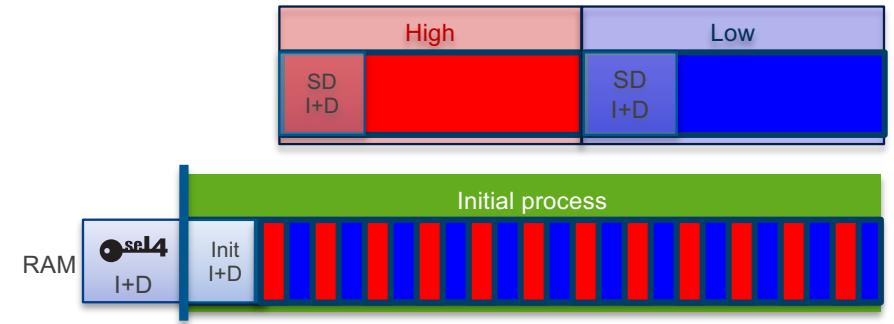


School of Computer Science & Engineering
COMP9242 Advanced Operating Systems

2022 T2 Week 10 Part 2

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



Copyright Notice

**These slides are distributed under the
Creative Commons Attribution 4.0 International (CC BY 4.0) License**

- You are free:
 - to share—to copy, distribute and transmit the work
 - to remix—to adapt the work
- under the following conditions:
 - **Attribution:** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:

“Courtesy of Gernot Heiser, UNSW Sydney”

The complete license text can be found at
<http://creativecommons.org/licenses/by/4.0/legalcode>

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)



Off-the-shelf
Drone airframe

Retrofit
existing
system!



Develop
technology



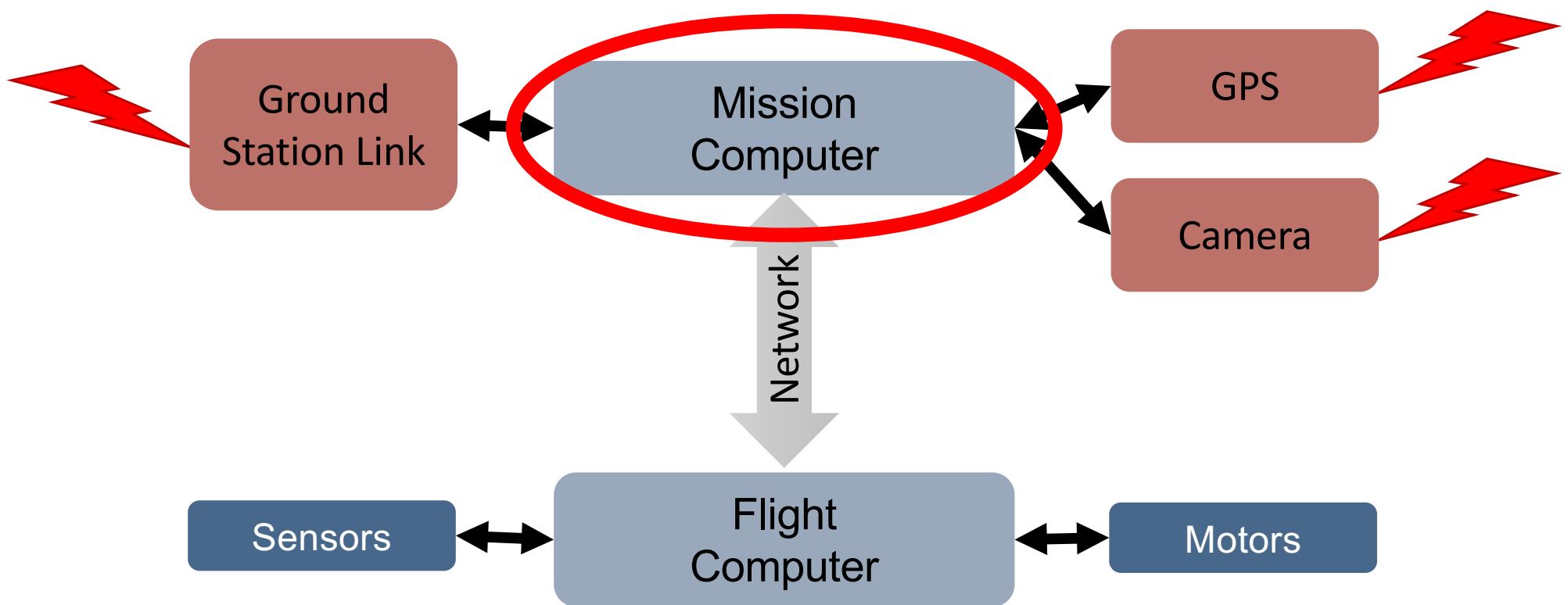
Autonomous trucks



GVR-BOT

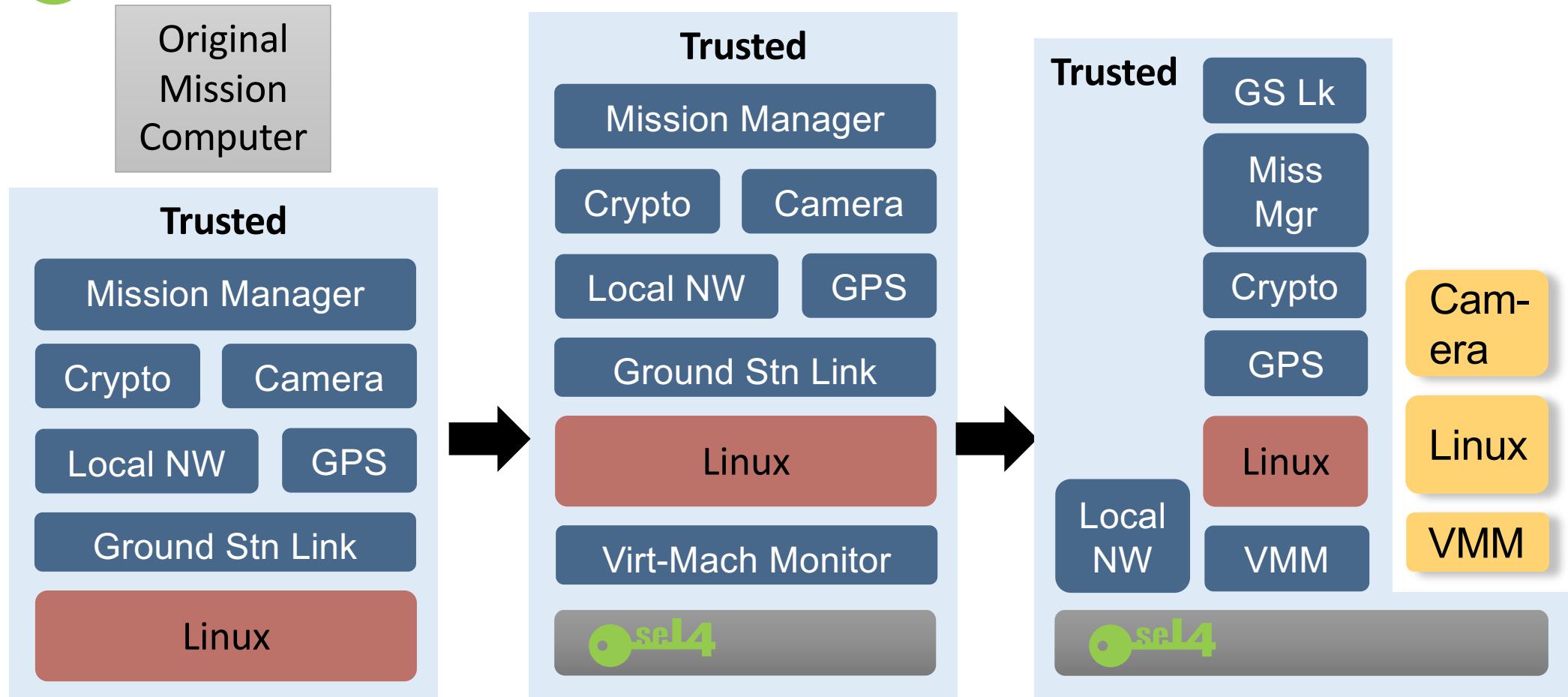


ULB Architecture

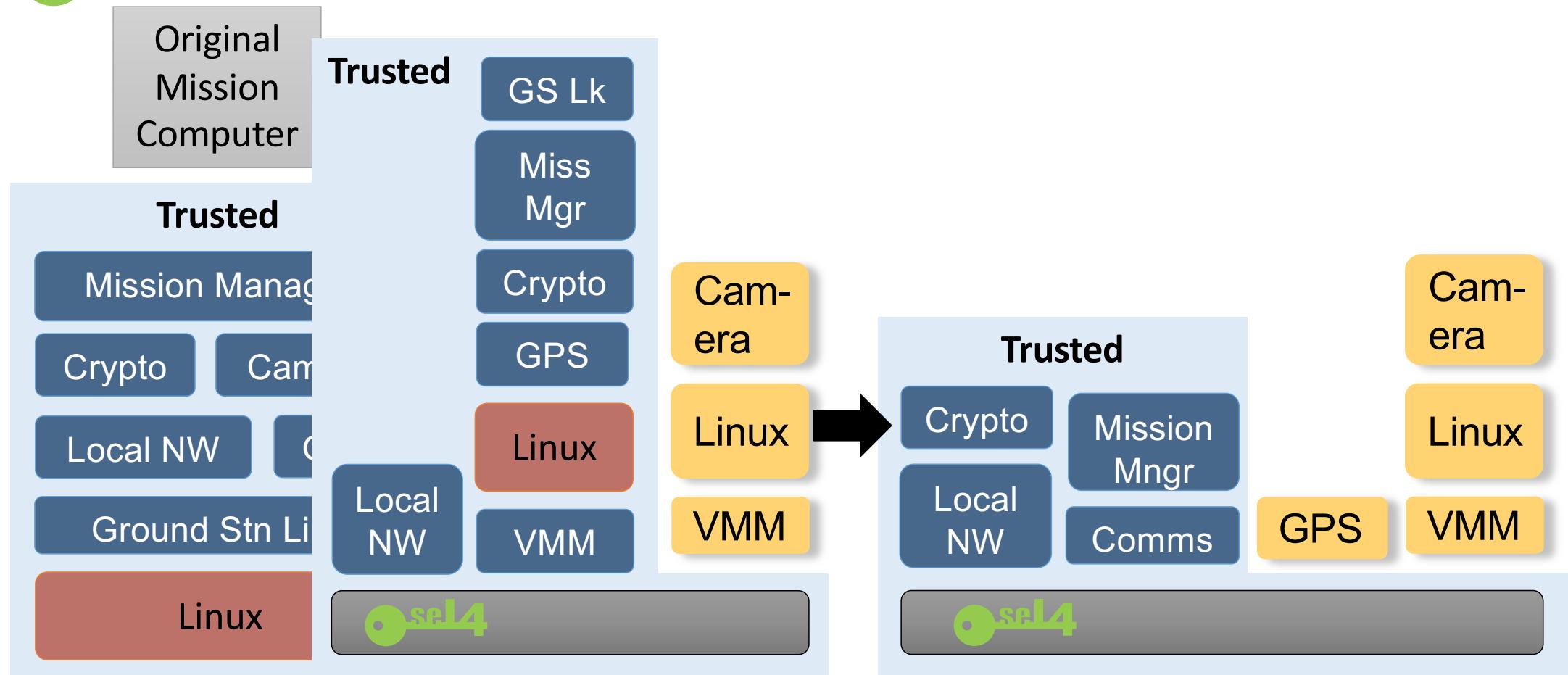




Incremental Cyber Retrofit



•sel4 Incremental Cyber Retrofit





Incremental Cyber Retrofit

Original
Mission Computer

[Klein et al, CACM, Oct'18]

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Cyber-secure Mission
Computer

