TriCheck: Memory Model Verification at the Trisection of Software, Hardware, and ISA

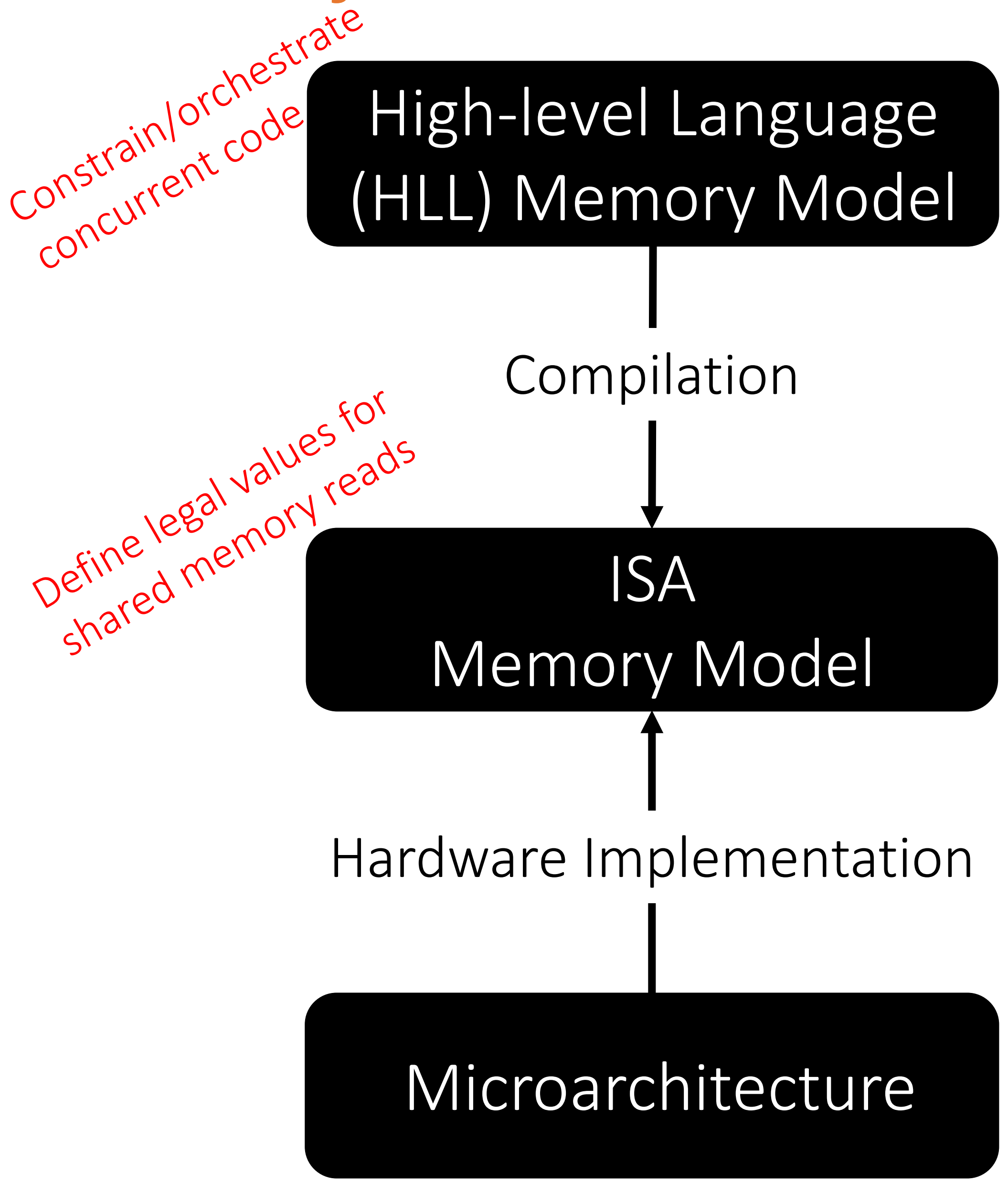
Caroline Trippel, Yatin A. Manerkar, Daniel Lustig*, Michael Pellauer*, Margaret Martonosi

