UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TESLA, INC.,
Petitioner

v.

Intellectual Ventures II LLC.,
Patent Owner

IPR2025-00638
U.S. Patent No. 8,898,395

Declaration of Robert Colwell, Ph.D. Under 37 C.F.R. § 1.68 In Support of Petition for *Inter Partes* Review of U.S. Patent No. 8,898,395 (Claims 1-2, 5, 7-8, and 11)

Table of Contents

I. Introduction....4

II. Qualifications and Professional Experience....4

III. Materials Considered....7

IV. Level of Ordinary Skill in the Art....10

V. Legal Principles....11

VI. The ’395 Patent....12

A. Overview of the ’395 Patent....12

B. Prosecution History....16

1. The ’127 Application....16

2. The ’531 Application....18

VII. Technology Overview....20

VIII. Claim Construction....23

IX. Prior Art....27

A. Moir....27

B. Martínez....34

1. Availability of Martínez....36

X. Summary of Opinions....38

XI. Analysis....38

A. Ground 1: Claims 1-2, 5, 7-8, and 11 are rendered obvious over Moir in view of Martínez....39

1. Reasons to Combine Moir and Martínez....39

a) Analogous Art....39

b) Reasons to Combine Moir and Martínez 41

2. Claim 1 48

3. Claim 2 76

4. Claim 5 84

5. Claim 7 87

6. Claim 8 89

7. Claim 11 89

XII. Availability for cross-examination 90

XIII. Conclusion 91

I, Robert Colwell, do hereby declare as follows:

I. INTRODUCTION

1. I have been retained by counsel for Tesla, Inc. ("Tesla") as an independent expert witness for the above-captioned Petition for *Inter Partes* Review ("IPR") of U.S. Patent No. 8,898,395 ("the '395 Patent"). I am being compensated at my usual and customary rate for the time I spend in connection with this IPR. My compensation is not affected by the outcome of this IPR.

2. I have been asked to provide my opinions regarding whether claims 1-2, 5, 7-8, and 11 (the "Challenged Claims") of the '395 Patent are unpatentable as they would have been obvious to a person having ordinary skill in the art ("POSITA") as of the earliest claimed priority date.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

3. My complete qualifications and professional experience are described in my curriculum vitae, a copy of which can be found in Exhibit 1004. The following is a brief summary of my relevant qualifications and professional experience.

4. I have over 40 years of professional experience in the field of processor and system architecture design. I consider myself an expert in, among other things, CPU architecture, computer system architecture, computer hardware, and architecture design.

5. I received an undergraduate Bachelor of Science degree in Electrical Engineering from the University of Pittsburgh in 1977. I received a Master of Science degree in Computer Engineering from Carnegie Mellon University in 1978 and a Ph.D. in Computer Engineering in 1985.

6. From 1977 to 1980, I held an engineering position at Bell Telephone Laboratories where I worked on, among other things, microprocessor hardware design.

7. From 1980 to 1984, I held an engineering position at Perq Systems, where I worked on hardware design in computer environments. From 1985 to 1990, I held an engineering position at Multiflow Computer, where I served as a hardware architect. While at Multiflow Computer, I assisted in creating the first very long instruction word (VLIW) scientific supercomputer.

8. From 1990 to 2001, I held various positions at Intel including Senior CPU Architect and later Chief Architect (for Intel's IA-32). As part of my responsibilities at Intel, I assisted in the conception of Intel's P6 microarchitecture that formed the core of the Pentium II manufactured by Intel (as well as the Pentium III, Celeron, Xeon, and Centrino families). In addition, I led Intel's x86 Pentium CPU architecture endeavors. I was honored to be named an Intel fellow in 1997 in recognition of my contributions to the P6 microarchitecture development.

9. I became a self-employed industry consultant in 2001, working with computer industry clients such as Safeware, the University of Pittsburgh, Intel, several venture capital companies, Qualcomm, Samsung, Lawrence Berkeley National Lab, and the U.S. Department of Defense (DoD).

10. From 2011 to 2014, I worked at the Defense Advanced Research Projects Agency (DARPA) first as Deputy Director, then Director, of the Microsystems Technology Office (MTO). MTO had an annual budget of approximately $600M, and my job as office leader was to invest that money in promising new technologies for the DoD, including new energy-efficient computing systems, modular and adaptable radars, position/navigation/timing systems for GPS-denied environments, computer-mediated prosthetics for military (and civilian) amputees, traumatic brain injury detection devices for soldiers, fused multiple-band night vision sensors, extremely high power lasers, and much more.

11. I have been recognized by the industry for my contributions to processor design. I received the Eckert-Mauchly Award in 2005 for "outstanding achievements in the design and implementation of industry-changing microarchitectures, and for significant contributions to the RISC/CISC architecture debate." The Eckert-Mauchly Award is generally viewed as the highest possible recognition in the field of computer architecture.

12. I was inducted into the National Academy of Engineering in 2006, the nation's highest honorary society for engineering achievement. In 2012 I was inducted into the American Academy of Arts and Sciences; other inductees in my "class" that year included Sir Paul McCartney, Hillary Rodham Clinton, and Mel Brooks.

13. I have published many conference papers, sections of textbooks, and articles for magazines. I am a named inventor on 40 patents related to computer hardware and processor design. I have also been an editor for several IEEE publications, and have served on numerous conference committees, including IEEE Micro.

14. My curriculum vitae (Ex.1004) includes a list of all publications I have authored in the last ten years.

15. In summary, I have extensive familiarity with computer hardware, processors, computer architectures, unified memory architectures, and methods related to controlling memory access, and I am familiar with what the states of these technologies were at the relevant time of the '395 Patent invention and before.

III. MATERIALS CONSIDERED

16. In preparing this Declaration, I have reviewed:

- Ex.1001, the '395 Patent;

- Ex.1002, Prosecution History of U.S. Application No. 12/121,531 (issued as the ’395 Patent) (“’395 FH”);
- Ex.1005, U.S. Patent No. 7,206,903 to Moir et al. (“Moir”);
- Ex.1006, “Speculative synchronization: programmability and performance for parallel codes.” J. F. Martínez et al. IEEE Micro, December 2003) (“Martínez”);
- Ex.1007, Prosecution History of U.S. Application No. 11/102,127 (issued as US 7,376,798) (“’798 FH”);
- Ex.1009, Landing Page, IEEE Micro Vol. 23, No. 6, Internet Archive Capture, Feb. 1, 2004, https://web.archive.org/web/20040201212609/http://www.computer.org/micro/;
- Ex.1010, Table of Contents, IEEE Micro Vol. 23, No. 6, Internet Archive Capture, Feb. 14, 2004, https://web.archive.org/web/20040214122853/http://csdl.computer.org/comp/mags/mi/2003/06/m6toc.htm
- Ex.1011, U.S. Pat. Pub. No. 2002/0199066 to S. Chaudhry et al. (“Chaudhry”);

- Ex.1012, "Transactional Memory: Architectural Support for Lock-Free Data Structures," M. Herlihy and J. E. B. Moss, ISCA '93: Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 289-300, May 1993 ("Herlihy and Moss Paper");
- Ex.1013, U.S. Pat. No. 5,428,761 to Herlihy and Moss. ("Herlihy and Moss Patent");
- Ex.1019, Shen & Lipasti, Modern Processor Design, McGraw Hill 2005;
- Ex.1020, Concise Encyclopedia of Computer Science, Wiley 2004;
- Ex.1021, The Cache Memory Book, Jim Handy, 2nd ed., Academic Press 1998;
- Ex.1022, Hennessy & Patterson Computer Architecture: A Quantitative Approach 3rd ed;
- Ex.1023, IEEE Authoritative Dictionary of IEEE Standards Terms, 7th edition, 2000
- Ex.1024, Microsoft Computer Dictionary, Fifth Edition; and
- any other document cited below.

17. In forming the opinions expressed in this Declaration, I have considered:

a) the documents listed above,

b) any additional documents and references cited in the analysis below,

c) the relevant legal standards, including the standard for obviousness, and

d my knowledge and experience.

IV. LEVEL OF ORDINARY SKILL IN THE ART

18. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

19. For my analysis, I have been asked to assume that the time of the claimed invention is the filing date of the ’395 Patent, which is April 7, 2005.

20. A person of ordinary skill in the art at and before the claimed priority date of the ’395 Patent (April 7, 2005) (“POSITA”) would have had a bachelor’s degree in electrical engineering, computer engineering, computer science, or a related field, and 2-3 years of experience with speculative execution of instruction

groups or threads in a shared memory multiprocessor. Less experience may be sufficient with additional education.

V. LEGAL PRINCIPLES

21. I have been asked to provide my opinions regarding whether claims 1-2, 5, 7-8, and 11 of the '395 Patent are anticipated or would have been obvious to a person having ordinary skill in the art at the time of the alleged invention, in light of the prior art. I have been informed by counsel that, to anticipate a claim under 35 U.S.C. § 102, a reference must teach every element of the claim. Further, I have been informed that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed by counsel that the obviousness analysis takes into account factual inquiries including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

22. I have been informed by counsel that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the following: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to

obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

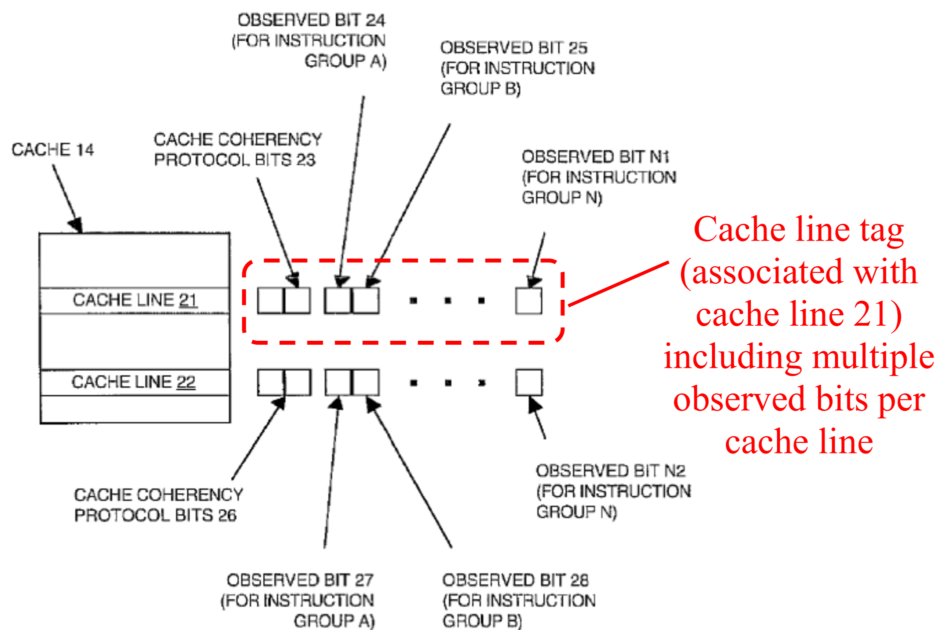
VI. THE ’395 PATENT

A. Overview of the ’395 Patent

23. The ’395 Patent is concerned with managing memory consistency in multiprocessors in which “operation reordering, optimization and speculation” are facilitated by allowing instruction groups to be “committed and rolled back atomically.” Ex.1001, 1:47-62. A POSITA would have understood these multiprocessor techniques for coordinating transactional (atomic) access to sets of memory locations to be examples of transactional memory.

24. More specifically, the ’395 Patent is directed to “maintain[ing] sequential consistency for lumped (in-order or out-of-order) [execution in multiprocessor] architectures.” Ex.1001, 1:66-2:1, 1:47-48. The ’395 Patent indicates that “[p]rior attempts to achieve a sequentially consistent, lumped architecture are limited…to a single memory operation per instruction group.”

Ex.1001, 2:1-5. To "permit sequentially consistent, lumped architectures in which multiple memory operations can be included in instruction groups," "observed bits" are provided (in addition to the conventional cache coherence protocol bits) in cache line tags associated with the cache lines of a cache of a processor. Ex.1001, 8:51-53, 4:44-46, 33-34. The '395 Patent's Fig. 2 below illustrates an example of a cache and its associated cache line tags including the observed bits. Ex.1001, 4:32-60.



Ex.1001, Fig. 2 (annotated)

25. Fig. 2 shows a cache 14 of a processor having a first cache line 21 and a second cache line 22, where the first cache line 21 is associated with coherency protocol bits 23 and observed bits 24, 25…N1, and the second cache line 22 is

associated with coherency protocol bits 26 and observed bits 27, 28…N2. Ex.1001, 4:32-48. The cache coherency protocol bits 23 and 26 are "used to indicate the state of the cache lines 21 and 22, respectively, according to a cache coherency protocol such as the MESI [Modified, Exclusive, Shared, and Invalid] protocol." Ex.1001, 4:41-43.

26. Each observed bit (e.g., observed bit 24 for the first cache line 21) extends the cache line tag of a cache line for each group of instructions that may be executed by the processor (e.g., processor 10 with its associated cache 14). Ex.1001, 4:51-56. For example, in Fig. 2, the cache line tag associated with cache line 21 is extended by N bits for N groups of instructions that may be executed by processor 10 associated with cache 14. Ex.1001, 4:51-56. Groups of instructions are committed or rolled back atomically— *i.e.*, as a unit, all together or not at all. Ex.1001, 1:51-60, 4:11-12; Ex.1024 (Microsoft Computer Dictionary defining "atomic transaction [as a] set of operations that follow an 'all or nothing' principle, in which either all of the operations are successfully executed or none of them is executed."). In this way, for a given instruction group, either all instructions are executed to completion and all state changes are committed, or the entire group of instructions is atomically rolled back, meaning that the speculative results created by that group of instructions are discarded and not permitted to change architectural state. Ex.1001, 4:11-12, 7:59-62. If execution of an instruction in a group of instructions causes a cache line

to be accessed (e.g., a load, store, read, or write), then the observed bit is set. The particular observed bit set is that associated with the particular group of instructions and belonging to the cache line tag of the particular cache line accessed. Ex.1001, 4:61-65. With respect to Fig. 2, for example, "observed bit 24 is set if an instruction in instruction group A, when executed by processor 10, causes cache line 21 to be accessed." Ex.1001, 4:66-5:1, Fig. 2.

27. The '395 Patent explains that "when execution of a group of instructions is ended (e.g., the instruction group is committed, rolled back, or aborted), each observed bit set by that group of instructions is cleared…." Ex.1001, 5:19-27. A group of instructions is atomically[1] rolled back and reissued when the following two conditions are met: (1) an observed bit (e.g., observed bit 24) is set (indicating that execution of an instruction of the group of instructions associated with the observed bit (e.g., instruction group A) has caused the associated cache line (e.g., cache line 21) to be accessed), and (2) another processor or a peripheral device

[1] The atoms that constitute all matter were once thought to be indivisible. Hence the word 'atomic' in this context is meant to convey that either *all* of the instructions within a designated "group of instructions" are committed to architectural machine state, or *none* of them are (the roll back case).

executes an instruction that requests access to that same cache line (e.g., cache line 21) or causes the cache line (cache line 21) to be accessed. Ex.1001, 5:50-56.

