COMP9242
Advanced Operating Systems

S2/2013 Week 12:
Local Systems Research

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

NICTA Funding and Supporting Members and Partners

Copyright Notice

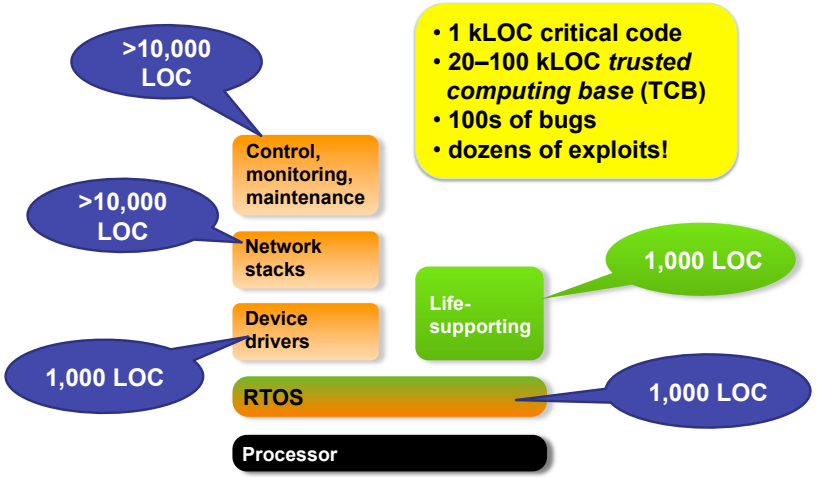
Present Systems are *NOT* Trustworthy!

Yet they are expensive:
- $1,000 per line of code for "high-assurance" software!

What's Next?

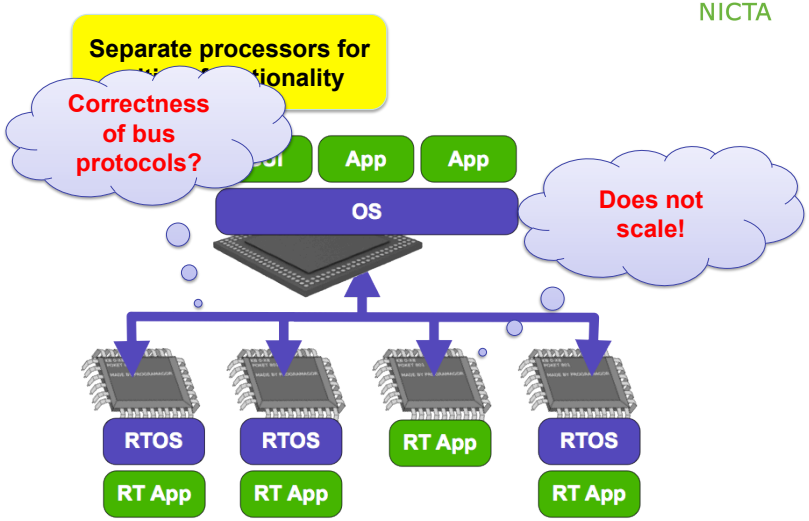
Fundamental issue: large stacks, need isolation

E.g. medical implant



- 1 kLOC critical code
- 20–100 kLOC *trusted computing base* (TCB)
- 100s of bugs
- dozens of exploits!

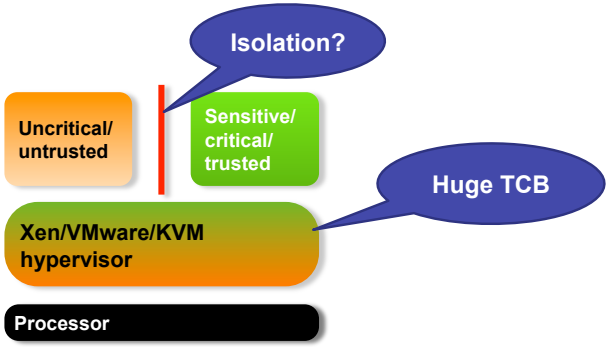
Dealing with Complexity: Physical Isolation



High Assurance *Bad* Practice

- TCB of millions of LOC
- Expect 1000s of bugs
- Expect 100s of vulnerabilities

Hacker's delight!

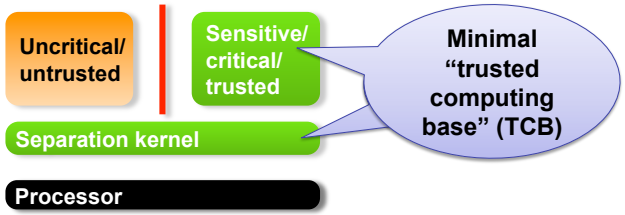


High Assurance Best Practice

NICTA

- Isolate
- Minimise the TCB
- Assure TCB by
  - testing
  - code inspection
  - bug-finding tools

Always incomplete!



NICTA

So, why don't we prove trustworthiness?

Claim:
A system must be considered *untrustworthy* unless *proved* otherwise!

Corollary [with apologies to Dijkstra]:

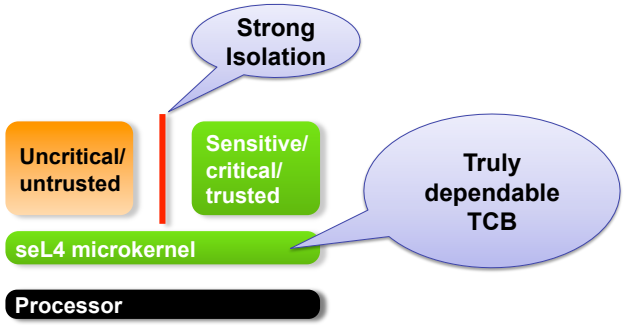
Testing, code inspection, etc. can only show *lack of trustworthiness*!

State of the Art: NICTA's seL4 Microkernel

NICTA

- Provable isolation!
- Provable assurance!

No place for bugs to hide!



Our Vision: Trustworthy Systems

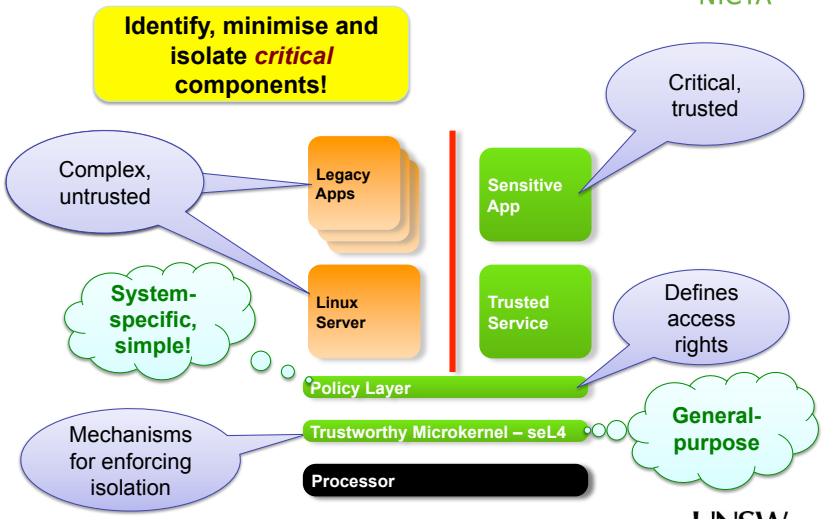
NICTA

Suitable for real-world systems

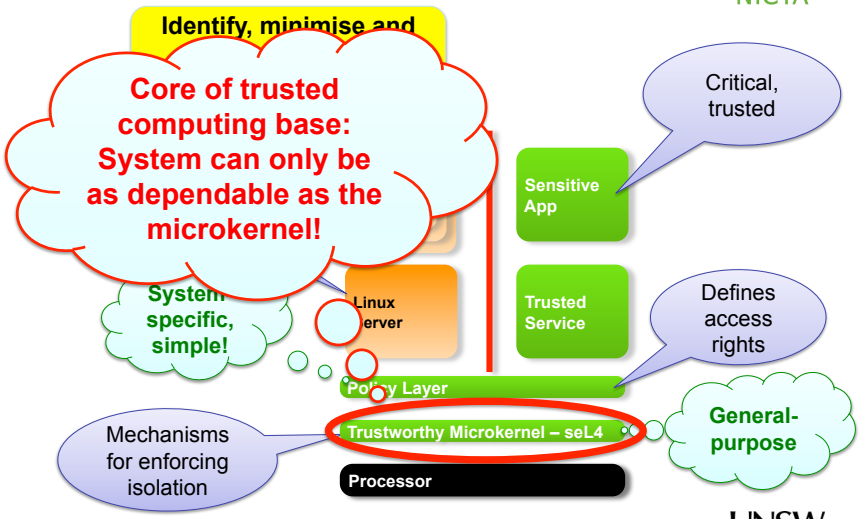
We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

Hard *guarantees* on safety/security/ reliability

Isolation is Key!



