Time Protection: Preventing Timing Channels

Nils Wistoff, Moritz Schneider, Frank K. Gürkaynak, Luca Benini, Gernot Heiser

- Presentation to RISC-V SSC+PSC, June'20

https://trustworthy.systems

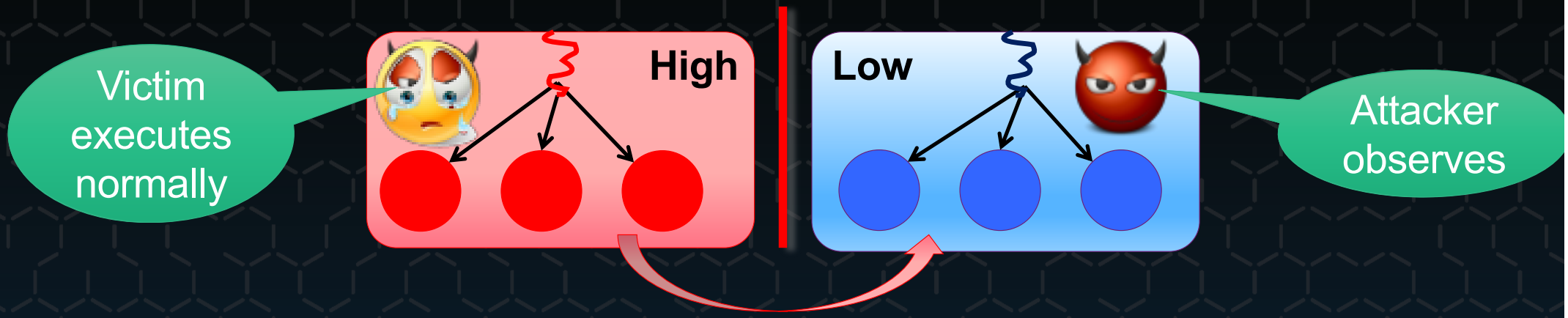
Aim: Eliminate Microarchitectural Timing Channels

Timing Channels

Information leakage through timing of events

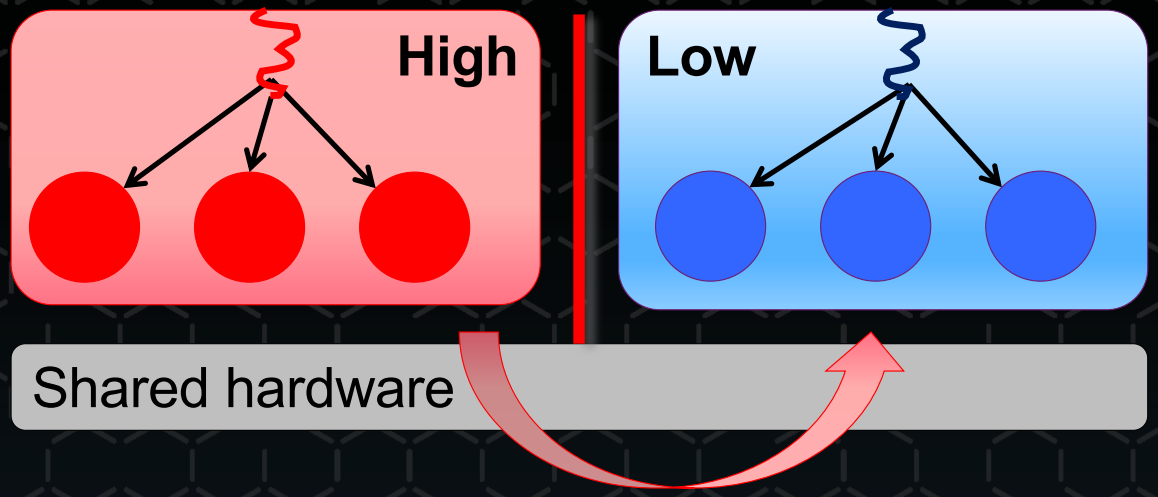
- Typically by observing response latencies or own execution speed

Covert channel: Information flow that bypasses the security policy



Side channel: Covert channel exploitable without insider help

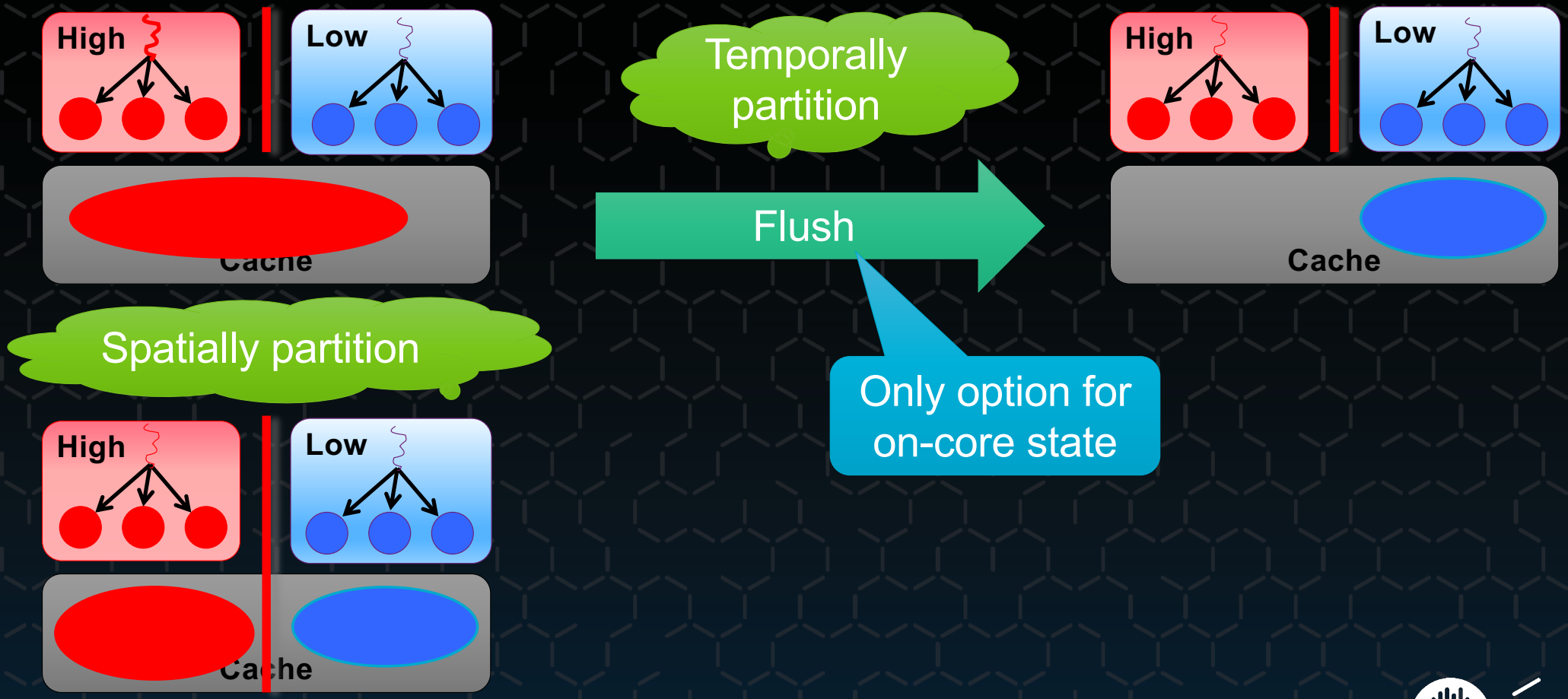
Cause: Competition for Shared HW Resources



