

**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**

**DECLARATION OF DAVID LOWENTHAL IN SUPPORT OF  
PETITION FOR *INTER PARTES* REVIEW  
OF U.S. PATENT NO. 9,146,777**

**Table of Contents**

|       |                                                                                                               |    |
|-------|---------------------------------------------------------------------------------------------------------------|----|
| I.    | Introduction and Qualifications .....                                                                         | 2  |
| II.   | Summary Of Materials Reviewed And Considered .....                                                            | 5  |
| III.  | Legal Understanding .....                                                                                     | 7  |
| IV.   | The '777 Patent .....                                                                                         | 11 |
| A.    | Summary of the '777 Patent .....                                                                              | 11 |
| V.    | Level of Skill in the Art .....                                                                               | 12 |
| VI.   | Claim Construction .....                                                                                      | 13 |
| B.    | “solidarity cell” .....                                                                                       | 13 |
| C.    | “thread” .....                                                                                                | 16 |
| VII.  | Overview of the Technology and Prior Art .....                                                                | 18 |
| A.    | Technology Background.....                                                                                    | 18 |
| 1.    | Distributed Computing .....                                                                                   | 18 |
| 2.    | Networking and The OSI Model .....                                                                            | 24 |
| VIII. | Petition Grounds and Prior Art Combinations.....                                                              | 30 |
| A.    | Overview of the Prior Art And Other Evidence .....                                                            | 30 |
| 1.    | Lurie (Ex. 1007) .....                                                                                        | 30 |
| 2.    | Ethernet Specification (Ex. 1009) .....                                                                       | 37 |
| 3.    | Andrews (Ex. 1011) .....                                                                                      | 41 |
| 4.    | Lippett (Ex. 1031) .....                                                                                      | 43 |
| IX.   | Ground 1: Claims 1-10 and 12-14 Would Have Been Obvious over<br>Lurie in View of Ethernet Specification ..... | 44 |
| A.    | Reasons to Implement Lurie Using Ethernet Specification ..                                                    | 44 |
| B.    | Independent Claim 14.....                                                                                     | 48 |
| 1.    | Preamble .....                                                                                                | 48 |
| 2.    | Elements 14[a]-[b]: CPU and task pool .....                                                                   | 51 |
| 3.    | Elements 14[c.1], [d.1], [g]: solidarity cells and their<br>agents .....                                      | 58 |
| 4.    | Elements 14[c.2], [d.2], [h] .....                                                                            | 64 |

|     |                                                                                                    |     |
|-----|----------------------------------------------------------------------------------------------------|-----|
| a.  | Element 14[h.1]: dispatch agent to task pool....                                                   | 70  |
| b.  | Element 14[h.2]: search the task pool.....                                                         | 76  |
| c.  | Element 14[h.3]: return to cell .....                                                              | 82  |
| 5.  | Element 14[e]: matching task in task pool.....                                                     | 84  |
| 6.  | Element 14[f]: each cell sends agent when not<br>processing a task.....                            | 84  |
| B.  | Independent claim 1 .....                                                                          | 90  |
| 1.  | Preamble .....                                                                                     | 90  |
| 2.  | Elements 1[a]-[b] and [d]: CPU populates the task pool<br>.....                                    | 90  |
| 3.  | Element 1[e]: tasks have a task descriptor .....                                                   | 94  |
| 4.  | Elements 1[c.1]-[c.2] and [f]-[g]: first solidarity cell<br>with agent retrieves task .....        | 95  |
| C.  | Dependent claims 2-13 .....                                                                        | 96  |
| 1.  | Claim 2: thread completion notification.....                                                       | 97  |
| 2.  | Claim 3: task types .....                                                                          | 97  |
| 3.  | Claims 4-7: matching task ready to be processed .....                                              | 98  |
| 4.  | Claim 8: task descriptor data storage location .....                                               | 99  |
| 5.  | Claim 9: task descriptor is a data structure .....                                                 | 100 |
| 6.  | Claim 10: task pool in physical memory.....                                                        | 100 |
| 7.  | Claim 12: second solidarity cell and agent .....                                                   | 101 |
| 8.  | Claim 13: send agent when no task to process.....                                                  | 101 |
| X.  | Ground 2: Claim 11 Would Have Been Obvious Over Lurie,<br>Ethernet Specification, and Lippett..... | 102 |
| XI. | Conclusion.....                                                                                    | 104 |

**I. Introduction and Qualifications**

1. I, David Lowenthal, submit this declaration on behalf of Amazon.com, Inc. and Amazon Web Services, Inc. (“Petitioner”) to provide my opinion concerning the validity of all claims 1–14 of U.S. Patent No. 9,146,777 (“the ’777 Patent”) in support of the Petition for *Inter Partes* Review. I am being compensated at an hourly rate of \$450. My compensation is not dependent on the substance of my statements in this declaration or the outcome of this action.

2. I am more than eighteen years of age, and I am a citizen of the United States, currently residing in the state of Arizona.

3. In formulating my opinions, I have relied upon my knowledge, training, and experience in the relevant art. My qualifications are stated more fully in my *curriculum vitae*, which has been provided as Exhibit 1004. Here, I provide a brief summary of my qualifications.

4. I am a professor in the Department of Computer Science at The University of Arizona, and have been since 2009. I also served as Associate Head of the Department of Computer Science from 2011 through 2020, and I served as Interim Head from 2020 through 2022. Since 1996, I have been involved in teaching and academia in the field of computer science.

5. I received a Bachelor of Science in Computer Science from The University of California, Davis in 1990. I received a Master of Science in

Computer Science from The University of Arizona in 1992. I received a Doctor of Philosophy in Computer Science from The University of Arizona in 1996. My dissertation is titled, “Fine-Grain Parallelism and Run-Time Decision Making.”

6. The courses I teach and have taught include parallel and distributed computing, operating systems (graduate and co-convened), C++ and systems programming, and program design and development. I have advised students, including both undergraduate and graduate students, in areas of research such as high performance computing (HPC), supercomputing, task scheduling, cloud computing, transmission control protocol (TCP), distributed shared memory (DSM), and parallelism.

7. Relatedly, my own research interests cover parallel and distributed computing, performance analysis and modeling, and run-time systems. For example, I recently received a grant from the National Science Office of Advanced Cyberinfrastructure (OAC) on “OAC Core: Improving Utilization of High-Performance Computing Systems via Intelligent Co-Scheduling.” I previously received a research grant from the Department of Energy Exascale Computing Program (subcontract from Lawrence Livermore National Laboratory) on “Co-Scheduling Tasks and Jobs on Sets of Nodes on Supercomputers.” Over the course of my career, I have received grants for research regarding multiprocessor systems, data distribution, and parallel and distributed computing.

8. I have over 50 refereed conference and workshop publications and 18 refereed journal publications. These publications focus on, for example, parallel and distributed computing, HPC, supercomputing, scalability, TCP, messaging passing interface (MPI) for parallel programming, and DSM.

9. I have also served as a National Science Foundation Grant Review Panel Member in 1999, 2002–2004, 2010, 2012, and 2014–2016.

10. I have served as a Program Committee Member for the International Conference on Parallel Processing (ICPP) in 2007 and 2008; the International Conference on Parallel Architectures and Compilation Techniques (PACT) in 2009; the Symposium on Principles and Practice of Parallel Programming (PPOPP) in 2007; the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC) in 2006, 2009, 2011–2016, and 2018. Further, I served as Area Chair of Performance for SC 2012 and SC 2021, and Area Co-Chair of Performance and Modeling for ICPP 2011.

11. I also served as General Chair for the International Conference on Supercomputing in 2011; Program Co-Chair for the IEEE International Conference on Green Computing in 2010; and Co-Chair of the Workshop on High-Performance, Power-Aware Computing in 2005 and 2006.

**II. Summary Of Materials Reviewed And Considered**

12. All of the opinions contained in this Declaration are based on the documents I reviewed and my knowledge and professional judgment. In forming the opinions expressed in this Declaration, while drawing on my expertise in the field of parallel computing, I reviewed the following documents:

| <b>Ex.</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                              |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1001       | U.S. Patent No. 9,146,777 B2 (the “777 Patent”)                                                                                                                                                                                                                                                                                                                 |
| 1004       | Curriculum Vitae of David Lowenthal, 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”)                                                                                                                                                         |
| 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”)                                                                                                                                                                                                                                                                                               |
| 1016       | Steve Tang, <i>Advantages of Ethernet vs. SONET/SDH</i> (TC Communications White Paper, rev. 2021) (“Tang”)                                                                                                                                                                                                                                                     |
| 1017       | Matt Hamblen, <i>Q&amp;A: Ethernet’s Creator Describes Its Past, Future</i> , COMPUTERWORLD (May 16, 2003, 12:00 AM PST), <a href="https://www.computerworld.com/article/2570092/q-a--ethernet-s-creator-describes-its-past--future.html">https://www.computerworld.com/article/2570092/q-a--ethernet-s-creator-describes-its-past--future.html</a> (“Hamblen”) |

Declaration for *Inter Partes* Review of USP 9,146,777

| Ex.  | Description                                                                                                                                                                                                            |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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”)                                                                                                            |
| 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”)                                                                             |
| 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”)                                                                                                                                                                       |
| 1030 | U.S. Patent No. 10,592,275 B2 (the “’275 Patent”)                                                                                                                                                                      |
| 1031 | U.S. Patent Application Publication No. 2007/0220517 A1 (“Lippett”)                                                                                                                                                    |
| 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”)                                                                                                                   |
| 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, Algorithms and Data Structures: The Basic Toolbox</i> 81 (2008) (excerpt) (“Mehlhorn”)                                                           |

| Ex.  | Description                                                                                                                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1060 | <i>The OSI Model – What It Is; Why It Matters; Why It Doesn’t Matter</i> , printed from <a href="https://www.tech.faq.com/osi-model.html">https://www.tech.faq.com/osi-model.html</a> (last visited Dec. 13, 2022) |
| 1061 | <i>OSI Model: 7 Layers Explained in Computer Network – Javatpoint</i> , <a href="https://www.javatpoint.com/osi-model">https://www.javatpoint.com/osi-model</a> (last visited Dec. 13, 2022)                       |

13. My opinions are additionally guided by my appreciation of how a person of ordinary skill in the art would have understood the claims of the ’777 Patent at the time of the alleged invention, which I have been asked to assume is January 25, 2013, which counsel informs me is the earliest possible priority date for the ’777 Patent.

### III. Legal Understanding

14. I am not an attorney. However, I have been informed by counsel of certain legal standards and have applied those standards, described below, in formulating my opinions.

15. I understand that the claims of a patent define the limits of the patentee’s exclusive rights. I understand that to determine the scope of the claimed invention, courts may construe claim terms, the meaning of which the parties may dispute. I understand that claim terms should generally be given their ordinary and customary meaning as understood by a person of ordinary skill in the art (“POSITA”) at the time of filing of the patent application.

16. I understand that claims must be construed in light of and consistent with the intrinsic evidence. In this context, I understand that intrinsic evidence includes the claims themselves, the written disclosure in the specification, and the patent's prosecution history, including prior art that was considered by the United States Patent and Trademark Office. I understand that the specification is always highly relevant to the claim construction analysis and often is the single best guide to the meaning of a disputed term. I understand that extrinsic evidence may also be considered when construing claims and may include, for example, technical dictionaries, technical publications and books, treatises, and expert testimony, though the intrinsic record will be more persuasive as to the meaning of a particular term.

17. I understand that a prior art reference can render a patent claim obvious to a POSITA if the differences between the subject matter set forth in the patent claim and the prior art are such that the subject matter of the claim would have been obvious at the time the claimed invention was made. In analyzing obviousness, I understand that it is important to consider the scope of the claims, the level of skill of a POSITA in the relevant art, the scope and content of the prior art, the differences between the prior art and the claims, and so-called 'secondary considerations' of non-obviousness.

18. I understand that the claimed subject matter is obvious when a POSITA would have been motivated to combine pre-existing elements and that combination yields no more than what a POSITA would have been expected. I also understand that in assessing whether a claim is obvious, one must consider whether the claimed improvement is more than the predictable use of prior art elements according to their established functions. I understand that there need not be a precise teaching in the prior art directed to the specific subject matter of a claim because one can take account of the inferences and creative steps that a POSITA would employ.

19. I understand that obviousness cannot be based on the hindsight combination of components selectively culled from the prior art. I understand that in an obviousness analysis, neither the motivation nor the avowed purpose of the inventors controls the inquiry. Any need or problem known in the field at the time of the invention and addressed by the patent can provide a reason for combining elements. For example, I understand that it is important to consider whether there existed at the time of the invention a known problem for which there was an obvious solution encompassed by the patent's claims. I understand that known techniques can have obvious uses beyond their primary purposes, and that in many cases a POSITA can fit the teachings of multiple pieces of prior art together.

20. I understand that certain issues relating to validity must be judged from the perspective of a POSITA. I further understand that factors that may be considered in determining the level of skill of a POSITA include, but are not limited to, (1) the type of problems encountered in the art; (2) prior art solutions to those problems; (3) rapidity with which innovations are made; (4) sophistication of the technology; and (5) the educational level of active workers in the field.

21. I have been informed and understand that a POSITA is a hypothetical person who is presumed to be aware of all pertinent prior art, thinks along conventional wisdom in the art, and is a person of ordinary creativity. Importantly, this hypothetical person is of ordinary skill in the art to which the claimed subject matter pertains.

22. I understand that, when there is a reason to solve a problem and there are a finite number of identified, predictable solutions, a POSITA has good reason to pursue the known options within his or her technical grasp. I further understand that, if this leads to the anticipated success, it is likely the product not of innovation but of ordinary skill and common sense, which bears on whether the claim would have been obvious.

23. I understand that secondary considerations of nonobviousness can include, for example, evidence of commercial success of the invention, evidence of a long-felt need that was solved by an invention, evidence that others copied an

invention, or evidence that an invention achieved an unexpected result. I further understand that such evidence must have a nexus, or causal relationship to the elements of a claim, in order to be relevant.

#### **IV. The '777 Patent**

##### **A. Summary of the '777 Patent**

24. The '777 patent is titled “Parallel Processing With Solidarity Cells By Proactively Retrieving From A Task Pool A Matching Task For The Solidarity Cell To Process.” '777 Patent (Ex. 1001) at Cover.

25. 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 '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.

26. The patent states that 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.” ’777 patent claim 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.” ’777 patent claim 1, 5:30-60.

**V. Level of Skill in the Art**

27. A POSITA at the time of the earliest claimed priority date, January 25, 2013, would have had a bachelor’s degree in computer science, computer engineering, or a related field of study, and about five years of industry or academic research experience in parallel or distributed computing, system architecture, or an equivalent field. Alternatively, a POSITA would have had a post-graduate degree or additional formal education to compensate for less experience, and vice-versa.

28. At the time of the earliest claimed priority date, and at the time of the alleged invention, I met the qualifications to be a POSITA. Although my experience and education at that time exceeded that of a POSITA, I would have been personally familiar with the knowledge and capabilities of one of ordinary skill in the art based on my experiences as a professor in the Department of Computer Science. In particular, I would have been exposed to the knowledge and capabilities of a POSITA based on my responsibilities as a professor, which included overseeing the research of and facilitating the education of both undergraduate and graduate students, and based on my exposure through

professional organizations and conferences to the experiences of those working in industry.

29. I have applied the perspective of a POSITA in forming and expressing my opinions. Unless otherwise stated, my testimony below refers to the knowledge of one of ordinary skill in the art at the relevant time.

## **VI. Claim Construction**

30. Other than the terms that counsel discussed in the subsections below, I have applied the plain and ordinary meanings of the claim terms in the '777 patent, as a POSITA would have understood them in the context of that patent in approximately January 2013. However, my analysis does not depend on any particular or precise construction of a term, because I did not find that to be necessary to provide my opinions regarding the relationship between the prior art and the claims of the '777 patent. The opinions that I provide in this declaration would remain true under any reasonable construction of the claim terms in the '777 patent.

31. As I stated in the previous paragraph, counsel asked me to address two claim terms: “solidarity cell” and “thread.” I discuss those terms in the following paragraphs.

### **B. “solidarity cell”**

32. Claim 1, for example, recites the term “solidarity cell.”

a first ***solidarity cell*** in electronic communication with the task pool, the first ***solidarity cell*** comprising a first agent

configured to proactively retrieve, from the task pool, without requiring an instruction from the CPU, a matching task for the ***solidarity cell*** to process;

. . .

wherein the first agent retrieves the matching task by:

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;

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

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.

33. The term “solidarity cell” did not (and does not) have a commonly accepted meaning. A POSITA reading the '777 patent, however, would have concluded that a “solidarity cell,” as that term is used in the patent, is any processor or processing system that performs the features recited in the claims.

34. I did not find any clearly proposed definition of the term in the '777 patent. The '777 patent simply states, in effect, that a solidarity cell may be any type of processor, general or special-purpose:

be executed by a central processing unit (“CPU”). A solidarity cell may be a general- or special-purpose processor, and therefore may have the same or different instruction set architecture and microarchitecture compared to the CPU and other solidarity cells in the apparatus. As with any computer system, the information to be processed is contained within one or more memory units. The CPU divides the information into

’777 patent 1:62-2:1. The patent continues, confirming the above and also indicating that the solidarity cell may be a processing system, too, such as a “standalone computer.”

