School of Computer Science & Engineering

Trustworthy Systems Group

It's Time For Secure Operating Systems

Gernot Heiser

gernot@unsw.edu.au
@microkerneldude.bsky.social
https://microkerneldude.org/

2003

PSOS Revisited

1 Historical Introduction

Many of the characteristic design flaws still common in today's systems were essentially avoided by the methodology and the specification language. Although some simple illustrative proofs were carried out, it would be a incorrect to say that PSOS was a *proven* secure operating system. Nevertheless, the approach clearly demonstrates how properties such as security could be formally proven — in the sense that the specification could be formally consistent with the requirements, the source code could be formally consistent with the specifications, and the compiler could be proven correct as well.

Operating Systems

R. Stockton Gaines, Editor

Specification and Verification of the UCLA Unix† Security Kernel

Bruce J. Walker,
Gerald J. Popek
University of Ca

Our research

operating system can be shown data secure, meaning that direct access to data must be possible only if the recorded protection policy permits it. The two major components

Data Secure Unix
tem, was constructed
UCLA to develop procedures by which operating systems can be produced and shown secure. Program verification methods were extensively applied as a constructive means of demonstrating security enforcement.

Here we report the specification and verification experience in producing a secure operating system. The work represents a significant attempt to verify a large-scale, production level software system, including all aspects from initial specification to verification of implemented code.

- ’70s optimism
- ‘90s disillusionment

Communications of the ACM

February 1980
Volume 23
Number 2

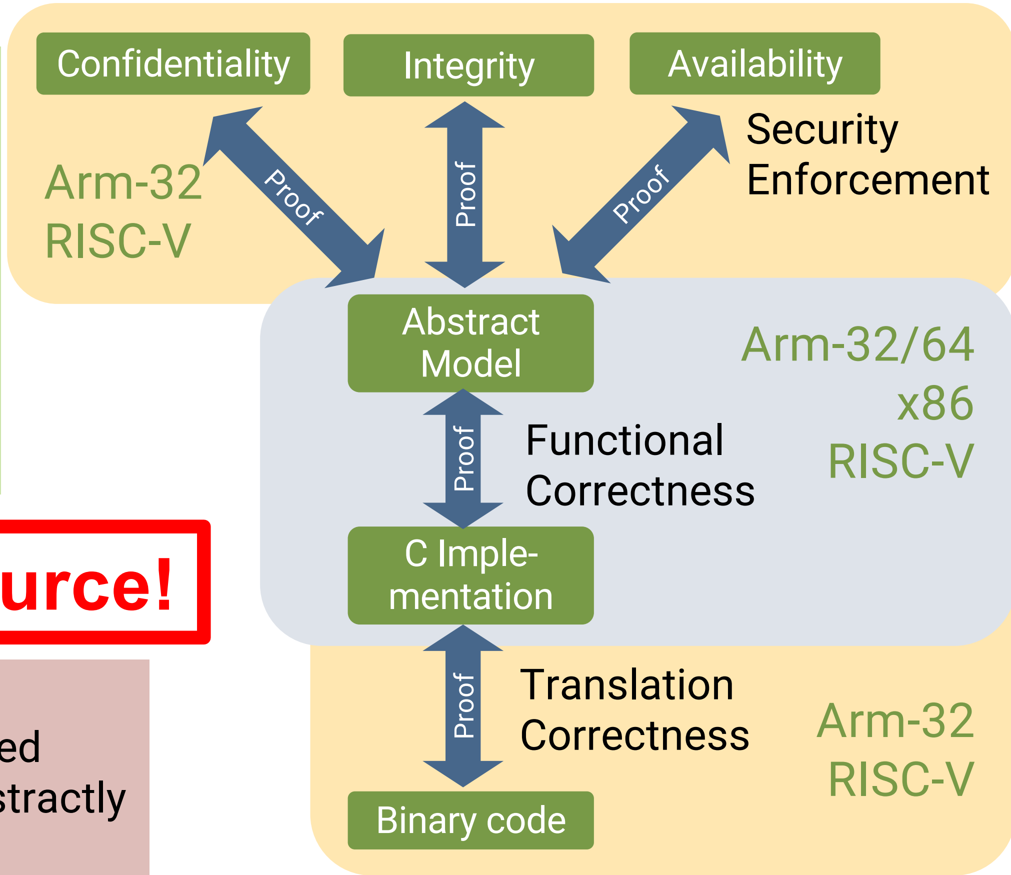
UNSW SYDNEY

2009: Verification of a Microkernel

- World's first OS kernel with correctness proof
- Most comprehensive verification
- Only verified OS with capability-based fine-grained protection
- Only protected-mode RTOS with sound and compete WCET analysis

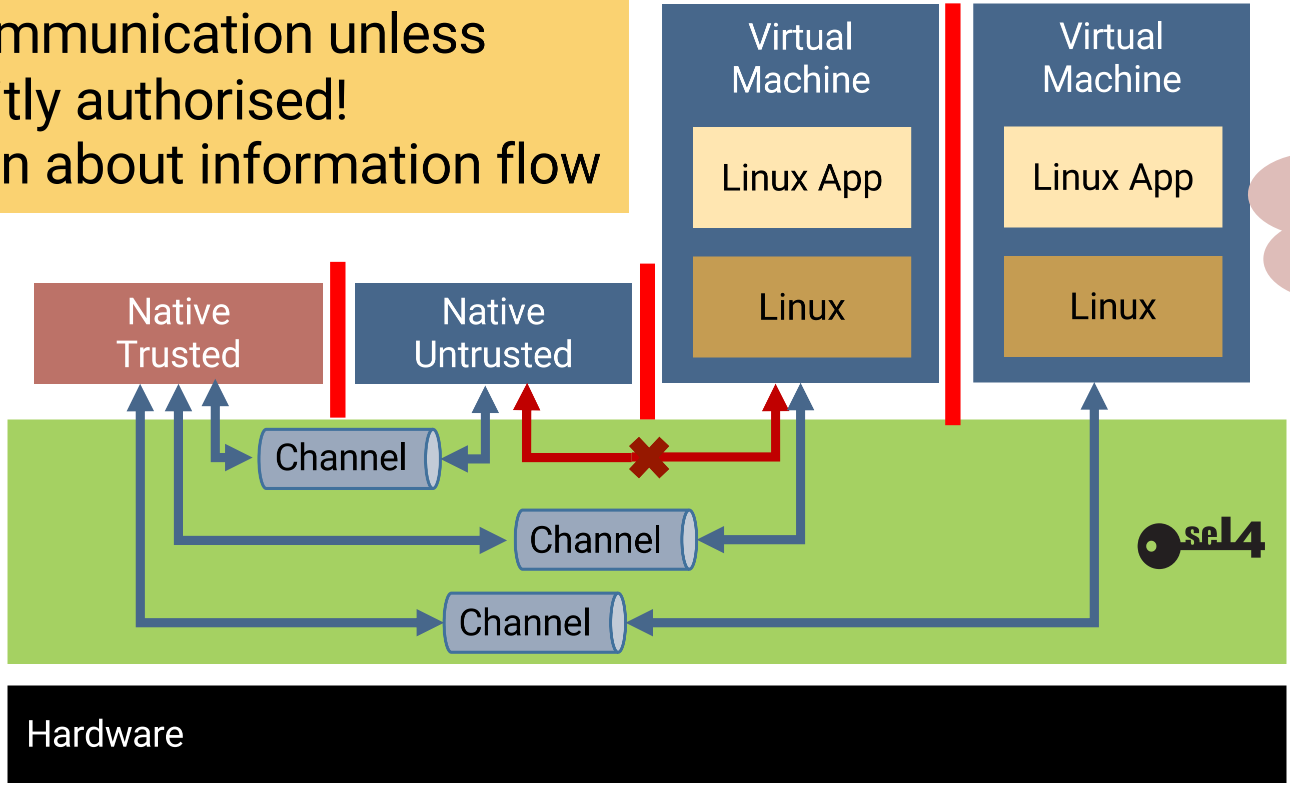
Open Source!

Present limitations
- Initialisation code not verified
- MMU, caches modelled abstractly
- Multicore not yet verified



Capabilities: Fine-Grained Protection

- Enforce *least privilege*
- No communication unless explicitly authorised!
- Reason about information flow



No capabilities? You're not serious about security!

The Benchmark for Performance

Smaller is better

Round-trip cross-address-space IPC on 64-bit Intel Skylake

	seL4	Fiasco.OC aka L4Re	Google Zircon
Latency (cycles)	986	2717	8157
Mandatory HW cost* (cycles)	790	790	790
Overhead absolute (cycles)	196	1972	7367
Overhead relative	25%	240%	930%

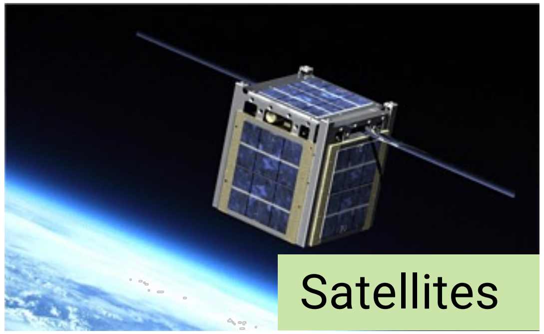
***:** The Cost of SYCALL + 2 × SWAPGS + SYSRET = 395 cycles, times 2 for round-trip

Source:

Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, Haibo Chen: "SkyBridge: Fast and Secure Inter-Process Communication for Microkernels", EuroSys, April 2019

Used in Real-World Systems

Autonomous vehicles



Satellites

Critical infrastructure protection

Secure communication device
In use in multiple defense forces

Cars

seL4 "World's Most Secure Drone"

