School of Computer Science & Engineering

Trustworthy Systems Group

Trustworthy Systems R&D Update

Gernot Heiser

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

What’s Happening at TS?

- LionsOS/sDDF/Microkit development
- LionsOS verification agenda
- Pancake
- Device interface specifications
- Other on-going work

LionsOS development

… including Microkit, sDDF

Recap: LionsOS Design Principles

Helps development **and** verification!

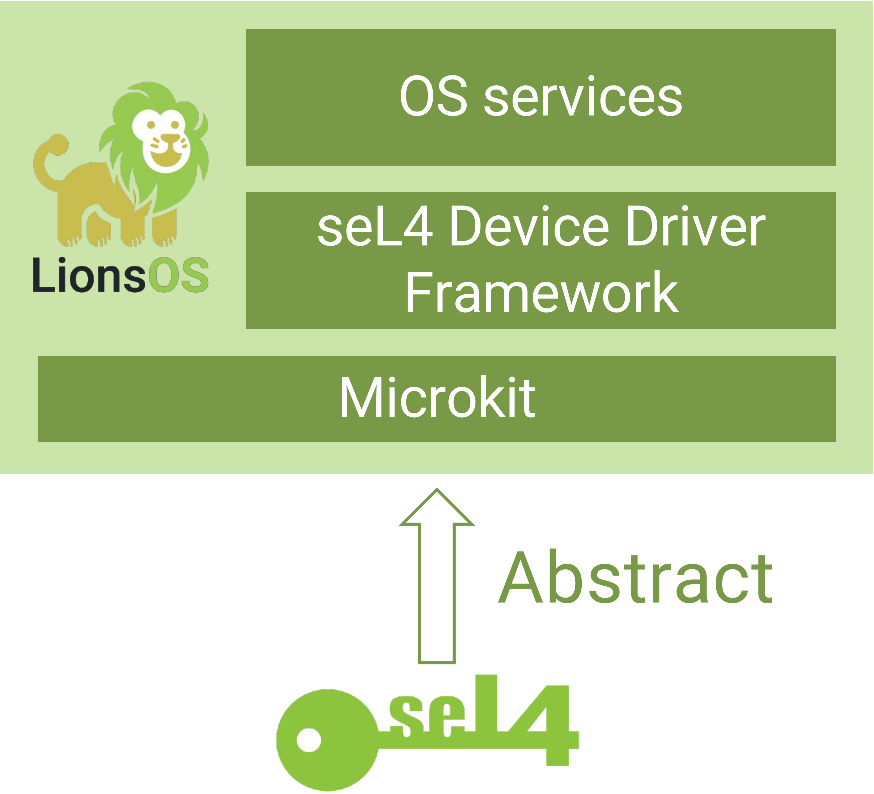
Radical simplicity:

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

Concurrency by distributing components

Use-case diversity by replacing components

Foundations: seL4 + Microkit + sDDF



LionsOS

OS services

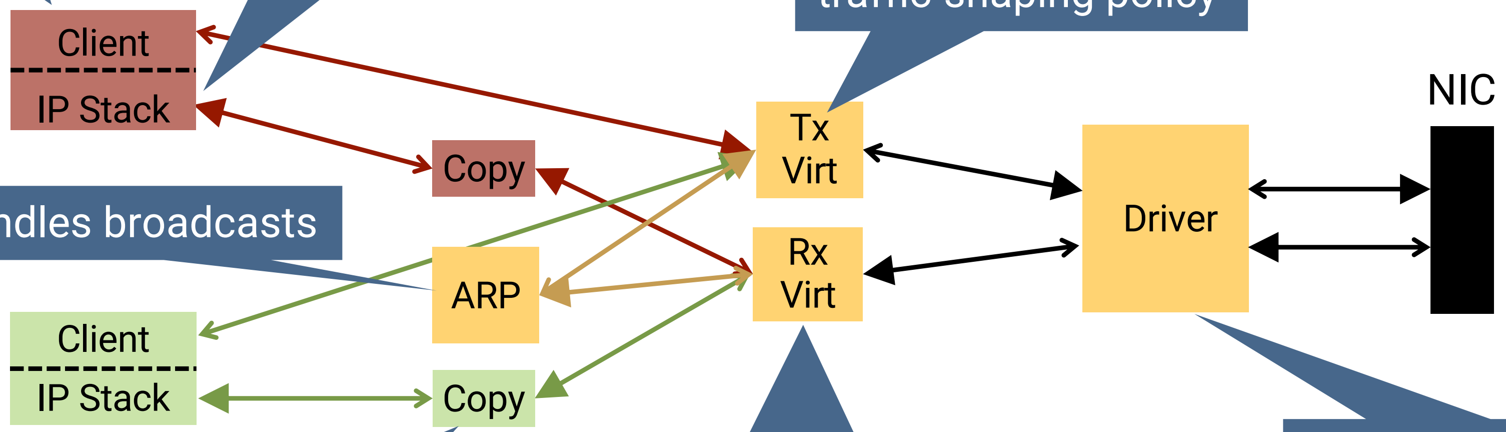
seL4 Device Driver Framework

Microkit

Abstract

seL4

Example: Networking Subsystem



Client can be a VM

IP stack is library – not in system's TCB!

Tx Virt encapsulates traffic-shaping policy

Strict separation of concerns!

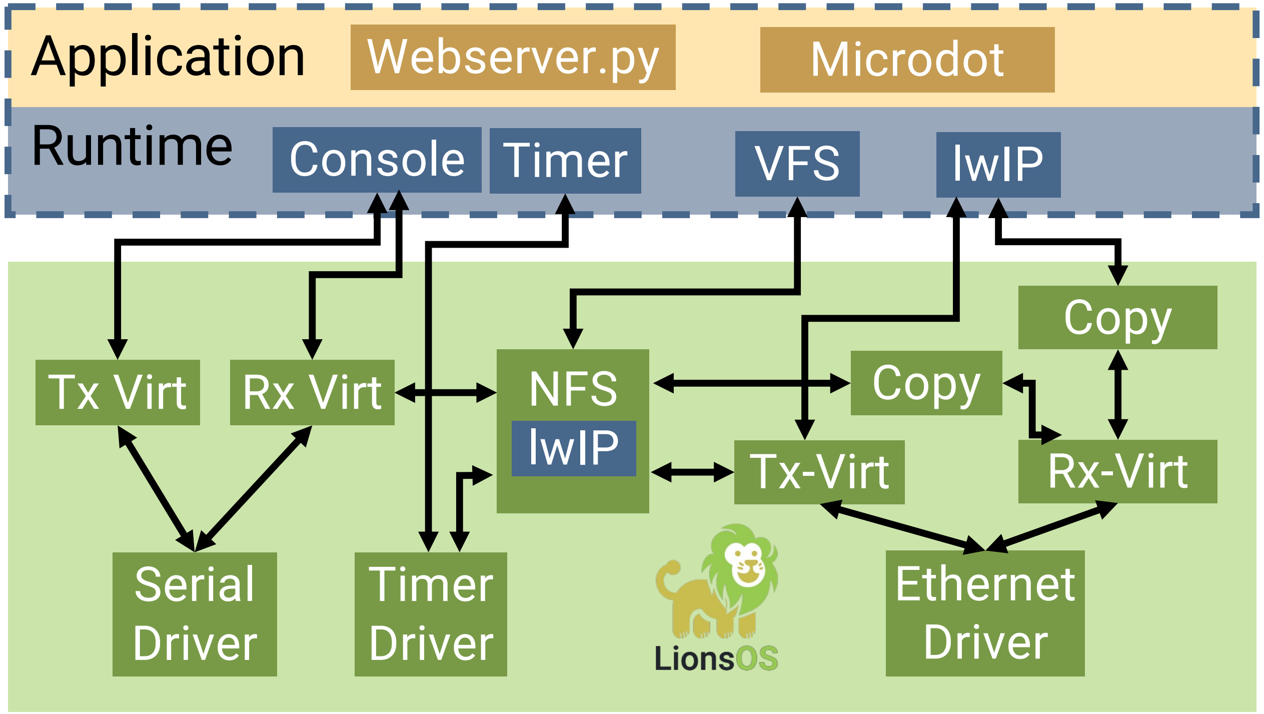
Handles broadcasts

Copier for security (if needed)

Virtualiser shares device, incl address mapping, cache maintanance

Translates HW-specific device interface to HW-independent device-class interface

Underneath https://sel4.systems/



Modules are:

- Small
- Single-threaded
- Asynchronous, zero-copy, shared-memory communication
- Location transparent
- Verification-friendly

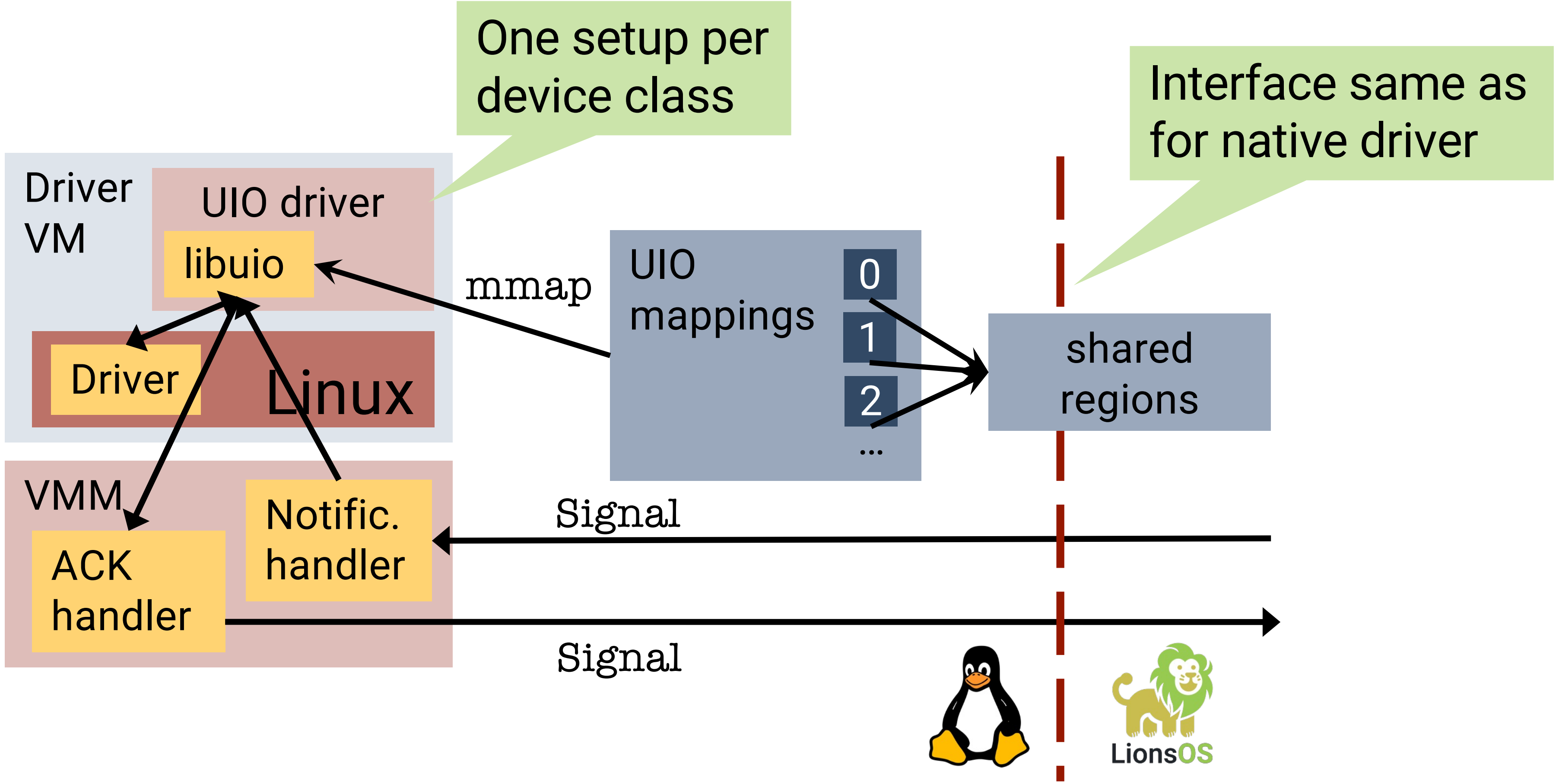
Web Server Code Sizes (all C)

