



# L4.verified 3 years later

Gerwin Klein



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

## NICTA Funding and Supporting Members and Partners



Australian  
National  
University

UNSW  
THE UNIVERSITY OF NEW SOUTH WALES



Trade &  
Investment



State Government  
Victoria



THE UNIVERSITY OF  
MELBOURNE



THE UNIVERSITY OF  
SYDNEY



Queensland  
Government



Griffith  
UNIVERSITY



Queensland University of Technology

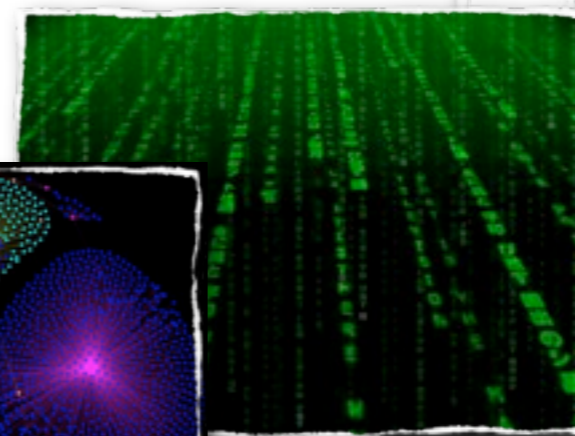


THE UNIVERSITY OF  
QUEENSLAND  
AUSTRALIA

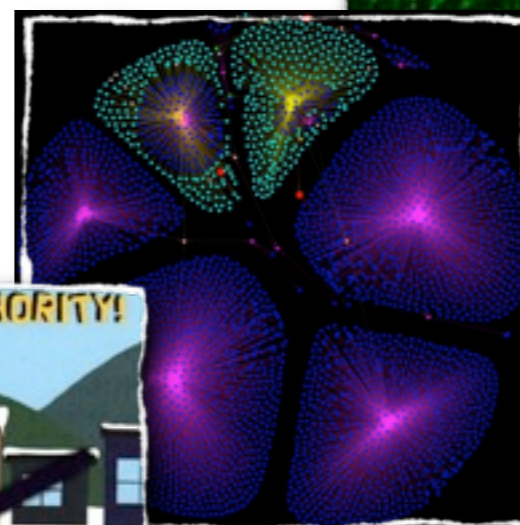
# Plan



**WCET**



**Binary Verification**



**capDL**



**Integrity & Non-Interference**



**History & Software Process**

# History & Software Process



NCET  
ation

## History & Software Process

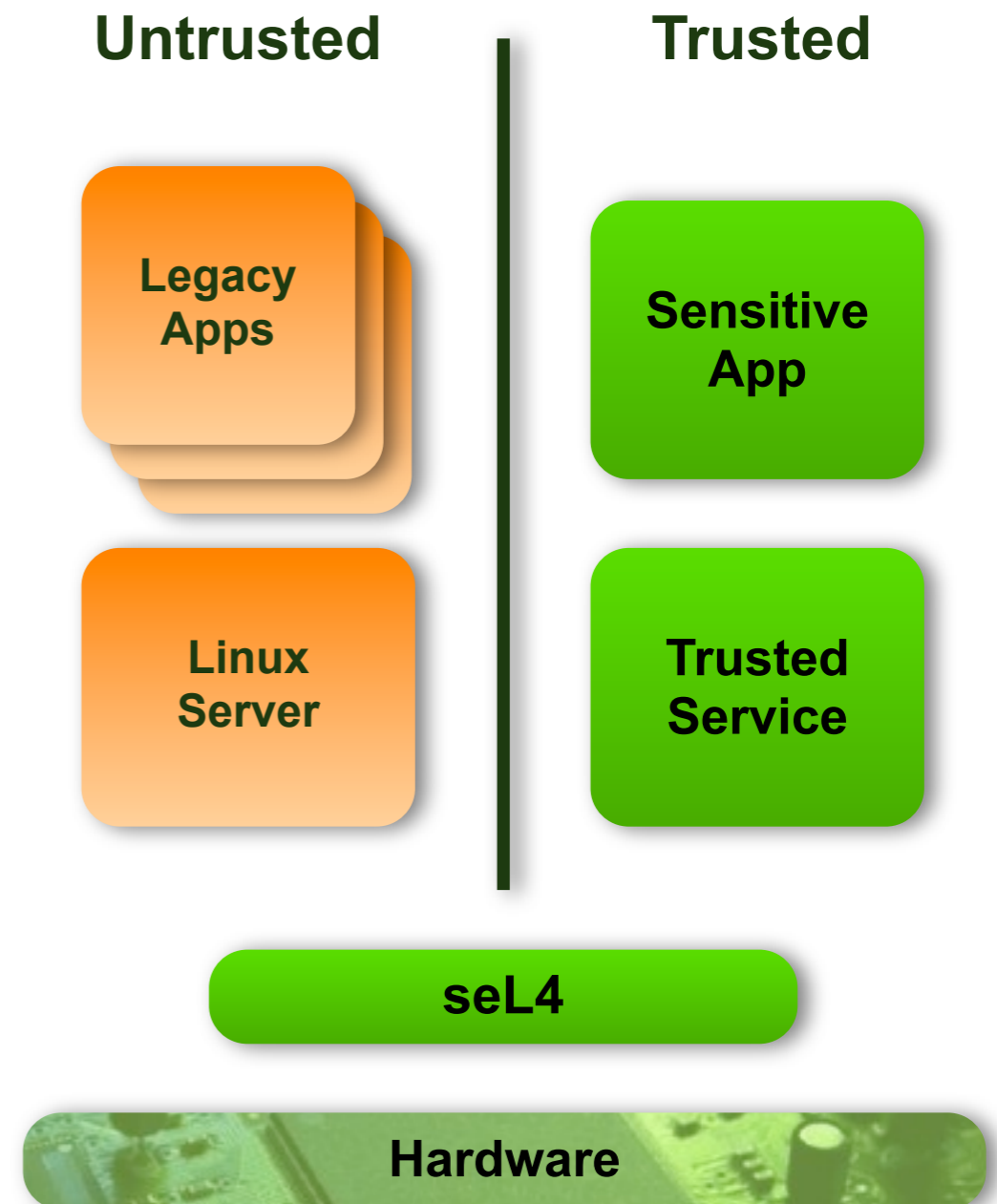
# Small Kernels

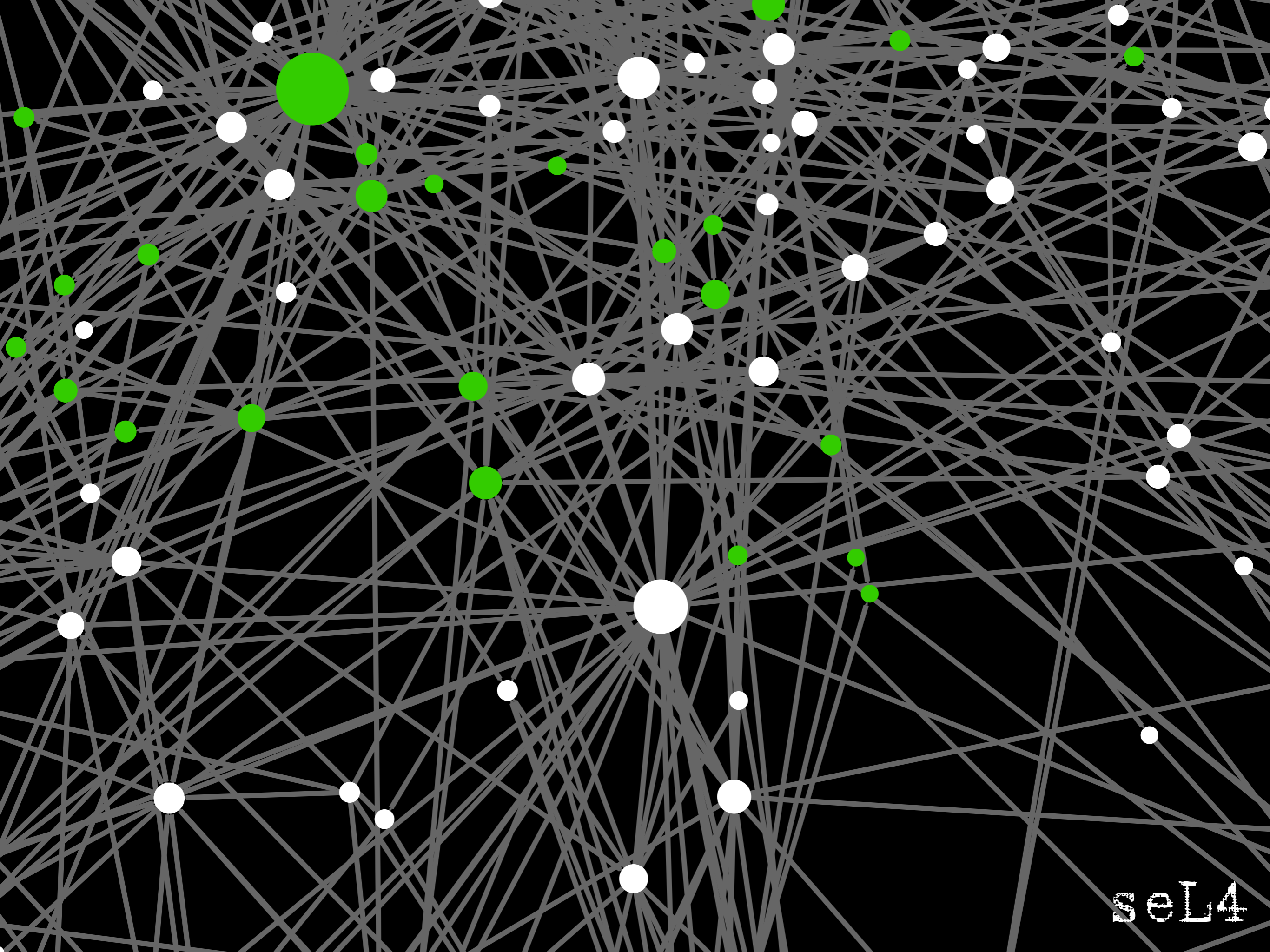
## Small trustworthy foundation

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

### seL4 API:

- IPC
- Threads
- VM
- IRQ
- Capabilities





seL4

# Functional Correctness

What

Specification

definition

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

Proof

How

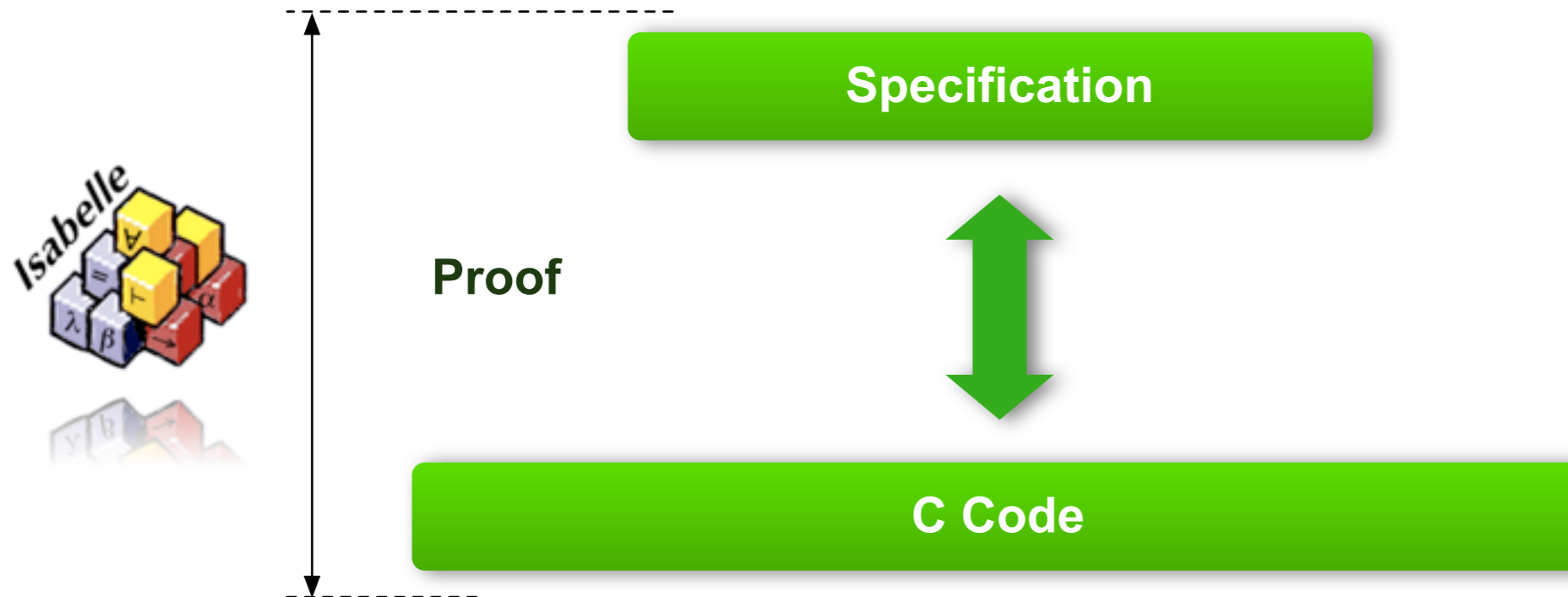
```
void
schedule(void) {
  switch ((word_t)ksSchedulerAction) {
    case (word_t)SchedulerAction_ResumeCurrentThread:
      break;

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

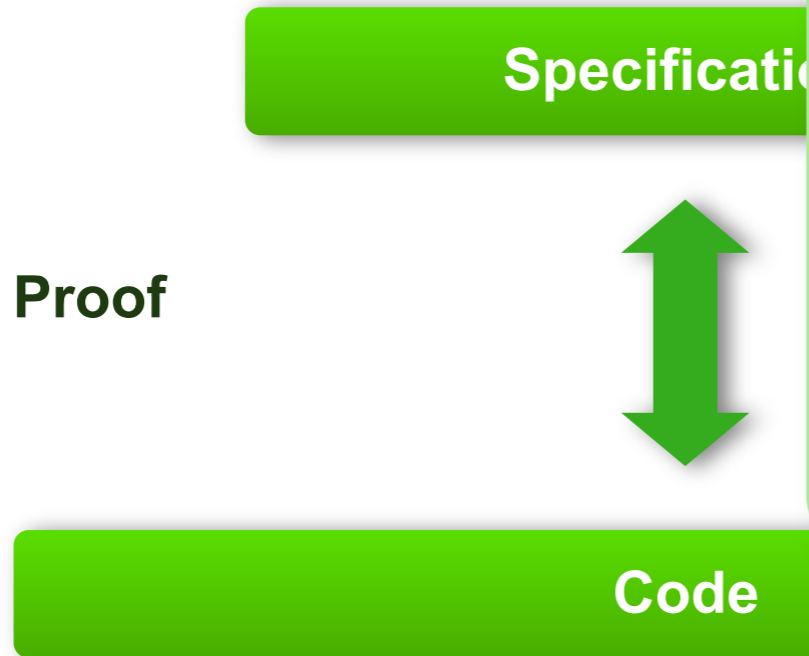
    default: /* SwitchToThread */
      switchToThread(ksSchedulerAction);
      ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
      break;
  }
}

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

# Proof Architecture



\*conditions apply



**Assume correct:**

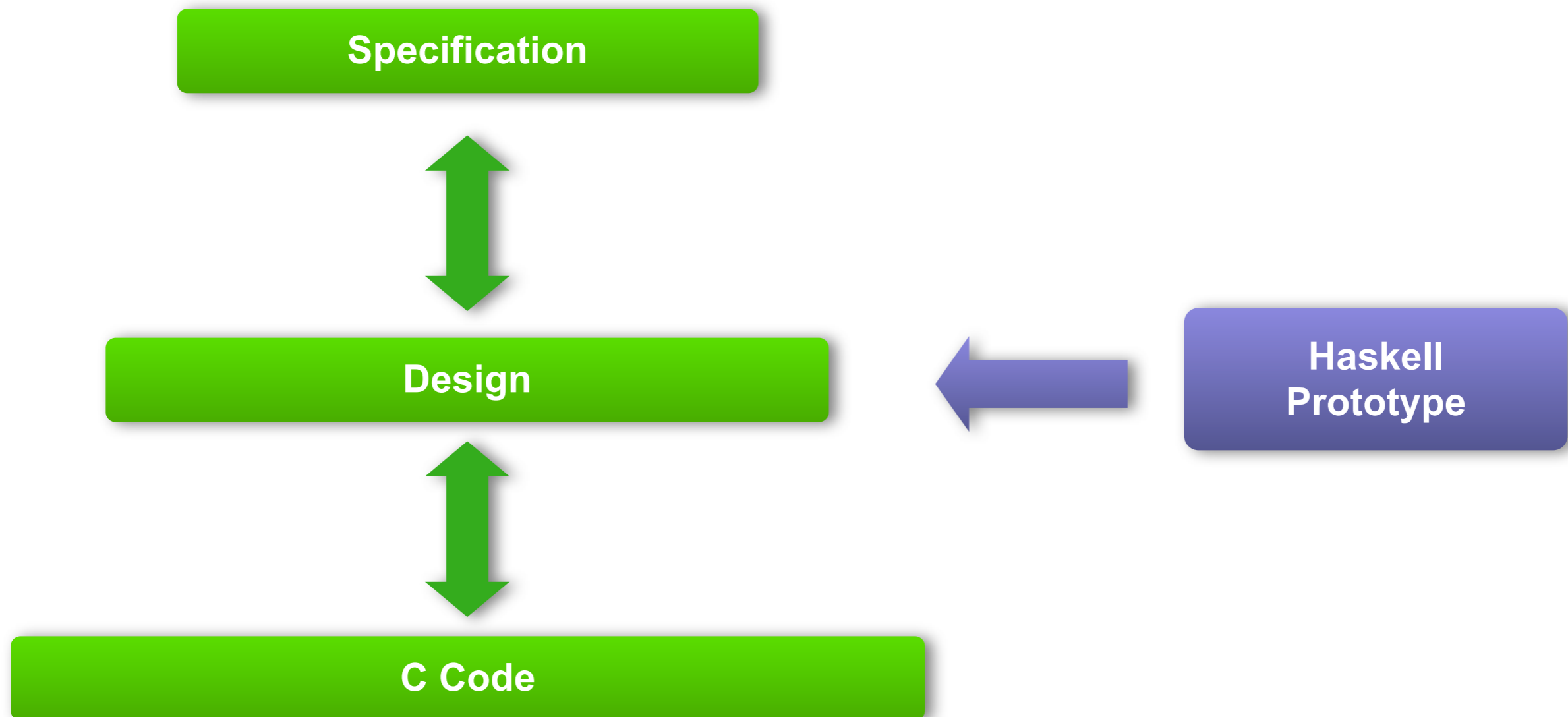
- compiler + linker (wrt. C op-sem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

**Assumptions**

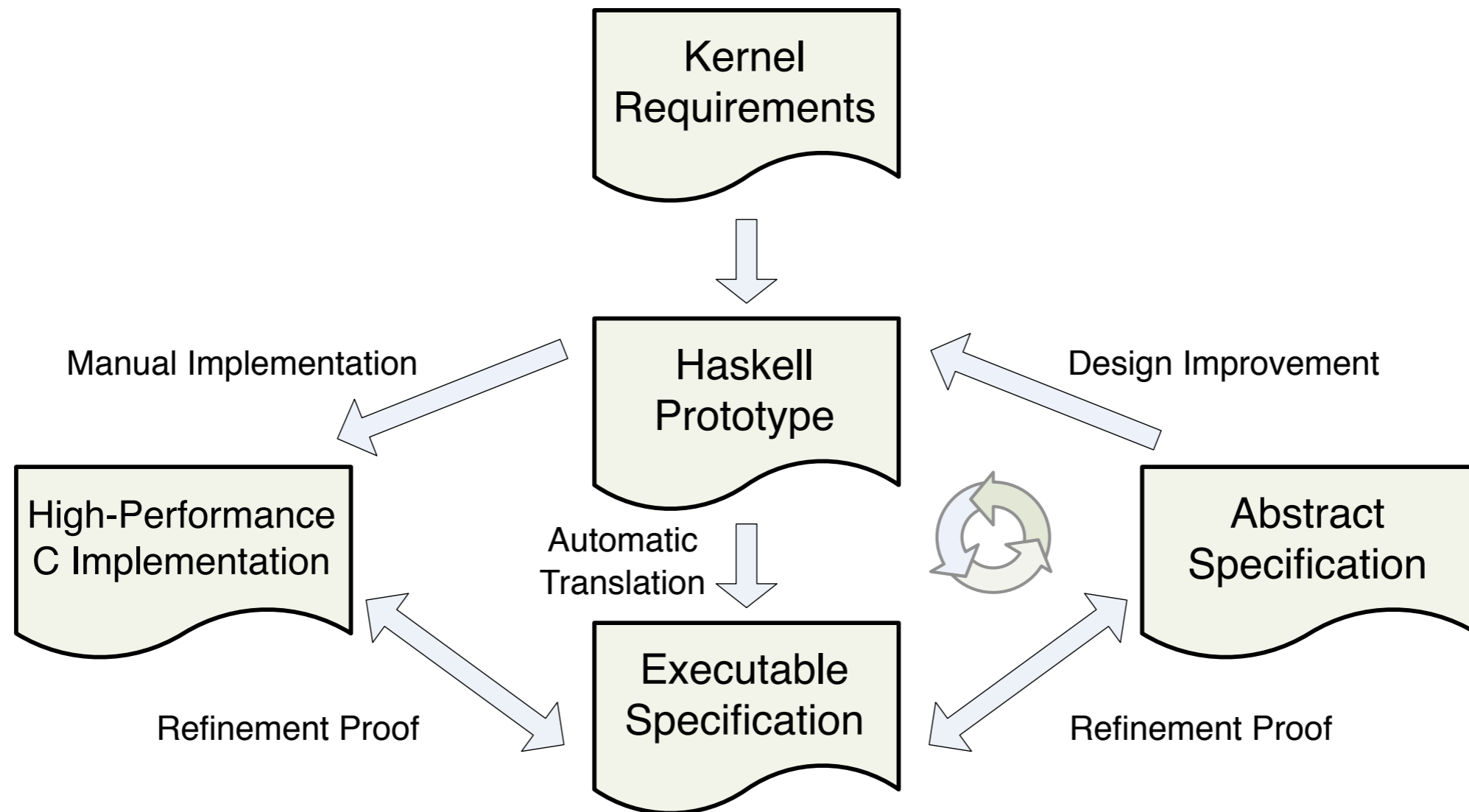




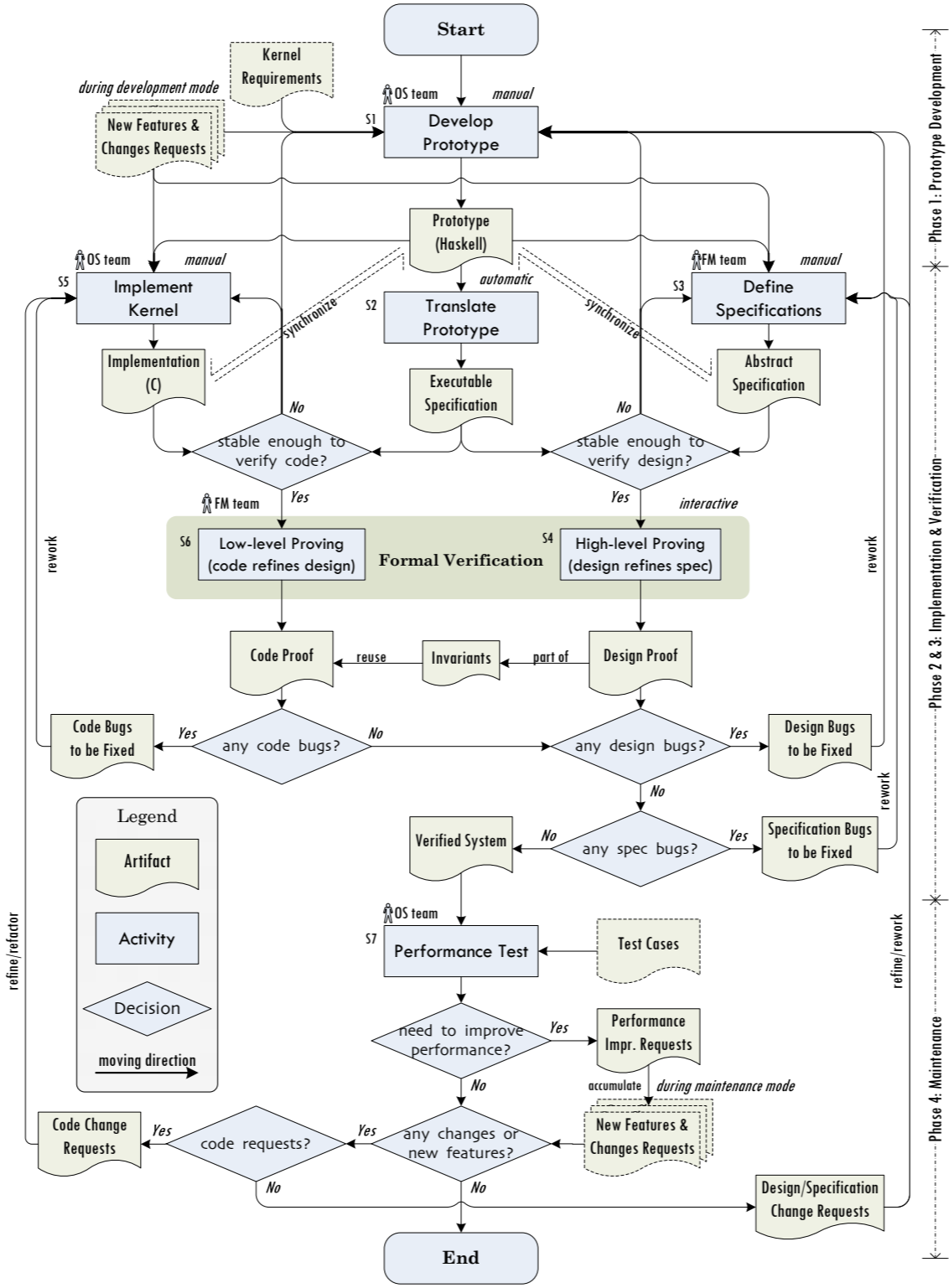
# Proof Architecture



# Conceptual process model



# Descriptive process model



# Data Mining

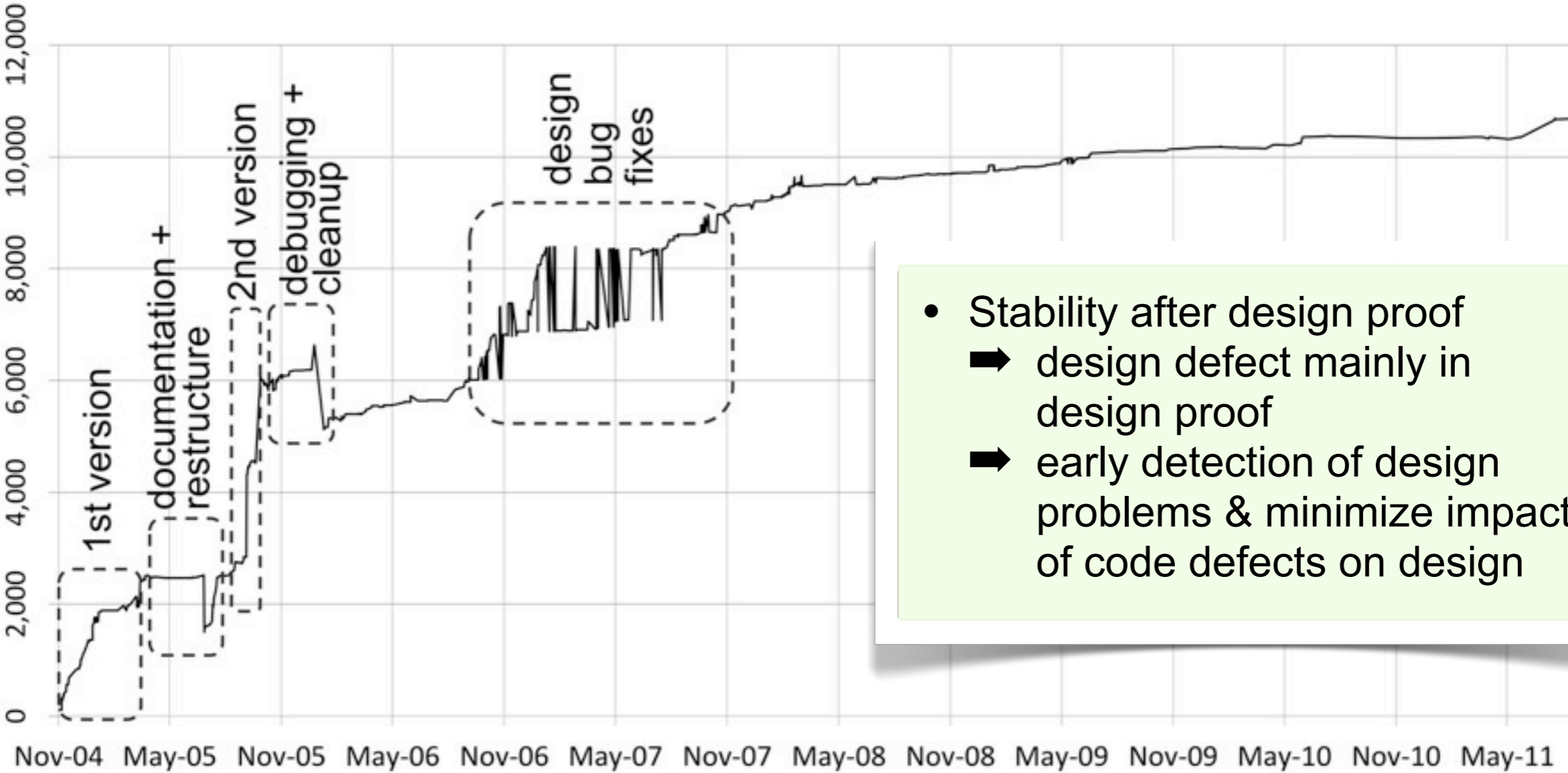


svn/  
hg

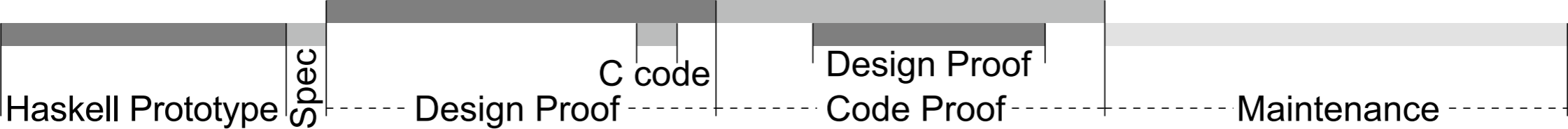
darcs