Cam-
era

Linux

GPS

VMM



Trusted

Crypto

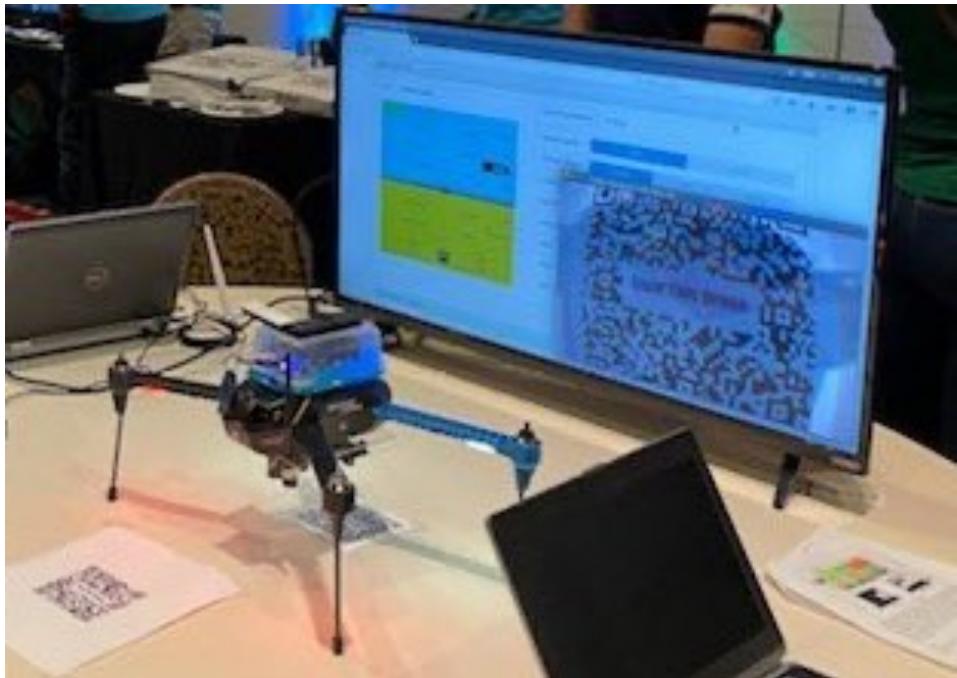
Mission
Mngr

Local
NW

Comms



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

•sel4 seL4 in Products





The seL4 Foundation

Premium Members



地平线
Horizon Robotics



jumptrading



UNSW
SYDNEY



General Members



Associate Members



in association with
National Cyber
Security Centre



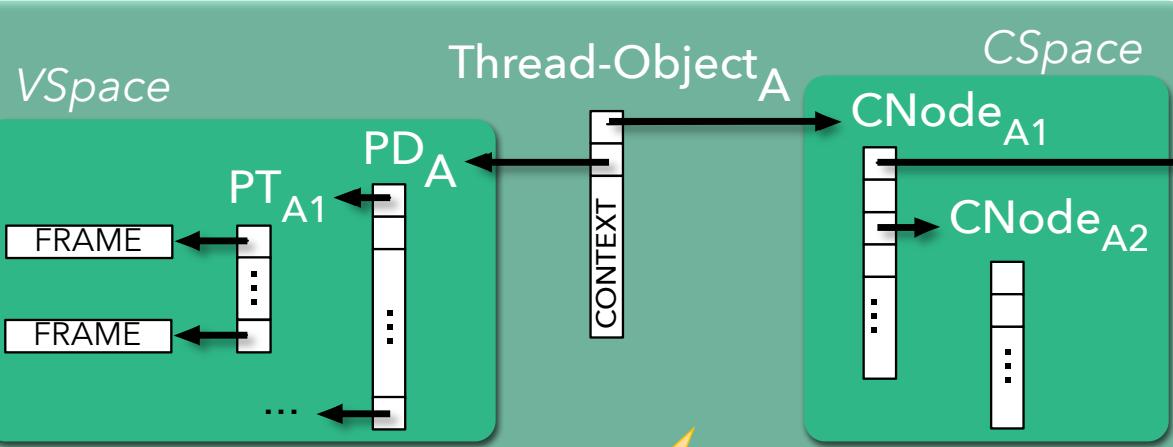
Usability

CAmkES and the seL4 Core Platform

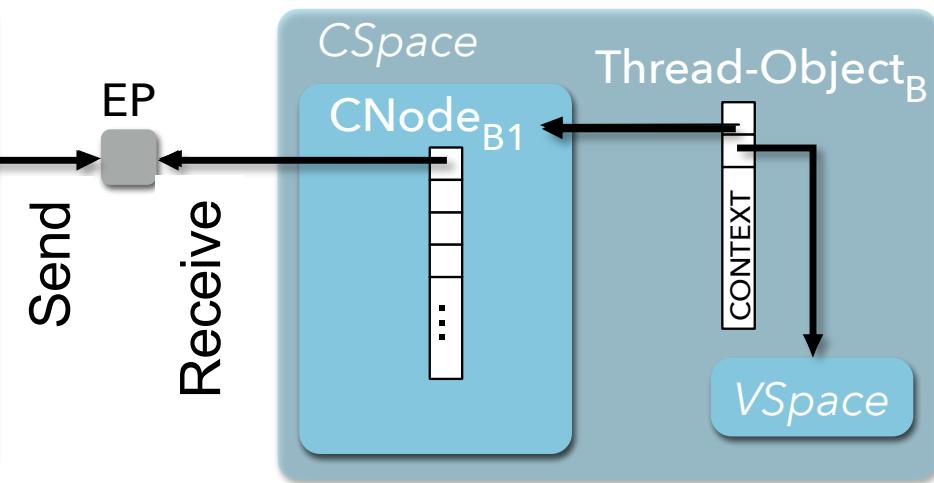


Issue: seL4 Objects are Low-Level

A



B

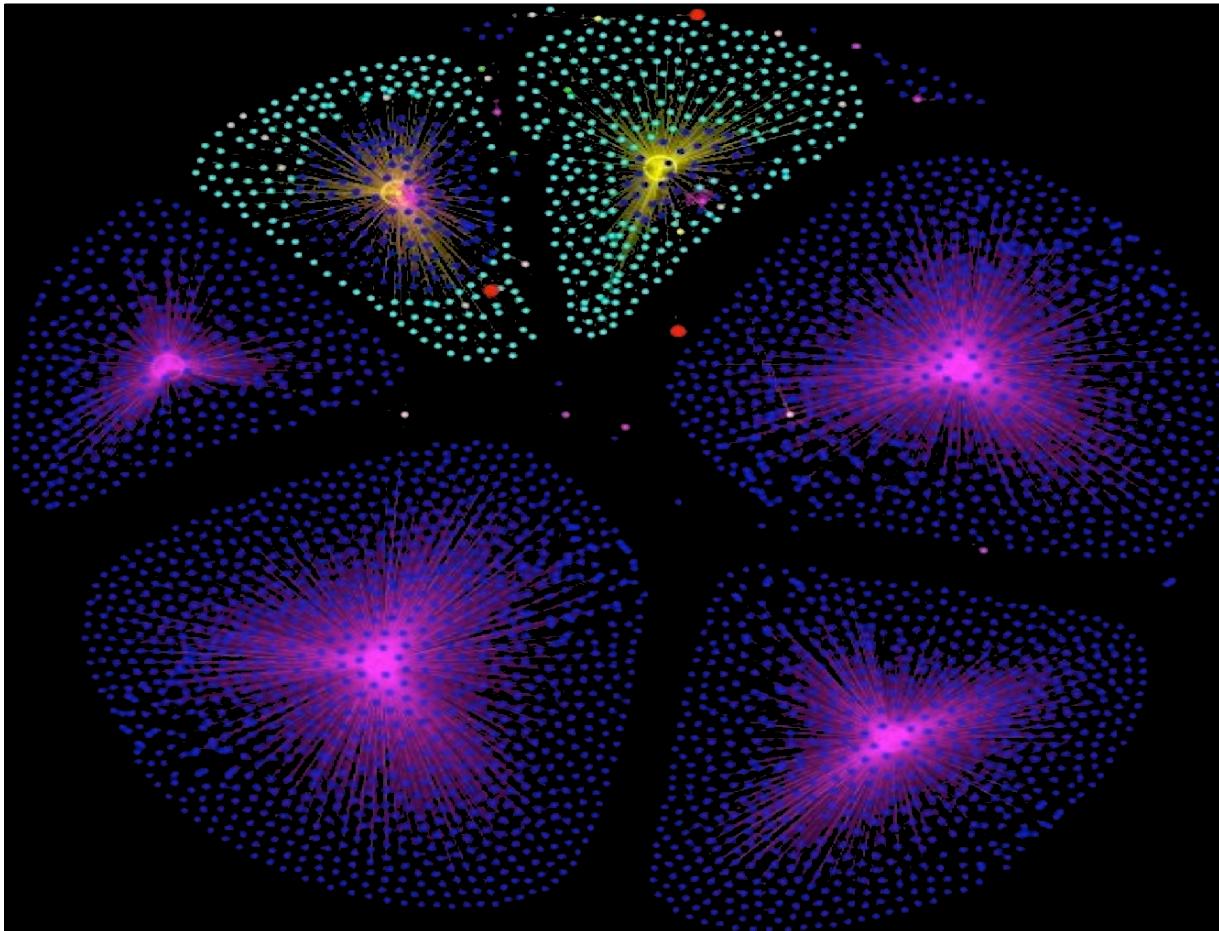


Send
Receive

>50 kernel objects
for trivial program!

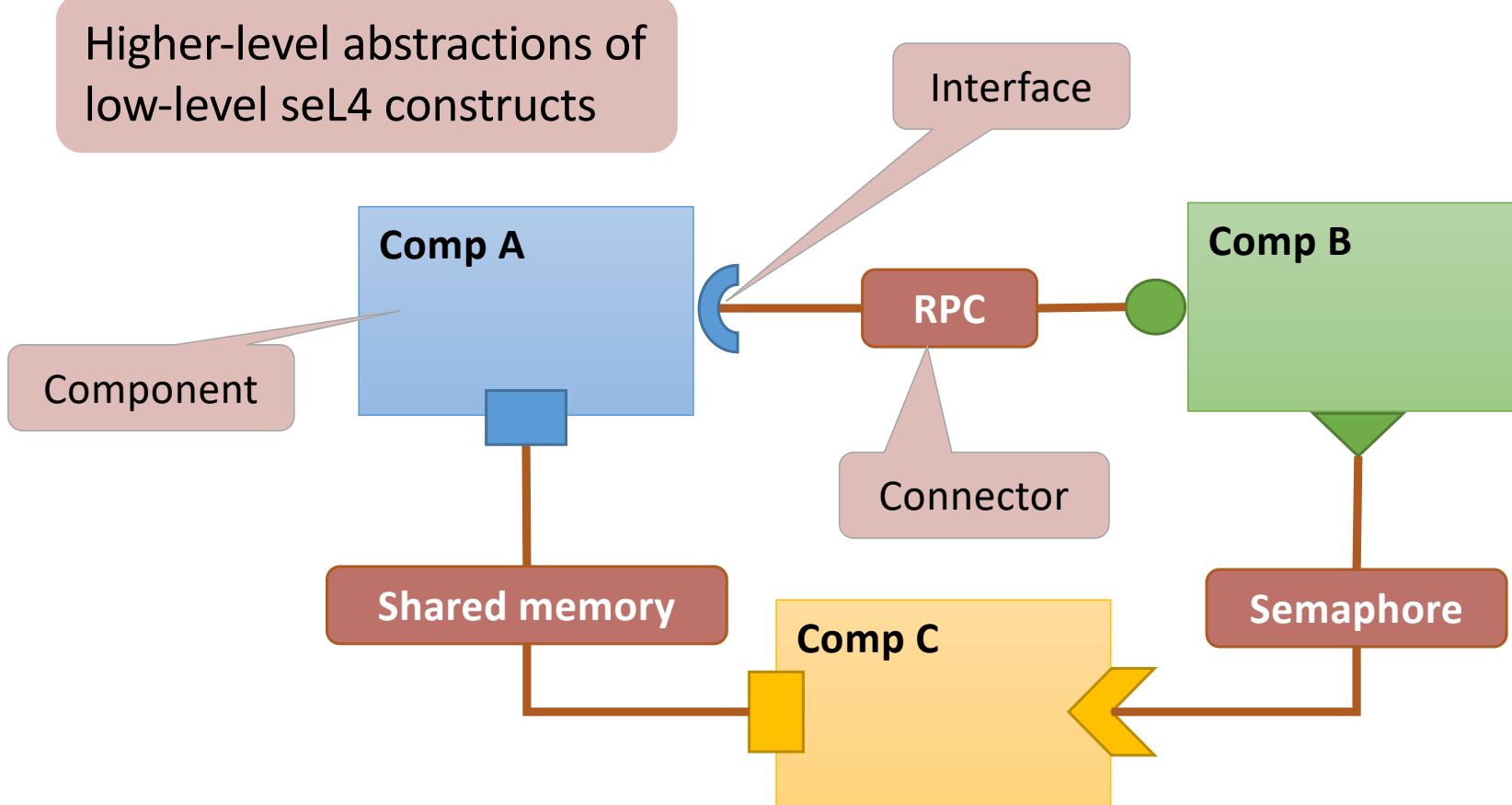


Simple But Non-Trivial System





Recommended Framework: CAmkES

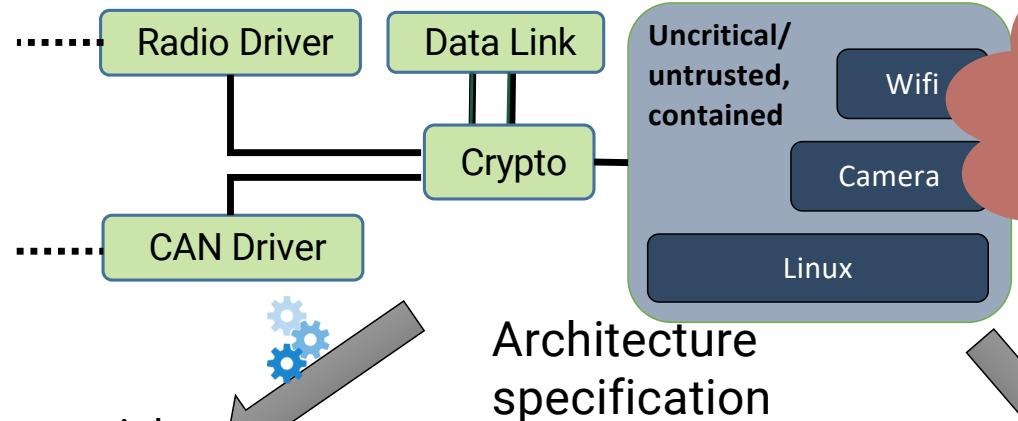
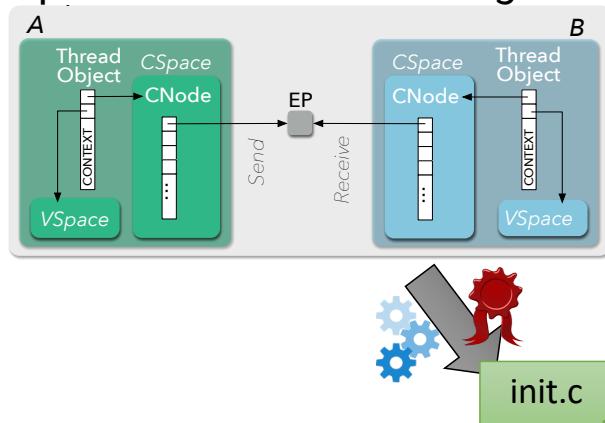