Affect execution speed

- Inter-process interference
- Competing access to micro-architectural features
- Hidden by the HW-SW contract!

Time Protection: Partition Hardware



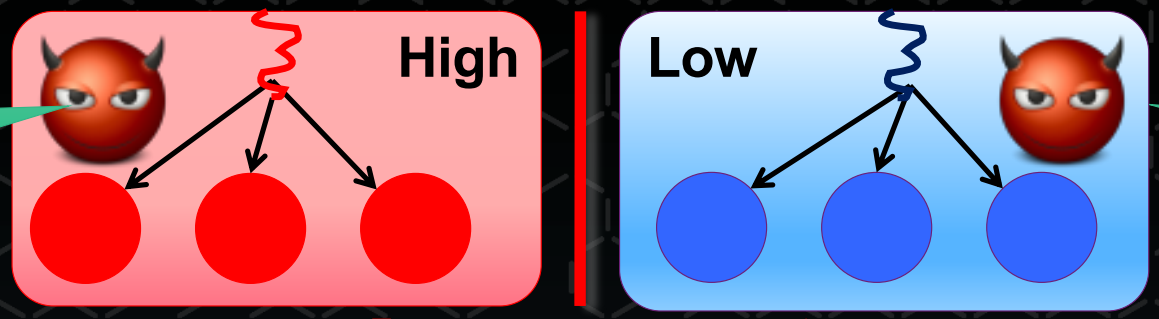
Flushing On-Core State

Evaluating Intra-Core Channels

Mitigation on Intel and Arm processors:

- Disable data prefetcher (just to be sure)
- On context switch, perform all architected flush operations:
  - Intel: wbinvd + invpcid (no targeted L1-cache flush supported!)
  - Arm: DCCISW + ICIALLU + TLBIALL + BPIALL

Methodology: Prime and Probe



Trojan encodes

Spy observes

1. Fill cache with own data

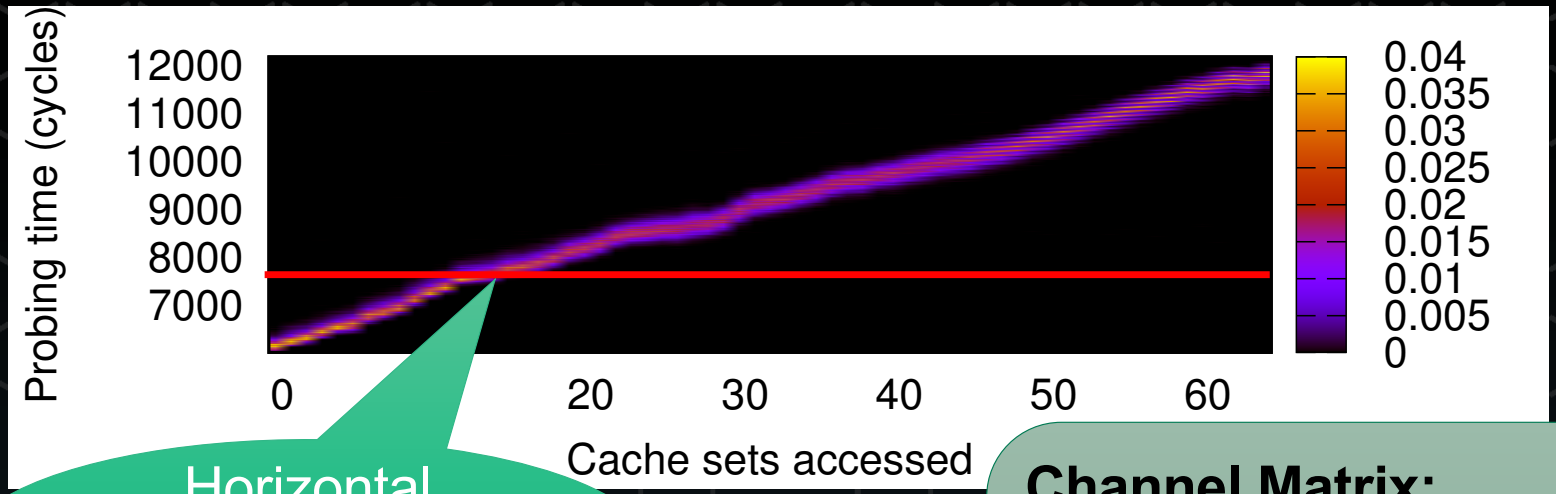
2. Touch *n* cache lines

Input Signal

3. Traverse cache, measure execution time

Output Signal

Methodology: Channel Matrix



Horizontal variation indicates channel

Uncloseable channels in all recent Intel and Arm processors examined!

