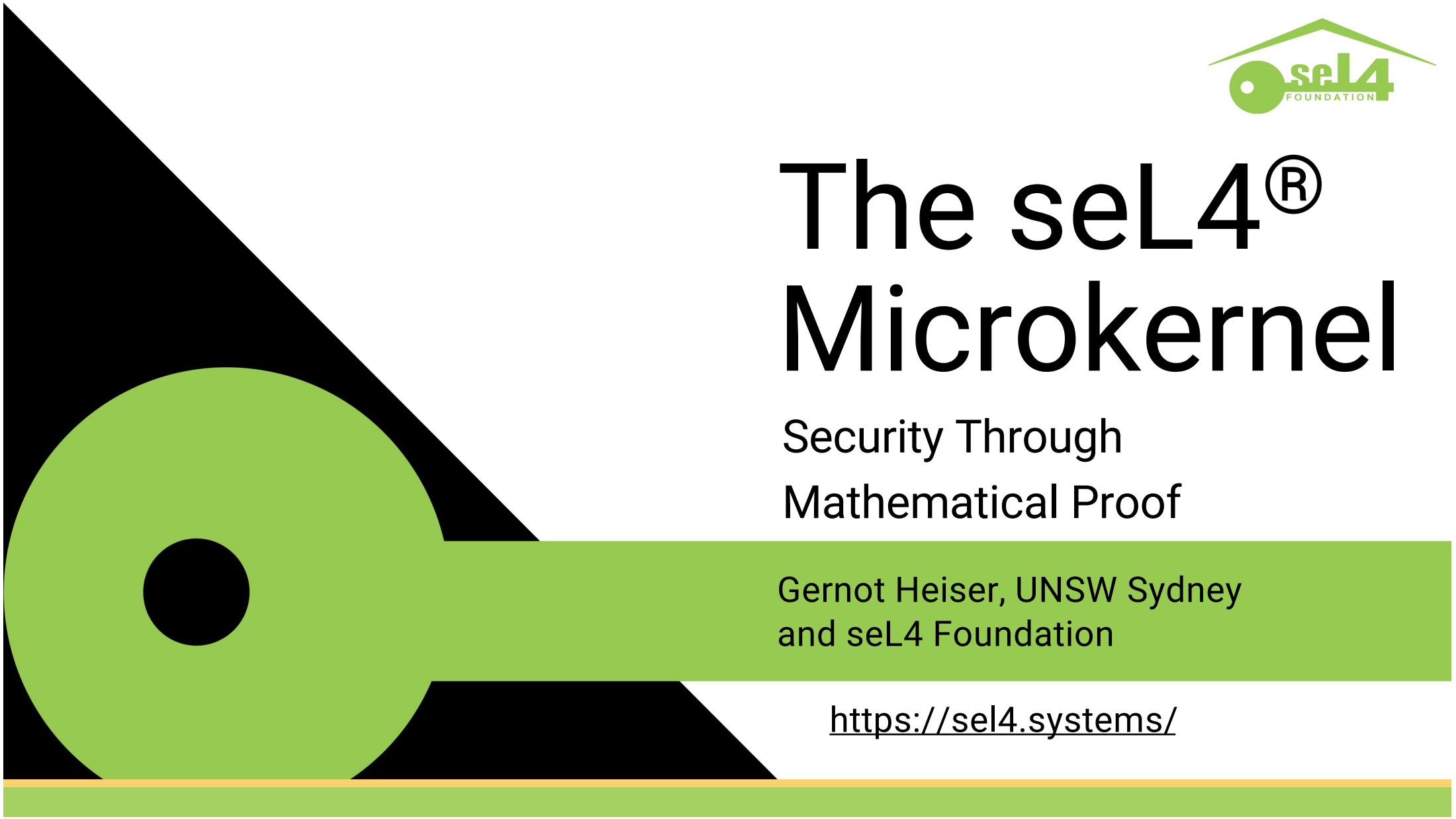


The seL4® Microkernel

Security Through Mathematical Proof

Gernot Heiser, UNSW Sydney
and seL4 Foundation

https://sel4.systems/

The Highlight of the Year

seL4 is verified on RISC-V!

2020/06/09

Sounds great! But what does it mean?

seL4

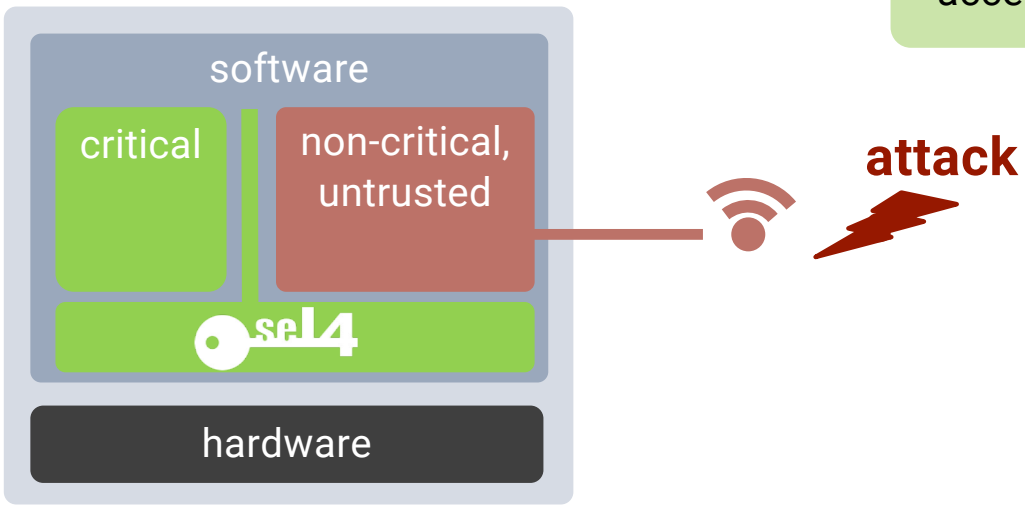
seL4 **(https://sel4.systems/)** (pronounced *ess-e-ell-four*) is arguably the world's most secure operating system (OS) kernel.

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

Background: What is seL4?

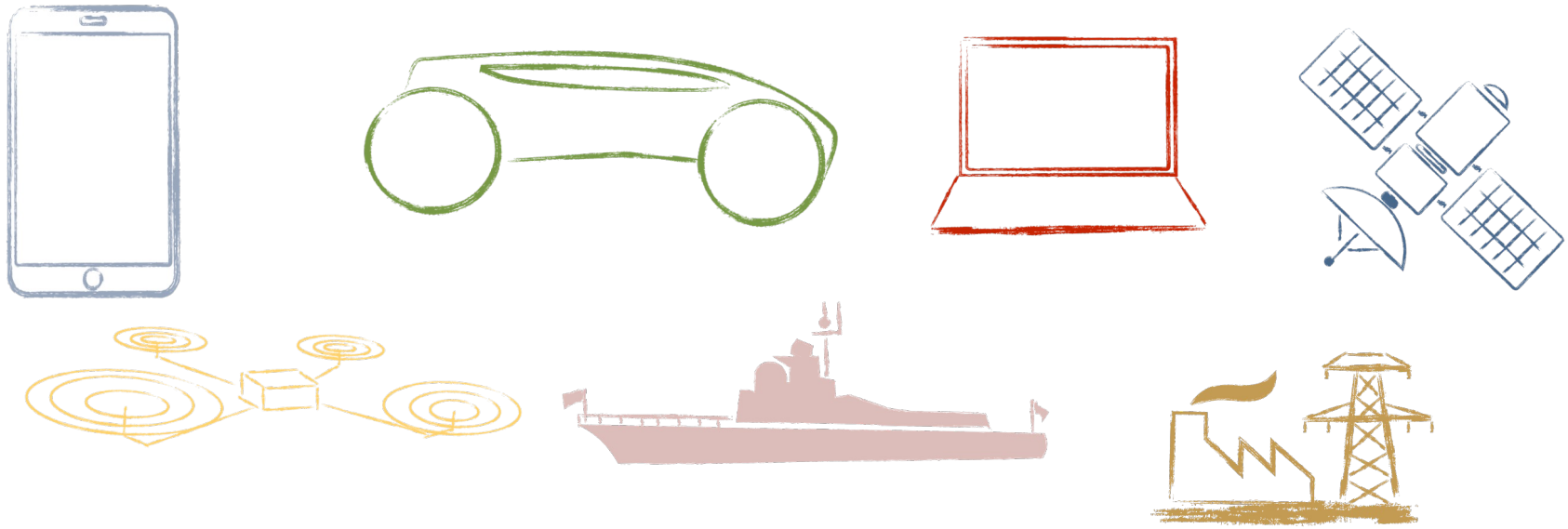
seL4 is an open source, high-assurance, high-performance operating system microkernel

- 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...

