Building Secure Systems With seL4

Gernot Heiser | Data61 and UNSW Sydney
Gernot.Heiser@data61.csiro.au | @GernotHeiser

FMATS, Cambridge, Sep'18

https://seL4.systems

What is seL4?

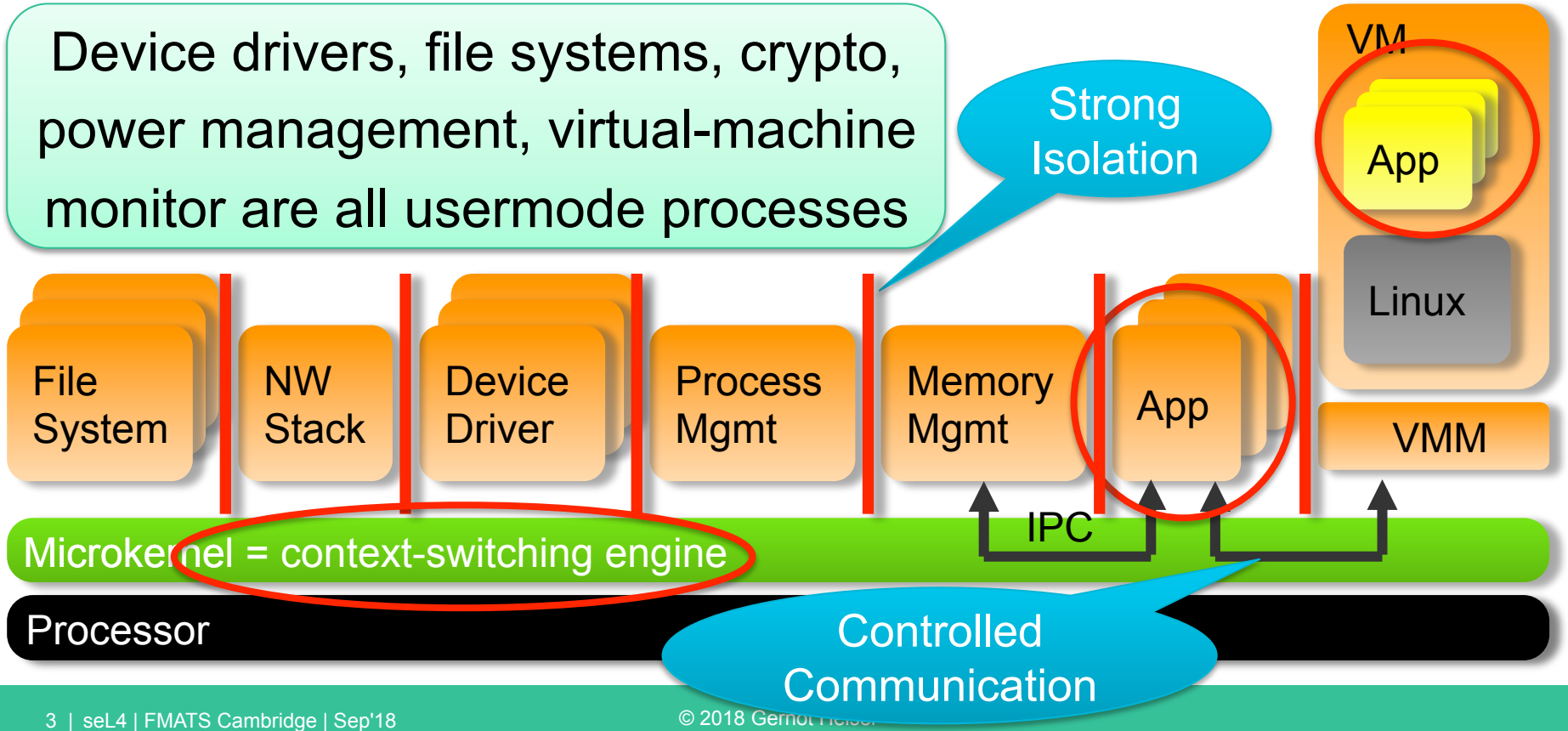
seL4: The world's only operating-system kernel with provable security enforcement

Open Source

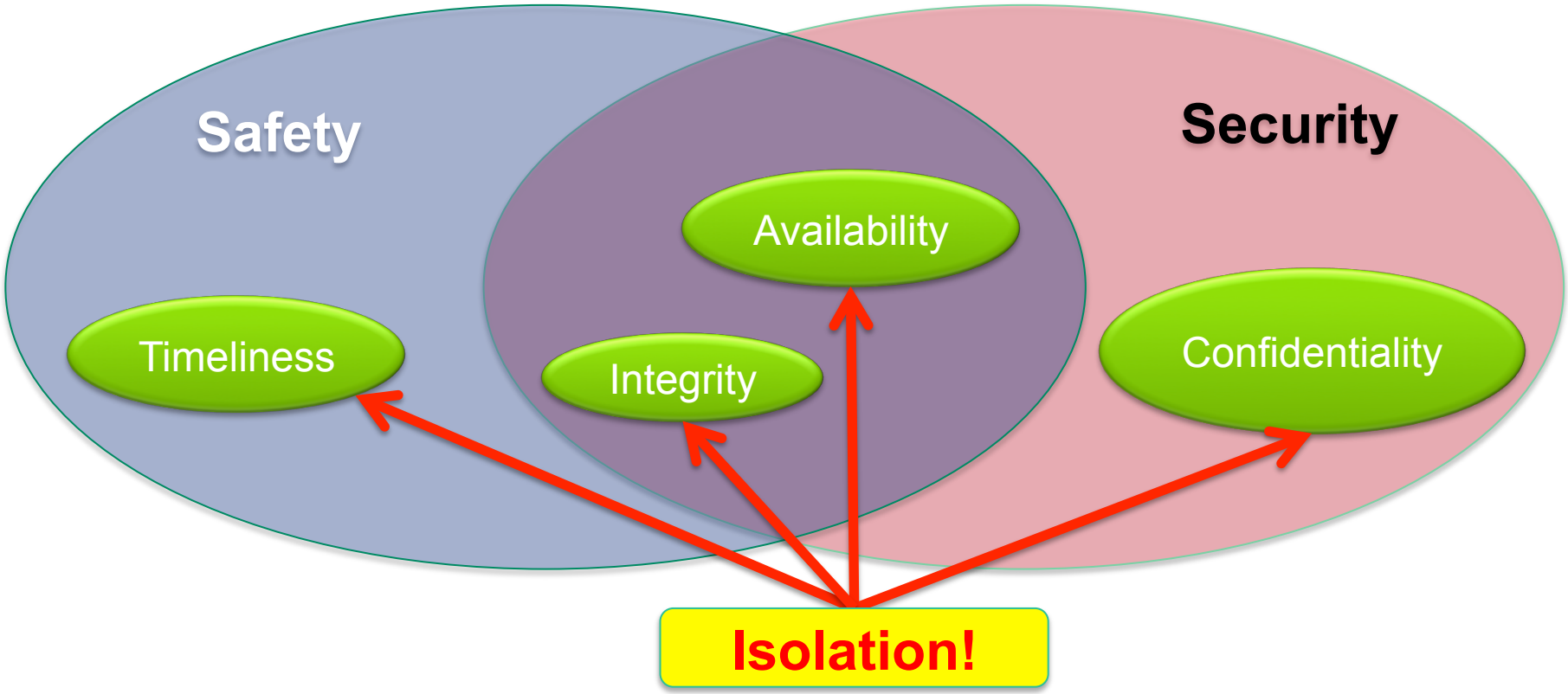
seL4: The world's only protected-mode OS with complete, sound timeliness analysis