A solidarity cell 12A . . . n is an independent computational unit capable of executing one or more tasks. The solidarity cell 12A . . . n may be a microcontroller, a microprocessor, or a standalone computer such as a machine in a Beowulf cluster. Preferably, the solidarity cell 12A . . . n is a general- or special-purpose coprocessor configured to supplement the functions of the CPU 11. A general-purpose coprocessor may be capable of performing all of the functions of the CPU 11, and may perform all of the functions or a limited range of functions as needed by the architecture of the system 10. A special-purpose processor may be a dedicated hardware module designed to perform a specialized task, or it may be a general-purpose processor configured to perform specialized tasks such as graphics processing, floating-point arithmetic, or data encryption. In the preferred embodiment, any solidar-

’777 patent 3:30-45.

35. I also note that the use of the term “solidarity cell” in the claims of the ’777 patent is similar to the use of the generic term “co-processor” in the claims of the ’275 and ’004 patents (Exhibits 1029-1030). I see nothing in the ’777 patent that would have led a POSITA to ascribe some special meaning to “solidarity cell.”

It is merely any type of processor or computer that is programmed to perform the functions recited by the claims.

**C. “thread”**

36. Claims 1 and 2, for example, recites “thread(s)”

Claim 1:

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

a central processing unit (“CPU”);

...

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

...

Claim 2:

2. The apparatus of claim 1 wherein the task pool notifies the CPU when the tasks of a *thread* are completed.

37. The '777 patent further states in the specification that

structures. The task pool 13 contains one or more task threads 21. A task thread 21 represents a computational task that a component of the larger requirement imposed on the CPU 11. In one embodiment, the CPU 11 may initialize and then populate the task pool 13 by dividing the large requirement into concurrently executable threads 21 and places the threads 21 in the task pool 13. A thread 21 is composed of one or more tasks 22. A task 22 may have a task type and a descriptor. The task type is an indicator that describes which solidarity cells 12A . . . n are capable of performing the task 22. The task pool 13 may also use the task type to prioritize tasks 22 having the same type. In one embodiment, which may be advantageous for a large-scale system 10, the task pool 13 may maintain a prioritization table (not shown) that documents the solidarity cells 12A . . . n present in the system 10, the types of tasks 22 each is capable of performing, and whether or not each cell 12A . . . n is presently processing a task 22. The task pool 13

'777 patent 4:20-38, and

available tasks that match the cell's instruction set architecture. The solidarity cells may execute the task threads sequentially or in parallel, and independently or collaboratively, depending on recipes provided by the CPU. Interdependent tasks within the task pool may be logically combined as needed by the recipe. The task pool notifies the CPU when a task thread is completed.

'777 patent at 2:12-19.

38. Thus, the '777 patent claims divide a "large computing requirement" (claim 1 preamble), i.e., a large computational job/task, into one or more "threads," i.e., generally smaller computational jobs/tasks, and those threads are composed of one or more generally smaller "tasks." In other words, the '777 patent uses the term "thread" to mean all or a portion of a computing requirement, and that thread is composed of one or more computational tasks. I also note that a thread "may 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. But the ’777 patent does not require a thread to have such a recipe.

## **VII. Overview of the Technology and Prior Art**

### **A. Technology Background**

#### **1. Distributed Computing**

39. Since at least the 1960s, computer scientists have used distributed (or parallel)<sup>1</sup> processing systems. Andrews (Ex. 1011) at v-vii, 2-3, 31-32. These systems were recognized as being beneficial due to their ability to increase processing power and efficiency. *Id.* Such systems used multiple processors to accomplish tasks that were otherwise too time-consuming, complex, or specialized for a single processor to handle. *Id.* at 3. For example, a large job could be split into multiple parts, and then each parallel processor within a system could work on one part simultaneously. As a result, the complete job could be performed much more quickly than if the job were run on a traditional single pipeline processing architecture. Because parallel processing allowed large jobs to be completed more quickly, such systems could perform large and complex jobs that might otherwise

---

<sup>1</sup> The terms “distributed” and “parallel” are interrelated. In some literature, the two terms may hold distinct meanings. However, for the purposes of the following discussion, the concepts that I discuss apply to both distributed and parallel processing systems.

have been too time-consuming to be practically performed on a non-parallel processing system.

40. In the simplest form of a distributed processing system, such as those developed in the early 1960s and 1970s, a central task scheduler is coupled to multiple processors. The task scheduler receives tasks that the system desires to be completed, and then assigns tasks to each processor to complete. However, computer scientists recognized that this was often an inefficient use of the central task scheduler. If a processor is dedicated solely to distributing tasks to other processors, then that processor is not able to contribute to execution of processing tasks. *See, e.g.,* Chow (Ex. 1008) at [0004] (“there is a need for a method of load balancing in a multiprocessor system, that more evenly balances the load among the processors than traditional methods, ***does not burden the central distributor***, and reduces the latency in task-loading”). Therefore, those of ordinary skill in the art recognized that it would be beneficial to free the central unit from the burden of performing affirmative scheduling, while still using multiple parallel processors to efficiently execute tasks. In this way, the system as a whole would minimize the time that any given processor spent either idle or waiting for a task. However, some mechanism of load balancing was still needed, otherwise a single processor could find itself flooded with too many tasks to execute.

41. One well-recognized and efficient distributed processing architecture utilizes what's known as a "bag of tasks" paradigm. In a "bag of tasks," all jobs to be stored are held in a shared data structure – equivalent to a 'bag' – that multiple processors have access to. Each processor, whenever it has availability, checks the "bag" for tasks that need to be performed. If there are tasks in the bag, the processor with availability takes a task and executes it. This technique was well-known at least as early as 2000, when it appeared in the Andrews textbook. Andrews (Ex. 1011) at 132-39. The "bag of tasks" paradigm offered several benefits, such as simple implementation, scalability (e.g., "one can use any number of processors merely by varying the number of workers"), and opportunity for load balancing as long as there are more tasks than workers. *Id.* at 132-33.

42. A decentralized "bag of tasks" system often still had a central unit, but the unit did not need to actively distribute tasks to other processors. Instead, the central unit, or bag, simply received a task, divided that task into smaller subtasks if necessary, and added those subtasks or tasks to a task pool.

43. In the "bag of tasks" paradigm, when the processors execute a task, the processors can also produce smaller, resulting tasks that are returned to the pool for further processing. As Andrews explains, "executing one task will usually lead to producing two smaller tasks, which a worker [solidarity cell] needs to put back into the bag." Andrews (Ex. 1011) at 135. An exemplary matrix multiplication

application implementing a parallel worker model has “[t]he worker generat[ing] a new task to replace the one removed.” Carriero (Ex. 1028) at 44 (further explaining that “the master [central unit] need not and in some cases cannot generate all tasks”).

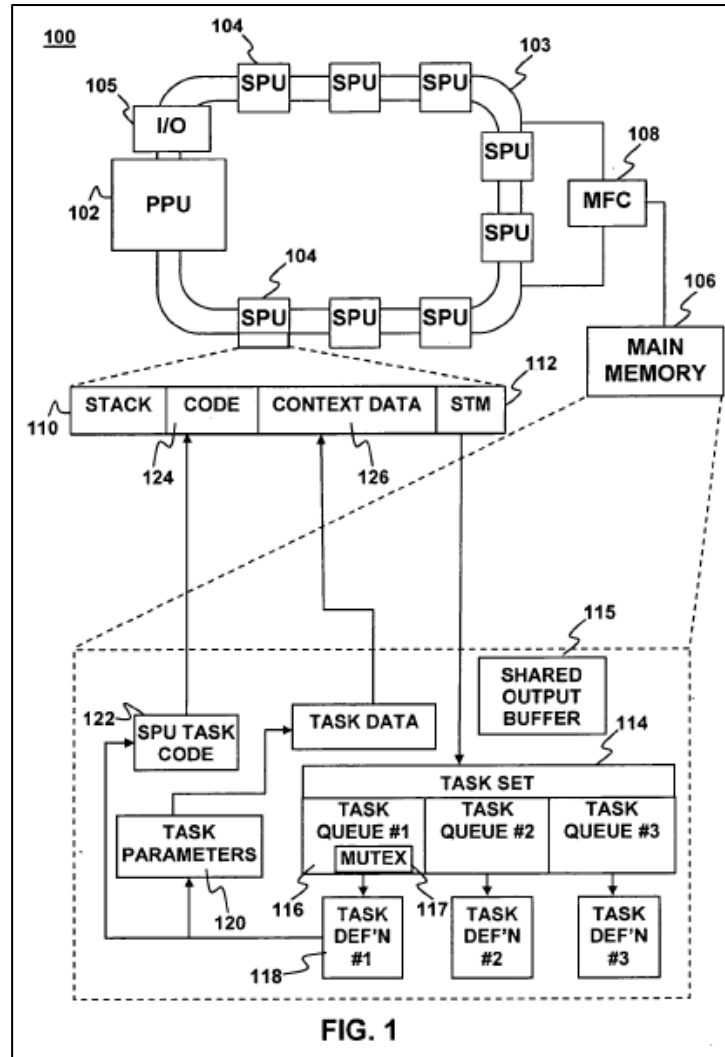
44. Systems that implement the “bag of tasks” paradigm require some way for the task pool and the workers to communicate with each other regarding what tasks in the pool are available for retrieval and what tasks the worker retrieves. There are generally two categories of system models that manage such processes and synchronization: procedure-oriented systems, which rely upon shared variables and data structures, and message-oriented systems, which rely upon message passing between processes. Lauer (Ex. 1027) at 2.

45. In procedure-oriented systems, “[c]ooperation among processes is achieved by some form of locks, semaphores, monitors, or other synchronizing data structures” whereby “a process attempts to claim a lock, and may be forced to wait on a queue until some other process releases it.” Lauer (Ex. 1027) at 6. In this manner, “[d]ata is shared directly among processes, and processes tend to lock only small parts of the data structures . . . .” *Id.* In contrast, in message-oriented systems, “[d]ata structures which must be manipulated by more than one process are passed (by reference) in messages.” *Id.* at 4. A “message” is another “data structure meant for sending information from one process to another,” and “typically contains a

small, fixed area for data which is passed by value and space for a pointer to larger data structures which must be passed by reference.” *Id.*

46. Procedure-oriented and message-oriented systems are duals of each other. Lauer (Ex. 1027) at 2. As duals, they are logically identical to each other. *Id.* As duals, a system constructed according to one model also can be mapped directly onto the other. *Id.*

47. One example of the “bag of tasks” paradigm can be seen in Bates. Bates uses a power processing unit (“PPU”) as its central unit, with synergistic processing units (“SPUs”) serving as the parallel processors that retrieve tasks from the shared pool or bag for execution. Bates (Ex. 1013) at Fig. 1 (below).



48. It was known in the art that multiprocessor systems did not have to be limited to a wired connection, as in Bates, but could be implemented over a network. Andrews (Ex. 1011) at 31-33. In fact, it was “[t]he introduction of computer networks in the 1970s [that] led to the development of distributed systems.” *Id.* at 32. Furthermore, “[t]he invention of the Ethernet in the late 1970s greatly accelerated the pace of activity.” *Id.*

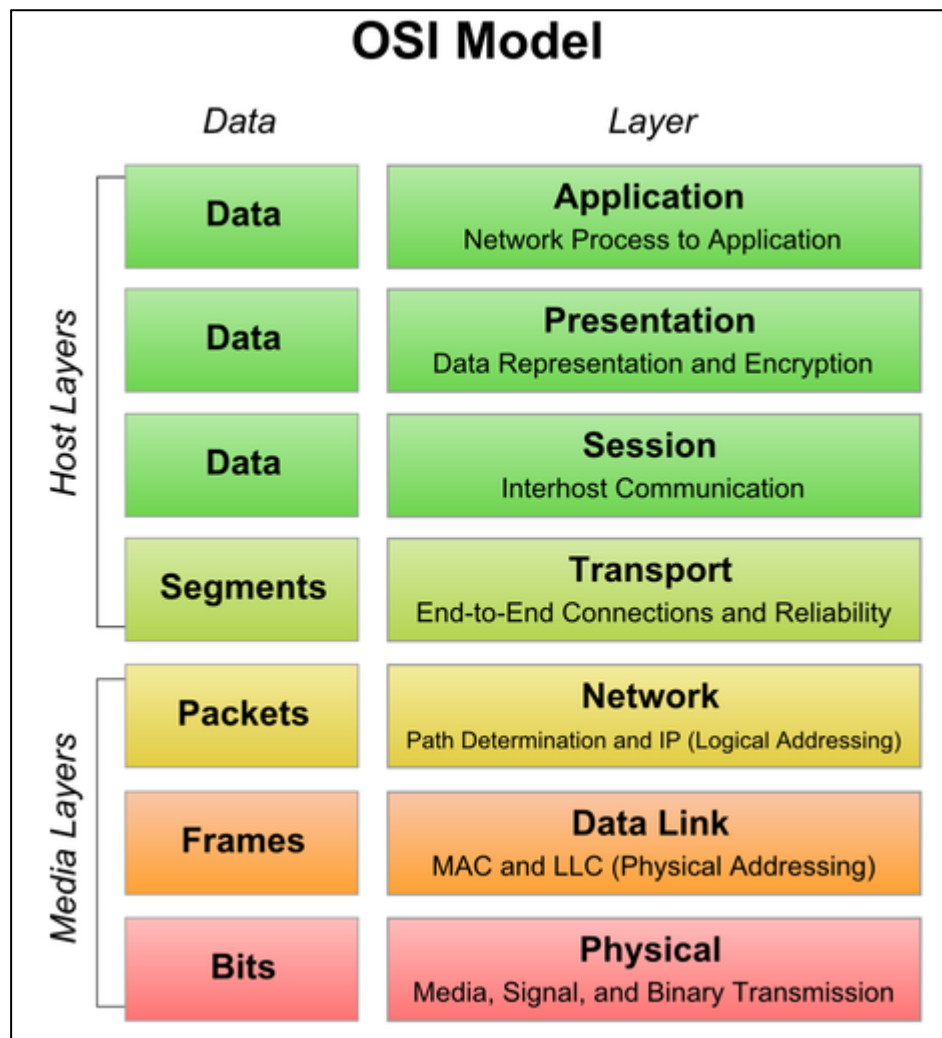
49. Accordingly, for processors to retrieve tasks, such systems often utilized network packets or data frames, described in the '777 patent as agents, to transmit data (*i.e.*, task details or processor characteristics regarding what tasks the processor could handle) between the processor and the task pool. Typically, to be functional, the agent included a source address (where it came from), destination address (where it was going), and data payload containing any relevant information.

## **2. Networking and The OSI Model**

50. Distributed or parallel processing systems can be implemented over a network. As I previously explained, it was the introduction of computer networks in the late 1970s and early 1980s that gave rise to distributed programming (processes interacting through message passing rather than reading and writing shared variables). Andrews (Ex. 1011) at v; Lauer (Ex. 1027) at 3-6 (describing message passing). The processors in distributed or parallel processing systems must be able to communicate with one another, which is typically achieved through standard communication protocols.

51. During the 1970's through the early 1980's, the International Standardization Organization (ISO) developed the Open System Interconnection ("OSI") network model that serves as a reference model for network protocol design and unification. Day (Ex. 1023) at 1334. In other words, the OSI model

describes how information or data from a software application of one device moves through a physical medium to a software application of another device in the network. The OSI model divides network communication into seven levels, or layers, each of which uses the services provided by an adjacent lower layer and in turn provides services to an adjacent upper layer. *Id.* at 1338-40; Meyer (Ex. 1024) at 18-19.



Ex. 1060 at 3.

52. The physical layer, the lowest layer, supports delimiting and encoding unstructured data bits onto the physical medium. Protocols at this layer define the electrical and physical specifications such as pin layout, cable specifications, signal timing, or voltages of the data connection, such as through a wire or cable.

53. The data link layer supports physical addressing and packages the unstructured data into data frames for node-to-node data transfer. The data link layer also provides flow and error control atop the physical layer. Common protocols for the data link layer include Ethernet, Token Ring, and High-Level Data Link Control (“HDLC”). For example, the protocol specification for Ethernet explains that it “is primarily for use in such areas as . . . distributed data processing . . . ,” and “provides detailed definitions of the lowest two layers of an overall network architecture,” which is “what is generally referred to as a link-level facility.” Ethernet Specification (Ex. 1009) at 1-2 (emphasis removed).

54. The network layer supports path determination and logical addressing to deliver the data frames to their intended destinations based on the addresses contained within the data frame. The network layer provides a means of transferring variable length data sequences, or datagrams, to nodes within the same network. Common protocols for the network layer include Internet Protocol (“IP”), which encompasses Internet Control Message Protocol (“ICMP”), IPsec, and Internet Group Management Protocol (“IGMP”). For example, the protocol

specification for IP explains that it “is designed for use in interconnected systems of packet-switched computer communication networks,” and “provides for transmitting blocks of data called datagrams from sources to destinations” and also “for fragmentation and reassembly of long datagrams, if necessary, for transmission through ‘small packet’ networks.” Internet Protocol (Ex. 1018) at 1.

55. The transport layer supports end-to-end connections between machines. The transport layer facilitates communication between processes on different machines, and regulates the transfer of datagrams or network packets between systems and hosts. Common protocols for the transport layer include Transmission Control Protocol (“TCP”) or User Datagram Protocol (“UDP”). For example, the protocol specification for UDP explains that UDP “is defined to make available a datagram mode of packet-switched computer communication in the environment of interconnected set of computer networks.” UDP Specification (Ex. 1019) at 1.

56. The session layer supports inter-host communications, and facilitates ongoing communications, or a session, to occur across a network. The session layer allows applications to exchange data for the duration of the session, and handles session setup and termination, message exchanges, and authentication. Common protocols for the session layer include Point-to-Point Tunneling Protocol (“PPTP”) and NetBios.

