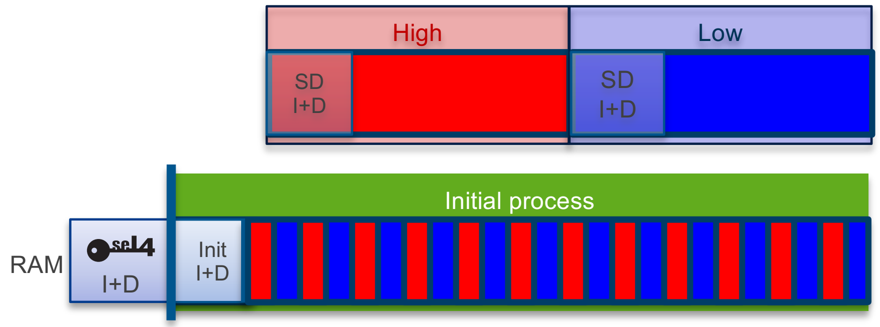
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2022 T2 Week 10 Part 2

seL4 in the Real World & seL4 Research at TS@UNSW

@GernotHeiser



Today's Lecture

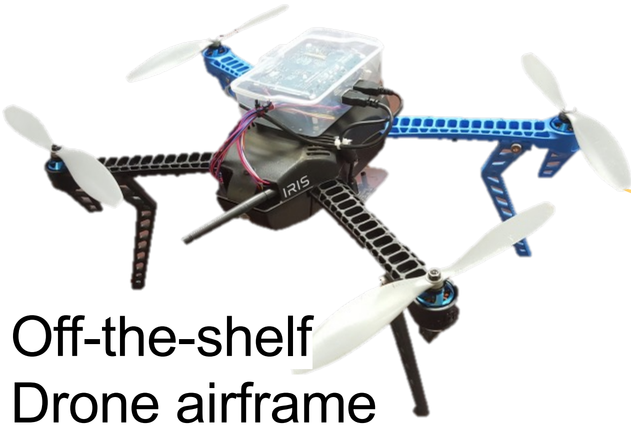
- seL4 in the real world
  - HACMS & incremental cyber-retrofit
  - Usability: CAmkES & seL4 Core Platform
- seL4-related research at UNSW Trustworthy Systems
  - sDDF: High-performance driver framework
  - Pancake: Verifying device drivers
  - Verifying the seL4CP
  - Secure multi-server OS
  - Time protection: Verified timing-channel prevention

seL4 in the Real World

DARPA HACMS

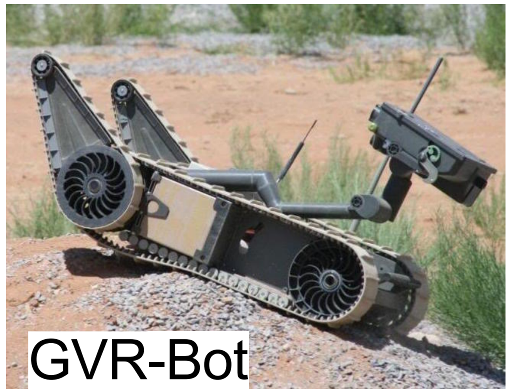
Unmanned Little Bird (ULB)

Retrofit existing system!

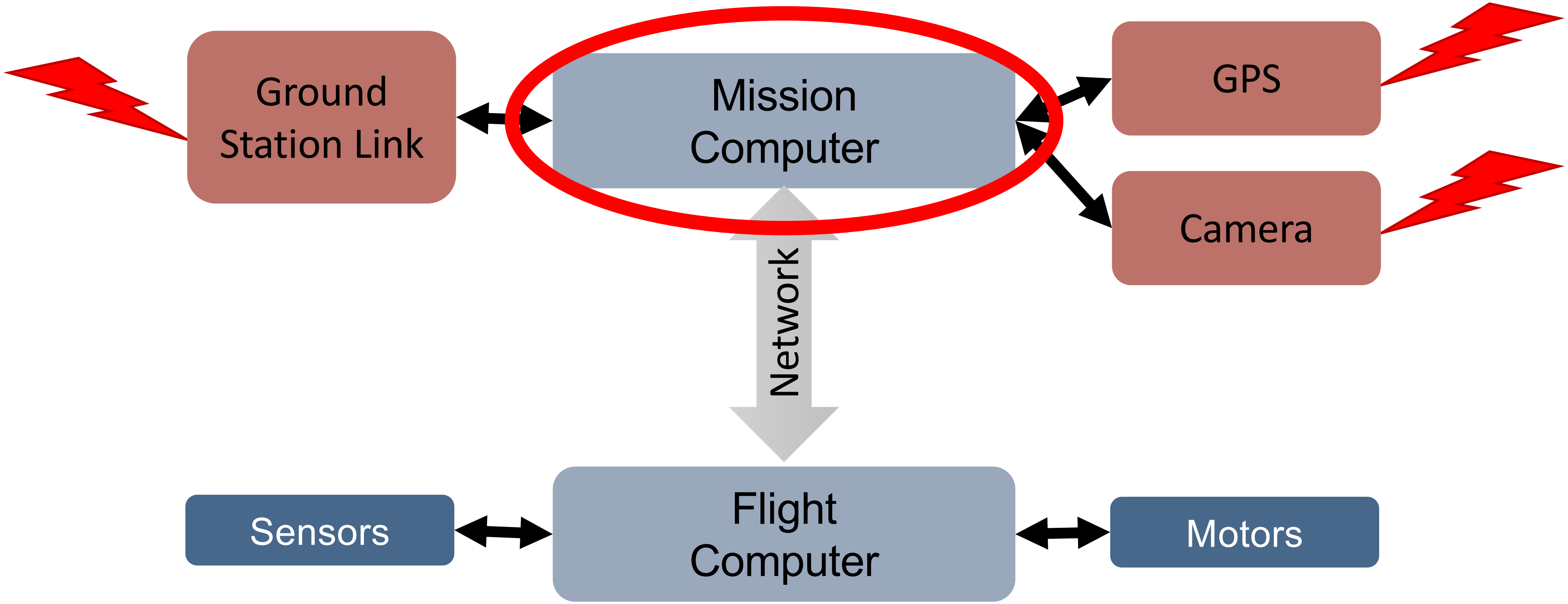


Off-the-shelf
Drone airframe

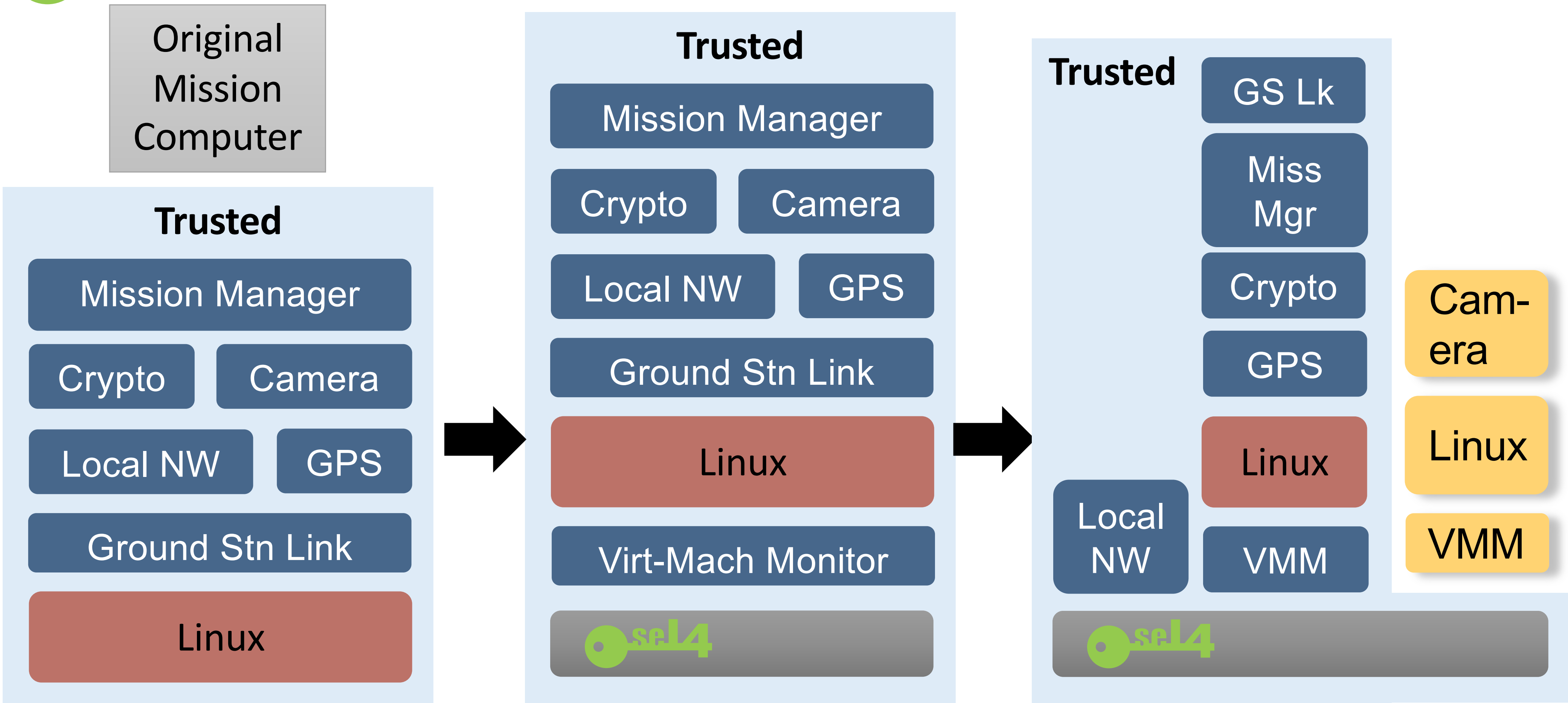
Develop technology



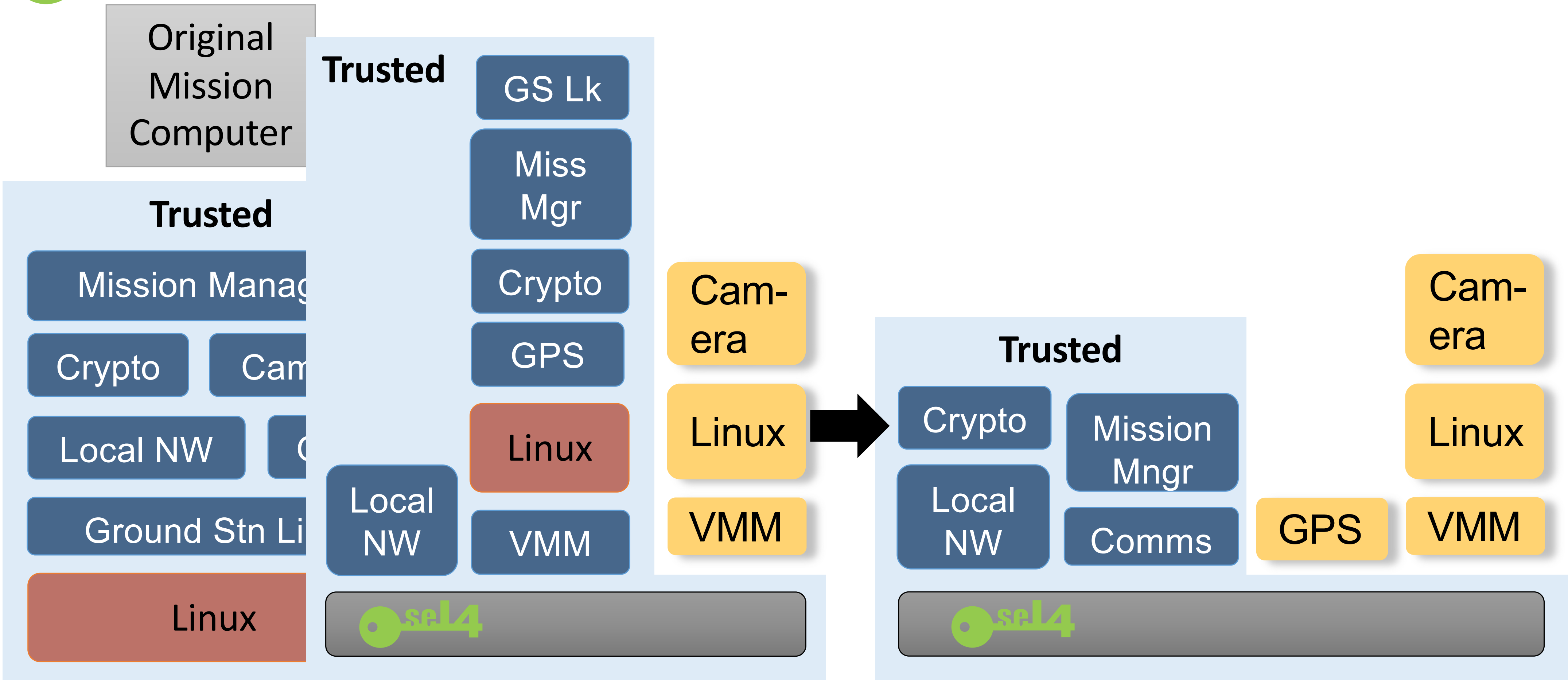
ULB Architecture



Incremental Cyber Retrofit

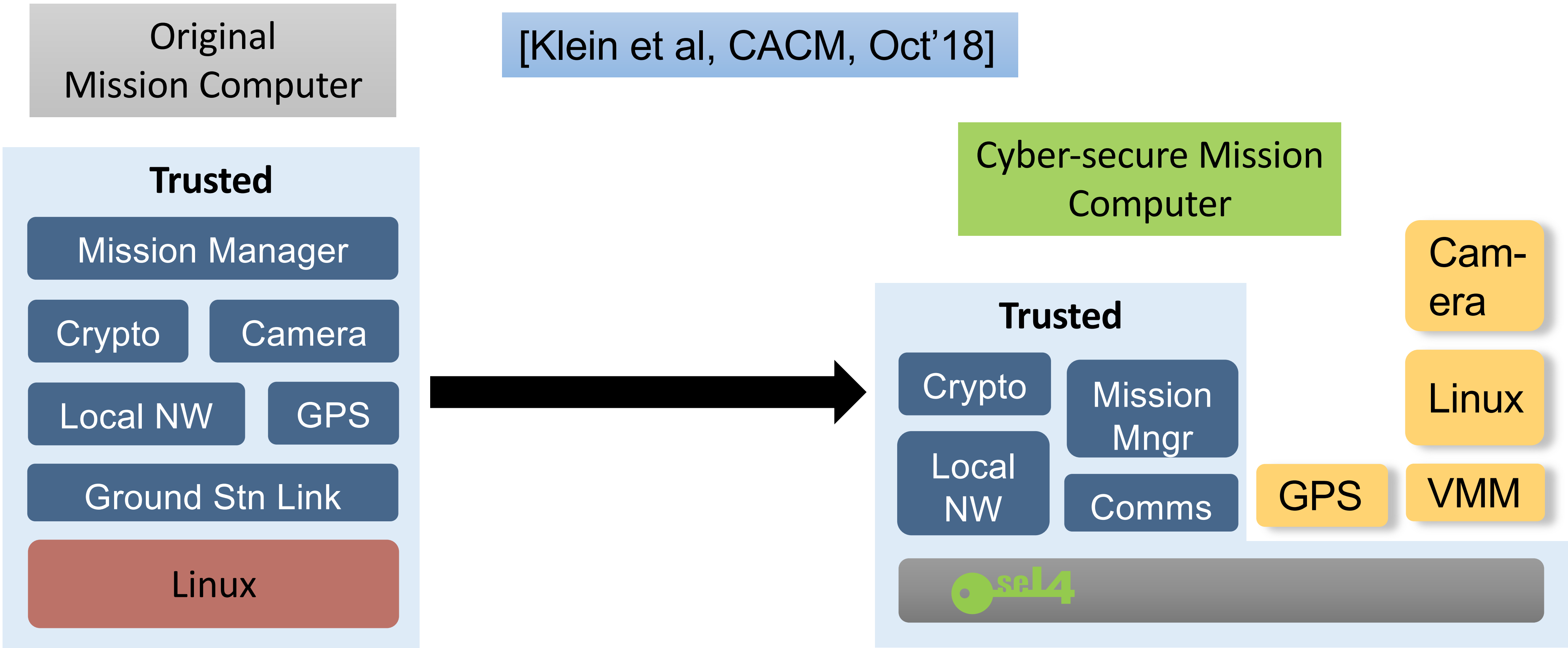


Incremental Cyber Retrofit



seL4 Incremental Cyber Retrofit

[Klein et al, CACM, Oct'18]



World's Most Secure Drone

Tweet

DARPA
@DARPA

We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!

