RISC-V® Summit

December 8-10 | Virtual Event

CONTRIBUTE.
COLLABORATE.
COMMERCIALIZE.

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT

December 8-10 | Virtual Event

Time Protection

Preventing Microarchitectural Timing Channels on RISC-V

Nils Wistoff
PhD Student
Integrated Systems Laboratory – ETH Zurich
Supervisors: Luca Benini, Gernot Heiser

#RISCVSUMMIT

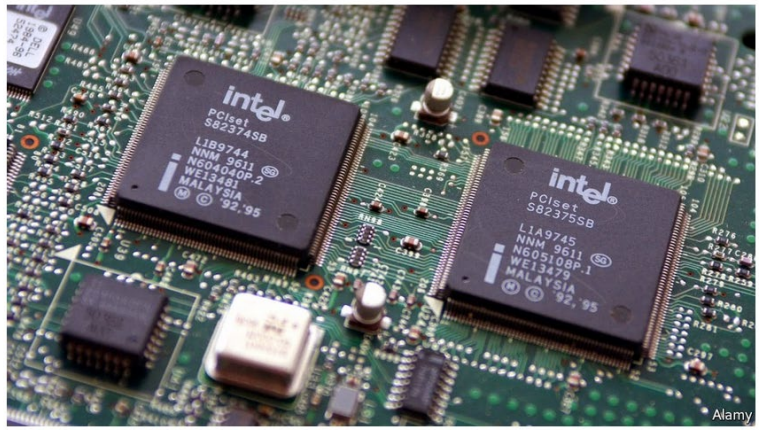
The Economist — Today — Weekly edition — Menu

Science & technology

The chips are down

Two security flaws in modern chips cause big headaches for the tech business

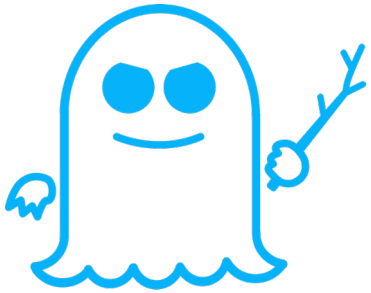
Fixing the underlying problems will take a long time



Jan 4th 2018

IT WAS a one-two punch for the computer industry. January 3rd saw the disclosure of two serious flaws in the design of the processors that power most of the world's computers. The first, appropriately called Meltdown, affects only chips made by Intel, and makes it possible to dissolve the virtual walls between the digital memory used by different programs, allowing hackers to steal sensitive data, such as passwords or a computer's encryption keys. The second,

[7]



SPECTRE

Speculative Execution + **Covert Channel**

ANDY GREENBERG SECURITY 01.03.2018 03:00 PM

A Critical Intel Flaw Breaks Basic Security for Most Computers

A Google-led team of researchers has found a critical chip flaw that developers are scrambling to patch in millions of computers.

[8]

make a contribution | subscribe | find a job | sign in | search | US edition — theguardian

news | opinion | sport | arts | lifestyle | more

us | world | environment | soccer | us politics | business | tech | science | homelessness

Data and computer security

Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers

Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors – and fix could slow devices

Spectre and Meltdown processor security flaws – explained

Samuel Gibbs

Thu 4 Jan '18 07.06 EST

[9]

THE VERGE — TECH | REVIEWS | SCIENCE | CREATORS | ENTERTAINMENT | VIDEO | MORE

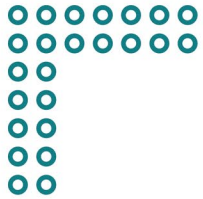
MICROSOFT | TECH | INTEL

Intel's processors have a security bug and the fix could slow down PCs

116

By Tom Warren | @tomwarren | Jan 3, 2018, 8:45am EST

[10]



PULP
Parallel Ultra Low Power

RISC-V®
Summit

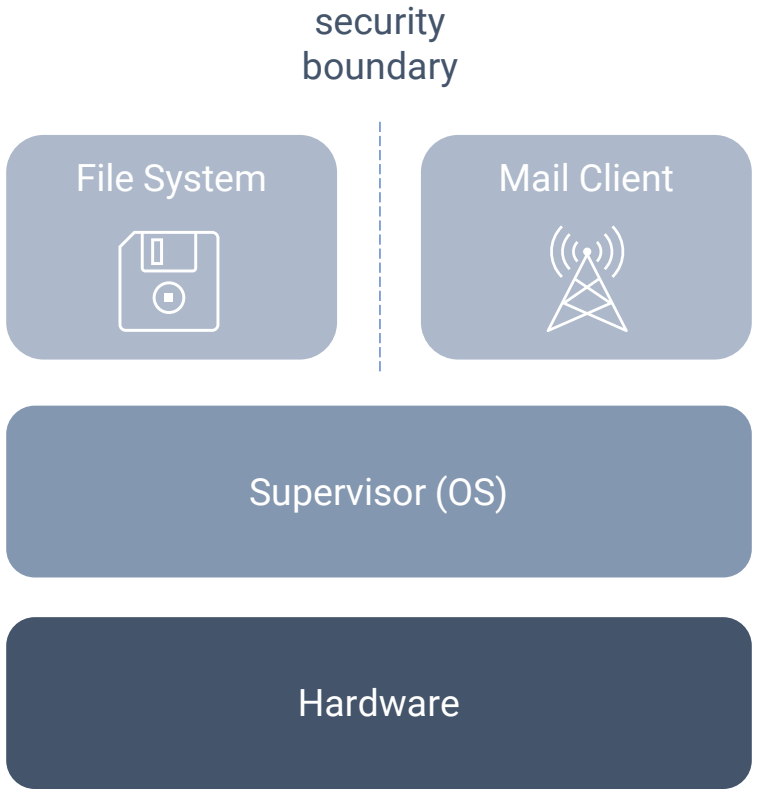
ETH zürich

HENSOLDT
Detect and Protect.

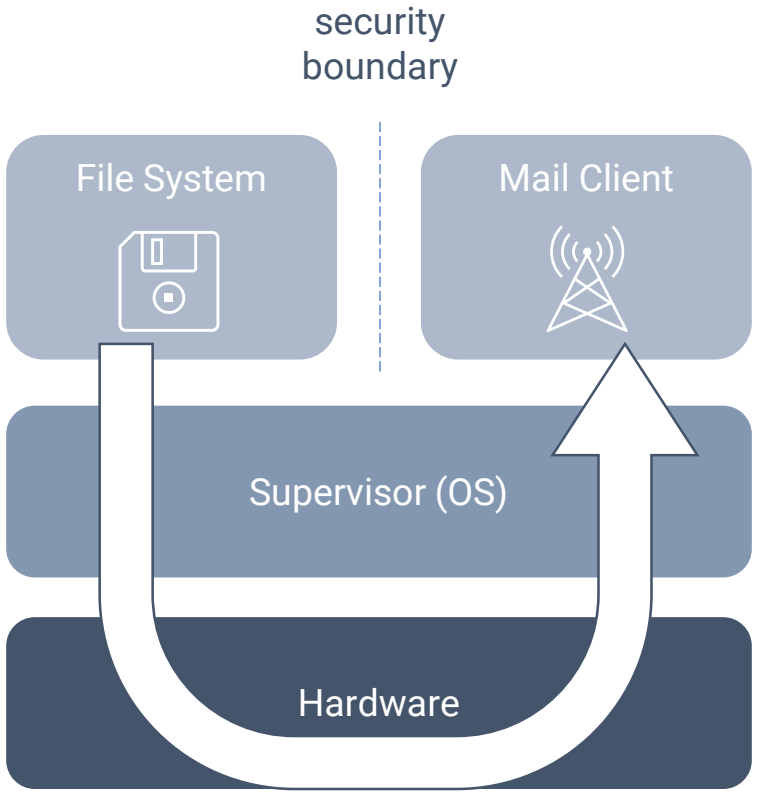
RWTH AACHEN
UNIVERSITY

CSIRO DATA 61

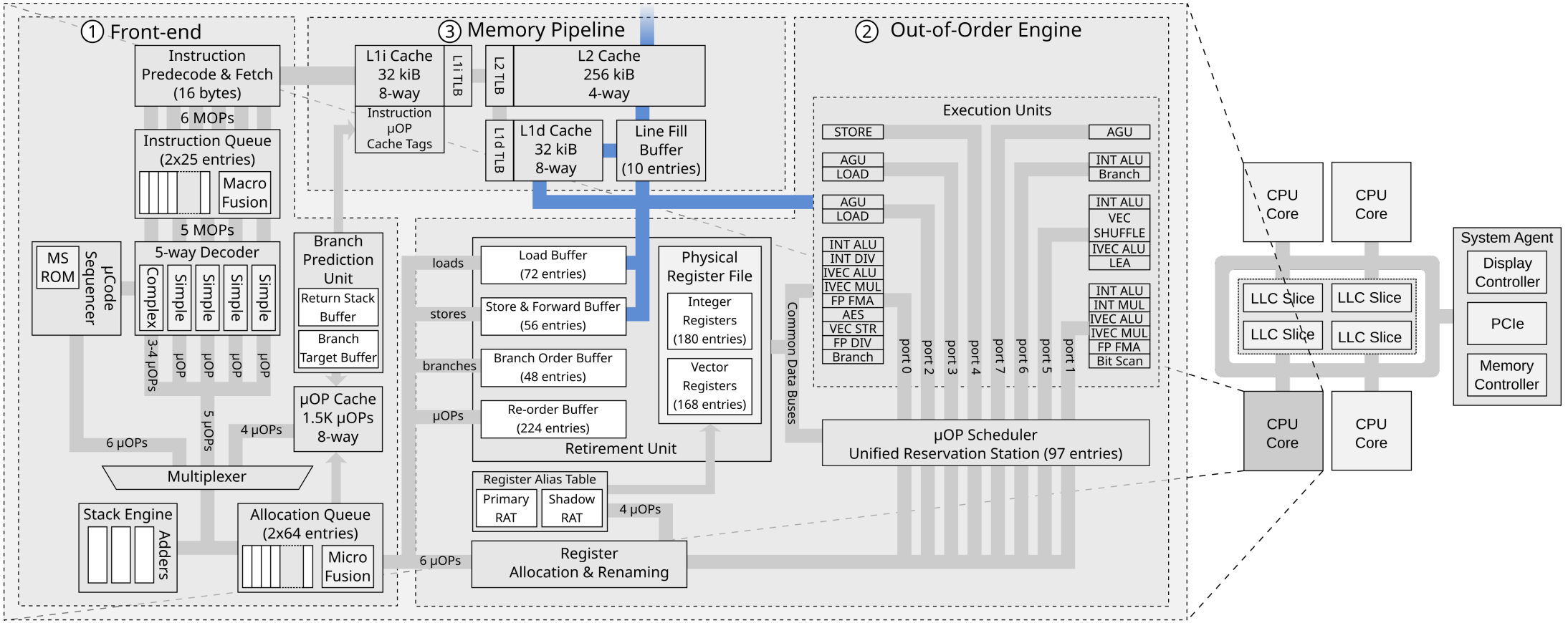
Covert Channel



Covert Channel

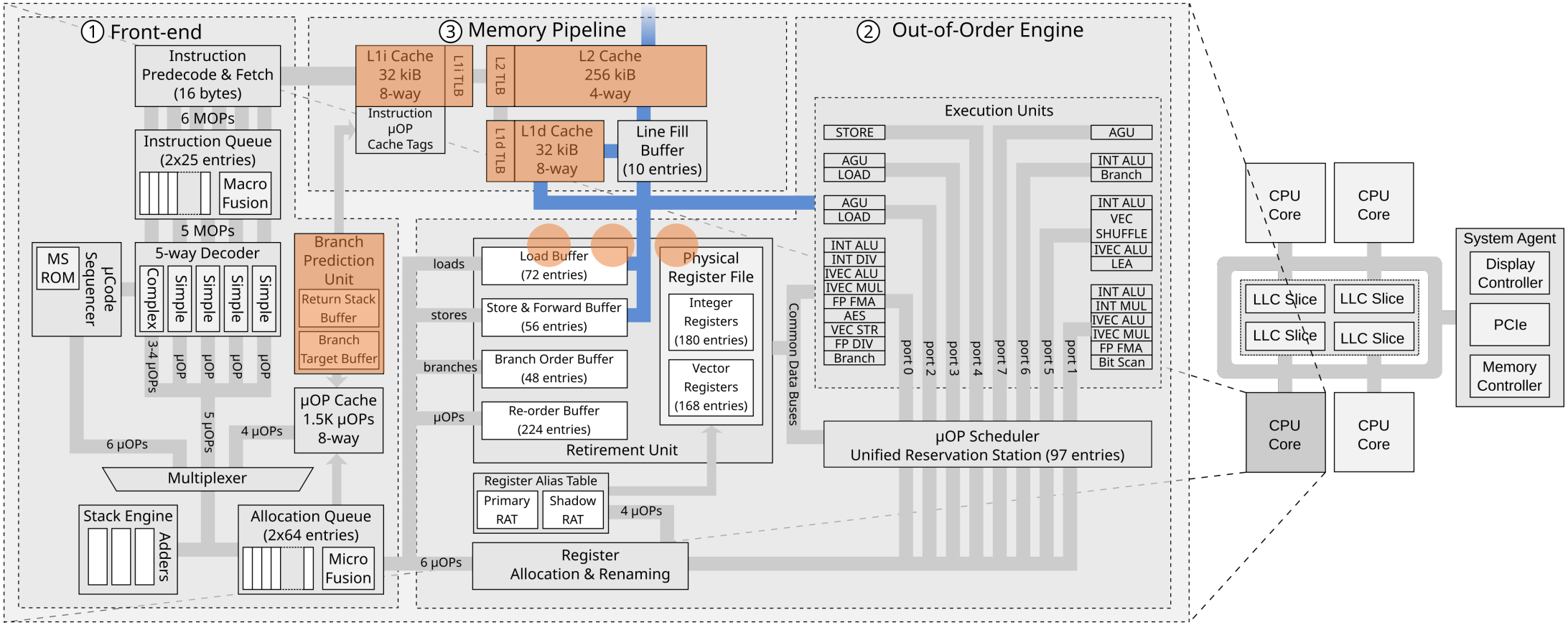


Intel Skylake Microarchitecture



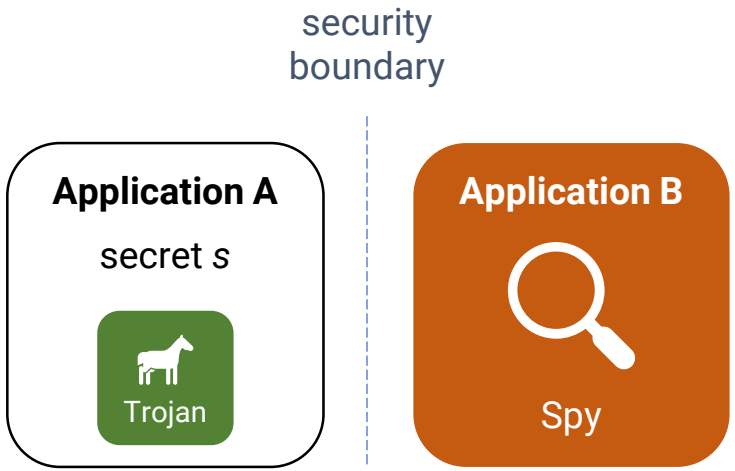
[6]

Intel Skylake Microarchitecture

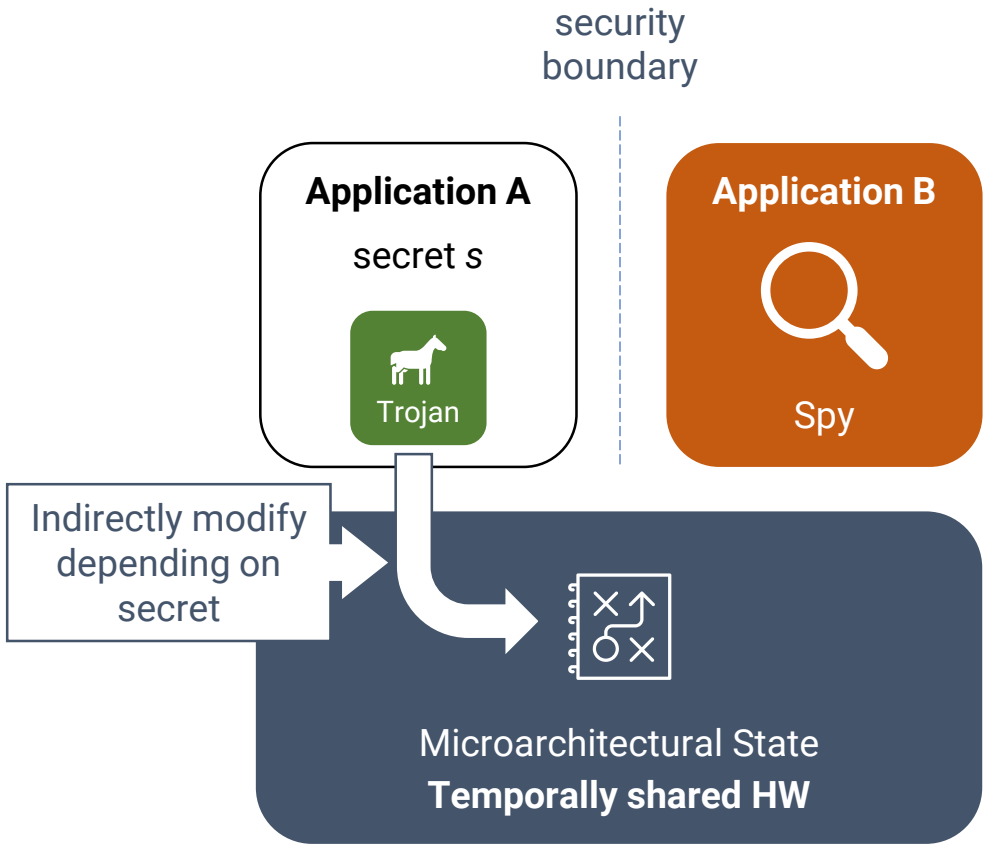


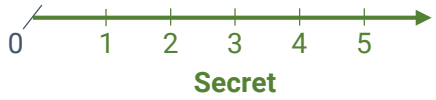
[6]