Component	LoC	Library	LoC
Timer Driver	139	Microkit	368
Serial Driver	231	Serial queue	169
Serial Tx Virt	159	Eth queue	140
Serial Rx Virt	109	Filesys queue & protocol	268
Eth Driver	397		
Eth Tx Virt	107		
Eth Rx Virt	151	Coroutines	848
Eth Copier	73	LWIP	16,280
Monitor	1,188	NFS	45,707
LionsOS trusted	**3,545**	**Untrusted**	**62,356**
Web server app	7,246	MicroPython	402,554

Trusted:
- 13 modules/ libraries
- Av 270 LoC

Untrusted

Re-Using Unmodified Legacy Drivers



Driver-VM Cost

In progress: using same setup to develop LionsOS modules under Linux

Driver	Kernel	RAM Disk	Runtime	Total
Default	29 MiB	6.7 MiB	70 MiB	106 MiB
Audio	3 MiB	2.4 MiB	18 MiB	23 MiB
Block	3 MiB	0.05 MiB	12 MiB	15 MiB

Optimised

Effort:

- Few days to set up UIO driver
- Total ≈ 2 weeks / device class

Status: Microkit

Available:

- Support for Arm, RISC-V
- Graphical editor
- GDB, profiling

Close to merging:

- x86 support
- Using CapDL loader

Should be sufficient for closing RFC

Multi-kernel support in progress:

- Demo version end of this month

Further out:

- Core on-/off-lining
  - needs kernel changes
- Template PDs
  - Dynamic assignment of channels, memory regions
  - MAC augmented by DAC

Status: sDDF Device Classes

Native

- Serial
- Timer
- Clock
- PinMux
- I^{2}C
- SPI
- Ethernet
- SDHC storage
- NFC card reader

Driver VMs

- GPU (2D)
- Ethernet
- Storage
- Sound (ALSA)
- Video capture in progress

- Implementations on Arm
- Subset on x86, RISC-V
- Rust drivers
- **sDDF no longer tied to Microkit**

Status: OS Services

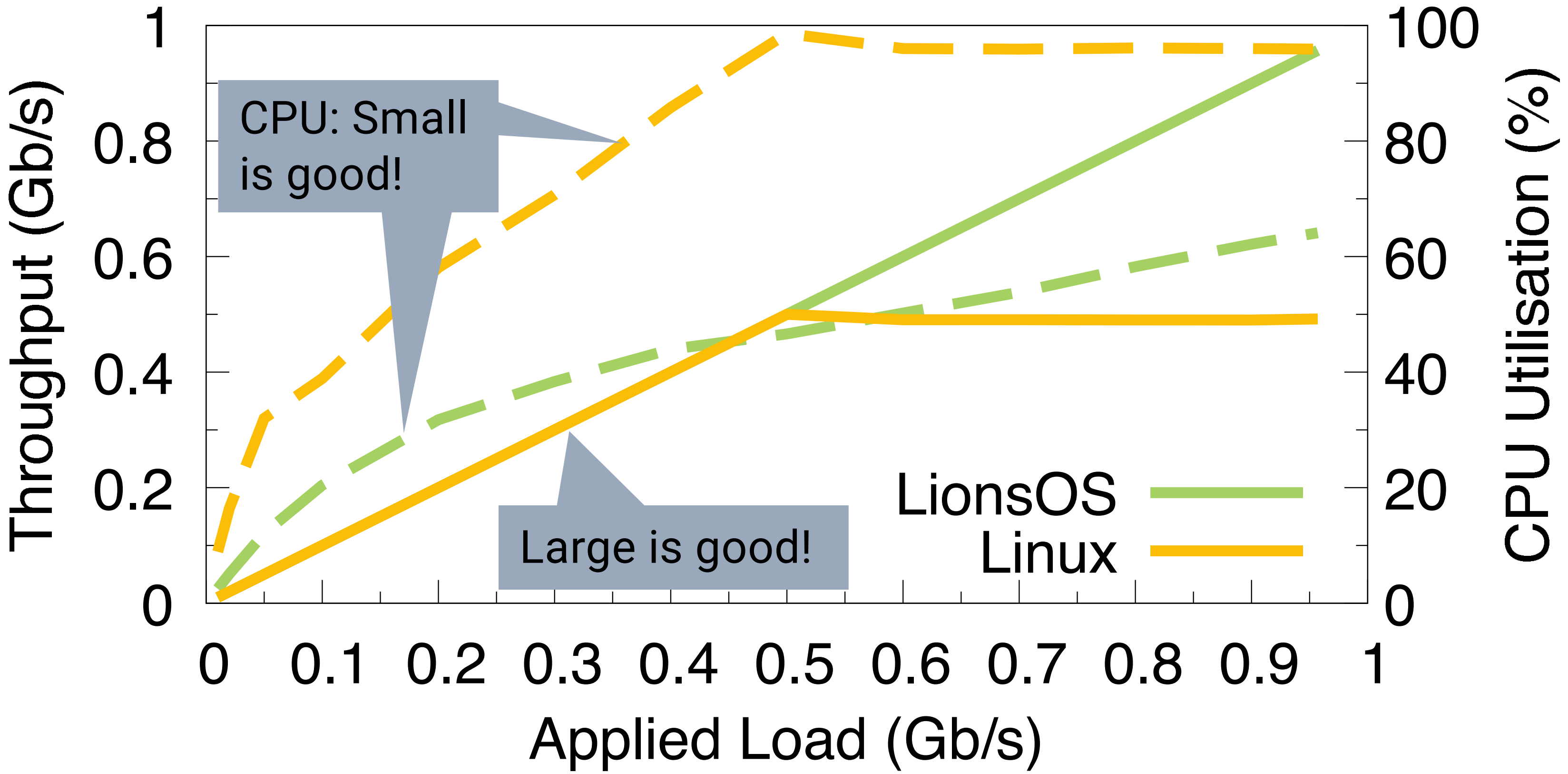
Supported:

- Networking – native (async)
- Storage (ported file system) – native (async) and blocking (Posix-like)
- Debugging (GDB)
- Profiling (preliminary)
- Multicore – full location transparency of OS modules

Work in progress:

- x86: to do IOMMU & virtualisation support
- Multikernel support (mostly abstracted by Microkit)

Performance: i.MX8MQ, 1Gb/s Eth, UDP

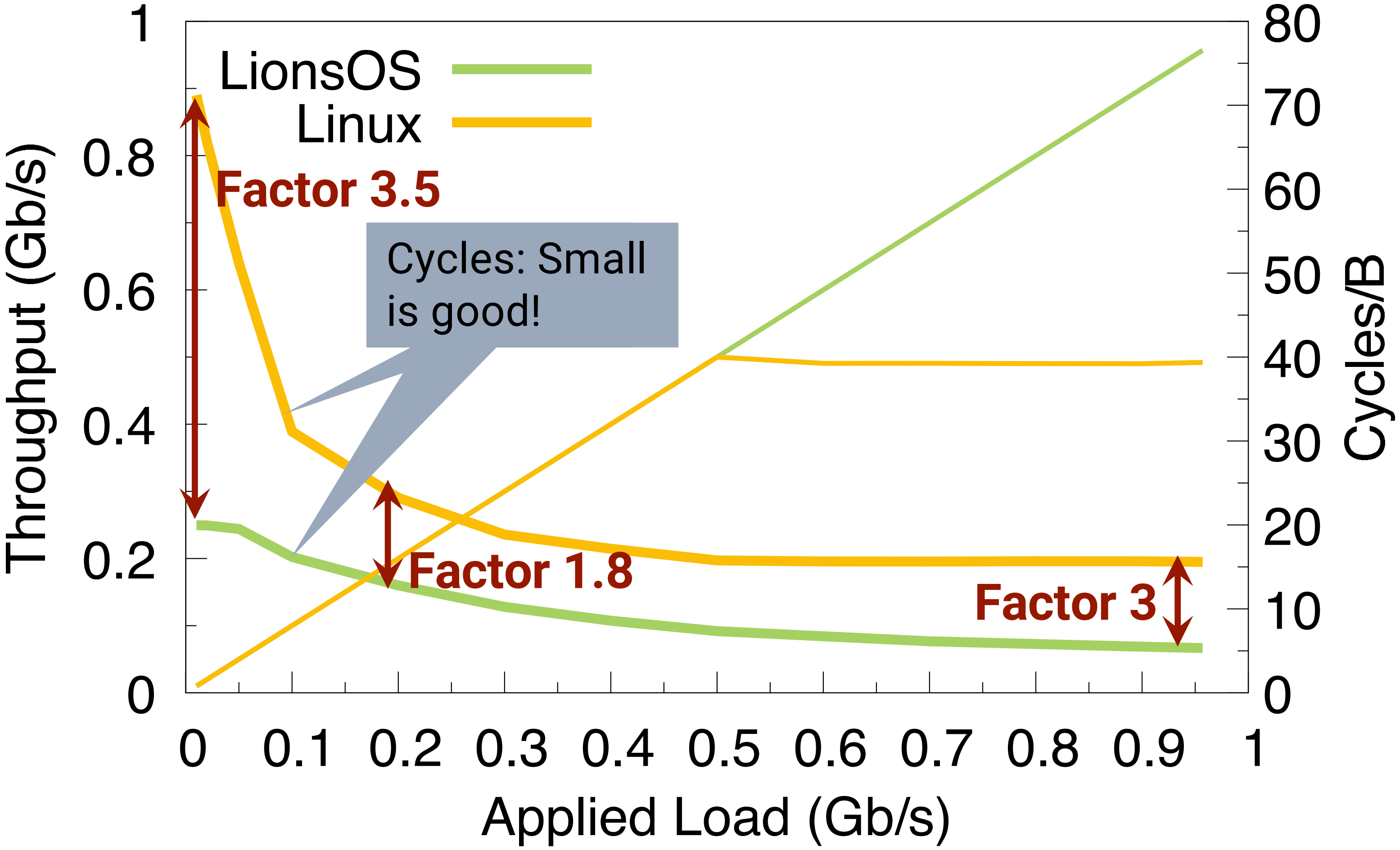


CPU: Small is good!