seL4: The world's fastest microkernel

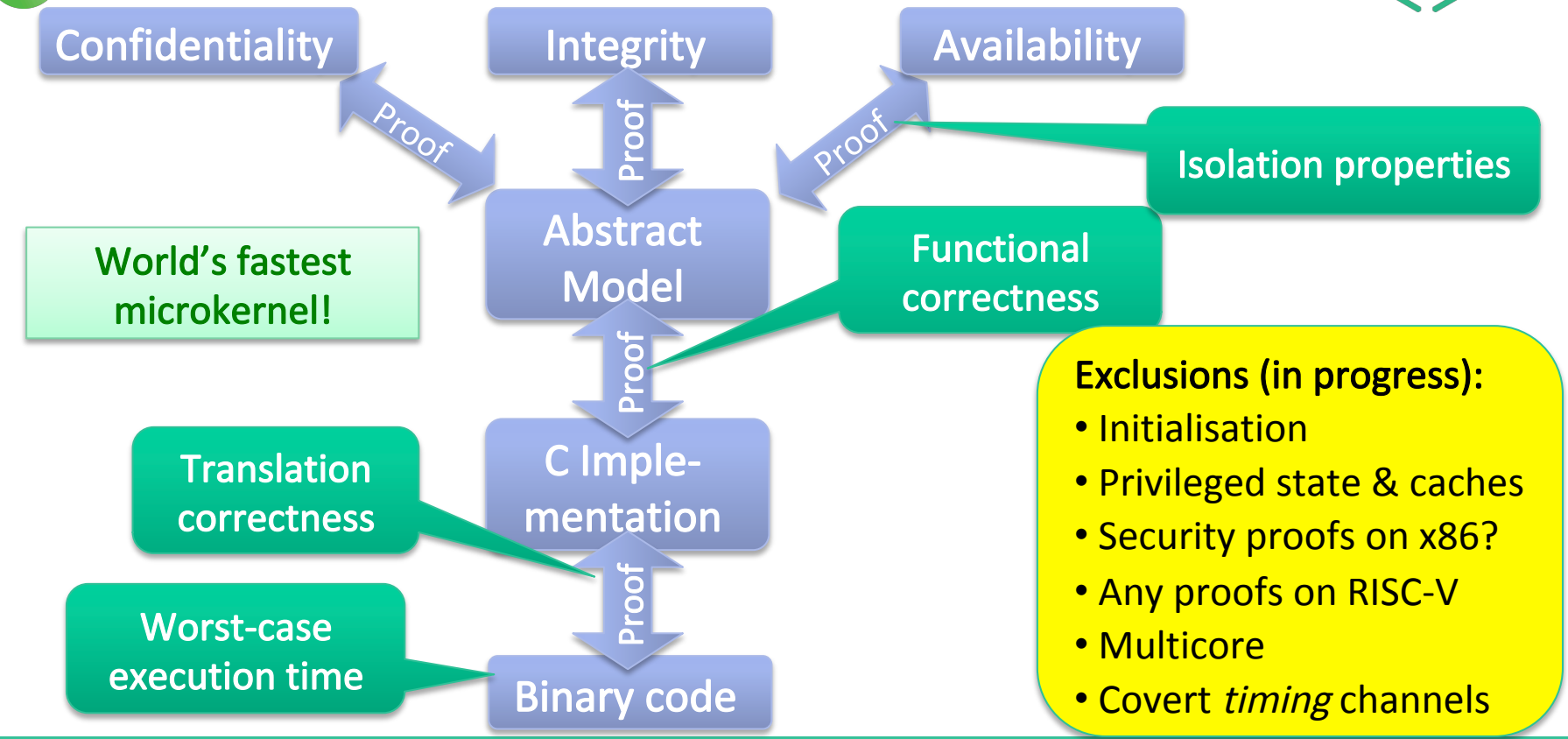
A Microkernel is not an OS



Security and Safety

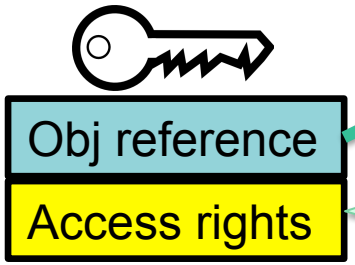


seL4 Provable Security Enforcement



Core Mechanism: Capability

Capability = Access Token:
Prima-facie evidence of privilege



Eg. thread, address space

Eg. read, write, send, execute…

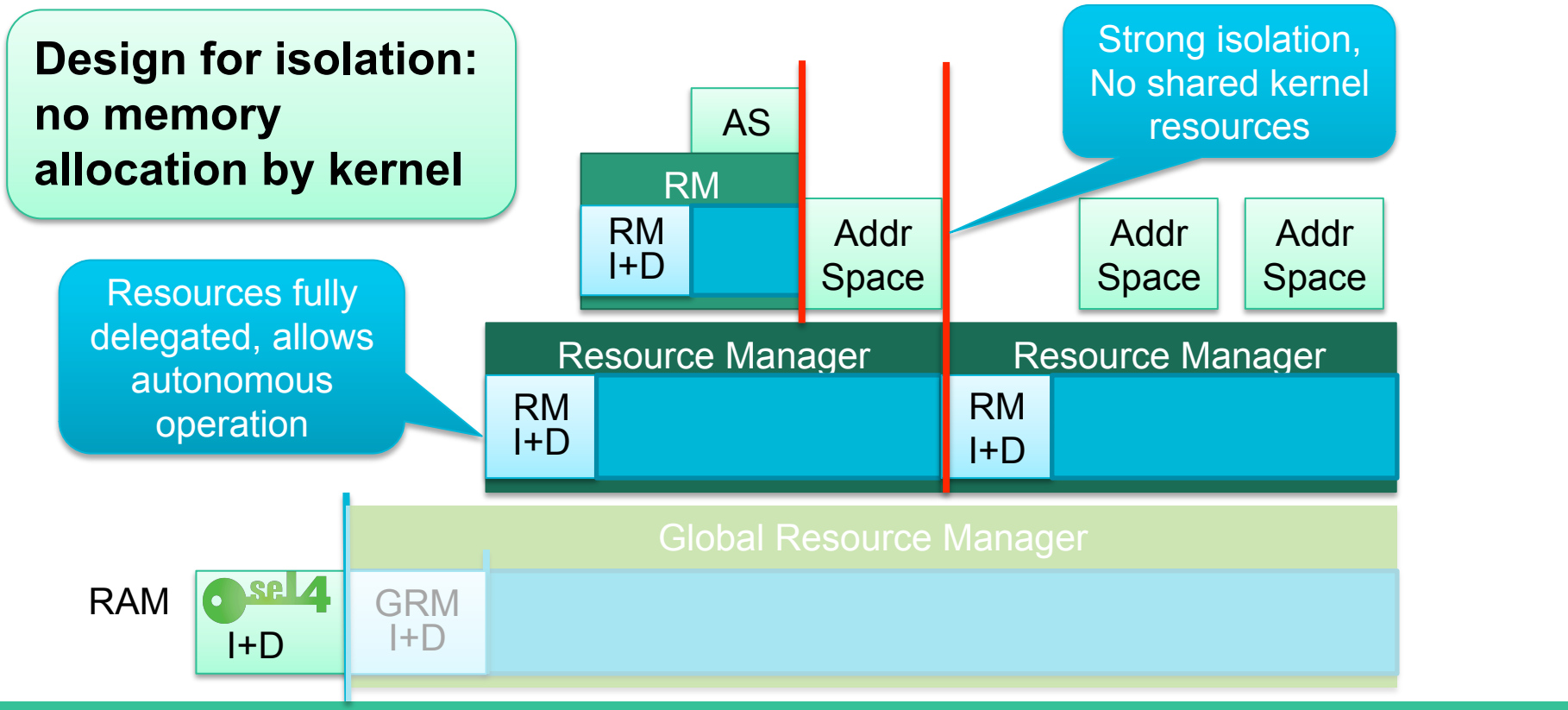
Capabilities provide:
- Fine-grained access control
- Reasoning about information flow

Any system call is invoking a capability:

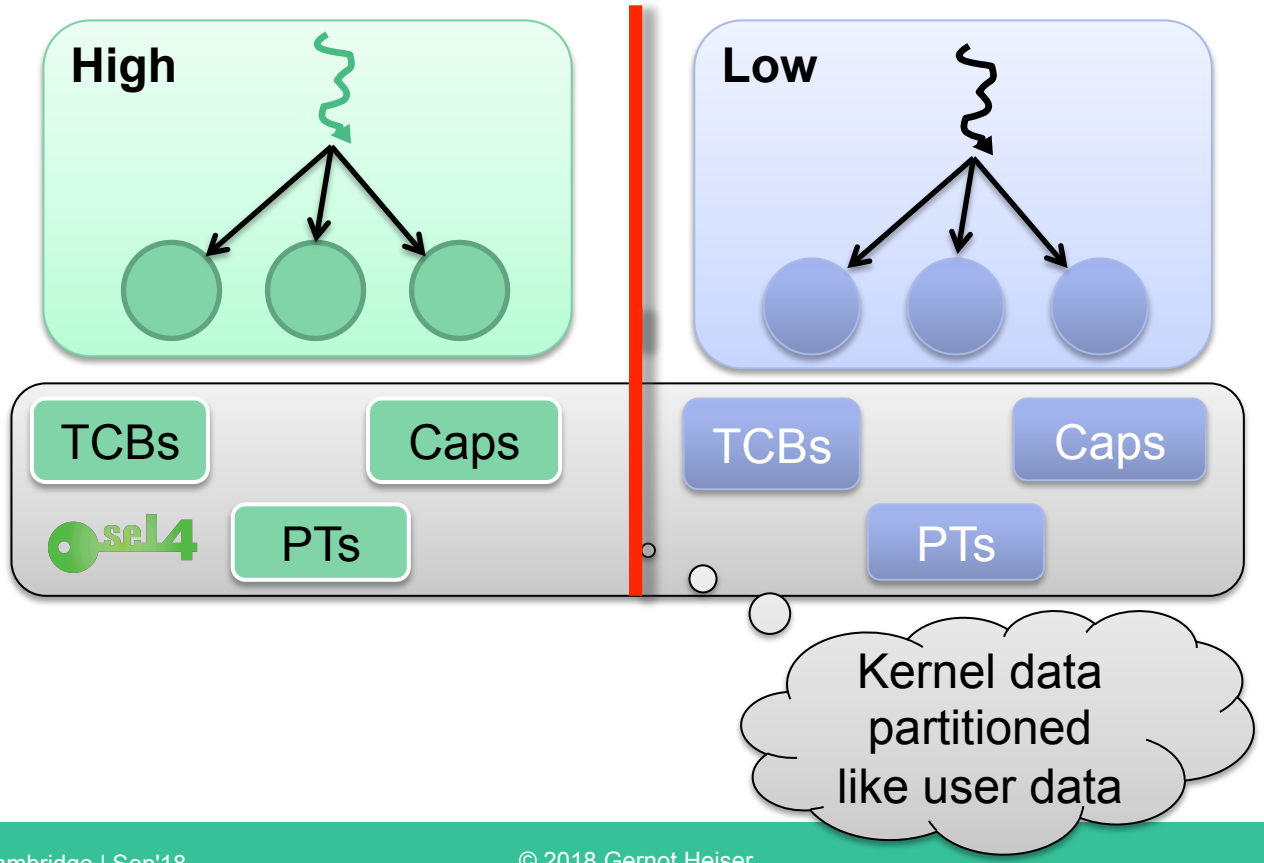
```
err = method( cap, args );
```

Difference To Other Kernels

Design for isolation: no memory allocation by kernel

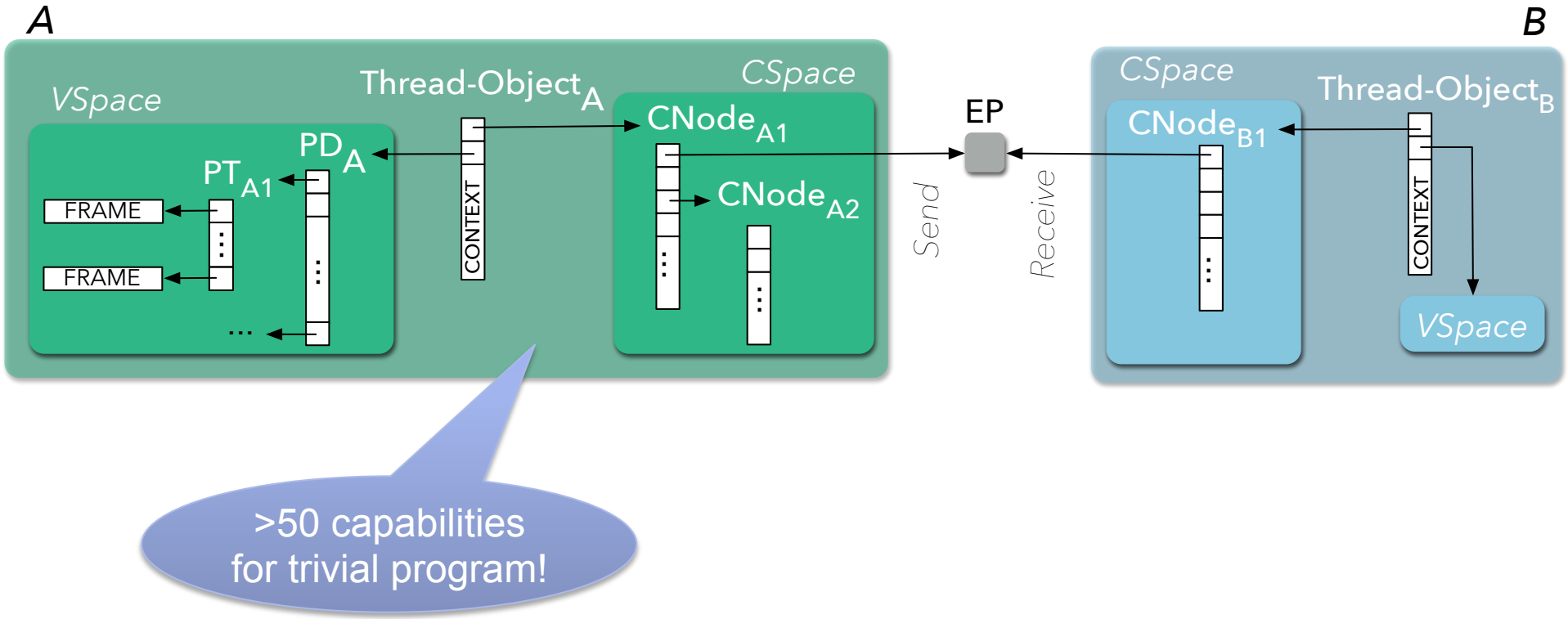


Isolation Goes Deep



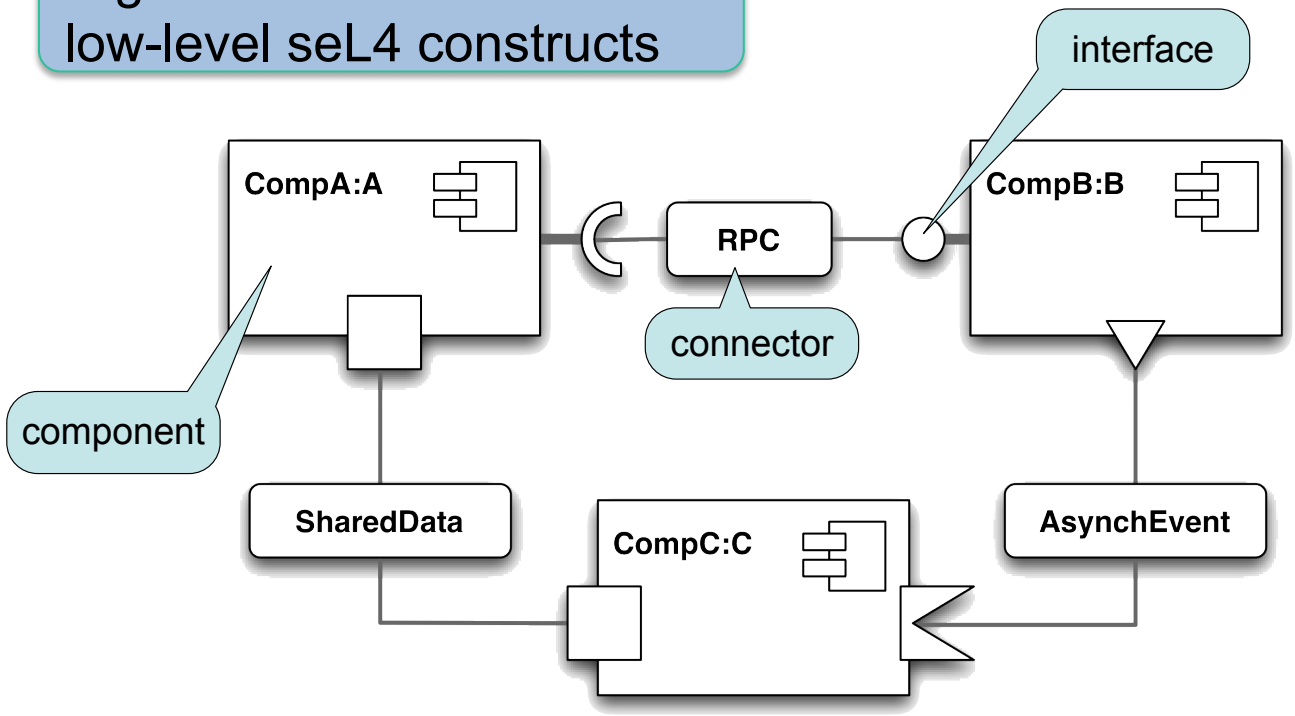

DATA
61
CSIRO
CAmkES:
Security by Architecture
se14

Issue: Capabilities are Low-Level



Component Middleware: CAmkES

Higher-level abstractions of low-level seL4 constructs

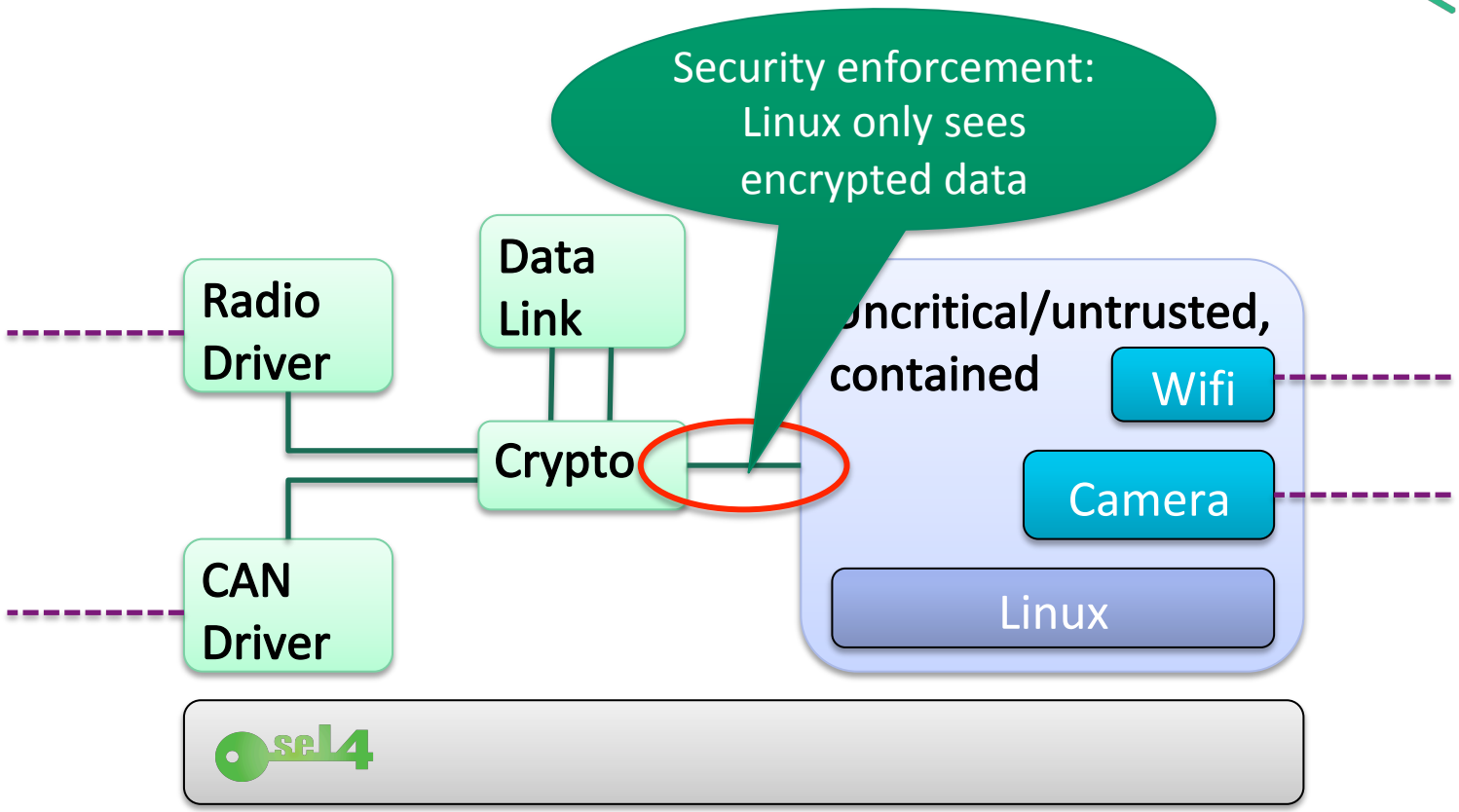


seL4 Example: DARPA HACMS

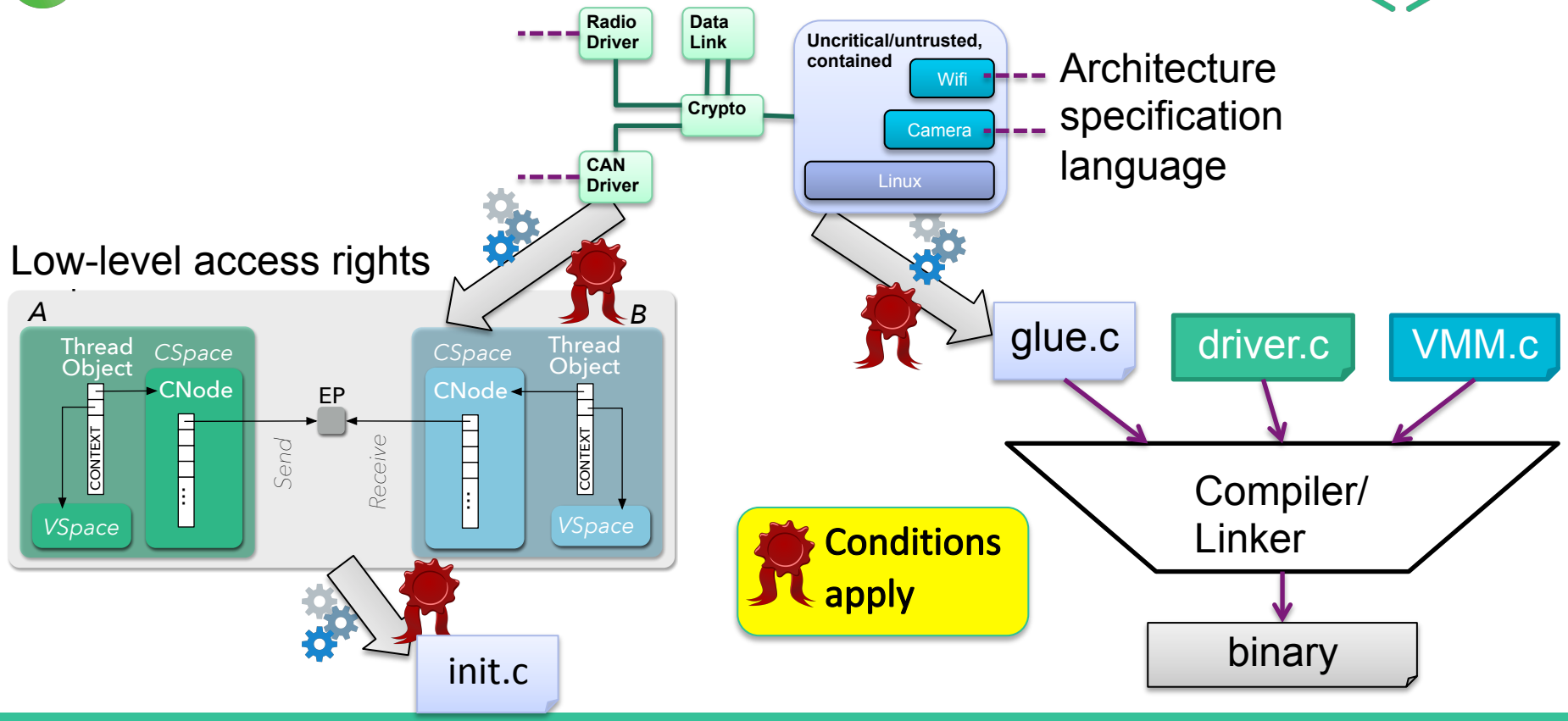
Retrofit existing system!

Develop technology

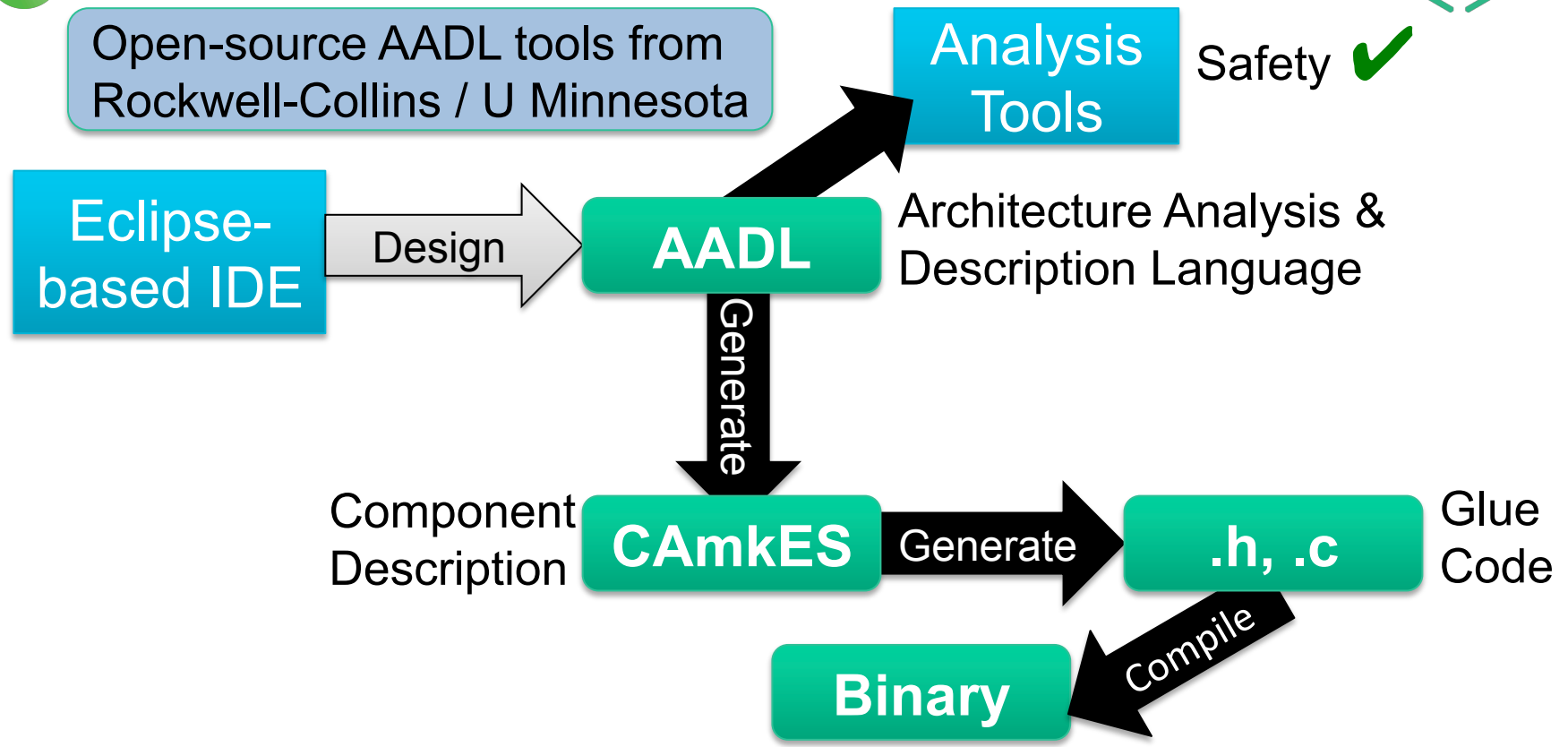