B. Prosecution History

28. It is my understanding that the ’395 Patent issued from U.S. Application No. 12/121,531 (the ’531 application). The ’531 application is a continuation of U.S. Application No. 11/102,127 (the ’127 application), issued as U.S. Patent No. 7,376,798 (the ’798 patent). Ex.1002, 235, 1; Ex.1007, 2.

1. The ’127 Application

29. The ’127 application was filed with its broadest claims reciting the use of an indicator, associated with a first group of instructions, to indicate a cache line of a cache memory in a computer system has been read when an instruction of the first group of instructions is executed. The ’127 application claims recited that the first instruction, when executed, causes the cache line to be read. The claims also recited maintaining the indicator until the execution of the first group of instructions is ended. Application claim 1 of the ’127 application read as:

> 1. A method of managing memory in a computer system, said method comprising:
>
> executing a first group of instructions, wherein said first group of instructions comprises multiple memory operations and a first instruction that when executed causes a cache line of a cache memory to be read; and
>
> in response to said first instruction causing said cache line to be read, changing an indicator associated with said first group of instructions to indicate that said cache line has been read, wherein said cache line is so indicated as having been read until execution of said first group of instructions is ended.

Ex.1007, 178-186.

30. In a subsequent Office Action, the Examiner rejected all the claims as being anticipated and obvious over prior art references. Ex.1007, 117-133. The applicant then amended the claims to recite that "multiple groups of instructions are executable in parallel by [the processor and that] said indicator comprises a bit per cache line for each group in said groups of instructions." Ex.1007, 102. The Examiner maintained the rejections. Ex.1007, 61-80. The applicant further amended the claims to recite (1) a second group of instructions and (2) a first and second bit of the indicator, where each such bit is set to indicate that a respective first or second group of instructions has caused the cache line to be read. Ex.1007, 43-53. The Examiner then allowed the amended claims, stating that the limitation "in response to said second instruction causing said cache line to be read, setting a second bit of said indicator while said first bit remains set to indicate said cache line has been

accessed by both said first group and said second group of instructions" was not taught by the prior art of record. Ex.1007, 17-18.

31. Allowed claim 1 of the '127 application read as:

> 1. (Currently Amended) A method of managing memory in a computer system, said method comprising:
>
> executing a first group of instructions on a first processor, wherein said first group of instructions comprises multiple memory operations and a first instruction that when executed causes a cache line of a cache memory to be read; ~~and~~
>
> in response to said first instruction causing said cache line to be read, ~~changing~~ setting a first bit of an indicator associated with said cache line ~~first group of instructions~~ to indicate that said cache line has been read, wherein said cache line is so indicated as having been read until execution of said first group of instructions is ended, wherein multiple groups of instructions are executable in parallel by said first processor and wherein said indicator comprises a bit ~~per cache line~~ for each group in said groups of instructions;
>
> executing a second group of instructions on said first processor, wherein said second group of instructions comprises multiple memory operations and a second instruction that when executed causes said cache line to be read; and
>
> in response to said second instruction causing said cache line to be read, setting a second bit of said indicator while said first bit remains set to indicate said cache line has been accessed by both said first group and said second group of instructions.

Ex.1007, 43.

2. The '531 Application

32. The '531 application, which issued as the '395 Patent, presented a new set of claims reciting subject matter that broadened the allowed claims of the '127 application. Ex.1002, 241-248. The claims retained, however, the first group of

instructions and the second group of instructions, as well as the cache-line-associated first and second bits of the indicator. Ex.1002, 241. Dependent application claim 35, which depends on independent application claim 33 via intervening dependent application claim 34, was one of the dependent claims that were indicated as containing allowable subject matter. Ex.1002, 165-166. Claims 33-35 read as:

33. (Previously Presented) A method of managing memory in a computer system, said method comprising:

setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions; and

setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions.

34. (Previously Presented) The method of Claim 33 further comprising:

executing a third group of instructions that causes said cache line to be accessed; and

processing said first group and said second group of instructions according to a value of said indicator.

35. (Previously Presented) The method of Claim 34 wherein said third group of instructions is executed by an agent other than said processor, wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access.

Ex.1002, 135 (annotated to emphasize antecedent basis[2]).

33. The Applicant incorporated application claims 34 and 35 into independent prosecution claim 33, which issued as claim 1 of the '395 Patent. Ex.1002, 32; Ex.1001, Claim 1. The Examiner allowed all the pending claims and identified the limitations of prosecution claim 35 and the limitation of prosecution claim 34 reciting, "executing a third group of instructions that causes said cache line to be accessed," as the limitations that were not taught by the prior art considered during prosecution. Ex.1002, 13. Independent claim 1 of the '395 Patent corresponds to application claim 33 that was amended to include the above-described limitations of application claims 34 and 35.

34. However, claim 1 recites well-known concepts that were taught by prior art not considered by the Examiner.

VII. TECHNOLOGY OVERVIEW

35. The '395 Patent relates to transactional memory techniques. Techniques to implement transactional memory (*e.g.*, as described by Herlihy and Moss) were known for more than a decade prior. Ex.1012, 289 *et seq.* (Herlihy and

---

[2] The Claim Construction section that follows addresses how "said processing" in the claim language as issued would be understood in view of the prosecution history, the claim language, and the specification of the '395 patent.

Moss Paper, describing transactional memory as multiprocessor architecture that may be implemented by straightforward extensions to a multiprocessor cache coherence protocol and that allows programmers to define customized read-modify-write operations that apply to multiple, independently-chosen words of memory); Ex.1013, 3:25-35 (Herlihy and Moss Patent, likewise describing transactional memory as providing the ability to implement customized read-modify-write operations that affect arbitrary regions of memory). By 2004, transactional memory and speculative execution techniques had been widely discussed in the academic literature.

36. The alleged novelty, according to the '395 Patent as issued (and the parent application 11/102,127, the "'127 application," as originally filed April 7, 2005), was handling more than a single memory operation per instruction group. Ex.1001, 2:1-7 (alleging that "[p]rior attempts to achieve sequential consisten[cy were]…limited to a single memory operation per instruction group" and that "[e]mbodiments in accordance with the present invention overcome this disadvantage"); *see also*, Ex.1007, 160 (same explanation in '127 application, as filed). During prosecution of the '127 application, when faced with clear teaching of prior art U.S. Patent 6,938,130 to Jacobsen—a transactional memory reference—the applicant pivoted to assert that novelty instead derived from the utilization of "multiple indicator bits associated with each cache line in a cache." Ex.1007, 54-55

(arguing that "Jacobsen appears to utilize only one 'store-marking' bit per cache line, in contrast to the claimed invention").

37. However, neither idea—not multiple memory operations per instruction group and not multiple indicator bits per cache line—was novel in April of 2005. Extensions of conventional multiprocessor cache coherence protocols with cache line marking to support atomic transactions involving speculatively executed sequences of instructions, <u>including multiple memory operations</u> (e.g., loads and/or stores) that define the read set or write set of a transaction was described at least as early as 1993 in Herlihy and Moss' seminal work entitled "Transactional Memory: Architectural Support for Lock-Free Data Structures." Ex.1012. In that work, and in a corresponding patent (U.S. Patent No. 5,428,761 (Ex.1013), filed in 1992 and issued to Herlihy and Moss in 1995), use of a transaction bit associated with each cache line was described. *See* Ex.1012 (Herlihy and Moss Paper), 291-92 (augmenting cache line states with transactional tags to detect conflicting memory accesses conflicts and based thereon atomically commit or abort the group of instructions that constitute a transaction); Ex.1013 (Herlihy and Moss Patent), 7:1-14, 10:58-11:41 (transactional memory implemented by extending any standard write-back protocol to mark each cache entry either transactional or non-transactional and using the dirty bit to code shared or exclusive and thereby detect

memory access conflicts and atomically commit or abort multi-instruction sequences).

38. In summary, more than a decade before the ’395 Patent’s earliest effective filing date (April 7, 2005), extensions to conventional cache coherence techniques using bits associated with cache lines to mark memory locations involved in transactional accesses, to detect memory access conflicts between groups of instructions, and to atomically commit or abort multi-instruction sequences, were all well documented. Claim recitations of multiple bits per cache line, of rollback, and of MESI-based cache coherence protocols are mere design or linguistic variations on the basic Herlihy and Moss techniques and terminology. In any case, and as detailed in the analysis below, all limitations of the challenged claims are disclosed and rendered obvious by Moir (Ex.1005) and Martínez (Ex.1006)—two complementary prior art references that detail concrete implementations of Herlihy and Moss’s transactional memory in modern shared memory multiprocessors.

VIII. CLAIM CONSTRUCTION

39. It is my understanding that in order to properly evaluate the ’395 Patent, the terms of the claims must first be interpreted. It is my understanding that the claims are to be construed according to the same claim construction standard that district courts use. Under the claim construction standard that district courts use (i.e., the so-called *Phillips* standard), claim terms are given their ordinary and customary

meaning from the perspective of a POSITA at the time of the invention. I have been informed that that the phrase "ordinary and customary meaning" in the so-called *Phillips* standard is often referred to as the "plain and ordinary meaning." I will use the phrase "plain and ordinary meaning" below.

40. In order to construe the Challenged Claims of the '395 Patent, I have reviewed the entirety of the '395 Patent along with portions of the prosecution history of the '395 Patent. Consistent with the '395 Patent disclosure, I have interpreted the terms in the Challenged Claims according to their plain and ordinary meaning as understood by a POSITA, and have considered any explicit definitions provided by the specification of the '395 Patent, where applicable.

41. I have been asked to provide my opinion regarding the construction of "said processing," and have been informed that the antecedent basis for the term is contested. The '395 specification fails to inform a POSITA with any reasonable certainty which structure performs "*said processing*" that "*comprises rolling back said first group of instructions…and rolling back said second group of instructions*." *See, e.g.*, Ex.1001, 5:54–56 (describing the result—"then the instruction groups associated with the observed bits that are set are forced to roll back"—without identifying the responsible structure), 5:58–61 (same), 6:6–11 (same), 7:59–62 (same, with reference to step 43 in Figure 4). Accordingly, I rely on the prosecution history to inform my construction of "said processing." For purposes of this

proceeding and to demonstrate applicability of the grounds herein to a construed claim, I construe "*said processing*" in [1.3B] as referring to the [1.4] "*processing of said first and said second group of instructions according to a value of said indicator*"—that is, the processing of such groups of instructions is *by said processor*, rather than by "*an agent other than said processor*."

42. As detailed in the prosecution history of the '395 patent (*see supra*, Section VI.B.2), claim 1 issued from subject matter recited in a dependent claim (claim 35) that was indicated allowable if rewritten in independent form. Ex.1002, 156, 165 (rejecting claims 33-34, but indicating "[c]laim[] 35…would be allowable…if rewritten in independent form including all of the limitations of the base claim and any intervening claims."). A POSITA would have understood "said processing" in the allowable claims to refer to the antecedent language of application claim 34, as reflected in the annotations below:

33. (Previously Presented) A method of managing memory in a computer system, said method comprising:

setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions; and

setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions.

34. (Previously Presented) The method of Claim 33 further comprising:

executing a third group of instructions that causes said cache line to be accessed; and

processing said first group and said second group of instructions according to a value of said indicator.

35. (Previously Presented) The method of Claim 34 wherein said third group of instructions is executed by an agent other than said processor, wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access.

Ex.1002, 135 (claims in Applicant's after-final response annotated to trace antecedent basis). Following affirmation of the Examiner's rejection during prosecution, the Applicant amended the claims in light of a prior indication that claim 35 was allowable if rewritten, such that representing that "Claim 33 is amended to incorporate the subject matter of allowable Claim 35 and intervening Claim 34." Ex.1002, 122, 40-41. Claim 33 so amended issued as claim 1 of the '395 patent. Ex.1002, 32; Ex.1001, Claim 1. Accordingly, this understanding of "said processing" is in line with the prosecution history.

43. Further, in the analysis below, claim limitation [1.4] is addressed prior to limitation [1.3] so as to present a coherent analysis of the term "said processing," for which a prior antecedent otherwise does not exist, in line with the above construction of "said processing." For the purposes of this proceeding and the grounds presented herein, no other claim term requires express construction.

IX. PRIOR ART

A. Moir

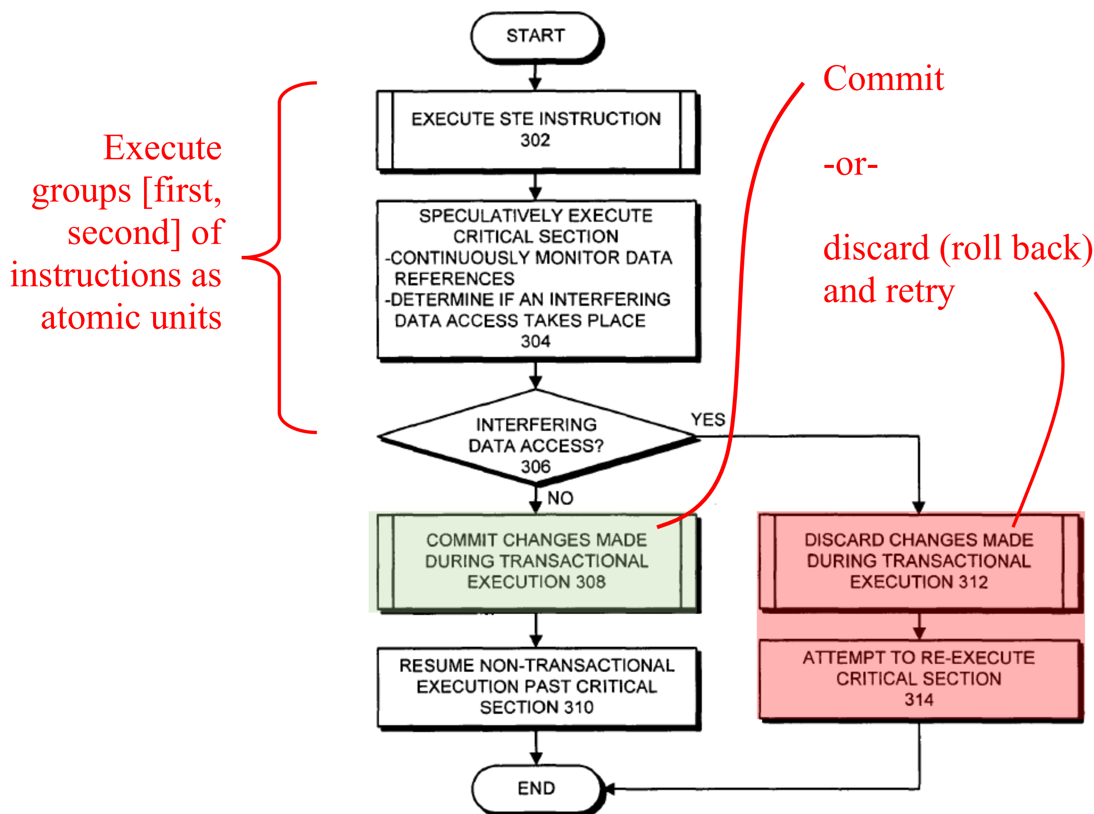
44. Moir is directed to techniques for improving the performance of shared memory multiprocessor computer systems by managing accesses to memory locations during transactional program execution. Ex.1005, 1:25-34. Moir describes techniques that seek to avoid the performance overhead and complexity of acquiring and releasing locks to coordinate multithreaded access to shared data structures. Ex.1005, 1:43-2:4. To this end, Moir "transactionally execute[s] a critical section, wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution completes without encountering an interfering data access from another thread." Ex.1005, 2:23-29. During transactional execution, cache lines that are accessed are marked, allowing interfering data access to be determined. Ex.1005, 2:34-44.

