School of Computer Science & Engineering

Trustworthy Systems Group

MCS Safety – An OS Perspective

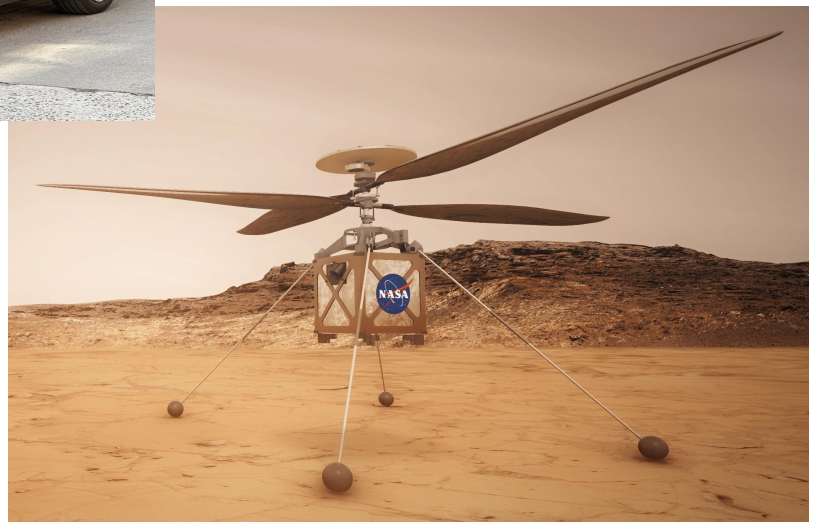
Gernot Heiser

gernot@unsw.edu.au

@microkerneldude.bsky.social

https://trustworthy.systems/

Mixed-Criticality Real-Time Systems



Source: Wikipedia

What Is a Mixed-Criticality System?

"A mixed-critical system […] supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform." [Barhorst'09]

Criticality of a component is defined by the impact of failure:

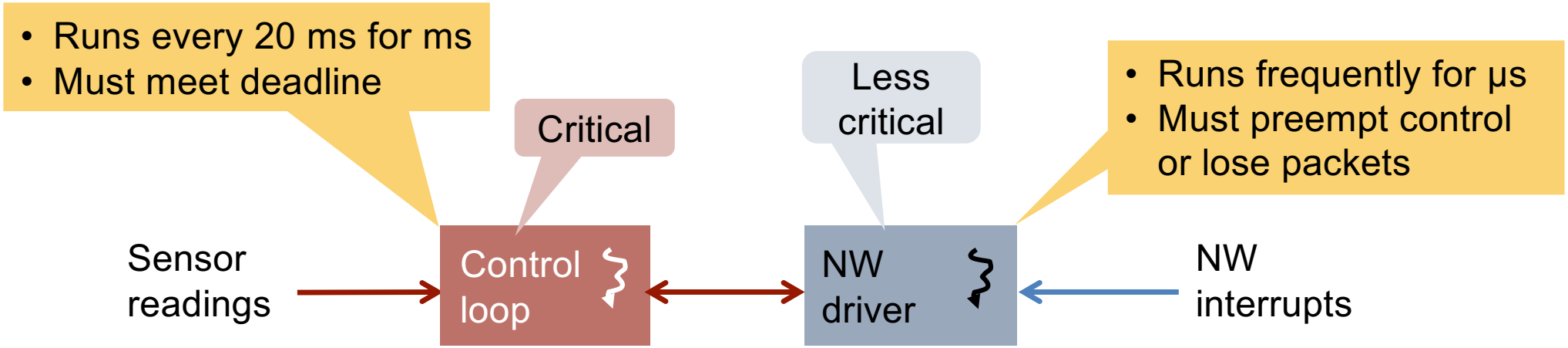
1. loss of life
2. injury
3. inconvenience

Paper review [Aug'17]:
[Gernot:] Temporal isolation is *necessary* for mixed criticality systems
[Reviewer:] Wrong, temporal isolation is *sufficient*

Certification of critical component must not depend on behaviour of less critical components
⇒ **must prevent any interference!**

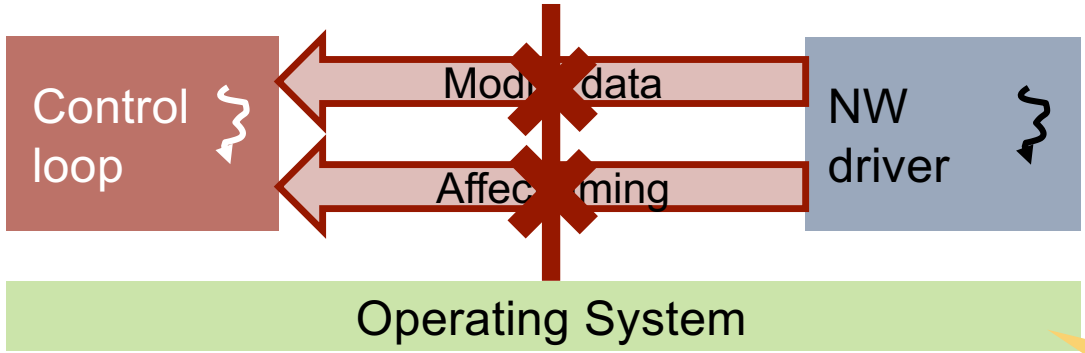
MCS: Simple Example

Requirements:
- Critical operation analysable without assumptions on less critical
- Spatial and temporal isolation



Preventing Interference – the OS's Job!

Requirements:
- Critical operation analysable without assumptions on less critical
- Spatial and temporal isolation



Need OS that guarantees absence of interference!

