

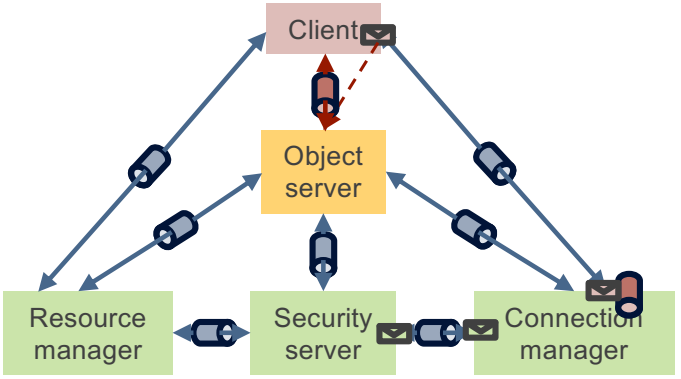
School of Computer Science & Engineering

COMP9242 Advanced Operating Systems

2026 T3 Week 10 Part 2

seL4-related Research at TS@UNSW

@GernotHeiser



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

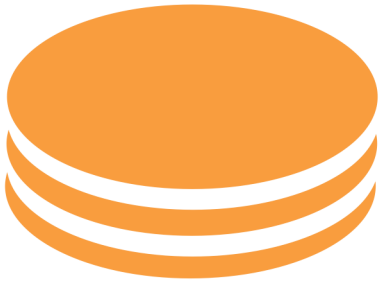
© Gernot Heiser 2019-24 – CC BY 4.0

Today's Lecture

- Pancake: System language for verified systems
- Carrells: Microservices on LionsOS
- Djawula: Secure General-purpose OS
- Time Protection: Prevention of μarch timing channels

© Gernot Heiser 2019-24 – CC BY 4.0

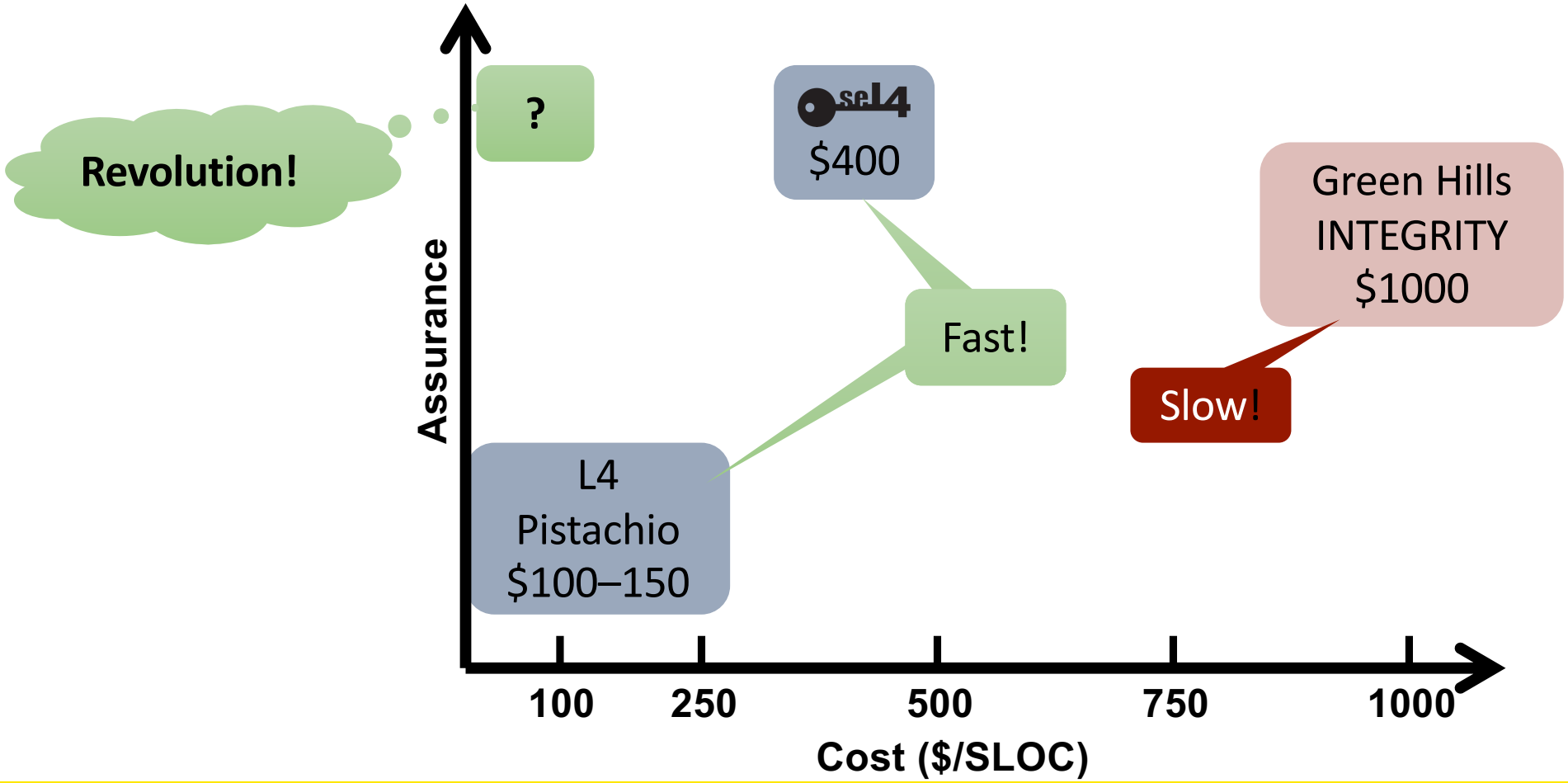
Scaling Verification



PANCAKE
A Language for
Verified Systems Programming

© Gernot Heiser 2019-24 – CC BY 4.0

Verification Cost in Context



© Gernot Heiser 2019-24 – CC BY 4.0

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
- SMT solvers
- de-compilation

© Gernot Heiser 2019-24 – CC BY 4.0

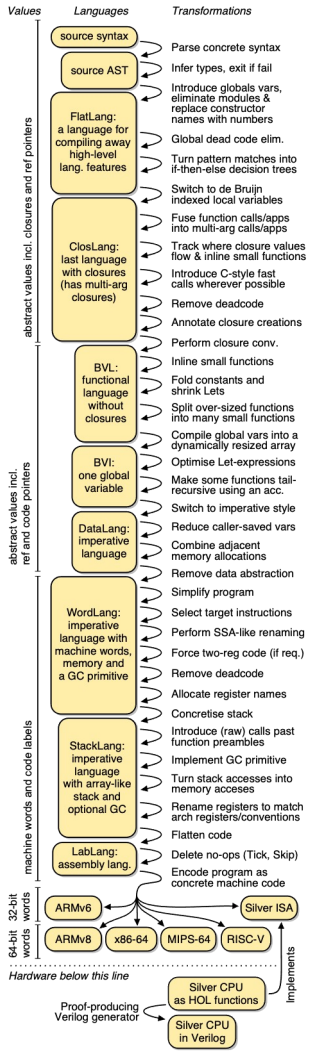
CakeML: Verified Implementation of ML



- ✓ 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

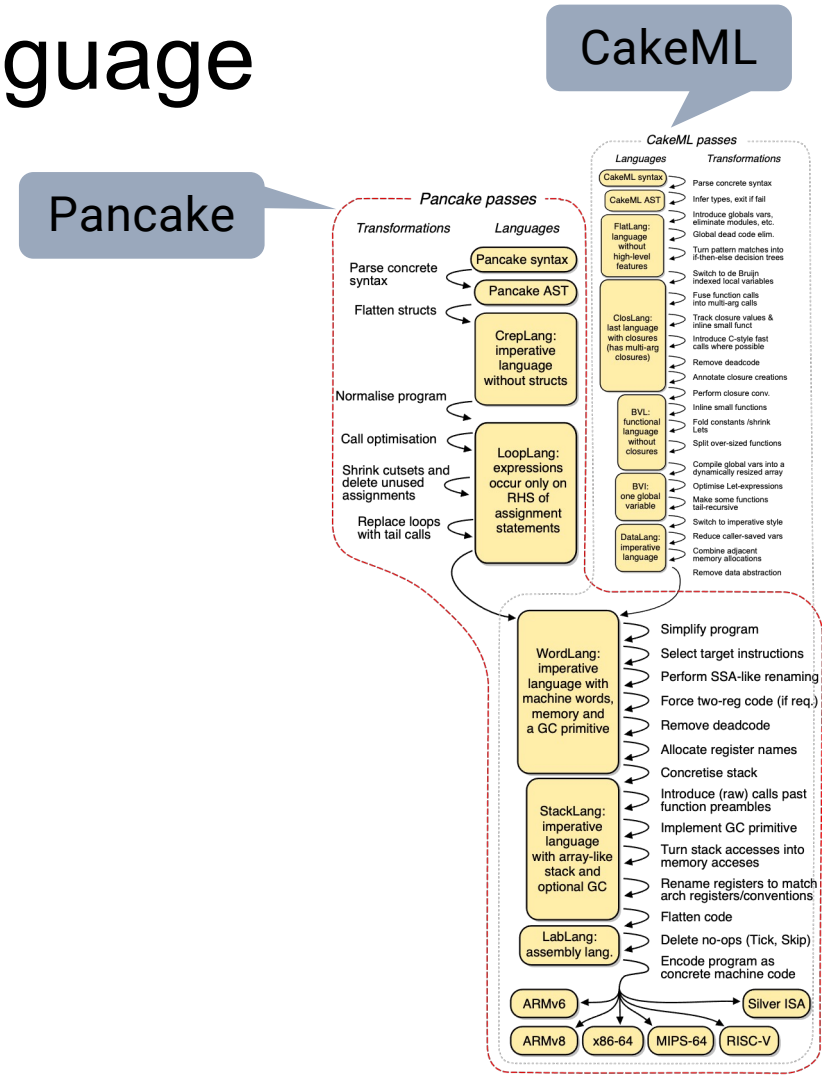


© Gernot Heiser 2019-24 – CC BY 4.0

Pancake: New Systems Language

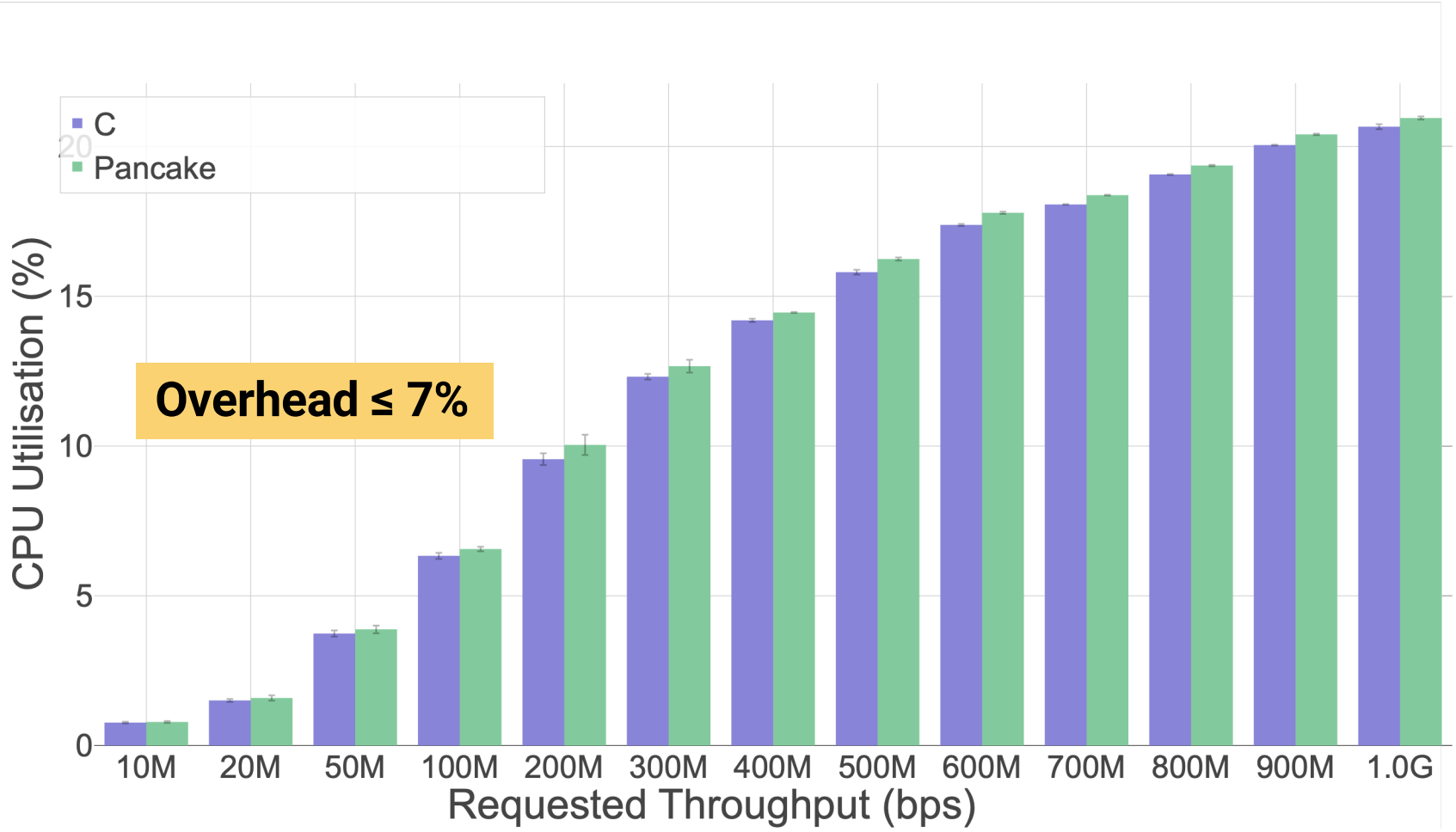
Approach:

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



© Gernot Heiser 2019-24 – CC BY 4.0

Performance: LionsOS Ethernet Driver

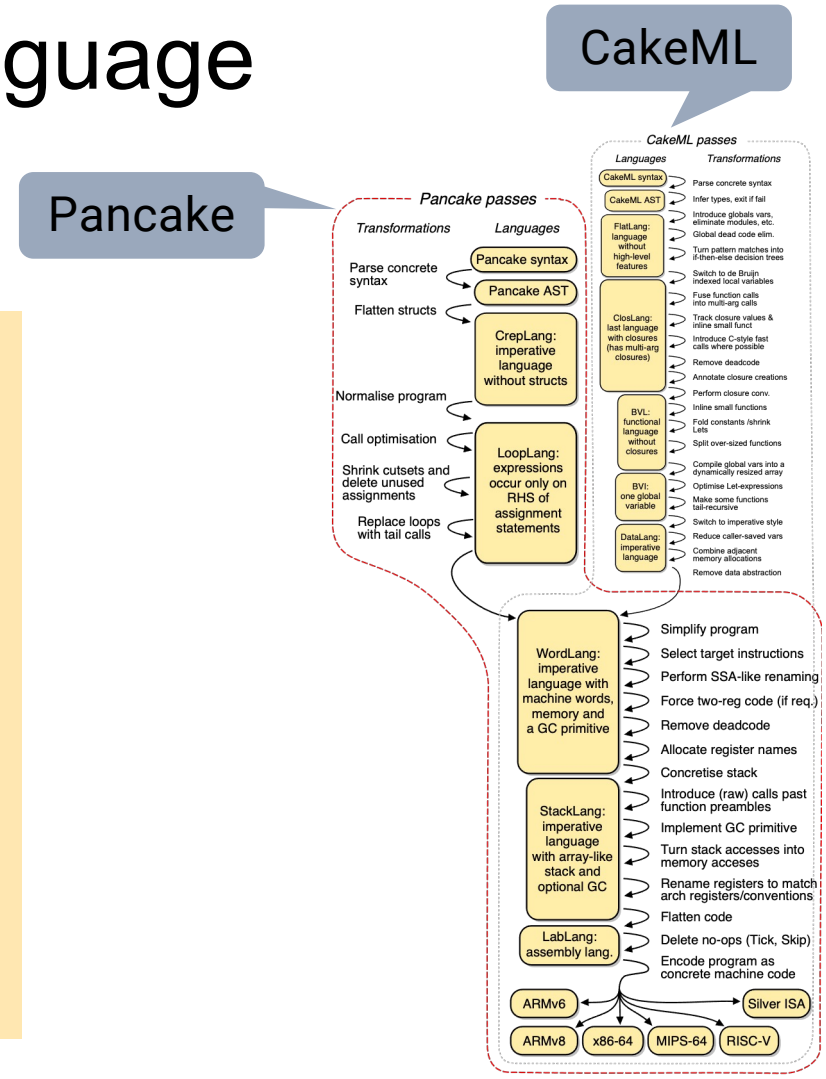


© Gernot Heiser 2019-24 – CC BY 4.0

Pancake: New Systems Language

Status:

- "Usable" rump language, sufficient for LionsOS
  - Implemented drivers & other components
  - Implemented & verified libmicrokit
  - Verified Ethernet driver
- **Verified compiler!**
- In progress:
  - Performance work (inlining, globals, …)
  - Engineering-friendly verification process
  - Remove need for C escapes
  - Verified export to SMT solvers
  - Decompilation from Pancake to HOL



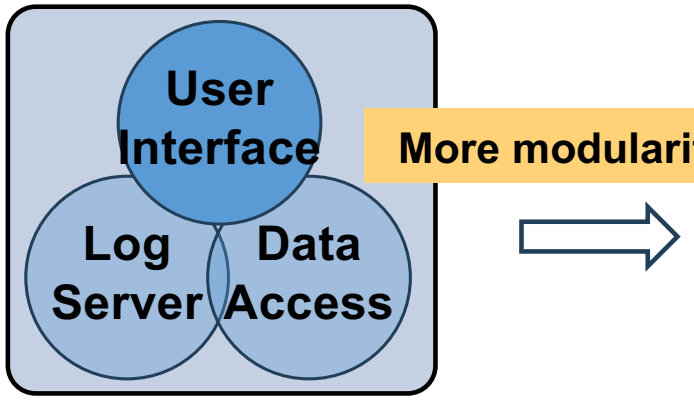
© Gernot Heiser 2019-24 – CC BY 4.0

Carrells: Microservices

© Gernot Heiser 2019-24 – CC BY 4.0

Microservices

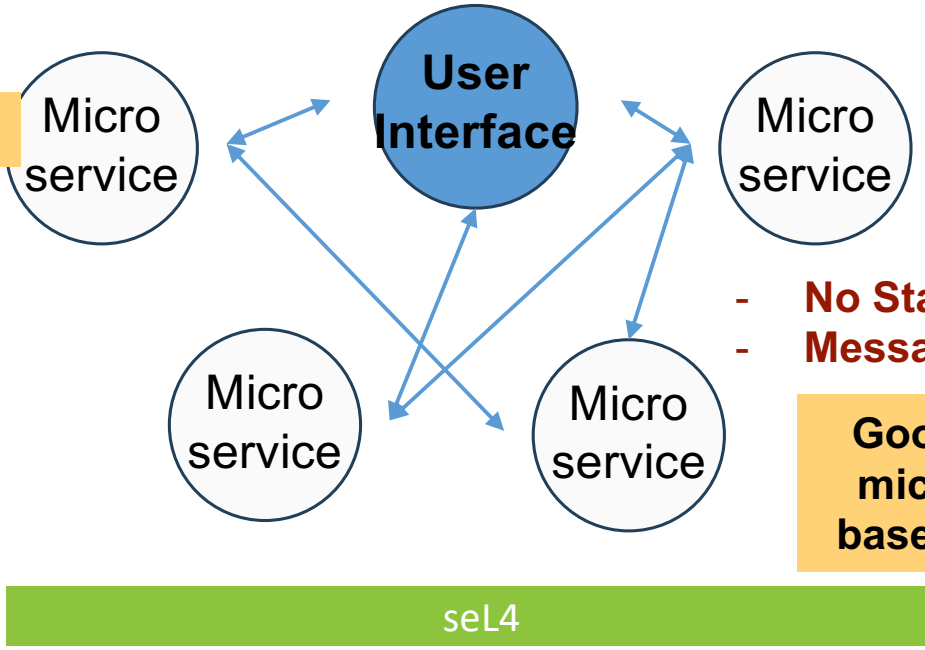
Monolithic Architecture



More modularity

- **Strong inter-dependencies due to shared state**
- **Difficult to scale or upgrade**

Microservice Architecture



- **No State Sharing**
- **Message-passing**

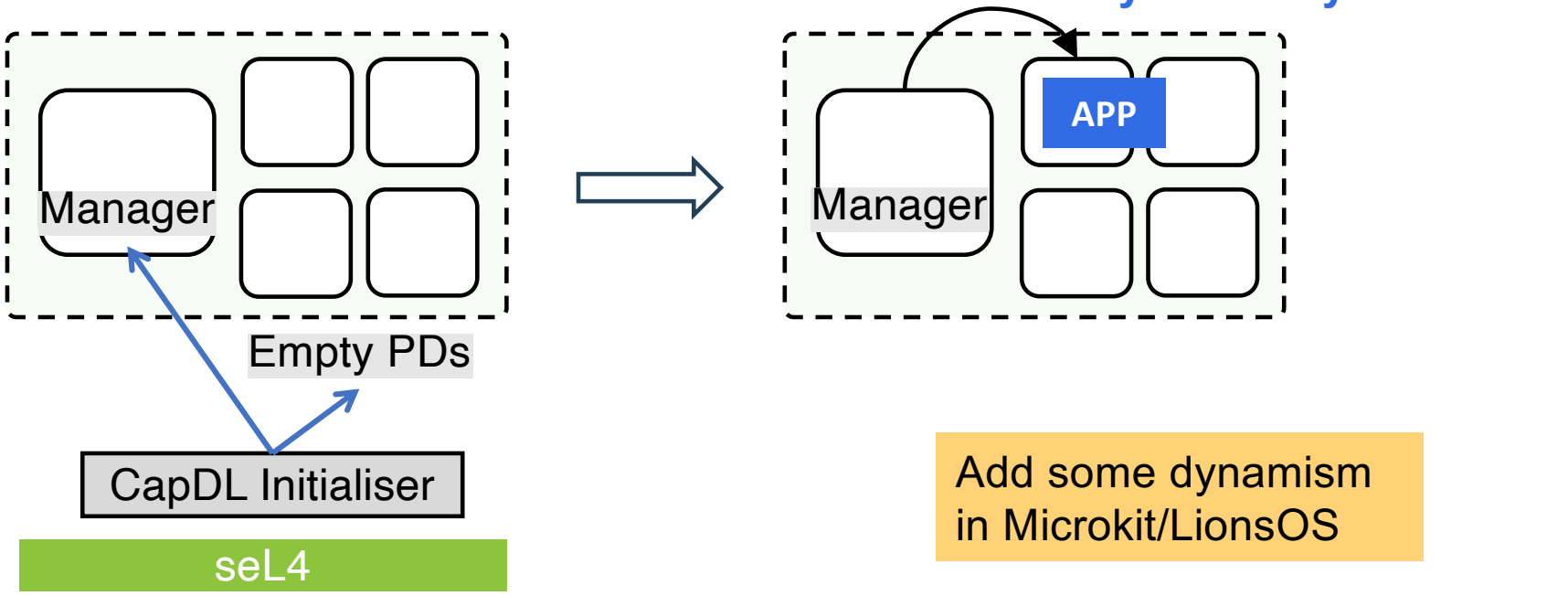
Good fit for a microkernel-based design?