Tweet

DARPA
@DARPA

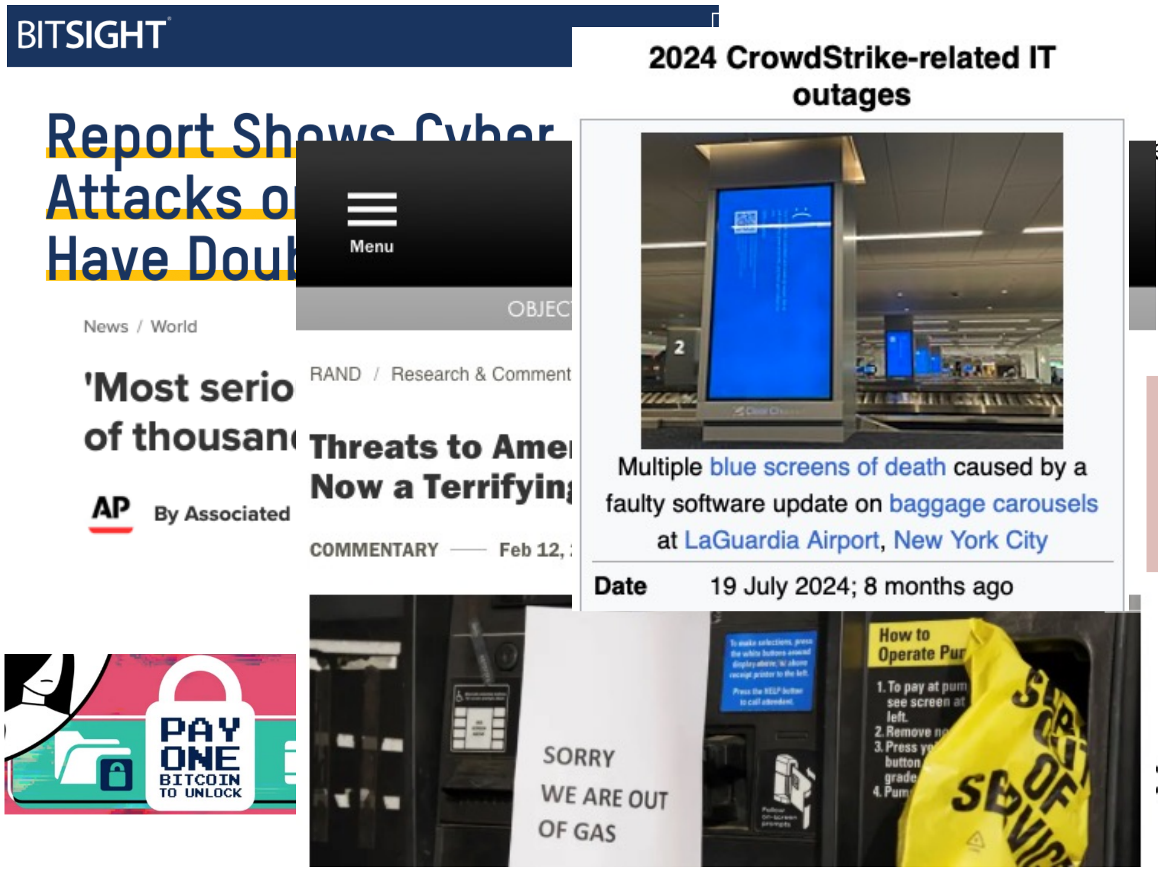
We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!

DEFCON'22

seL4 Timeline

- July'09: Proof of implementation correctness (Arm-32)
- Aug'11: Proof of integrity enforcement
- Nov'11: Sound worst-case execution-time analysis
- May'13: Proof of confidentiality enforcement
- Jun'13: Proof of compilation correctness
- Jul'14: **seL4 open-sourced (GPL)**
- 2012–17: DARPA HACMS: seL4 in real-world systems
- 2018: x86 verification
- Jun'20: RISC-V verification
- Mar'24: Arm-64 verification
- Sep'24: Commercial electric car

Yet Security Failures Are Everywhere



Cyberattacks on Automated Vehicles Rise by 99%: Report

By CISOMAG - June 9, 2020

Increasingly used by
- organised crime
- state actors

:auses delay at Zurich Airport

Why Still No Secure OS?

A Microkernel

Microkernel:
- OS code that must execute in privileged mode
- Everything else belongs in user mode servers
- Servers are subject to the microkernel's security enforcement!

Assembly language of operating systems

Consequence:
- Small: 10 kLOC
- Only fundamental, policy-free mechanisms
- No application-oriented services/abstractions
- **BYO file system, memory manager, device drivers**

Leave to community/ industry to build

seL4 Experience of the First 10+ Years

seL4's assurance and power are (still!) unrivalled

TS contributed poor designs too!

Good design on seL4 requires deep expertise

Arcane build system didn't help!

Rare beyond TS and ex-TSers

The world needs an OS that is:
- **secure**
- easy to use
- open source

Community did not deliver a secure OS!

UNSW SYDNEY

LionsOS

Stop The Train Wrecks!

UNSW SYDNEY

LionsOS Aims

Aim 1:
Practical, easy-to-use, open-source OS for wide range of *embedded/IoT/cyberphysical* use cases

Can use static architecture

Must be well designed!

Aim 2:
Uncompromising performance

Aim 3:
Most secure OS ever

Must be verified!

Overarching Design Principle: KISS!

LionsOS is what Posix/Linux isn't!

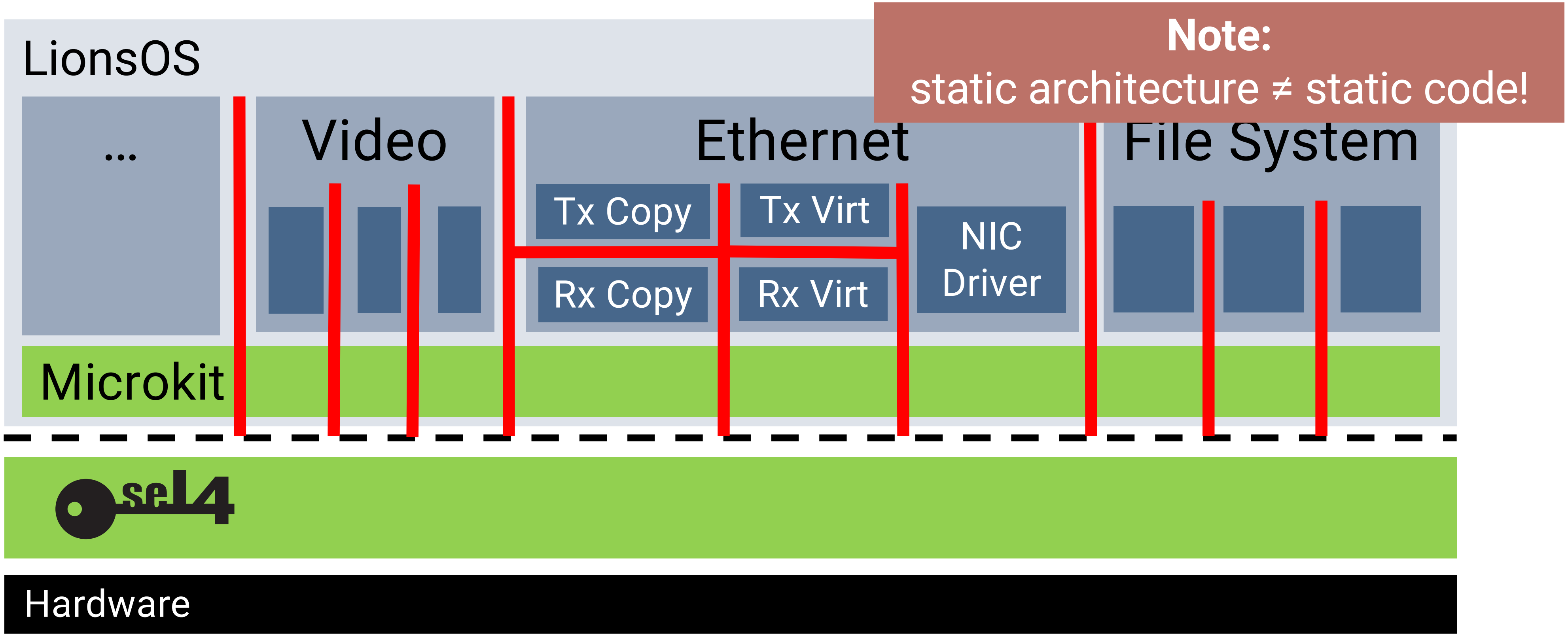
Helps development **and** verification!

Radical simplicity:
- fine-grained modularity, strict separation of concerns
- event-driven programming model
- use-case-specific policies

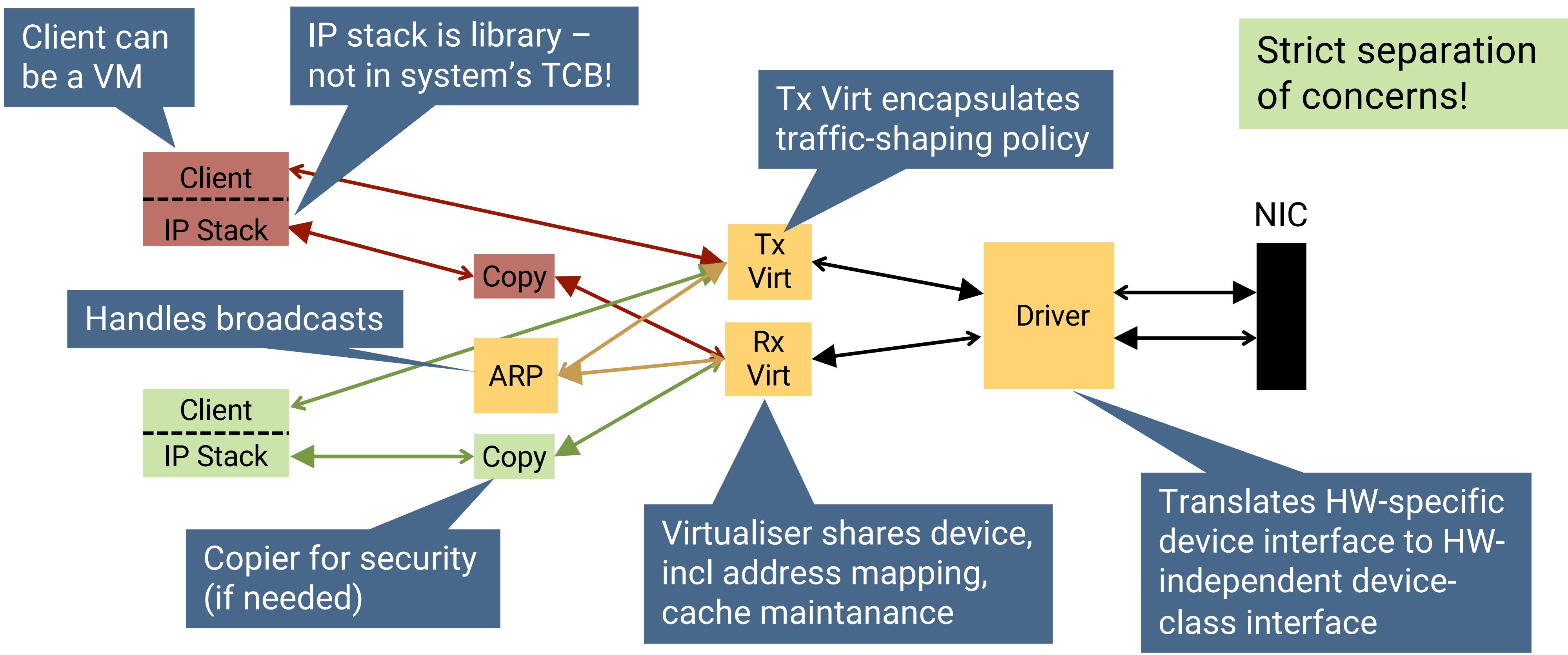
… but we'll have Posix-like I/O wrappers

