Formally Verifying a Micro-Kernel: Experiences from the seL4 Project

Burkhart Wolff

http://www.lri.fr/ ~wolff

Université Paris-Saclay

Abstract

I will give (as a close follower, Phd Examiner and member of a rival project) an overview of the "seL4 project" done by NICTA, Australia. The aim of the project was the development and comprehensive machine-checked formal verification of an general-purpose operating system microkernel.

The talk will cover development methodology, the kernel design used to make the verification tractable and the relevant refinement steps under-taken which link an abstract, powerful access-control-oriented security model down to an implementation model, which is again linked to assembly code running on COTS ARM processors.

A particularity of the project is that a variety of experimental data isavailable over the development costs of their approach, including modeling, coding, code-verification and model-maintenance over meanwhile a decade.

My talk will essentially follow the article which appeared in "ACM Trans. Comp. Sys, Vol 32, No 1, 2014, (same title) and contrast it by own experiences gained in the EUROMILS Hypervisor project.

Overview

- Context: Security-Critical Systems
- What is seL4 ?
- Abstract Model:
  Concepts and Functionality
- Verification Methodology
- Experimental Evaluation of the Development Process
- seL4 is free - what does this mean ?

GENERAL DYNAMICS
C4 Systems

NICTA

seL4 for Dependable Systems Software

Developing dependable systems requires built-in security and safety at all levels of the system, including in the lowest-level system software: the operating system and device access software.

For truly dependable systems, the software must be trustworthy: we must be able to provide the guarantee that it behaves correctly and has the required security and safety properties. These guarantees can be provided through testing, certification, and formal verification.

GENERAL DYNAMICS
C4 Systems

NICTA

seL4 for Dependable Systems Software

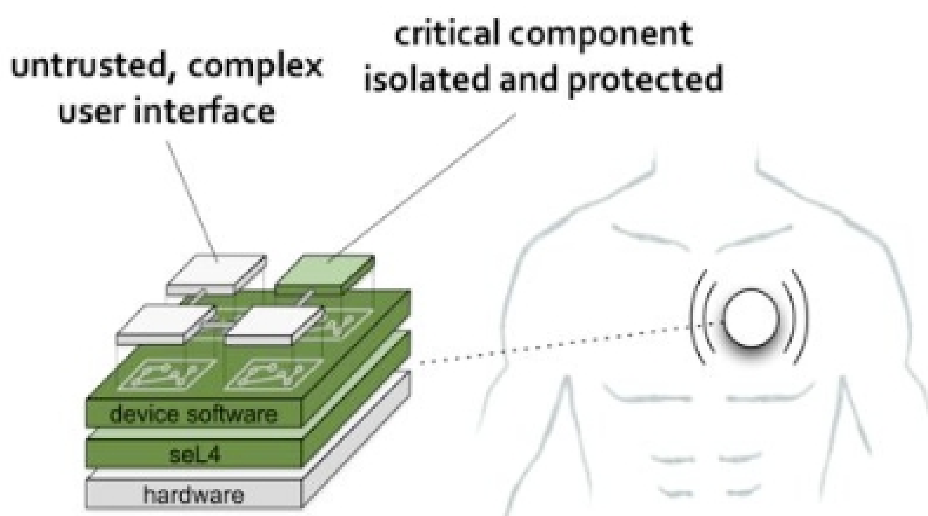
Developing dependable systems requires built-in security and safety at all levels of the system, including in the lowest-level system software: the operating system and device access software.

For truly dependable systems, the software must be trustworthy: we must be able to provide the guarantee that it behaves correctly and has the required security and safety properties. These guarantees can be provided through testing, certification, and formal verification.

Microkernels

- ... are a critical component in Embedded Systems
- ... sitting directly on (often relatively simple) hardware (ARM)
- ... enforce separation of critical and non-critical components

- they are complex, concurrent, but fairly small
- ... and an ideal target for verification by formal proof



Microkernels

- Applications:
  - Critical Embedded Systems
    (Medical, Railways, Avionics, ...)
  - Critical Common Infrastructure
    (secure network switches,
    security co-processors of iOS devices ...)
  - Systems with particularly high demands on
    integrity and confidentiality (military, “Merkel-phone”)

What is L4 ?

- Core Microkernel Design Principle:

Minimality

A concept is tolerated inside the microkernel only if moving outside the kernel, i.e. permitting competing implementations, would prevent the systems required functionality [Liedtke, SOSP '95]

What is L4 ?

- Core Microkernel Design Principle:

Minimality

⇒ atomic actions
(locked code region, usually system mode)
⇒ system API calls contain atomic actions
⇒ file-systems, IP-Stacks, drivers are in user-land.

What is L4 ?

- Mikro-Kernel Design was quite popular in the 80-ies (MACH, OS2)

⇒ atomic actions

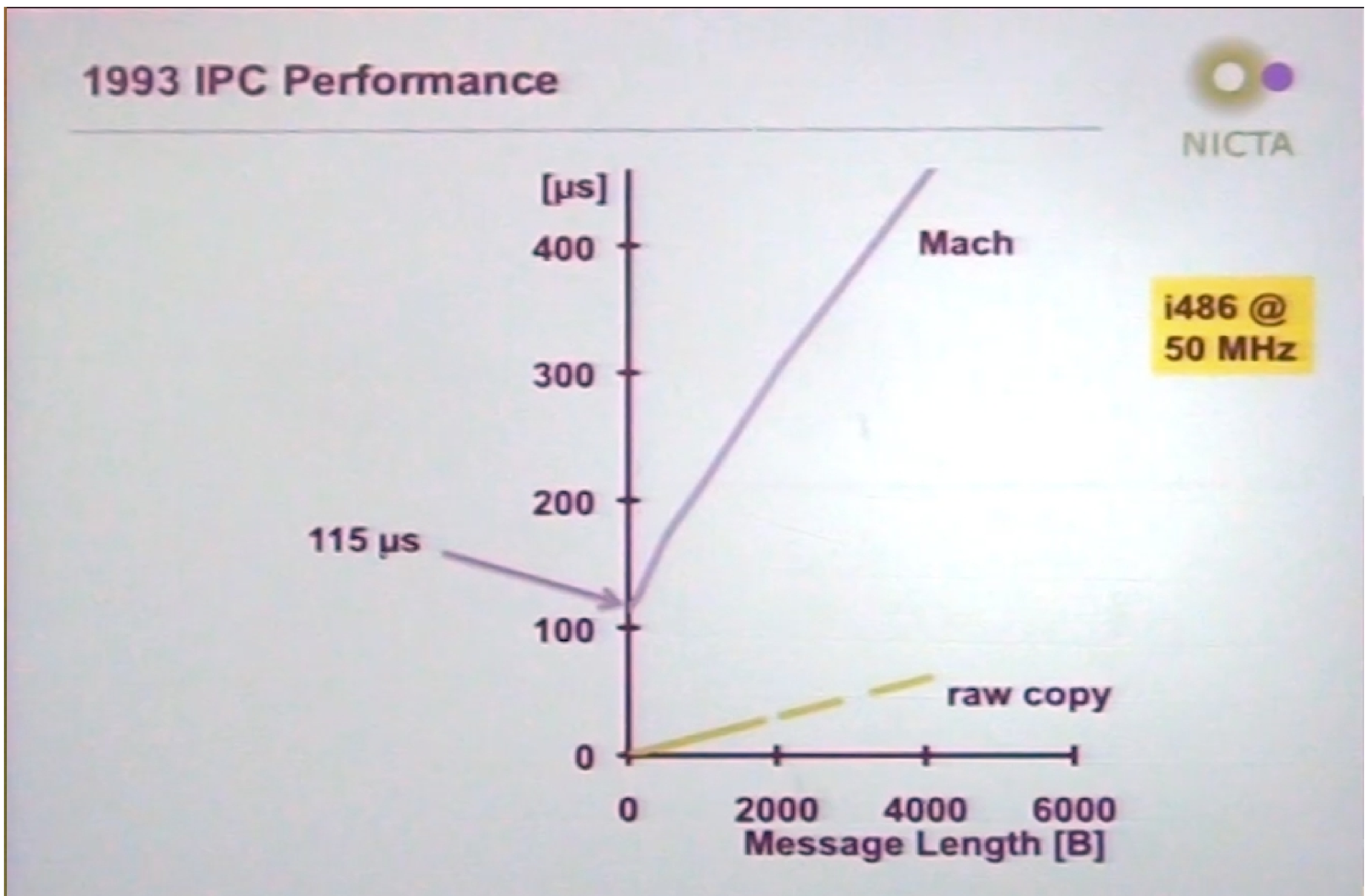
(locked code region, usually system mode)

⇒ system API calls contain atomic actions

⇒ file-systems, IP-Stacks, drivers are in user-land.

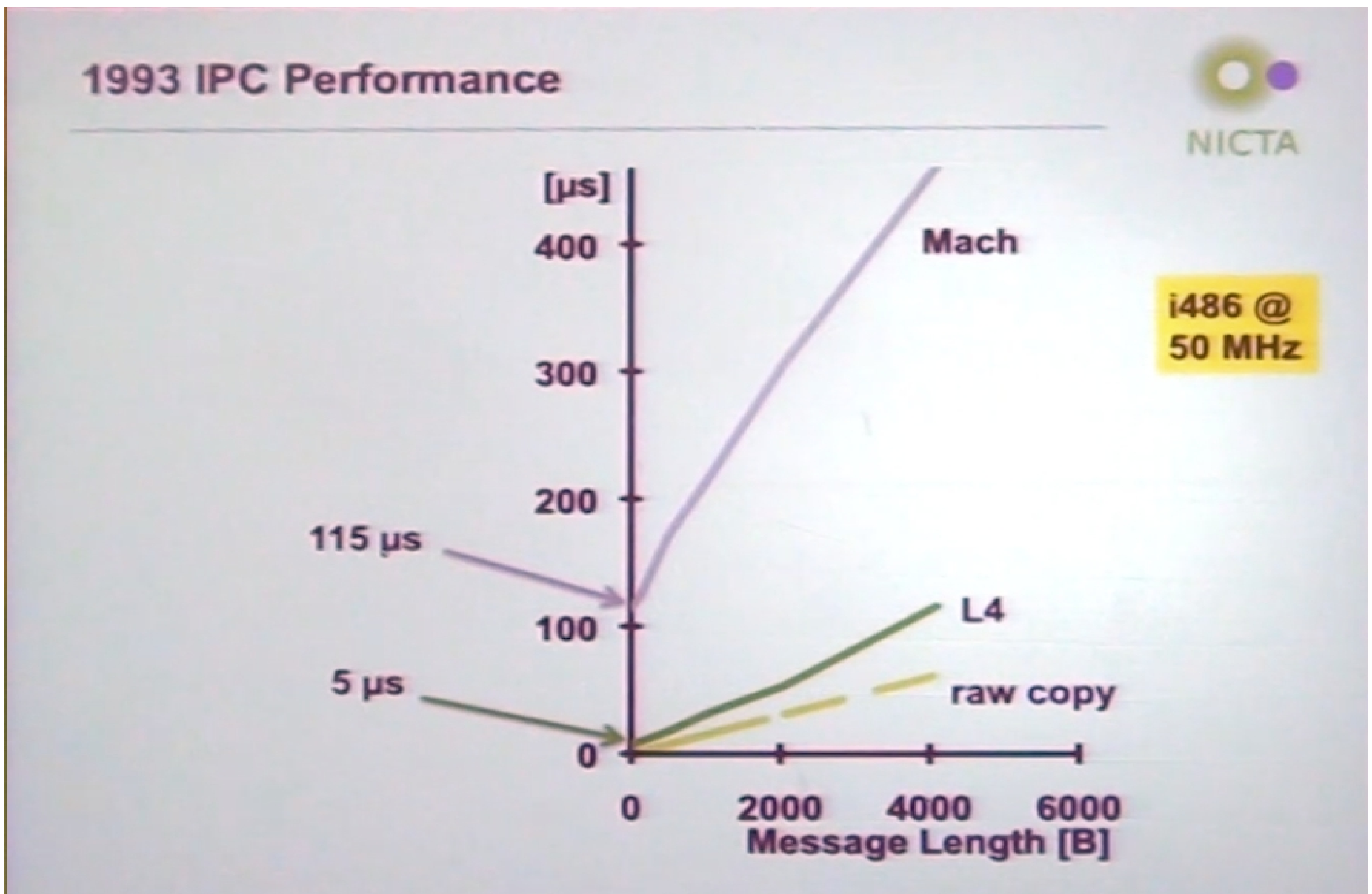
What is L4 ?