© Gernot Heiser 2019-24 – CC BY 4.0



PDs as Sandboxes

- Partition resources at boot time
- Pre-provision PDs

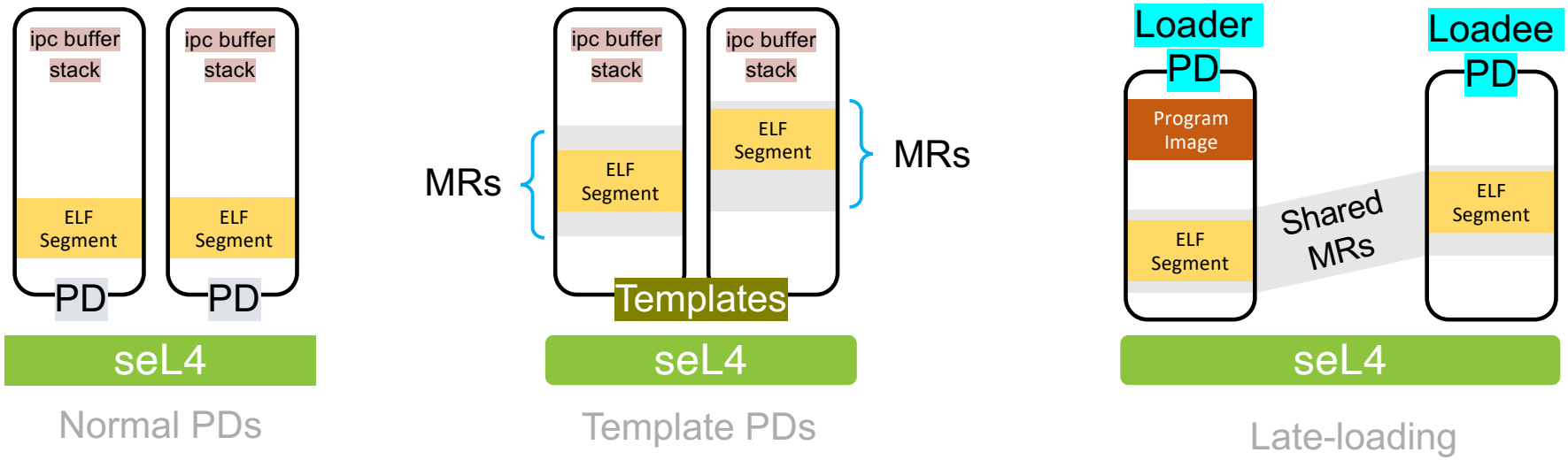


Add some dynamism in Microkit/LionsOS

© Gernot Heiser 2019-24 – CC BY 4.0

Loading a New Image

- **Template PD**
  - PDs without an image by default

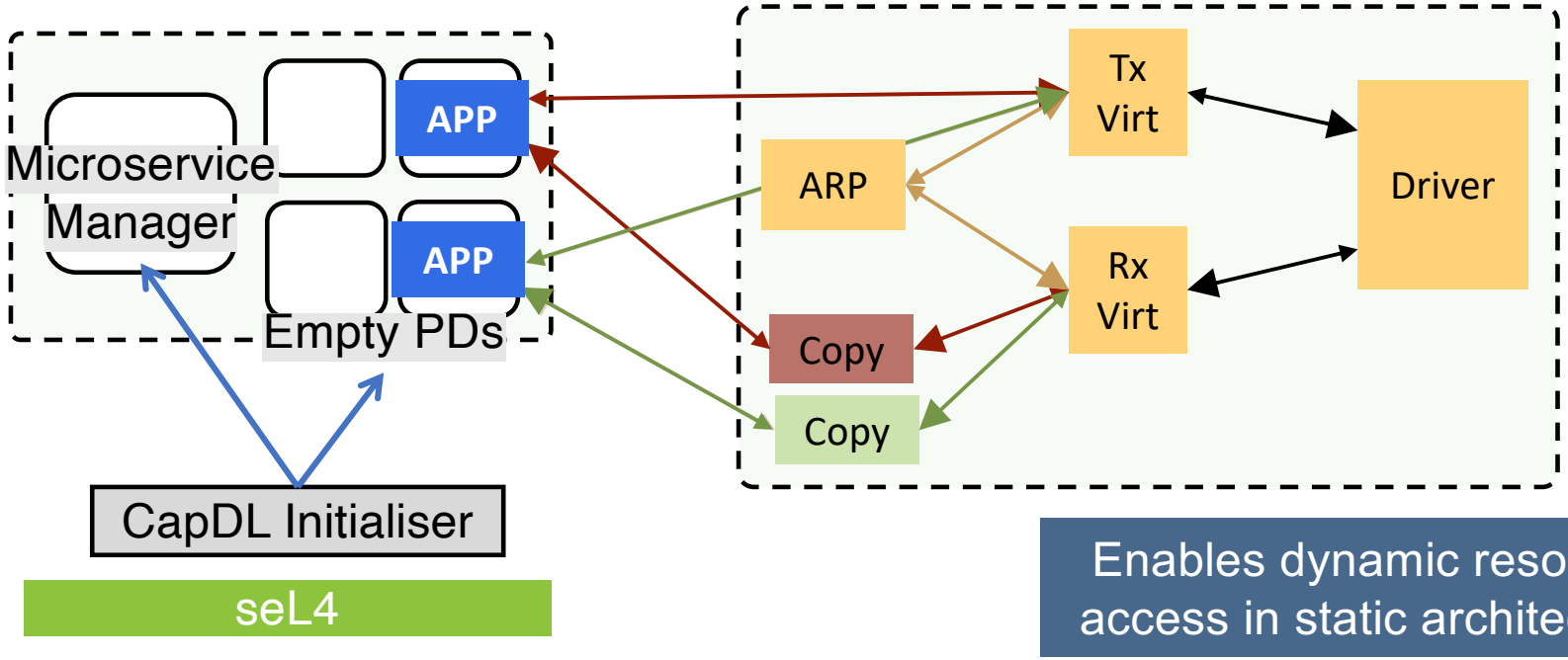


- Template PD needs a fault handler and a loader
  - The loader requires access to the cap of template PD's thread control block

© Gernot Heiser 2019-24 – CC BY 4.0

Resource Management

- Provision (ceiling of) connections at boot-time – MAC
- Establish authorised connections dynamically – DAC



Enables dynamic resource access in static architecture

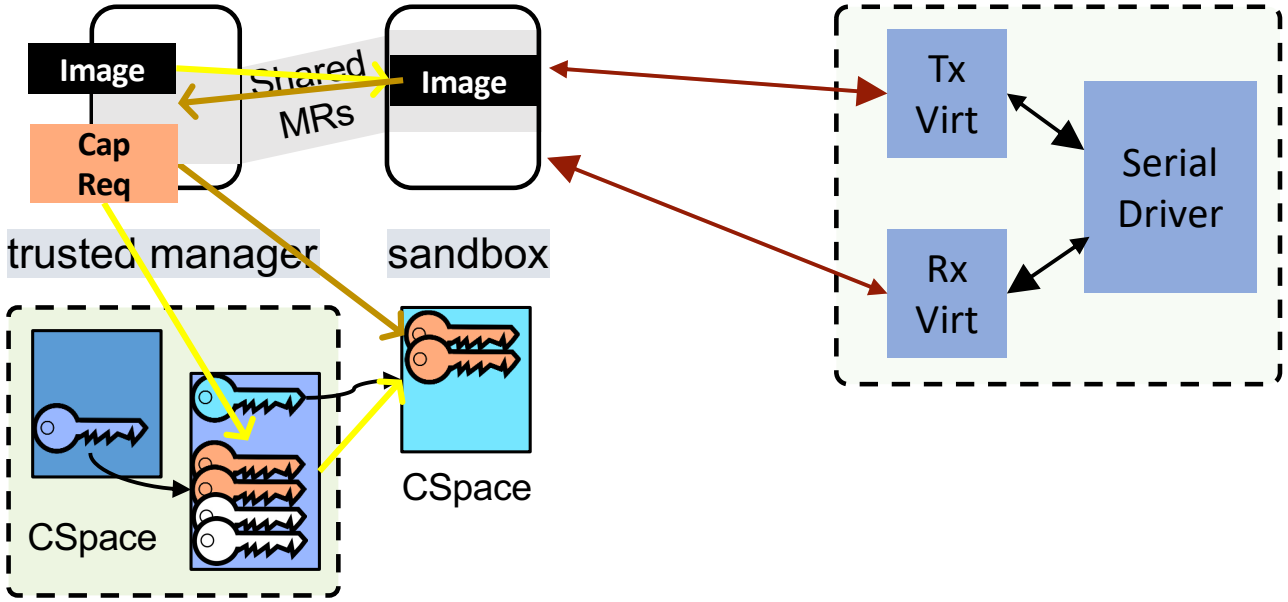
© Gernot Heiser 2019-24 – CC BY 4.0

Preliminary Results: Startup Latency

ISA	Platform	Startup-teardown	S-T (batch)
ARM64	Cortex-a55 @1.2Ghz	284 kcy (237 µs)	201 kcy (167 µs)
x86_64	Xeon® W-1250 @3.3Ghz	140 kcy (42 µs)	82 kcy (25 µs)

Add batched mappings

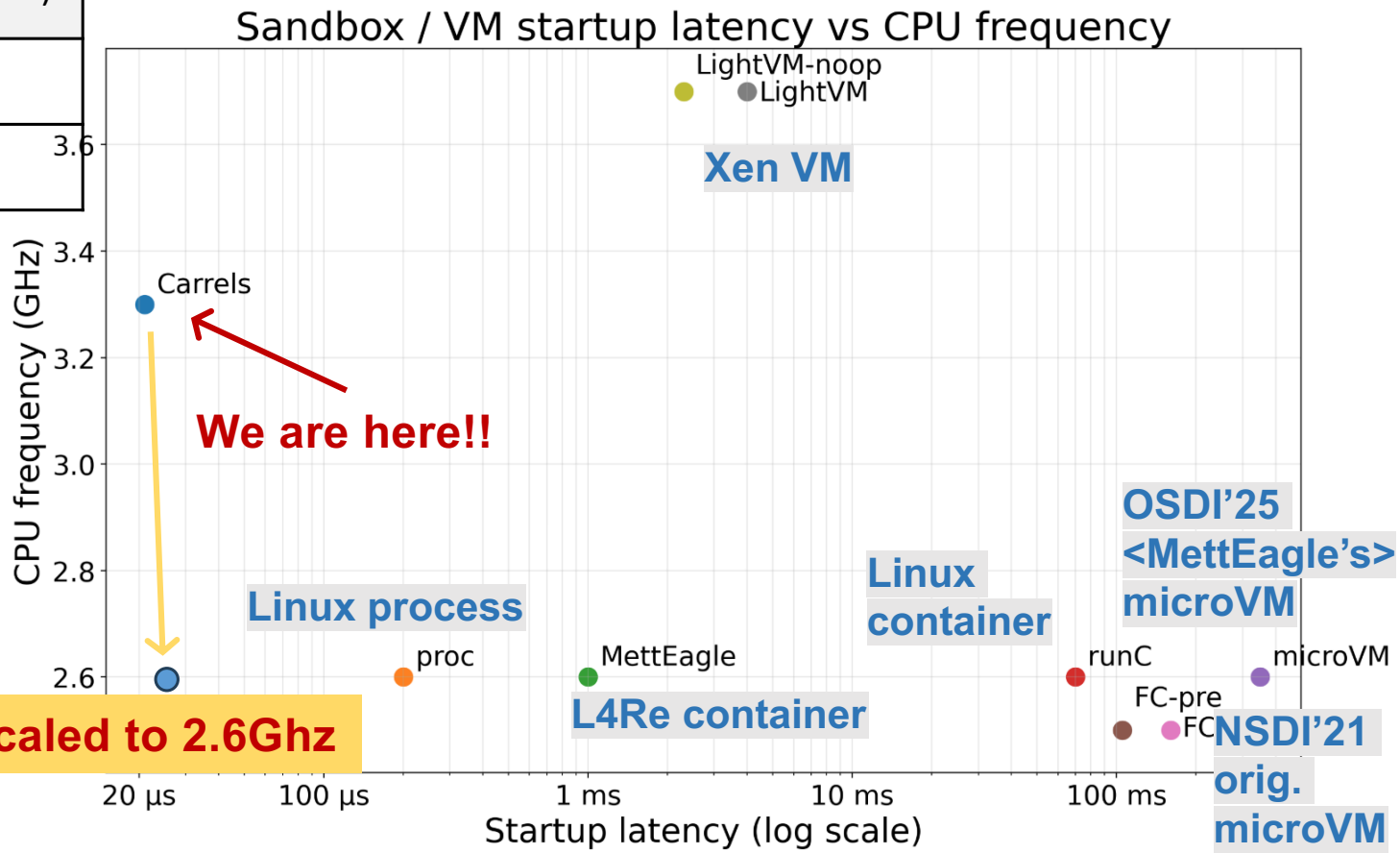
Payload setup
- Resource view
- (serial connection)
  - 34 frames to map
  - 1 channel to grant
- Image size
  - 156 KB in total
  - 16 KB to load



