

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

BMW of North America LLC,

Petitioner,

v.

Foras Technologies Ltd.,

Patent Owner.

IPR2024-01347
U.S. Patent 7,624,302

DECLARATION OF DR. MICHAEL C. BROGIOLI

TABLE OF CONTENTS

I.	Introduction.....	8
II.	Qualifications.....	8
III.	Materials Considered	13
IV.	Relevant Legal Standards	16
A.	Validity	17
B.	Level of Ordinary Skill	20
C.	Claim Construction.....	21
1.	“Detecting of Loss of Lockstep (LOL)” and “Detect Loss of Lockstep (LOL)”	22
2.	Means-Plus-Function Terms	24
3.	All Other Claim Terms	26
V.	The State of Art	26
A.	Using Lockstep Processors in a Multiprocessor System Was Known	26
B.	Using a Boot Processor to Initialize a Multiprocessor System Was Known	29
C.	Fault Recovery to a Spare Processor During System Runtime in a Multiprocessor System Was Known	33
D.	Using Firmware to Switch a Failing Processor to a Spare Processor During System Runtime Was Known	36
VI.	Overview of the '302 Patent.....	37
VII.	Overview of the Prior Art.....	41
A.	Overview of Fox (Ex. 1005)	41

B.	Overview of Safford (Ex. 1006).....	45
C.	Overview of Arai (Ex. 1007).....	48
D.	Overview of Landry (Ex. 1011).....	51
E.	Overview of Bigbee (Ex. 1012)	52
F.	Fox, Safford, Arai, Landry, and Bigbee are Analogous Art to the '302 Patent.....	56
VIII.	Ground of Unpatentability.....	57
IX.	Ground 1: Fox in view of Safford and Arai Renders Obvious Claim 1	58
A.	Fox in view of Safford and Arai.....	58
B.	A POSITA Would Have Found It Obvious to Combine Fox, Safford, and Arai	60
1.	A POSITA would have found it obvious to implement Fox's processors as lockstep processors with error detection logics that detect loss of lockstep, as taught in Safford, to improve system reliability and availability.	60
2.	A POSITA would have understood that Fox's multiprocessor system would have one of its processors designated as the boot processor and replaced the failing boot processor with the spare processor.	66
3.	A POSITA would have found it obvious to incorporate Arai's processor-switching implementation details in the Fox-Safford system to minimize system interruption during switching.....	70
C.	Independent Claim 1	75
1.	[1preamble]: "A method comprising:"	75
2.	[1a]: "detecting loss of lockstep (LOL) for a processor in a multi-processor system;"	75

3.	[1b]: “determining that the processor for which said LOL is detected is assigned a role of boot processor;”	78
4.	[1c]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said boot processor for which said LOL is detected, to a different value, which causes the system’s operating system to not recognize the processor for which said LOL is detected as the boot processor; and”	79
5.	[1d]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to a spare processor without shutting down the system’s operating system.”	82
X.	Ground 2: Fox in View of Safford, Landry, and Arai Renders Obvious	
	Claim 1	84
A.	Fox in View of Safford, Landry, and Arai	84
B.	A POSITA Would Have Found It Obvious to Combine Fox, Safford, Landry, and Arai	84
1.	A POSITA would have found it obvious to identify Fox’s boot processor as logical CPU0 as taught in Landry.....	85
2.	A POSITA would have found it obvious to switch logical CPU0 (boot processor) to the spare processor in response to detecting loss of lockstep in the logical CPU0, as taught in Safford.....	86
C.	Independent Claim 1	87
1.	[1preamble]: “A method comprising:”	87
2.	[1a]: “detecting loss of lockstep (LOL) for a processor in a multi-processor system;”	87
3.	[1b]: “determining that the processor for which said LOL is detected is assigned a role of boot processor;”	88

4.	[1c]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said boot processor for which said LOL is detected, to a different value, which causes the system’s operating system to not recognize the processor for which said LOL is detected as the boot processor; and”	90
5.	[1d]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to a spare processor without shutting down the system’s operating system.”	90
XI.	Fox in View of Safford and Arai (Ground 1) or Fox in View of Safford, Landry, and Arai (Ground 2) Renders Obvious Claims 2-25	91
A.	Independent Claim 13	91
1.	[13preamble]: “A method comprising:”	91
2.	[13a]: “system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system;”	91
3.	[13b]: “determining one of said lockstep pair of processors that is not the cause of said LOL;”	94
4.	[13c]: “copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to a spare processor;”	97
5.	[13d]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said lockstep pair of processors that are assigned the role of boot processor, to a different value, which causes the system’s operating system to not recognize the lock step pair of processors for which said LOL is detected as the boot processor; and”	99

6.	[13e]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to said spare processor.”	99
B.	Independent Claim 21	99
1.	[21preamble]: “A system comprising:”	100
2.	[21a]: “a plurality of processor modules that each include a master processor and a slave processor that operate in lockstep;”	100
3.	[21b]: “an operating system;”	105
4.	[21c]: “error detection logic operable to detect loss of lockstep (LOL) for at least one of said processor modules; and”	106
5.	[21d]: “system firmware operable, responsive to the detection of LOL for a first of said processor modules, to determine whether said first processor module is assigned a role of system boot processor;”	108
6.	[21e]: “wherein if determined that said first processor module is assigned the role of system boot processor, said system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system.”	109
C.	Dependent Claims 2, 14, 23	112
1.	Claims 2 and 14: “wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor.”	112
2.	Claim 23: “wherein the another processor module is held in idle until said system firmware causes said operating system to recognize said another processor module as said system boot processor.”	114

D.	Dependent Claims 3 and 15	115
E.	Dependent Claims 4 and 16	117
F.	Dependent Claim 5: “wherein the spare processor is a processor module that includes a master and slave processors that operate in lockstep.”	118
G.	Dependent Claims 6 and 25	119
1.	Claim 6: “wherein said determining that the processor for which said LOL is detected is assigned the role of boot processor comprises: accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of boot processor.”	119
2.	Claim 25: “non-volatile data storage storing information that assigns the role of system boot processor to one of said plurality of processor modules.”	122
H.	Dependent Claims 7, 8, and 22	123
1.	Claim 7: “determining the processor that is assigned the role of spare processor.”	123
2.	Claim 8: “wherein said determining the processor that is assigned the role of spare processor comprises: accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of spare processor.”	126
3.	Claim 22: “wherein the another processor module is a processor module assigned a role of spare processor.”	127
I.	Dependent Claim 9, 10, and 18-20.....	127
1.	Claims 9 and 18: “wherein said determining step is performed by system firmware.”	127

2.	Claims 10 and 20: “wherein said switching step is performed by system firmware.”	127
3.	Claim 19: “wherein said copying step is performed by system firmware.”	128
XII.	Fox in View of Safford, Arai, and Bigbee (Ground 3) or Fox in View of Safford, Landry, Arai, and Bigbee (Ground 4) Renders Obvious Claims 26 and 27	128
A.	Overview of the Combinations	129
B.	A POSITA Would Have Found It Obvious to Configure Fox’s System Combined with Safford and Arai or Safford, Landry, and Arai, with Itanium Family Processors and Firmware as Taught in Bigbee to Improve System Performance and Capacity	130
C.	Independent Claim 26	132
1.	[26preamble]: “A system comprising:”	132
2.	[26a]: “system firmware means for detecting loss of lockstep (LOL) for a processor in a multi-processor system;”	132
3.	[26b]: “system firmware means for determining whether the processor for which said LOL is detected is assigned the role of boot processor; and”	135
4.	[26c]: “system firmware means for switching the role of boot processor to a spare processor without shutting down the system’s operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor.”	136
D.	Dependent Claim 27: “wherein said processor in said multi-processor system is an Itanium Processor Family (IPF) processor.”	138
XIII.	Conclusion	138

I. Introduction

1. I, Michael C. Brogioli, submit this declaration to state my opinions on the matters described below.

2. I have been retained on behalf of Petitioner BMW of North America LLC (“BMW”) as an independent expert for the above-identified *inter partes* review proceeding involving U.S. Patent No. 7,624,302 (“the ’302 patent”) (Ex. 1001). Although I am being compensated for my time in connection with this IPR at my standard hourly consulting rate and reimbursed for reasonable out-of-pocket expenses, no part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I have been asked to provide my technical review, analysis, insights, and opinions regarding the ’302 Patent, the prior art references that form the basis for the invalidity grounds set forth in the Petition for *Inter Partes* Review of the ’302 Patent, and any other prior art publications cited in this declaration.

II. Qualifications

4. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience summarized below. My *curriculum vitae* (Ex. 1003) records my education, experience, and publications in greater detail.

5. I am currently an Adjunct Professor of Electrical and Computer engineering at Rice University in Houston, Texas, and Managing Director of Polymathic Consulting in Austin, Texas. I received my Bachelor of Electrical Engineering from Rensselaer Polytechnic Institute in 1999, my Master of Science in Electrical and Computer Engineering from Rice University in 2003, and my Doctorate of Electrical and Computer Engineering from Rice University in 2007.

6. While at Rice University, I developed various computer architecture designs for embedded and high-performance systems, including various microprocessor technologies. For example, from 1999 to 2003, I worked in the area of low-power computing, specifically focusing on dynamic power management and performance of configurable memory systems and related microprocessor technology. From 2002 to 2004, I developed Spinach, a computer architecture design and modeling toolset, which models system components common to all programmable computing environments, including memory systems, microprocessors, hardware accelerators, interconnects, etc. From 2004 to 2009, I developed Spinach DSP-FPGA, a modular and composable simulator design infrastructure for programmable and reconfigurable embedded SOC architectures specifically targeting mobile, low-power, and embedded and portable computing devices. From 2005 to 2009, I developed and published a retargetable

compiler infrastructure and hardware design exploration toolkit for systems related to embedded computing technologies, which is also used in the design space exploration of memory systems and related technologies. Many of these tools have been used at United States universities in the area of electrical and computer engineering research.

7. In 1997-1998, I was a Hardware Test Engineer at Stratus Computer in Massachusetts, debugging locked step CPU operation and memory management issues in Stratus' fault-tolerant UNIX systems and Hewlett Packard PA-8000 series CPU modules.

8. In 1999, I was a hardware, firmware, and software developer at Vicarious Visions in New York, developing third-party titles for Nintendo's handheld consoles, in addition to various hardware and firmware interfaces and software optimizations related to the limited memory I/O and size.

9. In 2000, I was a Compiler Engineer in the Microprocessor Research Labs of Intel Corporation in California, implementing speculative multi-threading support in Intel's IA-64 compiler specifically for Itanium[®] family processors, and developing new program analysis and back end code generation phases to support speculatively launching threads at runtime.

10. In 2000-2003, I was an Instructor at the Department of Electrical and Computer Engineering Rice University in Texas, teaching graduate level courses relating to Computer Architecture and Circuit Design, including topics related to fault-tolerant computer architecture design.

11. In 2004, I co-authored “Spinach: A Liberty–Based Simulator For Programmable Network Interface Architectures,” published in the ACM SIGPLAN/SIGBED Conference on Languages Compilers and Tools for Embedded Systems, Washington, D.C., 2004. Spinach was a tool for multiprocessor system architecture design and was made available by open source in 2004.

12. From June 2006 to August 2007, I worked as the Technical Co-Founder of Method Seven LLC in Boston, MA, working with high-performance software and hardware systems architecture.

13. From 2008 to 2009, I was Senior Engineer working in high-performance compiler designs and next-generation microprocessors and architectures at Freescale Semiconductor in Austin, TX.

14. From November 2009 to October 2011, I was Chief Architect, Senior Member Technical Staff, at Freescale Semiconductor in Austin, TX (formerly Motorola), responsible for management of technology, engineering roadmaps, design lead on software infrastructure, and next-generation microprocessor

architectures for embedded computing. During my tenure at Freescale Semiconductor, I was in charge of system developer tools for processor design, both in the hardware, firmware, and software spaces.

15. I have held the position of Adjunct Professor at Rice University since 2009 and the position of Managing Director at Polymathic Consulting since 2011. At Rice University, I instruct graduate-level curriculum in the areas of computer architecture, hardware and software systems, with a focus on computer architecture. I also advise on university research and various design initiatives. At Polymathic Consulting, I work with a range of technologists from early-stage start-ups to Fortune 500 companies on similar technologies.

16. I am recognized as an expert in the field of computer architecture, computer hardware, computer firmware, and computer software systems as they relate to the subject matter at hand. I am a member of the Institute of Electrical and Electronics Engineers (IEEE) and have been a Program Committee member for the IEEE and ACM Design Automation Conference since 2011, and have held the role of Program Chair of Design Automation Conference in the area of Embedded Computing.

17. Over the past 20 plus years, I have authored numerous peer-reviewed publications, as well as engineering books in the area of computer hardware and

software design. Many of these incorporate technologies specific to the subject matter at hand, including microprocessor technology and related aspects of computing. These publications are disclosed in my *curriculum vitae* (Ex. 1003). I have previously served as an engineering consultant and testifying witness on matters related to, and including, microprocessors, chip technology, and firmware design and optimization. My *curriculum vitae* contains more information on my background and experience, as well as the cases in which I have served as an expert witness over at least the past four years.

III. Materials Considered

18. In forming my opinions, I have reviewed the following documents and any other documents cited in this declaration:

Exhibit	Description
Ex. 1001	U.S. Patent No. 7,624,302 B2 to Michaelis et al. (“the ’302 Patent”)
Ex. 1004	Prosecution History of the ’302 Patent
Ex. 1005	US 2005/0172164 A1 (“Fox”), filed January 21, 2004, published August 4, 2005
Ex. 1006	US 2004/0006722 A1 (“Safford”), filed July 3, 2002, published January 8, 2004
Ex. 1007	US 2006/0036889 A1 (“Arai”), filed August 16, 2004, published February 16, 2006

Exhibit	Description
Ex. 1008	IA-32 Intel® Architecture Software Developer’s Manual, Vol. 3: System Programming Guide (“IA-32”), publicly available at least as of February 14, 2002
Ex. 1009	Declaration of Nathaniel E. Frank-White regarding Internet Archive copy of “IA-32 Intel® Architecture Software Developer’s Manual, Vol. 3: System Programming Guide,” dated July 5, 2024
Ex. 1010	U.S. Patent No. 7,296,181 B2 to Safford, filed April 6, 2004, granted November 13, 2007 (“Safford-181”)
Ex. 1011	U.S. Patent No. 5,408,647 (“Landry”), filed October 2, 1992, granted April 18, 1995
Ex. 1012	US 2003/0126498 A1 (“Bigbee”), filed January 2, 2002, published July 3, 2003
Ex. 1013	U.S. Patent No. 5,491,788 (“Cepulis”), filed September 10, 1993, granted February 13, 1996
Ex. 1014	US 2003/0233492 A1 (“Schelling”), filed June 13, 2002, published December 18, 2003
Ex. 1015	U.S. Patent No. 6,189,112 B1 (“Slegel”), filed April 30, 1998, granted February 13, 2001
Ex. 1016	U.S. Patent No. 7,426,657 B2 (“Zorek”), filed July 9, 2004, granted September 16, 2008
Ex. 1017	WO 2001080007 A2 (“Suffin”), filed April 12, 2001, published October 25, 2001
Ex. 1018	U.S. Patent No. 6,108,781 (“Jayakumar”), filed February 23, 1999, granted August 22, 2000
Ex. 1019	US 2004/0221193 A1 (“Armstrong”), filed April 17, 2003, published November 4, 2004

Exhibit	Description
Ex. 1020	US 2002/0144177 A1 (“Kondo”), filed January 31, 2002, published October 3, 2002
Ex. 1021	US 2004/0123201 A1 (“Nguyen”), filed December 19, 2002, published June 24, 2004
Ex. 1022	U.S. Patent No. 5,915,082 (“Marshall”), filed June 7, 1996, granted June 22, 1999
Ex. 1023	U.S. Patent No. 6,081,890 (“Datta”), filed November 30, 1998, granted June 27, 2000
Ex. 1024	U.S. Patent No. 5,784,599 (“Elkhoury”), filed December 15, 1995, granted July 21, 1998
Ex. 1025	US 2005/0086667A1 (“Jin”), filed September 30, 2003, published April 21, 2005
Ex. 1026	L. Spainhower and T.A. Gregg, <i>IBM S/390 Parallel Enterprise Server G5 fault tolerance: A historical perspective</i> , IBM Journal of Research and Development, vol. 43, no. 5/6, pp. 863-873, September/November 1999, doi: 10.1147/rd.435.0863.
Ex. 1027	M. Mueller et al., <i>RAS strategy for IBM S/390 G5 and G6</i> , IBM Journal of Research and Development, vol. 43, no. 5/6, pp. 875-888, September/November 1999, doi: 10.1147/rd.435.0875.
Ex. 1028	MultiProcessor Specification (“MP Specification”), Version 1.4, May 1997
Ex. 1029	Advanced Configuration and Power Interface Specification, Revision 2.0, July 27, 2000 (“ACPI 2.0”)
Ex. 1030	Intel® Itanium® Architecture Software Developer’s Manual, Volume 2: System Architecture, Revision 2.1, October 2002

Exhibit	Description
Ex. 1031	Prosecution History of <i>Ex Parte</i> Reexamination of the '302 Patent, Control No. 90/019,243
Ex. 1032	Prosecution History of <i>Ex Parte</i> Reexamination of the '781 Patent, Control No. 90/019,244
Ex. 1033	U.S. Patent No. 7,502,958 B2 to Michaelis et al. ("the '958 Patent"), filed October 25, 2004, granted March 10, 2009
Ex. 1034	U.S. Patent No. 7,627,781 B2 to Michaelis et al. ("the '781 Patent")

IV. Relevant Legal Standards

19. In preparing this declaration and forming my opinions, I am relying on certain legal principles that counsel explained to me. My understanding of these concepts is summarized below.

20. I have been asked to provide my opinions regarding whether claims 1-27 ("the Challenged Claims") of the '302 Patent are unpatentable, because they 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 advised and understand that prior art includes all analogous art, and that two separate tests define the scope of analogous prior art: (1) whether the art is from the same field of endeavor, regardless of the problem addressed and, (2) if the reference is not within the field

of the inventor's endeavor, whether the reference still is reasonably pertinent to the particular problem with which the inventor is involved. It is my opinion that the Challenged Claims would have been obvious to a person having ordinary skill in the art at the time of the alleged invention.

21. In forming my opinions expressed below, in addition to my knowledge and experience based upon my work in the fields of computer architecture design, including fault-tolerant computer architecture design, I have considered the documents listed in the Table in Section III above.

22. I have been advised and understand that a dependent claim is a patent claim that refers back to another patent claim. I have been informed and understand that a dependent claim includes all of the limitations of the claim to which it refers.

A. Validity

23. I have been advised and understand that there are two ways in which prior art may render a patent claim unpatentable. First, the prior art can "anticipate" the claim. Second, the prior art can make the claim "obvious" to a person of ordinary skill in the art ("POSITA"). I understand that for an invention claimed in a patent to be patentable, it must not be anticipated and must not be obvious based on what was known before the invention was made.

24. I have been advised and understand that a patent claim is unpatentable as anticipated under 35 U.S.C. § 102 if each element of that claim is present either explicitly or inherently in a single prior art reference. I have also been advised and understand that, to be an inherent disclosure, the prior art reference must necessarily disclose the limitation. The fact that the reference might practice or contain a claimed limitation is insufficient to establish that the reference inherently teaches the limitation.

25. I have been advised and understand that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the claimed subject matter 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 advised and understand 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.

26. I have further been advised and understand 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.

27. When considering the issue of obviousness, I have been advised and understand that I should consider (1) the scope and content of the prior art; (2) the differences between the prior art and the claims at issue; (3) the level of ordinary skill in the art; and (4) evidence of secondary indicia of nonobviousness. I have been informed that secondary indicia of nonobviousness include (i) “long-felt need” for the claimed invention, (ii) commercial success attributable to the claimed invention, (iii) unexpected results of the claimed invention, and (iv) “copying” of the claimed invention by others.

B. Level of Ordinary Skill

28. In rendering the opinions set forth in this declaration, I have been asked to consider the '302 Patent's claims and the prior art through the eyes of a POSITA. I understand that a POSITA is determined by considering (1) the type of problems encountered in the art, (2) prior art solutions to those problems, (3) the rapidity with which innovations are made, (4) the sophistication of the technology, and (5) the educational level and years of experience level of those working in the pertinent field.

29. I understand that I must evaluate the '302 Patent from the perspective of a POSITA in the art. That is, the '302 Patent must be evaluated through the eyes of a POSITA at the time of the alleged invention of this patent as of its earliest priority date. It is my opinion that such a POSITA as related to the '302 Patent would have at least a bachelor's degree in computer science, computer engineering, or equivalent, and at least two years of prior work experience with computer architecture design, including experience relating to fault-tolerant computer architecture design, as of the earliest priority date of the '302 Patent—October 25, 2004. If a person did not have a bachelor's degree, additional relevant and practical experience with fault-tolerant computer systems could provide an

acceptable substitute. Likewise, further education beyond a bachelor's degree could compensate for a deficiency in practical experience.

30. Based on my educational background and experience, such as my experience with computer architecture design, including fault-tolerant computer architecture design (*see* paragraphs 4-17), I am qualified as at least a POSITA with respect to the '302 Patent. Thus, I am familiar with the knowledge of the POSITA at the time of the alleged invention. I am able to opine on how a POSITA would have understood the disclosure and claims of the '302 Patent, the disclosures of the prior art, the motivation to combine the prior art, and what combinations would have been obvious and not have been obvious to one of ordinary skill in the art.

C. Claim Construction

31. I have been informed that in an *inter partes* review proceeding, claim terms should be construed under the same standard applied in federal district court cases. Under this standard, I have been informed that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand, however, that claim terms are generally not limited by the embodiments described in the specification.

32. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I understand that the specification is highly relevant to the claim construction analysis and can be the single best guide to the meaning of a disputed term. I have been informed that the specification acts as a dictionary when it expressly defines terms used in the claims or when it defines terms by implication. I have been informed that this additional evidence is often less significant and less reliable than the claims, specification, and prosecution history.

33. I have also been informed that claims should only be construed to the extent necessary to resolve any controversy.

1. “Detecting of Loss of Lockstep (LOL)” and “Detect Loss of Lockstep (LOL)”

34. Based on my review of these materials and my personal knowledge and experience, the terms “detecting of loss of lockstep (LOL)” and “detect loss of lockstep (LOL)” should be construed to include both mismatch errors and errors that may eventually result in a mismatch error constituting a “precursor to lockstep mismatch.” Ex. 1001, 2:40-3:6, 7:25-36. Thus, as used in the claims, the terms “detecting of loss of lockstep” and “detect loss of lockstep” should be construed as

including at least one of (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error.

35. The specification provides detecting these two types of errors as examples of detection of loss of lockstep (“LOL”). *See e.g., id.*, 2:40-3:6, 7:25-36. It states that an “example of LOL is detection of the output of the paired processors not matching, which is referred to herein as a ‘lockstep mismatch.’”—(mismatch errors, definition 1). *Id.*, 2:45-47. It also states that “[b]ecause the detection of LOL ... may occur before an actual lockstep mismatch occurs, the detection of LOL ... may be referred to as a detection of a ‘precursor to lockstep mismatch’”—(errors that may eventually cause a lockstep mismatch error, definition 2). *Id.*, 7:29-33. Both lockstep mismatch and precursors to lockstep mismatch are detection of loss of lockstep in the specification of the ’302 patent because, as the specification explains, (1) mismatch errors indicate that lockstep is already lost and (2) precursors to lockstep mismatch indicate that the “error may eventually propagate to a lockstep mismatch error.” *Id.*, 7:33-36.

