School of Computer Science & Engineering

Trustworthy Systems Group

Why Change the Kernel When You Have seL4?

Gernot Heiser

gernot@unsw.edu.au

@microkerneldude.bsky.social

https://gernot-heiser.org/

3rd Workshop on Kernel Isolation, Safety and Verification (KISV 2025)

October 13, 2025
Seoul, Republic of Korea

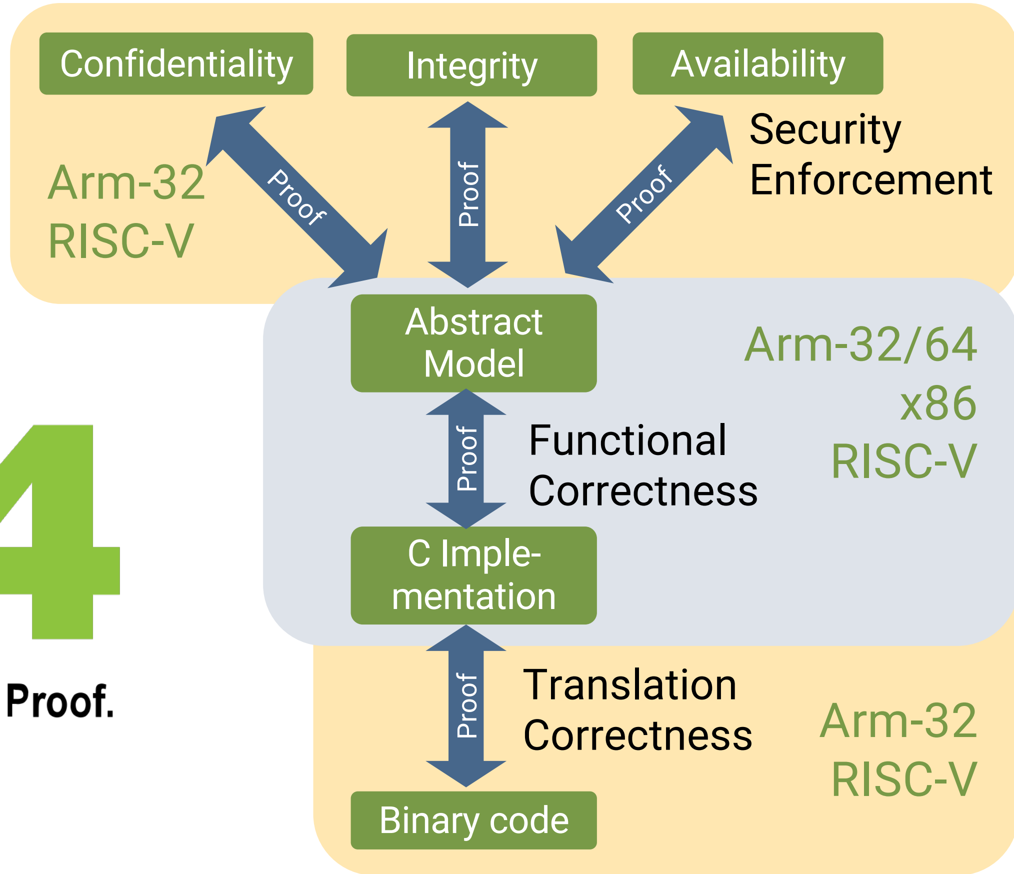
In conjunction with the
31st ACM Symposium on Operating Systems Principles (SOSP '25)

This workshop aims to bring together researchers and developers from the field of operating systems, programming languages, security, computer architecture and verification with the goal to accelerate changes in the kernel through a combination of isolation, programming language safety, and formal verification.

Which kernel?

Why?

Are We Talking About This Kernel?



seL4

Security. Performance. Proof.

Or This One?

Linux : Security Vulnerabilities, CVEs CVSS score between 9 and 10

Published in: 2025 January February March April May June July August September October

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9 In CISA KEV Catalog

Sort Results By : Publish Date Update Date CVE Number CVE Number CVSS Score EPSS Score

Page: 1 Copy

CVE-2024-47685

In the Linux kernel, the following vulnerability has been resolved: netfilter: nf_reject_ipv6: fix nf_reject_ip6_tcphdr_put() syzbot reported that nf_reject_ip6_tcphdr_put() was possibly sending garbage on the four reserved tcp bits (th->res1) Use skb_put_zero() to clear the whole TCP header, as done in nf_reject_ip_tcphdr_put() BUG: KMSAN: uninit-value in

Source: Linux

Max CVSS 9.1 · EPSS Score 0.79% · Published 2024-10-21 · Updated 2024-11-08

CVE-2024-42256

In the Linux kernel, the following vulnerability has been resolved: cifs: Fix server re-repick on subrequest retry When a subrequest is marked for needing retry, netfs will call cifs_prepare_write() which will make cifs repick the server for the op before renegotiating credits; it then calls cifs_issue_write() which invokes smb2_async_writev() - which re-repicks the server. If a different server is then

Source: Linux

Max CVSS 9.8 · EPSS Score 0.08% · Published 2024-08-08 · Updated 2024-09-06

CVE-2024-39462

In the Linux kernel, the following vulnerability has been resolved: clk: bcm: dvp: Assign ->num before accessing ->hws Commit f316cdff8d67 ("clk: Annotate struct clk_hw_onecell_data with __counted_by") annotated the hws member of 'struct clk_hw_onecell_data' with __counted_by, which informs the bounds sanitizer about the number of elements in hws, so that it can warn when hws is accessed out of bounds.

Source: Linux

Max CVSS 9.8 · EPSS Score 0.09% · Published 2024-06-25 · Updated 2025-03-24

CVE-2024-38623

In the Linux kernel, the following vulnerability has been resolved: fs/ntfs3: Use variable length array instead of fixed size Should fix smatch warning: ntfs_set_label() error: __builtin_memcpy() 'uni->name' too small (20 vs 256)

Source: Linux

Max CVSS 9.8 · EPSS Score 0.20% · Published 2024-06-21 · Updated 2025-03-24

CVE-2022-48716

In the Linux kernel, the following vulnerability has been resolved: ASoC: codecs: wcd938x: fix incorrect used of portid Mixer controls have the channel id in mixer->reg, which is not same as port id. port id should be derived from chan_info array. So fix this. Without this, its possible that we could corrupt struct wcd938x_sdw_priv by accessing port_map array out of range with channel id instead of port id.

