UNSW
SYDNEY
Australia's
Global
University

School of Computer Science & Engineering

Trustworthy Systems Group

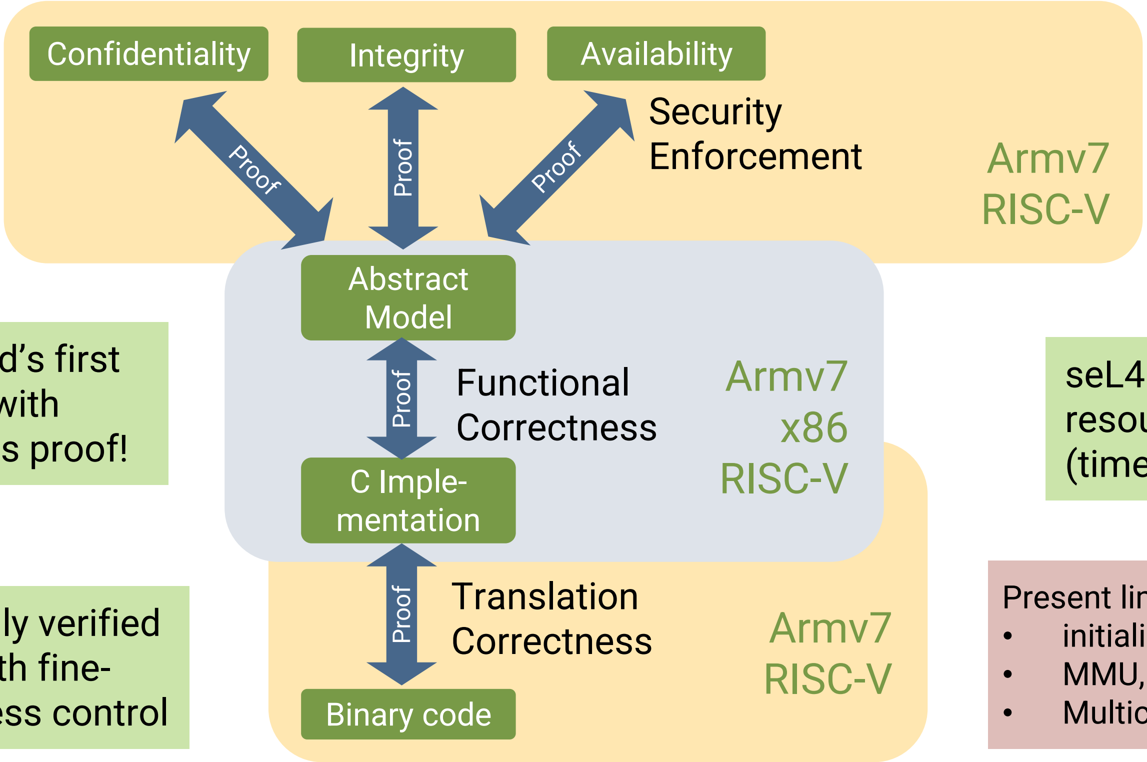
State of seL4-related Research
at Trustworthy Systems

Gernot Heiser
gernot@trustworthy.systems

seL4 SUMMIT

TS

seL4 Success Story – What's Next?



AArch64 in progress

seL4: World's first OS kernel with correctness proof!

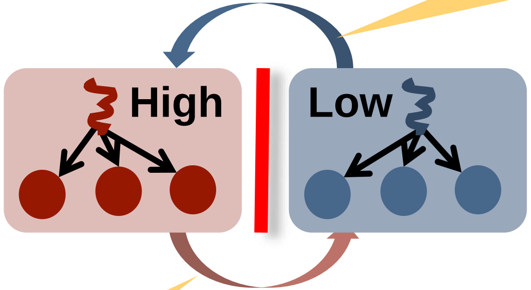
seL4: Unique, policy-free resource management (time and space)

seL4: Still only verified OS kernel with fine-grained access control

Present limitations
- initialisation code not verified
- MMU, caches modelled abstractly
- Multicore not yet verified

Time – The Final Frontier

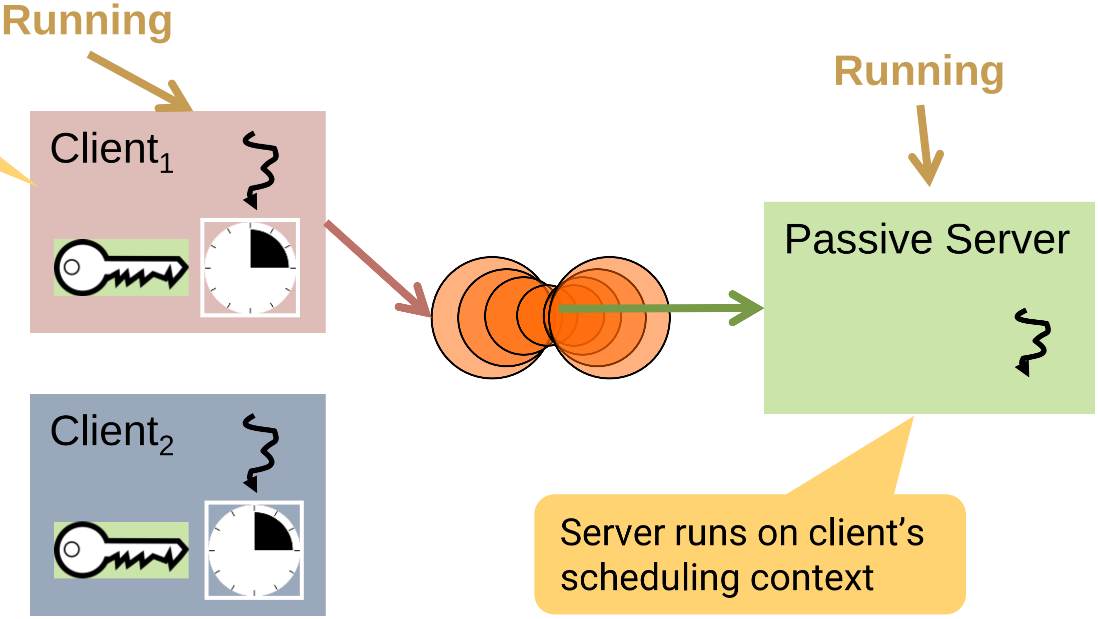
Issues With Time



Low affects High's progress
- Cause deadline miss
- **Integrity violation**

High affects Low's progress
- Information leakage
- **Confidentiality violation**

