Protecting Autonomous Operation With A High-Assurance OS

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

WG10.4 Winter'19, Champéry, CH

https://seL4.systems

Military-Grade Autonomous System?

Hacked within 2 weeks by professional pen-testers!

No safety without cyber-security!

DARPA HACMS: Protected Autonomy

Unmanned Little Bird (ULB)

Retrofit existing system!

Autonomous Truck

Develop technology

Outline

1. seL4 Introduction
2. Mixed-criticality support
3. Security enforcement by architecture
4. High-assurance user-level components

Foundation:
seL4 Microkernel

seL4 Foundation for Security

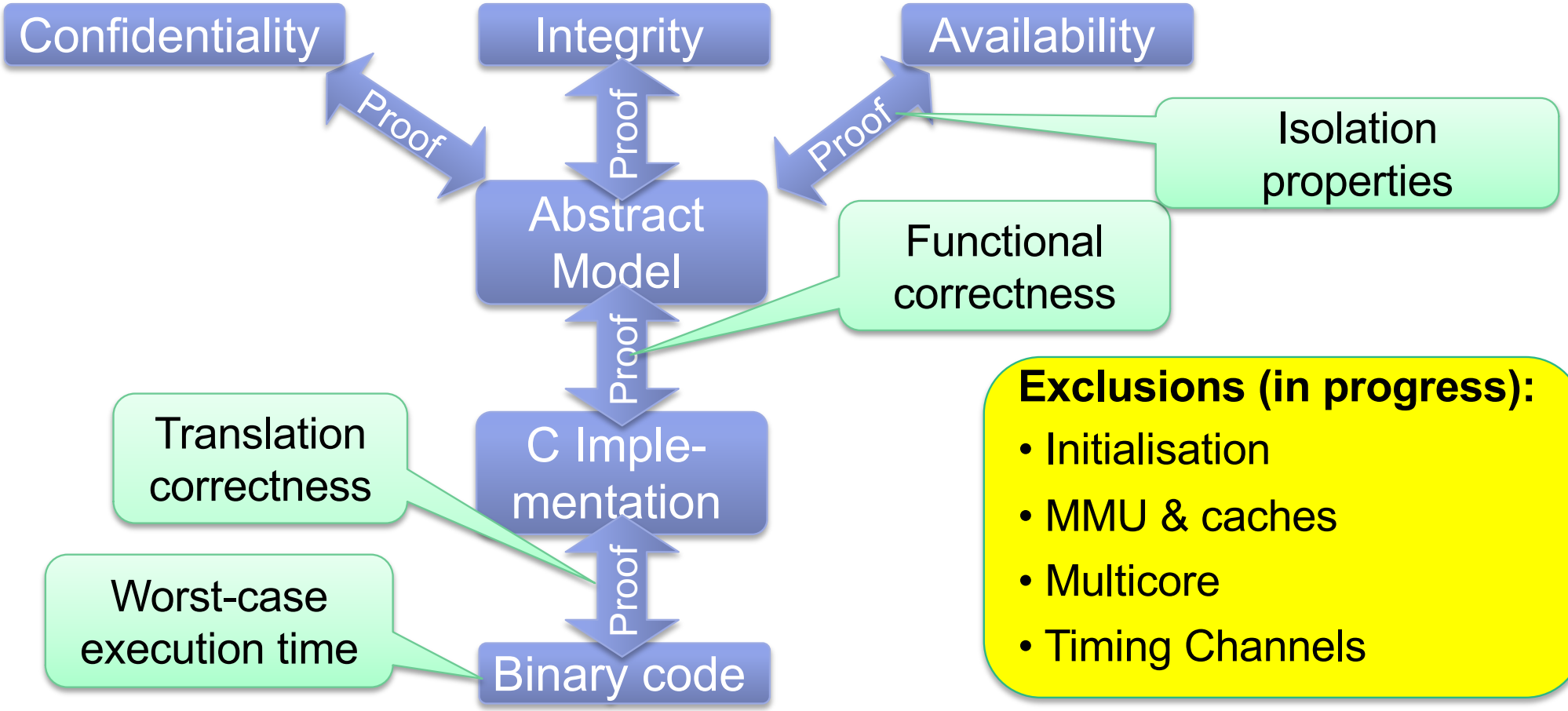
seL4: The world's **only** operating-system kernel with **provable** security enforcement (incl. memory protection)

Open Source

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

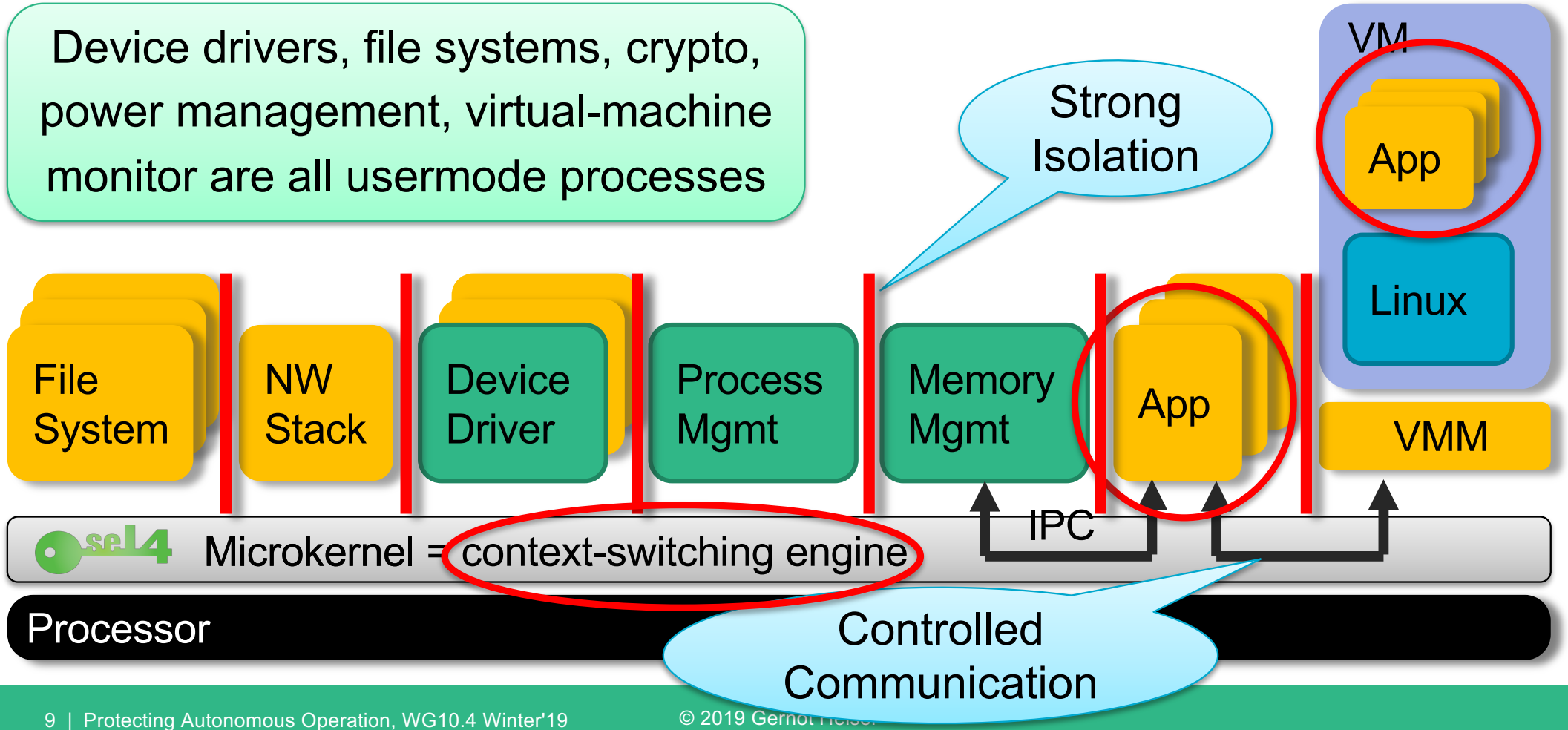
seL4: The world's **fastest** microkernel