Microarchitectural Timing Channel

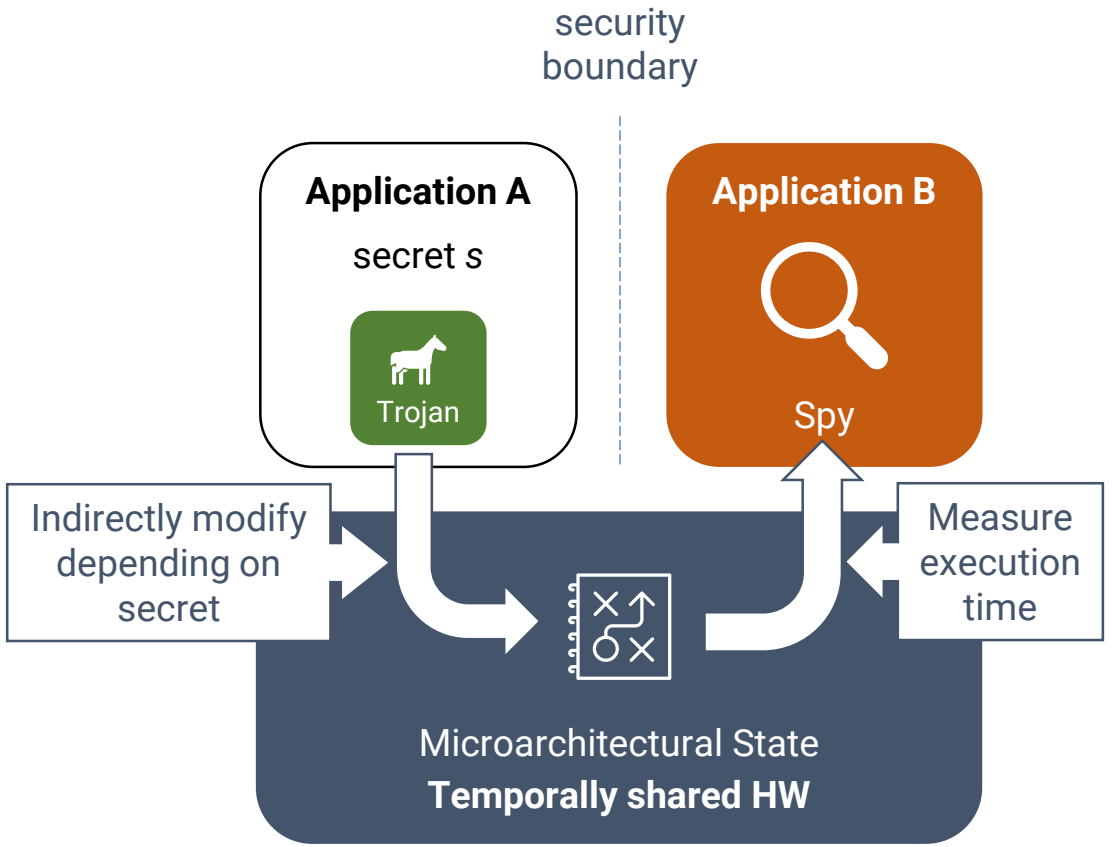


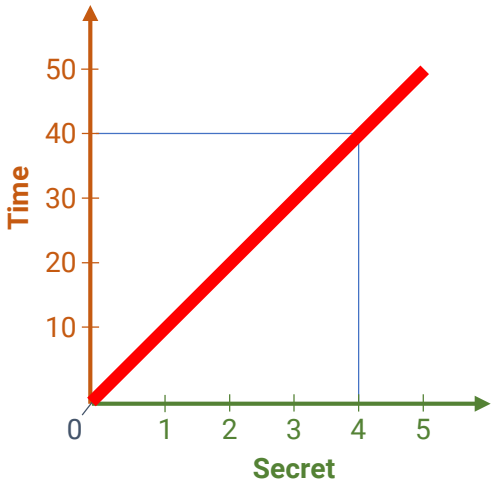
Microarchitectural Timing Channel



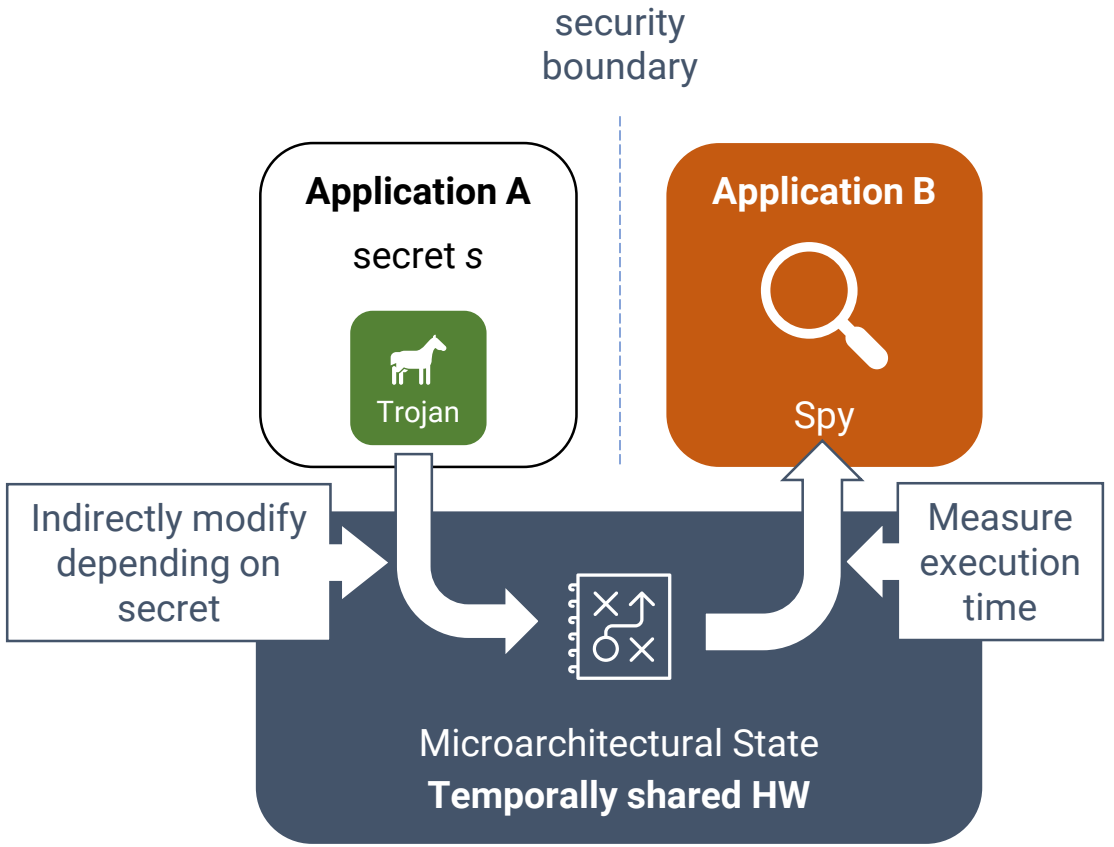


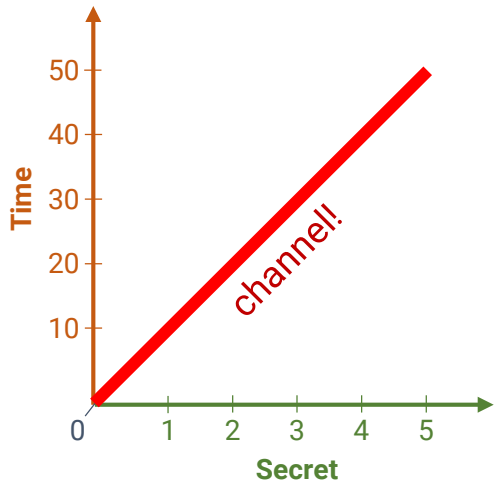
Microarchitectural Timing Channel



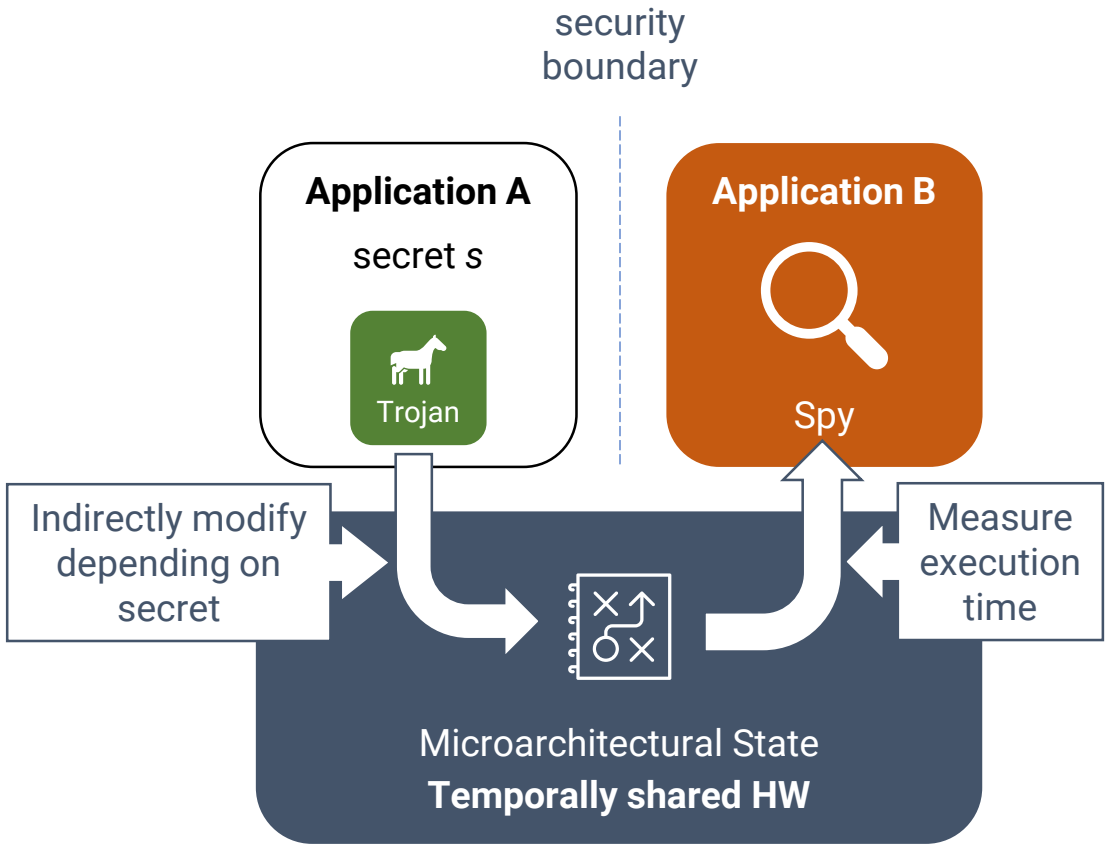


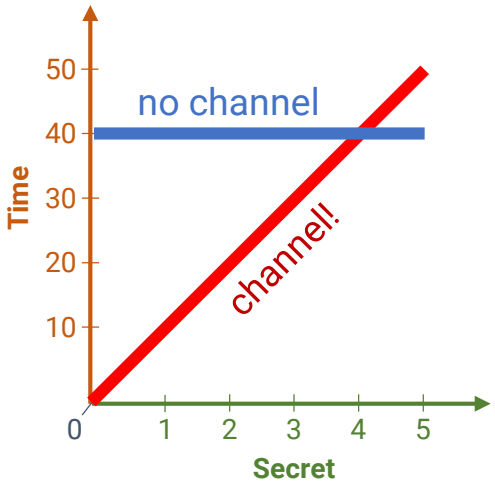
Microarchitectural Timing Channel



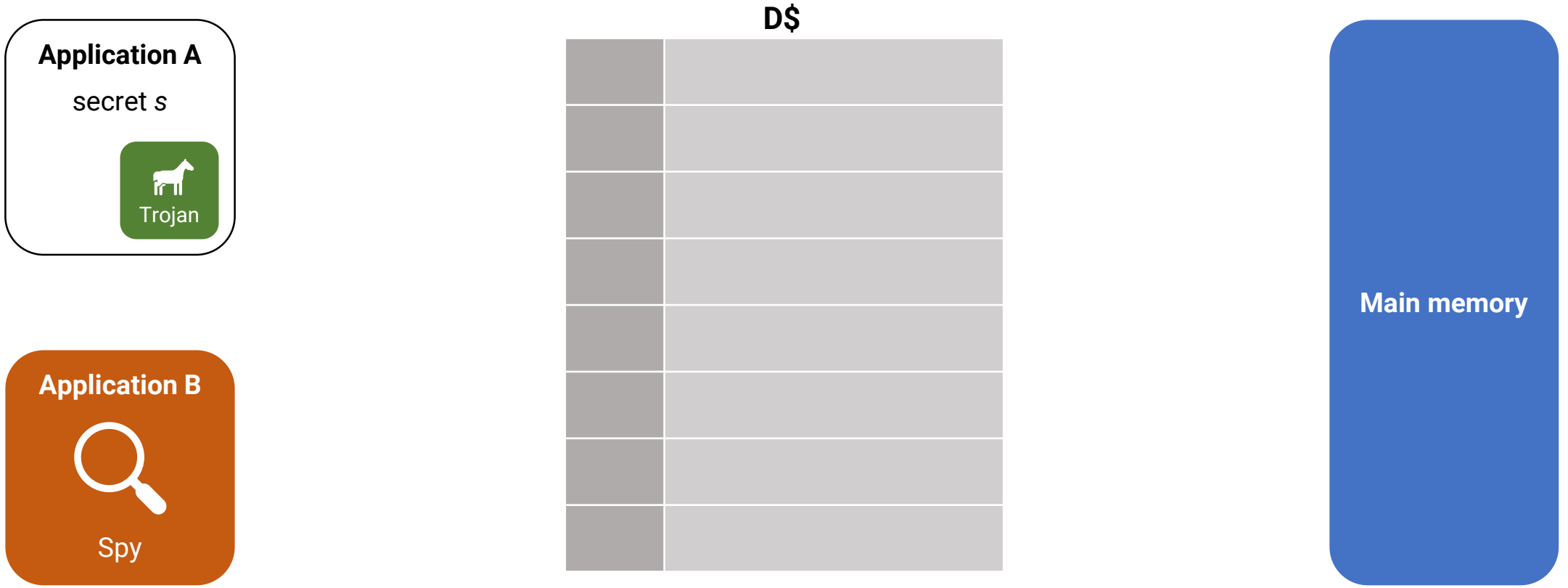


Microarchitectural Timing Channel





Example: D$ Timing Channel



(1) Spy: **Prime**

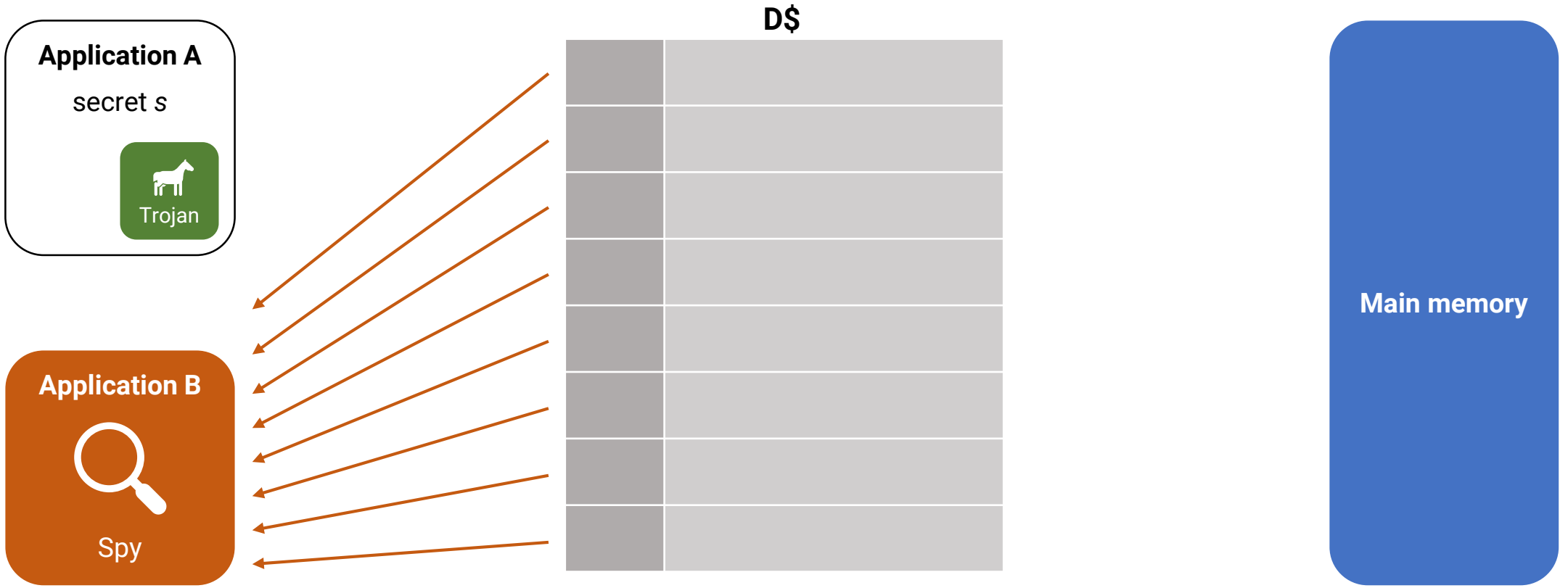
(2) OS: **Cont. sw.**

(3) Trojan: **Encode *s***

(4) OS: **Cont. sw.**

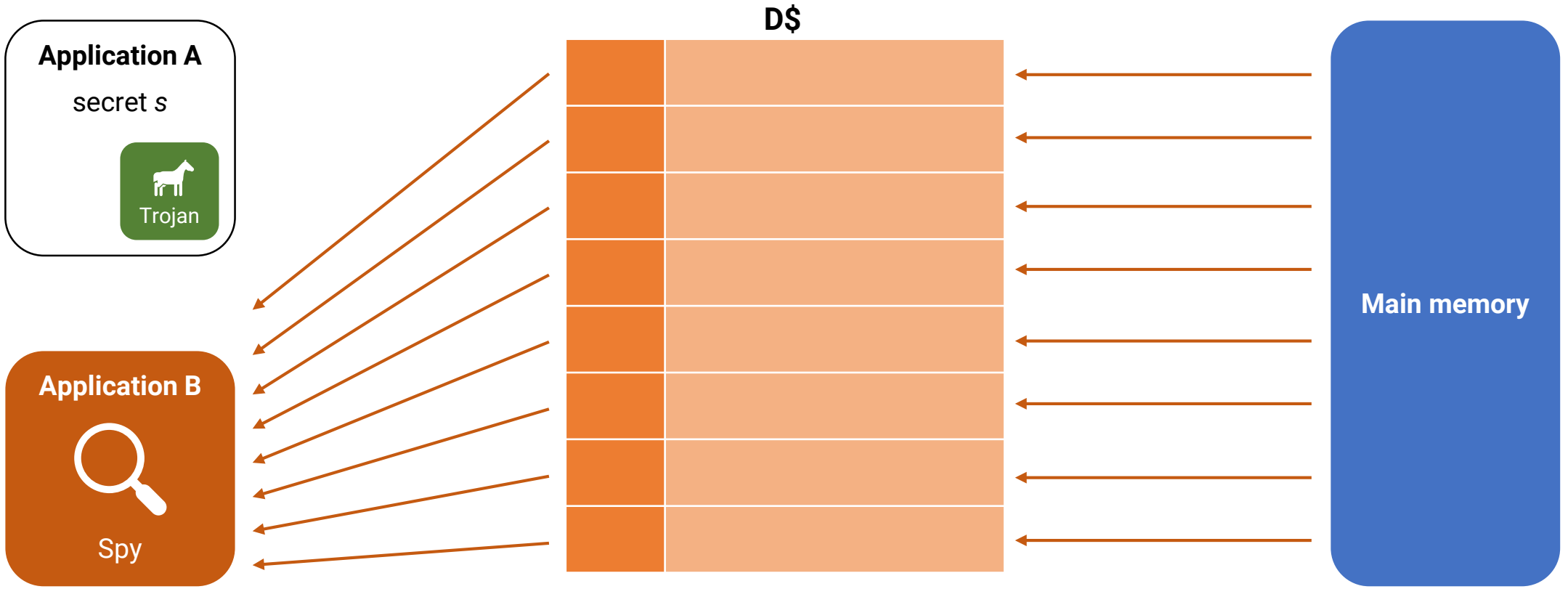
(5) Spy: **Probe**

Example: D$ Timing Channel - Prime



(1) Spy: **Prime** (2) OS: **Cont. sw.** (3) Trojan: **Encode *s*** (4) OS: **Cont. sw.** (5) Spy: **Probe**

Example: D$ Timing Channel - Prime



(1) Spy: **Prime**

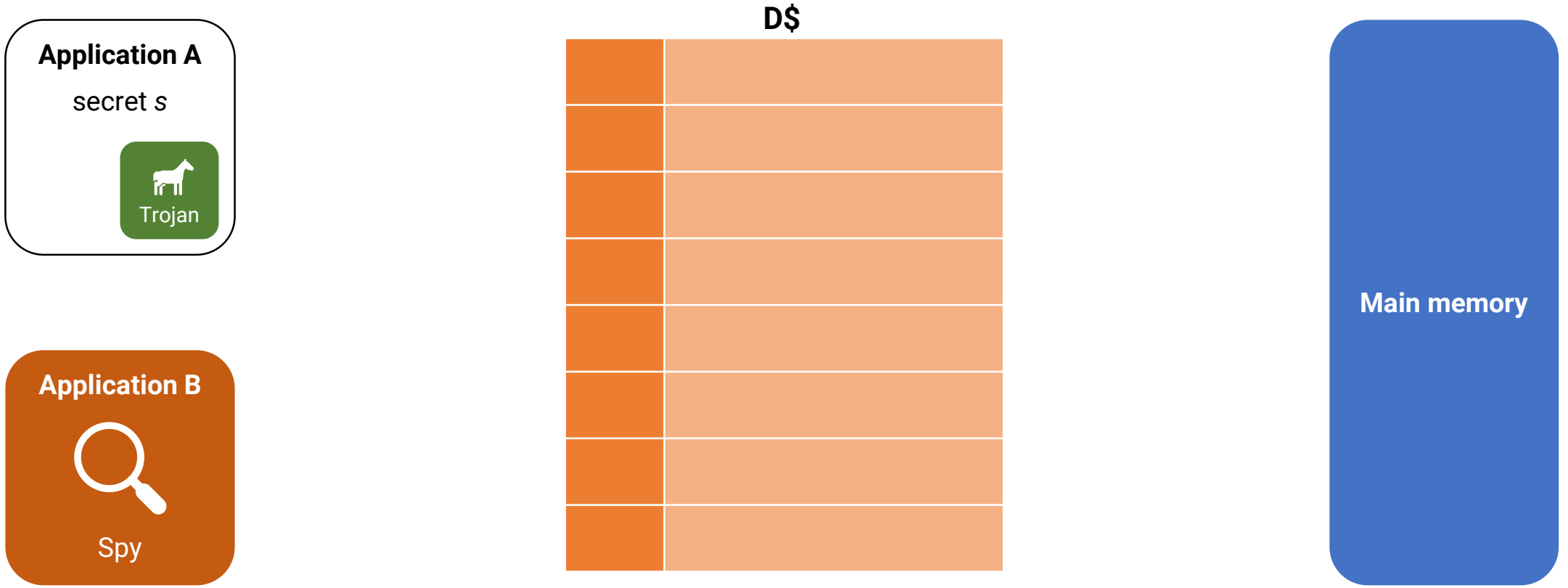
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Example: D$ Timing Channel – Context Switch