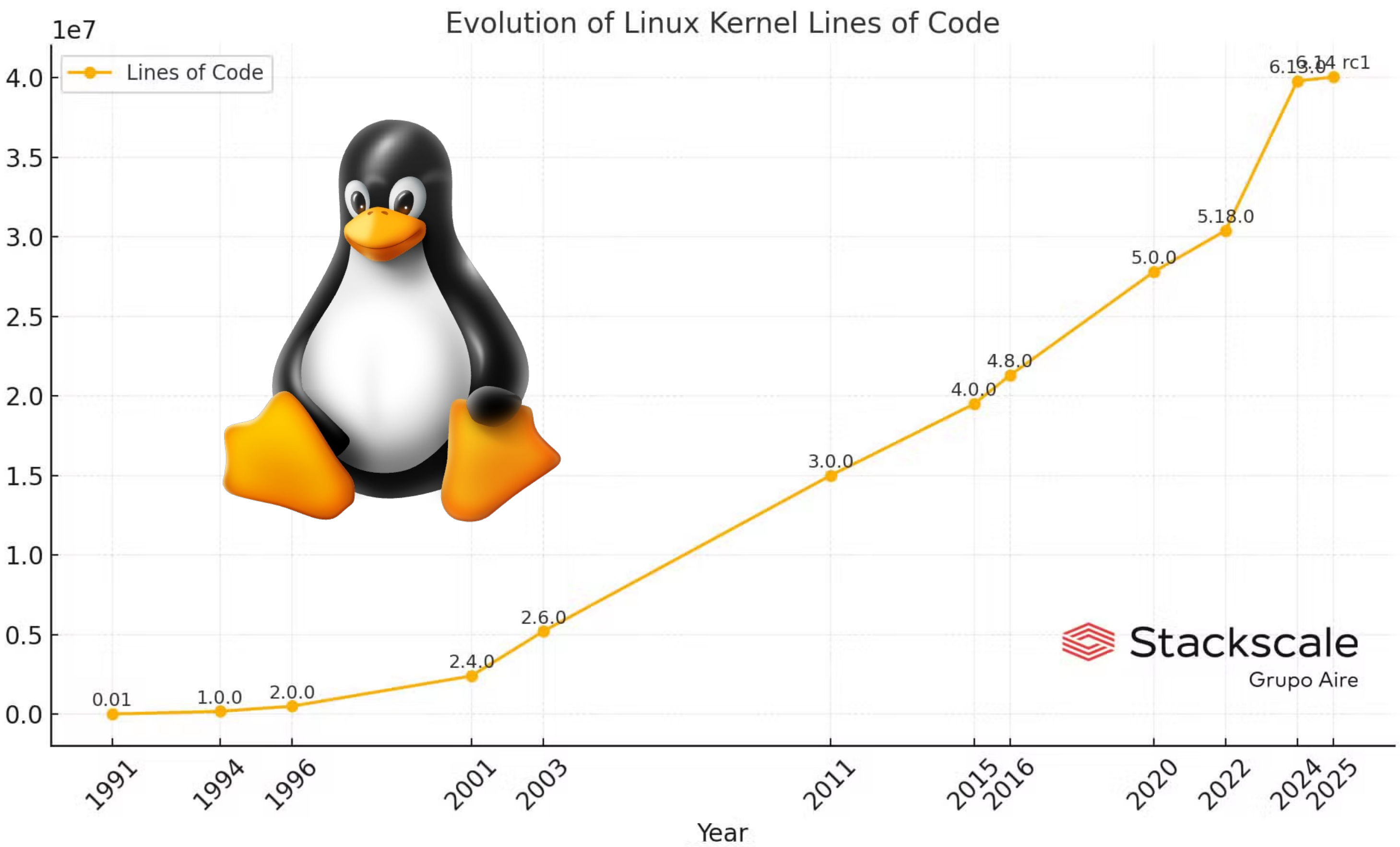
Source: Linux

Max CVSS 9.8 · EPSS Score 0.11% · Published 2024-06-20 · Updated 2025-04-01

CVE-2024-38612

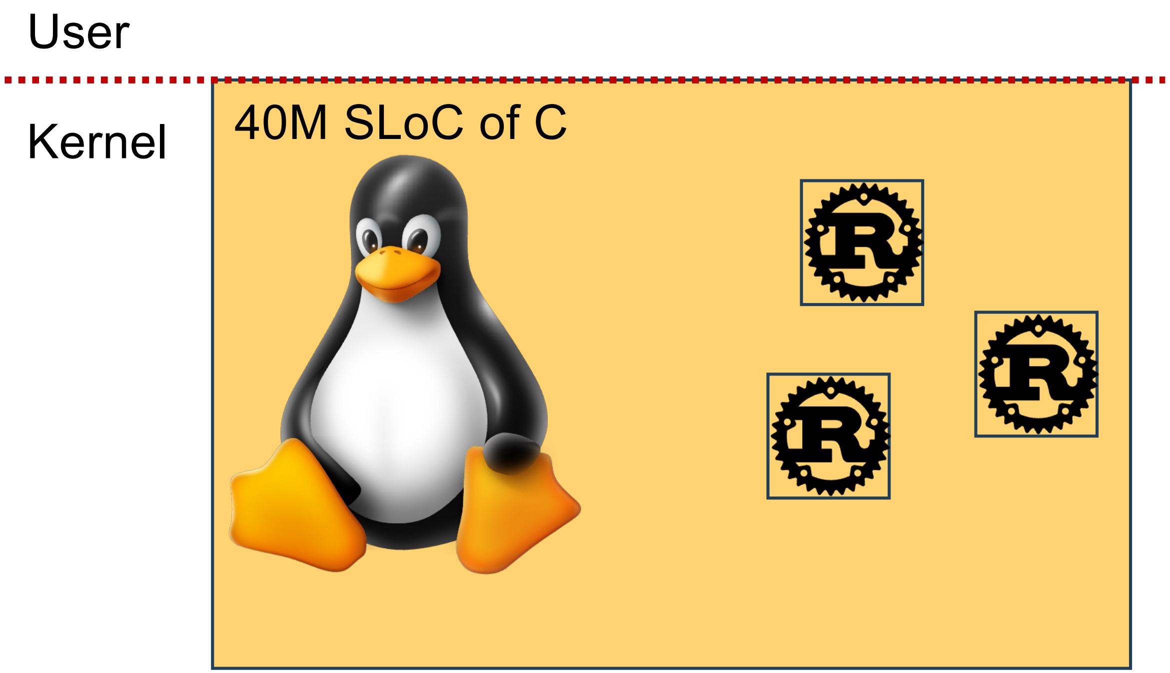
In the Linux kernel, the following vulnerability has been resolved: ipv6: sr: fix invalid unregister error path The error path of seg6_init() is wrong in case CONFIG_IPV6_SEG6_LWTUNNEL is not defined. In that case if seg6_hmac_init() fails,

Max CVSS 9.8 · EPSS Score 0.13% · Published 2024-06-19



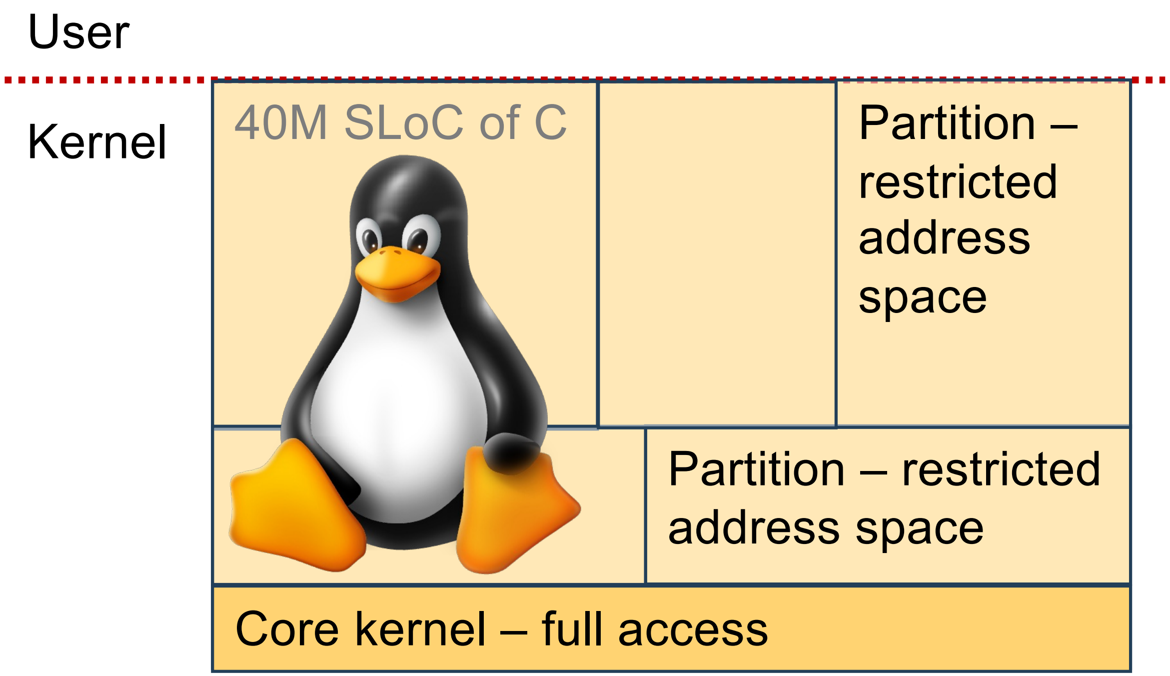
UNSW SYDNEY

Changing The Kernel: Modules in Rust



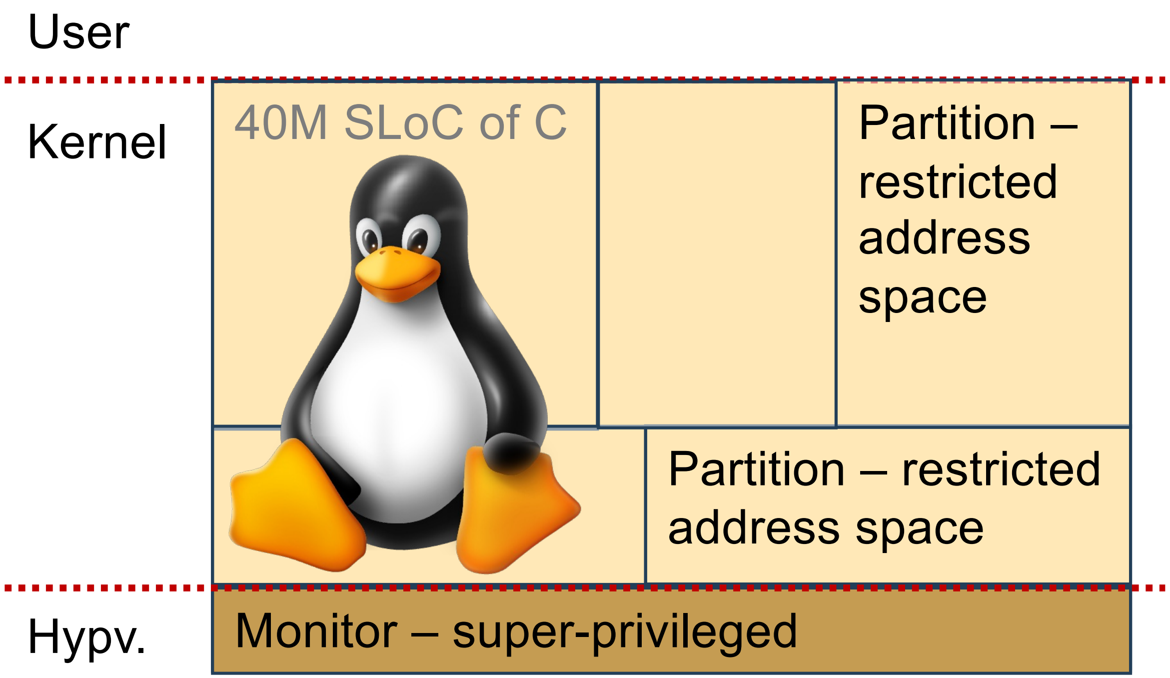
- ✓ Protects rest of kernel from Rust modules
- ❖ Unsafe code?
- ❖ Doesn't protect Rust module from rest of kernel
- ❖ Requires writing modules from scratch

Changing: Partition Kernel Space



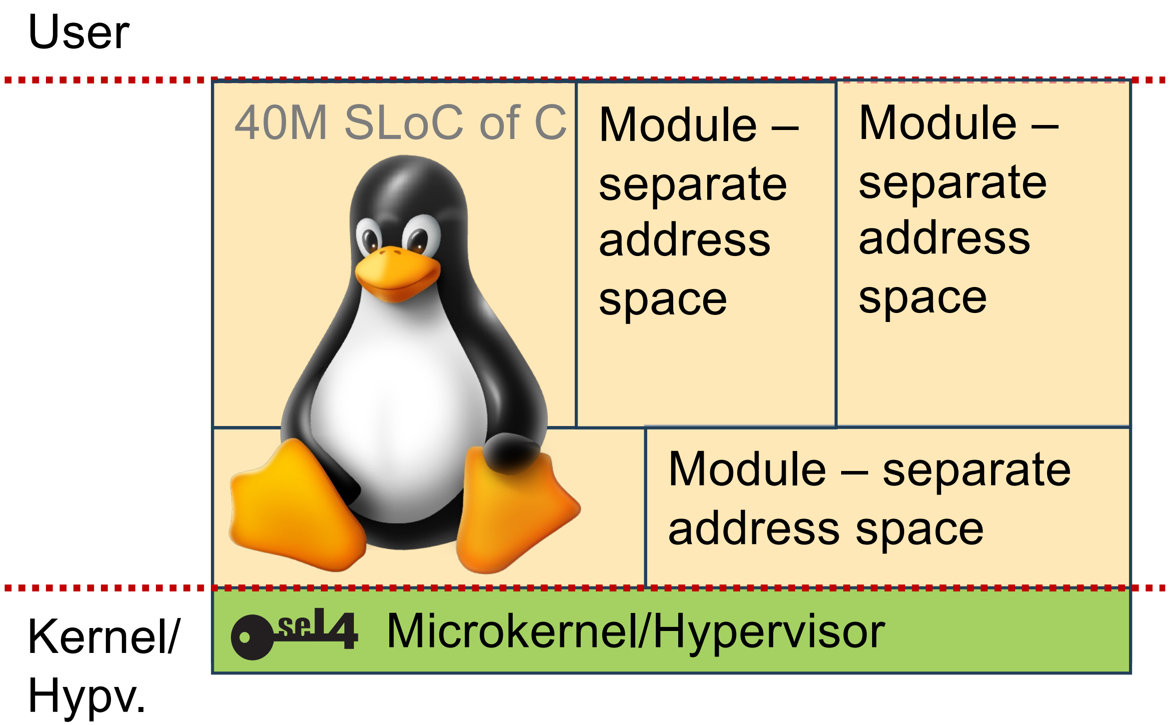
- ✓ Protects partitions from each other
- ❖ Requires HW extensions ⇒ not on older platforms
- ❖ Increases ISA/manufacturer dependence
- ❖ Still fully trust core kernel
- ❖ Privilege revocation?
- ❖ Cost?

Changing: Partition Kernel Space



- ✓ Protects partitions from each other
- ❖ Requires HW extensions ⇒ not on older platforms
- ❖ Increases ISA/manufacturer dependence
- ❖ Fully trust monitor (but it does very little)
- ❖ Privilege revocation?
- ❖ Cost?
- ❖ **Squats hypervisor mode – lose virtualisation support?**

Changing: De-Privileging "Kernel"



- ✓ Protects ***all*** modules from each other
- ✓ Requires no special HW
- ✓ Verified kernel
- ✓ Retain virtualisation support
- ❖ High cost?

