The Formally Verified seL4 Microkernel

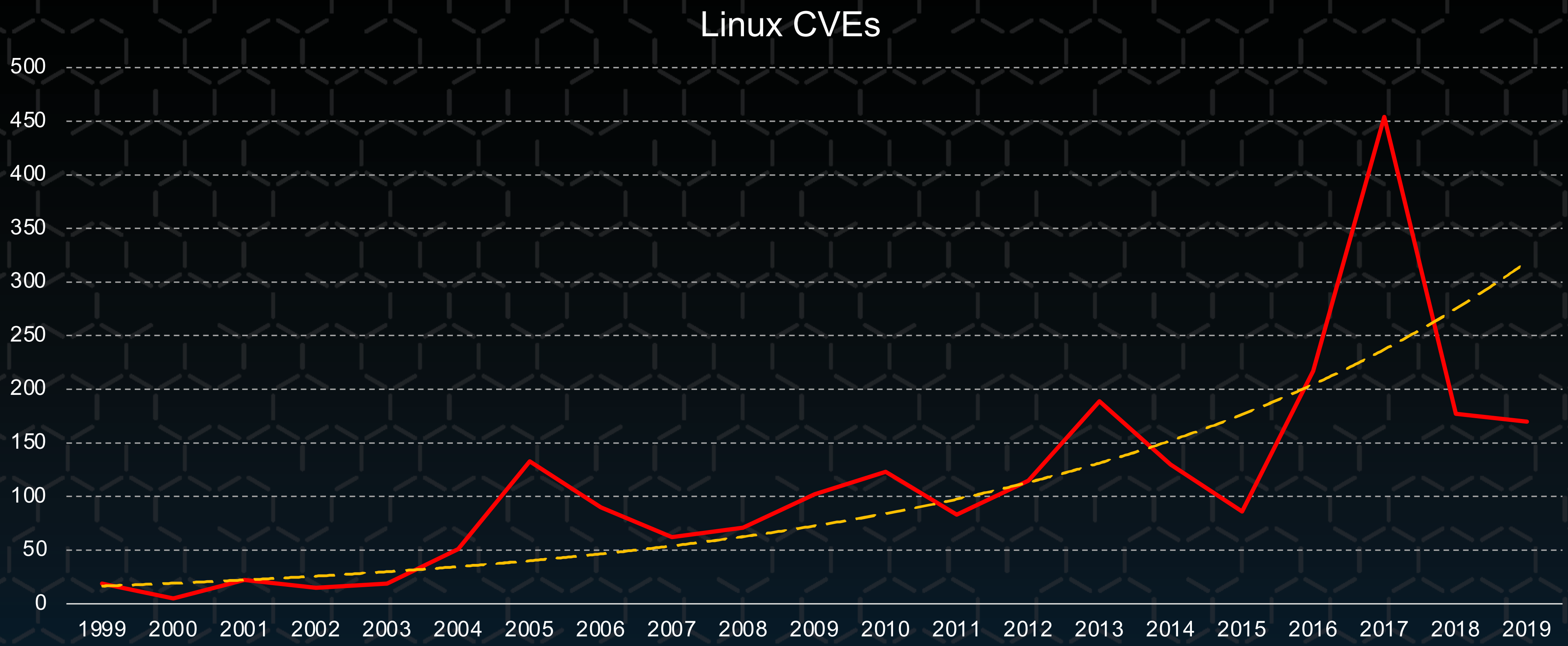
Present and Future

Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser

- Multicore World, Wellington NZ, Feb'20

https://trustworthy.systems

Linux Vulnerabilities Over Time



Source: https://cvedetails.com

A 30-Year Dream: Prove the OS Correct

Operating Systems — R. Stockton Gaines, Editor

Specification and Verification of the UCLA Unix† Security Kernel

Bruce J. Walker, Richard A. Kemmerer, and Gerald J. Popek
University of California, Los Angeles

Data Secure Unix, a kernel structured operating system, was constructed as part of an ongoing effort at UCLA to develop procedures by which operating systems can be produced and shown secure. Program verification methods were extensively applied as a constructive means of demonstrating security enforcement.

Here we report the specification and verification experience in producing a secure operating system. The work represents a significant attempt to verify a large-scale, production level software system, including all aspects from initial specification to verification of implemented code.

Key Words and Phrases: verification, security, operating systems, protection, programming methodology, ALPHARD, formal specifications, Unix, security kernel

CR Categories: 4.29, 4.35, 6.35

1. Introduction

Early attempts to make operating systems secure merely found and fixed flaws in existing systems. As these efforts failed, it became clear that piecemeal alterations were unlikely ever to succeed [20]. A more systematic method was required, presumably one that controlled the system's design and implementation. Then secure operation could be demonstrated in a stronger sense than an ingenuous claim that the last bug had been eliminated, particularly since production systems are rarely static, and errors easily introduced.

Our research seeks to develop means by which an operating system can be shown data secure, meaning that direct access to data must be possible only if the recorded protection policy permits it. The two major components of this task are: (1) developing system architectures that minimize the amount and complexity of software involved in both protection decisions and enforcement, by isolating them into *kernel* modules; and (2) applying extensive verification methods to that kernel software in order to prove that our of *data security* criterion is met. This paper reports on the latter part, the verification experience. Those interested in architectural issues should see [23]. Related work includes the PSOS operating system project at SRI [25] which uses the hierarchical design methodology described by Robinson and Levitt in [26], and efforts to prove communications software at the University of Texas [31].

Every verification step, from the development of top-level specifications to machine-aided proof of the Pascal code, was carried out. Although these steps were not completed for all portions of the kernel, most of the job was done for much of the kernel. The remainder is clearly more of the same. We therefore consider the project essentially complete. In this paper, as each verification step is discussed, an estimate of the completed portion of that step is given, together with an indication of the amount of work required for completion. One should realize that it is essential to carry the verification process through the steps of actual code-level proofs because most security flaws in real systems are found at this level [20]. Security flaws were found in our system during verification, despite the fact that the implementation was written carefully and tested extensively. An example of

Our research seeks to develop means by which an operating system can be shown data secure, meaning that direct access to data must be possible only if the recorded protection policy permits it. The two major components

Communications of the ACM
February 1980
Volume 23
Number 2

seL4: The Dream Come True

The world's **first** operating-system kernel with **provable** security enforcement

