Open Kernel Labs™

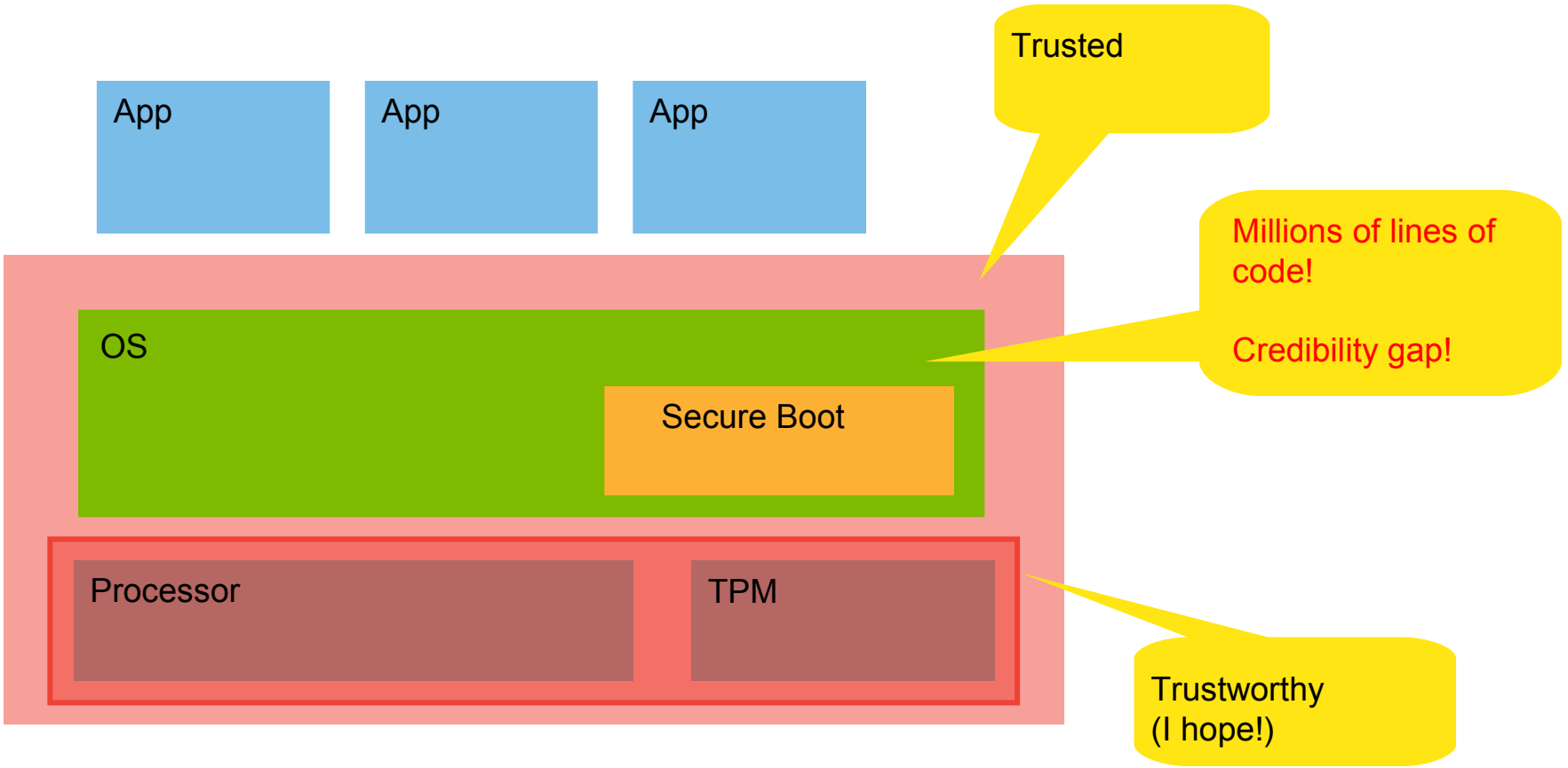
Be open. Be safe.

Formal OS Kernel Verification — Making Trusted Trustworthy

Gernot Heiser
NICTA and UNSW and Open Kernel Labs
Sydney, Australia

December, 2008

“Trusted Computing” a la TCG



Rehash of Yesterday

Open Kernel Labs™

Be open.
Be safe.

Operating systems are trusted, but not trustworthy

- Millions of lines of code (LOC)
- Thousands of bugs
- Hundreds of security holes
- Standard way out: minimise the *trusted computing base* (TCB)
  - Microkernels are good
  - Fewer LoC ⇒ fewer security-relevant bugs
- Not exactly a radical idea
  - QNX selling a microkernel since early '90s
  - Green Hills Integrity since 2000 or so
  - OKL4 from Open Kernel Labs deployed in 250 million devices

Also Mentioned: Communication Control & MAC

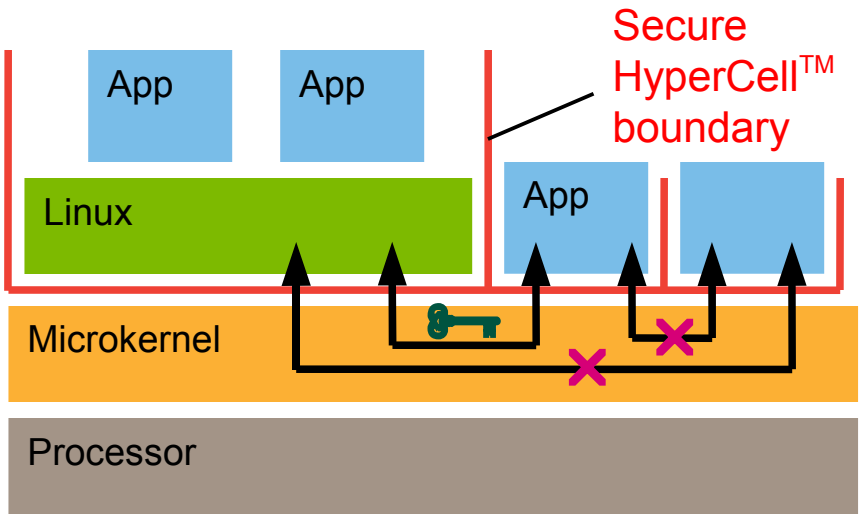
NICTA

Open Kernel Labs™

Be open. Be safe.