HACMS Outcomes

- Demonstrated real-world suitability of seL4 and formal methods
  - Reversal of bad vibes from over-promising and under-delivering
  - Major re-think in US defence
- Dis-proved “security must be designed in from the start”
- Led to follow-on funding for seL4 and deployment in the field

The seL4 Foundation

Premium Members

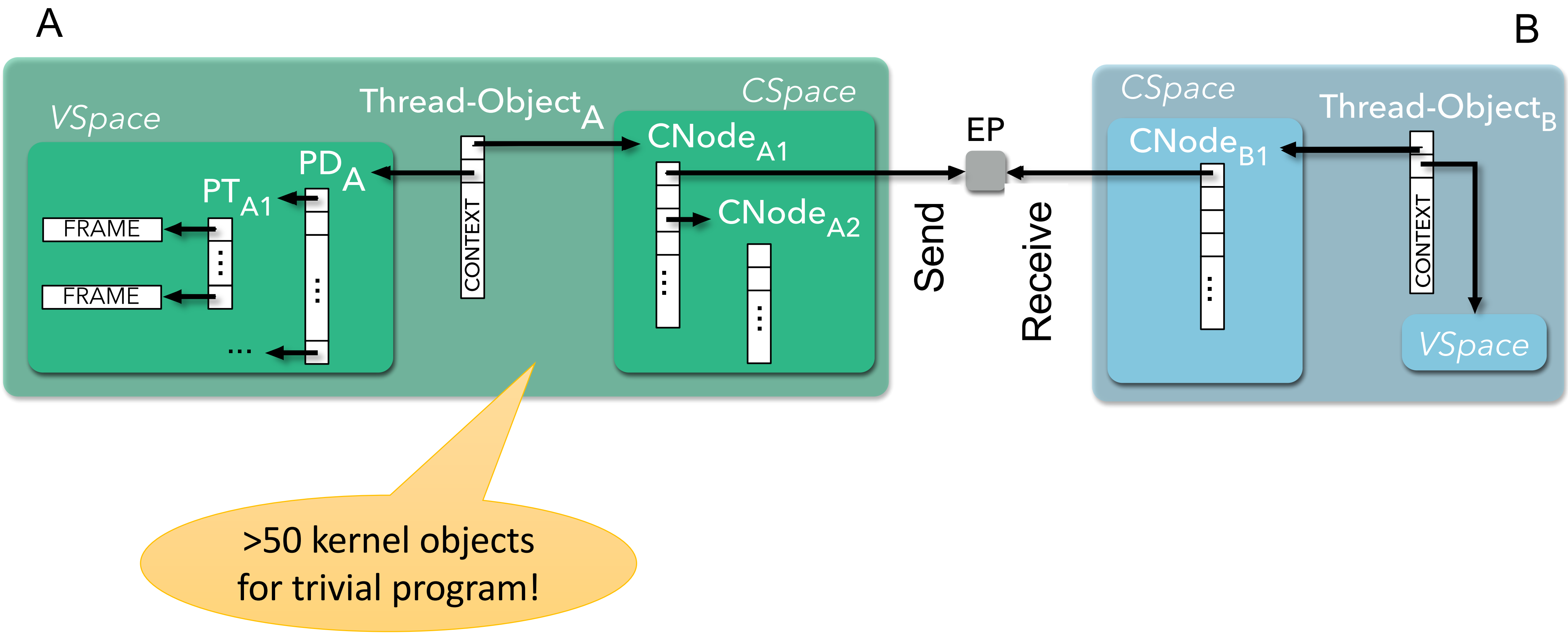
General Members

Associate Members

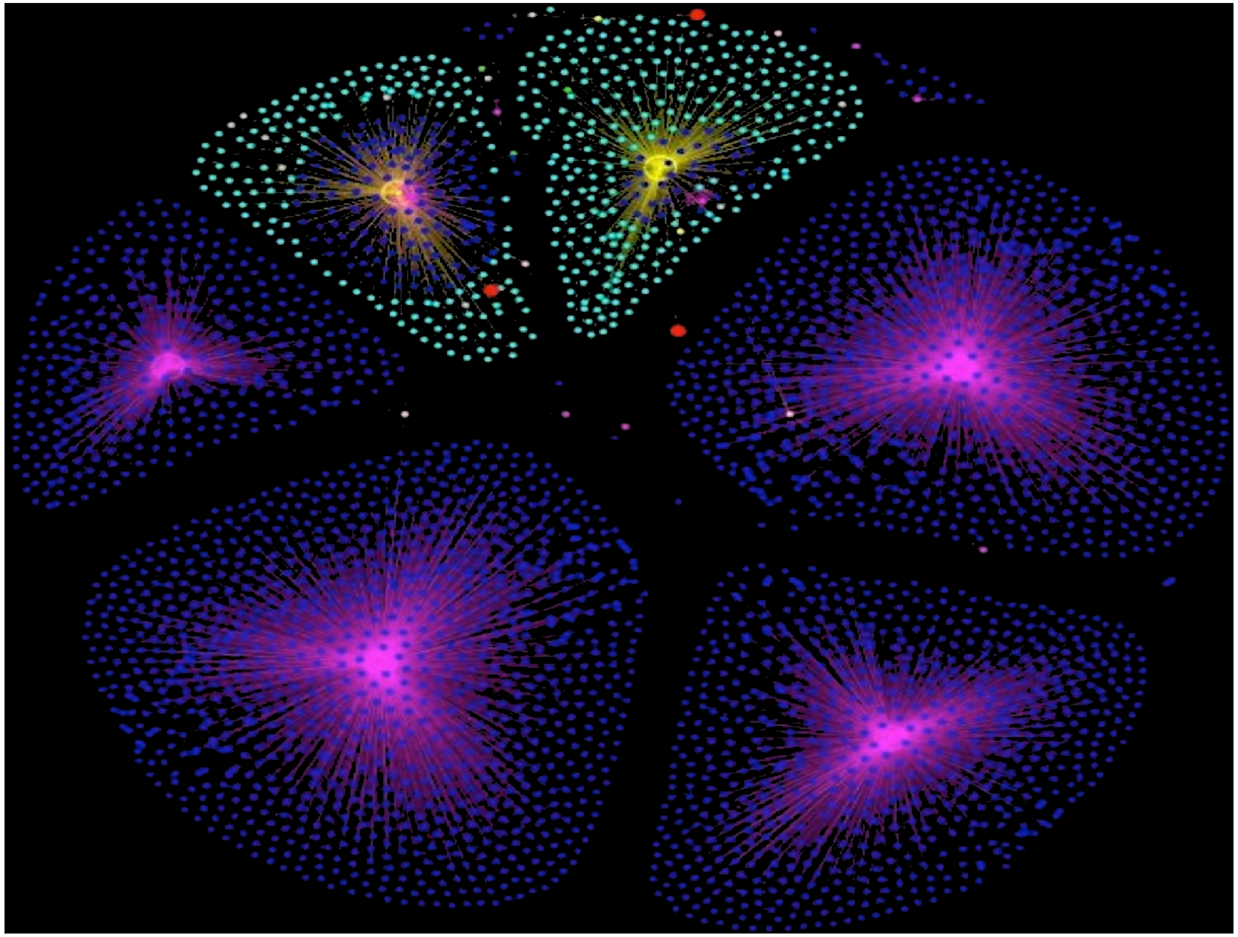
Usability

CAmkES and the seL4 Core Platform

Issue: seL4 Objects are Low-Level

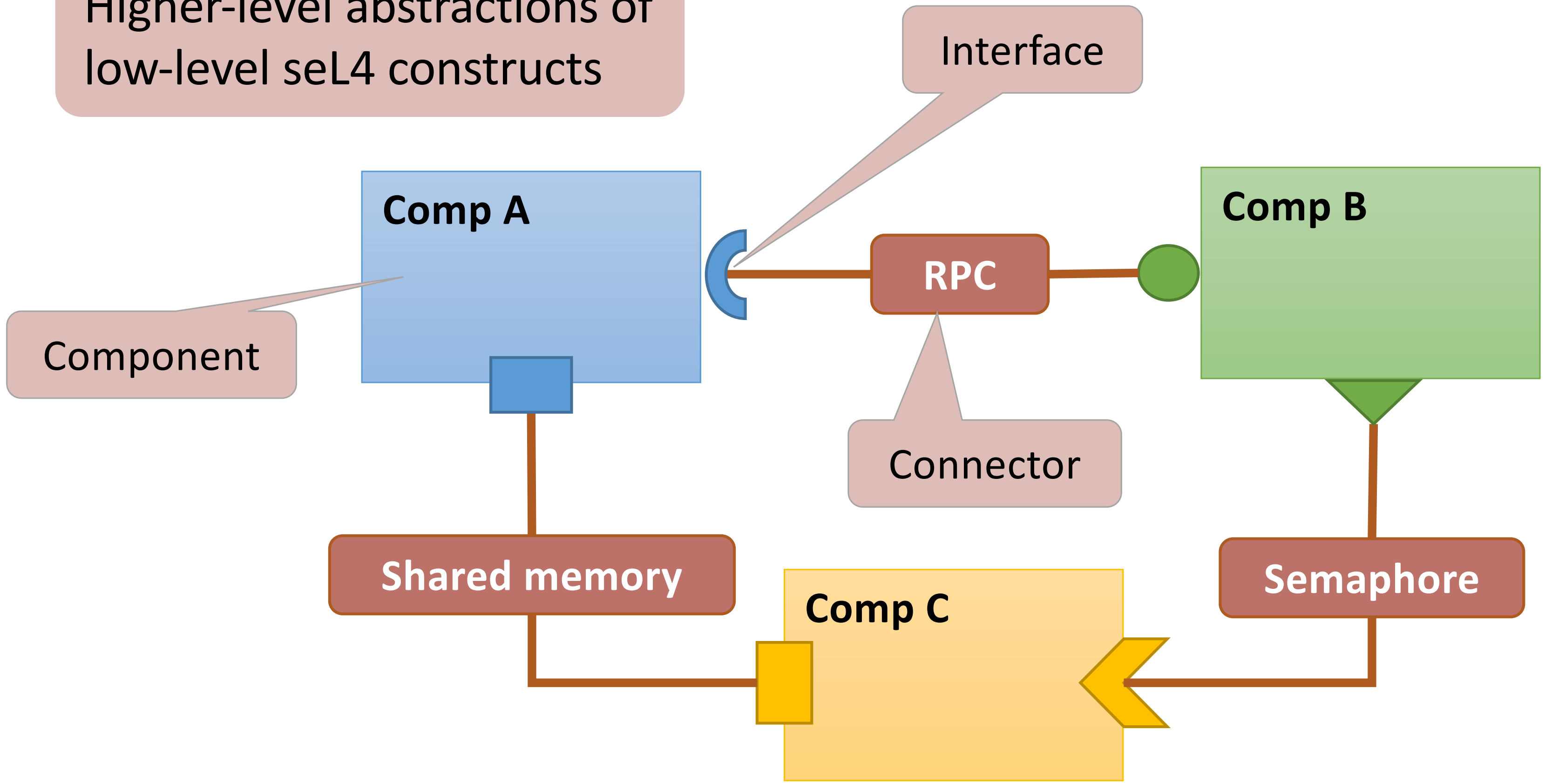


Simple But Non-Trivial System



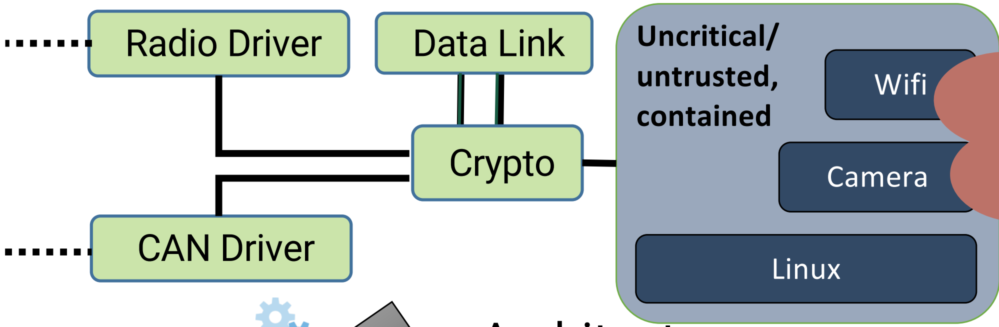
Recommended Framework: CAmkES

Higher-level abstractions of low-level seL4 constructs



seL4 CAmkES Framework

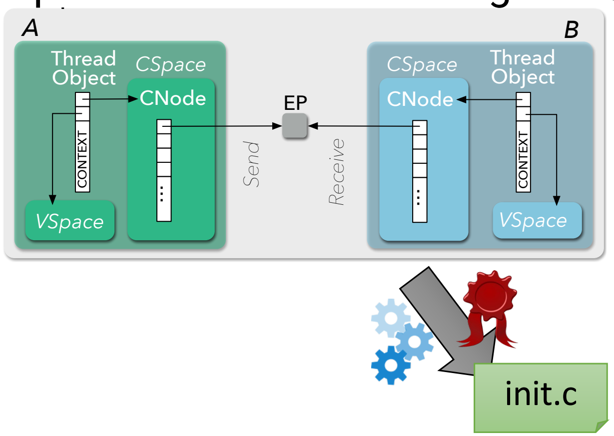
Conditions apply



- Good for assurance
- Bad for usability & functionality

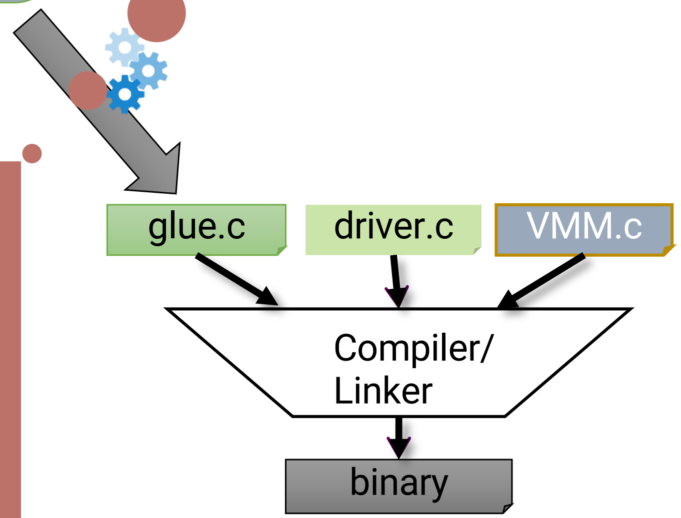
Architecture specification

CapDL: Low-level access rights



However:

- Forces use of kernel build system
- Fully static & hard to extend
- Significant overheads



UNSW SYDNEY

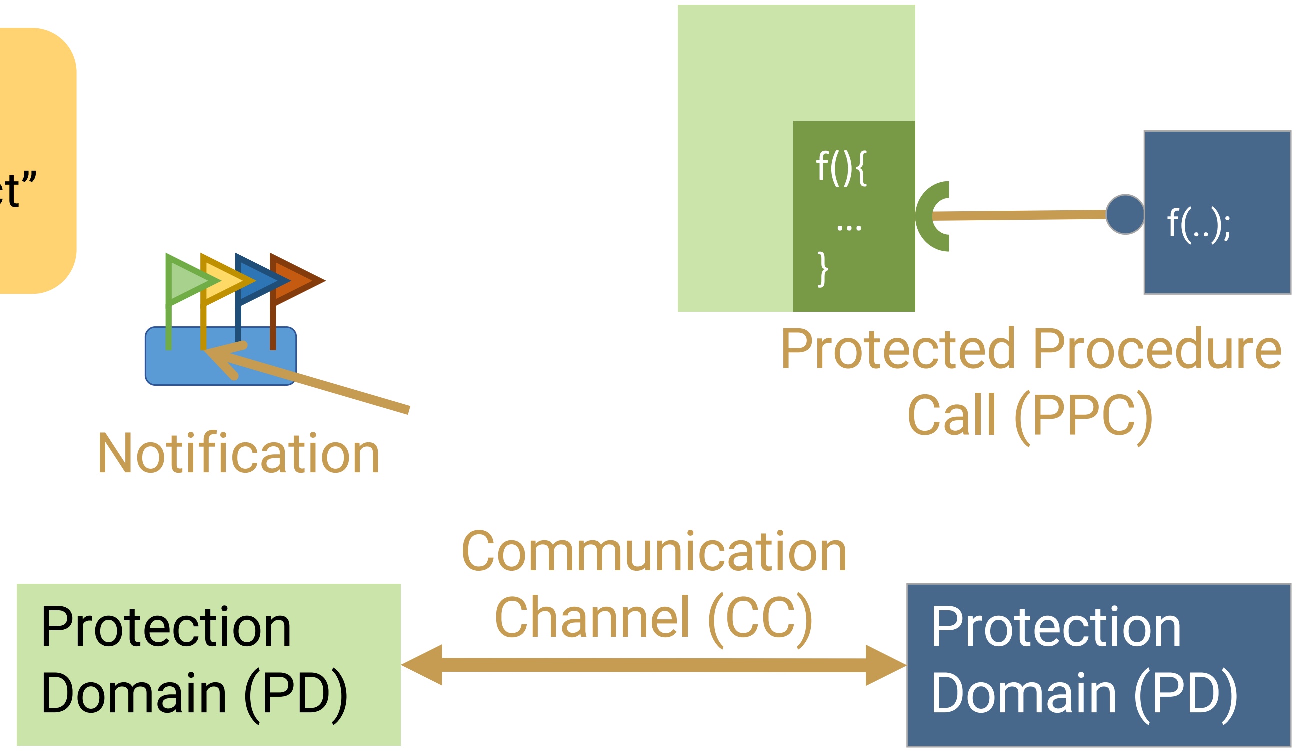
New Framework: seL4 Core Platform

Small OS for IoT, cyber-physical and other embedded use

- Leverage seL4-enforced isolation for strong security/safety
- Retain seL4's superior performance
- "Correct" use of seL4 mechanisms by default
- Ease development and deployment
  - SDK, integrate with build system of your choice
- Retain near-minimal trusted computing base (TCB)
- Be amenable to formal verification of the TCB

seL4CP Abstractions

- Thin wrapper of seL4 abstractions
- Encourage “correct” use of seL4

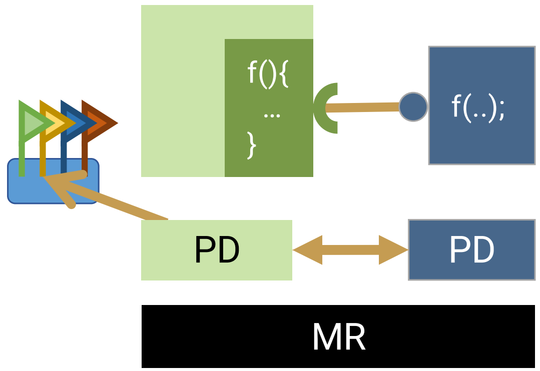


Memory Region (MR)

seL4CP Status

- Developed by Breakaway
- Used in products (Laot, AArch64-based)
- Virtualisation support in progress
- Platform and ISA ports in progress (x64, RV64)
- Dynamic features prototype:
  - fault handlers
  - start/stop protection domains
  - re-initialise protection domains
  - empty protection domains (for late app loading)

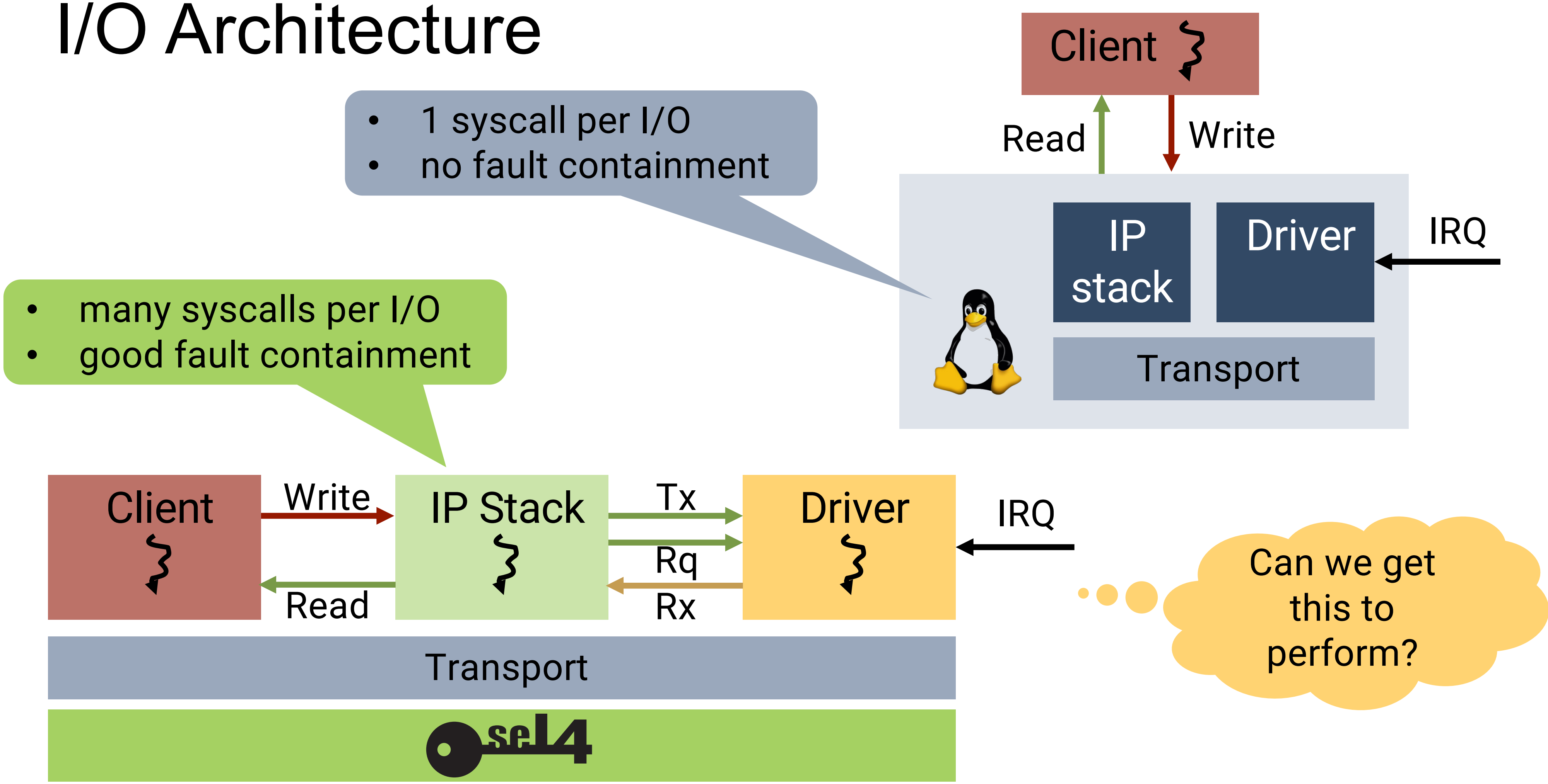
Ivan Velickovic



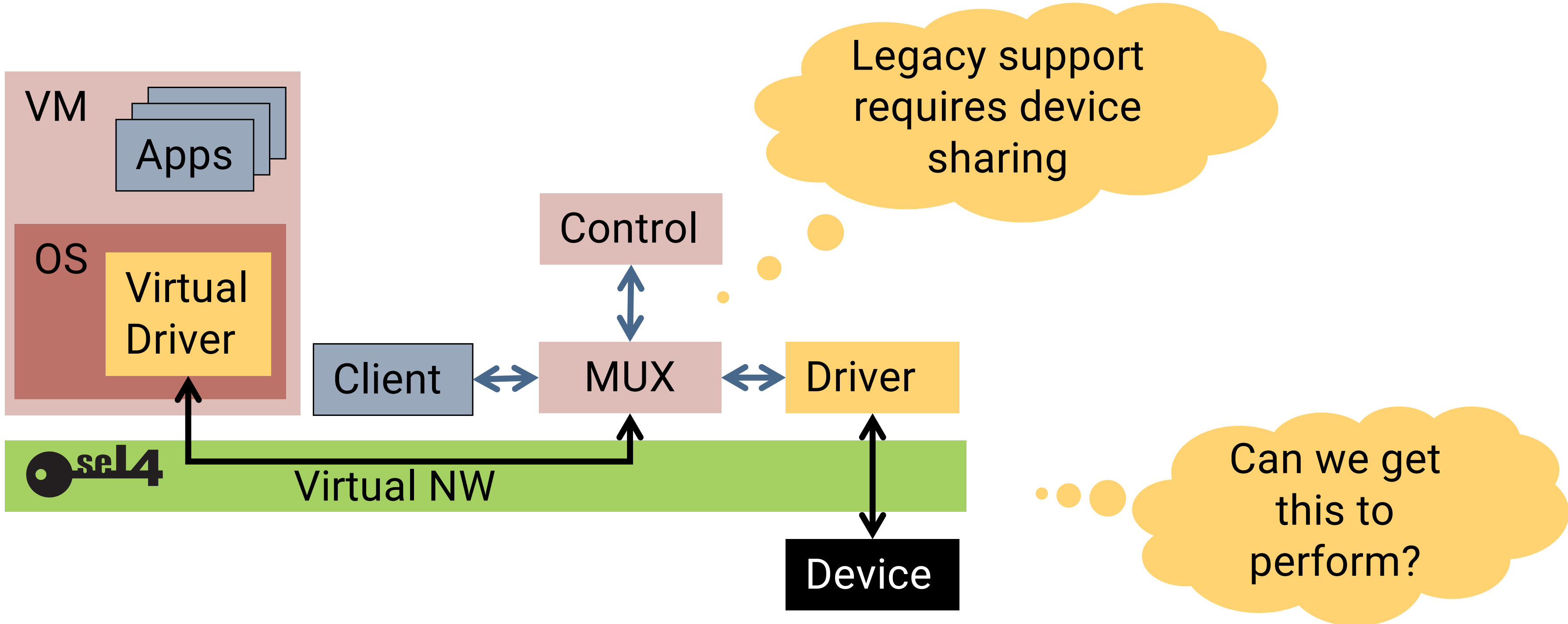
seL4-Related Research in TS

High-Performance I/O and I/O Virtualisation

I/O Architecture



Device Sharing (aka I/O Virtualisation)