CAmkES Framework

Conditions apply

CapDL: Low-level access rights

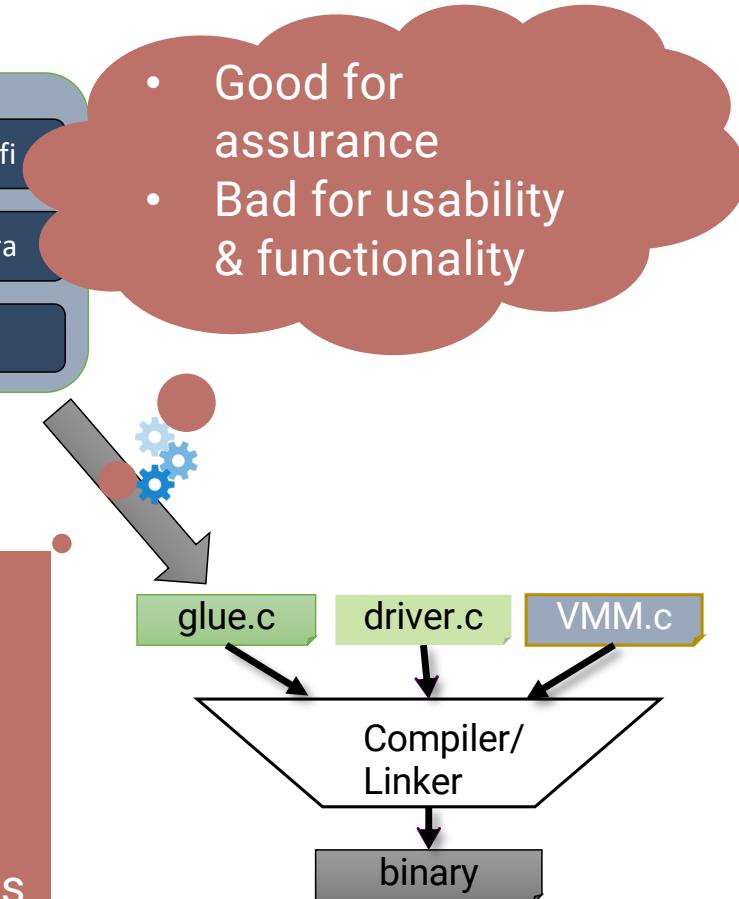


- Good for assurance
- Bad for usability & functionality

Architecture specification

However:

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



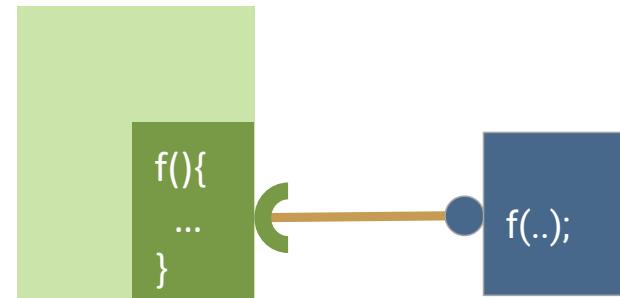
New Framework: seL4 Core Platform

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

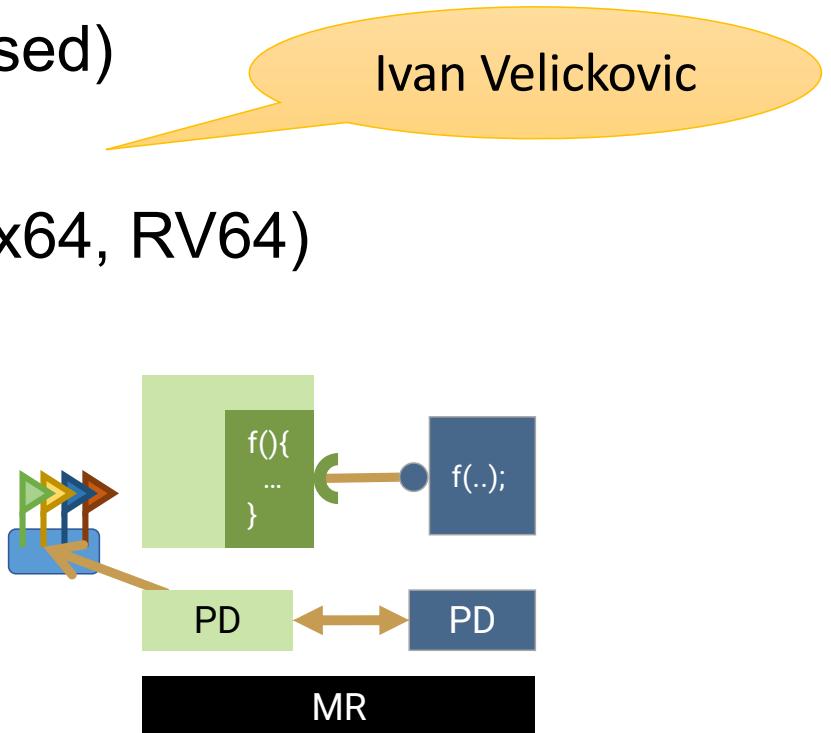
- Thin wrapper of seL4 abstractions
- Encourage “correct” use of seL4



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)

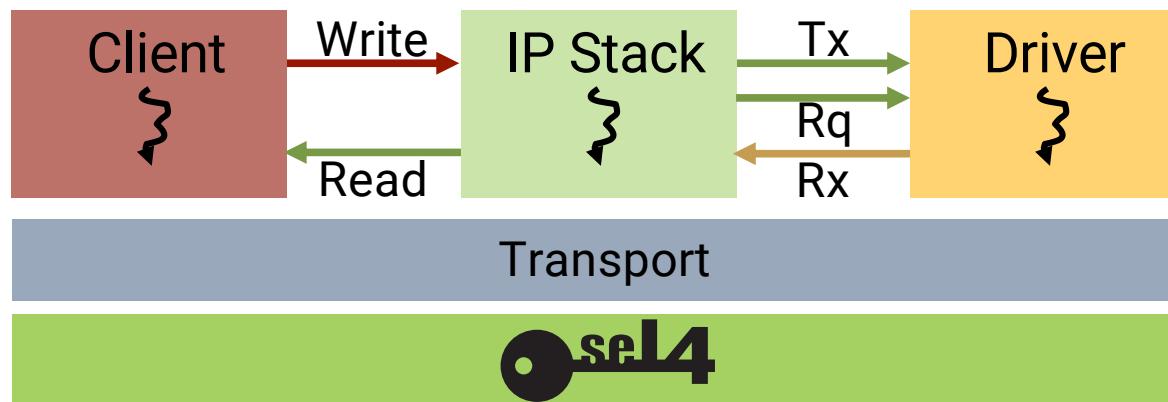
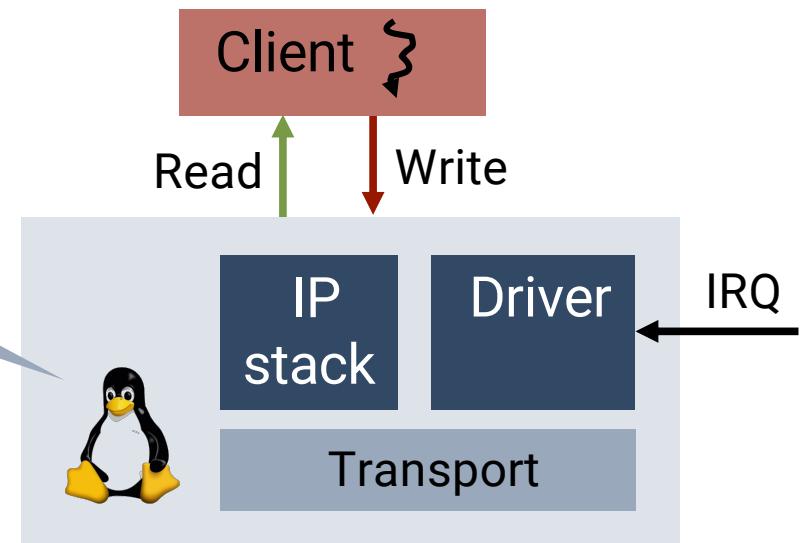


seL4-Related Research in TS

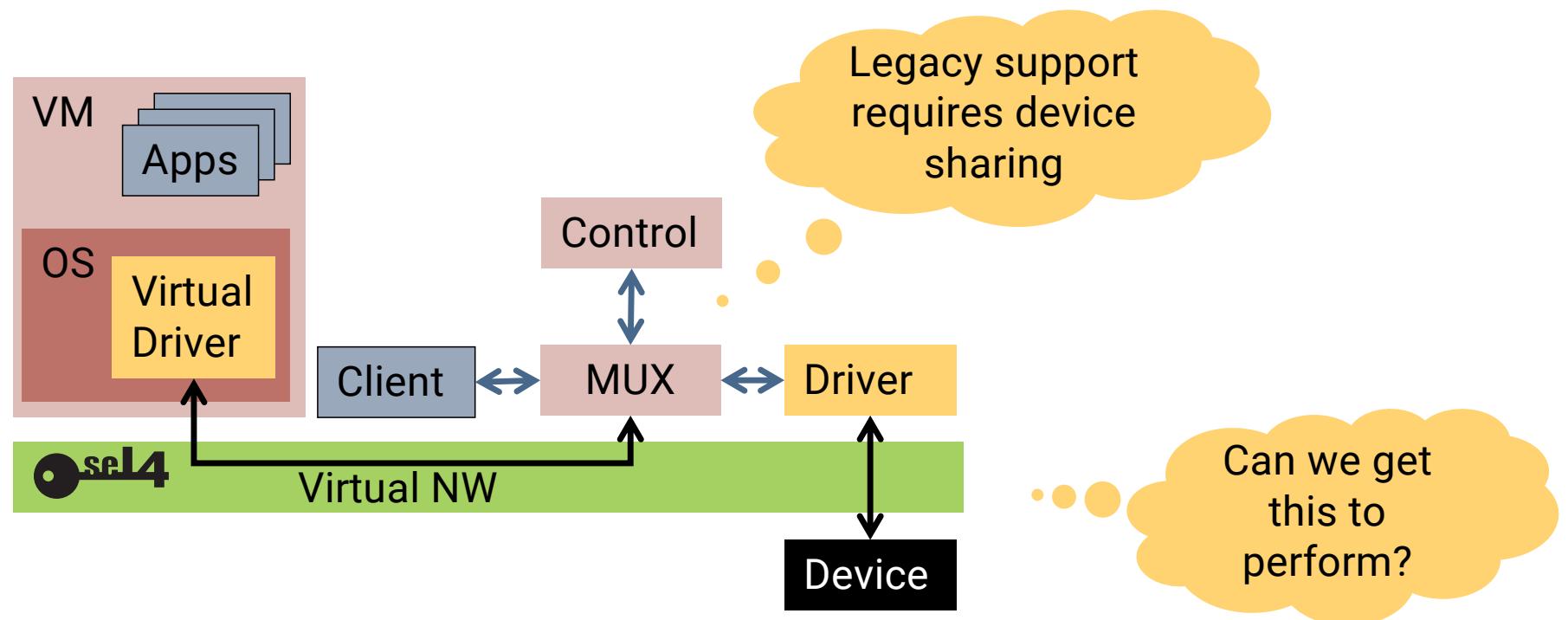
High-Performance I/O and I/O Virtualisation

I/O Architecture

- 1 syscall per I/O
 - no fault containment
-
- many syscalls per I/O
 - good fault containment



Device Sharing (aka I/O Virtualisation)



Advanced I/O Architecture

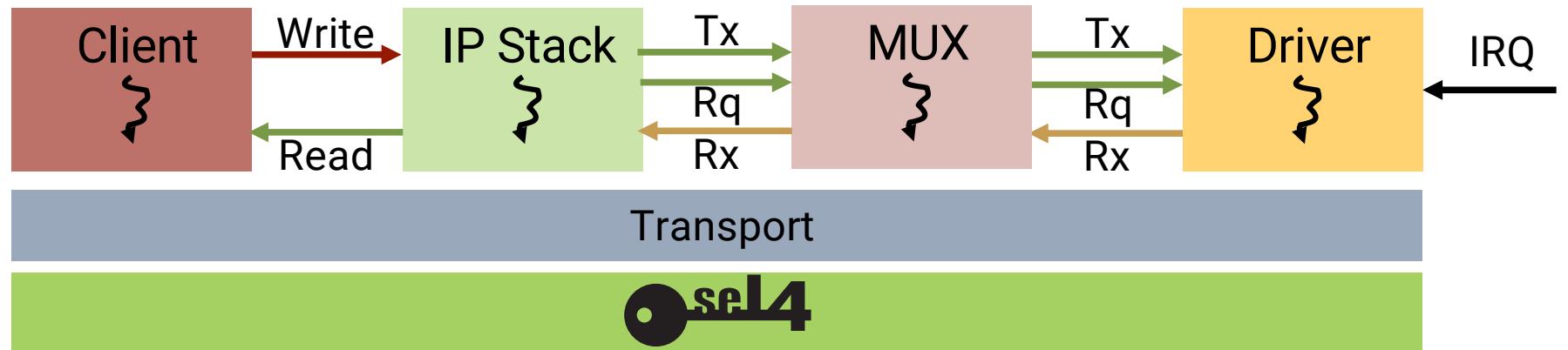
Challenge:

- Performance

Opportunities:

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

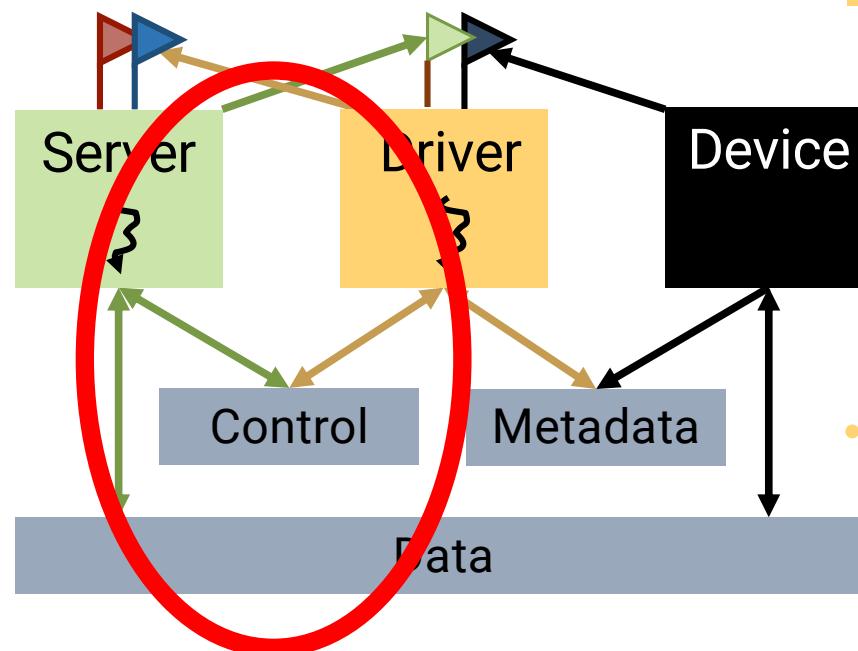
Enable verification?



