RISC-V Memory Consistency Model Tutorial

Dan Lustig

May 7, 2018

NVIDIA

ABOUT ME

- Senior Research Scientist at NVIDIA in Santa Clara, CA
- PhD from Princeton in 2015
- Chair of RISC-V Memory Consistency Model task group
- Co-responsible for NVIDIA GPU memory consistency model

OUTLINE

- Setting the Stage
- Litmus Tests
- RISC-V Weak Memory Ordering (“RVWMO”)
- Extensions: “Zam” and “Ztso”
- Documentation and Tools
- Conclude

WHAT IS A MEMORY CONSISTENCY MODEL?

Specifies the values that can be returned by loads

WHY DO WE NEED A MEMORY MODEL?

WHY DO WE NEED A MEMORY MODEL?

…to give everyone a headache?

WHY DO WE NEED A MEMORY MODEL?

For the same reason we need any other technical specification:

It is (one specific part of) the contract between the software and the implementation about the set of legal behaviors

WHY DO WE NEED A MEMORY MODEL?

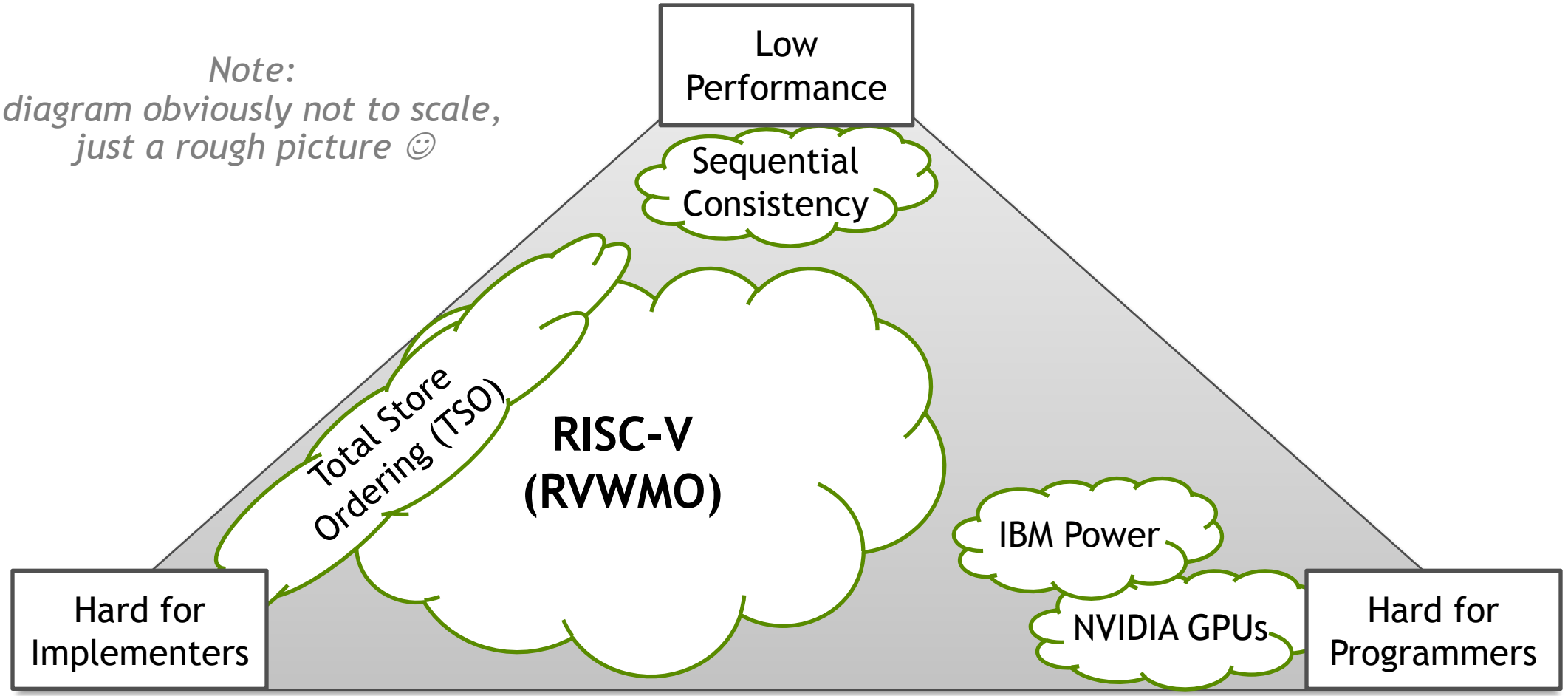
For the same reason we need any other technical specification:

It is (one specific part of) the contract between the software and the implementation about the set of legal behaviors

The other parts of the contract are defined by the rest of the ISA specification (including the ISA Formal Specification; see that TG's tutorial later today)

