The Quest for the Perfect API

Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser
Trustworthy Systems | Data61

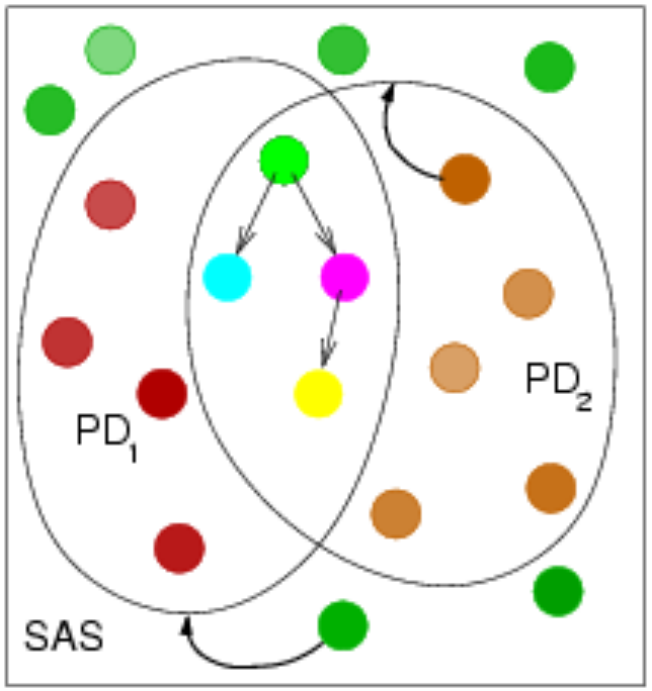
https://trustworthy.systems

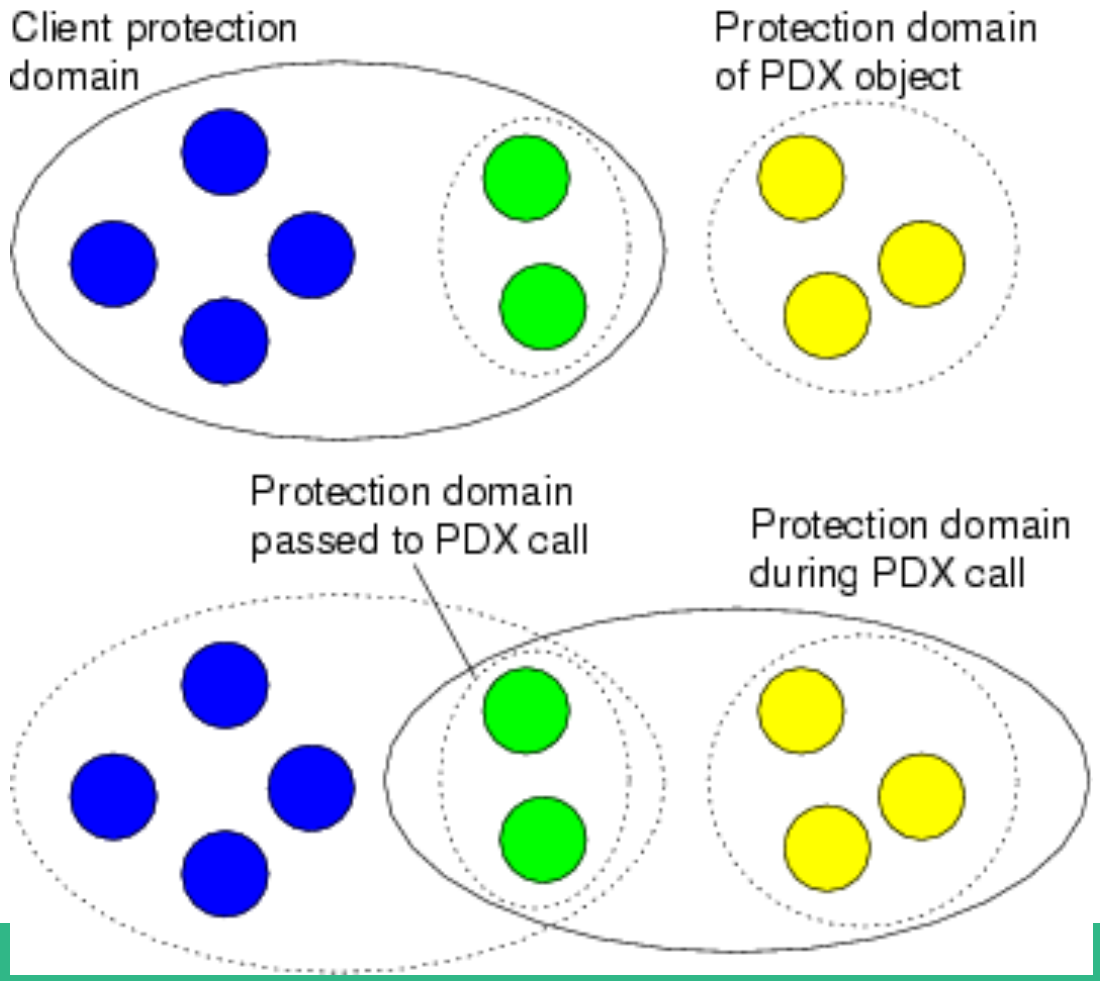
Observation: Operating Systems Suck

Mungi Single-Address-Space OS

Mungi: Single-Address-Space OS

De-couple translation
from protection





Mungi Retrospective

Mungi was a roaring success – not

- never found a convincing killer app
- workarounds for 32-bit limitations (Unix model) too accepted
- it was naïve to think we could change mainstream computing
- 64-bit address space already too small for all data