Temporal Integrity: MCS Kernel



Client is charged for server's time

Server runs on client's scheduling context

- Time as a first-class resource
- Restrict CPU access for high-prio threads
- Time-out exception if budget expires in server

However: Complex recovery, transaction semantics needed

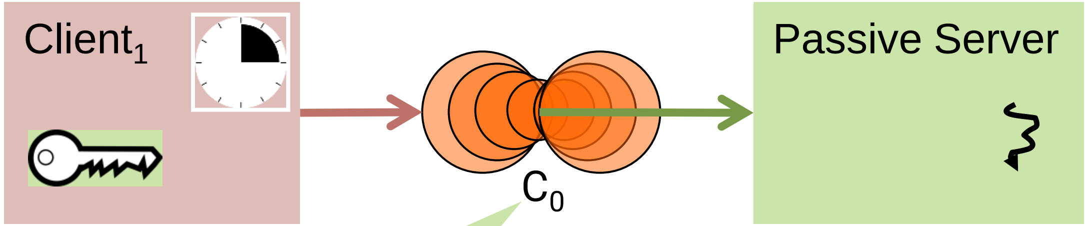
Goal: Simple Servers, Minimal Policy

Idea: Budget contract

1. Client cannot enter server with less than C_0 budget
2. Server cannot consume more than C_0 budget

No budget expiry in well-configured server

Protect client from mis-behaving server



Client$_1$

C_0

Passive Server

Threshold is Endpoint attribute

Invocation deferred if remaining budget is insufficient

Status:

- Student Mitch Johnston working through various implementation issues
- Expect RFC soon

Later: Formal Scheduling Analysis

Challenge: Prove timeliness of critical real-time components

- MCS provides mechanisms
- WCET analysis of kernel done (for old version on old HW 😢)
- In principle can reason about schedulability

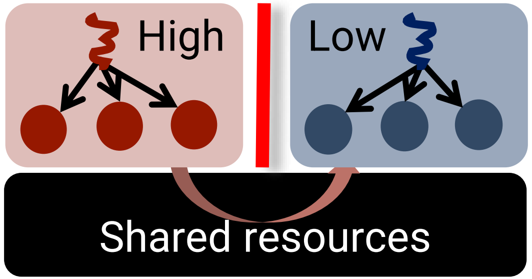
Reality:

- Need to resolve usability issues with MCS
- WCET analysis for old version on old HW 😢
- More theory work needed

Status:

- Not started yet
- Looking for good PhD student!

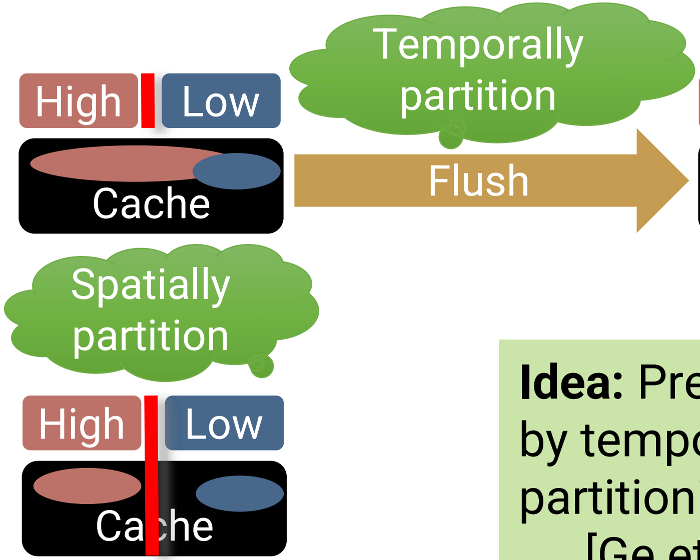
Confidentiality: Timing Channels



Microarchitectural timing channels: Contention for shared hardware resources affects execution speed

Standard approach: Patch & Pray

Time Protection: Principled Prevention



Aim: *Provably prevent* information flow through micro-architectural timing channels

Idea: Prevent channels by temporal or spatial partitioning of **all** HW [Ge et al, EuroSys'19]

Temporal Partitioning: Flush on Switch

Must remove any history dependence!

```
1. $T_0$ = current_time()
2. Switch user context
3. Flush on-core state
4. while ($T_0$+WCET < current_time()) ;
5. Reprogram timer
6. return
```

Latency depends on prior execution!