HACMS UAV Architecture



Enforcing the Architecture



Architecture Analysis



Military-Grade Security

Cross-Domain Desktop Compositor

Multi-level secure terminal
- Successful defence trial in AU
- Evaluated in US, UK, CA
- Formal security evaluation soon

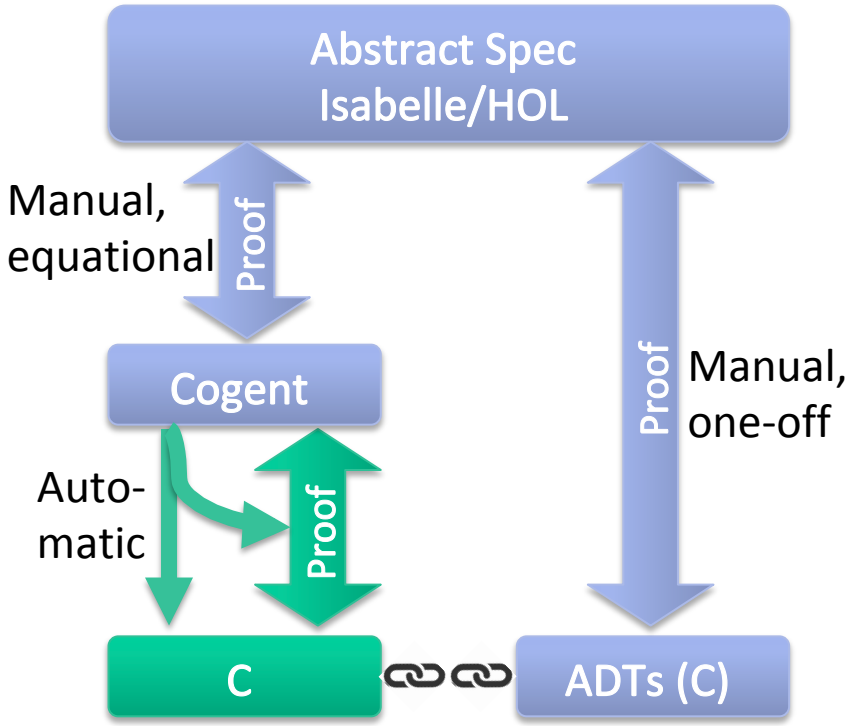
Pen10.com.au crypto communication device in use in AU defence

Cogent:
Reducing the Cost of Verification

Cogent: Code & Proof Co-Generation

- Reduces the cost of formally verifying systems code
- Restricted, purely functional language
- Uniqueness type system
- Case-studies: BilbyFs, ext2, F2FS, VFAT

[O'Connor et al, ICFP'16; Amani et al, ASPLOS'16]