Lesson: If you want to change the world, pick the right world to change

Long-term benefits of Mungi

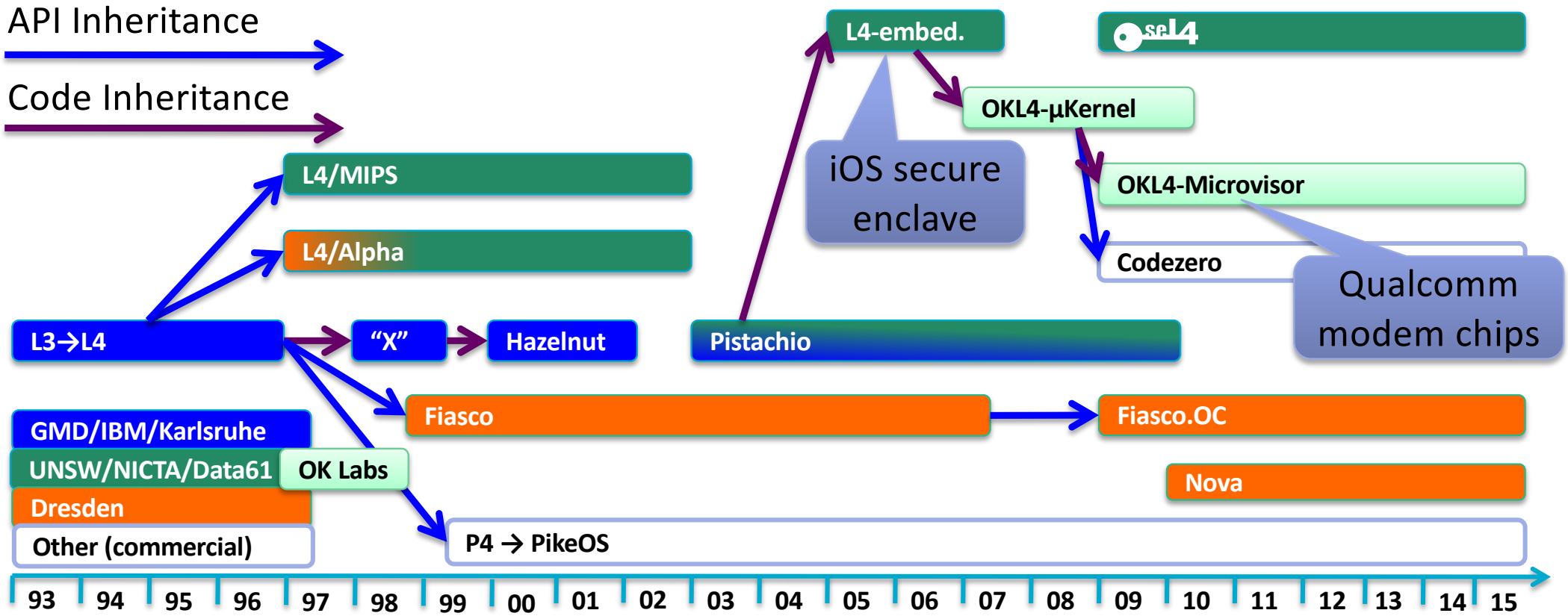
- built a systems group
- got us into L4

L4 Microkernel

L4 Microkernel

A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system's required functionality. [Liedtke, SOSP'95]

25 Years of L4 Microkernel R&D



L4 IPC Performance Over the Years

Name	Year	Processor	MHz	Cycles	µs
Original	1993	i486	50	250	5.00
Original	1997	Pentium	160	121	0.75
L4/MIPS	**1997**	**R4700**	**100**	**86**	**0.86**
L4/Alpha	**1997**	**21064**	**433**	**45**	**0.10**
Hazelnut	2002	Pentium 4	1,400	2,000	1.38
Pistachio	**2005**	**Itanium**	**1,500**	**36**	**0.02**
OKL4	**2007**	**XScale 255**	**400**	**151**	**0.64**
NOVA	2010	i7 Bloomfield (32-bit)	2,660	288	0.11
seL4	**2013**	**ARM11**	**532**	**188**	**0.35**
seL4	**2018**	**i7 Haswell (64-bit)**	**3,400**	**442**	**0.13**
seL4	**2018**	**Cortex A9**	**1,000**	**303**	**0.30**

Minimality: Source-Code Size

Name	Architecture	C/C++	asm	total kSLOC
Original	i486	0	6.4	6.4
L4/Alpha	**Alpha**	**0**	**14.2**	**14.2**
L4/MIPS	**MIPS64**	**6.0**	**4.5**	**10.5**
Hazelnut	x86	10.0	0.8	10.8
Pistachio	x86	22.4	1.4	23.0
L4-embedded	**ARMv5**	**7.6**	**1.4**	**9.0**
OKL4 3.0	**ARMv6**	**15.0**	**0.0**	**15.0**
Fiasco.OC	x86	36.2	1.1	37.6
seL4	**ARMv6**	**9.7**	**0.5**	**10.2**