Channel Matrix:

- Conditional probability of observing time, *t*, given input, *n*.
- Represented as heat map:
  - bright = high probability

Implementing Full Flush on RISC-V

Evaluation Platform

Application

Channel bench

- Measure covert channels
- Tailor attacks to Ariane microarch

OS

seL4

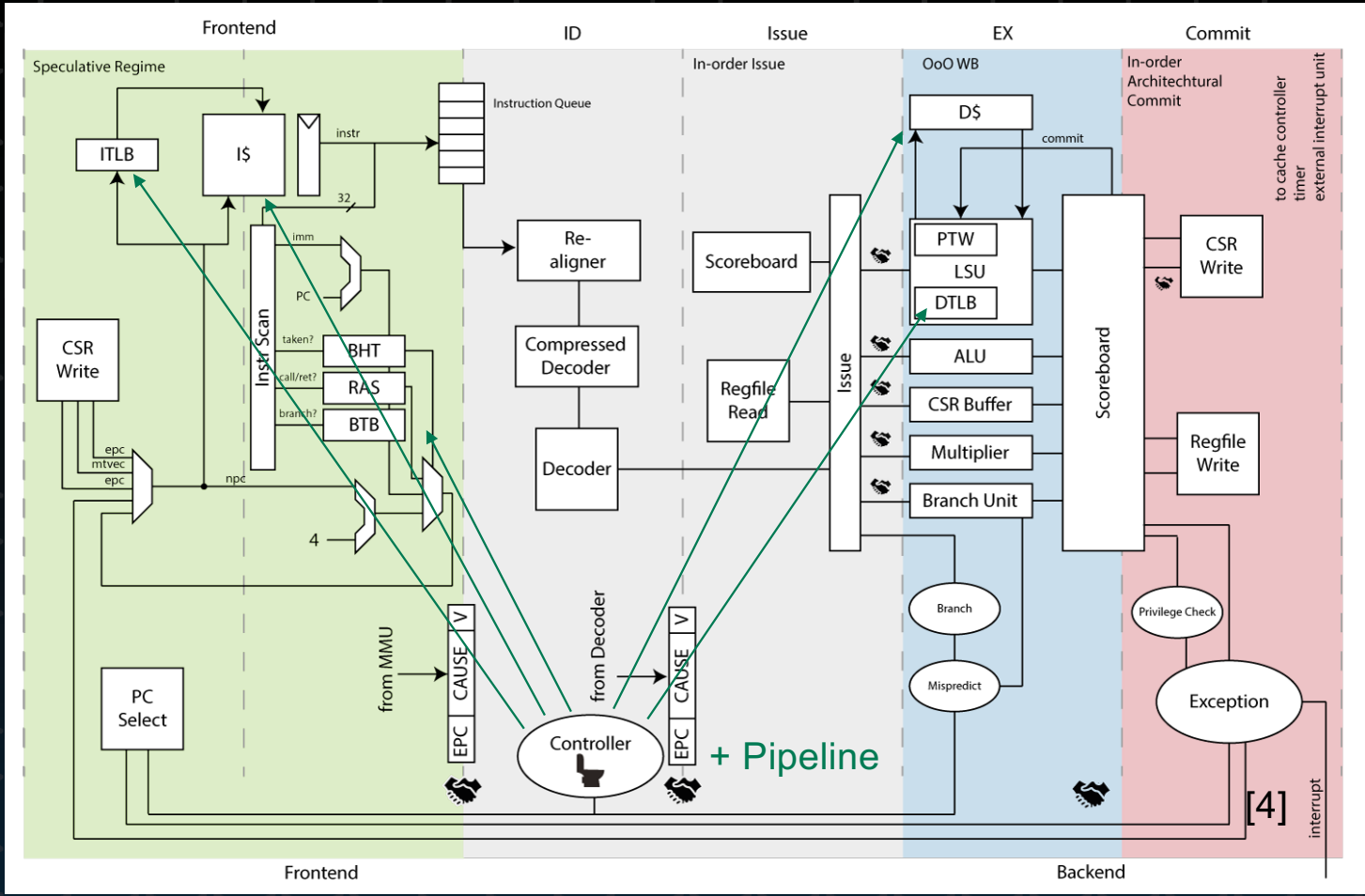
- Security-oriented OS microkernel
- Correctness proofs (x64, Arm, RV64)
- Experimental time-protection support

Hardware platform

Ariane RV64GC core

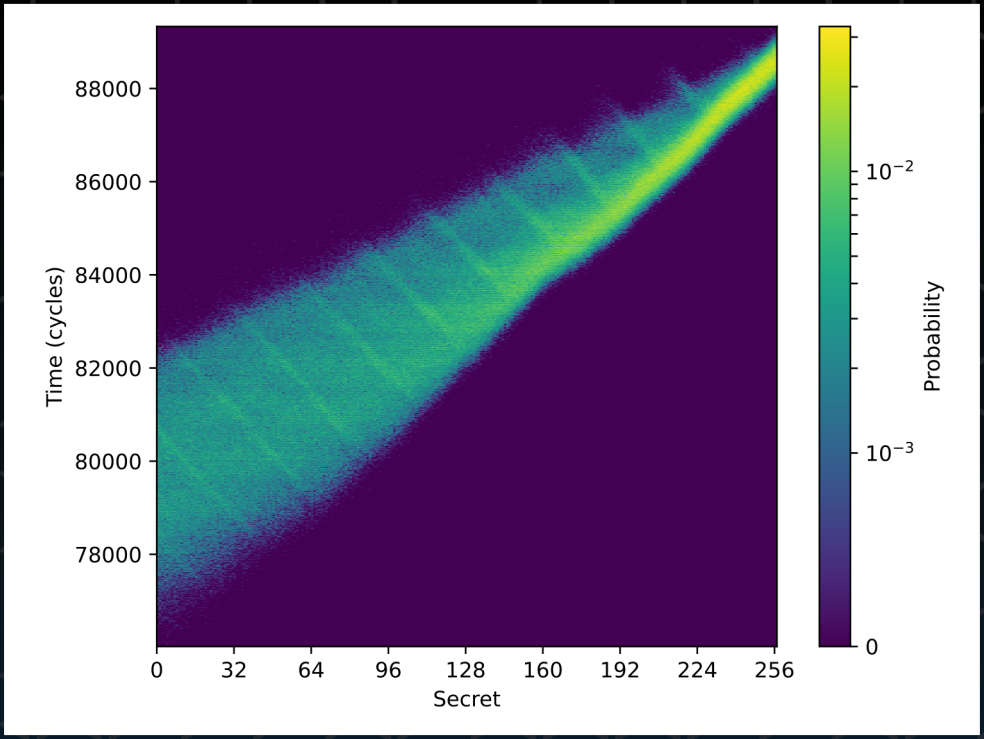
- FPGA (Genesys 2) @50MHz
- Add timer peripheral and 512KiB LLC [3]
- Write-through 32KiB L1D$ and 16KiB L1I$
- 16-entry DTLB, 16-entry BTB, 64-entry BHT

