NICTA

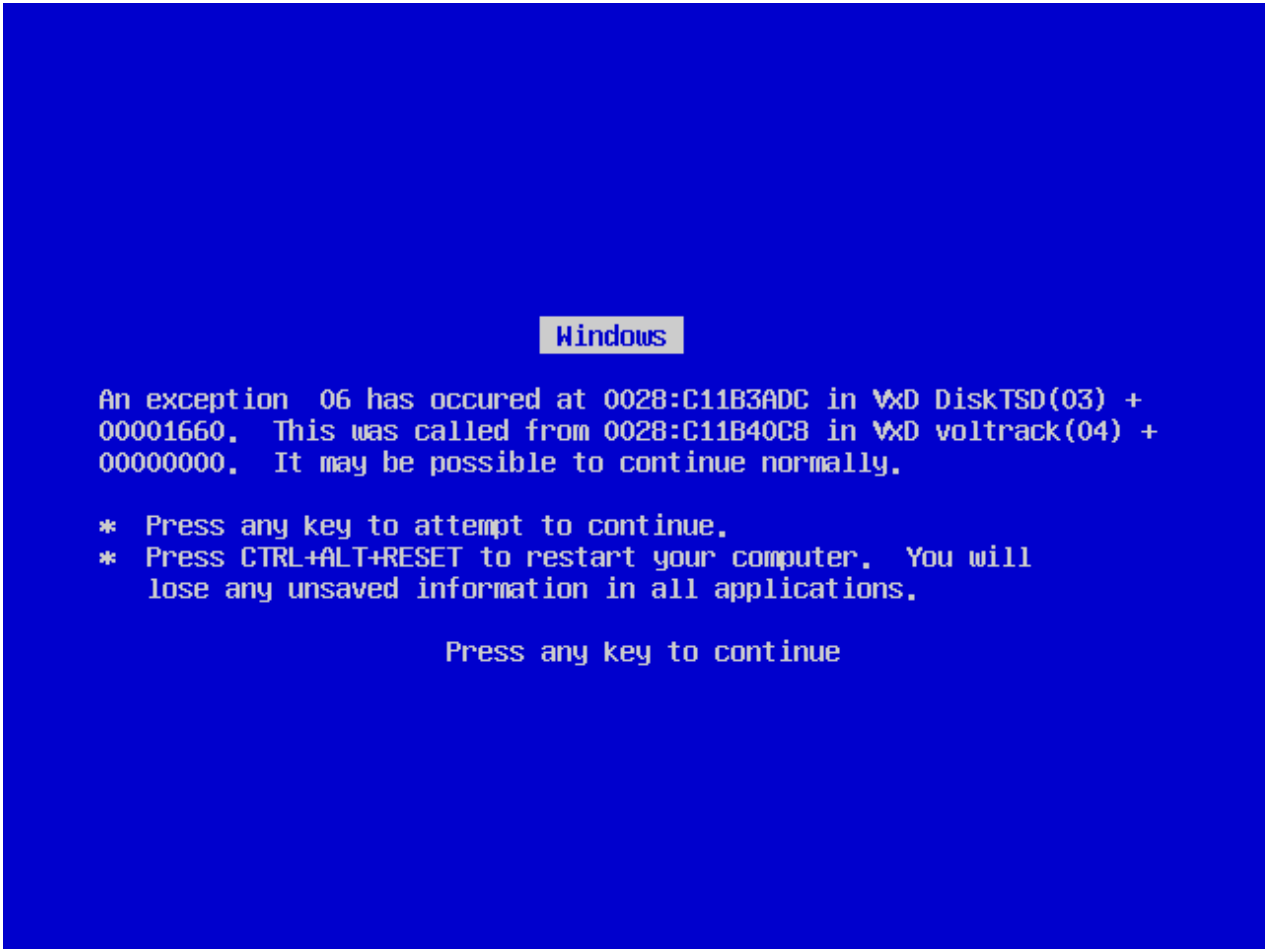
Towards a Platform for Trustworthy Systems

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
ACT 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
```

What's Next?

NICTA

Trust Without Trustworthiness

Core Issue: Complexity

- Massive functionality ⇒ huge software stacks
  - Expensive recalls of CE devices

- Increasing usability requirements
  - Wearable or implanted medical devices
  - Patient-operated
