

The Role of Language Technology in Trustworthy Operating Systems

Gernot Heiser NICTA and University of New South Wales Sydney, Australia



Australian Government

Department of Broadband, Communications and the Digital Economy

Australian Research Council



THE UNIVERSITY OF SYDNEY



Queensland

NICTA Funding and Supporting Members and Partners



Griffith







PLOS'11 Keynote

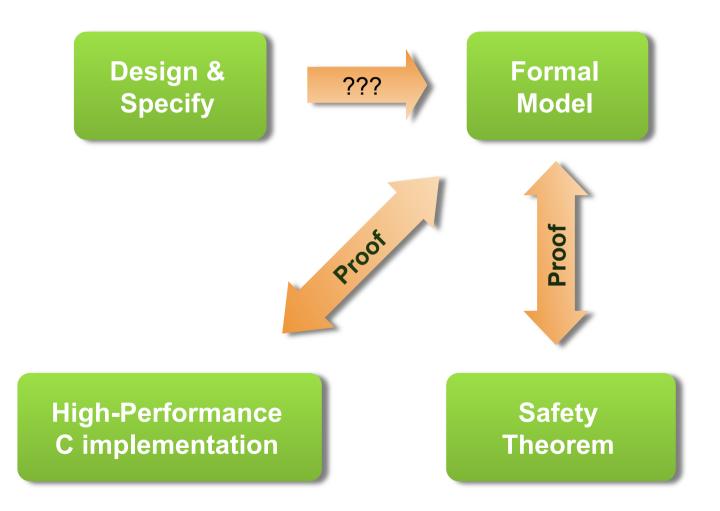




LANGUAGES AS TOOLS

seL4 Aim: Formal Verification





Clash of Mentalities





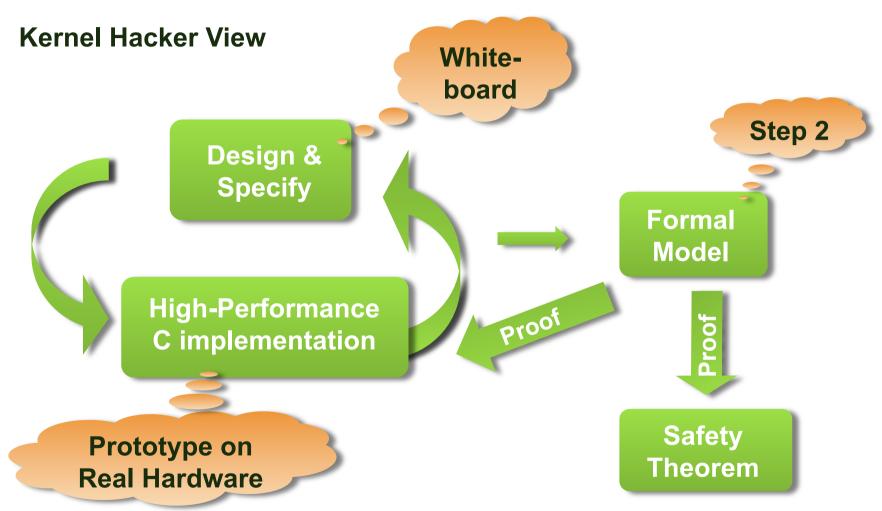


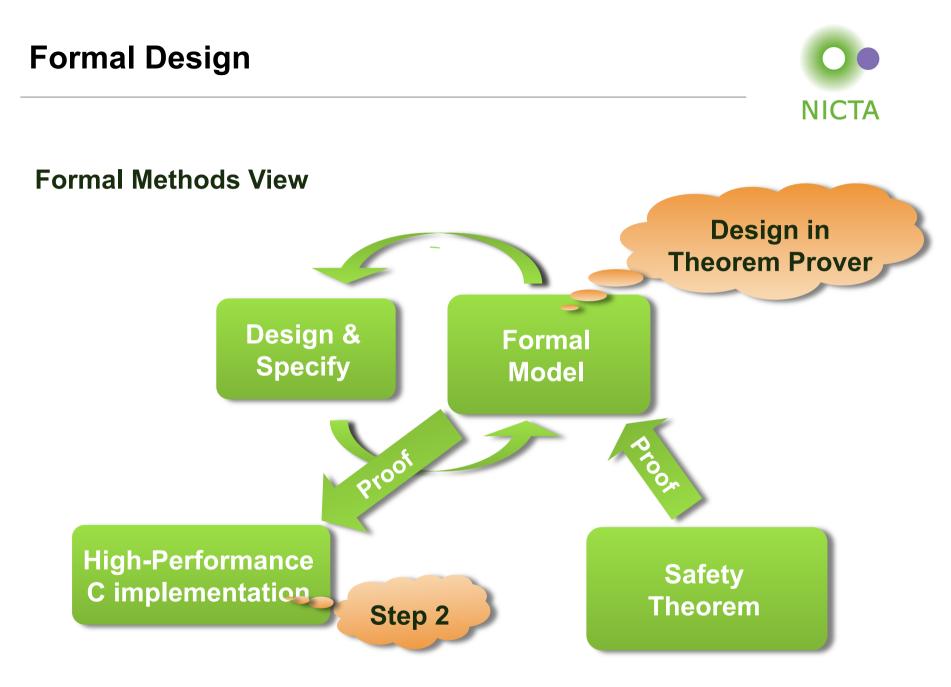
Formal Methods Practitioner