© Gernot Heiser 2019-24 – CC BY 4.0

Preliminary Results: Tear-Down Latency

ISA	Platf	S-T	S-T (batch)
ARM64	A55	237 µs	167 µs
x86_64	Xeon	42 µs	25 µs



Payload setup

- Resource view
- (serial connection)
  - 34 frames to map
  - 1 channel to grant
- Image size
  - 156 KB in total
  - 16 KB to load

© Gernot Heiser 2019-24 – CC BY 4.0

Programming API for microservices

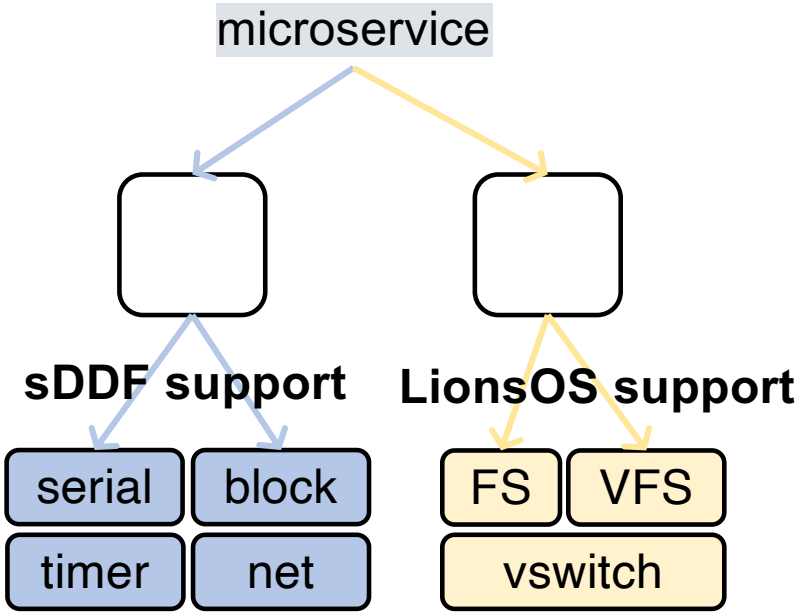
- **Native API**

- Flexible for building different OS personalities
- Not familiar to developers

libmicrokit

```
void init(void)
{
    microkit_dbg_puts("hello, world\n");
}

void notified(microkit_channel ch)
{
}
```



© Gernot Heiser 2019-24 – CC BY 4.0

Unikraft

RESEARCH-ARTICLE | OPEN ACCESS |

Unikraft: fast, specialized unikernels the easy way

Authors: Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, + 5 Authors

Info & Claims

EuroSys '21: Proceedings of the Sixteenth European Conference on Computer Systems • Pages 376 - 394
https://doi.org/10.1145/3447786.3456248

Applications	NGINX, SQLite, Redis, memcached, Click modular router, lighttpd (ongoing).
Frameworks	Intel DPDK, TensorFlow Lite, PyTorch.
Compiled Languages	C/C++, Go, Web Assembly (WAMR), Lua, Java/OpenJDK (ongoing), Rust (ongoing)
Interpreted Languages	Python, Micropython, Ruby, JavaScript/v8 (ongoing).

Table 3. Applications, frameworks and languages currently supported by Unikraft.

© Gernot Heiser 2019-24 – CC BY 4.0

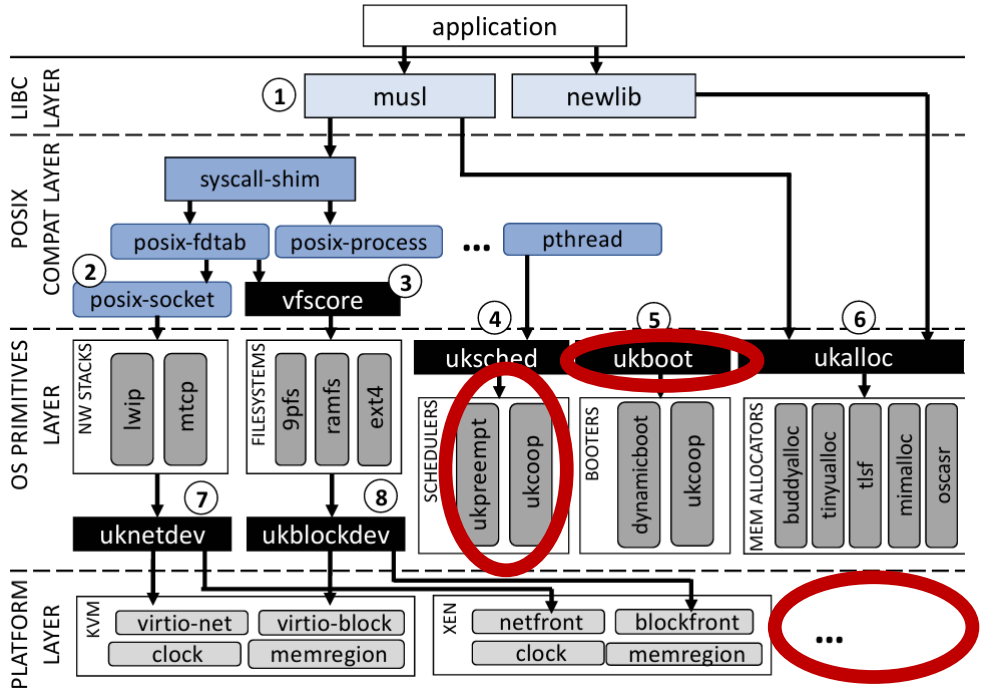
Experimental Carrels support for Unikraft

- lib/uksched, lib/ukschedcoop: +33 LoC
- lib/ukpcpuvar/arch/arm64: +38,-9 LoC

Patch needed

New

- plat/carrels: +1,101 LoC
- drivers/sddf: +530 LoC
  - network
  - timer
  - serial



plat/carrels

© Gernot Heiser 2019-24 – CC BY 4.0

But I Want A **Real** OS!



Djawula

© Gernot Heiser 2019-24 – CC BY 4.0



What's An OS?

Security

- Access control

Resource Mgmt

- Memory
- CPU
- Storage
- NW bandwidth

Services

- Files
- Networking

Dynamic!

Reuse?

© Gernot Heiser 2019-24 – CC BY 4.0



Performance Cost Of A Dynamic OS

- More complexity, larger code size
- Double book-keeping, multiple server invocations
- Higher startup times due to dynamic resource allocation
- Resource revocation may require indirection
- “Universal” policies are complex & costly

Might affect cache footprint?

IPC overheads in the noise

fork() will be the test!

seL4 caps can be revoked without

Do we need them?

© Gernot Heiser 2019-24 – CC BY 4.0



Do We Need “Universal” Policies?

Source of massive complexity

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

'80s model of computer use!

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

© Gernot Heiser 2019-24 – CC BY 4.0



Djawula: PoC of General-Purpose OS

Idea: Maximise re-use of LionsOS artefacts

- Microkit abstraction:
  - *Generalise* somewhat, possibly multiple versions of library
  - *Keep* event model, single-threaded
- Drivers:
  - *Keep* drivers
  - *Generalise* virtualisers (variable # clients)
  - *Add* dynamic IOMMU mappings
- I/O:
  - *Keep* asynchronous native API
  - *Add* access control
  - *Add* resource donation

