Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture

CARRV 2019 – June 22nd, 2019 - Phoenix, Arizona

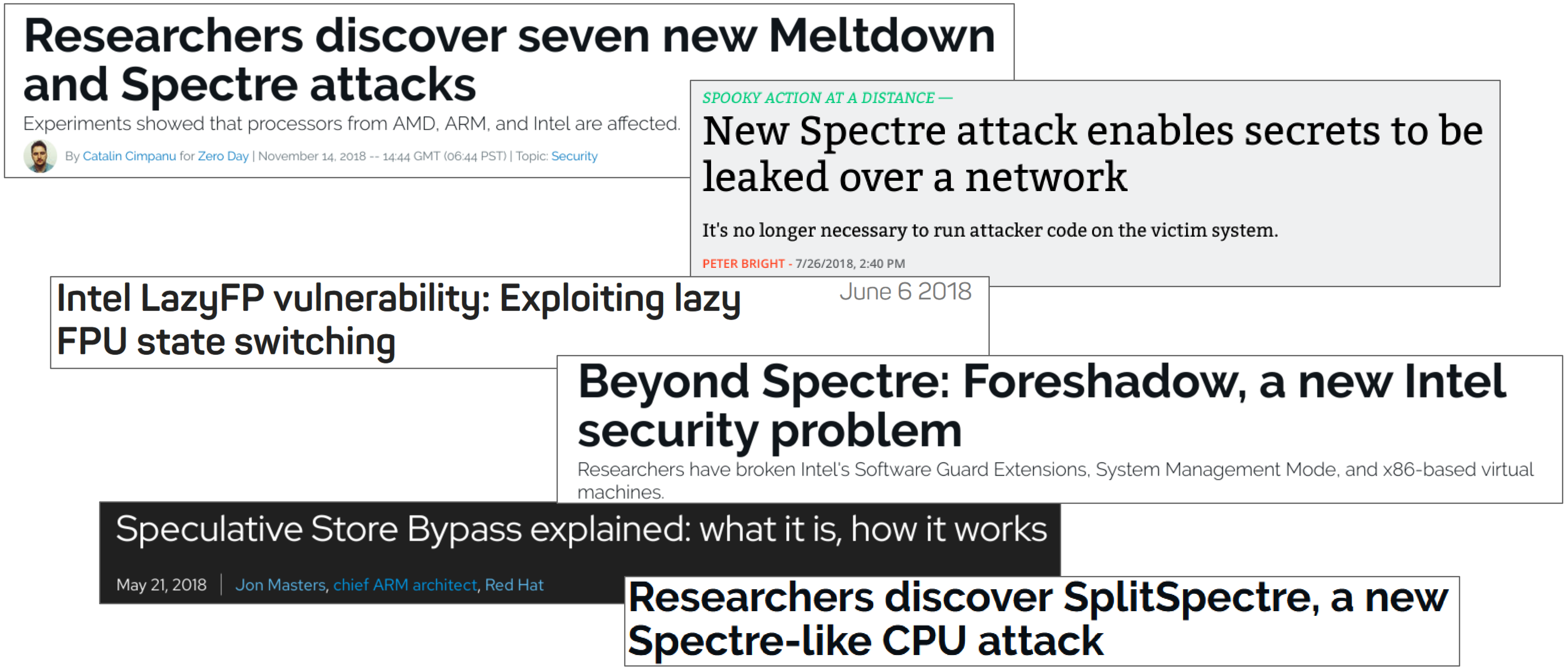
Abraham Gonzalez, Ben Korpan, *Jerry Zhao*, Ed Younis
Krste Asanović
University of California, Berkeley

Outline

- Motivation
- Open-source Approach to Hardware
  - BOOM: Berkeley Out-of-Order Machine
- Replicating Spectre Attacks on BOOM
- Implementing a Speculation Buffer
  - Comparisons
  - Implementation
- Conclusion

Motivation

Exploits Everywhere



Researchers discover seven new Meltdown and Spectre attacks

Experiments showed that processors from AMD, ARM, and Intel are affected.

By Catalin Cimpanu for Zero Day | November 14, 2018 -- 14:44 GMT (06:44 PST) | Topic: Security

SPOOKY ACTION AT A DISTANCE —

New Spectre attack enables secrets to be leaked over a network

It's no longer necessary to run attacker code on the victim system.

PETER BRIGHT - 7/26/2018, 2:40 PM

Intel LazyFP vulnerability: Exploiting lazy FPU state switching

June 6 2018

Beyond Spectre: Foreshadow, a new Intel security problem

Researchers have broken Intel's Software Guard Extensions, System Management Mode, and x86-based virtual machines.

Speculative Store Bypass explained: what it is, how it works

May 21, 2018 | Jon Masters, chief ARM architect, Red Hat

Researchers discover SplitSpectre, a new Spectre-like CPU attack

Why are Spectre-style attacks hard?

Spectre Variations

Attack Scenarios

- User process attacks kernel
- User process attacks user space
- Intra-process sandbox escape
- User process attacks enclaves
- Remote timing attacks
- ...

Leakage Mechanisms

- Conditional branch
- Indirect jump
- Return instructions
- Speculative store bypass
- Data speculation
- ...

Covert Channels

- Changes in cache state
- Power consumption
- Resource contention (FPUs, buffers)
- ...

Target CPUs

- ARM
- Intel
- AMD
- RISC-V
- …

Taken from "Panel On the Implications of the Meltdown & Spectre Design Flaws", ISCA 2018

Mitigation Approaches

InvisiSpec/SafeSpec: Blocking unsafe loads from altering the data cache

DAWG: Partition data cache between security domains

StealthMem/CATalyst: Hide visibility of a secure memory region

Context-based fencing: Dynamically stop speculation in secure code

Compiler-inserted fencing: Statically analyze program for Spectre-vulnerable snippets

Lots of interesting approaches, but how to compare them?

Use them together?

M. Yan, et. al. 2018. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In MICRO.
K. N. Khasawneh, et. al. 2018. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. Archived.
V. Kiriansky, et. al. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In MICRO.
T. Kim, et. al. 2012. STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud. In USENIX.
F. Liu, et. al. 2016. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In HPCA.
M. Taram, et. Al. 2019. Context-Sensitive Fencing: Securing Speculative Execution via Microcode Customization. In ASPLOS.
Microsoft. 2018. Microsoft's compiler-level Spectre fix shows how hard this problem will be to solve. In Ars Technica.

Open-source Approach to Hardware

Open-source HW + Agile Design Tools + Fast Simulation/Emulation = Security?