Microkernel Overheads

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

Microkernel Overheads

Chen et al, OSDI'24

High syscall rate = 61k/s

seL4 round-trip address-space switch = 1k cy

Conservative IMHO

Assume average 2 R-T AS switches / syscall:

Switch O/H = 2 × 61k/k × 1kcy = 122M cy/s

Assume 3GHz clock:

O/H = 122M cy/s / 3Gcy/s = 122/3k = 4%

Assume 4-core CPU:

O/H = 4%/4 = 1% of CPU!

Assume Linux max CPU load = 25%

relative O/H = 4 × 1% = 4%

Why would anyone care?

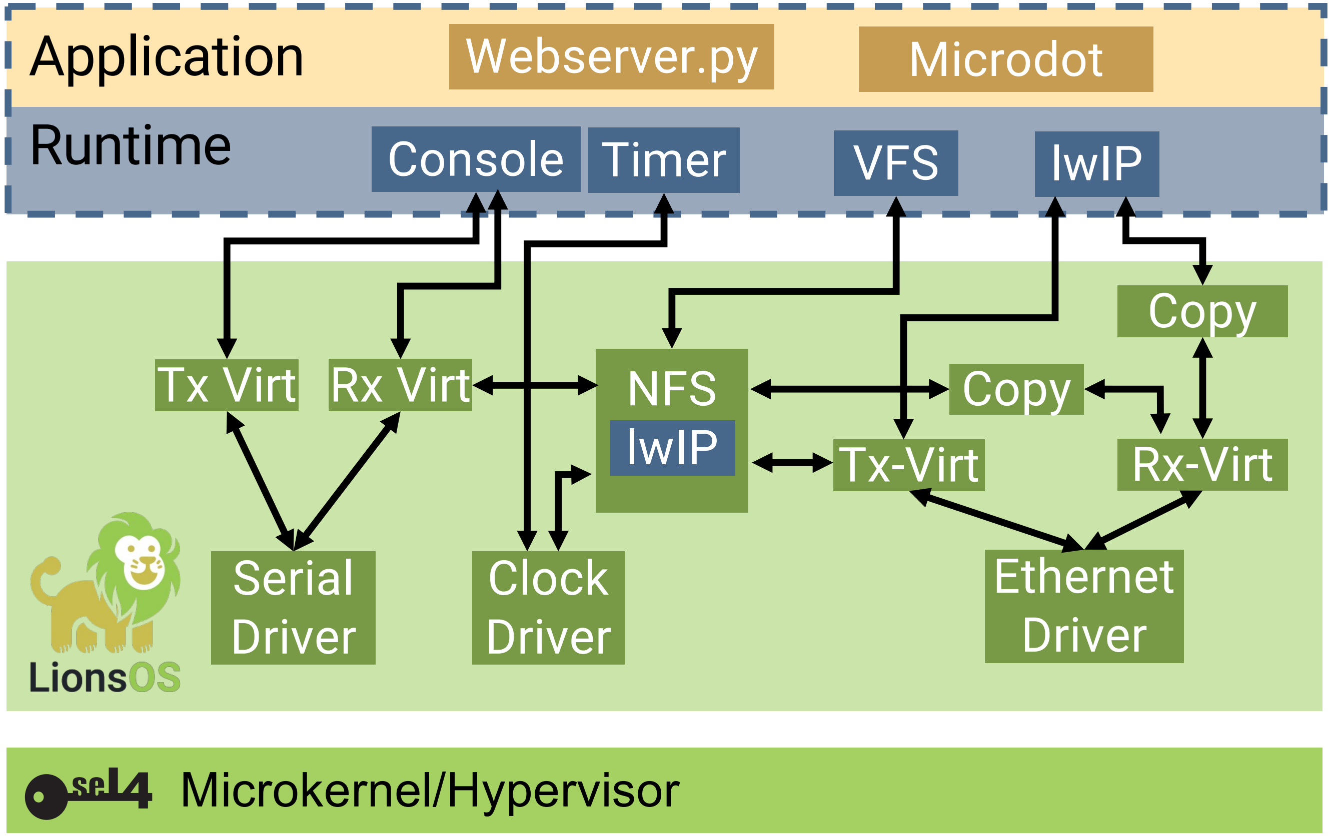
But Is This Real?

LionsOS

Test bed: LionsOS

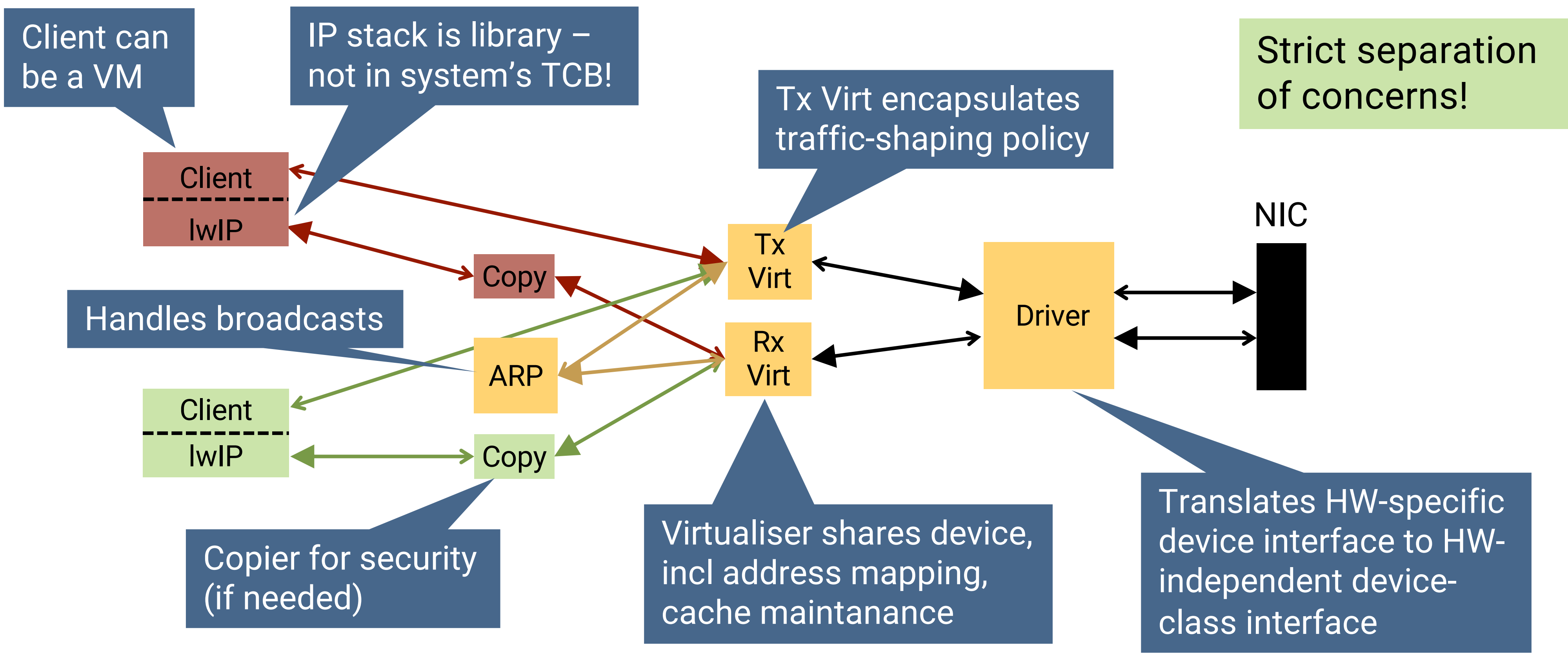
- Simple, from-scratch seL4-based OS
- Highly modular design,
  strict separation of concerns
- Adaptable "Lego® kit" approach
- Designed for embedded / cyber-physical systems