- ... but ran into the "100-micro-seconds desaster" for Inter-Process Com. (IPC)



What is L4 ?

- ... for which Liedke proposed a solution based on shared physical-memory.



What is L4 ?

- ... for which Liedtke proposed a solution based on shared physical-memory.

IPC Performance over 20 Years

NICTA

Name	Year	Processor	MHz	Cycles	μs
Original	1993	i486	50	250	5.00
Original	1997	Pentium	160	121	0.75
L4/MIPS	1997	R4700	100	86	0.86
L4/Alpha	1997	21064	433	45	0.10
Hazelnut	2002	Pentium 4	1,400	2,000	1.38
Pistachio	2005	Itanium	1,500	36	0.02
OKL4	2007	XScale 255	400	151	0.64
NOVA	2010	i7 Bloomfield (32-bit)	2,660	288	0.11
seL4	2013	i7 Haswell (32-bit)	3,400	301	0.09
seL4	2013	ARM11	532	188	0.35
seL4	2013	Cortex A9	1,000	316	0.32

 4 SOSP'13

What is seL4 ?

- seL4: secured L4
  (initiated by Gernot Heiser & Gerwin Klein)

  - OS Kernel in the L4 tradition
  - advanced Security (Access-Control) Model
    "Take-Grant Capabilities"
  - virtual memory, dyn. thread creation,
    IPC, Fast-Track-IPC, support of AnoCom.
  - designed to be formally verifiable (in Isabelle/HOL)
  - designed to be performant

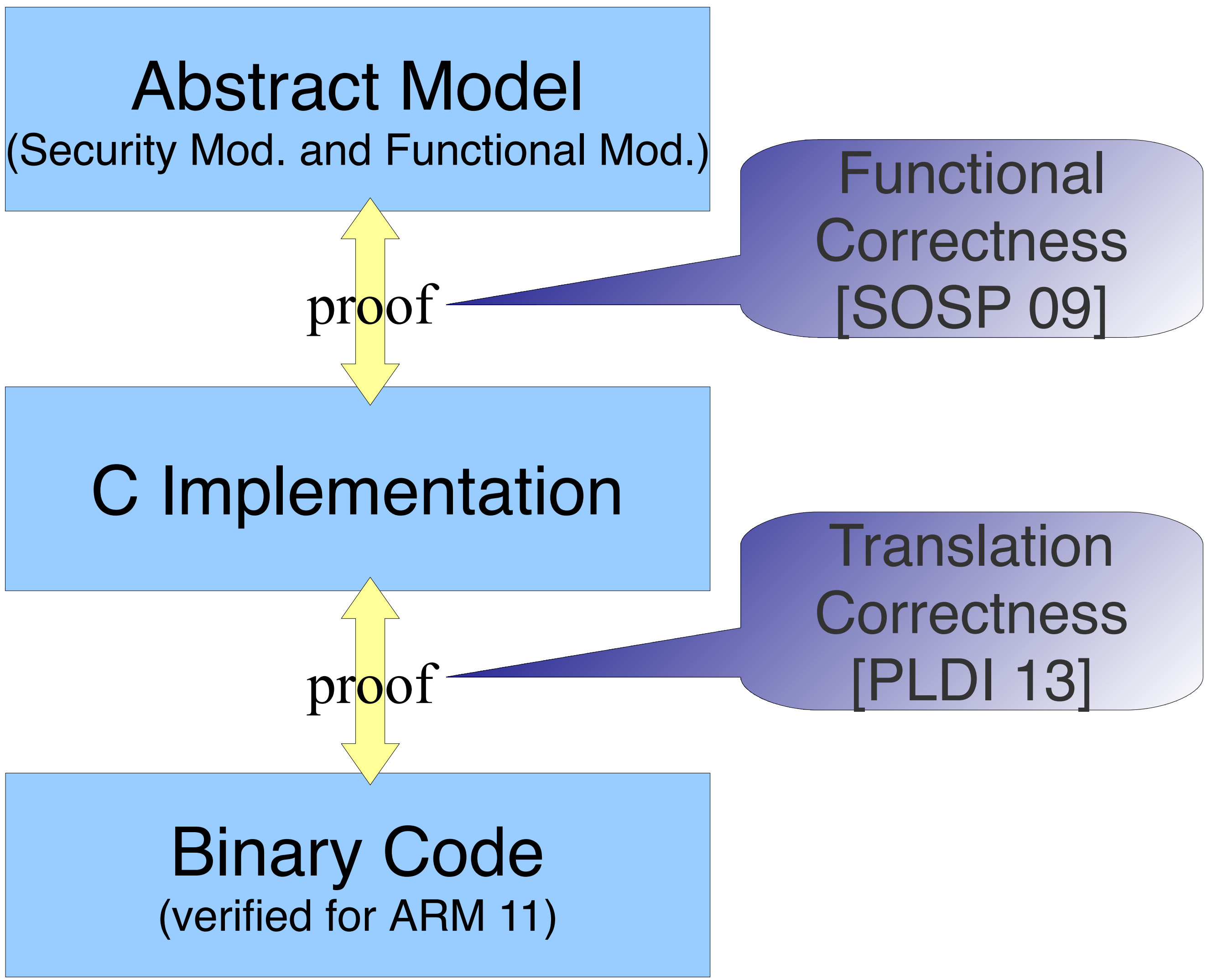
Models and Methodology

Abstract Model
(Security Mod. and Functional Mod.)

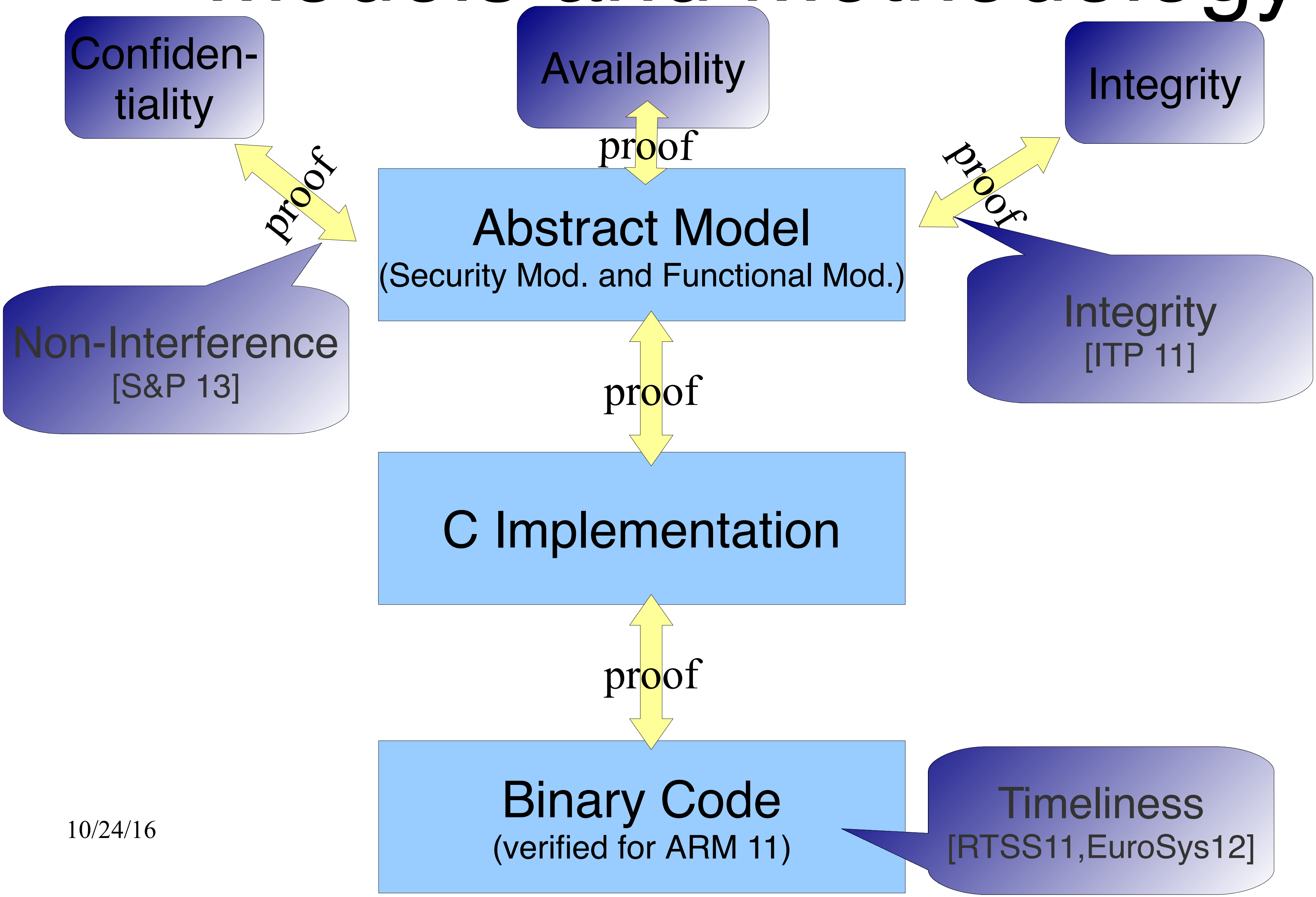
C Implementation

Binary Code
(verified for ARM 11)

Models and Methodology



Models and Methodology



Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):
  - untyped memory and (kernel-) object management
  - capability management
  - virtual address space management
  - thread management
  - inter-process communication (IPC)
  - device I/O management

Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):
  - untyped memory and (kernel-) object management
  - capability management

capability objects belong to a thread or a thread-pool representing permissions for executing kernel operations on them. Can refer to other capabilities in a dag.

Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):

  - untyped memory and (kernel-) object management
  - capability management
  - virtual address space management

The VSpace belongs to a thread and is composed of objects controlling the virtualization of virtual memory.
This is architecture-specific.
IA86 : Page-Directories, Page-Tables, Frames...

Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):
  - untyped memory and (kernel-) object management
  - capability management
  - virtual address space management

The VSpace belongs to a thread and is
composed of objects controlling the virtualization
of virtual memory.
This is architecture-specific.
IA86 : Page-Directories, Page-Tables, Frames...