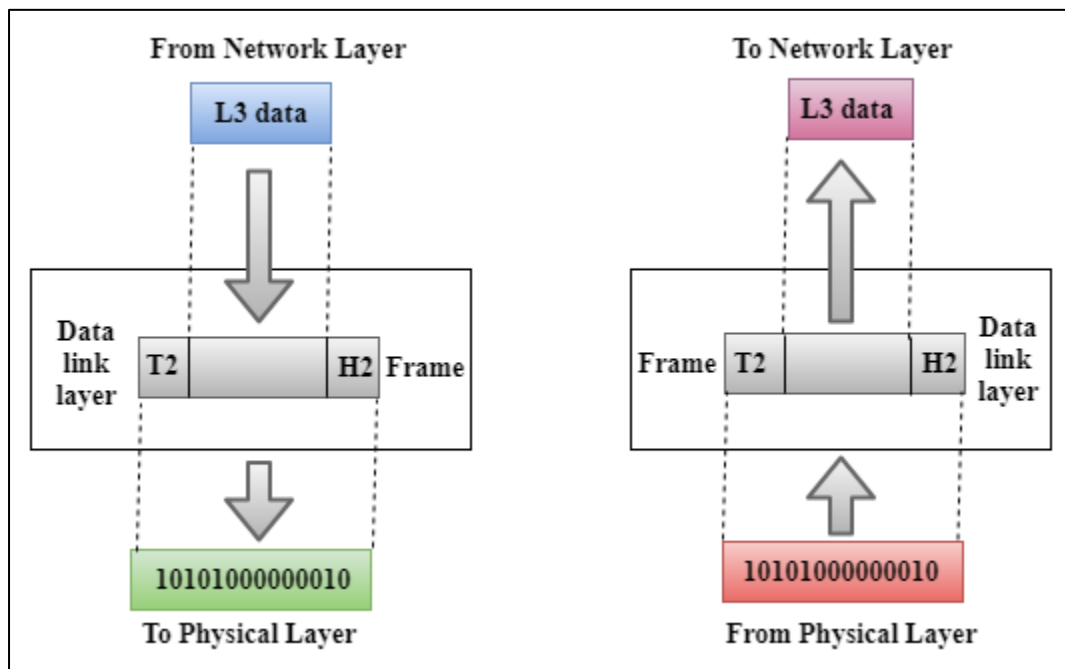
57. The presentation layer supports data representation and encryption. The presentation layer, sometimes referred to as the syntax layer, translates or formats data to one used by the application layer. This includes, for example, character code translation, data conversion, data compression, and data encryption. Common protocols for the presentation layer include Secure Sockets Layer (“SSL”) and Transport Layer Security (“TLS”).

58. The application layer supports application and end-user processes, such as file transfer, e-mail, and software services. Common protocols for the application layer include File Transfer Protocol (“FTP”), Hypertext Transfer Protocol (“HTTP”), Post Office Protocol version 3 (“POP3”), and Secure Shell (“SSH”).

59. As end-user data from a sending device passes down through each layer (application layer to physical layer), each layer adds a header. *See, e.g.*, Ethernet Specification (Ex. 1009) at 20, Fig 6-1 (depicting destination address, source address, and type fields that form the header of the data frame); Internet Protocol (Ex. 1018) at 2 (“The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations.”); UDP Specification (Ex. 1019) at 1, (depicting UDP header format), 2 (“The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length.”), 3 (further explaining that

“[t]he UDP module must be able to determine the source and destination internet addresses and the protocol field from the internet header”). In the case of the data link layer, both a header and trailer are added.

60. Subsequently, at the physical layer of the sending device, the data is passed to the physical layer of the receiving device. Next, just as how headers are added as the data moves downward through the layers on the sending device’s side, the headers are removed as the data moves upward through the layers on the receiving device’s side. Below is a diagram depicting how this occurs at the data link layer, between the physical layer and the network layer.



Ex. 1061 at 5.

### **VIII. Petition Grounds and Prior Art Combinations**

61. For clarity, I have annotated the claim elements of the '777 Patent using numbered and lettered bracketed labels. I have included a full listing of the claims of the '777 Patent, with labels, in the Appendix at the end of this declaration. I will use these labels to refer to each element through this declaration.

62. I will discuss each element in further detail below, as well as explaining how the prior art discloses each element, or how a POSITA would consider the element obvious in view of what the prior art discloses.

63. In some sections of this declaration, I refer to my explanation of a different past or future claim element. I intend that reference to fully incorporate my explanation of that past or future element, as if it was copied and pasted in full to where I reference it.

#### **A. Overview of the Prior Art And Other Evidence**

##### **1. Lurie (Ex. 1007)**

64. U.S. Publication No. 2007/0124363 A1 ("Lurie") is titled "Instrument-Based Distributed Computing Systems." Lurie identifies Roy Lurie and Thomas Gaudette as the inventor. Lurie was published on May 31, 2007. *See* Lurie (Ex. 1007) at Cover.

65. I understand from counsel that Lurie is prior art to the '777 Patent.

66. Lurie discloses a networked, distributed computing system in which a job, which may include one or more tasks, is completed by one or more workers,

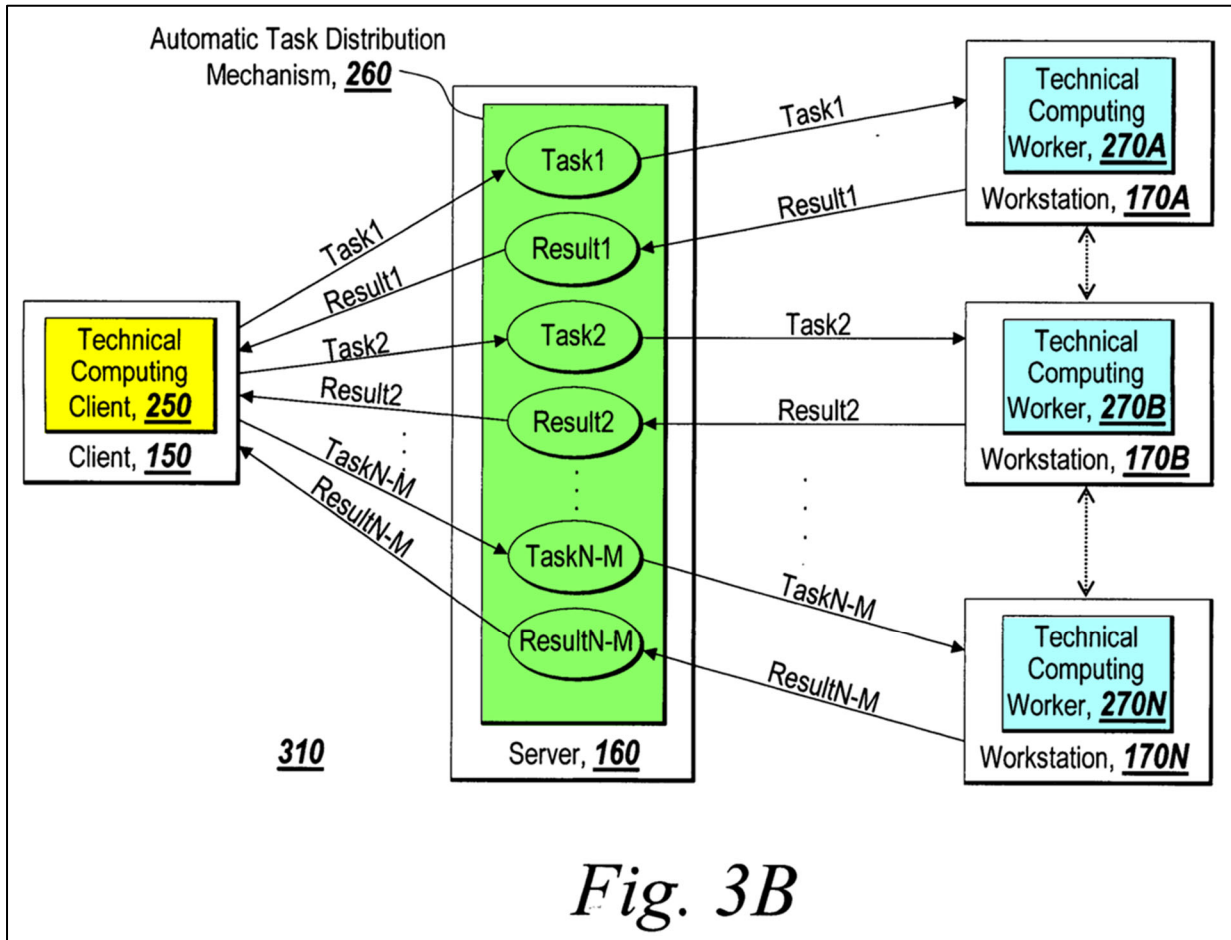
each of which executes a portion of the job and returns the results. *Id.* at [0005]. Lurie refers to this as “distributed execution of the job.” *Id.* Lurie’s solution “allows the use of instrument-based distributed system on a network to conduct the job and facilitate decreasing the time for executing the job.” *Id.*

67. As provided by way of background, Lurie’s disclosed system addresses the problem of larger and complex computations that are “resource intensive and time-consuming.” *Id.* at [0004] (“Therefore, it is desirable to perform a computation in a distributed manner when the computation becomes so large and complex that it cannot be completed in a reasonable amount of time on a single computer.”).

68. Lurie discloses various embodiments whereby the technical computing environment “dynamically distribute[s] tasks or jobs directly or indirectly, via the automatic task distribution mechanism, to one or more technical computing worker.” *Id.* at [0049]. In one such embodiment, the distributed processing system operates in an “automated task distribution mode.” *Id.* at [0071]-[0072] (describing this embodiment), Fig. 3B.

69. Below, I have provided an annotated version of Figure 3B of Lurie to indicate certain components of its distributed computing system: the technical computing client (“TCC,” annotated in yellow), the automatic task distribution

mechanism (annotated in green), and the technical computing workers (“TCWs,” annotated in blue).



70. The TCC is in communication with the automatic task distribution mechanism, which in turn is in communication with a number of TCWs. *Id.* at [0071].

71. In the “automated task distribution mode,”<sup>2</sup> the TCC submits one or more tasks to the automatic task distribution mechanism. *Id.* at [0072]. The TCC can submit individual tasks to the automatic task distribution mechanism, or a group of tasks, in the form of a job. *Id.* at [0043], [0047] (“A job is a logical unit of activities, or tasks that are processed and/or managed collectively.”), [0048].

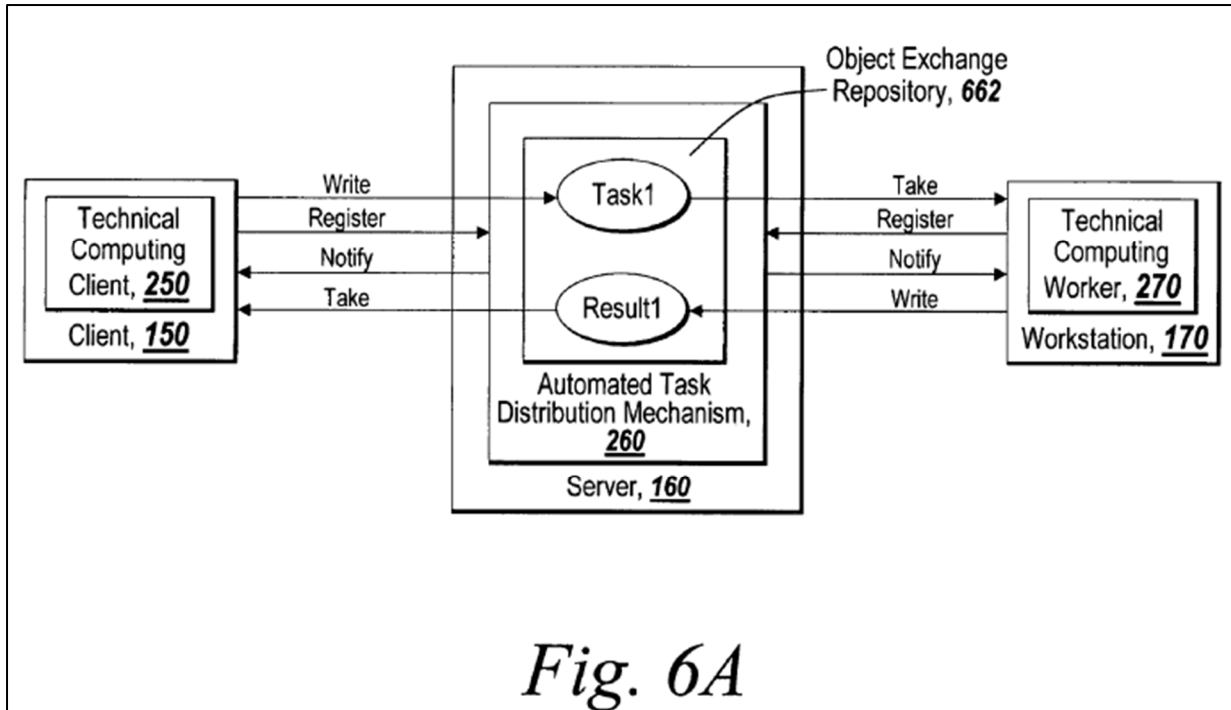
72. The automatic task distribution mechanism obtains the submitted tasks and makes them available to the TCWs. *Id.* at [0072]. Each TCW is configured to retrieve a task from the automatic task distribution mechanism, compute a result, and provide the result back to the automatic task distribution mechanism. *Id.* The TCW’s choice of a task can be based on operational and/or performance characteristics of the TCW. *Id.* at [0081]. The automatic task distribution mechanism then makes the results available to the TCC, and the TCC obtains the results of the computed tasks from the automatic task distribution mechanism at a time or method determined by the technical computing client. *Id.* at [0072].

---

<sup>2</sup> “Automatic task distribution” is, in part, a misnomer because, as explained in the following paragraphs, the TCWs retrieve tasks from the automatic task distribution mechanism; no component in the processing system is actively “distributing” or assigning tasks to the TCWs.

73. In the “automated task distribution mode,” the TCC “is not required to have any specific knowledge” of the TCWs, or their availability to perform technical computing of a task. *Id.* at [0071] (“the technical computing client 250 can choose not to specify the name of a particular technical computing worker to perform the task, and let the automated distribution mechanism distribute the task to any available technical computing worker 270A-270N”), [0061] (“the automatic task distribution mechanism provides a level of indirection between the technical computing client 250 and the technical computing worker 270”). Lurie explains that this “eases the distributed programming and integration burden on the technical computing client 250.” *Id.* at [0062].

74. The automated task distribution mechanism can comprise an “object exchange repository.” *Id.* at [0090], Fig. 6A (below).



75. Each TCW “registers with the object exchange repository 662 to be notified when a task object appears in the object exchange repository.” *Id.* at [0093]. When the TCC deposits its first task or set of tasks (e.g., a job), the ATD notifies registered TCWs that tasks are available. Each registered TCW will perform a “take” operation (e.g., “getTask” in Lurie) to retrieve a task that the TCW can perform, if it finds one. *Id.* at [0093], Fig. 6A. Any TCW may “register[]” with the object exchange repository so that the TCW is available to the system for retrieving and completing tasks. *Id.* at [0093].

76. Lurie also discloses a “batch mode of the automated task distribution” where a job manager obtains a job—comprising multiple tasks—from the TCC, submits the job and each task associated with the job to the automatic task

distribution mechanism for computing by the TCWs, and likewise obtains the set of results of each task associated with the job for the completed job and provides the set of results related to that job and provides that set of results to the TCC. *Id.* at [0073]-[0074],<sup>3</sup> Fig. 3C (below). There can be any number of job managers associated with any number of automatic distribution mechanisms, which “allows the distributed system to scale” and for the system to handle multiple TCCs distributing tasks. *Id.*

---

<sup>3</sup> Paragraphs [0071]-[0072] and Figure 3B describe the “automated task distribution mode.” Paragraphs [0073]-[0074] and Figures 3C and 3D provide further details specific to the batch mode, which reflect the same architecture and functionality described in the “automated task distribution mode.” *See, e.g.*, Lurie (Ex. 1007) at [0074] (“Technical computing worker 270A may take Task1 from the automatic task distribution mechanism 260, compute a Result1 for Task1 and provide the Result1 to the automatic task distribution mechanism 260.”). Thus, any foregoing citations to paragraphs [0071]-[0072] and Figure 3B similarly apply to the batch mode as well.

