fence.t.s: Closing Timing Channels in High-Performance Out-of-Order Cores through ISA-Supported Temporal Partitioning

International Conference on Applications in Electronics Pervading Industry, Environment and Society (APPLEPIES), Turin, Italy, 2024-09-20

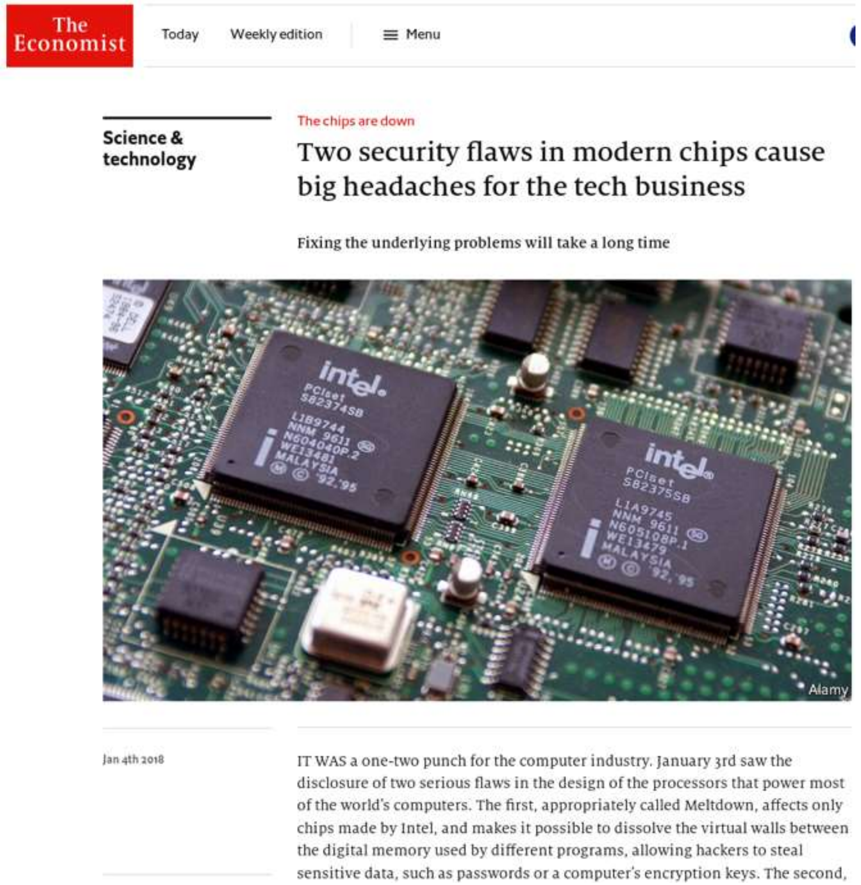
Nils Wistoff[1] nwistoff@iis.ee.ethz.ch
Gernot Heiser[2] gernot@unsw.edu.au
Luca Benini[1,3] lbenini@iis.ee.ethz.ch

[1] IIS, ETH Zurich, Switzerland
[2] UNSW Sydney, Australia
[3] DEI, University of Bologna, Italy

PULP Platform
Open Source Hardware, the way it should be!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

Spectre: Exploiting timing channels to leak data [1]


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

Alamy

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,

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.