```
...  
{  
  "author": "  
  "date": "  
  "hex": "  
  "is_me": "  
  "modul": "  
  {  
    "line": "  
    "line": "  
    "line": "  
    "nam": "  
  }  
],  
"seque": "  
"short": "  
"total_l": "  
"utc_tir": "  
},  
...
```

# Graphs: Haskell

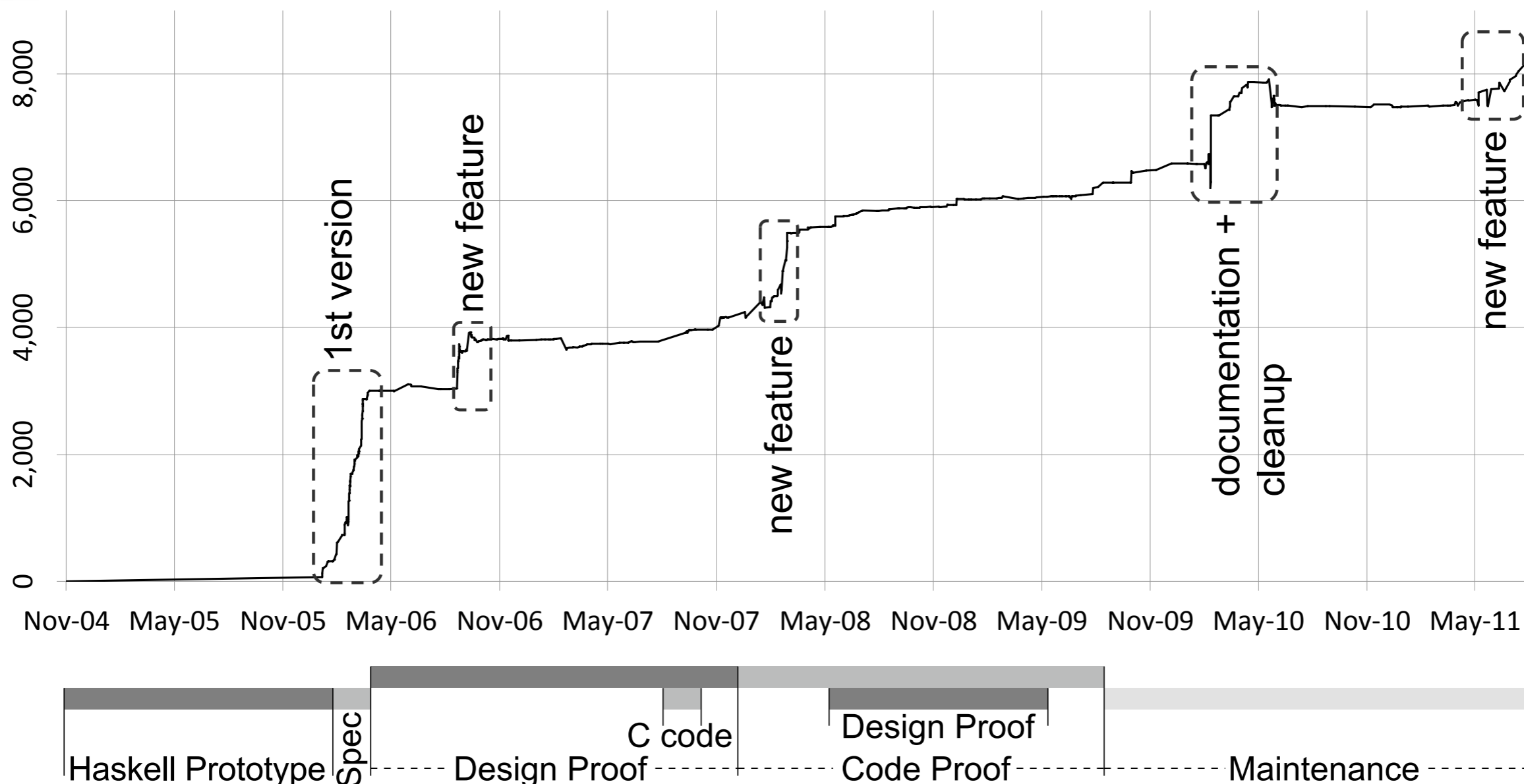


- Stability after design proof
  - ➔ design defect mainly in design proof
  - ➔ early detection of design problems & minimize impact of code defects on design



# Graphs: Abstract

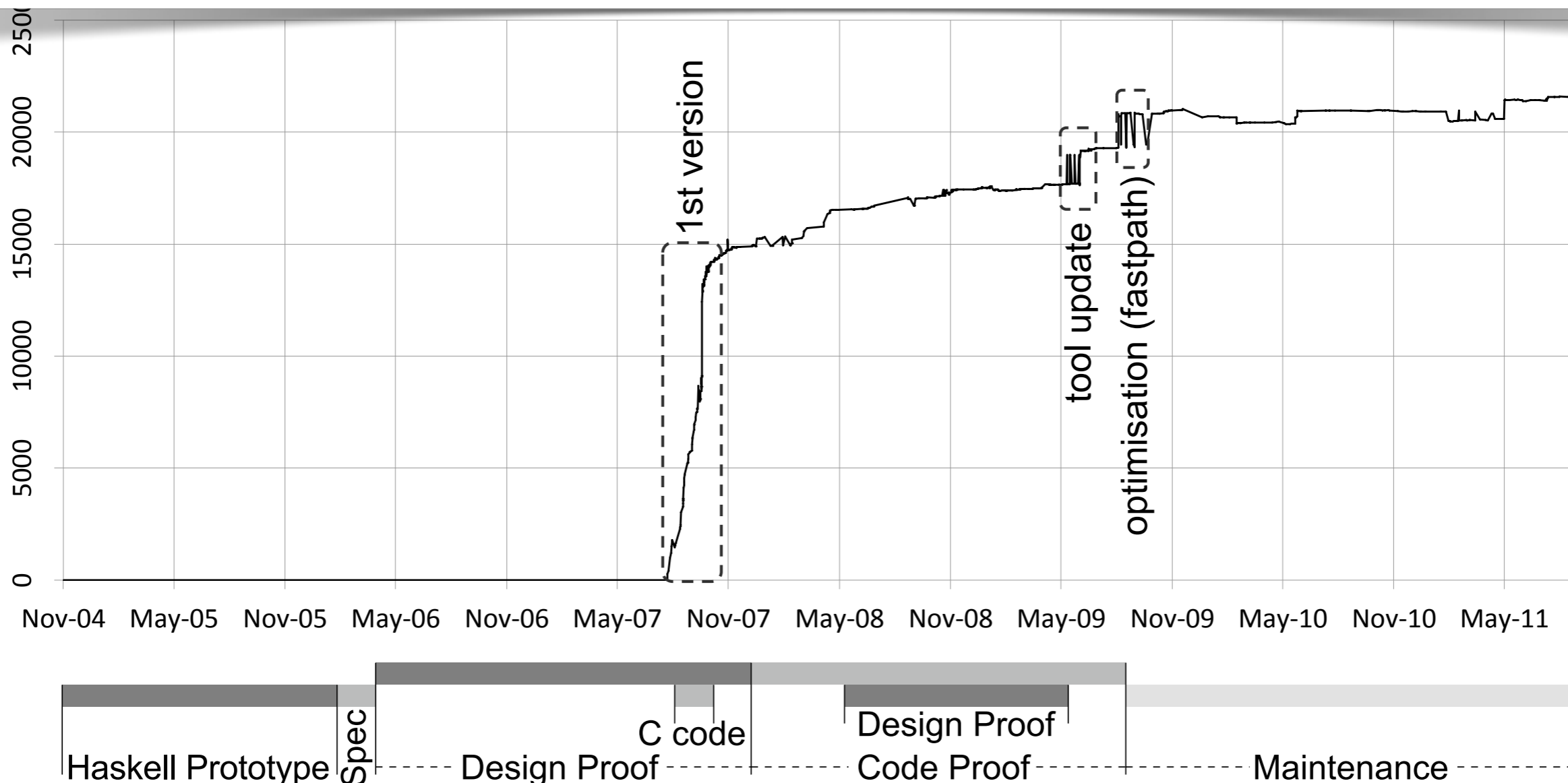
- Spec only started when prototype was stable
- New features happen at every stage and are the main big changes in spec size



# Graphs: C code



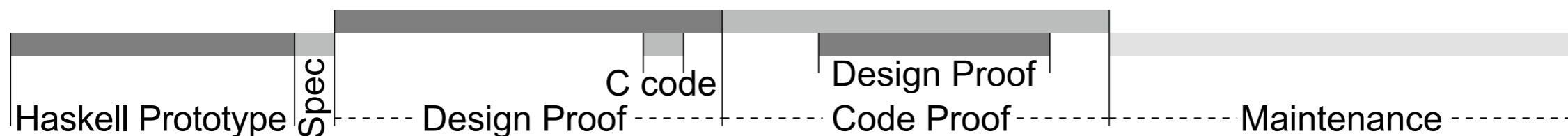
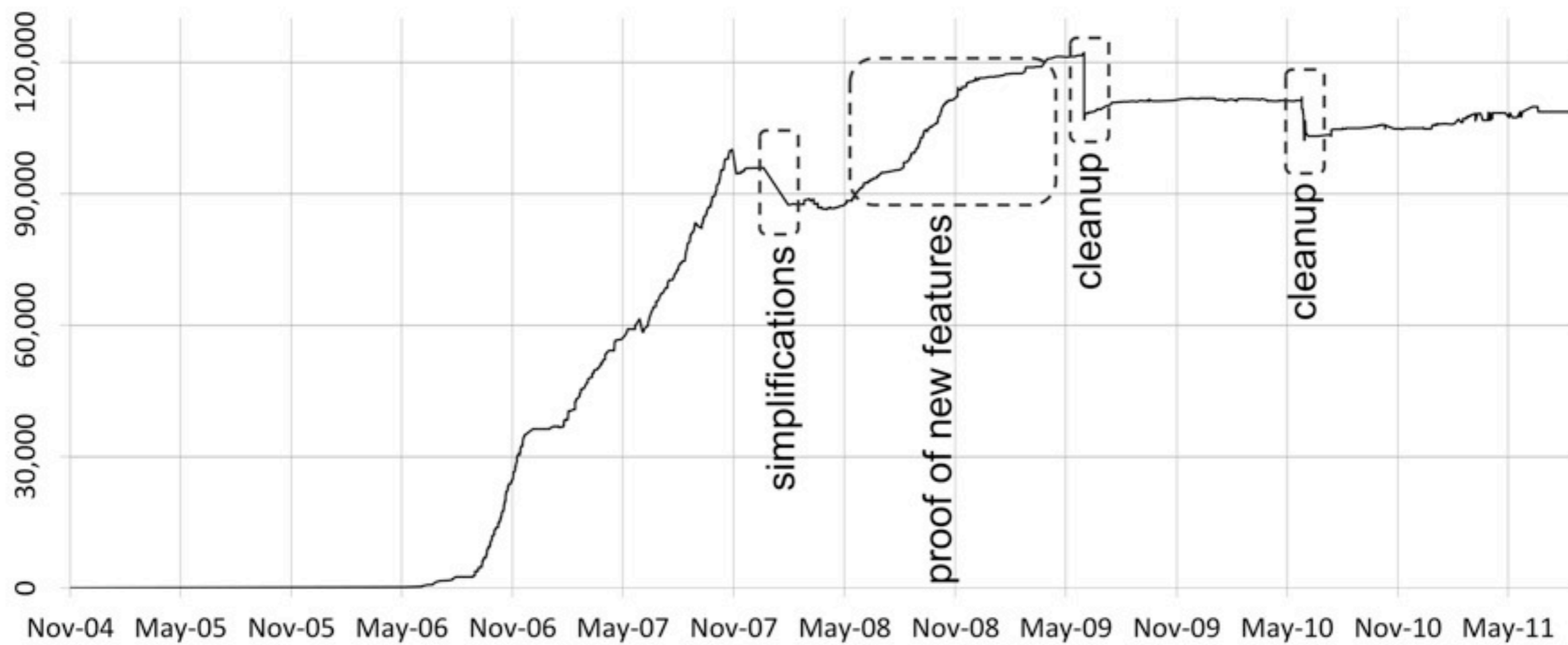
- C code implemented extremely rapidly (design decision already made and partly validated with ongoing design proof)
- separation of concern between design/spec/impl (eg: fastpath change didn't impact spec/design much, only code)



# Graphs: Design Proof



- highlight proof phases and how they may overlap: in parallel with other proofs
