seL4

Formal Verification of an OS Kernel

Gerwin Klein
June Andronick
Dhammika Elkaduwe
Michael Norrish

Kevin Elphinstone
David Cock
Kai Engelhardt
Thomas Sewell
Simon Winwood

Gernot Heiser
Philip Derrin
Rafal Kolanski
Harvey Tuch

NICTA

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
ANU
UNSW
Department of State and Regional Development
NICTA Partners
Victoria
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT
The University Of Queensland

L4Verified

1 microkernel

8,700 lines of C

0 bugs*

qed

*conditions apply

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

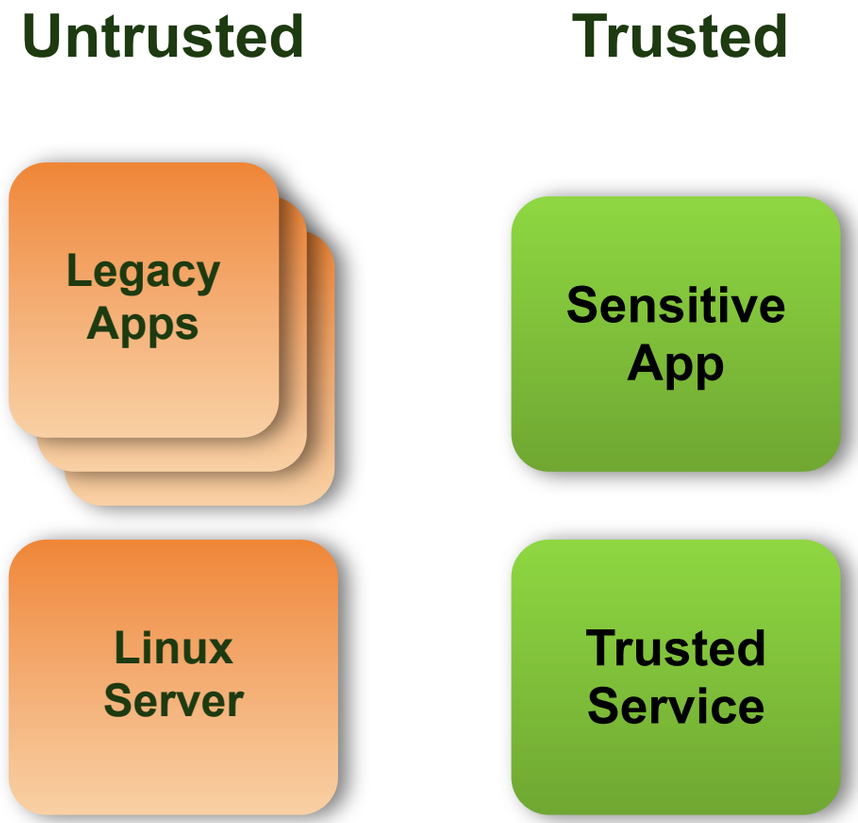
The Problem

Small Kernels

NICTA

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components



seL4 API:

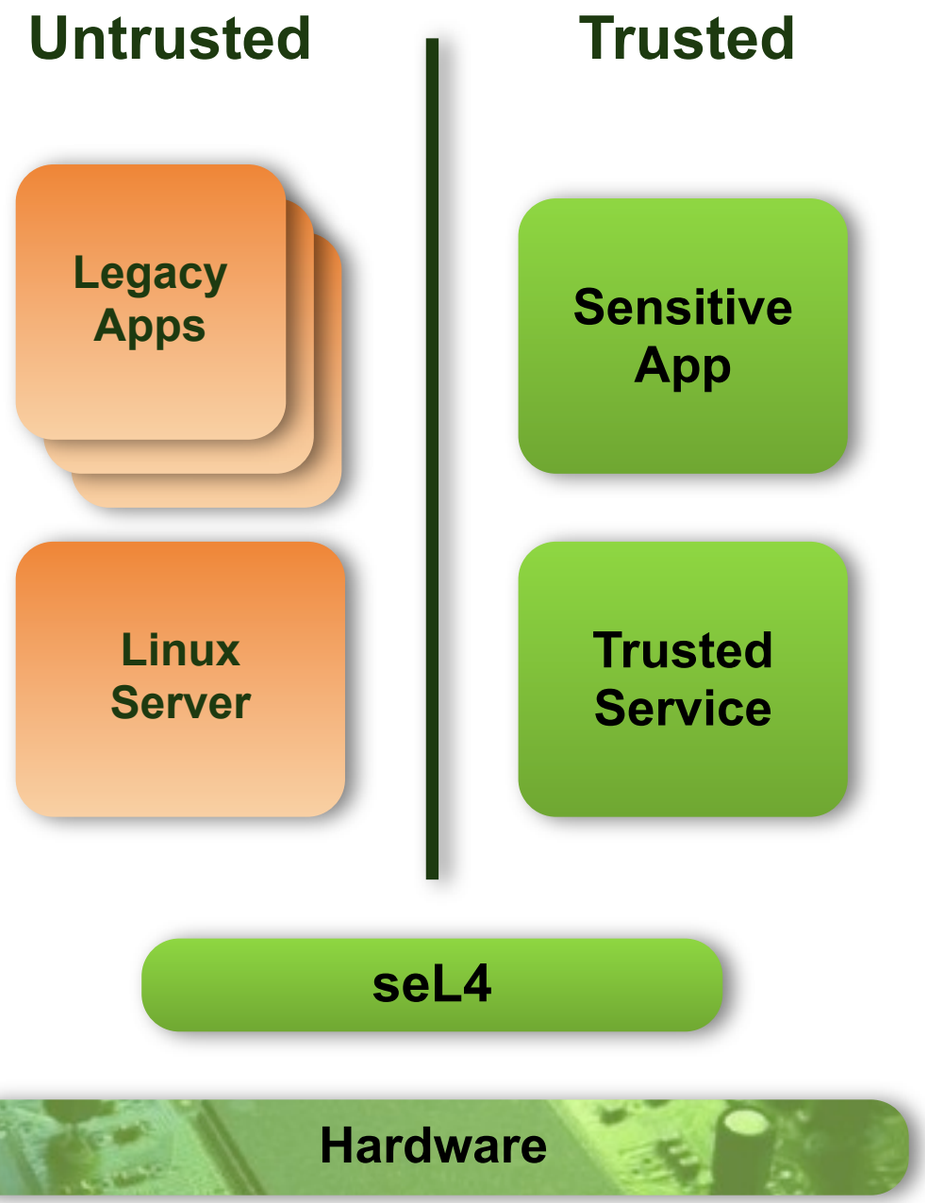
- IPC
- Threads
- VM
- IRQ
- Capabilities

Small Kernels

NICTA

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components



seL4 API:

- IPC
- Threads
- VM
- IRQ
- Capabilities

The Proof

The Proof

Functional Correctness

NICTA

Specification

Proof

Code

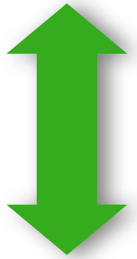
Functional Correctness

What

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

Specification

Proof



Code

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
```

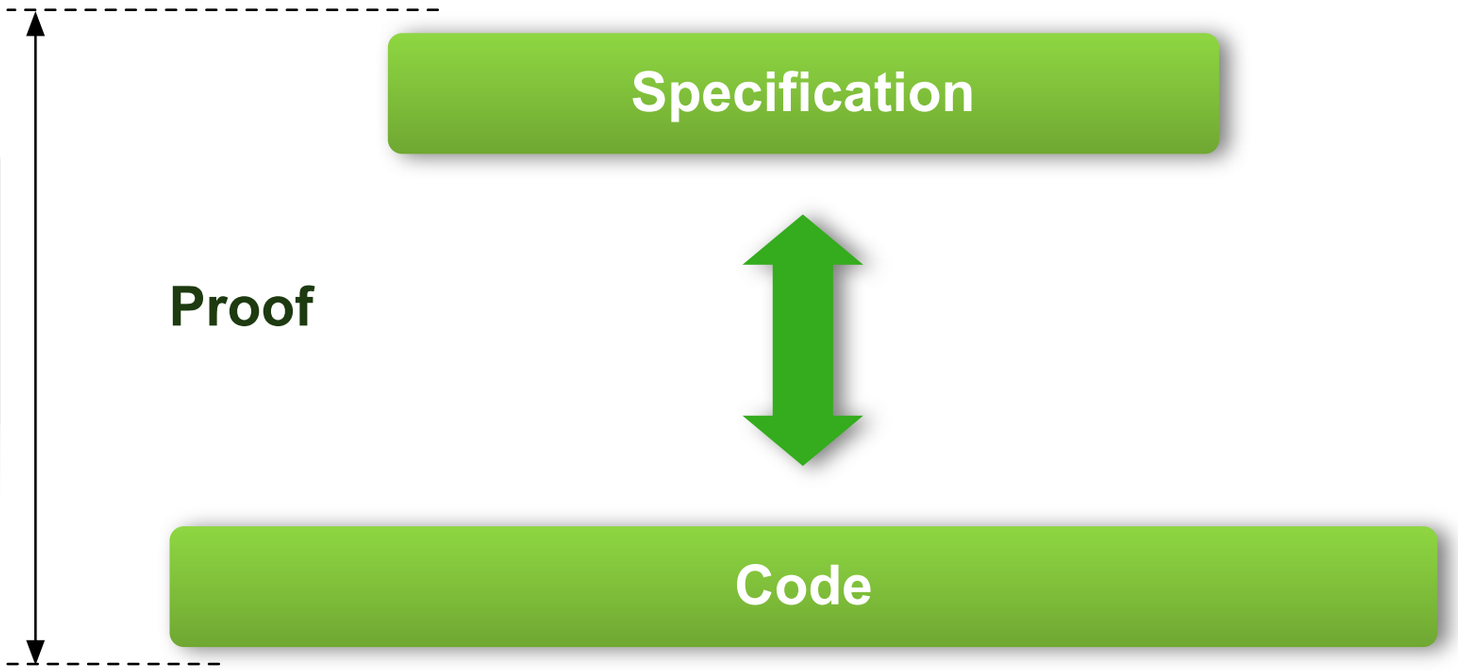
```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        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