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 GMT

SPECTRE

Speculative Execution

+

Timing Channel

THE VERGE TECH REVIEWS SCIENCE CREATORS ENTERTAINMENT VIDEO MORE

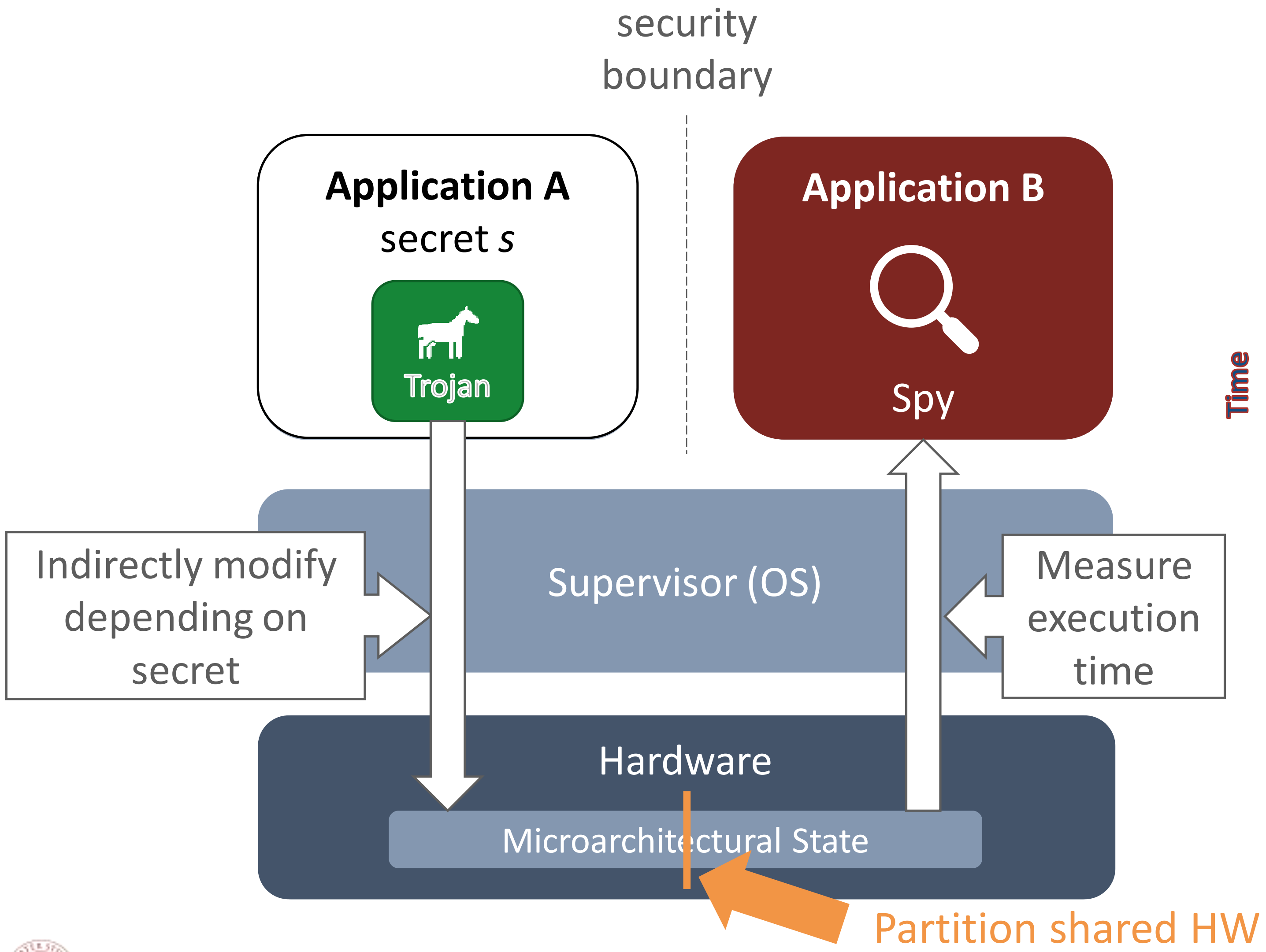
MICROSOFT TECH INTEL

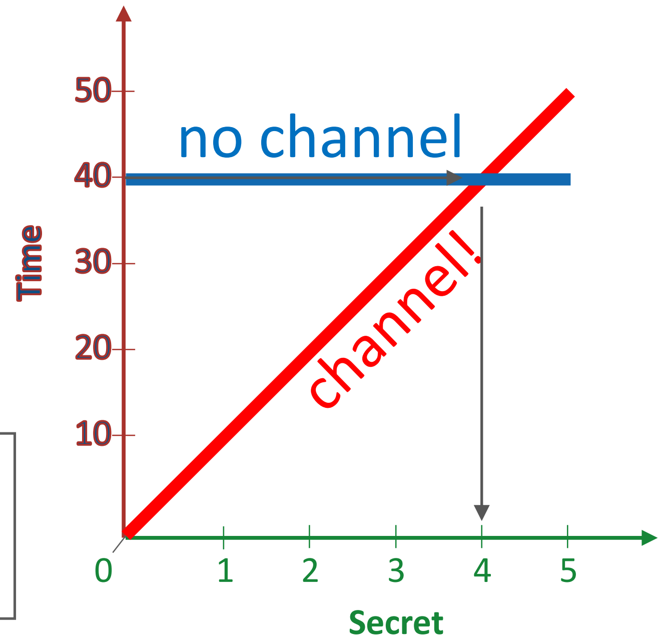
Intel's processors have a security bug and the fix could slow down PCs

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

[1] Kocher et al., *Spectre Attacks: Exploiting Speculative Execution*, IEEE S&P 2019

Timing Channel



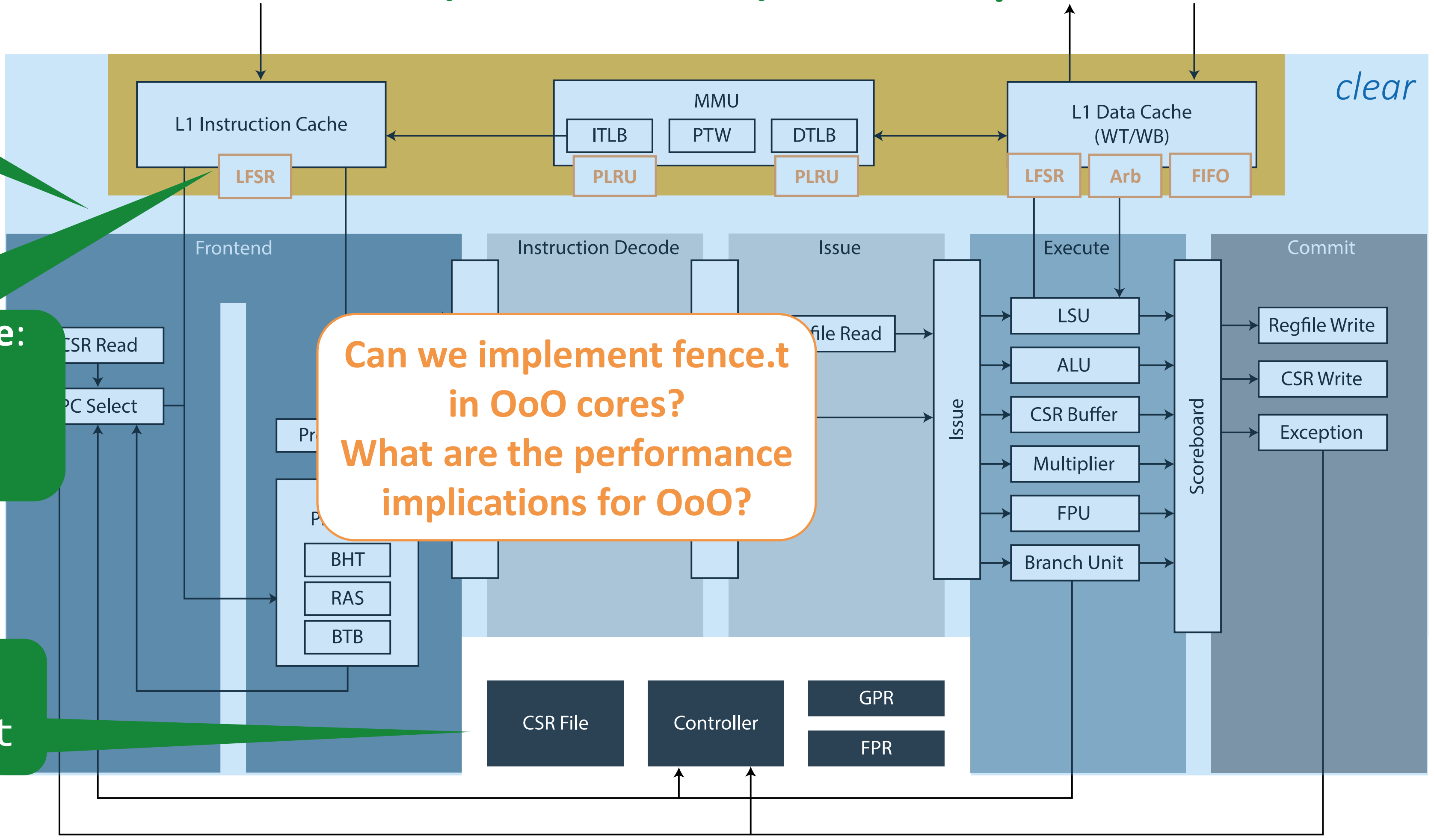


Temporal fence instruction (fence.t): Clear µArch [2]

CVA6: 6-stage, in-order RV64GC core

Microarchitectural state: Not directly visible or modifiable by SW. Cleared by `fence.t`

Architectural state: **Not cleared** by `fence.t`



Can we implement fence.t in OoO cores?
What are the performance implications for OoO?

