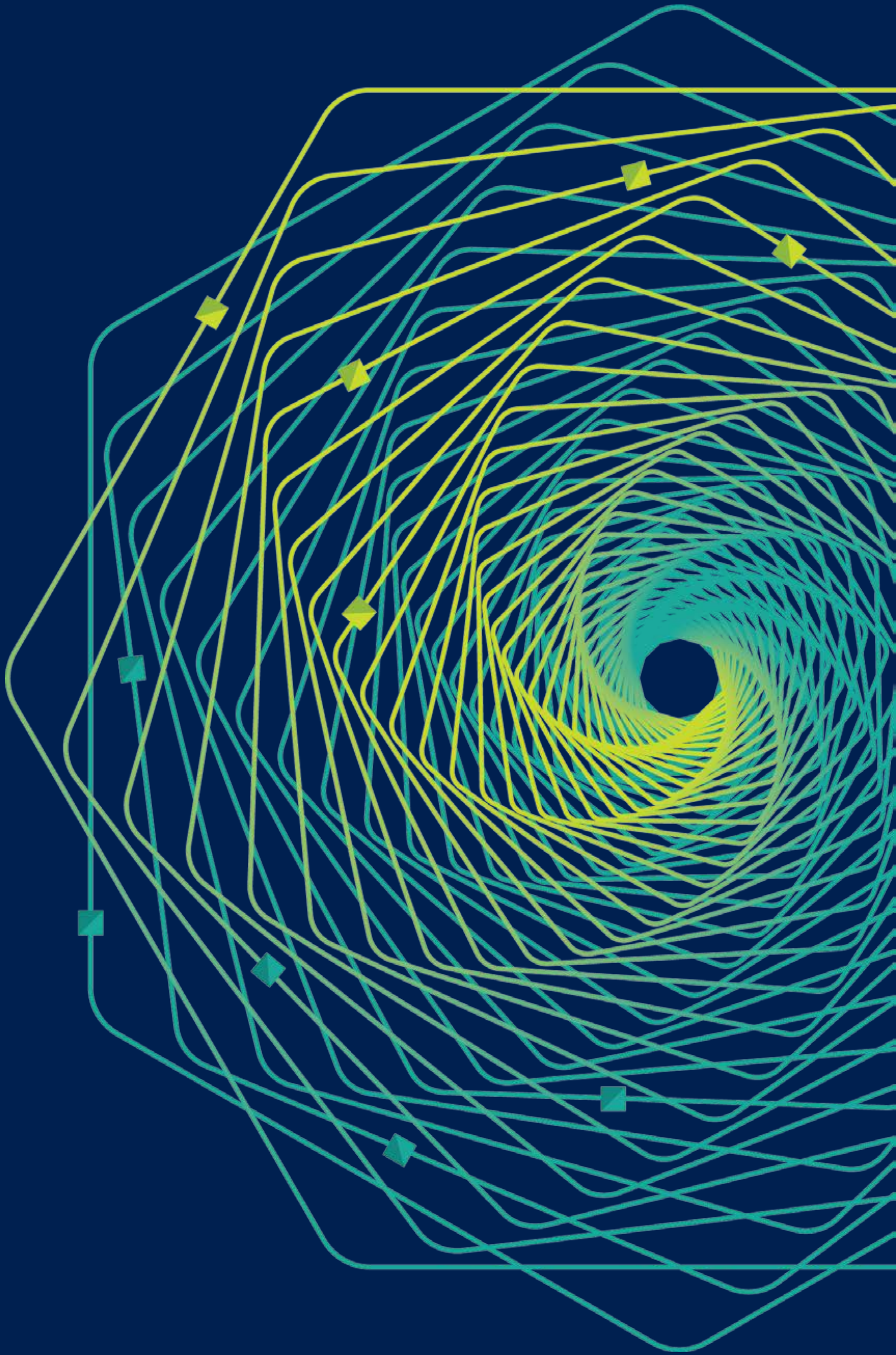
Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions

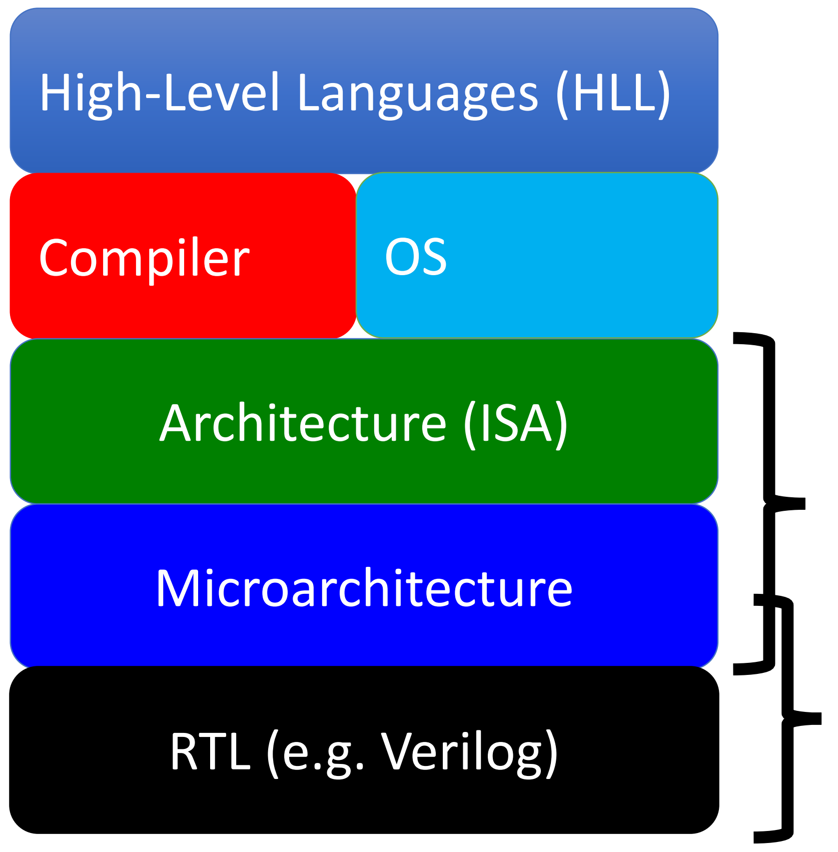


Hardware-Aware Security Verification and Synthesis

Margaret Martonosi
H. T. Adams ‘35 Professor
Dept. of Computer Science
Princeton University

Joint work with Caroline Trippel, Princeton CS PhD student and Dr. Daniel Lustig, NVIDIA

The Check Suite: An Ecosystem of Tools For Verifying Memory Consistency Model Implementations



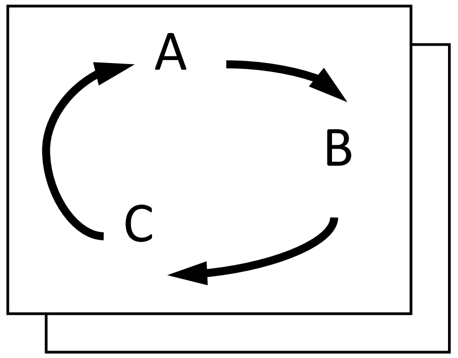
TriCheck [ASPLOS '17] [IEEE MICRO Top Picks]

COATCheck [ASPLOS '16] [IEEE MICRO Top Picks]

PipeCheck [Micro-47] [IEEE MICRO Top Picks]

CCICheck [Micro-48] [Nominated for Best Paper Award]

RTLCheck [Micro-50] [IEEE MICRO Top Picks Honorable Mention]