45. Moir proposes advanced techniques to "facilitate early release of memory locations during transactional program execution," to "reduce[] the number

of cache lines that need to be marked during transactional program execution" using "explicit time-to-live value" codings, to accommodate nested transactions. Ex.1005, 2:20-22, 1:27-29, 3:1-6, 3:35-38, 4:11-22. These are advanced considerations beyond the scope of what is taught by the '395 patent. However, Moir also describes the basics of a transactional memory implementation, delimiting a group or critical section of instructions to be either atomically committed or rolled back by "execut[ing] a start-transactional-execution (STE) instruction before entering the critical section. If the critical section is successfully completed without interference from other threads, the thread performs a commit operation, to commit changes made during transactional execution." Ex.1005, 7:26-32.

46. Fig. 3 of Moir below illustrates the transactional execution process, where "[a] thread first executes an STE instruction prior to entering of a critical section of code (step 302). Next, the system transactionally executes code within the critical section, without committing results of the transactional execution [block 304]." Ex.1005, 7:63-67. If no interfering data access occurs before the transactional sequence ends, then architectural state changes made during the transactional execution are committed (step 308); otherwise, if an interfering data access has

occurred, changes made during the transactional execution are discarded[3] and execution is "rolled back" to the start of the transactional sequence (step 312). Ex.1005, 8:1-12.



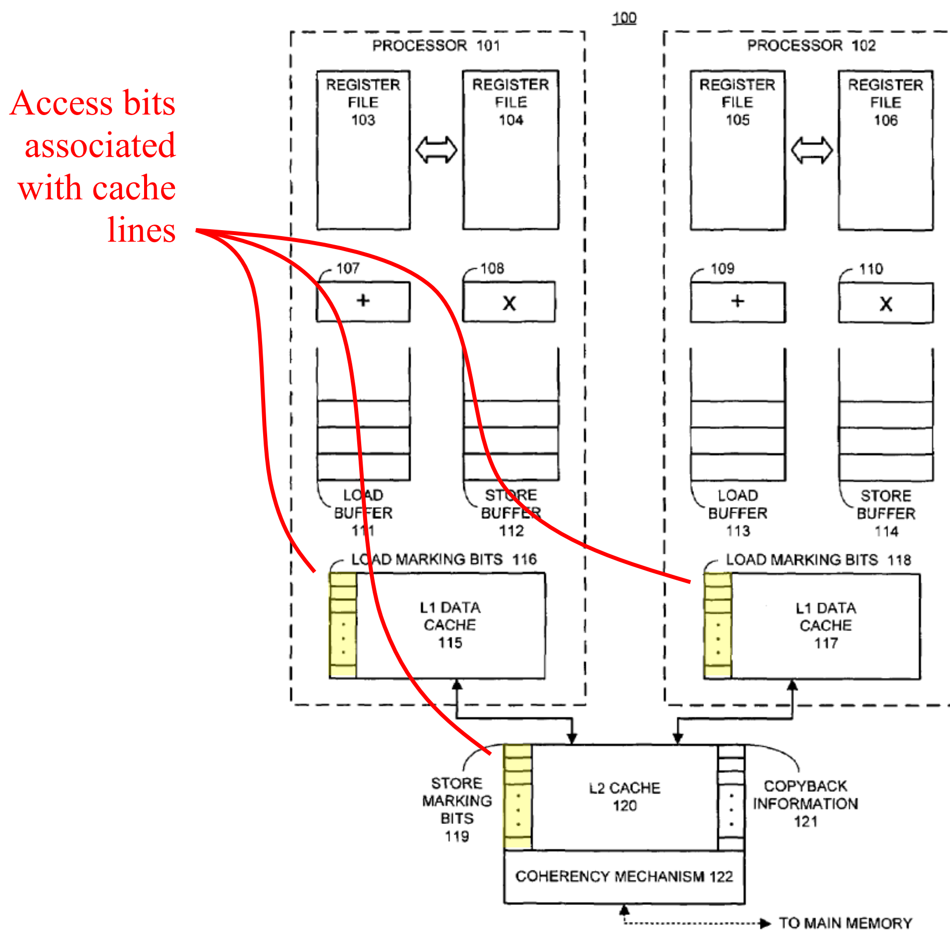
Ex.1005, Fig. 3 (annotated)

47. Transactional execution after "an STE instruction prior to entering of a critical section of code," shown as step 302 in Fig. 3 above, "involves load-marking

---

[3] Discarding changes in Moir's Fig. 3 (312), and the following reissue of instructions (314), are steps taken in a roll back.

and store-marking cache lines, if necessary, as well as monitoring data references in order to detect interfering references." Ex.1005, 7:63-64, 8:65-67. Fig. 1 of Moir below shows a "computer system 100 [that] includes processors 101 and level 2 (L2) cache 120," where "[p]rocessor 101 additionally includes a level one (L1) data cache 115, which stores data items that are likely to be used by processor 101." Ex.1005, 6:9-11. "[L]ines in L1 data cache 115 include load-marking bits 116, which indicate that a data value from the line has been loaded during transactional execution. These load-marking bits 116 are used to determine whether any interfering memory references take place during transactional execution." Ex.1005, 6:29-34. The computer system 100 can have additional processors such as processor 102 which are similar to processor 101. Ex.1005, 6:11-13.



Ex.1005, Fig. 1 (annotated)

48. The load-marking operation is performed on a per-word basis within a cache line. "If [a] load causes a cache hit, the system 'load-marks' the **corresponding word** in the cache line in L1 data cache 115 []. This involves setting the load-marking bit for the cache line. Otherwise, if the load causes a cache miss, the system retrieves the word accessed in the cache line from further levels of the memory hierarchy [], and proceeds to step 506 to load-mark the cache line in L1 data cache 115." Ex.1005, 9:6-16.

49. Moir provides for load marks on a word-granular basis. Ex.1005, Fig. 5 ("load mark word in cache line"), 4:5-7 ("a load-marked cache line contains a bit for each word indicating whether the word has been load-marked"), 4:36-29 ("system load-marks a corresponding word in the cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread"), 9:5-16 (system load-marks the corresponding word in the cache line in L1 data cache). Accordingly, and because it is fundamental knowledge that a cache line contains multiple words, a POSITA would have understood Moir to disclose multiple load marking bits per cache line. Store-marking is also performed, in which case "the system first prefetches a corresponding cache line for exclusive use." Ex.1005, 9:23-25. Moir's illustrated embodiment store-marks at an L2 cache for a configuration with write-through L1 semantics, but POSITA would understand write-marking at any appropriate level in a caching hierarchy.

50. Committing all changes made during transactional execution includes "the system treat[ing] store-marked cache lines as though they are locked []. This means other threads that request a store-marked line must wait until the line is no longer locked before they can access the line … Next, the system clears load-marks from L1 data cache 115." Further, the system "commits entries from store buffer 112 for stores that are identified as needing to be marked, which were generated during

the transactional execution, into the memory hierarchy []. As each entry is committed, a corresponding line in L2 cache 120 is unlocked." Ex.1005, 9:65-10:9.

51. Discarding changes made during the transactional execution includes the system "clear[ing] load-marks from cache lines in L1 data cache 115 [], and drain[ing] store buffer entries generated during transactional execution without committing them to the memory hierarchy." Ex.1005, 10:60-63. Further, "the system unmarks corresponding L2 cache lines." Ex.1005, 10:65-66. In addition, "the system [may] branch[] to a target location specified by the outermost STE instruction []. The code at this target location optionally attempts to re-execute the critical section…or takes other action in response to the failure, for example backing off to reduce contention." Ex.1005, 10:67-11:5.

52. Moir's above-discussed techniques for handling critical sections of code allow "early release of memory locations during transactional program execution" because when "the system receives a release instruction during transactional execution of a sequence of instructions within an application," the system clears a bit that had previously been set and, "if all bits associated with a cache line are cleared, the cache line is unmarked by virtue thereof." Ex.1005, 1:28-29, 13:4-6, 9-11. The benefits of early release include "cache lines that are unmarked [not having] to remain in cache memory until the transaction completes (or is killed),

and false failures [being] less likely to occur due to a large number of cache lines being marked." Ex.1005, 13:18-21.

B. Martínez

53. Martínez is directed to allowing "[a]pplication threads [to] execute speculatively past active barriers, busy locks, and unset flags instead of waiting." Ex.1006, 126, 127. To do so, threads are extracted from an application and submitted "for speculative execution in parallel with a safe thread" of the application. Ex.1006, 126. This guarantees continual progress of the application's execution because "safe threads cannot get squashed or stall" while "offending speculative threads [can be] squash[ed] and restart[ed] on the fly." Ex.1006, 127 (about safe threads), 126 (about speculative threads).

54. Martínez discloses that its techniques for allowing processors to execute code speculatively can be implemented in shared-memory multiprocessors using a "speculative synchronization unit (SSU), which consists of some storage and some control logic that [Martínez adds] to the cache hierarchy of each processor in a shared-memory multiprocessor." Ex.1006, 129. A processor uses the SSU to execute code speculatively, and "[w]hen a processor reaches an acquire point," such as the one depicted in Fig. 1a, it requests the SSU for a lock. Ex.1006, 129. "The SSU sets its Acquire and Release bits, fetches the lock variable into its extra cache line," and performs additional steps "to obtain lock ownership." Ex.1006, 129.

55. After the processor crosses the acquire point, it "continues execution into the critical section. As long as the Acquire bit is set, the SSU deems speculative all the processor's memory accesses after the acquire point in program order." Ex.1006, 130. When the processor accesses a cache line (e.g., of L1 or L2 above in Fig. 2) speculatively, "[t]he SSU uses the Speculative bit to locally mark cache lines that the processor accesses speculatively." Ex.1006, 130. Martínez's speculative bits are analogous to Moir's load- and store-marking bits.

56. Access conflicts manifest as "a thread receiving an external invalidation to a cached line, or an external intervention (read) to a dirty cached line." Ex.1006, 130. An originator thread of such external messages[4] to lines not marked speculative is not quashed whether it is a safe thread or a speculative one; however, "[i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread." Ex.1006, 130.

---

[4] It is reasonable to characterize cache coherence bus or memory traffic as "messages". While some system data traffic is associated with cache coherence protocols, such as a dirty-line writeback, other cache coherence traffic is not data per se. For instance, snoops only need the address and the type of access for an intended store operations, not the store data, in order to manage system caches correctly.

If the speculative thread that triggered the squash is an external read to a dirty speculative line in the cache, "the node replies without supplying any data. The coherence protocol then regards the state for that cache line as stale and supplies a clean copy from memory to the requester. This is similar to the case in conventional MESI (modified-exclusive-shared-invalid) protocols." Ex.1006, 130.[5]

1. Availability of Martínez

57. I personally knew of the IEEE website and accessed it before the priority date of the '395 Patent. Indeed, before the priority date of the '395 Patent, I received and accessed IEEE publications such as issues of IEEE Micro magazine, and was familiar with the IEEE both as a long-time member as well as due to my personal involvement in the organization.

58. As reflected in Ex.1008 and Ex.1009, the IEEE made available publications, including the IEEE Micro publications, on the IEEE website. *See* Ex.1008 (Internet Archive capture from Feb. 1, 2004, of the IEEE web page listing the November/December 2003 issue). Ex.1009 reflects that POSITAs would have

---

[5] A POSITA would have understood conventional coherence protocols, such as conventional MESI protocols, to use messages to convey state information. This information is passed between caches to inform other caches of interference or clearance of shared cache lines.

been able to access the individual publications included in the IEEE Micro publication following the release of the corresponding volume. *See* Ex.1009 (Internet Archive capture from Feb. 14, 2004, of the table of contents page and corresponding purchase and download links for the November/December issue of IEEE Micro magazine, which included Martínez on pages 126-134). Notwithstanding the IEEE paywall, a substantial portion of POSITAs would have been IEEE and/or IEEE computer society members at the time and would have had IEEE digital library access to IEEE publications such as Martínez and the other publications appearing in the November/December 2003 issue (Vol. 23, No. 6) as well as professional interest therein. *See also* Ex.1009 ("The full text of IEEE Micro is available to members of the IEEE Computer Society who have an online subscription and an web account.").

59. As an IEEE Member—and IEEE Fellow—I personally have accessed and reviewed IEEE publications and have authored numerous articles appearing in IEEE publications and journals. Additionally, I have served on many conference committees, including the organizing committee for the IEEE Micro conference. In my experience, many people in the field routinely read IEEE publications, including the IEEE Micro publications. Indeed, many POSITAs would have been IEEE Computer Society members, and would have received IEEE Computer Society publications and had access to the publications online.

60. It is thus my opinion that Martínez would have been publicly accessible to POSITAs, and indeed, would have been accessed by POSITAs, more than a year before the earliest priority date of the '395 Patent.

X. SUMMARY OF OPINIONS

61. It is my opinion that claims 1-2, 5, 7-8, and 11 of the '395 Patent are obvious over Moir in view of Martínez.

XI. ANALYSIS

62. I have been asked to provide my opinion as to whether the Challenged Claims of the '395 Patent would have been obvious in view of the prior art. The discussion below provides a detailed analysis of how the prior art references I reviewed disclose, teach, or suggest the limitations of the Challenged Claims of the '395 Patent.

63. I describe in detail below the scope and content of the prior art, as well as any differences between the claimed subject matter and the prior art, on an element-by-element basis for the Challenged Claims of the '395 Patent. This analysis supports my finding that the limitations of the Challenged Claims of the '395 Patent are disclosed in the prior art, or that the differences between the Challenged Claims of the '395 Patent and the prior art discussed herein are such that

the subject matter as a whole would have been obvious to a POSITA at the time of the claimed invention.

64. I describe, in the grounds below, on an element-by-element basis, how the prior art discloses or teaches all elements of the Challenged Claims. Unless otherwise noted, all *italics* refers to claim language, and **bold** or ***bold italics*** is added for emphasis.

A. Ground 1: Claims 1-2, 5, 7-8, and 11 are rendered obvious over Moir in view of Martínez

65. It is my opinion that the combined teachings of Moir and Martínez disclose or render obvious, to a POSITA, the subject matter of each and every element of claims 1-2, 5, 7-8, and 11 of the ’395 Patent.