36. Accordingly, “detection of loss of lockstep” and “detecting loss of lockstep” should be construed as including at least one of (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error.

2. Means-Plus-Function Terms

37. I have been told that claim terms “means” “creates a rebuttable presumption” that the term is means-plus-function. Counsel for Petitioner has informed me that for means-plus-function limitations, a prior art reference or combination of references must disclose the identical function in the claim limitation and must disclose a structure that performs the function that is either identical to or the equivalent of the structure in the specification of the challenged patent that performs the claimed function. I understand that a structure is equivalent if it performs the identical function in substantially the same way to achieve substantially the same result as the claim limitation at issue.

38. My review of the prosecution history indicates that the Applicant intended the claim limitations of claim 26 reciting “means” as “means plus function” limitations despite that the limitations recite the structure of system firmware. Ex. 1004, 180-182 (“[C]laim 29 [now claim 26] has been amended to add system firmware as additional structure for the means plus function claims.”). The Reexamination Examiner concluded, and the Applicant did not dispute, that these limitations are means-plus-function limitations. Ex. 1031, 920-938. To the extent the Board finds these limitations are means-plus-function limitations, I have been told to apply the following constructions proposed by Petitioner:

Claim	Term	Function	Structure	Support
[26a]	“system firmware means for detecting loss of lockstep (LOL) for a processor in a multi-processor system”	detecting loss of lockstep (LOL) for a processor in a multi-processor system	firmware including a Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI), or equivalents thereof	Ex. 1001, 8:34-59, 11:4-60, 12:36-59, FIGS. 1-2
[26b]	“system firmware means for determining whether the processor for which said LOL is detected is assigned the role of boot processor”	determining whether the processor for which said LOL is detected is assigned the role of boot processor	firmware including a Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), Extended Firmware Interface (EFI), and a device tree, or equivalents thereof	<i>Id.</i> , 5:34-48, 9:49-60, 11:4-60, 12:36-14:3, 18:50-58, FIGS. 1-2
[26c]	“system firmware means for switching the role of boot processor to a spare processor without shutting down the system’s operating	switching the role of boot processor to a spare processor without shutting down	firmware including a Processor Abstraction Layer (PAL), System Abstraction	<i>Id.</i> , 11:4-60, 12:36-14:3, 17:12-18:43, FIGS. 1-2

	system if determined that the processor for which said LOL is detected is assigned the role of boot processor”	the system’s operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor	Layer (SAL), and Extended Firmware Interface (EFI), or equivalents thereof	
--	--	--	--	--

3. All Other Claim Terms

39. For all other claim terms, I have considered and applied their plain and ordinary meaning as they would have been understood by one skilled in the art at the time of the alleged invention and consistent with the specification.

V. The State of Art

A. Using Lockstep Processors in a Multiprocessor System Was Known

40. Before the date of the '302 Patent, a typical approach to improve fault tolerance of a computer system was to use lockstep processing. For example, as shown below, two processors 22 and 24 (or two processor units or two processor cores) are paired together to execute the same instructions on identical data in parallel; and the outputs of the two processors are compared by a comparator or a lockstep logic 26, such as an XOR gate. Safford, [0002]-[0003]; *see also* Bigbee,

[0033]; Nguyen, [0006]; Kondo, [0004]; Ex. 1001, 2:27-3:60. A discrepancy between the outputs of the pair of processors detected by logic 26 is a loss of lockstep. Safford, [0003]-[0004].

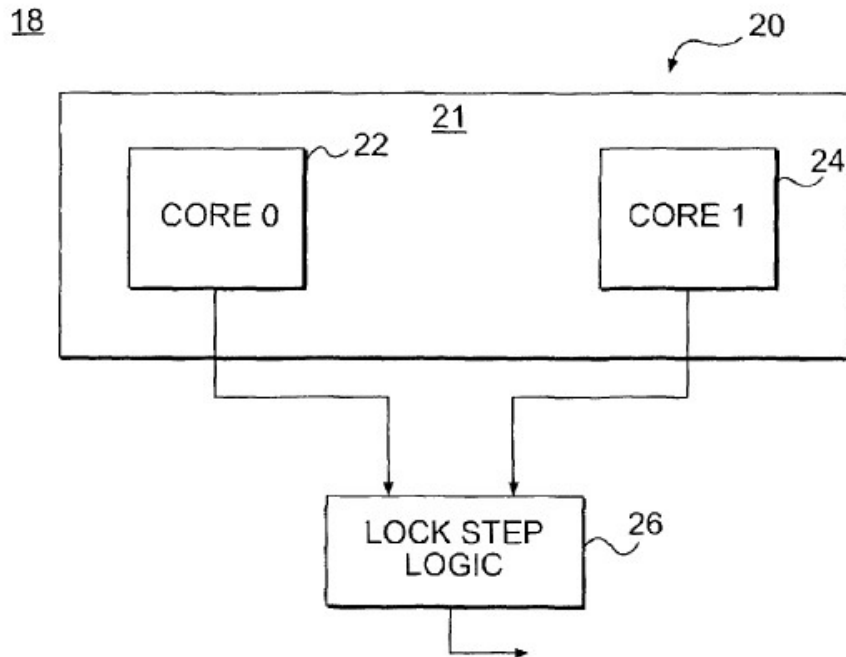


FIG. 1B
PRIOR ART

Id., FIG. 1B.

41. Because lockstep processors execute redundant instructions in parallel, lockstep systems are said to perform a “functional redundancy check.” Safford-181, 1:67-2:3. Thus, processors operating in lockstep were also referred to as processors operating in functional redundancy check mode or Functional Redundancy Check (“FRC”) mode. A POSITA would have understood that both

the terms FRC processor or a lockstep processor refer to a processor including two or more processor cores operating in FRC mode or in lockstep.

42. If comparing the outputs of the pair of processors indicates that they have produced the same outputs, it is assumed that there is no error in processing. If the comparison indicates a difference or mismatch, it is assumed that at least one processor has an error. Safford, [0003]-[0004]. An example of such an error is a data cache error or a soft error.¹ *Id.*, [0003]; Nguyen, [0008]. A loss of lockstep, if not promptly corrected, may cause the computer system to crash. That is, a failure of one of the pair of processors may halt the entire computer system, even if the other processor does not encounter an error. Safford, [0003].

43. It was conventional and well known to a POSITA that a pair of lockstep processors are often differentiated by designating one as a master

¹ A POSITA would have understood that a “soft error” results when, for example, a high-energy particle switches the state of a memory element causing a parity or other error. Nguyen, [0004]; Kondo, [0006]. Soft errors are generally correctable by rewriting the effected memory location. Soft errors are distinct from so-called “hard” errors which indicate the actual failure of a hardware element.

processor and the other as the slave processor, and that these roles may change over time, and that processors have status registers for recording such designations. Marshall, 1:11-16; Kondo, [0086]; Nguyen, [0027].

44. It was also known that an error detected in an individual processor of a pair of lockstep processors signals an impending mismatch error. Safford, [0017] (error detected in an single processor core of a lockstep processor signals an *impending* loss of lockstep); Kondo, [0101] (detecting an error in the processor provides *early warning* of a divergence), [0103] (“When utilizing lockstepped processors, a recoverable error on one processor ... *will* nevertheless result in a divergence, since it will take additional processing and some number of clock cycles on the part of the one processor to recover from the error.”), [0106] (divergence between the processor cores of an FRC processor can be *predictively detected* by detecting errors in the processor cores); Nguyen, [0008] (soft errors in FRC-enabled processors are likely to be manifested as FRC errors since they take the execution cores out of lockstep), [0022] (data corruption in a processor core of an FRC processor can *trigger* an FRC error).

B. Using a Boot Processor to Initialize a Multiprocessor System Was Known

45. It was conventional and well known to a POSITA that one of the processors in a multiprocessor system would be the boot processor (also referred to

as the bootstrap processor or BSP) that initializes the system, including initializing the other processors and booting the operating system. Schelling, [0003]-[0005] (a common approach to initialize a multiprocessor system is to select a single processor as the BSP that has complete control over the boot process while the other processors are in an idle state); Jayakumar, 1:30-43, 3:26-45 (a processor in a multiprocessor system is selected as the BSP to handle initialization procedures for the system as a whole, including loading the operating system and starting the remaining processors); Datta, 1:18-39 (in a multiprocessor system, a BSP is usually selected to handle the initialization procedures for the system as a whole, including checking the integrity of memory, identifying and initializing other resources in the system, loading the operating system into memory, and initializing the remaining processors); IA-32, 238-239 (The BSP is responsible for initializing the multiprocessor system and for booting the operating system, e.g., executing the BIOS's boot-strap code to configure the APIC environment, setting up system-wide data structures, and starting and initializing the applicable processors.).

46. For example, in a multiprocessor system configured with IA-32 architecture (*see* Fox, Abstract, [0014]-[0015]), following a power-up or reset, one of the processors in the system is selected as the BSP (boot processor or bootstrap processor) and the remaining processors are designated as APs (application

processors). IA-32, 238. As part of the selection process, a BSP flag of the selected processor is set to 1 indicate it is the boot processor while the BSP flag is set to 0 for each of the APs. *Id.*, 238, 276-277.

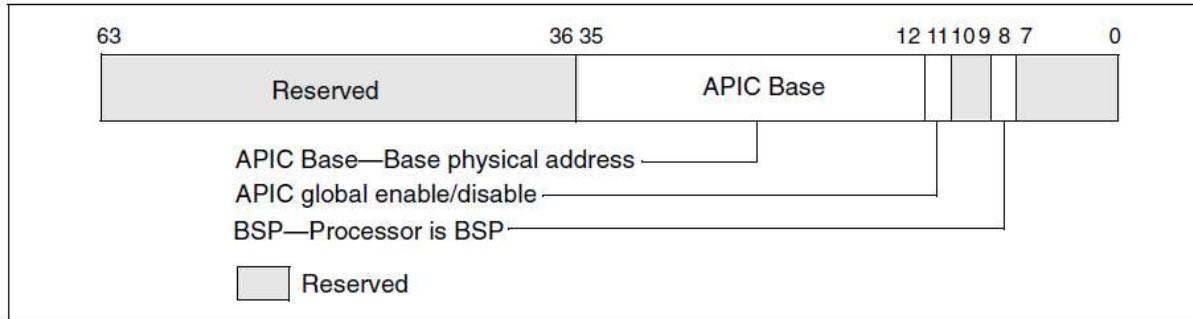


Figure 8-5. IA32_APIC_BASE MSR

Id., 276 (Fig. 8-5).

47. During the initialization and boot process, the BSP executes the BIOS code to boot the operating system, initialize the APICs, and build the MP configuration table to communicate to the operating system about the APICs and APs. *Id.*, 244; MP Specification, 69, 72-73, 76. After the initialization is completed, each processor is assigned and can be identified by a unique local APIC ID. IA-32, 244; MP Specification, 33. Once the operating system is up and running, the BSP functions as an AP. MP Specification, 75. “The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the BSP operating as the last running processor during system shutdown.” *Id.*, 75,

79. The BSP also controls all system hardware, including AP startup and shutdown. *Id.*

48. It was also known that conventionally, in multiprocessor systems, a logical CPU identifier, such as logical CPU 0, would have been used to designate the boot processor. *See* Landry, Abstract, 1:6-10, 2:9-41, 6:44-7:52, Figs. 1, 4 (logical CPU0 is a boot processor that awakens the remaining CPUs and boot up the rest of the computer system); Cepulis, Abstract, 2:30-64 (one operational CPU is selected as logical CPU0 that then awakens the remaining CPUs and boots the rest of the multiprocessor system); Jin, [0048] (logical CPU 0 is the bootstrap processor). A POSITA would have understood that using the logical CPU 0 for designating the boot processor ensures at least one of the processors is assigned as the boot processor and improves dependability of the system. Landry, 2:9-41. A POSITA would have understood that using the logical CPU 0 allows replacing the physical processor that does not function with another functional processor as the boot processor without interrupting the operating system. Slegel, 3:7-46 (the capability of the spare processor to assume the logical identification of a failing processor is critical to making the processor sparing operation appear transparent to the operating system).

C. Fault Recovery to a Spare Processor During System Runtime in a Multiprocessor System Was Known

49. Before the date of the '302 Patent, another typical approach to improve fault tolerance of a multiprocessor system is to use one or more spare processors to quickly respond to a processor failure. Such fault recovery process quickly replaces a failing or failed processor with a spare processor to minimize disruption to the system, allowing the executing processes and the operating system to continue execution during the replacement. Fox, [0013]; Arai, Abstract, [0019] (a processor with an error is replaced with a reserved replacement processor); Slegel, 6:16-36 (teaching transparent sparing where the spare processor assumes the role of the checkstopped processor); Armstrong, Abstract, [0019]-[0020], [0050] (replacing a failing processor with an available replacement processor in a manner transparent to the operating system running on the multiprocessor system).

50. For example, Fox discloses a multiprocessor system where at least one processor of the system is reserved as a hot-spare processor hidden from the operating system and other processors are initiated for regular processing during boot. Fox, [0034], [0037]. Upon detection of a processor failure during system runtime, the hot-spare processor is activated and replaces the failing processor. *Id.*, [0038]-[0046]. Arai also teaches a multiprocessor system where one or more

processors are reserved as inactive processors during system initialization. Arai, [0019]-[0022]. If an error requiring replacement of an active processor is detected and there is a reserved inactive processor, the failed active processor is replaced with the reserved inactive processor by switching the identification numbers of the failed active processor and the reserved inactive processor and transferring the execution context of the failed processor to the reserved processor. *Id.*, [0002]-[0023], [0025], FIGS. 4a-4b.

51. Similarly, Zorek discloses “a processor failure recovery mechanism for a multiprocessor system that does not interrupt system runtime operations.” Zorek, 3:18-21, 3:44-49, 6:16-22 (“transparent to the operating system,” “imposes minimal if any overall runtime processing disruption”). During system startup of Zorek’s multiprocessor system, one or more replacement agents (e.g., processor, processor core, or logical processor) are allocated. *Id.*, Abstract, 2:8-14. When a faulty processor agent (e.g., failed or failing processor) is detected, the operating state of the faulty agent is copied to a selected replacement agent and the interrupt controller ID of the faulty agent (e.g., APIC ID) is assigned to the replacement agent. *Id.*, 2:61-67, 3:1-17, 7:40-57. Having assumed the state and interrupt controller ID from the faulty agent, the replacement agent resumes runtime processing in place of the faulty agent. *Id.* Zorek discloses that such assignment of

the interrupt controller ID and copying the operating state enable the replacement processor agent to “seamlessly replac[e] a faulty processor agent in a manner that preserves the operating state of the agent and avoids data loss and system interruption.”). *Id.*, 8:9-19.

52. Indeed, it was known that using both lockstep processors and spare processors in a multiprocessor system improves reliability and availability the system. For example, IBM S/390 G5 server system uses both types of fault tolerance designs. In this server system, CPU units are paired to perform dual execution and a comparator for each pair of dual execution CPU units detects any mismatch and prevents erroneous results from spreading. Ex. 1027, 6-7. This server system also implements Dynamic CPU sparing (DCS) that replaces a failed processor with a spare processor and recovers the failed processor transparent to the operating system. Ex. 1026, 1-2, 5-7. The spare processor assumes the identity of the failed processor and begins executing the same instruction at the point where the failed CPU left off. *Id.*, 7 (teaching that to ensure that replacement of a failed processor with a spare processor is transparent to the operating system, the spare processor is tagged with the address of the failed processor, which makes the spare processor be ready to begin execution instructions to the failed processor);

Ex. 1027, 7-8 (“Once the replacement for the clock-stopped PU is identified, the physical/logical ID fields in the configuration area are updated.”).

D. Using Firmware to Switch a Failing Processor to a Spare Processor During System Runtime Was Known

53. It was conventional and well known to a POSITA that firmware would be used to switch a failing processor to a spare processor to minimize interruption to the operating system. For example, it was well known that BIOS is firmware typically embedded in a non-volatile memory (e.g., ROM, EEPROM, etc.) on the mother board. *See* Fox, [0028], FIG. 1; Zorek, 5:47-53, FIG. 2. Fox discloses using the BIOS of its IA-32 multiprocessor system to implement failover to the spare processor, such as using the BIOS to initialize the spare processor during boot, and using the Systems Management Interrupt (SMI) function, processor failure response logic, and interrupt handler of the BIOS to detect a processor failure and switch the failing processor to the spare processor. Fox, [0015], [0032], [0034]-[0035], [0040]-[0041].

54. Arai also discloses using the BIOS to replace an active processor with a reserved processor, such as designating one or more reserved processors and generating a table in the main memory identifying the operational state of each processor during system initialization, and using the BIOS interrupt handler to determine the cause of the error, determine whether the error requires replacement,

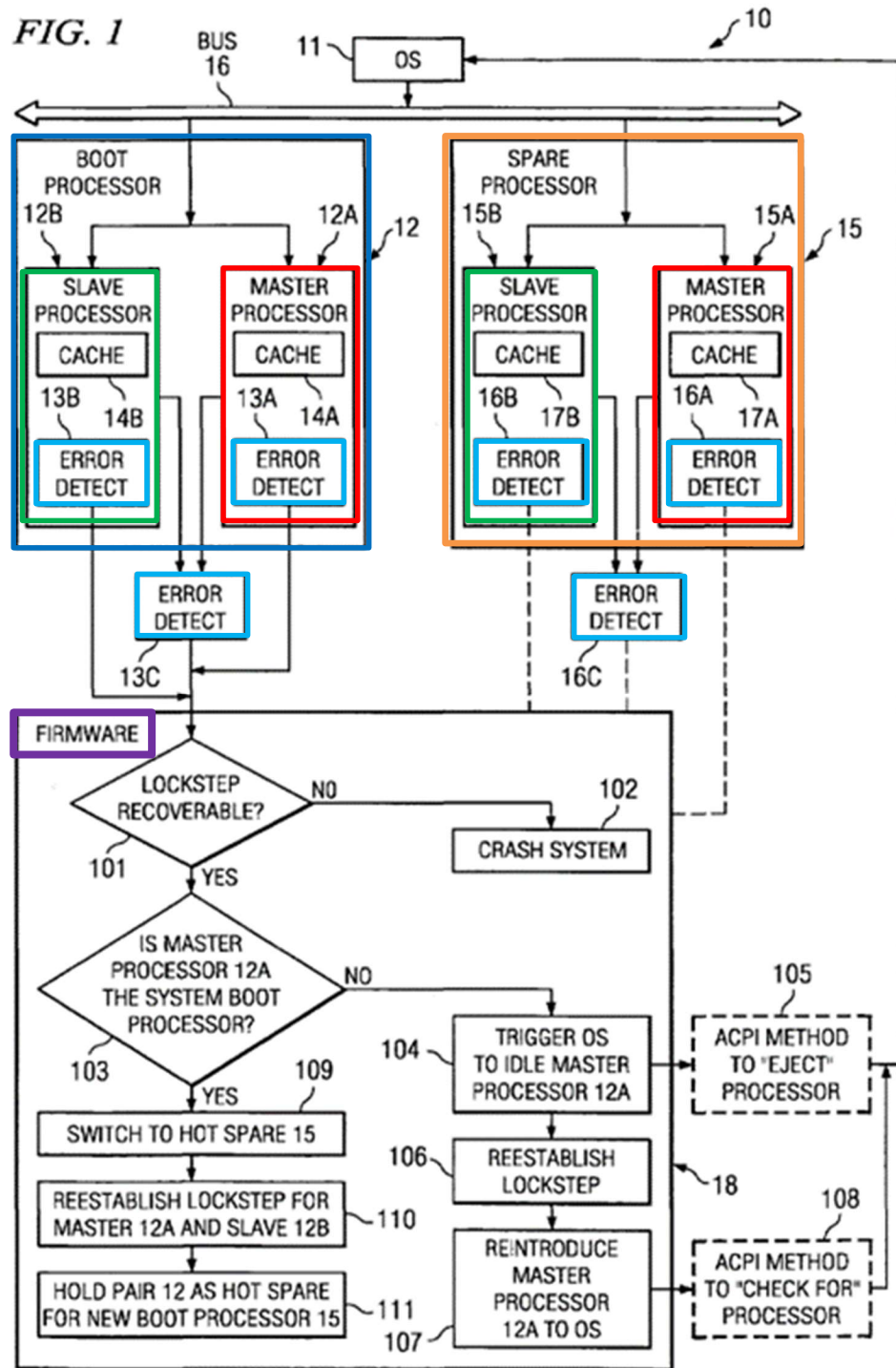
determine the availability of a reserved processor, and transfer execution context of the failing processor to the reserved processor, Arai [0020]-[0022].

55. Further, Armstrong discloses performing the replacement of a failing processor with a replacement processor while the system is running an operating system by “system firmware without intervention by the operating system (i.e., the replacement may be transparent to the operating system).” Armstrong, Abstract. Armstrong discloses using the system firmware to detect a failing processor, identify an available replacement processor, save a state of the operating system on the failing processor, redirect interrupts from devices owned by the operating system from the failing processor to the replacement processor, and restore the saved state of the at least one operating system on the replacement processor. *Id.*, Abstract, [0019]-[0022], [0044]-[0052], FIG. 4; *see also* Slegel, 2:35-43 (teaching using firmware to enable transferring the micro-architected state of a checkstopped processor to a spare processor); Ex. 1027, 7 (teaching the common hardware and firmware allow a spare CPU unit to replace any failing CPU unit).

VI. Overview of the '302 Patent

56. The '302 Patent claims priority to October 25, 2004. Ex. 1001, cover. The '302 Patent discloses a multi-processor system and method that switches a lockstep boot processor that lost lockstep to a spare processor. *Id.*, Abstract, 3:64-

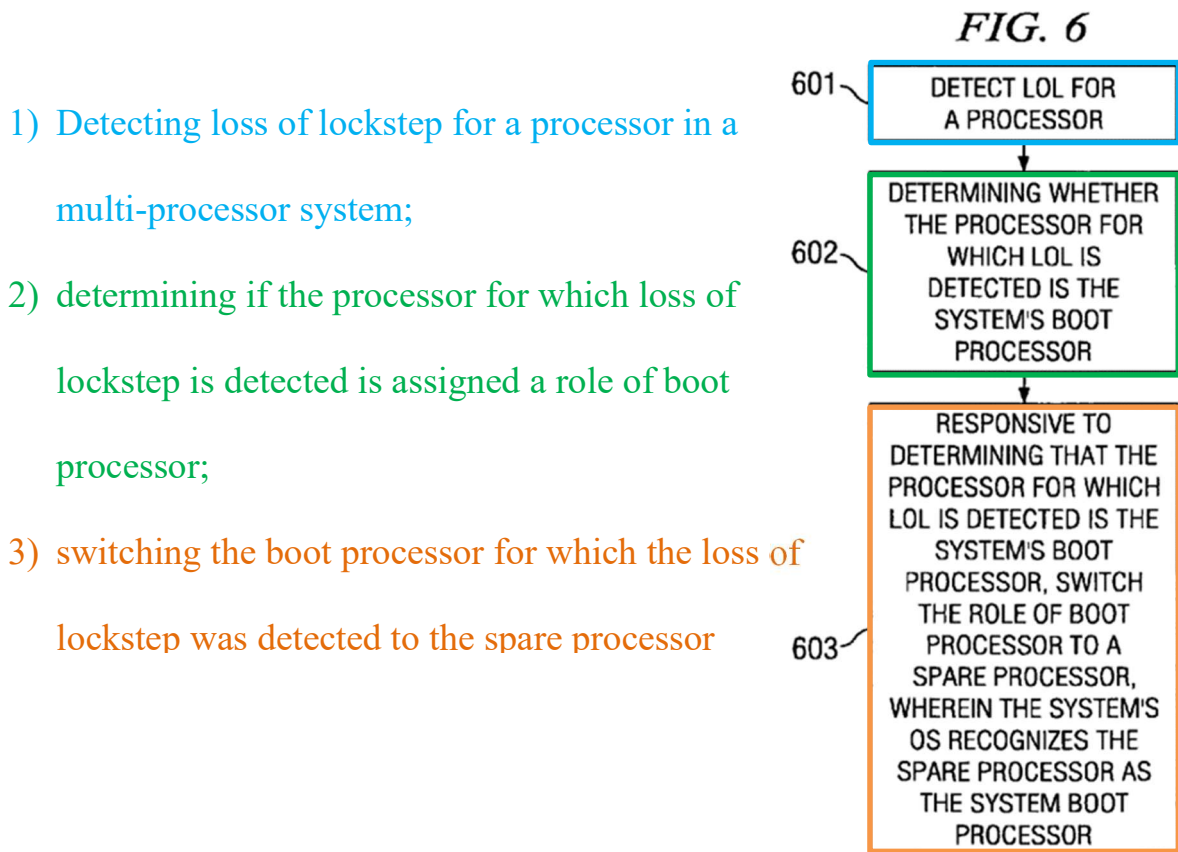
4:13, FIGS. 1-6. The multiprocessor system includes a **boot processor**, a **spare processor**, and processors other than the boot processor and spare processor. *Id.*, 4:14-27, 7:44-50, 12:22-35, 13:21-35. Each of the processors includes a lockstep processor pair, including a **master processor** and a **slave processor**, and **error detect logic** to detect loss of lockstep (“LOL”) between the processor pair. *Id.*, 4:14-19, 6:42-58, 7:6-13, FIG. 1 (reproduced below). The multiprocessor system includes **firmware** that, responsive to LOL, determines whether the detected LOL is recoverable. *Id.*, 9:32-45. If the LOL is recoverable, the firmware performs a recovery process to recover lockstep of the processor; if the LOL is not recoverable, the firmware crashes the system. *Id.*, 6:10-24, 9:32-45, 9:65-11:3.