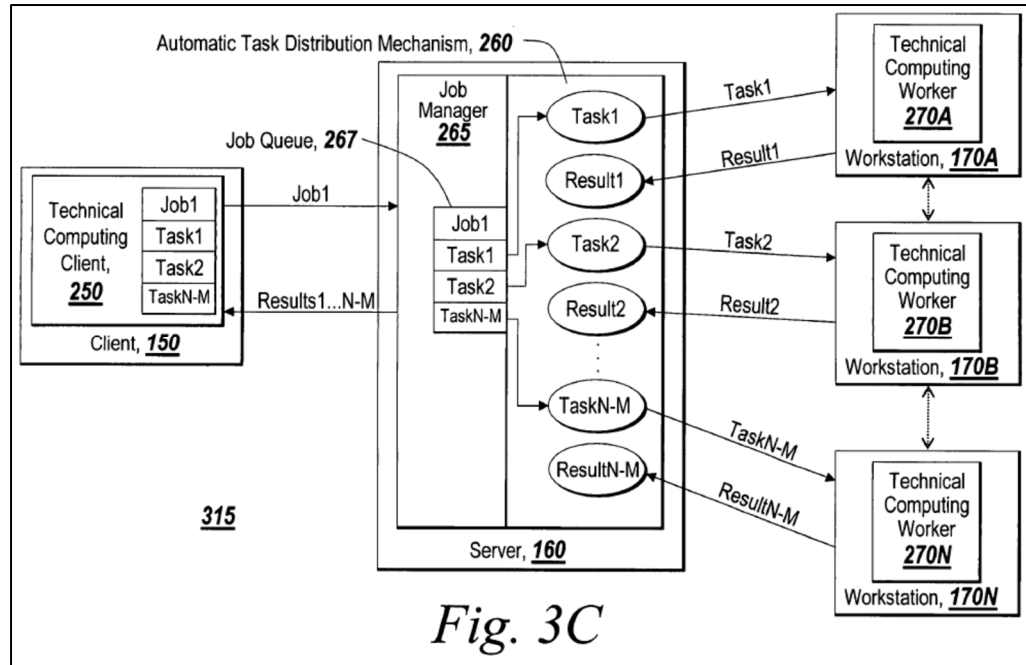


Fig. 3C

## 2. Ethernet Specification (Ex. 1009)

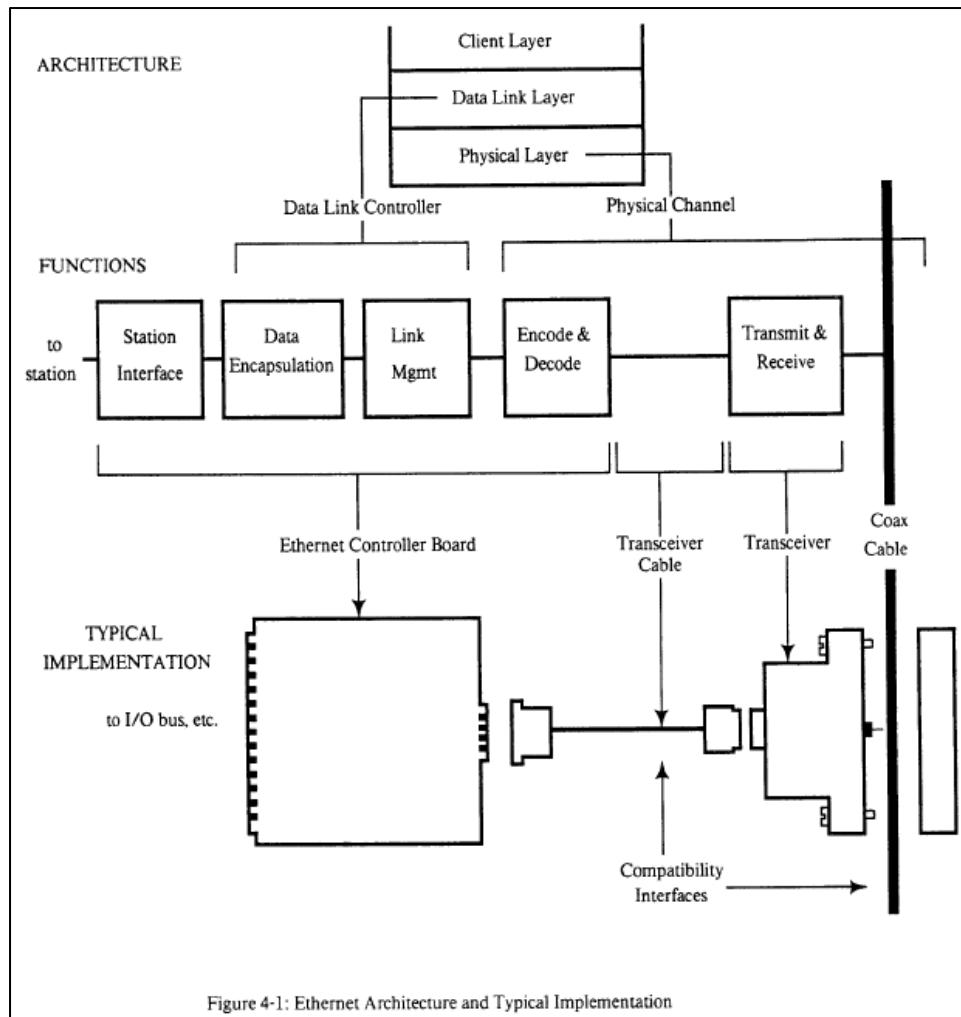
77. “The Ethernet – A Local Area Network, Data Link Layer and Physical Layer Specifications,” Version 1.0 (“Ethernet Specification”) is a specification that describes a local area network (“LAN”) known as Ethernet. Ethernet Specification (Ex. 1009) at i. Ethernet Specification is published by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. *Id.*

78. I understand from counsel that Ethernet Specification is prior art to the ’777 Patent.

79. Ethernet “provides a communication facility for high speed data exchange among computers and other digital devices located within a moderate-size geographic area.” Ethernet Specification (Ex. 1009) at 1. Its protocol “is intended primarily for use in such areas as . . . distributed data processing.” *Id.* at

1. One of Ethernet's "main objectives" is "compatibility," such that "every implementation of the Ethernet be able to exchange data with every other implementation." *Id.* at 2.

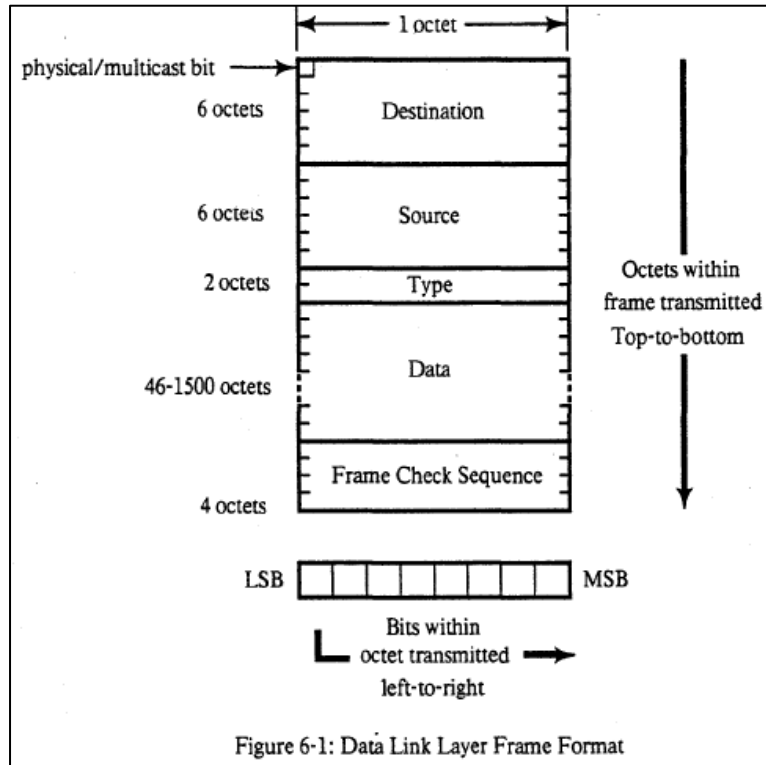
80. Ethernet pertains to "two layers of an overall network architecture," specifically the two lowest layers of the OSI model, which are the physical layer and the data link layer. *Id.* at 6-11, Fig. 4-1 (below). As I previously explained, these layers work underneath higher, client layers. *Id.* at 8 ("[t]he intent . . . is that the Ethernet Physical and Data Link Layers support the higher layers of the ISO model (Network Layer, Transport Layer, etc.)").



81. The data link layer, which handles data encapsulation (including framing, addressing, and error detection) and link management (including channel allocation and contention resolution), specifies a standard format for data frames used to pass information. *Id.* at 19. The size of an Ethernet data frame is at smallest 64 octets and at largest 1518 octets, where each octet is 8 bits or 1 byte. *Id.* at 22.

82. By standard, each frame has five fields: “the addresses of the frame’s source and destination, a type field . . . a data field containing the transmitted data,

and the frame check sequence field containing a cyclic redundancy check value to detect transmission errors.” Ethernet 19-20, Fig. 6-1 (below).



83. Among the five fields, the Destination, Source, and Data fields each specify particular information. “The destination address field specifies the station(s) for which the frame is intended.” *Id.* at 21. So that an Ethernet data frame received by the Data Link Layer from the Physical Layer can undergo decapsulation, “the data link control recognizes and accepts any frame whose destination field contains the physical address of the station.” *Id.* at 26. The size of the destination address field is 6 octets. *Id.* at Fig. 6-1.

84. “The source address field specifies the station sending the frame.” *Id.* at 21. The size of the source address field is 6 octets. *Id.* at Fig. 6-1.

85. “The data field contains a sequence of n octets [and] full data transparency is provided, in the sense that any arbitrary sequence of octet values may appear in the data field.” *Id.* at 22. The size of the data field is between 46 to 1500 octets. *Id.* at Fig. 6-1.

86. In contrast, the Type field is “reserved for use by the higher levels (in particular to identify the Client Layer protocol associated with the frame),” but is included at the Data Link Layer “because a uniform convention for the placement and value assignment of this field is crucial if multiple higher level protocols are to be able to share the same Ethernet network without conflict.” *Id.* at 21.

87. While providing a standard protocol, Ethernet Specification underscores the flexibility in implementing the Data Link Layer. *Id.* at 27 (“The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination . . . it must be emphasized that it is the **behavior** of Data Link Layer implementations that must match the specification, **not** their internal structure.”) (emphasis in original).

### 3. Andrews (Ex. 1011)

88. *Foundations of Multithreaded, Parallel, and Distributed Programming* (“Andrews”) is a textbook authored by Gregory R. Andrews. Andrews was published in 2000. See Andrews (Ex. 1011) at Cover.

89. I understand from counsel that Andrews is prior art to the ’777 Patent.

90. Andrews discloses the “bag of tasks” paradigm for parallel computations in which each “task is an independent unit of work.” Andrews (Ex. 1011) at 132. These tasks “are placed in a bag that is shared by two or more worker processes.” *Id.*

91. In a pseudocode implementation of the “bag of tasks” paradigm, 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 (Ex. 1011) at 132.

92. Because “executing one task will usually lead to producing two smaller tasks, which a worker needs to put back into the bag,” workers continue to execute tasks until the bag is empty and each worker is awaiting a new task. *Id.* 133, 135.

93. It is advantageous to use the “bag of tasks” paradigm for the benefits of scalability and load balancing. Andrews (Ex. 1011) at 132. The “bag of tasks” paradigm enables “any number of processors merely by varying the number of

workers.” *Id.* Scalability also ties into the ability to implement load balancing because processors can execute tasks (i.e., take a task from the bag) once the processor is available. *Id.* at 132-33. In the presence of a sufficient number of tasks over the number of workers, the total amount of computation executed by each worker should be approximately the same. *Id.*

#### **4. Lippett (Ex. 1031)**

94. Lippett is patent application U.S. Patent Application Publication No. 2007/0220517 A1, titled “Scheduling in a Multicore Processor.”

95. Lippett related to scheduling computational tasks among multiple processors. Lippett Abstract (“A method and computer-usable medium including instructions for performing a method for scheduling executable transactions within a multicore processor comprising a plurality of processor elements.”).

96. Lippett focuses on multicore processors with multiple processing elements, but the underlying concepts regarding scheduling are applicable to any multiprocessor application.

97. And relevant here, Lippett teaches that making the scheduling information needed for task distribution readily available in dedicated memory improves system performance. Lippett [0007].

**IX. Ground 1: Claims 1-10 and 12-14 Would Have Been Obvious over Lurie in View of Ethernet Specification**

98. I explain below, on an element-by-element basis, that a POSITA would have concluded that every element of claims 1-10 and 12-14 (i.e., all but claim 11, which I will discuss in Ground 2) of the '777 Patent would have been taught by Lurie alone or in combination with Ethernet Specification, as well as why a POSITA would have considered it obvious to implement Lurie using Ethernet.

99. Lurie teaches a processing system utilizing a controller (Lurie's TCC) that populates a task pool (Lurie's automatic task distribution mechanism), and solidarity cells (Lurie's TCWs) that retrieve, deliver, and complete tasks; generate resulting data; and update the task pool—all without communication with the controller. Lurie discloses transmitting information over “the Internet through a variety of connections,” including “LAN or WAN links.” Lurie (Ex. 1007) at [0051]-[0052]. A POSITA would have considered it obvious to implement Lurie using a well-known LAN protocol, such as Ethernet Specification, whose packets constitute “agents,” as further explained below.

**A. Reasons to Implement Lurie Using Ethernet Specification**

100. My analysis in this section applies to my opinion regarding all claims of the '777 patent.

101. A POSITA would have considered it obvious to, and have good reason to, implement the system described in Lurie using the Ethernet protocol, as taught in Ethernet Specification.

102. Lurie discloses a distributed multiprocessor system and details for implementing such a system comprising the TCC, or controller, automatic task distribution mechanism (task pool) and TCWs (solidarity cells). Lurie (Ex. 1007) at [0071]-[0074], Fig. 3C.

103. Ethernet Specification discloses further details for implementing the system described by Lurie because Ethernet Specification discloses a protocol, or framework, by which the TCWs communicate for locating and retrieving a particular task from the automatic task distribution mechanism, and likewise by which the TCWs communicate for providing a result after executing the task to the automatic task distribution mechanism.

104. For example, Lurie discloses:

The client 150, the server 160, and the workstation 170 can be connected 130 to the networks 140 and/or 140' through a variety of connections including, but not limited to, standard telephone lines, *LAN* or WAN links (e.g., 802.11, T1, T3, 56 kb, X.25), broadband connections (e.g., ISDN, Frame Relay, ATM), wireless connections, or some combination of any or all of the above. Each of the client 150, server 160 and workstation 170 can be any type of computing device (102', 102'' and 102''') as described above and respectively

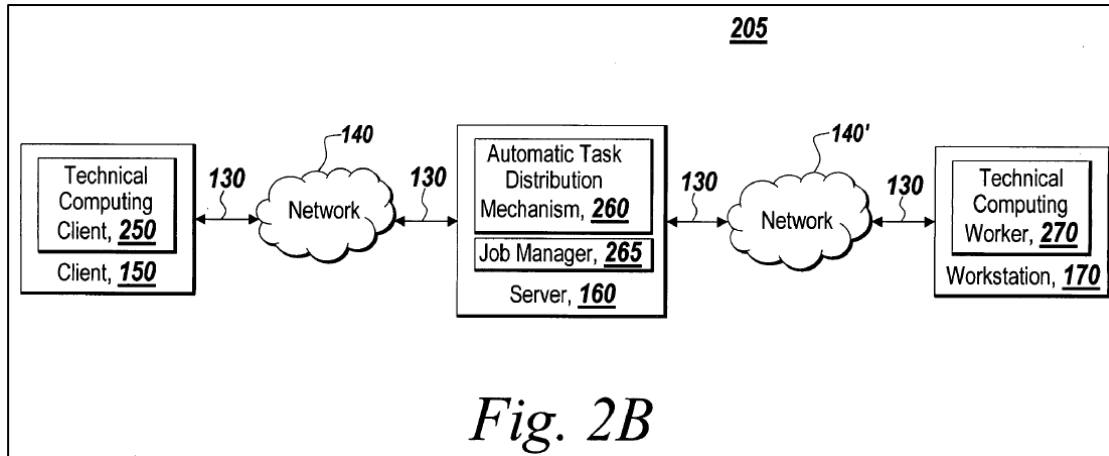
configured to be capable of computing and communicating the operations described herein.

Lurie (Ex. 1007) at [0052]. Lurie further describes connecting “computing device 102” to “a Local Area Network (LAN)” or “the Internet through a variety of connections.” *Id.* at [0051].

105. A POSITA would have understood that Lurie’s disclosure of LAN links is a reference to systems that use Ethernet, the LAN standard taught in the Ethernet Specification. Ethernet has been the most common, nearly ubiquitous, protocol specification for LAN networks since the 1980s. Ethernet Specification (Ex. 1009) at Cover (bearing publication date of September 30, 1980), i (explaining that “[t]he Ethernet specification arises from an extensive collaborative effort” of Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, and “several years of work at Xerox on an earlier prototype Ethernet”); *see also* Andrews (Ex. 1011) at 32 (explaining that “the development of distributed systems” was driven in part by the existence of Ethernet); Tang (Ex. 1016) (confirming that “Ethernet . . . has been the ubiquitous standard of choice for Local Area Networks (LAN)”); Hamblen (Ex. 1017) (“Ethernet originally meant a shared media LAN” and refers to “a set of networking protocols that have become ubiquitous”).

106. A POSITA would have also known that an Ethernet LAN could be used to connect the same kinds of computing devices, “such as a workstation, desktop computer, server, laptop, handheld computer or other form of computing or telecommunications device,” described in Lurie. Lurie (Ex. 1007) at [0051]. Ethernet Specification states that it “is intended primarily for use in such areas as . . . distributed data processing,” to which Lurie is directed. Ethernet Specification (Ex. 1009) at 1. Ethernet Specification further states that it was a “main objective” of its protocol to ensure compatibility – that is, to ensure that “every implementation of the Ethernet” is “able to exchange data with every other implementation,” ensuring compatibility. *Id.* at 2. Thus, a POSITA would have known that Lurie’s use of an Ethernet LAN ensured compatibility and interoperability between the multiple devices in Lurie’s system.

107. Lurie discloses that its system “allows the use of instrument-based distributed system *on a network* to conduct the job and facilitate decreasing the time for executing the job.” Lurie (Ex. 1007) at Abstract. Figure 2B of Lurie further identifies a network 140 connecting the different components of its processing system (*i.e.*, the TCC, the automatic task distribution mechanism, and the TCW). *Id.* at Fig. 2B (below).



108. A POSITA would have understood that Ethernet Specification allows the computing devices to be connected in a LAN, which Lurie explains is one of the possible ways in which its system can be configured. Lurie (Ex. 1007) at [0051]-[0052]. Therefore, when reading Lurie’s disclosure of the use of a “LAN” or a “network,” a POSITA would have looked to a well-known, familiar, successful, protocol, such as Ethernet Specification, to provide implementation details regarding how to connect the different computing devices in accordance with the processing system disclosed in Lurie.

