requirements” are subdivided into one or more computational jobs called “threads,” which may be further subdivided into one or more “tasks.” Lowenthal Decl. ¶¶36-38.

Threads “may,” but are not required to, “further comprise a ‘recipe’ describing the order in which the tasks 22 should be performed and any conditions that affect the order of performance.” ’777 Patent 4:62-64, Fig. 2 (showing threads); Lowenthal Decl. ¶38.

IV. Ground 1: Claims 1-10 and 12-14 are unpatentable over Lurie and Ethernet Specification

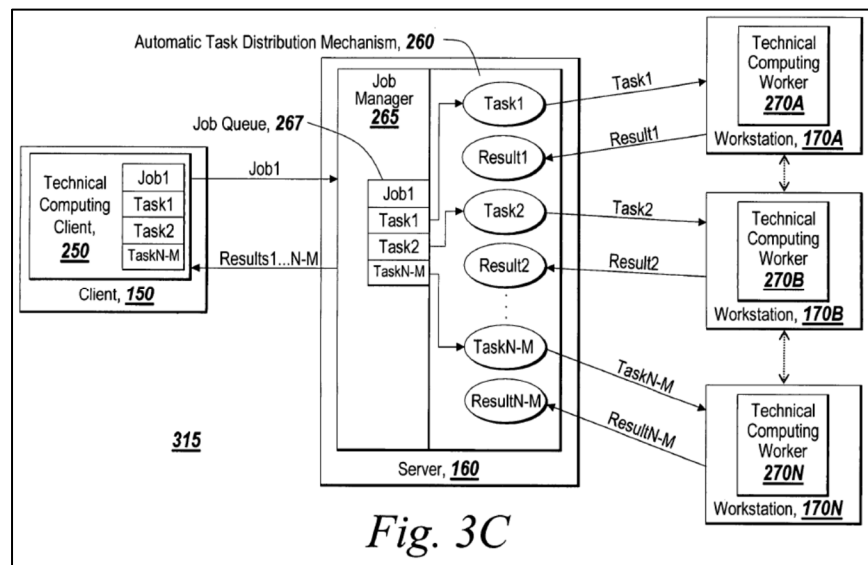
The challenged claims are unpatentable under § 103 over Lurie in view of Ethernet Specification. Lurie discloses an “instrument-based distributed system on a network” comprising a TCC (CPU), ATD Mechanism (task pool), and TCWs (solidarity cells). Lurie Abstract; Lurie Overview below. Lurie’s system transmits

information over “the Internet through a variety of connections,” including “LAN or WAN links,” which a POSITA would have considered obvious to implement using a well-known LAN protocol, such as that disclosed in Ethernet Specification. Lurie [0051]-[0052]. Andrews is a prior-art textbook that provides additional evidence of a POSITA’s knowledge regarding the common bag-of-tasks technique used by Lurie. Lowenthal Decl. ¶¶98-111.

A. Overview of the Prior Art

1. Lurie (Ex. 1007)

Lurie is § 102(b) prior art because it was published on May 31, 2007. Lurie discloses a networked parallel computing system in which a job, including one or more tasks, is completed via execution by one or more remote processors:



Lurie Figs. 3B-3D, [0005]; Lowenthal Decl. ¶¶66-76.

Lurie's technical computing client ("TCC") submits tasks grouped as jobs to its automatic/automated task distribution mechanism ("ATD"). Lurie [0071]-[0075]. The TCC communicates with the ATD, which (i) obtains tasks and makes them available for distribution, and (ii) communicates with one or more technical computing workers ("TCWs"). *Id.* [0071]-[0074]. Each TCW retrieves tasks from the ATD, computes a result, and provides the result back to the ATD. *Id.* [0071]-[0074], [0094], [0103]. The TCC "is not required to have any specific knowledge of the technical computing workers," and does "not ... specify the name of a particular technical computing worker to perform the task." *Id.* [0071]. Instead the task is taken by "any available [TCW]." *Id.* The choice of task can be based on operational and/or performance characteristics of the TCW. *Id.* [0081]; Lowenthal Decl. ¶¶71-76.

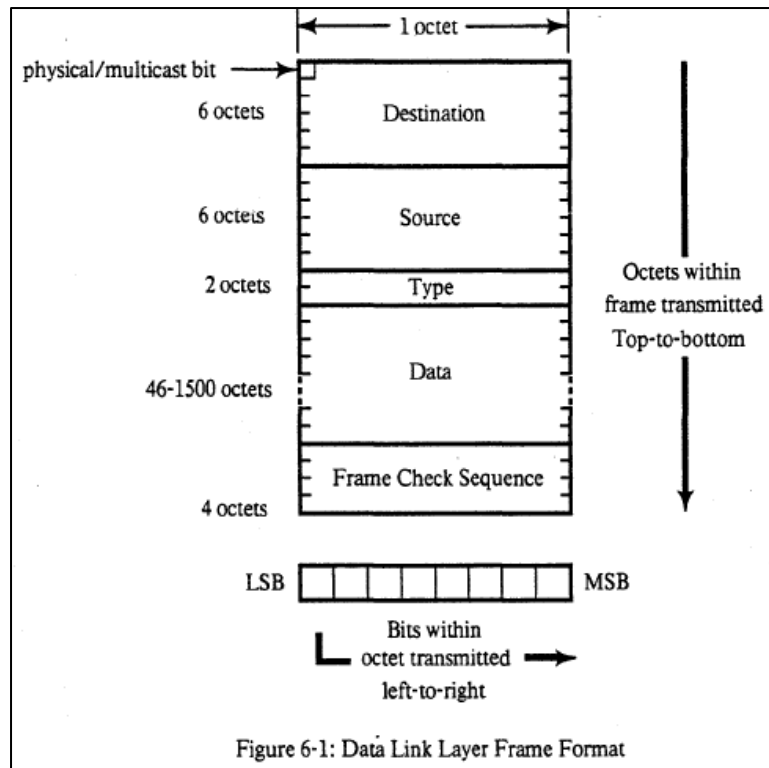
Lurie's ATD can include an "object exchange repository." Lurie [0090], Fig. 6A. Each TCW "registers with the object exchange repository 662 to be notified when a task object appears in the object exchange repository." *Id.* [0093]. When the TCC deposits a task, the repository notifies registered TCWs of the available task object, and a TCW "take[s]" the task for completion. *Id.* [0093], Fig. 6A. Any TCW may "register[]" with the object exchange repository to become available for retrieving and completing tasks. *Id.* [0093]; Lowenthal Decl. ¶¶71-76.

2. Ethernet Specification (Ex. 1009)

Ethernet Specification is § 102(b) prior art at least because it was originally published in 1980, published in *Computer Communication Review* in July 1981, made publicly available in print by the University of Wisconsin-Madison Library not later than October 11, 1981, and publicly available on the Internet Archive by July 25, 2011. Hsieh-Yee Decl. (Ex. 1005) ¶¶29-45.

Ethernet Specification describes the Ethernet protocol for a local area network (“LAN”). Ethernet Specification i, 1. Ethernet “is intended primarily for use in such areas as ... distributed data processing” and ensures “compatibility” between Ethernet implementations. *Id.* 1-2; Lowenthal Decl. ¶¶77-80.

Ethernet’s data link layer works between its physical layer and its client layer. Ethernet Specification 1-10, Fig. 4-2. Ethernet’s data link layer encapsulates and passes information in a standard format for data frames. *Id.* 19. Each frame has a source address, a destination address, and a payload, i.e., “a data field containing the transmitted data.”



Id. 19-22, Fig. 6-1; Lowenthal Decl. ¶¶80-87.

3. Andrews (Ex. 1011)

Andrews is § 102(b) prior art because it was published, and indexed and cataloged in at least the Library of Congress and Stanford University Libraries by February 2000. Hsieh-Yee Decl. ¶¶46-70.

Andrews describes the “bag-of-tasks” paradigm in which “[a] task is an independent unit of work” used “to implement parallel computations,” as shown below:

Tasks are placed in a bag that is shared by two or more worker processes.

Each worker executes the following basic code:

```
while (true) {
```

```
        get a task from the bag;  
        if (no more tasks)  
            break;           #exit the while loop  
        execute the task, possibly generating new ones;  
    }
```

Andrews 132. Workers continue to retrieve and execute tasks until the bag is empty.

Id. 132-33, 338-40, 427, 429, 561-63. Lowenthal Decl. ¶¶88-93.

B. Independent Claim 14

This Petition addresses independent claim 14 first because it does not recite claim 1's Element 1[d] (the CPU populating the task pool by dividing the requirement into threads) or Element 1[e] (tasks to comprise a descriptor having a function and memory location of data upon which the function is to be executed). Also note that the following discussion rearranges claim 14's elements for clarity by, e.g., placing "wherein" clauses with the claim elements they modify. The Claim Listing (Appendix) has the complete claim text, in order, and with the added reference letters.