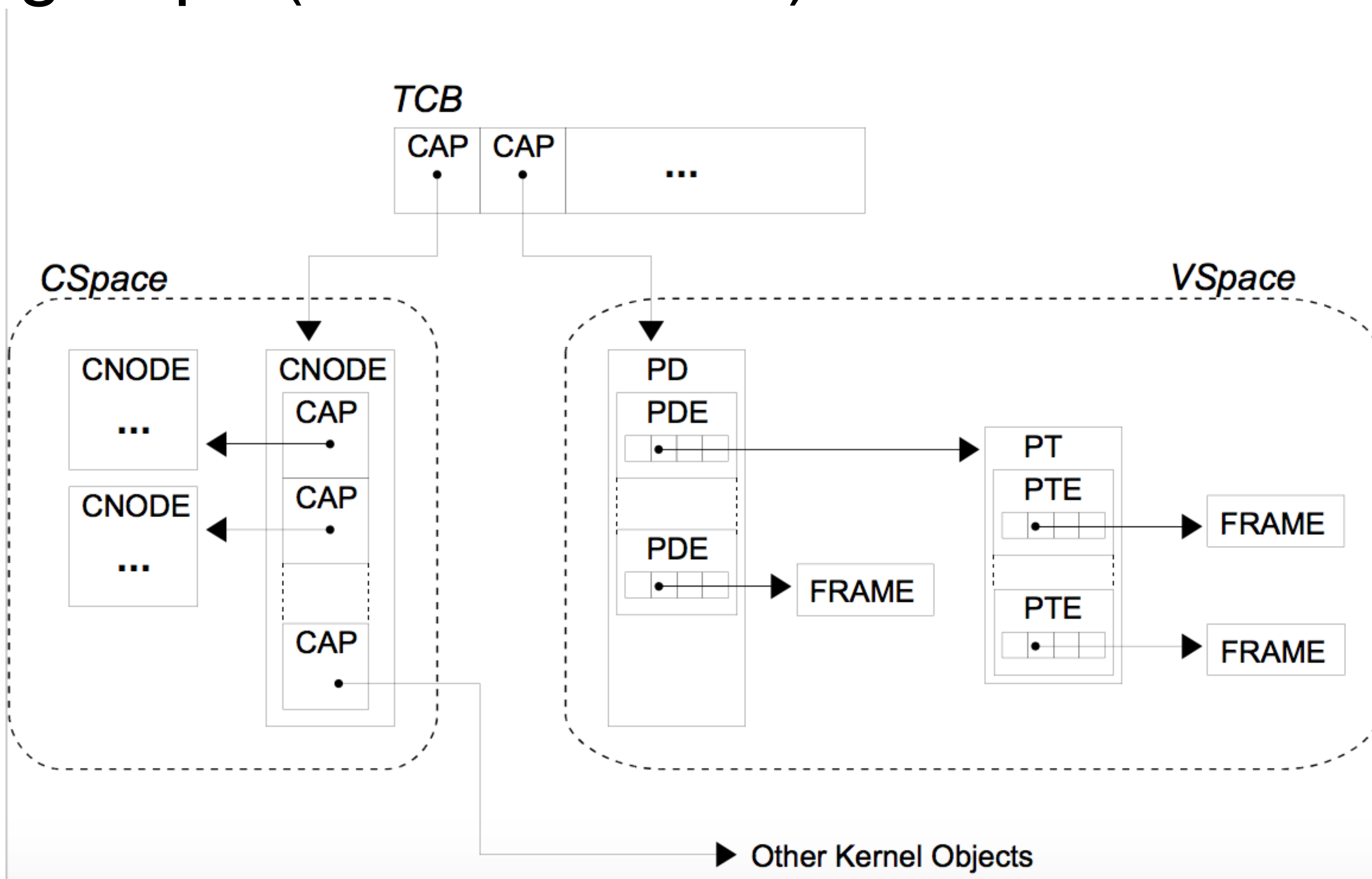
Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):
  - thread management

    represented by a Thread Control Block (TCB object) with VSpace and CSpace (capabilities)

Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):

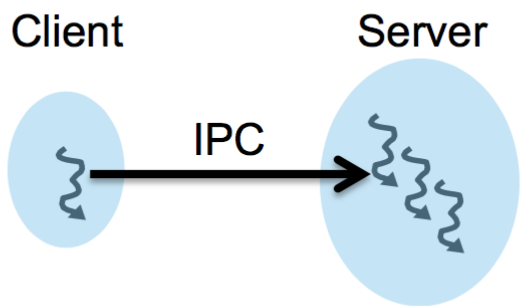


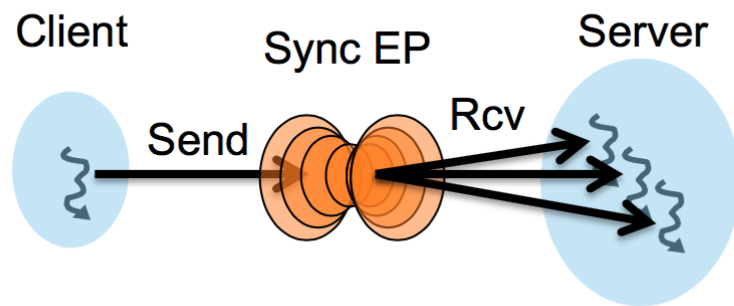
Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):
  - inter-process-communication (IPC)

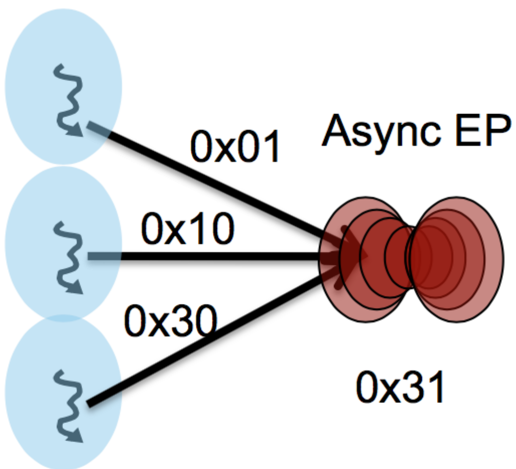
    based on „Endpoints“ (kind of mail-boxes)
    IPC_send and IPC_receive refer to
    synchronous (SEP) and asynchronous endpoints (AEP)

Endpoints



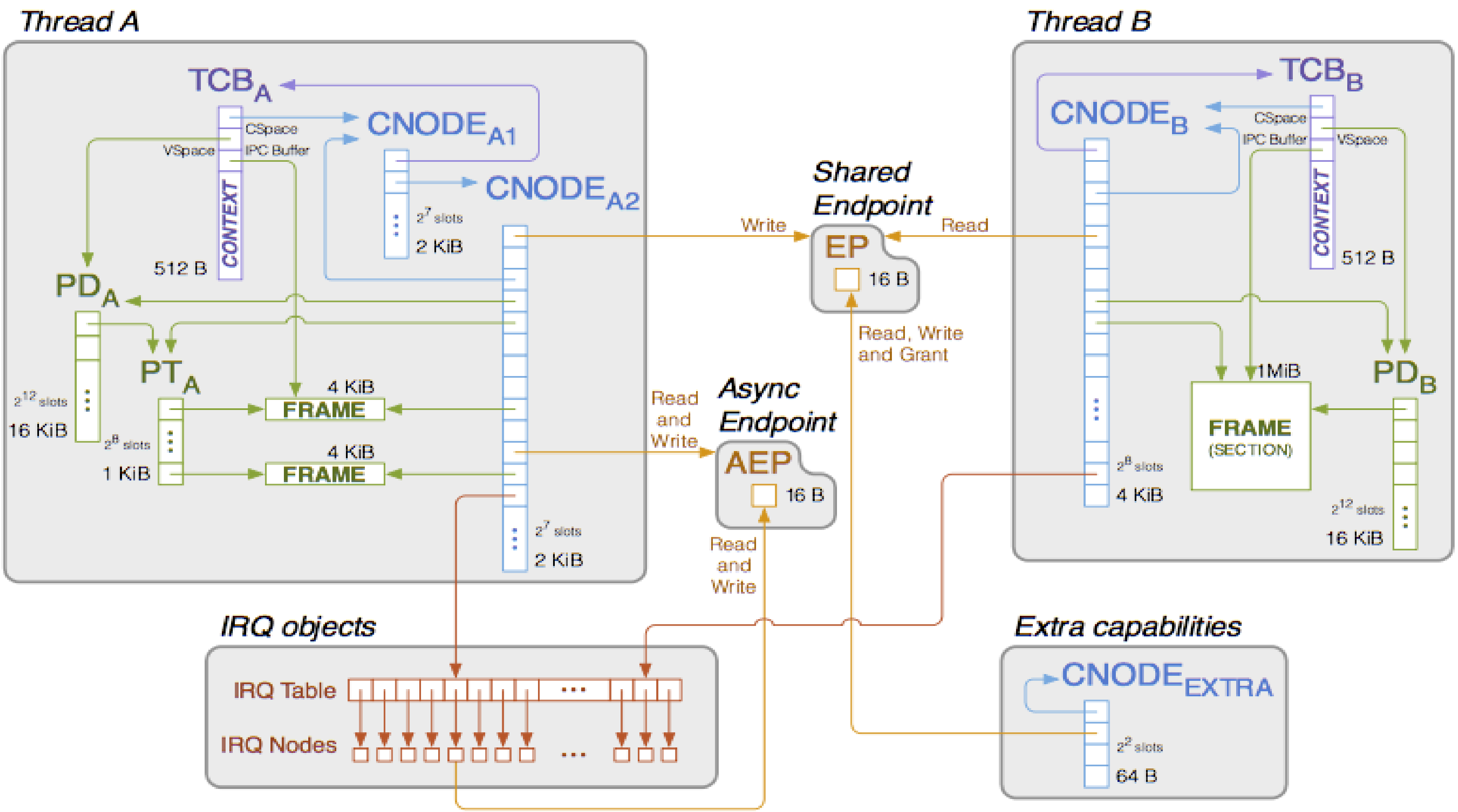


- Sync EP queues senders/receivers
- Does not buffer messages



- Async EP accumulates bits

IPC - Send - Receive Scenarios



Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):

  - device I/O management

    Device drivers run outside the kernel.
    To support this, seL4 implements I/O specific objects that provide access to I/O ports, interrupts, and I/O address spaces for DMA-based memory spaces.

Abstract Model: Concepts and Functionalities

- seL4 kernel operations can be devided into 6 groups (see Ref. Man.):

  - untyped memory and (kernel-) object management
  - inter-process-communication (IPC)

    based on „Endpoints“ (kind of mail-boxes)
    IPC_send and IPC_receive refer to
    synchronous (SEP) and asynchronous endpoints (AEP)

Security Model (SM)

The seL4-security model (SM) is based on access control (AC) kernel objects. Key features:

- take-grant protection model
  (Jones et al al 76, Snyder 77, Bishop and Snyder 79)

  can entity x ever gain access to entity y ?

- addition of shared capability storage
- „active and passive entities" (vulgo: subjects and objects)
- entity destruction and identifier reuse
- perm hierarchies avoided by delegatable AC model

Security Model (SM)

- Take

Take An entity e_x with a capability with a Take access right to another entity e_y can take a copy of one of that entity's capabilities α[3], as illustrated in Figure 4.2.

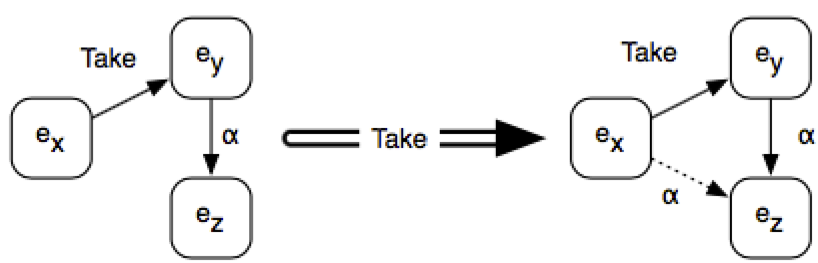


Figure 4.2: The take operation

Security Model (SM)

- Grant

Grant An entity e_x with a capability with a Grant access right to another entity e_y is able to grant a copy of one of its capabilities α to that entity, as illustrated in Figure 4.3.

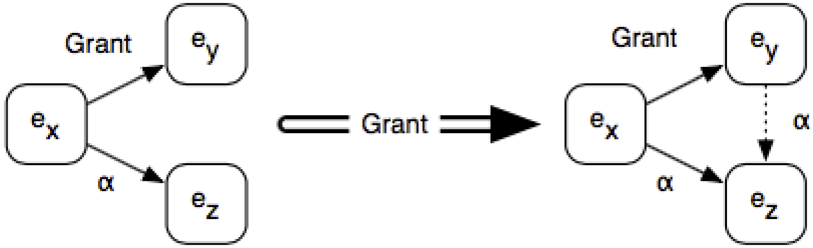


Figure 4.3: The grant operation

Security Model (SM)

- Create

Create Any entity e_x can create a new entity e_n, to which it has full access rights, as illustrated in Figure 4.4.

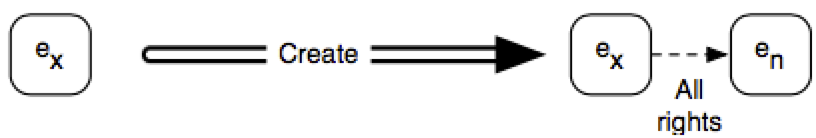


Figure 4.4: The create operation

Security Model (SM)

- Remove (with id-recup.)

Remove An entity e_x can remove one of its capabilities α, as illustrated in Figure 4.5.