Time padding to remove dependency

Proving Temporal Partitioning

Must remove any history dependence!

```
1. $T_0$ = current_time()
2. Switch user context
3. Flush on-core state
4. while ($T_0$+WCET < current_time()) ;
5. Reprogram timer
6. return
```

Prove: flush all non-partitioned HW

- Needs model of stateful HW
- Somewhat idealised on present HW … but matches RISC-V prototype
- **Functional property**

Prove: access to shared data is deterministic

- Each access sees same cache state
- Needs cache model
- **Functional property**

Prove: padding is correct

Padding: Use Minimal Clock Abstraction

Abstract clock = monotonically increasing counter

Operations:

- Add constant to clock value
- Compare clock values

To prove: padding loop terminates as soon as clock ≥ T0+WCET

- **Functional property!**

Time Protection Verification: Status

1. [Done] Specify isolation property
2. [Done] Prove enforcement on high-level model
3. [In progress] Connect to seL4 proofs
    1. [Done] Update seL4 abstract specification to account for memory accesses
    2. Prove these accesses are bounded according to security policy
    3. Connect 3.1-3.2 to high-level model to prove isolation property
    4. Prove preservation of 3.1-3.3 by refinement to lower-level seL4 specifications

Support:

- Australian Research Council
- USAF-AOARD
- NCSC (UK)

Hardware Support for Time Protection

Hardware Reality:
Mainstream processors do not allow resetting all history-dependent state!
[Ge et al., APSys’18]

```
1.  $T_0$ = current_time()
2.  Switch user context
3.  Flush on-core state
4.  while ($T_0$+WCET < current_time()) ;
5.  Reprogram timer
6.  return
```

RISC-V to the rescue!

- Add instruction to clean state
- Also help with padding
- **See talk by Nils Wistoff**

Multicore Performance

Getting Rid of the Big Kernel Lock?

Background:

- Multicore seL4 uses a single big lock
- Works because seL4 syscalls are short
- Makes sense as long cost of migrating cache line is small fraction of syscall cost

Aim:
Resolve locking issue before progressing with multicore verification

Issue:

- While not generally a performance issue, BKL leads to very pessimistic WCET
- Also large cross-core timing channels
- Removing take single-kernel image further

Getting Rid of the Big Kernel Lock?

Status:
- Done: Implementations for x86 and Arm
- Done: Proofs of desired properties
- In progress: Implementation in seL4

Writer has to wait at most 1 reader's locking time to obtain lock

Idea:
- Bounded reader-writer lock
- Lock-free updates

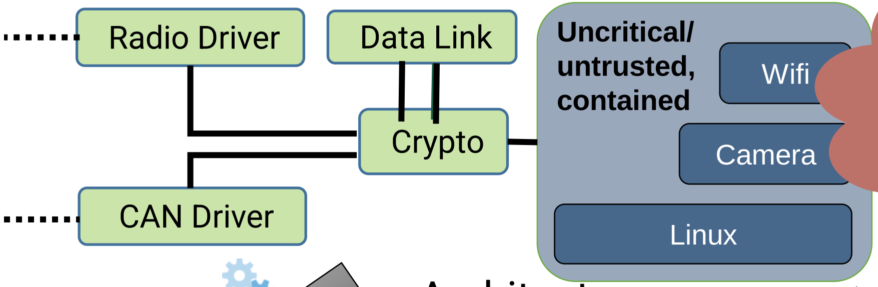
Support:
- NCSC (UK)

So, Why Isn’t seL4 Everywhere by Now?

- Usability
- Functionality: Native services
- Trustworthiness: More than the kernel
- Applicability: Embedded vs general-purpose

Usability

Recommended Framework: CAmkES

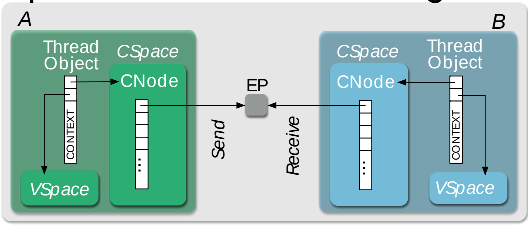


- Good for assurance
- Bad for usability & functionality

Conditions apply

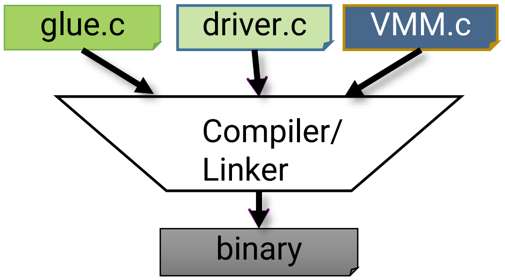
Architecture specification

CapDL: Low-level access rights



init.c

- Forces use of kernel build system on apps
- Fully static
- Hard to extend
- Significant overheads



New Framework: seL4 Core Platform

Small OS/SDK for IoT, cyber-physical and other embedded use cases

- Leverage seL4-enforced isolation for strong security/safety
- Lean, retain seL4's superior performance
- Retain near-minimal trusted computing base (TCB)
- Integrate with build system of your choice
- Support ”correct” use of seL4 mechanisms by default
- Be amenable to formal verification of the TCB

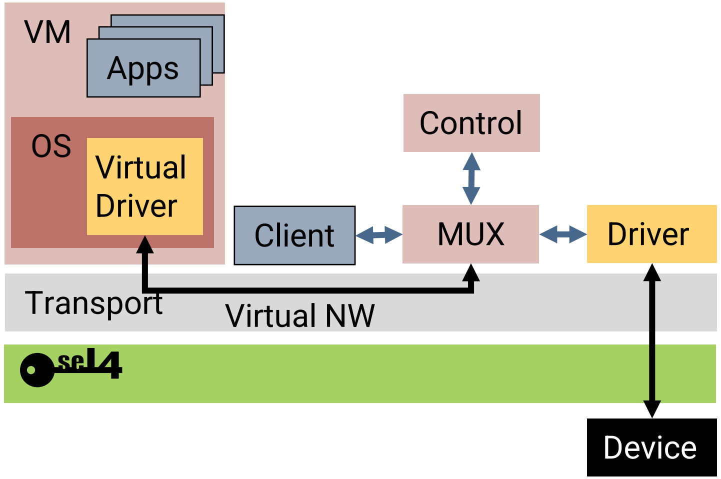
Details in Zoltan Kocsis' talk

Support:

- NCSC (UK)

Functionality: Native Services

Key Component: Driver Framework



Aim:

- Simple model for robust drivers
- Secure, low-overhead sharing of devices between components
- Low overhead

Approach:

- Zero-copy transport layer
- Standard interfaces, virtIO
- Re-use Linux drivers in per-device VM
- Investigate verifying MUX, Controller

Details in Lucy Parker's talk

Support:

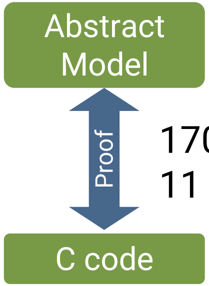
- seL4 Foundation
- TII

Trustworthiness

More than the kernel

Cost of Verification?