Lurie in view of Ethernet Specification renders obvious claim 14. Lurie discloses an "instrument-based distributed system on a network" comprising a TCC (CPU), ATD Mechanism (task pool), and TCWs (solidarity cells). Lurie Abstract; *see also* Lurie Overview above. Lurie's system transmits information over "the Internet through a variety of connections," including "LAN or WAN links," which a POSITA would have considered obvious to implement using a

well-known LAN protocol, such as that disclosed in Ethernet Specification, whose packets constitute “agents.” Lurie [0051]-[0052]; Lowenthal Decl. ¶¶98-111.

1. Preamble

Claim 14’s preamble recites “[a]n apparatus for parallel processing of a large computing requirement.” To the extent the preamble is limiting, Lurie discloses it.

Lurie explains that its parallel-processing system is used for processing large computing requirements. Lurie’s system performs, for example, “resource intensive and time-consuming” modeling and simulations that cannot be performed by a single workstation in “reasonable amount of time,” such as “a simulation of a large complex aircraft model” that needs to be repeatedly recalculated with different parameters. Lurie [0004]-[0005]; *see also id.* Abstract. Lurie “accelerates” processing by distributing “execution” to multiple workers. *Id.* Each of these “technical computing workers [TCWs] 270A-270N can be performing technical computing of their respective tasks independently and in parallel to each other.” *Id.* [0069]-[0070]; Lowenthal Decl. ¶¶109-113.

2. Elements 14[a]-[b]: CPU and task pool

Claim 14 recites:

[a] a central processing unit (“CPU”);

[b] a task pool in electronic communication with the CPU;

'777 Patent cl. 14. Lurie discloses these elements. Lurie discloses the claimed CPU (Lurie's TCC), which is in electronic communication with the claimed task pool (its ATD, which may include a job manager as its interface).

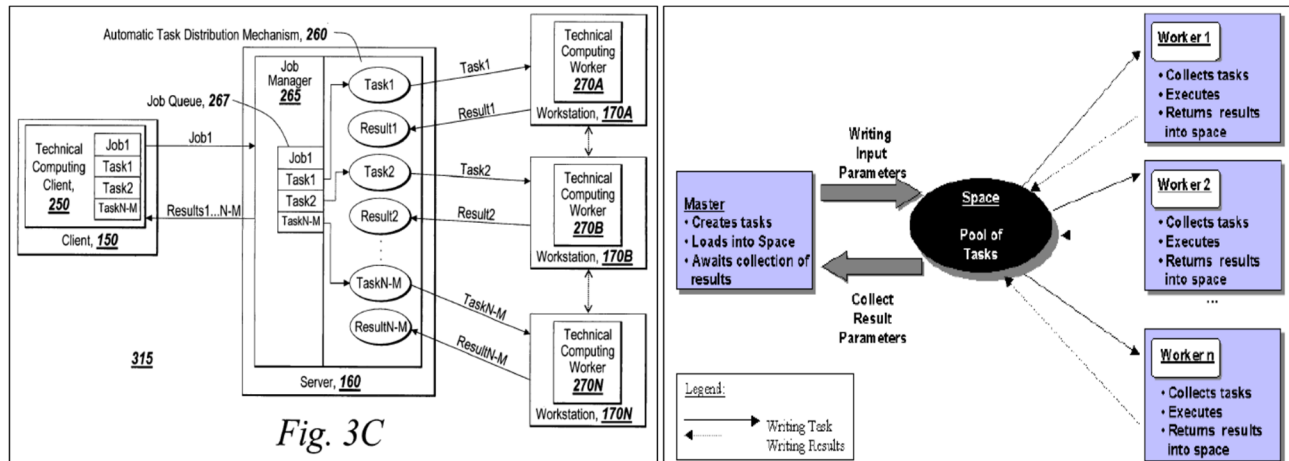
As an initial matter, the '777 patent uses the term "task pool" broadly. It states that a task pool can be "a dedicated hardware block" or "software based" and can be "represented" by any "data structures" that store tasks. *Id.* 4:12-21; Lowenthal Decl. ¶¶114-120. Indeed, Swarm contends in district court that:

"Task Pool" means any list, database, memory module, cache, random access memory, or other data or memory structure or construct, whether implemented in hardware, firmware, software, and/or executable code, that is configured to hold, store, monitor, track, display, populate with, and/or update a plurality of individual tasks.

Ex. 1014 at 4.

Lurie's "automatic task distribution [ATD] mechanism" is the claimed task pool because it is in communication with (and, as will be discussed below regarding claim 1, is populated by) Lurie's TCC (the claimed CPU), and it is a data or memory structure that holds, stores, monitors, is populated with, and updates tasks, and makes them available to TCWs (which are the claimed solidarity cells of Element 14[c], as will be demonstrated below). Lowenthal Decl. ¶¶114-120.

In addition and in the alternative, Lurie discloses implementing its ATD, i.e., a task pool, using a “object exchange repository” or “space” (in JavaSpaces terminology) that is a “pool of tasks.”



Lurie Fig. 3C; Batheja (Ex. 1012) Fig. 1; *see also* Lurie [0026], [0063] (Lurie’s ATD may be implemented “in various software components, applications and interfaces.”), [0090]-[0091] (JavaSpaces implementation), [0095], Fig. 6A; Batheja -002 (confirming that a JavaSpaces’ “space” (such as Lurie’s ATD), is a “bag-of-tasks”); Lowenthal Decl. ¶¶121-125.³ As Lurie explains, its ATD (task pool), may be implemented as a JavaSpaces “space” providing for “flow of objects [tasks and

³ Batheja is supporting evidence. Regardless, it is § 102(b) art that was published no later than 2002, and catalogued and shelved in multiple libraries. Hsieh-Yee Decl. ¶¶71-90.

results] into and out of an object exchange repository.” Lurie [0091], [0095]-[0096]; Lowenthal Decl. ¶¶122-126.

Claim 14 also requires that the ATD to be in electronic communication with a CPU. Claim 14 does not recite anything more regarding the CPU. Lurie’s TCC, as a computing device, comprises a CPU and is in communication with the ATD. Lurie [0052], [0064] (“[T]he [ATD] 260 may be in communication through the network communication channel 130 on the network 140 with one, some or all of the [TCWs] 270A-270N.”), Figs. 2C, 3C; Lowenthal Decl. ¶¶127-129. Although not recited in claim 14, it should also be understood that Lurie’s TCC (the claimed “CPU”) populates its ATD (“task pool”) by dividing a large computational requirement into one or more jobs and placing them in the ATD. *See* Element 1[d] below.

3. Elements 14[c.1], [d.1], [g]: solidarity cells and their agents

Claim 14 further recites:

[c.1] first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent ...; and

[d.1] a second solidarity cell comprising a second agent ...;

[g] wherein each agent comprises a source address, a destination address, and a payload,

A “solidarity cell” in the context of the ’777 patent’s claims is merely any processor that has the recited features—just like the “co-processor” recited in the ’275 and ’004 patents’ claims. *See* Claim Construction section above.

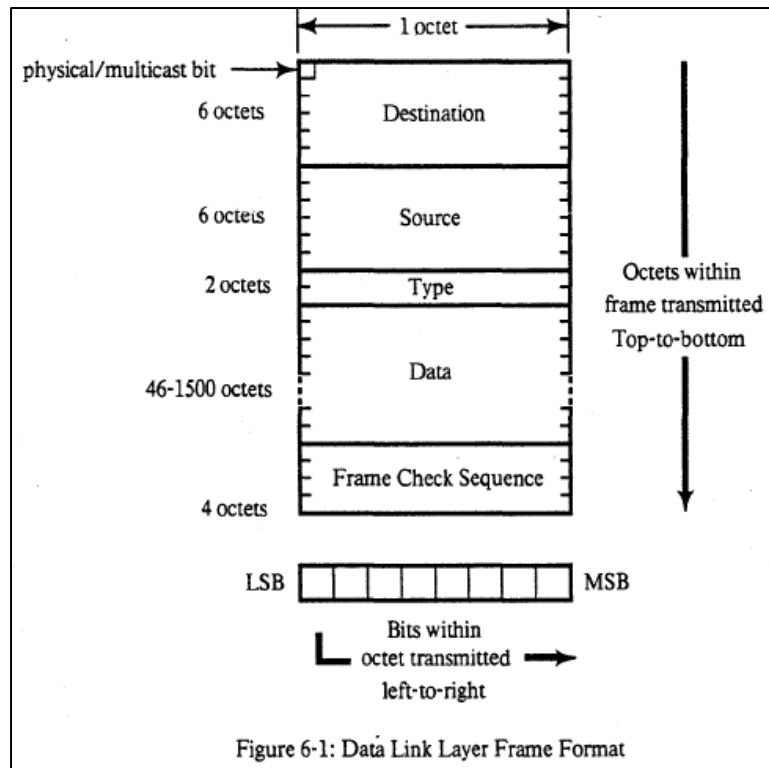
Lurie, alone or in view of Ethernet Specification, teaches these elements via Lurie’s TCWs. Any one of Lurie’s TCWs may serve as the first solidarity cell, and any other TCW may serve as the second solidarity cell. Claim 14 does not require the cells to have different functionality, but even if it did, Lurie discloses that different TCWs may be able to perform different tasks. *See* Element 14[h], discussed below.