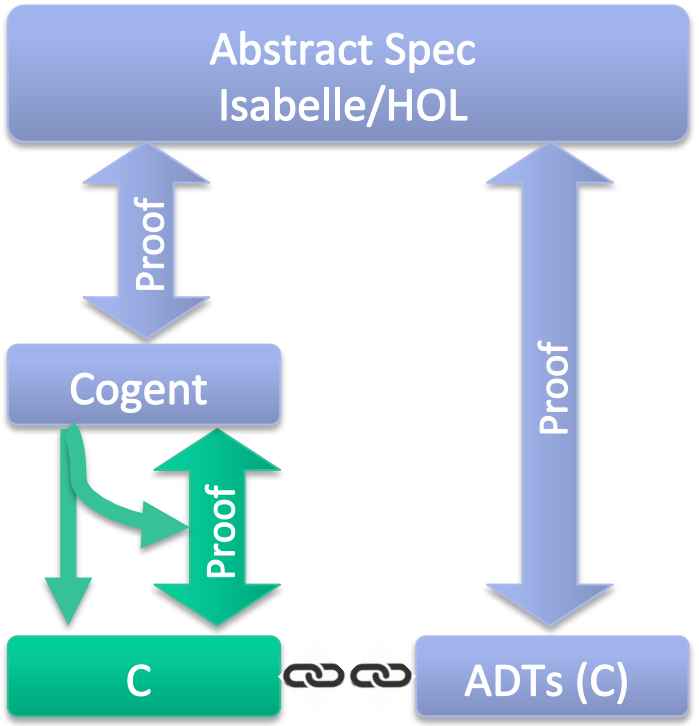
Manual Proof Effort

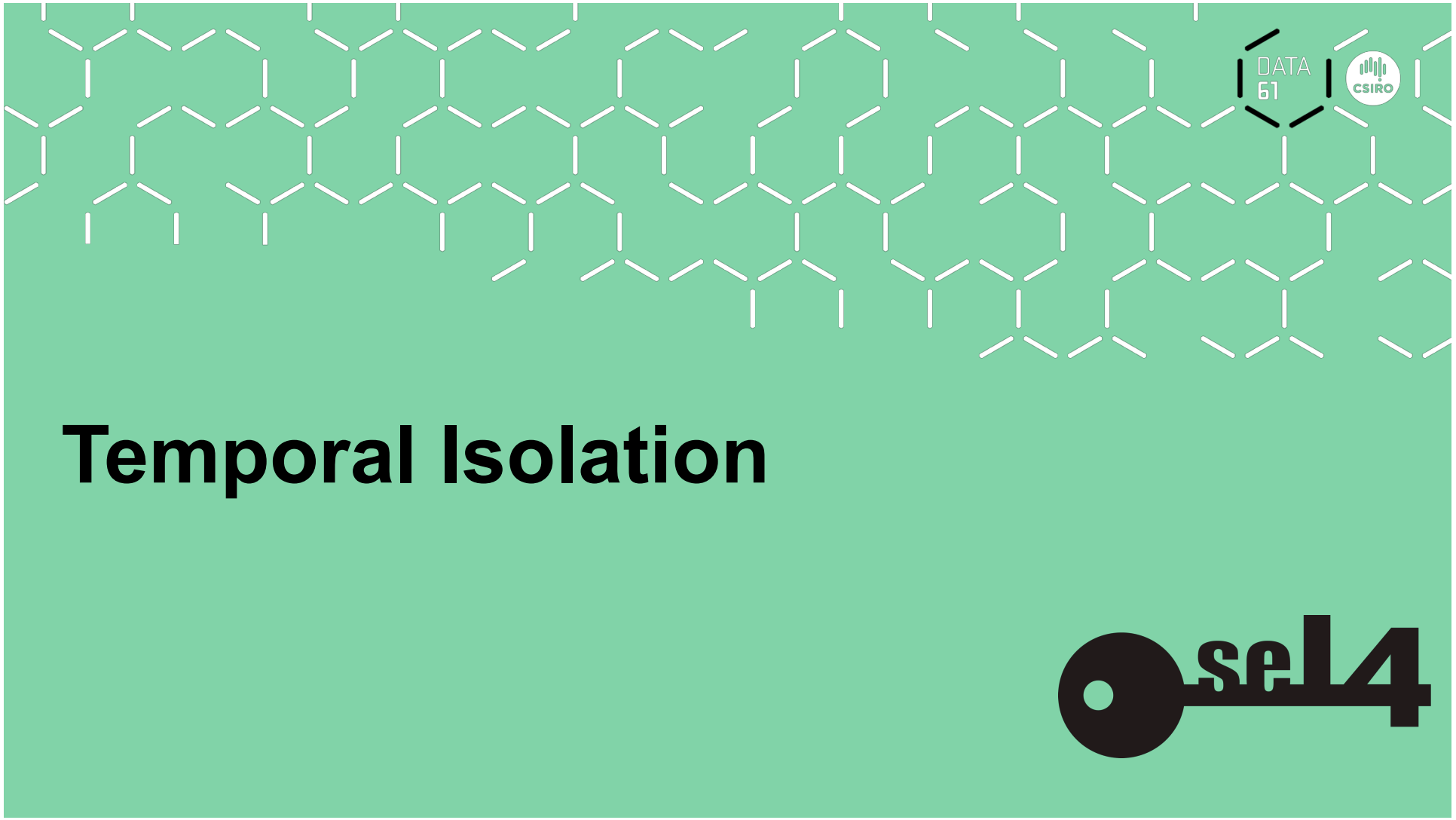
BilbyFS functions	Effort	Isabelle LoP	Cogent SLoC	Cost $/SLoC	LoP/ SLOC
isync() iget() library	9.25 pm	13,000	1,350	150	10
sync()-specific	3.75 pm	5,700	300	260	19
iget()-specific	1 pm	1,800	200	100	9
seL4	**12 py**	**180,000**	**8,700 C**	**350**	**20**

BilbyFS: 4,200 LoC Cogent

Cogent: Present Work

- Relax type-system restrictions
  - purely linear types lead to excessive copying, 100% overhead
- Extend base language with domain-specific syntactic sugar
  - increased expressiveness → less code
  - automate boilerplate code
- Apply to other systems code
  - device drivers
  - network protocols stacks