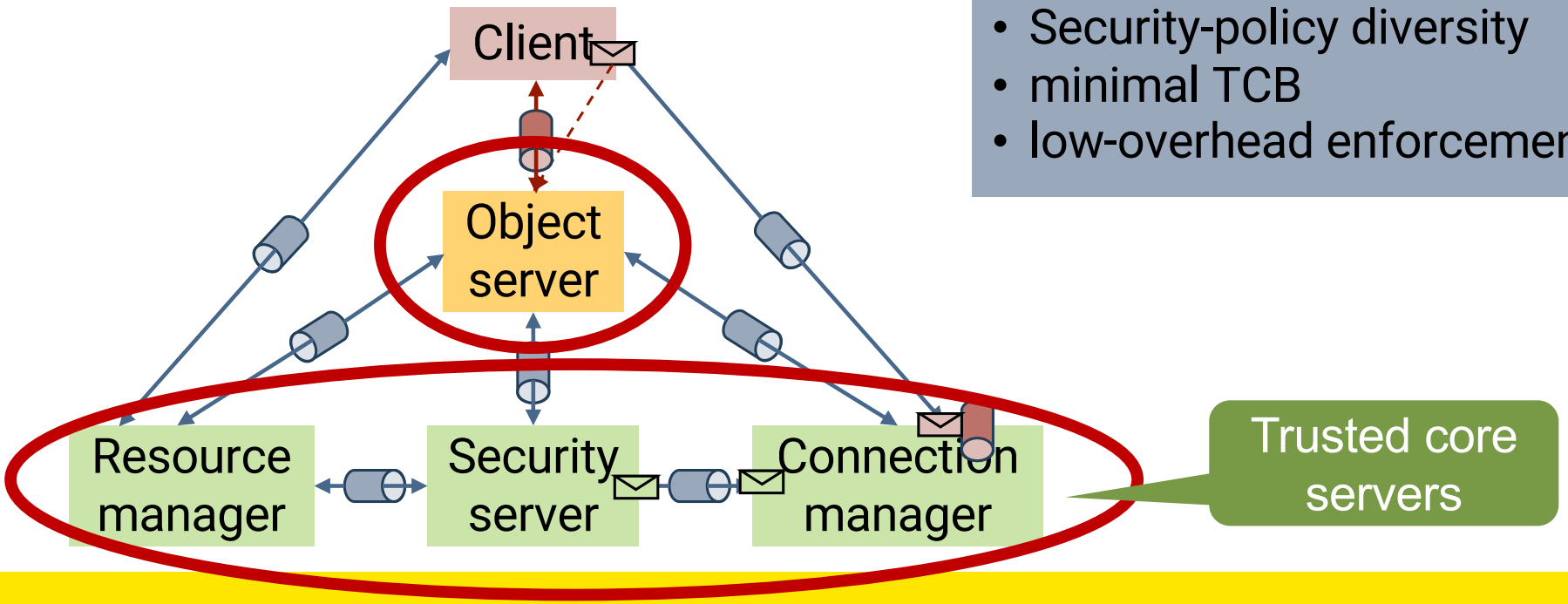
© Gernot Heiser 2019-24 – CC BY 4.0



Mandatory Access Control

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

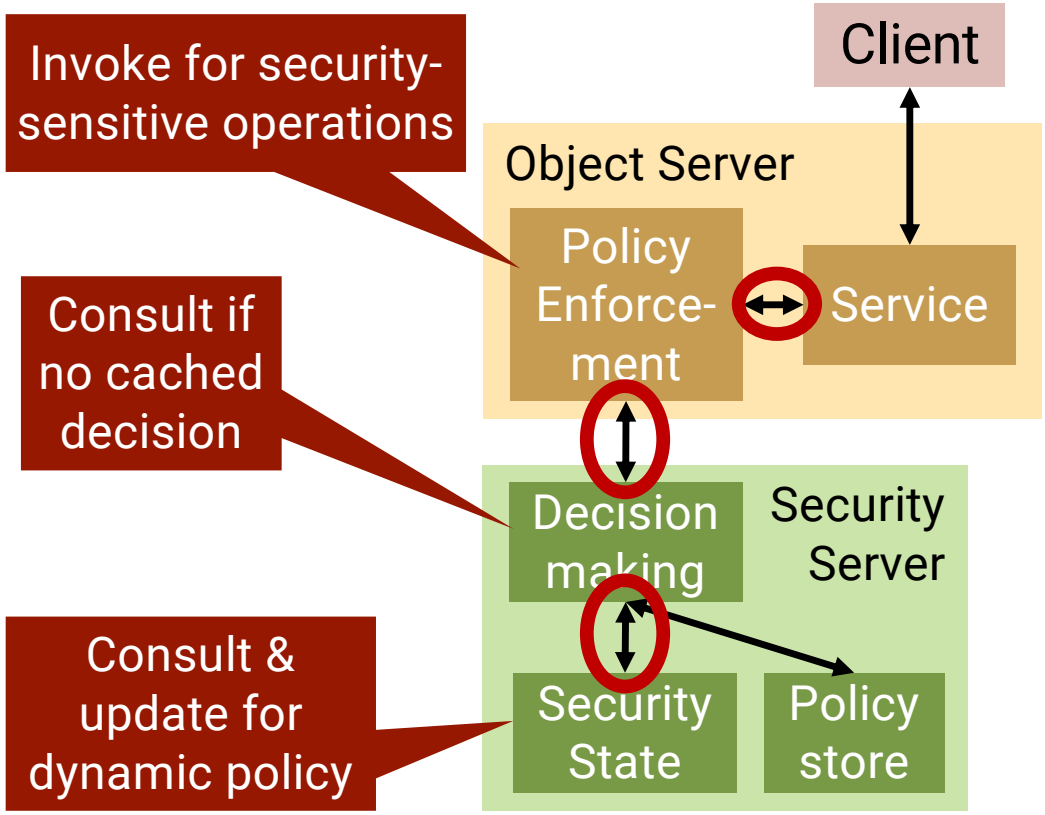
Requires:
- mandatory security-policy enforcement (*reference monitor*)
- Security-policy diversity
- minimal TCB
- low-overhead enforcement



© Gernot Heiser 2019-24 – CC BY 4.0



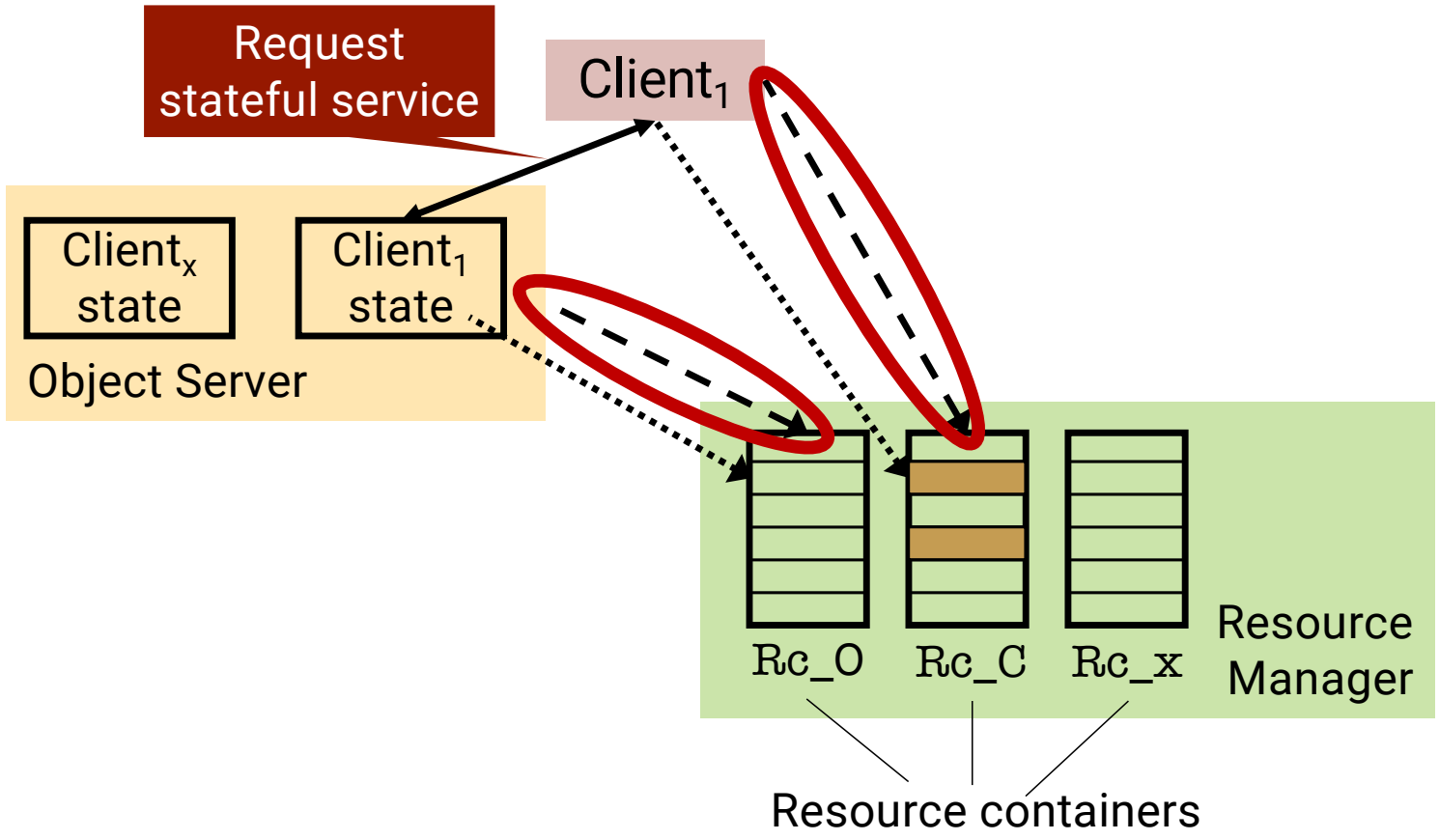
Core Idea: Dynamic Enforcement



© Gernot Heiser 2019-24 – CC BY 4.0



Core Idea: Resource Donation



© Gernot Heiser 2019-24 – CC BY 4.0



Djawula Status

- Rebooted after years of hibernation:
- Core protocols established
- Core servers being re-implemented cleanly
  - Naming, connection, security servers
- To come:
  - Sample security policies
  - File servers
  - Resource donation design
- Aim: Demonstrators on
  - Server cluster
  - Laptop

© Gernot Heiser 2019-24 – CC BY 4.0

Time Protection

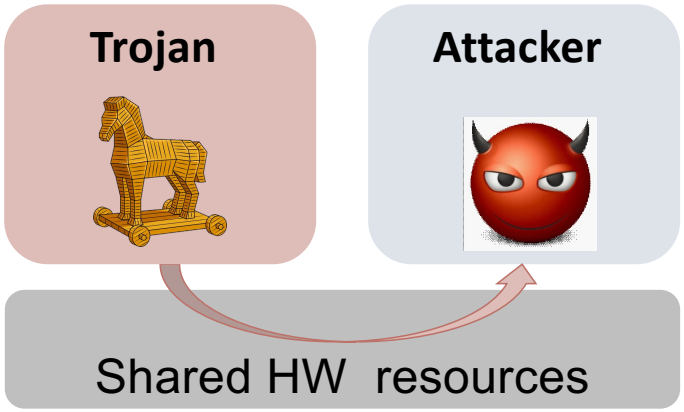
Principled Prevention of Microarchitectural Timing Channels

© Gernot Heiser 2019-24 – CC BY 4.0

Refresh: Covert Timing Channels

- Created by contention for shared resource whose effect on timing can be monitored
  - Cache, network bandwidth, CPU load...

Usually based on some hidden state (e.g. caches)
⇒ timing/storage channel distinction is not deep!

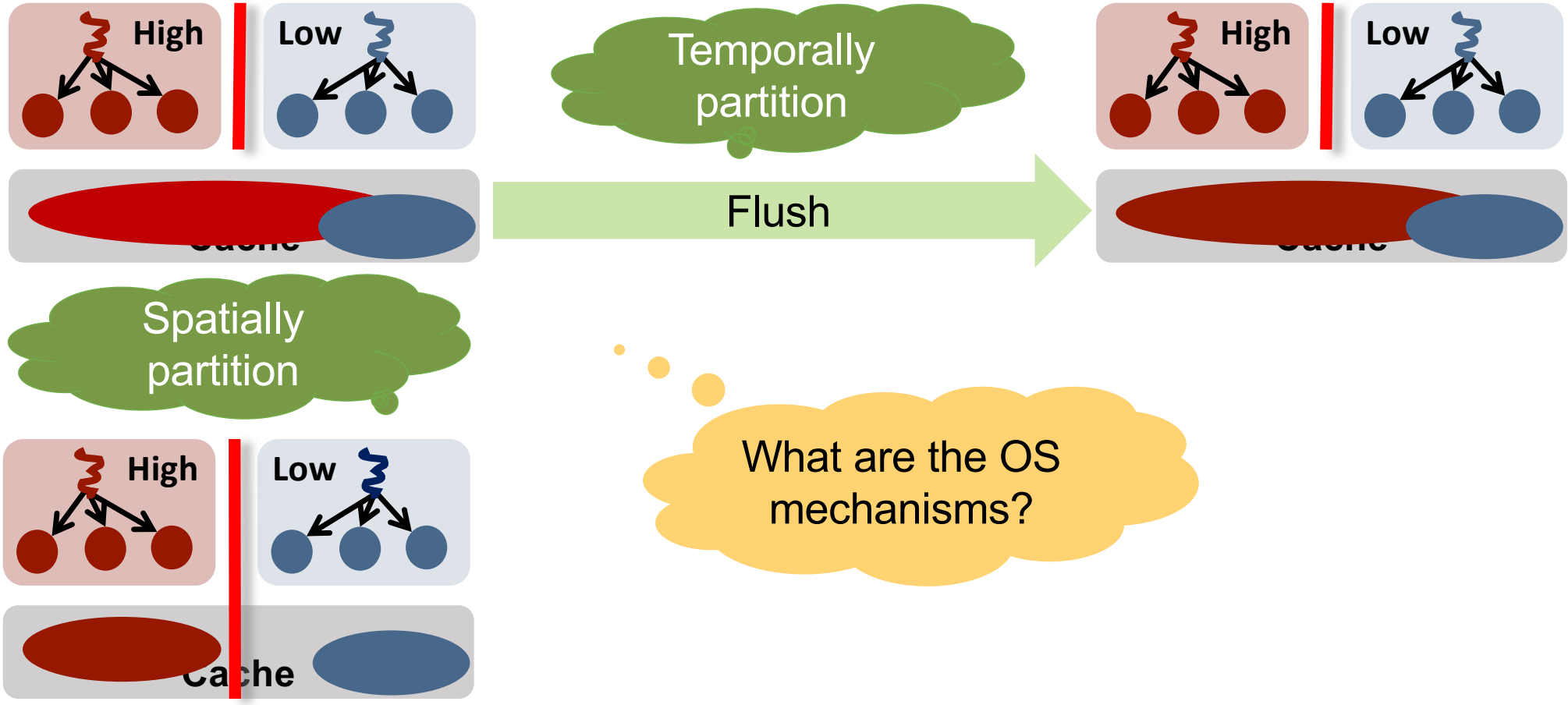


OS protection mechanisms:

- *Memory protection* for spatial isolation
- *Time protection* for temporal isolation