To start, Lurie’s TCWs each comprise a processor (*see, e.g.*, Lurie [0041]-[0045], [0054]-[0056]), and as discussed further with respect to Elements 14[c.2], [d.2], [h], that processor is programmed to perform the recited features of the claim. Lowenthal Decl. ¶¶130-134. Lurie’s TCWs are also in communication with the ATD (task pool), as required by the claims. *E.g.*, Lurie [0064] (“[T]he [ATD] 260 may be in communication through the network communication channel 130 on the network 140 with one, some or all of the [TCWs] 270A-270N.”), Figs. 2C, 3C; Lowenthal Decl. ¶135.

Lurie’s TCWs also include the recited “each agent comprises a source address, a destination address, and a payload,” as will be explained in the following sections. *See also* ’777 Patent cl. 1 (“the first agent is a data frame”),

5:28-32 (“An agent ... may be considered a data frame in the networking sense.”). Lurie teaches that its TCWs communicate with the ATD via network 140 or 140’, which may be a “LAN,” “WAN,” or “the Internet.” Lurie [0051]-[0052], [0071], [0093], Figs. 1A-B, 2A-C. Thus, a POSITA would have understood that TCWs communicate with the ATD using any standard network protocol utilizing a data frame, network packet, or equivalent protocol data unit that constitutes an “agent.” Lowenthal Decl. ¶¶101-108, 135-144. Ethernet was one such well-known protocol. *Id.* ¶136; Ethernet Specification Cover, 1; Andrews 32 (describing the central role of Ethernet in “the development of distributed systems”); Ex. 1016 at 001 (“Ethernet[] ... has been the ubiquitous standard of choice for Local Area Networks (LAN).”); Ex. 1017 at 001 (similar). It would have been obvious to implement Lurie’s “LAN” using the ubiquitous Ethernet standard described in Ethernet Specification, and thus use Ethernet data frames (agents). Lowenthal Decl. ¶¶135-139.

The TCW’s agent (whether an Ethernet data frame or some other standard data frame or packet) includes a source address, destination address, and payload as required by the claims. For example, Ethernet data frames have a source address, a destination address, and a payload, i.e., “a data field containing the transmitted data.”



Ethernet Specification 19-22, Fig. 6-1; Lowenthal Decl. ¶¶138-144; Ethernet Specification Overview above. To be clear, Ethernet is just one common protocol that would have been obvious to use in implementing Lurie, and common protocols at the time (and still today)—including those pertaining to the client layer such as TCP/IP or UDP—use data frames or network packets (“agents”) having source, destination, and payload/data fields. Lowenthal Decl. ¶¶142-144.

4. Elements 14[c.2], [d.2], [h]

Claim 14 further recites:

[c.2] [the first agent] configured to proactively retrieve, from the task pool, a matching task for the solidarity cell to process; and

[d.2] [the second agent] that proactively retrieves matching tasks from the task pool for the second solidarity cell to process;

[h] and wherein each agent retrieves a matching task by:

[h.1] being dispatched by its solidarity cell to the task pool ...;

[h.2] searching the task pool ...; and

[h.3] returning to its solidarity cell ... a descriptor of the matching task.

Lurie teaches these claim elements, which together spell out how the agent proactively retrieves a matching task. It is not required by claim 14, but Lurie's agents retrieve the matching task without requiring an instruction from the task pool.

Notably, the '777 patent claims recite that the agent, not the solidarity cell as recited in the related '275 patent, proactively retrieves the task according to the recited steps [h.1-3]. As explained in the subsections below, Lurie, alone or with Ethernet Specification, teaches this. Without receiving any instructions from the TCC (CPU), or the ATD for that matter, Lurie's TCWs (cell) send a data frame/packet (agent) to the ATD (task pool) to retrieve a matching task. Lowenthal Decl. ¶¶146-158.

Specifically, Lurie's TCWs send agents (data frames/packets) to "take" tasks from the ATD, and do so without requiring an instruction from the TCC to the TCW or its agent. Lurie [0074]; Lowenthal Decl. ¶¶149-158. "A take operation

takes an object ... from the space,” “remov[ing] the object from the space.” Lurie [0091]; *see also id.* [0128] (“getTask” and “getTaskIfAvailable” methods “take” a task from the space); Lowenthal Decl. ¶¶149-150. The TCWs perform the tasks retrieved by their agents, and provide the results to the ATD when each task is completed. Lurie [0074]; Lowenthal Decl. ¶158. The ATD, including its incorporated Job Manager (*see* Lurie [0063]), thus “obtains the set of results for the completed job ... and provides the results of each of the tasks to the [TCC].” *Id.* [0074]; Lowenthal Decl. ¶158. Thus, the ATD is an intermediary: “the [ATD Mechanism] provides a level of indirection between the [TCC] and the [TCWs],” including their agents. Lurie [0061], [0092]. For this reason, the TCC (CPU) and the data frames/packets dispatched by the TCWs (cells’ agents) do not communicate with the TCC.⁴

⁴ This lack of communication is not required by claim 14, but is required by claim 1, which will be discussed below.

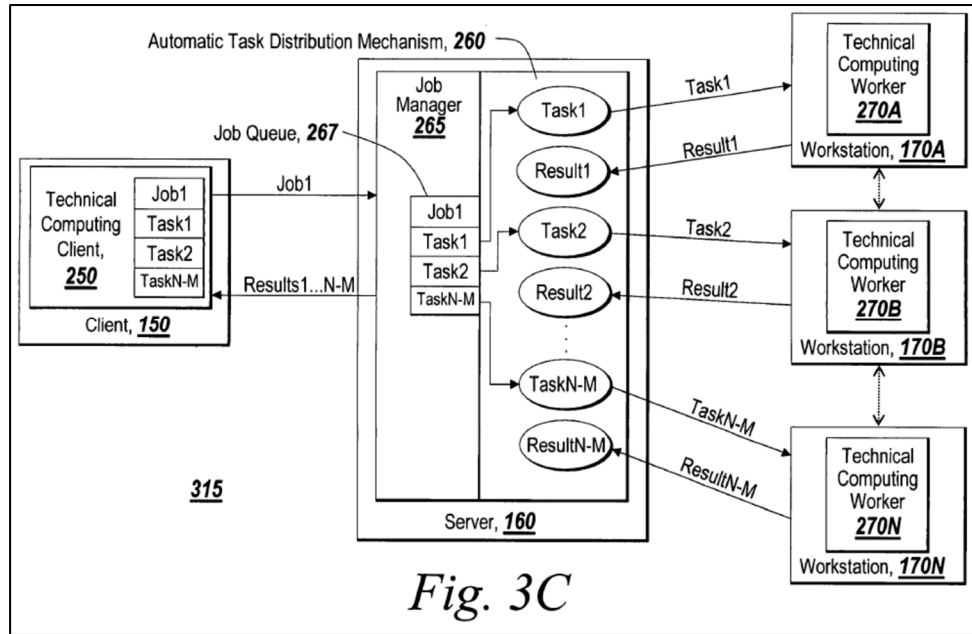


Fig. 3C

Lurie [0061], [0063], [0092], Figs. 3C-D, 4, 5C-D, 6B; Lowenthal Decl. ¶¶152-153.

Notably, the '777 patent specification states that the solidarity cells, not the agents as recited in the claims, are proactive and do not require any instructions from the CPU. To the extent Swarm argues that the claims require the cells (TCWs) to proactively dispatch their agents, the art teaches that, too. Lurie's TCW sends their agents without any instruction from the TCC, as just discussed. Indeed, the TCC "is not required to have any specific knowledge of the [TCWs]," such as "the name of the workstation ... or the availability of the [TCW]," so long as the system has an ATD. Lurie [0071]; Lowenthal Decl. ¶152.

Lurie's TCWs do not receive an instruction to retrieve a task from the ATD, either. Lurie's TCWs "register[]" with the ATD "to receive notification of one or

more tasks (step 527) becoming available, or appearing, in the [ATD].” Lurie [0087], [0097]; Lowenthal Decl. ¶154. In other words, the ATD’s notification merely informs the TCWs when tasks first become available. Such notifications are not instructions, as a TCW is not obligated to respond by retrieving a task. For example, a TCW that has sent a de-registration message to the ATD should not retrieve a task even if it receives a notification. Lowenthal Decl. ¶¶154-158; *see also* Element 14[f] below.

