A Memory Consistency Model For RISC-V
Formally Evaluated with TriCheck

Caroline Trippel

Princeton University

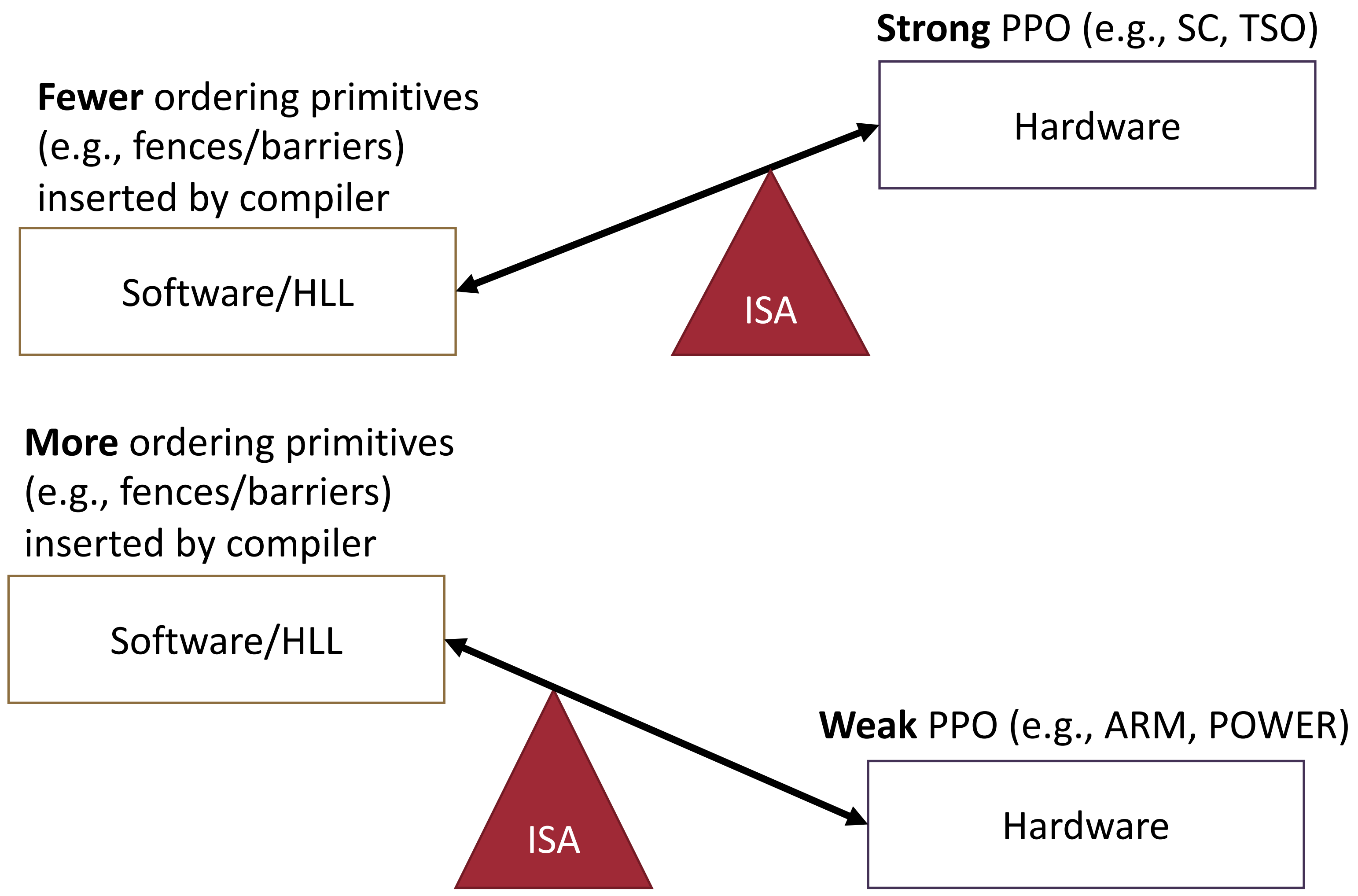
November 29, 2016

Caroline Trippel, Yatin Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. "TriCheck: Memory Model Verification at the Trisection Software, Hardware, and ISA". In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS '17).