World's most advanced mixed-criticality OS

Open Source

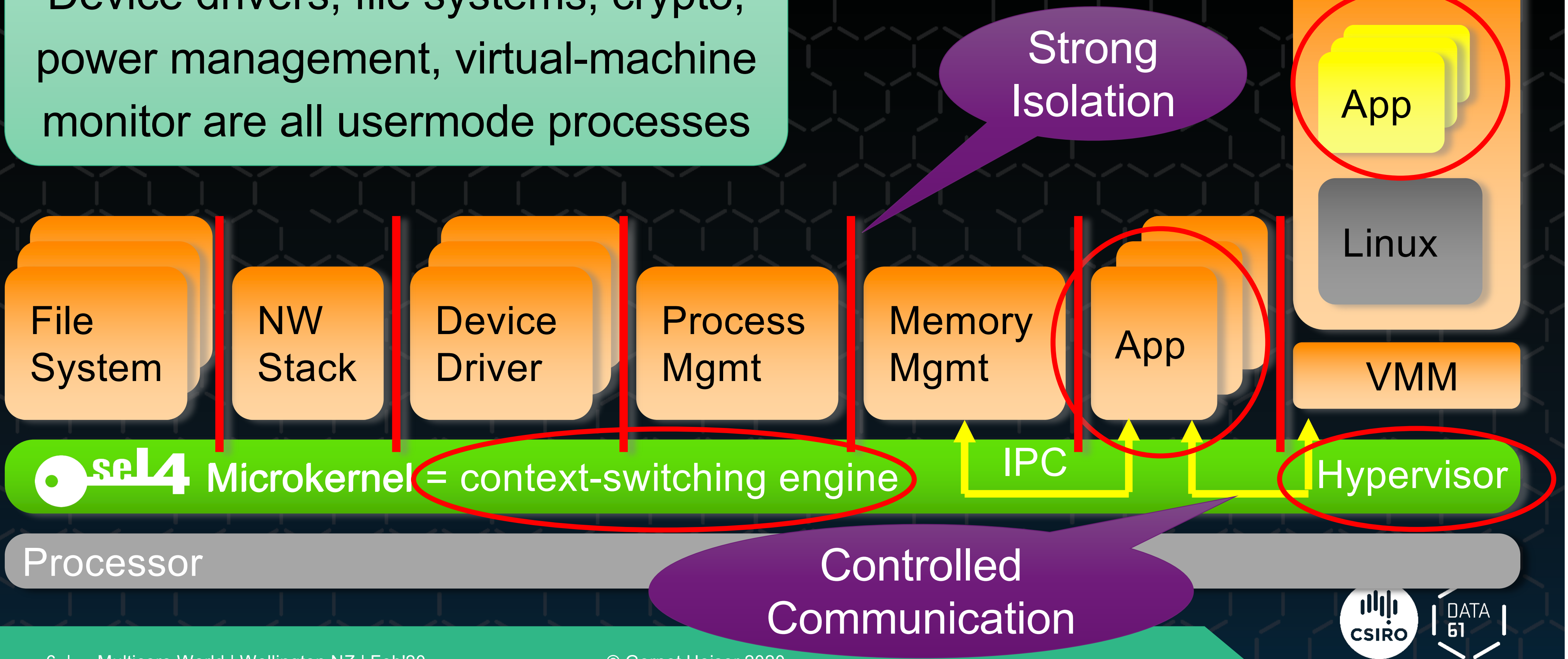
The world's **only** protected-mode OS with complete, sound timeliness analysis

The world's **fastest** general-purpose microkernel, designed for **real-world** use

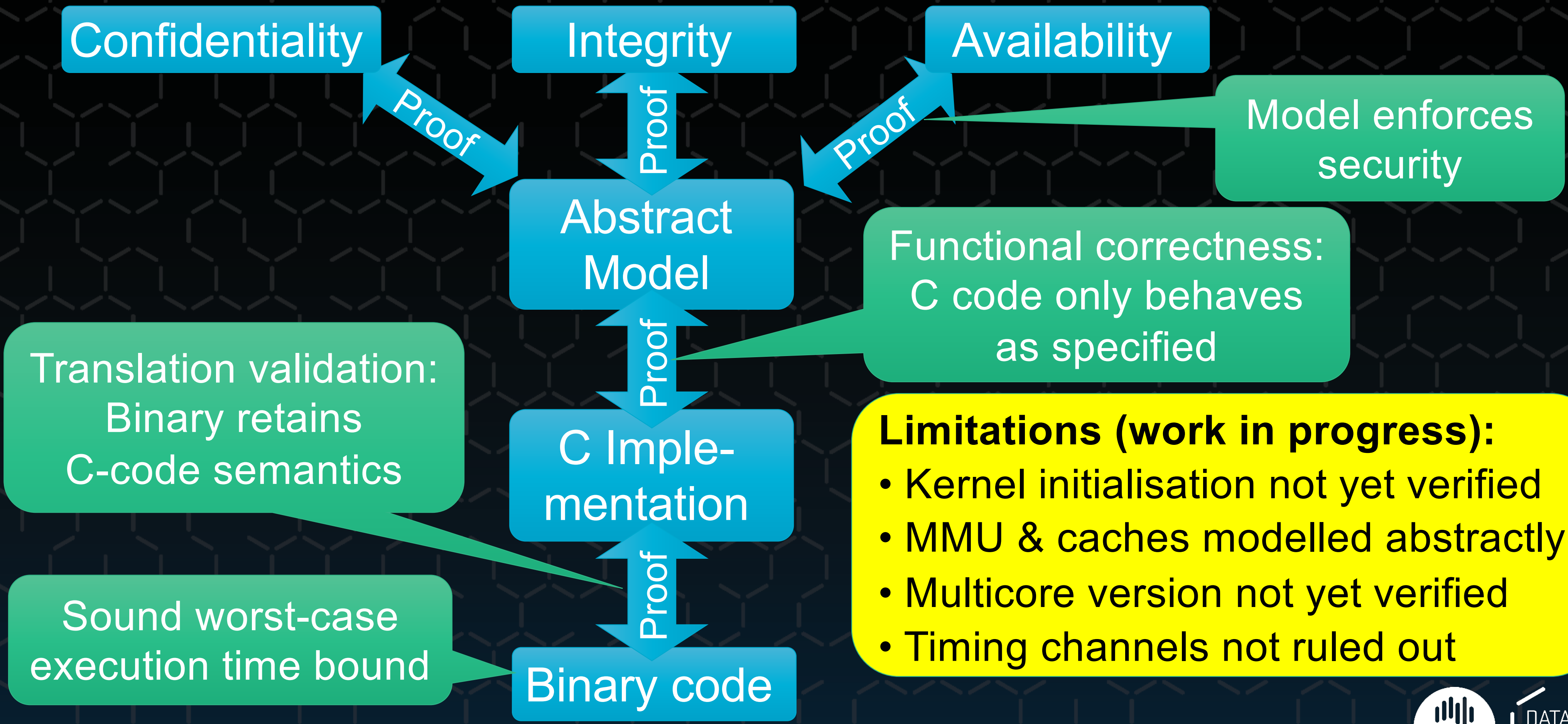
What is seL4?