OKL4 has it:

- Communication controlled by capabilities
  - Use of a communication channel requires a capability to it
- Define isolation domains called *Secure HyperCells*
- Impose mandatory communication control based on system-wide policy



How About Formal Verification?

- Never done before — why?
- E.g. Common Criteria:

Evaluation Level	Requirements	Functional Specification	Top-Down Design	Implementation
EAL1	not evaluated	Informal	not evaluated	not evaluated
EAL2	not evaluated	Informal	Informal	not evaluated
EAL3	not evaluated	Informal	Informal	not evaluated
EAL4	not evaluated	Informal	Informal	Informal
EAL5	not evaluated	Semi-Formal	Semi-Formal	Informal
EAL6	**Formal**	Semi-Formal	Semi-Formal	Informal
EAL7	**Formal**	**Formal**	**Formal**	Informal

- One system is close: NICTA's seL4 microkenel

Trusted Computing Limitations

Open Kernel Labs™

Be open.
Be safe.

Trusted computing without a trustworthy TCB is a fantasy!

- Assurance does *not* ensure that the TCB is trustworthy
  - Even at the highest, most thorough evaluation standards!
- Real trustworthiness can only be based on *proof*.
- Implication: Need an OS format that
  - Can be proved to satisfy security requirements
  - Can be proved to be correctly implemented
  - Can support arbitrary systems, including standard consumer electronics
  - Performs well enough to be usable on battery-powered mobile devices
    - Overheads must not significantly exceed traditional OSes
- Constructed proof of feasibility: **NICTA seL4 microkernel**

seL4

NICTA

Open Kernel Labs™

Be open.
Be safe.

The seL4 Microkernel

NICTA

Open Kernel Labs™

Be open.
Be safe.

Goals

- Formal specification of kernel and machine
- High-performance implementation
- Formal proof of security properties
- Formal verification of implementation

Innovation over other L4 kernels:

- All accesses mediated by capabilities
- Kernel resource accounting
  - complete internal separation of memory held on behalf of applications (page tables, control blocks)
  - memory explicitly provided to kernel
  - free from covert storage channels *by construction*
- No significant performance penalty for new features
  - 15 cycles per syscall ok. Maybe.

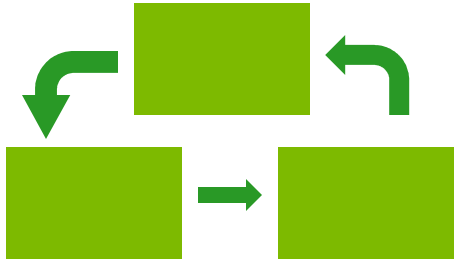
seL4 Verification Project Overview

NICTA

Open Kernel Labs™

Be open.
Be safe.

- Size of the project
  - Average 4–5 people (full time equivalent)
  - 5 years
  - Ends March 2009
- Interesting Problems
  - Designing and formalising an OS kernel
  - Refinement on monadic functional programs
  - Refinement on C programs
  - Formalizing machine details
  - Access control



seL4 Capability-based Protection

NICTA

Open Kernel Labs™

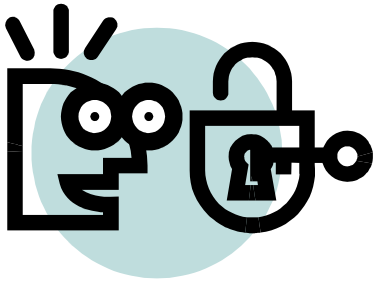
Be open.
Be safe.

All authority conferred via capabilities

- Capabilities are like keys
  - Possess the key, and you can invoke the operation
- All system calls are invoked via Capabilities
  - No ambient authority

Advantage: Established body knowledge on capabilities

- **Reason** about them
- Confining authority



Capability-based protection adopted by OK Labs

- first stage of introduction (for IPC) in OKL4 V3.0
- other resources to be covered later

seL4 Physical Memory Management

Some kernel memory is statically allocated at boot time

Remainder is divided into untyped (UT) objects

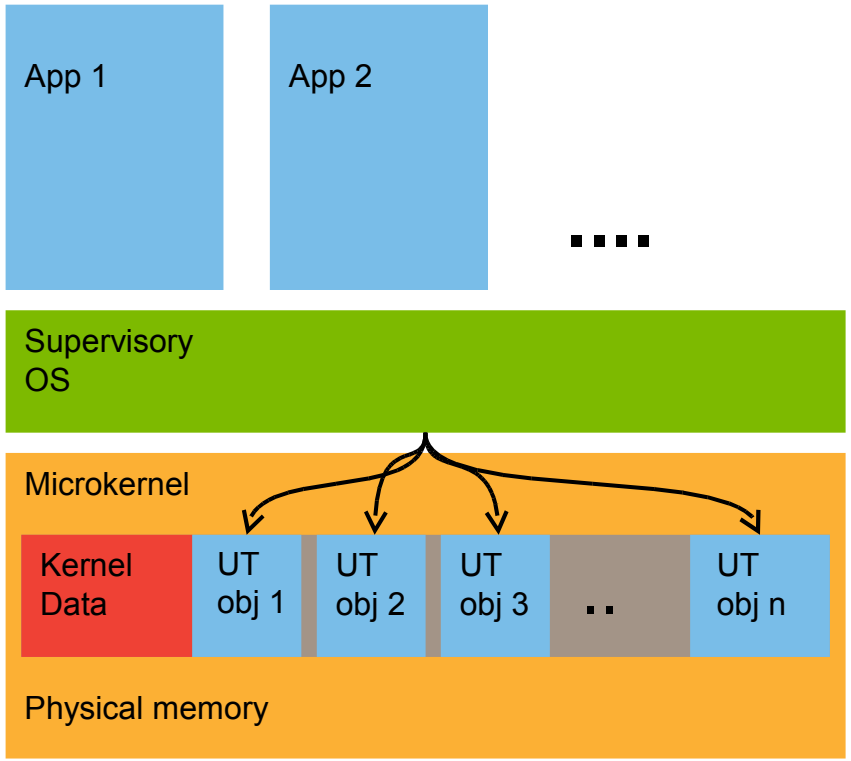
- 2^n region of physical memory
- size aligned

