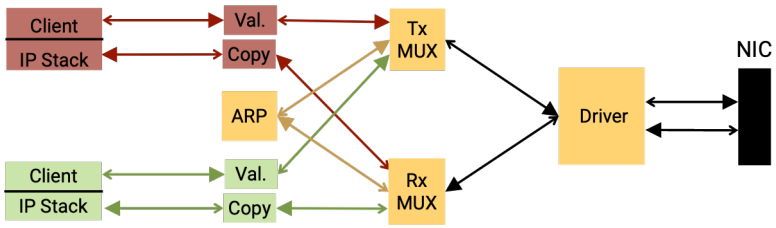
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2024 T3 Week 7 Part 2

seL4 in the Real World &
seL4 Research at TS@UNSW

@GernotHeiser



Copyright Notice

Today's Lecture

- seL4 in the real world
  - HACMS & incremental cyber-retrofit
  - Adption and seL4 Foundation
- seL4-related research at UNSW Trustworthy Systems
  - Usability 1: Microkit
  - Usability 2: Lions OS
  - Pancake: Verifying device drivers
  - Secure multi-server OS

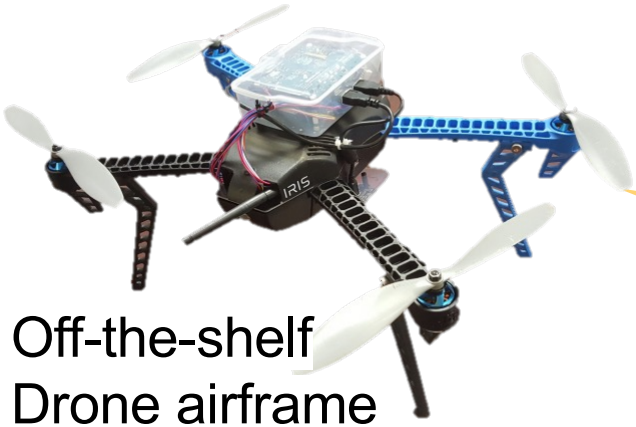
seL4 in the Real World

DARPA HACMS (2012–17)

Unmanned Little Bird (ULB)

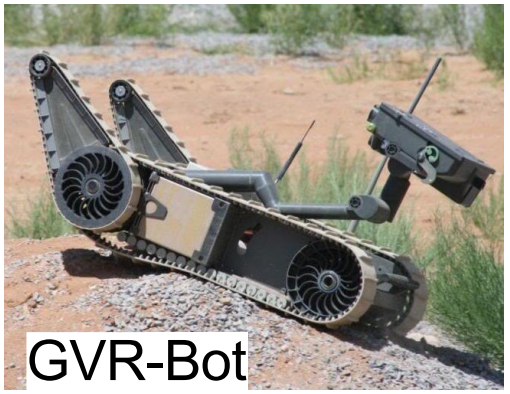
Retrofit existing system!

