School of Computer Science & Engineering

Trustworthy Systems Group

Trustworthy Systems R&D Update

Gernot Heiser

gernot@unsw.edu.au
@microkerneldude.bsky.social
https://microkerneldude.org/

What's Up at Trustworthy Systems?

- Overview of Microkit/sDDF/LionsOS developments
  - Pointers to work to be covered by separate talks
- Getting real about real-time
- Multikernel vs SMP

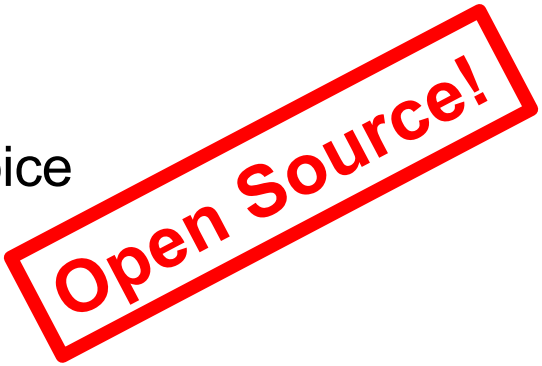
TS R&D

Microkit, sDDF, LionsOS, Pancake

Microkit (Refresher)

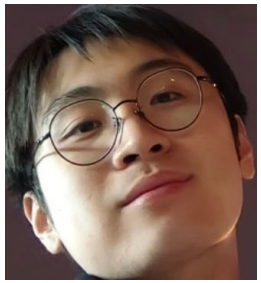
Purpose: Lower seL4's barrier to entry for developers

- Software development kit (SDK):
  - use pre-compiled kernel binary, use build system of choice
- Static architecture, defined at build time
- Hide seL4 complexities
  - Minimal abstractions: protection domain (PD), channel, memory region
  - Simple programming model: event handlers
- Easy-start tutorial

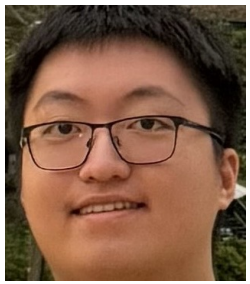


Simple but powerful:
Supports complete deployable OS (LionsOS)

Microkit Progress

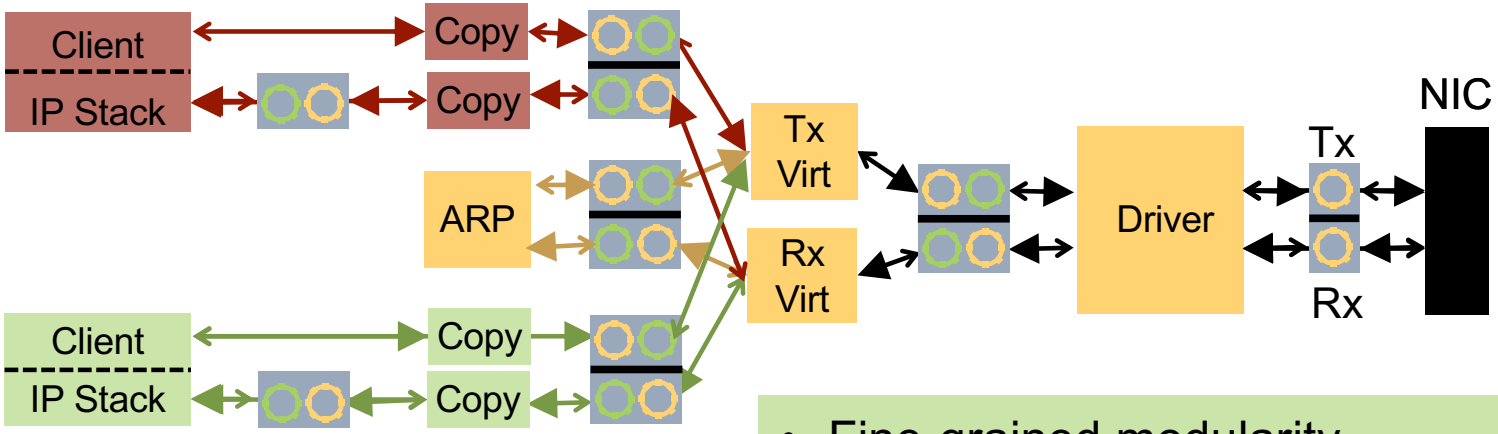


- Dynamic features:
  - Start, stop, upgrade PDs at run time
  - Seen at work in Carrels (see Carrels talk)
- Much improved/matured virtualisation support
  - x86_64 UEFI fully supported
  - Boots Windows 11 on x86_64 (see Bill's talk)



- Much improved tooling for system composition:
  - Acacia – see Lesley's talk

sDDF: seL4 Device Driver Framework



sDDF progress:
- Support for ACPI & PCIe
- Incremental progress on various device classes

- Fine-grained modularity
- Strict separation of concerns
- **Location transparency**
- Zero-copy data plane
- Control plane using bounded, lock-free, single-consumer, single-producer (SPSC) queues
- Outperforms Linux

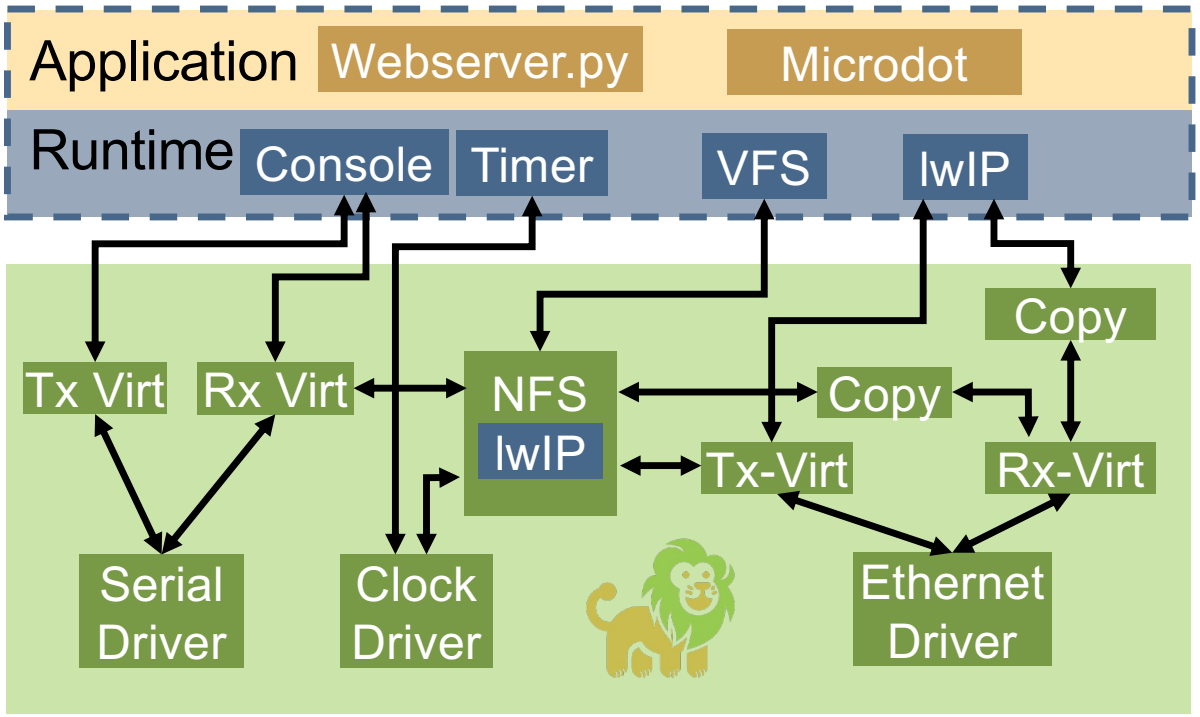
Open Source!

LionsOS: Microkit + sDDF + Services

- Static architecture
- Separation of concerns
- **Designed for verification**

Open Source!