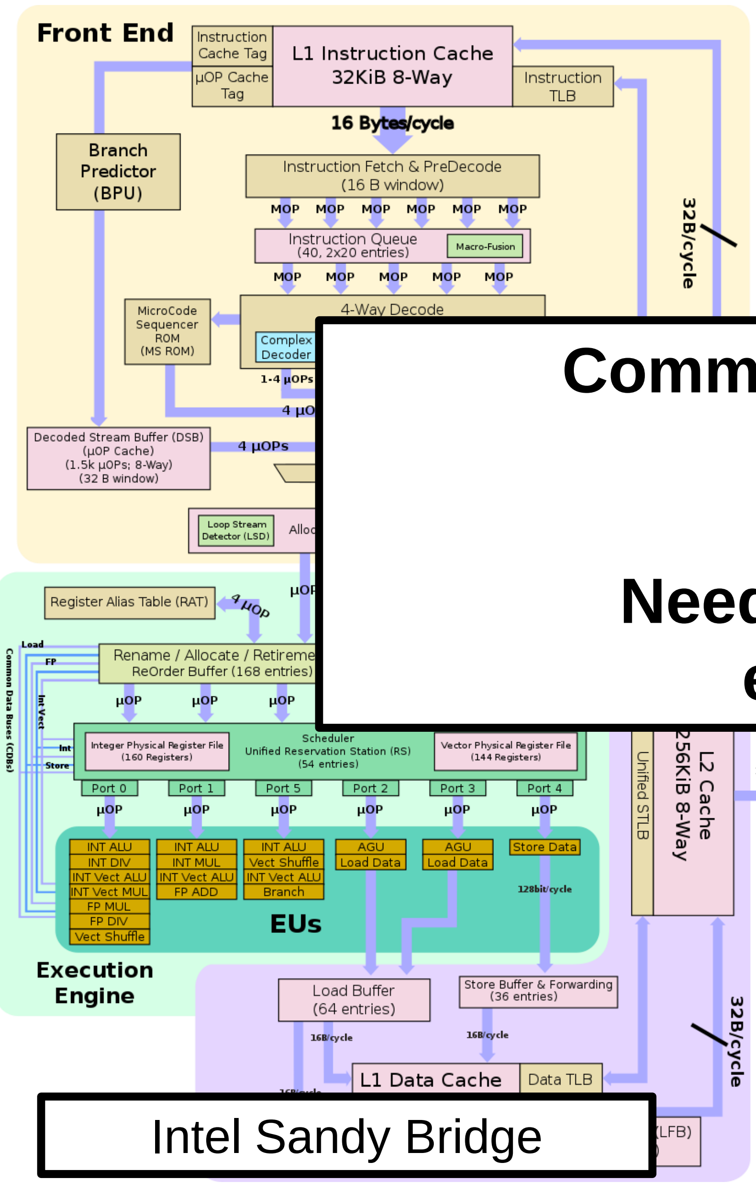
Large proliferation of open-source software stacks, cores, and simulation/design infrastructure

The Open-source RISC-V Approach

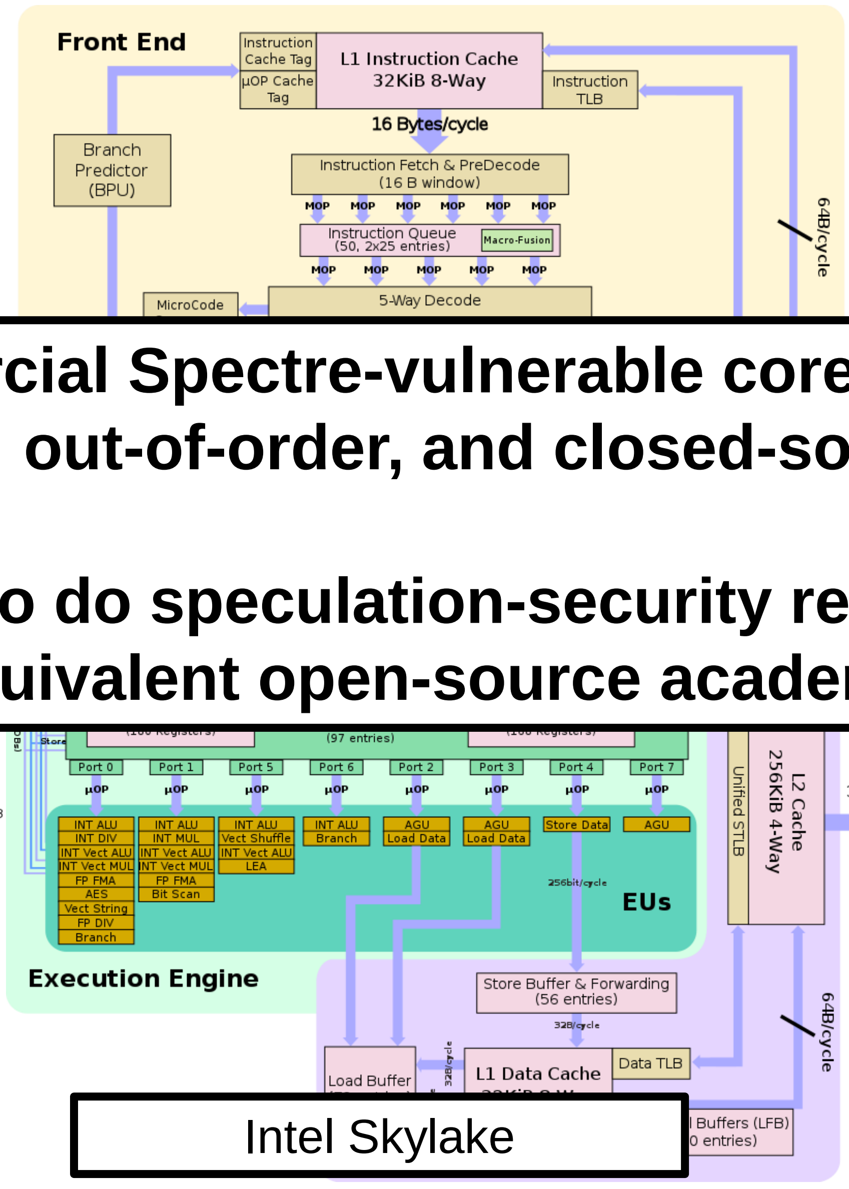
Security benefits from open-source work

1. Think of new security mitigation/exploit
2. Use open-source RTL to start implementation
3. Quickly iterate through design development with easy, fast, and free tooling
4. Open-source work and have others scrutinize or use your work

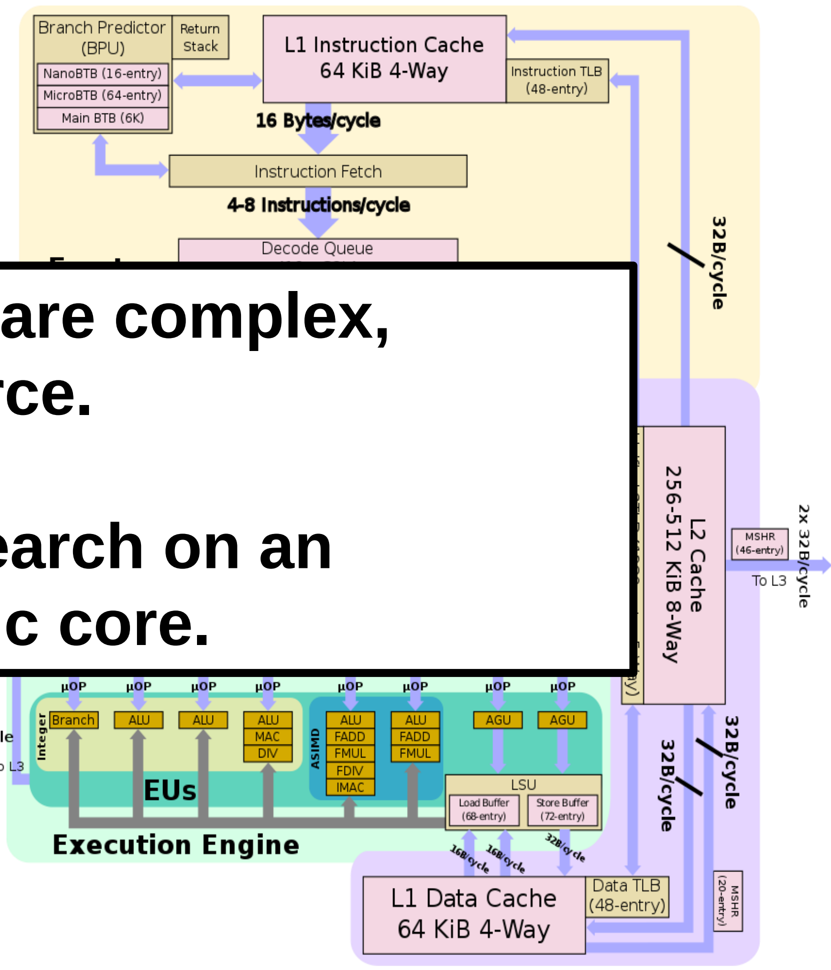
Modern Microarchitectures


Front End
Instruction Cache Tag
L1 Instruction Cache 32KiB 8-Way
µOP Cache Tag
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
Instruction Queue (40, 2x20 entries)
Macro-Fusion
MicroCode Sequencer ROM (MS ROM)
4-Way Decode
Complex Decoder
Decoded Stream Buffer (DSB) (µOP Cache) (1.5k µOPs; 8-Way) (32 B window)
Loop Stream Detector (LSD)
32B/cycle
Register Alias Table (RAT)
Rename / Allocate / Retirement ReOrder Buffer (168 entries)
Integer Physical Register File (160 Registers)
Scheduler Unified Reservation Station (RS) (54 entries)
Vector Physical Register File (144 Registers)
Port 0
Port 1
Port 5
Port 2
Port 3
Port 4
EUs
Execution Engine
Load Buffer (64 entries)
Store Buffer & Forwarding (36 entries)
L1 Data Cache
Data TLB
L2 Cache 256KiB 8-Way
Unified STLB
To L3