Princeton University *NVIDIA

ASPLOS 2017

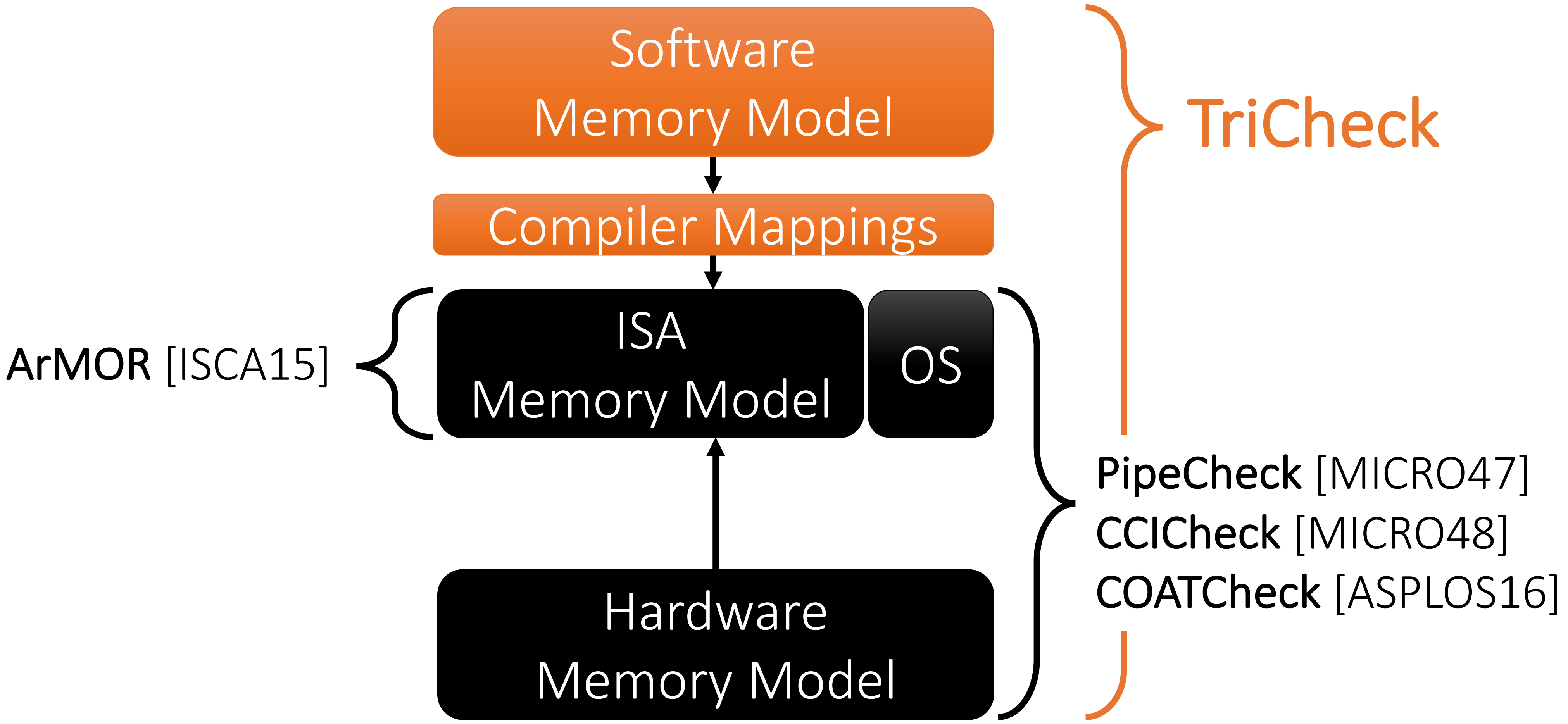
http://check.cs.princeton.edu/

Memory Models in the Hardware-Software Stack


Constrain/orchestrate concurrent code
High-level Language (HLL) Memory Model
Compilation
Define legal values for shared memory reads
ISA Memory Model
Hardware Implementation
Microarchitecture

- **What can go wrong?**
  - Ill-specified HLL memory model
  - Incorrect HLL→ISA compilation
  - Inadequate ISA specification
  - Incorrect hardware implementation
- **Current techniques verify only portions of stack**
  - Compiler mappings from HLL to ISA
  - Validity of hardware implementation

Our Work: Memory Consistency Model Verification



Why is TriCheck Necessary?

- Memory model bugs are real and problematic!
  - ARM Read-after-Read Hazard [Alglave et al. TOPLAS14]
  - RISC-V ISA is currently incompatible with C11
  - C11→POWER/ARMv7 “trailing-sync” compiler mapping **[Batty et al. POPL ‘12]**
  - C11→POWER/ARMv7 “leading-sync” compiler mapping [Lahav et al. PLDI17]