PRINCETON UNIVERSITY

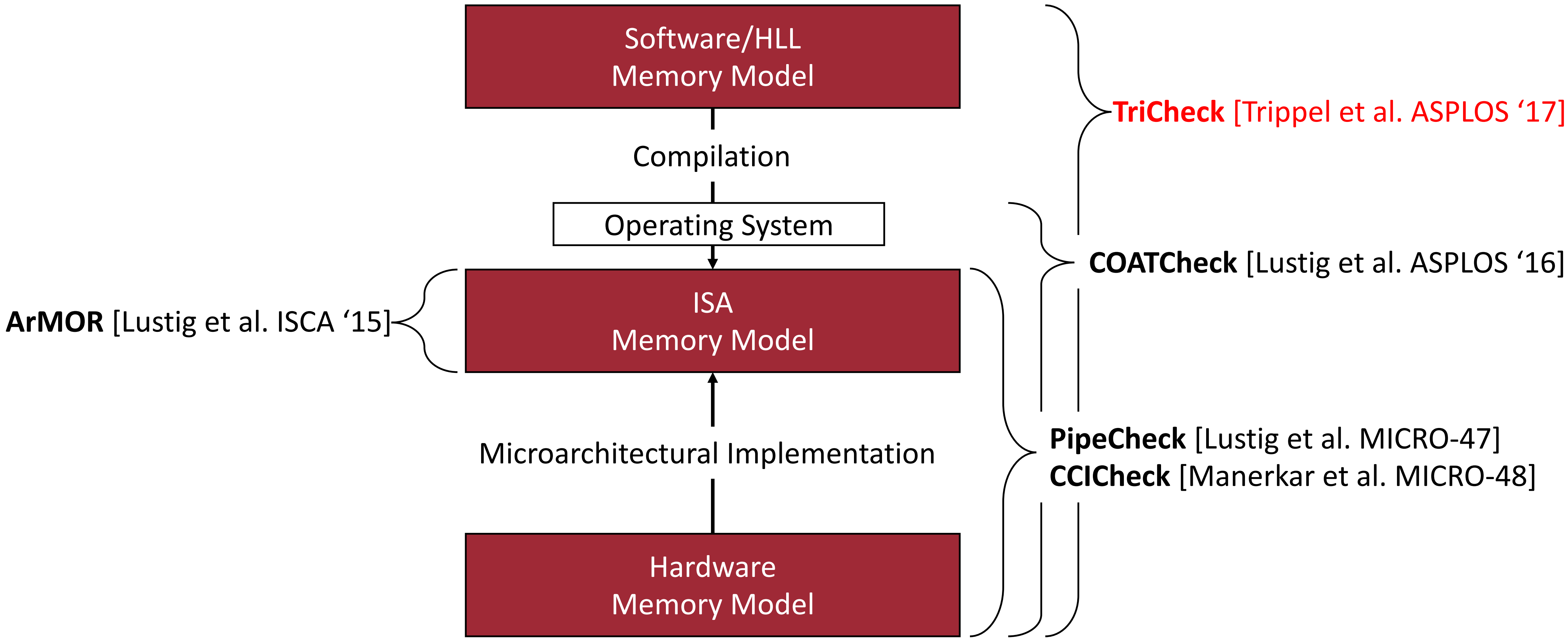
Role of the Instruction Set Architecture (ISA)

- Introduced in 1964 by IBM
  - 1 set of software
  - >1 hardware implementations
- Definitive spec. of hardware as seen by software:
  - Specification of what hardware must implement
  - Target for compiler translation


Strong PPO (e.g., SC, TSO)
Hardware
Fewer ordering primitives (e.g., fences/barriers) inserted by compiler
Software/HLL
ISA
More ordering primitives (e.g., fences/barriers) inserted by compiler
Software/HLL
ISA
Weak PPO (e.g., ARM, POWER)
Hardware

PRINCETON UNIVERSITY

Our Work: Memory Consistency Model Verification


Software/HLL Memory Model
Compilation
Operating System
ISA Memory Model
Microarchitectural Implementation
Hardware Memory Model
TriCheck [Trippel et al. ASPLOS '17]
COATCheck [Lustig et al. ASPLOS '16]
ArMOR [Lustig et al. ISCA '15]
PipeCheck [Lustig et al. MICRO-47]
CCICheck [Manerkar et al. MICRO-48]

Memory Models Bugs Observed in Practice

ARM Read-after-Read Hazard [Alglave et al. TOPLAS '14]

- Ambiguous ISA spec. regarding same-address Ld→Ld ordering
  - ARM compilers did not insert synchronization primitives (e.g., fences/barriers)
  - Some ARM implementations relaxed same-address Ld→Ld ordering (e.g., Cortex-A9, Snapdragon 805)
- C/C++ atomics require same-address Ld→Ld ordering
  - ARM issued errata[1]: Rewrite compilers to insert fences (with performance penalties)

We've identified and characterized flaws in the current RISC-V memory model (i.e., the memory model defined in the current manual) [Trippel et al. ASPLOS '17]

PRINCETON UNIVERSITY

Note that the modifications to fix these issues will be mostly compatible with current implementations.

Outline

- Role of Memory Models in ISAs
- What Should We Require From the Hardware?
- What Fences/Barriers Do We Need to Support C/C++?
- TriCheck Framework for Full-Stack Memory Model Verification
- On-Going Work & Conclusions

PRINCETON UNIVERSITY

Sequential Consistency

- Memory models specify the allowed behavior of a multithreaded program executing with shared memory

- First defined by [Lamport 1979], execution is the same as if:
  - **(R1)** Memory ops of each processor appear in program order
  - **(R2)** Memory ops of all processors were executed in some global sequential order

Program

Thread 0	Thread 1
x=1	r1=y
y=1	r2=x

Legal Executions

x=1	x=1	x=1	r1=y	r1=y	r1=y
y=1	r1=y	r1=y	r2=x	x=1	x=1
r1=y	y=1	r2=x	x=1	r2=x	y=1
r2=x	r2=x	y=1	y=1	y=1	r2=x

PRINCETON UNIVERSITY

Two Categories of Memory Model Relaxation

Preserved Program Order: Defines program orderings that hardware must preserve by default

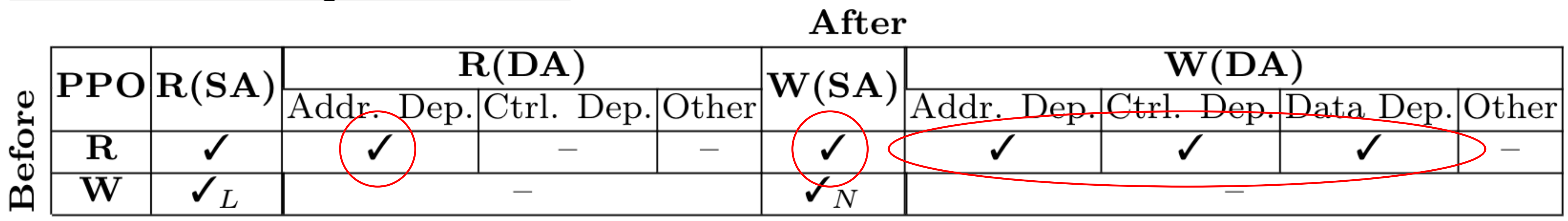
Store Atomicity: Defines order in which stores become visible to cores