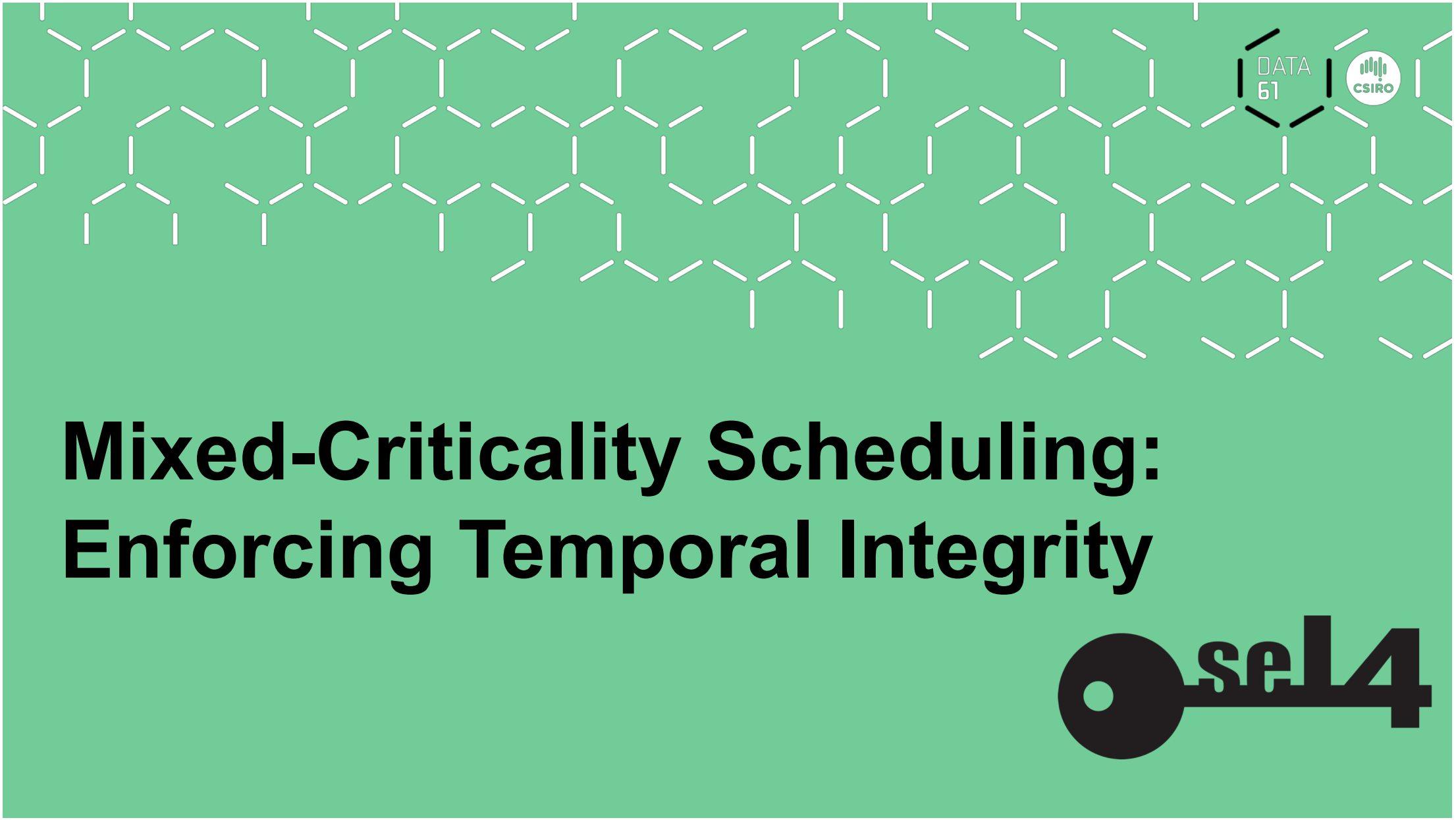
Assurance Proof Chain



A Microkernel is not an OS

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



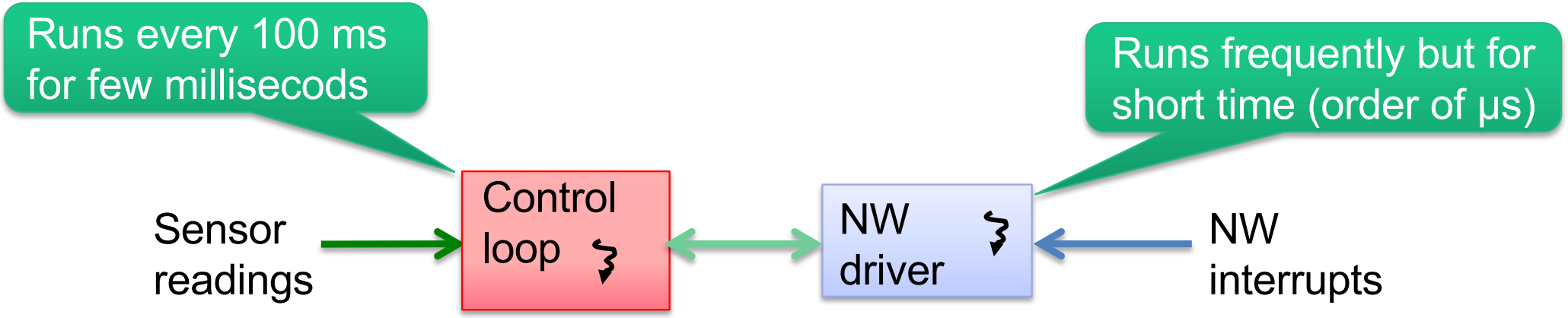


Mixed-Criticality Scheduling: Enforcing Temporal Integrity

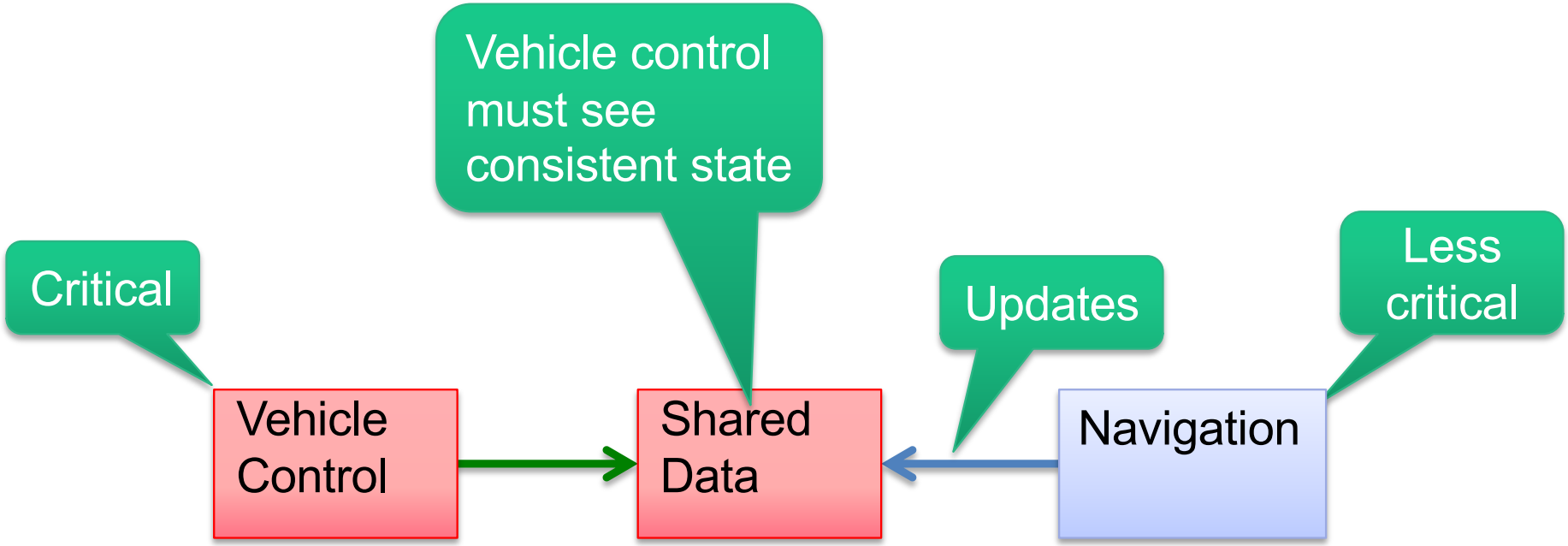
Integration Challenge: Mixed Criticality

NW driver must preempt control loop

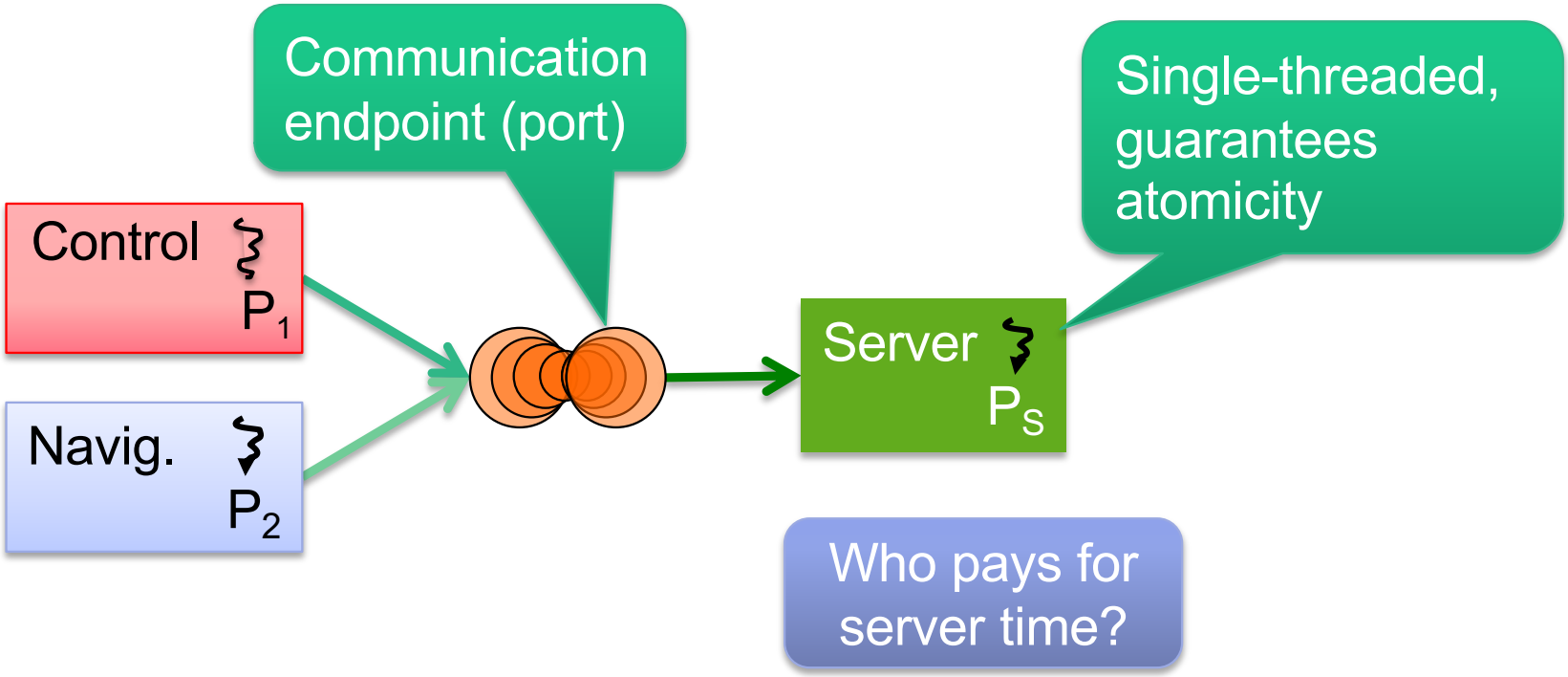
- … to avoid packet loss
- Driver must run at high prio
- Driver must be trusted not to monopolise CPU



Integration Challenge: Sharing



Sharing Through *Resource Server*



Scheduling Contexts: Time Caps

Classical thread attributes

- Priority
- Time slice

Not runnable if null

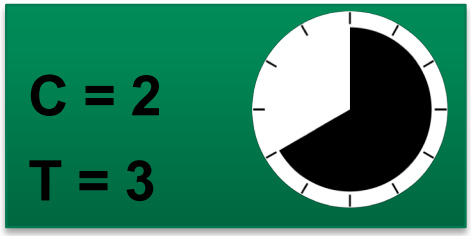
New thread attributes

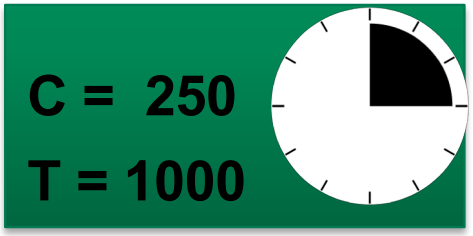
- Priority
- Scheduling context capability

Limits CPU access!

