

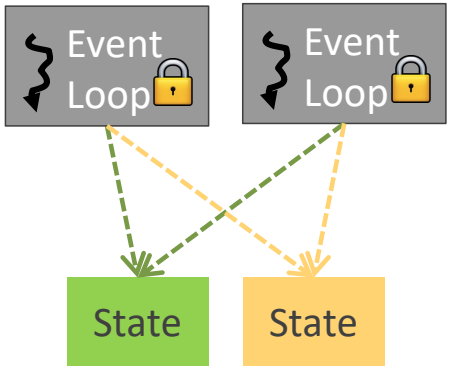
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2025 T3 Week 02 Part 2

Async I/O & LionsOS

Peter Chubb & Gernot Heiser



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

- Asynchronous I/O in Linux (Peter Chubb)
- seL4 usability
- LionsOS design principles
- Synchronous vs asynchronous I/O
- Revisit threads vs events
- LionsOS performance

© Gernot Heiser 2025 – CC BY 4.0

Linux's Asynchronous APIs

Peter Chubb

Senior Consultant, Trustworthy Systems

peter.chubb@unsw.edu.au

© Gernot Heiser 2025 – CC BY 4.0

- Only useful on I/O
  - Pipes
  - FIFOS
  - Sockets
  - Devices

© Peter Chubb 2026, CC-BY-SA 4.0

How do you know when to retry?

1: SIGIO

```
struct sigaction handle_IO = {
  .sa_handler = read_my_file;
  .sa_mask = 0;
  .sa_flags = 0;
};
sigaction(SIGIO, &handle_IO, NULL);
fcntl(fd, F_SETOWN, getpid());
```

calls `handle_my_file()` when IO is possible **2**: **`poll()`**

```
struct pollfd events = {
  .fd = fd;
  .events = POLLIN;
};
```


© Peter Chubb 2026, CC-BY-SA 4.0

Issues with `select()` and `poll()`

— Doesn't scale — O(n) linear search through `events` array
— Level triggered — Get complete list of ready fds every time

© Peter Chubb 2026, CC-BY-SA 4.0

`epoll()`

- *extended* `poll()`
- edge or level triggered
- scales well

© Peter Chubb 2026, CC-BY-SA 4.0

epoll()

```
struct epoll_event ev;
pollfd = epoll_create1(EPOLL_CLOEXEC);
ev.fd = fd;
ev.events = EPOLLIN|EPOLLET;
epoll_ctl(pollfd, EPOLL_CTL_ADD, fd, &ev);
for (;;) {
  ret = epoll_wait(pollfd, 1, &ev, TIMEOUT);
  if (ret == 1) {
    read(ev.fd, buf, sizeof buf);
  } else {
    /* Timeout or error */
  }
}
```


© Peter Chubb 2026, CC-BY-SA 4.0

epoll()

- `epoll_event`s returned only for ready fds
- Can associate arbitrary data with an fd (`.data` field)

© Peter Chubb 2026, CC-BY-SA 4.0

`libaio` — asynchronous IO for seekable fds

Do `man 3 io` for details and an example

- Allows queueing of IO requests for files
- Might reduce disc idle time
- man pages out-of-date
- Can take longer than a synchronous approach

© Peter Chubb 2026, CC-BY-SA 4.0

Benchmarks

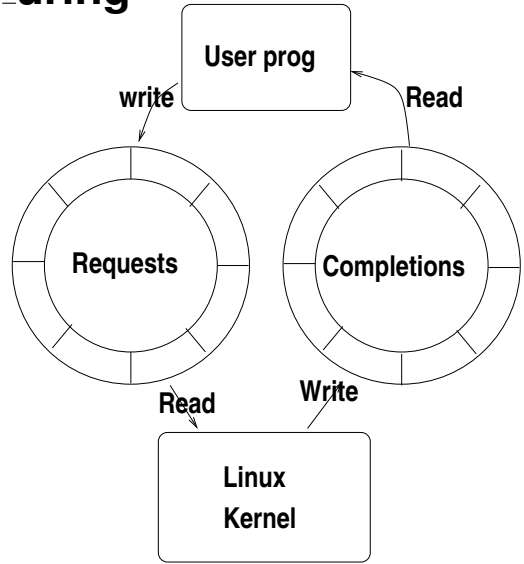
Time to copy a 1GB file (sec)

What	**Async**		`/usr/bin/cp`	
	Time	nsyscall	time	nsyscall
SSD, X86, cache-cold	0.89(8)s	16564	0,19(2)s	8167
SSD, X86, cache-hot	0.53(5)s	16564	0.17(2)s	8167
eMMC, aarch64	15(5)s	13560	24(5)s	8105

(too little RAM on ARM board for page cache to matter)

© Peter Chubb 2026, CC-BY-SA 4.0

io_uring



- a Linux-specific asynch IO mechanism
- Used to implement POSIX asynch IO
- Uses ring buffers: fewer system calls

© Peter Chubb 2026, CC-BY-SA 4.0

1. set up rings with `io_uring_setup()` and `mmap()`
2. Add requests to queue
3. invoke `io_uring_enter()`
4. poll for completion events

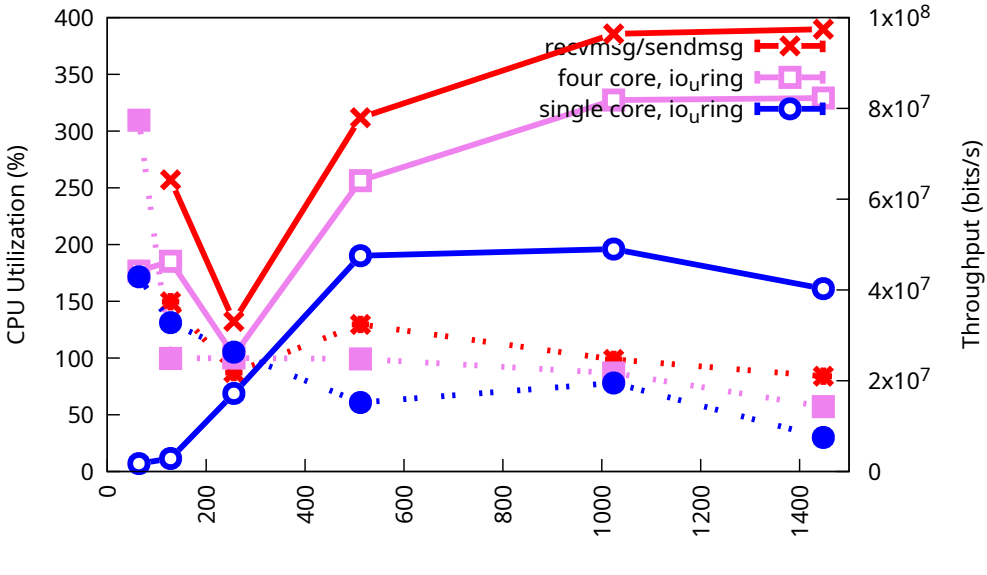
© Peter Chubb 2026, CC-BY-SA 4.0

`io_uring` gotchas

- Need multiple in-flight requests to gain much
- Significantly more complexity – 54 lines vs 331 lines
- Link against `liburing` to get interfaces
- Man page may be out of date
- portability issues between Linux versions