Verifying code not written for verification is infeasible, significant expertise required for writing verifiable code!



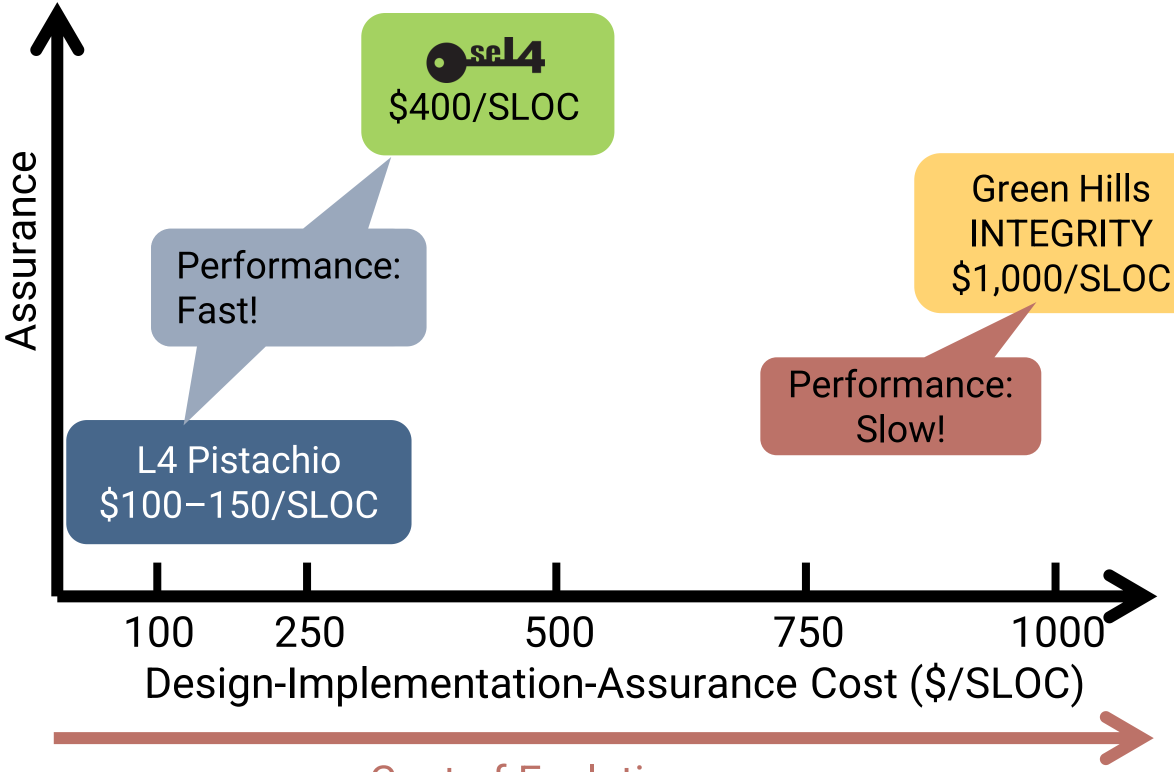
170,000 lines of proof
11 person years

10,000 lines of code

Designed and implemented for verification!

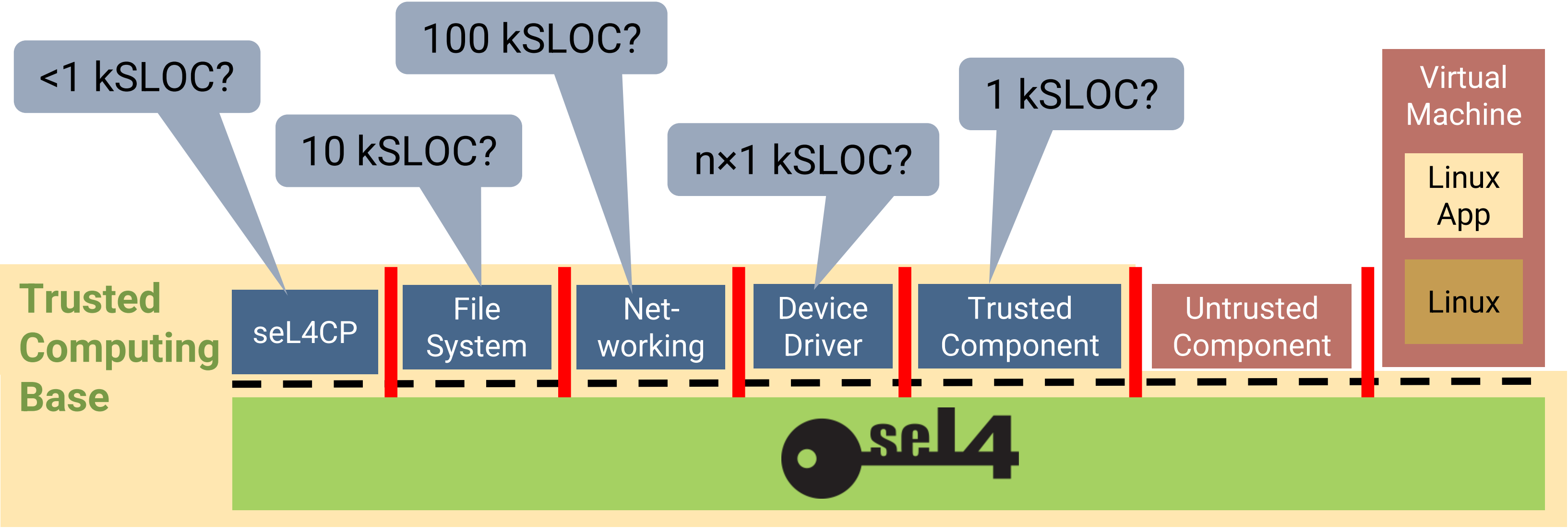
Complete seL4 proof base now ≫ 1,000,000 lines!