The Benchmark for Performance

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

Source	seL4	Fisco.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

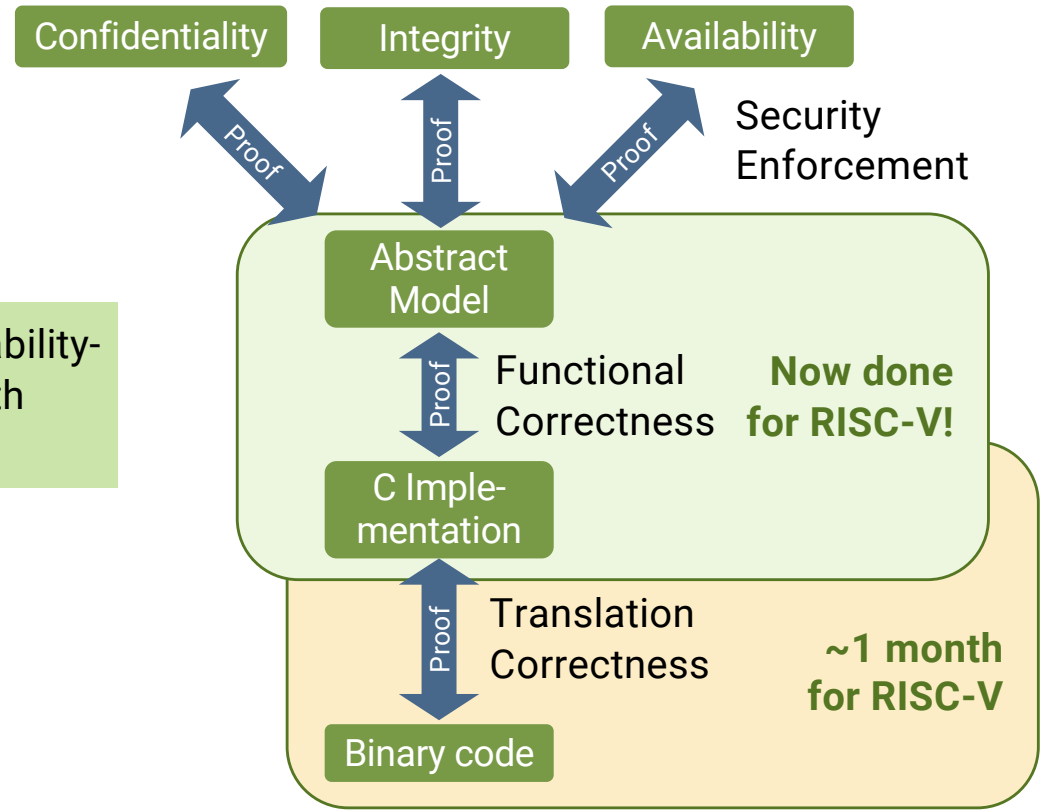
World's fastest microkernel!

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

Unique Verification by Mathematical Proof



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

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

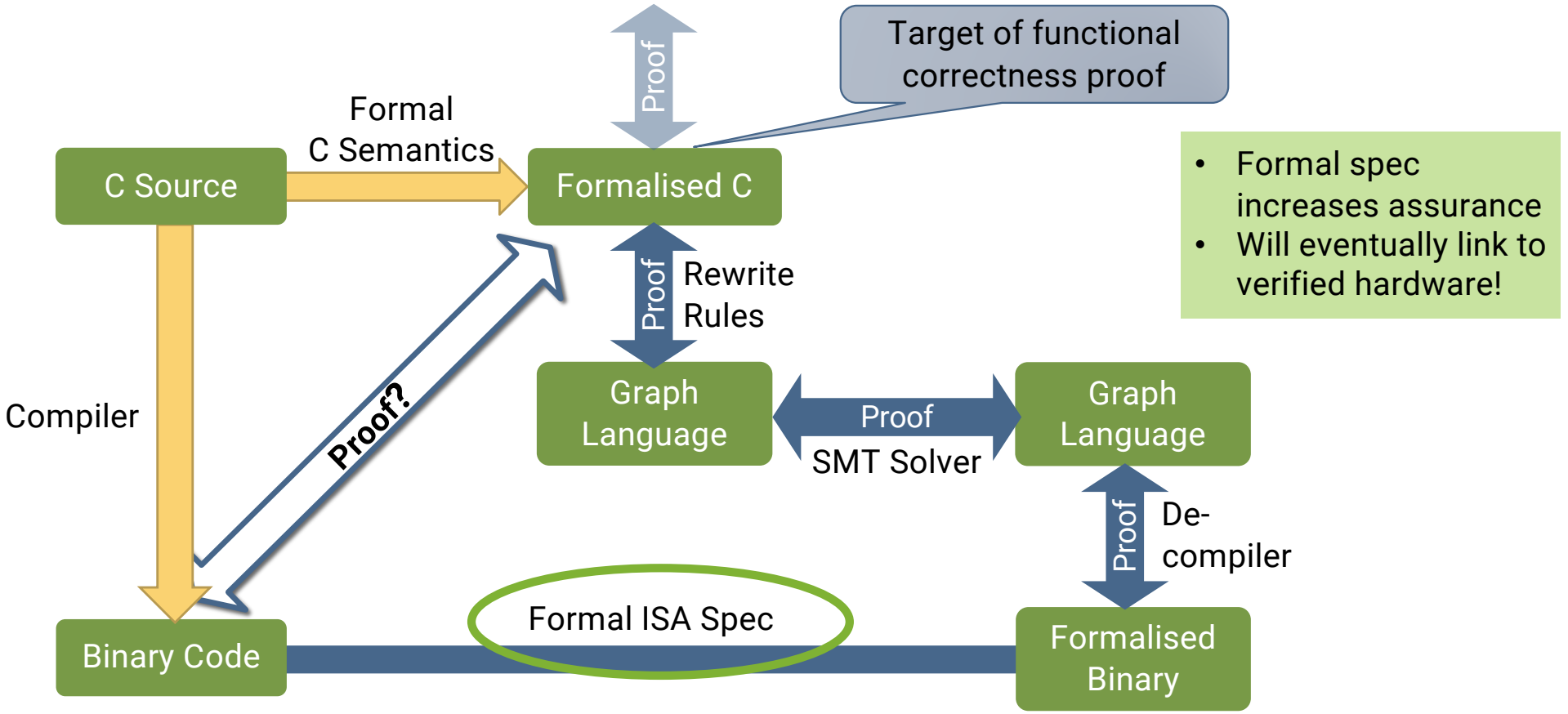
What Does This Mean?

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!

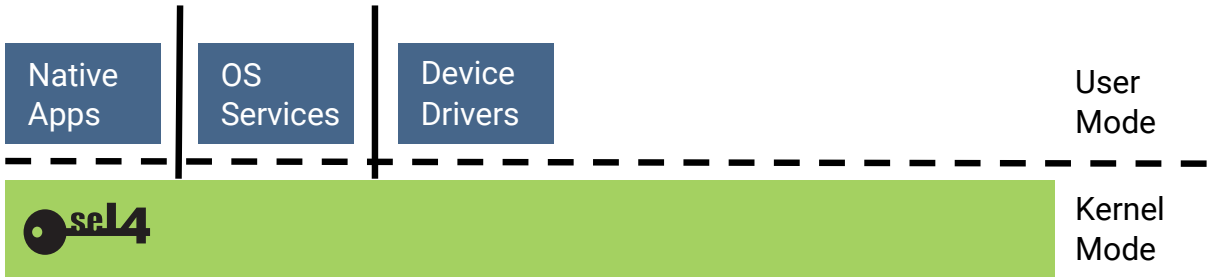
Verification of Binary (RISC-V in Progress)



- Formal spec increases assurance
- Will eventually link to verified hardware!

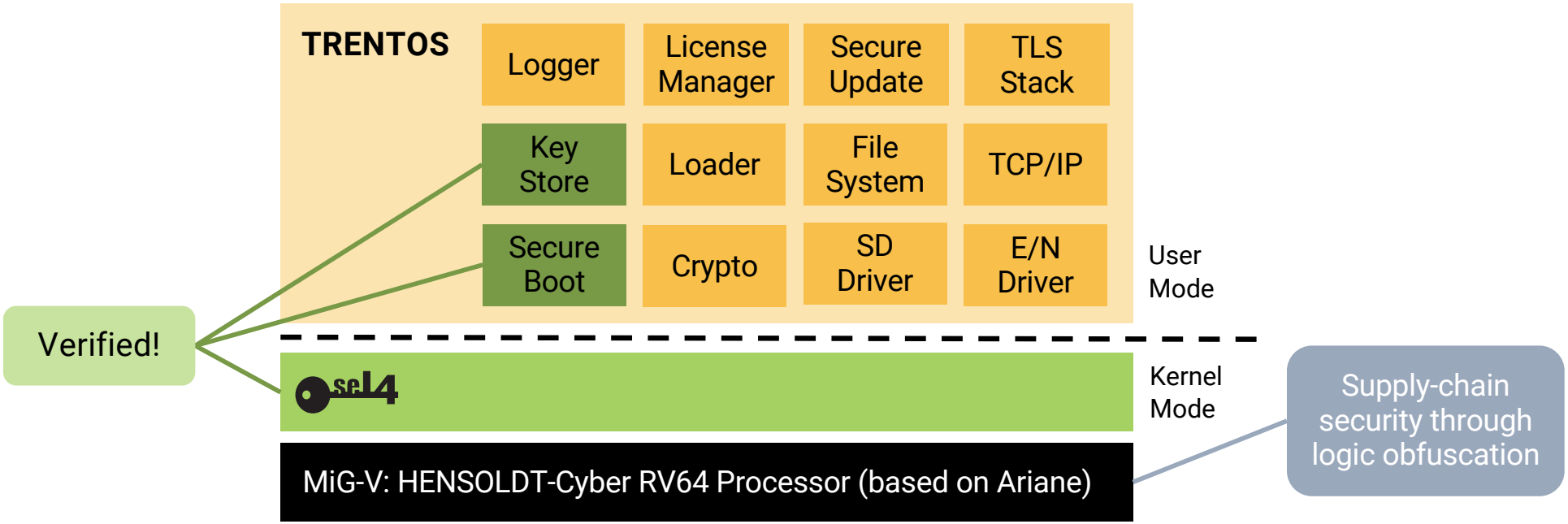
How Can I Use It?