Use-case diversity by replacing components

LionsOS: Highly Modular System



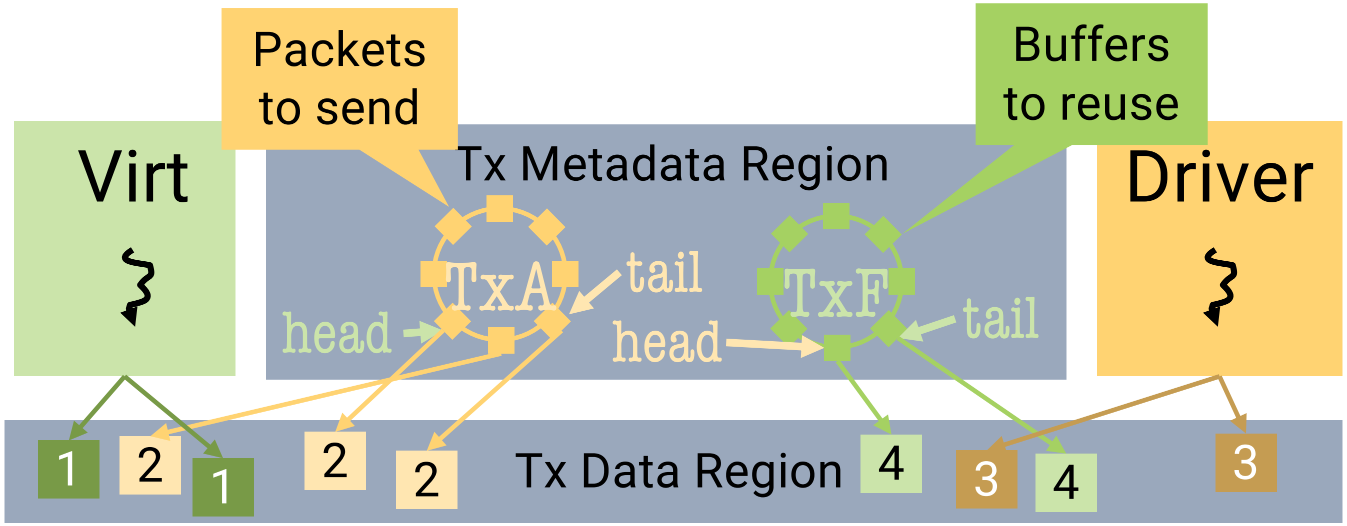
Example: Networking Subsystem



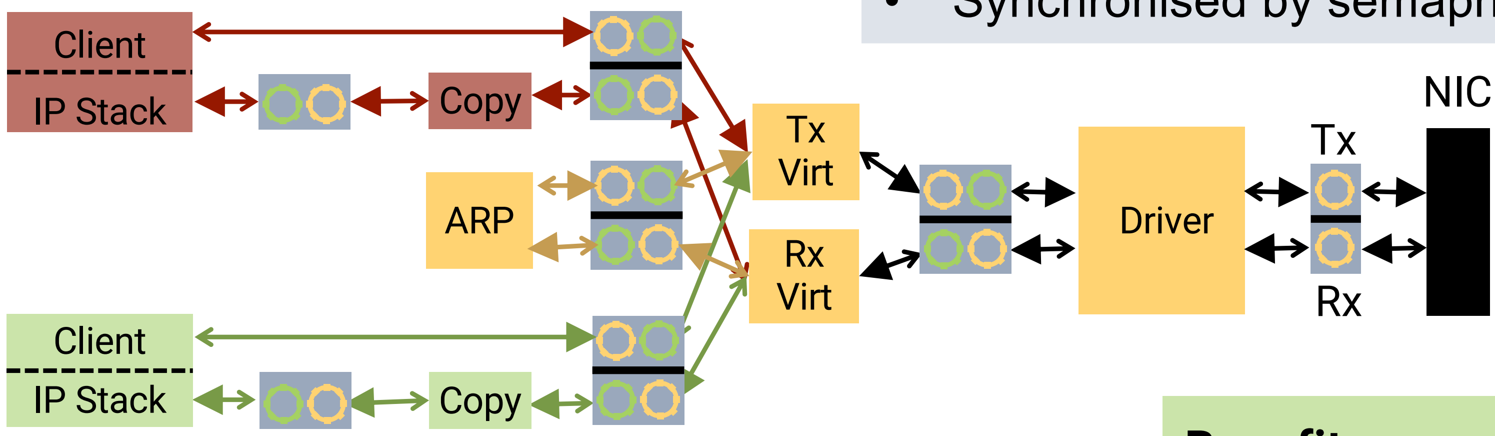
Zero-copy Data Transfer

Components are single-threaded – "Tamed" concurrency!

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



Networking Detail



Zero-copy communication:

- Lock-free, single-producer, single-consumer, bounded queues
- Synchronised by semaphores

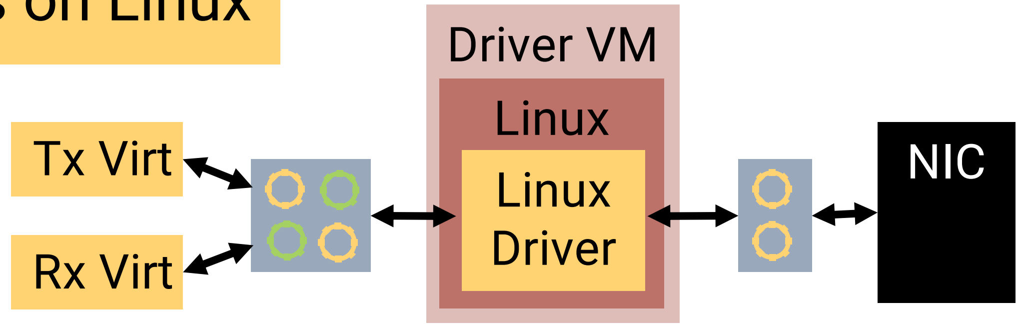
Benefits:

- simple components
- **location transparency**
- **suitable for verification**

Legacy Re-use: Driver VMs

Can re-use unmodified Linux drivers:

- Transparently use driver VM instead of native driver
- Linux app in VM uses UIO to communicate with in-kernel driver
- develop LionsOS components on Linux



Comparison to Linux on i.MX8M

Linux:
- NW driver: 3k lines
- NW system total: 1M lines

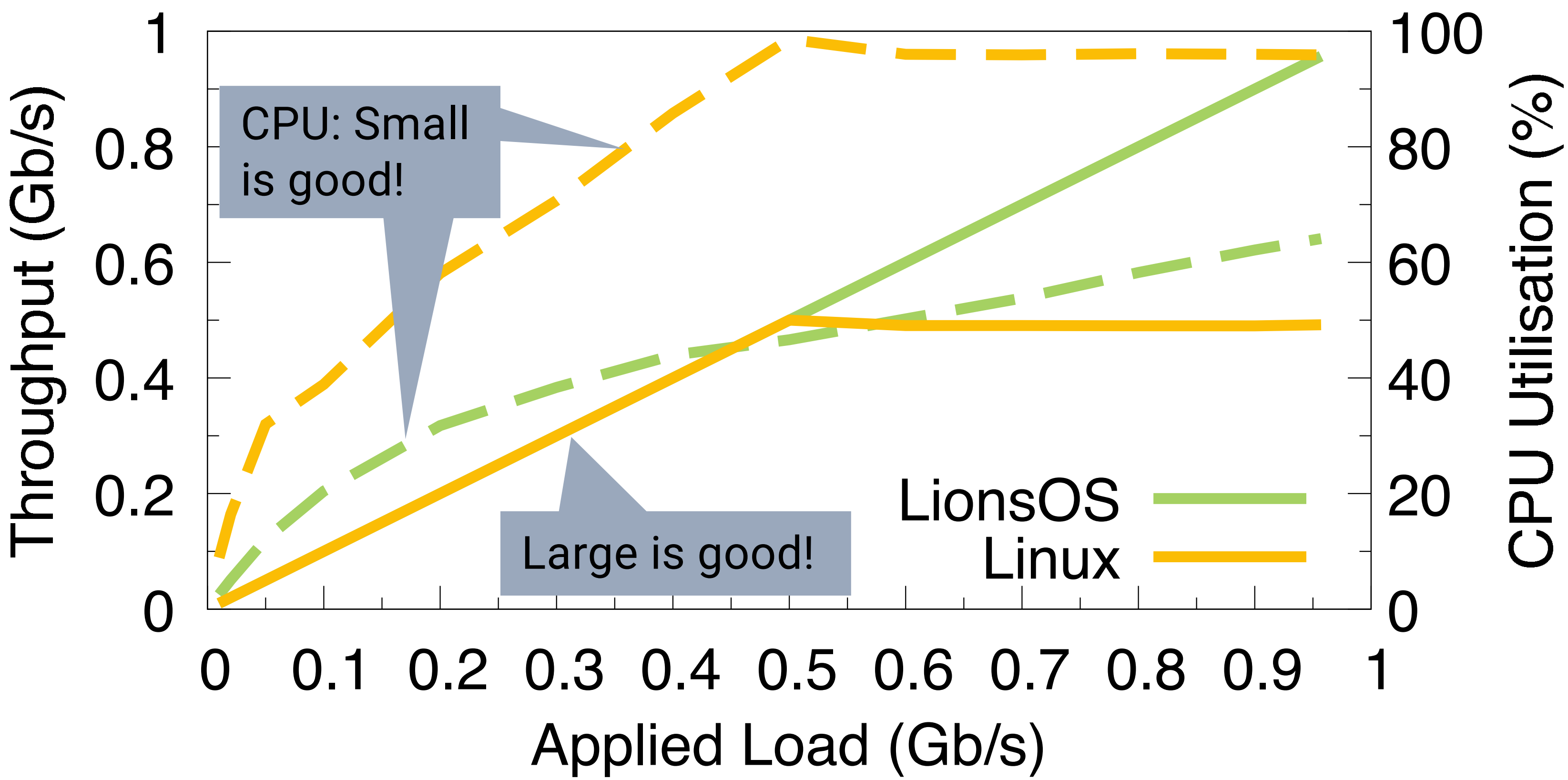
Performance?