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

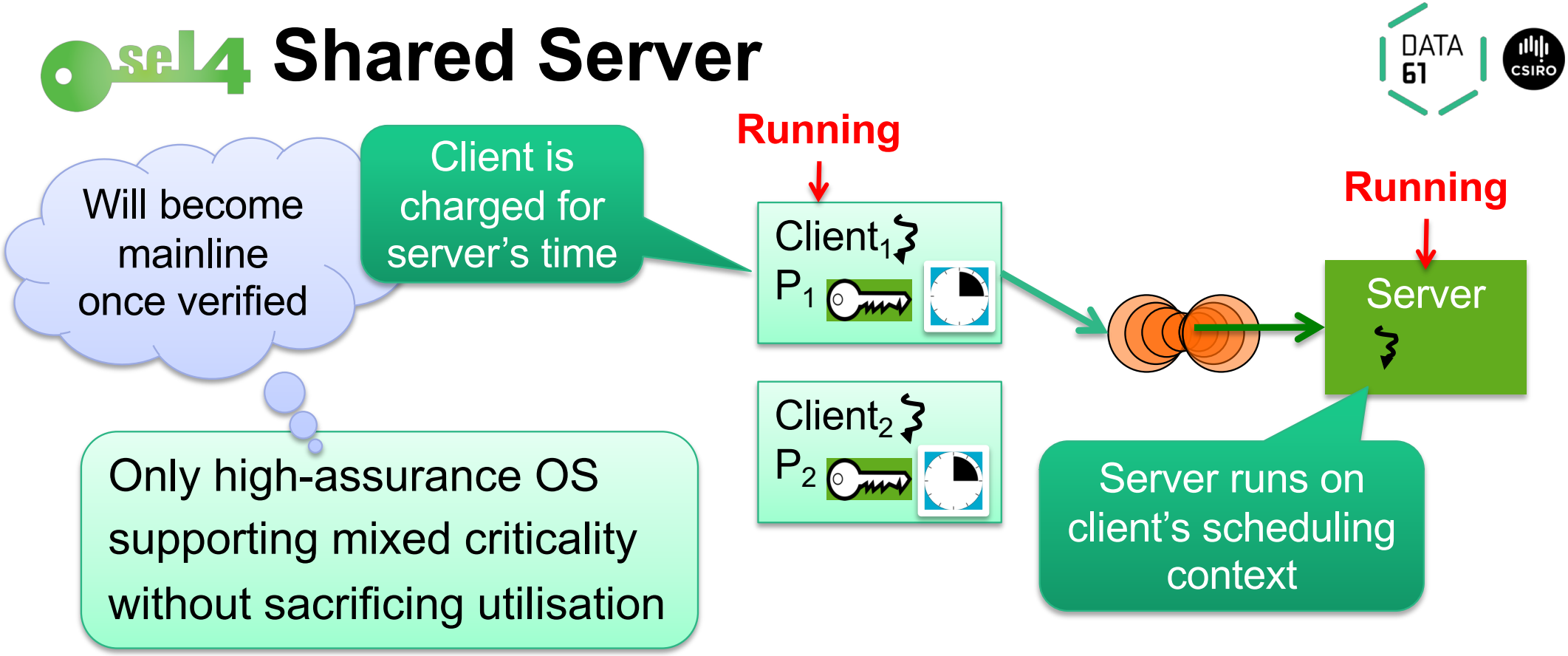
Capability for time





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

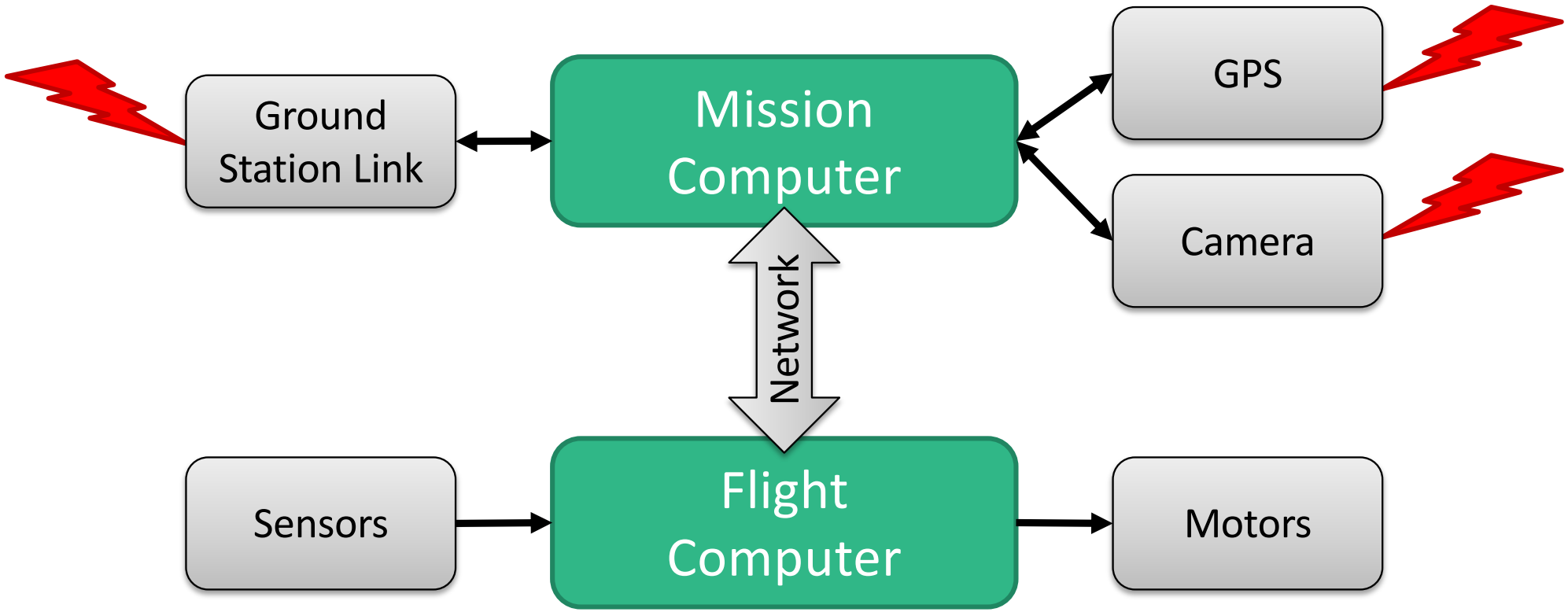
Shared Server



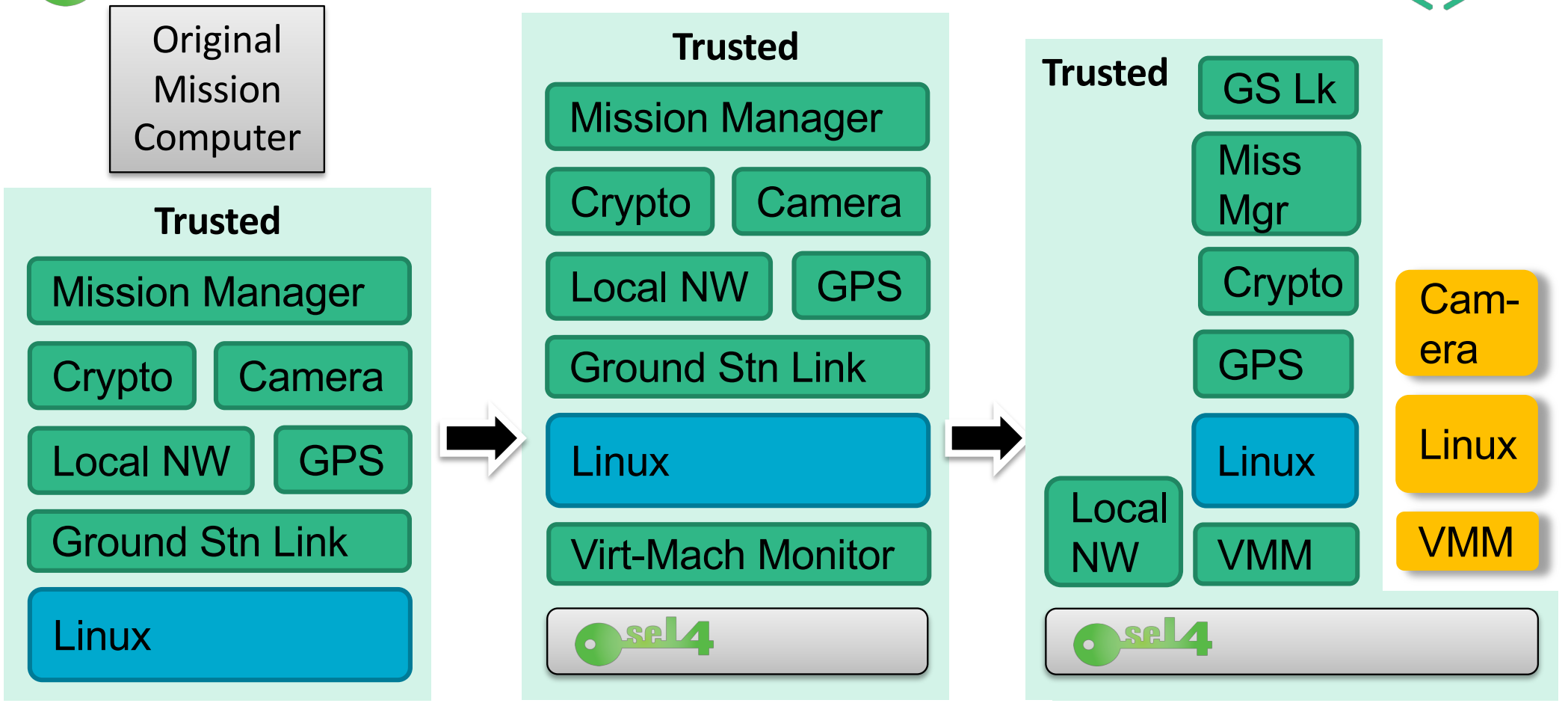
Scheduling-context capabilities: a principled, light-weight OS mechanism for managing time [Lyons et al, EuroSys'18]