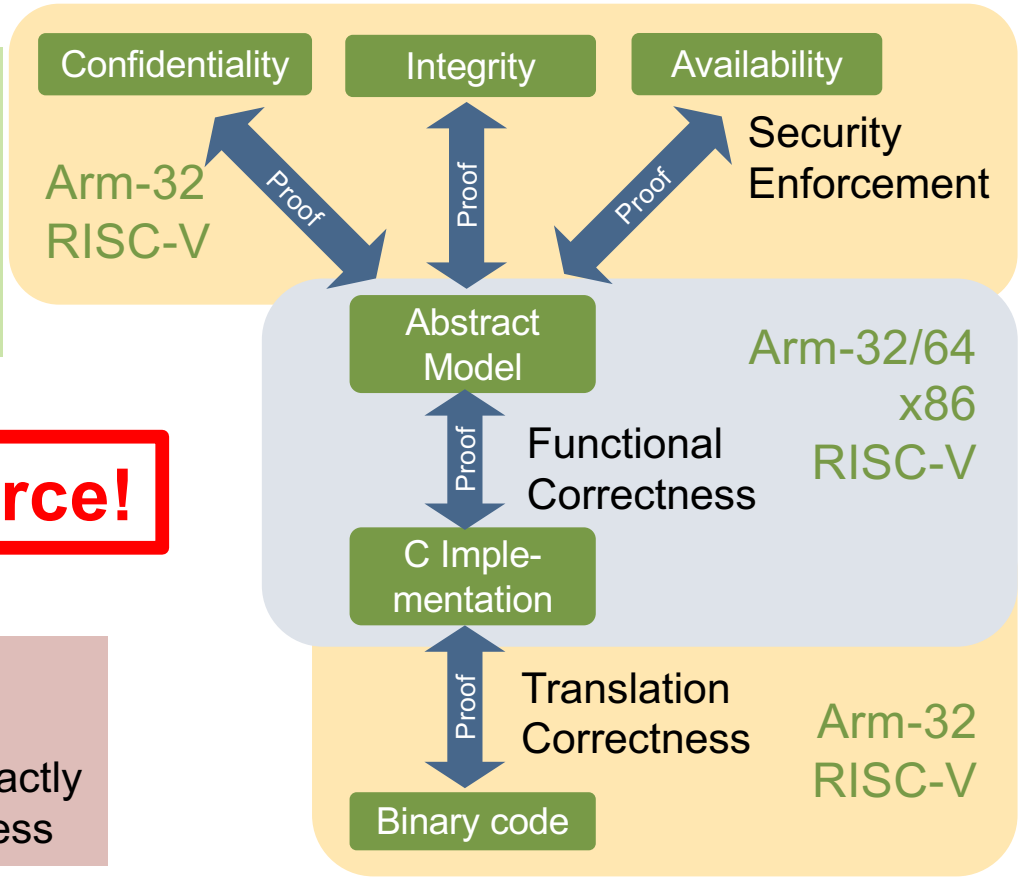
seL4: Verified Isolation

Formally Verified Microkernel

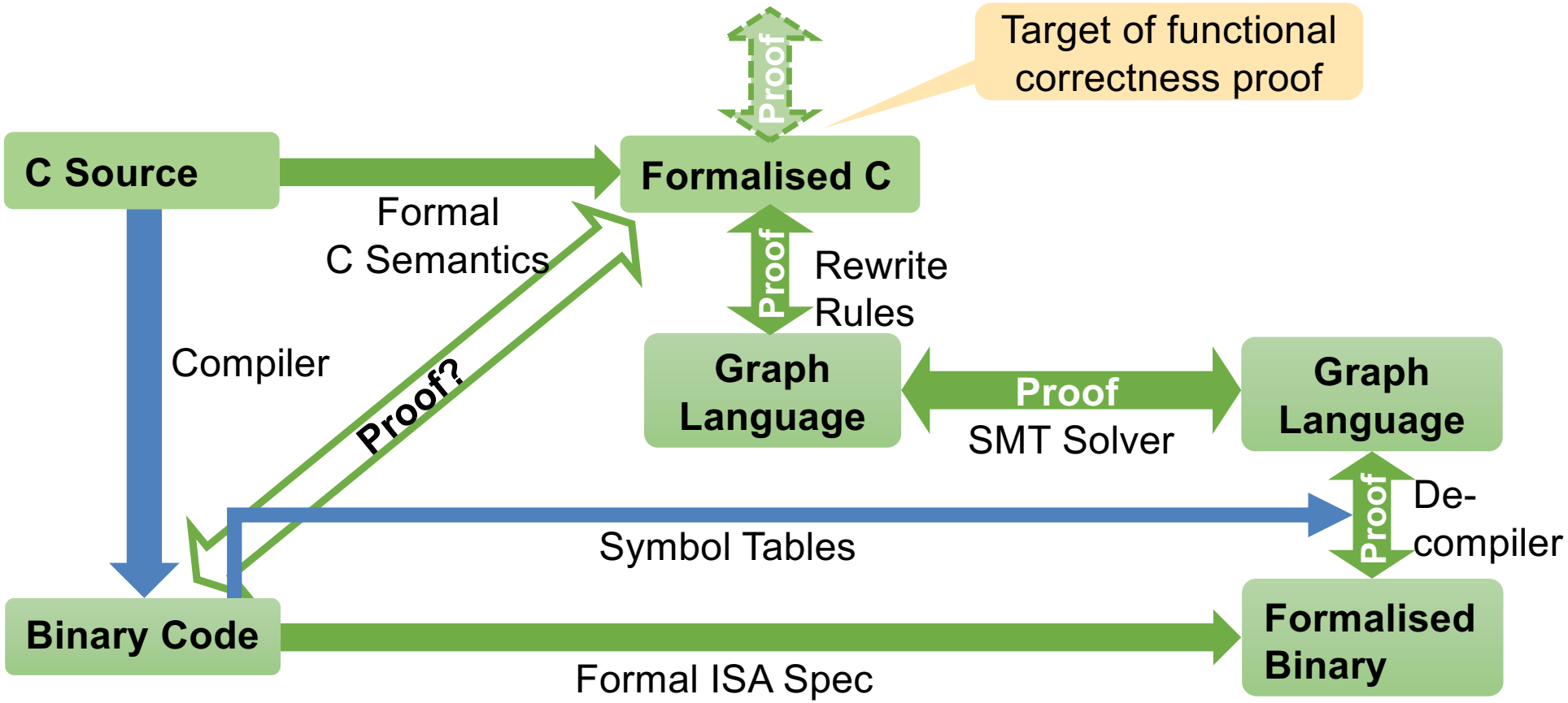
- World's first OS kernel with functional correctness proof
- Most comprehensive verification
- Only verified OS with capability-based fine-grained protection

Open Source!

Present limitations
- Initialisation code not verified
- MMU, caches modelled abstractly
- Multicore verification in progress



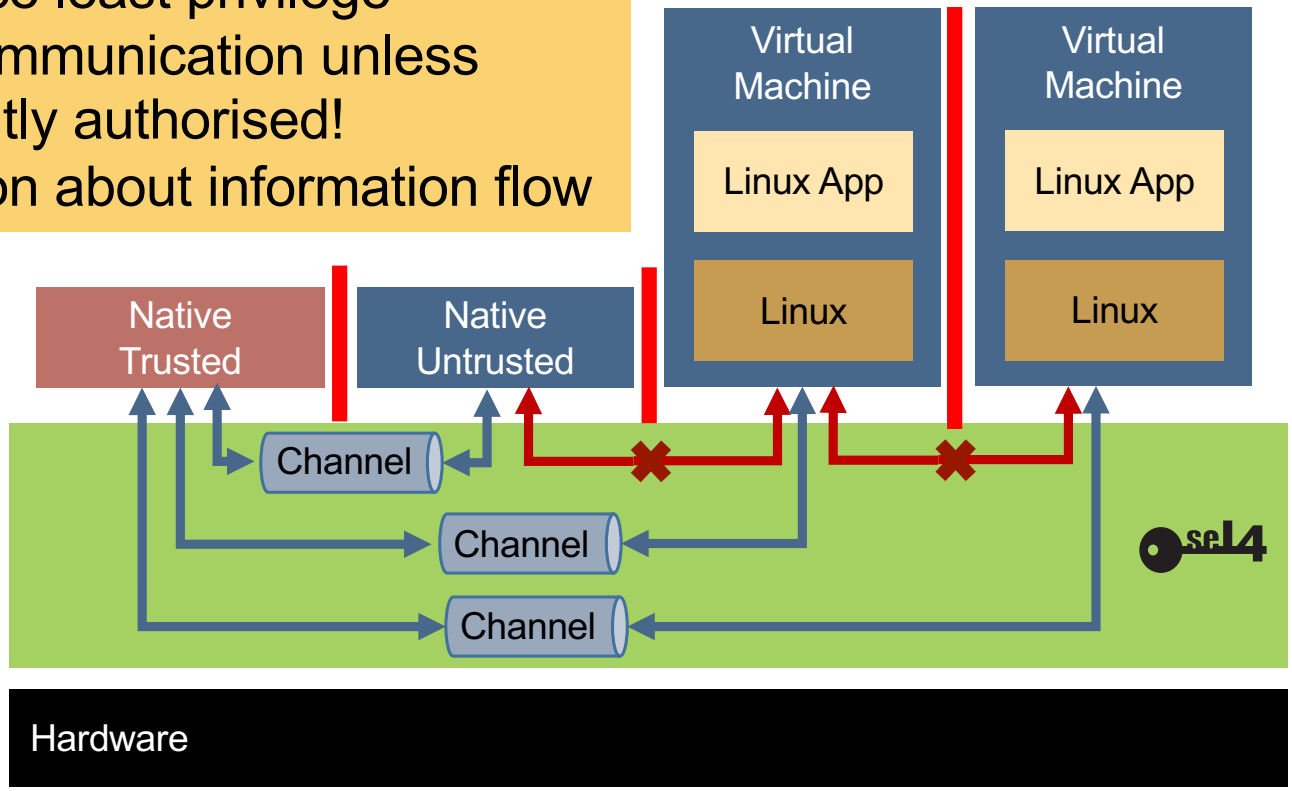
Binary Code Verification (Automatic)



Capabilities: Fine-Grained Protection

Isolation by architecture!