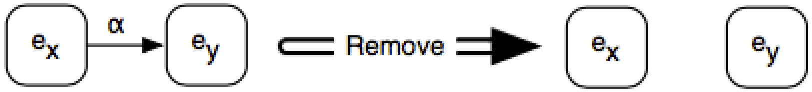


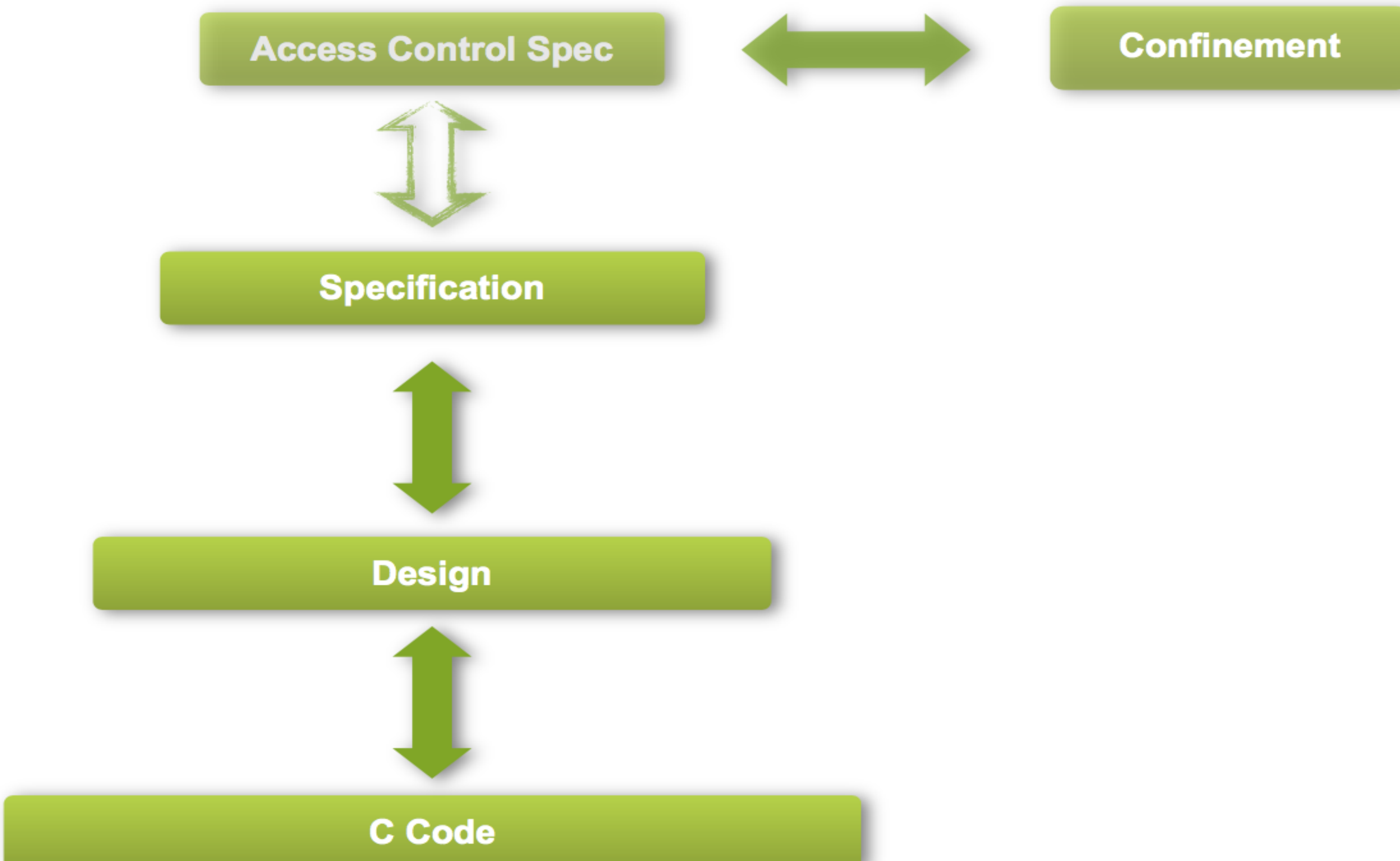
Figure 4.5: The remove operation

Verification Methodology

- Modeling in Isabelle/HOL and in Haskell

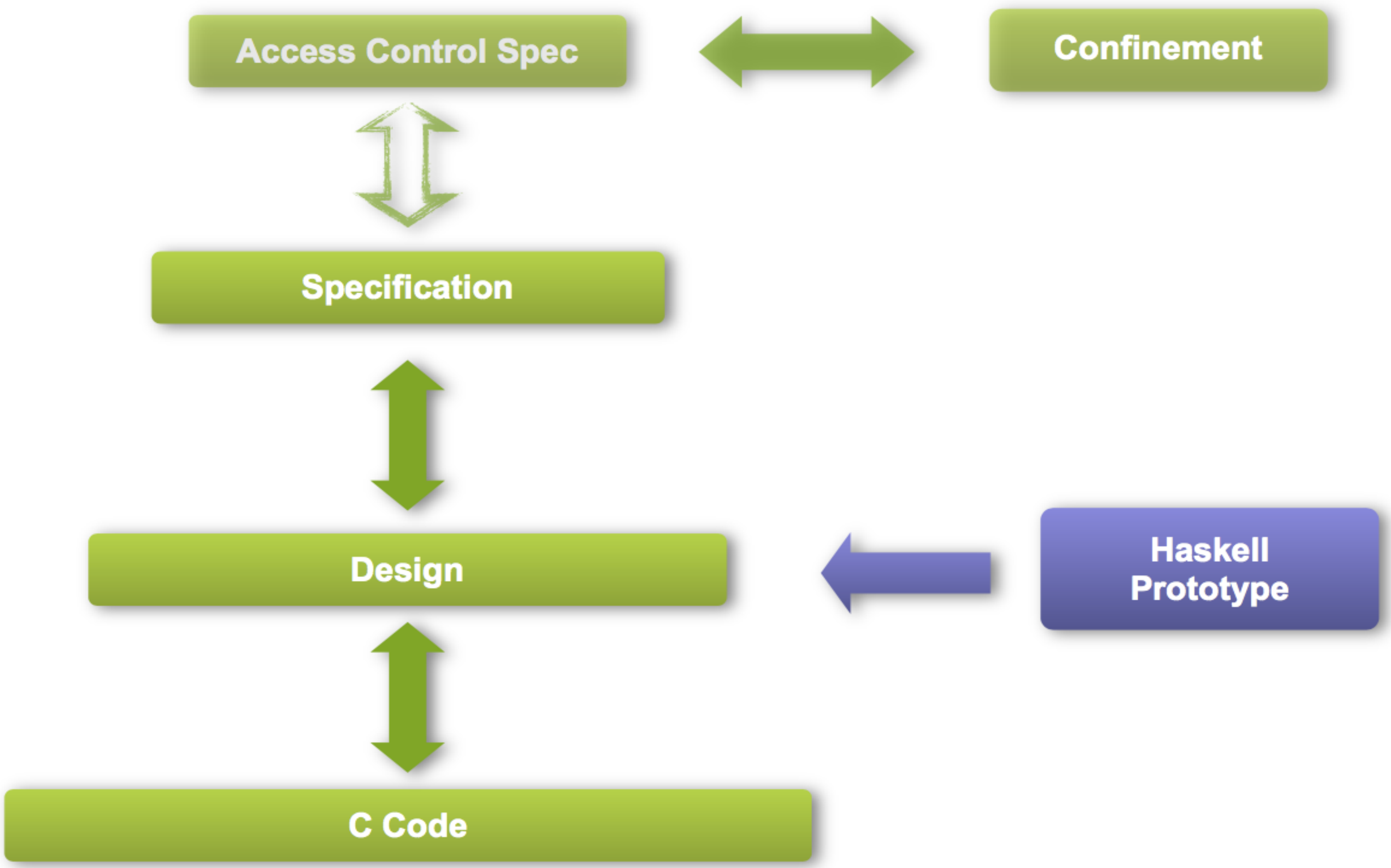
Verification Methodology

- Modeling in Isabelle/HOL and in Haskell



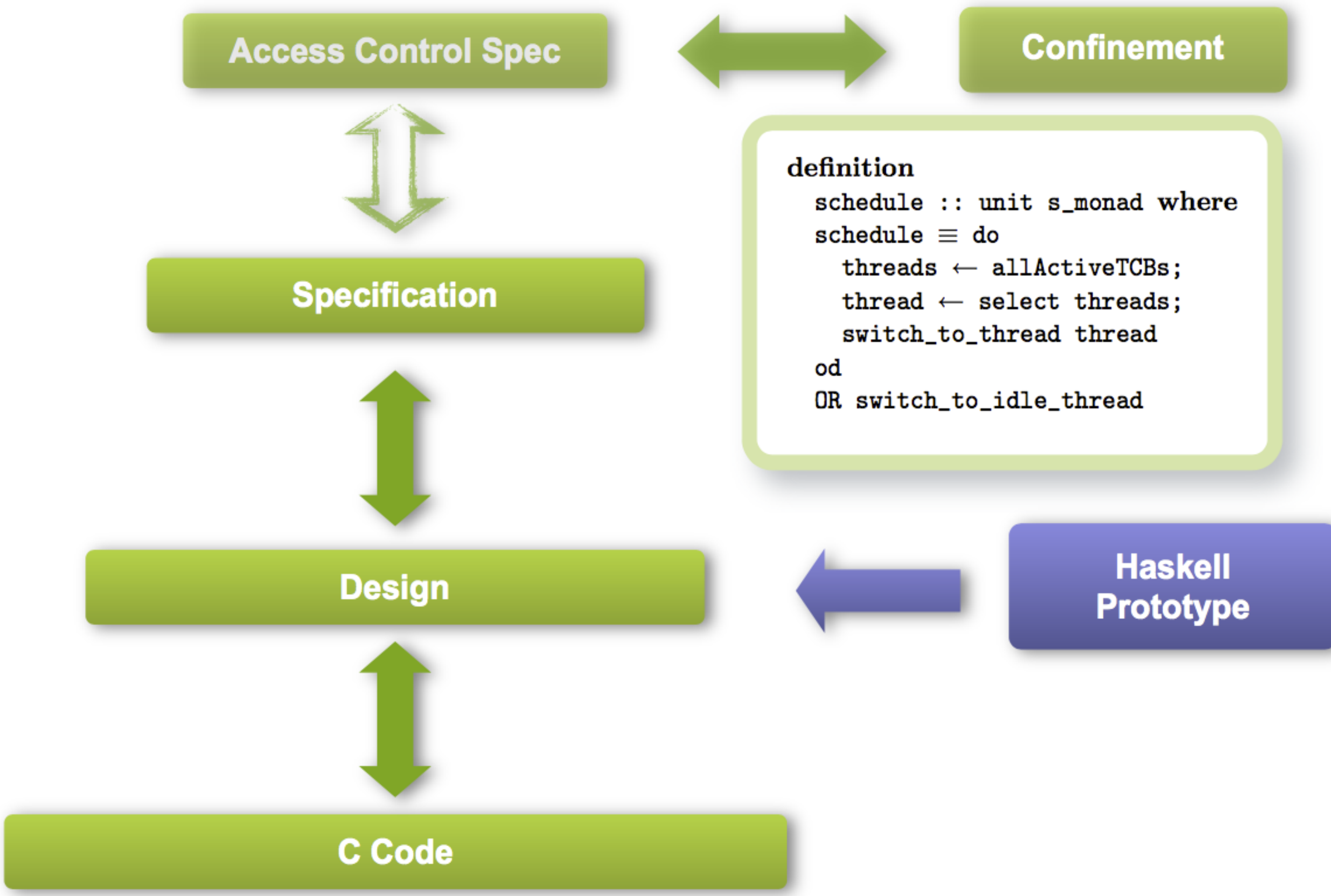
Verification Methodology

- Modeling in Isabelle/HOL and in Haskell



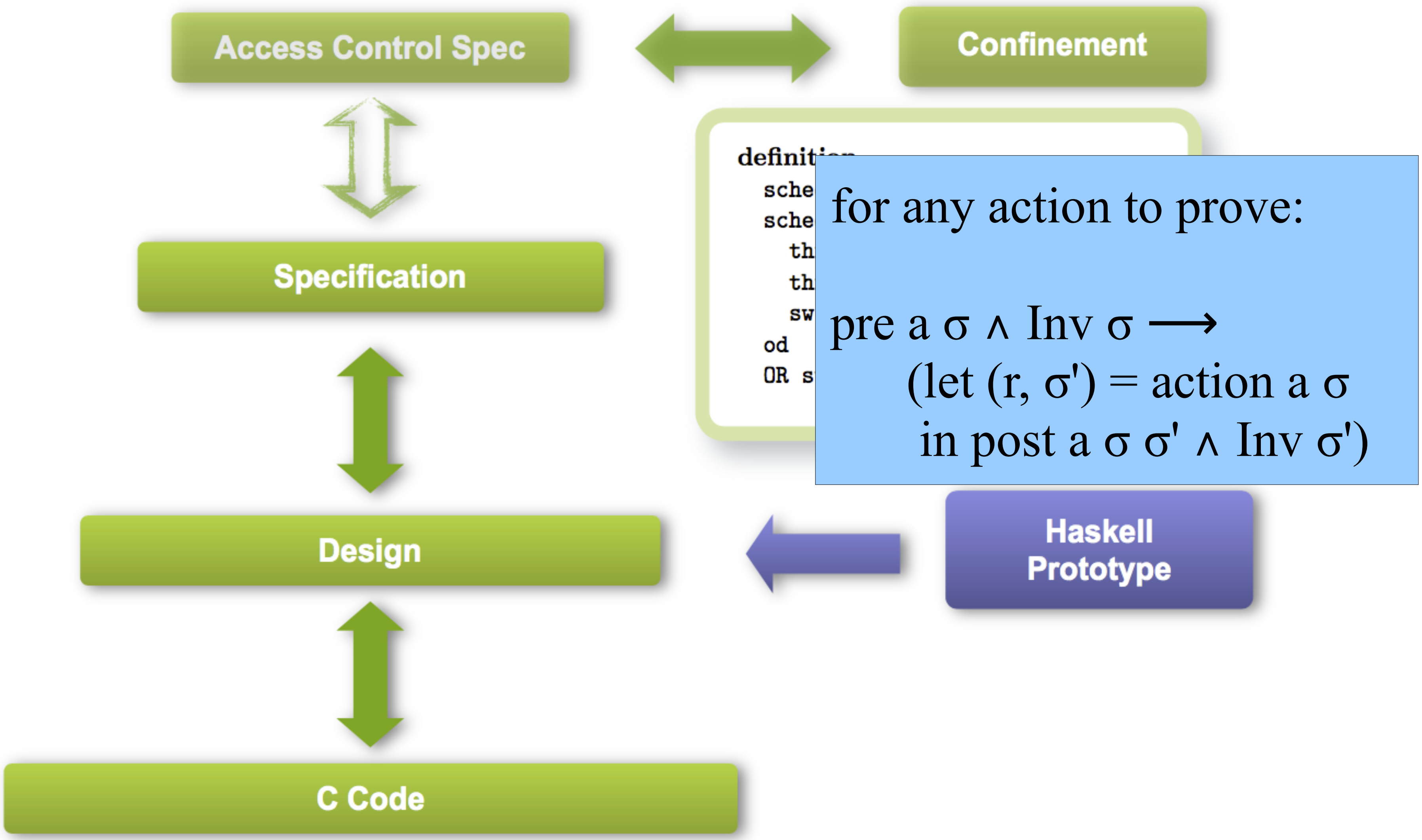
Verification Methodology

- Modeling in Isabelle/HOL and in Haskell



Verification Methodology

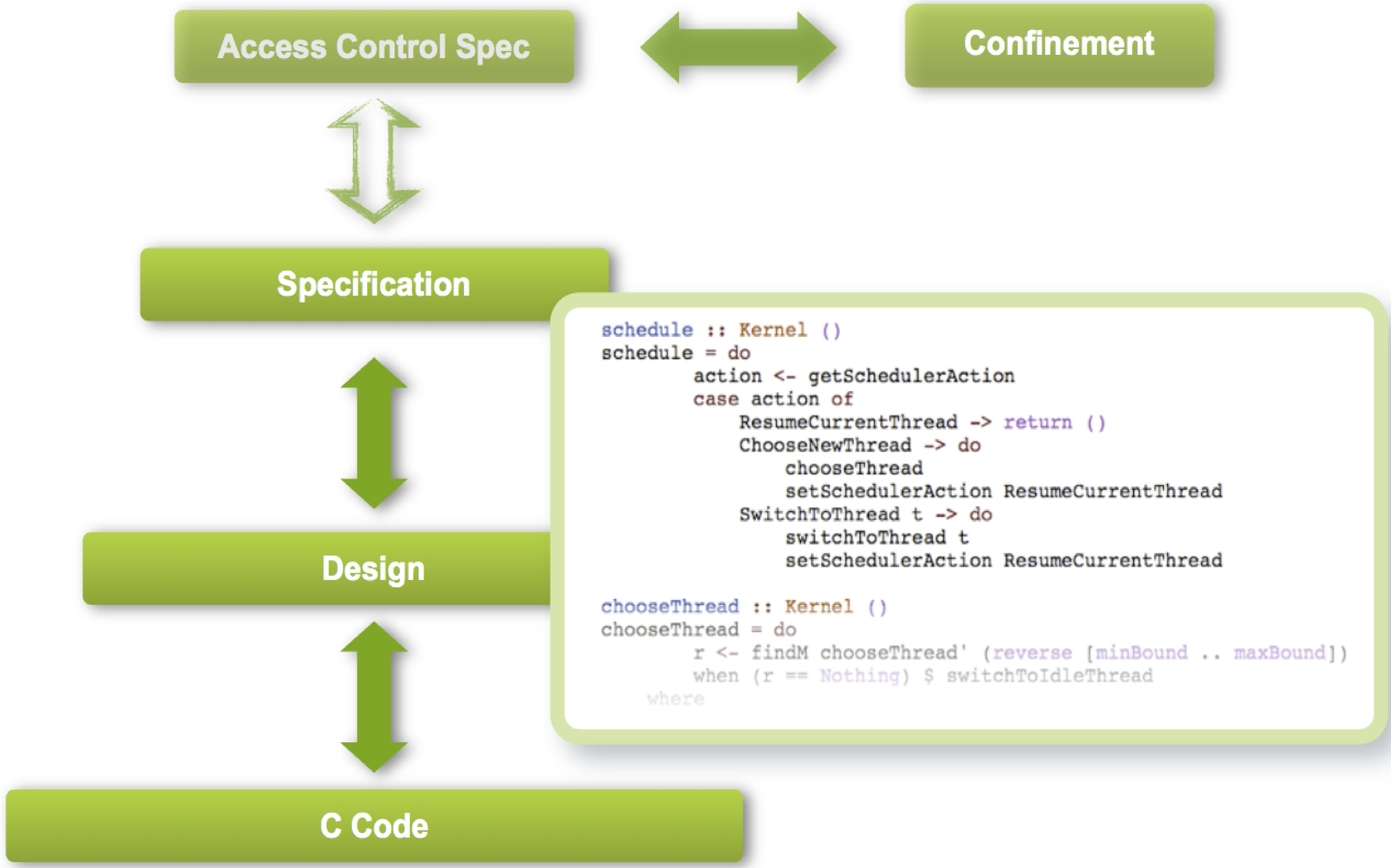
- Modeling in Isabelle/HOL and in Haskell



for any action to prove:

$$\text{pre } a\ \sigma \wedge \text{Inv } \sigma \longrightarrow (\text{let } (r, \sigma') = \text{action } a\ \sigma \text{ in post } a\ \sigma\ \sigma' \wedge \text{Inv } \sigma')$$