© Peter Chubb 2026, CC-BY-SA 4.0

100Mb/s load



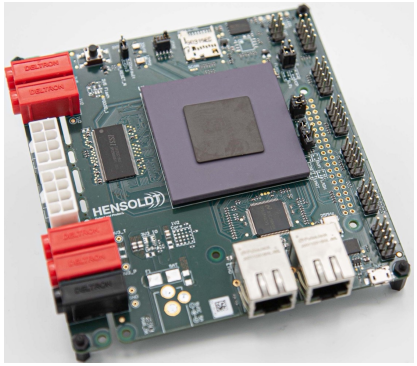
© Peter Chubb 2026, CC-BY-SA 4.0

seL4 Usability

© Gernot Heiser 2025 – CC BY 4.0

seL4 in Products

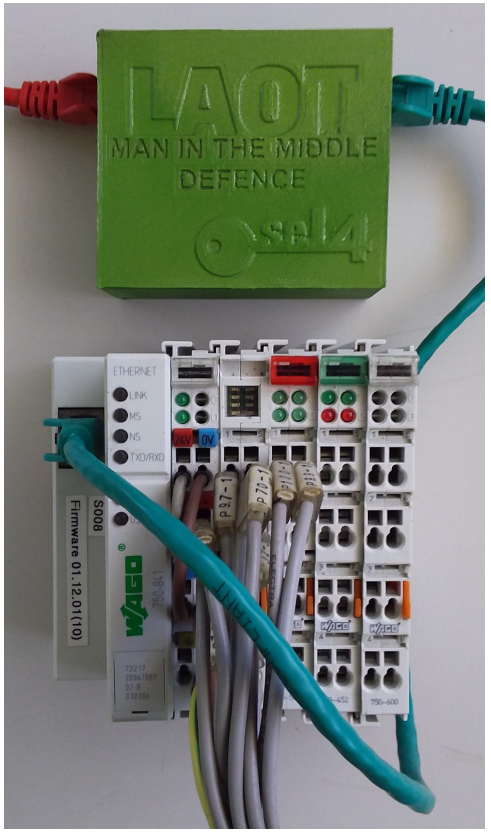




Commercial cars (NIO), Sep'24

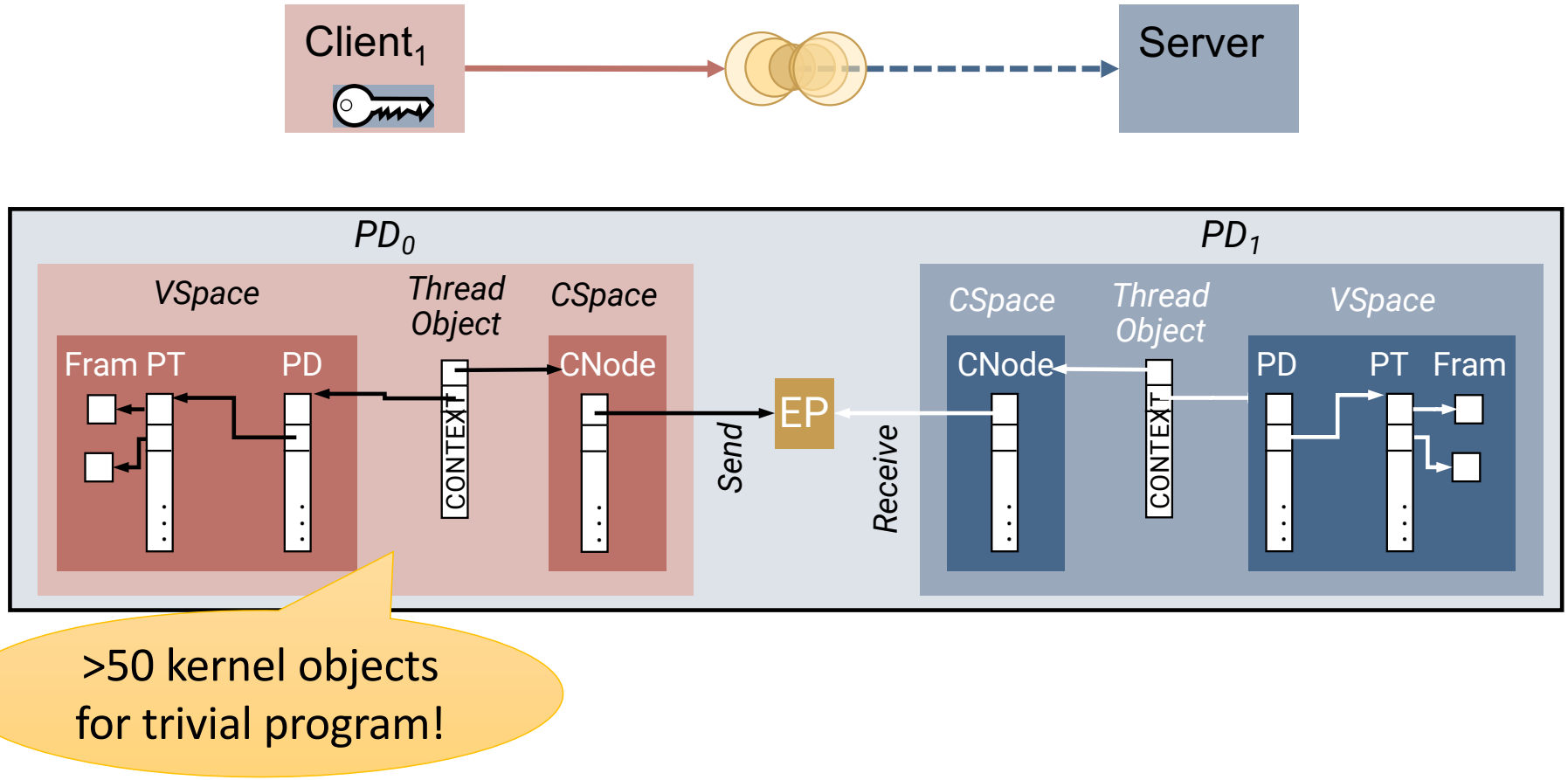






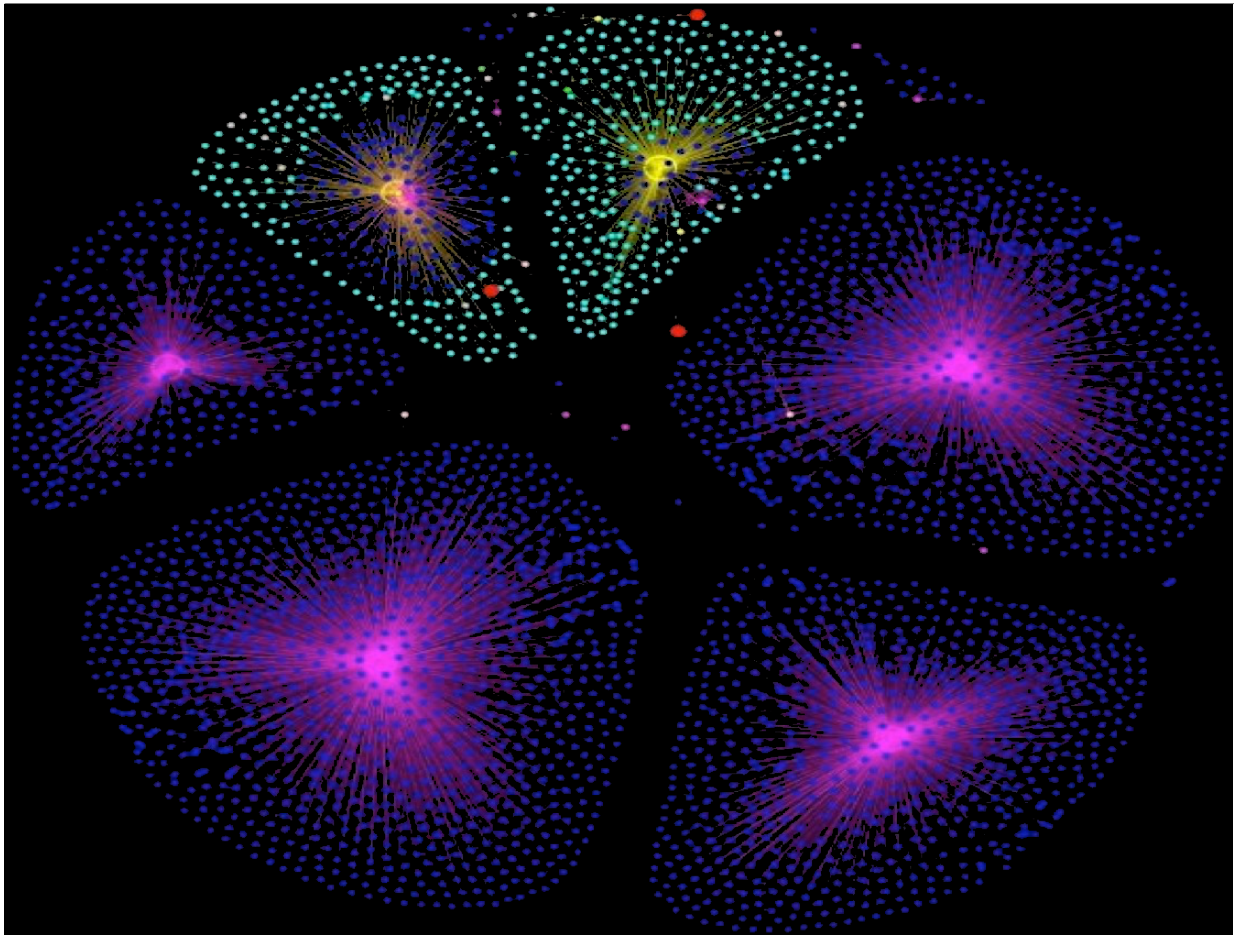
© Gernot Heiser 2025 – CC BY 4.0

Issue: seL4 Objects are Low-Level



© Gernot Heiser 2025 – CC BY 4.0

Simple But Non-Trivial System



© Gernot Heiser 2025 – CC BY 4.0

Microkernel: Assembly Language of OS

seL4 provides

- threads
- scheduling contexts
- pages
- endpoints
- notifications
- …

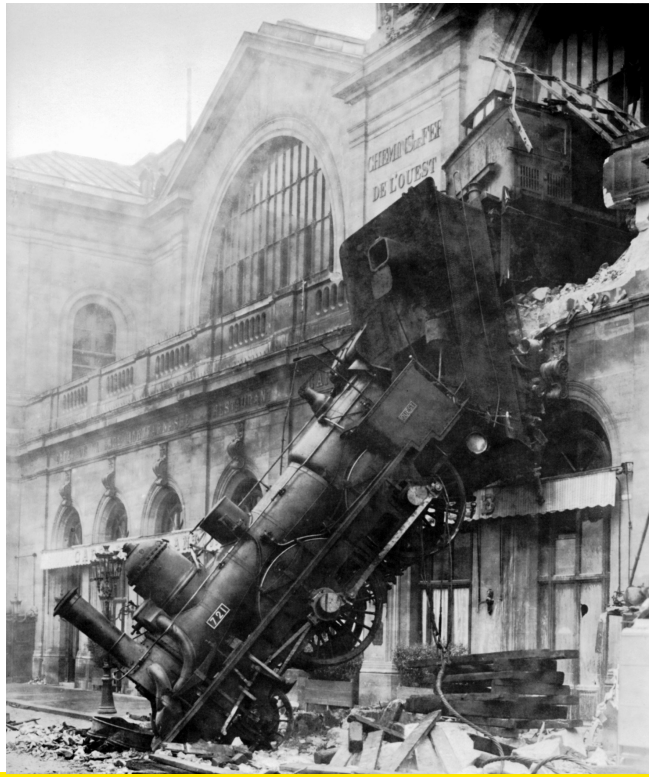
Result: everyone builds their own

... but good design on seL4 requires deep expertise

Programmer wants

- Processes
- Sockets
- Files

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

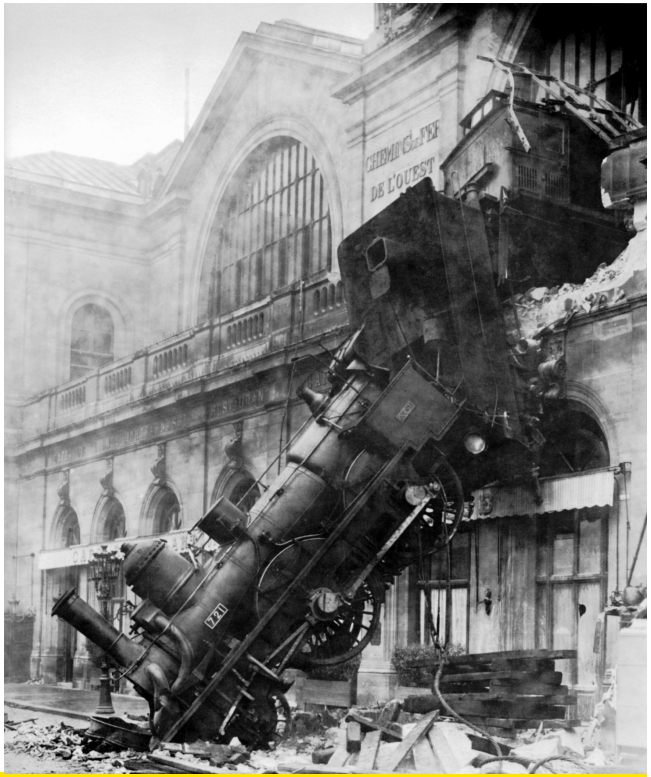


© Gernot Heiser 2025 – CC BY 4.0

Take-Away: seL4 Is Too Hard To Use!

Need seL4-based OS that is:

- secure
- performant
- easy to use
- open source



© Gernot Heiser 2025 – CC BY 4.0

Step 1: Microkit

Stop The Train Wrecks!



© Gernot Heiser 2025 – CC BY 4.0



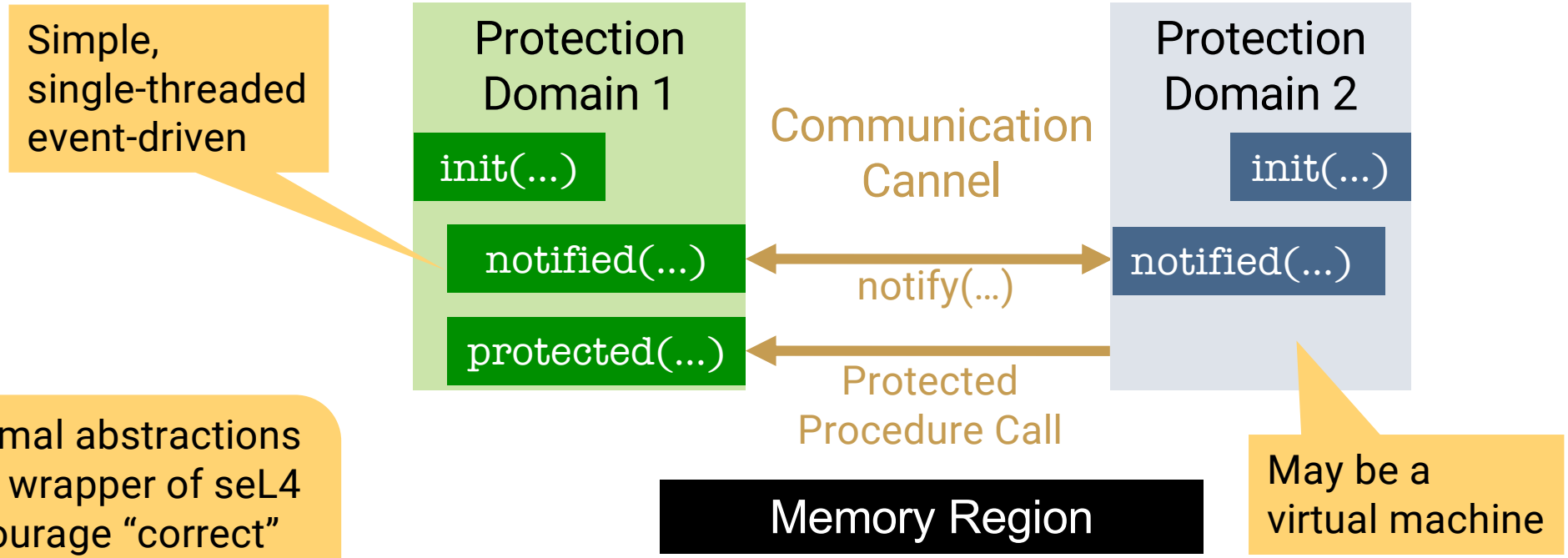
Step 1: Microkit – Simple seL4 Abstraction

Minimal base for IoT, cyberphysical, other embedded use

- Restrict to static architectures
  - i.e. components & communication channels defined at build time
- Ease development and deployment
  - SDK, integrate with build system of your choice
  - Highly approachable tutorial
- Retain near-minimal trusted computing base (TCB)
  - TCB suitable for formal verification
- Retain seL4's superior performance

© Gernot Heiser 2025 – CC BY 4.0

Microkit Abstractions



- Minimal abstractions
- Thin wrapper of seL4
- Encourage “correct” use of seL4 primitives
- Static architecture

© Gernot Heiser 2025 – CC BY 4.0



libmicrokit: Event-handler loop

```
1.  for (;;) {
2.      if (have_reply) {
3.          tag = seL4_ReplyRecv(INPUT_CAP, reply_tag, &badge, REPLY_CAP);
4.      } else if (have_signal) {