DATA
61
CSIRO
Security Enforcement
by Architecture
seL4

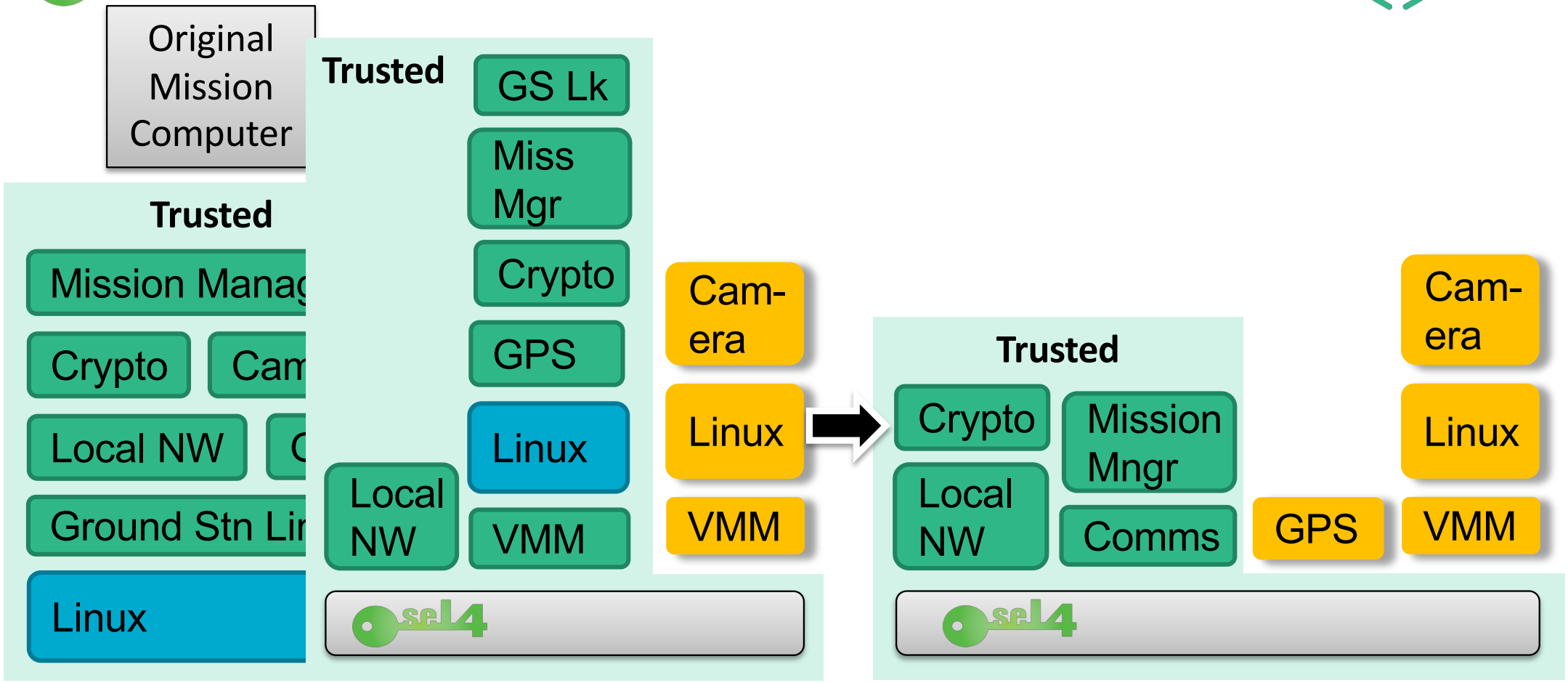
ULB Architecture



Incremental Cyber Retrofit



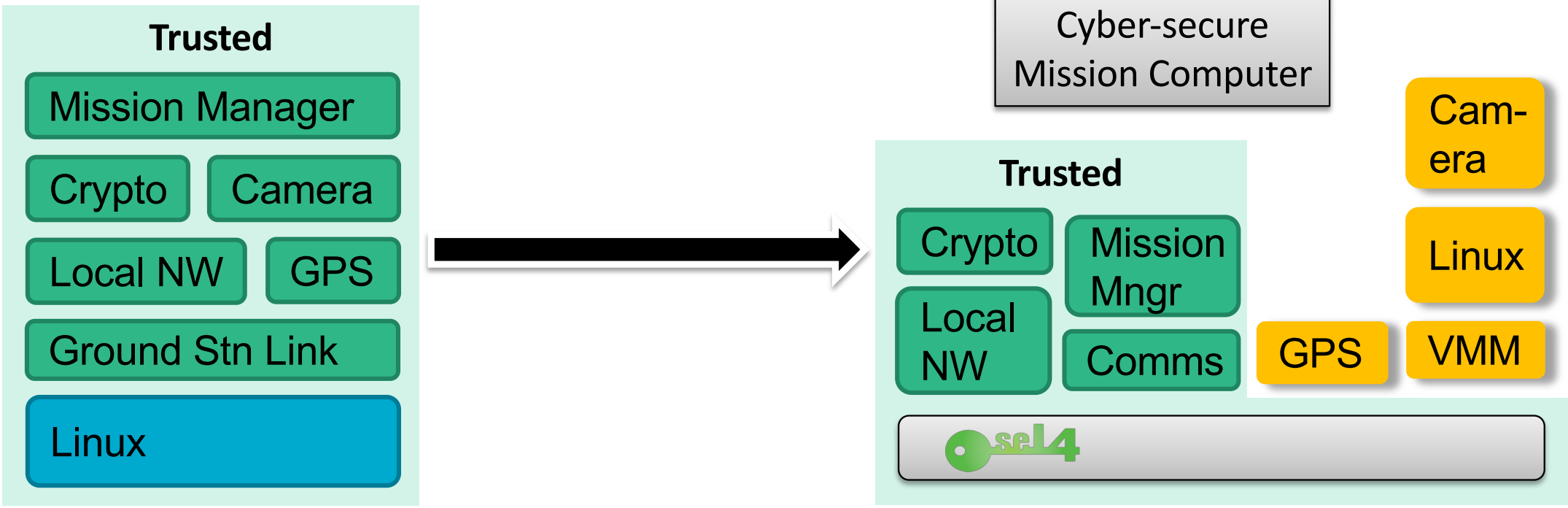
Incremental Cyber Retrofit



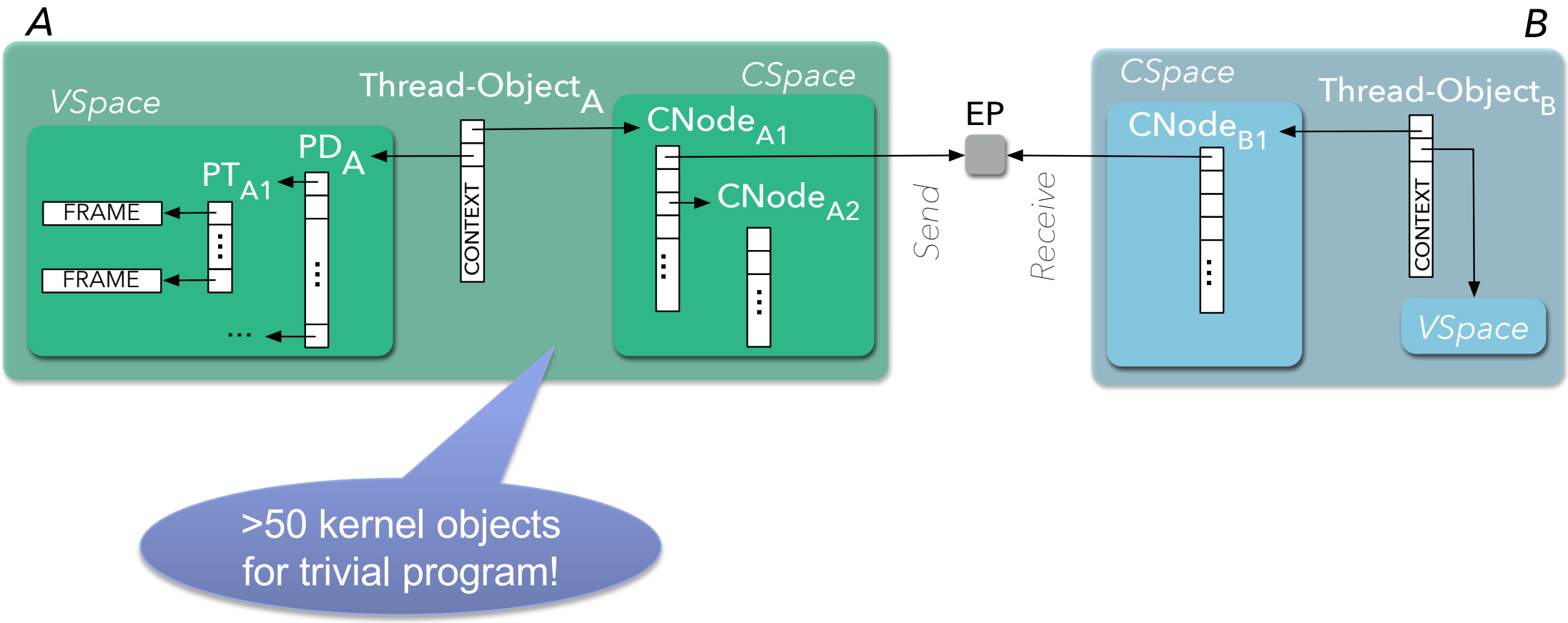
Incremental Cyber Retrofit

Original Mission Computer

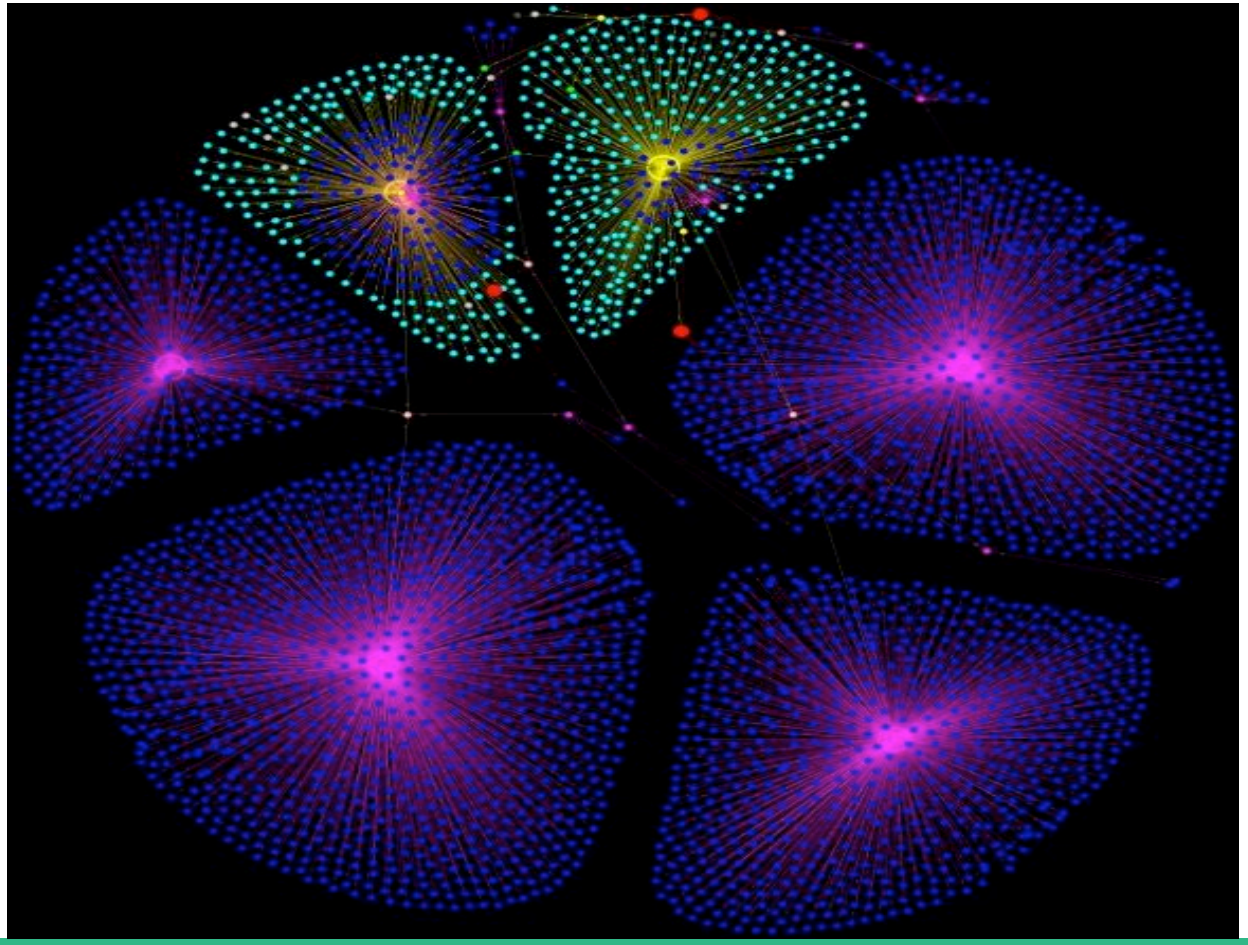
[Klein et al, CACM, Oct'18]