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



The Benchmark for Performance

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

Smaller is better

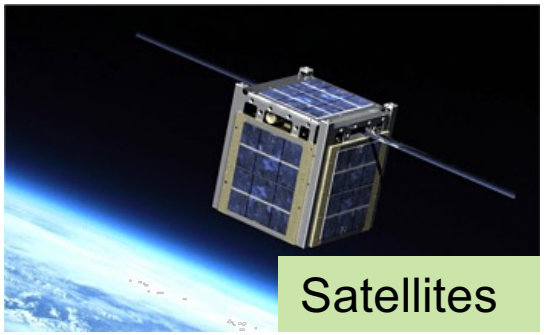
	seL4	Fiasco.OC aka L4Re	Google Zircon
Latency (cycles)	986	2717	8157
Mandatory HW cost* (cycles)	790	790	790
Overhead absolute (cycles)	196	1972	7367
Overhead relative	25%	240%	930%

*: 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

“World’s Most Secure Drone”

← Tweet

DARPA @DARPA

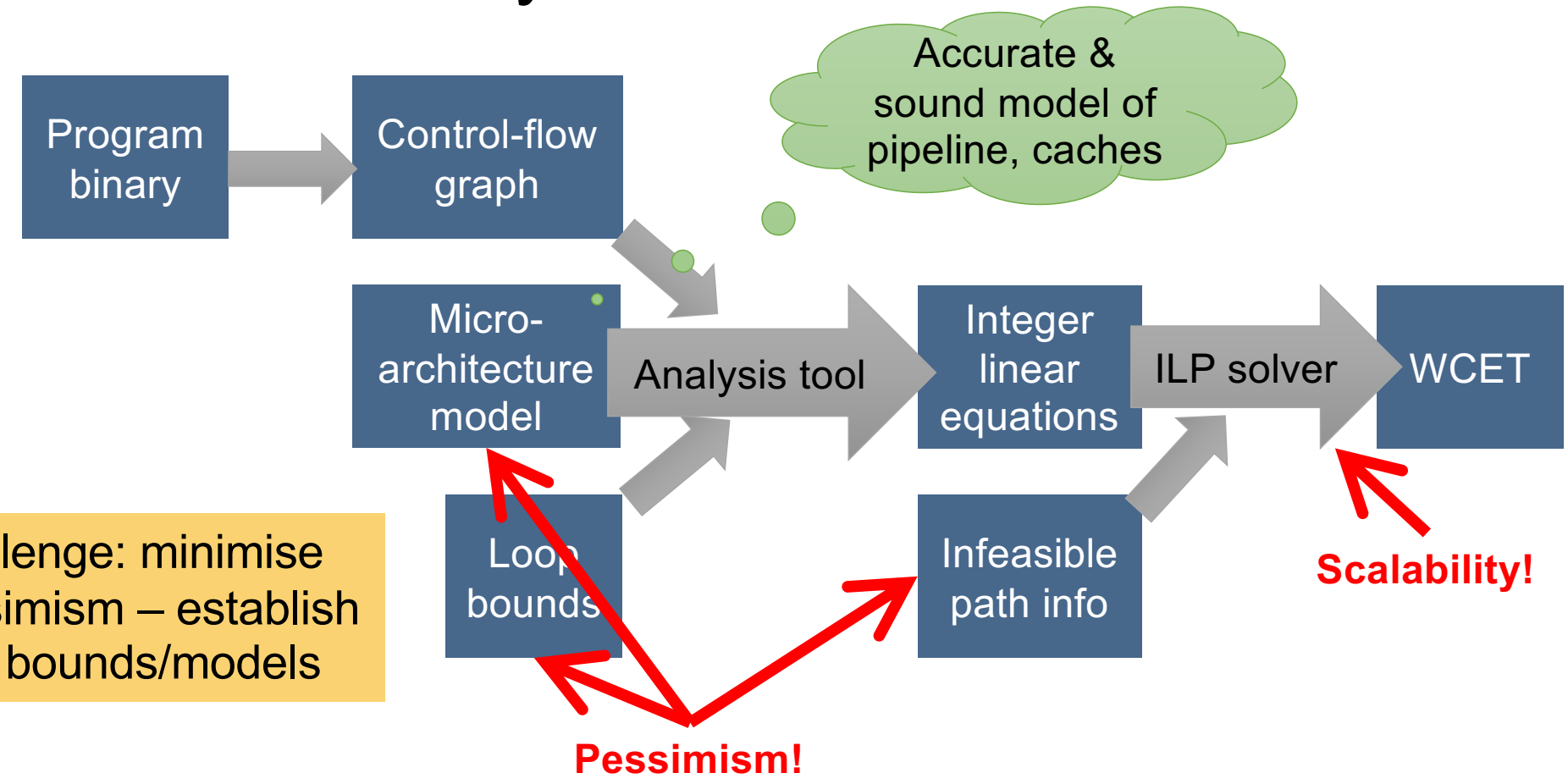
We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!

DEFCON’22

And Temporal Isolation?

- **Kernel WCET analysis**
- **Enforcing time budgets**

WCET Analysis

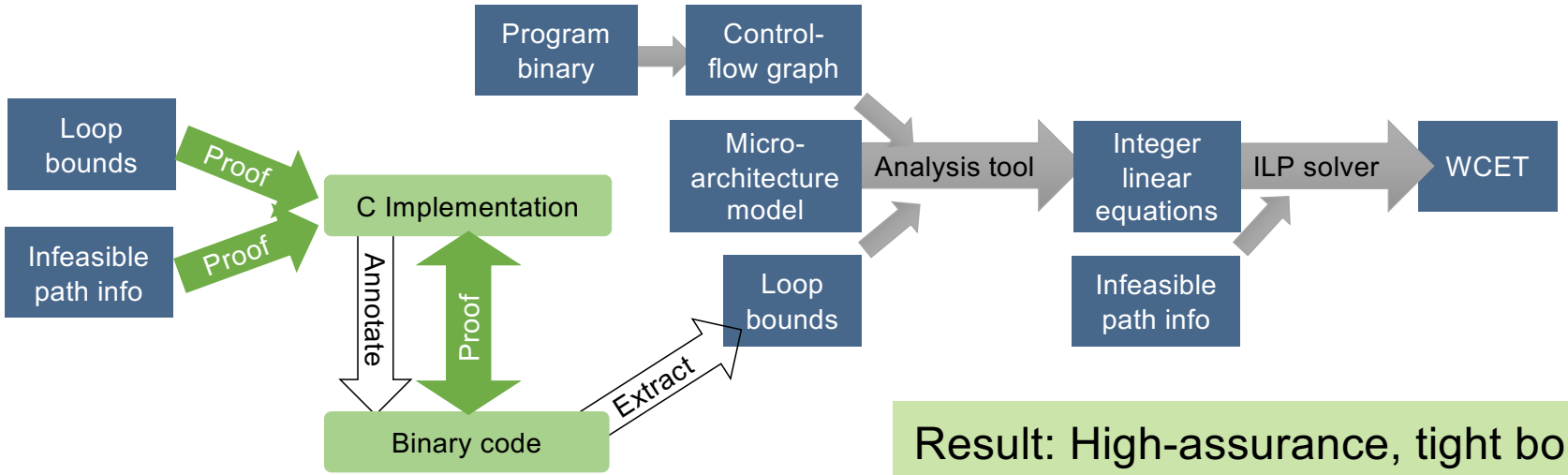


Challenge: minimise pessimism – establish tight bounds/models

Loop Bounds & Infeasible Paths

Tight loop bounds and infeasible path refutations infeasible to obtain from binary – lack of semantic information, especially pointer aliasing analysis

Idea:
- prove on C level
- transfer to binary using translation-validation toolchain



Result: High-assurance, tight bounds!

WCET Analysis: Status

Original analysis based on NUS Chronos, Armv6/7 (32-bit), pre-MCS support [RTSS'11, RTAS'13, RTAS'14, RTAS'16]

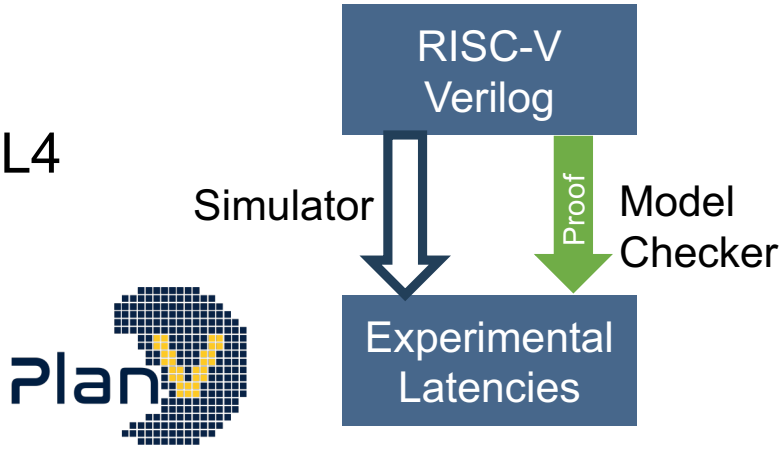
But:

- Chronos unmaintained
- Arm stopped publishing latencies

⇒ Analysis bit-rotted

Recently revived using Heptane

- targeting RISC-V, 64-bit, MCS version of seL4
- PlanV determining latencies from Verilog



Time Budgets: Scheduling Contexts

Classical seL4 thread attributes

- Priority
- Time slice

Not runnable if null

seL4-MCS thread attributes

- Priority
- Scheduling context capability

Capability for time

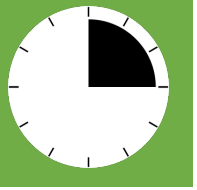
Limits CPU access!