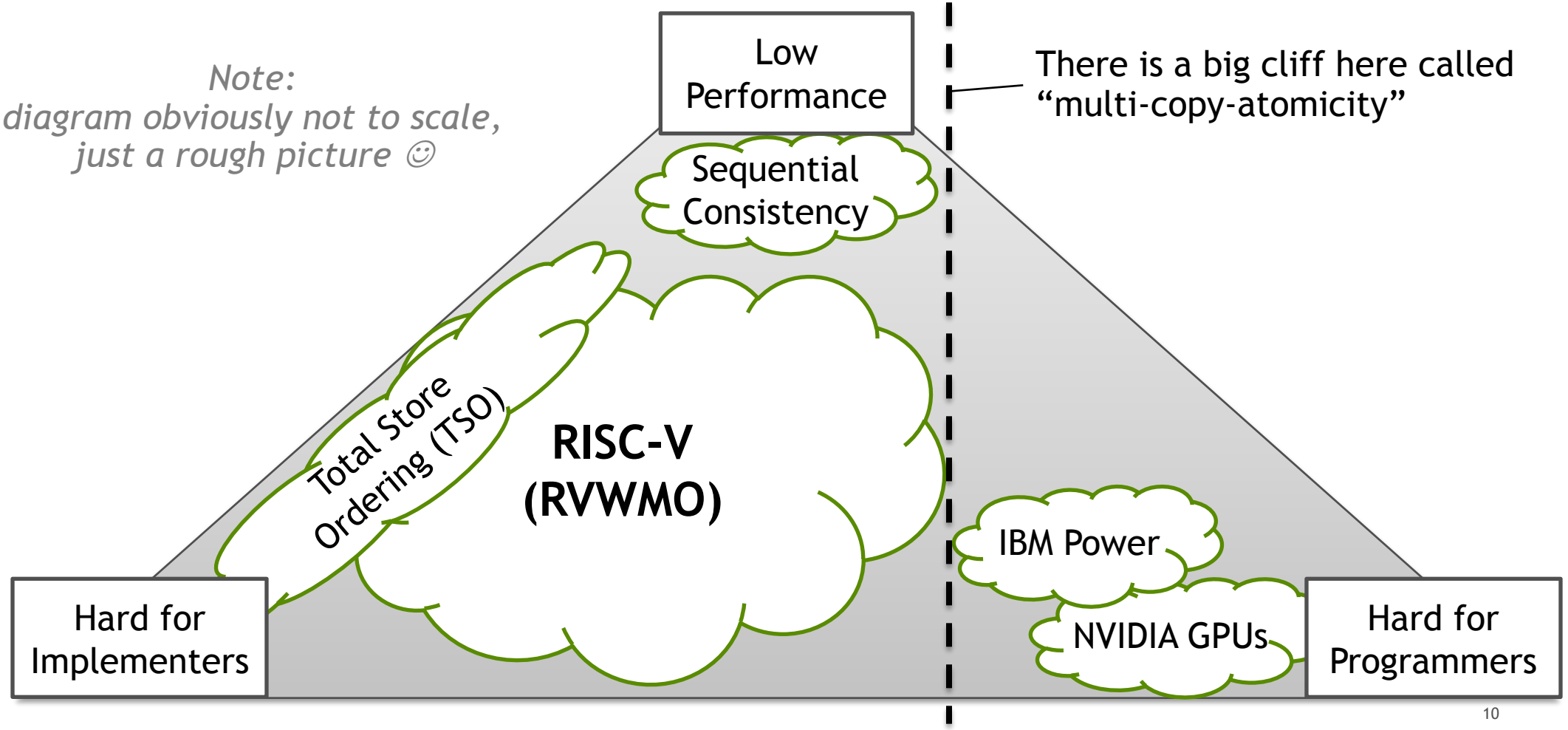
A WIDE RANGE OF MEMORY MODELS

Note:
diagram obviously not to scale,
just a rough picture ☺



A WIDE RANGE OF MEMORY MODELS

Note:
diagram obviously not to scale,
just a rough picture ☺



SOME CASES ARE EASY (RELATIVELY)…

Initial condition on both harts: s0 == address x; s1 == address y.
Initial conditions in memory: all locations initialized to 0

Hart 0	Hart 1
li t1, 1	loop:
sw t1, 0(s0)	lw a0, 0(s1)
fence w,w	beqz a0, loop
sw t1, 0(s1)	fence r,r
	lw a1, 0(s0)

Final output: what are the possible final values of a0 and a1 on hart 1?

SOME CASES ARE EASY (RELATIVELY)…

Initial condition on both harts: s0 == address x; s1 == address y.
Initial conditions in memory: all locations initialized to 0

Hart 0	Hart 1
li t1, 1	loop:
sw t1, 0(s0)	lw a0, 0(s1)
fence w,w	beqz a0, loop
sw t1, 0(s1)	fence r,r
	lw a1, 0(s0)

Final output: what are the possible final values of a0 and a1 on hart 1?