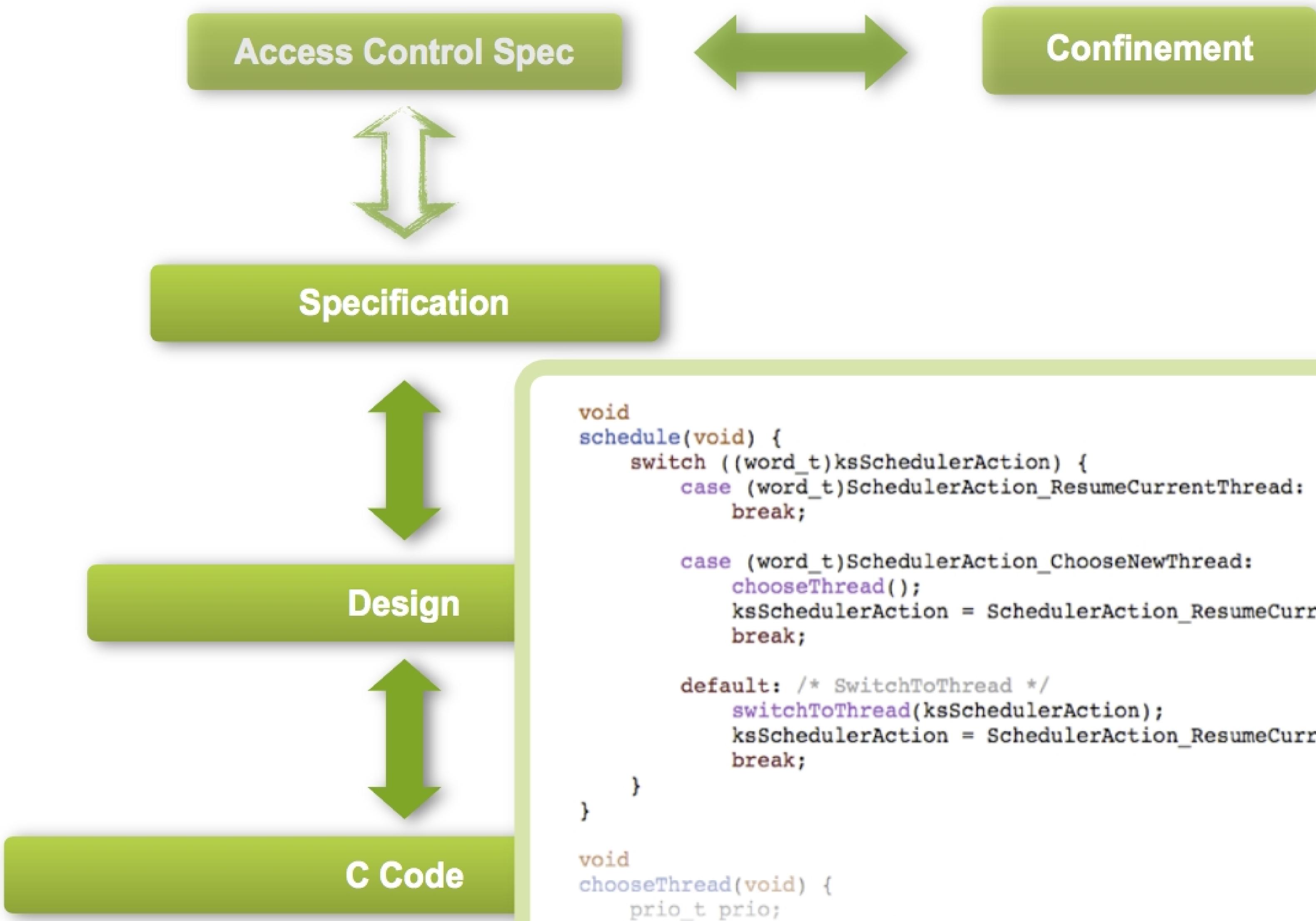
Verification Methodology




```
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
```

Verification Methodology




```
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;
```

Verification Methodology

- Prefinal step :
  Eliminate Haskell proto-type.
  - hand-written C-Code
  - compiled over
    C - 2 - Isabelle/HOL/Simp compiler
  - define memory abstractions
  - link to former invariant proofs ...
    [Trust depends on this compiler]

Verification Methodology

- Supported C this way:

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

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

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

```
if(!isRunnable(thread)) {
    next = thread->tcbSchedNext;
    tcbSchedDequeue(thread);

else {
    switchToThread(thread);
    return;
```