Original L4: Design & Implementation

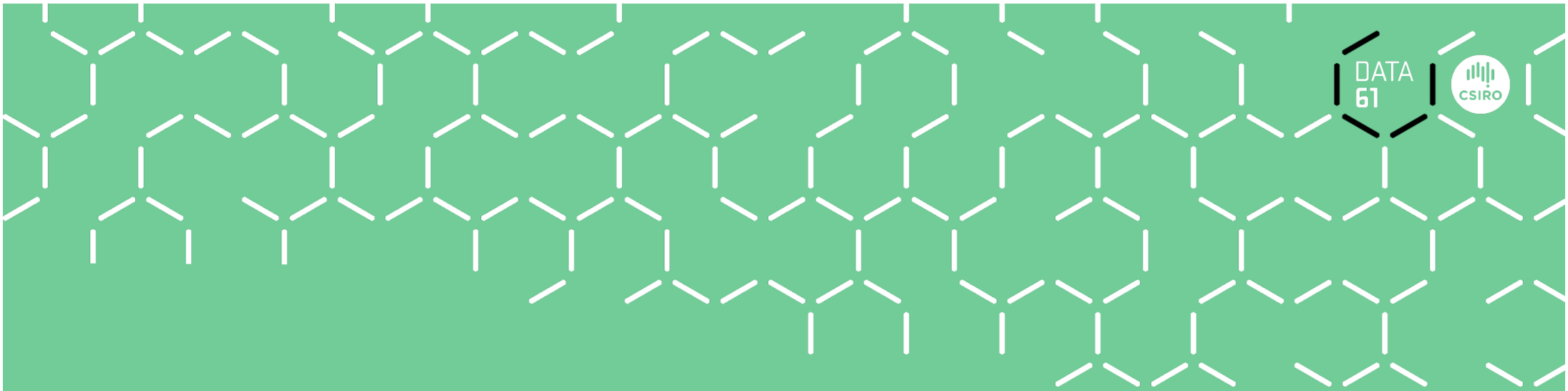
Implementation Tricks [SOSP'93]

- ~~Process kernel~~
- ~~Virtual TCB array~~
- Lazy scheduling
- Direct process switch
- Non-preemptible
- ~~Non-portable~~
- ~~Non-standard~~ calling convention
- ~~Assembler~~

Design Decisions [SOSP'95]

- Synchronous IPC
- ~~Rich message structure~~, arbitrary out-~~of-line messages~~
- Zero-copy register messages
- User-mode page-fault handlers
- ~~Threads as IPC~~ destinations
- ~~IPC timeouts~~
- ~~Hierarchical IPC~~ control
- User-mode device drivers
- ~~Process hierarchy~~
- Recursive address-space construction