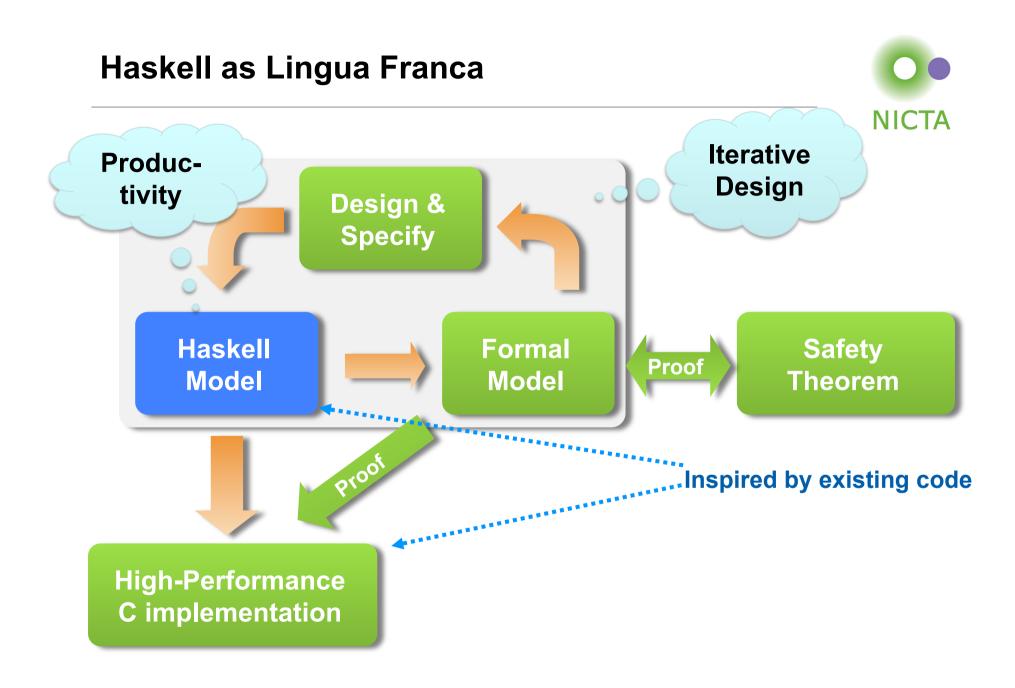
Kernel Hacker

Standard Kernel Design



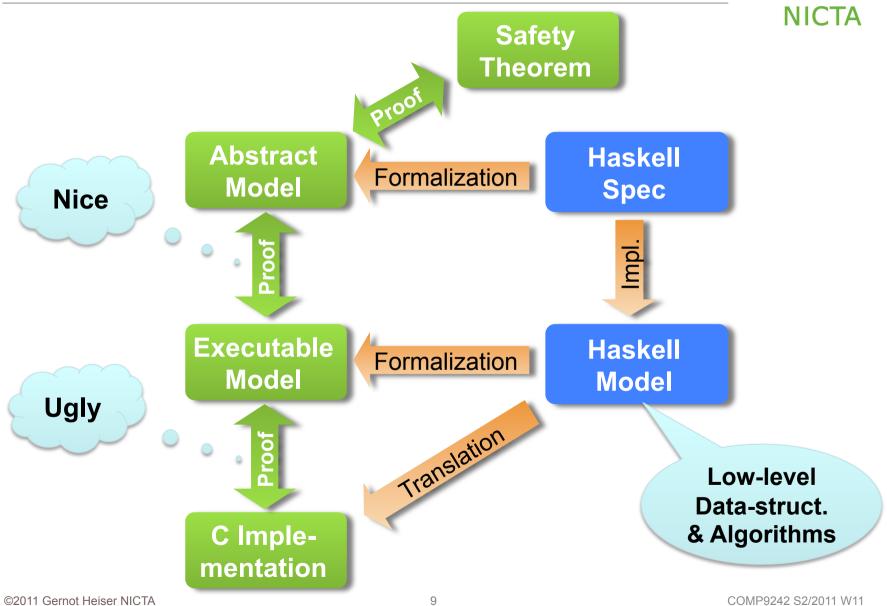






Haskell Model as Executable Prototype **NICTA Built with** Standard **Test App** Toolchain **Binary** Execute Syscall Haskell QEMU Model Return Custom Interface

Haskell Model as Intermediate Refinement







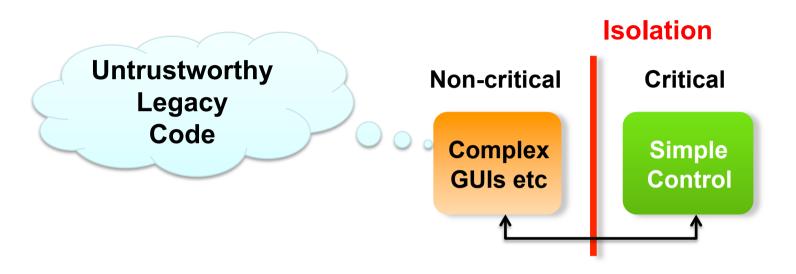
LANGUAGES FOR TRUSTWORTHINESS

Trustworthiness NICTA **Trustworthiness** Security Safety Reliability

These are full-system properties!

Real-World Trustworthiness





Controlled communication

Prerequisites: Isolation, communication and legacy support!