5.          tag = seL4_NBSendRecv(signal, signal_msg, INPUT_CAP, &badge, REPLY_CAP);
6.          have_signal = false;
7.      } else {
8.          tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
9.      }
10.     event_handle(badge, &have_reply, &reply_tag, &notified);
11. }
```


© Gernot Heiser 2025 – CC BY 4.0

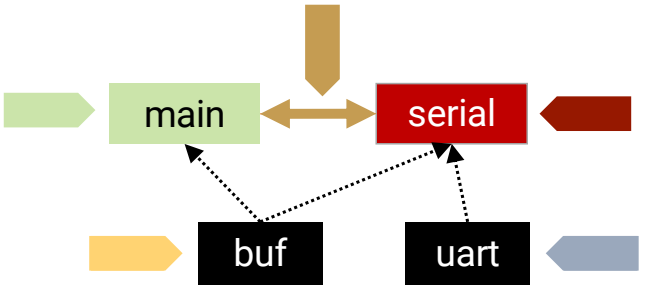
libmicrokit: Invoking user code

```
1.  event_handle(badge, &have_reply, &reply_tag, &notified) {
2.      if ((have_reply) = badge >> 63) {
3.          reply_tag = protected(badge & 0x3f, tag);
4.      } else {
5.          unsigned int idx = 0;
6.          do  {
7.              if (badge & 1) {
8.                  notified(idx);
9.              }
10.             badge >>= 1; idx++;
11.         } while (badge != 0);
12.     }
13.  }
```


© Gernot Heiser 2025 – CC BY 4.0

Microkit System Description File (SDF)

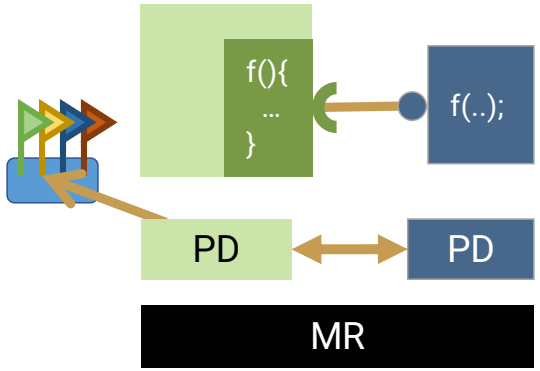
```
1.    <system>
2.        <memory_region name="uart" size="0x1000" phys_addr="0x9000000" />
3.        <memory_region name="buf" size="0x1000" />
4.        <protection_domain name="serial" priority="250">
5.            <irq irq="33" id="0" />
6.            <program_image path="serial_server.elf" />
7.            <map mr="uart" vaddr="0x4000000" perms="rw" cached="false" … />
8.            <map mr="buf" vaddr="0x4001000" perms="rw" setvar_vaddr="input" />
9.        </protection_domain>
10.       <protection_domain name="main">
11.           <program_image path="main.elf" />
12.       </protection_domain>
13.       <channel>
14.           <end pd="serial" id="1" />
15.           <end pd="client" id="0" />
16.       </channel>
17.   </system>
```



© Gernot Heiser 2025 – CC BY 4.0

Microkit Status

- Easy to use – non-experts productive within hours
- Supports AArch64, RV64, x64
- Limited dynamic features:
  - fault handlers
  - start/stop protection domains
  - empty protection domains (for late app loading)
  - re-initialise protection domains
  - “template PDs” – discretionary access



© Gernot Heiser 2025 – CC BY 4.0



Step 2: LionsOS

Stop The Train Wrecks!