  - GUIs next to life-critical functionality

- On-going integration of critical and entertainment functions
  - Automotive infotainment and engine control

Our Vision: Trustworthy Systems

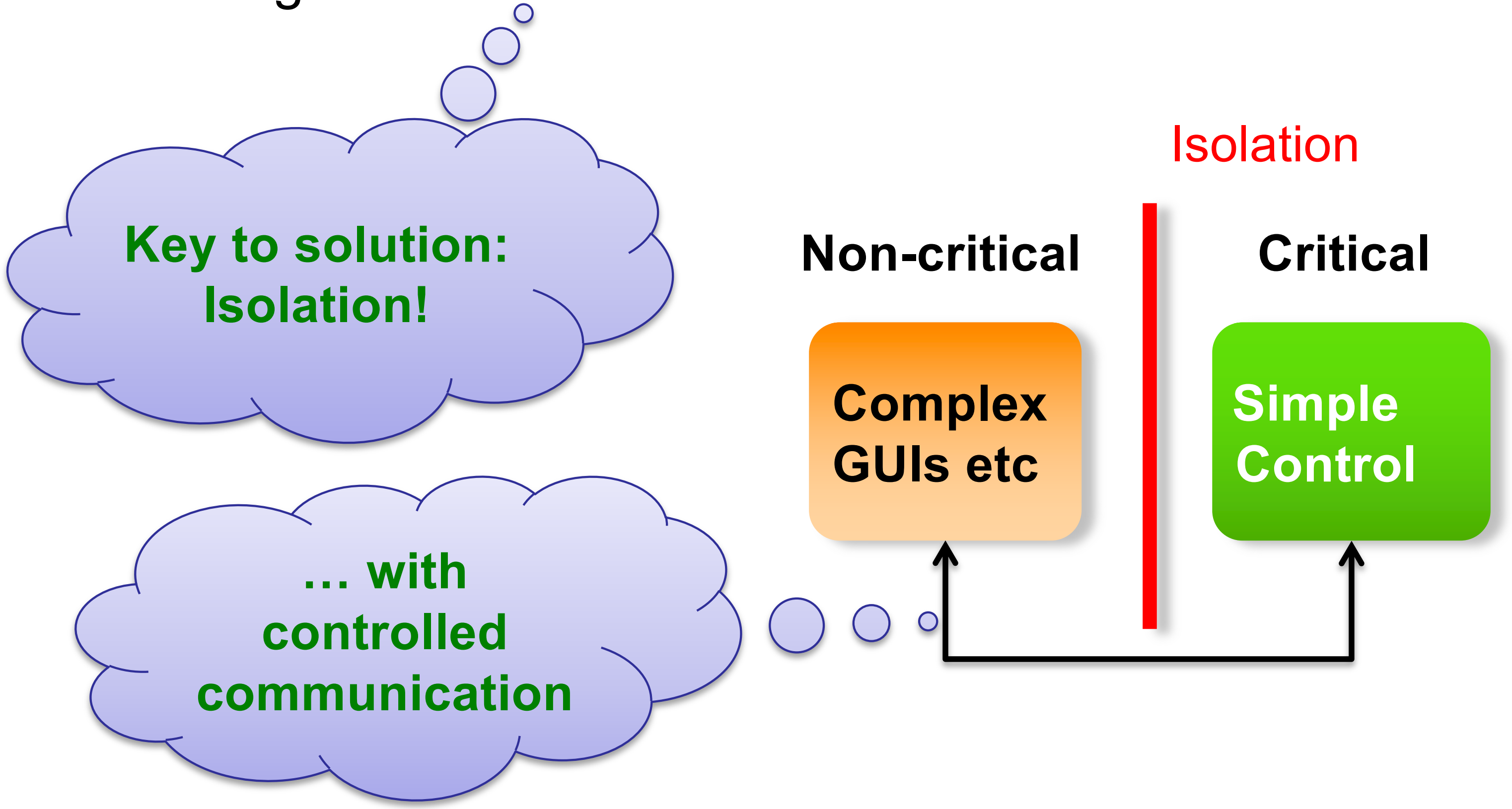
Suitable for real-world systems

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

Hard *guarantees* on safety/security/ reliability

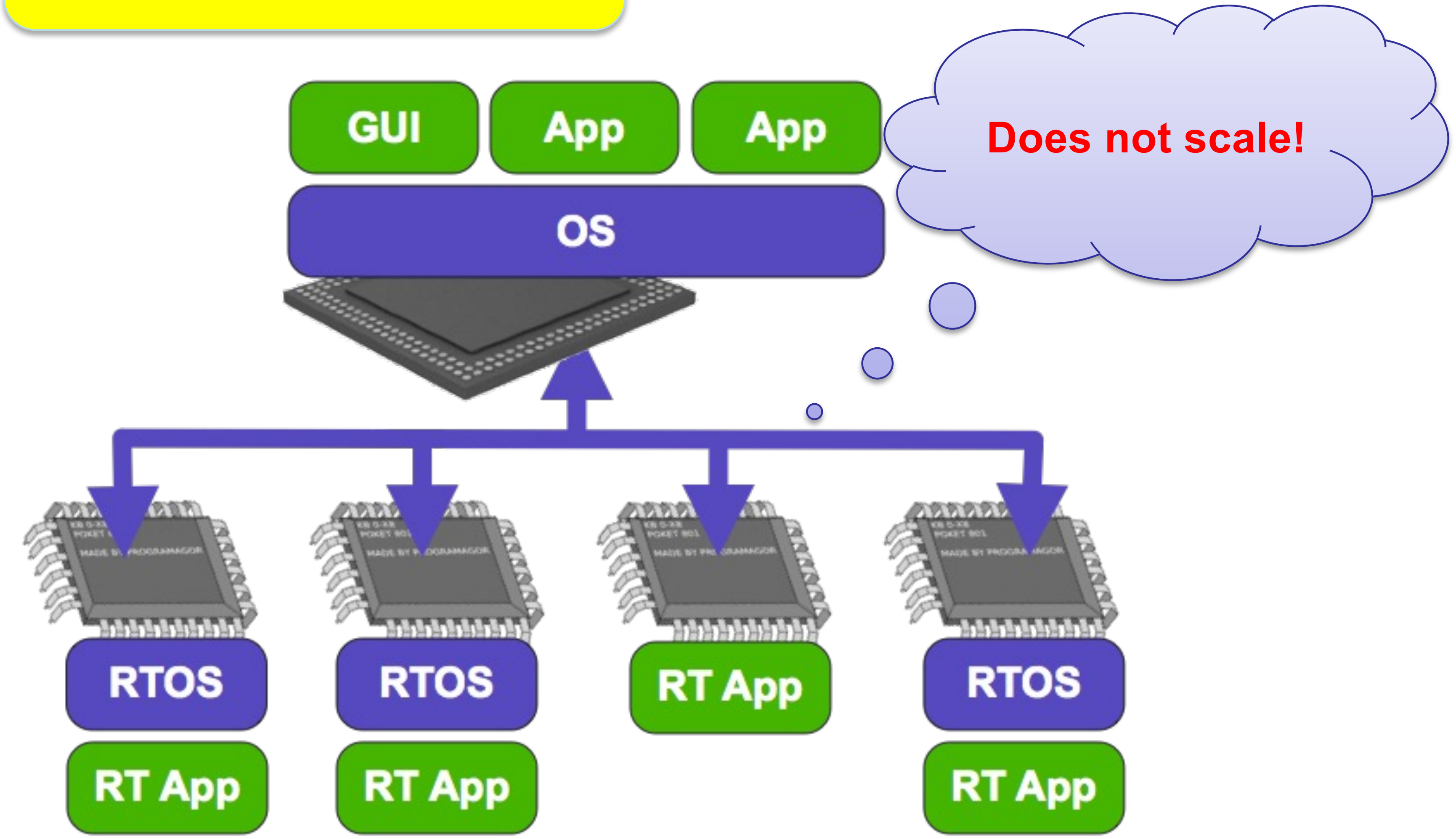
Dealing With Complexity

- Complexity of critical devices will continue to grow