1. Reasons to Combine Moir and Martínez

a) Analogous Art

66. Moir and Martínez are analogous art to the ’395 Patent because Moir and Martínez each pertain to the same field of endeavor as the ’395 Patent, processor methods and techniques for managing memory consistency in a multiprocessor where groups of instructions execute atomically—also known as transactional execution or transactional memory.

67. The ’395 Patent describes this transaction approach to memory in terms of cache consistency and atomic commitment or rollback. Ex.1001, abstract

("Methods and systems for maintaining cache consistency are described."), 1:15-17 (embodiments "relate to computer system memory, in particular the management of cache memory"), 1:51-60 (in lumped architectures, "instructions are lumped into instruction groups that can be committed and rolled back atomically"), 8:48-50 (seeking to "maintain sequential consistency for lumped [] architectures").

68. Moir is likewise directed to this transactional memory technique. Ex.1005, 3:45-47 (Moir provides "a method and apparatus to facilitate early release of transactional memory"), abstract (providing "a system for releasing a memory location from transactional program execution" operating by "executing a sequence of instructions during transactional program execution"), 8:4-8 (Moir "atomically commits all changes made during transactional execution").

69. Martínez also presents the transactional memory technique. Ex.1006, 127 (explaining that its thread-level speculation (TLS) mechanism builds on "[s]peculative synchronization [to] deal[] with explicitly parallel application threads that compete to access a synchronized region" of memory. … [I]f two speculative threads issue conflicting accesses, the hardware always squashes one of them and rolls it back to the synchronization point.…A speculative thread keeps its (speculative) memory state in a cache until it becomes safe. At that time, it commits (makes visible) its memory state to the system."), 128 ("all the speculative threads beyond the release point, one by one in some nondeterministic order, execute the

critical section atomically"). Accordingly, Moir and Martínez are in the same field of endeavor as, and analogous art to, the '395 Patent.

70. Moir and Martínez are also reasonably pertinent to a particular problem addressed by the '395 Patent, namely, maintaining sequential consistency despite interfering access by competing groups of multiple instructions that include multiple memory operations. Ex.1001, 1:66-2:7, *see also* Ex.1001, 2:17-21 ('395 Patent stating that if "the cache line…is indicated as having been accessed, then the instruction group is rolled back and reissued"); Ex.1005, 8:9-12 (Moir stating that "if an interfering data access is detected, the system discards changes made during the transactional execution…and attempts to re-execute the critical section"); Ex.1006, 127 (Martínez stating that "[i]f two conflicting accesses cause a dependency violation, the hardware rolls the offending speculative thread back to the synchronization point and restarts it on the fly."). Accordingly, and for this reason as well, Moir and Martínez are analogous art to the '395 Patent.

b) Reasons to Combine Moir and Martínez

71. As discussed above in Section IX.A, Moir discloses that if there is an interfering data access (or other type of failure) during transactional execution of a critical section of code by a thread, changes made during the transactional execution are discarded, allowing the thread "to revert back to the conventional technique of acquiring a lock on the critical section before entering the critical section." Ex.1005,

8:10-22. This is enabled by a "flash copy operation [that] checkpoints enough state to be able to restore the state to the state before the beginning of the outermost transaction[6]" and corresponding STE operation executed by the thread. Ex.1005, 8:55-57.

72. Moir does not explicitly discuss, however, the timing of the reversion or rolling back of the thread *vis-à-vis* the timing of the interfering data access. As such, although a POSITA would recognize from Moir alone to revert a thread executing a critical section of code back to its state before the critical section when there is an interfering data access, the conventional implementation details of the *timing* of the reversion or roll back are not the focus of Moir. Once interfering access is identified, rollback is inevitable and, absent some architectural efficiency to delaying rollback, sooner (rather than later) avoids additional wasted speculative execution and allows a processor to more quickly resume useful work once rollback has been completed. In this manner, continued execution at the expense of delayed rollback would result in 1) wasted effort by the processor in execution instructions

---

[6] Moir's implementation of transactional memory supports nested transactions.

For purposes of the present unpatentability analysis of the Challenged Claims, restoring state applies with or without transaction nesting—including where "the outermost transaction" devolves to just "the transaction."

that will be rolled back, 2) stalling the execution of the thread where instructions are being executed due to the wasted effort, and 3) stalling the execution of other threads that need to access a dirty cache line until the cache line can be resolved. A POSITA would therefore have recognized that design choices regarding timing of reversion or roll back would affect overall system computational efficiency and that timing taught by Martínez beneficially avoids wasted computation.

73. A POSITA therefore would have looked to Martínez for such conventional implementation details regarding timing of the reversion or roll back. As discussed above in Section IX.B, Martínez suggests the required timing because it provides that the reversion or roll back of a thread having an access conflict would occur before supplying any data, e.g., stalling the requester pending rollback. More specifically, Martínez discloses that "[i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread." Ex.1006, 130. Further, "[o]nce the SSU triggers a squash, it…**forces the processor to restore** its checkpointed register state, which **results in the thread's rapid rollback to the acquire point**." Ex.1006, 130. And "[i]f an external read to a dirty speculative line in the cache triggered the squash, the node replies **without supplying any data**." Ex.1006, 130. A "clean copy from memory [is then supplied] to the requester." Ex.1006, 130. Thus, Martínez provides implementation details of the timing of the reversion or roll back of a thread having an access conflict

with another thread. This timing avoids holding up an interfering access unnecessarily and avoids wasted execution.

74. A POSITA would have been motivated to combine the teachings of Moir with Martínez because it is the combination of prior art elements (Moir teaching reverting a thread executing a critical section of a code back to a point before entering the critical section if there is an interfering data access from another thread, with Martínez teaching that such roll back be performed before supplying data to the external, and interfering, thread). Moreover, the combination involves known techniques according to known methods (rolling back a speculative thread before supplying data to another thread causing an access conflict with the speculative thread, as taught by Martínez) to yield the predictable result of reverting transactional execution of multiple critical sections based on an interfering data access from another thread and word-level marking of the involved cache line (as taught by Moir) before supplying data to the interfering thread.

75. Further, a POSITA would have had a reasonable expectation of success in making the combination because the combination provides the inevitable outcome of the interfering data access (rollback) in a manner that avoids unnecessary inefficiency or wasted execution. As noted above, continued execution at the expense of delayed rollback would result in 1) wasted effort by the processor in executing instructions that will be rolled back, 2) stalling the execution of the thread

where instructions are being executed due to the wasted effort, and 3) stalling the execution of other threads that need to access a dirty cache line until the cache line interference can be resolved. The combination of Moir and Martínez simply brings together a small set of design techniques that are individually already well understood and interoperable (e.g., marking words in a cache line, detecting an interfering access, observing standard cache coherence protocols, and reverting a processor's execution back to a previously saved state. Accordingly, because Martínez provides efficiency-improving timing in the context of an analogous rollback, a POSITA would have had a reasonable expectation of success in the combination.

76. A POSITA also would have been motivated to combine the teachings of Moir with Martínez because Martínez describes the use of a known technique (coherence protocols that track the state of the cache line) to improve similar methods in the same way (as a coherency mechanism in Moir). Moir explains that it

employs "a coherency mechanism,"[7] and while it does not explicitly describe the conventional protocols it does reference a co-pending application (published as Ex.1011), naming two of his co-inventors, that does provide an example of a cache

---

[7] Caches speed up computations because computer programs typically exhibit predictable patterns in their memory accessing. Programs that repeatedly access certain variables in memory will run faster if those variables are accessible in a local fast cache, rather than residing in much slower main memory. But this convenience comes with considerable system complexity. In a multiprocessor where each processor has its own L1 cache, multiple different processors may require access to the same variables (or different variables that happen to reside in the same cache line.) It is imperative that these caches look coherent to a programmer. The IEEE Authoritative Dictionary of IEEE Standards Terms, 7th edition, 2000 defines cache coherence as follows: "A system of caches is said to be coherent with respect to a cache line if each cache and main memory in the coherence domain observes all modifications of that same cache line. A modification is said to be observed by a cache when any subsequent read would return the newly written value." Ex.1023, 135. In the context of Moir, a POSITA would understand that cache coherence protocol is synonymous with cache coherence mechanism.

coherence protocol. Ex.1005, 6:45-59; *see* Ex.1011, [0139] (describing a "cache coherency protocol [that] includes all of the usual state transitions between the following MOESI states: modified (M), owned (O), exclusive (E), shared (S) and invalid (I) states"). Consistent with Moir's reference to a co-pending application, Martínez itself expressly teaches a "coherence protocol [that] regards the state for that cache line as stale and supplies a clean copy from memory" in a manner similar to conventional MESI (modified-exclusive-shared-invalid) protocols. Ex.1006, 130. Thus, in making the combination, a POSITA would have been motivated to implement a coherence protocol (e.g., MESI or MOESI) that uses the state for a cache line, per Martínez, in its coherency mechanism.

77. Moreover, a POSITA would have had a reasonable expectation of success in making the combination because Moir explicitly calls for the use of a coherency protocol, and at minimum strongly suggests implementing a conventional MESI/MOESI protocol. Ex.1005, 6:45-59. Martínez provides such a protocol, explaining that its approach "is similar to the case in conventional MESI (modified-exclusive-shared-invalid) protocols[.]" Ex.1006, 130. Even the example coherence protocol shown in Hennessy & Patterson amounts to only a minor variation on MESI. *See* Ex.1022, 557. Accordingly, because Moir calls for a coherency protocol, and suggests that a conventional very- well- known protocol would be implemented,

and because Martínez provides such a coherence protocol, a POSITA would have had a reasonable expectation of success in the combination.

2. Claim 1

[1.0] A method of managing memory in a computer system, said method comprising:

78. To the extent that the preamble, [1.0], is limiting, Moir discloses it.

79. Specifically, Moir discloses methods in the context of "a system for releasing a memory location from transactional program execution." Ex.1005, abstract, 3:45-47. Thus, the disclosed methods by which that system operates constitute a *method of managing memory in a computer system*. More specifically, such methods cause "[t]he system to operate[] by executing a sequence of instructions during transactional program execution, wherein memory locations involved in the transactional program execution are monitored to detect interfering accesses from other threads, and wherein changes made during transactional execution are not committed until transactional execution completes without encountering an interfering data access from another thread." Ex.1005, 3:45-54. Moir's system is a *computer system*, and the release of memory locations involved in the execution of a transactional program thereon is a *method of managing memory in a computer system*.

80. Thus, Moir discloses *a method of managing memory in a computer system*.

[1.1] setting a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions;

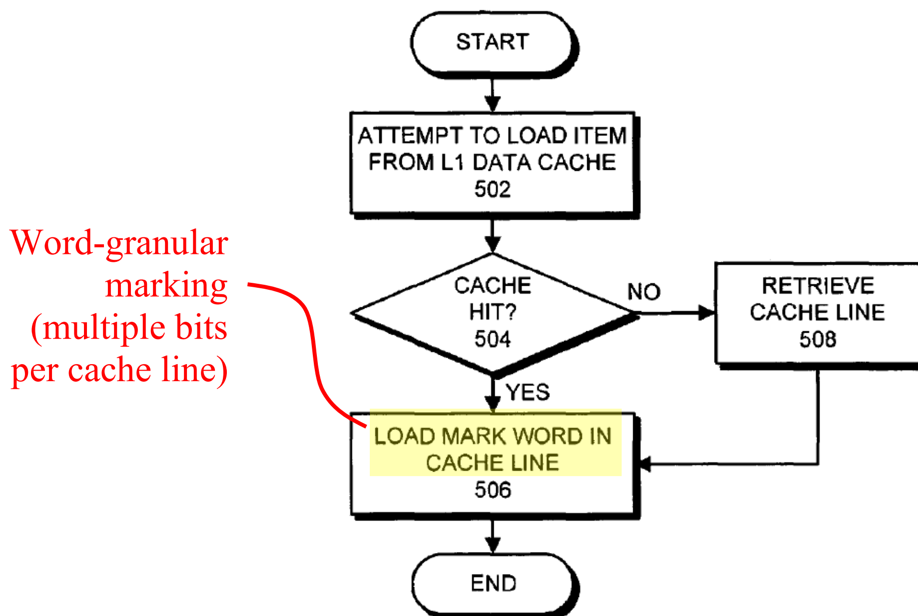
81. Moir discloses limitation [1.1] because Moir describes and illustrates, relative to Fig. 5, "how load-marking is performed during transactional execution." Ex.1005, 9:2-4, Fig. 5.

82. More specifically, Moir describes setting load-marking bits for an L1 cache line (*setting a first bit of an indicator associated with a cache line in a cache memory*) when a accesses a cache hit:

> In performing this load operation [during transactional execution] …, the system first attempts to load a data item from [level one] L1 data cache 115 (step 502). If the load causes a cache hit, the system **'load-marks' the corresponding word in the cache line in L1 data cache** 115 (step 506). This involves **setting the load-marking bit** for the cache line. Otherwise, if the load causes a cache miss, the system retrieves the word accessed in the cache line from further levels of the memory hierarchy (step 508), and proceeds to step 506 to load-mark the cache line in L1 data cache 115.

Ex.1005, 9:5-16. Figure 5 of Moir, below, further reflects load-marking on the sub-cache line basis, *e.g.*, per word, within a cache line. Ex.1005, Fig. 5; *see also*

Ex.1005, Fig. 6 (likewise illustrating the analogous store-marking for write- or store-type memory access operations).



Ex.1005, Fig. 5 (annotated)

83. Moir discloses multiple load-marking bits per cache line because Moir describes load-marking of individual words within a cache line, and a cache line includes multiple words within its cache line width. Caches speed up computer program execution because they provide much faster access to their stored values than does main memory. Per the Concise Encyclopedia of Computer Science (Edwin Reilly, Wiley, 2004) "caches are very effective…because of the principle of locality…which is an empirical observation that most of the time the information in use is either the same information that was recently in use (temporal locality) or is

information 'nearby' the information recently used (spatial locality)." Ex.1020, 85. To take advantage of spatial locality, as well as other system memory and bus characteristics, cache lines are not just one word (4 bytes) long, they are typically 16 words (64 bytes) wide. These 16 words correspond to the same cache tag, which amortizes the cache tag overhead considerably. A 16-word cache line also takes advantage of the wide internal organization of the DRAMs that constitute main memory. For instance, in the Intel processors for which I was chief designer, we used 32 byte L1's for the early processors (Pentium, Pentium II, Pentium III) and 64 byte L1's for all subsequent processors (Pentium 4, Core, Xeon). Similarly, in a SPARC-V9 instruction set architecture that would have been familiar to the Sun Microsystems' inventors in Moir, cache lines in the data cache were 64 bytes and integer load and store instructions supported byte, halfword (16-bit), word (32-bit), and doubleword (64-bit) accesses. Accordingly, a POSITA would understand Moir to be disclosing embodiments at least consistent with then-conventional instruction set architectures (ISAs) such as SPARC-V9, *e.g.*, 16 words per cache line in the data cache and 16 word-granular, load-marking bits, one per cache line entry. Ex.1027, 23 (explaining data types and widths); *see* Ex.1027, 310 (contemplating a 64 byte cache line size).