© Gernot Heiser 2019-24 – CC BY 4.0

Time Protection: No Sharing of HW State

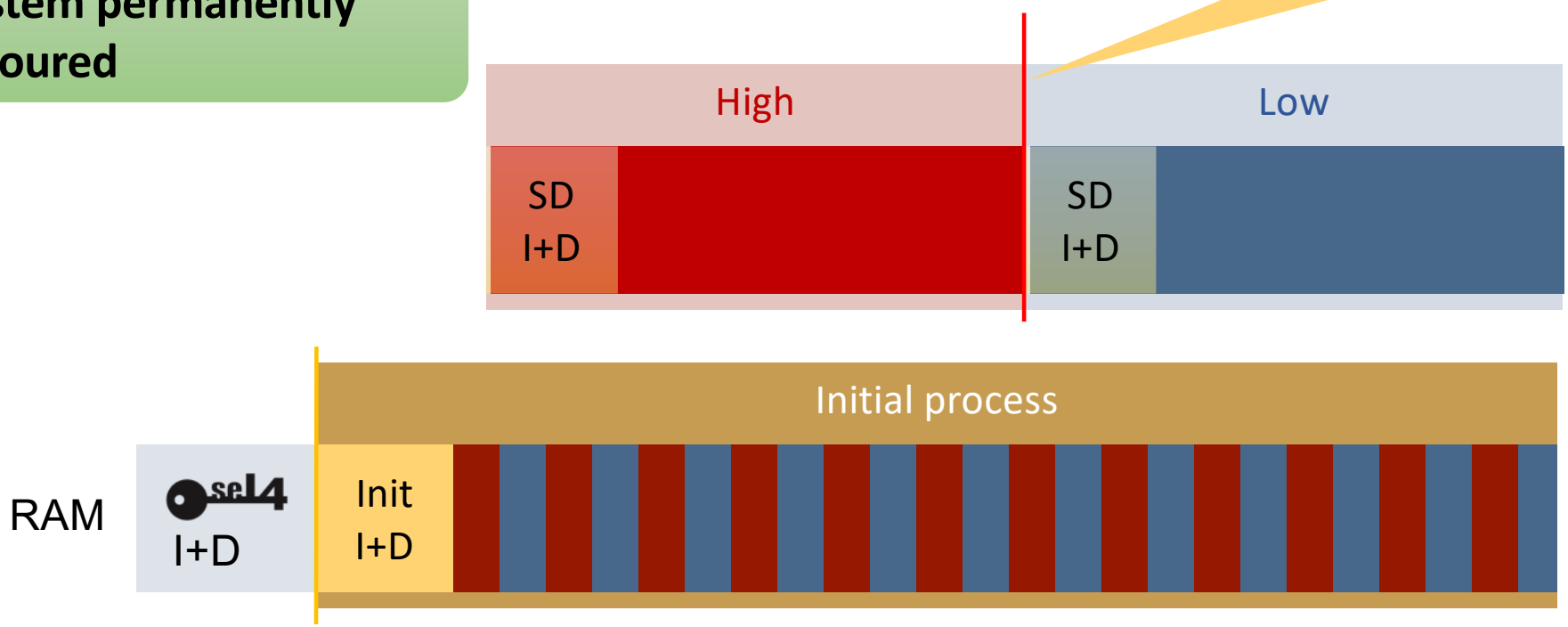


© Gernot Heiser 2019-24 – CC BY 4.0

Spatial Partitioning: Cache Colouring

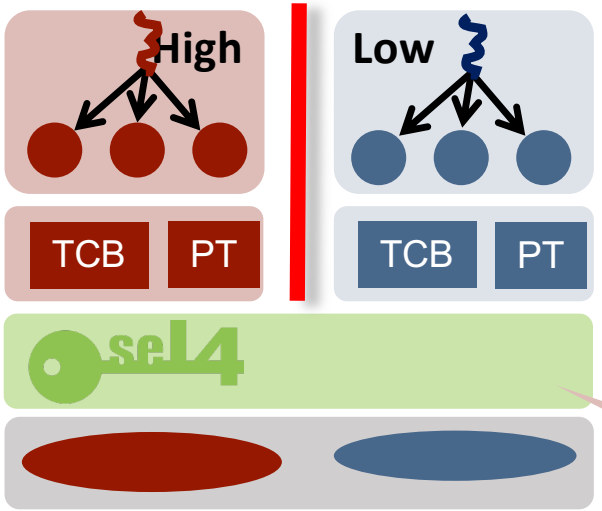
System permanently coloured

Partitions restricted to coloured memory



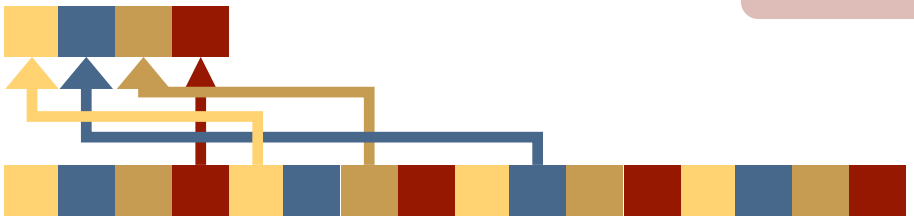
© Gernot Heiser 2019-24 – CC BY 4.0

Spatial Partitioning: Cache Colouring



- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours kernel memory

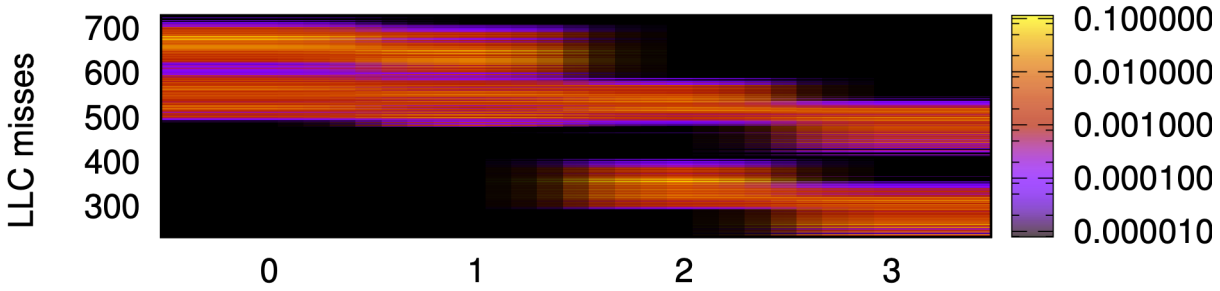
Shared kernel image



© Gernot Heiser 2019-24 – CC BY 4.0

Channel Through Kernel Code

Raw channel



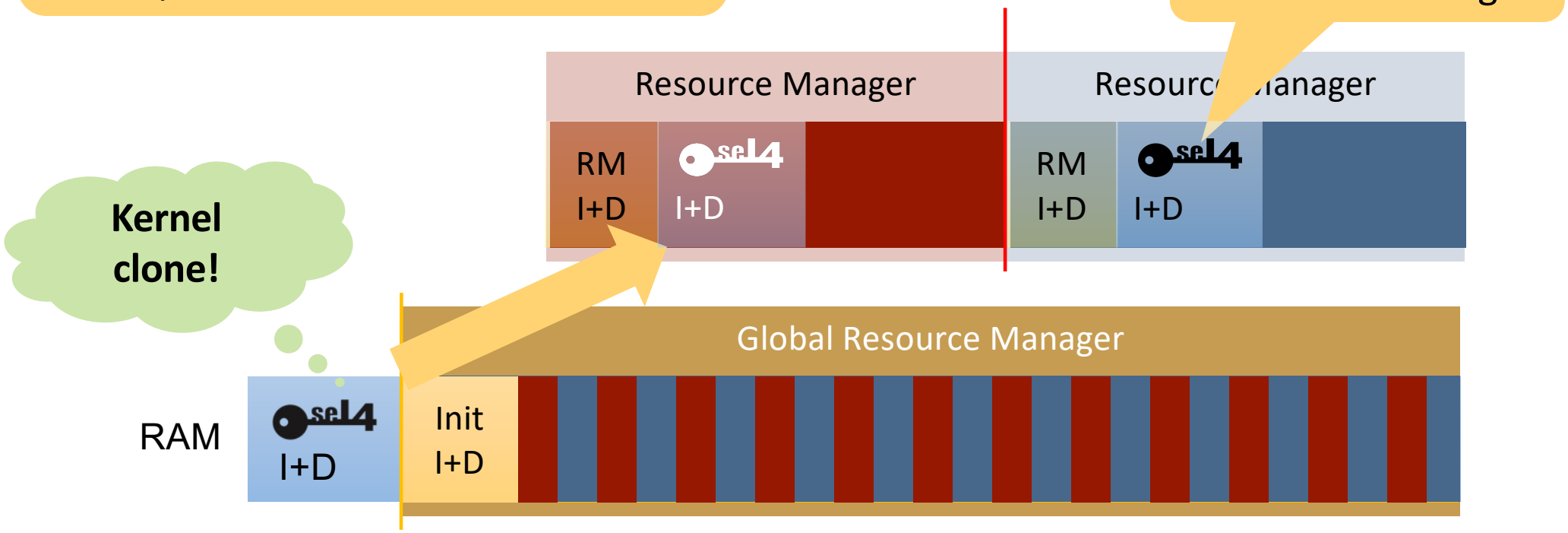
Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)

© Gernot Heiser 2019-24 – CC BY 4.0

Colouring the Kernel

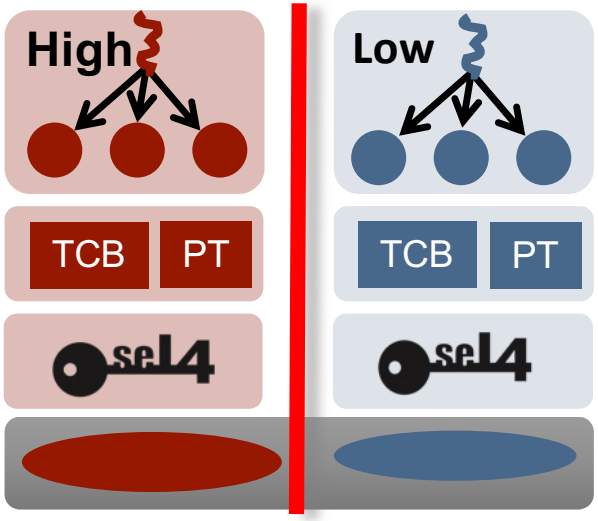
Remaining shared kernel data:
- Scheduler queue array & bitmap
- Few pointers to current thread state

Each partition has own kernel image



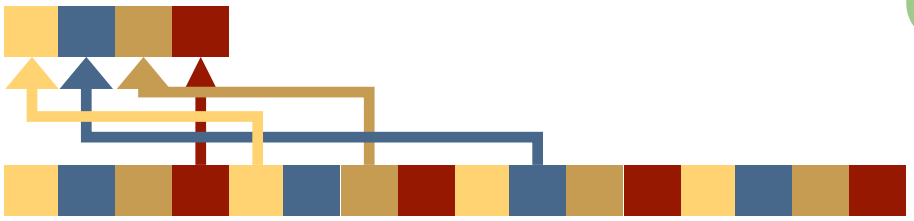
© Gernot Heiser 2019-24 – CC BY 4.0

Spatial Partitioning: Cache Colouring



- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory
  ⇒ colouring userland colours kernel memory
- Per-partition kernel image to colour kernel

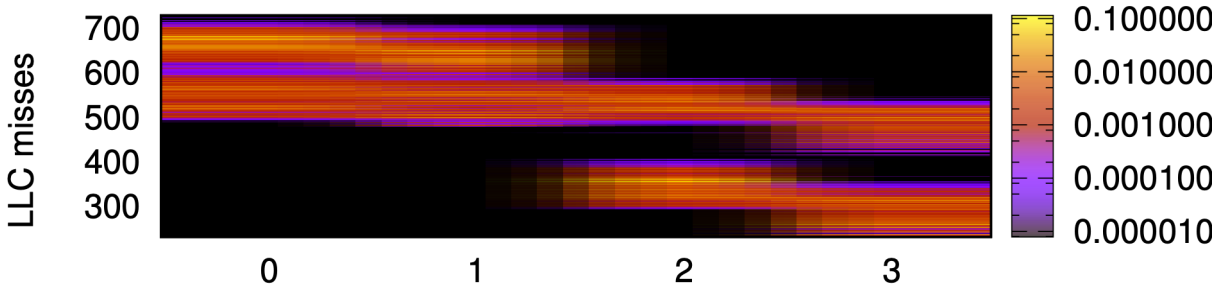
Must ensure deterministic access to remaining shared kernel state!



