NICTA

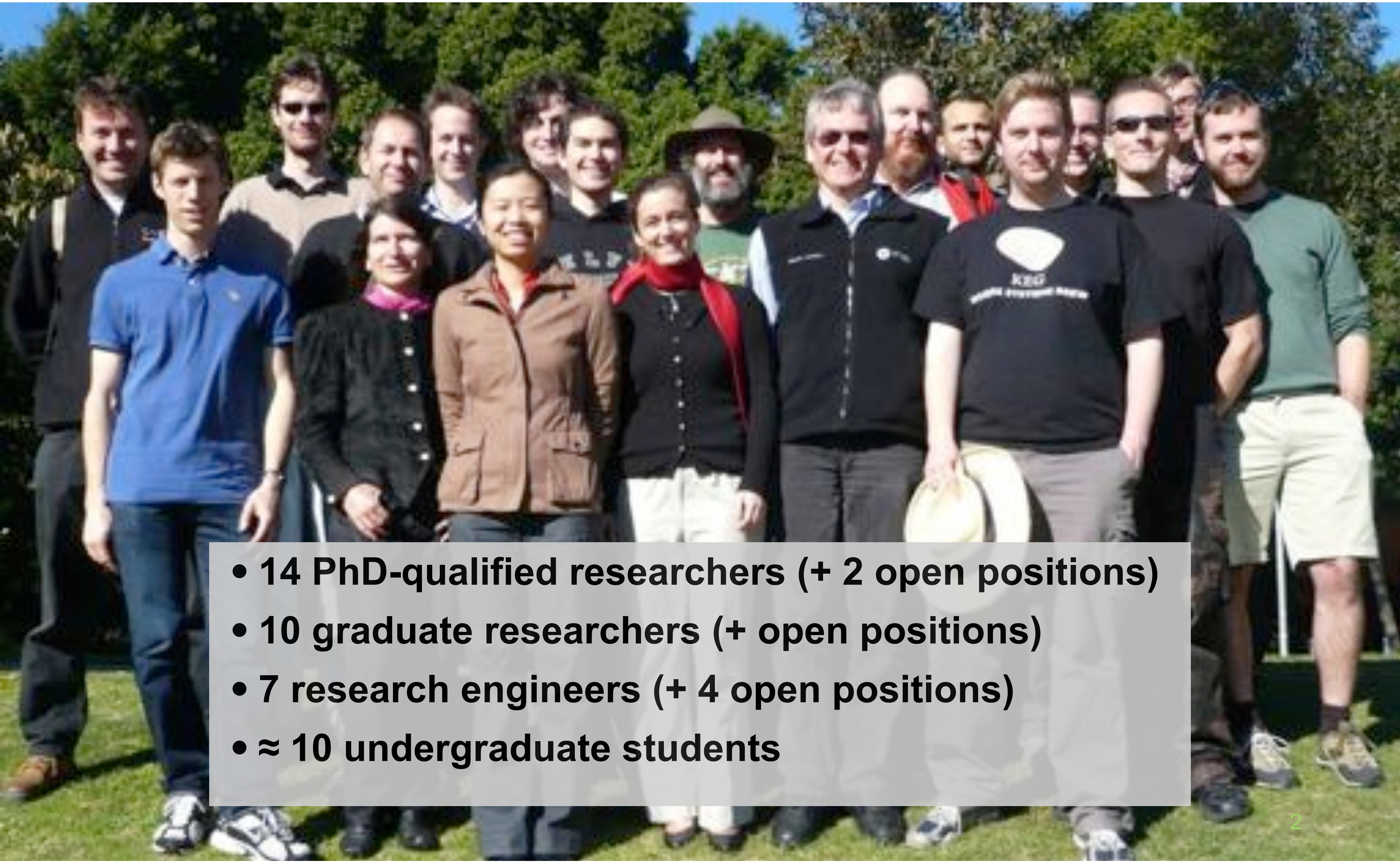
Formally-Verified OS Kernel
A Basis for Reliable Systems?

Gernot Heiser

John Lions Professor of Operating Systems, University of New South Wales
Leader, Trustworthy Embedded Systems, NICTA
CTO and Founder, Open Kernel Labs

Trustworth Embedded Systems
ERTOS.NICTA.com.au

NICTA



- 14 PhD-qualified researchers (+ 2 open positions)
- 10 graduate researchers (+ open positions)
- 7 research engineers (+ 4 open positions)
- ≈ 10 undergraduate students