In addition, or in the alternative, the ’777 patent states that a solidarity cell is “proactive” when “it obtains a task to perform by sending its agent to the task pool when the solidarity cell has no processing to perform.” ’777 Patent 2:8-10; *see id.* 5:9-21 (same). Lurie’s solidarity cells are proactive under this definition, too. Lurie teaches that its TCWs (solidarity cells) obtain tasks when they do not have a task to process. *See* Element 14[f] below.

a. Element 14[h.1]: dispatch agent to task pool

This claim element recites:

[h.1] being dispatched by its solidarity cell to the task pool, during which the source address is its solidarity cell’s address, the destination address is the task pool’s address, and the payload comprises a list of functions the agent’s solidarity cell is configured to perform;

Lurie, alone or in view of Ethernet Specification, teaches this claim element.

Lurie's TCW (solidarity cell) retrieves a task by dispatching a data frame/packet (agent) with a "source" field (source address) identifying "the station sending the frame," (the sending TCW). Ethernet Specification 21; Lowenthal Decl. ¶¶160-161. The "destination" field (destination address) of the data frame identifies "the station(s) for which the frame is intended," i.e., the ATD (task pool). Ethernet Specification 21; Lowenthal Decl. ¶162. The data field of the Ethernet data frame, or that of any suitable network data frame/packet (payload), allows for any type of data or payload, and when implementing Lurie would contain, e.g., one or more indices, names or pointers/references identifying the list of functions the sending TCW is capable of performing. Ethernet Specification 22; Lurie [0043], [0081]; Lowenthal Decl. ¶163; *see also* Element 14[g] above.

The '777 patent's specification does not limit or explain how the recited list of functions may be encoded into the agent's payload (i.e., the data field of the Ethernet data frame or other protocol's frame/packet). It states only that the payload "contains identifying information of the types of tasks the corresponding cell ... can perform." '777 Patent 5:55-57. Swarm interprets the "list of functions" phrase broadly, alleging, for example, that "defin[ing] a set of remote operations" is sufficient. *See* Ex. 1033 at -052 (infringement allegation). Also, the "list of functions" may simply be one function. *See* '777 Patent cl. 3 (solidarity cell can perform "one or more" task types); Lowenthal Decl. ¶171.

Lurie, alone or in view of Ethernet Specification, teaches that its agent's payload comprises a list of functions that its cell (TCW) can perform. Lurie provides more detail than the '777 patent, disclosing that its TCW will retrieve (via its agent) only tasks that the TCW is configured to perform—based on “operational and/or performance characteristics” or “*any* configuration or properties” of the TCW. Lurie [0081]. For example, Lurie's TCW “may be configured to and capable of running any of the modules, libraries, or software components of the MATLAB® product family,” i.e., the TCW “may have all or a portion of” MATLAB installed or accessible. *Id.* [0058]. In addition, the TCW “may work on a task based on the version of [MATLAB] that is installed,” which would include the release as well as, e.g., whether the TCW is running the 32-bit or 64-bit version, or no version at all. *Id.* [0081], [0065]. For example, some TCWs may run MATLAB, while others may run LabVIEW, MATRIXx, Mathematica, Mathcad, Maple, or something else entirely. *Id.* [0046]; Lowenthal Decl. ¶¶163-169.

An obvious way to implement Lurie's disclosure of matching the TCW based on its capabilities to a task function that it can perform would have been for the TCW to dispatch a data frame or agent carrying a payload that includes, e.g., one or more indices, names or pointers/references associated with one or more (list of) task types that the TCW can perform—such as a particular version of a particular platform (e.g., MATLAB, LabVIEW, etc.), or a subset of the

function/commands in that platform. Lurie [0081] (also using a “command” or “message” to communicate the TCW’s capabilities); Lowenthal Decl. ¶¶169-170. Indeed, Lurie implies as much. Lurie explains that if the “getTask” method is passed a “null,” it will return the TaskID of any available task—thus implying to a POSITA that passing the “getTask” method a parameter identifying the types of functions the TCW can perform will return a TaskID of an available one of those types of tasks. Lurie [0128]; Lowenthal Decl. ¶169.

b. Element 14[h.2]: search the task pool

This claim element recites:

[h.2] searching the task pool for a task that is ready to be processed and has a function that the agent’s solidarity cell can perform;

The ’777 patent does not limit how the claimed searching may be performed. It does, however, recognize that the agent may “search for a match between the task type of the ready tasks 22 and the set of tasks that the cell 12A is able to perform,” and that this “requires a particular hard-coding of the types of tasks that the CPU 11 will create.” ’777 Patent 6:15-19.

Lurie teaches this. First, Lurie discloses that the tasks made available at the ATD are ready to be processed. *E.g.*, Lurie [0063] (job manager helps manage “workflow”), [0073]-[0075], [0081], [0095]; Lowenthal Decl. ¶¶173-174.

Second, Lurie (alone or with Ethernet) teaches or suggests that the dispatched agent searches the task pool for a task function that the sending TCW can perform. As just discussed, a POSITA would have understood that Lurie's TCWs have different operational characteristics and are configured to perform different task functions (e.g., different subsets of MATLAB, different versions of MATLAB, and/or different platforms entirely), and that Lurie's system identifies appropriate functions for that TCW. As also discussed in the immediately preceding subsection, Lurie (alone or with Ethernet Specification) teaches TCWs (cells) that dispatch agents with a payload that comprises the list of functions that the particular TCW can perform.

A POSITA would have understood that an obvious way to ensure that each agent retrieves a task that matches the capabilities of its sending TCW is for the agent to search for and retrieve a task with a function that matches one found in the list of functions. Lowenthal Decl. ¶¶175-188. That is, a TCW's agent, when dispatched to the task pool, carries a payload that includes, e.g., one or more indices, names or pointers/references to the list of functions that the TCW (cell) can perform so that the agent can search for and find a matching task in the task pool. *Id.* ¶¶175-178. For example, Ethernet's "data field" contains a series of up to 1500 octets, which would carry the payload of, e.g., one or more indices, names or

pointers/references corresponding to the list of task types the TCW can perform.

Ethernet Specification Fig.6-1; Lowenthal Decl. ¶179.

Lurie discloses that an agent searches the ATD via an “associative lookup” to match the TCW with tasks of the appropriate type based on, e.g., the one or more indices, names or pointers/references provided in the agent’s payload. Lurie [0081], [0091]; Lowenthal Decl. ¶¶176-183; *see also* Element 1[e] above. In the context of the ’777 patent, the agent is configured to search the ATD because the agent carries the information necessary to perform the associative lookup and initiates that process. Lurie [0081], [0091]; Lowenthal Decl. ¶¶177, 183. This aligns with the disclosure in the ’777 patent that a cell “acquire[s] a task” by send[ing] an agent 30A to the task pool 13 to search for and retrieve an available task 22 that requires completion, is not locked, and has a task type that can be performed by the cell 12A.” ’777 Patent 5:25-28. The associative lookup conducted on the ATD’s space, which is enabled and initiated by the information contained in the agent’s payload such that the operational and/or performance characteristics of the TCW can be matched with the task type, is a method of searching the task pool for a task. Lowenthal Decl. ¶¶177, 184.

In addition and in the alternative, it would also have been obvious to implement Lurie to include in the payload a script, or reference to a script or function, for searching the ATD. *Id.* ¶¶141, 180-181. For example, the data

frame/packet's payload identifies itself to the ATD as an agent from a TCW seeking a task, which includes, or at least invokes the execution of, a search function (e.g., calling Lurie's "getTask" or "getTaskifAvailable" method) (Lurie [0128]), and passing as a parameter the one or more indices, names or pointers/references corresponding to the list of tasks that particular TCW can perform (or null if the TCW can perform any task). Lowenthal Decl. ¶¶169, 180-181.

In sum, Lurie and Ethernet Specification teach or suggest the agent searches the task pool for a matching task that is ready to be processed. Lurie discloses a computing system in which a CPU obtains, and populates the task pool with, multiple tasks of different types, and teaches the use of multiple solidarity cells (TCWs) to process tasks efficiently. Lurie discloses specific task types and retrieving tasks that are suited for a particular TCW. Lurie [0081]. A POSITA would have understood Lurie to teach, or else considered it obvious, that to ensure each cell processes a task appropriate for its operational and/or performance characteristics, Lurie's disclosed system uses agents on a network in the form of data frames, network packets, or equivalent protocol data units whose data field carries, e.g., one or more indices, names or pointers/references corresponding to the task types. Lowenthal Decl. ¶182.

c. Element 14[h.3]: return to cell

This claim element recites:

[h.3] returning to its solidarity cell, during which the source address is the task pool's address, the destination address is the agent's solidarity cell's address, and the payload comprises a descriptor of the matching task.

Lurie (alone or with Ethernet Specification) teaches this claim element.