(1) Spy: **Prime**

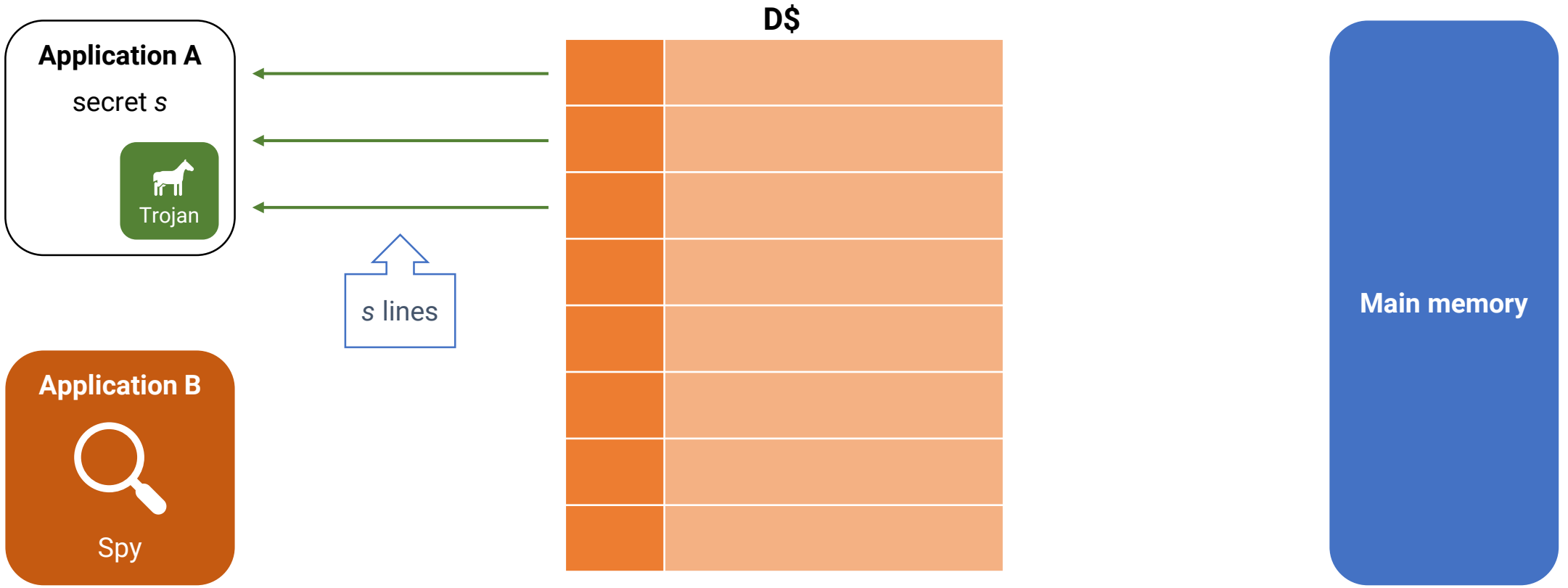
(2) OS: **Cont. sw.**

(3) Trojan: **Encode *s***

(4) OS: **Cont. sw.**

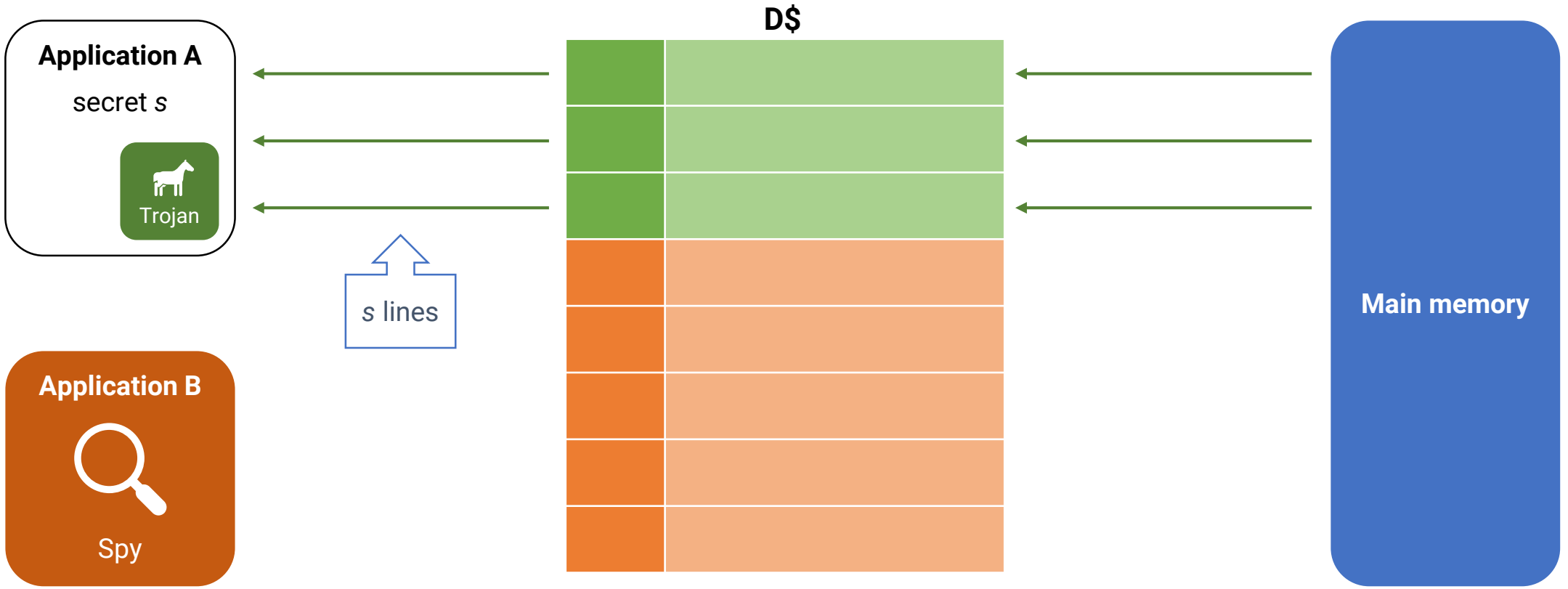
(5) Spy: **Probe**

Example: D$ Timing Channel – Encode *s*



(1) Spy: **Prime** (2) OS: **Cont. sw.** (3) Trojan: **Encode s** (4) OS: **Cont. sw.** (5) Spy: **Probe**

Example: D$ Timing Channel – Encode *s*



(1) Spy: **Prime**

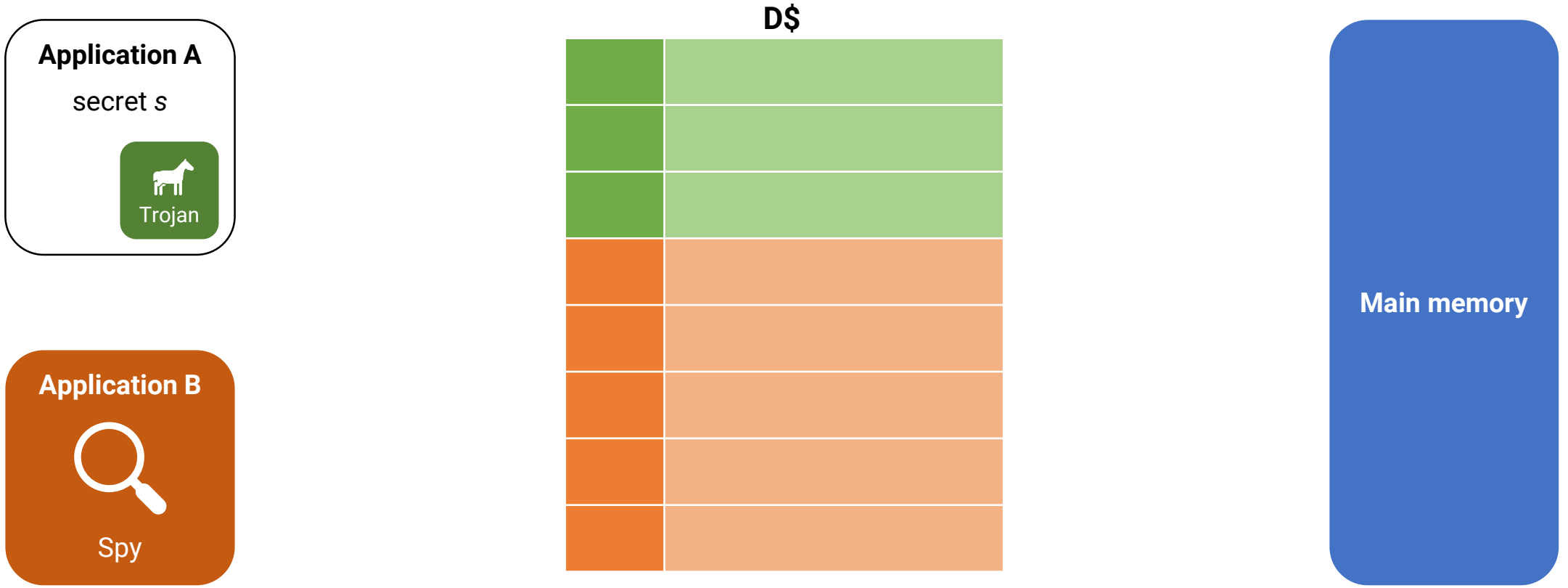
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Example: D$ Timing Channel – Context Switch



(1) Spy: **Prime**

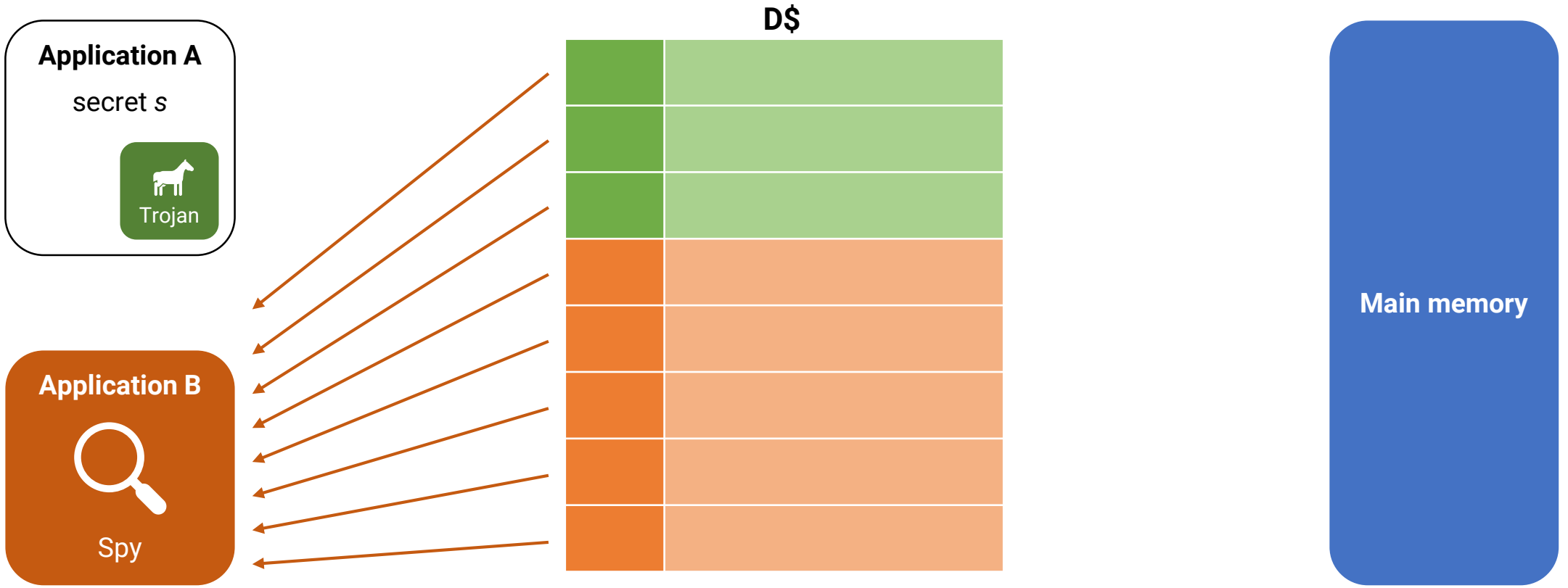
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Example: D$ Timing Channel - Probe



(1) Spy: **Prime**

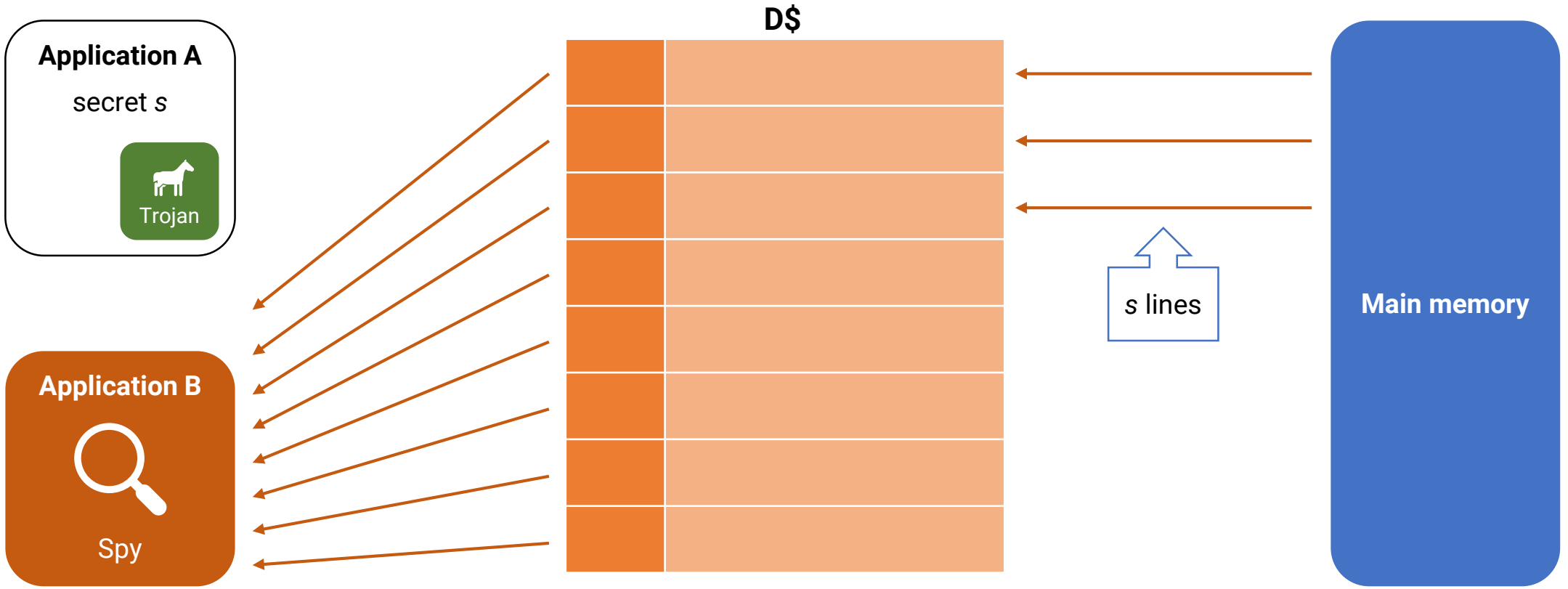
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Example: D$ Timing Channel - Probe



(1) Spy: **Prime**

(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

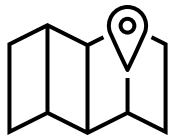
(4) OS: **Cont. sw.**

(5) Spy: **Probe**

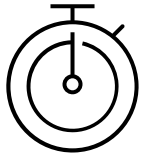
So what to Do?

Partition all shared resources!

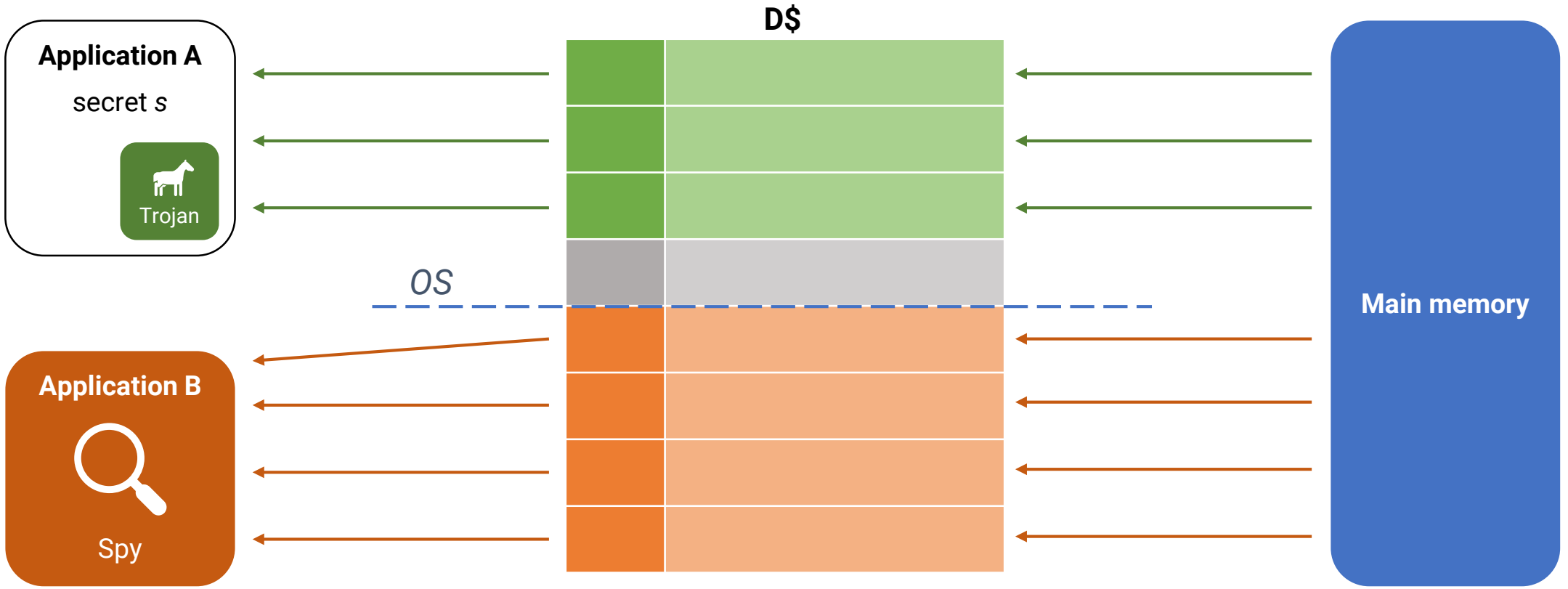
Spatially



Temporally



Spatial Partitioning



(1) Spy: **Prime**

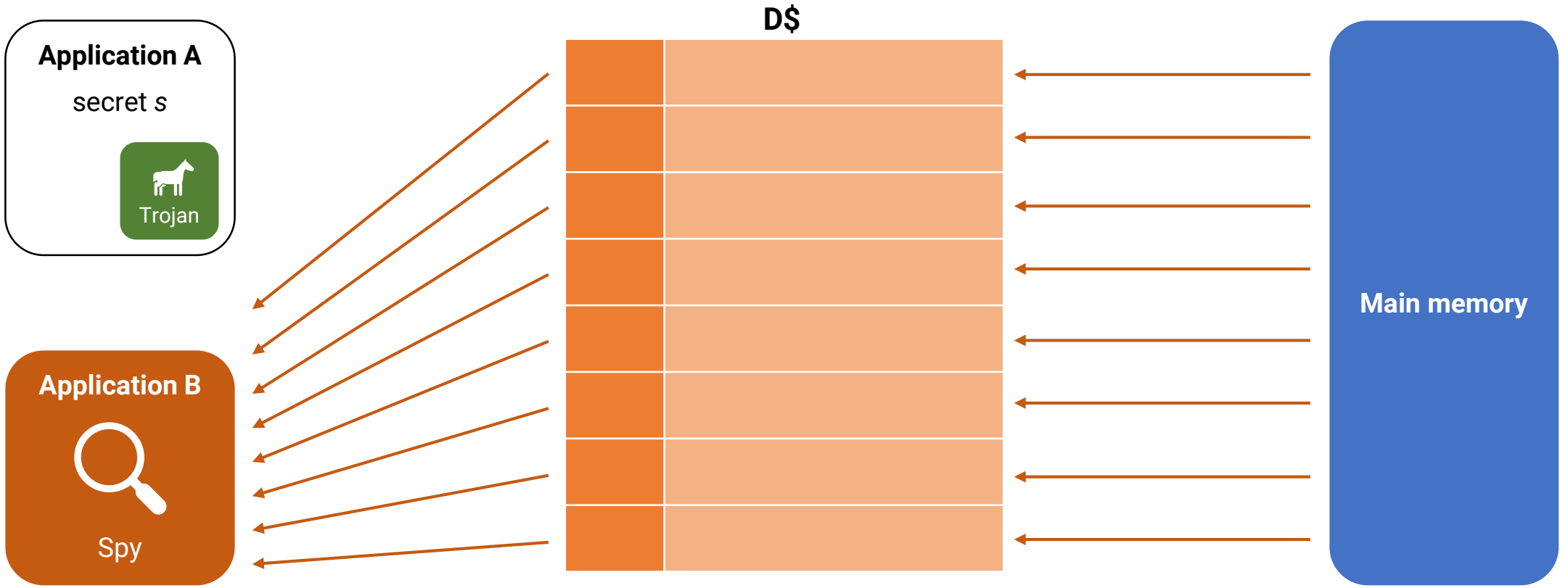
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime**

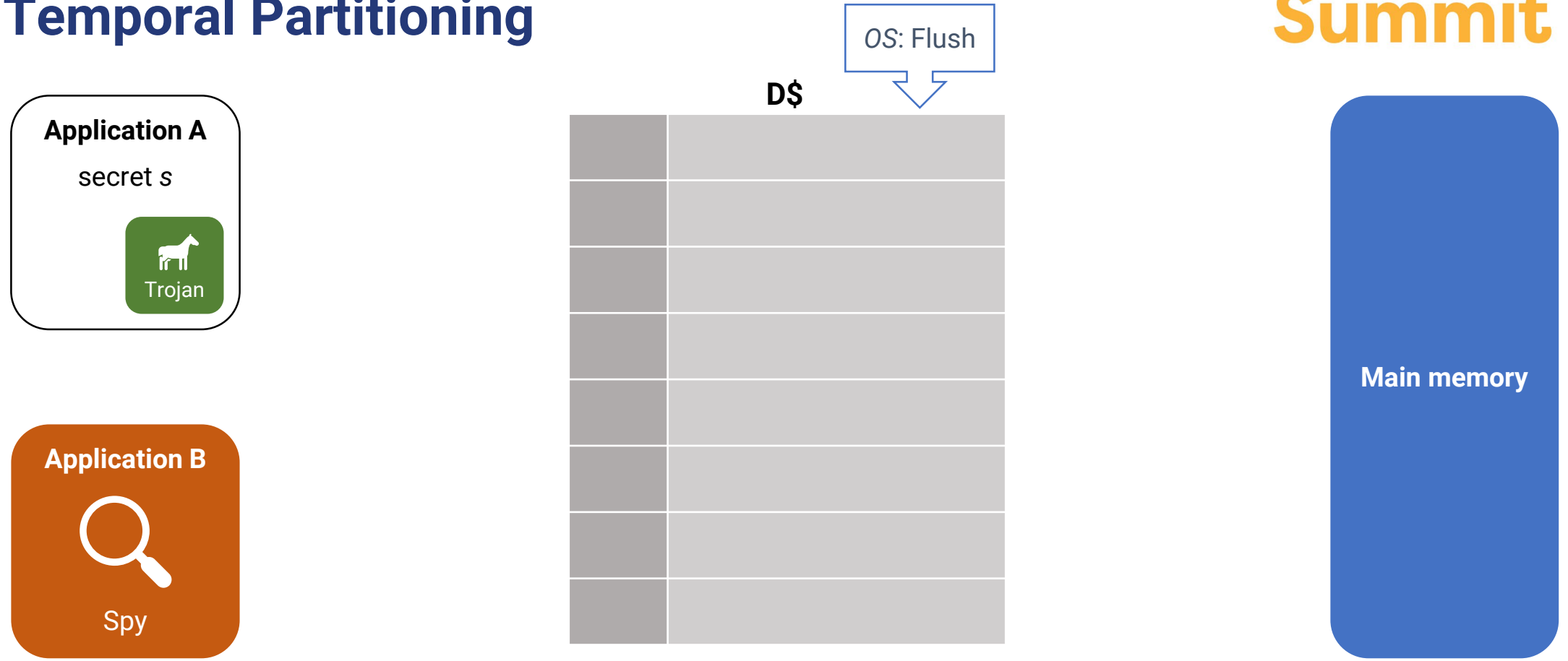
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime**