2


Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue

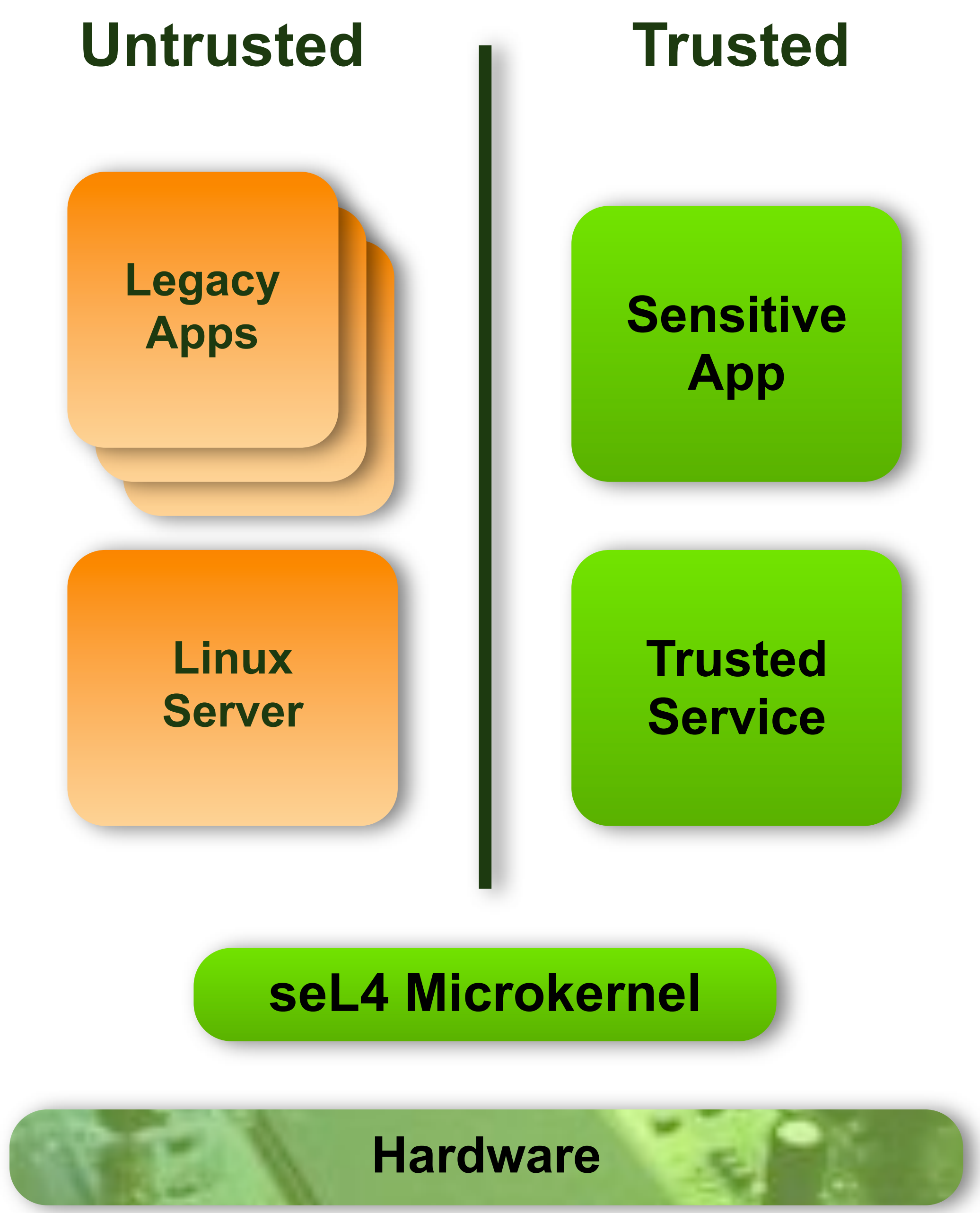
The Problem

seL4 Microkernel
Core of a Minimal TCB

NICTA

Small trustworthy foundation

- Fault isolation
- Fault identification
- IP protection
- Modularity
- High assurance components in presence of other components



Designed for verification

- small API

Designed for security

- novel kernel resource management

Aim: Suitable for Real-World Use

Model: OKL4 microkernel

- resulting from L4-based research at NICTA/UNSW
- Open Kernel Labs spun out as independent company in 2006
- deployed in >500 M devices

seL4 API based on L4:

- IPC
- Threads
- Virtual Memory
- IRQs, exception redirection
- *Capabilities (NEW)*
- **Performance like OKL4!**

Open Kernel Labs™
Be open. Be safe.

seL4 Requirements

NICTA

Real-world deployment for many uses

- General-purpose
  - virtual machines
  - lightweight environents
  - not just a separation kernel
- Performance
- Performance
- Performance
- C & assembler

Verification for *functional correctness*

- Formal model
- Tractable complexity
- Suitable representation of implementation