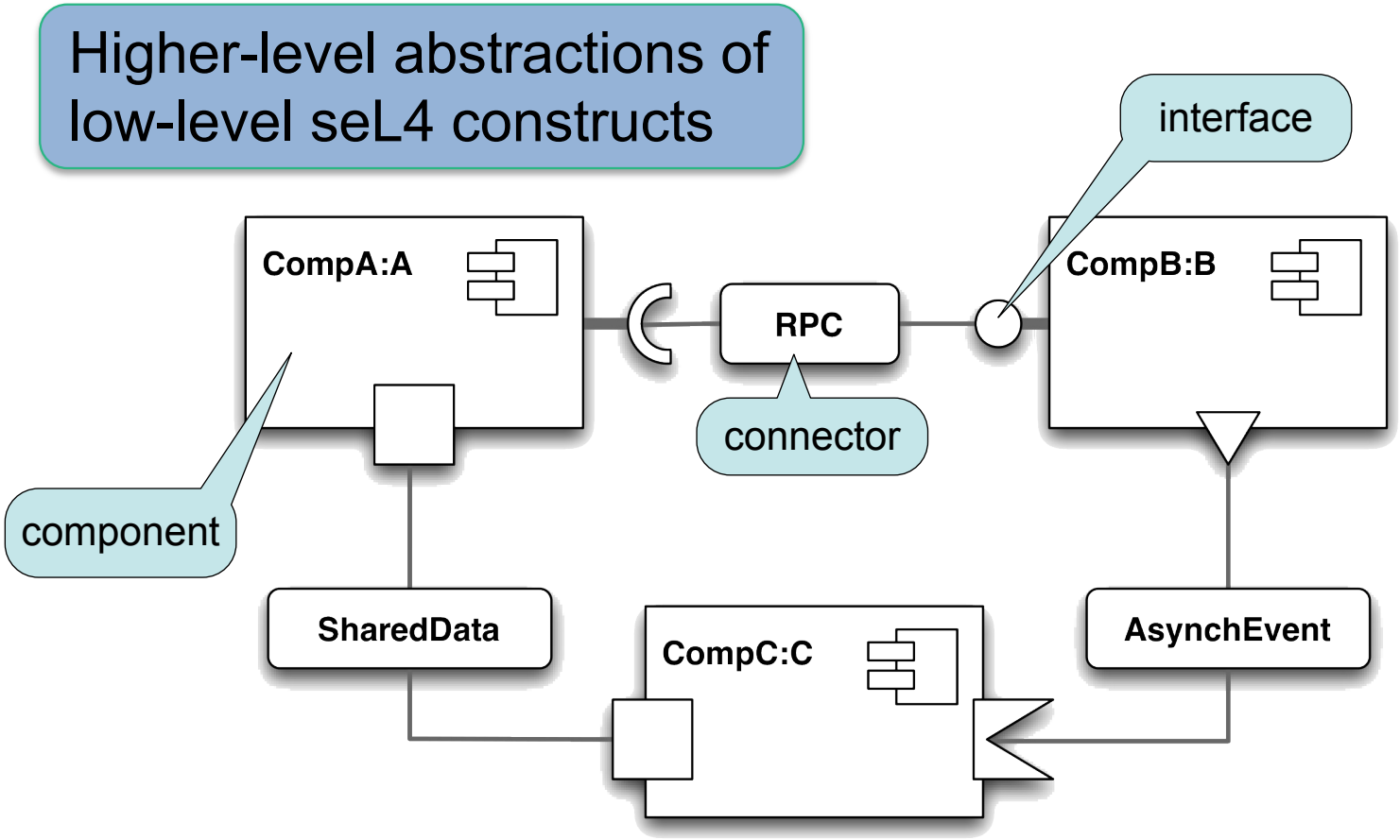
Issue: Primitives are Low-Level



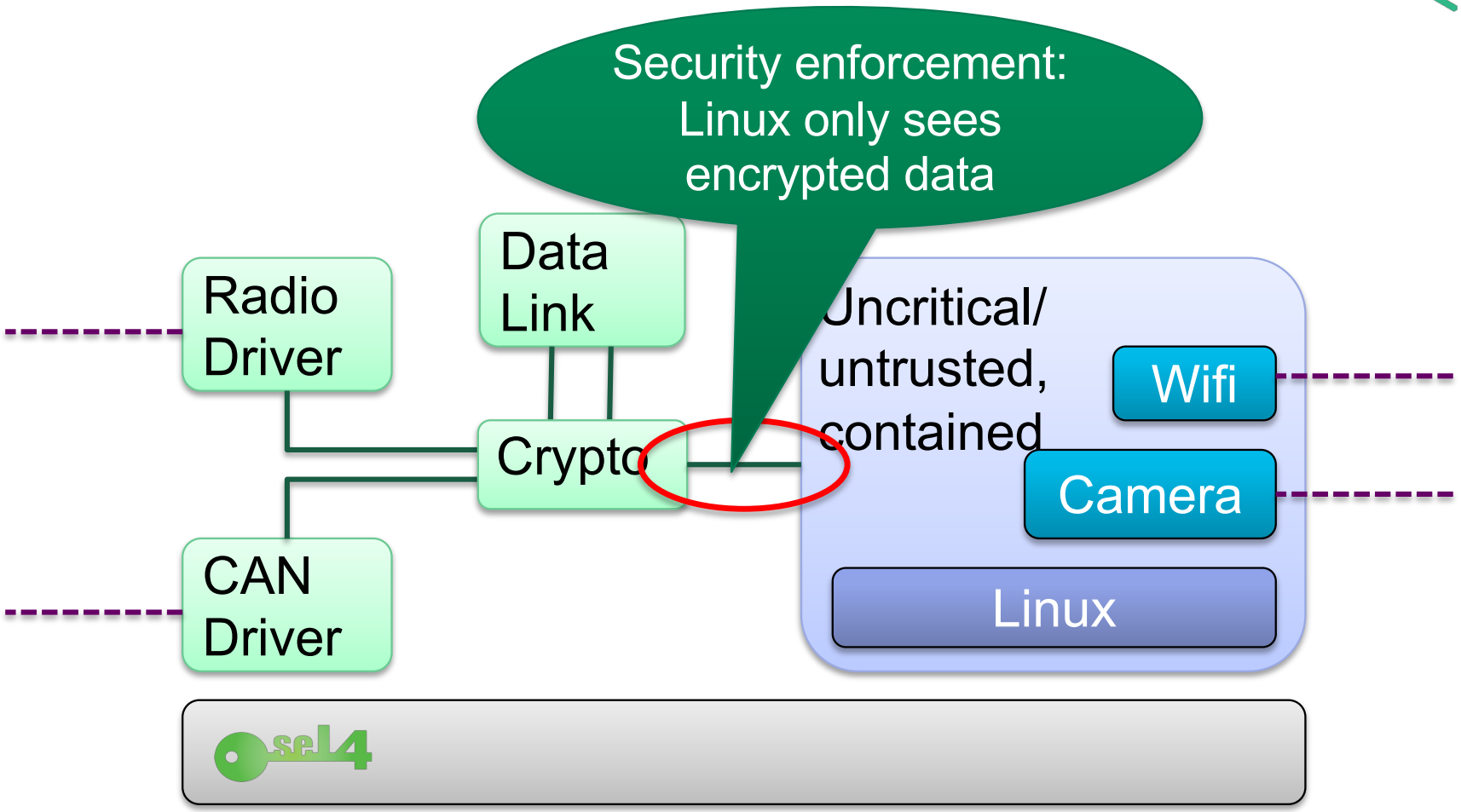
Non-Trivial But Simple System



Component Middleware: CAmkES

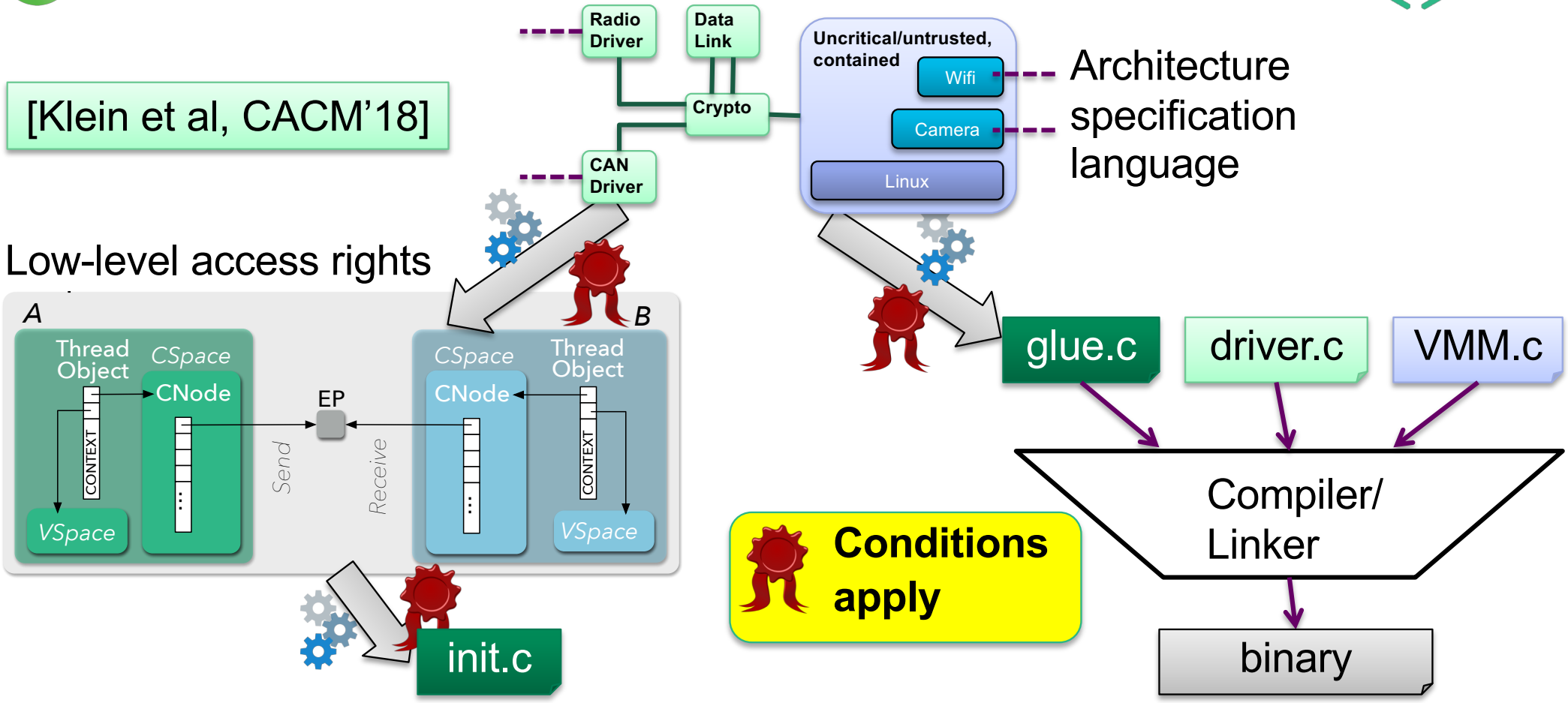


Simplified UAV Architecture

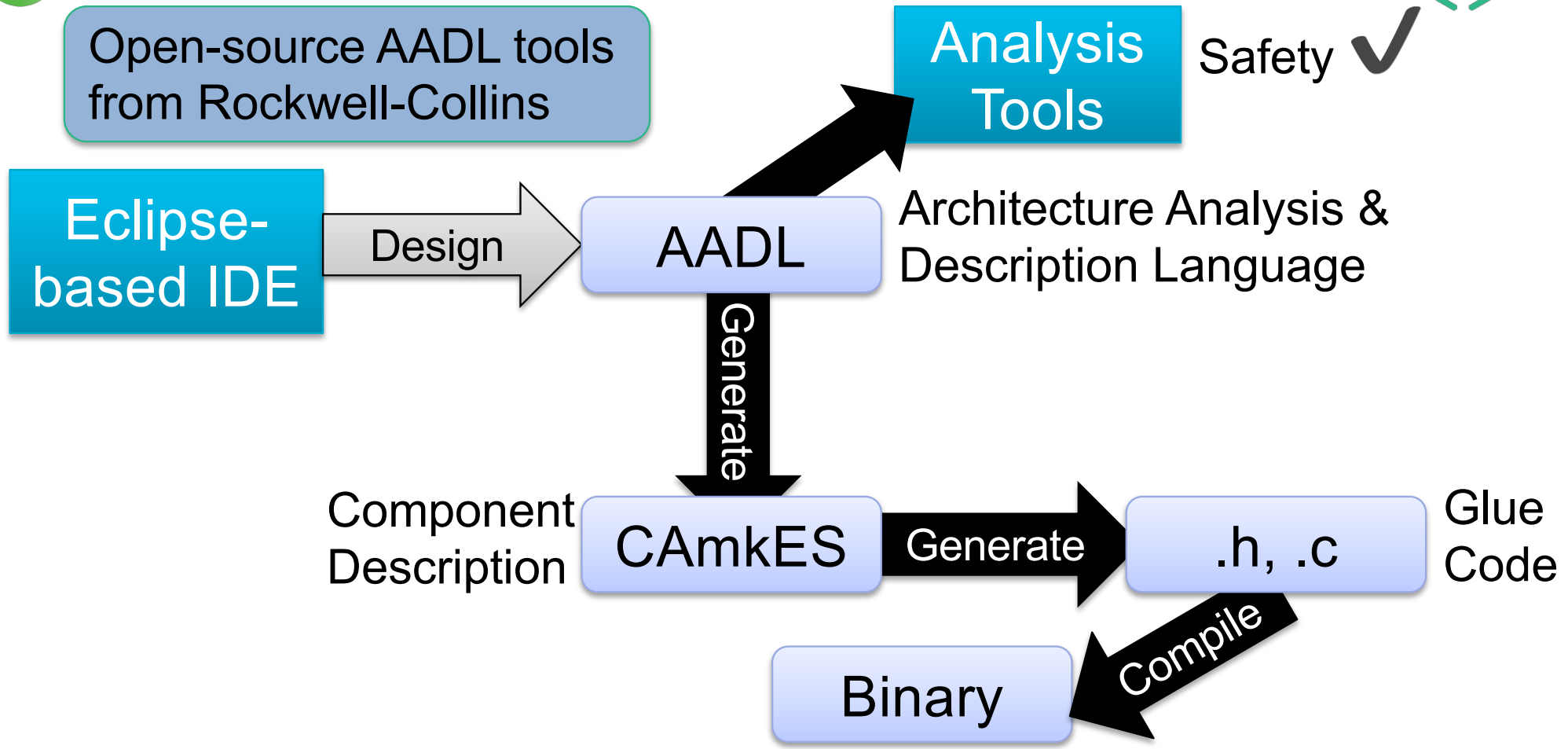


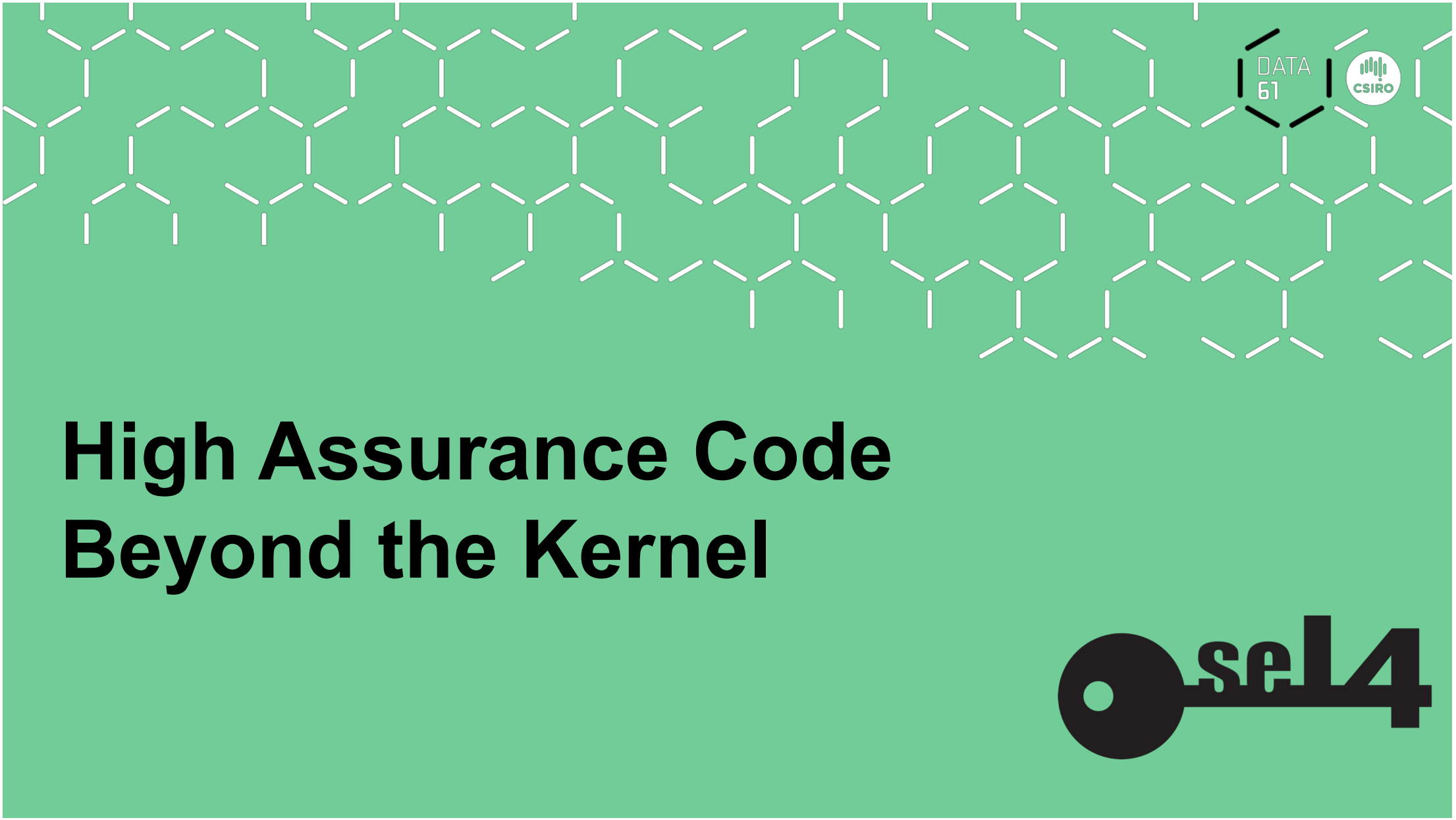