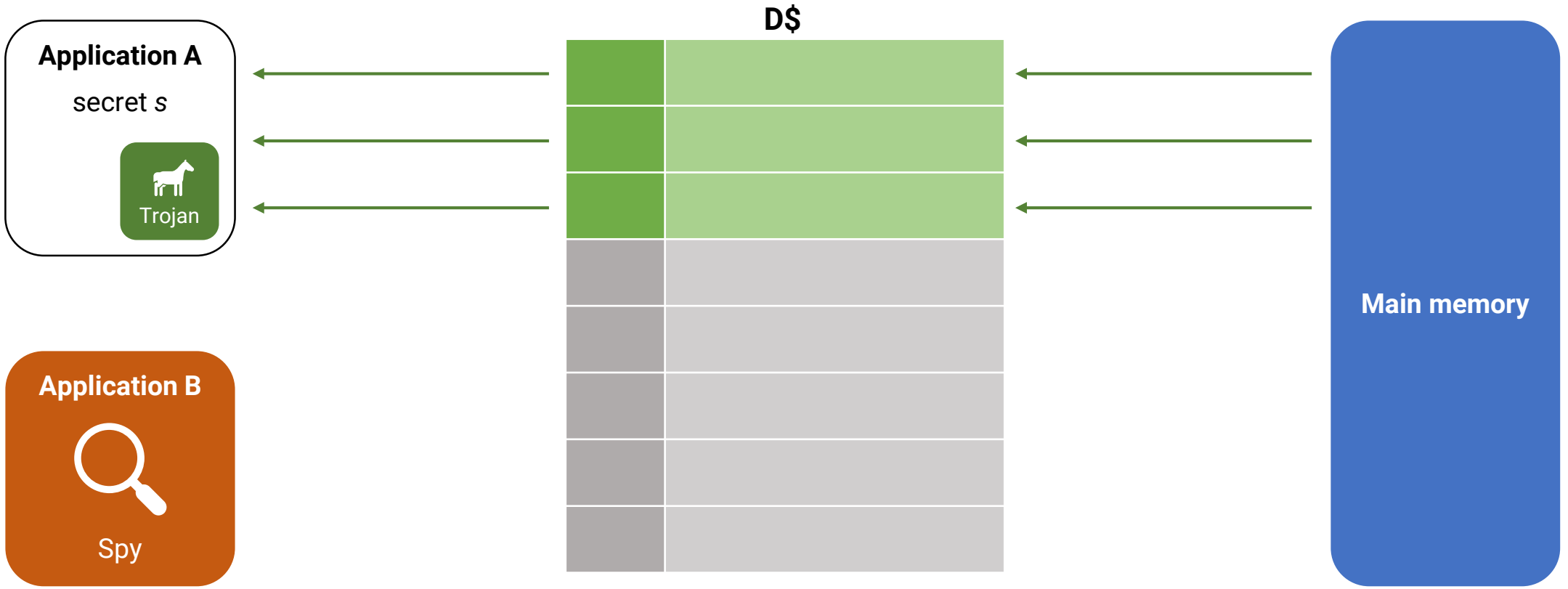
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

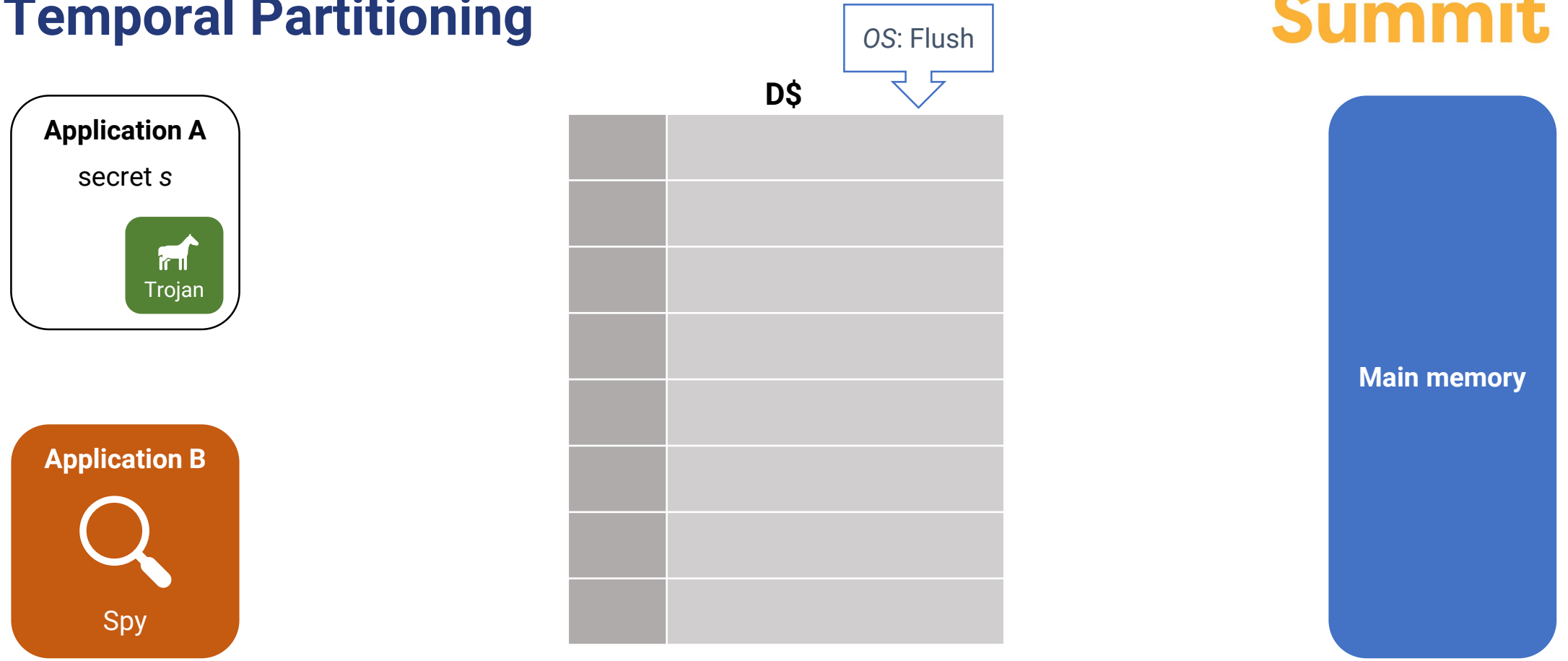
(5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime** (2) OS: **Cont. sw.** (3) Trojan: **Encode s** (4) OS: **Cont. sw.** (5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime**

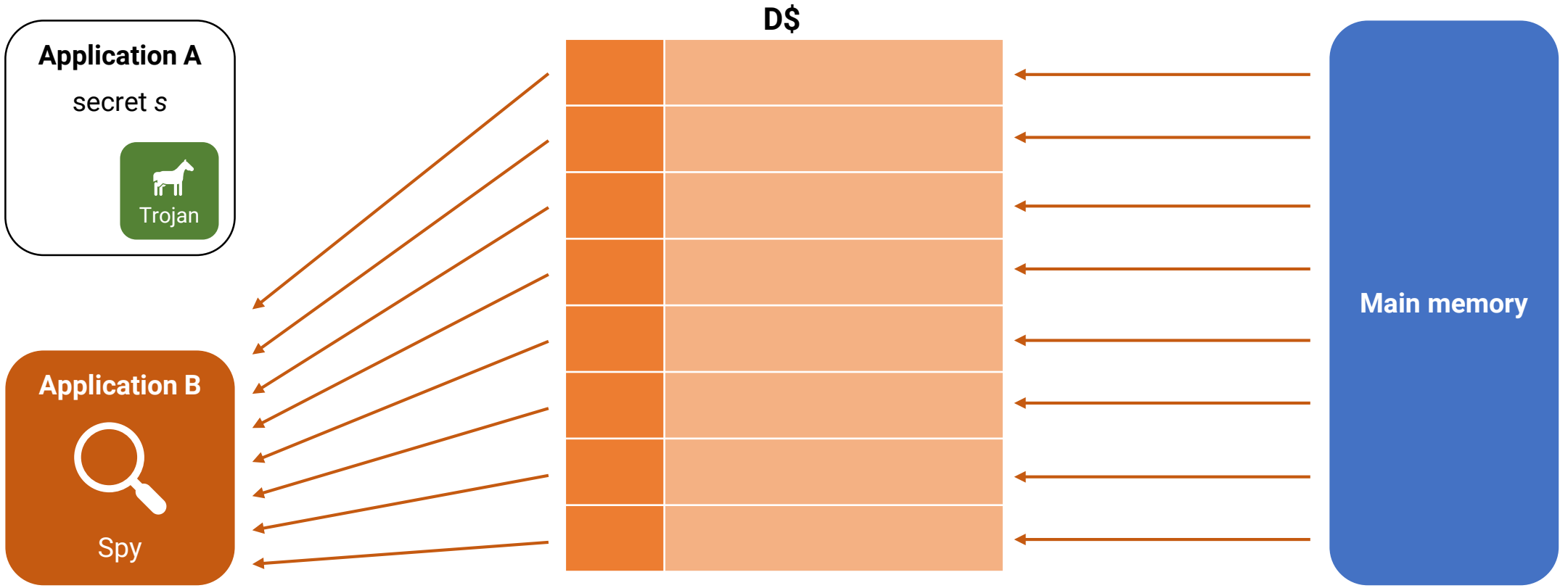
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime**

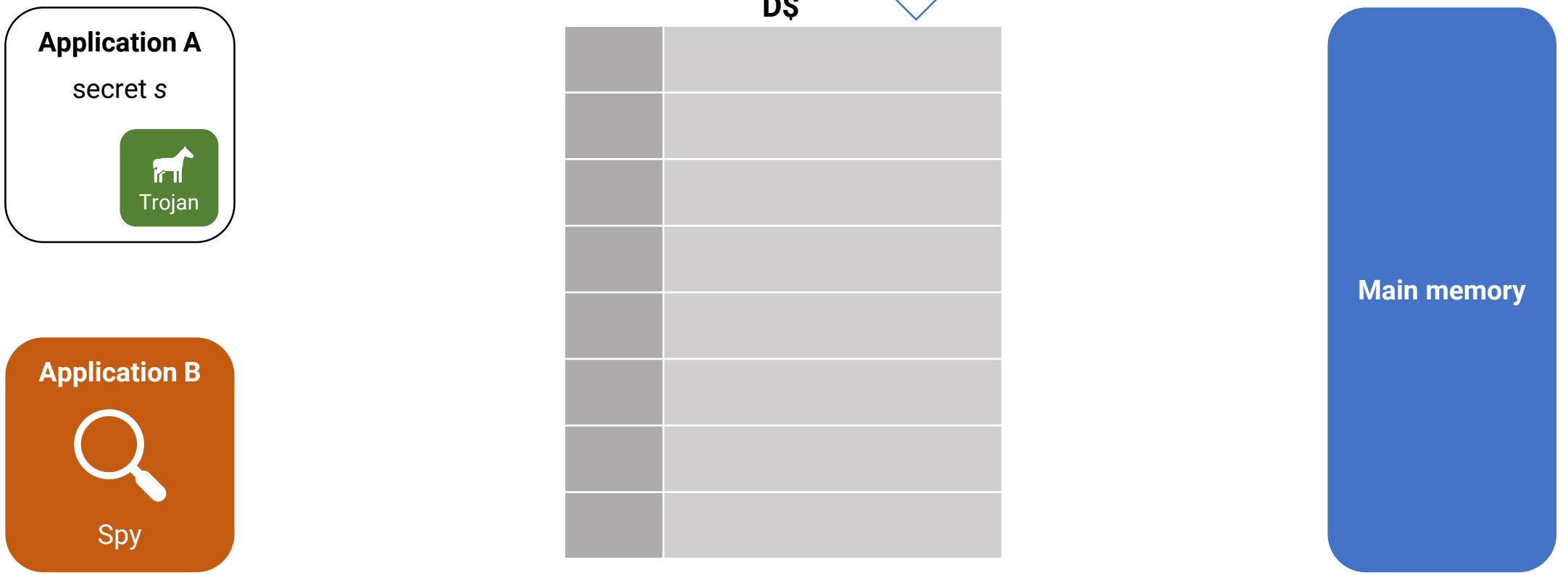
(2) OS: **Cont. sw.**

(3) Trojan: **Encode s**

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Temporal Partitioning



(1) Spy: **Prime**

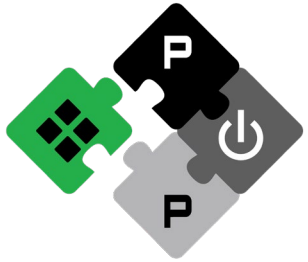
(2) OS: **Cont. sw.**

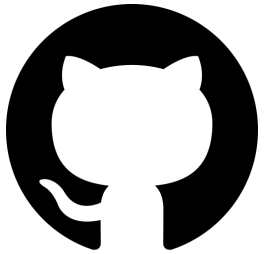
(3) Trojan: **Encode *s***

(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Evaluation Platform



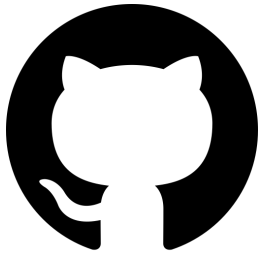


Hardware platform

Ariane RV64GC core [4]

- 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

Evaluation Platform



Hardware platform

CVA6 RV64GC core [4]

- 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

Evaluation Platform

[11]

Hardware platform

CVA6 RV64GC core [4]

- 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

Evaluation Platform

Supervisor — seL4 microkernel [5]

- Formally verified μKernel by Data61
- Experimental version with time protection
- Focus on security
- Port to CVA6
- Enable cache colouring of LLC

Hardware platform — CVA6 RV64GC core [4]

- 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

Evaluation Platform

Application

Channel bench [1]

- Measure covert channels
- Port to RISC-V
- Tailor attacks to CVA6's μArch

Supervisor

seL4 microkernel [5]

- Formally verified μKernel by Data61
- Experimental version with time protection
- Focus on security
- Port to CVA6
- Enable cache colouring of LLC

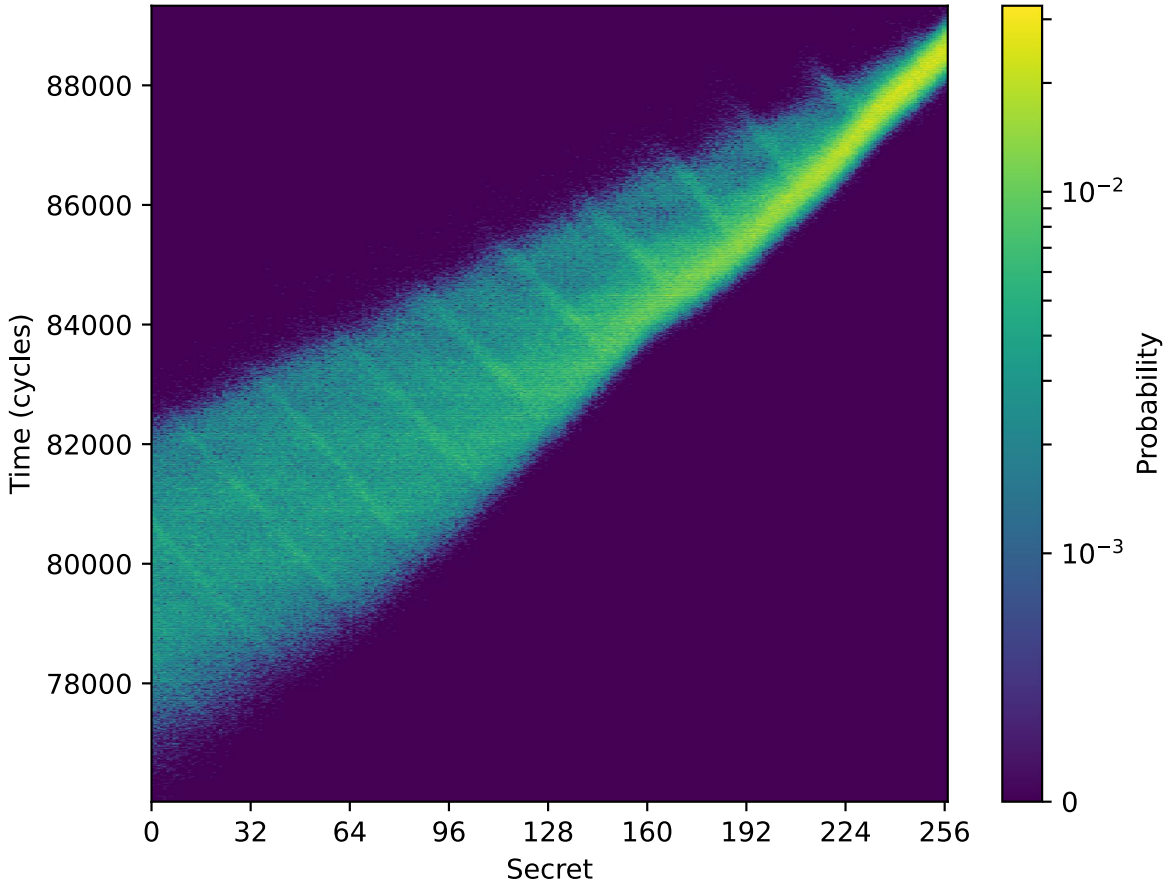
Hardware platform

CVA6 RV64GC core [4]