Kernel Design for Verification

Two Teams

Formal Methods Practitioners

The Power of Abstraction

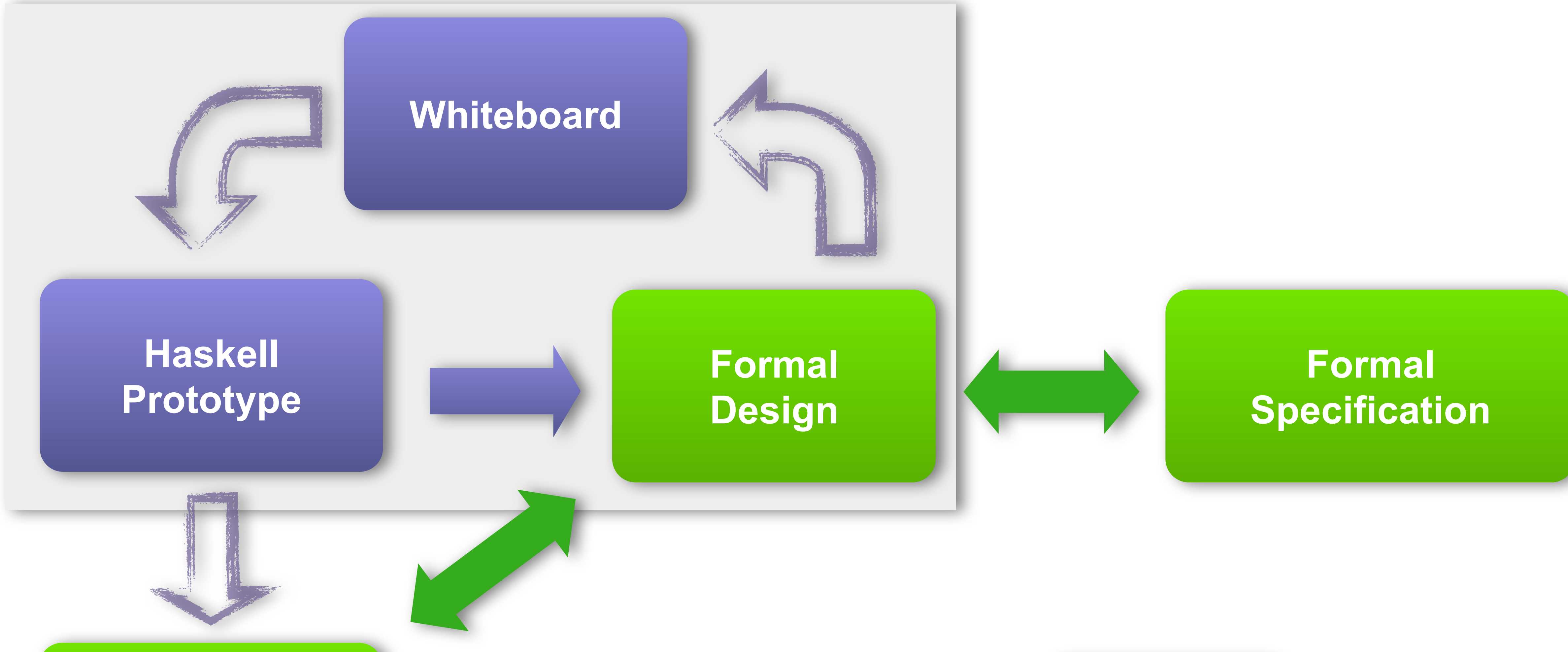
[Liskov 09]

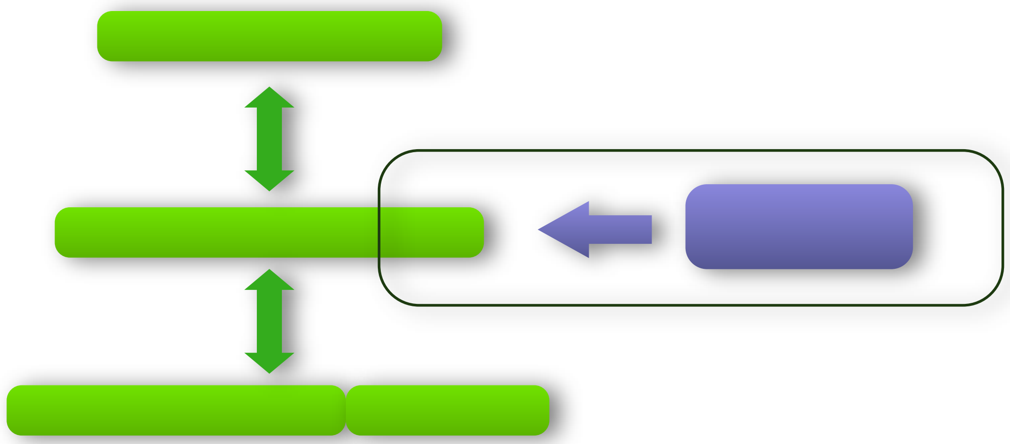
Kernel Developers

Exterminate All OS Abstractions!

[Engler 95]