Autonomous trucks



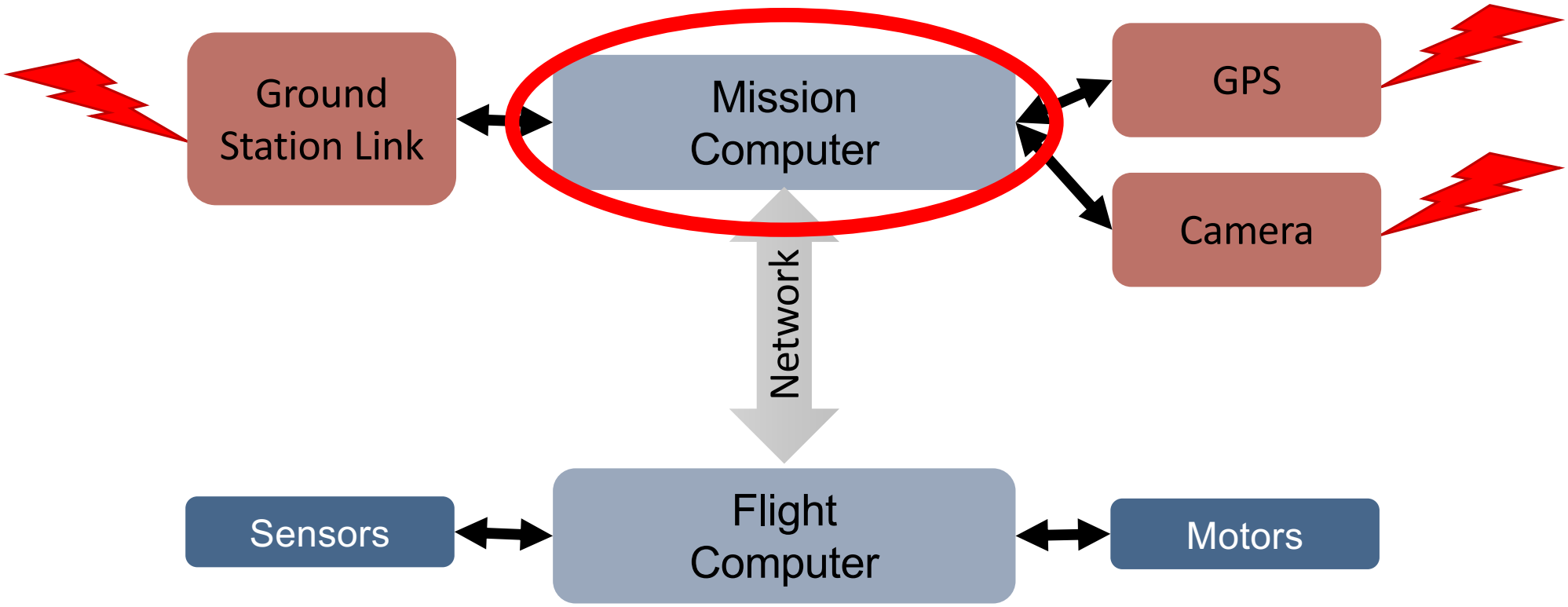
Off-the-shelf Drone airframe

Develop technology

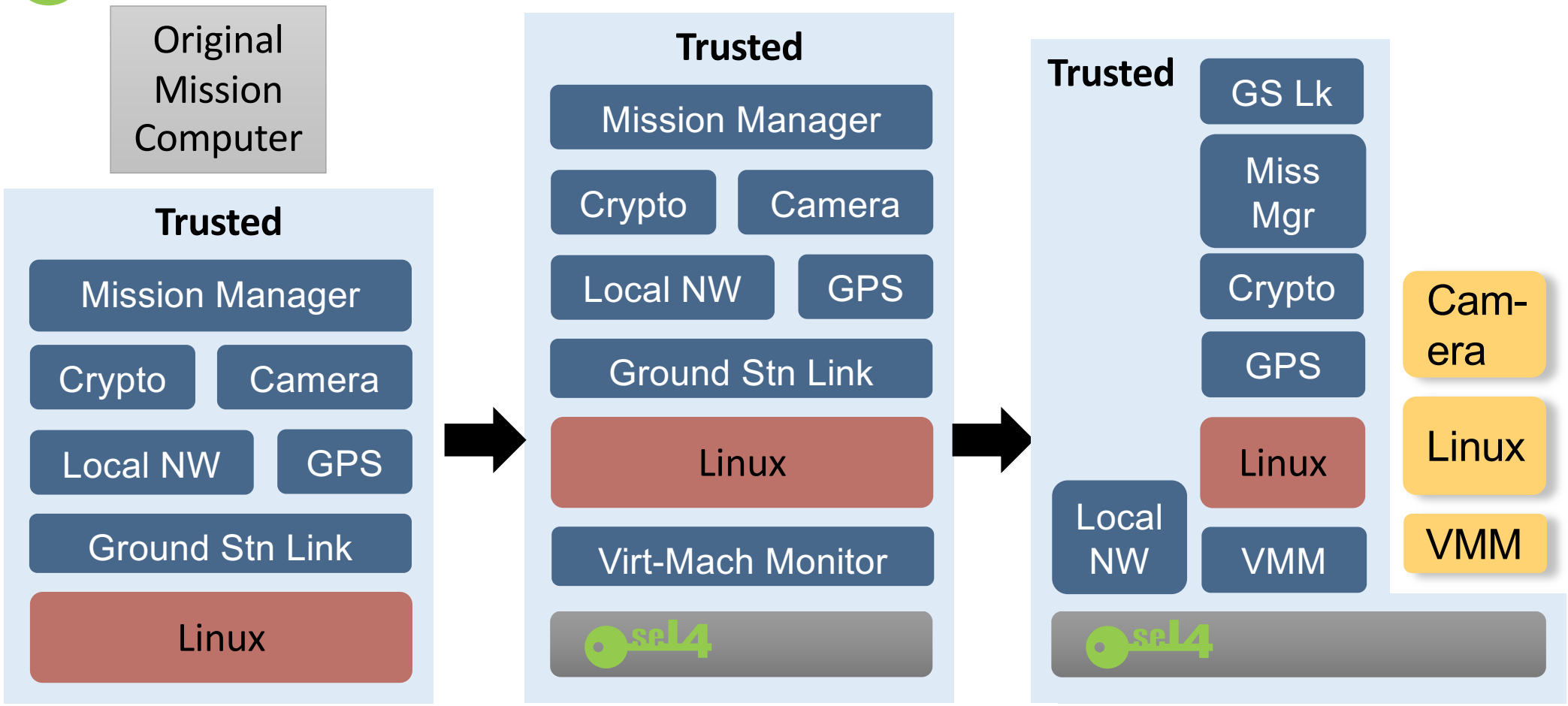


GVR-Bot

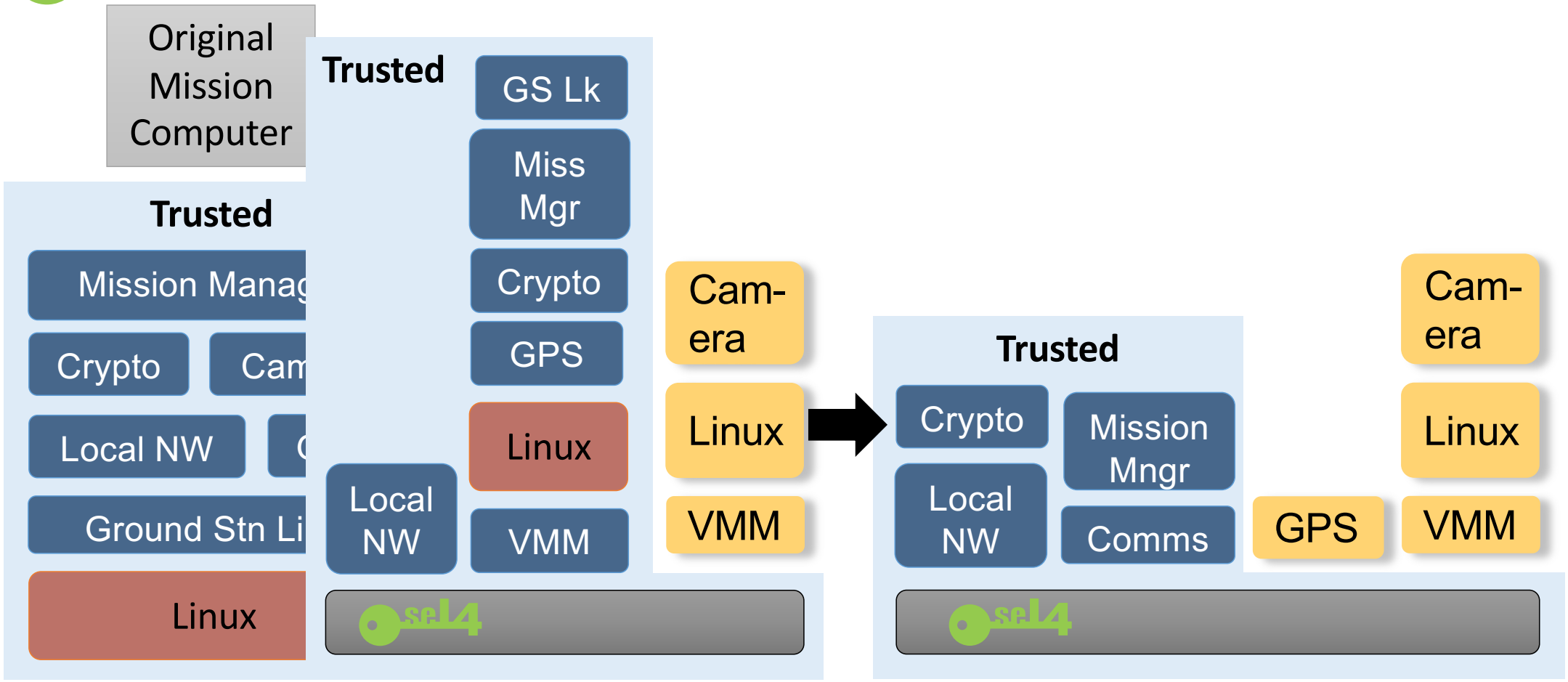
ULB Architecture



Incremental Cyber Retrofit



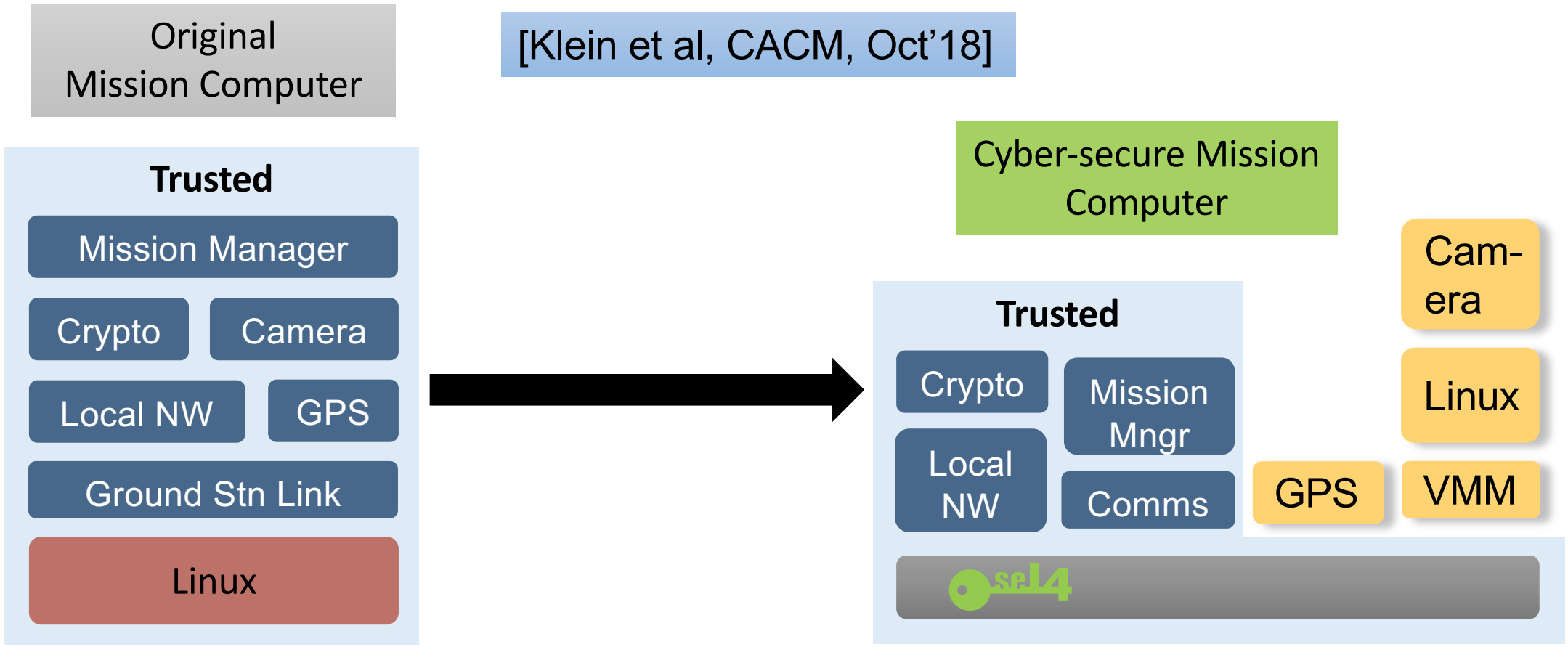
Incremental Cyber Retrofit



Incremental Cyber Retrofit

Original Mission Computer

[Klein et al, CACM, Oct'18]



World's Most Secure Drone

2021-08-06

← Tweet

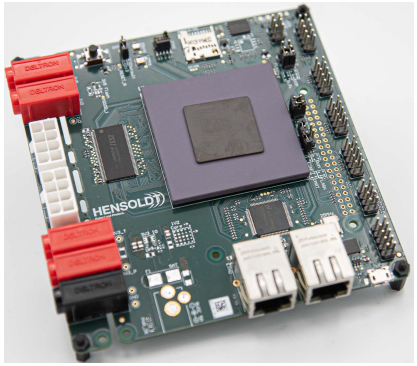
DARPA
@DARPA

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!

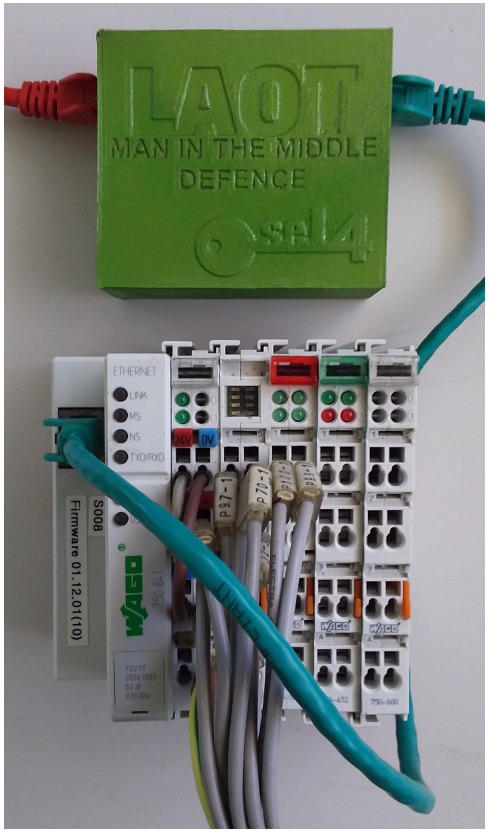
HACMS Outcomes & Consequences

- Demonstrated real-world suitability of seL4 and formal methods
  - Reversal of bad vibes from over-promising and under-delivering
  - Major re-think in US defence
- Dis-proved “security must be designed in from the start”
  - Retrofit is possible (under the right circumstances!)
- Led to follow-on funding for seL4 and deployment in the field
  - DARPA CASE, Feb’16 – Dec’22
  - seL4 Summits, since Nov’18 (initially sponsored by DARPA)
  - seL4 Foundation, since April’20
  - TII (UAE), Dec’21 – ongoing
  - NCSC (UK), Jan’22 – ongoing
  - DARPA PROVERS, Q1’24–Q3’26
  - More TBA soon!

seL4 in Products

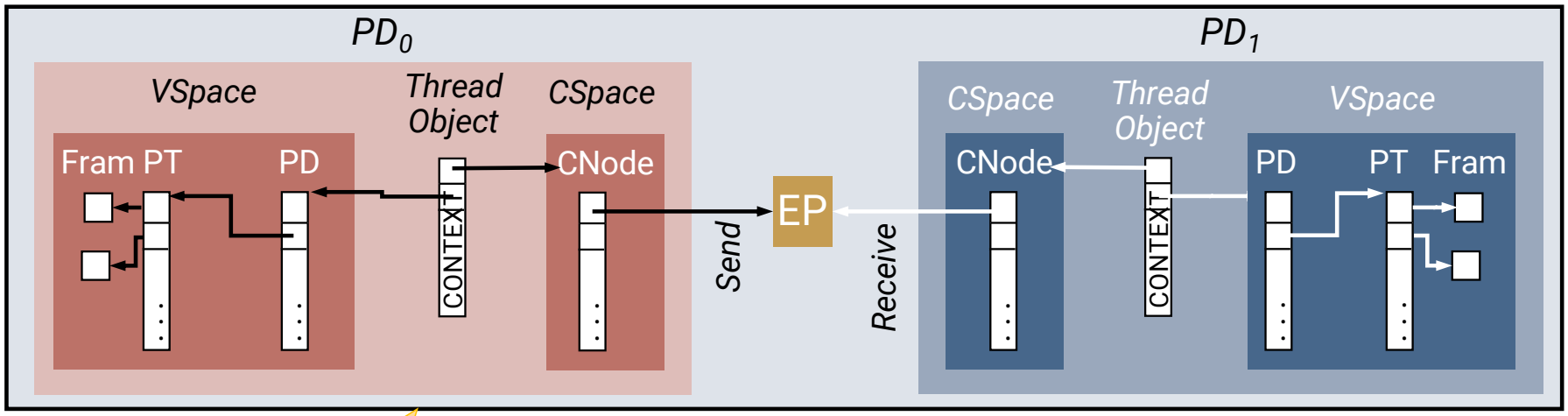


Commercial cars (NIO), Sep'24



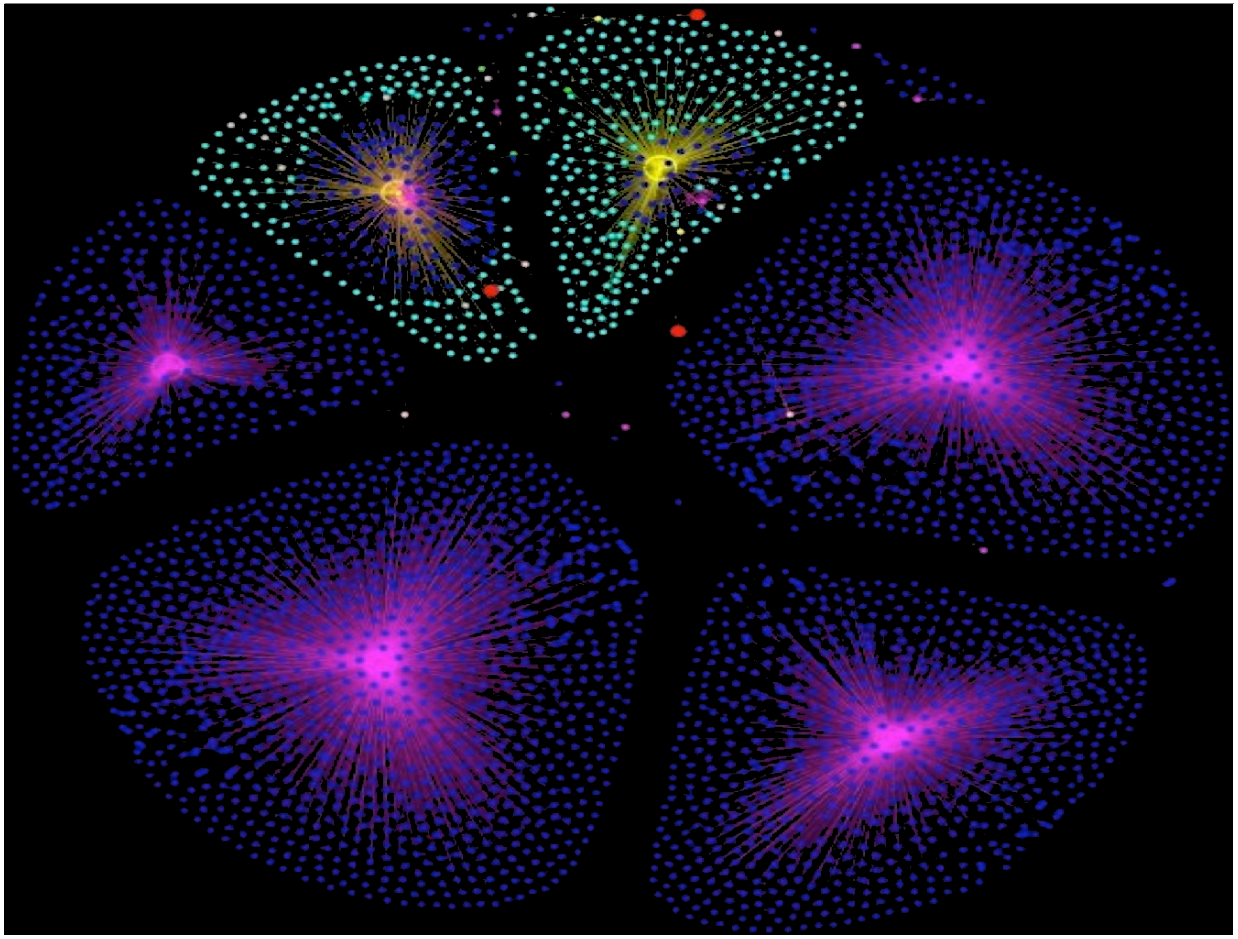
Usability Issues

Issue: seL4 Objects are Low-Level



>50 kernel objects for trivial program!

