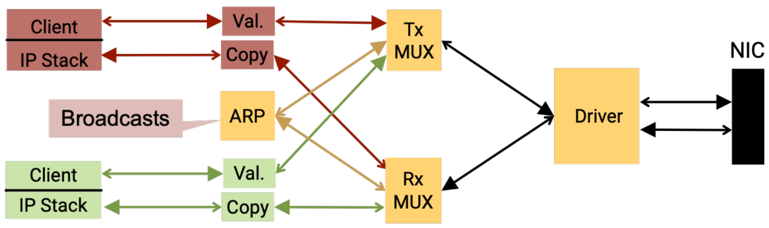
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2023 T3 Week 11

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
  - Verifying the seL4CP
  - Secure multi-server OS

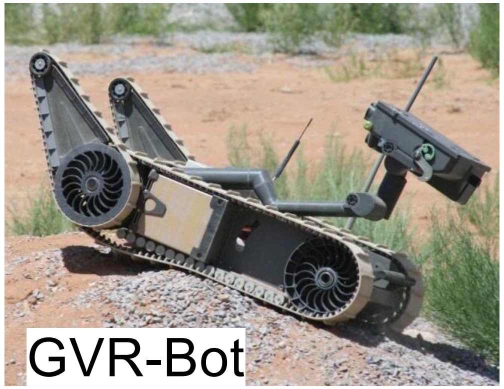
seL4 in the Real World

DARPA HACMS (2012–17)

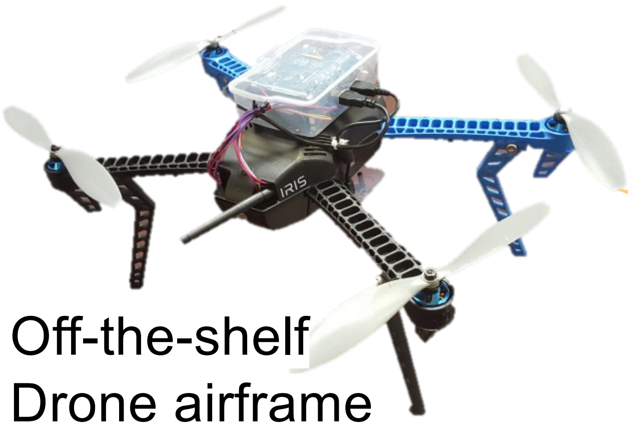
Unmanned Little Bird (ULB)

Retrofit existing system!