Only possible outcome is a0 == a1 == 1

SOME CASES ARE HARD…

- Should this outcome be permitted or forbidden? We're not even sure ourselves…

	Hart 0		Hart 1
	li t1, 1		li t1, 1
(a)	lw a0,0(s0)	(d)	sw t1,4(s1)
(b)	fence rw,rw	(e)	ld a1,0(s1)
(c)	sw t1,0(s1)	(f)	lw a2,4(s1)
			xor a3,a2,a2
			add s0,s0,a3
		(g)	sw a2,0(s0)
Outcome:	a0=1, a1=0x100000001, a1=1		

Figure A.22: Mixed-size discrepancy (permitted by axiomatic models, forbidden by operational model)

ARCHITECTURE VS. MICROARCHITECTURE

An implementation can do anything it wants under the covers, as long as the load return values satisfy RVWMO

i.e., implementations can speculate past a lot of these rules, as long as they make sure to, e.g., squash and replay whenever the violation might actually become observable

OPERATIONAL VS. AXIOMATIC

In modern practice, at ISA level, two common modeling approaches:

Axiomatic: define a set of criteria (“axioms”) to be satisfied

- Executions permitted unless they fail one or more axioms

Operational: define a golden abstract machine model

- Executions forbidden unless producible when executing this model

Ideally: figure out how to meet in the middle (can be difficult!)

- lots of gray area, obscure code, etc.

SEQUENTIAL CONSISTENCY [LAMPORT '79]

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order
3. Loads return the value written by the latest store to the same address in the total order

Operational

1. Harts take turn executing instructions. The order is non-deterministic.
2. Each hart executes its own instructions in order
3. Loads return the value written by the most recent preceding store to the same address

SEQUENTIAL CONSISTENCY [LAMPORT '79]

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order
3. Loads return the value written by the latest store to the same address in the total order

Global memory order

Preserved Program Order (PPO)

Load Value Axiom

Operational

1. …xecuting instructions. The order is non-deterministic.
2. …cutes its own instructions in order
3. Lo… …ue written by… …preceding store to the same address

GLOBAL MEMORY ORDER

A total order over all memory operations in a program

A memory operation “performs” (enters the global memory order) when:

- a load determines its return value
- a store becomes globally visible

SEQUENTIAL CONSISTENCY [LAMPORT '79]

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order
3. Loads return the value written by the latest store to the same address in the total order

Operational

1. Harts take turn executing instructions. The order is non-deterministic.
2. Each hart executes its own instructions in order
3. Loads return the value written by the most recent preceding store to the same address

TOTAL STORE ORDERING (SPARC, X86, RVTSO)

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order, except Store→Load ordering
3. Loads return the value written by the latest store to the same address in program or memory order (whichever is later)

Operational

1. Harts take turn executing steps. The order is non-deterministic.
2. Each hart executes its own instructions in order
3. Stores execute in two steps: 1) enter store buffer, 2) drain to memory
4. Loads first try to forward from the store buffer. If that fails, they return the value written by the most recent preceding store to the same address

ADDING A STORE BUFFER

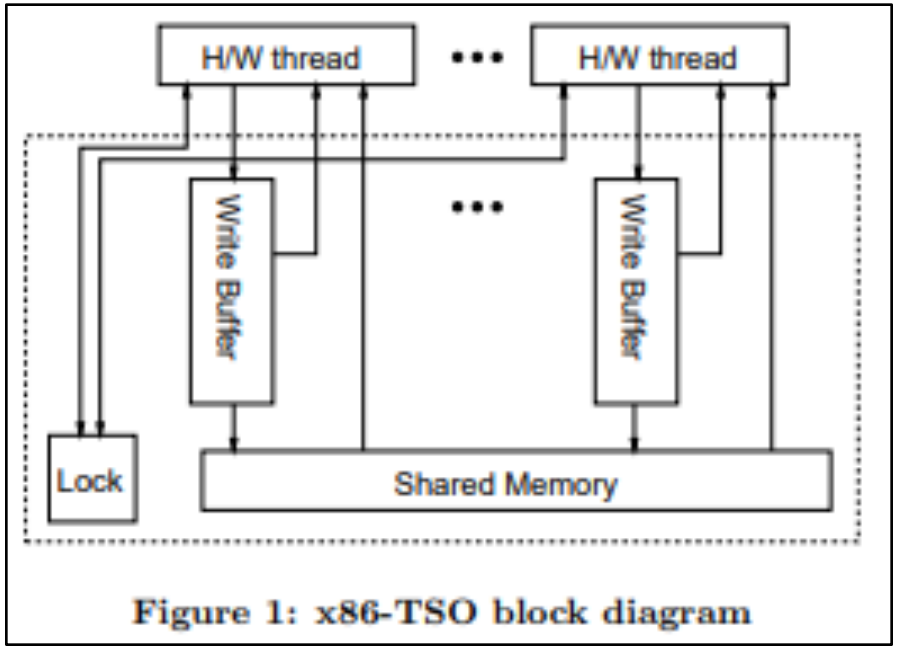


Figure 1: x86-TSO block diagram

- If a load bypasses a store in the (FIFO) store buffer, then the load appears before the store in global memory order
  - The load determines its return value before the store becomes globally visible
- The performance win is too important…the model needs to be changed to account for this!