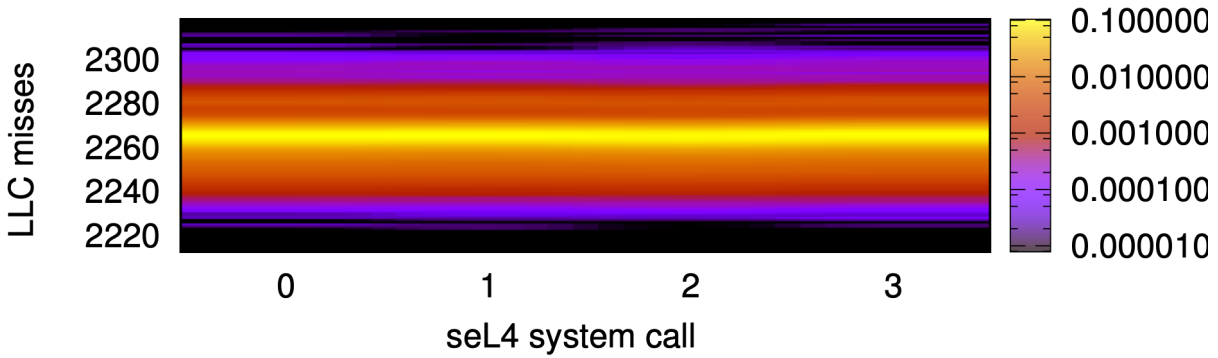
© Gernot Heiser 2019-24 – CC BY 4.0

Channel Through Kernel Code

Raw channel



Channel with cloned kernel



© Gernot Heiser 2019-24 – CC BY 4.0



Temporal Partitioning: Flush on Switch

Must remove any history dependence!

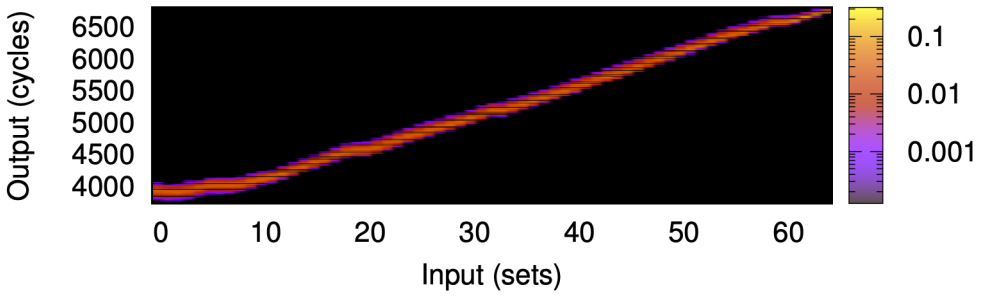
2. Switch user context
3. Flush on-core state

6. Reprogram timer
7. return

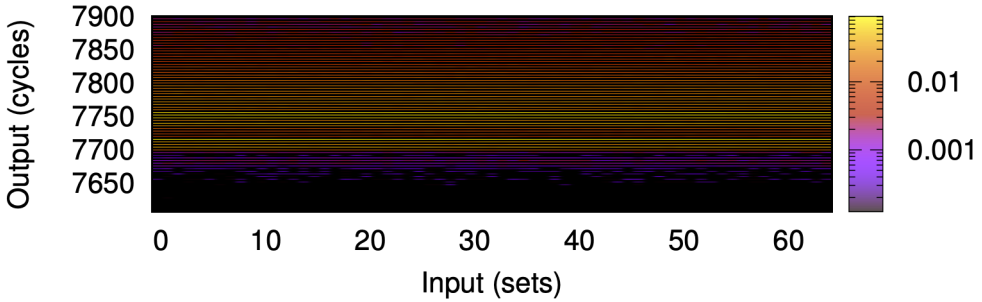
© Gernot Heiser 2019-24 – CC BY 4.0

D-Cache Channel

Raw channel



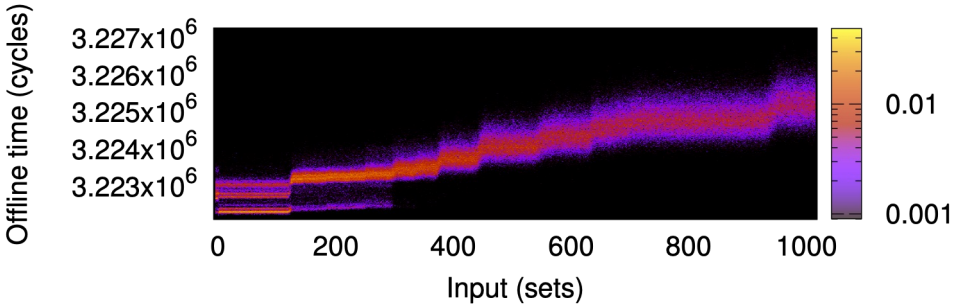
Channel with flushing



© Gernot Heiser 2019-24 – CC BY 4.0

Flush-Time Channel

Raw channel



© Gernot Heiser 2019-24 – CC BY 4.0

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. Touch all shared data needed for return
5. while (T_0+WCET < current_time()) ;
6. Reprogram timer
7. return

Latency depends on prior execution!

Time padding to remove dependency

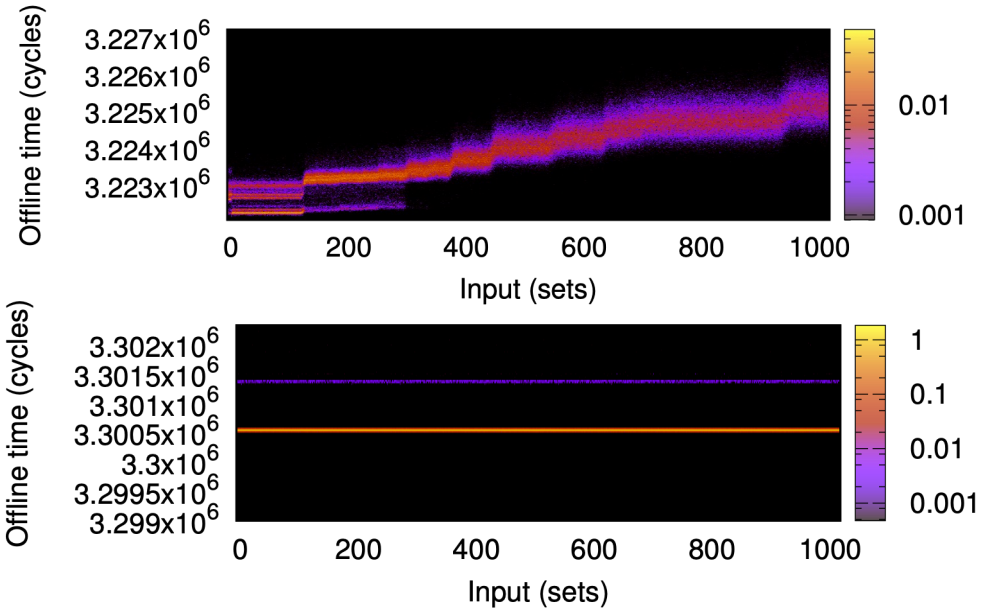
Ensure deterministic execution

© Gernot Heiser 2019-24 – CC BY 4.0

Flush-Time Channel

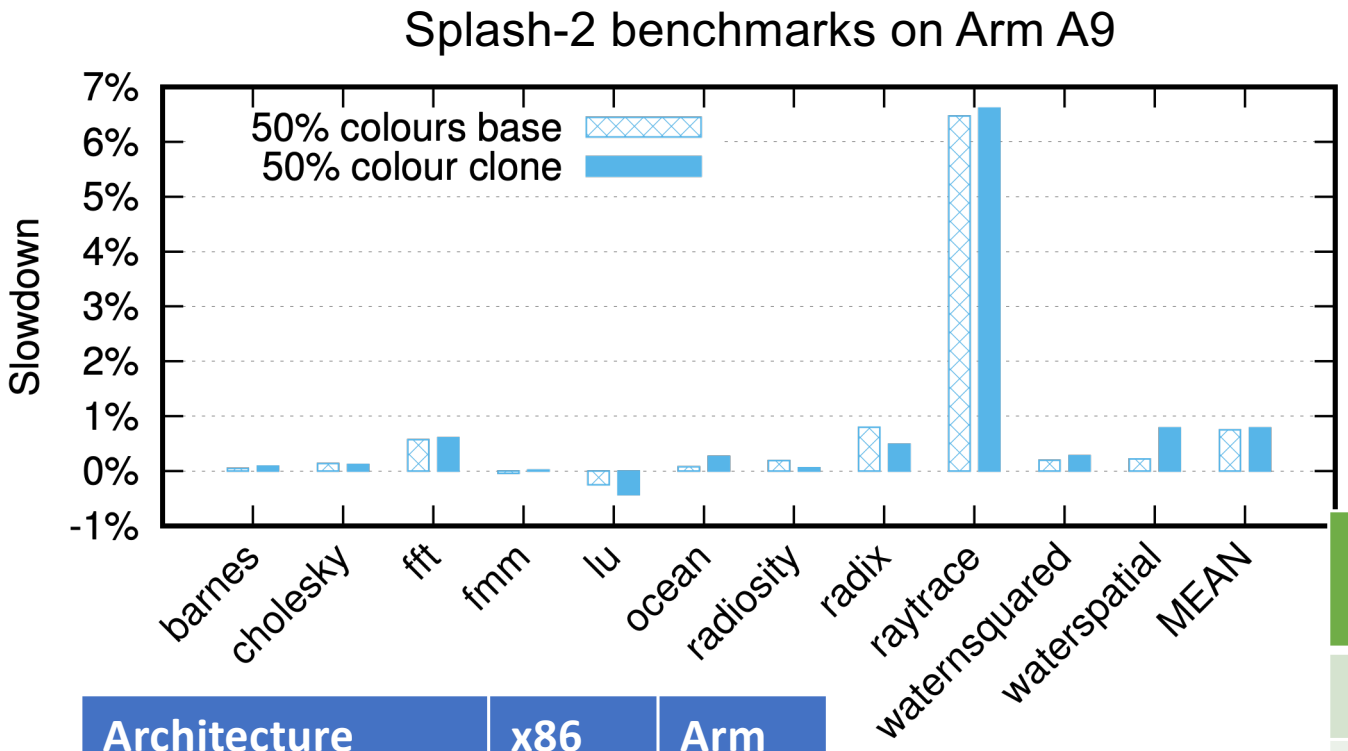
Raw channel

Channel with deterministic flushing



© Gernot Heiser 2019-24 – CC BY 4.0

Performance Impact of Colouring



Architecture	x86	Arm
Mean slowdown	3.4%	1.1%

- Overhead mostly low
- Not evaluated is cost of not using super pages

[Ge et al., EuroSys'19]

Arch	seL4 clone	Linux fork+exec
x86	79 µs	257 µs
Arm	608 µs	4,300 µs

© Gernot Heiser 2019-24 – CC BY 4.0



Temporal Partitioning: Flush State

Must remove any history dependence!

1. T_0 = current_time()
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while (T_0+WCET < current_time()) ;
6. Reprogram timer
7. return

Problem: Processors do *not* provide mechanisms for resetting all microarchitectural state!

© Gernot Heiser 2019-24 – CC BY 4.0

A New HW/SW Contract

For all shared microarchitectural resources:

aISA: augmented ISA

1. Resource must be spatially partitionable or flushable
2. Concurrently shared resources must be spatially partitioned
3. Resource accessed solely by virtual address must be flushed and not concurrently accessed

Cannot share HW threads across security domains!