  - Critical systems with millions of lines of code (LOC)
- We need to learn to ensure *dependability* despite complexity
  - Need to *guarantee* dependability
- Correctness guarantees for MLOCs unfeasible

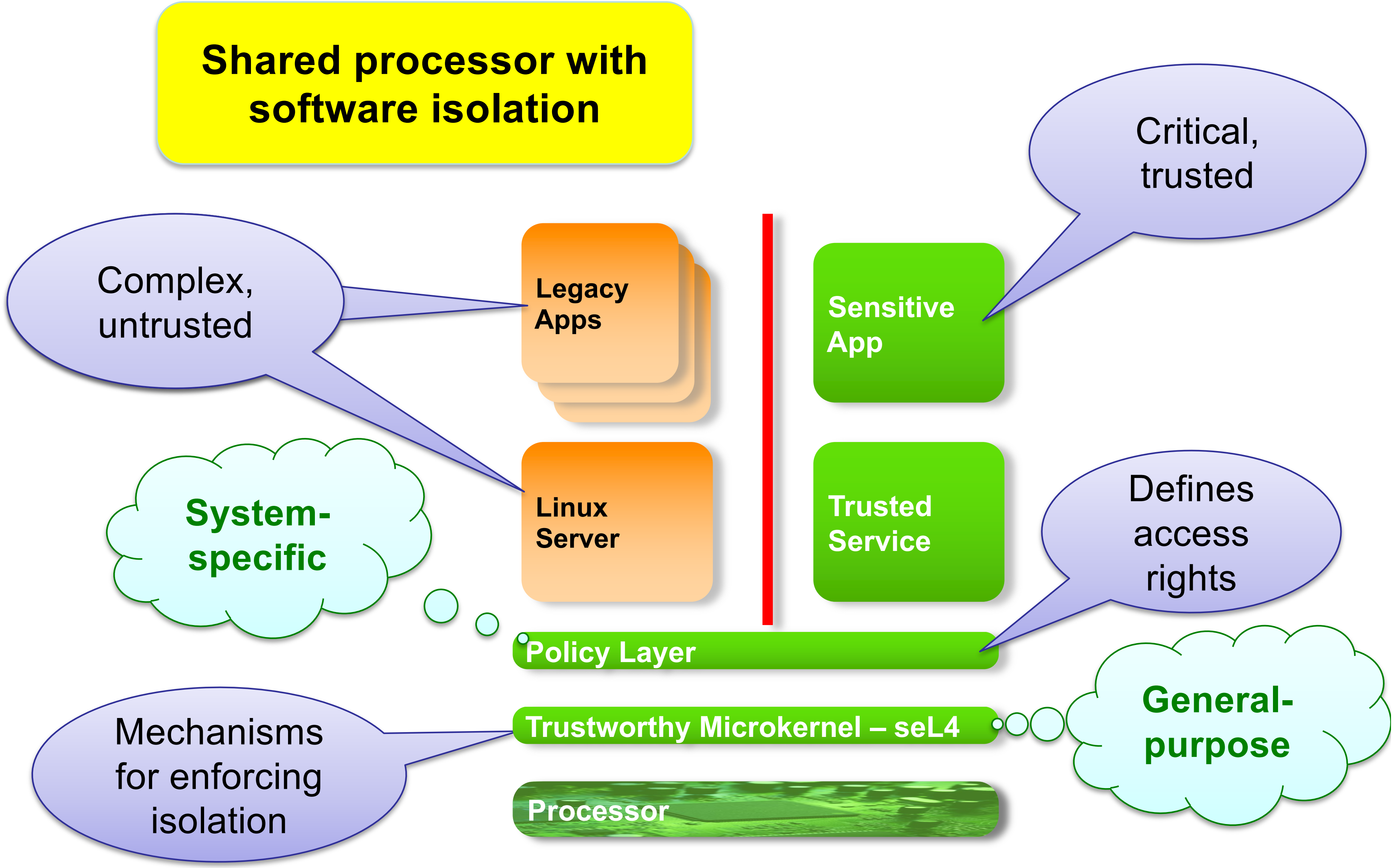


Isolation: Physical

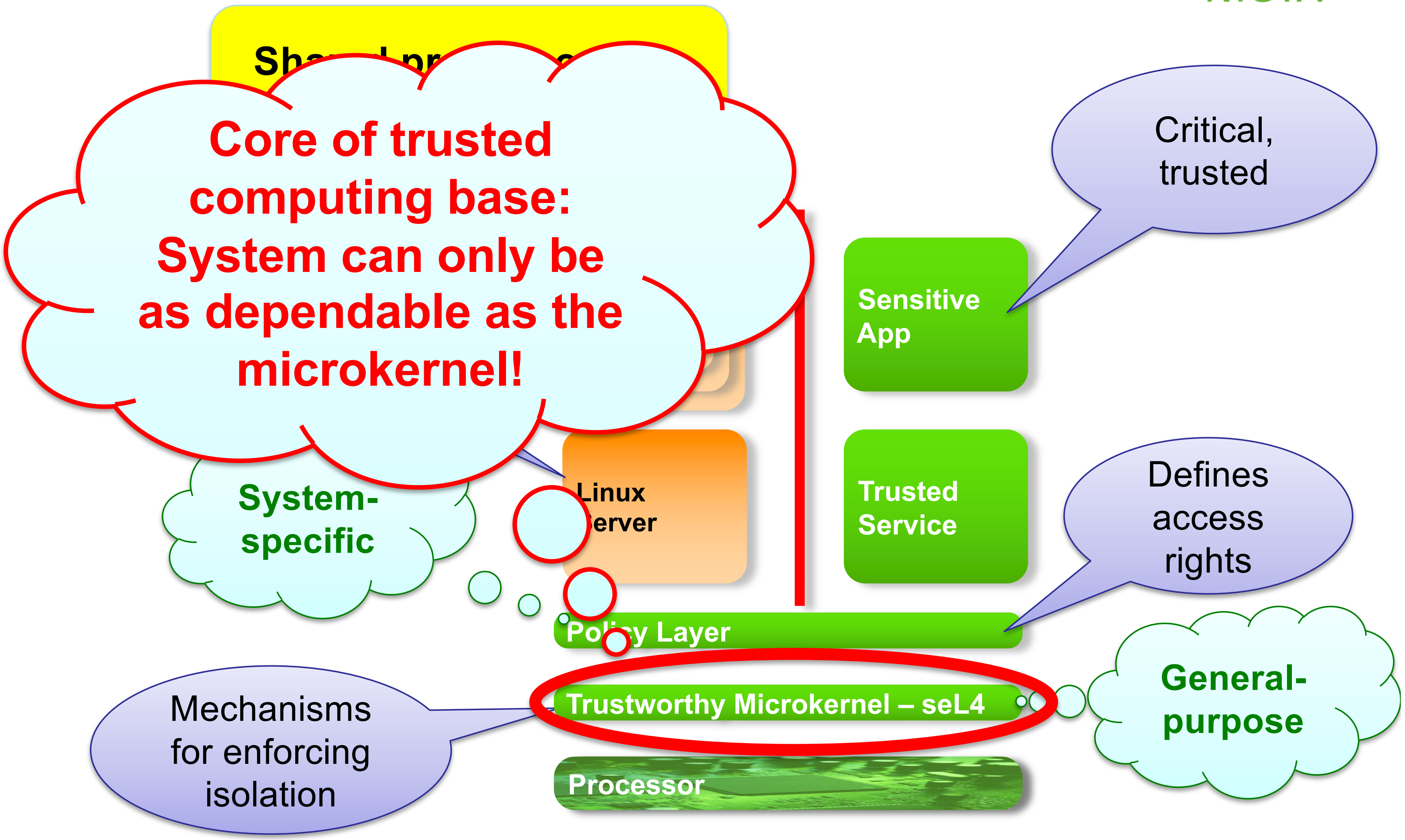
Separate processors for critical functionality



Isolation: Logical



Isolation: Logical


Shared pr
Core of trusted computing base: System can only be as dependable as the microkernel!