Large is good!

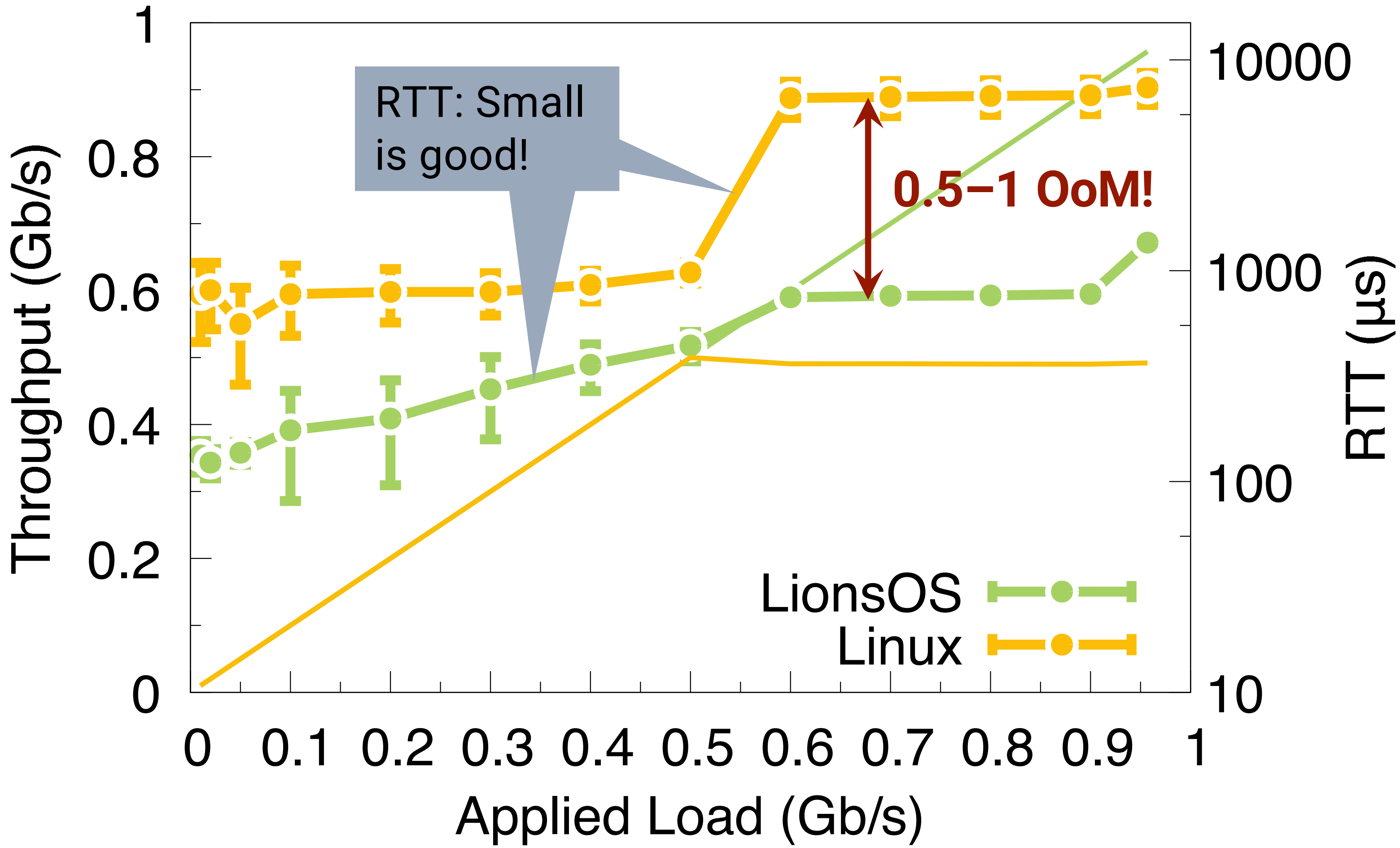
Single-core configuration

Performance: Processing Cost per Byte



Cycles: Small is good!

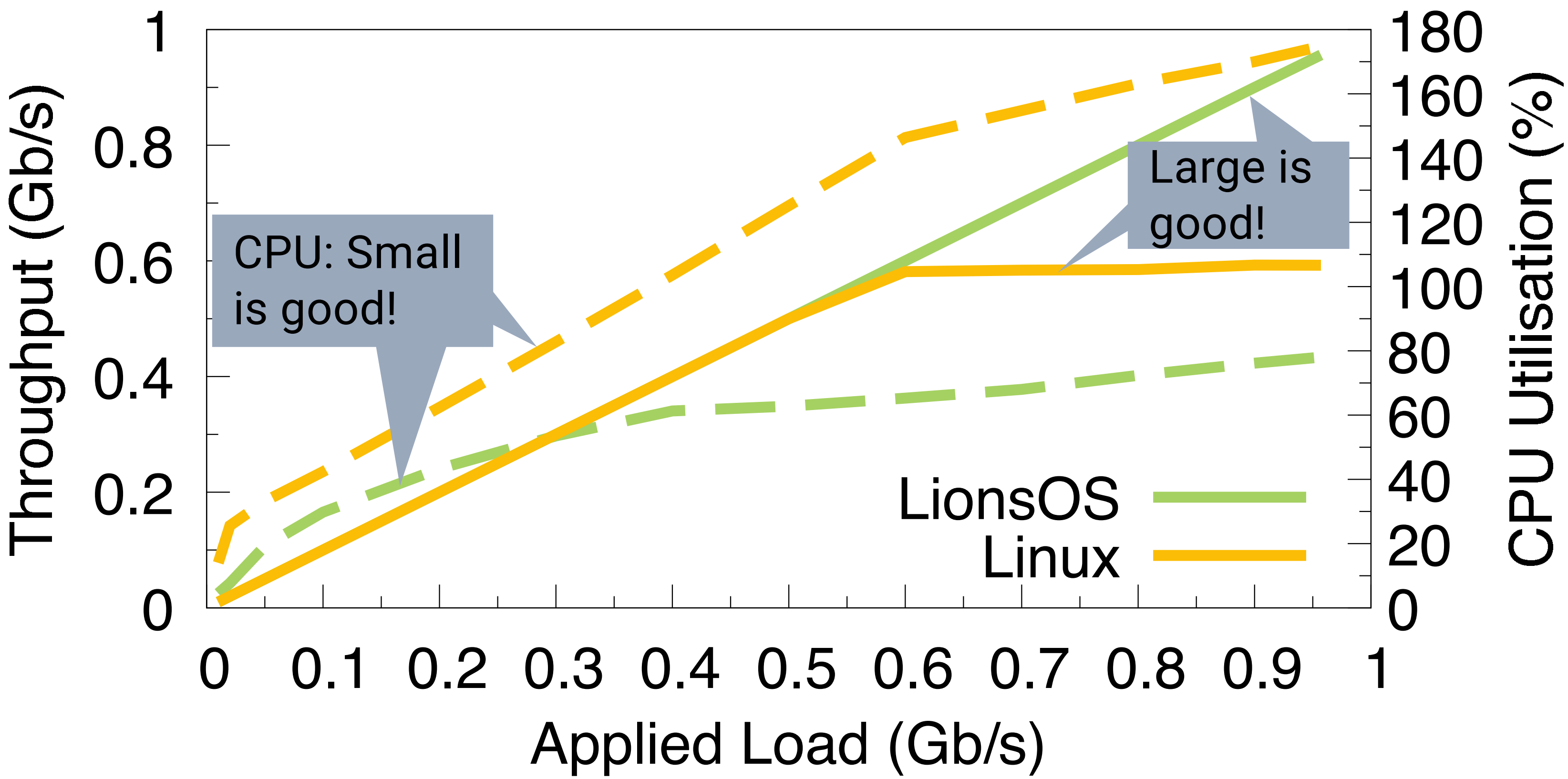
Performance: Round-Trip Times



RTT: Small is good!

0.5–1 OoM!

Performance: i.MX8MQ, 1Gb/s Eth, UDP

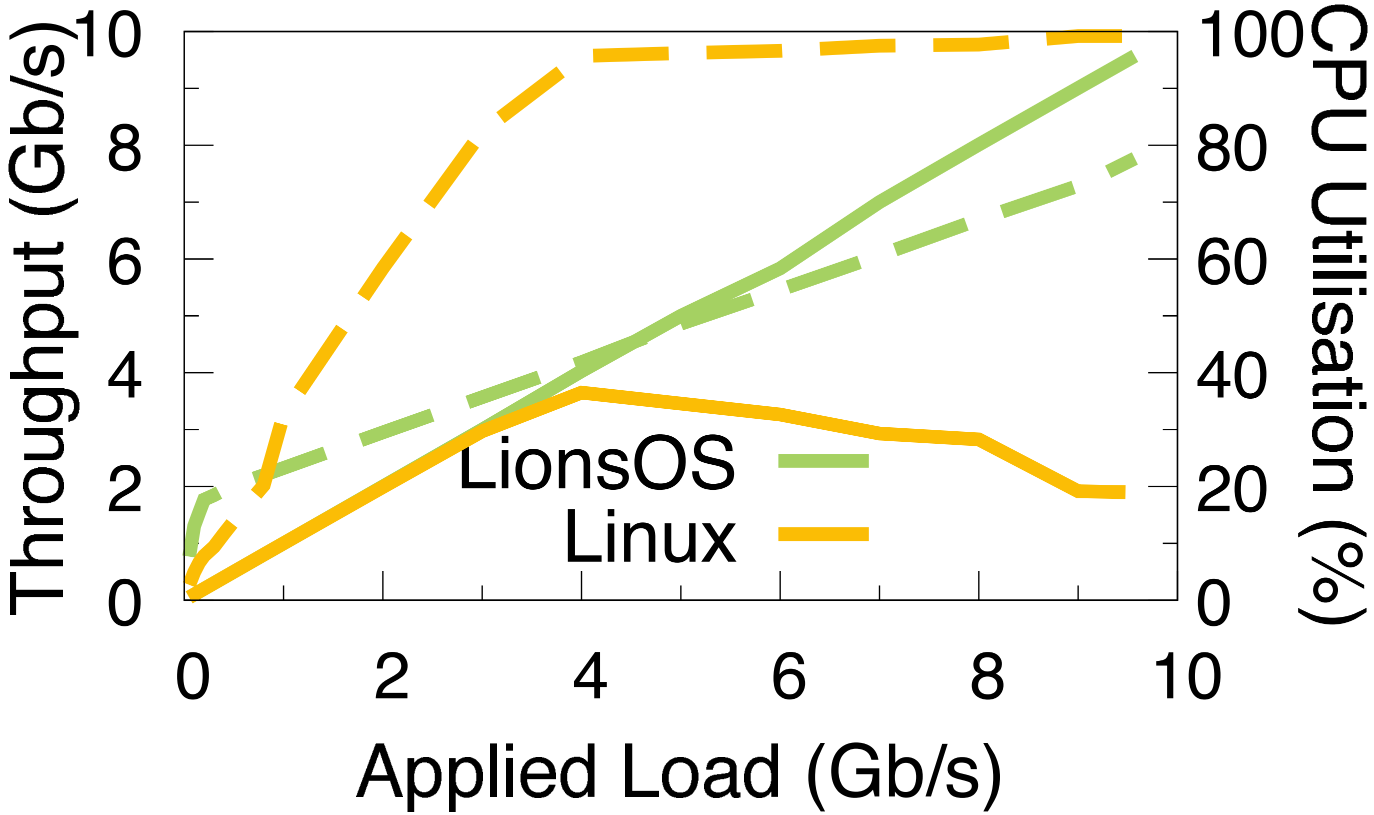


CPU: Small is good!