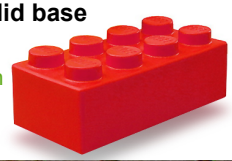
Isolation is Key!



NICTA Trustworthy Systems Agenda

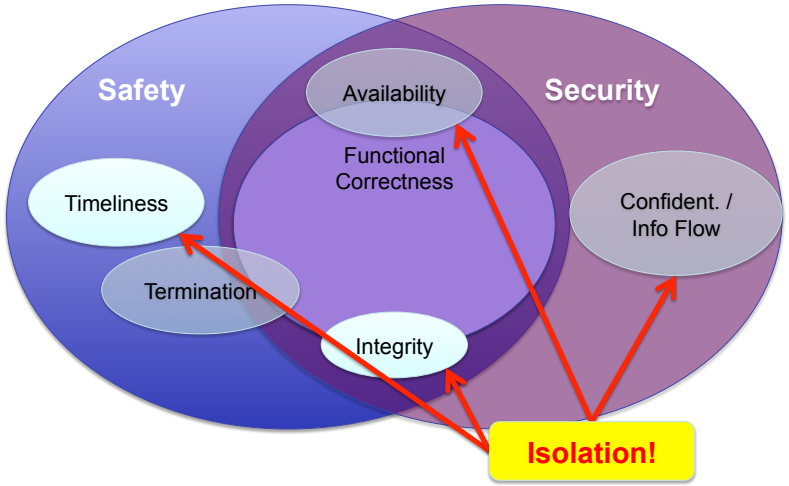
1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

DONE

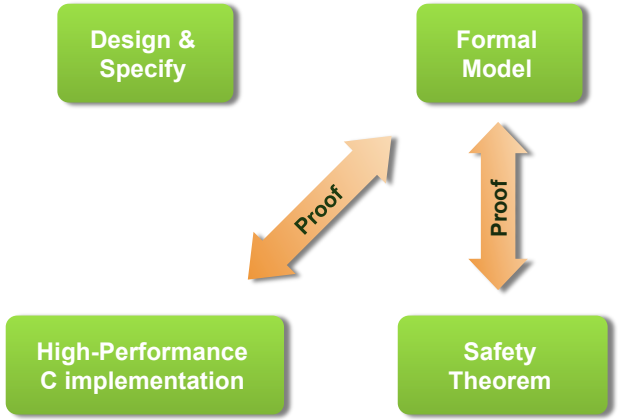


2. **Lift microkernel guarantees to whole system**
   - Use kernel correctness and integrity to guarantee critical functionality
   - Ensure correctness of balance of trusted computing base
   - Prove dependability properties of complete system
     - despite 99 % of code untrusted!

Requirements for Trustworthy Systems



seL4: Designing and Formalising

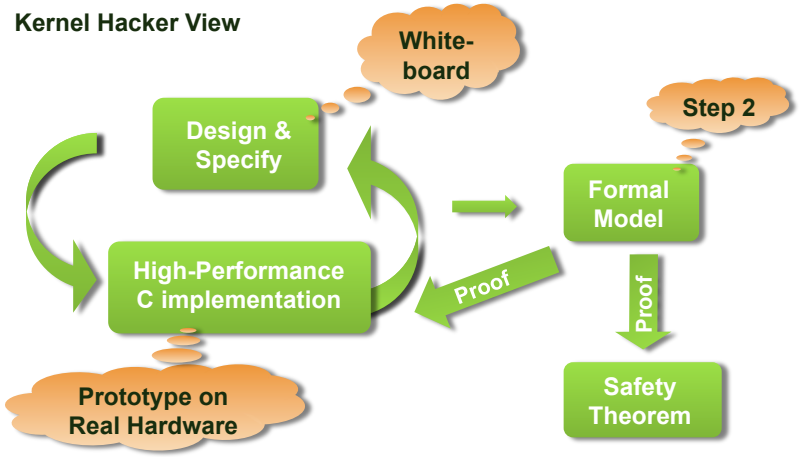


Two Mentalities

Formal Methods Practitioners Systems Researchers

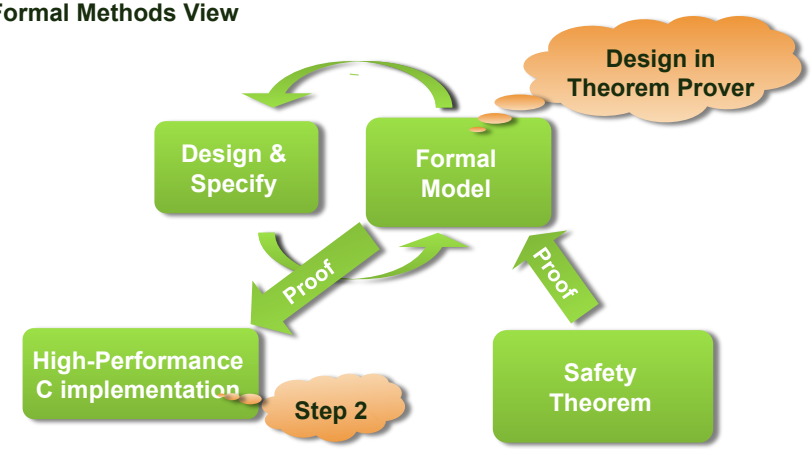
Standard Kernel Design

Kernel Hacker View

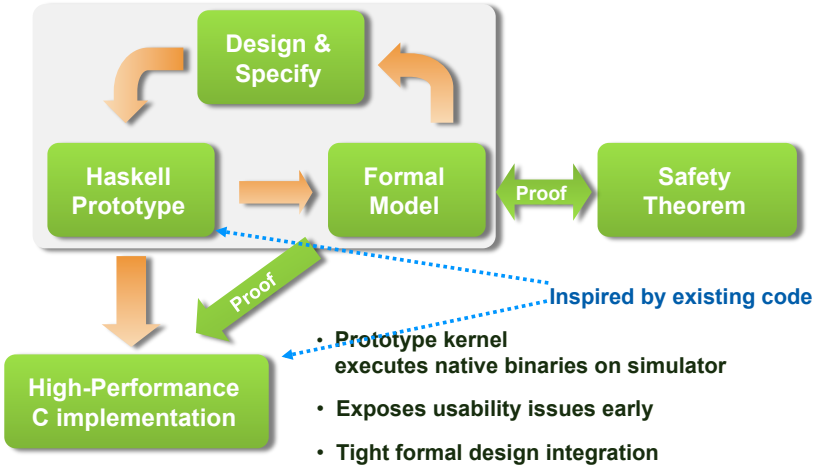


Formal Design

Formal Methods View



Iterative Design and Formalisation

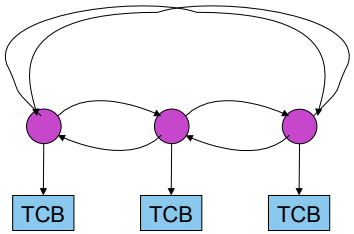


Kernel Design for Verification

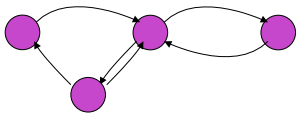
- Main objective: minimise complexity
  - global invariants must be proven for each state change
  - must prove pre- and post-conditions for statements/blocks
  - effort determined by complexity of conditions and state change
- … without sacrificing performance
- Affects design in many ways
  - global variables, side effects
  - kernel memory management
  - concurrency and non-determinism
  - I/O

Global Variables

- Not a difficulty per se, but potential source of complexity
- Eg: scheduler queue as doubly-linked list
  - Show that
    - all pointers are to valid nodes
    - front- and back-pointers are consistent
    - nodes point to TCBs

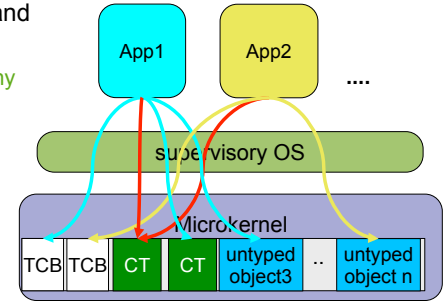


- Requires proof that any pointer operation maintains invariants
- Challenge is temporary violation
  - eg adding a node
  - Requires ensuring atomicity