Verification Cost in Context



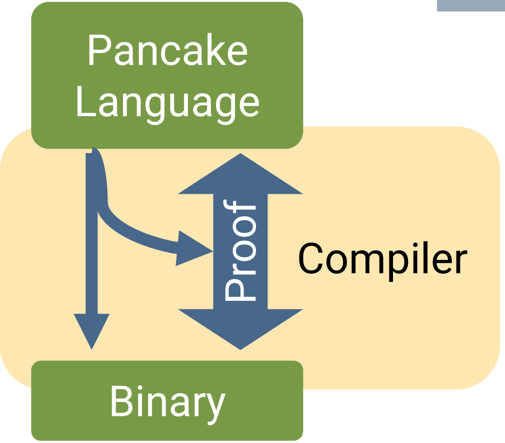
Cost of Evolution

Beyond the Kernel



Reducing Cost of Verified Systems Code

Aim: Simplify verifying user-level OS components



Idea:

- Use low-level but safe systems language with certifying compiler
- Gives many proof obligations for free

Systems language:

- memory safe
- not managed (no garbage collector)
- low-level (obvious translation)
- interfacing to hardware
- no run-time system

Approach: Re-Use CakeML Framework

CakeML:

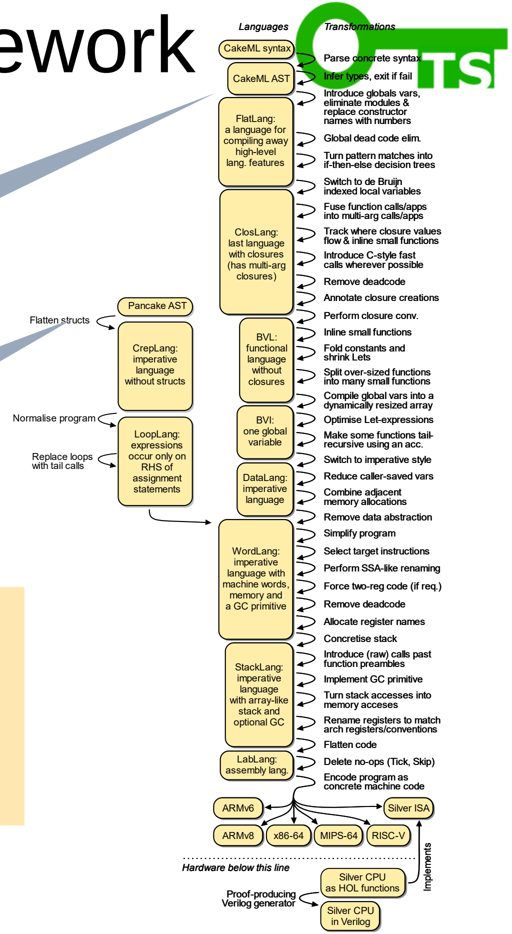
- functional language
- type & memory safe
- managed (garbage collector)
- high-level, abstract machine
- verified run time
- verified compiler
- mature system
- active ecosystem

Great, but too high-level!

CakeML compiler

Pancake compiler

Approach:
Re-use lower part of CakeML compiler stack for imperative language



Languages
Transformations
CakeML syntax
Parse concrete syntax
CakeML 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.
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
Pancake AST
Flatten structs
CrepLang: imperative language without structs
Normalise program
LoopLang: expressions occur only on RHS of assignment statements
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
ARMv8
x86-64
MIPS-64
RISC-V
Silver ISA
Hardware below this line
Silver CPU as HOL functions
Implements
Proof-producing Verilog generator
Silver CPU in Verilog

Verified Pancake Compiler

Pancake compiler is written in CakeML
⇒ can use CakeML compiler to produce verified Pancake compiler binary!

Status:

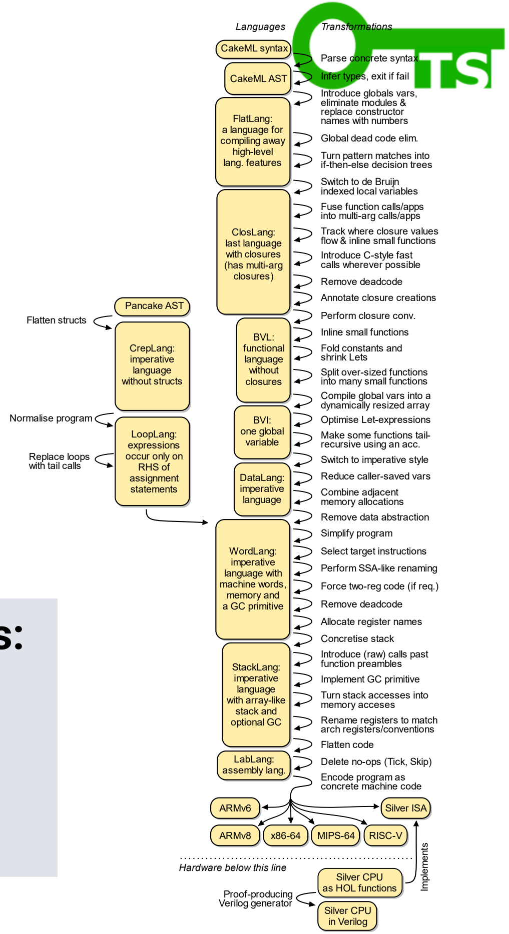
- Mostly done: Toy (serial) driver verification to explore semantics
- Prototype done: Parser
- Almost done: Verification of link to CakeML compiler:
- In progress: Binary compiler bootstrap
- Not started: Shared-memory driver-device, driver-client