- Multiple-copy atomic: E.g., monolithic memory
  - All cores see store simultaneously
- Read-Own-Write-Early-multiple-copy atomic: E.g., private store buffer
  - Storing core can read its own store before other cores
  - Stores made visible to all remote cores simultaneously
- Non-multiple-copy atomic: E.g., shared store buffer
  - Storing core can read its own store before other cores
  - Store is made visible to some remote cores before others

PRINCETON UNIVERSITY

RISC-V Proposed Preserved Program Order and Store Atomicity

Preserved Program Order:



After

	PPO	R(SA)	R(DA)			W(SA)	W(DA)			
			Addr. Dep.	Ctrl. Dep.	Other		Addr. Dep.	Ctrl. Dep.	Data Dep.	Other
Before	R	✓	✓	–	–	✓	✓	✓	✓	–
	W	✓$_L$	–			✓$_N$	–			

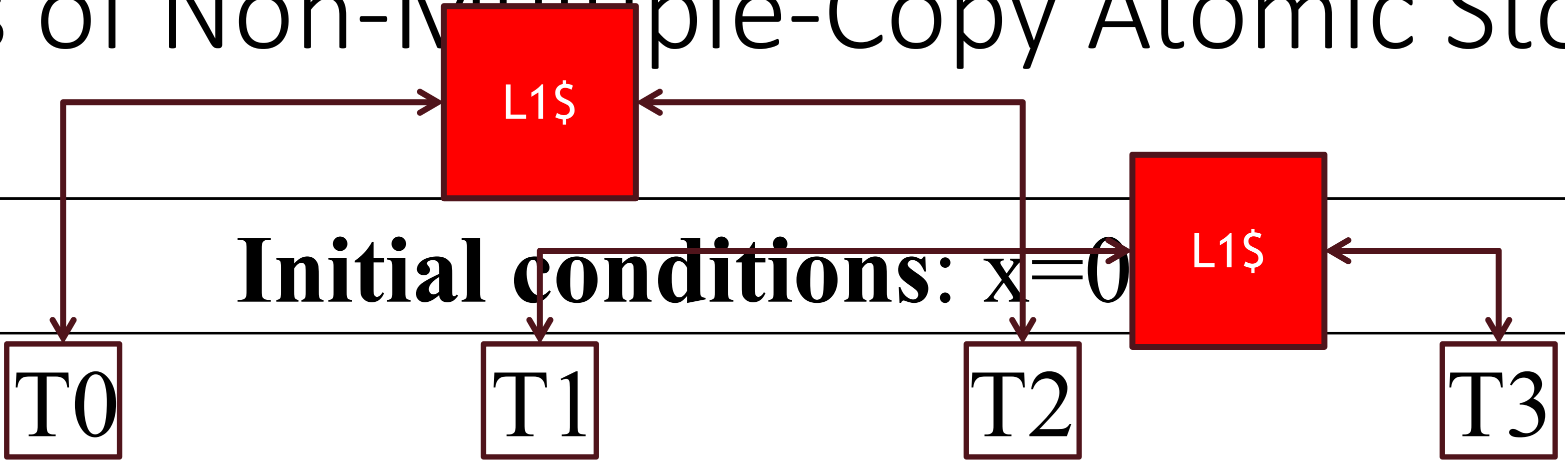
Store Atomicity:

Non-multiple-copy atomic:

- Storing core can read its own store before other cores
- Store is made visible to some remote cores before others

PRINCETON UNIVERSITY

Effects of Non-Multiple-Copy Atomic Stores



Initial conditions: x=0			
T0	T1	T2	T3
st [x] ← 1	st [y] ← 1	ld x → [r0] F R, R ld y → [r1]	ld y → [r2] F R, R ld x → [r3]
Non-SC Outcome: r0=1, r1=0, r2=1, r3=0			

This outcome corresponds to the case in which the stores on threads T0 and T1 arrive to threads T2 and T3 in different orders

PRINCETON UNIVERSITY

Why Allow Non-Multiple-Copy Atomic Stores?

- Commercial ISAs allow non-multiple-copy atomic stores (e.g. ARM, POWER)
- RISC-V is intended to be integrated with other vendor ISAs
- Potential deployment in non-multiple-copy atomic memory systems
- If sharing memory system, awareness that stores may be observed in orders that differ from other cores

PRINCETON UNIVERSITY

Outline

- Role of Memory Models in ISAs
- What Should We Require From the Hardware?
- What Fences/Barriers Do We Need to Support C/C++?
- TriCheck Framework for Full-Stack Memory Model Verification
- On-Going Work & Conclusions

PRINCETON UNIVERSITY

Fences to Restore Multiple-Copy Atomicity

Initial conditions: x=0, y=0			
T0	T1	T2	T3
st [x] ← 1	st [y] ← 1	ld x → [r0] pscF RW, RW ld y → [r1]	ld y → [r2] pscF RW, RW ld x → [r3]
Non-SC Outcome: r0=1, r1=0, r2=1, r3=0			

Predecessor-/Successor- Cumulative Fence: Necessary to Restore SC for Non-Multiple-Copy Atomic Memory Systems

PRINCETON UNIVERSITY

Other Fences/Barriers/Ordering Primitives

- Baseline Memory Model
  - PPO requires same-address R-R order to be maintained
  - PPO requires order to be maintained between most dependent instructions
  - Predecessor-/Successor-Cumulative F RW, RW; F IO, IO; F IORW, IORW
- Baseline + Atomics Extension
  - Predecessor-Cumulative F RW, W