Driver Model

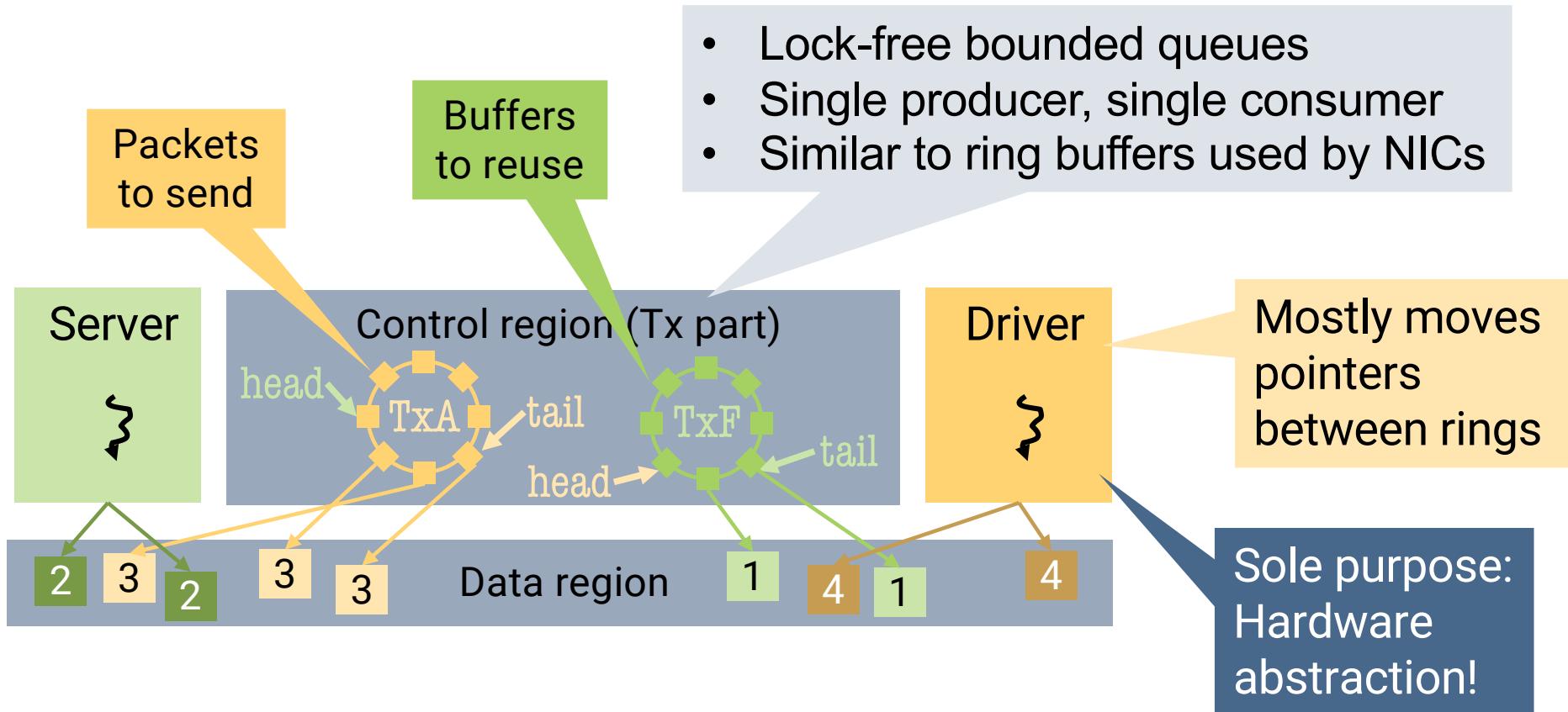
Driver model:

- Single-threaded
- Event-driven
- Simple!



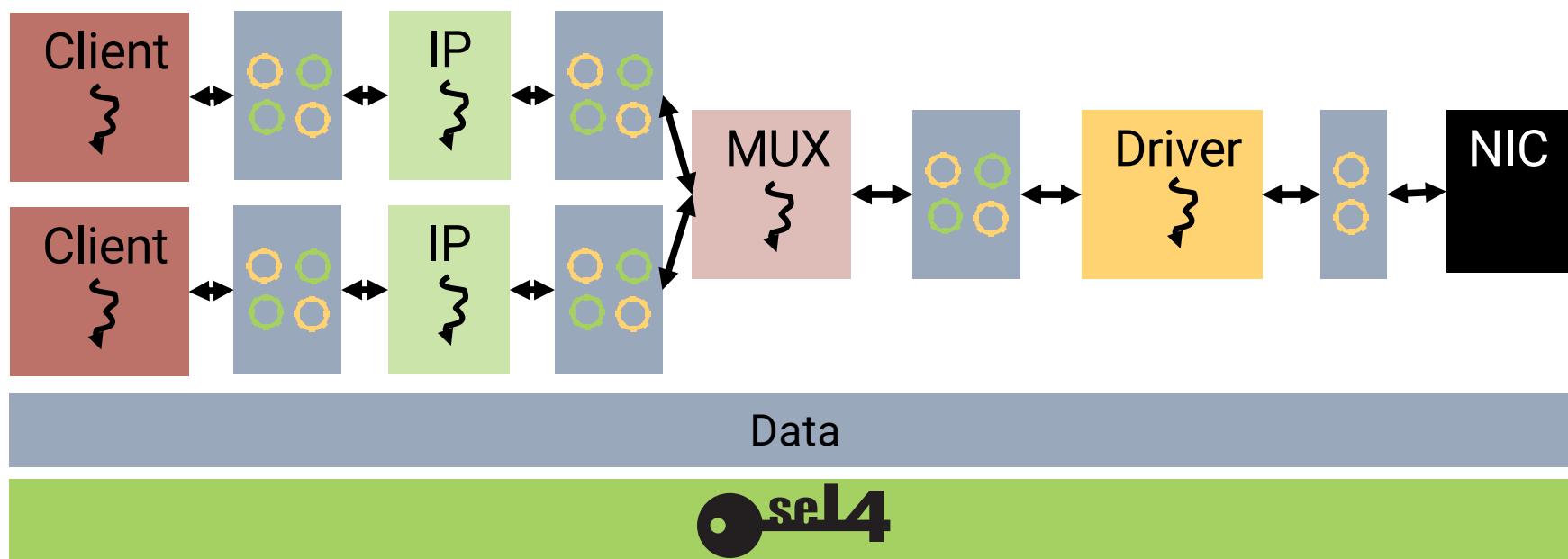
Can we get
this to
perform?

Transport Layer

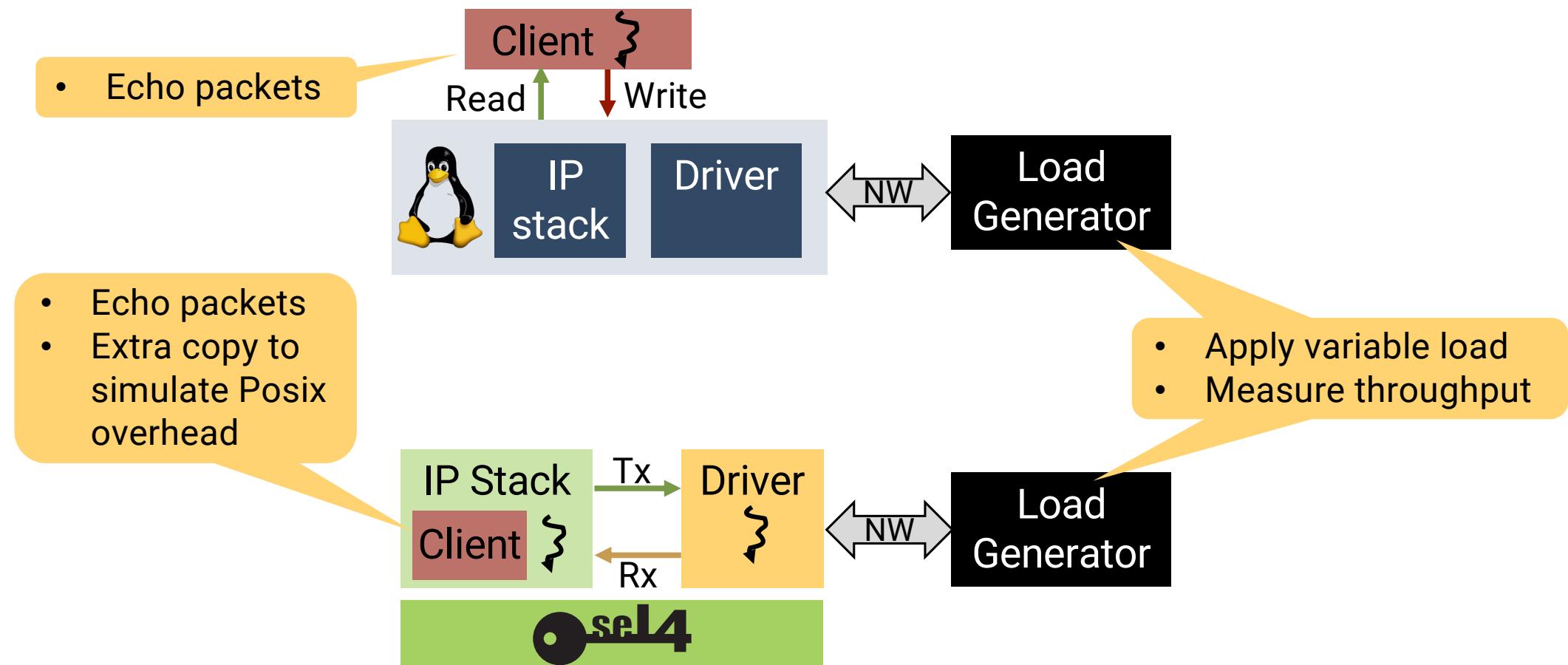


Transport Architecture Scales

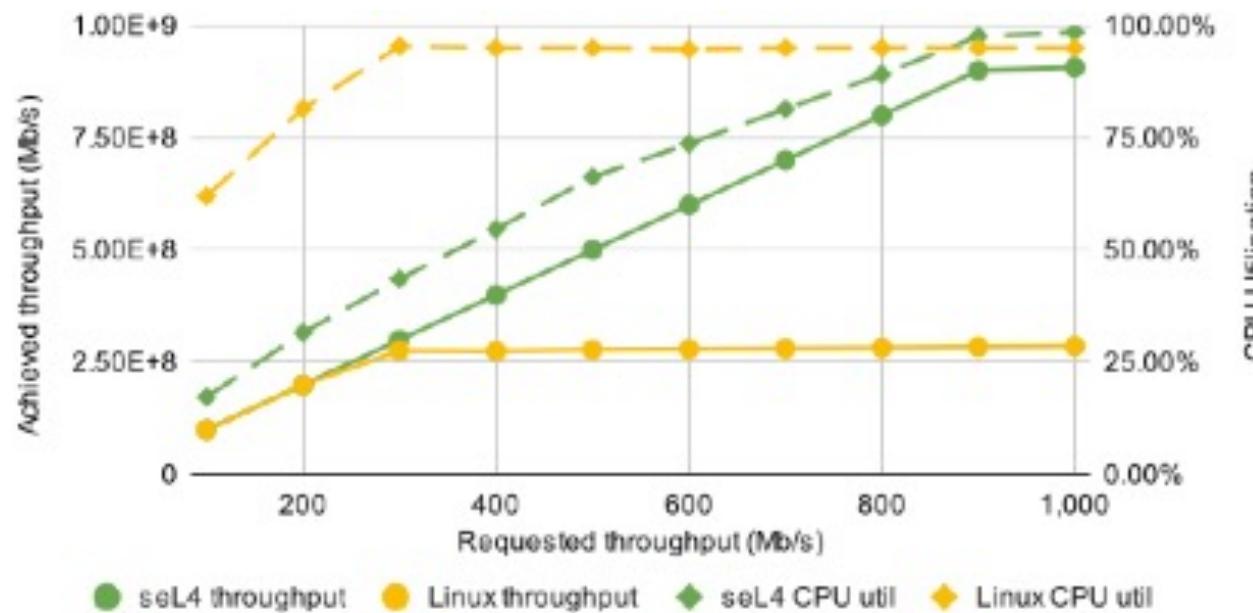
- Components can be on separate cores
- Driver, MUX close to minimal critical sections
- Should scale well without locks!