Lurie's TCW (cell) dispatches a data frame (agent) to retrieve a first task. *See* Element [g.1]. A POSITA would have understood that when Lurie's data frame (agent) returns from the ATD Mechanism to the TCW with the retrieved first task, its source address is the ATD (the "source" of the agent on the return trip) and its destination address is the TCW (the agent's "destination"). Lowenthal Decl. ¶¶184-186. The returning data frame's payload carries a descriptor of the task, as Lurie explains: "[t]he [TCW] obtains the name and arguments of the function to compute from the data structure of the task object." Lurie [0094]; Lowenthal Decl. ¶¶187-188. The "name and arguments of the function to compute" constitute descriptors of the first task that the TCW obtains via the Ethernet data frame (agent)'s payload on its return from the ATD Mechanism. Lowenthal Decl. ¶¶187-188; Element 1[e] below (explaining that Lurie's function "name and arguments" may form the claimed descriptor).

5. Element 14[e]: matching task in task pool

Claim 14 further recites:

[e] wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell;

Lurie teaches this. The “matching task” for each cell is one of the tasks placed in the ATD (task pool) by the TCC (CPU), and specifically the one that is retrieved and processed by the cell.

As explained above for Elements 14[c]-[d], and [h], Lurie’s TCWs take from the ATD matching tasks that the TCWs can process and are ready to be processed, compute the results, and provide the results back to the ATD. Lurie Figs. 3C-3D, [0073]-[0074] (The “technical computing workers 270A-270N ... take [the tasks] and compute results.”), [0081], [0087]-[0088], [0097]; Lowenthal Decl. ¶¶189-190.

6. Element 14[f]: each cell sends agent when not processing a task

Claim 14[f] further recites:

[f] wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process;

Lurie teaches or suggests this claim element.

TCWs register with the ATD to receive a notification when tasks first become available. *See* Elements 14[c.2], [d.2], [h], above. After that, each

registered TCW continues to retrieve and process tasks without further notifications from the ATD until the tasks are all completed and the ATD (task pool) is empty. That is, each registered TCW takes a task from the ATD (via its agent, i.e., data frame/packet), performs it, deposits the results in the ATD, and then takes another suitable task from the ATD, repeating the process until the ATD has no more tasks. Lurie [0087]-[0088], [0095-0097], [0128] (using, e.g., “getTask” or “getTaskIfAvailable”); Lowenthal Decl. ¶¶191-196.

A POSITA would therefore have understood that Lurie’s batch mode uses a textbook bag-of-tasks technique, with the addition of an initial notification informing registered TCWs when tasks first become available. Andrews 132-33 (workers continue to get and process tasks until bag is empty), 338-40 (same, with the bag manager telling workers the bag is empty by placing a “stop token” in the bag), 427, 429, 561-63 (each workers continues to get and process tasks until it retrieves a “‘bag is empty’ sentinel”); Lowenthal Decl. ¶196.

In addition and in the alternative, it would have been obvious to implement Lurie to have this functionality, i.e., so that only one notification is sent when a job (i.e., its one or more tasks) is made available to the TCWs. As just discussed, each worker in the textbook bag-of-tasks model set forth in Andrews retrieves a task, and then when done executing that task, retrieves another task—and does so until

the bag is empty.⁵ A POSITA would have implemented Lurie using that common, tried-and-true technique, sending Lurie's notification once, at the beginning of the first job, which starts the TCWs repeatedly retrieving (via their agents) and executing tasks until the bag of tasks is empty and all pending jobs are complete.⁶ Lurie's notification helps reduce unnecessary message traffic while eliminating a deadlock problem. Lowenthal Decl. ¶¶197-198.

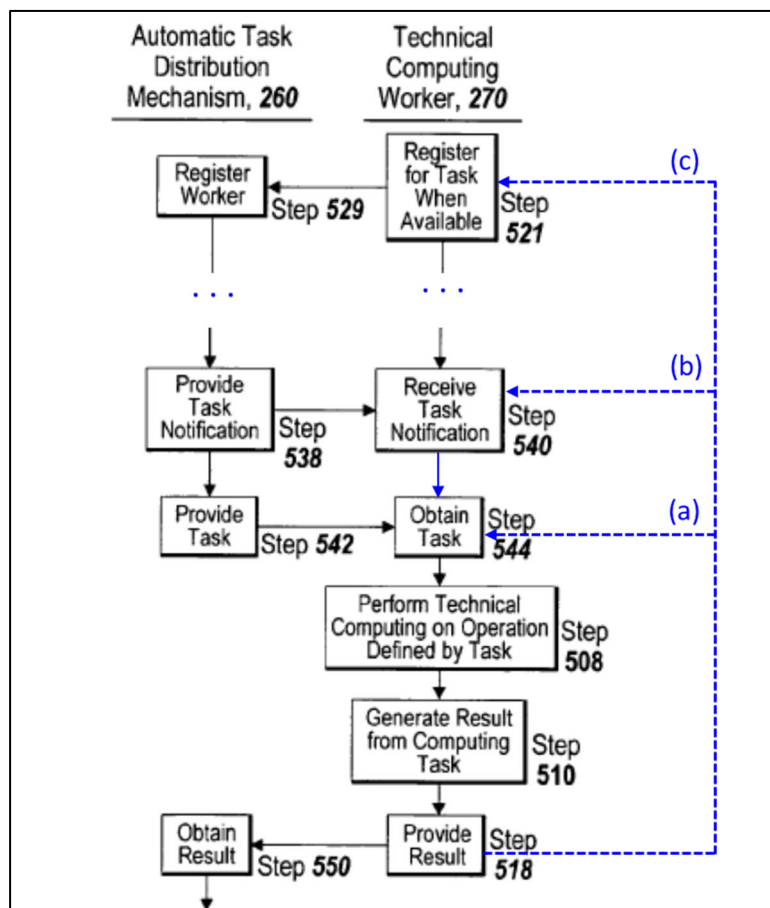
To explain, Lurie's ATD notifies the registered TCWs that "one or more tasks" constituting at least one job are available at the ATD. Lurie [0087]. Lurie walks through the first iteration of one TCW retrieving its task, but does not (and does not need to) explain to the POSITA what happens when a TCW finishes its task: i.e., how does the TCW get its next task? And to be sure, a POSITA would have understood that each TCW performs more than one task. At the time, POSITAs designed systems to have at least 2-3 times as many tasks in a job as

⁵ To the extent Swarm insists on a prior-art reference for this old and well-known textbook technique, the Andrews textbook is a prior-art printed publication. *See* Andrews Overview above.

⁶ An exception would be when registering and onboarding a newly added TCW while tasks are already being processed. In that case, the ATD would notify the new TCW(s) that tasks are already available. Lowenthal Decl. ¶196.

there are TCWs to ensure adequate load balancing with heterogenous tasks, as in Lurie. Andrews 133. In other words, a POSITA would understand that Lurie loops back, or would at least implement it to do so, to ensure all the tasks are completed in the pending job or jobs. Lowenthal Decl. ¶194.

A POSITA would have recognized there are three options in Lurie, and that one of them is preferable: (a) the TCW obtains its next task upon completion of its current task, (b) the TCW stop and waits for another notice that tasks are available, or (c) the TCW re-registers and then awaits another notice:



Lurie Figs. 5C-D (excerpt, annotated); Lowenthal Decl. ¶197. Given the small number of options, option (a) would have at least been obvious to try. *Uber Techs., Inc. v. X One, Inc.*, 957 F.3d 1334, 1339 (Fed. Cir. 2020) (obvious when there is a “‘finite number of identified, predictable solutions’ to a design need”) (quoting *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 421 (2007)).

Further, a POSITA would have chosen option (a) because it follows the textbook bag-of-tasks model that efficiently allows the TCWs to complete all of the tasks while avoiding deadlock—and without the unnecessary and redundant notifications and registrations of options (b) and (c). In the textbook bag-of-tasks process, each worker retrieves a task, performs the task, and then proactively retrieves another task, repeating that cycle until the bag is empty—without receiving any intervening notifications. Andrews 132-33, 338-40, 427, 429, 561-63. Andrews explains that the workers learn the bag is empty when they ask for a task but retrieve (via their agents), for example, a “‘bag is empty’ sentinel” instead of a task. *Id.* 563; Lowenthal Decl. ¶¶196-197.

A POSITA implementing Lurie would have used this same technique, using Lurie’s “getTask” or “getTaskIfAvailable” method. Lurie [0128]. Returning a bag-is-empty value informs the TCWs that the ATD is online and responsive (and not crashed or otherwise offline), but that no tasks are available because they have all

been completed.⁷ The still-registered TCWs may then simply and efficiently wait until the ATD notifies them that new tasks from more jobs are once again available, without repeatedly and futilely sending their agents to check for tasks. And once notified that tasks are again available, the TCWs would again proactively retrieve tasks without registering or receiving any additional notifications that tasks are available. Lowenthal Decl. ¶¶196-197.

In short, a POSITA would have implemented Lurie such that each TCW follows the textbook bag-of-tasks paradigm by sending its agent to retrieve another task from the ATD, processing that task, and repeating the procedure until the ATD is empty—without receiving an additional notification after the first one. *Id.* ¶198.