Scheduling context object

- T: period
- C: budget (≤ T)

C = 2
T = 3

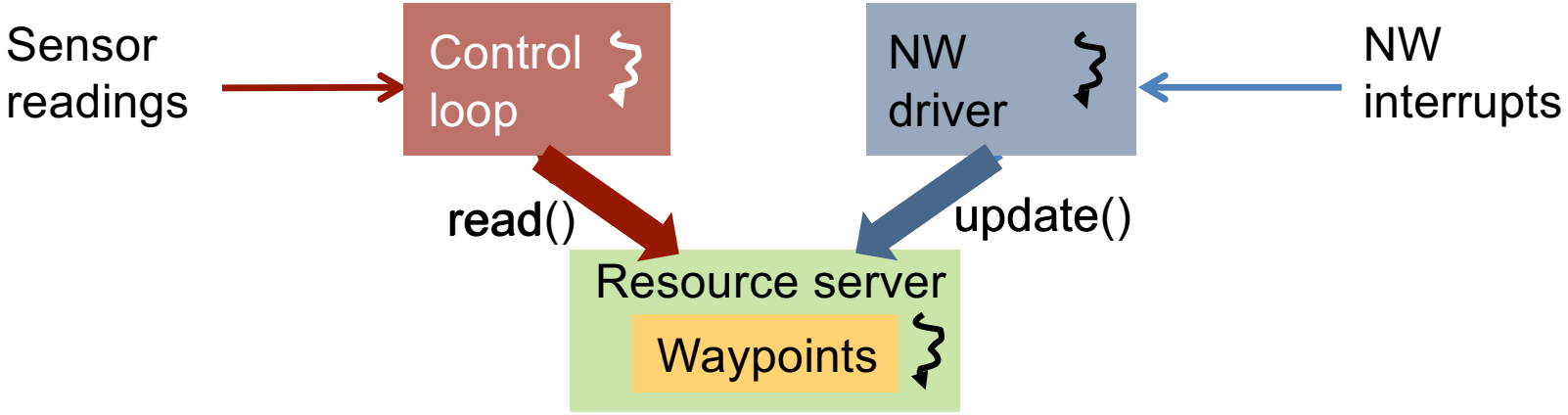
C = 250
T = 1000



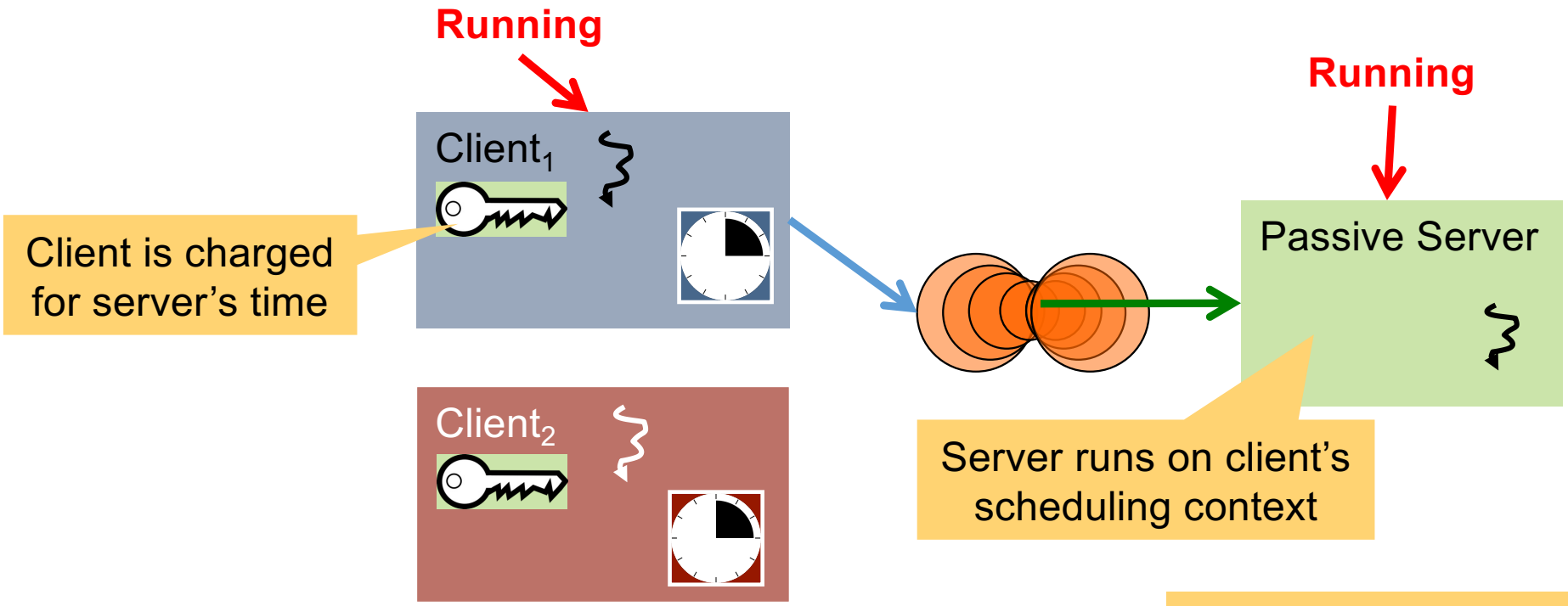
Recall: Simple MCS Example

- Critical control reads waypoints
- Less critical communication component updates waypoints

OS must provide:
- Isolation (temporal & spatial)
- Safe cross-criticality resource sharing



Shared Servers



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

Effectively migrating threads with immediate priority ceiling protocol (IPCP)

Timeout Exceptions

Policy-free mechanism for dealing with budget depletion

Possible actions:

- Provide emergency budget to leave critical section
- Cancel operation & roll-back server
- Reduce priority of low-crit client (with one of the above)
- Implement priority inheritance (if you must…)

Not ideal: better prevent timeout completely
Pending RFC against seL4: budget thresholds

MCS Support: Status

- Functional verification of seL4-MCS due to finish mid-’27
- Currently reviewing/analysing/formalising scheduling
- Formal schedulability framework due Sep’27

Aim: Provable timeliness

Why Isn’t seL4 Pervasive?

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

seL4 Experience of the First 10+ Years

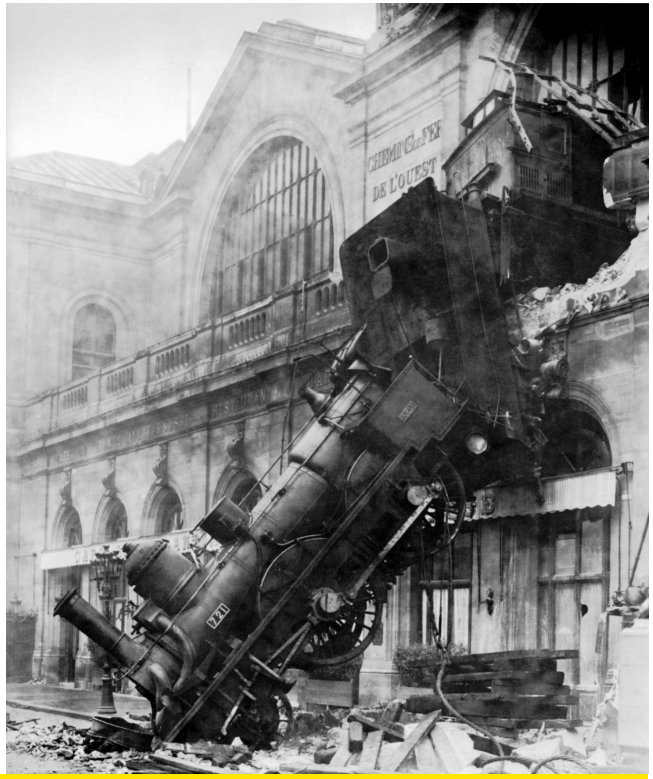
seL4's assurance and power are (still!) unrivalled

Good design on seL4 requires deep expertise

Need seL4-based OS that is:

- secure
- performant
- easy to use
- open source

We contributed poor designs too! [Martins, RTAS'23]



LionsOS

Stop The Train Wrecks!

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 safe & secure OS ever

Must be verified!

Overarching Design Principle: KISS!