[Sewell et al., CACM ‘10]

TOTAL STORE ORDERING (SPARC, X86, RVTSO)

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order, except Store→Load ordering
3. Loads return the value written by the latest store to the same address in program or memory order (whichever is later)

Operational

1. Harts take turn executing steps. The order is non-deterministic.
2. Each hart executes its own instructions in order
3. Stores execute in two steps: 1) enter store buffer, 2) drain to memory
4. Loads first try to forward from the store buffer. If that fails, they return the value written by the most recent preceding store to the same address

TOTAL STORE ORDERING (SPARC, X86, RVTSO)

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects program order, except Store→Load ordering
3. Loads return the value written by the latest store to the same address in program or memory order (whichever is later)

Global memory order

Preserved Program Order (PPO)

Load Value Axiom

Operational

1. …uting steps. The
2. …tes its own …rder
3. Stores execute in two steps: 1) enter store buffer, 2) drain to memory
4. Lo… …d from the sto… …er. If that fails, they return the value written by the most recent preceding store to the same address

RISC-V WEAK MEMORY ORDERING (RVWMO)

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects **thirteen specific patterns (next slide)**
3. Loads return the value written by the latest store to the same address in program or memory order (whichever is later)

Operational

1. Harts take turn executing steps. The order is non-deterministic.
2. Each hart executes its own instructions in order
3. Multiple steps for each instruction (see spec Appendix B)
4. Loads first try to forward from the store buffer. If that fails, they return the value written by the most recent preceding store to the same address

RISC-V WEAK MEMORY ORDERING (RVWMO)

Axiomatic

1. There is a total order on all memory operations. The order is non-deterministic.
2. That total order respects **thirteen specific patterns (next slide)**
3. Loads return the value written by the latest store to the same address in program or memory order (whichever is later)

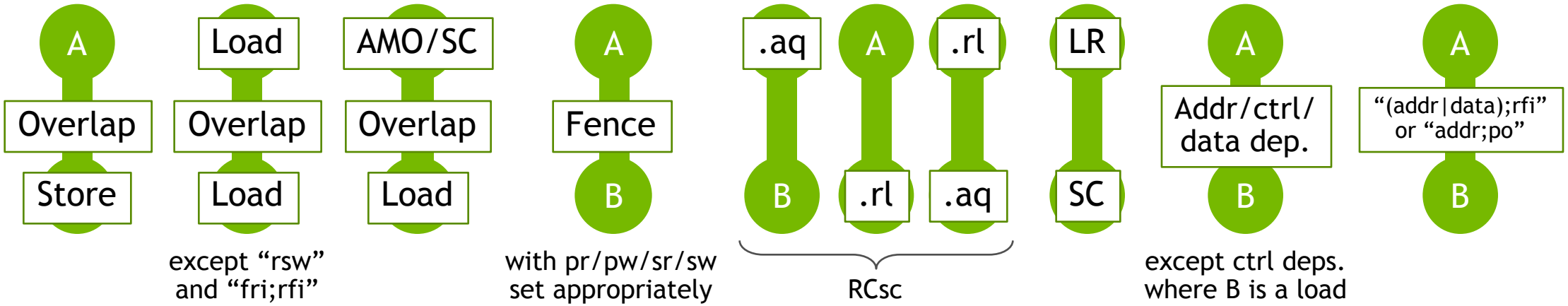
Global memory order

Preserved Program Order (PPO)

Load Value Axiom

RVWMO PPO RULES IN A NUTSHELL

- **Preserved Program Order:** if **A** appears before **B** in program order, and **A** and **B** match one of the patterns below, then **A** appears before **B** in global memory order.