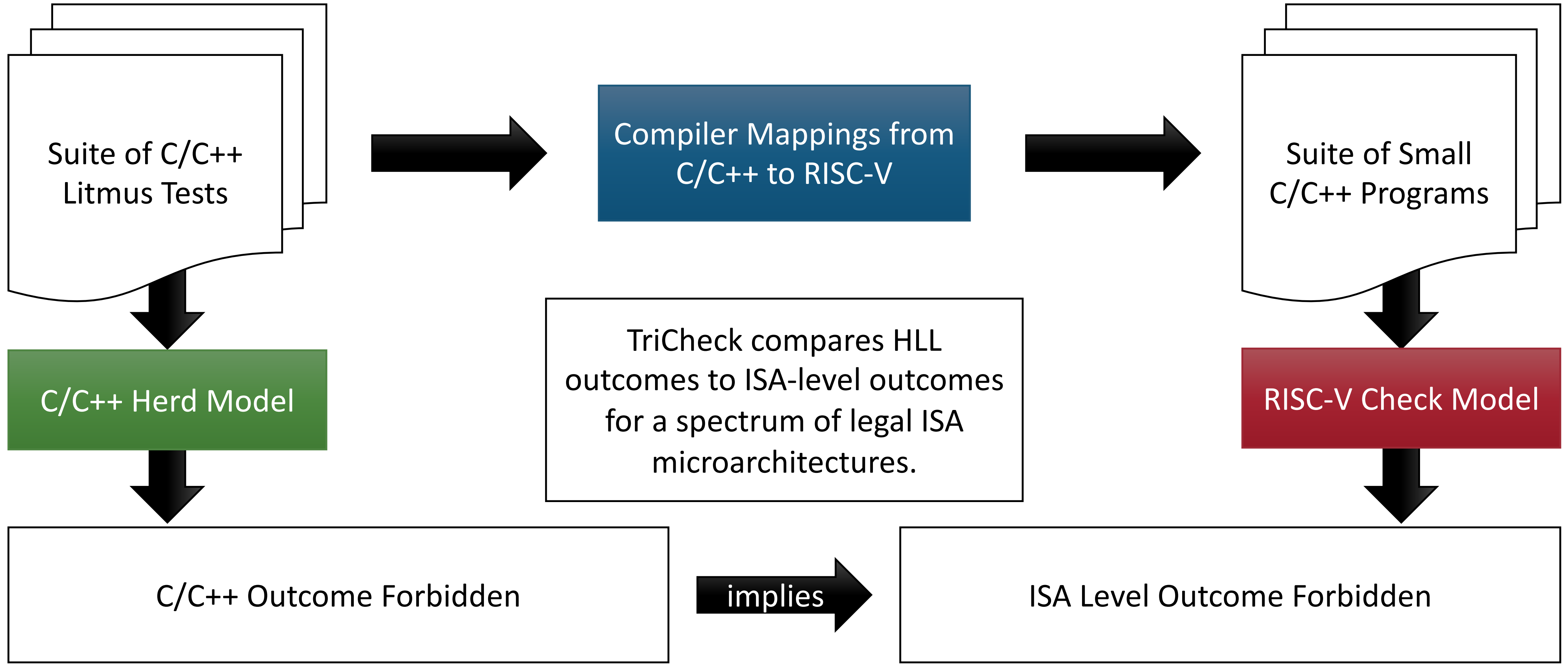
PRINCETON UNIVERSITY

Outline

- Role of Memory Models in ISAs
- What Should We Require From the Hardware?
- What Fences/Barriers Do We Need to Support C/C++?
- TriCheck Framework for Full-Stack Memory Model Verification
- On-Going Work & Conclusions