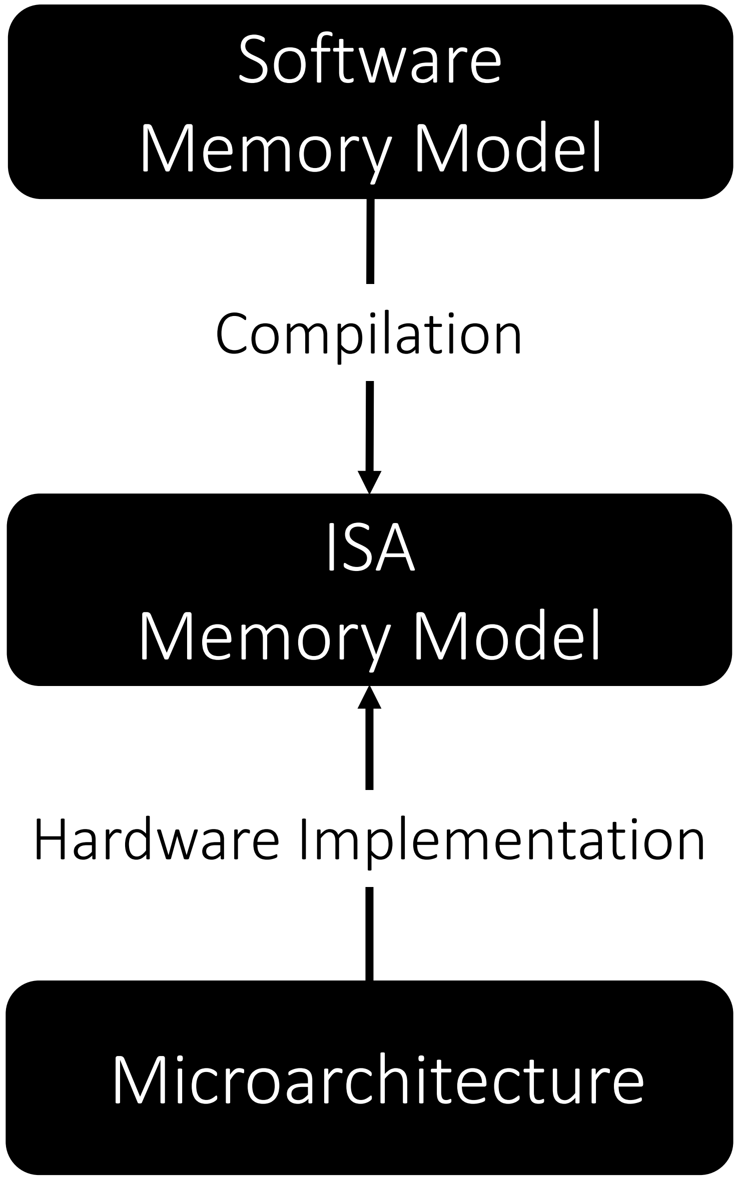
This work

- ISAs are an important and still-fluid design point!
  - Often, ISAs designed in light of desired HW optimizations
  - ISA places some constraints on hardware and some on compiler
  - Many industry memory models are still evolving: C11, ARMv7 vs. ARMv8
  - New ISAs are designed, e.g., RISC-V CPUs, specialized accelerators
- Correctness requires cooperation of the whole stack

Outline

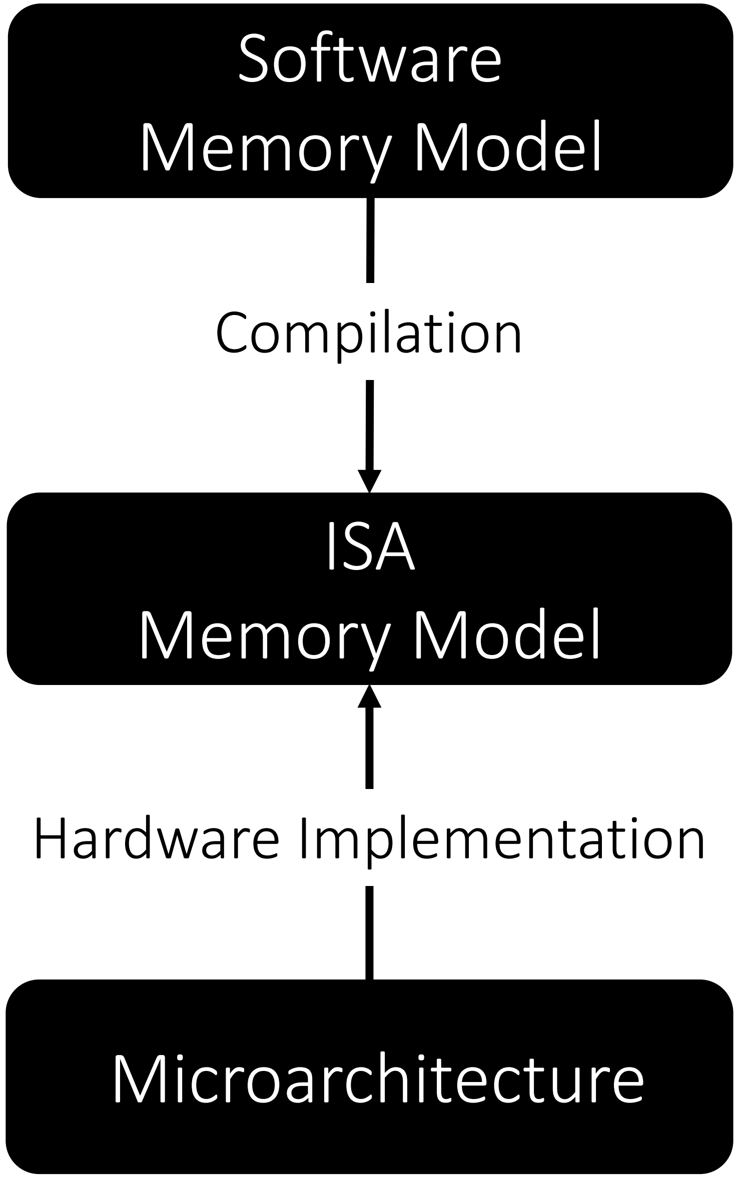
- Memory Consistency Model Verification
- Full-Stack Verification: Motivating Example
- TriCheck Framework for Full-Stack Memory Model Verification
- Bugs Found with TriCheck: RISC-V Case Study and Compiler Mappings
- Ongoing Work & Conclusions