- 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

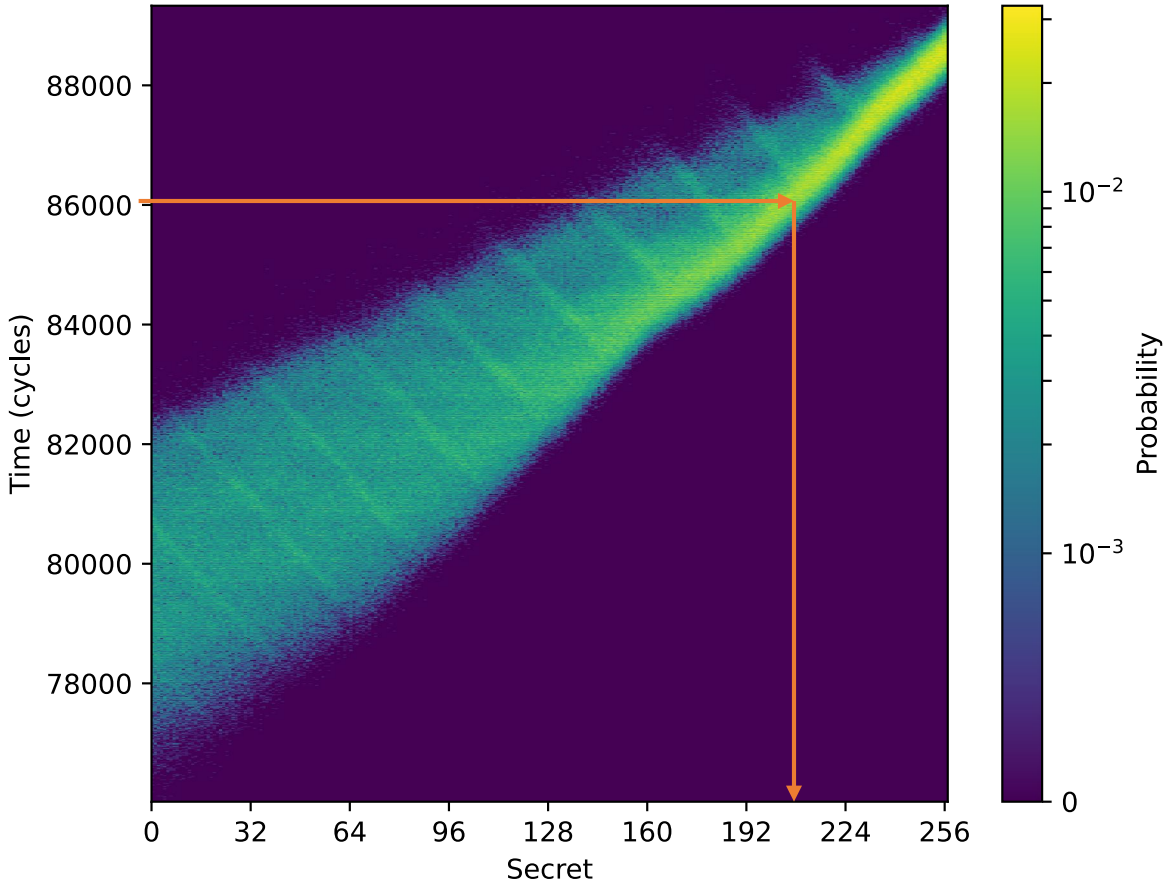
Channel Matrix: L1 D$

$N = 10^6$



Channel Matrix: L1 D$

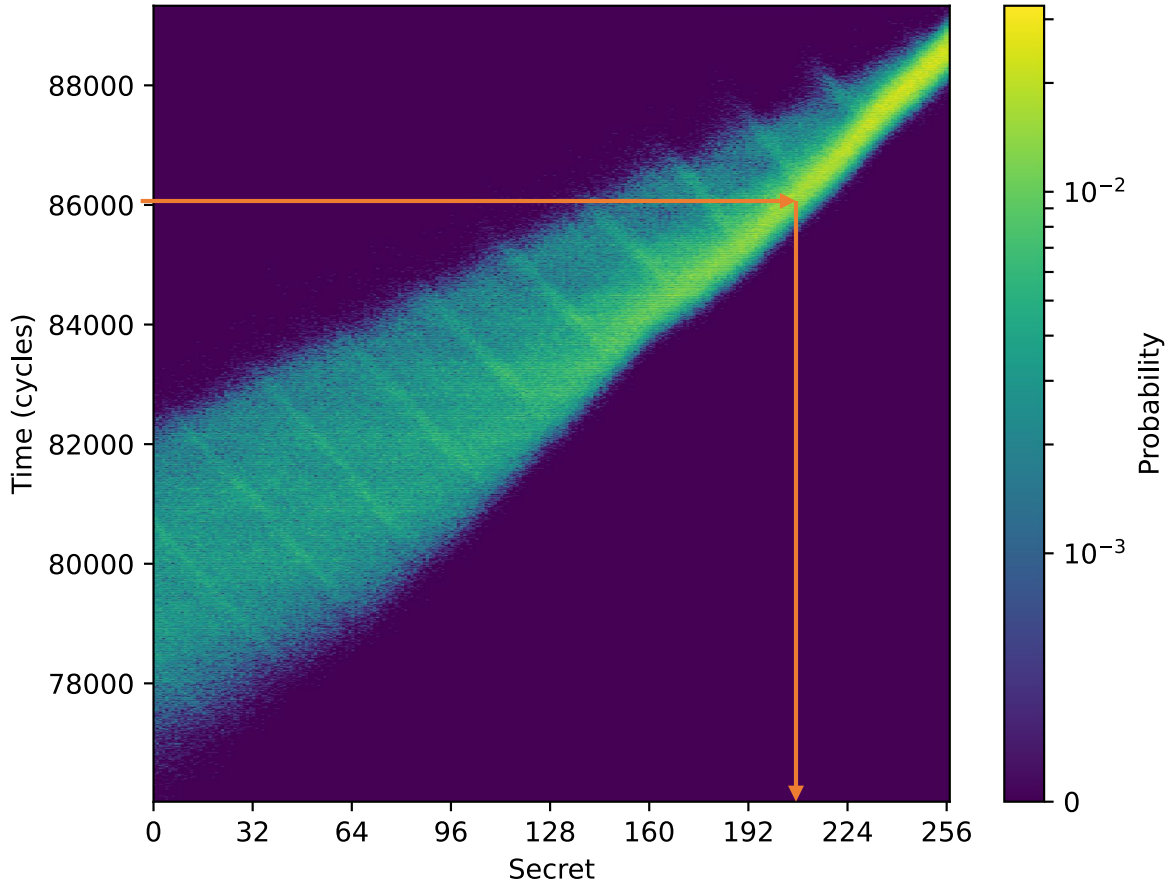
$N = 10^6$



Channel Matrix: L1 D$

$N = 10^6$

$M = 1667.3$ mb

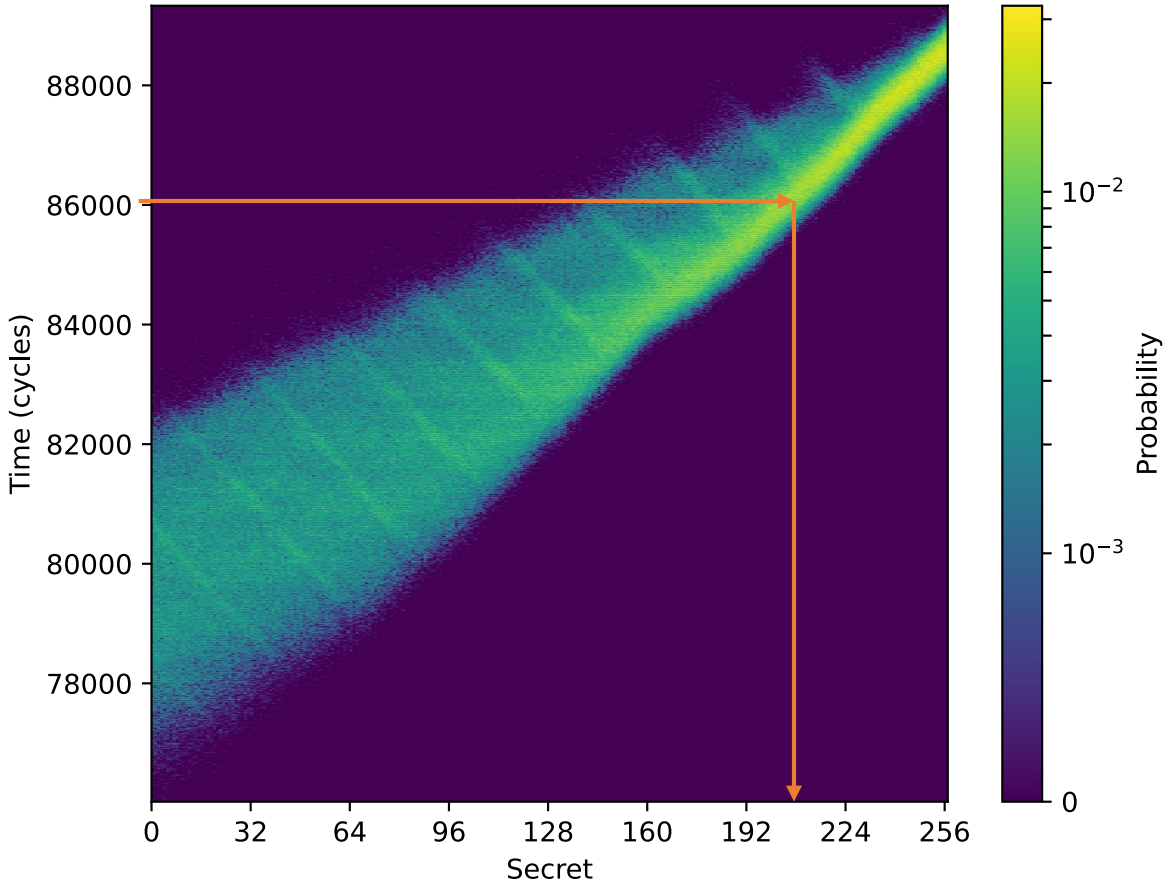


Channel Matrix: L1 D$

$N = 10^6$

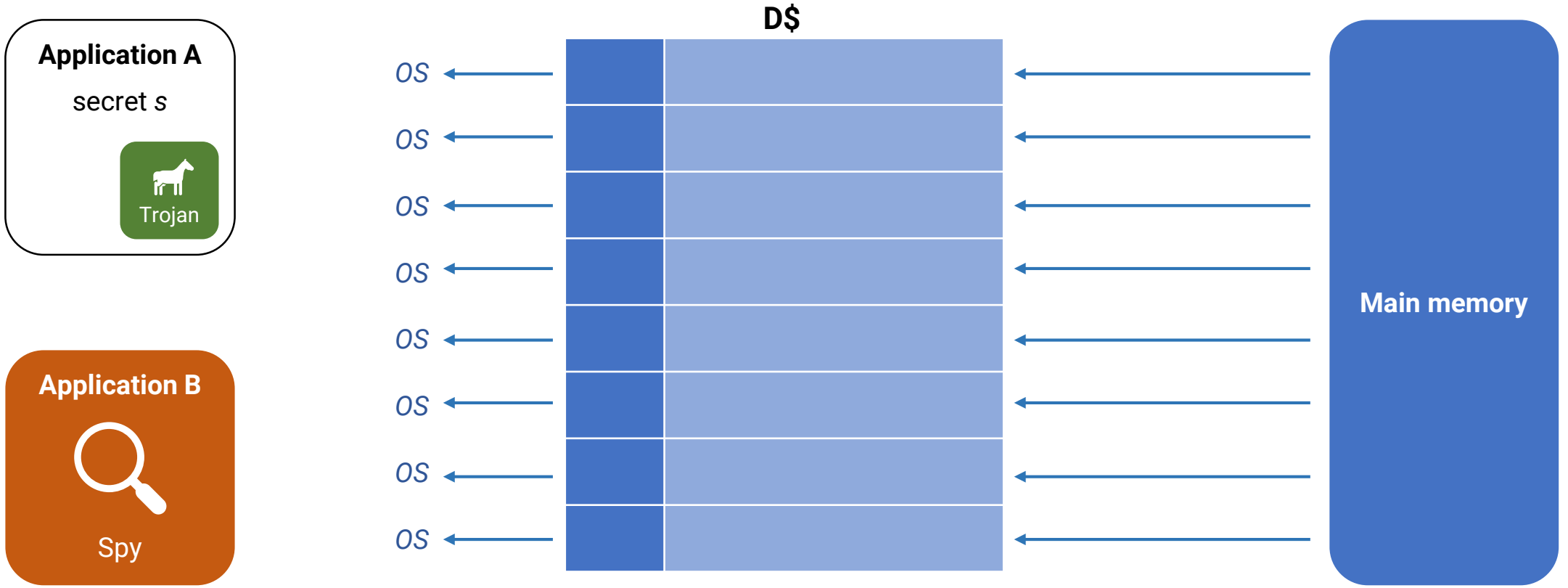
M = 1667.3 mb

M_0 = 0.5 mb



M_0 varies between Measurements!

Let‘s try to flush in Software!