4. Mechanisms must be sufficiently specified for OS to partition or reset
5. Mechanisms must be constant time, or of specified, bounded latency
6. Desirable: OS should know if resettable state is derived from data, instructions, data addresses or instruction addresses

[Ge et al., APSys'18]

© Gernot Heiser 2019-24 – CC BY 4.0



RISC-V To The Rescue: fence.t

fence.t instruction:

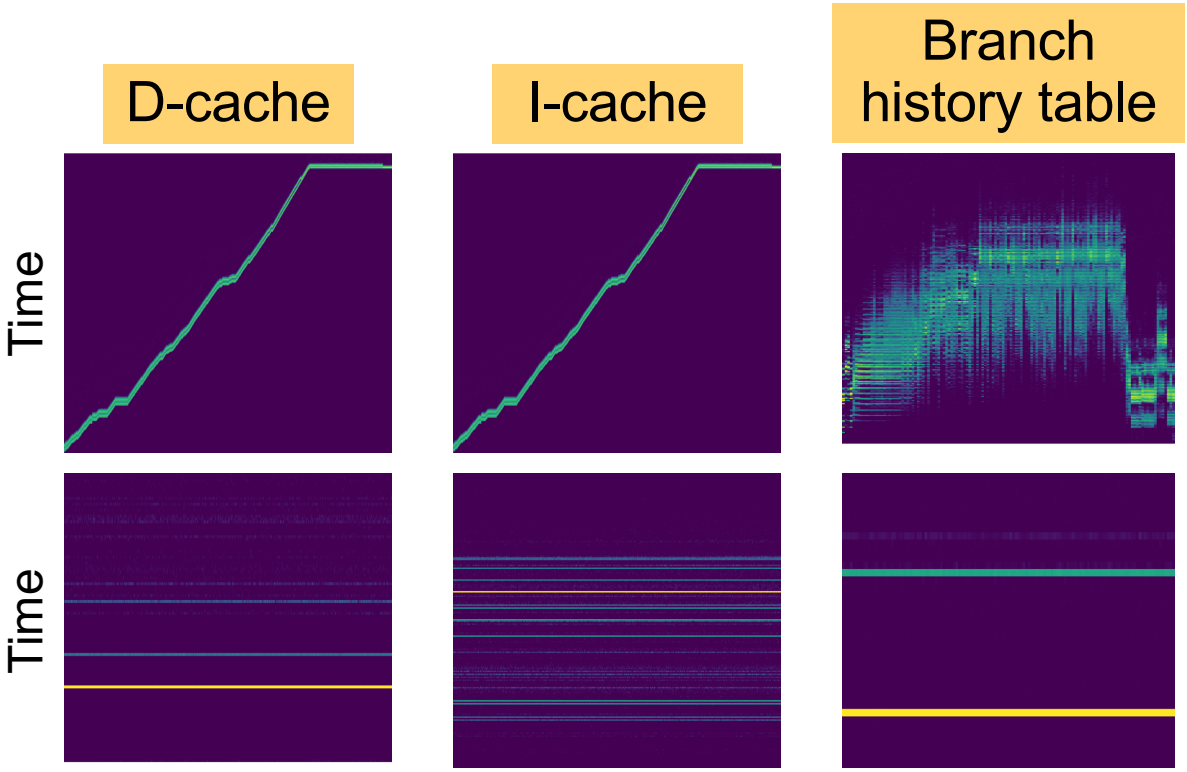
- Flush d-cache
- Microreset: Reset all flip-lops that are **not** part of architected state



- Prototyped on in-order (CVA6) and OoO (C910) RISC-V processors
- Latency bounded by d-cache flush
- HW cost in the noise

© Gernot Heiser 2019-24 – CC BY 4.0

fence.t Instruction on C910



Defeats all known attacks!

Wistoff et al, IEEE-TC'22
Wistoff et al, ApplePies'24

© Gernot Heiser 2019-24 – CC BY 4.0



Can Time Protection Be Verified?

1. Correct treatment of spatially partitioned state:
   - Need hardware model that identifies all such state (augmented ISA)
   - To prove:
     No two domains can access the same physical state

Functional property!

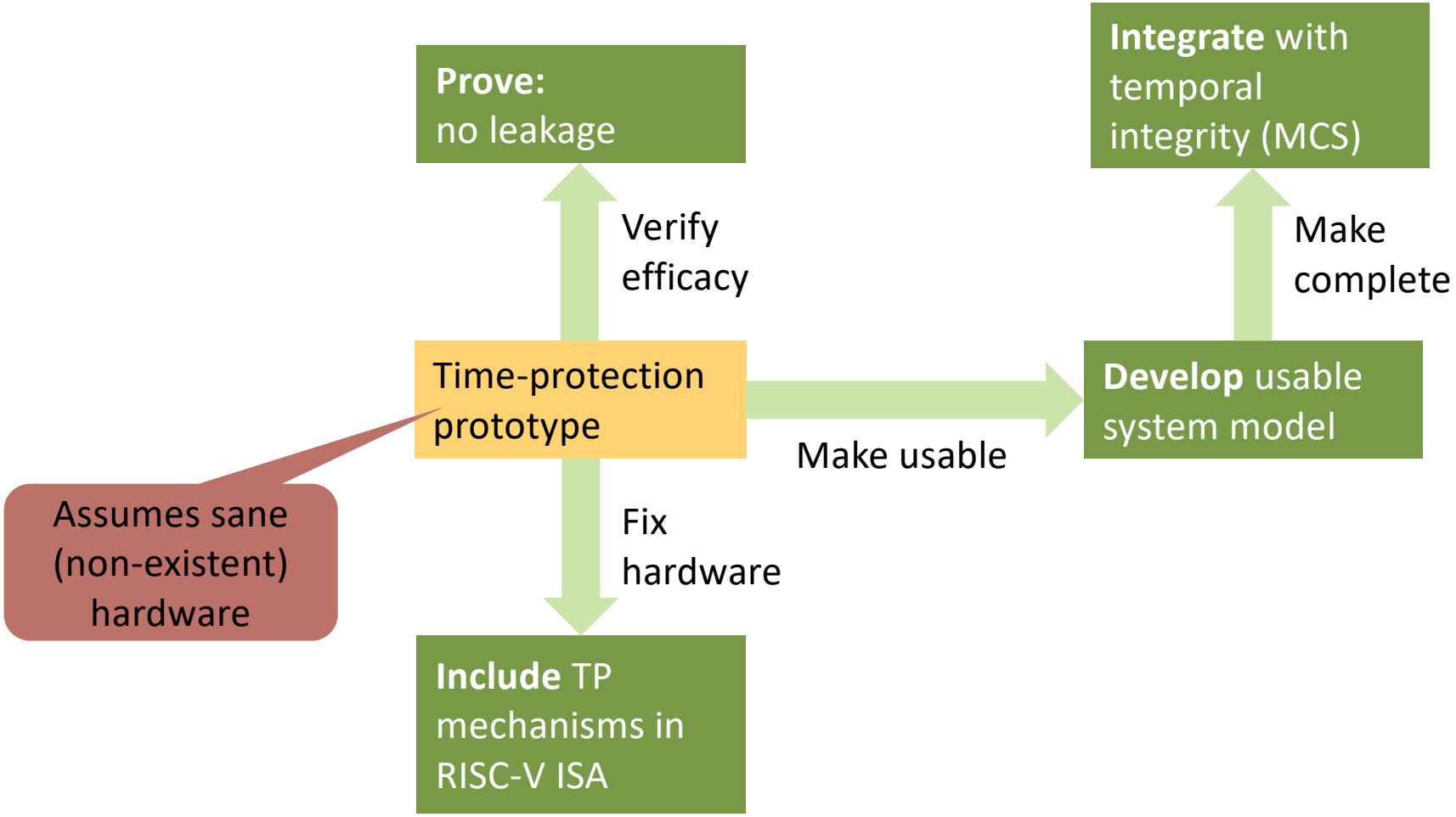
Transforms timing channels into storage channels!

2. Correct flushing of time-shared state
   - Not trivial: eg proving all cleanup code/data are forced into cache after flush
     - Needs an actual cache model
   - Even trickier: need to prove padding is correct
     - … without explicitly reasoning about time!

Functional property!

© Gernot Heiser 2019-24 – CC BY 4.0

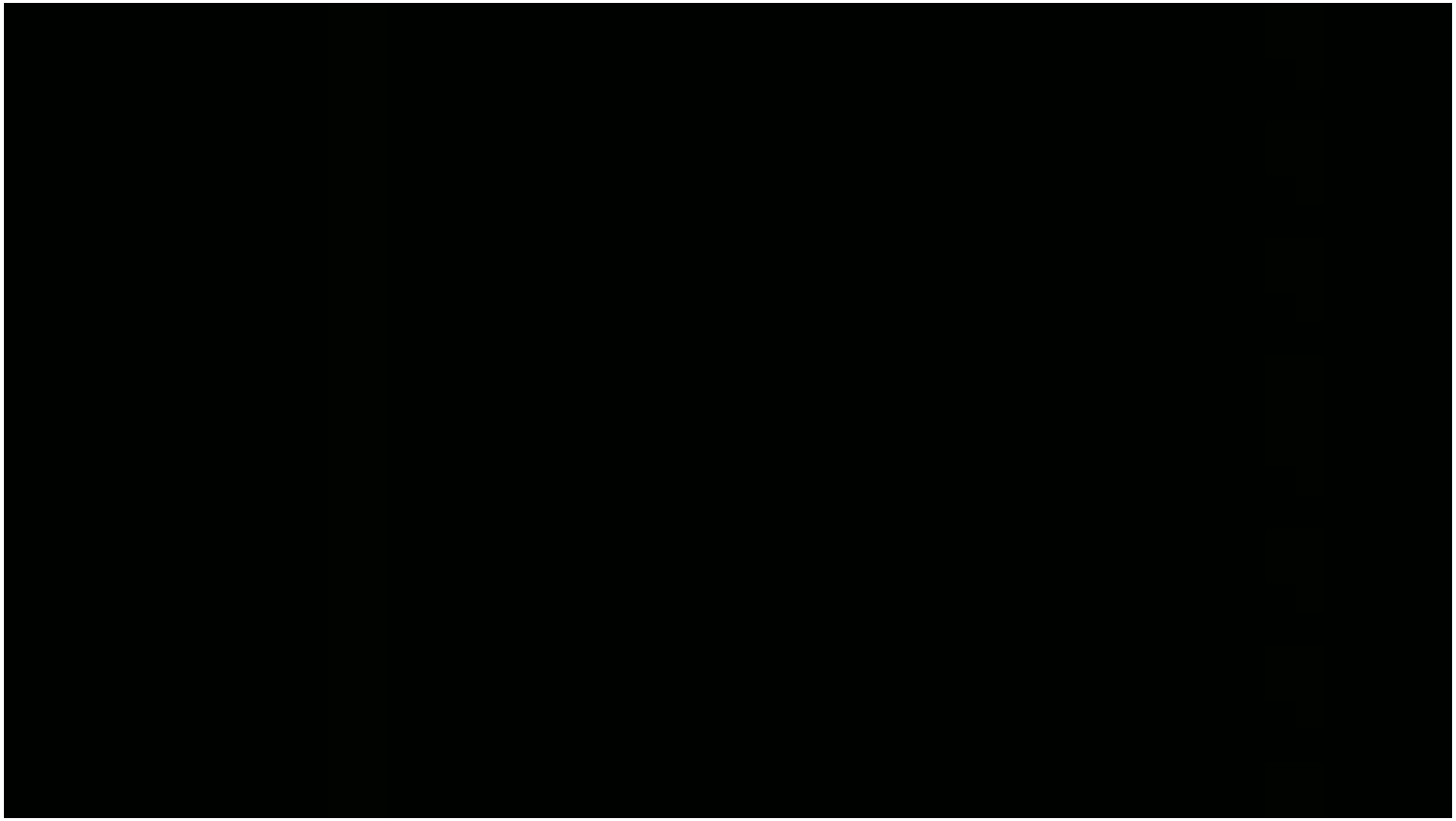
Time Protection: On-Going Work



© Gernot Heiser 2019-24 – CC BY 4.0



Real-World Use
Courtesy Boeing, DARPA



© Gernot Heiser 2019-24 – CC BY 4.0

Thank you!

To the brave AOS students for their interest and dedication

To the awesome AOS demonstrators

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

© Gernot Heiser 2019-24 – CC BY 4.0