Id., FIG. 1 (annotated).

57. The '302 Patent explains that in certain system architectures, performing its recovery process for the boot processor of the system would cause problems. *Id.*, 6:25-41, 11:4-11. Thus, the '302 Patent discloses in some embodiments, a spare processor is used for recovering the boot processor from LOL. *Id.*; *see also id.*, 7:53-64, 9:16-31. For example, after determining the detected LOL is recoverable, and determining that the processor having LOL is the boot processor, the firmware switches the designation of the boot processor to the spare processor for recovering from the LOL. *Id.*, 7:53-64, 8:14-16, 8:34-62, 9:46-10:40, 11:6-31, 11:45-60, 13:60-14:3, 14:43-65, 16:1-18:48, FIGS. 4-5.

58. The '302 Patent's independent claims recite three components, all of which were disclosed in the prior art and all of which would have been obvious to a POSITA at the time (*id.*, 18:46-19:3, 19:6-20, 20:1-19, 20:40-56, 21:5-22:5, FIG. 6):



VII. Overview of the Prior Art

A. Overview of Fox (Ex. 1005)

59. Fox was filed on January 21, 2004, and published on August 5, 2005.

Fox, cover. Fox was filed prior to the priority date of the '302 Patent, and thus qualifies as prior art to the '302 Patent under 35 U.S.C. § 102(e).

60. Fox discloses a method and system that switches a failing processor to a spare processor in a multiprocessor server system. Fox, Abstract, [0014], [0016], [0019], [0023]-[0024], [0044], FIG. 1. Fox's multiprocessor server system is configured with IA-32 architecture and uses Systems Management Interrupt (SMI)

functionality provided within **basic input output system (BIOS)** of the IA-32 architecture. *Id.*, Abstract, [0014]-[0015], [0028], [0033]. Fox discloses that, during system boot, the BIOS reserves at least one processor of the multiprocessor system as a **hot-spare processor** hidden from the operating system and initiates other **main processors** for regular processing. *Id.*, [0023]-[0025], [0034], [0037], [0040], [0045]-[0046]. The hot-spare processor remains in an idle, off, or low-power state until it is activated when a processor failure occurs. *Id.*, [0015], [0035], [0038], [0040]. Fox's system's BIOS comprises a processor failure response logic that generates the SMI when a processor failure is detected. *Id.*, [0040]-[0041], [0045]-[0046]. The SMI identifies to the operating system (OS) and the BIOS the failing processor and initiates the activation of the spare processor and a switch over response that replaces the failing processor with the spare processor by the BIOS. *Id.*, [0038]-[0046], FIG. 3.

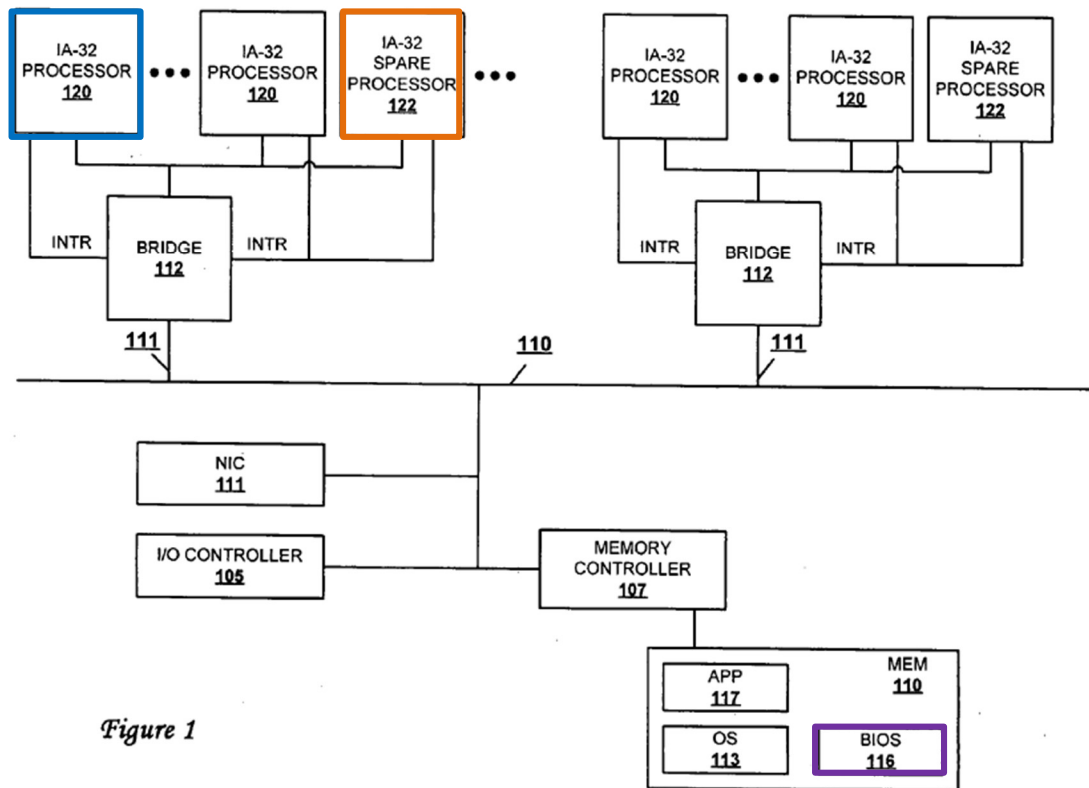


Figure 1

Id., FIG. 1 (annotated).

61. Fox explains that its method and system maintains system uptime even while a processor in the multiprocessor system is failing, and provides high availability during processor failure without crashing all of the processors. *Id.*, [0006], [0016], [0036], [0044], [0047]. Fox also explains that its method and system apply to operations in a critical server environment that requires continuous uptime of the system. *Id.*, [0036].

62. Fox discloses an IA-32 multiprocessor system that mandates a boot processor. Therefore, a POSITA would have understood that one of the processors

of Fox's multiprocessor system is a boot processor because a boot processor is necessary to execute Fox's BIOS to initialize Fox's multiprocessor system configured with IA-32 architecture. *Id.*, [0034] (disclosing performing "BIOS boot"); *see also id.*, [0015], [0040], [0045]; *see* IA-32, 238 (teaching during system initialization of an IA-32 multiprocessor system, selecting one of the processors as the bootstrap processor (BSP) to execute the BIOS's boot-strap code and initialize the system); *see also* Schelling, [0003]-[0005] (teaching selecting a boot processor to prepare an IA-32 based multiprocessor system for full operation); MP Specification, 17 (teaching that the BSP is responsible for initializing the system and for booting the operating system).

63. A POSITA would have understood that Fox's SMI identification of the failing processor to the OS and BIOS would have identified whether the failing processor is the boot processor because (1) the OS and BIOS would have known the identification of all the processors of the system, including the boot processor; and (2) Fox teaches transferring an interrupt transaction in a manner that identifies the processor to be interrupted. Fox, [0030]-[0032], [0034], [0042].

B. Overview of Safford (Ex. 1006)

64. Safford was filed on July 3, 2002, and published on January 8, 2004. Safford, cover. Safford was published prior to the priority date of the '302 Patent, and thus qualifies as prior art to the '302 Patent under 35 U.S.C. § 102(a).

65. Safford discloses a multiprocessor computer system 100, wherein each of the processors includes a pair of processor cores 111 and 113 operating in lockstep. Safford, Abstract, [0015], FIGS. 2-3. Safford discloses the lockstep processor pair includes a master/checker pair that process the same code sequences. *Id.*, [0015]. Safford's lockstep processors include a logic circuit 115 to detect differences in the outputs of the pair of processors indicating a loss of lockstep and error detection, and signaling logic 112 and 114 to detect errors in the respective processor cores to signal an impending loss of lockstep. *Id.*, [0015]-[0017].

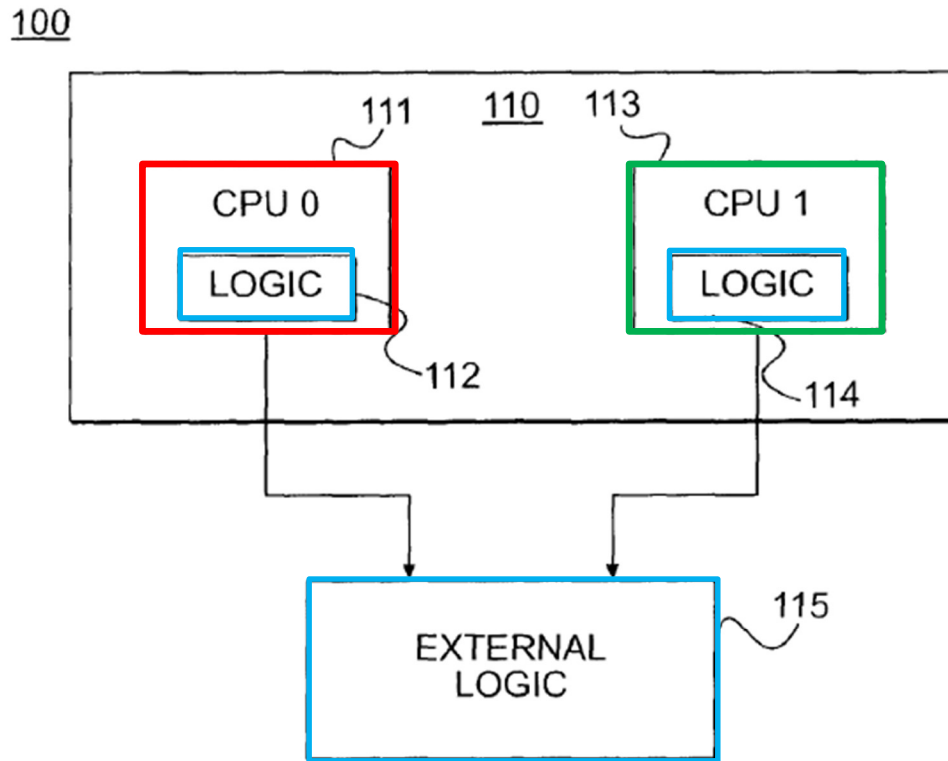


FIG. 2

Id., FIG. 2 (annotated).

66. Safford discloses a process that uses a hot spare processor to recover a loss of lockstep of a lockstep processor pair. *Id.*, [0018], [0020]-[0021], FIGS. 3-4. As shown below, in addition to a first **lockstep pair of processors 111 and 113**, Safford's computer system 100 includes a second lockstep pair of processors 121 and 123 and a **third pair of processors 125 and 127**. *Id.*, [0018]. Each pair of lockstep processors appear as a single logical processor and are coupled to a lockstep logic as shown in above Figure 2. *Id.* Processor pair 125/127 is designated

as a **spare processor pair**. *Id.*, [0020]-[0021]. When a loss of lockstep error is detected in processor pair 111/113 (logical CPU 0), the **spare processor pair 125/127** replaces **processor pair 111/113** and becomes the logical CPU 0. *Id.*, [0021], FIGS. 3-4. After the first processor pair 111/113 is recovered, it becomes the new hot spare processor pair of the system. *Id.*

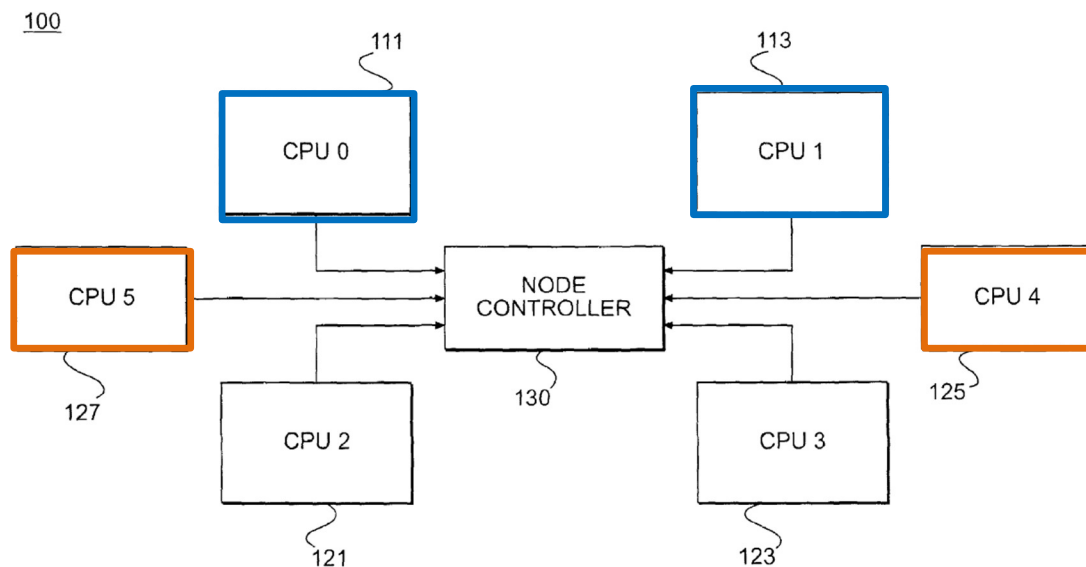


FIG. 3

Id., FIG. 3 (annotated).

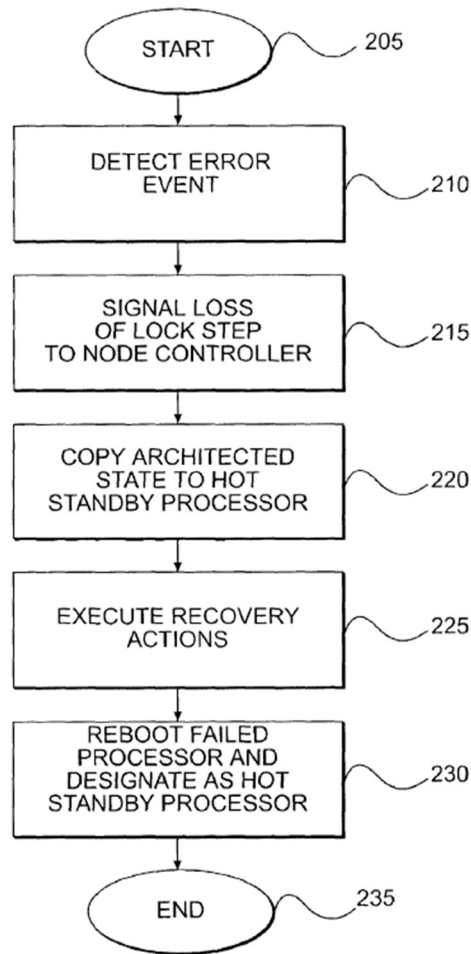


FIG. 4

Id., FIG. 4.

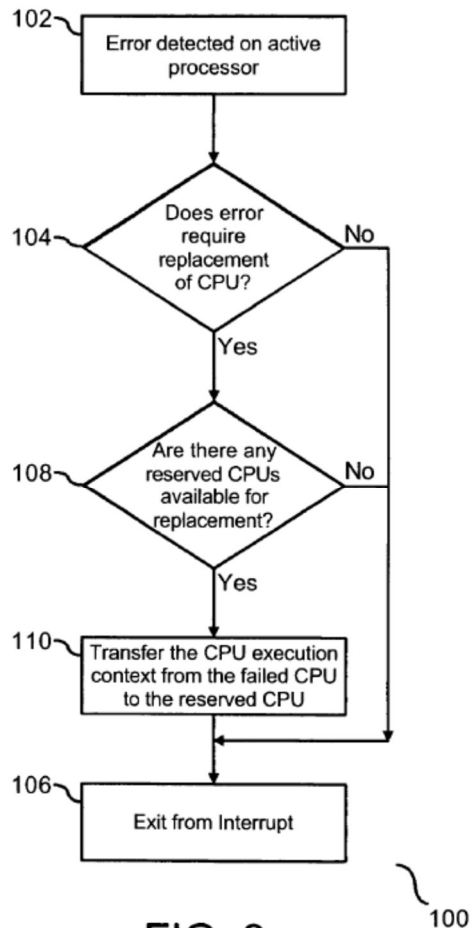
C. Overview of Arai (Ex. 1007)

67. Arai was filed on August 16, 2004, and published on February 16, 2006. Arai, cover. Arai was filed prior to the priority date of the '302 Patent, and thus qualifies as prior art to the '302 Patent under 35 U.S.C. § 102(e).

68. Arai discloses a system and method that replaces a processor with an error with a reserved replacement processor in a multiprocessor system. Arai, Abstract, [0002], [0008]-[0009], [0019]. Arai discloses that during system initialization, the BIOS creates a table that identifies each processor in the system and its designated operational state, i.e., active or disabled. *Id.*, [0019]-[0022]. An active designation of a processor indicates the processor is available to the operating system, and a disabled designation of a processor indicates the processor is in an idle state and not available to the operating system. *Id.* Each logical processor has an identification number and an associated flag to indicate whether the processor is enabled or disabled. *Id.* The table is stored in the main memory (e.g., RAM) and can be ACPI (Advanced Configuration and Power Interface) MAPIC (Multiple Advanced Programmable Interrupt Controller) table, a structure used by ACPI. *Id.*

69. Arai discloses that following the boot of the operating system, when an error is detected on an active processor, the BIOS determines the cause of the error and determines whether the error requires replacement of the active processor and the availability of an inactive processor. *Id.*, [0022]-[0023], FIG. 3. If the BIOS determines the error requires replacement and if there is a reserved inactive processor, the identification numbers of the active processor in error and the

reserved inactive processor are switched, and the execution context of the active processor is transferred to the reserved inactive processor. *Id.*, [0022]-[0023], [0025], FIGS. 4a-4b. The execution context includes information in the processor registers and other data that supports communication between the processor and the operating system. *Id.* Arai explains its method and system allow for error recovery transparent to the operating system. *Id.*, [0004]-[0006].

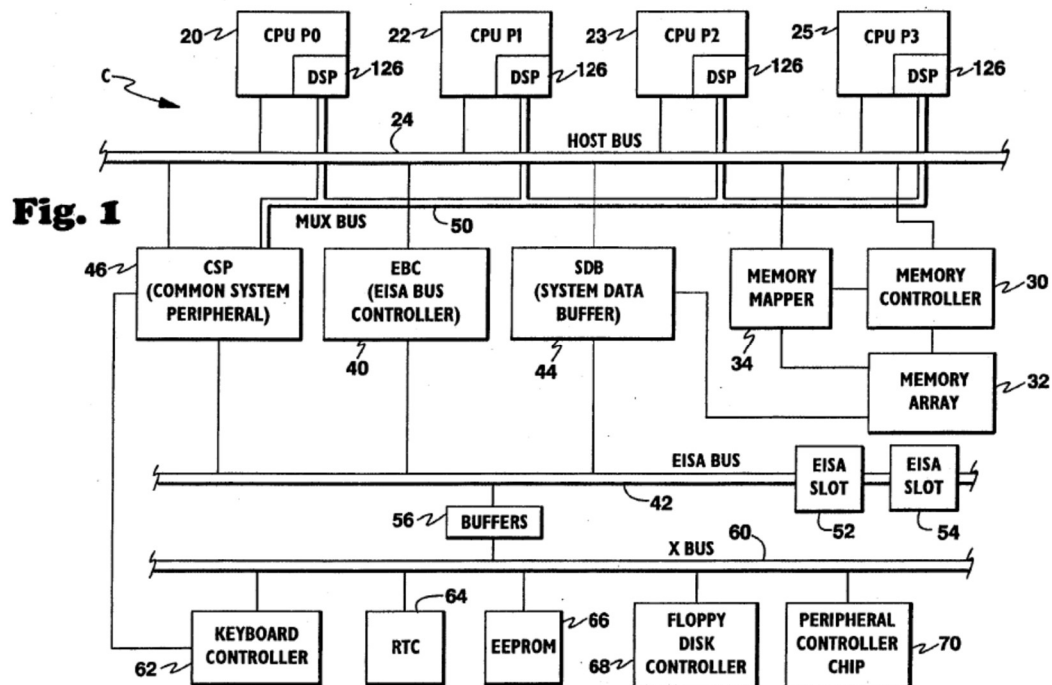


Id., FIG. 3.

D. Overview of Landry (Ex. 1011)

70. Landry was filed on October 2, 1992, and granted on April 8, 1995 (Landry, cover), over one year before the priority date of the '302 Patent, and thus qualifies as prior art to the '302 Patent under 35 U.S.C. § 102(b).

71. Landry discloses a multiprocessor computer system that includes multiple CPUs 20, 22, 23, 25. Landry, Abstract, 2:59-3:3, Fig. 1.



Id., Fig. 1.

72. Landry discloses a process of assigning a functional CPU as the logical CPU0 (CPU L0) among multiple physical CPUs, CPU P0, P1, P2, and P3, to boot up the system. *Id.*, Abstract, 1:6-10, 2:9-41, Figs. 1, 4. For example, when

the system is powered up, all of the physical CPUs except CPU P0 are initially placed in a sleep mode and CPU P0 performs its power on self-test (POST). If CPU P0 functions properly, CPU P0 is designated as CPU L0. *Id.*, Abstract, 1:6-10, 2:9-41, Figs. 1, 4. If CPU P0 is not functioning properly, CPU P0 is designated as inoperative and the system repeats the process of testing CPU P1. *Id.*, Abstract, 1:6-10, 2:9-41, 6:44-7:52, Figs. 1, 4. If CPU P1 is functioning properly, CPU P1 is designated as CPU L0. If CPU P1 is not functioning properly, this process repeats until an operational microprocessor is found operational and designated as the CPU L0. *Id.*, Abstract, 1:6-10, 2:9-41, 6:44-7:52, 7:4-25, Figs. 1, 4. A POSITA would have understood Landry's CPU L0 would be the boot processor of the system because it boots the rest of the system and executes the boot software. *Id.*; see Section V.B.