## **B. Independent Claim 14<sup>4</sup>**

### **1. Preamble**

109. The preamble of claim 14 recites:

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

<sup>4</sup> See Claim Listing (Appendix) for assigned reference letters.

110. I have been asked by counsel to assume that the preamble is limiting, and determine whether Lurie discloses the preamble. It does.

111. First, Lurie states that its system is for processing large computing tasks, i.e., the “large computing requirement” of the preamble, using distributed processors that perform their individual portions of the large requirement in parallel to reduce execution time. For example, Lurie states in its Abstract that “[a] large computing work can be performed in a distributed fashion using” its “distributed system.” Lurie Abstract. “The client may distribute a portion of the job to one or more remote workers on a network.... The workers execute the received portion of the job and may return execution results to the client. As such, the present invention ... decreas[es] the time for executing the job.” Lurie Abstract.

112. Lurie continues:

[0004] As a desktop application, MATLAB® allows scientists and engineers to interactively perform complex analysis and modeling in their familiar workstation environment. With many engineering and scientific problems requiring larger and more complex modeling, computations accordingly become more resource intensive and time-consuming. However, a single workstation can be limiting to the size of the problem that can be solved, because of the relationship of the computing power of the workstation to the computing power necessary to execute computing intensive iterative processing of complex problems in a reasonable time. For example, a simulation of a large complex aircraft model may take a reasonable time to run with a single computation with a specified set of parameters. How-

ever, the analysis of the problem may also require the model be computed multiple times with a different set of parameters, e.g., at one-hundred different altitude levels and fifty different aircraft weights, to understand the behavior of the model under varied conditions. This would require five-thousand computations to analyze the problem as desired and the single computer would take an unreasonable or undesirable amount of time to perform these simulations. Therefore, it is desirable to perform a computation in a distributed manner when the computation becomes so large and complex that it cannot be completed in a reasonable amount of time on a single computer. In particular, since some instruments are provided on a PC-based platform and have capacities to run additional software, it is also desirable to use the instruments for performing a large computation in a distributed manner.

[0005] The present invention provides an instrument-based distributed computing system that accelerates the measurement, analysis, verification and validation of data in a distributed computing environment. In the present invention, a large computational job can be performed in a distributed fashion using the instrument-based distributed system. The instrument-based distributed system may include a client that creates a job. The job may include one or more tasks. The client may distribute a portion of the job to one or more remote workers for the distributed execution of the job. The client may reside in an instrument. The workers may also reside in instruments. The workers execute the received portion of the job and may return execution results to the client. As such, the present invention allows the use of instrument-based distributed system on a network to conduct the job and facilitate decreasing the time for executing the job.

Lurie [0004]-[0005]. Thus Lurie explains that its distributed processors (workers) perform portions of a large computing requirement in parallel.

113. Lurie later calls these worker processors “technical computing workers,” and I use the acronym TCW to refer to them. Lurie states that these TCWs perform their tasks in “parallel,” removing any possible doubt that Lurie is

contemplating parallel processing with its “distributed” processors. Lurie [0069], [0070].

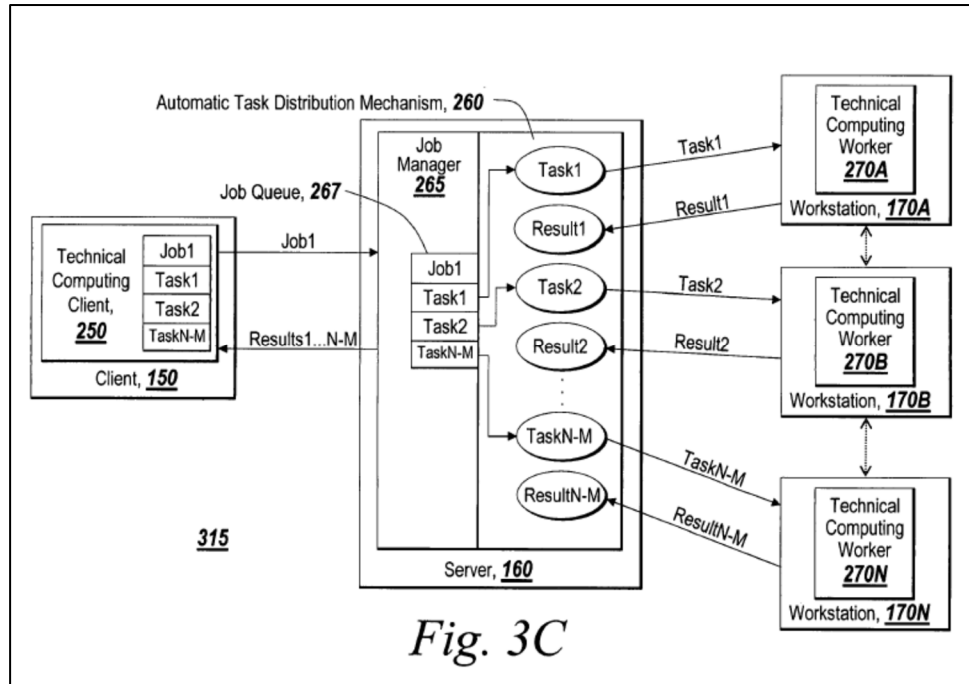
**2. Elements 14[a]-[b]: CPU and task pool**

114. Claim 14 recites:

- [a] a central processing unit (“CPU”);
- [b] a task pool in electronic communication with the CPU;

115. Lurie teaches these claim elements. As I will explain in the following paragraphs, Lurie discloses the claimed CPU (Lurie’s TCC), which is in electronic communication with the claimed task pool (Lurie’s ATD, which may include a job manager as its interface).

116. I will start by addressing the claimed “task pool.” The ’777 patent 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. Lurie’s automated task distribution mechanism (“ATD”) is a task pool as that term is used in the claims. To explain, Lurie discloses an automatic task distribution mechanism. Lurie (Ex. 1007) at Fig. 3C (below).



117. A POSITA would have understood that the automatic task distribution mechanism is a task pool because it collects tasks from the TCC and makes the tasks available to TCWs. “[T]he technical computing client 250 submits one or more tasks . . . to the automatic task distribution mechanism 260,” which “obtains the tasks . . . to make the[m] available for distribution to any of the [TCWs].” Lurie (Ex. 1007) at [0072]. The TCW then “takes a task from the automatic task distribution mechanism 260 for technical computing of the task, computes a result for the task, and provides the result to the automatic task distribution mechanism 260.” *Id.*

118. Based on Lurie’s descriptions of the functions of its ATD, a POSITA would have understood that the ATD is a data or memory construct that is configured to hold, store, monitor, track, display, populate with, and/or update

tasks. Lurie also discloses that its ATD can be implemented “in various software components, applications and interfaces.” Lurie (Ex. 1007) at [0063]. The ATD therefore performs the same functions and has the same construct as a typical “bag of tasks” common to parallel computing systems.

119. A POSITA would have understood that a “task pool,” as the term is used in the ’777 Patent, is just another way to refer to the “bag of tasks” paradigm commonly used in parallel processing where tasks, which are each independent units of work, “are placed in a bag that is shared by two or more worker processes.” Andrews (Ex. 1011) at 132.

120. Both the “bag of tasks” paradigm and Lurie’s ATD are consistent with how the ’777 Patent uses the term “task pool.” For example, the ’777 Patent states that “[m]ultiple embodiments are envisioned for the task pool,” which can be “represented” by unspecified “data structures.” ’777 patent 4:12-28. Thus, a POSITA would have understood that Lurie’s ATD is a “task pool” as the term is used in the ’777 Patent.

121. In addition, Lurie’s ATD “may be object-oriented and comprise an object exchange repository 662, such as a Javaspace . . . .” *Id.* at [0090]-[0091].

122. A POSITA would have understood that the “space” is a task pool based on the characteristics of the space in the automatic task distribution mechanism disclosed in Lurie. Lurie explains that “the JavaSpace technology

views an application as a collection of processes cooperating via a flow of objects into and out of an object exchange repository 662, known as a space.” Lurie (Ex. 1007) at [0091]. The spaces are shared, and so remote processes such as TCWs can “interact with the network accessible object storage of a space.” *Id.* Further, “[s]paces are also persistent” and “[s]paces are also associative in that objects in the space can be located by associated lookup rather than by memory location or identifier, e.g., in a shared memory solution.” *Id.* Lurie further discloses exemplary key operations for handling exchanging of objects, including a write operation, a take operation, a read operation, and other operations allowing TCWs to register for notifications when objects appear in the space. *Id.* These key operations align with the functions that the task pool must perform, such as holding, storing, monitoring, tracking, displaying, populating with, and/or updating tasks.

123. Moreover, simply based on the existing state of the art regarding JavaSpaces, a POSITA would have understood that the “space” in the context of JavaSpace technology is a task pool. For example, Batheja describes the framework architecture of JavaSpaces known in the state of the art. Batheja (Ex. 1012) at 2. Batheja states, much like Lurie does, that “JavaSpaces is a shared, network accessible repository for Java objects [], and provides a programming model that views applications as a collection of processes cooperating via the flow of objects into and out of one or more spaces.” *Id.* Batheja further states that

JavaSpaces “leverages the Java type semantics and provides operations to store, retrieve and lookup Java objects in the space.” *Id.* Batheja continues to describe how JavaSpaces supports parallel computing, where the “space” serves as the task pool, or “pool of tasks”:

JavaSpaces naturally supports the master-worker parallel/distributed-computing paradigm. In this paradigm (see Figure 1), the application is distributed across available worker nodes using the *bag-of-tasks* model. The master produces independent application tasks and puts them into the space. The worker consumes the tasks from the space, computes on these tasks, and places results back into the space. This approach supports coarse-grained applications that can be partitioned into relatively independent tasks.

Batheja at 2.

124. Indeed, Figure 1 of Batheja identifies a parallel computing architecture in which the JavaSpace “space” is identified as a “pool of tasks.”

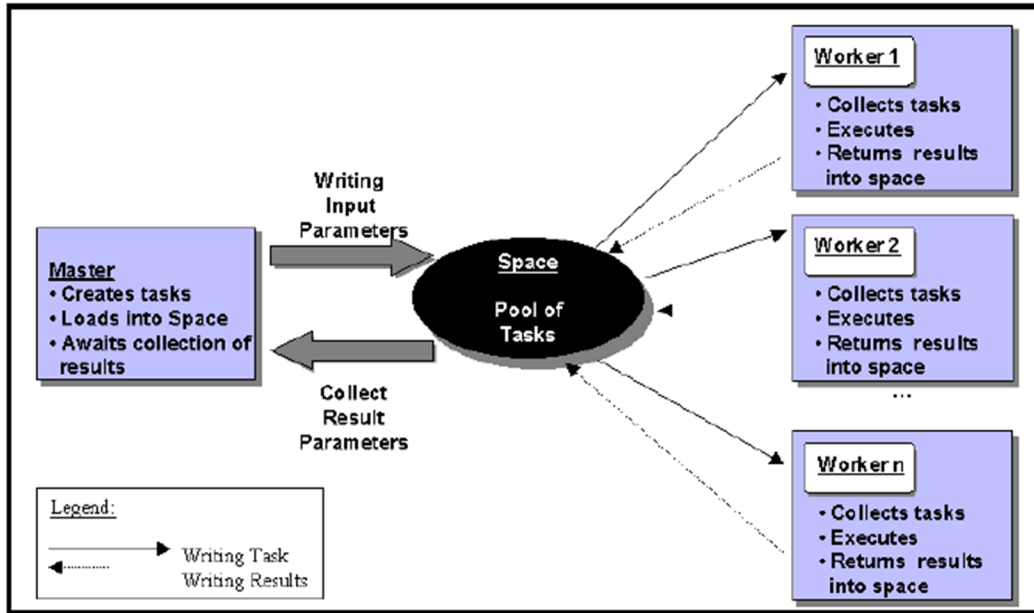
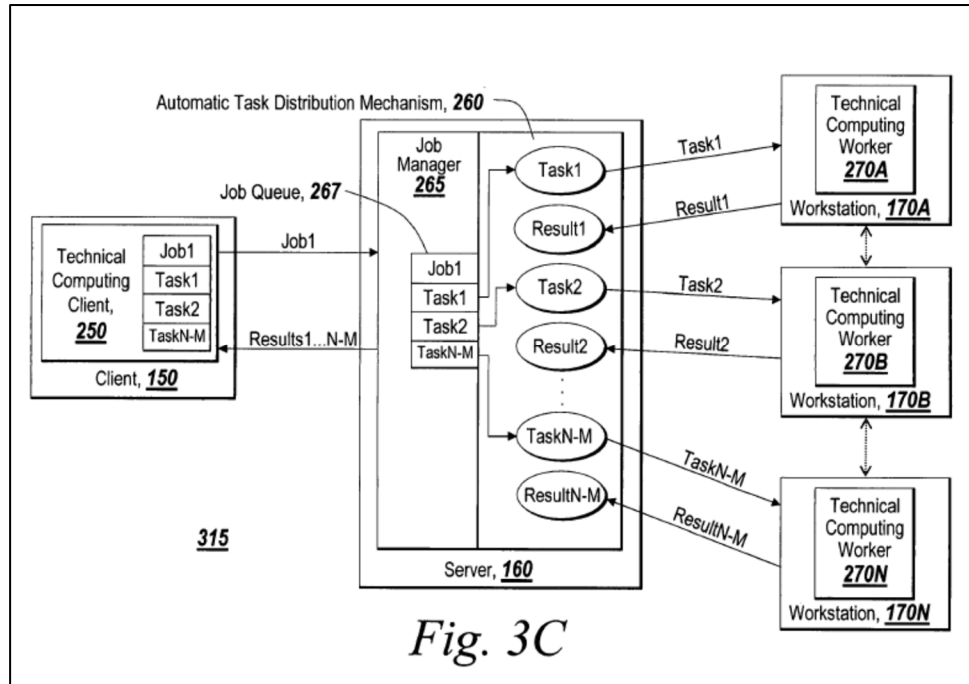


**Figure 1. Master-worker parallel computing using JavaSpaces**

*Id.* at Fig. 1.

125. Thus, a POSITA would have understood that Lurie’s automatic distribution mechanism, which can comprise a space, is a task pool.

126. Further, Lurie discloses the claimed CPU via its Technical Computing Client (“TCC”), such as TCC 250 shown in Figure 3C:



127. A POSITA would have understood that Lurie's TCC includes a central processing unit and is in communication with the ATD, as required by the claimed CPU. *E.g.*, Lurie [0052] ("[T]he client 150 ... can be any type of computing device"). Lurie's TCC is in communication with the ATD, as illustrated above, and populates the ATD (task pool) with tasks. *See also* 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."), Fig. 2C.

128. Lurie's TCC fills the well-known role in the bag of tasks model, dividing a large requirement into smaller jobs and populating the task pool. *See, e.g.*, Bates (Ex. 1013) at [0033] ("The PPU 102 acts as a controller for the SPUs 104 [solidarity cells], which handle most of the computational workload."), [0036]

(“The PPU 102 can add new task queues to the task set 114 but has little other involvement with the management of the task set 114.”), Fig. 1; Chow (Ex. 1008) at [0013] (“When tasks arrive at the system kernel 102 for processing, they are subdivided into subtasks and placed into the library task queue 106.”), Fig. 1.

129. As Lurie explains, the TCC “submits one or more task . . . to the automated task distribution mechanism,” which “obtains the tasks . . . to make the[m] available for distribution to any of the technical computing workers.” Lurie (Ex. 1007) at [0072]. I note that Lurie’s TCC populates its ATD by dividing a large computational requirement into one or more jobs and placing the jobs in the ATD (task pool). I will discuss this point at length below, with respect to claim 1’s Element 1[d].

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

130. 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,

131. Lurie, alone or with Ethernet Specification, teaches the claimed solidarity cells. Lurie’s TCWs correspond to the claimed solidarity cells, and they include the claimed agents that each comprise a source address, a destination address, and a payload. I will discuss the agents’ other characteristics (e.g., that

they are configured to retrieve tasks from the ATD) in the next subsection which focuses on the language replaced by ellipses above and also the way the claims recite that the agents retrieve tasks.

132. I also note that any TCW may serve as the first solidarity cell and another TCW may serve as the second solidarity cell.

133. As I explained in the Claim Construction section above, a POSITA would have understood that a “solidarity cell” in the context of the ’777 patent’s claims is merely any processor or processing system that performs the functionality recited in the claims. Lurie’s TCWs meet those requirements.

134. **First**, Lurie’s TCWs (which Lurie sometimes calls “workers”) each include a processor that executes their assigned tasks. See Lurie [0041]-[0045], [0054]-[0056]. Lurie’s TCWs are also programmed or otherwise configured to perform the functions recited in claim 14, as I will discuss at length with respect to Elements 14[c.2], [d.2], and [h].