* * *

For the reasons set forth above, claim 14 would have been obvious over Lurie alone or with Ethernet Specification.

⁷ Without this mechanism, a TCW that sends an agent and receives no response from the ATD does not “know” whether the ATD is offline or simply empty and awaiting more jobs/tasks.

C. Independent claim 1

Claim 1 would have been obvious for similar reasons as claim 14, as explained in the following sections.

Claim 1 is similar to claim 14, except that claim 1 further requires Element 1[d] (the CPU populating the task pool by dividing the requirement into threads) and Element 1[e] (tasks to comprise a descriptor having a function and memory location of data upon which the function is to be executed), and minor differences that will be discussed below.

Note that the following discussion rearranges claim 1's elements for clarity.

1. Preamble

Claim 1's preamble is identical to claim 14's, and if limiting, is taught by the art for the same reasons. Lowenthal Decl. ¶¶199-200.

2. Elements 1[a]-[b] and [d]: CPU populates the task pool

Like claim 14, claim 1 recites:

[a] a central processing unit ("CPU");

[b] a task pool in electronic communication with the CPU;

These elements are the same as claim Element 14[a]-[b], and the prior art teaches them for the same reasons as in claim 14. Lowenthal Decl. ¶¶201-202. Unlike claim 14, claim 1 further recites:

[d] wherein the CPU populates the task pool by dividing the requirement into one or more threads and placing the threads in the task

pool, each thread comprising one or more tasks, and the matching task being one of the tasks;

Lurie discloses this element, too, because its CPU (TCC) populates the task pool as claimed.

To explain in detail, Lurie's TCC (the claimed "CPU") populates its ATD ("task pool") by dividing a large computational requirement into one or more jobs (the claimed "threads") and placing them in the ATD. Lurie's system performs, for example, "resource intensive and time-consuming" modeling and simulations (i.e., the claimed "requirement") that cannot be performed by a single workstation.

Lurie [0004]-[0005]; Lowenthal Decl. ¶¶203-204; *see also* Claim 1 and 14

Preamble Elements above. Lurie's TCC associates the tasks of that large computing requirement with at least one "job" (i.e., the claimed "one or more threads"), and submits the job(s) and its/their tasks to the ATD (the claimed task pool) via the ATD's "job manager" interface. Lurie [0005] ("[A] client creates a job."), [0006] (The client "generate[s] a job in the computing environment, wherein the job includes one or more tasks."), [0008]-[0009], [0043] ("Tasks can be declared on a technical computing client and additionally organized into jobs."), [0047] (same), [0073]-[0074] (TCC defines a job, associates a batch of related tasks with that job, and submits the job to the ATD via the ATD's job manager), [0075] (multiple jobs), [0077]-[0078] (multiple jobs), [0087]-[0088], [0095]-

[0096]; *see also id.* [0063] (the ATD “can include the functionality and object-oriented interface of the job manager” as part of a “single application[] or software component”), Figs. 3C-3D, 5C-5D; Lowenthal Decl. ¶¶204-206.

Lurie’s jobs are the claimed threads because they are components of a large computing requirement and are composed of one or more tasks, as just discussed. *See* Claim Construction section above (construing “thread”); Lowenthal Decl.

¶207. The claims do not require the recited threads to further include a “recipe” or other information for controlling performance of the threads’ tasks. But even if there was such a requirement, Lurie teaches that, too. Lurie’s job manager (which may be included in its ATD, i.e., task pool) and ATD manage the execution of the tasks of the submitted jobs to ensure the jobs are done correctly. Lurie [0063] (“managing a group of tasks collectively as a single unit called a job” and the ATD can include the job manager’s functionality), [0074]-[0075], [0078], [0087]-[0088], [0095]-[0096], [0100]; Lowenthal Decl. ¶¶207-208.

And as required, the “matching task” recited in Element 1[d] is one of the tasks of the job(s) (“threads”) placed in the ATD (“task pool”) by the TCC (“CPU”). As fully explained below for Element 1[c], Lurie’s TCWs correspond to the claimed solidarity cells. Each TCW takes a matching task from the ATD, computes the results, and provides the results back to the ATD. Lurie Figs. 3C-3D,

[0073]-[0074] (The “technical computing workers 270A-270N ... take [the tasks] and compute results.”), [0081], [0087]-[0088], [0097]; Lowenthal Decl. ¶209.

3. Element 1[e]: tasks have a task descriptor

Claim 1, unlike claim 14, further recites:

[e] wherein each task comprises a descriptor, the descriptor containing at least: a function to be executed; and a memory location of data upon which the function is to be executed;

Lurie teaches this claim element. Lurie’s tasks include a descriptor, i.e., the “name of [its] function and parameters,” and Lurie’s TCW can “obtain[]” that descriptor, i.e., “the name and arguments of the function to compute from the data structure of the task object.” Lurie [0120]-[0121], [0094]. In Lurie, “[a] task defines a technical computing command ... to be executed, and the number of arguments and any input data to the arguments.” *Id.* [0043], [0085] (same), [0056] (TCC defines/declares a “function” that is executed by TCWs), [0061], [0087]-[0088] (“The technical computing client 250 defines one or more[] tasks ... [each] comprising an operation or function for technical computing” that will be performed by the TCW.), [0091], [0171]-[0194] (creating a task includes specifying the function and its arguments); Lowenthal Decl. ¶¶210-211.

Lurie also teaches that its tasks—including the data upon which the function will be executed—are stored in memory locations. The claimed descriptor of each task, including the “the name and arguments of [its] function,” is stored in a “data

structure,” i.e., memory location(s) where the function’s data is organized and stored. Lurie [0094]. Thus, the task’s descriptor includes the memory location of data upon which the function will be executed. Lowenthal Decl. ¶212. Indeed, Lurie’s jobs store tasks, including their function name, data, and results, in a “cell array,” i.e., an array of memory locations. Lurie [0131]-[0132] (a job’s properties include “tasks” stored in a “cell array of task objects.”), [0133], [0177] (task’s arguments in an array within the task); Lowenthal Decl. ¶212.

Lurie also discloses that the task (and thus its function and data’s memory location(s), which comprise the descriptor) may be obtained by “associative lookup” (which is identifying a memory location using, e.g., a key value) “rather than by memory location or identifier” (which is identifying a memory location using a pointer/reference or a variable). Lurie [0091]; Lowenthal Decl. ¶213 (also explaining that associative lookup is not limited to JavaSpaces). This teaching in Lurie also disclosed to a POSITA that Lurie’s task descriptors each includes a function and a memory location for the function’s data, whether implemented in JavaSpace or not. Lowenthal Decl. ¶213; *see also* Claims 8 and 9 discussions below.

4. Elements 1[c.1]-[c.2] and [f]-[g]: first solidarity cell with agent retrieves task

Claim 1 further recites:

[c.1] a first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent

[c.2] configured to proactively retrieve, from the task pool, without requiring an instruction from the CPU, a matching task for the solidarity cell to process;

[f] wherein the first agent is a data frame comprising: a source address, a destination address and a payload;

[g] wherein the first agent retrieves the matching task by:

[g.1] being dispatched ... to the task pool ...;

[g.2] searching the task pool ...; and

[g.3] returning to the first solidarity cell....

As in claim 14, claim elements 1[c.1]-[c.2] and [f]-[g] recite a first solidarity cell in communication with the task pool that includes a first agent configured to proactively retrieve, from the task pool, a matching task for the solidarity cell to process according to the same three steps.⁸ Lurie teaches all of these elements for the same reasons as Elements 14[c.1]-[c.2] and [g]-[h], and as further discussed below. Lowenthal Decl. ¶¶214-215.

Claim 1, unlike claim 14, further requires the agent to retrieve the matching task “without requiring an instruction from the CPU.” But as explained above with respect to Elements 14[c.2], [d.2] and [h] and 14[f], Lurie, alone or with Ethernet

⁸ Unlike claim 14, claim 1 does not recite a second cell.

Specification, teaches this. Lurie’s TCWs (cell) send a data frame/packet (agent) to travel between the TCW and the ATD (task pool). The TCW’s data frame/packet (cell’s agent) retrieves a matching task, while not receiving any instructions from the TCC (CPU), or the ATD for that matter. *Id.* ¶216.

And to the extent Swarm argues that claim 1 should be interpreted to require the agent to be dispatched “when the solidarity cell does not have a matching task to process” even though that is not recited in the claim, the art teaches or renders that obvious, too—as demonstrated above for Element 14[f], where it *is* recited.

* * *

For the reasons set forth above, claim 1 would have been obvious over Lurie alone or with Ethernet Specification.

D. Dependent claims 2-10, 12-13

The following claims are unpatentable for the same reasons as claim 1 and as further explained below. Conversely, claim 1 is unpatentable for the same reasons as its dependent claims.

1. Claim 2: thread completion notification

Claim 2 depends from claim 1, and further recites that “the task pool notifies the CPU when the tasks of a thread are completed.”