[2] Wistoff et al., *Systematic Prevention of On-Core Timing Channels by Full Temporal Partitioning*, IEEE Trans. Comp. 2023

Contributions

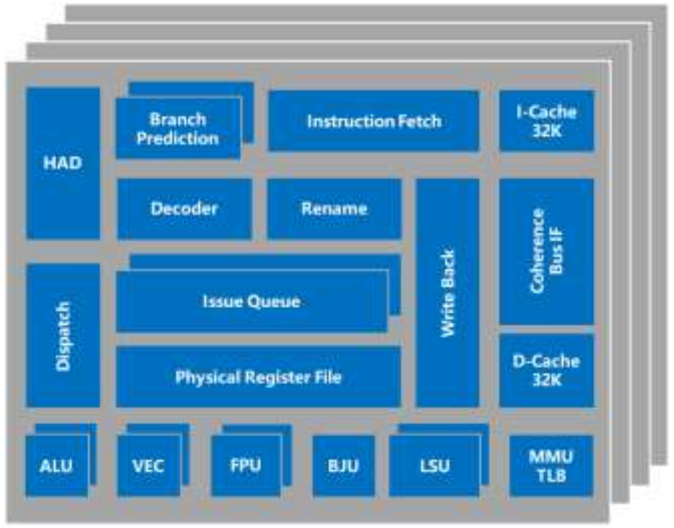
- We show that adding `fence.t` to **out-of-order** (OoO) cores comes with **new challenges**
- We present the **SW-supported temporal fence** (`fence.t.s`) to address these challenges.
- We **implement** `fence.t.s` in a fully-featured, open-sourced, commercial, high-performance 64-bit RISC-V core, namely **OpenC910**.
- We show that `fence.t.s` **closes all on-core timing channels** in OpenC910 without measurable HW overhead at a **performance overhead of 1.0 %**.

T-Head OpenC910 [3]

- T-Head Semiconductor Co., Ltd.
- 64-bit, application-class, **12-Stage**, **superscalar**, **Out-of-Order**, **RISC-V** RV64GC**Xtheadc** core.
- Verilog RTL **open-sourced** in 2021 under the Apache license.

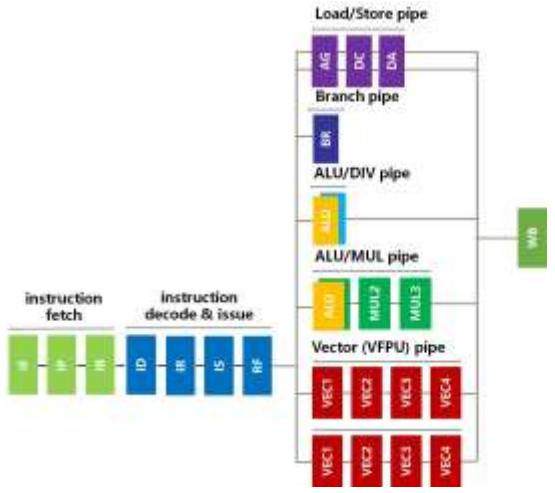
Operating Frequency	2.0-2.5 GHz (12nm FinFET)
Pipeline stages	12
ROB	up to 192 instructions
Decode width	3
Issue width/FUs	8

[3]



[3]

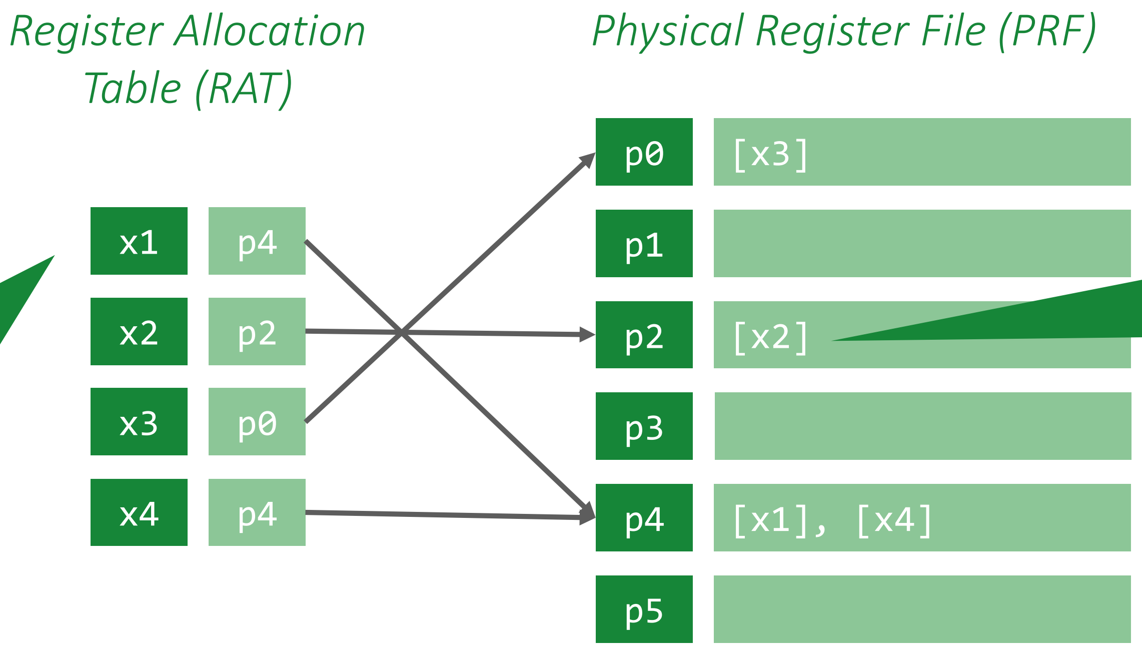
github.com/XUANTIE-RV/openc910



[3]