Written by second-year student!

LionsOS:
- NW driver: 400 lines
- Virtualiser: 160 lines
- Copier: 80 lines
- IP stack: much simpler, client library
- shared NW system total < 1,000 lines

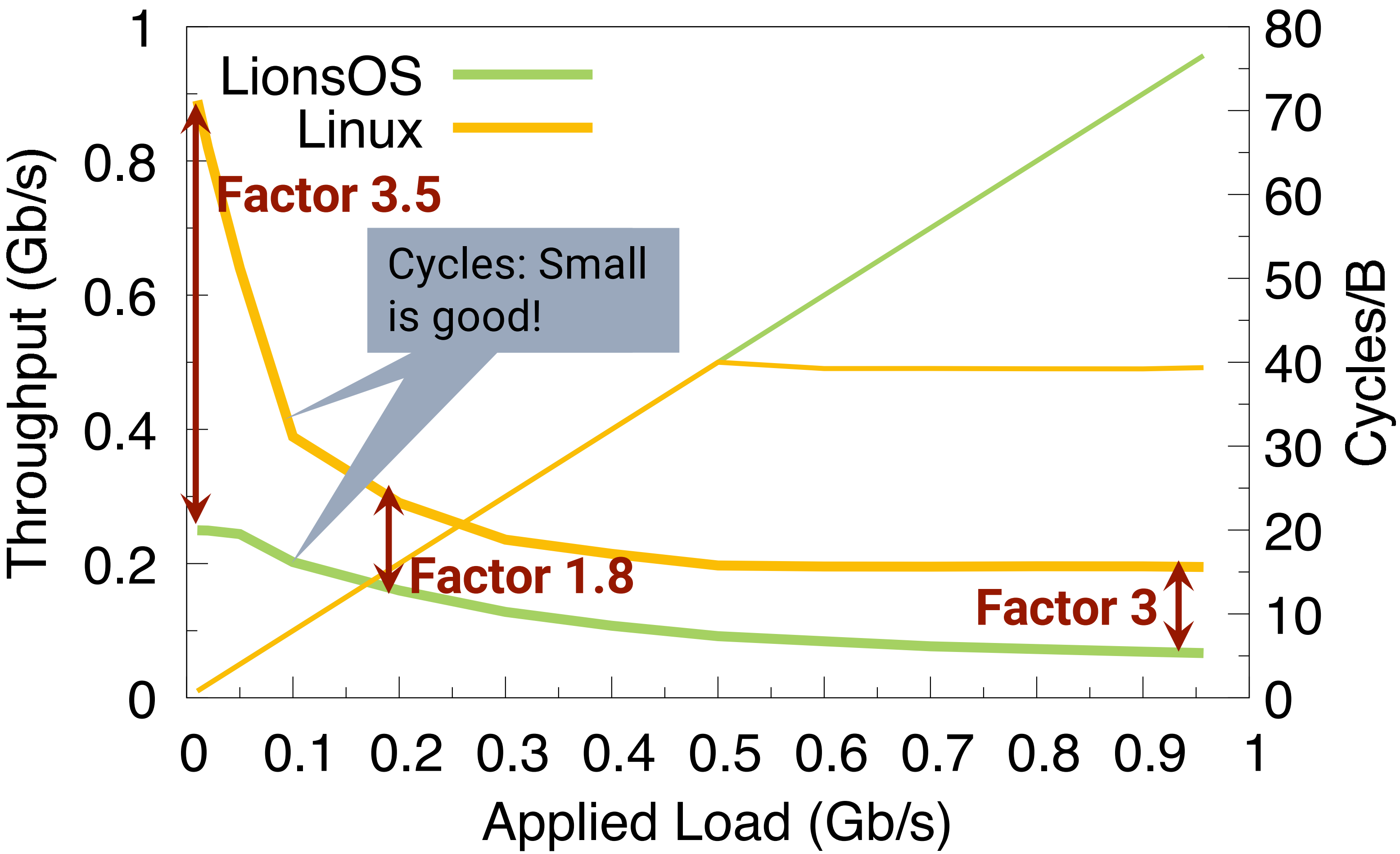
Presently use lwip

Performance: i.MX8M, 1Gb/s Eth, UDP

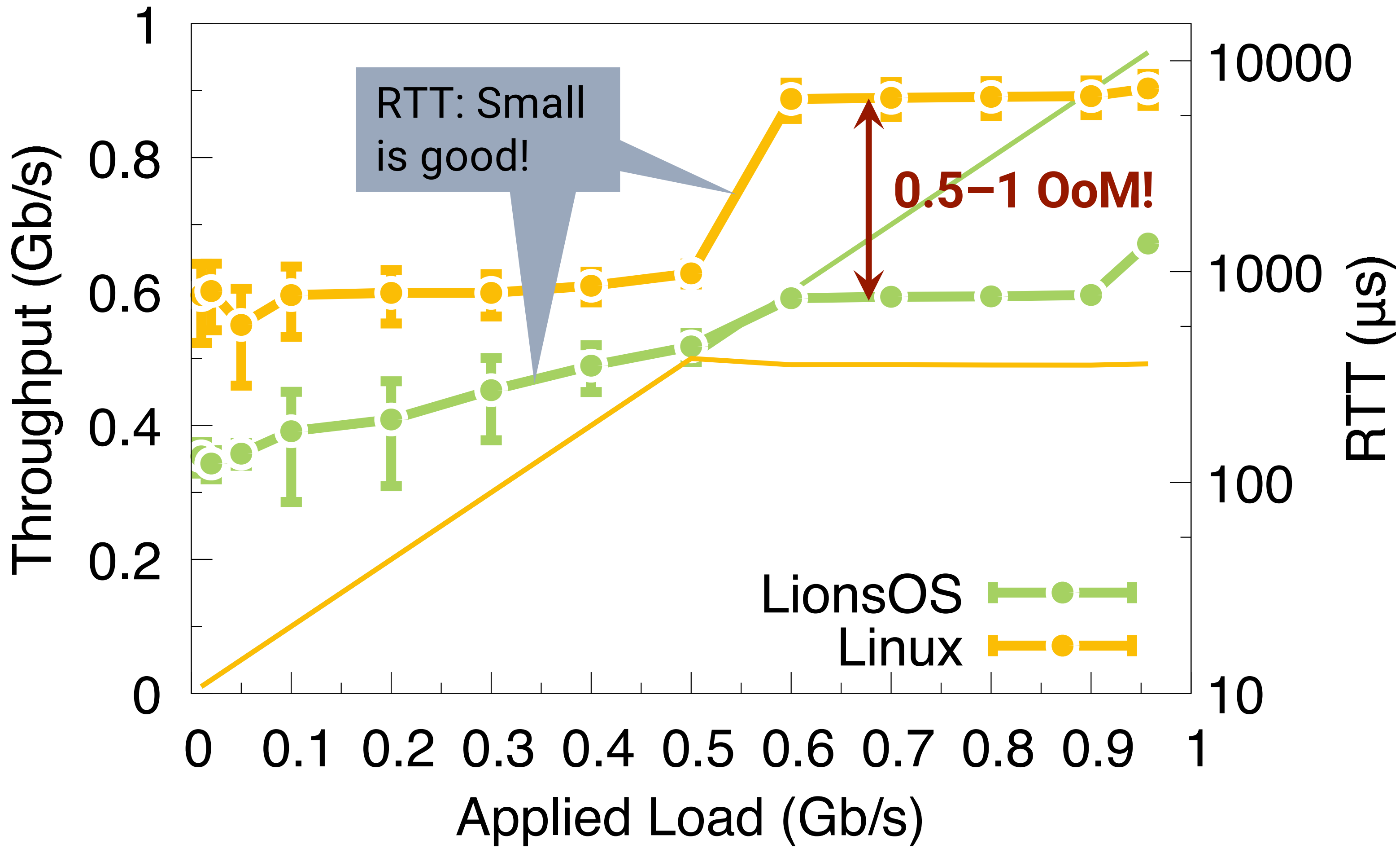


Single-core configuration

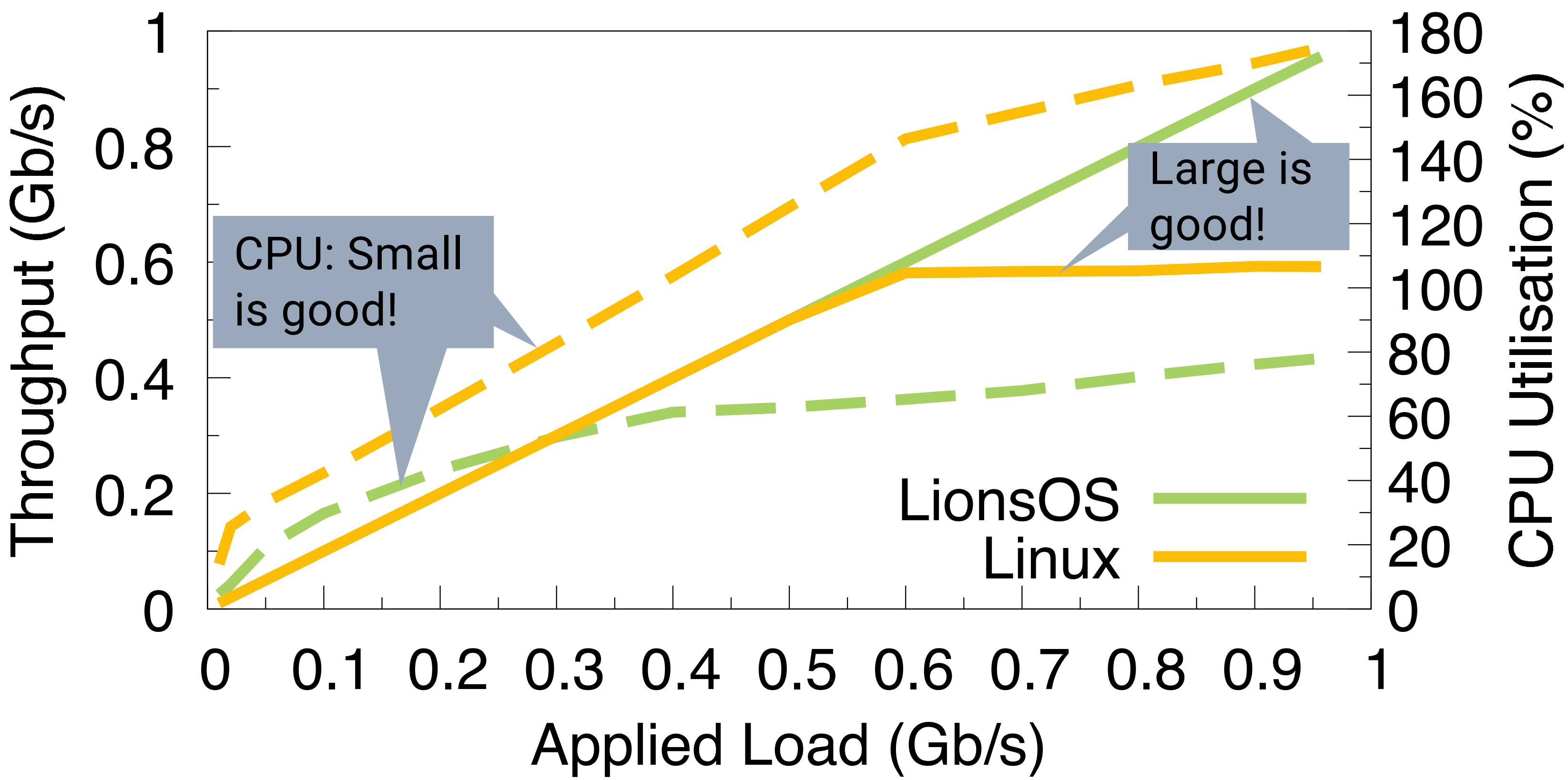
Performance: Processing Cost per Byte



Performance: Round-Trip Times



Performance: i.MX8M, 1Gb/s Eth, UDP