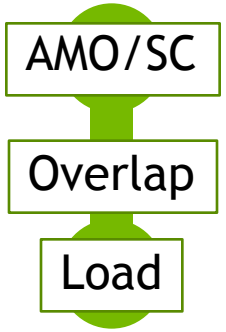


PPO RULE 1

If A and B access the same address (or have any overlapping footprint), then A must appear before B in global memory order:

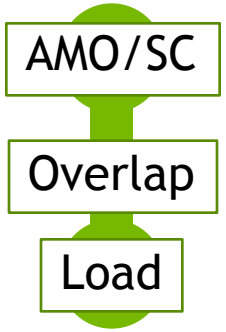
- A load A must determine its value before B becomes globally visible
- A store A must become globally visible before B becomes globally visible

PPO RULE 3



A load B cannot determine its return value by forwarding from an Atomic Memory Operation or Store-Conditional operation that has not yet become globally visible

PPO RULE 3



A load B cannot determine its return value by forwarding from an Atomic Memory Operation or Store-Conditional operation that has not yet become globally visible

(Recall: this defines the architectural rules. Implementations can do whatever they want, as long as all outcomes are legal)

PPO RULE 4

with pr/pw/sr/sw
set appropriately

fence [r][w][i][o], [r][w][i][o]

Orders operations in the *predecessor set* before operations in the *successor set*

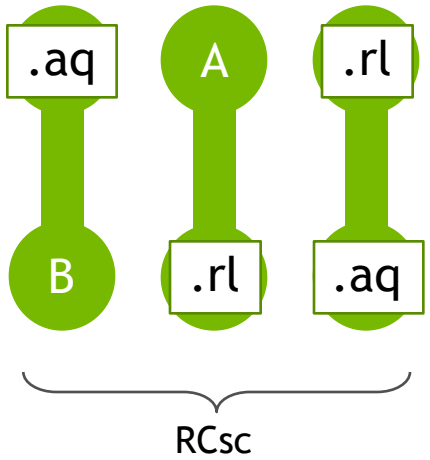
PR: previous reads. SR: subsequent reads

PW: previous writes. SW: subsequent writes

PI: previous I/O reads. SI: subsequent I/O reads

PO: previous I/O writes. SO: subsequent I/O writes

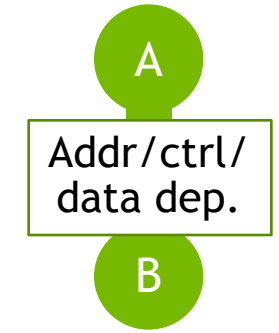
PPO RULES 5-7



AMOs and LR/SC have optional *acquire* and *release* annotations for release consistency

- All operations following an *acquire* in program order also following it in global memory order
- All operations preceding a *release* in program order also precede it in global memory order
- A *release* that precedes an *acquire* in program order also precedes it in global memory order
  - i.e., the RCsc variant of release consistency

PPO RULES 9-11



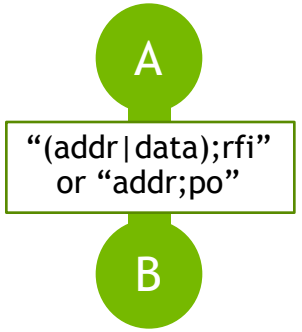
except ctrl deps.
where B is a load

If B has a syntactic *address*, *control*, *or data* dependency on A, then A precedes B in global memory order

- Except control dependencies where B is a store
- Address dependency: the result of A is used to determine the address accessed by B
- Control dependency: the result of A feeds a branch that determines whether B is executed at all
- Data dependency: the result of A is used to determine the value written by store B

Note: ordering maintained regardless of actual values!

PPO RULES 12-13



1. B follows M in program order, and M has an address dependency on A
2. B returns a value from an earlier store M in the same hart, and M has an address or data dependency on A

Most processors will maintain these naturally, yet most programmers won’t ever use them anyway

We made them explicit rules so that the operational and axiomatic models all agree

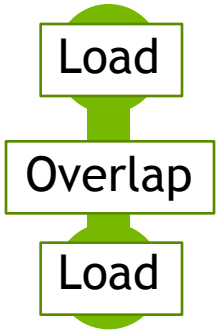
- And also for Linux, which has similar rules too

PPO RULE 8

A load-reserve operation determines its value before the paired store-conditional becomes globally visible

(Mostly redundant with rules 1 and 11, except in rare cases of mismatched addresses and no data dependency)

PPO RULE 2



except “rsw” and “fri;rfi”

Same-address load-load ordering is also maintained, with two exceptions:

1. Both return values come from the same store
   - A form of architecturally-visible speculation
   - Common in many implementations
2. B forwards from a store M between A and B in program order
   - B can determine its value from the store buffer while A is still fetching an older value from memory

ATOMICITY OF AMO AND LR/SC

AMOs grab an old value in memory, perform an arithmetic operation (except for swap), and write the new value to memory, all in one single atomic operation

- One node in the global memory order

LR grabs a reservation. SC performs a store if the reservation is still valid, and then releases the reservation.