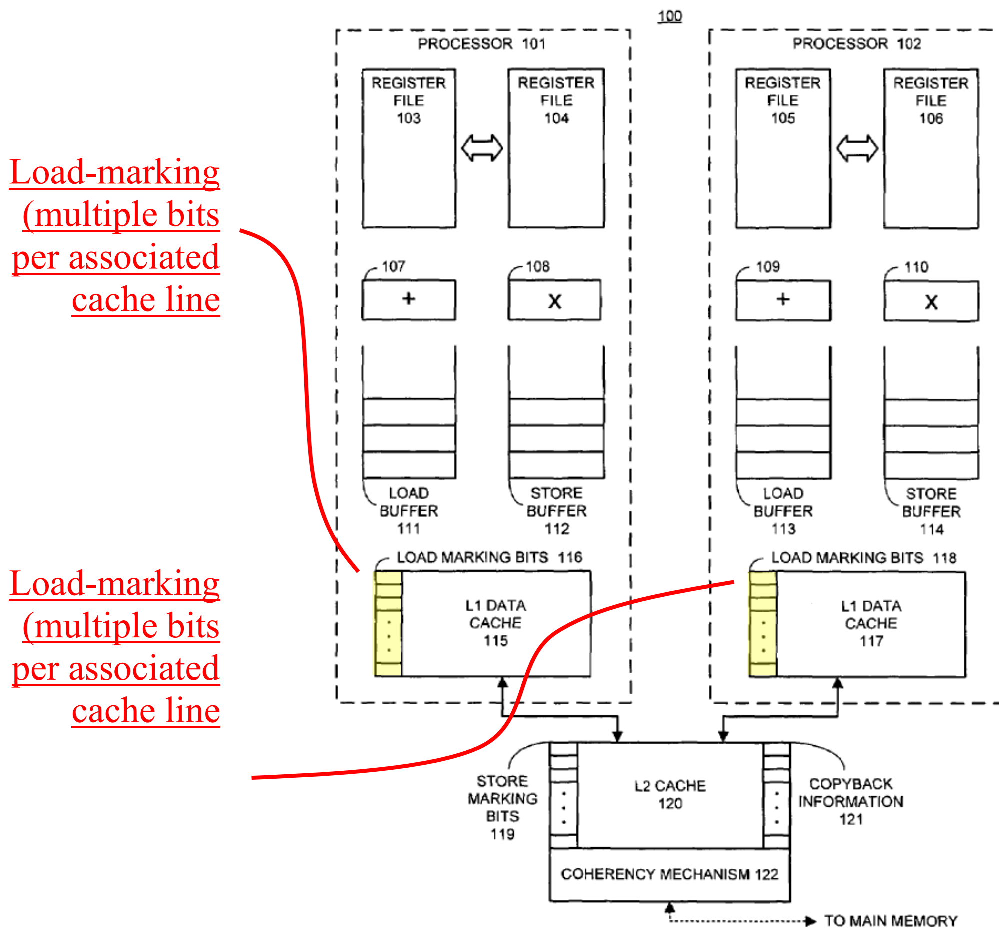
84. Thus, the set of load-marking bits for an L1 cache line constitute *an indicator associated with* that *cache line*, and load marking a particular word

corresponding to the addressable word target of a particular load access instruction is *setting a first bit*[8] *of an indicator associated with a cache line in a cache memory*.

85. Moir describes and illustrates an exemplary multiprocessor computer system 100 in Fig. 1. Ex.1005, 6:1-35 (explaining that Fig. 1 "illustrates a computer system 100" that includes processors 101 and 102). Each processor "includes a level one (L1) data cache" (*e.g.*, L1 data cache 115 and 117) that include "load-marking bits" (*e.g.*, load marking bits 116 and 118) "which indicate that a data value from the line has been loaded during transactional execution." Ex.1005, 6:29-32. Moir explains that the load-marking bits "are used to determine whether any interfering memory references take place during transaction execution[.]" Ex.1005, 6:33-35. Fig. 1 of Moir, below, shows illustrative processors 101 and 102 and the load-marking bits for each cache line of the respective L1 caches for the processors:

---

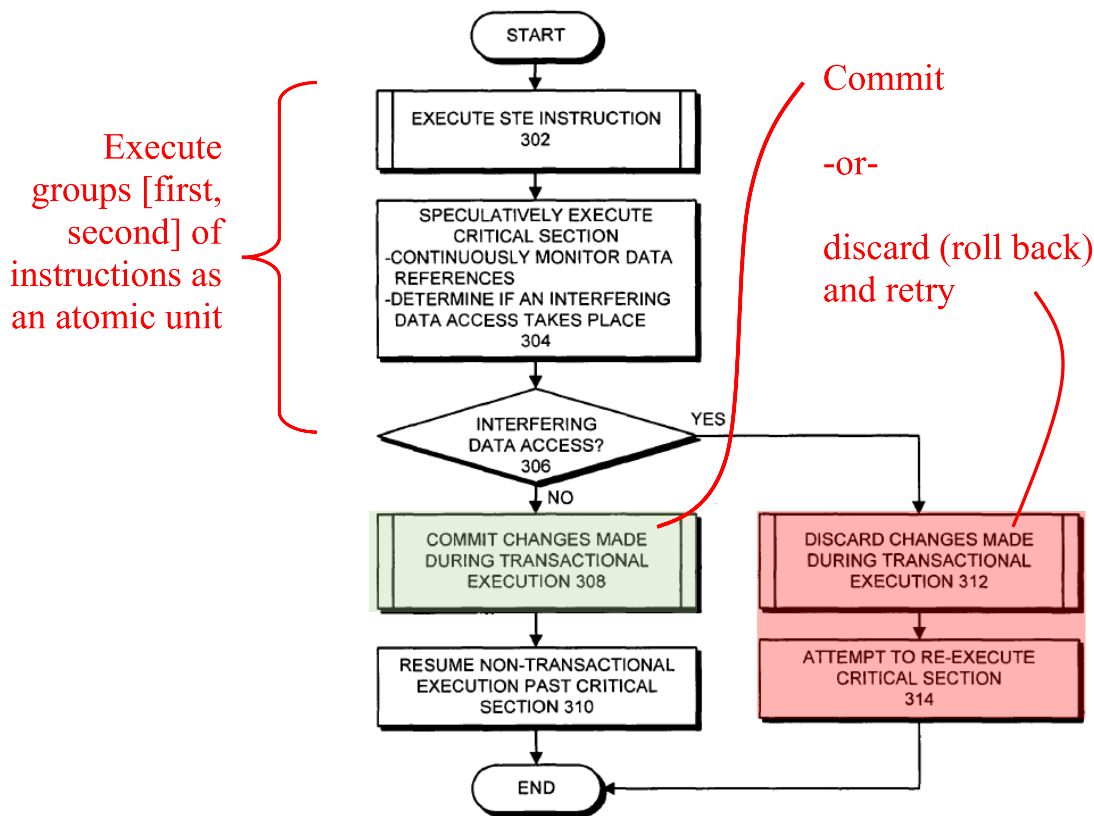
[8] Ordinals *first* and *second* are understood to consistently identify respective bits throughout the claim language re, without implication of an order or position of the respective bits within the indicator. *See, e.g.,* Ex.1001, 8:37-39 ("an observed bit can be set for each cache line accessed").



Ex.1005, Fig. 1 (annotated)

86. It would have been well-understood by a POSITA that processors such as processors 101 and 102 execute instructions. For example, Moir discloses multiple instructions executed in its system that are pertinent to this discussion, including (1) a start-transactional-execution (STE) instruction (*see* Ex.1005, 7:60-8:67, Figs. 3, 4); (2) a monitored load instruction (*see* Ex.1005, 11:59-12:10, Fig. 9B) that performs load-marking; and (3) a commit instruction (*see* Ex.1005,

9:48-10:13, Fig. 7). Figure 3, below, is "a flow chart illustrating how transactional execution takes place[.]" Ex.1005, 7:61-63.



Ex.1005, Fig. 3 (annotated)

87. Moir also explains how its load-marking (and store-marking) of cache lines is used to detect interfering access and trigger roll back. As reflected in Fig. 3, "[a] thread first executes an STE instruction prior to entering of [sic] a critical section of code (step 302)." Ex.1005, 7:63-64. "During [] transactional execution, the system continually monitors data references made by other threads, and determines if an interfering data access…takes place during transactional execution. If not, the

system atomically commits all changes made during transactional execution…. On the other hand, if an interfering data access is detected, the system discards changes made during the transactional execution (step 312), and attempts to re-execute the critical section (step 314)." Ex.1005, 8:1-8:12. Moir explains "that an interfering data access can include a store by another thread to a cache line that has been load-marked by the thread. It can also include a load or a store by another thread to a cache line that has been store-marked by the thread." Ex.1005, 8:23-27.

88. Insofar as Moir's transactional execution is concerned, a POSITA would have understood that an STE instruction and those that follow in a critical section of instructions through commit or discard/re-execute constitute a *group of instructions* because multiple instructions handled together are a group. Moir specifically explains that "transactional execution…involves load-marking and store-marking cache lines, if necessary, as well as monitoring data references in order to detect interfering references." Ex.1005, 8:64-67. Thus, the critical section that follows an STE instruction includes the monitored load instruction(s) that *sets a first bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions*.

89. Thus, Moir discloses or renders obvious *setting a first bit* (load marking a particular word corresponding to the addressable word target of a particular load

access instruction) *of an indicator associated with a cache line in a cache memory* (set of load-marking bits for a cache line) *if said cache line has been accessed in response to a processor executing an instruction in a first group of instructions* (following an STE instruction).

[1.2] setting a second bit of said indicator while said first bit remains set if said cache line has also been accessed in response to said processor executing an instruction in a second group of instructions;

90. Moir discloses or renders obvious limitation [1.2] because, as discussed for limitation [1.1], Moir describes and illustrates, relative to Fig. 5, "how load-marking is performed during transactional execution," and Moir allows multiple critical sections to separately mark addressable cache line words during transactional execution. Ex.1005, 9:2-4, Fig. 5.

91. For the reasons detailed above with respect to limitation [1.1], Moir's set of load-marking bits for an L1 cache line constitute *said indicator*, and load marking another addressable word within *said* (same) *cache line* is *setting a second bit of said indicator*. Moreover, unless the critical section that follows the STE instruction, which includes a monitored load that *set* the *first bit of* said *indicator* (as described above relative to limitation [1.1]) has been committed or the mark has been released, the first bit remains set and the setting of a second bit is therefore performed *while said first bit remains set*. In Moir's operation, multiple critical sections access addressable words within a same cache line. In general, a nested

critical section of the same thread (or a critical section of another thread entirely) that includes second monitored load instruction targeting a second addressable word of the same cache line (e.g., another addressable word representing another variable) also accesses the same *cache line* and, in Moir's system, sets *a second bit* (the load-marking bit for the second addressable word) of the same cache line.

92. Moir specifically contemplates executing sequences of instructions during transactional program execution, "wherein memory locations involved in the transactional program execution are monitored to detect interfering accesses from other threads, and wherein changes made during transactional execution are not committed until transactional execution completes without encountering an interfering data access from another thread." Ex.1005, 3:45-54. Indeed, that is the definition of a nested transaction, where in the course of executing its code, a transactional thread encounters a new access to another critical section. Moir explains the inability to handle such nested transactions as a shortcoming of preceding designs: "if [procedure] P starts a transaction and calls [procedure] Q, and then Q itself starts and commits a transaction, the effects of Q's transaction should be part of P's transactions in a seamless way." Ex.1005, 2:34-36. A POSITA would have understood that in the ordinary course of multithreaded computations contemplated by Moir, nested critical sections (each constituting a group of

instructions and each including a load-marking instruction) execute on a same processor. *See* Ex.1005, 8:38-67 (describing nested transaction support).

93. More generally, a POSITA would have understood that another monitored load targeting another addressable word within the same cache line—e.g., a second critical section nested within the first—constitutes an instruction that *also accesses said cache line in response to said processor executing* that *instruction in a second group of instructions*. Indeed, this is what distinguishes multithreading from multiprocessing. In multiprocessing, multiple different processors execute code simultaneously. With multithreading, multiple threads of execution are interleaved, one at a time, on one processor. Thus, in each case described above, the *second group of instructions* is executed on *said* (same) *processor* as the first group of instructions.

94. Thus, because Moir allows multiple critical sections to separately mark addressable words within a cache line, and for the reasons explained above with reference to limitation [1.1], a second critical section that follows another STE instruction executed by the same processor (whether for a same or separate thread or as a nested critical section of the same thread) includes the monitored load instruction that *sets a second bit of an indicator associated with a cache line in a cache memory if said cache line has been accessed in response to said processor executing an instruction in a second group of instructions*.

[1.4]* [9] *processing said first group and said second group of instructions according to a value of said indicator.

95. Moir discloses or renders obvious limitation [1.4] because Moir discloses that it atomically commits or discards, for transactionally executed sequences of instructions, based on the load-marking bits for an L1 cache line.

96. **First**, as explained above relative to limitations [1.1] and [1.2], Moir discloses or renders obvious transactional execution on a processor of first and second critical sections that each include the monitored load instructions which perform word-granular, load-marking of cache lines. *See* Ex.1005, 9:5-16 (as applied to load-marking of a first word of a cache line accessed by first critical section and as also applied to load-marking of a second word of the same cache line accessed by a second critical section). These first and second critical sections constitute *first and second groups of instructions*, as claimed.