Kernel Memory

- seL4 kernel memory management model pushes policy to userland
  - aids verification
  - need to ensure strict hierarchy
  - capability derivation tree
- Challenge is re-use
  - most difficult part of verification!
  - use derivation tree to detect all references
  - global data structure that requires invariants in all parts of the system



Concurrency

- Proofs about concurrent programs are inherently hard!
- seL4 strictly limits concurrency to the bare minimum
  - Single processor
    - multicore via big kernel lock or multikernel approach
  - User-level device drivers
  - Non-preemptible, event-based
    - single kernel stack
  - Interrupt points to limit real-time latencies
    - poll interrupt status
    - insert new kernel event (ahead of user)
    - return to user boundary and re-enter kernel
    - allows maintaining all invariants

Concurrency

Preempting object destruction:

- Keep one cap as zombie during object cleanup
  - only retained to reference partially cleaned-up object
  - stores state of cleanup, maintaining invariants
  - attempt by preemptor to remove zombie can just execute

Exceptions in kernel:

- Prevent memory exceptions
  - ensure kernel page tables are complete
  - map into every address space
- Disallow other exceptions
  - verification is its own friend ☺

I/O

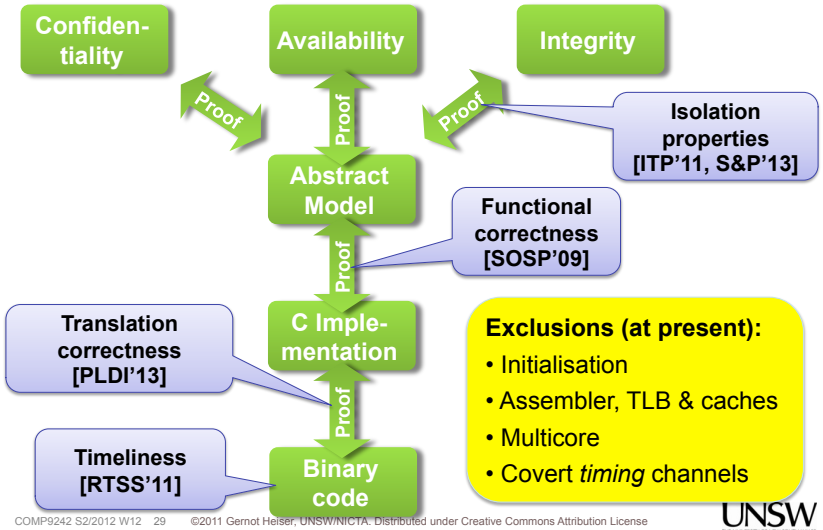
- Mostly a non-issue
  - user-level drivers
    - IOMMU support for DMA security
  - non-preemptible kernel
- Exception is timer tick
  - essentially a source of interrupts
  - handled in-kernel as separate event
  - no real complication

Lessons for Kernel Design

- Need to reduce complexity forced simple and clean design
  - beneficial even with traditional validation
  - does not necessarily impact performance
- Some design decision beneficial for other reasons too
  - single kernel stack for memory footprint
  - interrupt handling by polling has performance advantages

NICTA's seL4: Mathematical *Proof* of Isolation

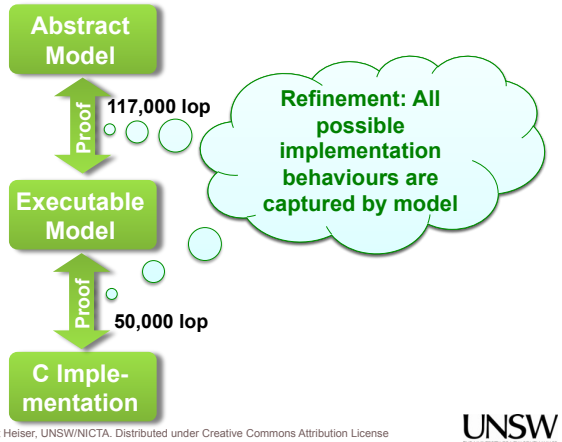
NICTA



UNSW

Proving Functional Correctness

NICTA



UNSW

Proving Functional Correctness

NICTA

```
constdefs
  schedule :: "unit s_monad"
  "schedule ≡ do
     threads ← allActiveTCBs;
     thread ← select threads;
     do_machine_op flushCaches OR return ();
     modify (λs. s ⦇ cur_thread := thread ⦈)
   od"
```

```
schedule :: Kernel ()
schedule = do
        action <- getSchedulerAction
```

```
void
setPriority(tcb_t *tptr, prio_t prio) {
    prio_t oldprio;

    if(thread_state_get_tcbQueued(tptr->tcbState)) {
        oldprio = tptr->tcbPriority;
        ksReadyQueues[oldprio] = tcbSchedDequeue(tptr, ksReadyQueues[
        if(isRunnable(tptr)) {
            ksReadyQueues[prio] = tcbSchedEnqueue(tptr, ksReadyQueues
        }
        else {
            thread_state_ptr_set_tcbQueued(&tptr->tcbState, false);
        }
    }

    tptr->tcbPriority = prio;
}

void
yieldTo(tcb_t *target) {
    target->tcbTimeSlice += ksCurThread->tcbTimeSlice;
```

```
ad
curThread
meSlice curThread
ime == 0) chooseThread
```


UNSW

MIT Technology Review

LISTS | INNOVATORS UNDER 35 | DISRUPTIVE COMPANIES | BREAKTHROUGH TECHNOLOGIES

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2011

Share

Crash-Proof Code

Making critical software safer

7 comments
WILLIAM BULKELEY
May/June 2011

seL4 Formal Verification Summary

NICTA

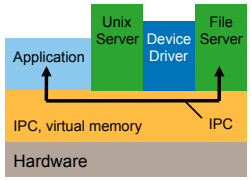
Kinds of properties proved

- Behaviour of C code is fully captured by abstract model
- Behaviour of C code is fully captured by executable model
- Kernel never fails, behaviour is always well-defined
  - assertions never fail
  - will never de-reference null pointer
  - cannot be subverted by misformed input
- All syscalls terminate, reclaiming memory is safe, ...
- Well typed references, aligned objects, kernel always mapped…
- Access control is decidable

Can prove further poperties on abstract level!

How About Performance?

NICTA



Let's face it, seL4 is basically slow!

- C code (semi-blindly) translated from Haskell
- Many small functions, little regard for performance

IPC: one-way, zero-length	
Standard C code:	1455 cycles
C fast path:	188 cycles

Bare "pass" in Advanced Operating Systems course!

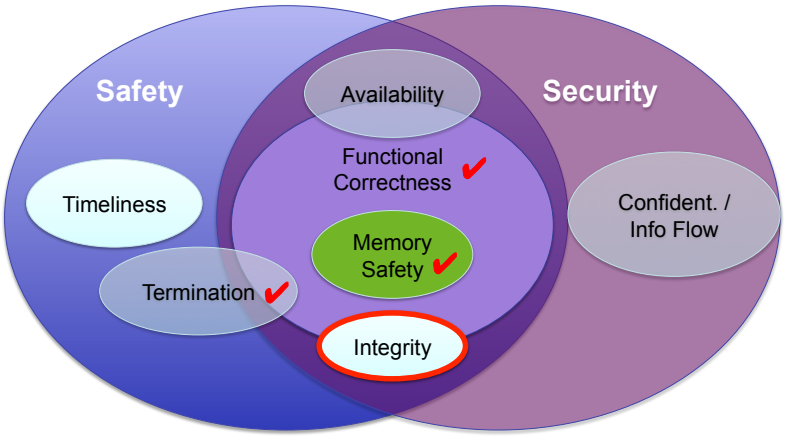
Fastest-ever IPC on ARM11!

But can speed up critical operations by short-circuit "fast paths"

- … without resorting to assembler!

