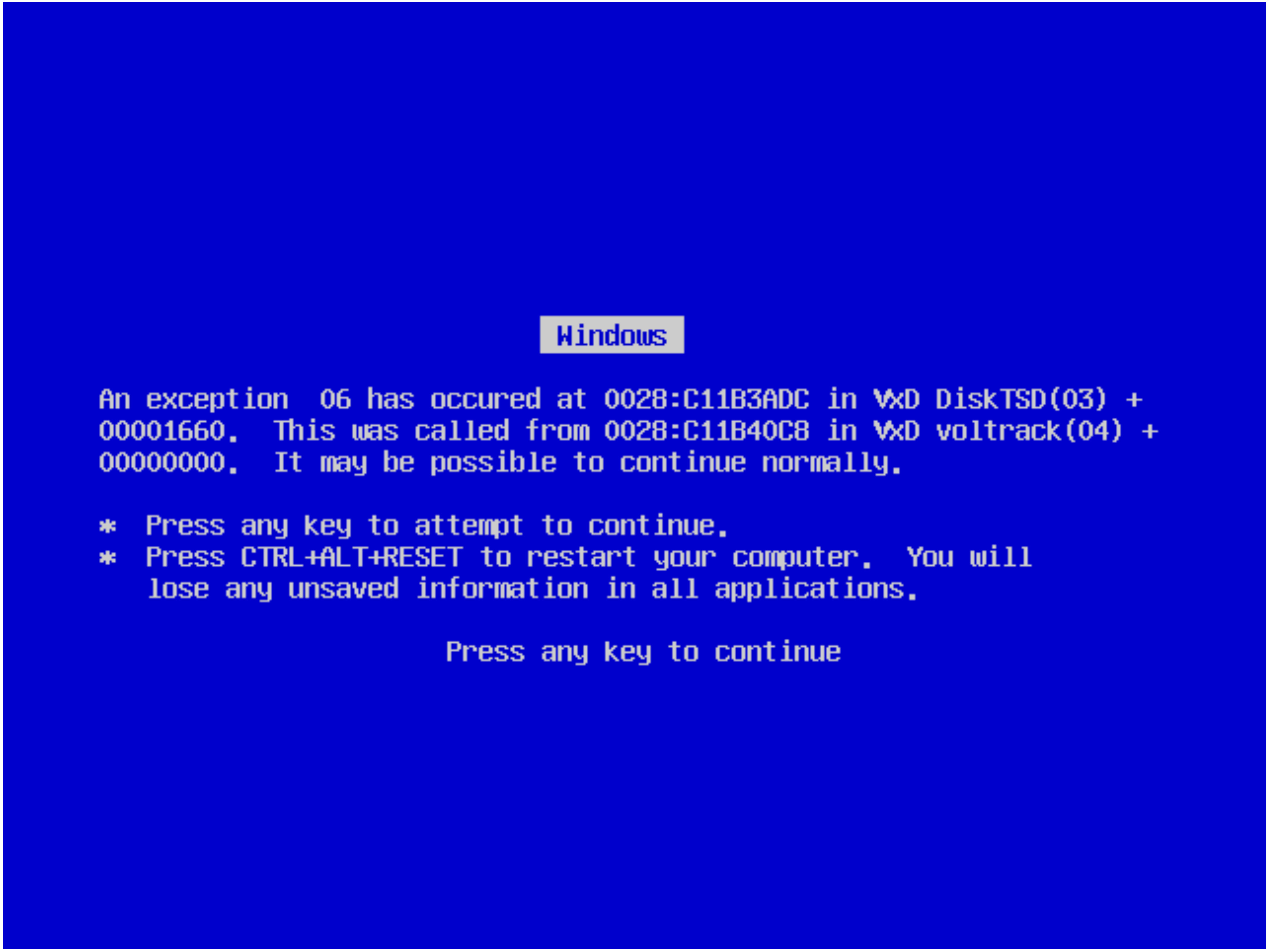
Towards *Trustworthy* Embedded Systems

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



```
                               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* Trustworthy!

NICTA

What's Next?

So, why don't we prove security?

***Claim*:**

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

Corollary [with apologies to Dijkstra]:

Testing, code inspection, etc. can only show *insecurity/unsafety*, not security or safety!

Core Issue: Complexity

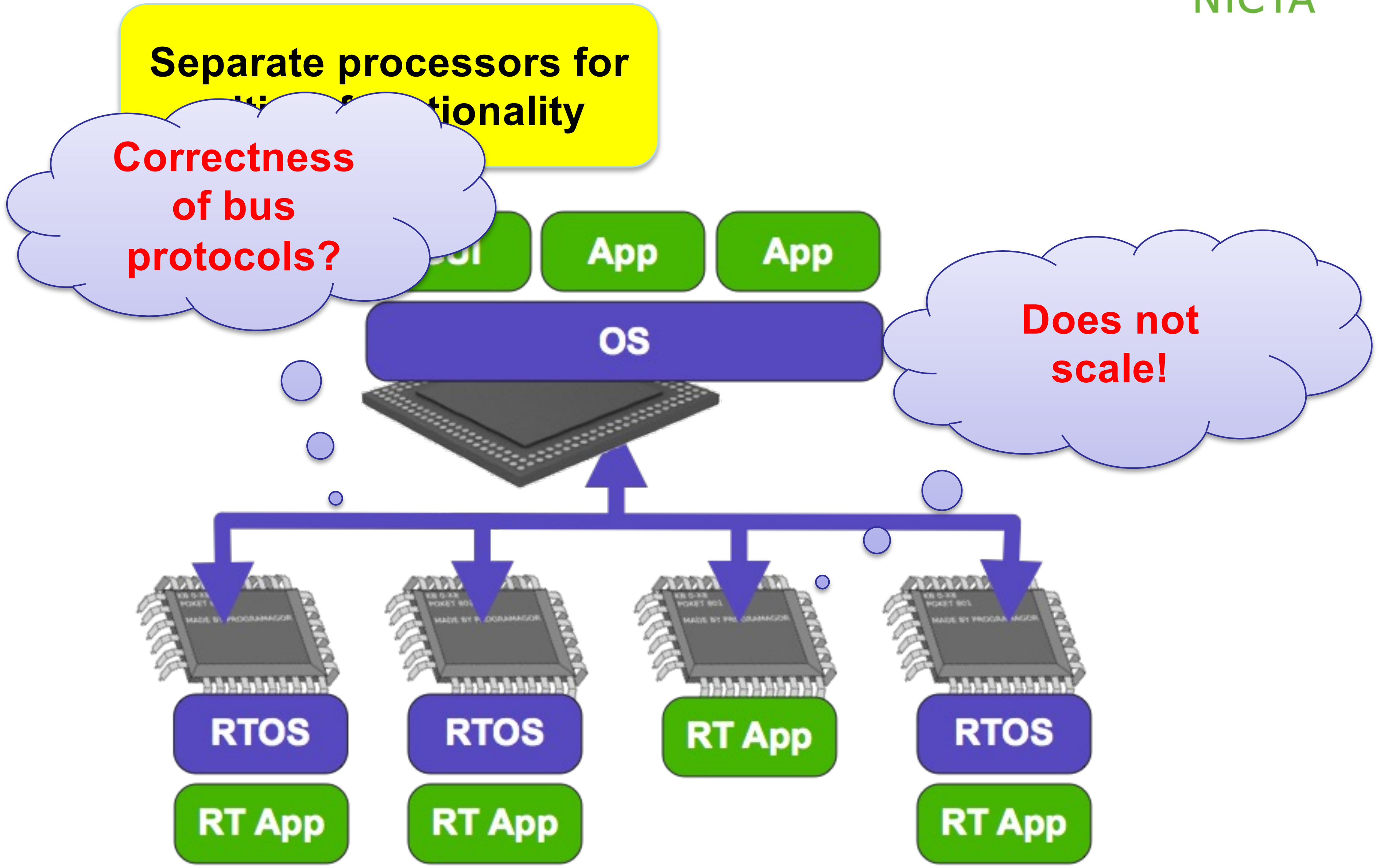
- Massive functionality of C devices
  ⇒ huge software stacks
  - How secure are your paym ?

- 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…

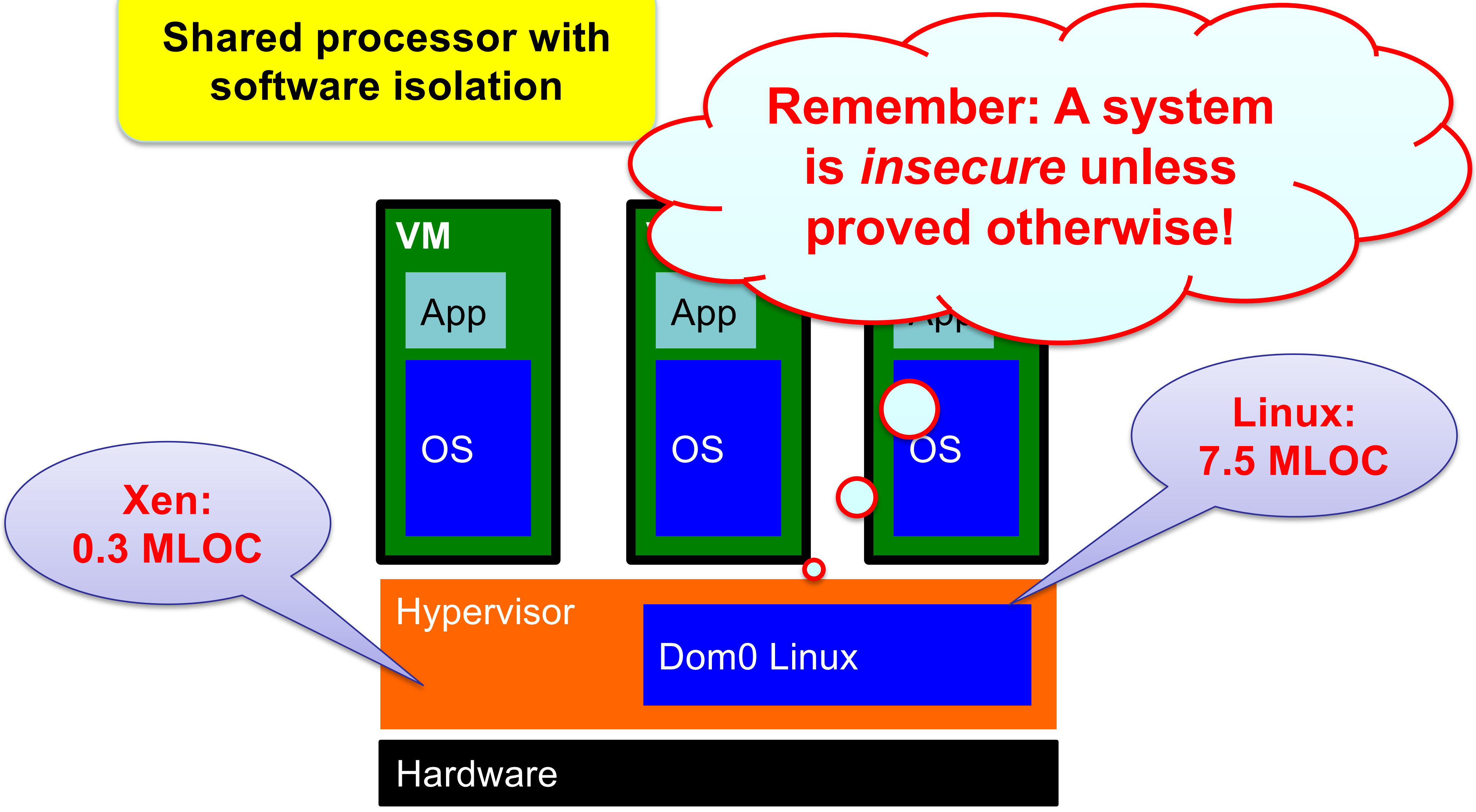
Systems far too complex to prove their trustworthiness!

Dealing with Complexity: Physical Isolation


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

How About Logical Isolation?

Shared processor with software isolation



Our Vision: Trustworthy Systems

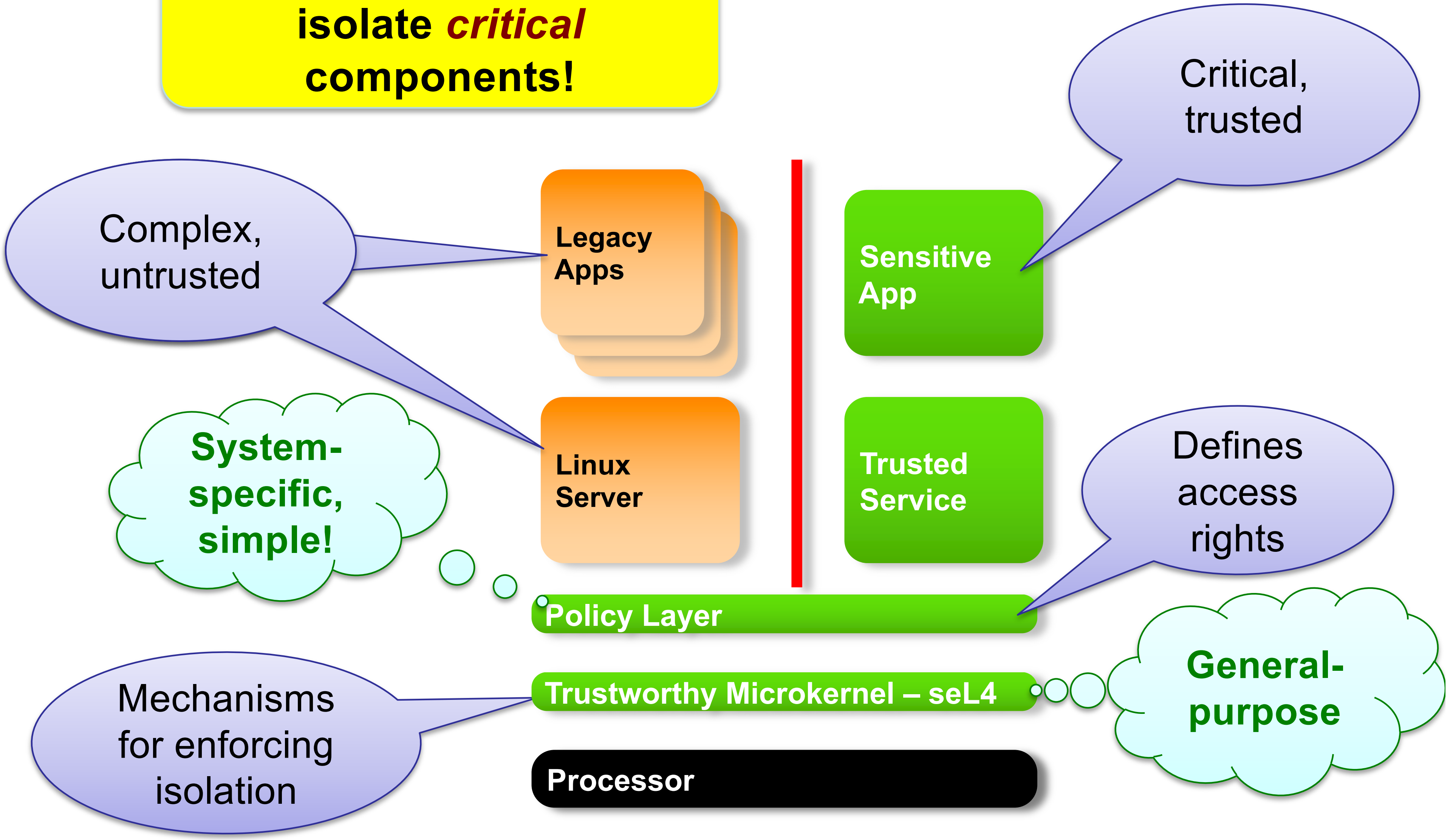
Suitable for real-world systems

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

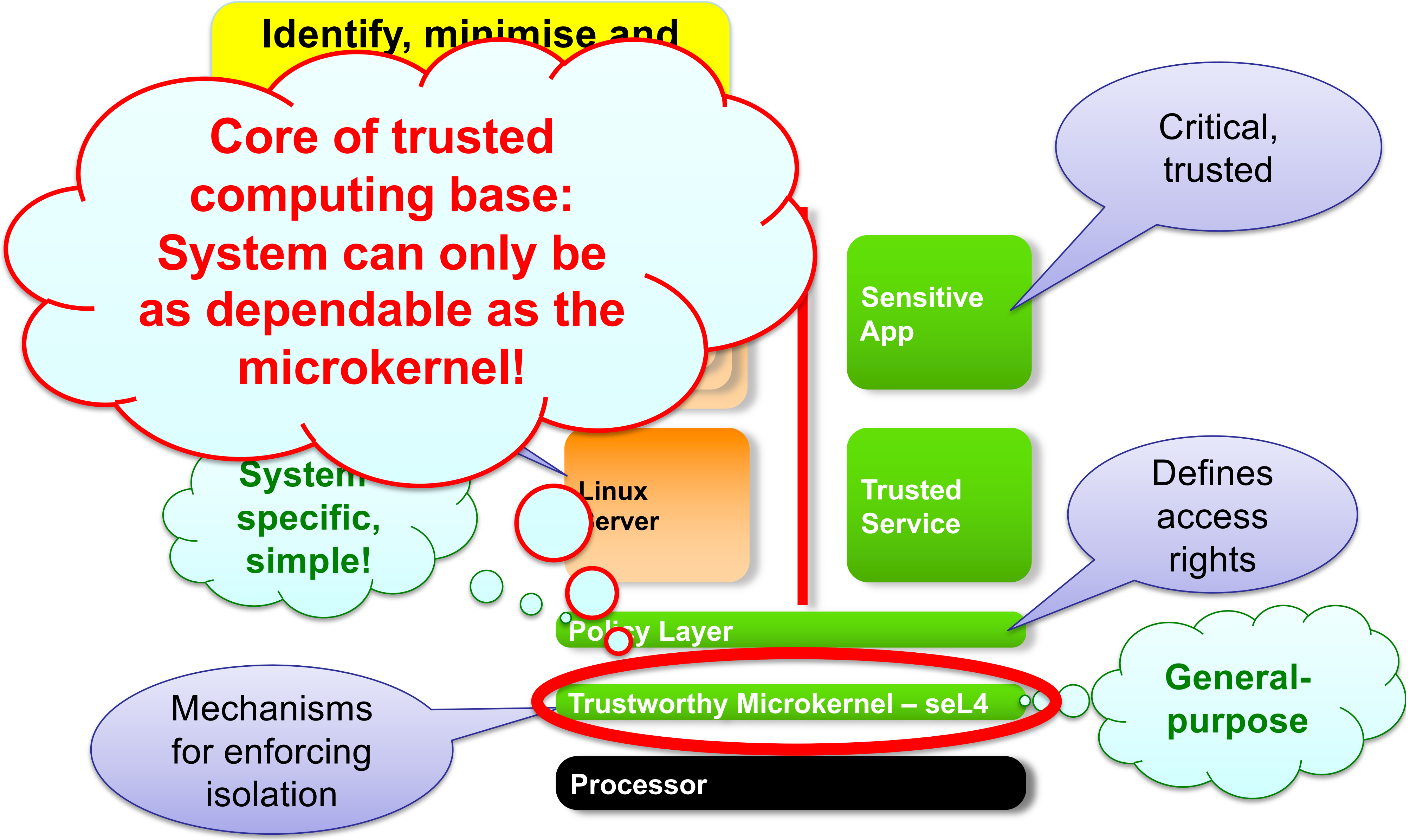
Hard *guarantees* on safety/security/ reliability

Isolation is Key!

Identify, minimise and isolate *critical* components!



Isolation is Key!

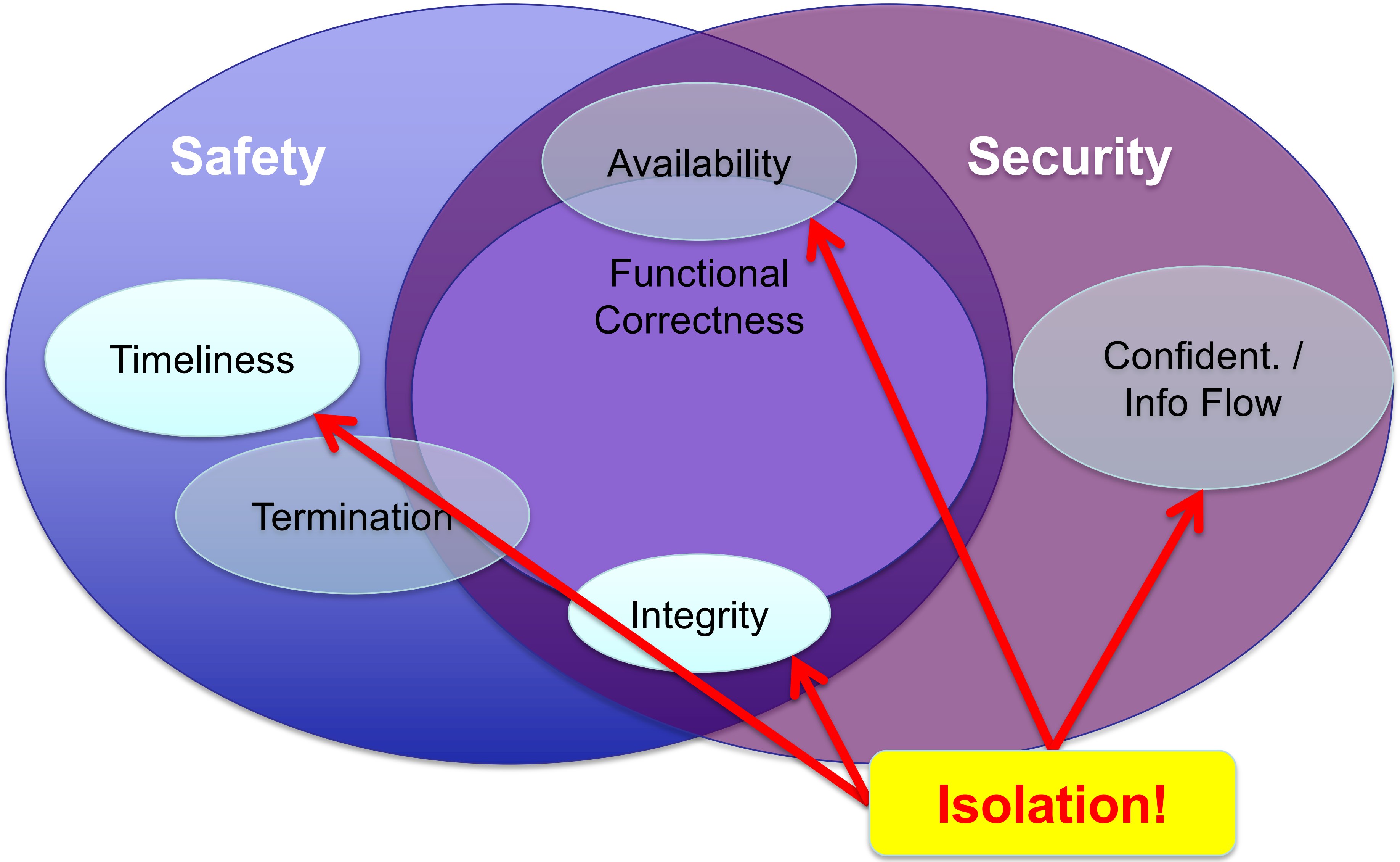


NICTA Trustworthy Systems Agenda

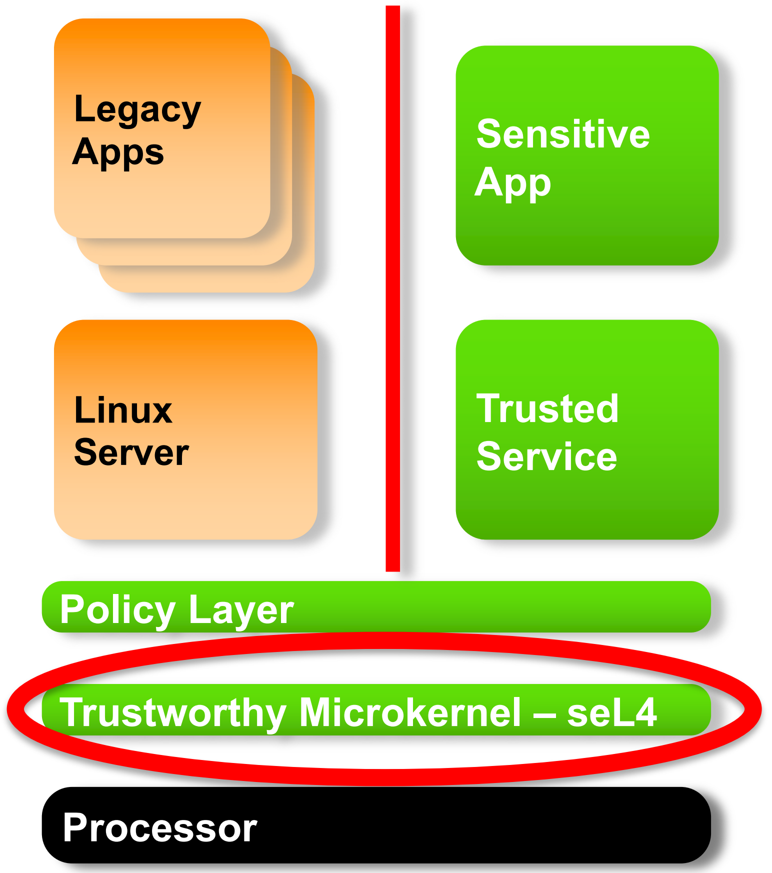
1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

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
     - despite 99 % of code untrusted!

Requirements for Trustworthy Systems



seL4 Design Goals



1. Isolation
   - Strong partitioning!
2. Formal verification
   - Provably trustworthy!
3. Performance
   - Suitable for real world!

Fundamental Design Decisions for seL4

1. Memory management is user-level responsibility
   - Kernel never allocates memory (post-boot)
   - Kernel objects controlled by user-mode servers

Isolation

2. Memory management is fully delegatable
   - Supports hierarchical system design
   - Enabled by capability-based access control

Perfor-
mance

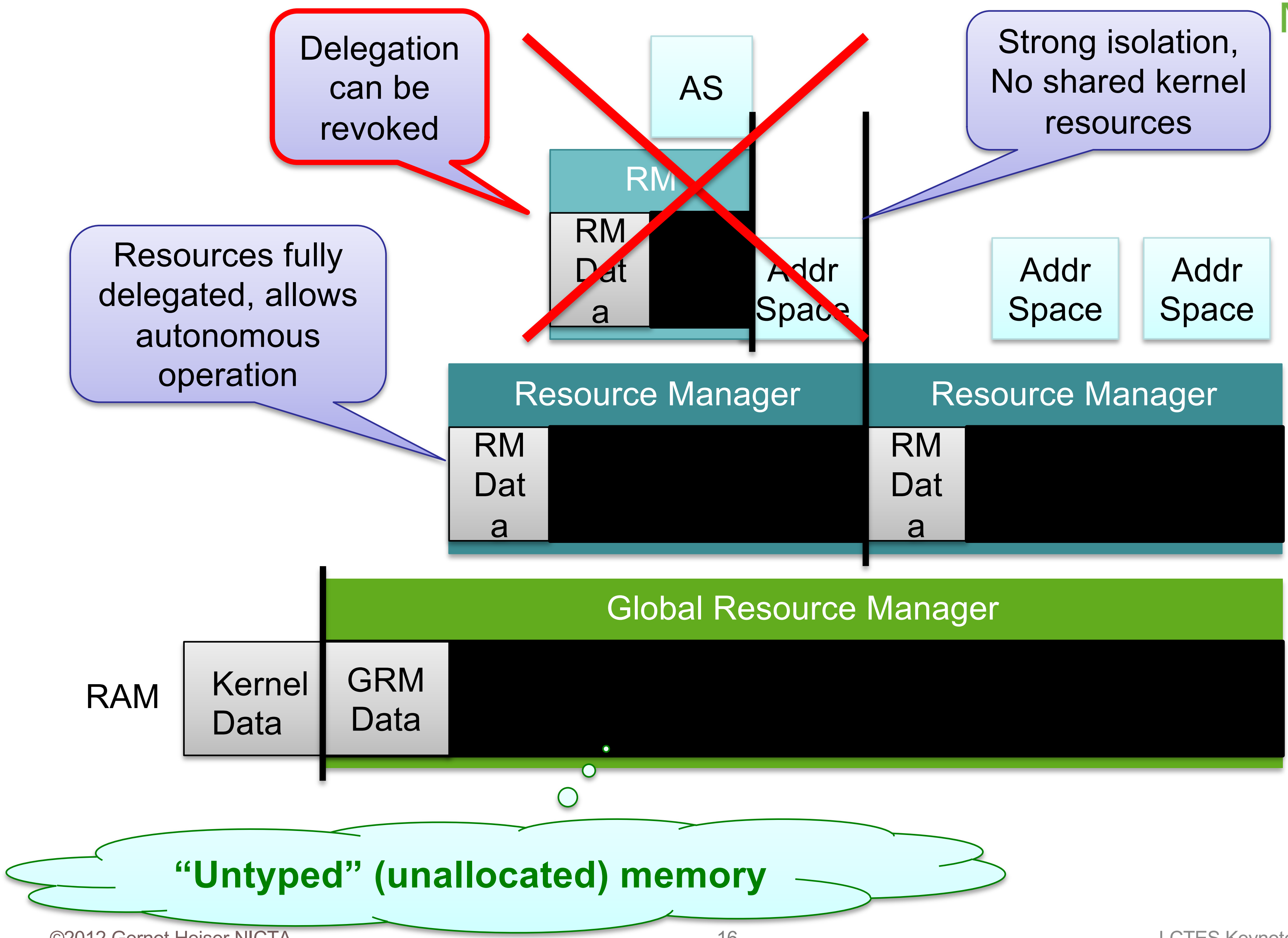
3. “Incremental consistency” design pattern
   - Fast transitions between consistent states
   - Restartable operations with progress guarantee

Real-time

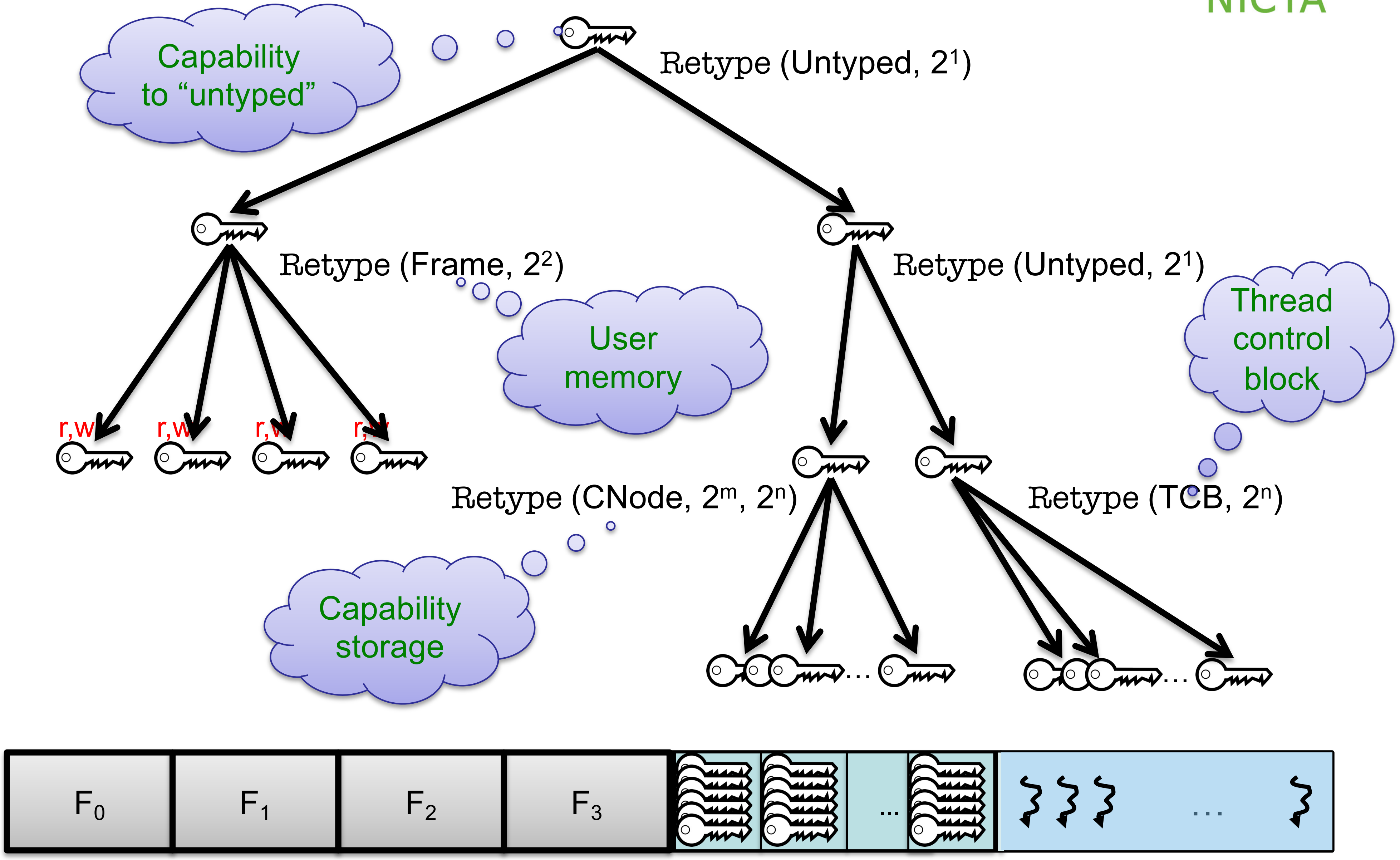
4. No concurrency in the kernel
   - Interrupts never enabled in kernel
   - Interruption points to bound latencies
   - Clustered multikernel design for multicores

Verification

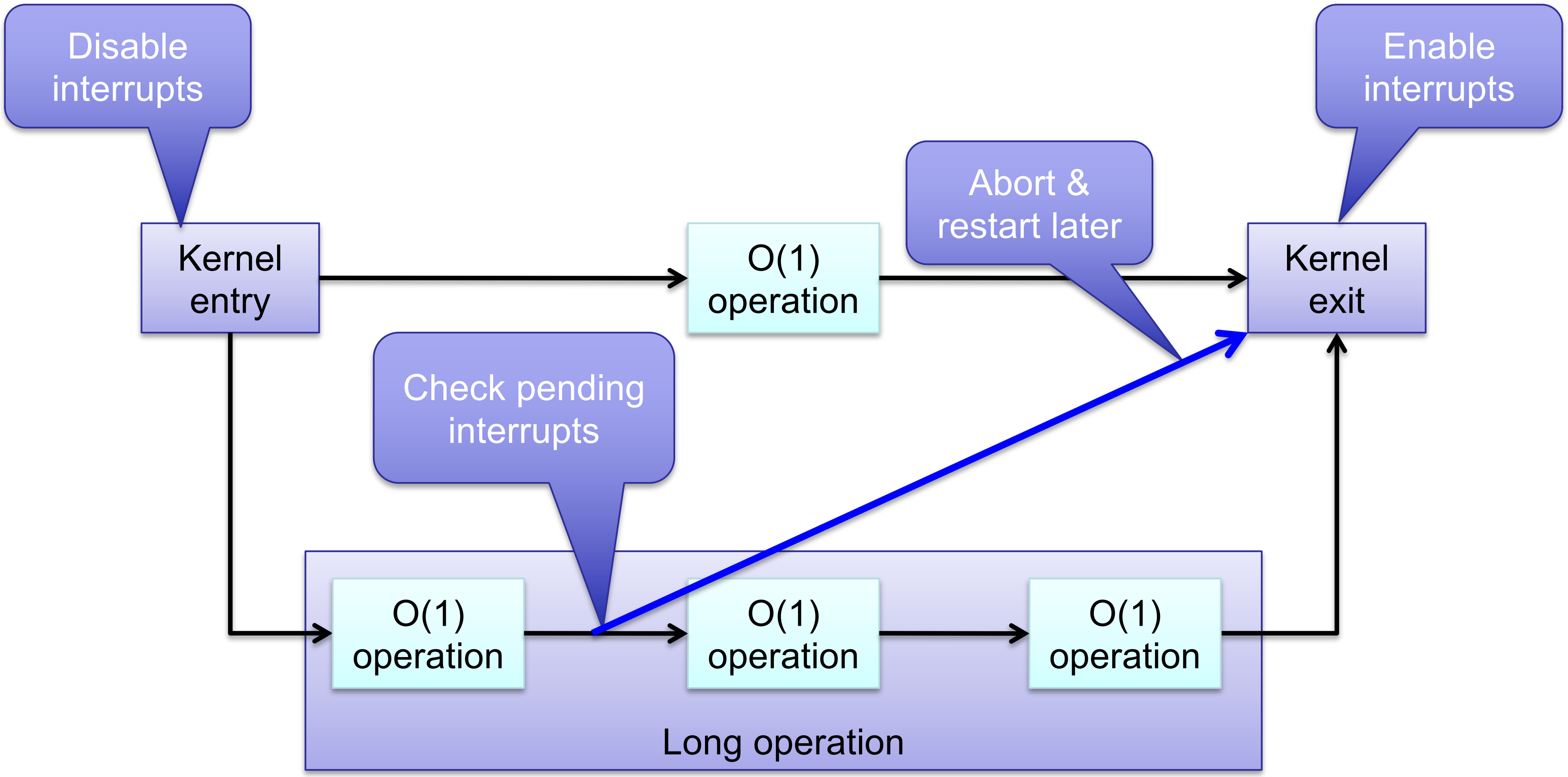
seL4 User-Level Memory Management


Delegation can be revoked
AS
RM
RM Dat a
Addr Space
Strong isolation, No shared kernel resources
Addr Space
Addr Space
Resources fully delegated, allows autonomous operation
Resource Manager
Resource Manager
RM Dat a
RM Dat a
Global Resource Manager
RAM
Kernel Data
GRM Data
"Untyped" (unallocated) memory

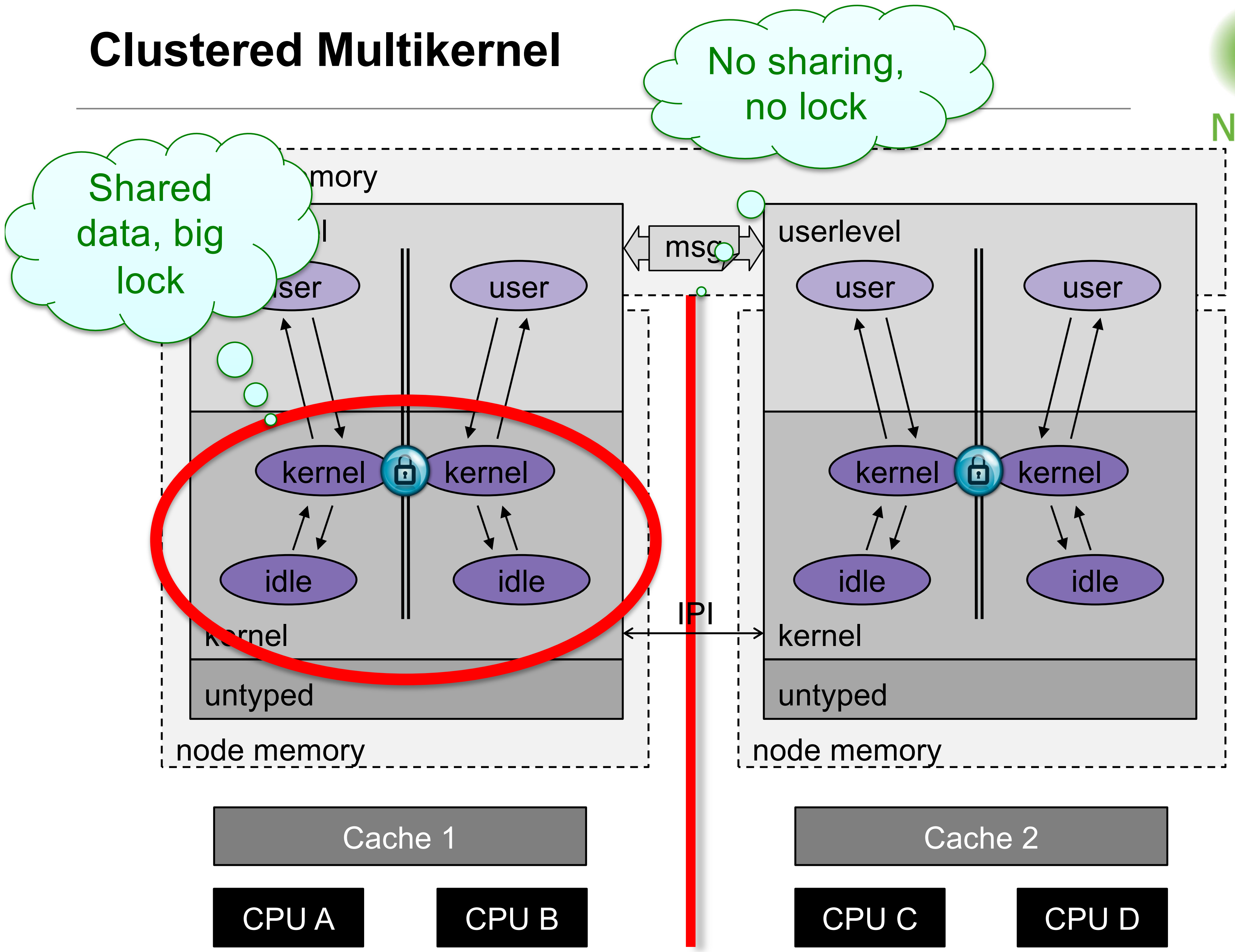
seL4 Memory Management Mechanics: **Retype**



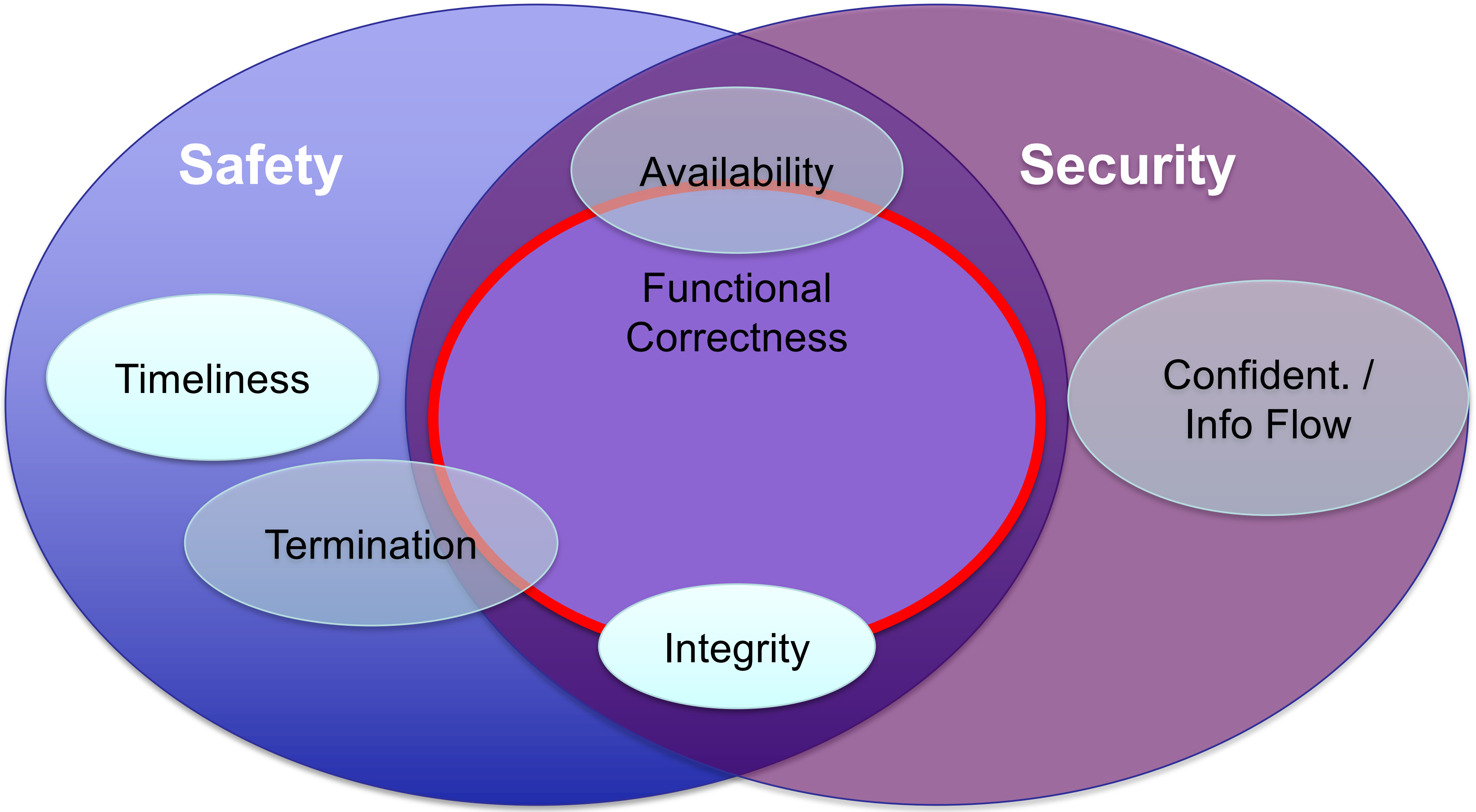
Incremental Consistency



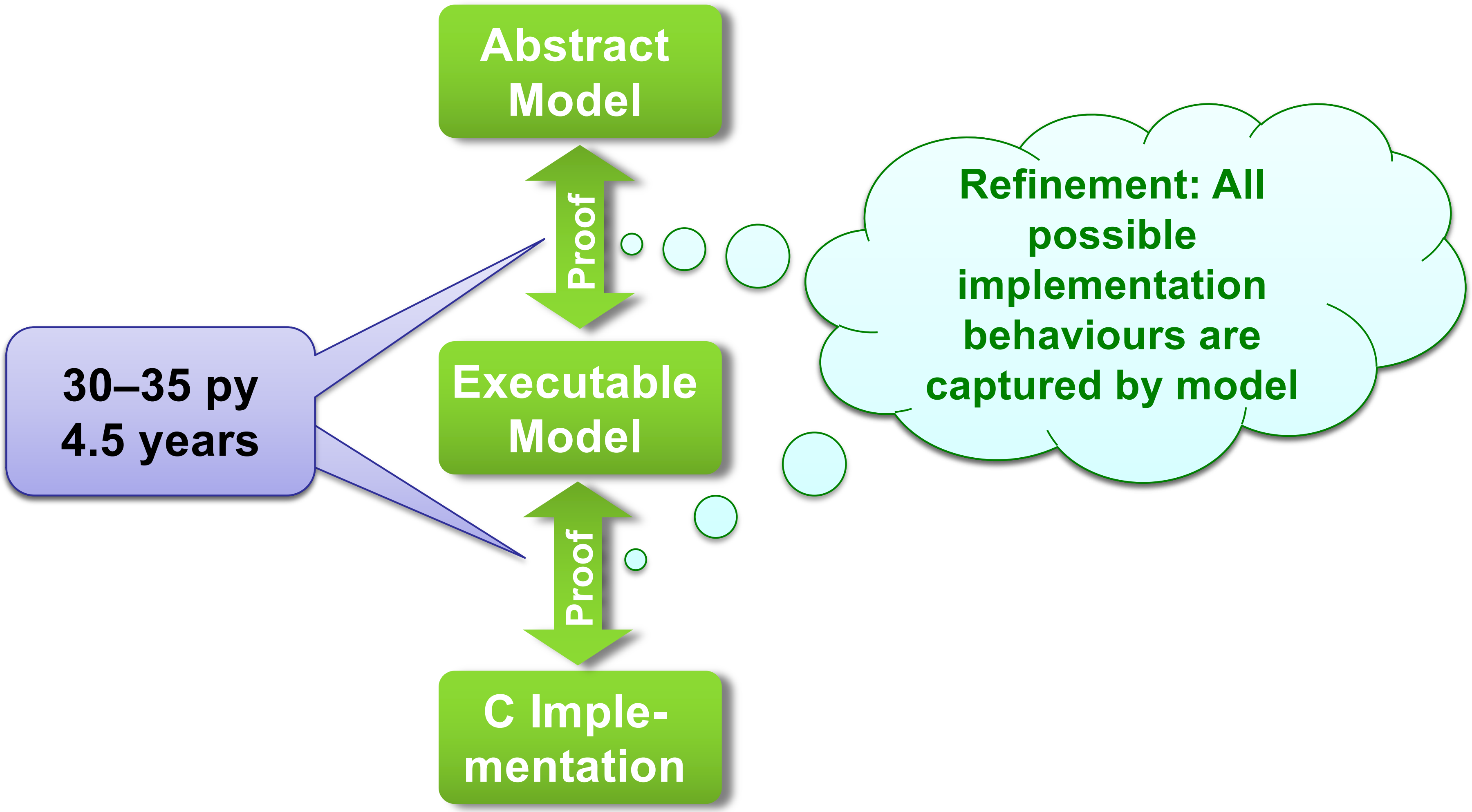
Clustered Multikernel



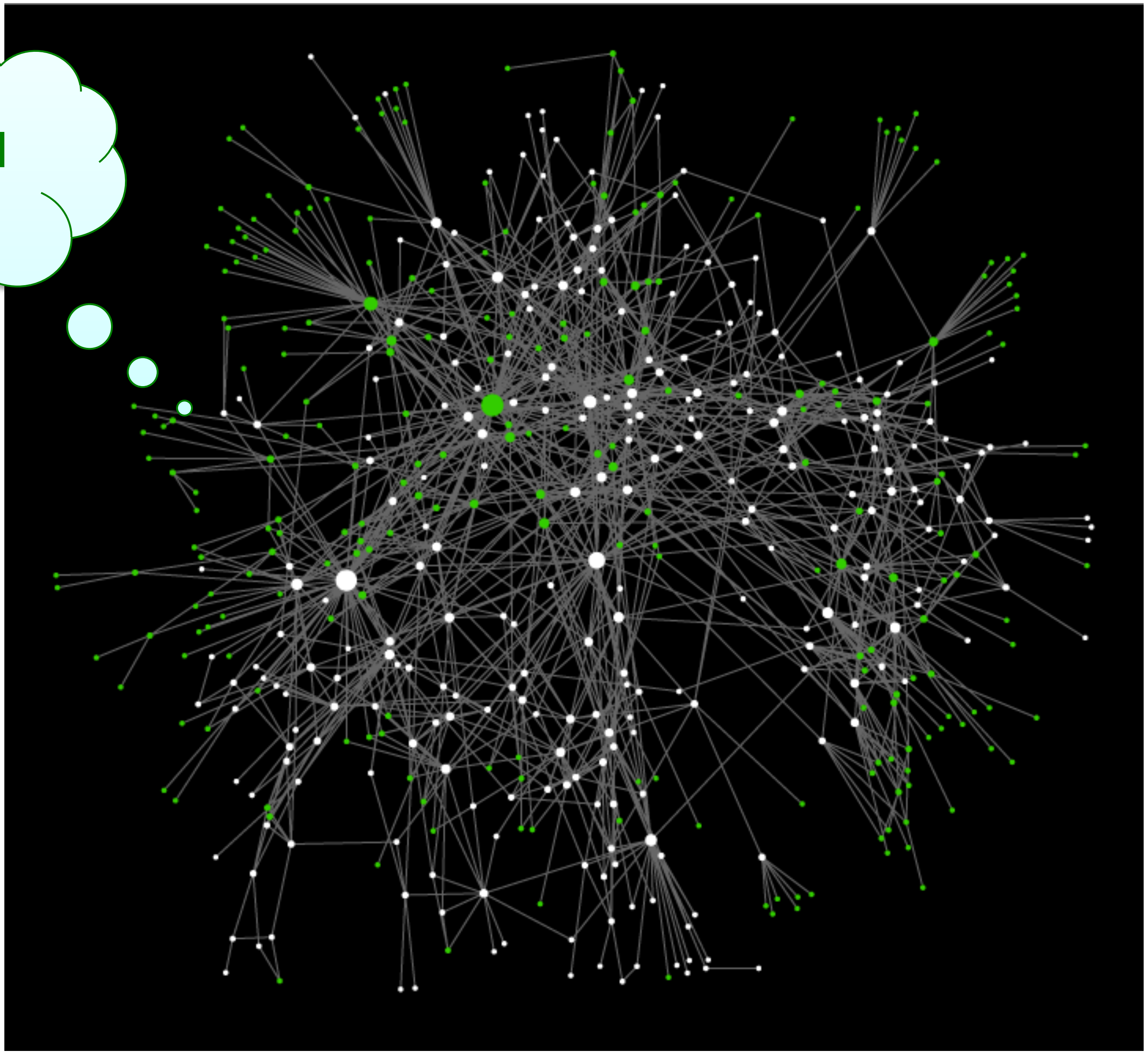
seL4 as Basis for Trustworthy Systems



Proving Functional Correctness

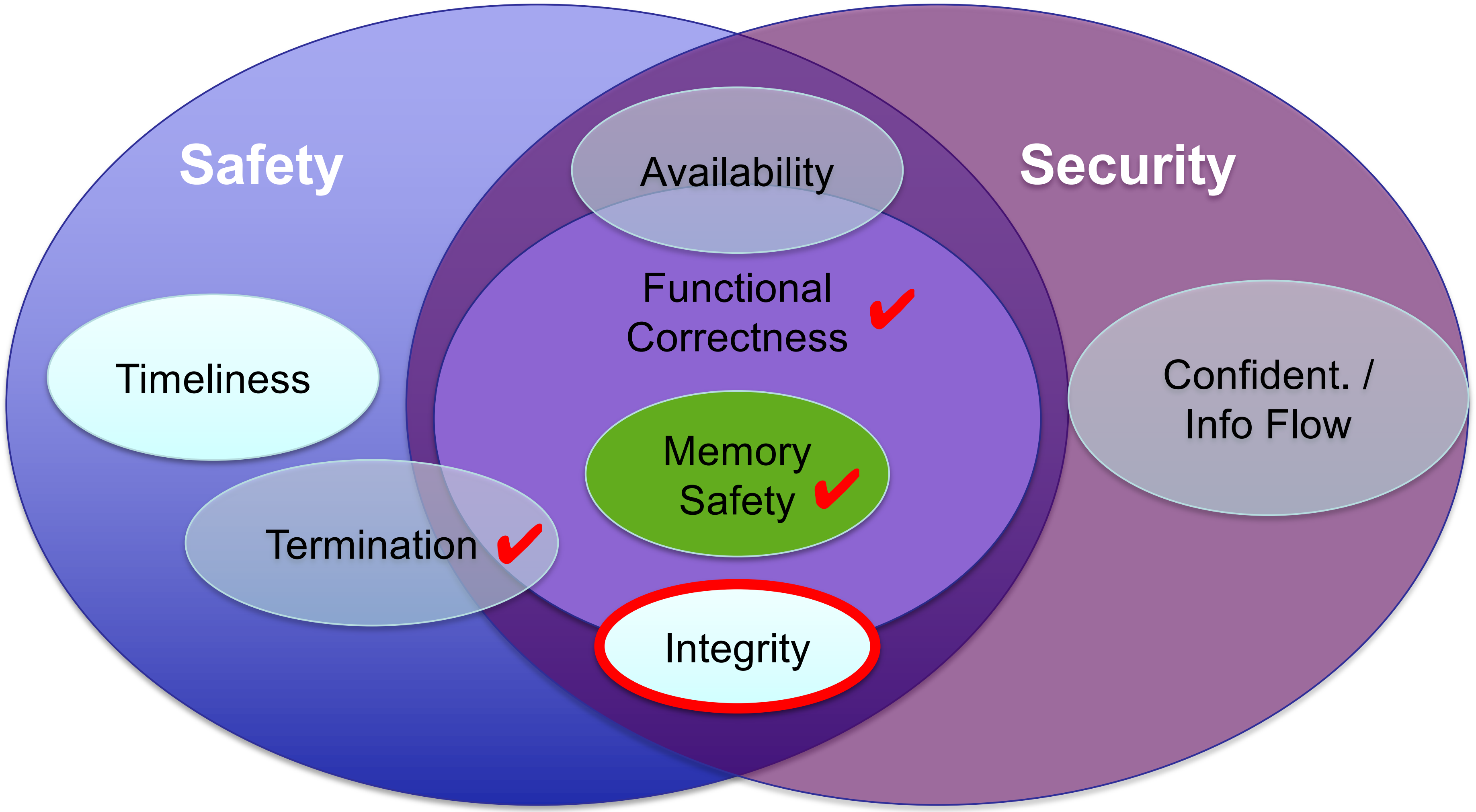


Why So Long for 9,000 LOC?

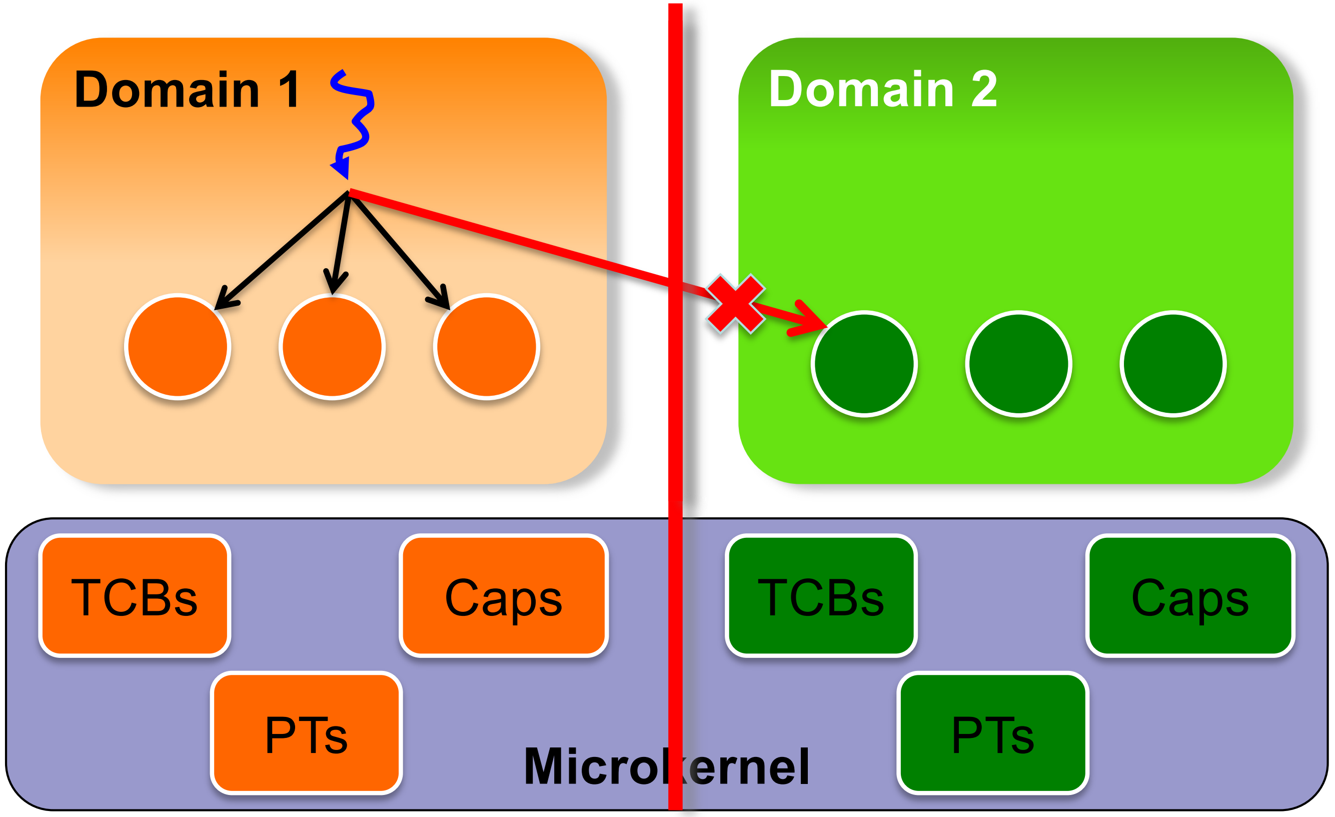


seL4 call graph

seL4 as Basis for Trustworthy Systems



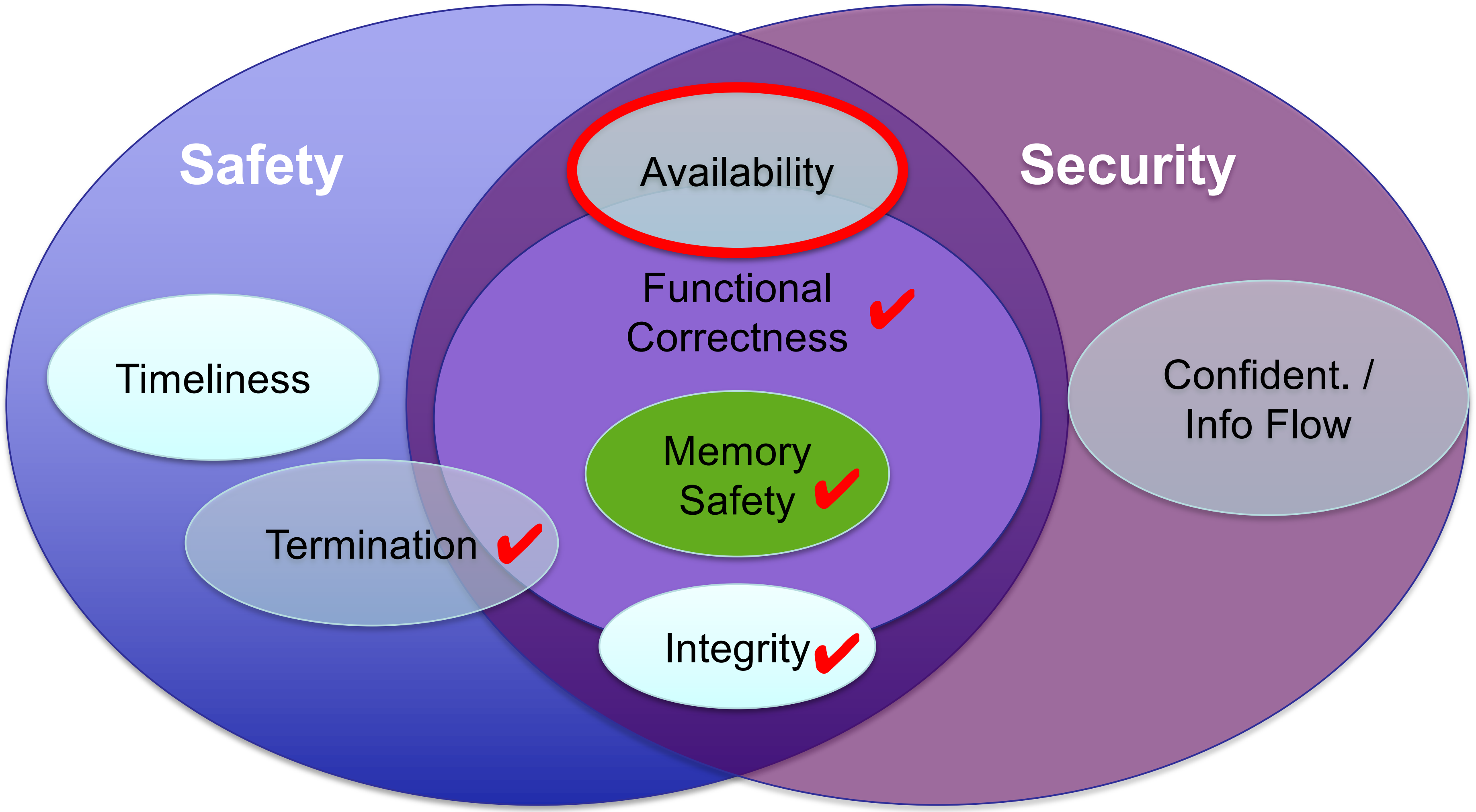
Integrity: Limiting Write Access



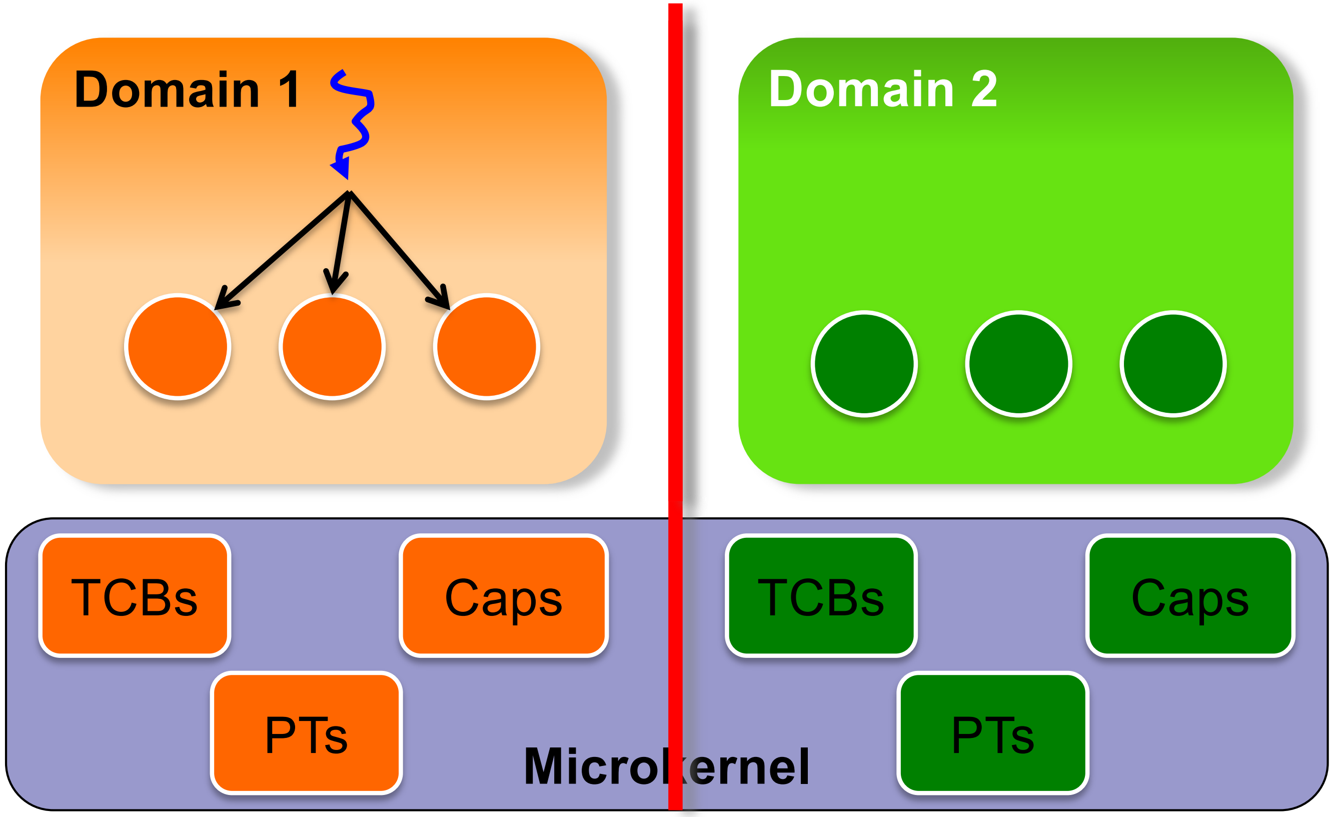
To prove:

- 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!*
  – Prove kernel only allows write upon capability presentation

seL4 as Basis for Trustworthy Systems

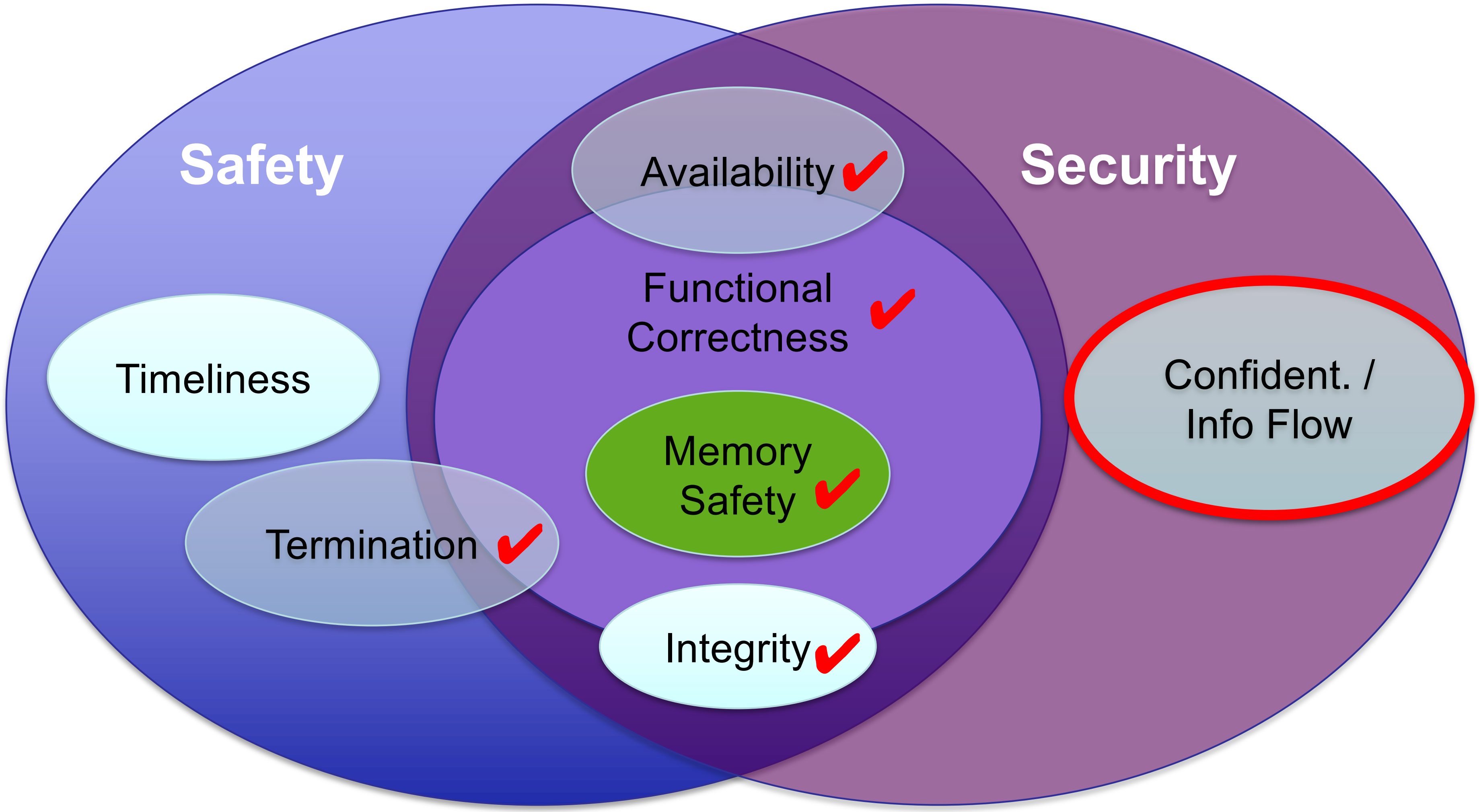


Availability: Ensuring Resource Access

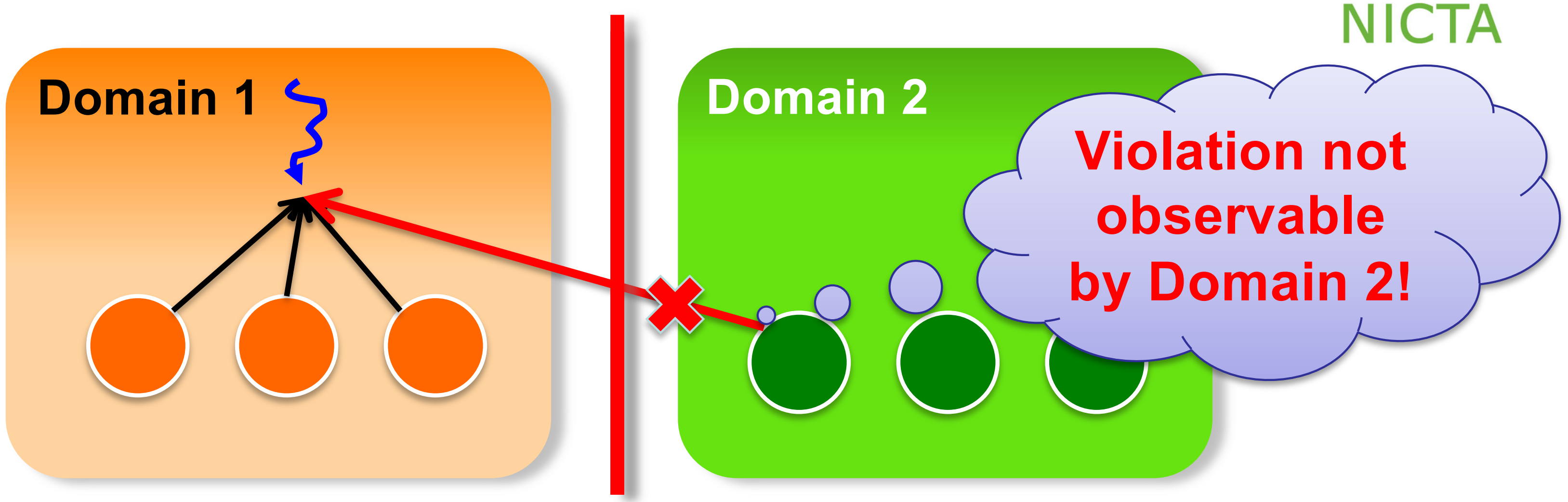


- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources

seL4 as Basis for Trustworthy Systems



Confidentiality: Limiting Read Accesses



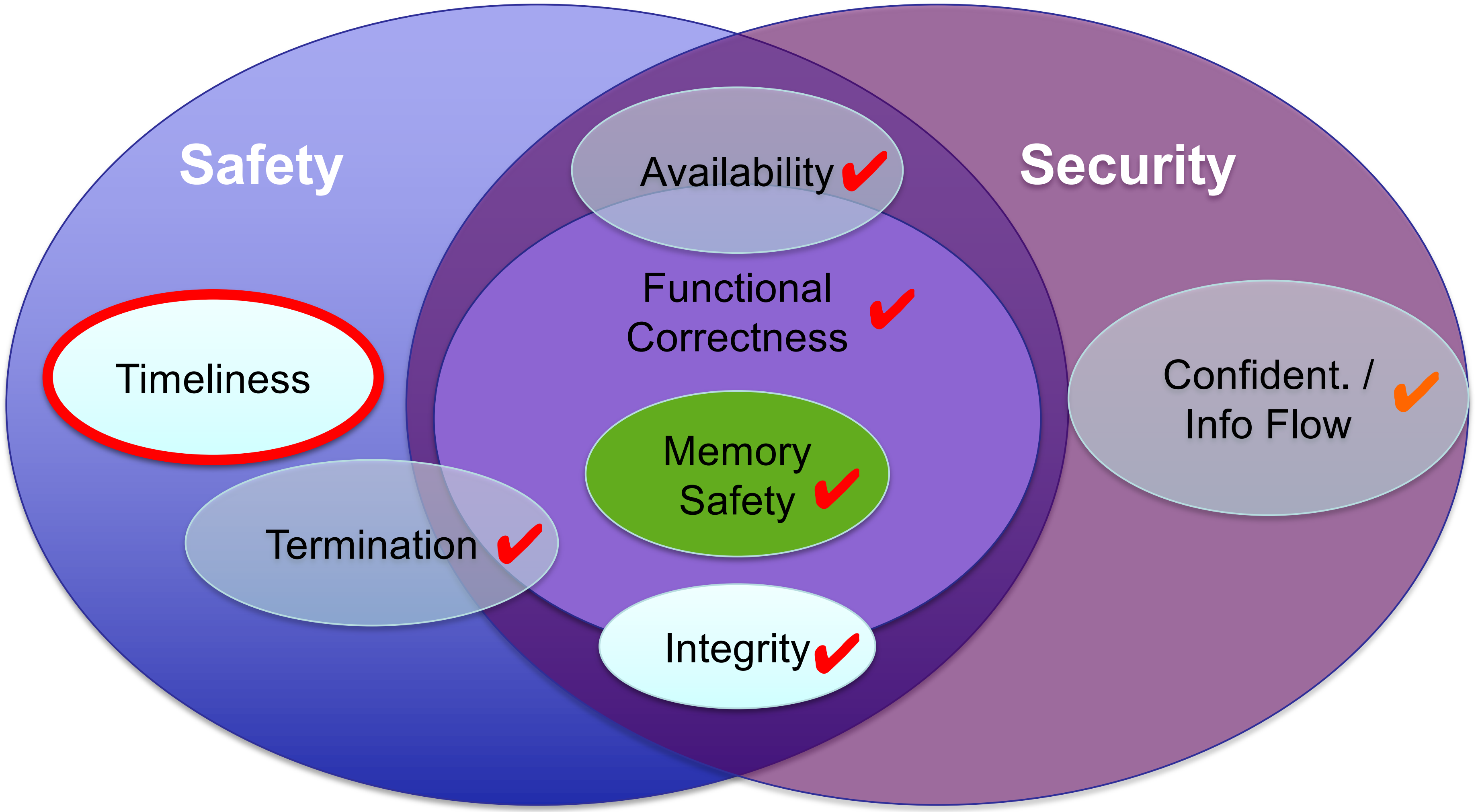
To prove:

- Domain-1 doesn't have read capabilities to Domain-2 objects
  ⇒ no action of any agents will reveal Domain-2 state to Domain-1

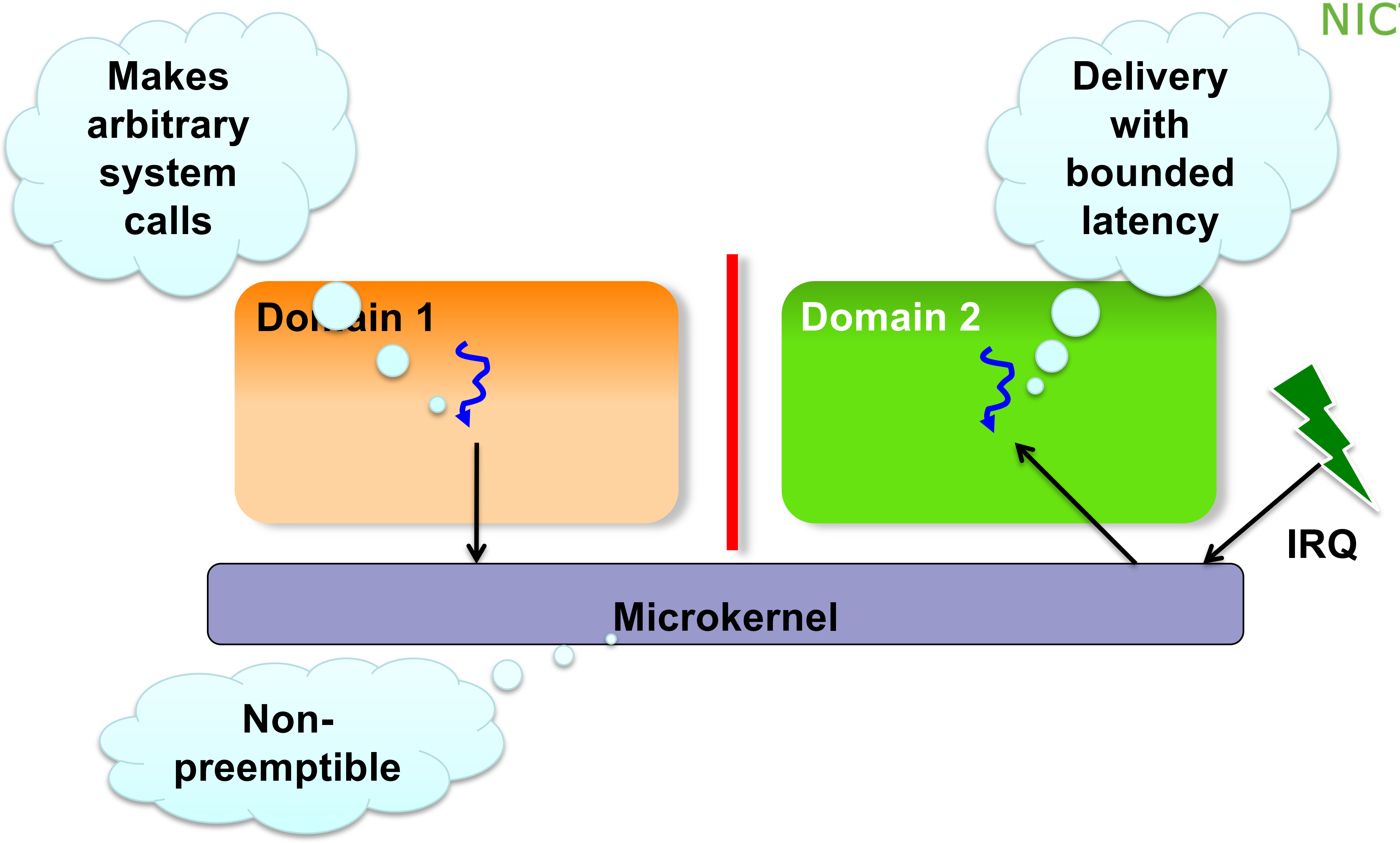
Non-interference proof in progress:

- Evolution of Domain 1 does not depend on Domain-2 state
- Presently cover only overt information flow

seL4 as Basis for Trustworthy Systems

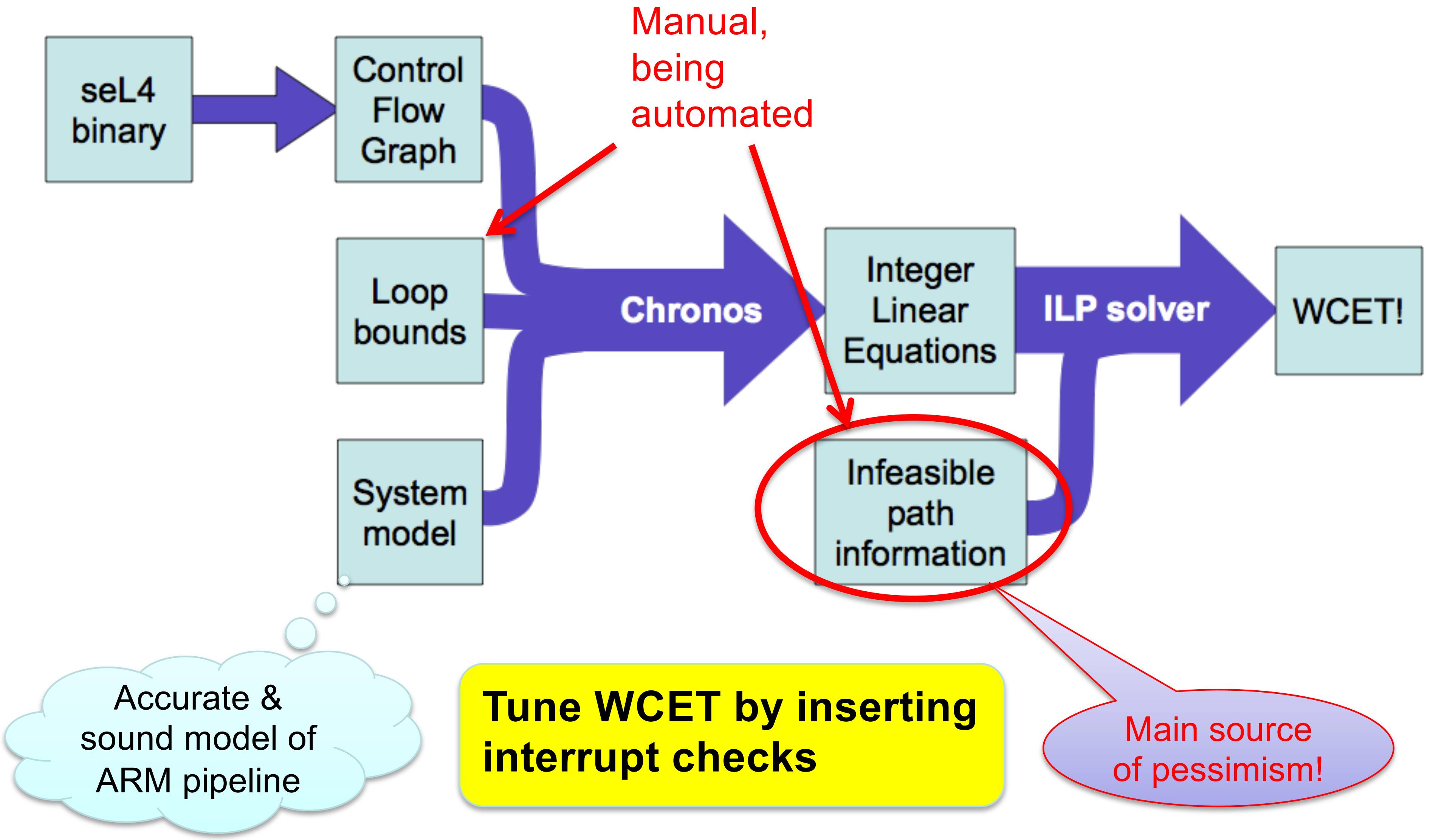


Timeliness



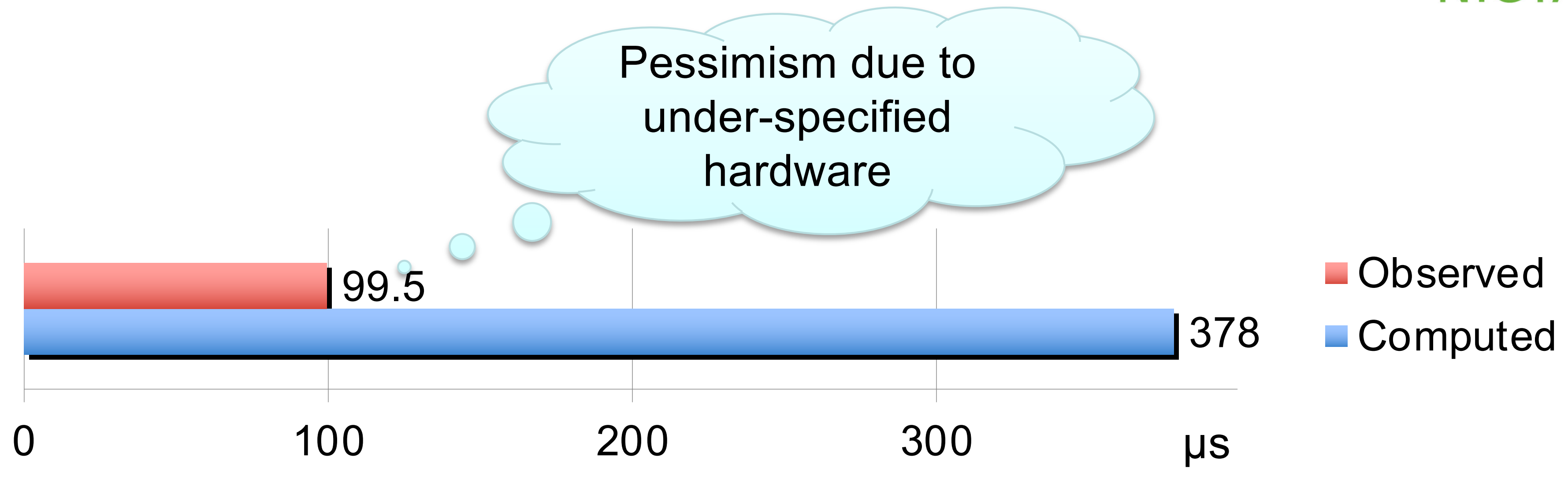
Need worst-case execution time (WCET) analysis of kernel

WCET Analysis Approach



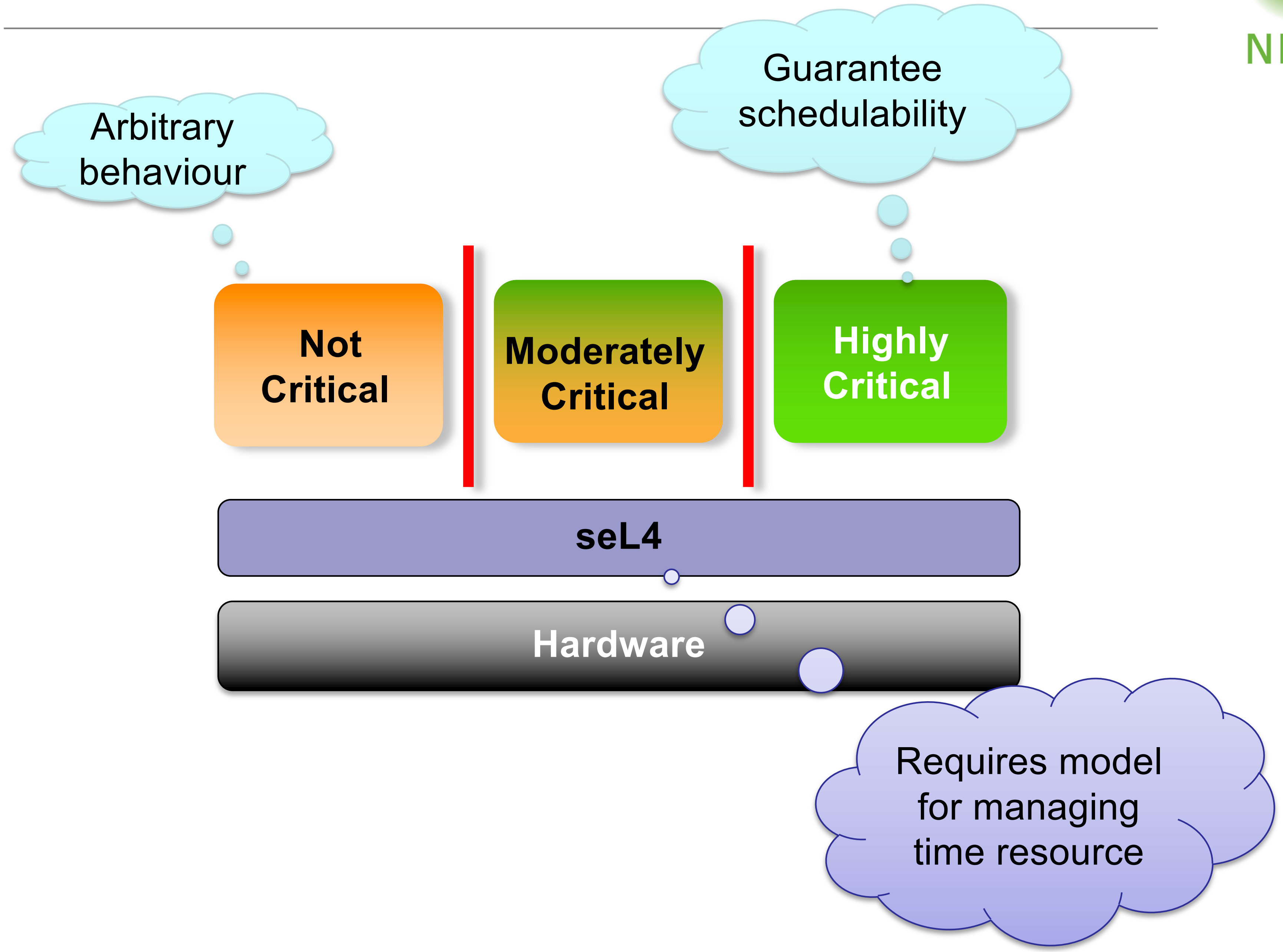
Tune WCET by inserting interrupt checks

Result

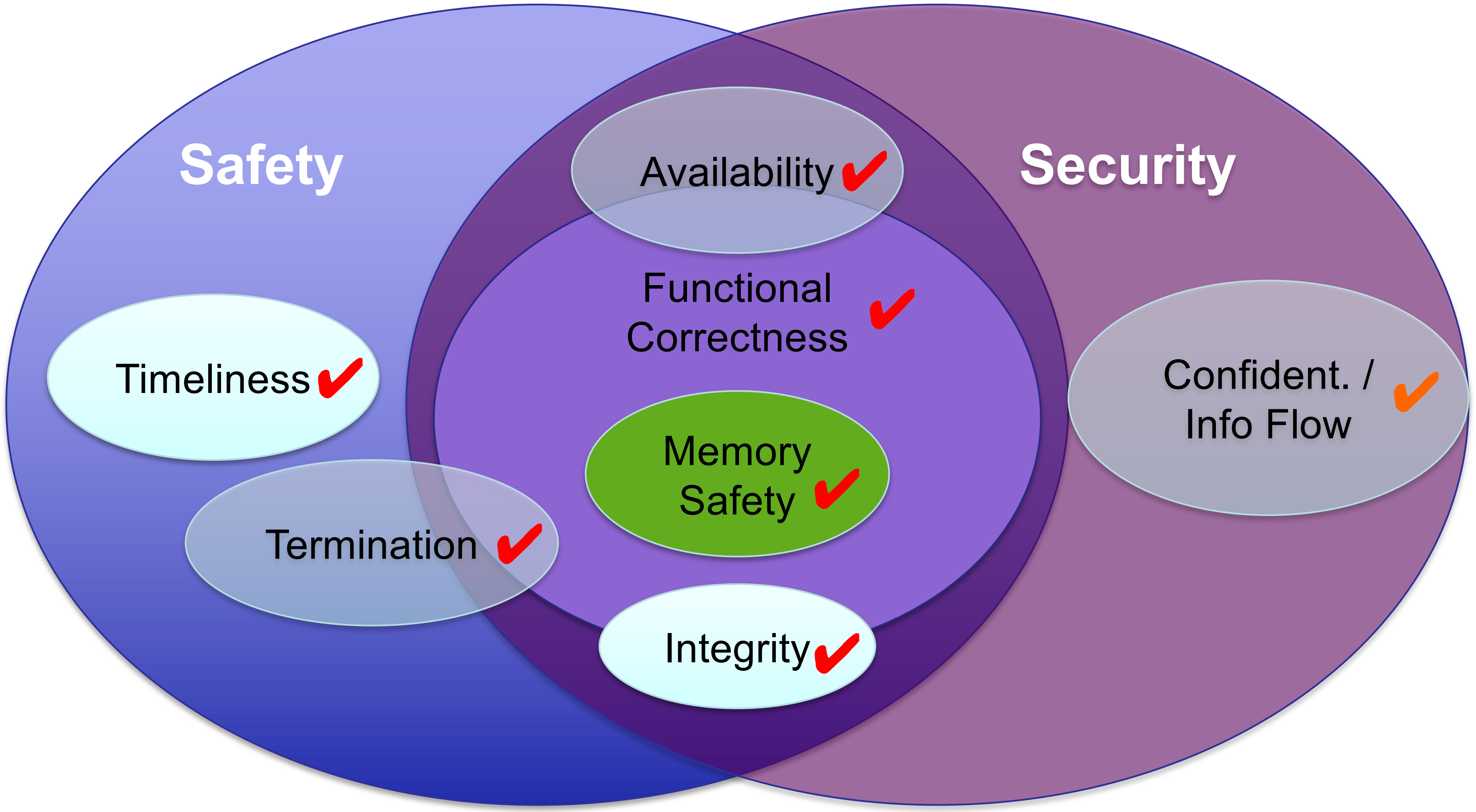


WCET presently limited by verification practicalities
- **10 µs seem achievable**

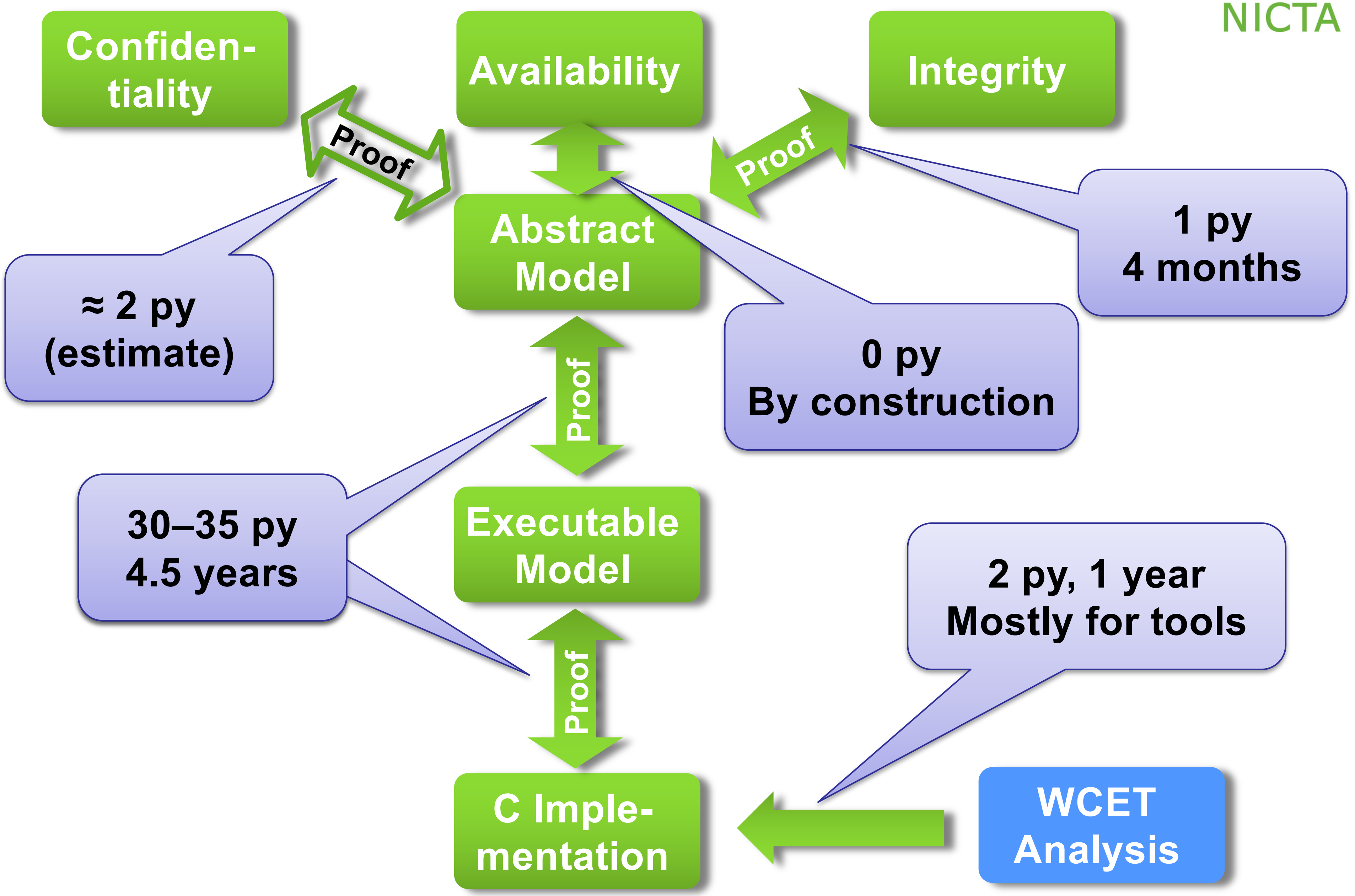
Future: Whole-System Schedulability



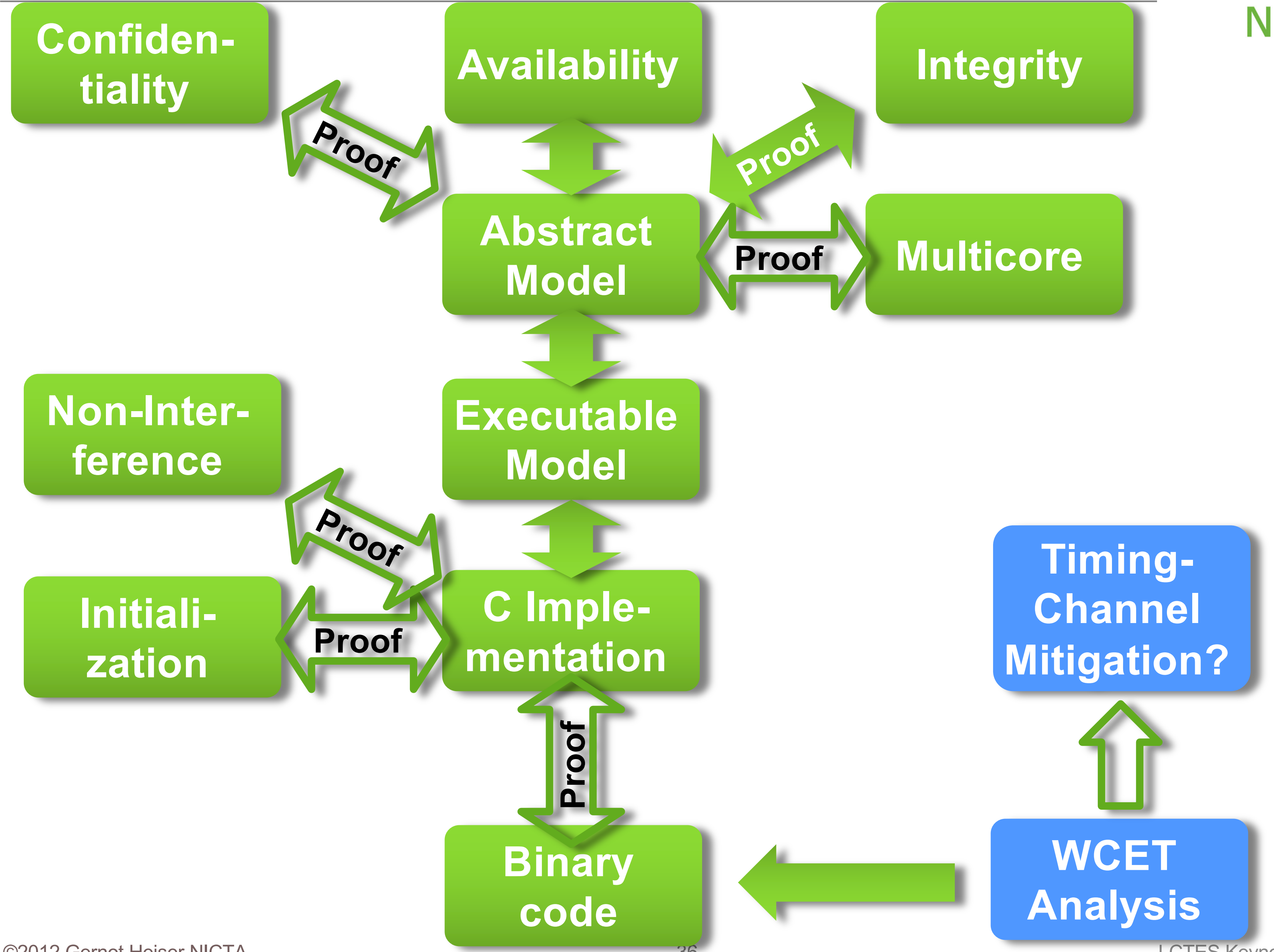
seL4 as Basis for Trustworthy Systems



Proving seL4 Trustworthiness



seL4 – the Next 24 Months

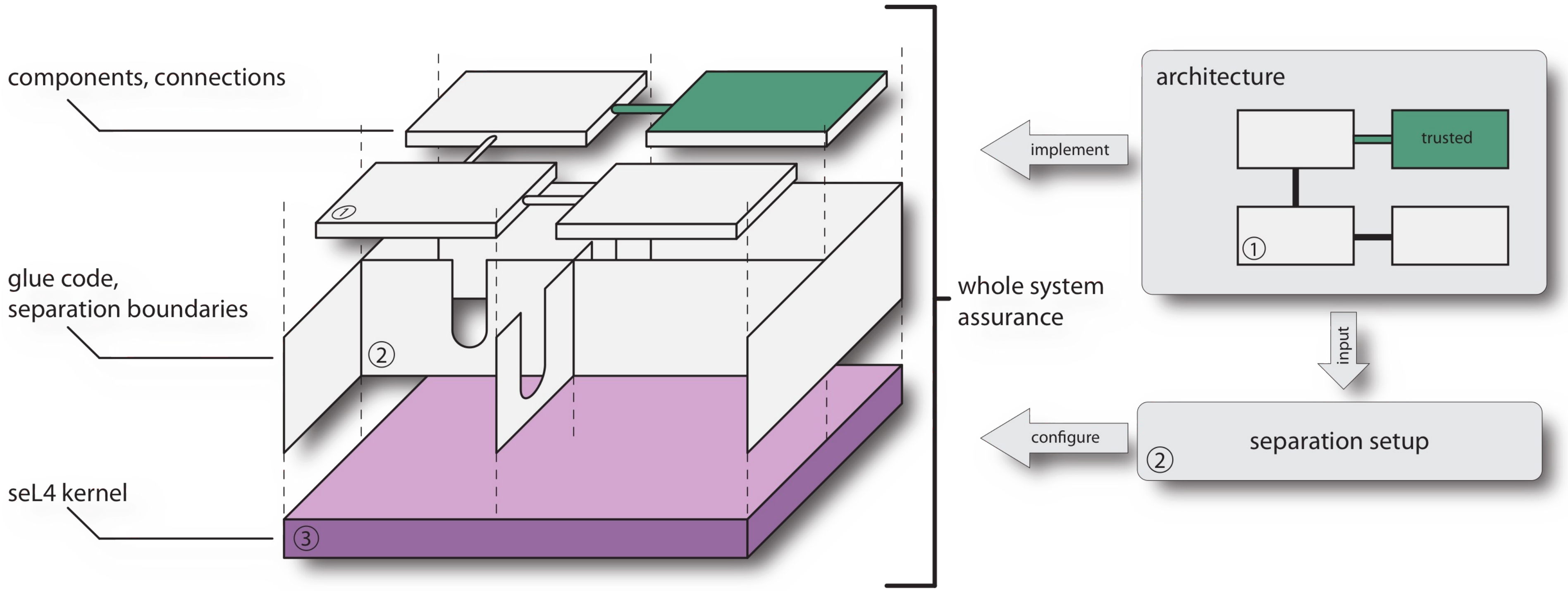


Phase Two: Full-System Guarantees

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

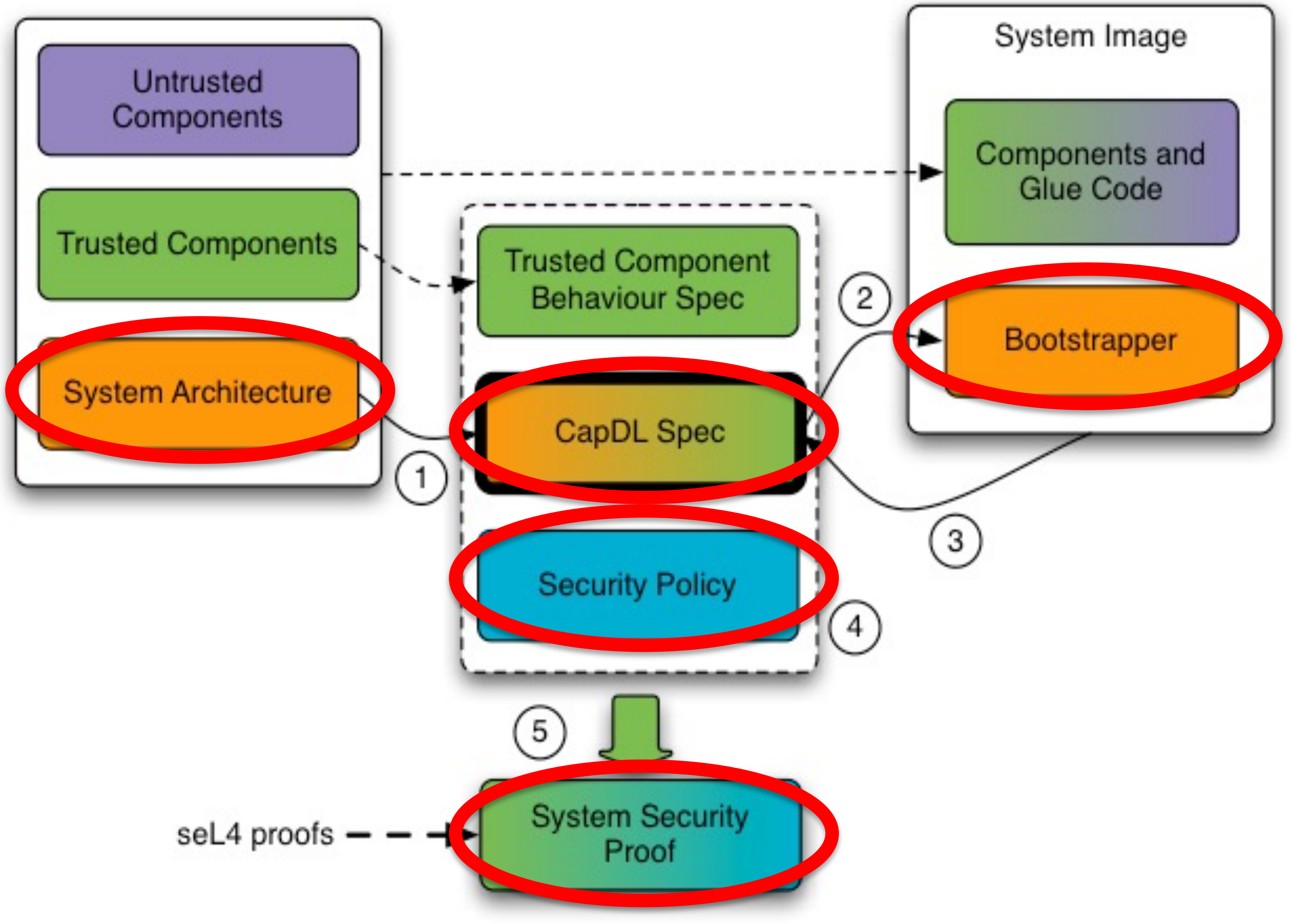
- Next step: Guarantees for real-world systems (1,000,000 LOC)

Overview of Approach

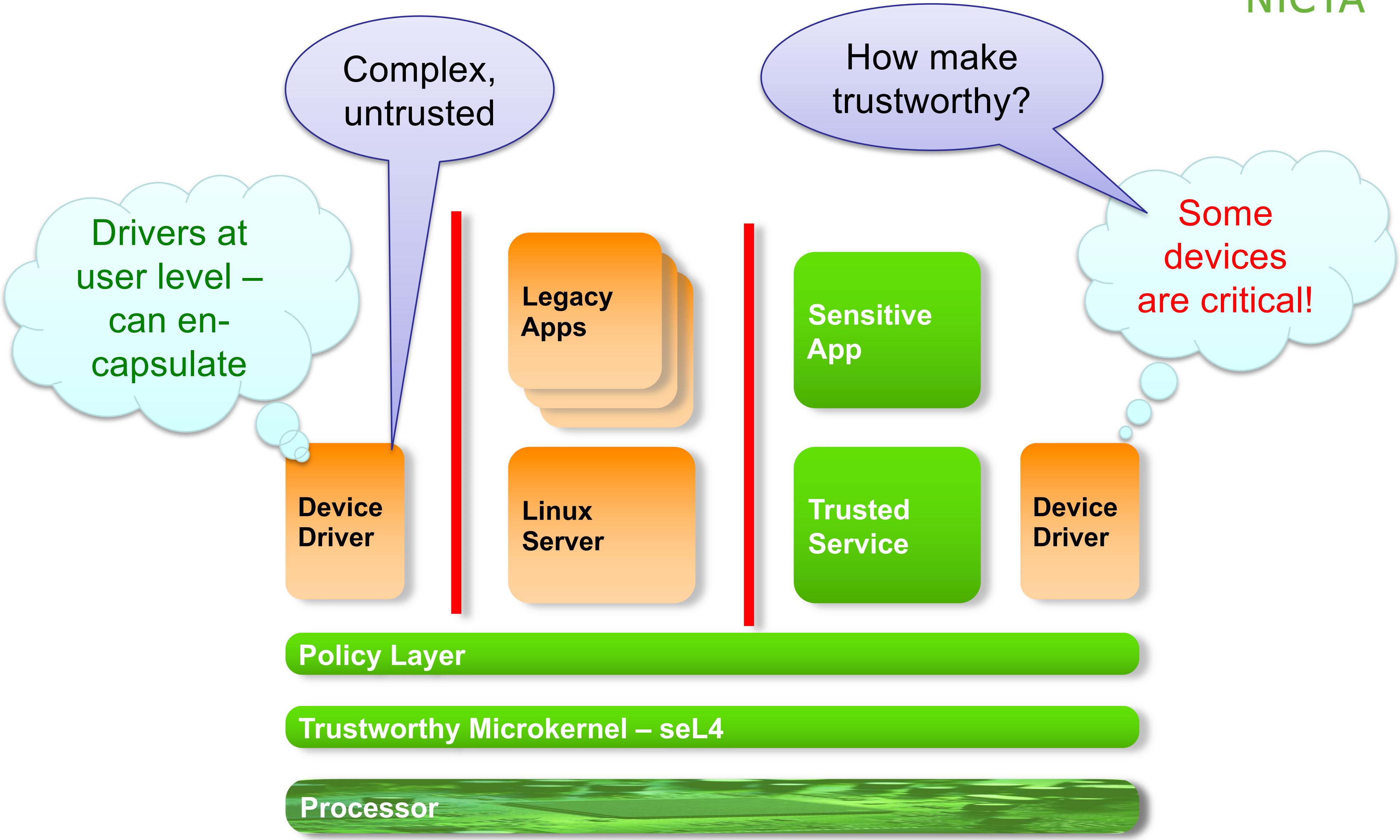


- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance

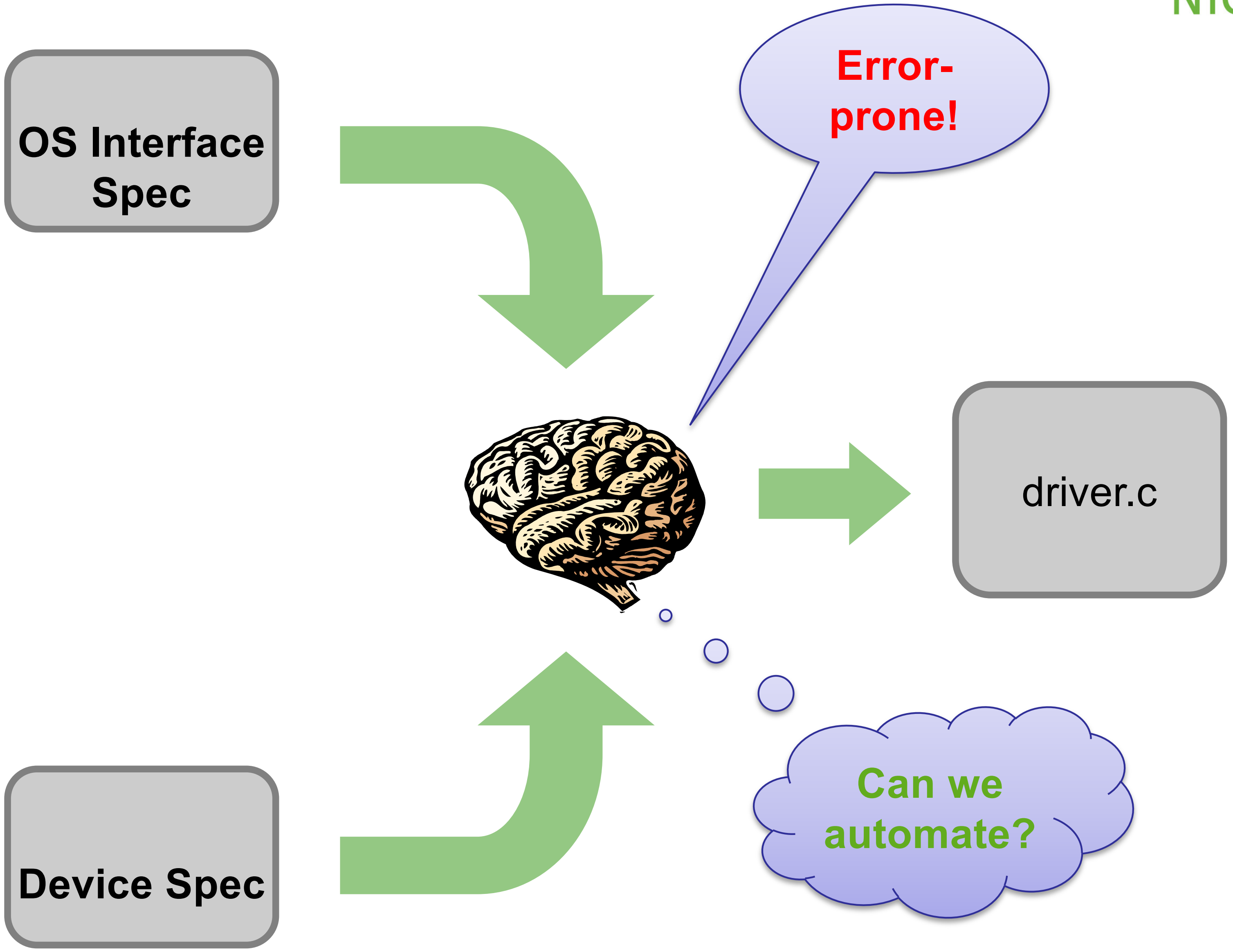
Specifying Security Architecture



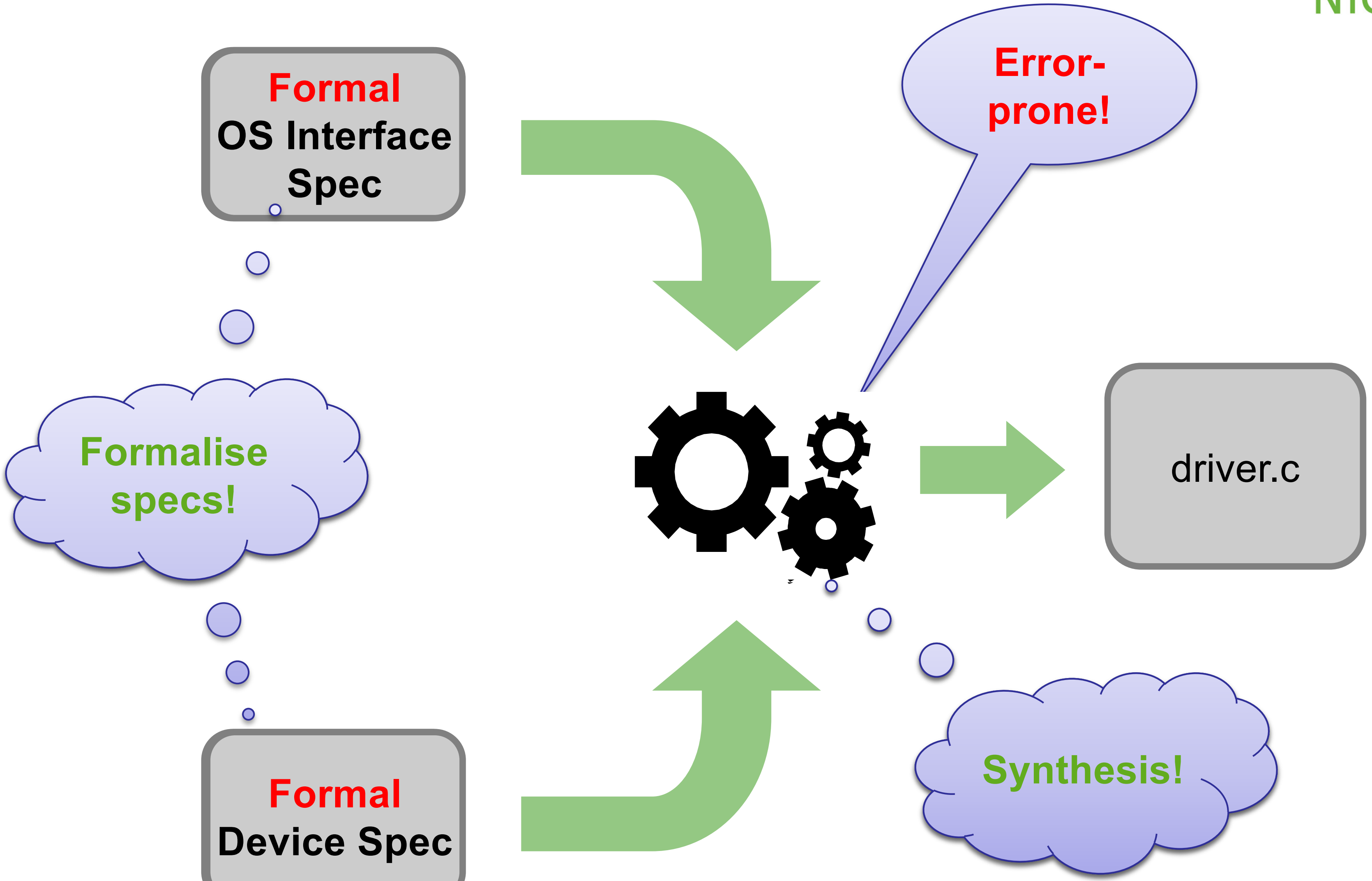
Device Drivers



Driver Development



Driver Development

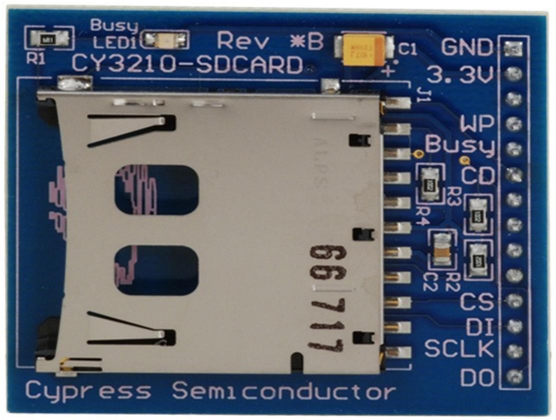


Drivers Synthesised (To Date)

IDE disk controller

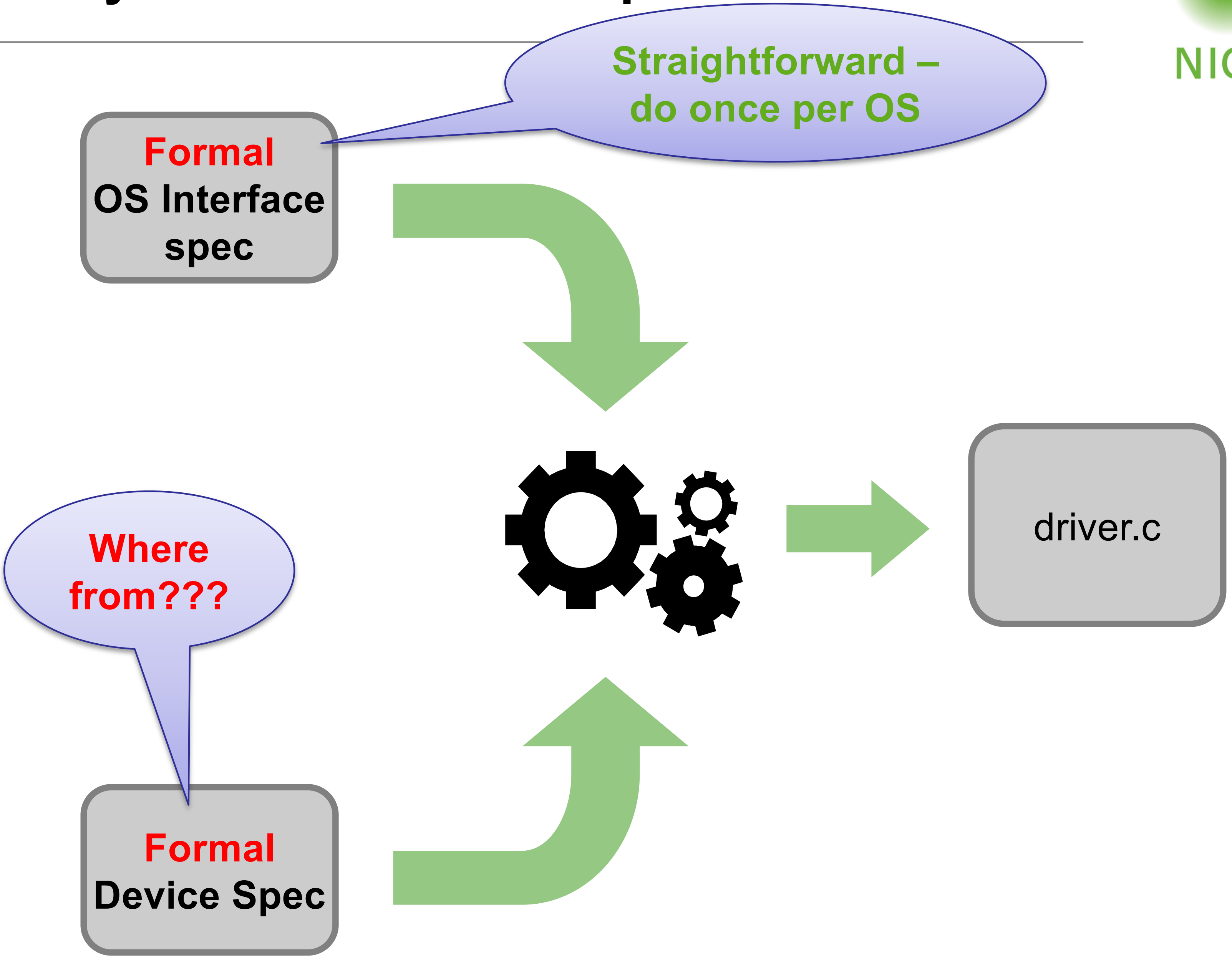
W5100 Eth shield

Asix AX88772
USB-to-Eth adapter

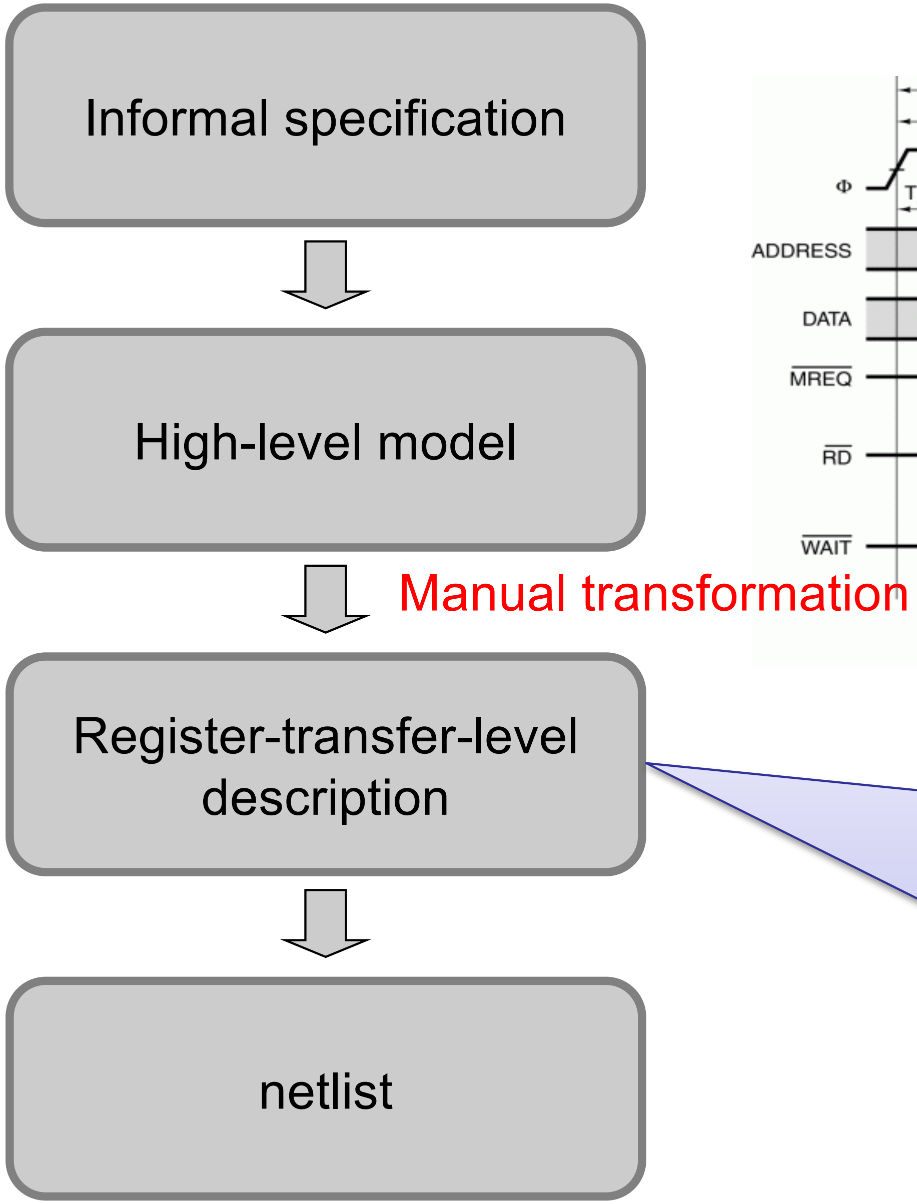


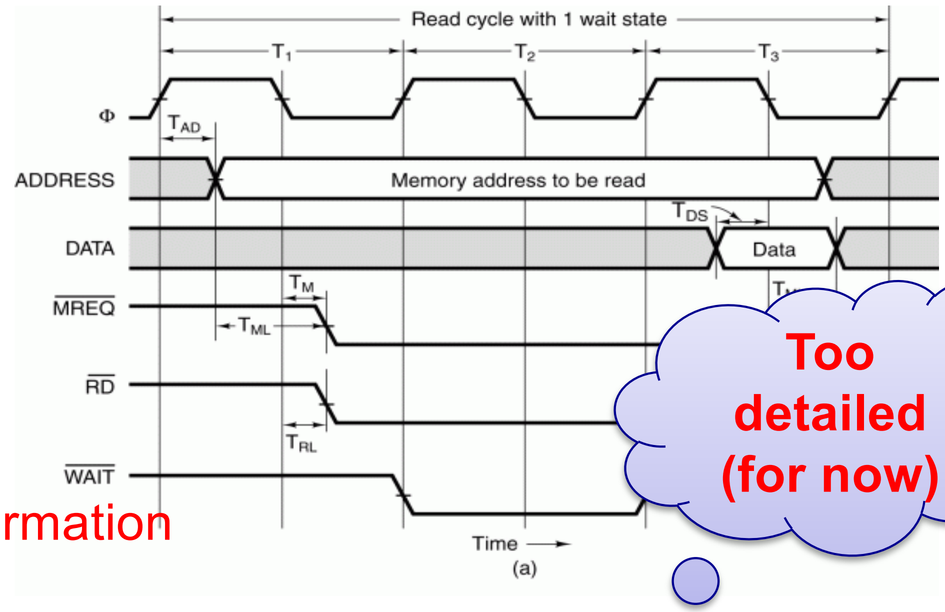
SD host controller

Driver Synthesis: Interface Specs



Hardware Design Workflow

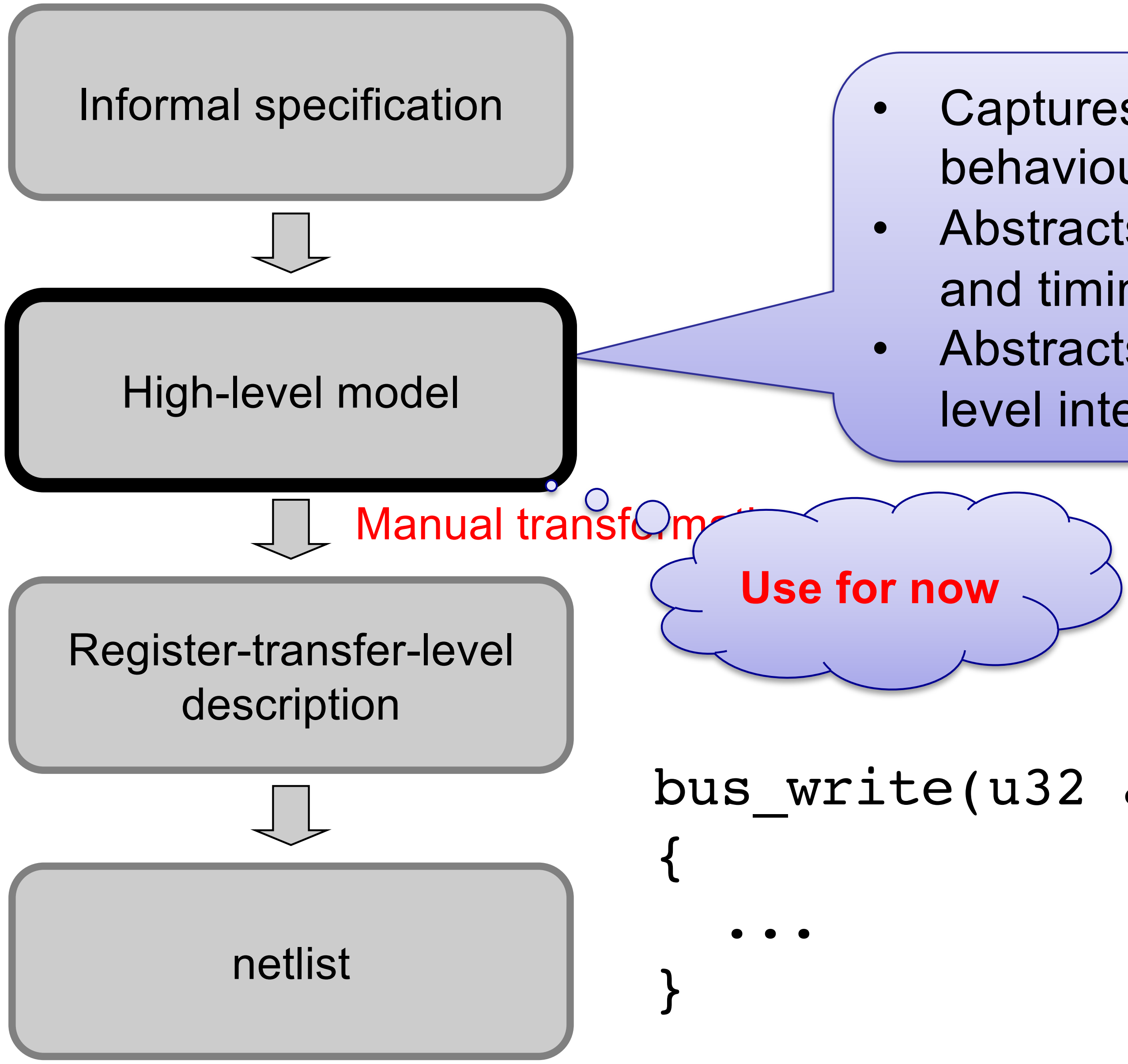




Too detailed (for now)

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

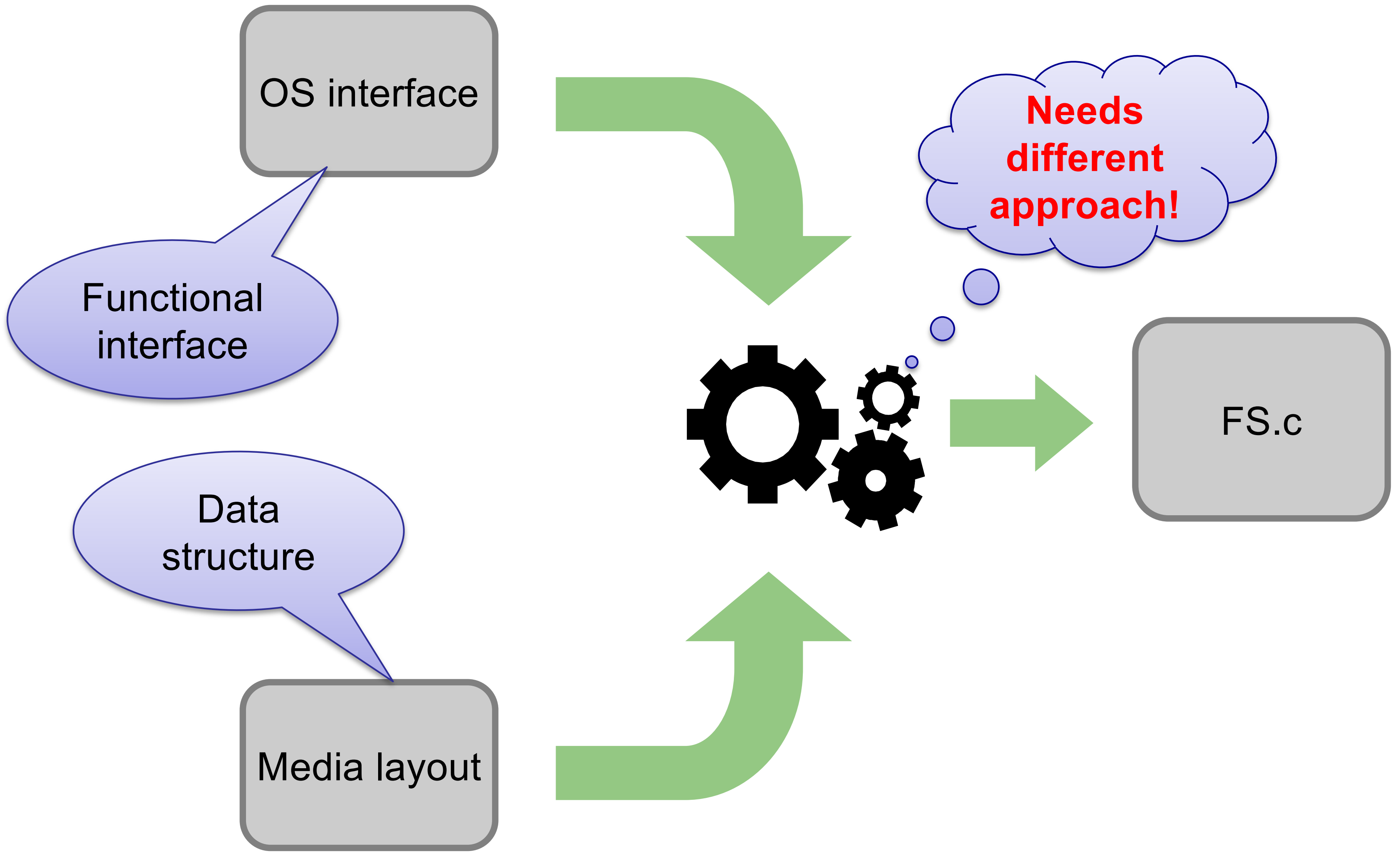
Hardware Design Workflow



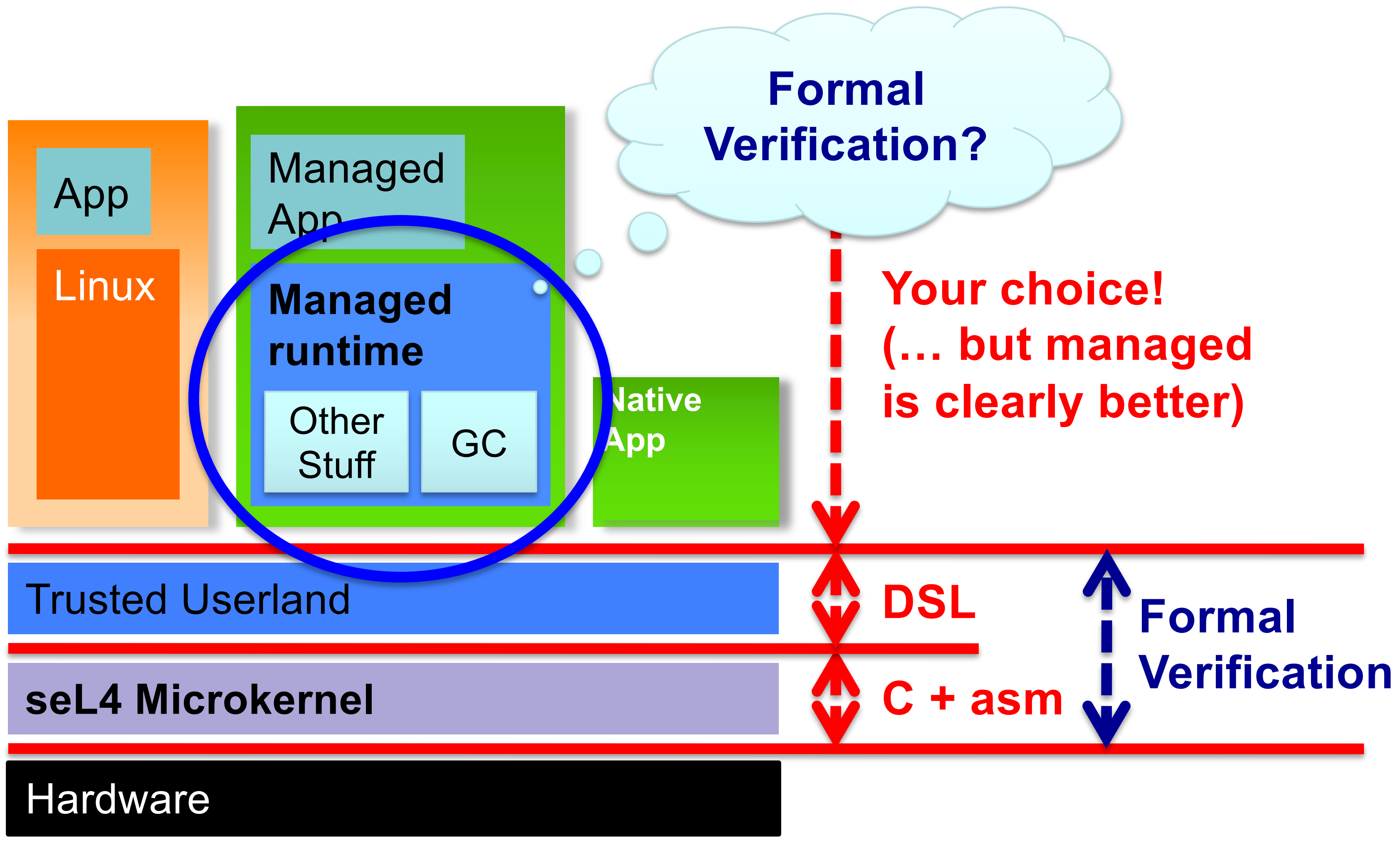
- Captures external behaviour
- Abstracts away structure and timing
- Abstracts away the low-level interface

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

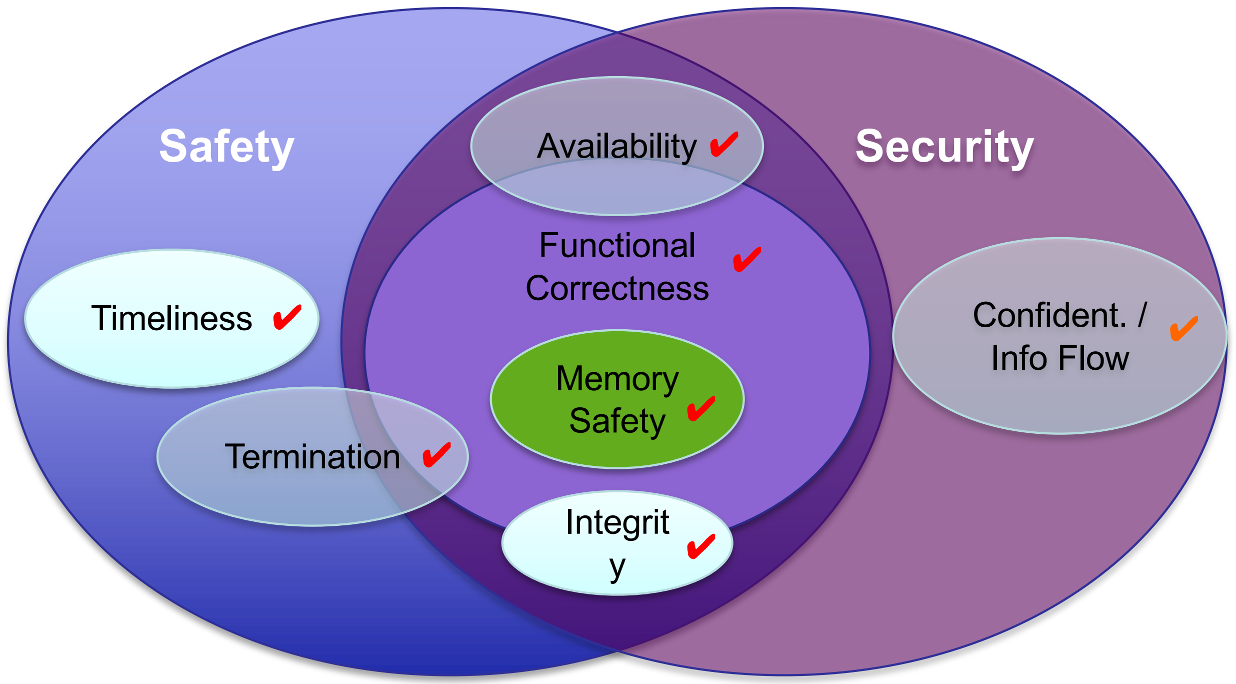
From Drivers to File Systems?



Building Secure Systems: Long-Term View



Trustworthy Systems – We've Made a Start!



Thank You!

mailto:gernot@nicta.com.au

@GernotHeiser

Google: "nicta trustworthy systems"