Intel Sandy Bridge



Intel Skylake



ARM A76

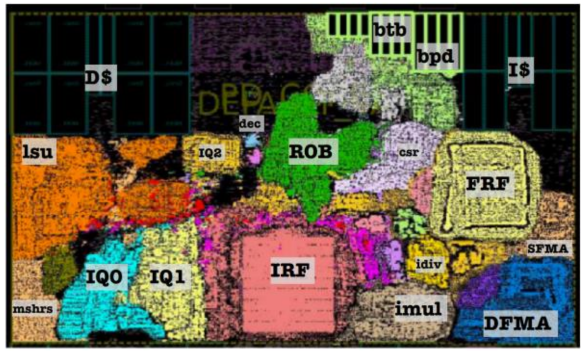
Commercial Spectre-vulnerable cores are complex, out-of-order, and closed-source.

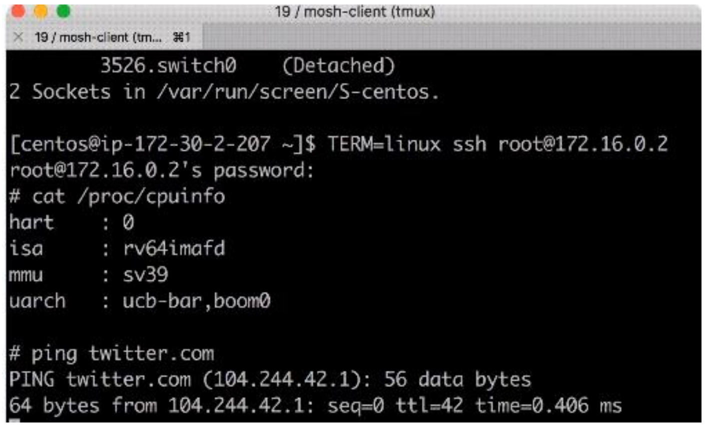
Need to do speculation-security research on an equivalent open-source academic core.

BOOM: The Berkeley Out-of-Order Machine

BOOM Overview

- Open-source, out-of-order, superscalar RISC-V core
- Runs RISC-V ISA RV64GC
- Linux-capable - boots Fedora + Buildroot
- Silicon-proven - taped out
- ~18K LoC of open-source Chisel RTL
- Highly parameterizable and configurable
- Full integration with Rocket Chip, FireSim, HAMMER





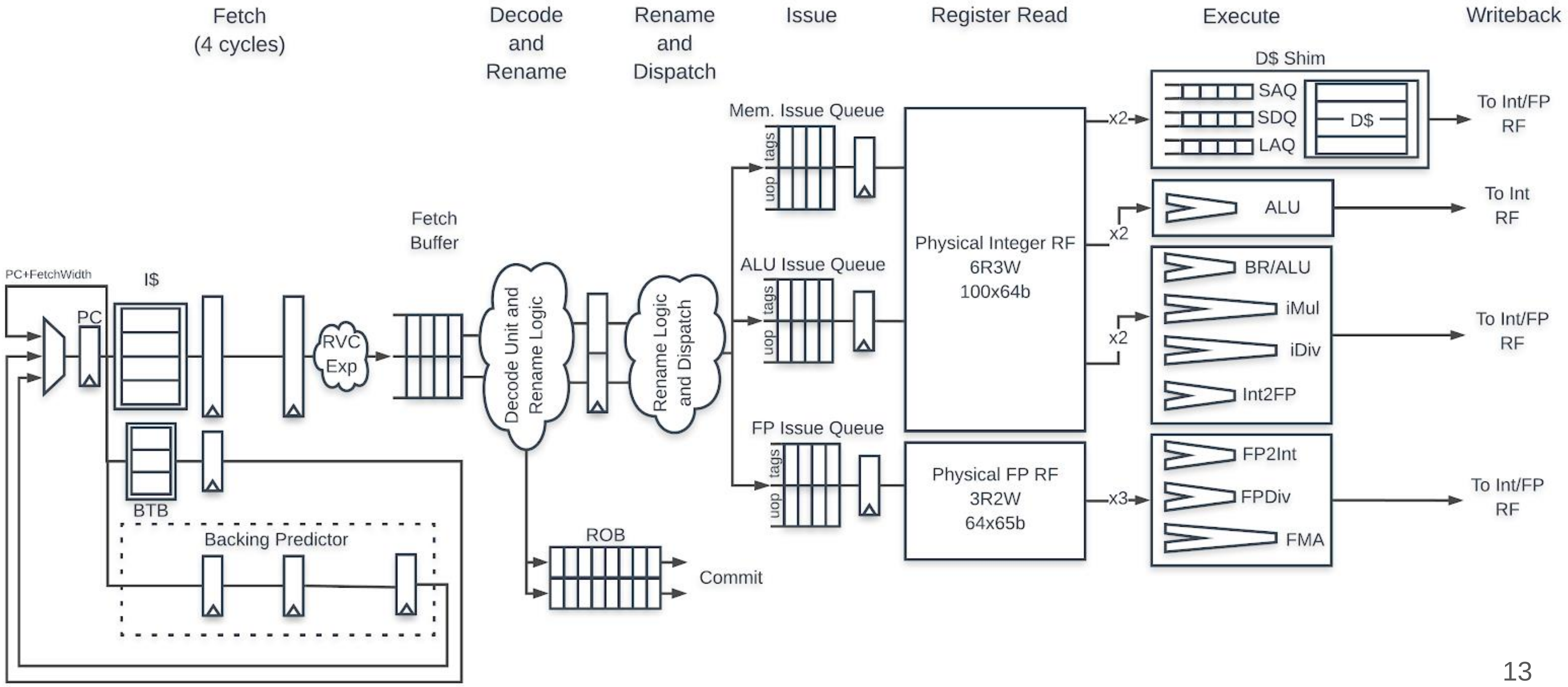
```
        3526.switch0    (Detached)
2 Sockets in /var/run/screen/S-centos.

[centos@ip-172-30-2-207 ~]$ TERM=linux ssh root@172.16.0.2
root@172.16.0.2's password:
# cat /proc/cpuinfo
hart    : 0
isa     : rv64imafd
mmu     : sv39
uarch   : ucb-bar,boom0

# ping twitter.com
PING twitter.com (104.244.42.1): 56 data bytes
64 bytes from 104.244.42.1: seq=0 ttl=42 time=0.406 ms
```

J. Bachrach, et. al. 2012. Chisel: constructing hardware in a scala embedded language. In DAC.
K. Asanovic, et. al. 2016. The Rocket Chip Generator. Technical Report.
S. Karandikar, et. al. 2018. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In ISCA.
E. Wang, et. al. 2018. Hammer: Enabling Reusable Physical Design. In WOSET.