Autonomous trucks



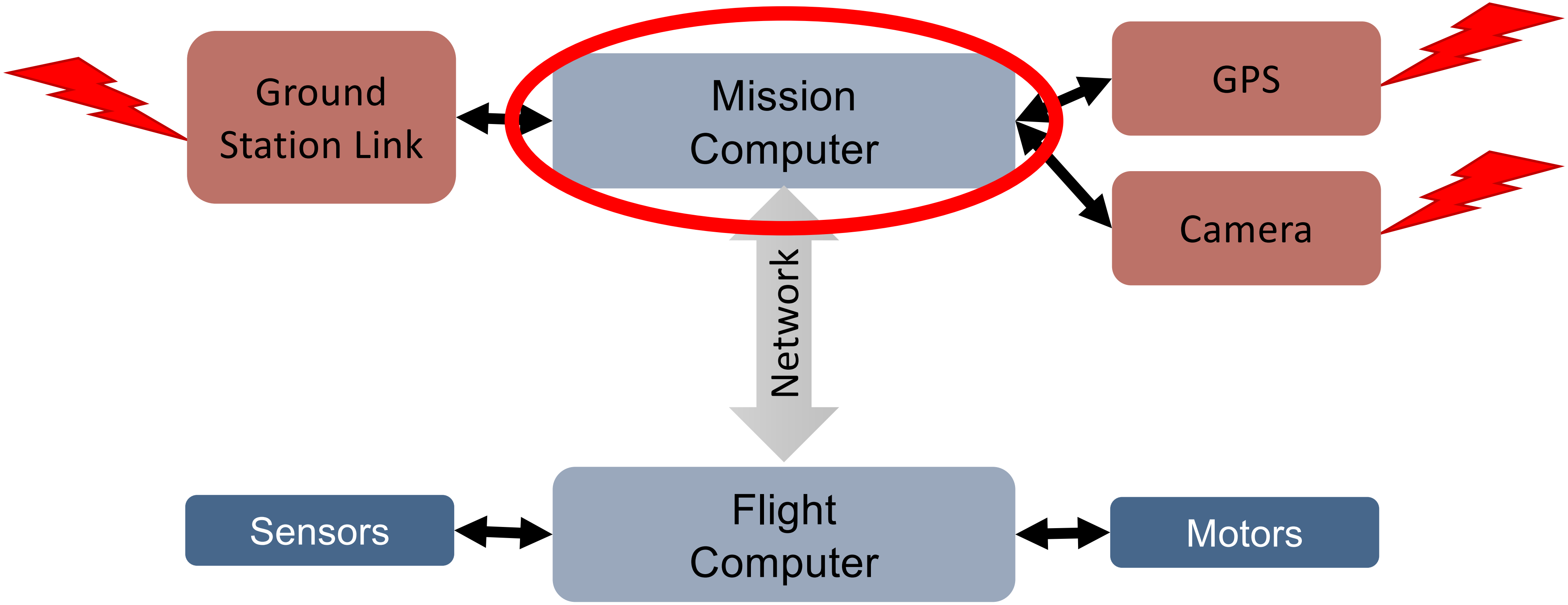
GVR-Bot



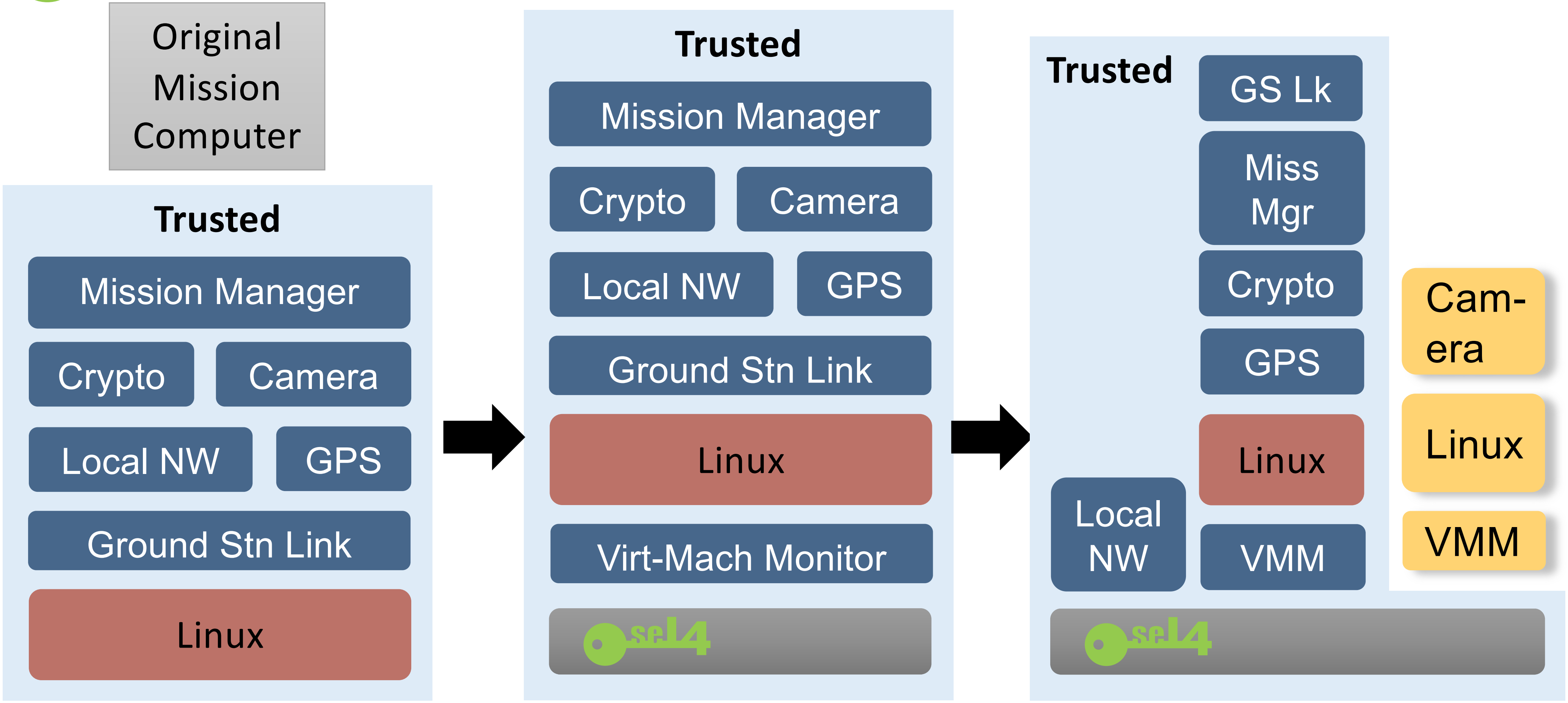
Off-the-shelf Drone airframe

Develop technology

ULB Architecture



Incremental Cyber Retrofit



Original Mission Computer

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Virt-Mach Monitor

seL4

Trusted

GS Lk

Miss Mgr

Crypto

GPS

Linux

Local NW

VMM

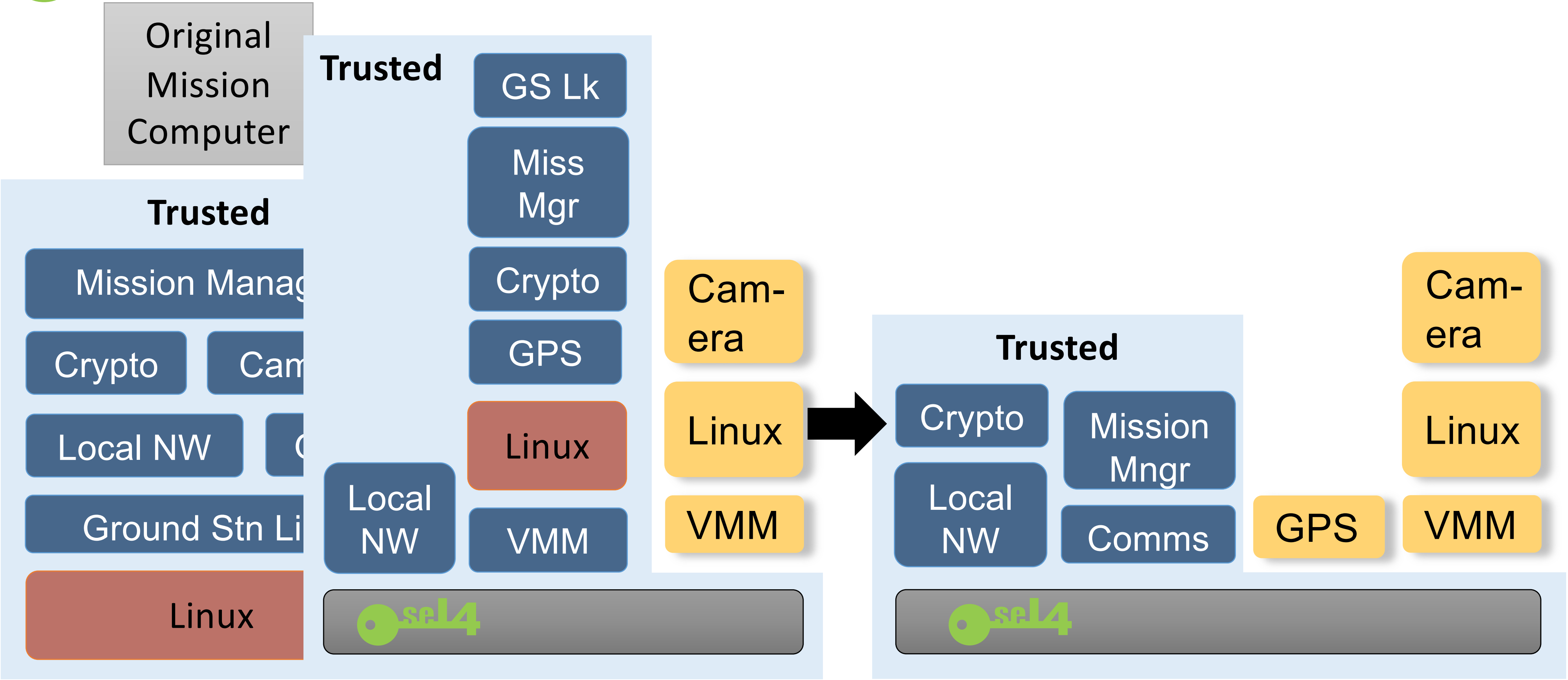
Cam-era

Linux

VMM

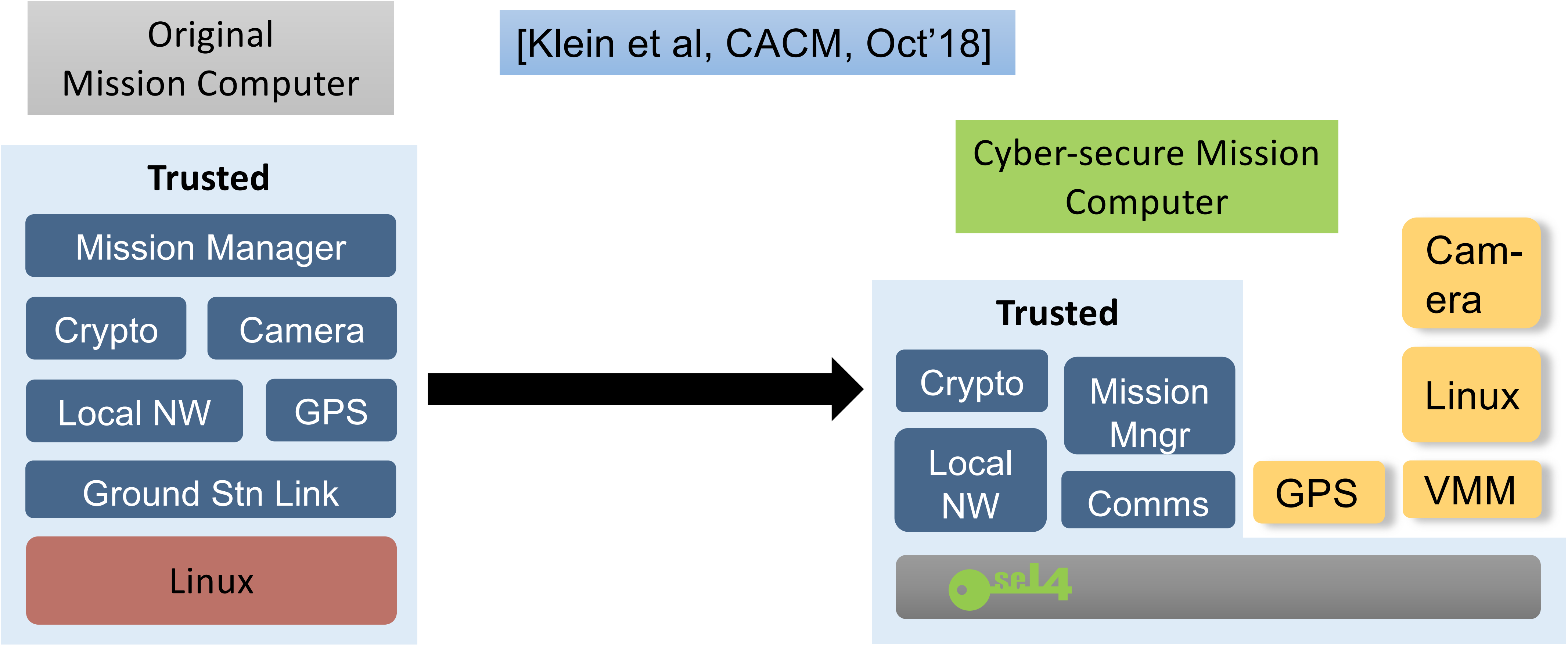
seL4

Incremental Cyber Retrofit



Incremental Cyber Retrofit

[Klein et al, CACM, Oct’18]



Original Mission Computer

Trusted

Mission Manager

Crypto

Camera

Local NW

GPS

Ground Stn Link

Linux

Cyber-secure Mission Computer

Trusted

Crypto

Mission Mngr

Local NW

Comms

GPS

Cam-era

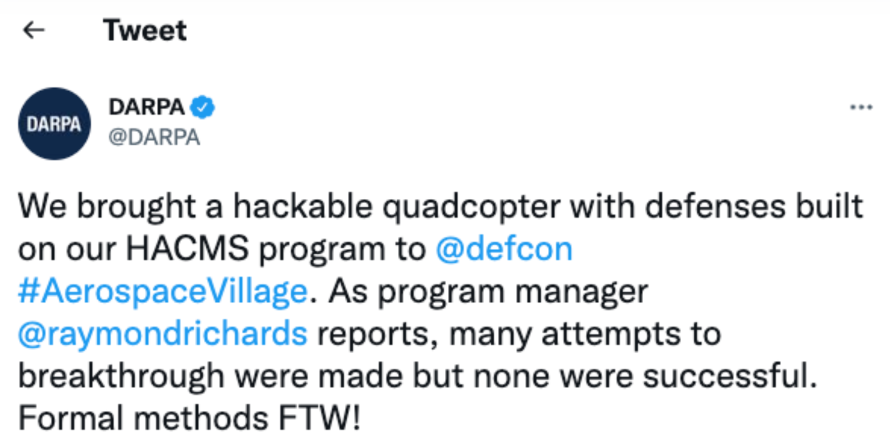
Linux

VMM

seL4

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

The seL4 Foundation

Premium Members

地平线 Horizon Robotics

jumptrading

NIO

UNSW SYDNEY

General Members

Cog™

DORNERWORKS

galois

Google

HENSOLDT Detect and Protect.

KRY10

LOTUS

penten

Proofcraft

Raytheon Technologies

SPACEMIT 进迭时空

TII Technology Innovation Institute

xcalibyte

Associate Members

THE AUTOWARE FOUNDATION

ETH zürich

KANSAS STATE UNIVERSITY

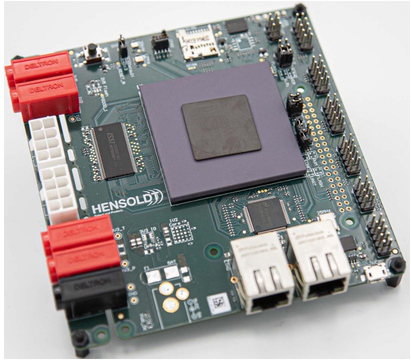
in association with National Cyber Security Centre