Enforcing the Architecture

[Klein et al, CACM'18]



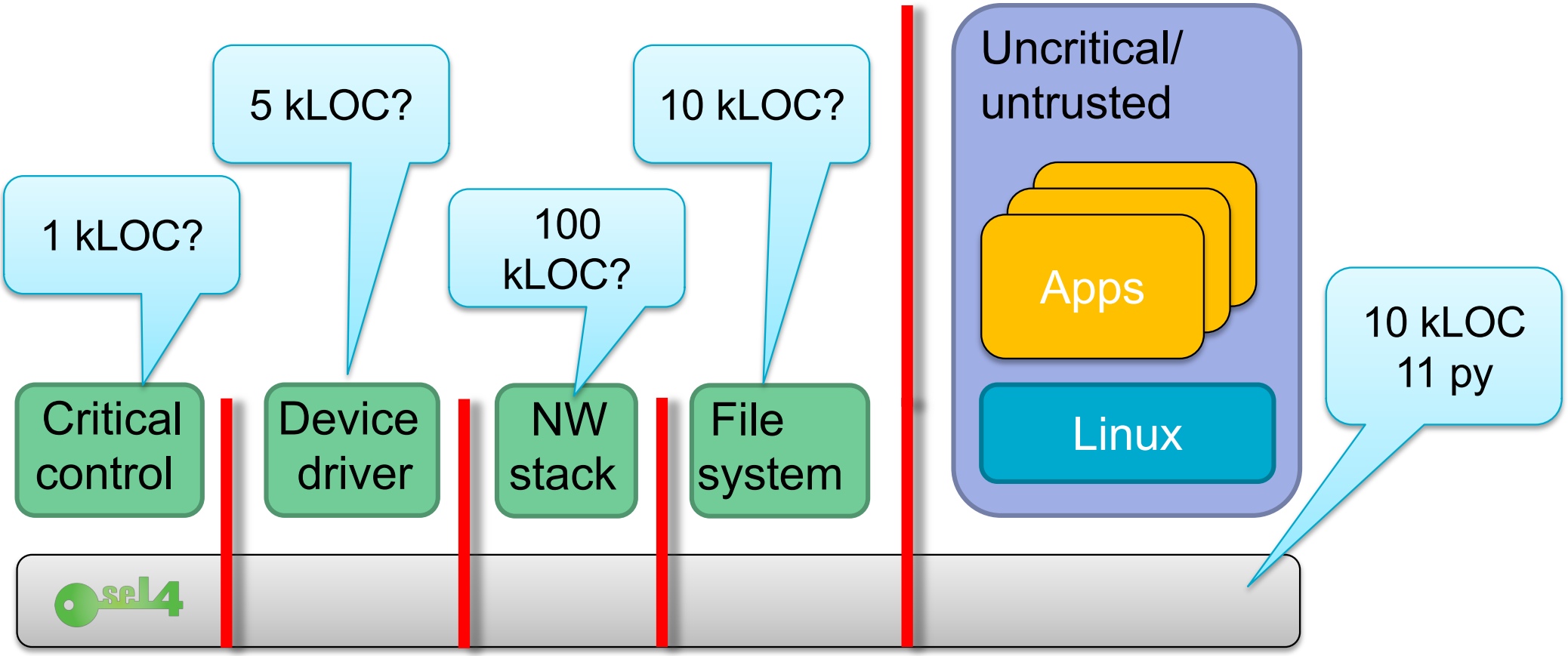
Architecture Analysis





High Assurance Code Beyond the Kernel

Beyond the Kernel

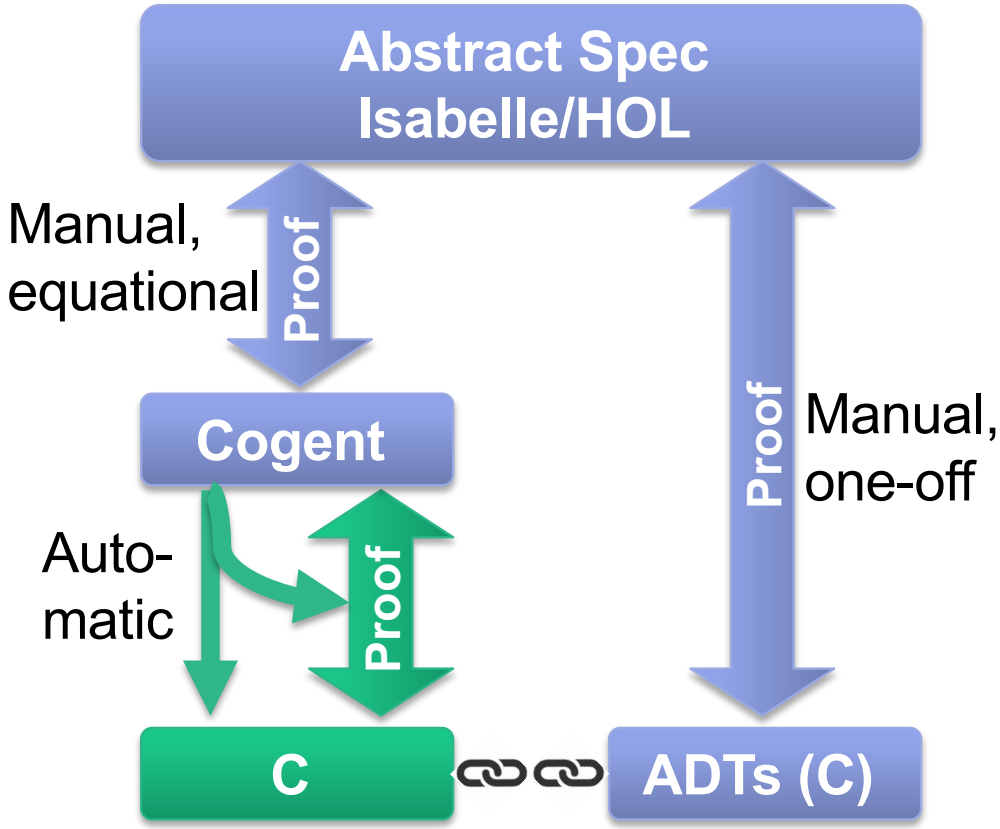


Cogent: Code & Proof Co-Generation

Aim: Reduce cost of verified systems code

