NICTA

Towards A Platform for Secure Systems

Gernot Heiser

NICTA and University of New South Wales

Sydney, Australia

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

NSW GOVERNMENT

State Government Victoria

THE UNIVERSITY OF MELBOURNE

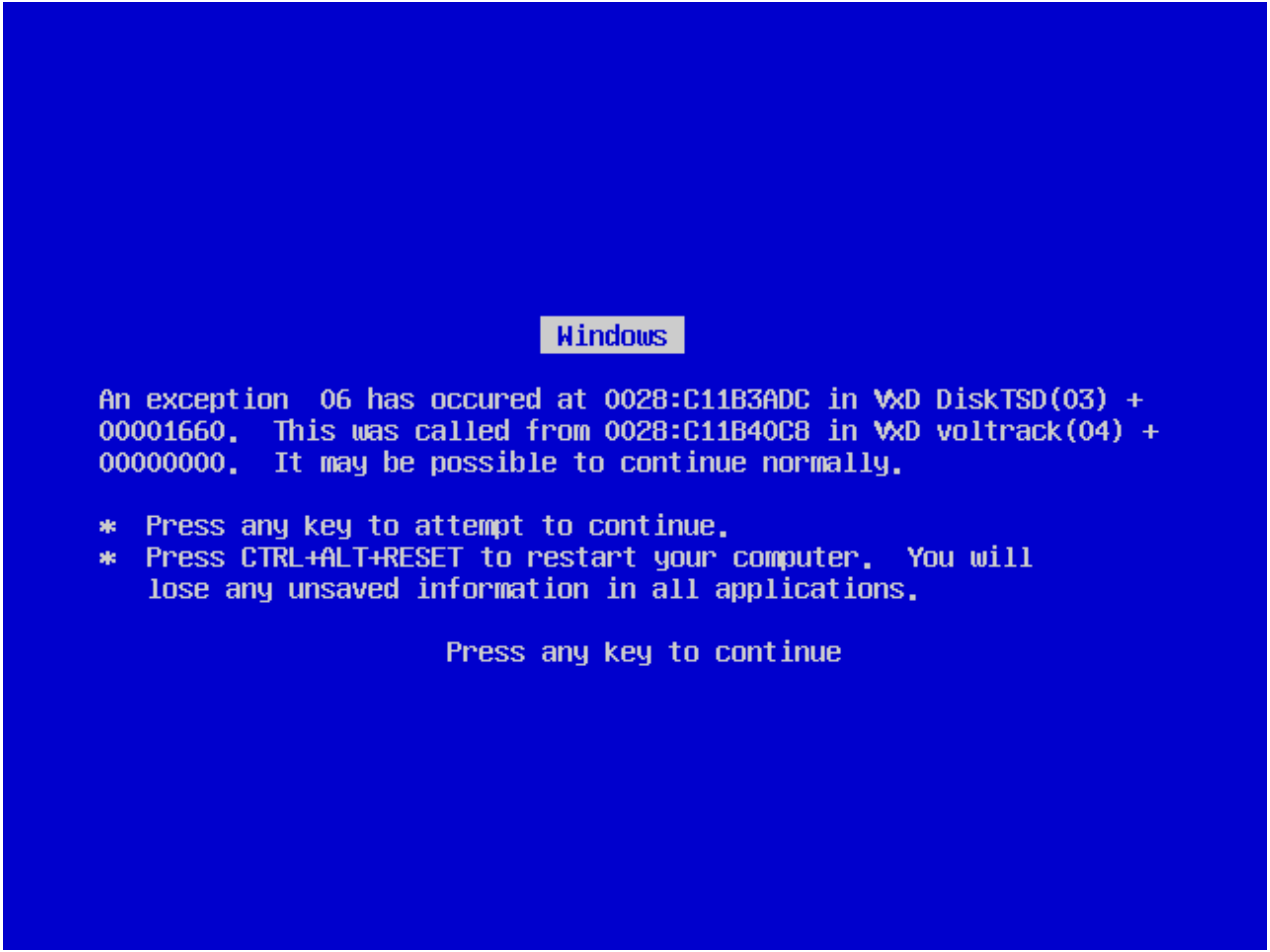
THE UNIVERSITY OF SYDNEY

Queensland Government

Griffith UNIVERSITY

QUT Queensland University of Technology

THE UNIVERSITY OF QUEENSLAND AUSTRALIA



```
                              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
```

Present Systems are *NOT* Secure!

What’s Next?

So, why don't we prove security?

Claim:

A system must be considered *insecure* unless *proved* otherwise!

Corollary [with apologies to Dijkstra]:
Testing, code inspection, etc. can only show *insecurity*, not security!

Core Issue: Complexity

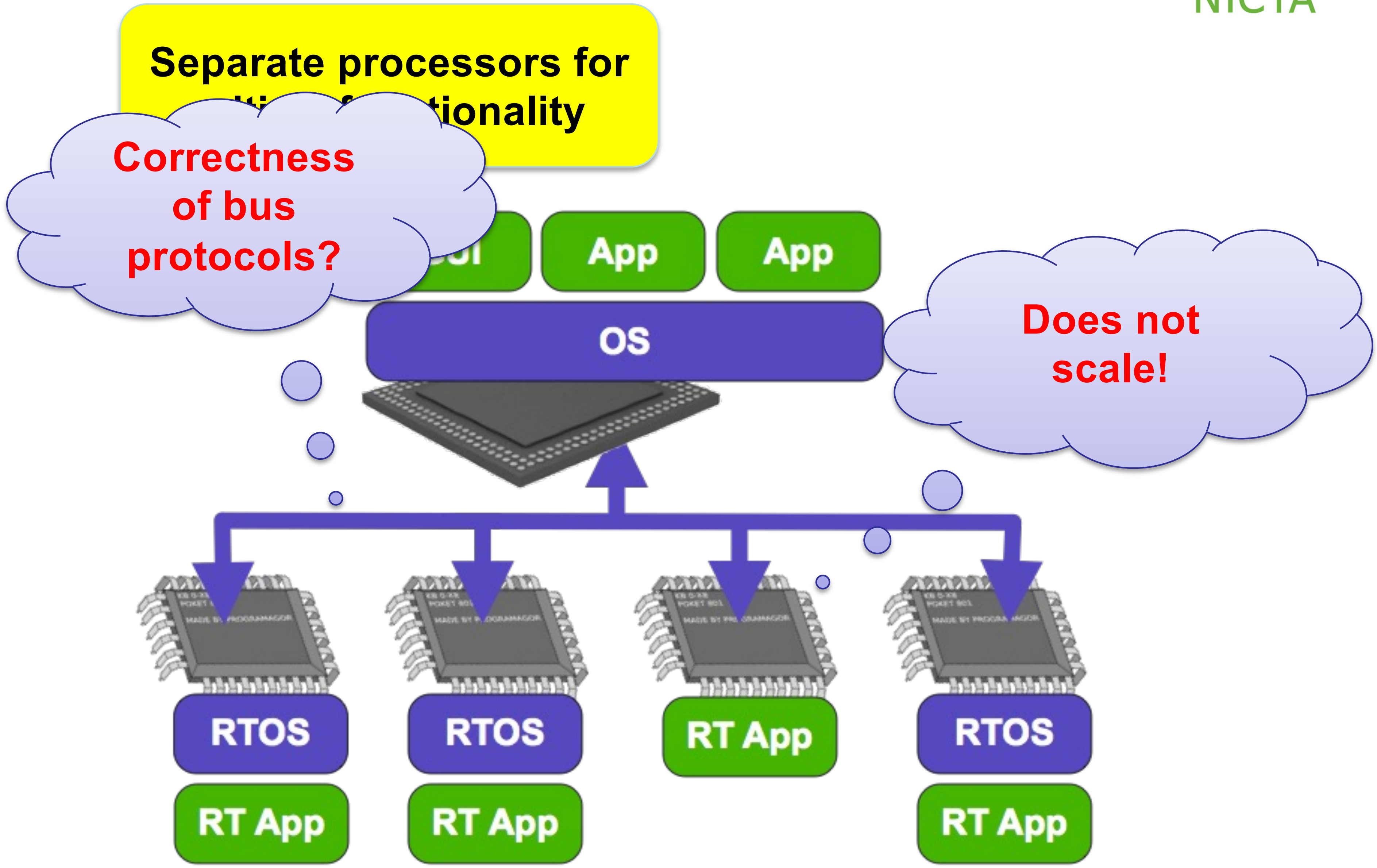
- Massive functionality of C devices
  ⇒ huge software stacks
  - How secure are your paym ?

- Increasing usability requ
  - Wearable or implante
  - Patient-operated
  - GUIs next to life-c

- On-going integration o s
  - Automotive infotainment an
  - Gigabytes of software on 100 CPUs…

Systems far too complex to prove their security!

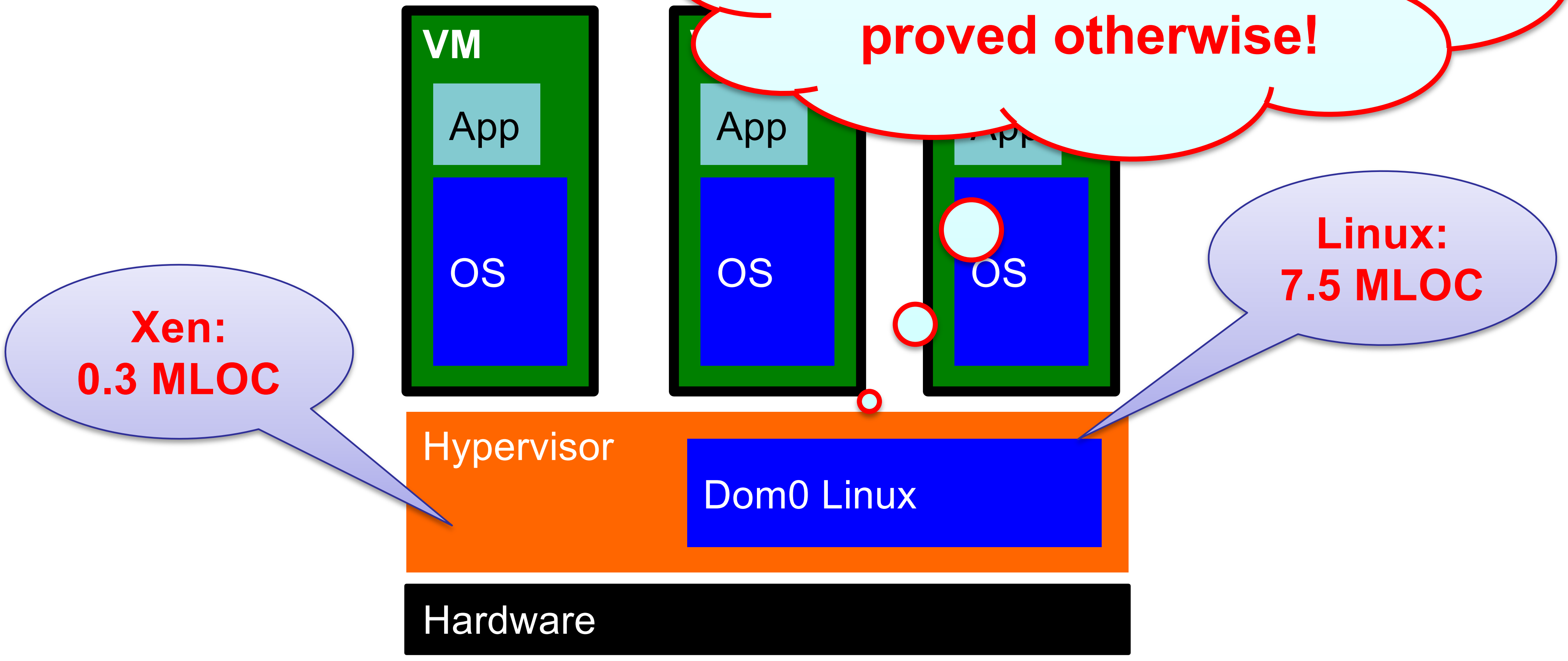
Dealing with Complexity: Physical Isolation


Separate processors for critical functionality
Correctness of bus protocols?
GUI
App
App
OS
Does not scale!
RTOS
RT App
RTOS
RT App
RT App
RTOS
RT App

How About Logical Isolation?

Shared processor with software isolation

Remember: A system is *insecure* unless proved otherwise!



Our Vision: Trustworthy Systems

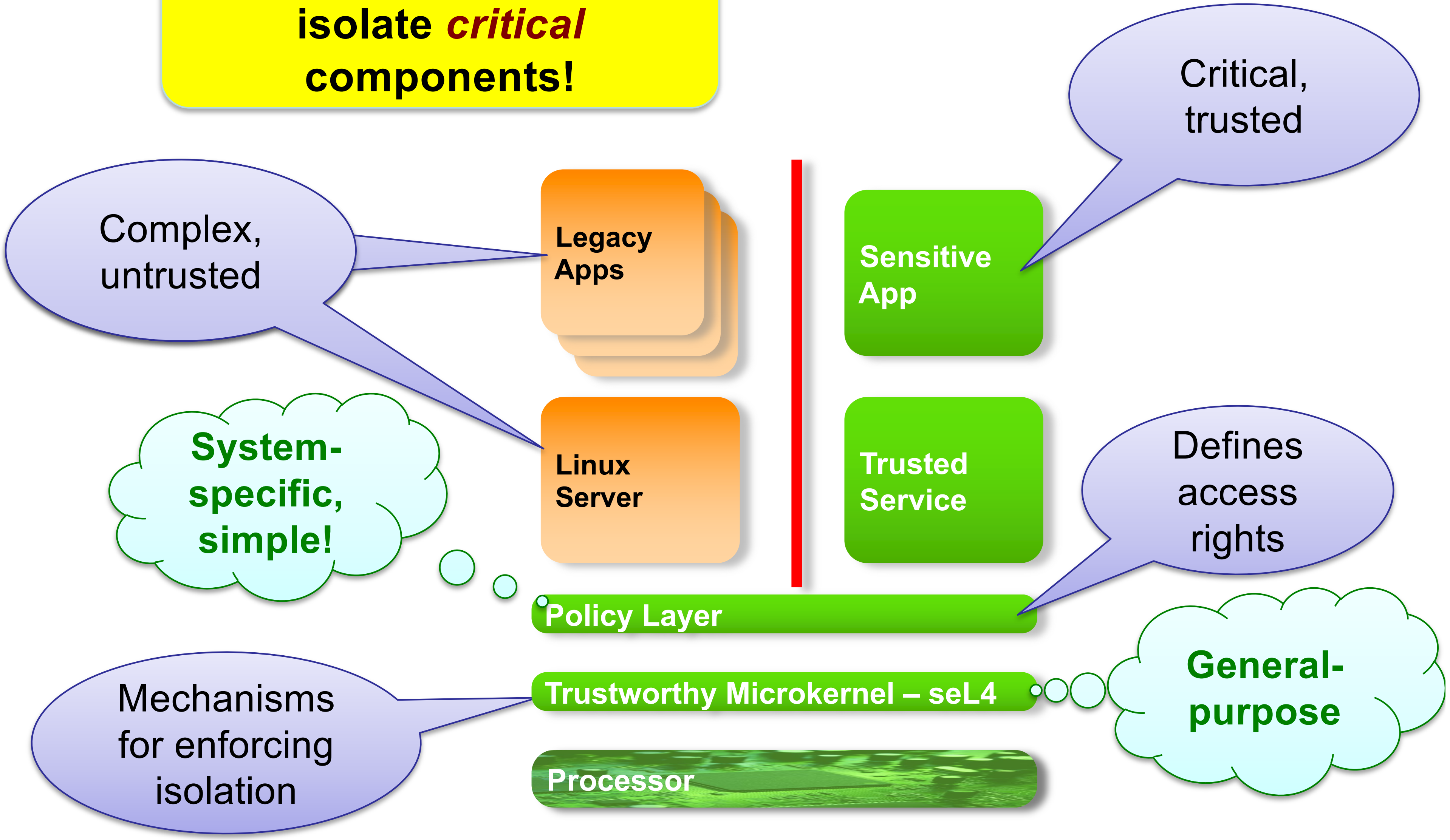
Suitable for real-world systems

We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

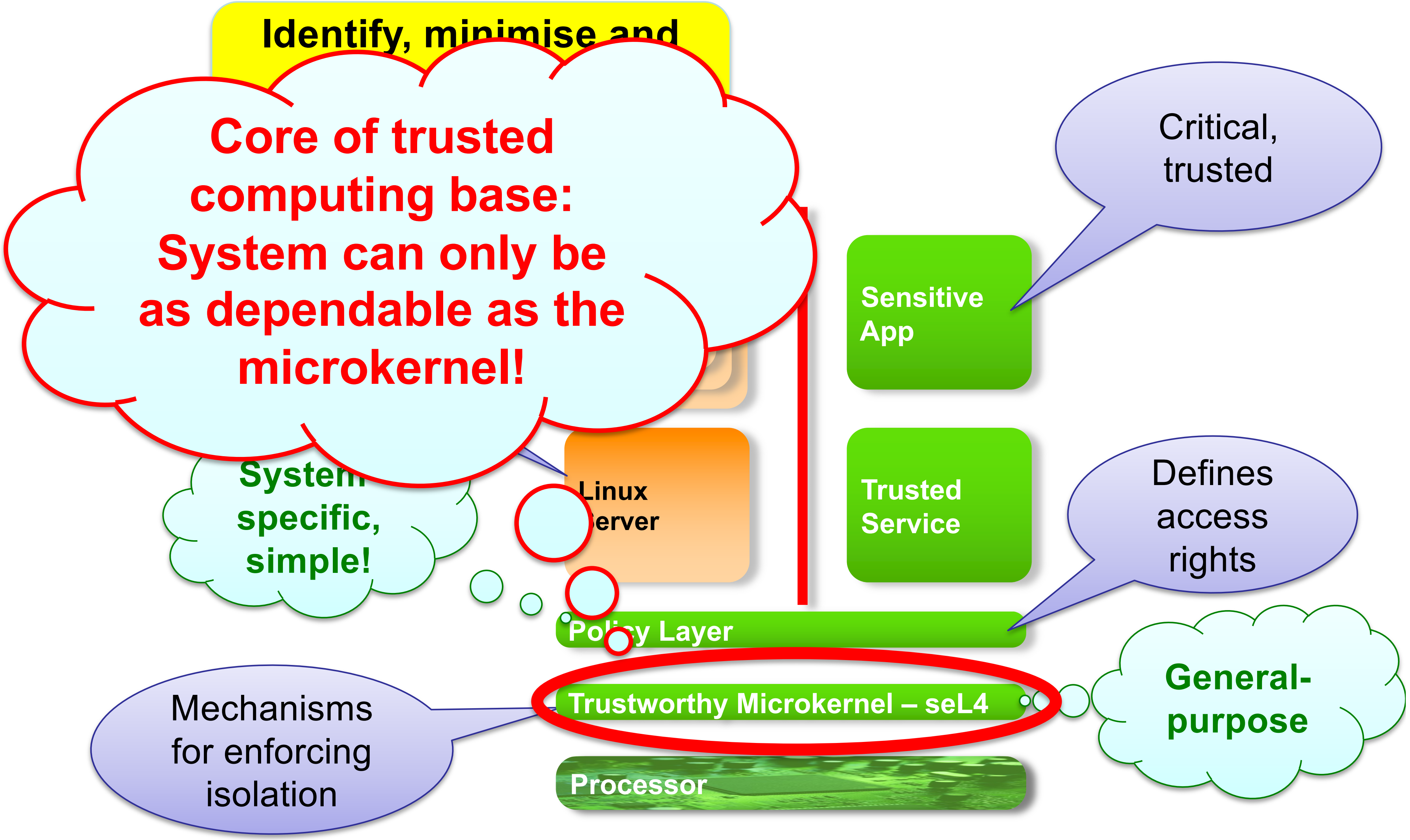
Hard *guarantees* on safety/security/ reliability

Isolation is Key!