Temporal Fence Instruction: fence.t



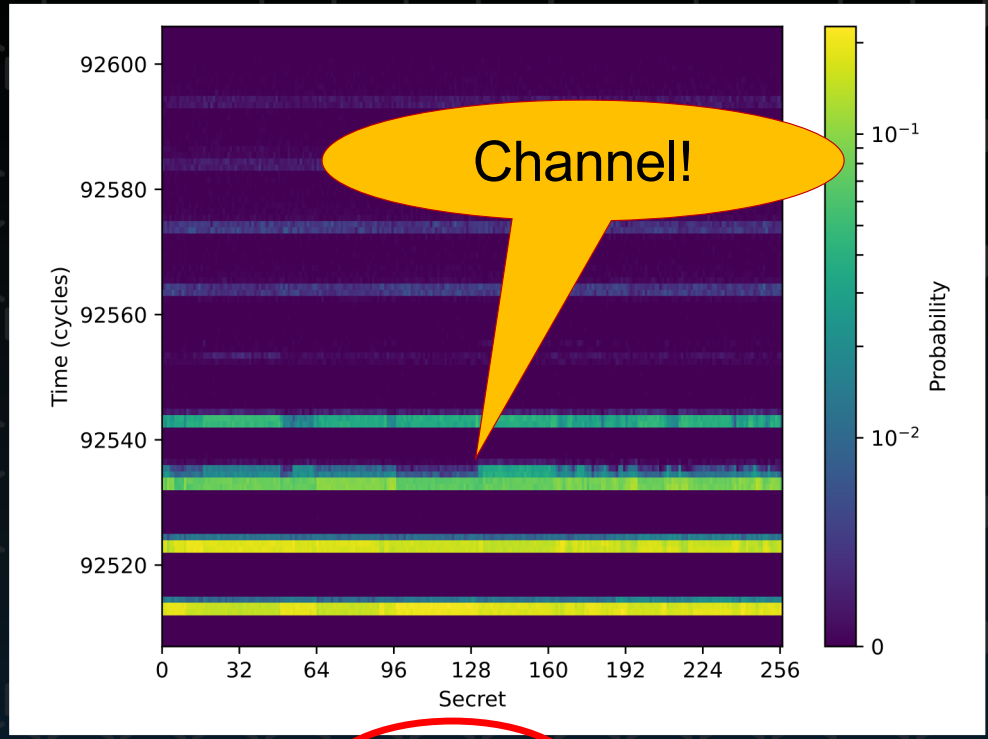
Data Cache

Unmitigated



$N = 10^6$, $M = 1667.3$ mb, $M_0 = 0.5$ mb

fence.t: Clear *valid* bits



$N = 10^6$, $M = 7.7$ mb, $M_0 = 1.4$ mb

Second-Order State Creates Channels Too

D-Cache

- First-order state:
  - valid bits
  - dirty bits, dirty lines (for WB cache)
- Second-order state:
  - pseudo-LRU state machine (LFSR)
  - memory arbiter
  - transmit FIFO
  - write-buffer arbiters

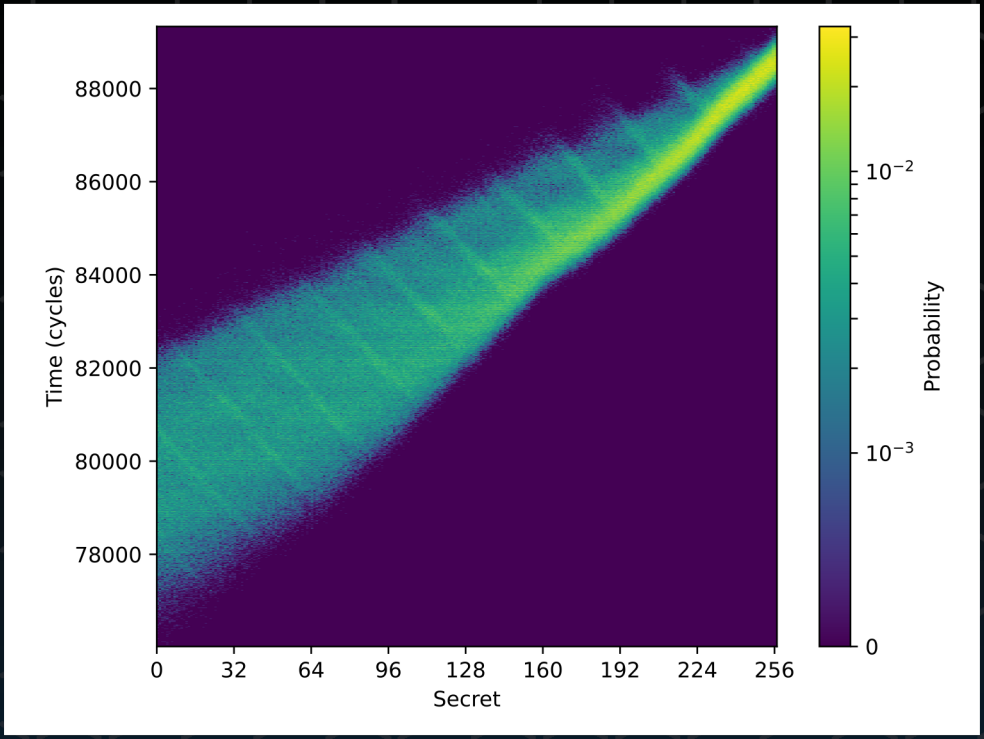
I-Cache, TLB

- First-order state:
  - valid bits
- Second-order state:
  - pseudo-LRU state machine