- Restricted, purely functional *systems* language
- Type- and memory safe, not managed
- Turing incomplete
- Case-studies: BilbyFs, ext2, F2FS, VFAT

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

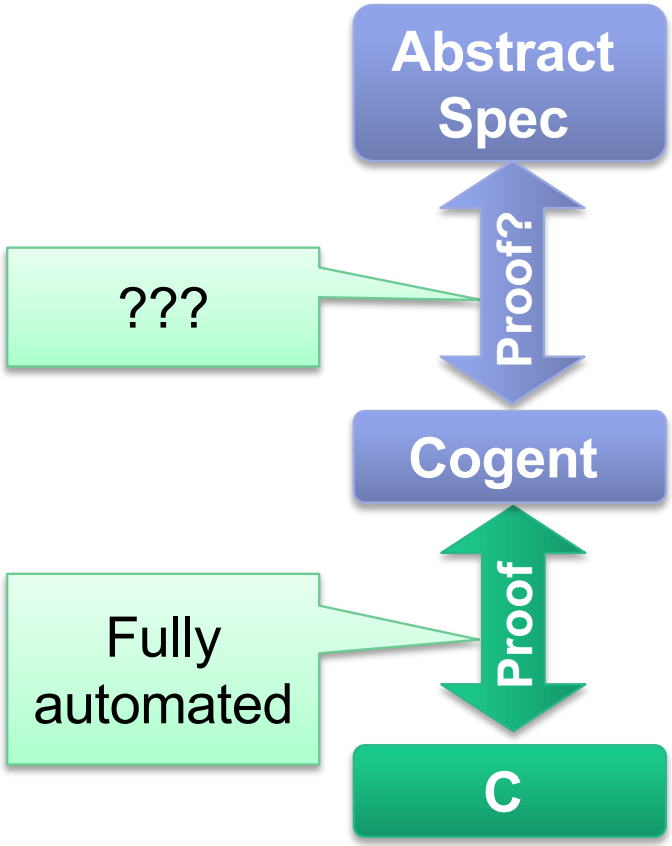


Manual Proof Effort

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

Dependable And Affordable?