Identify, minimise and isolate *critical* components!



Isolation is Key!



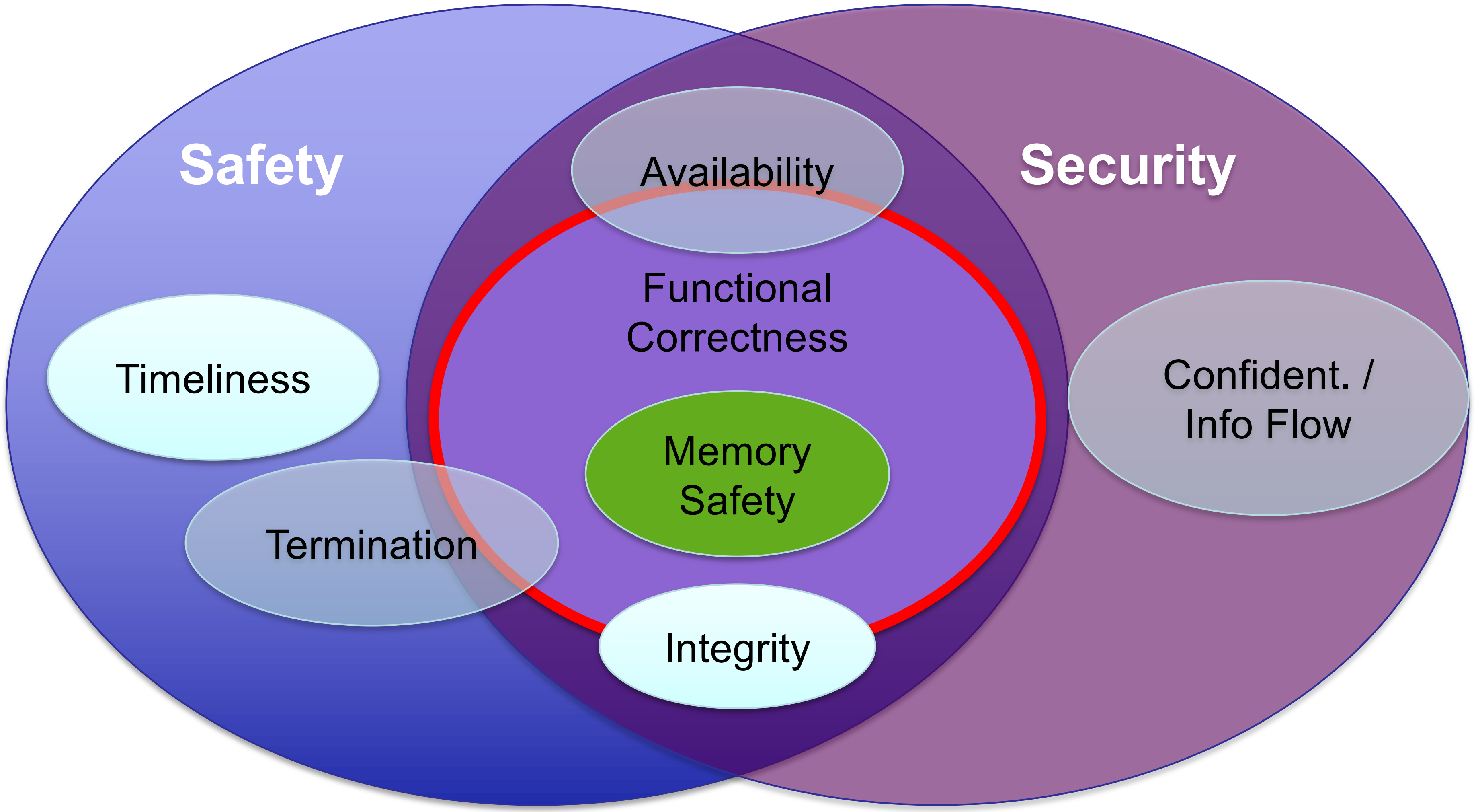
NICTA Trustworthy Systems Agenda

NICTA

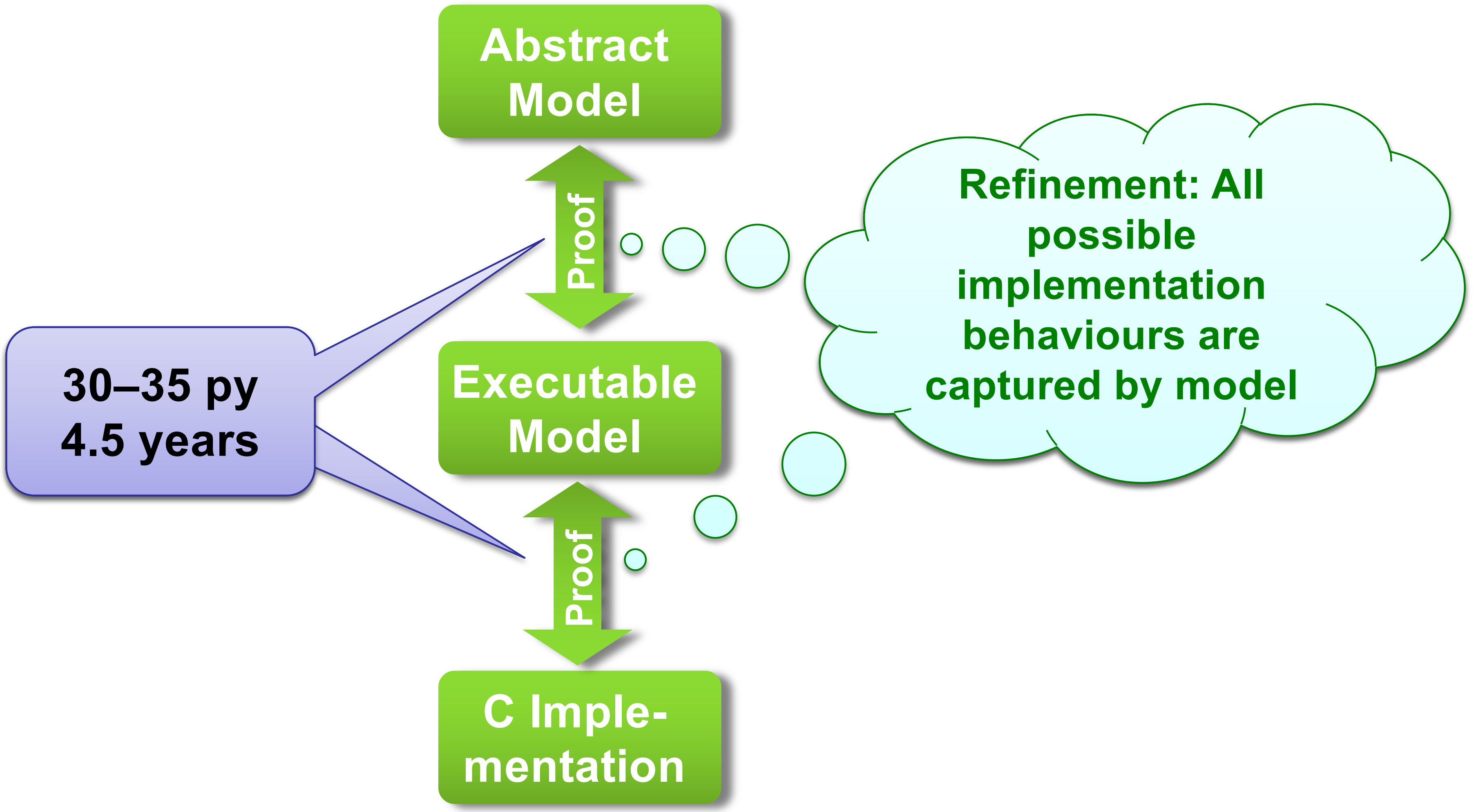
1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

2. **Lift microkernel guarantees to whole system**
   - Use kernel correctness and integrity to guarantee critical functionality
   - Ensure correctness of balance of trusted computing base
   - Prove dependability properties of complete system

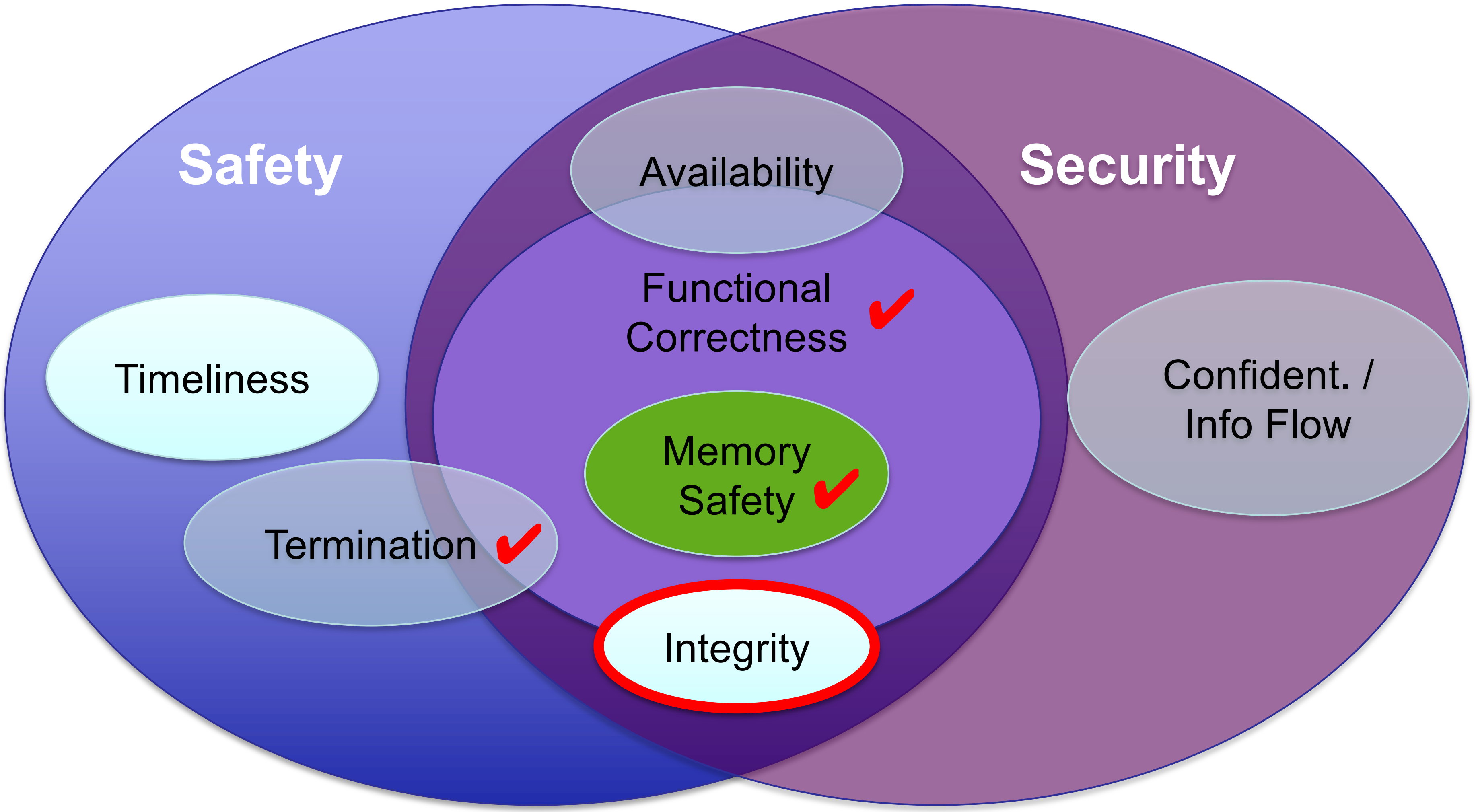
seL4 as Basis for Trustworthy Systems



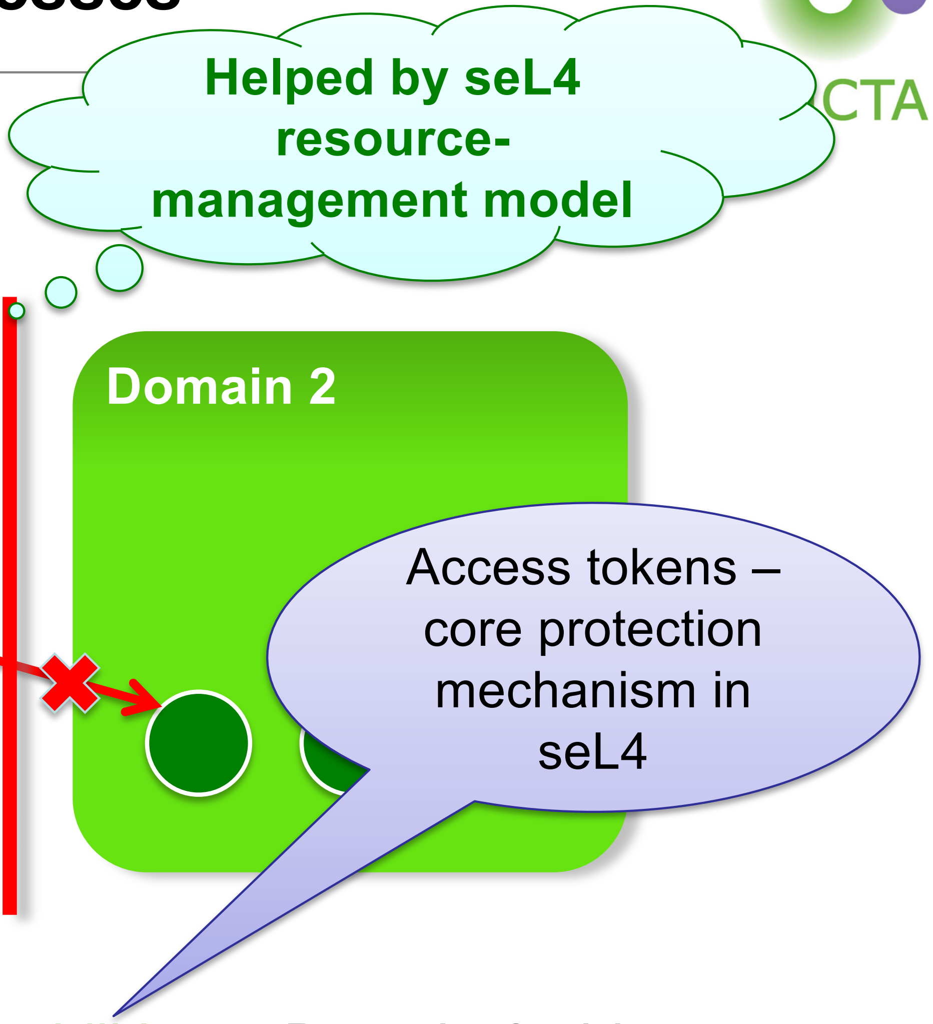
Proving seL4 Security/Safety



seL4 as Basis for Trustworthy Systems



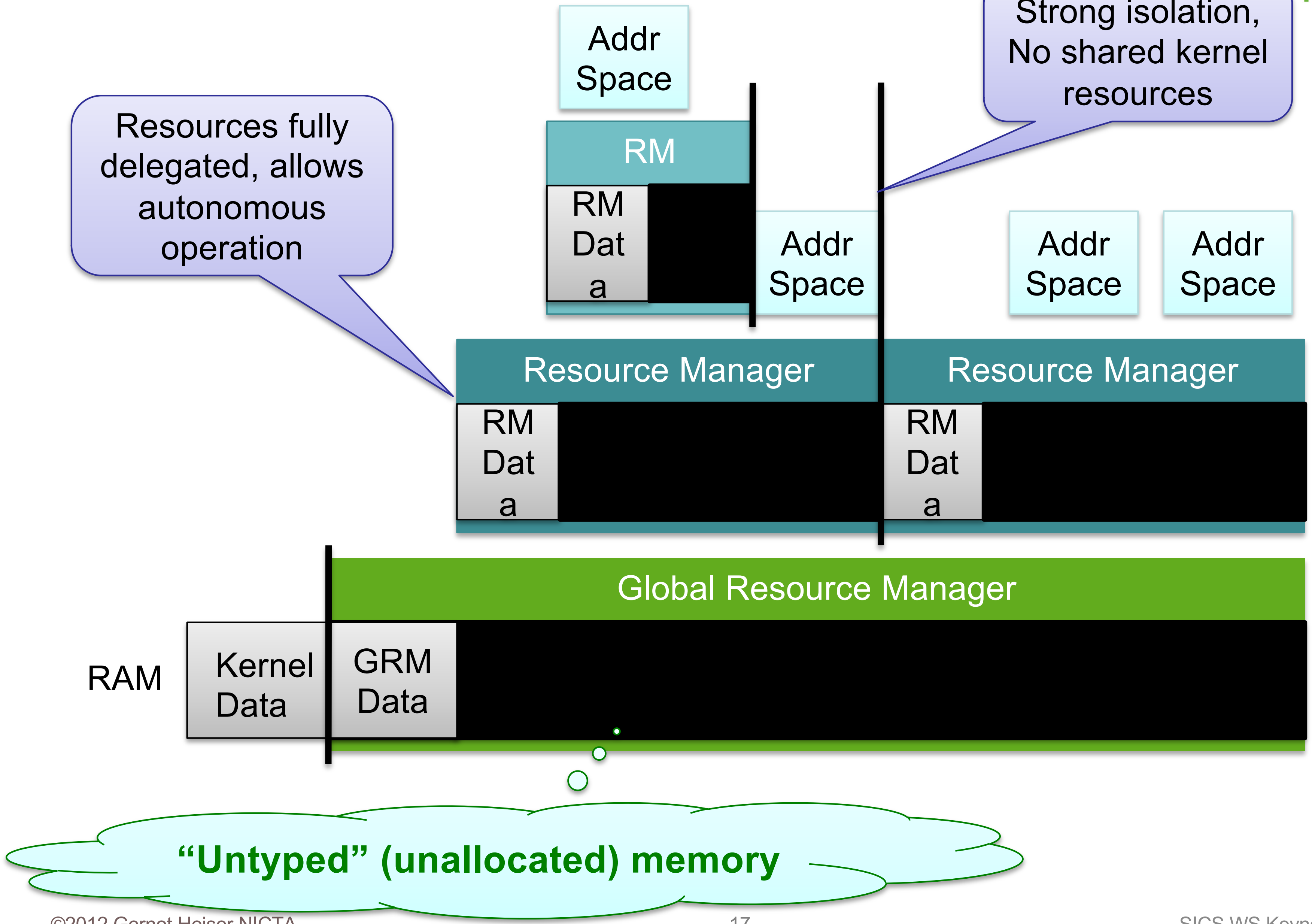
Integrity is about Write Accesses


Helped by seL4 resource-management model
Domain 1
Domain 2
Access tokens – core protection mechanism in seL4

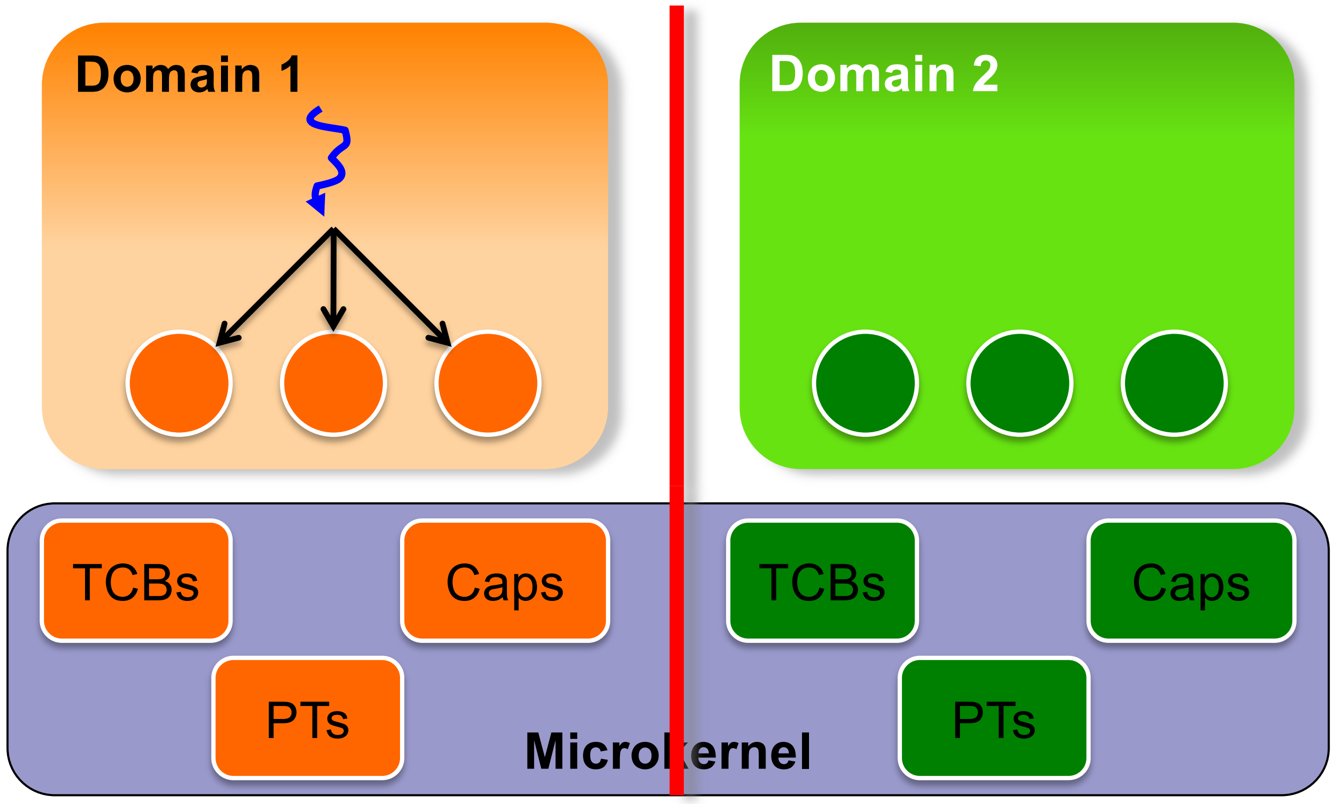
To prove:

- Domain-1 doesn't have write *capabilities* to Domain-2 objects
  ⇒ no action of Domain-1 agents will modify Domain-2 state
- Specifically, *kernel does not modify on Domain-1's behalf!*

seL4 Memory Management Approach

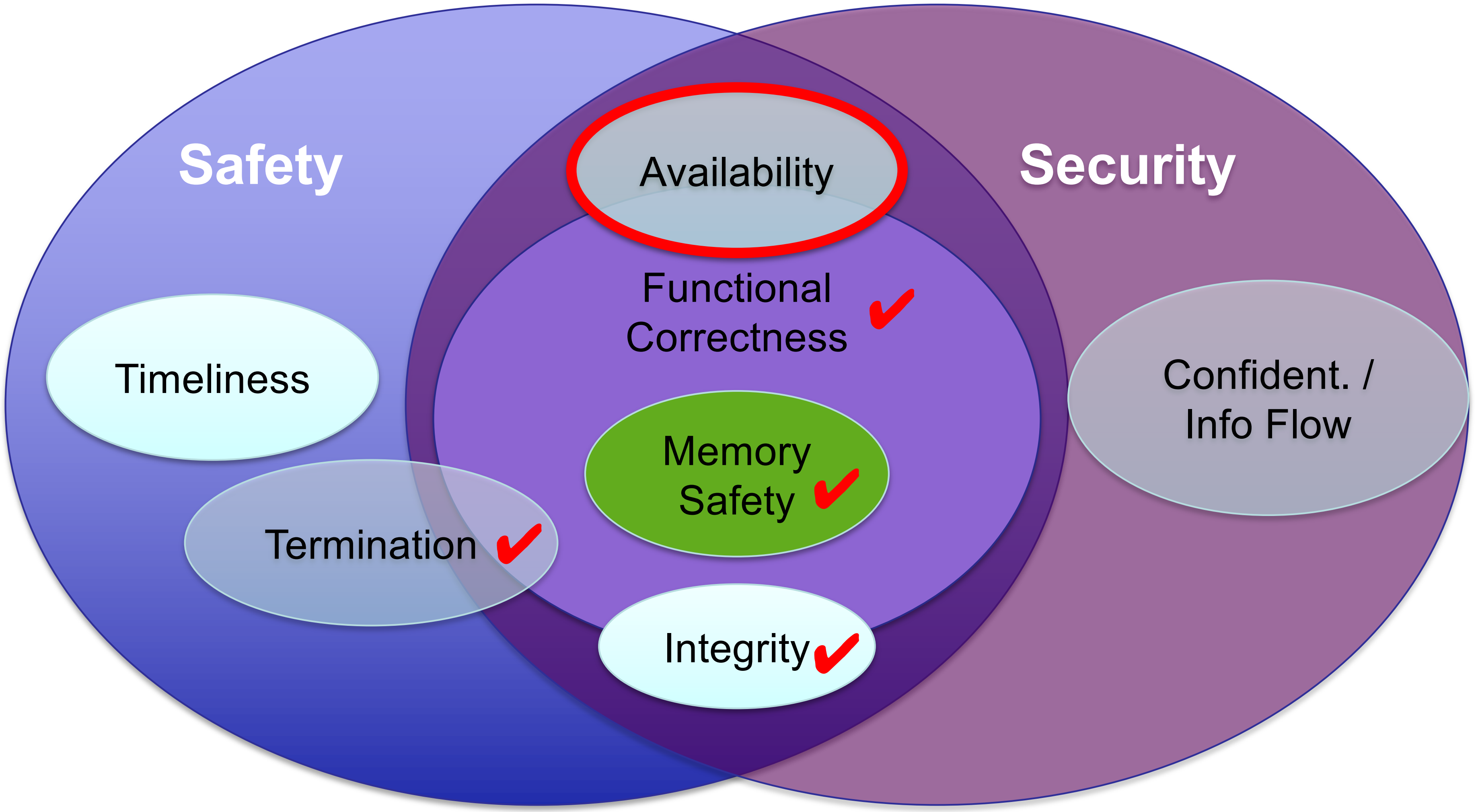


Separation of Kernel Data

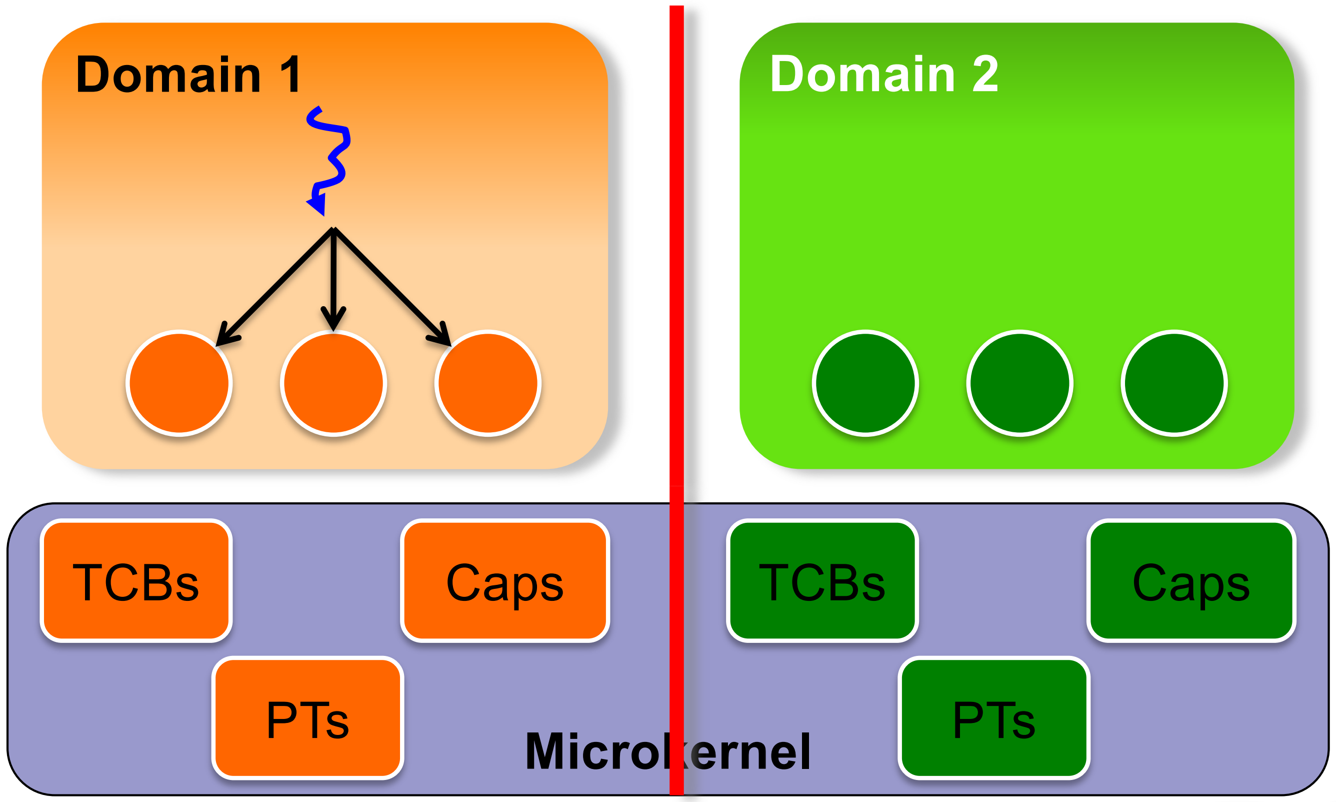


- Kernel data structures allocated/managed by user
  - Protected by capabilities just as user data!
- For integrity show that *no* object can be *modified* without a *write cap*

seL4 as Basis for Trustworthy Systems



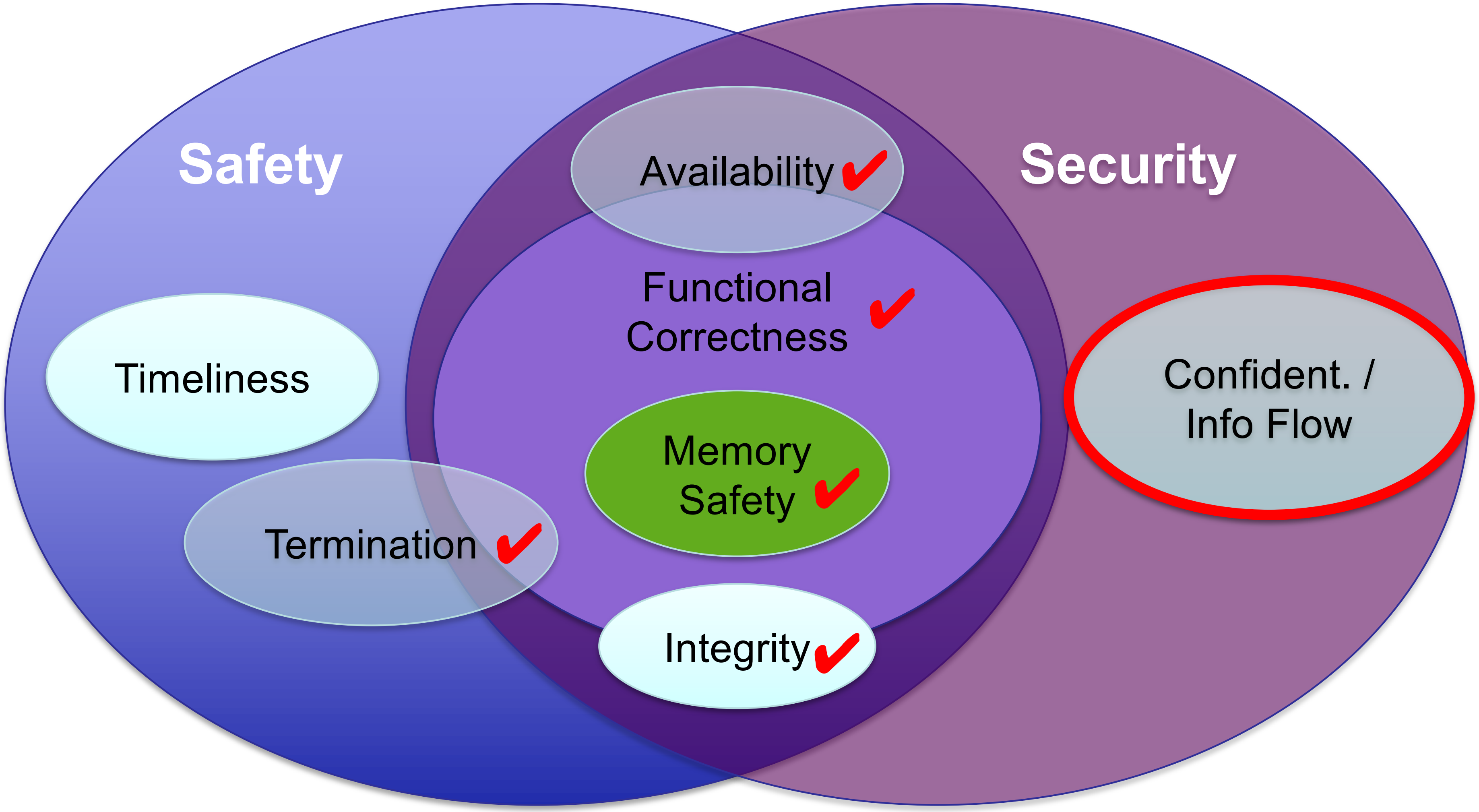
Availability is Trivially Ensured at Kernel Level



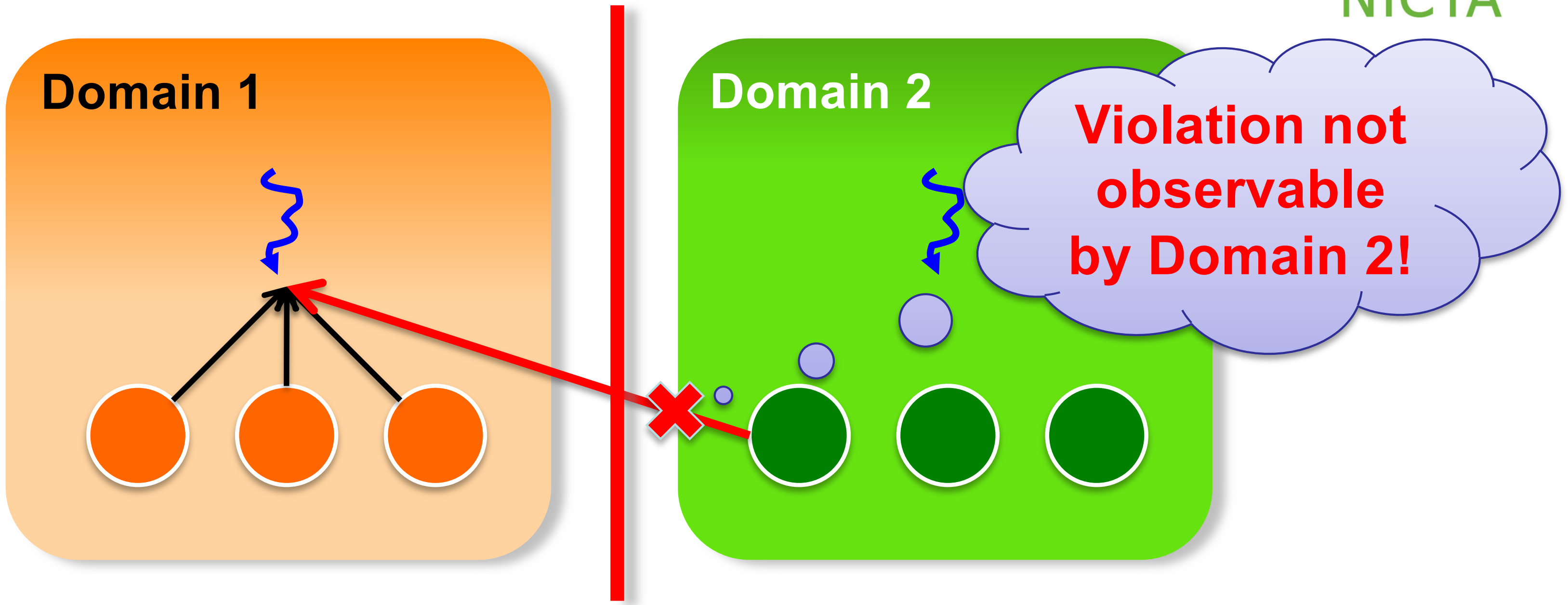
Managing resource availability is user-level issue!

- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources

seL4 for Safety and Security



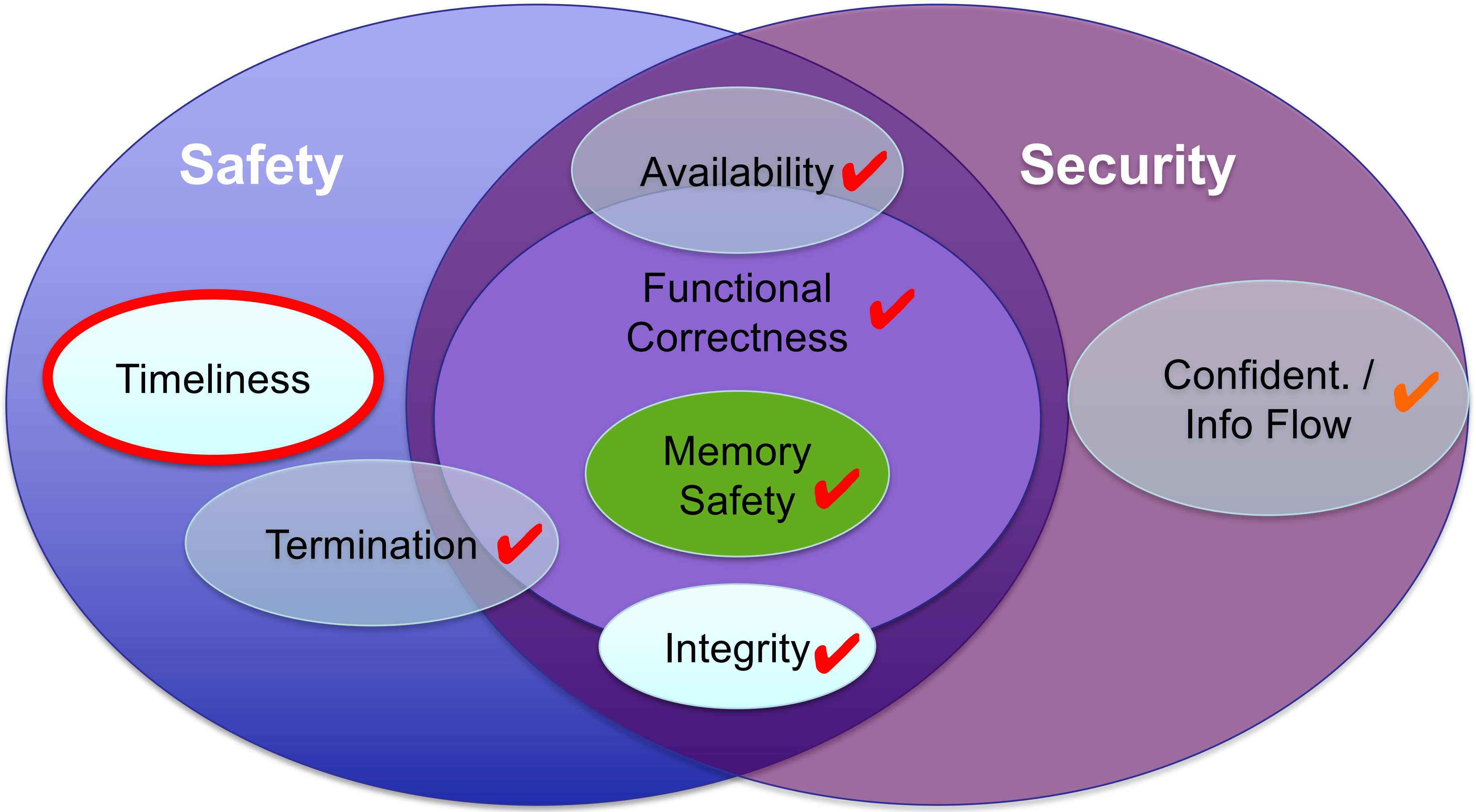
Confidentiality is about Read Accesses



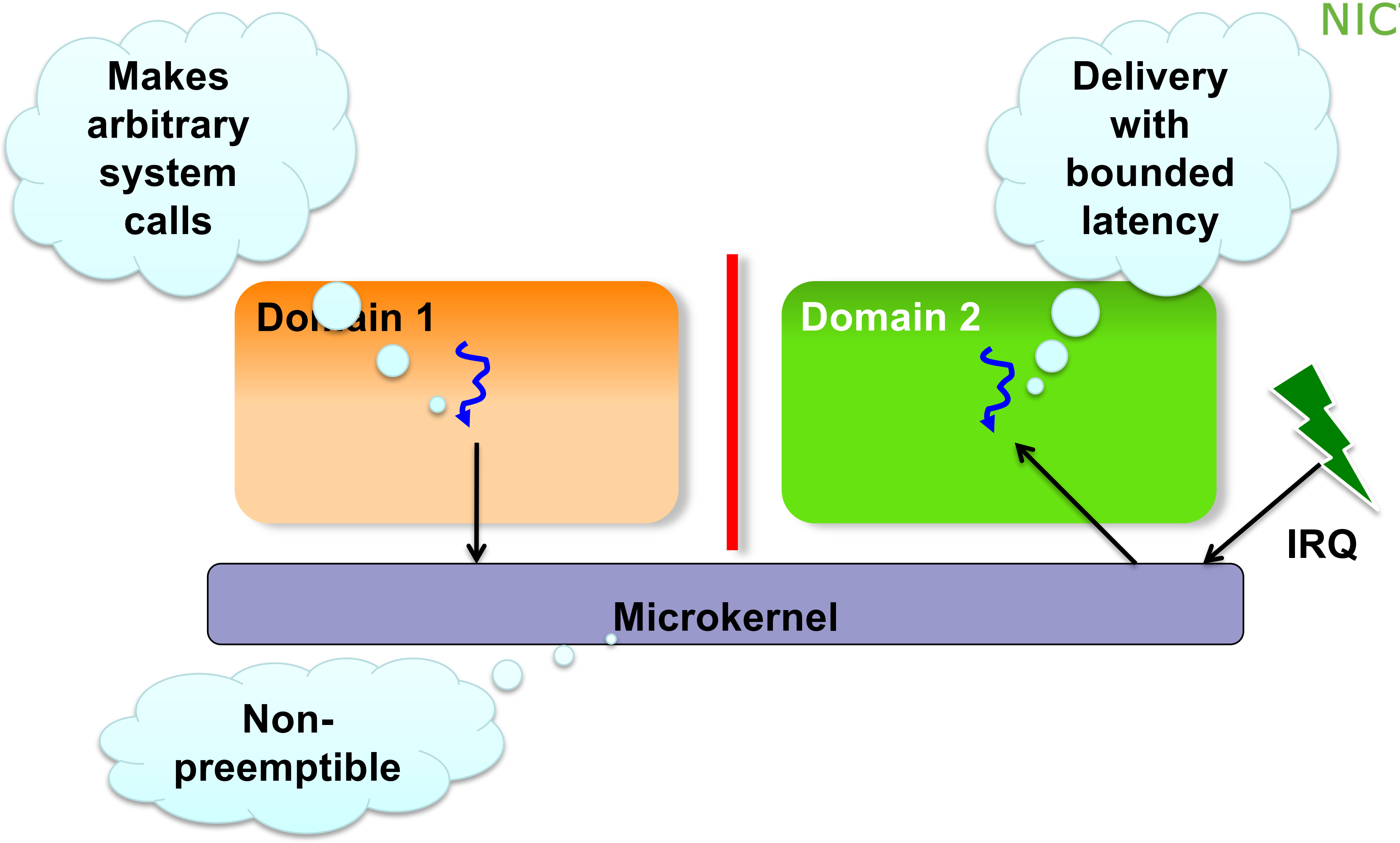
To prove:

- Domain-1 doesn't have read capabilities to Domain-2 objects
  ⇒ no action of any agents will reveal Domain-2 state to Domain-1
- Harder than write, as protected data doesn't change
- Non-interference proof in progress…
  - Show that Domain-1's evolution cannot depend on Domain-2's state
- Presently only looking at *overt* information flow!

seL4 as Basis for Trustworthy Systems

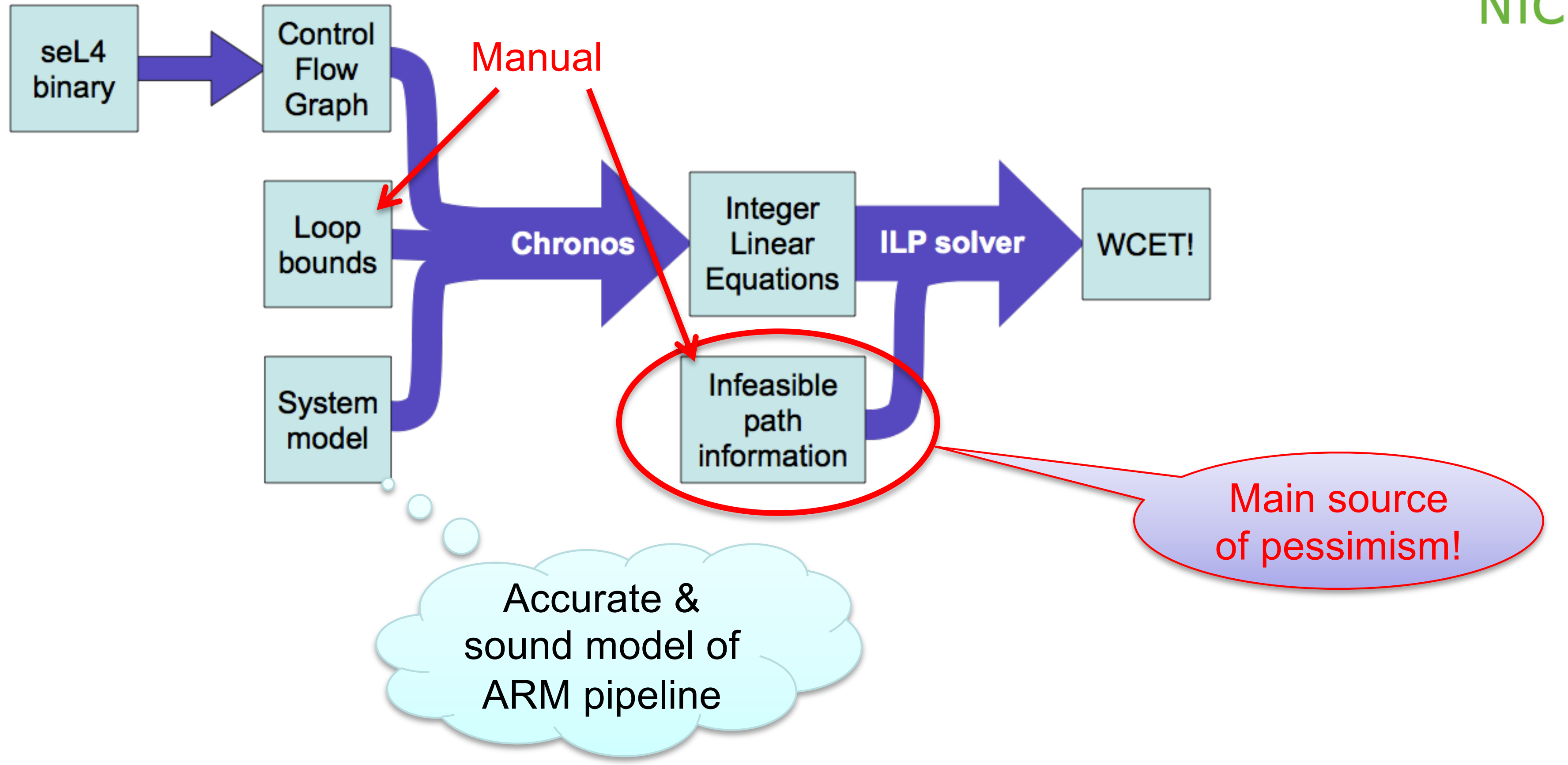


Timeliness



Need worst-case execution time (WCET) analysis of kernel

WCET Analysis Approach



Result: WCET >1 sec!

- Pessimism of analysis (loop bounds, infeasible paths)

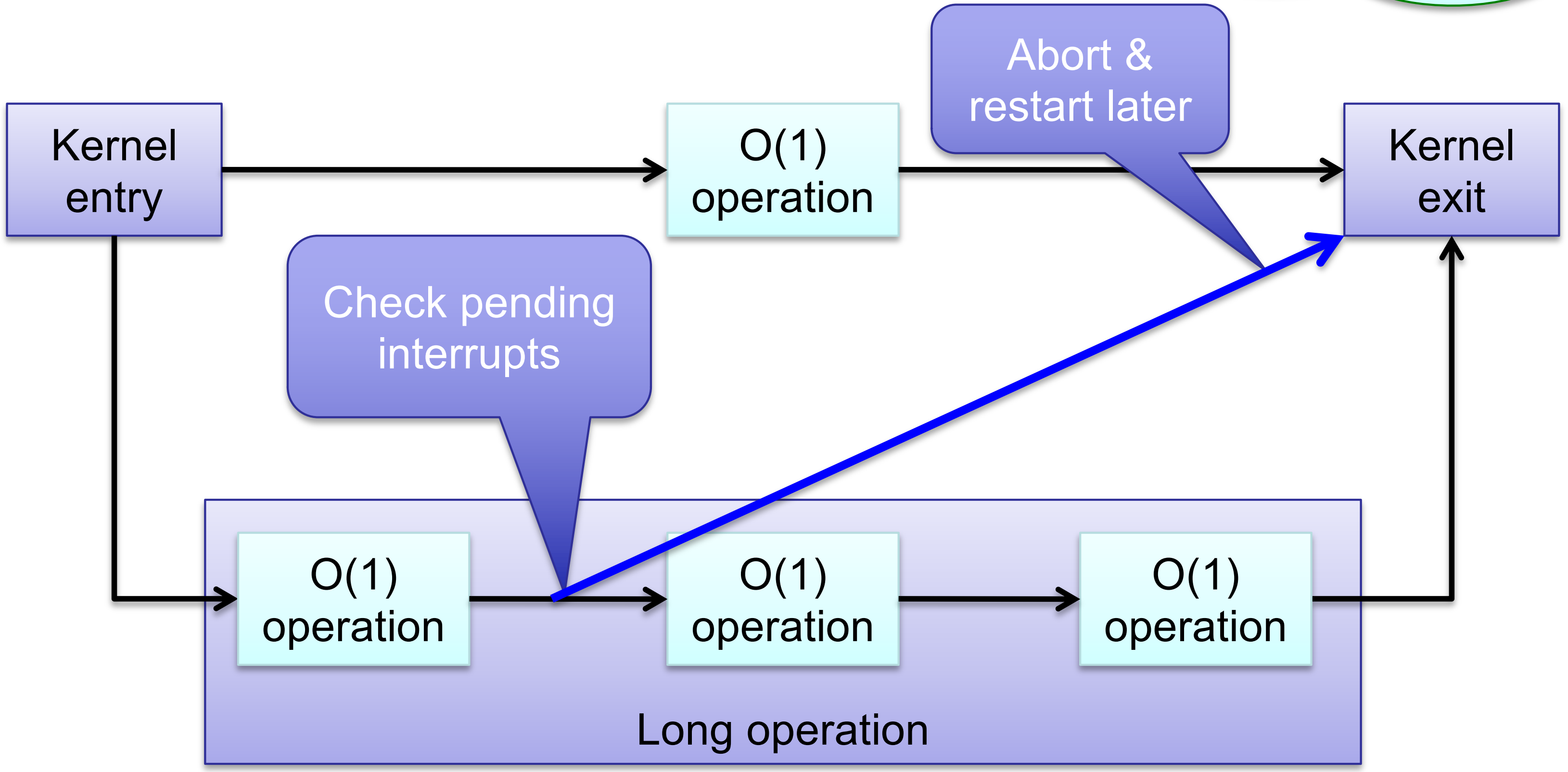
⇒ Manual elimination of infeasible paths

- Result: 600 ms ☹

Improving Real-Time Behaviour of seL4

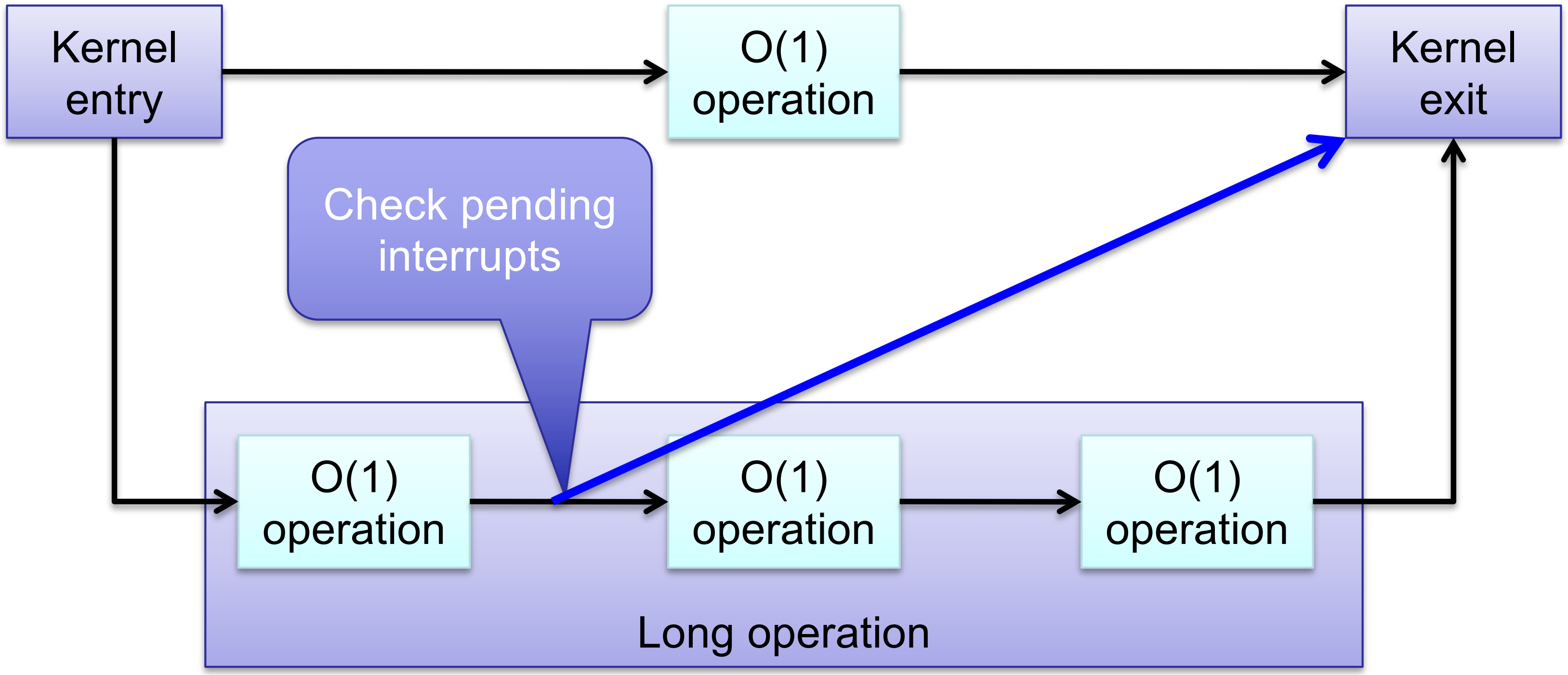
- Challenge: Improving WCET while
  - retaining ability to verify
  - maintaining high average-case performance

Event-oriented kernel running with interrupts disabled!

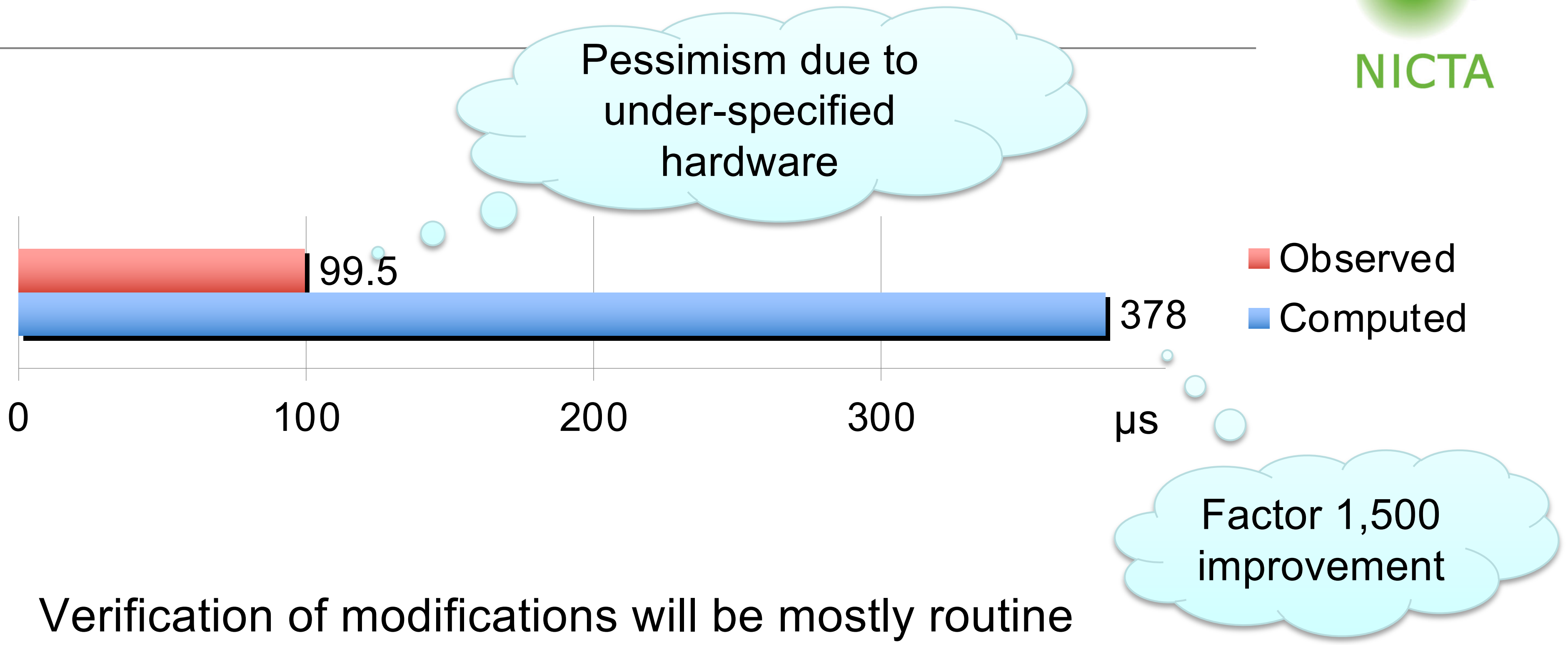


Placing Preemption Points

- 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

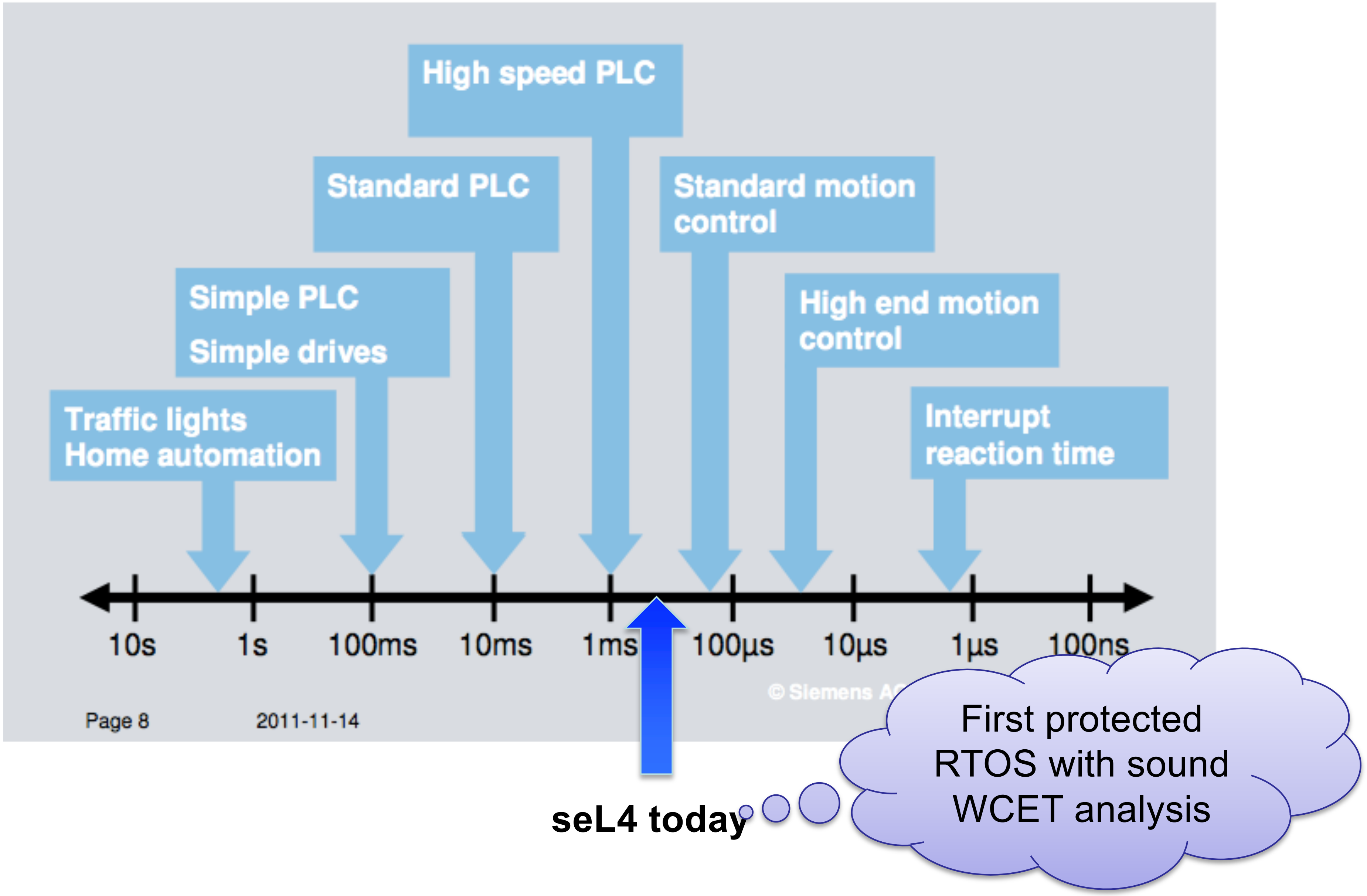


Result

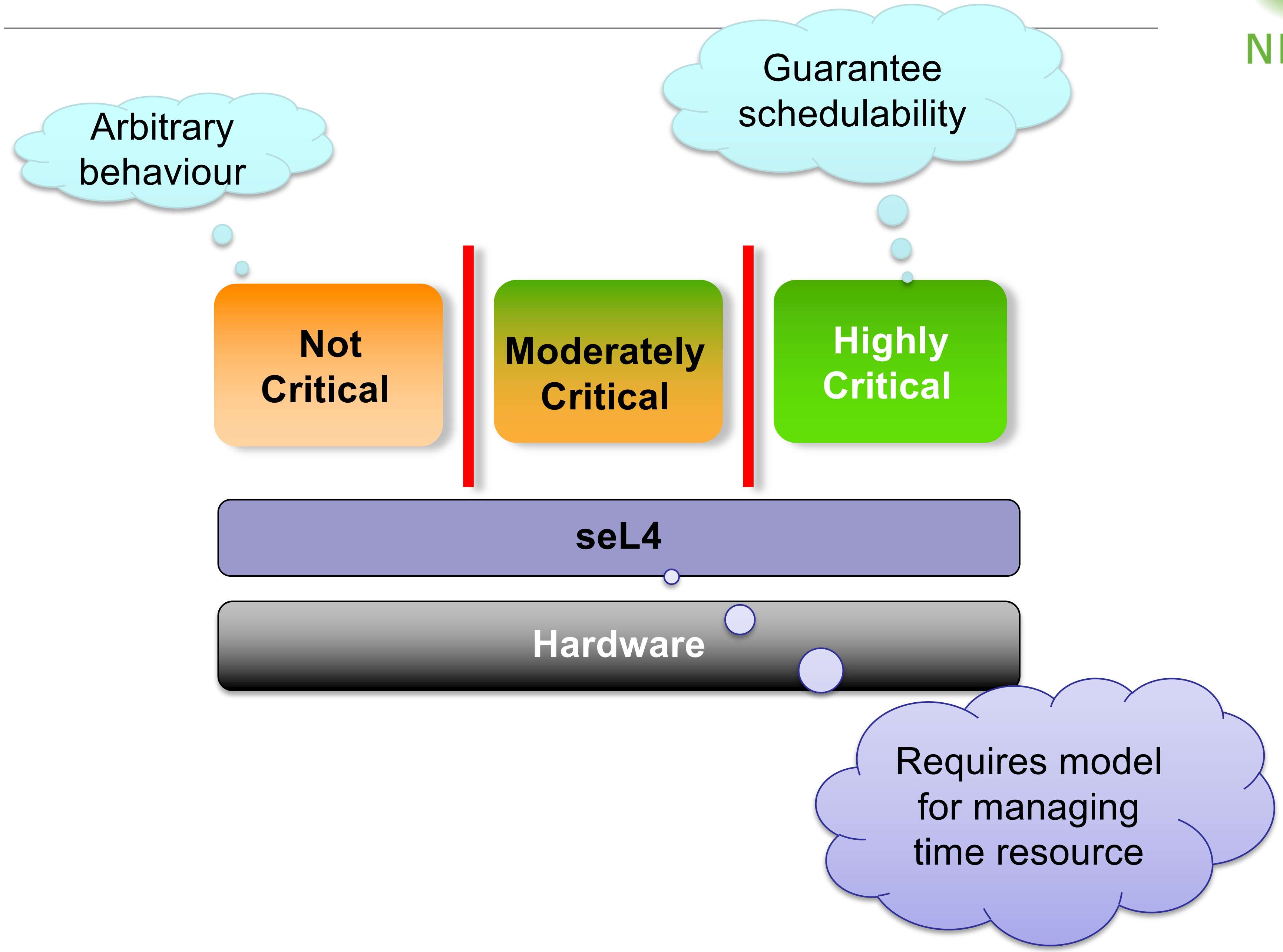


- Verification of modifications will be mostly routine
- In progress (almost complete):
  - automatic determination of loop counts
  - automatic infeasible path elimination

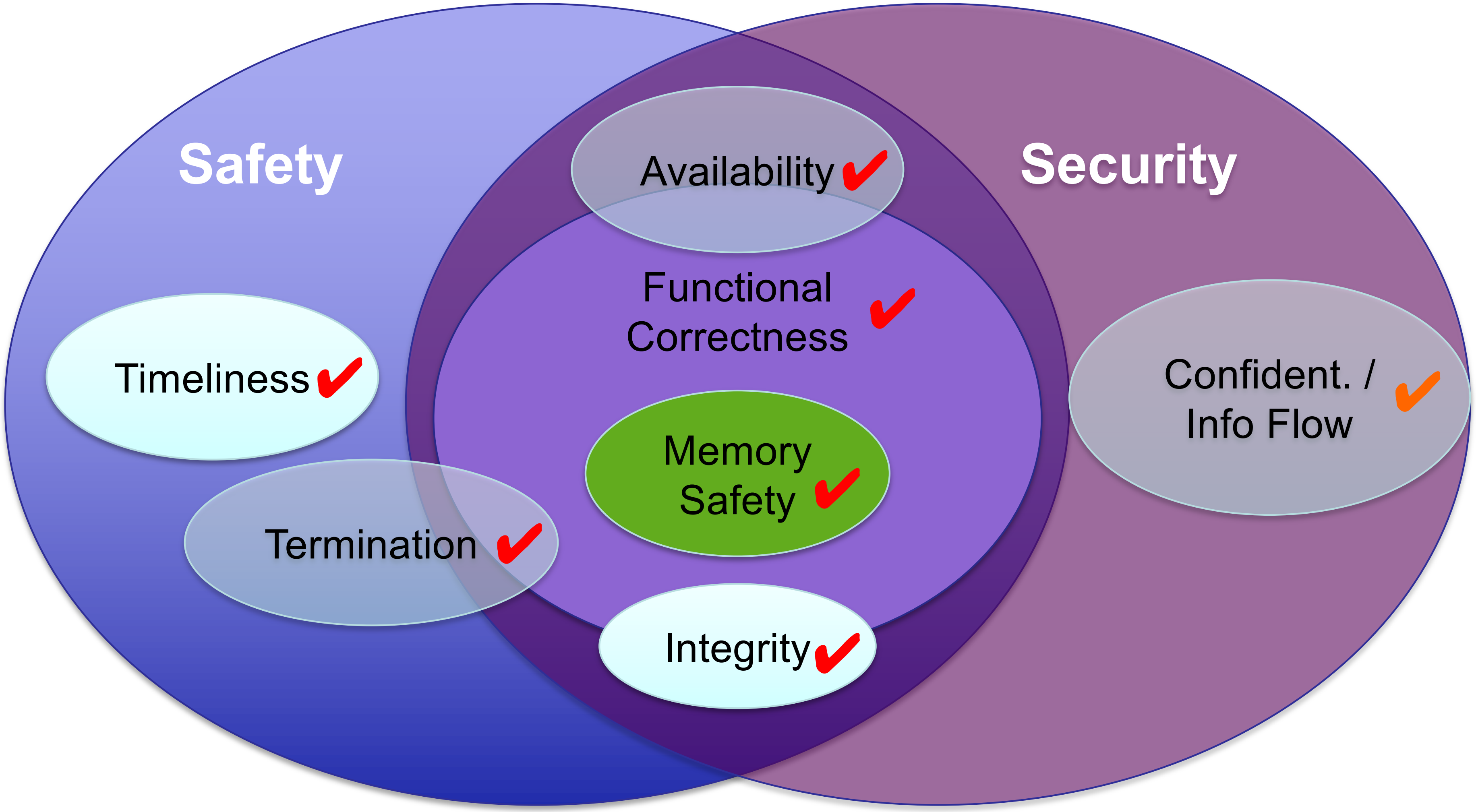
RT Requirements in Industrial Automation



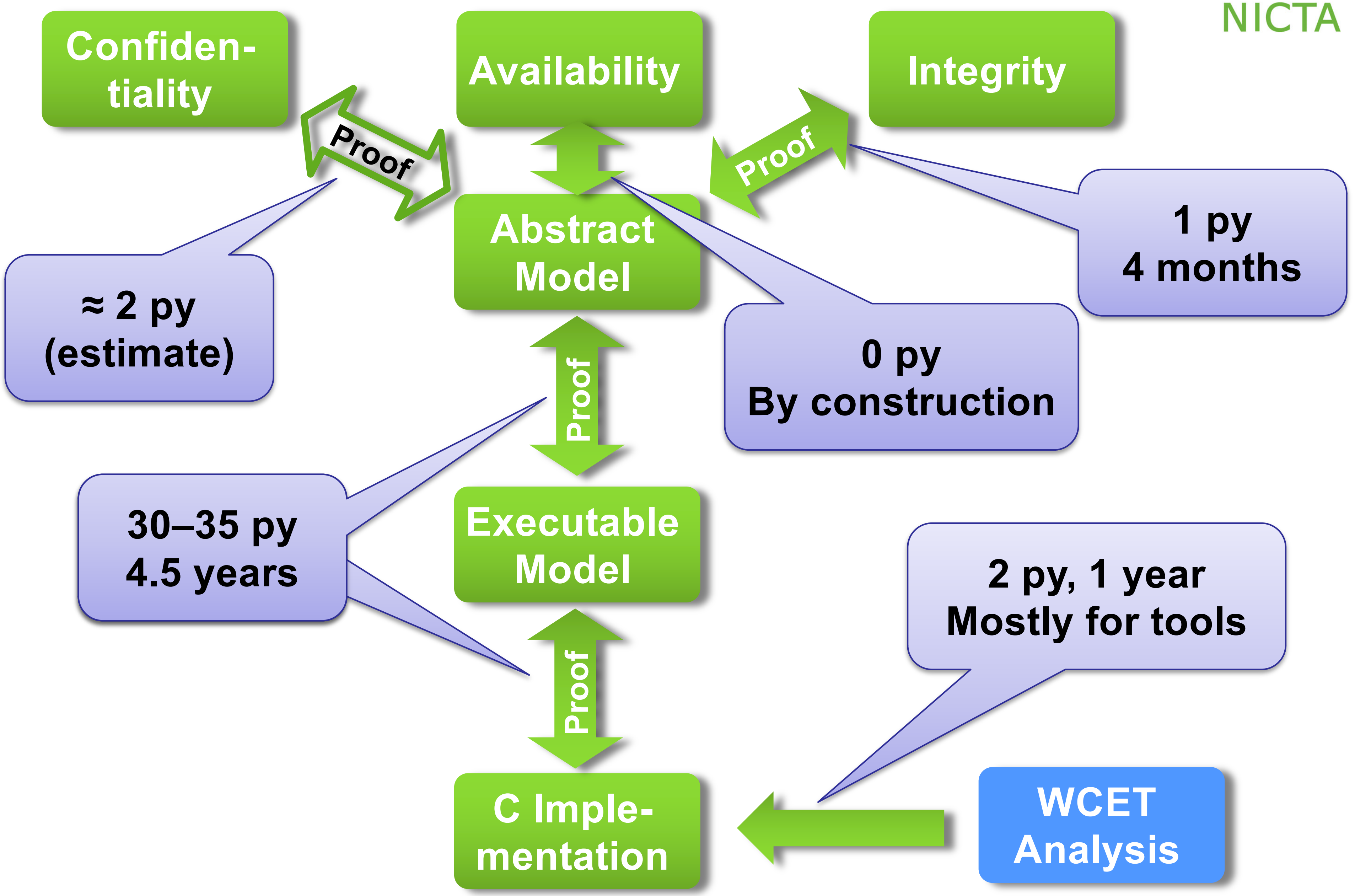
Future: Whole-System Schedulability



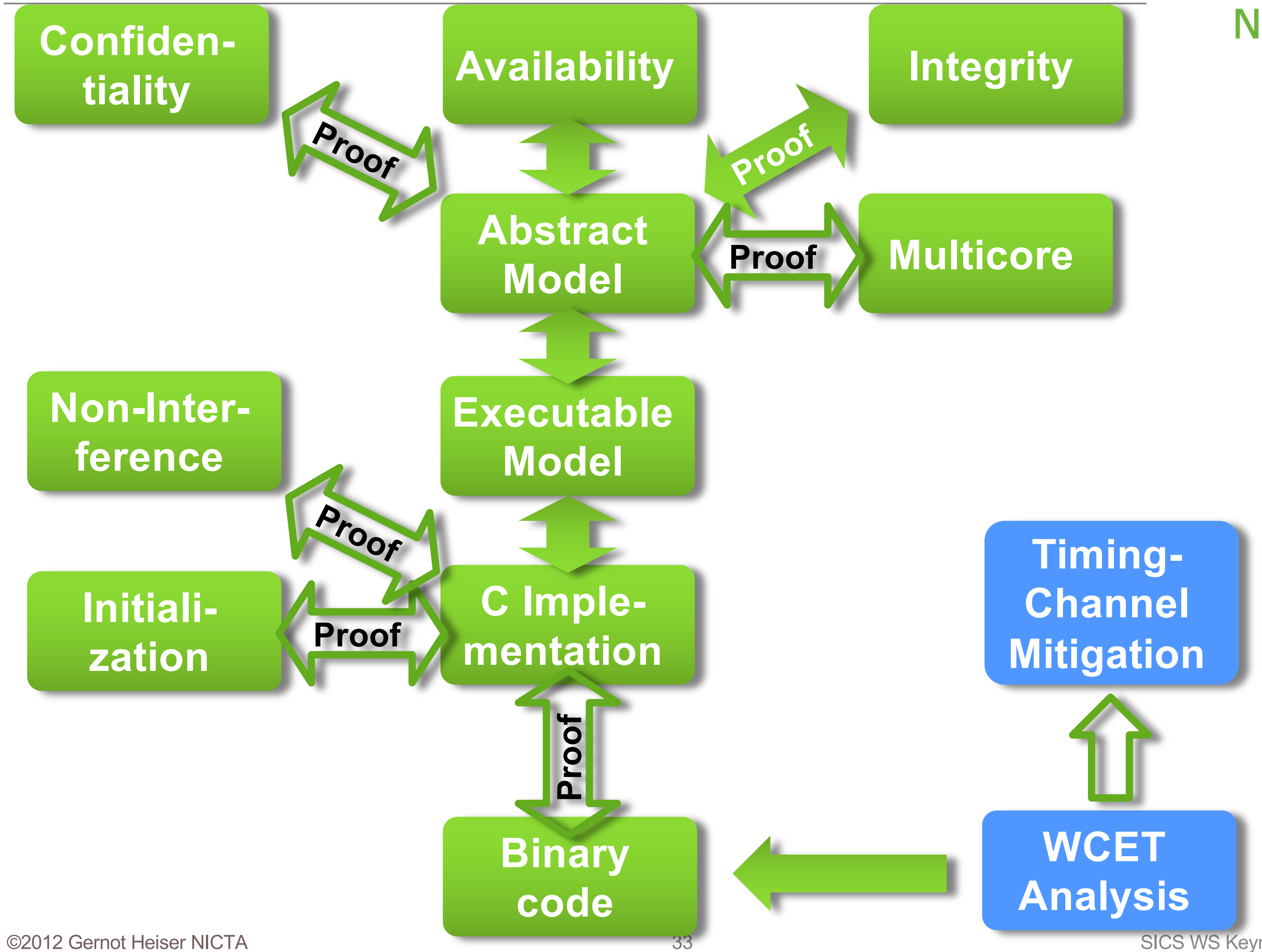
seL4 for Safety and Security



Proving seL4 Security/Safety



seL4 – the Next 24 Months

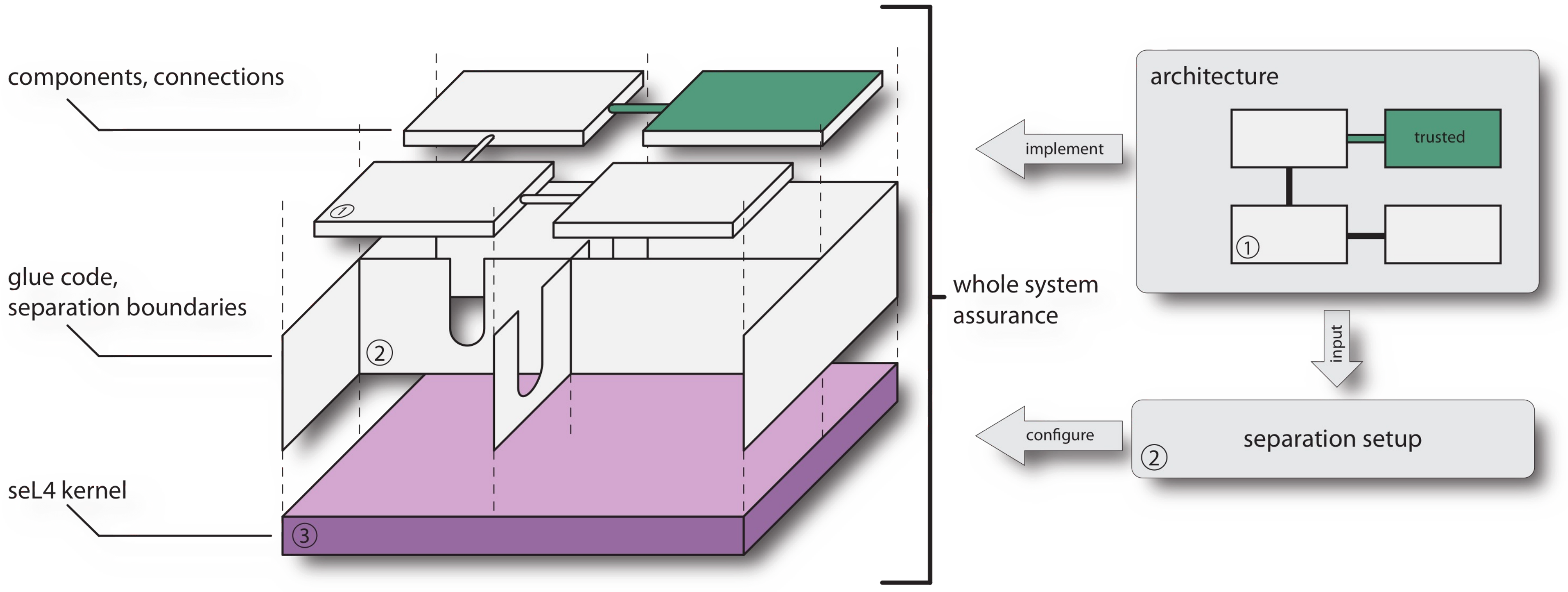


Phase Two: Full-System Guarantees

- Achieved: Verification of microkernel (8,700 LOC)

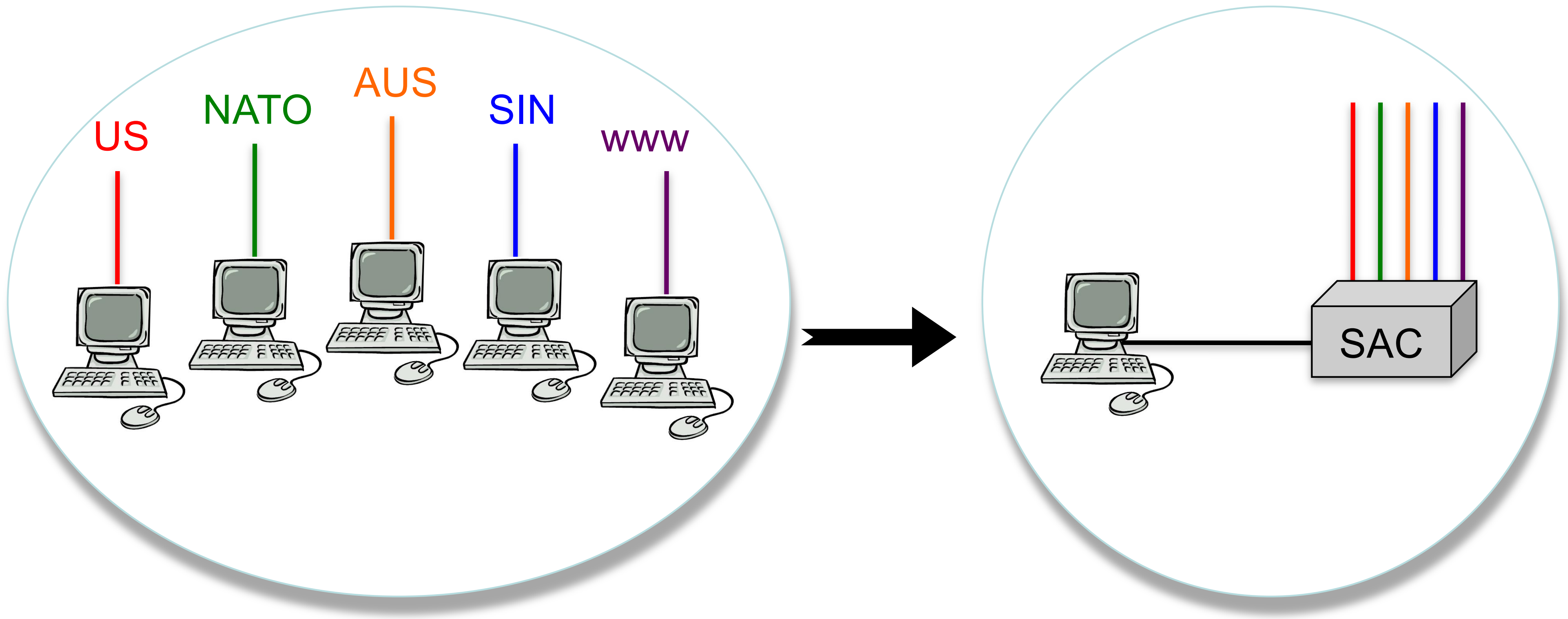
- Next step: Guarantees for real-world systems (1,000,000 LOC)

Overview of Approach

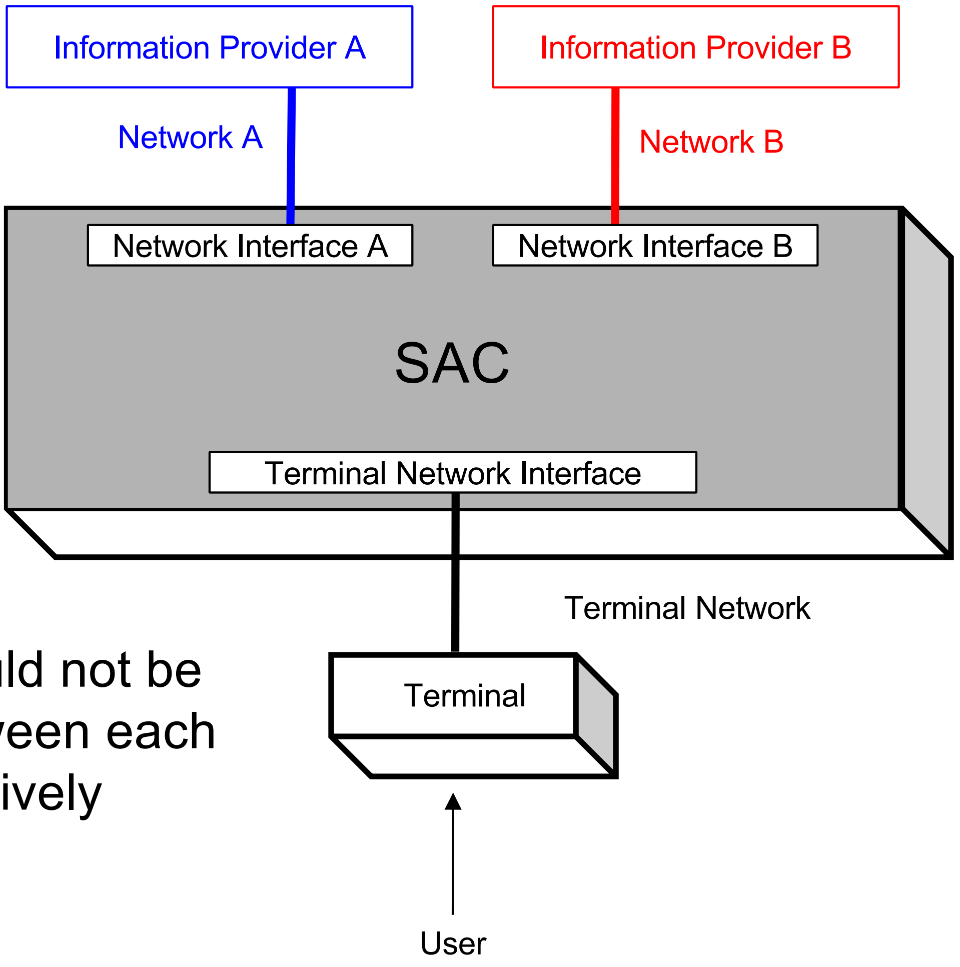


- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance

Proof of Concept: Secure Access Controller

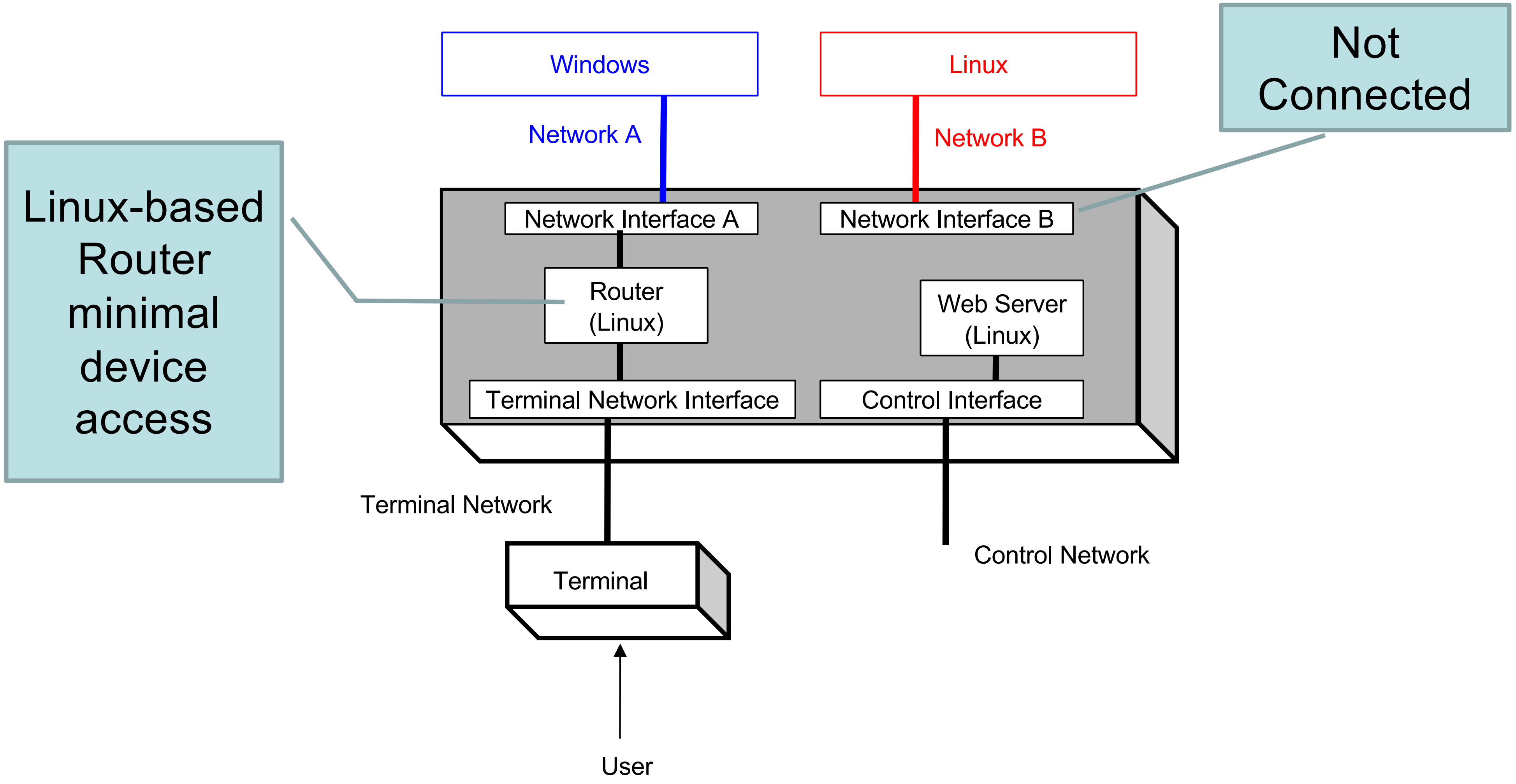


SAC Aim

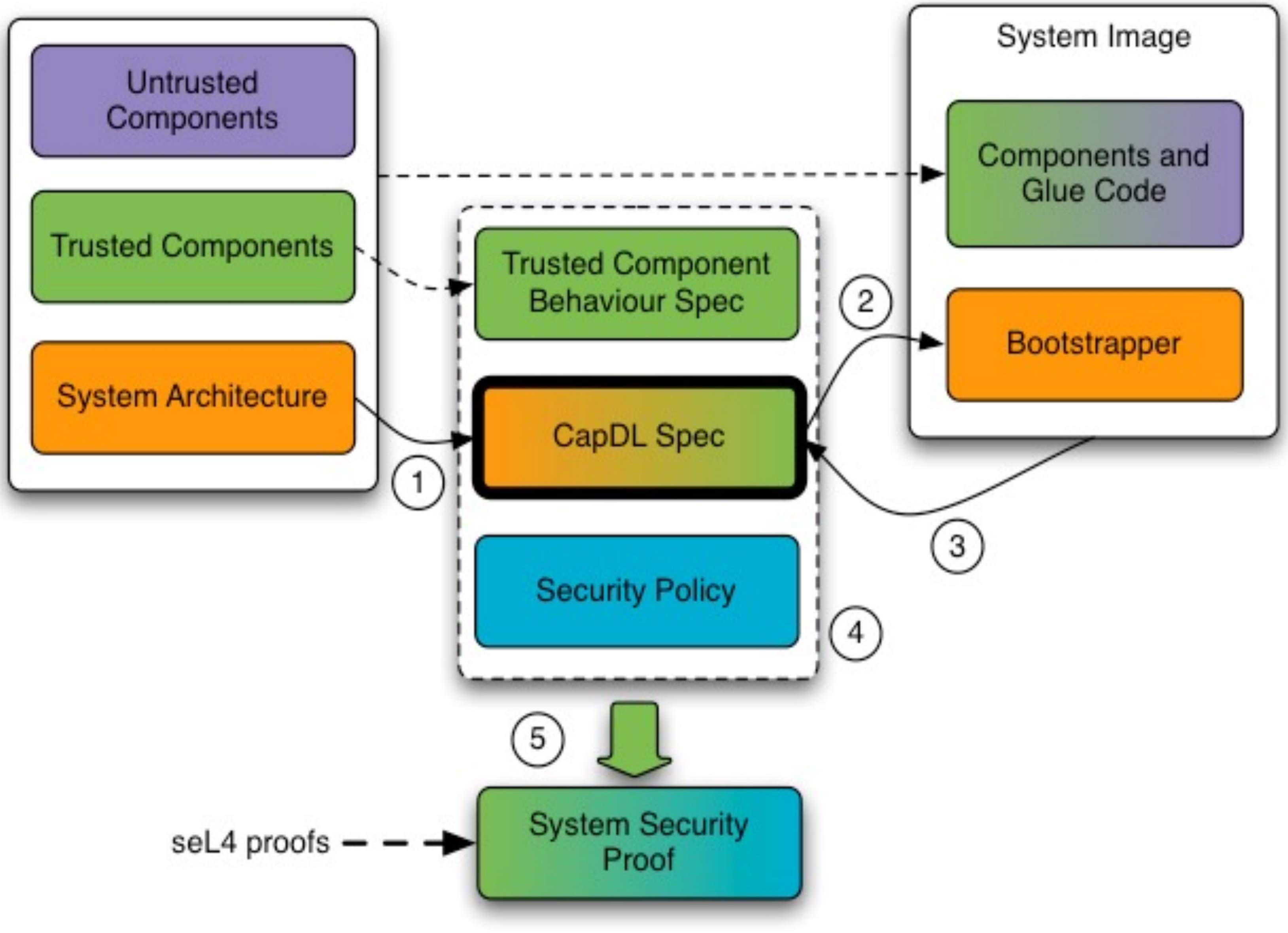


Providers A & B should not be able to leak info between each other even if they actively cooperate

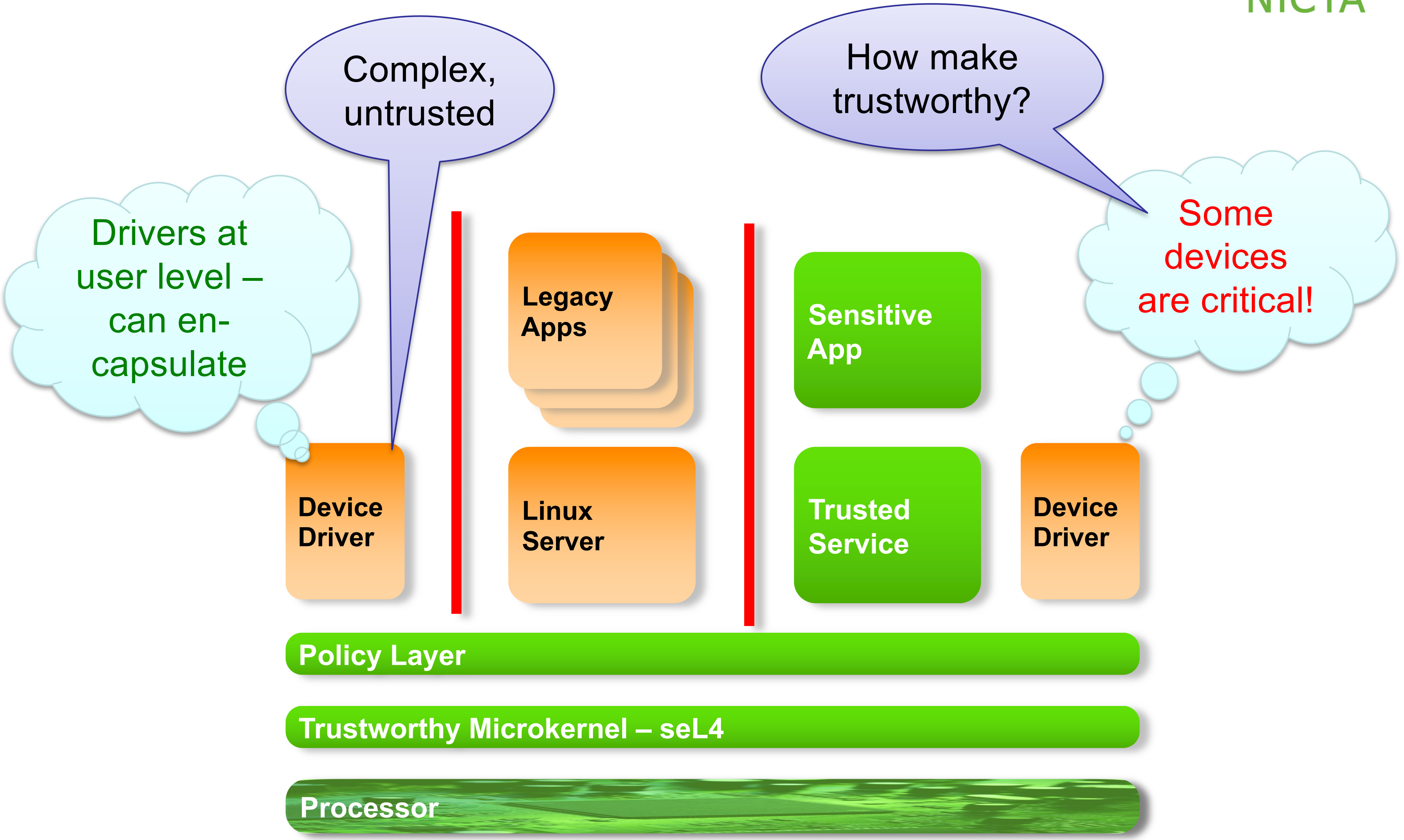
Solution Overview



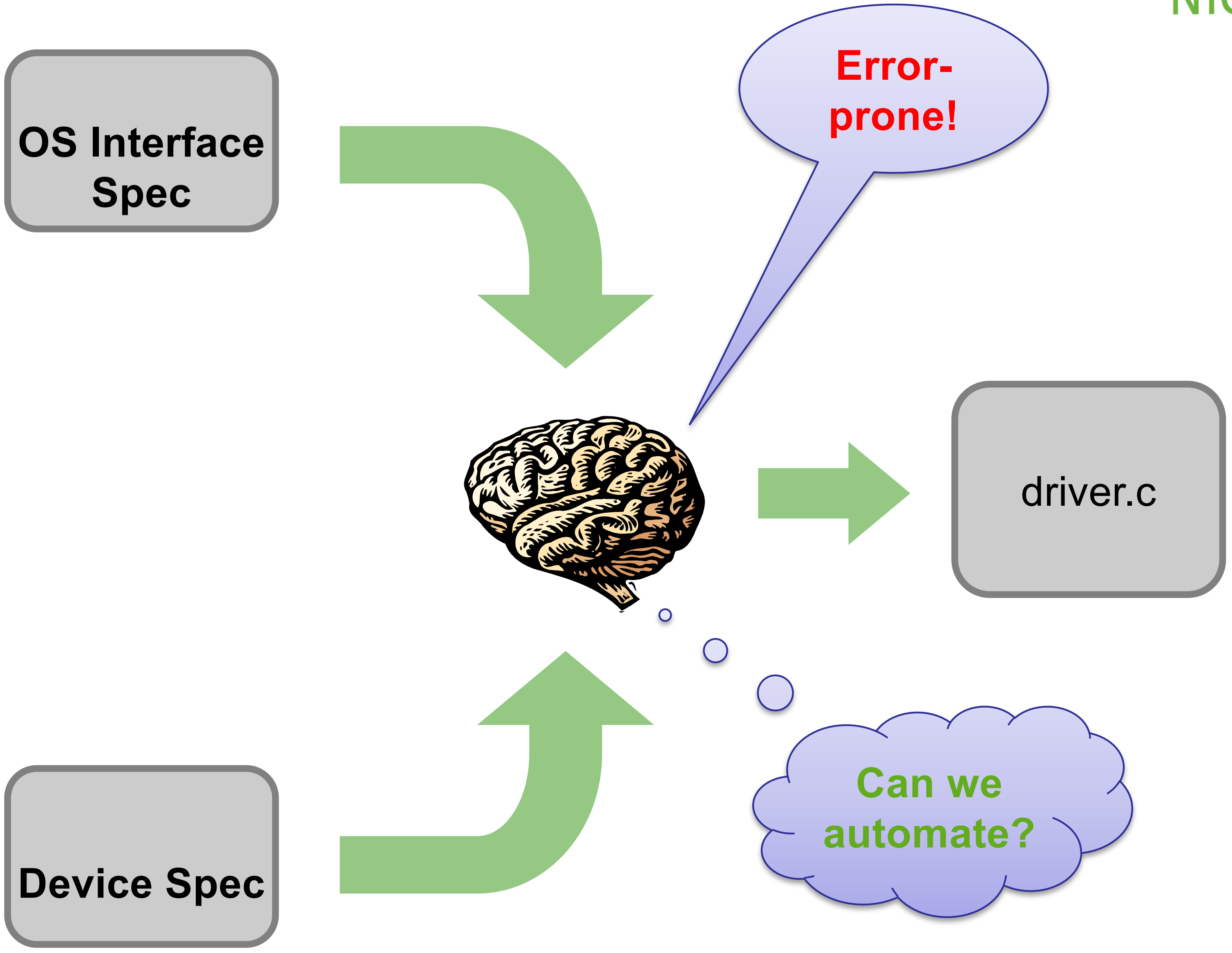
Specifying Security Architecture



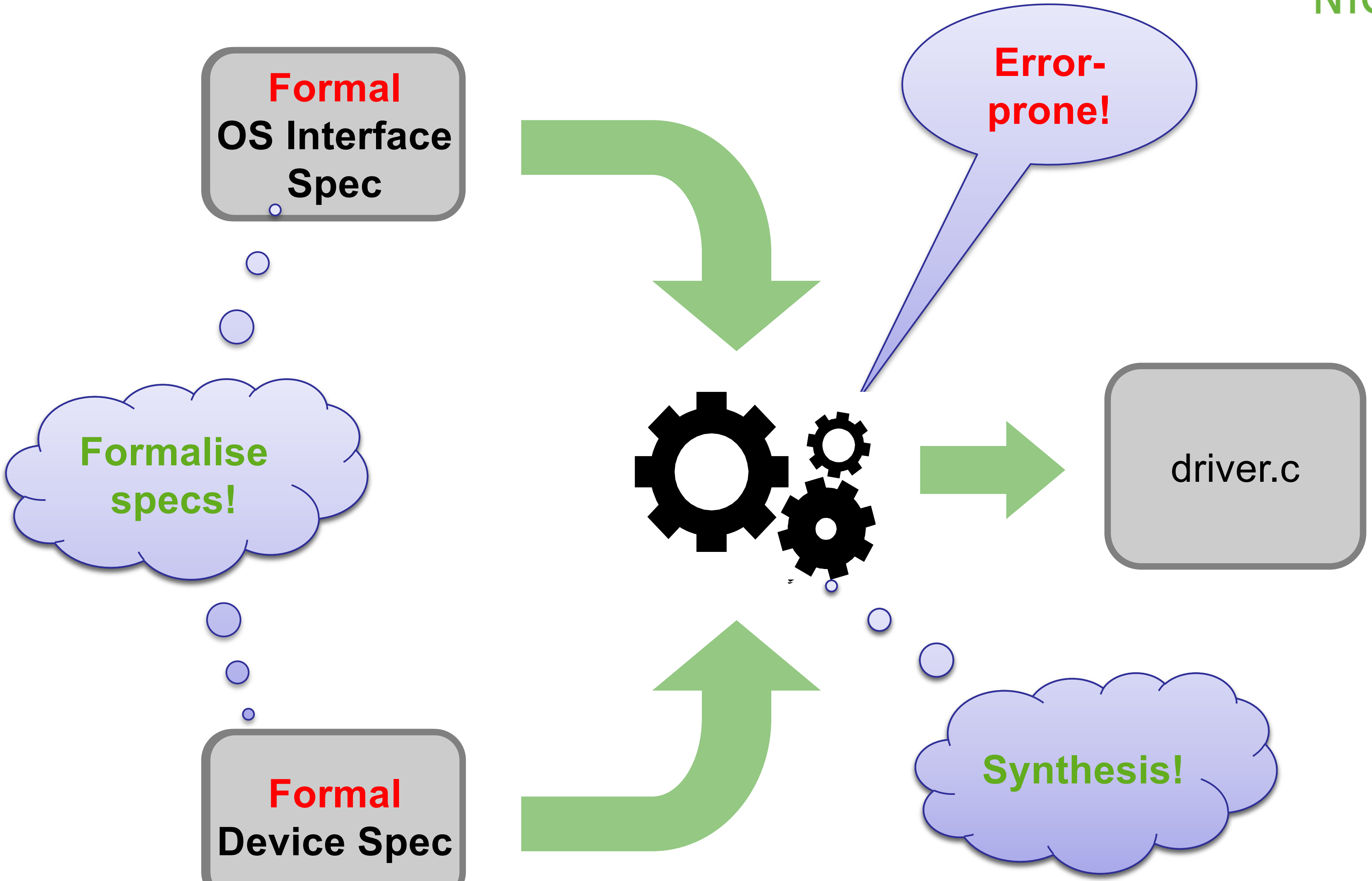
Device Drivers



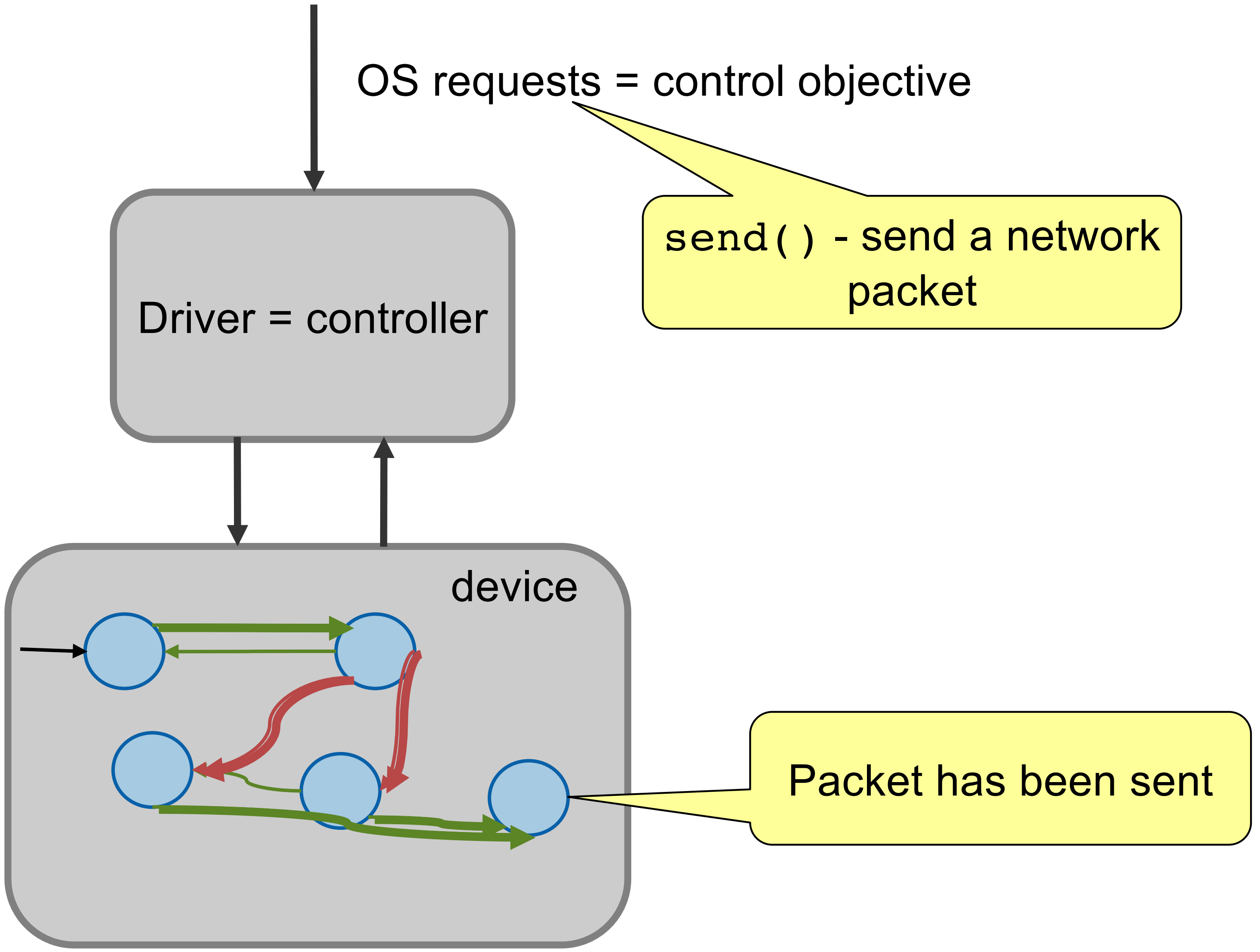
Driver Development



Driver Development

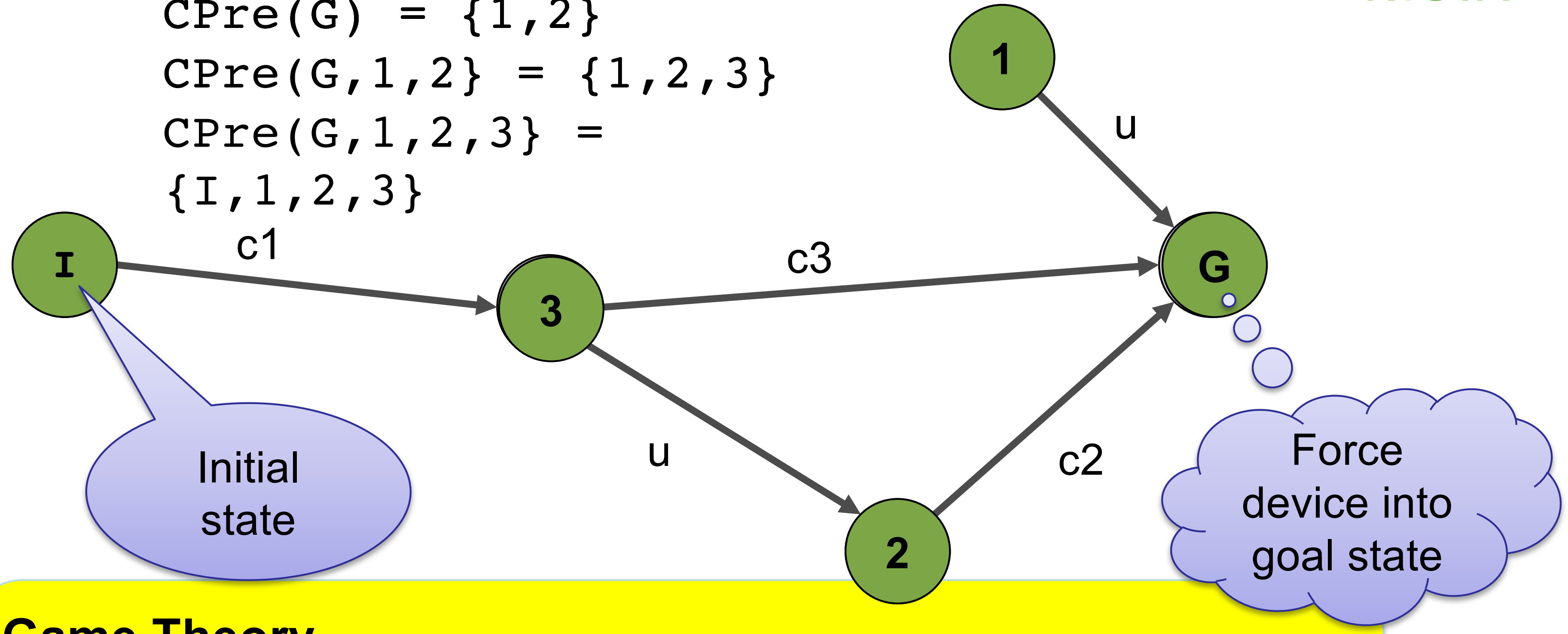


Driver Synthesis as Controller Synthesis


OS requests = control objective
`send()` - send a network packet
Driver = controller
device
Packet has been sent

Synthesis Algorithm (Main Idea)

```
CPre(G) = {1,2}
CPre(G,1,2} = {1,2,3}
CPre(G,1,2,3} =
{I,1,2,3}
```



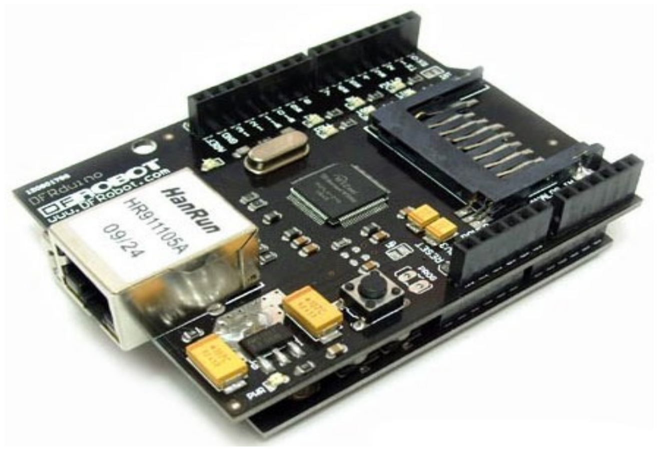
Game Theory

- Framework for verification and synthesis of reactive systems
- Provides classification of games and complexity bounds
- Provides algorithms for winning strategies!

Device driver!