[3] Chen et al., *Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension*, ACM/IEEE ISCA 2020

Challenge 1: Mixed state



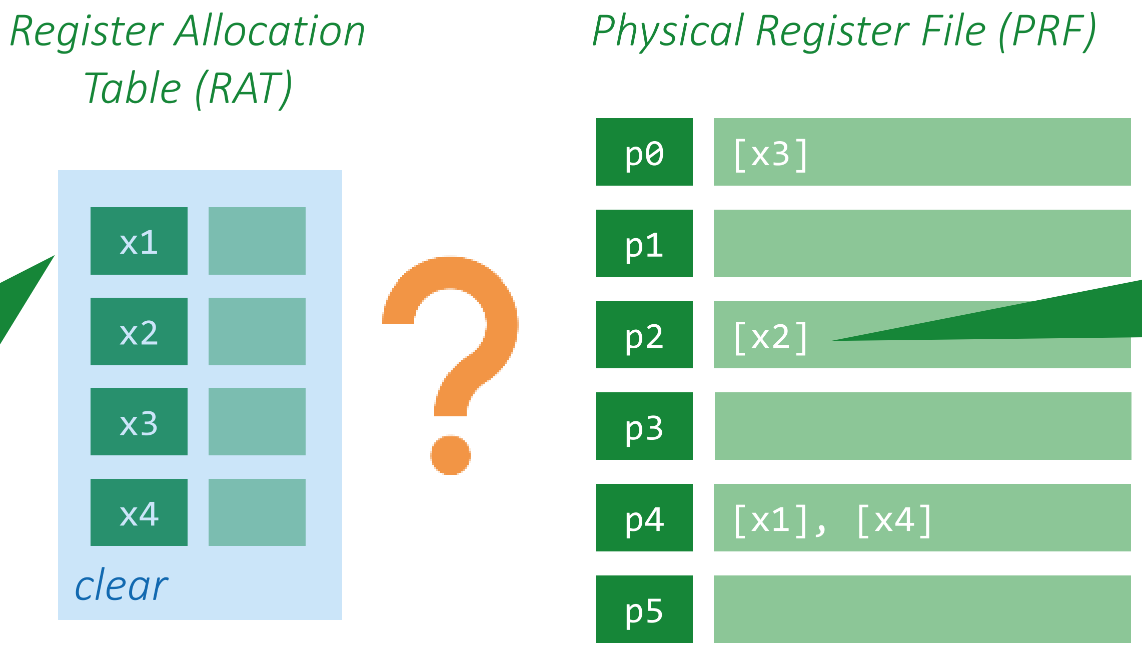
Register Allocation Table (RAT)
Physical Register File (PRF)
x1
p4
x2
p2
x3
p0
x4
p4
p0
[x3]
p1
p2
[x2]
p3
p4
[x1], [x4]
p5

Cannot be directly accessed by SW
→ **Microarchitectural** state
→ **clear** on temporal fence

Registers contain **architectural** state
→ keep on temporal fence

Challenge 1: Mixed state

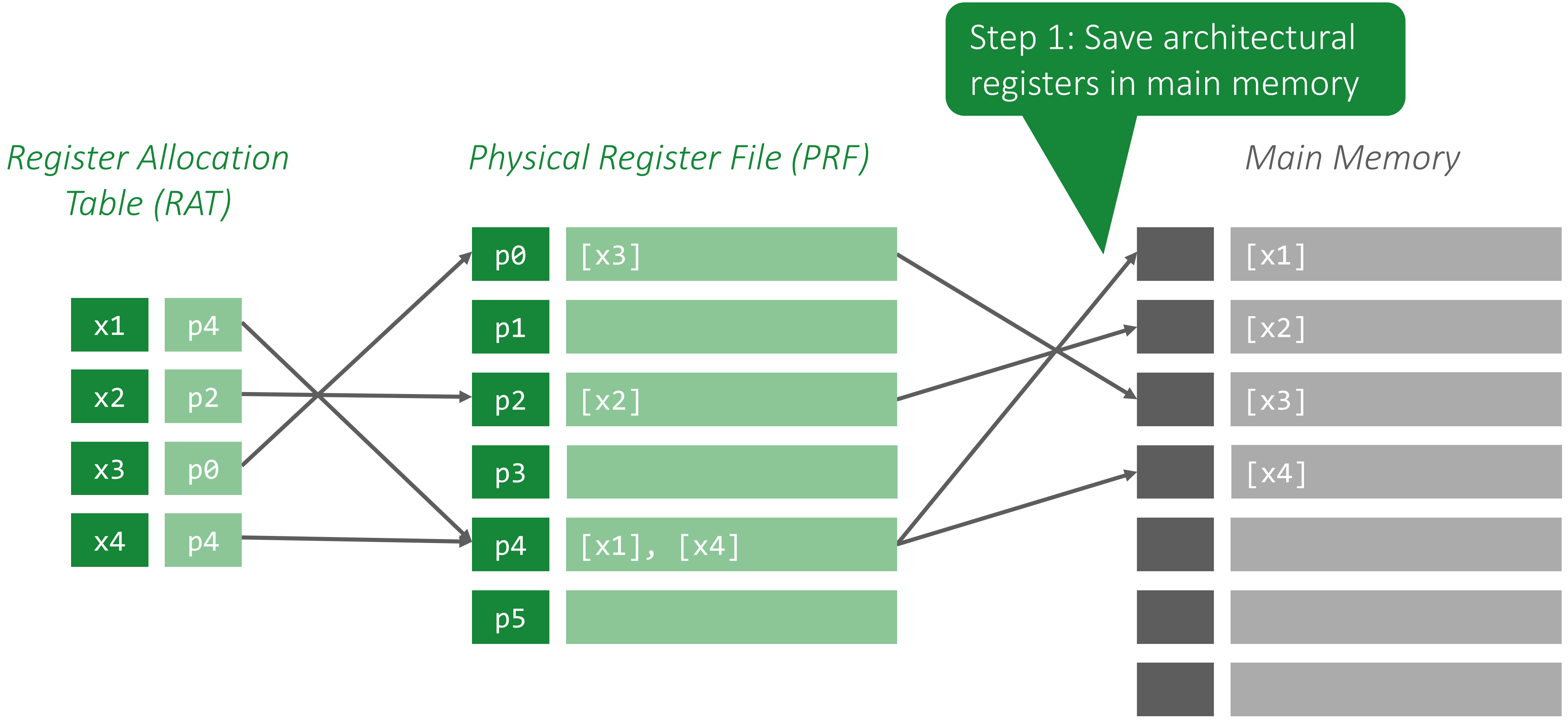
Cannot be directly accessed by SW
→ **Microarchitectural** state
→ **clear** on temporal fence