Underneath https://sel4.systems/

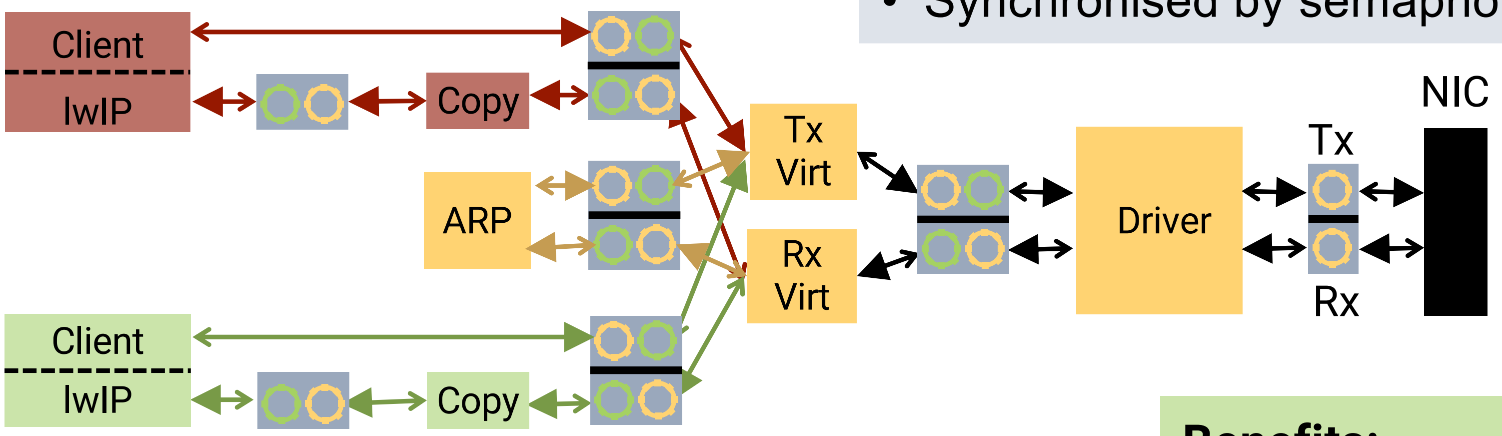


Web-server OS:
- 10 modules
- 3 libraries

Networking Layer



Networking Layer

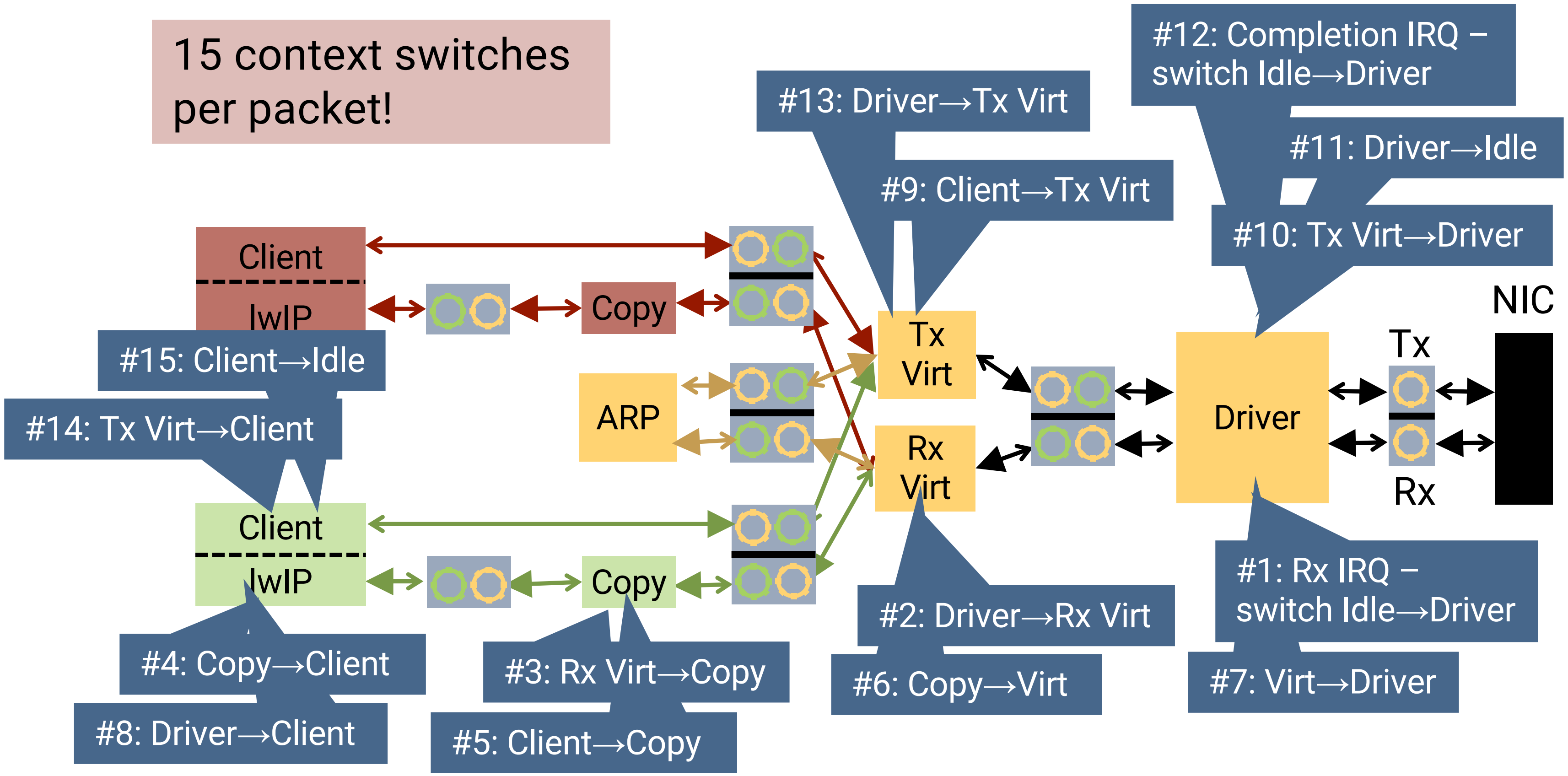


Zero-copy communication:
- Lock-free, single-producer, single-consumer, bounded queues
- Synchronised by semaphores

Benefits:
- simple components
- **location transparency**

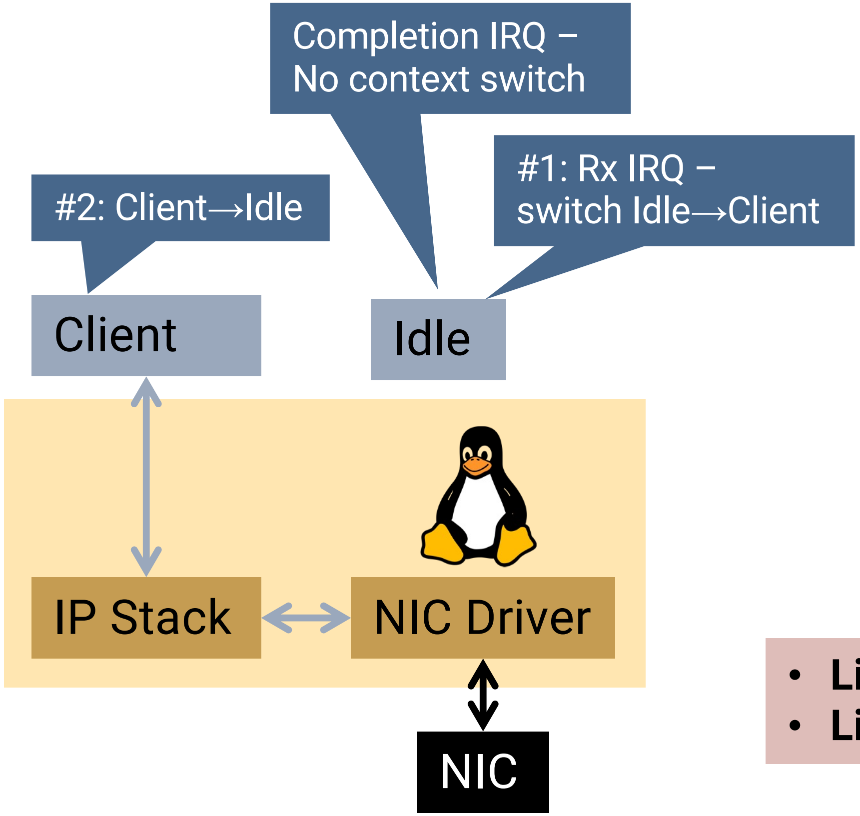
Packet Round-Trip Context Switches

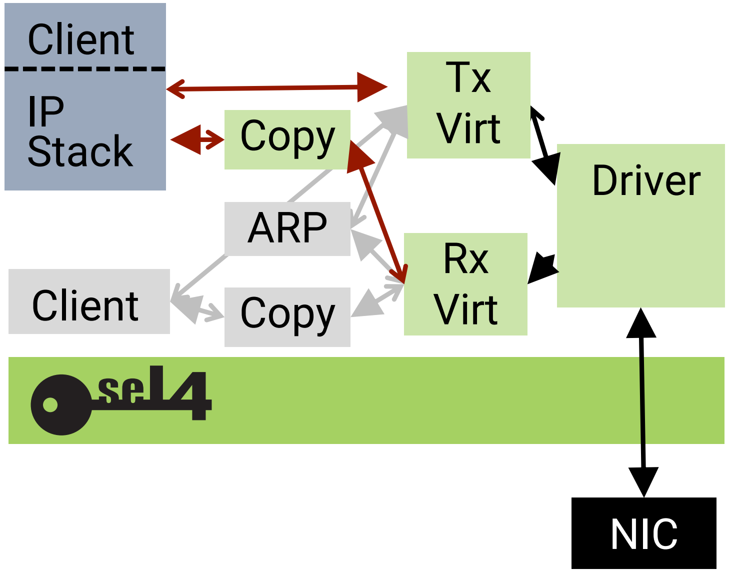
15 context switches per packet!



Must return free buffers!

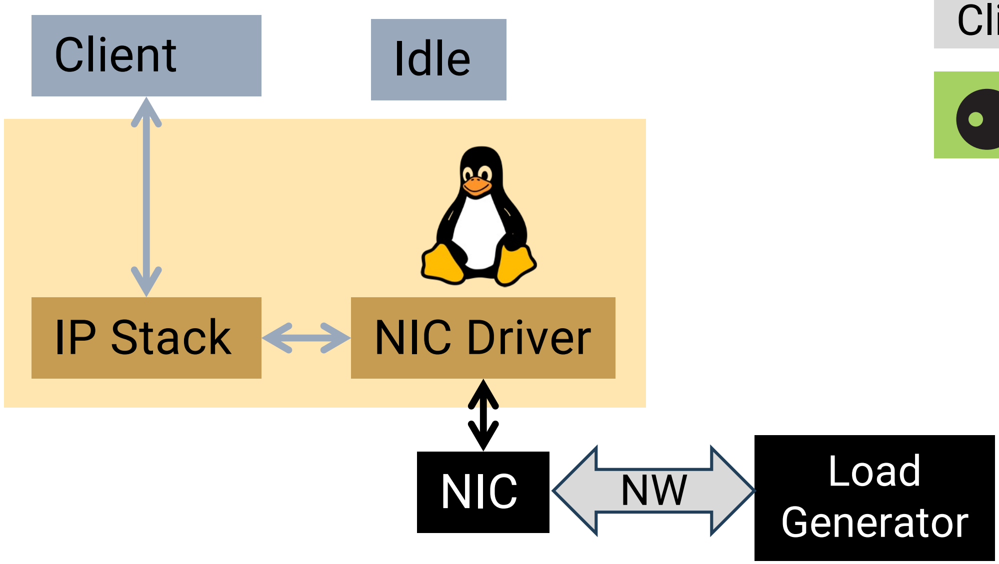
Comparing to Linux

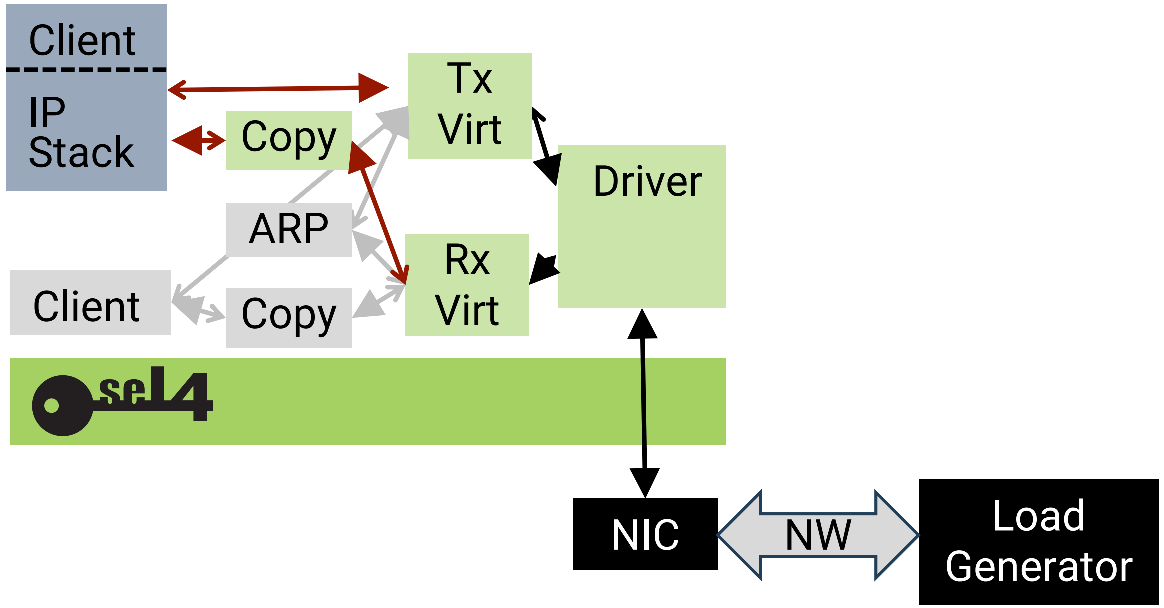




- **LionsOS:** 30 mode switches, 15 context switches
- **Linux:** 6 mode switches, 2 context switches

Comparing Performance: Setup





- External load generator
- Measures throughput, latency
- Client echoes packets

What Do We Expect?