Objective: Minimise cache footprint and TLB misses



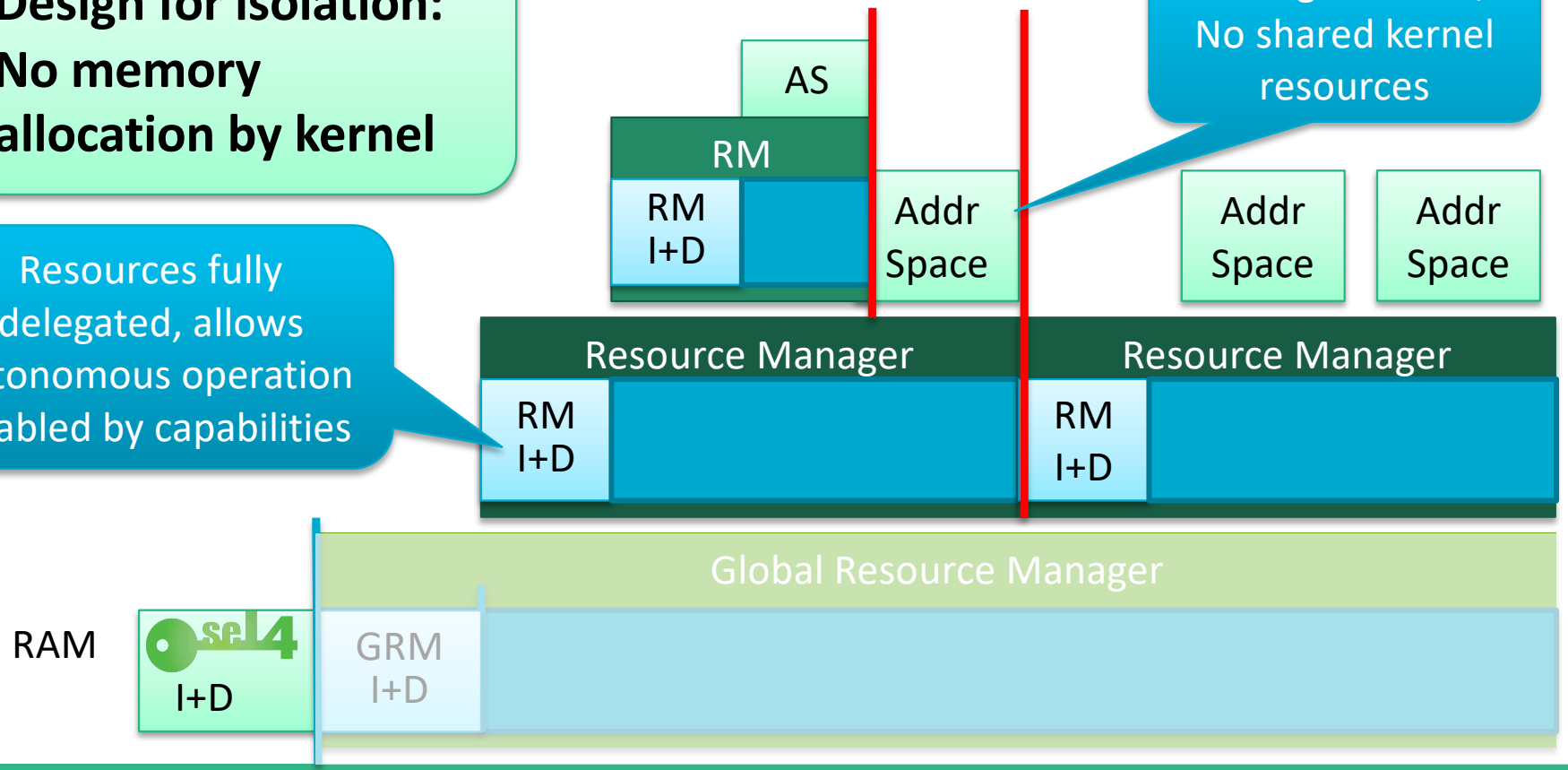
seL4:
Rethinking Resource Management

Memory Management

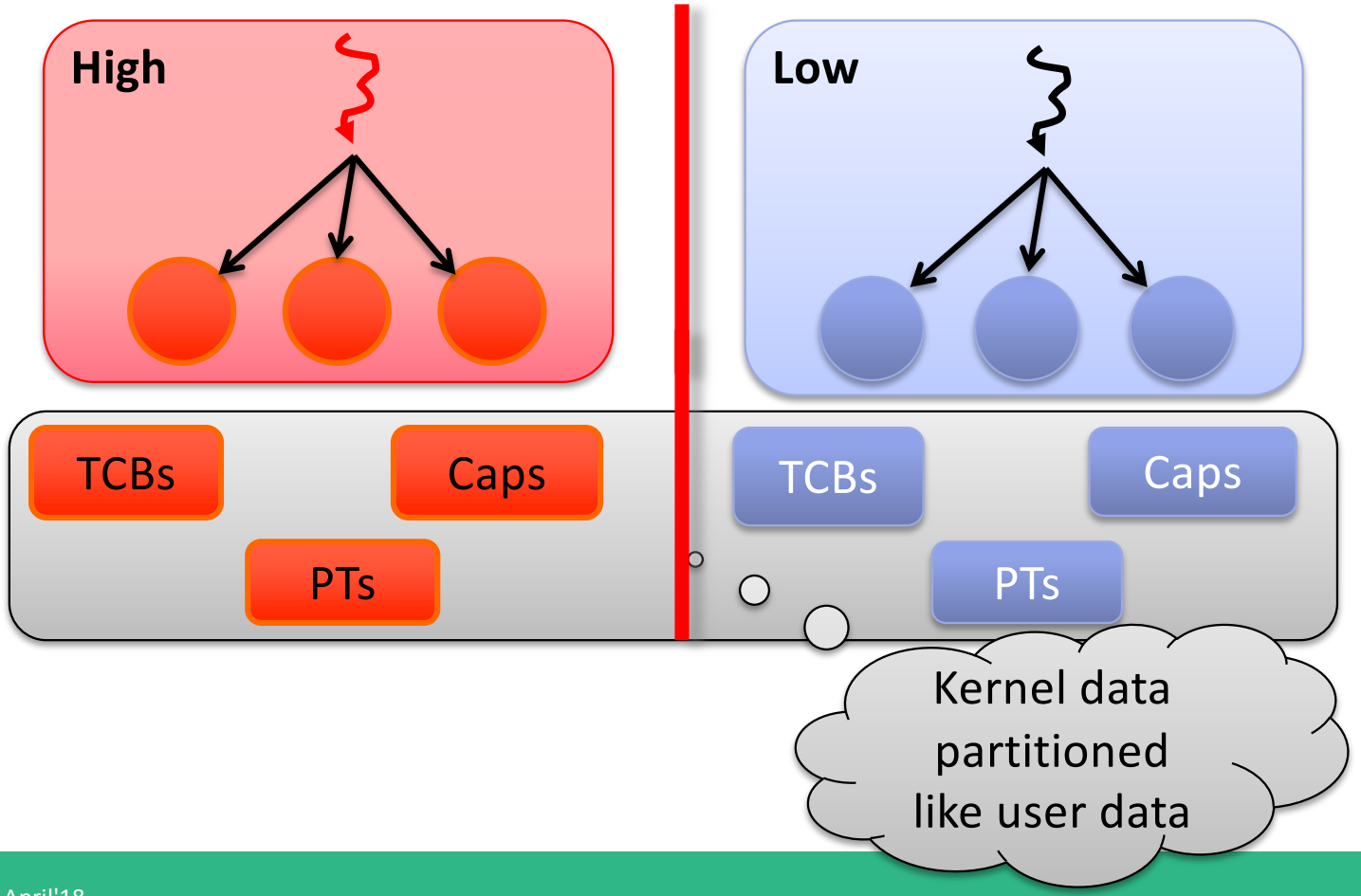
Design for isolation:
No memory allocation by kernel

Strong isolation, No shared kernel resources

Resources fully delegated, allows autonomous operation enabled by capabilities



Isolation Goes Deep



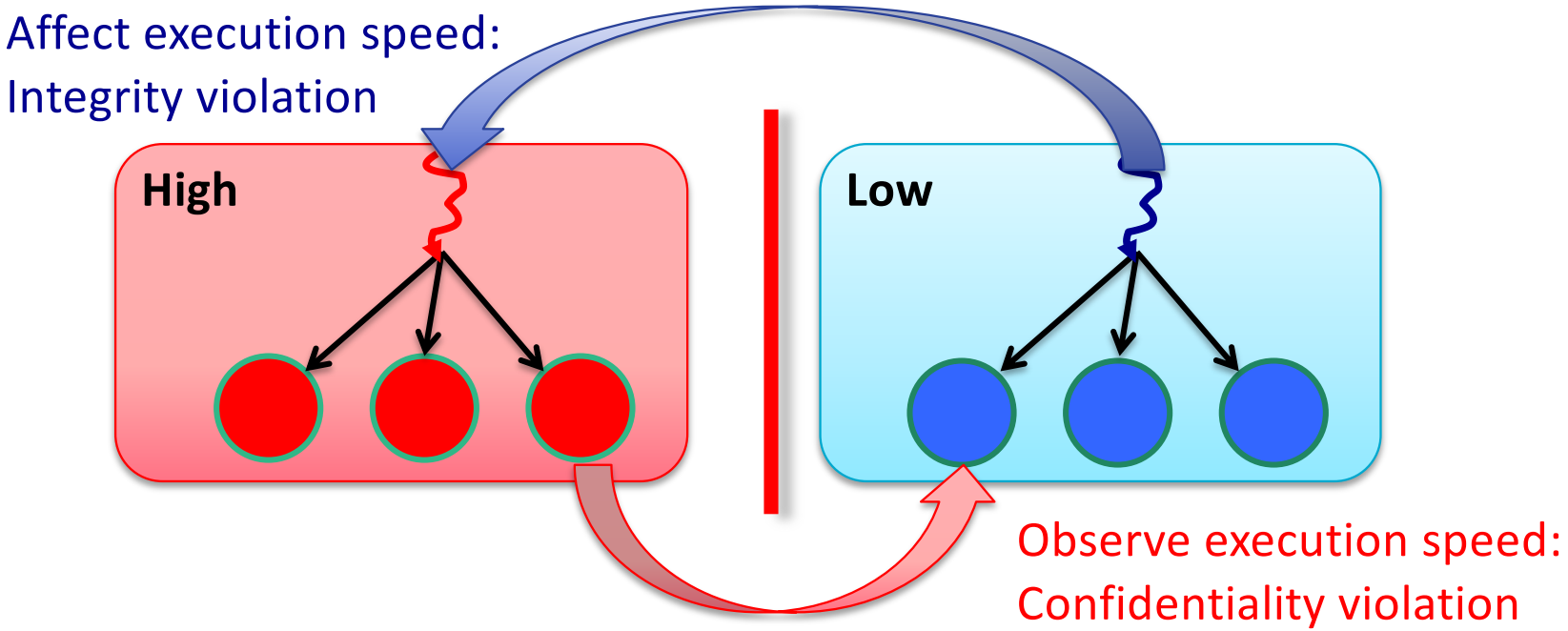
How About Temporal Isolation?