Dependability-cost tradeoff:

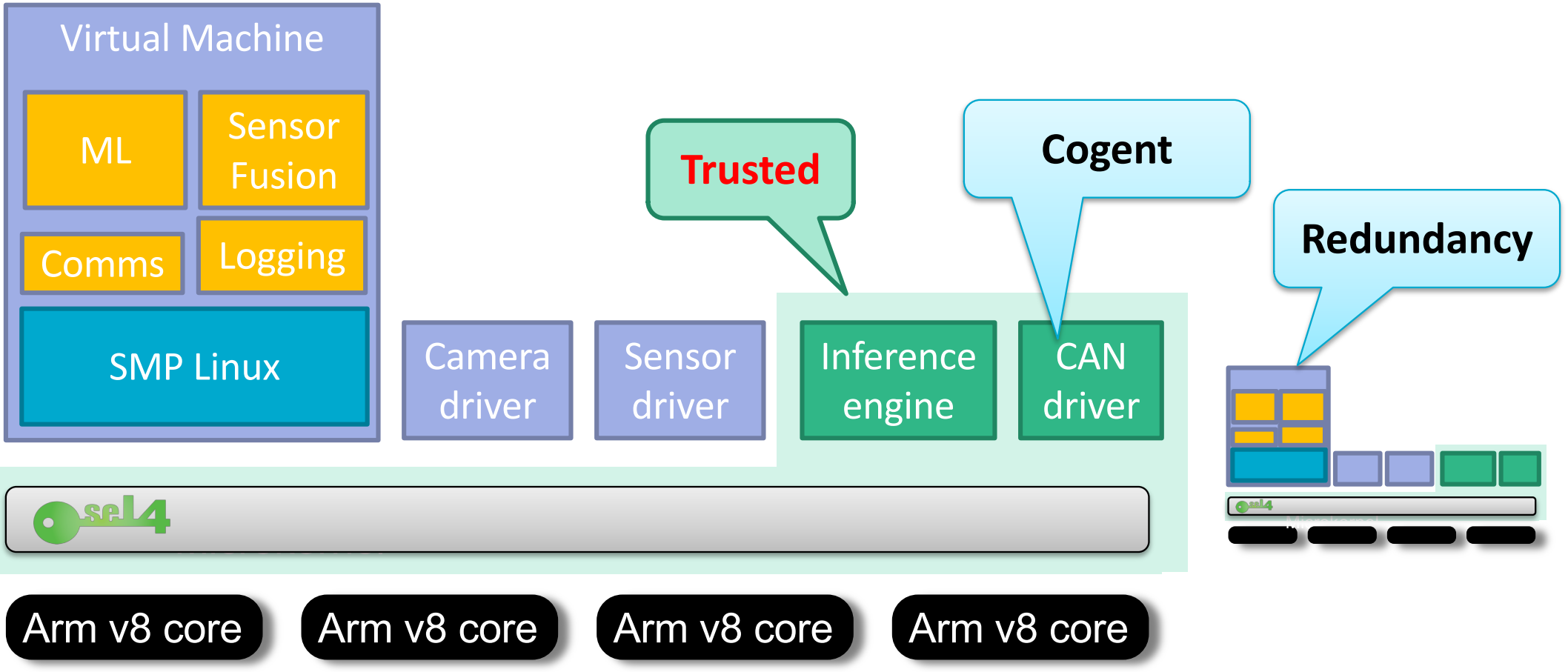
- Reduced faults through safe language
- Property-based testing (QuickCheck)
- Model checking
- Full functional correctness proof

Spec reuse!

Work in progress:

- Language expressiveness
- Reduce boiler-plate code
- Network stacks
- Device drivers

Application to Autonomous Cars



Trustworthy Systems Team


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

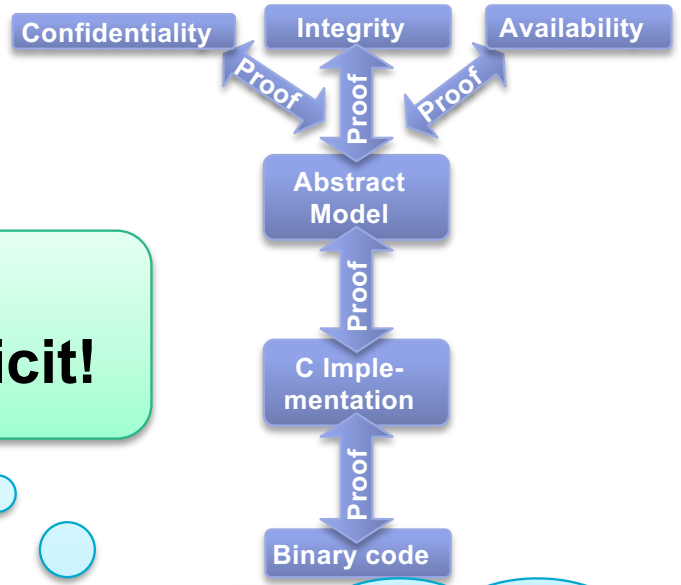
Verification Guarantees

Verification rules out unspecified behaviour:

- Buffer/stack overflow
- Null-pointer dereference
- Code injection
- Use after free
- Memory leaks
- Kernel crash
- Privilege escalation
- Covert *storage* channels, …

… as long as the assumptions are satisfied!

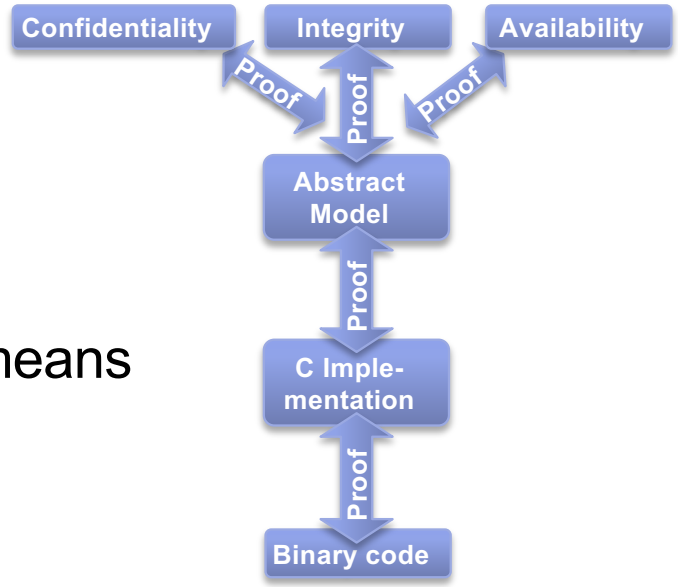
Verification forces you to **make assumptions explicit!**



Reason many bugs are found just from writing the spec!

Verification Assumptions

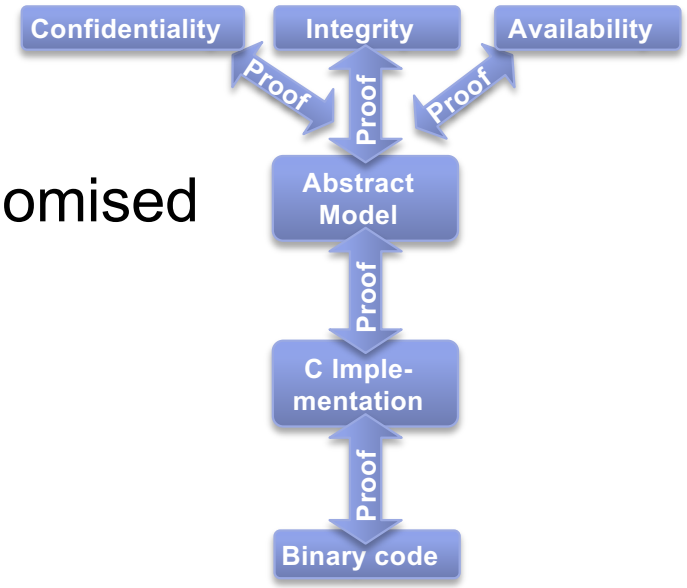
1. Hardware behaves as expected
   - Formalised hardware-software contract (ISA)
   - Hardware implementation free of bugs, Trojans, …
2. Spec matches expectations
   - Can only prove “security” if specify what “security” means
   - Spec may not be what we think it is
3. Proof checker is correct
   - Isabel/HOL checking core that validates proofs against logic



With binary verification do **not** need to trust C compiler!

Present Verification Limitations

- Not verified boot code
  - **Assume** it leaves kernel in safe state
- Caches/MMU presently modeled at high level / axiomised
  - This is in progress of being fixed
- Not proved any temporal properties
  - Presently not proved scheduler observes priorities, properties needed for RT
  - Worst-case execution-time analysis applies only to dated ARM11/A8 cores
  - No proofs about timing channels



Verification Matrix

Feature	Core spec to C	C to binary	Security enforcem.	Mixed-criticality	Virtual machines	Multicore
Arm 32	done	done	done	in progr.	done	in progr.
Arm 64	unfunded	in progr.	unfunded	unfunded	unfunded	???
x64	done	no plans	no plans	easy?	no plans	???
R-V 64	in progr.	in progr.	unfunded	in progr.	unfunded	???

- **Security**: CIA enforcement proofs
- **Mixed criticality**: advanced real-time support with temporal isolation;
  This will replace the mainline kernel once verified
- **Virtual machines**: verified use of hardware virtualisation support