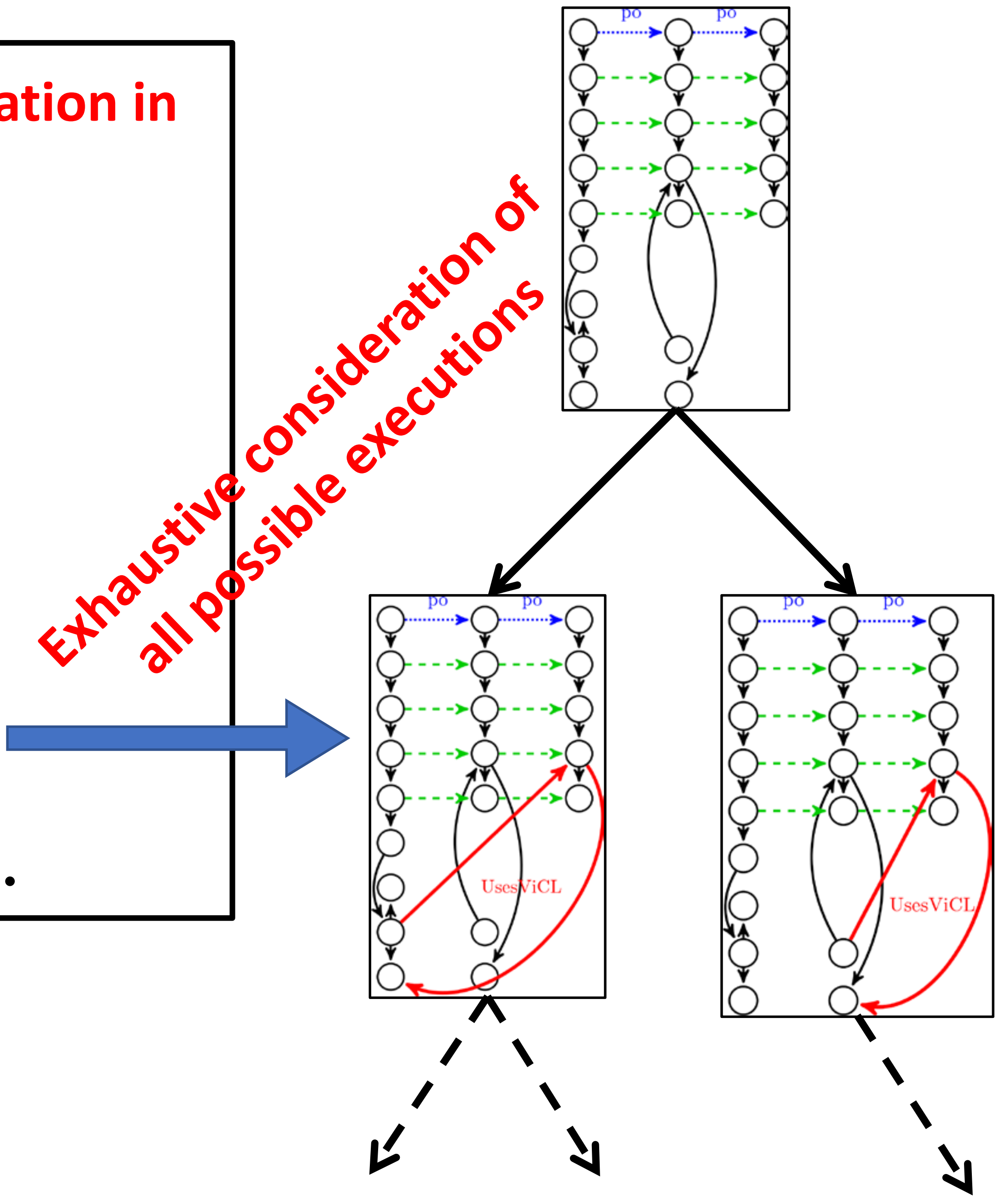
Our Approach

- Axiomatic specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
    - Cyclic => A->B->C->A… Can't happen
    - Acyclic => Scenario is observable

Check: Formal, Axiomatic Models and Interfaces

Microarchitecture Specification in *μSpec* DSL

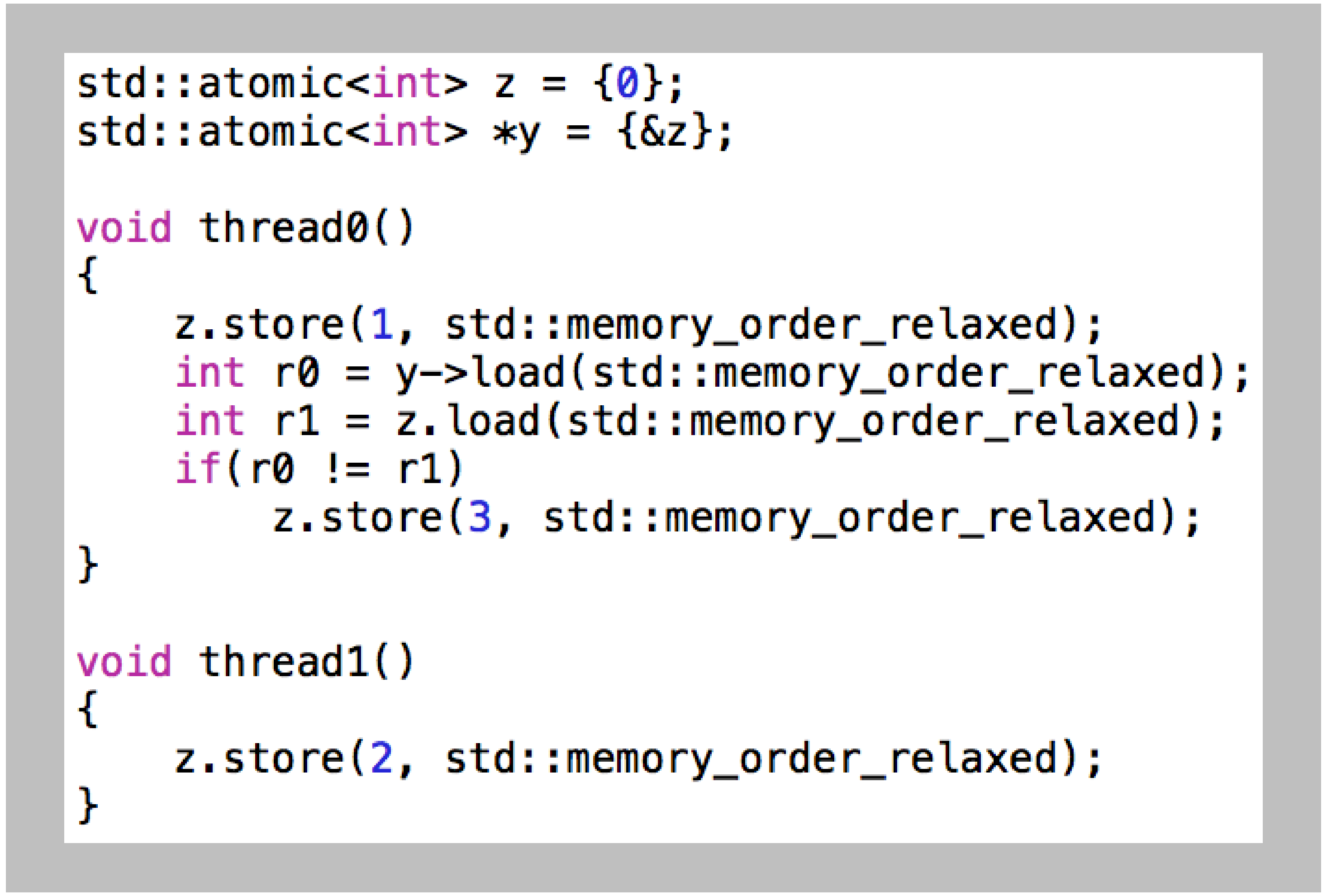
```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

Axiom "Execute_stage_is_in_order":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\
 EdgeExists ((i1, Fetch), (i2, Fetch)) =>
   AddEdge ((i1, Execute), (i2, Execute), "PPO").
```



Microarchitectural happens-before (μhb) graphs

Example: ARM Read-Read Hazard

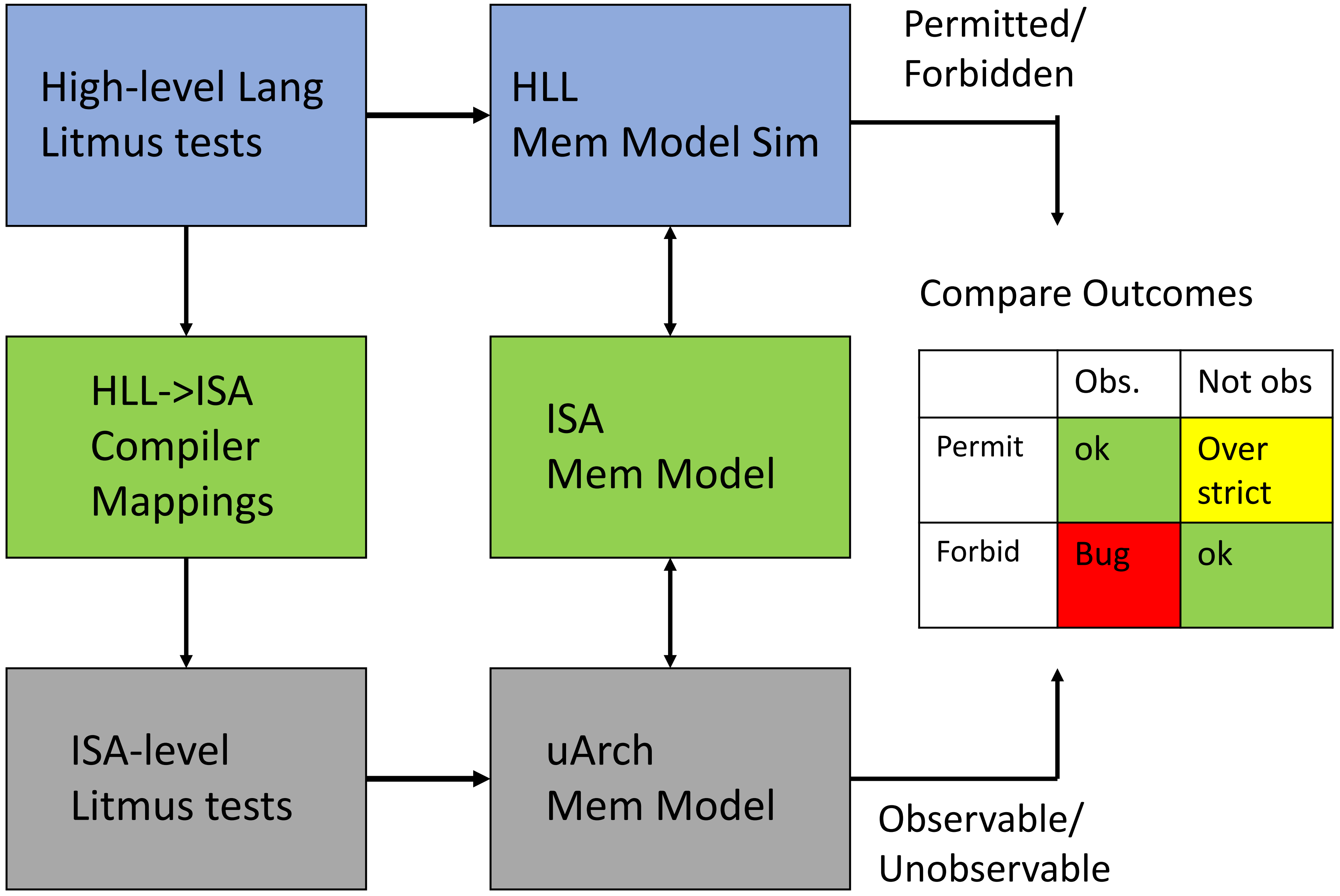
- ARM ISA spec ambiguous regarding same-address Ld→Ld ordering:
  - Compiler's job? Hardware job?
- C/C++ variables with atomic type require same-addr. Ld→Ld ordering
- ARM issued errata1:
  - Rewrite compilers to insert fences (ordering instructions) with performance penalties
- ARM ISA had the right ordering instructions – just needed to use them.



```
std::atomic<int> z = {0};
std::atomic<int> *y = {&z};

void thread0()
{
    z.store(1, std::memory_order_relaxed);
    int r0 = y->load(std::memory_order_relaxed);
    int r1 = z.load(std::memory_order_relaxed);
    if(r0 != r1)
        z.store(3, std::memory_order_relaxed);
}

void thread1()
{
    z.store(2, std::memory_order_relaxed);
}
```