Supervisor gets authority over these objects

- authority conferred by capabilities

Kernel never allocates dynamic memory

- user must provide memory for kernel objects
- re-typing untyped memory to kernel object type

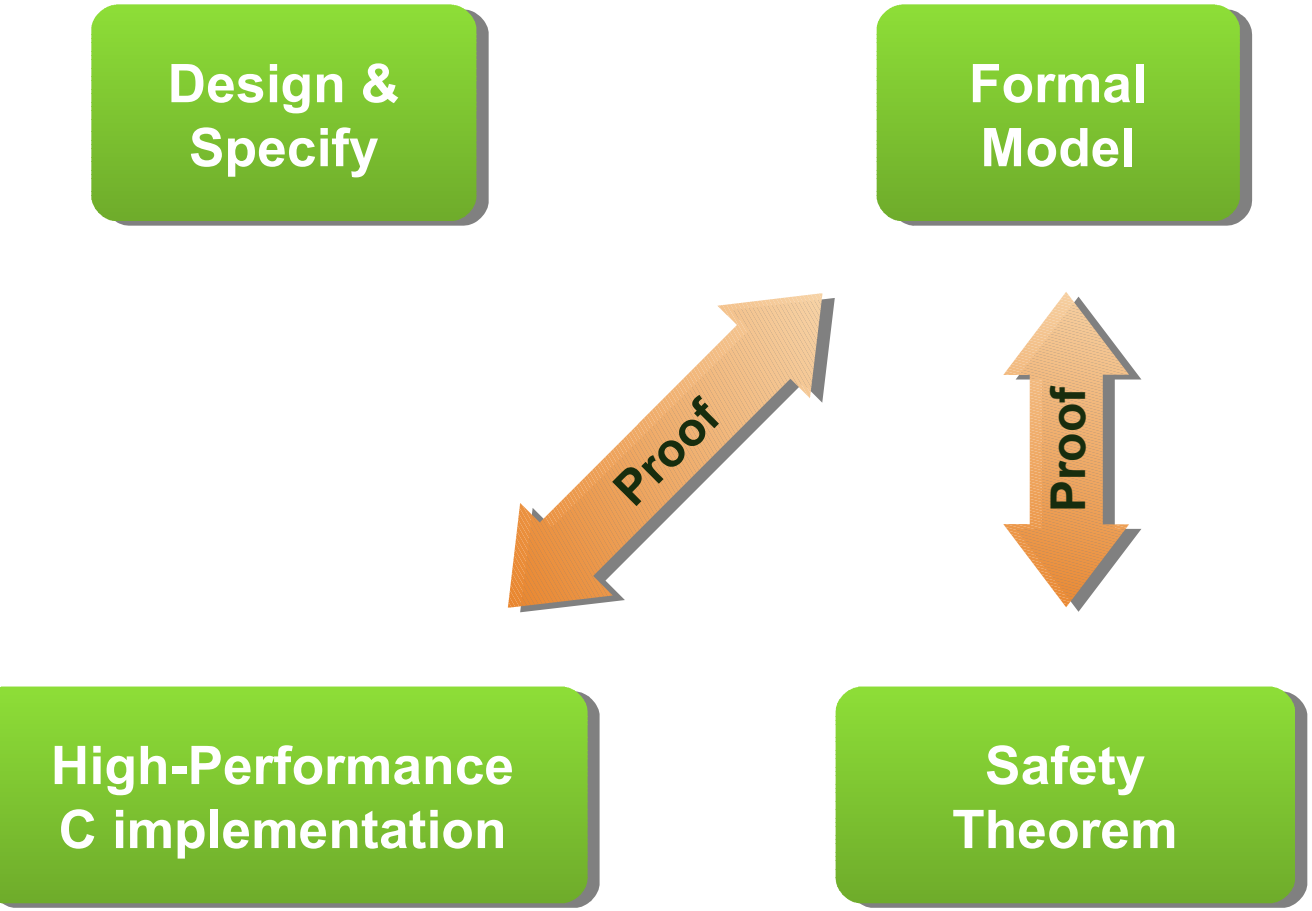


Designing and Formalising

NICTA

Open Kernel Labs™

Be open.
Be safe.



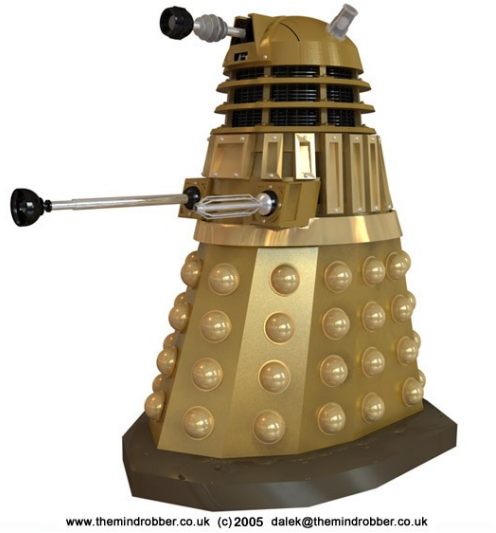
Two Teams

NICTA

Open Kernel Labs™

Be open.
Be safe.