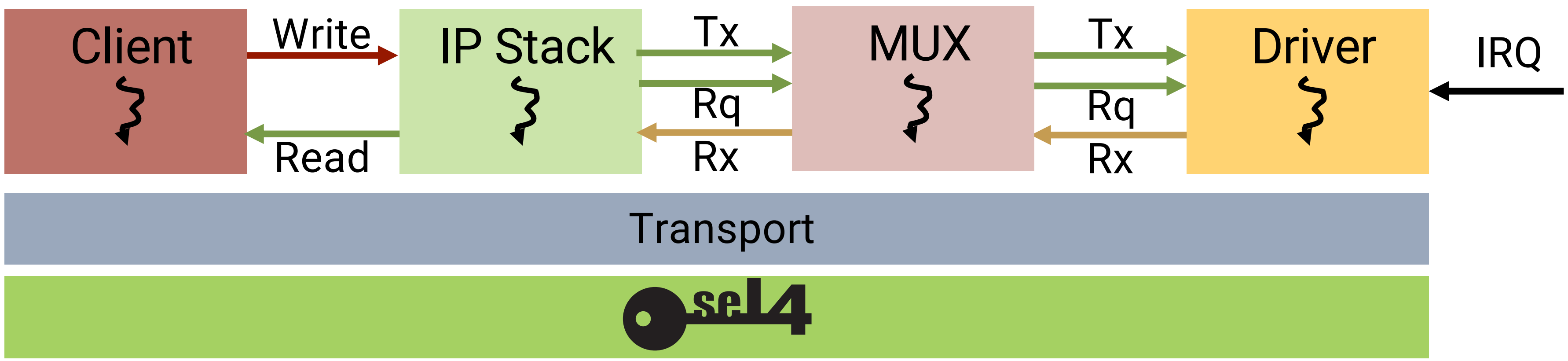


Advanced I/O Architecture

Challenge:
- Performance

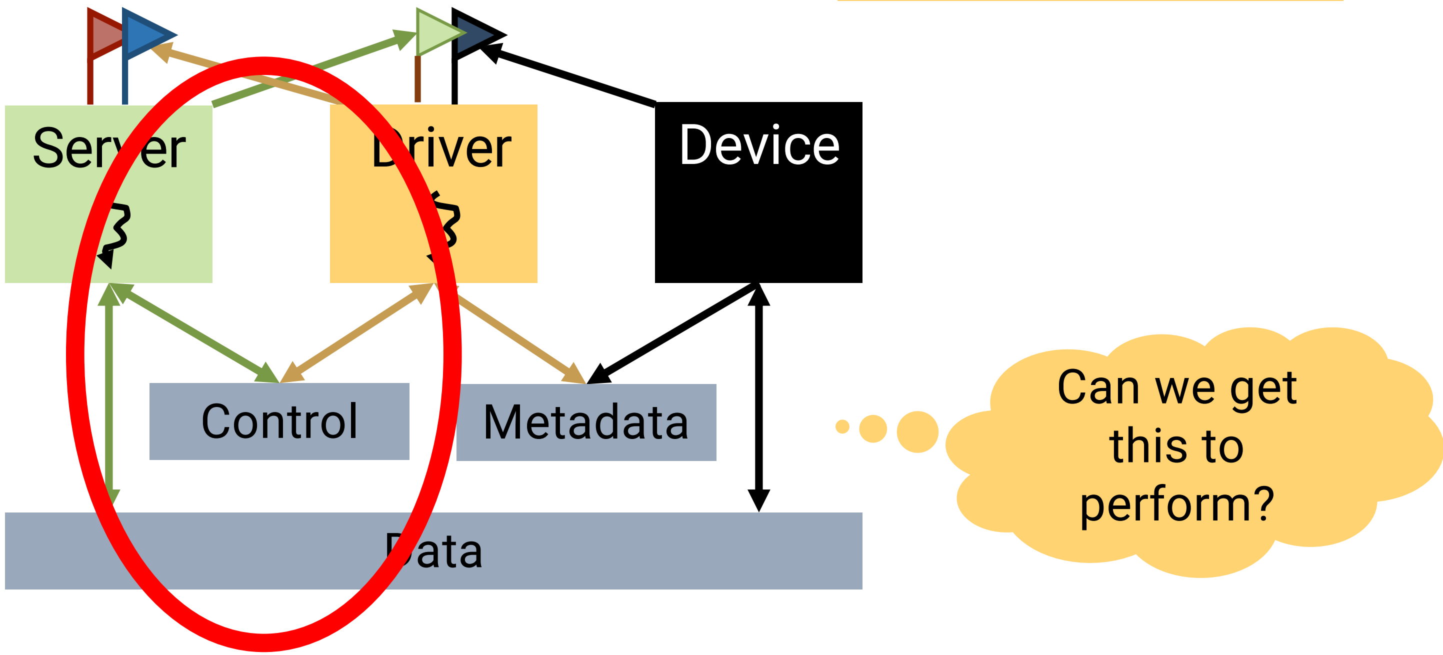
Opportunities:
- Re-think design
- Simplify driver model
- Simplify IP stack
- Reduce (avoid?) locking

Enable verification?



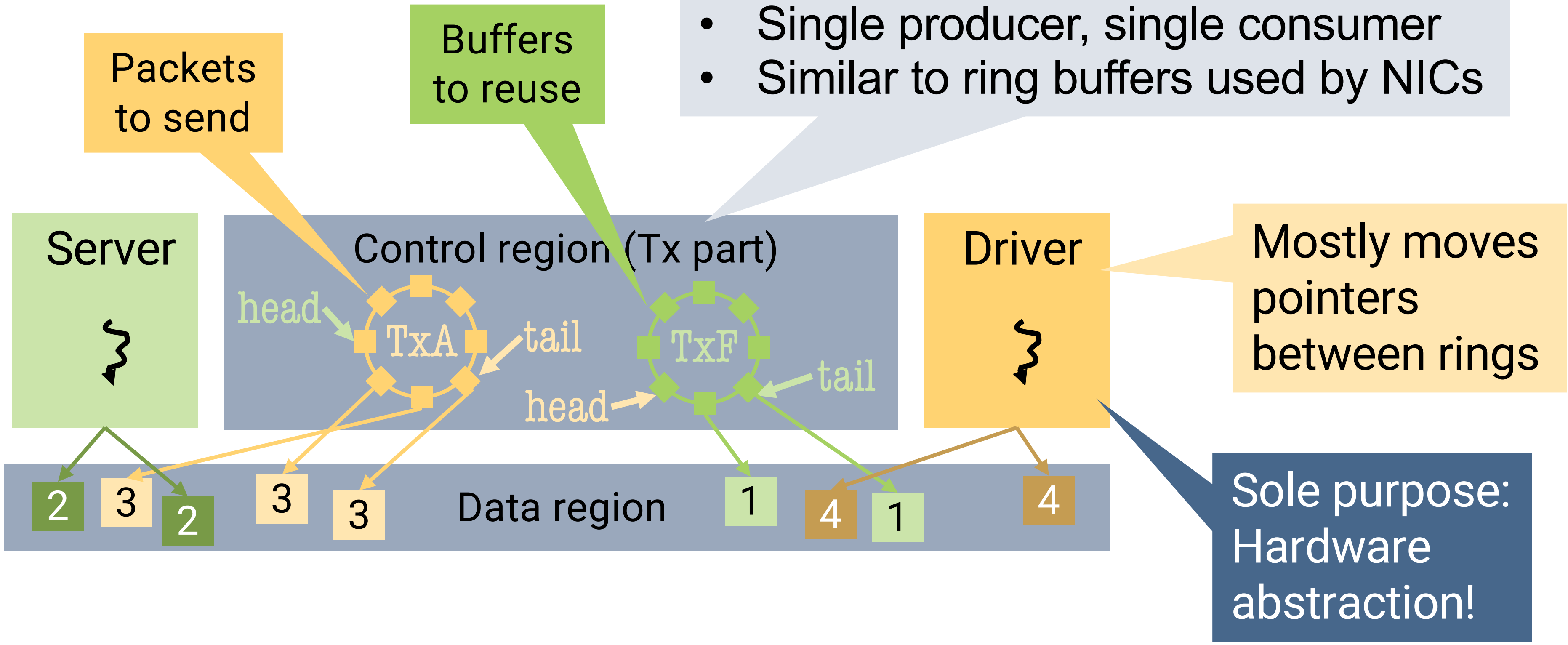
Driver Model

Driver model:
- Single-threaded
- Event-driven
- Simple!



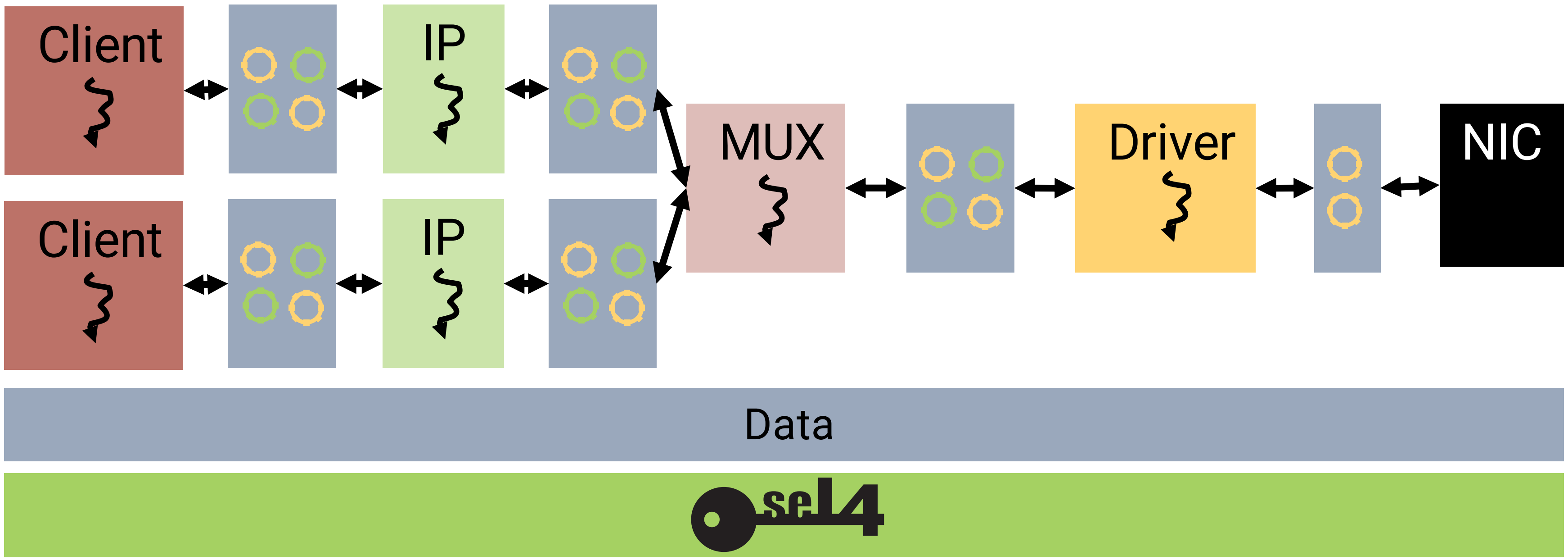
Transport Layer

- Lock-free bounded queues
- Single producer, single consumer
- Similar to ring buffers used by NICs

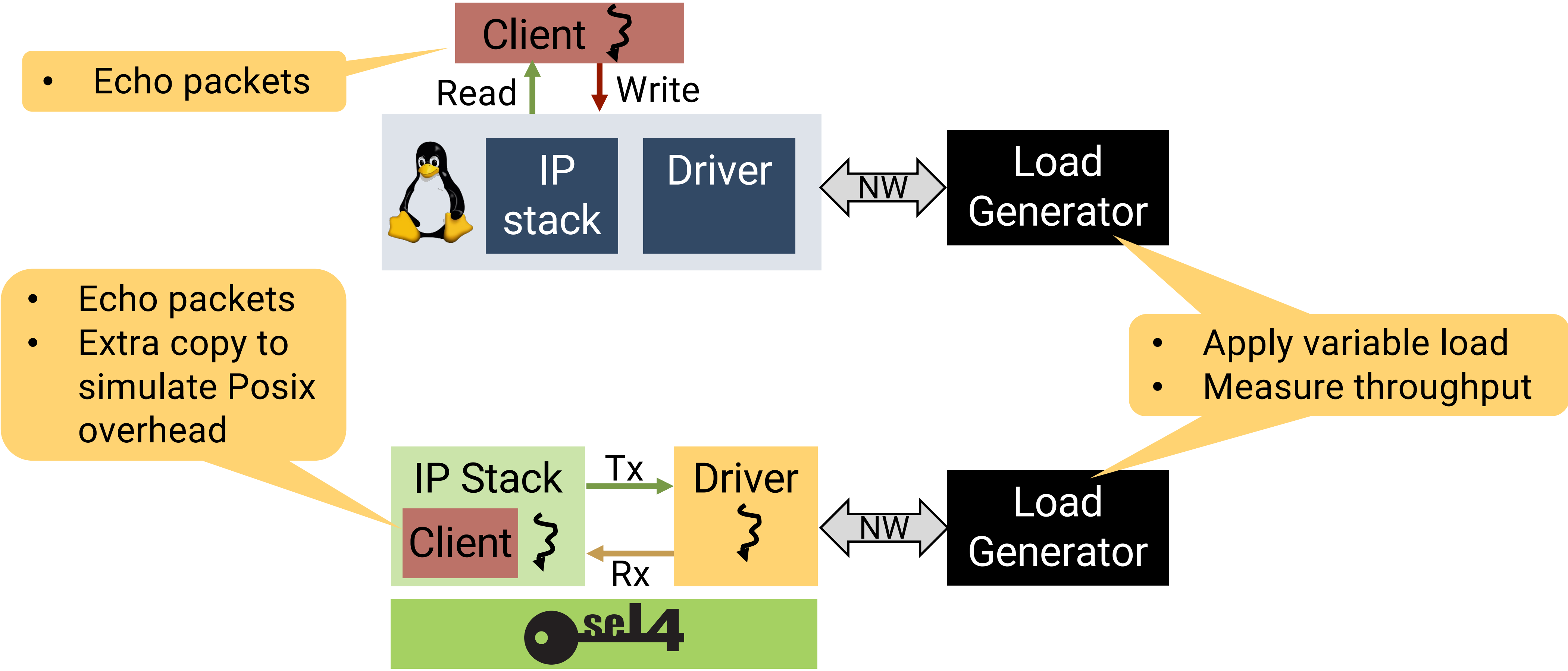


Transport Architecture Scales

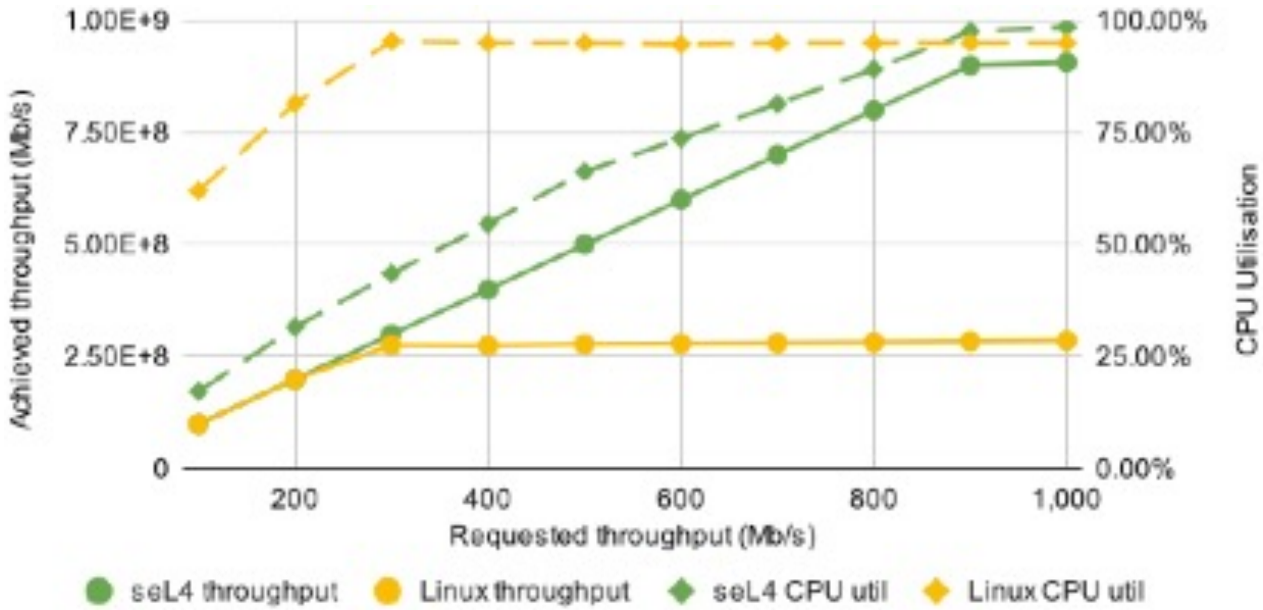
- Components can be on separate cores
- Driver, MUX close to minimal critical sections
- Should scale well without locks!