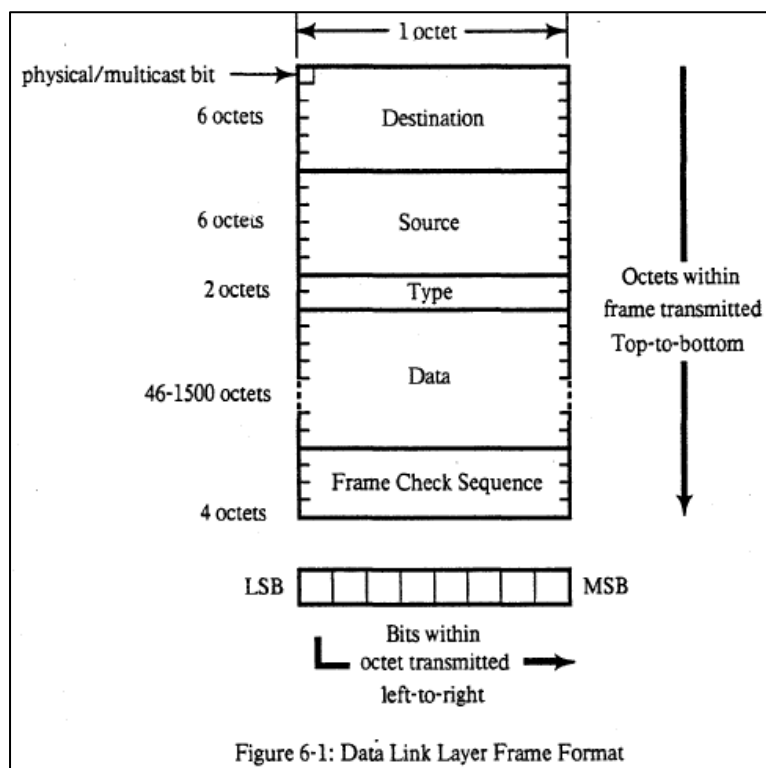
135. **Second**, Lurie’s TCWs communicate with the ATD via a network using the claimed agents that have a source address, destination address, and a payload. Lurie (Ex. 1007) at [0071] (“the technical computing workers 270A-270N available on the network”). I will explain this point at length in the following paragraphs.

136. Specifically, Lurie discloses “LAN,” “WAN,” or “Internet” as examples of these networks. *Id.* at [0051]-[0052]. A POSITA would have understood that information or data transmitted over such a network is transmitted in accordance with a standard protocol, utilizing a data frame, network packet, or equivalent protocol data unit that serves as an “agent.” Moreover, a POSITA would have understood that “LAN” refers to “Local Area Network,” the most common of which is Ethernet. Thus, a POSITA would have considered it obvious to implement Lurie on an Ethernet network because, as I explained among the reasons to implement Lurie using Ethernet Specification, Ethernet was a common network over which to implement distributed systems. And as I explained in my Reasons to Implement section, a POSITA would have been motivated to look to a well-established and ubiquitous LAN protocol, such as Ethernet, for additional detail regarding implementation of how the solidarity cell retrieves a task from the task pool according to the system described in Lurie. *See also* Tang (Ex. 1016) at 1 (“Ethernet ... has been the ubiquitous standard of choice for Local Area Networks (LAN).”); Hamblen (Ex. 1017) at 1 (similar).

137. As I explained in my Overview of Ethernet Specification, the Ethernet protocol pertains to the data link and physical layers of the OSI model. Ethernet Specification (Ex. 1009) at 22, Fig. 6-1. Ethernet networks use data frames when sending information. Ethernet Specification (Ex. 1009) at 19 (“The data

encapsulation function of the Data Link Layer comprises the construction and processing of frames.”).

138. Ethernet data frames include a source address, a destination address, and a payload as data fields. *Id.* at 21-22. Each data frame comprises a source field address, a destination field address, and a data field. *Id.* at 21-22 (explaining each of these fields), Fig. 6-1 (below).



139. The “source” field identifies “the station sending the frame” either by its unique “physical address” or “multicast address.” Ethernet Specification (Ex. 1009) at 21. This can be either a physical address, or a name or other address provided by higher level convention in the OSI model. *Id.* In Lurie, when a TCW sends a communication to the automatic task distribution mechanism, this field

would identify the address or name of the TCW as the “station sending the frame,” serving as a “source address.”

140. The “destination” field identifies “the station(s) for which the frame is intended,” which may also be a physical or multicast address. Ethernet Specification (Ex. 1009) at 21. In Lurie, when a TCW sends a communication to the automatic task distribution mechanism, this field would identify the address or name of the ATD’s machine as the “station for which the frame is intended,” serving as a “destination address.”

141. The “data” field allows for any type of data, up to 1500 octets in length, to be transmitted. Ethernet Specification (Ex. 1009) at 22. In Lurie, the data field would contain a payload. For example, as I explain below with respect to Elements 14[c.2], [d.2], and [h], in order to ensure that each of Lurie’s TCWs receives tasks that match that TCWs “operational and/or performance characteristics,” as Lurie describes, the data field would contain a payload that includes an one or more indices, names or pointers/references referencing the functions that the TCW is configured to perform, so that the Ethernet data frame can search for and find a task type matching that TCW’s capabilities. Alternatively, the data field could contain a script, executable, or function that performs searching.

142. Additionally, a POSITA would have understood that nearly all such protocols commonly known in the art—including those such as TCP/IP or UDP that pertain to higher, client layers—utilize some data unit (e.g., data frames, network packets, datagrams) similarly include a source address, a destination address, and a payload as data fields. Internet Protocol (Ex. 1018) at 1, 34-35; UDP Specification (Ex. 1019) at 1; Day (Ex. 1023) at 1338-40; Meyer (Ex. 1024) at 18-19. It would also have been obvious to use those commonly used protocols for similar reasons.

143. A POSITA would have also understood that an Ethernet data frame (or other standard network packet) is an agent, as the term is used in the '777 Patent. Claim 1 specifically recites “the first agent is a data frame,” and the specification states that “[a]n agent ... may be considered a data frame in the networking sense.” '777 patent 5:28-32. Further, Ethernet data frames have the recited source address, destination address, and payload, as I just explained.

144. Thus, a POSITA would have understood that Ethernet (or similar networking protocol) data frames are sent from the TCW to the task pool, which are in communication over a network, with a payload having information that can be used to identify tasks appropriate to that TCW. A POSITA also would have understood that these agents are transmitted according to a standard protocol,

where these protocol data units are “agents” of the TCW having the recited source address, destination address, and payload.

145. I will discuss these agents further in the next section, especially their payloads.

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

146. 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.

147. I note that this patent’s claims, unlike the ’004 or ’275 patents that I analyzed previously, recite that each **agent** (rather than each co-processor or solidarity cell) proactively retrieves the matching task according to the recited steps.

148. Lurie teaches this. Lurie’s TCWs send their agents (data frames/packets) to the ATD, without an instruction from the TCC, to proactively retrieve a matching task as recited in steps 14[h.1]-[h.3]. The task retrieved by a TCW’s agent is a “matching task” as recited in the claim because, as I will explain

in detail in the subsections below, the agent's payload specifies the type(s) of task that its TCW can perform, and the agent searches for and retrieves a task that matches that type.

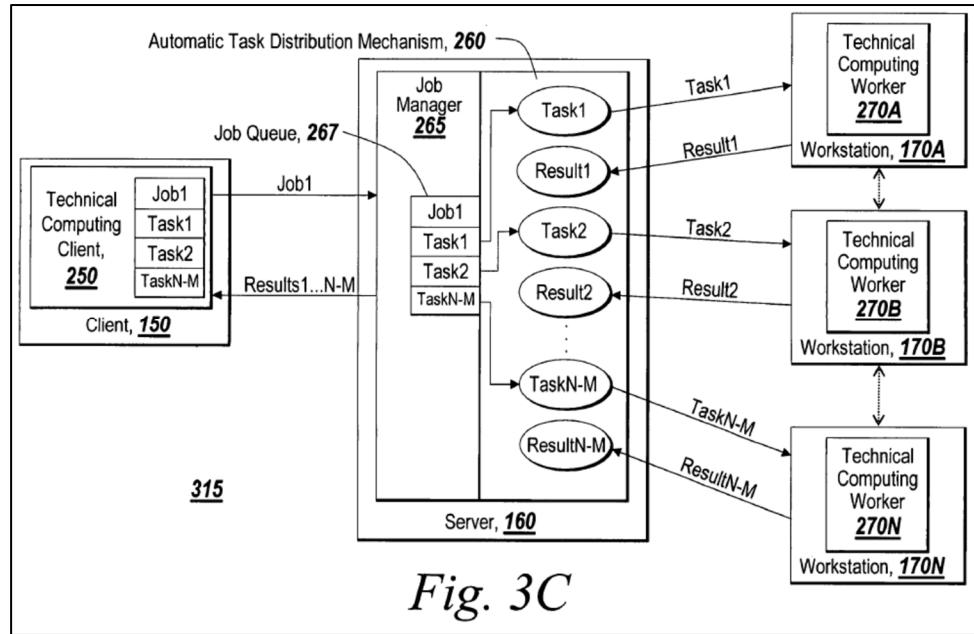
149. For now, I note that the agents (data frames/packets) proactively retrieve matching tasks. Each TCW sends an agent that "takes a task from the automatic task distribution mechanism." Lurie (Ex. 1007) at [0072], [0074]. When implemented through JavaSpaces, "[a] take operation takes an object . . . from the space," which "is the equivalent of a read and removes the object from the space." *Id.* at [0091], [0093]. Lurie also describes an exemplary *getTask* method for the space (task pool) for TCWs to invoke (via their agents) to take a task. Lurie at [0128].

150. A POSITA would have understood that this take (e.g., *getTask*) operation is invoked by the TCW, thus sending its agent to proactively take a task from the ATD (task pool). The agent is proactive because, as soon as it is sent by the TCW, the agent proceeds to the ATD to retrieve a matching task. The TCW's agent does not need, and Lurie does not disclose, that it receives any instruction from the TCC or the ATD to perform this operation.

151. I also note that the claims do not require the TCWs (as opposed to their agents) to be proactive, but they are.

152. First, the TCWs do not receive any instructions from the TCC. Lurie discloses that “the technical computing client 250 is not required to have any specific knowledge of the technical computing workers 270A-270N, e.g., the name of the workstation running a technical computing worker 270A-270N, or the availability of the technical computing worker 270A-270N to perform technical computing of a task” when using an ATD, i.e., not in direct distribution mode. Lurie at [0071]. Rather, “each of the technical computing workers 270A-270N is in communication with the automatic task distribution mechanism 260.” Lurie at [0073].

153. In other words, “[T]he automatic task distribution mechanism provides a level of indirection between the technical computing client 250 and the technical computing worker 270.” Lurie [0061], [0063], [0092] (same “level of indirection” in the JavaSpaces embodiment). This is illustrated in Lurie’s figures. For example:



Lurie Fig. 3C; see also Figs. 3D, 4, 5C-D, 6B.

154. Second, the ATD does not instruct the TCWs to retrieve a task. Rather, the ATD notifies all of the TCWs when tasks are first available. To explain, 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], [0091]-[0093], [0097].<sup>5</sup> This notification goes out to all of the registered TCWs. It

<sup>5</sup> I also note that Lurie's ATD makes tasks available for distribution, allowing TCWs to pick matching tasks via their agents. It is well known in the art that the problem of "data distribution" (also termed "task distribution") focuses on what data is distributed to certain machines, which is agnostic to how that data gets to those machines. For example, in Chow, the library processors (co-processors) partake in the "distribution" of tasks, which is simultaneously described as the library processors "fetch[ing]" tasks. *See, e.g.*, Chow (Ex. 1008) at [0013] ("the library processors 108, 110, 112, 114, and 116 fetch the subtasks from the library task queue"), [0014] ("the library processor fetches a task from the library task queue ... [i]n addition, the library processor 200 shares in the work of the distribution of tasks from the library task queue 106"), [0015] ("Referring to FIG.

is not an instruction to each ATD to retrieve a task. Lurie's ATD Mechanism simply provides the information, or notification, that one or more tasks are available. Lurie at [0086]-[0088]. This notification is essentially a flag, conveying information (i.e., that tasks are available), but without instructing (i.e., obligating or commanding), a TCW to take any action. For example, a TCW that previously registered may have sent a de-registration message to the ATD Mechanism. In that case, a POSITA would understand that a de-registered TCW will not retrieve a task, even if it receives this notification.

155. By contrast, an "instruction" is understood to be a programming statement specifying a specific operation that must be performed, often an executable statement.

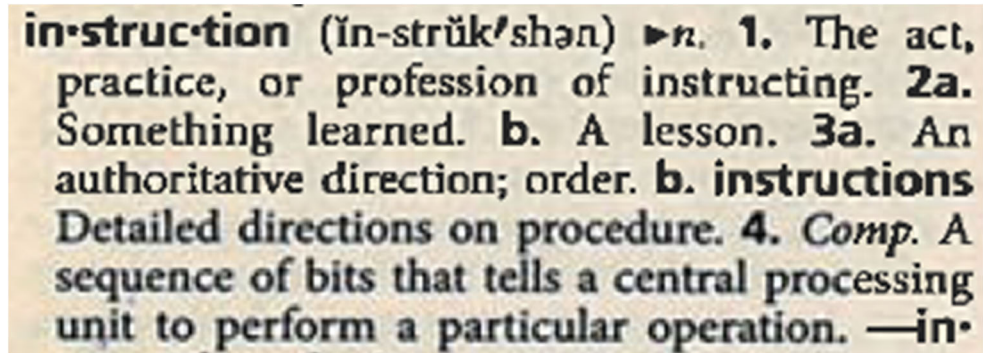
**computer instruction** A machine instruction for a specific computer. (C) [20], [85]

(2) (A) (software) A statement in a programming language, specifying an operation to be performed by a computer and the addresses or values of the associated operands; for example, Move A to B. *See also:* instruction set; instruction format. (B) (software) Loosely, any executable statement in a computer program. (C) 610.12-1990, 610.10-1994

---

3 of the drawings, illustrated is a flow chart of the subdivision of tasks into subtasks and their distribution to the library processors."), [0016] ("The library processors 108, 110, 112, 114, and 116 fetch the subtasks from the library task queue."); Chow at Abstract ("The library processes fetch tasks from the queue for execution.").

IEEE 100 – The Authoritative Dictionary of IEEE Standards Terms (7<sup>th</sup> Ed., 2000) (Ex. 1052).



**in·struc·tion** (in-strük'shən) ►*n.* **1.** The act, practice, or profession of instructing. **2a.** Something learned. **b.** A lesson. **3a.** An authoritative direction; order. **b. instructions** Detailed directions on procedure. **4. Comp.** A sequence of bits that tells a central processing unit to perform a particular operation. —*in·*

156. American Heritage Dictionary (5<sup>th</sup> Ed., 2012) (Ex. 1051). Lurie's notification is not an "instruction" within this commonly understood definition. It does not specify any operation to be performed by the TCW, nor is it an executable statement.

157. In addition, Lurie's TCWs proactively dispatch their agents when they have no processing to perform. I will explain this further with respect to Element 14[f] below.

158. Although not required by the claims, I note that after its agent proactively retrieves the task, each TCW then computes a result for the task. Lurie at [0074] ("Technical computing worker 270A may take Task1 from the automatic task distribution mechanism 260, compute a Result1 for Task1 and provide the Result1 to the automatic task distribution mechanism 260"). I also note that Lurie is following the textbook bag-of-tasks model. *See* Andrews (Ex. 1011) at 135 ("A worker repeatedly takes a task from the bag and executes it."). Lurie's TCW then

provides the result back to the ATD. Lurie [0071]-[0074], [0094], [0103] (updating the task object in the pool).

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

159. 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;

160. Lurie, alone or in view of Ethernet Specification, teaches this claim element.<sup>6</sup>

161. As I just explained, Lurie's TCW retrieves tasks (via its agent) from the ATD. In particular, 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," in this case, the sending TCW. Ethernet Specification 21; *see also* Ethernet Specification Overview and Element 14[g] above. Ethernet Specification (Ex. 1009) at 21. This can be either a physical address, or a name or other address provided by higher level convention in the OSI model. *Id.* In Lurie, when a TCW sends a communication to the automatic task

---

<sup>6</sup> Also, Ethernet is just an example, as I have mentioned above. The same idea (source, destination, and payload) was, and still is, found in common standard network protocols.

distribution mechanism, this field would identify the address or name of the TCW as the “station sending the frame,” serving as a “source address.”

162. The “destination” field (destination address) of the TCW’s agent (data frame) identifies the ATD because it is “the station(s) for which the frame is intended.” Ethernet Specification 21; *see also* Ethernet Specification Overview and Element 14[g] above. In Lurie, when a TCW sends a communication to the automatic task distribution mechanism, this field would identify the address or name of ATD’s machine as the “station for which the frame is intended,” serving as a “destination address.”

163. The agent’s payload is the data field of the Ethernet data frame. Ethernet’s “data” field allows for any type of data, up to 1500 octets in length, to be transmitted. Ethernet Specification (Ex. 1009) at 22. In Lurie, the data field would contain a payload that identifies the list of functions that the sending TCW can perform. As I will explain in detail in the following paragraphs, Lurie’s TCW would populate its agent’s payload with information to ensure that it retrieves tasks that match that TCWs “operational and/or performance characteristics,” such as 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).

164. To explain, Lurie discloses that its TCWs are configured to perform one or more task types, and—as one should expect—retrieve tasks that they can perform. Specifically, Lurie explains that its TCW will retrieve (via its agent) 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]. Lurie discloses myriad different exemplary characteristics, configurations, or properties of its TCWs, each of which correspond to a list of (one or more) functions that the TCW can perform, as I will further explain in the following paragraphs.

165. A task in Lurie “defines a technical computing command, such as a MATLAB® command to be executed, and the number of arguments and any input data to the arguments.” Lurie (Ex. 1007) at [0043]. The use of the word “command” here is broad. For example, “command” can refer to pre-defined MATLAB functions such as an arithmetic operator (e.g., +), or an M-file function. MATLAB Reference (Ex. 1057) at 2-2 to 2-8. It is possible to add functions to MATLAB based on the pre-defined MATLAB functions:

You add new functions to MATLAB’s vocabulary by expressing them in terms of existing functions. The existing commands and

functions that compose the new function reside in a text file called an *M-file*.