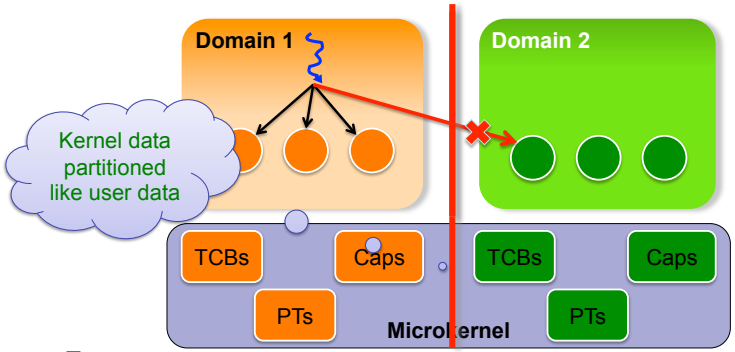
seL4 as Basis for Trustworthy Systems

NICTA



Integrity: Limiting Write Access

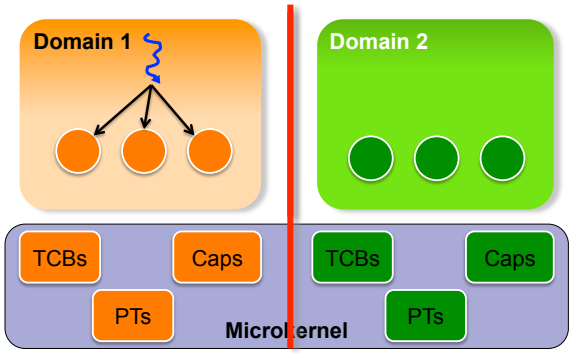
NICTA



To prove:

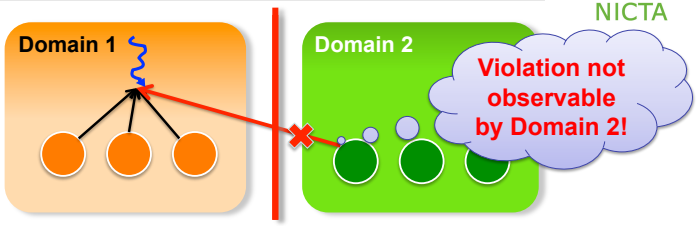
- Domain-1 doesn't have write *capabilities* to Domain-2 objects
  ⇒ no action of Domain-1 agents will modify Domain-2 state
- Specifically, *kernel does not modify on Domain-1's behalf!*
  - Event-based kernel operates on behalf of well-defined user thread
  - Prove kernel only allows write upon capability presentation

Availability: Ensuring Resource Access



- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources
- Nothing to do: implied by other properties

Confidentiality: Limiting Read Accesses



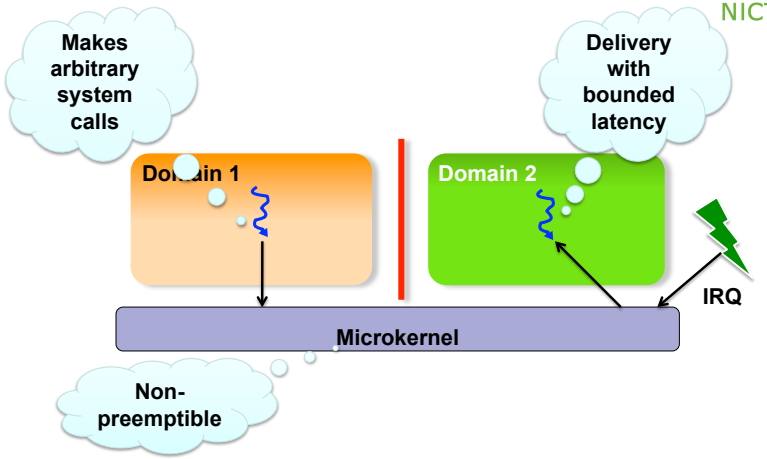
To prove:

- Domain-1 doesn't have read capabilities to Domain-2 objects
  ⇒ no action of any agents will reveal Domain-2 state to Domain-1

Non-interference proof :

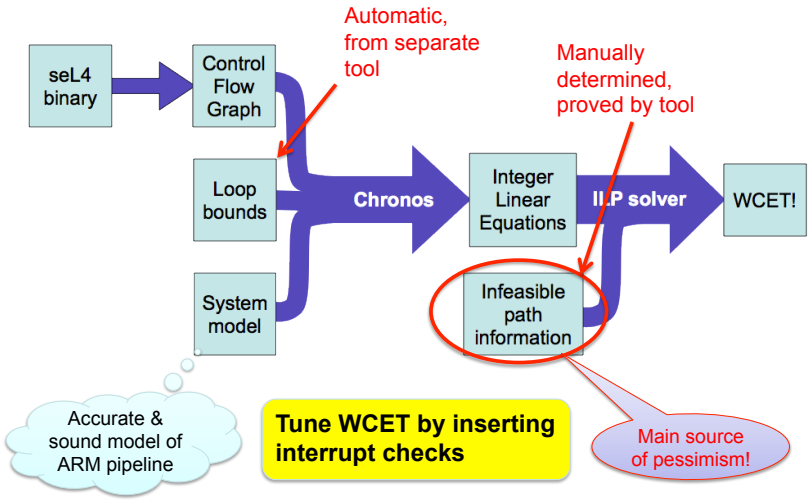
- Evolution of Domain 1 does not depend on Domain-2 state
- Also shows absence of covert *storage* channels

Timeliness

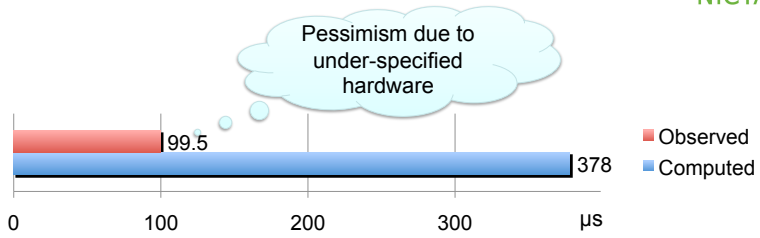


Need worst-case execution time (WCET) analysis of kernel

WCET Analysis Approach

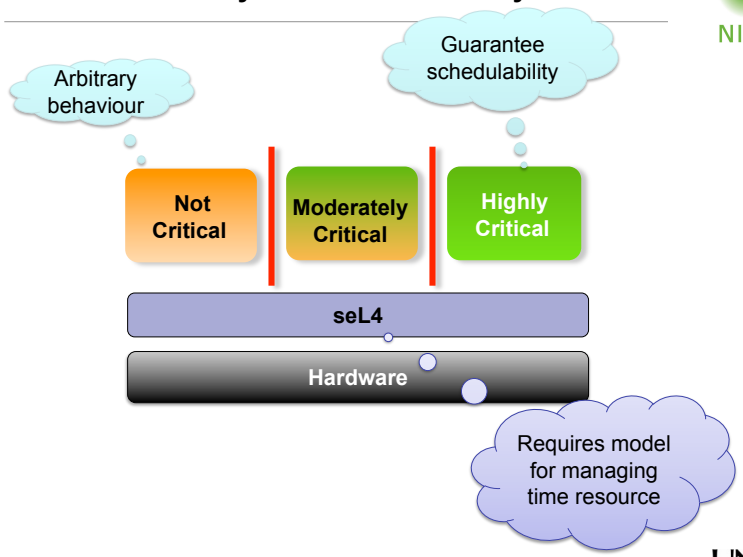


Result

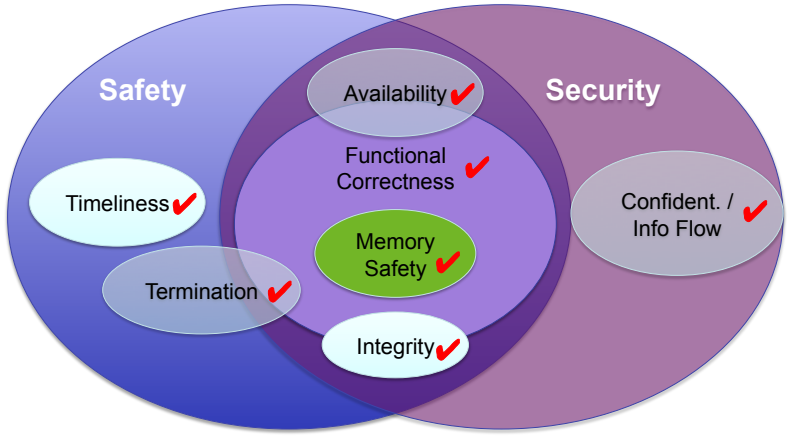


WCET presently limited by verification practicalities
- **10 µs seem achievable**