© Gernot Heiser 2025 – CC BY 4.0



LionsOS Aims: Fast, Secure, Adaptable

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: *Best-performing* microkernel-based OS ever

Aim 3: *Most secure* OS ever

Must be verified!

© Gernot Heiser 2025 – CC BY 4.0



Lions OS: Principles

Least Privilege

Strict separation of concerns

Overarching principle: KISS
“Keep it simple, stupid!”

Radical simplicity

Use-case–specific policies

Fine-grained modularity

© Gernot Heiser 2025 – CC BY 4.0



Radical Simplicity™

Provide **exactly** the functionality needed, not more

Simple programming model:

- strictly sequential code (Microkit)
- event-based (Microkit)
- single-producer, single-consumer queues
- location transparency
- …

Static **architecture**, mostly static resource management

© Gernot Heiser 2025 – CC BY 4.0

What's An OS?

Security

- Access control

Resource Mgmt

- Memory
- CPU
- Storage
- NW bandwidth

Services

- Files
- Networking

© Gernot Heiser 2025 – CC BY 4.0



What's A Statically Architected OS?

Security

- Access control

Resource Mgmt

- Memory
- CPU
- Storage
- NW bandwidth

Services

- Files
- Networking

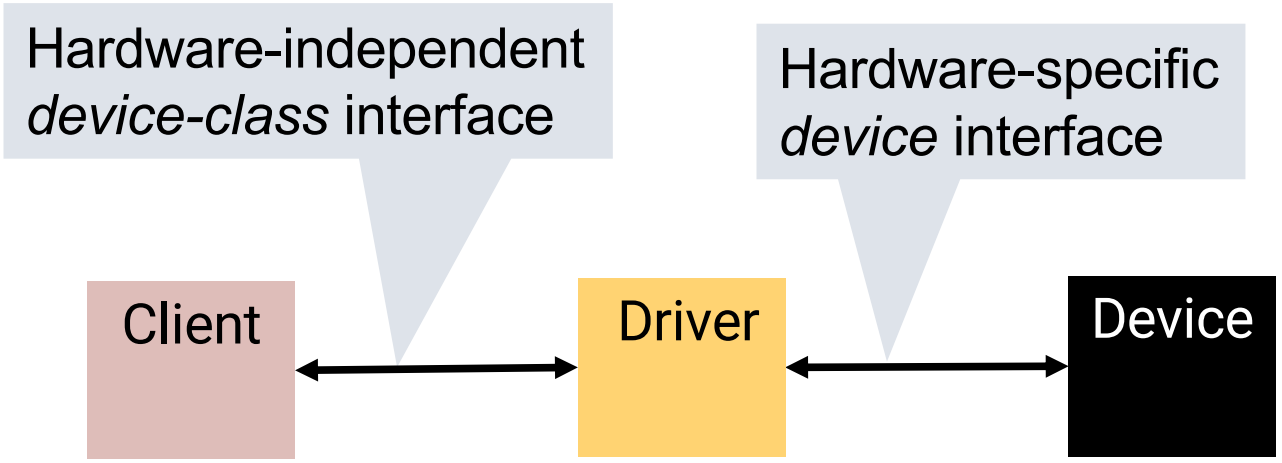
By architecture!

Policies

© Gernot Heiser 2025 – CC BY 4.0



Device Drivers: Separation of Concerns

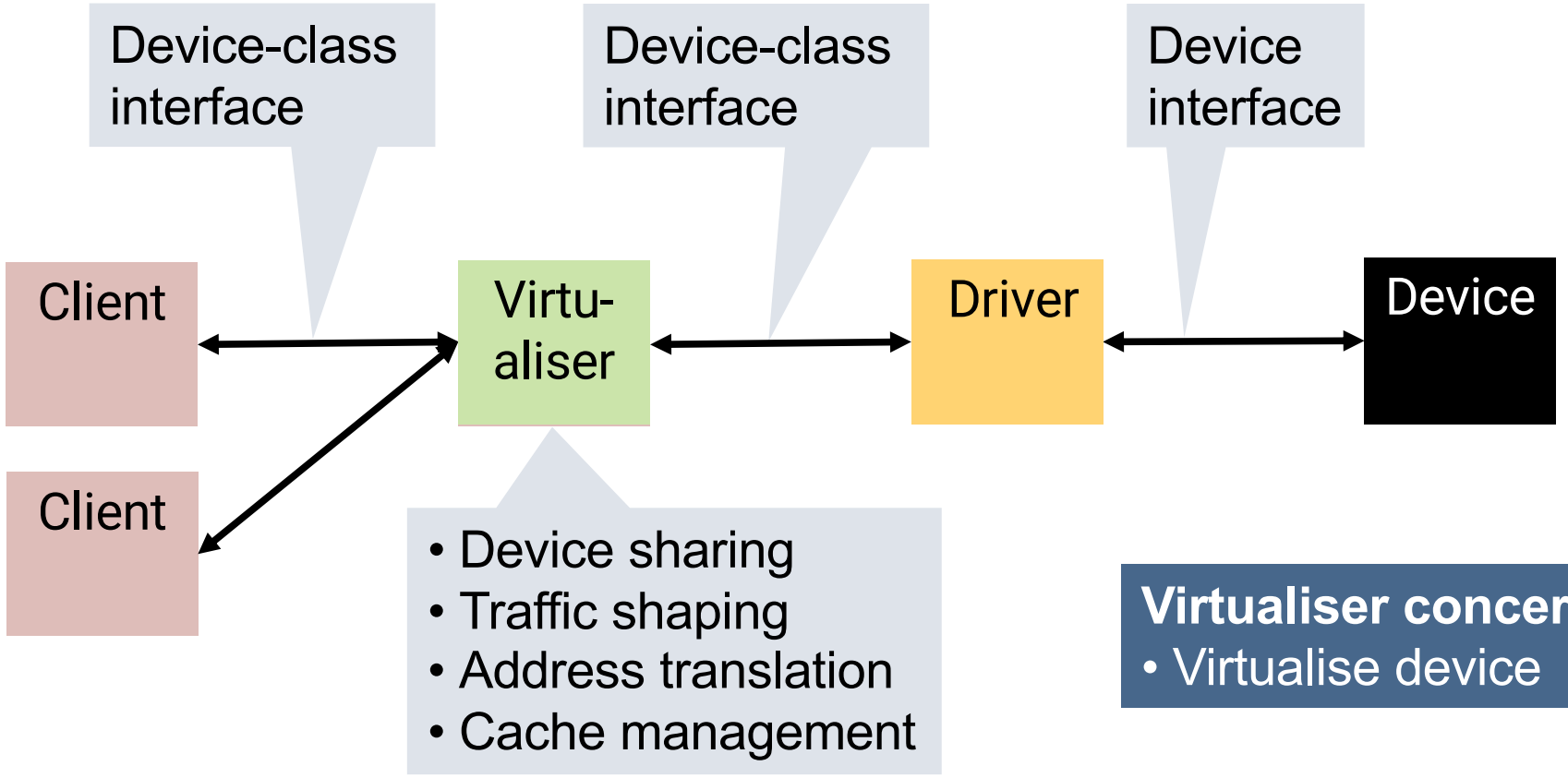


Driver concern:
- abstract over specific hardware

© Gernot Heiser 2025 – CC BY 4.0



Device Drivers: Device Sharing?