Iterative Design and Formalisation





Design for Verification

Reducing Complexity

Hardware

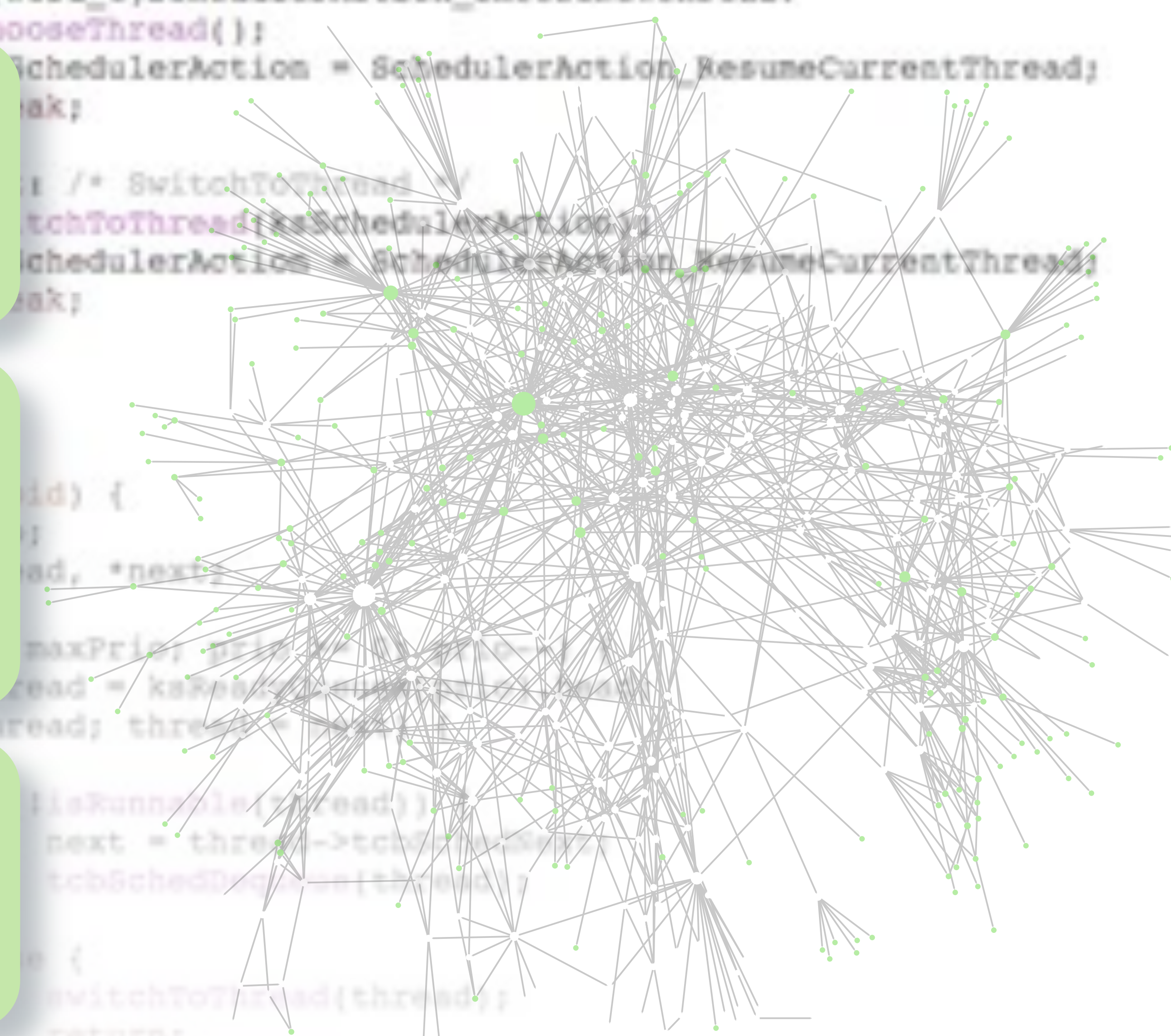
- drivers outside kernel

Concurrency

- event-based kernel
- limit preemption

Code

- derive from functional representation



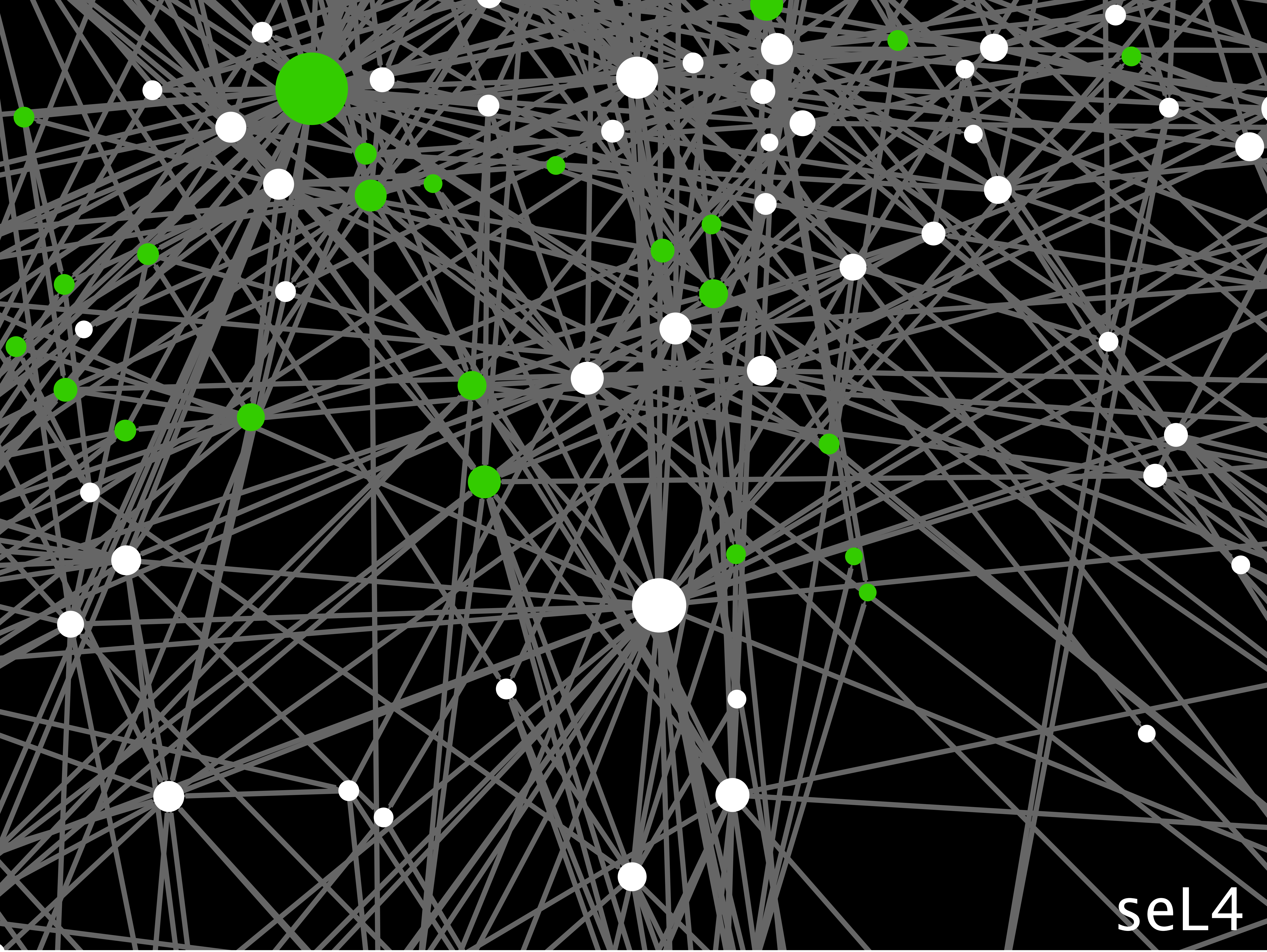
C subset

Everything from C standard

- **including**:
  - pointers, casts, pointer arithmetic
  - data types
  - structs, padding
  - pointers into structs
  - precise finite integer arithmetic