A Microkernel is not an OS

Device drivers, file systems, crypto, power management, virtual-machine monitor are all usermode processes



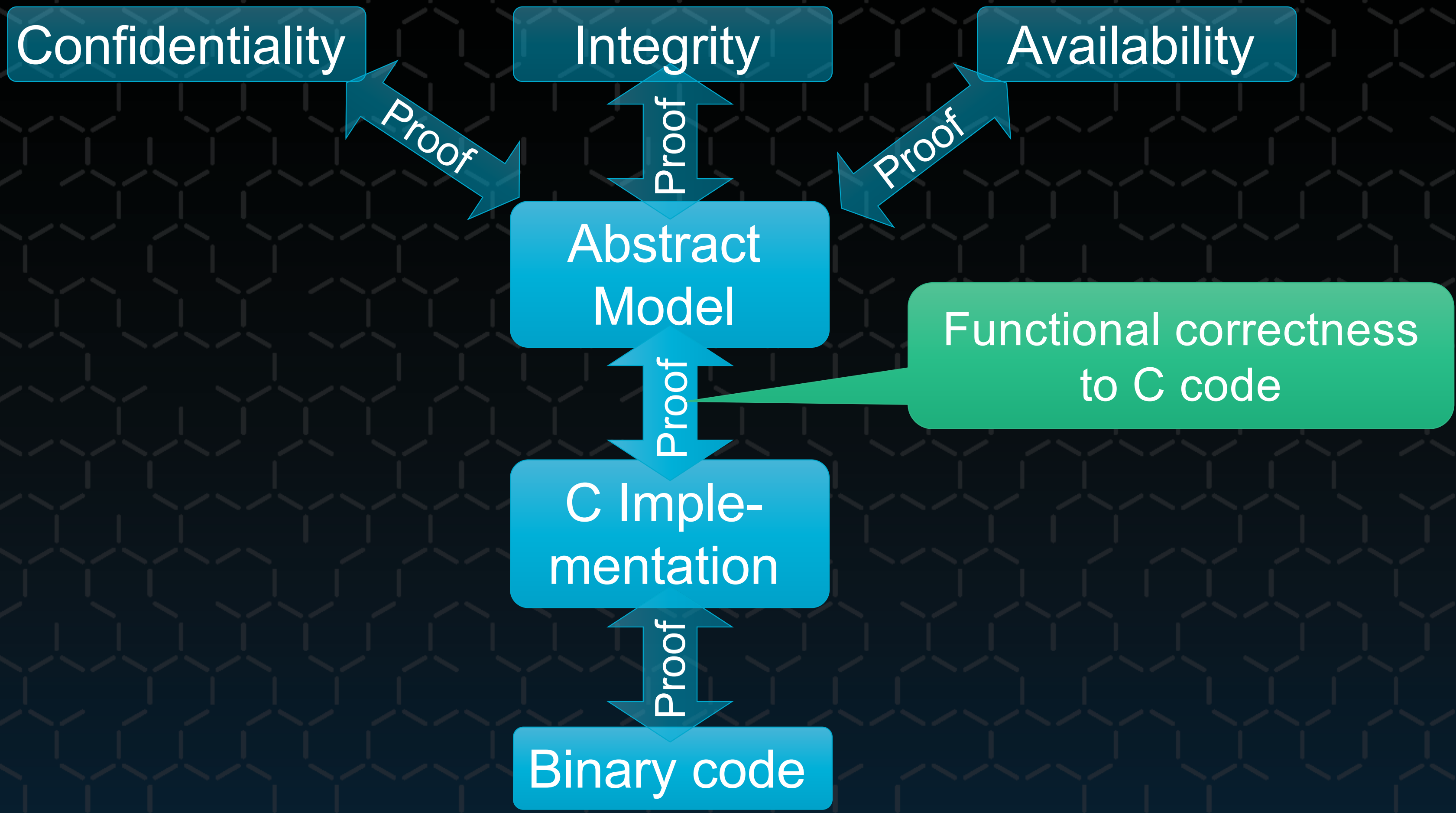
World's Most Secure OS: Arm v7



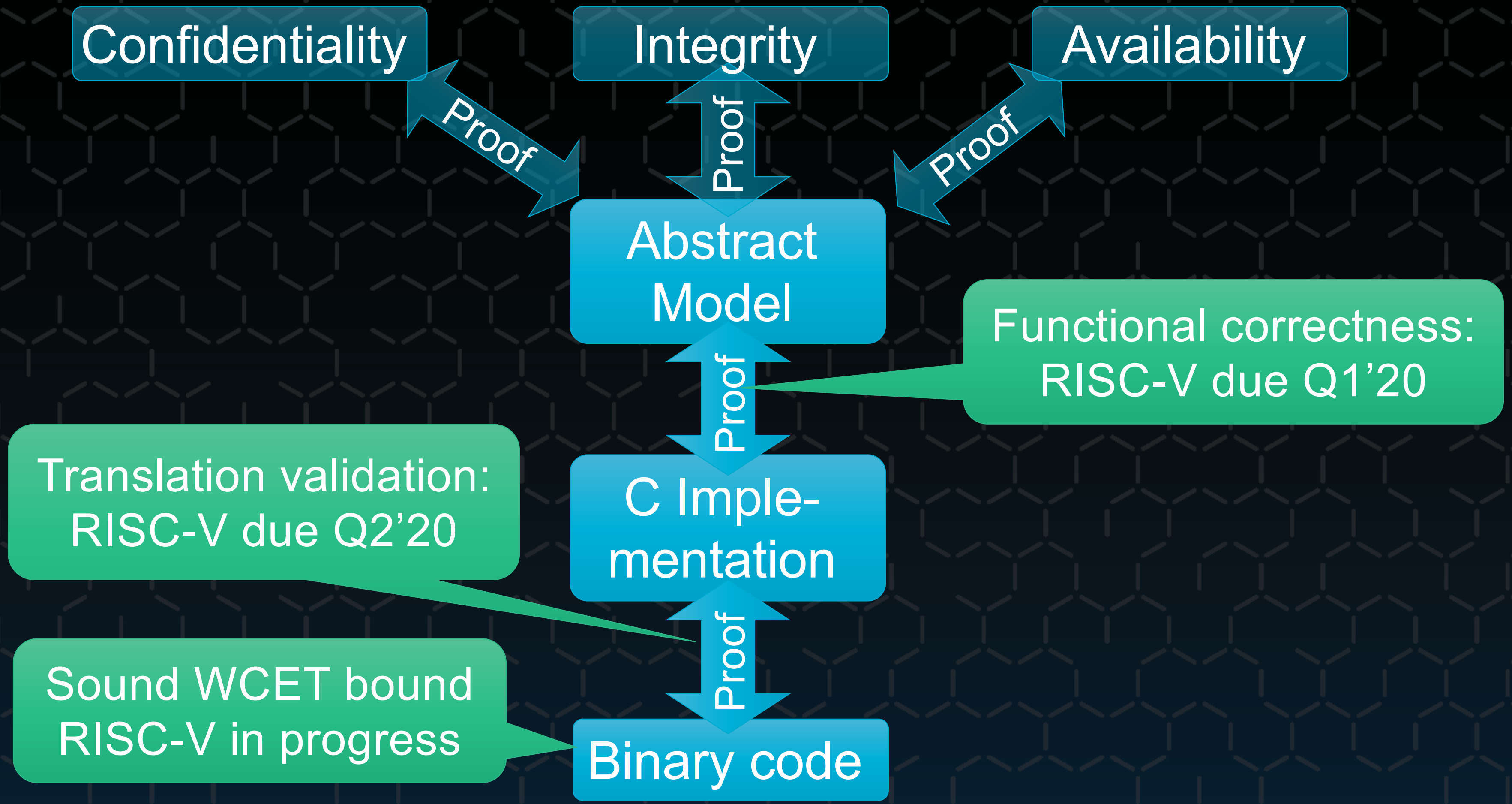
Limitations (work in progress):