Multicore configuration

Why This Difference?

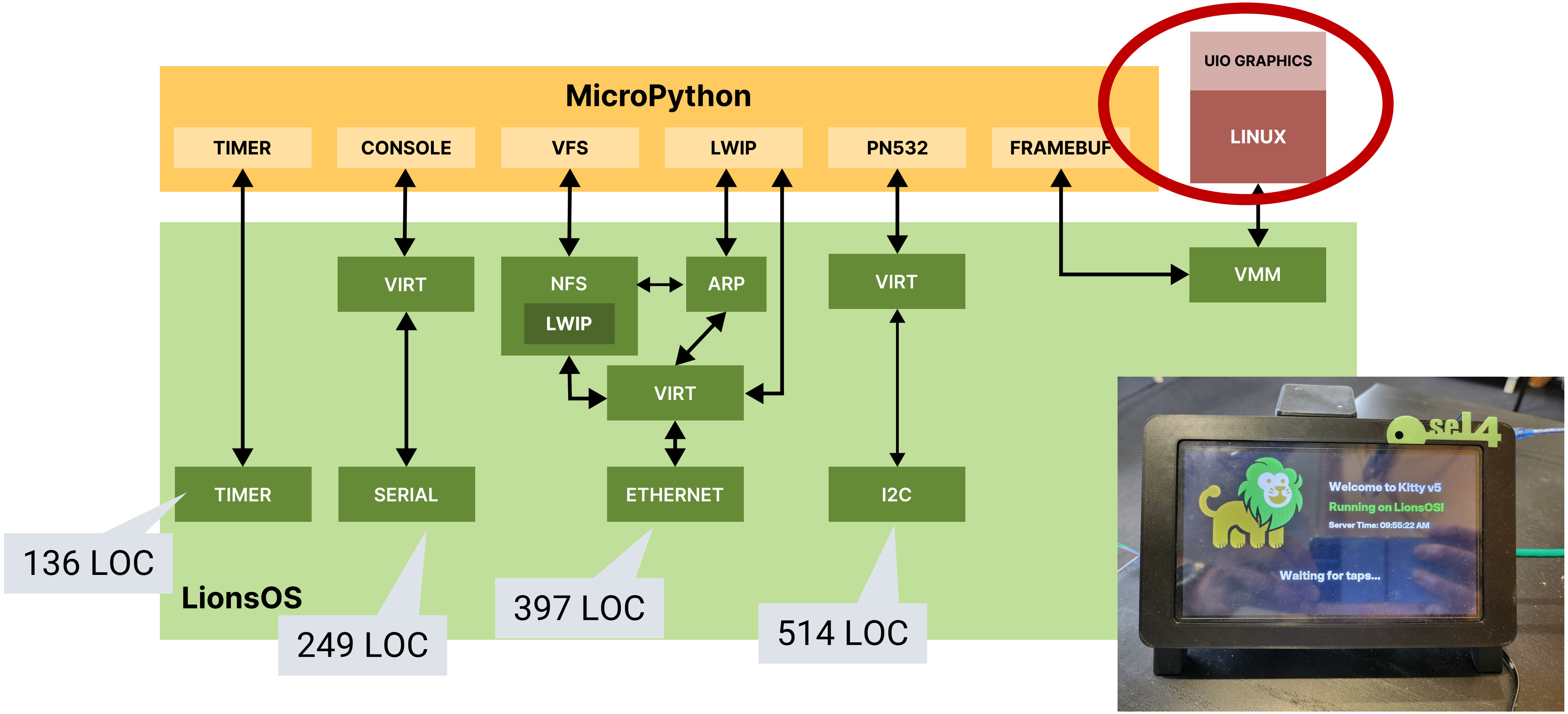
Linux:
- NW driver: 3k lines
- NW system total: 1M lines

Simplicity Wins!

LionsOS executes less code!
- ➢ Direct consequence of use-case-specific policies!

LionsOS:
- NW driver: 400 lines
- Virtualiser: 160 lines
- Copier: 80 lines
- IP stack: much simpler, client library
- shared NW system total < 1,000 lines

PoC: Point-of-Sale Terminal: "Kitty"



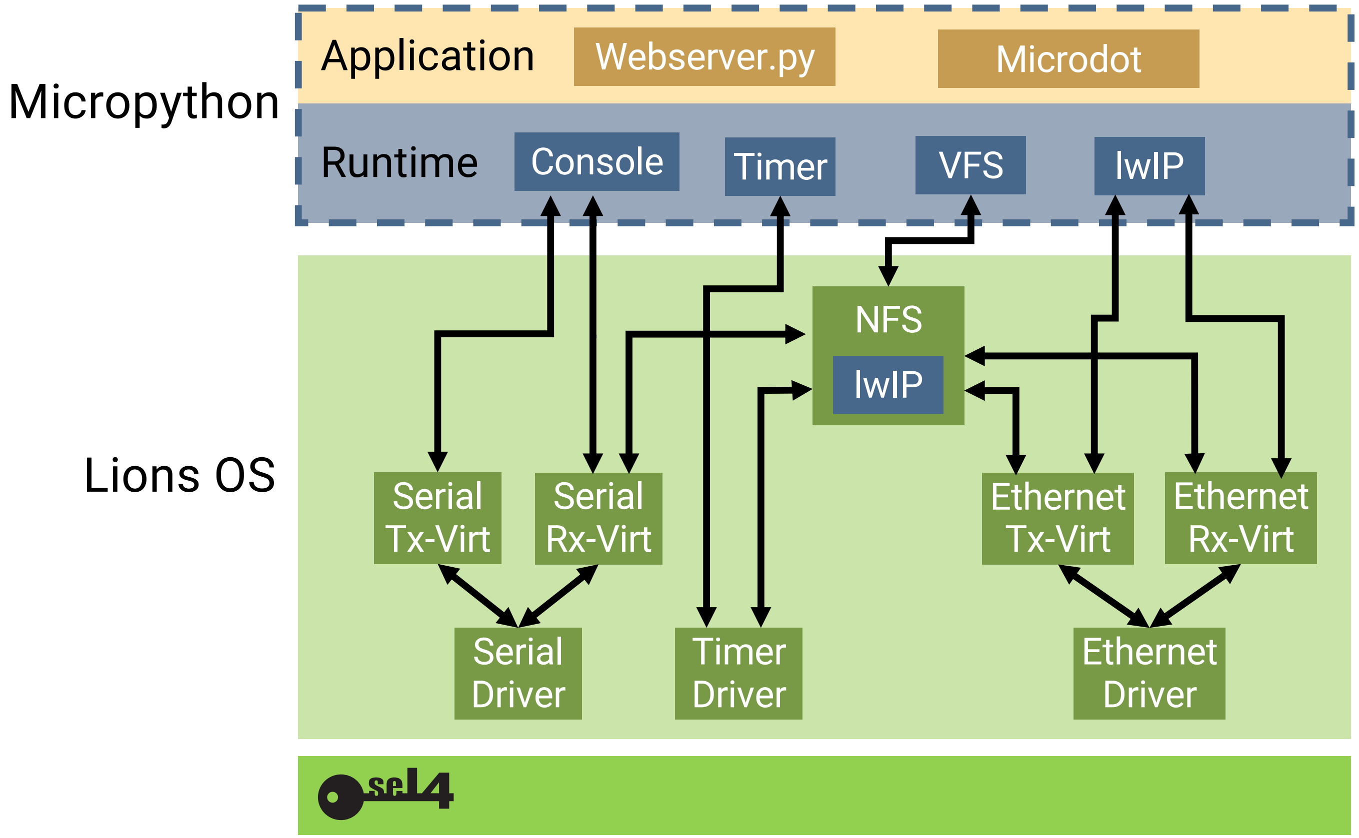
PoS LionsOS Code Sizes (all C)

