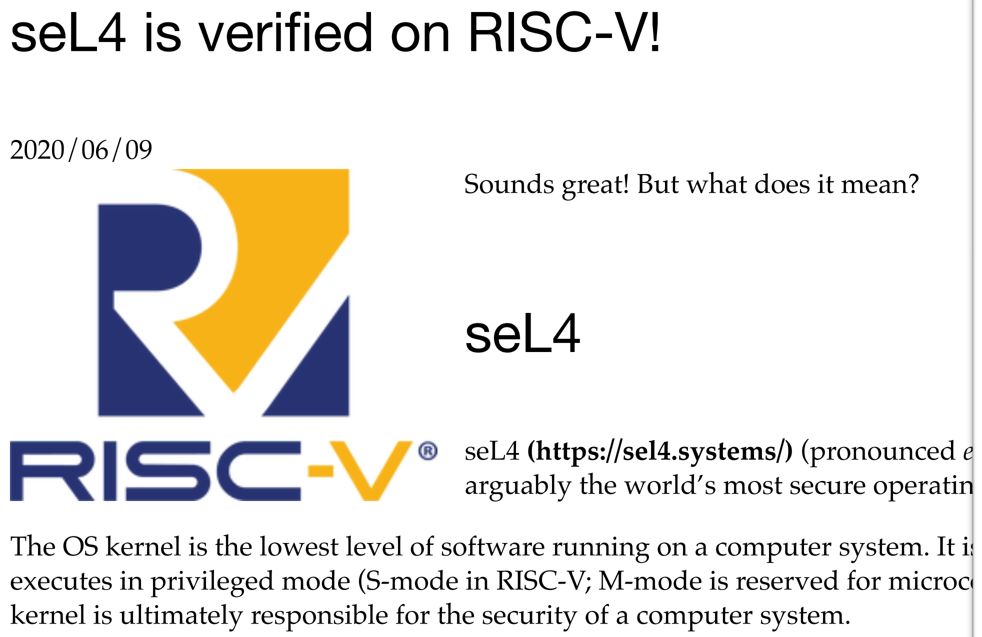
The seL4® Report

aka State of the Union

Gernot Heiser, Trustworthy Systems Group

https://sel4.systems/

The Highlights of the Year



seL4 is verified on RISC-V!

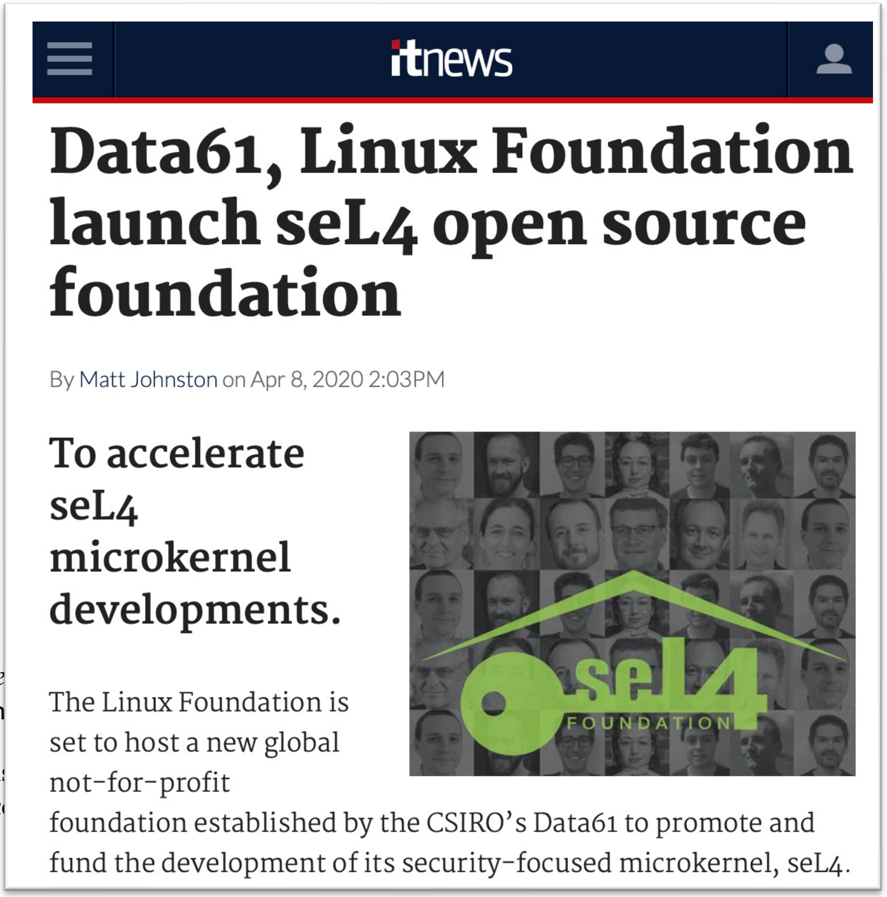
2020/06/09

Sounds great! But what does it mean?

seL4

seL4 **(https://sel4.systems/)** (pronounced *e*
arguably the world's most secure operatin

The OS kernel is the lowest level of software running on a computer system. It i
executes in privileged mode (S-mode in RISC-V; M-mode is reserved for microc
kernel is ultimately responsible for the security of a computer system.



itnews

Data61, Linux Foundation launch seL4 open source foundation

By Matt Johnston on Apr 8, 2020 2:03PM

To accelerate seL4 microkernel developments.

The Linux Foundation is set to host a new global not-for-profit foundation established by the CSIRO's Data61 to promote and fund the development of its security-focused microkernel, seL4.

Major Developments in seL4 Land

- The seL4 Foundation (June's talk on Wednesday)
- seL4 on RISC-V (RV64) functional correctness proof done!
- Interim Endorsements for Trusted Service Providers and Training (June's talk on Wed)
- seL4 White Paper: https://sel4.systems/About/seL4-whitepaper.pdf
- UNSW Advanced Operating Systems teaching videos released
- Trademark registration in Australia and US
- RV64 binary verification (translation correctness) progressing
- **MCS kernel** verification progressing
- Draft seL4 Core Platform (my talk on Wednesday)
- seL4 Device Driver Framework (Ihor's talk on Wednesday)
- **Research: Verifying Time Protection**
- Research: Secure Multiserver OS

Background: What is seL4?

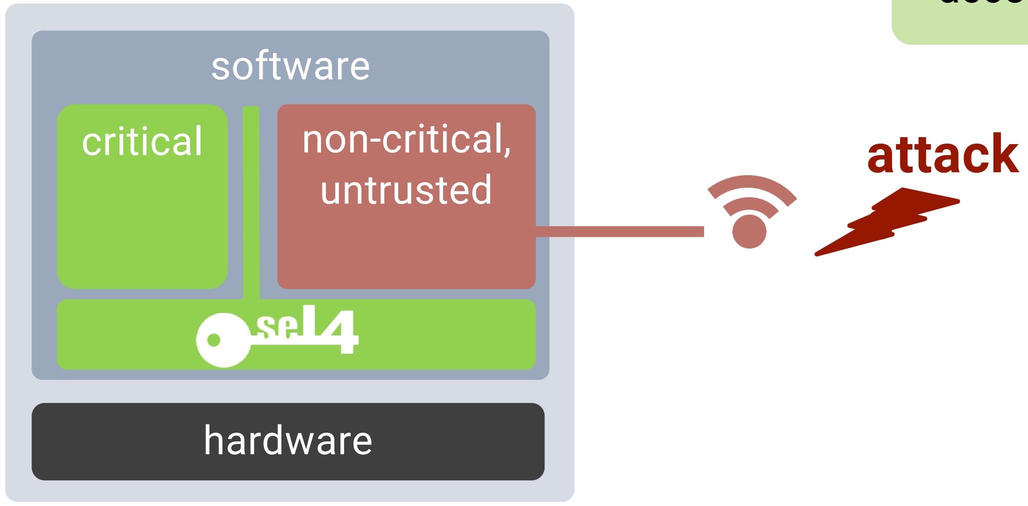
seL4 is an open source, high-assurance, high-performance operating system microkernel