M-files can be either *scripts* or *functions*. Scripts are simply files containing a sequence of MATLAB statements. Functions make use of their own local variables and accept input arguments.

MATLAB Reference (Ex. 1057) at 2-296 (emphasis in original). Each of the commands, or functions, in MATLAB are different and therefore of different types.

166. Further, Lurie teaches that different TCWs may have different portions and/or versions of MATLAB installed. Lurie states that a TCW “may be configured to and capable of [sic] running any of modules, libraries, or software components of the MATLAB® product family” and therefore “may have all *or a portion* of” MATLAB installed or accessible. Lurie at [0058].

167. Lurie further states that a 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. Lurie at [0081], [0065]. Lurie also acknowledges that not every TCW will be able to perform every task (or it may not be desirable to have every TCW perform every task). And while some TCWs may run some version and/or subset of MATLAB, others may run LabVIEW, MATRIXx, Mathematica, Mathcad, Maple, or something else entirely. Lurie at [0046].

168. Lurie also discloses that the TCWs have “performance characteristics” that further differentiate its TCWs from each other. Lurie at [0081]. Thus, a POSITA would have understood that one TCW (perhaps less capable or slower) may better suited to perform simpler MATLAB (or other platform) commands such as scalar addition, whereas another TCW (generally more capable or faster) is better suited to perform complex commands such as matrix multiplication. Thus, even among MATLAB commands from the same version of MATLAB, there are different task types.

169. Lurie further teaches that each TCW retrieves tasks of the type that it can perform. As I pointed out above, Lurie states that a TCW “may work on a task based on the version of [MATLAB] that is installed. Lurie at [0081]. Indeed, Lurie repeatedly discloses that the TCWs perform their tasks—thus showing that the TCWs are indeed capable of performing them. Further, Lurie provides an exemplary *getTask* method for the space (task pool) for TCWs to invoke to take a task. Lurie’s *getTask* method includes the following description: “[A] task will be taken from the space. This will block until a task is found. ***If passed a null, a task with any TaskID will be returned.***” Lurie at [0128]. As a result, a TCW able to perform any type of task invokes *getTask* (via its agent) without specifying the task type. That TCW will take any task regardless of its type. But a TCW able to perform only a certain type or types of tasks invokes *getTask* (via its agent) with

the list of functions the TCW can perform as a parameter—thus searching the ATD Mechanism for a task that the TCW can perform.

170. A POSITA would have known that Lurie’s TCWs need a way to identify (in the agent’s payload, e.g., as a parameter in the *getTask* invocation) which of these many different functions they can perform. And an obvious way to do so would be to encode the list of functions that a particular TCW can perform as, for example, one or more indices, names or pointers/references associated with one or more (list of) task types that the TCW can perform, and put that into the agent’s payload. This could be, for example, a string or other parameter indicating which MATLAB commands, version, or library, or other software environment the task requires, which serves as a list of one or more functions that the TCW can perform. Lurie at [0182]-[0183] (task object has “specified property values” in “any format”).

171. I note that the ’777 patent claims do not recite how the “list of functions” may be encoded into the payload. I could find no further guidance in the ’777 patent specification, which merely states that the payload “contains identifying information of the types of tasks the corresponding cell ... can perform.” ’777 patent at 5:55-57. In fact, the “list of functions” may even be a “list” of just one function. The patent claims expressly contemplate a list of just one item, as claim 3 recites that the solidarity cell (i.e., Lurie’s TCW) “is configured to

perform tasks of one or more of the task types.” Thus, the “list of functions” may include just one task type, or more, and it may be encoded into the payload in any manner, including the ways I discussed above.

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

172. 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;

173. The ’777 patent states that an 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. That “hard-coding” is what I discussed in the previous section, and Lurie teaches the claimed searching recited above.

174. As an initial matter, the tasks made available in Lurie’s ATD are ready to be processed. Otherwise, they would not be made available for TCWs. Lurie [0063] (job manager, which may be a portion of the ATD, helps manage “workflow”); [0072]-[0075] (ATD obtains tasks and makes them available for distribution), [0081], [0095].

175. And as I explained in the previous section, Lurie (alone or with Ethernet Specification) teaches that a TCW dispatches an agent to the ATD to

retrieve a task that the TCW can perform, and that the agent includes in its payload the list of functions that the TCW can perform.

176. A POSITA implementing Lurie would have found it obvious to ensure that the TCW receives a task that it can perform by dispatching an agent that searches for and retrieves a task of the appropriate type. As I explained above, a POSITA would understand that the agent, when dispatched to the task pool, carries a payload that includes the one or more indices, names or pointers/references to the list of functions that the TCW can perform so that the agent can search for and find a task type matching the function (*i.e.*, a task suitable for that TCW).

177. Lurie further discloses that the search undertaken by its agents is performed by associative lookup. Lurie [0091]. The TCW sends an agent that contains information pertinent to the associative lookup, and its arrival at the ATD Mechanism initiates the search. Further, associative lookup using keys was well known in the art as a way to search and locate information. Mehlhorn (Ex. 1058) at 81 (“Hash tables are one implementation of associative arrays, or dictionaries.”), 82-83 (explaining the basic concept of using a key for lookup).<sup>7</sup> The TCW’s agent is therefore configured to search.

---

<sup>7</sup> I also note that an associative array “is an array with a potentially infinite or at least very large index set . . . only a small number of indices are actually in use.” Mehlhorn (Ex. 1058) at 81. In other words, where the number of indices in use is

178. The one or more indices, names or pointers/references provided in the payload enables an associative lookup, which matches the operational and/or performance characteristics of the TCW with the right task type. Lurie (Ex. 1007) at [0081] (describing matching the task type with “a description of any operational and performance characteristics”), [0091] (“Spaces are also associative in that objects in the space can be located by associative lookup rather than by memory location or identifier, e.g., in a shared memory solution.”).

179. I also note that Ethernet Specification discloses that its “data field” contains a series of up to 1500 octets, which a POSITA would have understood is capable of carrying the payload of an one or more indices, names or pointers/references corresponding to the list of task types the TCW can perform. Ethernet Specification (Ex. 1009) at 22 (“The data field contains a sequence of  $n$  octets, where  $46 \leq n \leq 1500$ .”), Fig. 6-1 (depicting Ethernet data frame format).

180. Although I do not believe a POSITA would interpret the claim this way, I understand that the Patent Owner may argue that “searching the task pool” as recited in the claim requires that the agent contain an executable file or script configured to perform actual searching. Lurie discloses that, too. Lurie provides an

---

constrained, there is the possibility that two keys map to the same value. To handle this scenario, the array is often implemented as a linked list called a chain—and the agent will need to search that chain for the desired type task. Mehlhorn at 83 (“Hashing with chaining maintains an array  $t$  of linear lists (*see id.* Fig. 4.1). The associative-array operations are easy to implement.”).

exemplary *getTask* method for the space (task pool) for TCWs to invoke to take a task. Lurie's *getTask* method includes the following description: "[A] task will be taken from the space. This will block until a task is found. ***If passed a null, a task with any TaskID will be returned.***" Lurie at [0128]. As a result, a TCW able to perform any type of task invokes *getTask* (via its agent) without specifying the task type. That TCW will take any task regardless of its type. But a TCW able to perform only a certain type or types of tasks invokes *getTask* (via its agent) with the certain type(s) of tasks the TCW can perform as a parameter—thus searching the ATD Mechanism for a task that the TCW can perform.<sup>8</sup> Thus, when the TCW via its agent invokes the *getTask* method (via its agent) with the list of functions it can perform, the agent will search the ATD for a task of a type that the TCW can perform.

181. In addition, a POSITA would also have considered it obvious to include a file or script in the payload of a data frame using a standard protocol, such as Ethernet, the Internet Protocol, or UDP. Even if such a file or script is too large to send over a single data unit, the same information is able to be transmitted and routed over multiple data units through fragmentation:

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and

---

<sup>8</sup> Lurie's *getTaskIfAvailable* method operates similarly, but returns a null if no tasks are available in the task pool (ATD Mechanism).

copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be).

*See, e.g.*, Internet Protocol (Ex. 1018) at 8-9 (further explaining assembly of fragments); *see also id.* at 8 (alternatively noting that “[a]n internet datagram can be marked ‘don’t fragment’”). A POSITA would have understood that agents carrying these kinds of payloads, whether sent as a single data unit or split over a number of data units, are nevertheless configured to search the task pool for a task of the first type.

182. To summarize the above points, the combination of Lurie and Ethernet Specification discloses “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 it would have been obvious to a POSITA to so implement Lurie. Lurie discloses a processing system in which a TCC populates the task pool with tasks of different types, and with multiple solidarity cells (TCWs) that each retrieve and complete those tasks to process them efficiently. *See, e.g.*, Lurie (Ex. 1007) at [0071]-[0074]. To maximize efficiency, Lurie also discloses that each solidarity cell (TCW) performs tasks with specific task type(s), and therefore retrieves and completes tasks that it can perform. *See, e.g., id.* at [0081] (“the system can determine or

select the technical computing worker 270A-270N to work on a task by operational and/or performance characteristics of the technical computing worker 270A-270N”). Lurie also discloses that the TCC, ATD, and TCWs communicate over a network. *See, e.g., id.* at Fig. 1B. A POSITA would have understood, or considered it obvious, that to ensure each TCW processes a task appropriate for its operational and/or performance characteristics, Lurie’s system, with components that communicate over a network, uses an agent in the form of a data frame, network packet, or equivalent protocol data unit whose data field carries one or more indices, names or pointers/references corresponding to the task types.

183. A POSITA further would have understood that an agent, such as a data frame, network packet, or equivalent protocol data unit, is configured to search the JavaSpace “space” of the automatic task distribution mechanism because the agent carries the information necessary to perform the associative lookup and initiates that process. Lurie (Ex. 1007) at [0091]. A POSITA would have understood that this associate lookup is a method of searching the task pool for a task. For example, this is the same methodology explained in the ’777 Patent: “[t]o acquire a task 22, a cell 12A sends 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 (Ex. 1001) at 5:22-37. A POSITA would have understood that the associative lookup conducted on

the space, or search, is enabled by the information contained in the agent's payload, which allows the agent to search for and retrieve only tasks whose indicated type matches the operational and/or performance characteristics of the TCW.

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

184. 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.

185. Lurie (alone or with Ethernet Specification) teaches this claim element, mainly for the reasons I have already discussed in the previous subsections above.

186. In short, Lurie discloses the interaction between a TCW (solidarity cell) and ATD (task pool); which would be implemented using the Ethernet Specification, utilizing data frames, or agents, to provide a source address, destination address, and payload when dispatched to the task pool. A POSITA would have understood that an agent in Lurie returning from the automatic task distribution system to the TCW with a task would have a source address corresponding to that of the automatic task distribution system (the "source" of the agent) and a destination address corresponding to that of the TCW (the end destination).

187. Lurie discloses that the payload includes a descriptor of the task when the agent is returning from the task pool. Lurie discloses that “[t]he technical computing work[er] 270 *obtains the name and arguments of the function to compute* from the data structure of the task object.” Lurie (Ex. 1007) at [0094]. A POSITA would have understood that the name and arguments of the function to compute constitute descriptors of the first task, which are obtained from the ATD, or task pool, and are conveyed to the TCW. A POSITA would have understood, or at least considered it obvious to, implement Lurie’s system such that, the TCW obtains this information through the agent, which stores this information in the form of a payload collected by the agent when the agent is returning from the task pool to the solidarity cell. As I explained above, this payload would be stored in the data region of an Ethernet datagram when Lurie is implemented using Ethernet. Moreover, it would also have been obvious to a POSITA to store this payload in the data region of a datagram using another common protocol, such as TCP/IP or UDP, which invoke similar headers and were similarly well-known for transmitting data.

188. I will address the ’777 patent’s “descriptors” more when I discuss Element 1[e], which expressly recites that the descriptor contains the function to be executed and a memory location of data upon which the function is to be executed.

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

189. 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;

190. Lurie teaches this. As I explained with respect to Elements 14[c]-[d], and [h] above, 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].

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

191. 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;

192. Lurie teaches this claim element.

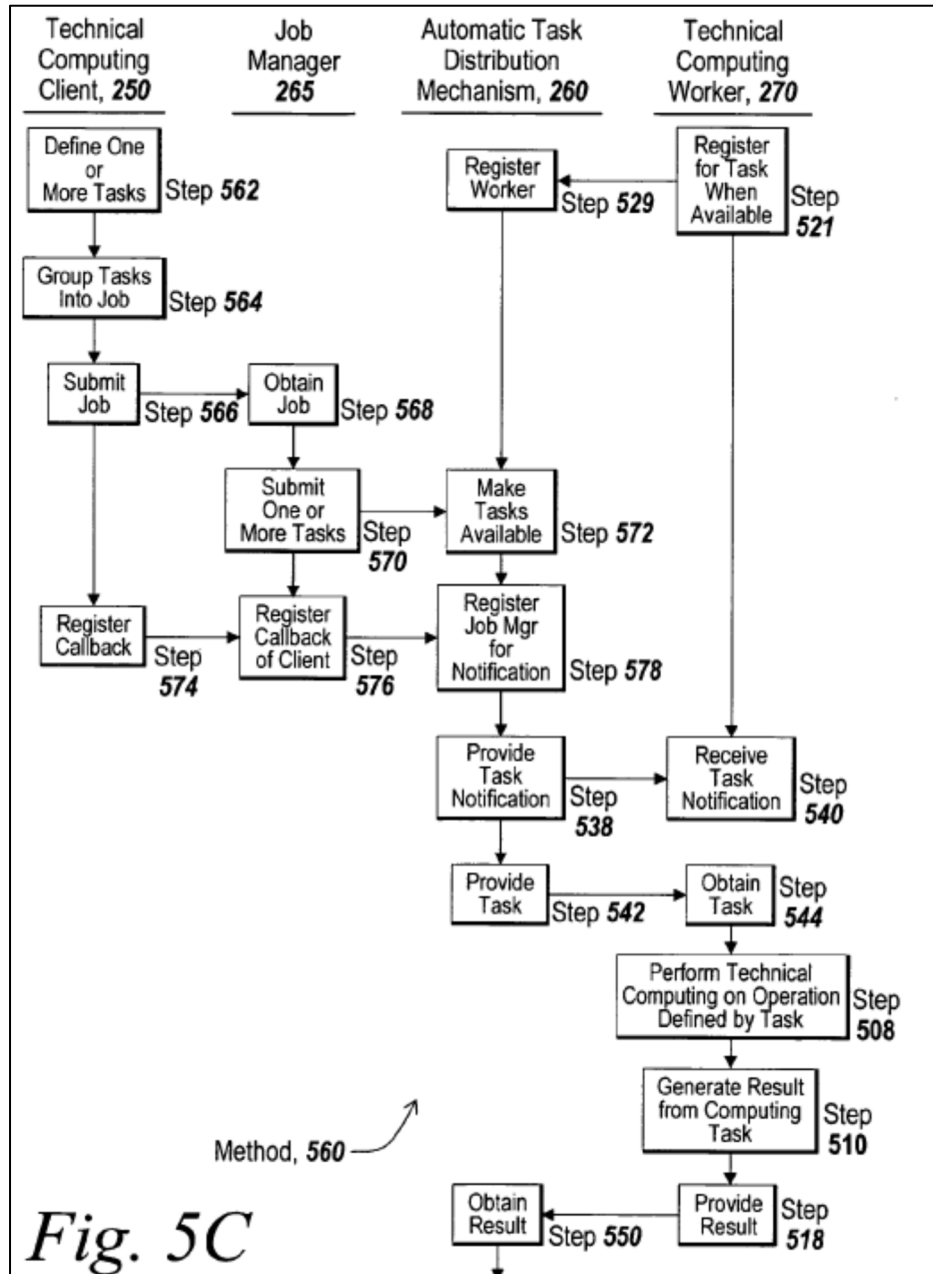
193. As I discussed above with respect to Elements 14[c.2], [d.2], [h], Lurie's TCWs send their agents (data frames/packets) to the ATD, without an instruction from the TCC, to proactively retrieve a matching task as recited in steps 14[h.1]-[h.3]. I also note that Lurie's ATD—including its incorporated job manager (Lurie [0063]) "obtains the set of results for the completed job ... and

provides the results of each of the tasks to the [TCC].” Lurie [0074]. The TCC does not communicate with the TCWs because the ATD provides a level of indirection between the TCC and the TCWs.

194. And the ATD does not direct or instruct the TCWs to retrieve tasks. Rather, the ATD notifies all of the TCWs when tasks are first available, as I also explained above with respect to Elements 14[c.2], [d.2], [h]. Further, the ATDs continue to proactively dispatch their agents to the task pool when they complete a task and therefore do not have a matching task to process. Lurie walks the reader (a POSITA) through the first task performed by a TCW upon receiving a notification that tasks are now available. That POSITA would have understood that each TCW performs more than one task—POSITAs at the time ensured that there would be at least 2-3 times as many tasks in a job as there are workers. Andrews 133. As Andrews explains, this ensures adequate load balancing in systems (such as Lurie) with heterogenous tasks In other words, a POSITA would understand that Lurie’s TCWs loop back, or would at least implement them to do so, to ensure all of the tasks in the ATD (task pool) are completed—the same as the textbook bag-of-tasks model.

195. To explain further, the ATD sends a notification to all of the registered TCWs when the ATD first makes available a task (or tasks, e.g., a job). Then, each

registered TCW will search for, and if one is available, obtain a matching task, complete it, and provide results.