- ✓ 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

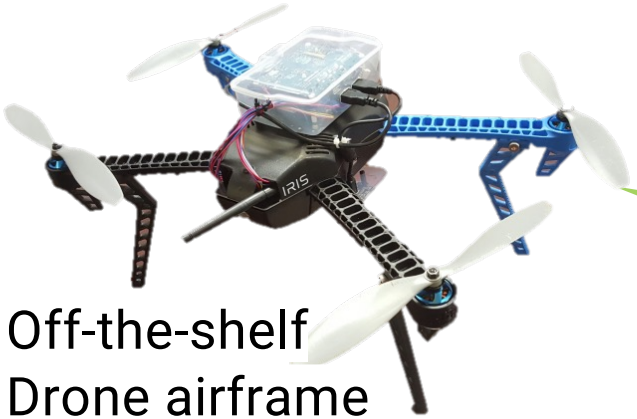


Incremental Cyber-Retrofit: DARPA HACMS

Unmanned Little Bird (ULB)

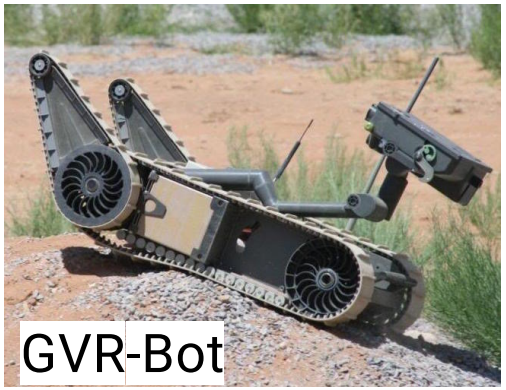
Retrofit existing system!