Large is good!

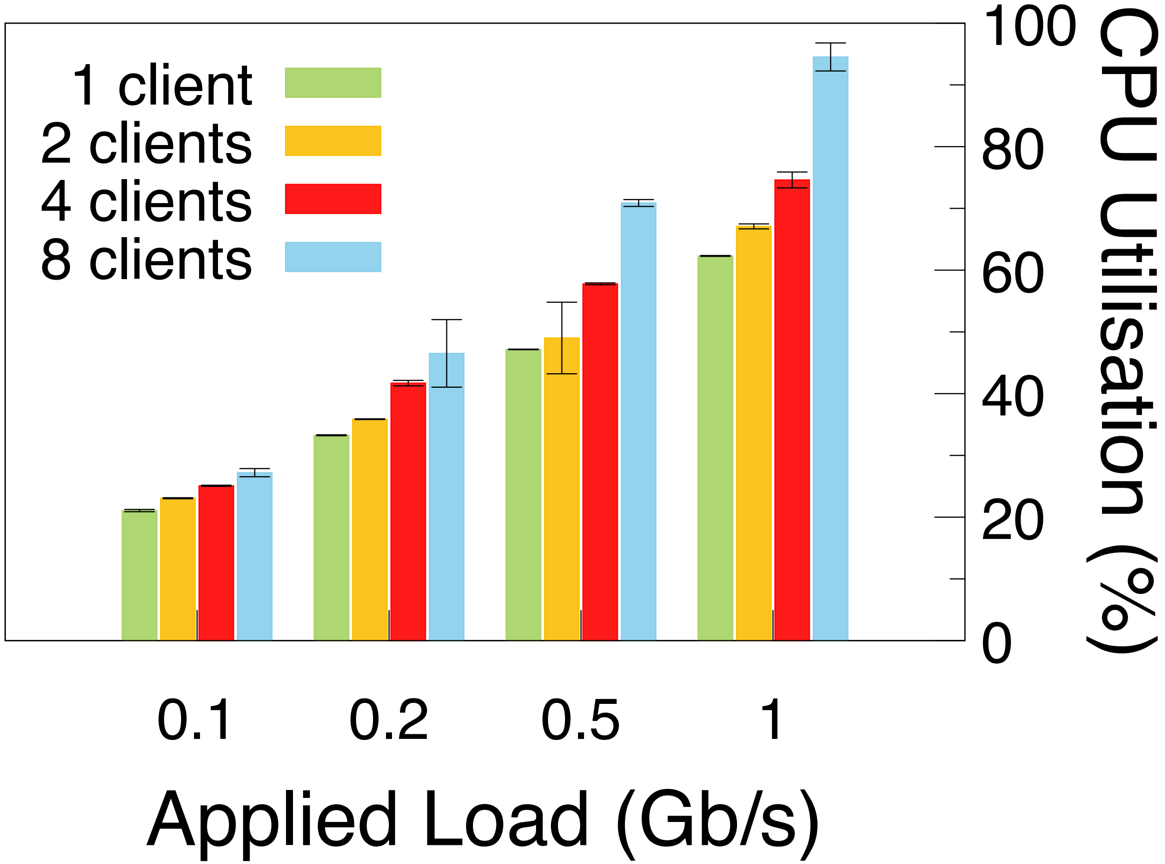
Multicore configuration

Performance: x86, 10Gb/s Eth, UDP



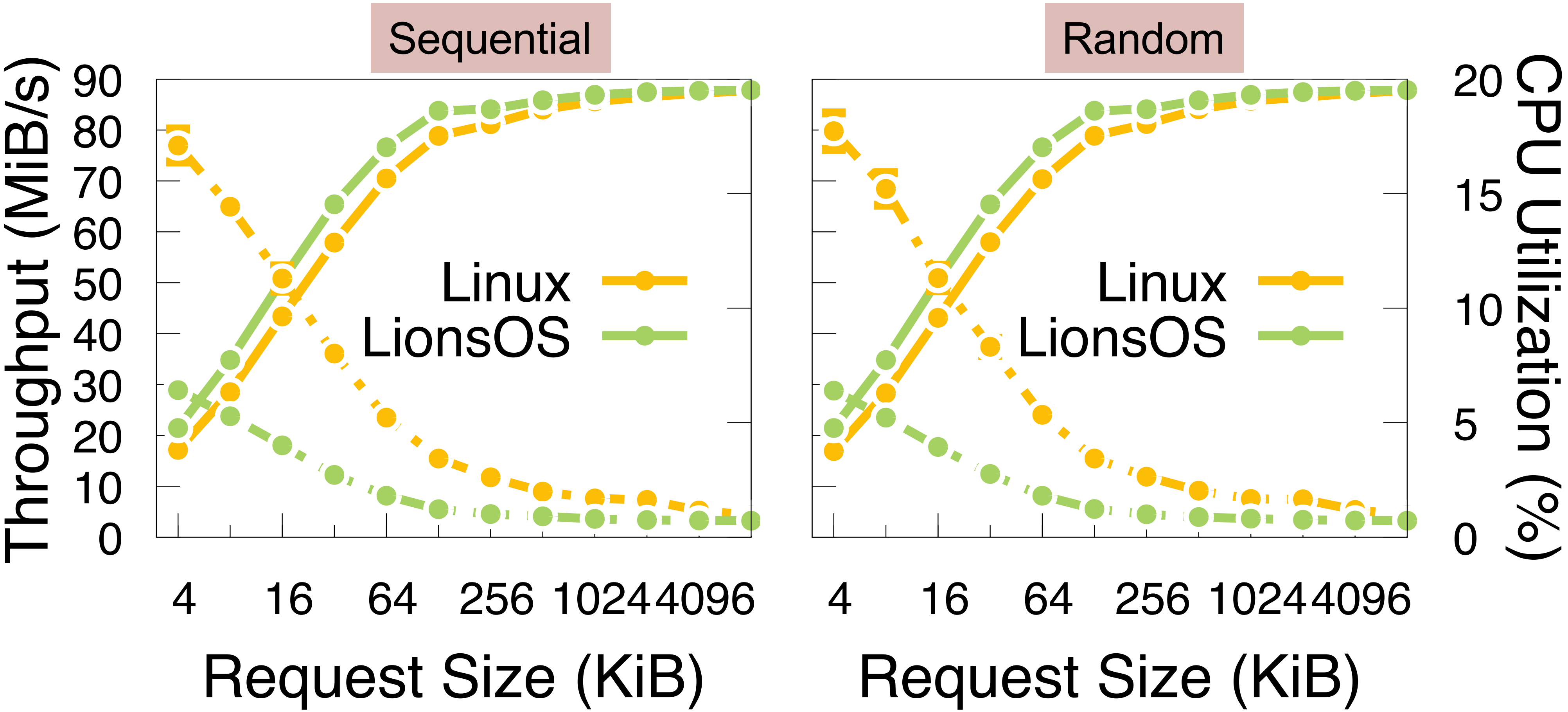
Single-core configuration

Client Scalability (i.MX8M)