- A reservation can be killed for any reason. A reservation must be killed if there is a store to the reserved address range from any other hart.
- Certain constrained LR/SC sequences guaranteed to eventually succeed (see spec)

PROGRESS AXIOM

No operation can be preceded in the global memory order by an infinite sequence of operations from other harts

- Very intentionally the weakest forward progress guarantee that is needed to make the memory model work
- Does not imply any stronger notion of fairness!

…AND THAT'S IT!

MEMORY MODEL ISA EXTENSIONS

- **"Zam"** extends "A" by permitting misaligned AMOs
  - "A" without "Zam" now forbids misaligned AMOs or LR/SC pairs
- **"Ztso"** strengthens the baseline memory model to TSO
  - TSO-only code is ***not*** backwards-compatible with RVWMO

ONGOING/FUTURE WORK

- Mixed-size, partially-overlapping memory acceses
- Formalize instruction fetches and FENCE.I TLB flushes and SFENCE.VMA, etc.

- Integration with other extensions (V, J, N, T, …)
- Integration with the ISA formalization task group's effort
- Cache flush/writeback/etc. operations
- (The task group logistics for all this are still TBD)

DOCUMENTATION & TOOLS

- Appendix A: two dozen pages explaining the details in plain English
- Appendix B: Two axiomatic models and one operational model, with associated tools (Alloy, herd, rmem)
- More than 7000 litmus tests online
  - (also to be used to test compliance)

Hart 0		Hart 1	
			li t1, 1
(a)	lw a0,0(s0)	(d)	lw a1, 0(s1)
(b)	fence rw,rw	(e)	lw a2, 0(a1)
(c)	sw s2,0(s1)	(f)	sw t1, 0(s0)
	Outcome:		a0=1, a1=t

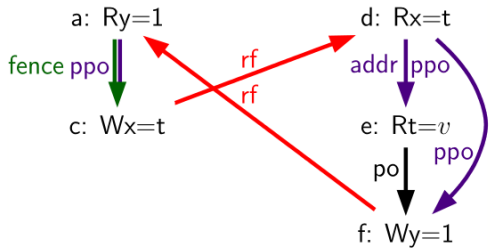
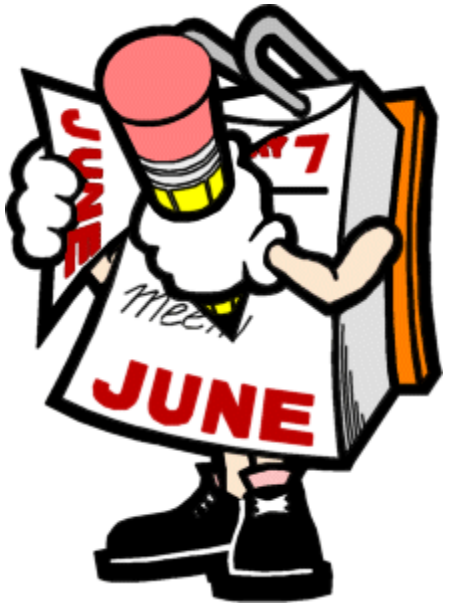


Figure A.16: Because of the address dependency from (d) to (e), (d) also precedes (f) (outcome forbidden)

MEMORY MODEL RATIFICATION TIMELINE

- Released for public review on 5/2/18
- Foundation requires at least 45 days for public review. This will end no earlier than 6/16/18.
- If you have comments or feedback:
  - send to isa-dev
  - send as a PR or issue on riscv-isa-manual GitHub repo
  - send to me directly



TOTAL STORE ORDERING (SPARC, X86, RVTSO)

Axiomatic

ppo := (program order) – W→R

acyclic(ppo U rfe U co U fr U fence)

acyclic(po_loc U rf U co U fr)

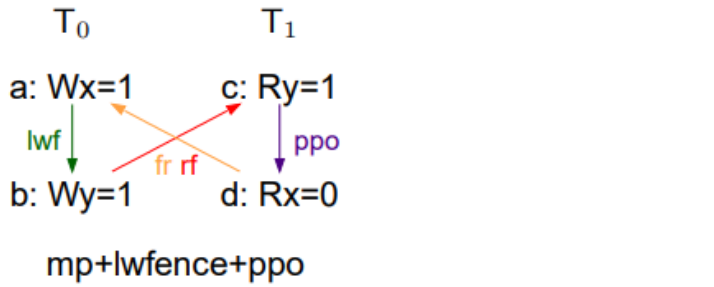


Fig. 8. The message passing pattern mp with lightweight fence and ppo (forbidden)

[Alglave et al., TOPLAS '09]