Collaborators:

- ANU
- Chalmers

Support:

- TII



Languages

Transformations

CakeML syntax

Parse concrete syntax

CakeML 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.

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

Pancake AST

Flatten structs

CrepLang: imperative language without structs

Normalise program

LoopLang: expressions occur only on RHS of assignment statements

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

Hardware below this line

Implements

Silver CPU as HOL functions

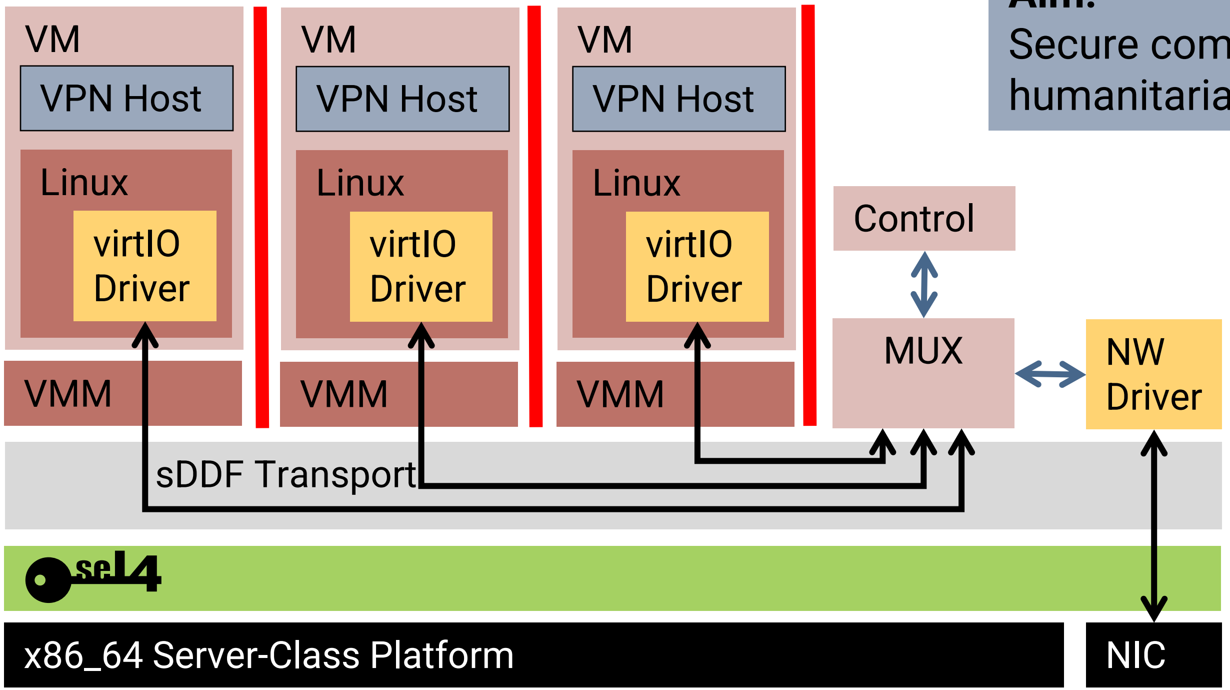
Proof-producing Verilog generator

Silver CPU in Verilog

Applicability

Embedded vs General-Purpose

Makatea: Secure VPN Service



Aim:
Secure communications for humanitarian organisations

Requires:
- Support for server-class Intel platform
- Efficient network virtualisation

Client:
- Neutrality

Support:
- NLnet

Provably Secure General-Purpose OS

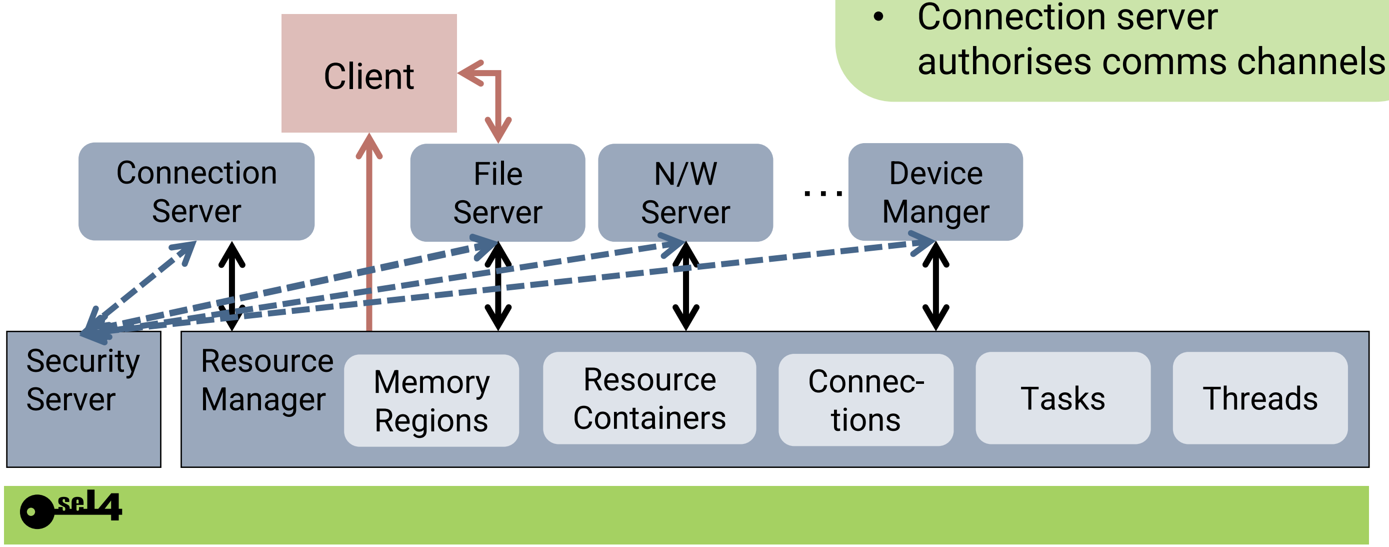
Problem:
- GP-OS with security policy diversity
- Proof that policy is enforced
- Performance

