The seL4® Report

An Update From seL4 Land

Gernot Heiser
Chair, seL4 Foundation

gernot@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.

The Highlights of the Year

- ✓ **The seL4 Foundation** (June's talk right after this):
  - Open governance for the seL4 ecosystem
  - Trademark registration in Australia and US, others in progress
- ✓ **Verification:** RISC-V (RV64) functional correctness proof done!
  - Binary verification (translation correctness) progressing
  - MCS verification progressing (see my FOSDEM'20 talk)
- ✓ **seL4 System** development
  - RFCs for seL4 Core, seL4 Core Platform
  - soon: RFC for seL4 driver framework
- ✓ **Research:**
  - Verifying *time protection*
  - Secure multi-server OS

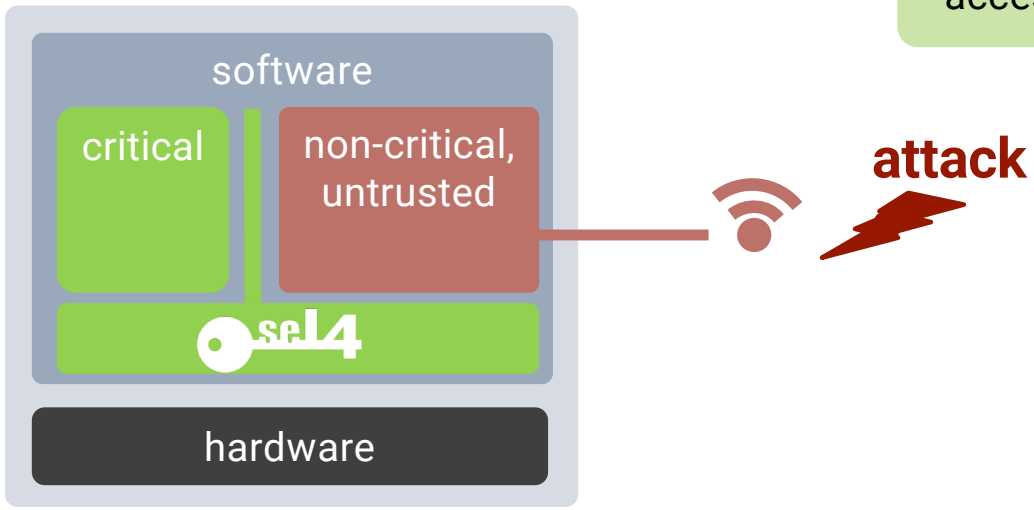
Background

What is seL4?

Background: What is seL4?

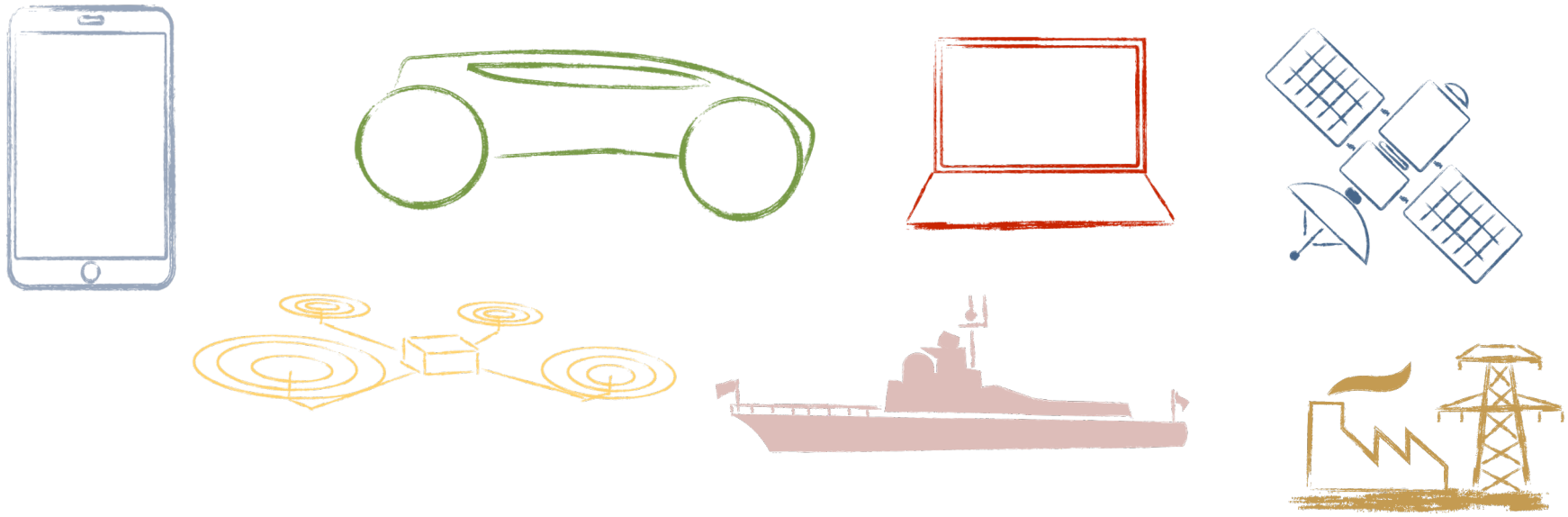
seL4 is an open source, high-assurance, high-performance operating system microkernel

- 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



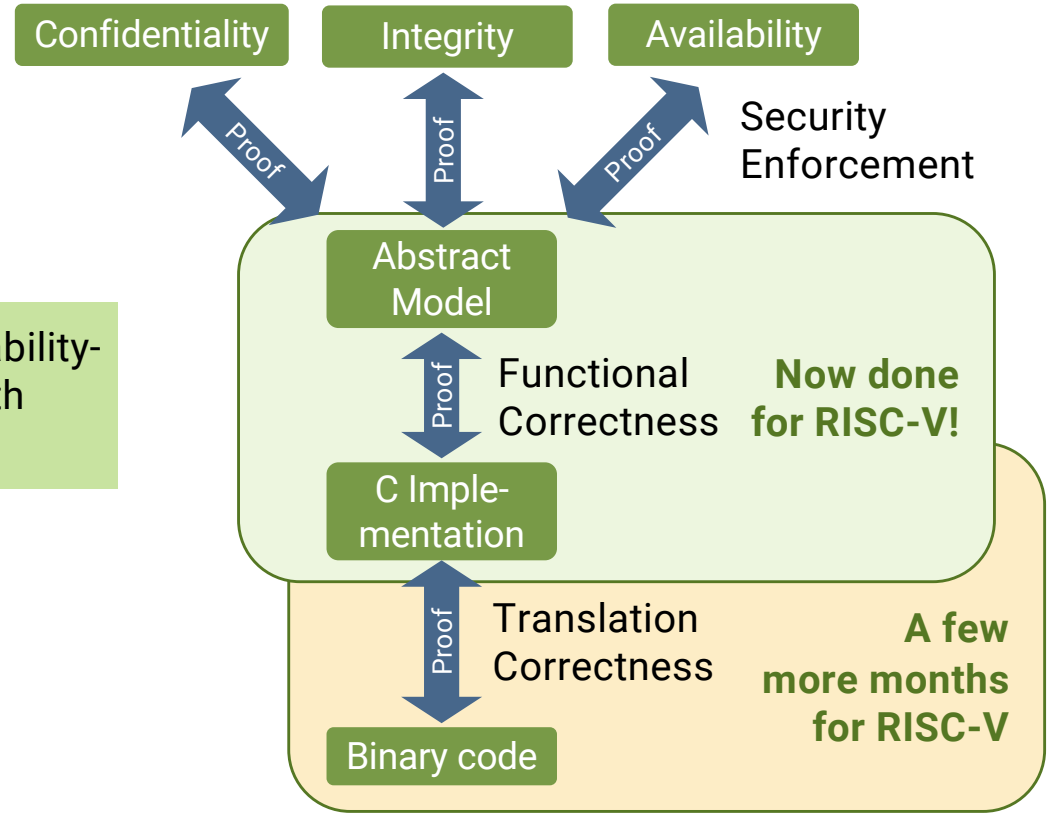
What is seL4?

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...

Unique Verification by Mathematical Proof



seL4: Still only capability-based OS kernel with correctness proof!

Complete proof chain for ARMv7 (32-bit)

seL4: The only OS on RISC-V with correctness proof

… 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

Making seL4 Easier to Use

The seL4 Core Platform

Why seL4 Core Platform?

With seL4 we achieved unprecedented levels of security and user-unfriendliness [2015]

The seL4 API is (for good reason):

- very general
- very low-level
- architecture-dependent
- very spartan

… and **requires a lot of expertise to use correctly**

See https://microkerneldude.wordpress.com/2020/03/11/sel4-design-principles/

Almost all present deployments are:

- embedded/cyber-physical systems
- simple, static architectures

Aims of the seL4 Core Platform

Small OS for IoT, cyber-physical and other embedded use cases

- ✓ Ease development and deployment
- ✓ Provide reasonable degree of application portability, defined HW interfaces
- ✓ Support implementation diversity through well-defined interfaces
- ✓ Support code re-use between related deployments
- ✓ Simple programming model ensures ”correct” use of seL4 mechanisms
- ✓ Retain near-minimal trusted computing base (TCB)
- ✓ Leverage seL4-enforced isolation for strong security/safety
- ✓ Retain seL4’s superior performance
- ✓ Be amenable to formal verification of the TCB

The seL4 Core Platform is POSIX-Compliant

Of Course **Not**

Posix is past its use-by date,
and too inefficient for seL4.
See Curtis Millar's seL4 Summit talk

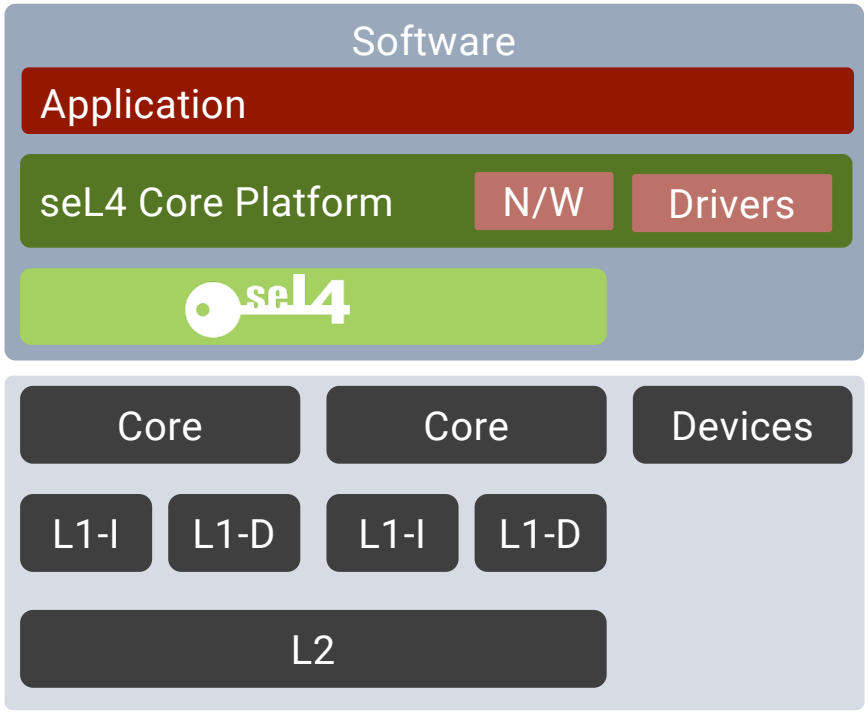
For legacy software
use virtual machines!