Registers contain **architectural** state
→ keep on temporal fence

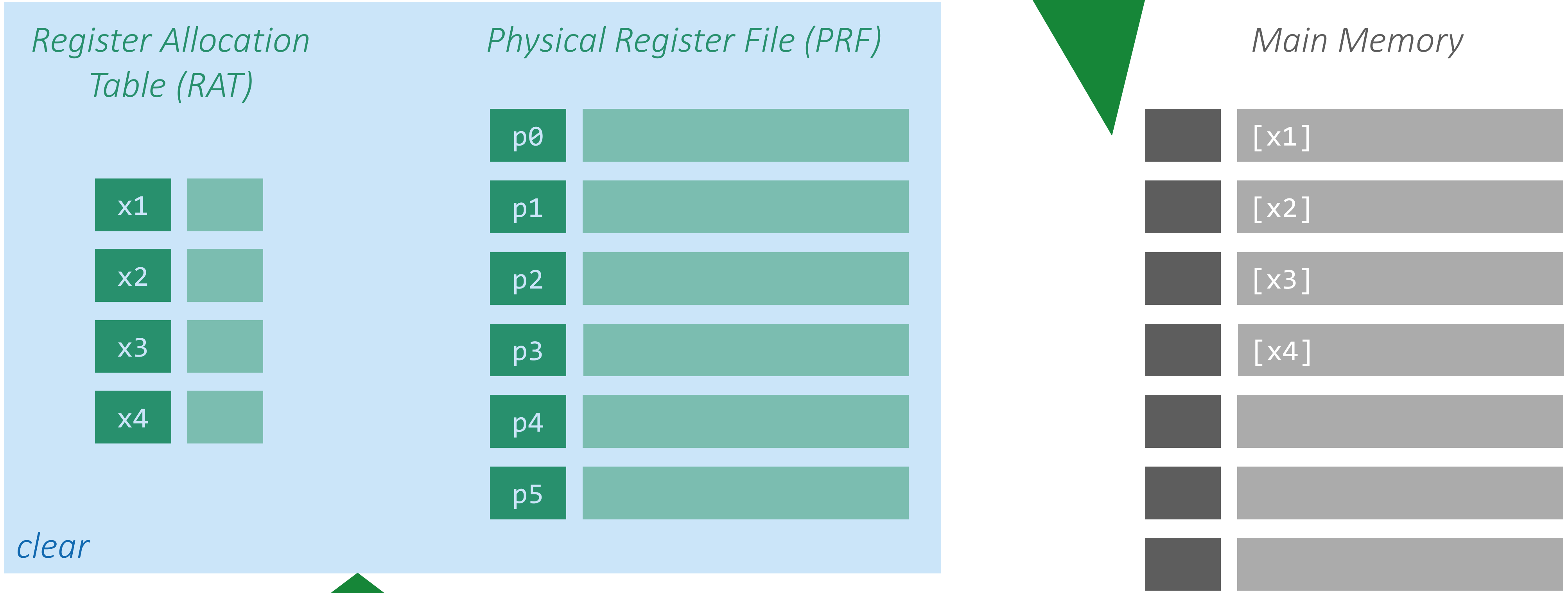
Problem: *Where* to find our architectural state?

Solution 1: Save context before clearing


Step 1: Save architectural registers in main memory
Register Allocation Table (RAT)
Physical Register File (PRF)
Main Memory
x1
p4
x2
p2
x3
p0
x4
p4
p0
[x3]
p1
p2
[x2]
p3
p4
[x1], [x4]
p5
[x1]
[x2]
[x3]
[x4]