E. Overview of Bigbee (Ex. 1012)

73. Bigbee was filed on January 2, 2002, and published on July 3, 2003 (Bigbee, cover), over one year before the priority date of the '302 Patent, and thus qualifies as prior art to the '302 Patent under 35 U.S.C. § 102(b).

74. Bigbee discloses a data processing system including hardware 102, firmware 104, and an operating system 106. Bigbee, [0014]-[0025], FIG. 1 (reproduced below).

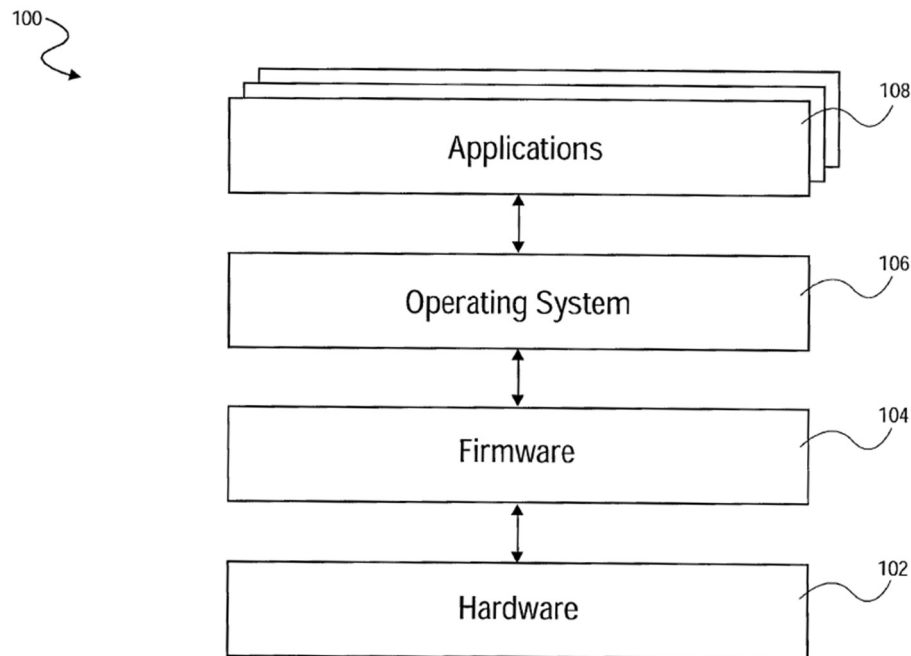


FIG. 1

Id., FIG. 1.

75. Hardware 102 includes two or more processor packages, each processor package including two processor cores, e.g., Itanium[®] family processors, operating in lockstep functional redundancy check (FRC) mode as a single logical processor from the point of view of operating system 106.² *Id.*, [0019], [0032]-

² A POSITA would understand that functional redundancy check mode, or FRC mode, is a mode in which processors are processing the same instructions at the

[0033], FIG. 3 (reproduced below). The two processor cores concurrently execute identical instructions on identical data and each processor generates results. *Id.*, [0002], [0033]. An FRC logic 308 monitors and detects inconsistencies in the results of the two processor cores that indicate an FRC mismatch error. *Id.*

same time and producing outputs from that processing at the same time, or “in lockstep.” Thus, FRC mode is synonymous with operating in lockstep mode or, simply, in lockstep. If comparing the output of the processors indicates that they have produced the same output, it is assumed that there is no error in processing. If the comparison indicates a difference, it is assumed that at least one processor has a fault.

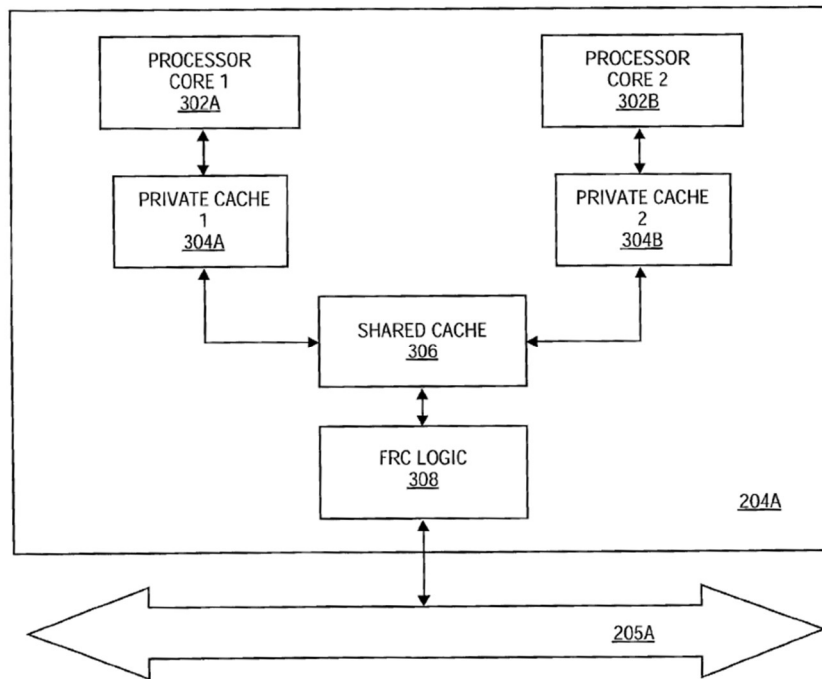


FIG. 3

Id., FIG. 3.

76. The FRC logic also detects an error within each of the processor cores and generates a system machine check to cause the firmware and the operating system to perform an error recovery routine that recovers the FRC operation (“lockstep”) of the cores. *Id.*, [0035]-[0037], FIGS. 4A-4B. Bigbee further discloses implementing its FRC logic in executable firmware. *Id.*, [0034].

77. Bigbee discloses the processor architecture of the Itanium family. *Id.*, [0033]; Ex. 1030, 267-268. As was known for the Itanium-based firmware model, Bigbee discloses its firmware includes “an extensible firmware interface (EFI), a system abstraction level (SAL), and a processor abstraction level (PAL).” Bigbee,

[0018]; Ex. 1030, 267-269. Bigbee also discloses its system is ACPI-compliant, and its firmware comprises an ACPI BIOS, ACPI control methods, objects, registers, and tables. Bigbee, [0016].

F. Fox, Safford, Arai, Landry, and Bigbee are Analogous Art to the '302 Patent

78. Like the '302 Patent, Fox, Safford, Arai, Landry, and Bigbee all relate to fault-tolerant systems that can operate despite the occurrence of processor error in the system. A POSITA would have understood that, like the '302 Patent, Fox's multiprocessor system detects a processor failure and, in response to detecting the processor failure, replaces the failing processor with a spare processor. Fox, Abstract, [0006], [0014], [0016], [0019], [0023]-[0024], [0036], [0044], [0047], FIG. 3. Similarly, Safford's multiprocessor system detects loss of lockstep in a pair of lockstep processors and, in response to detecting the loss of lockstep, replaces the lockstep processor pair in error with a spare processor pair. Safford, [0018]-[0021], FIGS. 3-4. Also, Arai's multiprocessor system detects an error in an active processor and, in response to detecting the error, replaces the active processor in error with a reserved inactive processor. Arai, [0022]-[0023], [0025], FIGS. 4a-4b. Landry discloses a fault-tolerant power-up logic in finding a functional CPU to operate as the logical CPU 0 or boot processor to boot the system. Landry, Abstract, 1:6-10, 2:9-41, 6:44-7:52, Figs. 1, 4. Bigbee discloses a system including

a plurality of processors, each including two processor cores operating in FRC or lockstep mode. Bigbee, [0026]-[0027], [0033], FIG. 2. Bigbee discloses detecting loss of lockstep in a lockstep pair of processors and recovering the loss of lockstep in response to the detection. *Id.*, [0018]-[0023], [0035]-[0037], FIG. 4A. Fox, Safford, Arai, Landry, and Bigbee are each in the same field of endeavor as the '302 Patent and are analogous art to the '302 Patent.

VIII. Ground of Unpatentability

79. Based on my review of the materials set forth above, including my application of the knowledge of a POSITA, it is my opinion that the Challenged Claims of the '302 Patent would have been rendered obvious by certain prior art to one of ordinary skill in the art.

80. In particular, it is my opinion that the Challenged Claims of the '302 Patent would have been rendered obvious to one of ordinary skill in the art, based on the ground shown below:

Ground	Claims	Reference(s)	Basis
1	1-25	Fox in view of Safford and Arai	§ 103
2	1-25	Fox in view of Safford, Landry, and Arai	§ 103

3	26, 27	Fox in view of Safford, Arai, and Bigbee	§ 103
4	26, 27	Fox in view of Safford, Landry, Arai, and Bigbee	§ 103

IX. Ground 1: Fox in view of Safford and Arai Renders Obvious Claim 1

81. In my opinion, Fox in view of Safford and Arai renders obvious claim 1.

A. Fox in view of Safford and Arai

82. Fox discloses a multiprocessor system with firmware that detects a processor failure and switches the failing processor with a spare processor without interrupting the operating system. *See* Section VII.A.

83. Fox does not disclose that its multiprocessor system includes lockstep processors or detecting loss of lockstep, but Safford discloses a multiprocessor system including pairs of lockstep processors with error detection logics and switching a pair of lockstep processors detected to have loss of lockstep to a spare processor pair. *See* Section VII.B. A POSITA would have found it obvious to implement the processors of Fox's multiprocessor system as lockstep processor pairs with error detect logics as taught in Safford to improve the reliability and availability of Fox's server system. Fox, [0016], [0044], [0047] (teaching providing high availability of the system); Suffin, 6:4-12 (teaching lockstep

processors provide high level of availability); Safford, [0002] (lockstep processors enhance reliability of multiprocessor systems), [0017] (teaching both internal and external error detection logics improve availability of processor assets).

84. Although Fox does not explicitly disclose a boot processor, a POSITA would have understood that Fox's multiprocessor system would have a boot processor to initialize the system and load the operating system. *See* Section V.B. As explained in more detail below, a POSITA also would have understood that Fox's BIOS would have determined whether the failing processor is the boot processor. A POSITA also would have understood that Fox's spare processor would be used for replacing the boot processor in response to determining that the failing processor is the boot processor. A POSITA would have further understood that in Fox combined with Safford's lockstep processors and error detection logics, detecting failure in a processor would include detecting loss of lockstep between the pair of lockstep processors.

85. Further, Arai discloses switching the role of a failing processor with a spare processor. Arai discloses switching the identifications between the failing processor and the spare processor and transferring the context of the failing processor to the spare processor. *See* Section VII.C. A POSITA would have

understood that Arai provides simple implementation details for how to switch the failing processor and the spare processors, as suggested by Fox.

B. A POSITA Would Have Found It Obvious to Combine Fox, Safford, and Arai

86. A POSITA would have found it obvious to combine Fox's multiprocessor system with Safford's lockstep processor and error detection logics and Arai's processor-switching implementation details.

1. A POSITA would have found it obvious to implement Fox's processors as lockstep processors with error detection logics that detect loss of lockstep, as taught in Safford, to improve system reliability and availability.

87. As discussed in Section V.A, it was well known and common to use lockstep processors in multiprocessor systems to improve system reliability and availability. And it was also known to use both lockstep processors and spare processors in a fault-tolerant multiprocessor system. *See* Section V.C. A POSITA would have been motivated to implement the processors in Fox as lockstep processors to improve reliability and availability of Fox's server system for critical processes, for which higher reliability and availability of the system are desired. Fox, [0006], [0016], [0044], [0047]. Other systems that would benefit from high reliability and availability includes online real-time transaction systems, aircraft control, and telecommunications systems, etc. A POSITA also would have been

motivated to implement in Fox the error detection logics that detect loss of lockstep together with the lockstep processors as taught by Safford to implement lockstep functions and further improve availability of the system. Safford, [0005], [0017]. A POSITA would have understood that when Fox's processors are implemented as lockstep processors, a loss of lockstep between the lockstep processor pair would have been a processor failure to be detected and handled to achieve system high reliability and availability.

88. A POSITA would have understood how to implement Fox's processors as lockstep processors with error detection logics and expect success. A POSITA would only have applied well-known configurations and ordinary hardware and software configuration skills. For example, a POSITA would have implemented each processor of Fox as a dual-core processor with two processor cores operating in lockstep mode. Bigbee, [0002] (two or more processor cores incorporated into a single processor package); Nguyen, [0020] (an FRC-enabled processor including two execution cores operating in FRC model or lockstep mode). Alternatively, a POSITA also would have configured pairs of processors in Fox's system in lockstep mode and held off one or more pairs of processors as spare processor pairs. Fox, [0028], [0034], [0040]; Safford, [0015], [0018], [0020]-[0021] (teaching holding a pair of processors in lockstep mode as a spare processor

pair); Ex. 1027, 5-6 (teaching configuring processor units in dual execution mode and detecting mismatch between the processors). Fox teaches holding at least one processor as a spare processor. *See* Fox, [0034], [0040]; *see also* Arai, [0019] (reserving one or more processors as replacement processors). Fox also teaches the number of processors in the multiprocessor system can be increased or decreased. Fox, [0028].

89. Indeed, a POSITA would have known that processors of a multiprocessor system configured with IA-32 architecture like that in Fox would operate in either independent mode or in lockstep mode with lockstep error detection. IA-32, 470-471 (Table 14-2, showing error code indicating FRC error). A POSITA also would have known that Intel® IA-32 processors were configurable to operate in lockstep mode, such as Intel Xeon® and Pentium® processors. Suffin, 6:13-22; Elkhoury, 6:36-47.

90. Fox also teaches other system resources can be added or removed. Fox, [0028]. A POSITA would have understood how to add the error detection logics, as taught by Safford and well known in the art, to improve system fault tolerance. Nguyen, [0021], [0028], [0032] (teaching similar error detection logics); Bigbee, [0032]-[0033] (teaching similar error detection logics); *see* Section V.A.

91. A POSITA would have understood that loss of lockstep is a processor error condition detected by hardware logic, which would generate the SMI interrupt to be forwarded from the failing processor to another good processor for handling the failure, as taught in Fox. Fox, [0041] (the SMI is generated by hardware during various processor failure conditions); *see also* IA-32, 142 (sources of interrupts include hardware), 269-270, 287-288 (the local APIC and I/O APIC of multiprocessor systems can forward external interrupts, such as SMI, from one processor to the destination processor). When Fox's processors are implemented as lockstep processors with error detection logics, in response to the detection of the loss of lockstep by the error detection logics, Fox's BIOS processor failure response logic would generate the SMI as taught in Fox. Fox, [0032].

92. A POSITA would have understood that when Fox's processors are implemented as lockstep processors, each lockstep processor pair would operate as a single logical processor having the same logical identification, as taught in Safford and known in the art. Safford, [0015] (the processors 111 and 113 will appear to the computer system 100 to be a single processor core, or a logical CPU 0); Bigbee, [0002], [0019], [0027], [0033], [0035] (two or more lockstep processor cores function as a single logical processor).

93. Fox teaches that the multiprocessor system configured with IA-32 architecture includes the APICs—each processor is connected to the APIC. Fox, [0024]-[0026], FIG. 2A. A POSITA would have understood that each APIC would be associated with a unique physical identification, e.g., the local APIC ID. IA-32, 277-278; MP Specification, 33, 43. A POSITA would have understood that in Fox-Safford's system, APIC-related identifications also would be available and both processors of a lockstep pair would share the same logical identification (e.g., logical APIC ID) as taught in Safford. *See* IA-32, 273-274 (APIC logical destination register), 294 (logical APIC ID is contained in the logical destination register), 307 (in logical destination mode, interrupt messages are directed to those processors that are part of the logical group of processors based on the processor's logical APIC ID).

94. A POSITA would have further understood that the logical identification (e.g., logical APIC ID) of each processor would be stored in the register or table of the BIOS that is known to the operating system and BIOS after system initialization. *See* Section V.B. A POSITA would have understood that the processor logical identification (e.g., logical APIC ID) would be used for interrupt communication like that in Fox. Fox, [0026], [0030] (teaching all interrupt transactions on the system bus contain target node and IA-32 processor

identification); IA-32, 267 (in multiprocessor systems, I/O APIC allows for distributing external interrupts to the local APICs of selected processors), 269 (I/O APIC receives interrupts generated by system hardware and forwards them to local APIC as interrupt messages), 289-295 (APICs allow for sending interrupts, such as SMIs, to the destination processor based on either the processor physical identification (e.g., local APIC ID) or the logical identification (e.g., logical APIC ID)).

95. A POSITA would have also understood that the processor logical identification (e.g., logical APIC ID) would be used for switching processors such that the switching would appear to be transparent to the operating system, as desired by Fox. Fox, [0035], [0041], [0045]; Arai, [0023], [0025], FIGS. 4a-4b (teaching switching logical APIC IDs of the failing processor and the reserved processor); Slegel, 3:22-40 (teaching that the spare processor assuming the logical identifier of a checkstopped processor is critical to make the processor sparing operation appear transparent to the operating system); Ex. 1027, 7-8 (“Once the replacement for the clock-stopped [processing unit] is identified, the physical/logical ID fields in the configuration area are updated.”).

96. Accordingly, a POSITA would have found it obvious to implement Fox’s processors as lockstep processors with error detection logics as taught by

Safford according to their known functions. And a POSITA would have understood that in Fox combined with Safford's lockstep processors and error detection logics, a loss of lockstep for a lockstep processor pair would have been detected as a processor failure triggering processor switching. Such modification merely involves using a known technique to improve similar devices in the same way taught and is well within the technical grasp of a POSITA.

2. A POSITA would have understood that Fox's multiprocessor system would have one of its processors designated as the boot processor and replaced the failing boot processor with the spare processor.

97. Fox discloses a multiprocessor system configured with IA-32 architecture. Fox, Abstract, [0014]-[0015]. A POSITA would have understood that such system would have designated one of its processors as the boot processor responsible for initializing the multiprocessor system and booting the operating system. IA-32, 238-239; *see* Sections V.B, VII.A. Indeed, Fox expressly discloses system BIOS boot, which a POSITA would have understood to be performed by a boot processor. Fox, [0006], [0034], Claims 4-6. Thus, a POSITA would have understood that Fox's system would have designated, or it would have been obvious to designate, one of Fox's processors as the boot processor.

98. Fox suggests and a POSITA would have understood that Fox's spare processor would be used for replacing the boot processor when failure in the boot processor is detected to improve system reliability.

99. First, a POSITA would have understood that Fox's spare processor would be used for any of the other main processors, including the designated boot processor and application processors. As explained in Section V.B, once the operating system is up and running, the boot processor functions as an application processor. MP Specification, 75. Thus, a POSITA would have understood that both the boot processor and application processors would be active processors in Fox's system, and Fox's spare processor would replace any of the active processors that are failing. *See* Fox, [0032]-[0033], [0035] (teaching the system has a register or table for tracking active processors), [0045] (teaching during initialization, the spare processor is inactive).

100. A POSITA also would have understood that Fox's SMI's identification of the failing processor to the BIOS would have determined that the failing processor is the designated boot processor or assigned the role of the processor. *Id.*, [0042]. Fox discloses all interrupt transactions contain the processor identification. *Id.*, [0030], [0042]. Each processor has a unique physical or logical identification, and that Fox's SMI's identification of the failing processor would

have included it. IA-32, 277 (system hardware assigns a unique APIC ID to each local APIC at power up), 278 (software must ensure that each APIC ID for each local APIC is unique throughout the system), 255 (system hardware assigns a unique APIC ID to each logical processor), 294 (establishing unique logical APIC ID); MP Specification, 33, 43 (the assignment of APIC IDs must be unique). Fox's BIOS would know the unique identification of each processor stored in the BIOS register or table (MP Specification, 75, 79; IA-32, 277; Fox, [0035]), and would have determined whether the failing processor is the boot processor through the boot processor's unique identification in the SMI.

101. Furthermore, a POSITA would have found it obvious that Fox's BIOS would have determined whether the failing processor is the boot processor to ensure performance of boot processor responsibilities during system runtime and improve system reliability. It was known that besides functioning like an application processor, the boot processor controls various system functions after the initial boot process. For example, during system runtime, the boot processor controls system hardware, including AP startup and shutdown, and is kept as the last running processor during system shutdown. MP Specification, 75, 79. A POSITA would have known that the boot processor would also perform BIOS runtime services, test other application processors, manage the system state, check

which processors are available and not available, control system power down, perform system self-test, etc. When a failure is detected in one of the application processors, the operating system would simply reallocate workloads to other application processors based on the active processor list (Fox, [0035]). But when a failure is detected in the boot processor, the boot processor should be promptly recovered to ensure continued performance of the boot processor functions during system runtime. MP Specification, 75 (“The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the BSP operating as the last running processor during system shutdown.”), 79. Thus, a POSITA would have been motivated to determine the failing processor is the assigned boot processor to promptly replace the boot processor with the spare processor. This would allow the responsibilities of the boot processor to continue to be performed during system runtime. A POSITA would have understood such prompt replacement would improve the reliability of the system by maintaining the uptime of proper functions of the boot processor, reduce system interruption during system runtime, and improve system reliability during system runtime, shutdown, reset, and restart.

102. A POSITA would have expected success in determining whether the failing processor in Fox is the boot processor. For example, as discussed above,

Fox's SMI's identification of the failing processor to the BIOS and the operating system would have determined the failing processor is the assigned boot processor based on the unique identification of the boot processor (e.g., APIC ID or logical APIC ID). Furthermore, a POSITA would have understood that the BSP flag of the assigned boot processor would be readily examined by the BIOS and operating system for determining whether the failing processor is the boot processor. IA-32, 238, 276. Indeed, a POSITA would have been motivated to determine whether the failing processor is the designated boot processor such that the system would know whether to set the BSP flag of the spare processor to 1 during the processor switching. Such determination and BSP flag setting would ensure a processor performs the responsibilities of the boot processor during system runtime and improve system reliability.

3. A POSITA would have found it obvious to incorporate Arai's processor-switching implementation details in the Fox-Safford system to minimize system interruption during switching.

103. Fox discloses that processor identifications are saved in a register or table (e.g., BIOS table, ACPI table, MP table), and all interrupt transactions include processor identifications. Fox, [0032]-[0033], [0035], [0038]. Fox does not explicitly disclose how the processor identifications would be used in the switching process. Arai discloses such implementation detail.

104. Arai discloses switching a failing processor with a spare processor by switching the logical identifications of the failing processor and the spare processor and transferring the context of the failing processor to the spare processor. Arai, [0020]-[0022]. Indeed, such implementation detail for processor switching was common and well known. *See* Section V.C; Slegel, 3:22-40 (teaching the spare processor assuming the logical identifier of a checkstopped processor is critical to make the processor sparing operation appear transparent to the operating system), 6:16-43 (teaching the spare processor to assume the role of the checkstopped processor loads the micro-architected state of the checkstopped processor). Incorporating Arai's teaching into Fox's system as modified by Safford simply involves the addition of a well-known and suitable implementation detail in Fox's processor-switching process.

105. A POSITA would have been motivated to incorporate Arai's teaching of transferring processor context from the failing processor to the spare processor not to interrupt the operating system and its running applications as desired by Fox. Fox teaches that during the SMI, the system state of each processor is completely saved to memory, the caches and pipelines are flushed, and the BIOS switches the memory save state from the old processor (i.e., the failing processor) to the new processor (i.e., the spare processor). Fox, [0041]. Similarly, Arai teaches

transferring the execution context from the active processor that is experiencing the error to the spare processor. Arai, [0022], [0025]. The execution context may include information in the processor registers and other data that supports communication between the processor and the operating system, such as processor identification and instructions. *Id.*, [0022]; Slegel, 6:21-43 (teaching a processor's micro-architecture state includes processor physical ID and instructions). A POSITA would have understood that both the identification of the failing processor and its context/state would be transferred to the spare processor such that the instructions and processes previously performed in the failing processor can be continued in the spare processor without system interruption. This would allow the role of the failing processor to be switched to the spare processor.