© Gernot Heiser 2025 – CC BY 4.0



Driver Programming Model

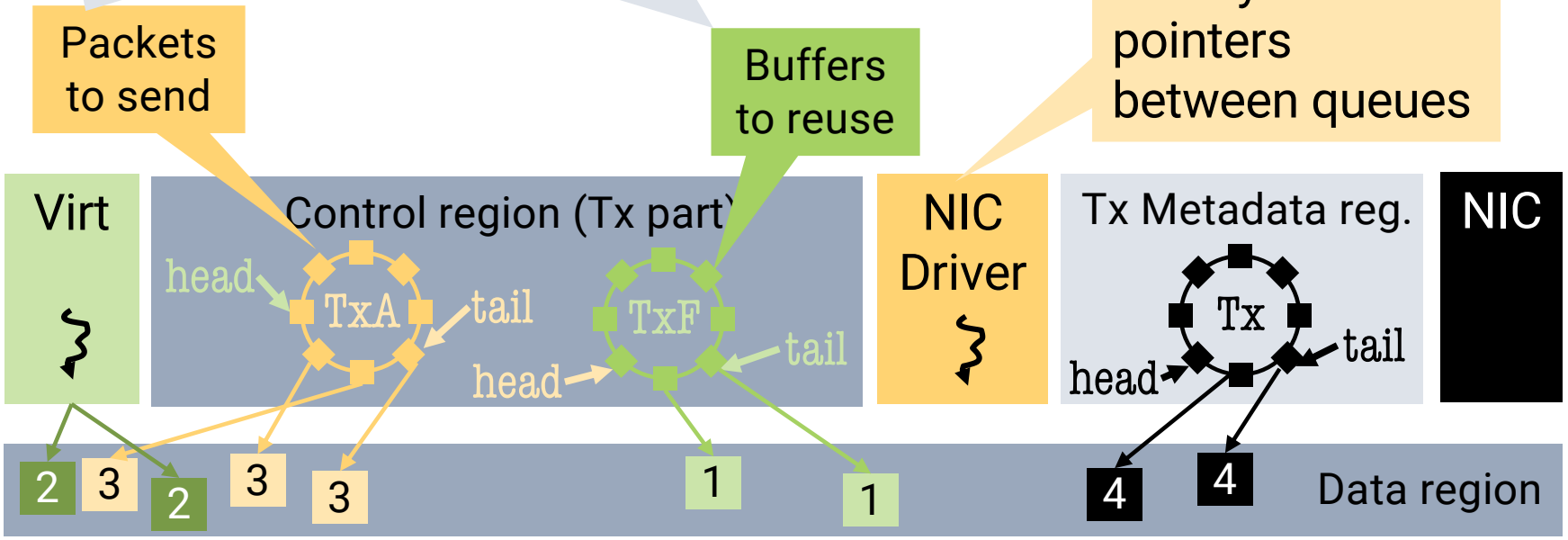
- Lock-free, bounded queues
- Single producer, single consumer
- Similar to ring buffers used by NICs

Zero-copy data transfer

Packets to send

Buffers to reuse

Mostly moves pointers between queues



© Gernot Heiser 2025 – CC BY 4.0

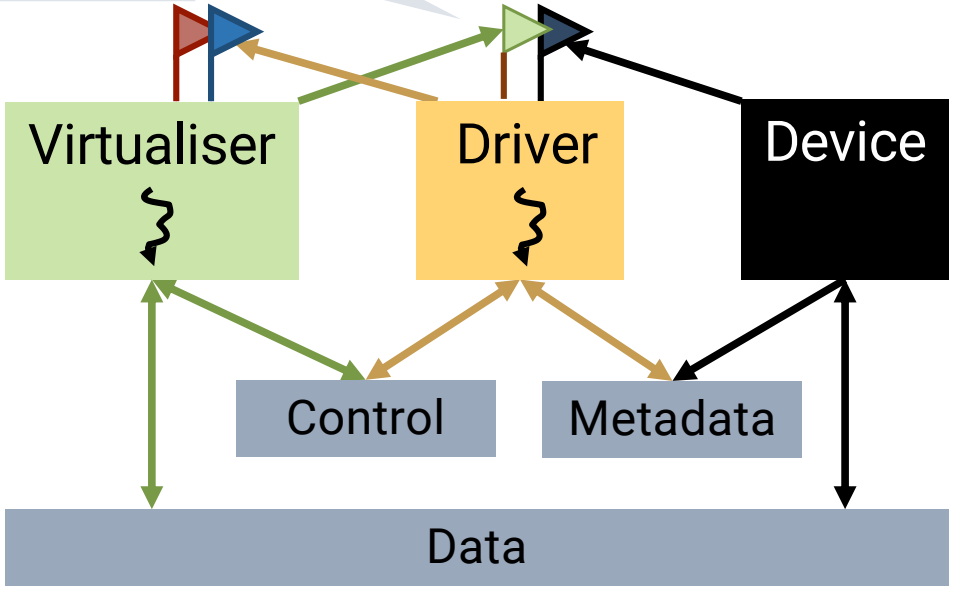


Device Drivers: Synch & Least Privilege

Queues synchronised via Semaphores (Notifications):

- "queue not empty"
- "queue not full"

Driver does not need access to data region!



© Gernot Heiser 2025 – CC BY 4.0



Use-Case–Specific Policies

Source of massive complexity

'80s model of computer use!

~~**Traditional OS**: achieve adaptability by universal policies~~

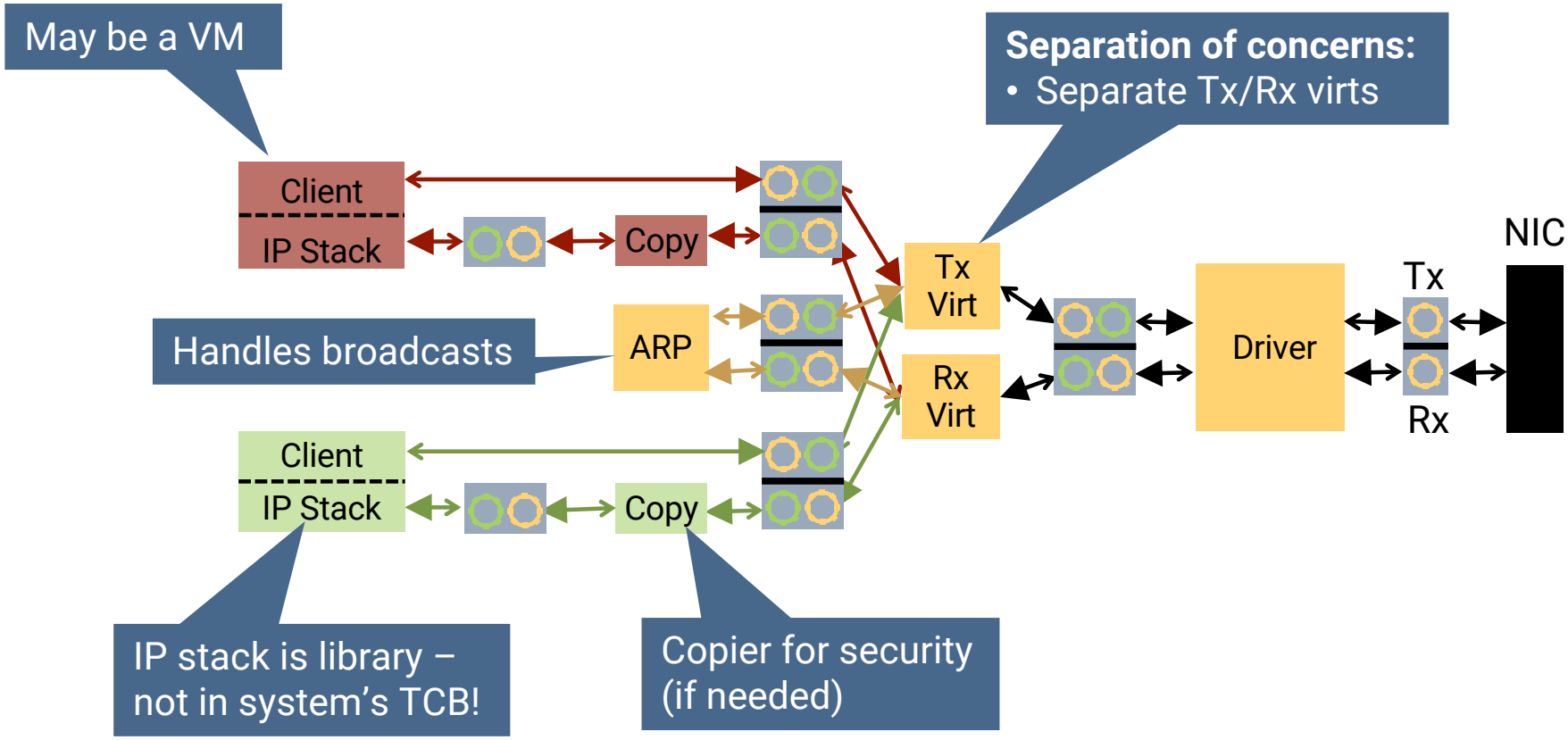
Use-case diversity through policies that are:
- optimised for one specific use case
- simple, localised implementation
- easy to replace by swapping component

Eg virtualiser traffic shaping:
- round-robin vs
- client priority vs
- bandwidth limit