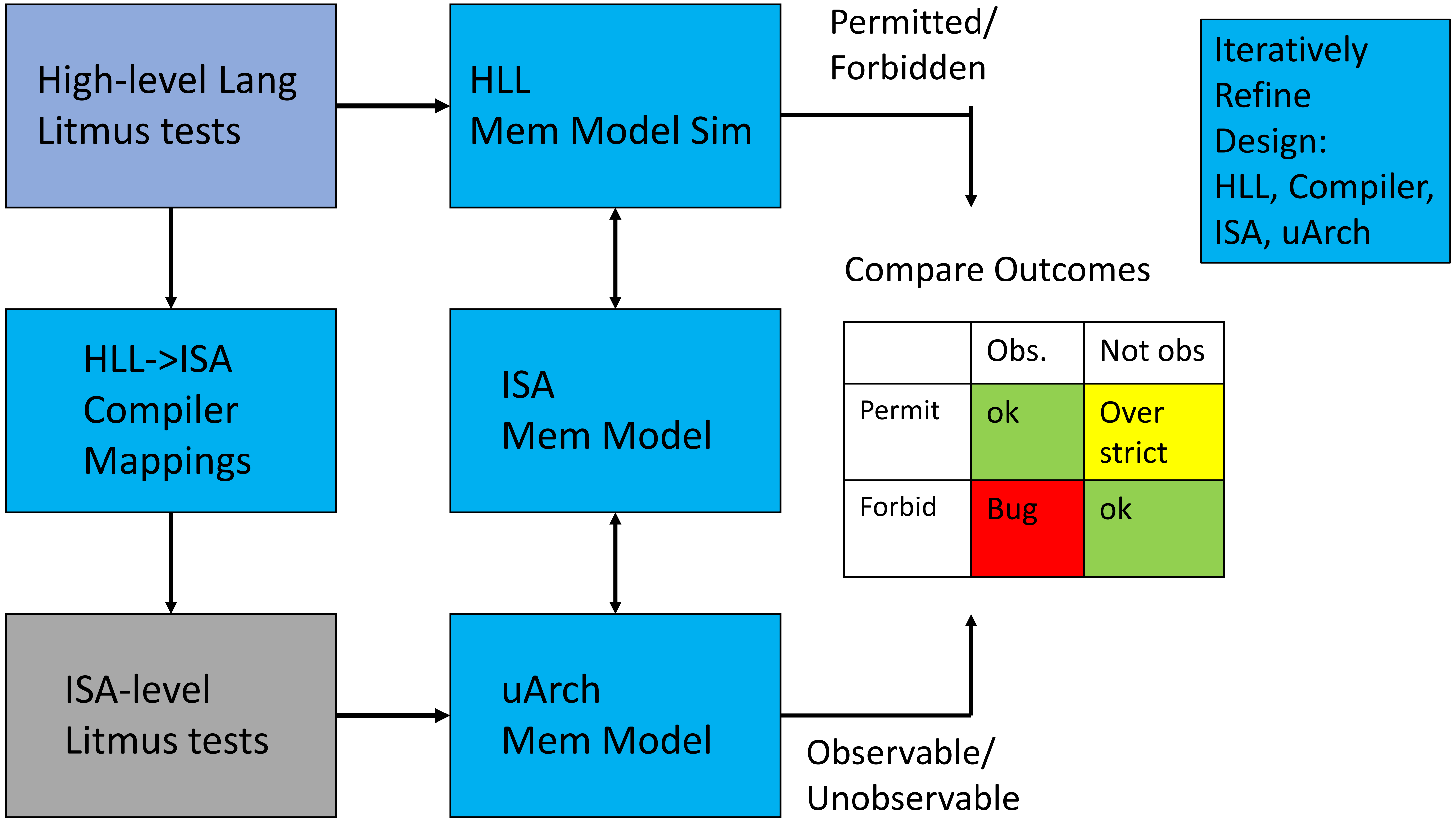
Original: Alglave 2011
Google Nexus 6: http://check.cs.princeton.edu/tutorial_extras/SnapVideo.mov