Formal Methods Practitioners
vs
Kernel Developers



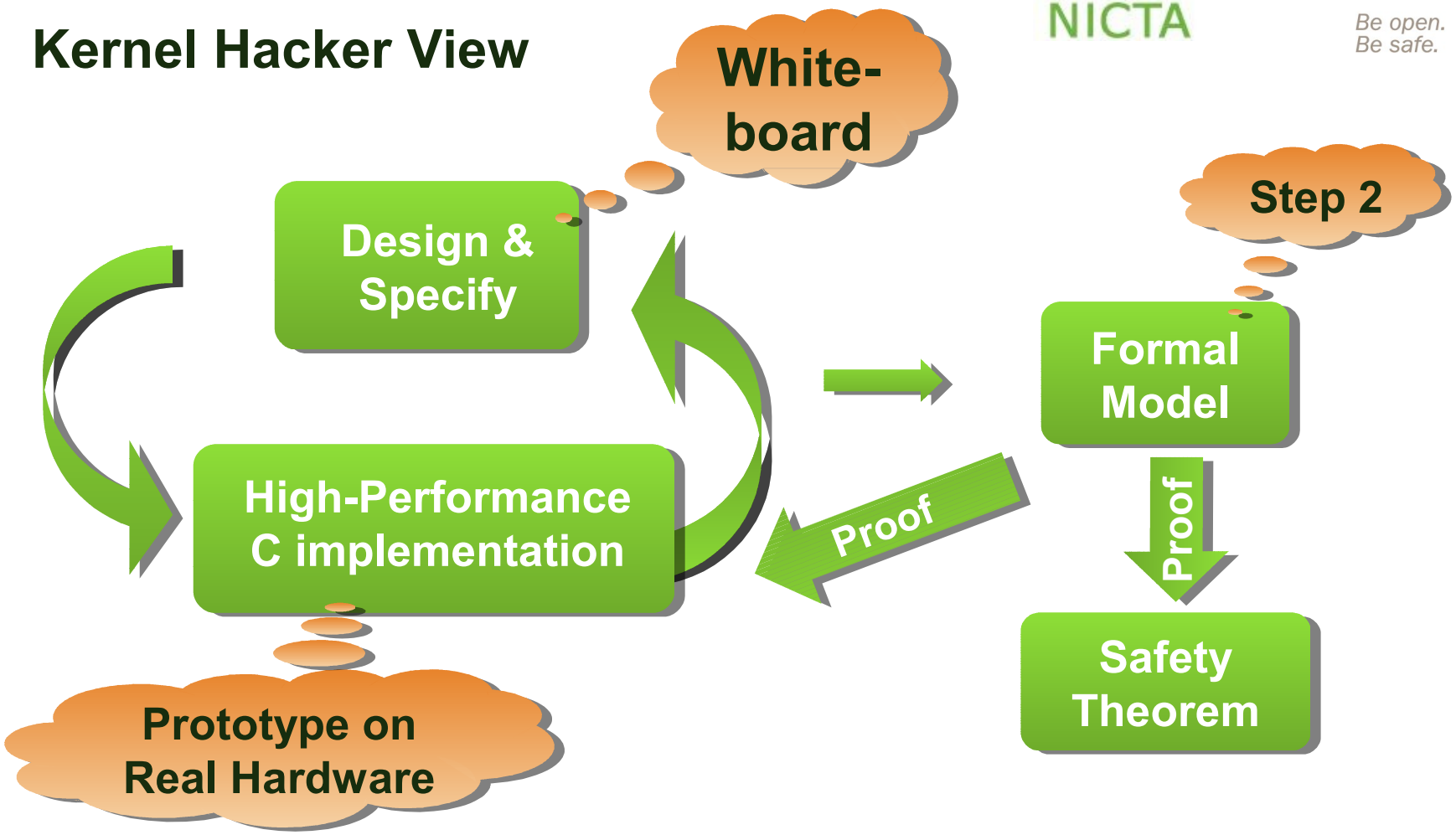
Standard Kernel Design

NICTA

Open Kernel Labs™

Be open.
Be safe.