Core Platform properties:

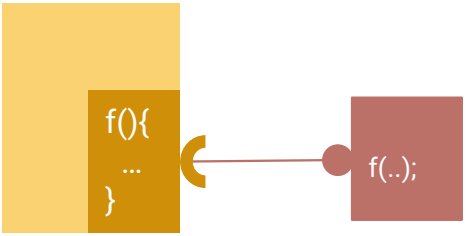
- Simple execution model
- Simple communication model
- Real-time capable
- Efficient
- Deadlock-free
- Some integrity properties enforced by build tools
- Suitable for formal reasoning

Target Hardware: Embedded SoCs

- Homogenous multicore
- Shared L2 cache
- Single system image
- Uniform memory access
- Accelerators (GPUs etc) are “devices”



Core Platform Abstractions

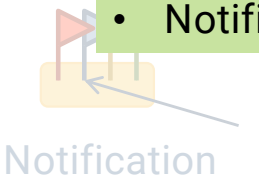


Protected Procedure Call (PPC)

Notification

Memory Region (MR)

Abstractions: Protection Domain



Notification

Protection Domain (PD)

Superficially similar to Unix process
- … but much more lightweight

Consists of several seL4 objects:
- VSpace – virtual memory map
- CSpace – access right
- Scheduling context (SC)
- Priority (fixed)
- Notification

Contains:
- Init procedure
- Notification procedure
- Optional: Protected procedure (PP)

- Runs once, on
- PD's prio and SC

- Invoked when PD's notification is signalled
- Runs on PD's prio and SC
- Bound to a Core!

- All three execute atomically with respect to each other!
- They may
  - signal another PD's notification
  - call another PD's PP

f(..);

Protected Procedure Call (PPC)

- Invoked from other PD
- Runs on PD's prio but *caller PD's SC*

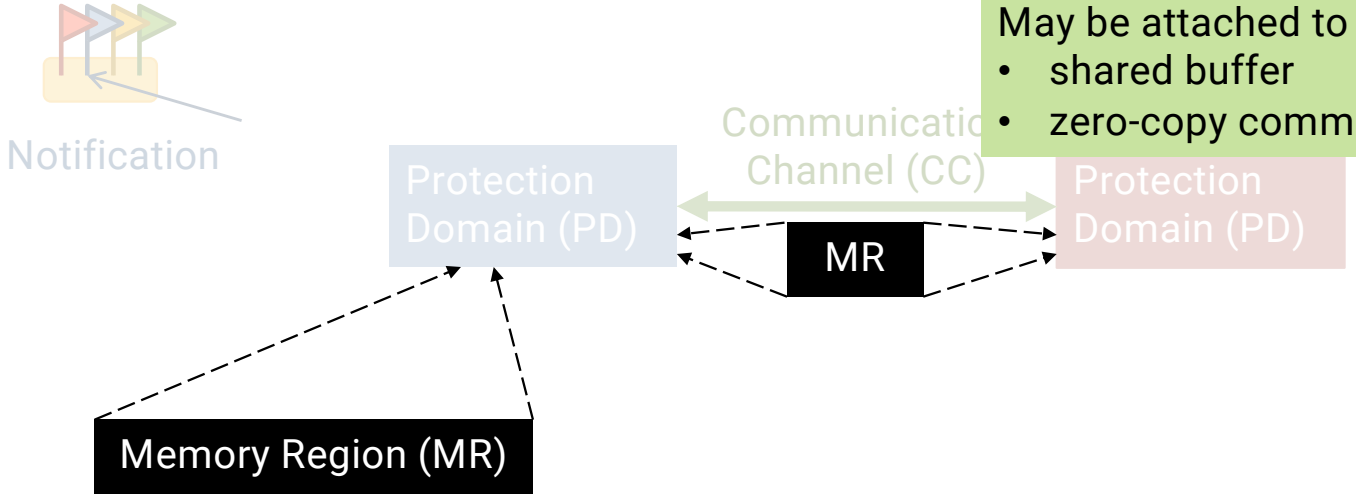
See https://microkerneldude.wordpress.com/2019/03/07/how-to-and-how-not-to-use-sel4-ipc/

Abstractions: Memory Region

Represents physical memory
- contiguous
- integer multiple of page size
- page-aligned

May be mapped into one or more PDs
- at a virtual address
- with defined caching attributes
- with specific permissions

May be attached to a CC
- shared buffer
- zero-copy communication



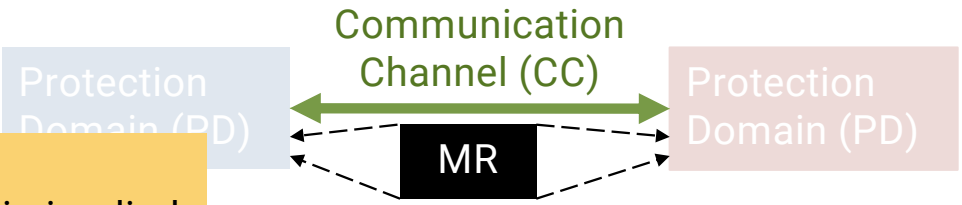
Abstractions: Communication Channels

Supported channel operations:
- reading/writing the channel's MR
  – if mapped into the accessing PD
- referencing channel buffer locations
- signalling the other PD's Notification
- calling the other PD's PP (if available)

Supports communication between PDs
- connects exactly two PDs
- any pair of PDs has at most one common CC
- data flow can be uni- or bidirectional
- information flow is bi-directional (no data diode!)

Using reference wrappers, not plain pointers!

Potentially cross-core

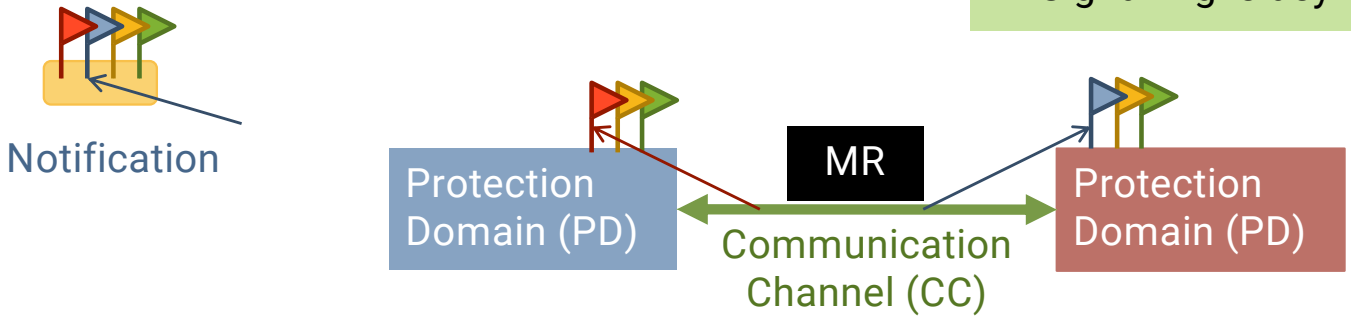


In general:
- no trust relationship implied
- CCs form non-directed, cyclic graph

Abstractions: Notifications

- Notifications are binary semaphores
  - Multiple signals from same PD may not invoke *notification procedure* multiple times
- Processing of signals from multiple PDs should happen in priority order
  - Ideally enforced by Core Platform tooling
- For now limit of 64 CCs per PD (seL4 limitation)

Support triggering of events:
- can signal PD's Notification through CC
- this invokes target PD's *notification procedure*
- Platform provides source PD's identity
  - uses seL4's badged capabilities
- Signalling is asynchronous

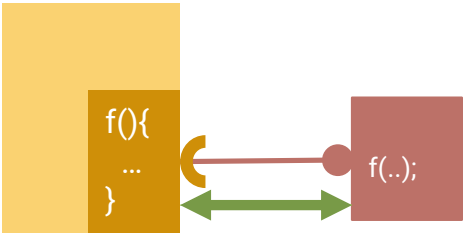


Are associated with a PD
- exactly one per PD

Memory Region (MR)

Abstractions: Protected Procedure Calls

Supports execution of code in a different PD
- Callee (“server”) must have a PP
- Caller (“client”) and callee must share a CC
- PPC arguments may reference locations in the CC’s MR (using reference wrappers)
- Arguments limited to 16 words in total
- PPCs may nest



Protected Procedure Call (PPC)

Notification

Protec
Domai

Asymmetric relationship:
- Client trusts server
- PPC must not block
- Server must be higher prio than client
- Remember: PPC runs on client’s SC and thus the client’s core!
- PPCs form acyclic, directed graph

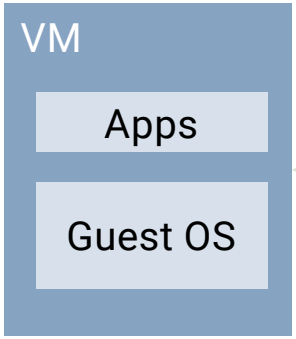
Enforceable by build tools!