*conditions apply

NICTA

Specification

Proof

Code

Assumptions

*conditions apply

NICTA

Expectation

Specification

Proof

Code

Assumptions

*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

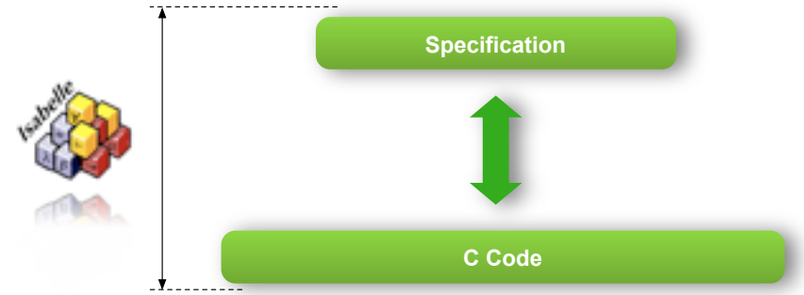
Assumptions

Implications

NICTA

Execution always defined:

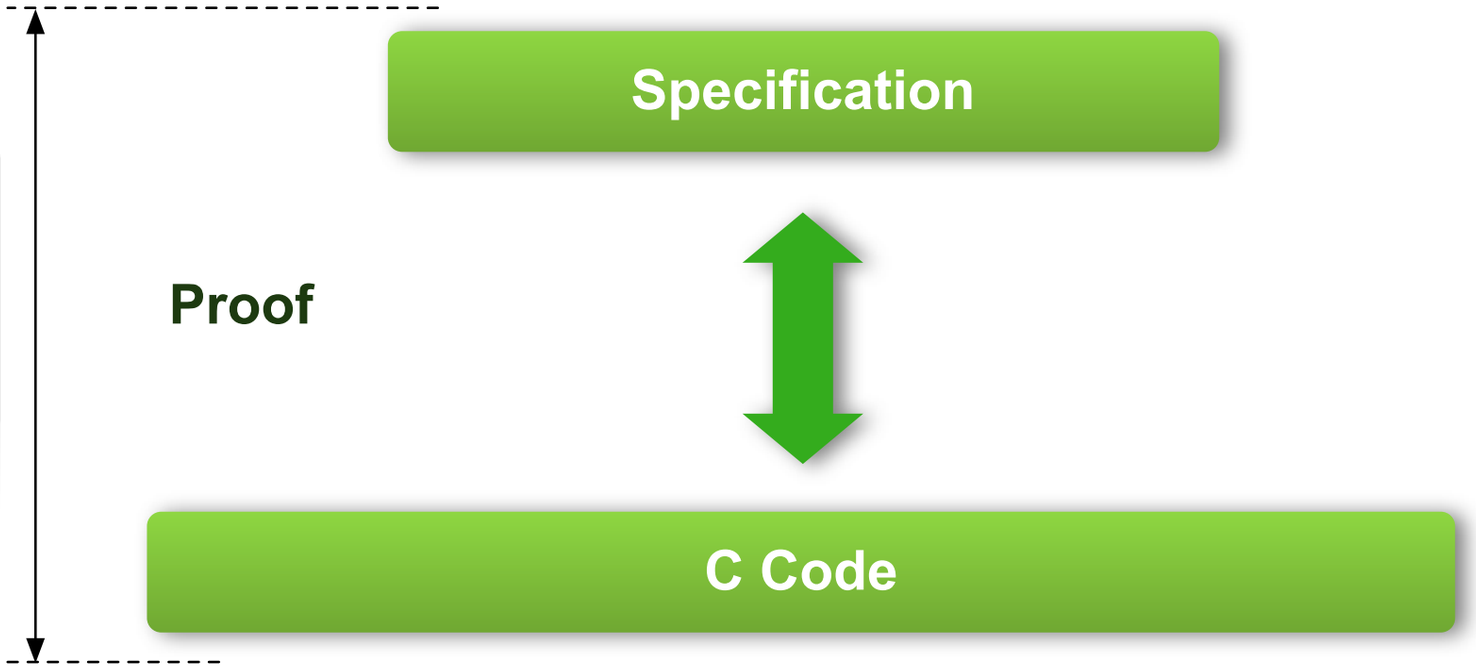
- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion



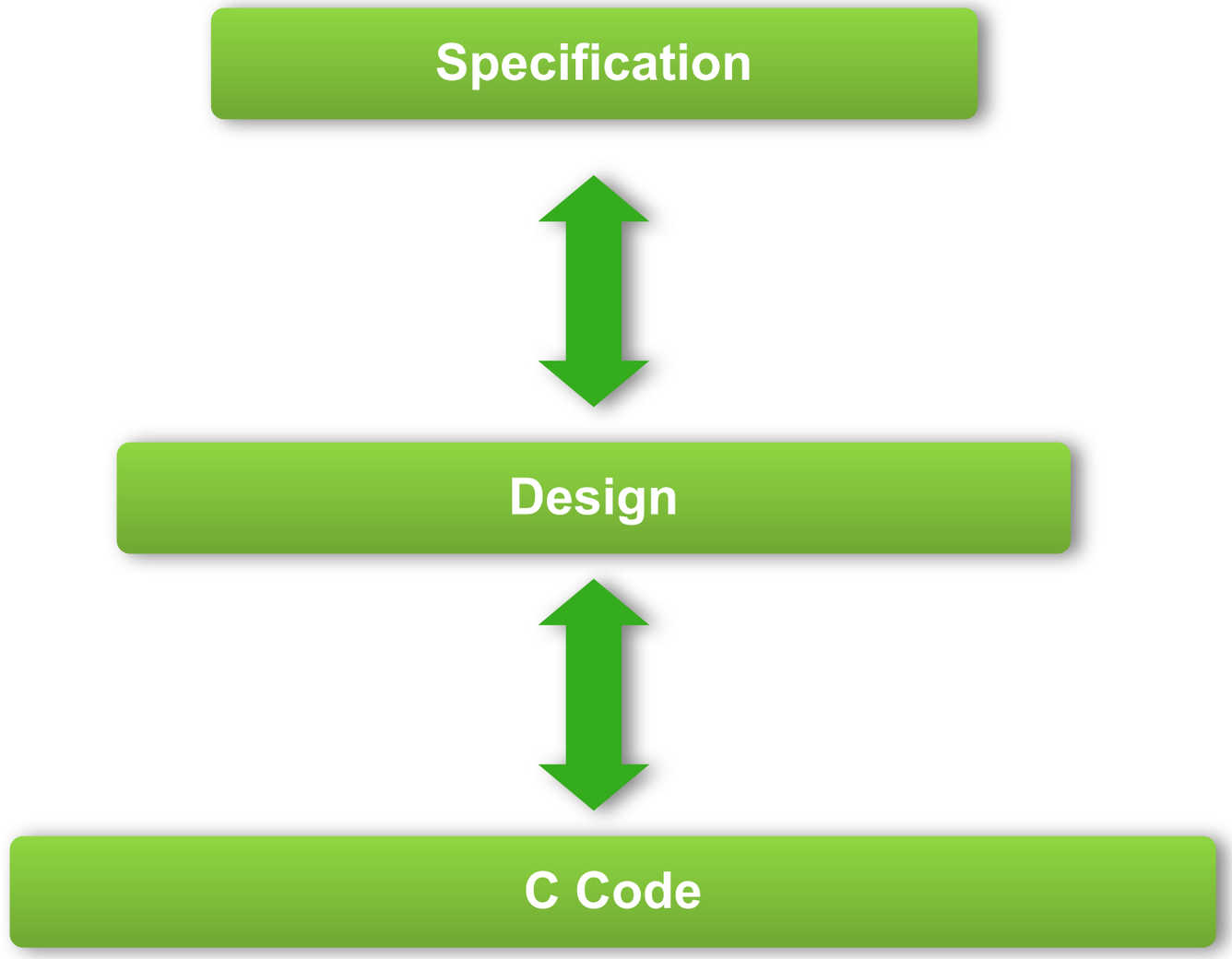
Not implied:

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

Proof Architecture

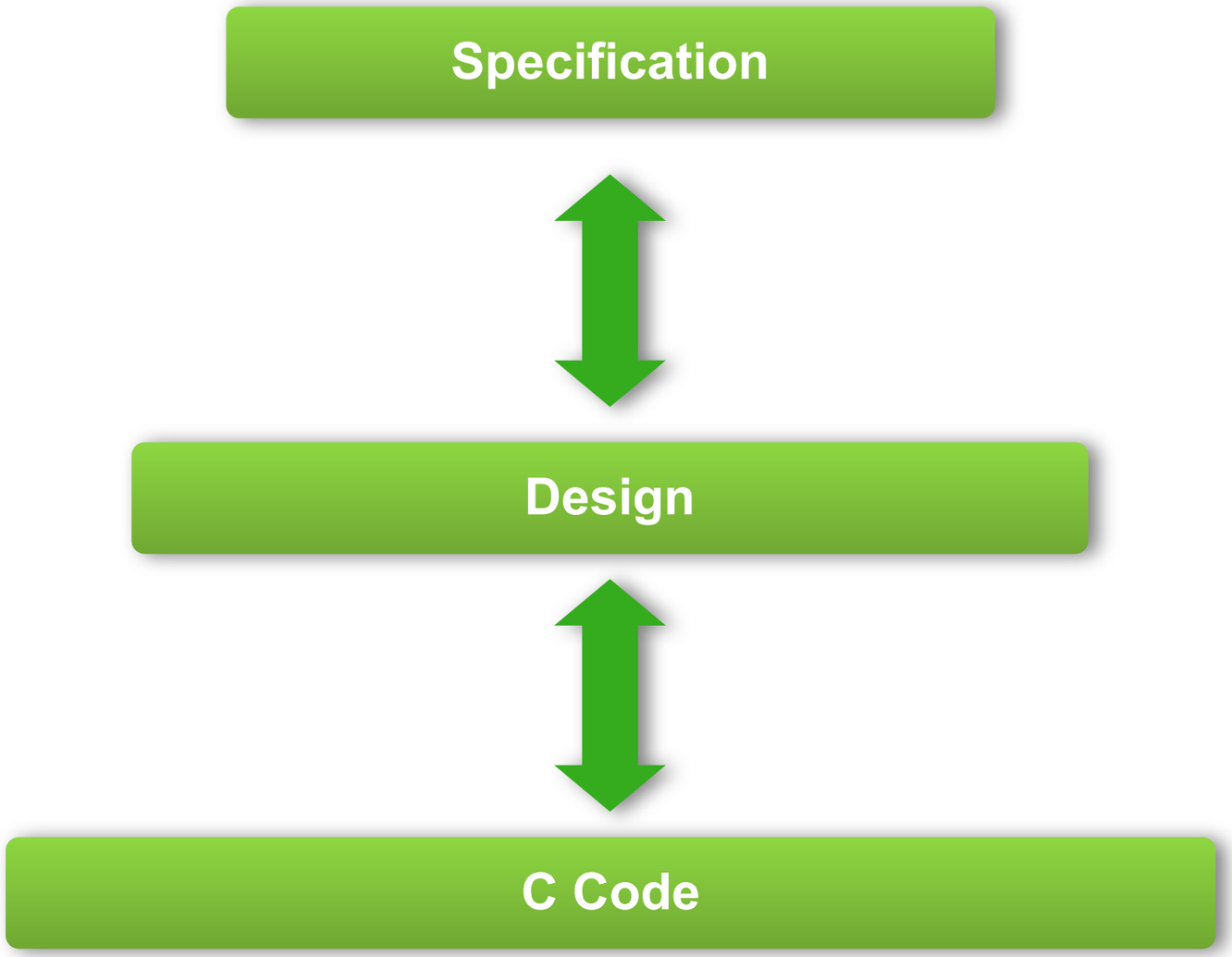


Proof Architecture

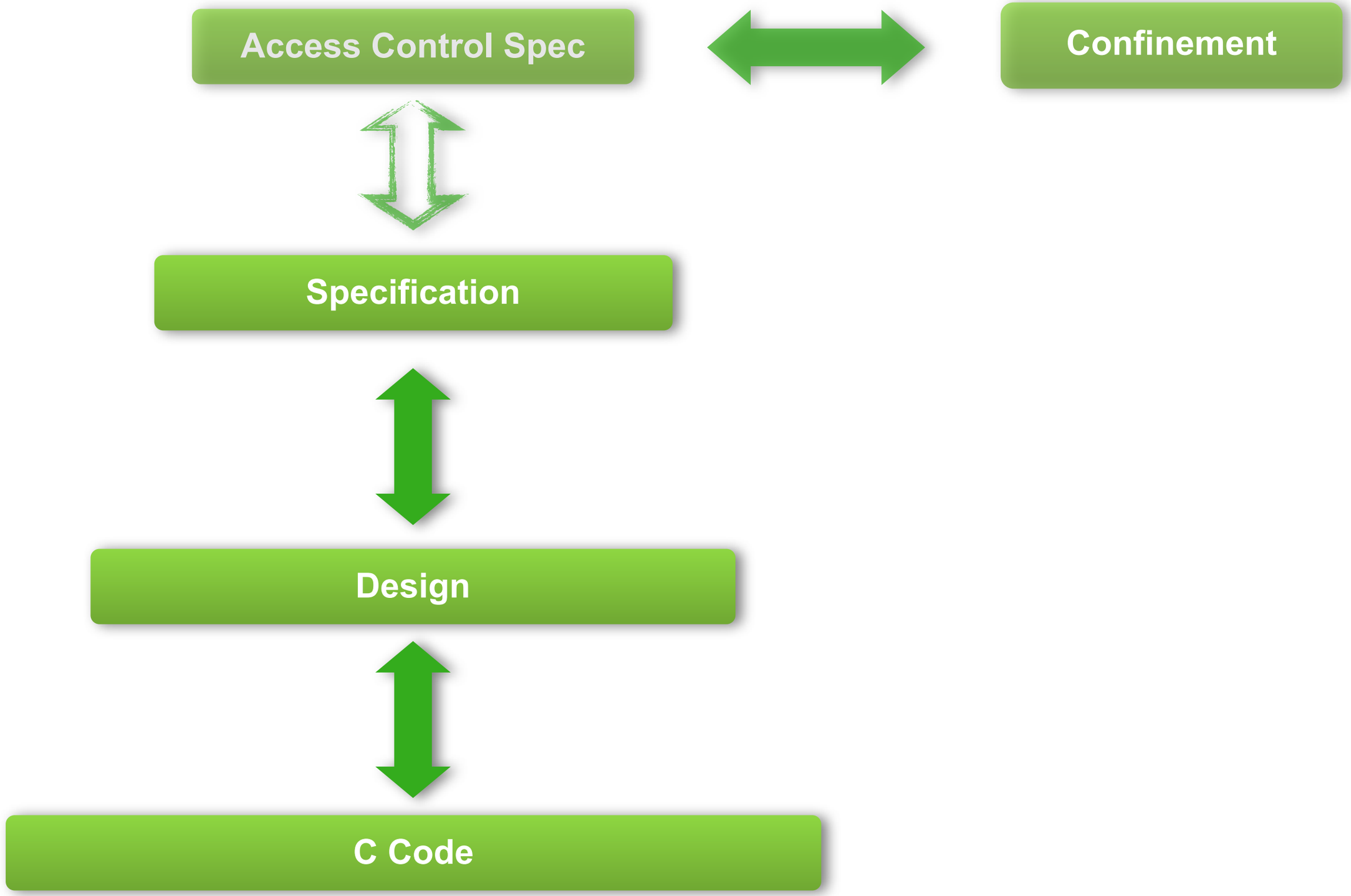


Proof Architecture

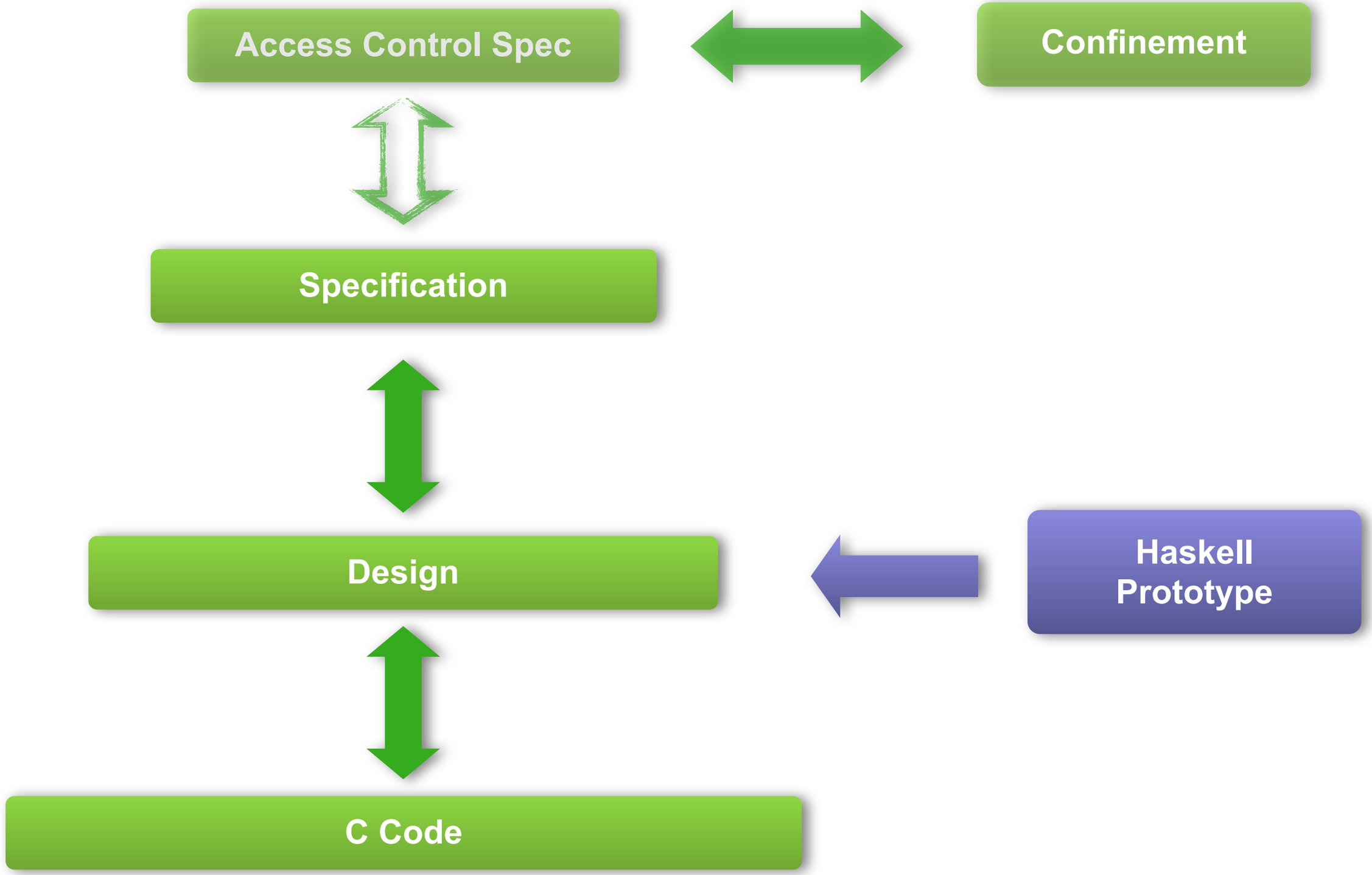
NICTA



Proof Architecture



Proof Architecture



Proof Architecture

NICTA

Access Control Spec

Confinement

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

Specification

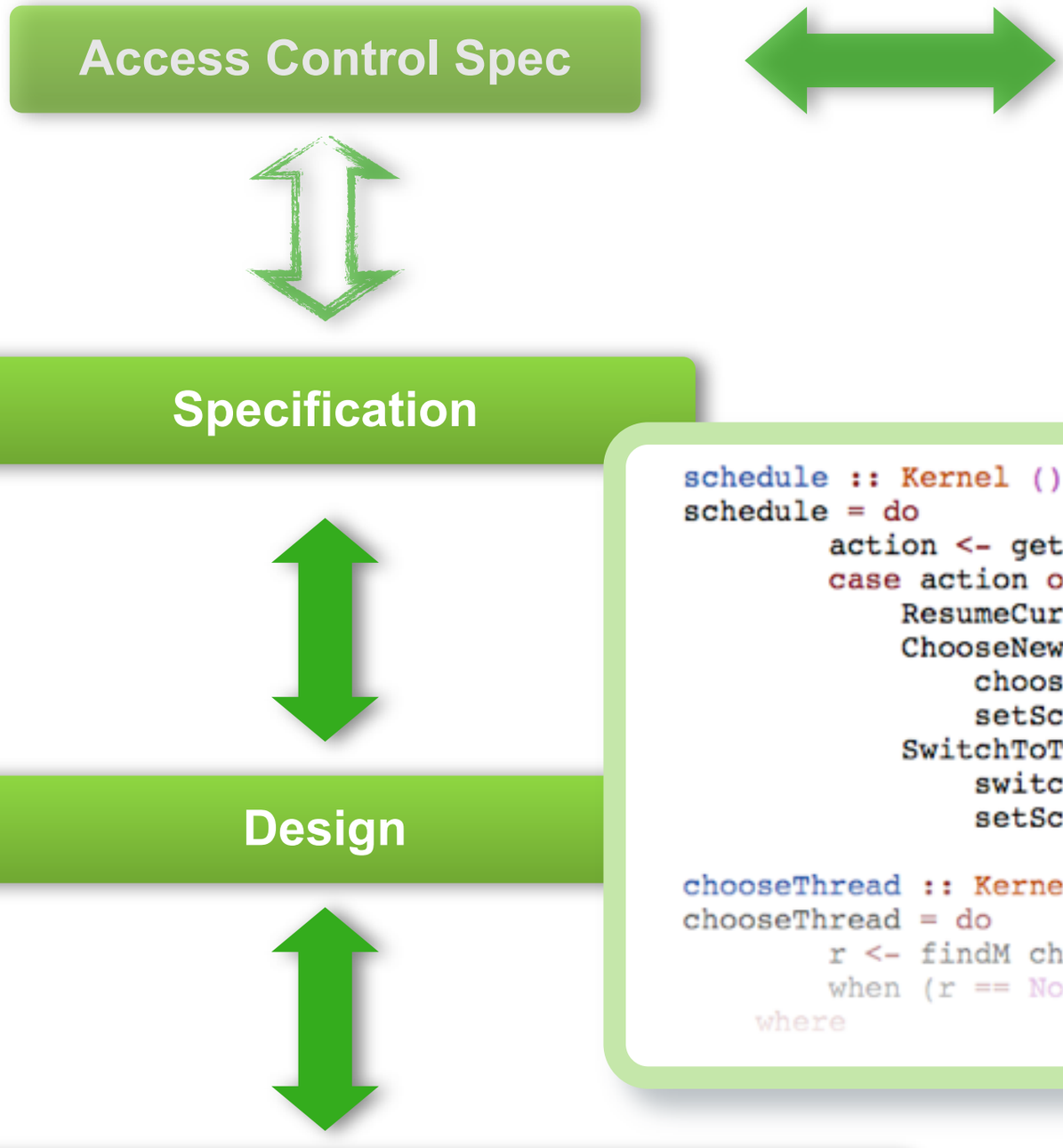
Design

Haskell Prototype

C Code

Proof Architecture

NICTA




```
schedule :: Kernel ()
schedule = do
        action <- getSchedulerAction
        case action of
            ResumeCurrentThread -> return ()