Ethernet packet size = 1.5kB

Assume Linux mode switch = half context switch

LionsOS O/H = 12/pk × 0.5k cy = 6k cy/pkt

Max packet rate for 1Gb/s NIC:

rate = 1Gb/s / 1.5kB = 1Gb/s / 12kb = 833k/s

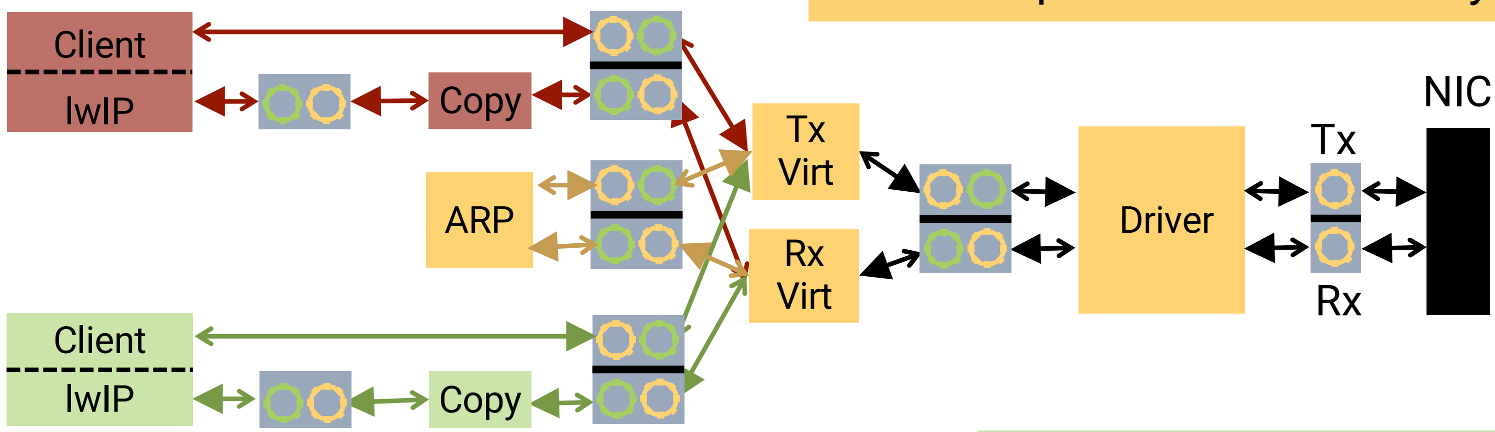
Worst-case O/H for 1Gb/s NIC:

O/H = 6k cy/pkt * 833k pkt/s = 0.5G cy/s

Assume 3GHz clock:

rel O/H = 0.5G cy/s / 3G cy/s = 17% of core

However, There's Batching



- Each component will process everything in its queue before signalling another component
- No component will ever busy-poll!

- Dramatically reduces context switches under load!
- Measure 5–10 pkt/IRQ!

What Do We Expect?

Ethernet packet size = 1.5kB

Assume Linux mode switch = half context switch

LionsOS O/H = 12/pk × 0.5k cy = 6k cy/pkt

Max packet rate for 1Gb/s NIC:

rate = 1Gb/s / 1.5kB = 1Gb/s / 12kb = 833k/s

Worst-case O/H for 1Gb/s NIC:

O/H = 6k cy/pkt * 833k pkt/s = 0.

Assume 3GHz clock:

rel O/H = 0.5G cy/s / 3G cy/s = 17% of core

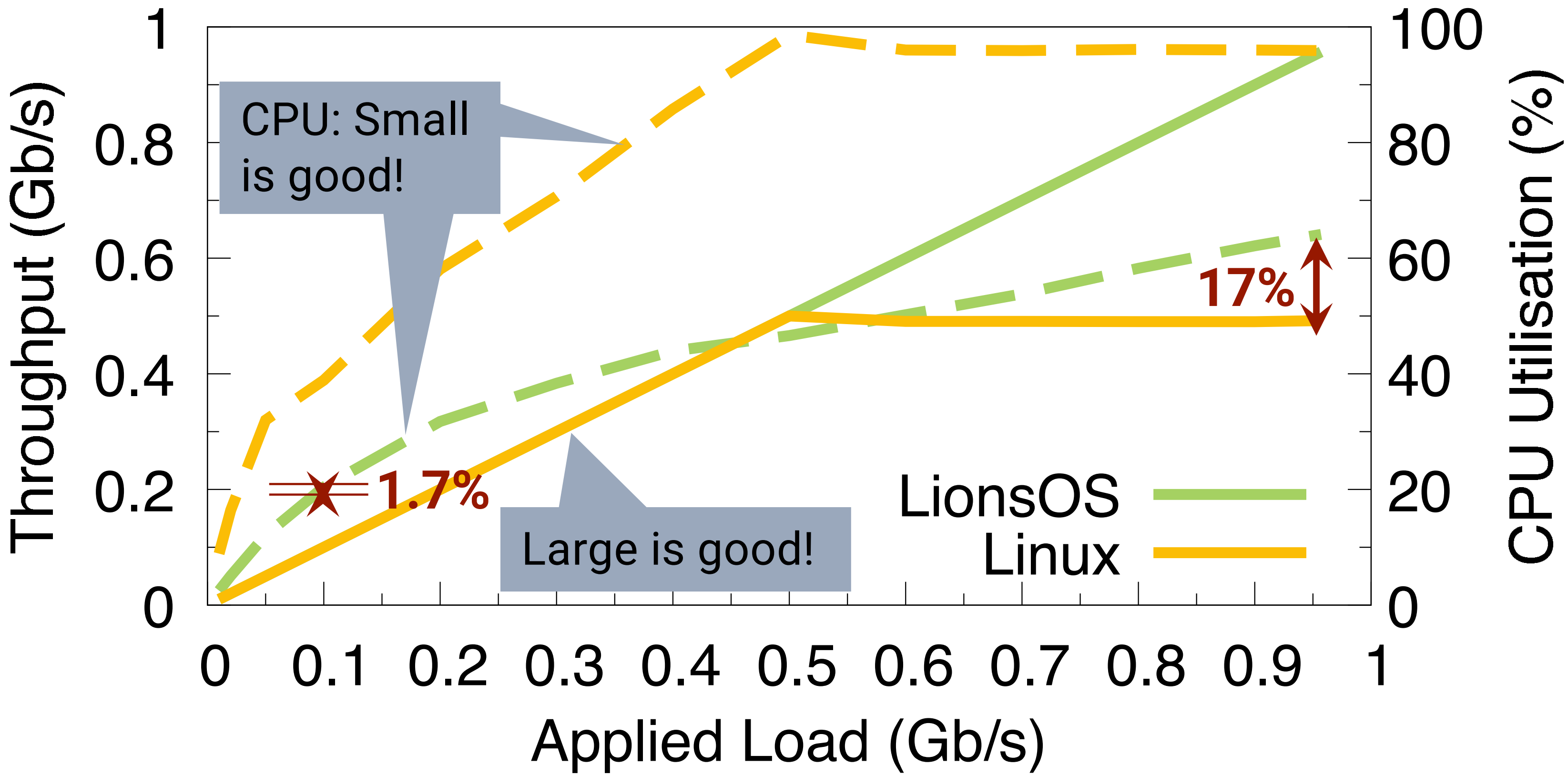
At 100Mb/s, packet spacing = 1/(83k/s) = 12µs

rel O/H = 17%/10 = 1.7% of core

Highly pessimistic due to natural batching!

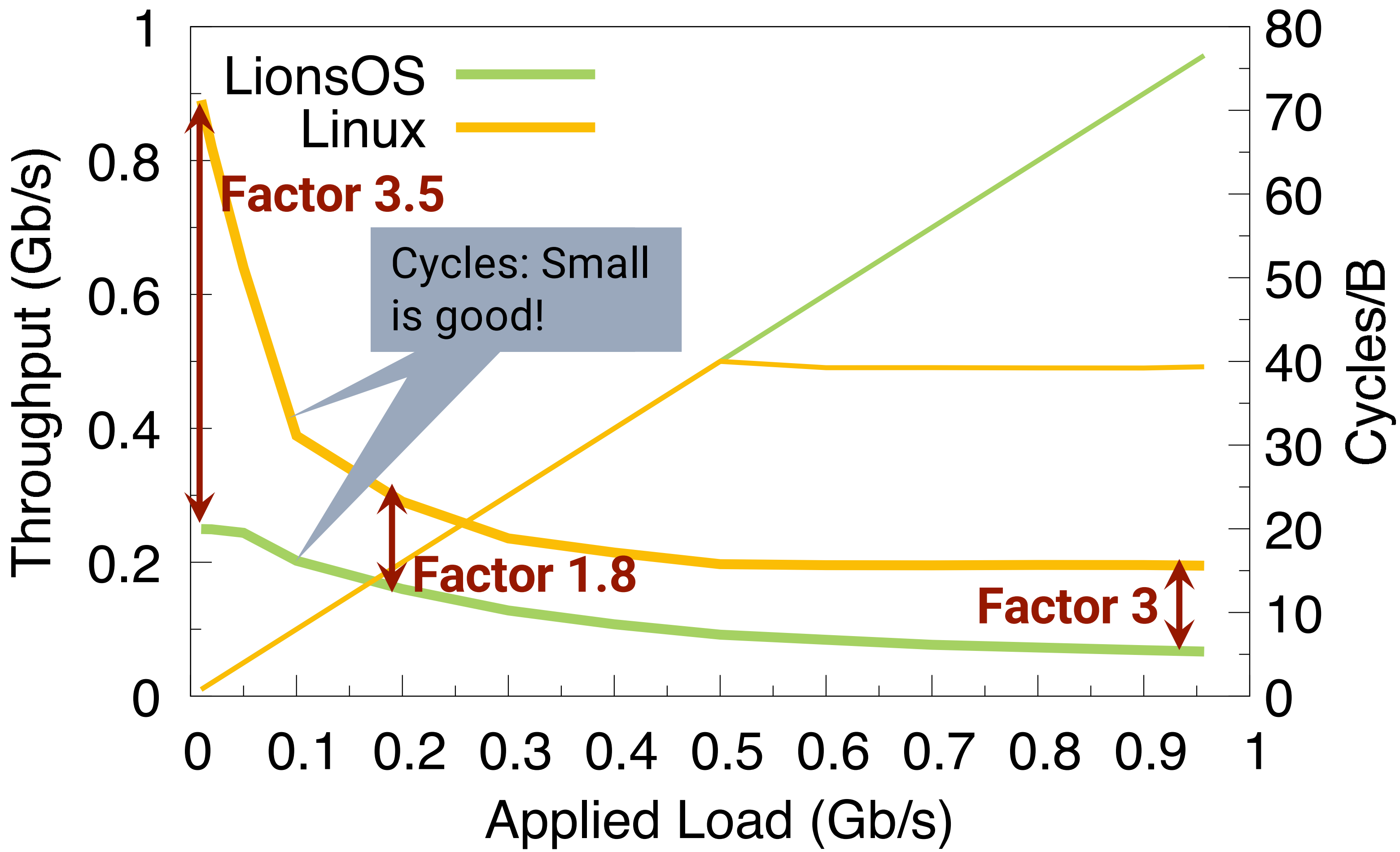
Avoid batching by spacing packets!