Any state that depends on execution history

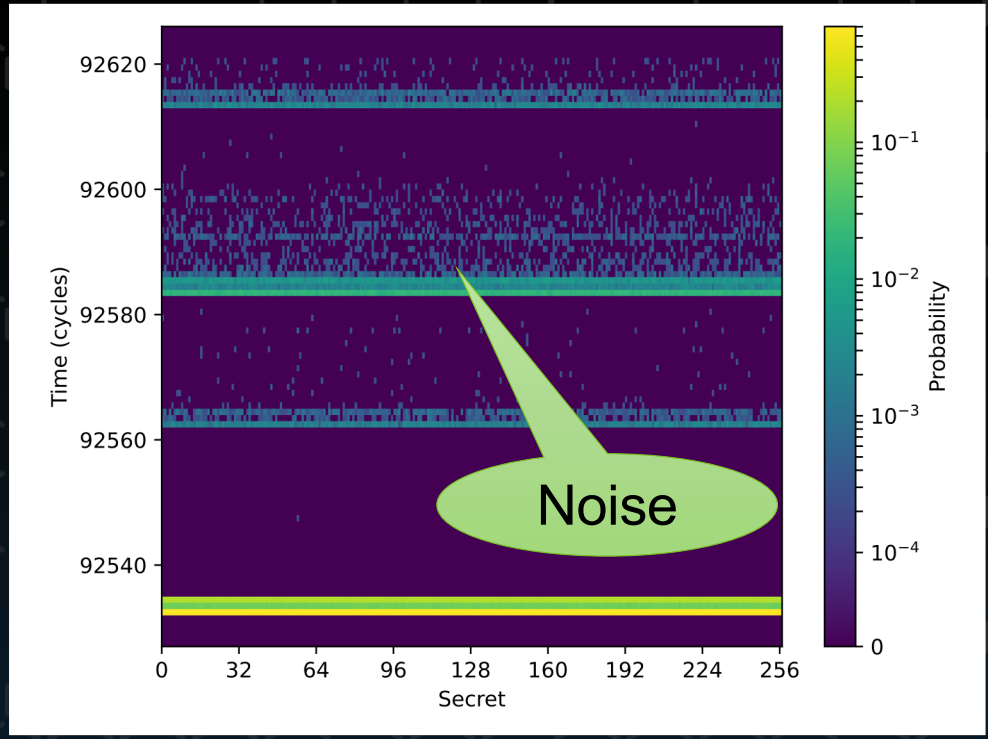
Data Cache

Unmitigated



$N = 10^6$, $M = 1667.3$ mb, $M_0 = 0.5$ mb

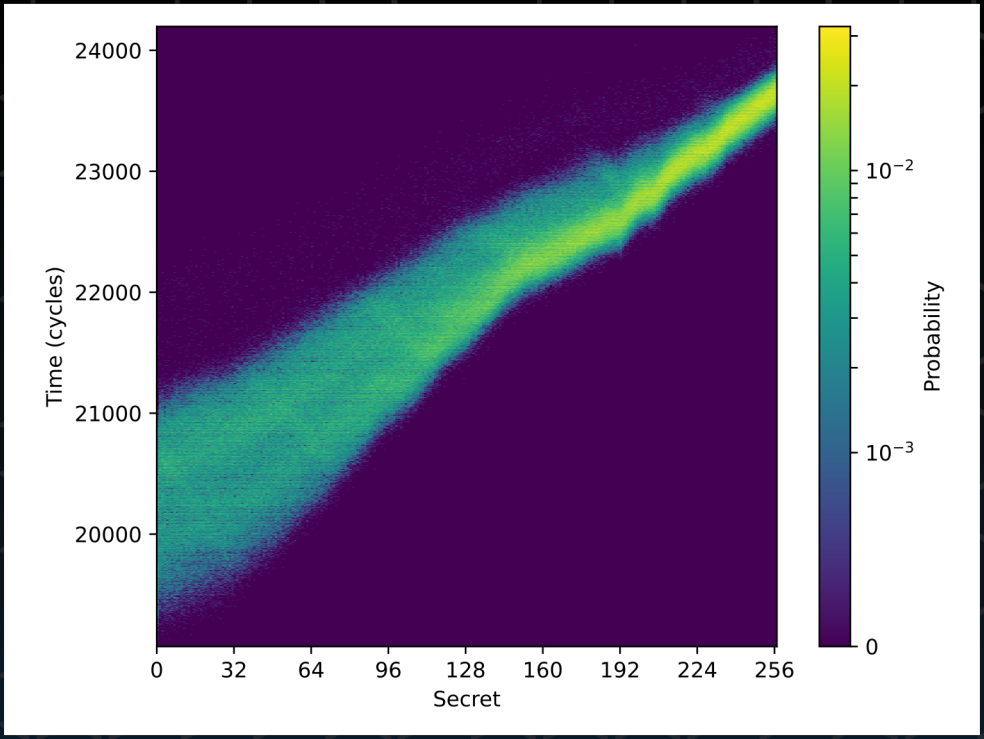
fence.t: Clear 1st & 2nd-order state



$N = 10^6$, $M = 8.4$ mb, $M_0 = 9.6$ mb

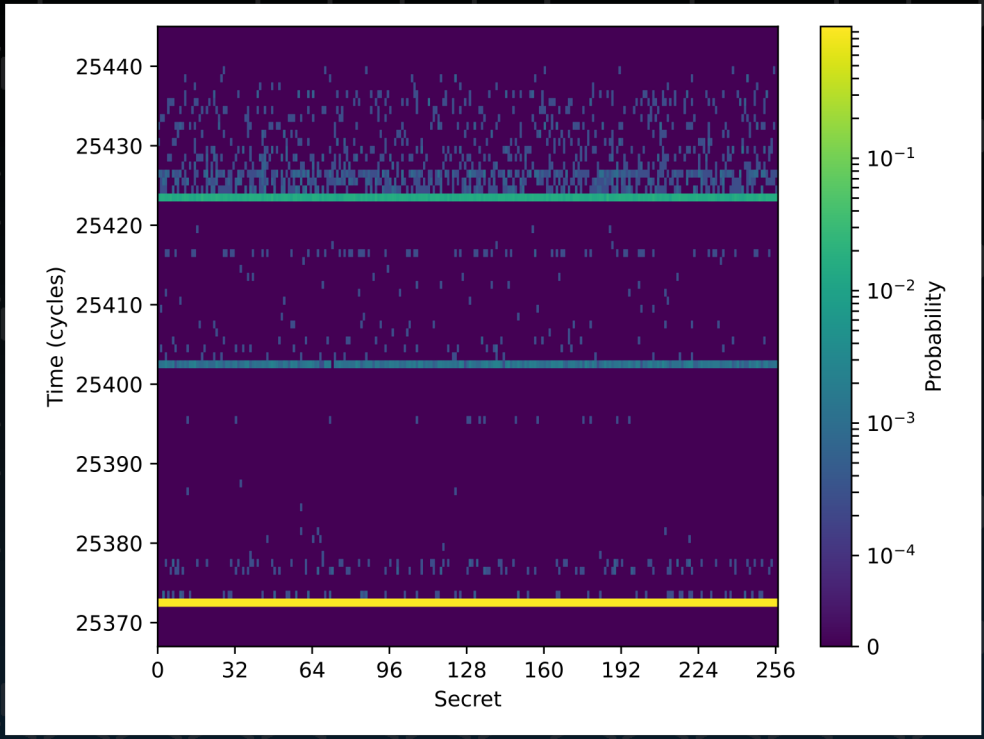
Instruction Cache

Unmitigated



$N = 10^6$, $M = 1905.0$ mb, $M_0 = 0.5$ mb

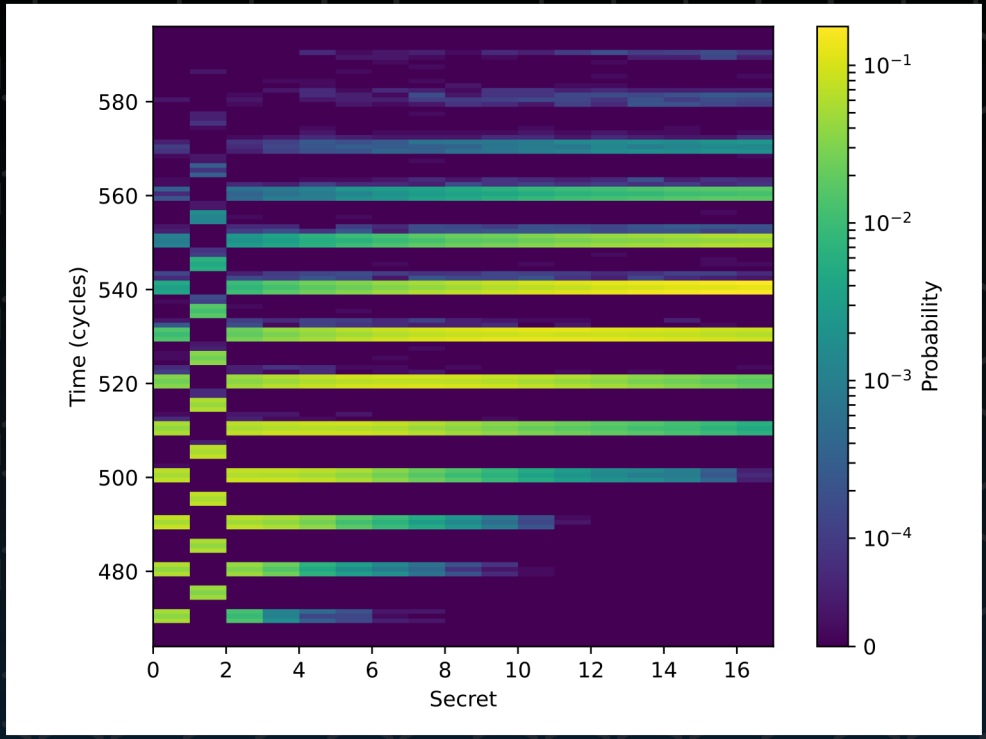
Full fence.t



$N = 10^6$, $M = 19.5$ mb, $M_0 = 20.5$ mb