Kernel Hacker View



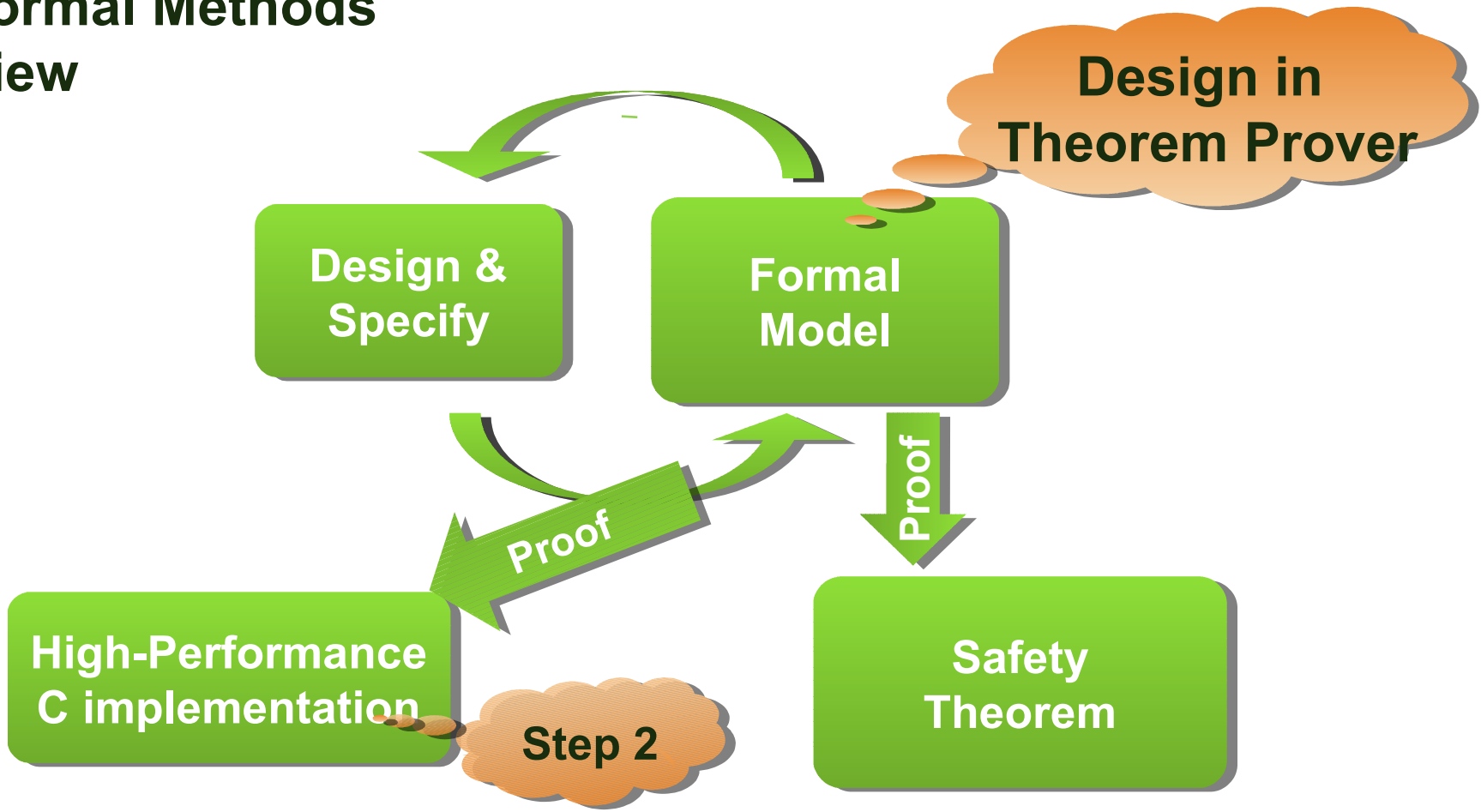
Formal Design

NICTA

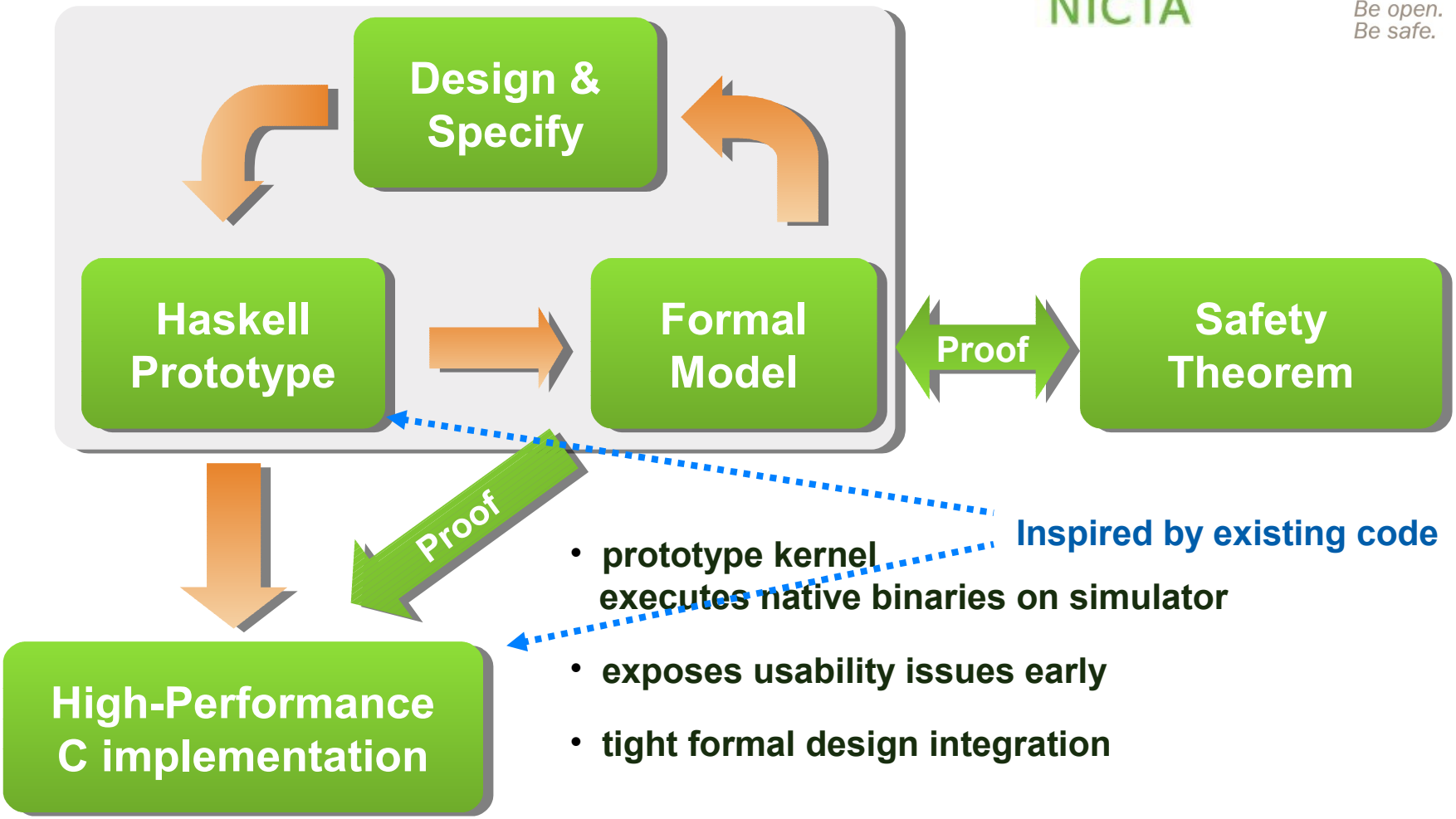
Open Kernel Labs™

Be open.
Be safe.