106. A POSITA would have understood that incorporating Arai's processor-switching implementation details into Fox-Safford would provide the benefit of switching the role of the failing processor to the spare processor without affecting the operating system and running applications, which is expressly taught as desirable by both Fox and Arai and well known in the art. Fox, [0008], [0013], [0036] (allowing the operating system to continue as normal despite a failing processor during dynamic replacement of the failing processor); Arai, [0005] (teaching "it is very desirable not to lose the work being done" on the processor

that encounters an error), [0006] (“There is therefore a need to provide a method and system for transparently reassigning responsibilities of a failed processor to a reserved processor without a modification to the operating system.”); Slegel, 3:7-20 (moving state of failed processor to spare processor in the system transparently and dynamically); *see* Section V.C. A POSITA would have understood that such transparent switching would improve availability of the multiprocessor system in Fox as modified by Safford because the processes of the operating system would continue despite processor failure (e.g., loss of lockstep).

107. A POSITA would have understood how to implement Arai’s processor-switching details in Fox as modified with Safford’s lockstep processors and error detection logics and expect success. Both Fox and Arai disclose or suggest:

- replacing a failing processor with a spare processor in a multiprocessor system (Fox, Abstract, [0014]-[0015]; Arai, Abstract, [0002], [0008]-[0009], [0019]);
- using registers or tables to store processor information available to the OS and BIOS and use the stored information to implement the switching (Fox, [0033], [0035], [0038], [0041], [0045]-[0046], Claims

2, 13; Arai, [0019]-[0022], [0025]-[0026], Claims 3, 9-11, 18-19, FIG. 1);

- using a BIOS interrupt handler for responding to processor error and implementing the processor switching or replacement (Fox, [0024], [0038]-[0039], [0041]-[0042]; Arai, [0022]-[0024]);
- using APIC identification for identifying processors (Fox, [0026]; IA-32, 244, 273-274; Arai, FIGS. 2, 4a-4b, 5a-5b (showing logical APIC ID));
- saving the state or context of the failing processor and transferring it to the spare processor (Fox, [0041]; Arai, [0022], [0025]); and
- using firmware to designate the processor roles during system initialization, identify the failing processor, and replace the failing processor with the spare processor (Fox, [0040]-[0042]; IA-32, 244; MP Specification, 75, 79; Arai, [0020], [0022]-[0024], [0026]).

108. Accordingly, Fox's multiprocessor includes the hardware (e.g., APIC structure) and firmware (e.g., BIOS, BIOS interrupt functions and handlers, BIOS table, ACPI table, or MP table) used by Arai's processor-switching process.

Implementing Arai's processor-switching details in Fox as modified by Safford would have been applying a known technique to a similar device in the same way

and achieve predictable results of processor switching with minimal disruption to the system as desired by Fox and known in the art. Fox, [0008], [0010], [0013], [0036]; *see* Section V.C.

C. Independent Claim 1

109. In my opinion, Fox in view of Safford and Arai renders obvious claim 1.

1. [1preamble]: “A method comprising:”

110. If the preamble is limiting, Fox discloses “a method ... for dynamically replacing a failing processor” with a spare processor. Fox, Abstract, [0002], [0014], [0023], [0046].

2. [1a]: “detecting loss of lockstep (LOL) for a processor in a multi-processor system;”

111. Fox in view of Safford renders obvious [1a].

112. Fox discloses a multiprocessor computer system configured with IA-32 architecture, the system including a spare processor and a plurality of main processors. *Id.*, [0019], [0023]-[0025], FIG. 1. Fox does not expressly disclose that its processors are lockstep processors or detecting loss of lockstep (LOL). Safford does.

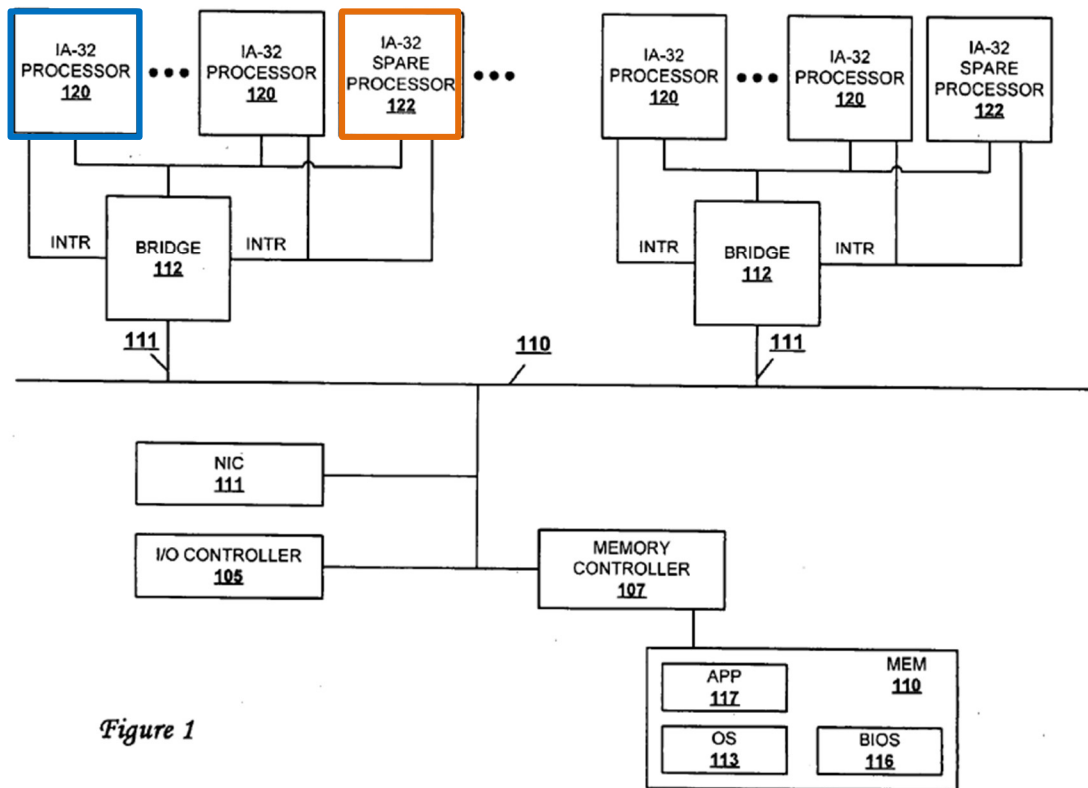


Figure 1

Id., FIG. 1 (annotated).

113. Safford discloses a multiprocessor computer system 100, wherein each of the processors includes a pair of processor cores 111 and 113 operating in lockstep. Safford, Abstract, [0015], FIGS. 2-3. Safford's lockstep processors include a logic circuit 115 to detect differences in the outputs of the pair of processors indicating a loss of lockstep and error detection, and signaling logic 112 and 114 to detect errors in the respective processor cores to signal an impending loss of lockstep. *Id.*, [0015]-[0017]. Safford also discloses a fault recovery process that replaces the lockstep pair of processors 111 and 113 with a spare processor

pair in response to detection of a loss of lockstep error by the error detection logics. *Id.*, [0007], [0018], [0021], FIGS. 3-4.

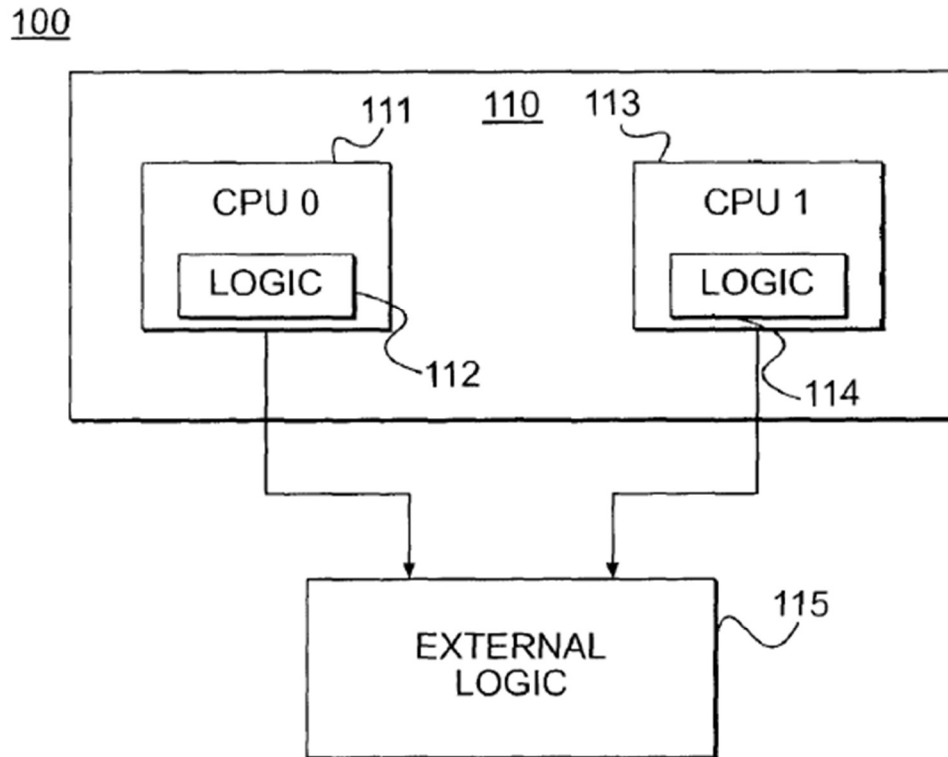


FIG. 2

Id., FIG. 2.

114. As explained in Section IX.B.1, a POSITA would have been motivated to implement Fox's processors as lockstep processors with error detection logics as taught by Safford to improve the reliability of Fox's server system and expected success. *Id.*, [0003]. A POSITA would have understood that when Fox's processors are implemented as lockstep processors with error detection

logics as taught in Safford, detecting a processor failure would have included “detecting loss of lockstep (LOL).” *See* Section IX.B.1.

3. [1b]: “determining that the processor for which said LOL is detected is assigned a role of boot processor;”

115. Fox in view of Safford renders obvious [1b].

116. As discussed in Section IX.B.2, a POSITA also would have understood that Fox’s system would have designated a boot processor, and Fox’s SMI’s identification of the failing processor to the BIOS would have determined that the failing processor is the designated boot processor (“assigned a role of boot processor”). Fox, [0042]. Also, a POSITA would have understood that Fox’s BIOS would have determined whether the failing processor is the designated boot processor to ensure continued performance of boot processor responsibilities during system runtime and improve system reliability.

117. As discussed in [1a] and Section IX.B.1, in Fox, as combined with Safford’s lockstep processors and error detection logics, detecting a processor failure would have included “detecting loss of lockstep (LOL).” Thus, in Fox combined with Safford, Fox’s determining that the failing processor is assigned a role of boot processor would be “determining that the processor for which said LOL is detected is assigned a role of boot processor.”

4. **[1c]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said boot processor for which said LOL is detected, to a different value, which causes the system’s operating system to not recognize the processor for which said LOL is detected as the boot processor; and”**

118. Fox in view of Safford and Arai renders obvious [1c].

119. As discussed for [1a]-[1b], Fox in view of Safford renders obvious detecting a loss of lockstep for a processor, and determining that the processor for which said LOL is detected is assigned a role of boot processor. A POSITA would have further understood that in Fox combined with Safford, Fox’s spare processor would be a lockstep pair and used for replacing the boot processor for which said LOL is detected. *See also* Sections IX.B.1-IX.B.2.

120. Fox also discloses that processor identifications are saved in a register or table (e.g., BIOS table, ACPI table, MP table) and including processor identifications in all interrupt transactions. Fox, [0026], [0033], [0035], [0038], [0041], [0045]-[0046], Claims 2, 13. All of these were well known for IA-32 multiprocessor systems. IA-32, 273-274, 267, 269, 289-297, 709-727. Indeed, it was known that all processors in an IA-32 multiprocessor system, like that in Fox, would have their unique identifications assigned during the system initialization; the identifications are stored in the ACPI table and MP table created by the boot processor and accessible to the BIOS and operating system. *Id.*, 244, 256 (e.g.,

local APIC ID or logical APIC ID); MP Specification, 38, 69, 72-73, 76. A POSITA would have understood that Fox would have used the processor identifications for implementing the switching of the failing processor to the spare processor. *See* Section V.C. Furthermore, Arai discloses such implementation detail.

121. Arai discloses replacing an active processor detected with an error with a reserved inactive processor in a multiprocessor system. Arai, Abstract, [0002], [0008]-[0009], [0019]-[0024]. Arai discloses that during system initialization, the BIOS creates a table that identifies each processor in the system and its designated operational state, i.e., active or disabled. *Id.*, [0019]-[0022]. If the BIOS determines the error requires replacement of the processor, and if there is reserved inactive processor, the identification numbers of the active processor in error and the reserved inactive processor are switched. *Id.*, [0022]-[0023], [0025]. For example, Arai discloses switching the logical APIC ID of the active processor in error with that of the reserved processor. *Id.*, [0023]-[0024], FIGS. 4a-4b. A POSITA would have understood that Arai's logical APIC IDs would be used by Arai's operating system to recognize the processor in error, and thus constitutes "a system identifier." *Id.*, [0020]-[0022], [0025]-[0026]. A POSITA would have further understood that such switching would have changed the logical APIC ID

(“system identifier”) of the processor in error to the logical APIC ID of the reserved processor (“a different value”). *Id.*

122. In Fox combined with Safford and Arai, a POSITA would have understood that such identification switching would have switched the logical identification, e.g., logical APIC ID (“system identifier”), of the lockstep processor pair of the boot processor detected to have LOL with that of the spare processor pair. *See* Safford, [0021] (teaching the spare processor becomes the new logical CPU 0). A POSITA would have understood that such switching would have “cause[d] the system’s operating system to not recognize the processor for which said LOL is detected as the boot processor.” This is because the unique logical identification of the boot processor having LOL as it was known to the BIOS and operating system has been assigned to that of the spare processor. The operating system would have instead recognized the spare processor now assigned with the unique logical identification of the boot processor as the boot processor.

123. As discussed in Section IX.B.3, a POSITA would have been motivated to implement Arai’s switching processor identifications (“changing a system identifier ... to a different value”) in Fox as modified by Safford and expected success because it simply involves the addition of a well-known and suitable implementation detail to Fox’s processor-switching process.

5. **[1d]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to a spare processor without shutting down the system’s operating system.”**

124. Fox in view of Safford and Arai renders obvious [1d].

125. As described above, Arai discloses switching the identification numbers of the active processor in error and the reserved inactive processor. Arai, [0022]-[0023], [0025]. In Fox combined with Safford and Arai, a POSITA would have understood that such identification switching would have switched the logical identification of the boot processor lockstep pair with that of the spare processor pair. A POSITA would have understood that such switching would have changed the logical identification, e.g., logical APIC ID, of the spare processor to that of the boot processor (“changing a system identifier of the spare processor to that of the boot processor”).

126. Further, Fox discloses that the BIOS switches the memory save state from the failing processor to the spare processor. Fox, [0041]. Similarly, Arai discloses “transfer[ring] of the execution context from the active processor that is experiencing the error to at least one of the inactive processors that has been designated to replace the active processor.” Arai, [0022], [0025], FIG. 4b. Arai further discloses that “[t]he execution context may include information in the processor registers and other data that supports communication between the

processor and the operating system.” *Id.*, [0022]. As discussed in Section IX.B.3, a POSITA would have been motivated to incorporate Arai’s teaching of transferring the execution context in Fox’s switching process, as Fox suggested, to allow the processes previously performed by the failing processor to be continued by the spare processor without interrupting the operating system. Fox, [0008], [0013], [0036]; Arai, [0006]; Slegel, 3:6-20; *see also* Section V.C.

127. A POSITA would have understood that in Fox as modified by Safford, in response to determining that the LOL is detected in the boot processor, implementing Arai’s identification switching and context transfer in the switching process would have reassigned the processor logical identification and responsibilities of the failing boot processor pair to the spare processor pair without interrupting the operating system (“switching the role of boot processor to a spare processor without shutting down the system’s operating system”). Fox, [0008], [0010], [0013]; Arai, [0006].

128. As discussed in Section IX.B.3, a POSITA would have understood that incorporating Arai’s processor-switching implementation detail would improve availability and reliability of the multiprocessor system in Fox as modified by Safford. A POSITA would have expected success in such

modification because it merely involves using known techniques to improve a similar device in the same way taught in the prior art.

X. Ground 2: Fox in View of Safford, Landry, and Arai Renders Obvious Claim 1

129. In my opinion, Fox in view of Safford, Landry, and Arai renders obvious claim 1.

A. Fox in View of Safford, Landry, and Arai

130. As discussed above, the combination of Fox, Safford, and Arai renders obvious every element of claim 1. For reasons discussed in Section X.B.2, a POSITA would have understood that Fox’s system would have, or it would have been obvious to have, designated a boot processor and used the spare processor to replace the boot processor in response to detecting failure in the boot processor. And as discussed in Ground 1, 1[b] (*see* Section IX.C.3), Fox combined with Safford’s lockstep processors and error detection logics render obvious “determining that the processor for which said LOL is detected is assigned a role of boot processor.” This is also rendered obvious further in view of Landry.

B. A POSITA Would Have Found It Obvious to Combine Fox, Safford, Landry, and Arai

131. A POSITA would have found it obvious to combine Fox, Safford, Landry, and Arai for the reasons discussed in Ground 1, Section IX.B, and the additional reasons below.

1. A POSITA would have found it obvious to identify Fox's boot processor as logical CPU0 as taught in Landry.

132. A POSITA would have understood that Fox's system would have, or it would have been obvious to have, designated a boot processor. *See* Section IX.B.2. Indeed, Fox's multiprocessor system configured with IA-32 architecture follows the MP initialization protocol that dynamically designates one of its processors as the boot processor. Fox, [0014]; IA-32, 238; *see* Section V.B. Landry discloses a fault-tolerant boot process that dynamically designates a functional CPU as logical CPU0 that performs the unique functions of a boot processor. Landry, Abstract, 1:6-10, 2:9-41, 6:44-7:42, Figs. 1, 4; *see* Section VII.D. A POSITA would have understood Landry's process would have been a suitable option for selecting a boot processor in Fox's system consistent with the initialization protocol of the IA-32 multiprocessor system.

133. A POSITA would have understood the benefits of implementing Landry's boot processor selection process in Fox. First, implementing Landry's process in Fox would improve system availability by allowing the system to power up despite initially starting on a non-functional processor. Landry, 1:47-62, 2:9-41. A POSITA would have further understood that using the logical CPU0 as the identification for the boot processor allows for transparent replacement of the physical processor experiencing a failure with another processor without

interrupting the operating system. Slegel, 3:7-46. A POSITA also would have understood that logical CPU0 would have been a known and conventional identification of the boot processor. *See* Section V.B; Landry, Abstract, 1:6-10, 2:9-41, 6:44-7:52, Figs. 1, 4; Cepulis, Abstract, 2:30-64; Jin, [0048].

134. A POSITA would have expected success in implementing Landry's process in Fox because Fox's IA-32 system provides the necessary hardware, firmware, and data structures for implementing the processor selection. Also, identifying the boot processor as logical CPU0 would only require ordinary computer programming skills to assign logical CPU0 as the unique identification of the boot processor in Fox's system and store such unique identification in Fox's table that is accessible to the BIOS and operating system. *See* Fox, [0026], [0033], [0035], [0038], [0041], [0045]-[0046]; IA-32, 294 (e.g., logical APIC ID); MP Specification, 75, 79.

2. A POSITA would have found it obvious to switch logical CPU0 (boot processor) to the spare processor in response to detecting loss of lockstep in the logical CPU0, as taught in Safford.

135. Safford discloses a lockstep pair of processors appearing to the system as a single logical CPU0, and in response to detecting loss of lockstep in the lockstep pair, switching logical CPU0 to a spare pair of processors. Safford, [0015], [0018]-[0021]. A POSITA would have found it obvious to switch Fox-

Landry's logical CPU0 (boot processor) to the spare processor to improve system availability and reliability. *See* Section IX.B.2. A POSITA would have expected success in implementing the switching of logical CPU0 to the spare because such switching only requires ordinary computer programming skills to assign unique logical identifications to processors in Fox's system and update Fox's processor register or table storing the processor's identifications, as known in the art. *See* Fox, [0015], [0026], [0033], [0035], [0038], [0040]-[0041], [0045]-[0046]; Arai, [0019]-[0021], [0025]-[0026]; IA-32, 294 (e.g., logical APIC ID); MP Specification, 75, 79.

C. Independent Claim 1

136. In my opinion, Fox in view of Safford, Landry, and Arai renders obvious claim 1.

1. [1preamble]: "A method comprising:"

137. Fox discloses [1preamble] for the reasons explained in Ground 1, limitation [1preamble].

2. [1a]: "detecting loss of lockstep (LOL) for a processor in a multi-processor system;"

138. Fox in view of Safford renders obvious [1a] for the reasons explained in Ground 1, limitation [1a].

3. [1b]: “determining that the processor for which said LOL is detected is assigned a role of boot processor;”

139. Fox in view of Safford and Landry renders obvious [1b].

140. As explained in Sections IX.B.1 and IX.C.2(Ground 1, limitation [1a]), a POSITA would have found it obvious to implement the processors in Fox’s multiprocessor systems as lockstep processors with error detection logics and detect loss of lockstep between a lockstep processor pair as a processor failure to improve system reliability. And as explained in Ground 1, limitation [1b], Fox combined with Safford renders obvious detecting loss of lockstep (LOL) for a processor implemented as a pair of lockstep processors in Fox’s multiprocessor system.

141. Fox in view of Safford and Landry further renders obvious determining that the processor for which said LOL is detected is assigned a role of boot processor.

142. Safford discloses a multiprocessor system having a plurality of lockstep processors pairs, 111/113, 121/123, and 125/127, where each pair appears as a single logical processor to the system. Safford, [0015], [0018]. Safford discloses processor pair 111/113 is **logical CPU0** and processor pair 125/127 is designated as a **spare processor pair**. *Id.*, [0015], [0020]-[0021]. When a loss of lockstep error is detected in processor pair 111/113 (logical CPU0), the spare

processor pair 125/127 replaces processor pair 111/113 and becomes the logical CPU0. *Id.*, [0021], FIGS. 3-4. Thus, Safford discloses determining that the processor for which said LOL is detected is logical CPU0 as compared to other logical CPUs of the system such that the logical CPU0 would be switched to the spare processor pair.

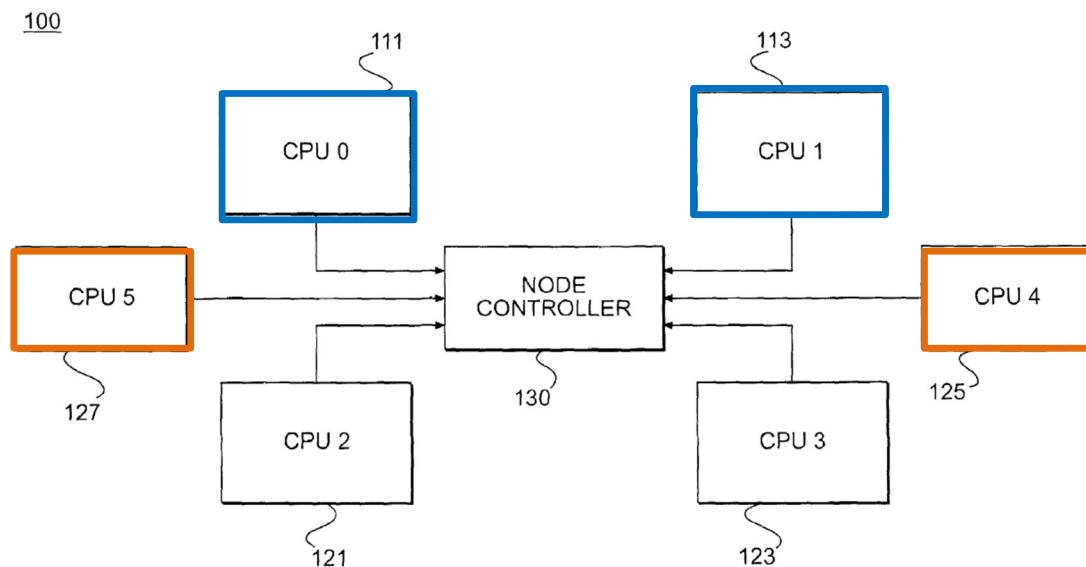


FIG. 3

Id., FIG. 3 (annotated).