Trusted:
- 15 modules/ libraries
- Av 210 LoC

Component	LoC	Library	LoC
Serial Driver	249	Microkit	303
Serial Tx Virt	175	Serial queue	219
Serial Rx Virt	126	I^2C queue	101
I^2C Driver	514	Eth queue	140
I^2C Virt	154	Filesys queue & protocol	268
Timer Driver	136		
Eth Driver	397	Coroutines	848
Eth Tx Virt	122	**LWIP**	**16,280**
Eth Rx Virt	160	**NFS**	**45,707**
Eth Copier	79	**VMM**	**3,098**

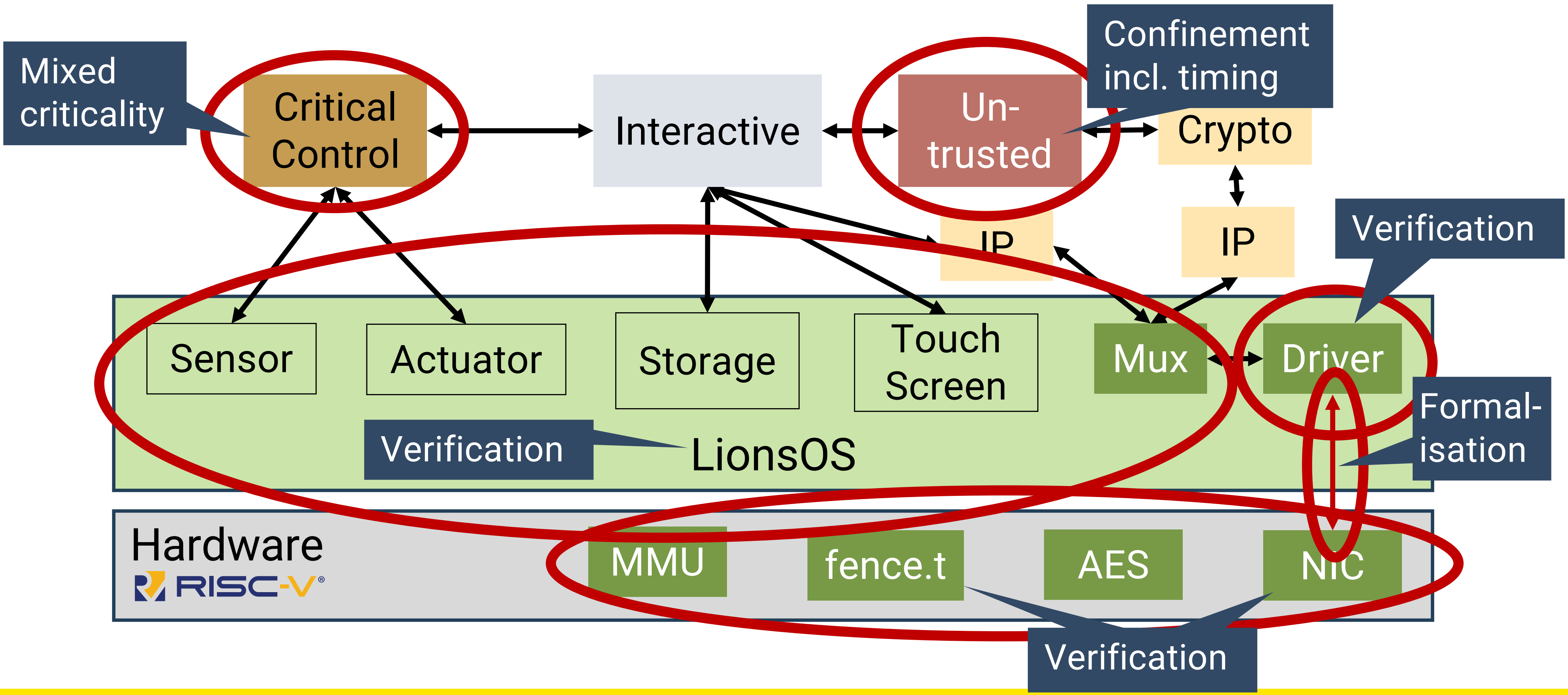
Untrusted

Underneath https://sel4.systems/



How About Verification?

Agenda for Next 3 Years



Verifying LionsOS – How?

- LionsOS programming model:
  - simple event handlers
  - strictly sequential code

Very little time spent on debugging component logic

Suitable for SMT solvers
Demonstrated on NIC driver!

Automatic proofs!

- Fine-grained modularity:
  - concurrency by distribution, "tamed" concurrency
  - complex signalling protocols

Protocol bugs are mostly performance problems

Ideal for model checking!

Challenge:
composition of proofs

Confinement?

Operating Systems — C. Weissman, Editor

A Note on the Confinement Problem

Butler W. Lampson
Xerox Palo Alto Research Center

Communications of the ACM — October 1973, Volume 16, Number 10

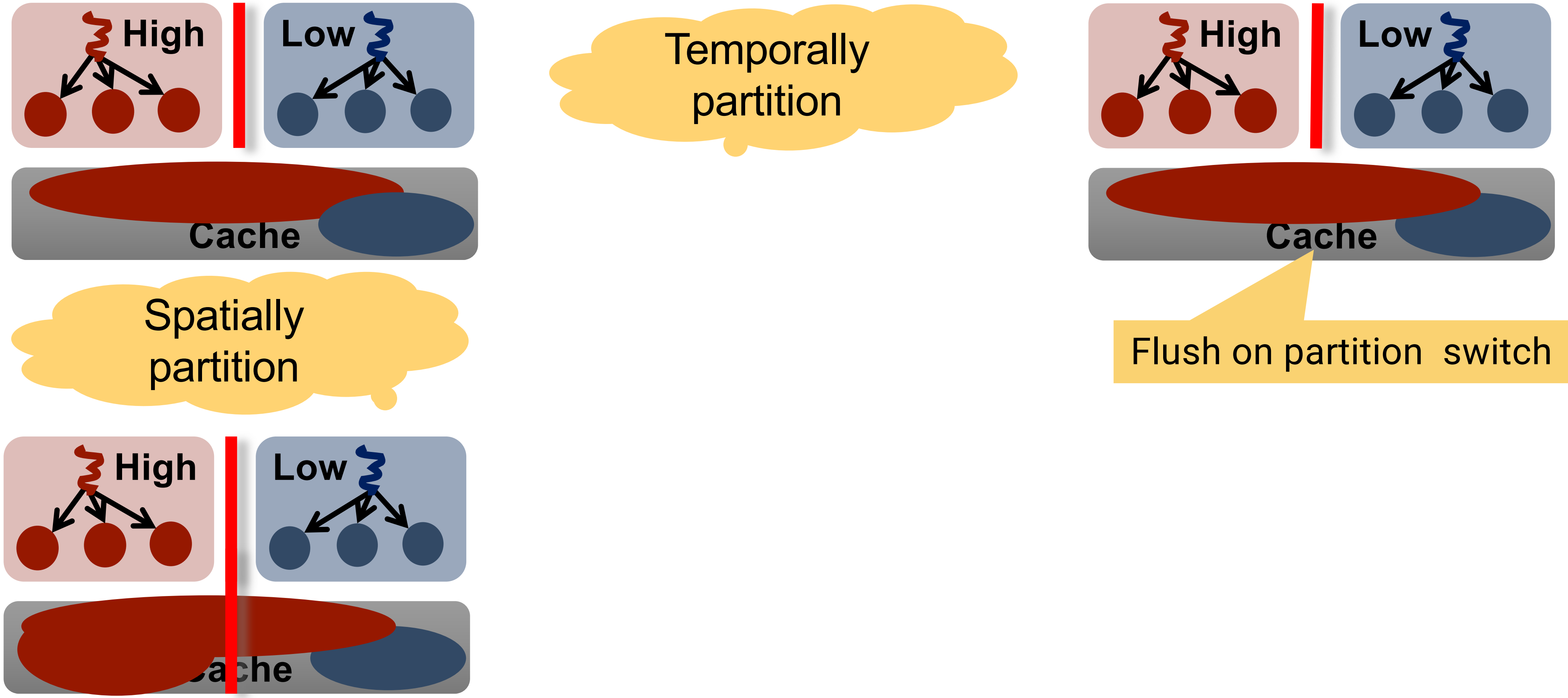
Interactive ⟷ Un-trusted ⟷ Crypto

Must prevent covert channels

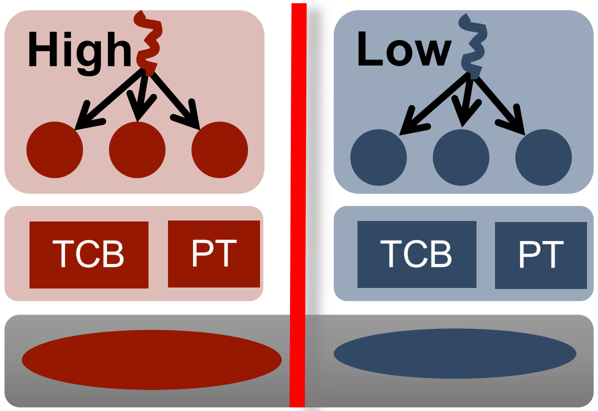
seL4 proved free of covert storage channels [Murray et al, S&P’13]

How about timing channels?