DATA
61
CSIRO
Temporal Isolation
seL4

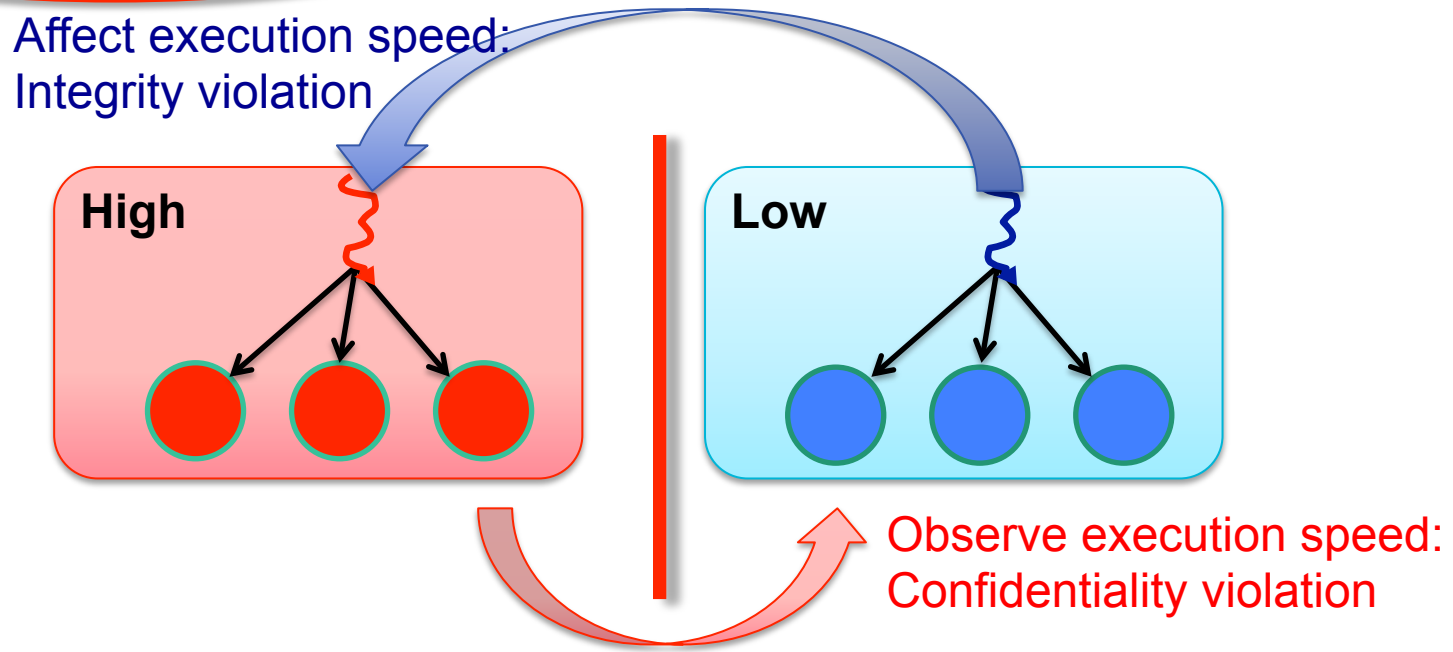
Temporal Interference

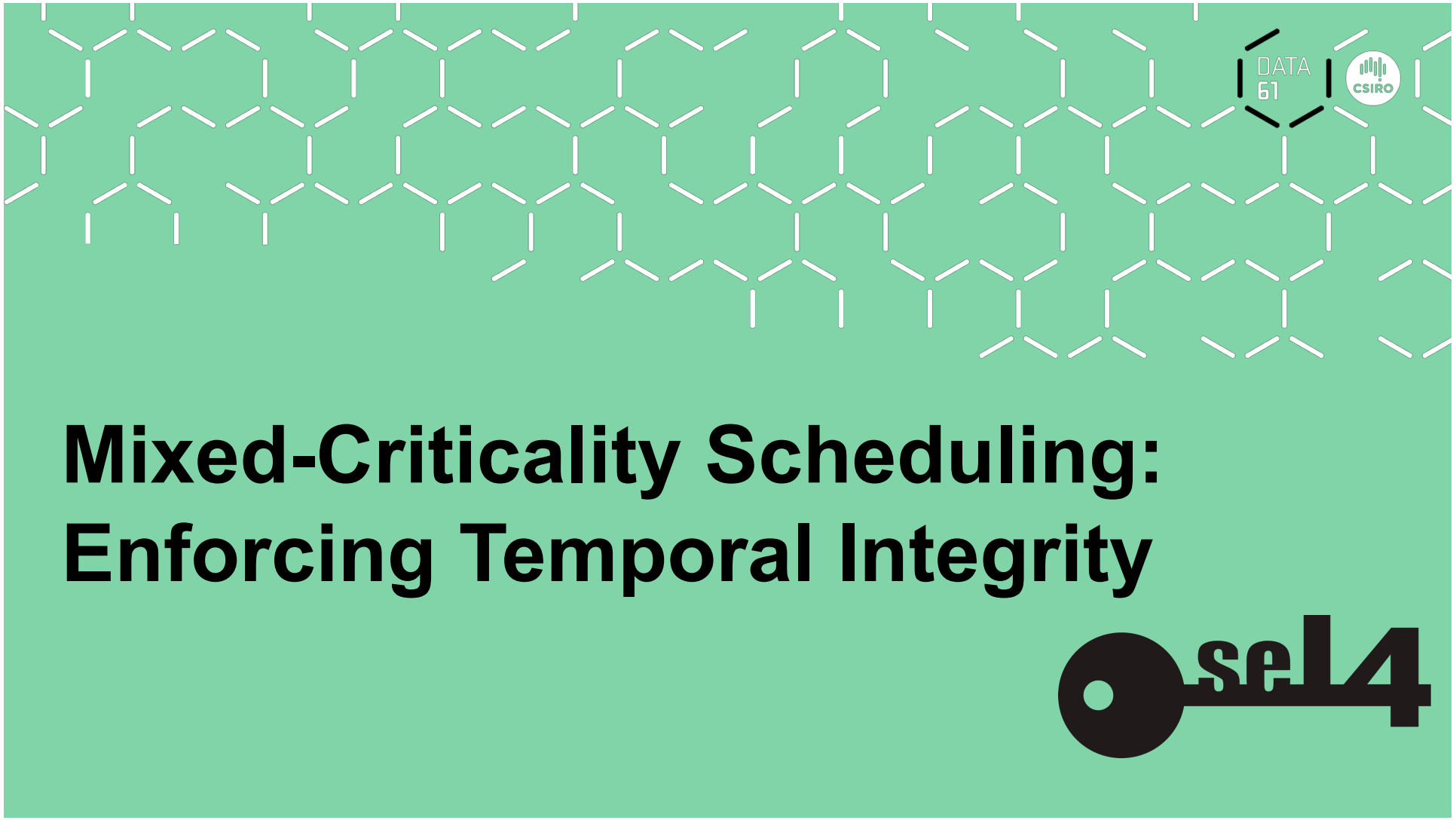
Safety: Timeliness

- Execution interference

Security: Confidentiality

- Leakage via timing channels



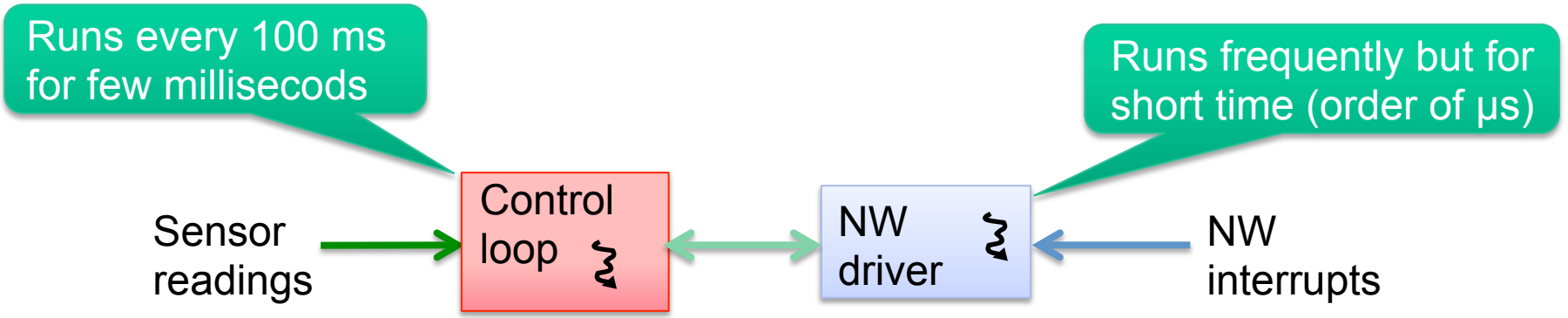


Mixed-Criticality Scheduling: Enforcing Temporal Integrity

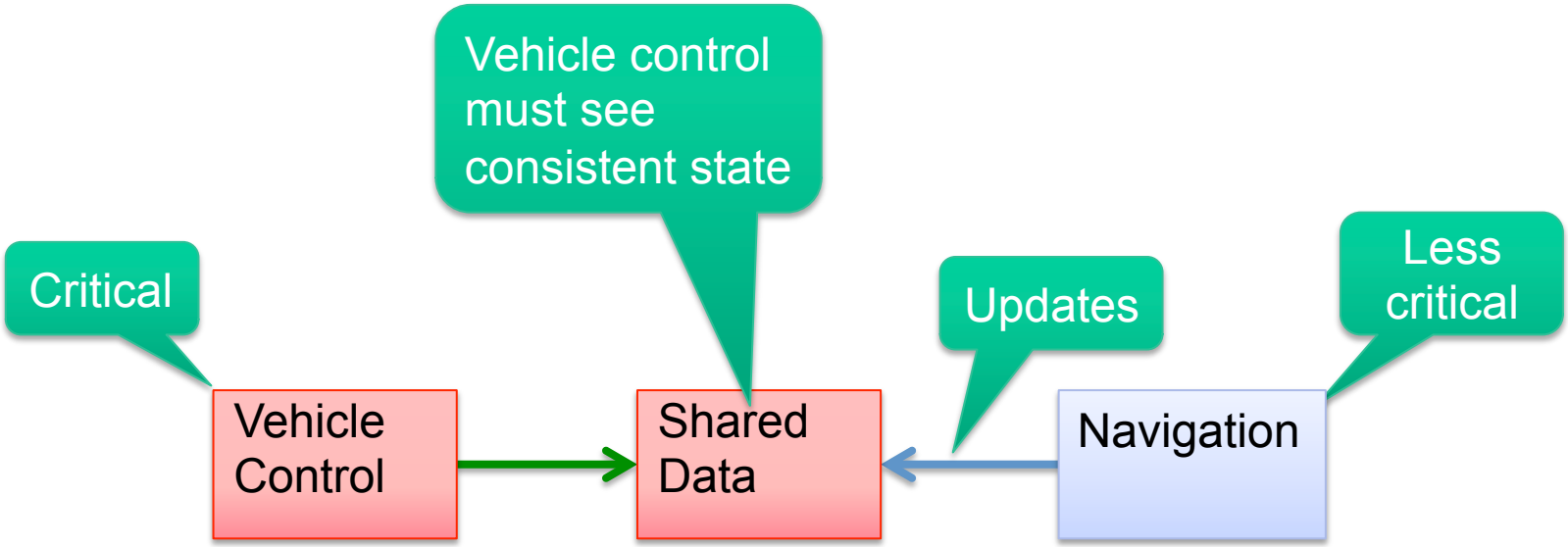
Integration Challenge: Mixed Criticality

NW driver must preempt control loop

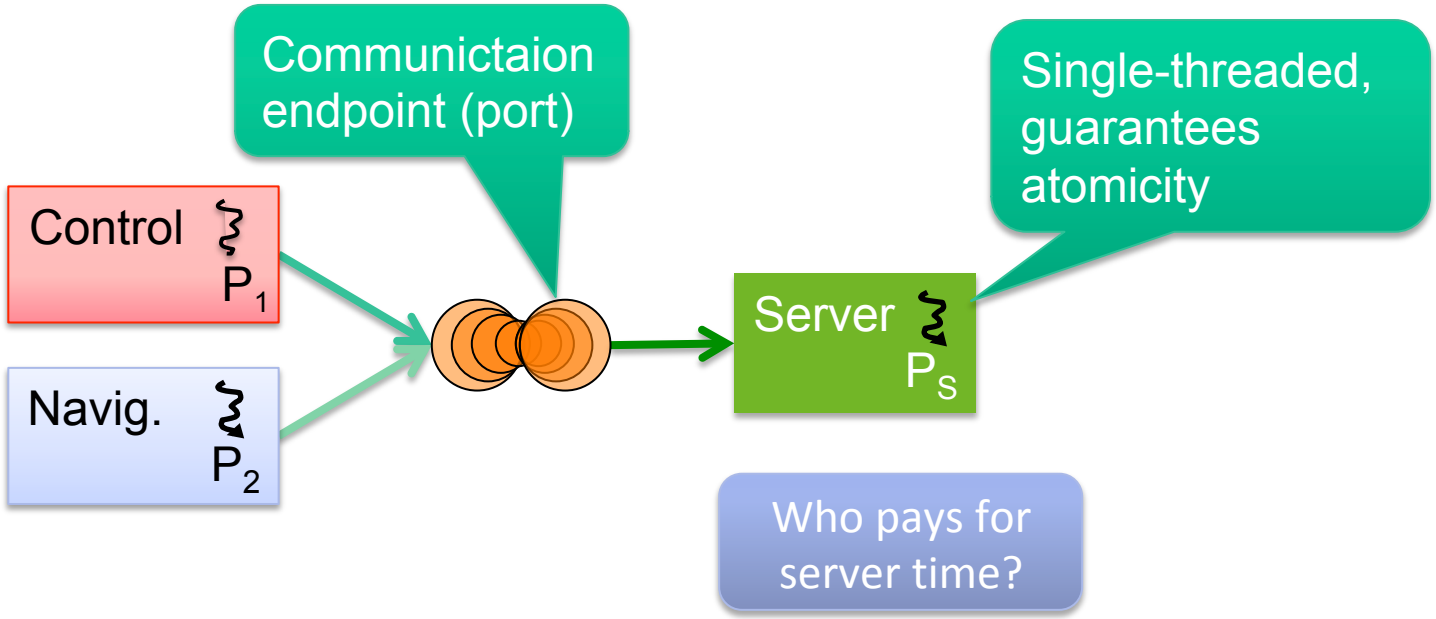
- … to avoid packet loss
- Driver must run at high prio
- Driver must be trusted not to monopolise CPU



Integration Challenge: Sharing



Sharing Through *Resource Server*



Scheduling Contexts: Time Caps

Classical thread attributes

- Priority
- Time slice

Not runnable if null

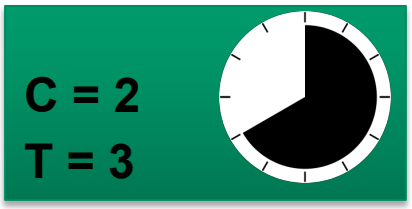
Limits CPU access!