TriCheck Framework: Verifying Memory Event Ordering from Languages to Hardware



	Obs.	Not obs
Permit	ok	Over strict
Forbid	Bug	ok

TriCheck Framework: Verifying Memory Event Ordering from Languages to Hardware

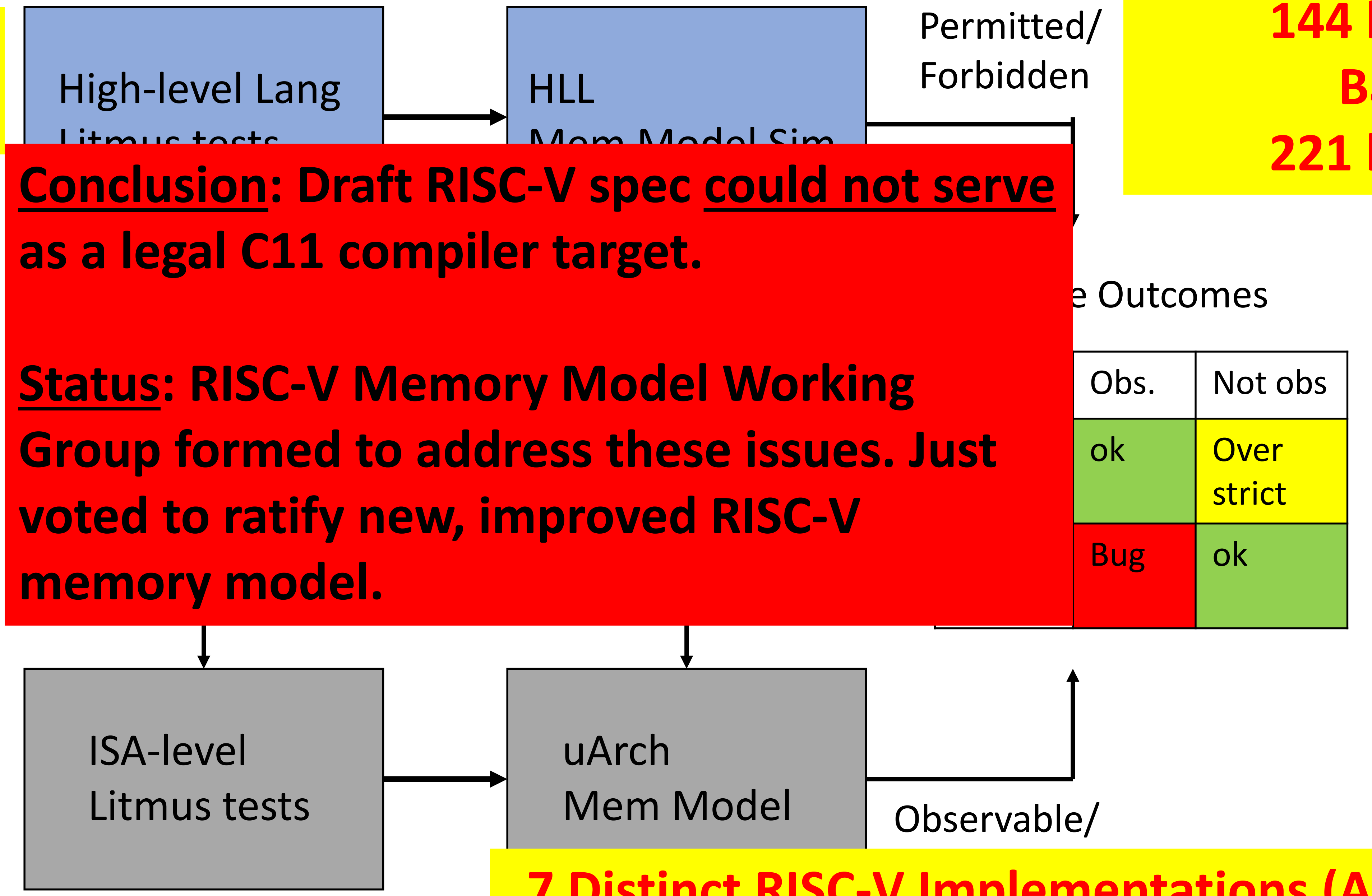


	Obs.	Not obs
Permit	ok	Over strict
Forbid	Bug	ok

TriCheck Framework: RISC-V Case Study

1701 C11 Programs

Base RISC-V ISA: 144 buggy outcomes
Base+Atomics: 221 buggy outcomes



	Obs.	Not obs
	ok	Over strict
	Bug	ok

<u>Conclusion</u>: Draft RISC-V spec <u>could not serve</u> as a legal C11 compiler target.

<u>Status</u>: RISC-V Memory Model Working Group formed to address these issues. Just voted to ratify new, improved RISC-V memory model.

7 Distinct RISC-V Implementations (All abide by RISC-V specifications, but vary in reordering / performance

CheckMate: From Memory Consistency Models to Security

January 2018: Spectre & Meltdown

Well-known cache side-channel attack { Flush+Reload

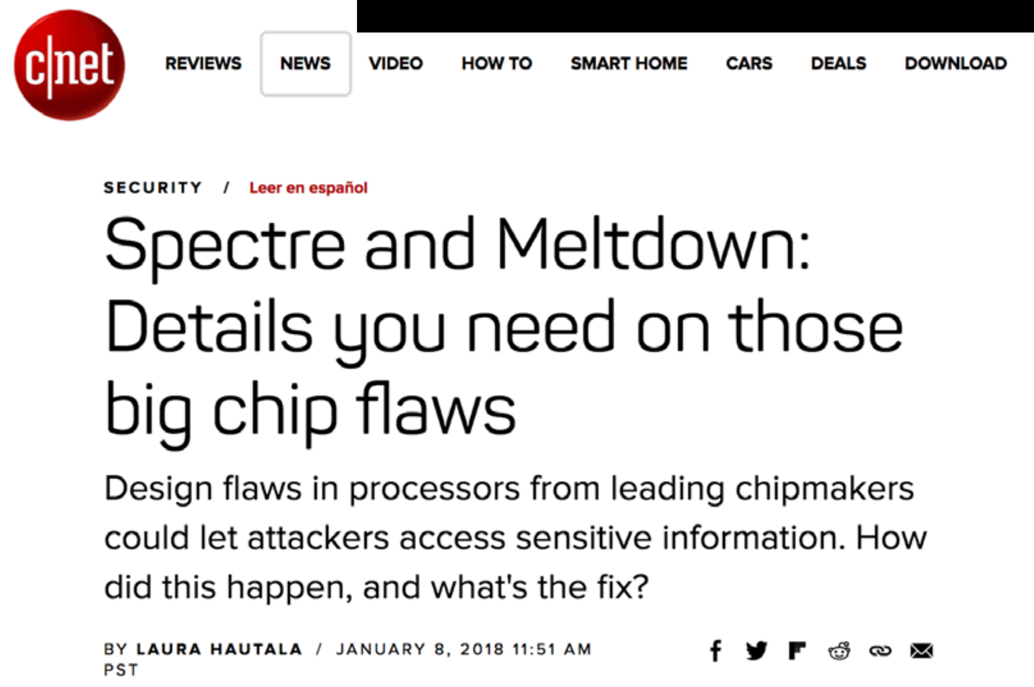
+

Widely-used hardware feature { Speculation

=

New exploit

2 new attacks {

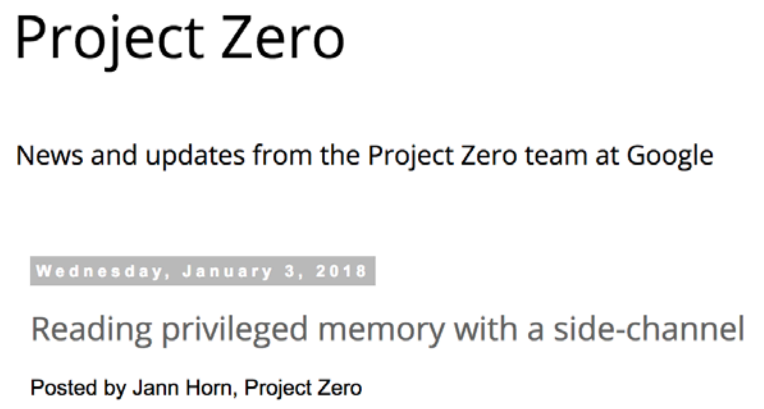

REVIEWS NEWS VIDEO HOW TO SMART HOME CARS DEALS DOWNLOAD

SECURITY / Leer en español

Spectre and Meltdown: Details you need on those big chip flaws

Design flaws in processors from leading chipmakers could let attackers access sensitive information. How did this happen, and what's the fix?

BY LAURA HAUTALA / JANUARY 8, 2018 11:51 AM PST

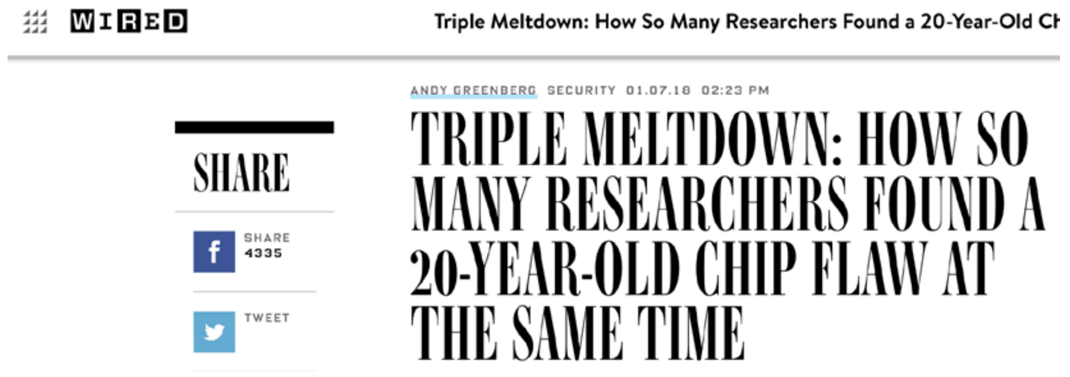

Project Zero

News and updates from the Project Zero team at Google

Wednesday, January 3, 2018

Reading privileged memory with a side-channel

Posted by Jann Horn, Project Zero


WIRED Triple Meltdown: How So Many Researchers Found a 20-Year-Old Ch

ANDY GREENBERG SECURITY 01.07.18 02:23 PM

SHARE

SHARE 4335

TWEET

TRIPLE MELTDOWN: HOW SO MANY RESEARCHERS FOUND A 20-YEAR-OLD CHIP FLAW AT THE SAME TIME

Attack Discovery & Synthesis: What We Would Like

1. Specify system to study	Formal interface and specification of given system implementation
2. Specify attack pattern	E.g. Subtle event sequences during program's execution
3. Synthesis	Either output synthesized attacks. Or determine that none are possible

Attack Discovery & Synthesis: CheckMate TL;DR

1. Specify system to study

2. Specify attack pattern

3. Synthesis

- **What we did**: Developed a tool to do this, based on the uHB graphs from previous sections.
- **Results**: Automatically synthesized Spectre and Meltdown, as well as two new distinct exploits and many variants.