RISC-V

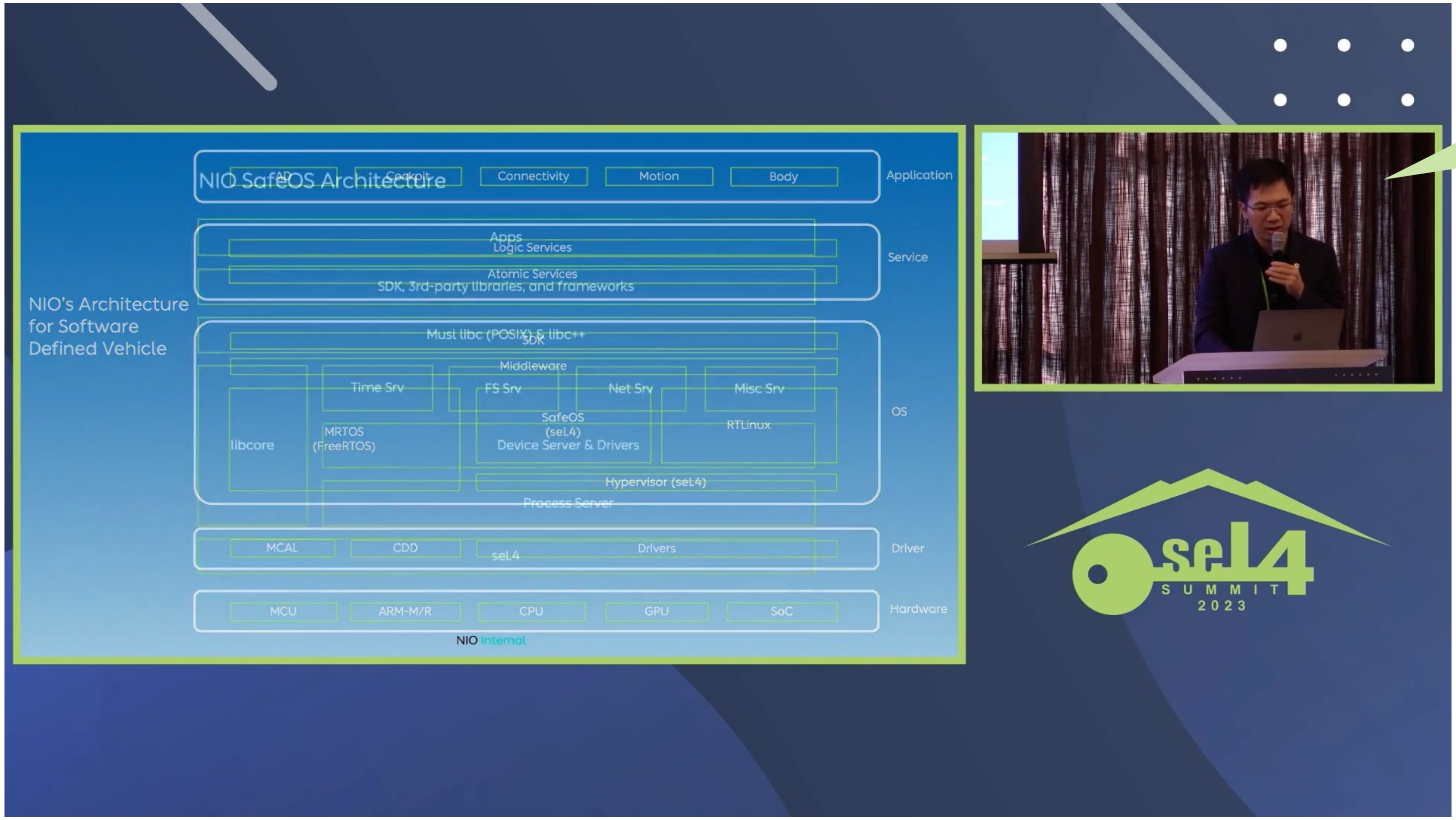
TUM

KU INSTITUTE FOR INFORMATION SCIENCES The University of Kansas

seL4 in Products



5th seL4 Summit, 2023-09-20

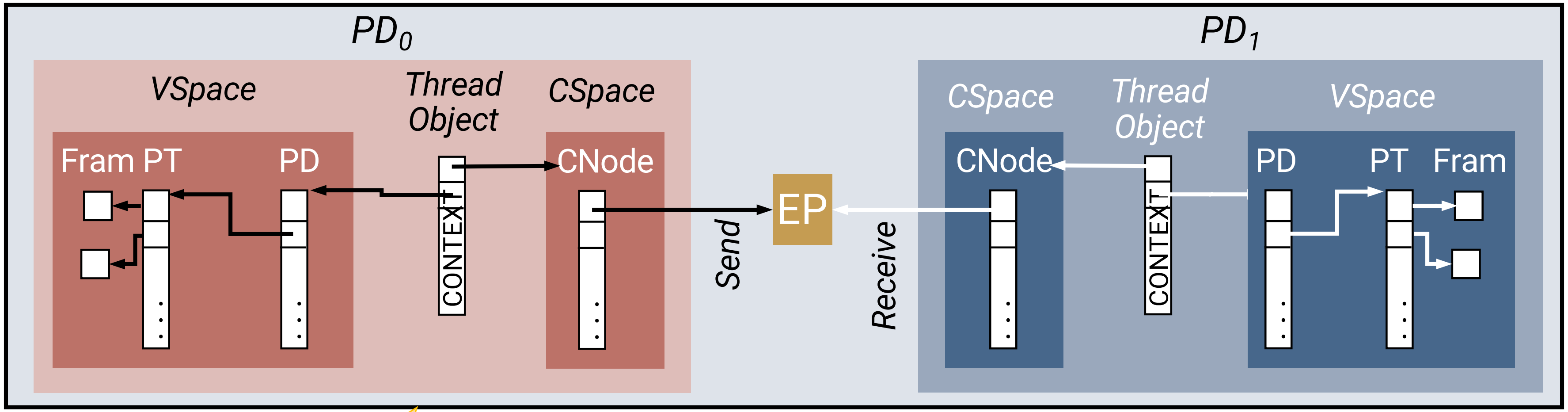


Qiyan Wang,
Global VP Engineering,
Digital Systems
Electric car maker NIO

"this OS, based on seL4, will in our mass-production cars next year"

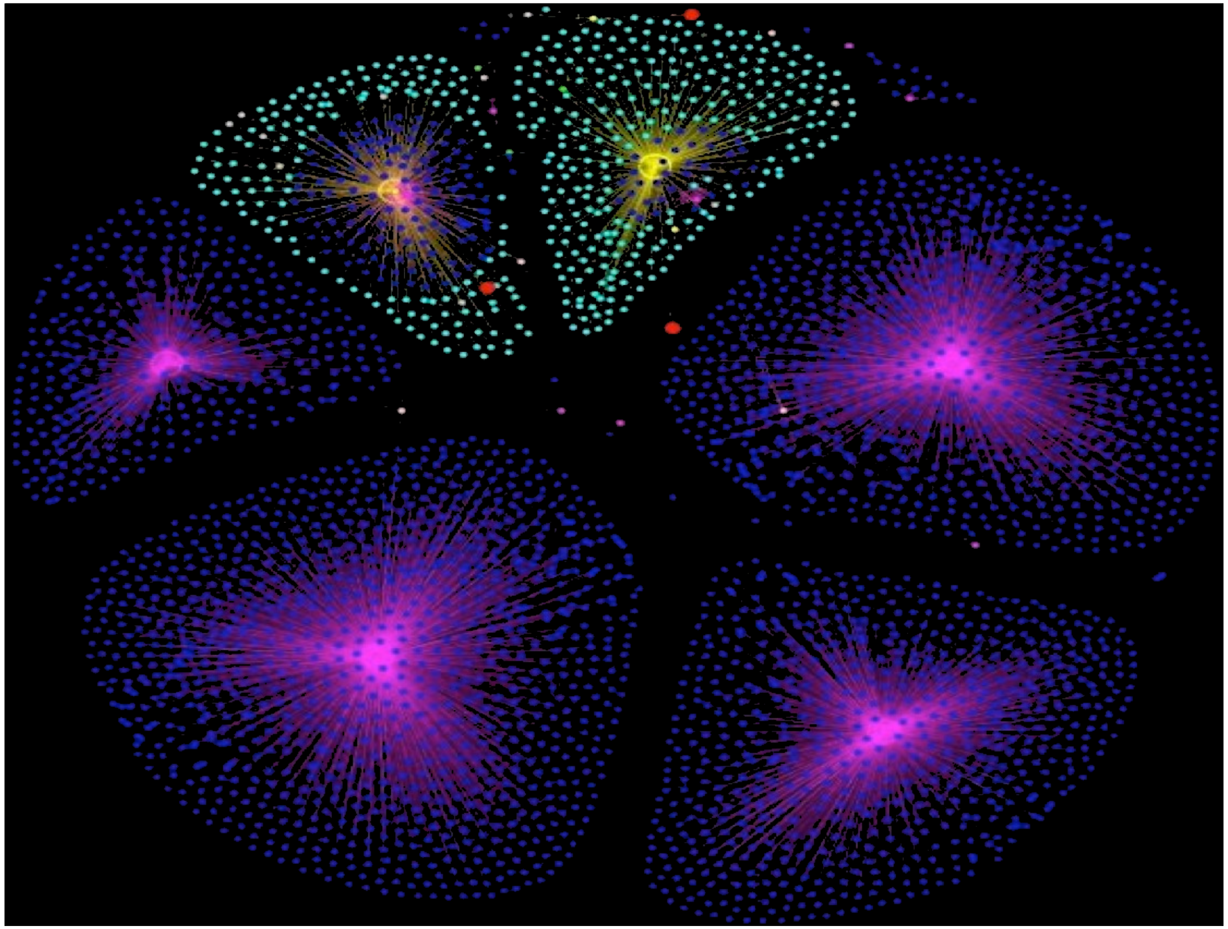
Usability 1: Microkit

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

- Doing it right requires good abstractions
- abstractions introduce policy limit application space

Result:

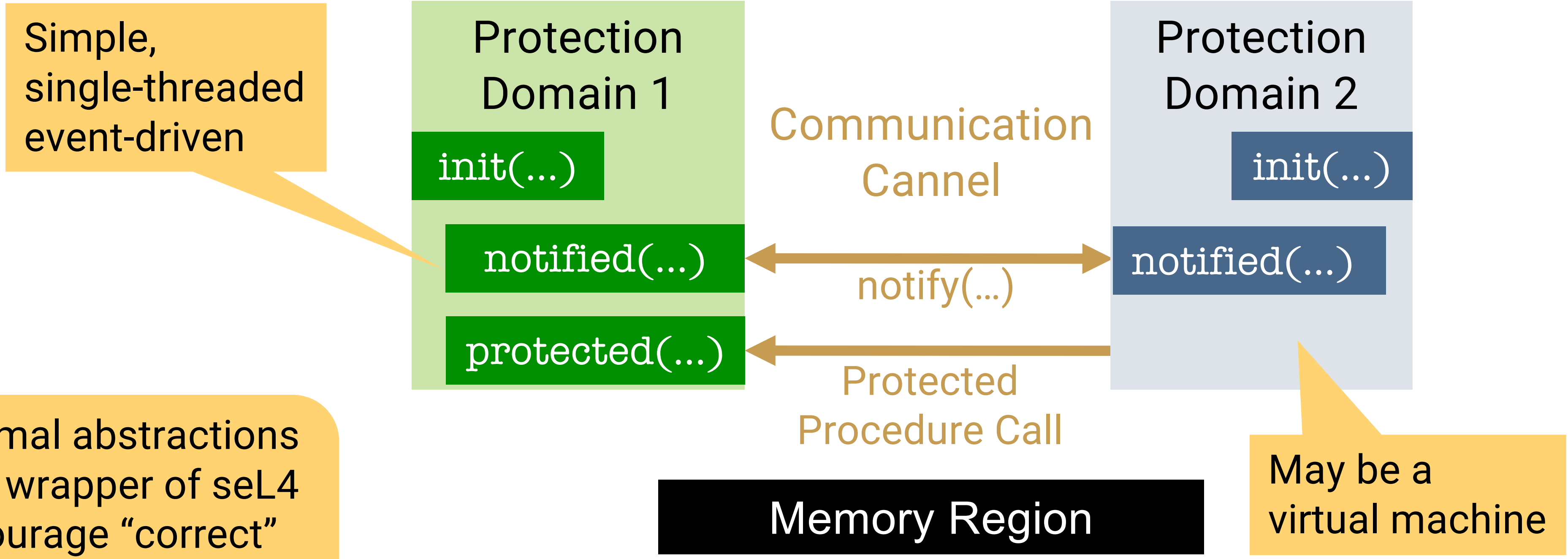
- everyone builds their own
- proliferation of bad designs
- huge waste of effort

Step 1: seL4 Microkit

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

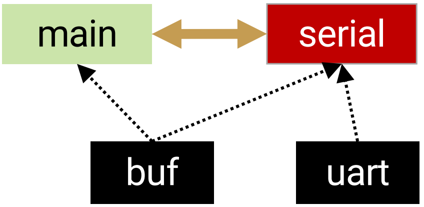
```
for (;;) {
    if (have_reply) {
        tag = seL4_ReplyRecv(INPUT_CAP, reply_tag, &badge, REPLY_CAP);
    } else if (have_signal) {
        tag = seL4_NBSendRecv(signal, signal_msg, INPUT_CAP, &badge, REPLY_CAP);
        have_signal = false;
    } else {
        tag = seL4_Recv(INPUT_CAP, &badge, REPLY_CAP);
    }
    event_handle(badge, &have_reply, &reply_tag, &notified);
}
```

libmicrokit: Invoking user code

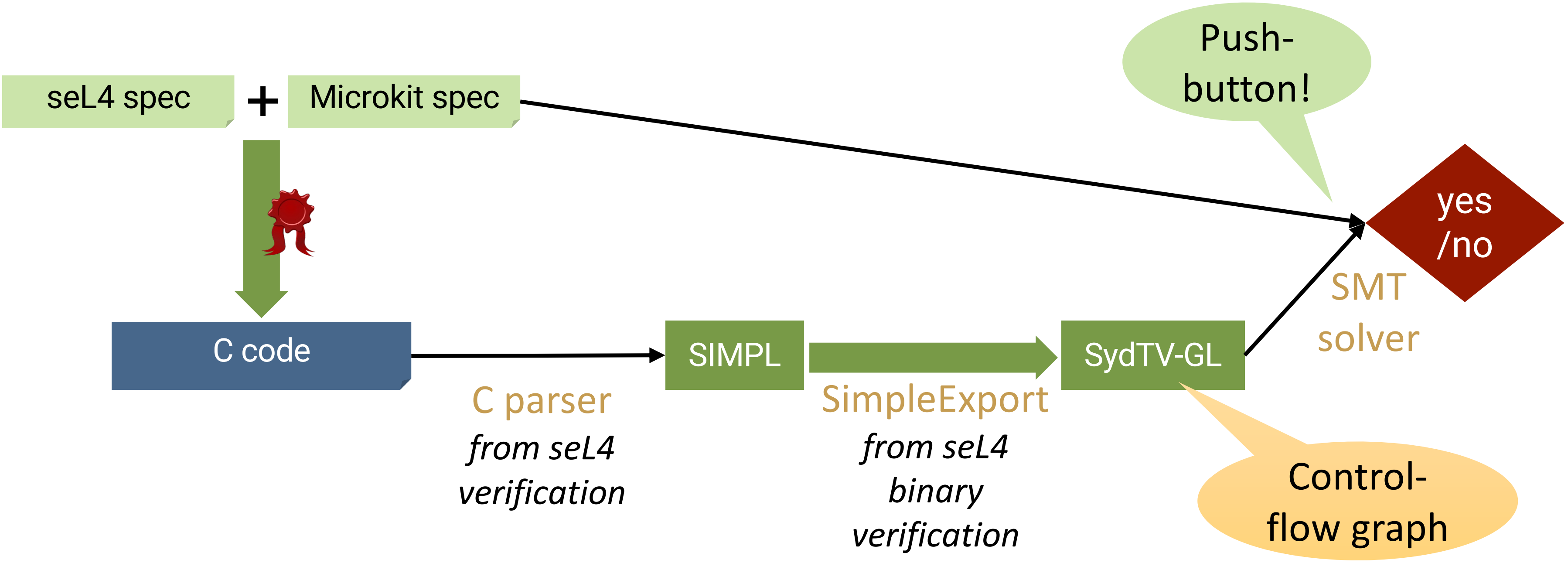
```
event_handle(badge, &have_reply, &reply_tag, &notified) {
    if ((have_reply) = badge >> 63) {
        reply_tag = protected(badge & 0x3f, tag);
    } else {
      unsigned int idx = 0;
      do  {
        if (badge & 1) {
          notified(idx);
        }
        badge >>= 1; idx++;
      } while (badge != 0);
    }
  }
```