BOOM Microarchitecture



Replicating Spectre Attacks

Spectre v1 Overview

Speculation:

- Performance-seeking behavior of modern processors
- Execute instructions before we know they will commit

Side-channel:

- Microarchitectural state which holds interacts with program execution
- Caches, TLBs, power…

Typical Spectre attack:

1. Setup processor to misspeculate in victim code (e.g. train branch predictors)
2. Misspeculation leaks secret into a side channel
3. Attacker recovers secret from side channel

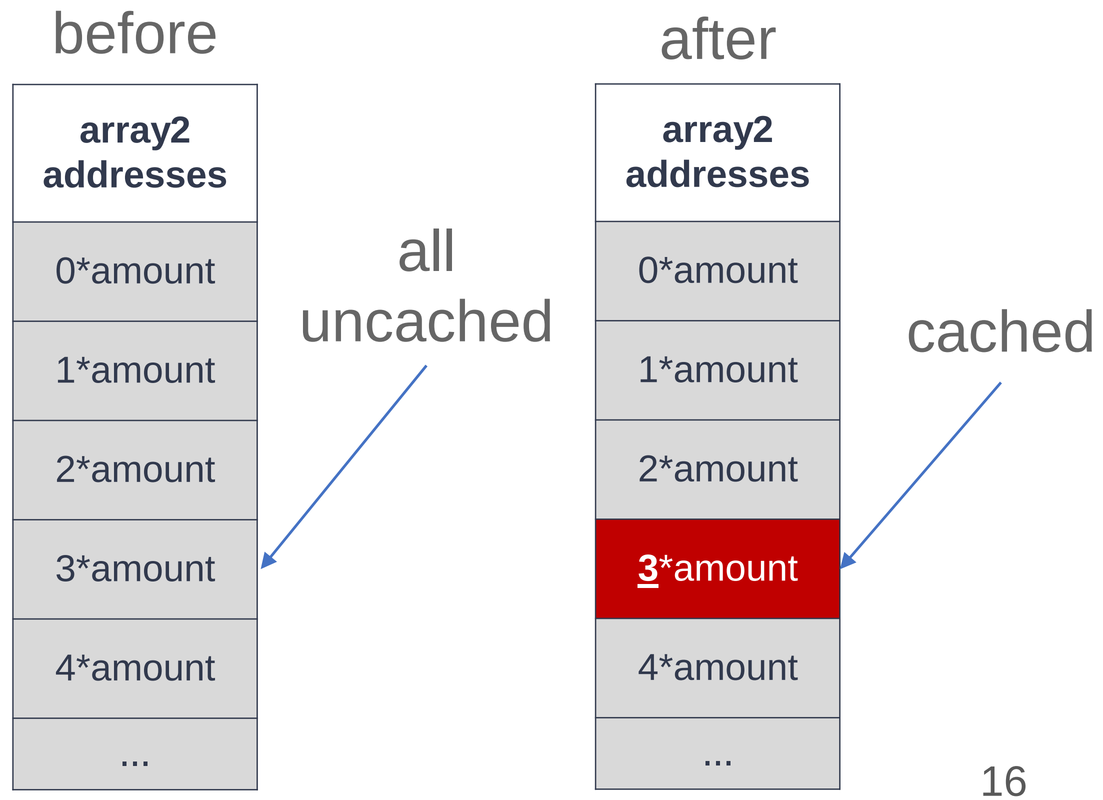
P. Kocher, et. al. 2018. Spectre attacks: Exploiting speculative execution. Archived.

Spectre v1 Example

Steps:

1. Access *if* statement multiple times correctly (predict *if* to fall-through)
2. Give *x > array1_sz*
3. Predict the *if* to be true and bring in *secret* and *array2* value
4. Use the time difference between cached and uncached lines to determine *secret*
5. Repeat!

```
if (x < array1_sz):
    secret = array1[x]
    out = array2[secret * amount]
```

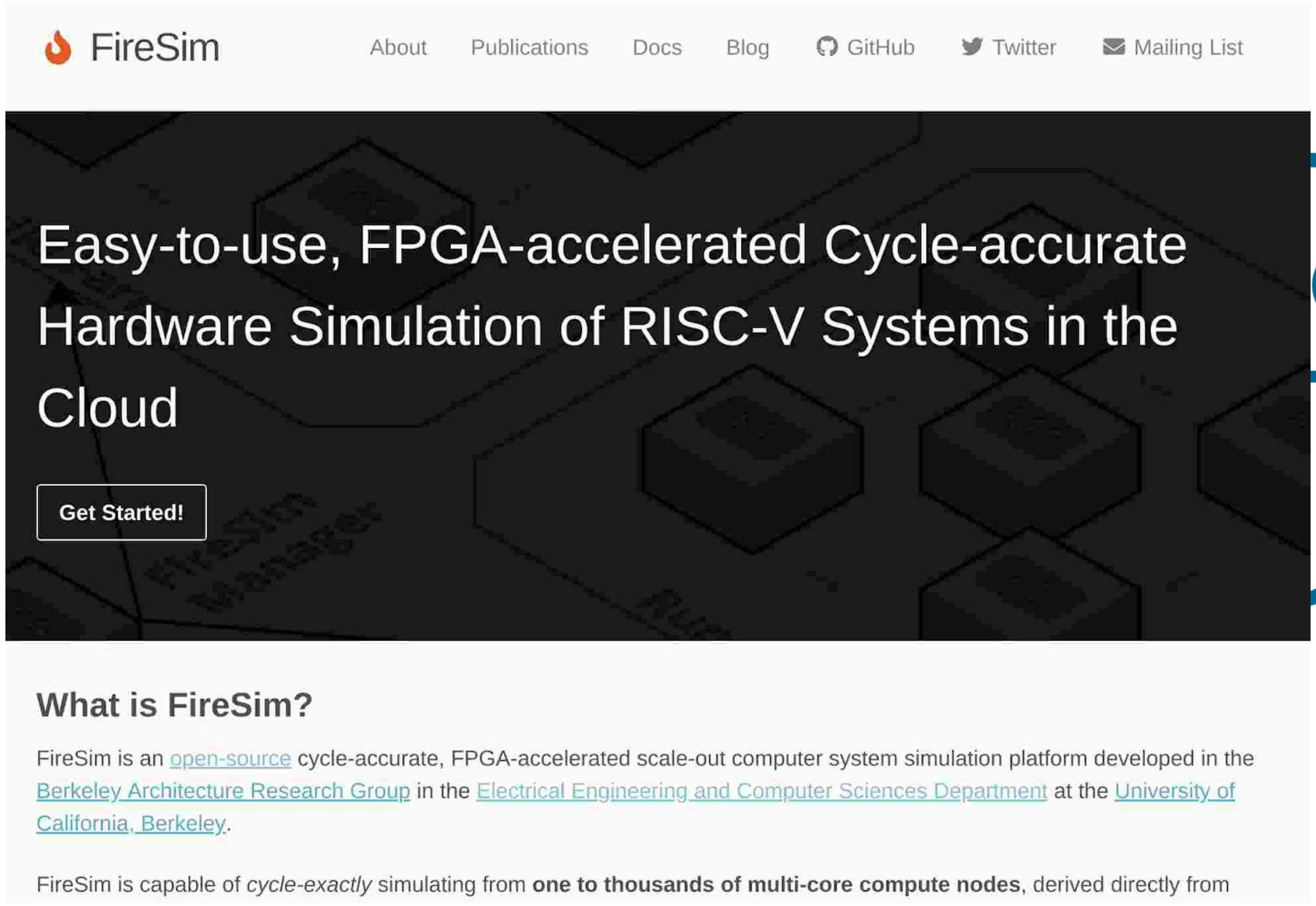


Components Needed – With BOOM?

- Branch Prediction
  - Set associative BTB and GShare branch predictors
- Speculative Execution
  - Out-of-order execution and branch kill masks for speculative execution
- Caching
  - L1 data cache and a outer memory set to the latency of an L2 cache
- Cache Manipulation
  - Custom-made L1 data cache *`clflush`*

BOOM provides all the elements to replicate Spectre!