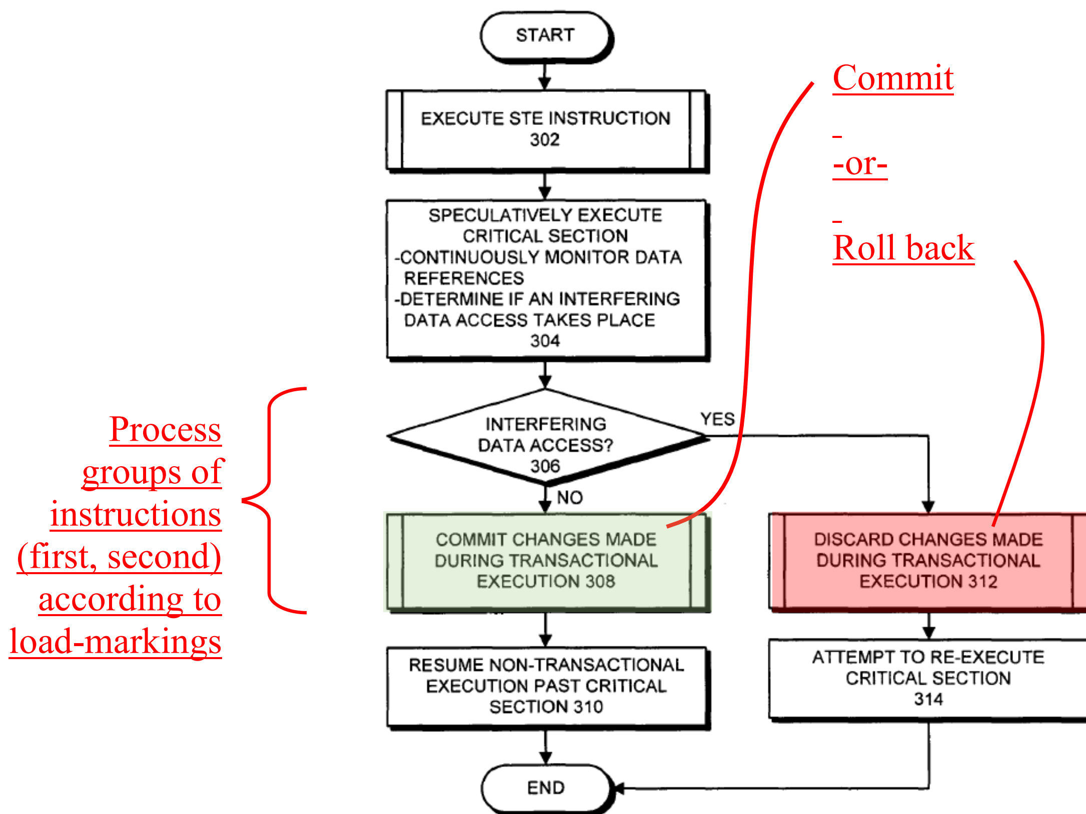
97. **Second**, Moir's technique operates *according to a value of said indicator* because the first and second load-marking bits (together an indicator) are

[9] As noted above, this analysis addresses limitation [1.4] prior to limitation [1.3] so as to present a coherent analysis of the term "said processing," for which a prior antecedent otherwise does not exist, in a manner consistent with the prosecution history.

used to decide whether to commit or roll back. Moir specifically describes relative to block 304 Fig. 3 below that it "transactionally executes code within the critical section[s], without committing results of the transactional execution." Ex.1005, 7:65-67, Fig. 3. "During this transactional execution, the system continually monitors data references made by other threads, and determines if an interfering data access…takes place during transactional execution. **If not, the system atomically commits** all changes made during transactional execution (step 308) …. On the other hand, **if an interfering data access is detected, the system discards changes** made during the transactional execution (step 312), and attempts to re-execute the critical section [block 314]. Ex.1005, 8:1-12, *see also* Fig. 3.

98. Discarding changes made during transaction execution is rolling back. More specifically, a transactional execution instruction sequence begins by checkpointing part of the machine's state, in case the transactional execution sequence ends up being rolled back. Such a rollback requires that (a) any results from the transactional sequence be discarded; (b) the processor be returned to the checkpointed machine state; and (c) instruction processing on the rolled back thread be restarted. Transactional processing is often referred to as "atomic", in the sense that all instructions within the transaction group are treated as one indivisible unit: either the results of all transaction group's instructions are committed to permanent architectural state, or none of them are. Like the '395 Patent, Moir atomically

commits or rolls back (discards) transactionally executed sequences of instructions. Ex.1005, 7:65-67, 8:1-12. Moreover, like the '395 Patent, Moir's decision regarding whether to atomically commit or roll back (discard) its transactionally executed sequences is based on monitoring to detect an interfering access by another thread, using the cache line markings described above relative to [1.1] and [1.2]. Ex.1005, 7:65-67, 8:1-12.



Ex.1005, Fig. 3 (annotated)

99. This process in Moir is *processing said first group and said second group of instructions according to* the word-granular, load-marking of L1 cache

lines previously described. As explained in this section and for limitations [1.1], [1.2], Moir uses load-marking memory access instructions (monitored loads) to mark respective word-granular bits associated with a given cache line. And as further explained in Moir, "an interfering data access can include a store by another thread to a cache line that has been load-marked by the thread [e.g., by execution of a load marking memory access instruction]. It can also include a load or a store by another thread to a cache line that has been store-marked by the thread." *See* Ex.1005, 8:23-27.

100. Thus, Moir's decision to atomically commit or discard (such as for transactionally executed sequences of instructions relative to [1.1] and [1.2]) discloses or renders obvious *processing said first group and said second group of instructions according to a value of said indicator* (load marking bits associated with the L1 cache line).

[1.3A] executing a third group of instructions that causes said cache line to be accessed, wherein said third group of instructions is executed by an agent other than said processor,

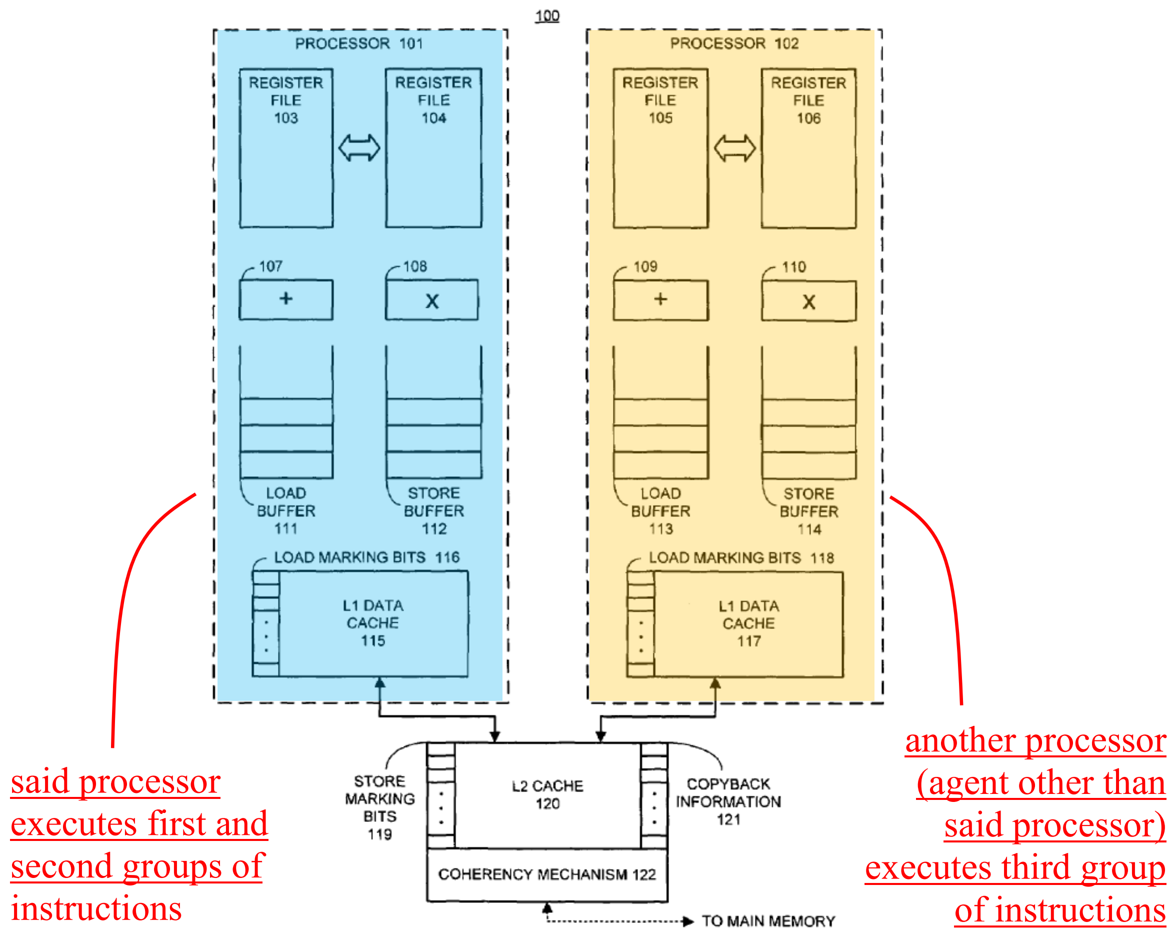
101. Moir and Martínez render obvious limitation [1.3A] because both Moir and Martínez disclose speculative execution techniques for multithreaded computations in a shared memory multiprocessor, and as such, each contemplate *first*, *second*, *third* … and, indeed, nth *groups of instructions* executing as individual threads of a computation running on respective processors of a shared memory

multiprocessor. Both Moir and Martínez assume multiprocessors in which two or more processors share access to the same main memory. Sharing memory affords the processors a fast, efficient means by which to communicate, and simplifies system programming. If there were no caches, the natural serialization that occurs when processors contend for a connection to memory would ensure that memory state is always correct. But multiple distributed L1 caches introduce complex challenges. Chief among these challenges is managing overall accesses to these distributed caches in such a way that no processor ever receives "stale" data, data that does not represent the latest committed state for a given memory value. *See generally* Ex.1021. Both Moir and Martínez provide solutions through their use of standard cache coherence techniques. As mapped above, the first and second groups of instructions execute on a first processor (*see supra*, discussion of limitations [1.1], [1.2]), while the third group of instructions executes on a second processor.

102. More specifically, Moir and Martínez each contemplate speculative execution of groups of instructions (*e.g.*, transactional execution of critical sections or threads) that access shared memory with interfering or conflicting accesses detected based on marked cache lines, and specifically individual words within those cache lines, in which data corresponding to shared addressable memory locations are cached. *See* Ex.1005, Figs. 1, 3, 7:60-8:12, 8:23-35; *compare* Ex.1006, 127 ("[a] speculative thread uses its processor's caches to buffer speculatively accessed

data…. The hardware looks for conflicting accesses—accesses from two threads to the same location that include at least one write….”), Fig. 2, 129 (“local cache…serves as the buffer for speculative data…. [Speculative synchronization unit] SSU keeps one Speculative bit (S) per line in the local cache hierarchy…[and] sets the Speculative bit of a line when the processor reads or writes the line speculatively”).

103. As reflected in Moir’s Fig. 1, below, Moir discloses a system implementing multiple processors, such as processor 101 and processor 102. Ex.1005, 6:1-36, Fig. 1. Moreover, the ’395 patent itself establishes that its use of the term agent encompasses another processor. Ex.1001, 2:15-25 (“external agent (e.g., another processor or a DMA system)”), Fig. 3 (same), Fig. 1 (processors 11, 12 alternatively referenced as agents 11, 12 in associated description).



Ex.1005, Fig. 1 (annotated)

104. Further, Moir explicitly contemplates multiple processors accessing a given cache line because Moir explains that "conventional cache coherence circuitry presently generates signals indicating whether a given cache line has been accessed by another processor" to "determine whether an interfering data access has taken place." Ex.1005, 8:30-35.

105. Thus, Moir discloses, and Moir in view of Martínez renders obvious,[10] *executing a third group of instructions* (a thread with an interfering or conflicting instruction as in Moir or Martínez) *that causes said cache line to be accessed, wherein said third group of instructions is executed by an agent other than said processor* (*e.g.*, a second processor 102 in Moir or any processor in Martínez on which a thread executes an instruction that attempts or performs an interfering or conflicting access).

[1.3B] wherein said processing comprises rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set before allowing an instruction in said third group to access said cache line, and otherwise granting said access;

106. Moir and Martínez render obvious limitation [1.3B] because both Moir and Martínez describe what happens when another group of instructions (e.g., *said third group of instructions*) *is executed by* another *processor* and *causes said cache*

[10] Moir discloses, as does Martínez, the basic framework of a multiprocessor in which memory access by a group of instructions executing on one processor may interfere with access by instructions executing on another processor. The Moir-Martínez combination is noted here for consistency with the analysis of the next limitation of [1.3B] in which the timing of rollback is addressed given the combined teachings of the two references.

line to be accessed in a manner that conflicts or interferes with other accesses such as by the *first* or *second groups of instructions* discussed above with reference to limitations [1.1] and [1.2].

107. Specifically, Moir explains that "if an interfering data access is detected, the **system discards changes made** during the transactional execution…**and attempts to re-execute** the critical section." Ex.1005, 8:9-12 (explaining relative to Fig.3). Likewise, though with greater detail, Martínez explains that "hardware looks for conflicting accesses—accesses from two threads to the same location that include at least one write…. If two conflicting accesses cause a dependency violation, the hardware **rolls the offending speculative thread back** to the synchronization point **and restarts it on the fly**." Ex.1006, 127.

108. Given the mapping of first and second groups of instructions to Moir's processor 101, processor 102 constitutes another processor (an agent other than said processor, *see* [1.3A]) that executes a third group of instructions can initiate an interfering data access. *See* Ex.1005, Fig. 1. The analysis applies equally if the mapping to processors 101 and 102 is reversed. See Ex.1005, 6:11-12 ("Processor 102 is similar in structure to processor 101"). Likewise, a POSITA would understand a processor in the cache-coherent multiprocessors contemplated by Martínez to be illustrative of another processor that initiates the conflicting access. *See* Ex.1006, 133.

109. Because Moir explicitly describes multiple load marking bits associated with a cache line, *e.g.*, word-granular load marking within the cache line, a POSITA would have understood Moir to teach that its discard and re-execute processing relative to a conflicting access to a cache line *comprises* both (i) *rolling back said first group of instructions provided said first bit is set and* (ii) *rolling back said second group of instructions provided said second bit is set*. *See supra*, limitations [1.1] and [1.2] (explaining first and second bits set based on first and second groups of instruction that each include a monitored load to respective words within the same cache line).

110. Martínez, on the other hand, more completely describes the interaction of a conflicting access (e.g., by an instruction in *said third group of instructions*) with a speculatively executed *first* or *second group of instructions*. Specifically, Martínez explains that "conflicts manifest as a thread receiving an external invalidation to a cached line…[and i]f a speculative thread receives an external message for a line marked speculative, the SSU at the receiving node squashes the local thread." Ex.1006, 130. "Once the SSU triggers a squash, it…gang-invalidates all dirty cache lines with the Speculative bit set…and…forces the processor to restore its checkpointed register state, which results in the thread's rapid rollback to the acquire point." Ex.1006, 130. Moreover, "[i]f an external read to a dirty speculative line in the cache triggered the squash, the node replies without supplying

any data." Ex.1006, 130. Indeed, Martínez teaches that its "coherence protocol then regards the state for that cache line as stale and supplies a clean copy from memory" in a manner similar to conventional MESI (modified-exclusive-shared-invalid) protocols. Ex.1006, 130.

111. There are only 4 possible scenarios under MESI for how data transfers would be handled in the event of a possible interfering Load access.