Solution:
- Multi-server OS with policy isolated in security server
- Object servers provable to ensure complete mediation
- Connection server authorises comms channels

Status:
- prototyping core servers

Partners
Penn State

Support
NCSC



FAQ: If You Did It Again, What Would Be Different?

Major Issues?

Addressed by
- reply caps
- reply objects

Main issues with original seL4:
- Need protocols for establishing reply channel
- Naïve scheduling with no serious time management

Addressed by scheduling contexts (MCS)

Annoyances [1/2]: Map/Unmap Args

Issue:

- Mapping operates on frame, taking address space as argument:
  `frame_c.Map(AS_c, vaddr)`
- User view is that the the mapping is added to the AS, which is modified:
  `AS_c.Map(frame_c, vaddr)`

Cost:

- Mapping multiple frames requires one syscall per frame
- Same for Unmap

Multi-frame operations:

- Process creation
- Write-protecting/unprotecting for
  - copy-on-write
  - garbage collection

Better:

- `AS_c.Map(frame_c, vaddr,`

  `frame_c, vaddr, …)`
- `AS_c.Unmap(vaddr, vaddr, …)`

Status:

- SMOS, AutoOS will demonstrate costs

Annoyances [2/2]: Lazy FPU Switch

Present FPU context switching is lazy:

1. At context switch, disable FPU
2. Access causes fault
3. On fault, switch FPU state & enable

Issue:

- Compilers use FPU registers for string ops, etc
- Most app code uses FPU
- No benefit from lazy switching

Cost:

- Extra kernel entry
- For servers not using FPU:
  - wastes memory in thread control block
  - WCET must assume FPU switch!

Better:

- Principled resource management: make FPU access a right, provided by FPU object
- Switch FPU eagerly

Issues Under Investigation

Issue:

- Signal that unblocks thread moves it to front of scheduling queue
- ACKing IRQ requires a syscall
- Can we abort IPC by Signal?

Messes with scheduling analysis

Why not implicit in waiting on IRQ Notification?

- Would much simplify timeout implementation
- Idea is to have a mask that says which Signals may abort

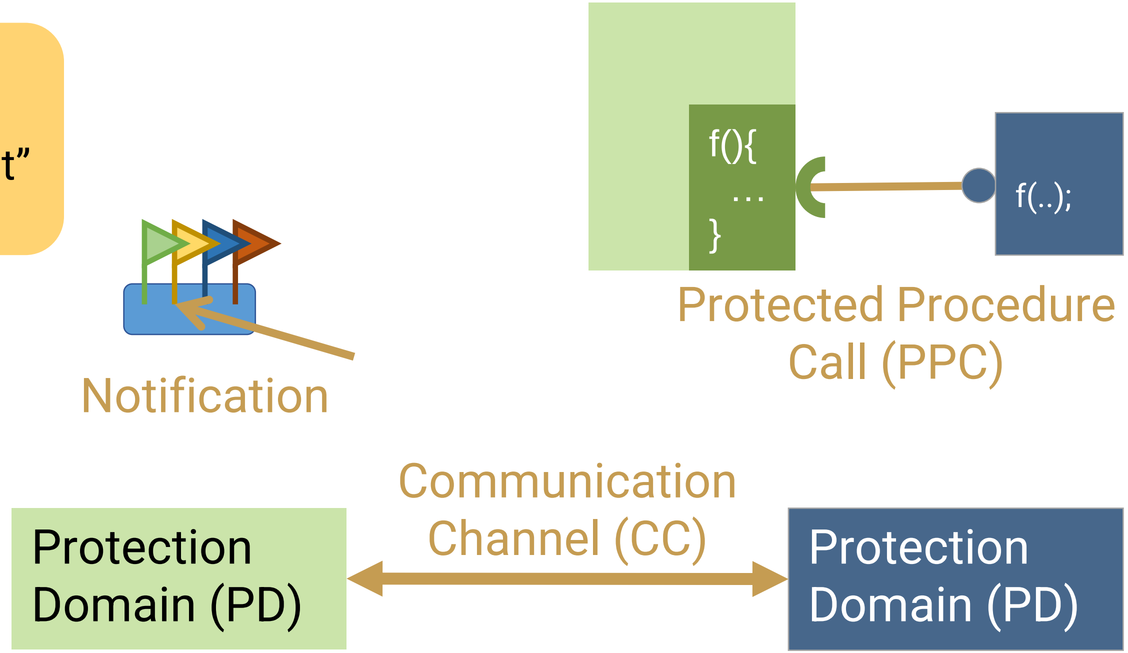
Summary

- **seL4 is the best – but we can still improve it!**
  - Budget thresholds: simplify implementation of passive servers
  - Time protection: principled way for *preventing* timing channels
  - Improved locks: make multicore better
  - Hopefully get rid of some long-standing annoyances
- **seL4 is real-world capable – but we can make it easier!**
  - seL4 Core Platform: lean & easy to deploy
  - seL4 Device Driver Framework: ease driver writing
  - Pancake: towards verified device drivers
- **seL4 can own the embedded space – but we can take it further!**
  - seL4 on server platforms
  - General-purpose, provably-secure system