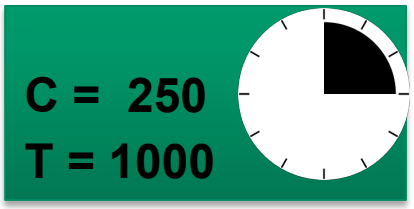
New thread attributes

- Priority
- Scheduling context capability

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

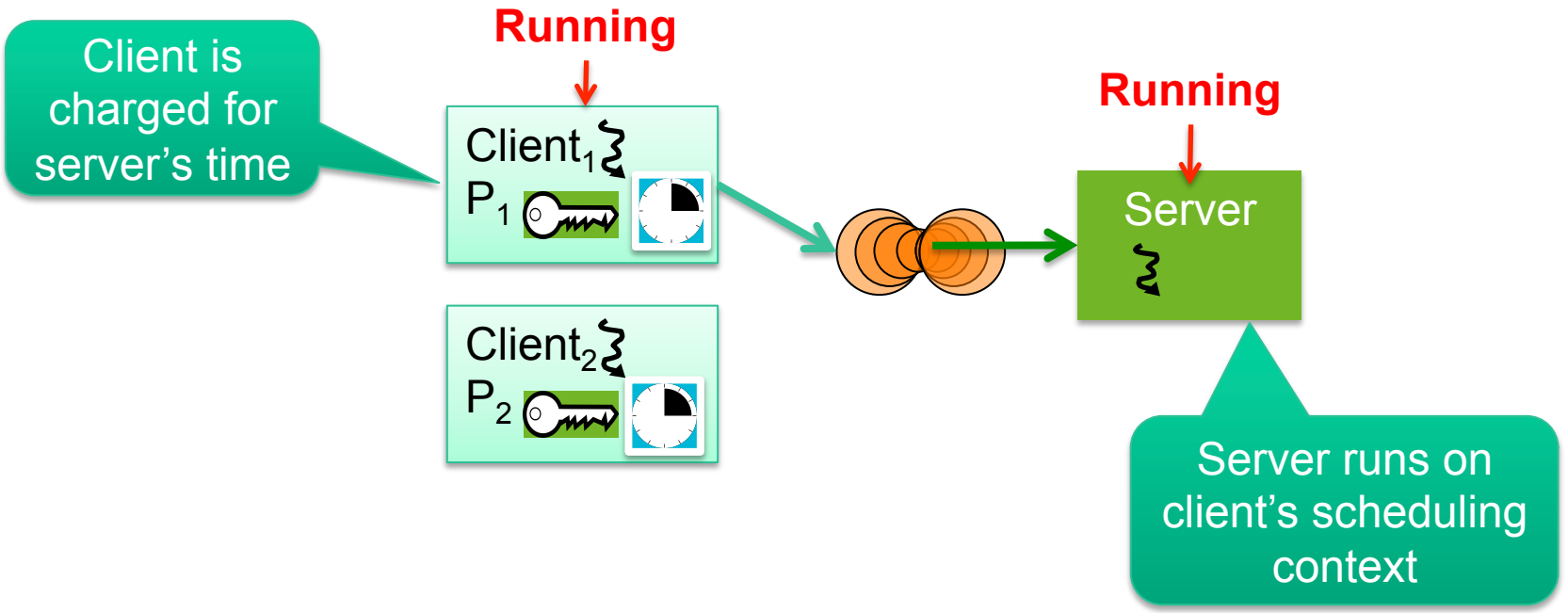
Capability for time





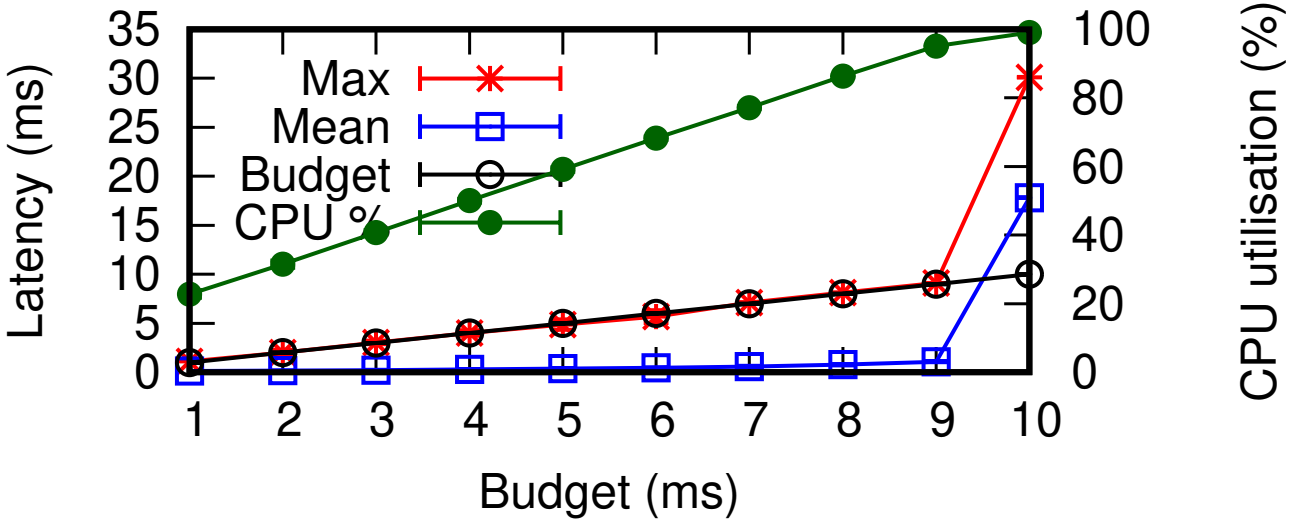
SchedControl capability conveys right to assign budgets (i.e. perform admission control)

Shared Server



seL4 Isolation in Action

- High-prio CPU hog, budget limited, 10ms period
- Lower-prio UDP echo server, 10ms period