(1) Spy: **Prime**

(2) OS: **Cont. sw.**

(3) Trojan: **Encode *s***

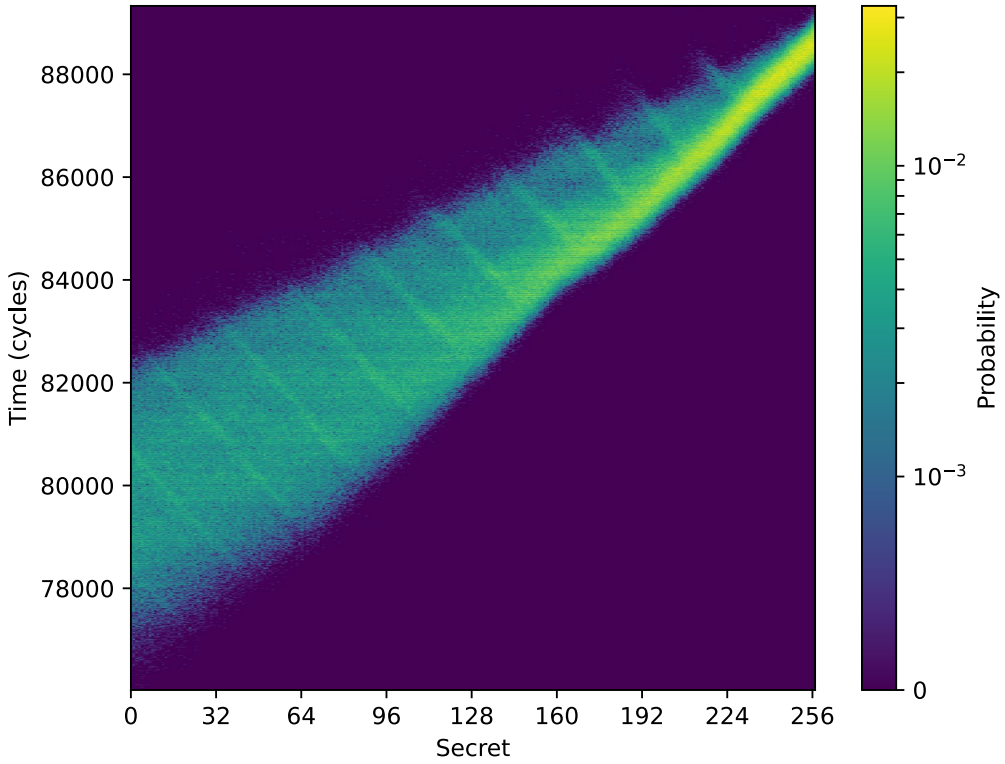
(4) OS: **Cont. sw.**

(5) Spy: **Probe**

Software Mitigation

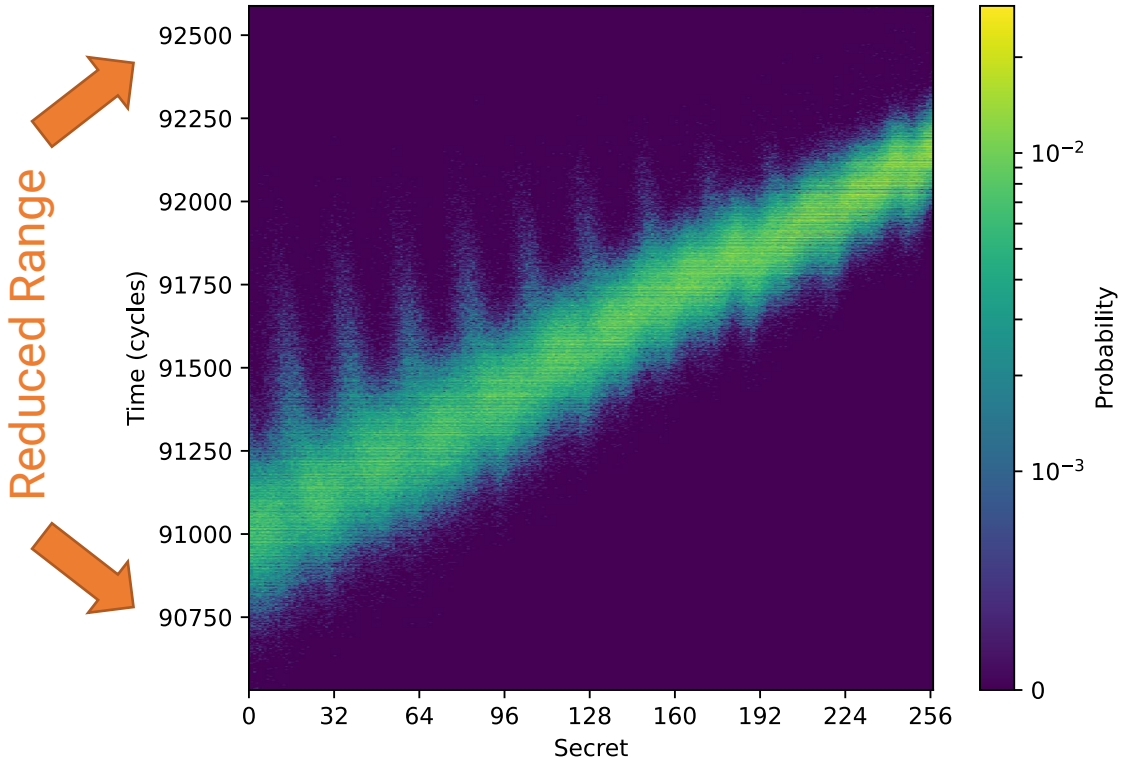
L1 D$ Channel

Unmitigated



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

L1 D$ prime on context switch

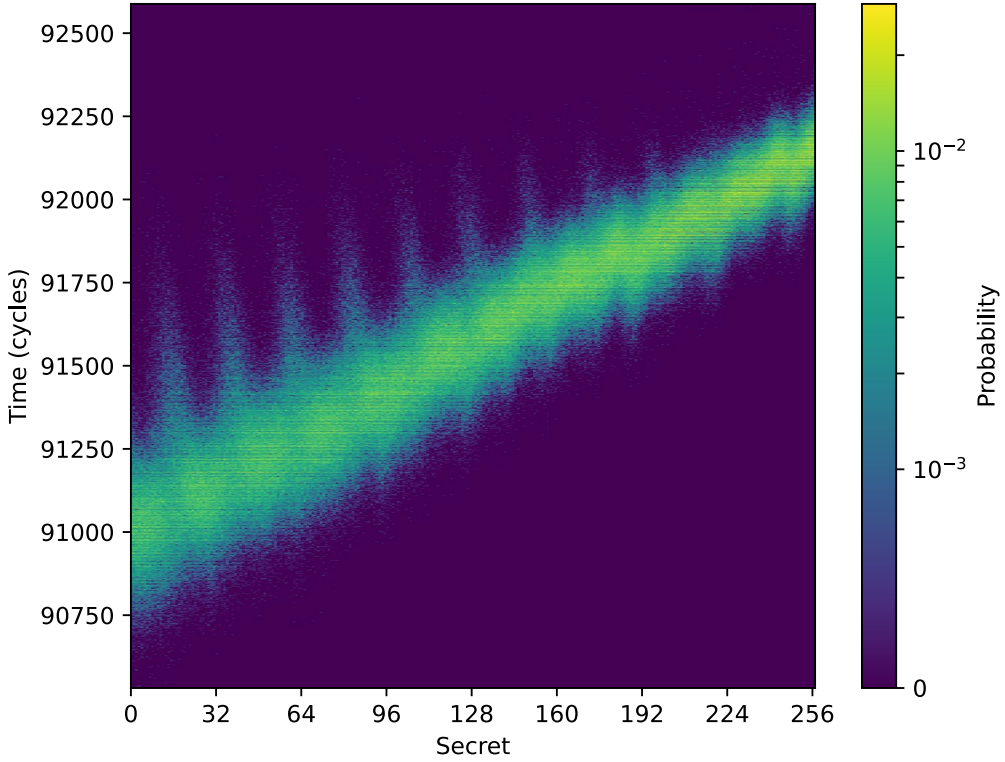


$N = 10^6$, M = 1471.5 mb, M_0 = 0.6 mb

Software Mitigation

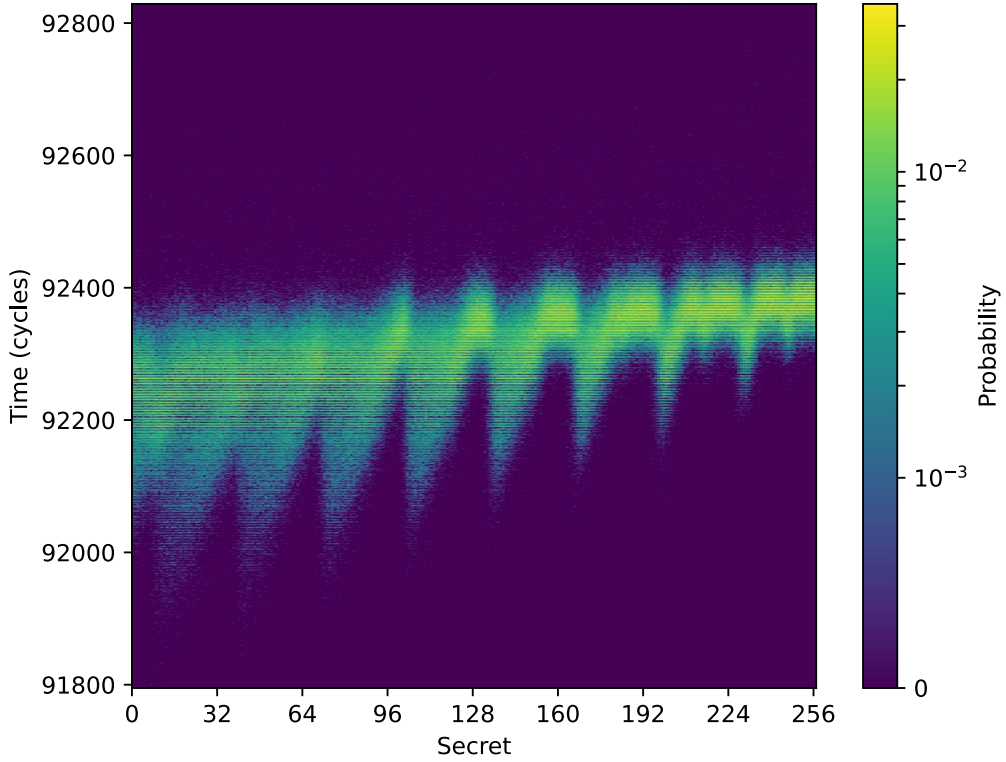
L1 D$ Channel

Single L1 D$ prime on context switch



$N = 10^6$, M = 1471.5 mb, M_0 = 0.6 mb

Double L1 D$ prime on context switch

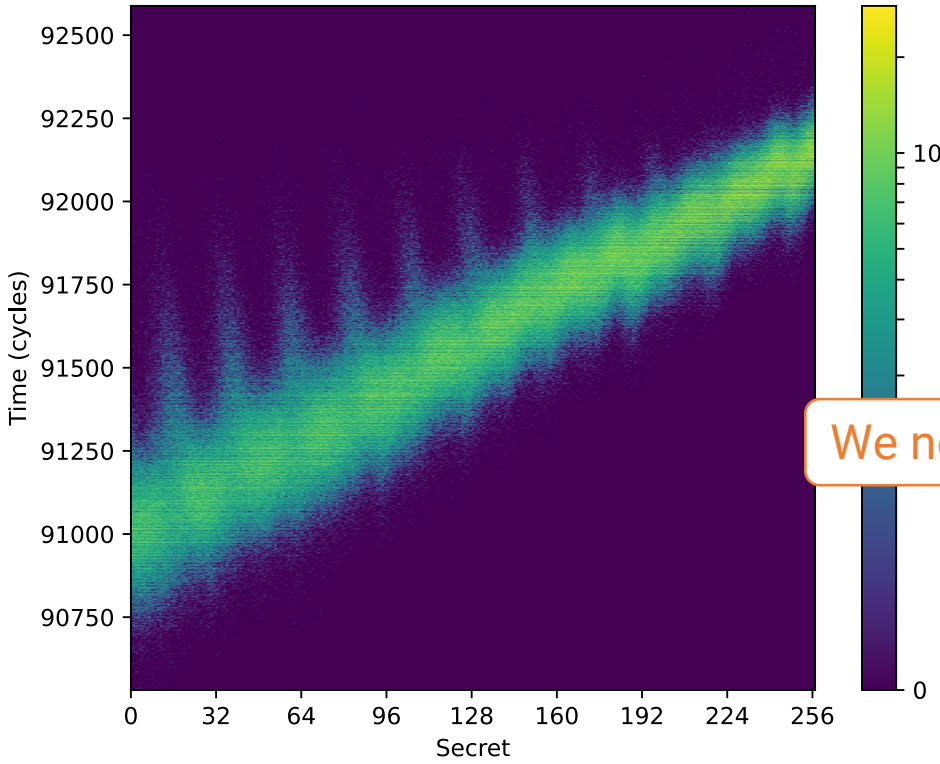


$N = 10^6$, M = 515.7 mb, M_0 = 1.1 mb

Software Mitigation

L1 D$ Channel

Single L1 D$ prime on context switch

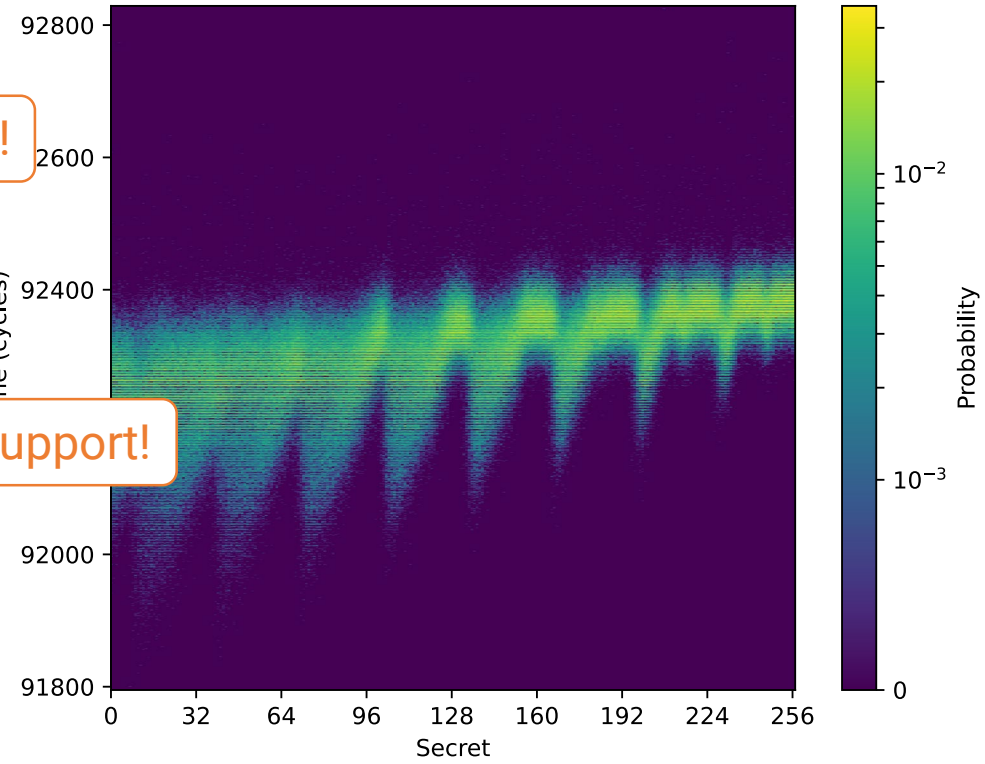


$N = 10^6$, M = 1471.5 mb, M_0 = 0.6 mb

Still a Channel!

We need Hardware Support!

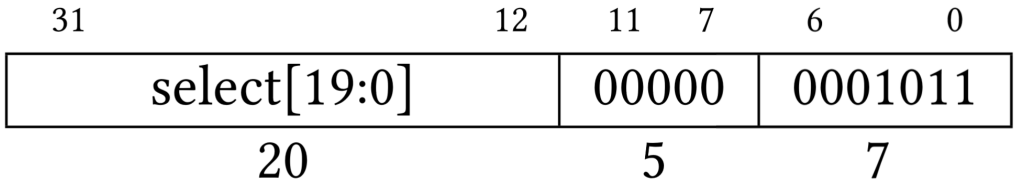
Double L1 D$ prime on context switch



$N = 10^6$, M = 515.7 mb, M_0 = 1.1 mb

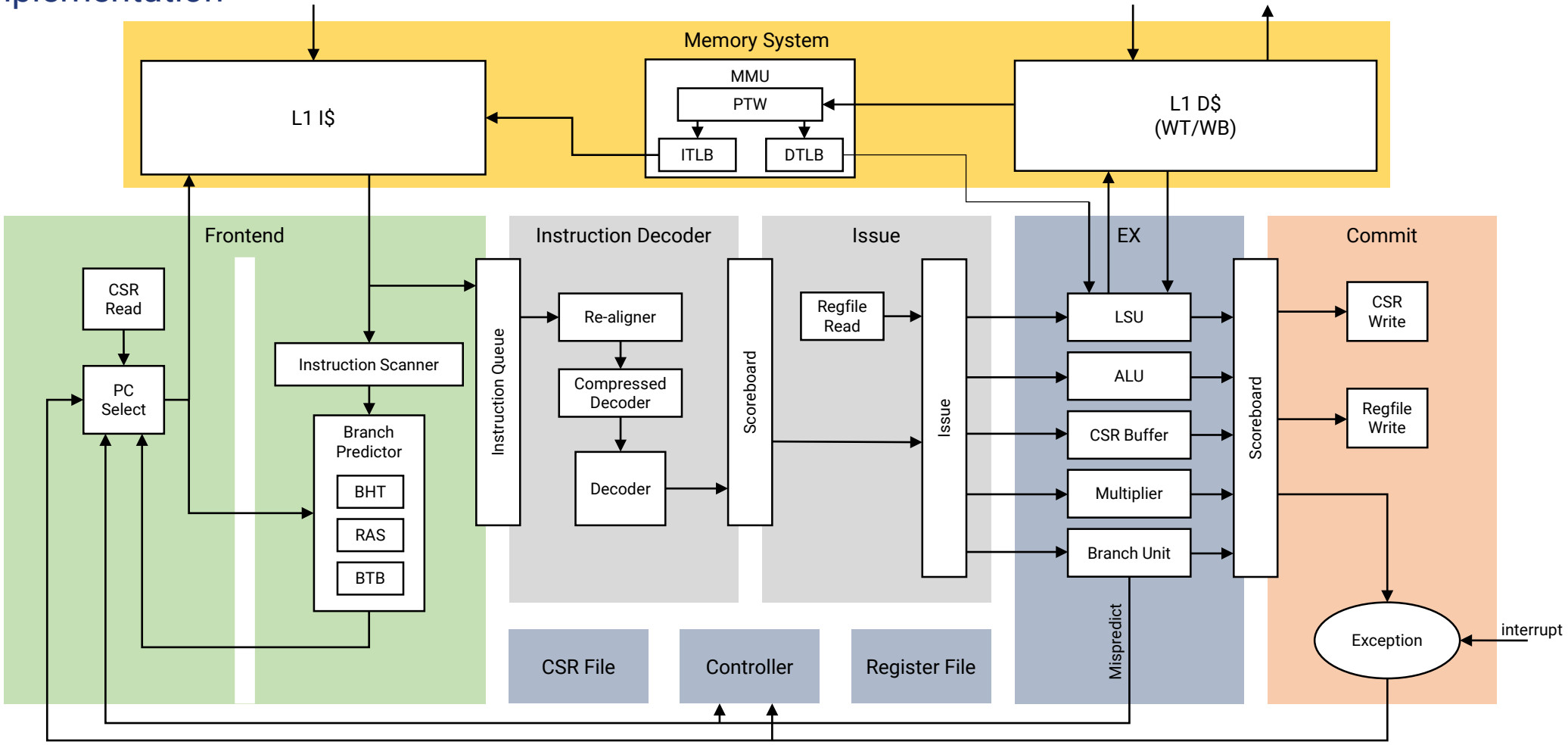
Temporal Fence Instruction (fence.t)

Encoding



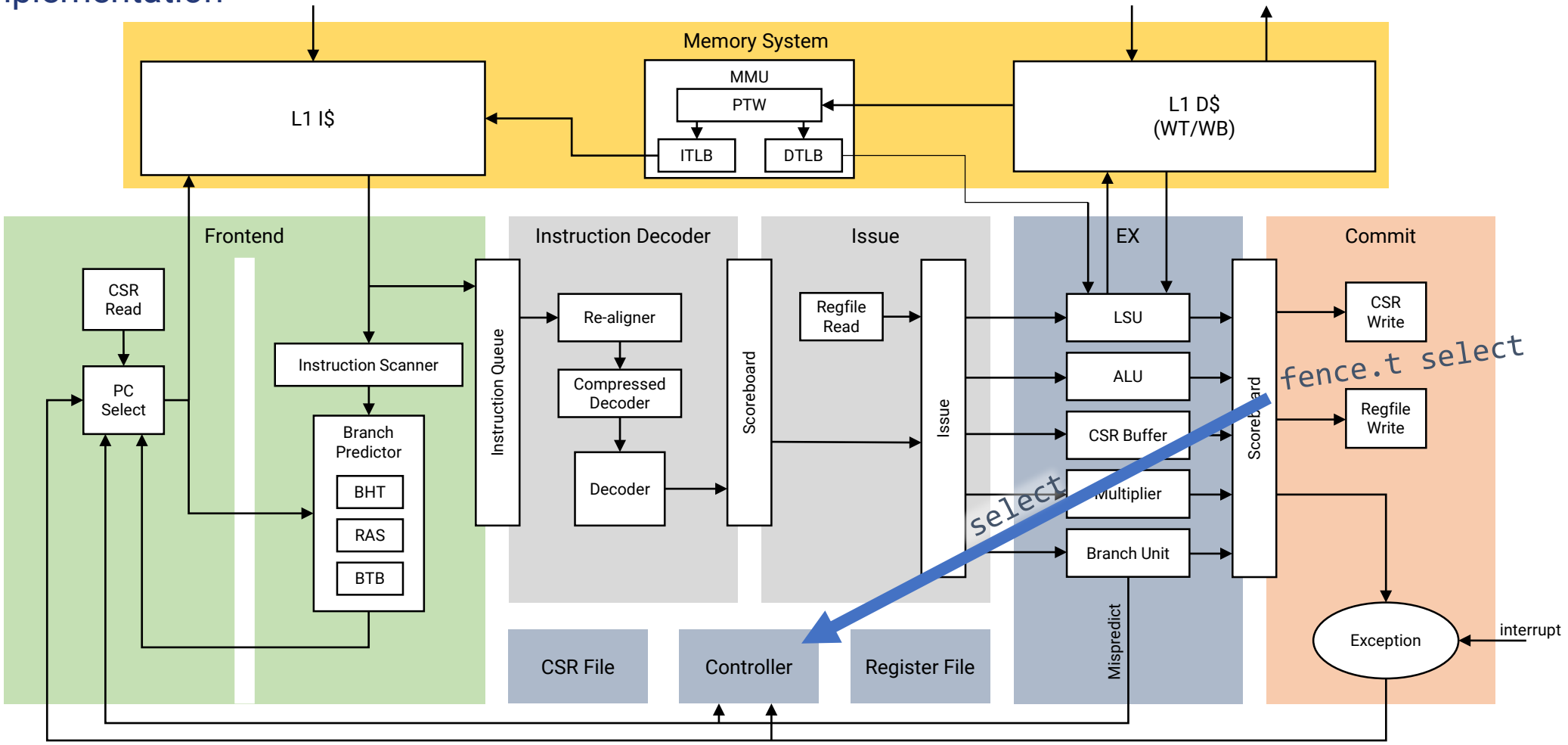
Temporal Fence Instruction (fence.t)

Implementation



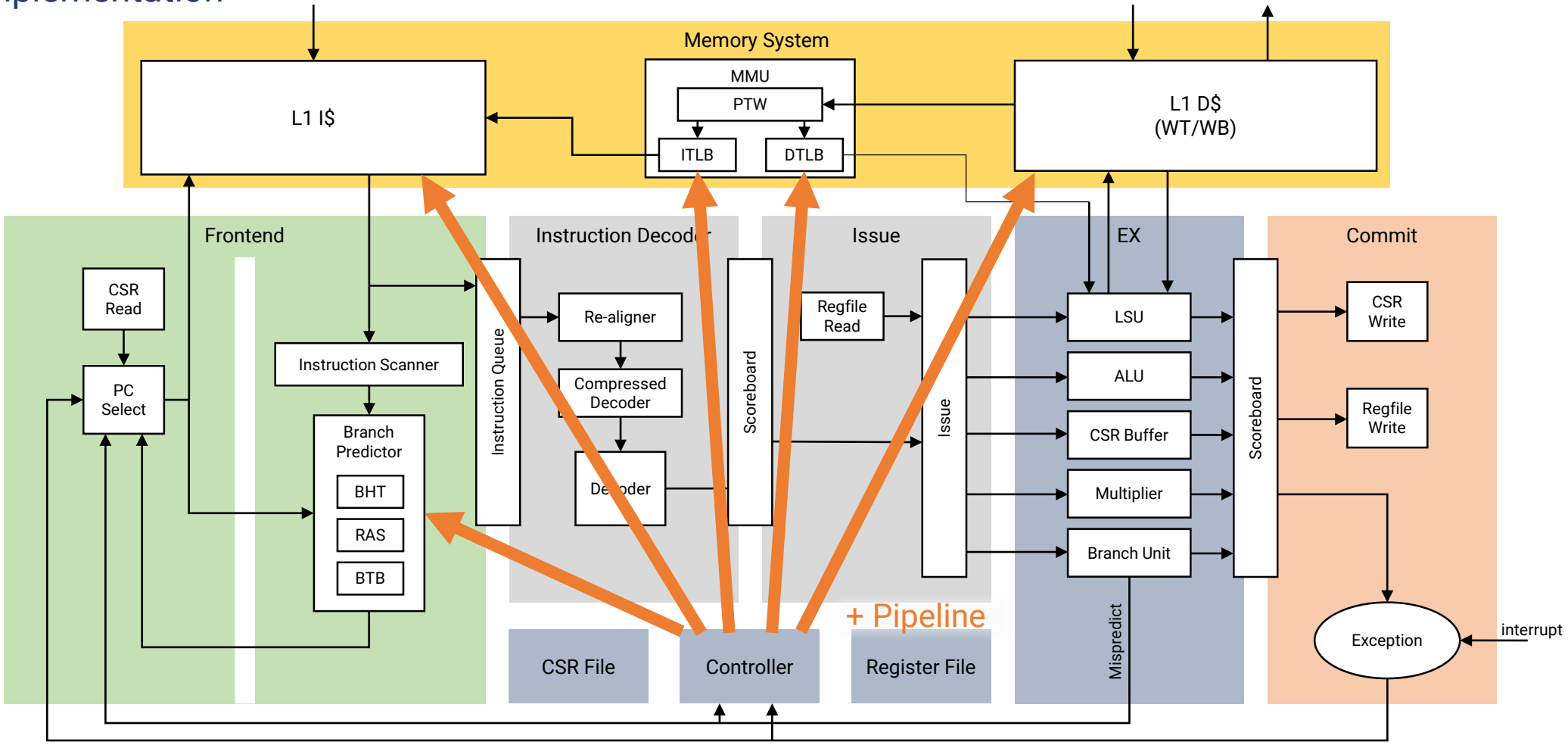
Temporal Fence Instruction (fence.t)

Implementation



Temporal Fence Instruction (fence.t)