Available on GitHub under GPLv2 license

World's most comprehensive mathematical proofs of correctness and security

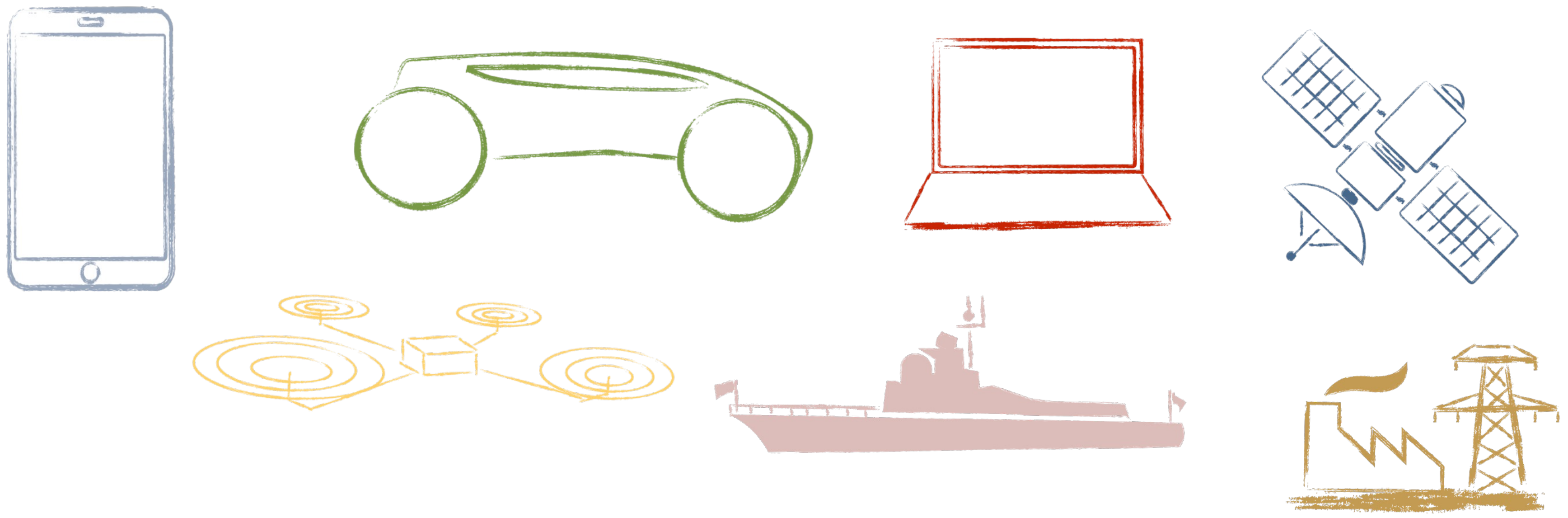
World's fastest microkernel

Piece of software that runs at the heart of any system and controls all accesses to resources



What is seL4?

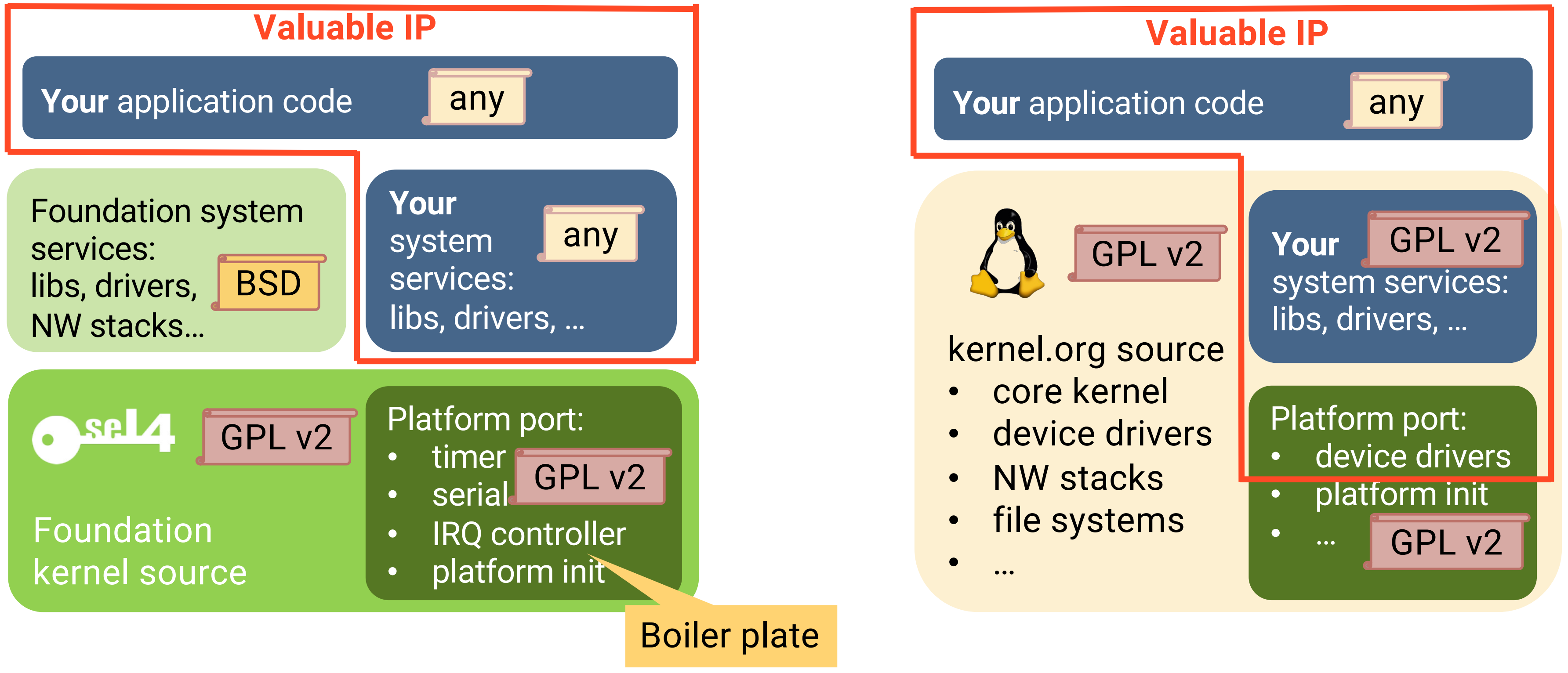
seL4 is the most trustworthy foundation for safety- and security-critical systems



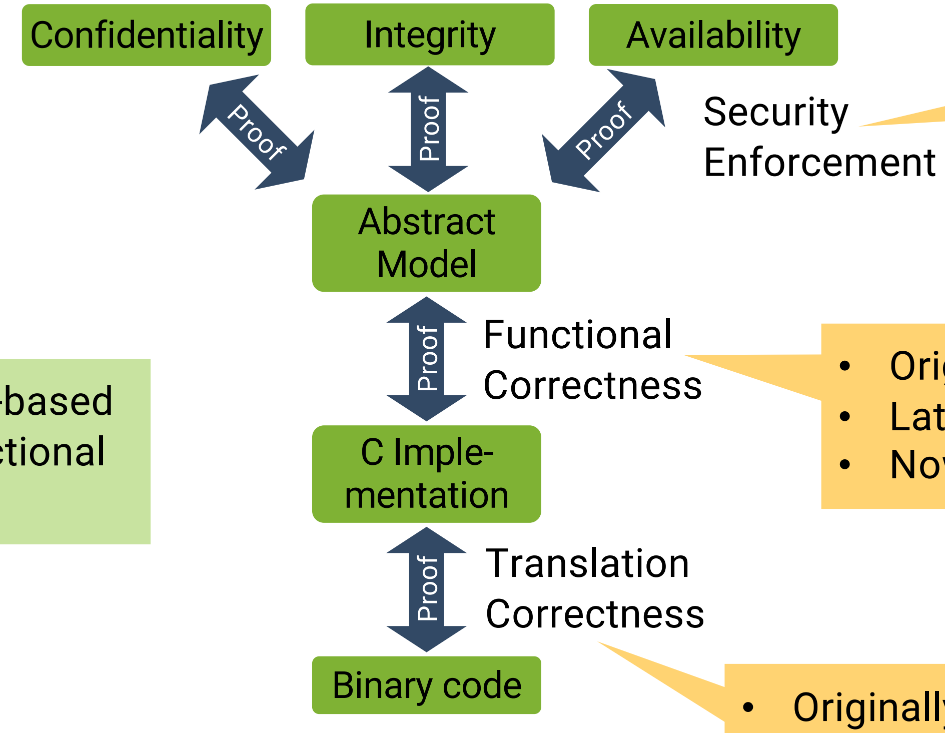
Already in use across many domains:

automotive, aviation, space, defence, critical infrastructure,
cyber-physical systems, IoT, industry 4.0, certified security...

Licensing: What Does the GPL Imply?