Fine-grained Modularity

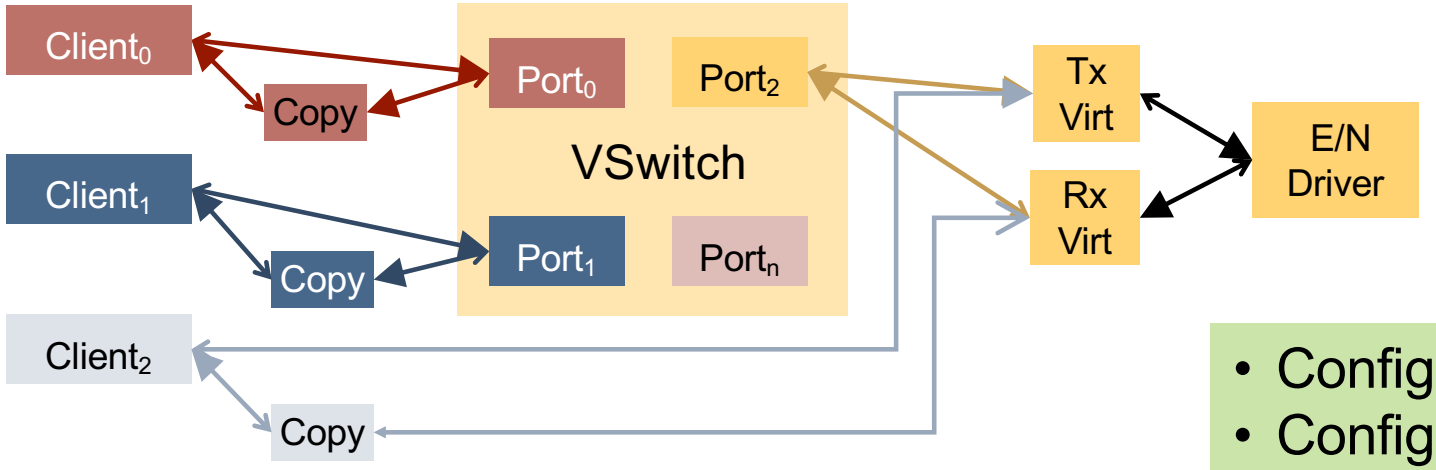


Server for https://sel4.systems

See Lesley's talk on analysing LionsOS networking performance

LionsOS Progress

- Firewall community project, released Nov'25
  - Multiple student projects added functionality since
- Virtual network switch (VSwitch)



- Configurable connections
- Configurable broadcast
- Used in Carrels (see talk)

PANCAKE
A Language for
Verified Systems Programming

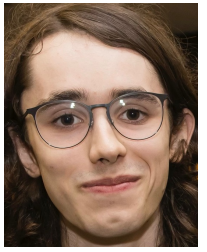
Verification Enabler: Pancake

C-like systems language:

- Simple, well-defined semantics
- Verified compiler
- Used to implement & verify drivers, virtualisers, libmicrokit
- Performance approaching C

Details in Miki's talk

Verification Agenda for Next Year



Real Real-Time

… aka provable MCS safety

What Is a Mixed-Criticality System?

"A mixed-critical system […] supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform." [Barhorst'09]

Criticality of a component is defined by the impact of failure:

1. loss of life
2. injury
3. inconvenience

Certification of critical component must not depend on behaviour of less critical components
⇒ **must bound any interference!**

MCS: Simple Example

Requirements:
- Critical operation analysable without assumptions on less critical
- Needs spatial and temporal isolation

- Runs every 20 ms for ms
- Must meet deadline

Critical

Less critical

- Runs frequently for µs
- Must preempt control or lose packets

Sensor readings

Control loop

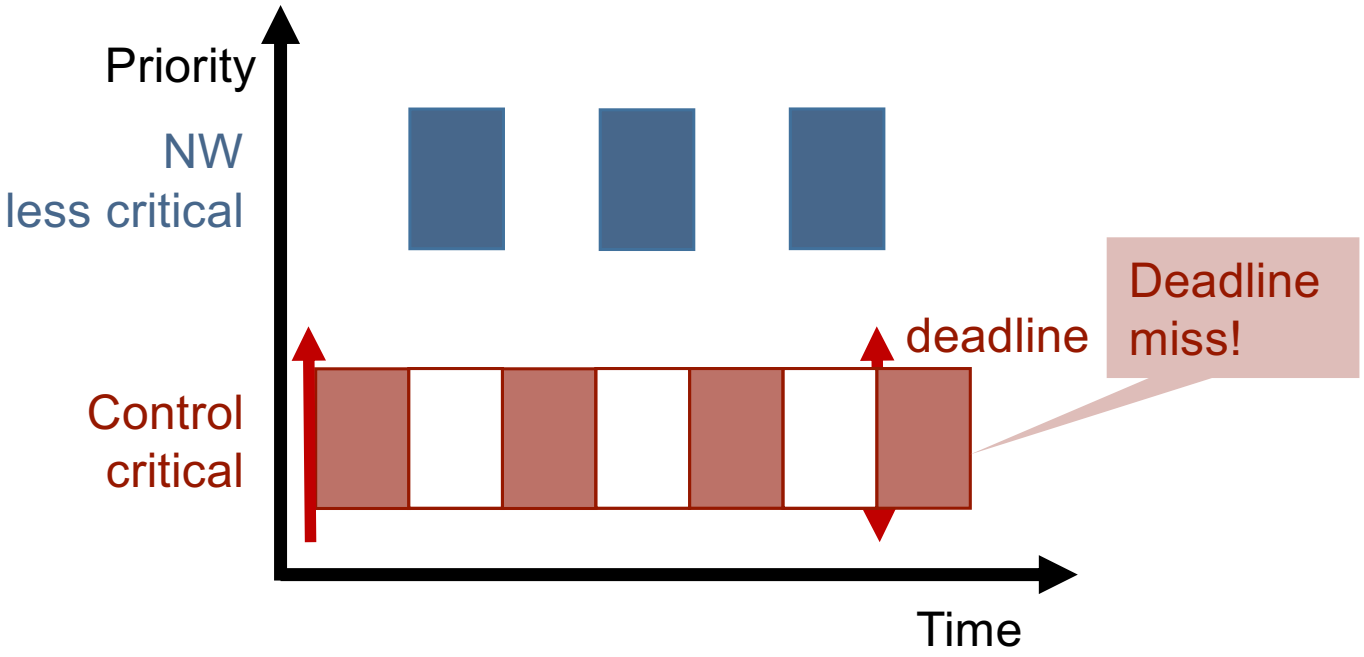
NW driver

NW interrupts

Waypoints

What Does This Mean for the OS?

- Spatial isolation ✔
- Temporal isolation ?

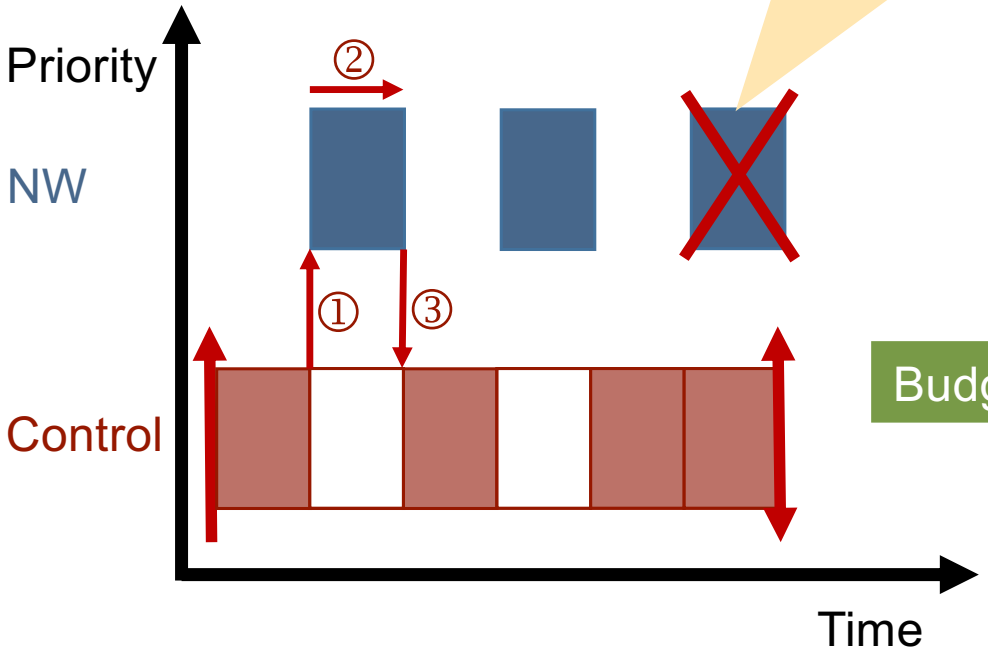


What Does This Mean for the OS?

- Spatial isolation ✔
- Temporal isolation ?

Limit preemption:
- Budget enforcement
- Rate limit

Need worst-case execution-time (WCET) analysis of the kernel!



Budget

Preemption delay:
Sum of
1. Preemption cost
2. Preempter execution
3. Re-scheduling cost

Kernel overhead

Don't We Have a WCET Analysis?

Timing Analysis of a Protected Operating System Kernel

Bernard Blackham†, Yao Shi†, Sudipta Chattopadhyay‡, Abhik Roychoudhury‡ and Gernot Heiser†

† *NICTA and University of New South Wales, Sydney, Australia*
‡ *National University of Singapore*

RTSS'11

Sequoll: a Framework for Model Checking Binaries

Bernard Blackham and Gernot Heiser

NICTA and University of New South Wales
Sydney, Australia

RTAS'13

Trickle: Automated Infeasible Path Detection Using All Minimal Unsatisfiable Subsets

Bernard Blackham†, Mark Liffiton‡ and Gernot Heiser†

†*NICTA and University of New South Wales, Sydney, Australia*
‡*Illinois Wesleyan University, Bloomington IL 61701, USA*

RTAS'14

Complete, High-Assurance Determination of Loop Bounds and Infeasible Paths for WCET Analysis

Thomas Sewell, Felix Kam, Gernot Heiser
Data61 (formerly NICTA) and UNSW
Sydney, Australia

RTAS'16

We Did, But…

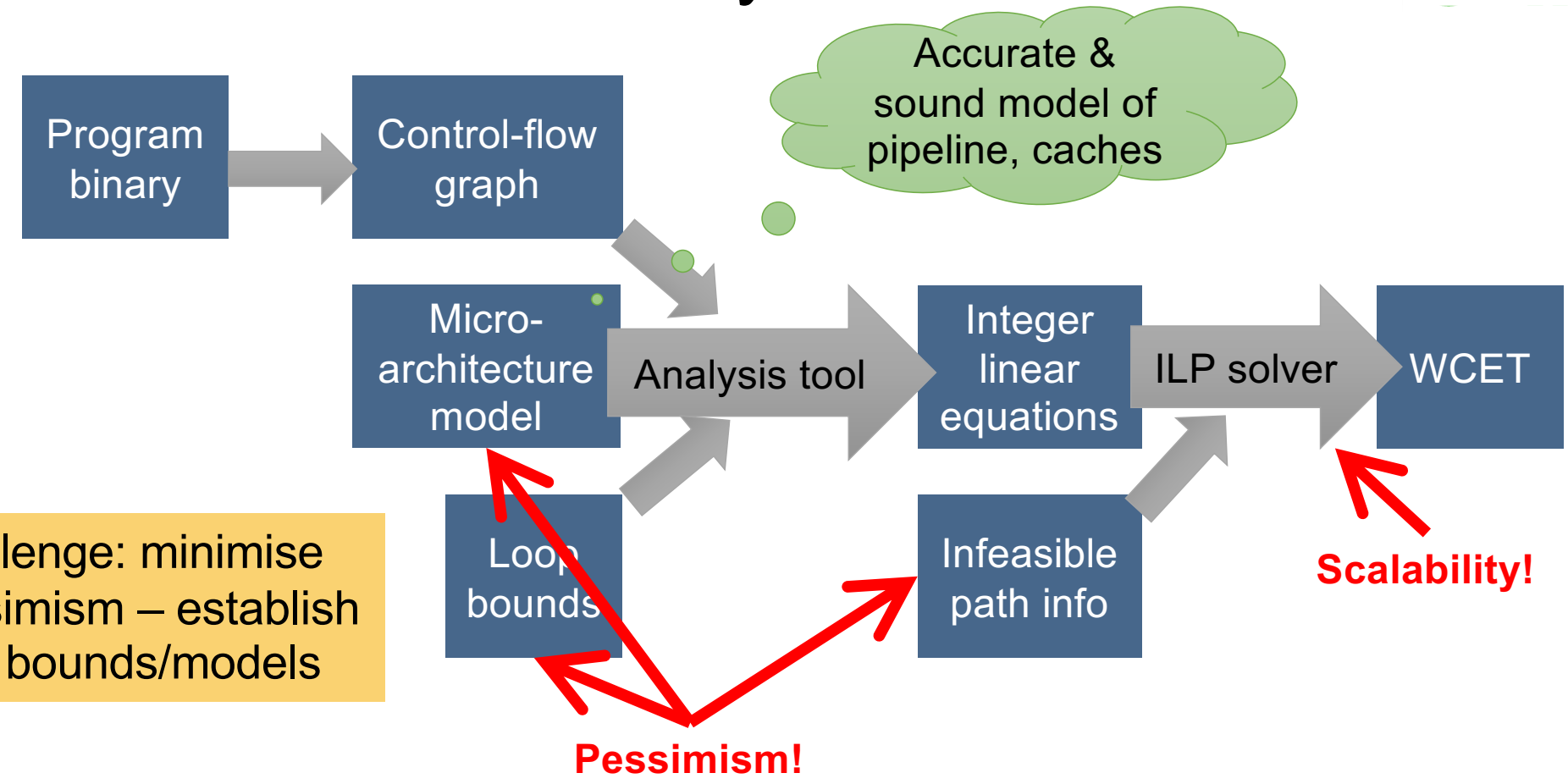
Analysis for Armv6/7 (first ever for a protected-mode OS kernel) based on heavily modified NUS Chronos tool

Chronos became unsupported halfway through the project

Arm stopped (for a while) providing instruction latencies when moving to OoO cores

WCET analysis bit-rotted

How Does WCET Analysis Work?

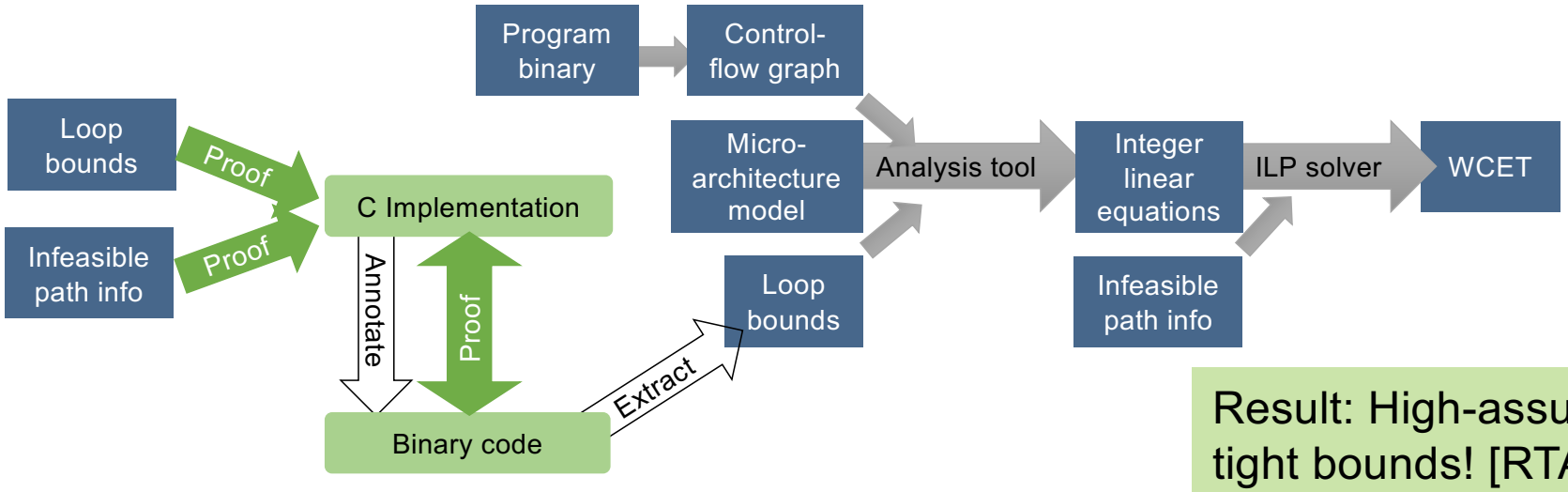


Challenge: minimise pessimism – establish tight bounds/models

Loop Bounds & Infeasible Paths

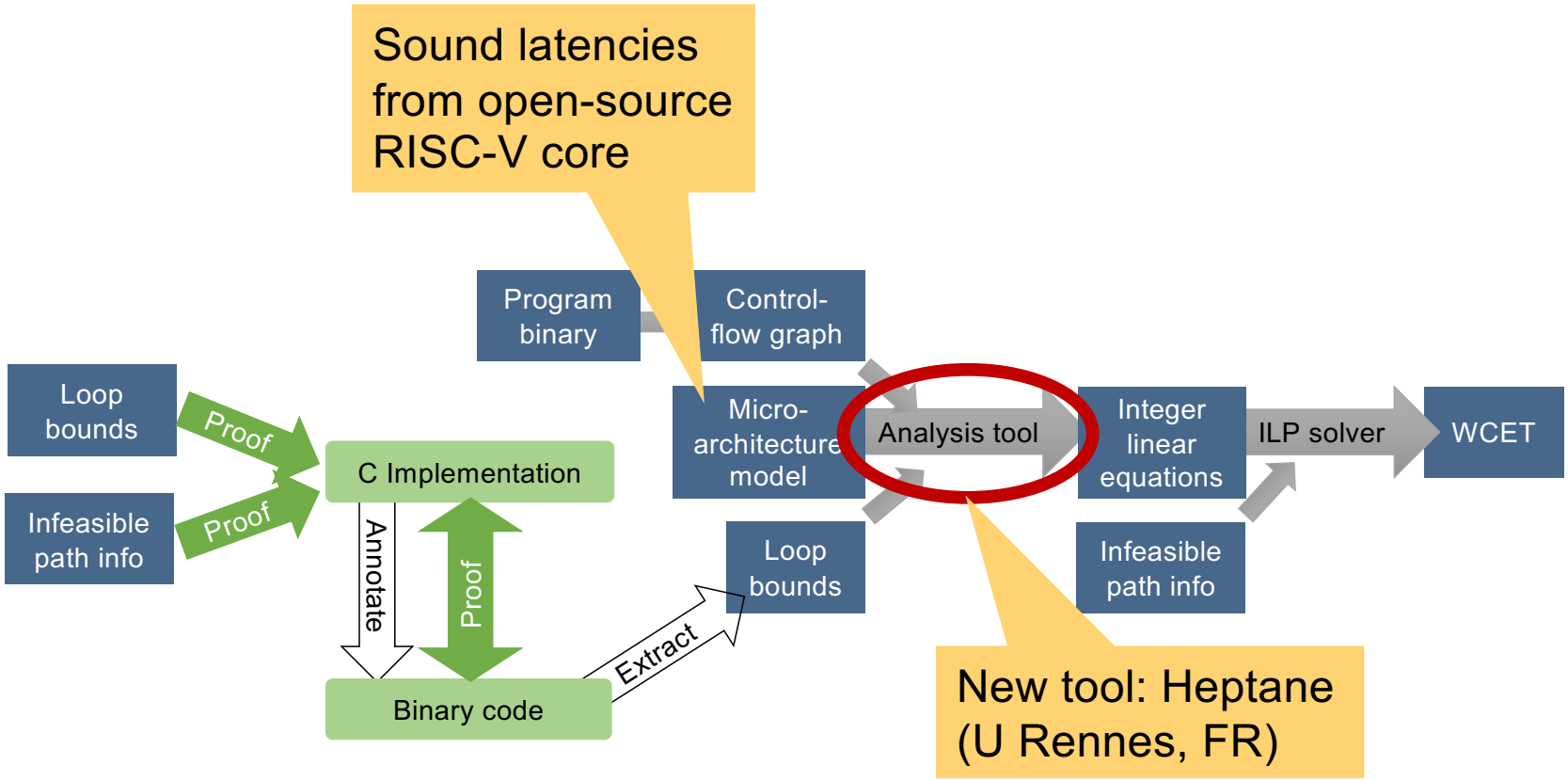
Tight loop bounds and infeasible path refutations infeasible to obtain from binary – lack of semantic information, especially pointer aliasing analysis

Idea:
- prove on C level
- transfer to binary using translation-validation toolchain



Result: High-assurance, tight bounds! [RTAS'16]

WCET Reboot



Kernel WCET Challenges

Heptane assumed …	… seL4 reality
Program exits by returning to and exiting from its entrypoint (e.g. `main()`)	Mode switch to user (e.g. `sret`) is a terminal CFG node
Standard call/return structure	Non-local jumps; tail & sibling calls bypass the caller
No recursion	Bounded recursion
Data accesses: stack + globals only	Arbitrary pointer dereferences
App-grade ISA subset suffices	`Zicsr`/CSRs, supervisor instructions, other arith/shift variants
Simple control code	Complex kernel code

Scale of the revival

Target: RISCV64_MCS_verified, RVC off, -fno-jump-tables

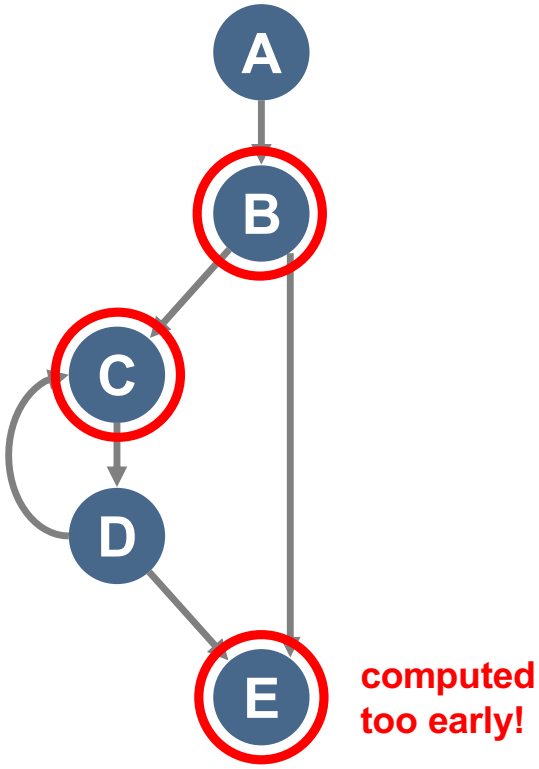
STATIC CFG
154 functions, 3.2k basic blocks, 50 loops

≈ 470 times more basic blocks, once each is analysed per calling context

AFTER CONTEXT-SENSITIVITY
110k calling contexts, 1.5M contextual basic blocks
ILP fed to the solver: 3.1M variables, 5.9M constraints

Consequence:
Analysis doesn't complete!

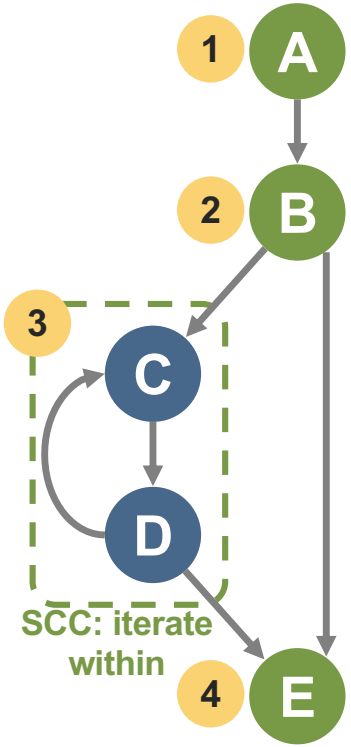
Scaling: Why the Analysis Never Finished



- Abstract cache state per basic block; states *join* at merge points; iterate to a fixed point.
- Heptane drove the iteration with a worklist. Sound, but blind to the graph's shape.
- After B, its successors, including E are added. E is computed too early.
- When C, D finally deliver, E and *everything downstream* re-run. Every branch multiplies the rework.

On seL4's 1.5 M contextual blocks: worklist grew to 600k entries. The analysis never completed.

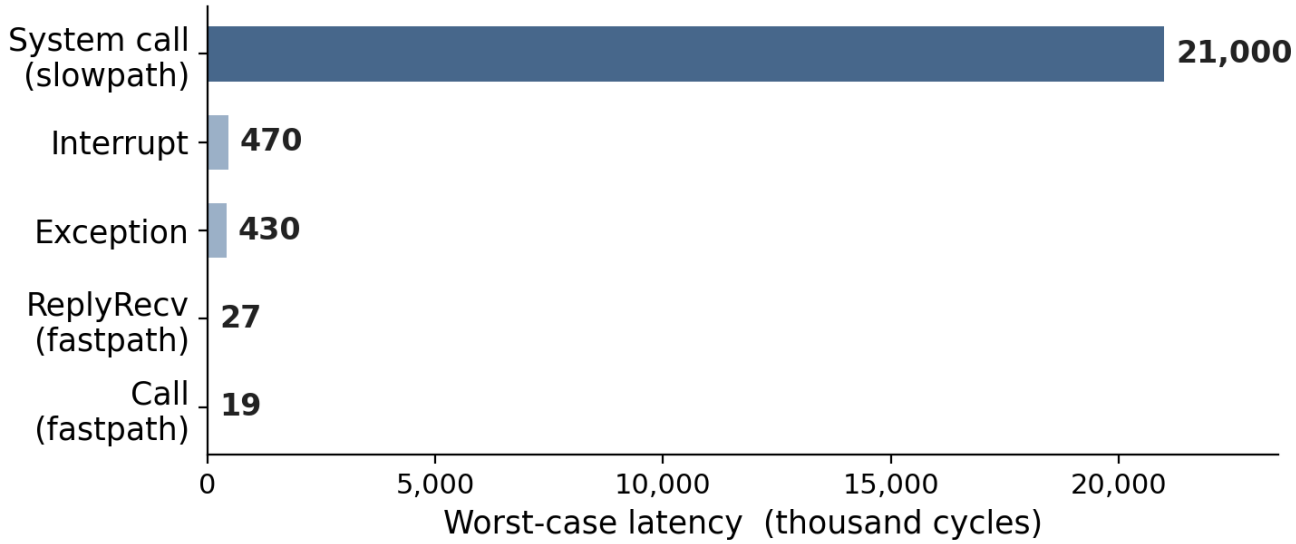
The fix: iterate SCCs in topological order



- Condense the CFG into its SCC DAG; process SCCs in *topological order.*
- Acyclic regions computed exactly once.
- Iterate only *inside* an SCC, until its state fixes.
- Same fixed point: iteration order doesn't change the result. [cf. Bourdoncle, FMPA'93]
- *Plus engineering hygiene: fewer ILP variables, improved memory management, less copying of large analysis state*

All full-kernel cache analyses: from infeasible to ≈ 1 h

First end-to-end result (OoM)



Order-of-magnitude only

- *placeholder latencies,*
- *uniform loop bounds,*
- *no infeasible path elimination*
- Validates *scalability*: running the pipeline end-to-end

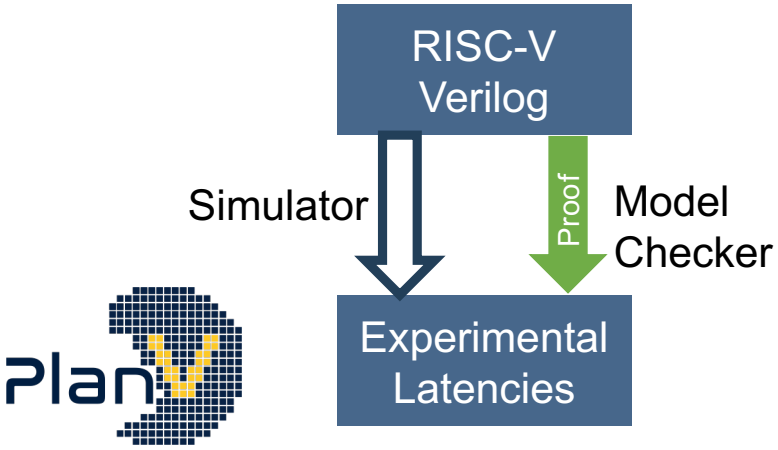
OSPERT'26

Kernel WCET ≈ 21M cycles; the slowpath (general system-call entry) dominates

Scope: single-core, in-order.

WCET Work In Progress

Sound instruction latencies



Complete kernel WCET model:

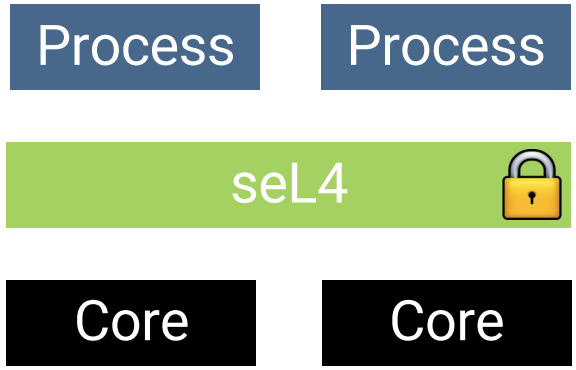
- ✓ Revive verified loop-bound import
- Revive infeasible-path refutations
- Detailed model
  - Kernel entry points
  - Distinguish syscalls
  - Parameterise syscall latencies

Aim: Verified response-time analysis (RTA)!

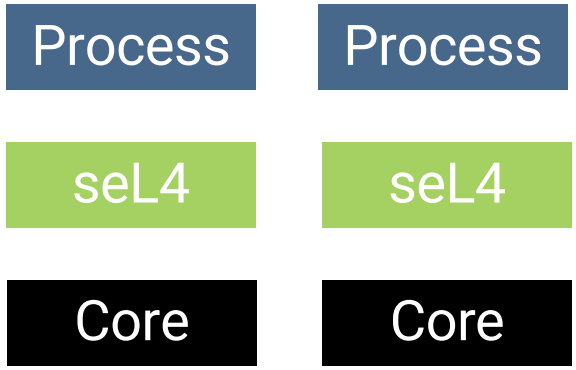
SMP vs Multikernel

SMP vs Multikernel

Symmetric multi-processing (SMP)



Multikernel



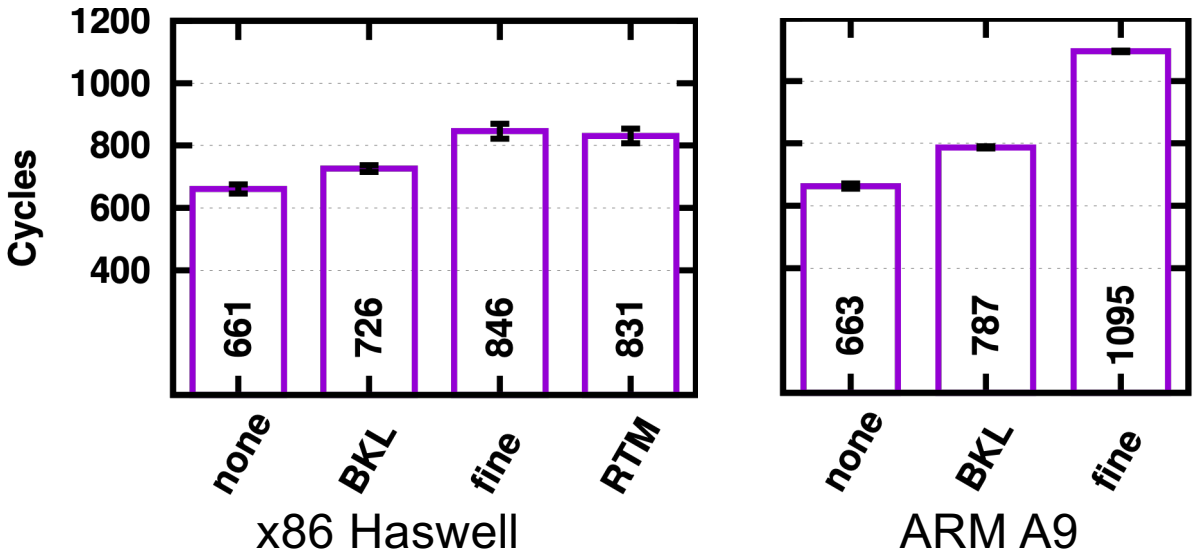
11 Years Ago: BKL vs FGL

For a Microkernel, a Big Lock Is Fine

APSys'15

Sean Peters, Adrian Danis, Kevin Elphinstone, Gernot Heiser

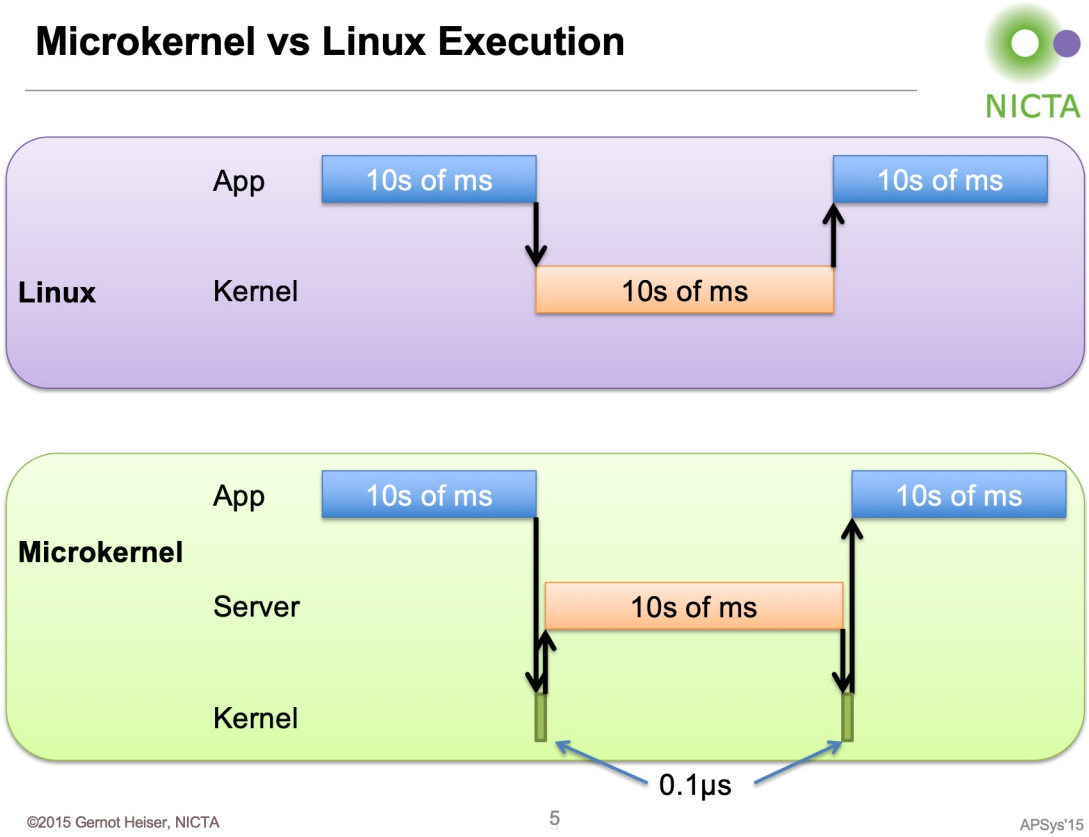
NICTA and UNSW Australia



Findings:

- Locks are expensive
  - Big kernel lock (BKL) adds 10% to syscall
  - Fine-grained locking (FGL) adds 20–70%

11 Years Ago: Observations

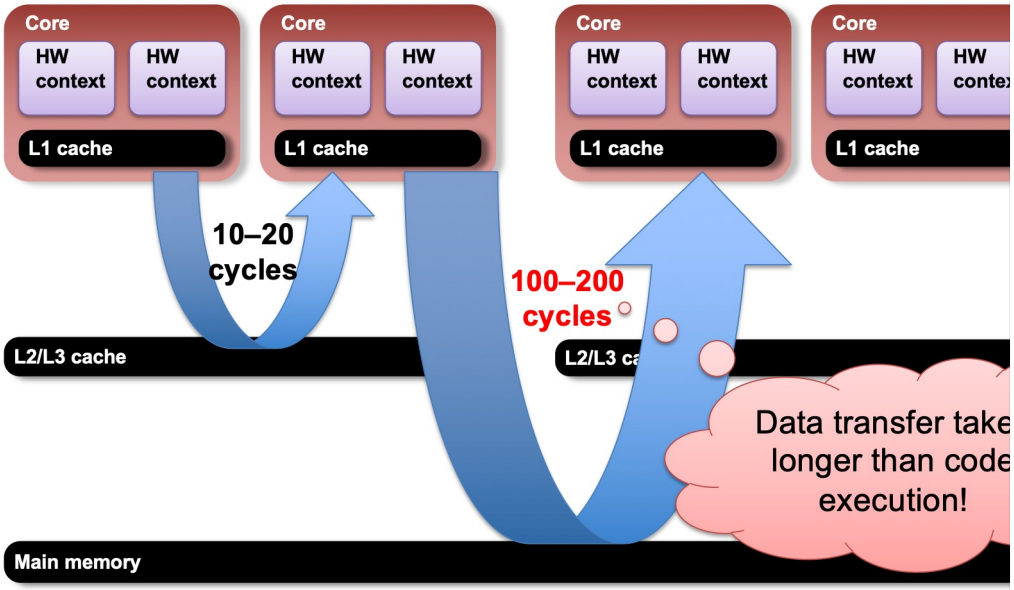


Microkernels are different!

- Syscalls are ≈ 0.1µs
- Linux syscalls ≈ 10s ms

11 Years Ago: Assertions

Cache Line Migration Latencies



Microkernel Multicore Design

Assertion 1: Minimise locks, not locked code

- Amount of locked code is small anyway, 100–200 instructions
- Corresponds to medium-grained locks in Linux
- Cost of locks is within an OoM of kernel execution time
- Kernel times are short ⇒ contention is low

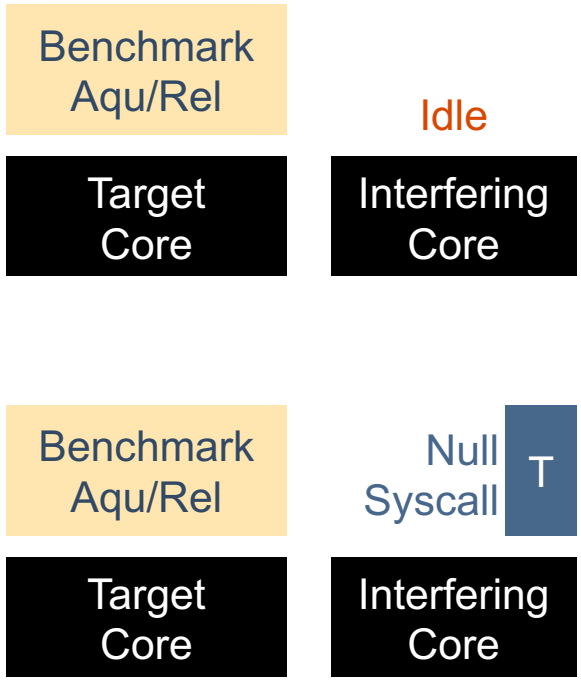
Assertion 2: Don't share mirokernel data without shared cache

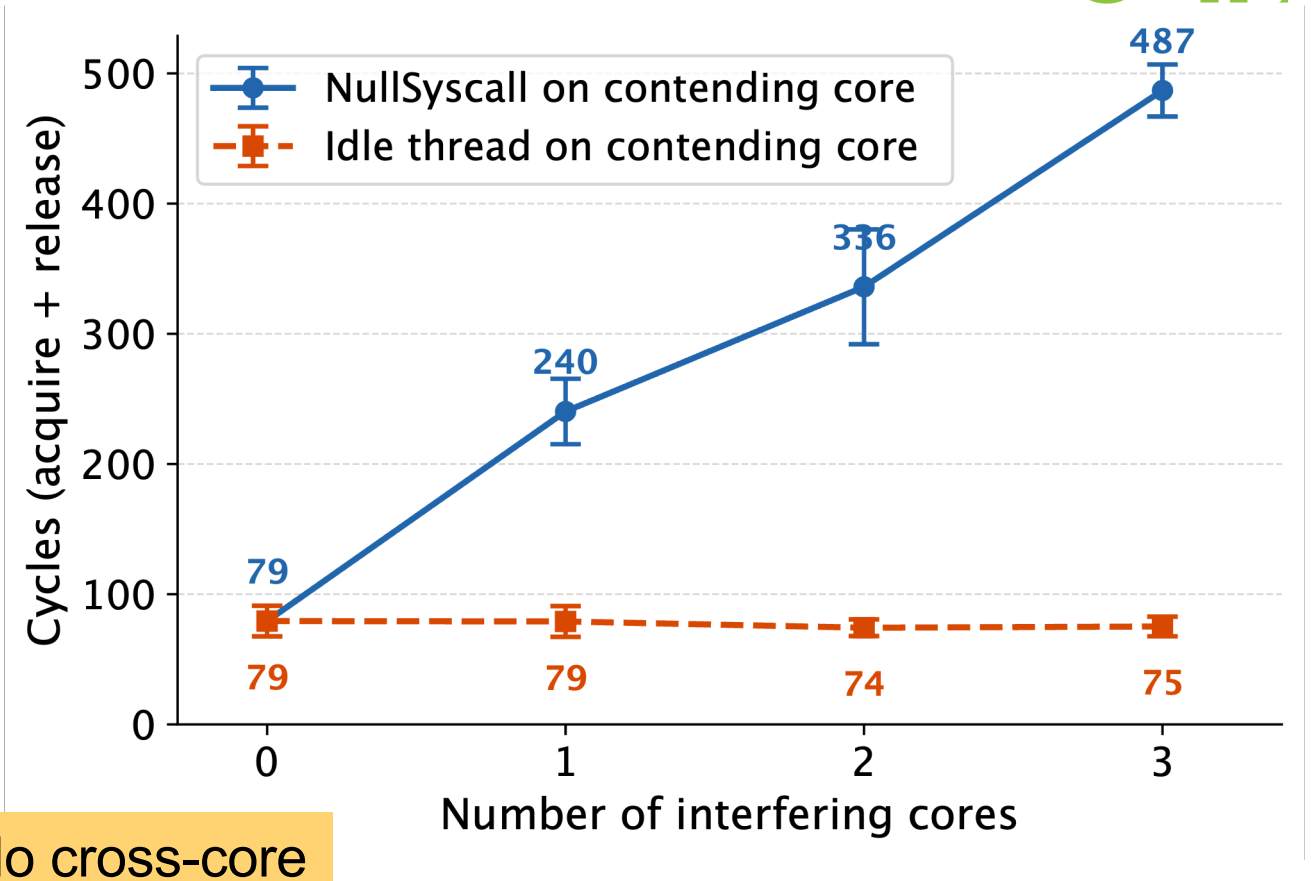
- Migrating only a few cache lines takes longer than rest of syscall

Assertion 3: Big lock will perform for closely-coupled cores

- Shared caches presently have moderate core counts
- Big lock in a *well-designed* microkernel will scale there

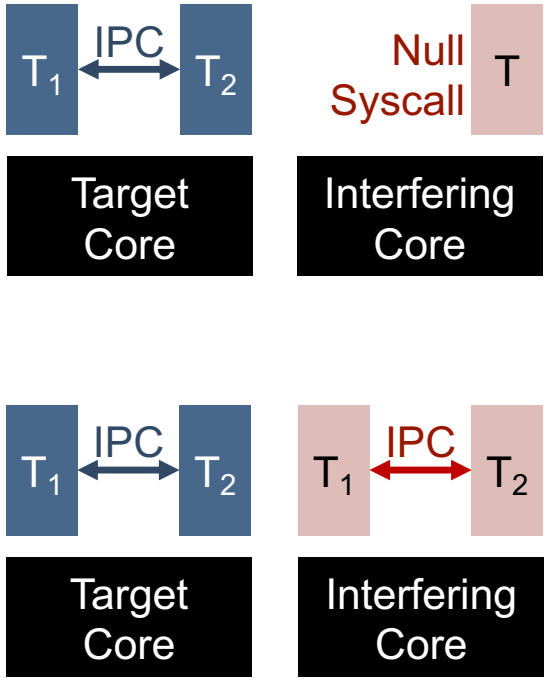
Let's Revisit the BLK

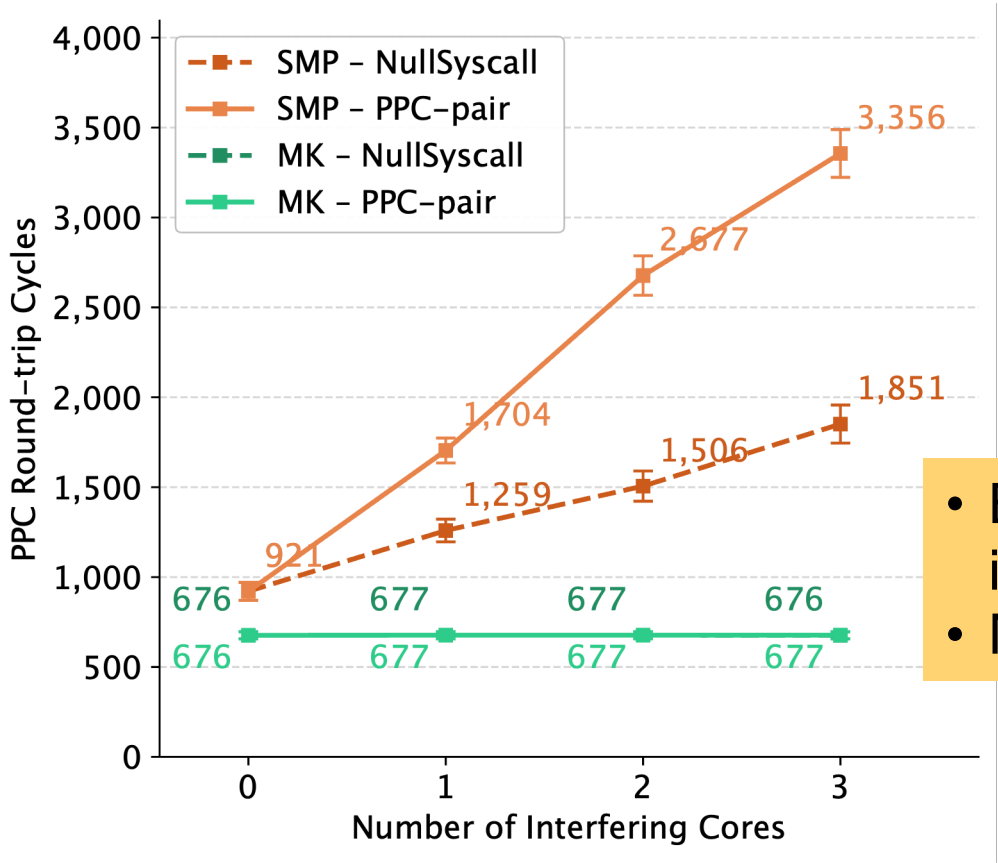




No cross-core operations!

Ping-Pong Interference Benchmark



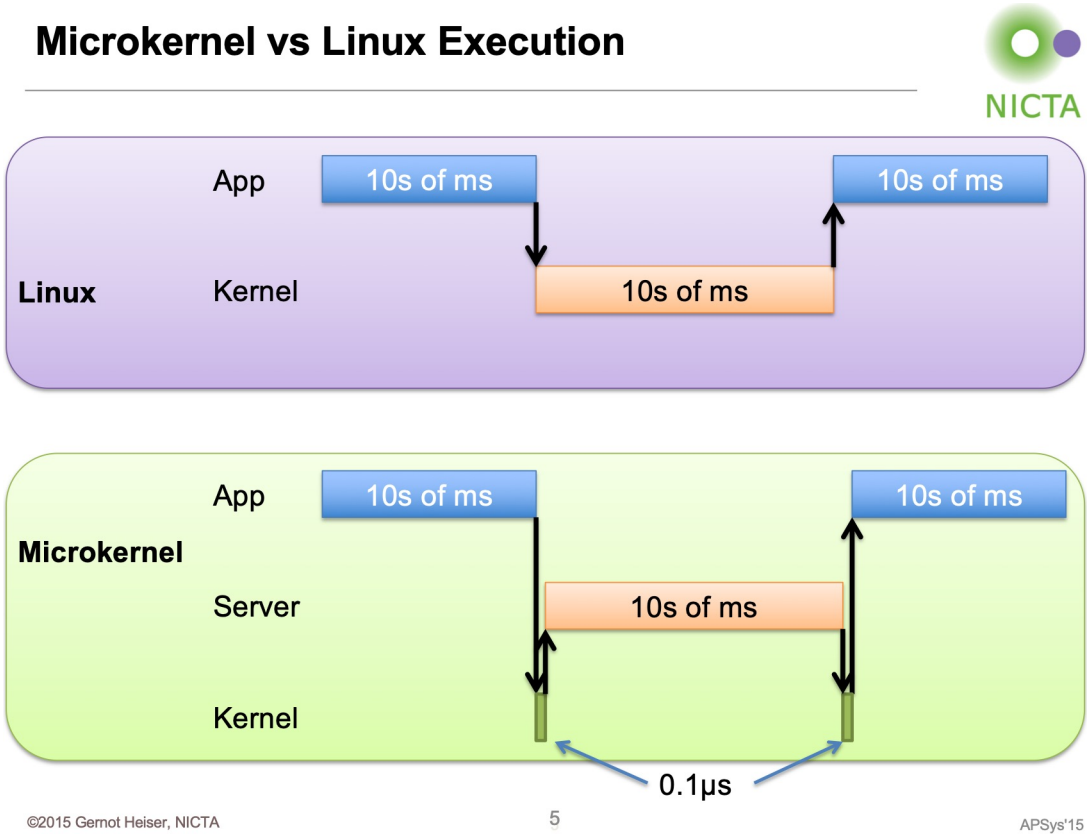


- BKL: strong interference
- Multikernel none

Observations

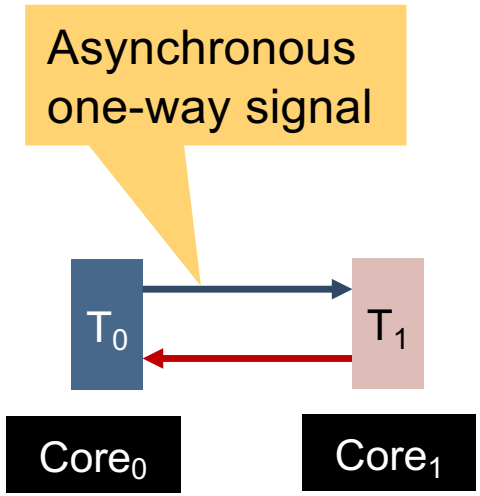
- None of this is surprising
- APSys'15 argument:
  - Syscalls are short, with sufficient usermode execution will not interfere
  - SMP is the preferred choice for cores sharing an L2 cache

Really?

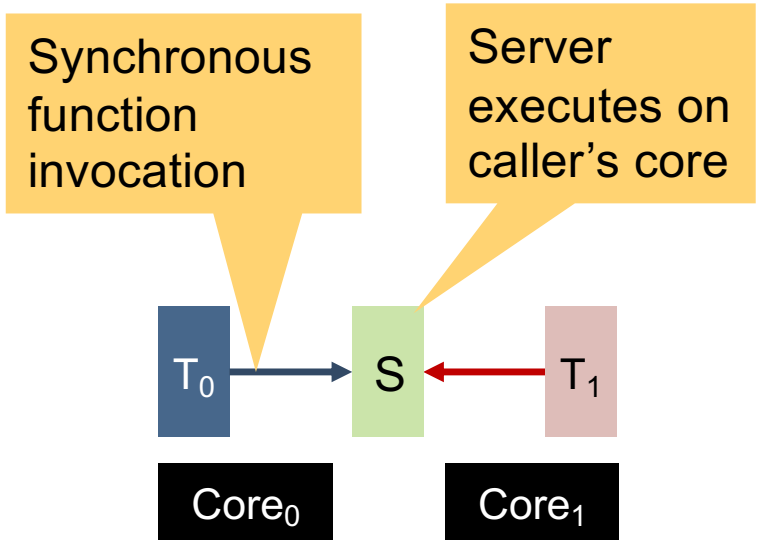


How About Cross-Core Operations?

Two relevant cross-core operations:



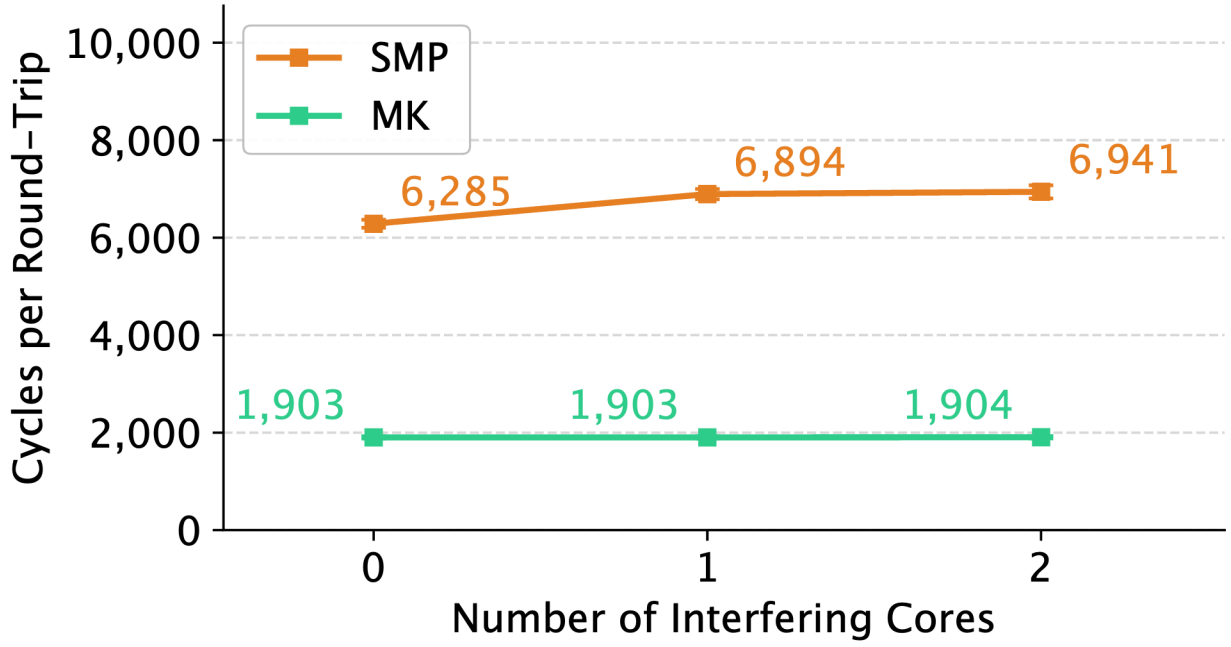
Cross-core Notifications



Shared passive servers

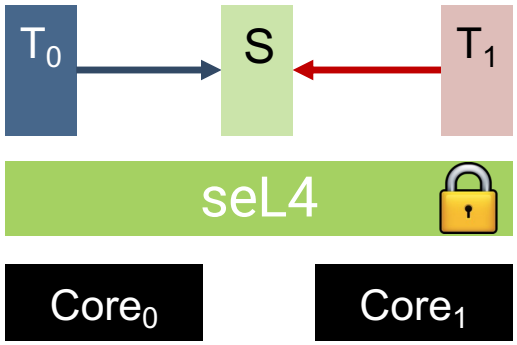
Cross-Core Notifications

Two cores ping-pong Notifications

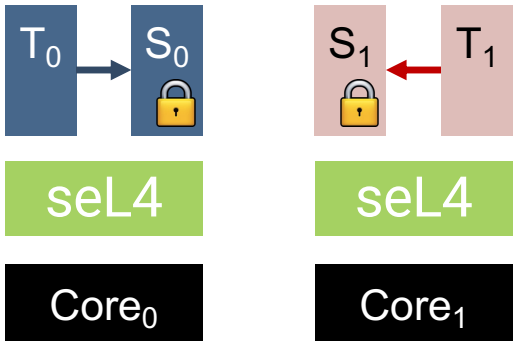


Doing everything in the kernel is **more** expensive than in userland!

Passive Server on Multikernel

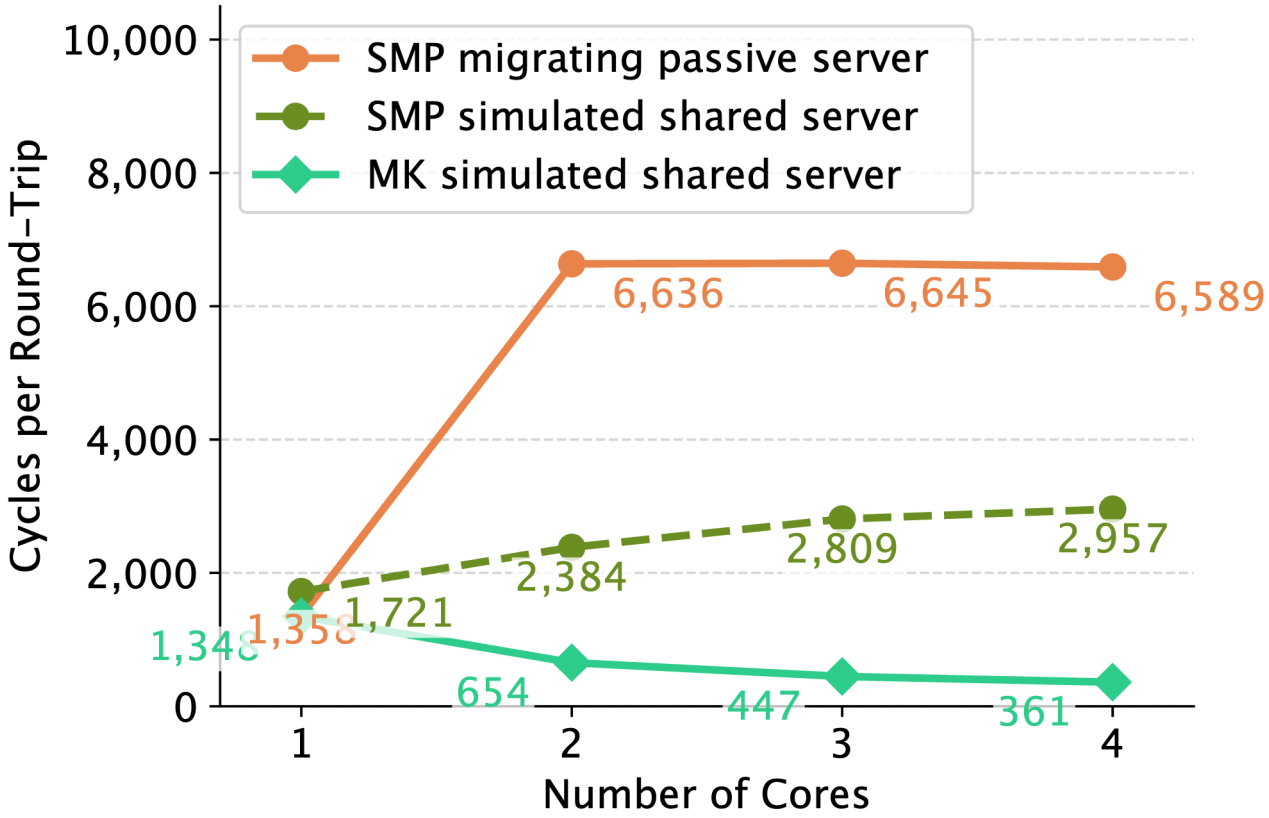


SMP



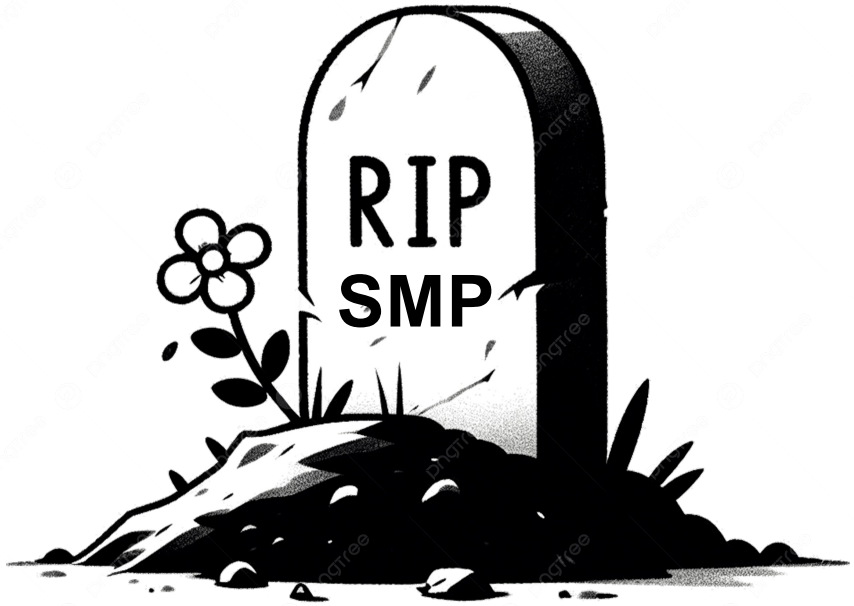
Multikernel

Passive-Server Performance



Cores compete for server
⇒ frequent migrations

Minimality principle:
A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system's required functionality.
[Liedtke SOSP'95]



https://trustworthy.systems