Future: Whole-System Schedulability



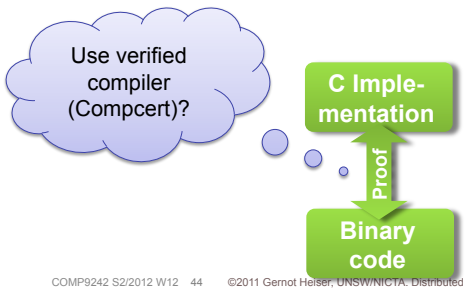
seL4 as Basis for Trustworthy Systems



Binary Verification

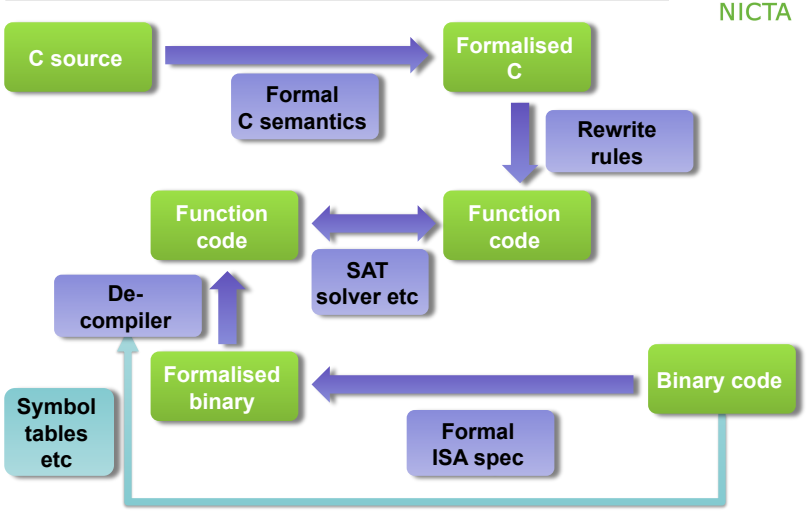
IPC: one-way, zero-length		
Compiler	**gcc**	**Compcert**
Standard C code:	1455 cycles	3749 cycles
C fast path:	188 cycles	730 cycles

Uncompetitive performance!

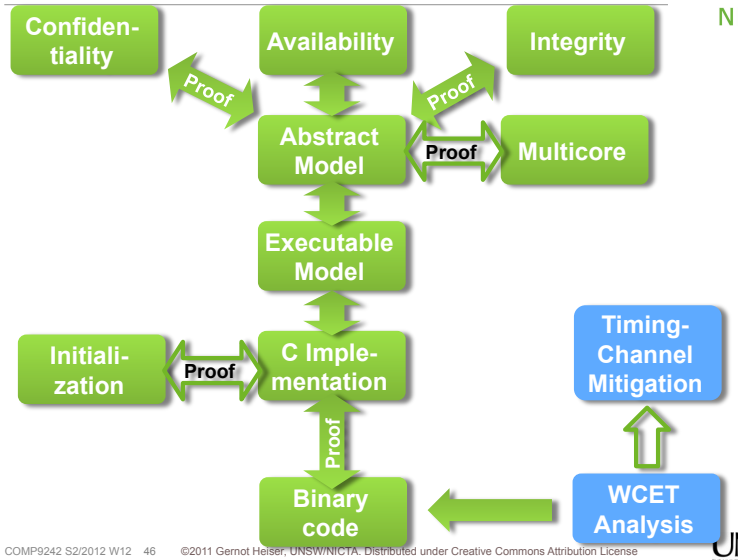


Bigger problem:
- Our proofs are in Isabel/HOL, Compcert uses Coq
- We cannot prove that they use the same C semantics!

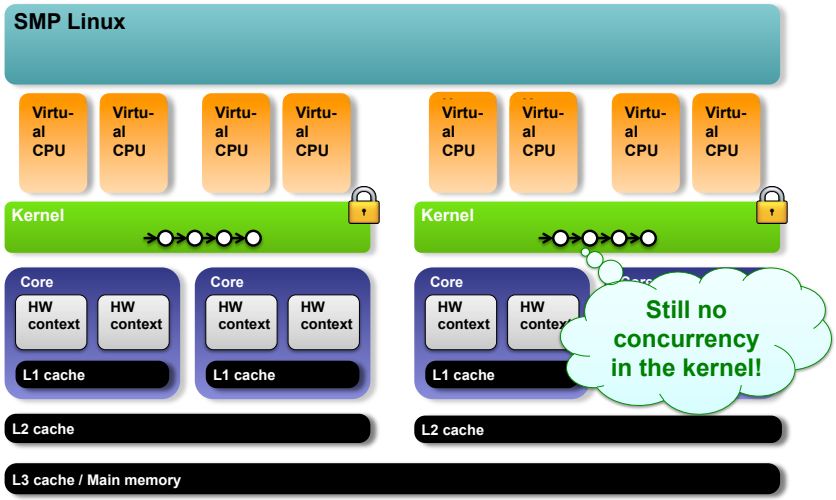
Binary Code Verification



seL4 – the Next 24 Months



seL4 Multicore Design: Clustered Multikernel



Multikernel Verification

- By definition, multikernel images execute independently
  - except for explicit messaging

- To prove:
  - isolated images are initialised correctly
  - images maintain isolation at run time

Essentially non-interference

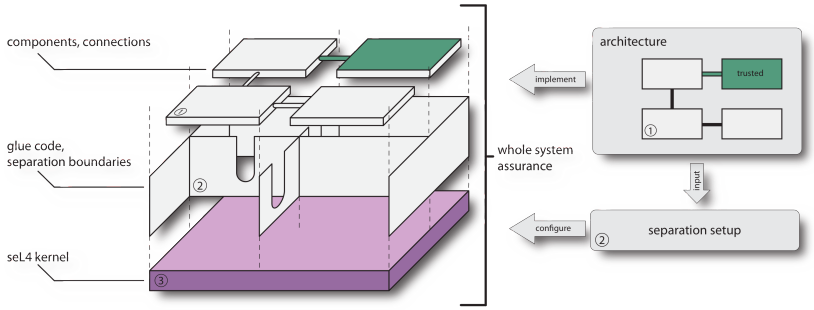
Phase Two: Full-System Guarantees

NICTA

- Achieved: Verification of microkernel (8,700 LOC)
- Next step: Guarantees for real-world systems (1,000,000 LOC)

Overview of Approach

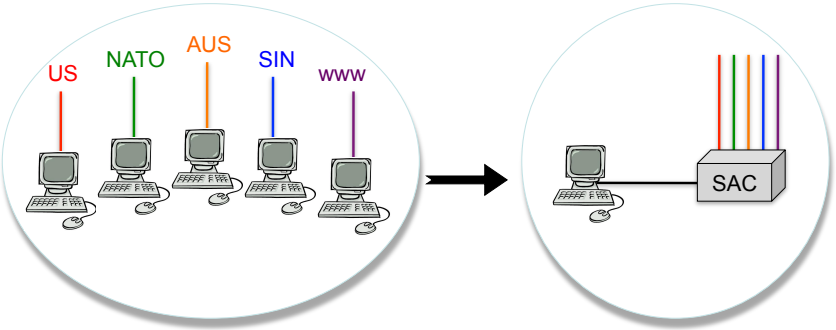
NICTA



- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance

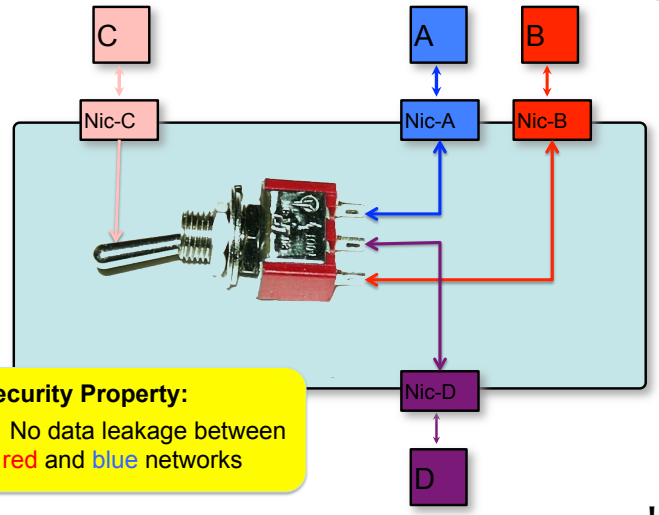
Proof of Concept: Secure Access Controller