Critical, trusted
Sensitive App
System-specific
Linux Server
Trusted Service
Defines access rights
Policy Layer
Trustworthy Microkernel – seL4
General-purpose
Mechanisms for enforcing isolation
Processor

NICTA Trustworthy Systems Agenda

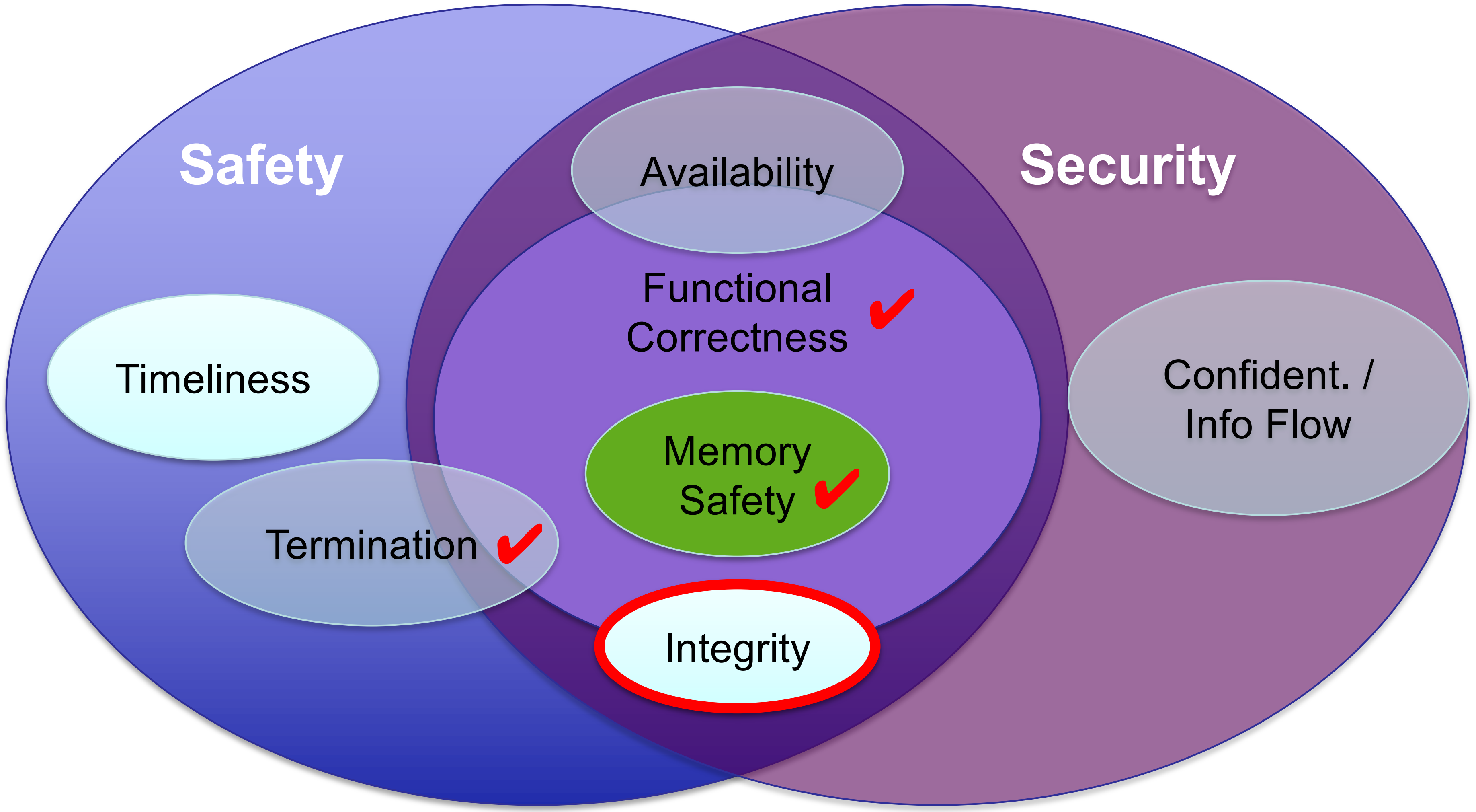