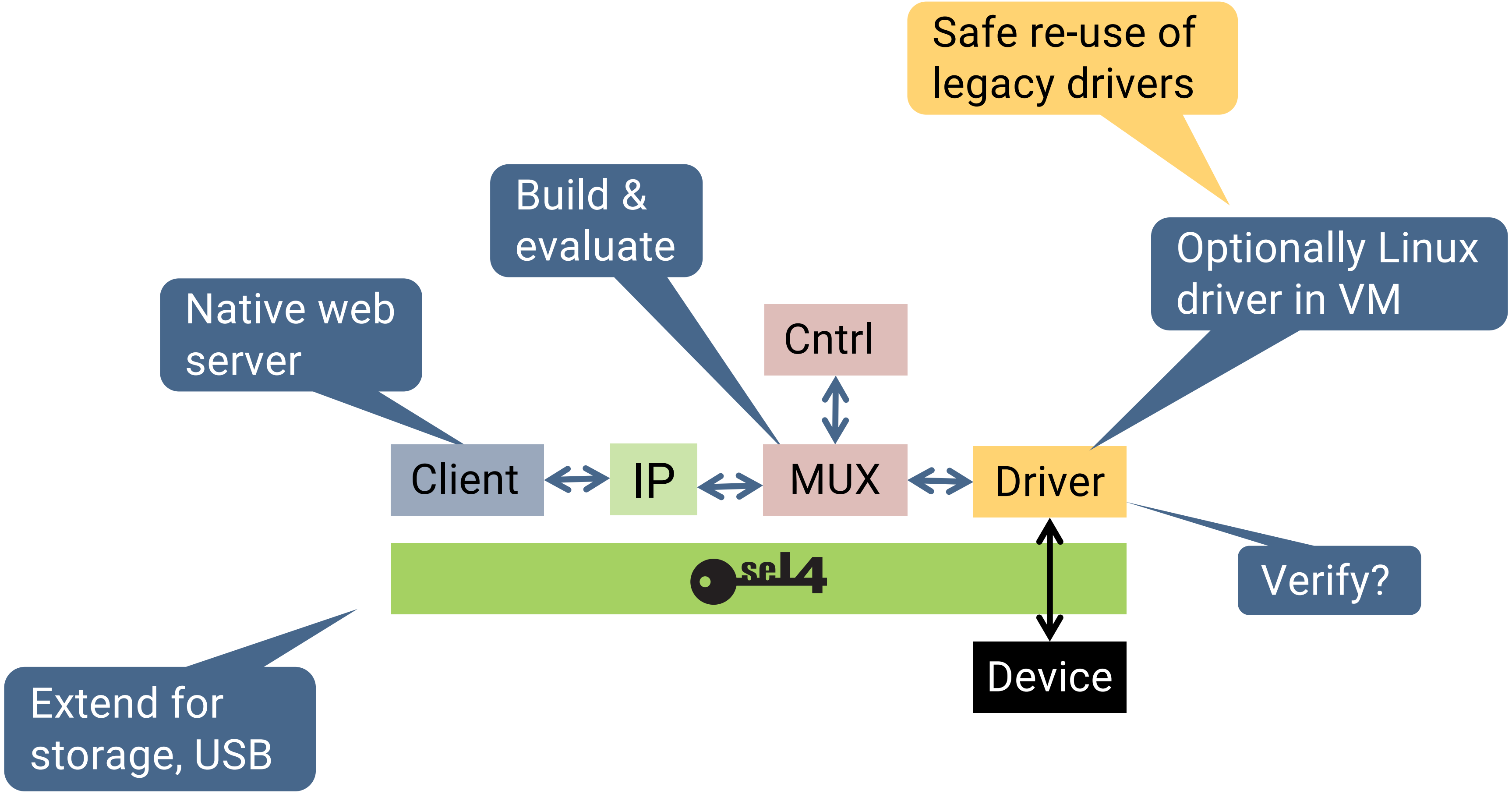
Preliminary Evaluation: Setup



Preliminary Evaluation: Performance



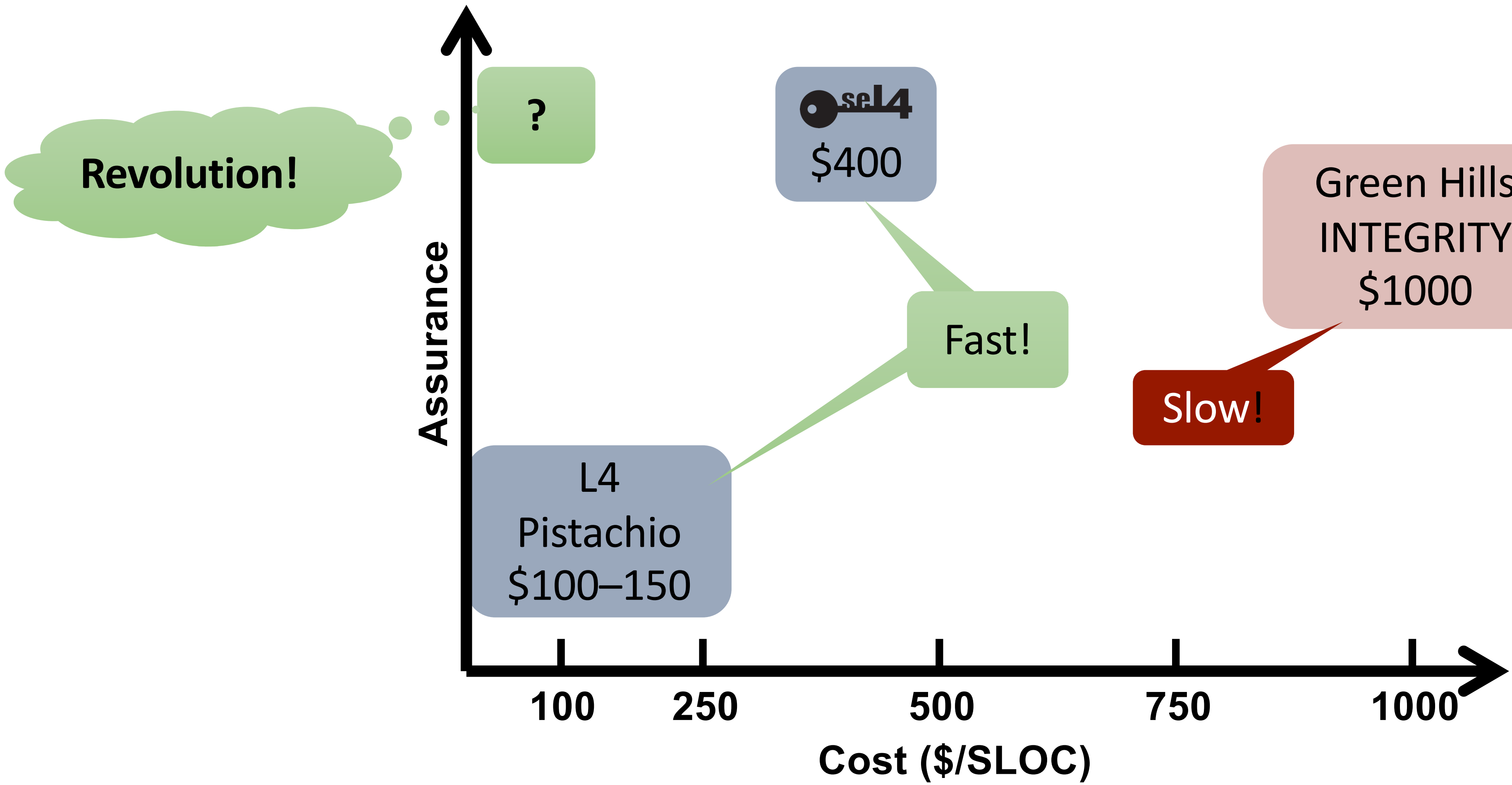
sDDF: Next Steps



seL4-Related Research in TS

Verifying Device Drivers?

Remember: Verification Cost in Context



Driver Dilemma

High seL4 verification costs partially due to C language

seL4 is one-off, justifies cost

Drivers are low-level, need C-like language

Better language would reduce cost

Drivers are commodity, must be cheap!

sDDF driver model!

Idea:
1. Simplify drivers
2. Design verification-friendly systems language
3. Automate (part of) verification

- Well-defined semantics
- Memory-safe

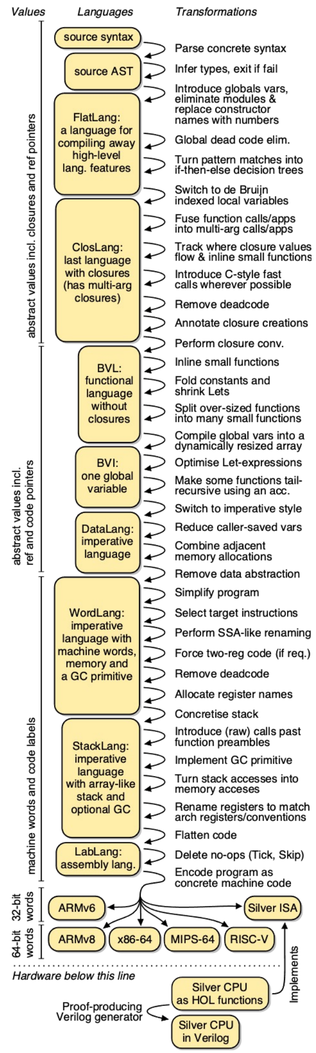
Verified compiler

CakeML: Verified Implementation of ML

- ✓ Mature functional language
- ✓ Large and active ecosystem of developers and users
- ✓ Code generation from abstract specs
- ❑ Managed ⇒ not suitable for systems code
- ✓ Used for verified application code

Re-use framework for new systems language: Pancake

https://cakeml.org



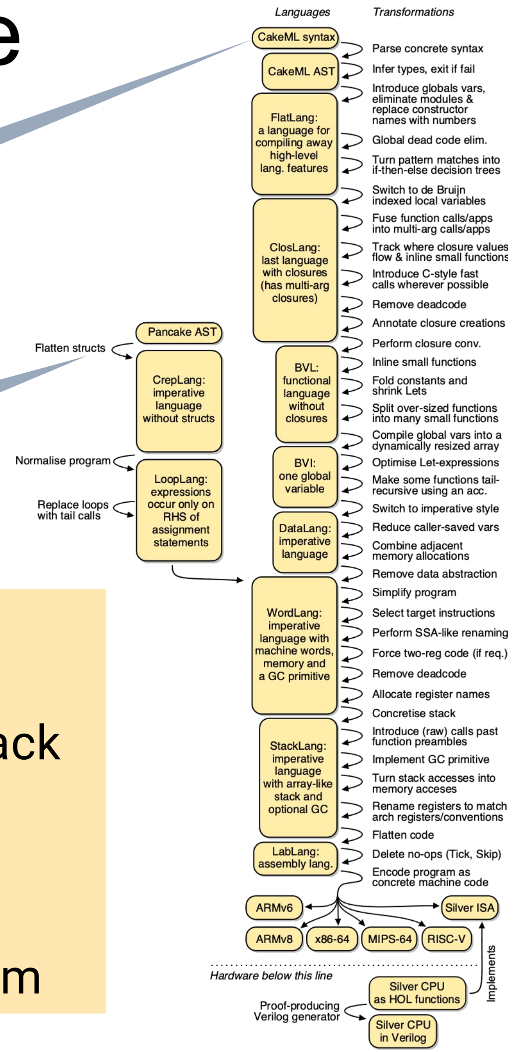
Pancake: New Systems Language

CakeML:
- functional language
- type & memory safe
- managed (garbage collector)
- high-level, abstract machine
- verified run time
- verified compiler
- mature system
- active ecosystem

In progress

CakeML

Pancake



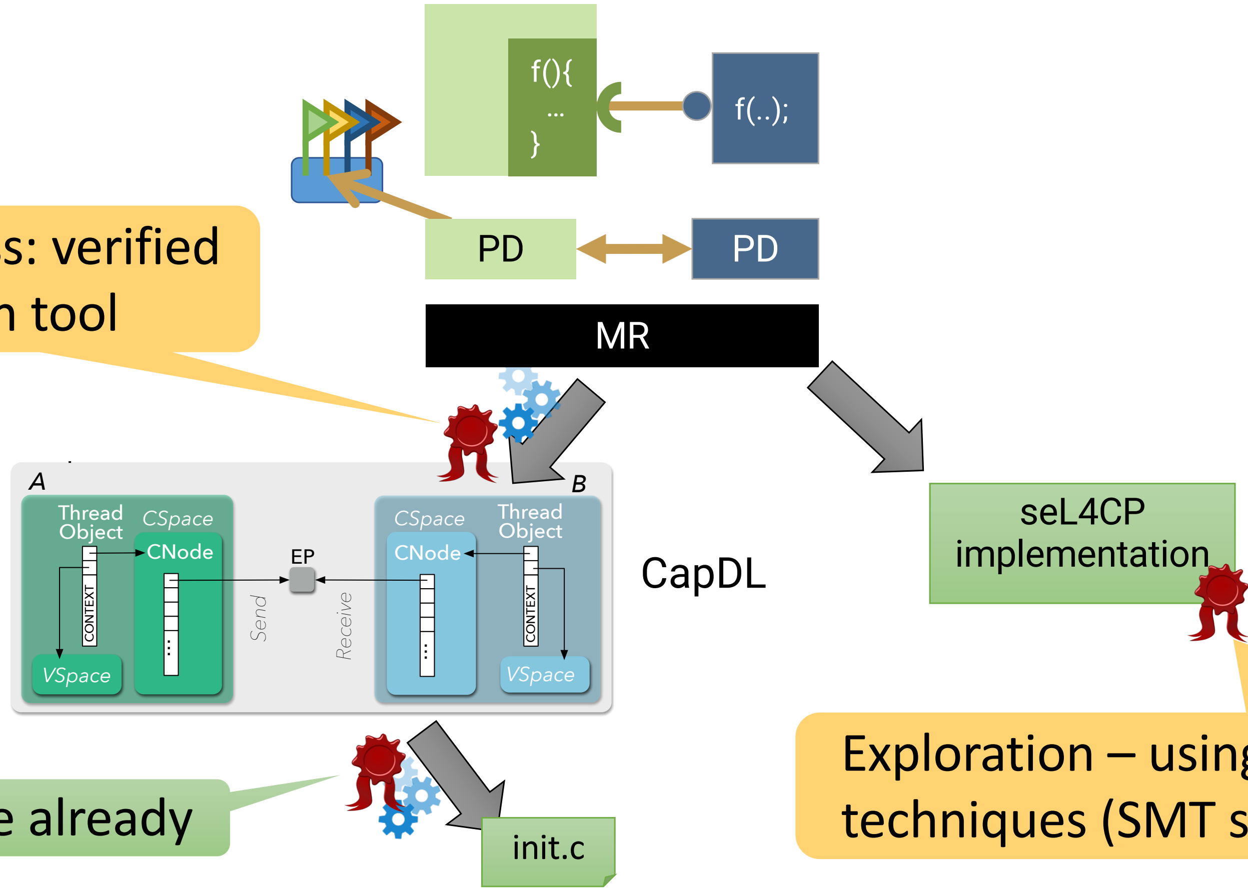
Approach:
- re-use lower part of CakeML compiler stack
- pathway to verified Pancake compiler
- Retain mature framework/ecosystem

UNSW SYDNEY

seL4-Related Research in TS

Verifying the seL4 Core Platform

seL4CP Verification



In progress: verified translation tool

Done already

Exploration – using automated techniques (SMT solvers)