Memory Region (MR)

Abstractions: Virtual Machine (VM)

A VM is a PD with extra attributes
- supports extra execution mode (guest mode)
- else behaves like any PD
  - may share MRs
  - may signal/be signalled
  - may be client or server of a PPC

Notification



- Not yet fully specified
- Not supported in first version

Memory Region (MR)

Core Platform Considerations

Initially all PDs will be single-core
- Single scheduling context means single core

This restriction will be removed in the near future for *pure client* PDs
- PDs without a PPC

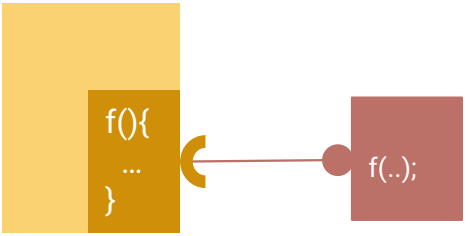
Supporting multi-threaded servers is possible
- a bit more complicated
- ≤ 1 thread per core

- For now targeting static architectures:
  - All PDs known at build time

Will (eventually) support late loading/re-loading of (known) PDs

Summary: seL4 Core Platform

- Designed to ease construction of well-designed seL4-based embedded systems
- Design mostly complete: RFC-5
- Will integrate with the seL4 Driver Framework
- We'll provide best-practice training material



Protected Procedure Call (PPC)

Notification

Memory Region (MR)

Research Update

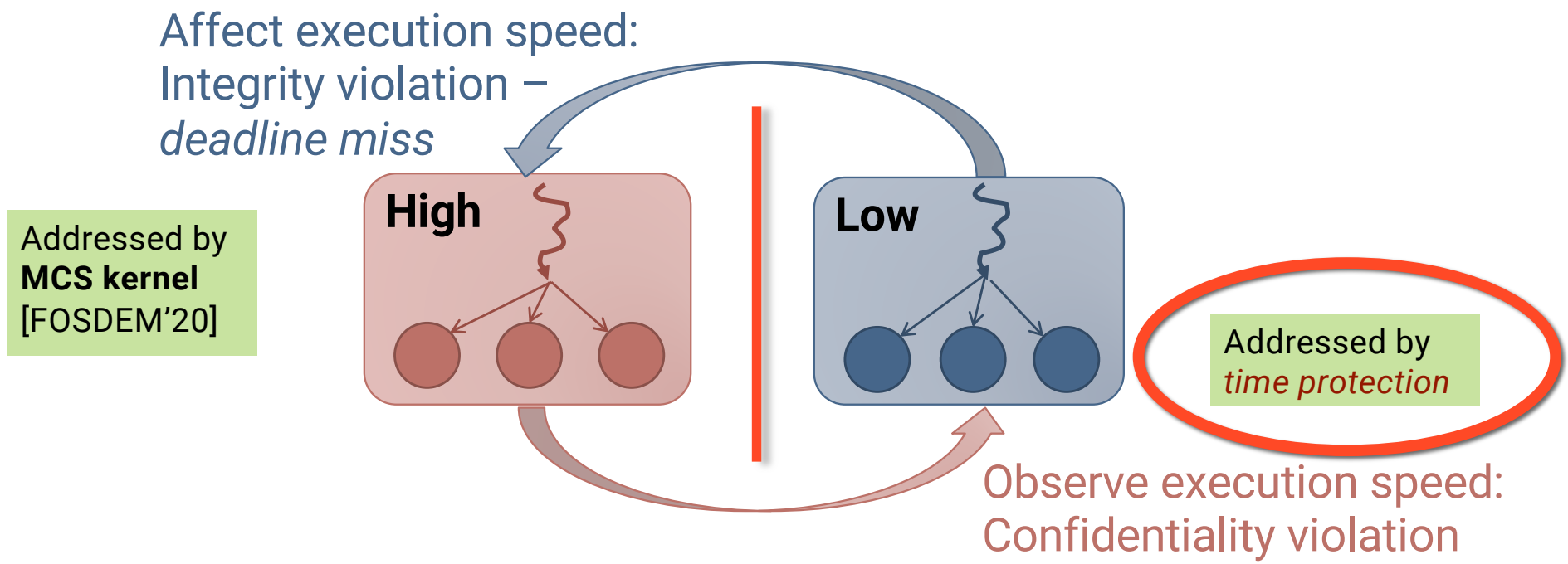
Verifying time protection

Secure multi-server OS

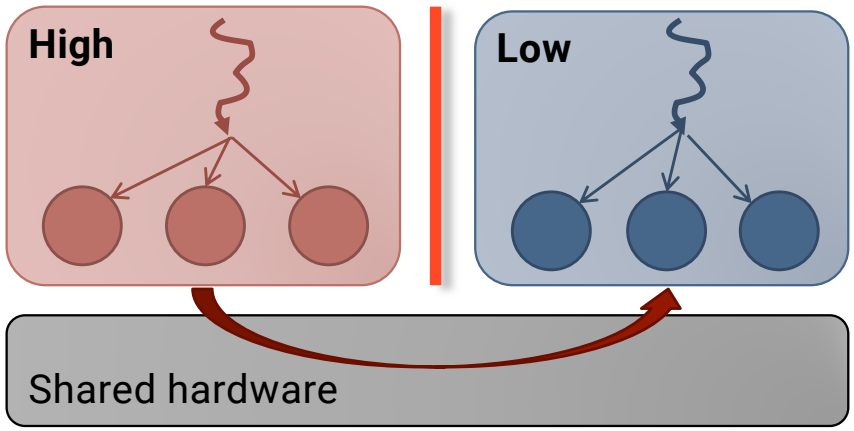
What's the Issue with Temporal Isolation?

Safety: Timeliness
- Execution interference

Security: Confidentiality
- Leakage via timing channels



Cause: Competition for HW Resources

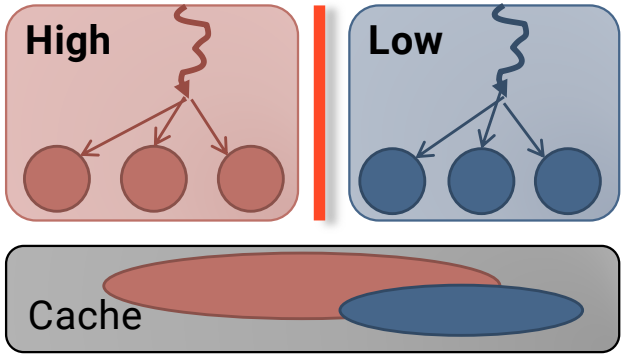


Affect execution speed

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

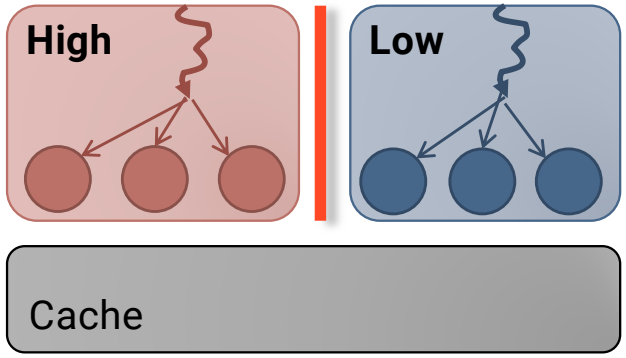
Solution: *Time Protection* – Eliminate interference by preventing sharing

Time Protection: Partition all Hardware State

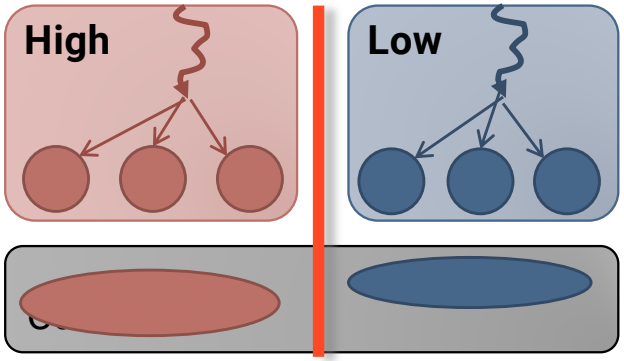


Temporally partition

Flush



Spatially partition



Need both!

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

- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

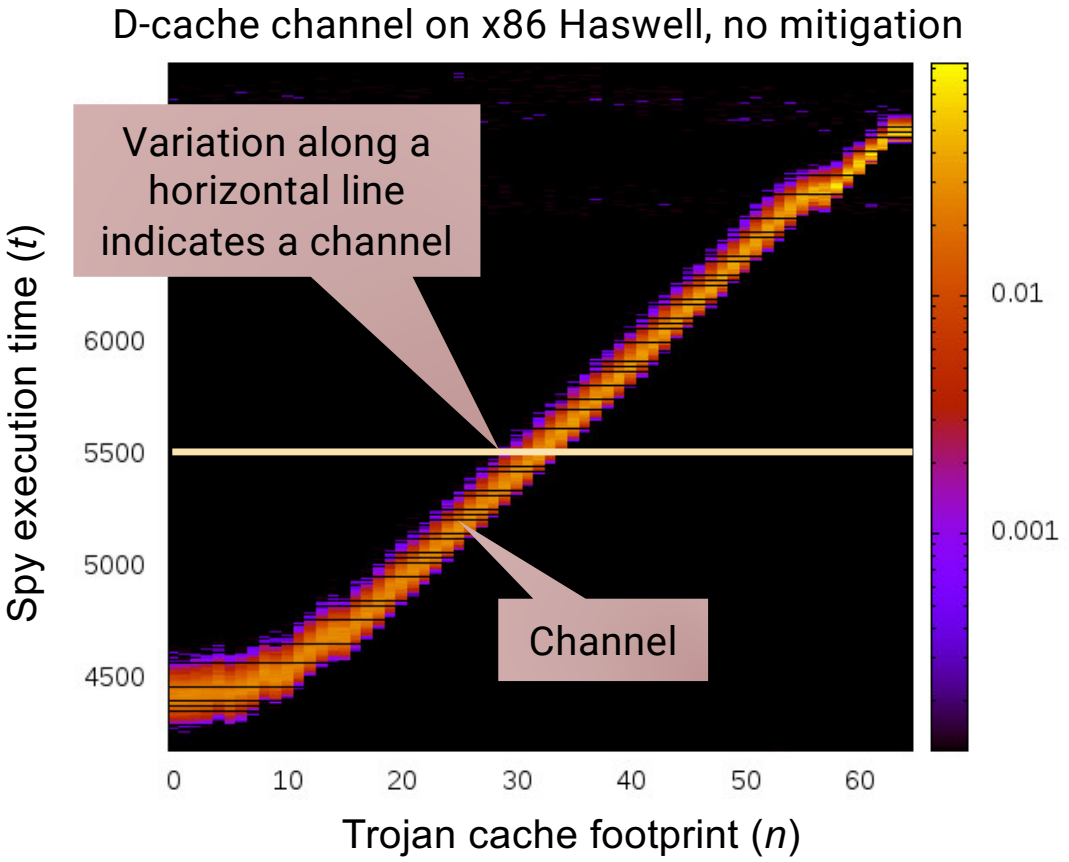
- HW threads
- cores

More details:

- [Heiser, FOSDEM'20]
- [Ge et al, EuroSys'19]