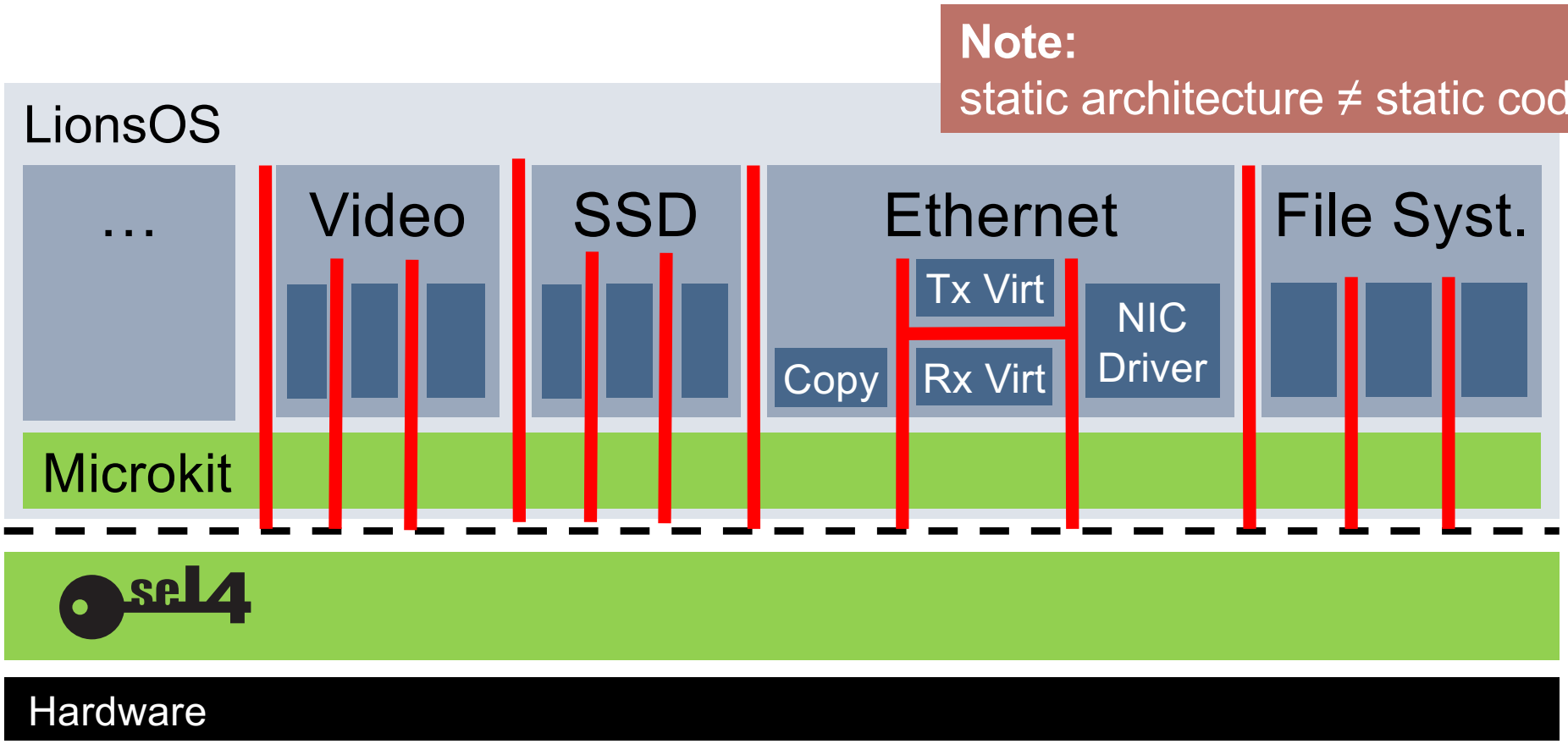
Helps development and verification!

Radical simplicity:

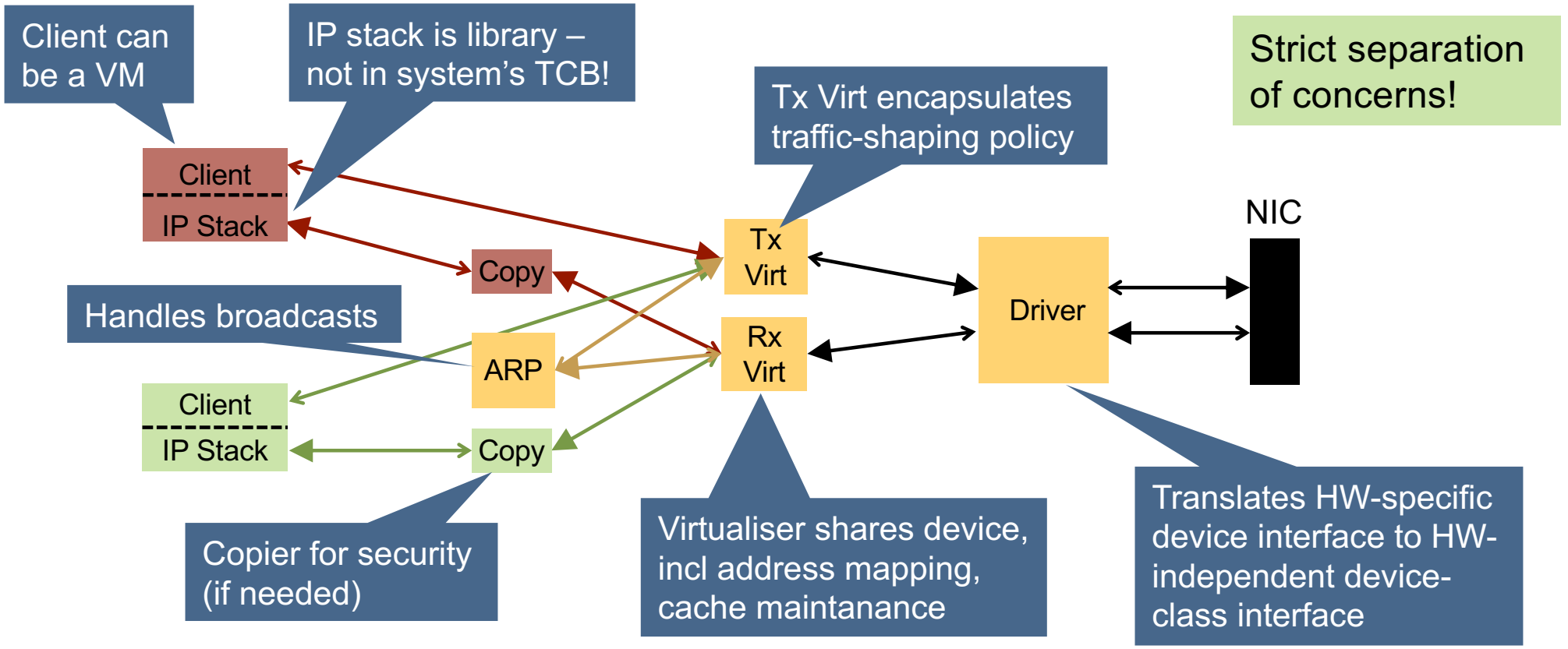
- fine-grained modularity, strict separation of concerns
- event-driven programming model
- use-case-specific policies

Use-case diversity by replacing components

LionsOS: Highly Modular System



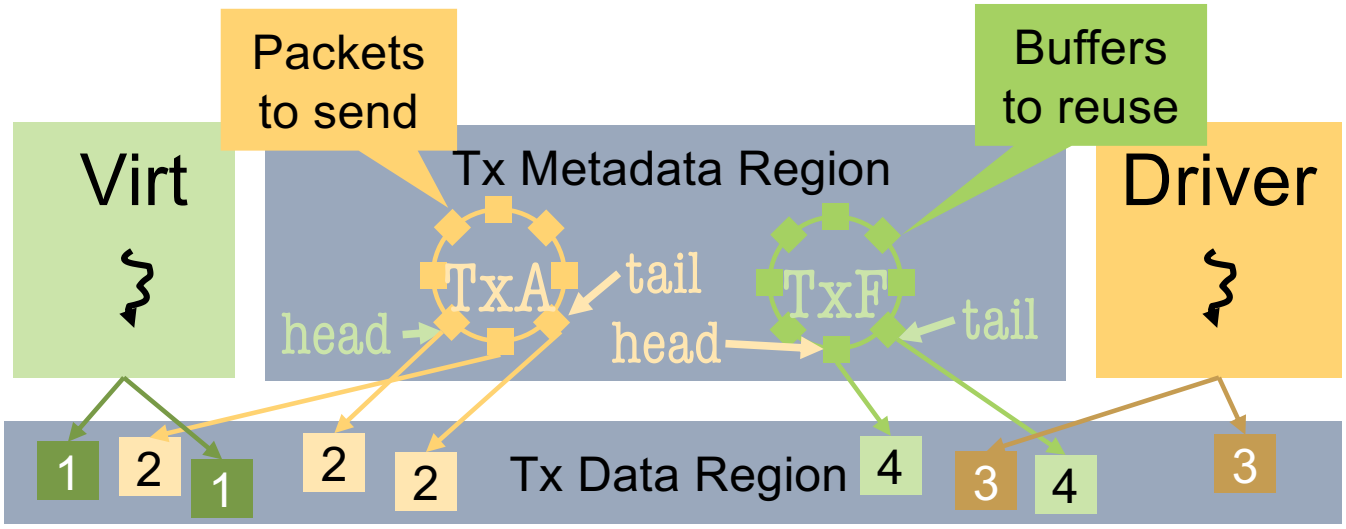
Example: Networking Subsystem



Zero-copy Data Transfer

Components are single-threaded but location-transparent – “tamed” concurrency!