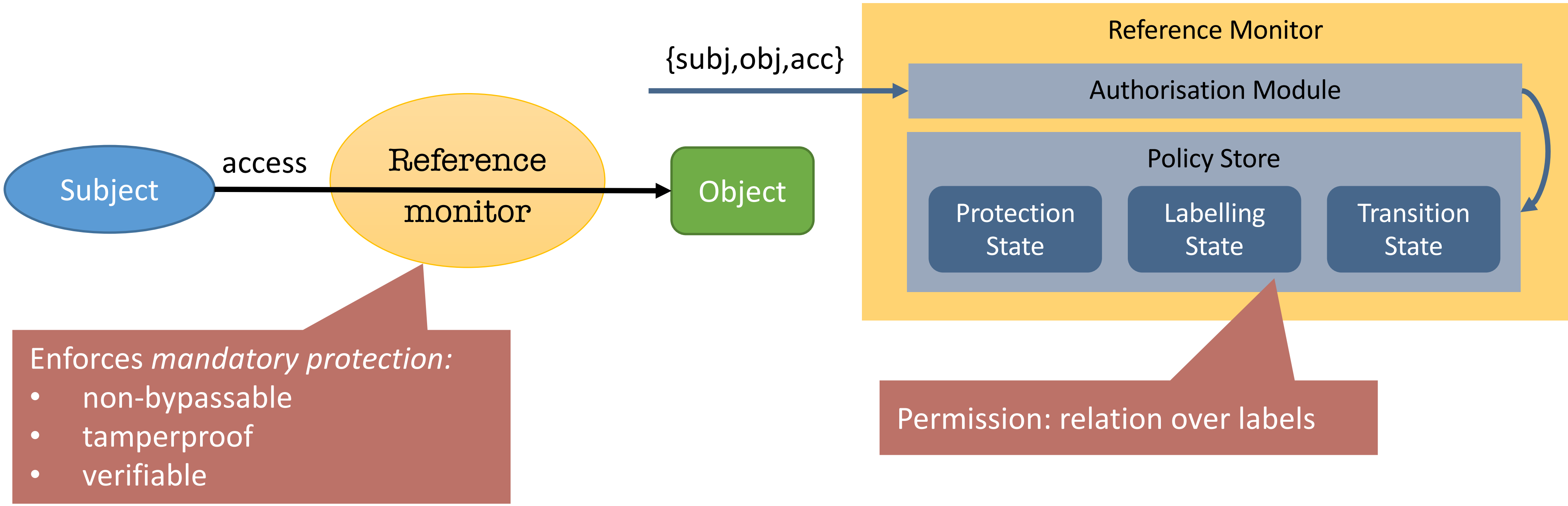
seL4-Related Research in TS

Secure Multi-Server OS

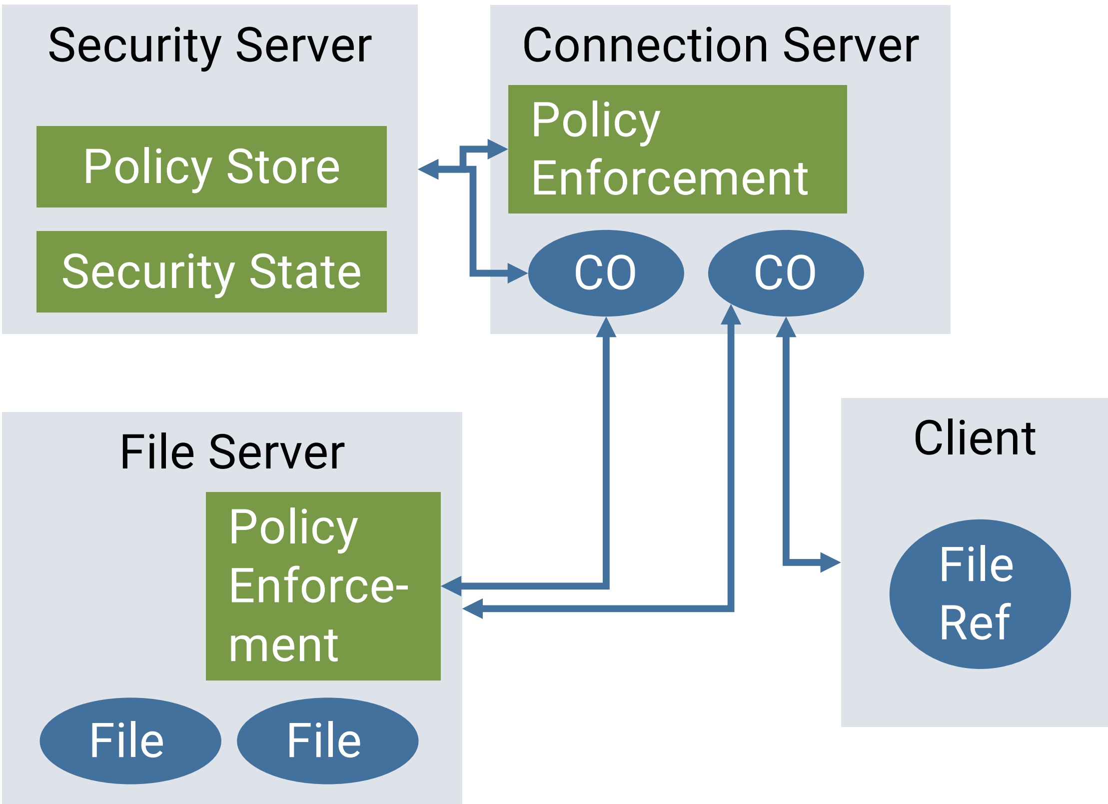
Recap: Secure Operating Systems

Secure OS: [Jaeger: OS Security]

Access enforcement satisfies the *reference monitor* concept



Secure, General-Purpose OS



Aim: General-purpose OS that provably enforces a security policy

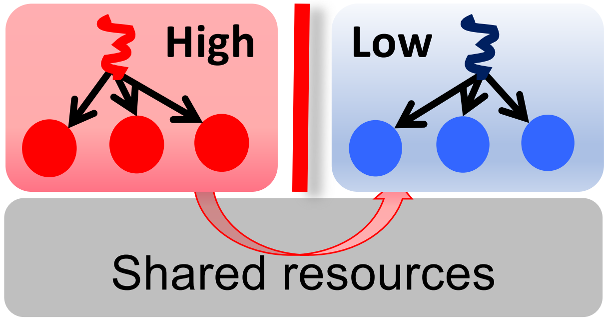
Requires:

- mandatory policy enforcement
- policy diversity
- minimal TCB
- low-overhead enforcement

seL4-Related Research in TS

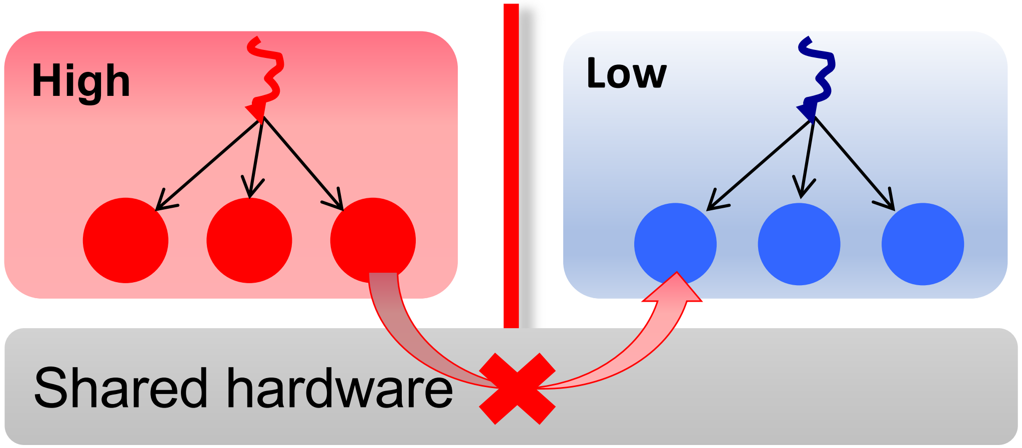
Time Protection: Verified Prevention of Microarchitectural Timing Channels

Refresh: Microarchitectural Timing Channels



Contention for shared hardware resources affects execution speed, leading to timing channels

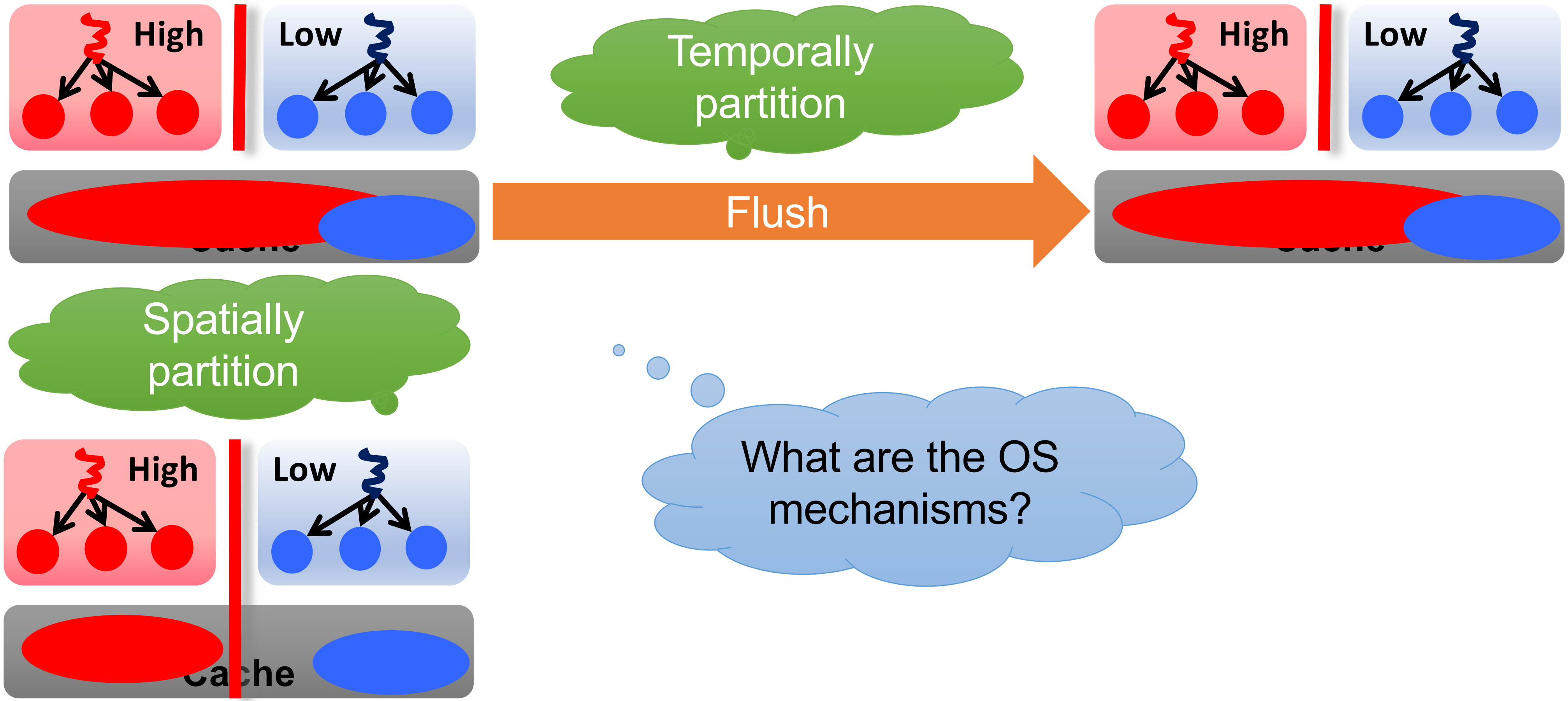
OS Must Enforce *Time Protection*



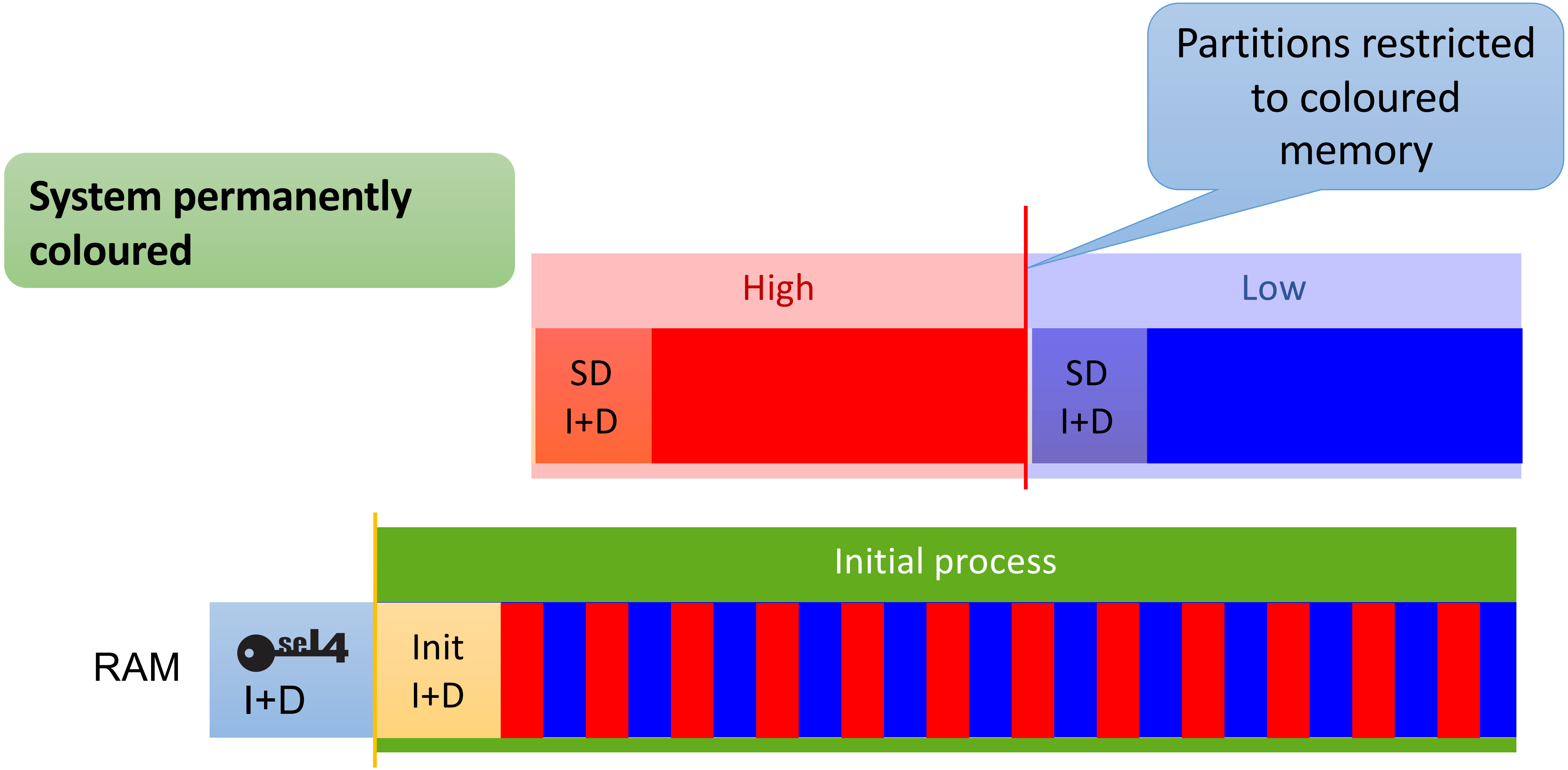
Preventing interference is core duty of the OS!