Uniqueness: Proofs



- Arm v6/7 (32-bit) so far

Still only capability-based OS kernel with functional correctness proof

- Originally Arm v6/7 (32-bit)
- Later x86 (64-bit)
- Now RISC-V (64-bit)

- Originally Arm v6/7 (32-bit)
- RISC-V (64-bit) in progress

… and Performance

Latency (in cycles) of a round-trip cross-address-space IPC on x64

Source	seL4	Fiasco.OC	Zircon
Mi et al, 2019	986	2717	8157
Gu et al, 2020	1450	3057	8151
seL4.systems, Nov'20	797	N/A	N/A

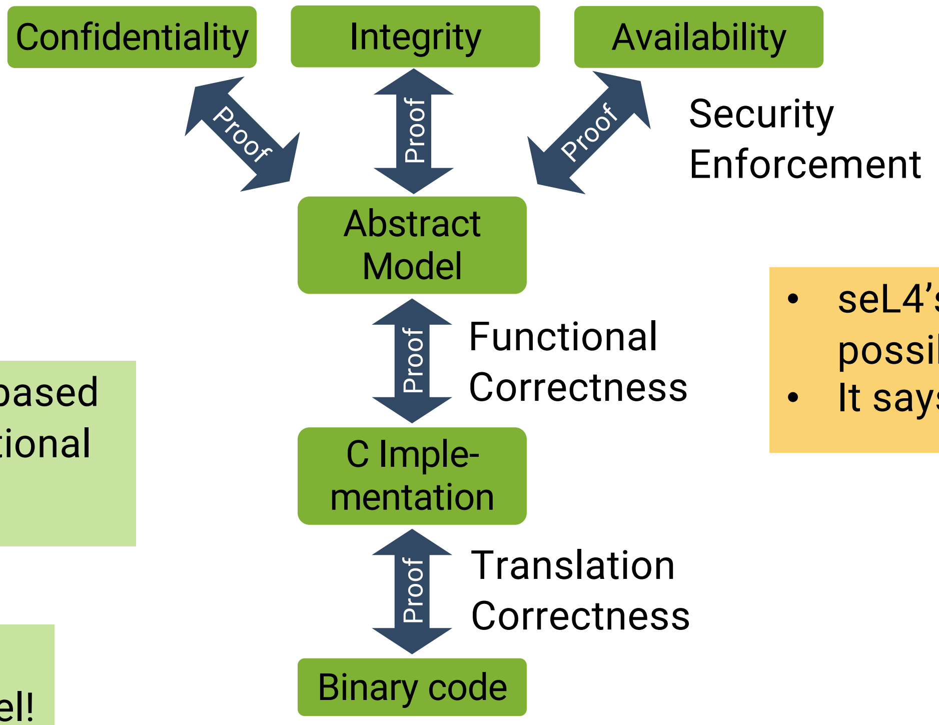
Still the world's fastest microkernel!

Temporary performance regression in Dec'19

Sources:

- Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, Haibo Chen: "SkyBridge: Fast and Secure Inter-Process Communication for Microkernels", EuroSys, April 2020
- Jinyu Gu, Xinyue Wu, Wentai Li, Nian Liu, Zeyu Mi, Yubin Xia, Haibo Chen: "Harmonizing Performance and Isolation in Microkernels with Efficient Intra-kernel Isolation and Communication", Usenix ATC, June 2020
- seL4 Performance, https://sel4.systems/About/Performance/, accessed 2020-11-08

So, Why Aren't We Done?



- seL4's verification provides the best possible guarantee of *spatial isolation*
- It says nothing about *temporal isolation*

Still only capability-based OS kernel with functional correctness proof

Still the world's fastest microkernel!

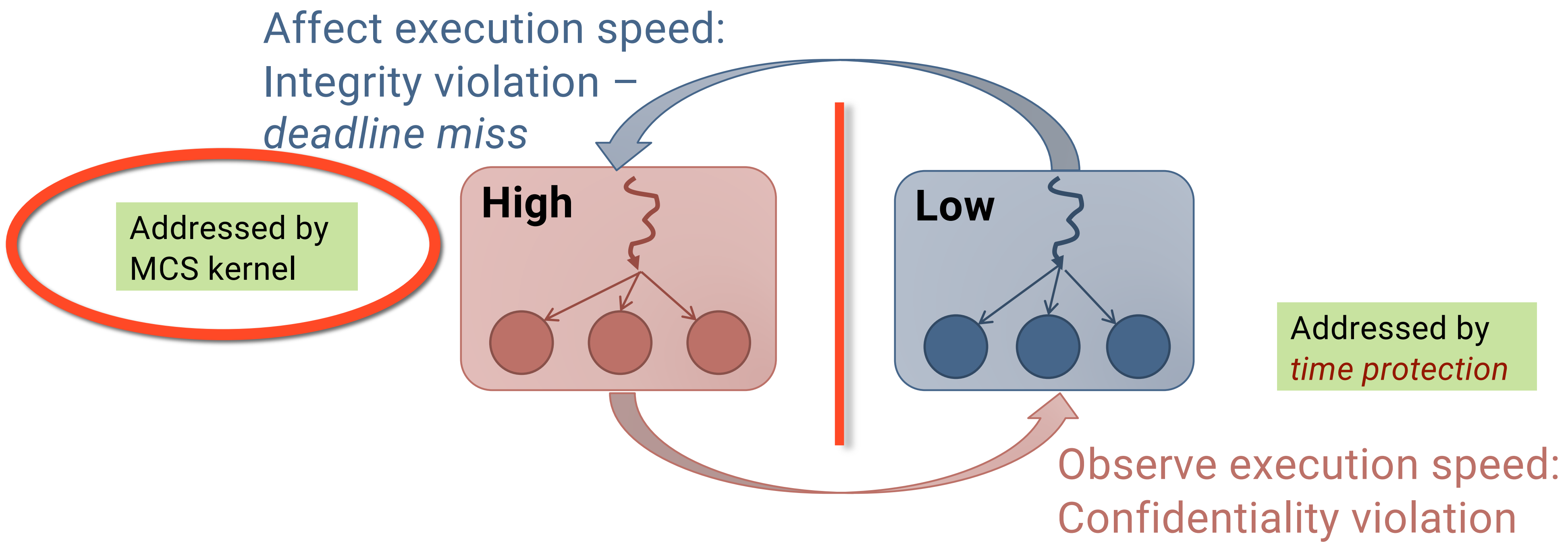
What’s the Issue with Temporal Isolation?

Safety: Timeliness

- Execution interference

Security: Confidentiality

- Leakage via timing channels


Affect execution speed:
Integrity violation – *deadline miss*
Addressed by MCS kernel
High
Low
Addressed by *time protection*
Observe execution speed:
Confidentiality violation

MCS Kernel: Capabilities for Time

Traditional seL4: Capabilities authorise access to spatial resources:
- Memory
- Threads
- Address spaces
- Communication endpoints
- Interrupts
- …

MCS model: Capabilities also authorise CPU time
- Scheduling objects

Scheduling Contexts

Classical thread attributes

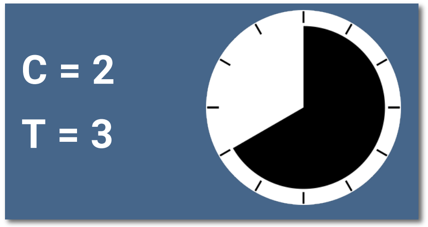
- Priority
- Time slice

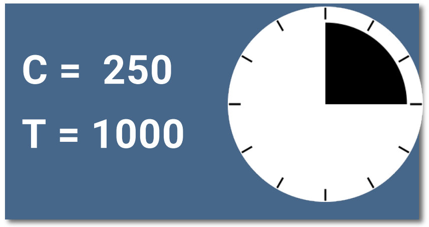
New thread attributes

- Priority
- Scheduling context capability

Scheduling context object
T: period
C: budget (≤ T)