Best Paper Award

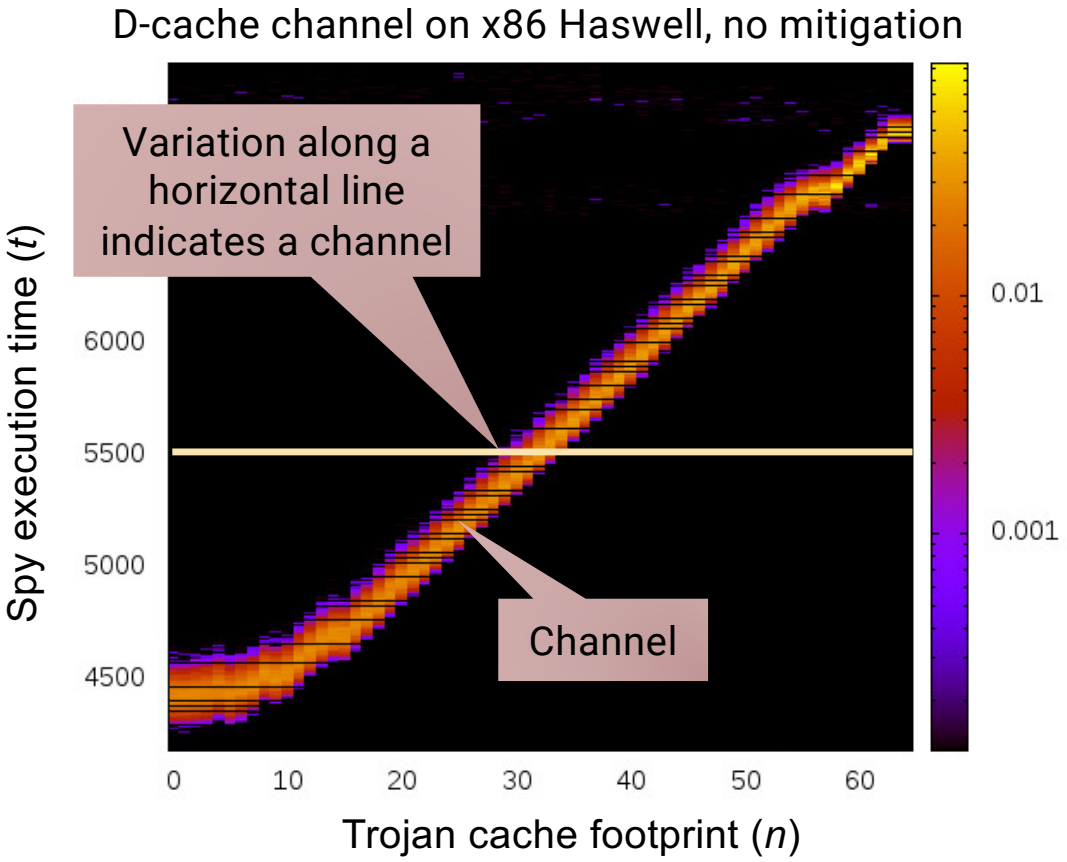
Measuring Leakage: Channel Matrix

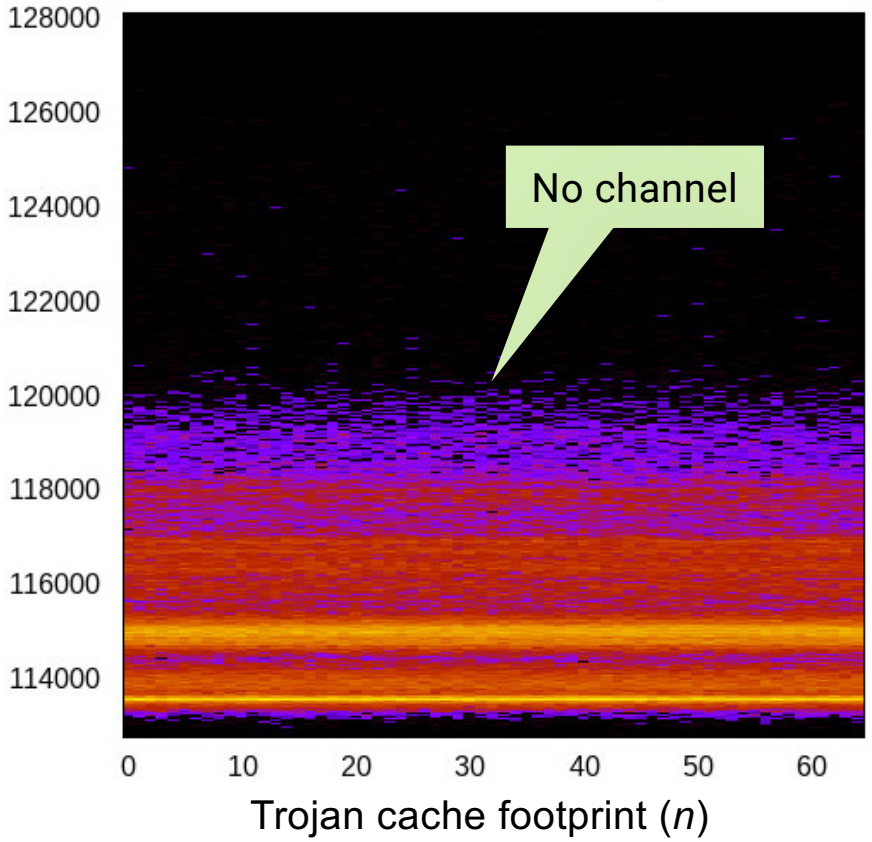


Channel matrix:

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

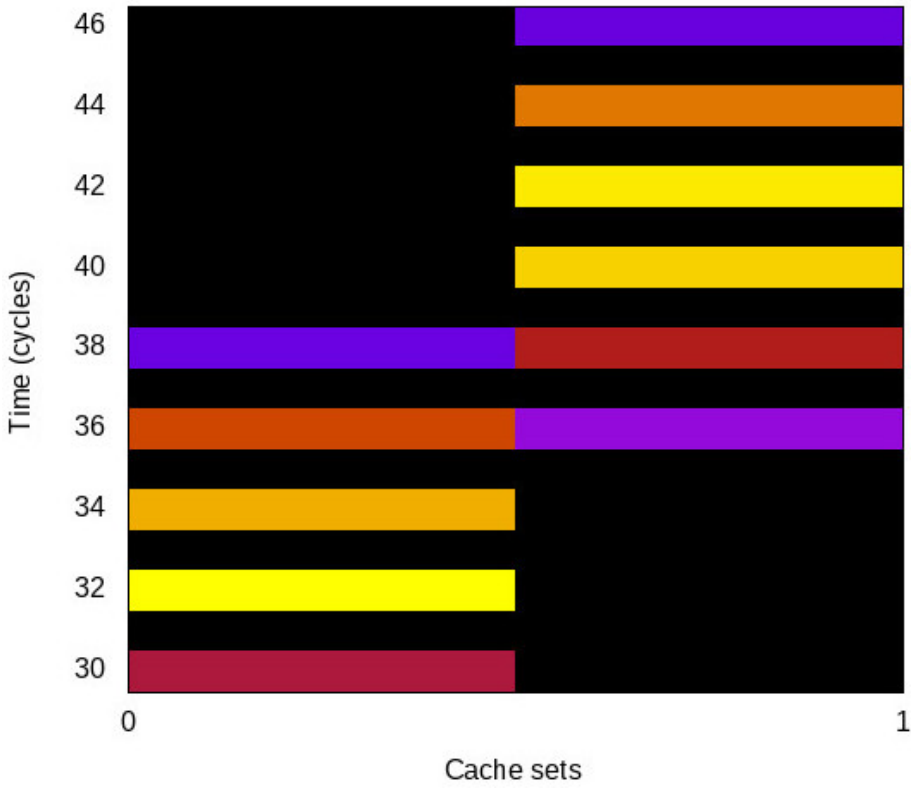
Measuring Leakage: Channel Matrix



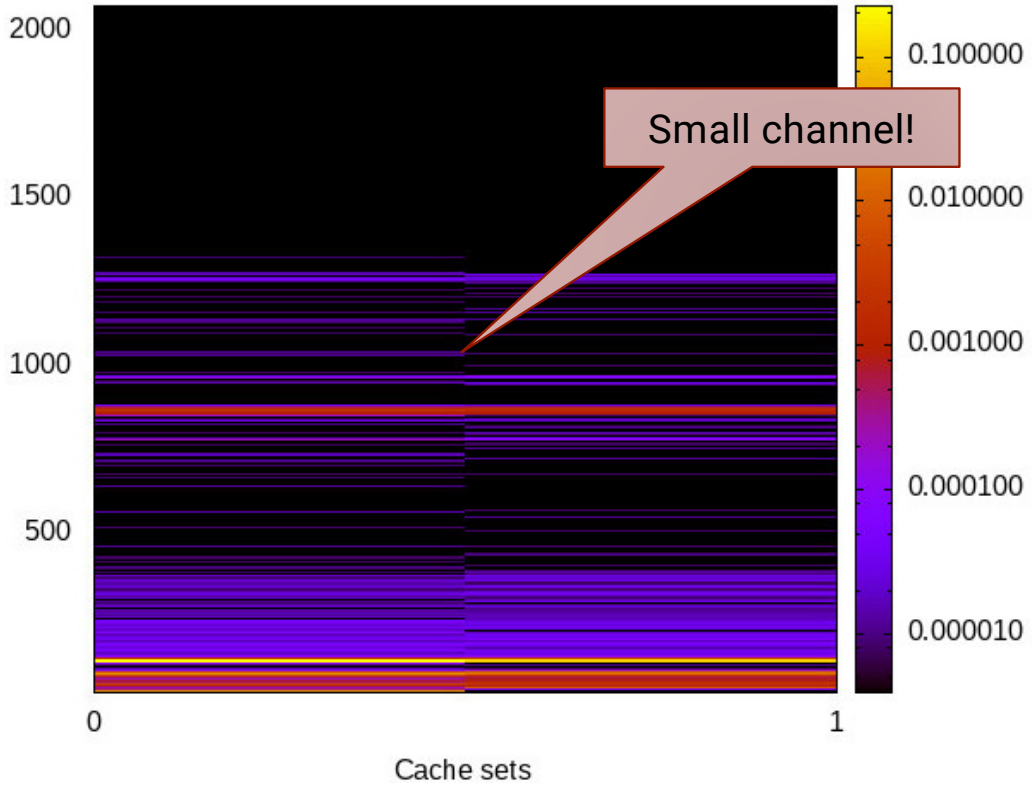


Challenge: Broken Hardware

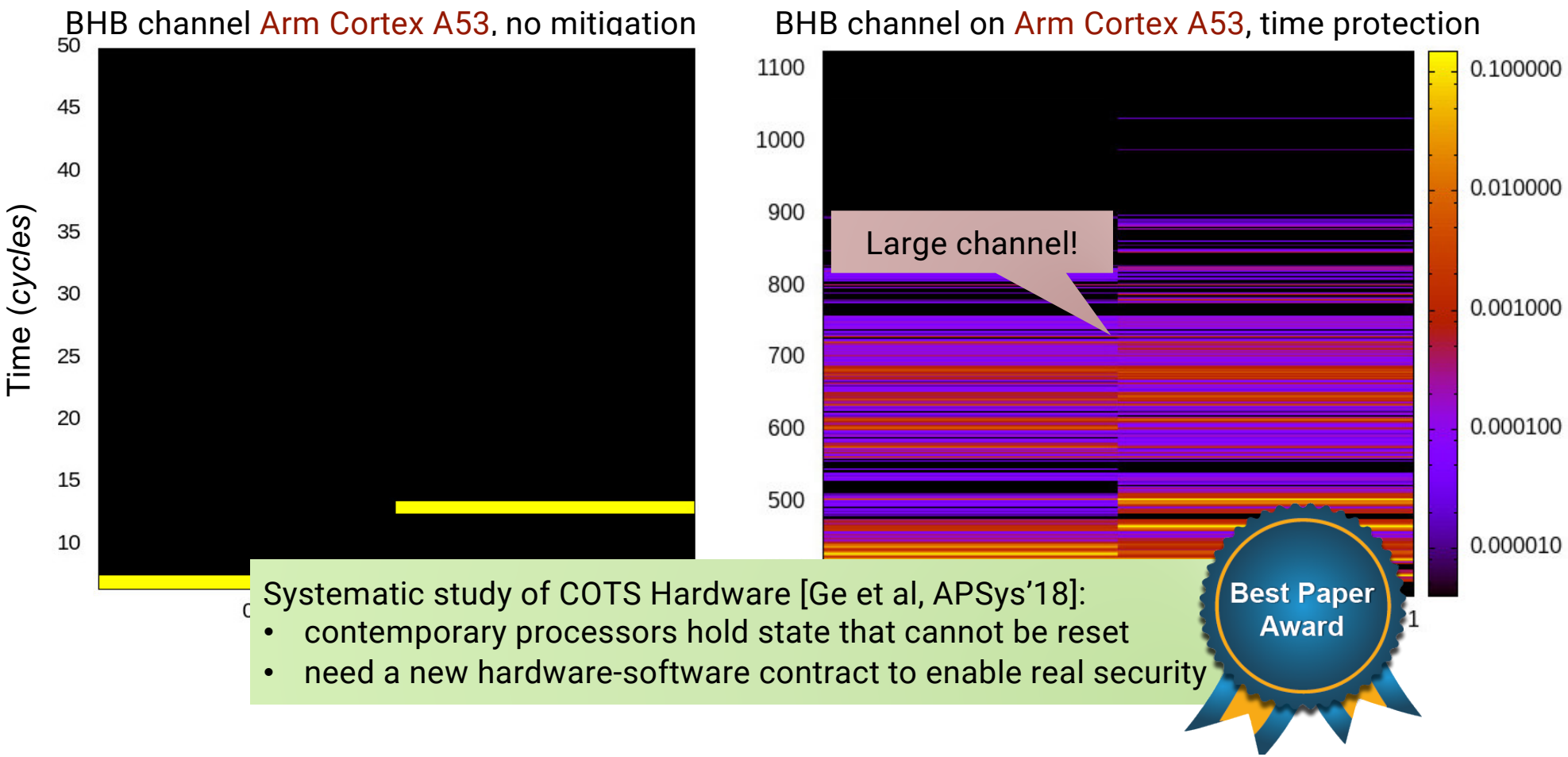
BHB channel on x86 Sky Lake, no mitigation



BHB channel on x86 Sky Lake, time protection



Challenge: Broken Hardware



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

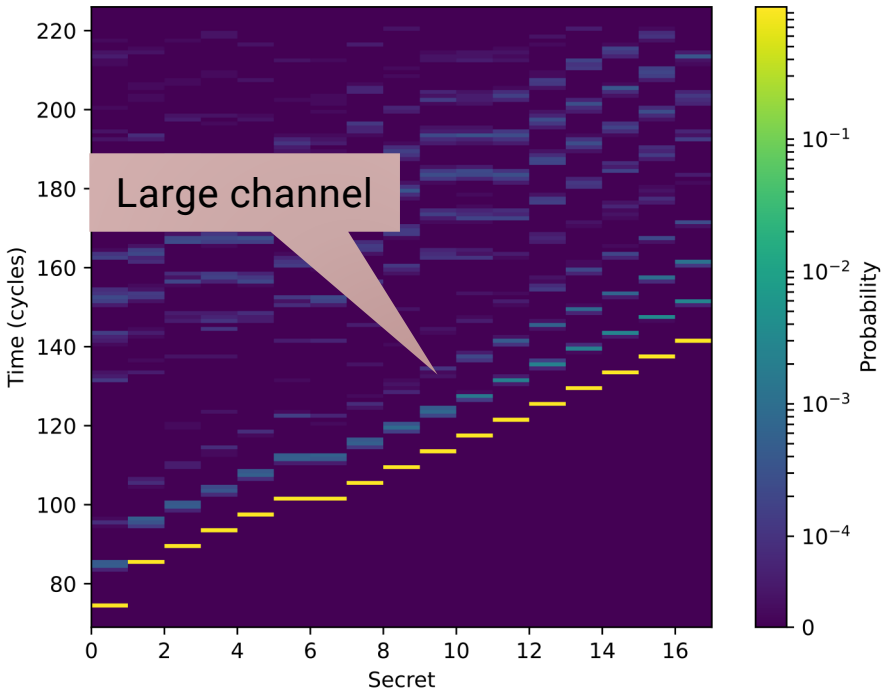
RISC-V To The Rescue!

New instruction `fence.t`: flush of *all* micro-architectural state in ETH Ariane processor and evaluated channels on FPGA implementation

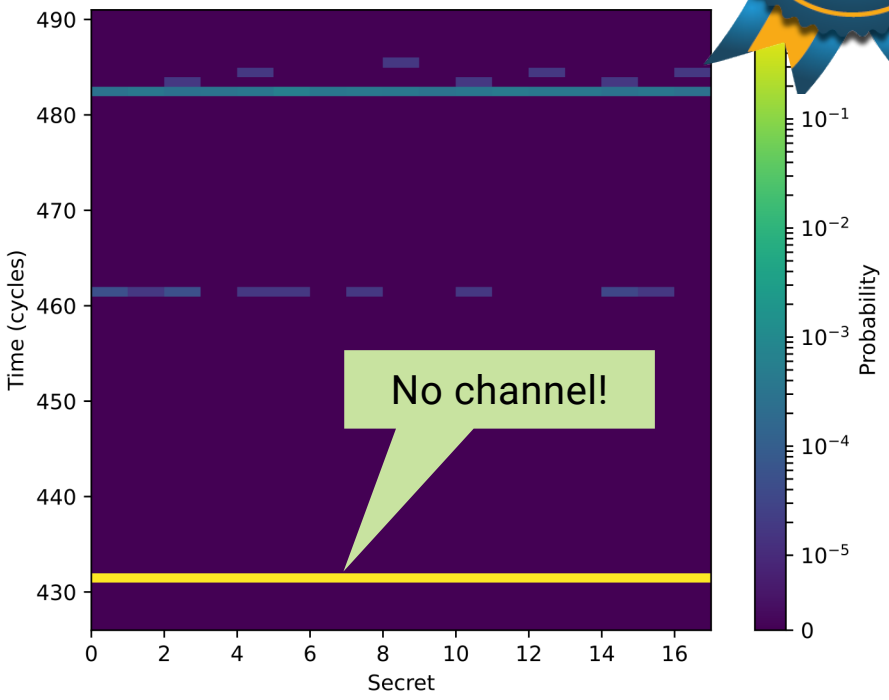
Similar result for all other channels [Wistoff et all, DATE'21]

Best Paper Award

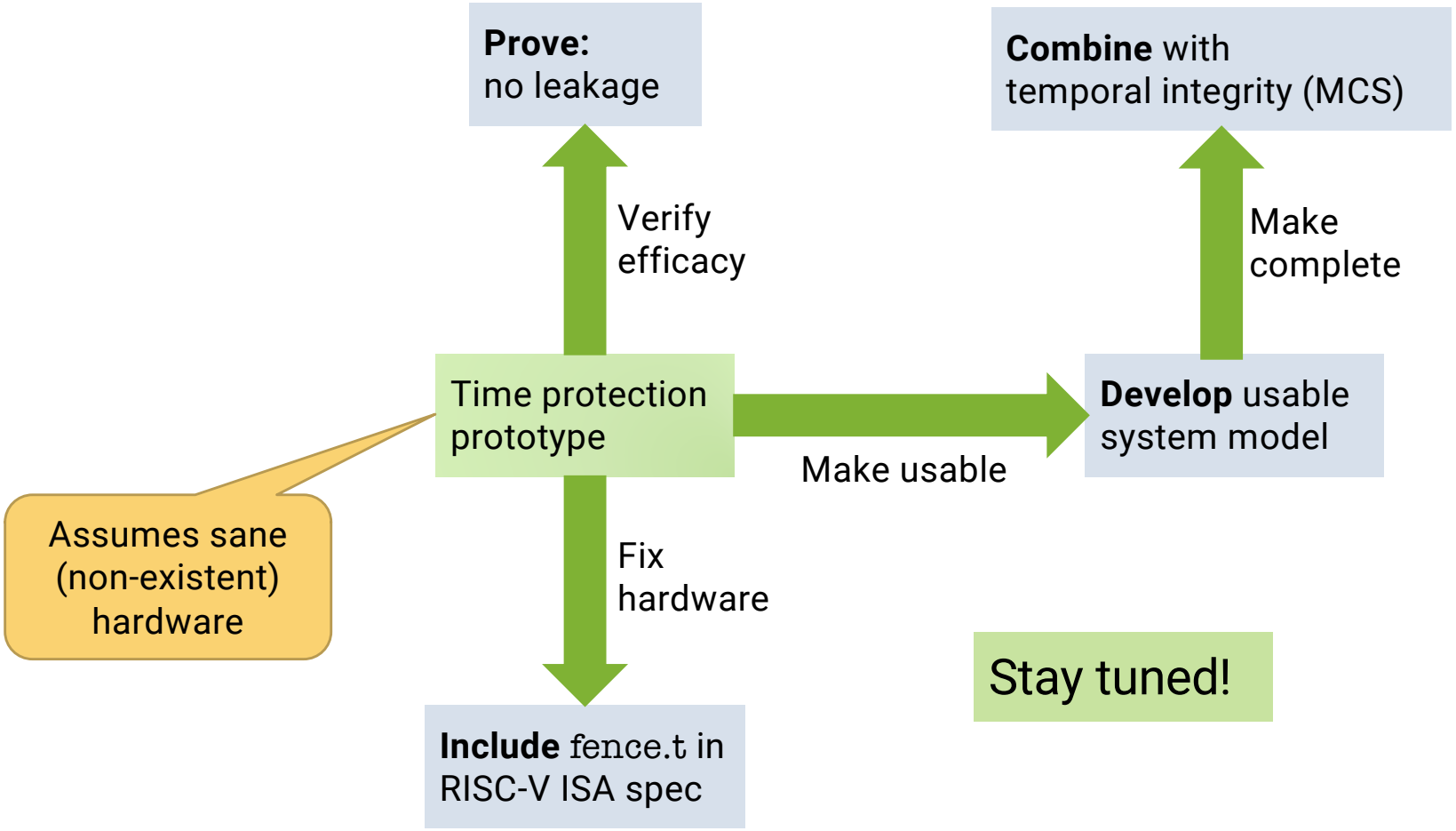
BHB channel Ariane, no mitigation



BHB channel Ariane, time protection



On-Going Work



Research: Secure Multi-Server OS

Aim: A truly secure, general-purpose OS

- ✓ Support wide class of use cases, fully dynamic
- ✓ Support wide class of security policies
- ✓ Support changes of security policy during execution
- ✓ Support least privilege (aka principle of least authority, POLA)
- ✓ Support formal verification of security policy enforcement
  - ➢ Incl confidentiality, integrity, availability
- ✓ Performance comparable to monolithic systems

Secure Multi-Server OS Features

- ✓ Policy-mechanism separation:
  - ➢ Servers implement abstractions independent of security policy
  - ➢ Policy is encapsulated in a single *security server*
- ✓ Dynamic information-flow control:
  - ➢ Communication limited by security policy
- ✓ Resource-availability guarantee through *resource donation*
- ✓ Performance by minimising security overhead
  - ➢ Checks only on connection establishment
  - ➢ Connection removed on policy change
- ✓ Design for formal verification

Stay tuned for detailed white paper!