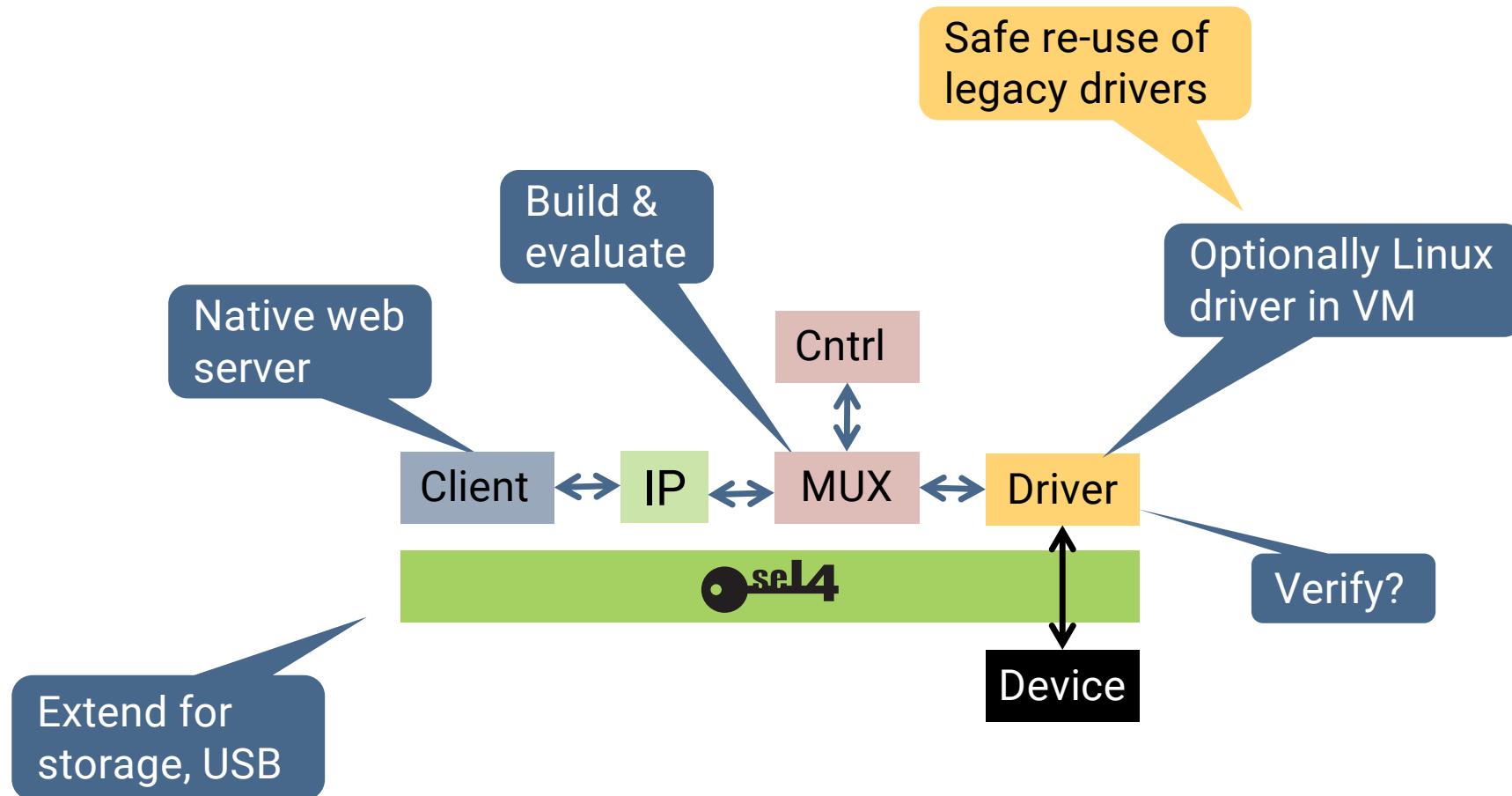
Preliminary Evaluation: Setup



Preliminary Evaluation: Performance



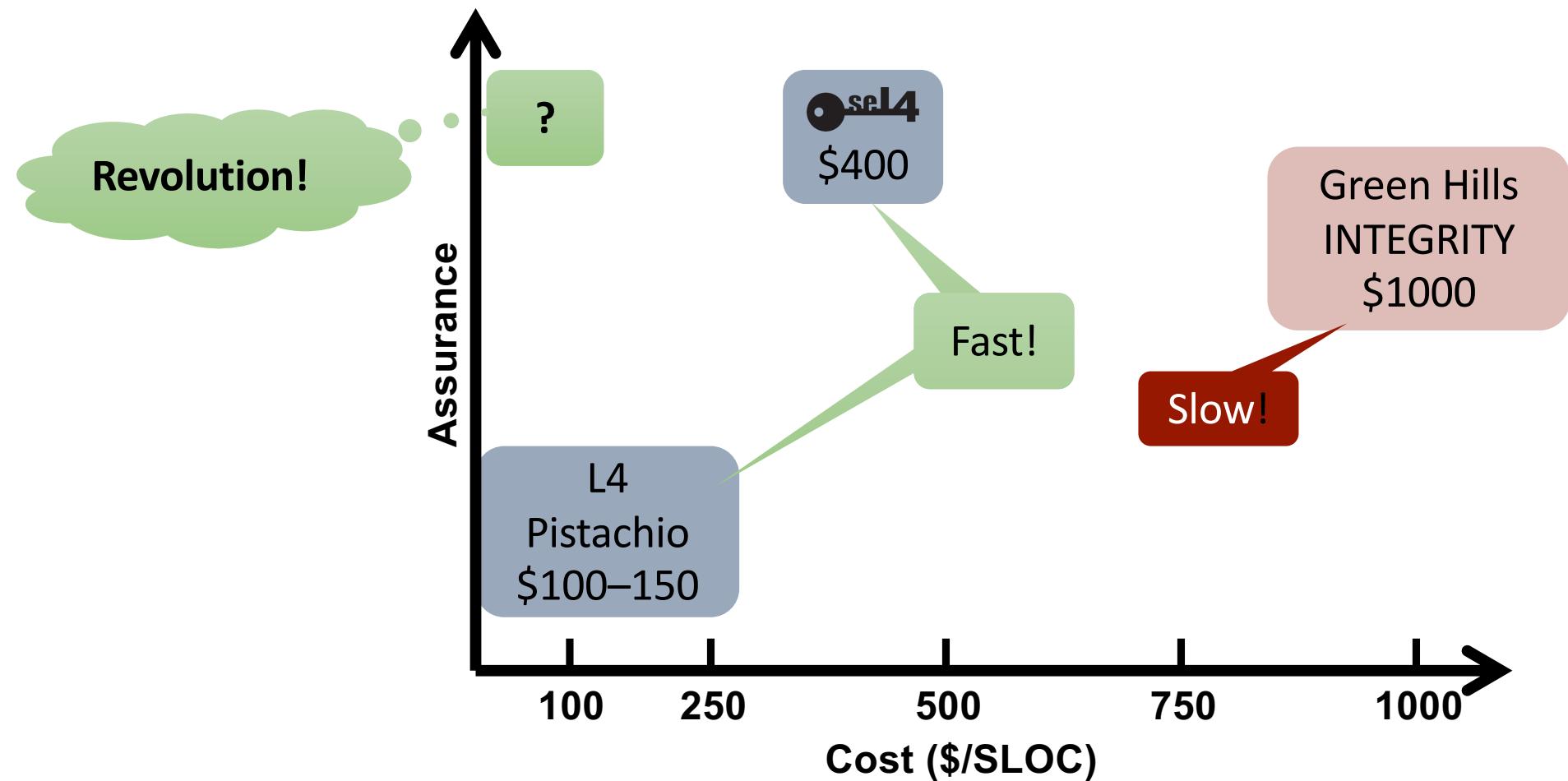
sDDF: Next Steps



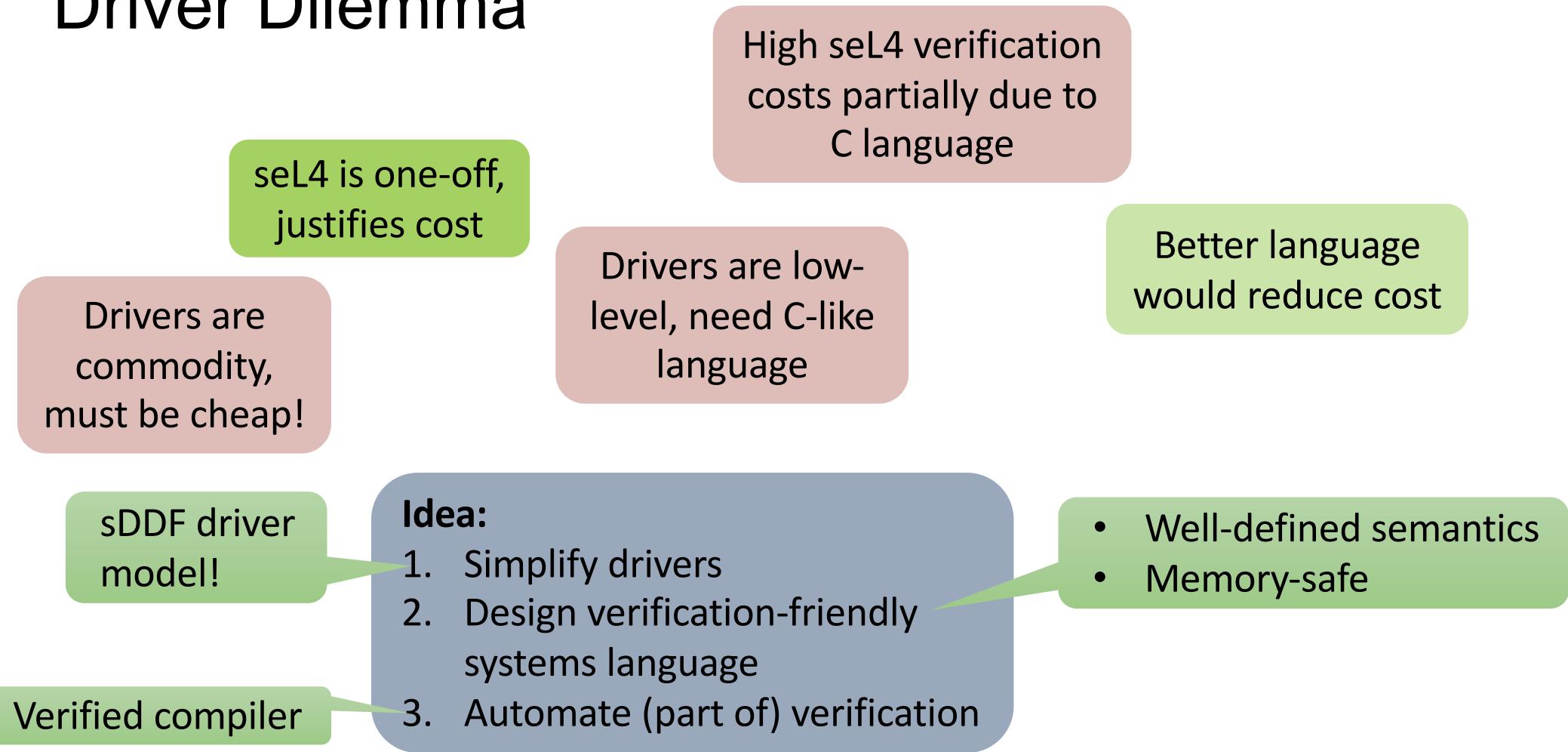
seL4-Related Research in TS

Verifying Device Drivers?

Remember: Verification Cost in Context



Driver Dilemma



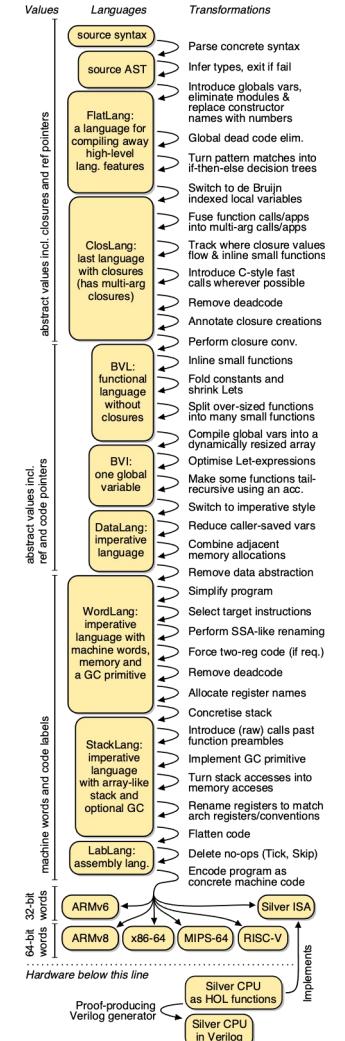
CakeML: Verified Implementation of ML



Re-use framework for
new systems
language: Pancake

- ✓ 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

<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

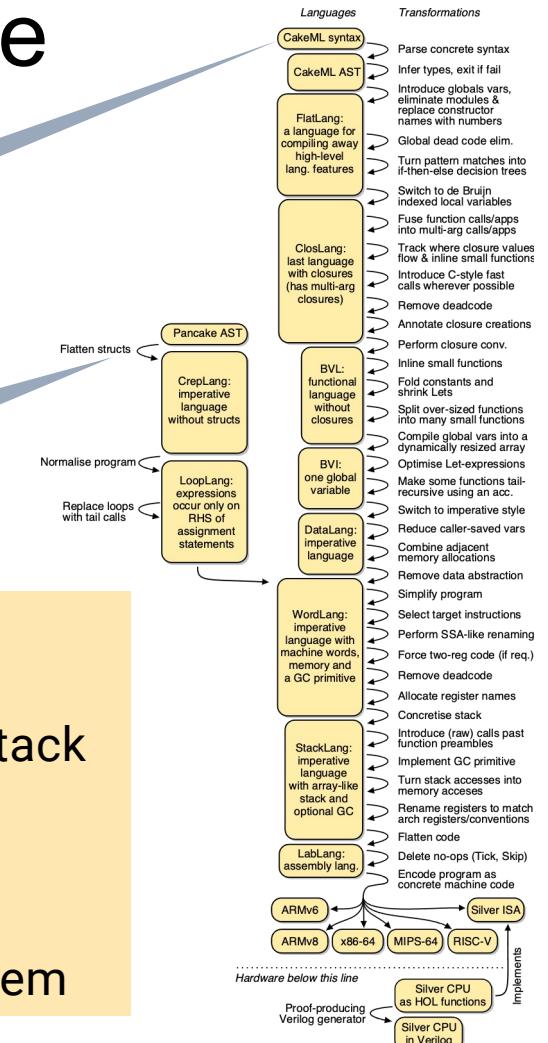
In progress

CakeML

Pancake

Approach:

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

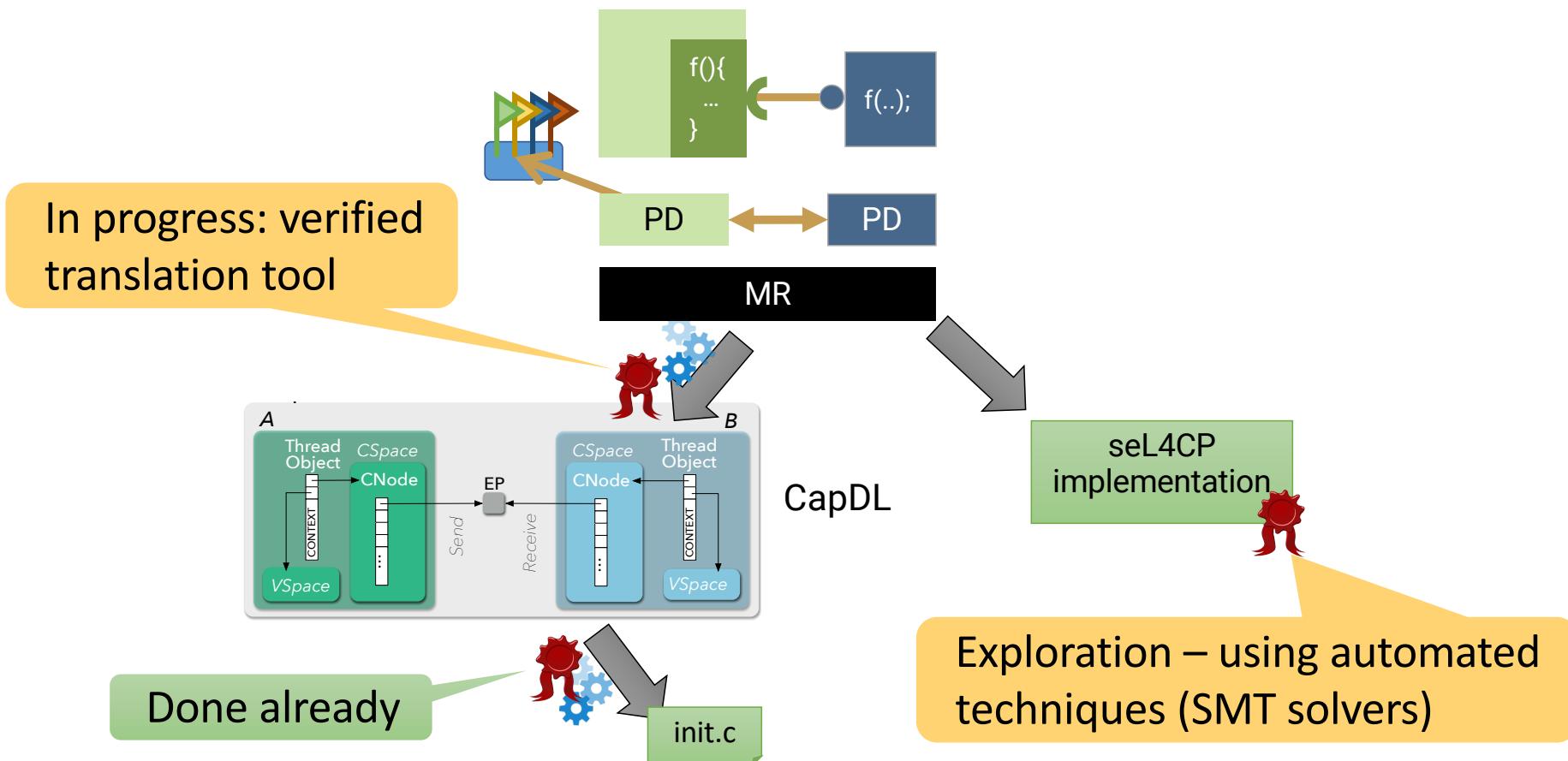


seL4-Related Research in TS

Verifying the seL4 Core Platform



seL4CP Verification



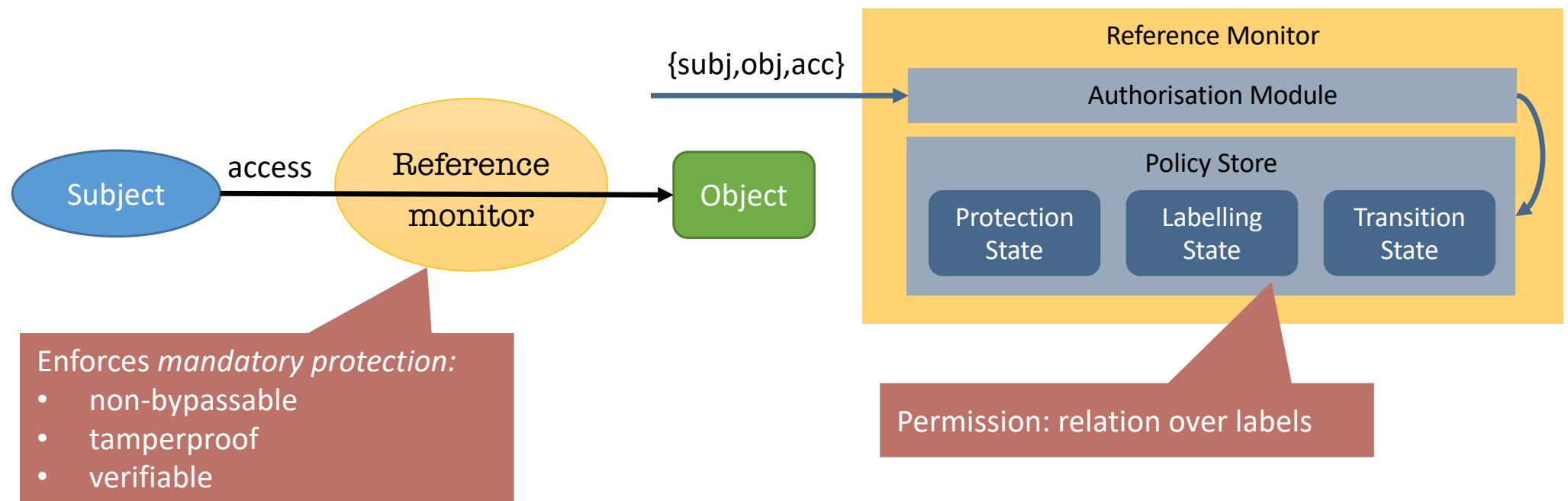
seL4-Related Research in TS

Secure Multi-Server OS

Recap: Secure Operating Systems

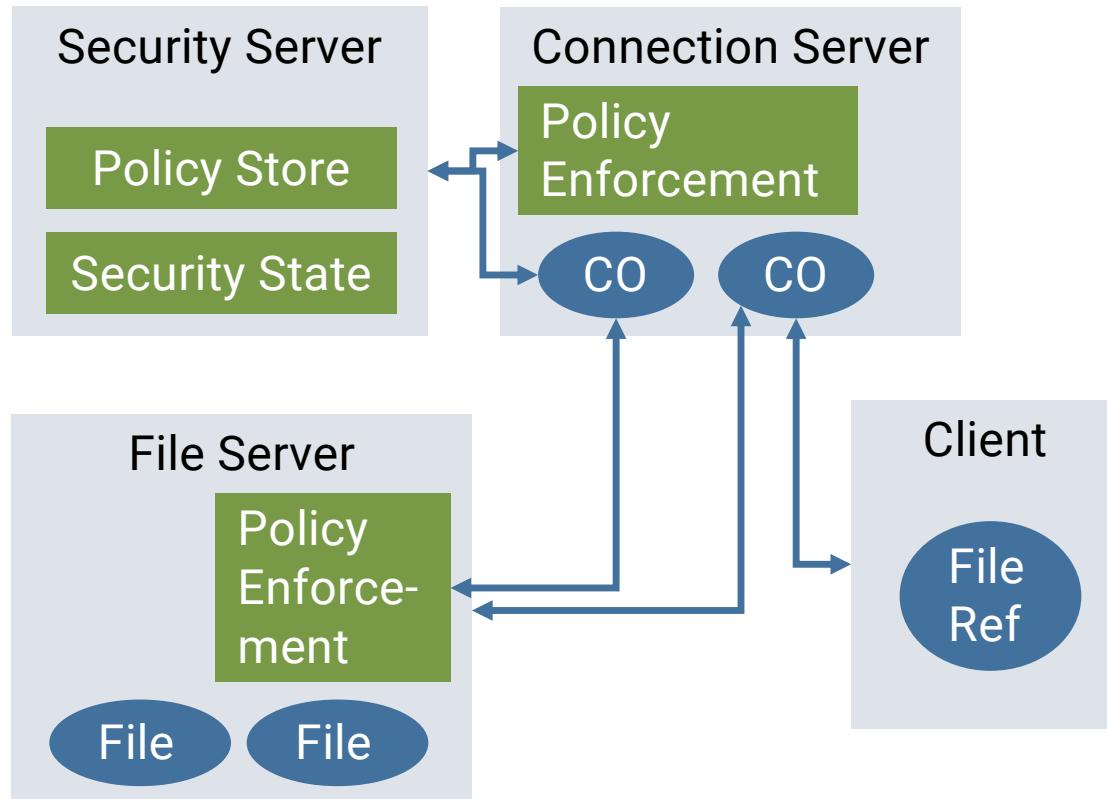
Secure OS: [Jaeger: OS Security]

Access enforcement satisfies the *reference monitor* concept





Secure, General-Purpose OS



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

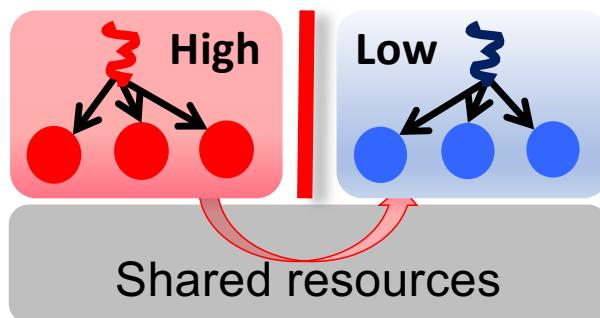
Requires:

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

seL4-Related Research in TS

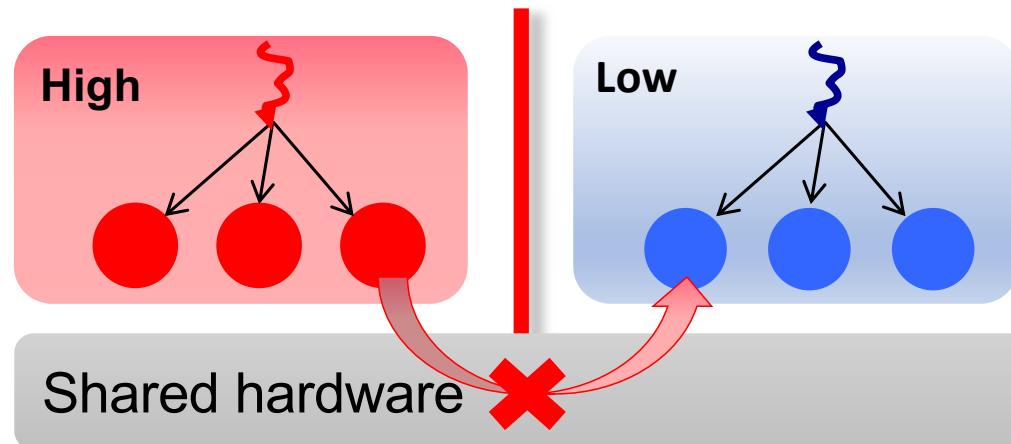
Time Protection: Verified Prevention of Microarchitectural Timing Channels

Refresh: Microarchitectural Timing Channels



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

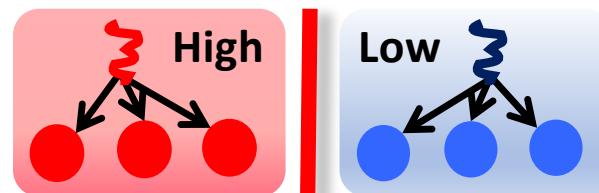
OS Must Enforce *Time Protection*



Preventing interference is core duty of the OS!