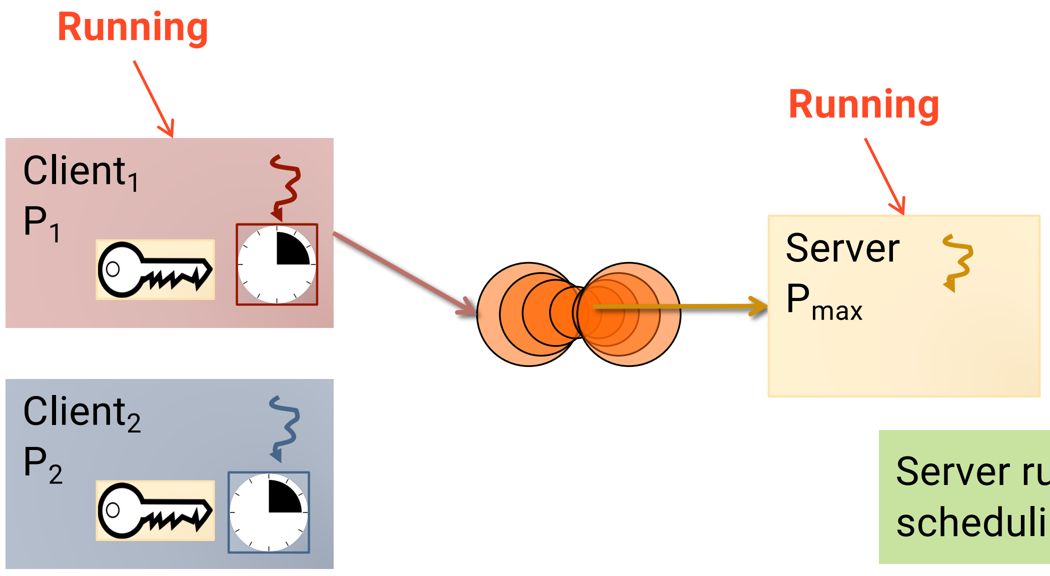
Scheduling-context object specifies CPU bandwidth limit





Ensure time available to lower-priority threads

Budget Donation



Client is charged for server's time

Server runs on client's scheduling context

Scheduling-context capabilities: a principled, light-weight OS mechanism for managing time [Lyons et al, EuroSys'18]

MCS Summary

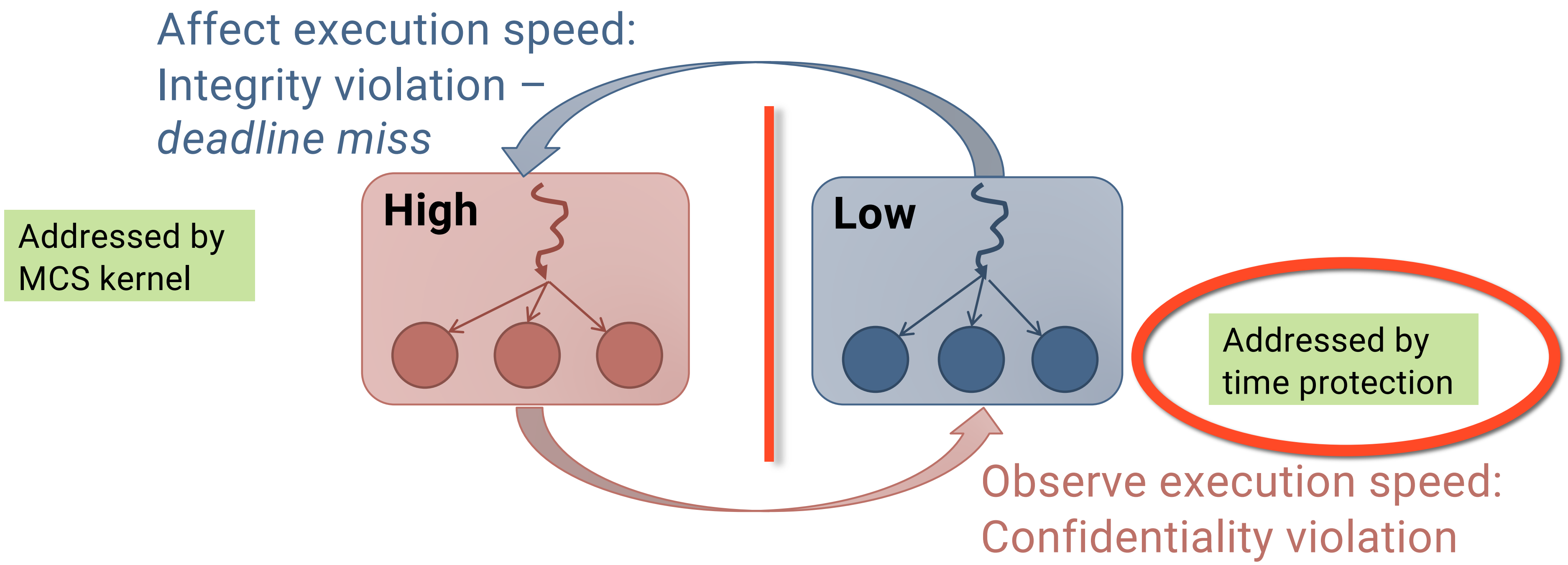
Generally much cleaner model,
cleans up a number of other things
⇒ Use for all new work!

- Verification getting close (Arm v7 and RV64)
- Legacy model will be *archived* once verification is done

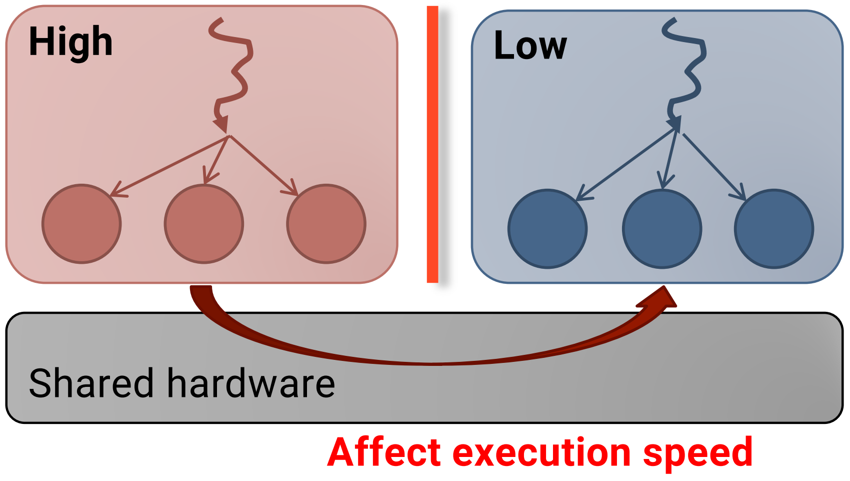
What's the Issue with Temporal Isolation?