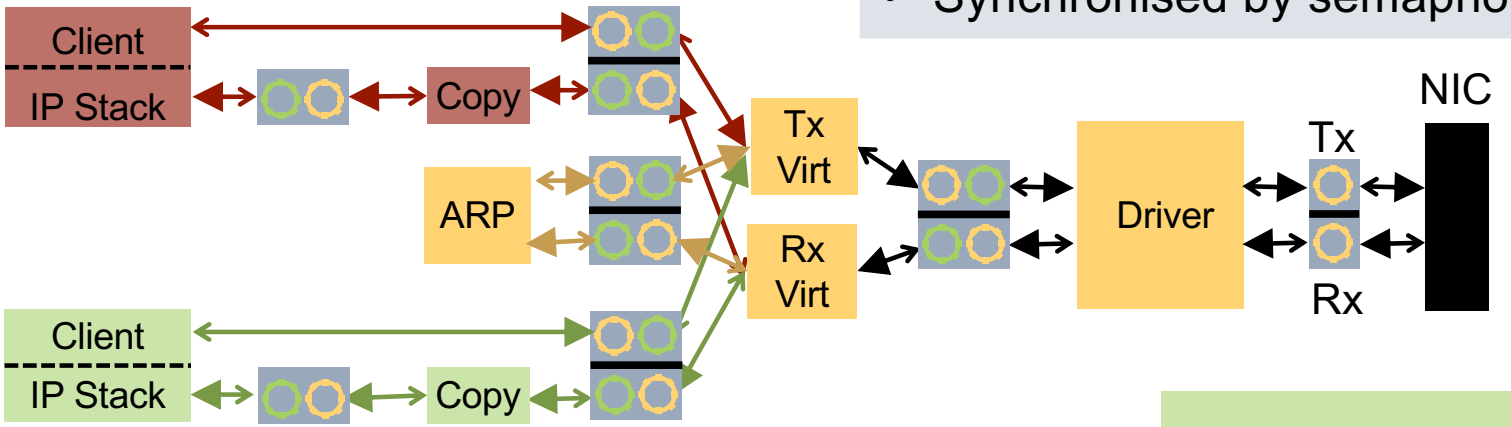
- Lock-free bounded queues
- Single producer, single consumer
- Similar to ring buffers used by NICs
- Synchronised by semaphores



Networking Detail

Zero-copy communication:

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



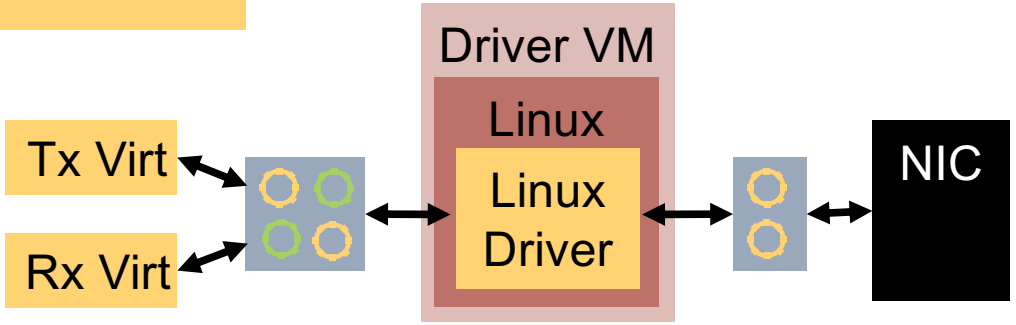
Benefits:

- 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

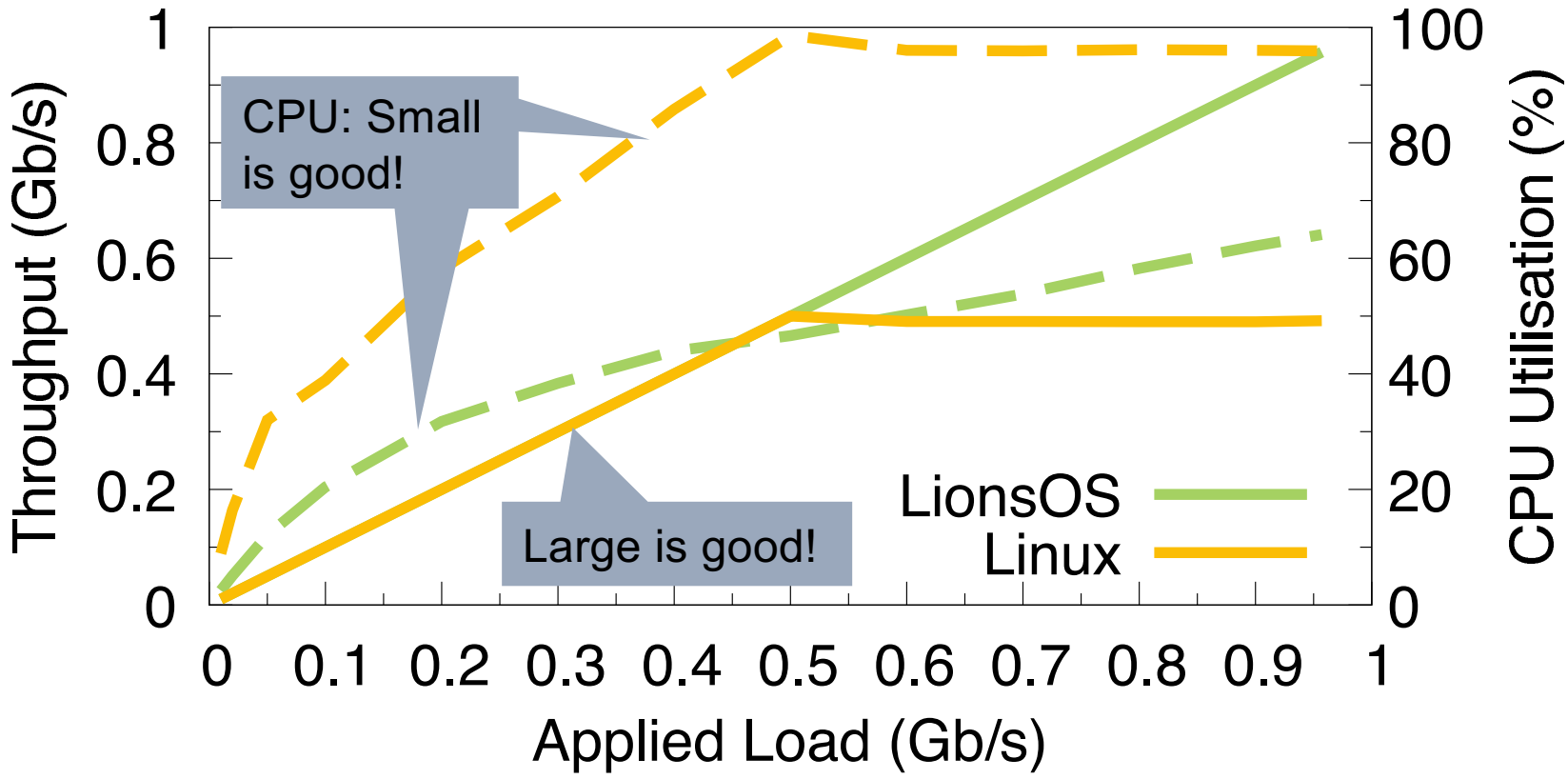
Performance?

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

Why This Difference?