[Trippel, Lustig, Martonosi. https://arxiv.org/abs/1802.03802]
[Trippel, Lustig, Martonosi. MICRO-51. October, 2018]

In more detail…

CheckMate Methodology

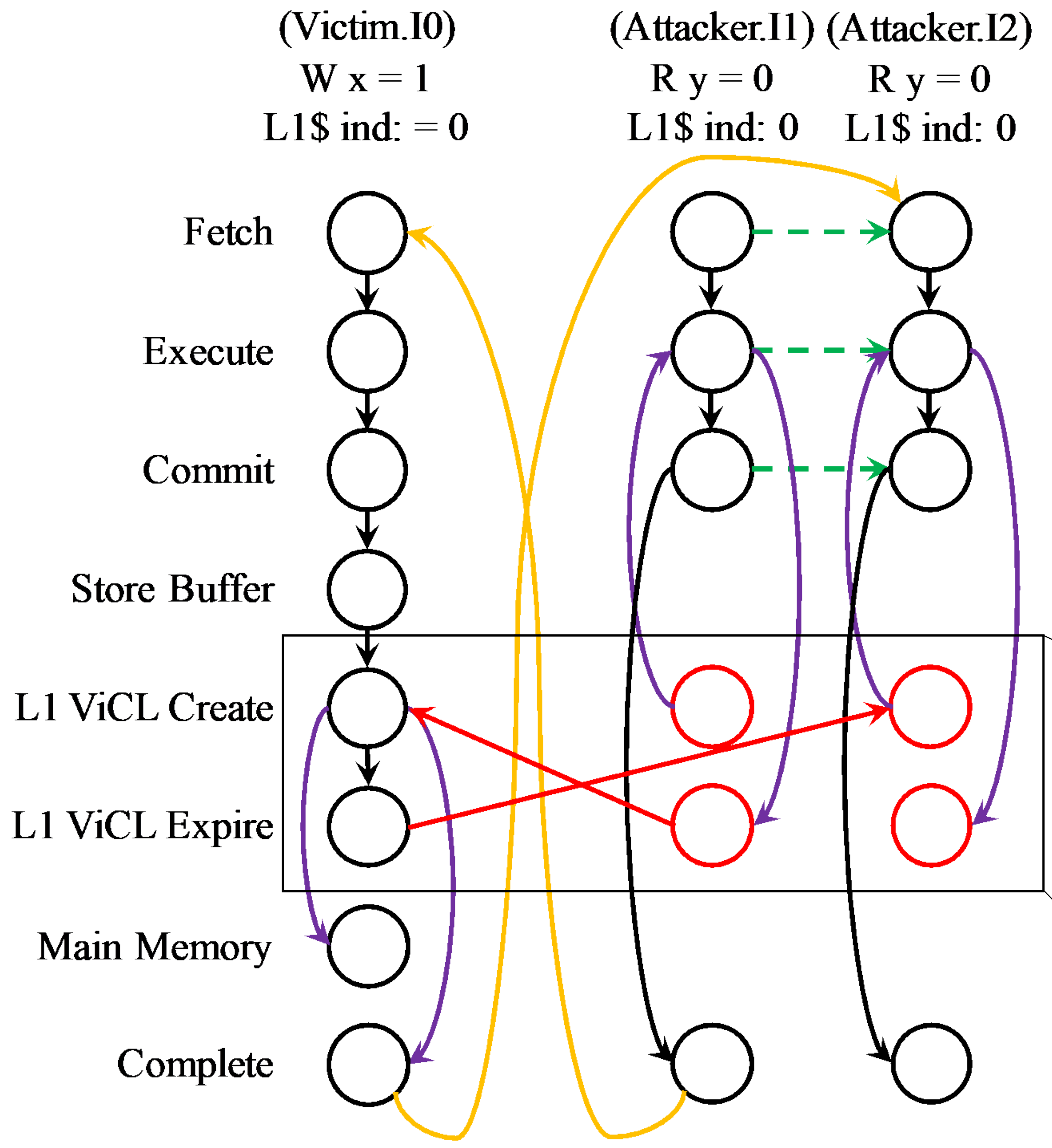
1. Frame classes of attacks as patterns of event interleavings?

   -> Essentially a snippet out of a happens-before graph

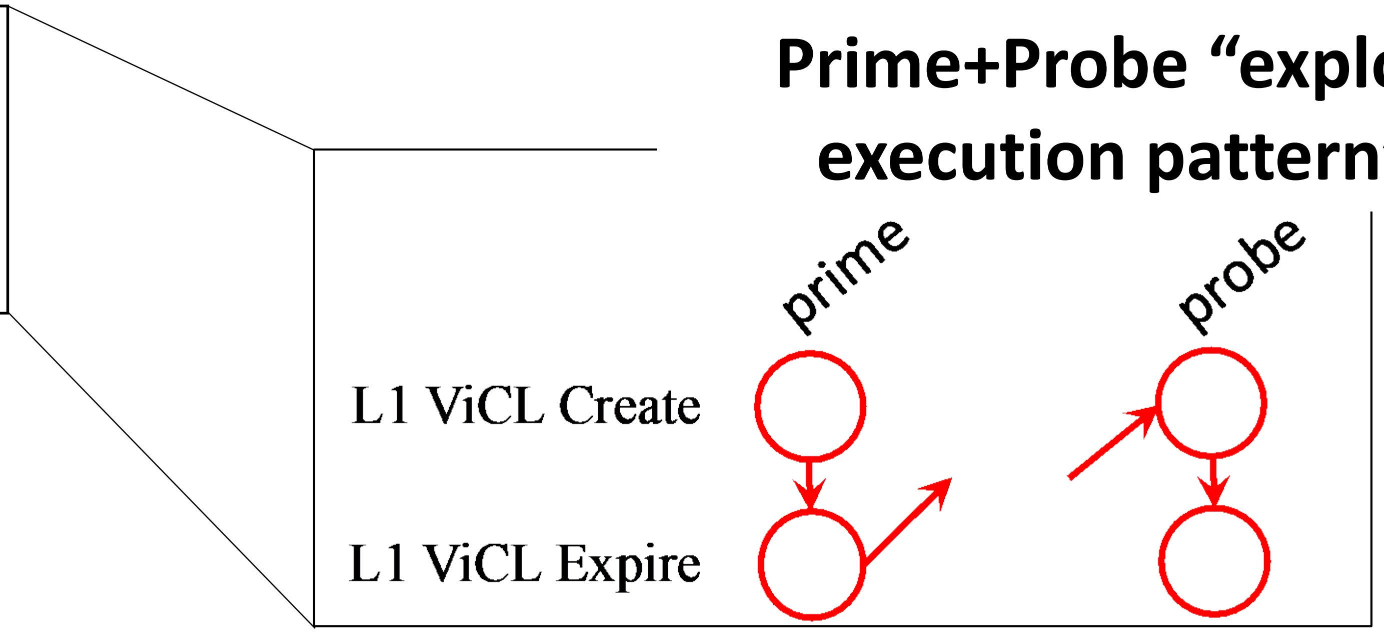
2. Specify hardware using uSpec axioms

   -> Determine if attack is realizable on a given hardware implementation

Exploit Programs: µhb Graphs featuring Exploit Patterns

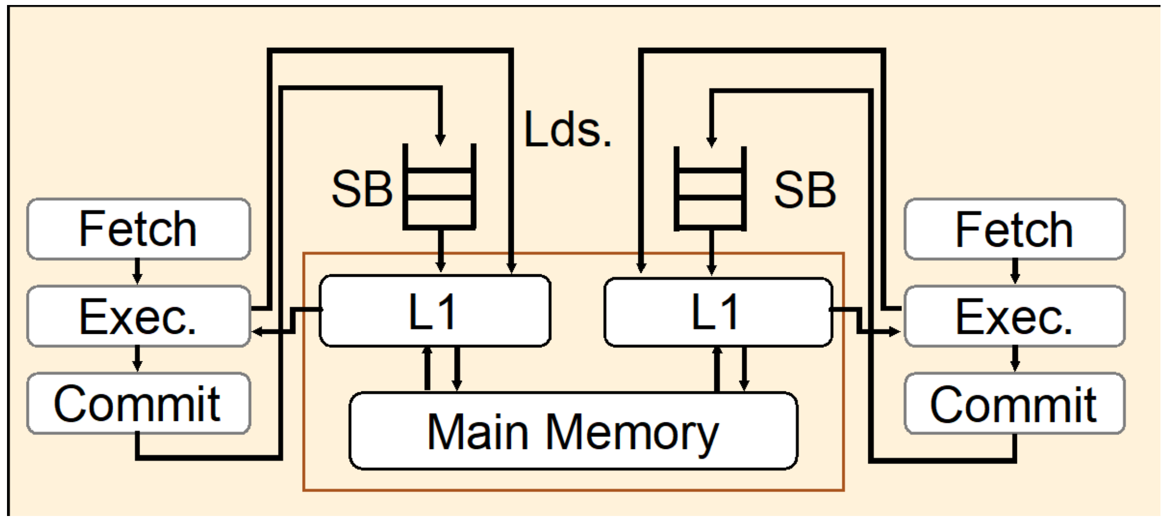


1. Model subtle hardware-specific event orderings/inter-leavings: **<u>µhb graphs</u>**
2. Determine if an exploit is possible for a given implementation: **<u>cycle checks</u>**



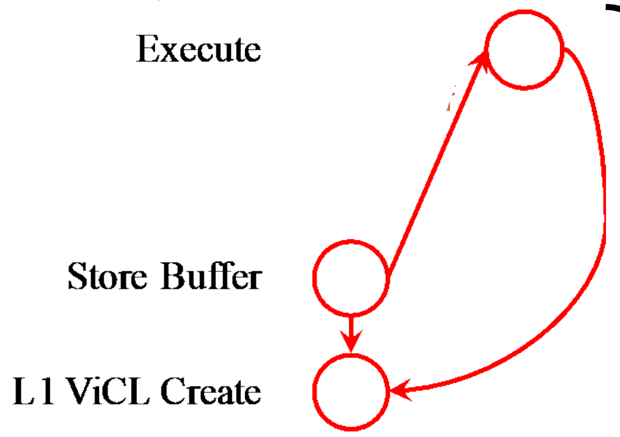
Microarchitecture-Aware Program Synthesis