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

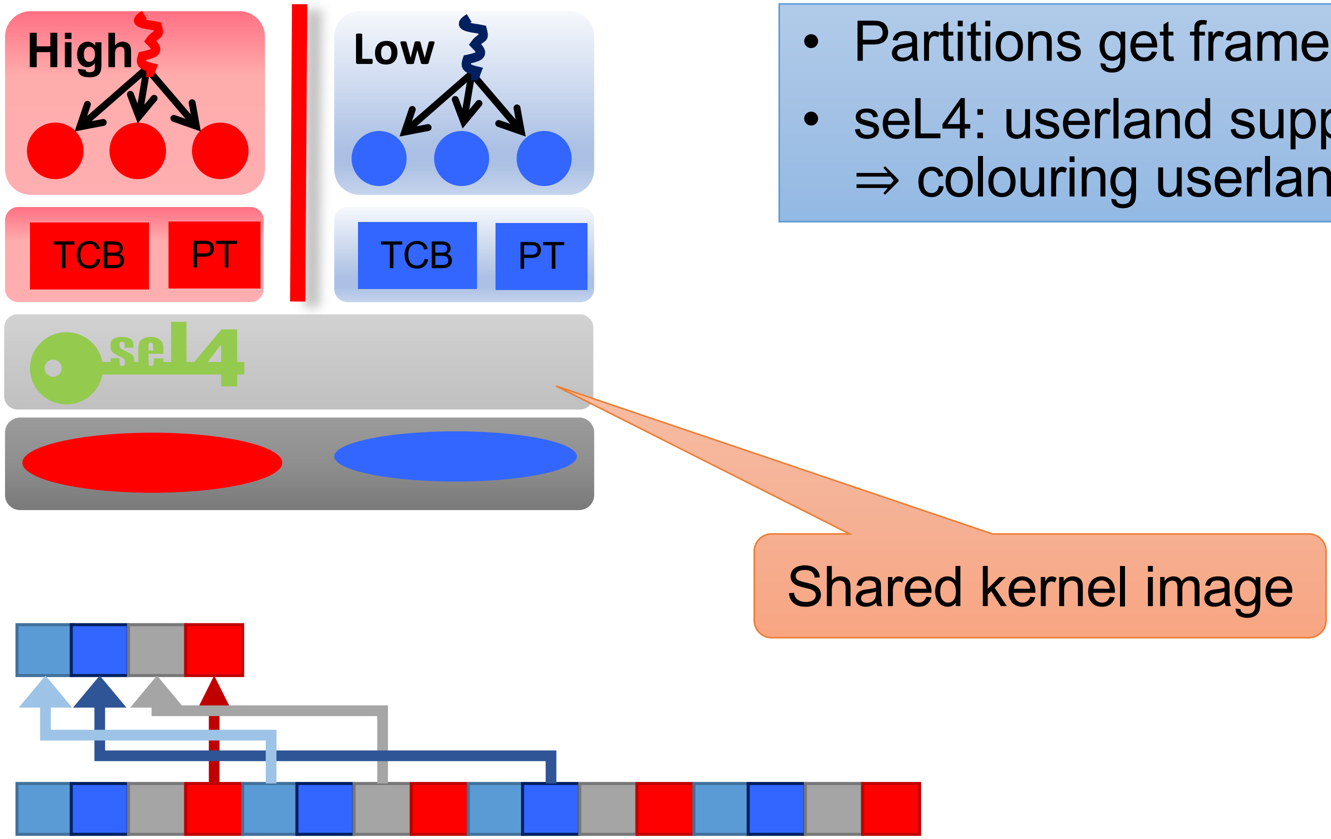
Time Protection: No Sharing of HW State



Spatial Partitioning: Cache Colouring



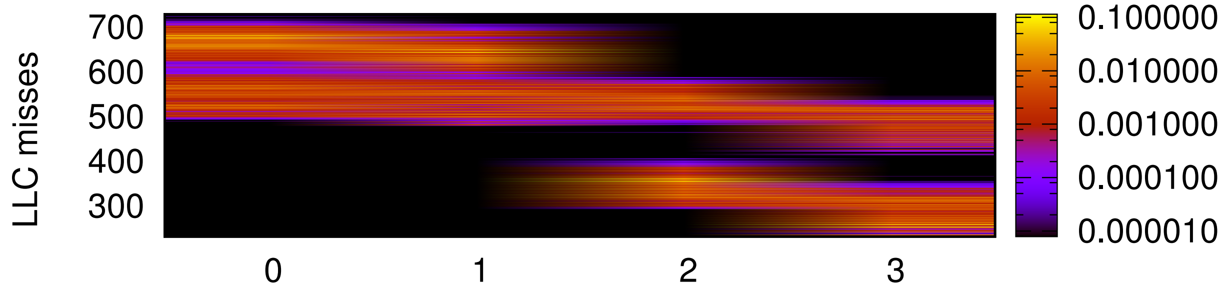
Spatial Partitioning: Cache Colouring



- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours kernel memory

Channel Through Kernel Code

Raw channel

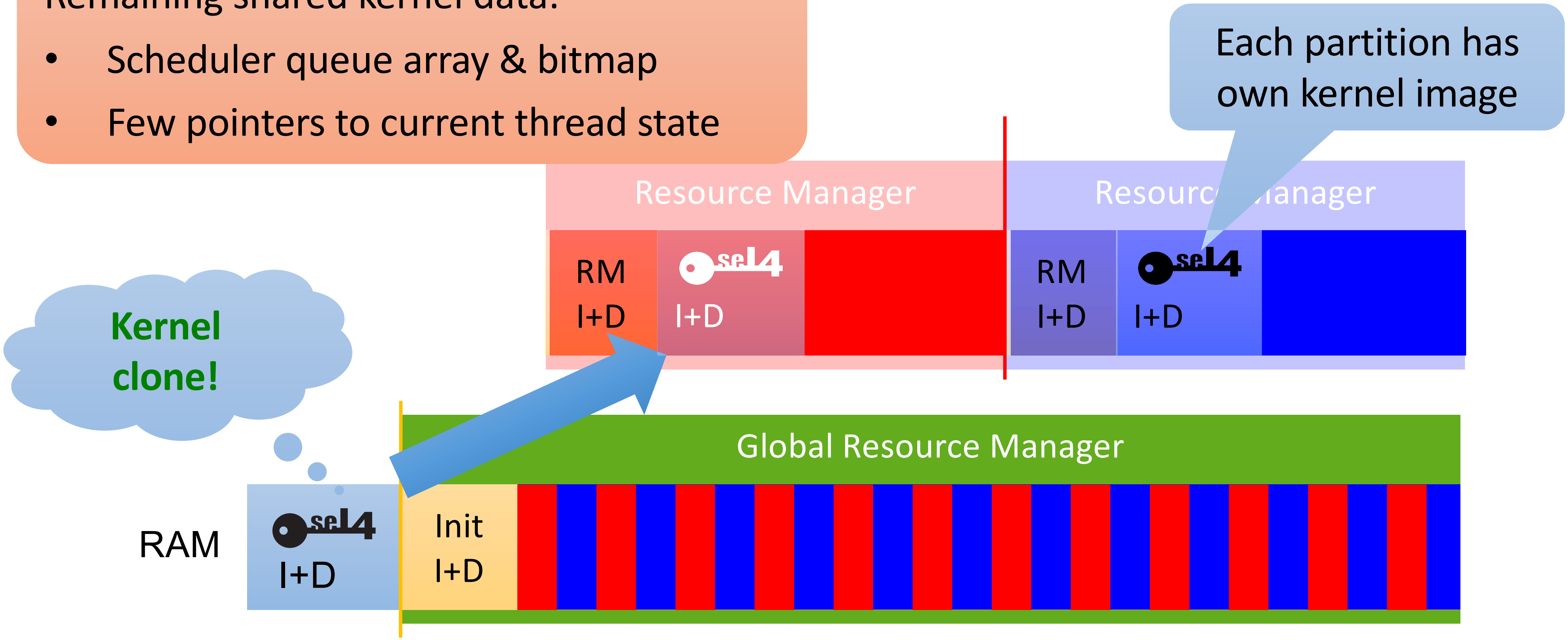


Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)

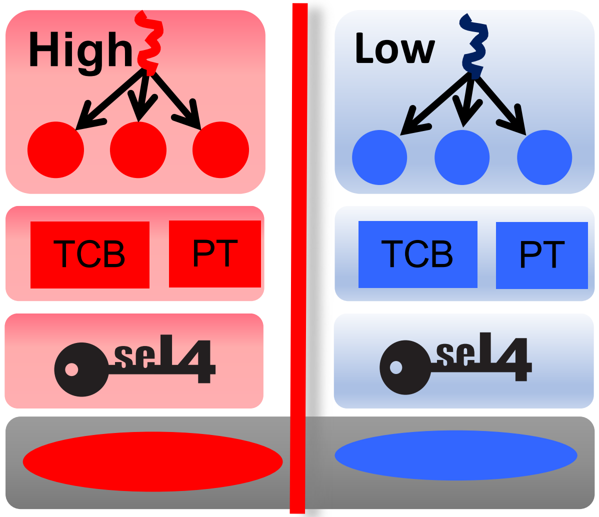
Colouring the Kernel

Remaining shared kernel data:

- Scheduler queue array & bitmap
- Few pointers to current thread state

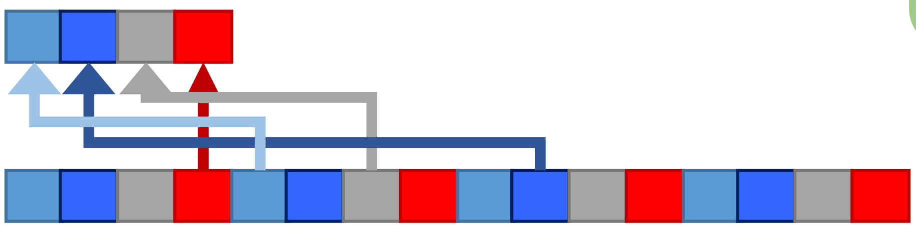


Spatial Partitioning: Cache Colouring



- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours kernel memory
- Per-partition kernel image to colour kernel

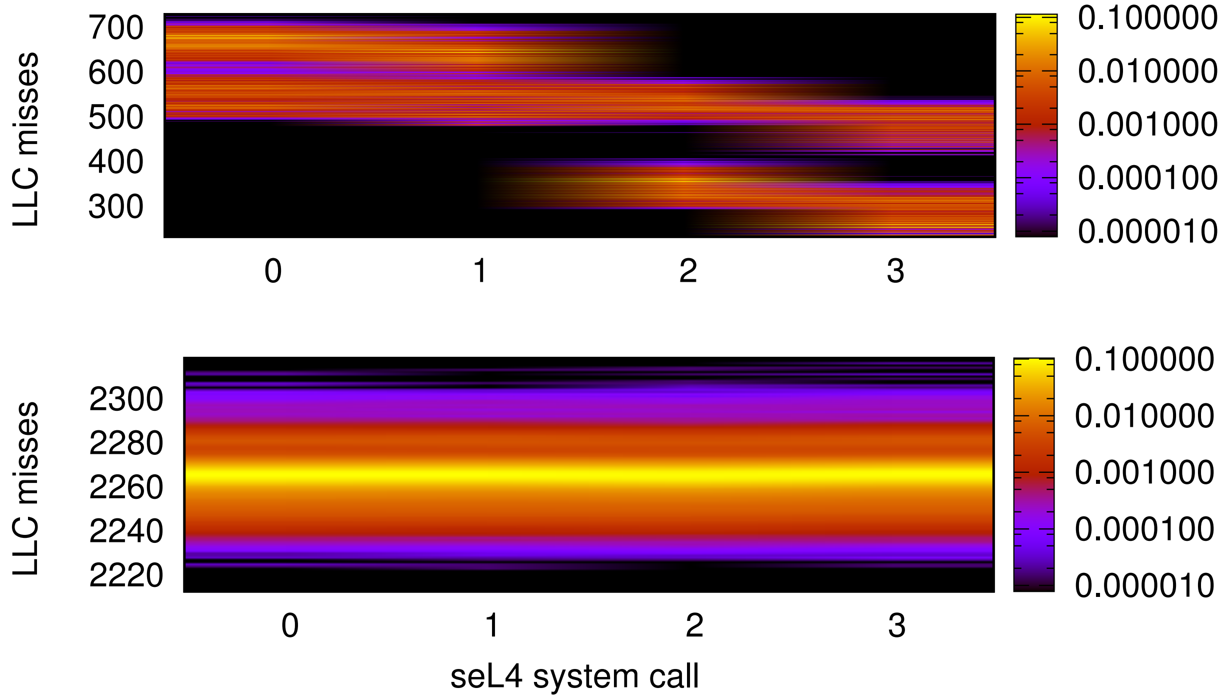
Must ensure deterministic access to remaining shared kernel state!



Channel Through Kernel Code

Raw channel

Channel with cloned kernel



Temporal Partitioning: Flush on Switch

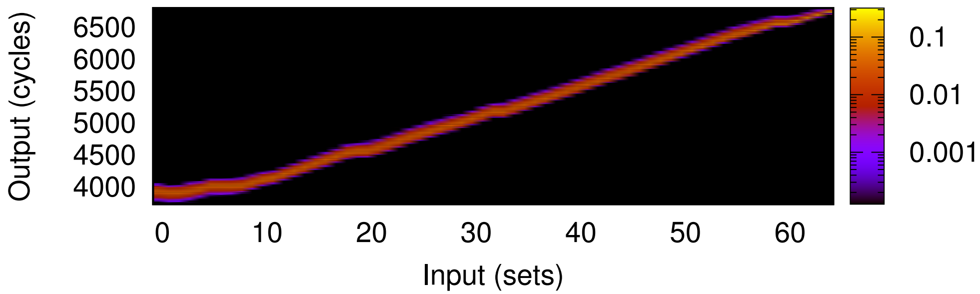
Must remove any history dependence!