Operational

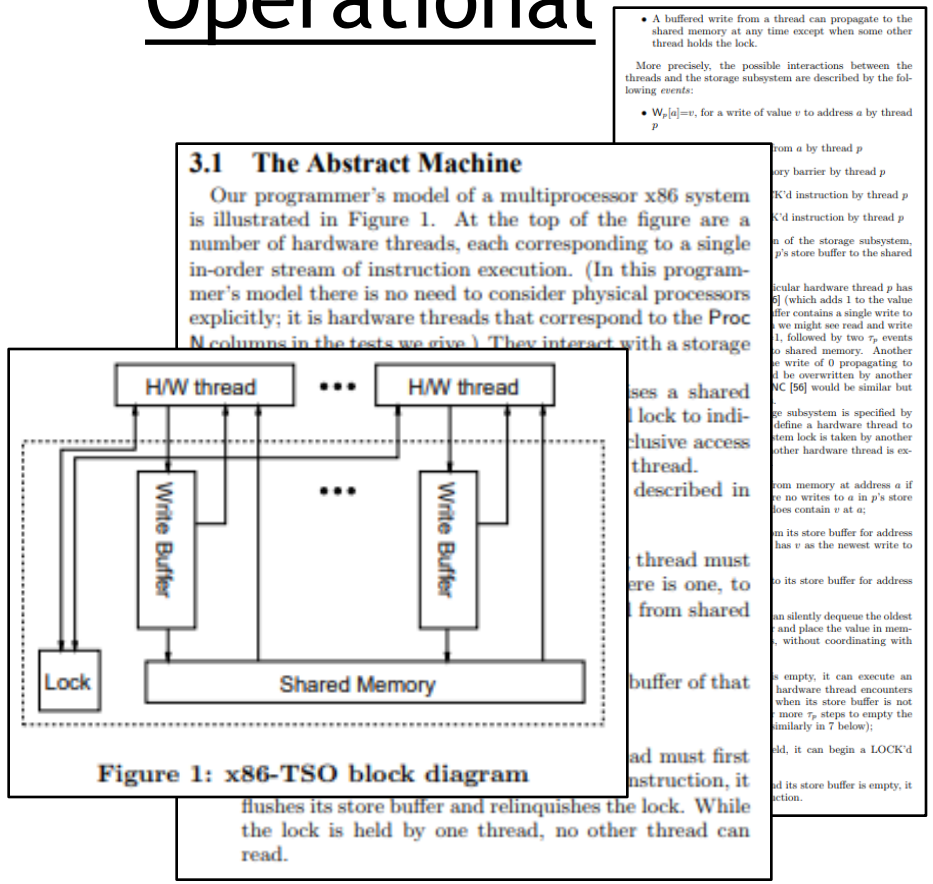


Figure 1: x86-TSO block diagram

3.1 The Abstract Machine

Our programmer's model of a multiprocessor x86 system is illustrated in Figure 1. At the top of the figure are a number of hardware threads, each corresponding to a single in-order stream of instruction execution. (In this programmer's model there is no need to consider physical processors explicitly; it is hardware threads that correspond to the Proc N columns in the tests we give.) They interact with a storage

flushes its store buffer and relinquishes the lock. While the lock is held by one thread, no other thread can read.

[Sewell et al., CACM '10]

RVWMO

Axiomatic (App. B.2)

ppo := (13 rules, on next slide)

acyclic(ppo U rfe U co U fr)

acyclic(po_loc U rf U co U fr)

Initial values: 0(s0)=1; 0(s1)=1

Hart 0		Hart 1	
(a)	ld a0,0(s0)	(e)	ld a3,0(s2)
(b)	lr a1,0(s1)	(f)	sd a3,0(s0)
(c)	sc a2,a0,0(s1)		
(d)	sd a2,0(s2)		