Autonomous trucks



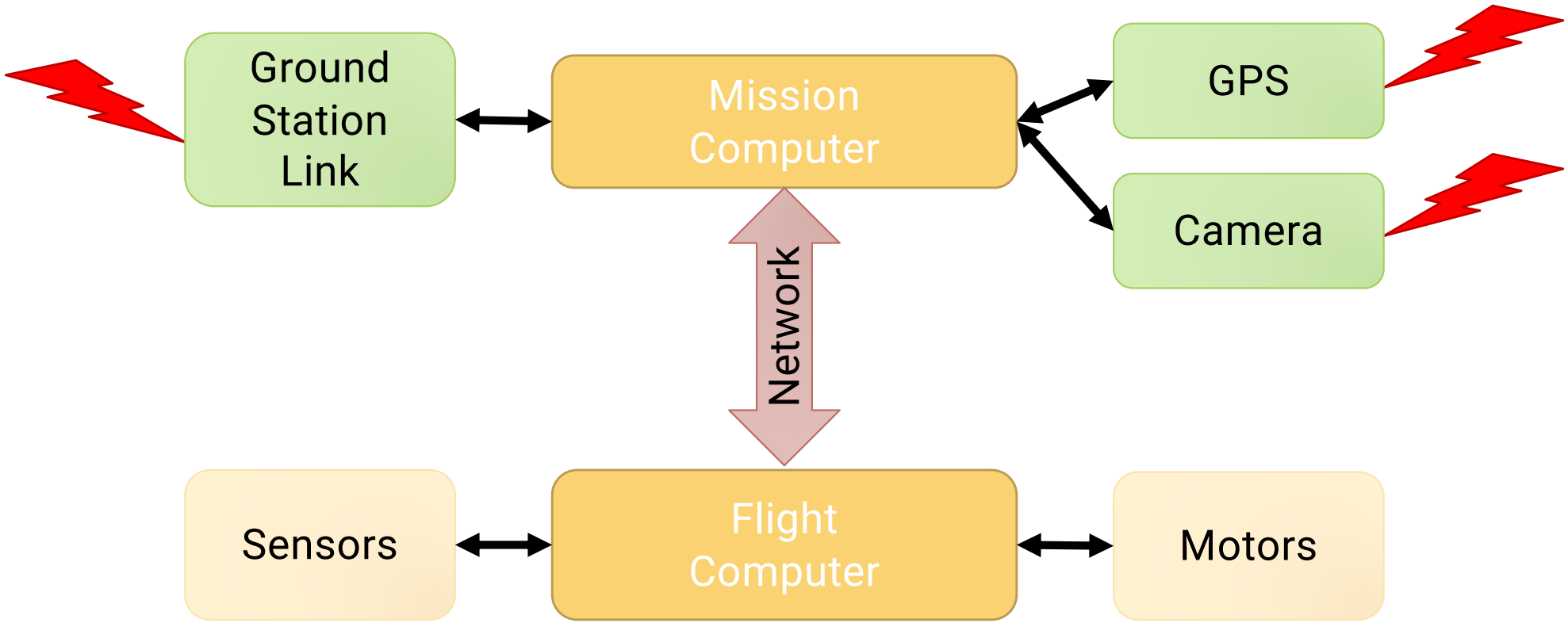
Off-the-shelf Drone airframe

Develop technology

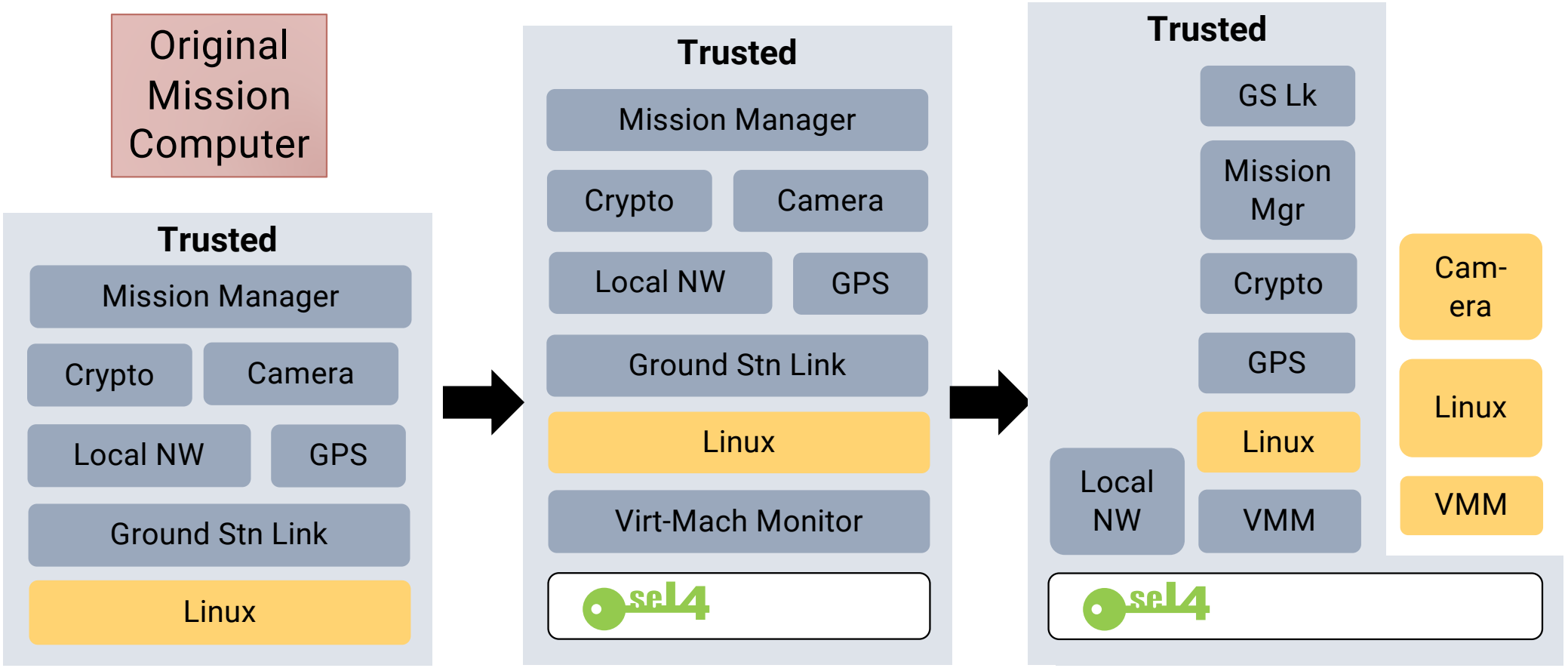


GVR-Bot

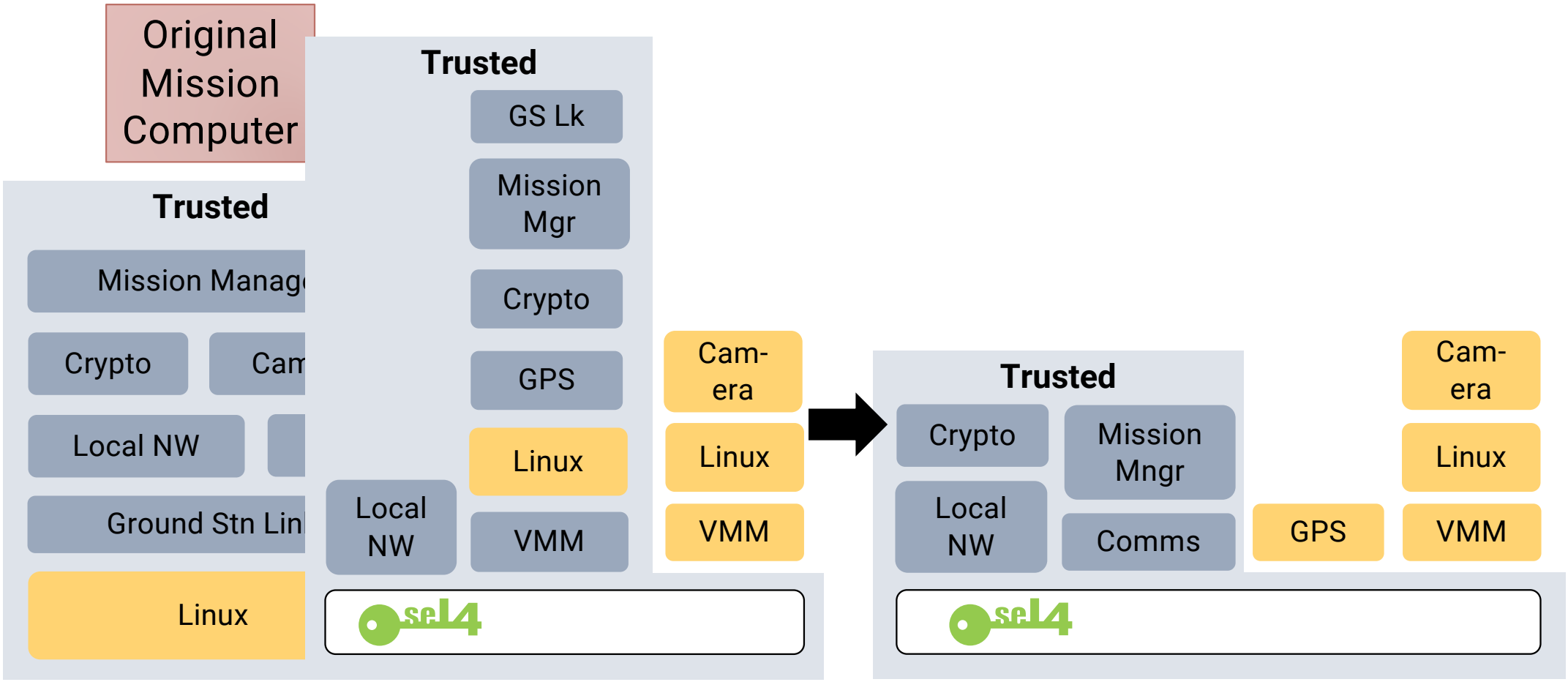
ULB Architecture



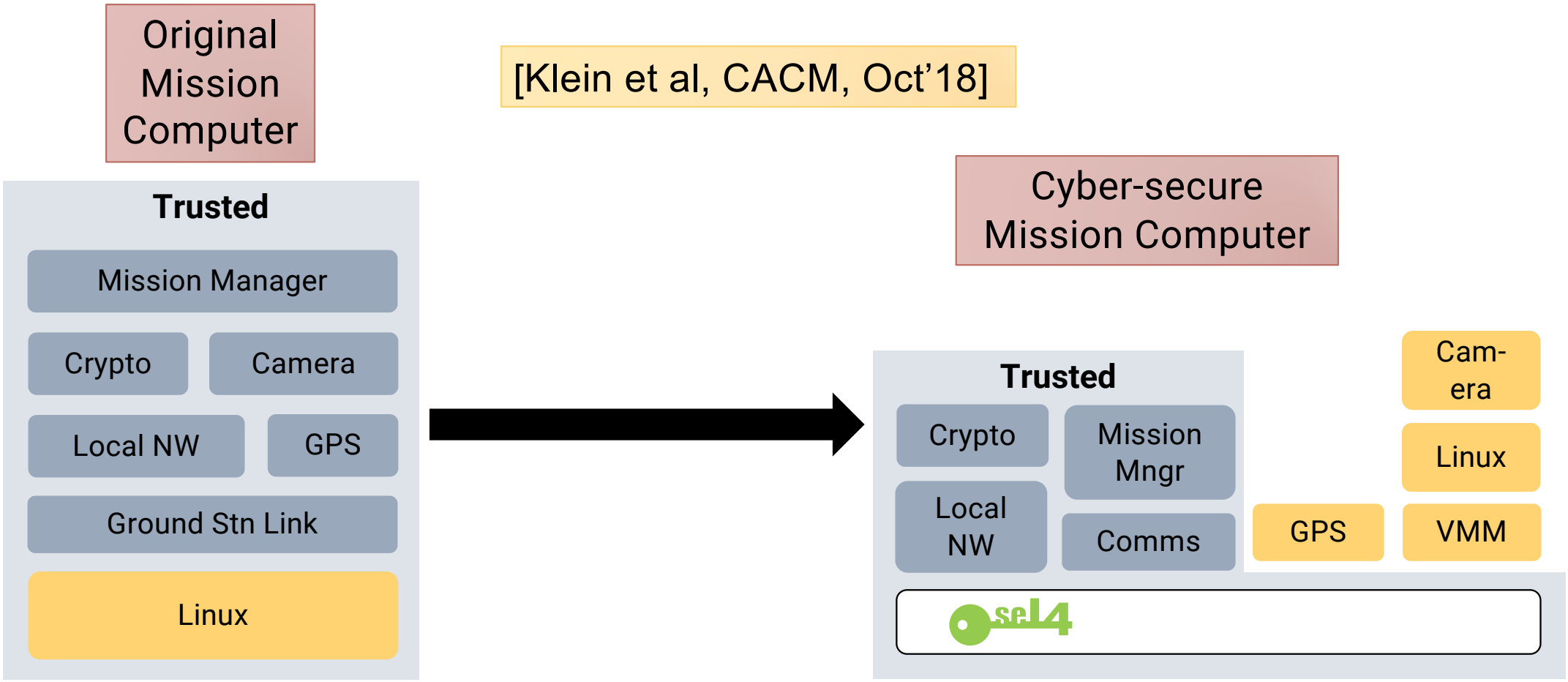
Incremental Cyber Retrofit



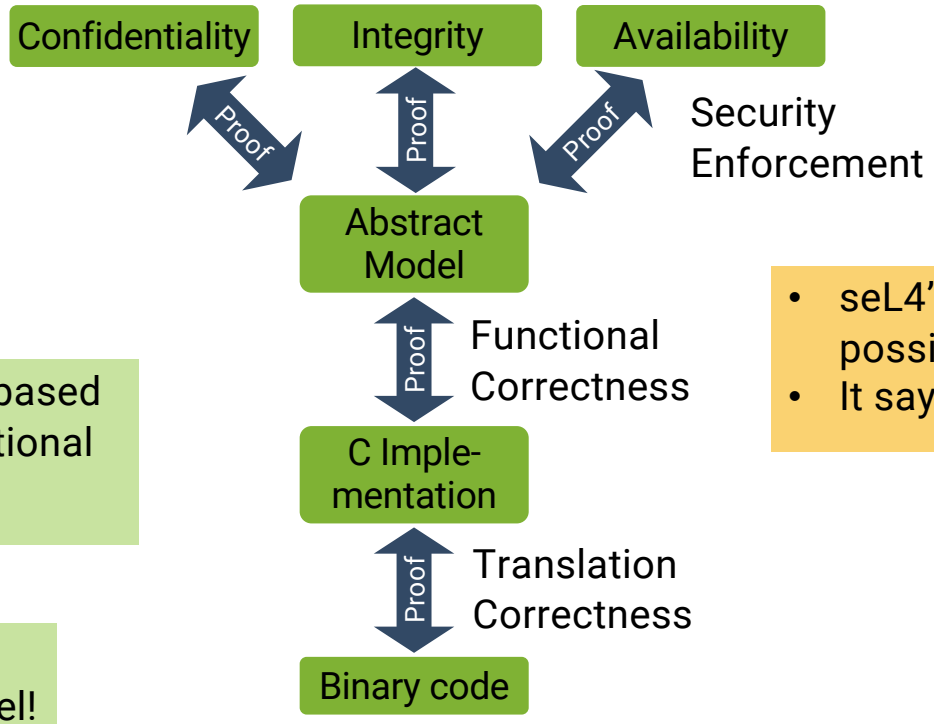
Incremental Cyber Retrofit



Incremental Cyber Retrofit



So, Why Aren't We Done?