ARM Read-Read Hazard



ARM Cortex-A9

ARM Read-Read Hazard

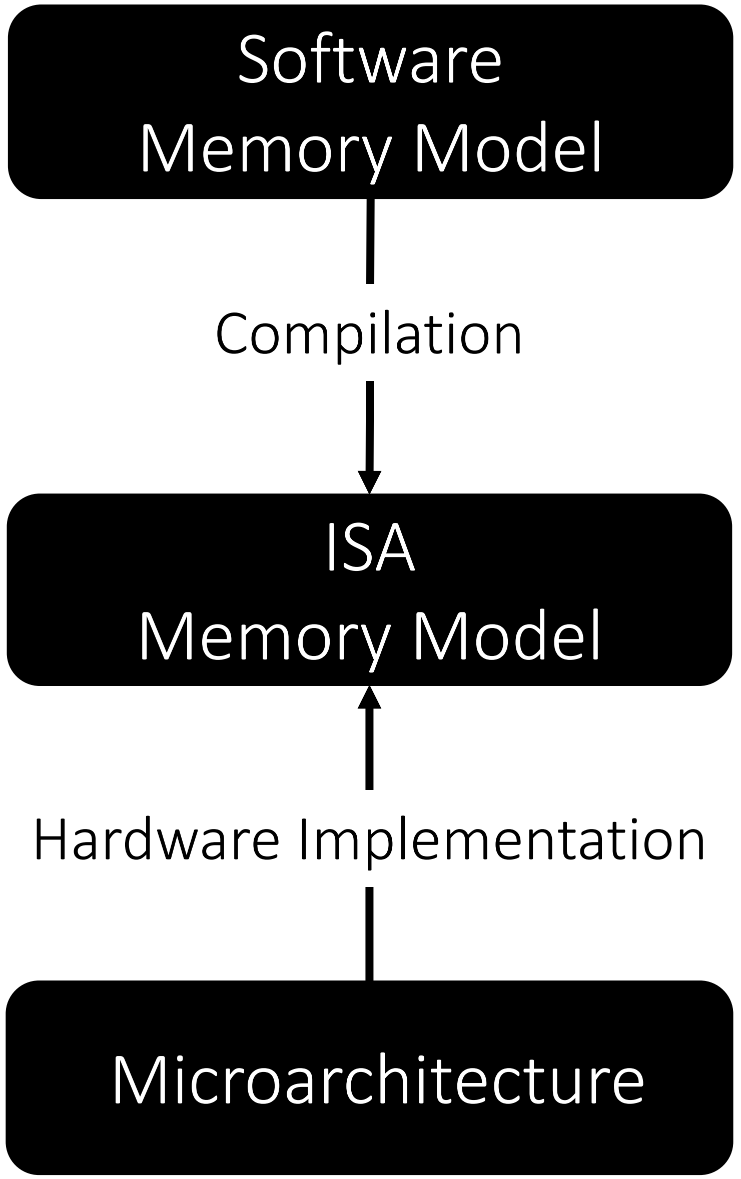


Which HLL(s) to support?

C11/C++11	ARMv7
st(rlx)	STR
ld(rlx)	LDR
ld(acq)	LDR; DMB
…	…

ARM Cortex-A9

ARM Read-Read Hazard



How does this affect real programs?

Loading data through a pointer

Initial conditions: data=0, *ptr=&data

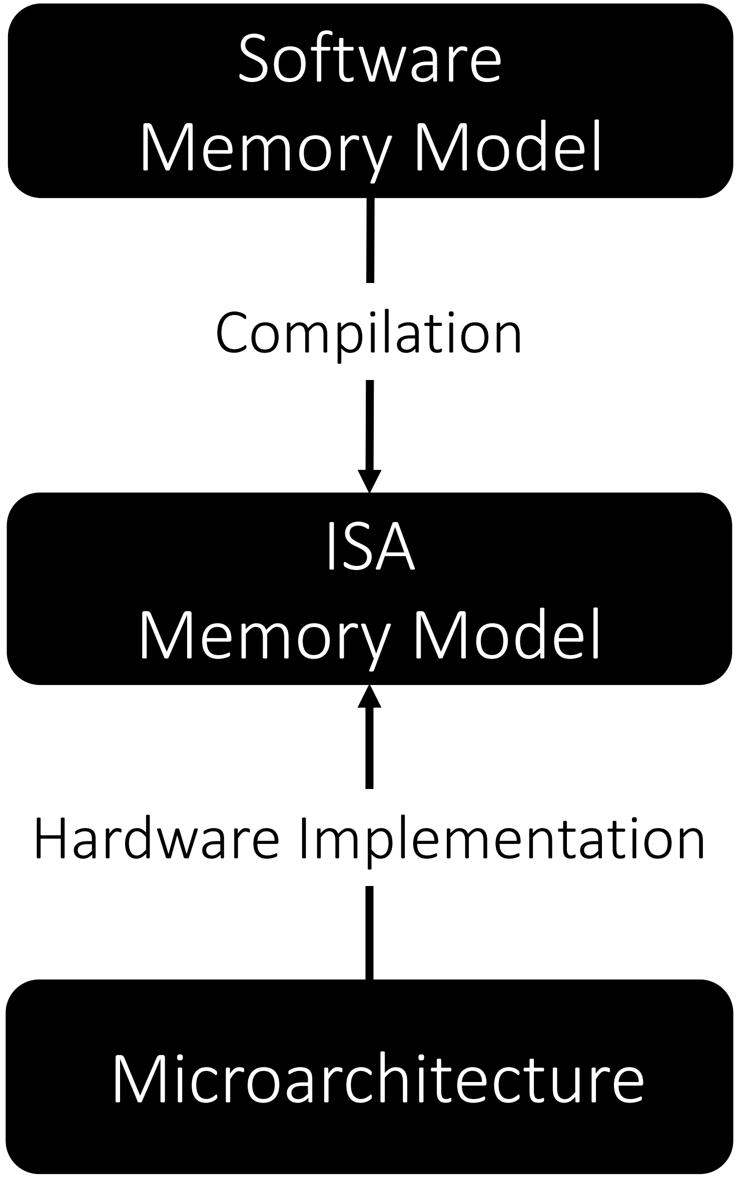
Forbidden by C11: r1=2, r2=1

T0	T1
st(data,1,rlx)	st(data,2,rlx)
r1=ld(ptr,rlx)	
r2=ld(data,rlx)	

C11/C++11	ARMv7
st(rlx)	STR
ld(rlx)	LDR
ld(acq)	LDR; DMB
...	...