Simple But Non-Trivial System



Microkernel: Assembly Language of OS

seL4 provides

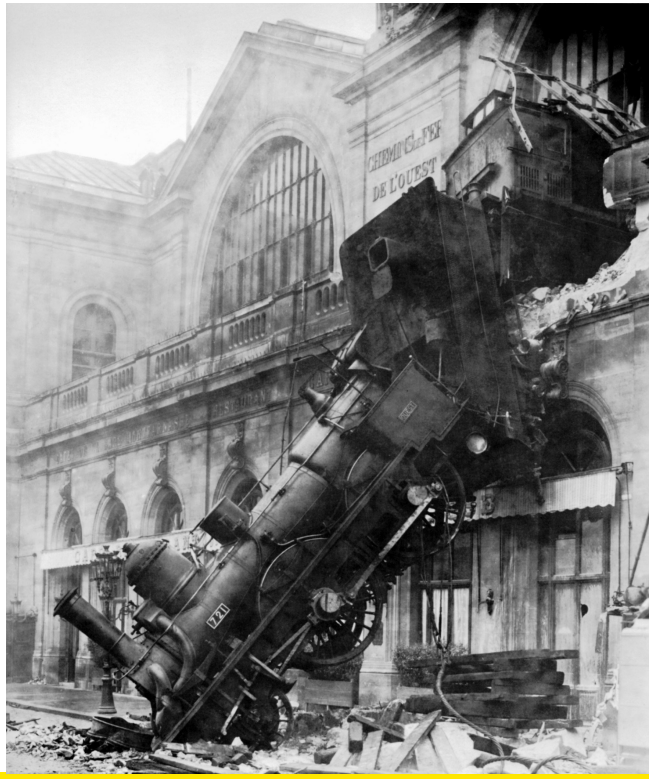
- threads
- scheduling contexts
- pages
- endpoints
- notifications
- …

Programmer wants

- Processes
- Sockets
- Files

Result: everyone builds their own

… but good design on seL4 requires deep expertise



Enter LionsOS

Stop The Train Wrecks!

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

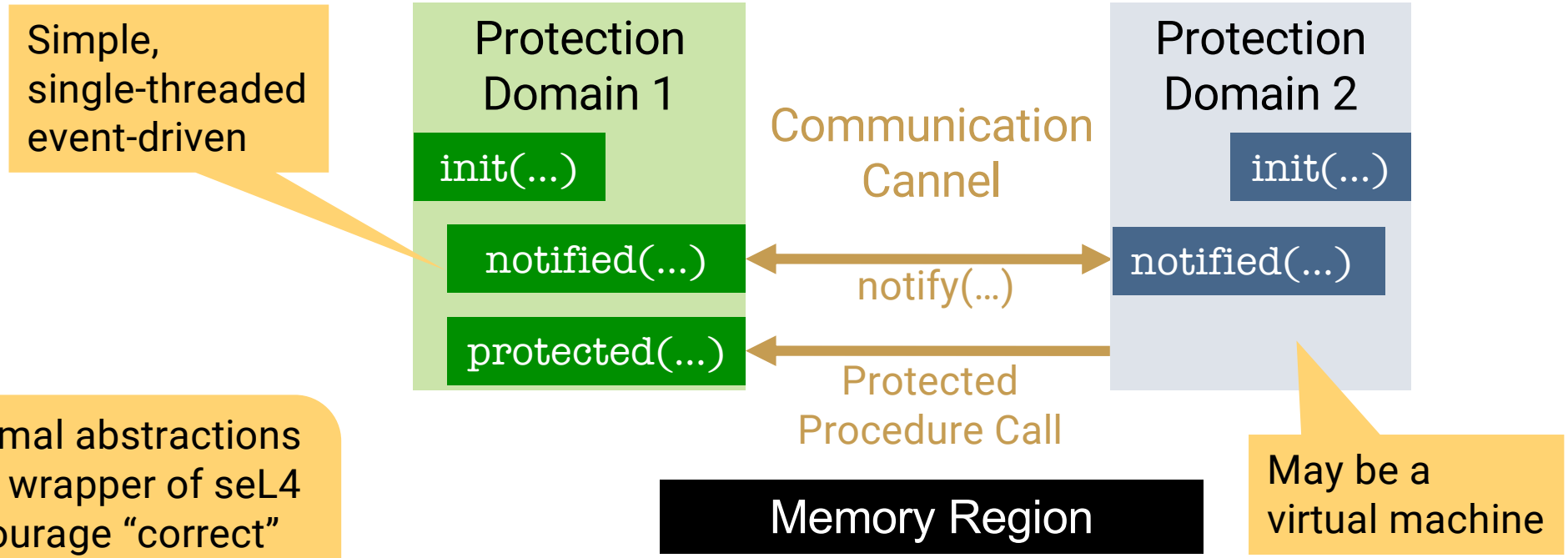
Must be verified!

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
- Retain near-minimal trusted computing base (TCB)
  - TCB suitable for formal verification
- Retain seL4's superior performance

Microkit Abstractions



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

libmicrokit: Event-handler loop

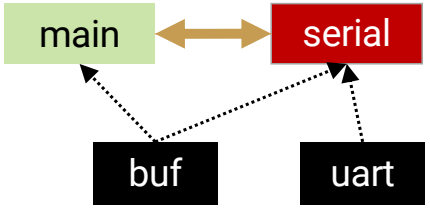
```
1.  for (;;) {
2.      if (have_reply) {
3.          tag = seL4_ReplyRecv(INPUT_CAP, reply_tag, &badge, REPLY_CAP);
4.      } else if (have_signal) {
5.          tag = seL4_NBSendRecv(signal, signal_msg, INPUT_CAP, &badge, REPLY_CAP);
6.          have_signal = false;
7.      } else {
8.         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

libmicrokit: Invoking user code

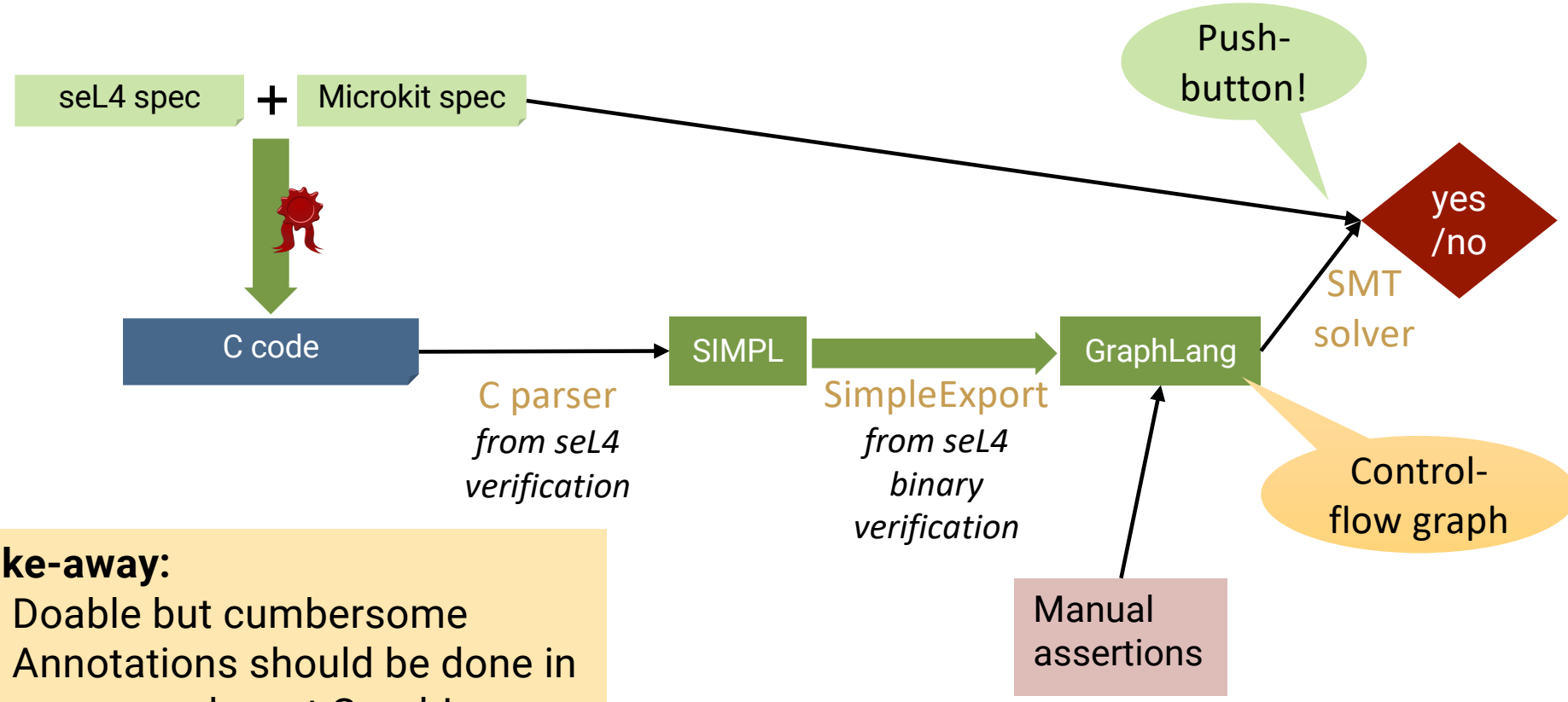
```
1.  event_handle(badge, &have_reply, &reply_tag, &notified) {
2.      if ((have_reply) = badge >> 63) {
3.          reply_tag = protected(badge & 0x3f, tag);
4.      } else {
5.          unsigned int idx = 0;
6.          do  {
7.              if (badge & 1) {
8.                  notified(idx);
9.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