Lurie Figs. 5C-5D, [0086]-[0088]. After providing the result of the first task, however, Lurie's TCWs *do not* halt and wait for another notification. Rather, they return to step 544, and continue to retrieve and complete tasks, without waiting for

any notification or instruction, until the ATD Mechanism has no more tasks. *See* Lurie at [0087]-[0088], [0093]-[0097] (describing bag-of-tasks). For example, Lurie teaches two methods for retrieving tasks, *getTask* and *getTaskIfAvailable*, the latter of which returns null if the ATD Mechanism has no tasks in it. Lurie [0128]. If each TCW waited for a notification before each task retrieval (because the ATD Mechanism purportedly assigns tasks to each TCW), there would be no need for the *getTaskIfAvailable* method.

196. In this sense, Lurie's TCWs perform the textbook bag-of-tasks technique, with one initial notification to kick off the procedure. The initial notification informs all registered TCWs that one or more tasks are available. After that, as described in the textbook bag of tasks paradigm, each TCW continues to retrieve a task from the ATD Mechanism, process it, deposit the result, and then retrieve a new task, repeating this process until the ATD Mechanism is empty (and TCWs are informed as such). Lurie at Figs. 5C-5D, [0087]-[0088]; *see, e.g.*, Andrews 132-133 (workers continue to get and process tasks until bag is empty), 338-340 (same, with the bag manager telling workers the bag is empty by placing a "stop token" in the bag), 427, 429, 561-563 (each workers continues to get and process tasks until it retrieves a "'bag is empty' sentinel"). The only exception is when new TCWs are onboarded while tasks are being processed. In that case, the new TCWs, which are starting the flowchart in Figs. 5C-5D, for the first time,

would need to be notified that tasks are already available. Lurie at [0086]-[0088].

A POSITA would understand that this can be accomplished by the ATD Mechanism sending notification to the new TCW during the registration process.

197. To the extent that this bag of tasks paradigm is not expressly disclosed in Lurie, it would have been obvious to implement Lurie in this manner, as taught, for example, in Andrews. As I described above, Lurie describes the first iteration of the loop in which a TCW retrieves a task. Lurie at [0087]-[0088]. But it does not (and does not need to) explain how the TCW then retrieves its next task. A POSITA would understand that there are only three options: (a) the TCW obtains another task upon completion of its first task, (b) the TCW awaits another notice that tasks are available, or (c) the TCW re-registers and then awaits another notice:

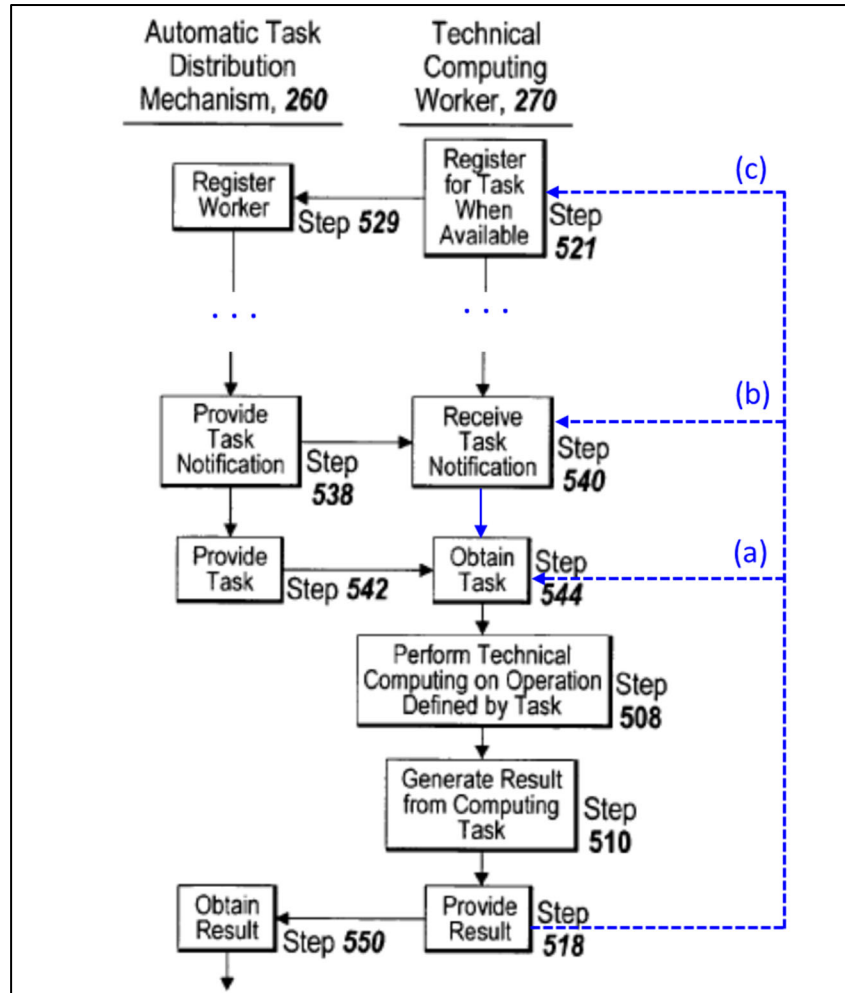


Fig. 5C-D (excerpt, annotated). Option (a) follows the textbook bag-of-tasks paradigm as explained in Andrews, and therefore is preferable to the other two options, and is how a POSITA would understand Lurie to operate. Option (c) would require each TCW to re-register every time it selected a new task, unnecessarily increasing processing overhead, and sending redundant registration messages. Option (b) would require each TCW to halt after each task completion until the ATD Mechanism sends another notification, contrary to the purpose of the bag-of-tasks and distributed processing, which is to minimize idle worker

downtime. This would also again create unnecessary notification traffic. There is no need for the ATD Mechanism to send a repeated notification every cycle in the flowchart, when the TCW could just proceed directly to take the next task (as in Option (a) and the bag of tasks).

198. Accordingly, Lurie discloses that its TCWs proactively retrieve tasks without being instructed by the ATD (or the TCC for that matter). Or, to the extent Lurie does not expressly disclose that, it would have been obvious to implement Lurie in that manner, using the bag of tasks scheme as long taught in textbooks and well within a POSITA's skill, for example, in Andrews.

**B. Independent claim 1**

**1. Preamble**

199. The preamble of claim 1 recites:

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

200. Claim 1's preamble is identical to claim 14's. It is taught by Lurie for the same reasons.

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

201. Like claim 14, claim 1 recites:

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

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

202. These elements are also in claim 14. Lurie teaches them for the same reasons as in claim 14. *See* Elements 14[a]-[b] above.

203. 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;

204. Lurie also discloses this element. Lurie's TCC (the claimed CPU) receives the large computational requirement (as I explained regarding the preamble, e.g., a large simulation), and divides it into "one or more threads" ("jobs" in Lurie's terminology), and places the jobs—which contain one or more tasks—in the task pool (Lurie's ATD, including its job manager). I will explain this in detail in the following paragraphs.

205. As I explained with respect to the preamble, Lurie's system performs the claimed "requirement," such as a large simulation that is computed multiple times with different parameters. Lurie [0004]-[0005].

206. Lurie's TCC creates the one or more jobs, each composed of one or more tasks of the requirement. Lurie [0005] ("[A] client creates a job. The job may include one or more tasks."), [0006] ("The method also includes the step of enabling the client to generate a job in the computing environment, wherein the job includes one or more tasks."), [0008] ("The system includes a first instrument

for generating a job, wherein the job includes one or more tasks.”), [0009] (“The method also includes the step of enabling the client to generate a job in the computing environment, wherein the job includes one or more tasks.”), [0010] (The “job [was] generated by a remote client, wherein the job includes one or more tasks.”), [0043] (“Tasks can be declared on a technical computing client and additionally organized into jobs.”), [0047] (same), [0074] (“The [TCC] defines the job” by “declar[ing] tasks, declar[ing] a job and associate[ing] the tasks with the job); [0075] (The job manager receives and handles multiple jobs), [0077] (similar), [0079] (similar), [0083] (The TCC [technical computing client 250B can operate in batch automated distribution mode but submit jobs separately to a first job manager 265A”), [0087] (“The [TCC] groups one or more tasks of the tasks into a job.”), [0088], [0095] (multiple jobs).

207. Lurie’s jobs are the claimed threads because they form at least a portion of a large computing requirement, and are composed of one or more tasks. *See* Claim Construction section above (“thread”).

208. Lurie’s TCC populates the ATD by submitting the jobs (i.e., the claimed threads) to the ATD via the ATD’s job manager. Lurie [0063] (the ATD “can include the functionality and object-oriented interface of the job manager” as part of a “single application[] or software component”), [0074] (“In batch mode as depicted in FIG. 3C, the technical computing client 250 defines the job. .... Instead

of submitting each task separately as depicted in FIG. 3B, the technical computing client 250 submits the job containing all the associated tasks to the job manager 265.”), [0075], [0078] (In batch mode operation as depicted in FIG. 3D, the technical computing client 250 defines the job comprised of related tasks. Instead of submitting each task separately as depicted in FIG. 3B, the technical computing client 250 submits the job containing all the related tasks to the job manager 265.”), [0087] (“The [TCC] then submits the job (step 566 [of Fig. 5C/D]) to the job manager 265.”), [0088], [0095] (Discussing Fig. 6B: “Under batch processing, tasks are grouped into a job in the [TCC] and then the job is submitted to the job manager 265 for task distribution and task processing by a [TCC].”), [0096] (“Still referring to FIG. 6B, the [TCC] submits a job to the job manager 265....”), Figs. 3C-3D, 5C-5D. And explained in the cited paragraphs, Lurie’s ATD (including its job manager) manages the execution of the tasks. 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].

209. I also note that “the matching task” recited in this claim element is the “matching task” retrieved by the agent(s) for the TCW(s). *See* Claim Elements 14[c.2], [h], and [h.1]-[h.3], discussed above. The matching task(s) is (are) the

tasks that are made available at the ATD and processed by the TCWs, as discussed above.

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

210. 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;

211. Lurie teaches this claim element. Lurie explains that “[a] task defines a technical computing command ... to be executed, and the number of arguments and any input data to the arguments.” Lurie [0043], [0085] (same). Lurie also explains that its TCC defines “functions .... representing a technical computing task” that will be executed by a TCW. Lurie [0056]. For example, “the [TCC] may declare a function to generate a random set of ten numbers.” Lurie [0056], [0061] (“The [TCC] can define a function to submit the task to the [ATD], and get a result of the task from the [ATD].”), [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], [0094] (“The [TCW] obtains the name and arguments of the function to compute from the data structure of the task object.”); [0171]-[0194] (creating a task includes specifying the function and its arguments). Thus, Lurie’s tasks include functions to be executed by the TCWs and the functions’ arguments/parameters.

212. Lurie also teaches that the task includes the functions' arguments/parameters—the actual values—are stored in memory locations of the task. The “name and arguments of the function to compute” are stored in a “data structure of the task object.” Lurie [0094]. Thus, the “arguments,” which are the “data upon which the function is to be executed” as recited in the claims, are stored in a “memory location” of the task. Further, Lurie's jobs store tasks—including their functions and parameters, in a “cell array,” which is 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). Thus, Lurie's tasks include memory locations of data upon which the function is to be executed.

213. Lurie also teaches that the agent may search for and find a task “by associative lookup rather than by memory location or identifier.” Lurie [0091]. Associative lookup is a common technique, not limited to JavaSpaces, that uses a key to find a memory location. *See* Mehlhorn (Ex. 1058) at 81 (“Hash tables are one implementation of associative arrays, or dictionaries.”), 82-83 (explaining the basic concept of using a key for lookup). Thus, a POSITA would have understood from Lurie that its tasks include “a memory location of data upon which the function is to be executed” as recited in the claim.

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

214. 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....

215. Lurie alone and in combination with Ethernet Specification teaches these claim elements for the same reasons as Elements 14[c.1]-[c.2] and [g]-[h], which I discussed above.

216. The only difference between these elements and the corresponding elements in claim 14 is that these recite that the first agent is configured to proactively retrieve the matching task “without requiring an instruction from the CPU.” But as I explained above respect to Elements 14[c.2], [d.2] and [h] and Element 14[f], Lurie’s TCWs send their agents (data frames/packets) to the ATD, without an instruction from the TCC to the agent (or the TCW) to proactively retrieve a matching task.

### **C. Dependent claims 2-13**

217. Lurie, alone or with Ethernet Specification, teaches all of the claim elements recited in these claims for the same reasons explained above for claim 1 and as further explained below.

**1. Claim 2: thread completion notification**

218. Claim 2 recites:

2. The apparatus of claim 1 wherein the task pool notifies the CPU when the tasks of a thread are completed.

219. This is taught by Lurie. The TCC registers with the job manager portion of Lurie's ATD (task pool), which notifies its TCC (CPU) when the job (thread) is completed. Lurie [0063] (Job manager may be part of ATD), [0088] (Job manager uses a callback function to inform the TCC that the job is completed), [0096] (same), Figs. 5C-D, 6B.

**2. Claim 3: task types**

220. Claim 3 recites:

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.

221. This is taught by Lurie. As I explained with respect to Elements 14[h] and 1[g], Lurie's TCWs are configured to perform one or more task types in a list (or set) of task types. As I explained, Lurie discloses myriad different task types,

including different portions or subsets of MATLAB or other platforms such as LabVIEW®, MATRIXx, Mathematica®, Mathcad, Maple™, or none of these.

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

222. Claim 4 recites:

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.

223. Claim 5 recites:

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.

224. Claim 6 recites:

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.

225. Claim 7 recites:

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.

226. These dependent claims do not add anything to claims that I have already discussed. As I explained regarding 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.

**4. Claim 8: task descriptor data storage location**

227. Claim 8 recites:

8. The apparatus of claim 1 wherein the descriptor further contains a memory location where processed data is to be stored.

228. This is taught by Lurie. As I explained with regard to Element 1[e] above, Lurie’s task descriptors are stored in the memory locations that form the task’s data structure. Specifically, the data structure includes the task’s function name, data, and results. In addition, those data structures are in a “cell array,” i.e., an array of memory locations. Those memory locations may be accessed by variable, pointer/reference to a particular memory location, or by associative lookup.

229. 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®.” Lurie [0103]. The memory locations

that will store the “output arguments” are the claimed “memory location where processed data is to be stored.”

**5. Claim 9: task descriptor is a data structure**

230. Claim 9 recites:

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.

231. This is taught by Lurie. As I explained regarding Element 1[e] and again in claim 8 above, Lurie stores task descriptors that include task’s function name, data, and results, all stored in a data structure, and that these data structures may be stored in a cell array.

232. A POSITA would have understood that cell array elements (i.e., the task data structures including the descriptors) are accessed via indices that identify a memory location in the cell array. Thus, a POSITA would have understood that each task contains a reference to the memory location of the descriptor as required by this claim.

**6. Claim 10: task pool in physical memory**

233. Claim 10 recites:

10. The apparatus of claim 1 wherein the task pool occupies a region of physical memory.

234. This is taught by Lurie. Lurie explains that its ATD is on a server, and that it comprises one or more application software components” on the server. Lurie [0060], [0061]. Lurie’s server has “memory 106,” which is physical memory such as “DRAM, SRAM, EDO RAM, etc.,” which stores the ATD. Lurie [0050]. Lurie’s ATD therefore occupies a region of physical memory as required by this claim.

**7. Claim 12: second solidarity cell and agent**

235. Claim 12 recites:

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.

236. This is taught by Lurie alone or in view of Ethernet Specification for the same reasons I explained for Element 14[d]. Any one of Lurie’s TCWs may serve as the first solidarity cell, and any other TCW may serve as the second solidarity cell.

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

237. Claim 13 recites:

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.

238. This is taught by Lurie alone or in view of Ethernet Specification for the same reasons I explained for Element 14[f].

**X. Ground 2: Claim 11 Would Have Been Obvious Over Lurie, Ethernet Specification, and Lippett**

239. Claim 11 recites:

11. The apparatus of claim 10 wherein the task pool is disposed in a hardware block dedicated to the task pool.

240. Claim 11 depends from claim 10 (which in turn, depends from claim 1). Claim 11 would have been obvious for the same reason as claim 10 (and thus its independent claim, claim 1), and for the additional reasons I explain here.

241. Claim 11's claim element is taught by Lurie alone, or Lurie in view of Lippett.

242. As I explained above with respect to claim 10, Lurie teaches that its ATD (task pool) occupies physical memory (e.g., "DRAM, SRAM, EDO RAM, etc.," Lurie [0050]) on a server. To the extent claim 11 covers such memory, the block(s) of physical memory storing the ATD (task pool) is a "hardware block dedicated to the ATD" as recited in the claims. For example, the ATD running on a server resides in the server's memory.

243. If the claims require a hardware block to be always dedicated to the task pool, Lippett teaches that. Lippett uses "dedicated memory" to "improve performance" in scheduling systems because it reduces the amount of time it takes

to read and write to that tightly coupled memory. The use of such “tightly coupled memory” containing “scheduling configuration and dynamic descriptors used to represent units of work within the system” would have been implemented by a POSITA using, for example, “a standard compiler SSRAM technology supplied by numerous EDA vendors.” Lippett [0110], [0440]-[0042]; *see also* Lippett Overview above. A POSITA would have done so to “improve performance” because, as was well known in the art, it reduces the amount of time it takes to read and write to memory.

**XI. Conclusion**

244. I have provided these opinions based on the information currently available to me. I reserve the right to expand or modify my opinions as this proceeding continues and as my investigation and study continues. I may also supplement my opinions in response to any additional information that becomes available to me, including opinions provided by Patent Owner or its expert(s).

245. All statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true. Further, I am aware that these statements are made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001. I declare under penalty of perjury that the foregoing is true and correct.

Executed on 12/19, 2022 in Tucson, AZ.



David Lowenthal

**APPENDIX: CLAIM LISTING FOR '777 PATENT**

The following 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.