Still only capability-based OS kernel with functional correctness proof

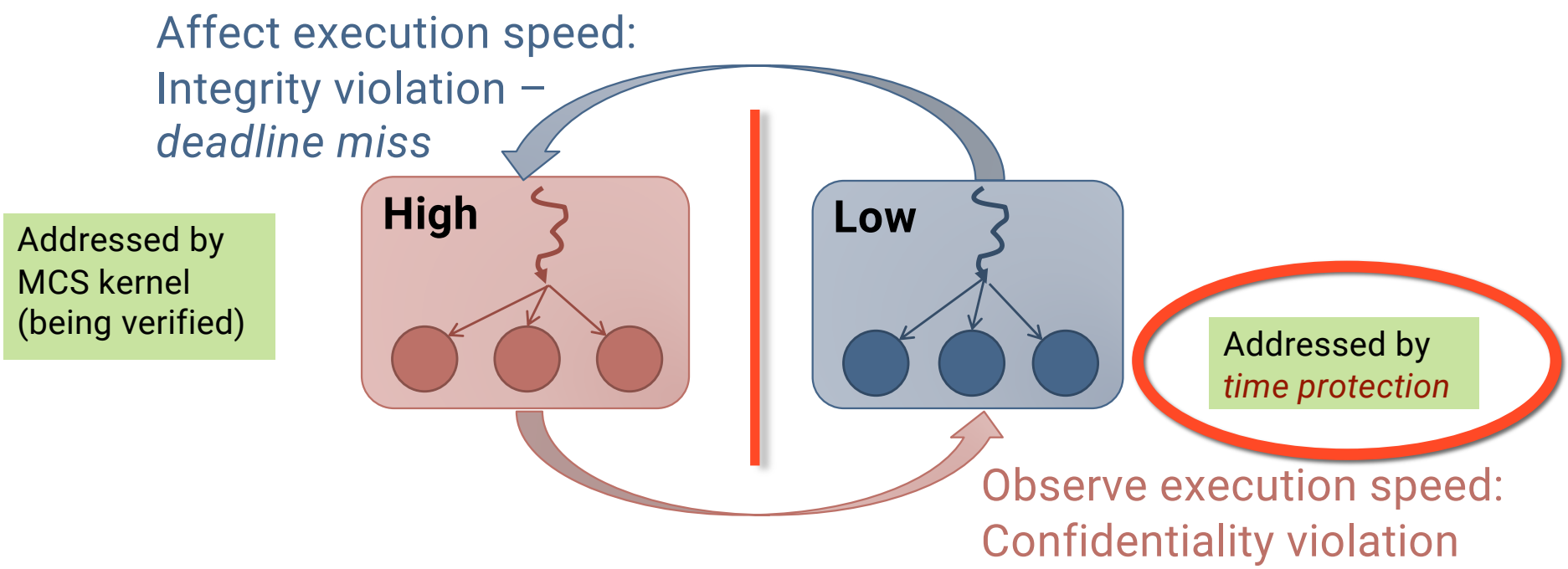
Still the world's fastest microkernel!

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

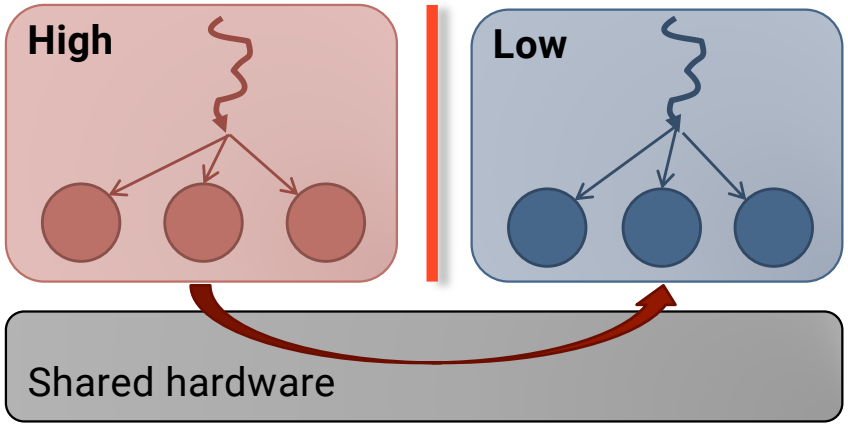
What's the Issue with Temporal Isolation?

Safety: Timeliness
- Execution interference

Security: Confidentiality
- Leakage via timing channels



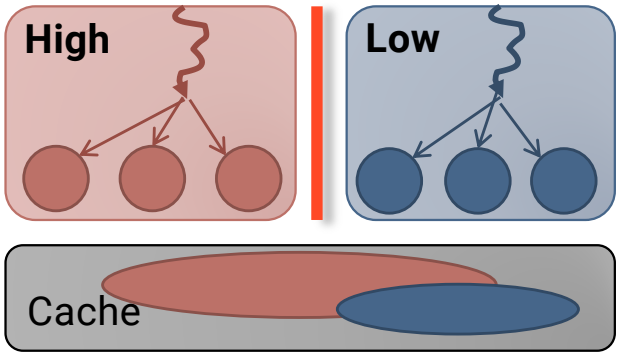
Cause: Competition for HW Resources



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

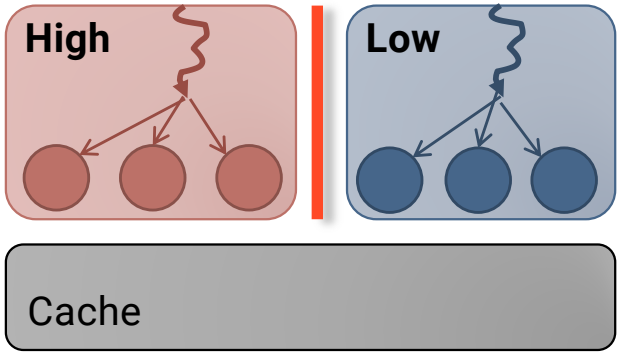
Solution: *Time Protection* – Eliminate interference by preventing sharing

Time Protection: Partition all Hardware State

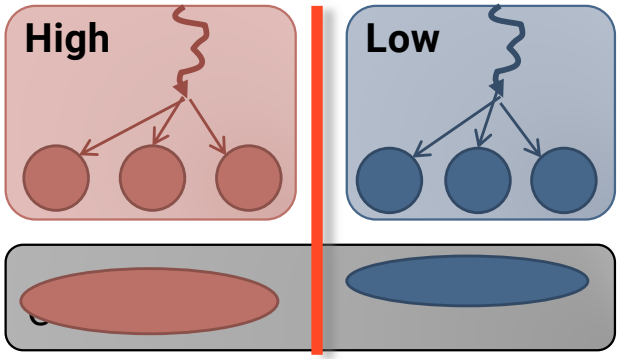


Temporally partition

Flush



Spatially partition



Need both!

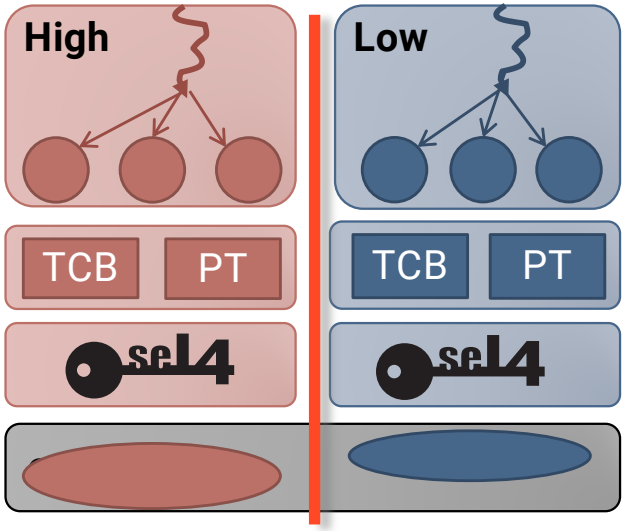
Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)

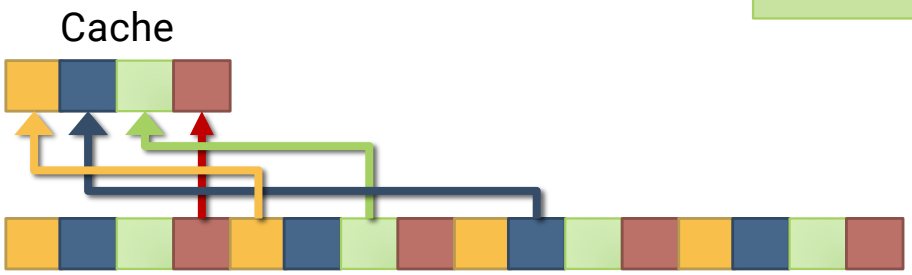
- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

- HW threads
- cores

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!

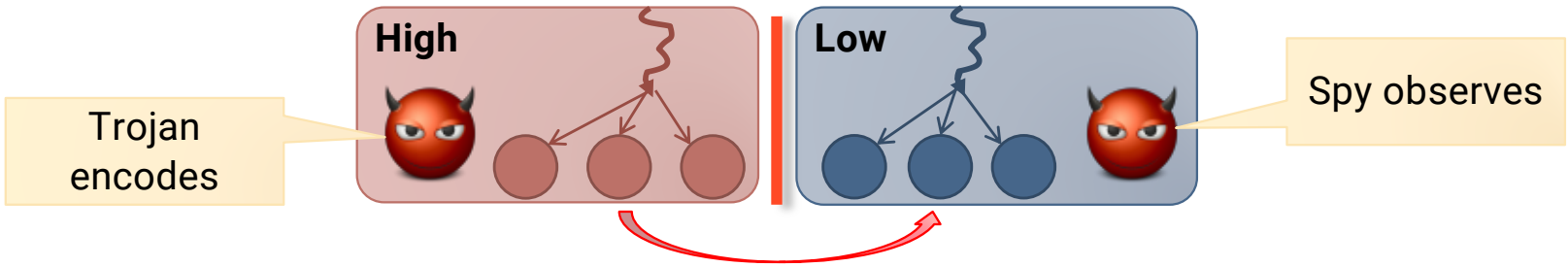
1. T_0 = current_time()
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while (T0+WCET < current_time()) ;
6. Reprogram timer
7. return

Latency depends on prior execution!