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>
```



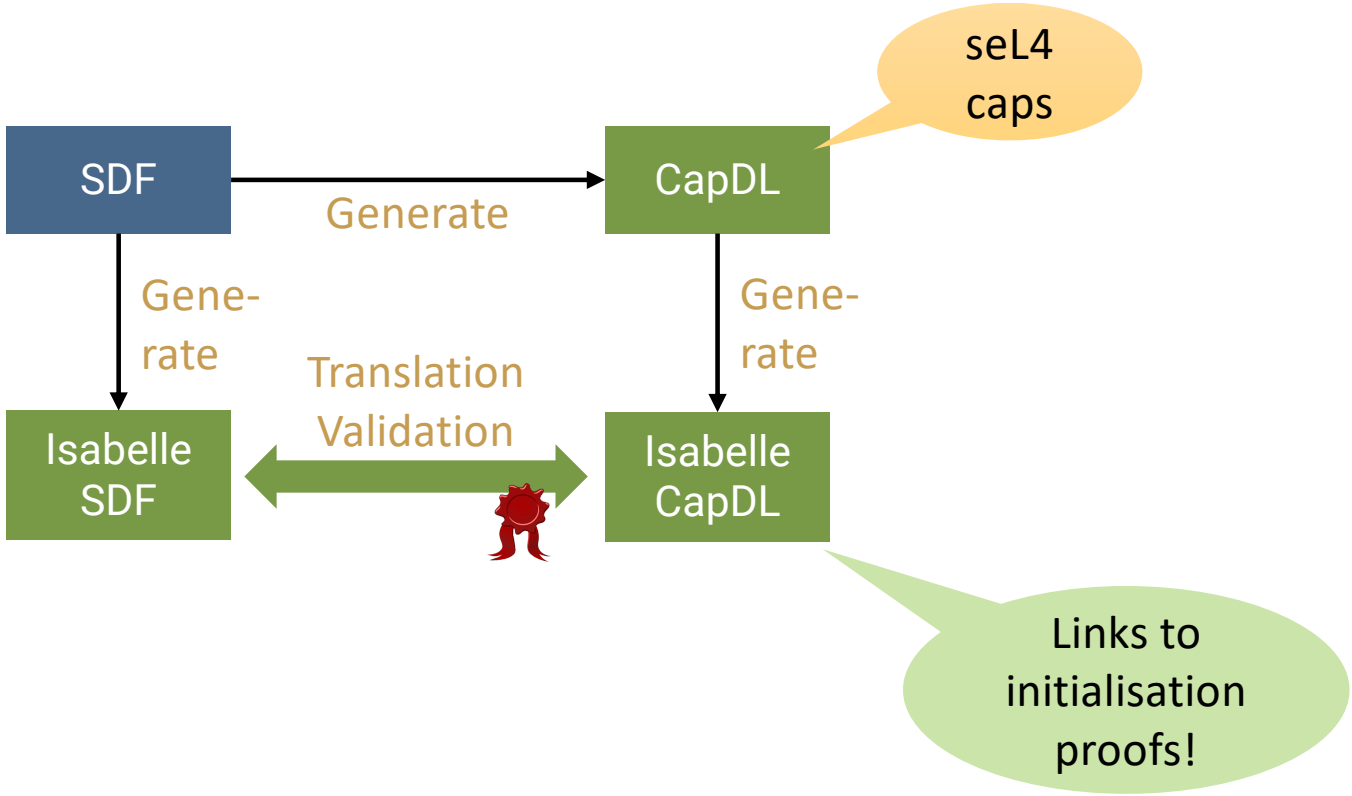
Verifying Microkit: libmicrokit



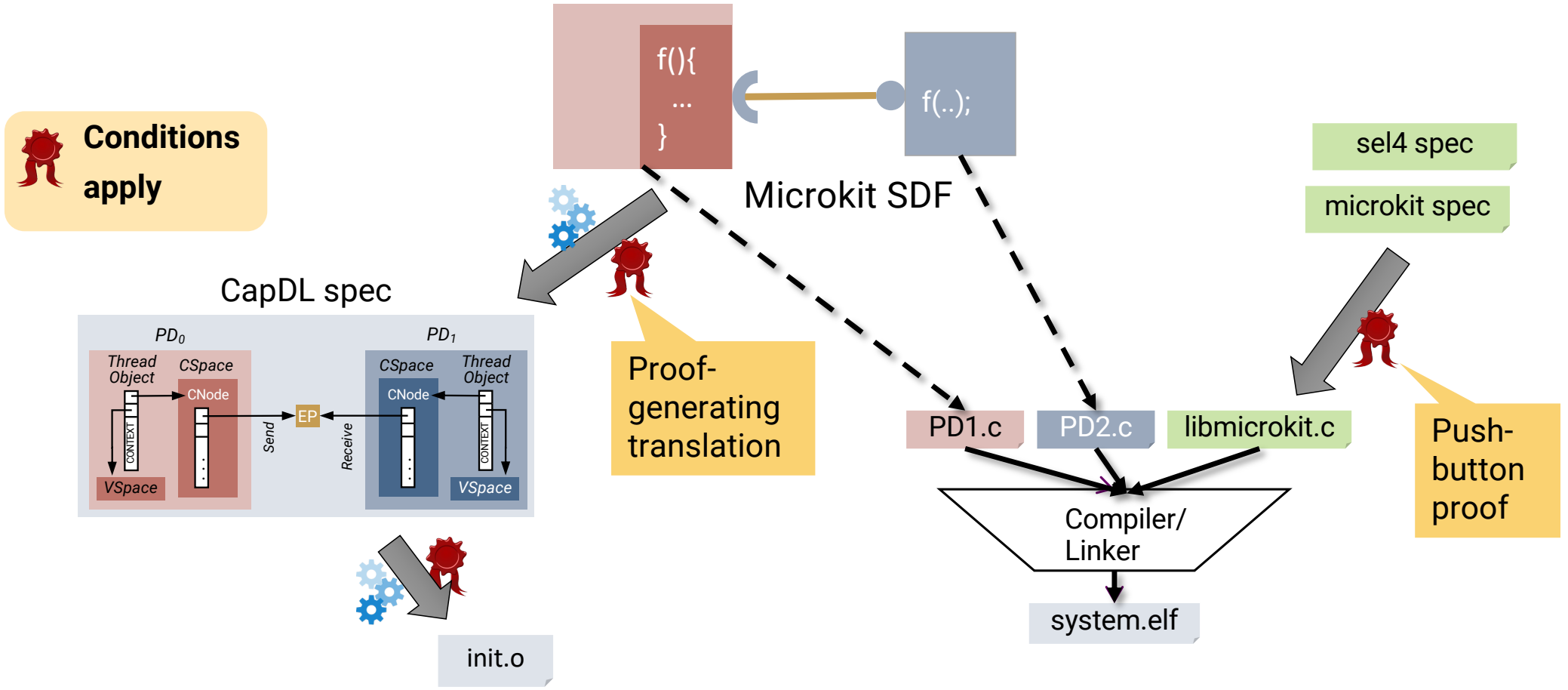
Take-away:

- Doable but cumbersome
- Annotations should be done in source code, not GraphLang

Verifying Microkit: System Initialisation

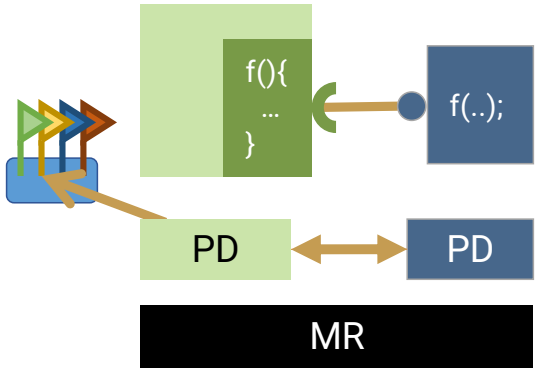


Microkit Verification in Context



Microkit Status

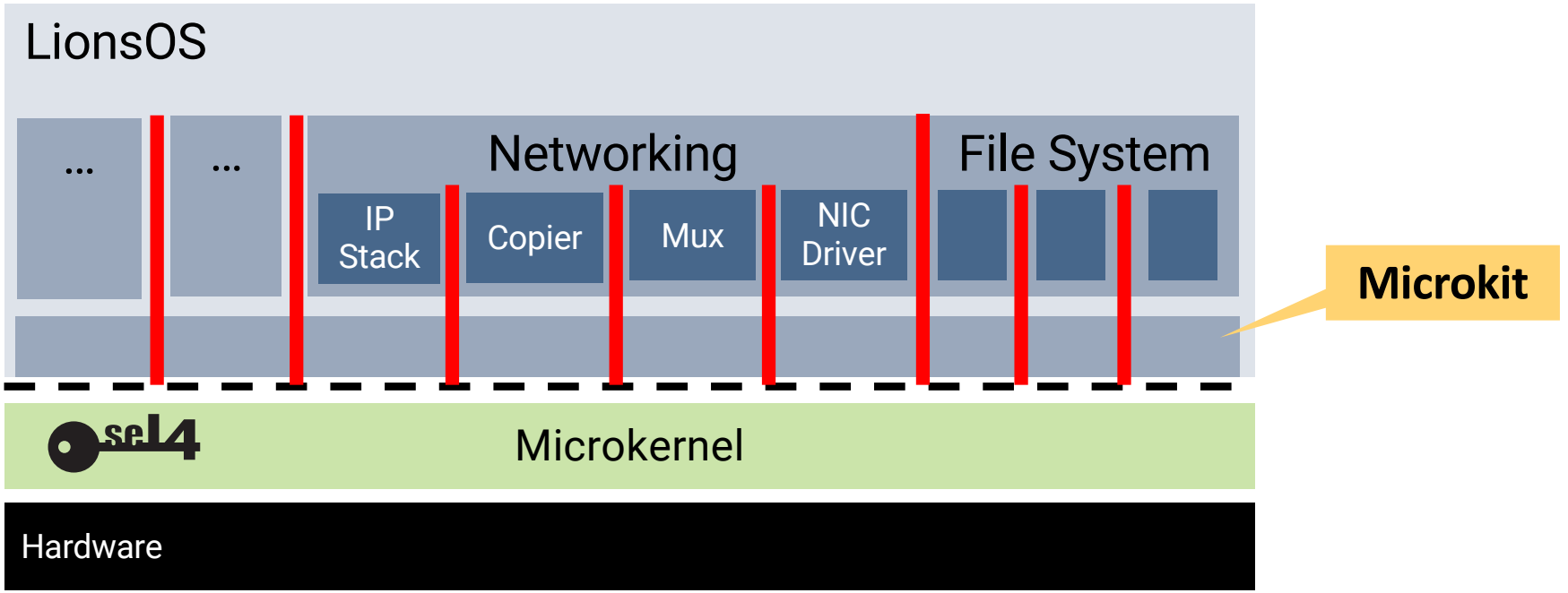
- Easy to use – non-experts productive within hours
- Supports AArch64, RV64 (x64 in progress)
- Verification presently for initial version & hacky, doing properly
- Limited dynamic features:
  - fault handlers
  - start/stop protection domains
  - empty protection domains (for late app loading)
- To come:
  - re-initialise protection domains
  - “template PDs” – discretionary access



LionsOS

Fast – secure – adaptable!

Lions OS: Highly Modular OS on Microkit



Lions OS: Principles

Least Privilege

Strict separation of concerns

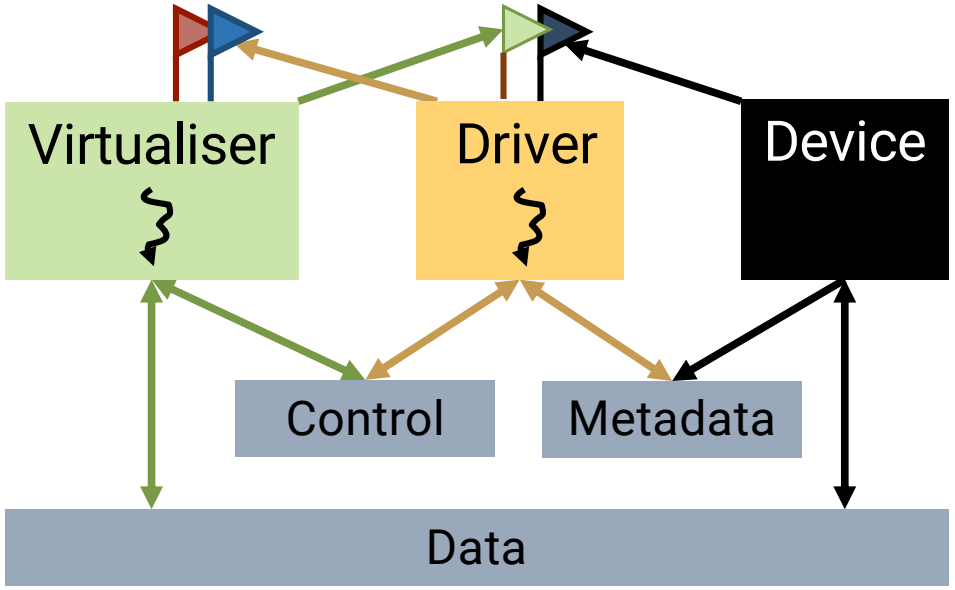
Overarching principle: KISS
"Keep it simple, stupid!"