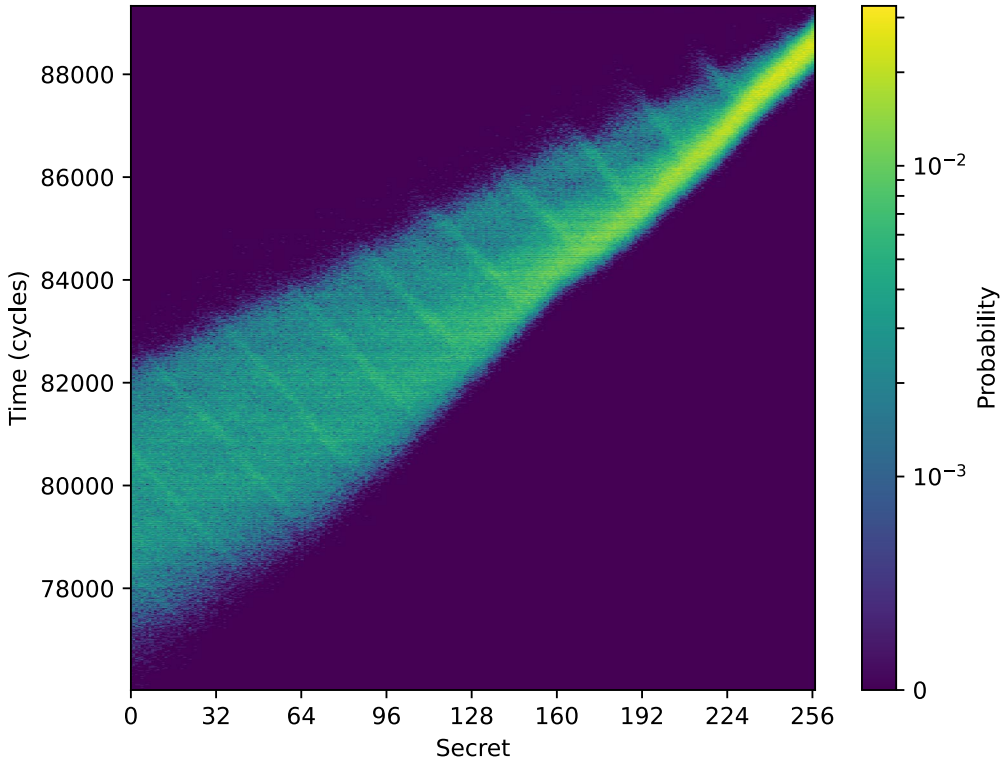
Implementation



fence.t

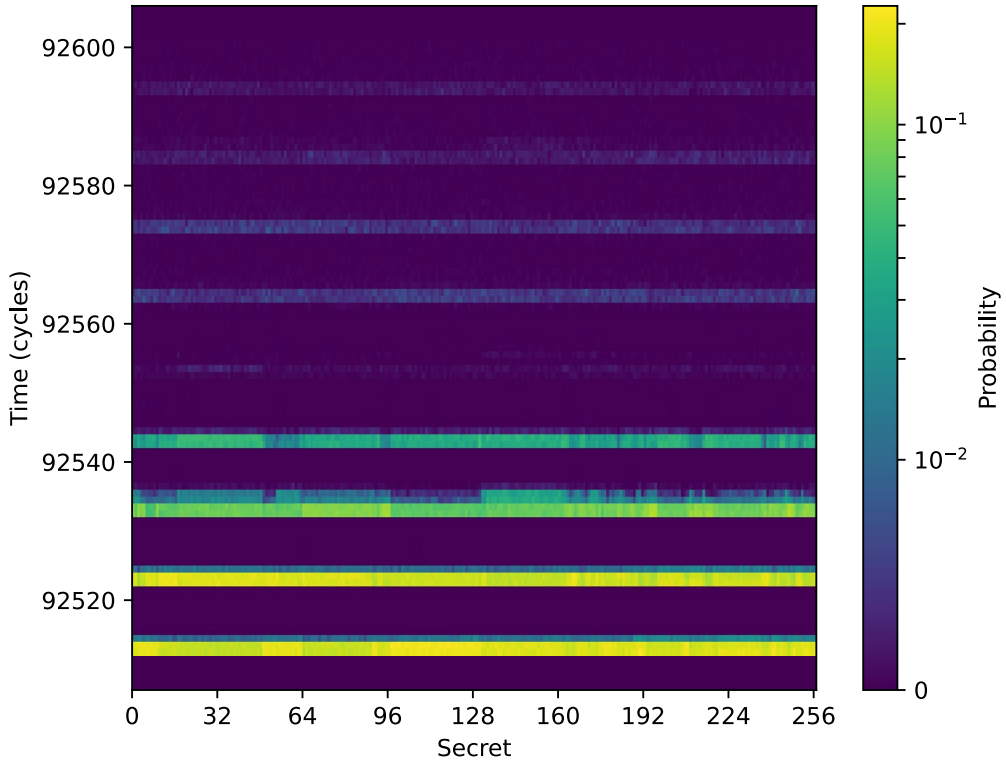
L1 D$ Channel

Unmitigated



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

Flush targeted components on context switch

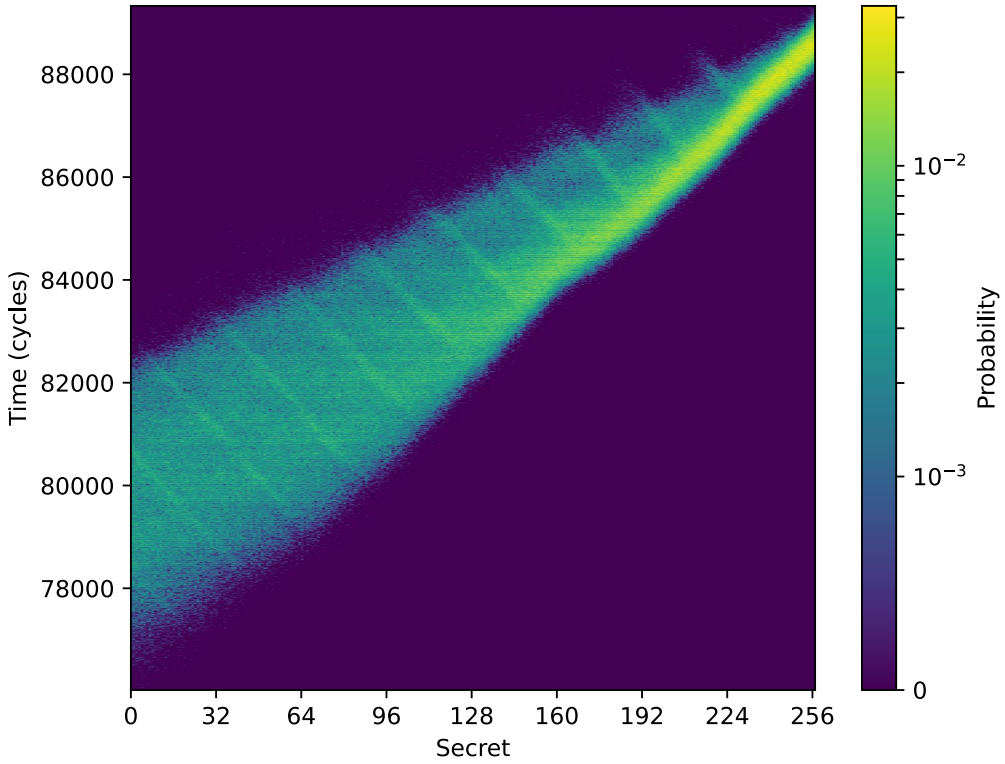


$N = 10^6$, M = 7.7 mb, M_0 = 1.4 mb

fence.t

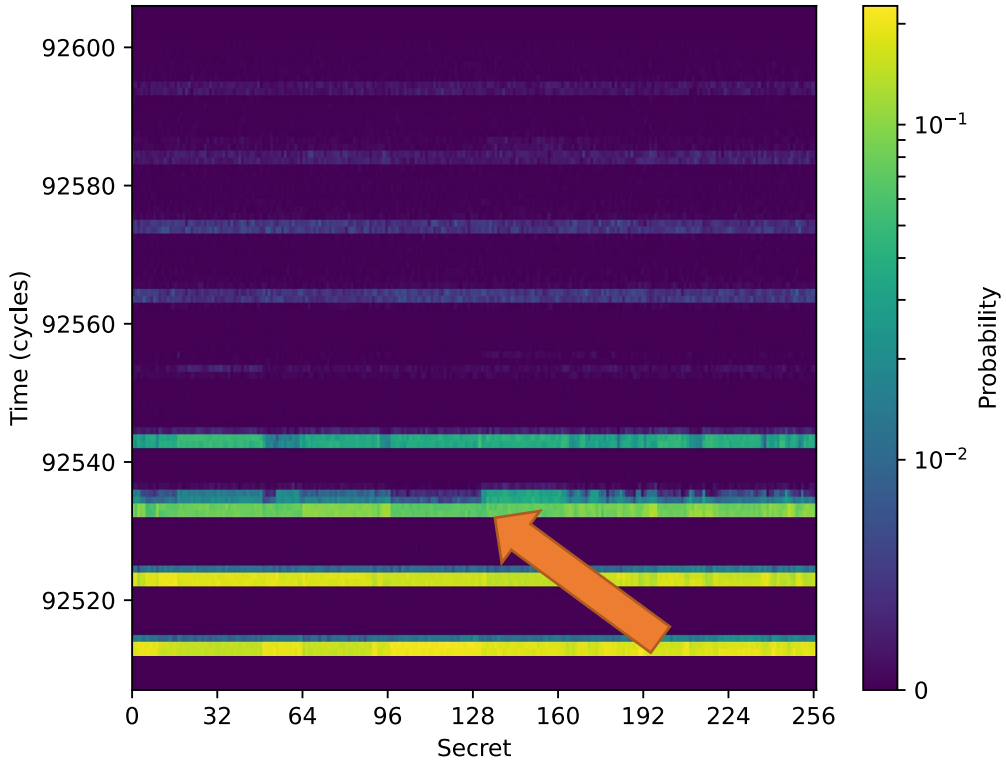
L1 D$ Channel

Unmitigated



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

Flush targeted components on context switch

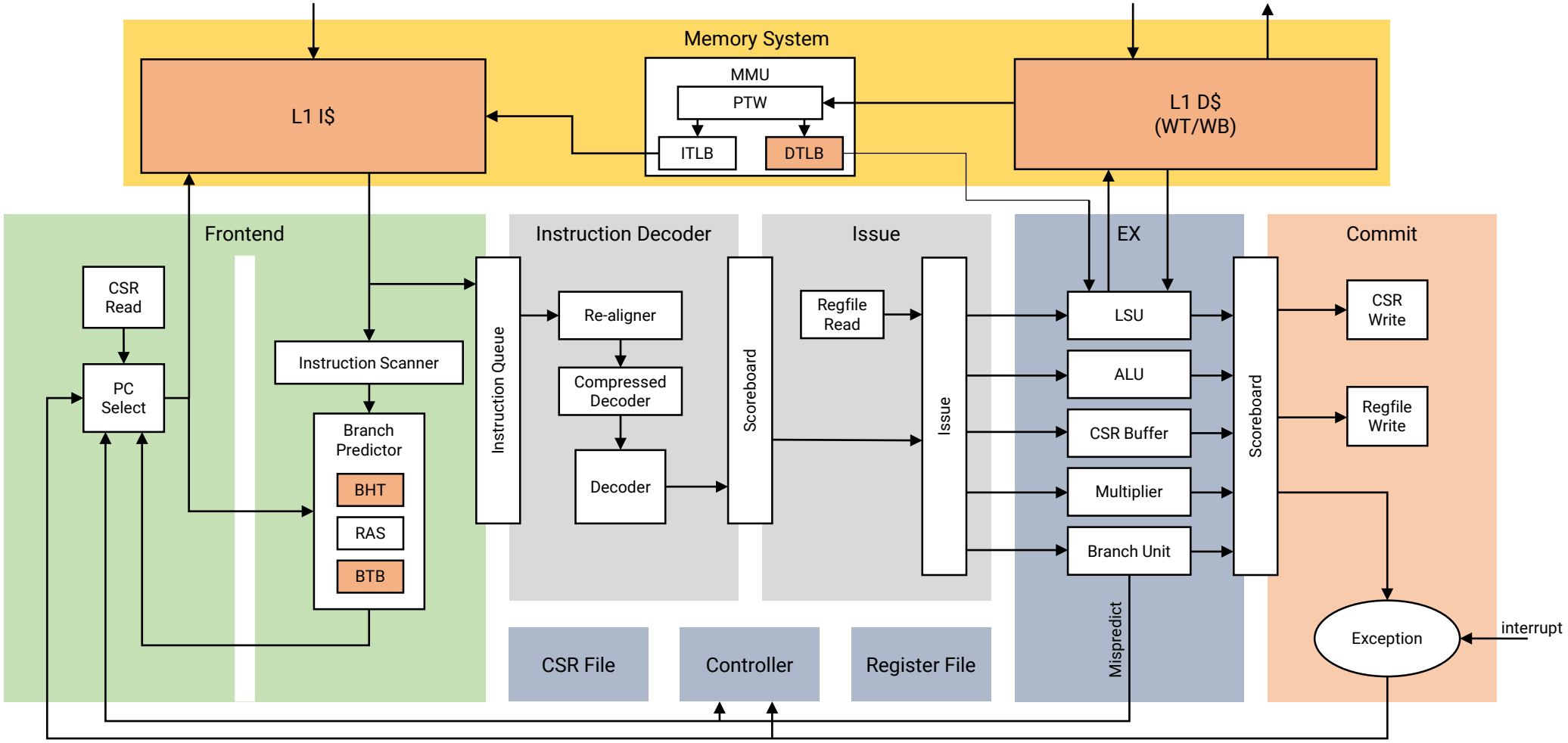


$N = 10^6$, M = 7.7 mb, M_0 = 1.4 mb

Vulnerable 2nd Order State-Holding Components

- **L1 D$:**
  - LFSR for pseudo-random replacement policy
  - Memory arbiter
  - TX FIFO
  - Write-buffer arbiters
- **L1 I$:**
  - LFSR for pseudo-random replacement policy
- **TLBs:**
  - Pseudo-LRU tree for replacement policy

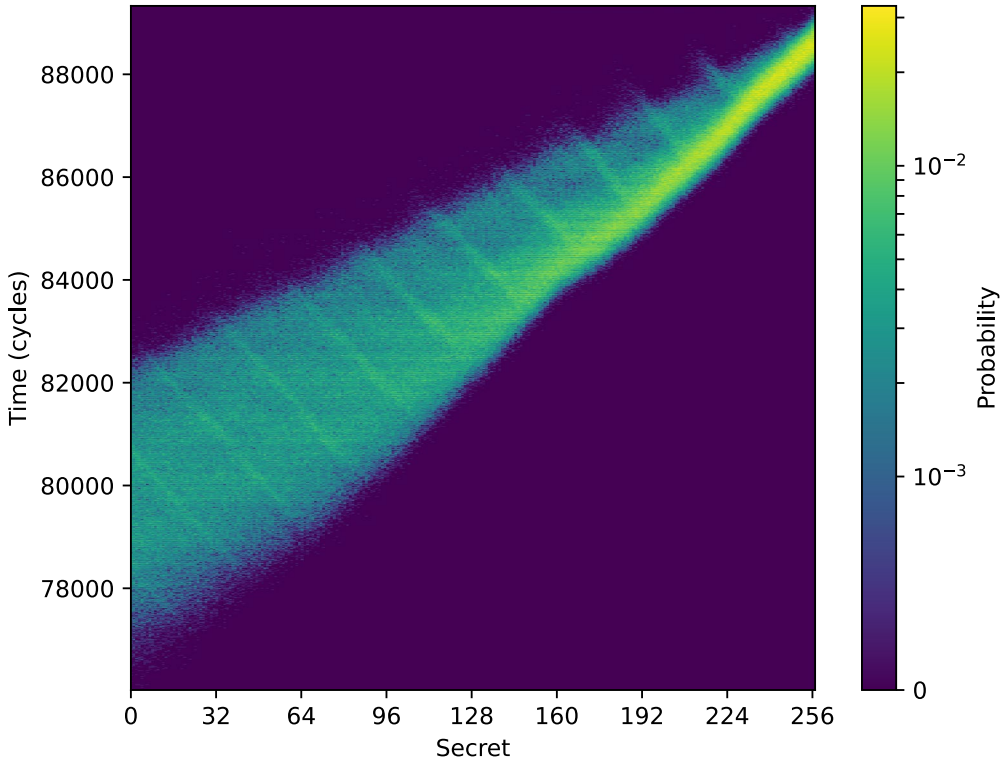
Let's Have a Look at All Targeted Channels!