            ChooseNewThread -> do
                chooseThread
                setSchedulerAction ResumeCurrentThread
            SwitchToThread t -> do
                switchToThread t
                setSchedulerAction ResumeCurrentThread

chooseThread :: Kernel ()
chooseThread = do
        r <- findM chooseThread' (reverse [minBound .. maxBound])
        when (r == Nothing) $ switchToIdleThread
    where
```

Proof Architecture

NICTA

Access Control Spec

Confinement

Specification

Design

C Code

```
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;
```

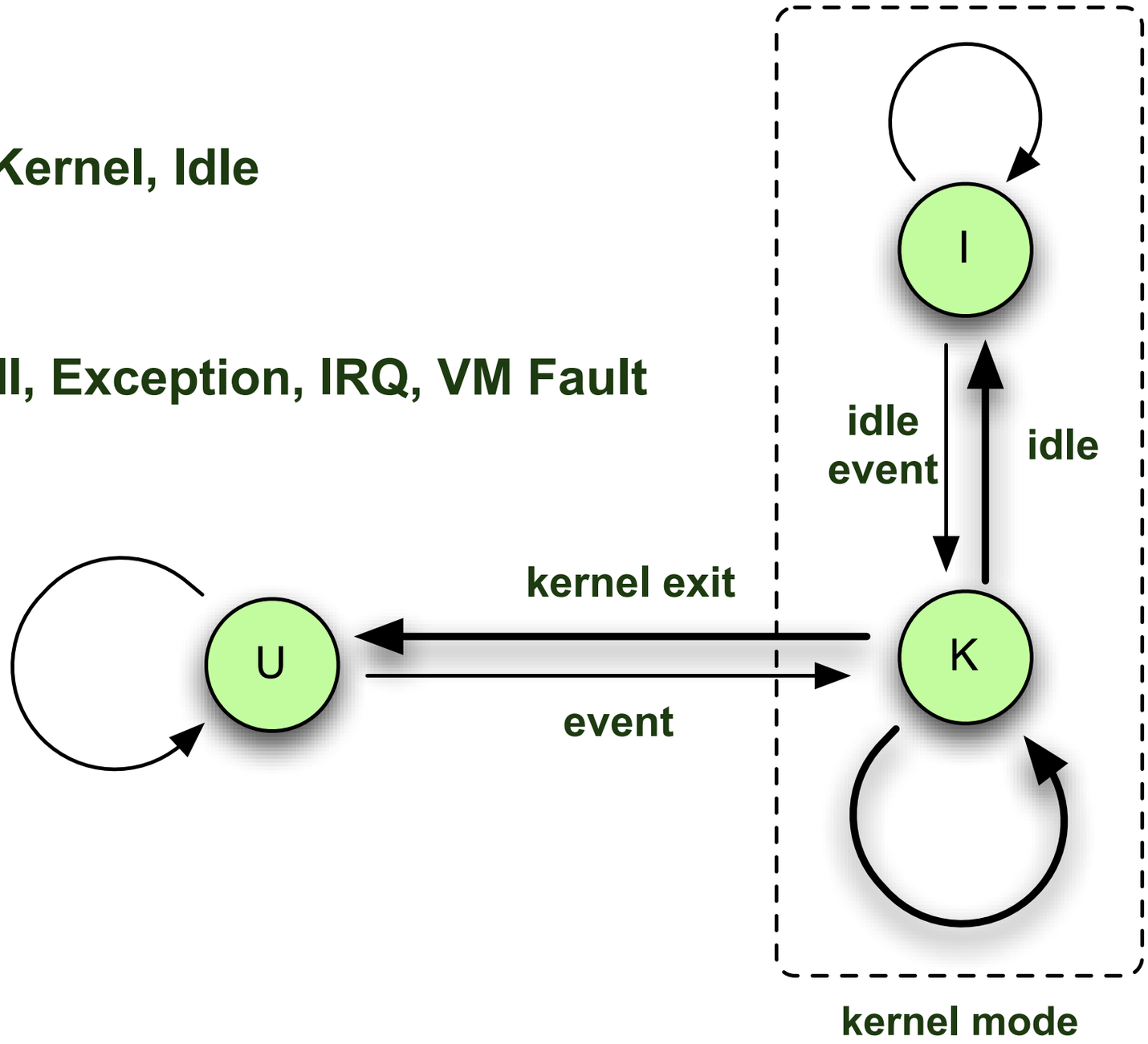
System Model

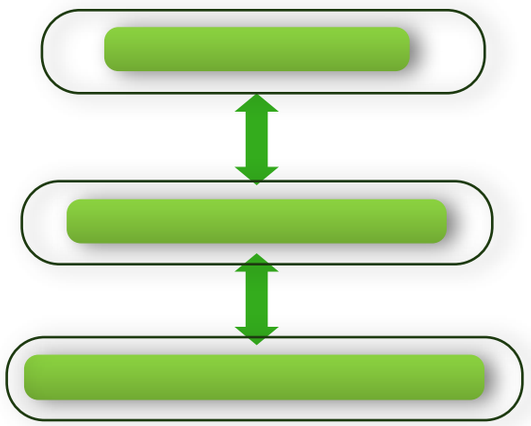
States:

User, Kernel, Idle

Events:

Syscall, Exception, IRQ, VM Fault





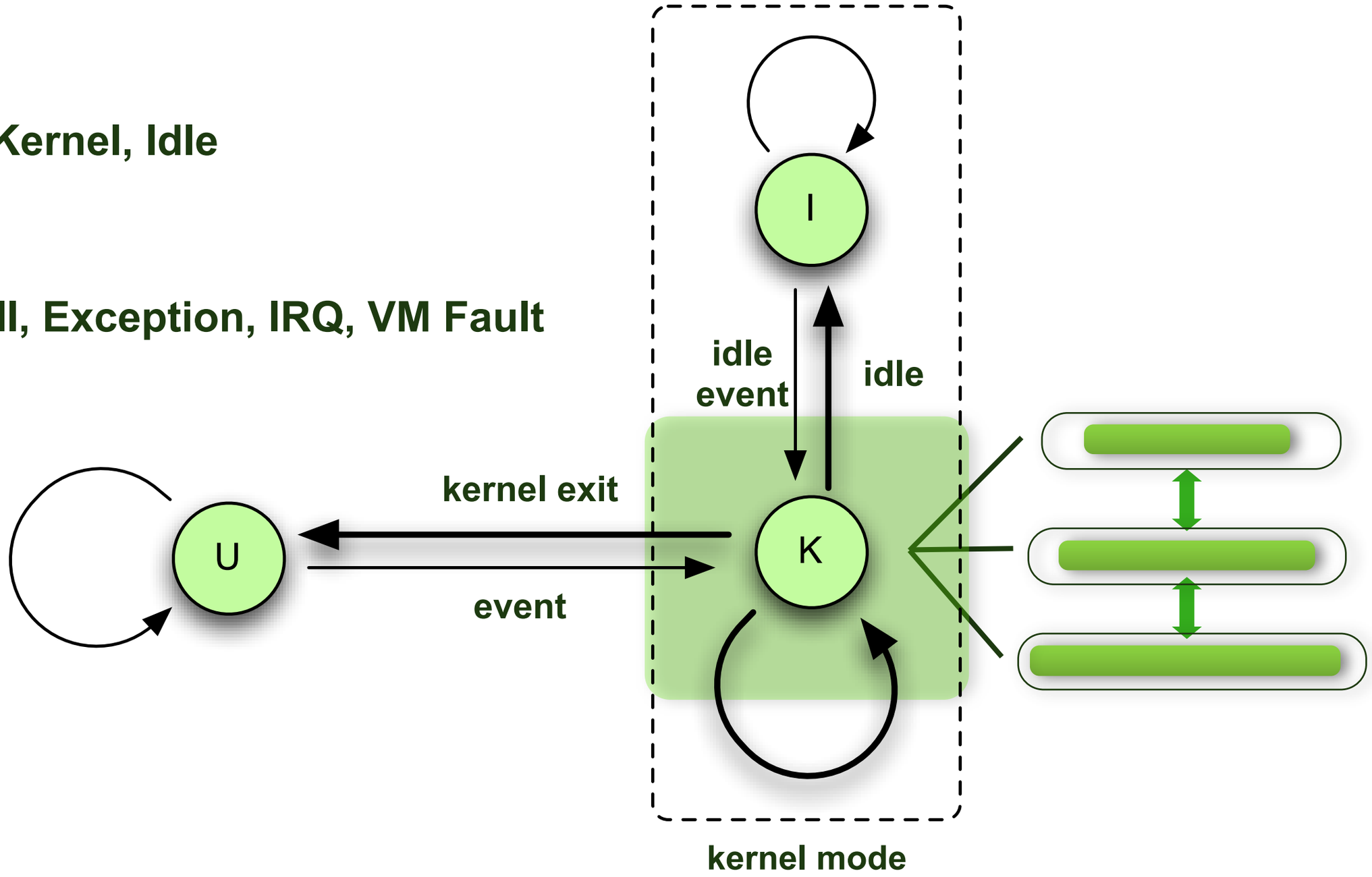
System Model

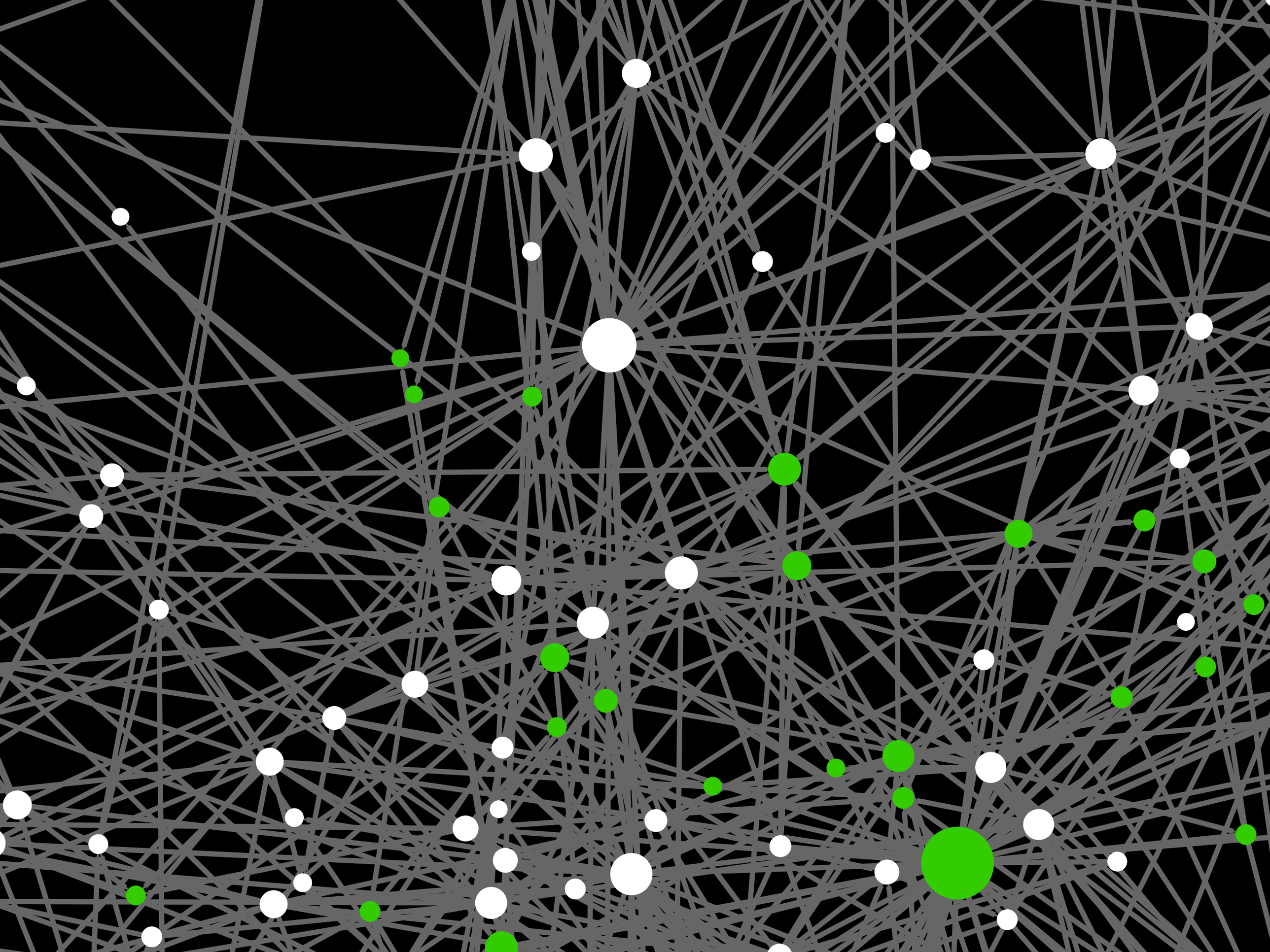
States:

User, Kernel, Idle

Events:

Syscall, Exception, IRQ, VM Fault

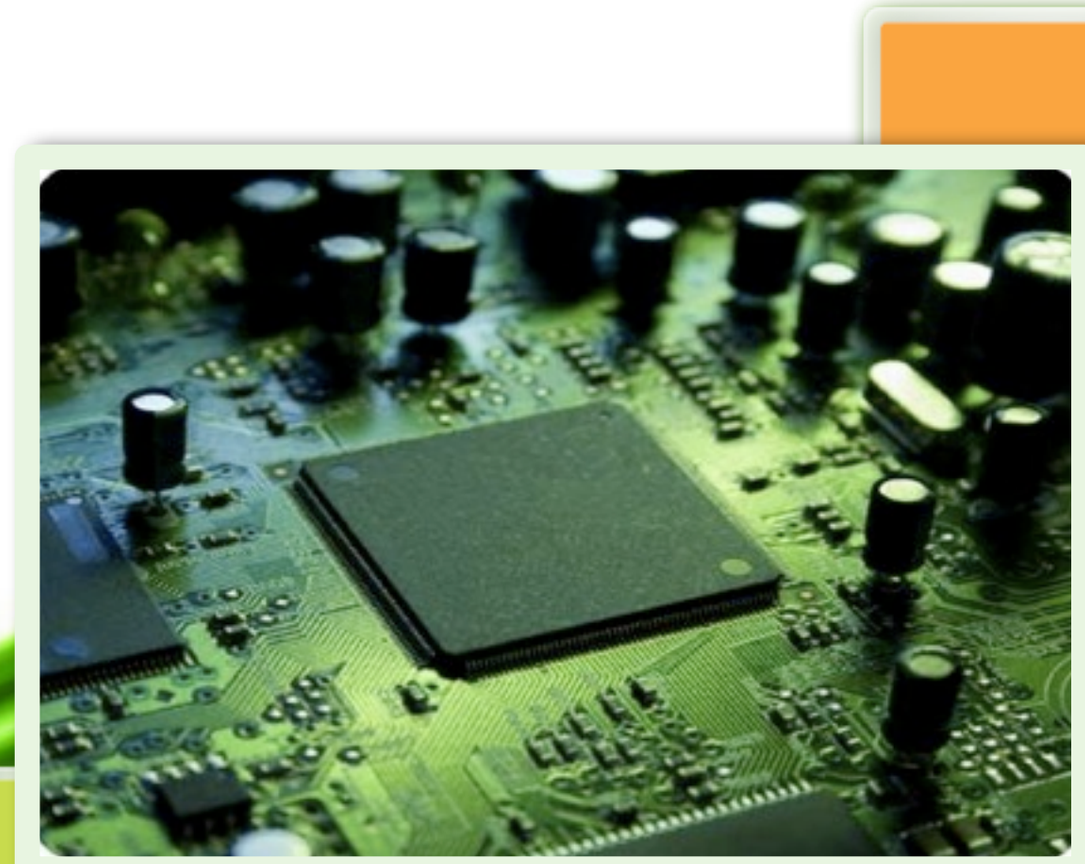





seL4

Kernel Design for Verification

Kernel Design for Verification



Two Teams

NICTA

Formal Methods Practitioners

Kernel Developers

Two Teams

NICTA

Formal Methods Practitioners

The Power of Abstraction

(Liskov 09)

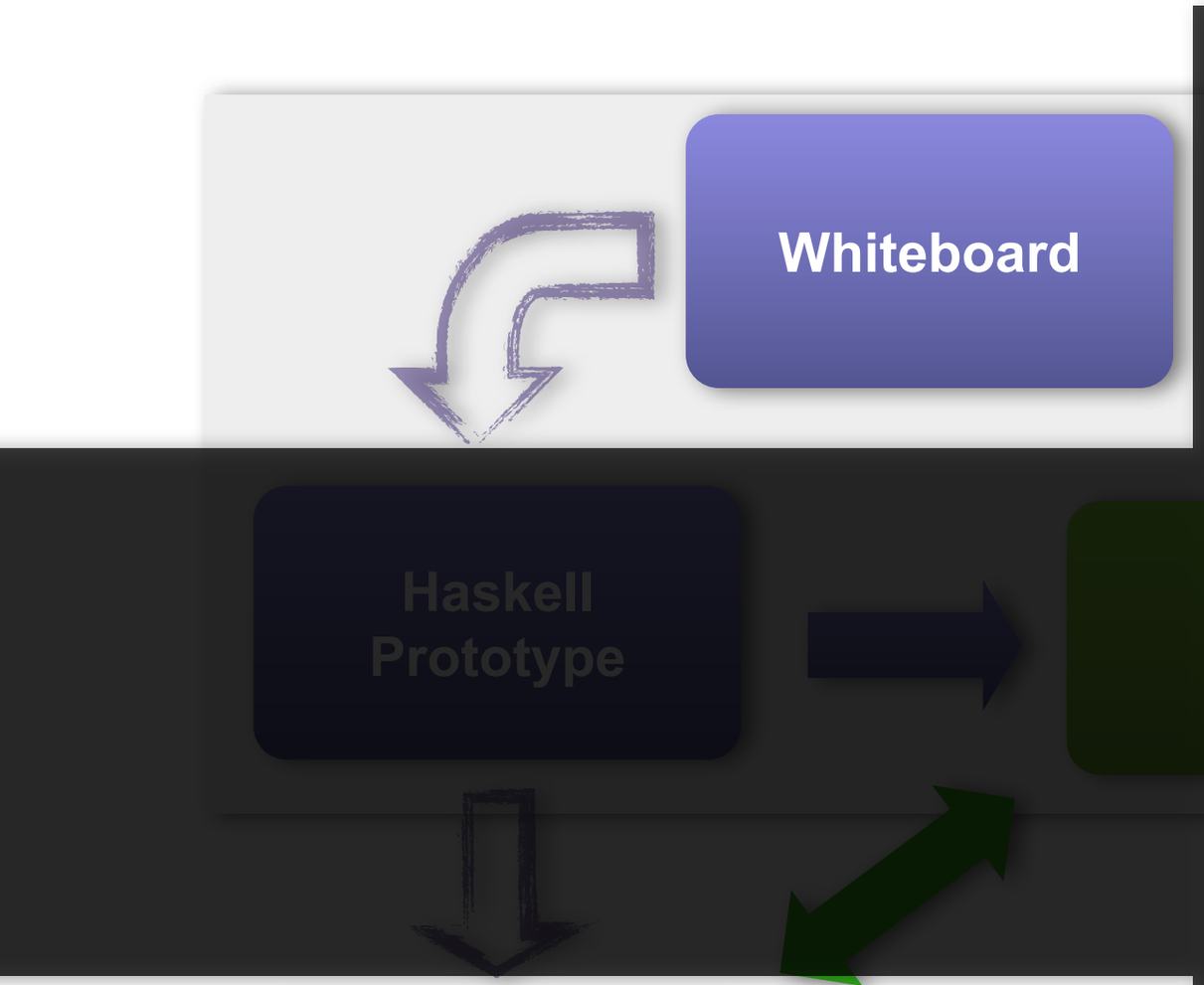
Kernel Developers

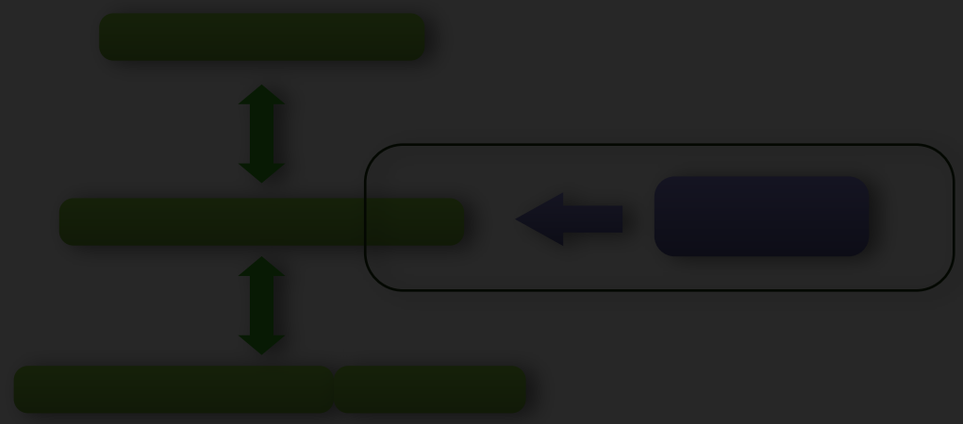
Exterminate All OS Abstractions!

(Engler 95)