Radical simplicity

Use-case–specific policies

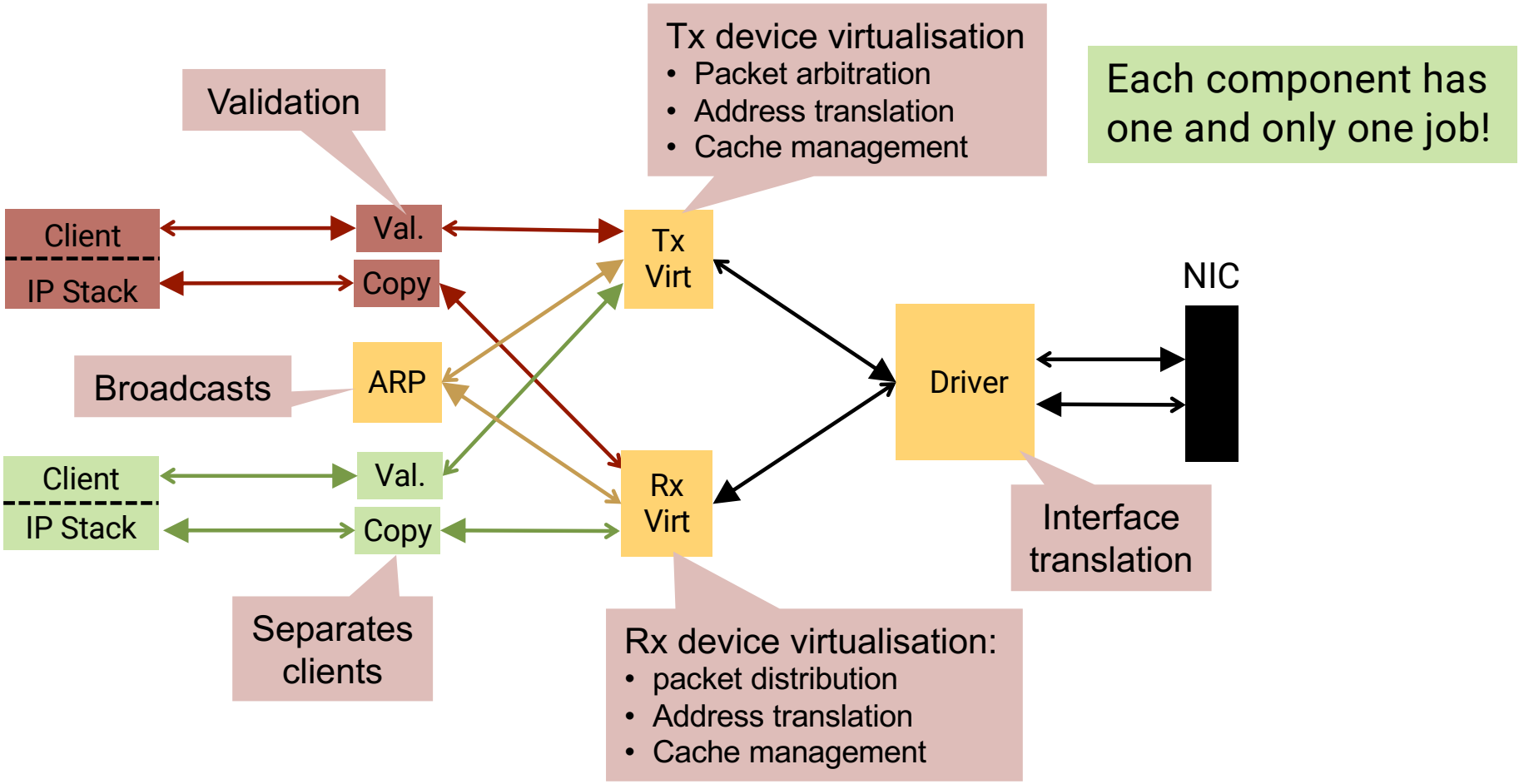
Design for verification

Least Privilege: Device Drivers



Driver does not need access to data region!

Strict Separation of Concerns: Networking



Radical Simplicity™

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

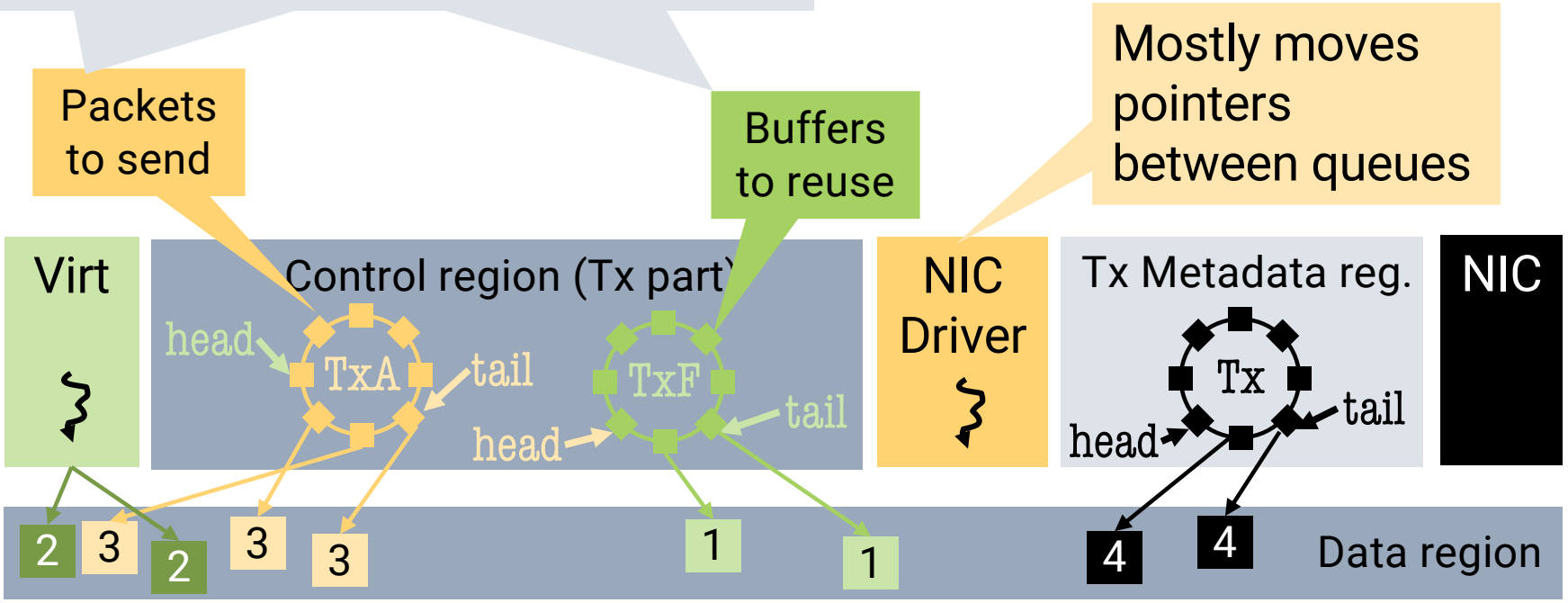
Simple programming model:
- strictly sequential code (Microkit)
- event-based (Microkit)
- single-producer, single-consumer queues
- location transparency
- …

Static **architecture**, mostly static resource management

Driver Programming Model

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

Driver model:
- Single-threaded
- Event-driven
- Simple!



Use-Case–Specific Policies

Source of massive complexity

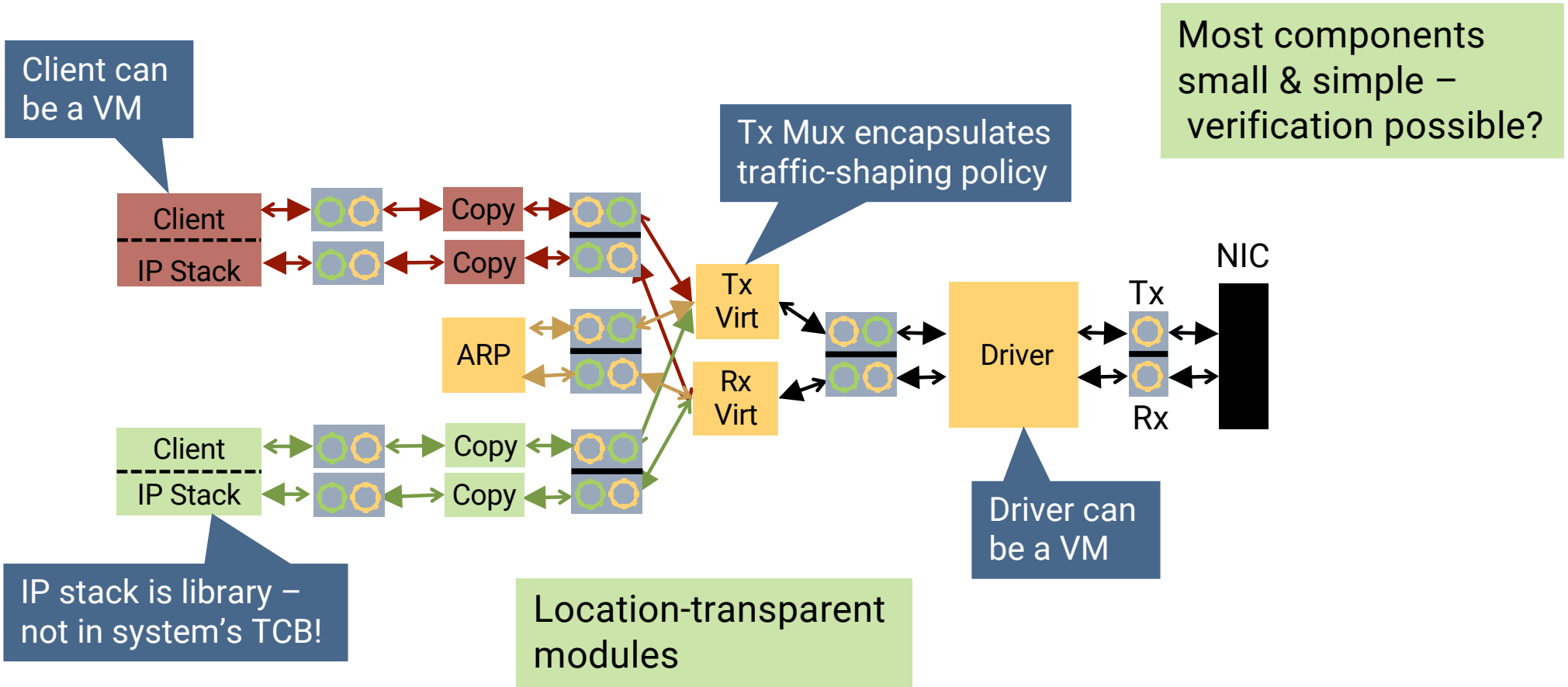
'80s model of computer use!

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

Lions-OS: Use-case diversity through policies that are:

- optimised for one specific use case
- simple, localised implementation
- easy to replace by swapping component

