School of Computer Science & Engineering

Trustworthy Systems Group

Resurrecting seL4's WCET Analysis

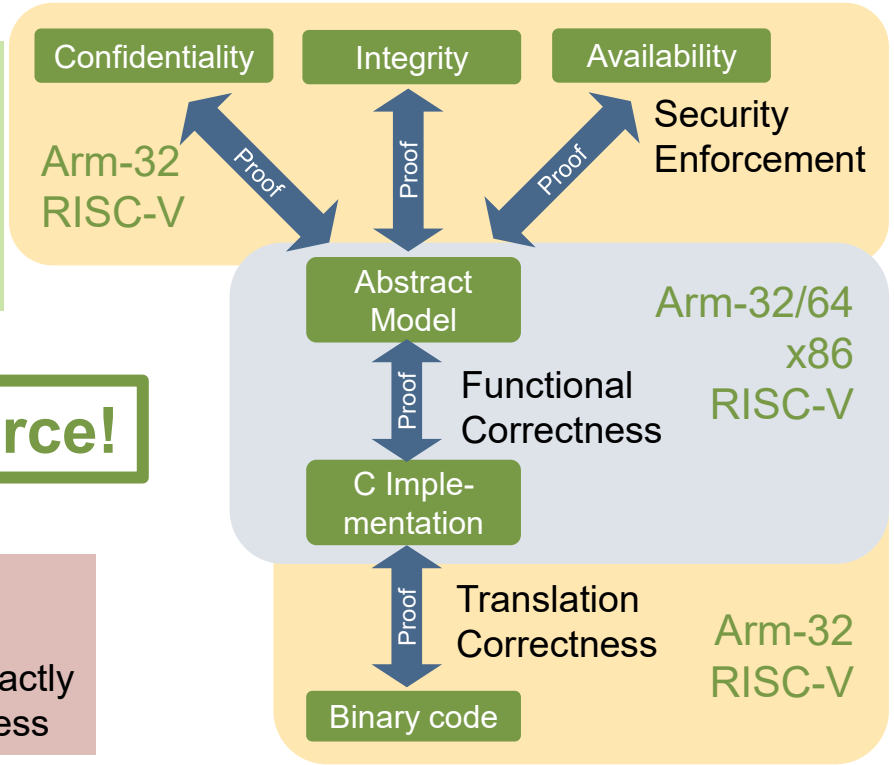
Simon Sillitoe (UNSW, Sydney), Abdul Basit (PlanV Tech, Munich),
Massimiliano Giacometti (PlanV Tech) and Gernot Heiser (UNSW)
s.sillitoe@unsw.edu.au
https://trustworthy.systems/

Overview

- What is seL4?
- Prior WCET analysis of seL4
- Why redo it, and the challenge
- Where we are now and what's next

seL4: Formally Verified Microkernel

- World's first OS kernel with functional correctness proof
- Most comprehensive verification
- Only verified OS with capability-based fine-grained protection



Open Source!

Present limitations
- Initialisation code not verified
- MMU, caches modelled abstractly
- Multicore verification in progress

Why timing needs its own guarantee

A verified kernel is proved to *behave* exactly as specified, not necessarily that it does so *in time*.

- seL4: functional correctness proven; a trusted base for safety- and security-critical systems [Klein, SOSP'09]
- Hard real-time needs a timing guarantee on top of functional correctness
- seL4 previously had a sound, tight WCET analysis, but the tool and hardware latencies it relied on became unusable
- **This work: how we are getting it back on 64-bit RISC-V, and where we are**

The original 2011 analysis

First sound, complete WCET of a protected-mode kernel
Armv7, ~10k SLOC; later used to optimise the kernel [Blackham, RTSS'11; EuroSys'12]

Chronos WCET tool
heavily modified for a production kernel

Source → binary proof chain
loop bounds & infeasible paths proved in C, carried to the binary by translation validation
[Sewell, RTS'17]

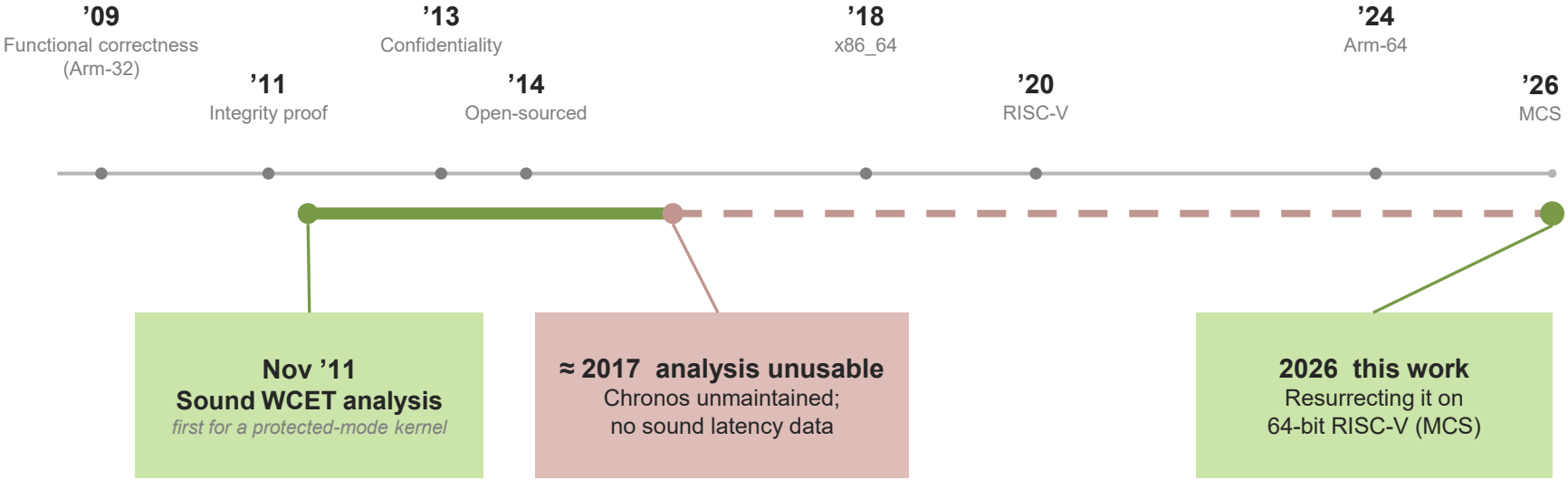
Verified seL4 – the high-assurance foundation
functional correctness proven; targeting hard real-time

Why the analysis became unusable

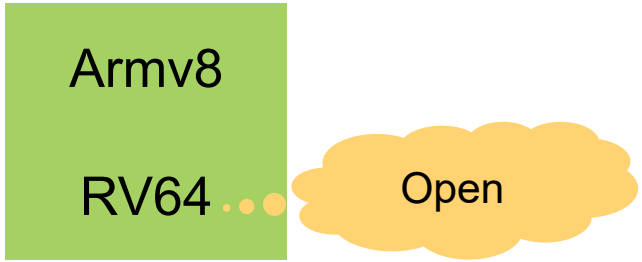
- Chronos was unmaintained before that line of work even finished
- No latency data we could treat as sound for the kernel

- **The analysis could no longer be reproduced**

seL4 timeline: the WCET analysis fell out of use



What changed: 64-bit, MCS, open RISC-V



seL4 MCS: time as a capability-controlled resource

Classical seL4 thread attributes

- Priority
- Time slice

Not runnable if null

seL4-MCS thread attributes

- Priority
- Scheduling context capability

Capability for time

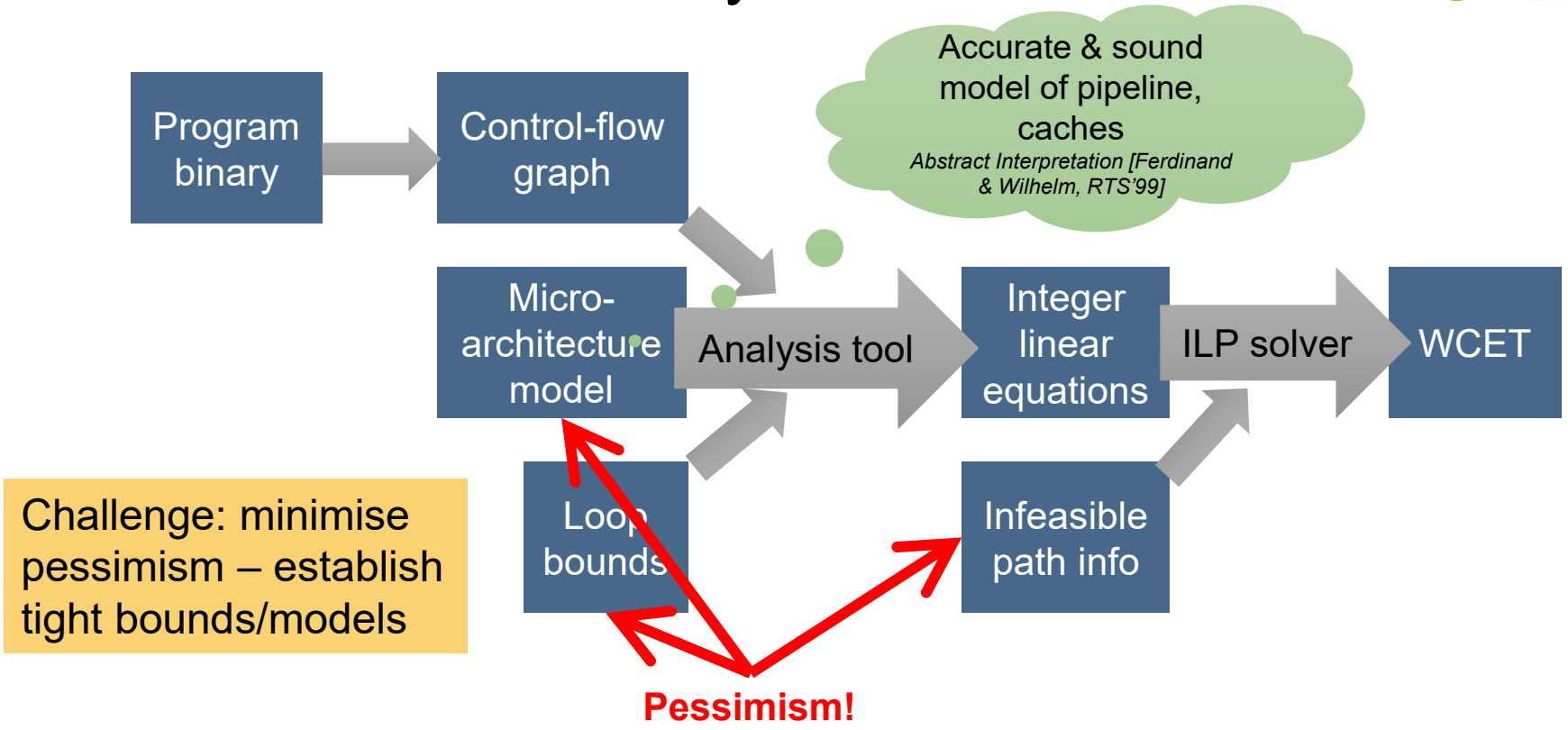
Scheduling context object

- T: period
- C: budget (≤ T)

Limits CPU access!

Reservation

How static WCET analysis works



Challenge: minimise pessimism – establish tight bounds/models

Choosing a WCET tool: requirements

Requirement: open-source, actively maintainable, binary-level.

	OTAWA	Platin [Maroun, WCET'24]	Heptane [Hardy, WCET'17]
Open-source	✓	✓	✓
Actively maintained and extensible	✗ no maintainer response	✓	✓ we extend, scale & upstream
Binary-level (compiler out of TCB)	✓	– compiler-coupled; trusts IR→binary map	✓

Tools (often) assume application code

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

Scale of the revival

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

STATIC CFG

154 functions, 3.2k basic blocks, 50 loops

≈ 470×

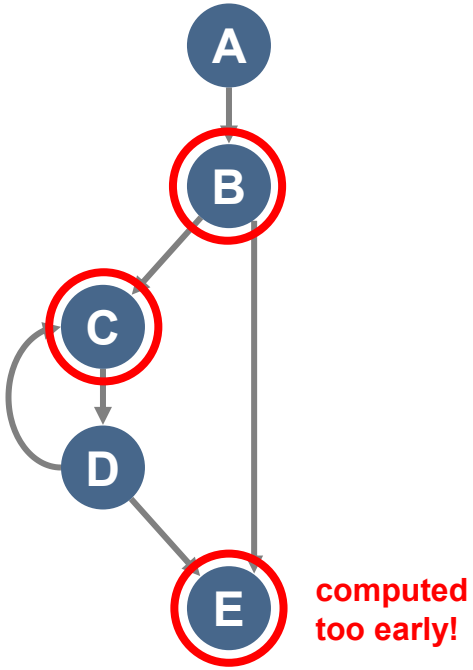
more basic blocks once each is analysed per calling context

AFTER CONTEXT-SENSITIVITY

110k calling contexts, 1.5M contextual basic blocks

ILP fed to the solver: 3.1M variables, 5.9M constraints

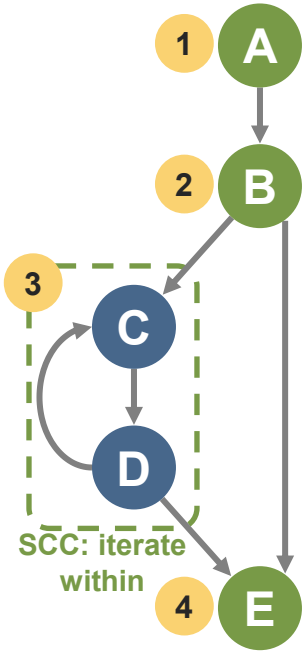
Scaling: why the analysis never finished



- 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.

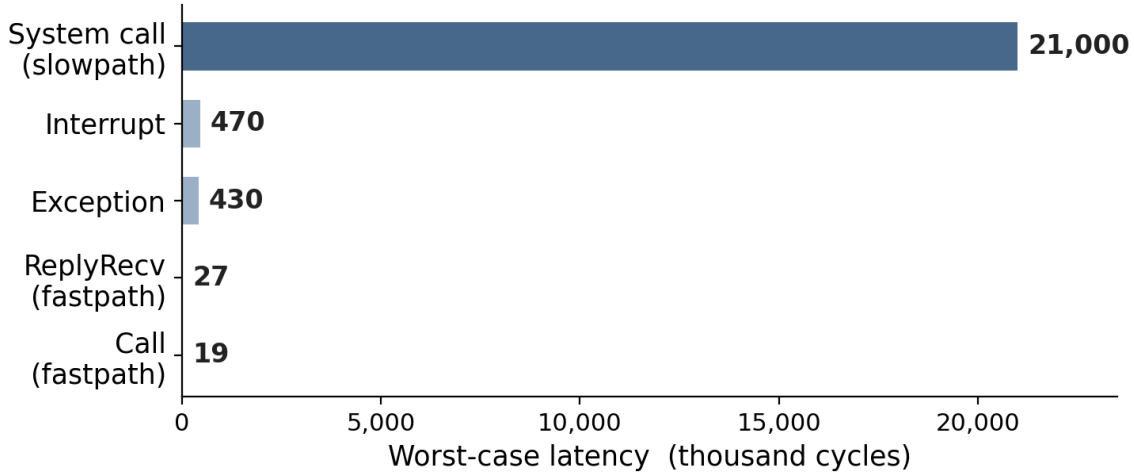
The fix: iterate SCCs in topological order



- 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 (order-of-magnitude)



Order-of-magnitude only

placeholder latencies, uniform loop bounds

Validates *scalability*: running the pipeline end-to-end

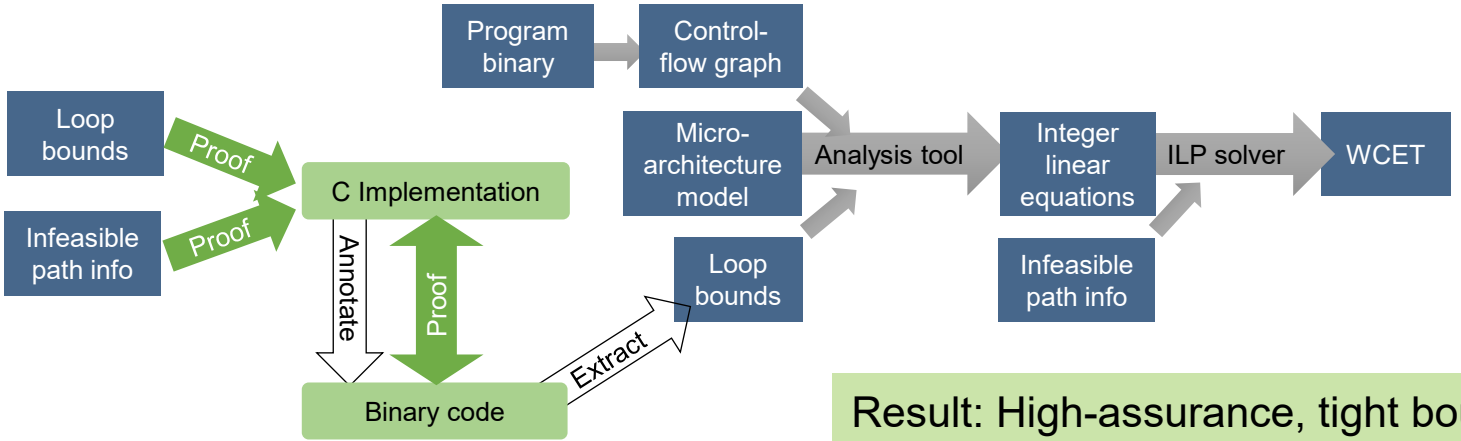
Kernel WCET ≈ 21M cycles; the slowpath (general system-call entry) dominates

Scope: single-core, in-order.

Tight loop bounds from source-level proofs

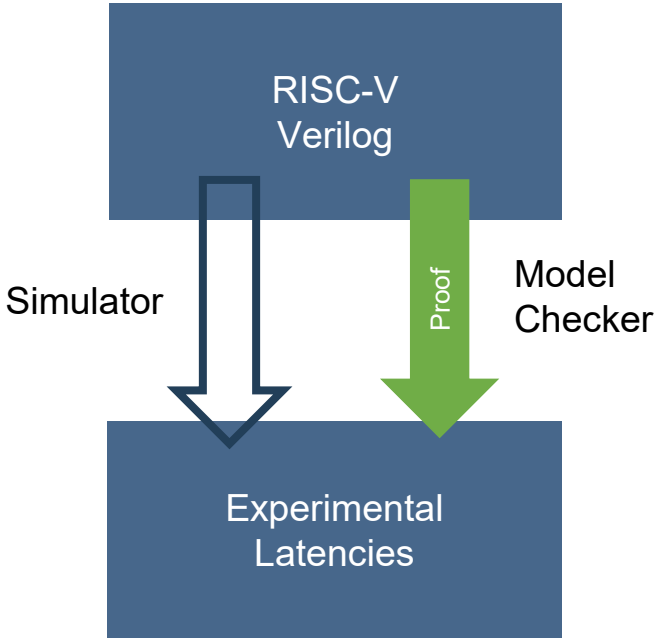
Tight loop bounds and infeasible path refutations infeasible to obtain from binary – lack of semantic information, especially pointer aliasing analysis

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



Result: High-assurance, tight bounds!

Deriving latencies from hardware



Conclusion

- WCET analysis re-established for recent seL4-MCS on 64-bit RISC-V
- Heptane proved a good base: open, binary-level, maintainable; our extensions are upstreamed
- Per-entry context-sensitivity (syscalls, interrupts, exceptions, fastpaths) feeds less-pessimistic schedulability
- **Today an estimate; sound CVA6 latencies and verified bounds will make it a guarantee**

WCET analysis is back on seL4.
Hard real-time is now within reach.

Questions?