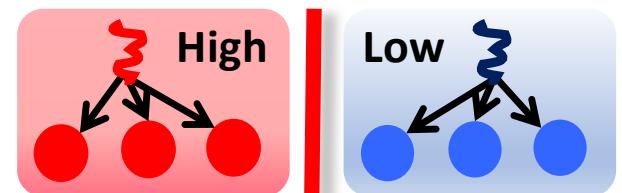
- *Memory protection* is well established
- *Time protection* is completely absent

Time Protection: No Sharing of HW State



Temporally
partition

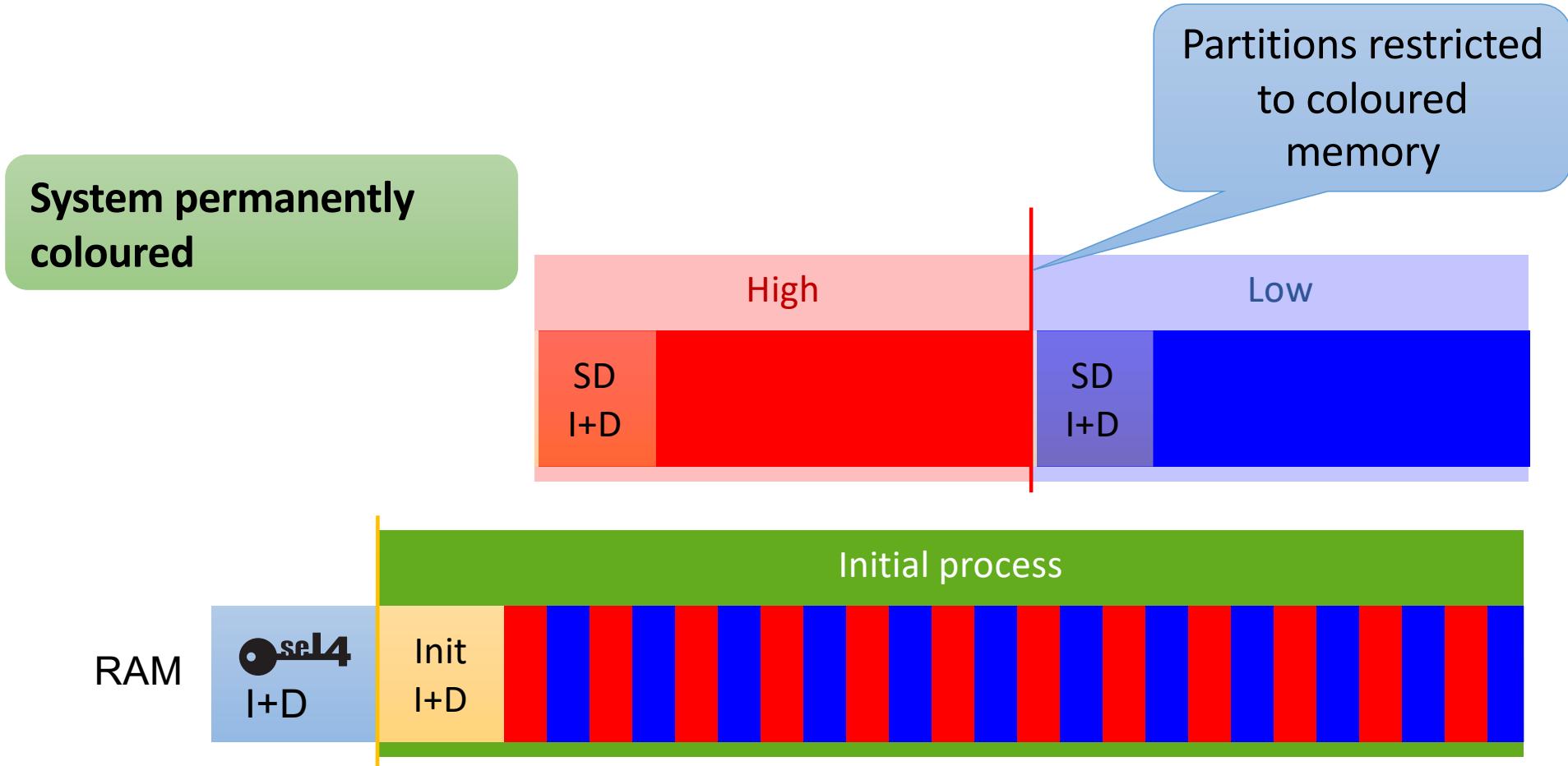
Flush



What are the OS
mechanisms?

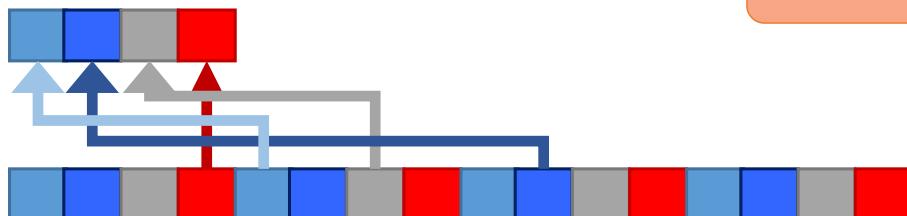
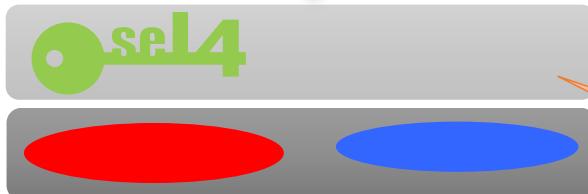
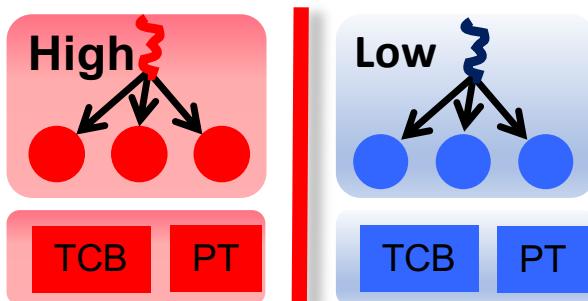


Spatial Partitioning: Cache Colouring





Spatial Partitioning: Cache Colouring

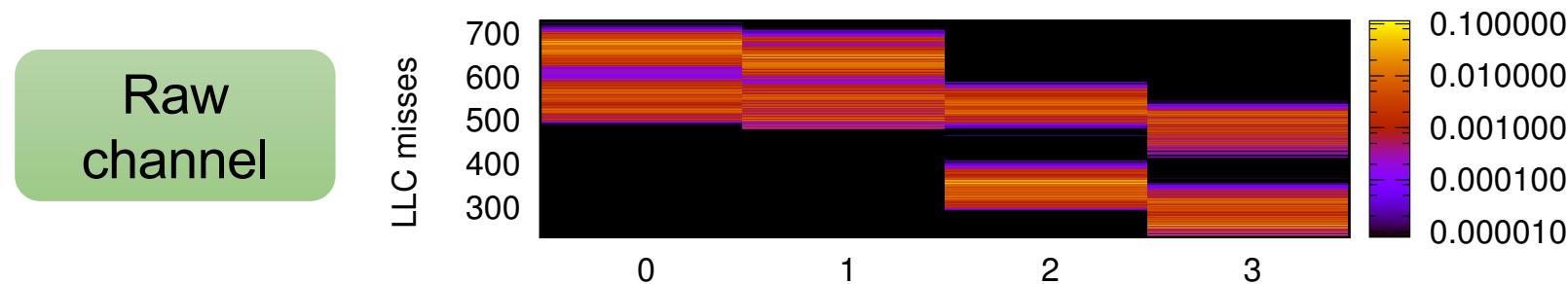


- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
⇒ colouring userland colours kernel memory

Shared kernel image



Channel Through Kernel Code



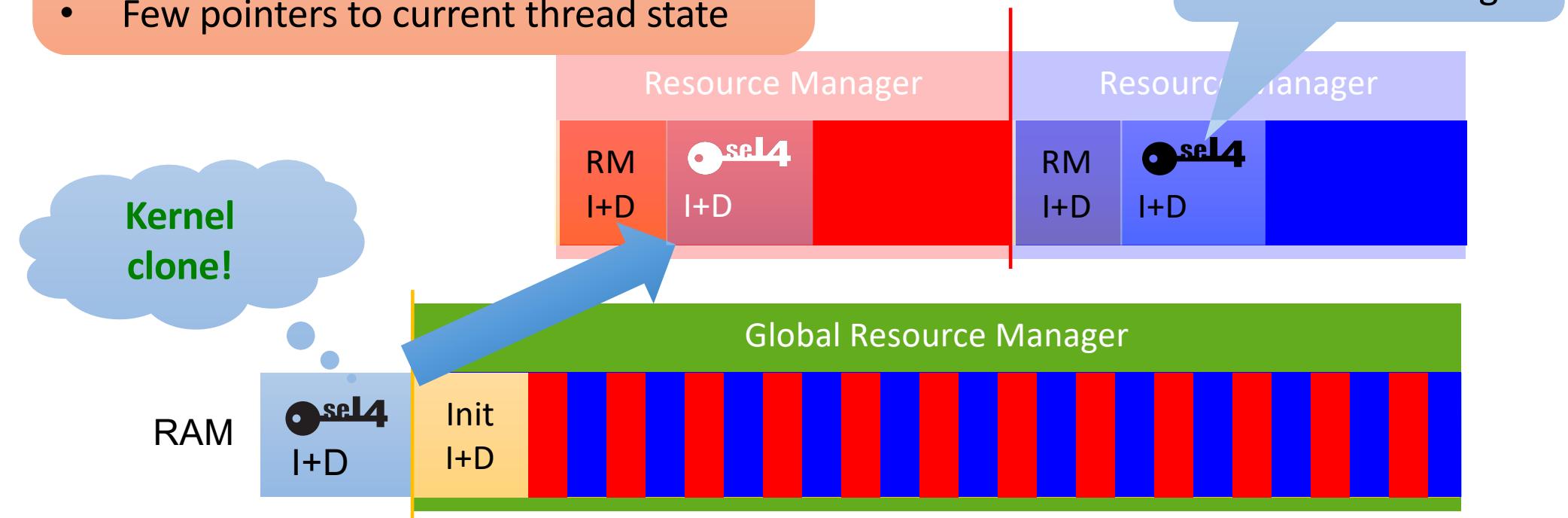
Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)

• sel4 Colouring the Kernel

Remaining shared kernel data:

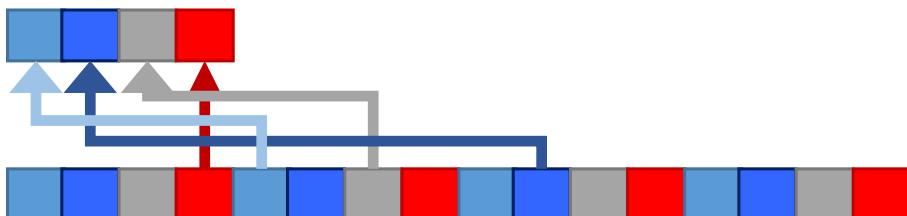
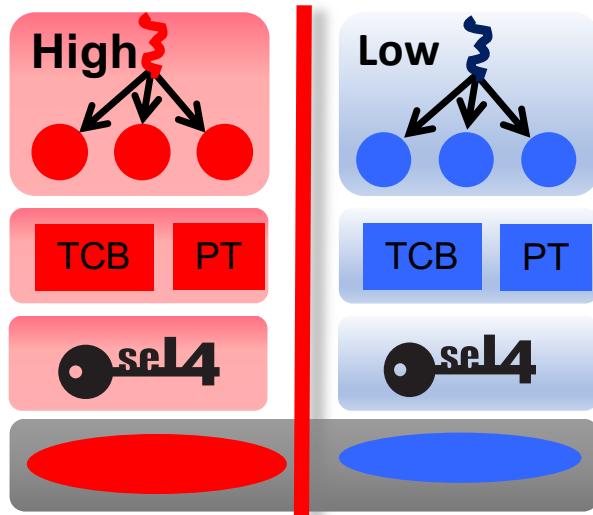
- Scheduler queue array & bitmap
- Few pointers to current thread state

Each partition has own kernel image





Spatial Partitioning: Cache Colouring



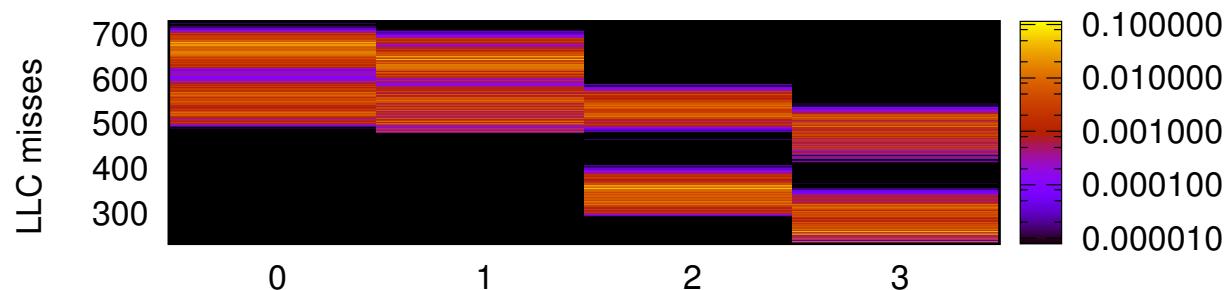
- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
⇒ colouring userland colours kernel memory
- Per-partition kernel image to colour kernel

Must ensure deterministic access to remaining shared kernel state!