1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

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

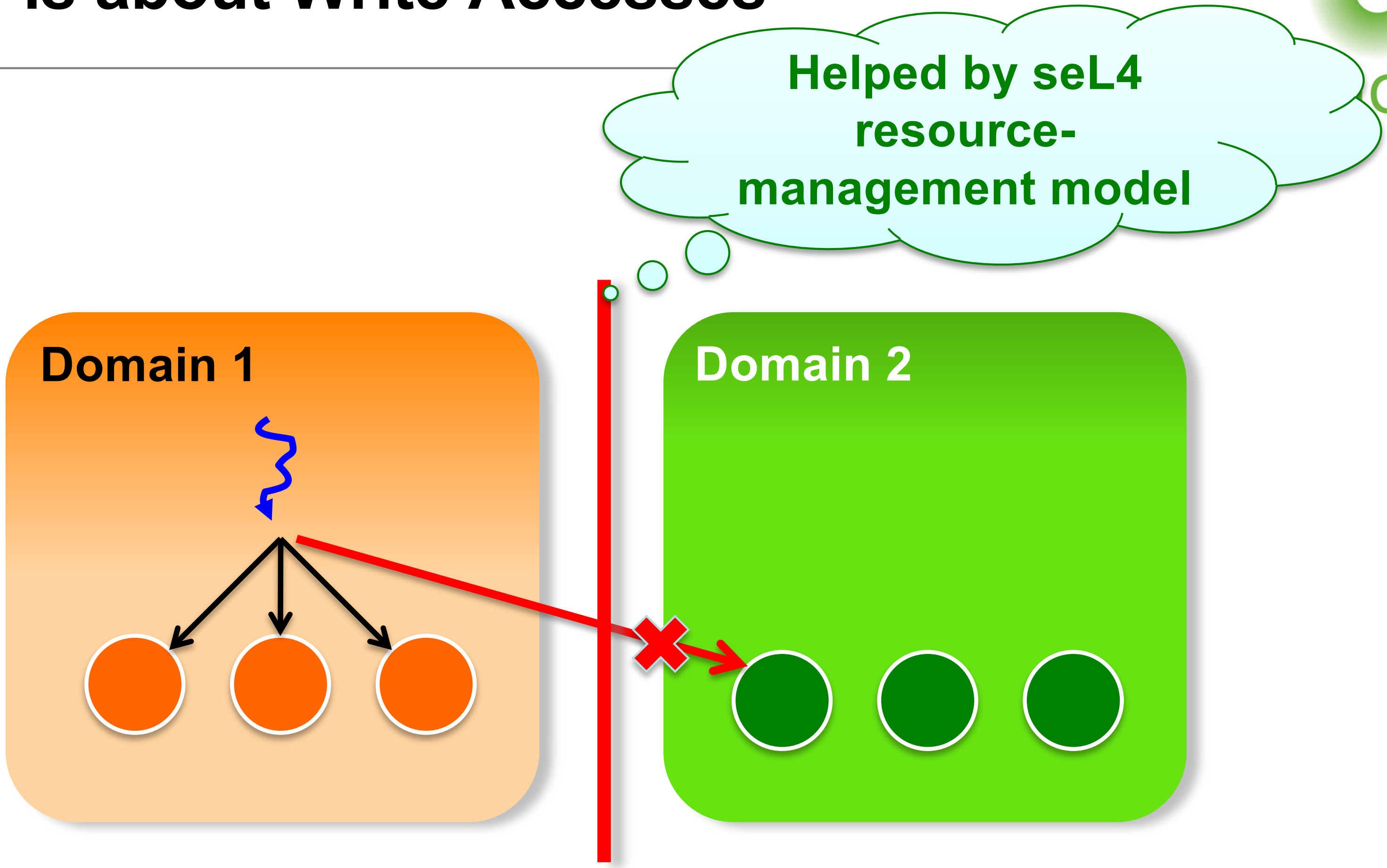
Core Ingredients: People

Formal Methods Practitioners **Systems Researchers**

seL4 as Basis for Trustworthy Systems