143. As explained above in Section X.B.1, a POSITA would have understood that Fox's system would have, or it would have been obvious to have, a boot processor, and would have implemented Landry's boot processor selection process in Fox and identified logical CPU0 in Fox as the boot processor. Thus, Fox combined with Safford and Landry, would have determined that the processor for

which said LOL is detected is logical CPU0, which is the processor assigned a role of boot processor.

4. **[1c]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said boot processor for which said LOL is detected, to a different value, which causes the system’s operating system to not recognize the processor for which said LOL is detected as the boot processor; and”**

144. Fox in view of Safford, Landry and Arai renders obvious [1c] for the reasons discussed in Ground 1, limitation [1c], and the additional reasons discussed above in Ground 2, limitation [1b]. A POSITA would have understood that in Fox combined with Safford, Landry, and Arai, the system identifier used by the operating system to recognize the boot processor would be the identifier of logical CPU0.

5. **[1d]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to a spare processor without shutting down the system’s operating system.”**

145. Fox in view of Safford, Landry and Arai renders obvious [1d] for the reasons discussed in Ground 1, limitation [1d], and the additional reasons discussed above in Ground 2, limitation [1b]. A POSITA would have understood that in Fox combined with Safford, Landry, and Arai, the system identifier of the

spare processor would be changed to the identifier of logical CPU0 to replace the boot processor that has lost lockstep.

XI. Fox in View of Safford and Arai (Ground 1) or Fox in View of Safford, Landry, and Arai (Ground 2) Renders Obvious Claims 2-25

146. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious claims 2-25.

A. Independent Claim 13

147. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious claim 13.

1. [13preamble]: “A method comprising:”

148. Fox discloses [13preamble] for the reasons explained in Grounds 1-2, limitation [1preamble].

2. [13a]: “system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system;”

149. Limitation [13a] is similar to limitation [1b] except additionally reciting “system firmware.” Fox in view of Safford, or alternatively, Fox in view of Safford and Landry, renders obvious [13a].

150. As explained above in Grounds 1-2, limitation [1b], Fox in view of Safford or Fox in view of Safford and Landry renders obvious “determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are

assigned the role of boot processor in a system.” Fox in view of Safford or Fox in view of Safford and Landry further renders obvious that the BIOS (“system firmware”) performs this determination.

151. Fox discloses using the BIOS of its IA-32 multiprocessor system to implement switching of a failing processor to the spare processor. Fox discloses when a failure is detected in one of its main processors, the processor failure response logic of the BIOS generates an SMI (System Management Interrupt) in response to detecting the processor failure to facilitate the processor replacement. Fox, [0015], [0032]. The SMI identifies to the OS and the BIOS the processor that is failing, and the BIOS, utilizing the SMI handler, takes control of the system and replaces the failing processor with the spare processor. *Id.*, [0041]-[0042], [0048].

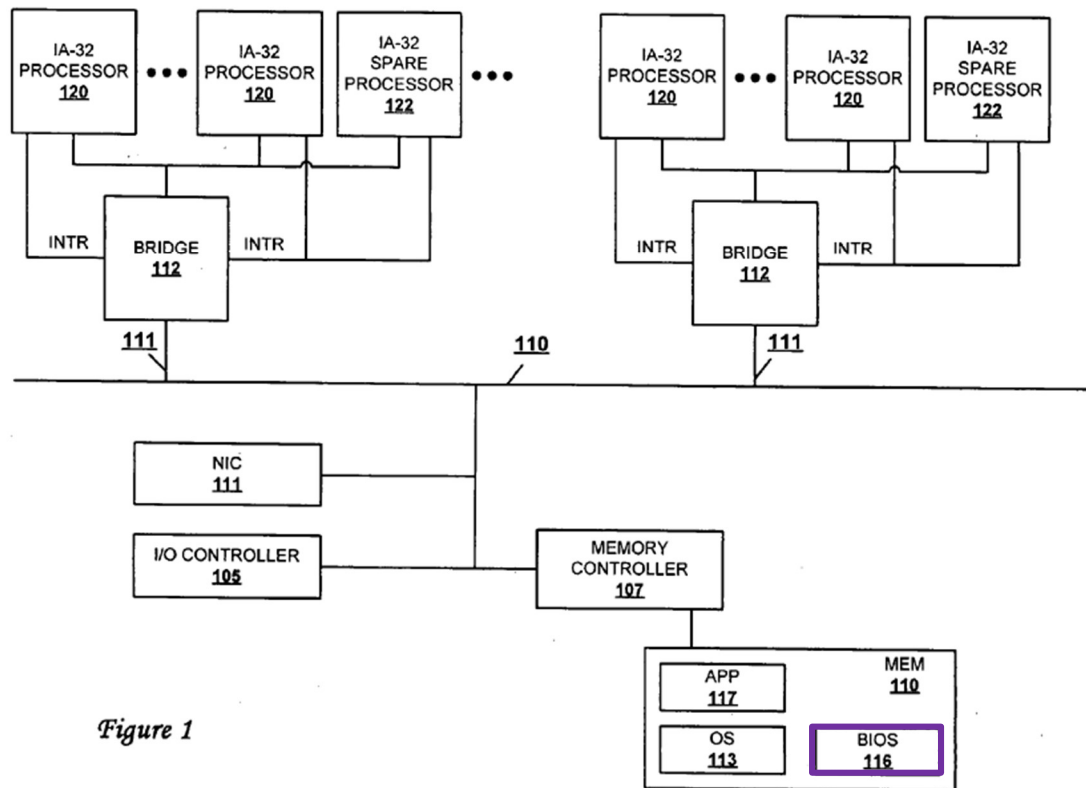


Figure 1

Id., FIG. 1 (annotated).

152. And as discussed in Ground 1, limitation [1b] and Section IX.B.2, a POSITA would have understood that the identification of the failing processor by SMI to the BIOS would have determined that the failing processor is the designated boot processor (“assigned the role of the processor”) because such identification would be unique for the boot processor and known to the BIOS. *Id.*, [0042]; MP Specification, 75, 79. Furthermore, it would have been obvious to a POSITA that Fox’s or Fox-Landry’s BIOS would have determined whether the failing processor is the boot processor to ensure performance of boot processor

responsibilities during system runtime and thus improve system reliability. *See* Sections V.B, IX.B.2, and X.B.2. And in Fox combined with Safford or Safford and Landry, Fox’s determining that the failing processor is assigned a role of boot processor would be “determining that the processor for which said LOL is detected is assigned a role of boot processor.” *See* Sections IX.C.3, X.C.3. A POSITA would have understood that Fox’s BIOS is “system firmware” that acts as an interface between the system’s hardware and operating system. *See* Section V.D. Thus, Fox in view of Safford or Safford and Landry renders obvious “system firmware determining that the processor for which said LOL is detected is assigned a role of boot processor.”

3. [13b]: “determining one of said lockstep pair of processors that is not the cause of said LOL;”

153. Fox in view of Safford renders obvious [13b].

154. For reasons explained in Section IX.B.1 and Grounds 1-2, limitation [1a], a POSITA would have found it obvious to implement the processors in Fox’s multiprocessor systems as lockstep processors with error detection logics as taught in Safford to improve the reliability of Fox’s server system. And a POSITA would have understood that when Fox’s processors are implemented as lockstep processors with error detection logics as taught in Safford, detecting a processor failure would have included “detecting loss of lockstep (LOL)” in a lockstep pair

of processors. Fox combined with Safford further discloses determining one of the lockstep pair of processors that is not the cause of said LOL.

155. Safford discloses that a lockstep processor pair includes a logic circuit 115 to detect differences in the outputs of the pair of processors indicating a loss of lockstep, and **error detection and signaling logics 112 and 114** to detect errors in the respective processor cores to signal an impending loss of lockstep. Safford, [0016]-[0017]; *see* Section VII.B.

100

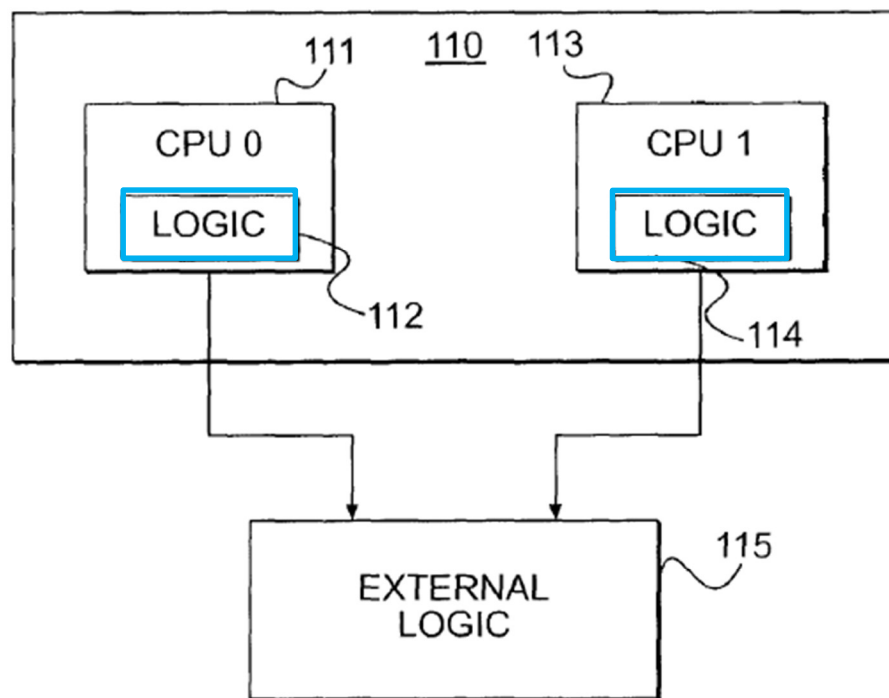


FIG. 2

Safford, FIG. 2 (annotated).

156. The error detection and signaling logics 112 and 114, by detecting errors in the individual processors separately, would signal to the logic circuit 115 which processor is experiencing the error that will cause a loss of lockstep and that the processor is a “bad” processor. *Id.*, [0017]. Logic circuit 115 may turn off and processing may continue using the other “good” processor. *Id.* Safford’s error detection logics 112, 114, and 115 together determine the “bad” processor as the processor that is experiencing an error and causing a loss of lockstep, and determine the “good” processor as the processor that is not the cause of the loss of lockstep. *Id.*; *see* Section IV.C.1.

157. Safford further discloses saving the architected state of the “good” processor of the lockstep pair of processors and copying the architected state of the lockstep pair of processors that has lost lockstep to the hot standby processor pair. Safford, [0005], [0017], FIG. 4; *see also id.*, [0018], [0020]-[0021]. Similarly, Fox also discloses, during an SMI, saving the system state of each processor to memory, flushing the caches and pipelines, and switching the memory save state of the failing processor to the spare processor. Fox, [0041]. In Fox combined with Safford’s lockstep processors and error detection logics, a POSITA would have been motivated to determine which processor is not the cause of loss of lockstep (e.g., Safford’s “good” processor) such that the state of the processor without error

would be switched to the spare processor as taught by Fox and Safford and improve reliability in the processor recovery. *See* Nguyen, [0046] (teaching localizing error to one of the execution cores to allow the machine state of the good core to be saved and used for recovery). A POSITA would have expected success in implementing Fox's processors as lockstep processors with error detection logics as taught in Safford. *See* Section IX.B.1.

4. [13c]: “copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to a spare processor;”

158. Fox in view of Safford renders obvious [13c].

159. As discussed in [13b], Fox in view of Safford discloses the “good” processor of the lockstep pair of processors “that is not the cause of said LOL.” Fox in view of Safford further renders obvious “copying the state” of the “good” processor to the spare processor.

160. Fox discloses the BIOS switches the saved memory state of the failing processor to the spare processor during processor switching. Fox, [0041].

161. Safford discloses copying the state of “good” processor that is not the cause of said LOL to a spare processor. As described above for element [13b], Safford discloses saving the architected state of the “good” processor and copying the architecture state of the lockstep processor pair 111/113 experiencing loss of

lockstep to the hot standby processor pair 125/127 (“a spare processor”), as shown below. Safford, [0005], [0017]-[0018], [0020]-[0021], FIG. 4.

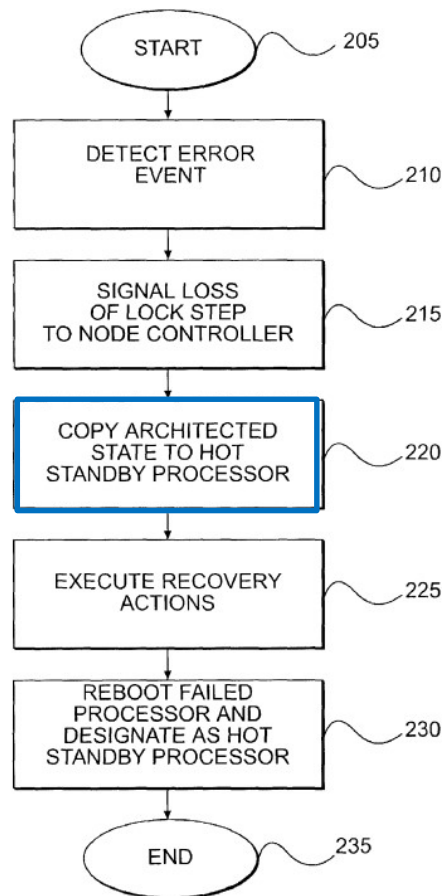


FIG. 4

Id., FIG. 4 (annotated).

162. As described above in [13b], in Fox combined with Safford’s lockstep processors and error detection logics, a POSITA would have been motivated to determine which processor is not the cause of loss of lockstep (e.g., Safford’s “good” processor) and copy the state of this processor to the spare processor as

taught by Fox and Safford to improve reliability in the processor recovery. *See* Nguyen, [0046] (teaching saving the machine state of the good processor core of a lockstep processor pair to use the saved machine state to recover the processor cores); *see* Arai, [0005] (teaching “it is very desirable not to lose the work being done” on the processor that encounters an error).

5. **[13d]: “changing a system identifier, used by the system’s operating system to recognize the boot processor, of said lockstep pair of processors that are assigned the role of boot processor, to a different value, which causes the system’s operating system to not recognize the lock step pair of processors for which said LOL is detected as the boot processor; and”**

163. Element [13d] is substantially similar to [1c]. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious [13d] for the reasons explained in Grounds 1-2, limitation [1c].

6. **[13e]: “changing a system identifier of the spare processor to that of the boot processor, thereby switching the role of boot processor to said spare processor.”**

164. Element [13e] is substantially similar to [1d]. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious [13e] for the reasons explained in Grounds 1-2, limitation [1d].

B. Independent Claim 21

165. In my opinion, Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious claim 21.

1. [21preamble]: “A system comprising:”

166. If the preamble is limiting, Fox discloses “a system for dynamically replacing a failing processor” with a spare processor. Fox, Abstract, [0002], [0014], [0023]-[0024].

2. [21a]: “a plurality of processor modules that each include a master processor and a slave processor that operate in lockstep;”

167. Fox in view of Safford renders obvious [21a].

168. Fox discloses a multiprocessor computer system configured with IA-32 architecture, the system including a spare processor and main processors (“a plurality of processor modules”), as shown below. *Id.*, [0019], [0023]-[0025]; *see* Section VII.A.

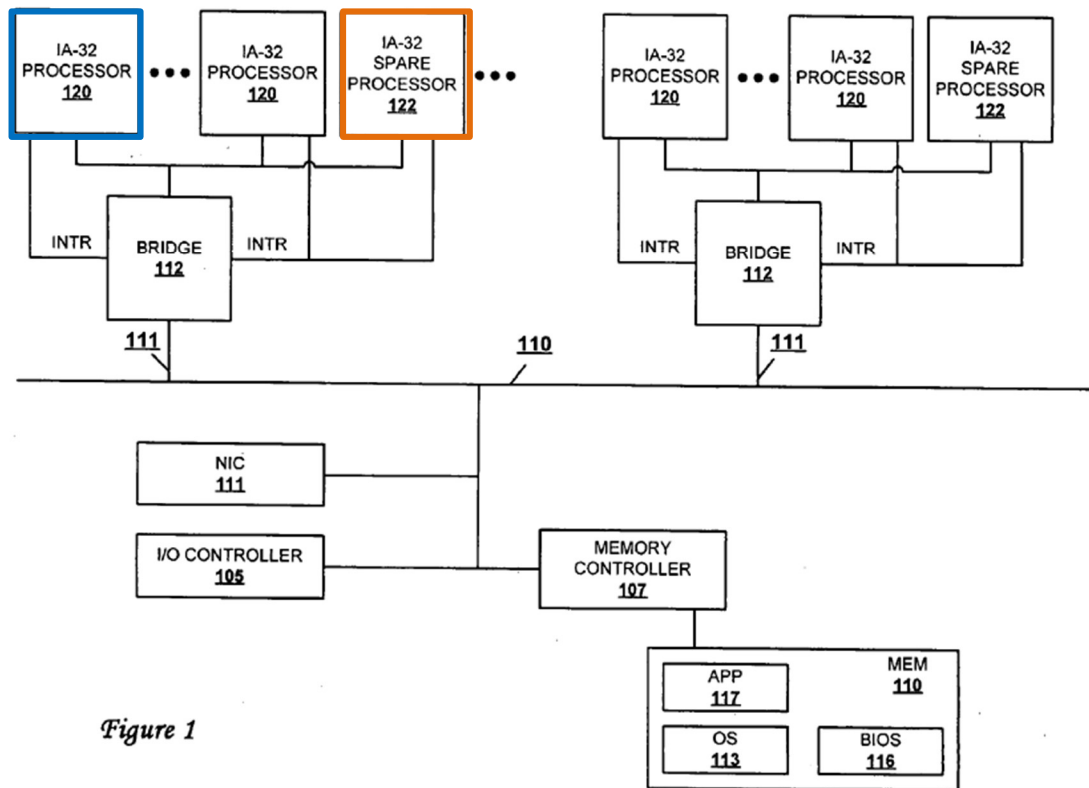


Figure 1

Fox, FIG. 1 (annotated).

169. As explained in Section IX.B.1, a POSITA would have found it obvious to implement Fox's processors as lockstep processors with error detection logics as taught in Safford to improve system reliability and availability. Safford further discloses its lockstep processor pair "include a master processor and a slave processor."

170. Safford discloses each of the processors 110 of a multiprocessor computer system 100 includes a pair of processor cores 111 and 113 operating in

lockstep. Safford, Abstract, [0015], [0018], FIGS. 2-3. Safford discloses the lockstep processor pair includes a master/checker pair. *Id.*, [0015].

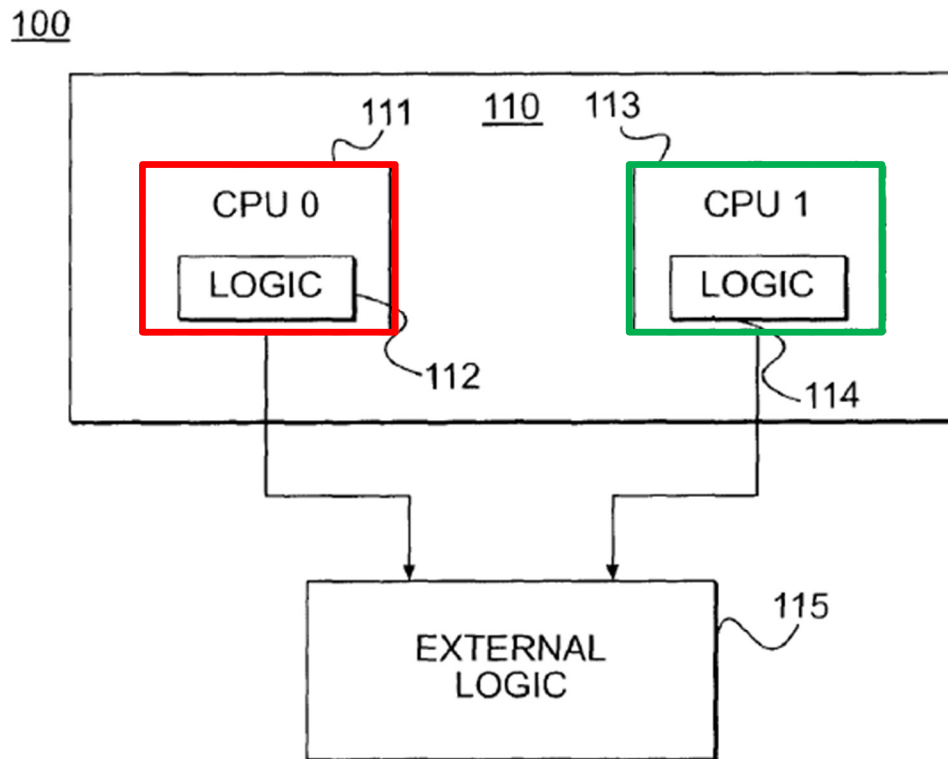


FIG. 2

Id., FIG. 2 (annotated).

171. Safford further discloses that “[e]ach of [the] two processors in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit,” and that difference in the processor outputs indicates loss of lockstep. *Id.*, [0015].

172. A POSITA would have understood Safford's lockstep processor's master/checker pair constitutes "a master processor and a slave processor that operate in lockstep."

173. It was common and well known to a POSITA that a lockstep pair of processors operates as a single processor to the system, where one of the processors ("master processor") writes data to the memory and output to the system and the other processor ("slave processor") performs redundant processing to check whether there is mismatch between the two processors indicating a processor error. *See* Kondo, [0094] ("[I]t is only the master processor unit 12 that is actually writing data to memory; the data 'written' by the shadow processor unit 14 is compared in the memory checker 16 with that written to memory 20 by the master processor unit 12."); Marshall, 1:11-20 (lockstep processor systems have "at least two processors (commonly designated as master and slave) processing independently, but in lockstep, the same task with their independently generated results compared in order to detect an error originating in one processor."), 1:25-32 ("One of the processors is declared as the master and is allowed to drive the outputs, while the other processor is the slave and is only allowed to receive on its outputs. The slave processor compares the master's outputs with its internal outputs to ensure that they are the same."); Nguyen, [0047] ("One of execution

cores 120 may be designated as the master core and the other as the slave core when processor 10 is operating in FRC mode.... [S]ignals generated by the master and slave cores are compared [S]ignals generated by the master core are transferred to shared resource(s) 170, and signals generated by the slave core are dropped.”).

174. A POSITA would have understood Safford’s master/checker pair operating in lockstep as the “master processor” and “slave processor” because the master/checker pair would perform the same functions as the commonly known master and slave processors. Further, a POSITA would also have found it obvious to designate Safford’s master/checker pair as a “master processor” and a “slave processor,” respectively, as was commonly known in the art. A POSITA would have understood that such designation would be simple and can be readily implemented. *See* Nguyen, [0027], [0054]-[0055] (teaching master and slave designation can be implemented using a status bit and the designations can be changed dynamically).