ARM Cortex-A9

ARM Read-Read Hazard



Initial conditions: data=0, *ptr=&data

Forbidden by C11: r1=2, r2=1

T0	T1
st(data,1,rlx)	st(data,2,rlx)
r1=ld(ptr,rlx)	
r2=ld(data,rlx)	

Loading data through a pointer

C11/C++11	ARMv7
st(rlx)	STR
ld(rlx)	LDR
ld(acq)	LDR; DMB
...	...

Naïve compilation from C11 to ARMv7

C0	C1
ST [data]←1	ST [data]←2
LD [ptr]→r0	
LD [r0]→r1	
LD [data]→r2	

ARM Cortex-A9

ARM Read-Read Hazard

Initial conditions: data=0, *ptr=&data

Forbidden by C11: r1=2, r2=1

T0	T1
st(data,1,rlx)	st(data,2,rlx)
r1=ld(ptr,rlx)	
r2=ld(data,rlx)	

Setting flag1=1 causes flag2=1

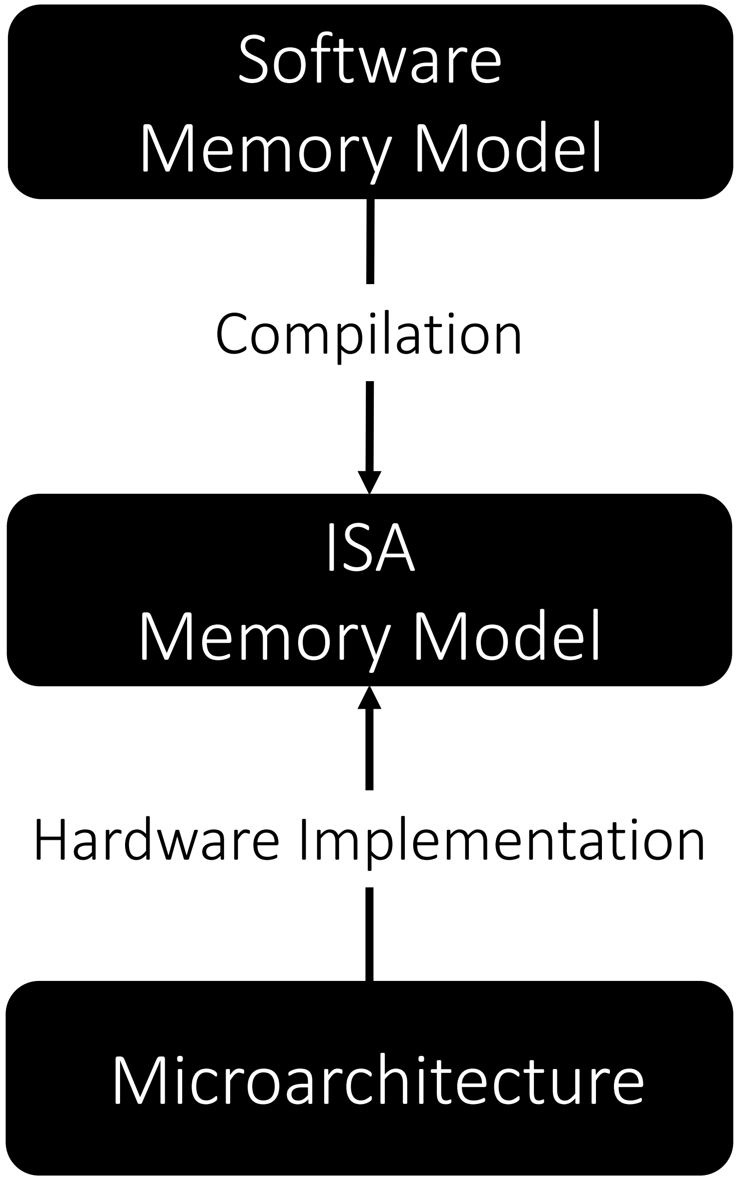
C11/C++11	ARMv7
st(rlx)	STR
ld(rlx)	LDR
ld(acq)	LDR; DMB
…	…