Microarchitecture



+

µhb Pattern



Load being sourced from the store buffer

+

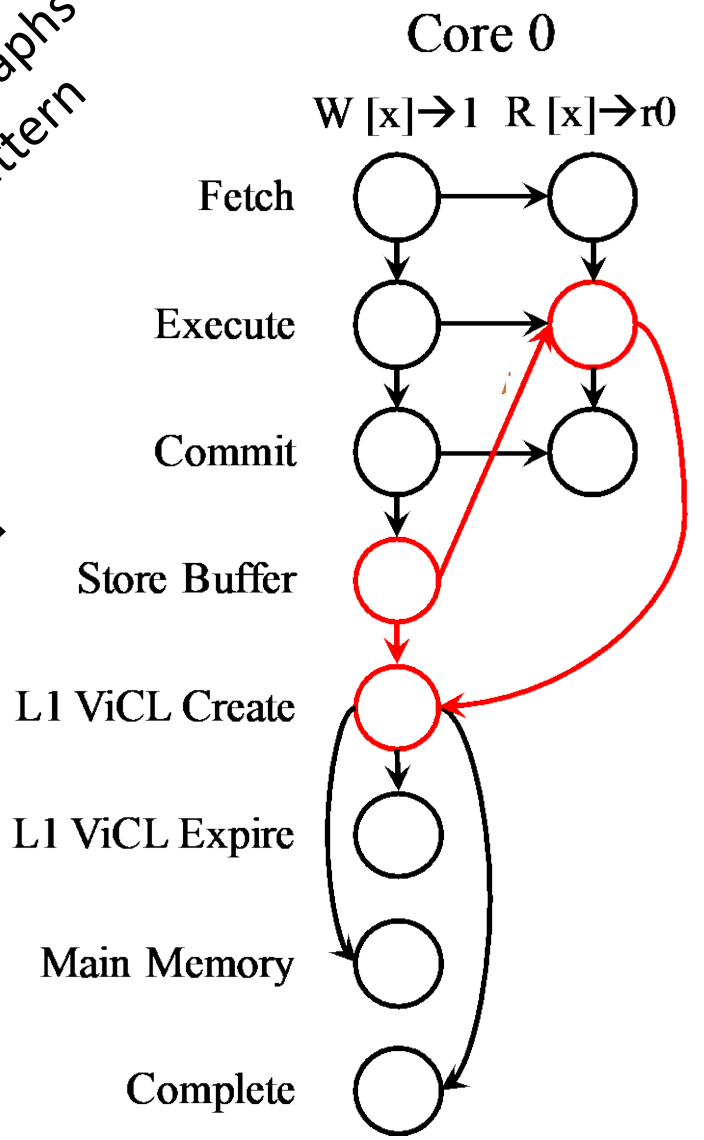
Execution Constraints

#cores = 1
#threads = 1
#instr ≤ 2

Check Mate

Enumerate all possible execution graphs feat. pattern

µhb Graph



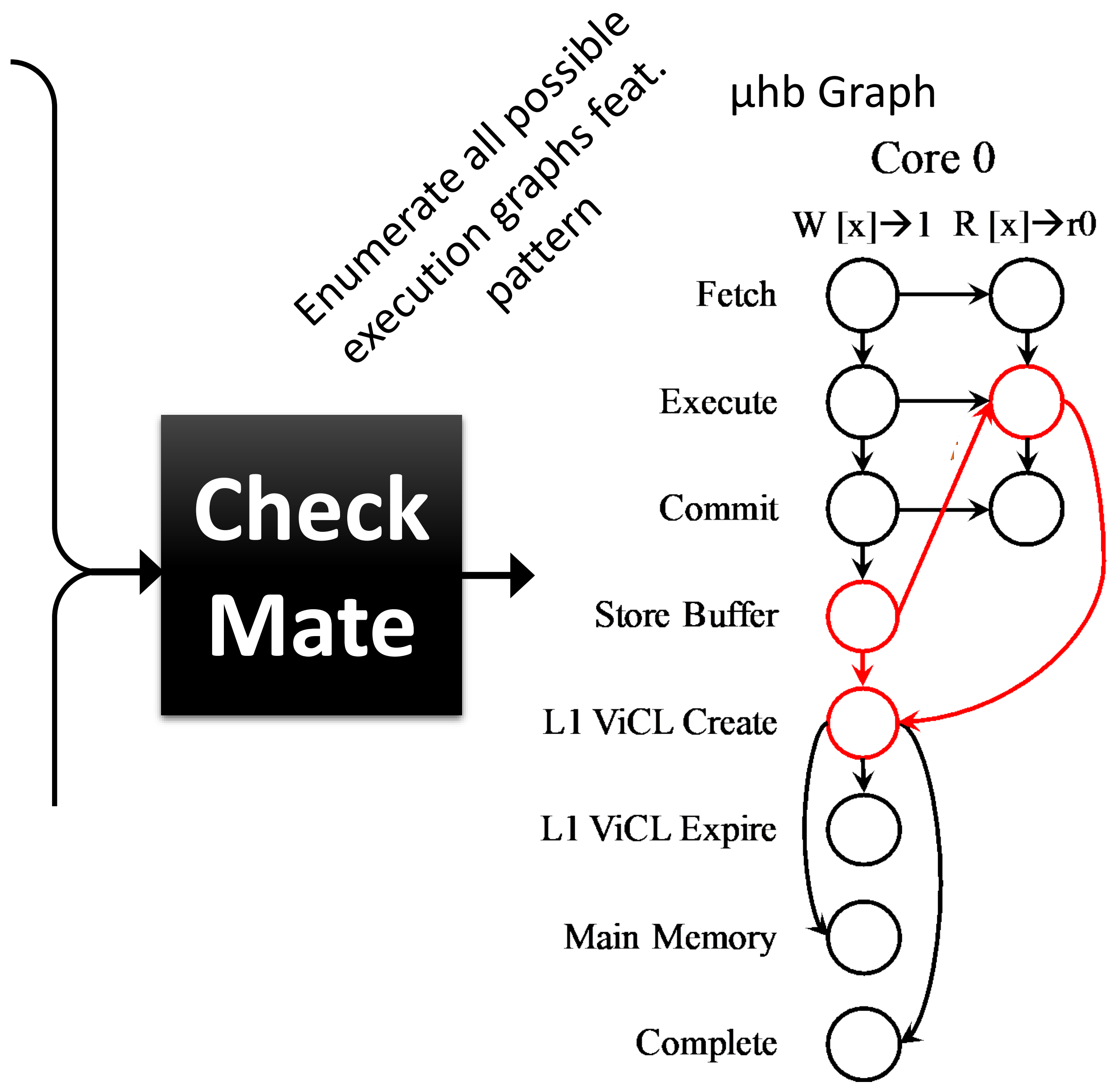
Microarchitecture-Aware Program Synthesis

Microarchitecture Specification

```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

Axiom "Execute_stage_is_in_order":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\
 EdgeExists ((i1, Fetch), (i2, Fetch)) =>
   AddEdge ((i1, Execute), (i2, Execute), "PPO").
```