- first attempt followed by updates due to new features

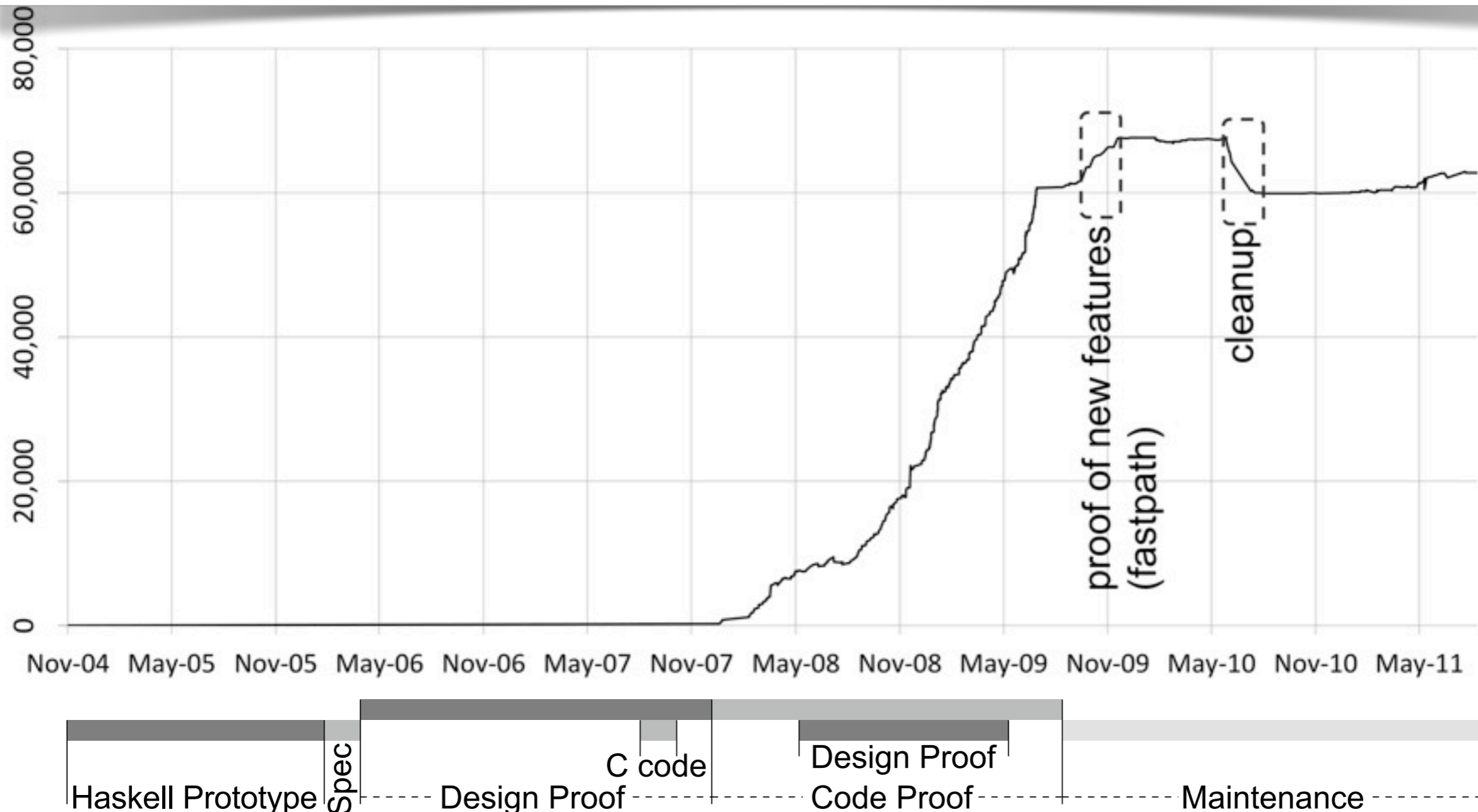




# Graphs: Code Proof



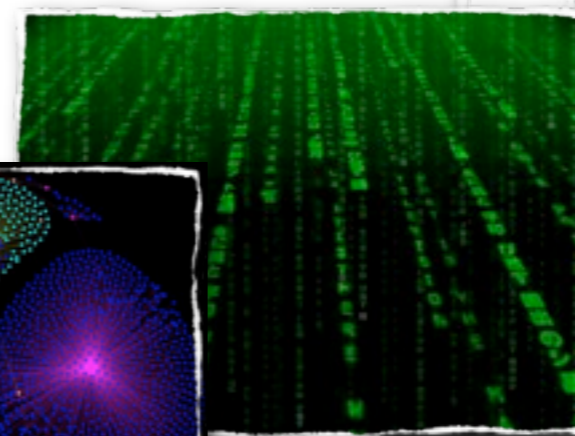
- highlight proof phases and how they may overlap: in parallel with other proofs
- first attempt followed by updates due to new features



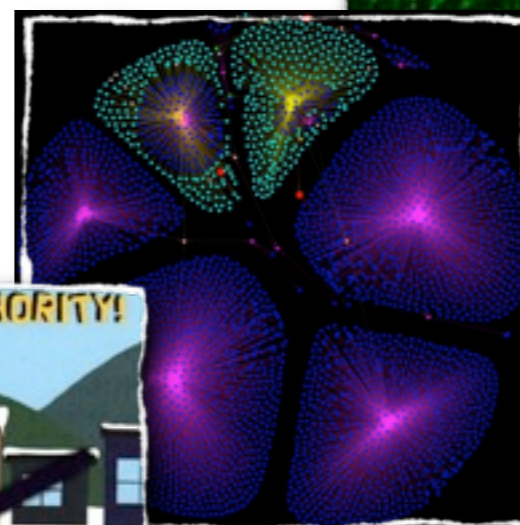
# Plan



**WCET**



**Binary Verification**



**capDL**



**Integrity & Non-Interference**



**History & Software Process**

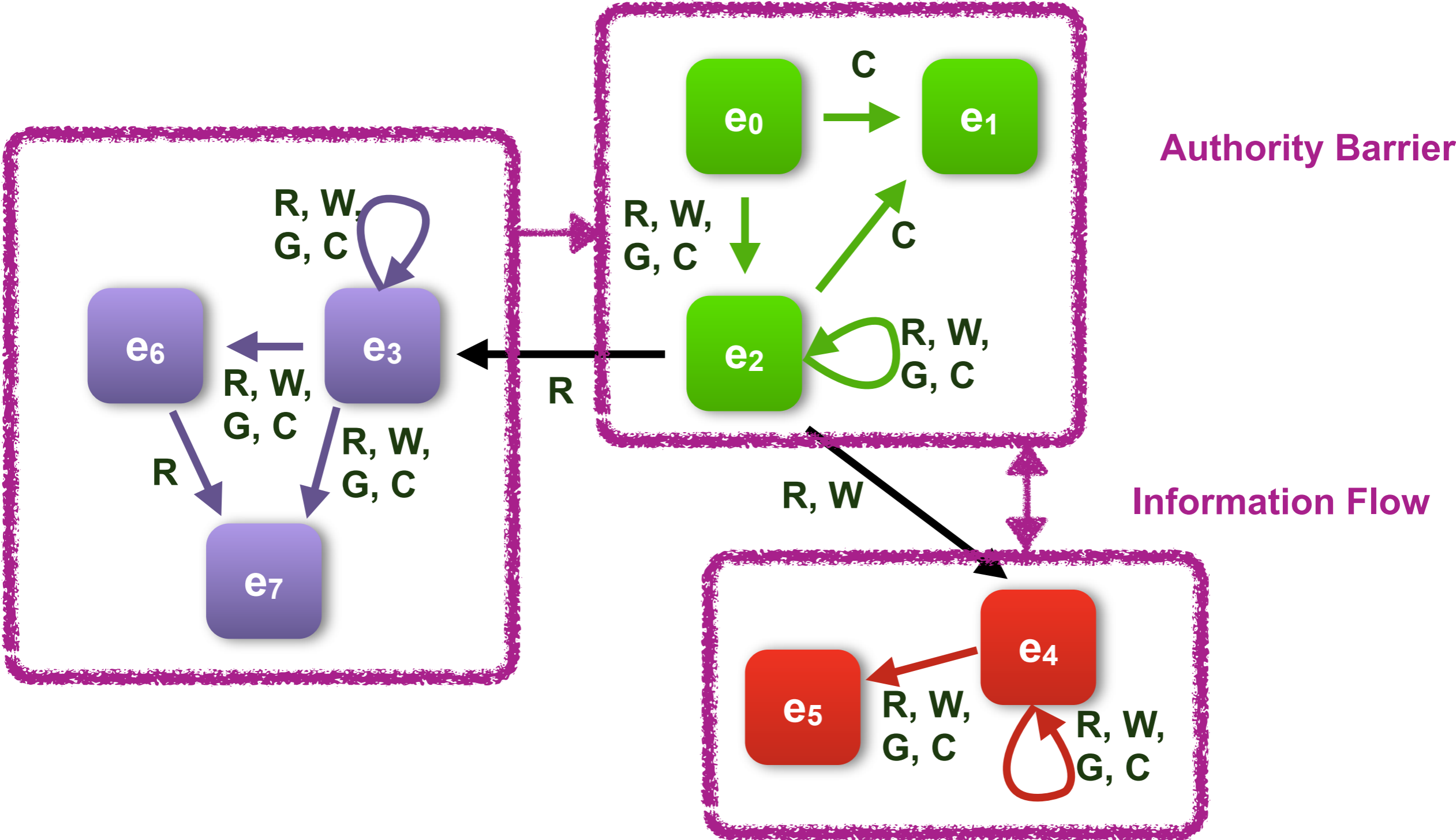
# Integrity & Non-Interference



**Integrity & Non-Interference**

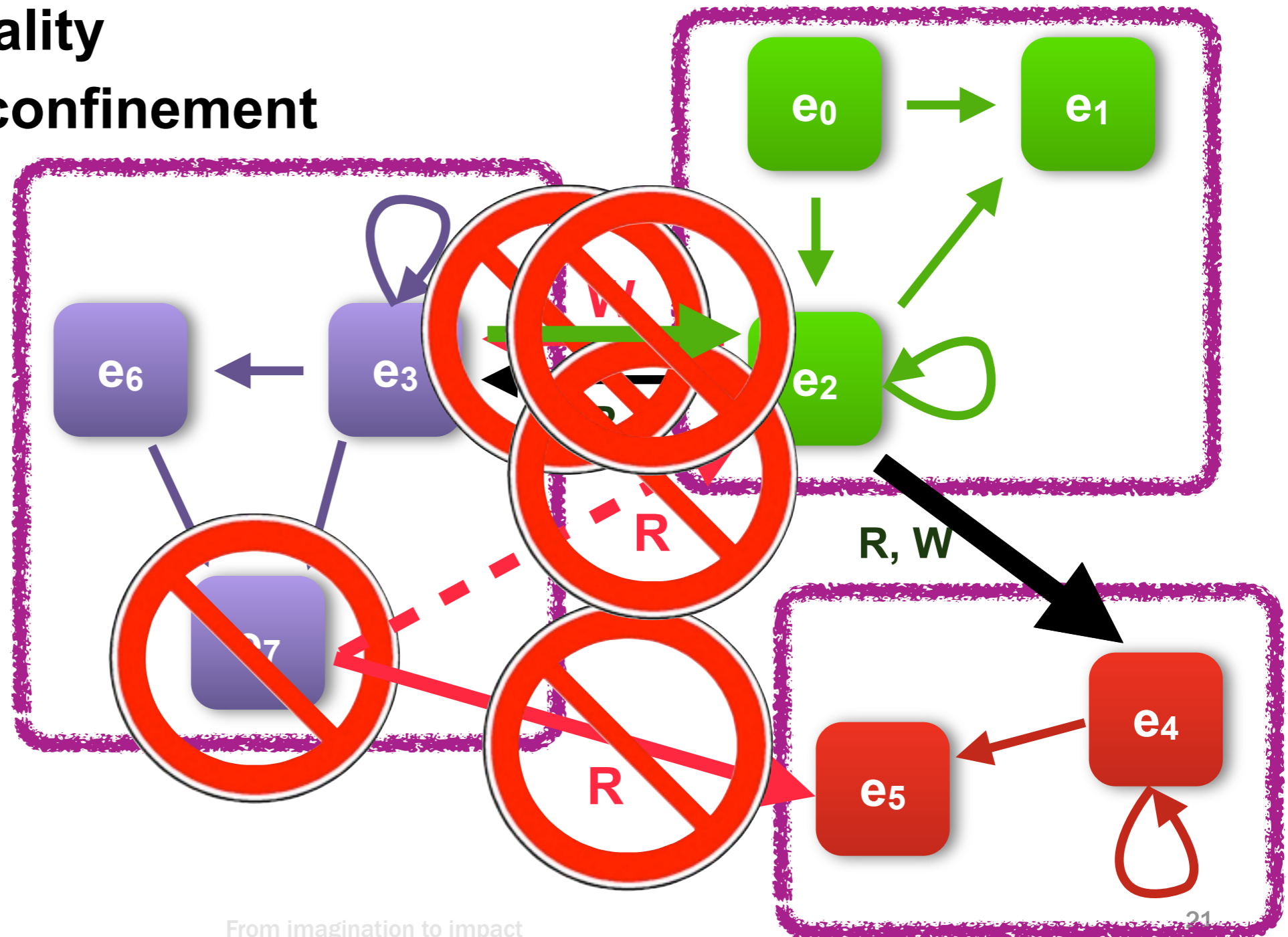


# Capability Access Control

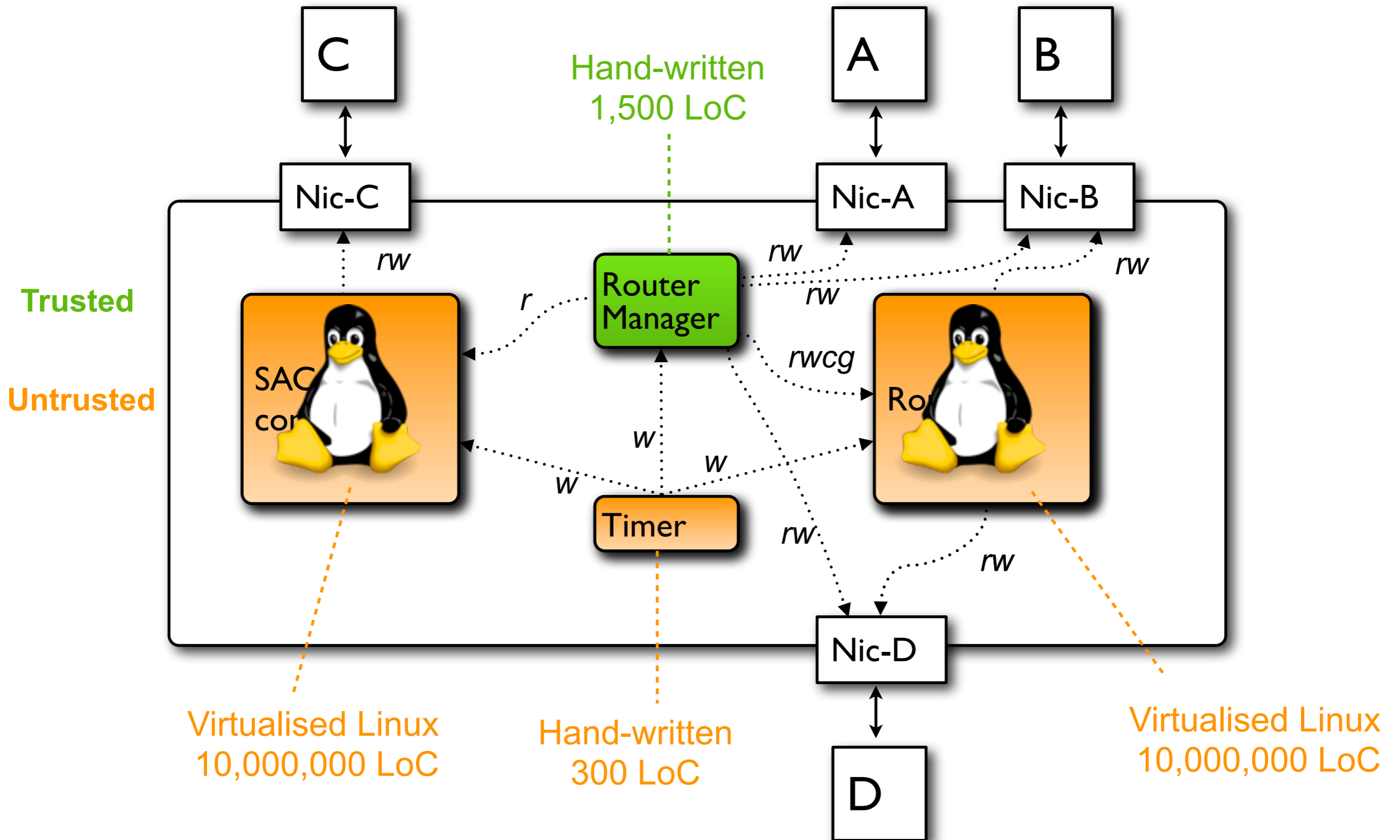


# Access Control Enforcement

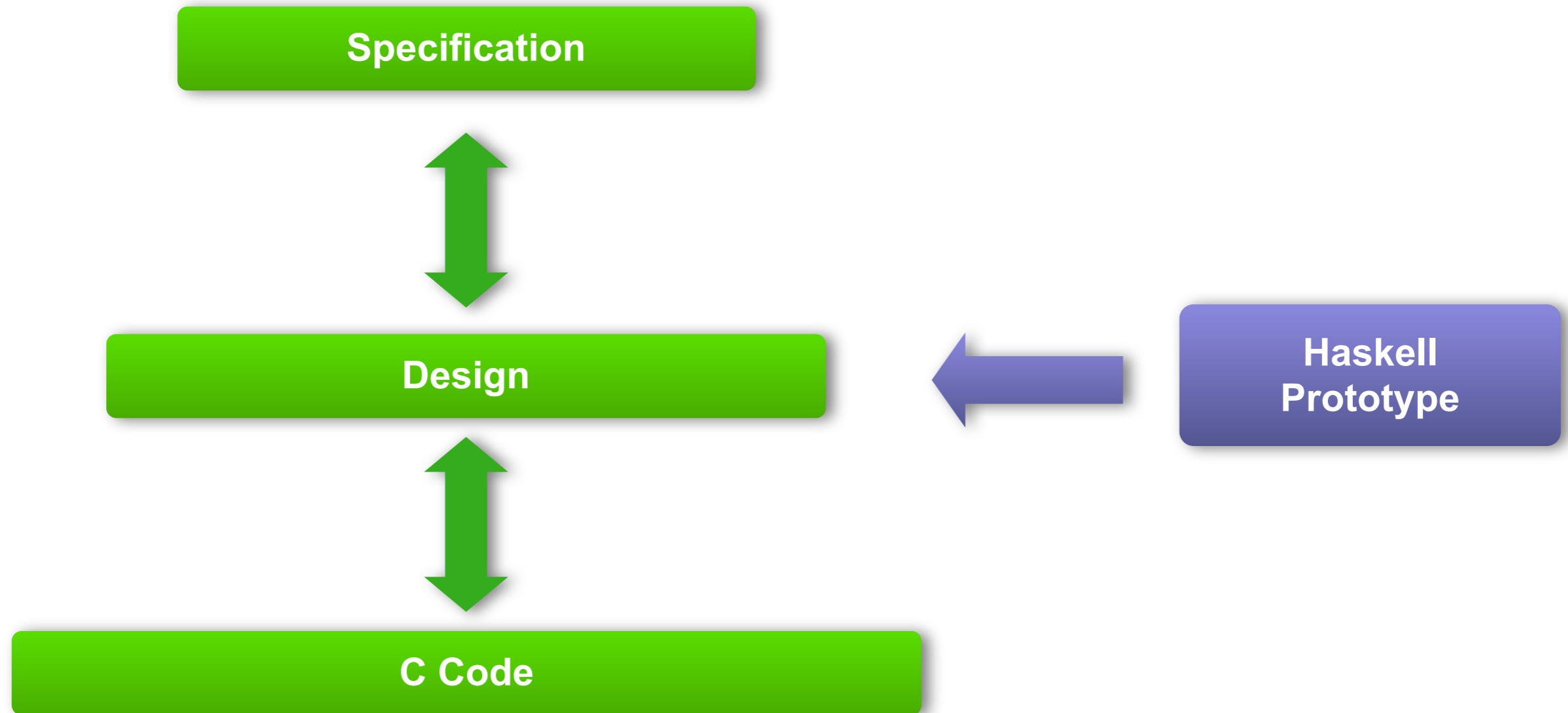
- Correctly enforcing the model
  - Integrity
  - Confidentiality
  - Authority confinement



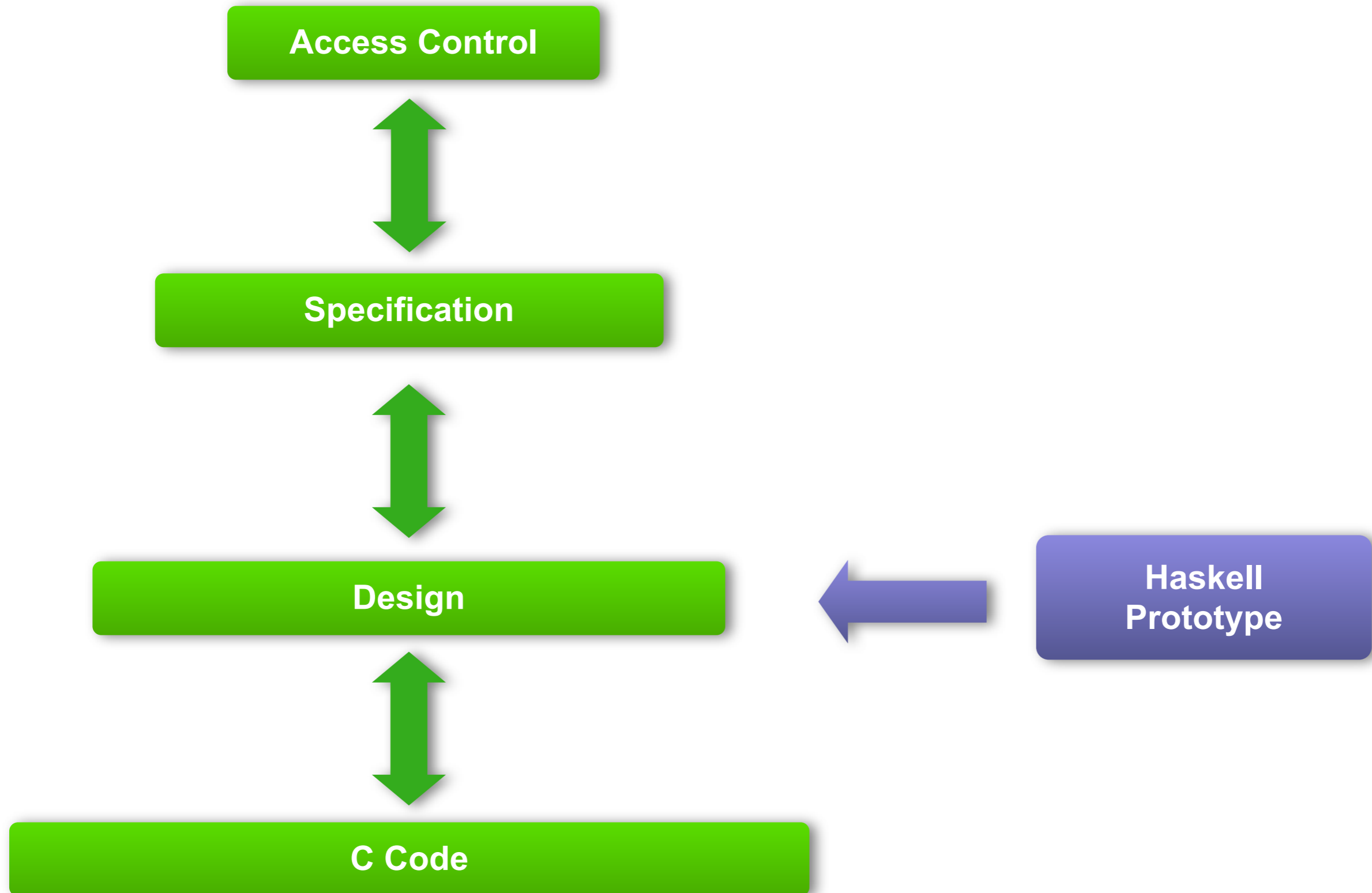
# Example System



# Proof Architecture

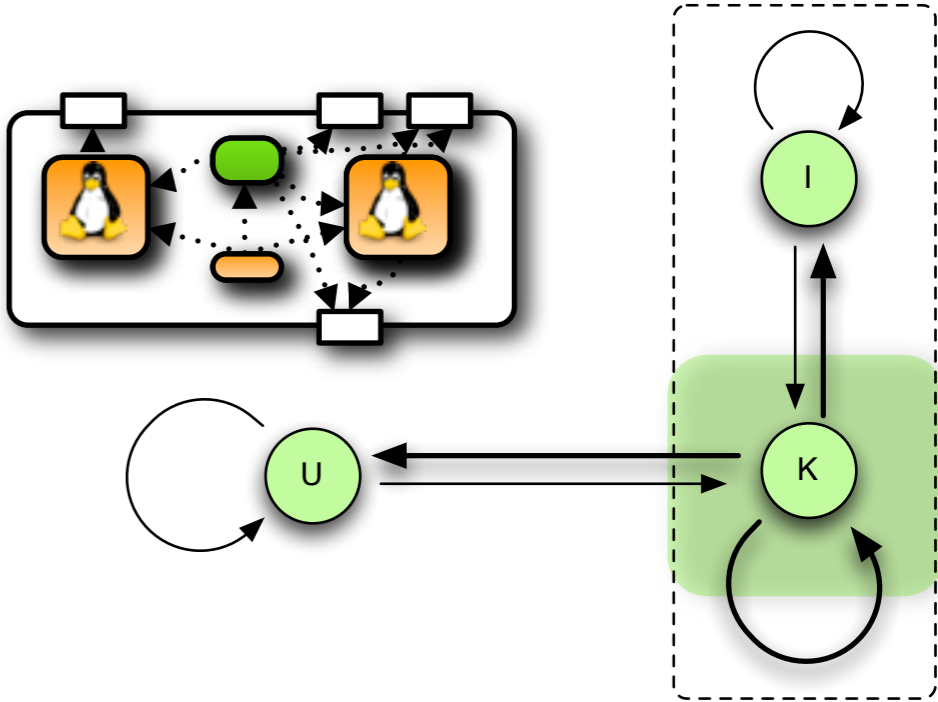


# Proof Architecture





# Execution Trace

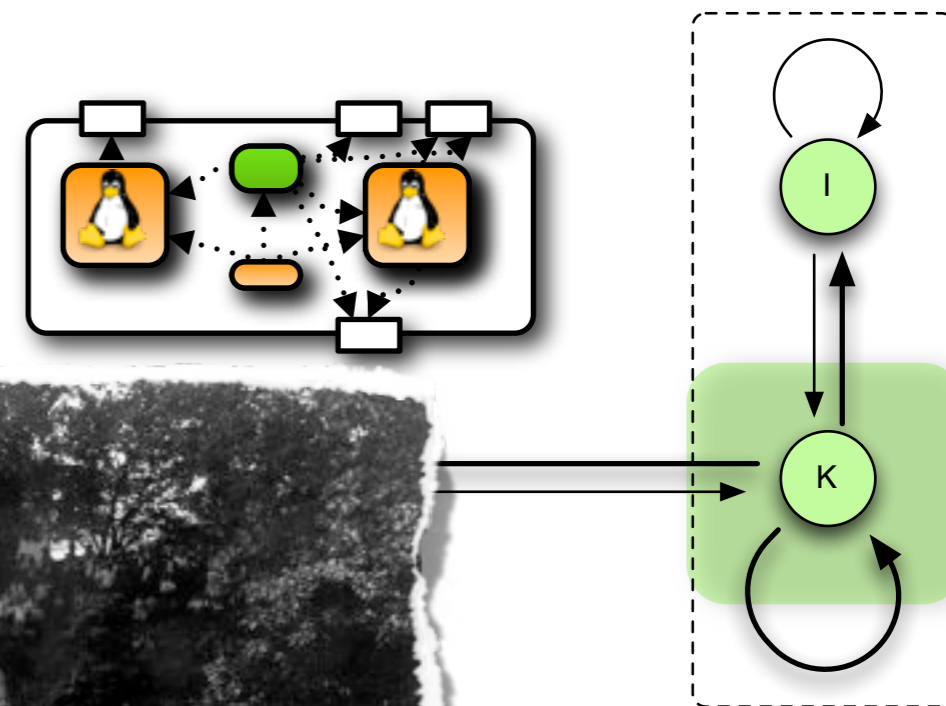
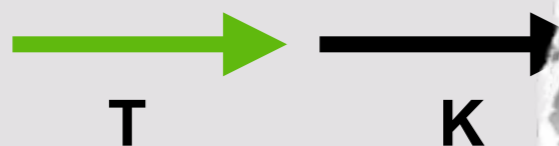