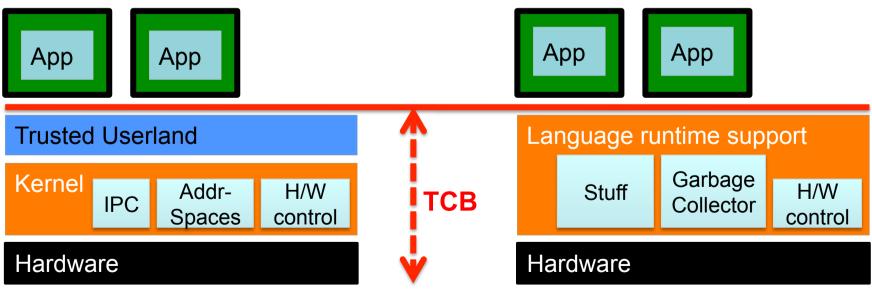
Two Approaches to Isolation

MMU-enforced protection

- Kernel:
 - controls HW
 - IPC for communication
 - Address spaces for isolation

Type Safety

- Language runtime
 - controls HW
 - manages memory
 - ...?





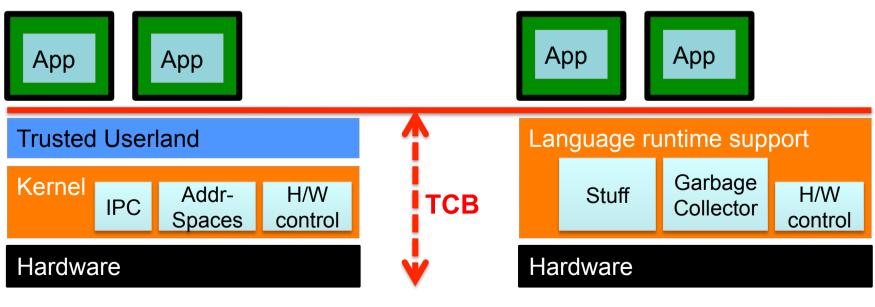
Representative Systems

MMU-protected: L4

- State-of-the-art microkernels
 for 18 years
- IPC performance still unbeaten
 - lots of published data
- Widely deployed:
 - OKL4 on 1.2 billion devices

Type-safe: Singularity

- Most complete recent system
- Some published performance
 - Surprisingly no L4 comparison!