- Kernel initialisation not yet verified
- MMU & caches modelled abstractly
- Multicore version not yet verified
- Timing channels not ruled out

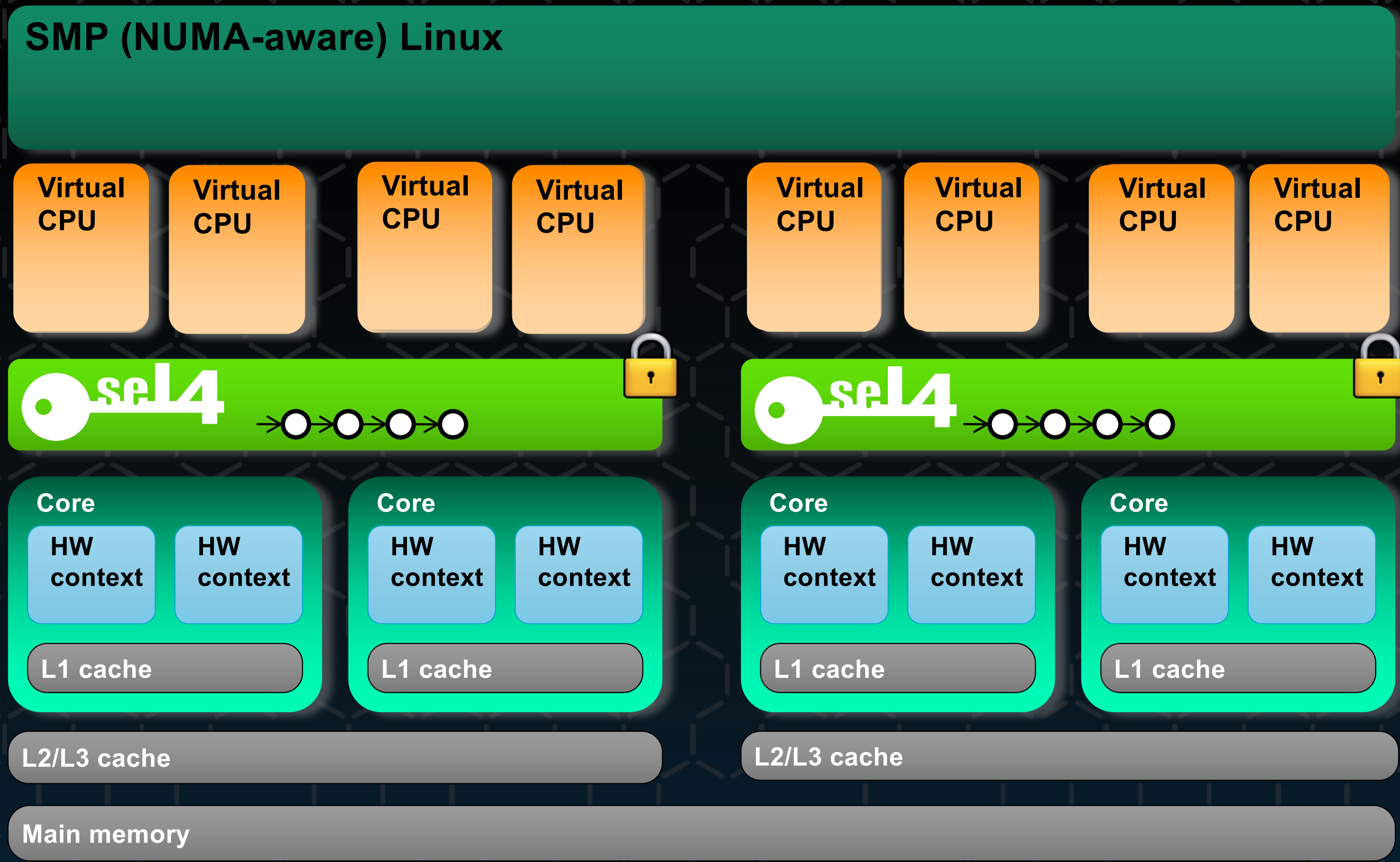
Verification: x86-64 Status



Verification: RISC-V (RV64) Status



Multicore: Clustered Multikernel



Military-Strength Security

Unmanned Little Bird (ULB)

DARPA HACMS: Retrofit existing system!

Autonomous trucks

Secure Comms Dongle

Cross-Domain Desktop Compositor

Take-Aways

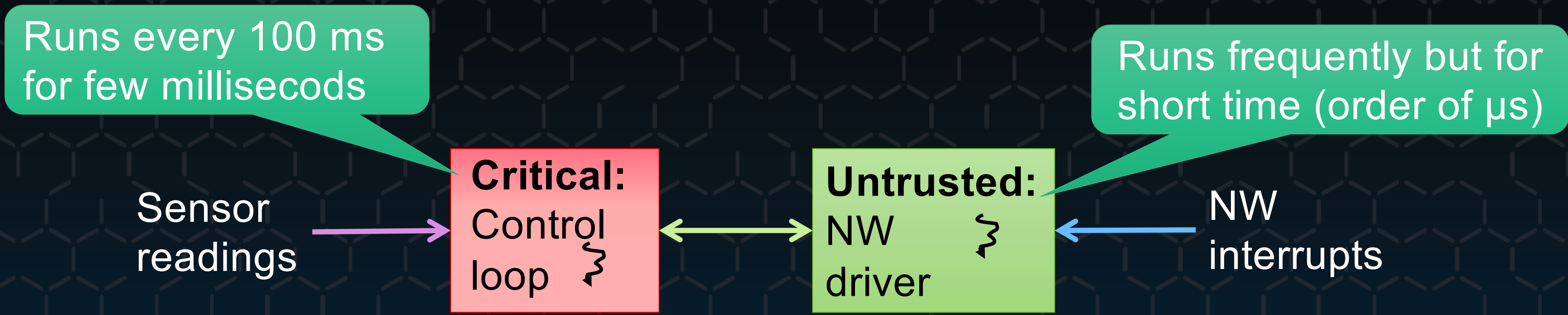
- seL4 provides strong isolation – *enabler* of system security
- seL4 is suitable for real-world use, even for retrofitting security
  - … but software and tool support is still quite limited
  - … and multikernel support is presently non-existent
- seL4 won't stop you designing a system with no security at all
  - Using (static) architecture for security enforcement is well understood
  - Achieving security in dynamic systems much less so