Microkit System Description File (SDF)

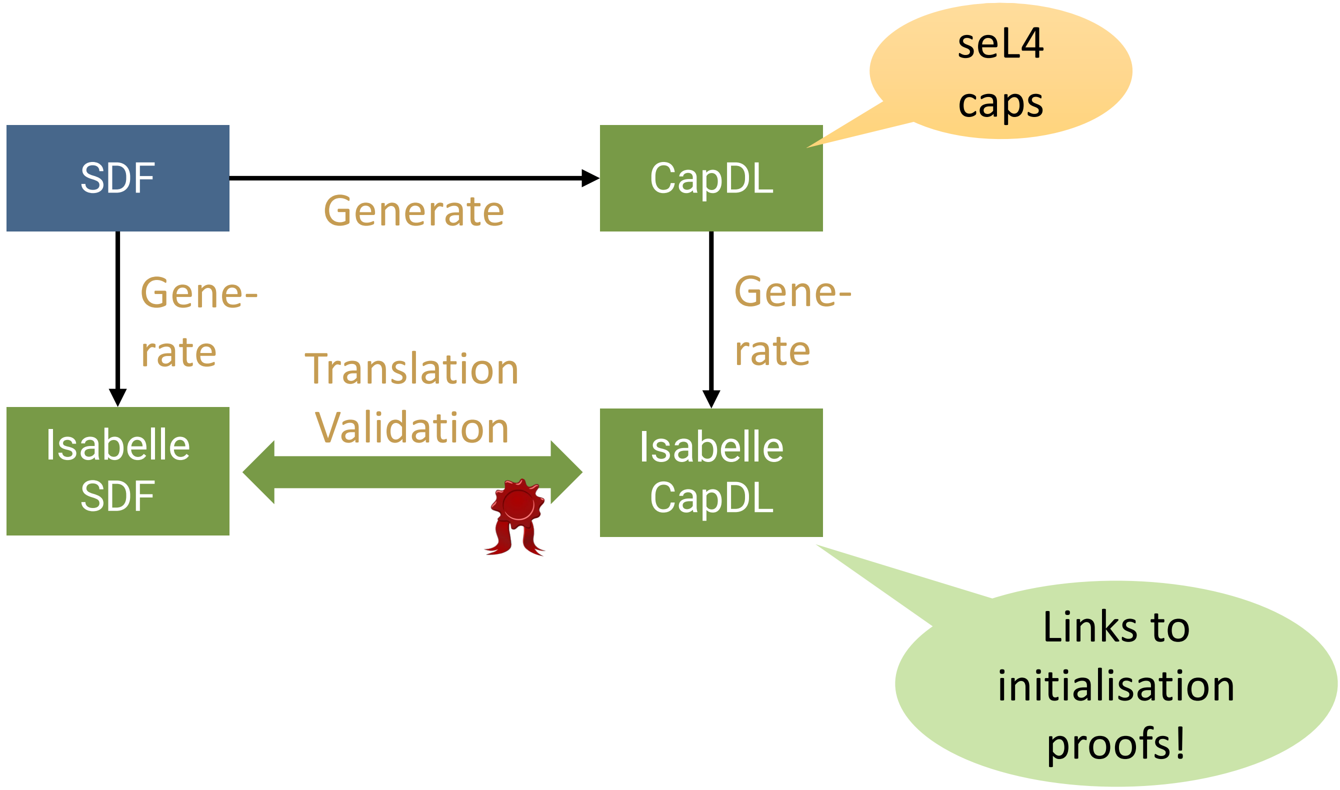
```
<system>
    <memory_region name="uart" size="0x1000" phys_addr="0x9000000" />
    <memory_region name="buf" size="0x1000" />
    <protection_domain name="serial” priority="250”>
        <irq irq="33" id="0" />
        <program_image path="serial_server.elf" />
        <map mr="uart" vaddr="0x4000000" perms="rw" cached="false” … />
        <map mr="buf" vaddr="0x4001000" perms="rw" setvar_vaddr="input" />
    </protection_domain>
    <protection_domain name=”main">
        <program_image path=”main.elf" />
    </protection_domain>
    <channel>
        <end pd="serial" id="1" />
        <end pd="client" id="0" />
     </channel>
</system>
```



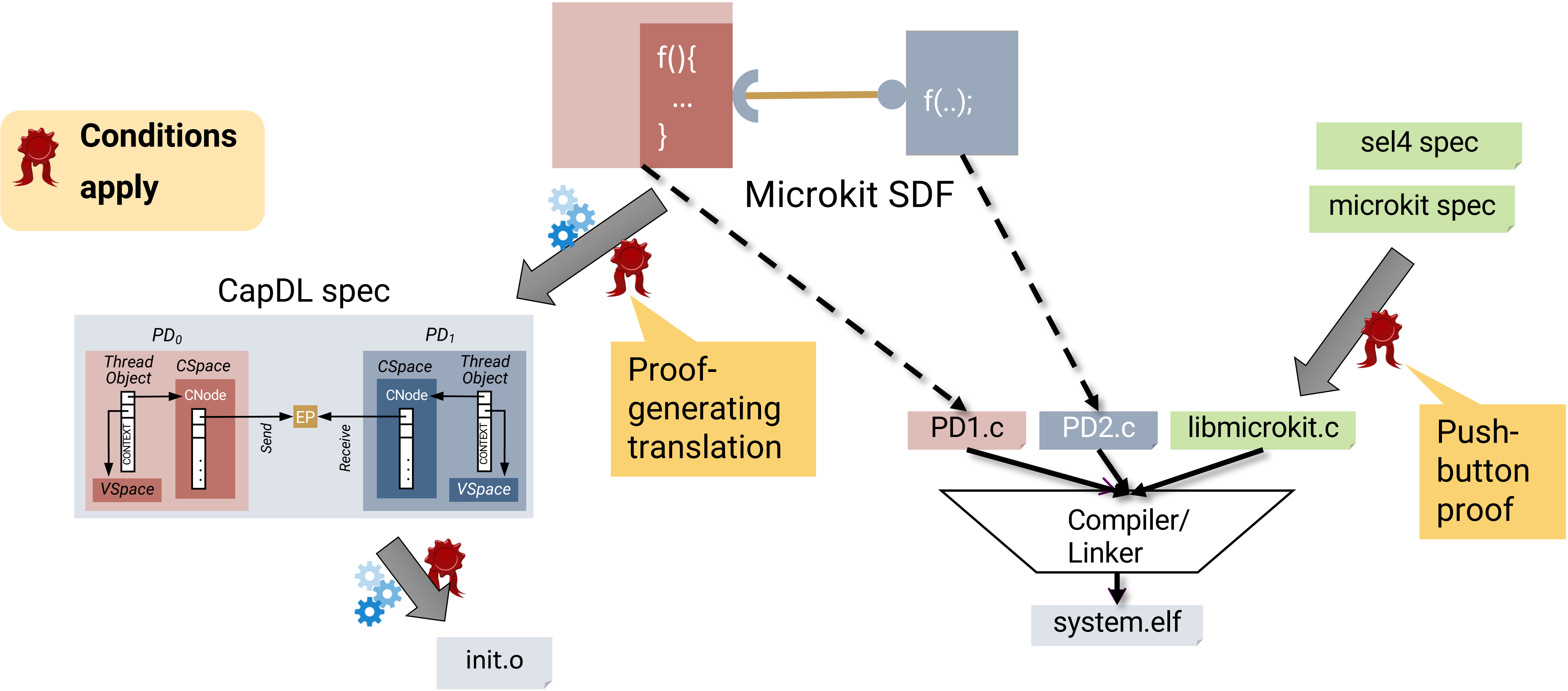
Verifying the Microkit: libmicrokit



Verifying the Microkit: System Initialisation

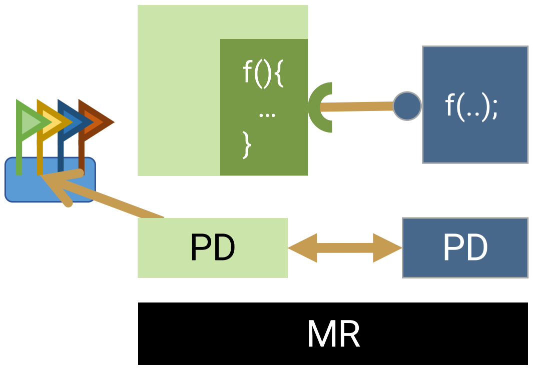


Microkit Verification in Context



Microkit Status (evolving quickly…)

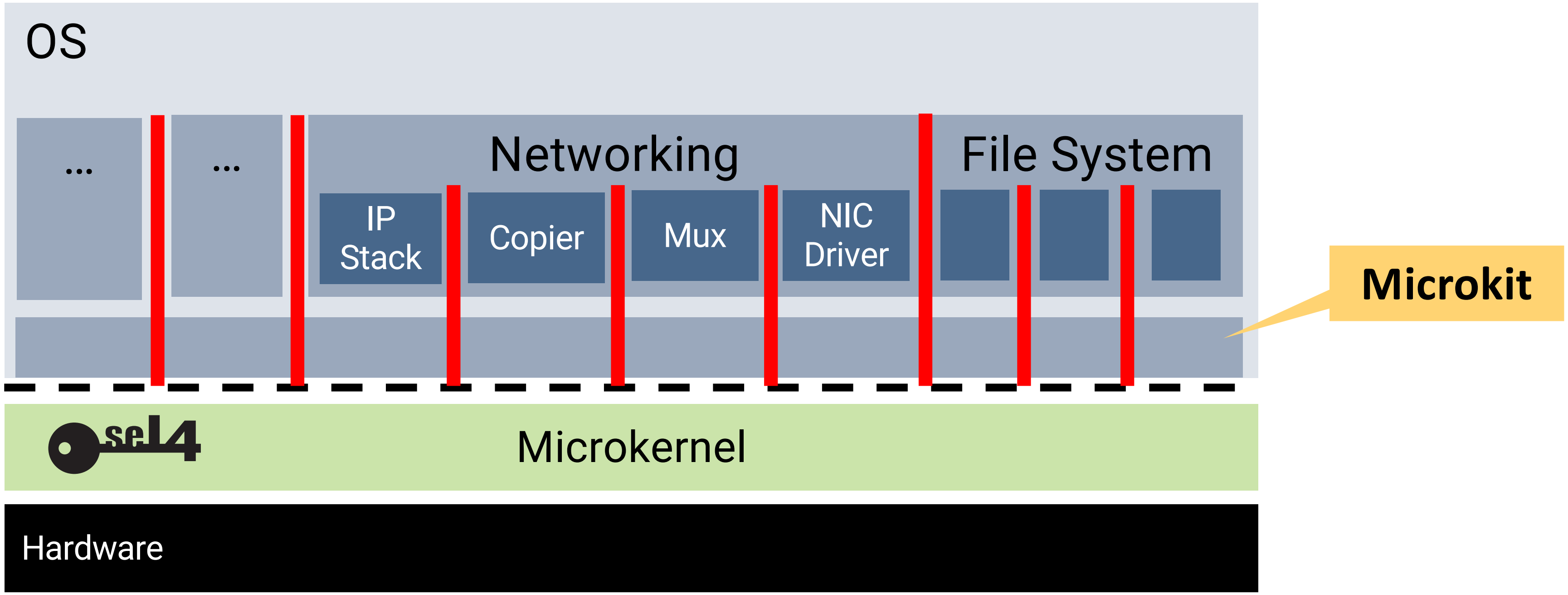
- Officially adopted by seL4 Foundation in Sep’23
- Supports AArch64, RV64 (x64 in progress)
- Verification presently for initial version only, catching up
- Dynamic features prototype:
  - fault handlers
  - start/stop protection domains
  - re-initialise protection domains
  - empty protection domains (for late app loading)