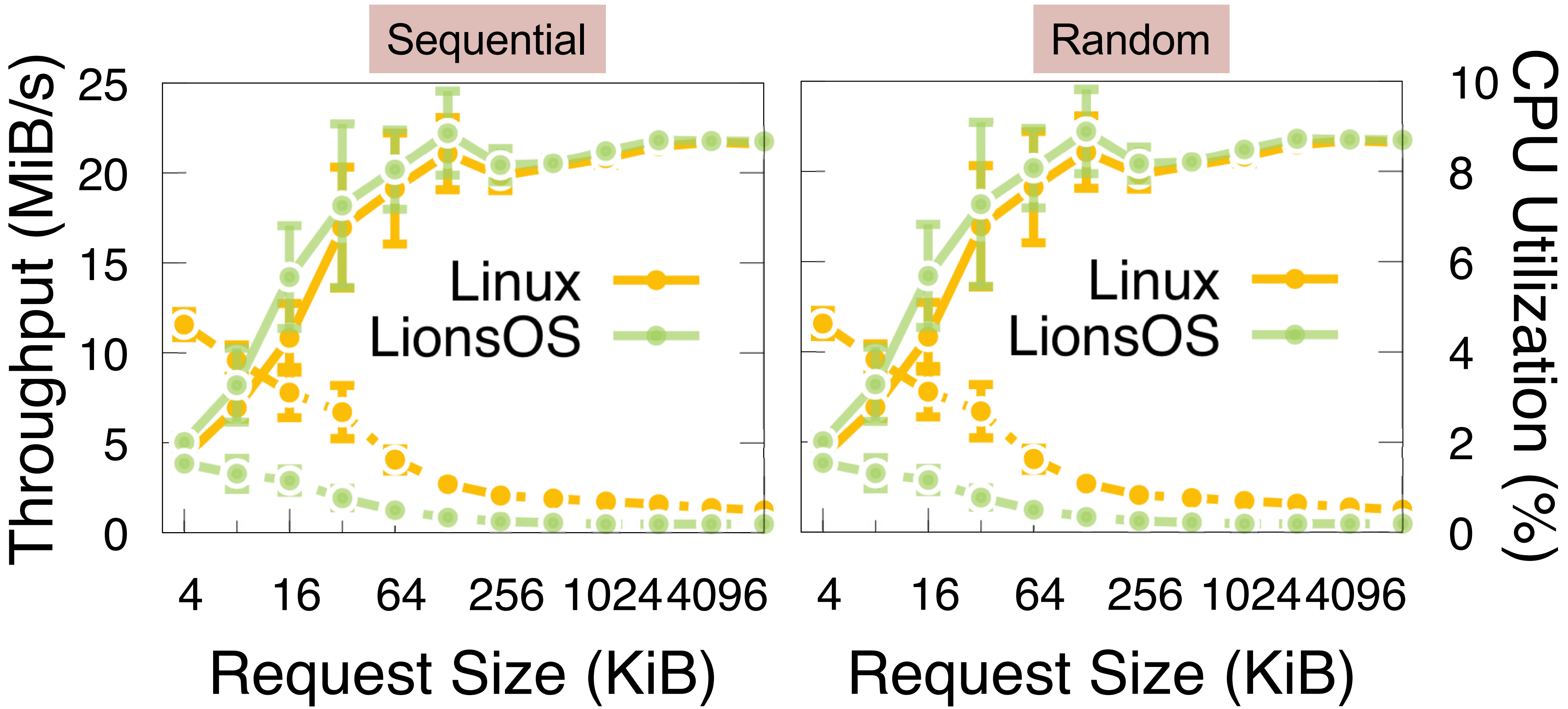


Unicore, equal bandwidth per client

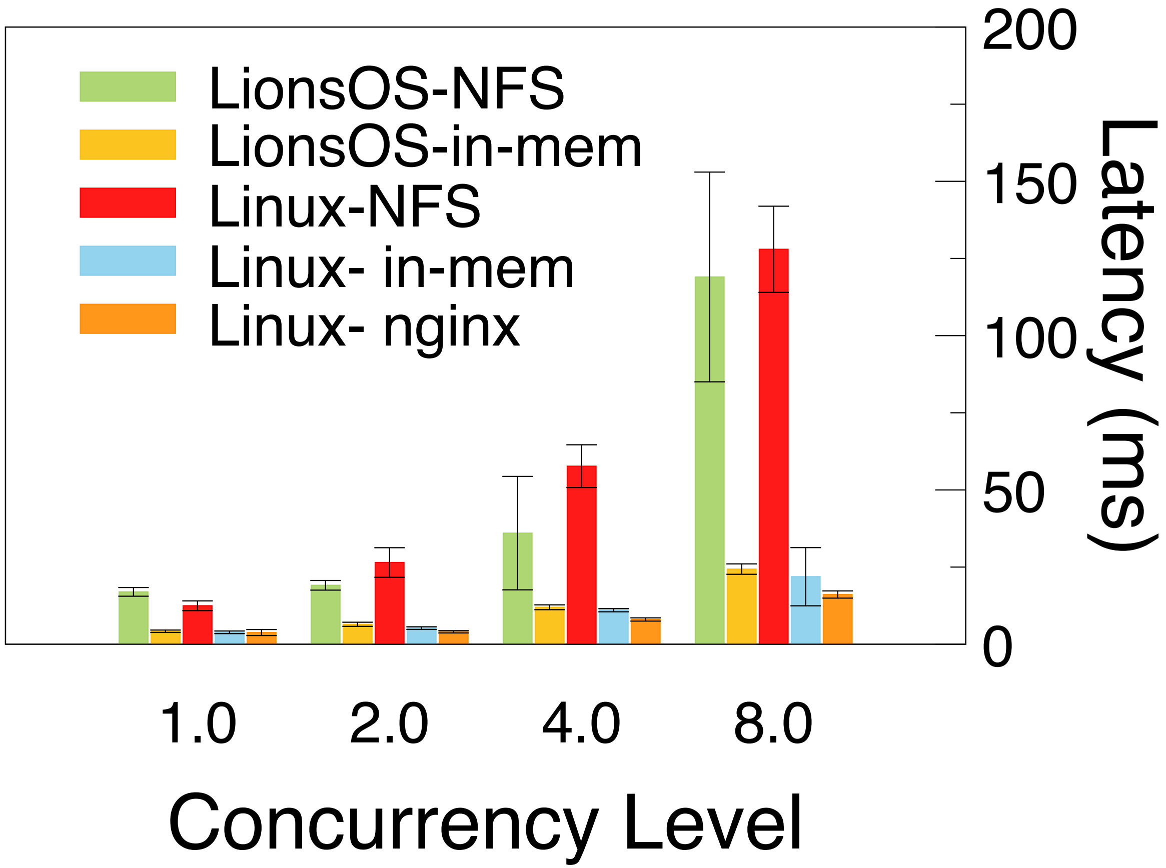
Performance: Sandisk, Arm, Read



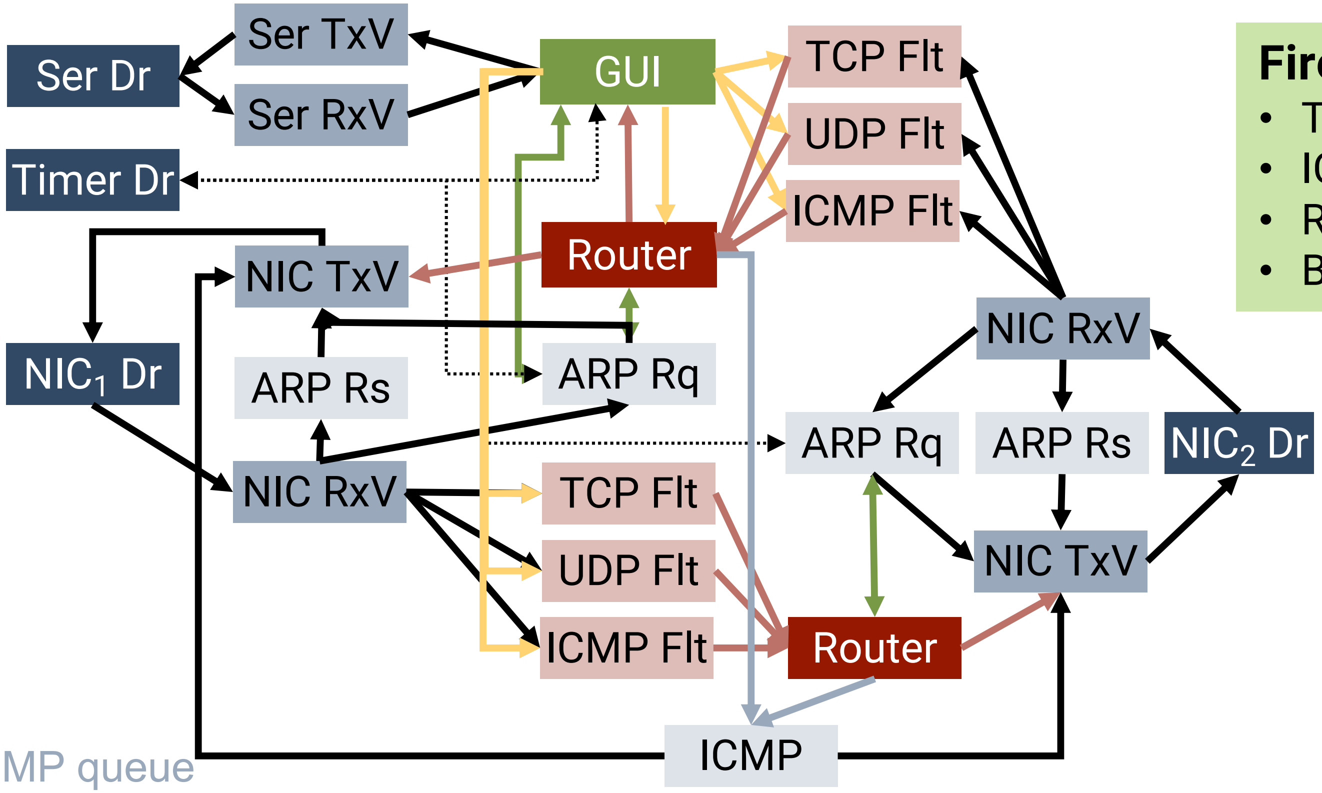
Performance: Sandisk, Arm, Write



Web Server System Benchmark



Firewall Project



ICMP queue
Firewall queue
Software system

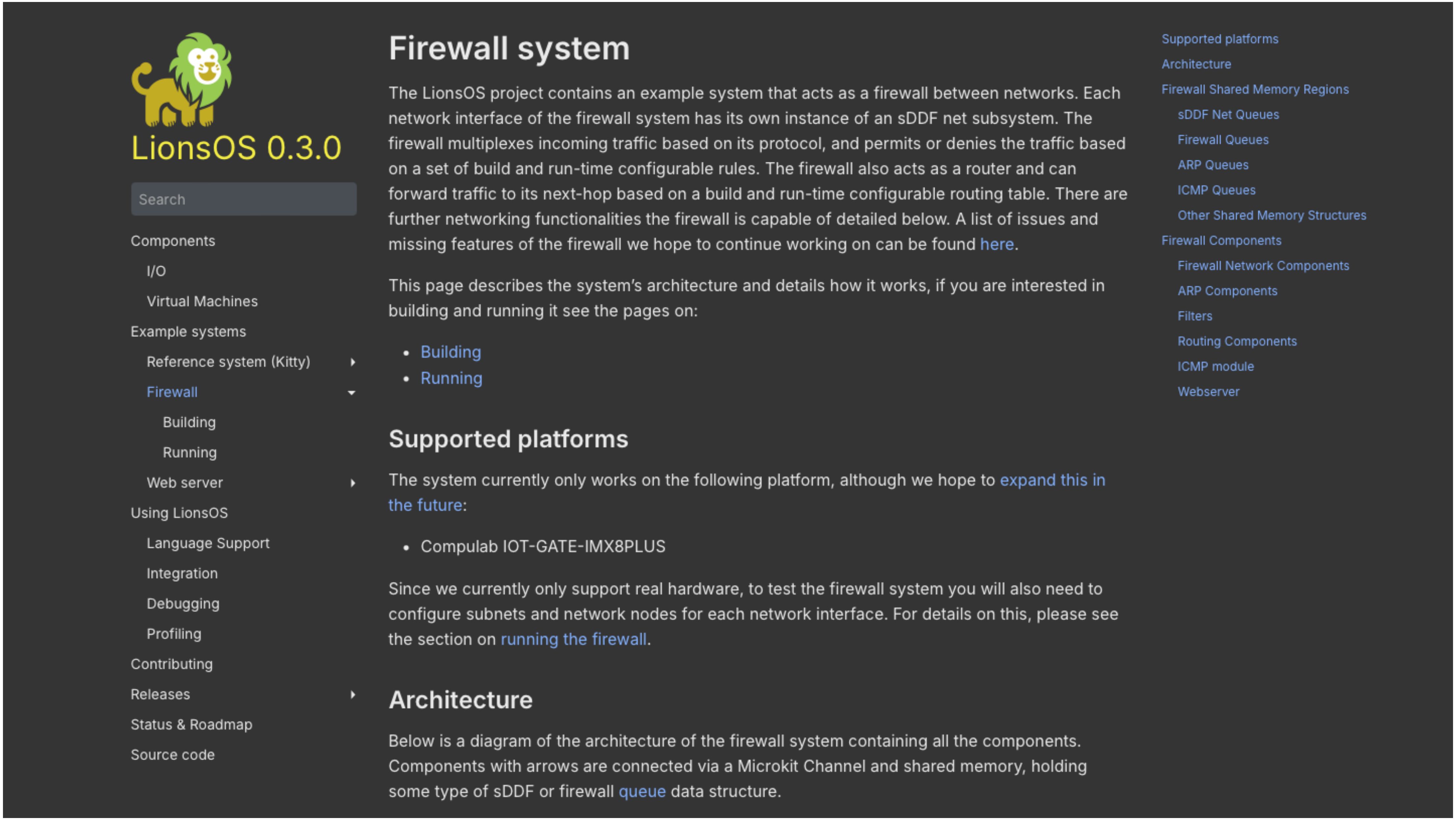
Firewall has:

- TCP connection tracking
- ICMP, TCP and UDP
- Rudimentary GUI, web interface
- Basic routing

Firewall wants:

- QoS queuing for VOIP, video…
- NAT and port forwarding
- More complete ICMP
- SNMP monitoring
- Spanning Tree protocols
- Port to commercial platform
- Better GUI…

https://lionsos.org/docs/examples/firewall/



LionsOS 0.3.0

Search

Components
- I/O
- Virtual Machines

Example systems
- Reference system (Kitty)
- Firewall
  - Building
  - Running
- Web server

Using LionsOS
- Language Support
- Integration
- Debugging
- Profiling

Contributing

Releases

Status & Roadmap

Source code

Firewall system

The LionsOS project contains an example system that acts as a firewall between networks. Each network interface of the firewall system has its own instance of an sDDF net subsystem. The firewall multiplexes incoming traffic based on its protocol, and permits or denies the traffic based on a set of build and run-time configurable rules. The firewall also acts as a router and can forward traffic to its next-hop based on a build and run-time configurable routing table. There are further networking functionalities the firewall is capable of detailed below. A list of issues and missing features of the firewall we hope to continue working on can be found here.