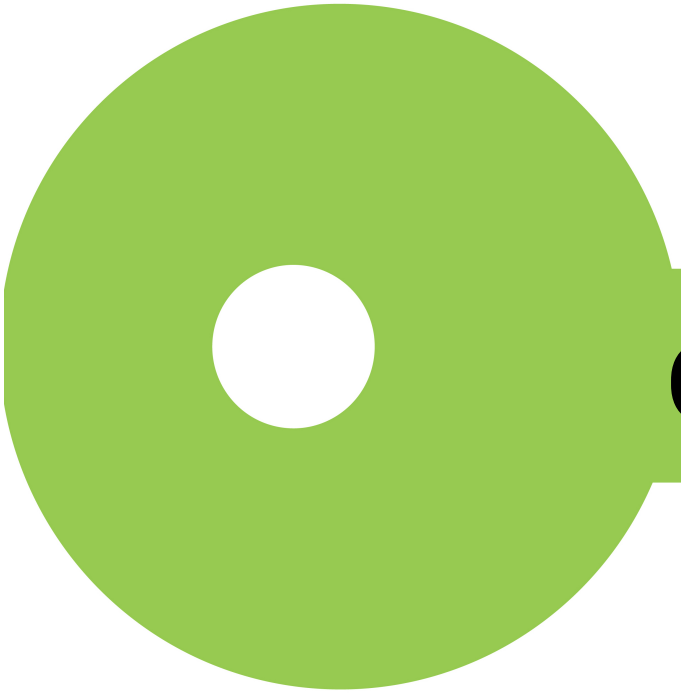
Take-Aways:

- ✓ seL4 Foundation takes seL4 to the next level
  - ➢ open development
  - ➢ open governance
  - ➢ community funding
  - ➢ maturing ecosystem
  - ➢ increasing deployments

If you're a member of the seL4 Community, please let us know how you want the next seL4 Summit to look!

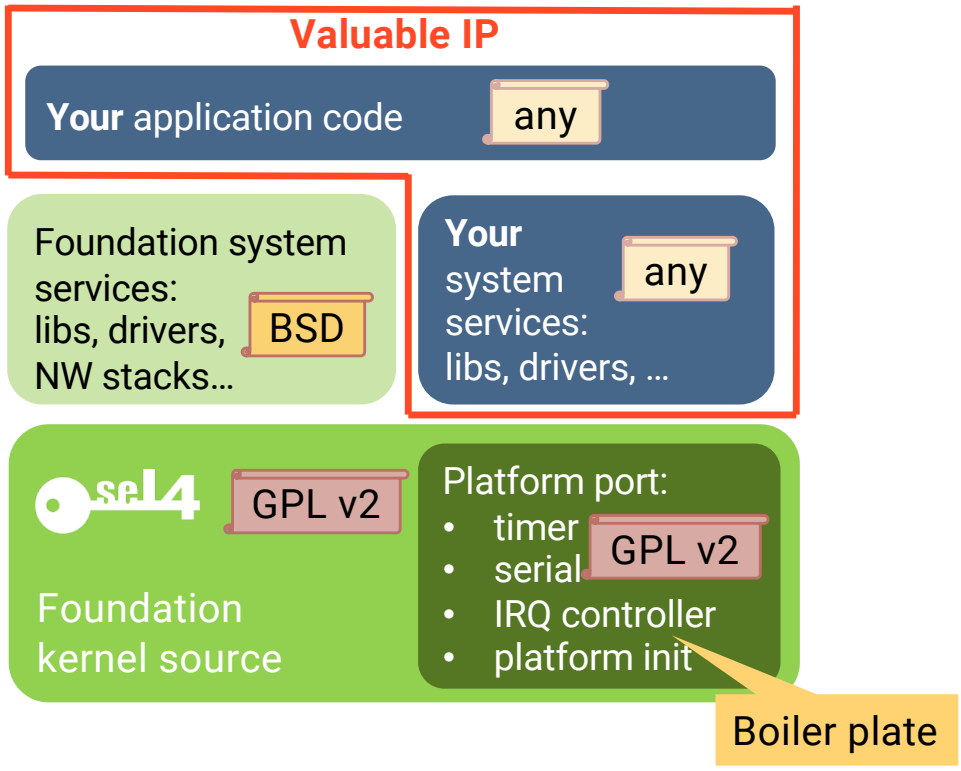
- ✓ RISC-V is now a first-class seL4 architecture
  - ➢ functional correctness done, other verification in progress
- ✓ Ambitious research agenda:
  - ➢ provably eliminate timing channels
  - ➢ secure, general-purpose multi-server OS

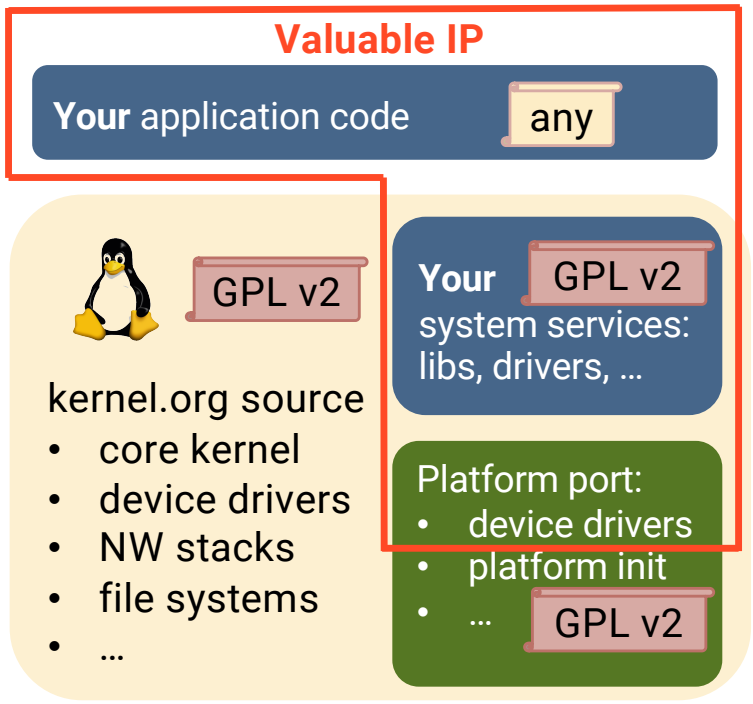
seL4: Defining the state of the art in secure OS since 2009



Questions?

Licensing: What Does the GPL Imply?





What Does This Mean?

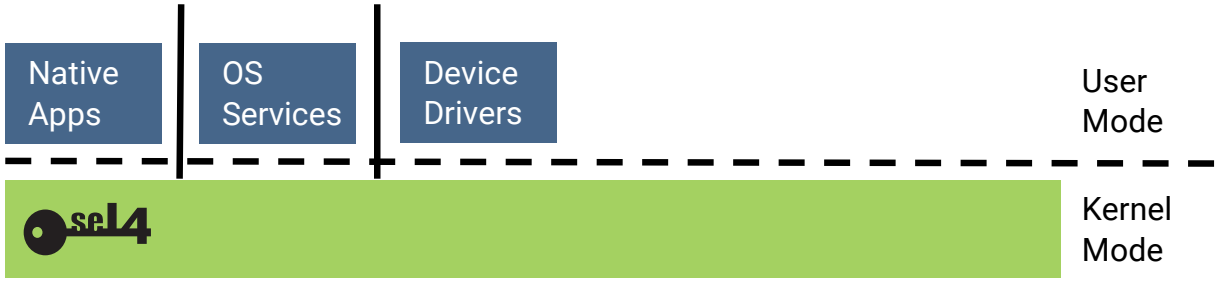
Kinds of properties proved

- Behaviour of C code is fully captured by abstract model
- Behaviour of C code is fully captured by executable model
- Kernel never fails, behaviour is always well-defined
  - assertions never fail
  - will never de-reference null pointer
  - will never access array out of bounds
  - cannot be subverted by misformed input
  - …
- All syscalls terminate, reclaiming memory is safe, ...
- Well typed references, aligned objects, kernel always mapped…
- Access control is decidable

Can prove further properties on abstract level!

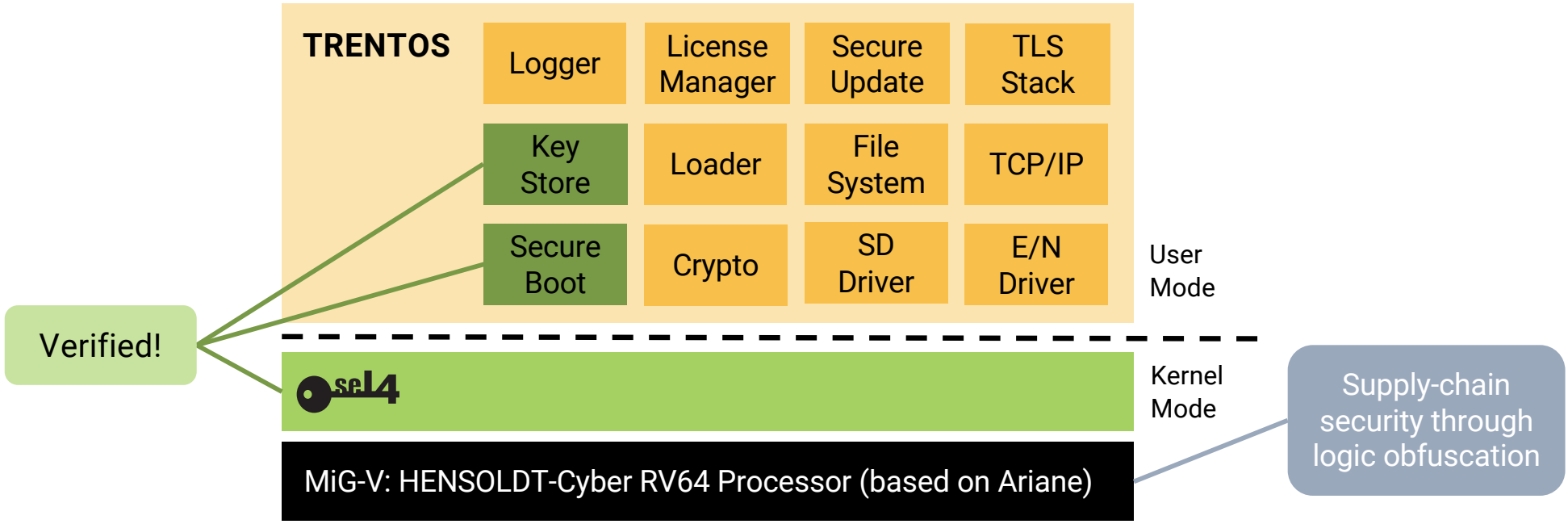
How Can I Use It?