C0	C1
ST [data]←1	ST [data]←2
LD [ptr]→r0	
LD [r0]→r1	
LD [data]→r2	

Two loads of the same address

ARM Cortex-A9

Forbidden outcome observable on Cortex-A9



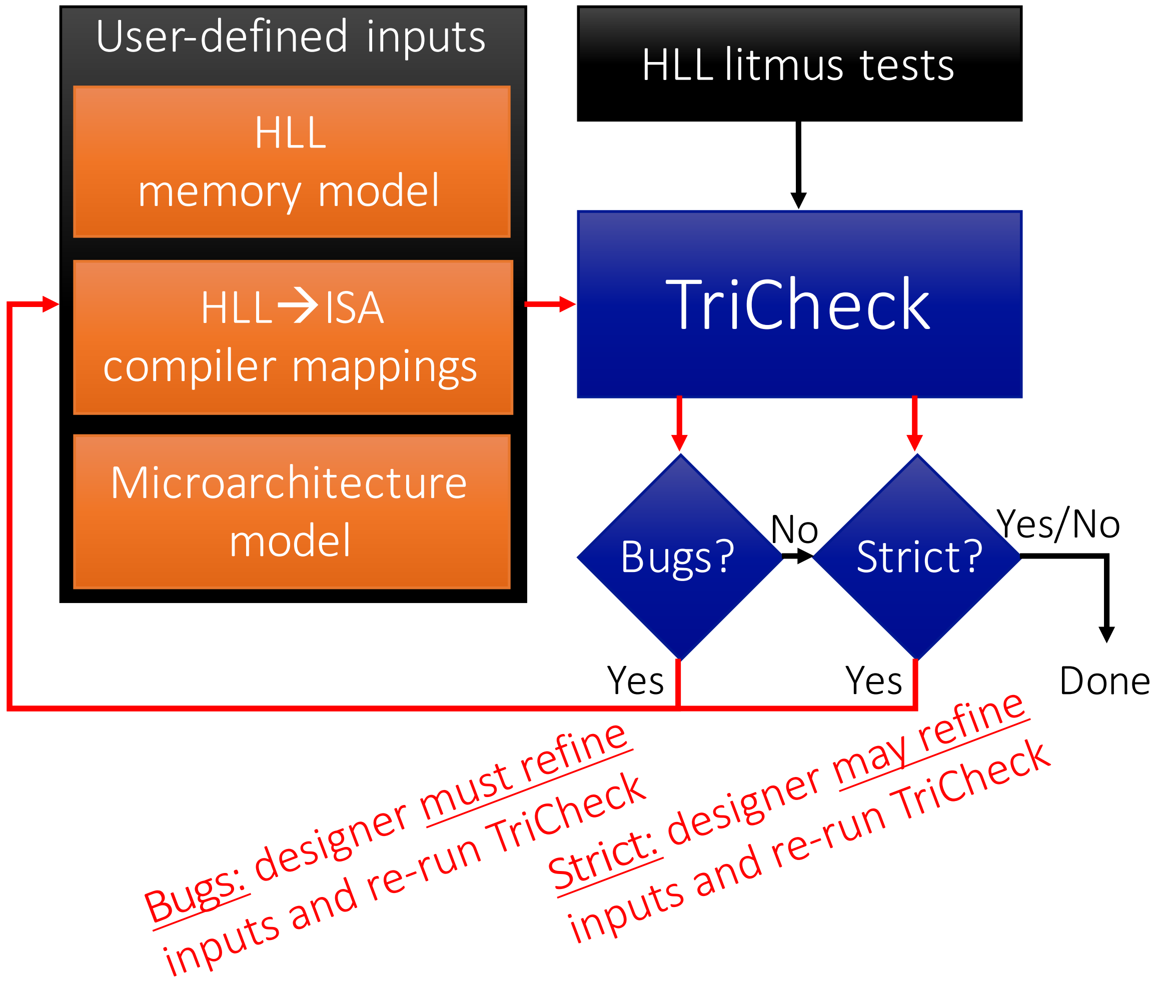
Outline

- Memory Consistency Model Verification
- Full-Stack Verification: Motivating Example
- TriCheck Framework for Full-Stack Memory Model Verification
- Bugs Found with TriCheck: RISC-V Case Study and Compiler Mappings
- Ongoing Work & Conclusions

TriCheck Key Ideas

- **First tool capable of full stack memory model verification**
  - Any layer can introduce real bugs
- **Litmus Tests + Auto-generators**
  - Comprehensive families of tests across HLL ordering options, compiler mapping variations, ISA options
- **Happens-before, graph-based analysis**
  - Nodes are memory accesses & ordering primitives
  - Edges are event orders discerned via memory model relations
- **Efficient top-to-bottom analysis: Runtime in seconds or minutes**
  - Fast enough to find real bugs; Interactive design process

TriCheck Methodology



User-defined inputs
HLL memory model
HLL→ISA compiler mappings
Microarchitecture model
HLL litmus tests
TriCheck
Bugs?
No
Strict?
Yes/No
Yes
Yes
Done
Bugs: designer must refine inputs and re-run TriCheck
Strict: designer may refine inputs and re-run TriCheck

- User-defined TriCheck inputs
  - HLL memory model (*Herd [Alglave et al. TOPLAS14]*)
  - HLL→ISA compiler mappings
  - Hardware model (*μspec* DSL)
- Auto-generated TriCheck inputs
  - HLL litmus test suite from templates
- Each iteration: bugs analyzed to identify cause
  - Compiler bug, hardware implementation bug, ISA bug
  - Blame may be debated
  - Blame != Fix

Outline

- Memory Consistency Model Verification
- Full-Stack Verification: Motivating Example
- TriCheck Framework for Full-Stack Memory Model Verification
- Bugs Found with TriCheck: RISC-V Case Study and Compiler Mappings
- Ongoing Work & Conclusions