Spectre v1 Running on FireSim


FireSim About Publications Docs Blog GitHub Twitter Mailing List

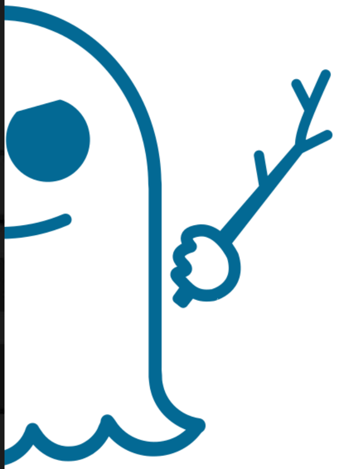
Easy-to-use, FPGA-accelerated Cycle-accurate Hardware Simulation of RISC-V Systems in the Cloud

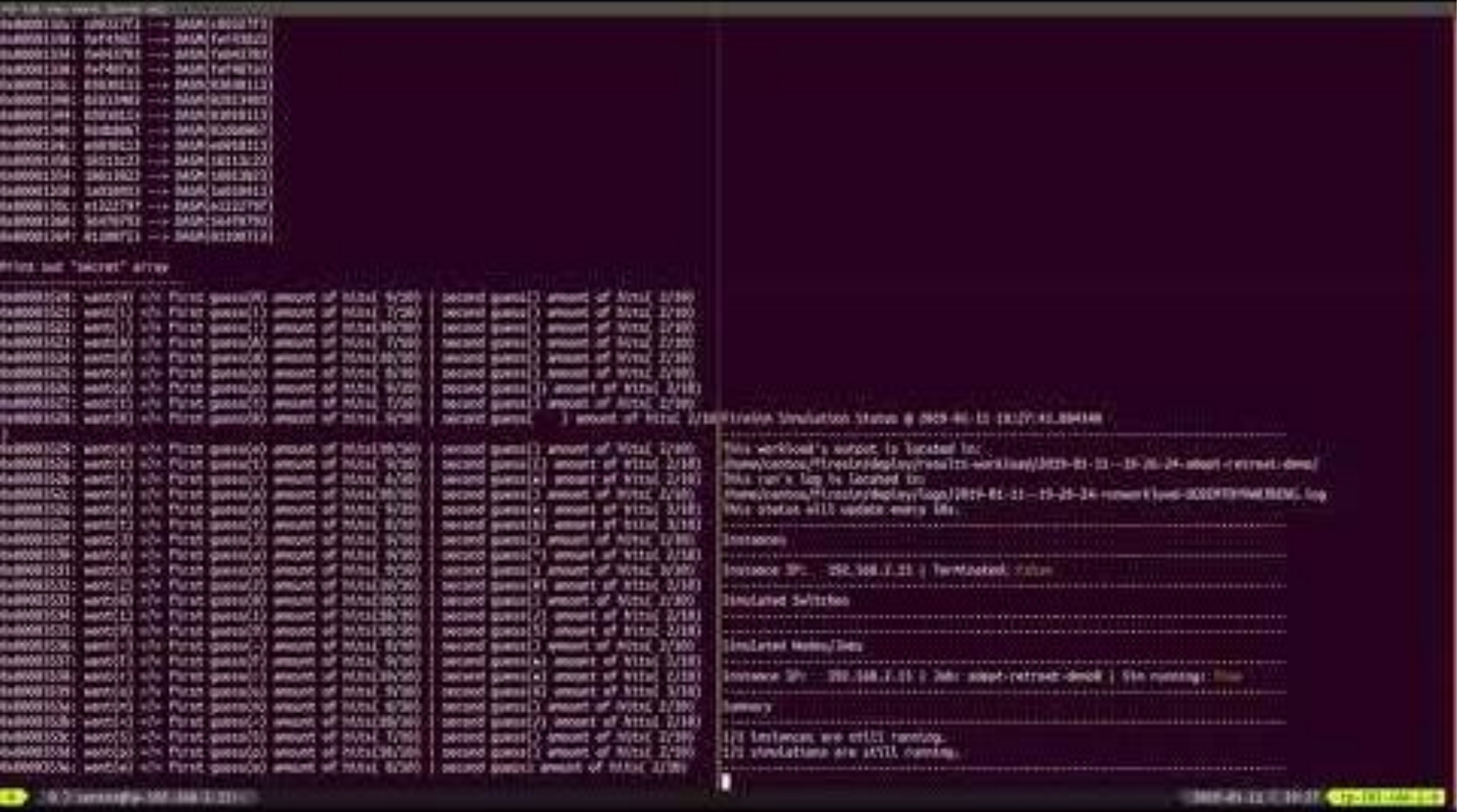
Get Started!

What is FireSim?

FireSim is an open-source cycle-accurate, FPGA-accelerated scale-out computer system simulation platform developed in the Berkeley Architecture Research Group in the the Electrical Engineering and Computer Sciences Department at the University of California, Berkeley.

FireSim is capable of *cycle-exactly* simulating from **one to thousands of multi-core compute nodes**, derived directly from





```
[illegible]
0x8000130c: [illegible] --> DASM([illegible])
0x80001330: [illegible] --> DASM([illegible])
0x80001334: [illegible] --> DASM([illegible])
0x80001338: [illegible] --> DASM([illegible])
0x8000133c: [illegible] --> DASM([illegible])
0x80001340: [illegible] --> DASM([illegible])
0x80001344: [illegible] --> DASM([illegible])
0x80001348: [illegible] --> DASM([illegible])
0x8000134c: [illegible] --> DASM([illegible])
0x80001350: [illegible] --> DASM([illegible])
0x80001354: [illegible] --> DASM([illegible])
0x80001358: [illegible] --> DASM([illegible])
0x8000135c: [illegible] --> DASM([illegible])
0x80001360: [illegible] --> DASM([illegible])
0x80001364: [illegible] --> DASM([illegible])

Print out "secret" array
------------------------
0x80001520: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess() amount of hits( 2/100)
0x80001521: want([illegible]) =?= First guess([illegible]) amount of hits( 7/100) | second guess() amount of hits( 2/100)
0x80001522: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x80001523: want([illegible]) =?= First guess([illegible]) amount of hits( 7/100) | second guess() amount of hits( 2/100)
0x80001524: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x80001525: want([illegible]) =?= First guess([illegible]) amount of hits( 8/100) | second guess() amount of hits( 2/100)
0x80001526: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess() amount of hits( 3/100)
0x80001527: want([illegible]) =?= First guess([illegible]) amount of hits( 7/100) | second guess() amount of hits( 2/100)
0x80001528: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess(   ) amount of hits( 2/100)
}
0x80001529: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x8000152a: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess() amount of hits( 2/100)
0x8000152b: want([illegible]) =?= First guess([illegible]) amount of hits( 6/100) | second guess([illegible]) amount of hits( 2/100)
0x8000152c: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x8000152d: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess([illegible]) amount of hits( 3/100)
0x8000152e: want([illegible]) =?= First guess([illegible]) amount of hits( 8/100) | second guess([illegible]) amount of hits( 3/100)
0x8000152f: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess() amount of hits( 2/100)
0x80001530: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess([illegible]) amount of hits( 2/100)
0x80001531: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess() amount of hits( 3/100)
0x80001532: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess([illegible]) amount of hits( 3/100)
0x80001533: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x80001534: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess([illegible]) amount of hits( 2/100)
0x80001535: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess([illegible]) amount of hits( 2/100)
0x80001536: want([illegible]) =?= First guess([illegible]) amount of hits( 8/100) | second guess() amount of hits( 2/100)
0x80001537: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess([illegible]) amount of hits( 2/100)
0x80001538: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess([illegible]) amount of hits( 2/100)
0x80001539: want([illegible]) =?= First guess([illegible]) amount of hits( 9/100) | second guess([illegible]) amount of hits( 3/100)
0x8000153a: want([illegible]) =?= First guess([illegible]) amount of hits( 8/100) | second guess() amount of hits( 2/100)
0x8000153b: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess([illegible]) amount of hits( 2/100)
0x8000153c: want([illegible]) =?= First guess([illegible]) amount of hits( 7/100) | second guess() amount of hits( 2/100)
0x8000153d: want([illegible]) =?= First guess([illegible]) amount of hits(10/100) | second guess() amount of hits( 2/100)
0x8000153e: want([illegible]) =?= First guess([illegible]) amount of hits( 8/100) | second guess() amount of hits( 2/100)
```

```
FireSim Simulation Status @ 2019-01-11 19:27:41.884148
--------------------------------------------------------------------------------
This workload's output is located in:
/home/centos/firesim/deploy/results-workload/2019-01-11--19-26-24-smoot-retreat-demo/
This run's log is located in:
/home/centos/firesim/deploy/logs/2019-01-11--19-25-24-runworkload-[illegible].log
This status will update every 10s.
--------------------------------------------------------------------------------
Instances
--------------------------------------------------------------------------------
Instance IP:   192.168.2.21 | Terminated: False
--------------------------------------------------------------------------------
Simulated Switches
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Simulated Nodes/Jobs
--------------------------------------------------------------------------------
Instance IP:   192.168.2.21 | Job: smoot-retreat-demo0 | Sim running: True
--------------------------------------------------------------------------------
Summary
--------------------------------------------------------------------------------
1/1 instances are still running.
1/1 simulations are still running.
--------------------------------------------------------------------------------
```