- ✓ Open source (GPL v2): Download from https://github.com/sel4
- ✓ But keep in mind: seL4 is an OS microkernel and hypervisor, not an OS!
- ✓ Many OS components available on the seL4 GitHub

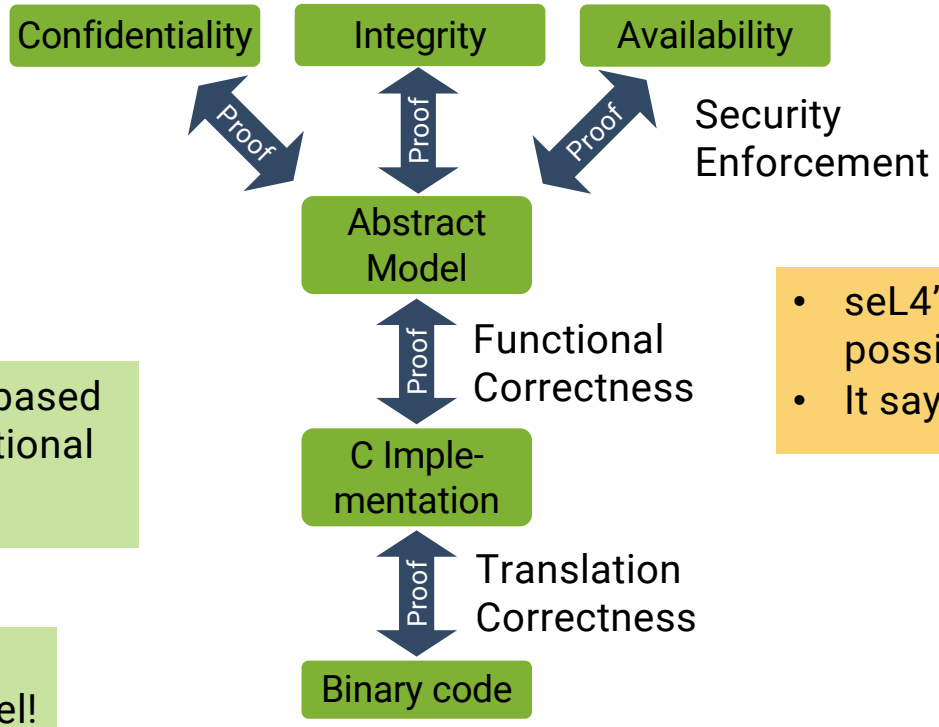


How Can I Use It?

- ✓ Open source (GPL v2): Download from https://github.com/sel4
- ✓ But keep in mind: seL4 is an OS microkernel and hypervisor, not an OS!
- ✓ Many OS components available on the seL4 GitHub
- ✓ Alternative: HENSOLDT Cyber's TRENTOS



So, Why Aren't We Done?



Still only capability-based OS kernel with functional correctness proof

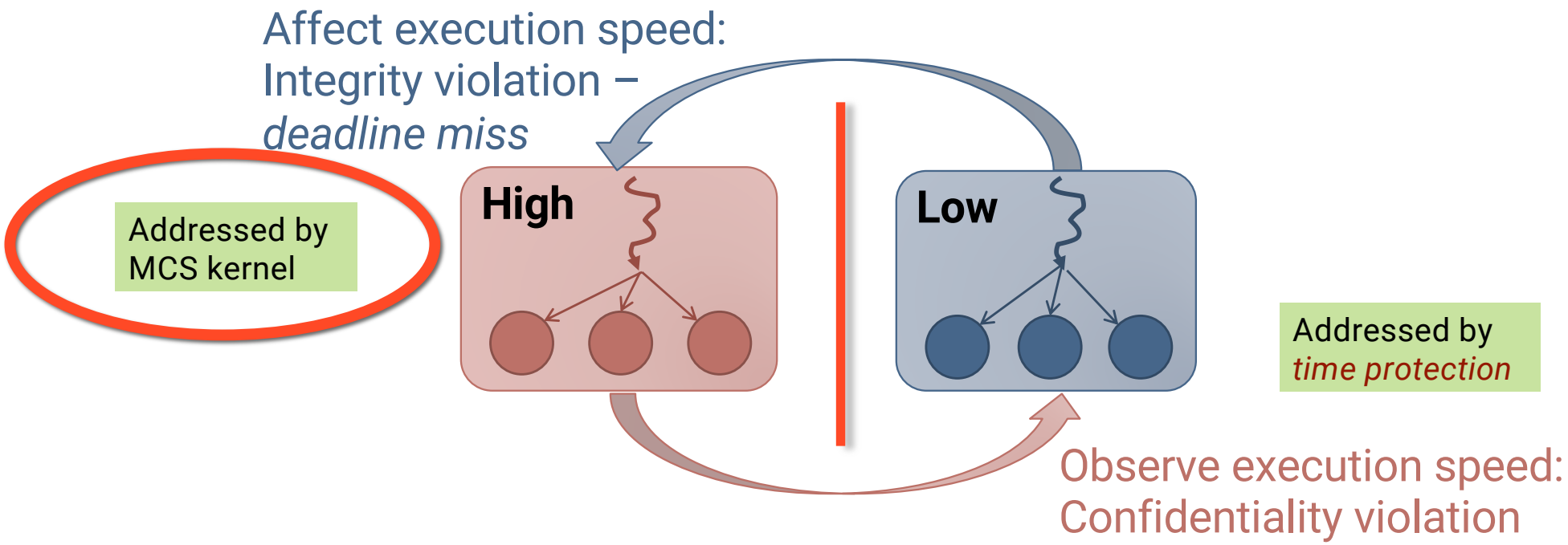
Still the world's fastest microkernel!

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

What's the Issue with Temporal Isolation?

Safety: Timeliness
- Execution interference

Security: Confidentiality
- Leakage via timing channels



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

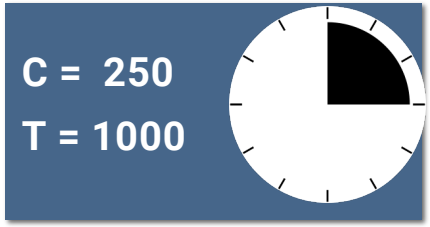
- Priority
- Time slice

New thread attributes

- Priority
- Scheduling context capability

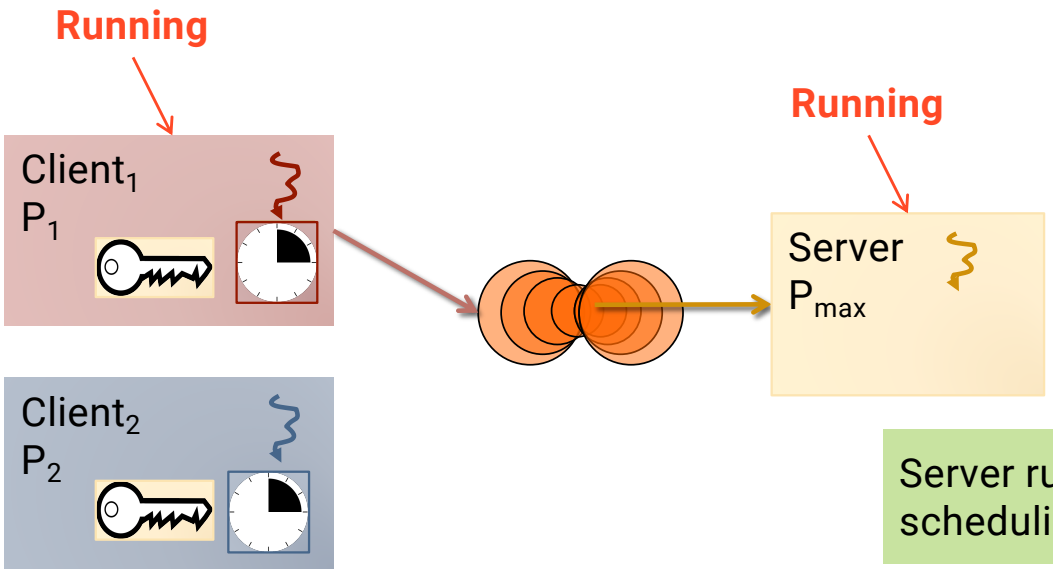
Scheduling context object
T: period
C: budget (≤ T)

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

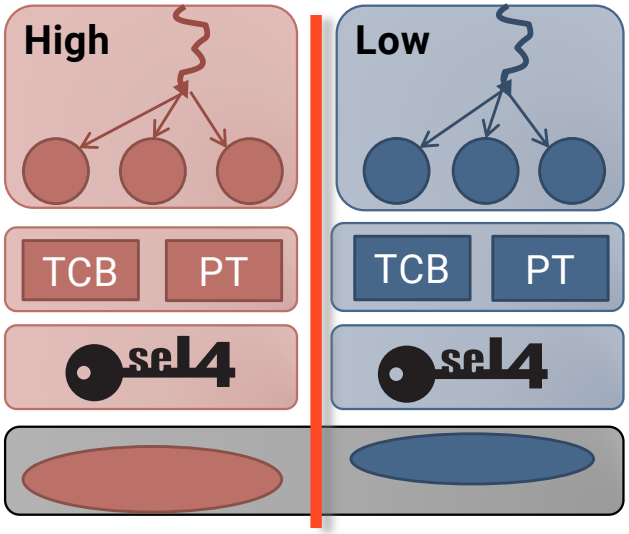
Scheduling-context capabilities: a principled, light-weight OS mechanism for managing time [Lyons et al, EuroSys'18]