Defining the state of the art in trustworthy systems since 2009

seL4CP Abstractions

- Thin wrapper of seL4 abstractions
- Encourage “correct” use of seL4



Notification

Protected Procedure Call (PPC)

Communication Channel (CC)

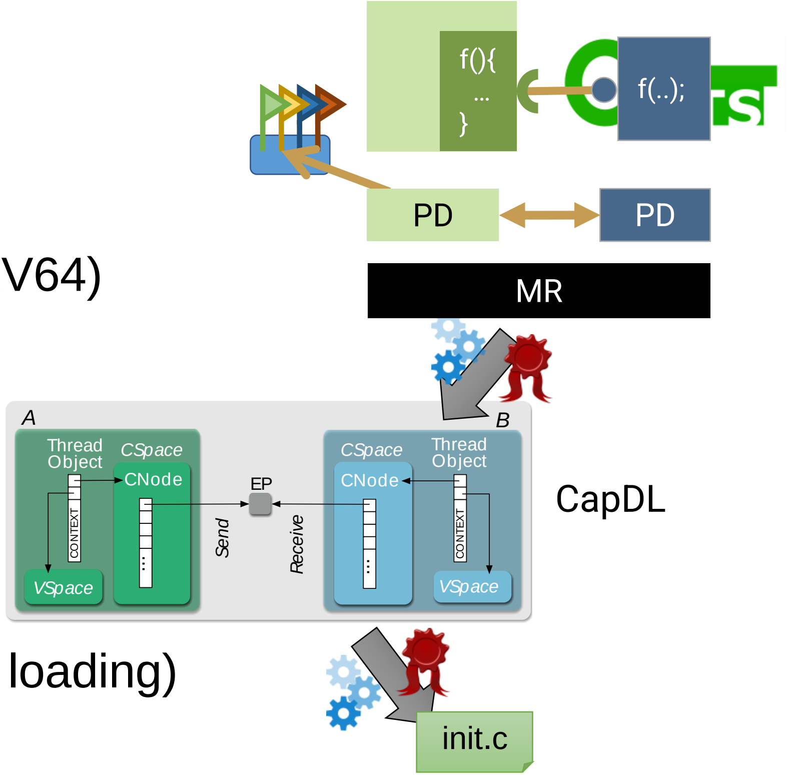
Protection Domain (PD)

Protection Domain (PD)

Memory Region (MR)

seL4CP Status

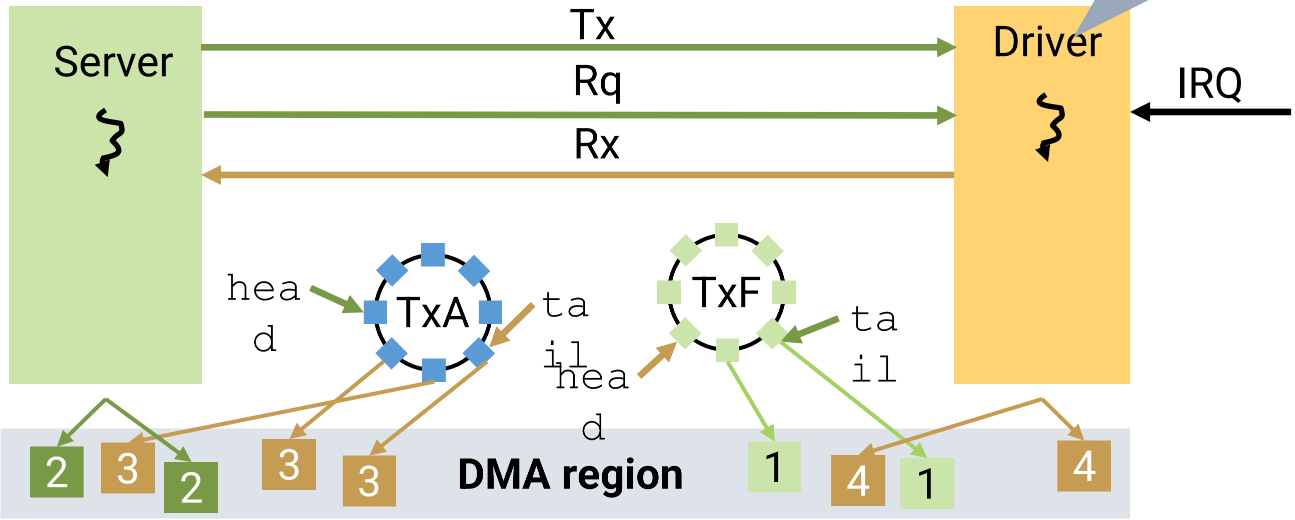
- Used in products (AArch64-based)
- Platform and ISA ports in progress (x64, RV64)
- Virtualisation support in progress
- Dynamic features prototype:
    - fault handlers
    - start/stop protection domains
    - re-initialise protection domains
    - empty protection domains (for late app loading)
- Verified mapping to CapDL in progress
- Push-button verification of CapDL under investigation



Low-Overhead Transport

Status:
- Optimising transport layer
- Release soon

- Single-threaded
- Event-driven


Server
Tx
Rq
Rx
Driver
IRQ
head
TxA
tail
TxF
head
tail
2
3
2
3
3
DMA region
1
4
1
4