Networking System



Comparison to Linux (i.MX8)

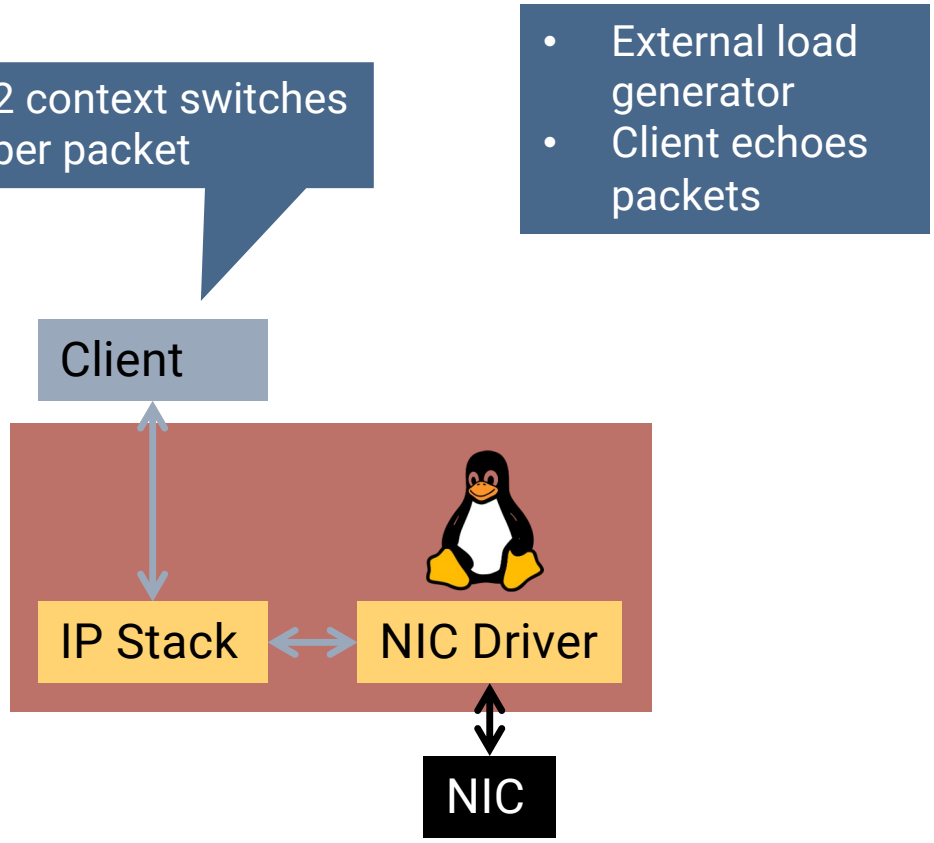
Linux:
- NW driver: 4k lines
- NW system total: 1M lines

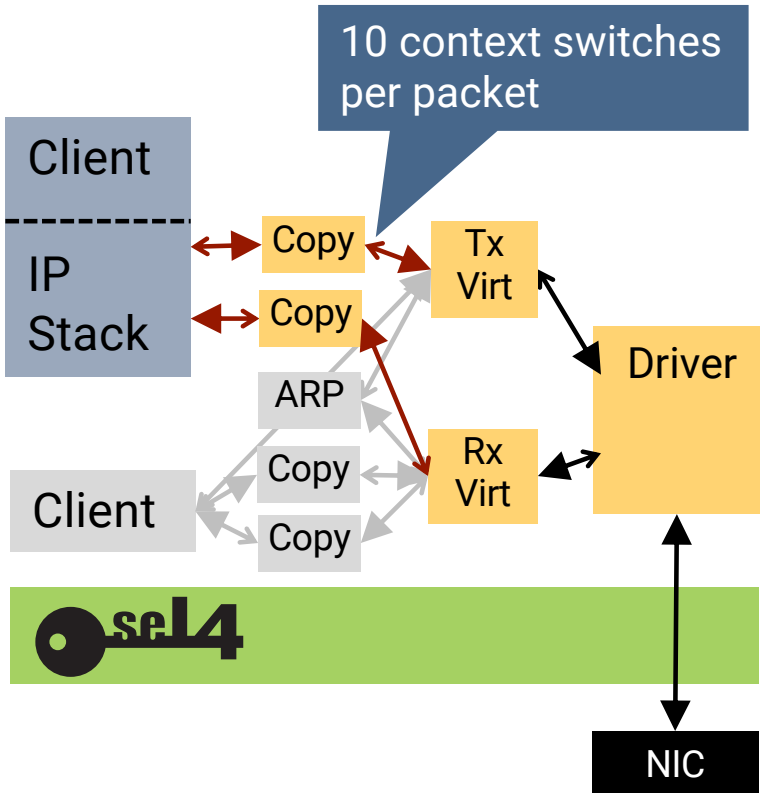
Performance?

Written by second-year student!

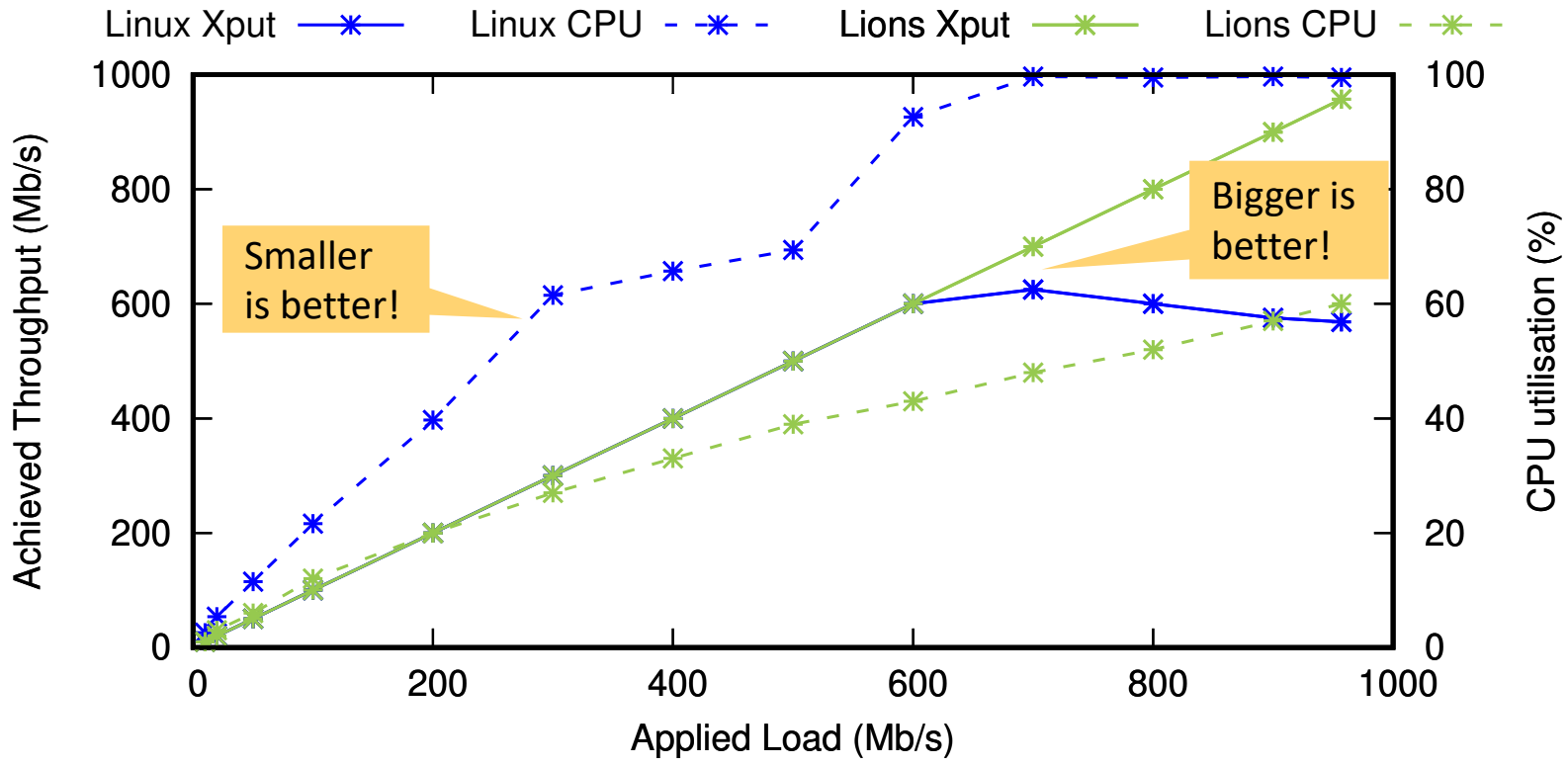
seL4 design:
- NW driver: 700 lines
- Virtualiser: 400 lines
- Copier: 200 lines
- IP stack: much simpler, client library
- shared NW system total < 2,000 lines

Evaluation Setup



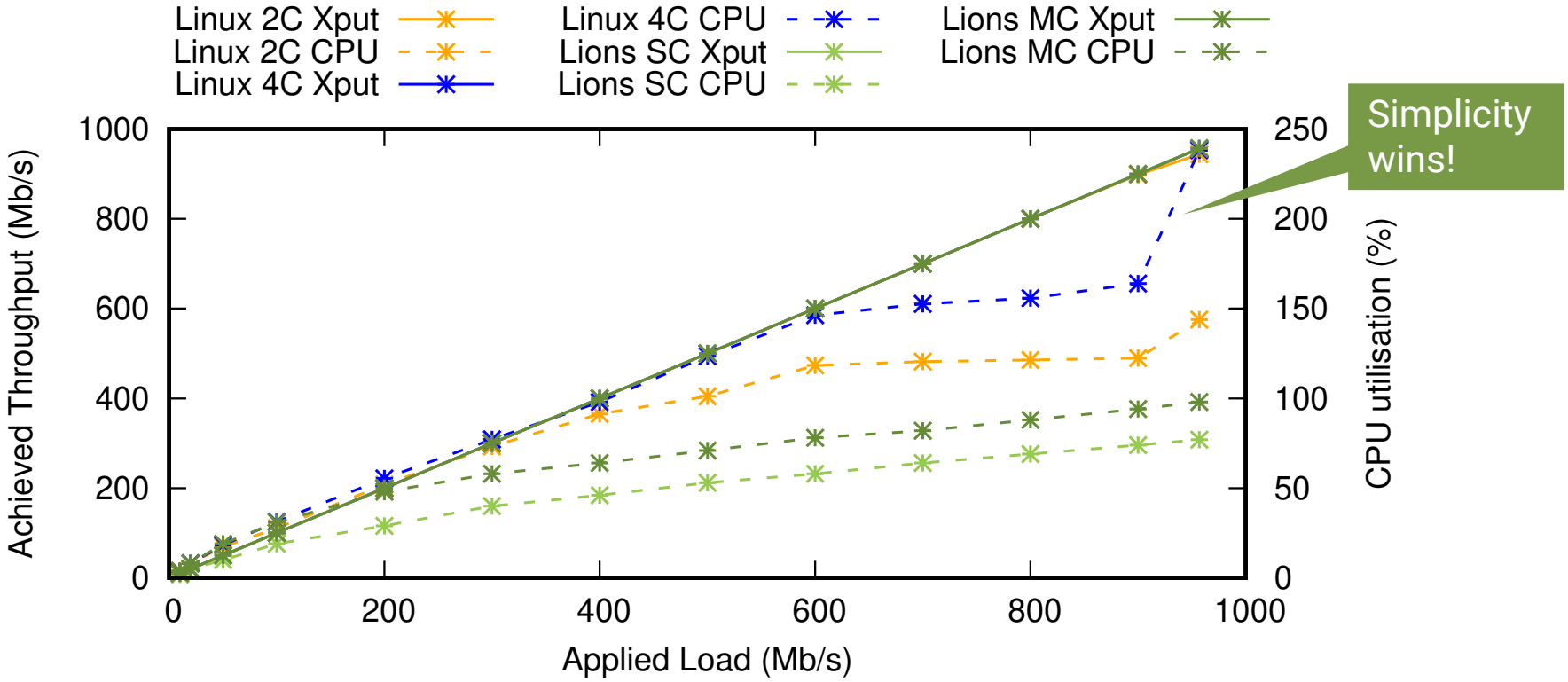


Performance: i.MX8M, 1Gb/s E/N, UDP



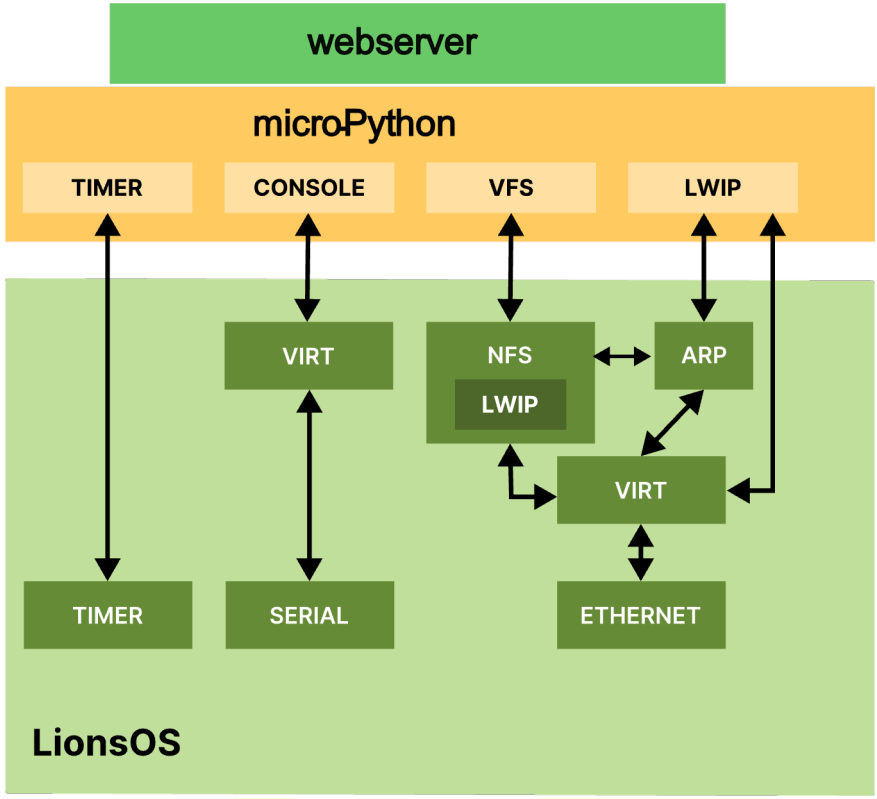
Single-core configuration

Performance: i.MX8M, 1Gb/s E/N, UDP

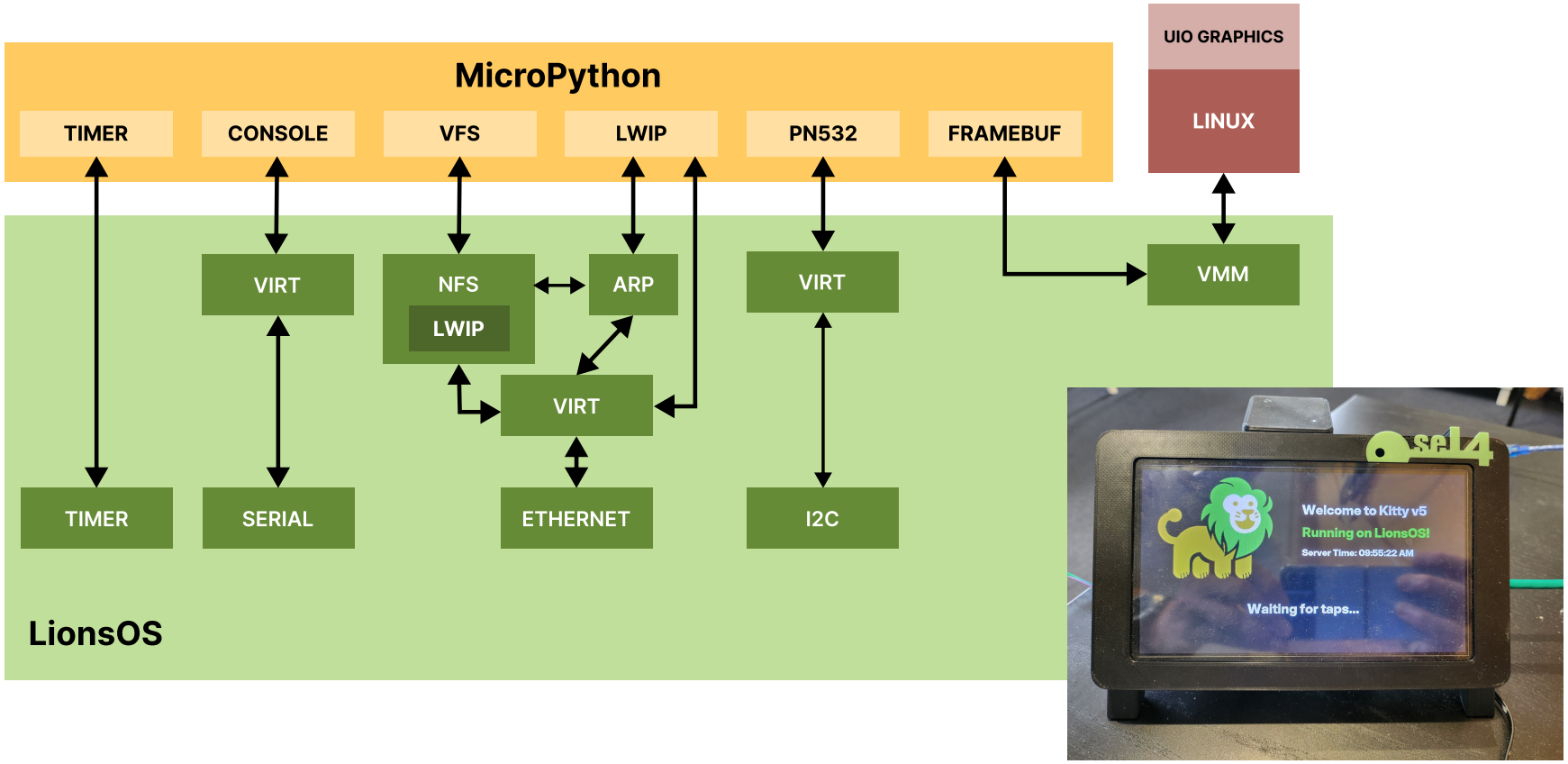


Multicore configuration

Underneath https://sel4.systems/



The TS “Kitty”



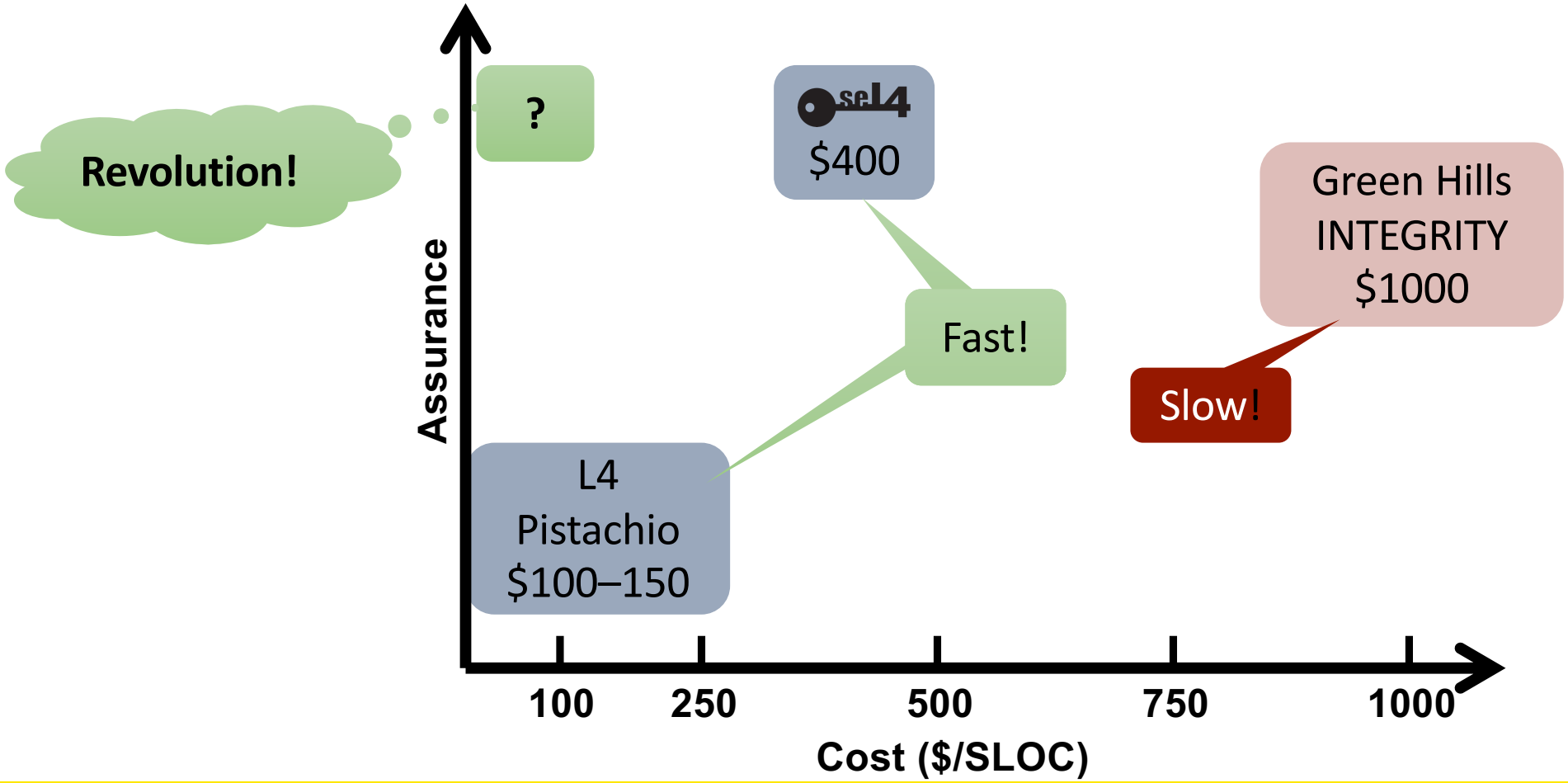
Lions OS Status

- Funding secured (DARPA, NIO, …)
- Networking, storage done
- Sound, I^2C, file system, hot-plugging close to merging
- Display supported by frame driver in Linux VM
- Deployed:
  - seL4.systems web server
  - Point-of-sale system
- Working on verification