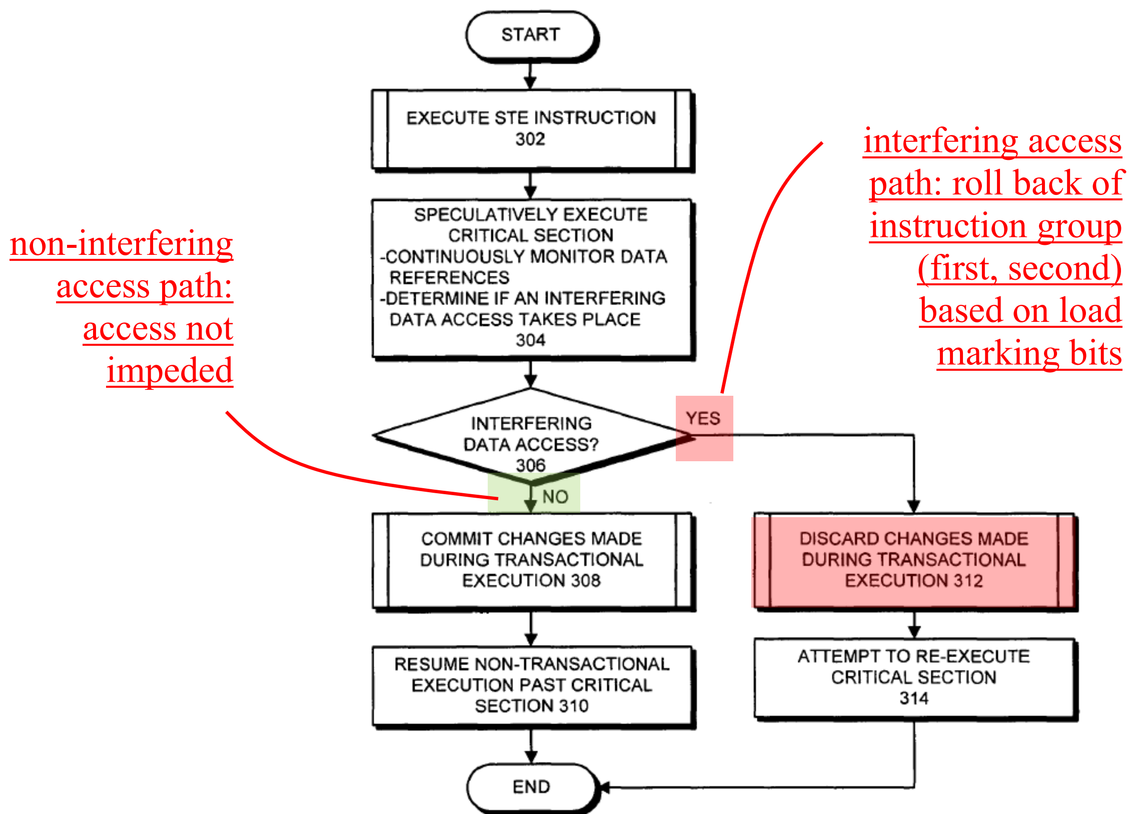
- (Invalid) There is no hit in the L1 cache, so no message is returned during the snoop window and no data is sent or received.
- (Exclusive) The incoming Load hits a cache line marked Exclusive by the MESI protocol. The Exclusive state means that the cache line is exclusively present in this L1, and no other caches, but the line is 'clean', meaning its contents are identical to the corresponding main memory locations. Thus, this access need not trigger a rollback, and the requested data can be supplied by this L1 or by main memory. Per conventional MESI, the L1 cache state changes to Shared.
- (Shared) The incoming Load hits a cache line marked Shared. The Shared state means multiple caches in the system have this line, and its contents match main memory. Thus, this access need not trigger a rollback, and the requested data can be supplied by this L1 or by main memory.
- (Modified) The incoming Load hits a cache line marked Modified, indicating that the line is 'dirty' and its contents do not match main memory.
    i. If there are no load-marks set within the dirty line, then the incoming Load hit causes a dirty line write-back, per conventional MESI protocol. If the system caches have been provisioned to allow "snarfing", then the cache of the requesting Load copies the writeback data as it is being written to memory. Otherwise the requesting Load stalls

until the dirty line writeback completes, and then is satisfied via a direct memory access.

ii. If there are load-marks set within the dirty line, the line is invalidated and a rollback is initiated. No cache hit is indicated during the snoop window, and the Load is filled from memory.

Accordingly, "without supplying any data" means *before allowing* the conflicting instruction *to access the cache line.* Because there are no data transfers associated with a rollback scenario, Martínez teaches rollback *before allowing an instruction in said third group to access said cache line*.

112. Both Moir and Martínez rely on cache line marking and cache coherence circuitry to manage speculative execution of individual threads of computation executing in parallel on respective processors and ultimately roll-back an interfered with thread. Ex.1005, 7:65-67, 8:1-12; Ex.1006, 129-30. Where Moir and Martínez differ is in the aggressiveness of roll-back strategy. Moir does not explicitly require a particular timing of roll back. *See* Ex.1005, 8:1-12, Fig. 3.



Ex.1005, Fig. 3 (annotated)

113. Because Moir is silent as to the status of an interfering third group of instruction's access to a cache-line for which first and/or second groups of instructions have marked respective first and second word-granular load marking bits, a POSITA would reasonably understand the "interfering data access?" check 306 of Fig. 3 to be detection of the **attempt** to access a memory location within a marked cached line, and not necessarily a final check to be performed once and only once at the end of the transactional instruction sequence. A POSITA would understand that there is no point to continuing to execute a transactional sequence

past a detection of an interfering memory access (as opposed to stopping said execution and initiating the rollback procedure). To continue performing "doomed" execution would waste energy and power, unnecessarily delay the restart of the thread, and could also delay the interfering thread.

114. Moir's disclosure is itself consistent with a roll-back procedure that, following conventional cache coherence protocol (e.g., MESI), an interfering memory access is never permitted to access a speculative dirty cache line. Whether the interfering access is a load or a store, the snoop process will evict the speculative dirty line with no writeback, and the interfering access will then be permitted to complete with no further interaction with the L1 cache in question.

115. Martínez more explicitly teaches an interaction of the conflicting access operation (*e.g.*, of *instructions in said third group*) with speculatively executed *first* or *second groups of instructions* such that the interfering access attempt stalls and such that the adjustments to memory state associated with rollback (cache line eviction, clearing the load-mark bits) take effect ***before** allowing* the conflicting operation to successfully read or write data for the addressable memory location(s) associated with *the cache line*. A POSITA would have been motivated to perform rollback *before allowing* the conflict triggering instruction to *access said cache line* (as taught by Martínez) in the context of Moir's system because Moir's discard and re-execution (rollback) of instructions for the interfered with threads or critical

sections would be performed more promptly without the wasted execution cycles for additional instructions to reach commit points of the interfered-with threads or critical sections and only then roll back, and roll back needs to happen to discard dirty speculative results and the correct roll back state reinstated to allow the interfering access to proceed independently. Benefits of more promptly rolling back before allowing access therefore include fewer wasted compute cycles and the ability to more quickly retry/reissue instructions once rolled back.

116. Finally, a POSITA would have understood that where accesses are non-interfering or non-conflicting (*e.g.*, because they do not exhibit a dependency hazard or because they are not executed with transactional memory or speculative execution controls that result in load- or store-marking)—whether in Moir, Martínez or indeed, Moir in view of Martínez—access is not impeded. Accordingly, Moir in view of Martínez *otherwise grants said access*.

117. Thus, Moir in view of Martínez renders obvious *wherein said processing comprises rolling back* (roll back and restart as taught by Martínez) *said first group of instructions provided said first bit is set* (the first word-granular load marking bit associated with a first processor's cache line as per [1.1]) *and rolling back* (roll back and restart as taught by Martínez) *said second group of instructions provided said second bit is set* (the second word-granular load marking bit associated with the same processor's same cache line as per [1.2]) *before allowing an*

instruction in said third group to access said cache line (as taught by Martínez)*, and otherwise* (in the non-interfering case) *granting said access*.

118. Finally, the forgoing analysis treats the claim language "wherein said processing comprises rolling back said first group…and rolling back said second group…" as finding antecedent basis in limitation [1.4] ("processing said first group and said second group of instructions according to [cache line marking]"). This interpretation is consistent with the scope of originally presented claim 35 for which the Examiner indicated allowability and which forms the basis for the re-written claim that ultimately issued as claim 1. *See* Ex.1002, 241-242 (dependent claim 35 reciting "said handling comprising rolling back said first group and said second group…" referring to "handling said first group and said second group of instructions…" in claim 34 which, in turn, depended from claim 33). As explained in Section VII.B above, application claims 33, 34 and 35 were combined as the claim that issued as claim 1. Ex.1002, 32.

119. Thus, the foregoing analysis explains how the Moir-Martínez combination's handling of first group and second group of instructions provides roll back(s). Nonetheless, because roll back is triggered by a conflicting instruction of the *third group* executed by another processor that seeks to *access said cache line*, the Moir-Martínez grounds embrace "*said processing comprising rolling back said first group* [if first bit set] *and rolling back second group* [if second bit set]," whether

(a) the processor/agent executing the third group initiates or triggers the roll back(s); (b) the processor executing the first group and second group performs respective roll back(s); or both (a) and (b).

[1.3B-alternate] Note Regarding Possible Alternative Constructions

120. The forgoing analysis treats the claim language "*wherein* ***said processing*** *comprises rolling back said first group…and rolling back said second group…*" as finding antecedent basis in limitation [1.4] ("***processing*** *said first group and said second group of instructions according to* [cache line marking]"). This interpretation (in which ***said processing*** is performed by ***said processor***, rather than by "*an agent other that said processor*") is consistent with the scope of originally presented claim 35 for which the Examiner indicated allowability and which forms the basis for the re-written claim that ultimately issued as claim 1. *See* Ex.1002, 241-242 (dependent claim 35 reciting "said handling comprising rolling back said first group and said second group…" referring to "handling said first group and said second group of instructions…" in claim 34 which, in turn, depended from claim 33). As explained in Section V.A.2 above, application claims 33, 34 and 35 were combined as the claim that issued as claim 1. Ex.1002, 32.

121. Accordingly, the foregoing analysis explains how the Moir-Martínez combination's handling of first and second groups of instructions provides roll

back(s). Nonetheless, this declaration has explained that roll back is triggered by a conflicting instruction of the *third group* executed by another processor that seeks to *access said cache line*, the Moir-Martínez grounds embrace "*said processing comprising rolling back said first group* [if first bit set] *and rolling back second group* [if second bit set]," whether (a) the processor/agent executing the third group initiates or triggers the roll back(s); (b) the processor executing the first group and second group performs respective roll back(s); or both (a) and (b)..

3. Claim 2

[2.1] The method of claim 1 wherein said processing further comprises: determining a state of said cache line, wherein said state is specified according to a cache coherency protocol comprising at least a modified state, a shared state, and an invalid state; and

122. As discussed above, the combination of Moir and Martínez renders obvious *the method of claim 1*. Additionally, the Moir-Martínez combination renders obvious limitation [2.1] because each reference describes or incorporates cache coherence mechanisms that POSITA would have understood to *determine state of a cache line* via snooping techniques and using *states specified according* to then-conventional MESI/MOESI (*modified*, [owned], exclusive, *shared*, *invalid*) *cache coherency protocols*.

123. For example, Moir explains that its "L2 cache 120 operates in concert with L1 data cache 115…in processor 101…and with L1 data cache 117…in

processor 102" and employs "a coherency mechanism 122" such as described in a co-pending application (published as Ex.1011) naming two of his co-inventors. Ex.1005, 6:45-59 (specifically citing Ex.1011 for supporting directory structure). A POSITA would have understood the referenced coherency mechanism to involve *cache coherency protocols* amongst the various caches and, indeed, the referenced co-pending application indicates that its "**cache coherency protocol** includes all of the usual state transitions between the following MOESI **states**: **modified** (M), owned (O), exclusive (E), **shared** (S) and **invalid** (I) states."; *see also* Ex.1011, [0139].

124. Further, Moir explains that its "coherency mechanism 122 maintains 'copyback information' 121 for each cache line. This copyback information 121 facilitates sending a cache line from L2 cache 120 to a requesting processor in cases where a cache line must be sent to another processor." Ex.1005, 6:55-59. In this way, separate processors (*i.e.*, a processor and an agent other than the processor) are capable of accessing the same cache lines, and a coherency mechanism is used to prevent stale cache lines from being accessed, as well as determining "whether any interfering memory references take place during transactional execution," as is discussed throughout Moir. Ex.1005, 6:60-65. A POSITA would understand that the combination of speculative transactional execution and a conventional cache coherence protocol would require changes to be made to the interfered-with cache

line. As discussed earlier, interfering accesses are of three types, loads and stores, and explicit cache coherence messages such as invalidate. If an external load attempts to access a cache line for which no speculative bits are set, then that external load is handled in exactly the same way the cache coherence protocol prescribes: an exclusive line changes to shared; a shared line remains shared; a modified line is forced to writeback and change to shared. If that external load hits a line with speculative bits set, then rollback is initiated and part of the rollback invalidates that cache line. Interfering stores are handled in similar fashion, via the same combination of conventional cache coherence protocol and transaction rollbacks.

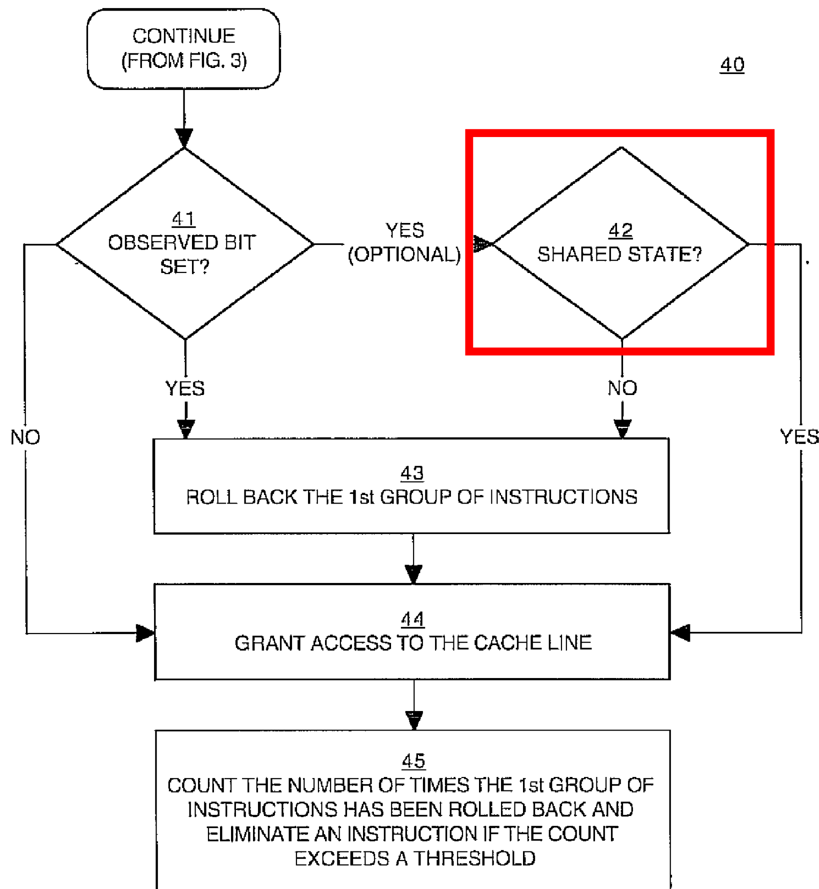
125. Thus, Martínez and/or Moir disclose (or render obvious to a POSITA) both the use of *states* that are *specified according to a cache coherency protocol comprising at least a modified state, a shared state, and an invalid state* and the *determining* of *a state of said cache line* using such protocols.