Safety: Timeliness

- Execution interference

Security: Confidentiality

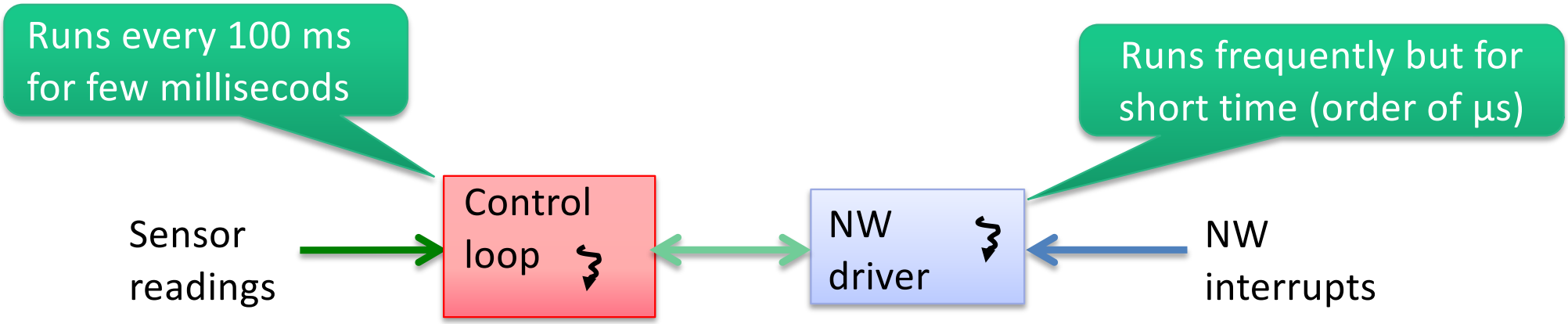
- Leakage via timing channels



Integrity Challenge: Mixed Criticality

NW driver must preempt control loop

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



Scheduling Contexts: Caps for Time

Classical thread attributes

- Priority
- Time slice

Not runnable if null

Limits CPU access!

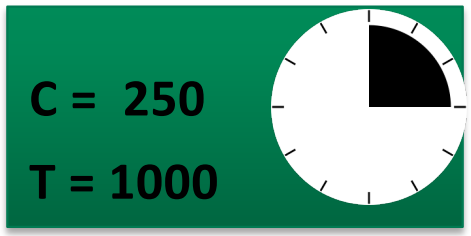
New thread attributes

- Priority
- Scheduling context capability

Scheduling context object

- T: period
- C: budget (≤ T)

Capability for time

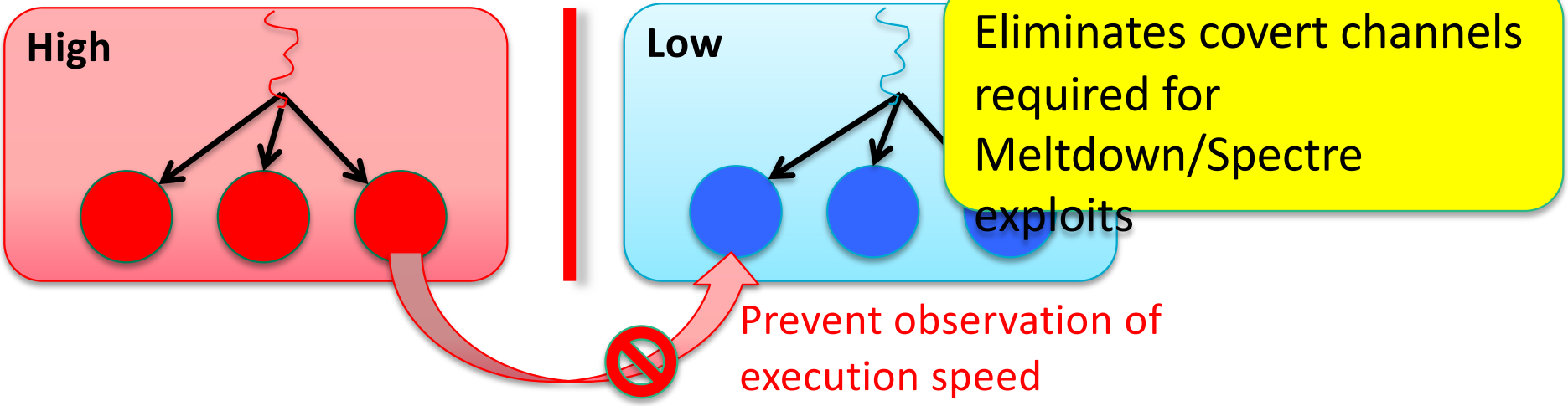


`SchedControl` capability conveys right to assign budgets (i.e. perform admission control)