© Gernot Heiser 2025 – CC BY 4.0



Example: Networking Subsystem



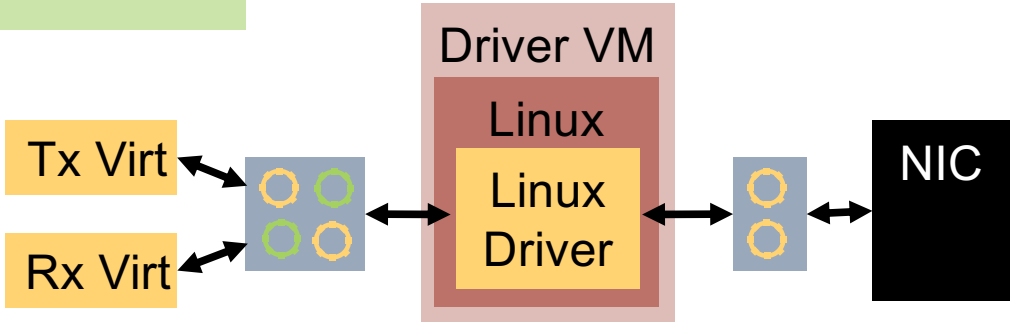
© Gernot Heiser 2025 – CC BY 4.0



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

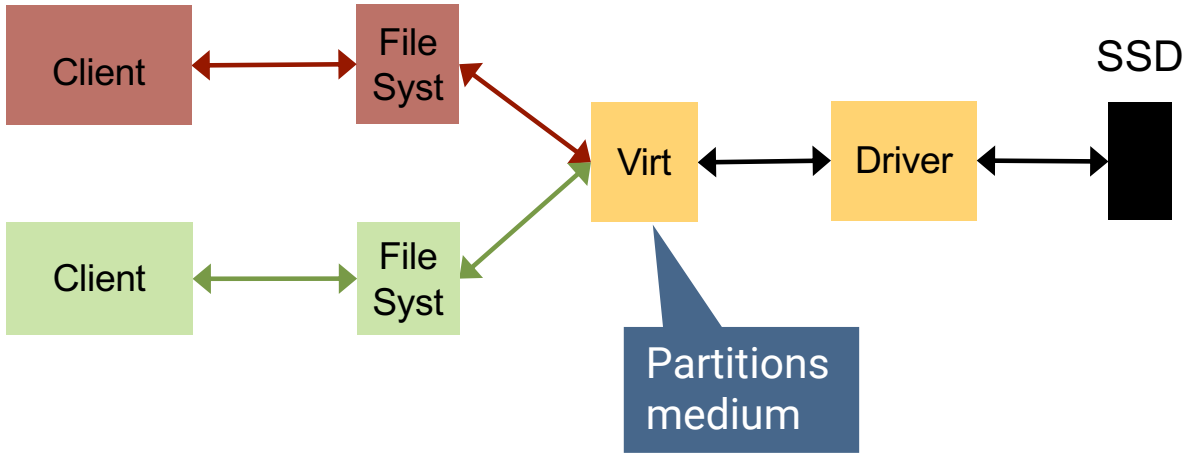


© Gernot Heiser 2025 – CC BY 4.0



Example: Storage Subsystem

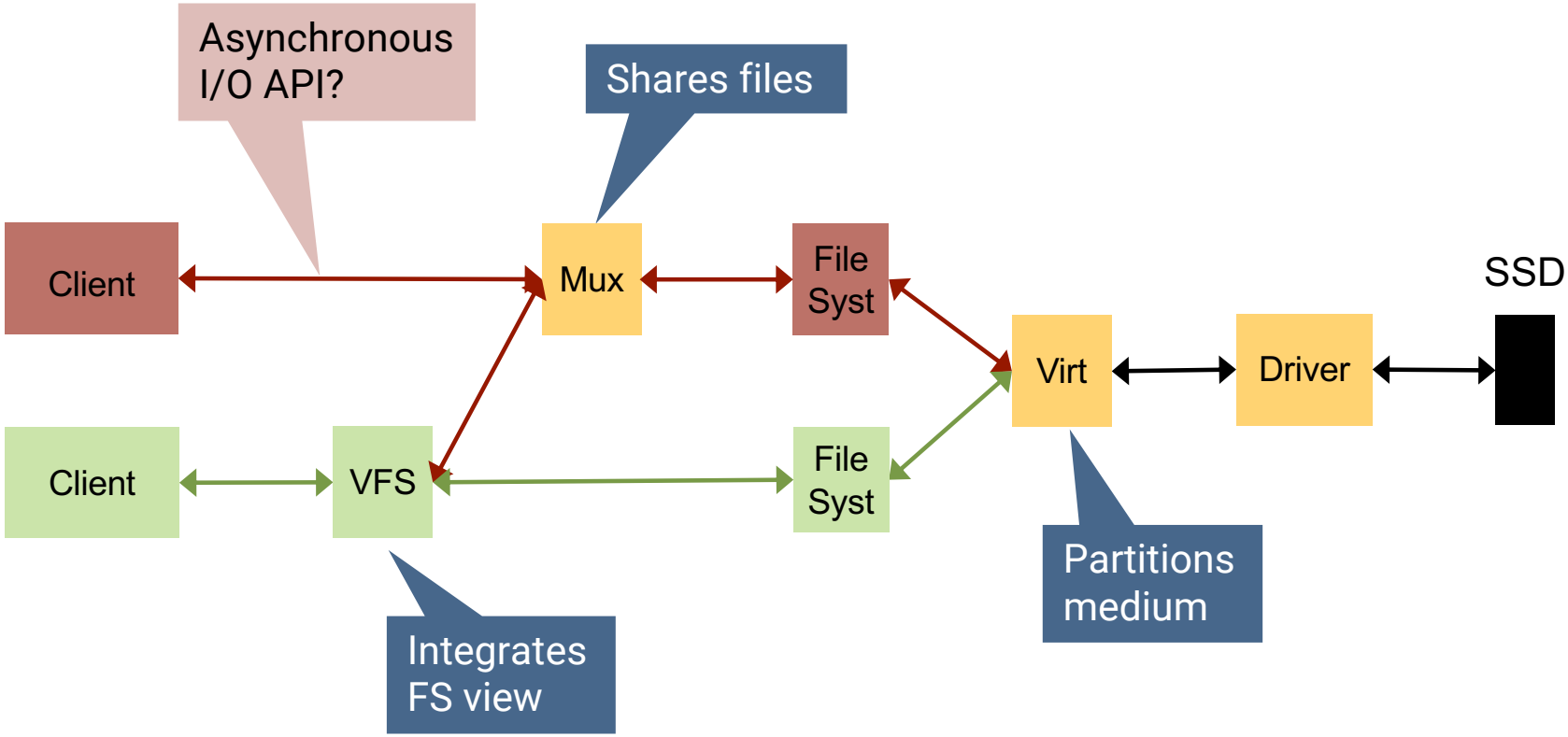
How about sharing?



© Gernot Heiser 2025 – CC BY 4.0



File Sharing



© Gernot Heiser 2025 – CC BY 4.0

Synchronous vs Asynchronous I/O

Synchronous

- Simpler to program
- Can implement over async I/O

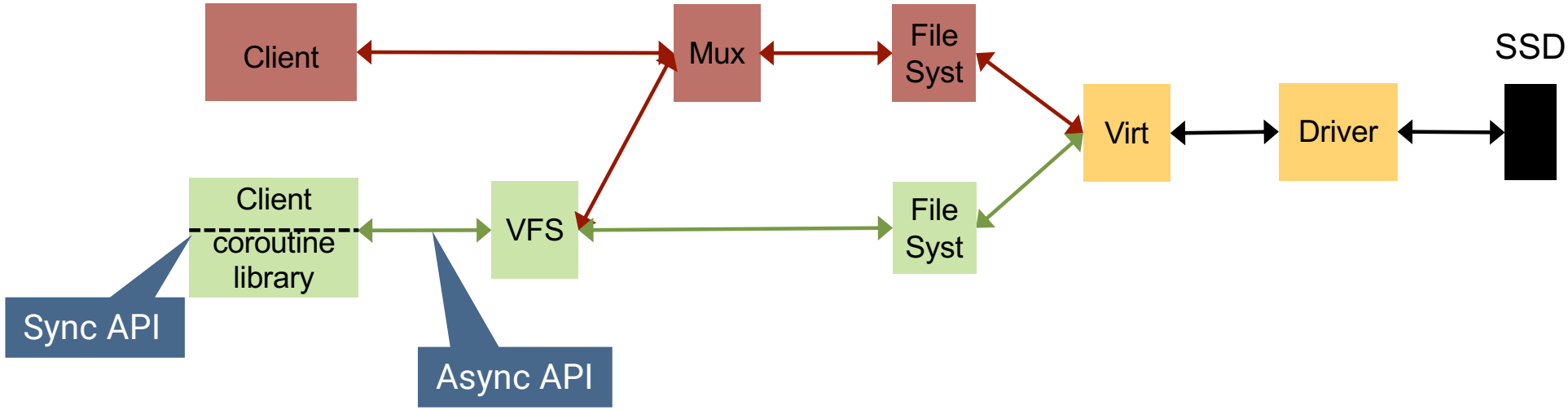
Asynchronous

- Minimal: Matches physical I/O
- Can implement over sync I/O
  … but more complex

© Gernot Heiser 2025 – CC BY 4.0



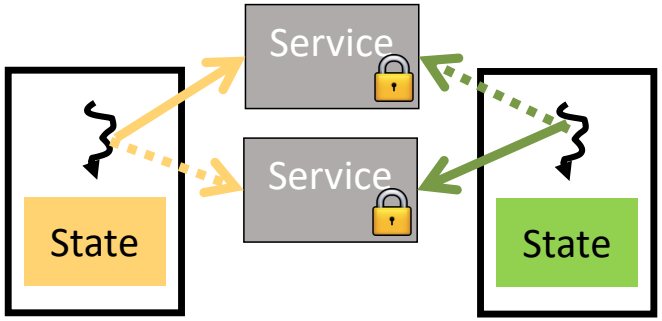
Synchronous vs Asynchronous API



© Gernot Heiser 2025 – CC BY 4.0

Remember: Threads vs Events

Threads

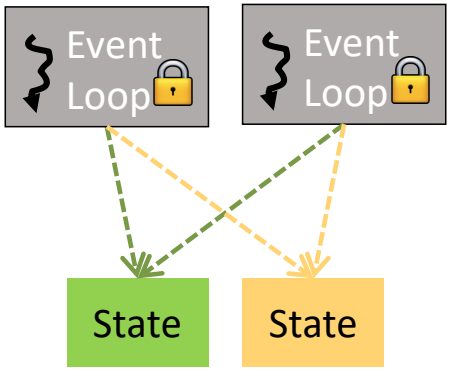


- OS thread per user process
  ⇒ extensive locking
- State with thread
- Suitable for multicore!