user/kernel trace



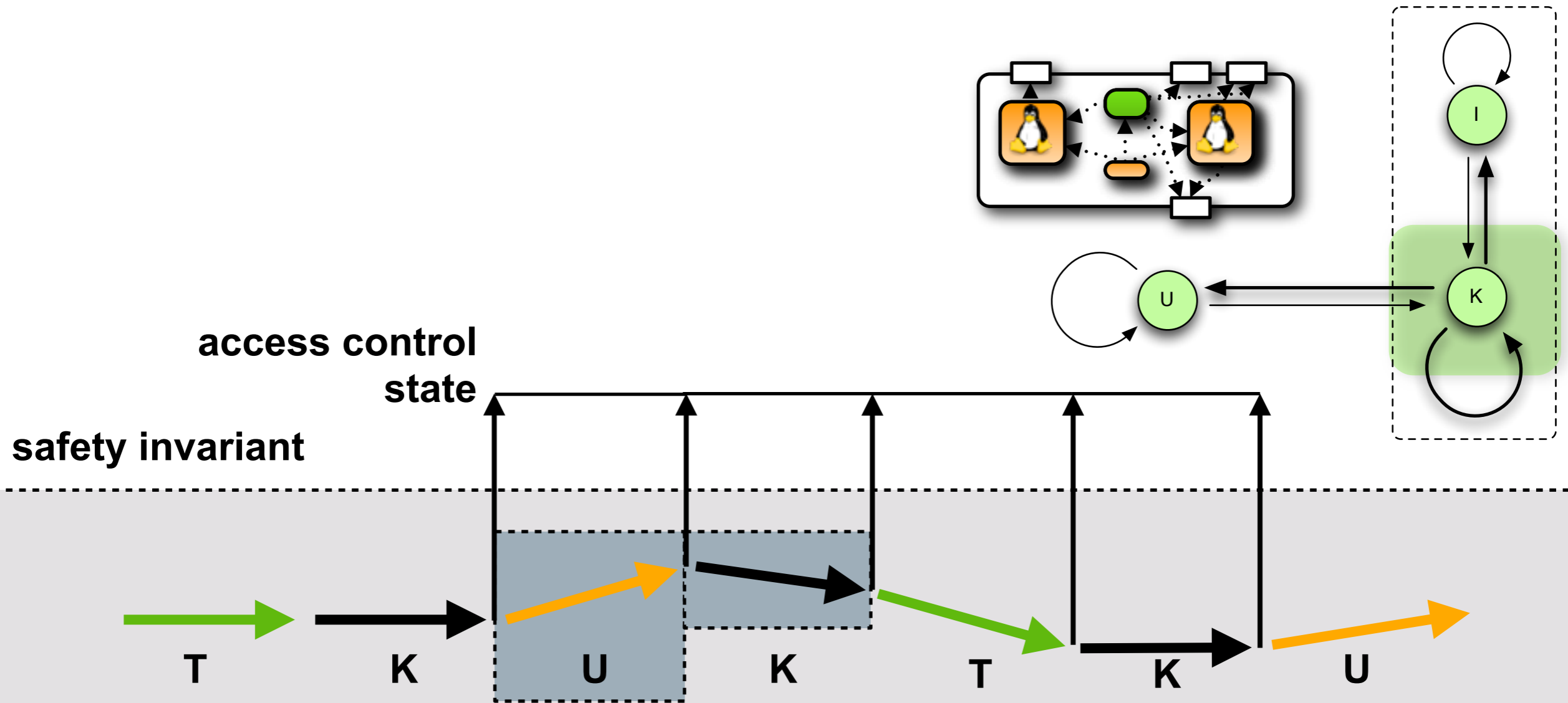
# Safety Invariant

safety invariant



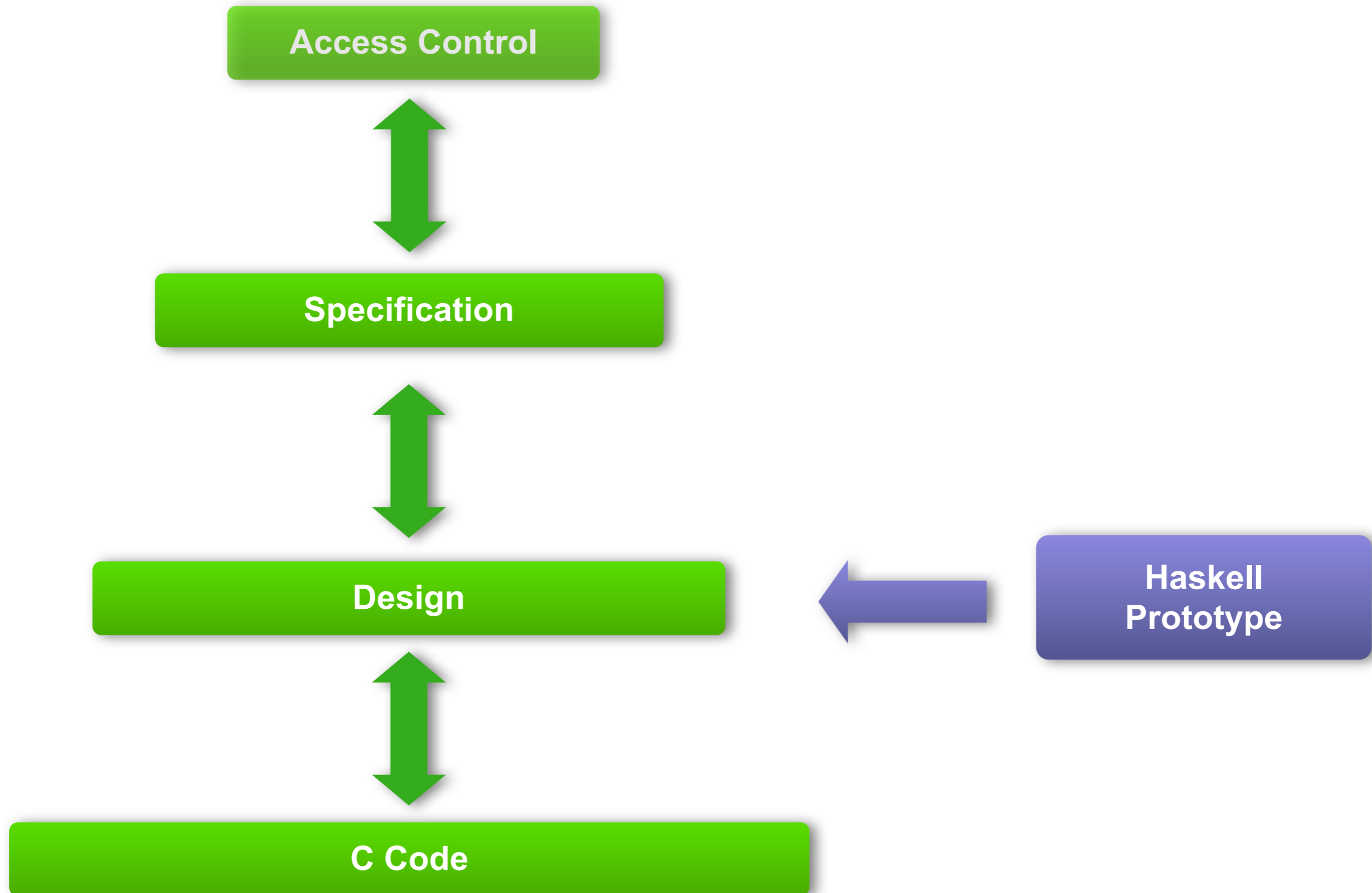
precise behaviour constrained  
precise hardware constrained  
precise knowledge constrained  
unconstrained input  
unconstrained output

# Access Control!

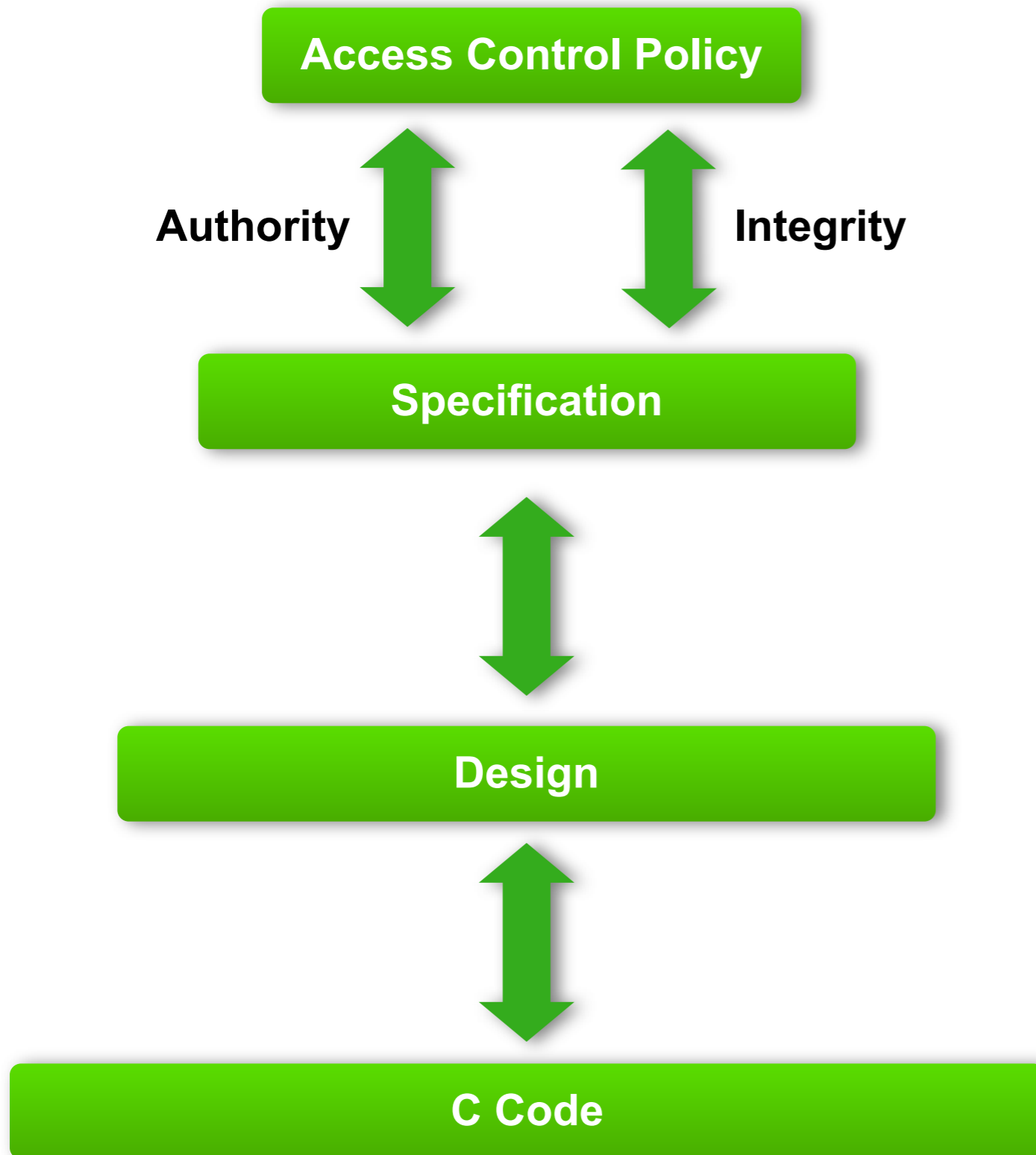


precise behaviour  
 precise behaviour  
 precise behaviour  
 integrity bound

# Proof Architecture



# Proof Architecture



**Authority + Integrity:**  
= trace properties  
preserved by refinement

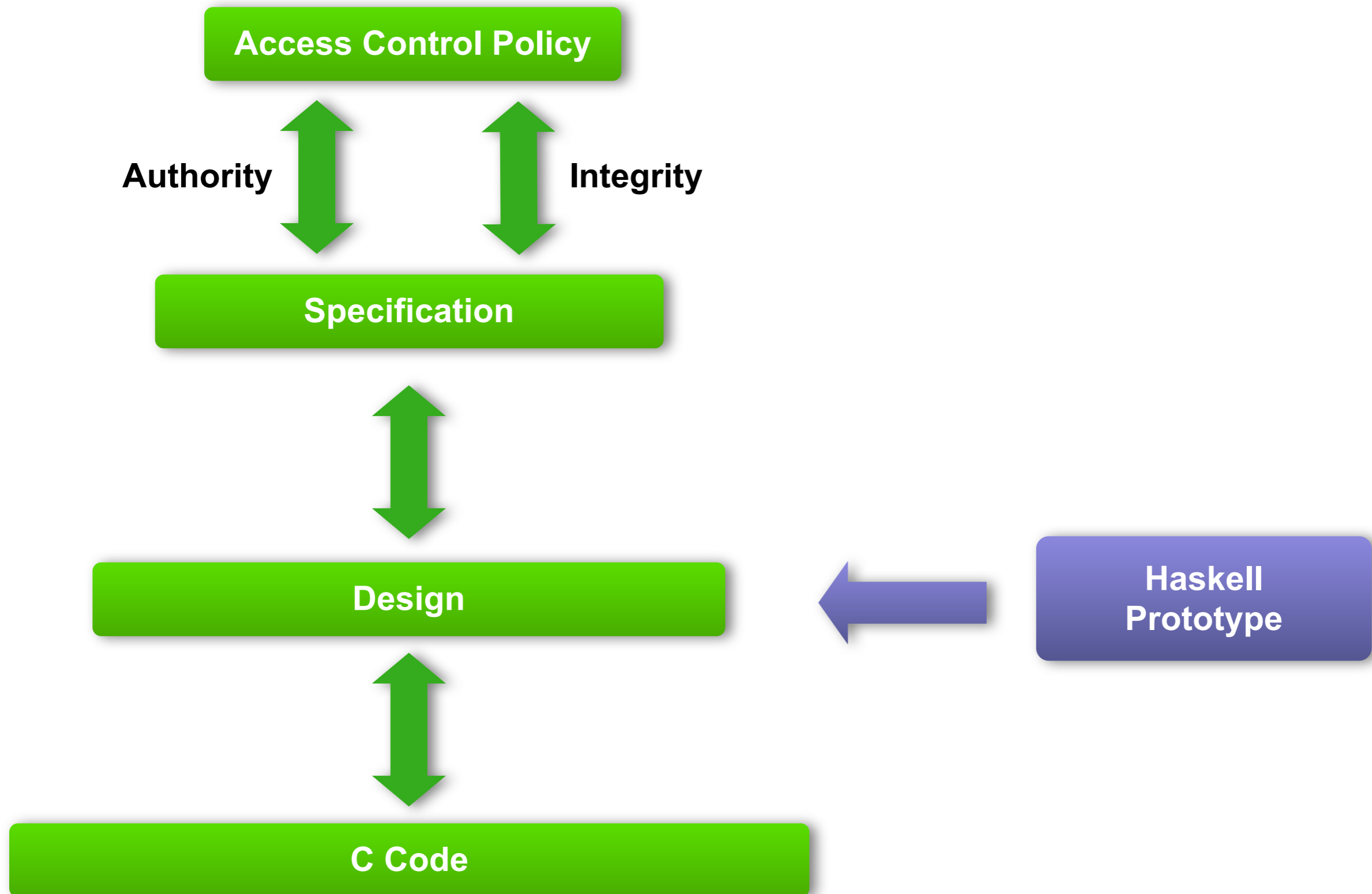
**Confidentiality:**  
= hyperproperty  
refinement needs care

# What about Confidentiality?

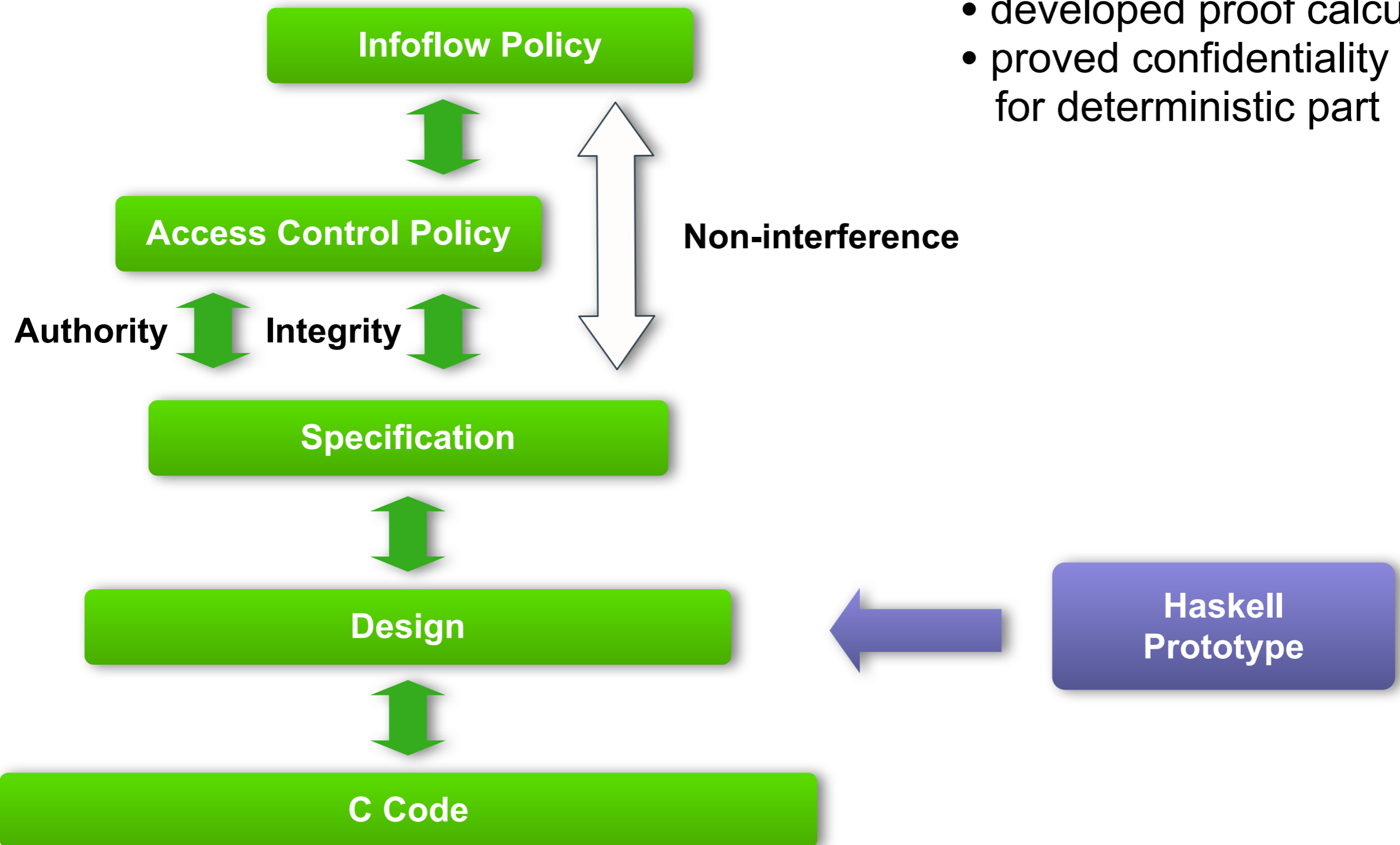
---

- **Harder to prove:**
  - Read not observable in the state
  - Need to compare two executions
  - Standard formulation:
    - non-interference + unwinding conditions
- **Hyperproperty:**
  - Not always preserved by refinement
- **Non-determinism:**
  - Non-determinism could hide “bad” implementation

# Proof Architecture



# Current State

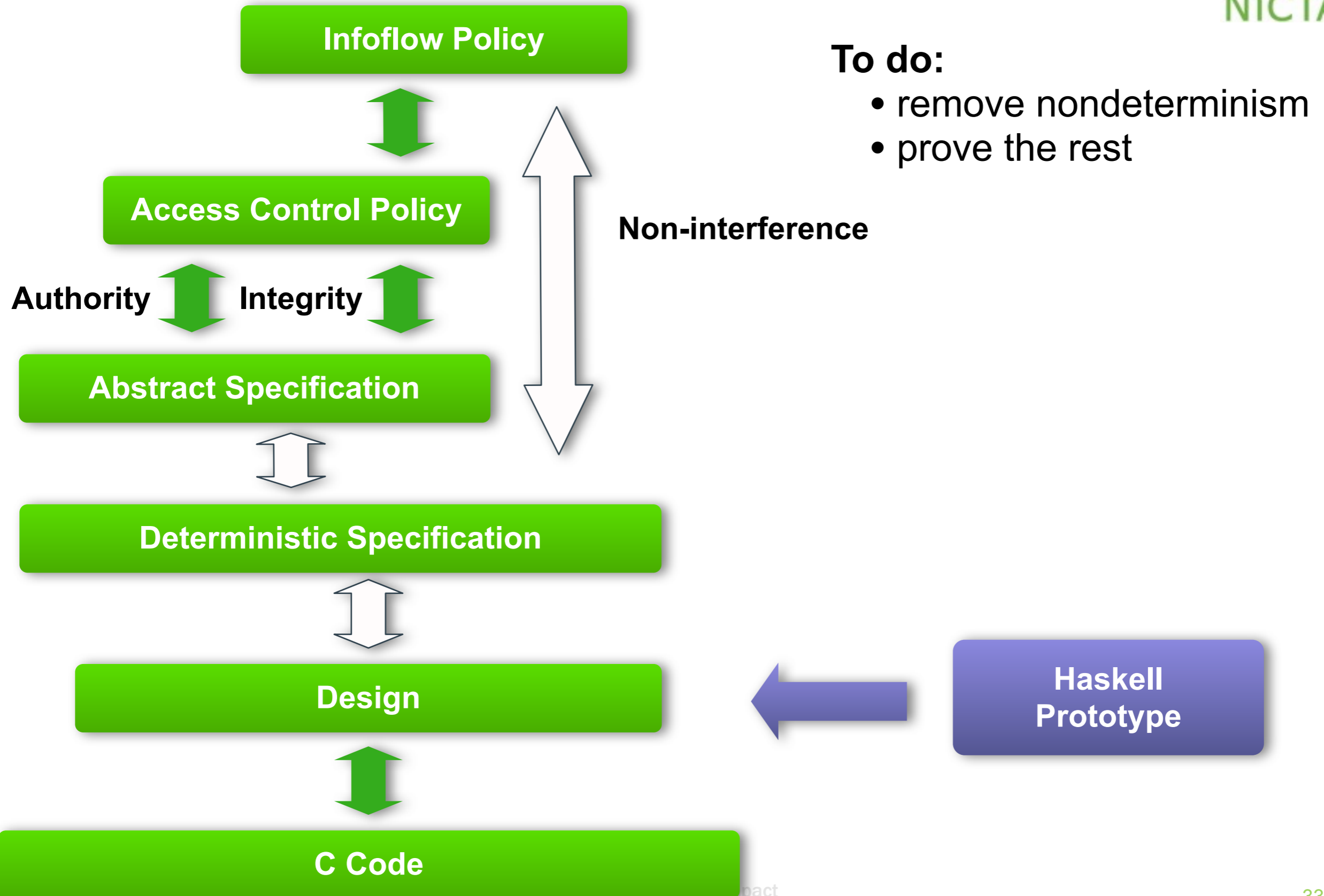