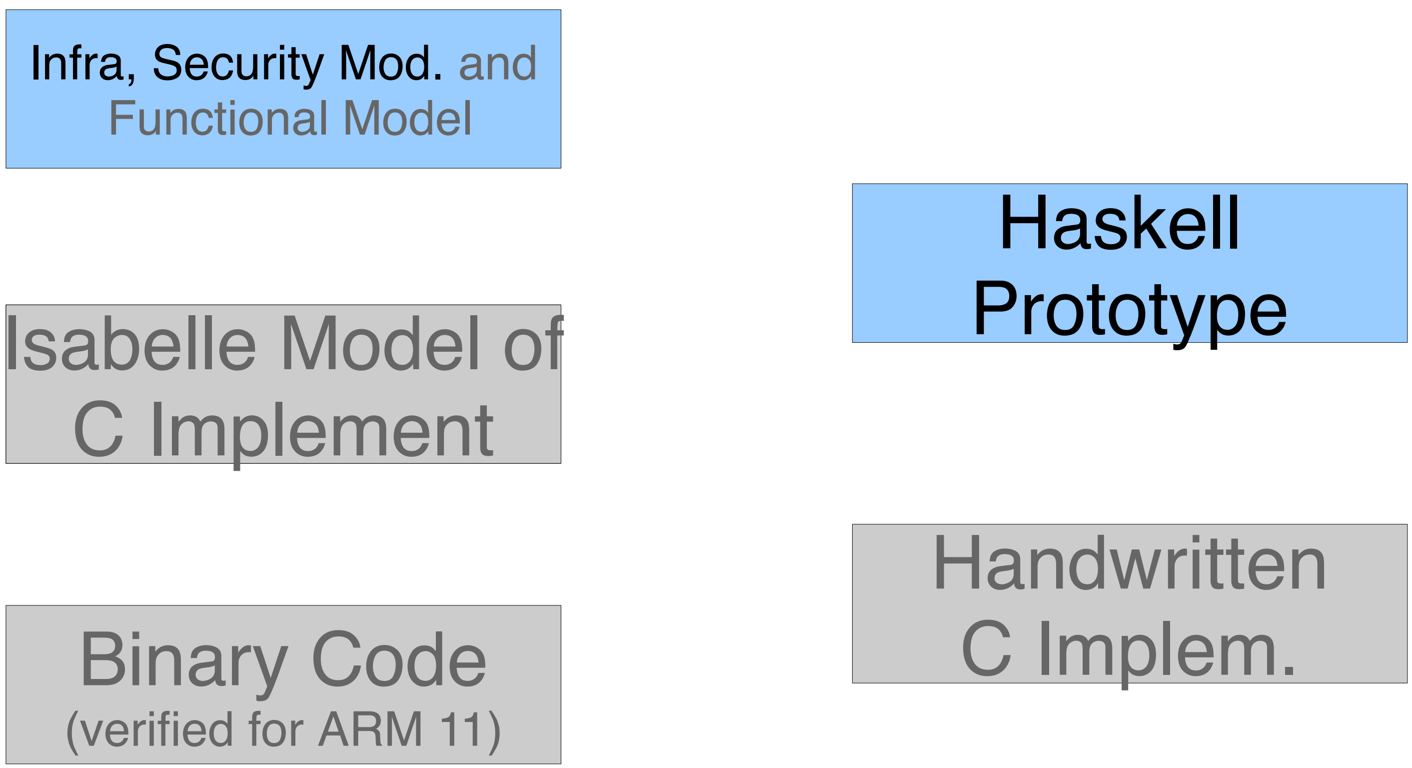
Verification Methodology

- Final step :
  Eliminate C - 2 - Isabelle/HOL/Simpl

  - generated optimized ARM assembly
    (conventionally via gcc -o4 ...)
  - re-use an ARM operational semantics
    model(going back to A. Fox)
  - use smt technology to verify that
    action contracts are still valid on machine level ...