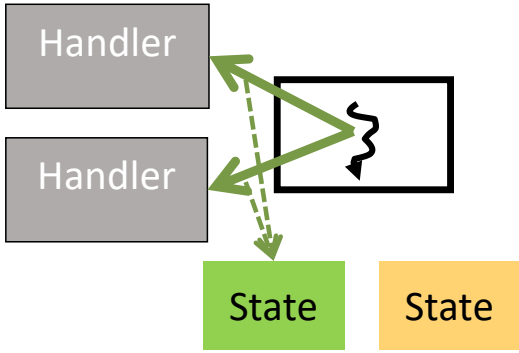
Many (micro-)services:
- 1 OS thread each
- Single-threaded
- Event-based
- Stateless

Suitable for multicore!

LionsOS



Events



- Stateless
- Single OS thread
  ⇒ no multicore!

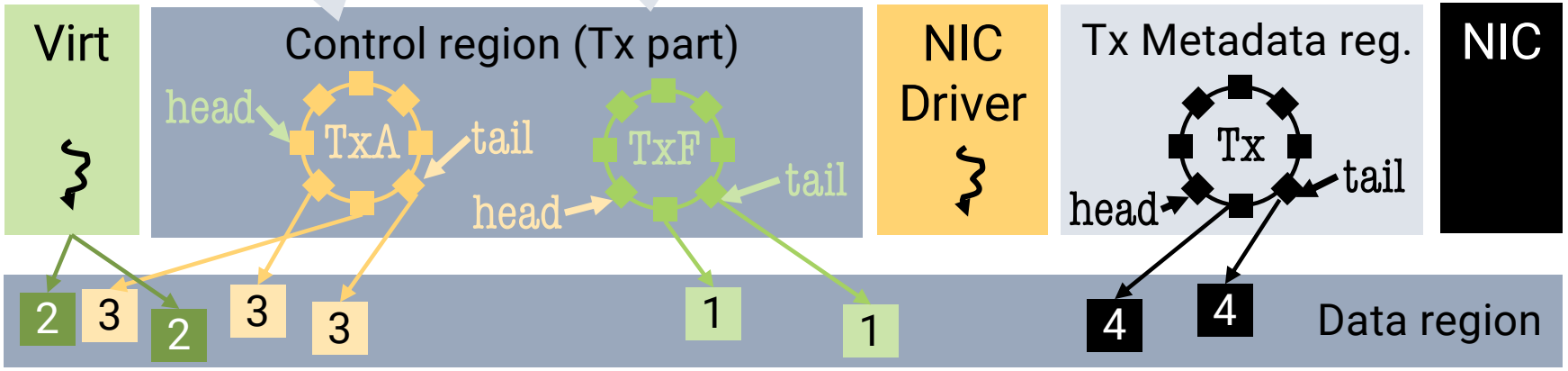
© Gernot Heiser 2025 – CC BY 4.0



Remember: Driver Programming Model

- Single-threaded modules
- Location transparent
- Concurrency control limited to en-/de-queue

- Lock-free, bounded queues
- Single producer, single consumer

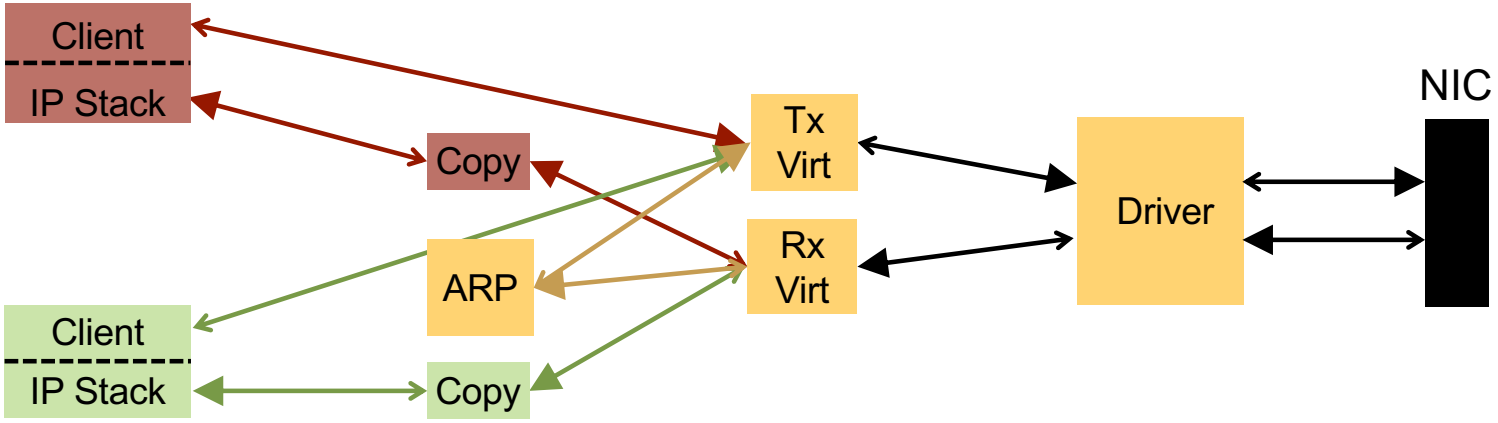


© Gernot Heiser 2025 – CC BY 4.0



Example: Networking Subsystem

Location transparent:
- Each component can be on any core
- Components can transparently migrate



© Gernot Heiser 2025 – CC BY 4.0

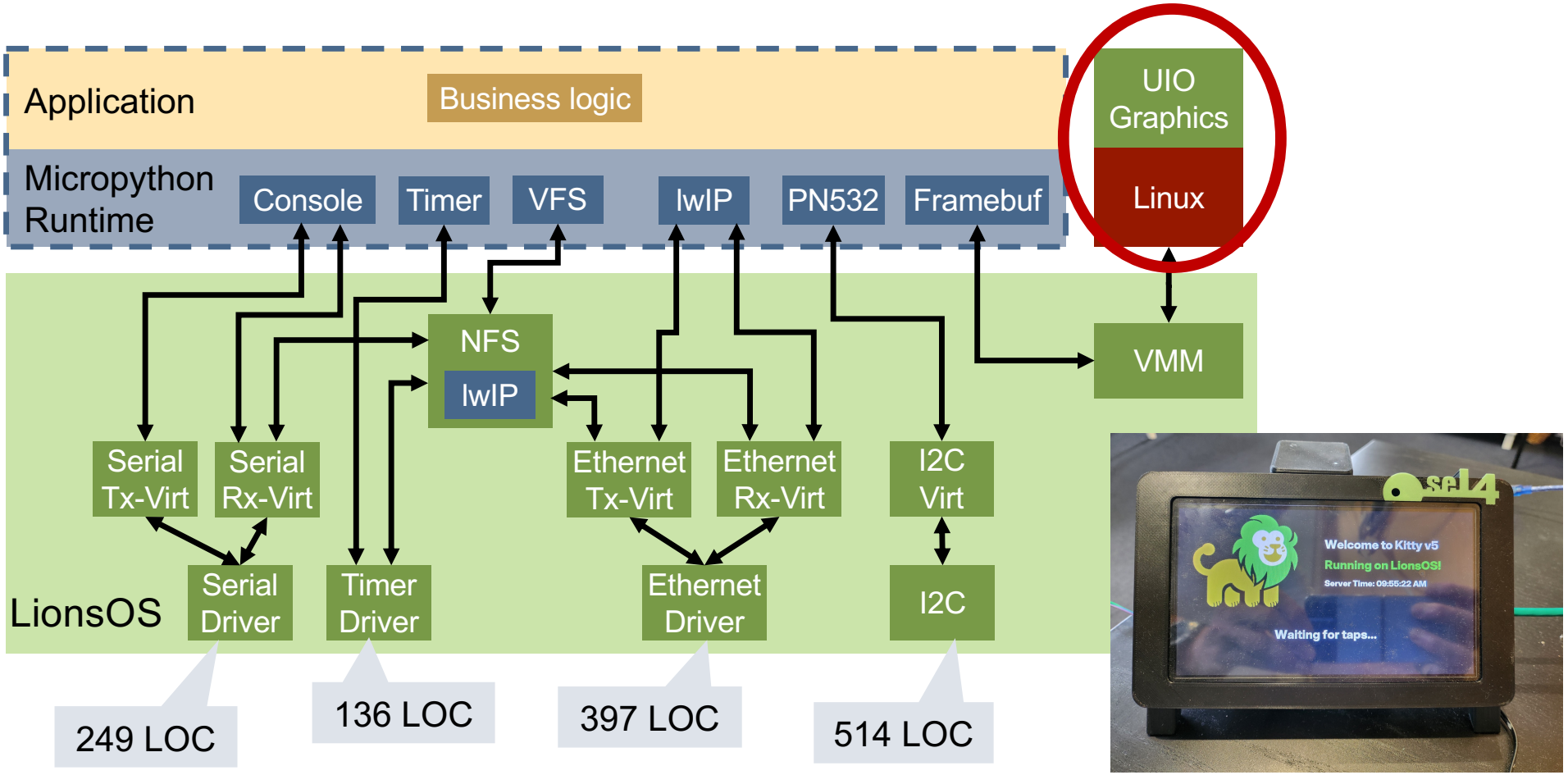


LionsOS Evaluation

© Gernot Heiser 2025 – CC BY 4.0



PoC: Point-of-Sale Terminal: “Kitty”