## Current State:

- developed proof calculus
- proved confidentiality for deterministic part



# Current State



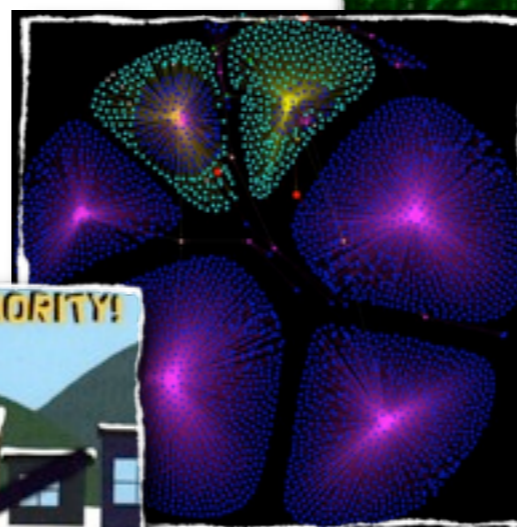
# Plan



**WCET**



**Binary Verification**



**capDL**

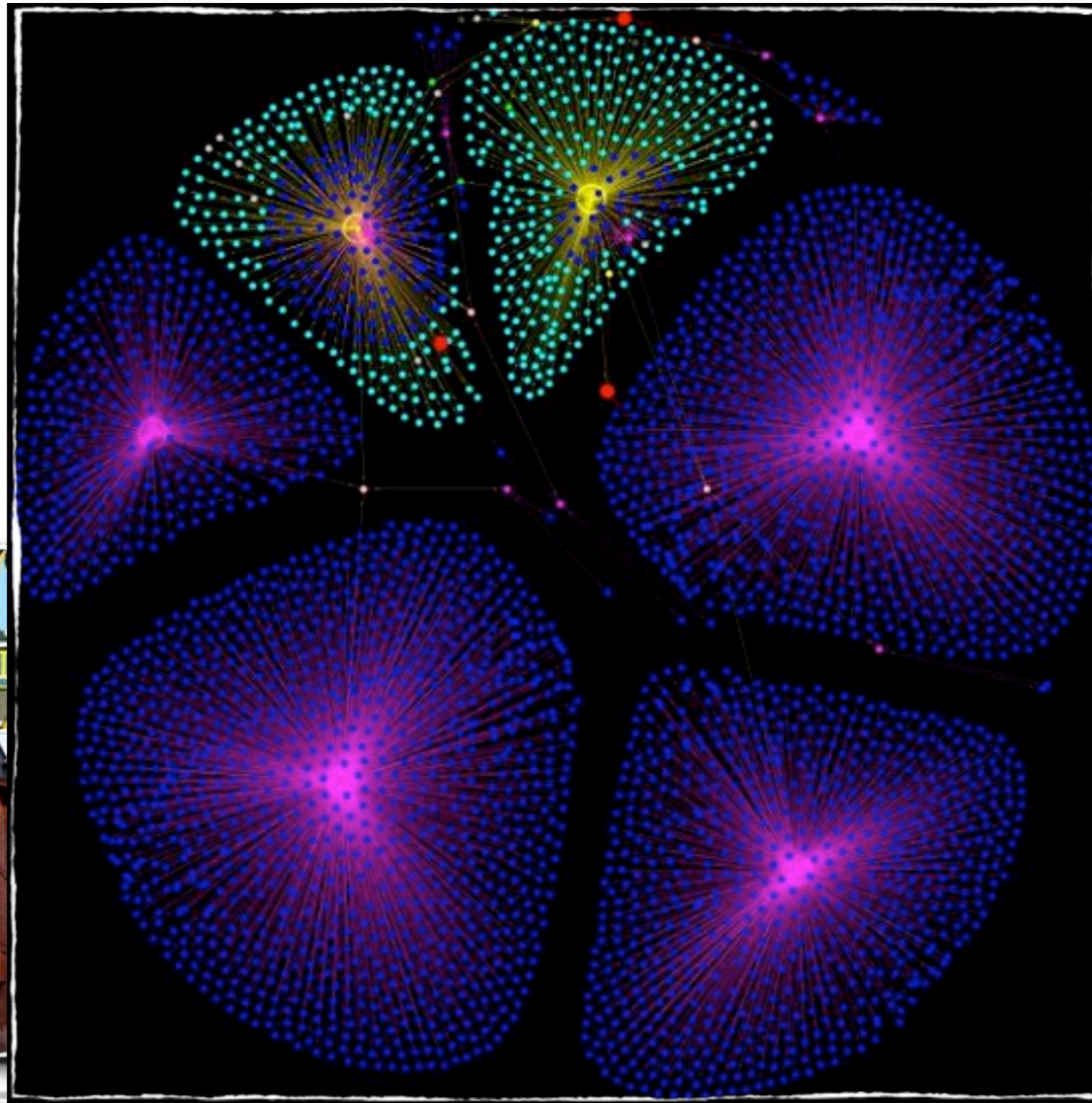


**Integrity & Non-Interference**



**History & Software Process**

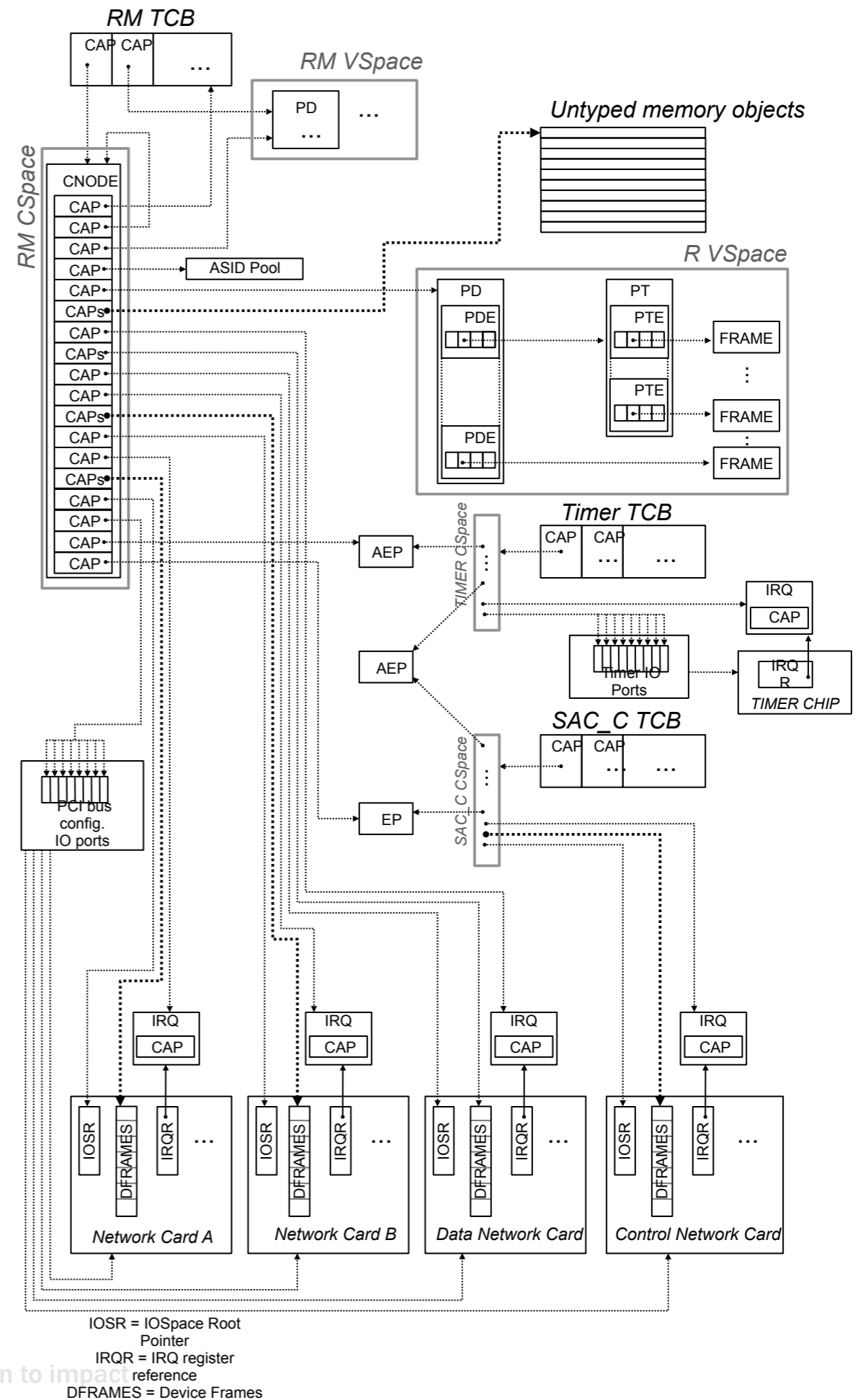
# capDL & Verified System Startup

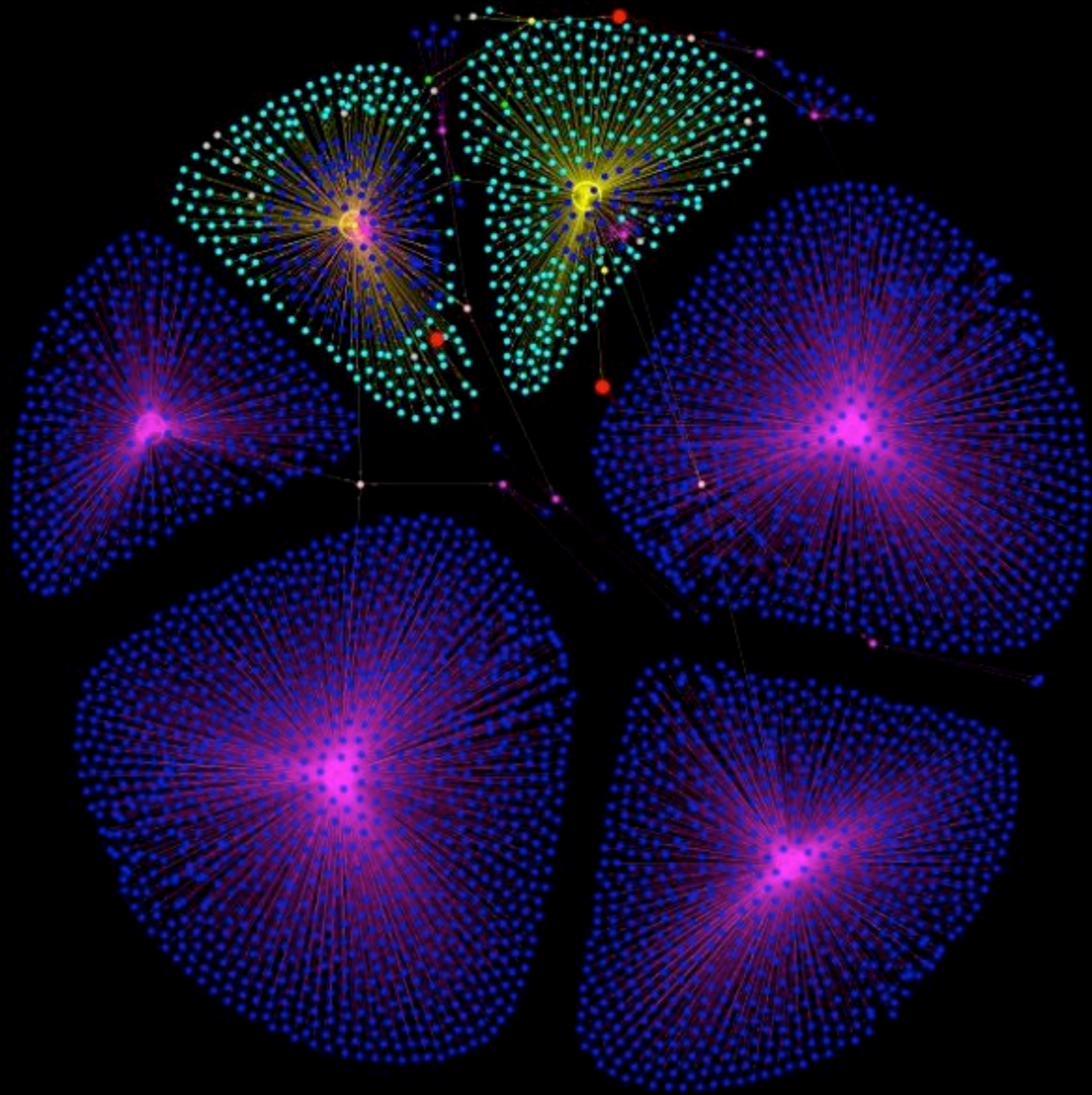


**capDL**

# seL4 cap distribution

## Simplified seL4 caps and objects of example system





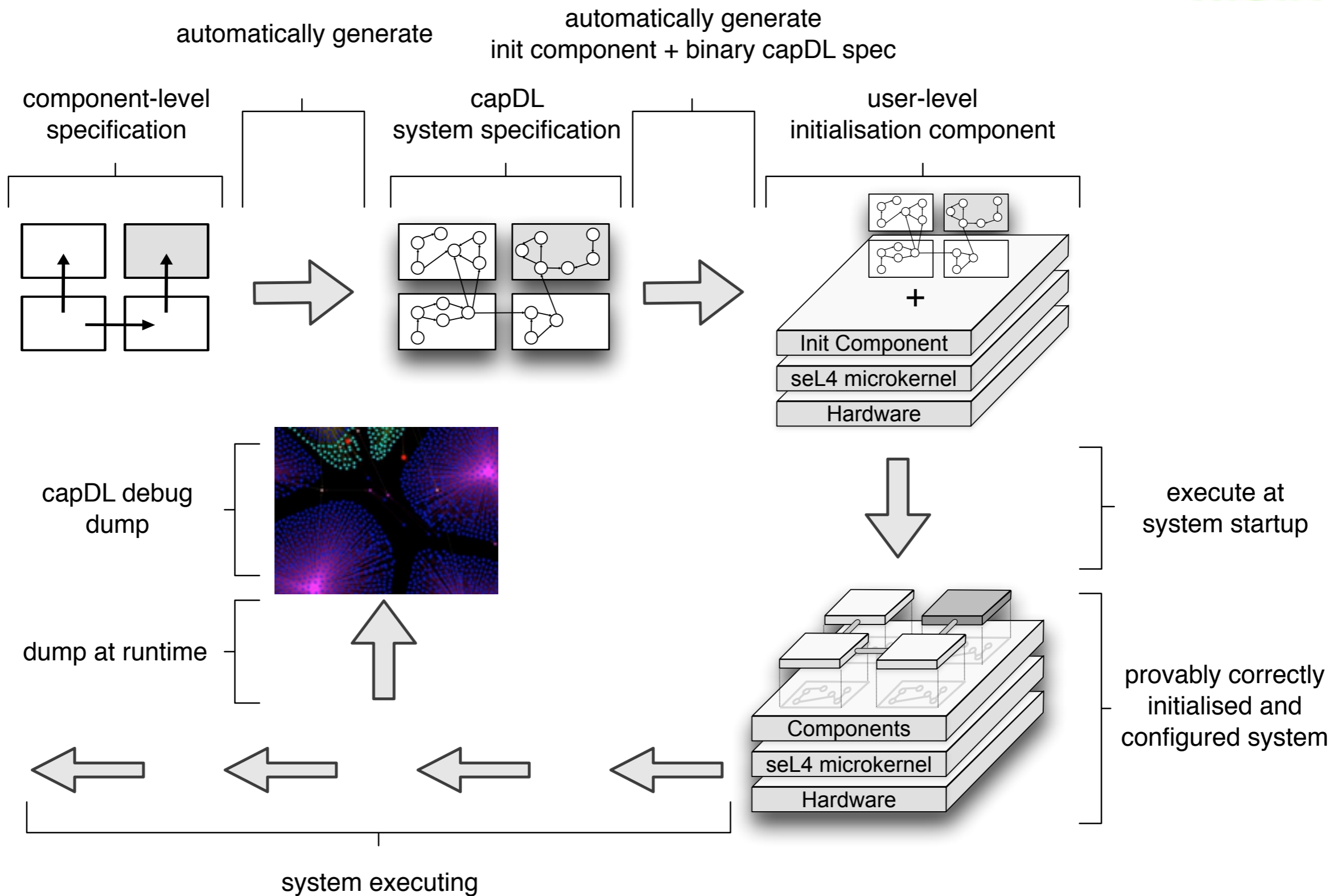
# System Startup

---

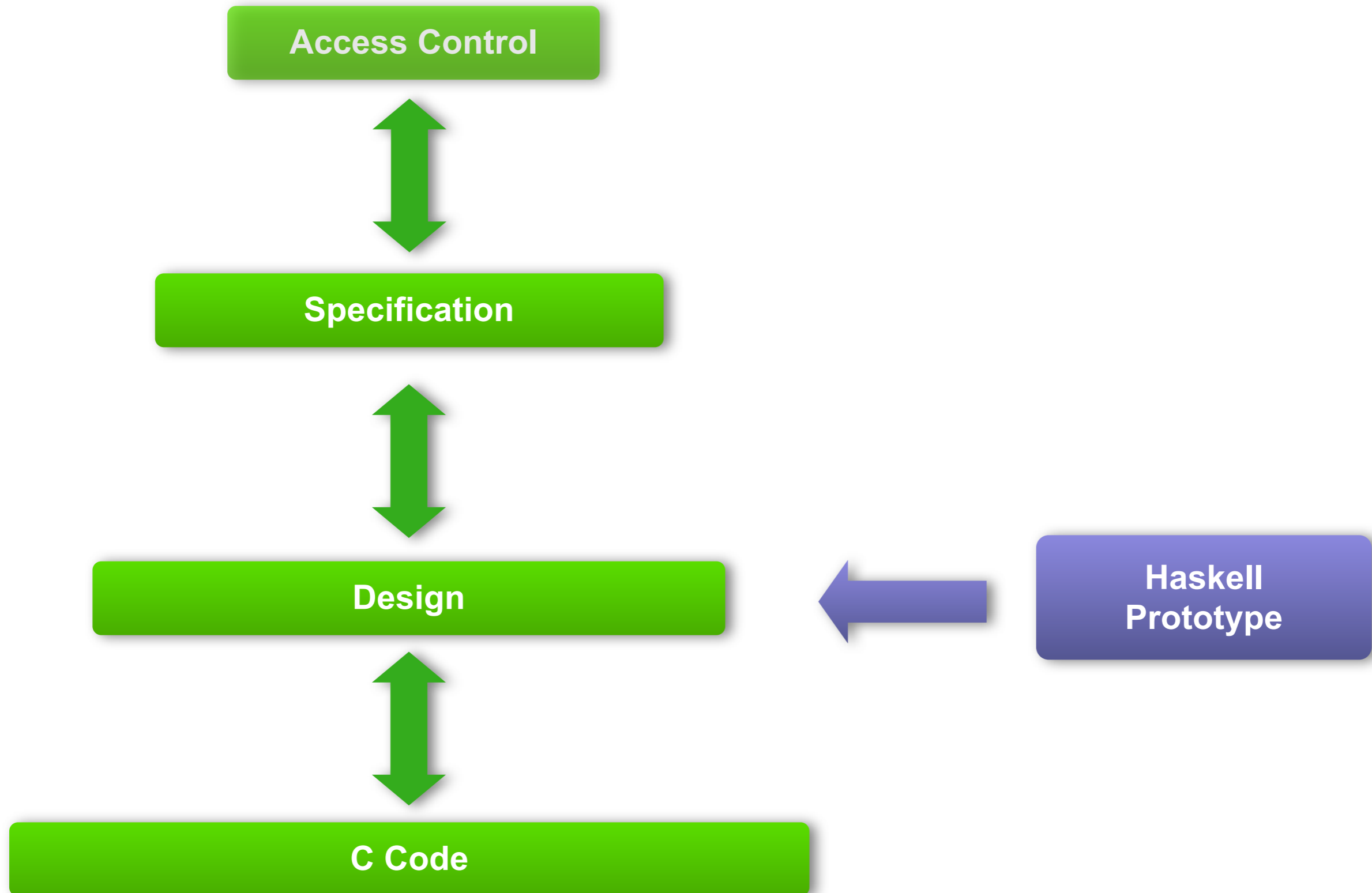


- **Capability distribution determines:**
  - components & system architecture
  - access control & security
- **How do we describe cap distributions?**
  - abstraction of system state
  - capability distribution language: capDL
- **How to get from fresh boot to specific distribution?**

# capDL process

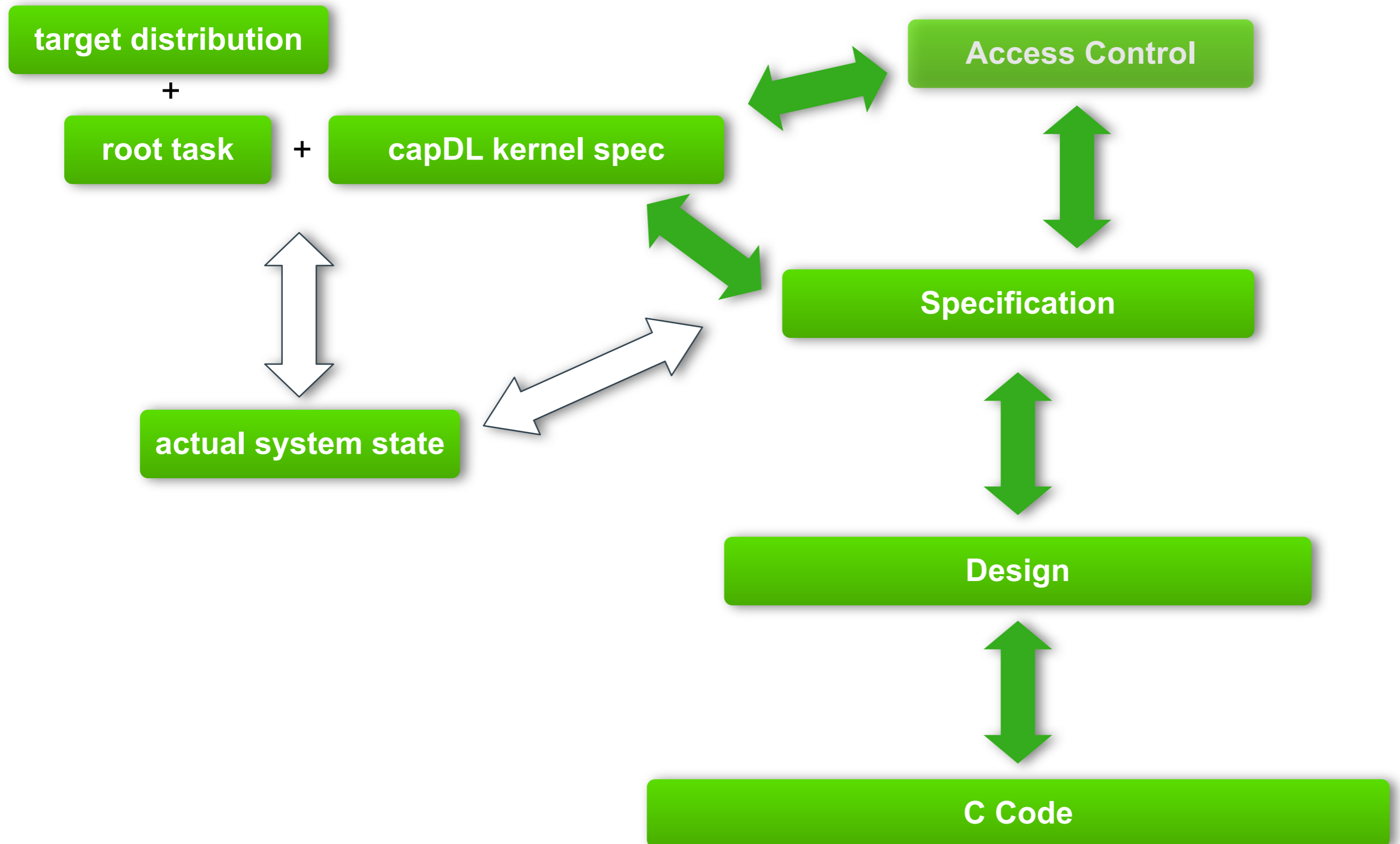


# Proof Architecture





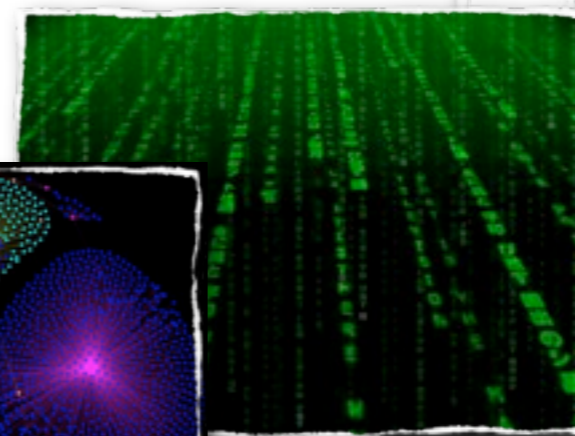