Solution 1: Save context before clearing

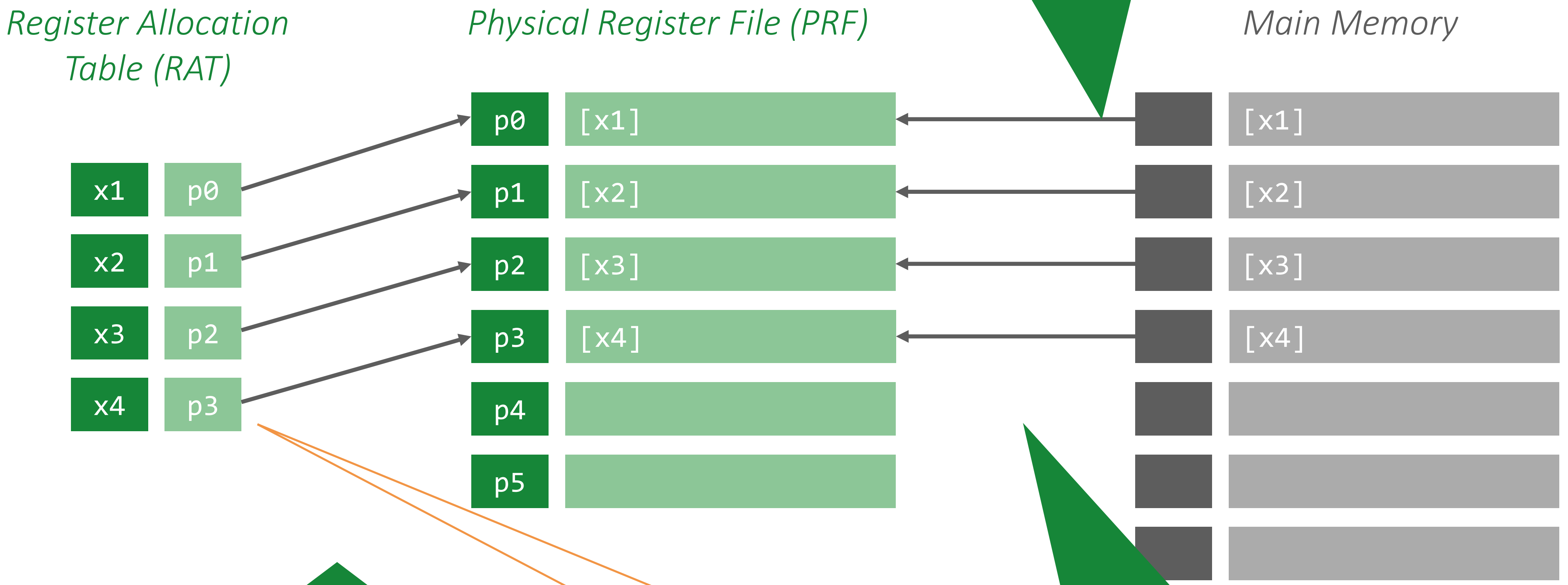
Step 1: Save architectural registers in main memory



Step 2: Clear **all** state including RAT and PRF

Solution 1: Save context before clearing

Step 1: Save architectural registers in main memory



Step 2: Clear **all** state including RAT and PRF

RAT state is now history-independent!

Step 3: Load architectural registers from main memory

Challenge 2: Reusability of Instructions

- One **monolithic** temporal fence instruction has several **disadvantages**
  - **No reuse** of the implemented mechanisms.
  - **No fine-grain control** over which components to clear (security-performance tradeoff).
  - **Complex control logic**.

- Solution: Split temporal fence into **multiple instructions**.

SW-supported temporal fence (`fence.t.s`)

Save architectural registers to main memory

Save stack pointer to a well-known, protected location

Write back dirty L1D state

Invalidate SRAMs

Clear flip-flops

Restore stack pointer

Restore architectural registers

Pad execution time to prevent leakage through context-switch latency

```
for all r ∈ ArchRegs do
        stack ← r
end for

scratch ← sp

ClearL1D

InvalSRAMs

ClearFFs

sp ← scratch

for all r ∈ ArchRegs do
        r ← stack
end for

PadTime
```

ETH zürich   ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

fence.t.s in OpenC910

Standard RISC-V

```
addi sp, sp, -31*8
csrw sscratch, sp
sd   x1, 0*8(sp)
…
sd   x31, 30*8(sp)
```

Xtheadc

```
dcache.call
li   t0, 0x70011
csrs mcor, t0
```

```
la   t0, post_ff_clr
csrw mrvbr, t0
sync.i
ff.clr
post_ff_clr:
sync.i
```

New instruction: Clear all on-core FFs except CSRs

Standard RISC-V

```
csrr sp, sscratch
ld   x1, 0*8(sp)
…
ld   x31, 30*8(sp)
addi sp, sp, 31*8
```

```
for all r ∈ ArchRegs do
      stack ← r
end for

scratch ← sp

ClearL1D

InvalSRAMs

ClearFFs

sp ← scratch

for all r ∈ ArchRegs do
      r ← stack
end for

PadTime
```

ETH zürich
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Experimental setup

Channel bench [4]