© Gernot Heiser 2025 – CC BY 4.0



PoS LionsOS Code Sizes (all C)

Trusted:
- 15 modules/ libraries
- Av 210 LoC

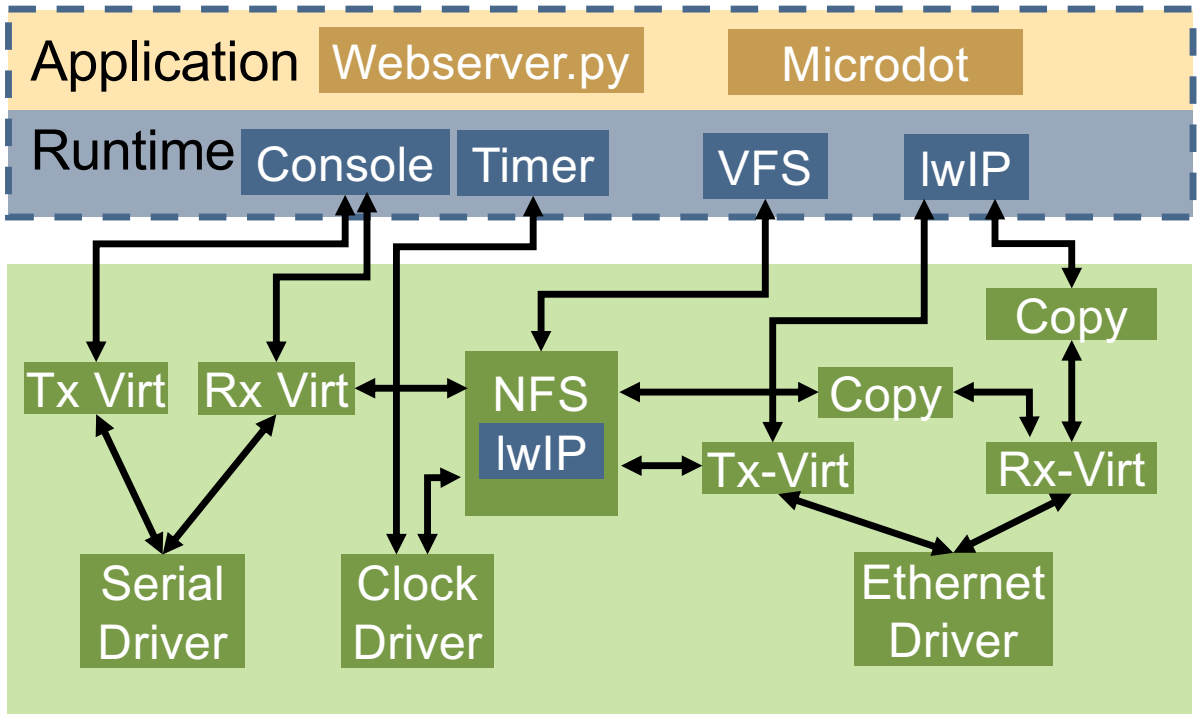
Component	LoC	Library	LoC
Serial Driver	249	Microkit	303
Serial Tx Virt	175	Serial queue	219
Serial Rx Virt	126	I²C queue	101
I²C Driver	514	Eth queue	140
I²C Virt	154	Filesys queue & protocol	268
Timer Driver	136		
Eth Driver	397	Coroutines	848
Eth Tx Virt	122	LWIP	16,280
Eth Rx Virt	160	NFS	45,707
Eth Copier	79	VMM	3,098

Untrusted

© Gernot Heiser 2025 – CC BY 4.0



Underneath https://sel4.systems/



© Gernot Heiser 2025 – CC BY 4.0



Comparison to Linux on i.MX8M

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

Performance?

Written by second-year student!

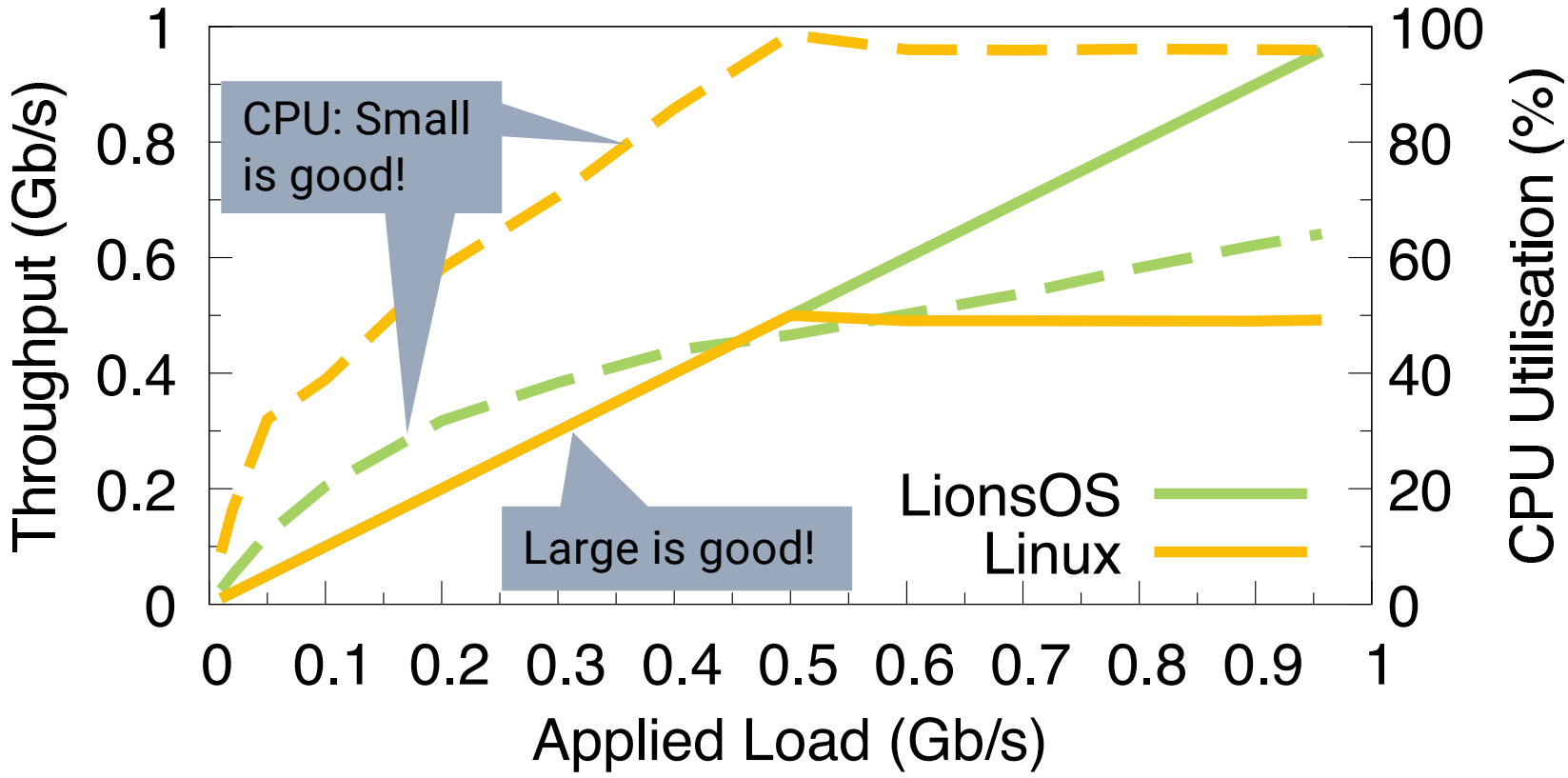
LionsOS:
- NW driver: 400 lines
- Virtualiser: 160 lines
- Copier: 80 lines
- IP stack: much simpler, client library
- shared NW system total: < 1,000 lines

Presently use lwip

© Gernot Heiser 2025 – CC BY 4.0



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



Single-core configuration

© Gernot Heiser 2025 – CC BY 4.0