NICTA



Logical Function

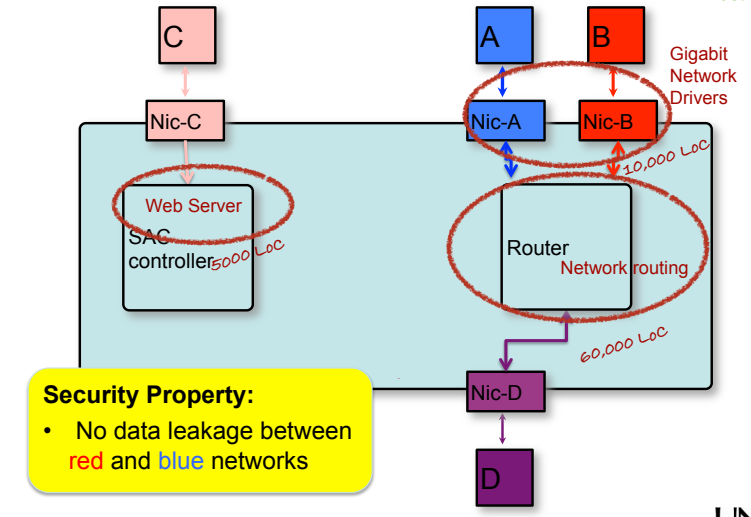
NICTA



Security Property:

- No data leakage between red and blue networks

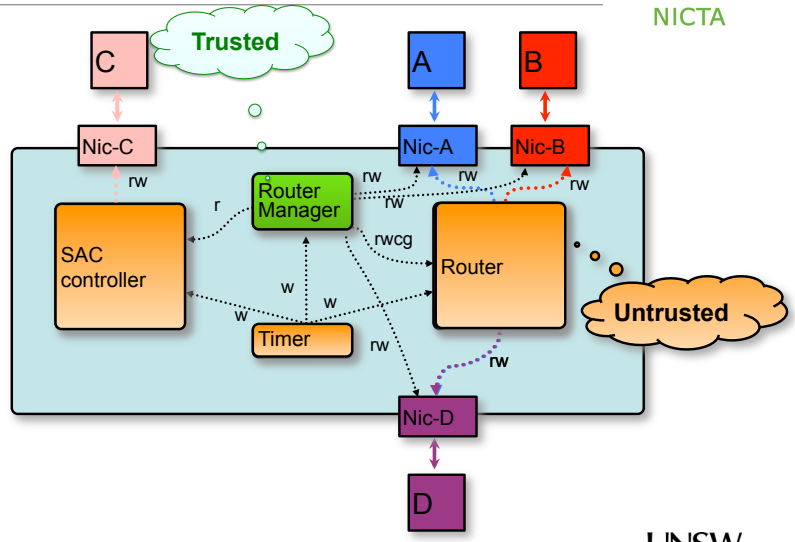
Logical Function



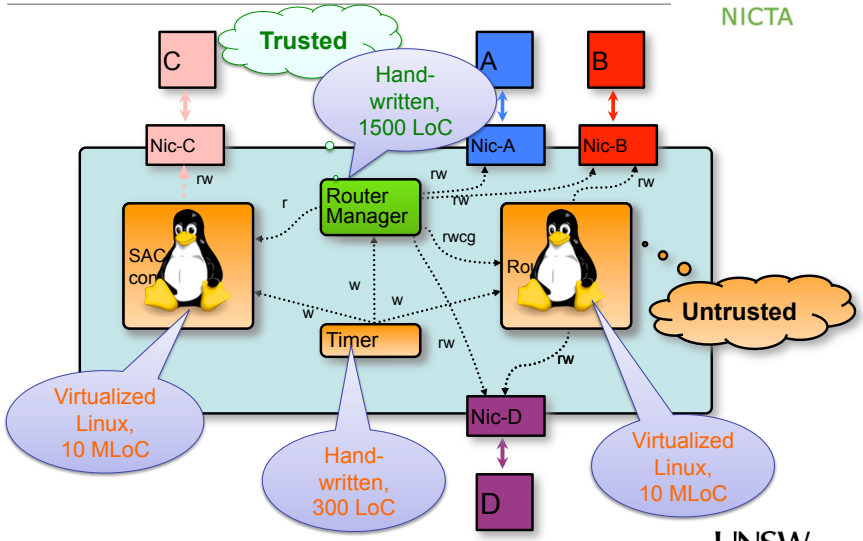
Security Property:

- No data leakage between red and blue networks

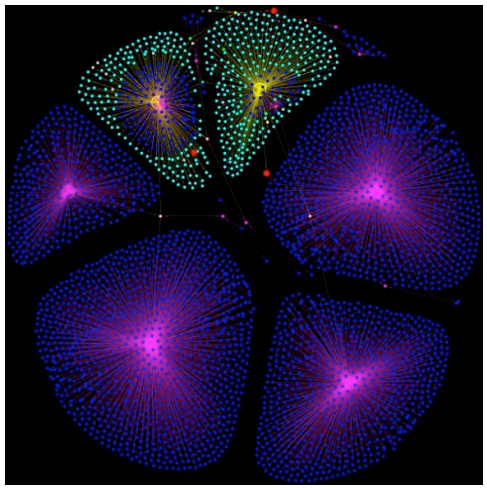
Minimal TCB



Implementation



Access Rights



Next Step: Full System Assurance

DARPA HACMS Program:

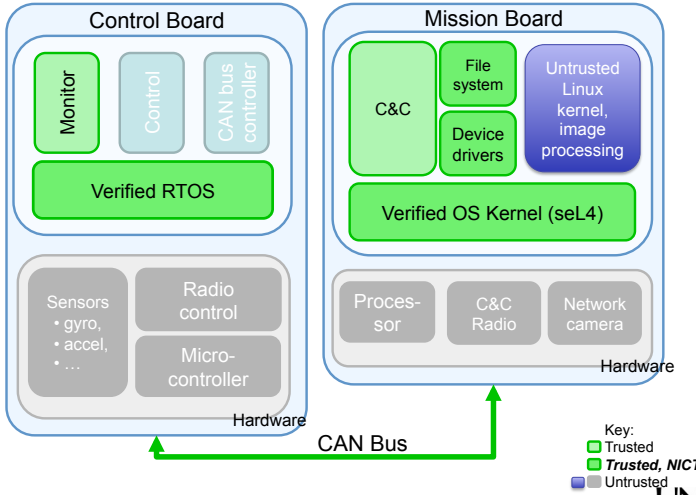
- Provable vehicle safety
- "Red Team" must not be able to divert vehicle

SMACCMcopter Research Vehicle

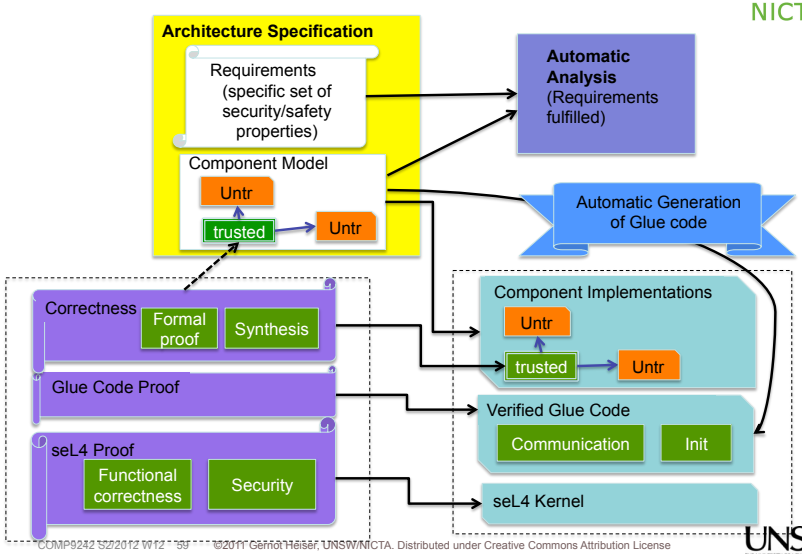
Boeing Unmanned Little Bird (AH-6) Deployment Vehicle