[2.2] rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access provided said cache line is in other than said shared state, and otherwise granting said access only when said cache line is in said shared state.

126. The combination of Moir and Martínez renders obvious limitation [2.2] because, as discussed in limitation [1.3B], Moir in view of Martínez renders obvious rolling back respective first and second groups of instructions under conditions where respective first and second bits are set before allowing the referenced access.

And consistent with the discussion of limitation [1.3B] and [1.3B-alternative], a POSITA would have further understood that, in the course of rolling back any affected instruction groups, the cache line is updated, and memory access is *then granted* because interfering memory access necessitates the rollback to be completed before granting access to the cache line to ensure that valid data is being accessed. Accordingly, Moir in view of Martínez renders obvious *rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access.*

127. As reflected in Figure 4 of the ’395 Patent, below, a POSITA would have understood that in line with the claim language, a first group of instructions and/or a second group of instructions are rolled back provided the cache line is in a state other than a shared state, such as a modified state, while no rollback would occur if the cache line is in a shared state. Ex.1001, 7:53-62. MESI dictates a transition from shared to modified or exclusive on an access that could interfere. Thus, a POSITA would have understood that, in cases where *said cache line is in said shared state,* the “*otherwise*” condition of *granting said access* simply based on *said shared state* applies, and in cases where *said cache line is in other than said shared state*, the previously explained dependence of roll back on observed bits applies.



Ex.1001, Fig. 4 (annotated)

128. Regarding the applied Moir-Martínez combination and execution sequence dependence on whether *said cache line is in said shared state*, POSITA would have understood the claimed dependence to follow from a conventional understanding of the MESI coherency protocol and its *shared* (S) state. The MESI cache coherence protocol is often explained with reference to a graph showing the states, the transitions between the states, and the events causing those transitions.

For instance, the following figure is from Modern Processor Design, and shows the four states M (Modified), S (shared), I (Invalid), and E (Exclusive). Ex.1019, 571.

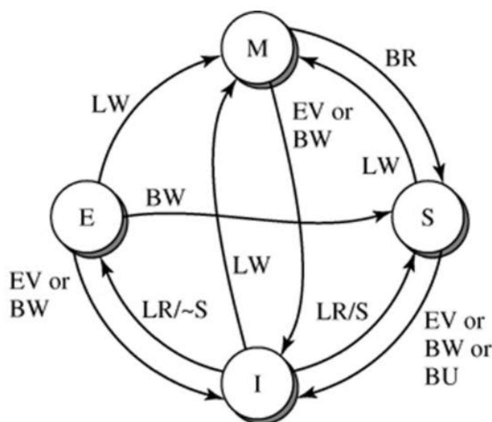
MESI Cache Coherence Protocol

State Table (s' = next state)

- view from local cache to local events (LR,LW,EV) and bus events (BR,BW,BU)

Current State s	Local Read (LR)	Local Write (LW)	Local Eviction (EV)	Bus Read (BR)	Bus Write (BW)	Bus Upgrade (BU)
Invalid (I)	Issue bus read; if no sharers s'=E else s'=S	Issue bus write s'=M	s'=I	Do nothing	Do nothing	Do nothing
Shared (S)	Do nothing	Issue bus upgrade s'=M	s'=I	Respond shared	s'=I	s'=I
Exclusive (E)	Do nothing	s'=M	s'=I	Respond shared s'=S	s'=I	Error
Modified (M)	Do nothing	Do nothing	Write data back; s'=I	Respond dirty; Write data back s'=S	Respond dirty; Write data back s'=I	Error

State Diagram



In response to local and bus events the coherence controller may need to change the local coherence state of a line, and may also need to fetch or supply the cache line data.

Ex.1019, 571.

129. Any given cache line must be in one, and only one, of these four states. Lines marked as Shared indicate that (a) the line itself is a clean copy of the corresponding memory locations, and (b) other caches may also have an identical unmodified copy. As long as accesses to this cache line are loads, either local loads or remote loads from another processor, these loads do not change any values within the line, and the cache line remains in Shared state. This remains true even if there are load-marked words in that cache line, because the Shared state indicates that whatever loads caused those load-mark bits to become set, themselves got correct (consistent with main memory) data. This is shown in the MESI State Table above, for the Shared (S) line: local reads simply hit and return the cached data, while "Bus Read" (an access from a different processor) sees a "Shared" response, which tells that processor it can freely use its own copy of that cache line. For an implementation that tracks cache line state in accord with MESI/MOESI coherency protocols (such as Moir and/or Martínez), states other than the shared state (S), *e.g,* **modified**/dirty (M) or **owned** (O), if provided, or **exclusive (E)** state(s), would have been the appropriate states for which access conflicts would be managed in the manner discussed in limitation [1.3B] and reprised in the language of limitation [2.2].

130. Thus, Moir and Martínez render obvious *rolling back said first group of instructions provided said first bit is set and rolling back said second group of instructions provided said second bit is set, and then granting said access provided*

said cache line is in other than said shared state, (rolling back when an interfering load access takes place as described relative to [1.3B] and [1.3B-alternative], as reflected through, e.g., a Modified (M) state for the cache line) *and otherwise granting said access only when said cache line is in said shared state* (granting access in the Shared (S) state).

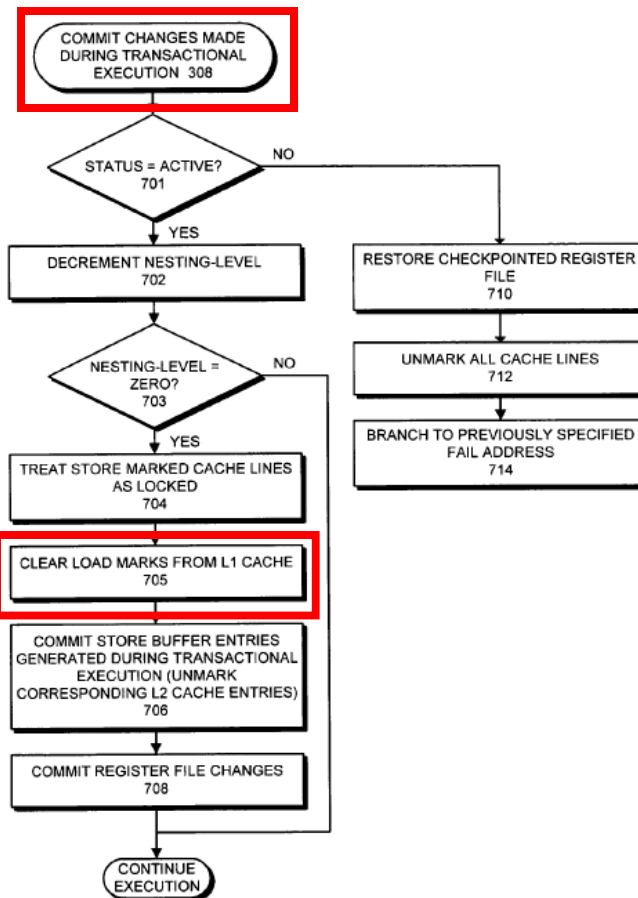
4. Claim 5

[5.1] The method of claim 1 further comprising clearing said first bit when said execution of said first group of instructions is ended, wherein said second bit remains set until execution of said second group of instructions is ended.

131. As discussed in Section XI.A.2 above, the combination of Moir and Martínez renders obvious *the method of claim 1*. Additionally, Moir renders obvious claim 5 because Moir explains that "the system clears load-marks from [the] L1 data cache" after a transaction execution completes. Ex.1005, 9:49-52, 10:3-4, 10:50-53, 10:60-64, Figs. 7, 8.

132. As discussed above in limitation [1.4], Moir explains that "[d]uring this transactional execution, the system continually monitors data references made by other threads, and determines if an interfering data access…takes place during transactional execution. **If not, the system atomically commits** all changes made during transactional execution (step 308)[.]" Ex.1005, 8:1-8, *see also* Fig. 3. As part of the process of committing changes made during the transaction execution (step

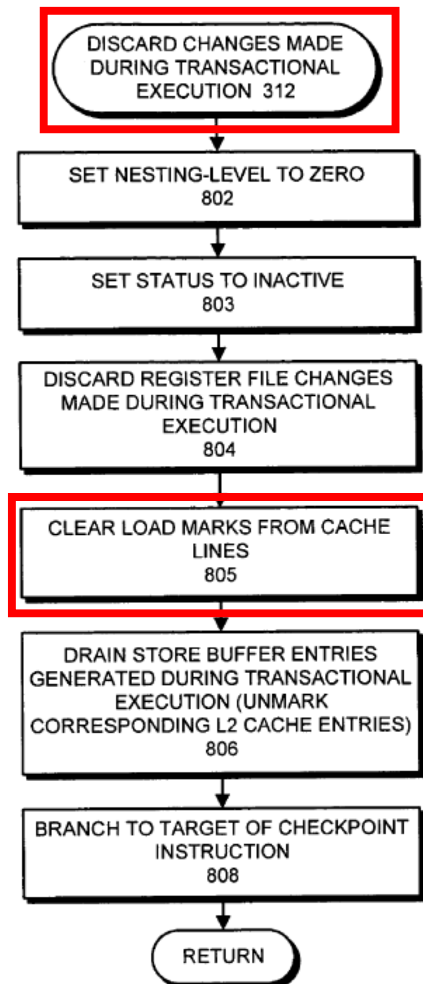
308 in Fig. 3), "the system clears load-marks from [the] L1 data cache" (step 705 in Fig. 7). Ex.1005, 9:52-53, 10:3-4.



Ex.1005, Fig. 7 (annotated)

133. Similarly, as reflected in Figure 8, below, Moir discloses that "[t]he system also clears load-marks from cache lines in [the] L1 data cache" when changes are discarded (i.e., rolled back). Ex.1005, 10:60-64. This occurs if there is an interfering data access (or other type of failure), such that "the system discards changes made during the transactional execution (step 312)." Ex.1005, 8:10-12. As

part of the process of discarding changes made during the transaction execution (step 312 in Fig. 3), "the system also clears load-marks from [the] L1 data cache" (step 805 in Fig. 7). Ex.1005, 10:53-54, 10:60-64.



Ex.1005, Fig. 8 (annotated)

134. Moreover, Moir clears load-marks from the L1 data cache that are used to mark respective word-granular bits associated with a given cache line. As discussed above in limitations [1.1] and [1.2], Moir allows multiple critical sections

to separately mark addressable words within a cache line. Accordingly, a POSITA would have understood that, following either the successful or unsuccessful completion of a transactional execution for a group of instructions, the load-marks for the corresponding words within a cache line would be cleared, *clearing said first bit when said execution of said first group of instructions is ended*, because the processor would have proceeded through steps following completion that include clearing load-marks from the L1 cache. Ex.1005, 9:49-52, 10:3-4, 10:50-53, 10:60-64, Figs. 7, 8. However, for other groups of instructions for which the transactional execution has not yet completed, *said second bit remains set until execution of said second group of instructions is ended* because changes have neither been committed nor discarded, and thus the processor would not yet have cleared the load-marks corresponding to those instructions.

135. Thus, Moir's disclosure of clearing load-marks from the L1 data cache after a transactional execution completes renders obvious *clearing said first bit when said execution of said first group of instructions is ended, wherein said second bit remains set until execution of said second group of instructions is ended*.

5. Claim 7

136. Claim 7 is a system claim and is obvious over Moir in view of Martínez for the reasons detailed in corresponding limitations [1.0], [1.1], [1.2], [1.4], [1.3A] and [1.3B] of corresponding method claim 1. The limitations "*a processor*" and

"*cache memory for use by said processor*" are disclosed by Moir as explained in [1.0].

137. Moreover, the additionally recited "*memory unit coupled to said processor and having stored therein instructions, said instructions comprising: instructions to* [perform each of the steps recited in the corresponding elements of claim 1]" is disclosed by Moir in the form of an L1 instruction cache ("*a memory under coupled to said processor*"). Ex.1005, 6:35-36 (explaining that "[p]rocessor 101 also includes an L1 instruction cache (not shown [in Fig. 1])"). POSITA would have understood that the L1 instruction cache stores instructions executed by Moir's processor (*i.e.*, "*instructions to execute…set…execute…set…execute…roll back* [*and*] *process*…" as recited in the respective limitations of claim 7).[11] Because each of those limitations of claim 7 recites a function that corresponds to limitation [1.1], [1.2], [1.4], [1.3A] or [1.3B] of corresponding method claim 1, and because POSITA

[11] Instruction caches are similar to data caches, in that they provide fast access to instructions fetched by the code being executed. Instruction fetches are generally effective because (a) most code exhibits substantial reference locality, thus taking advantage of a multiple-words-per-line organization of the cache, and (b) most code spends much of its execution time in loops, which amortizes the initial cache fill overhead.

would have understood Moir's processor to perform those functions based on instructions executed from its instruction cache, claim 7 is obvious over Moir in view of Martínez for analogous reasons to those detailed above with respect to claim 1. Moreover, Martínez' safe and speculative threads also include (and render obvious in the combination) the recited "*instructions to…*" perform recited functions.

6. Claim 8

138. Claim 8 depends from claim 7 and is obvious over Moir in view of Martínez for the reasons detailed above, including those detailed for corresponding limitations [2.1] and [2.2] of corresponding method claim 2. Each of these limitations of claim 8 recites a function that corresponds to a limitation of corresponding method claim 2 that POSITA would have understood Moir's processor to perform based on execution of instructions executed from its L1 instruction cache.

7. Claim 11

139. Claim 11 depends from claim 7 and is obvious over Moir in view of Martínez for the reasons detailed above, including those detailed for limitation [5.1] of corresponding method claim 5. Claim 11 recites the same *clearing* function as method claim 5, and POSITA would have understood Moir's processor to "*clear*"

the recited bit associated with the cache line as explained relative to [5.1] based on execution of instructions executed from L1 instruction cache of Moir's processor.

XII. AVAILABILITY FOR CROSS-EXAMINATION

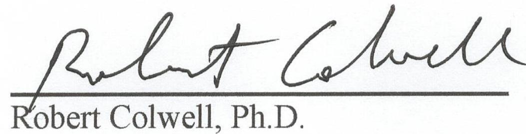
140. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross examination in the case and that cross examination will take place within the United States. If cross examination is required of me, I will appear for cross examination within the United States during the time allotted for cross examination.

XIII. CONCLUSION

141. I hereby declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct, and that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true. I understand that willful false statements are punishable by fine or imprisonment or both. *See* 18 U.S.C. § 1001.

Date: 3/5/25

Respectfully submitted,



Robert Colwell, Ph.D.