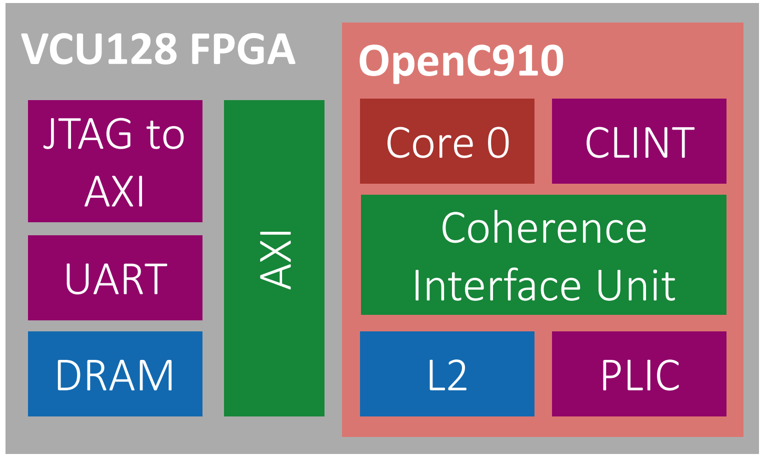
Application

+

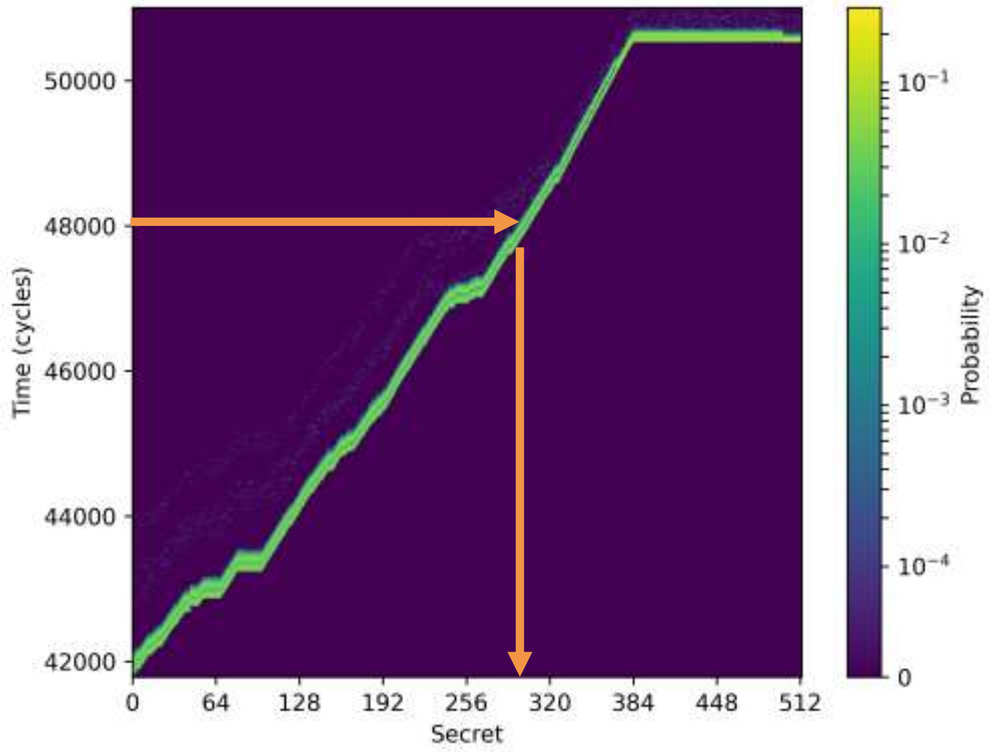
OS

+

HW Platform



N = 10^6



M = 4283 mb

M_0 = 0.7 mb

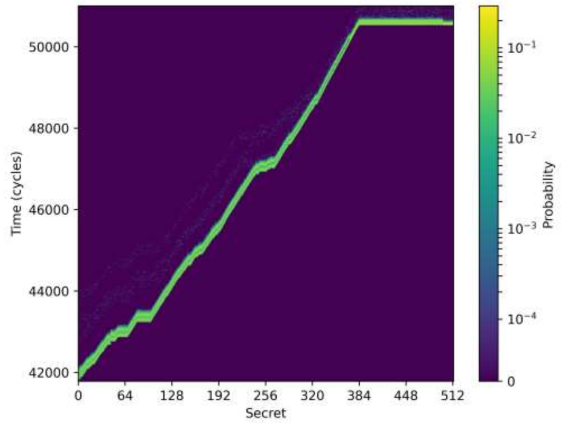
Characterises noise

[4] Ge et al., *No security without time protection: We need a new hardware-software contract*, ACM APSys 2018

`fence.t.s` closes all timing channels!