NICTA Rockwell Collins BOEING galois University of Minnesota

SMACCMcopter System Structure



Architecting Security/Safety



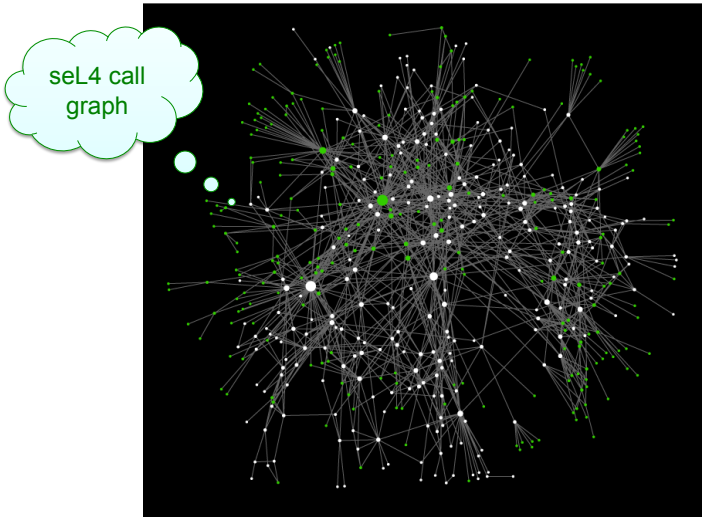
seL4 Verification Cost Breakdown

Haskell design	2 py
C implementation	2 months
Debugging/Testing	2 months
Kernel verification	12 py
Fastpath verification	5 months
Formal frameworks	9 py
Total	**24 py**
Repeat (estimated)	6 py
Traditional engineering	4–6 py

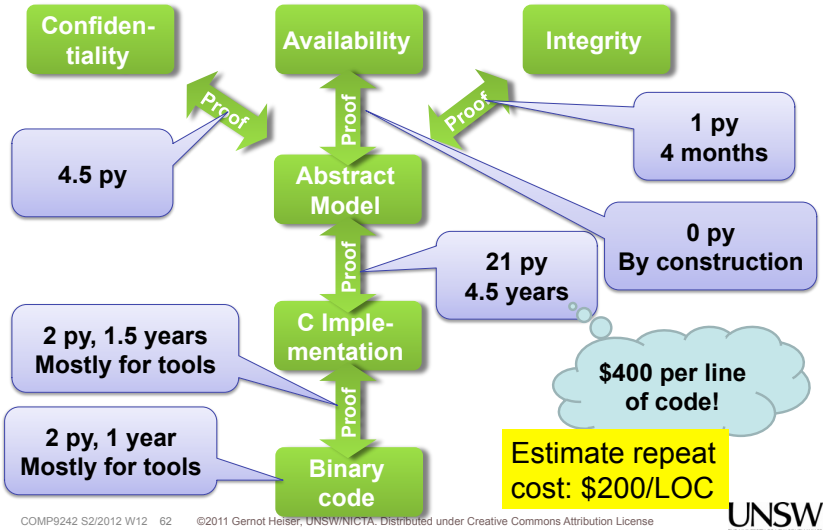
Did you find bugs???

- During (very shallow) testing: 16
- During verification: 460
  - 160 in C, ~150 in design, ~150 in spec

Why So Hard for 9,000 LOC?



seL4: Cost of Assurance



Cost of Assurance

Industry Best Practice:

- "High assurance": $1,000/LOC, no guarantees, *unoptimised*
- Low assurance: $100–200/LOC, 1–5 faults/kLOC, *optimised*

State of the Art – seL4:

  - $400/LOC, 0 faults/kLOC, *optimised*
- Estimate repeat would cost half
  - that's about the development cost of the predecessor Pistachio!
- Aggressive optimisation [APSys'12]
  - much faster than traditional high-assurance kernels
  - as fast as best-performing low-assurance kernels

What Have We Learnt?

Formal verification *probably* didn't produce a more *secure* kernel

- In reality, traditional separation kernels are *probably* secure

But:

- We now have certainty
- We did it *probably* at less cost

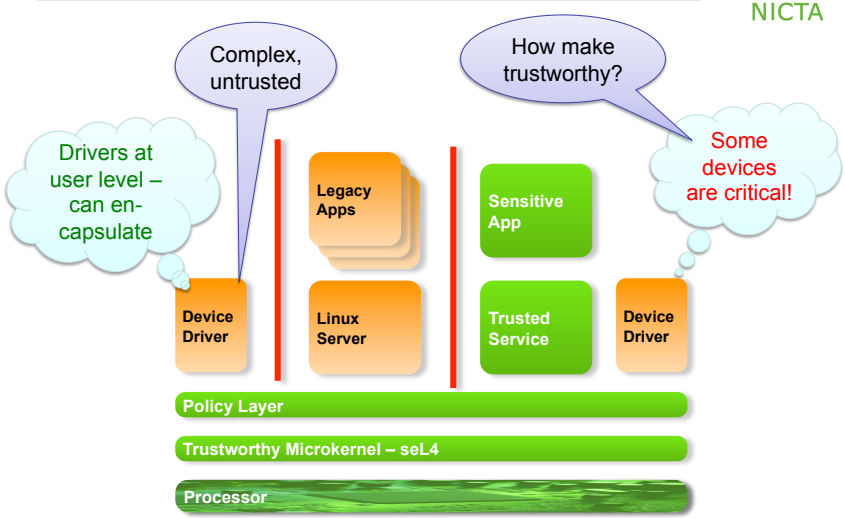
Real achievement:

- Cost-competitive at a scale where traditional approaches still work
- Foundation for scaling beyond: **2 × cheaper, 10 × bigger!**

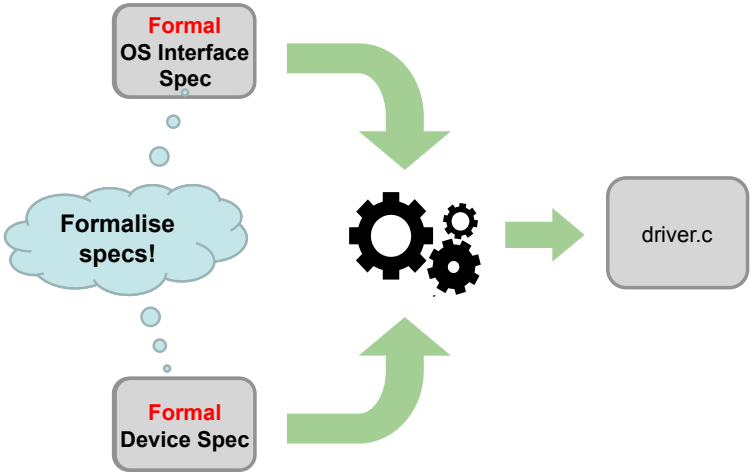
How?

- Combine theorem proving with
  - synthesis
  - domain–specific languages (DSLs)

Device Drivers



Synthesis: Device Drivers [SOSP'09]



Actually works! (On Linux & seL4)