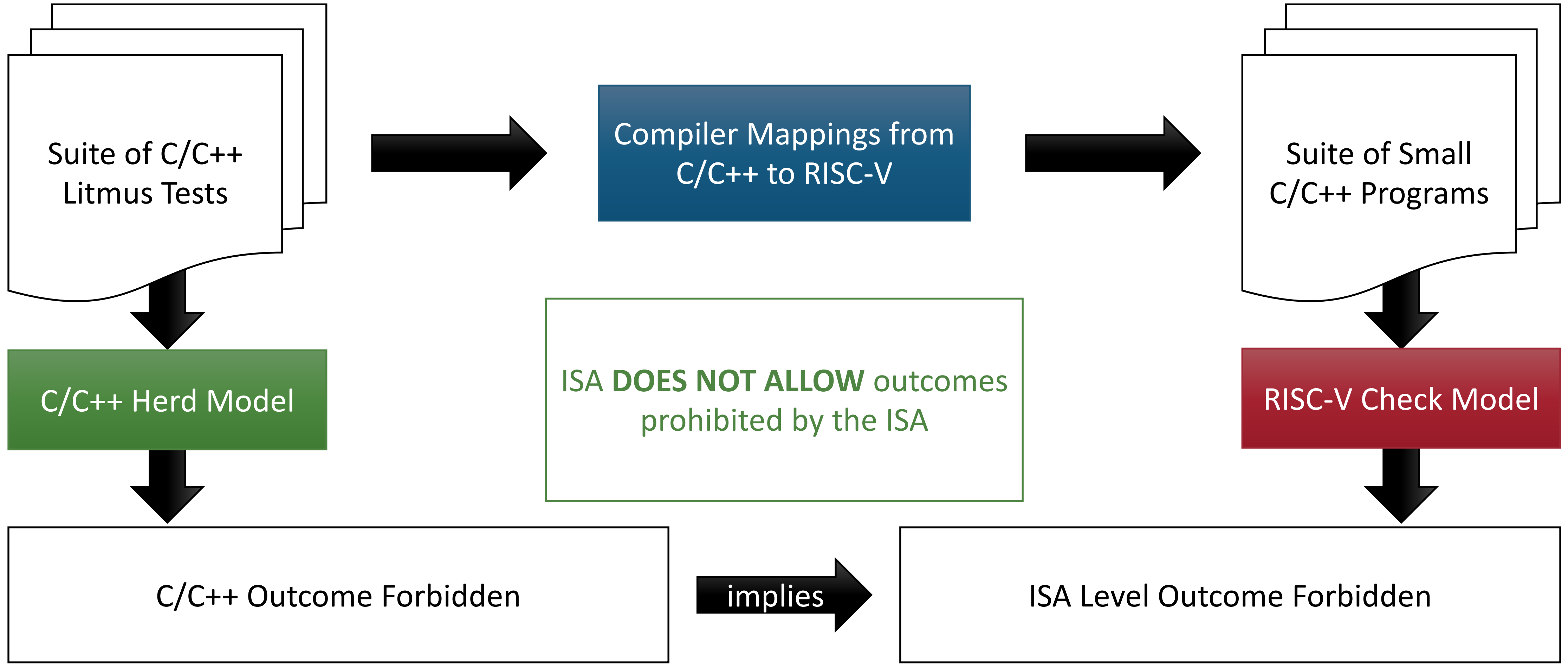
PRINCETON UNIVERSITY

TriCheck Full-Stack Verification Framework


Suite of C/C++ Litmus Tests
Compiler Mappings from C/C++ to RISC-V
Suite of Small C/C++ Programs
C/C++ Herd Model
RISC-V Check Model
C/C++ Outcome Forbidden
implies
ISA Level Outcome Forbidden

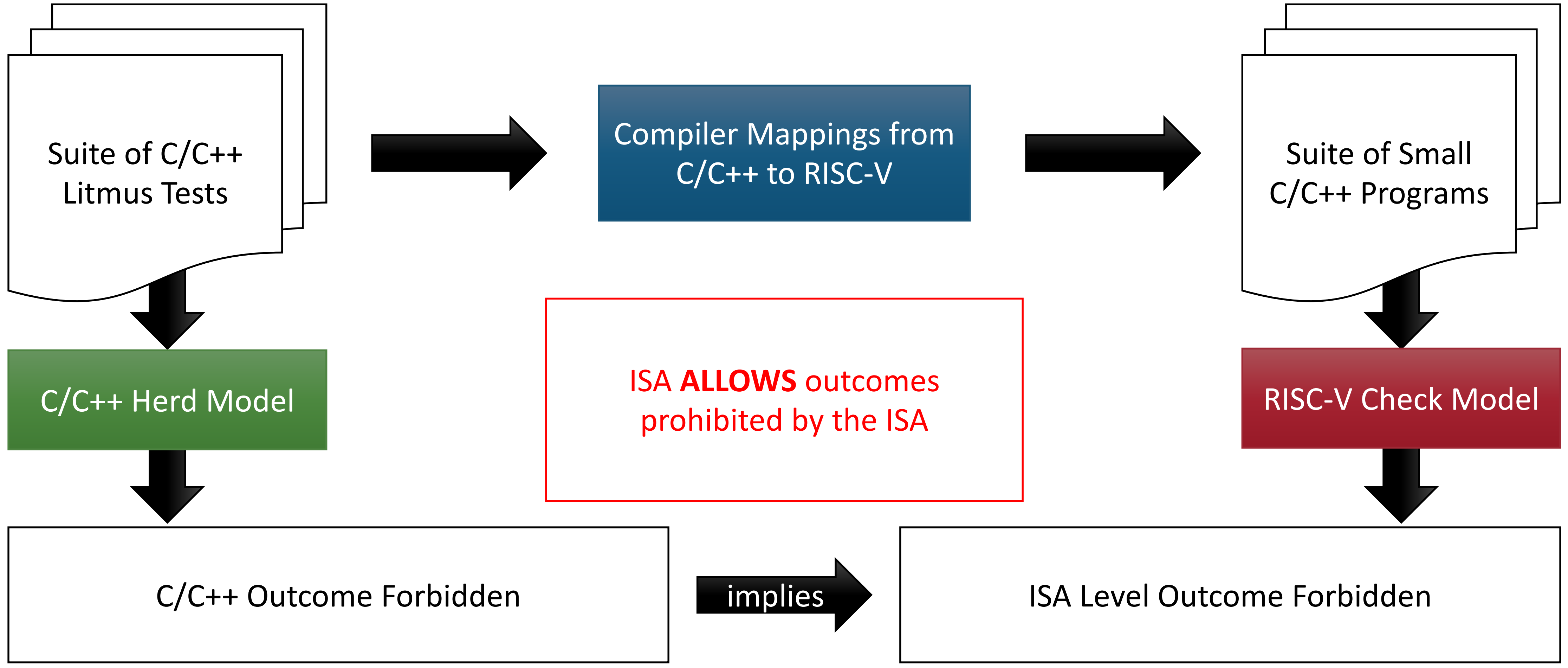
TriCheck compares HLL outcomes to ISA-level outcomes for a spectrum of legal ISA microarchitectures.

TriCheck Full-Stack Verification Framework



Suite of C/C++ Litmus Tests
Compiler Mappings from C/C++ to RISC-V
Suite of Small C/C++ Programs
C/C++ Herd Model
ISA **DOES NOT ALLOW** outcomes prohibited by the ISA
RISC-V Check Model
C/C++ Outcome Forbidden
implies
ISA Level Outcome Forbidden