Mixed-Criticality Systems

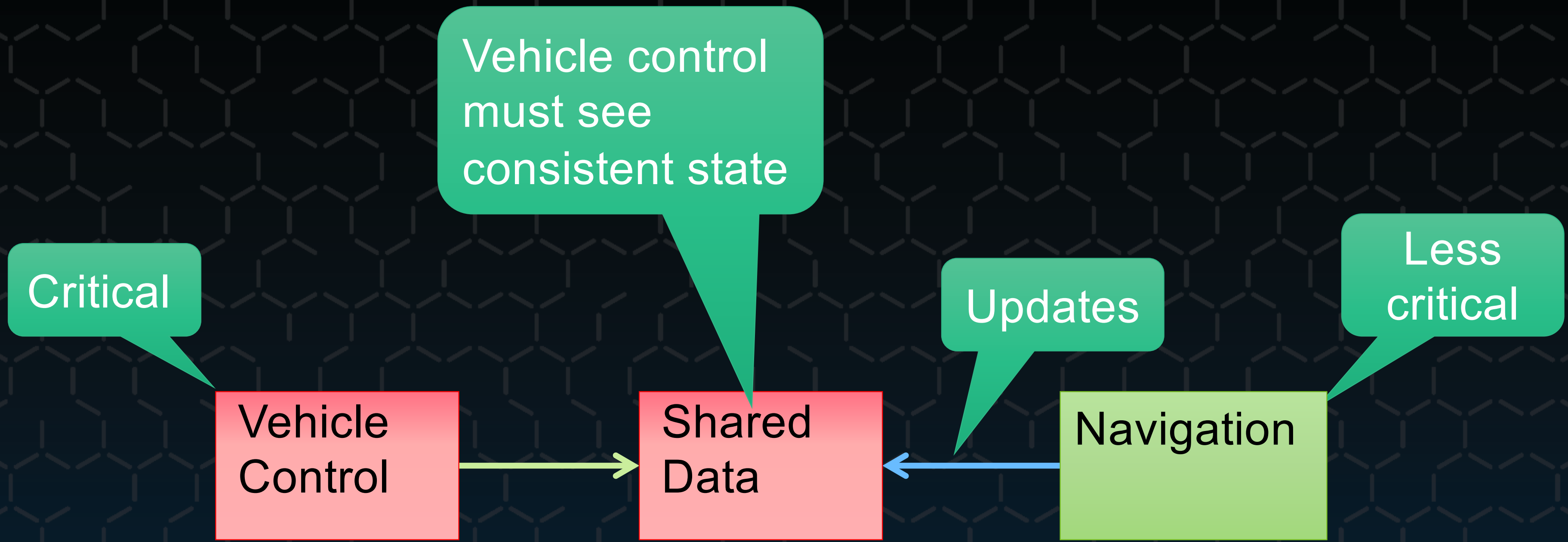
Mixed Criticality: Critical + Untrusted

NW driver must preempt control loop

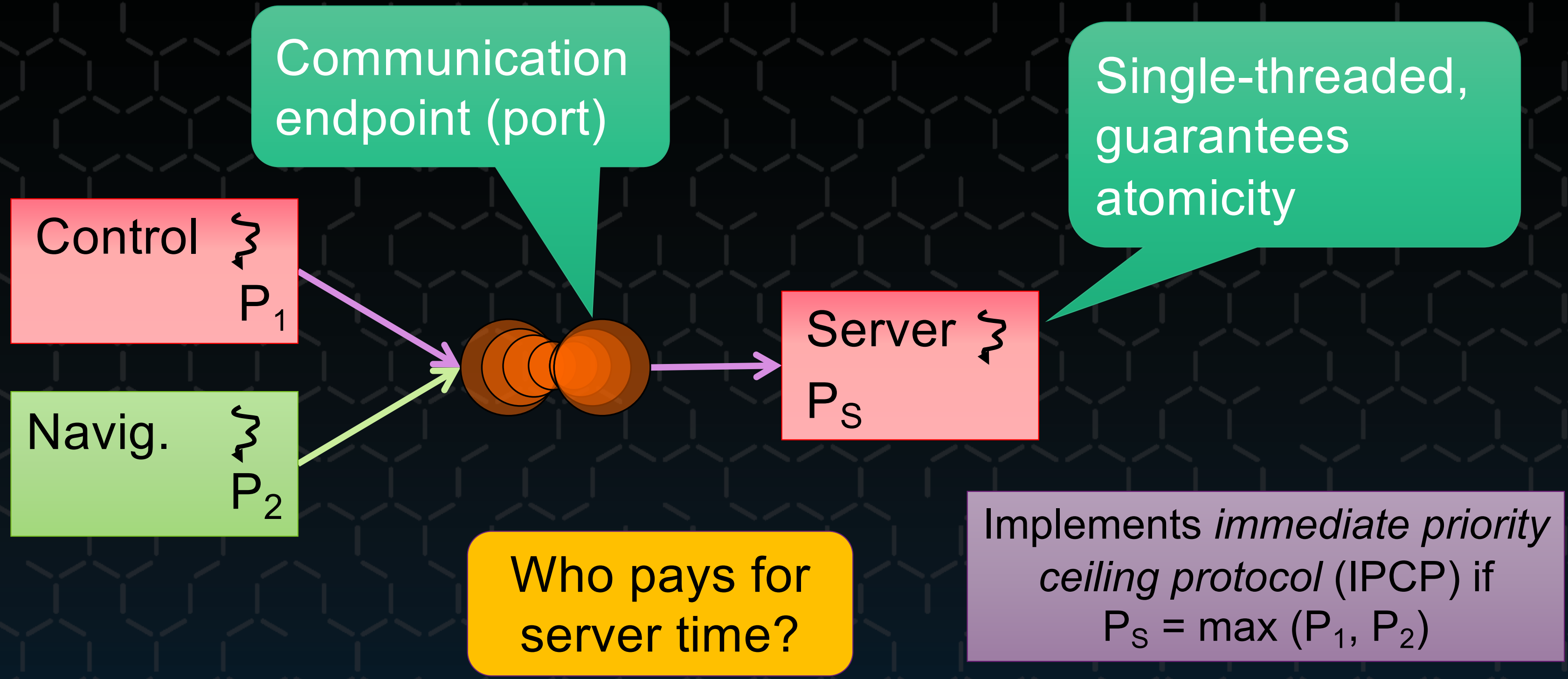
- … to avoid packet loss
- Driver must run at high prio
- Driver must be trusted not to monopolise CPU



MCS Challenge: Sharing



Sharing: Delegation to *Resource Server*



Solution: Time Capabilities

Classical thread attributes

- Priority
- Time slice

Not runnable if null

New thread attributes

- Priority
- Scheduling context capability

Capability for time

Limits CPU access!

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

C = 2
T = 3

Enables reasoning about time and temporal isolation for mixed-criticality systems

MCS with Scheduling Contexts

Runs every 100 ms for few millisecods

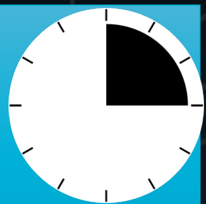
Runs frequently but for short time (order of µs)