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

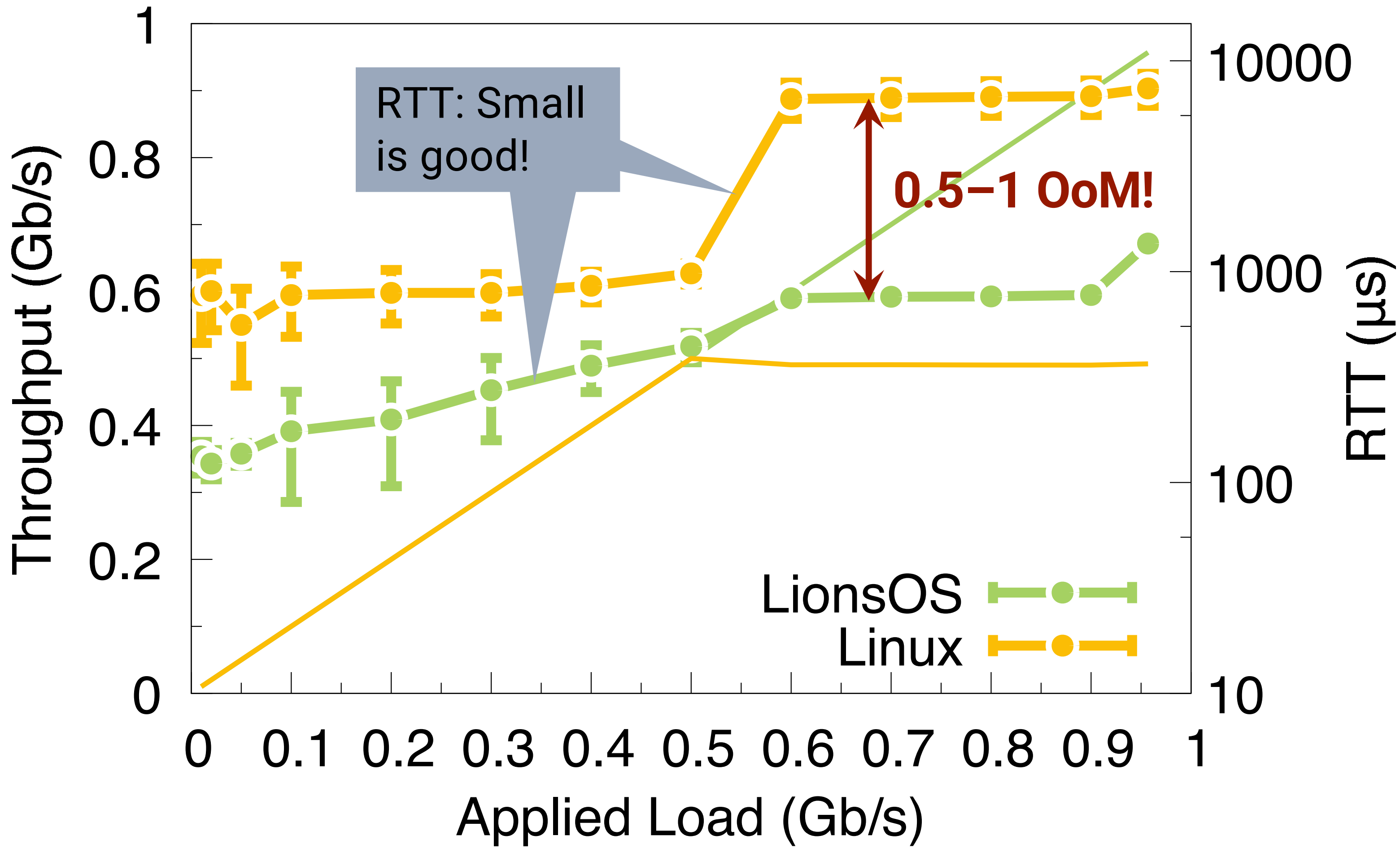


Single-core configuration

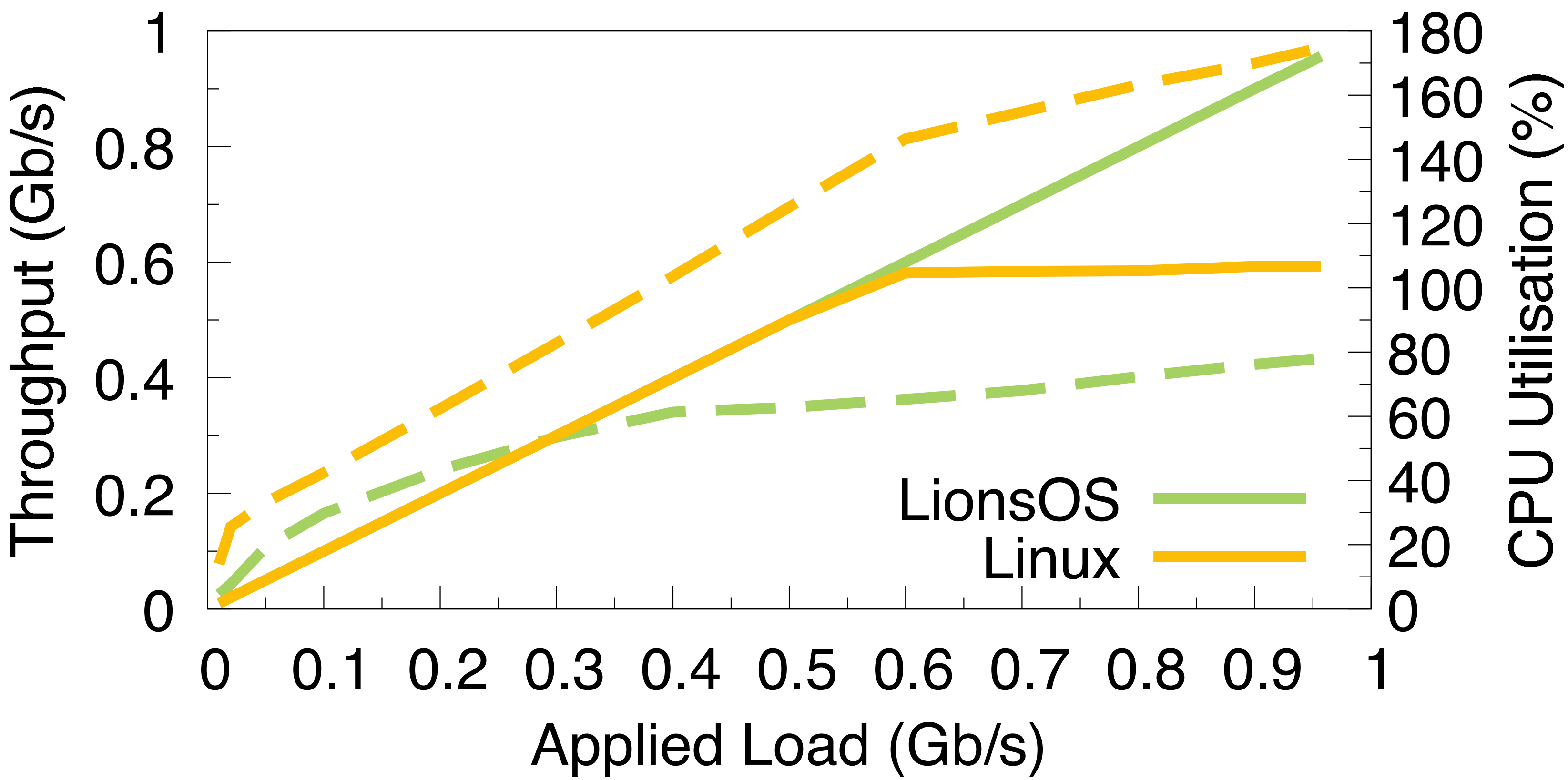
Performance: Processing Cost per Byte



Performance: Round-Trip Times

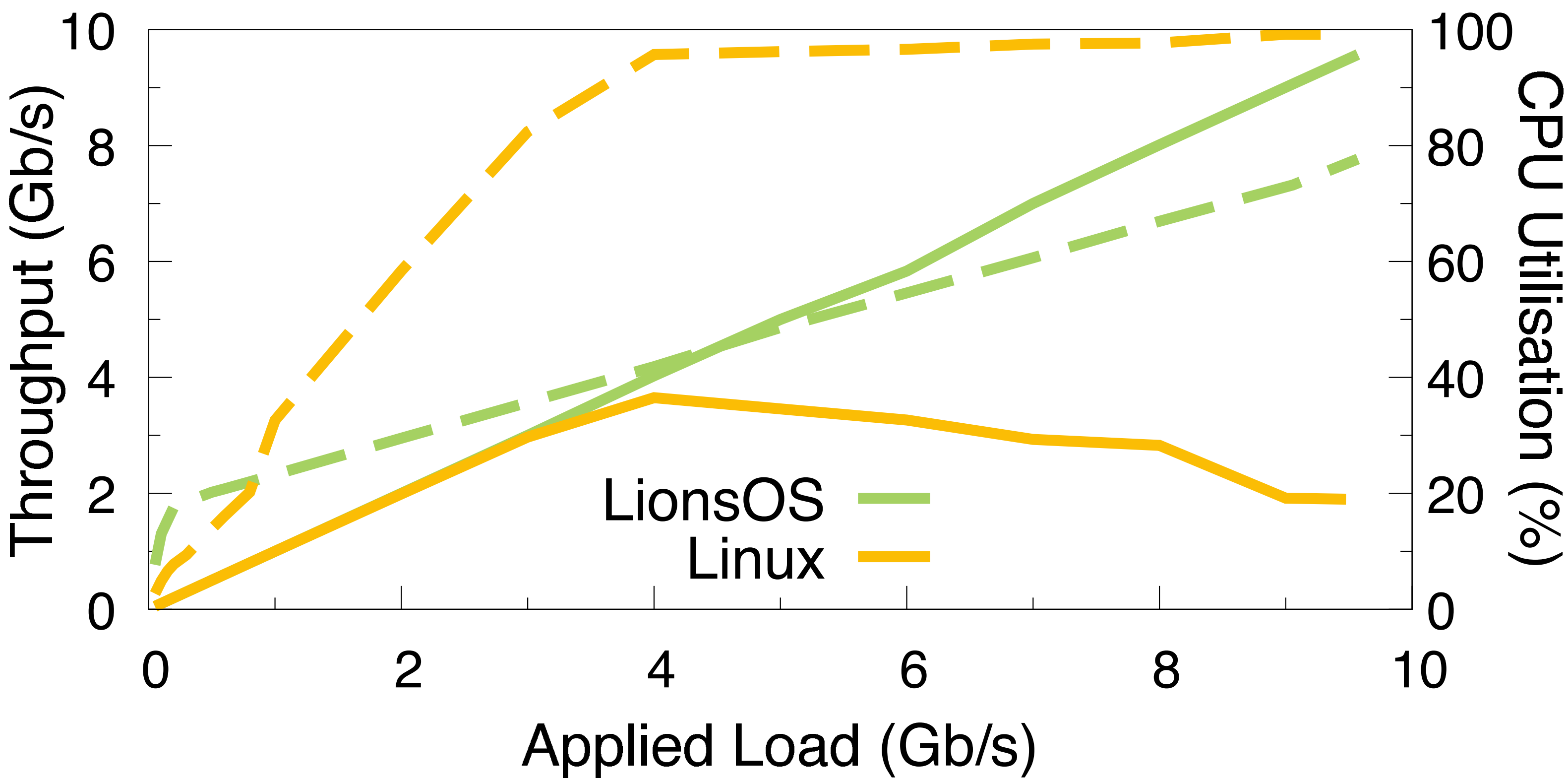


Performance: i.MX8M, 1Gb/s Eth, UDP



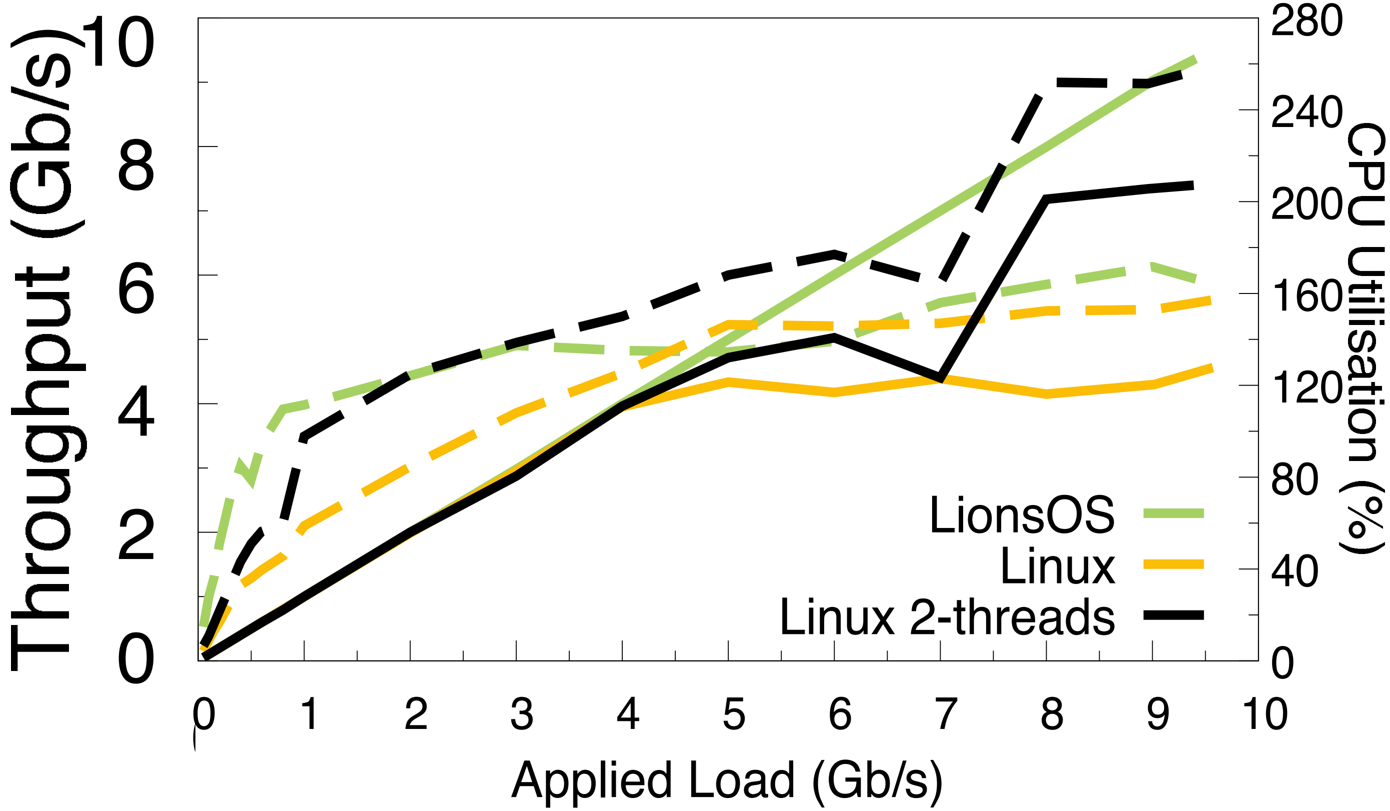
Multicore configuration

Performance: x86, 10Gb/s Eth, UDP



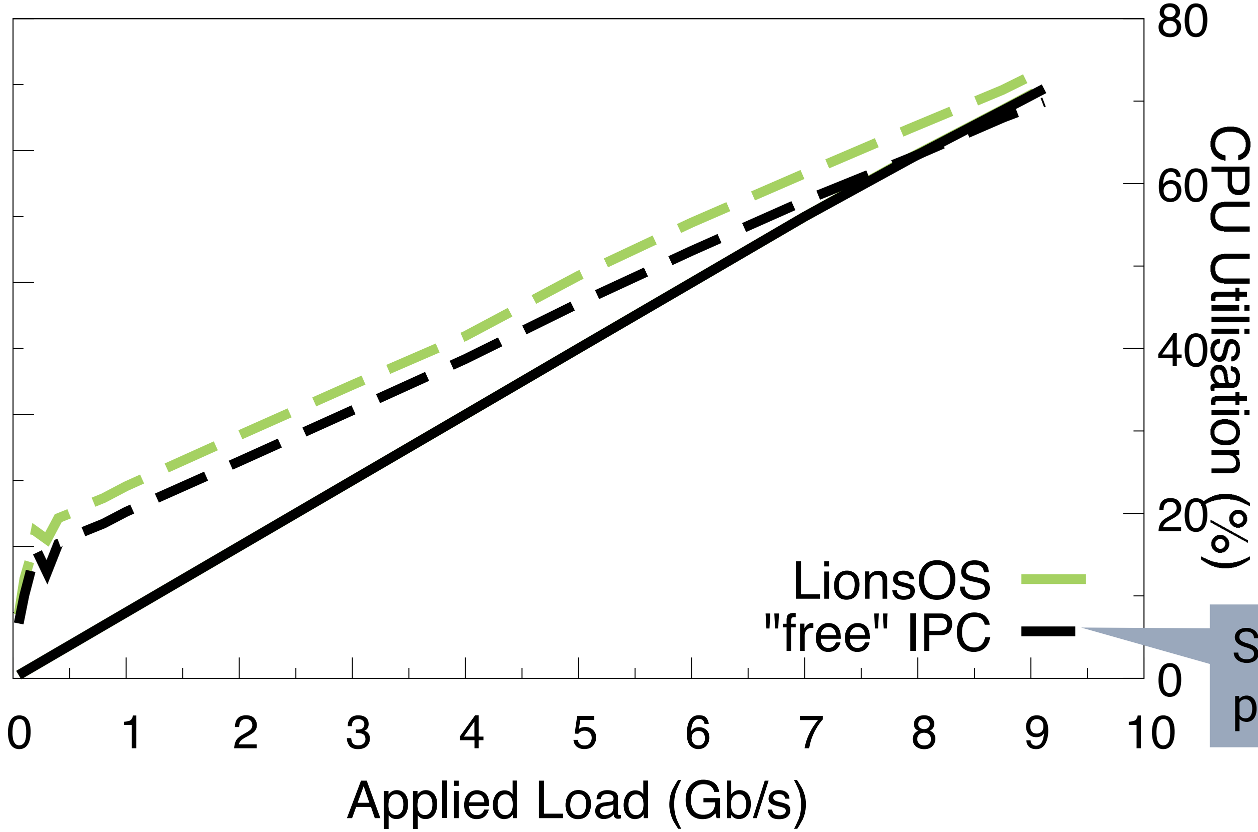
Single-core configuration

Performance: x86, 10Gb/s Eth, UDP



Multicore configuration

Syscall cost simulations (x86)



Subtract 500cy per kernel entry

Single-core configuration

Why Is LionsOS Faster Than Linux?

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

Microkernel overheads are in the noise!

LionsOS executes less code!

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

Why Is LionsOS So Simple?

Helps development **and** correctness!

Radical simplicity:
- Fine-grained modularity, strict separation of concerns
- Event-driven programming model, strictly sequential modules
- Static architecture
- Use-case-specific policies

Concurrency by distributing modules across cores

Matches embedded space – little dynamic resource management

Use-case diversity by replacing components

But I Want A **Real** OS!

Cost Of A Dynamic OS

- More complexity, larger code size

Might affect cache footprint?

- Double book-keeping, multiple server invocations

IPC overheads in the noise

- Higher startup times due to dynamic resource allocation

fork() will be the test!

- Resource revocation may require indirection

seL4 caps can be revoked without

- "Universal" policies are complex & costly

Do we need them?

UNSW SYDNEY

Do We Need "Universal" Policies?

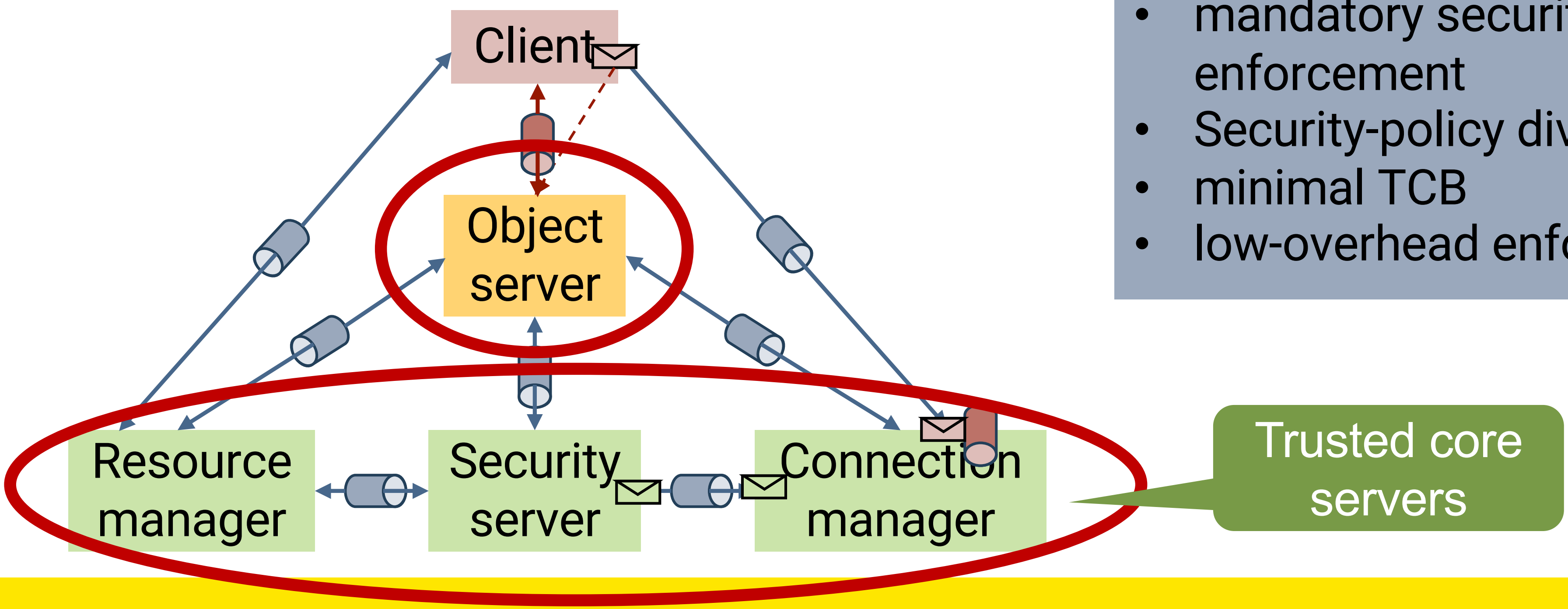
Claim:
- Systems rarely change policies on-the-fly
- Can change policy by replacing policy module

Keep configuration complexity off-line!