Safety: Timeliness
- Execution interference

Security: Confidentiality
- Leakage via timing channels



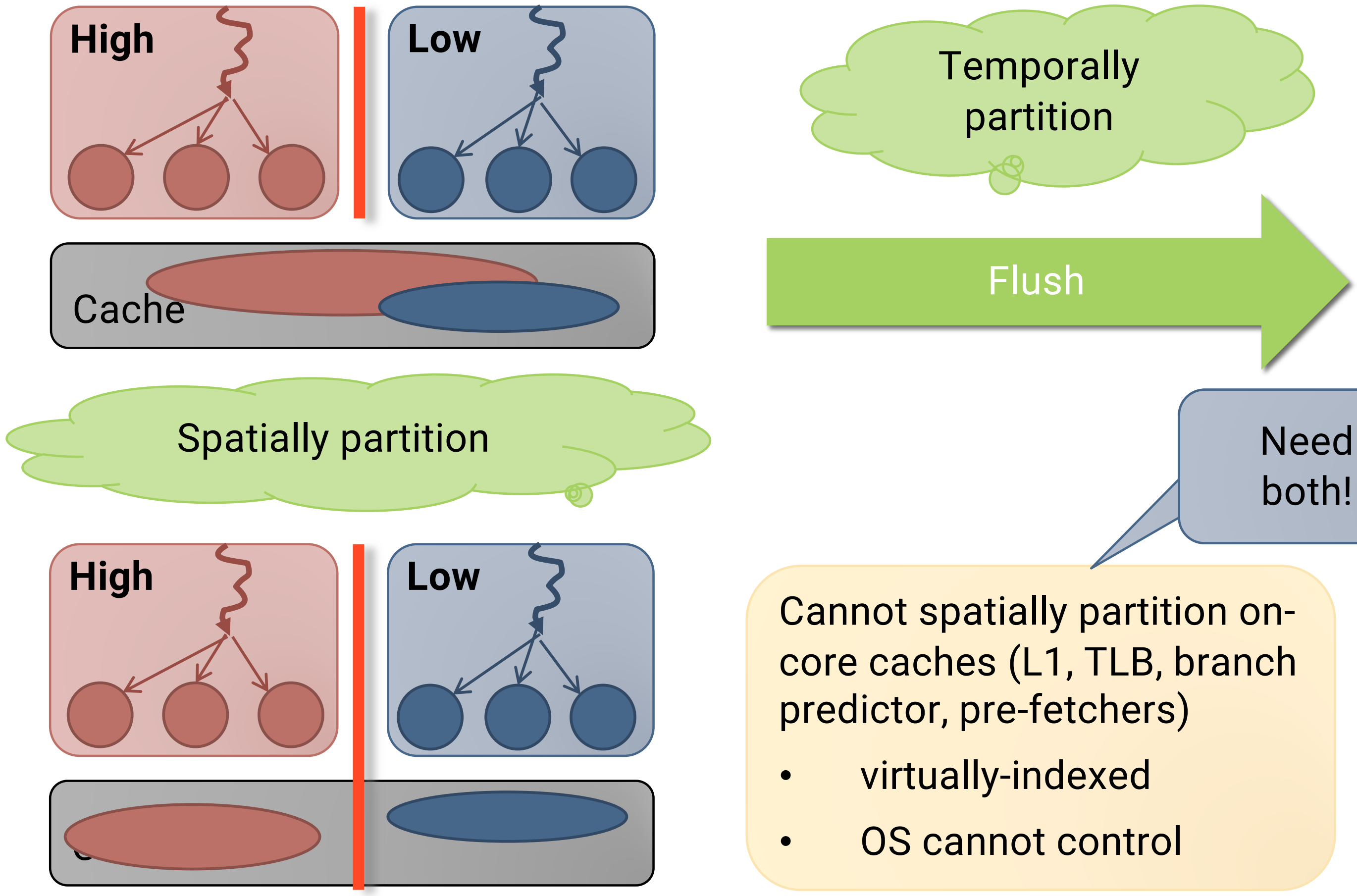
Cause: Competition for HW Resources



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

Solution: *Time Protection* – Eliminate interference by preventing sharing

Time Protection: Partition all Hardware State



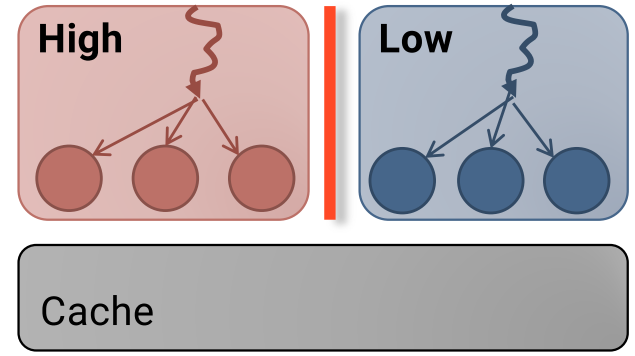
Need both!

Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)

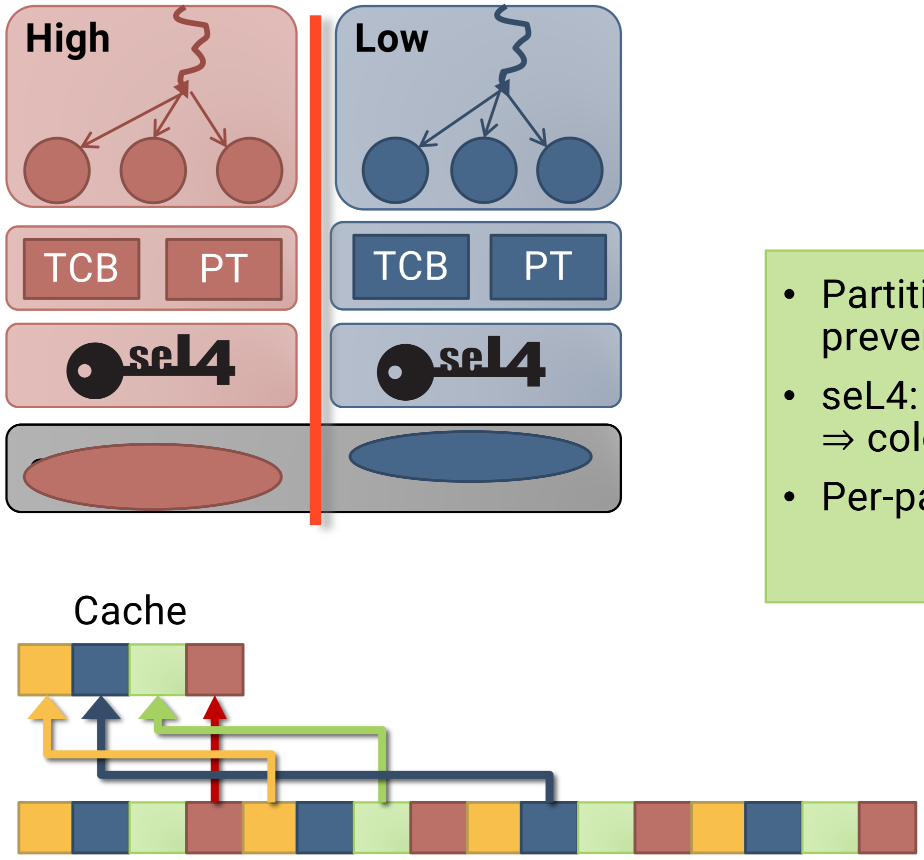
- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

- HW threads
- cores



Partition Hardware: Page Colouring



- Partitions get frames of disjoint colours preventing interference
- seL4: userland supplies kernel memory ⇒ colouring userland colours dynamic kernel memory
- Per-partition kernel image to colour kernel

[Ge et al. EuroSys'19]

Small amount of static kernel memory needs special handling

Temporal Partitioning: Flush on Switch

Must remove any history dependence!

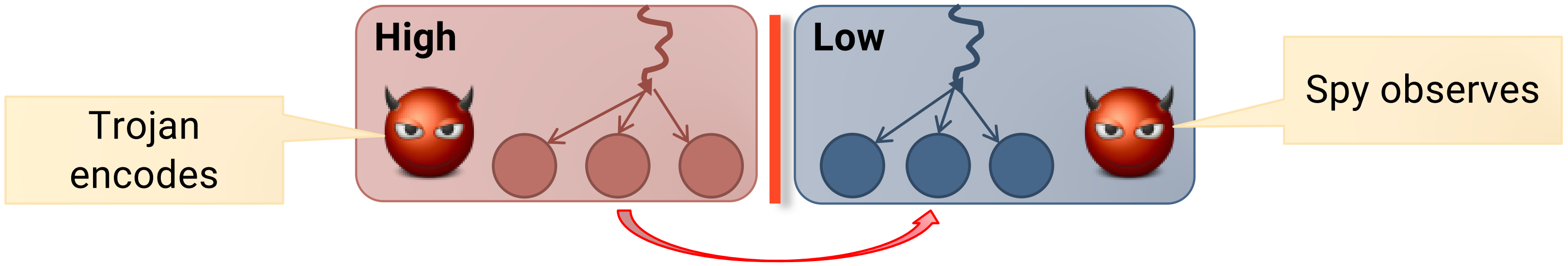
1. T_0 = current_time()
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while (T0+WCET < current_time()) ;
6. Reprogram timer
7. return

Latency depends on prior execution!

Ensure deterministic execution

Time padding to remove dependency

Evaluation: Prime & Probe Attack



1. Fill cache with own data

2. Touch *n* cache lines

Input signal

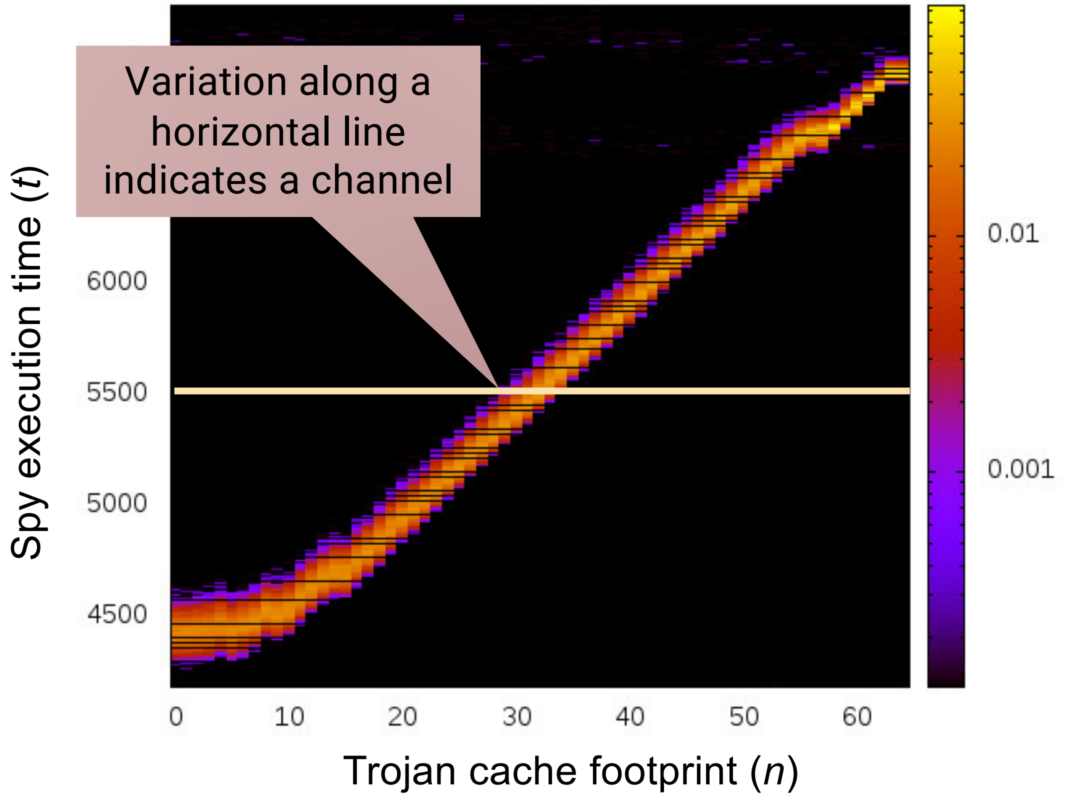
2.