Cost of Isolation

MMU-enforced protection

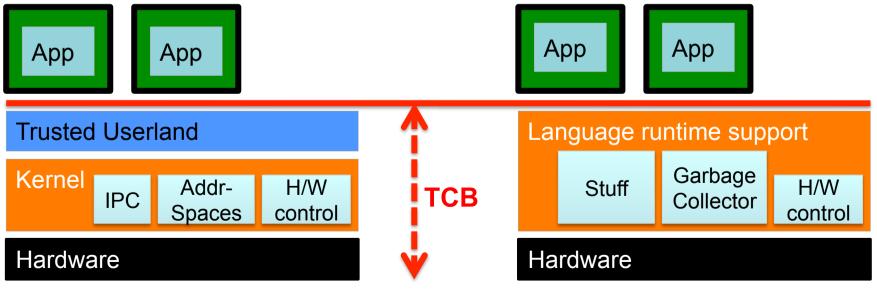
- Context switching
 - thread context
 - protection context
 - IPC semantics
- Other execution at full speed

Large per-switch overhead

Type Safety

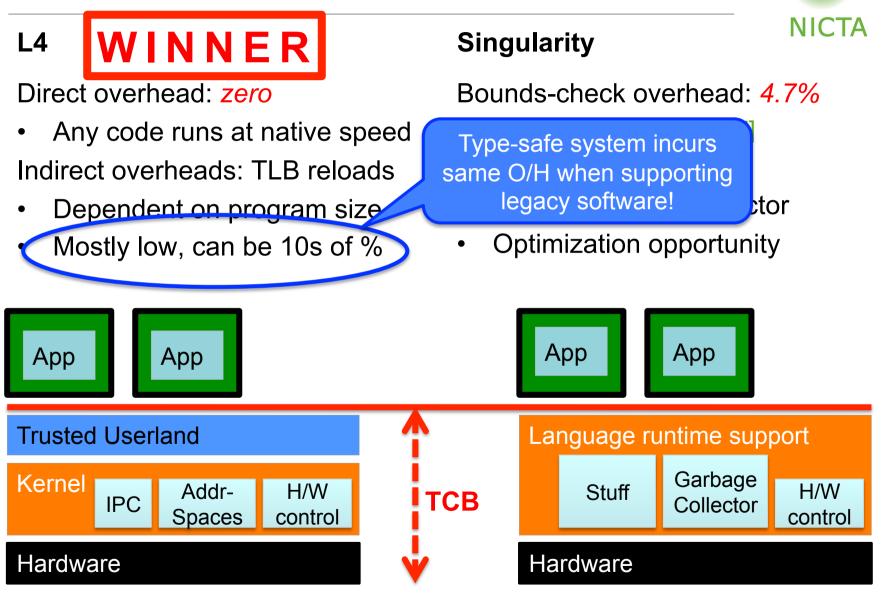
- Run-time bounds checks
- Garbage collection
- Switching is just function call

Small continuous overhead





Performance: Intra-Domain Execution



Performance: Cross-Domain IPC





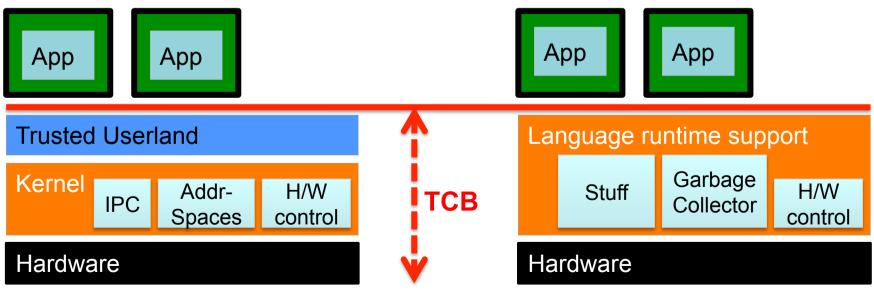
AMD-64 @ 1.6 GHz:

 230 cycles for 0–24 bytes [http://www.l4ka.org/126.php]

Singularity

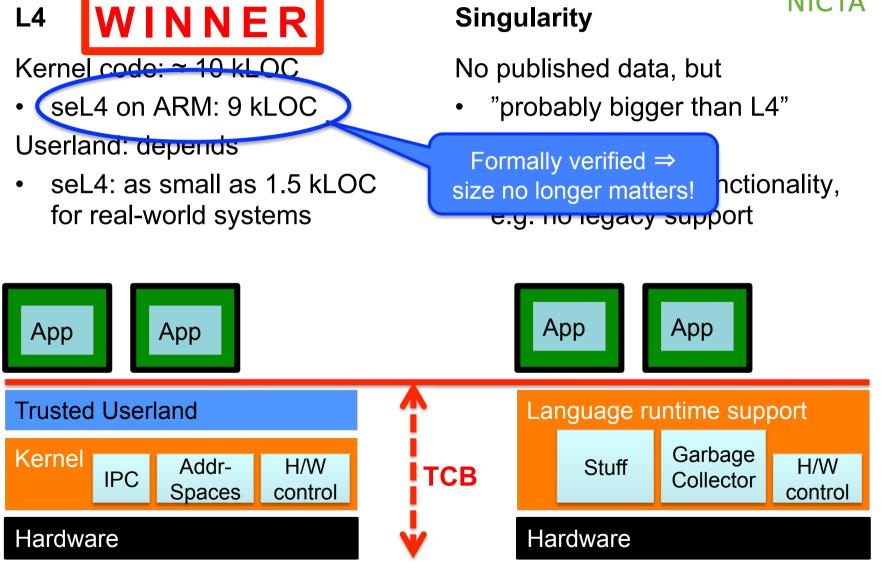
AMD-64 @ 2.0 GHz

- 803 cycles for 1 byte
- 933 cycles for 4 bytes [Hunt & al, EuroSys'07]



TCB Size





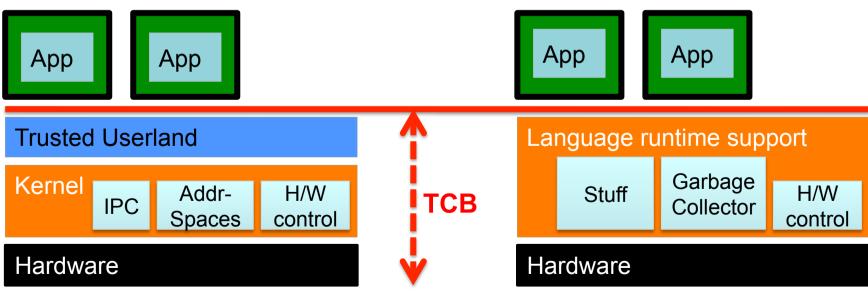
Summary

MMU-enforced protection

- Faster
- Probably smaller TCB
- Functionally-correct

Type Safety

• ???





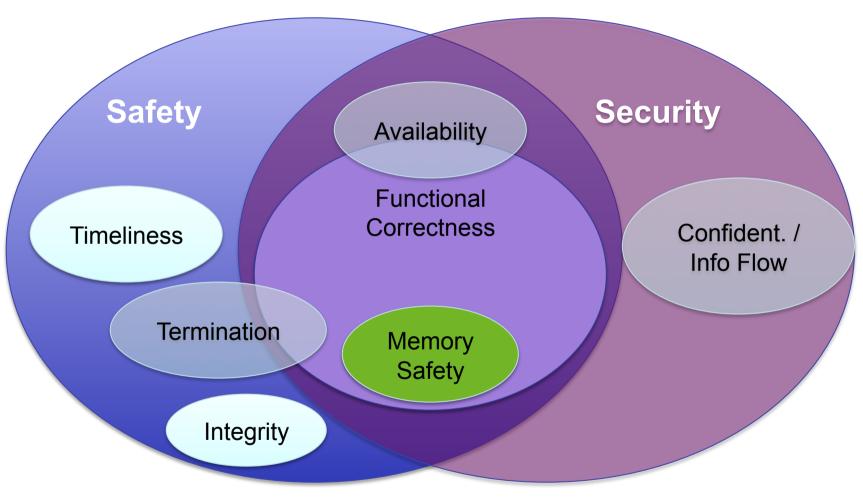
Does Memory-Safety Help Safety or Security?