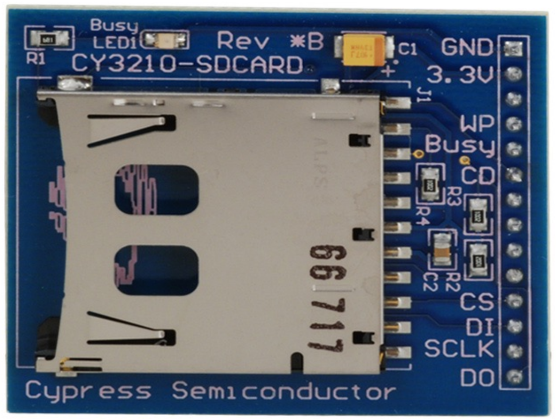
Drivers Synthesised (To Date)

IDE disk controller



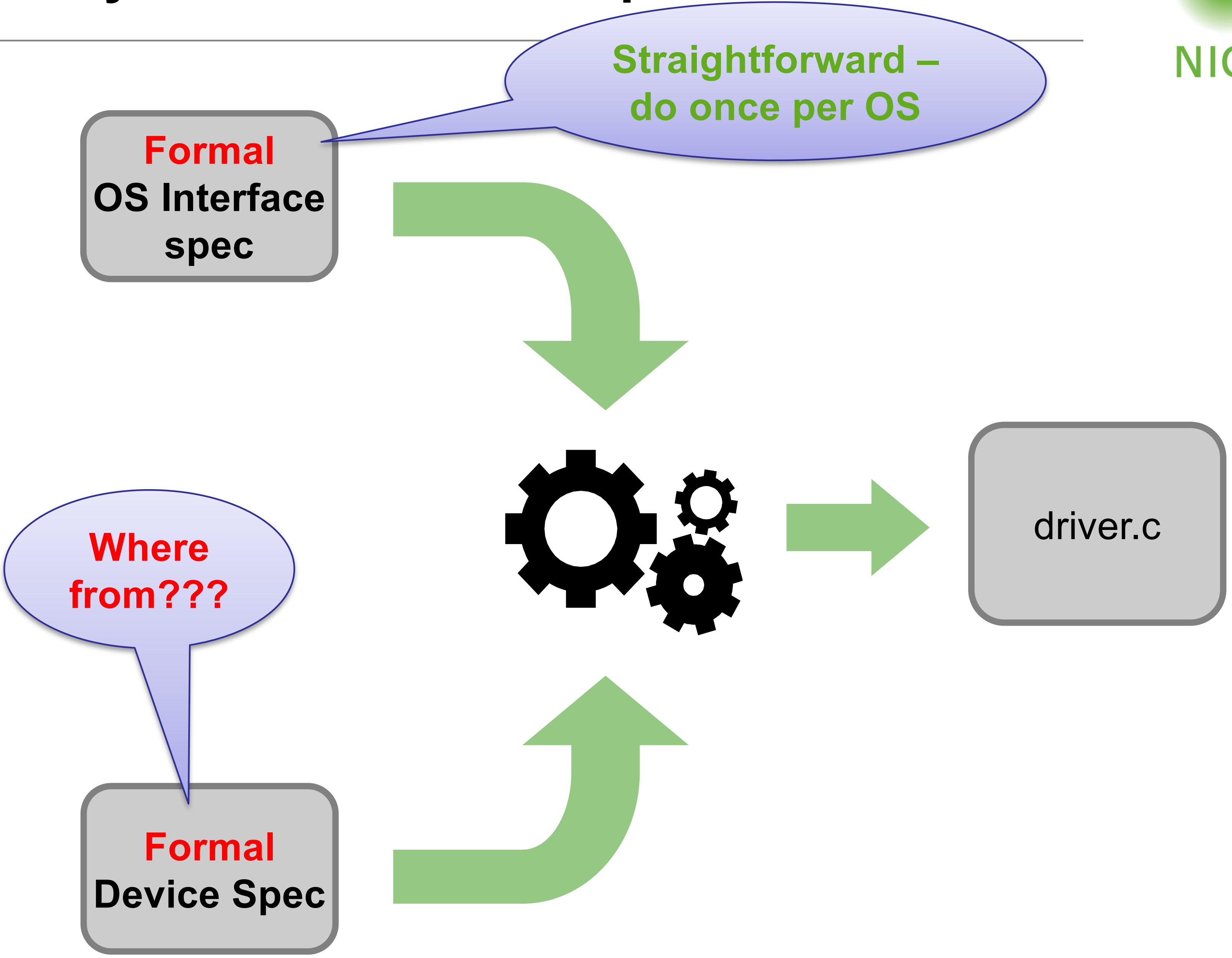
W5100 Eth shield

Asix AX88772
USB-to-Eth adapter



SD host controller

Driver Synthesis: Interface Specs



Hardware Design Workflow

Informal specification

⇩

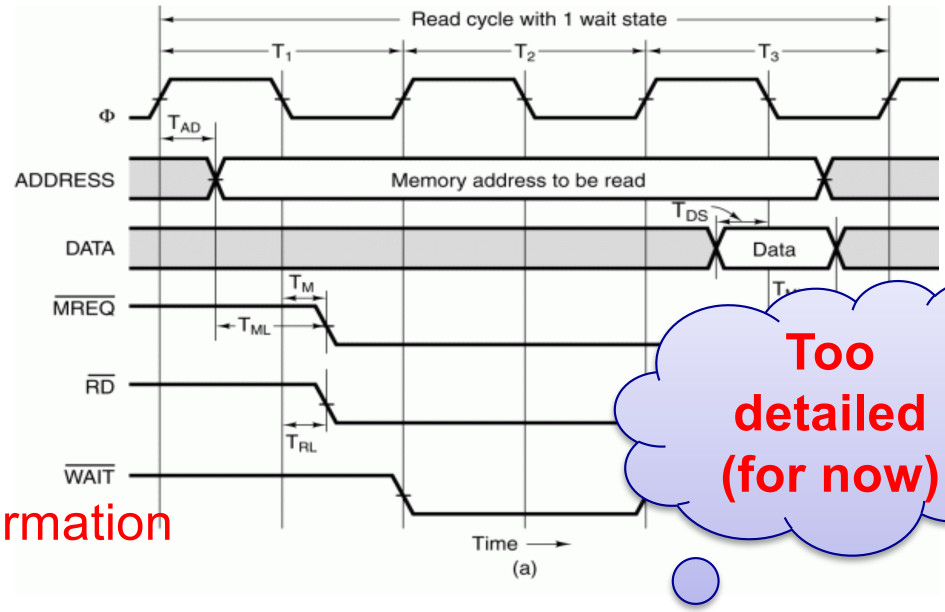
High-level model

⇩ Manual transformation

Register-transfer-level description

⇩

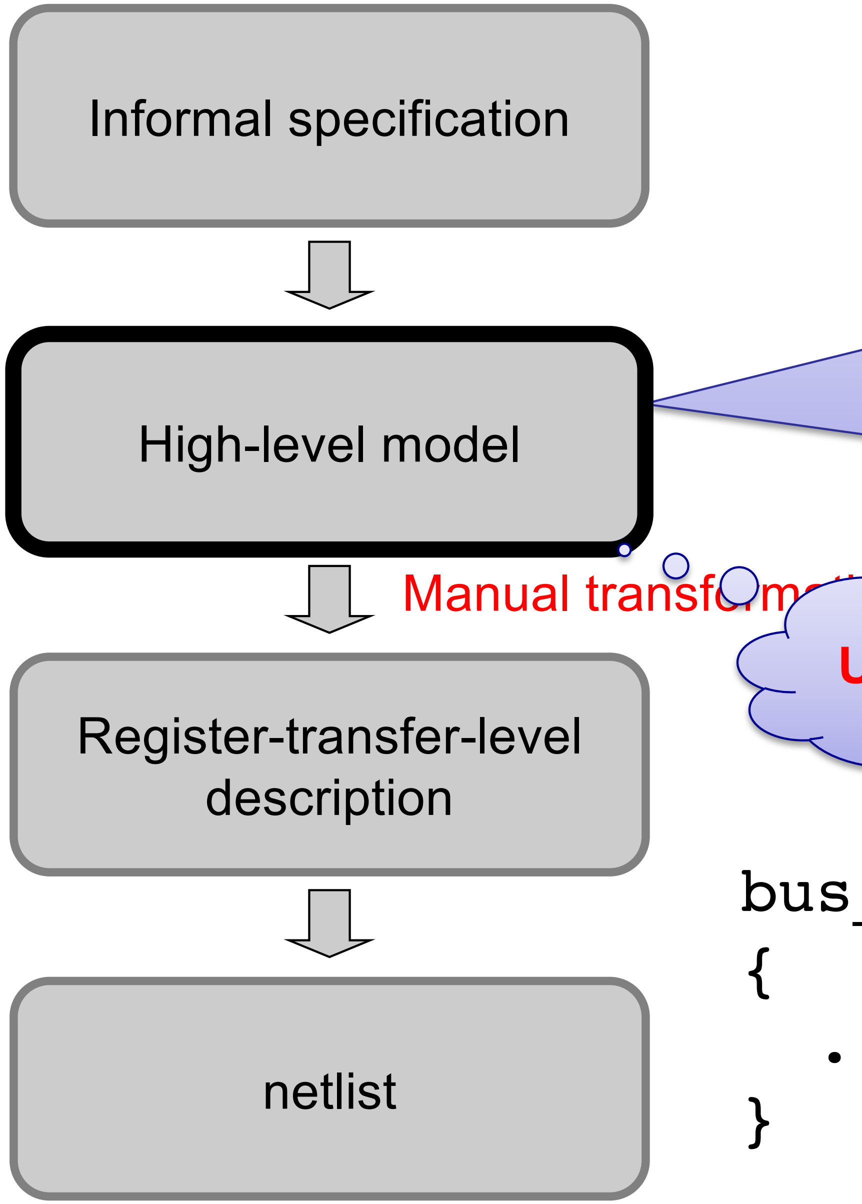
netlist



Too detailed (for now)

- Low-level description: registers, gates, wires.
- Cycle-accurate
- Precisely models internal device architecture and interfaces
- “Gold reference”

Hardware Design Workflow

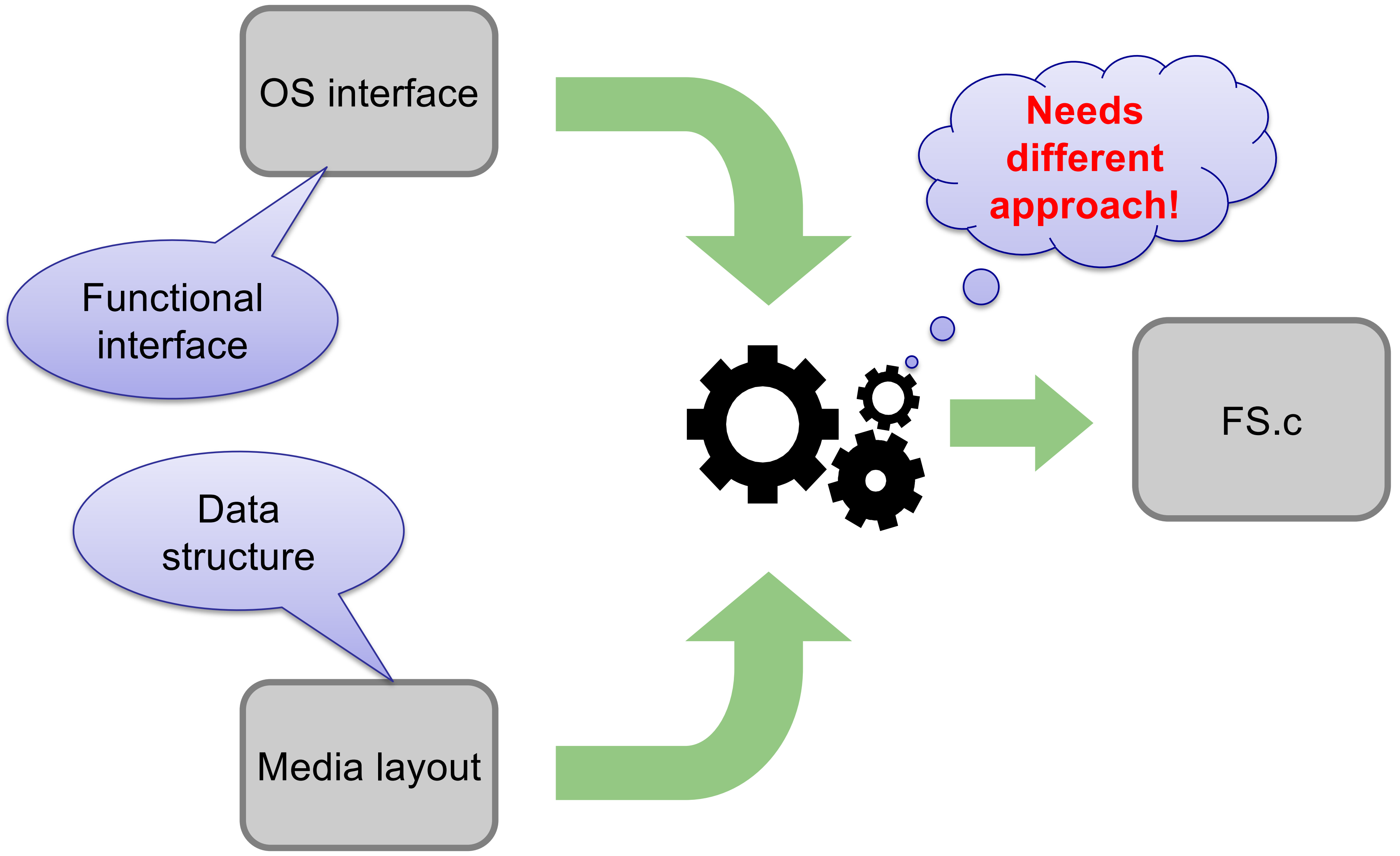


- Captures external behaviour
- Abstracts away structure and timing
- Abstracts away the low-level interface

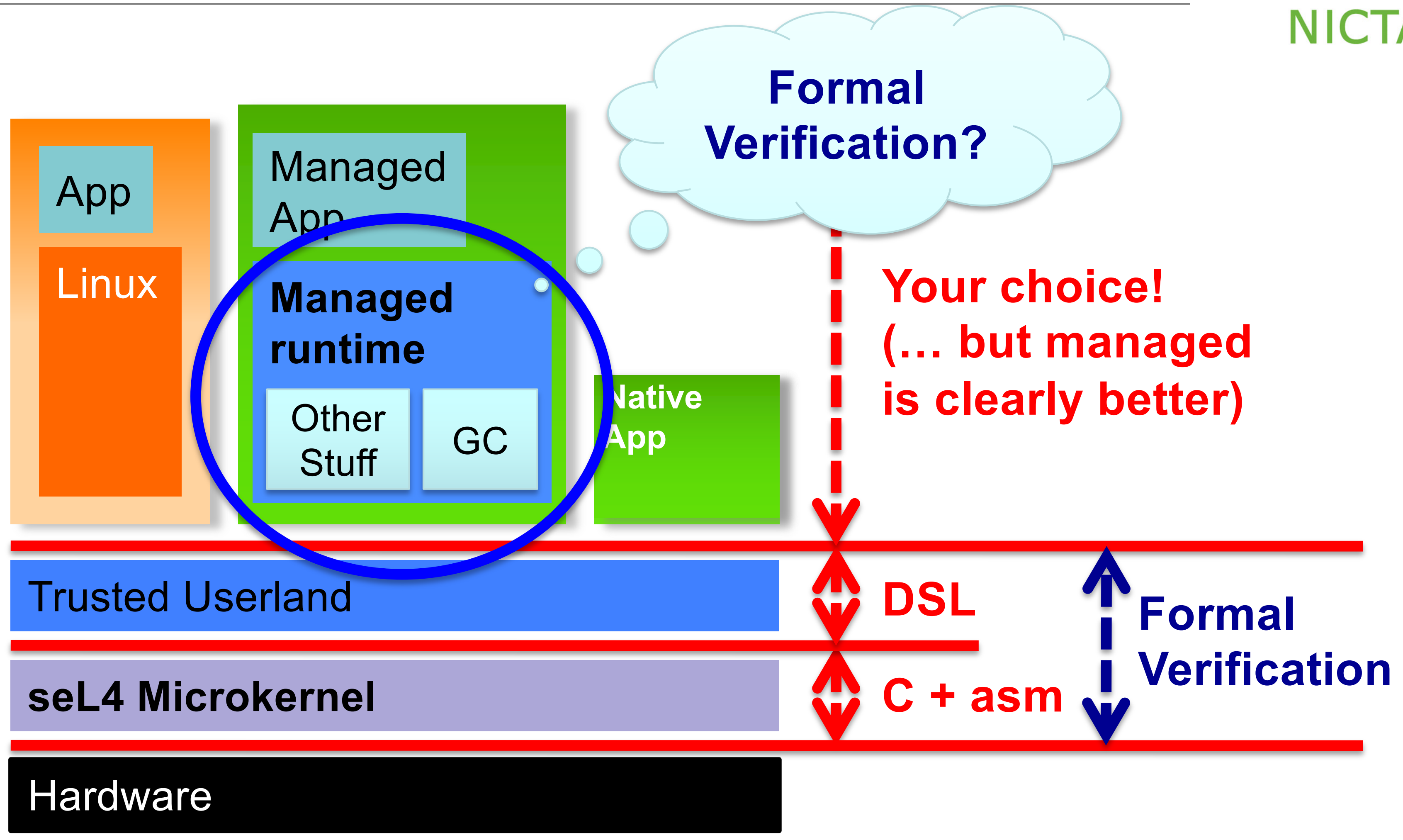
Use for now

```
bus_write(u32 addr, u32 val)
{
   ...
}
```

From Drivers to File Systems?



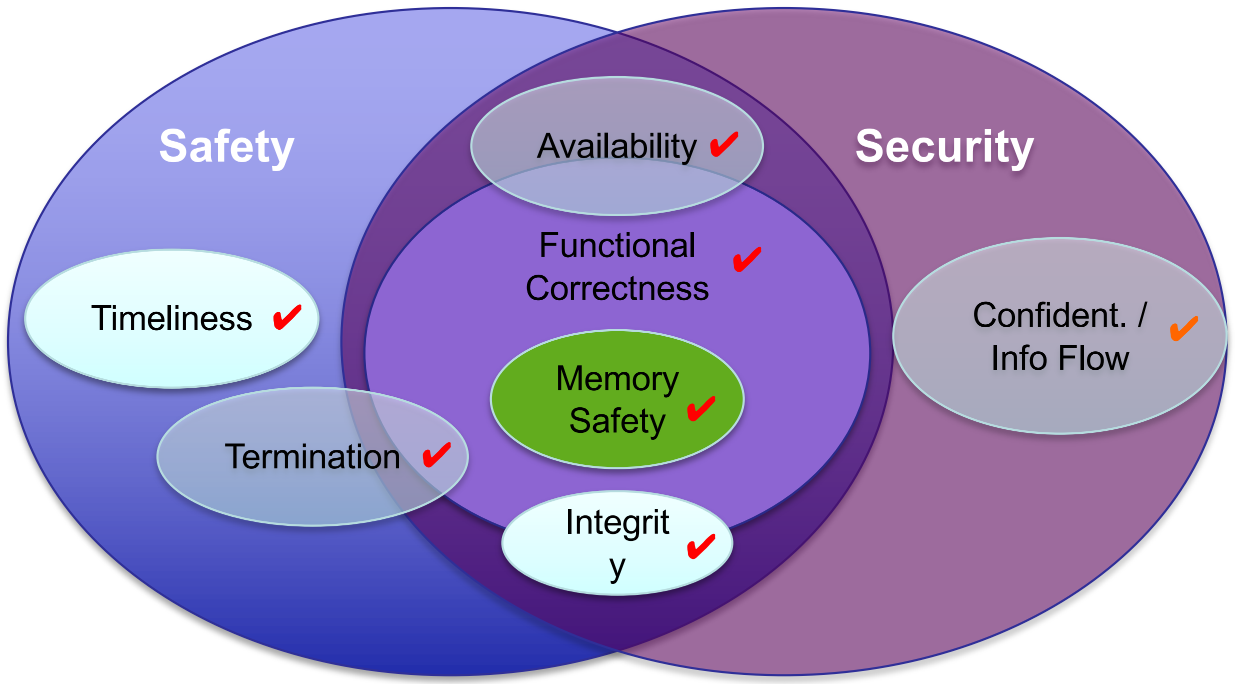
Building Secure Systems: Long-Term View



Core Ingredients: People

Formal Methods Practitioners **Systems Researchers**

Secure Systems Platform: Almost There!



Thank You!

mailto:gernot@nicta.com.au

@GernotHeiser

Google: “nicta trustworthy systems”