Prior work addresses the problem of proving this correct with respect to RTL

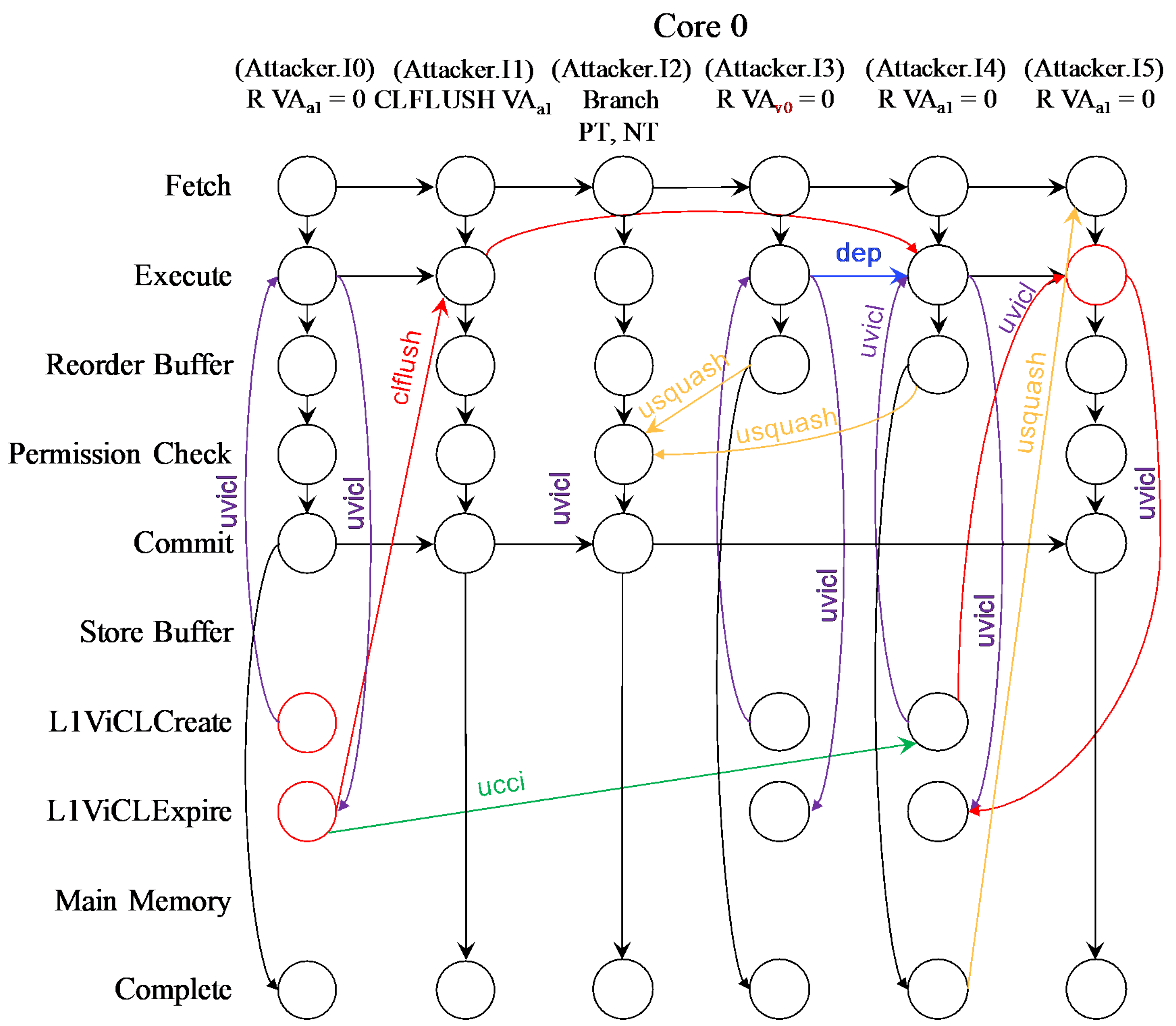
- SW/OS/HW events and locations
- SW/OS/HW ordering details
- Hardware optimizations, e.g., speculation
- Processes and resource-sharing
- Memory hierarchies and cache coherence protocols



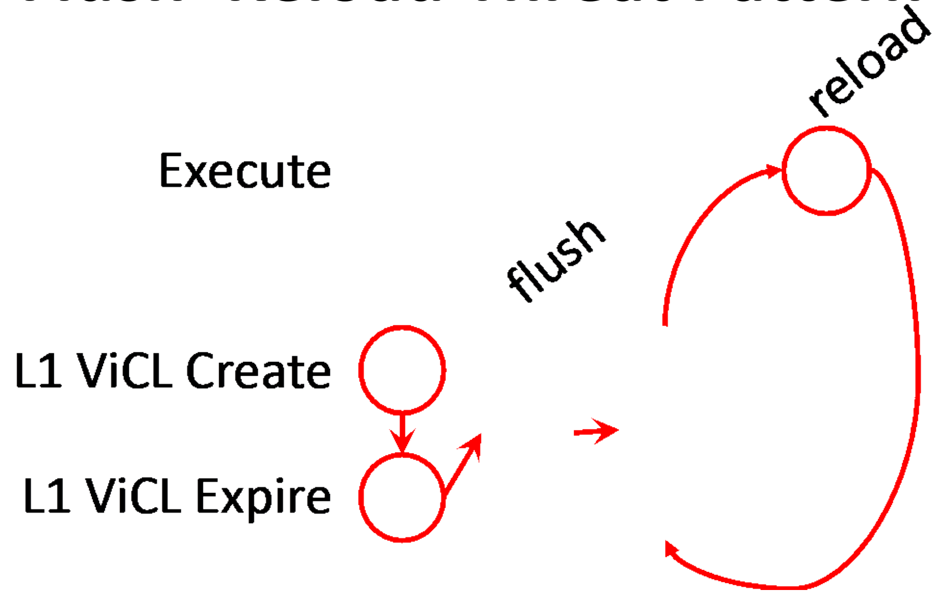
Relational Model Finding (RMF): A Natural Fit for Security Litmus Test Synthesis

- A relational model is a set of constraints on an abstract system (for CheckMate, **a μhb graph**) of:
  - Set of abstract objects (for CheckMate, **μhb graph nodes**)
  - Set of N-dimensional relations (for example., 2D **μhb graph edges** relation connecting 2 nodes)
- For CheckMate, the constraints are a **μhb pattern** of interest
- RMF attempts to find and satisfying "instance" (or μhb graph)
- Implementation: Alloy DSL maps RMF problems onto Kodkod model-finder, which in turn uses off-the-shelf SAT solvers
- CheckMate Tool maps μspec HW/OS spec to Alloy

Spectre (Exploits Speculation)


Core 0
(Attacker.I0) R VA_{a1} = 0
(Attacker.I1) CLFLUSH VA_{a1}
(Attacker.I2) Branch PT, NT
(Attacker.I3) R VA_{v0} = 0
(Attacker.I4) R VA_{a1} = 0
(Attacker.I5) R VA_{a1} = 0
Fetch
Execute
Reorder Buffer
Permission Check
Commit
Store Buffer
L1ViCLCreate
L1ViCLExpire
Main Memory
Complete
uvicl
clflush
usquash
dep
ucci