Implementing a Speculation Buffer

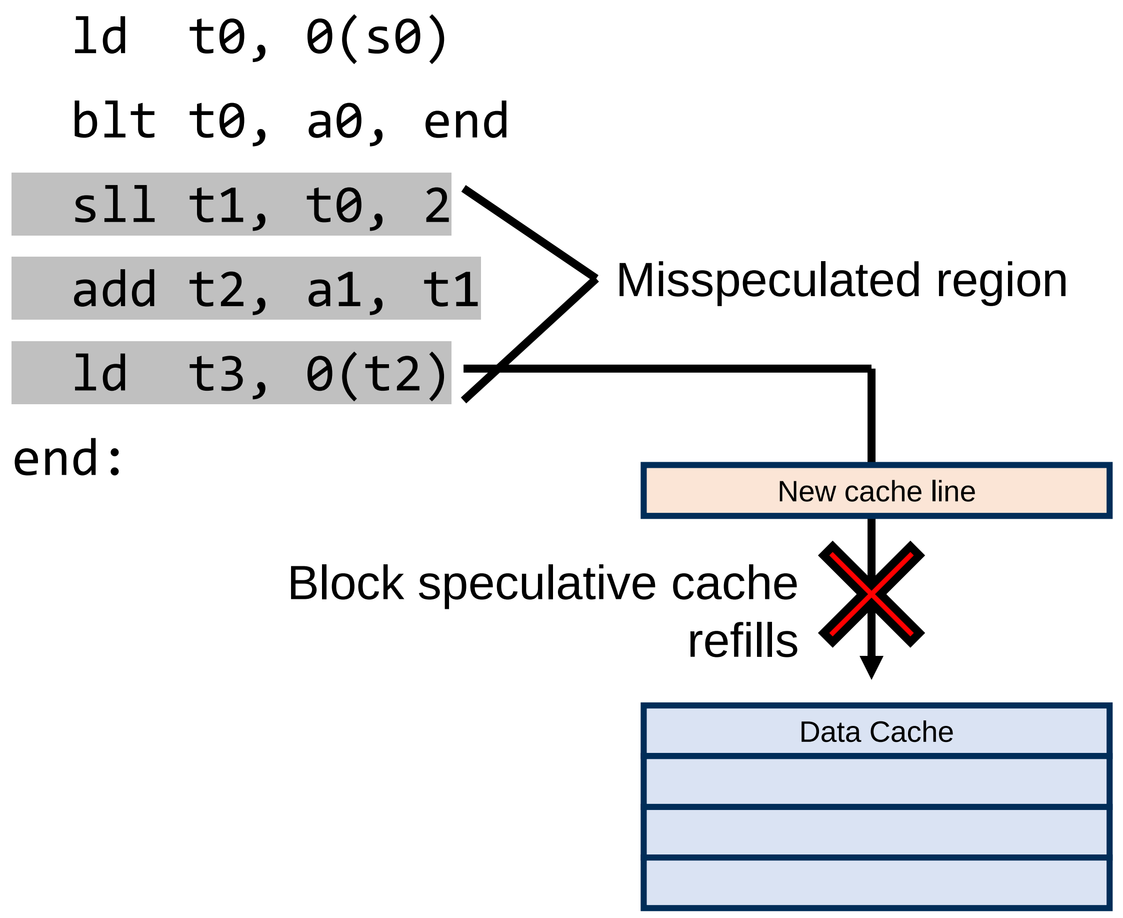
Protecting Data Caches

Problem: Load refills are not subject to architectural guarantees

- Misspeculated loads leave **side-effects**, creating a side-channel

Solution: Treat the data cache as an architectural structure

- Only alter the cache state when instructions **commit**
- Implement a working prototype in BOOM RTL



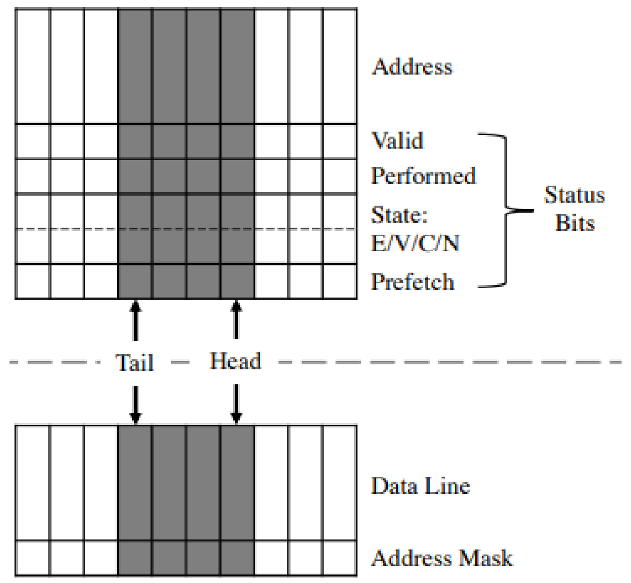
```
  ld  t0, 0(s0)
  blt t0, a0, end
  sll t1, t0, 2
  add t2, a1, t1
  ld  t3, 0(t2)
end:
```

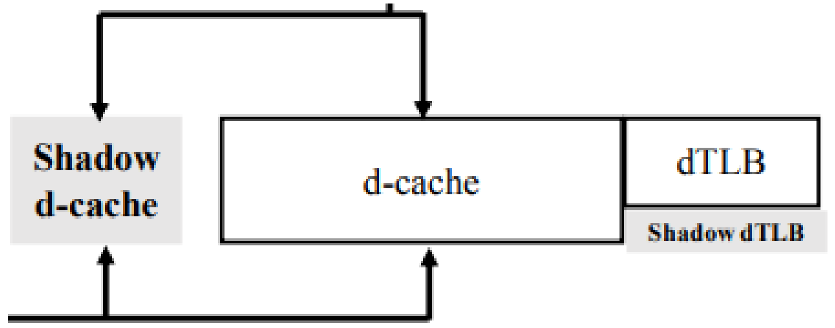
Prior Work

InvisiSpec
- Per load-queue-entry speculation buffer
- Speculation-aware cache-coherence policy