Lurie discloses this. The job manager portion of Lurie’s ATD (task pool) notifies its TCC (CPU) when the job (thread) is completed, as required by this

claim. Lurie [0063] (Job manager may be part of ATD), [0088] (Job manager uses callback function to inform the TCC that the job is completed), [0096] (same), Figs. 5C-D, 6B; Lowenthal Decl. ¶¶218-219.

2. Claim 3: task types

Claim 3 depends from claim 1, and recites that “the tasks each comprise a task type selected from a set of task types, and wherein the first solidarity cell is configured to perform tasks of one or more of the task types.”

Lurie discloses this. As discussed above with respect to Elements 14[h] and 1[g], Lurie discloses its TCWs are configured to perform one or more task types (i.e., tasks having particular functions) in a list (i.e., set) of task types, such as tasks requiring functions in any MATLAB version or subset of a MATLAB version, or any version/subset of other platforms such as LabVIEW®, MATRIXx, Mathematica®, Mathcad, Maple™, or something else entirely. Lowenthal Decl. ¶¶220-221.

3. Claims 4-7: matching task ready to be processed

Claim 4 depends from claim 3, and further recites that “the matching task is a task that is ready to be processed and has a task type that the first solidarity cell can perform.”

Claim 5 depends from claim 4, and similarly recites that “the first agent retrieves the matching task by: searching the task pool for a task that is ready to be

processed and has a task type that the first solidarity cell can perform; and identifying the matching task.”

Claim 6 depends from claim 1, and similarly recites that “wherein the matching task is a task that is ready to be processed and the function of the matching task can be performed by the first solidarity cell.”

Claim 7 depends from claim 6, and similarly recites that “the first agent retrieves the matching task by searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform, and identifying the matching task.”

Lurie (alone or with Ethernet Specification) teaches all of these claim elements. As explained with respect to Elements 14[c] and [h.2] (and Elements 1[c.2] and [h.2]), Lurie searches for, identifies, and retrieves a “matching task” that is ready to be processed and has a function that the TCW can perform, as required by those claim elements. As discussed with respect to claim 3 above, in the context of the ’777 patent claims, different task types include tasks that have or require different functions. Lowenthal Decl. ¶¶222-226.

4. Claim 8: task descriptor data storage location

Claim 8 depends from claim 1, and recites that “the descriptor further contains a memory location where processed data is to be stored.”

Lurie discloses this. As explained above for Element 1[e], Lurie stores task descriptors, including the task's function name, data, and results, in a data structure, i.e., memory location(s) where the function's data is organized and stored, which are in turn, stored in a "cell array," i.e., an array of memory locations, that may be accessed by variable, pointer/reference to a particular memory location, or by associative lookup. Lowenthal Decl. ¶¶227-228. Lurie further discloses that the "data structure of the task object" can include "a field representing the output arguments from the execution of the function defined by the task," such as "one or more arrays of data as allowed by the programming language of MATLAB®," which is also a memory location where the processed data is to be stored. Lurie [0103]; Lowenthal Decl. ¶¶228-229.

5. Claim 9: task descriptor is a data structure

Claim 9 depends from claim 1, and recites that "the descriptor is a data structure and the task contains a reference to the memory location of the descriptor."

Lurie discloses this. As explained above for Element 1[e], Lurie stores task descriptors, including the task's function name, data, and results, in a data structure, that in turn, may be stored in a cell array. A POSITA would have understood that each task of the cell array is accessed by providing indices to the cell array, i.e., a memory location of the descriptor. Thus, a POSITA would have

understood that each task contains a reference to the memory location of descriptor. Lowenthal Decl. ¶¶230-232.

6. Claim 10: task pool in physical memory

Claim 10 depends from claim 1, and recites that “the task pool occupies a region of physical memory.”

Lurie discloses this. Lurie’s ATD “comprises one or more application software components” and is on a server 160. Lurie [0060], [0061]. That server includes “memory 106,” which is physical memory such as “DRAM, SRAM, EDO RAM, etc.” *Id.* [0050]. Lurie’s ATD therefore occupies a region of physical memory. Lowenthal Decl. ¶¶233-234.

7. Claim 12: second solidarity cell and agent

Claim 12 depend from claim 1, and recites the same “second solidarity cell” feature as independent claim 14. It is taught by the art for the same reasons explained with respect to claim Element 14[d], namely, any one of Lurie’s TCWs may serve as the first solidarity cell, and any other TCW may serve as the second solidarity cell. Lowenthal Decl. ¶¶235-236.

8. Claim 13: send agent when no task to process

Claim 13 depends from claim 12, and recites requiring each of the recited solidarity cells to send its agent to the task pool when the cell does not have a task to process. It is taught for the same reasons explained with respect to Element 14[e]. Lowenthal Decl. ¶¶237-238.

V. Ground 2: Claim 11 is unpatentable over Lurie, Ethernet Specification, and Lippett

Claim 11 depends from claim 10, and recites that the task pool is disposed in a hardware block dedicated to the task pool. Claim 11 would have been obvious for the same reasons as claim 10 and for the additional reasons explained in this section.

A. Overview of Lippett (Ex. 1031)

Lippett is § 102(b) art because it was published on September 20, 2007. Lippett disclosed “scheduling executable transactions within a multicore processor comprising a plurality of processor elements.” Lippett Abstract. Lippett is analogous art because it, like Lurie and the ’777 patent, is within the field of scheduling executable tasks. It is also reasonably pertinent to the problem with which Lurie and the ’777 patent are involved, namely implementing a scheduler in a multiprocessing (e.g., multiprocessor or multicore) environment. Lowenthal Decl. ¶¶94-97.

B. Claim 11: task pool in a dedicated hardware block

Claim 11 depends from claim 10, and recites that “the task pool is disposed in a hardware block dedicated to the task pool.”

Lurie discloses this claim element alone or in view of Lippett. As explained with respect to claim 10, the task pool occupies a region of physical memory. That

block of physical memory is the claimed “hardware block dedicated to the task pool”

In addition, using “dedicated memory” as taught by Lippett (i.e., the claimed hardware block dedicated to the task pool) was a well-known technique to “improve performance” in such systems because it reduces the amount of time it takes to read and write to memory. *E.g.*, Lippett [0007]; Lowenthal Decl. ¶¶240-243. The use of such “tightly coupled memory” containing “scheduling configuration and dynamic descriptors used to represent units of work within the system” could have been implemented using, for example, “a standard compiler SSRAM technology supplied by numerous EDA vendors.” Lippett [0110], [0440]-[0442]; Lowenthal Decl. ¶243.

VI. Secondary Considerations

There are no secondary considerations known to Amazon that affect—let alone overcome—the strong cases of obviousness set forth above. Should Swarm offer any relevant evidence of secondary considerations in its preliminary response, Amazon should be given leave to file a reply.

VII. The Board Should Reach the Merits of This Petition

Amazon is aware of no discretionary basis warranting denial under the facts and law as they stand today, but reserves the right to seek a reply to address any discretionary arguments raised in the POPR.

Amazon also notes that this IPR trails its other two against the related '004 and '275 patents⁹ because the '777 patent was not asserted against Amazon until August 26, 2022. *See* First Am. Compl., *Swarm Tech. LLC v. Amazon.com Inc.*, No. 2:21-cv-00438 (D. Ariz. Aug. 26, 2022), ECF No. 102; Ex. 1033 ('777 patent claim charts attached to the Amended Complaint). Amazon had no reason to file this IPR until Swarm asserted the '777 patent against Amazon. Amazon gained no advantage from Juniper's IPR2021-01317, which asserted completely different art. Rather, Amazon applies the same art here that it did in its previous IPRs against the related, and much earlier asserted, '004 and '275 patents. This art has never been applied against the '777 patent.

Also, Swarm's co-pending district court case against Amazon is stayed pending its IPRs of the related '004 and '275 patents. *See* Ex. 1034. When the Board issues its decisions in those decisions, the parties, alone or jointly, may seek to extend the stay pending this IPR. *Id.*

VIII. Mandatory Notices under 37 C.F.R. § 42.8

A. Real Parties-in-Interest

The real parties-in-interest are Petitioner Amazon.com, Inc. and Amazon Web Services, Inc. No other party had access to, control over, or contributed to

⁹ IPR2022-00283 and -00633.

funding of the preparation or filing of this Petition.

B. Related Matters

Public PAIR indicates that U.S. Patent Application Nos. 15/852,480 (issued as the '275 Patent); 14/340,332, filed July 24, 2014 (issued as U.S. Patent No. 9,852,004); and 16/788,122, filed February 11, 2020 (currently pending), are related to the '777 patent, which issued from Application No. 13/750,696, filed January 25, 2013. The '004 patent is at issue in currently pending IPR2021-01317 (filed by Juniper) and IPR2022-00283 (filed by Amazon). The '275 patent is at issue in currently pending IPR2021-00141 (filed by Juniper) and IPR2022-00633 (filed by Amazon).