Scaling Verification

PANCAKE
A Language for
Verified Systems Programming

Remember: Verification Cost in Context



Driver Dilemma

High seL4 verification costs partially due to C language

seL4 is one-off, justifies cost

Drivers are low-level, need C-like language

Better language would reduce cost

Drivers are commodity, must be cheap!

Lions OS

Idea:

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

- Well-defined semantics
- Memory-safe

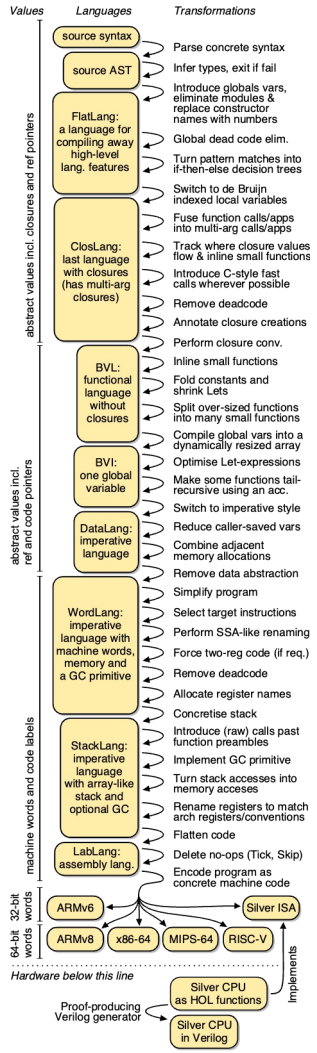
- Verified compiler
- de-compilation

CakeML: Verified Implementation of ML

- ✓ Mature functional language
- ✓ Large and active ecosystem of developers and users
- ✓ Code generation from abstract specs
- ❑ Managed ⇒ not suitable for systems code
- ✓ Used for verified application code

Re-use framework for new systems language: Pancake

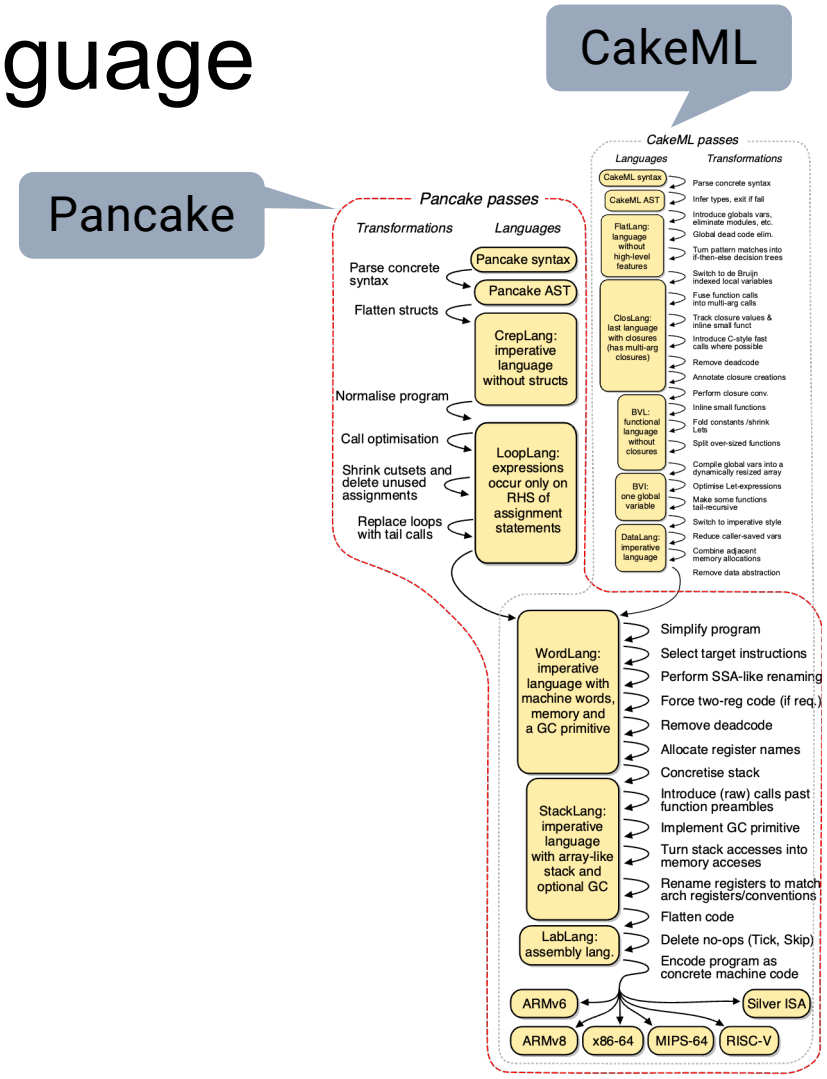
https://cakeml.org



Pancake: New Systems Language

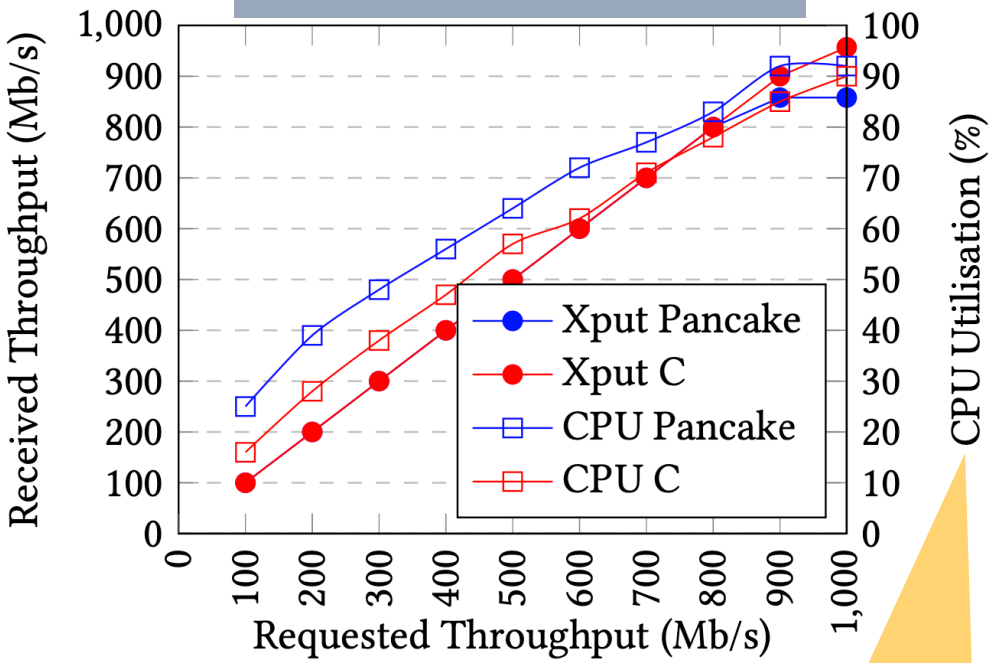
Approach:

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



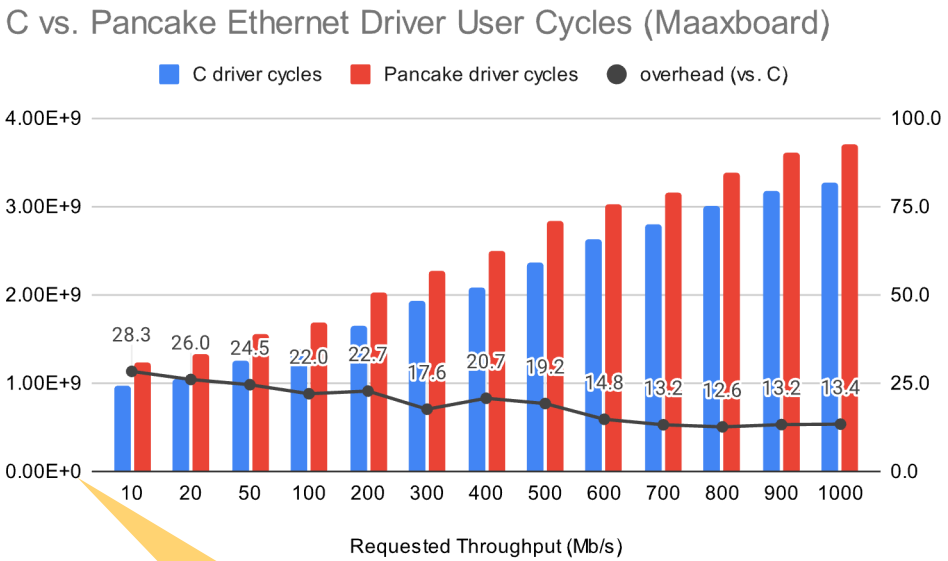