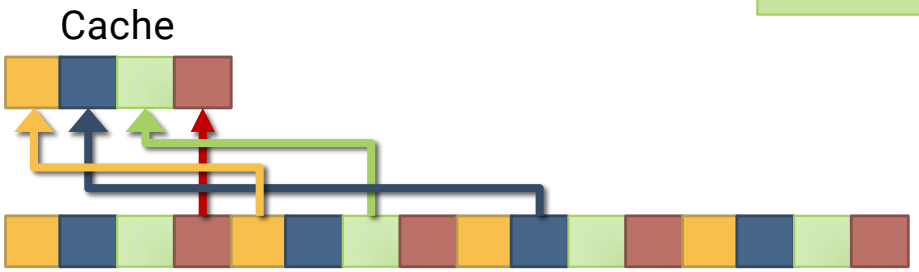
MCS Summary

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

Partition Hardware: Page Colouring





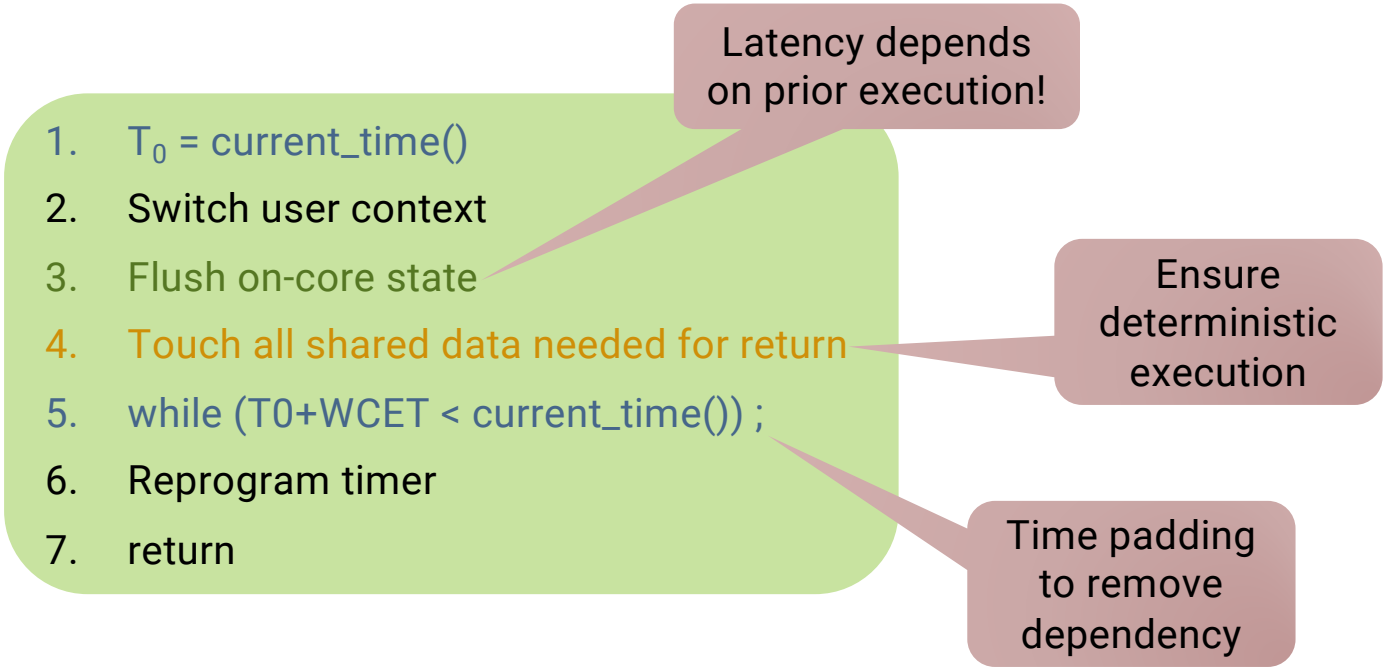
- Partitions get frames of disjoint colours preventing interference
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours dynamic kernel memory
- Per-partition kernel image to colour kernel

[Ge et al. EuroSys'19]

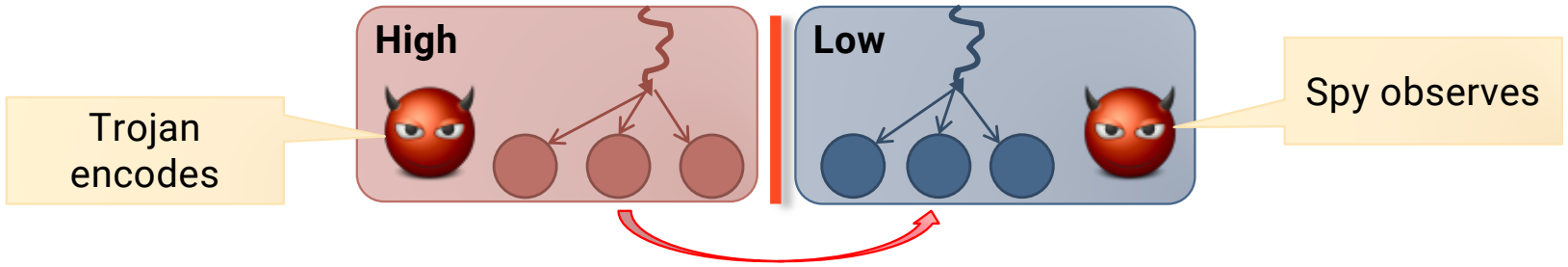
Small amount of static kernel memory needs special handling

Temporal Partitioning: Flush on Switch

Must remove any history dependence!



Evaluation: Prime & Probe Attack



1. Fill cache with own data
2.
3. Traverse cache, measure *execution time*

Output signal

2. Touch *n* cache lines

Input signal

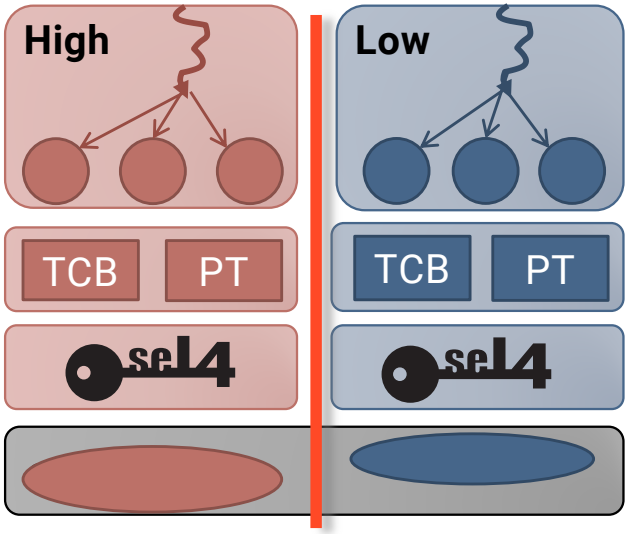
Can We Verify Time Protection?

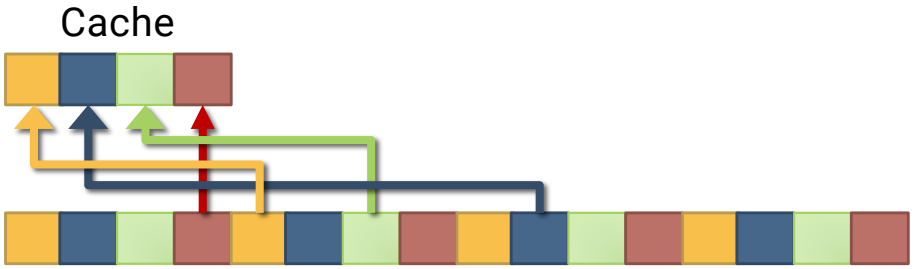
Assume we have:

- 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





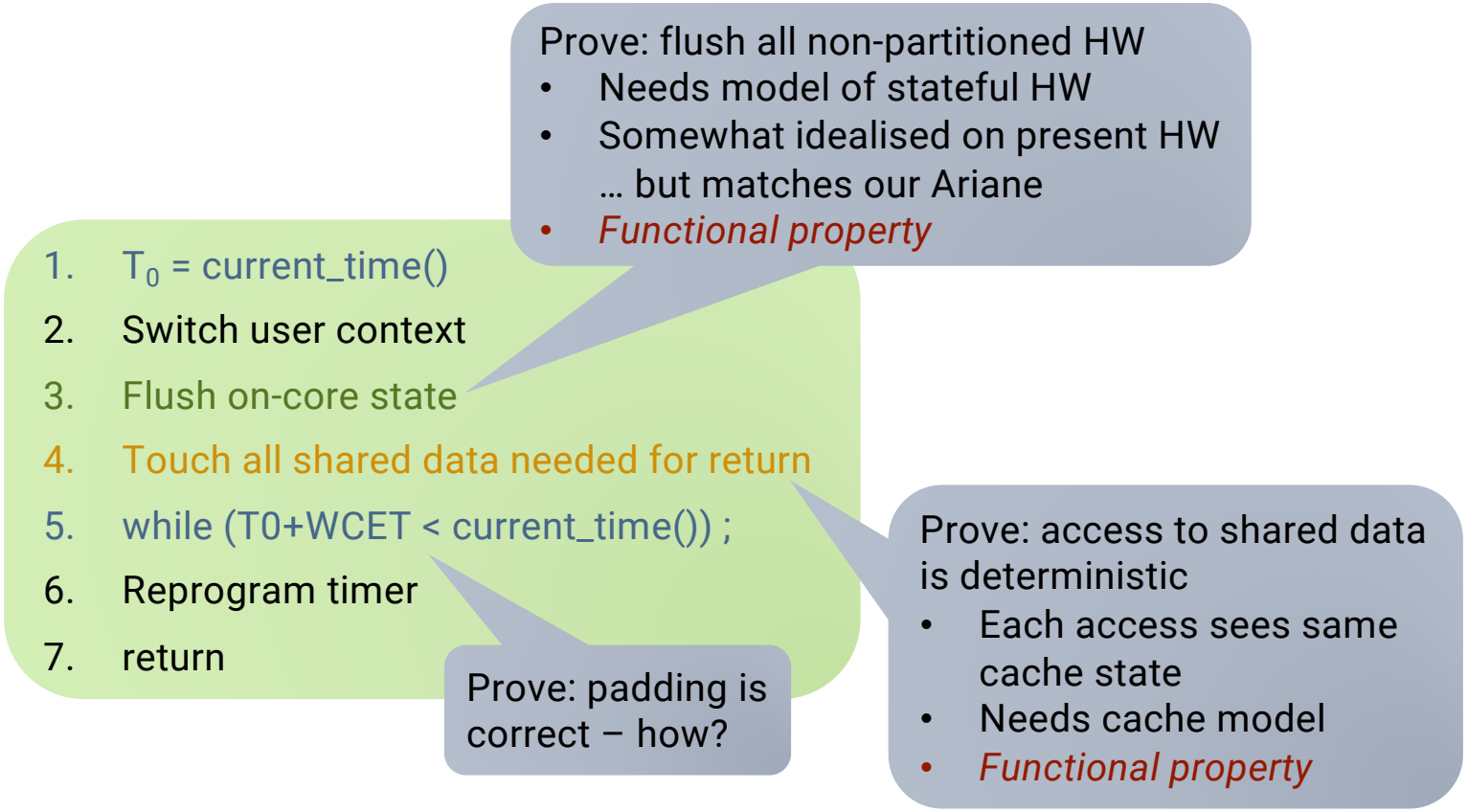
To prove: No two domains share hardware†

- 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



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