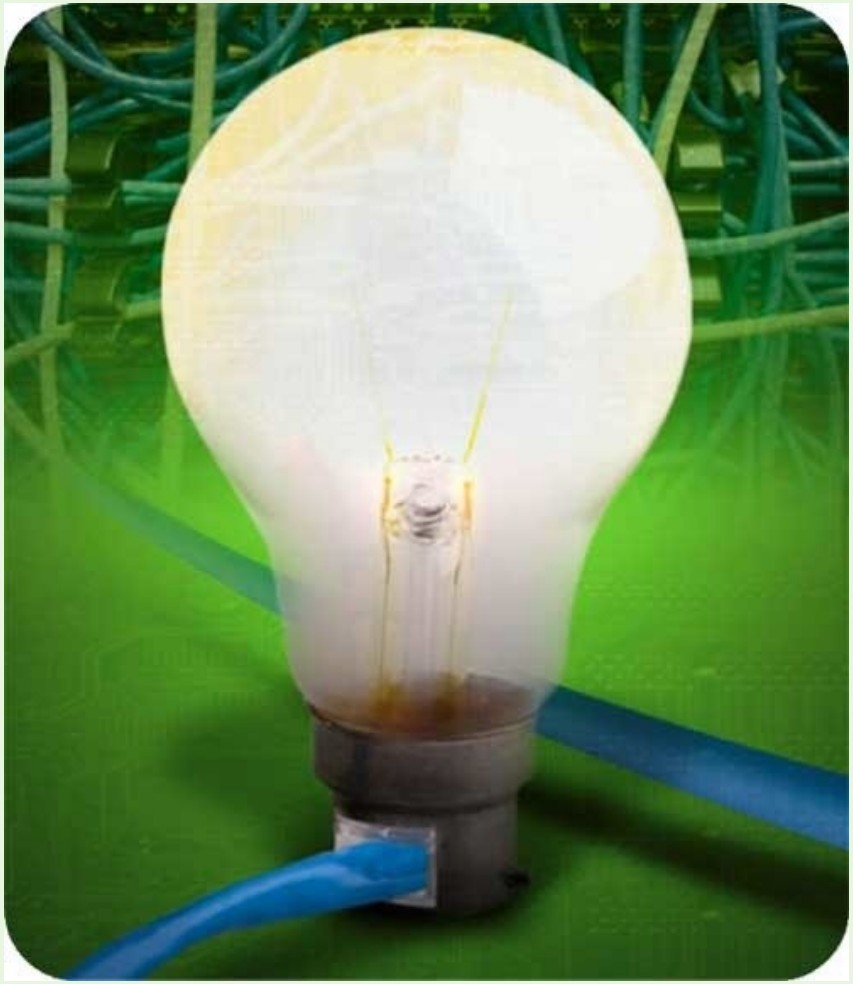
Formal Methods View



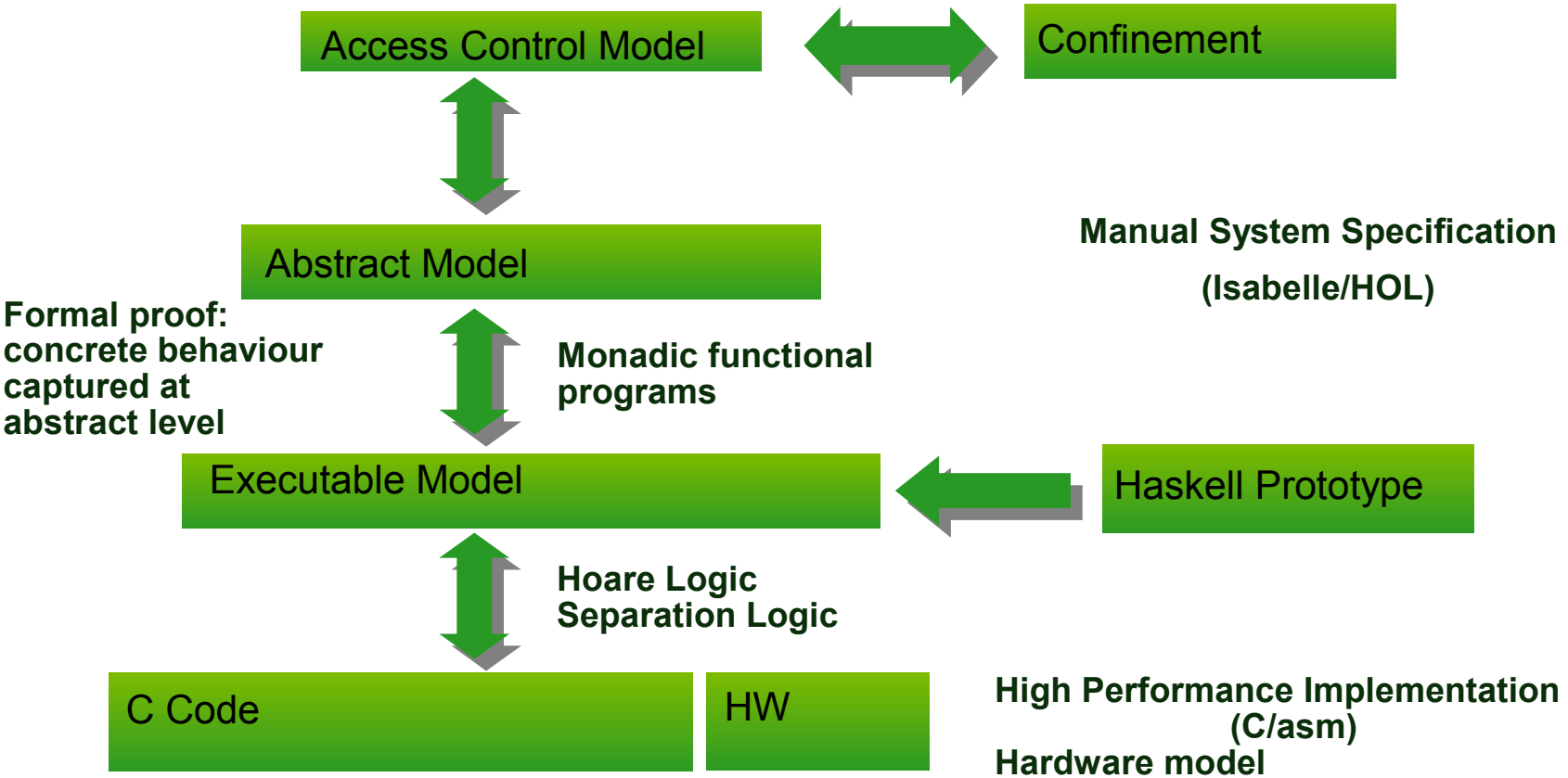
Iterative Design and Formalisation



The Proof



The Proof



```
datatype
  rights = Read
         | Write
         | Grant
         | Create

record cap =
  entity :: entity_id
```

```
lemma isolation:
  "⟦sane s;
    s' ∈ execute cmds s;
    isEntityOf s e_S;
    sEntityOf s e;
    ntity c = e;
    :> subSysCaps s e_S⟧
    Caps s' e_S"
```

```
constdefs
  schedule :: "unit s_monad"
  "schedule ≡ do
```

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

```
tcb_t * scheduler_t::find_next_thread(prio_queue_t * prio_queue)
{
    ASSERT(DEBUG, prio_queue);

    if (prio_queue->index_bitmap) {
        word_t top_word = msb(prio_queue->index_bitmap);
        word_t offset = BITS_WORD * top_word;

        for (long i = top_word; i >= 0; i--)
        {
            word_t bitmap = prio_queue->prio_bitmap[i];

            if (bitmap == 0)
                goto update;

            do {
                word_t bit = msb(bitmap);
                word_t prio = bit + offset;
                tcb_t *tcb = prio_queue->get(prio);
```

ecification

DL)

read

be

ce Implementation
C/asm)

Common Criteria and seL4

Open Kernel Labs™

Be open.
Be safe.

Evaluation Level	Require-ments	Functional Specification	Top-Down Design	Imple-mentation
EAL1	not eval.	Informal	not eval	not eval
EAL2	not eval.	Informal	Informal	not eval
EAL3	not eval.	Informal	Informal	not eval
EAL4	not eval.	Informal	Informal	Informal
EAL5	not eval.	Semi-Formal	Semi-Formal	Informal
EAL6	**Formal**	Semi-Formal	Semi-Formal	Informal
EAL7	**Formal**	**Formal**	**Formal**	Informal
seL4	**Formal**	**Formal**	**Formal**	**Formal**

Common Criteria and L4.verified

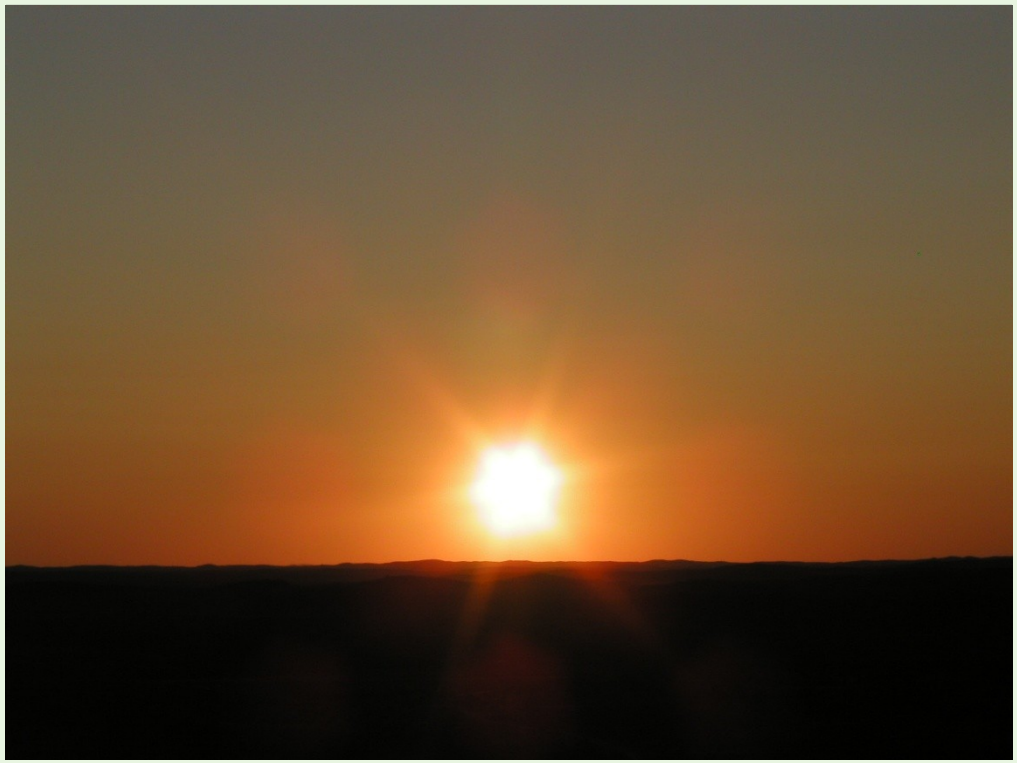
Evaluation Level	Require-ments	Functional Specification	Top-Down Design	Imple-mentation	Cost
EAL1	not eval.	Informal	not eval	not eval	
EAL2	not eval.	Informal	Informal	not eval	
EAL3	not eval.	Informal	Informal	not eval	
EAL4	not eval.	Informal	Informal	Informal	
EAL5	not eval.	Semi-Formal	Semi-Formal	Informal	
EAL6	**Formal**	Semi-Formal	Semi-Formal	Informal	$10k/LoC
EAL7	**Formal**	**Formal**	**Formal**	Informal	
seL4	**Formal**	**Formal**	**Formal**	**Formal**	$0.6k/LoC

Wrapping Up

NICTA

Open Kernel Labs™

Be open.
Be safe.



seL4 Summary

NICTA

Open Kernel Labs™

Be open.
Be safe.

Statistics

- 3.5k LoC abstract, 7kLoC concrete spec (about 3k Haskell)
- Abstract → Haskell: 100kLoP (more features coming)
- Haskell → C/asm: 80kLoP (estimated)
- Access control model + isolation proofs done (1kLoP)
- 109 patches to Haskell kernel, 132 to abstract spec
- Performance in line with other L4 kernels
- average 6 people over 5 years

Kinds of properties proved

- Well typed references, aligned objects, ..
- Well formed thread states, endpoint and scheduler queues, ...
- All syscalls terminate, reclaiming memory is safe, ...
- Authority is distributed by caps only
- Access control is decidable

Summary

NICTA

Open Kernel Labs™

Be open.
Be safe.

seL4 verification status

- Refinement to LLD complete
- C level refinement in progress (due February)
- Working on proving more security properties
- Already most formally verified kernel ever
- Performance comparable to other L4 kernels
- Commercialization by Open Kernel Labs

Conclusion:

- Verification of OS kernels is possible
- ... but it ain't easy
  - limited to small kernels
  - but can leverage guarantees of verified kernel
  - however, doing this is an unsolved and highly non-trivial problem

How About Hardware?

Open Kernel Labs™

Be open.
Be safe.

- Hardware has the appearance of being more trustworthy
  - because it's unchangeable, people think more about it
- But: if it's broken in hardware, I can't fix it in software
  - hardware is too complex to be completely formally verified
  - putting more complexity into hardware is the wrong way to go
  - keep it simple, and let me control it by software
- What hardware should be like
  - sufficient for building secure software (doesn't need much!)
  - well-defined APIs (simplicity is a bonus)
  - correctly implemented
- Formally-verified kernel becomes more like hardware
  - it needs to be extremely well-designed
  - once verified, don't change it, as this will break your proofs!

A Final Word on Commercial Realities

Open Kernel Labs™

Be open.
Be safe.

Is it possible to commercialise a verified OS?

- Formal verification can be less expensive than CC assurance
  - ... but delivers more
- seL4 is correct to a much higher degree than can be assured by CC EAL7
  - ... but it won't even be acceptable where EAL4 is required
- Problem with common criteria:
  - too expensive
  - no rewards for doing better
- Unless this is changed, there is no business case for formal verification
  - no business case ⇒ no commercial system will be verified
  - no formal verification ⇒ no trustworthy systems
- Requires leadership by governments (NSA, BSI, ...)

Thank You

NICTA

Open Kernel Labs™

Be open.
Be safe.

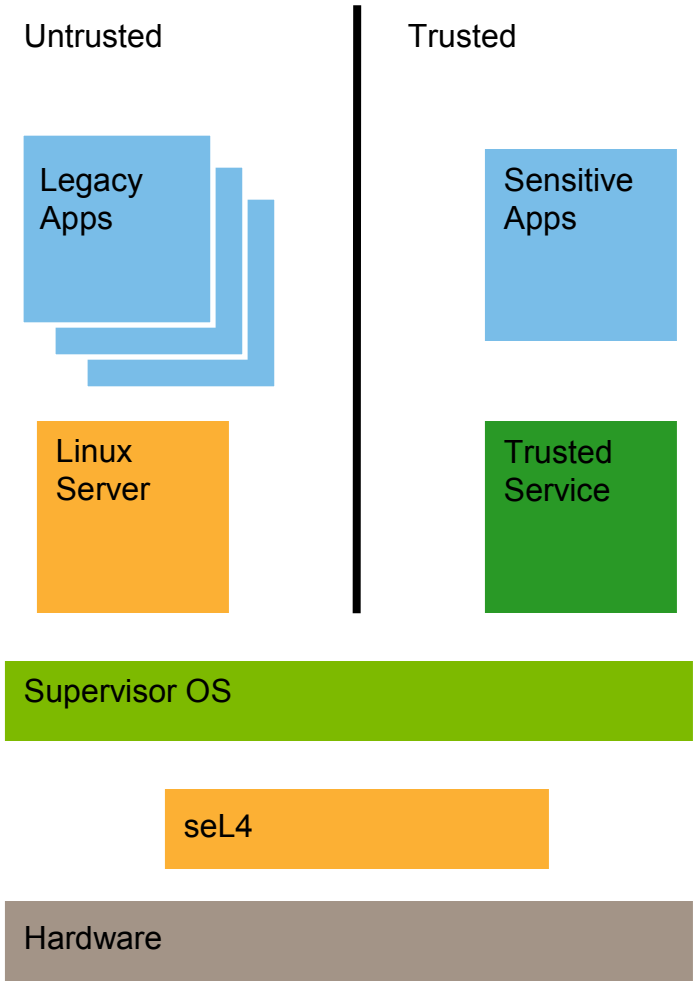
Small Kernels

NICTA

Open Kernel Labs™

Be open.
Be safe.

- Small trustworthy foundation
- Hypervisor micro kernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- Applications:
  - Fault isolation
  - Fault identification
  - IP Protection
  - Modularity
  - ...
- High assurance components in presence of other components



seL4 Physical Memory Management

Some kernel memory is statically allocated at boot time

Remainder is divided into untyped (UT) objects

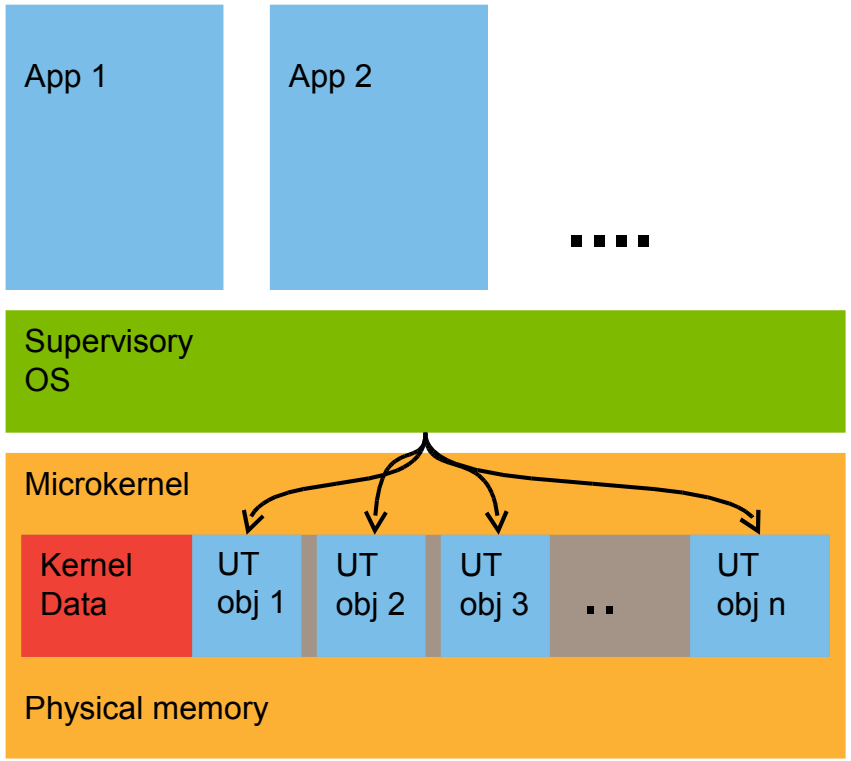
- 2^n region of physical memory
- size aligned

Supervisor gets authority over these objects

- authority conferred by capabilities

Kernel never allocates dynamic memory

- user must provide memory for kernel objects
- re-typing untyped memory to kernel object type



Refinement

- The old story
  - C refines A if all behaviours of C are contained in A
- Sufficient: forward simulation