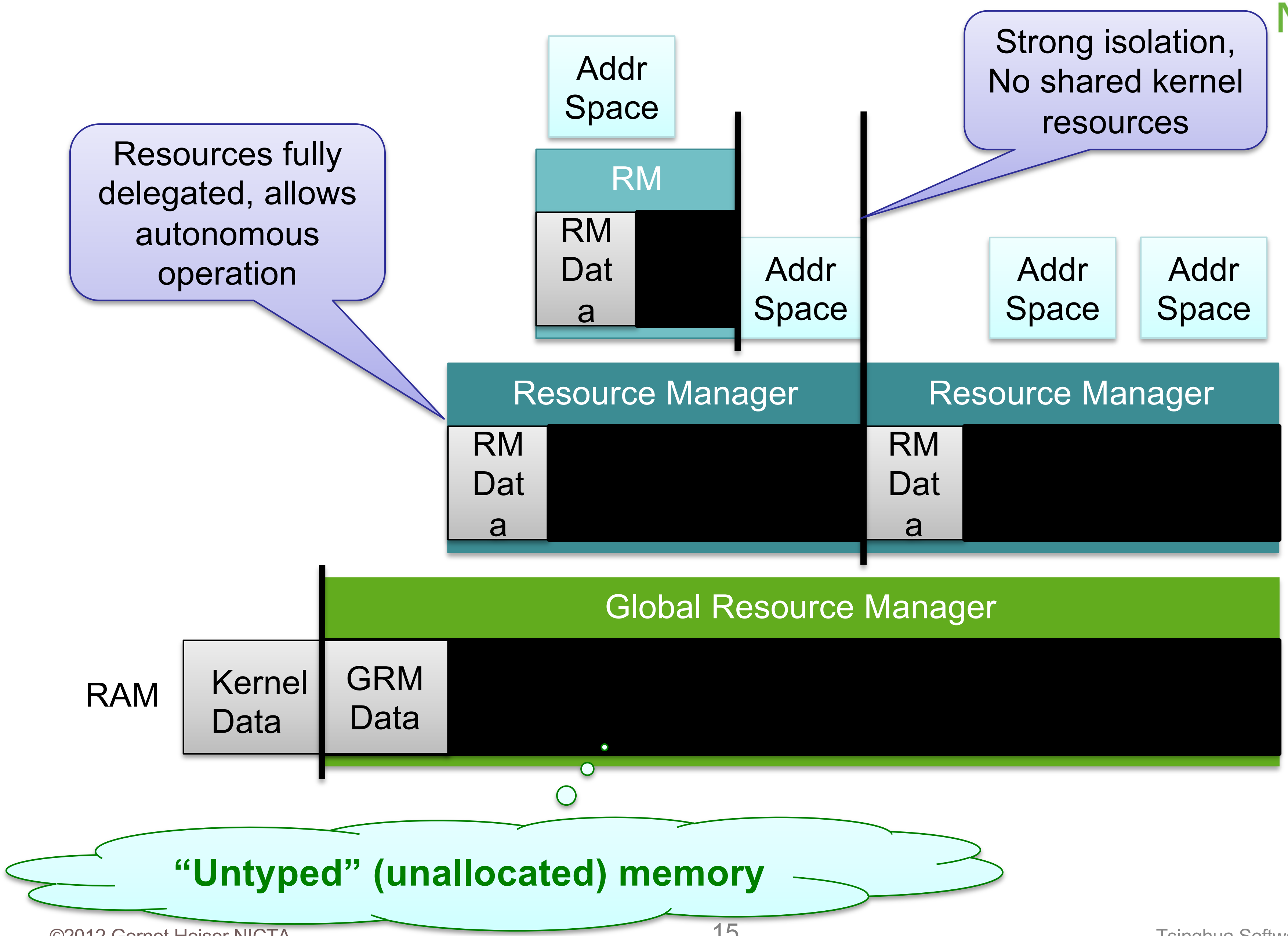
Integrity is about Write Accesses



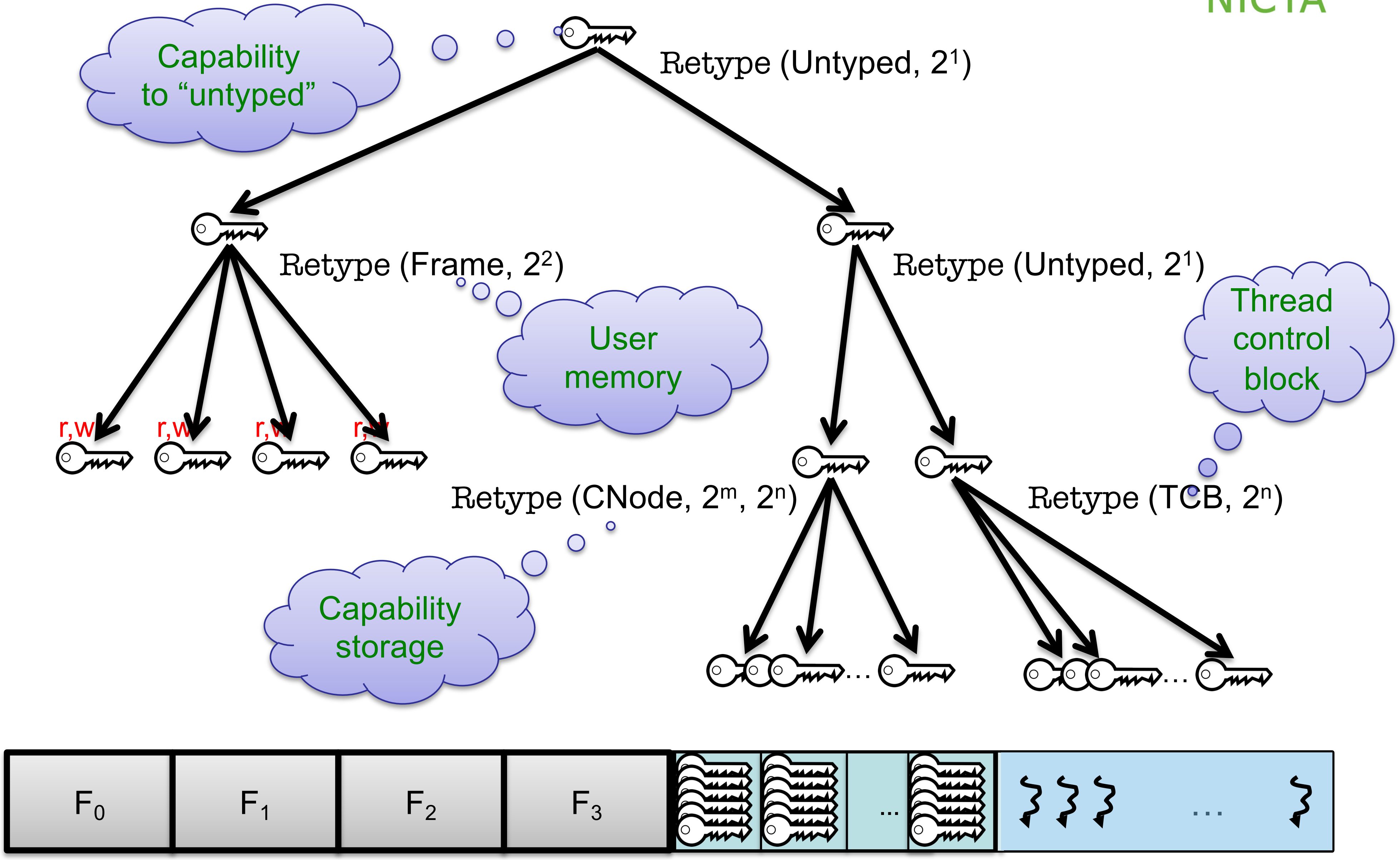
To prove:

- 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!*

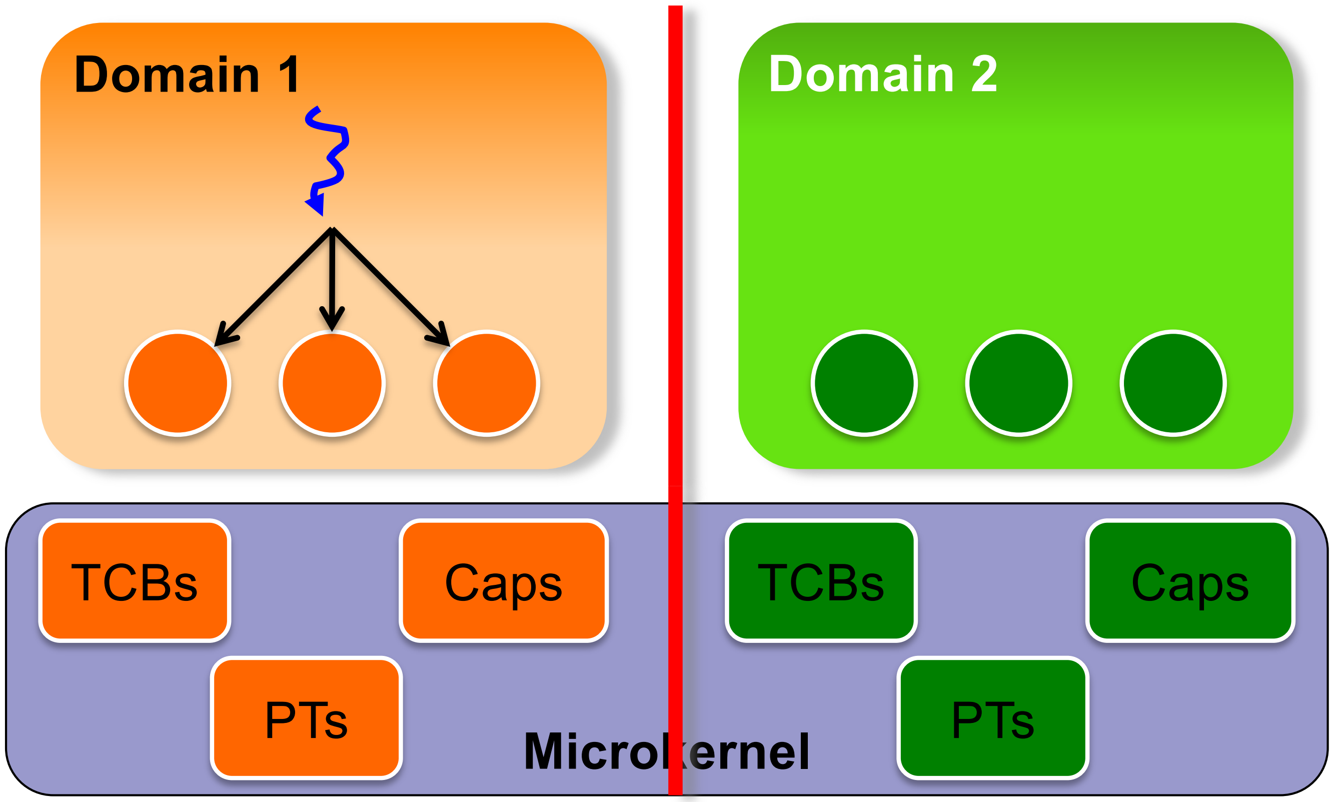
seL4 Memory Management Approach