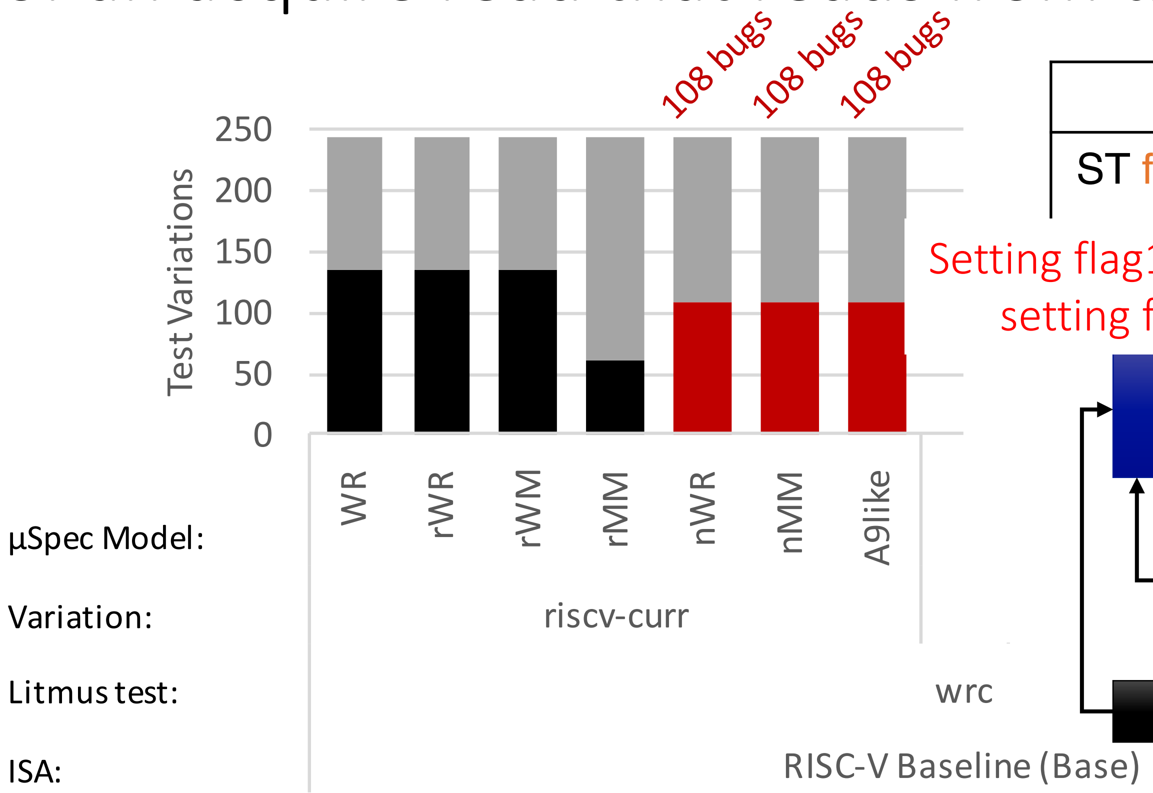
RISC-V Case Study

- Create μspec models for 7 distinct RISC-V implementation possibilities:
  - All abide by current RISC-V spec
  - Vary in preserved program order and store atomicity
- Started with stricter-than-spec microarchitecture: RISC-V Rocket Chip
  - TriCheck detects **bugs**: refine for correctness
  - TriCheck detects **over-strictness**: Performed legal (per RISC-V spec) microarchitectural relaxations
- Impossible to compile C11 for RISC-V as specified.
- Out of 1,701 tested C11 programs:
  - RISC-V-Base-compliant design allows 144 buggy outcomes
  - RISC-V-Base+A-compliant design allows 221 buggy outcomes

RISC-V Base: Lack of Cumulative Fences

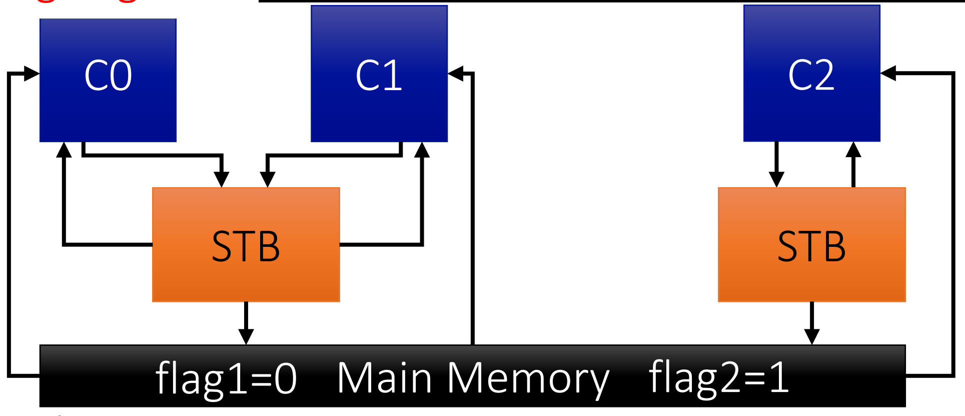
Initial conditions: x=0, y=0		
T0	T1	T2
a: sw x1, (x5)	b: lw x2, (x5)	e: lw x3, (x6)
	c: fence rw, w	f: fence r, rw
	d: sw x2, (x6)	g: lw x4, (x5)
Forbidden HLL Outcome: x1=1, x2=1, x3=1, x4=0		

C11 acquire/release synchronization is transitive: accesses before a release write in program order, and observed by the releasing core prior to the release write must be ordered before the release from the viewpoint of an acquire read that reads from the release write



C0	C1	C2
ST flag1←1	if (LD flag1==1)	if (LD flag2==1)
	FENCE[LD.ST,ST]	FENCE[LD,LD.ST]
	ST flag2←1	LD flag1→test