TriCheck Full-Stack Verification Framework

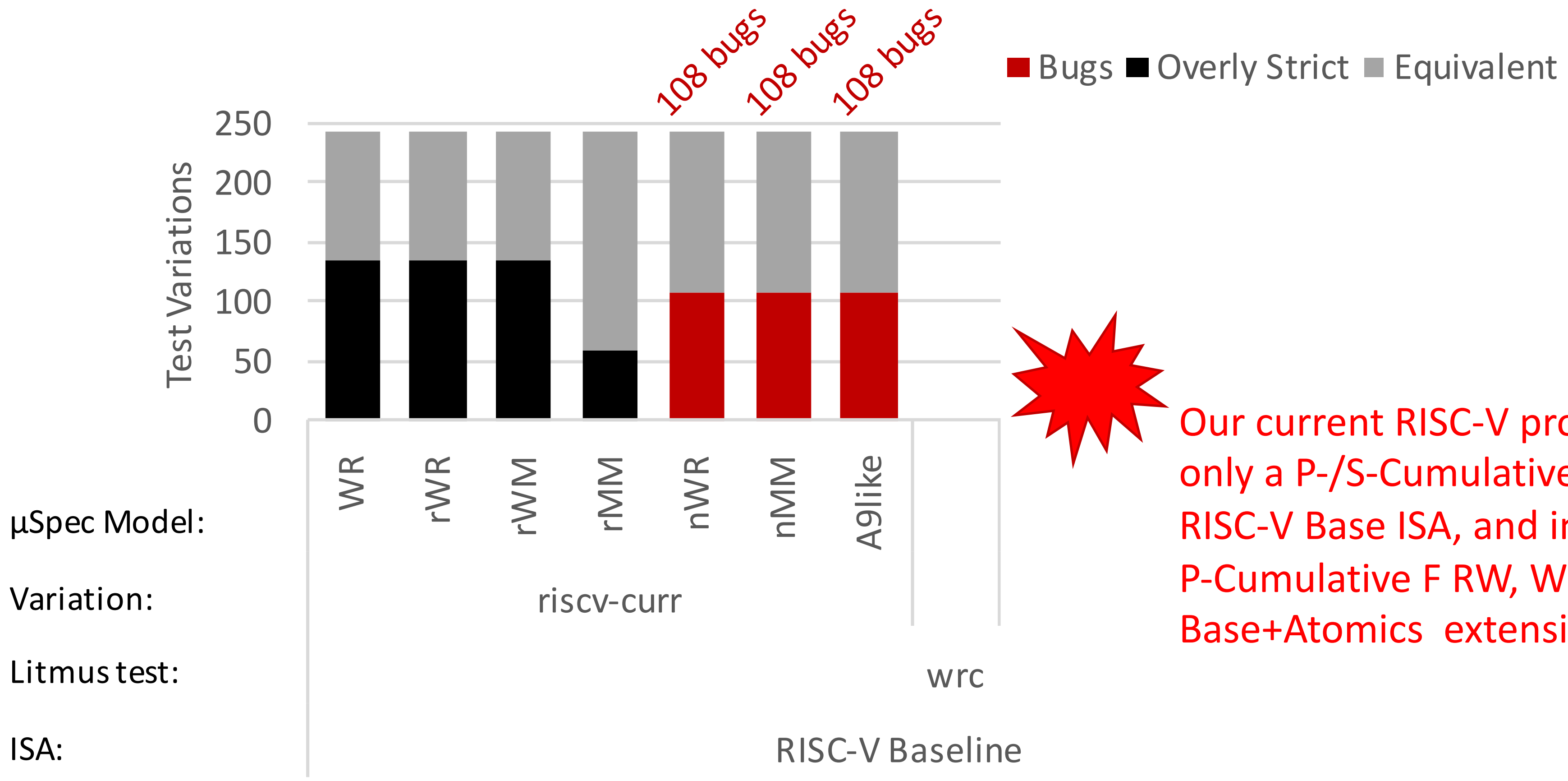

Suite of C/C++ Litmus Tests
Compiler Mappings from C/C++ to RISC-V
Suite of Small C/C++ Programs
C/C++ Herd Model
ISA **ALLOWS** outcomes prohibited by the ISA
RISC-V Check Model
C/C++ Outcome Forbidden
implies
ISA Level Outcome Forbidden

PRINCETON UNIVERSITY

RISC-V Base: Lack of Cumulative Fences

Initial conditions: x=0, y=0		
T0	T1	T2
a: sw x1, (x5)	b: lw x2, (x5)	e: lw x3, (x6)
	c: fence rw, w	f: fence r, rw
	d: sw x2, (x6)	g: lw x4, (x5)

- Base RISC-V ISA lacks cumulative fences
  - Minimally, the ISA requires a Predecessor-/Successor Cumulative F RW, RW
  - Cannot fix bugs by modifying compiler currently



Our current RISC-V proposal requires only a P-/S-Cumulative F RW, RW in the RISC-V Base ISA, and includes a weaker P-Cumulative F RW, W Fence in the Base+Atomics extension.

PRINCE UNIVER

Outline

- Role of Memory Models in ISAs
- What Should We Require From the Hardware?
- What Fences/Barriers Do We Need to Support C/C++?
- TriCheck Framework for Full-Stack Memory Model Verification
- On-Going Work & Conclusions

PRINCETON UNIVERSITY

On-Going Work & Conclusions

- We have formulated an English language diff. of the current spec. with our proposed changes
- Currently we are constructing a formal model in Herd [Alglave et al., TOPLAS '14] of our proposed memory model modifications
- Memory model design choices are complicated and involve reasoning about the subtle interplay between many diverse features
- Defining an ISA specification in light of the evaluation of a single microarchitecture is not sufficient
- TriCheck is generalizable to any ISA and uncovered/quantified flaws in the RISC-V memory mode.