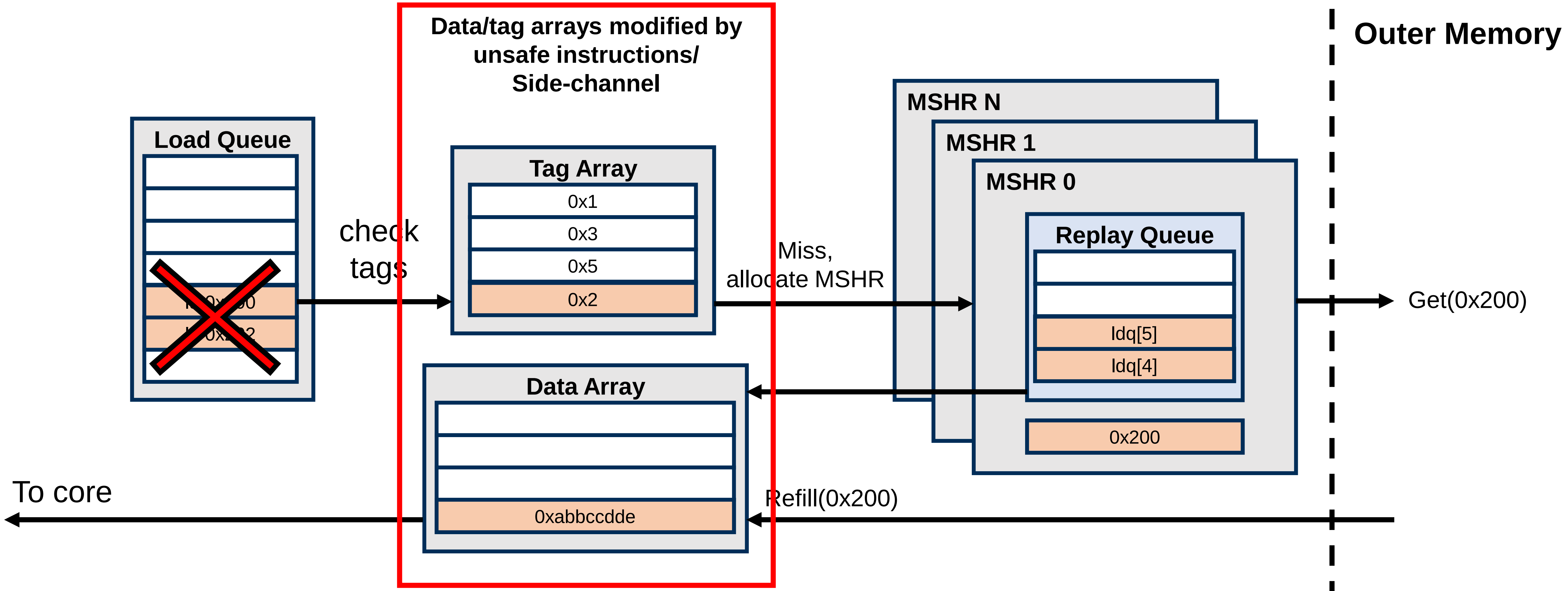
Safespec
- Speculation-depth sized “shadow structures”
- Protect DCache, ICache, TLBs