Pancake Performance: LionsOS Networking

Oct'23: Pancake Muxes



Overall utilisation –
Pancake overhead >100%!

Oct'24: Pancake Driver

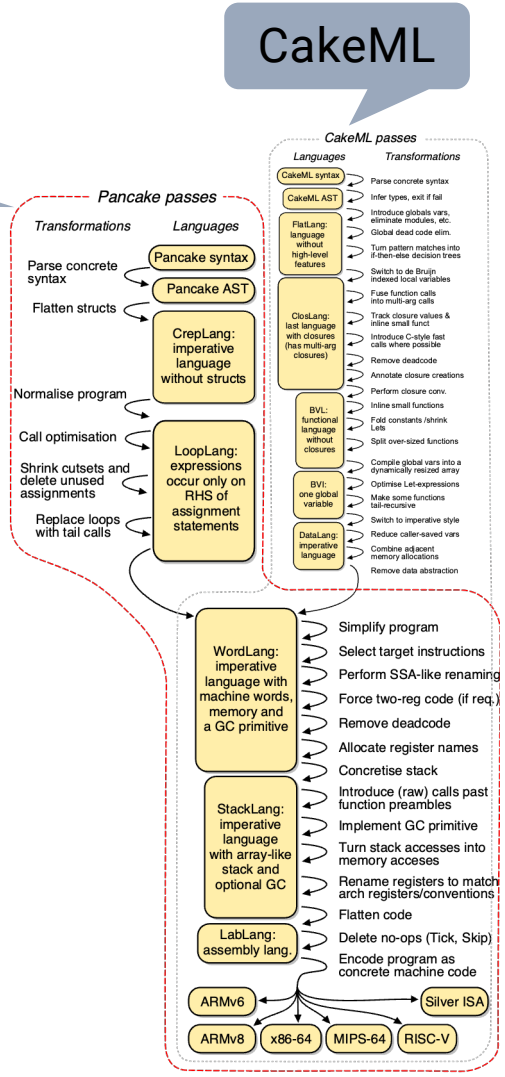


Cycles in driver only –
Pancake overhead 13–28%

Pancake: New Systems Language

Status:

- "Usable" rump language
  - sufficient for drivers
  - presently need C-escapes for cache management instructions
- **Verified compiler!**
- In progress:
  - performance improvements
  - automatic translation to SMT input language (Viper)
  - decompilation from Pancake to HOL
  - new semantics for non-terminating programs
  - efficient verification framework (Hoare logic)



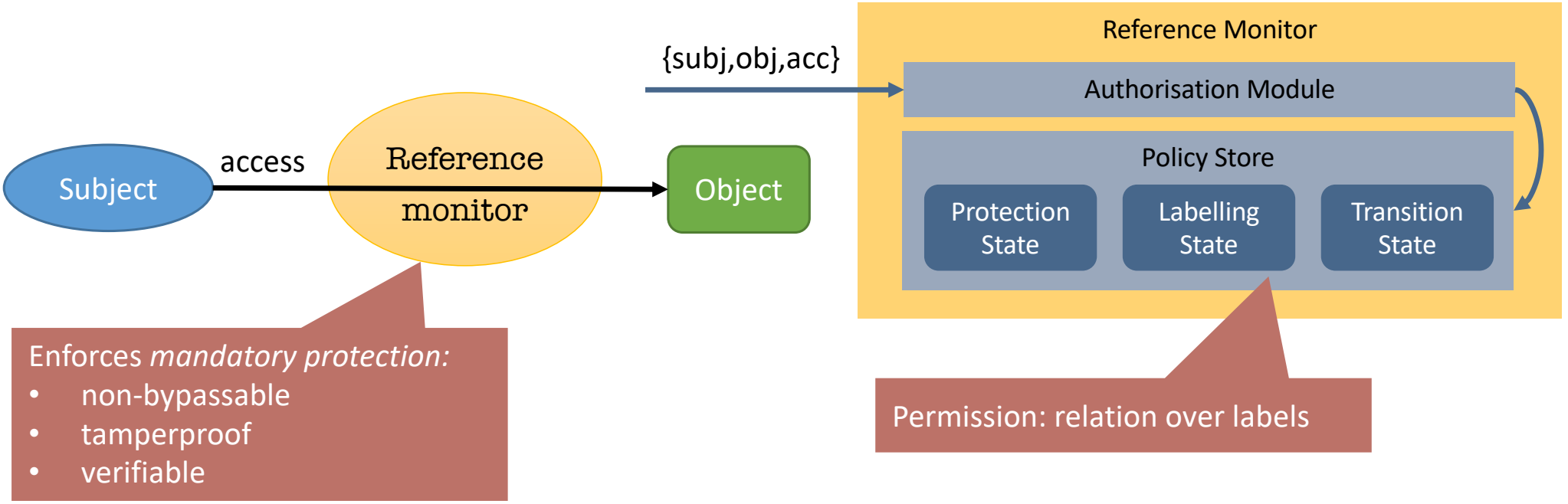
seL4-Related Research in TS

Secure Multi-Server OS

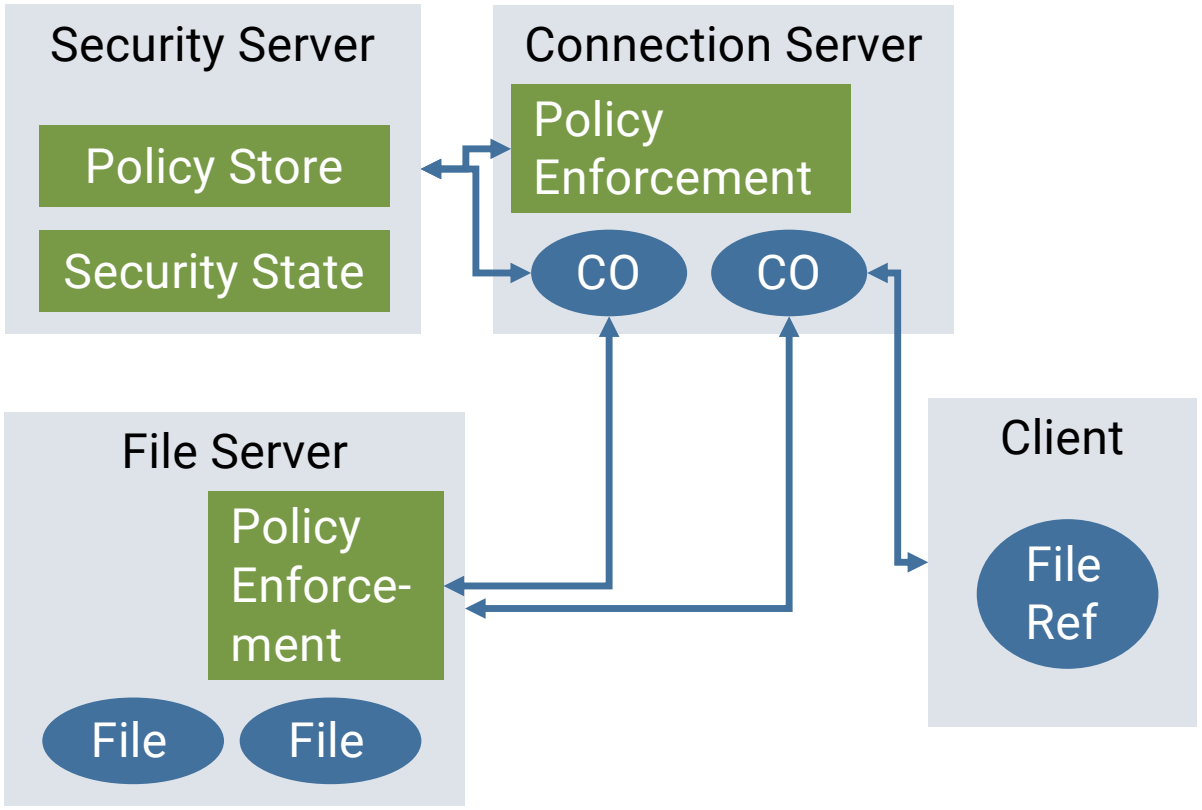
Recap: Secure Operating Systems

Secure OS: [Jaeger: OS Security]

Access enforcement satisfies the *reference monitor* concept



Secure, General-Purpose OS



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

Requires:

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

Real-World Use
Courtesy Boeing, DARPA