The '777 Patent is known to be at issue in the following matters:

- *Juniper Networks, Inc. and Apstra, Inc. v. Swarm Technology LLC*, No. 3:20-cv-03137 (N.D. Cal.), filed May 7, 2020.
- *Juniper Networks, Inc. v. Swarm Technology LLC*, IPR2021-01317, filed July 26, 2021, identifies different grounds from this Petition. The Board denied institution. Exs. 1021, 1032.
- *Swarm Technology, LLC v. Amazon.com, Inc. and Amazon Web Services, Inc.*, No. 2:21-cv-00438 (D. Ariz.), filed March 15, 2021, with an Amended Complaint adding the '777 patent filed on August

26, 2022. This case is currently stayed per the parties' stipulation, pending Amazon's IPRs of the '275 and '004 patents.

C. Grounds for Standing

Petitioner certifies that the '777 patent is available for *inter partes* review and Petitioner is not barred from requesting this proceeding.

D. Lead and Backup Counsel and Service Information

Pursuant to 37 C.F.R. §§ 42.8(b)(3), 42.8(b)(4), and 42.10(a), Petitioner designates the following lead counsel:

- Jonathan M. Strang (Reg. No. 61,724), jonathan.strang@lw.com, Latham & Watkins LLP; 555 Eleventh Street, NW, Ste. 1000; Washington, DC 20004-1304; 202.637.2362 (Tel.); 202.637.2201 (Fax).

Petitioner also designates the following backup counsel:

- Adam M. Greenfield (Reg. No. 60,331), adam.greenfield@lw.com, Latham & Watkins LLP; 555 Eleventh Street, NW, Ste. 1000; Washington, D.C. 20004-1304; 202.637.2308 (Tel.); 202.637.2201 (Fax).
- David Zucker (Reg. No. 74,095), david.zucker@lw.com, Latham & Watkins LLP; 555 Eleventh Street, NW, Ste. 1000; Washington, D.C. 20004-1304; 202.637.2219 (Tel.); 202.637.2201 (Fax).

- Kimberly Q. Li (Reg. No. 74,713), kimberly.li@lw.com, Latham & Watkins LLP; 200 Clarendon Street, Boston, MA 02116; 617.880.4529 (Tel.); 617.948.6001 (Fax).

Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney from Petitioner is attached.

Petitioner consents to electronic service.

E. Fee for *Inter Partes* Review

The Director is authorized to charge the fee specified by 37 C.F.R.

§ 42.15(a) to Deposit Account No. 506269.

IX. Conclusion

For the reasons set forth above, Petitioner Amazon respectfully requests *inter partes* review of the challenged claims of the '777 patent.

Respectfully submitted,

Dated: December 21, 2022

By: / Jonathan M. Strang /

Jonathan M. Strang (Reg. No. 61,724)
jonathan.strang@lw.com
Latham & Watkins LLP
555 Eleventh Street, NW, Ste. 1000
Washington, D.C. 20004-1304
Telephone: 202.637.2200
Fax: 202.637.2201

*Counsel for Petitioner Amazon.com, Inc.
and Amazon Web Services, Inc.*

Appendix: Claim Listing

The following claim listing adds letters to the claim elements for clarity:

1. [preamble] An apparatus for parallel processing of a large computing requirement, the apparatus comprising:
 - [a] a central processing unit (“CPU”);
 - [b] a task pool in electronic communication with the CPU; and
 - [c.1] a first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent
 - [c.2] configured to proactively retrieve, from the task pool, without requiring an instruction from the CPU, a matching task for the solidarity cell to process;
 - [d] wherein the CPU populates the task pool by dividing the requirement into one or more threads and placing the threads in the task pool, each thread comprising one or more tasks, and the matching task being one of the tasks;
 - [e] wherein each task comprises a descriptor, the descriptor containing at least:
 - a function to be executed; and
 - a memory location of data upon which the function is to be executed;
 - [f] wherein the first agent is a data frame comprising: a source address, a destination address and a payload;
 - [g] wherein the first agent retrieves the matching task by:
 - [g.1] being dispatched by the first solidarity cell to the task pool, during which the source address is the first solidarity cell’s address, the destination address is the task pool’s address, and the payload comprises a list of functions the first solidarity cell is configured to perform;
 - [g.2] searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform; and
 - [g.3] returning to the first solidarity cell, during which the source address is the task pool’s address, the destination address is the first solidarity cell’s address, and the payload comprises the descriptor of the matching task.

2. The apparatus of claim 1 wherein the task pool notifies the CPU when the tasks of a thread are completed.
3. The apparatus of claim 1 wherein the tasks each comprise a task type selected from a set of task types, and wherein the first solidarity cell is configured to perform tasks of one or more of the task types.
4. The apparatus of claim 3 wherein the matching task is a task that is ready to be processed and has a task type that the first solidarity cell can perform.
5. The apparatus of claim 4 wherein the first agent retrieves the matching task by:
searching the task pool for a task that is ready to be processed and has a task type that the first solidarity cell can perform; and
identifying the matching task.
6. The apparatus of claim 1 wherein the matching task is a task that is ready to be processed and the function of the matching task can be performed by the first solidarity cell.
7. The apparatus of claim 6 wherein the first agent retrieves the matching task by searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform, and identifying the matching task.
8. The apparatus of claim 1 wherein the descriptor further contains a memory location where processed data is to be stored.
9. The apparatus of claim 1 wherein the descriptor is a data structure and the task contains a reference to the memory location of the descriptor.
10. The apparatus of claim 1 wherein the task pool occupies a region of physical memory.
11. The apparatus of claim 10 wherein the task pool is disposed in a hardware block dedicated to the task pool.
12. The apparatus of claim 1 further comprising a second solidarity cell comprising a second agent that proactively retrieves matching tasks from the task pool for the second solidarity cell to process, wherein the matching task for each solidarity cell

is a task in the task pool that is ready to be processed and can be performed by the solidarity cell.

13. The apparatus of claim 12 wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process.

14. An apparatus for parallel processing of a large computing requirement, the apparatus comprising:

- [a] a central processing unit (“CPU”);

- [b] a task pool in electronic communication with the CPU;

- [c.1] first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent

- [c.2] configured to proactively retrieve, from the task pool, a matching task for the solidarity cell to process; and

- [d] a second solidarity cell comprising a second agent

- [d.2] that proactively retrieves matching tasks from the task pool for the second solidarity cell to process,

- [e] wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell;

- [f] wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process; and

- [g] wherein each agent comprises a source address, a destination address, and a payload, and

- [h] wherein each agent retrieves a matching task by:

- [h.1] being dispatched by its solidarity cell to the task pool, during which the source address is its solidarity cell’s address, the destination address is the task pool’s address, and the payload comprises a list of functions the agent’s solidarity cell is configured to perform;

- [h.2] searching the task pool for a task that is ready to be processed and has a function that the agent’s solidarity cell can perform; and

- [h.3] returning to its solidarity cell, during which the source address is the task pool’s address, the destination address is the agent’s solidarity cell’s address, and the payload comprises a descriptor of the matching task.

CERTIFICATE OF COMPLIANCE WITH 37 C.F.R. § 42.24

I hereby certify that this Petition complies with the word count limitation of 37 C.F.R. § 42.24(a)(1)(i) because the Petition contains a total of 10,294 words as calculated by Microsoft Word's word-count feature. This total excludes the cover page, signature block, and the parts of the Petition exempted by 37 C.F.R. § 42.24(a)(1).

By: / Jonathan M. Strang /

Jonathan M. Strang (Reg. No. 61,724)
jonathan.strang@lw.com
Latham & Watkins LLP
555 Eleventh Street, NW, Ste. 1000
Washington, D.C. 20004-1304
Telephone: 202.637.2200
Fax: 202.637.2201

*Counsel for Petitioner Amazon.com, Inc.
and Amazon Web Services, Inc.*

CERTIFICATE OF SERVICE

The undersigned certifies that a complete copy of this Petition for *Inter Partes* Review of U.S. Patent No. 9,146,777 and all Exhibits and other documents filed together with this Petition were served on the official correspondence address for the patent shown in Patent Center:

Daniel Pote (Reg. No. 43,011)
Eugene Vamos (Reg. No. 68,004)
Jennings, Strouss and Salmon, PLC
One East Washington Street, Suite 1900
Phoenix, AZ 85004-2554
Phone: 602.262.5832
Phone: 520.222.6879

via FEDERAL EXPRESS next business day delivery, on December 21, 2022.

By: / Jonathan M. Strang /

Jonathan M. Strang (Reg. No. 61,724)
jonathan.strang@lw.com
Latham & Watkins LLP
555 Eleventh Street, NW, Ste. 1000
Washington, DC 20004-1304
Telephone: 202.637.2200
Fax: 202.637.2201

*Counsel for Petitioner Amazon.com, Inc.
and Amazon Web Services, Inc.*