MCS kernel will replace current mainline once verified

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


DATA
61
CSIRO
Time Protection:
Removing Timing Channels
seL4

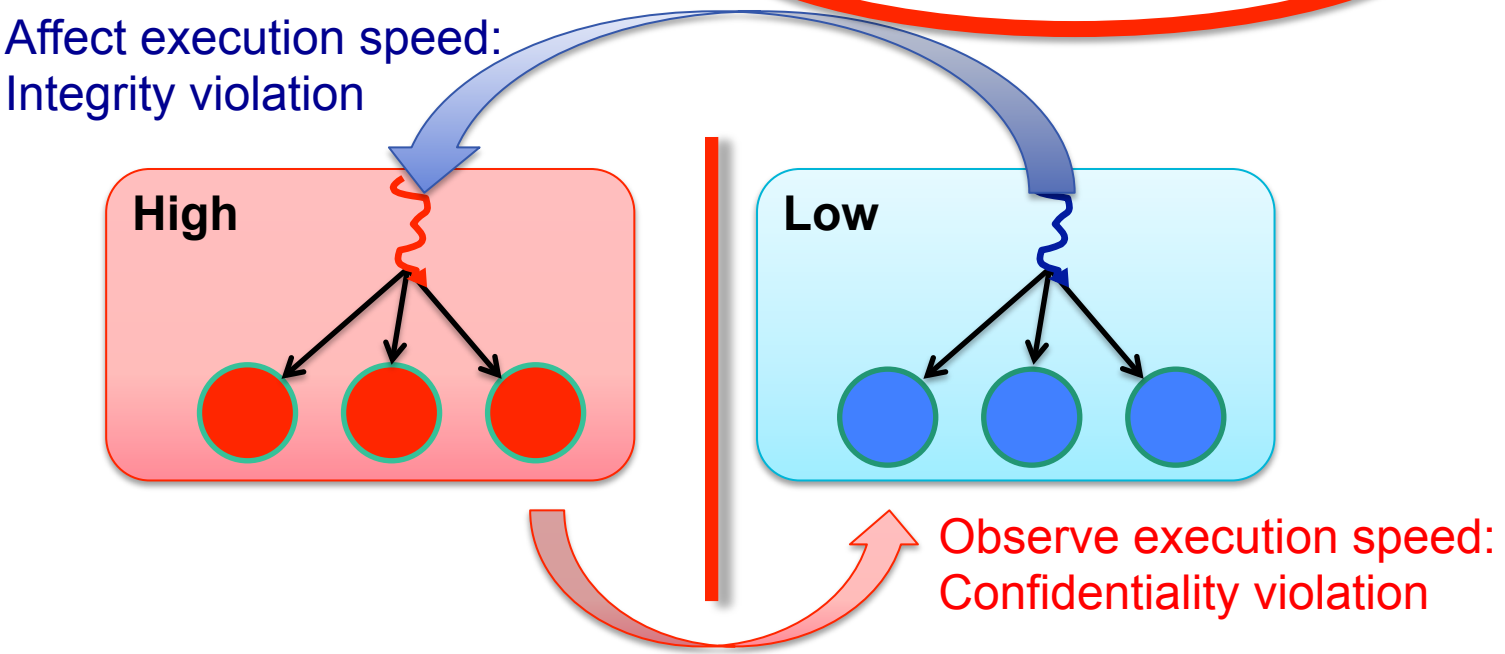
Temporal Interference

Safety: Timeliness

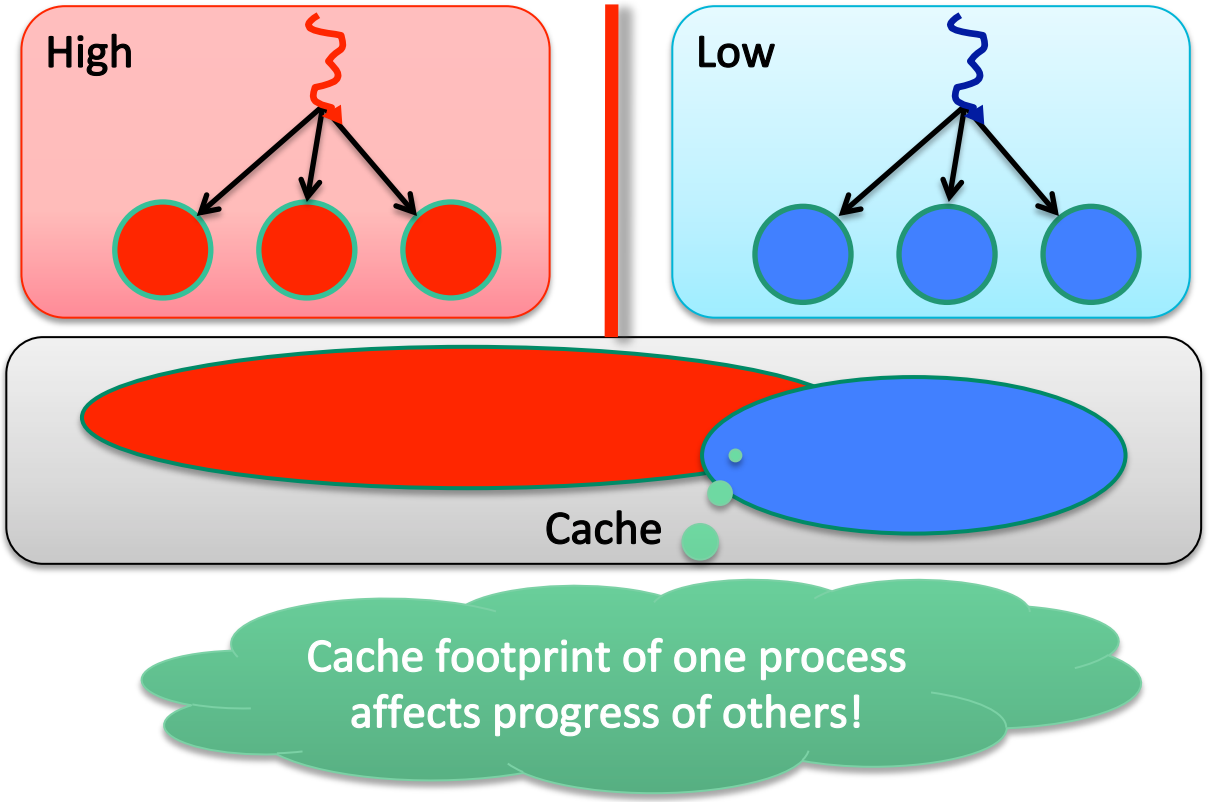
- Execution interference

Security: Confidentiality

- Leakage via timing channels



Cause: Conflicts on Shared HW



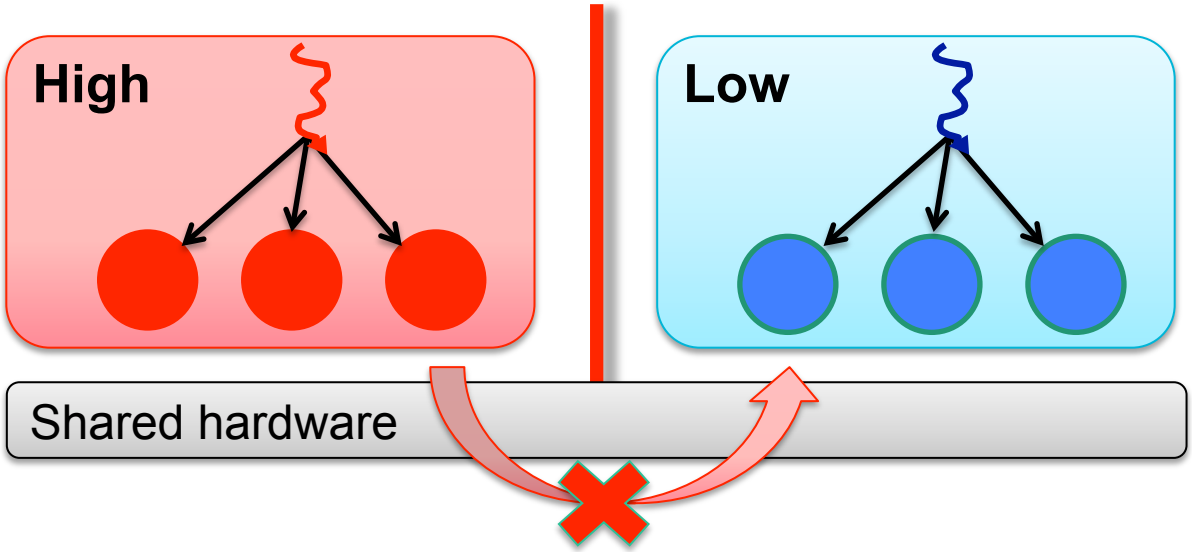
Sharing can be:

- Concurrent multicore, HW thread
- Time-shared

"Caches" include:

- L1 I-, D-cache
- TLB
- Branch predictor
- Instr. prefetcher
- Data prefetcher
- off-core caches & busses

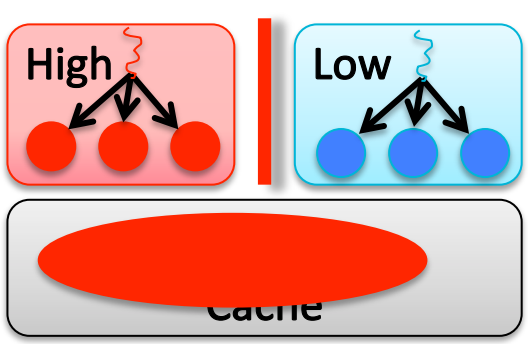
OS Must Enforce *Time Protection*

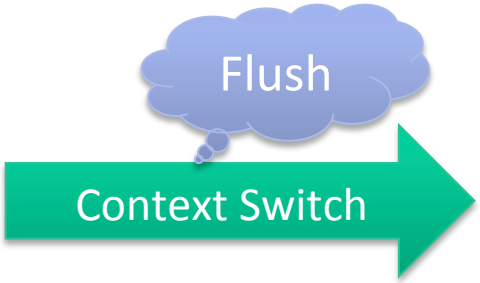


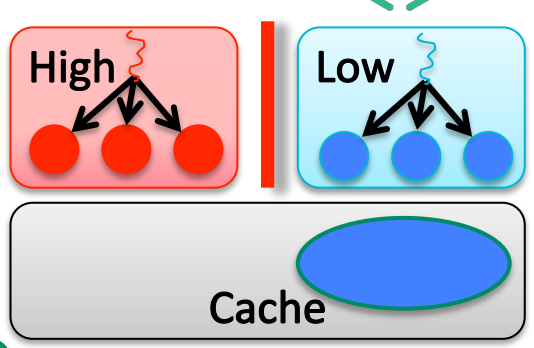
Preventing interference is core duty of the OS!

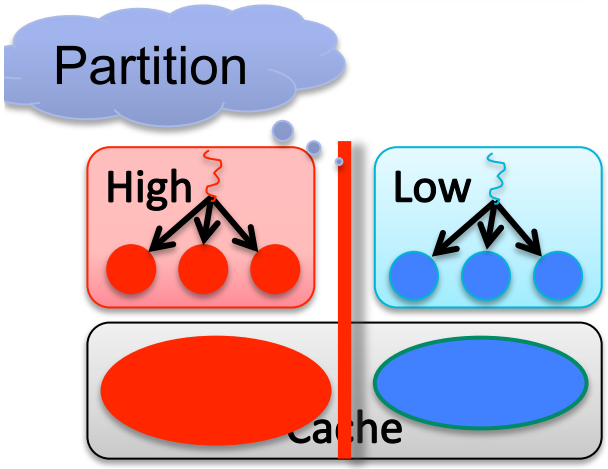
- *Memory protection* is well established
- *Time protection* is completely absent

Time Protection: No Sharing of State









Need both!

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

- virtually-indexed
- OS cannot control

Flushing useless for concurrent access

- between HW threads, cores
- for stateless HW

Flushing on Domain Switch

Latency depends on prior execution!

1. T_0 = current_time()
2. Switch context
3. Flush caches
4. Touch all code/data needed for return
5. Reprogram timer
6. while (T_0+WCET < current_time()) ;
7. return

Ensure deterministic execution

Remove dependency

Can Time Protection Be Verified?

1. Correct treatment of partitionable state:
   - Need hardware model that identifies all such state (augmented ISA)
   - Enables *functional correctness* argument:
     No two domains can access the same physical state

Transforms timing channels into storage channels!

2. Correct flushing of non-partitionable state
   - Not trivial: eg proving all cleanup code/data are forced into cache after flush
     – Needs an actual cache model
   - Even trickier: need to prove padding is correct
     – … without explicitly reasoning about time!

How Can We Prove Time Padding?

- Idea: Minimal formalisation of hardware clocks (logical time)
  - Monotonically-increasing counter
  - Can add constants to time values
  - Can compare time values

To prove: padding loop terminates as soon as timer value ≥ T_0+WCET

Functional property

Summary

- **seL4** has most comprehensive verified spatial isolation
- **Cogent** project aims to reduce cost of verified system components
  - file systems, device drivers, network protocol stacks
- **Scheduling contexts** introduce capability-controlled time management
  - support *mixed-criticality systems* where priority ≠ criticality
  - temporal integrity enforcement
- **Time protection** aims to eliminate microarchitectural timing channels
  - extend isolation to time
  - ideas on how to verifiy

Thank you!


DATA
61
Security is no excuse for poor performance!
Gernot Heiser | Microkernel Dude
Gernot.Heiser@data61.csiro.au | @GernotHeiser
https://sel4.systems
seL4
CSIRO