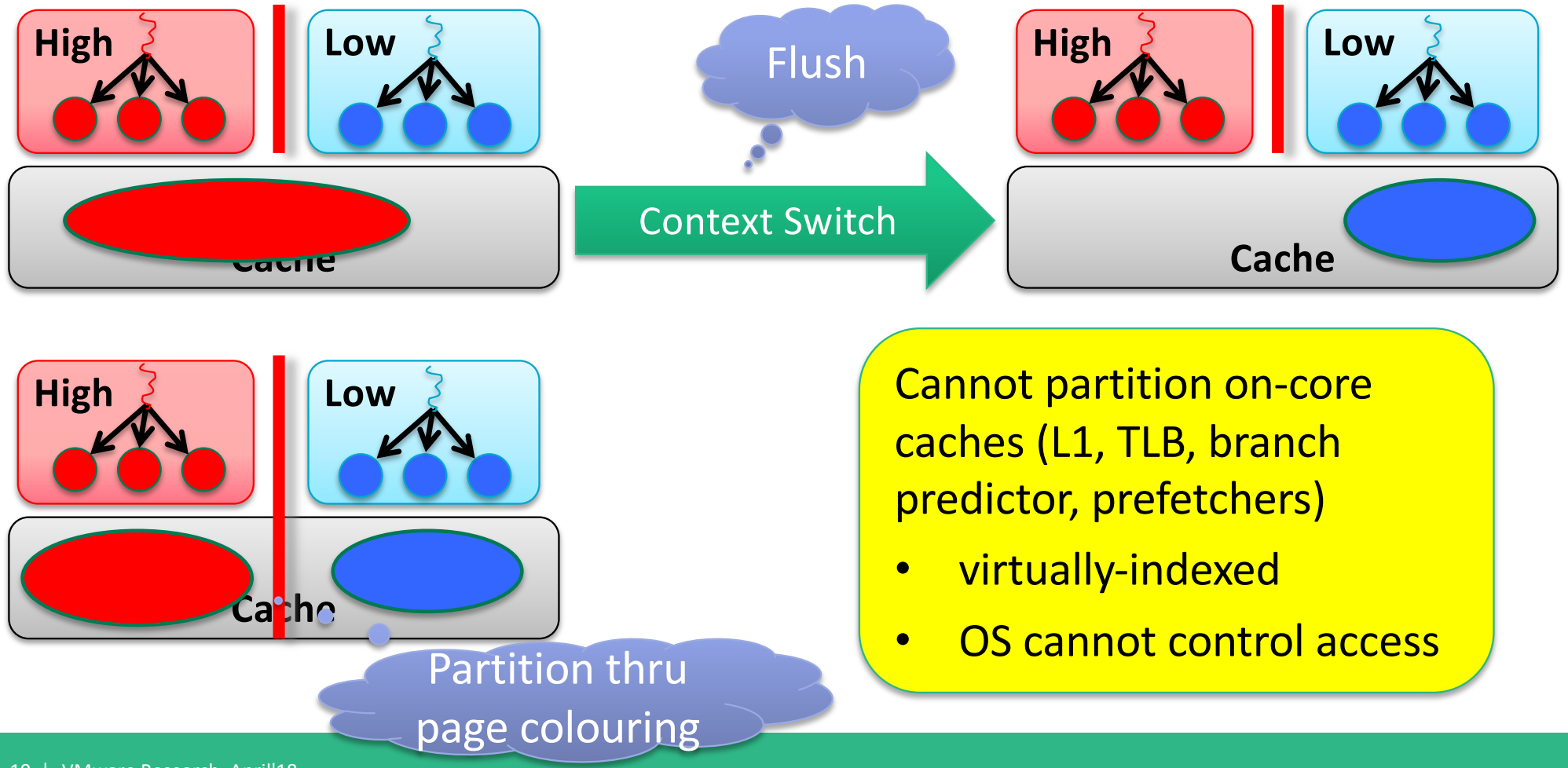
Confidentiality: Closing Timing-Channels

- Black-box, OS-enforced isolation
- No requirement to trust High code not to leak
- No requirement for modifying High code
- High and Low code untrusted – mandatory confinement
- Should also protect against data-dependent execution time