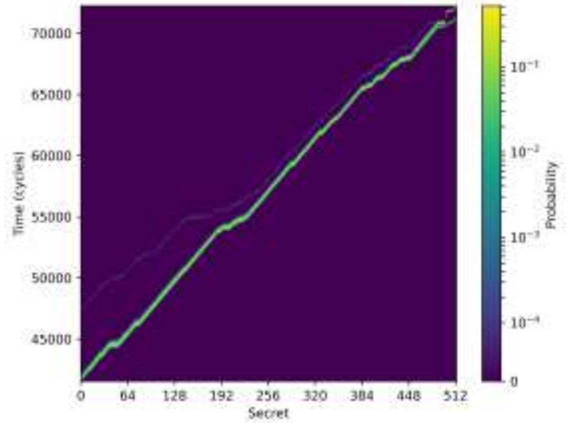
unmitigated

L1D



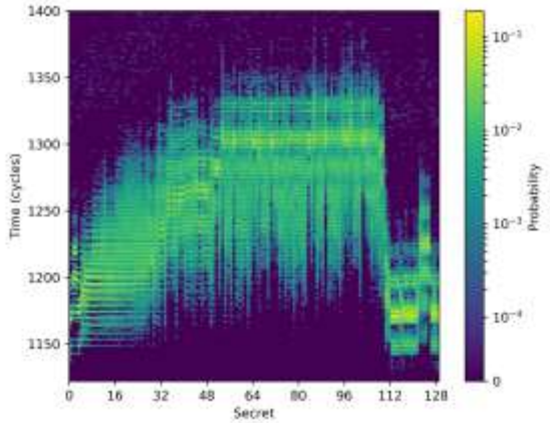
M = **4283 mb**, M_0 = 0.7 mb

L1I



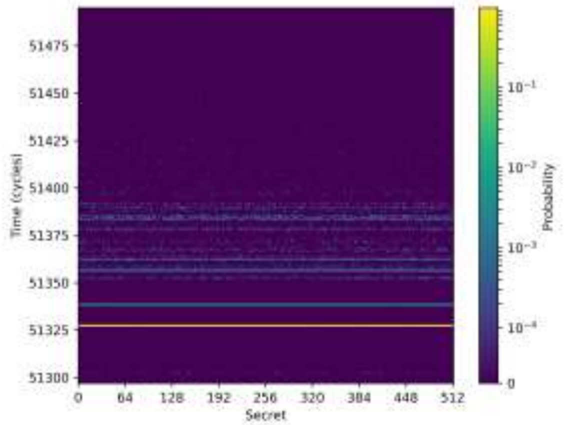
M = **5940 mb**, M_0 = 0.8 mb

BHT

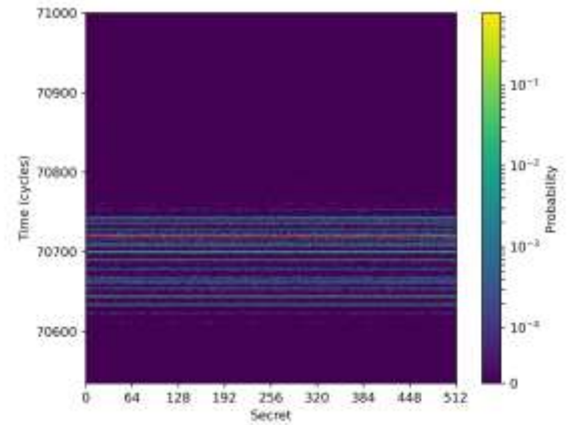


M = **698.8 mb**, M_0 = 1.1 mb

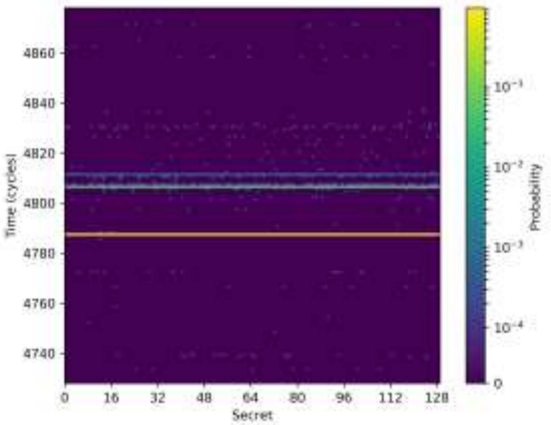
`fence.t.s`



M = 64.3 mb, M_0 = 71.0 mb



M = 3.2 mb, M_0 = 3.5 mb



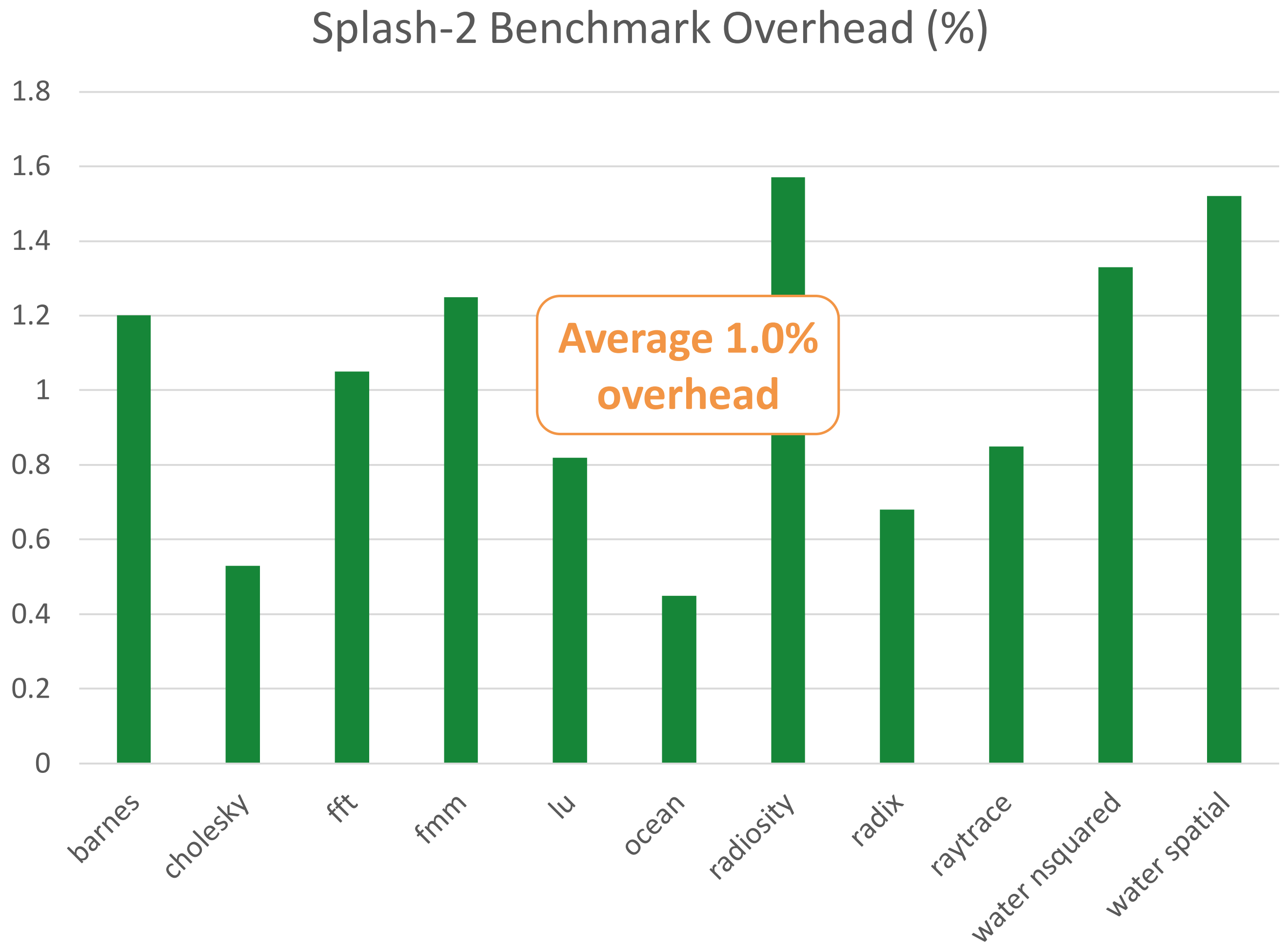
M = 3.3 mb, M_0 = 6.4 mb

`fence.t.s` is inexpensive!

- 2 threads: 1 benchmark + 1 idle
- 1GHz system clock
- 10ms timeslice
- Context switch every 10M cycles

- Synthesis in GF12 LP+ FinFET @2GHz

Negligible hardware costs



Conclusion

- We **presented the SW-supported temporal fence** (`fence.t.s`) methodology to close timing channels **even in high-performance out-of-order cores**.
- We showed that `fence.t.s` can reliably **close all observed timing channels** in OpenC910.
- We showed that `fence.t.s` comes at a **negligible HW overhead** and a **minimal performance overhead** of 1.0 %.
- We found that **OpenC910 already provides most mechanisms** hat are required to enable time protection. Specifying them would enable time protection **across implementations**.

RISC-V®

uSC SIG
Timing Fences TG

Nils Wistoff	nwistoff@iis.ee.ethz.ch
Gernot Heiser	gernot@unsw.edu.au
Luca Benini	lbenini@iis.ee.ethz.ch

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI – Università di Bologna
Viale del Risorgimento 2
Bologna, Italy

Q&A

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

Case study: T-Head OpenC910 Xtheadc extension

- Custom instructions:
  - `dcache.call`: **clear** the **L1 data cache**.
  - `sync.i`: **execution barrier** in the instruction stream.
  - …
- Custom control and status registers (CSRs):
  - `mcor`: **invalidate** selected **SRAMs** in the core.
  - `mrvbr`: **reset address**.
  - …

`github.com/XUANTIE-RV/openc910`

[3] Chen et al., *Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension*, ACM/IEEE ISCA 2020