- It's better than nothing
 - ... but on its own it doesn't help much in *proving safety*
- Type safety doesn't stop:
 - your garbage collector being buggy
 - possibly destroying type safety
 - your scheduler being buggy
 - leading to unsafe thread execution order
 - leaks information through scheduling decisions
 - your IPC primitive having unsafe side effects
 - affecting or leaking data to third threads



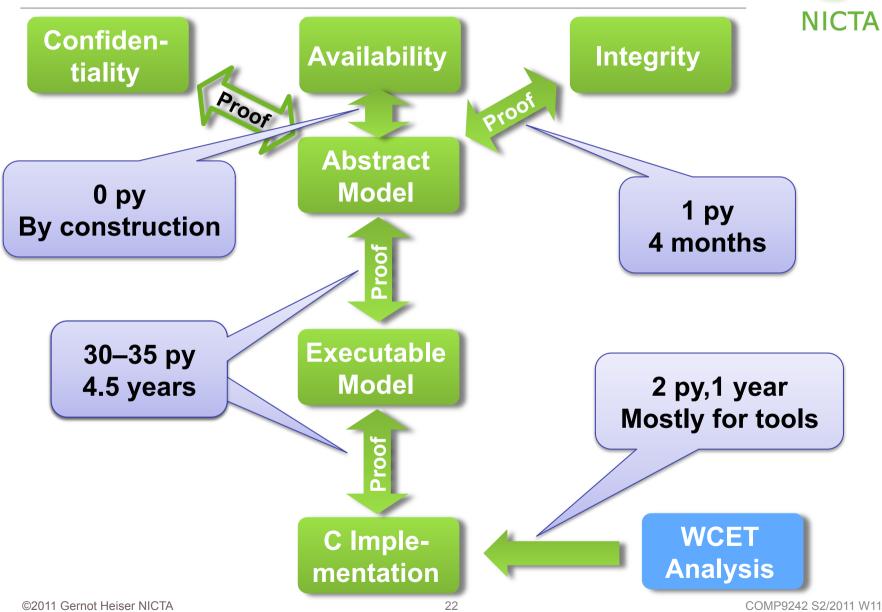
Safety and Security





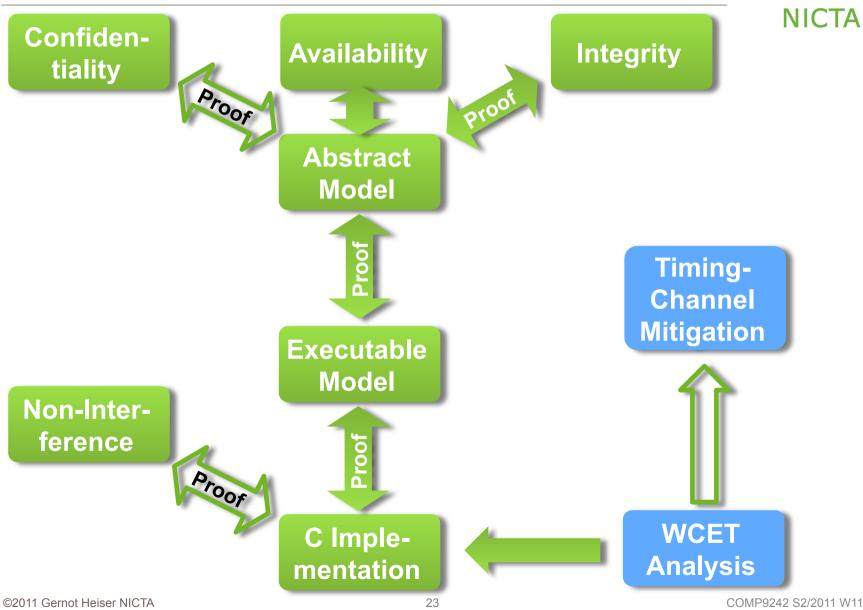
The seL4 Experience





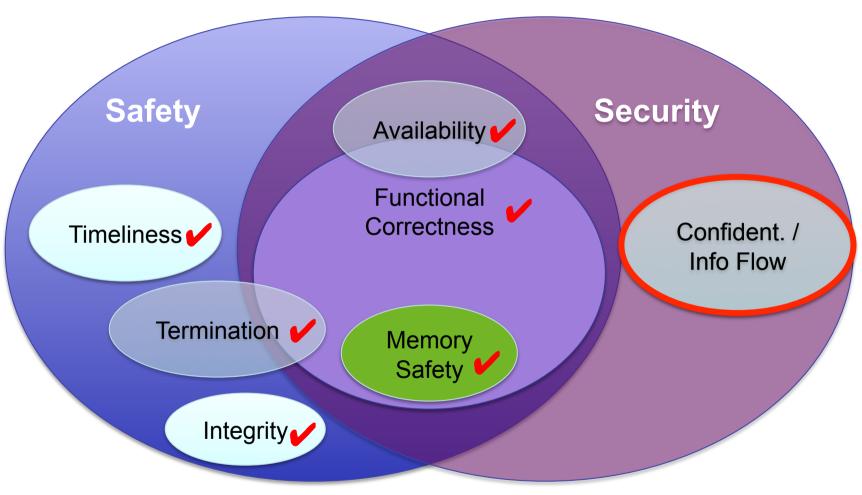
seL4: Next 12 Months





seL4 for Safety and Security





seL4 Summary



- First (and still only) general-purpose OS kernel with
 - Functional correctness proof
 - Integrity proof
 - Complete, sound WCET analysis
- Yet, performance at par with any comparable system!
 - 200 cycle IPC on ARM11
- Likely to be the first kernel with
 - Confidentiality proof
 - Non-interference proof
 - Sound covert-channel mitigation

Let's Stop Kidding Ourselves

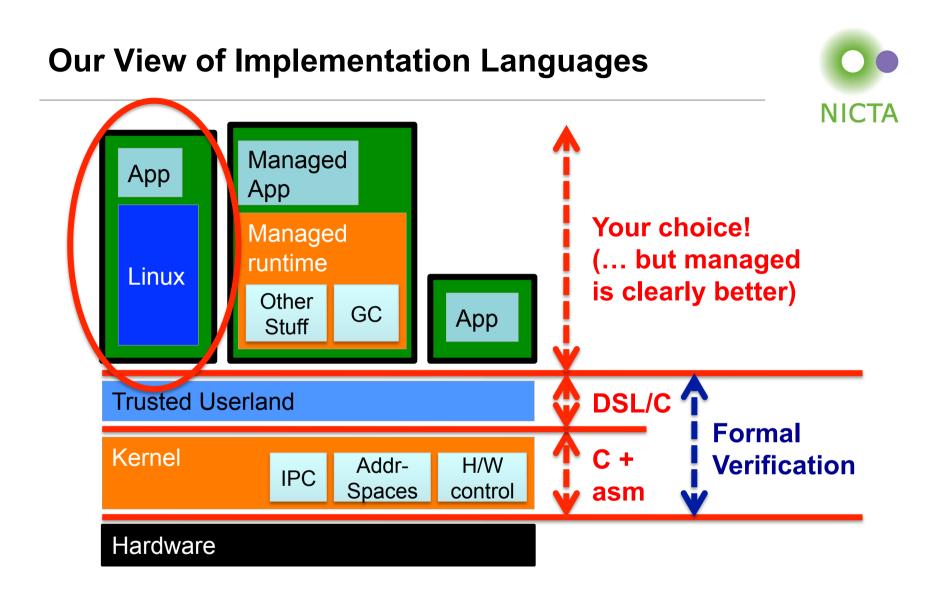


... and the people who trust our expertise!

- By implying that type-safe = safe
 - Type-safe ≪ safe; type-safe ≪ secure
 - ... and there's no easy way to get there
- By implying that a system where *all* code is managed is practicable
 - Nothing will be used if it can't provide legacy support
 - Test: Can it run Linux?

Trustworthiness is best achieved through functional correctness!

• Excellent basis for showing integrity and confidentiality



mailto:gernot@nicta.com.au

Google: "ertos"