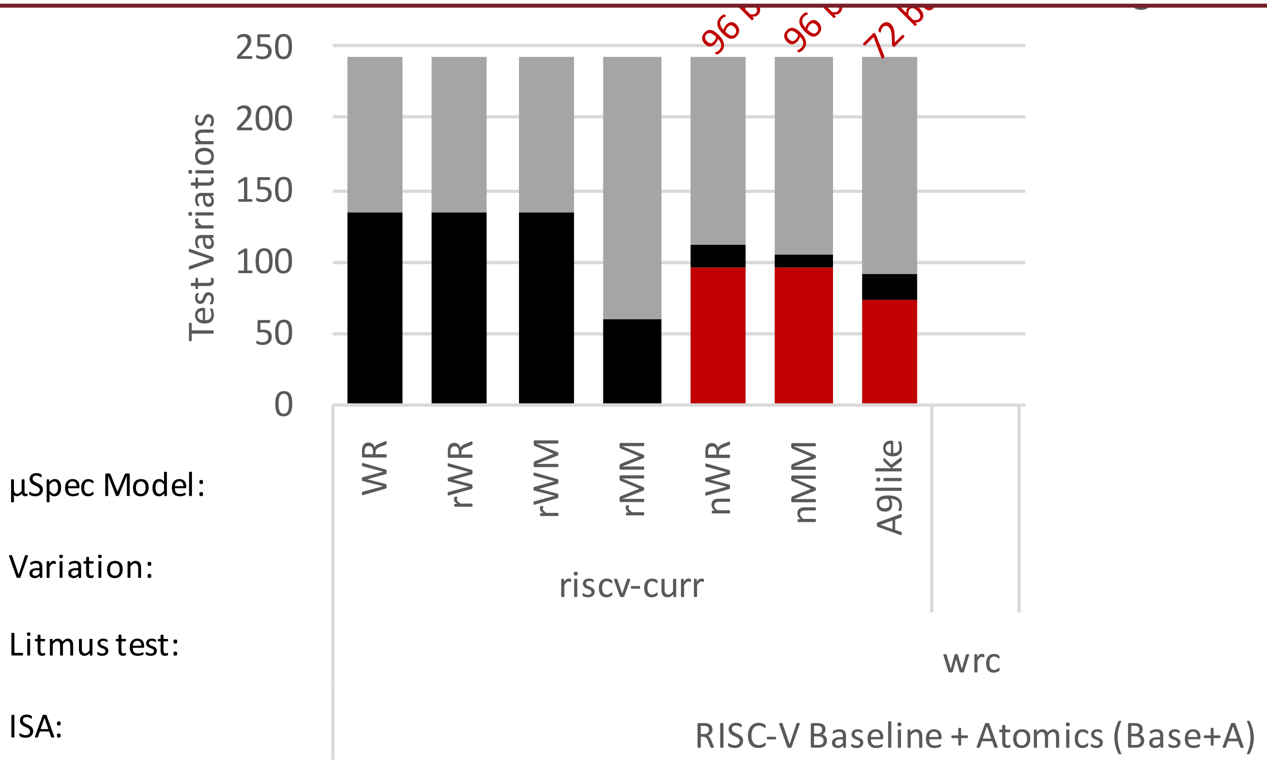
PRINCETON UNIVERSITY

ctrippel@princeton.edu
http://check.cs.princeton.edu/

RISC-V Base+A: Lack of Transitive Releases

Initial conditions: x=0, y=0		
T0	T1	T2
a: sw x1, (x5)	b: lw x2, (x5)	d: amoadd.w.aq x0, x3, (x6)
	c: amoswap.w.rl x2, x0, (x6)	e: lw x4, (x5)
Forbidden HLL Outcome: x1=1, x2=1, x3=1, x4=0		

- Base+A RISC-V ISA lacks transitive releases
  - i.e., RISC-V acquires do not synchronize with RISC-V releases as required by C/C++
  - AMO.rl and stronger AMO.aq.rl are both insufficeint
  - Cannot fix bugs by modifying compiler
- <u>Our solution:</u> redefine release operations in the Base+A RISC-V ISA to be transitive

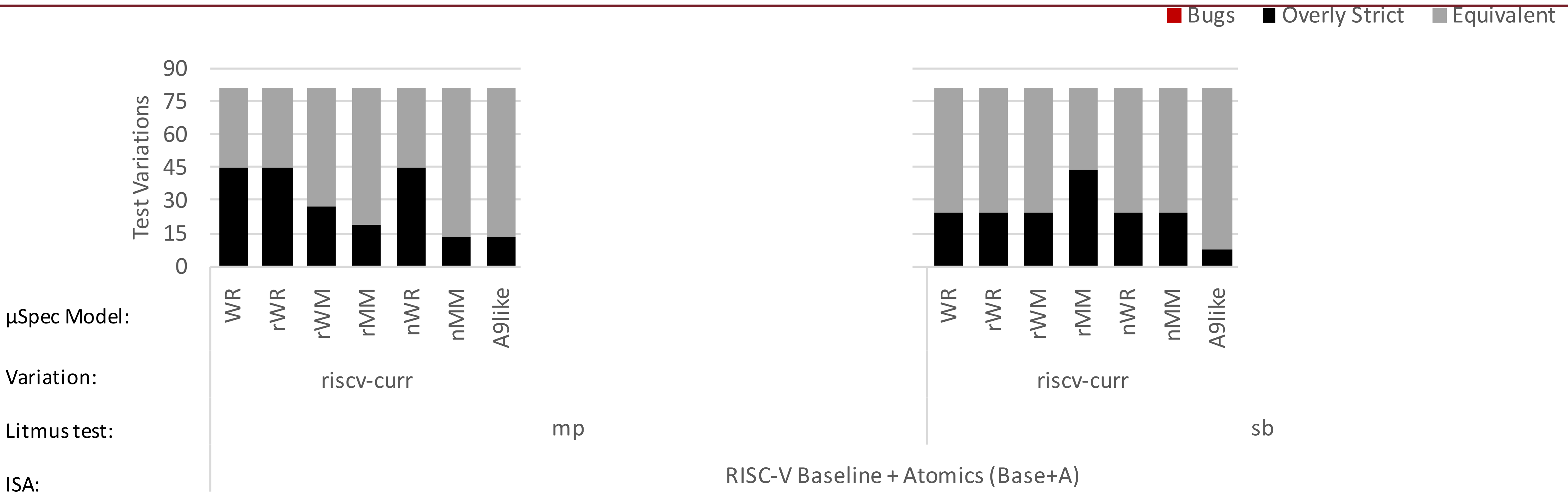


RISC-V Base+A: No Roach-Motel Movement for SC Atomics

Initial conditions: x=0, y=0	
T0	T1
a: amoswap.w.aq.rl x1, x0, (x4)	c: amoadd.w.aq.rl x0, x2, (x5)
b: sw x1, (x5)	d: amoadd.w.aq.rl x0, x3, (x4)
Allowed Non-SC Outcome: x1=1, x2=1, x3=0	