Time Protection: No Sharing of HW State



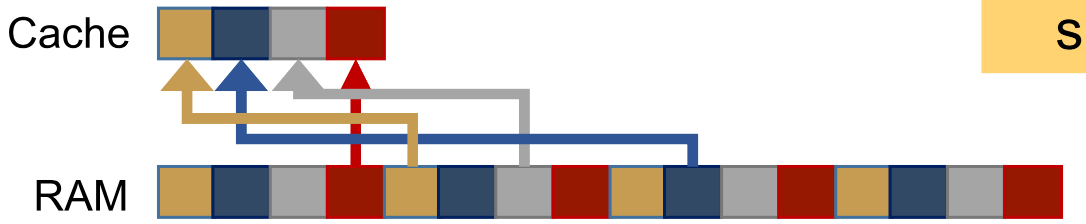
Spatial Partitioning: Cache Colouring



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

How about kernel memory?

- Minimise shared kernel memory by giving each partition own kernel image
- Ensure deterministic cache state of shared kernel memory at partition switch



Temporal Partitioning: Flush State

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!

Ensure deterministic execution

Time padding to remove dependency

Problem: Processors do *not* provide mechanisms for resetting all microarchitectural state!

Ge et al., "Time protection, the missing OS abstraction, EuroSys'19

Solution: fence.t Instruction

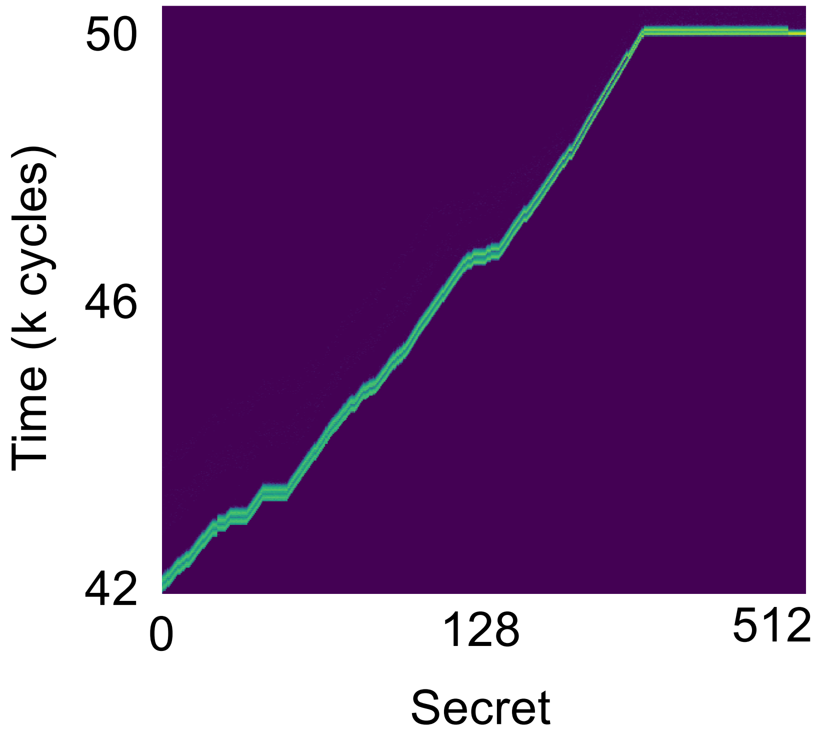
fence.t operation:

- Flush d-cache
- Reset all flip-lops that are **not** part of architected state

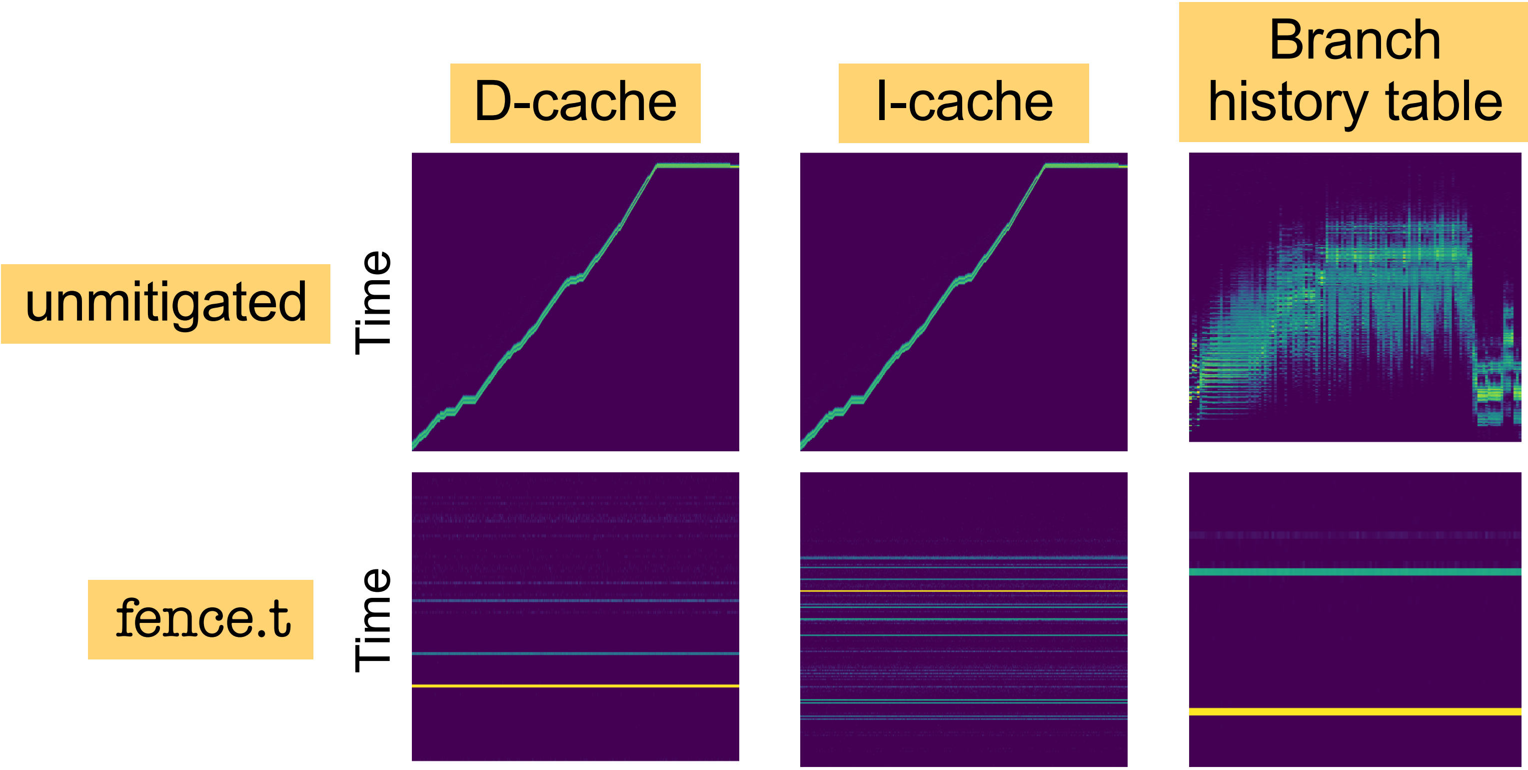
- Prototyped on in-order (CVA6) and OoO (C910) RISC-V processors
- Latency bounded by d-cache flush
- HW cost in the noise

fence.t Instruction on C910

D-cache channel matrix



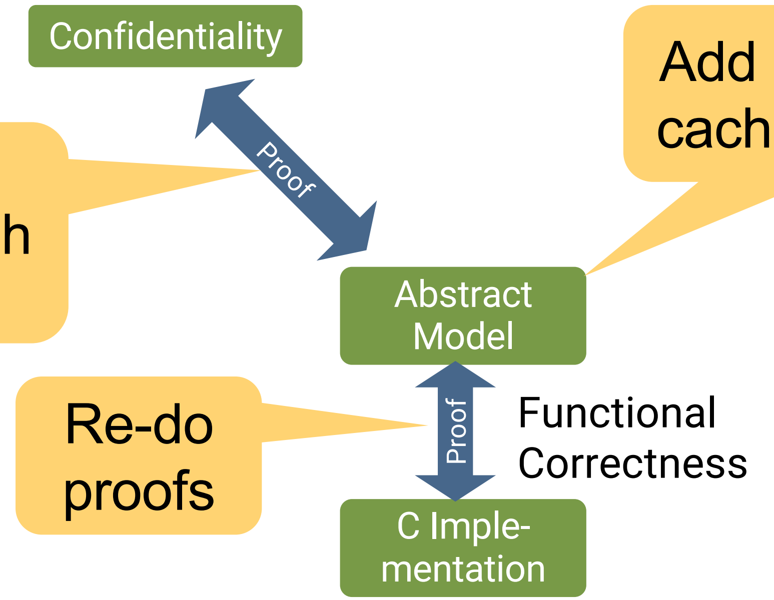
fence.t Instruction on C910



Defeats all known attacks!