LionsOS experiment:
- Reload component with new policy implementation
- Cost: **17µs** on i.MX8M

Djawula: PoC Of General-Purpose OS

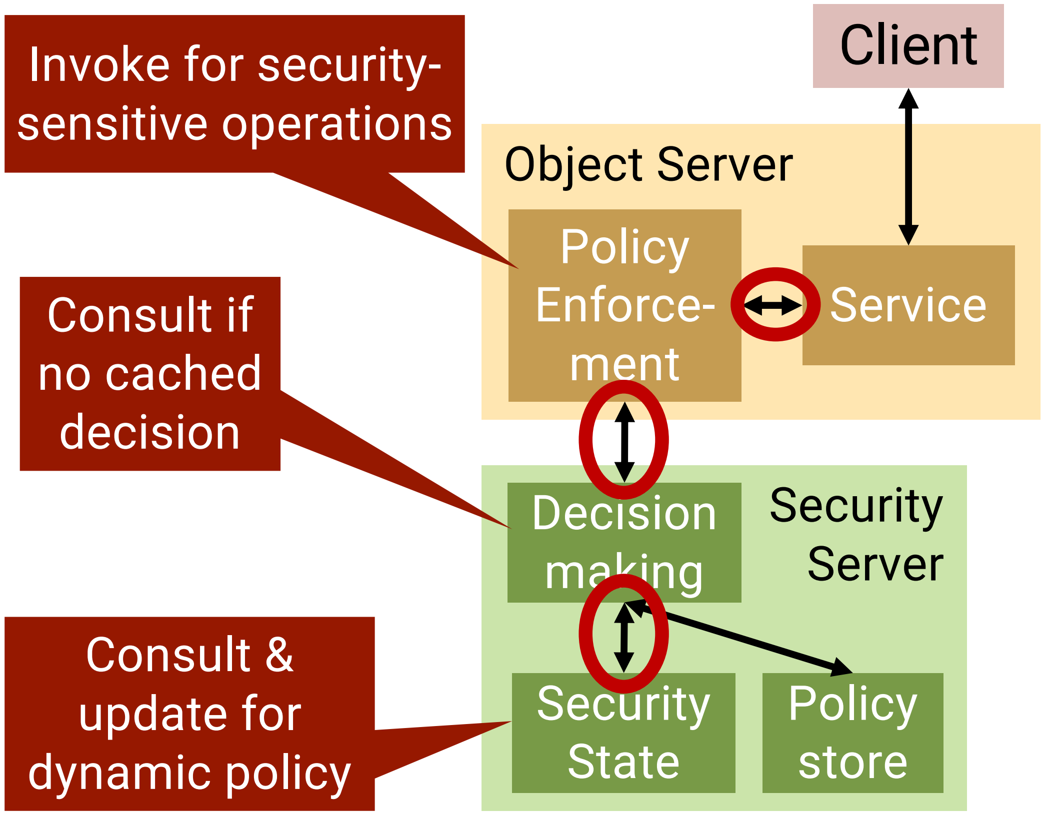
Aim: General-purpose OS that **provably** enforces a general security policy



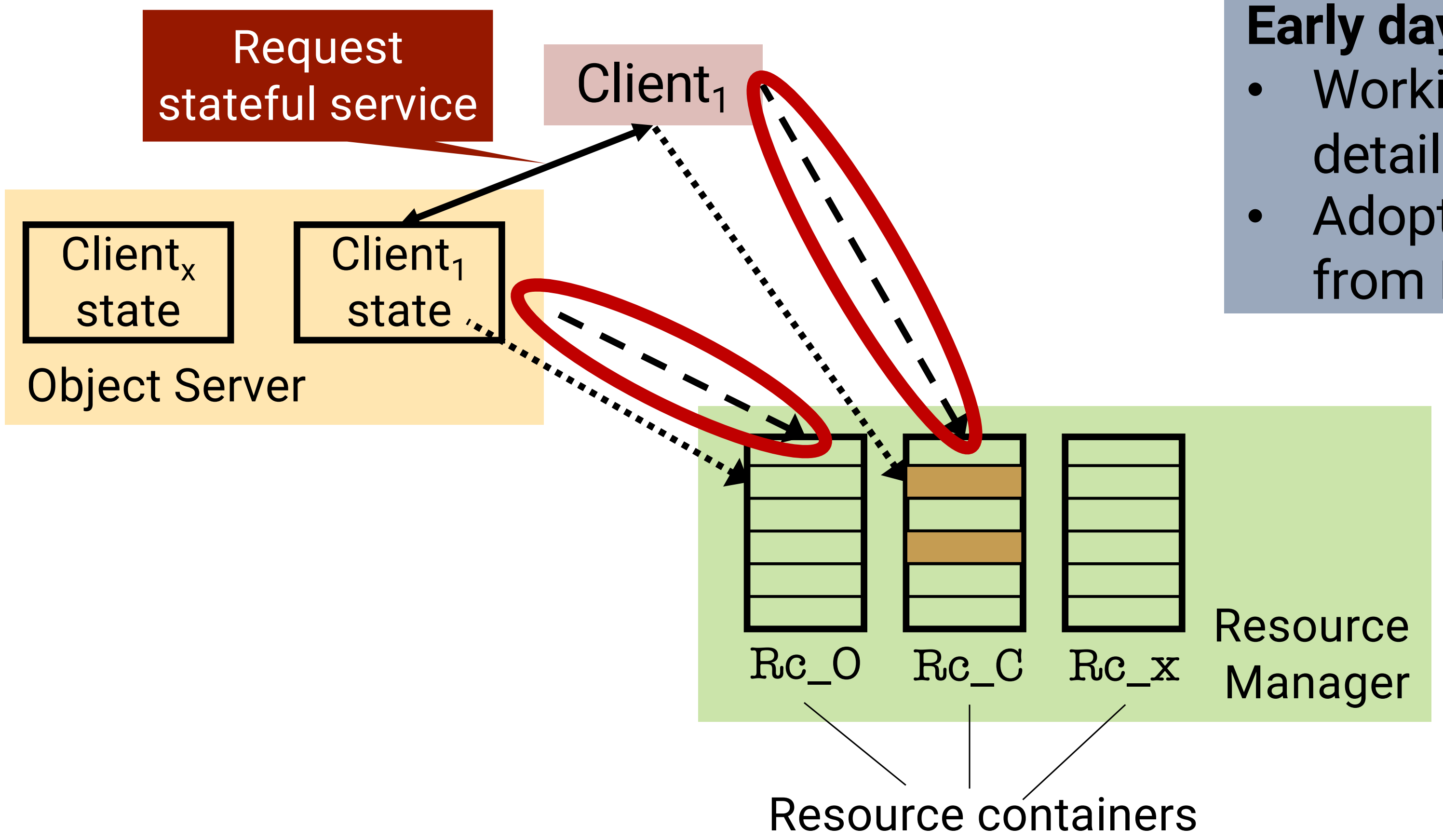
Requires:

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

Core Ideas: Dynamic Enforcement



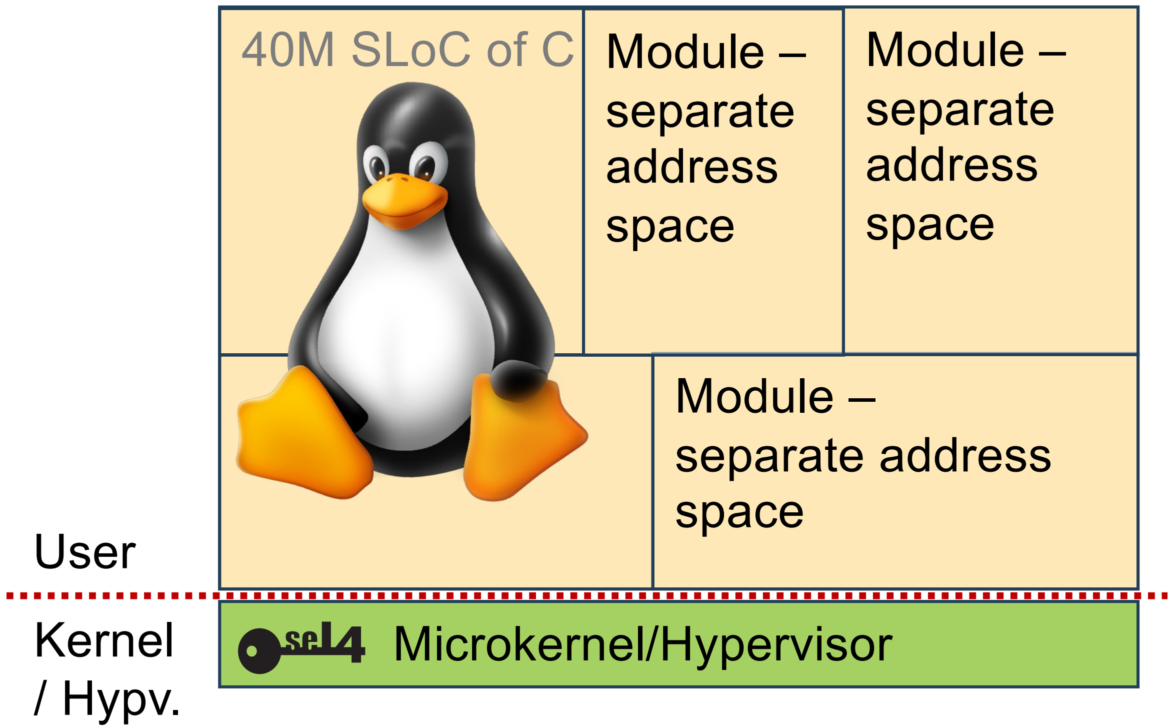
Core Ideas: Resource Donation



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

My View Of “Changing The Kernel”

"Changing The Kernel"



- ✓ Protects ***all*** modules from each other
- ✓ Requires no special HW
- ✓ Verified kernel
- ✓ Retain virtualisation support
- ✓ **Cost is in the noise with the right design!**

https://trustworthy.systems

We're hiring!
Operating-systems faculty