Usability 2: Lions OS

Current research

Lions OS: Highly Modular OS on Microkit



Lions OS: Aims

Fast:
Best-performing microkernel-based OS ever

Secure:
Most secure real-world OS ever

Adaptable:
Suitable for a wide range of cyberphysical / IoT / embedded systems

Lions OS: Principles

Least Privilege

Strict separation of concerns

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

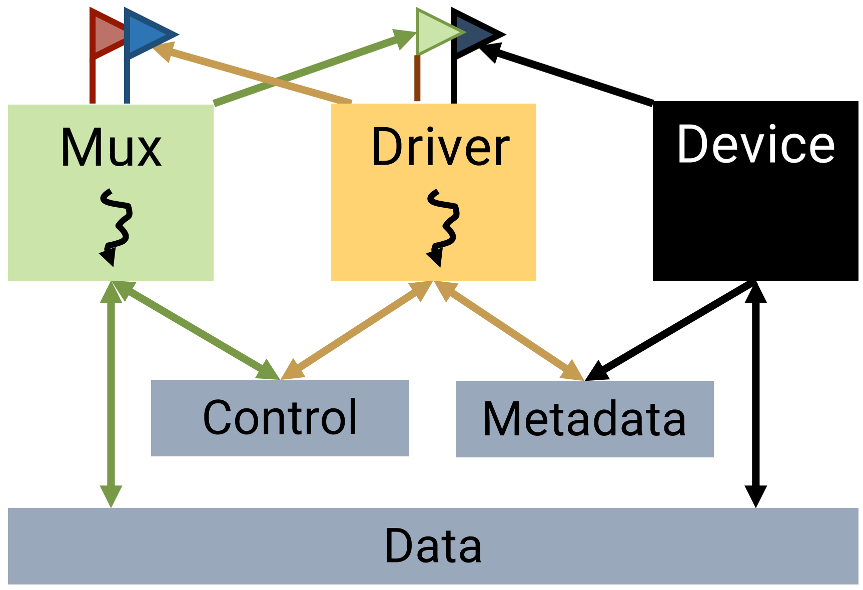
Radical simplicity

Use-case–specific policies

Design for verification

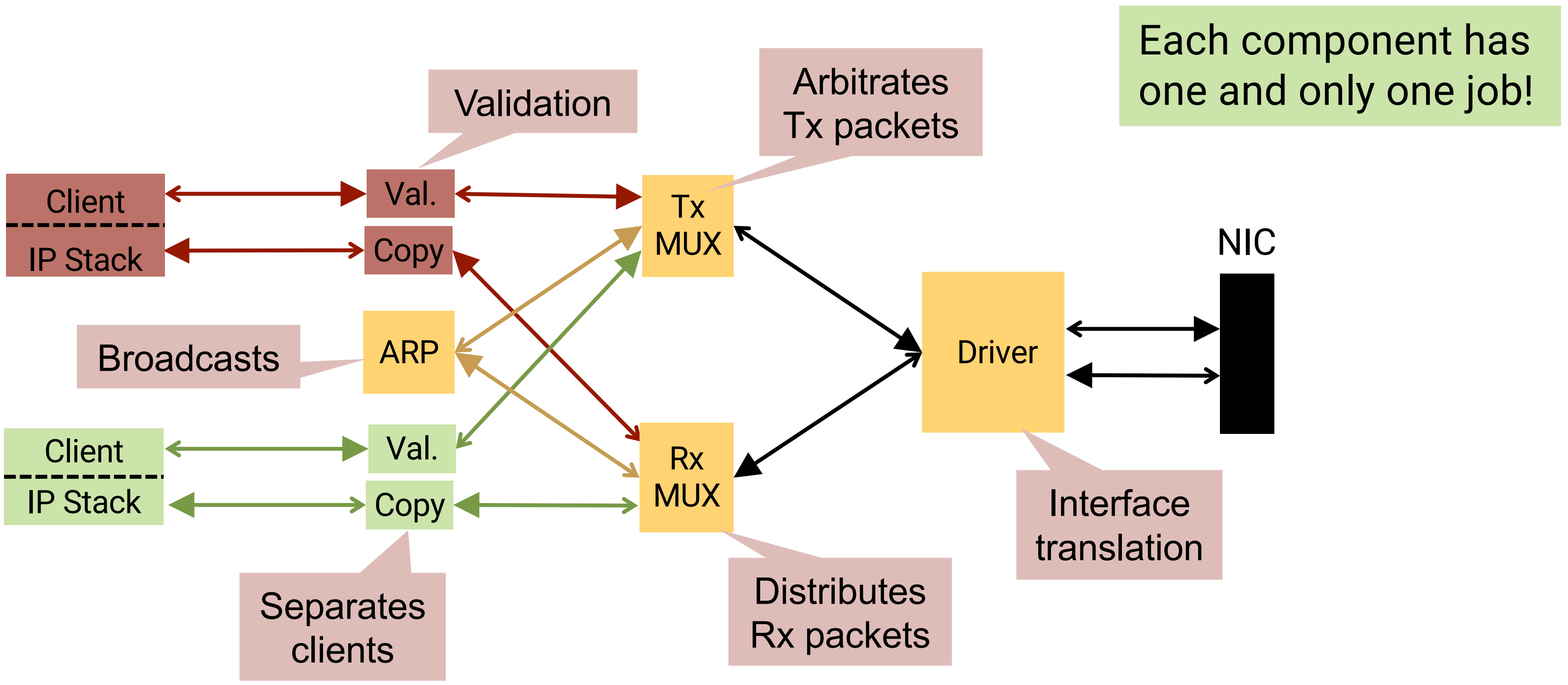
Least Privilege: Device Drivers

Time-honoured security principle
[Saltzer & Schroeder, 1975]



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
- …

Static **architecture**, mostly static resource management

Driver Programming Model

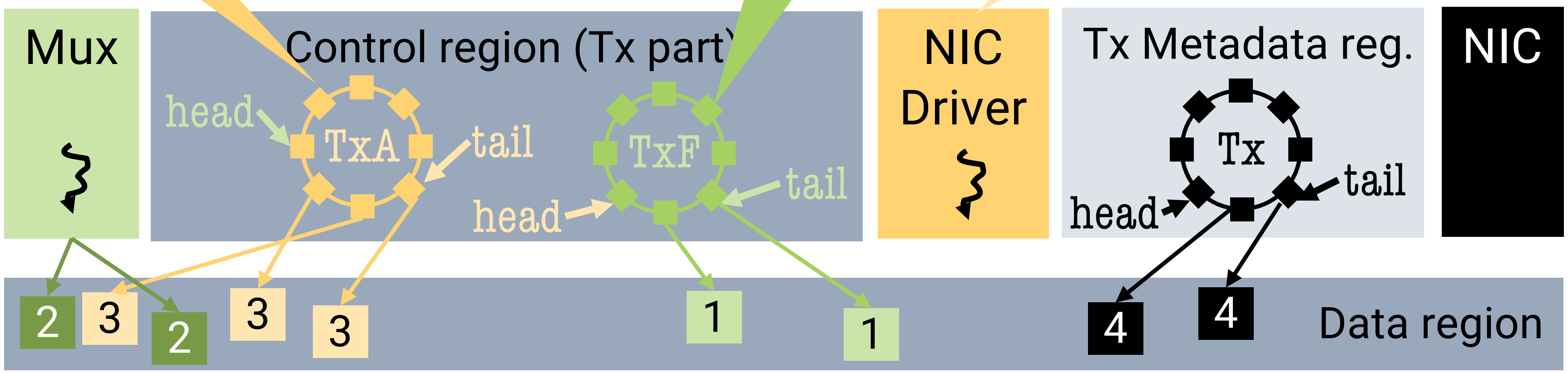
Driver model:
- Single-threaded
- Event-driven
- Simple!

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

Packets to send

Buffers to reuse

Mostly moves pointers between rings



Use-Case–Specific Policies

Source of massive complexity

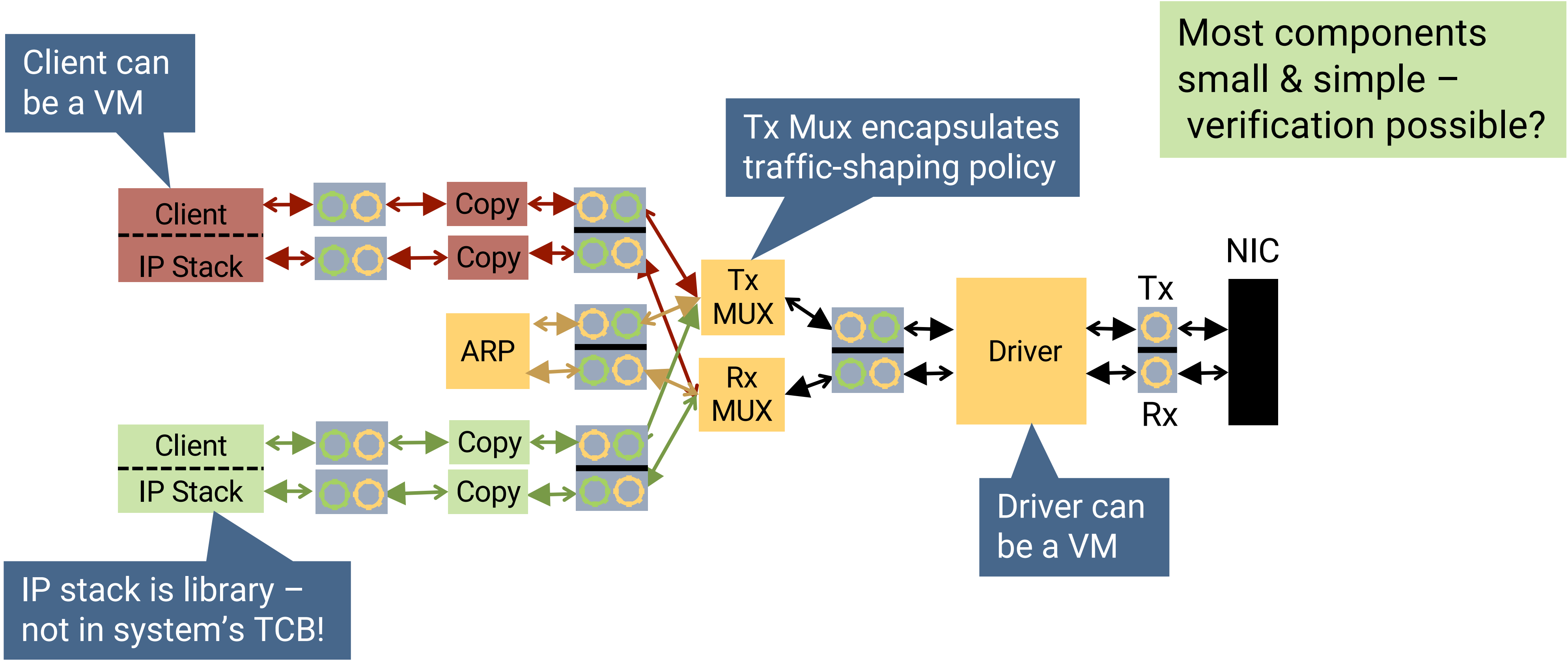
'80s model of computer use!