BOOM Speculation Buffer:
- Hold speculated loads in **line-fill-buffers**

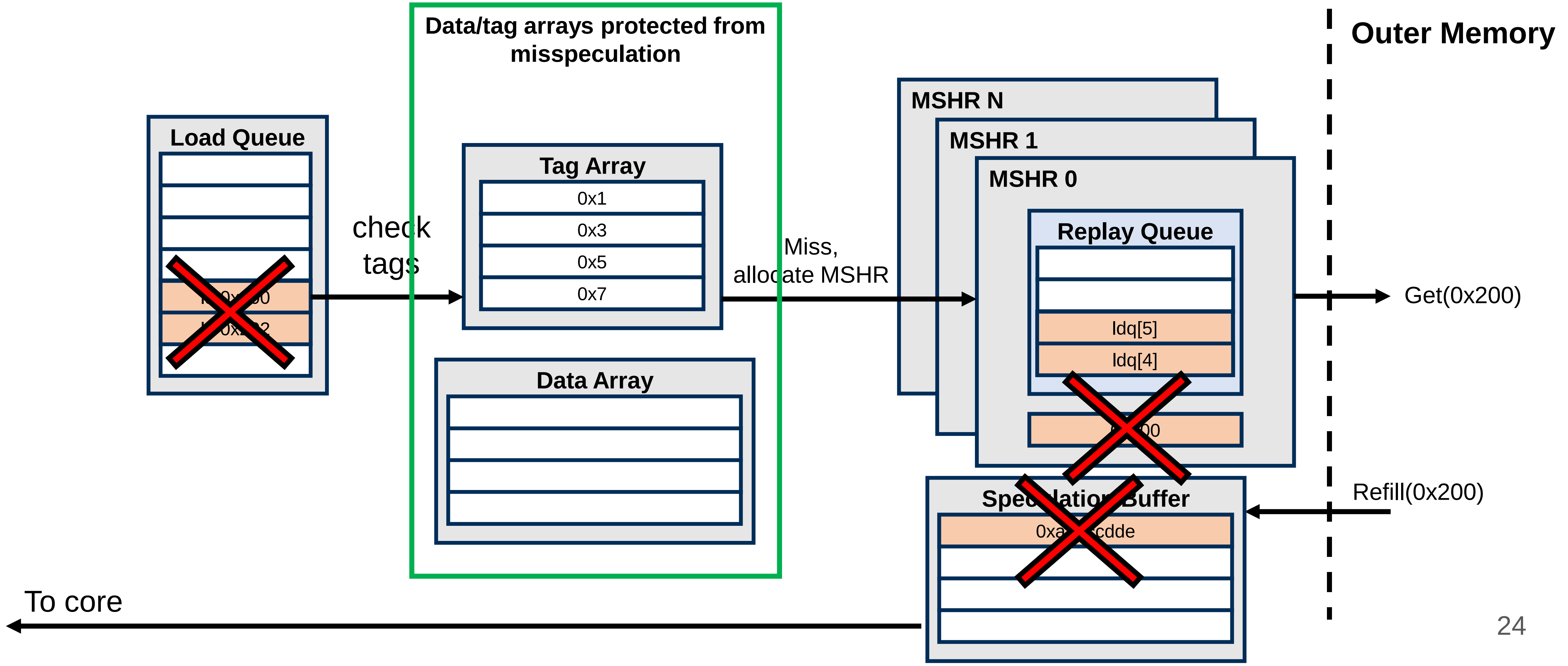




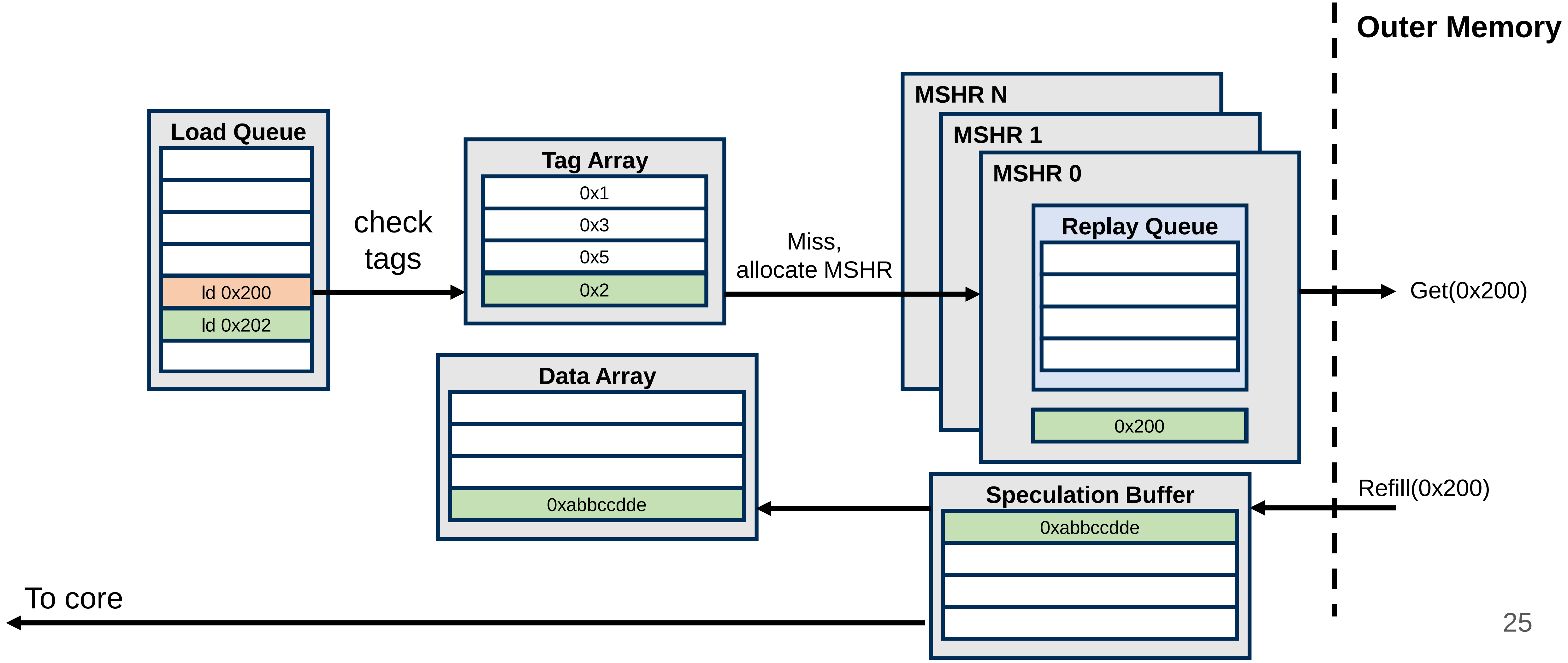
Life of a Misspeculated Load



Blocking Misspeculated Loads



Blocking Misspeculated Loads



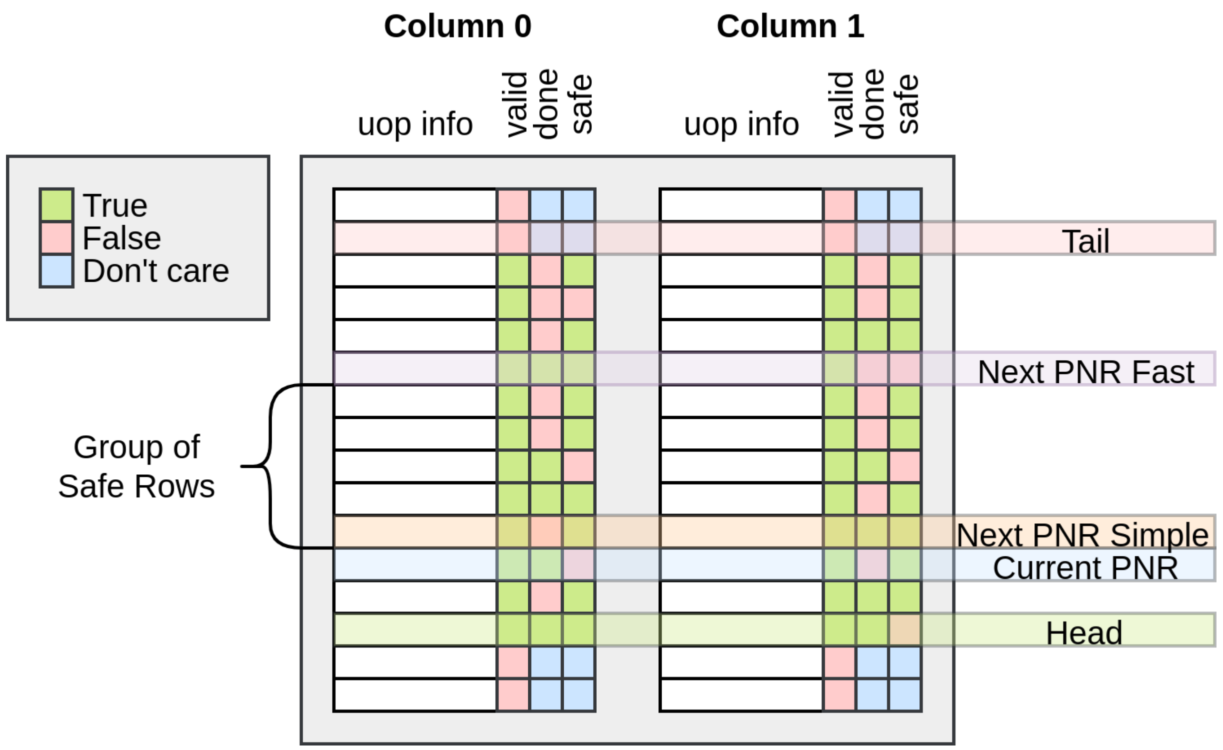
Blocking Misspeculated Loads

- Load refills wait in the buffer until one of their misses has committed
- Stall writeback until one of the following occurs
  - A load-miss to that line has committed OR
  - A store-miss hits that line (stores are non-speculative)
- If all load misses to that line were misspeculated, discard it
- Bypass loads out of the load-fill-buffer
  - Subsequent loads “see” the data in the DCache
  - Minimizes performance penalty

Committing Loads

When to commit load refills to the DCache?

- When the ROB commits the load?
  - Most secure.
  - Huge performance penalty for load misses
- When the load is free from branches?
  - Does not consider exceptions/interrupts
  - Minimal performance penalty
- When the load reaches the **point-of-no-return**
  - New ROB pointer, tracks instructions which are guaranteed to commit



Speculation Buffer Results

1 month implementation time

Microbenchmarks

- Set of assembly routines to test edge cases

Dhrystone results

- Original: 2176 dps
- W. Speculation buffer: 2216 dps
- Impact: ~2% better IPC

Preliminary physical results in TSMC 45nm

- ~3% larger area

	Version of BOOM		
Benchmark	Normal	With Speculation Buffer	% Difference
Non-speculative LD misses to same sets	540 cycles	640 cycles	-19%
Non-speculative LD misses to different sets	264 cycles	297 cycles	-11%
MSHR evicted speculative LD misses	48 cycles	67 cycles	-40%
Dhrystone	2176 dps	2216 dps	+2%

Comparison

	InvisiSpec	SafeSpec	BOOM Speculation Buffer
Implementation Platform	Custom GEM5	Marssx86	BOOM RTL
Buffer size	Additional cacheline * load-queue-size	Additional cacheline * speculation depth	Repurposed line-fill-buffers
Commit condition	Wait for branch OR Wait for non-speculative	Wait for branch OR Wait for commit	Wait for point-of-no-return
Physical design feedback	CACTI estimates	CACTI estimates	Trial TSMC 45nm implementation
Protected components	L1D, LLC, multicores	L1D, L1I, TLBs	L1D
Performance impact	-22% performance	+3% performance	+2% performance

Conclusion

Conclusion

Demonstrated application of RISC-V ecosystem towards secure hardware

- Working demonstrations of Spectre attacks on a RISC-V core
- RTL of Spectre mitigation available in an open-source core

Continue improving BOOM security

- Secure other structures: TLBs, ICache, LLC, branch predictors
- Enable secure enclave execution

BOOMv3 Tapeout + More Attacks

- Planning to add Speculation Buffer and CSRs to enable/disable it
- More attacks with different predictors/structures (TAGE, RAS, etc)

Questions?

Thanks CARRV19!

Contact: {abe.gonzalez,bkorpan,jzh,edyounis,krste}@berkeley.edu

Links:

- *Core*: boom-core.org
- *Github*: github.com/riscv-boom
- *FireSim*: fires.im
- *HAMMER*: github.com/ucb-bar/hammer

Thanks:

- Chris Celio, David Kohlbrenner
- UCB ADEPT Lab