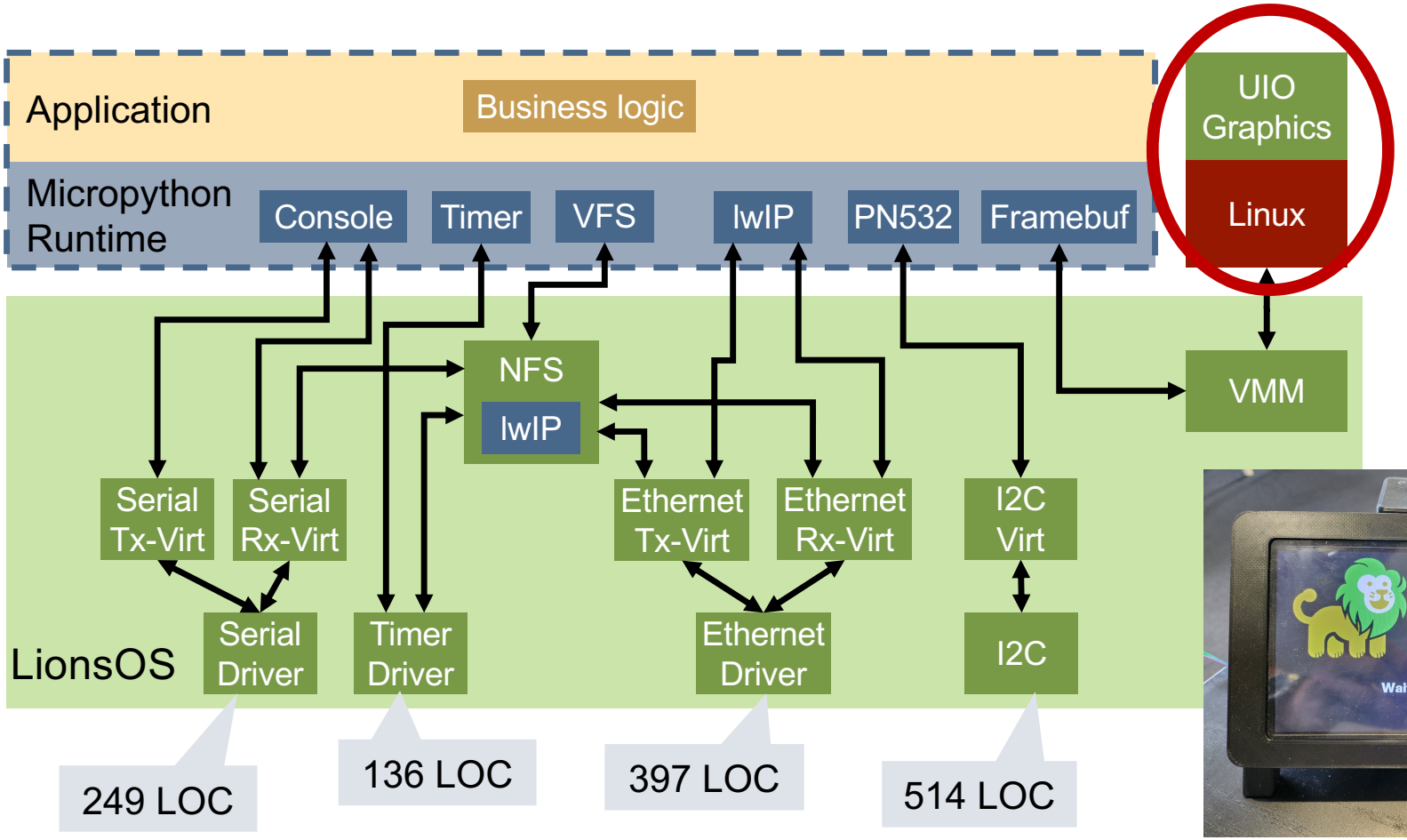
Linux:
- NW driver: 3k lines
- NW system total: 1M lines

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”



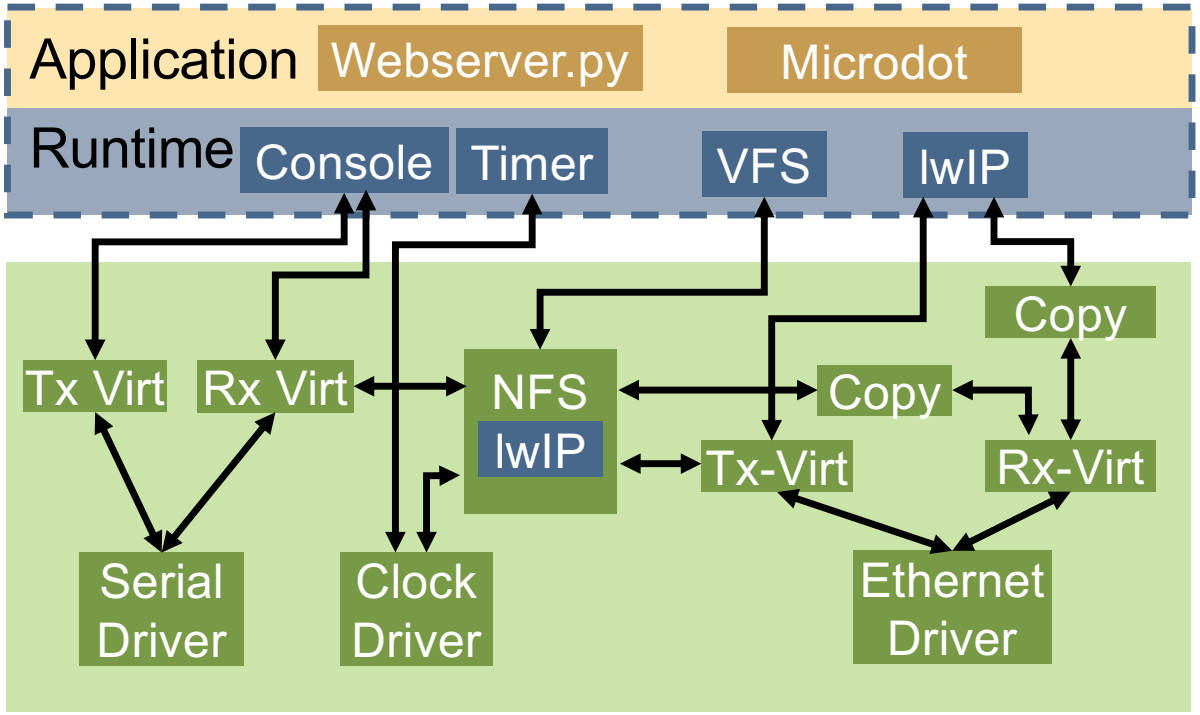
PoS LionsOS Code Sizes (all C)

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	I²C queue	101
I²C Driver	514	Eth queue	140
I²C 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