- **minus:**
  - goto, switch fall-through
  - reference to local variable
  - side-effects in expressions
  - function pointers (restricted)
  - unions

- **plus** compiler assumptions on:
  - data layout, encoding, endianess


seL4

The Proof

Functional Correctness*

NICTA

What

Proof

How

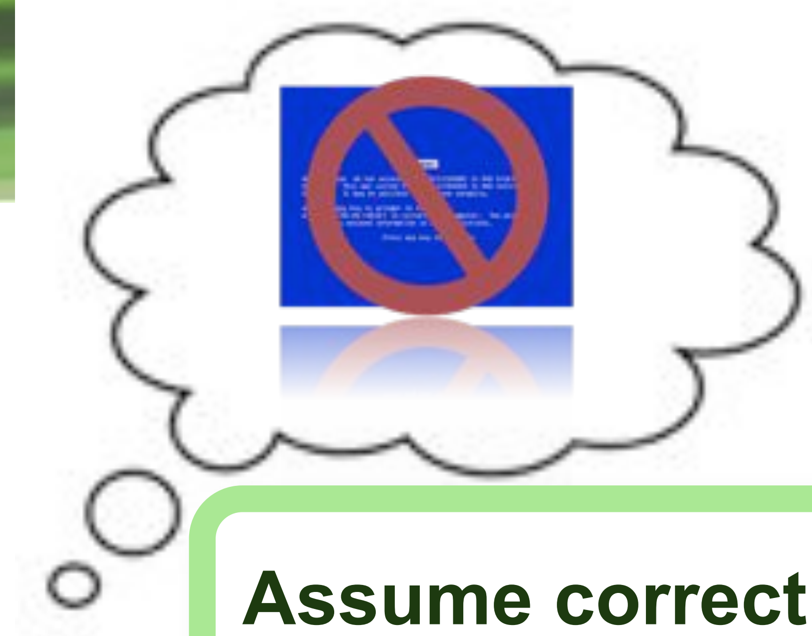
Specification

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

```c
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```

*conditions apply

NICTA



Assume correct:

- compiler + linker (wrt. C op-sem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

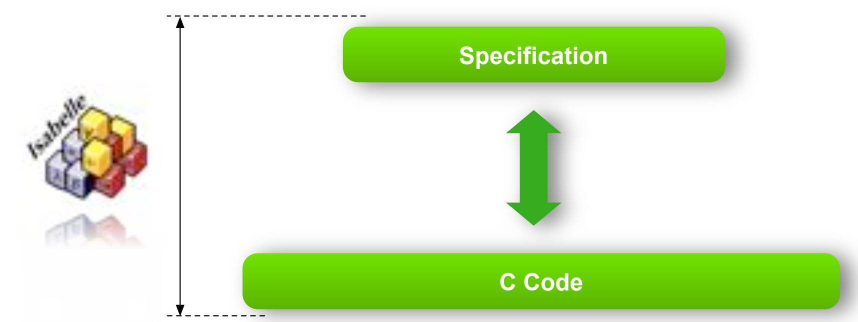
Specificati

Proof

Code

Assumptions

Implications



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:

- "secure" (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

Proof Architecture

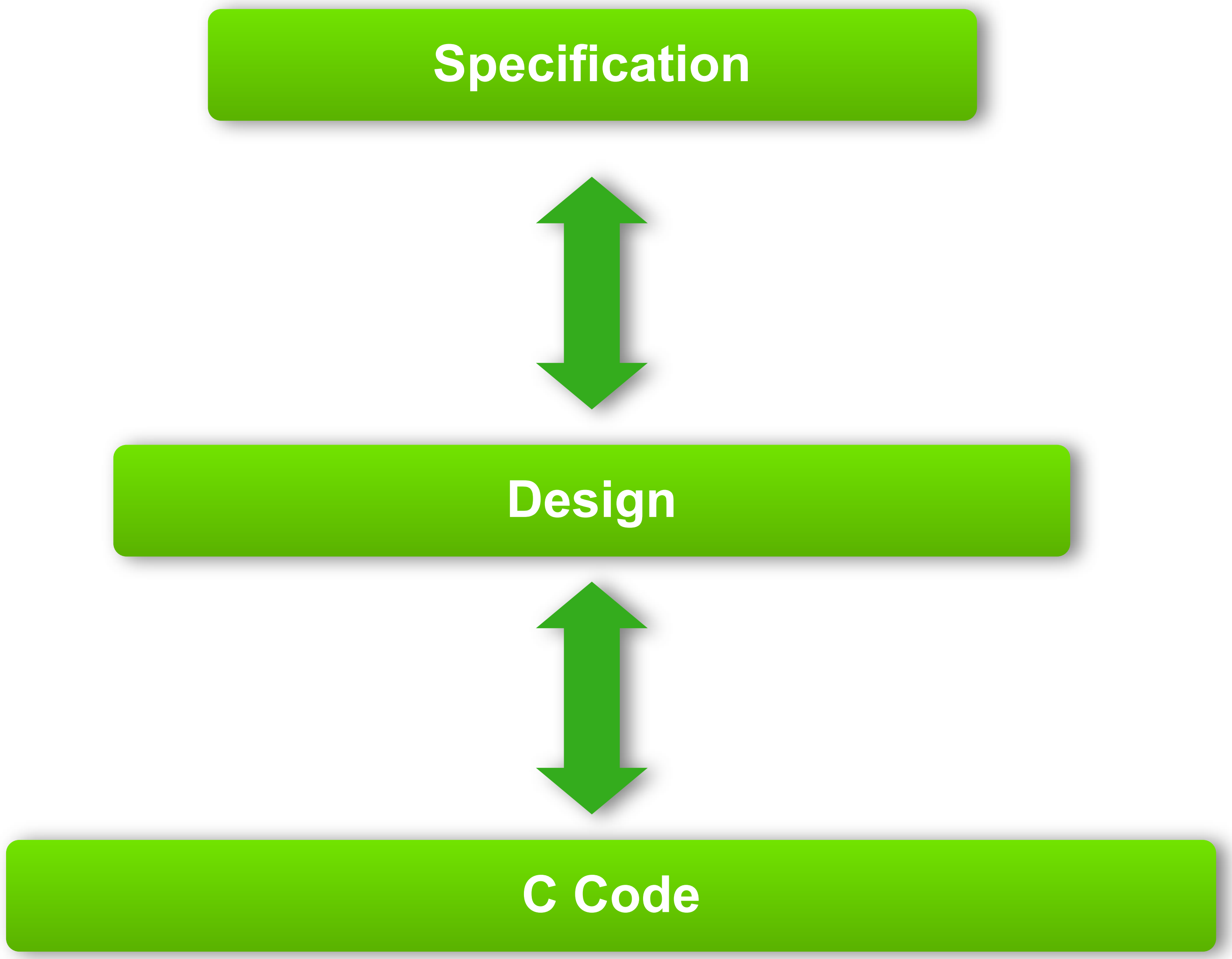
NICTA

Specification

Proof

C Code

Proof Architecture



Proof Architecture

NICTA

Access Control Spec

Confinement

Specification

Design

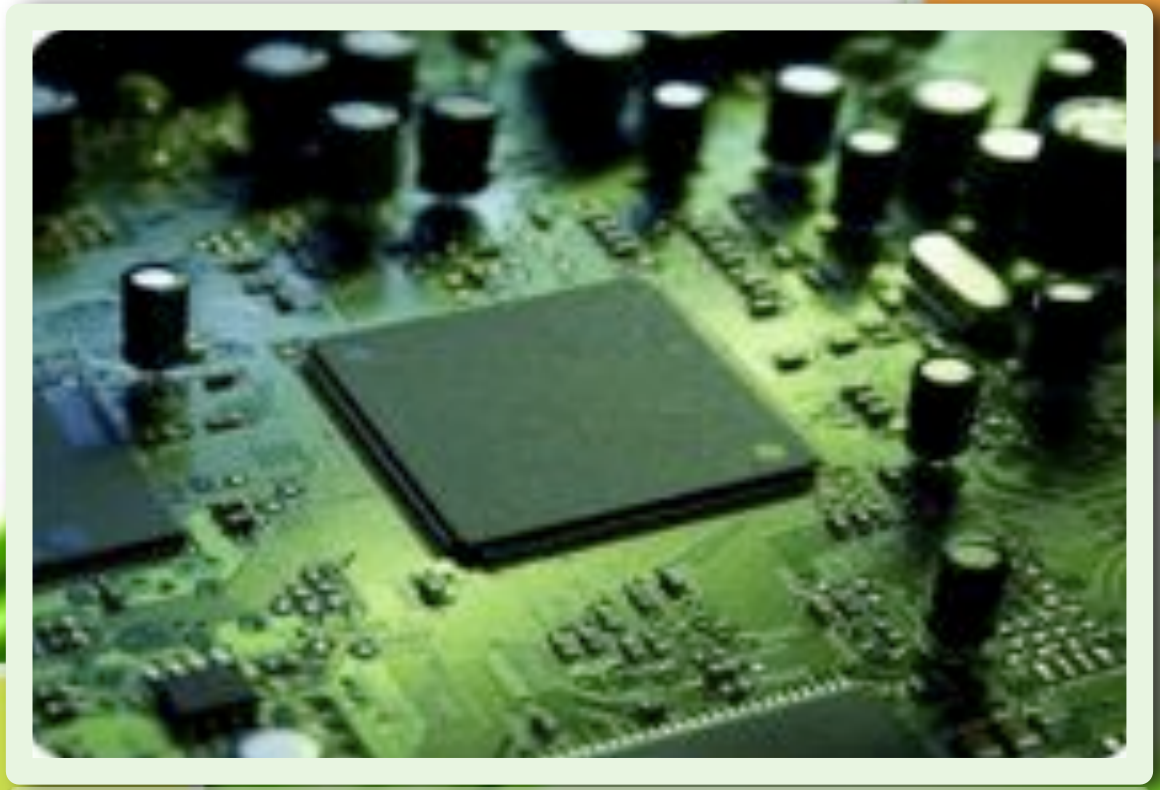
C Code

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
```

```
schedule :: Kernel ()
```

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```

Experience



Common Criteria

EAL	Requirem.	Funct Spec	TDS	Implem.
EAL1		Informal		
EAL2		Informal	Informal	
EAL3		Informal	Informal	
EAL4		Informal	Informal	Informal
EAL5		Semiformal	Semiformal	Informal
EAL6	Formal	Semiformal	Semiformal	Informal
EAL7	Formal	Formal	Formal	Informal
l4.verified	Formal	Formal	Formal	Formal

Did you find any Bugs?