~~**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


Client can be a VM
Client
IP Stack
Copy
Copy
ARP
Client
IP Stack
Copy
Copy
Tx Mux encapsulates traffic-shaping policy
Tx MUX
Rx MUX
Driver
Tx
Rx
NIC
Most components small & simple – verification possible?
Driver can be a VM
IP stack is library – not in system's TCB!

Comparison to Linux (i.MX8)

Linux:

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

Performance?

seL4 design:

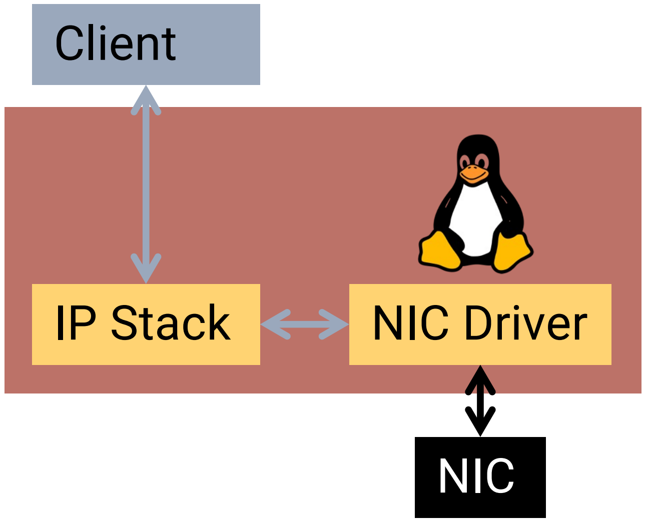
- NW driver: 700 lines
- MUX: 400 lines
- Copier: 200 lines
- IP stack: much simpler, client library
- shared NW system total < 2,000 lines

Written by second-year student!

Evaluation Setup

2 context switches per packet

- External load generator
- Client echoes packets

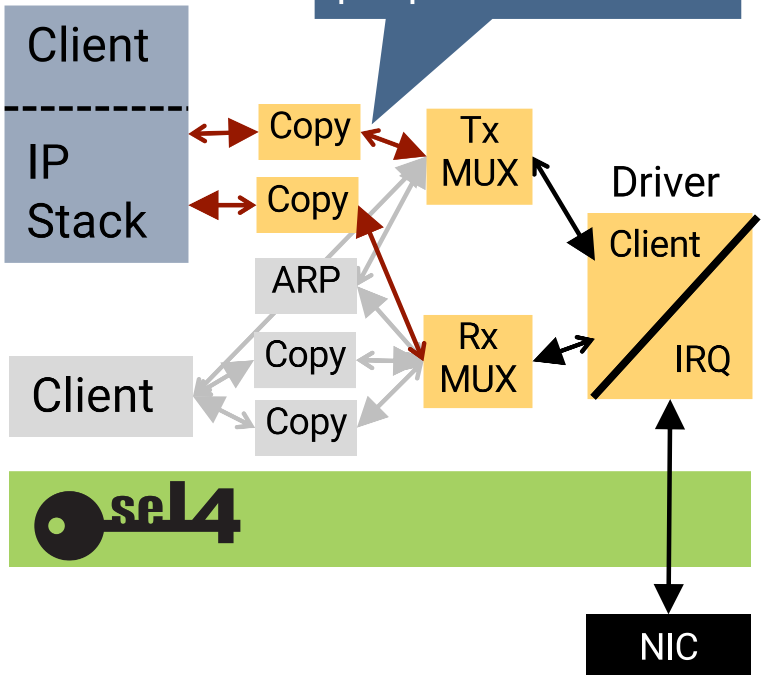

Client

IP Stack

NIC Driver

NIC

10 context switches per packet


Client

IP Stack

Copy

Copy

ARP

Copy

Copy

Client

Tx MUX

Rx MUX

Driver

Client

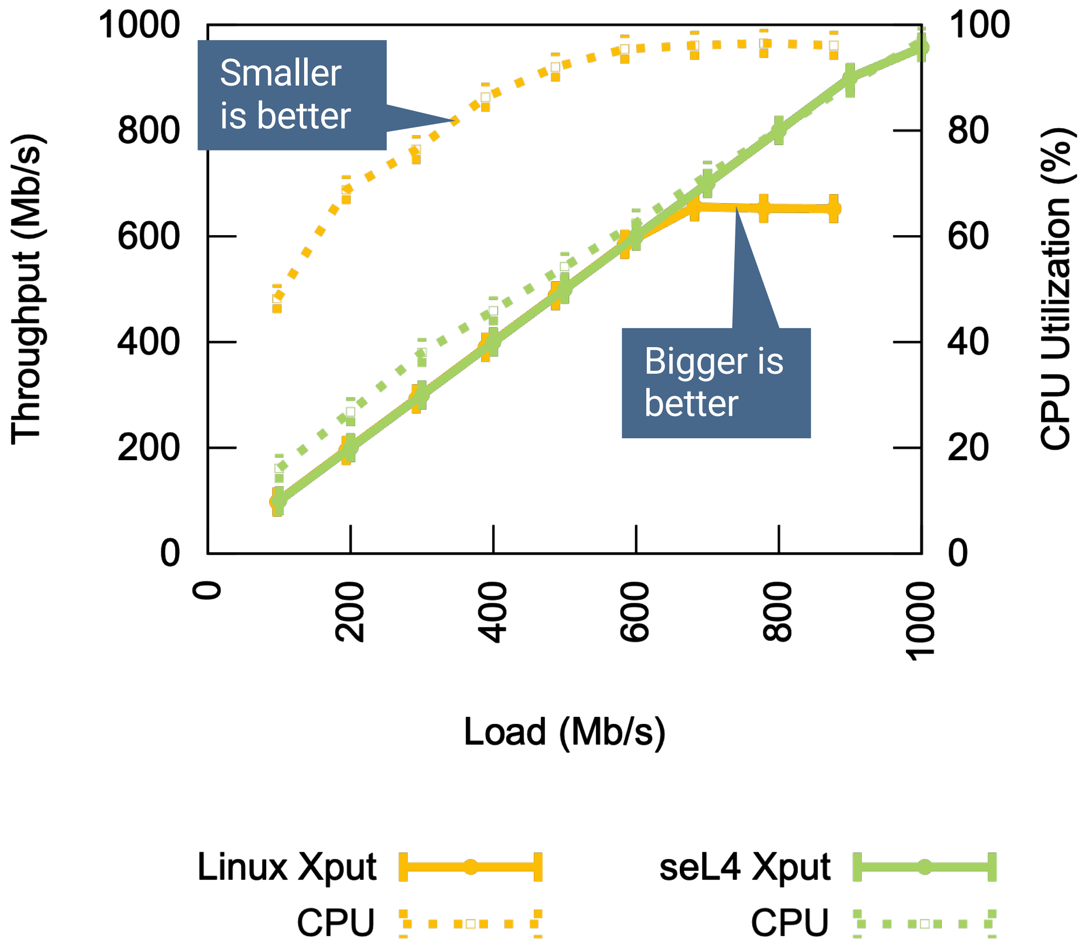
IRQ

seL4

NIC

Achieved Performance: i.MX8

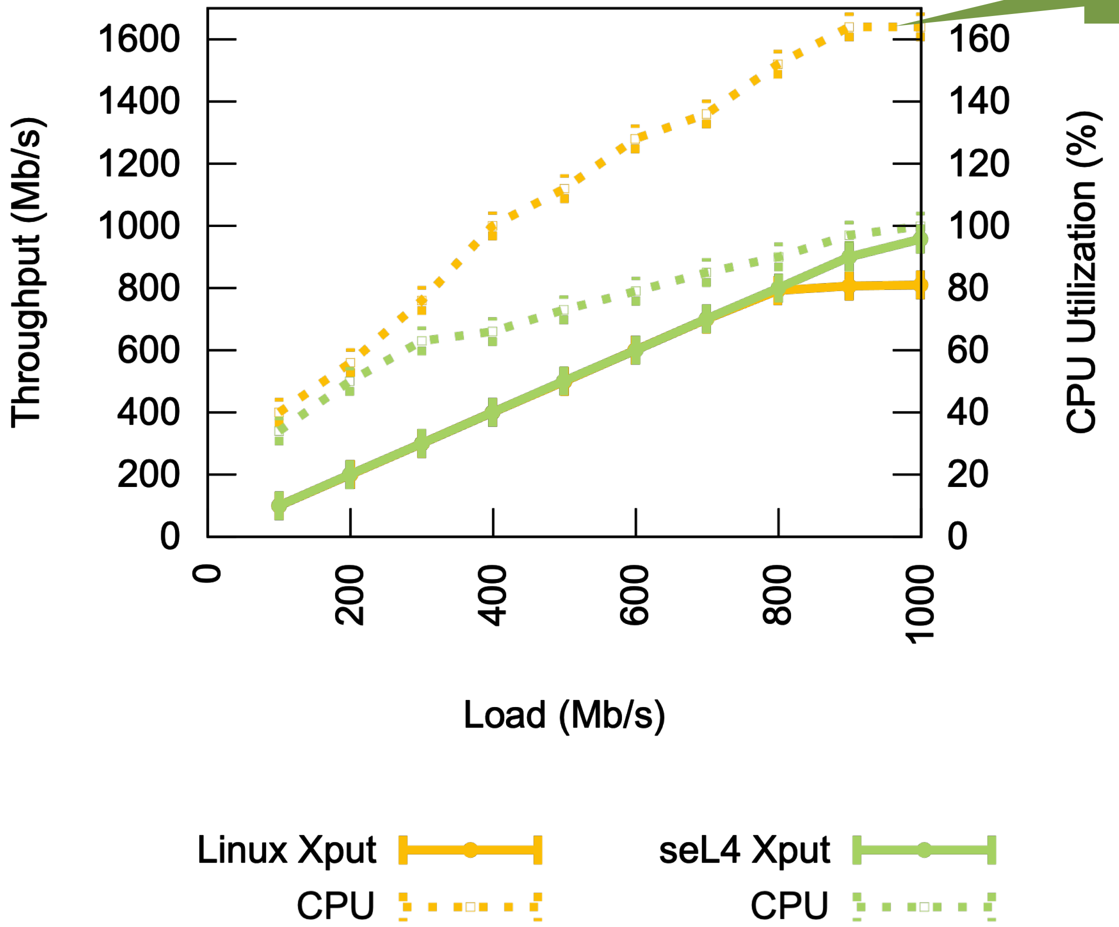
- Gigabit Ethernet
- single core



Graphs Courtesy Lucy Parker

Achieved Performance: i.MX8

- Gigabit Ethernet
- multicore



Simplicity wins!