# Proof Architecture



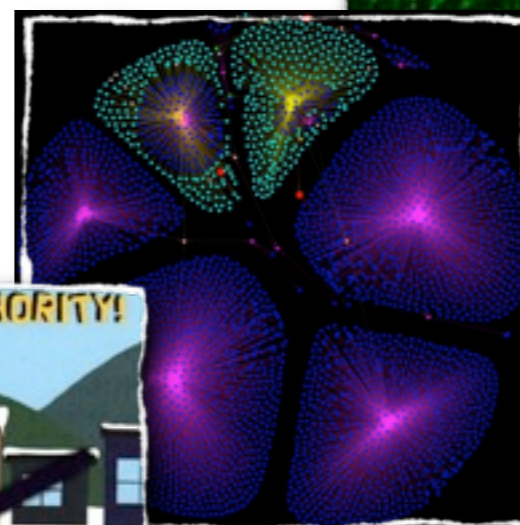
# Plan



**WCET**



**Binary Verification**



**capDL**



**Integrity & Non-Interference**



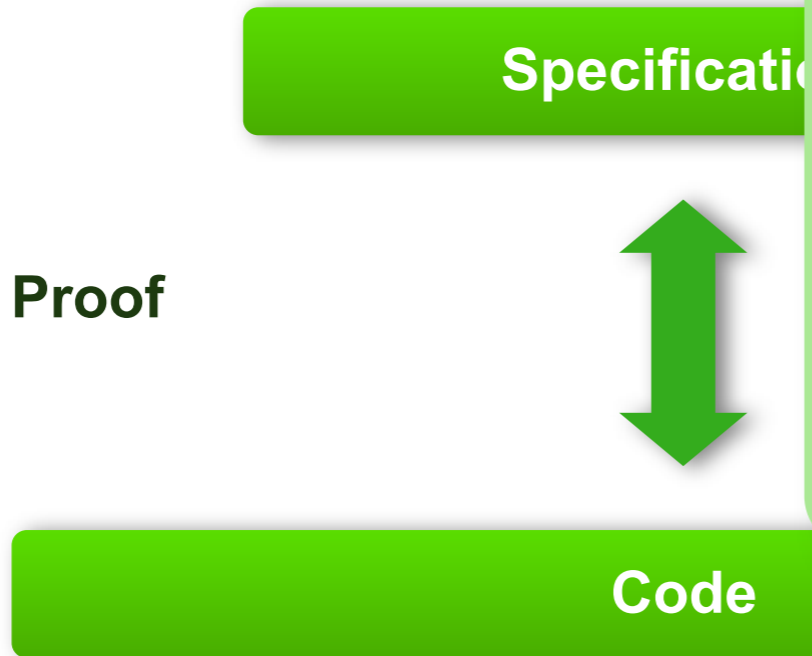
**History & Software Process**

# Binary Verification



# Binary Verification

\*conditions apply



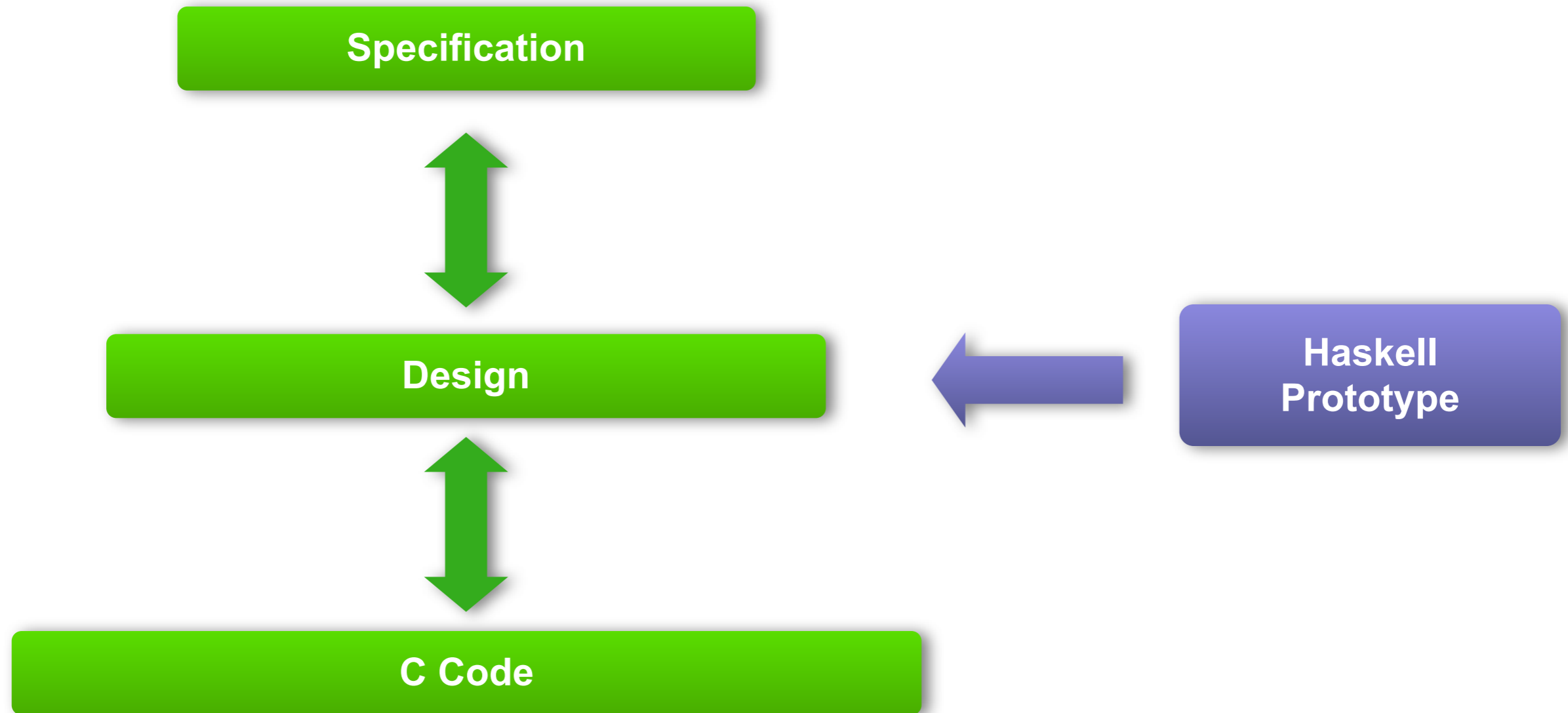
**Assume correct:**

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

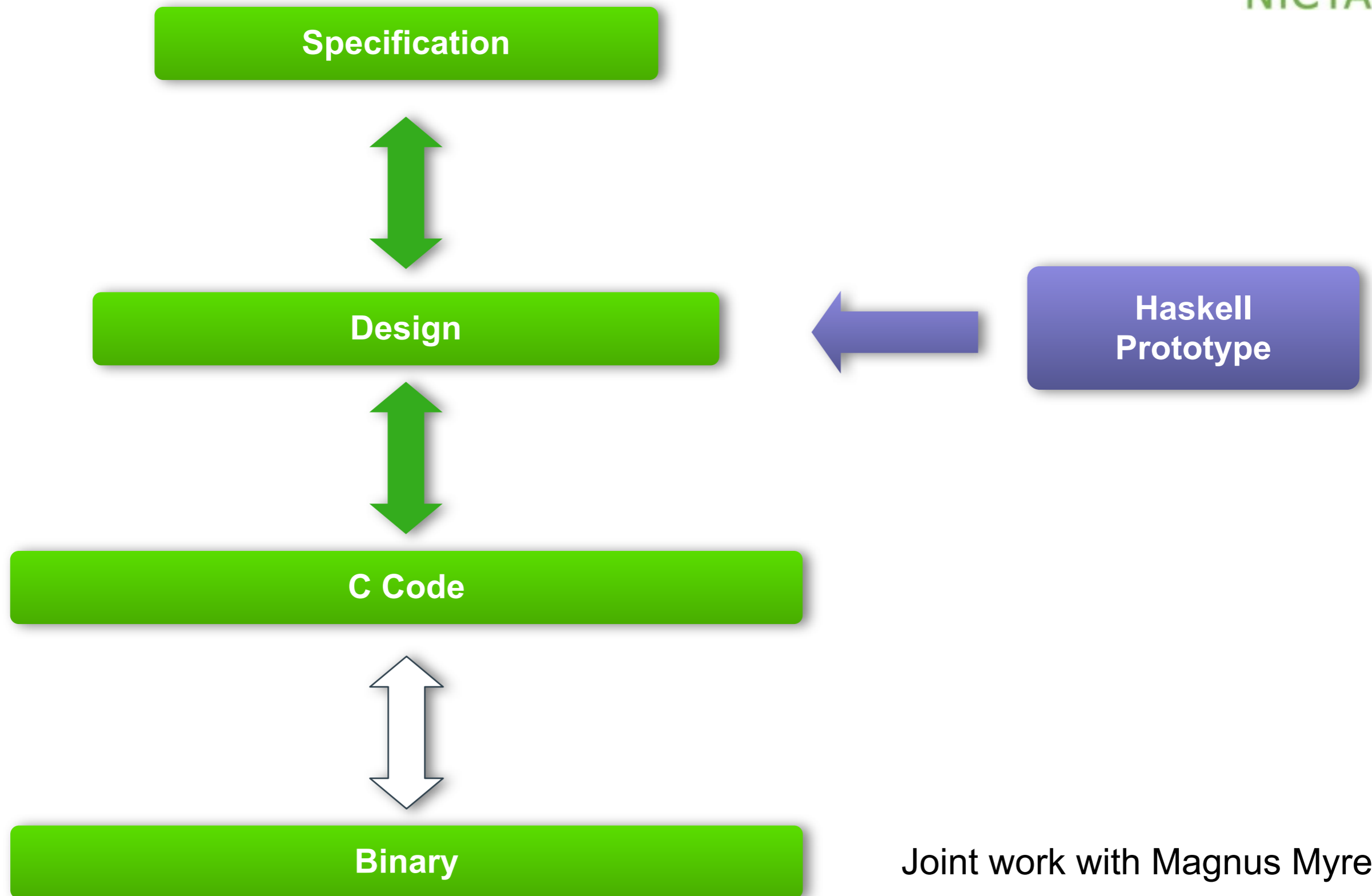
**Assumptions**



# Proof Architecture

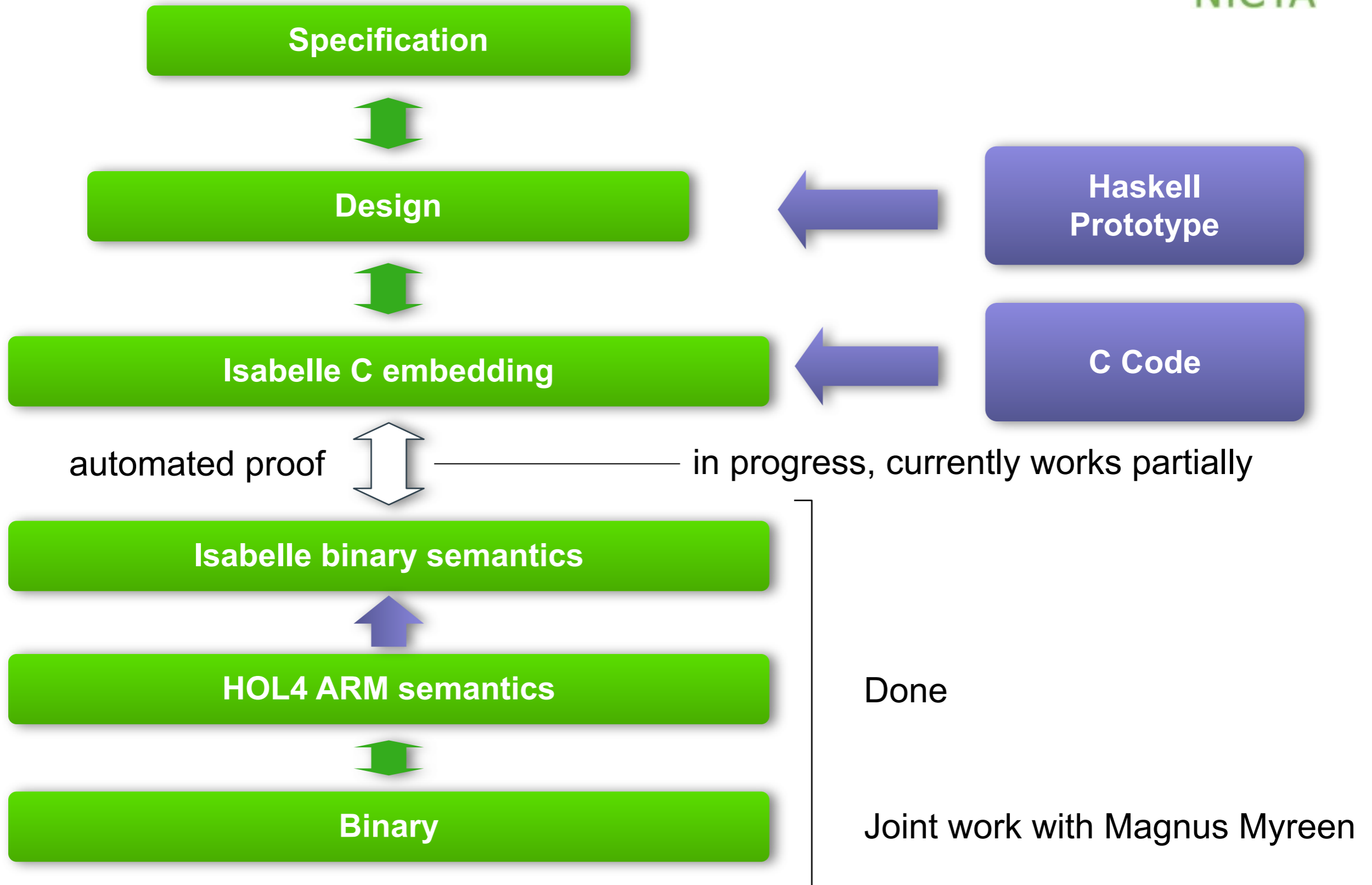


# Binary Verification



Joint work with Magnus Myreen

# Binary Verification



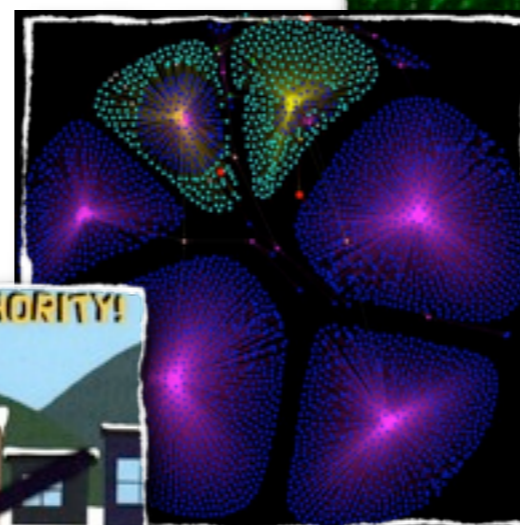
# Plan



**WCET**



**Binary Verification**



**capDL**



**Integrity & Non-Interference**



**History & Software Process**



# WCET



**WCET**

# seL4: Made for Real-World Use

---

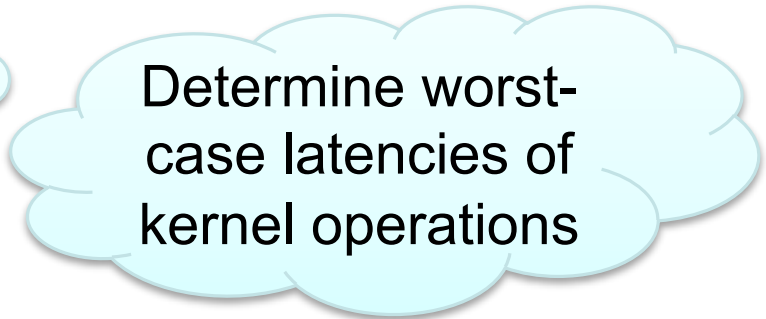


- **Customer product prototypes**
  - Military-grade cross-domain (multi-level secure) devices
  - Safety-critical monitoring devices (mining)
- **RapiLog: Leverage seL4 reliability to improve DBMS performance**