Ensure deterministic execution

Time padding to remove dependency

Evaluation: Prime & Probe Attack



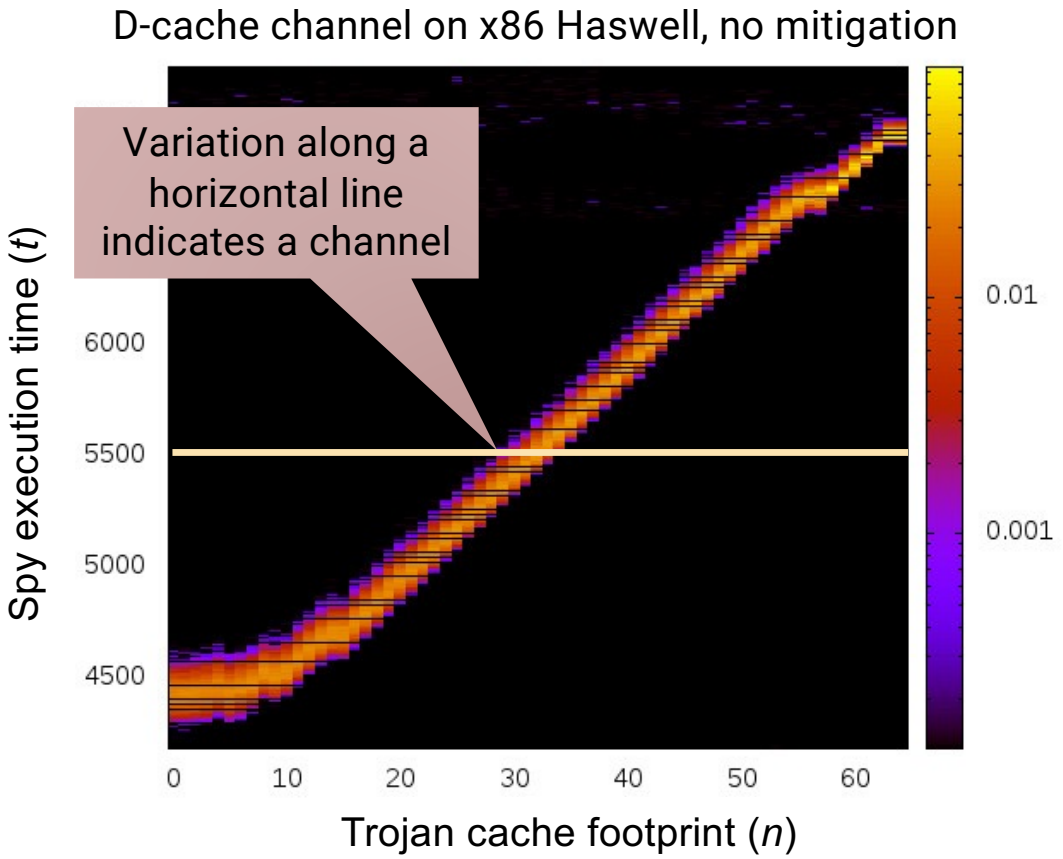
1. Fill cache with own data
2.
3. Traverse cache, measure *execution time*

Output signal

2. Touch *n* cache lines

Input signal

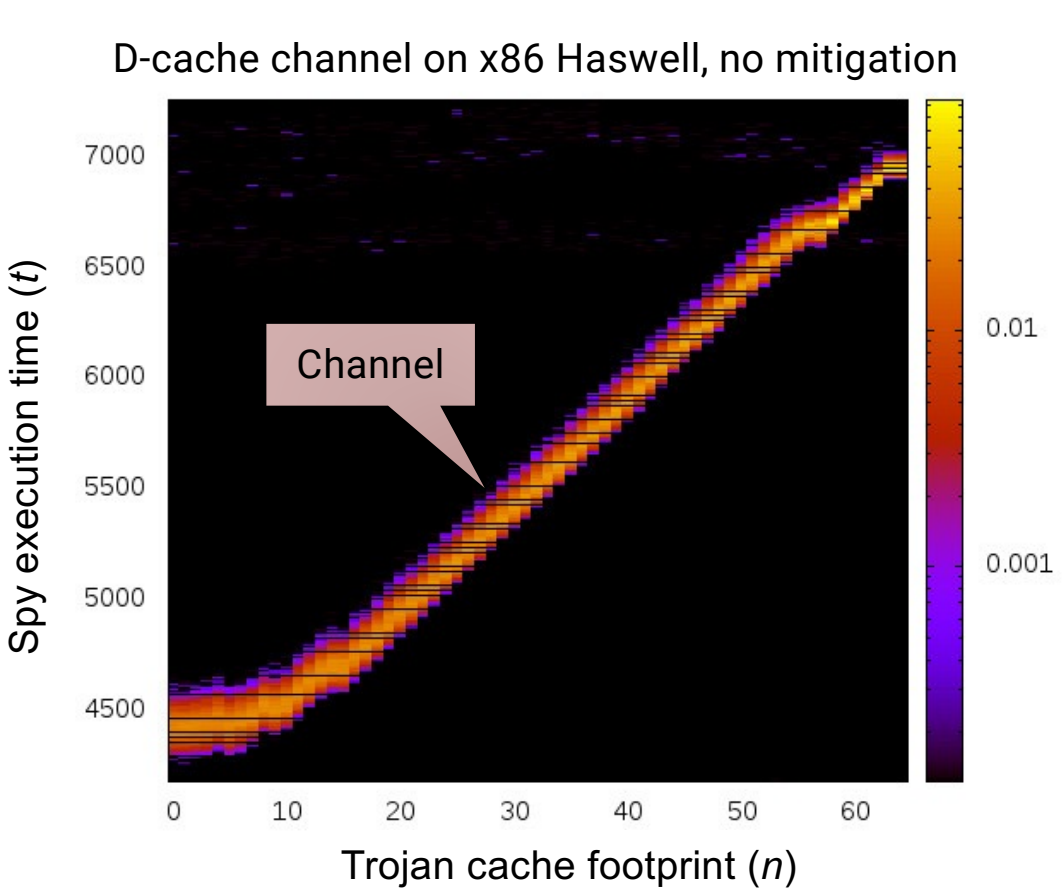
Methodology: Channel Matrix

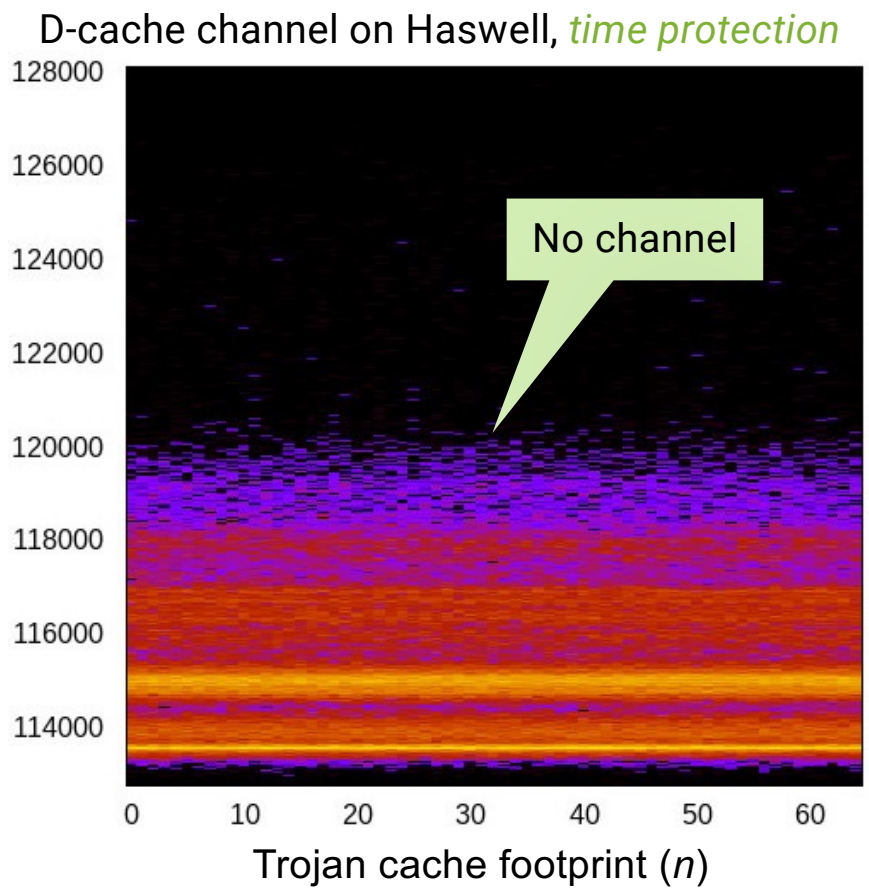


Channel matrix:

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

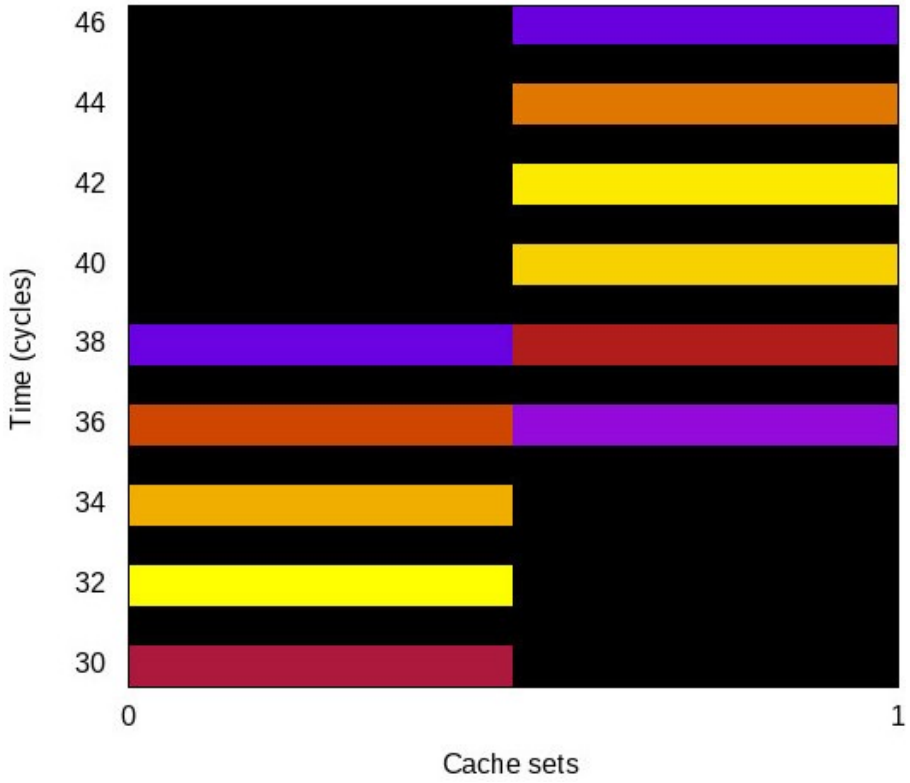
Applying Time Protection



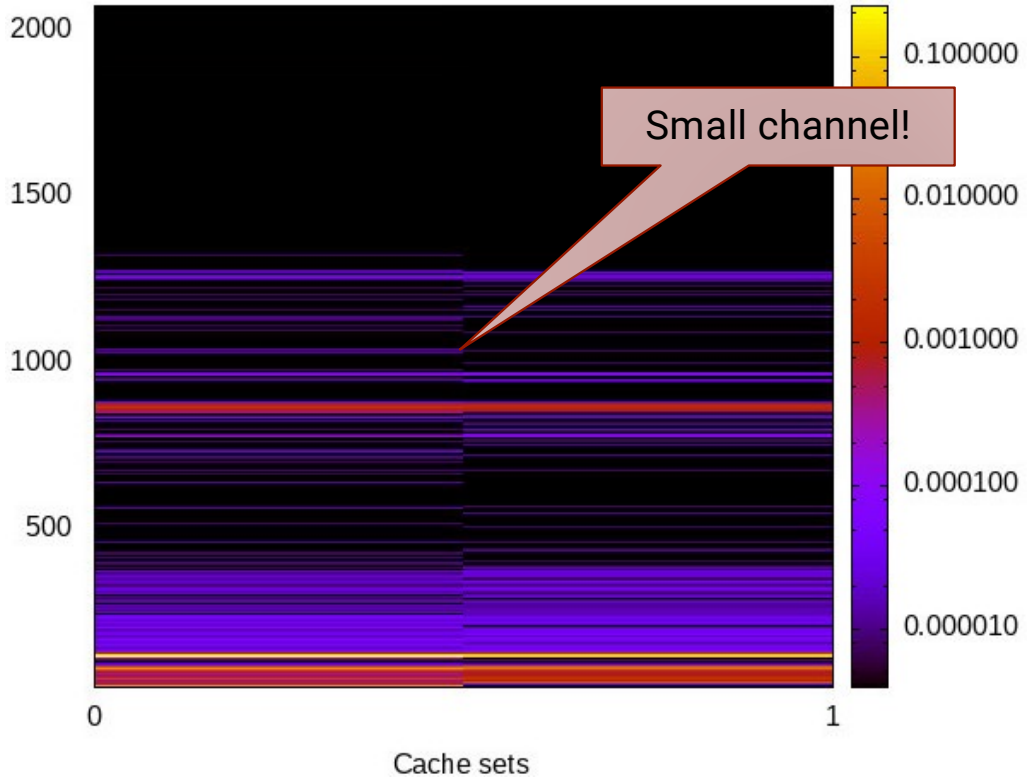


Challenge: Broken Hardware

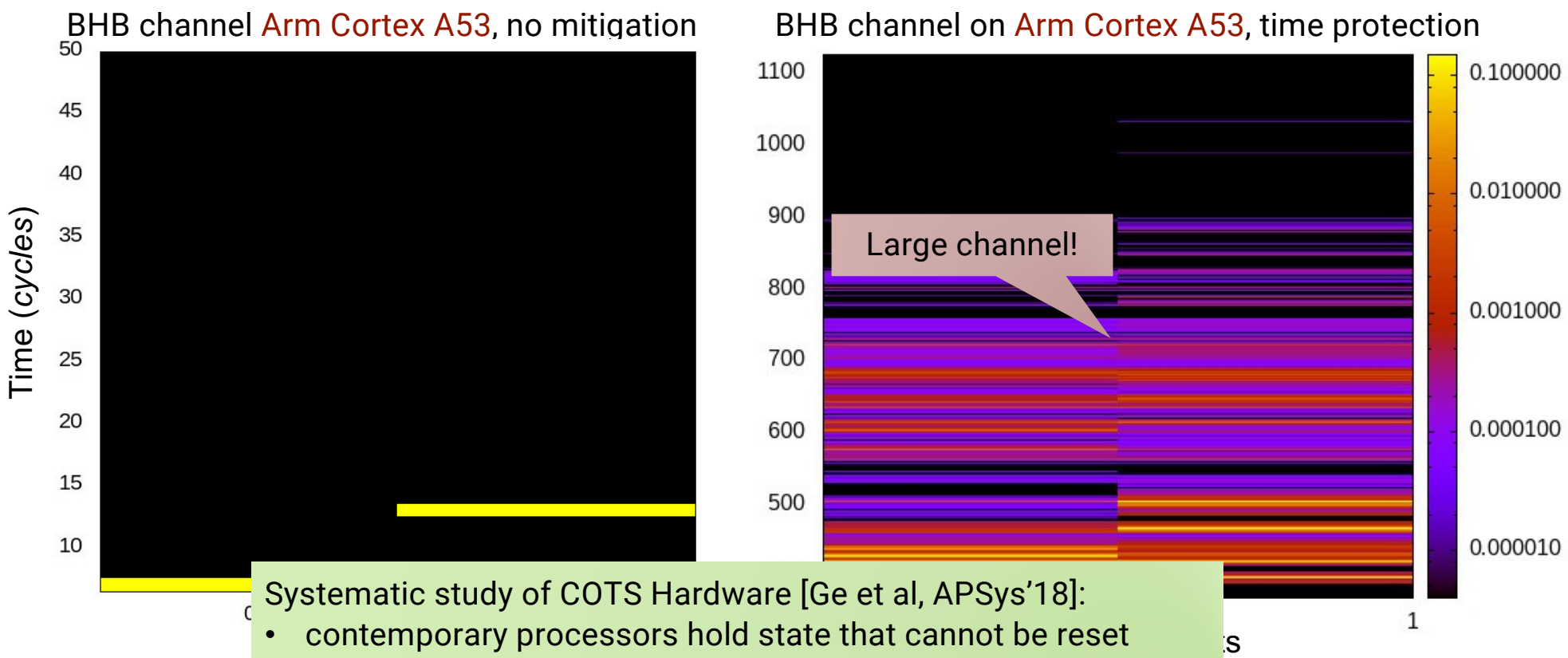
BHB channel on x86 Sky Lake, no mitigation



BHB channel on x86 Sky Lake, time protection



Challenge: Broken Hardware



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

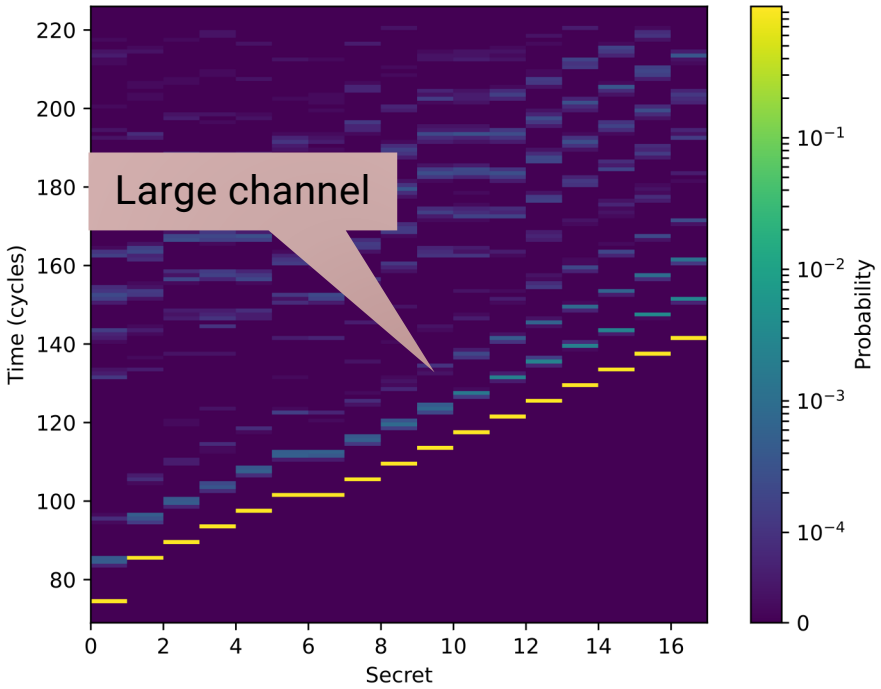
- contemporary processors hold state that cannot be reset
- need a new hardware-software contract to enable real security

RISC-V To The Rescue!