Graphs Courtesy Lucy Parker

Design for Verification

Verification enabled by:

- modularity
- radical simplicity

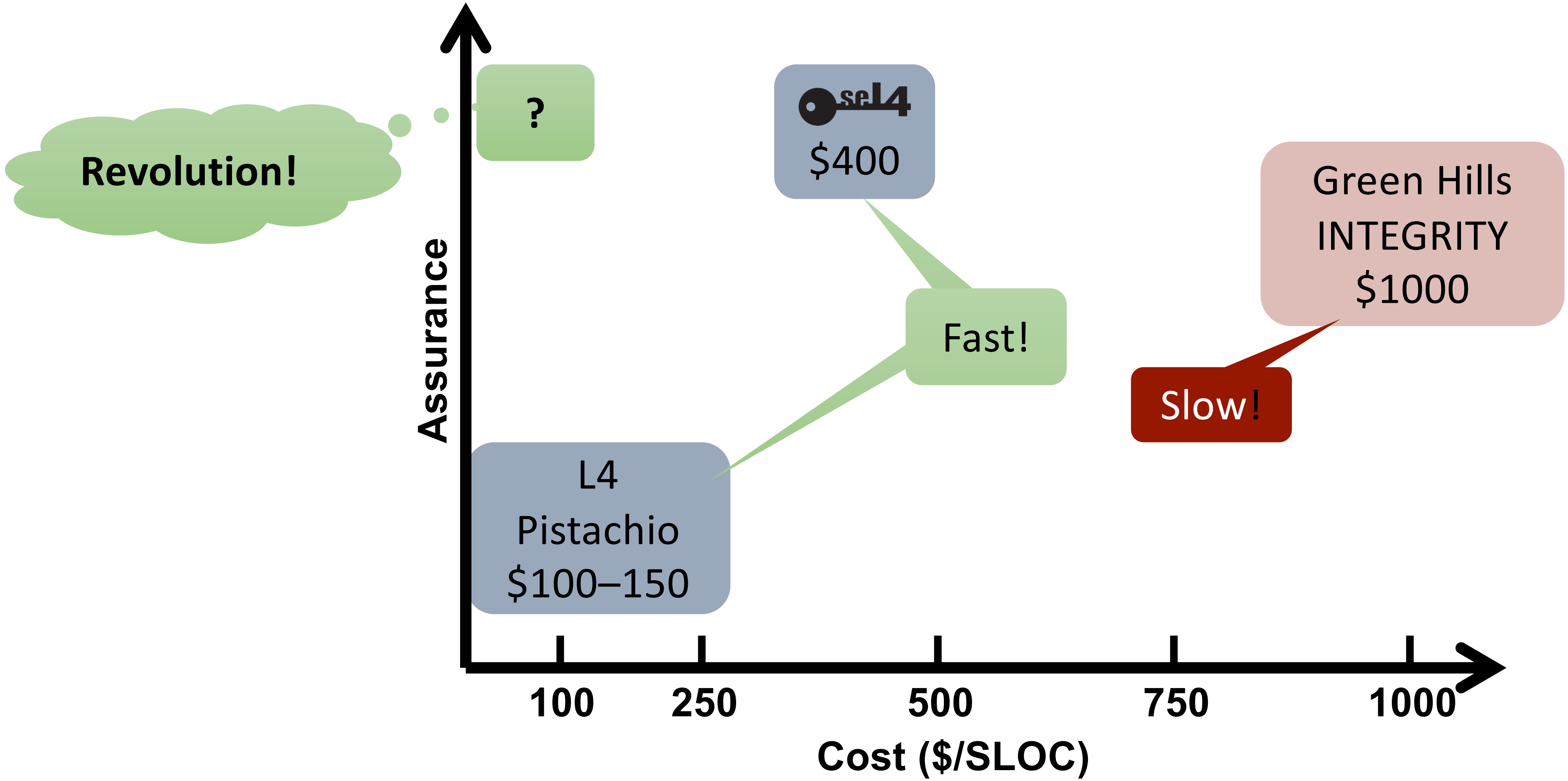
Lions OS Status

- Bulk of funding secured (DARPA, NIO, …)
- Networking system mostly done (Lucy’s thesis)
- File system prototype (design not final)
- First release in Dec’23
  - Complete point-of-sale system
  - Network, storage, touch screen, card reader
  - Components in Rust, Python
- Looking at push-button verification

Other TS Research

Verifying Device Drivers

Remember: Verification Cost in Context



Driver Dilemma

High seL4 verification costs partially due to C language

seL4 is one-off, justifies cost

Drivers are commodity, must be cheap!

Drivers are low-level, need C-like language

Better language would reduce cost

Idea:

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

Lions OS

- Well-defined semantics
- Memory-safe

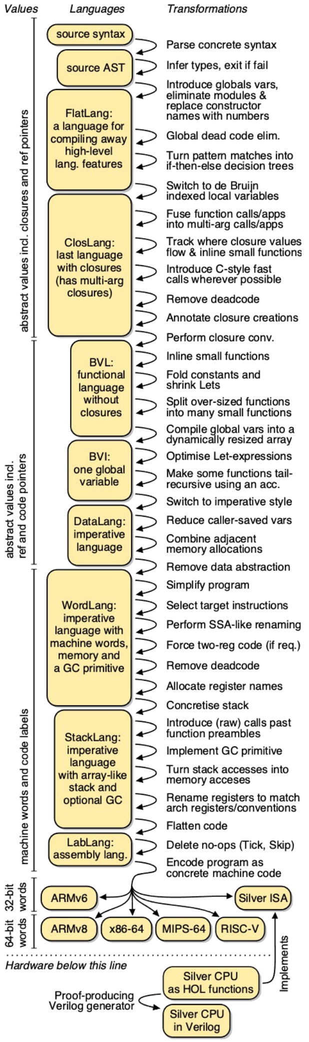
Verified compiler

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

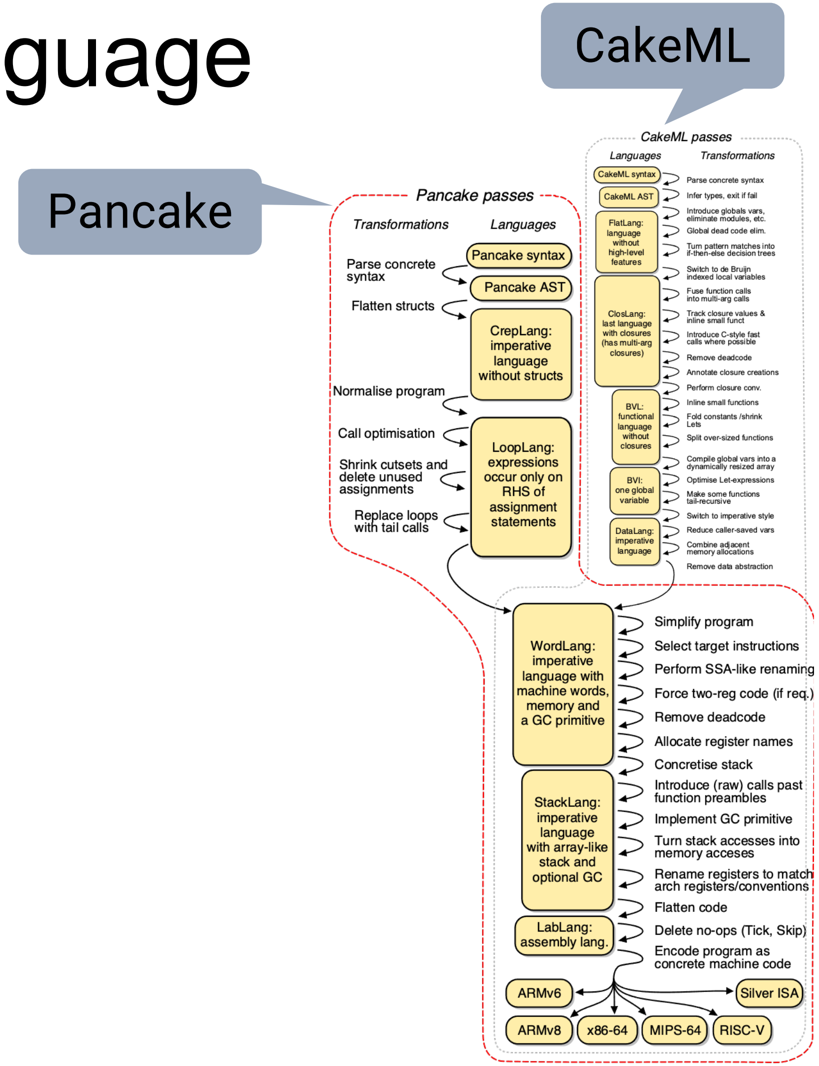
https://cakeml.org


Values
Languages
Transformations
source syntax
Parse concrete syntax
source AST
Infer types, exit if fail
Introduce globals vars, eliminate modules & replace constructor names with numbers
FlatLang: a language for compiling away high-level lang. 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/apps into multi-arg calls/apps
Track where closure values flow & inline small functions
Introduce C-style fast calls wherever possible
Remove deadcode
Annotate closure creations
Perform closure conv.
abstract values incl. closures and ref pointers
BVL: functional language without closures
Inline small functions
Fold constants and shrink Lets
Split over-sized functions into many small functions
Compile global vars into a dynamically resized array
BVI: one global variable
Optimise Let-expressions
Make some functions tail-recursive using an acc.
Switch to imperative style
DataLang: imperative language
Reduce caller-saved vars
Combine adjacent memory allocations
Remove data abstraction
abstract values incl. ref and code pointers
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
StackLang: imperative language with array-like stack and optional GC
Introduce (raw) calls past function preambles
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
machine words and code labels
32-bit words
ARMv6
Silver ISA
64-bit words
ARMv8
x86-64
MIPS-64
RISC-V
Hardware below this line
Implements
Silver CPU as HOL functions
Proof-producing Verilog generator
Silver CPU in Verilog

Pancake: New Systems Language

Approach:

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



CakeML

Pancake

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

Pancake passes

Transformations *Languages*

Pancake syntax — Parse concrete syntax

Pancake AST — Flatten structs

CrepLang: imperative language without structs — Normalise program

LoopLang: expressions occur only on RHS of assignment statements — Call optimisation; Shrink cutsets and delete unused assignments; Replace loops with tail calls

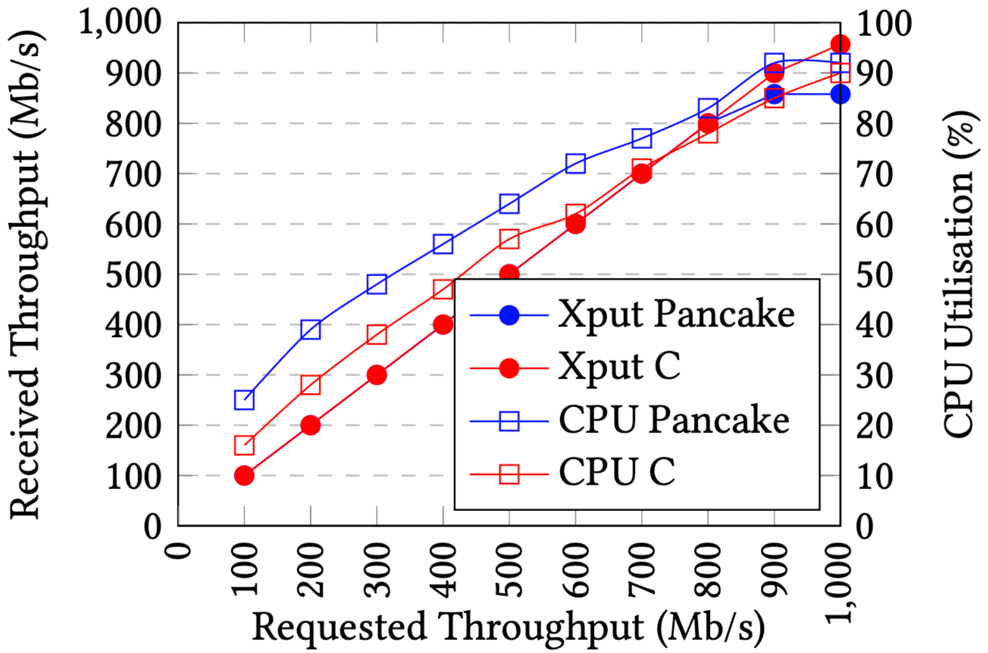
WordLang: imperative language with machine words, memory and a GC primitive — Simplify program; Select target instructions; Perform SSA-like renaming; Force two-reg code (if req.); Remove deadcode; Allocate register names; Concretise stack; Introduce (raw) calls past function preambles