Outcome: a0=0, a3=0

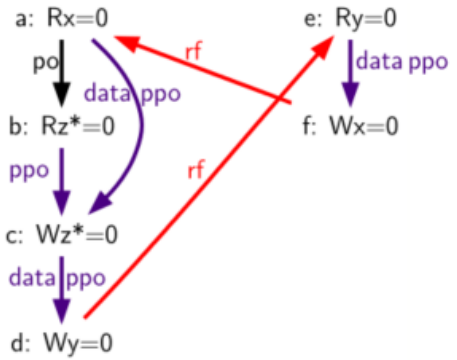
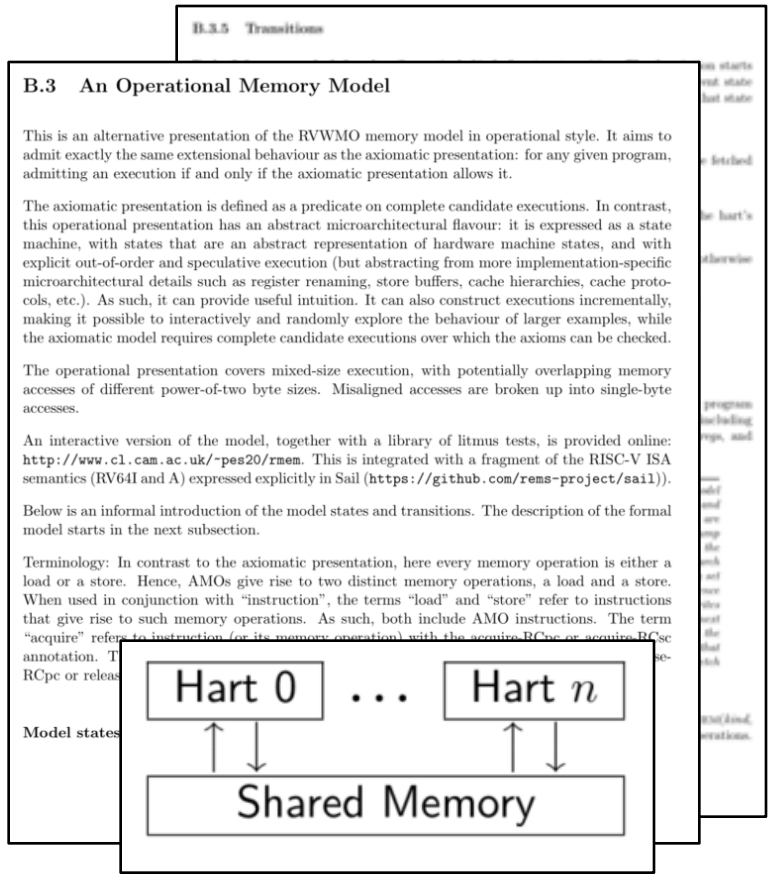


Figure A.13: A variant of the LB litmus test (outcome forbidden)

Operational (App. B.3)



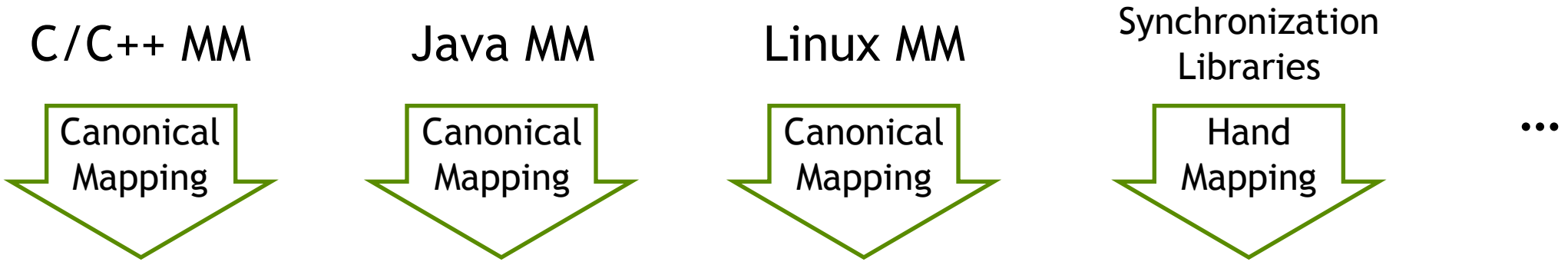
MULTI-COPY ATOMICITY

A load may only return a value from:

- An earlier store from the same hart (“hardware thread”)
- A store that is globally visible

In other words, a store may not “peek” into a neighbor hart’s private store buffer

WHO FEELS THE PAIN?



RISC-V ISA Memory Consistency Model

- Misconception: end users will have to deal with the memory model
- Reality: end users rarely interact with the ISA memory model directly
- Burden falls instead on library/compiler writers and microarchitects

MEMORY MODEL TASK GROUP PROGRESS

- **May 2017 Workshop:** Formed the task group

(…debate…)

- **November 2017 Workshop:** Settled on the basics
  - RVWMO baseline, and optional RVTSO extension

(…refinement…)

- **May 2018 Workshop: released for ratification!**
  - Public review period runs May 2 through June 16

RISC-V MEMORY MODEL SPECIFICATION

- Chapter 6: RISC-V Weak Memory Ordering (“RVWMO”)

- Chapter 20: “Zam” Std. Extension for Misaligned AMOs
- Chapter 21: “Ztso” Std. Extension for Total Store Ordering

- Appendix A: Explanatory Material and Litmus Tests
- Appendix B: Formal Memory Model Specifications

RVWMO RULES IN A NUTSHELL

- **Load Value Axiom:** each byte of each load *i* returns the value written to that byte by the store that is the latest in global memory order among the following stores:
  1. Stores that write that byte and that precede *i* in the global memory order
  2. Stores that write that byte and that precede *i* in program order
- **Atomicity Axiom:** no store from another hart can appear in the global memory order between a paired LR and successful SC
  - *(this axiom simplified here for clarity…see spec for complete definition)*
- **Progress Axiom:** no memory operation may be preceded in the global memory order by an infinite sequence of other memory operations