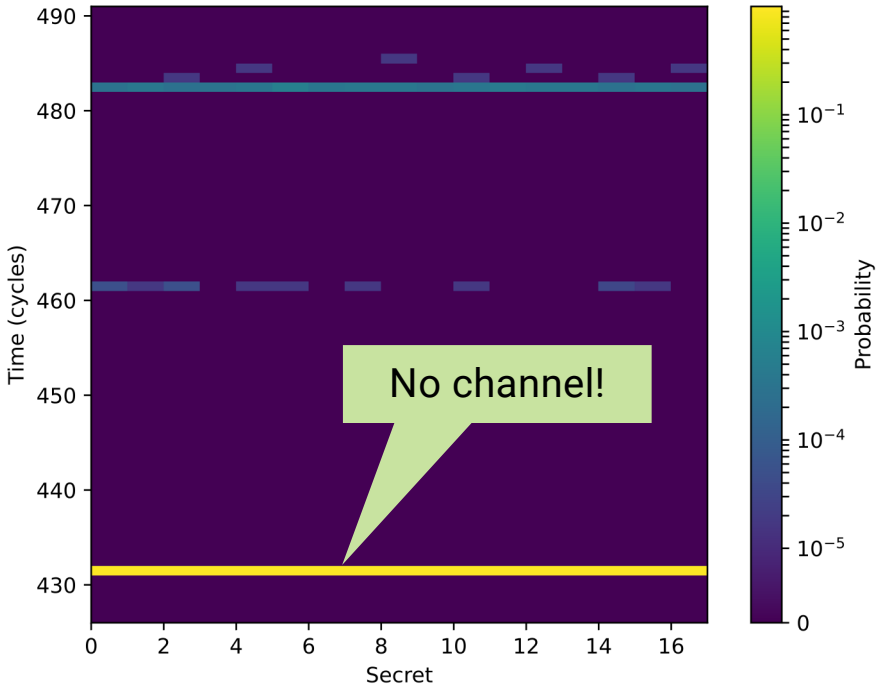
Implemented flush of *all* microarchitectural state in ETH Ariane processor and evaluated channels on FPGA implementation

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

BHB channel Ariane, no mitigation



BHB channel Ariane, time protection



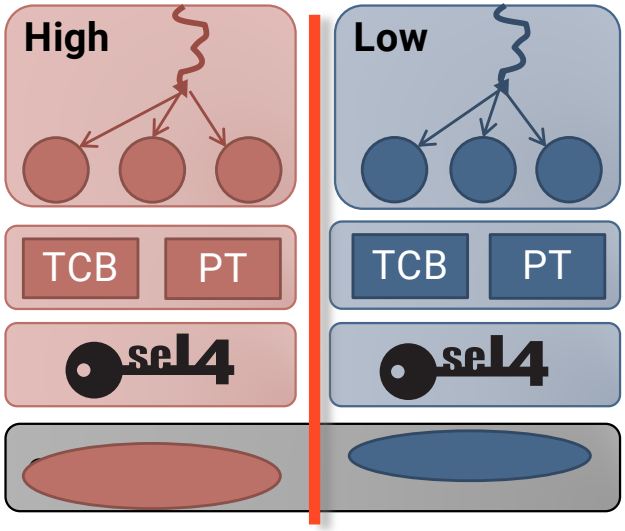
Can We Verify Time Protection?

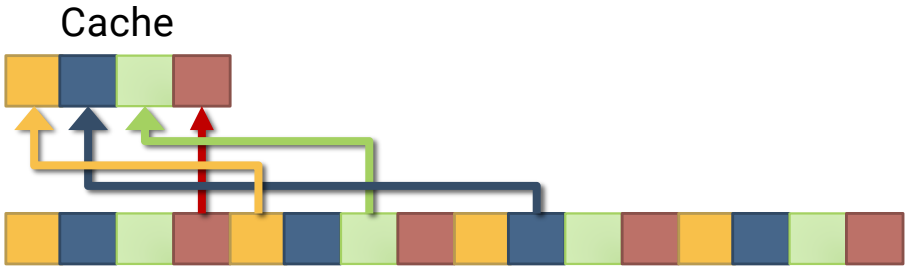
Assume we have:

- 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





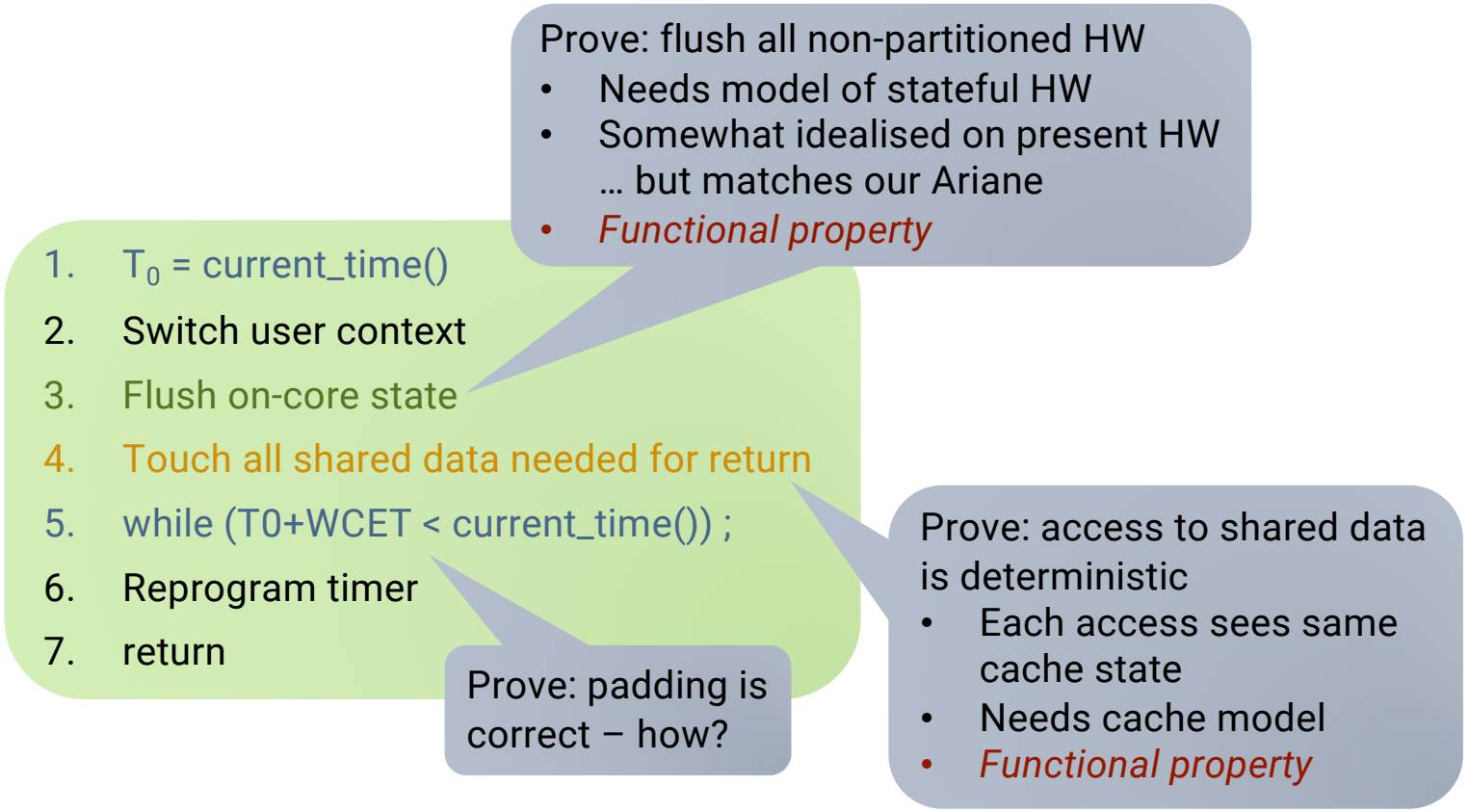
To prove: No two domains share hardware†

- 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



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!*

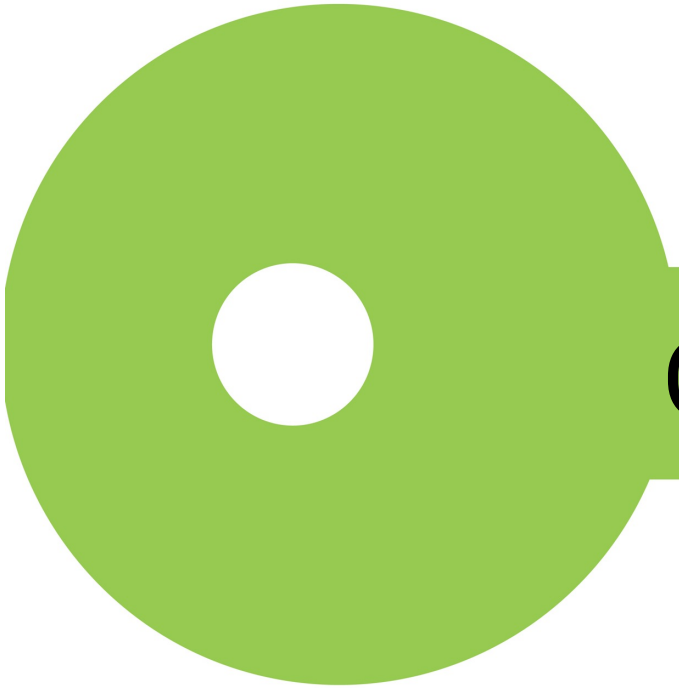
Status

- ✓ 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 (DATE'21)
  - 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

Defining the state of the art
in trustworthy operating systems
for over 10 years
Now proved correct on RISC-V!

Further Reading:

- About seL4:
  https://sel4.systems/
- seL4 whitepaper:
  https://sel4.systems/About/seL4-whitepaper.pdf
- seL4 Foundation:
  https://sel4.systems/Foundation



Questions?