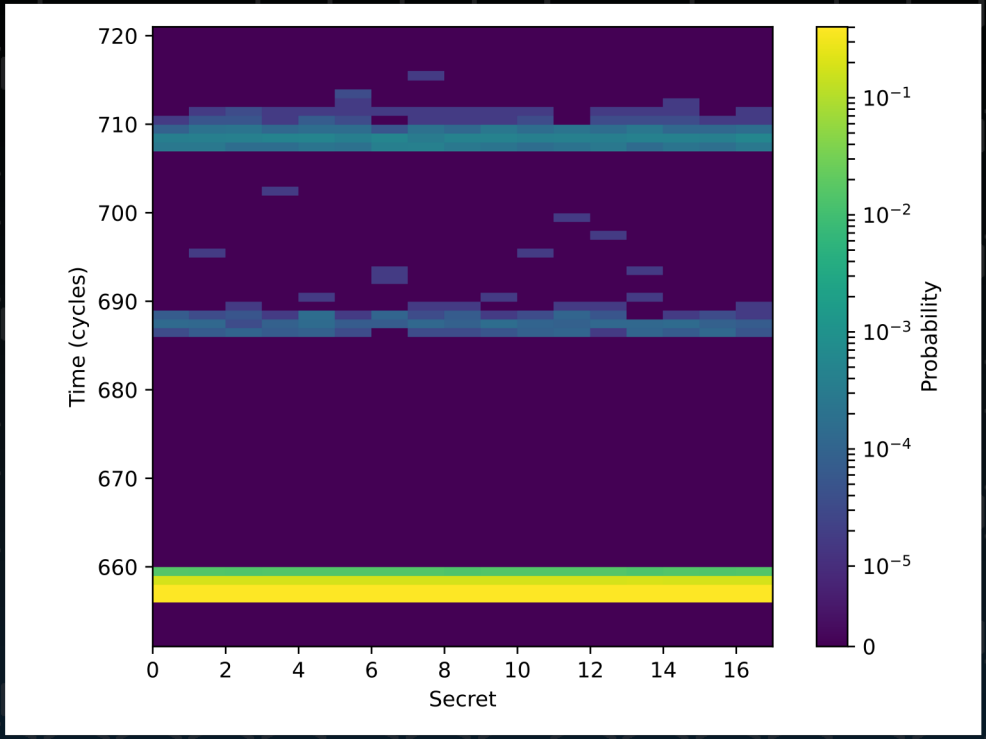
TLB

Unmitigated



$N = 10^6$, $M = 409.2$ mb, $M_0 = 0.1$ mb

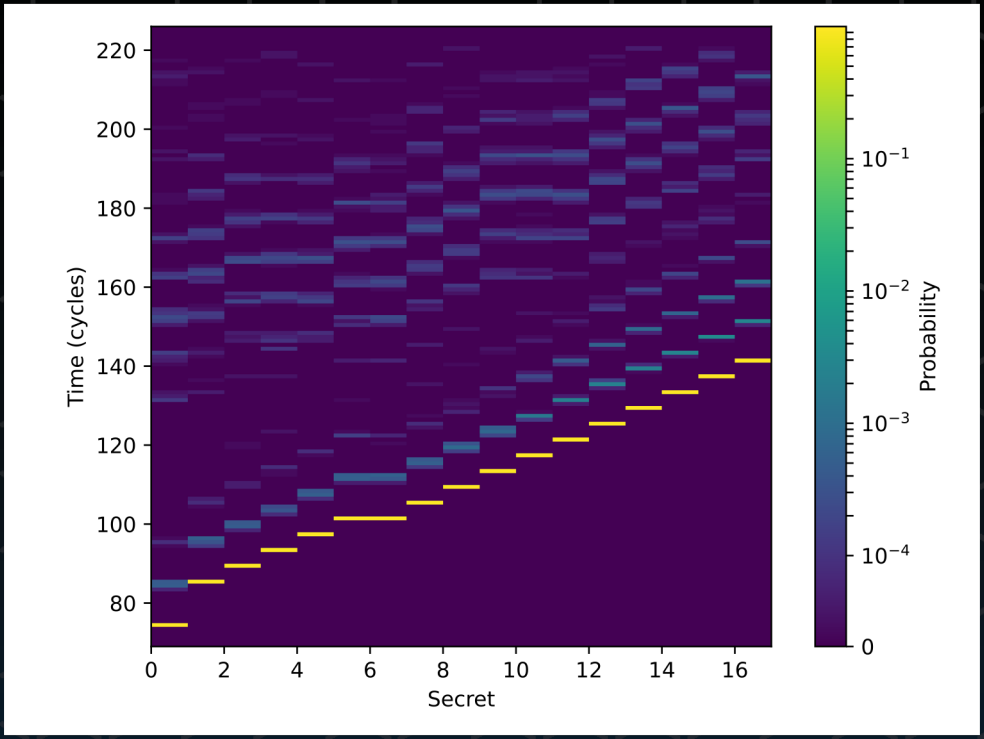
fence.t



$N = 10^6$, $M = 2.7$ mb, $M_0 = 5.4$ mb

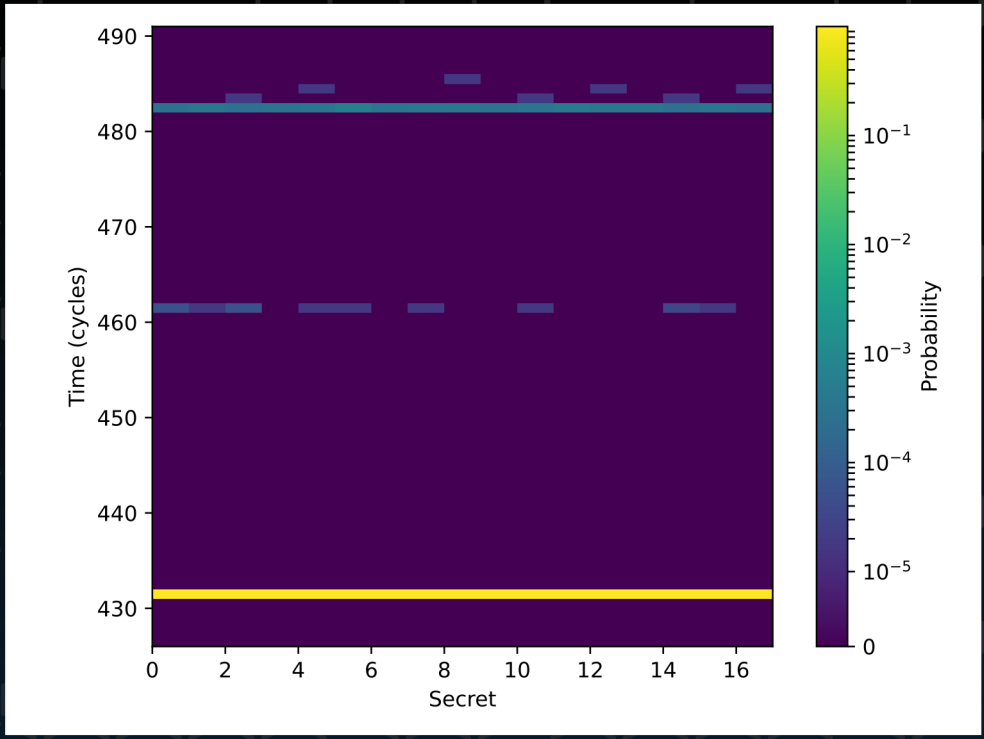
Branch Target Buffer

Unmitigated



$N = 10^6$, $M = 3481.3$ mb, $M_0 = 0.1$ mb

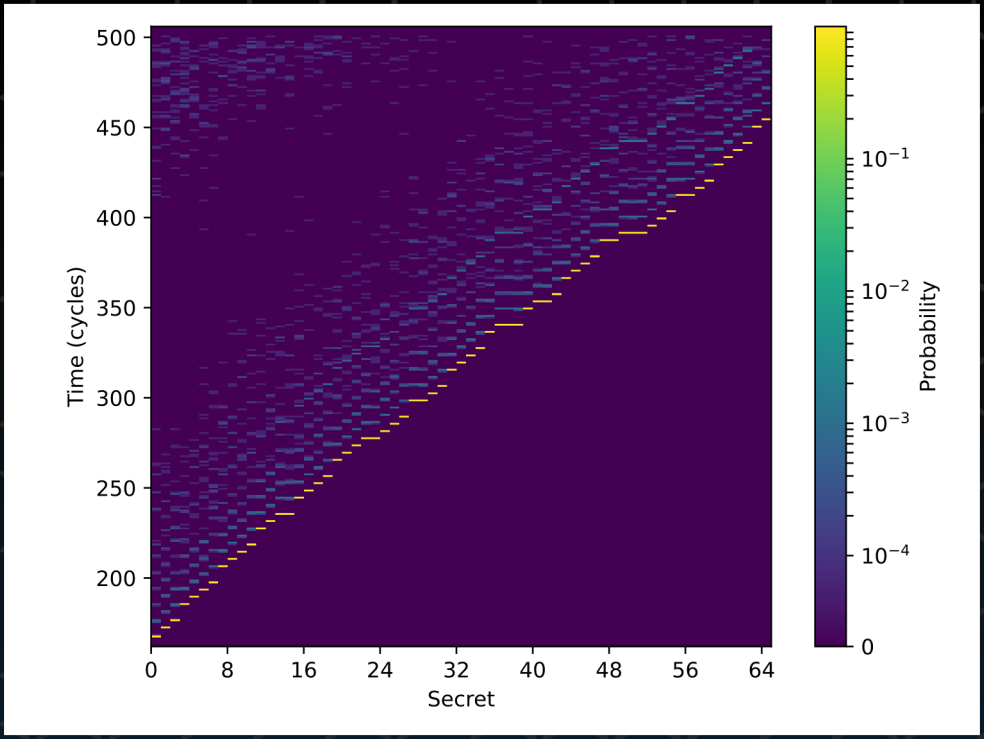
fence.t



$N = 10^6$, $M = 33.0$ mb, $M_0 = 57.6$ mb

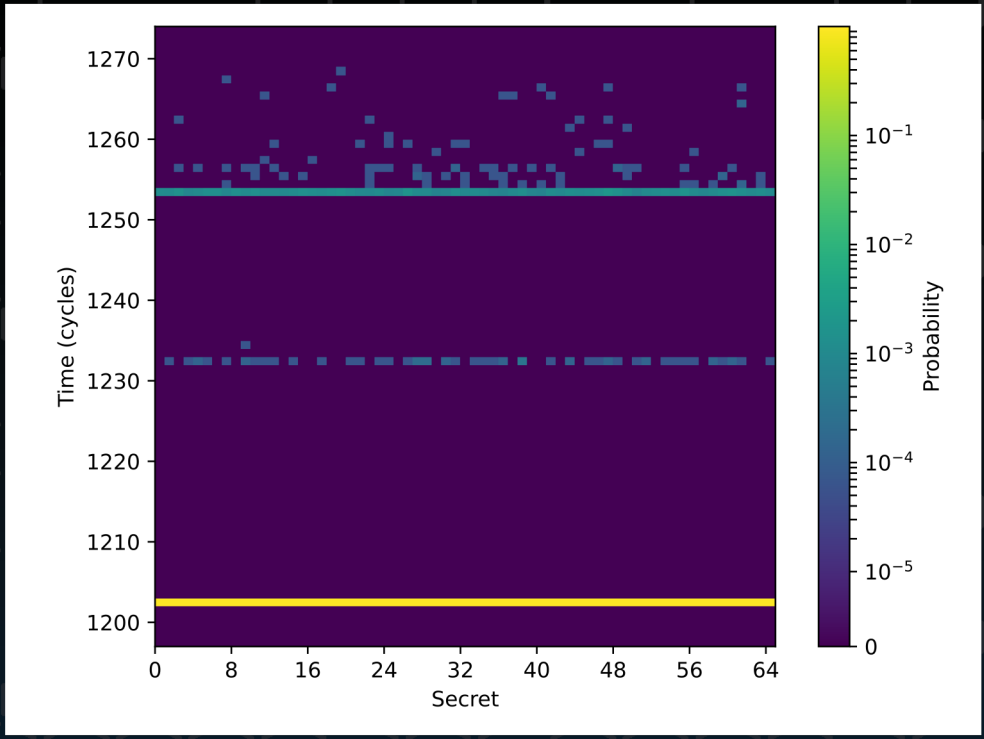
Branch History Table

Unmitigated



$N = 10^6$, $M = 4873.3$ mb, $M_0 = 0.1$ mb

fence.t



$N = 10^6$, $M = 44.1$ mb, $M_0 = 58.8$ mb

Overhead: Context-Switch Latency

Unmitigated		fence.t
Hot	Cold	
430±7	1,180±1	1,502±1

With WB cache add write-back latency

Overhead: Hardware (FPGA)

	LUTs	Registers	Muxes
Original	102,796±10	58,957±208	13,590±38
with fence.t	102,792±57	60,607±5	15,038±2
Increase	0%	2.8%	11%

Take-Aways: Flush Ops for Time Protection

Real question:
What's the right abstraction?

- Easy to implement
- Effective
  - but really need to clean *all* history-dependent state!
- Low increase in HW complexity
- Low latency
  - … but obviously somewhat higher cost with write-back cache
  - only pay when needed

Wistoff et al.: *Prevention of Microarchitectural Covert Channels on an Open-Source 64-bit RISC-V Core*, CARRV'20
https://carrv.github.io/2020/papers/CARRV2020_paper_10_Wistoff.pdf

THANK YOU

Gernot Heiser | gernot@unsw.edu.au | @GernotHeiser

- RISC-V SSC+PSC, June’20

https://trustworthy.systems