Setting flag1 causes setting flag2



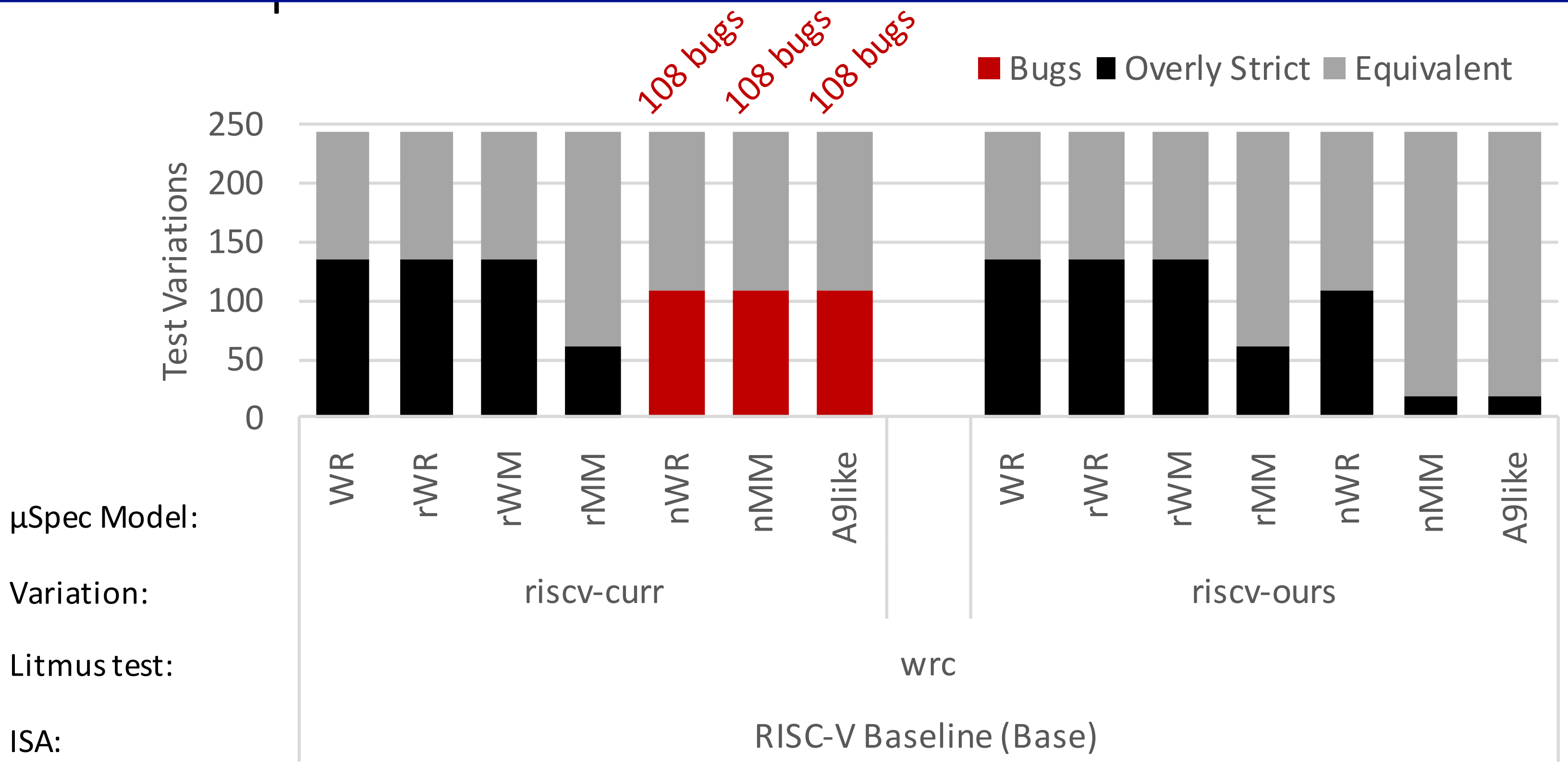
RISC-V Base: Lack of Cumulative Fences

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)
Forbidden HLL Outcome: x1=1, x2=1, x3=1, x4=0		

Base RISC-V ISA lacks cumulative fences

- Cumulative fence needed to enforce order between different-thread accesses
- Cannot fix bugs by modifying compiler

Our solution: add cumulative fences to the Base RISC-V ISA



More results in the paper:

- Both Base and Base+A:
  - Lack of cumulative lightweight fences
  - Lack of cumulative heavyweight fences
  - Re-ordering of same-address loads
  - No dependency ordering, but Linux port assumes it
- Base+A only:
  - Lack of cumulative releases; no acquire-release synchronization
  - No roach-motel movement

Takeaway: Current RISC-V cannot serve as a compiler target for C11

Next Steps: We are members of RISC-V memory model working group, working to formalize a memory model for RISC-V that meets the needs of RISC-V users and supports C11.

Evaluating Compiler Mappings with TriCheck

- During RISC-V analysis, we discovered two counter-examples while using the “proven-correct” *trailing-sync* mappings for compiling C11 to POWER/ARMv7
- Also incorrect: the *proof* for the C11 to POWER/ARMv7 trailing-sync compiler mappings [Manerkar et al., CoRR ‘16]

Conclusions

- Memory model design choices are complicated =>
  - Verification calls for automated analysis to comprehensively tackle subtle interplay between many diverse features.
- TriCheck uncovered flaws in the RISC-V memory model…
  - But more generally, TriCheck can be used on any ISA.
- Languages and Compilers matter too…
  - TriCheck uncovered bugs in the trailing-sync compiler mapping from C11 to POWER/ARMv7

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