  - driver for virtualization performance, multicore
- **Fiji on seL4: Enable RT programming in HLL (Java)**
  - driver for RT work, potential for verified run time
- **Secure system components: web browser, banking clients**
  - performance, resource-management practicalities
  - remote attestation of critical software (TPM support)
- **Energy management**
  - managing energy as a resource
- **Eat your own dog food (web server, solar racing car)**
  - performance, functionality

# WCET Research Challenges

---

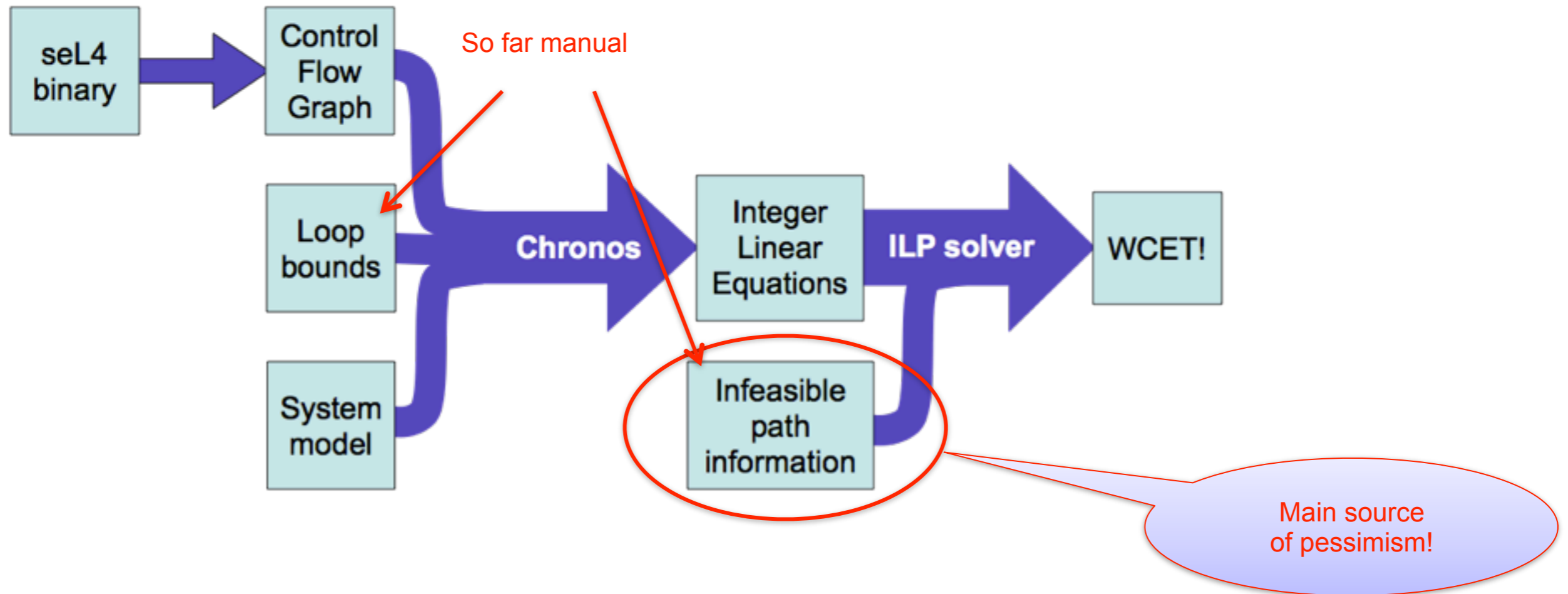
- Sound timing model of kernel



Determine worst-case latencies of kernel operations

- Reasoning about timeliness of apps using kernel mechanisms
  - Scheduling abstractions
  - Extend resource management model to time (capabilities)
  - Whole-system schedulability analysis

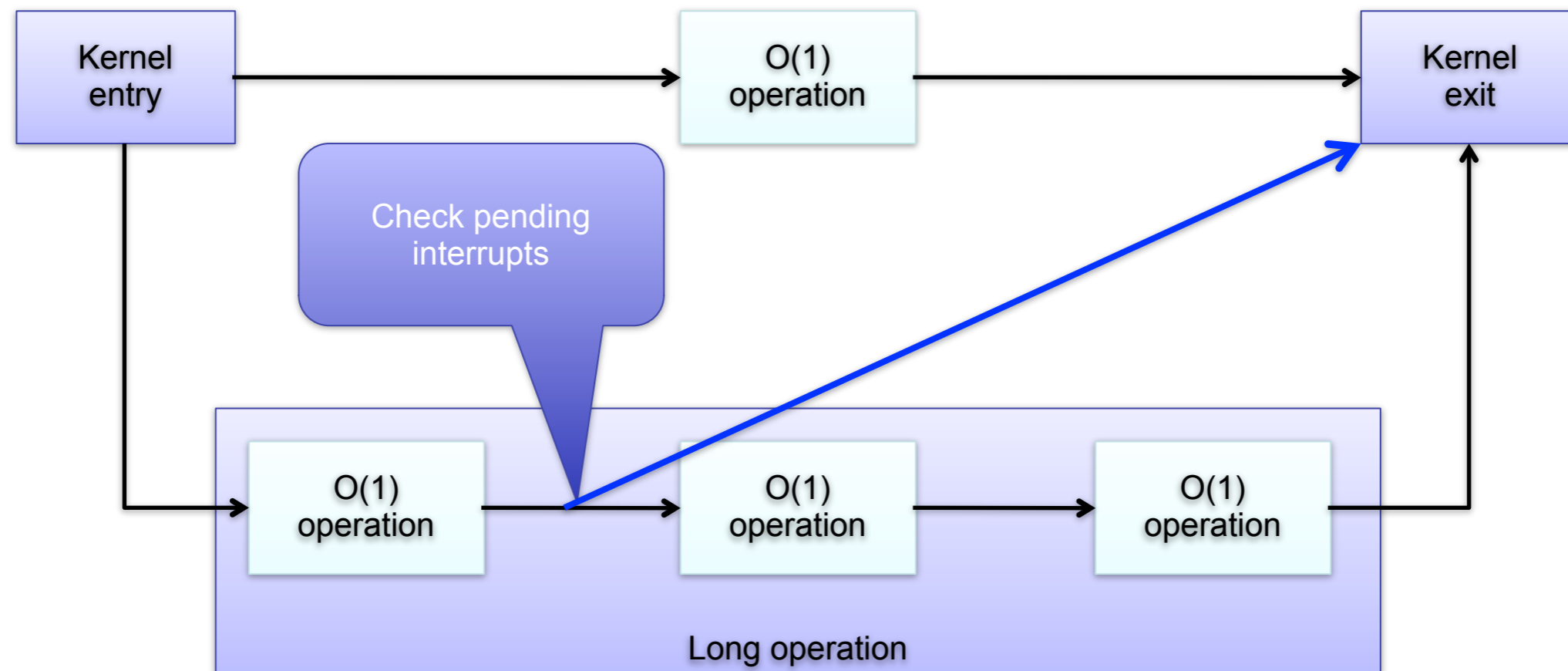
# WCET Analysis Approach



- **Result: WCET >1 sec!**
  - Pessimism of analysis (loop bounds, infeasible paths)
    - Manual elimination of infeasible paths
    - Result: 600 ms :-)

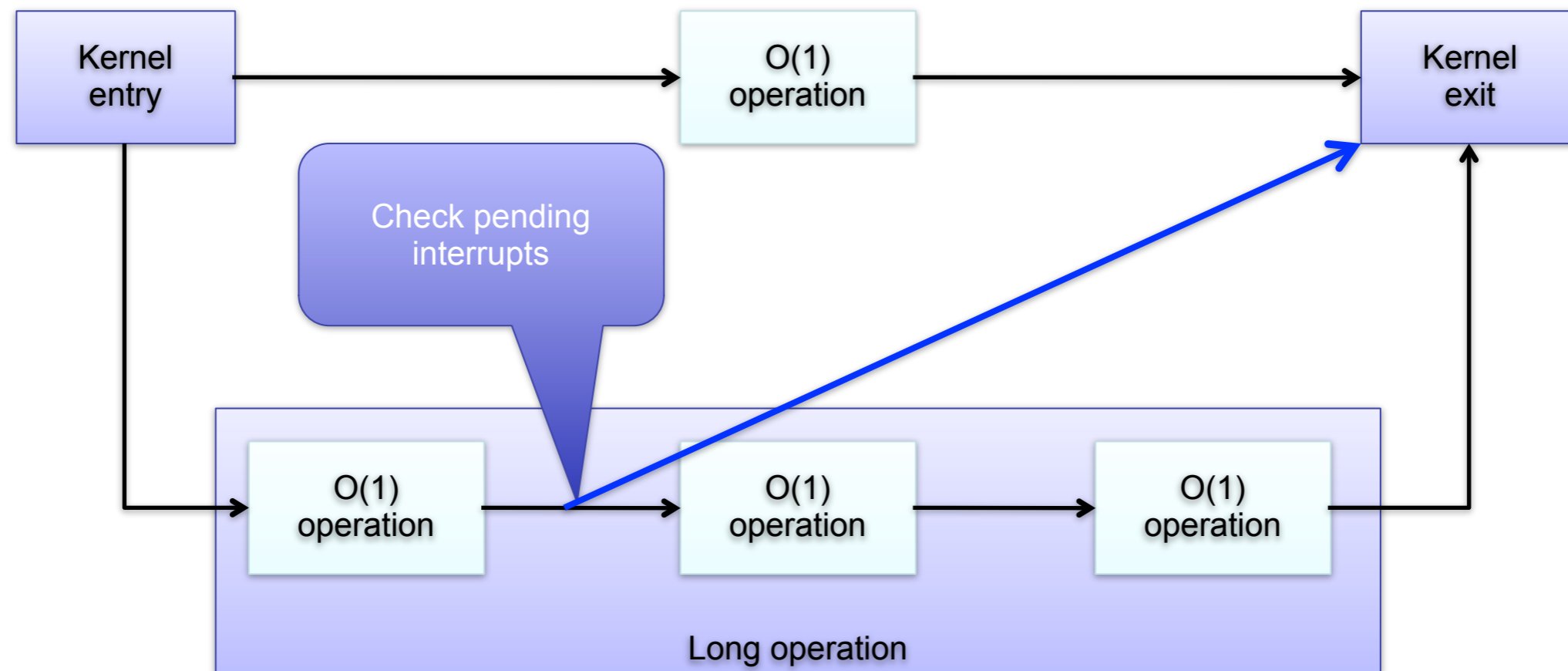