2. Switch user context
3. Flush on-core state

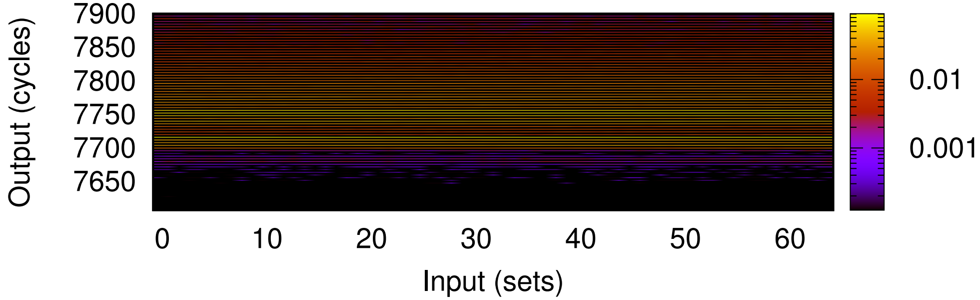
6. Reprogram timer
7. return

D-Cache Channel

Raw channel

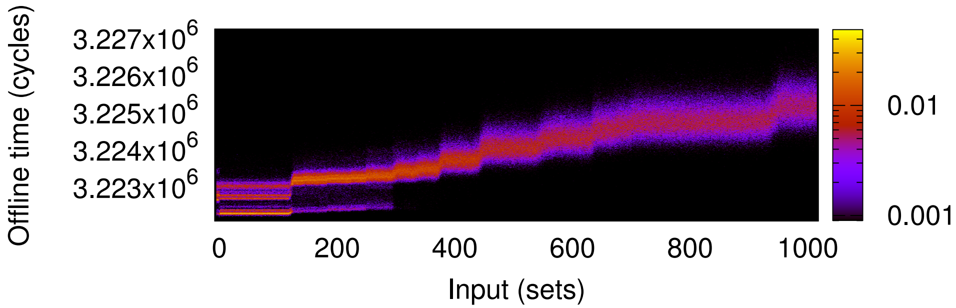


Channel with flushing



Flush-Time Channel

Raw channel



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 (T_0+WCET < current_time()) ;
6. Reprogram timer
7. return

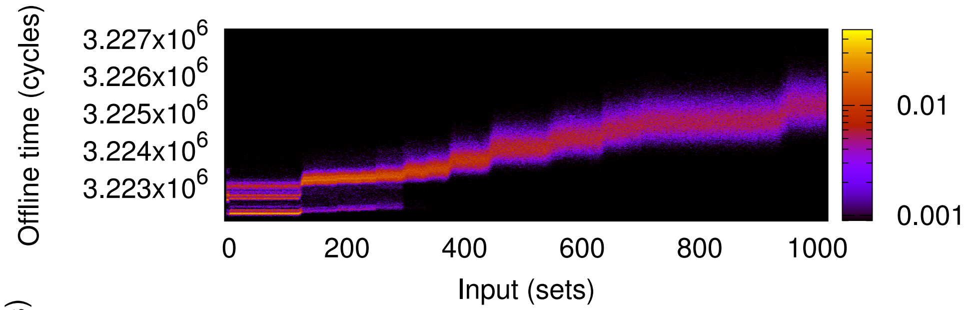
Latency depends on prior execution!

Time padding to remove dependency

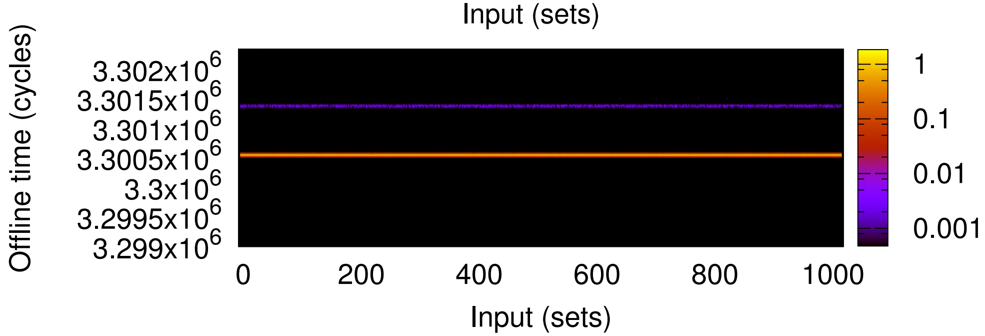
Ensure deterministic execution

Flush-Time Channel

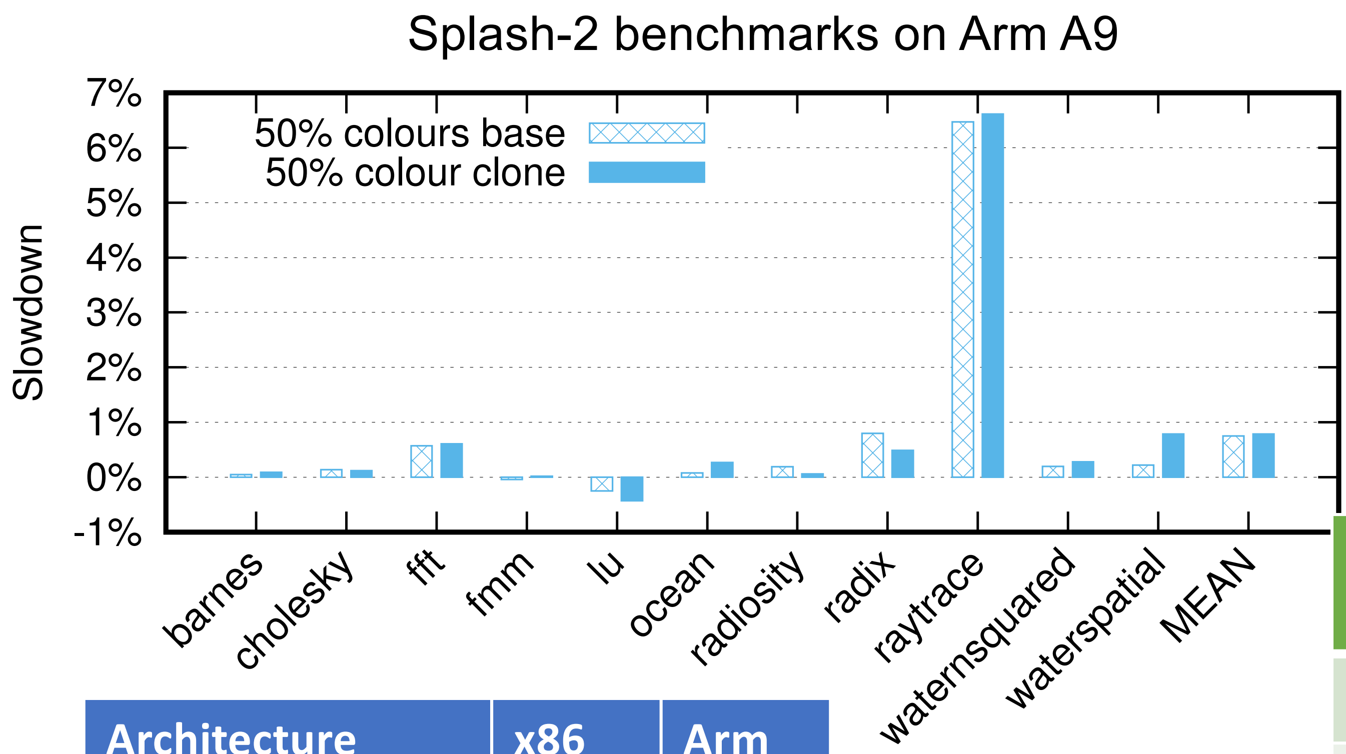
Raw channel



Channel with deterministic flushing



Performance Impact of Colouring



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

[Ge et al., EuroSys'19]

Architecture	x86	Arm
Mean slowdown	3.4%	1.1%

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

A New HW/SW Contract

aISA: augmented ISA

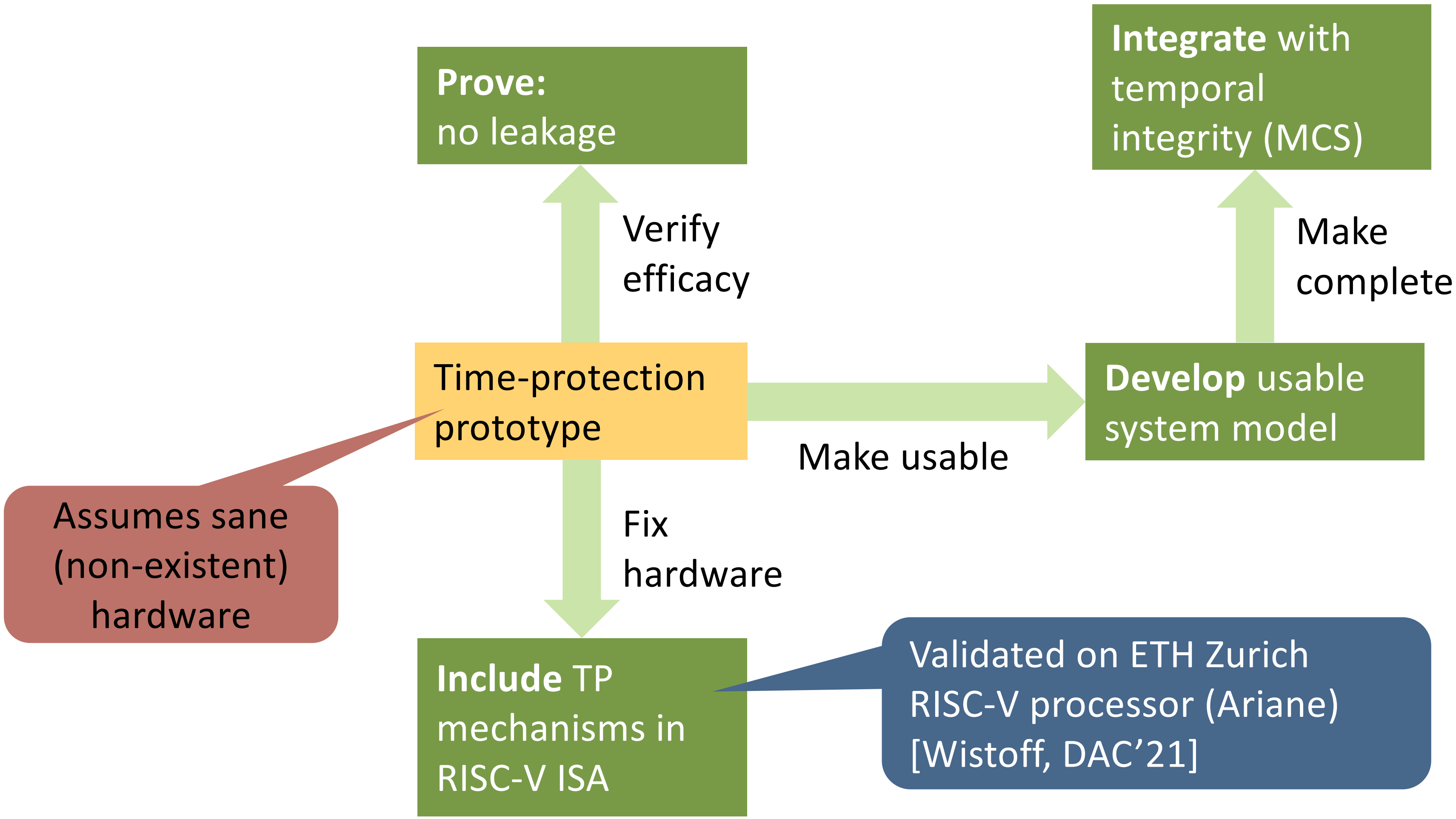
For all shared microarchitectural resources:

1. Resource must be spatially partitionable or flushable
2. Concurrently shared resources must be spatially partitioned
3. Resource accessed solely by virtual address must be flushed and not concurrently accessed
4. Mechanisms must be sufficiently specified for OS to partition or reset
5. Mechanisms must be constant time, or of specified, bounded latency
6. Desirable: OS should know if resettable state is derived from data, instructions, data addresses or instruction addresses

Cannot share HW threads across security domains!

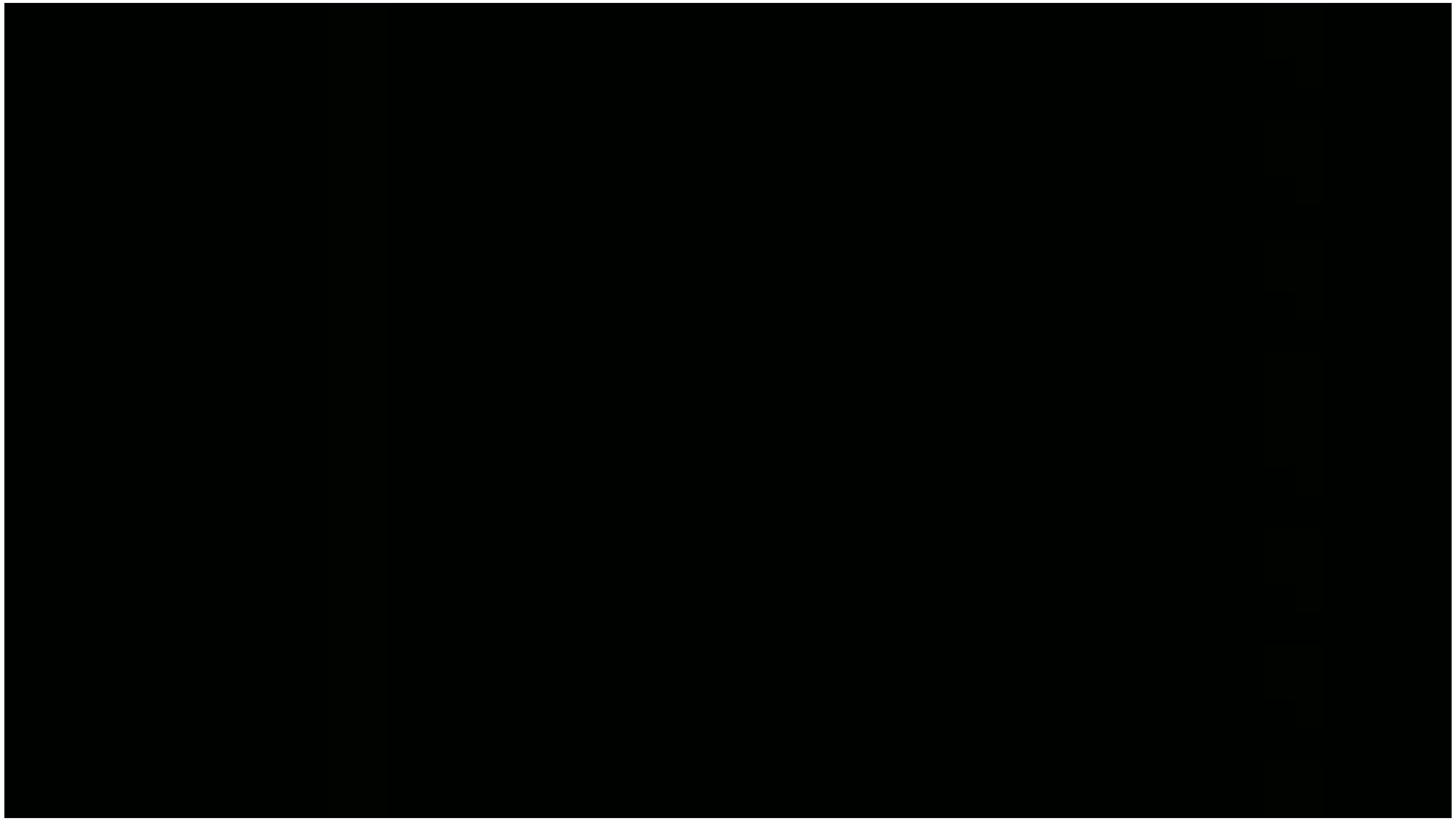
[Ge et al., APSys'18]

Time Protection: On-Going Work



Real-World Use

Courtesy Boeing, DARPA



Thank you!

To the dedicated AOS students for their interest and dedication

To the world-class Trustworthy Systems team for making all possible

Please remember to do the myExperience survey

There'll also be a more detailed one we'll invite you to fill in