This page describes the system's architecture and details how it works, if you are interested in building and running it see the pages on:

- Building
- Running

Supported platforms

The system currently only works on the following platform, although we hope to expand this in the future:

- Compulab IOT-GATE-IMX8PLUS

Since we currently only support real hardware, to test the firewall system you will also need to configure subnets and network nodes for each network interface. For details on this, please see the section on running the firewall.

Architecture

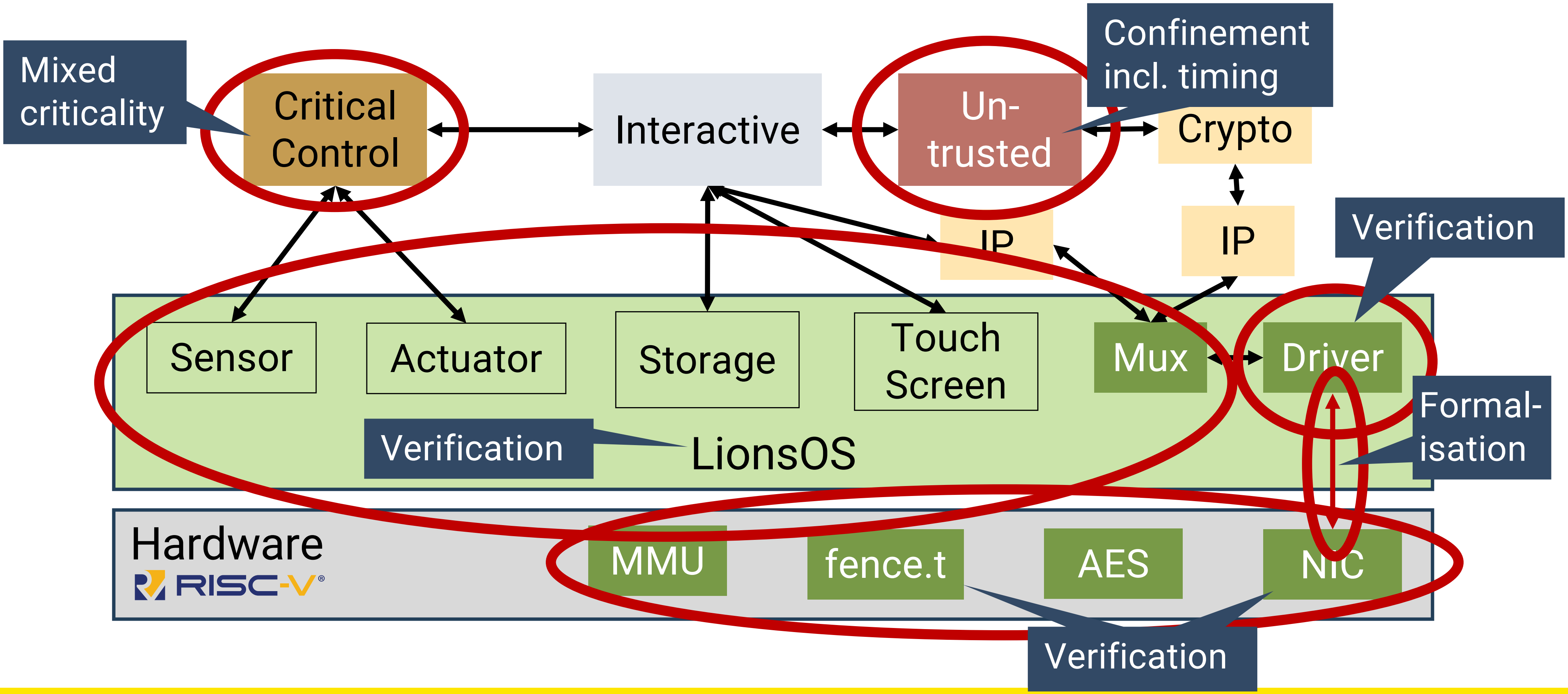
Below is a diagram of the architecture of the firewall system containing all the components. Components with arrows are connected via a Microkit Channel and shared memory, holding some type of sDDF or firewall queue data structure.

Supported platforms
Architecture
Firewall Shared Memory Regions
- sDDF Net Queues
- Firewall Queues
- ARP Queues
- ICMP Queues
- Other Shared Memory Structures

Firewall Components
- Firewall Network Components
- ARP Components
- Filters
- Routing Components
- ICMP module
- Webserver

Verification Agenda

Agenda for Next 2(–3) Years

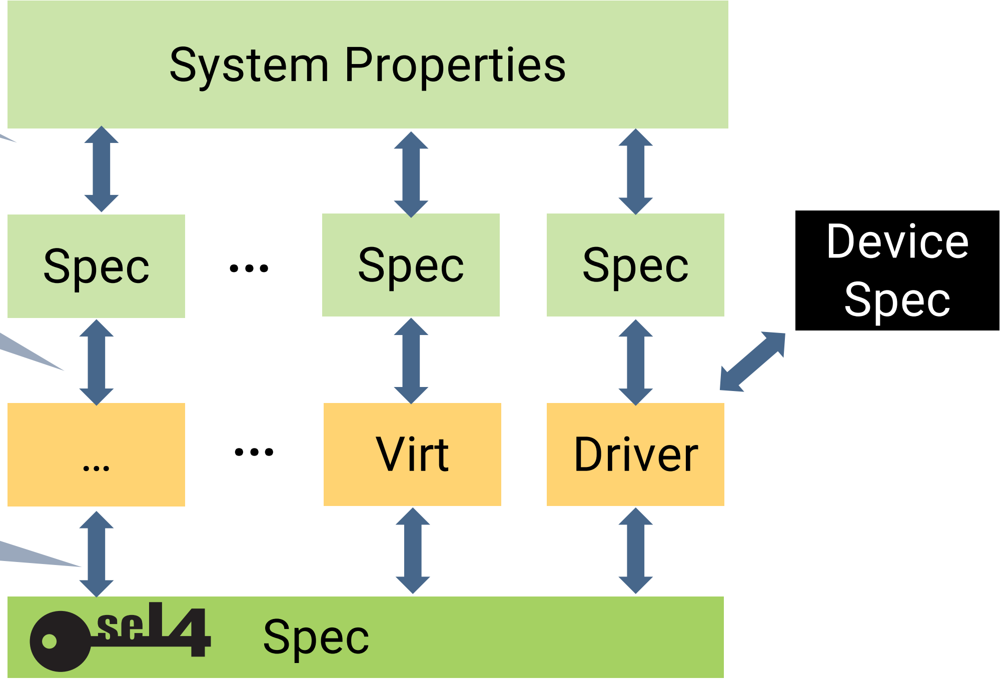


System Verification Challenges

Verify

Verify Component

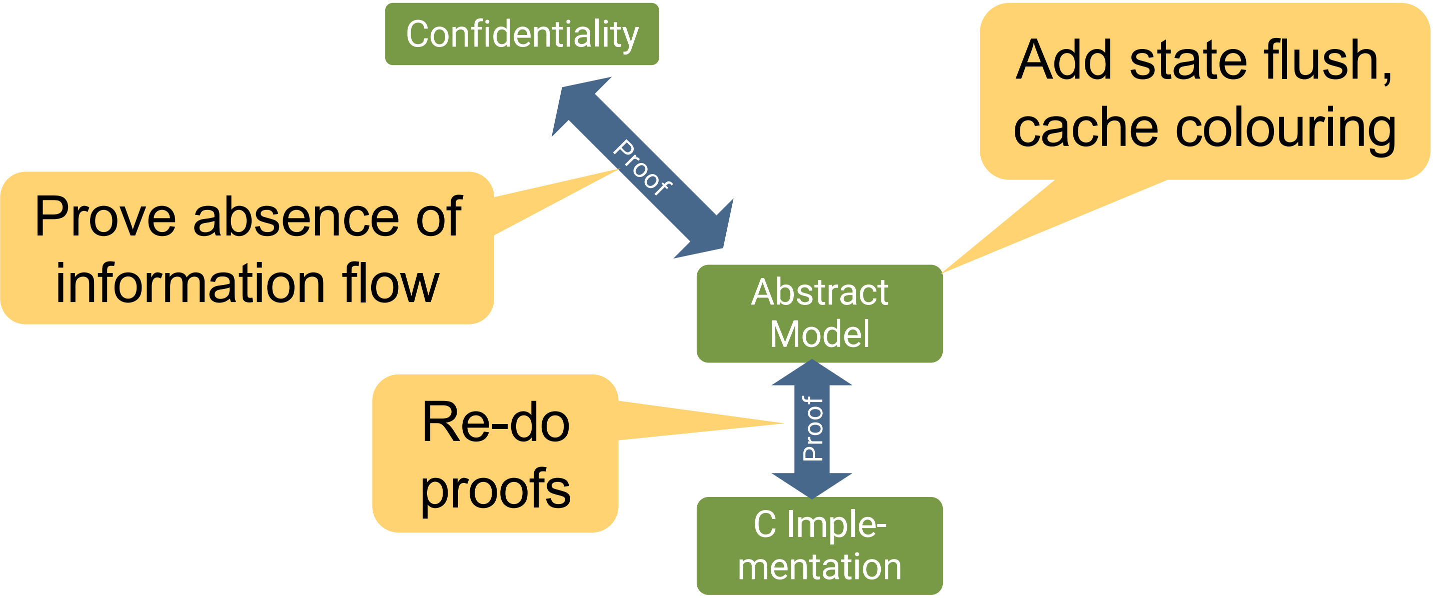
Connect to seL4 Spec



seL4 Time Protection Progress

Preventing micro-architectural timing channels

- Usable system model: allow overt cross-domain channels
- Verification progressing