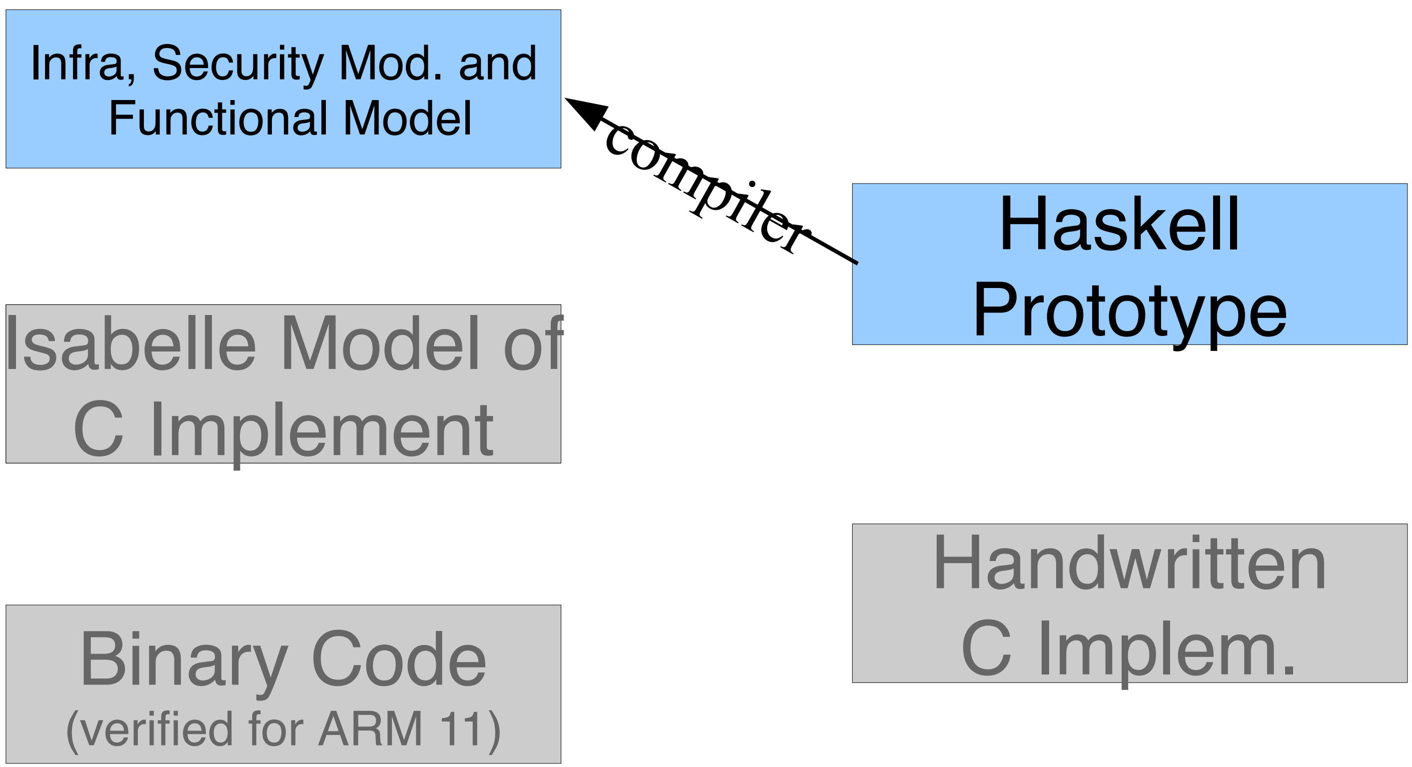
Evaluation

- A Revision of the Development Process



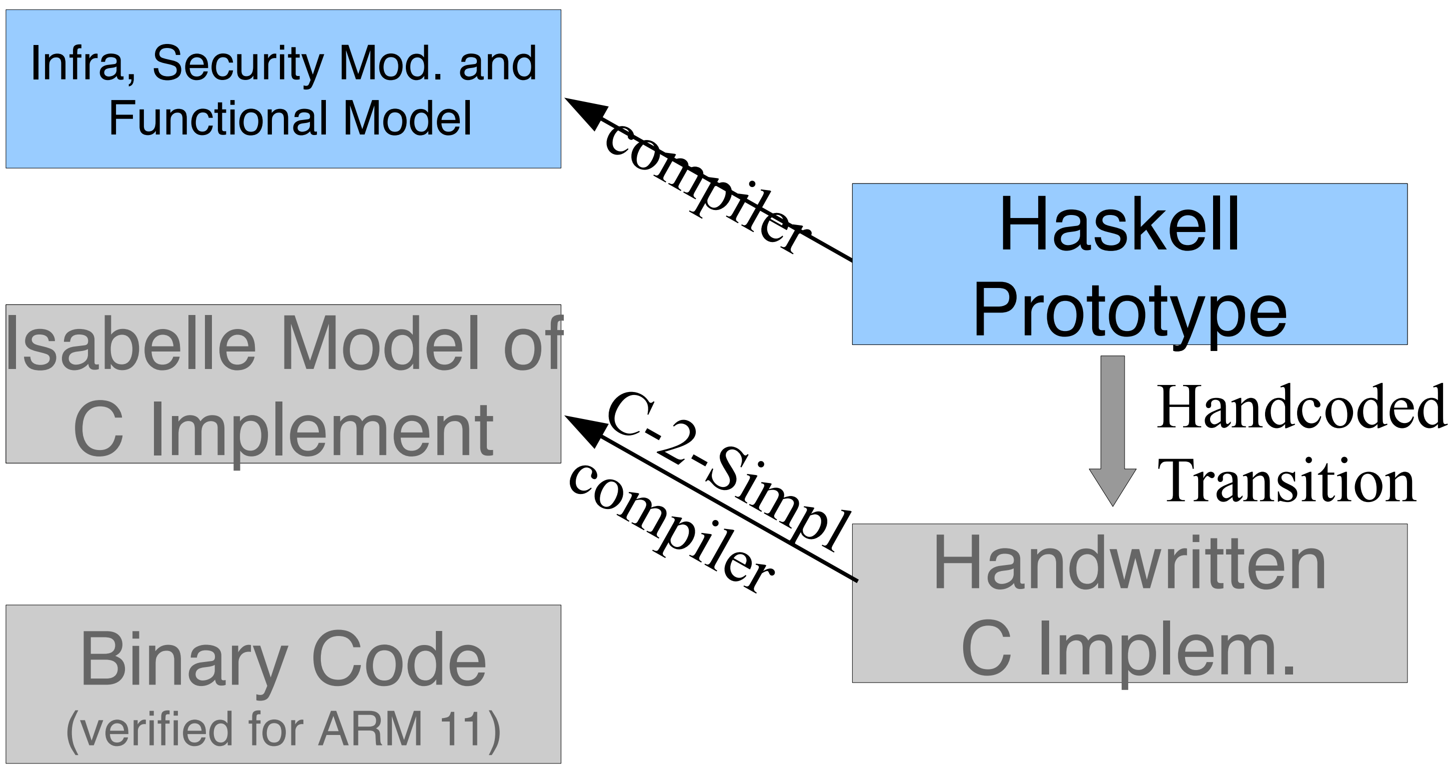
Evaluation

- A Revision of the Development Process



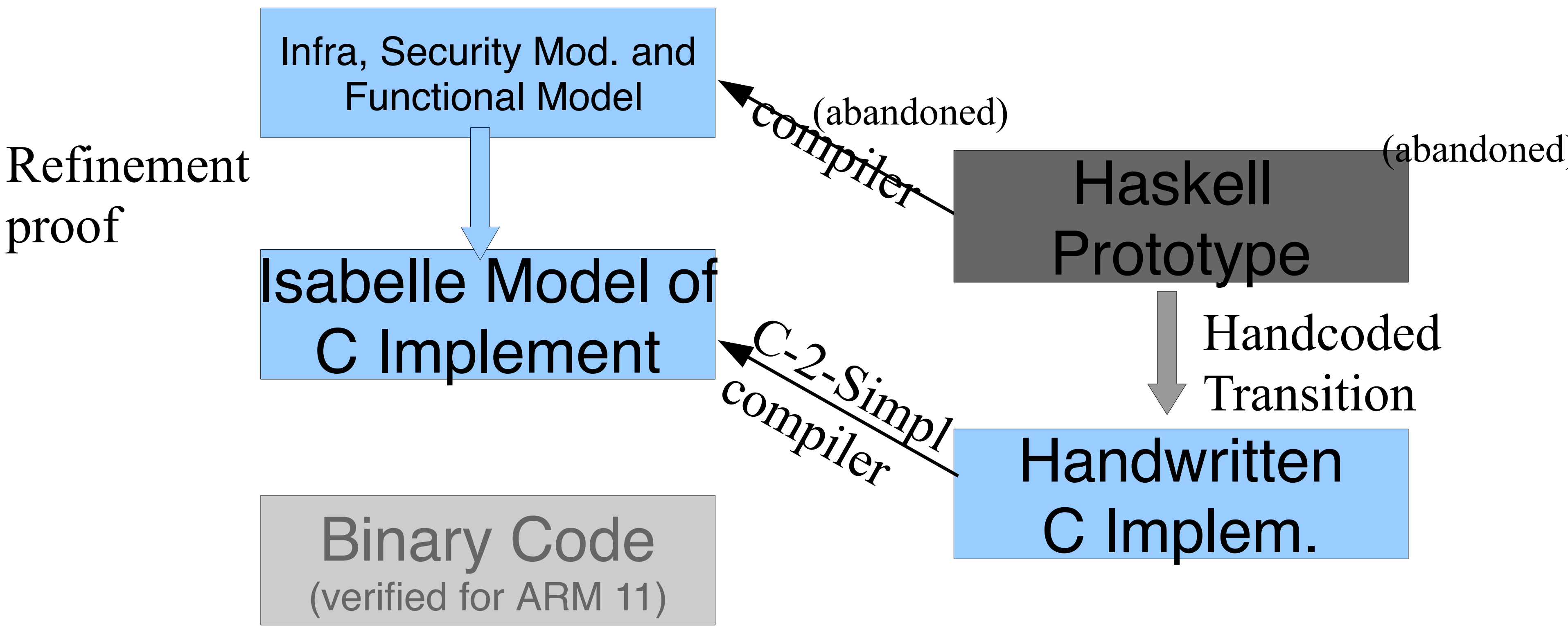
Evaluation

- A Revision of the Development Process



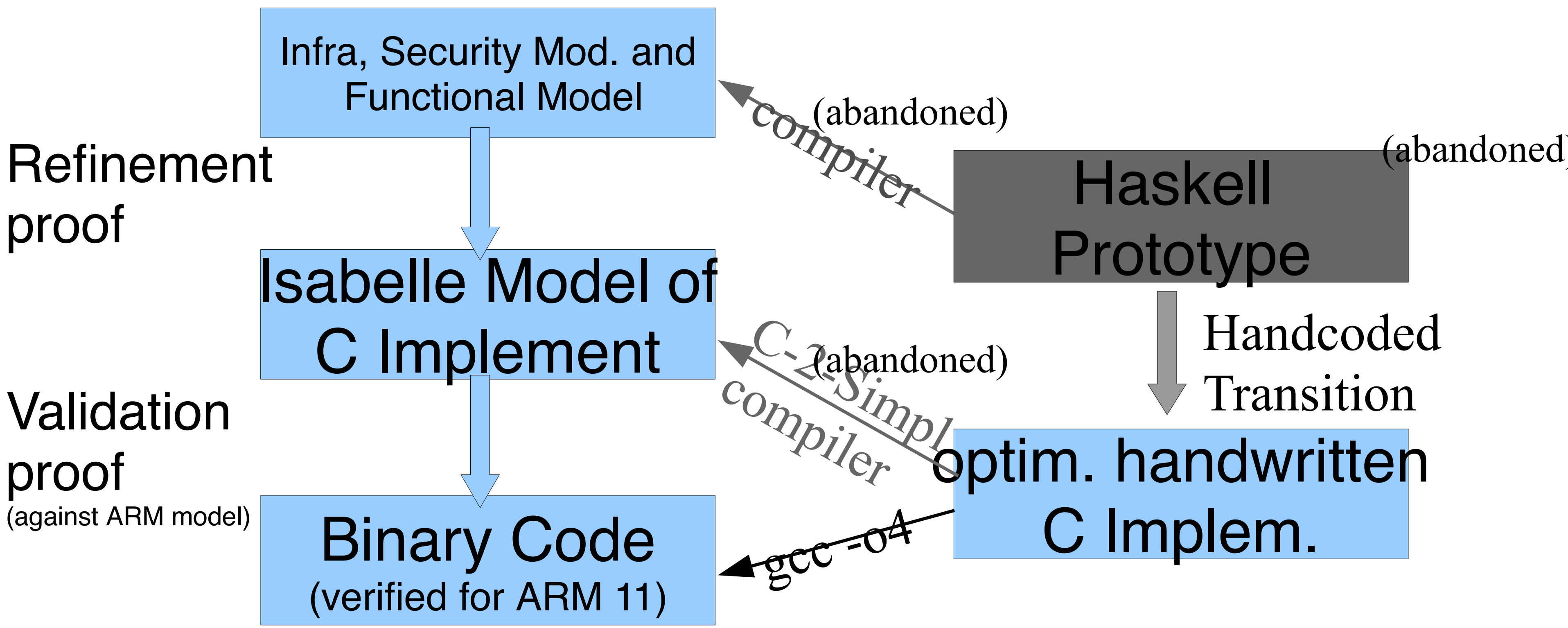
Evaluation

- A Revision of the Development Process



Evaluation

- A Revision of the Development Process



Evaluation

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**

⇒ **Ratio 1 to 20 between code and proof !**

Experimental Evaluation

- in more detail:

Bugs found

during testing: 16

during verification:

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

460 bugs

Experimental Evaluation

- implem errors covered in more detail:

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

Evaluation

- cost analysis

	Total Effort	Artefacts	Effort
Development Effort	2.2 py	Haskell	2 py
		C implementation	0.2 py (2 pm)

(a) Overall Effort for seL4 Development

	Total Effort	Artefacts		Effort
Correctness Proof Effort	20.5 py	Generic framework & tools		9 py
		seL4 formal models	Abstract Spec	0.3 py (4 pm)
			Exec. Spec	0.2 py (3 pm)
		seL4 formal proofs	Refinement 1	8 py
			Refinement 2	3 py

(b) Correctness Proof Effort

	Total Effort	Artefacts	Effort
Optimisation Proof Effort	0.4 py *	Fast Path	0.4 py (5 pm) *

(c) Optimisation Proof Effort

Evaluation

- cost analysis

Security Proof Effort	Total Effort	Artefacts		Effort
	4.1 py *	Integrity		0.6 py (7.4 pm) *
		Confid.	Scheduler Update	0.2 py (1.8 pm) *
			Determinising Spec and Updating Proofs	1.5 py (18.5 pm) *
			Confidentiality Proofs	1.7 py (20.4 pm) *

(d) Security Proof Effort

Binary Verif. Effort	Total Effort	Artefacts	Effort
	2 py	Binary Verification	2 py

(e) Binary Verification Effort

CapDL Effort	Total Effort	Artefacts	Effort
	2 py *	capDL Spec	0.6 py (7.2 pm) *
		capDL-to-Abstract Spec refinement proof	1.4 py (17.2 pm) *

(f) capDL Effort

Evaluation

- cost analysis
  - overall : 25 py investment, mostly for the refinement proof
  - about 10 py infrastructure (reusable?)
  - arguably cost effective:

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**

Evaluation

- what is missing

  - well, seL4 is a kernel, not an OS with, say, an POSIX interface.
  - Components such as filesystems TCP/IP stacks, firewalls and posix-libraries are missing.
  - proof methodology for applications using take-grant security model.

seL4 is free - what does this mean to you ?

- seL4 became an open source project
  (see video https://www.youtube.com/watch?v=lRndE7rSXiI)

The seL4 Microkernel

Security is no excuse for poor performance!

seL4 Security. Performance. Proof.

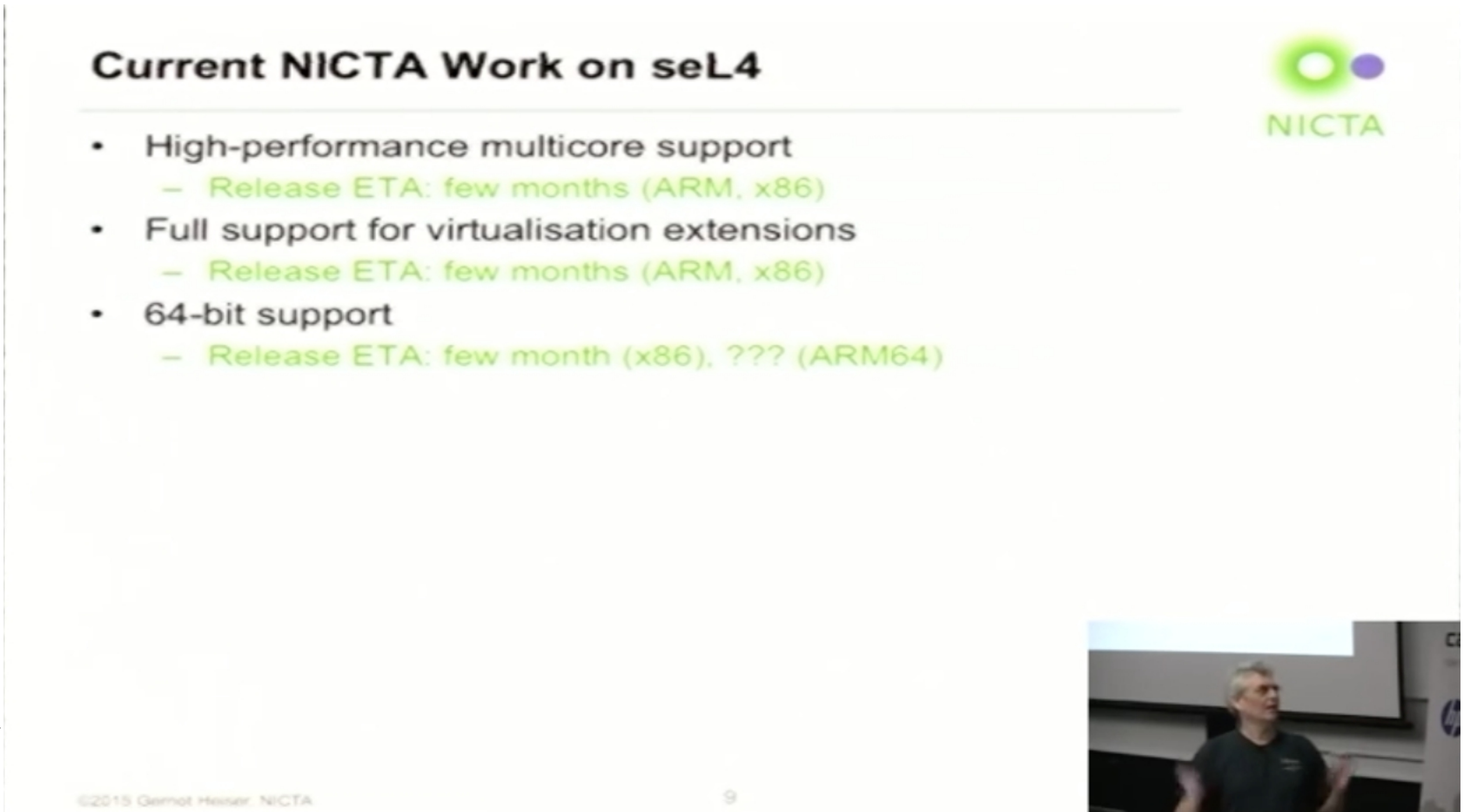
The world's first operating-system kernel with an end-to-end proof of implementation correctness and security enforcement is available as open source.