3. Traverse cache,
   measure *execution time*

Output signal

Methodology: Channel Matrix

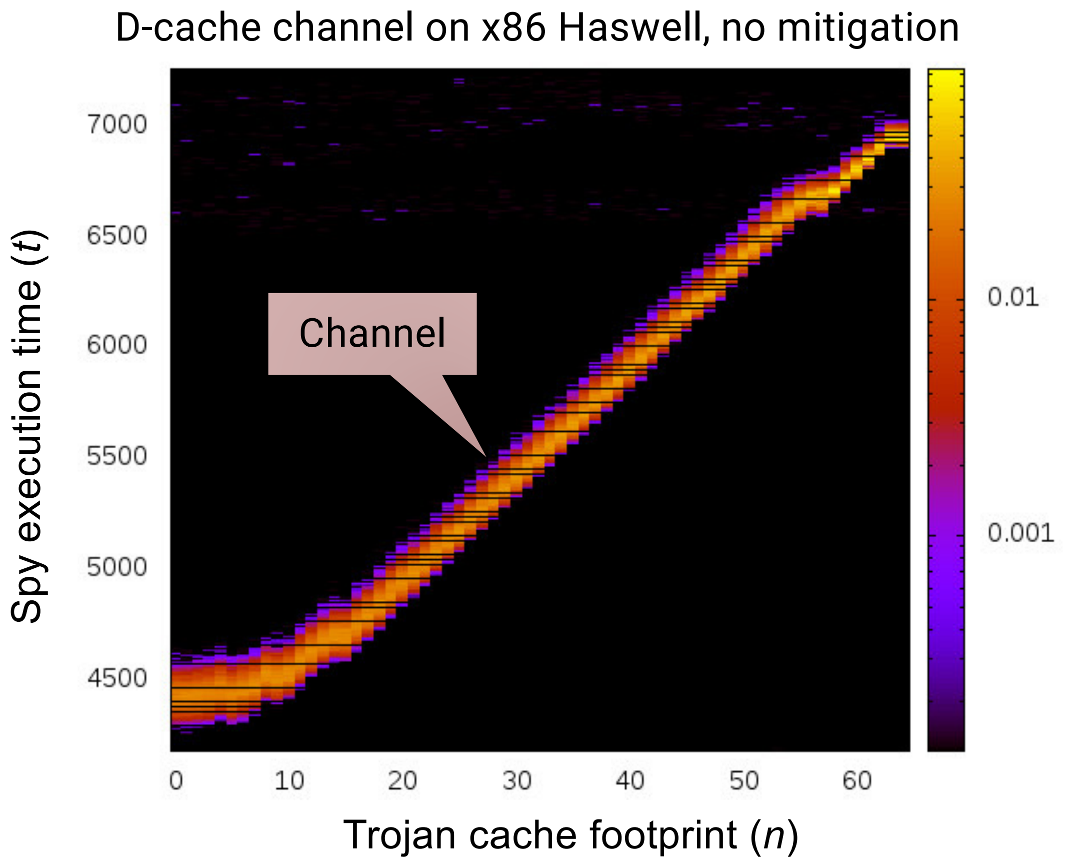
D-cache channel on x86 Haswell, no mitigation

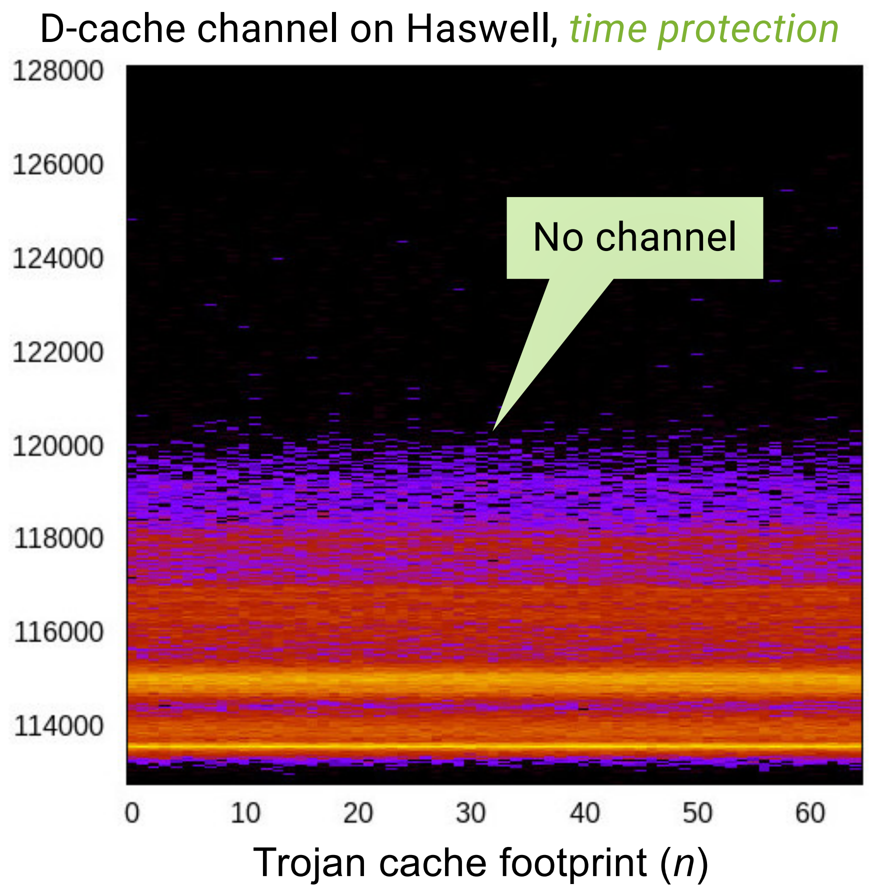


Channel matrix:

- Conditional probability of observing output signal (t), given input (n)
- Represented as heat map:
  - bright: high probability
  - dark: low probability