Sensor readings → Control loop (P = low) → NW driver (P = high) ← NW interrupts

C = 25,000

T = 100,000

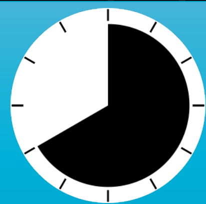
Utilisation = 25%



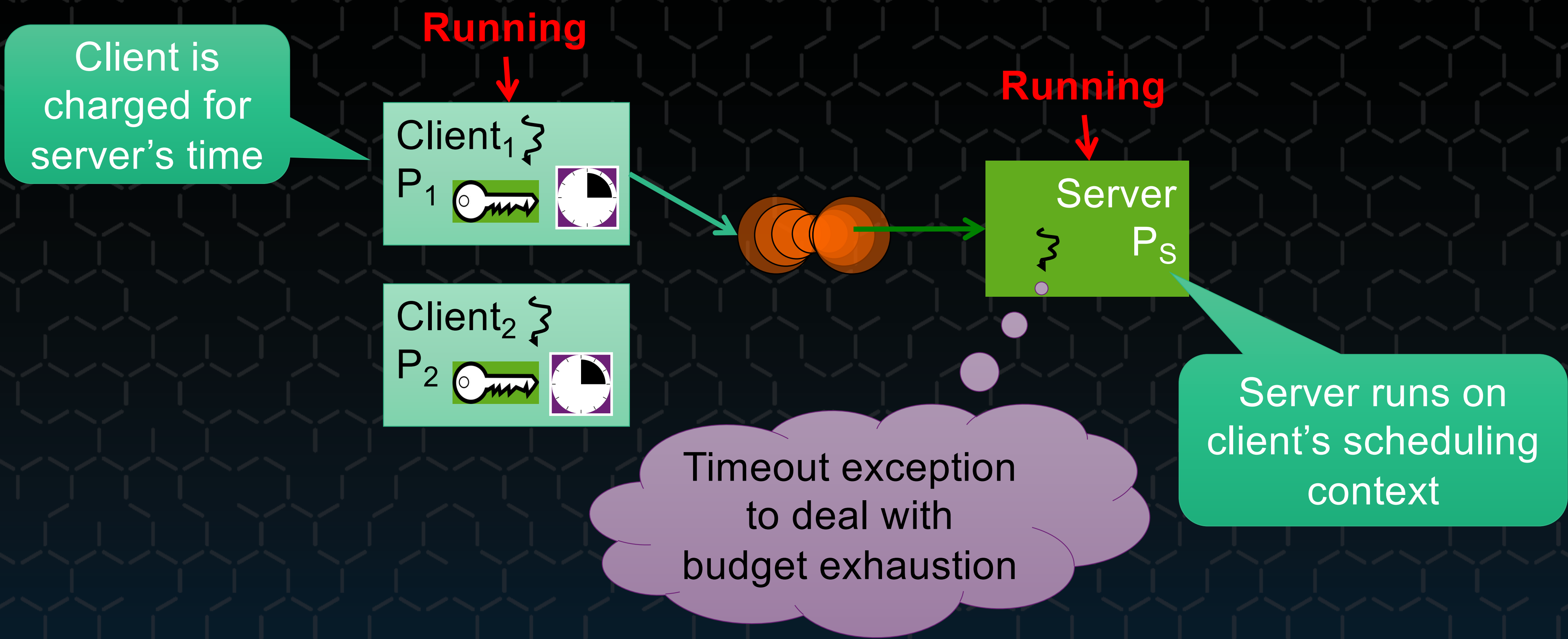
C = 2

T = 3

Utilisation = 67%



Shared Server Time Charged to Client



Take-Aways: MCS Support

- Time as a first-class resource
  - Enforcement of delegatable time budgets
  - Suitable for formal reasoning
  - Verification to be completed this year
- Supports mixed-criticality systems without strict time and space partitioning
  - avoids by-design low utilisation
  - avoids high interrupt latencies