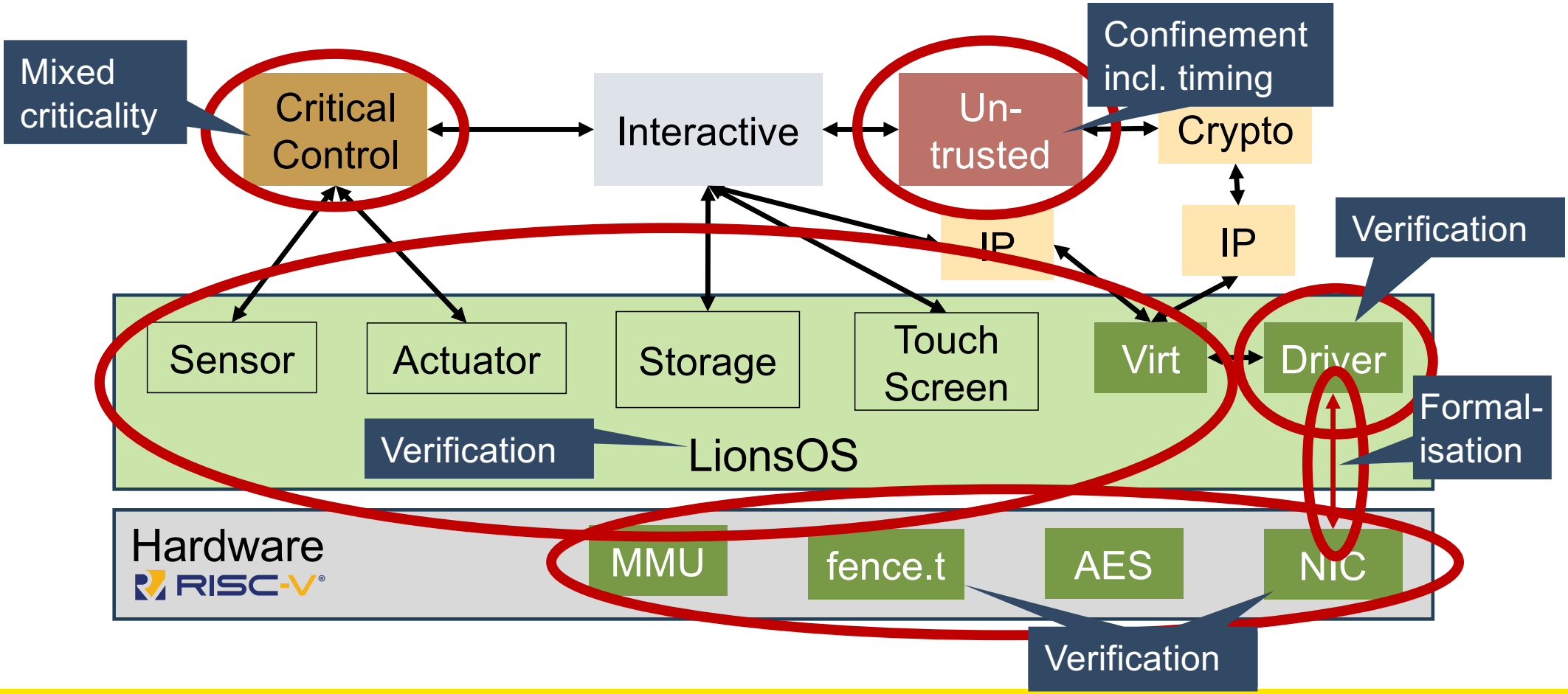
Untrusted

Underneath https://sel4.systems/



How About Verification?

Agenda for Next 1.5 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!

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

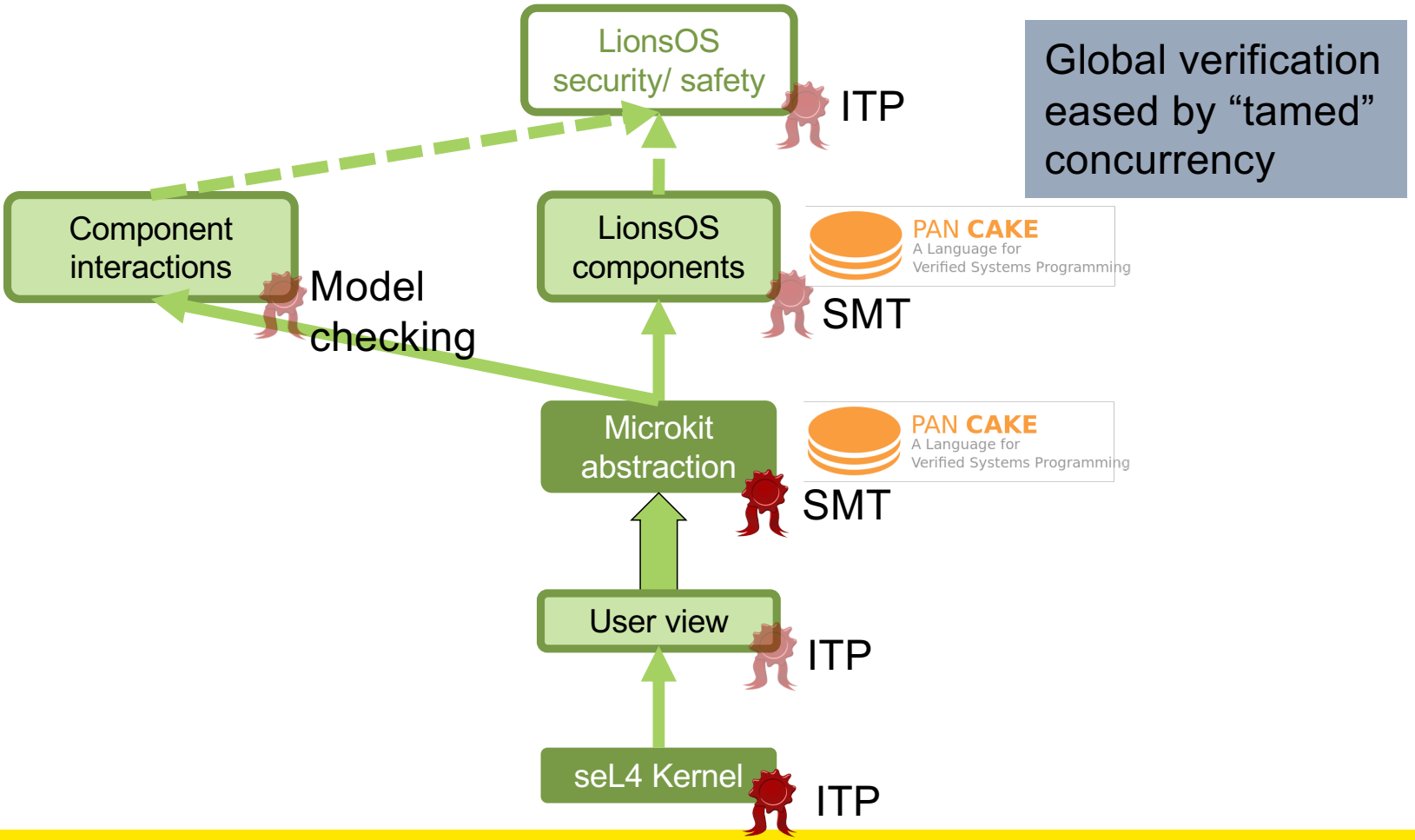
Protocol bugs are mostly performance problems

Ideal for model checking!

Automatic proofs!

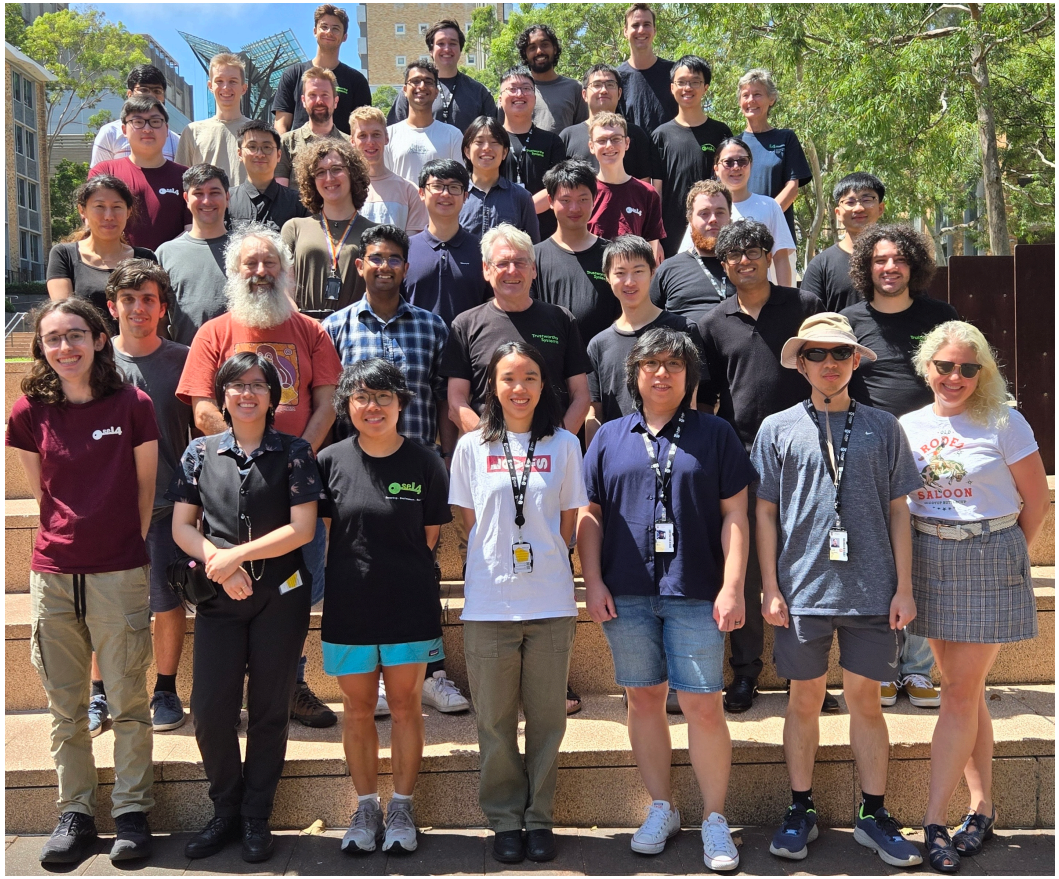
Challenge: composition of proofs

Verifying Lions OS



Global verification eased by “tamed” concurrency

https://trustworthy.systems



We're hiring!
Operating-systems faculty