IDE disk controller

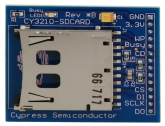
W5100 Eth shield

Intel PRO/1000 Ethernet

Working on proving correctness

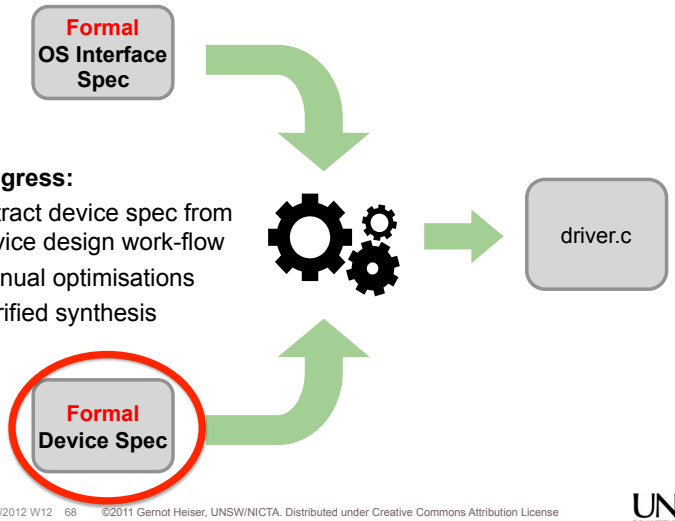
UART controller

Asix AX88772 USB-to-Eth adapter



SD host controller

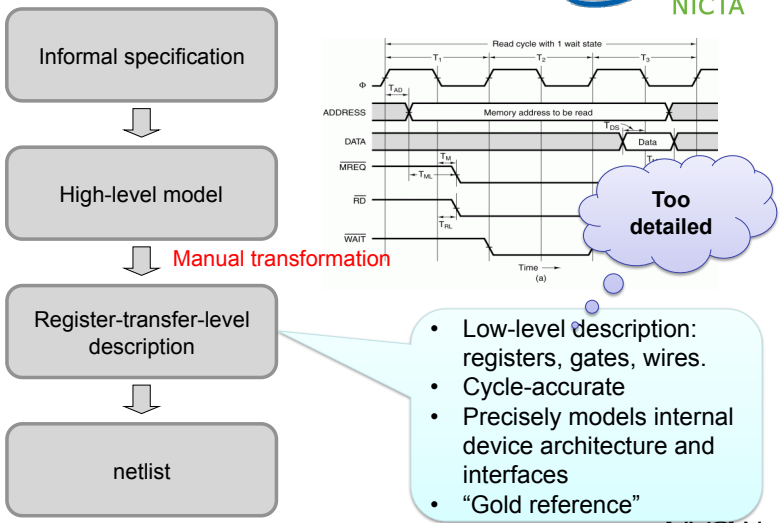
Synthesis: Device Drivers



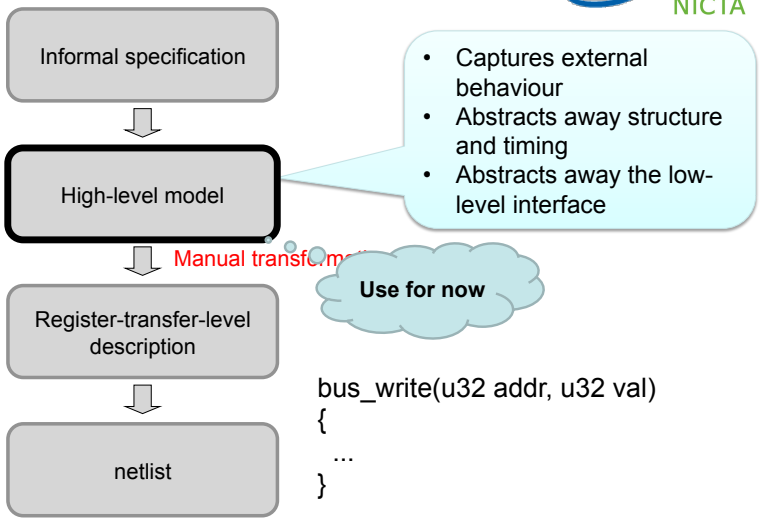
In progress:

- Extract device spec from device design work-flow
- Manual optimisations
- Verified synthesis

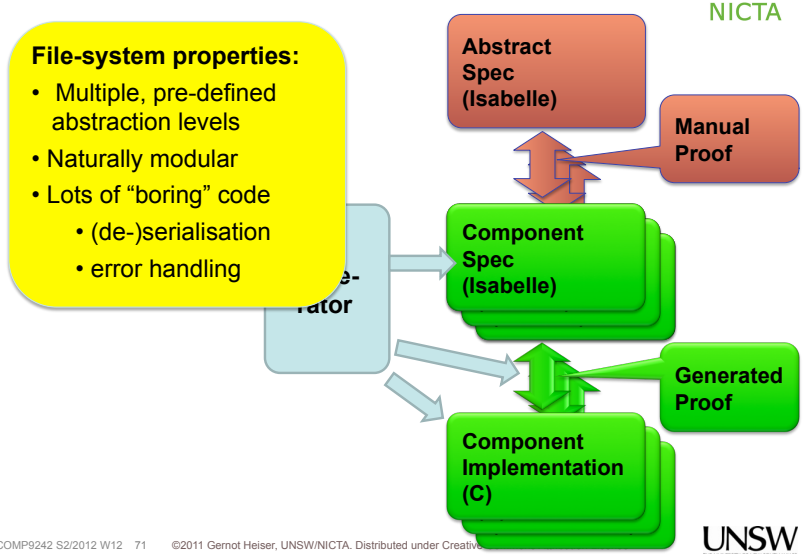
Hardware Design Workflow



Hardware Design Workflow

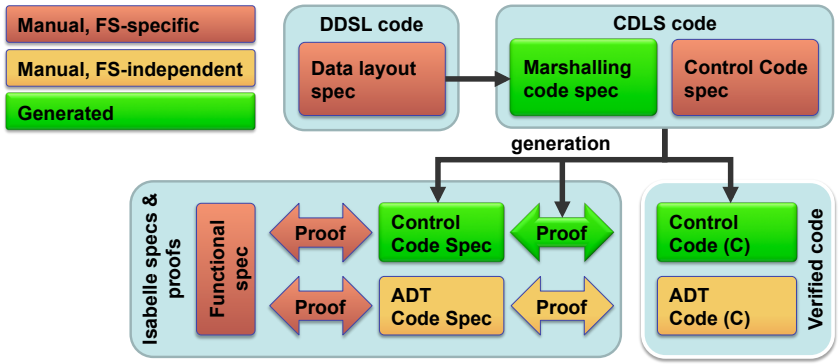


Generation: File System

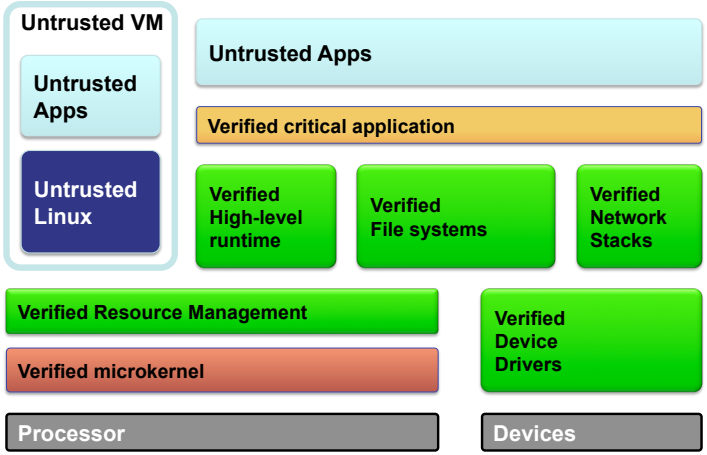


Code and Proof Co-Generation from DSLs

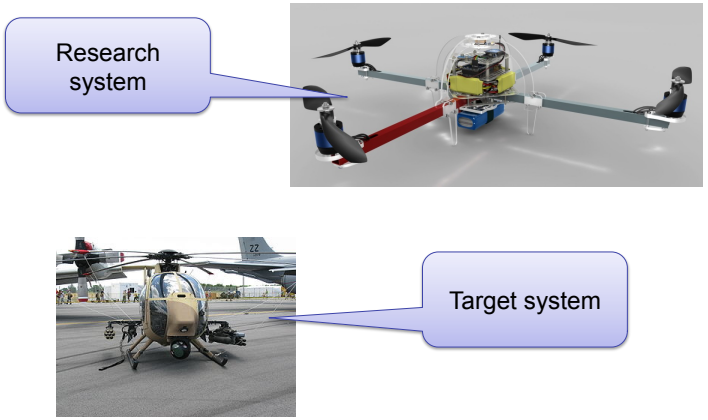
Pilot project: Flash file system
- Linux-compatible
- Fits between VFS and flash abstraction (UBI)



Future: Full-Scale Trustworthy System



Full-System Assurance in 3 Years!



Trustworthy Systems–Be Part of It!

Slashdot

Stories Recent Popular Searc

Slashdot is powered by your subm

Technology: World's Firs

Posted by Soulskill on Thursday Aug
from the wait-for-it dept.

An anonymous reader writes

"Operating systems usually have
and so forth are known by almos
to prove that a particular OS ker
formally verified, and as such it
researchers used an executable
the Isabelle theorem prover to ge
matches the executable and the

Does it run Linux? "We're pleased to

New Scientist
Saturday 29/8/2009
Page: 21
Section: General News
Region: National
Type: Magazines Science / Technology
Size: 196.31 sq.cms.
Published: -----S-

The ultimate way to keep your computer safe from harm

FLAWS in the code, or "kernel", that sits at the heart of modern computers leave them prone to occasional malfunction and vulnerable to attack by worms and viruses. So the development of a secure general-purpose microkernel could pave the

just mathematics, and you can reason about them mathematically," says Klein.

His team formulated a model with more than 200,000 logical steps which allowed them to prove that the program would always behave as its

kernelét alkotó 7 500 sornyi C forráskód helyességét igazolni egyedülálló teljesítm
eredményeképpen pedig egy olyan megbízhatóságot kapnak a szoftvertől, amely e