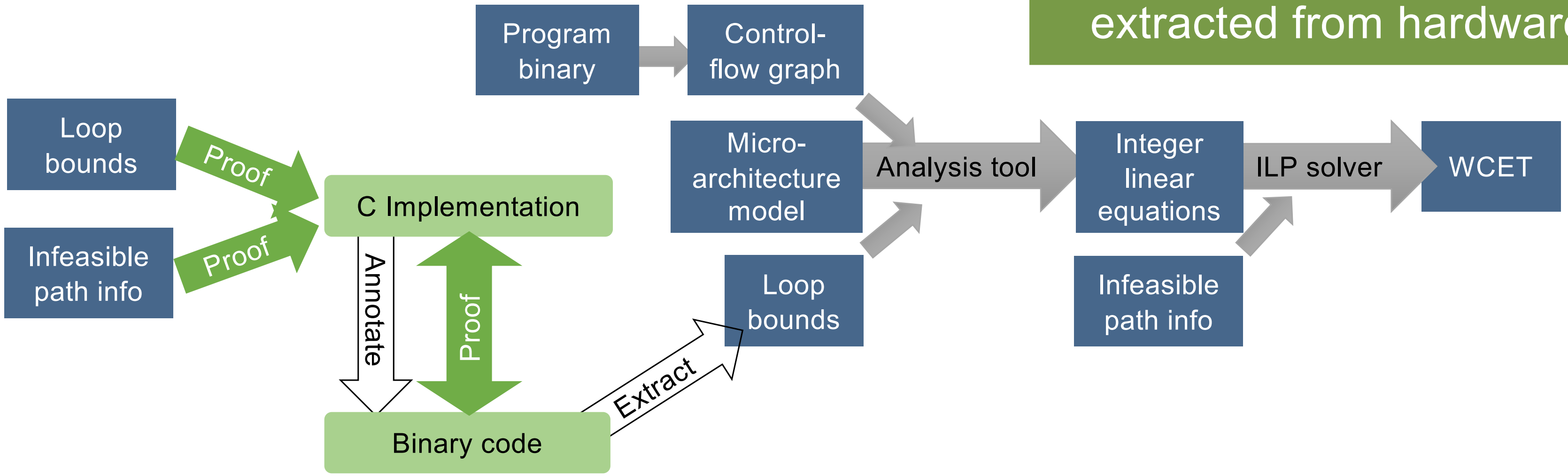
WCET Analysis

Sound worst-case execution times

- Originally done 2011–17
- 32-bit Arm only, no MCS
- Bit-rotted

Now re-done using Heptane

- 64-bit
- RISC-V
- MCS kernel
- Place-holder latencies
- Accurate latencies to be extracted from hardware



Pancake

Device Driver Dilemma

seL4 is one-off, justifies cost

High seL4 verification costs partially due to C language

Drivers are commodity, must be cheap!

Drivers are low-level, need C-like language

Better language would reduce cost

Idea:

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

sDDF

- Well-defined semantics
- Memory-safe

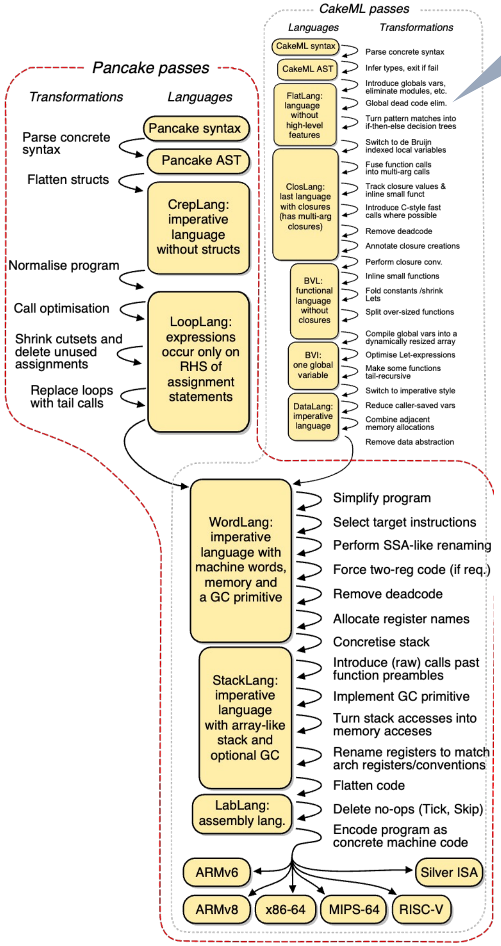
- Verified compiler
- de-compilation
- ATP

PANCAKE
A Language for
Verified Systems Programming

Approach:

- Re-use lower part of CakeML compiler stack
- Get verified Pancake compiler quickly
- Retain mature framework/ecosystem

CakeML



Pancake passes
Transformations
Languages
Pancake syntax
Parse concrete syntax
Pancake AST
Flatten structs
CrepLang: imperative language without structs
Normalise program
Call optimisation
Shrink cutsets and delete unused assignments
LoopLang: expressions occur only on RHS of assignment statements
Replace loops with tail calls
CakeML passes
Languages
Transformations
CakeML syntax
Parse concrete syntax
CakeML AST
Infer types, exit if fail
Introduce globals vars, eliminate modules, etc.
FlatLang: language without high-level features
Global dead code elim.
Turn pattern matches into if-then-else decision trees
Switch to de Bruijn indexed local variables
ClosLang: last language with closures (has multi-arg closures)
Fuse function calls into multi-arg calls
Track closure values & inline small funct
Introduce C-style fast calls where possible
Remove deadcode
Annotate closure creations
Perform closure conv.
BVL: functional language without closures
Inline small functions
Fold constants /shrink Lets
Split over-sized functions
Compile global vars into a dynamically resized array
BVI: one global variable
Optimise Let-expressions
Make some functions tail-recursive
Switch to imperative style
DataLang: imperative language
Reduce caller-saved vars
Combine adjacent memory allocations
Remove data abstraction
WordLang: imperative language with machine words, memory and a GC primitive
Simplify program
Select target instructions
Perform SSA-like renaming
Force two-reg code (if req.)
Remove deadcode
Allocate register names
Concretise stack
Introduce (raw) calls past function preambles
StackLang: imperative language with array-like stack and optional GC
Implement GC primitive
Turn stack accesses into memory acceses
Rename registers to match arch registers/conventions
Flatten code
LabLang: assembly lang.
Delete no-ops (Tick, Skip)
Encode program as concrete machine code
ARMv6
Silver ISA
ARMv8
x86-64
MIPS-64
RISC-V

Pancake Progress

Functionality:

- Shared memory support – eliminates many FFIs
- 16-bit and 32-bit load/stores (64-bit archs) – eliminates more FFIs
- Global variables
- Shape checking (word vs struct)

Usability:

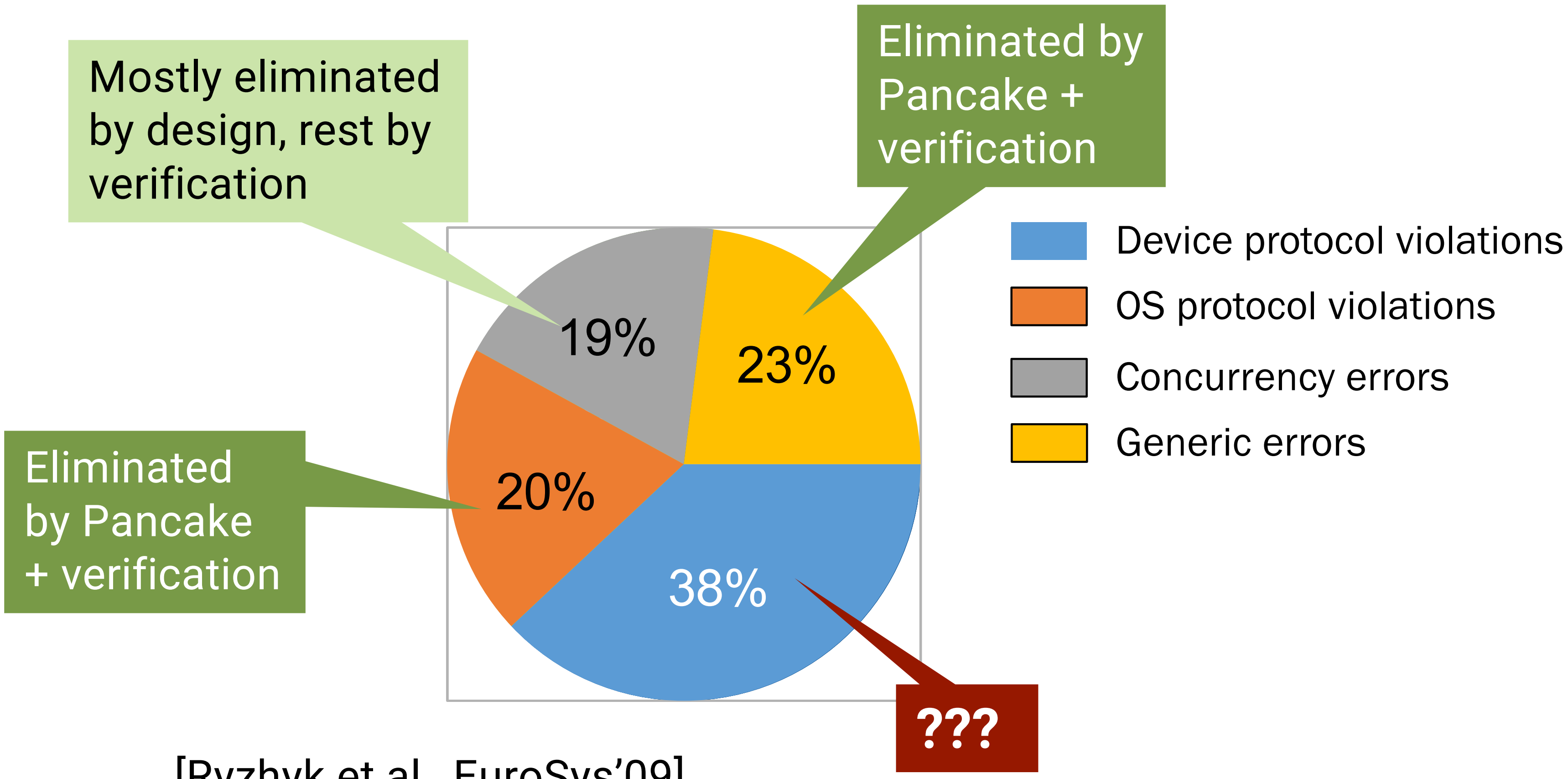
- Very few FFIs needed (mostly memory sync instructions)
- Performance within 15–20% of C (dominated by FFI overheads)
- Re-written most Maaxboard drivers
- `libmicrokit` re-write in progress

Pancake To Come

- Function inlining
- More compiler optimisations (eg memcpy)
- Decompilation into logic
- Hoare logic (see Junming’s talk)
- Verified transpiler (see Junming’s talk)