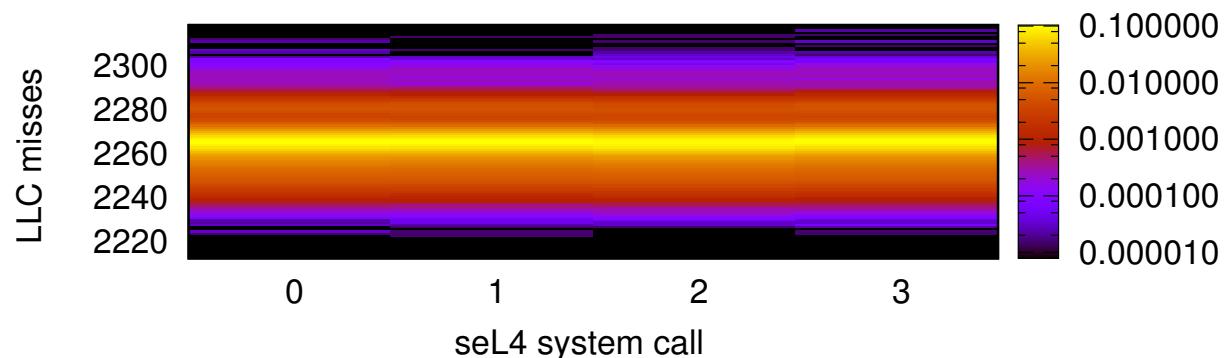


Channel Through Kernel Code

Raw
channel



Channel with
cloned kernel





Temporal Partitioning: Flush on Switch

Must remove any
history dependence!

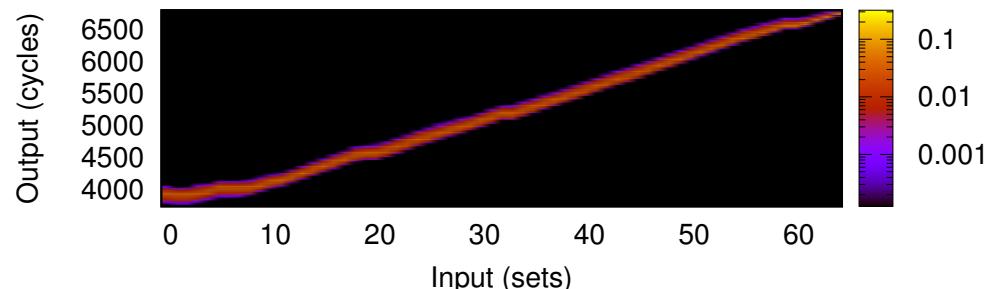
2. Switch user context
3. **Flush on-core state**

6. Reprogram timer
7. return

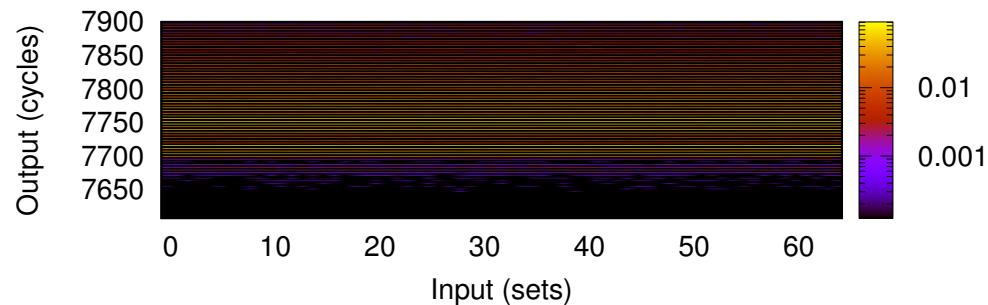


D-Cache Channel

Raw
channel



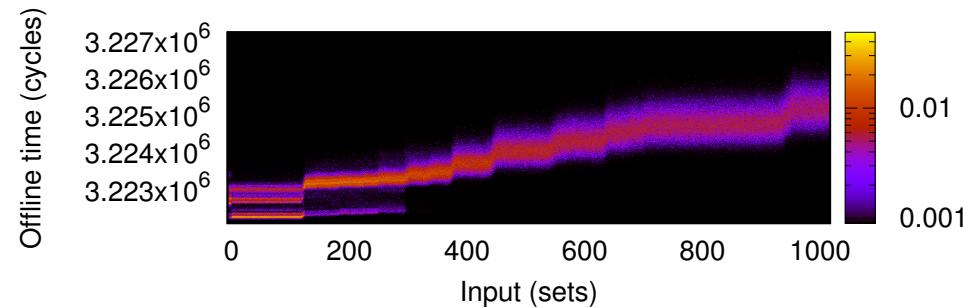
Channel with
flushing





Flush-Time Channel

Raw
channel





Temporal Partitioning: Flush on Switch

Must remove any history dependence!

1. $T_0 = \text{current_time}()$
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. $\text{while } (T_0 + \text{WCET} < \text{current_time}()) ;$
6. Reprogram timer
7. return

Latency depends on prior execution!

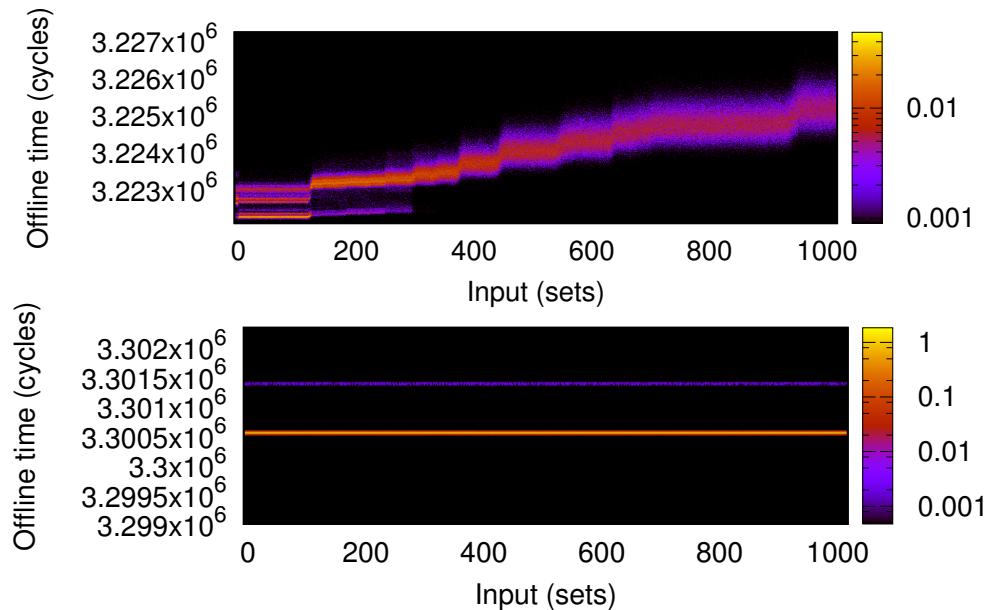
Time padding to remove dependency

Ensure deterministic execution



Flush-Time Channel

Raw
channel

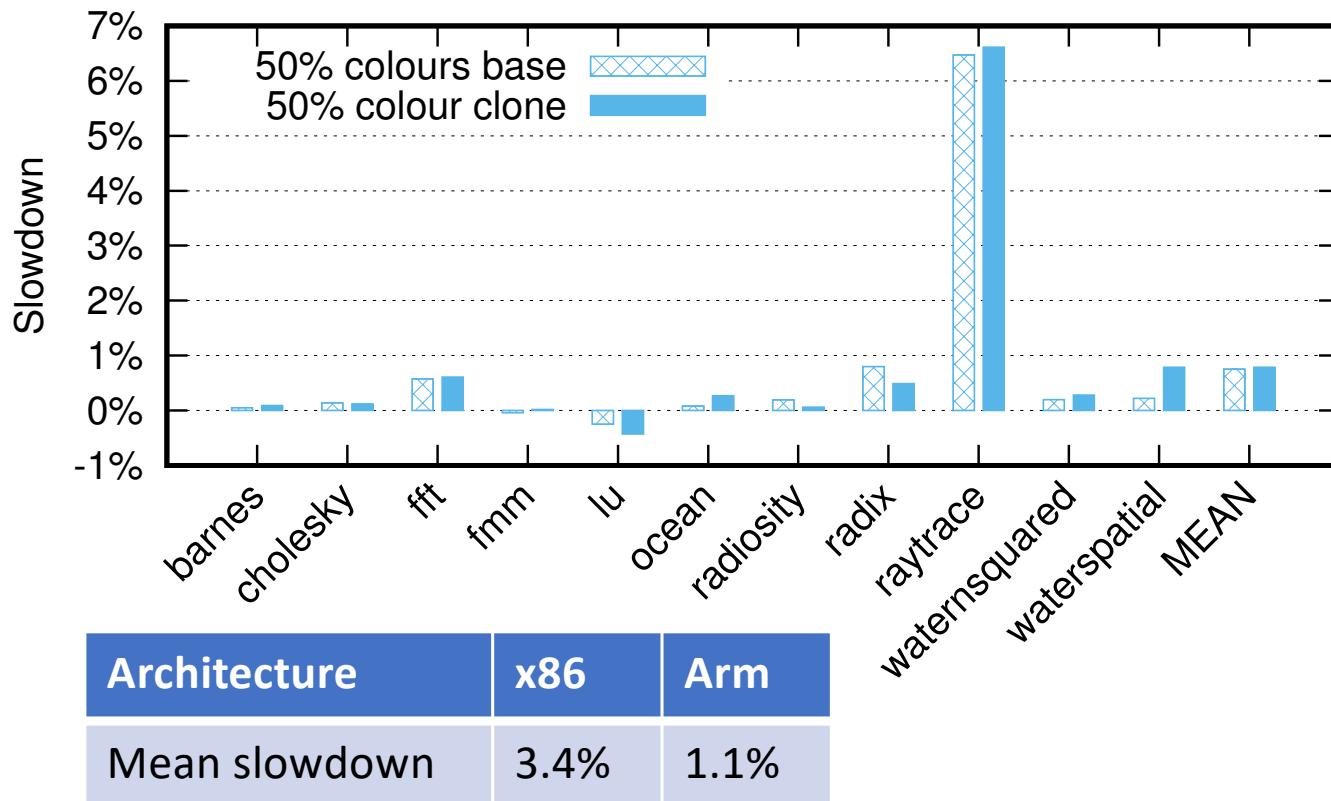


Channel with
deterministic
flushing



Performance Impact of Colouring

Splash-2 benchmarks on Arm A9



- 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

A New HW/SW Contract

For all shared microarchitectural resources:

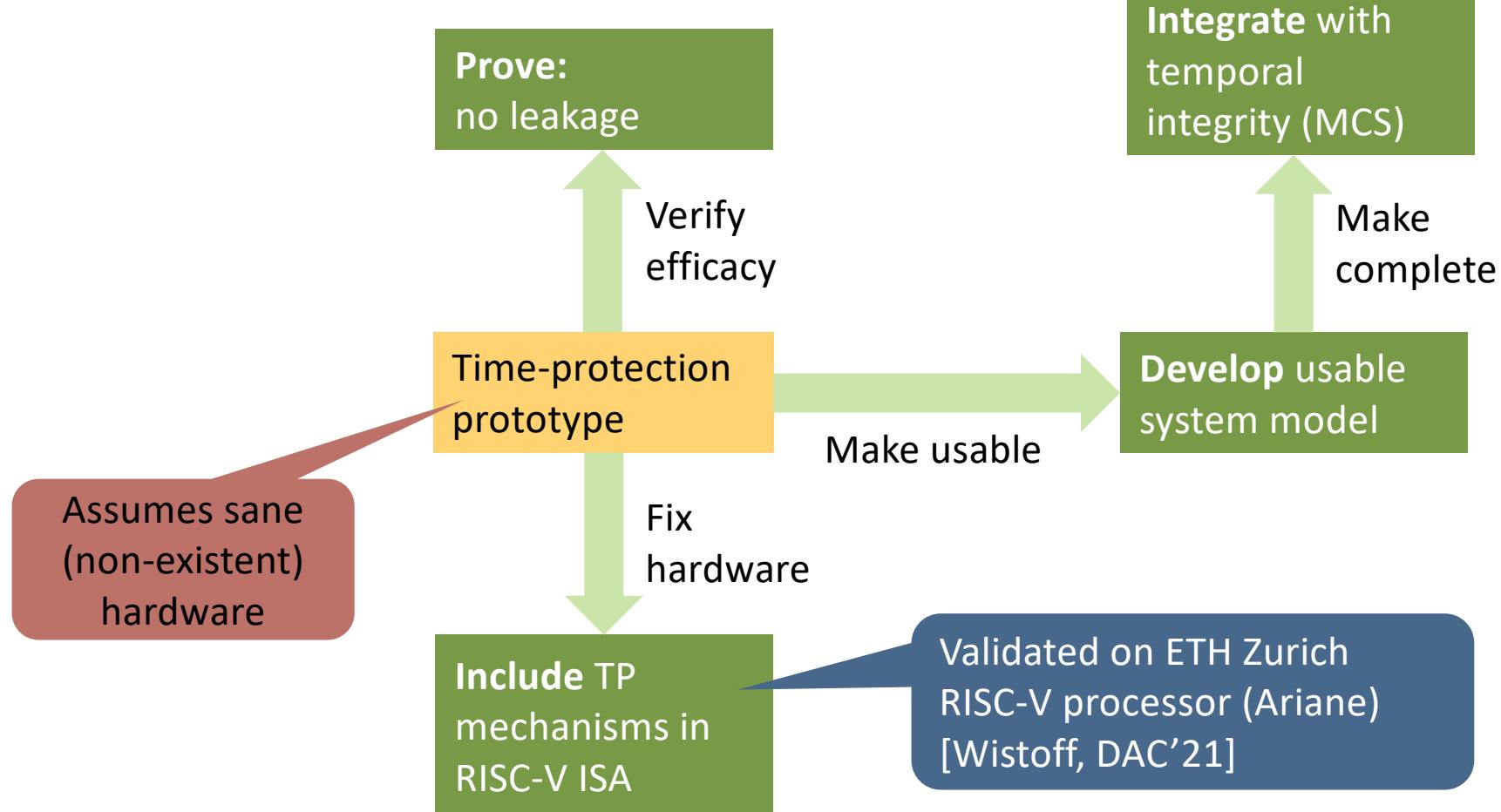
aISA: augmented ISA

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