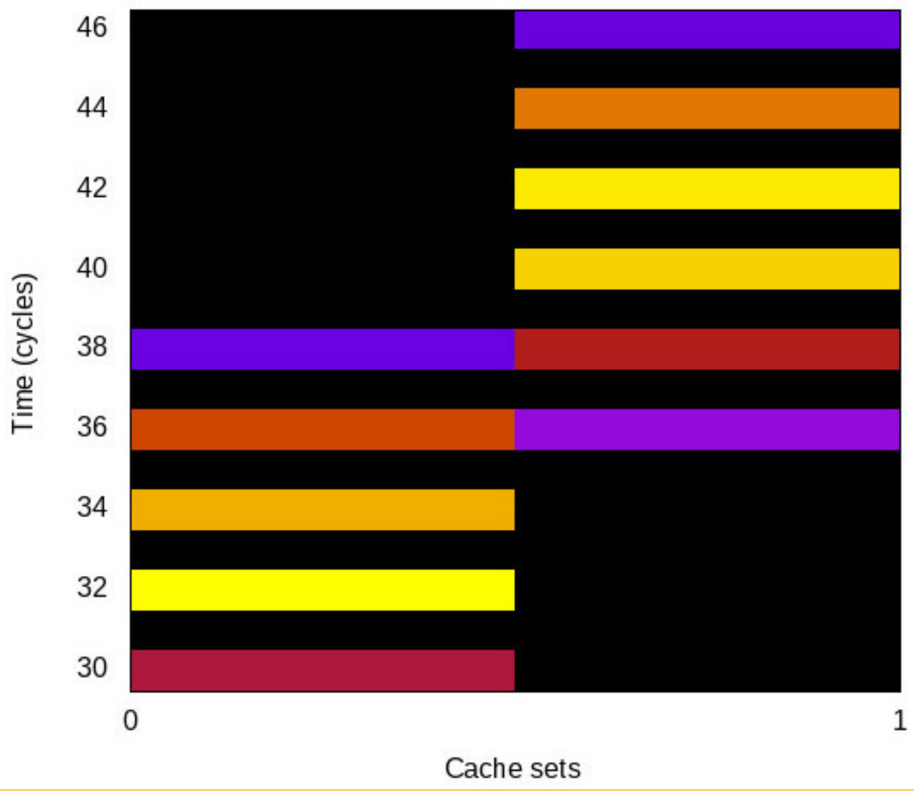
Applying Time Protection



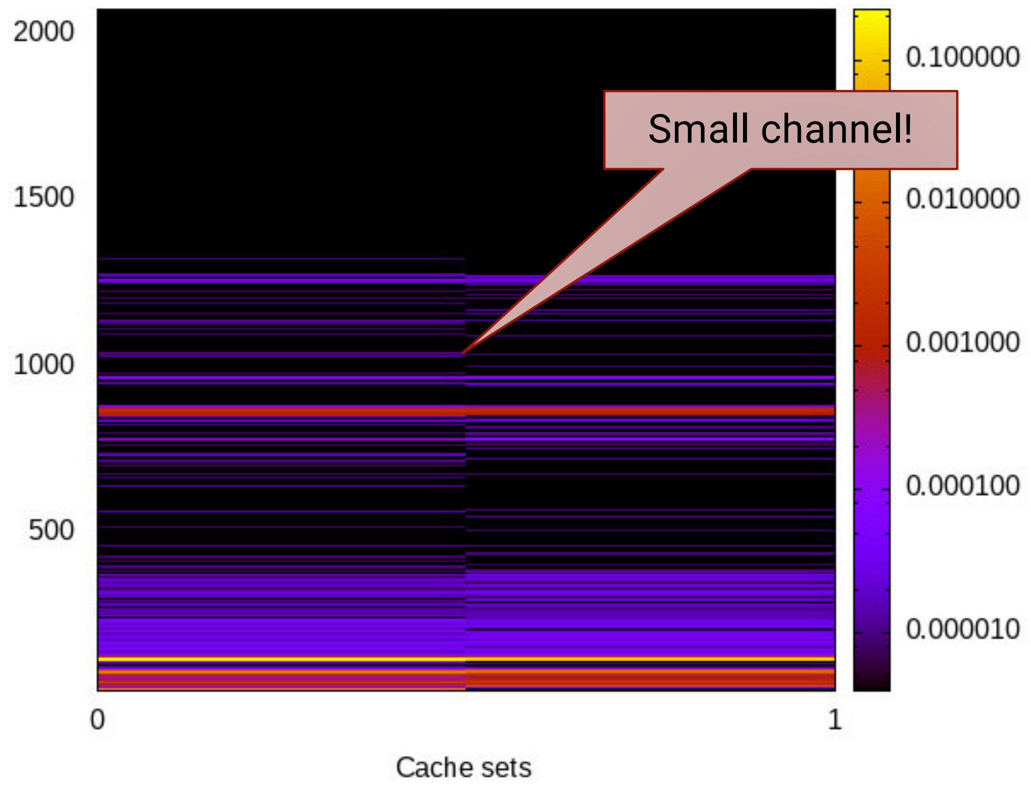


Challenge: Broken Hardware

BHB channel on x86 Sky Lake, no mitigation

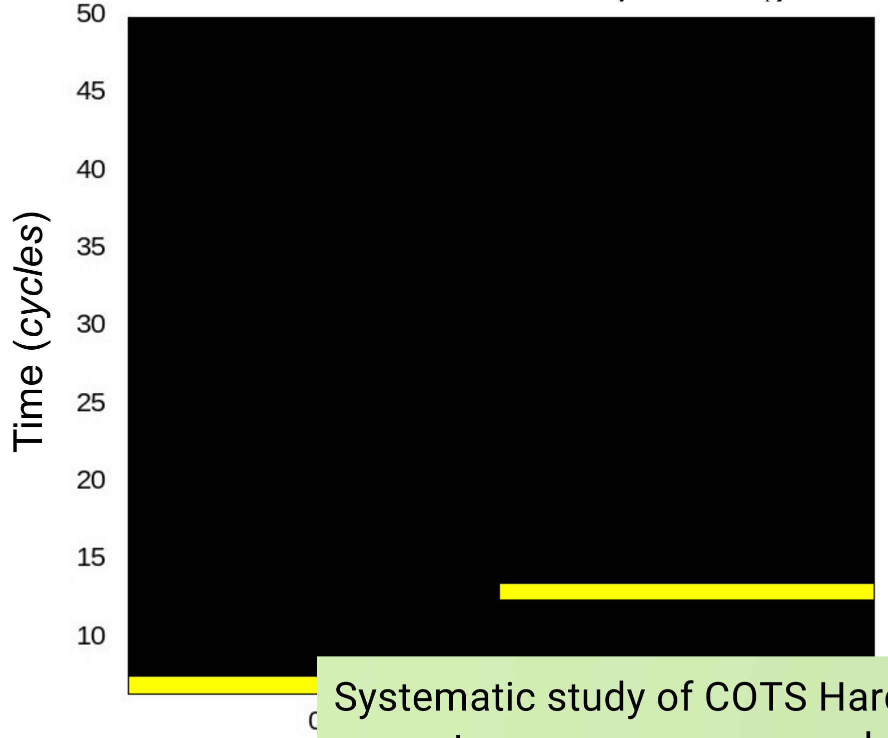


BHB channel on x86 Sky Lake, time protection

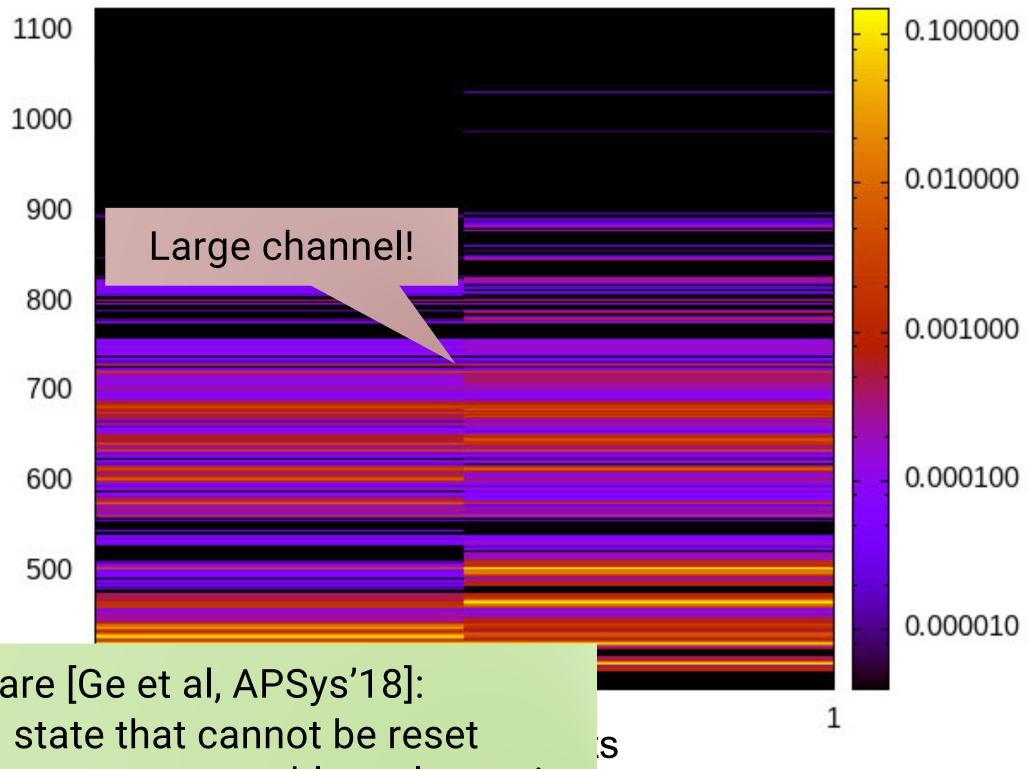


Challenge: Broken Hardware

BHB channel Arm Cortex A53, no mitigation



BHB channel on Arm Cortex A53, time protection



Systematic study of COTS Hardware [Ge et al, APSys'18]:

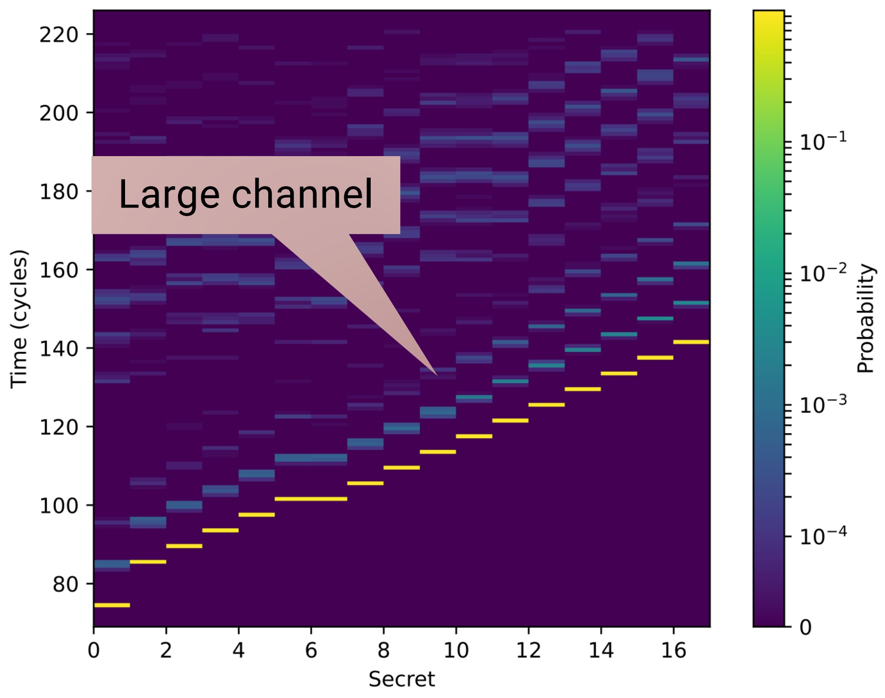
- contemporary processors hold state that cannot be reset
- need a new hardware-software contract to enable real security

RISC-V To The Rescue!

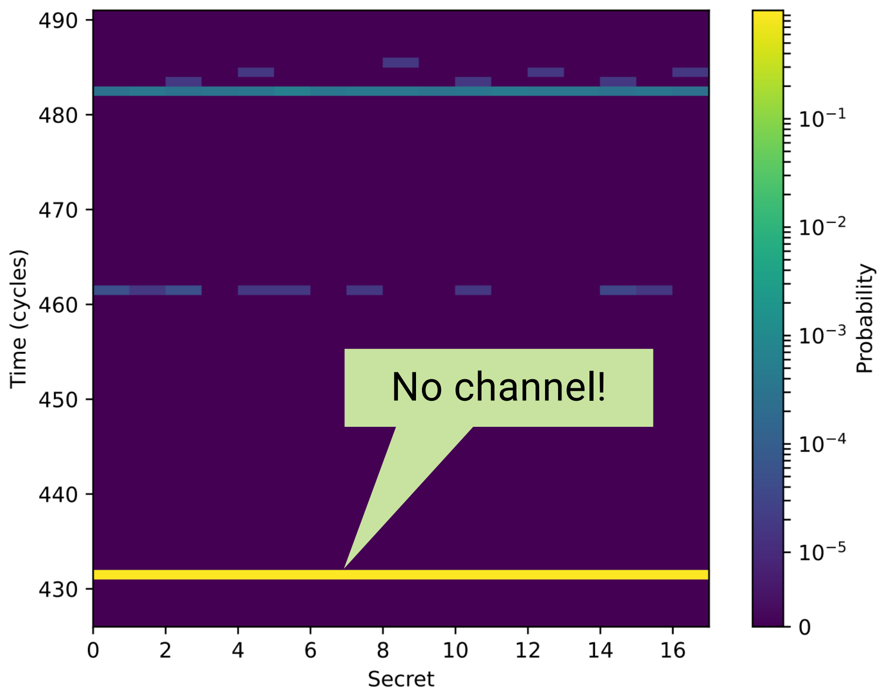
Implemented flush of *all* microarchitectural state in ETH Ariane processor and evaluated channels on FPGA implementation

Similar result for all other channels [Wistoff et all, CARRV'20]

BHB channel Ariane, no mitigation



BHB channel Ariane, time protection

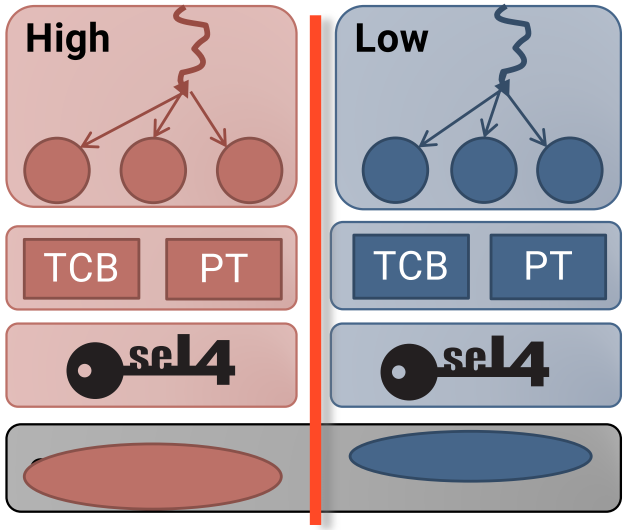


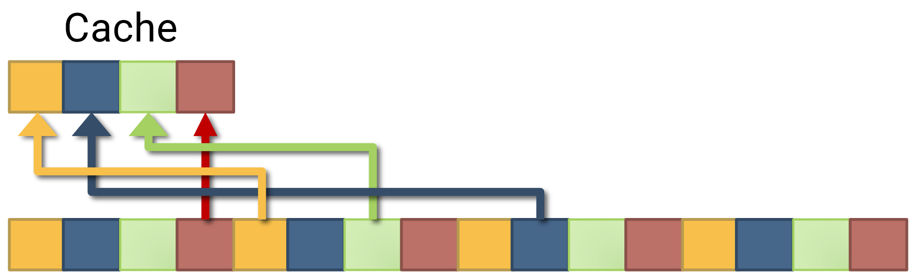
Can We Verify Time Protection?

Assume we have:
- hardware that implements a suitable contract,
- a formal specification of that hardware,

can we prove that our kernel eliminates all timing channels?

Proving Spatial Partitioning





To prove: No two domains share hardware[†]

- Requires abstract model of partitionable hardware (cache model)
- *Functional property, use existing techniques*

[†]Remaining shared kernel data needs separate argument

[†]Core idea: Convert timing channels into storage channels!

Proving Temporal Partitioning

1. T_0 = current_time()
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while (T0+WCET < current_time()) ;
6. Reprogram timer
7. return

Prove: flush all non-partitioned HW
- Needs model of stateful HW
- Somewhat idealised on present HW … but matches our Ariane
- *Functional property*

Prove: padding is correct – how?

Prove: access to shared data is deterministic
- Each access sees same cache state
- Needs cache model
- *Functional property*

Use Minimal Abstraction of Clocks

Abstract clock = monotonically increasing counter

Operations:

- Add constant to clock value
- Compare clock values

To prove: padding loop terminates as soon as **clock ≥ T0+WCET**

- *Functional property!*

Status

- ✓ Published analysis of hardware mechanisms (APSys'18) – *Best Paper*
- ✓ Published time protection design and analysis (EuroSys'19) – *Best Paper*
  - demonstrated effectiveness within limits set by hardware flaws (Arm, x86)
- ✓ Published planned approach to verification (HotOS'19)
- ✓ Published minimal hardware support for time protection (CARRV'20)
  - evaluation demonstrated efficacy and performance
- ➢ Working on:
  - Integrating time-protection mechanisms with clean seL4 model
    - **Done:** Rebased experimental kernel off latest seL4 mainline (x86, Arm, RISC-V)
    - **In progress:** Real system model that integrates the mechanisms
  - Proving timing-channel absence (on conforming hardware)
    - **Done:** Confidentiality proofs for flushing and time padding on simplified HW model
    - **In progress:** Include pre-fetching of data
    - **To do:** Extend to realistic hardware model

Questions?