Wistoff et al, IEEE-TC'22
Wistoff et al, ApplePies'24

On-Going: Verifying Time Protection



Add state flush, cache colouring

Prove absence of information flow though microarchitecture

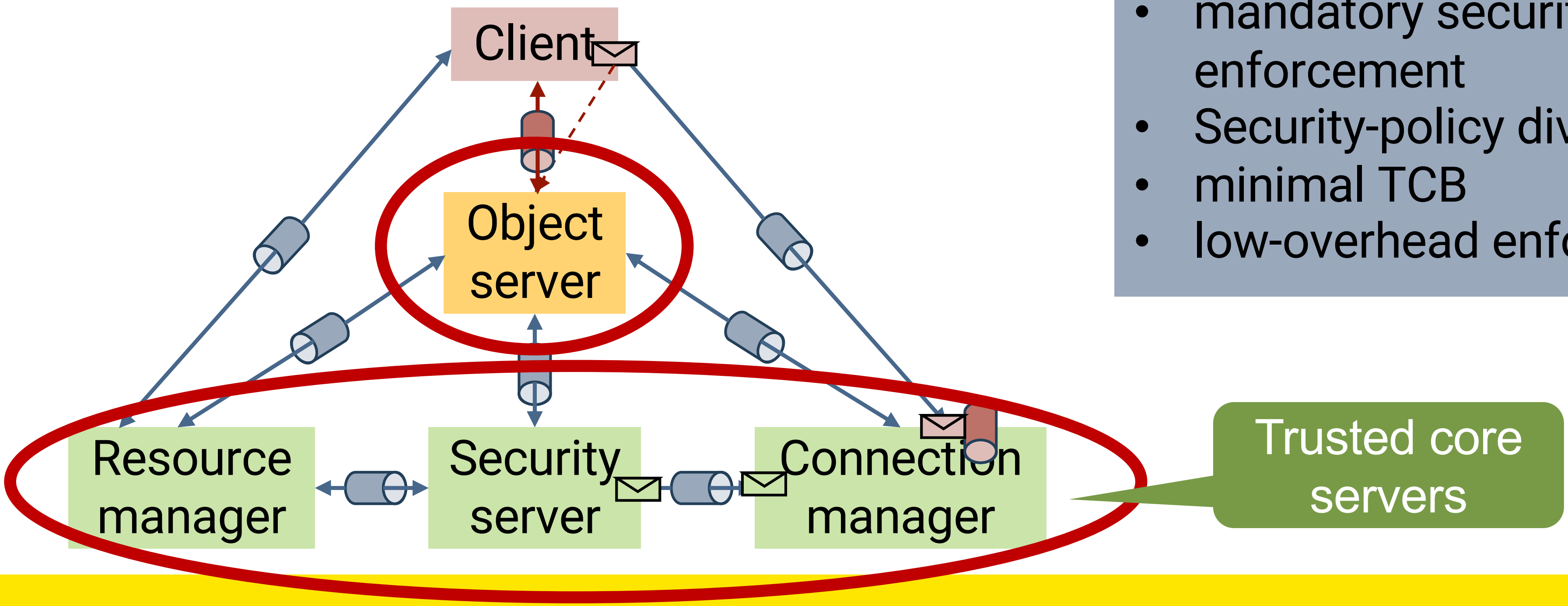
Re-do proofs

Verification enabled by tying leakage to (abstract) micro-architectural state – turning timing into storage channels!

Looking ahead:
Provably secure general-purpose OS

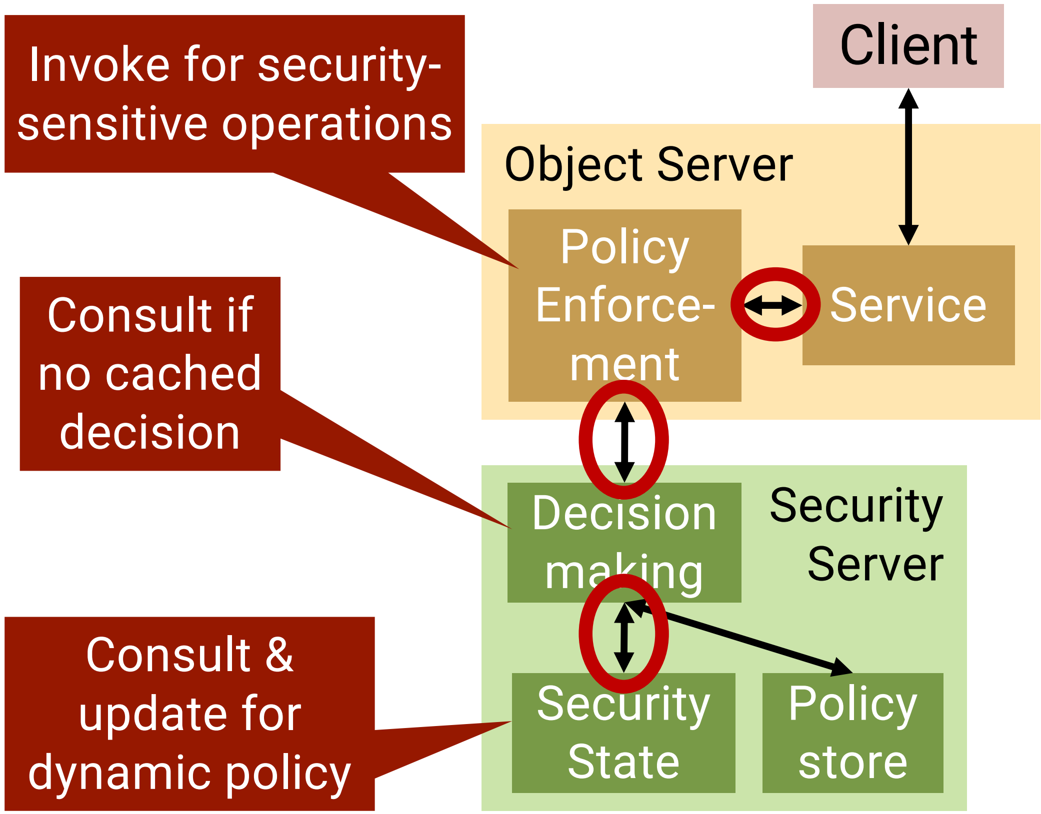
Beyond LionsOS: General Purpose OS

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

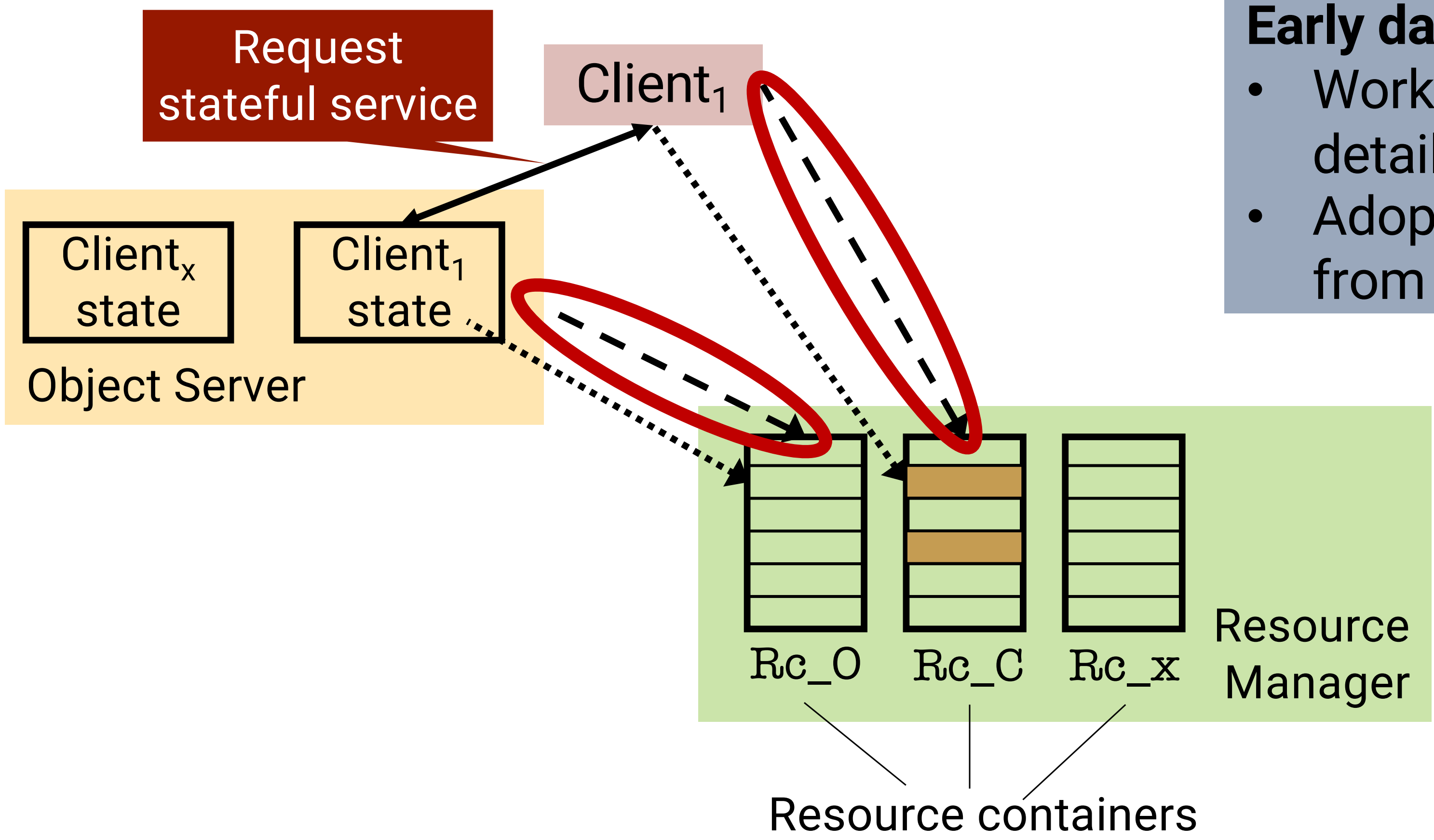


Requires:
- mandatory security-policy enforcement
- Security-policy diversity
- minimal TCB
- low-overhead enforcement

Core Ideas: Dynamic Enforcement



Core Ideas: Resource Donation



Early days:
- Working on framework, details of model
- Adopt components from LionsOS

Truly Secure OSes – Finally Happening?

LionsOS:
- Highly performant
- First components verified
- 3-Year plan for end-end proofs
- Limited to static architectures

General-purpose OS:
- Very early days
- … but optimism from LionsOS experience

https://trustworthy.systems

We're hiring!
Operating-systems researchers