Bugs found

during testing: 16

during verification:

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

Effort

Haskell design	2	py
First C impl.	2	weeks
Debugging/Testing	2	months
Kernel verification	12	py
Formal frameworks	10	py
Total	25	py

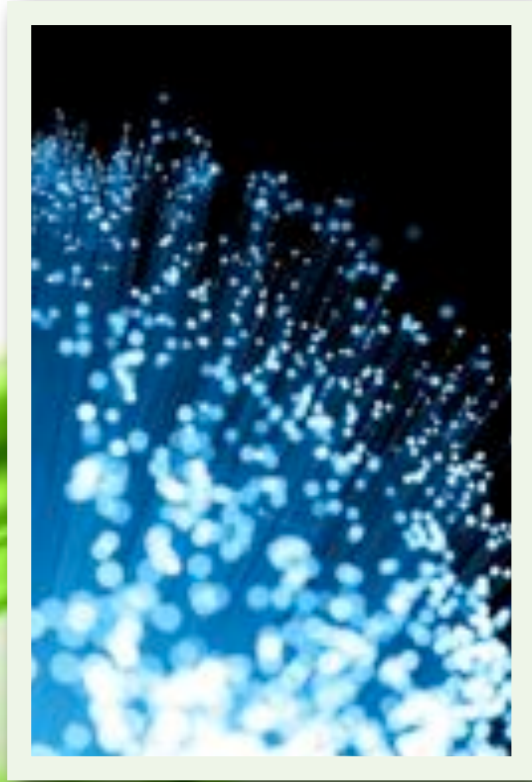
Comparison of approaches

Trad. engineering	4-6	py
Repeat verification	6	py

Cost

Common Criteria EAL6: **$60M**
L4.verified: **$6M**

What’s next?



Future Work: Trustworthy *Systems*

NICTA

Remove limitations

- verify assembler code
- verify bootstrap code
- verify MMU operations
- multicore version
- verify x86 version
- temporal isolation
- information flow

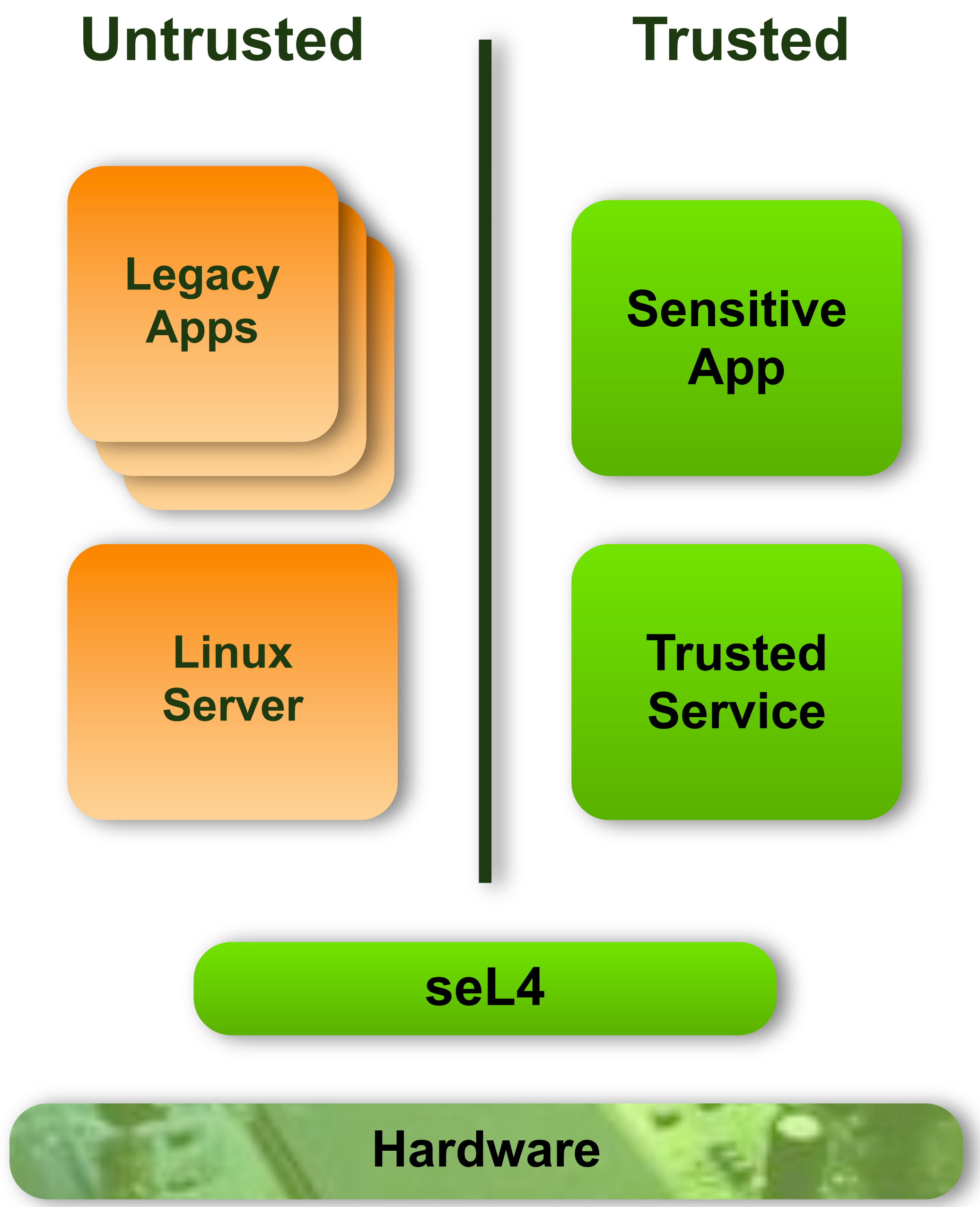
Towards real systems

- 1 MLoC, legacy components
- real-time analysis
- power management

How?

Exploit:

- seL4 isolation
- verified properties
- MILS architectures / virtualization



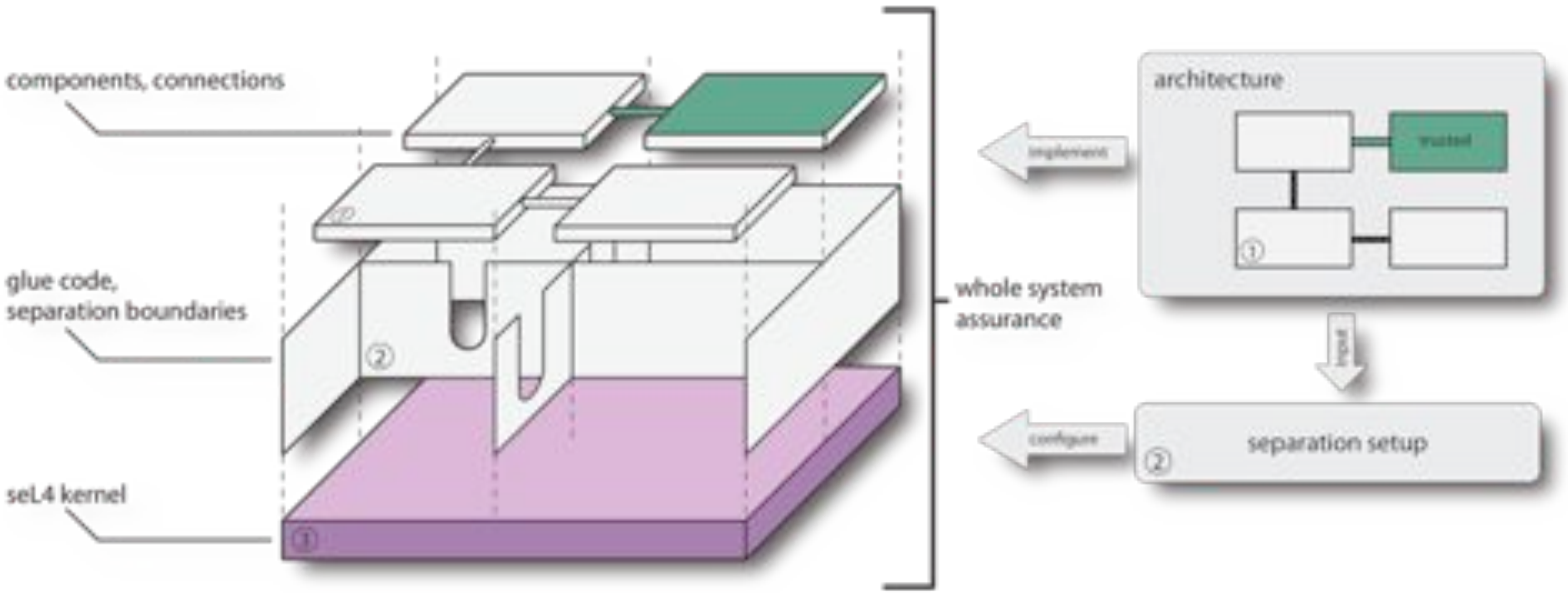
Application Areas

Multilevel Secure Terminal Demonstrator

also:

- **automotive**
- **financial**
- **aerospace**

Global View of Project



- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup

Summary

NICTA

Formal proof all the way from spec to C

- **200 kLoC** handwritten, machine-checked proof, **10 k** theorems
- ~**460** bugs (160 in C)
- Verification on **code**, **design**, and **spec**
- **Hard in the proof** → **Hard in the implementation**

Formal Code Verification up to 10 kLoC:

It works.
It's feasible.
It's cheaper.

(It's fun, too...)

The Team (Past and Present)

- June Andronick
- Timothy Bourke
- Andrew Boyton
- David Cock
- Jeremy Dawson
- Philip Derrin
- Dhammika Elkaduwe
- **Kevin Elphinstone**
  - *leader, kernel design*
- Kai Engelhardt
- David Greenaway
- Lukas Haenel
- Gernot Heiser
- **Gerwin Klein**
  - *leader, verification*
- Rafal Kolanski
- Jia Meng
- Catherine Menon
- Michael Norrish
- Thomas Sewell
- David Tsai
- Harvey Tuch
- Michael von Tessin
- Adam Walker
- Simon Winwood

Thank You