Time protection, just like standard memory protection



Mitigation: Prevent Sharing of State

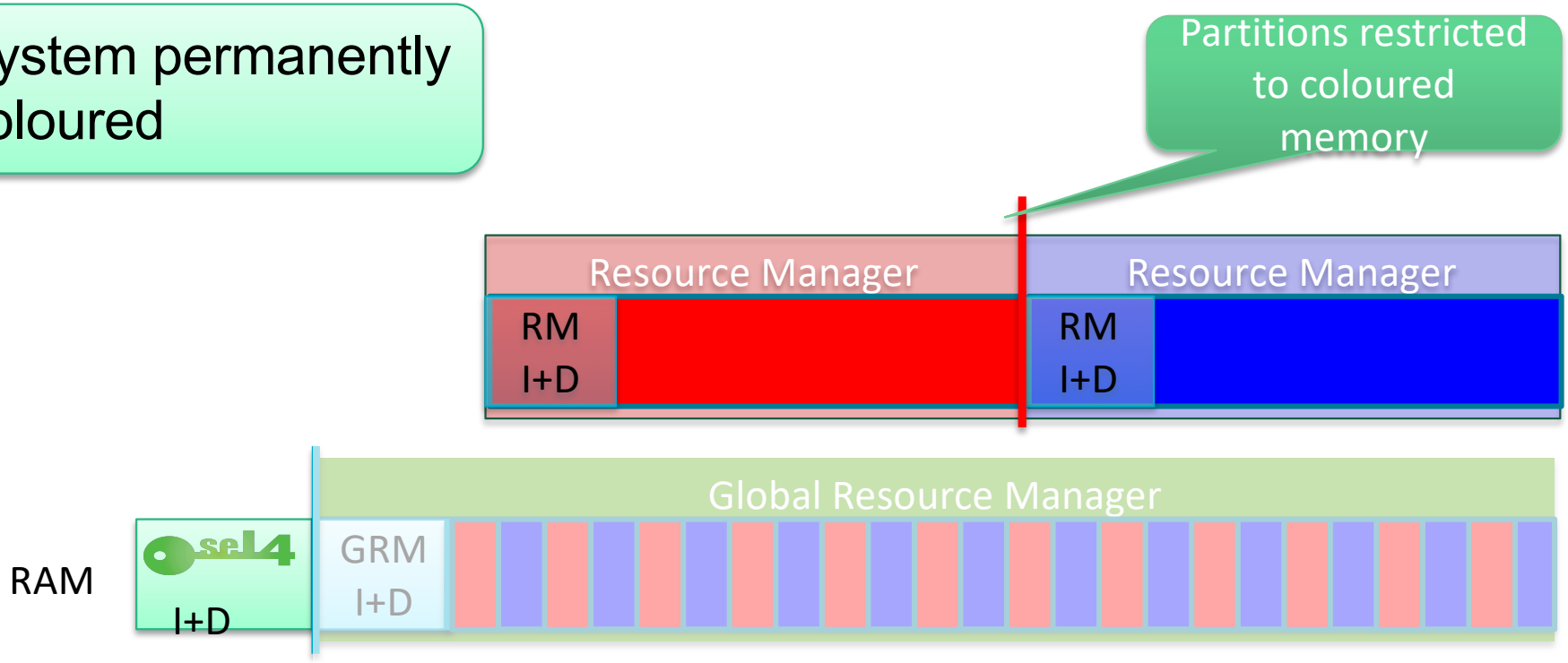


Cannot partition on-core caches (L1, TLB, branch predictor, prefetchers)