Time Protection

Threats

=

Speculation

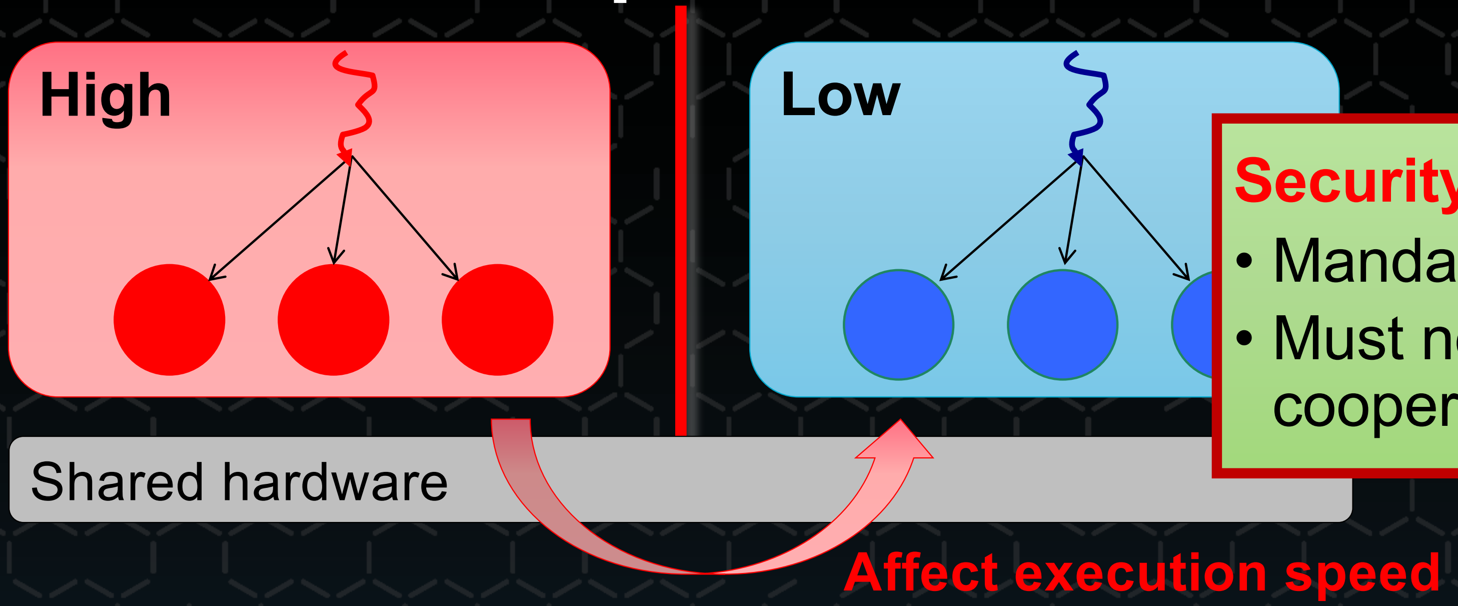
An “unknown unknown” until recently

+

A “known unknown” for decades

Microarchitectural Timing Channel

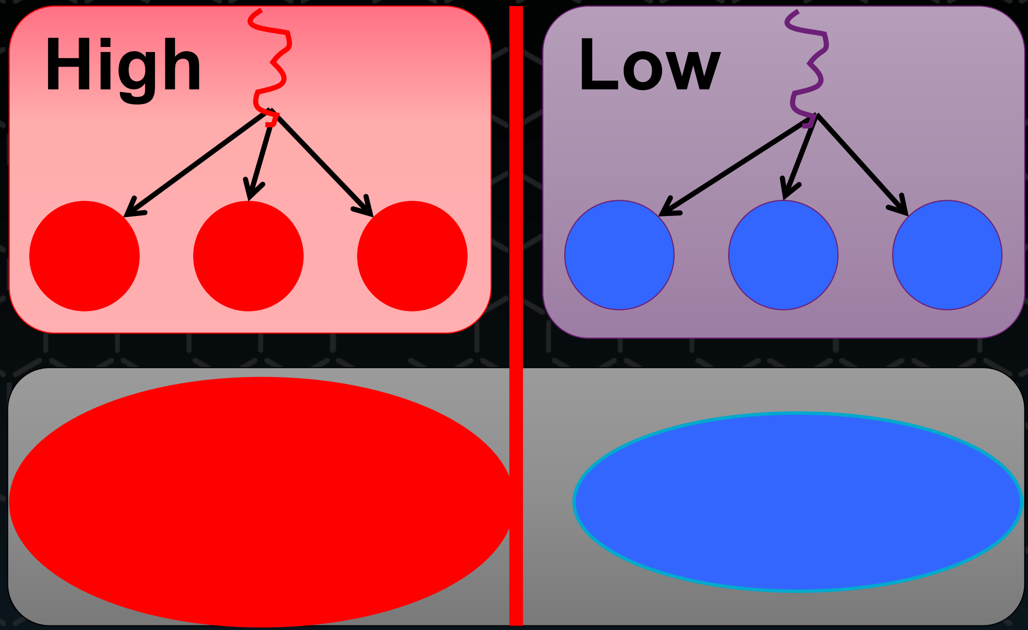
Cause: Competition for HW Resources



Security is a core OS job
- Mandatory enforcement
- Must not depend on app cooperation

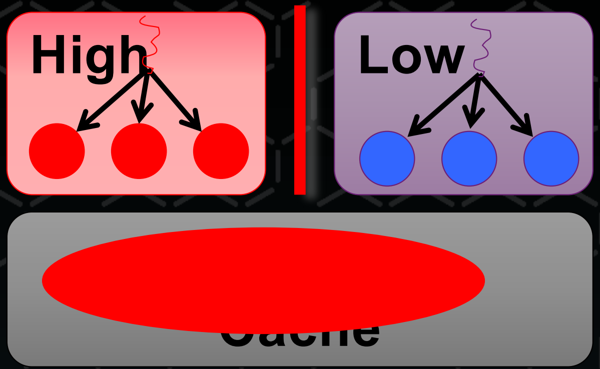
- Inter-process interference
- Competing access to micro-architectural features
- Hidden by the HW-SW contract!

Systematic Defence: Time Protection



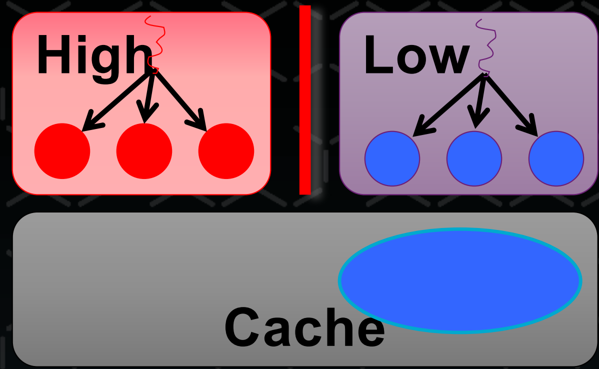
Time protection:
A collection of *OS mechanisms* which collectively *prevent interference* between security domains that make execution in one domain dependent on the activities of another.
[Ge et al. EuroSys'19]

Time Protection: Partition Hardware

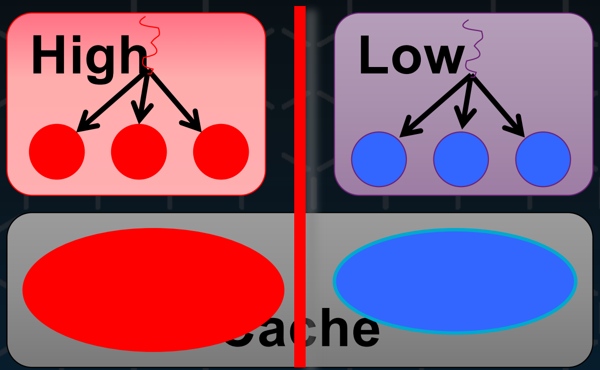


Temporally partition

Flush



Spatially partition



Need both!

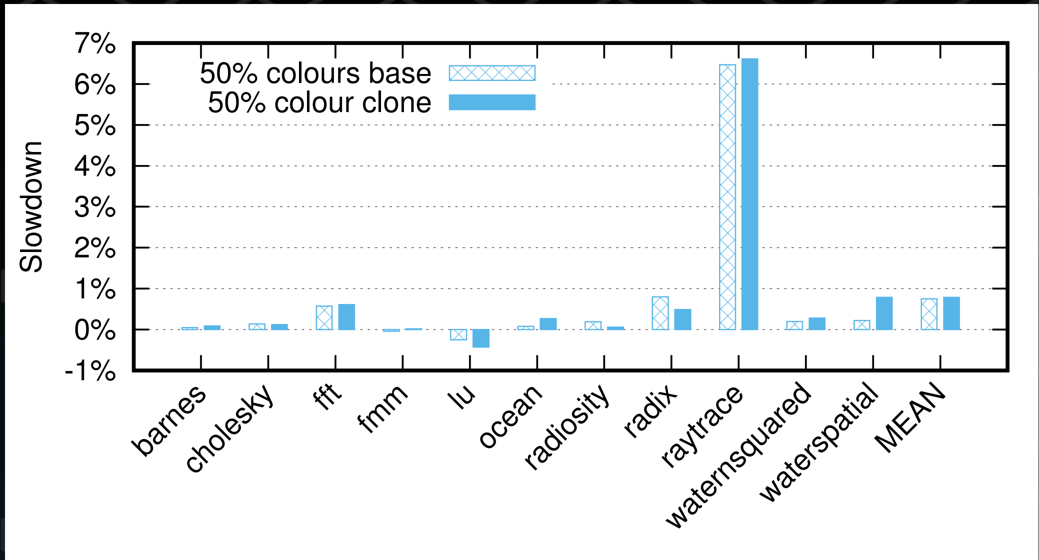
Cannot spatially partition on-core caches (L1, TLB, branch predictor, pre-fetchers)

- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

- HW threads
- cores

Performance Impact



- Overhead mostly low
- Not evaluated is cost of not using super pages

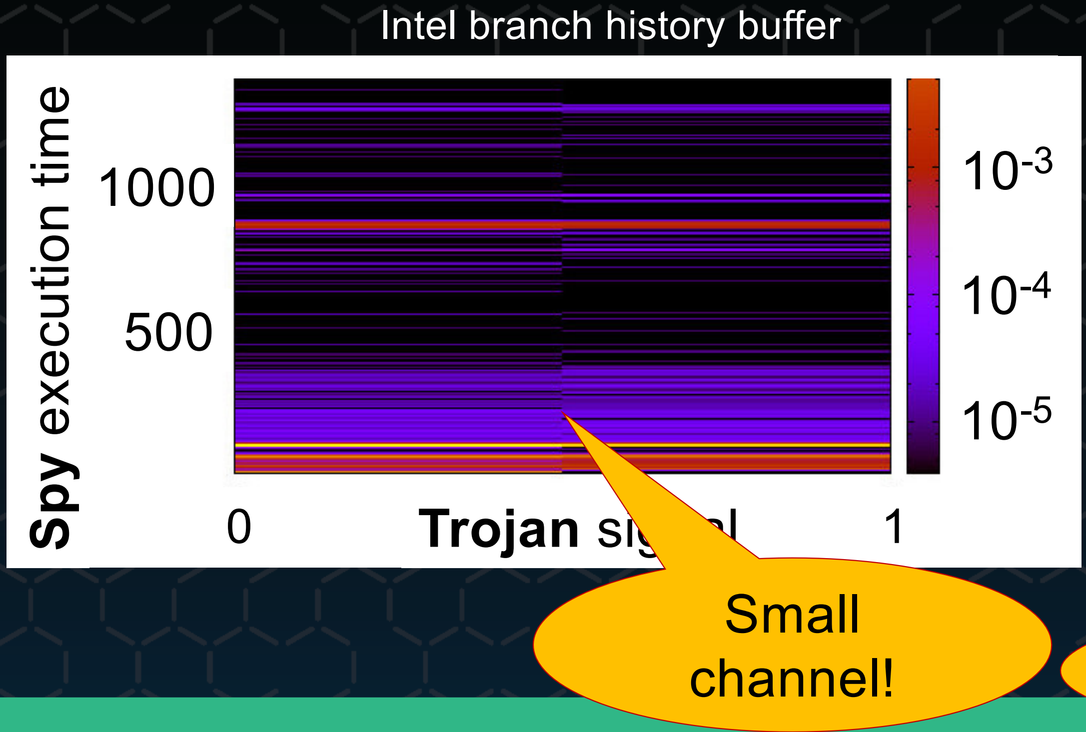
[Ge et al., EuroSys'19]

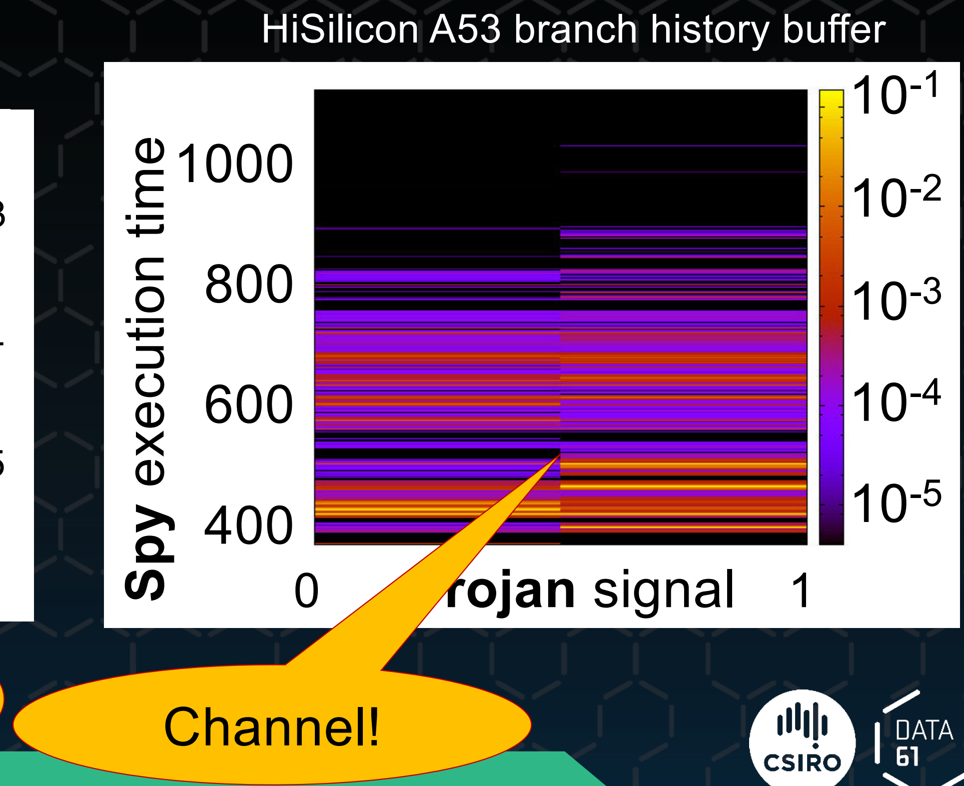
Arch	seL4 clone	Linux fork+exec
x86	79 µs	257 µs
Arm	608 µs	4,300 µs

Architecture	x86	Arm
Mean slowdown	3.4%	1.1%

Challenge: Broken Hardware

- Systematic study of COTS hardware (Intel and Arm) [Ge et al, APSys'18]:
  - contemporary processors hold state that cannot be reset

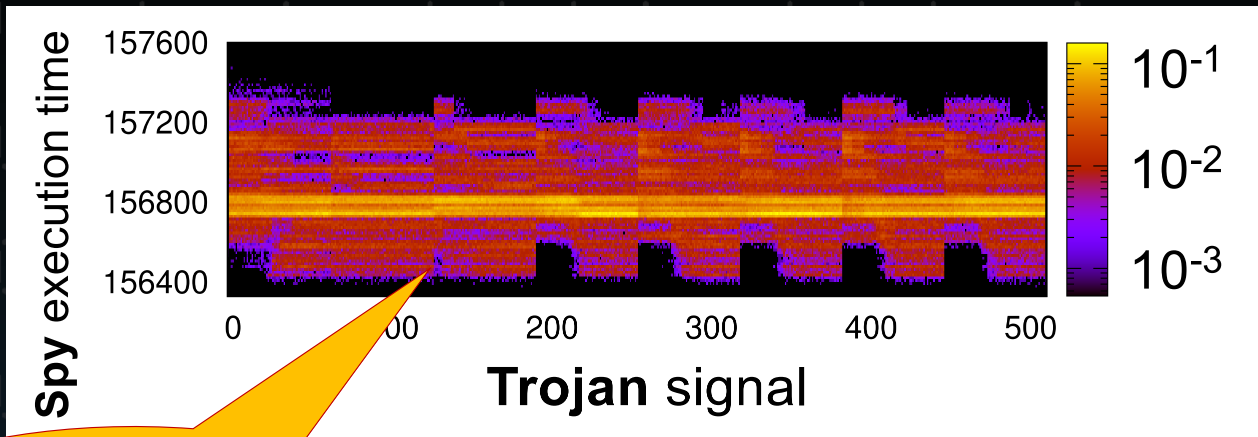




Challenge: Broken Hardware

- Systematic study of COTS hardware (Intel and Arm) [Ge et al, APSys'18]:
  - contemporary processors hold state that cannot be reset

Intel L2 cache



Channel resulting from data prefetcher

Solution: New HW-SW Contract!

ISA is purely functional contract, abstracts too much away

New contract (augmented ISA):

All shared HW resources must be spatially or temporally partitionable by OS

[Ge et al, APSys'18]

RISC-V to the rescue:
Strong commitment to making it happen!

Take-Aways: Time Protection

- Security is a core operating-system job
  - Enforcement must be mandatory
  - OS-provided isolation must be extended from space to time domain
- We understand the mechanisms required to do this
  - implementation shows low overhead
  - can verify against suitable hardware model
- Outstanding problems:
  - turn models into an actual operating-system security model
  - Fix the hardware!

THANK YOU

Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser

https://trustworthy.systems