seL4 Memory Management Mechanics: **Retype**

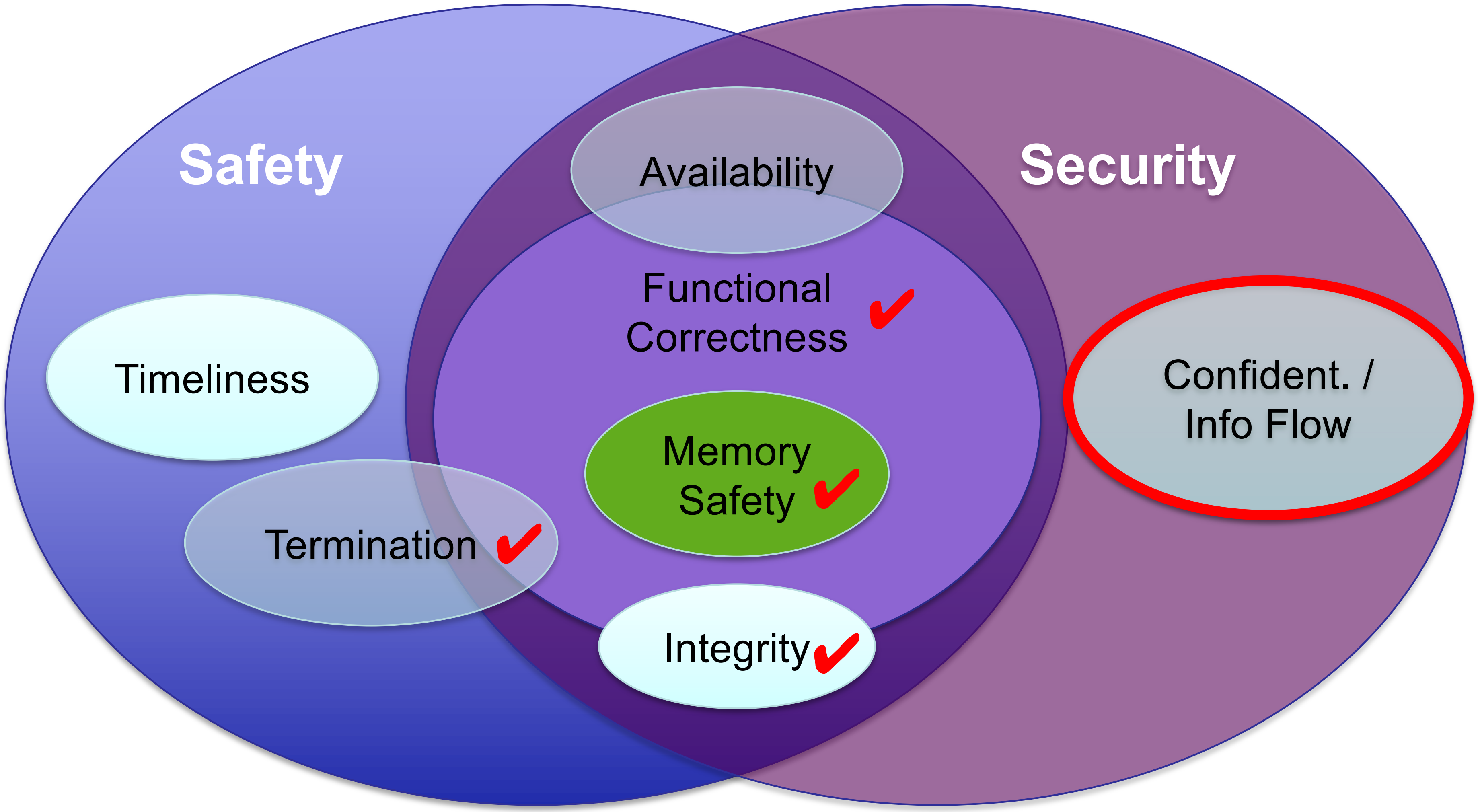


Separation of Kernel Data

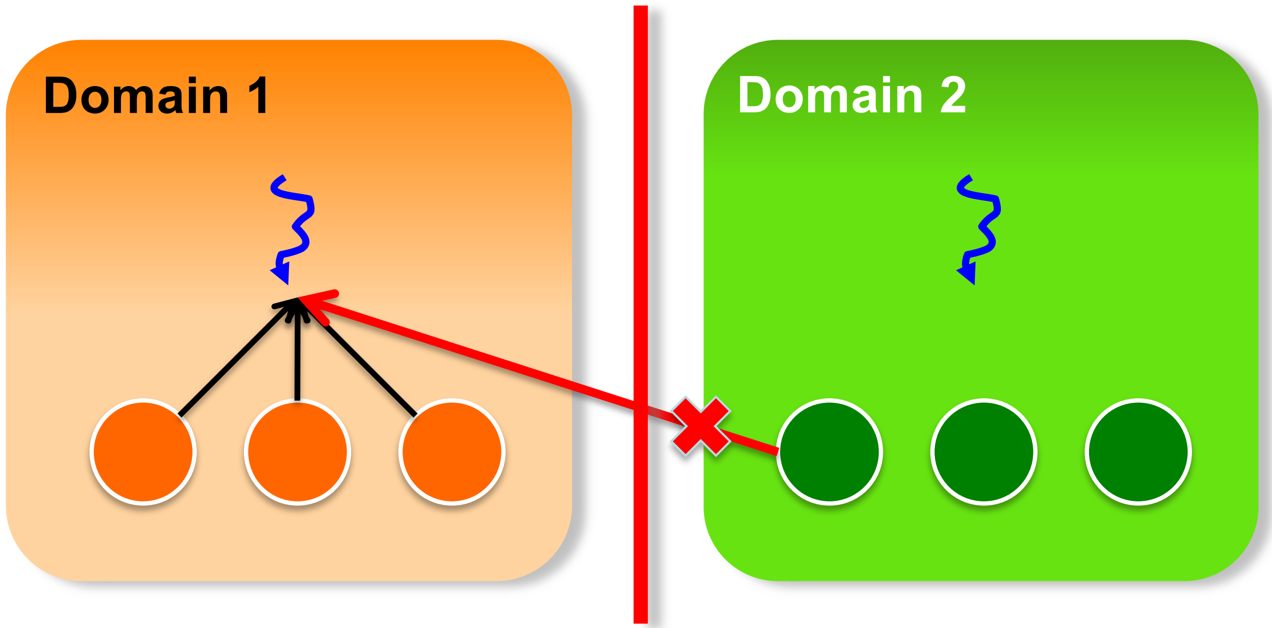


- Kernel data structures allocated/managed by user
  - Protected by capabilities just as user data!