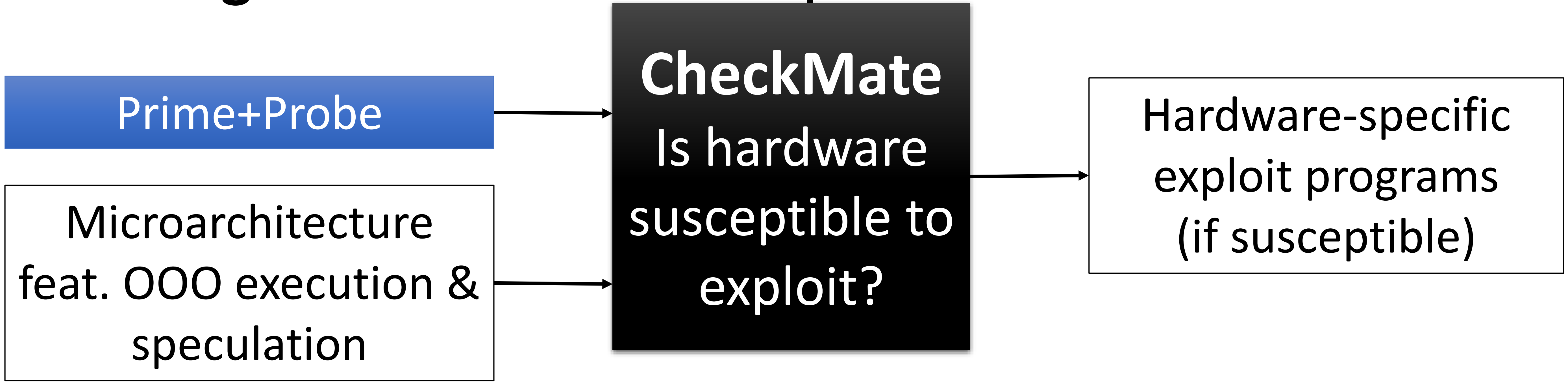
Flush+Reload Threat Pattern

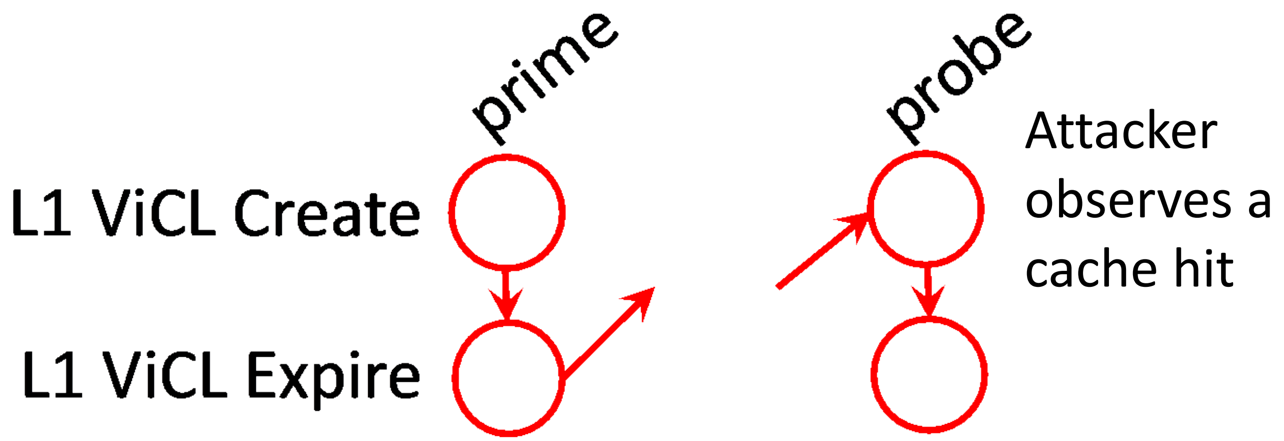


Spectre Security Litmus Test

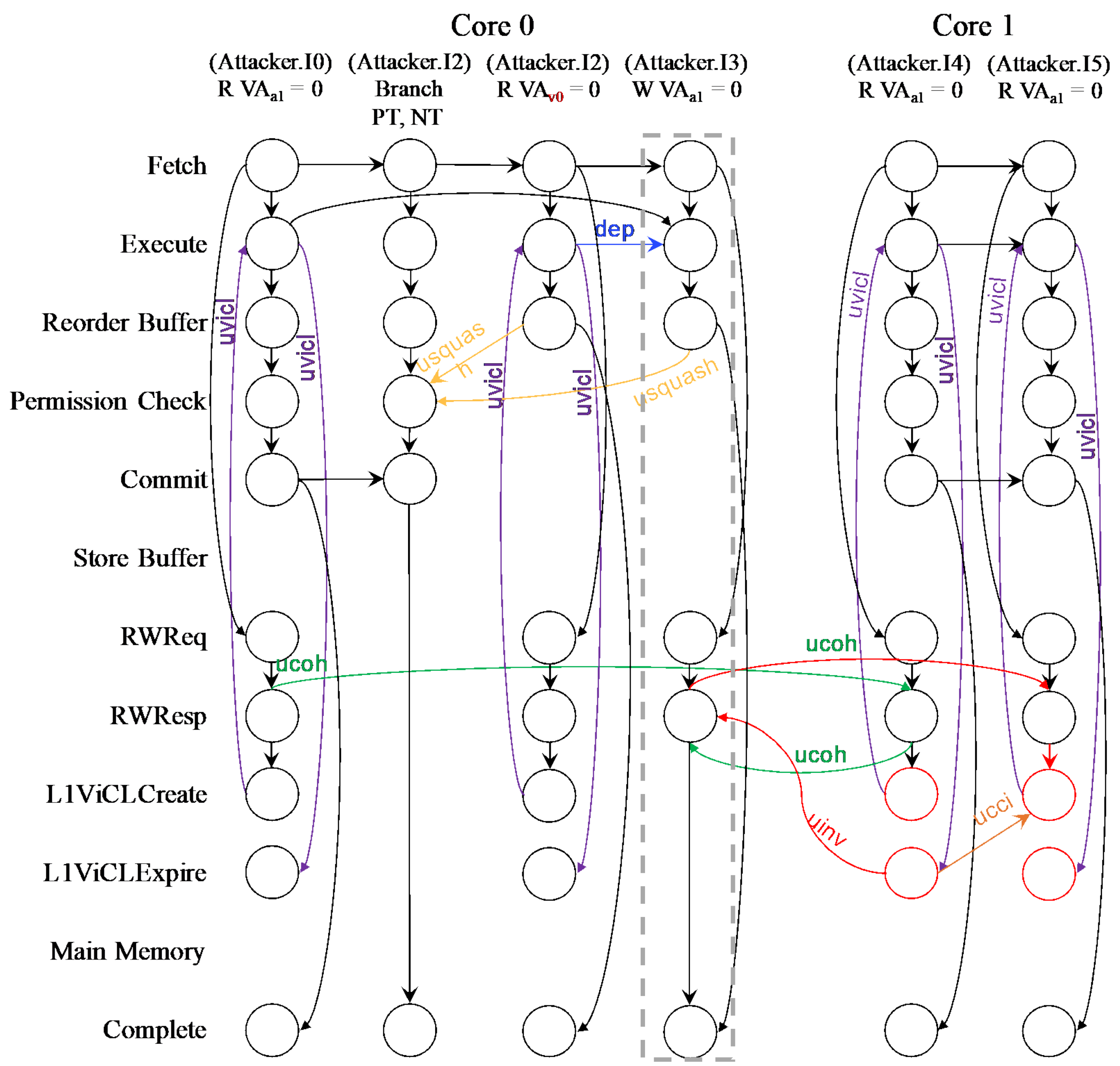
Initial conditions: [x]=0, [y]=0	
Attacker T0	
R [VA_{a1}]→0	
CLFLUSH [VA_{a1}]	← Flush
Branch → PT,NT	
R [VA_{v0}]→r1	
R [f(r1)=VA_{a1}]→0	
R [VA_{a1}]→0	← Reload

Prime&Probe Attack Pattern: Synthesizing MeltdownPrime & SpectrePrime


Prime+Probe
Microarchitecture feat. OOO execution & speculation
CheckMate
Is hardware susceptible to exploit?
Hardware-specific exploit programs (if susceptible)

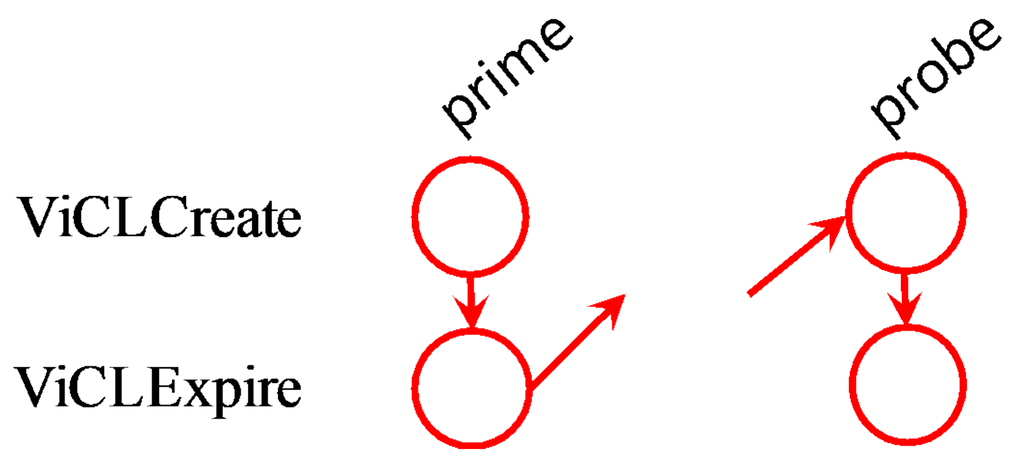


SpectrePrime uhb Graph



Core 0
Core 1
(Attacker.I0) R VA$_{a1}$ = 0
(Attacker.I2) Branch PT, NT
(Attacker.I2) R VA$_{v0}$ = 0
(Attacker.I3) W VA$_{a1}$ = 0
(Attacker.I4) R VA$_{a1}$ = 0
(Attacker.I5) R VA$_{a1}$ = 0
Fetch
Execute
Reorder Buffer
Permission Check
Commit
Store Buffer
RWReq
RWResp
L1ViCLCreate
L1ViCLExpire
Main Memory
Complete
dep
usquash
usquash
uvicl
ucoh
ucoh
ucoh
uinv
ucci

Prime+Probe Threat Pattern



Spectre Security Litmus Test

Initial conditions: [x]=0, [y]=0	
Attacker T0	Attacker T0
R [VA$_{a1}$]→0 Branch → PT,NT R [VA$_{v0}$] → r1 W [f(r1)=VA$_{a1}$] → 0	R [VA$_{a1}$]→0 ← Prime R [VA$_{a1}$]→0 ← Probe

Overall Results: What exploits get synthesized? And how long does it take?

Exploit Pattern	#Instrs (RMF Bound)	Output Attack	Minutes to synthesize 1st exploit	Minutes to synthesize all exploits	#Exploits Synthesized
Flush +Reload	4	Traditional Flush+Reload	6.7	9.7	70
	5	Meltdown	27.8	59.2	572
	6	Spectre	101.0	198.0	1144
Prime +Probe	3	Traditional Prime+Probe	5.4	6.7	12
	4	MeltdownPrime	17.0	8.2	24
	5	SpectrePrime	71.8	76.7	24

CheckMate: Takeaways

- New Variants reported: SpectrePrime and MeltdownPrime
  - Speculative cacheline invalidations versus speculative cache pollution
  - Software mitigation is the same as for Meltdown & Spectre

- Key overall philosophy:
  - Move from ad hoc analysis to formal automated synthesis.
  - Span software, OS, and hardware for holistic hardware-aware analysis

[Trippel, Lustig, Martonosi. https://arxiv.org/abs/1802.03802]
[Trippel, Lustig, Martonosi. MICRO-51. October, 2018]

Acknowledgements

- CheckMate Co-Authors: Caroline Trippel, Princeton CS PhD student and Daniel Lustig, NVIDIA

- Funding: NSF, NVIDIA Graduate Fellowship

- Check Tools, additional co-authors: Yatin Manerkar, Abhishek Bhattacharjee, Michael Pellauer, Geet Sethi

Me: http://www.princeton.edu/~mrm

Group Papers: http://mrmgroup.cs.princeton.edu

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

Thank you!

Microsoft