High-Assurance Device Spec

Device Driver Bugs in Linux



Mostly eliminated by design, rest by verification

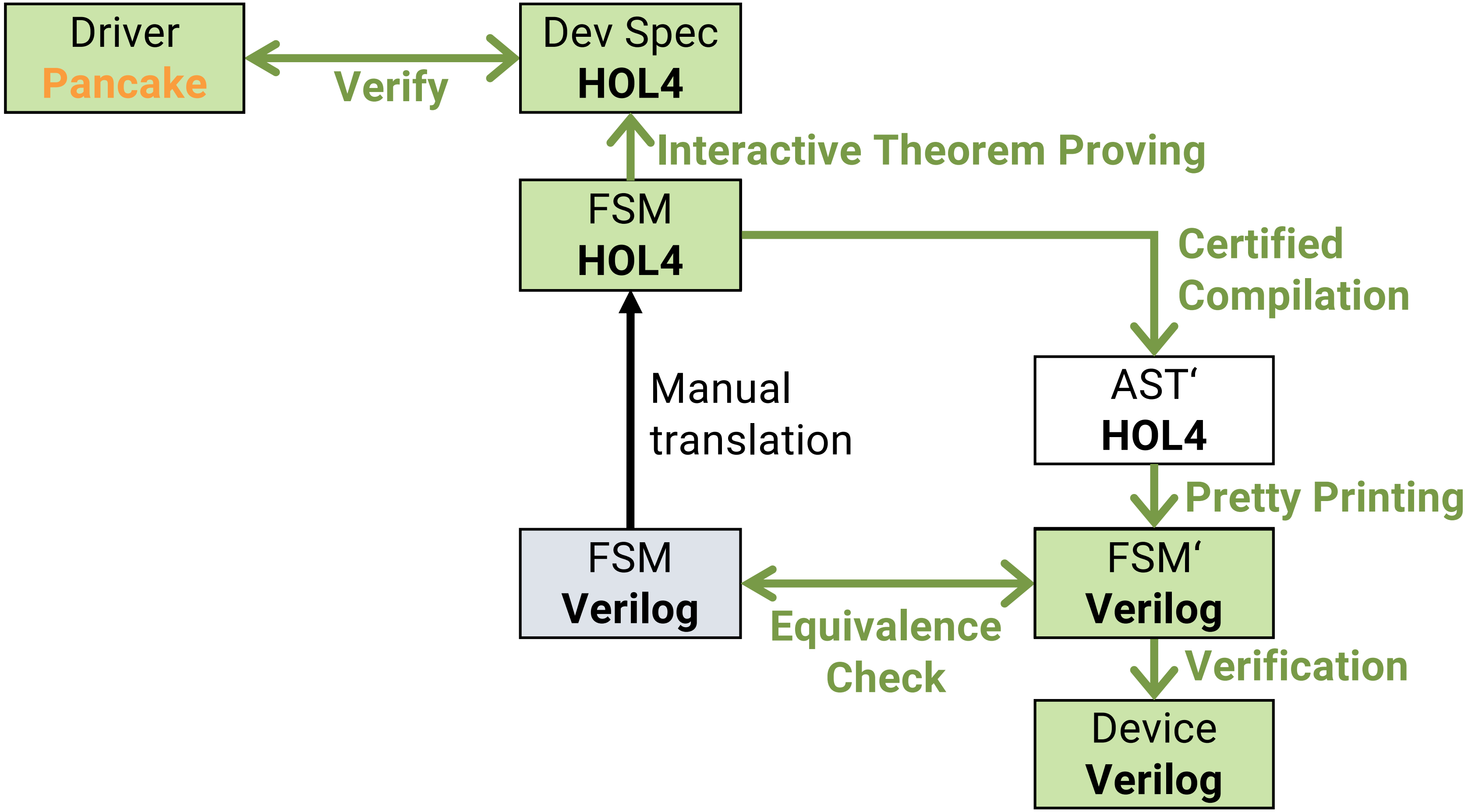
Eliminated by Pancake + verification

Eliminated by Pancake + verification

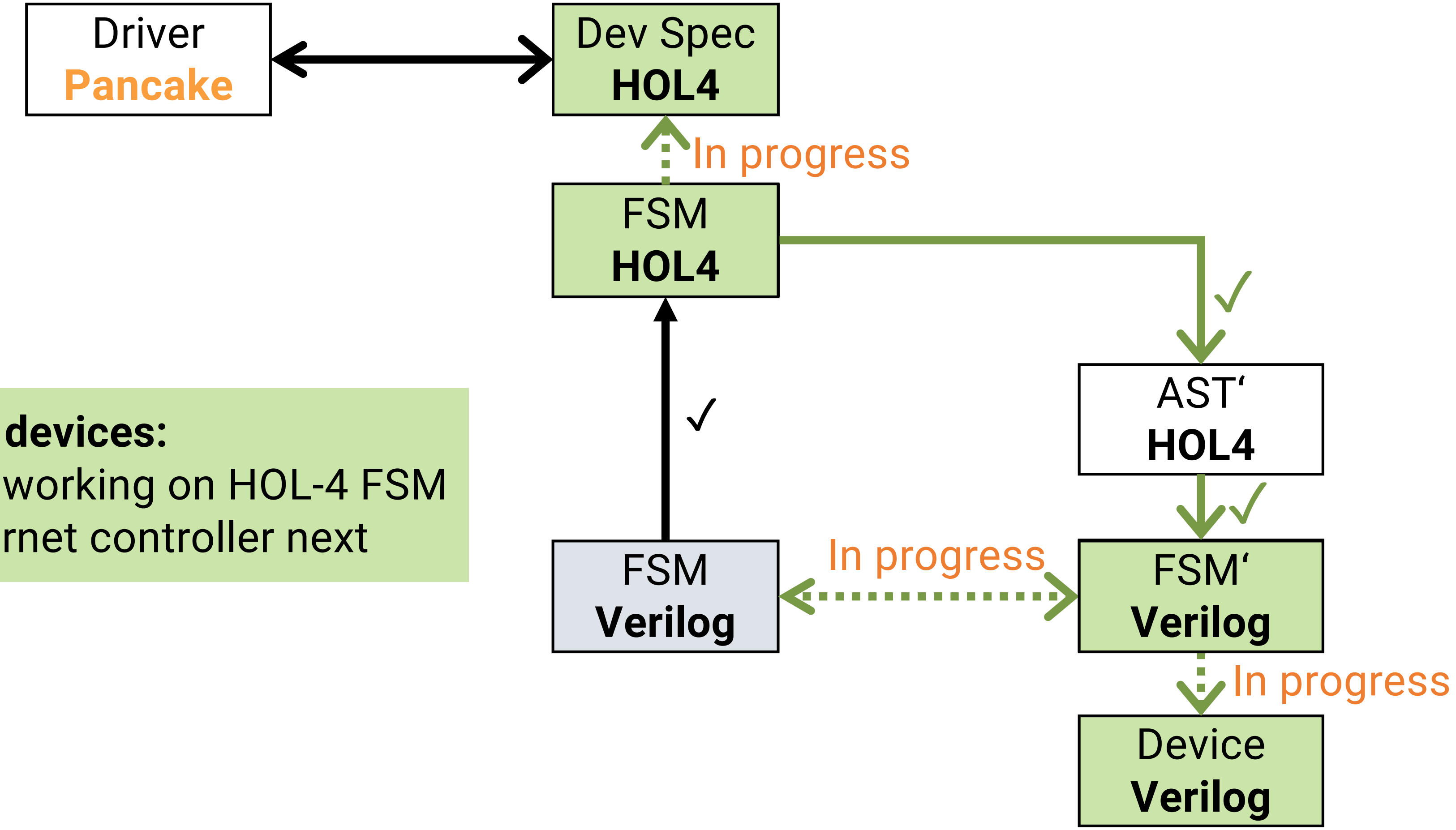
???

[Ryzhyk et al., EuroSys’09]

Correct Device Specifications



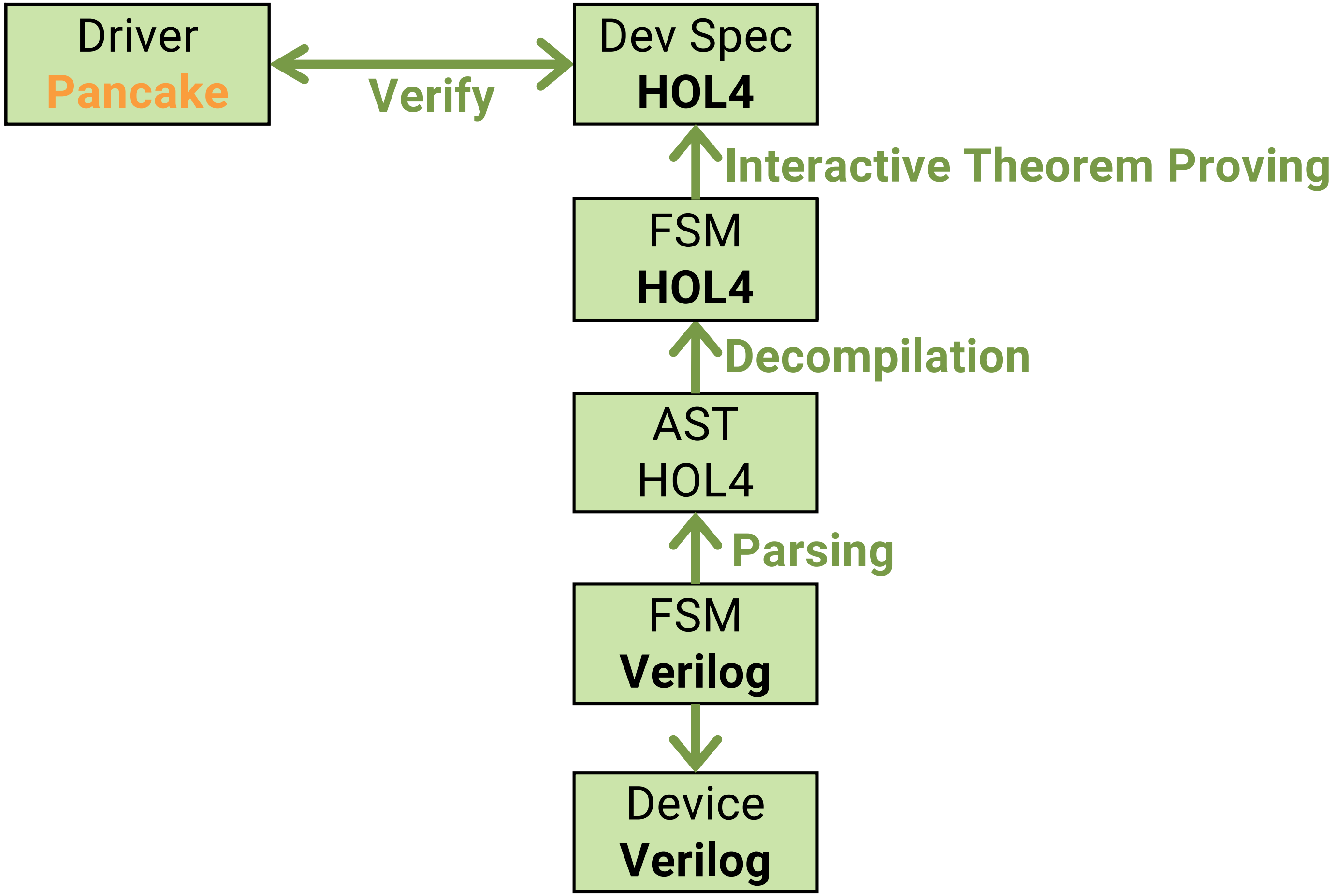
Status: I²C



Other devices:

- SPI: working on HOL-4 FSM
- Ethernet controller next

Aim: Simplified Process



Other On-Going Work

Other Work

Secure Microservices on seL4

- Joint work with UCR, funded by AFRL
- Shared R/O, partitioned R/W file system
- Based in template PDs, DAC

Djawula – provably secure, general-purpose OS

- Several new PhD students
- About to gather steam

Summary

- Performance isn't a weakness, it's a strength!
- Device driver availability is no longer a problem (for most embedded use)
- System services are maturing
- Verification of user-level components is happening (see Junming's talk)
- We're about to solve the problem of driver bugs for good
- Hard real-time is back!

https://trustworthy.systems