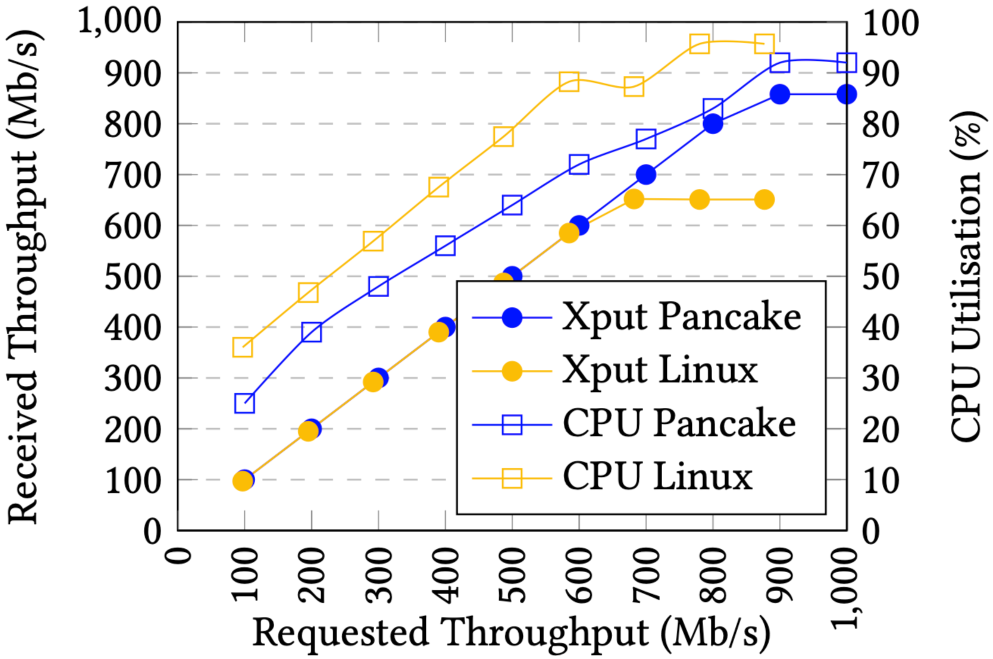
StackLang: imperative language with array-like stack and optional GC — Implement GC primitive; Turn stack accesses into memory acceses; Rename registers to match arch registers/conventions; Flatten code

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 Performance





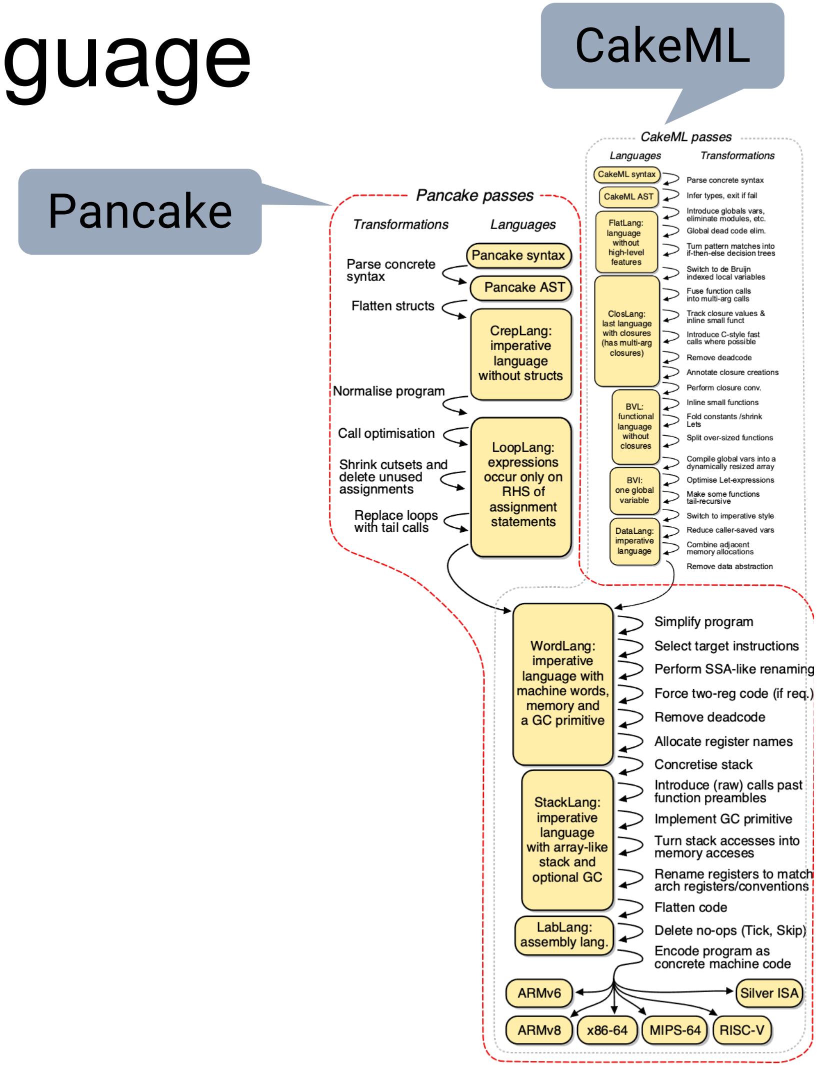
Lions OS setup with Pancake Muxes

Performance degradation for well-understood reasons

Pancake: New Systems Language

Status:

- "Usable" rump language
  - still requires C code for HW access
  - inefficient invocation – re-initialise run-time each time
- Verified compiler
- Limitations well understood and remedies in progress



CakeML
Pancake
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
Pancake passes
Transformations
Languages
Pancake syntax
Parse concrete syntax
Pancake AST
Flatten structs
CrepLang: imperative language without structs
Normalise program
Call optimisation
LoopLang: expressions occur only on RHS of assignment statements
Shrink cutsets and delete unused assignments
Replace loops with tail calls
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
StackLang: imperative language with array-like stack and optional GC
Introduce (raw) calls past function preambles
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

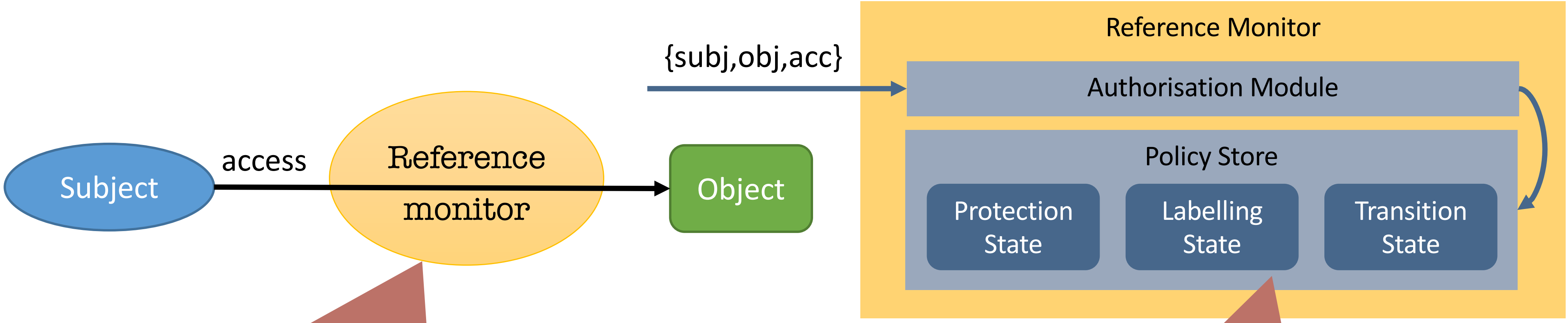
seL4-Related Research in TS

Secure Multi-Server OS

Recap: Secure Operating Systems

Secure OS: [Jaeger: OS Security]

Access enforcement satisfies the *reference monitor* concept

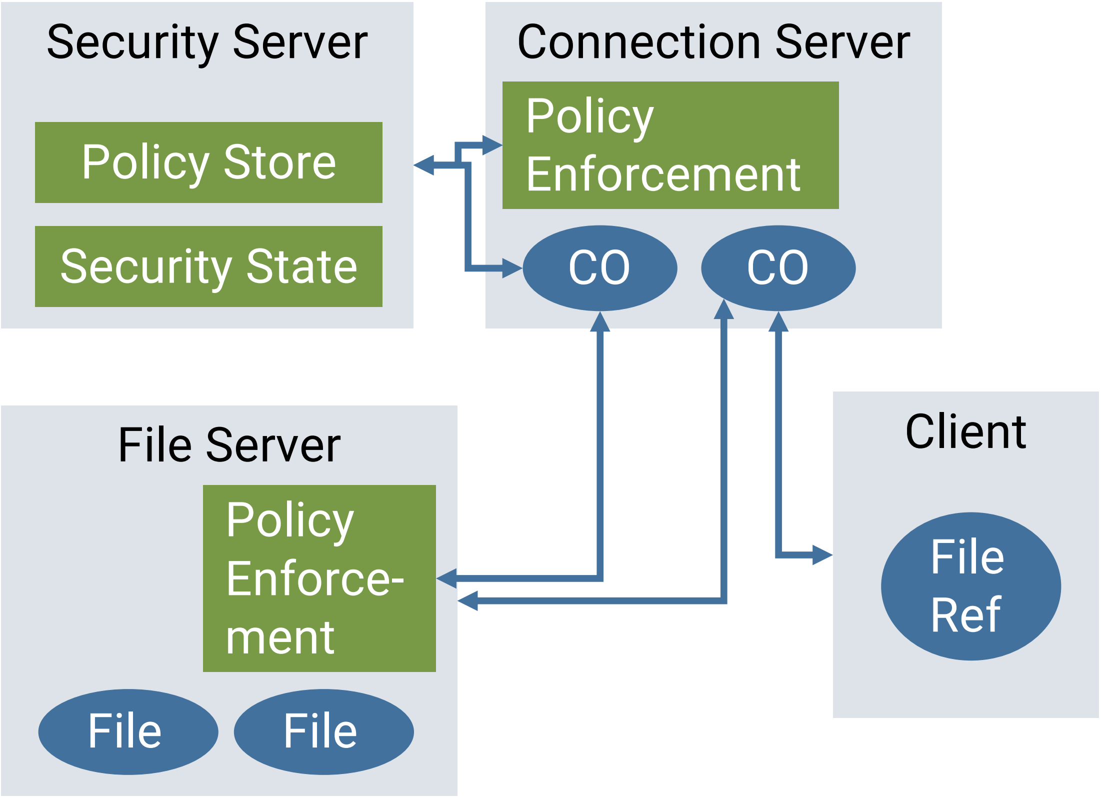


Enforces *mandatory protection:*

- non-bypassable
- tamperproof
- verifiable

Permission: relation over labels

seL4 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

Thank you!

To the brave AOS students for their interest and dedication

To the world-class Trustworthy Systems team for making all possible

Please remember to do the myExperience survey

There'll also be a more detailed one we'll invite you to fill in

John Lions Honours Scholarship closes this week!

https://www.scholarships.unsw.edu.au/scholarships/id/1757/6077