Sign up to sel4-announce | Sign up to sel4-devel

How to get it | on GitHub | FAQ

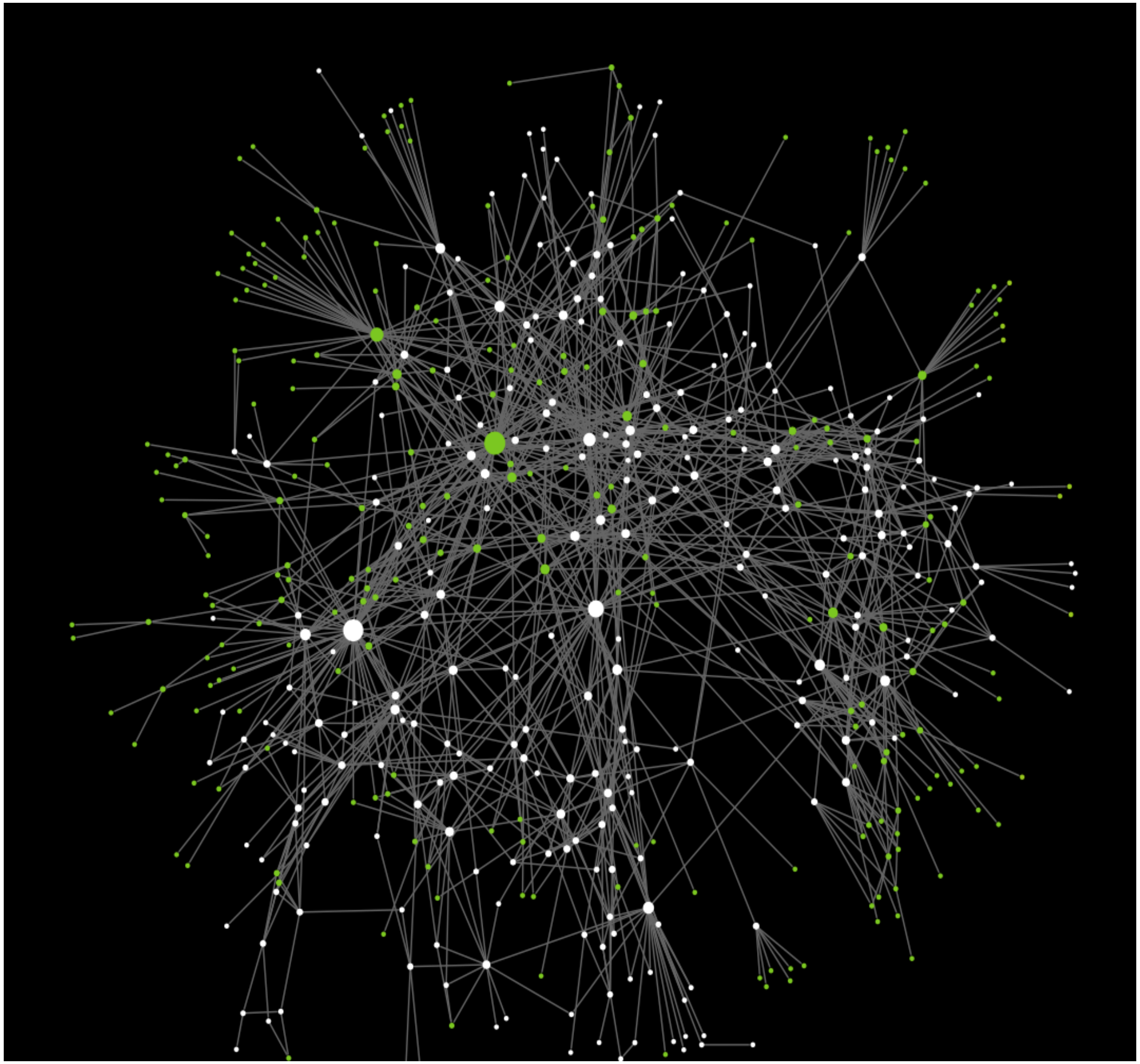
seL4 is free - what does this mean to you ?

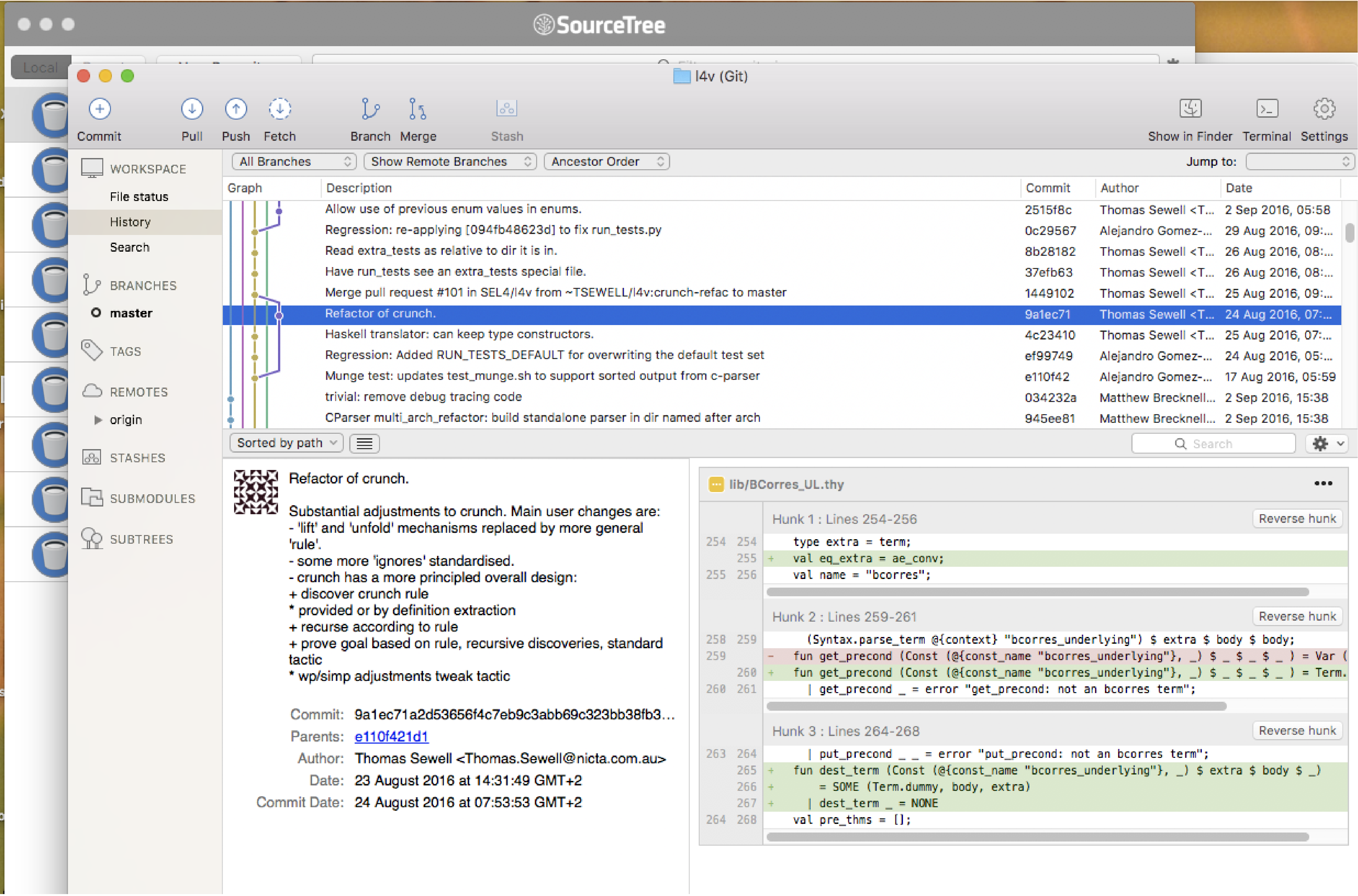
- seL4 became an open source project
  (see video https://www.youtube.com/watch?v=lRndE7rSXiI)



seL4 is free - what does this mean to you ?

- anybody can contribute
  (and chances of acceptance are high if proof provided)
- consistency can be maintained even in distributed collaboration
  (easy impact analysis in Isabelle)




SourceTree
l4v (Git)
Commit
Pull
Push
Fetch
Branch
Merge
Stash
Show in Finder
Terminal
Settings
WORKSPACE
File status
History
Search
BRANCHES
master
TAGS
REMOTES
origin
STASHES
SUBMODULES
SUBTREES
All Branches
Show Remote Branches
Ancestor Order
Jump to:
Graph
Description
Commit
Author
Date
Allow use of previous enum values in enums. 2515f8c Thomas Sewell <T... 2 Sep 2016, 05:58
Regression: re-applying [094fb48623d] to fix run_tests.py 0c29567 Alejandro Gomez-... 29 Aug 2016, 09:...
Read extra_tests as relative to dir it is in. 8b28182 Thomas Sewell <T... 26 Aug 2016, 08:...
Have run_tests see an extra_tests special file. 37efb63 Thomas Sewell <T... 26 Aug 2016, 08:...
Merge pull request #101 in SEL4/l4v from ~TSEWELL/l4v:crunch-refac to master 1449102 Thomas Sewell <T... 25 Aug 2016, 09:...
Refactor of crunch. 9a1ec71 Thomas Sewell <T... 24 Aug 2016, 07:...
Haskell translator: can keep type constructors. 4c23410 Thomas Sewell <T... 25 Aug 2016, 07:...
Regression: Added RUN_TESTS_DEFAULT for overwriting the default test set ef99749 Alejandro Gomez-... 24 Aug 2016, 05:...
Munge test: updates test_munge.sh to support sorted output from c-parser e110f42 Alejandro Gomez-... 17 Aug 2016, 05:59
trivial: remove debug tracing code 034232a Matthew Brecknell... 2 Sep 2016, 15:38
CParser multi_arch_refactor: build standalone parser in dir named after arch 945ee81 Matthew Brecknell... 2 Sep 2016, 15:38
Sorted by path
Search
Refactor of crunch.
Substantial adjustments to crunch. Main user changes are:
- 'lift' and 'unfold' mechanisms replaced by more general 'rule'.
- some more 'ignores' standardised.
- crunch has a more principled overall design:
+ discover crunch rule
* provided or by definition extraction
+ recurse according to rule
+ prove goal based on rule, recursive discoveries, standard tactic
* wp/simp adjustments tweak tactic
Commit: 9a1ec71a2d53656f4c7eb9c3abb69c323bb38fb3...
Parents: e110f421d1
Author: Thomas Sewell <Thomas.Sewell@nicta.com.au>
Date: 23 August 2016 at 14:31:49 GMT+2
Commit Date: 24 August 2016 at 07:53:53 GMT+2
lib/BCorres_UL.thy
Hunk 1 : Lines 254-256
Reverse hunk
254 254 type extra = term;
255 + val eq_extra = ae_conv;
255 256 val name = "bcorres";
Hunk 2 : Lines 259-261
Reverse hunk
258 259 (Syntax.parse_term @{context} "bcorres_underlying") $ extra $ body $ body;
259 - fun get_precond (Const (@{const_name "bcorres_underlying"}, _) $ _ $ _ $ _) = Var (
260 + fun get_precond (Const (@{const_name "bcorres_underlying"}, _) $ _ $ _ $ _) = Term.
260 261 | get_precond _ = error "get_precond: not an bcorres term";
Hunk 3 : Lines 264-268
Reverse hunk
263 264 | put_precond _ _ = error "put_precond: not an bcorres term";
265 + fun dest_term (Const (@{const_name "bcorres_underlying"}, _) $ extra $ body $ _)
266 + = SOME (Term.dummy, body, extra)
267 + | dest_term _ = NONE
264 268 val pre_thms = [];

seL4 is free - what does this mean to you ?

- further increases of cost-effectiveness

What Else Is Cooking?

NICTA

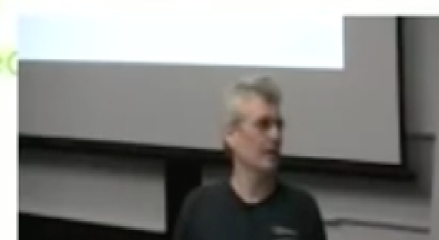
- Aim: Cost reduction by automation and abstraction
  - Present seL4 cost: $400/SLOC, high-assurance, high-performance
  - Other "high" assurance: $1,000/SLOC, no proof, poor performance
  - Low assurance (Pistachio): $200/SLOC, no proof, high performance

seL4 is free - what does this mean to you ?

How Can YOU Contribute?

NICTA

- Libraries presently extremely rudimentary
  - POSIX! …
- Platform ports
  - Especially popular ARM boards: Tegra, RK3188, Beaglebone, …
- Drivers!!!!!!
  - Very few available ATM
- Network stacks and file systems
  - Presently have lwIP, incomplete functionality
- Tools
  - Have component system (CAmkES), glue generators
- Languages
  - Core C++ support just released, lacks std template lib
  - Haskell presently in progress (with Galois) – stay tune
  - Python would be **awesome!**



seL4 is free - what does this mean to you ?

Conclusion

- Formal Development based on ITP technology is feasible for critical systems of 10 k size C...
- ... and can be cost-effective for high-quality, complex code in a certification process.
- collaborative and open-source development is strong point of FM developments; impact analysis is easy for changes
- seL4 is reusable, but so far not much trusted code for libraries exists ...