Iterative Design and Formalisation

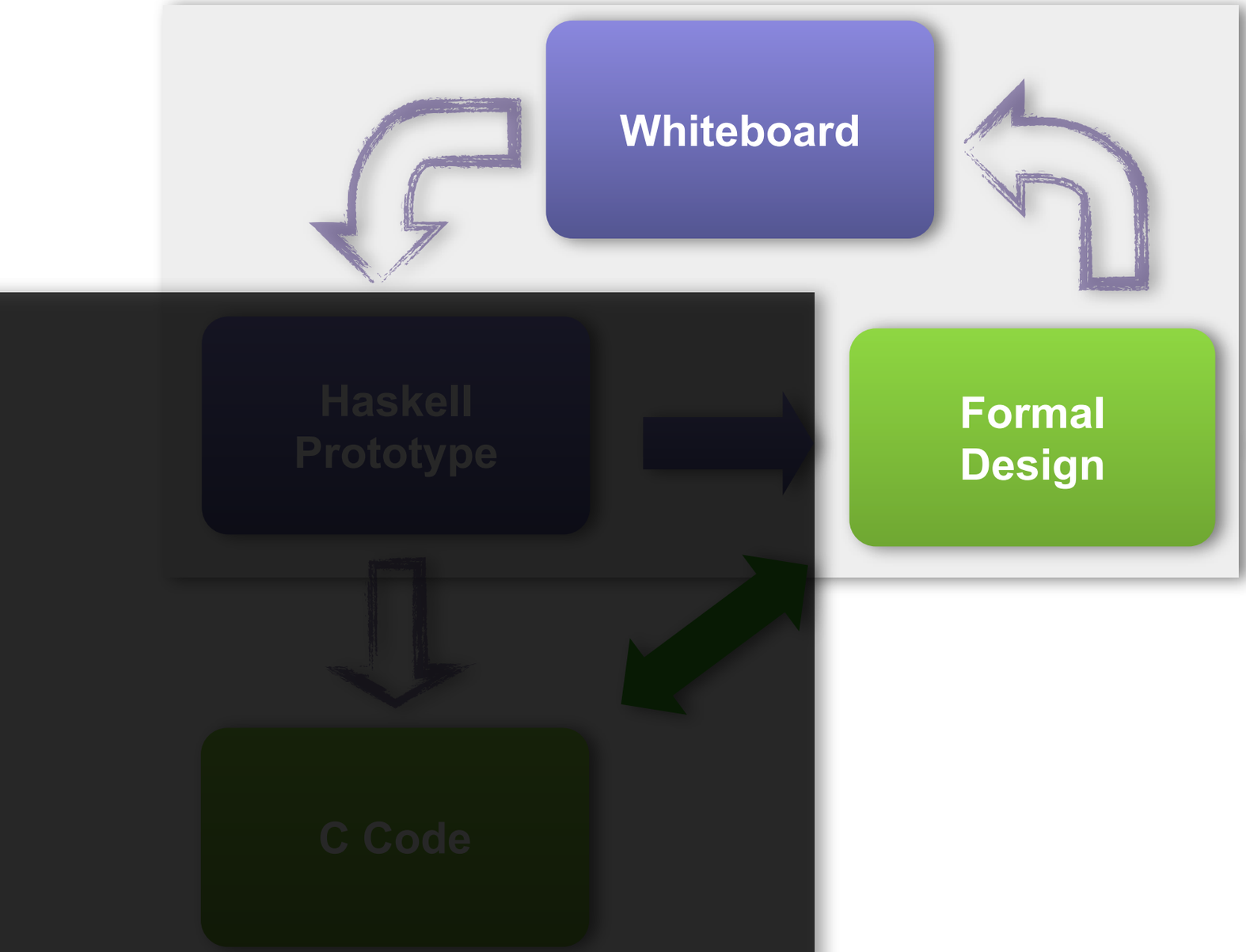
NICTA

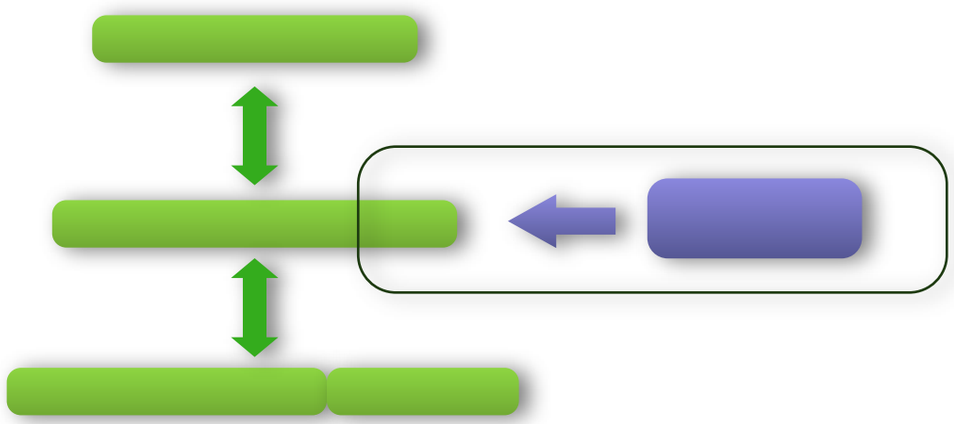




Iterative Design and Formalisation

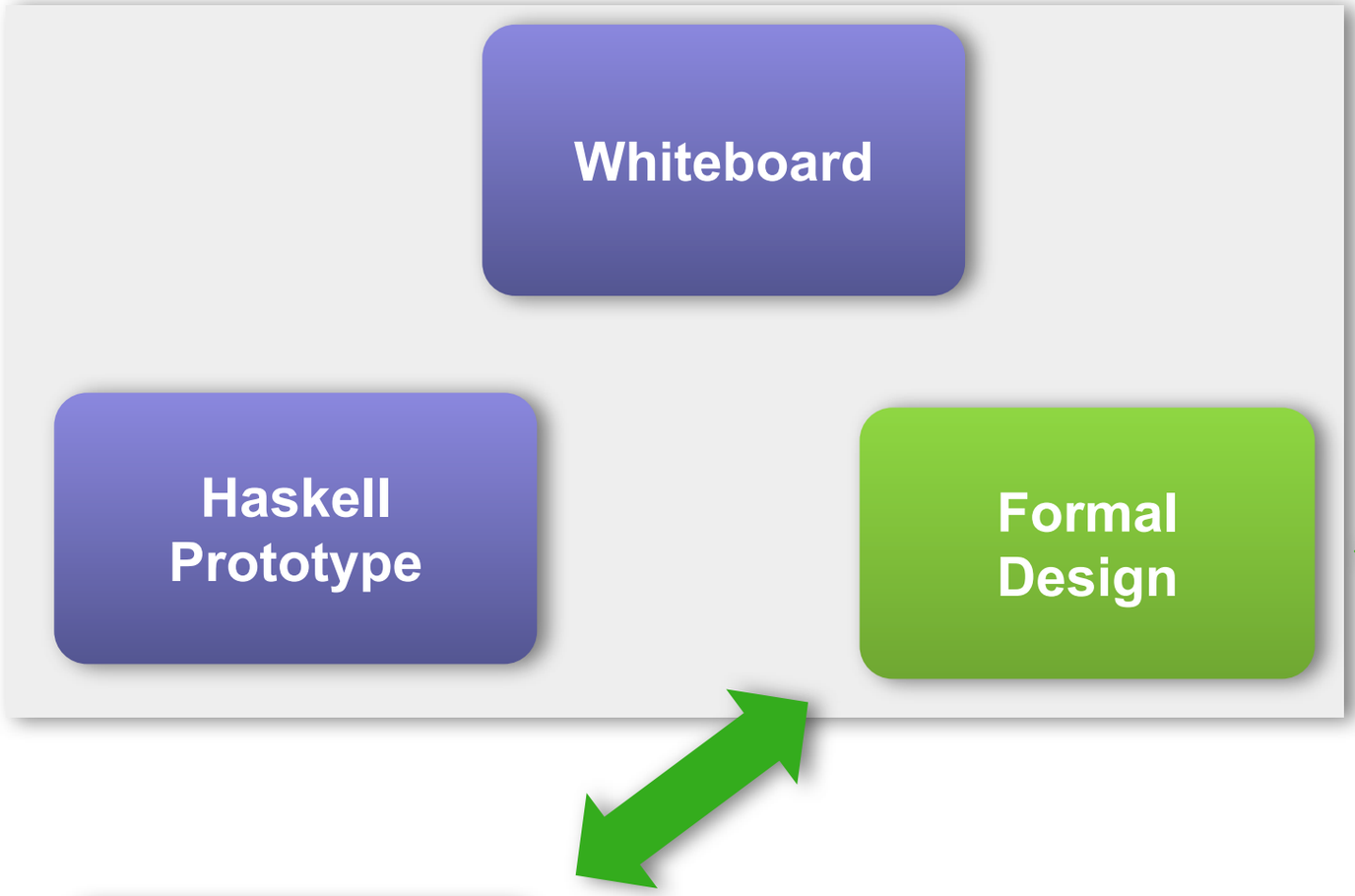
NICTA

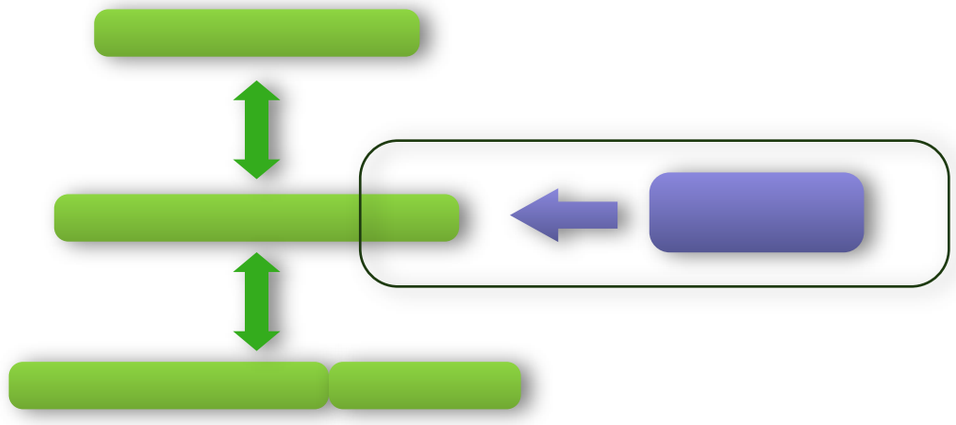




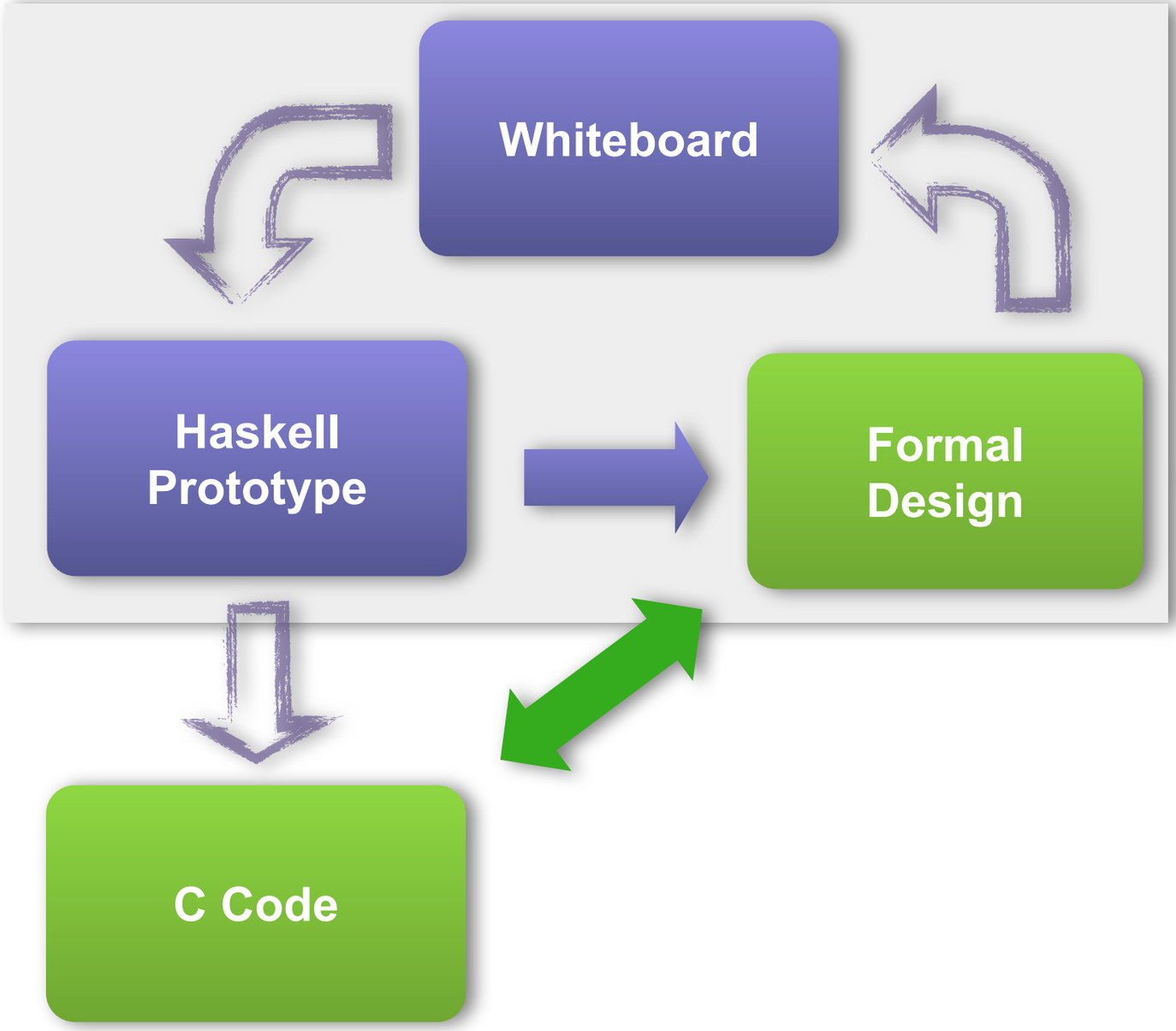
Iterative Design and Formalisation

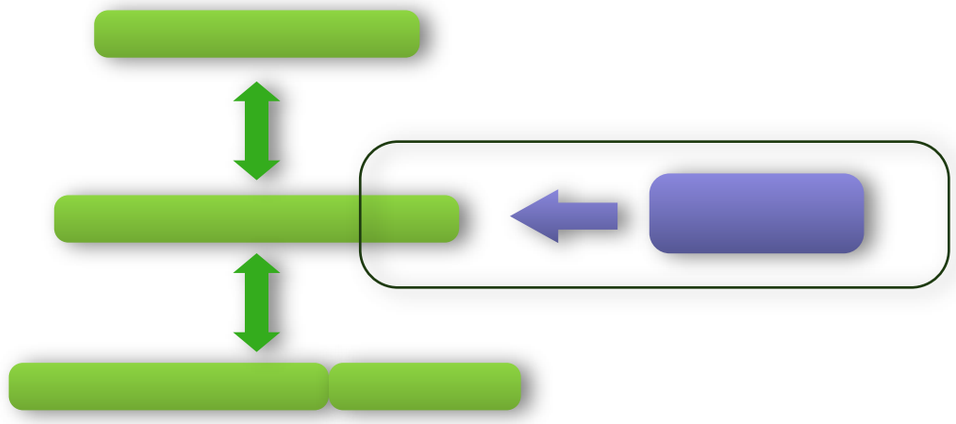
NICTA





Iterative Design and Formalisation





Design for Verification

NICTA

Reducing Complexity

Hardware

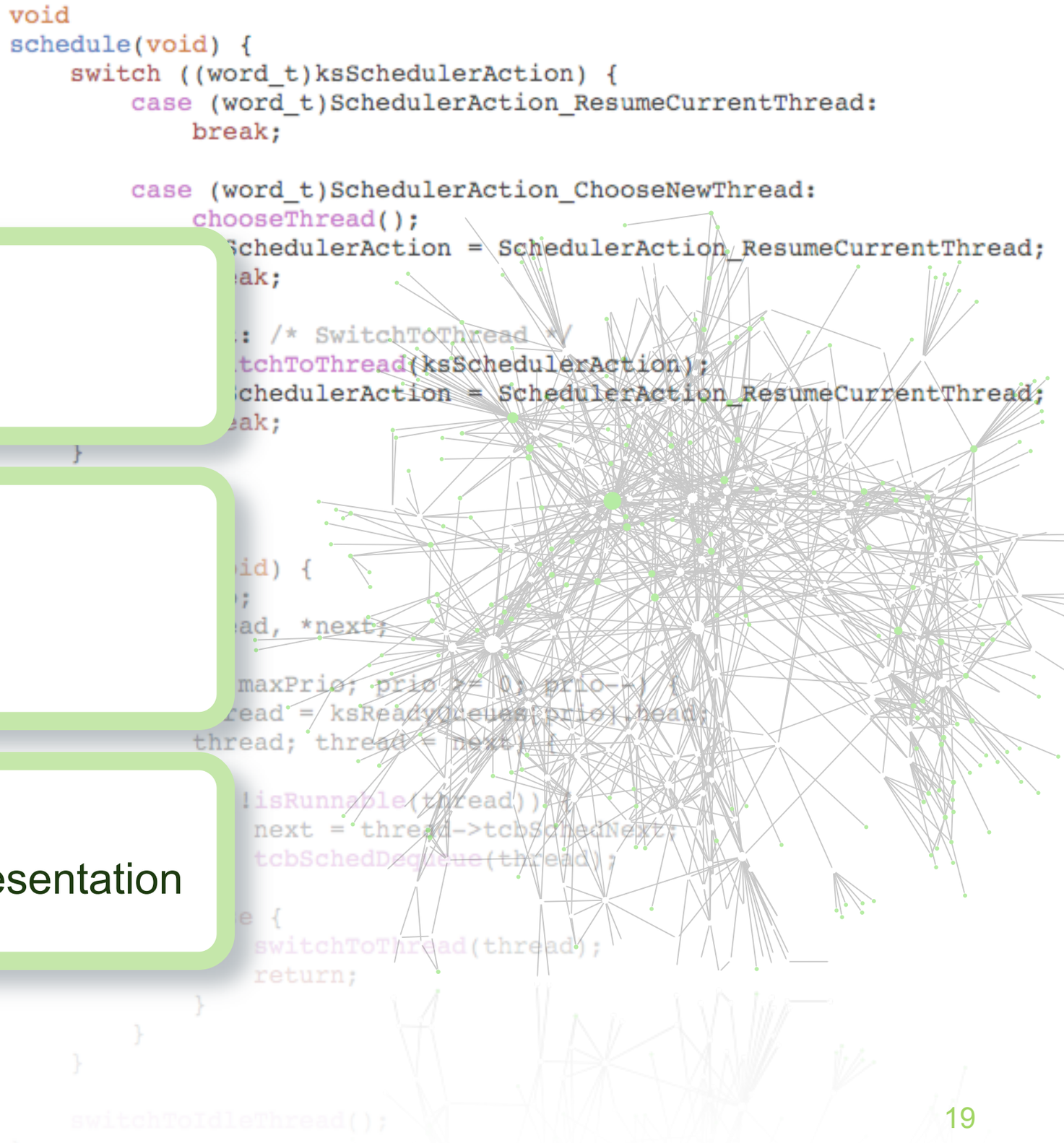
- drivers outside kernel

Concurrency

- event based kernel
- limit preemption

Code

- derive from functional representation



C subset

NICTA

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

Did you find any Bugs?

NICTA

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

Cost

Common Criteria EAL6:	**$87M**
L4.verified:	**$6M**

Did you find any Bugs?

NICTA

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

Cost

Common Criteria EAL6:	**$87M**
L4.verified:	**$6M**

Summary

NICTA

Formal proof all the way from spec to C.

- **200kloc** handwritten, machine-checked proof
- **~460** bugs (160 in C)
- Verification on **code**, **design**, and **spec**
- **Hard in the proof** ➡ **Hard in the implementation**

Formal Code Verification up to 10kloc:

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

(It's fun, too.
And we're hiring..)

Thank You

Thank You