Initial conditions: x=0, y=0

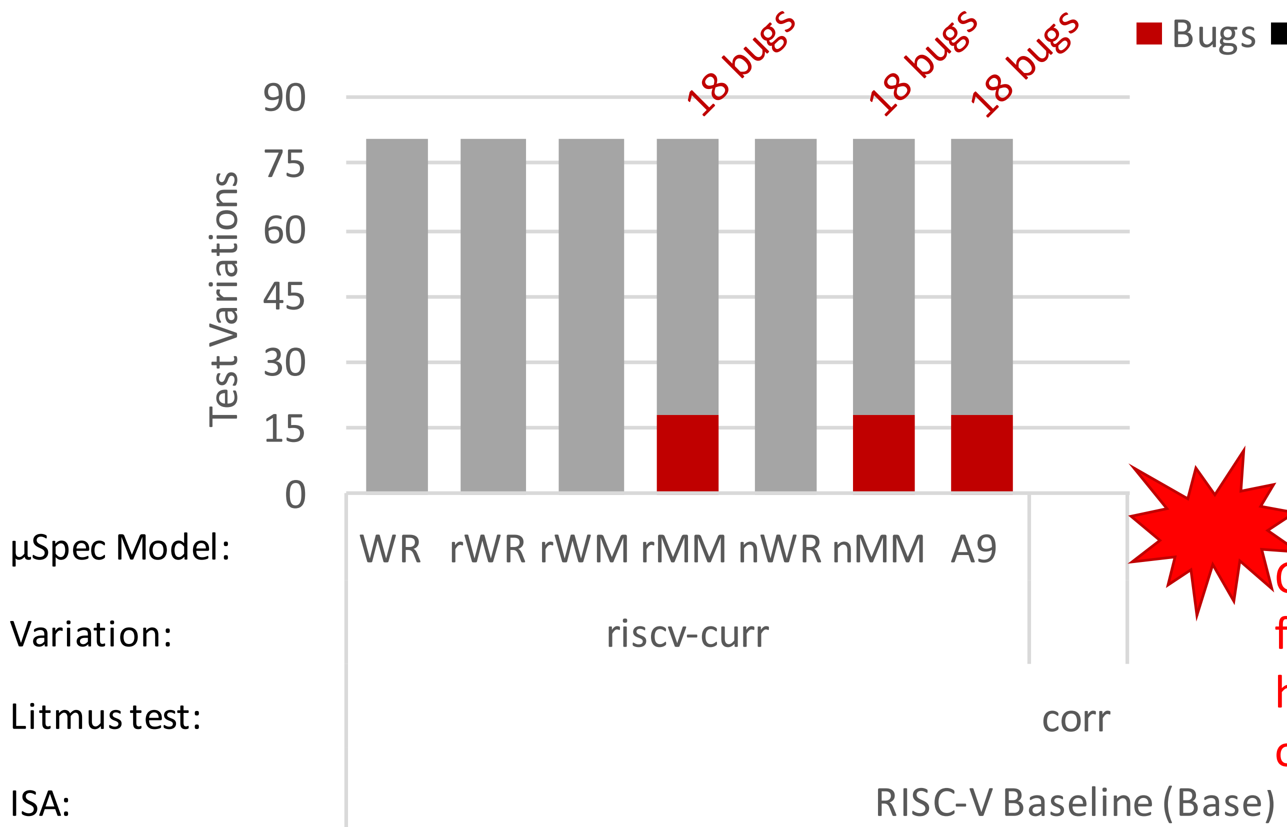
- RISC-V SC loads and stores require both aq and rl bits set on AMOs
  - Operation has acquire and release semantics
  - Prohibits roach-motel movement
- <u>Our solution:</u> add an sc bit for implementing AMO.aq.sc and AMO.rl.sc instructions which are capable of implementing C/C++ SC loads and stores



RISC-V Base: Same Address Ld→Ld Re-Ordering

Initial conditions: x=0, y=0	
T0	T1
a: sw x1, (x5)	c: lw x3, (x5)
b: sw x2, (x5)	d: lw x4, (x5)

- Base RISC-V ISA includes F R, R
  - Possible to fix bugs by modifying compiler with potential performance penalty
  - 20.3% preliminary estimate of fence insertion performance penalty for ARM
- Our solution: modify Base RISC-V memory model to require same-address Ld→Ld ordering



PRINCETON UNIVERSITY

Our current RISC-V proposal elimites F R, R from the RISC-V Base ISA, and requires hardware to enforce same-address Ld→Ld order by default.

Re-ordering Dependent Operations

- RISC-V does not require ordering for dependent instructions
- Many commercial ISAs – x86, ARM, Power – respect dependencies
  - Can also be used as lightweight synchronization
- Explicit synchronization/fences needed when dependency ordering is required but not enforced by default, e.g., Linux
  - Macro read_barrier_depends() optionally inserts a barrier
    - Inserts a fence for Alpha, which does not respect dependencies[1]
    - Inserts nothing for RISC-V, which does not respect dependencies[2]
- Our solution: modify Base RISC-V memory model to require the preservation of dependency orderings.

[1]Linus Torvalds et al. Linux kernel, 2016. https: //github.com/torvalds/linux/blob/master/arch/alpha/include/asm/barrier.h

[2]RISC-V Foundation. RISC-V port of Linux kernel, 2016. https://github.com/riscv/riscv-linux/blob/master/rch/riscv/include/asm/barrier.h

PRINCETON UNIVERSITY

ctrippel@princeton.edu
http://check.cs.princeton.edu/

PRINCETON UNIVERSITY