L1 D$ Channel

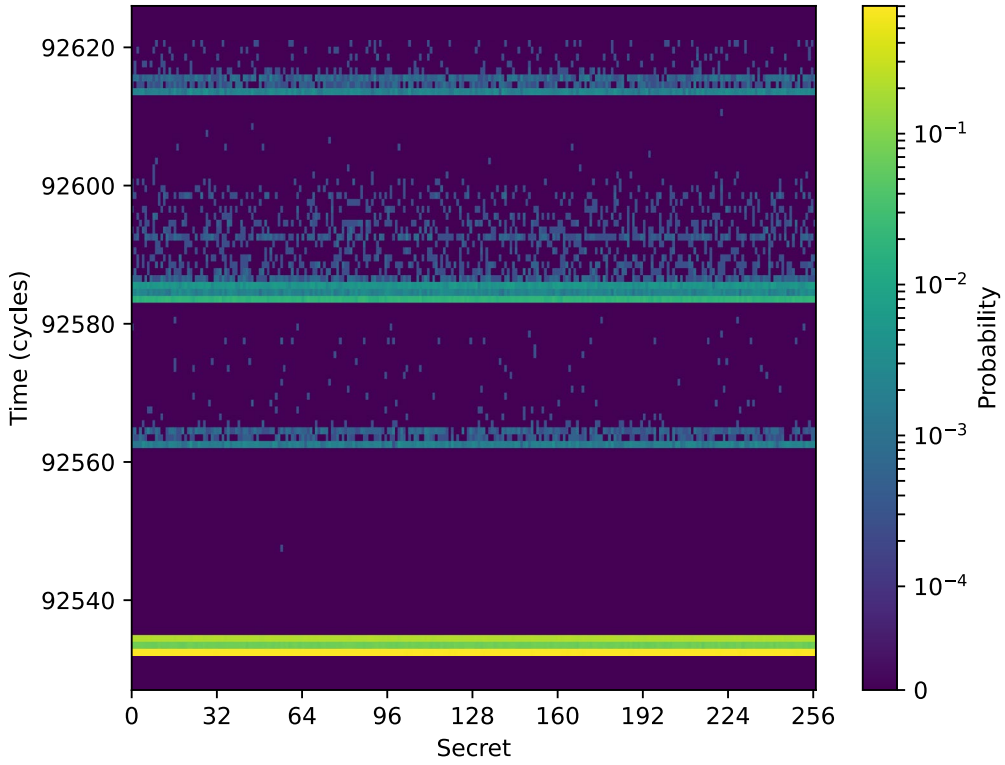
Full fence.t

Unmitigated



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

Flush all vulnerable components on context switch

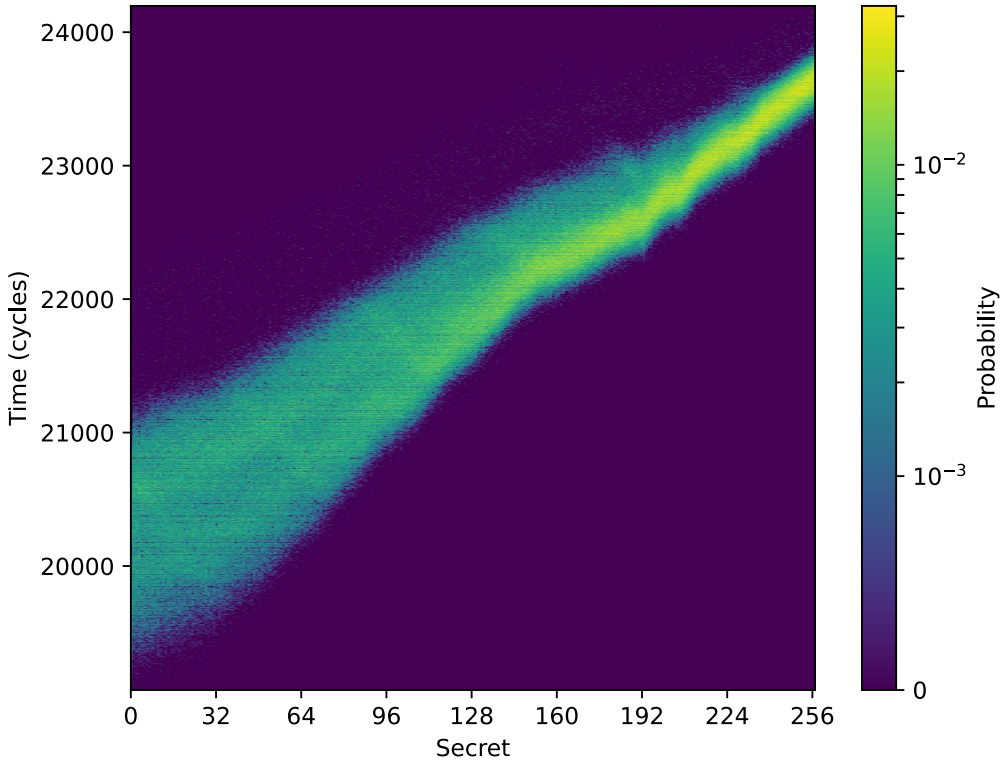


$N = 10^6$, M = 8.4 mb, M_0 = 9.6 mb

L1 I$ Channel

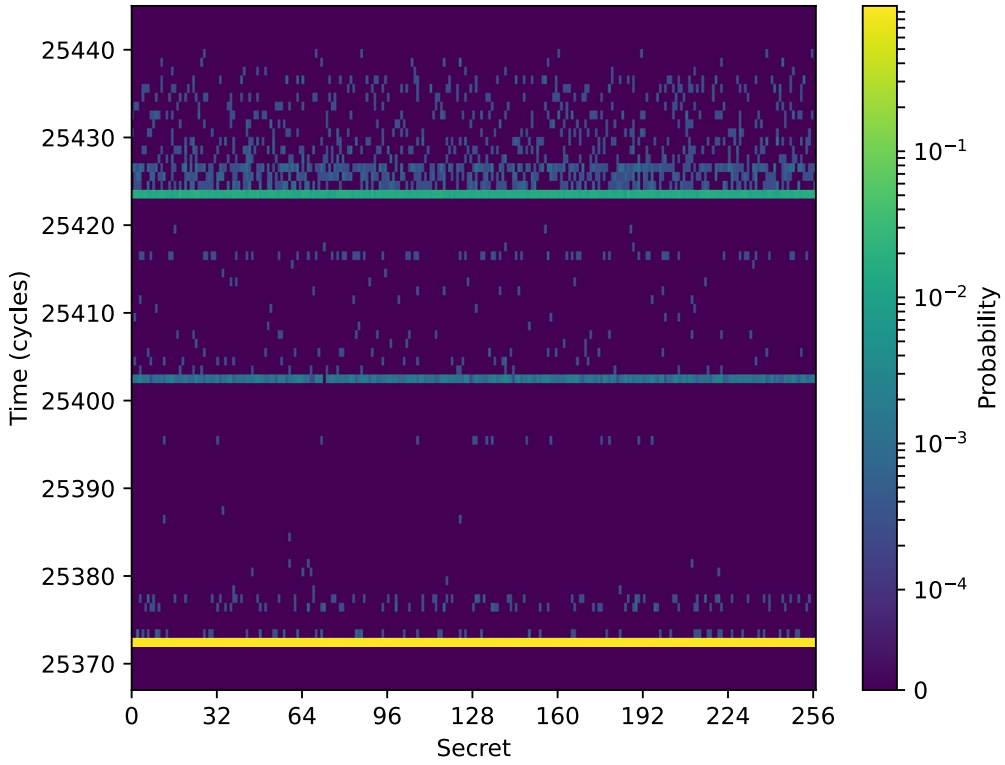
Full fence.t

Unmitigated



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

Flush all vulnerable components on context switch

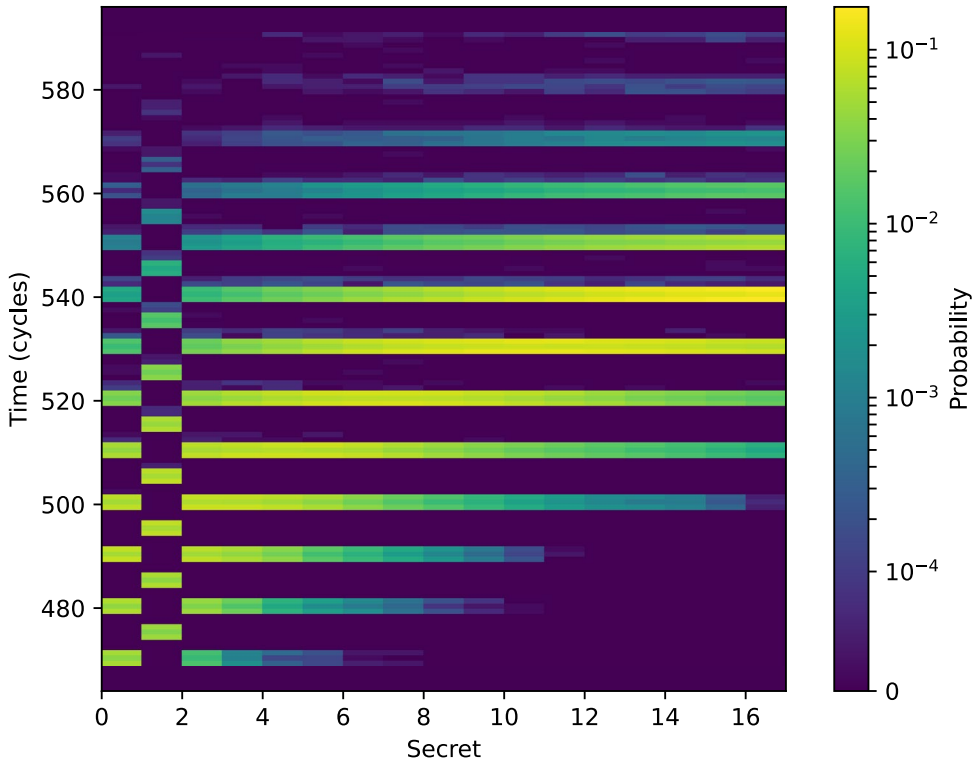


$N = 10^6$, M = 19.5 mb, M_0 = 20.5 mb

TLB Channel

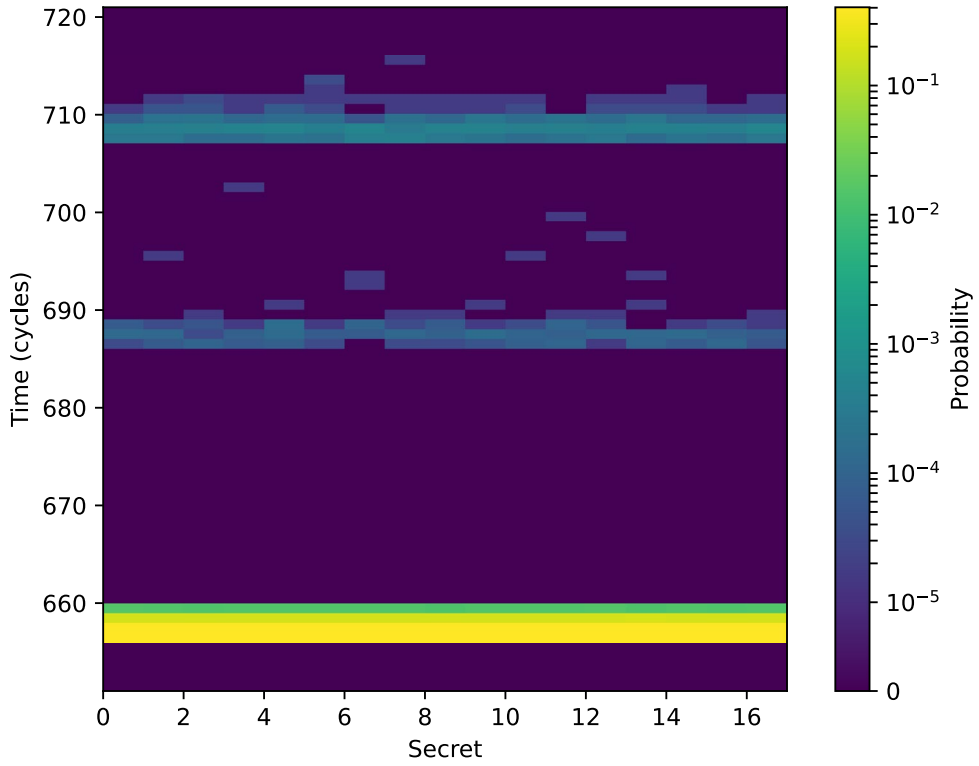
Full fence.t

Unmitigated



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

Flush all vulnerable components on context switch

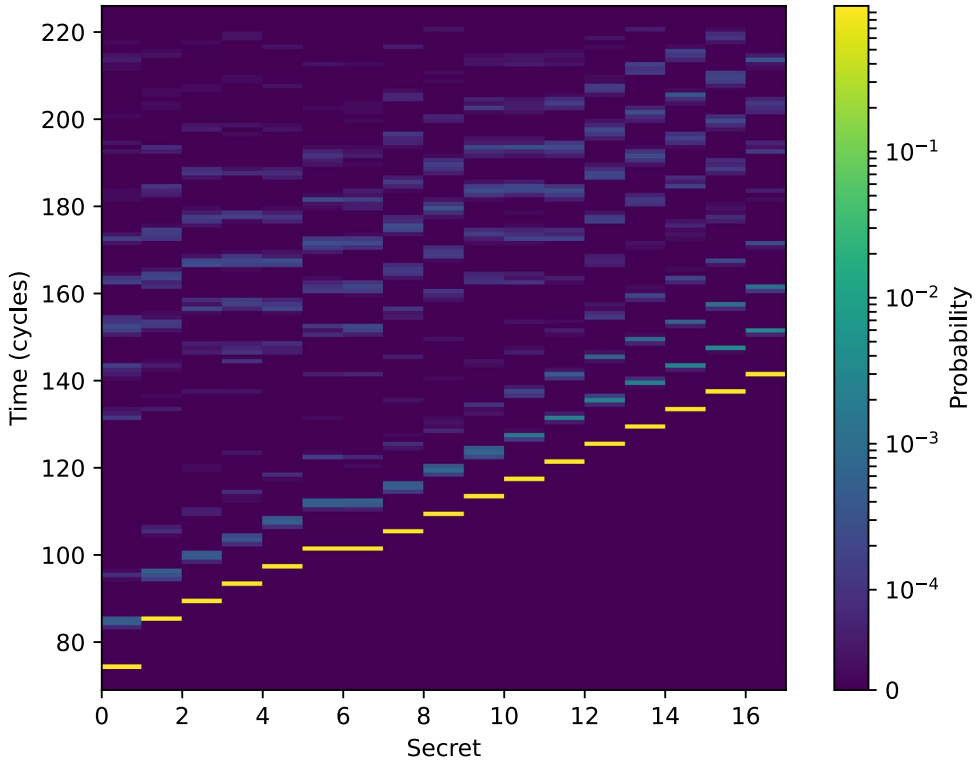


$N = 10^6$, $M = 2.7$ mb, $M_0 = 5.4$ mb

BTB Channel

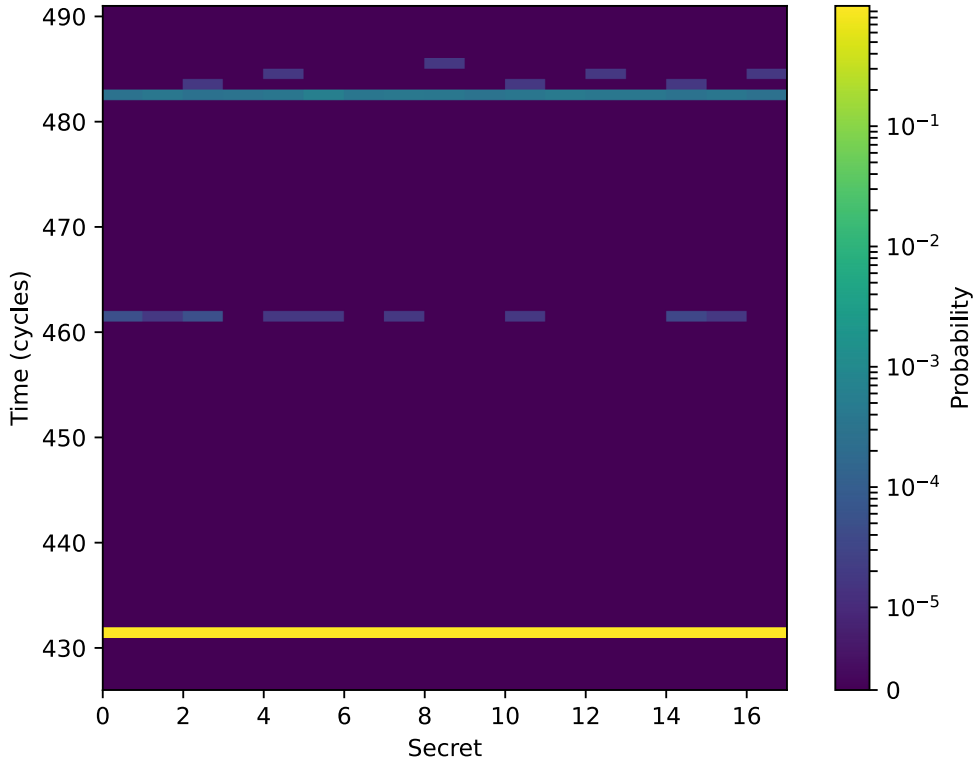
Full fence.t

Unmitigated



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

Flush all vulnerable components on context switch

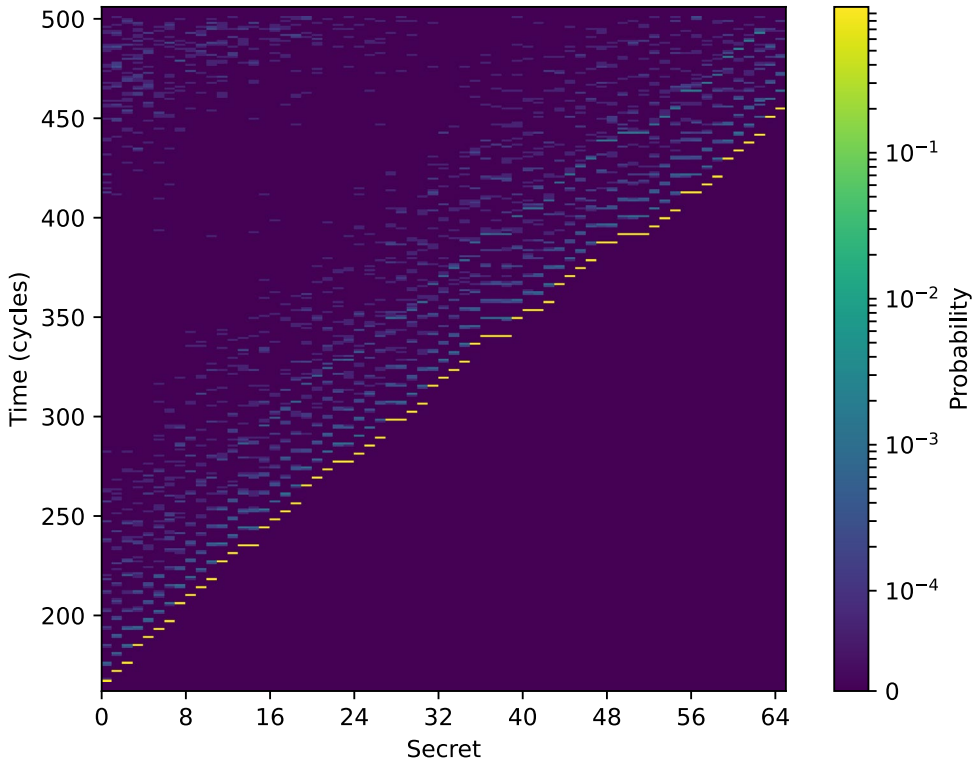


$N = 10^6$, M = 33.0 mb, M_0 = 57.6 mb

BHT Channel

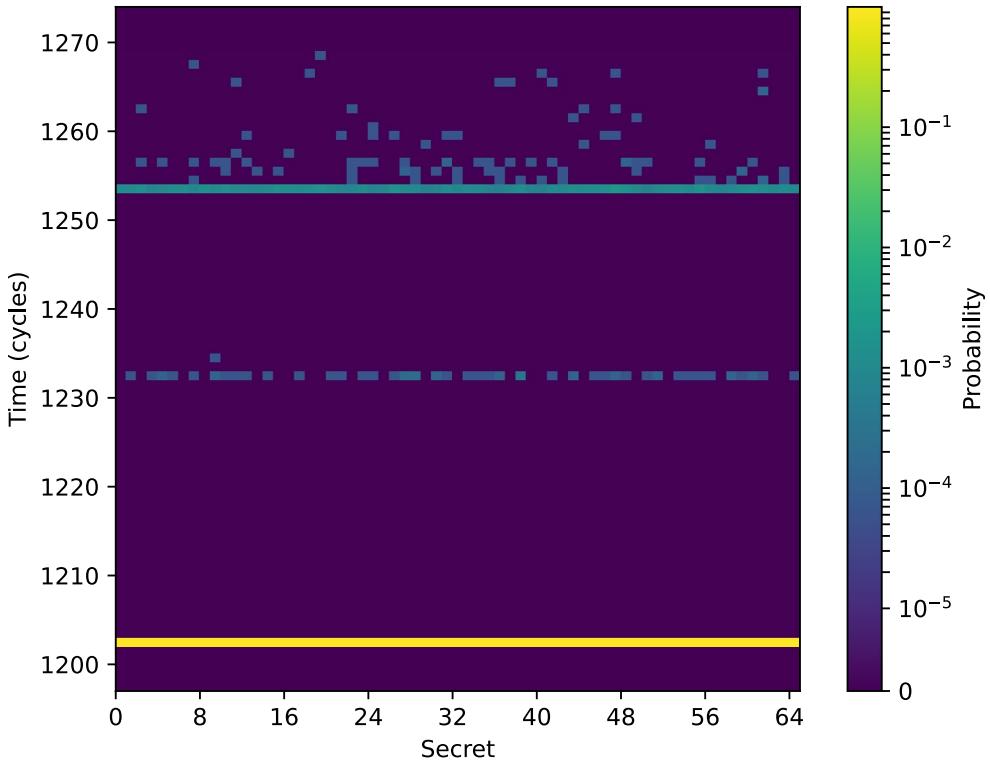
Full fence.t

Unmitigated



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

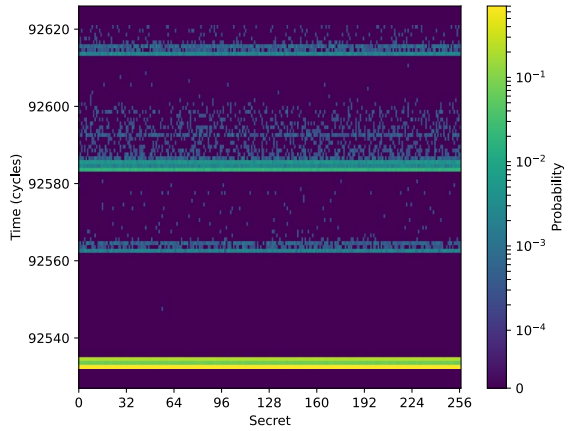
Flush all vulnerable components on context switch



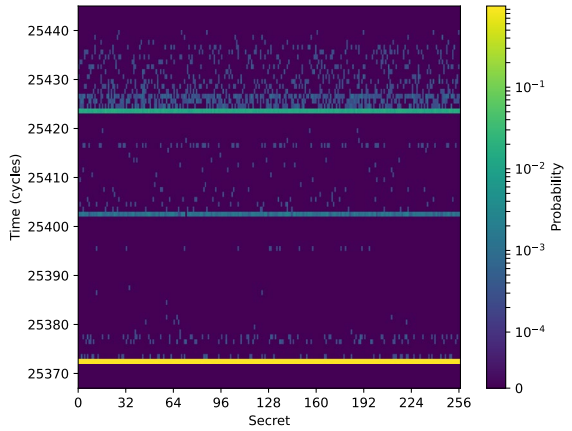
$N = 10^6$, $M = 44.1$ mb, $M_0 = 58.8$ mb

All Evaluated Channels Closed!

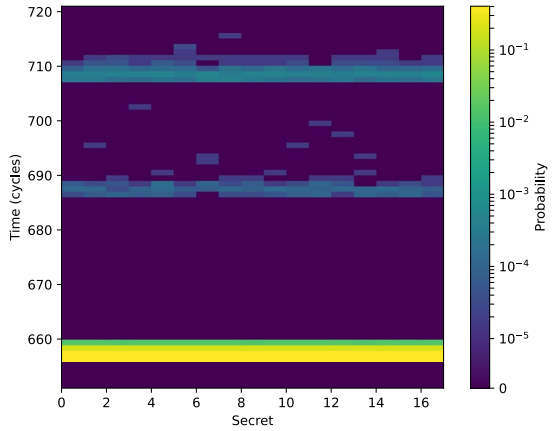
L1 D$



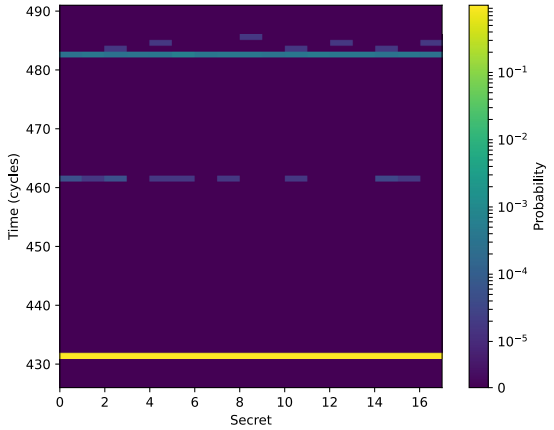
L1 I$



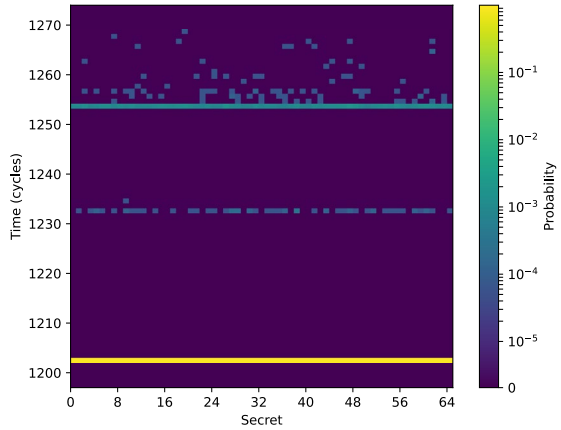
TLB



BTB



BHT



So What Are the Costs?

Context Switch Latency

seL4 one-way inter-address-space IPC microbenchmark

Unmitigated		D$ Software Flush		HW Flush
Hot	Cold	Single	Double	
430 (±7.0)	1,180 (±1.0)	12,099 (±52)	51,876 (±256)	1,502 (±0.9)

So What Are the Costs?

Context Switch Latency

seL4 one-way inter-address-space IPC microbenchmark

Unmitigated		D$ Software Flush		HW Flush
Hot	Cold	Single	Double	
430 (±7.0)	1,180 (±1.0)	12,099 (±52)	51,876 (±256)	1,502 (±0.9)

320 cycles overhead per context switch

Clk @1GHz, CS @1KHz: **+ 0.032%**

So What Are the Costs?

Context Switch Latency

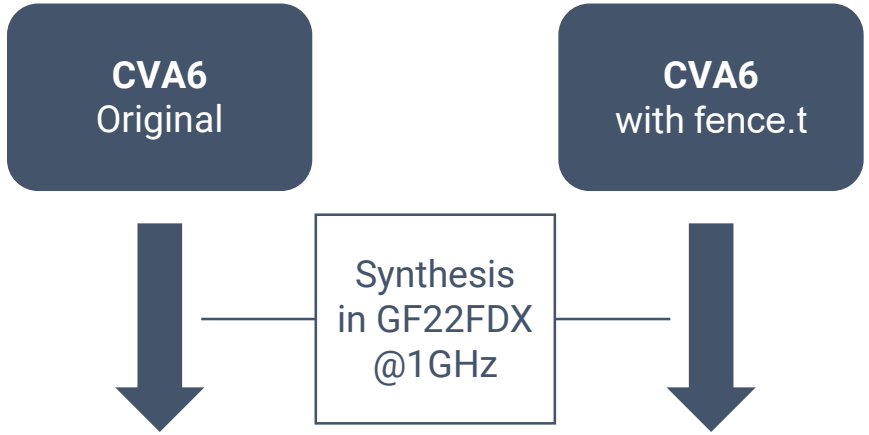
seL4 one-way inter-address-space IPC microbenchmark

Unmitigated		D$ Software Flush		HW Flush
Hot	Cold	Single	Double	
430 (±7.0)	1,180 (±1.0)	12,099 (±52)	51,876 (±256)	1,502 (±0.9)

320 cycles overhead per context switch

Clk @1GHz, CS @1KHz: **+ 0.032%**

Hardware Costs



Difference in area **less than 0.6%**

Conclusion

- **Covert channels exist on RISC-V cores**
  - We measure five distinct channels on Ariane
- **Confirmed: OS needs HW-support for time protection [1]**
  - Pure SW solutions cannot be comprehensive
- **First HW platform with (experimental) support for time protection!**
  - We propose a temporal fence (`fence.t`) instruction
  - Closes all evaluated channels at negligible costs
- **HW-mechanism must flush *all* μArch state**
  - Identifying μArch state not always straight-forward
  - Systematic approach for HW / Security codesign needed
- **Future Work**
  - Evaluate on write-back L1 data cache
  - Systematic evaluation of μArch state

Special Thanks to…

Moritz Schneider (ETH Zurich)

Curtis Millar (Data61)

Qian Ge (Data61)

Florian Zaruba (ETH Zurich)

Wolfgang Rönninger (formerly ETH Zurich)

Sascha Kegreiß (Hensoldt Cyber GmbH)

Frank K. Gürkaynak (ETH Zurich)

Rainer Leupers (RWTH Aachen)

Luca Benini (ETH Zurich and University of Bologna)

Gernot Heiser (UNSW Sydney and Data61 CSIRO)

Sources

[1] Qian Ge, Yuval Yarom, Tom Chothia, and Gernot Heiser: “Time Protection: The Missing OS Abstraction”, EuroSys, 2019

[2] R. E. Kessler and Mark D. Hill: “Page Placement Algorithm for Large Real-Indexed Caches”, ACM Trans. Comp. Syst. 19, 1992

[3] Wolfgang Rönninger: “Memory Subsystem for the First Fully Open-Source RISC-V Heterogeneous SoC”, Master’s thesis, ETH Zurich, 2019

[4] Florian Zaruba and Luca Benini: “The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology”, IEEE Trans. on VLSI Systems 27, 2019

[5] Gerwin Klein, June Andronick, Kevin Elphistone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser: “Comprehensive Formal Verification of an OS Microkernel”, ACM Trans. Comp. Syst. 32, 2014

[6] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Christiano Giuffrida: “RIDL: Rogue In-flight Data Load”, 2019

[7] “Two security flaws in modern chips cause big headaches for tech business”, The Economist, January 4, 2018

[8] Andy Greenberg: “A Critical Intel Flaw Breaks Basic Security for Most Computers”, Wired, January 3, 2018

[9] Samuel Gibbs: “Meltdown and Spectre: ‘worst ever’ CPU bugs affect virtually all computers”, The Guardian, January 4, 2018

[10] Tom Warren: “Intel’s processors have a security bug and the fix could slow down PCs”, The Verge, January 3, 2018

[11] Wikipedia, https://en.wikipedia.org/wiki/File:C-3PO_droid.png, accessed November 12, 2020

RISC-V® Summit

December 8-10 | Virtual Event

Thank you for joining us.

Contribute to the RISC-V conversation on social!

#RISCVSUMMIT @risc_v

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT