

Real-time and Real Trustworthiness: Timing Analysis of a Protected OS Kernel



Bernard Blackham†
Yao Shi†
Sudipta Chattopadhyay*
Abhik Roychoudhury*
Gernot Heiser†

† The University of New South Wales & NICTA, Sydney, Australia
* National University of Singapore



Australian Government
Department of Broadband, Communications
and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners



Australian
National
University

UNSW
THE UNIVERSITY OF NEW SOUTH WALES



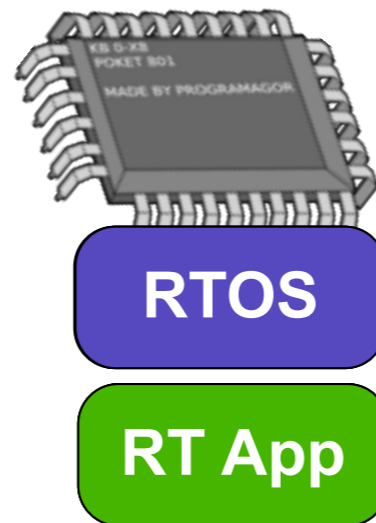
Griffith
UNIVERSITY

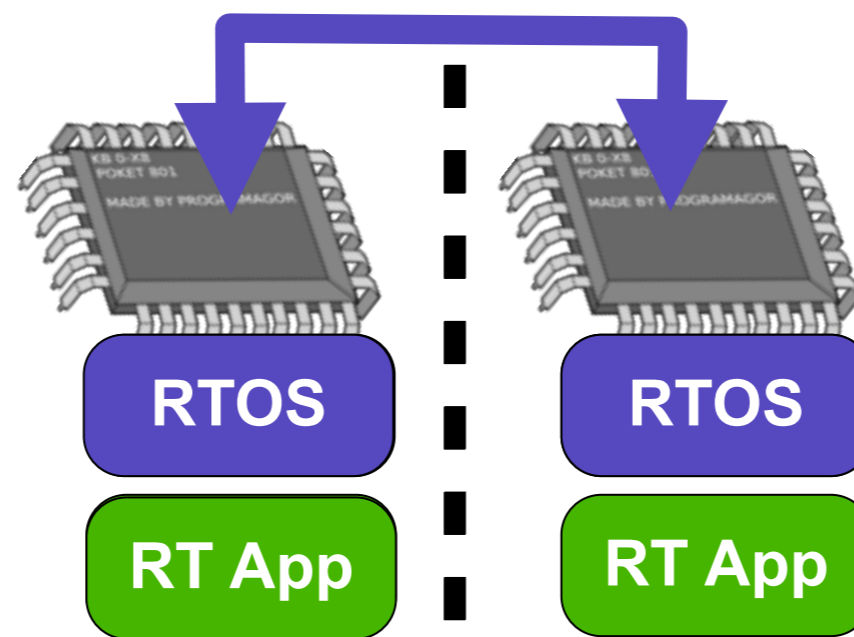


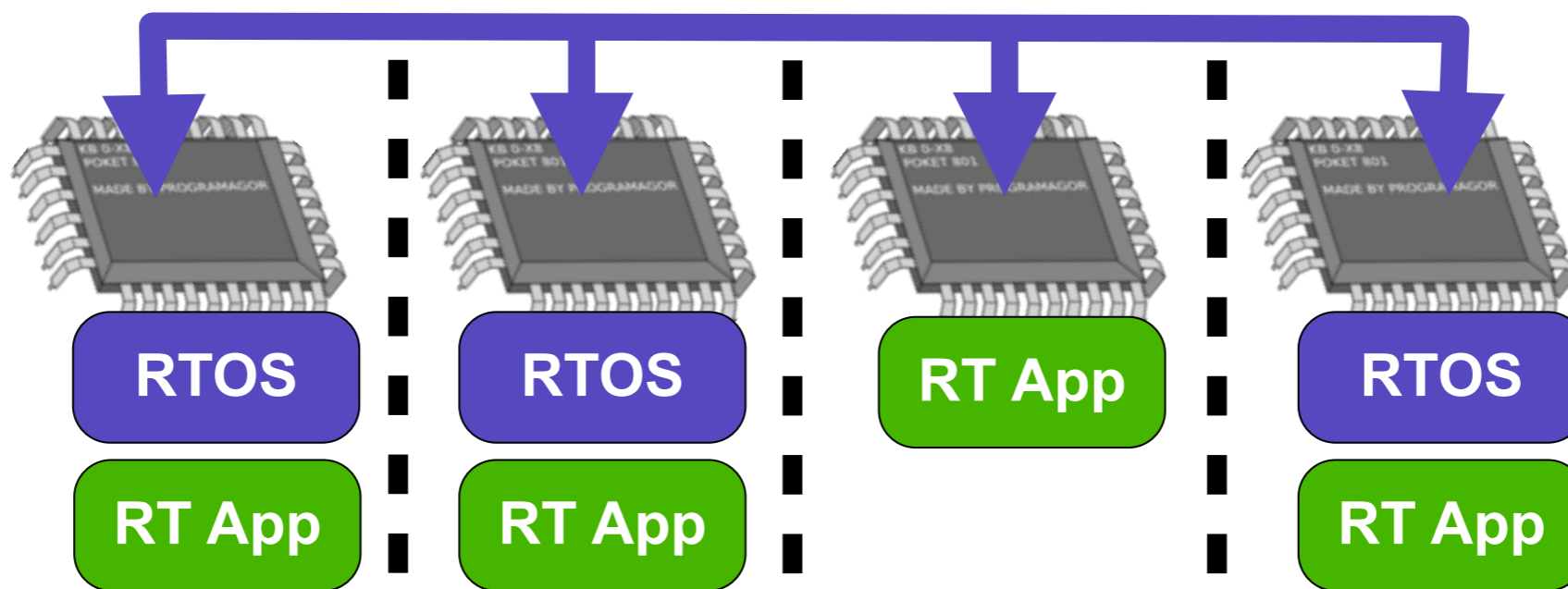
State Government
Victoria

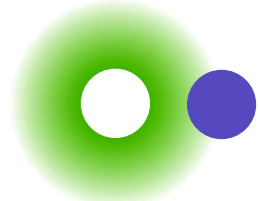


In the beginning...

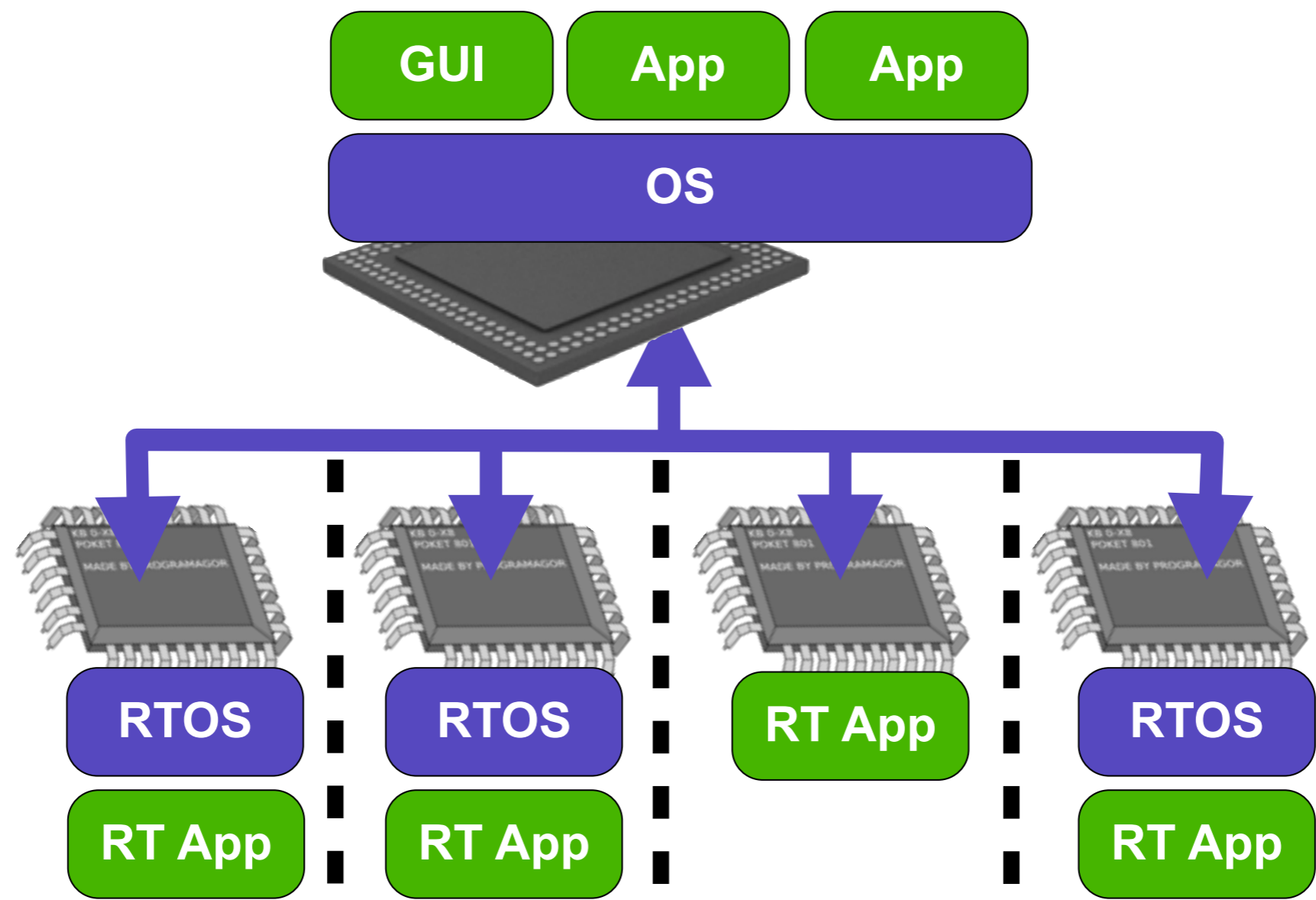






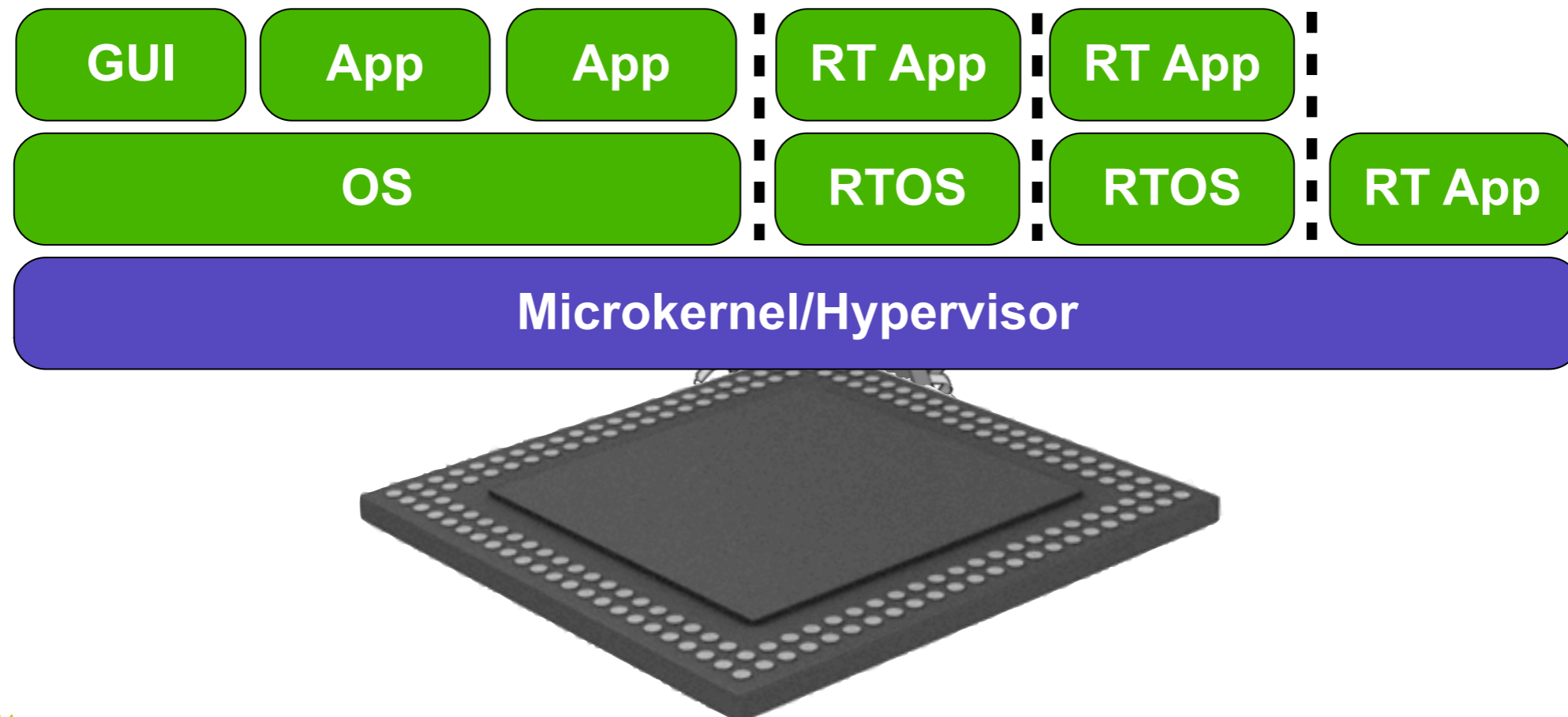


Today...



Can we regain sanity?

- Merge subsystems onto one CPU
 - As suggested by [Mehnert et al, RTSS'02], and many others



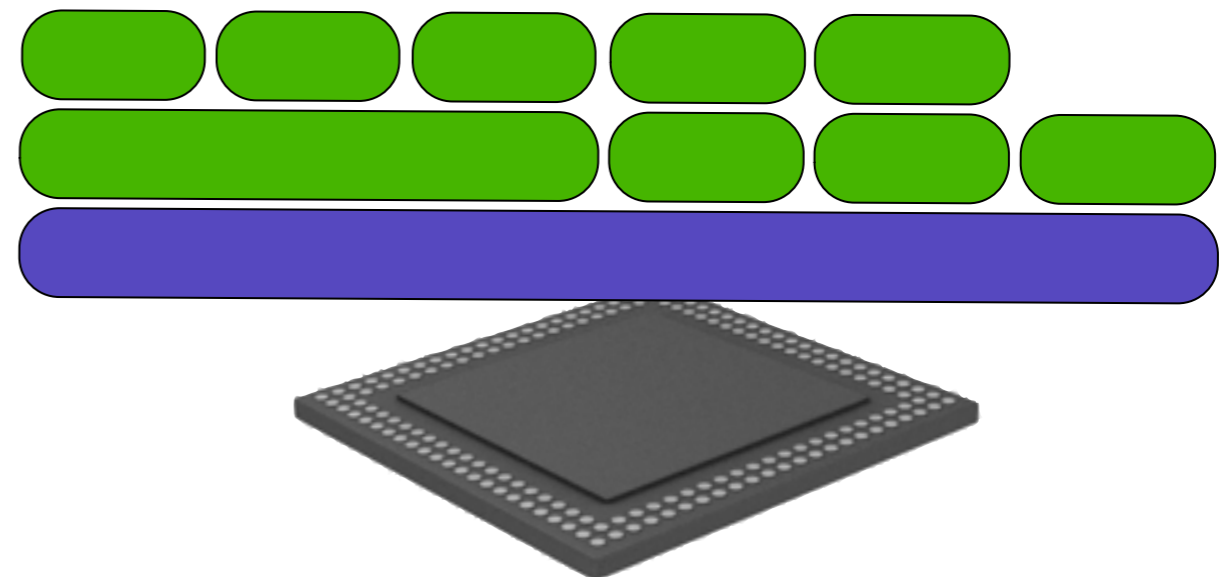
Hard real-time, consolidated!

- Advantages:

- Reduced hardware complexity and cost
- Simpler communication between components
- Easier to debug a single entity

- Disadvantages:

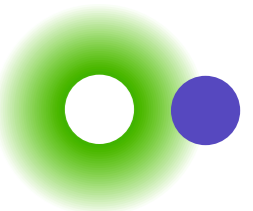
- Single point of failure (the trusted OS kernel)
- Less predictable timing behaviour



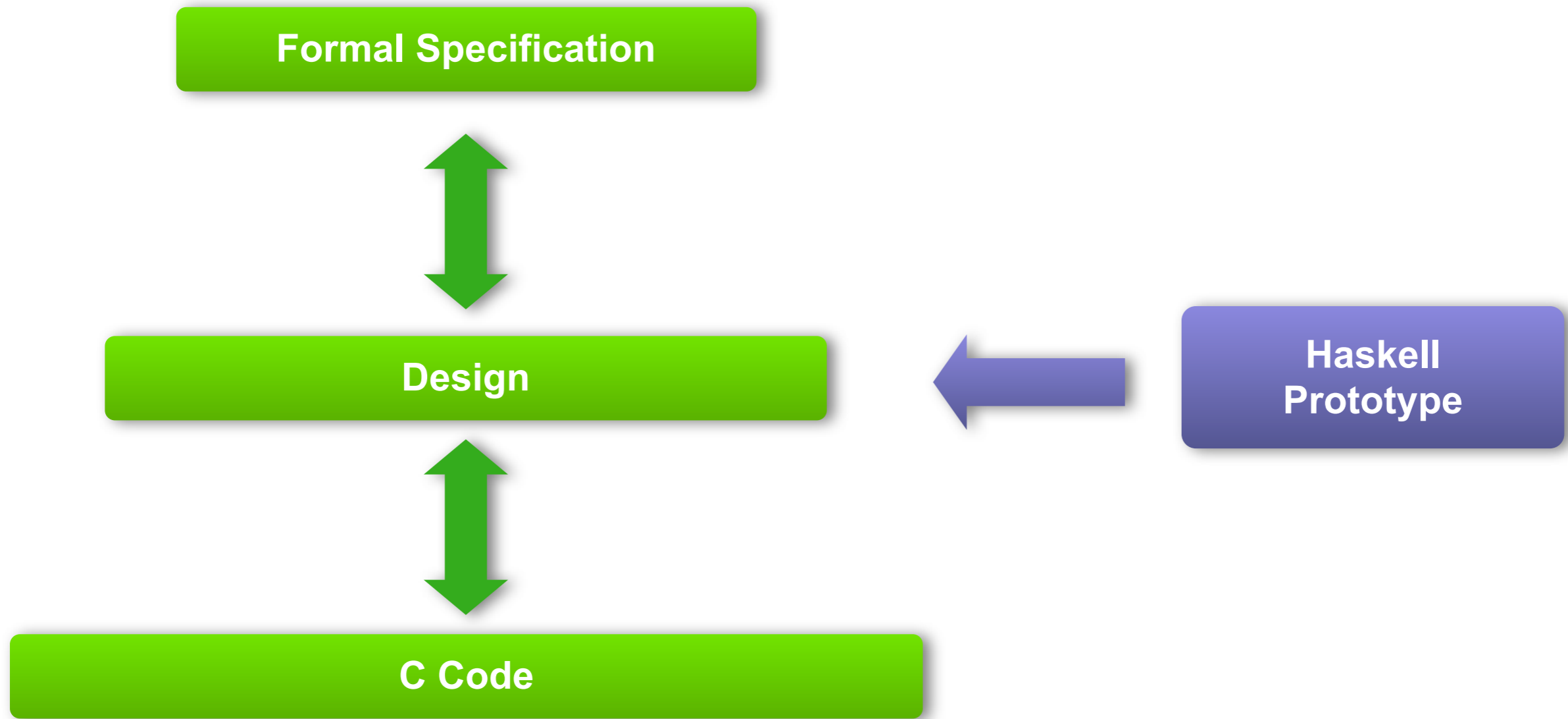
Mitigating the issues

- seL4 microkernel gives *trustworthiness* using
 - MMU-based isolation
 - Small trusted code base (TCB)
- ★ Formal specification of functional behaviour
- ★ Machine-checked formal proof of compliance to specification

[Klein et al, SOSp'09]



seL4: a formally verified μ -kernel



Proofs with Benefits!

- **Execution always defined:**

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

- **Not implied:**

- zero bugs from expectation to physical world

Mitigating the issues

- seL4 microkernel gives *trustworthiness* using
 - MMU-based isolation
 - Small trusted code base (TCB)
 - Formal specification of functional behaviour
 - Machine-checked proof of compliance to specification
[Klein et al, SOSp'09]
- For hard real-time systems we also want **predictable temporal behaviour, i.e.,**
 - Interrupt latency guarantees
 - Timing guarantees on OS requests

Mitigating the issues

- seL4 microkernel gives *trustworthiness* using
 - MMU-based isolation
 - Small trusted code base (TCB)
 - Formal specification of functional behaviour
 - Machine-checked proof of compliance to specification [Klein et al, SOSp'09]
- For hard real-time systems we also want predictable temporal behaviour, i.e.,
 - Interrupt latency guarantees
 - Timing guarantees on OS requests

Mitigating the issues

- seL4 microkernel gives *trustworthiness* using
 - MMU-based isolation
 - Small trusted code base (TCB)
 - Formal specification of functional behaviour
 - Machine-checked proof of compliance to specification [Klein et al, SOSp'09]
- For hard real-time systems we also want predictable temporal behaviour, i.e.,
 - Interrupt latency guarantees
 - Timing guarantees on OS requests

SOLVED!

???

Why is seL4 suited to WCET?

- Small code base

- 8,700 LoC

➔ Allows more in-depth analysis

- Clean code base

- No function pointers

- Exception-free code

➔ Simplifies analysis



Why is seL4 suited to WCET?

- Static mapping of all kernel memory
 - Event-based design – single kernel stack
- ➔ Context switching is straightforward to analyse

```
void switchToThread(tcb_t *thread) {  
    ...  
    ksCurThread = thread;  
}
```

Why is seL4 suited to WCET?

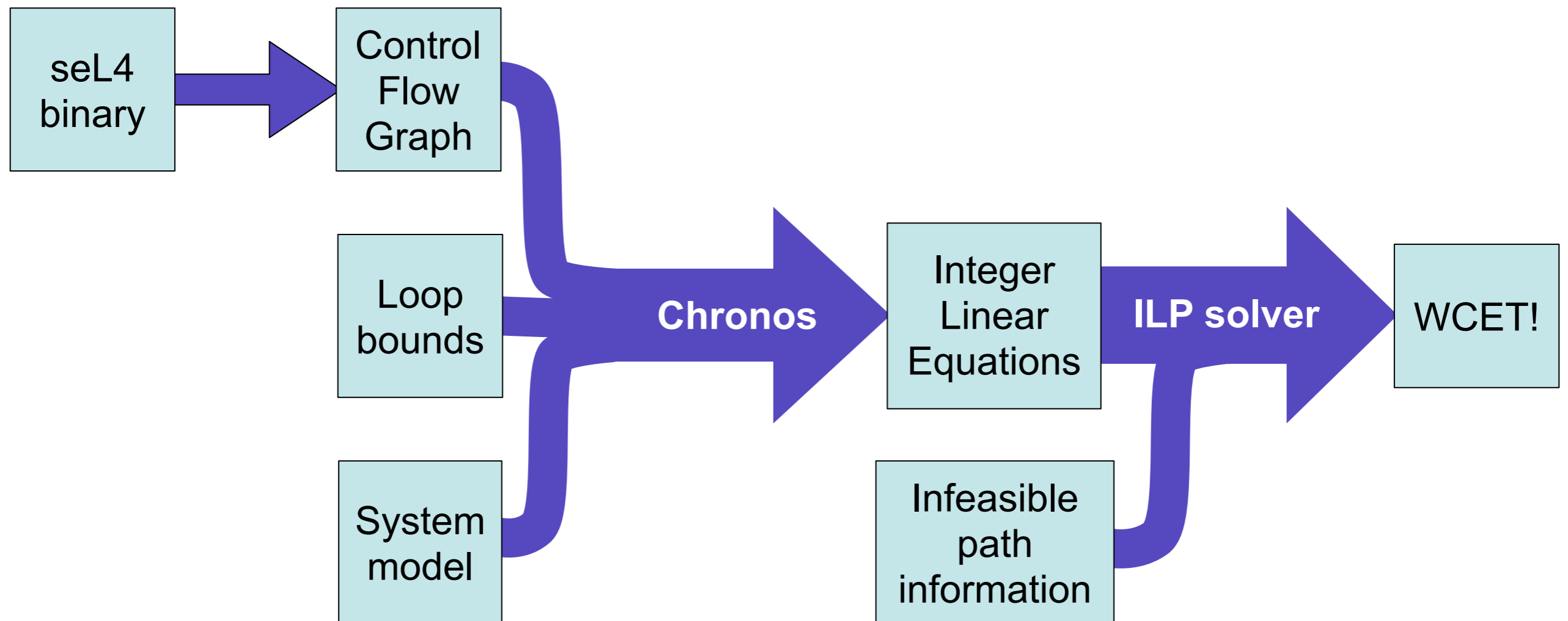
- Explicit preemption points, interrupts disabled elsewhere

```
void delete_children() {  
    for (...) {  
        ...  
  
        if (irq_pending())  
            return EXCEPTION_PREEMPTED;  
    }  
    ...  
}
```

What? Not fully preemptible?

- Well-placed preemption points reduce latency
- Embedded CPU speeds getting faster
- Controlled preemption enables formal verification
- No locks = better average case performance
= longer battery life!

Analysis method

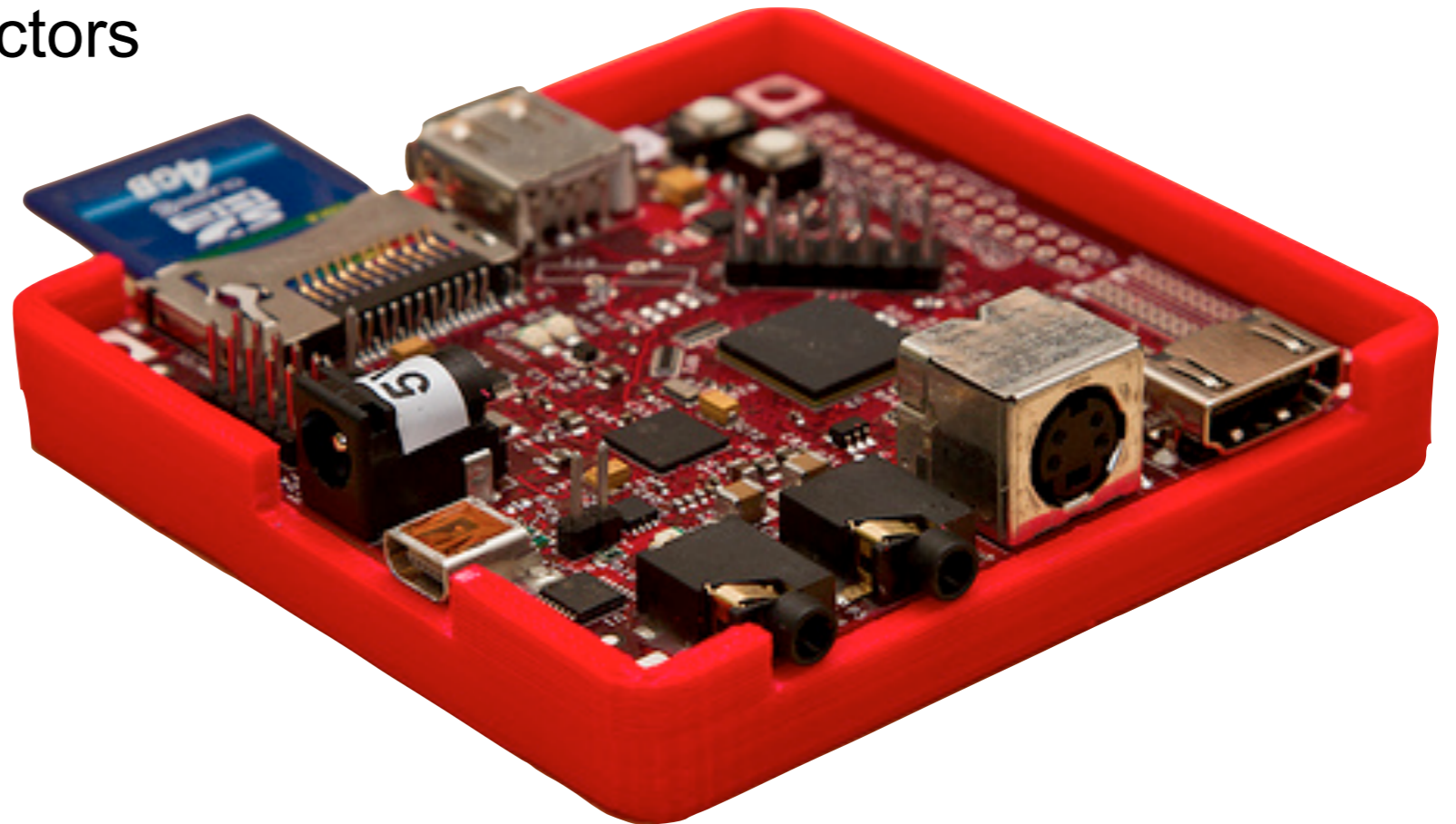


Inside Chronos (4.1)

- Virtually inline function calls
- Unroll first iteration of loops
- Compute WCET of each basic block using:
 - Micro-architectural modelling of the CPU pipeline
 - Scope-aware must/may analysis to determine cache hits/misses
- Generate equations for each basic block and edge

Evaluation Platform

- OMAP3-based BeagleBoard-xM
 - ARM Cortex-A8 @ 800 MHz
 - 128 MB memory
 - 32 KB 4-way set-associative L1 instruction cache
 - 32 KB 4-way set-associative L1 data cache
 - Disabled branch predictors
 - Disabled L2 cache



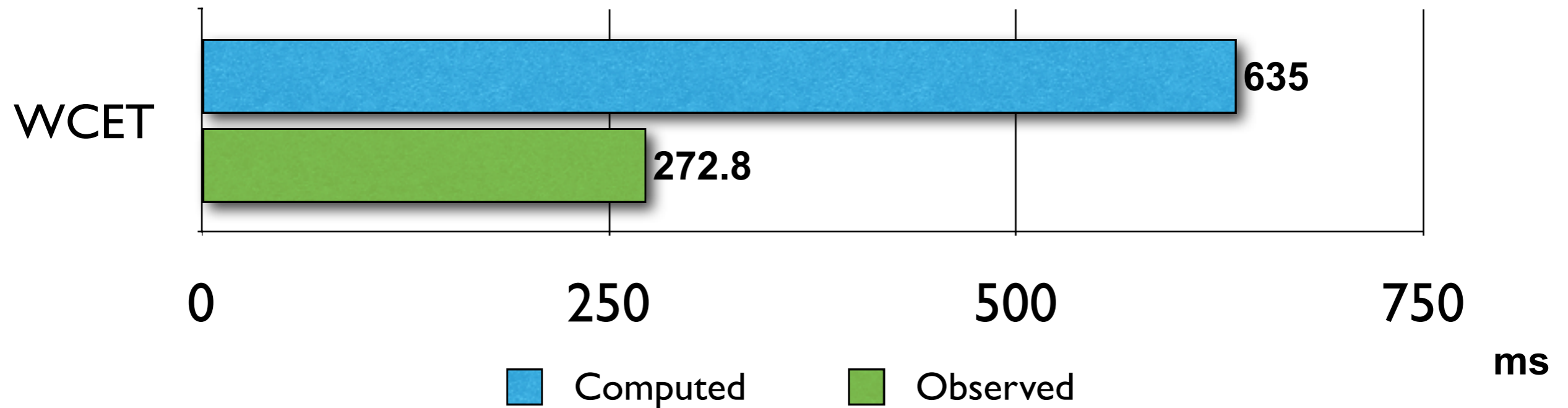
Cortex-A8 is “fun”

- Broadly:
 - 13-stage pipeline
 - dual-issue pipeline (for select instruction pairs)
 - memory access: 100 cycles
 - branch: 13 cycles
- Incomplete & inaccurate documentation
- Random cache replacement policy
 - We need to model caches as direct-mapped equivalent of a single way (i.e. 8 KB, direct-mapped)

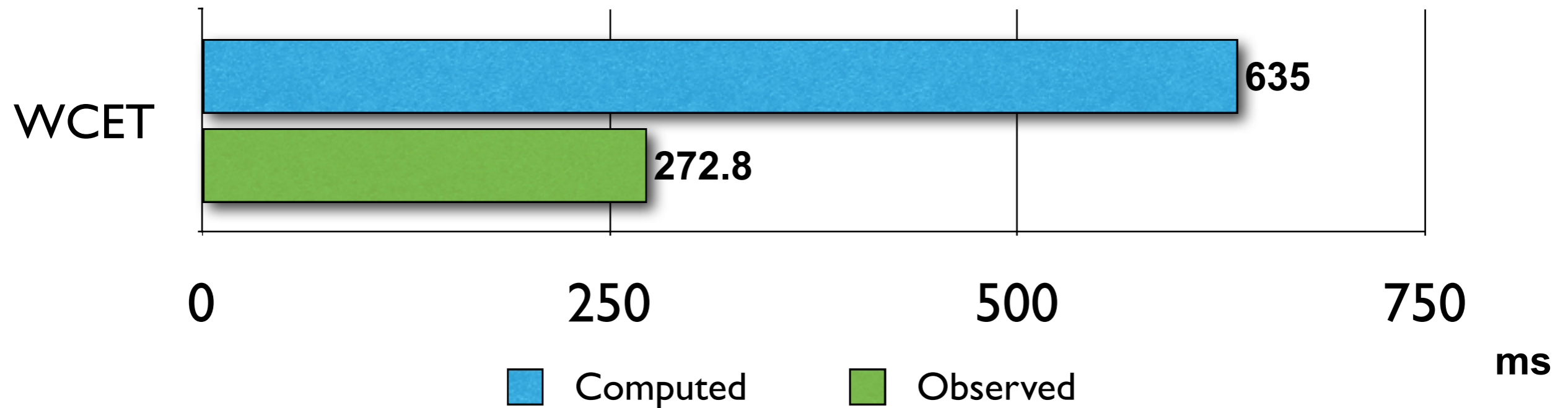
Analyzing seL4

- Five entry points into seL4:
 - System call: generally has the largest WCET
 - Unknown system call
 - Undefined instruction
 - Page fault
 - Interrupt
- Interrupts dispatched to userspace threads
- Worst-case interrupt latency =
 $WCET(*) + WCET(\text{Interrupt dispatcher})$

What went wrong...



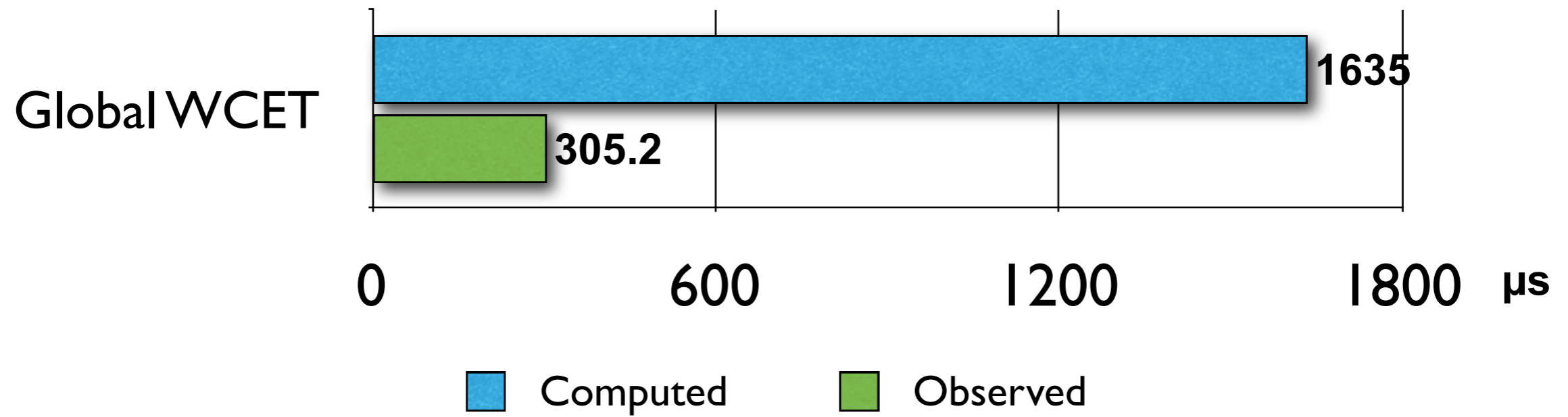
What went wrong...



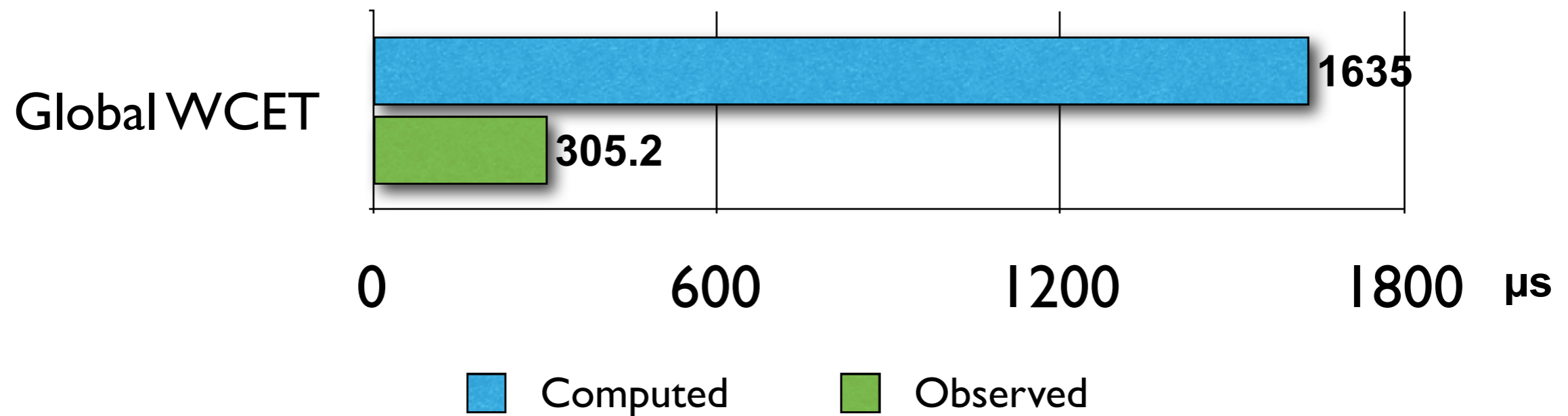
Lazy scheduling:

- Great for average case!
- NOT a suitable optimization for minimising WCET

Results



Results



Object deletion:

deleting an ASID pool

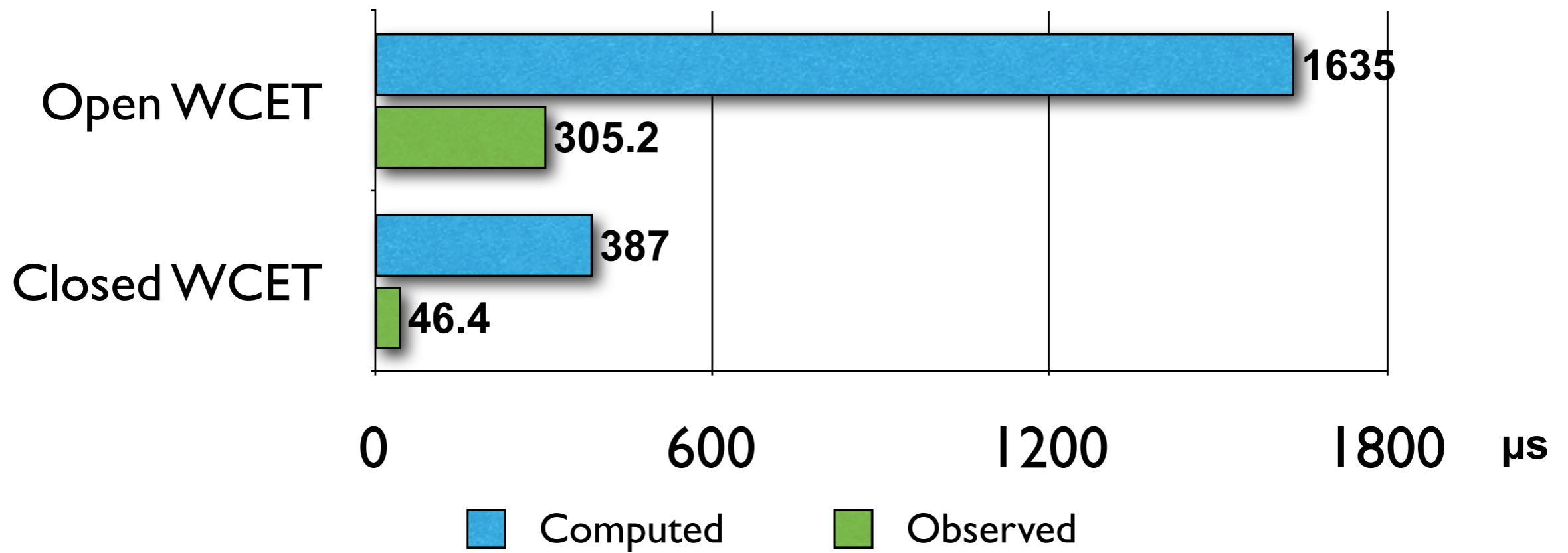
- Malicious entity with sufficient privileges could force this scenario

Open vs Closed Systems

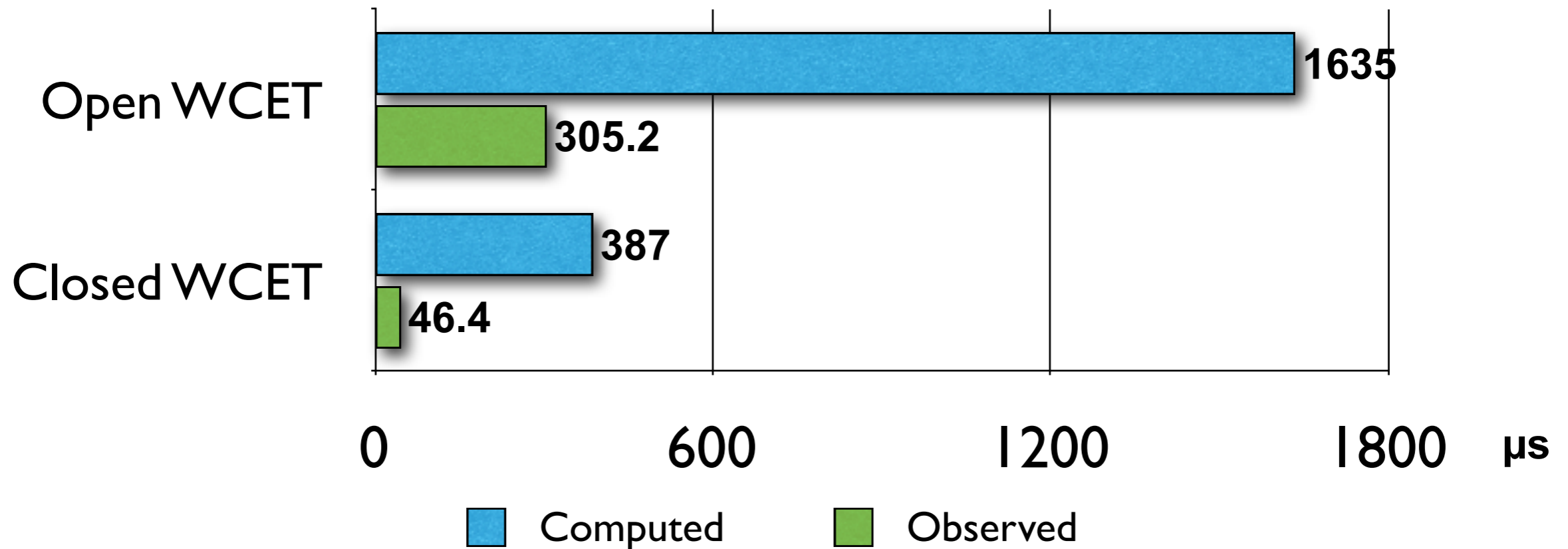
- **Open systems:**
 - Permit anything!

- **Closed systems:**
 - Assume no object creation or deletion at run-time
 - Easily enforced with seL4 authority model
 - Still supports dynamic systems (e.g. virtualized Linux)

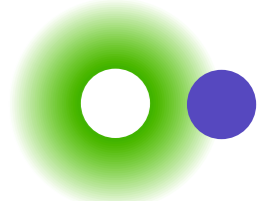
Results



Results

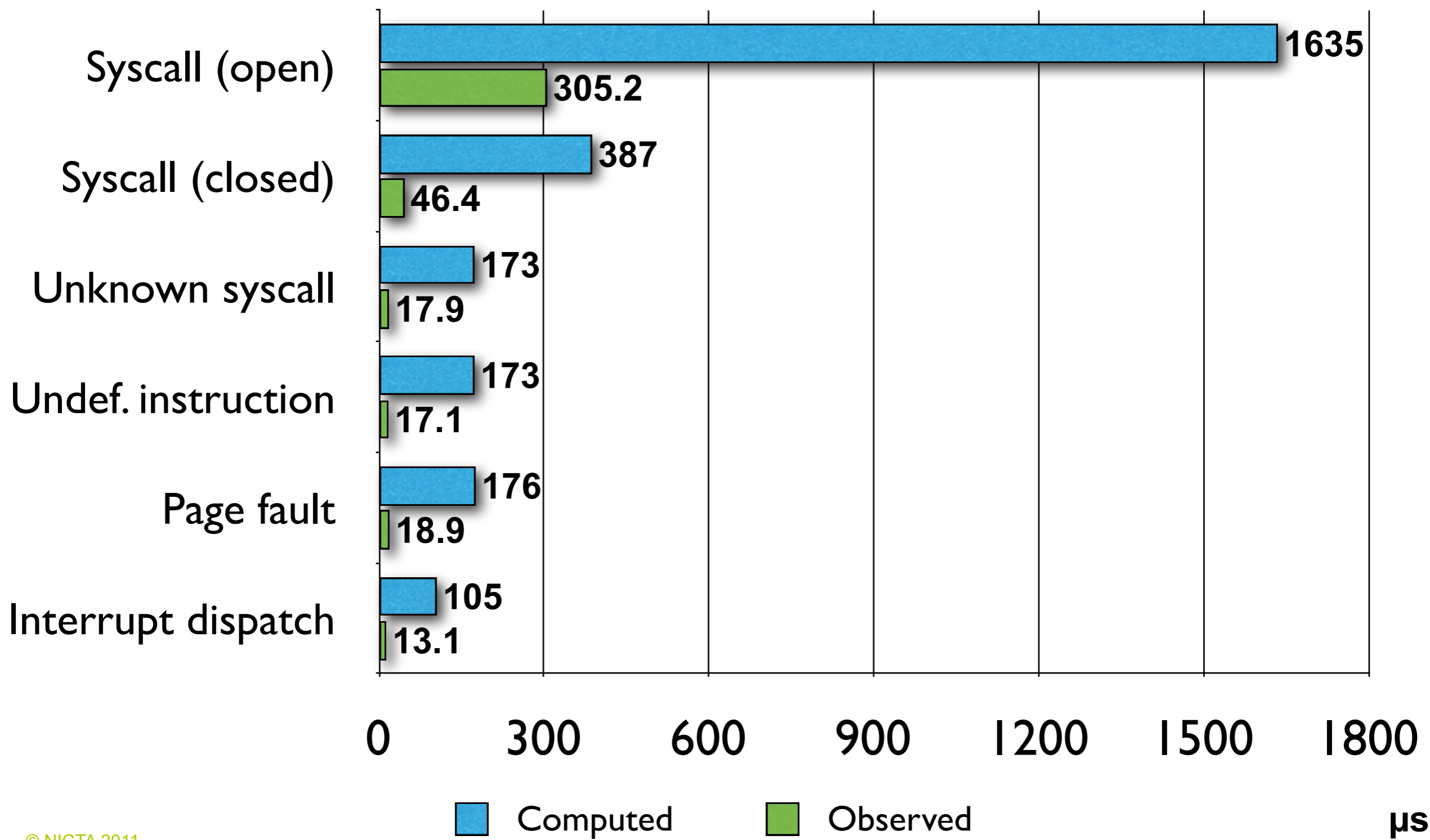


- **Avoiding deletion, WCET bounded by IPC itself**
 - Maximum length message transfer (120 words)
 - Transfer of access rights to kernel objects

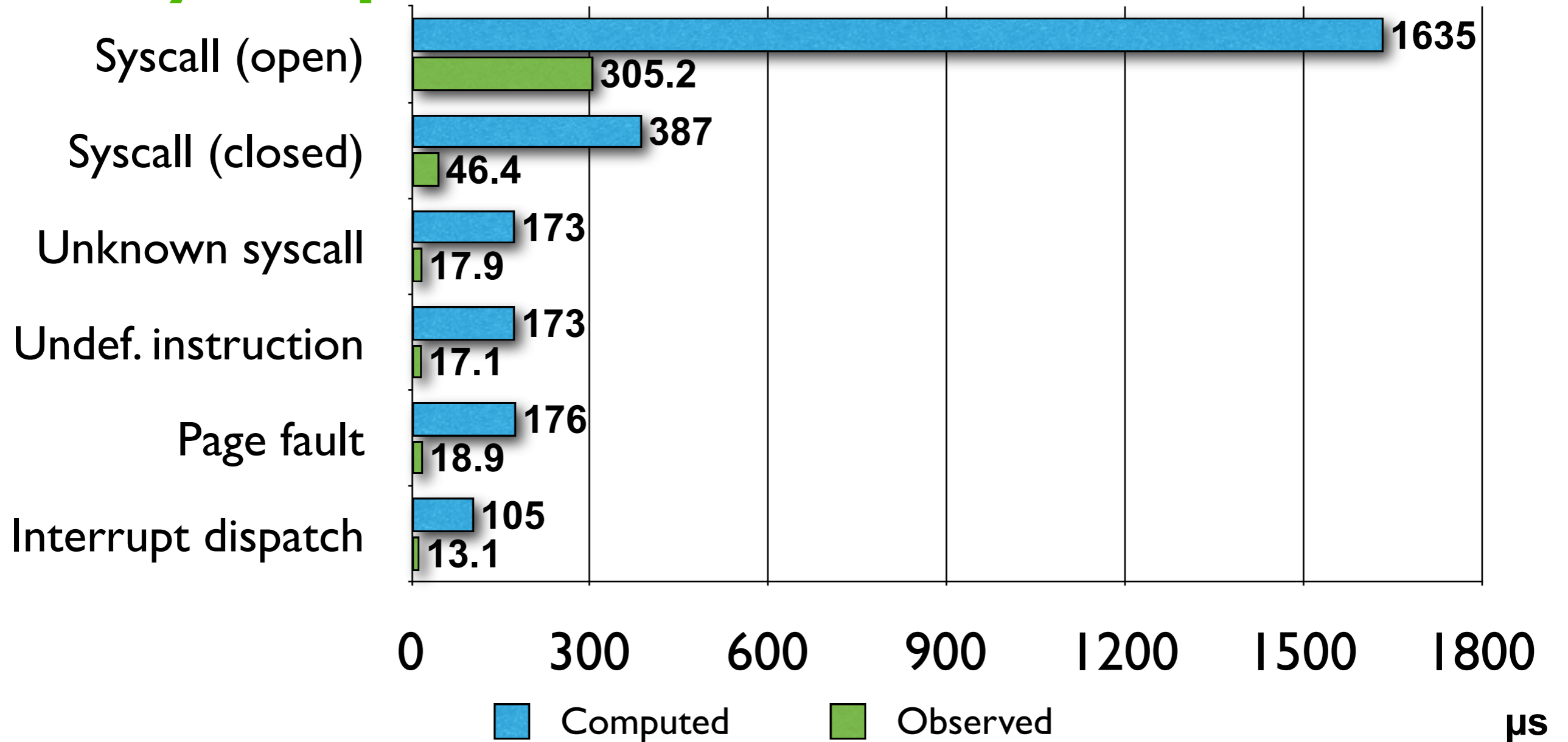


NICTA

Results



Why so pessimistic?



~ 5x : Model pessimism (caches, pipeline)

1.5-2x : Infeasible paths

Future directions

- How can these results be improved by leveraging proof invariants?
- How can we reduce latency without compromising verifiability?
- How can seL4 be improved for real-time applications?

In summary...

- Mixed-criticality hard real-time systems require a *trustworthy* platform to build upon
- Functional + timing guarantees require thoughtful design
- Consolidate to regain your sanity!
 - Save \$\$\$ in psychologist bills (and hardware/development costs).

Bernard.Blackham@nicta.com.au

<http://ertos.nicta.com.au/>