# Improving WCET

- **Challenge: Improving WCET while**
  - retaining ability to verify
  - maintaining high average-case performance
- **seL4 is an event-oriented kernel running with interrupts disabled**

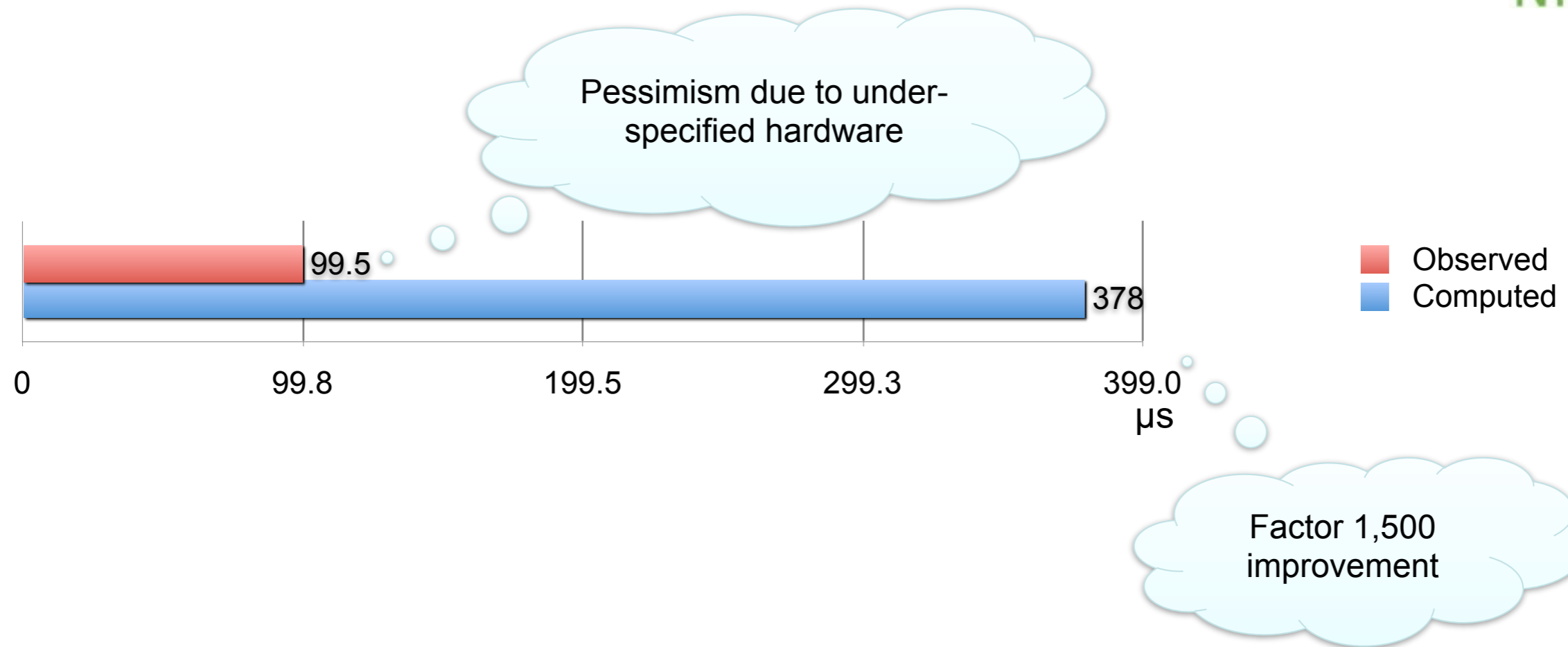


# Placing Preemption Points

- Enabled by design pattern of “incremental consistency”:
  - Large composite objects can be constructed (or deconstructed) from individual components
  - Each component can be added/removed in  $O(1)$  time
  - Intermediate states are consistent

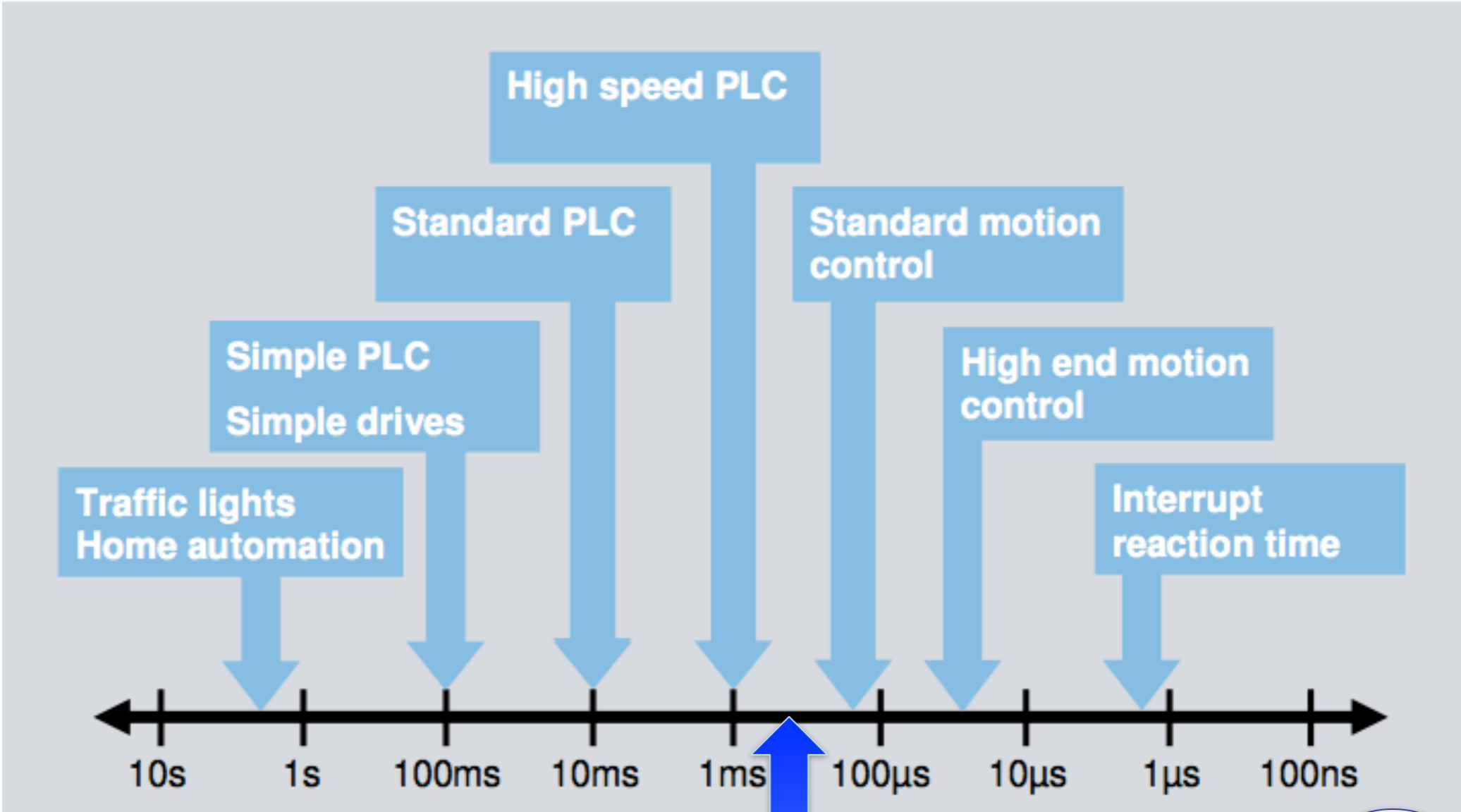


# Result



- Verification of modifications will be mostly routine
- Also now mostly automated:
  - loop counts
  - infeasible path elimination

# RT in Industrial Automation



© Siemens AG 2011. All Rights Reserved

Page 8

2011-11-14

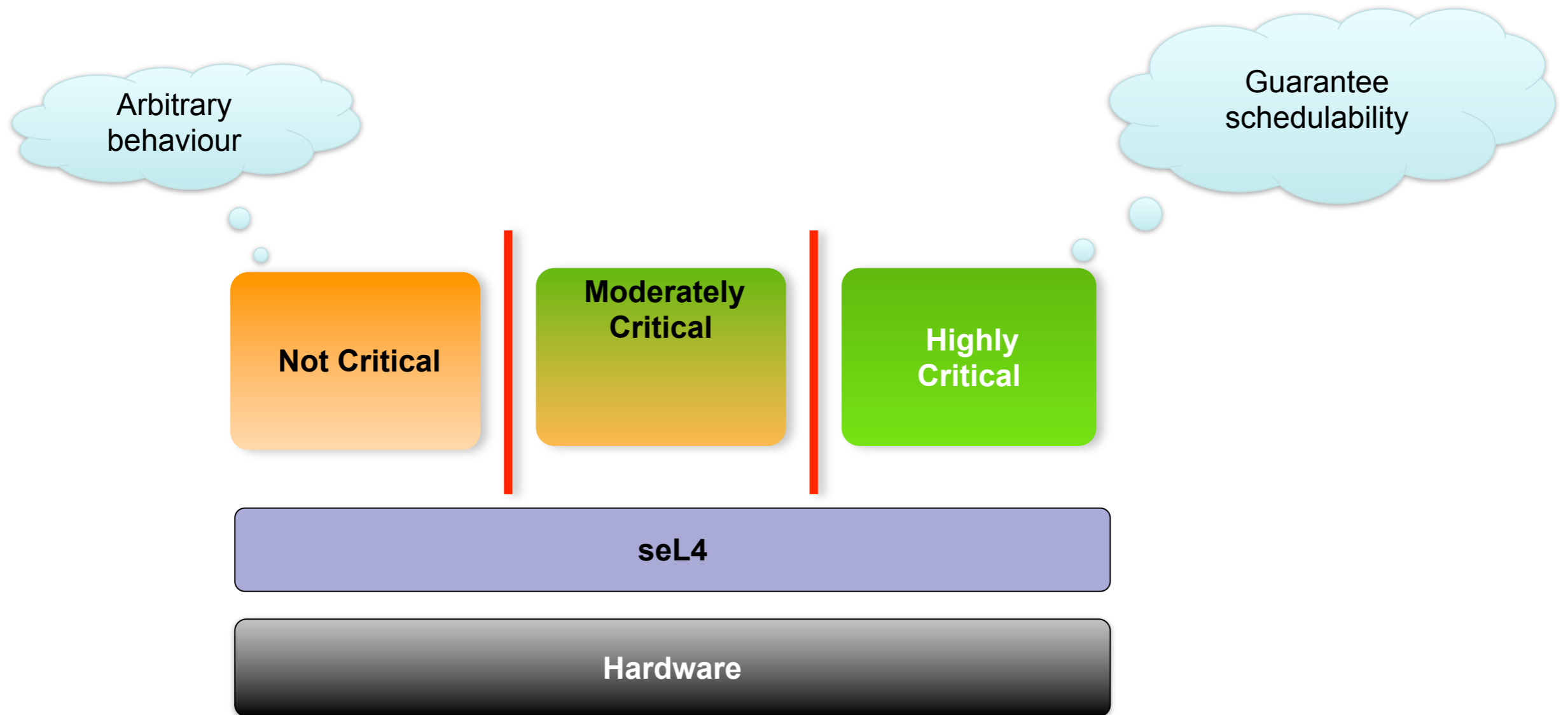
seL4 today



First protected RTOS with sound WCET analysis!



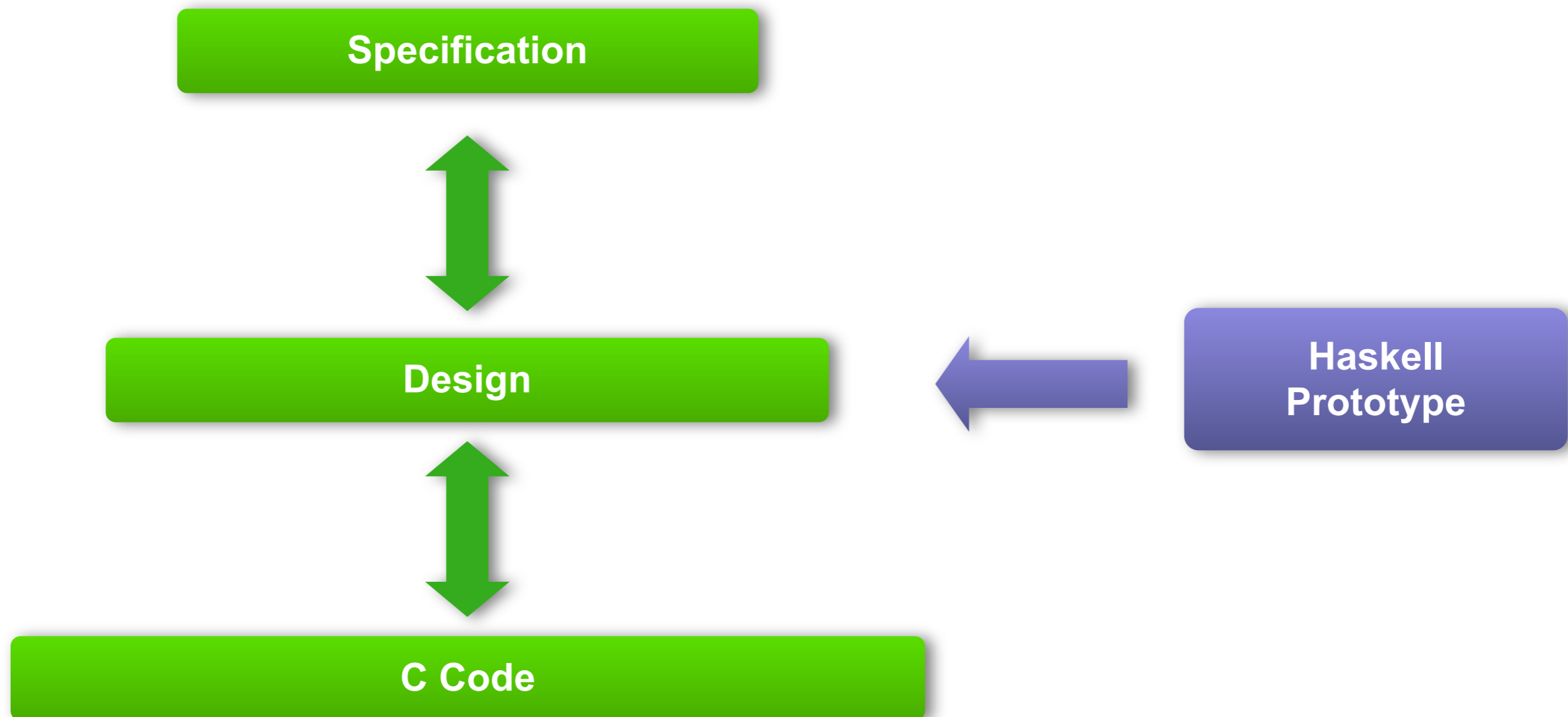
# Future: Whole-System Schedulability



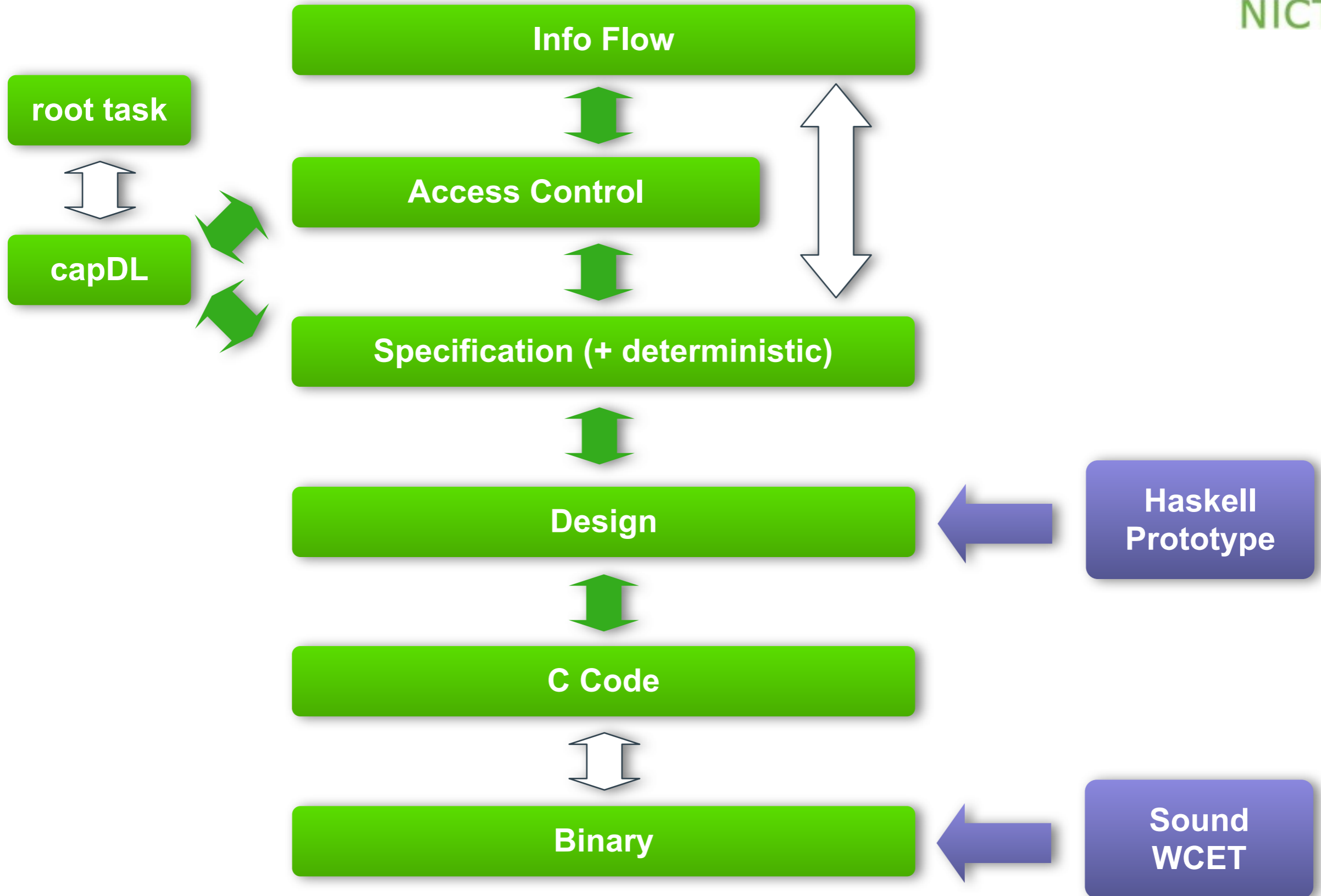
# Summary



# Proof Architecture



# 3 Years Later (simplified)





# Thank You

Google