- virtually-indexed
- OS cannot control access

Colouring User Memory is Easy

System permanently coloured

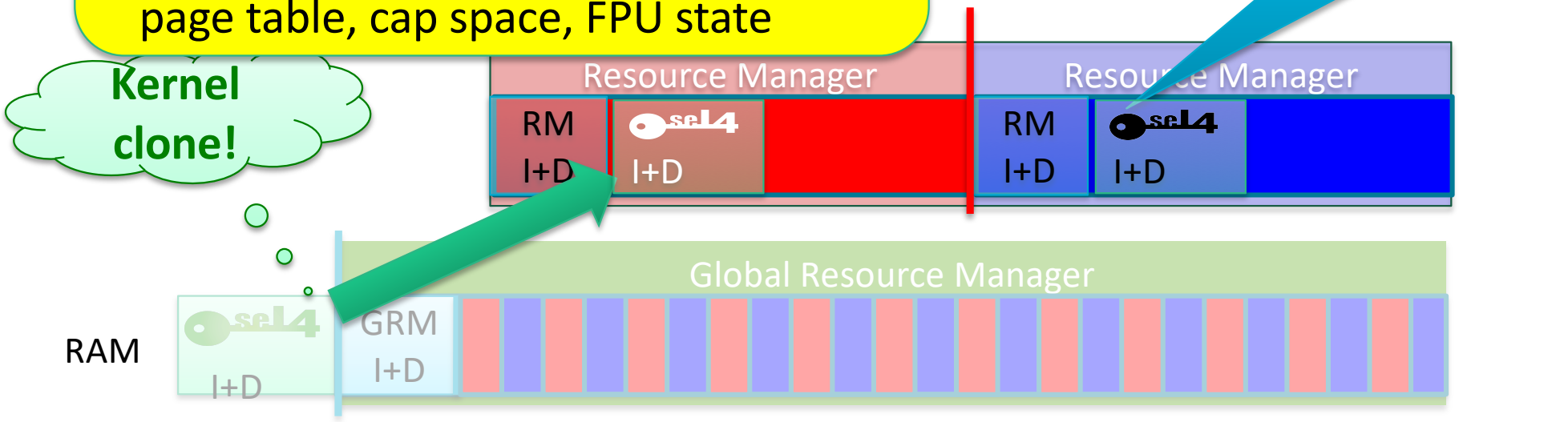


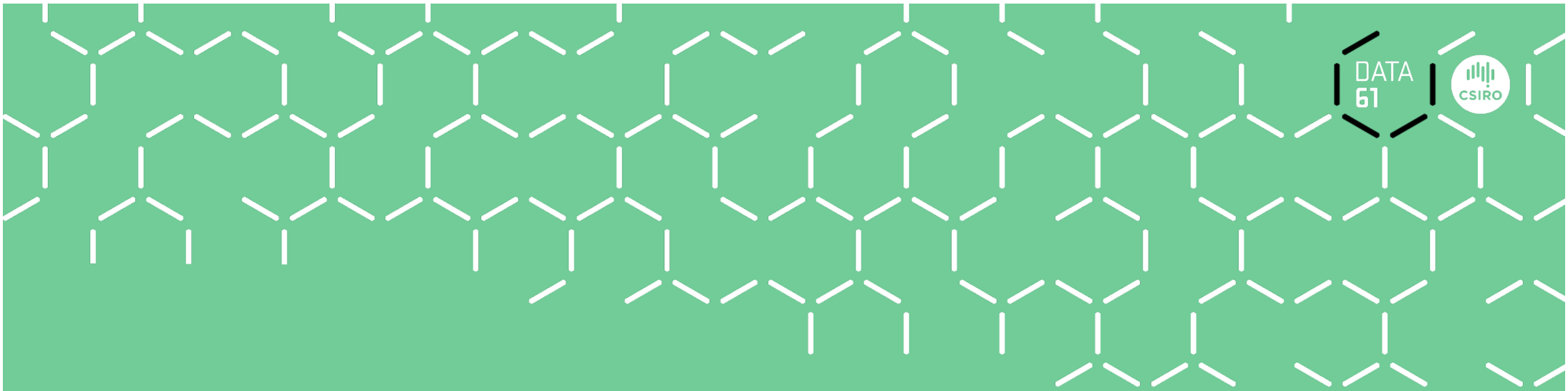
Colouring the Kernel

Only shared kernel data:

- Scheduler queue array & bitmap
- Pointers to current: thread, kernel, page table, cap space, FPU state

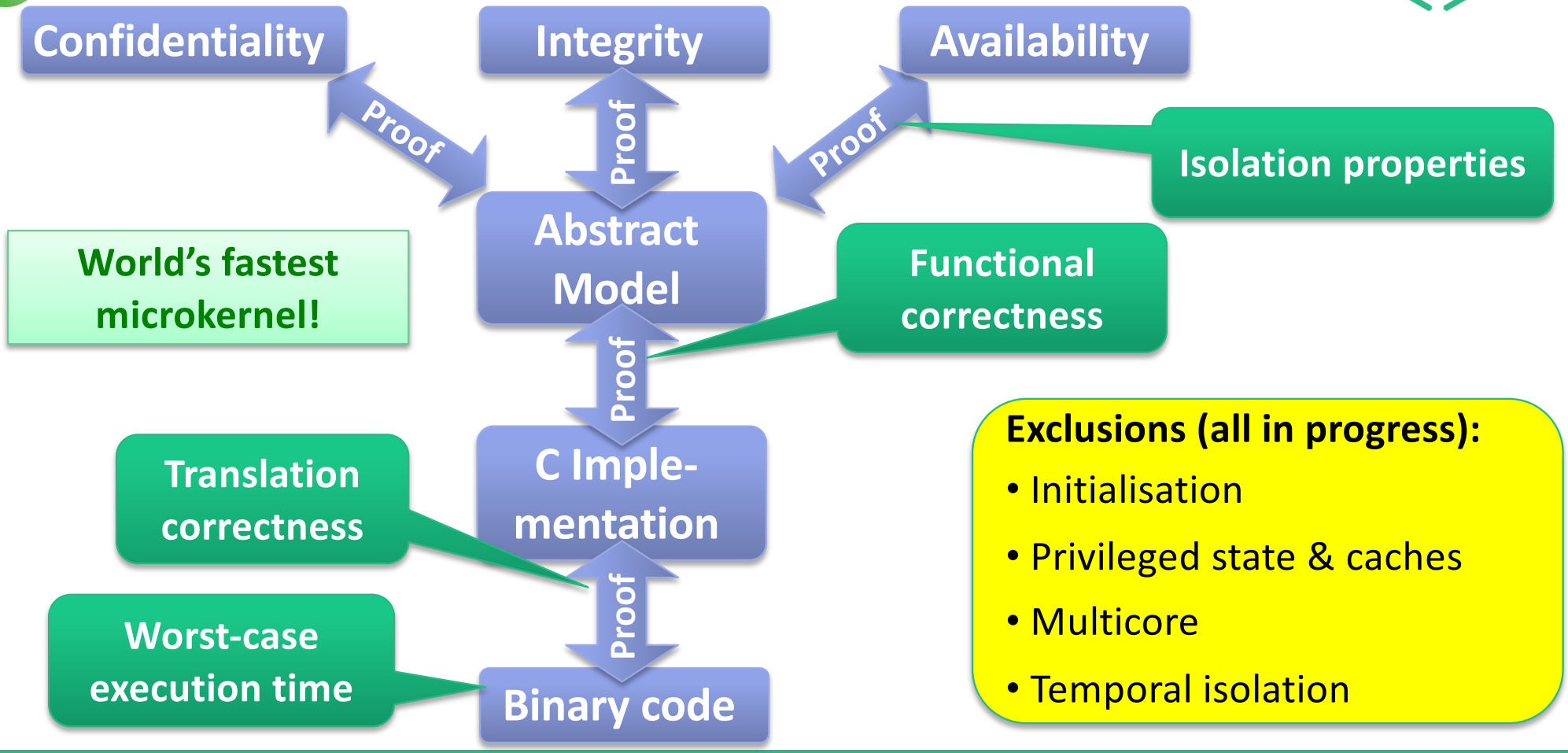
Each partition has own kernel image





Formal Verification – The Killer “App”

Provable Security Enforcement



Exclusions (all in progress):

- Initialisation
- Privileged state & caches
- Multicore
- Temporal isolation

What Made Verification Possible?

- Suitable design:
  - Microkernel, of course!
  - Isolation-oriented resource management helped proving global invariants
  - Resource-management model crucial for proving isolation properties
- From-scratch implementation:
  - Verifying code not written for verification is infeasible
  - Feedback loop between implementers and verifiers is essential

Why Build Your Own OS?

- Cannot really rethink abstractions on an legacy OS
- From-scratch implementation feasible and necessary for high-performance microkernels
- Verification of existing code bases infeasible

DATA
61
Thank you!
https://trustworthy.systems
CSIRO