175. Additionally, for reasons explained in Section IX.B.1, a POSITA would have found it obvious to implement the processors in Fox’s multiprocessor systems as lockstep processors, each including a master processor and a slave

processor, with error detection logics as taught in Safford to improve the reliability of Fox's server system.

3. [21b]: “an operating system;”

176. Fox discloses [21b].

177. Fox's multiprocessor system includes a system memory 110, stored with software for controlling the system, the software including [operating system 113](#), as shown below. Fox, [0028], FIG. 1. Fox discloses the operating system is executed by the processors of the system and controls work allocation to the processors. *Id.*, [0032], Claim 12. Fox discloses the operating system continues to execute during switching of a failing processor to the spare processor. *Id.*, [0012]-[0016], [0023], [0032], [0044].

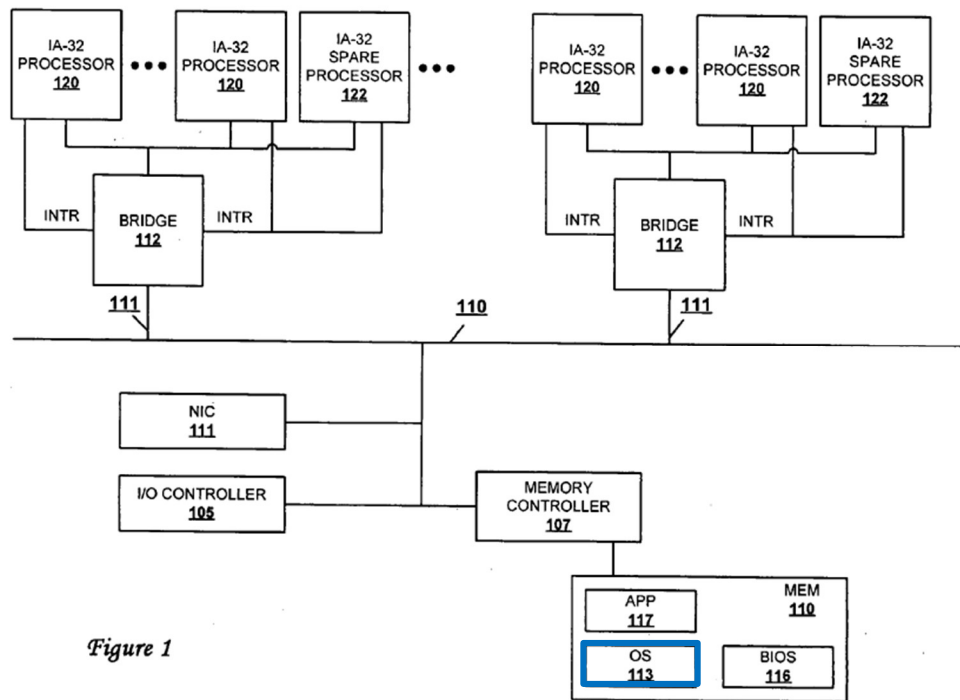


Figure 1

Id., FIG. 1.

4. [21c]: “error detection logic operable to detect loss of lockstep (LOL) for at least one of said processor modules; and”

178. Element [21c] is similar to [1a] except additionally reciting “error detection logic” to detect loss of lockstep. Fox in view of Safford renders obvious [21c] for the reasons explained in Grounds 1-2, limitation [1a] and the additional reasons below.

179. And as discussed in Section IX.B.1, a POSITA would have found it obvious to implement Fox’s processors as lockstep processors with error detection logics as taught in Safford to improve system reliability and availability. And as

discussed in Grounds 1-2, limitation [1a], Fox in view of Safford discloses detecting loss of lockstep (LOL) for a processor of the multiprocessor system (“for at least one of said processor modules”). In Fox combined with Safford, **error detection logics** of the lockstep processor pair would be used to detect loss of lockstep for the lockstep processor pair, as shown below.

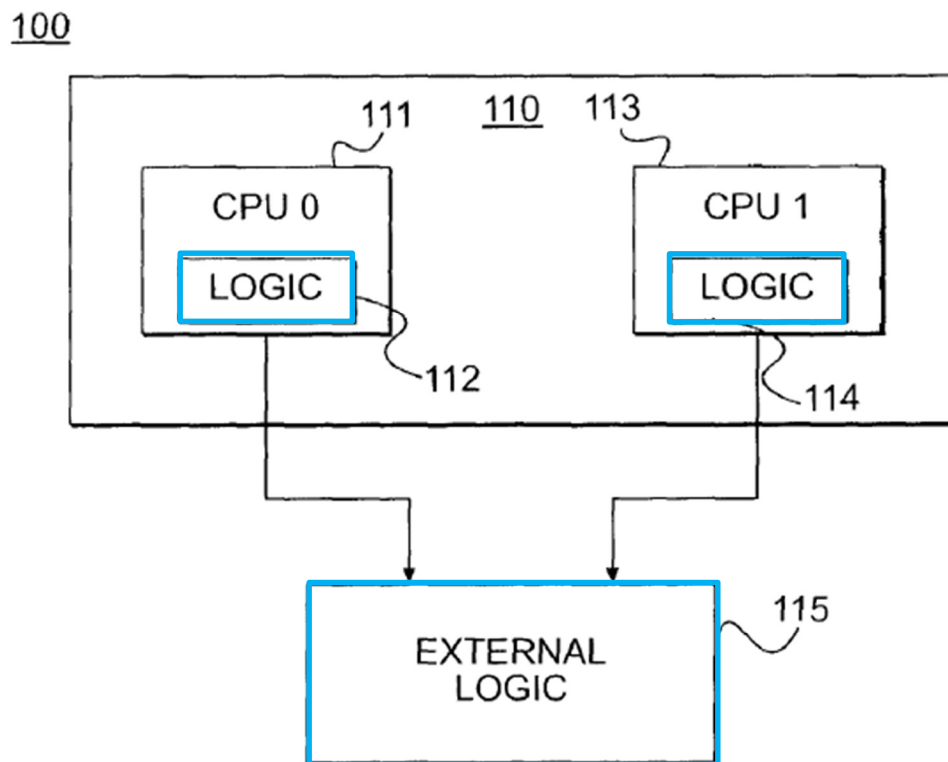


FIG. 2

Safford, FIG. 2; *see* Section VII.B.

180. A POSITA would have known that such error detection logics would be implemented in hardware, firmware, or a combination of hardware and

firmware. *See* Bigbee, [0034] (FRC logic may be implemented in firmware);

Safford-181, 15:60-64 (lockstep logic may be implemented in firmware).

5. [21d]: “system firmware operable, responsive to the detection of LOL for a first of said processor modules, to determine whether said first processor module is assigned a role of system boot processor;”

181. Limitation [21d] is substantially similar to [13a] except additionally reciting “responsive to the detection of LOL for a first of said processor modules.” Fox in view of Safford or Fox in view of Safford and Landry renders obvious limitation [21d] for the reasons discussed above in [13a] and the additional reasons discussed below.

182. As discussed above for [13a], Fox in view of Safford or Fox in view of Safford and Landry render obvious that Fox’s system firmware performs the determination that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system (“system firmware operable ... to determine whether said first processor module is assigned a role of system boot processor.”). Fox in view of Safford further renders obvious that the determination is responsive to the detection of LOL for a processor.

183. As shown in [13a], Fox discloses when a failure is detected in one of its main processors, the processor failure response logic of BIOS generates an SMI (System Management Interrupt) in response to detecting the processor failure to

facilitate replacement of the failing processor with the spare processor by the BIOS. Fox, [0015], [0032], [0041]-[0042], [0048].

184. As discussed in Grounds 1-2, limitation [1a], a POSITA would have understood that when Fox's processors are implemented as lockstep processors with error detection logics as taught in Safford, detecting a processor failure would have included "detecting loss of lockstep (LOL)." A POSITA would have understood that in Fox combined with Safford, the generation of SMI by the BIOS ("system firmware") to identify the failing processor and facilitate the processor switching by the BIOS would have been "system firmware operable, responsive to the detection of LOL."

6. **[21e]: "wherein if determined that said first processor module is assigned the role of system boot processor, said system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system."**

185. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious [21e] for the reasons explained above in Grounds 1-2, limitations [1b]-[1d], [13a], and [21d], and the additional reasons below.

186. As discussed in [13a], Fox in view of Safford or Fox in view of Safford and Landry renders obvious "system firmware" determining that loss of lockstep (LOL) is detected for a lockstep pair of processors ("first processor

module”) that are assigned the role of boot processor in a system. Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai further renders obvious that “system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system.”

187. As discussed in Grounds 1-2, limitation [1c], Fox discloses saving processor identifications in a register or table (e.g., BIOS table, ACPI table, MP table) accessible to the BIOS and operating system. Fox, [0026], [0033], [0035], [0038], [0041], [0045]-[0046], Claims 2, 13; IA-32, 273-274, 267, 269, 289-297, 745 (e.g., local APIC ID or logical APIC ID). Fox further discloses the register or table indicates currently available processors to the operating system. Fox, [0035], Claim 13; *see also* Arai, [0025] (ACPI table or MP table to identify location and status of each processor to the operating system).

188. And as discussed in Grounds 1-2, limitations [1c]-[1d], in Fox combined with Safford and Arai or Safford, Landry, and Arai, the logical identifications (e.g. logical APIC IDs) of the lockstep processor pair of the boot processor detected to have LOL (“first processor module is assigned the role of system boot processor”) and the spare processor pair would have been switched. *See* Safford, [0021] (teaching the spare processor becomes the new logical CPU 0).

This identification switching results in the unique logical identification of the boot processor (e.g., logical CPU0), as known to the operation system, to be assigned to the spare processor. Thus, a POSITA would have understood that such identification switching would have caused Fox's operating system to recognize the spare processor as the boot processor based on the unique logical identification of the boot processor ("cause said operating system to recognize another processor module as system boot processor").

189. A POSITA would have understood that Fox's BIOS ("system firmware") would have performed the switching of the processor identifications because: (1) the BIOS would assign processor configurations; and (2) both Fox and Arai disclose that the BIOS performs the processor switching without interrupting the operating system. Fox, [0035], [0038], [0041]-[0043]; Arai, [0004]-[0006], [0023], [0025].

190. Indeed, a POSITA would have understood that the processor switching by the BIOS of Fox's system as modified with Safford and Arai or Safford, Landry, and Arai would be performed "without shutting down said operating system" because it was well known that using firmware to switch processor logical identifications would allow the operating system to continue execution and thereby improve system availability despite processor failure. *See*

Section V.D; Slegel, 3:7-46, 6:16-36; Armstrong, Abstract, [0019]-[0020]; Zorek, Abstract, 2:61-67, 3:1-17; Fox, [0014], [0044], [0047].

C. Dependent Claims 2, 14, 23

1. Claims 2 and 14: “wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor.”

191. Claims 2 and 14 depend from claims 1 and 13, respectively.

192. As discussed in Grounds 1-2, limitations [1b] and [1d], Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious “determining that the processor for which said LOL is detected is assigned a role of boot processor” and “switching the role of boot processor to a spare processor.” Fox in view of Safford further renders obvious that the spare processor is “a hot spare that is held idle until switched.”

193. Fox discloses that at least one processor of the multiprocessor system is reserved as a hot-spare processor that remains in an idle, off, or low-power mode (“held in idle”), and while in that mode, the OS is prevented from initially utilizing the hot-spare processor. Fox, Abstract, [0015], [0040], FIG. 1.

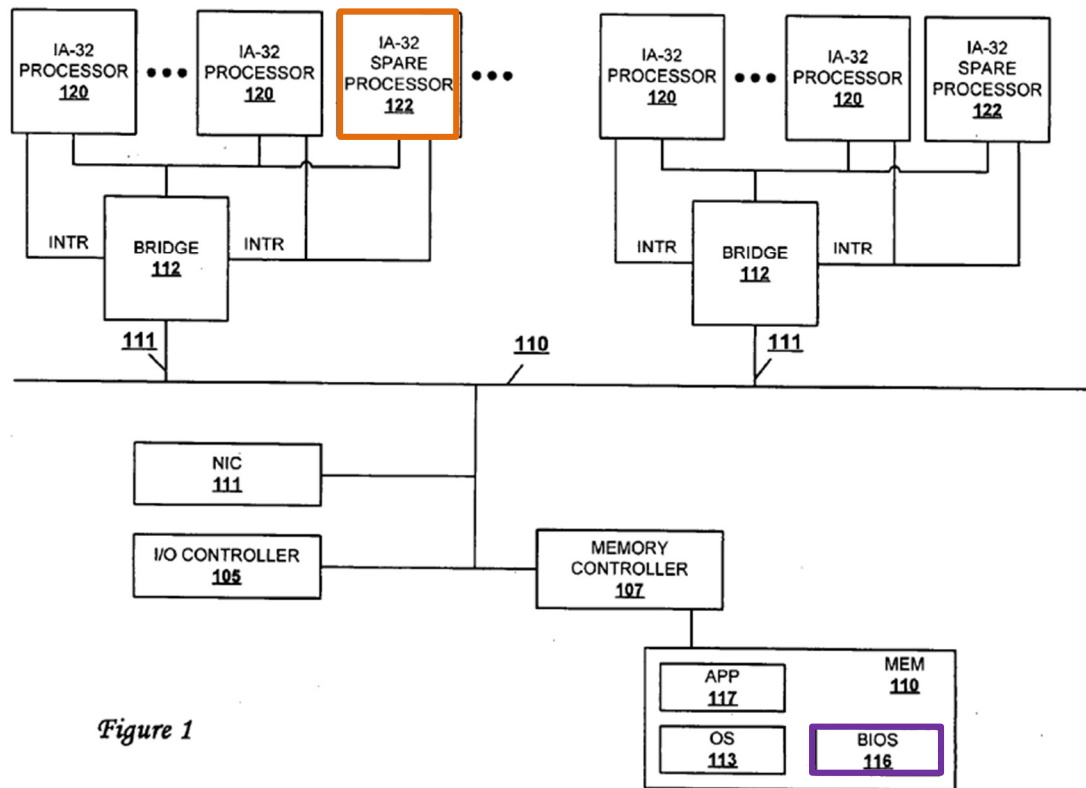


Figure 1

Id., FIG. 1 (annotated).

194. Fox discloses when a processor failure is detected, SMI activates and completes an initialization of the hot-spare processor, and the BIOS switches the failing processor to the active hot-spare processor (“held in idle until switched”).

Id., [0040]-[0041].

195. Likewise, Safford discloses that the spare processor pair 125/127 is designated as “a hot standby” (“hot spare”) and “sitting idle in lock step mode” (“held in idle”). Safford, [0020]. Safford discloses when loss of lockstep is detected in the processor pair 111/113, the architecture state of processor pair

111/113 is copied to the hot standby processor pair 125/127, which becomes an active processor. *Id.*, [0018], [0020]-[0021].

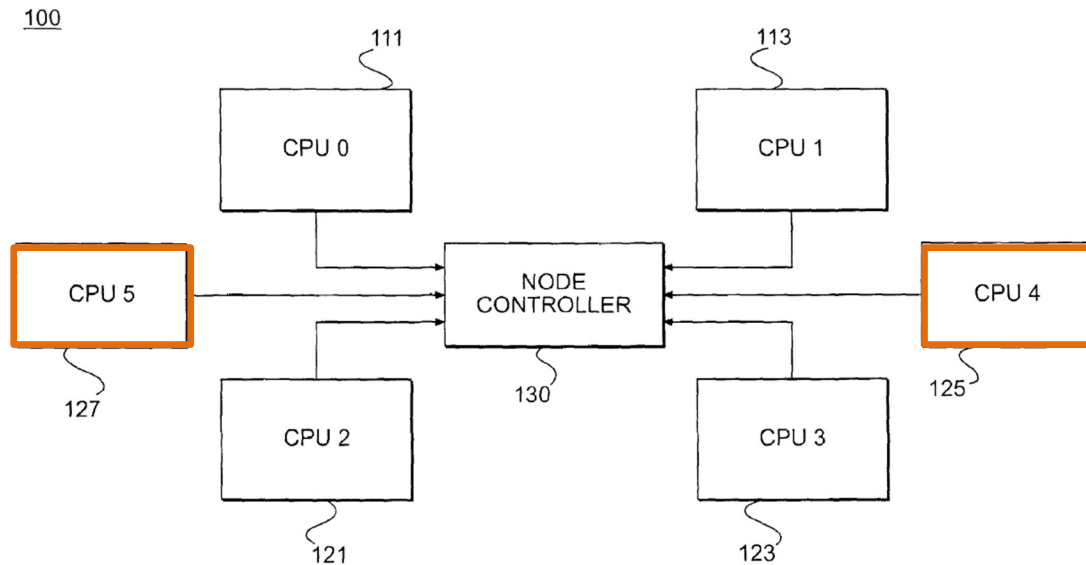


FIG. 3

Id., FIG. 3 (annotated).

196. A POSITA would have understood that in Fox combined with Safford, Fox's hot-spare processor would be a hot-spare processor pair "held in idle" in lockstep mode until being switched with the role of the boot processor detected of loss of lockstep, as taught in Fox and Safford, and as described above in Grounds 1-2, limitations [1b] and [1d].

2. **Claim 23: "wherein the another processor module is held in idle until said system firmware causes said operating system to recognize said another processor module as said system boot processor."**

197. Claim 23 depends from claim 21.

198. As discussed in limitation [21e], Fox in view of Safford and Arai or Fox in view of Safford, Landry, and Arai renders obvious “system firmware [that] causes said operating system to recognize said another processor module as said system boot processor.” For the additional reasons discussed above for claims 2 and 14, Fox in view of Safford further renders obvious the hot-spare processor (“the another processor module”) is held in idle until being switched to the role of boot processor. For reasons discussed in Grounds 1-2, limitations [1d] and [21e], a POSITA would have understood that switching the role of the spare processor includes changing the identification of the spare processor (“the another processor module”) to that of the boot processor, which would have caused the operating system to recognize the spare processor as the boot processor.

D. Dependent Claims 3 and 15

199. Claims 3 and 15 depend from claims 1 and 13, respectively. Claims 3 and 15 recite that “the spare processor is an active spare.” Fox discloses or renders obvious this additional limitation.

200. Fox discloses that, “in another embodiment[,], the system is made to enable the ‘hot spare processor’ feature on-the-fly by the OS with a driver-OS-SMI handshake. Tracking which processors are active may be completed by [a] software table of active processors, which is available to be read by the OS, when

determining workload allocation to each of the active processors.” Fox, [0035]. A POSITA would have understood that configuring a processor “on-the-fly” to mean that the configuration is performed dynamically as needed while the operating system is running. Thus, a POSITA would have understood that Fox discloses or renders obvious configuring one of the active processors to be the spare processor when a spare processor is needed for recovering a failing processor. A POSITA would also have understood such active processor to be “an active spare” because it would have been initialized as an active processor with a software tag indicating it is a spare processor. *Id.*, [0034].

201. A POSITA would have been motivated to use “an active spare” because allowing the spare processor to be available to the operating system as an active spare improves the processing capacity of the system when no processor failure occurs. A POSITA would have understood how to configure “an active spare” using ordinary computer programming skills and expect success. For example, Fox discloses the processor initialization/identification code includes a software tag that can be set to identify the processor as a spare processor during system BIOS boot. *Id.*, [0034]. Instead of reserving the spare processor in an idle state during initialization, a POSITA would have initialized the active spare processor like other active processors in the system. When a processor failure is

detected, the state of the active spare would have been set to idle to be switched with the failing processor, as taught in Fox. A POSITA would have known that various software tools and data structures of Fox's system would be readily available to achieve the runtime idling of the active spare for replacing the failing processor, including the SMI and BIOS, the operating system, and IA-32 processor register, table, and flags. *See* IA-32, 276-277, 279, 304-305 (teaching software can disable or enable a local APIC by setting the software enable/disable flag); MP Specification, 44 (CPU flags can be set to zero so that operating system does not access this processor).

202. A POSITA would have understood that in Fox combined with Safford, the active spare processor would have included a lockstep pair of processors as taught in Safford to replace the lockstep pair of processors detected to have loss of lockstep. Safford, [0020]-[0021], FIG. 4.

E. Dependent Claims 4 and 16

203. Claim 4 depends from claim 3, which depends from claim 1. Claim 16 depends from claim 15, which depends from claim 13. Claims 4 and 16 recite "causing the system's operating system to idle the active spare." Fox discloses or renders obvious this limitation.

204. For reasons above for claims 3 and 15, a POSITA would have understood that Fox discloses or renders obvious “an active spare” to be idled when needed during system runtime, and that Fox’s system allows using software to idle the active spare. A POSITA would have understood that Fox’s operating system would have been a suitable option to idle the active spare because the operating system would have access to the register of the processor, such as the APIC register, and set the processor flag indicating the state of the processor to the operating system. Fox, [0041] (providing a software register of active processors in the operating system), [0035] (using a software table to track active processors available to the operating system); IA-32, 305 (Figure 8-22 showing APIC software Enable/Disable Flag).

F. Dependent Claim 5: “wherein the spare processor is a processor module that includes a master and slave processors that operate in lockstep.”

205. Claim 5 depends from claim 1. Fox in view of Safford discloses or renders obvious this limitation.

206. As discussed above for limitation [21a], Safford discloses or renders obvious a lockstep processor (“a processor module”) that includes “a master and slave processors that operate in lockstep.” And for reasons explained in Section IX.B.1, a POSITA would have found it obvious to implement the processors in

Fox's multiprocessor systems as lockstep processors, each including a master processor and a slave processor, with error detection logics as taught in Safford to improve the reliability of Fox's server system. A POSITA would have implemented Fox's spare processor as a lockstep processor pair like other processors such that the spare processor pair would replace the lockstep processor pair having loss of lockstep. *See* Safford, [0018], [0021], FIG. 3.

G. Dependent Claims 6 and 25

1. **Claim 6: “wherein said determining that the processor for which said LOL is detected is assigned the role of boot processor comprises: accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of boot processor.”**

207. Claim 6 depends from claim 1.

208. As discussed above in Grounds 1-2, limitation [1b], Fox in view of Safford or Fox in view of Safford and Landry renders obvious “determining that the processor for which said LOL is detected is assigned the role of boot processor.” Fox and Safford (or Safford and Landry) further in view of Arai renders obvious “accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of boot processor.”

209. As explained in Grounds 1-2, limitation [1c], Fox discloses saving processor identifications in a register or table (e.g., BIOS table, ACPI table, MP table) accessible to the BIOS and operating system. Fox, [0026], [0033], [0035], [0038], [0041], [0046]; IA-32, 273-274, 267, 269, 289-297, 745 (e.g., local APIC ID or logical APIC ID). A POSITA would have understood that each processor would have a unique identification saved in Fox's register or table, including the boot processor and its BSP flag ("information ... that identifies one of the processors in said multi-processor system that is assigned the role of boot processor"). *See* Section V.B. Fox's BIOS would have accessed the register or table ("accessing information") to determine the failing processor identification and BSP flag. Fox, [0042] (SMI identifies to the BIOS which one of the processors is failing). *See* Grounds 1-2, limitation [1b].

210. Fox discloses a system memory for storing the BIOS, but does not expressly disclose the processor register or table is stored in the system memory. Fox, [0028], FIG. 1. Arai discloses this. Arai discloses the table that identifies all processors in the system (e.g., ACPI table or MP table) is stored in a main memory ("data storage"). Arai, [0019]-[0021], [0025]-[0026]. Both Fox and Arai disclose using registers or tables to store processor information available to the OS and the BIOS, and use the stored information to implement the switching. A POSITA

would have found it suitable to store Fox's register or table accessed by the BIOS for determining the failing processor in its system memory ("data storage"), like in Arai. A POSITA would have understood that Fox's system memory would be "non-volatile" because it was common and well known that the BIOS would be embedded in a non-volatile memory (e.g., ROM, EEPROM, etc.). *See* Fox, [0028], FIG. 1.

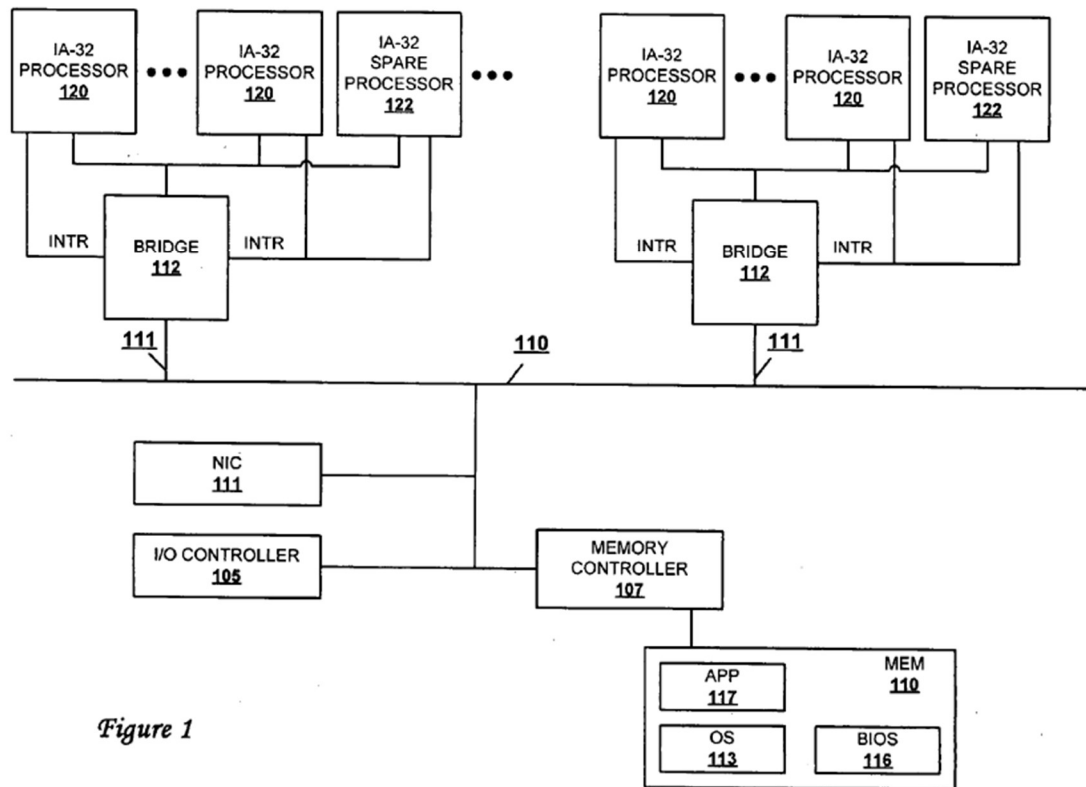


Figure 1

Id., FIG. 1.

211. A POSITA would have found such modification routine and ordinary because it was well known that the processor register or table should be stored in a

memory and accessible to the BIOS. *See* IA-32, 242 (saving ACPI and MP tables in RAM); MP Specification, 38 (MP configuration table must be stored either in a non-reported system RAM or within the BIOS read-only memory space).

212. The '302 Patent similarly discloses a device tree or ACPI table that is stored in a RAM, accessible to the firmware, and designates which processor is the boot processors. Ex. 1001, 9:4-15, 12:60-66, 13:21-44.

2. Claim 25: “non-volatile data storage storing information that assigns the role of system boot processor to one of said plurality of processor modules.”

213. Claim 25 depends from claim 21.

214. For reasons discussed in claim 6 above, Fox and Safford (or Safford and Landry) further in view of Arai renders this limitation obvious. Fox in view of Arai discloses or renders obvious the system memory (“non-volatile data storage”) storing a processor register or table containing the unique identifications of the processors of the system, including the boot processor and its BSP flag (“information that assigns the role of system boot processor to one of said plurality of processor modules”).

H. Dependent Claims 7, 8, and 22

1. Claim 7: “determining the processor that is assigned the role of spare processor.”

215. Claim 7 depends from claim 1. Fox in view of Arai renders obvious this limitation.

216. Fox discloses the spare processor is reserved and identified with a software tag during system initialization (“assigned the role of spare processor”), and in response to detecting a processor failure, the BIOS switches the failing processor to the spare processor. Fox, [0032]-[0034], [0040]-[0042]; *see* Section VII.A. Fox does not explicitly disclose determining the processor that is assigned the role of spare processor before processor switching. Arai does.

217. Arai discloses that in response to detecting an error on an active processor, the BIOS interrupt handler performs a [test 108](#) to determine if there are any processors reserved during system initialization available to replace the active processor (“determining the processor that is assigned the role of spare processor”). Arai, [0022], FIG. 3. Arai further discloses generating a table (e.g., ACPI table, MP table, or an alternative table) that identifies each processor and the state of each processor. *Id.*, [0019]-[0021], [0025]-[0026]. A POSITA would have understood that Arai’s determination would include having the BIOS interrupt

handler examine Arai's table and determine the processors that are in an inactive, reserved, or idle state. *Id.*, [0019]-[0021].

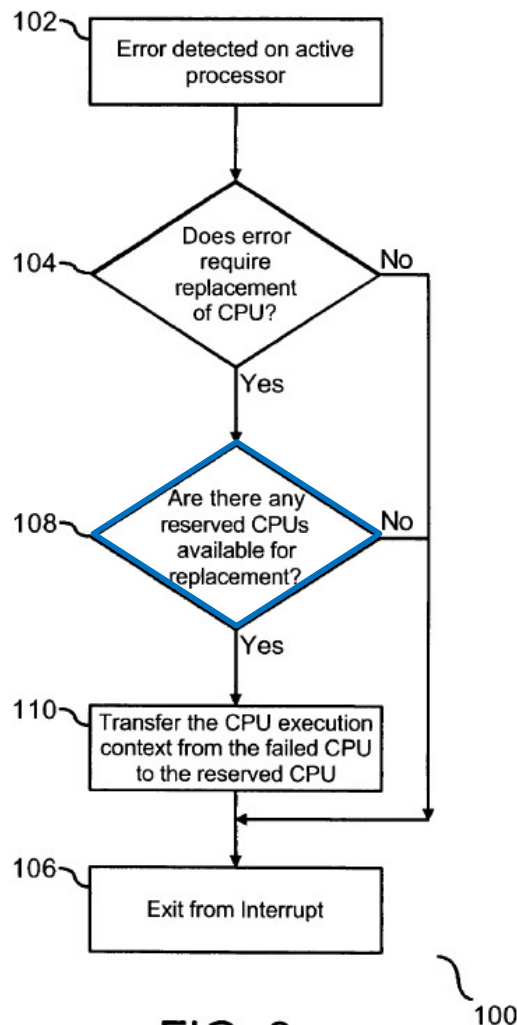


FIG. 3

Id., FIG. 3 (annotated).

218. A POSITA would have been motivated to modify Fox to include Arai's determination step in response to detecting a processor failure to ensure a spare processor is available before switching the failing processor to the spare

processor. For example, Fox's system may detect a second failing processor during the switching of a first failing processor. Adding Arai's determination step would reduce the risk of unsuccessful failover. Furthermore, Fox's system includes one or more spare processors. Fox, [0015], [0024]. A POSITA would have been motivated to add Arai's determinization of the currently available spare processor to allow the BIOS to select one of the spare processors for replacing the failing processor.

219. A POSITA would have expected success in adding Arai's determination in Fox's processor-switching process using ordinary computer programming skills. Both Arai and Fox use the BIOS interrupt handler to facilitate the processor switching. Arai, [0022]; Fox, [0041]. And as discussed above, like Arai, Fox's system also includes a register or table (e.g., BIOS table, ACPI table, MP table) accessible to the BIOS and operating system. Fox, [0026], [0033], [0035], [0038], [0041], [0046]; IA-32, 273-274, 267, 269, 289-297, 745 (e.g., local APIC ID or logical APIC ID). A POSITA would have understood that such table would include the information of the state of the processor in the form of a software tag or flag. *See* Fox, [0034] (a software tag to identify the processor as a spare processor; IA-32, 305 (Figure 8-22 showing APIC software Enable/Disable Flag). Programming Fox's BIOS interrupt handler to examine the tags or flags of

the processors in the register or table to determine the processors' states merely involves ordinary computer programming skills.

220. The '302 Patent similarly discloses the system firmware examines the device tree to determine which processor is an active spare. Ex. 1001, 17:15-19, FIG. 5.

2. **Claim 8: “wherein said determining the processor that is assigned the role of spare processor comprises: accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of spare processor.”**

221. Claim 8 depends from claim 7. Fox in view of Arai renders obvious this limitation.

222. As discussed above in claim 7, Fox in view of Arai renders obvious examining a register or table (e.g., ACPI table, MP table, or BIOS table) to determine which spare processor is available to replace the failing processor (“accessing information ... that identifies one of the processors in said multi-processor system that is assigned the role of spare processor”). For reasons discussed in claim 6, a POSITA would have found it obvious to store such information in Fox's system memory (“non-volatile data storage”).

3. Claim 22: “wherein the another processor module is a processor module assigned a role of spare processor.”

223. Claim 22 depends from claim 21. Fox discloses or Fox in view of Arai renders obvious this limitation.

224. Fox discloses setting a software tag to identify a processor as a spare processor during system BIOS boot (“assigned a role of spare processor”) and replacing the failing processor with the spare processor. Fox, [0023], [0034], [0046]. And for reasons discussed in claim 7, Fox in view of Arai renders obvious a table that identifies each processor and the state of each processor (e.g., reserved, idle, or inactive), and examines the table to determine the processor that is assigned the role of spare processor. Arai, [0019]-[0021], [0025]-[0026].

I. Dependent Claim 9, 10, and 18-20

1. Claims 9 and 18: “wherein said determining step is performed by system firmware.”

225. Claims 9 and 18 depend from claims 1 and 13, respectively. Fox discloses this limitation for the reasons discussed above for limitation [13a].

2. Claims 10 and 20: “wherein said switching step is performed by system firmware.”

226. Claims 10 and 20 depend from claims 1 and 13, respectively. As discussed above for limitation [13a], a POSITA would have understood that Fox’s BIOS as system firmware. As discussed in Section VII.A and Grounds 1-2,

limitation [1d], Fox discloses that the BIOS performs the switching of the failing processor to the spare processor (“switching step is performed by system firmware”). Fox, [0041].

3. Claim 19: “wherein said copying step is performed by system firmware.”

227. Claim 19 depends from claim 13.

228. For the reasons explained above for limitation [13a], a POSITA would have understood Fox’s BIOS as system firmware, and Fox discloses the BIOS switches the memory state of the failing processor to the spare processor during processor switching (“copying step is performed by system firmware”). *Id.*, [0041].

229. For the reasons explained above for limitation [13c], Fox in view of Safford renders obvious the “copying step.” A POSITA would have understood that in Fox combined with Safford, the “copying step” would be performed by the BIOS as taught by Fox.

XII. Fox in View of Safford, Arai, and Bigbee (Ground 3) or Fox in View of Safford, Landry, Arai, and Bigbee (Ground 4) Renders Obvious Claims 26 and 27

230. In my opinion, Fox in View of Safford, Arai, and Bigbee or Fox in View of Safford, Landry, Arai, and Bigbee render obvious claims 26 and 27.

A. Overview of the Combinations

231. As discussed above, Fox in view of Safford and Arai (Ground 1) or Fox in view Safford, Landry, and Arai (Ground 2) renders obvious system firmware “detecting loss of lockstep (LOL) for a processor in a multi-processor system” (*see* Grounds 1-2, limitation [1a]), system firmware for “determining whether the processor for which said LOL is detected is assigned the role of boot processor” (*see* Grounds 1-2, limitation [13a]), and system firmware for “switching the role of boot processor to a spare processor without shutting down the system’s operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor” (*see* Grounds 1-2, limitation [1d], Claims 10, 20). And a POSITA would have understood that Fox’s BIOS is system firmware. *See* limitation [13a], Section V.D. To the extent that the Board finds means-plus-function interpretation applies to the claim limitations reciting “system firmware means” (*see* Section IV.C.2) and construes these limitations to require additional firmware structures, these limitations are further rendered obvious in view of Bigbee.

B. A POSITA Would Have Found It Obvious to Configure Fox's System Combined with Safford and Arai or Safford, Landry, and Arai, with Itanium Family Processors and Firmware as Taught in Bigbee to Improve System Performance and Capacity

232. Fox's multiprocessor server system is configured with IA-32 architecture and comprises IA-32 processors and a BIOS of the IA-32 architecture. Fox, Abstract, [0014]-[0015], [0024], [0028], [0033]. Bigbee discloses a multiprocessor system comprising Itanium family processors and architectures. Bigbee, [0033]; Ex. 1030, 267-268. A POSITA would have known that Intel's IA-32 architecture is a 32-bit instruction set architecture and Itanium family processors' architecture is a 64-bit instruction set architecture. Ex. 1008; Ex. 1030, 19. A POSITA would have been motivated to upgrade Fox's system's IA-32 processors and architecture to more advanced Itanium family processors and architecture (as modified by Safford and Arai or Safford, Landry, and Arai) to improve performance and processing capacity, for example, for high-end server systems.

233. A POSITA would have known that 64-bit instruction set architecture is designed to operate on 64-bit data, enabling more advanced computational tasks than legacy 32-bit instruction set architecture. Systems and processors configured with 64-bit architecture, like the Itanium family processors, would run more instructions and processes than those configured with 32-bit architecture because

the processors have increased compute resources, buses, memories, and caches, allowing for increased computational throughput. *See* Ex. 1030.

234. A POSITA also would have understood that the firmware model of Itanium family processors is ACPI-compliant and provides numerous advantageous over legacy BIOS solutions. For example, the ACPI evolves the existing collection of power management BIOS code, Advanced Power Management (APM) application programming interfaces (APIs, PNPBIOS APIs, Multiprocessor Specification (MPS)) tables and so on into a well-defined power management and configuration interface specification. ACPI provides the means for an orderly transition from existing (legacy) hardware to ACPI hardware, and it allows for both ACPI and legacy mechanisms to exist in a single machine and to be used as needed. Ex. 1029, 15.

235. A POSITA would have expected success in upgrading Fox's system with Itanium family processors and architecture using ordinary computer programming skills and common knowledge of Itanium processors and architecture. A POSITA would have known the Itanium-based firmware model and how the various firmware layers (PAL, SAL, EFI) work together. *See* Ex. 1030, 267-276. Indeed, Intel and other sources have extensively documented the Itanium architecture, including its firmware model (*see* Ex. 1030), and Itanium family

processors had been used in many fault-tolerant multiprocessor systems like that of Fox. *See* Bigbee, [0033]; Nguyen, [0045].

C. Independent Claim 26

236. Fox in view of Safford, Arai, and Bigbee (Ground 3) or Fox in view of Safford, Landry, Arai, and Bigbee (Ground 4) renders obvious claim 26.

1. [26preamble]: “A system comprising:”

237. Fox discloses [26preamble] for reasons explained in Grounds 1-2, limitation [21preamble].

2. [26a]: “system firmware means for detecting loss of lockstep (LOL) for a processor in a multi-processor system;”

238. Fox in view of Safford and Bigbee renders obvious [26a].

239. As discussed in Section IV.C.2, this presumptive means-plus-limitation should be construed to have the claimed function of “detecting loss of lockstep (LOL) for a processor in a multi-processor system,” and the corresponding structure in the specification is firmware including a Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI) or equivalents thereof. Ex. 1001, 8:34-59, 11:4-60, 12:36-59, FIG. 2. Bigbee discloses this structure, and Fox in view of Safford and Bigbee renders obvious using this structure to perform the claimed function.

240. As discussed above in Grounds 1-2, limitation [1a], Safford discloses a single logical processor comprising a pair of processors in lockstep and error detection logics for detection loss of lockstep between the lockstep pair (Safford, [0017]-[0018]), and Fox in view of Safford renders obvious detecting loss of lockstep (LOL) for a processor in a multi-processor system.

241. Like Safford, Bigbee discloses a single logical processor comprising two processor cores operating in FRC mode (lockstep) and FRC logic for detecting loss of lockstep between the two processor cores, as shown below. Bigbee, [0019], [0032]-[0033], FIG. 3.

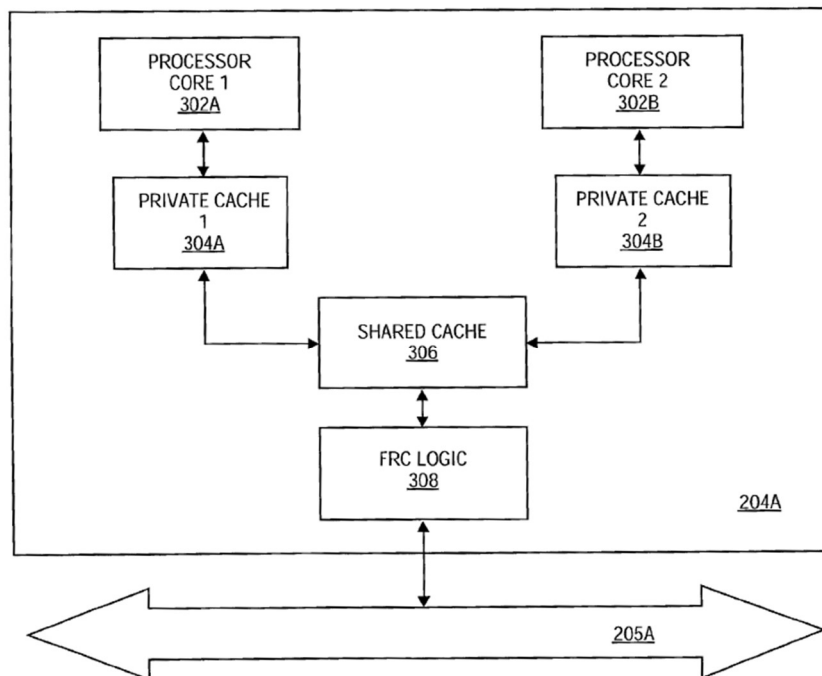


FIG. 3

Id., FIG. 3.

242. Bigbee discloses implementing its FRC logic in executable firmware. *Id.*, [0034]. A POSITA would have implemented the error detection logics of Fox combined with Safford in firmware as taught in Bigbee. A POSITA would have understood that firmware is one of three suitable options for detecting loss of lockstep—firmware, hardware, or a combination—all of which would have been obvious to try, and any of which would be a suitable alternative for the others. A POSITA also would have understood the benefits of using firmware because firmware is flexible. It can be modified after production, updated, and supported relatively quickly and inexpensively.

243. Bigbee discloses its processors are Itanium family processors and its firmware includes “an extensible firmware interface (EFI), a system abstraction level (SAL), and a processor abstraction level (PAL).” *Id.*, [0018], [0034]. As discussed in Section XII.B, a POSITA would have found it obvious to configure Fox’s system with Itanium processors and architecture, including its firmware model comprising EFI, SAL, and PAL as disclosed in Bigbee. Thus, Fox combined with Safford’s lockstep processors and error detection logics implemented in Itanium firmware as taught in Bigbee renders obvious this limitation.

3. [26b]: “system firmware means for determining whether the processor for which said LOL is detected is assigned the role of boot processor; and”

244. Fox in view of Safford and Bigbee (Ground 3) or Fox in view of Safford, Landry, and Bigbee (Ground 4) renders obvious limitation [26b].

245. As discussed in Section IV.C.2, this presumptive means-plus-limitation should be construed to have the claimed function of “determining whether the processor for which said LOL is detected is assigned the role of boot processor,” and the corresponding structure in the specification is firmware including a Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), Extended Firmware Interface (EFI), and a device tree, or equivalents thereof. Ex. 1001, 5:34-48, 9:49-60, 11:4-60, 12:36-14:3, 18:50-58, FIG. 2. Bigbee discloses this structure, and Fox in view of Safford and Bigbee or Fox in view of Safford, Landry, and Bigbee renders obvious using this structure to perform the claimed function.

246. Further as discussed above for limitation [26a], Bigbee discloses its processors are Itanium family processors and its firmware includes “an extensible firmware interface (EFI), a system abstraction level (SAL), and a processor abstraction level (PAL).” Bigbee, [0018], [0034].

247. Bigbee also discloses a device tree or its equivalent as described in the '302 Patent. Bigbee discloses that its system comprises “an advanced configuration and power interface (ACPI)-compliant processing system,” and that, accordingly, the operating system includes an ACPI device driver and firmware that includes “ACPI machine language (AML) program code describing what hardware devices are present within data processing system 100, as well as their configuration and interfaces, via ACPI control methods, objects, registers, and tables.” *Id.*, [0016]. A POSITA would have understood that this is similar to the '302 Patent disclosing that the firmware builds a device tree, converts this information to ACPI tables format, and presents it to the operating system. Ex. 1001, 12:60-13:2.

248. A POSITA also would have understood that Fox's register or table (e.g., BIOS table, ACPI table, MP table) is created by the BIOS (firmware) during system initialization as a device tree or its equivalent. Fox, [0032]-[0033], [0035].

4. **[26c]: “system firmware means for switching the role of boot processor to a spare processor without shutting down the system's operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor.”**

249. Fox in view of Safford, Arai and Bigbee (Ground 3) or Fox in view of Safford, Landry, Arai, and Bigbee (Ground 4) renders obvious limitation [26c].

250. As discussed in Section IV.C.2, this presumptive means-plus-function limitation should be construed to have the claimed function of “switching the role of boot processor to a spare processor without shutting down the system’s operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor,” and the corresponding structure in the specification is firmware including a Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI) or equivalents thereof. Ex. 1001, 11:4-60, 12:36-14:3, 17:12-18:43, FIG. 2. Bigbee discloses this structure, and Fox in view of Safford, Arai and Bigbee or Fox in view of Safford, Landry, Arai, and Bigbee renders obvious using this structure to perform the claimed function.

251. As discussed above for Grounds 1-2, limitation [1d], claims 10 and 20, Fox in view of Safford and Arai (Ground 1) or Fox in view of Safford, Landry, and Arai (Ground 2) renders obvious system firmware (e.g., BIOS) for switching the role of boot processor to a spare processor without shutting down the system’s operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor.

252. Further as discussed above for limitation [26a], Bigbee discloses system firmware that includes “an extensible firmware interface (EFI), a system

abstraction level (SAL), and a processor abstraction level (PAL).” Bigbee, [0018]. For reasons discussed in Section XII.B, a POSITA would have found it obvious to configure Fox’s system (as modified with Safford and Arai, or Safford, Landry, and Arai) with Itanium architecture, including its firmware model comprising EFI, SAL, and PAL as disclosed in Bigbee.

D. Dependent Claim 27: “wherein said processor in said multi-processor system is an Itanium Processor Family (IPF) processor.”

253. Fox in view of Safford, Arai, and Bigbee (Ground 3) or Fox in view of Safford, Landry, Arai, and Bigbee (Ground 4) renders obvious claim 27.

254. Bigbee discloses using Itanium family processors in a multiprocessor system. *Id.*, [0033]. For reasons discussed in Section XII.B, a POSITA would have been motivated to upgrade Fox’s system to use Itanium family processors and architecture and expect success.

XIII. Conclusion

255. For the reasons set forth above, in my opinion, a POSITA as described in paragraph 29 would have considered the Challenged Claims of the ’302 Patent to be obvious over the prior art discussed above.

256. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be

subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

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

Dated: August 29, 2024

By: 

Michael C. Brogioli, Ph.D.