- For integrity show that no object can be *modified* without a *write cap*

seL4 for Safety and Security



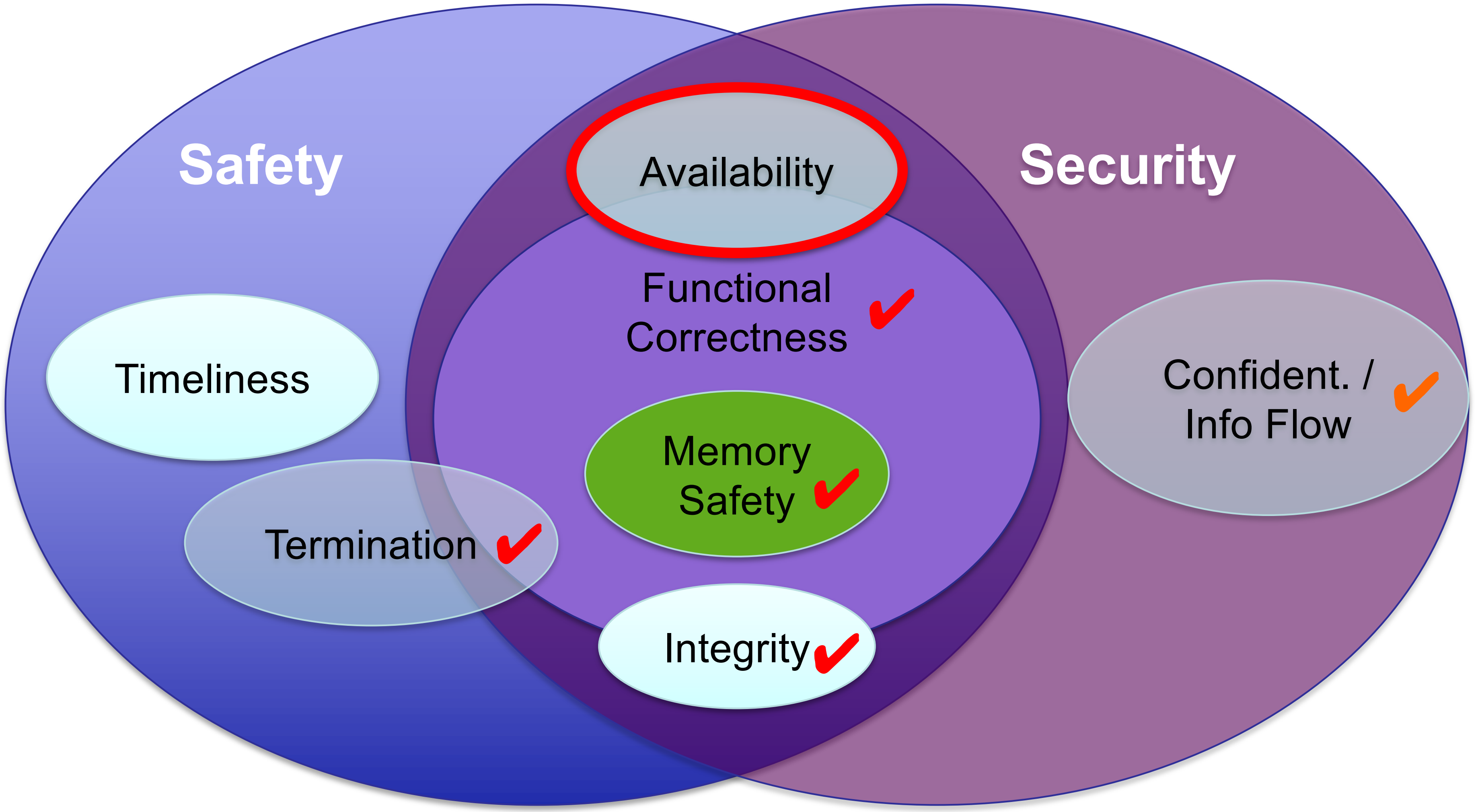
Confidentiality is about Read Accesses



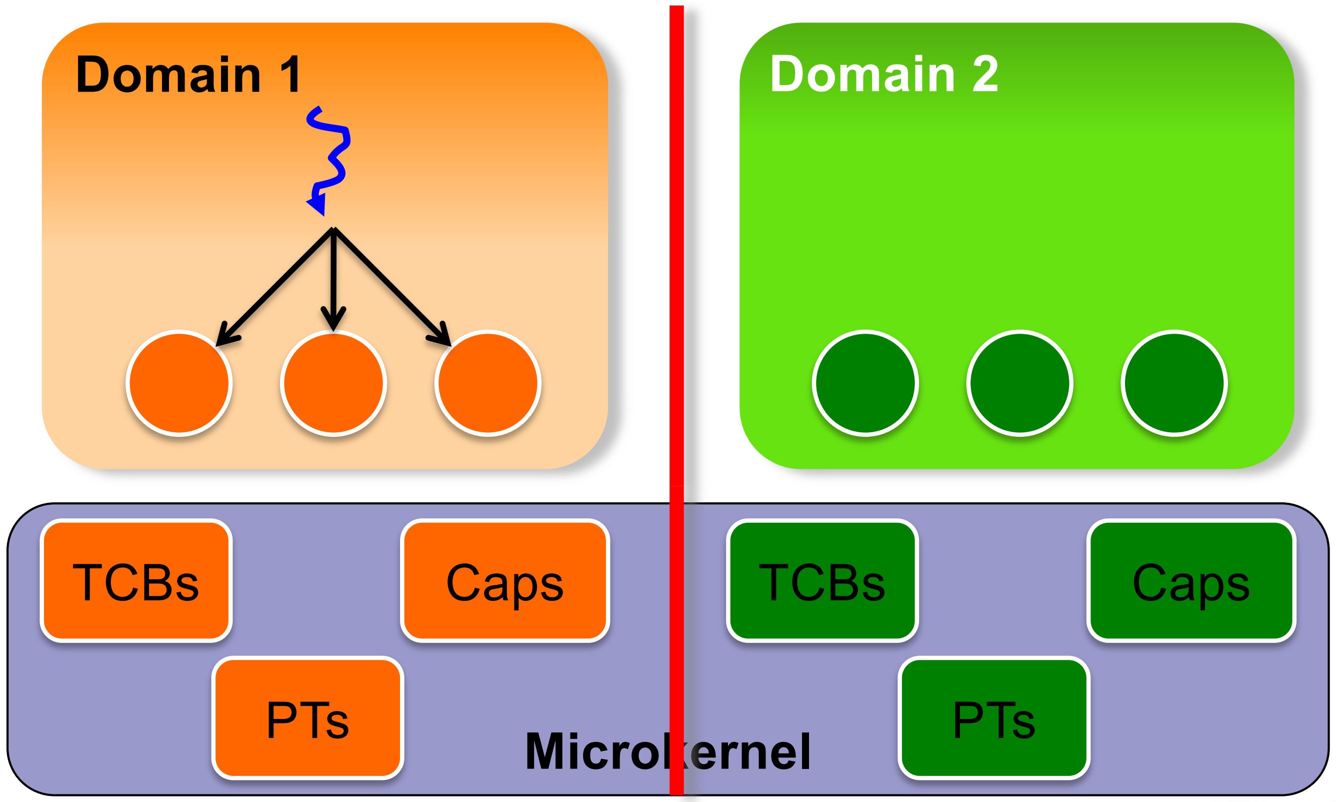
To prove:

- 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
  - Violation not observable in Domain-2!
- In progress – details in Gerwin's talk

seL4 as Basis for Trustworthy Systems



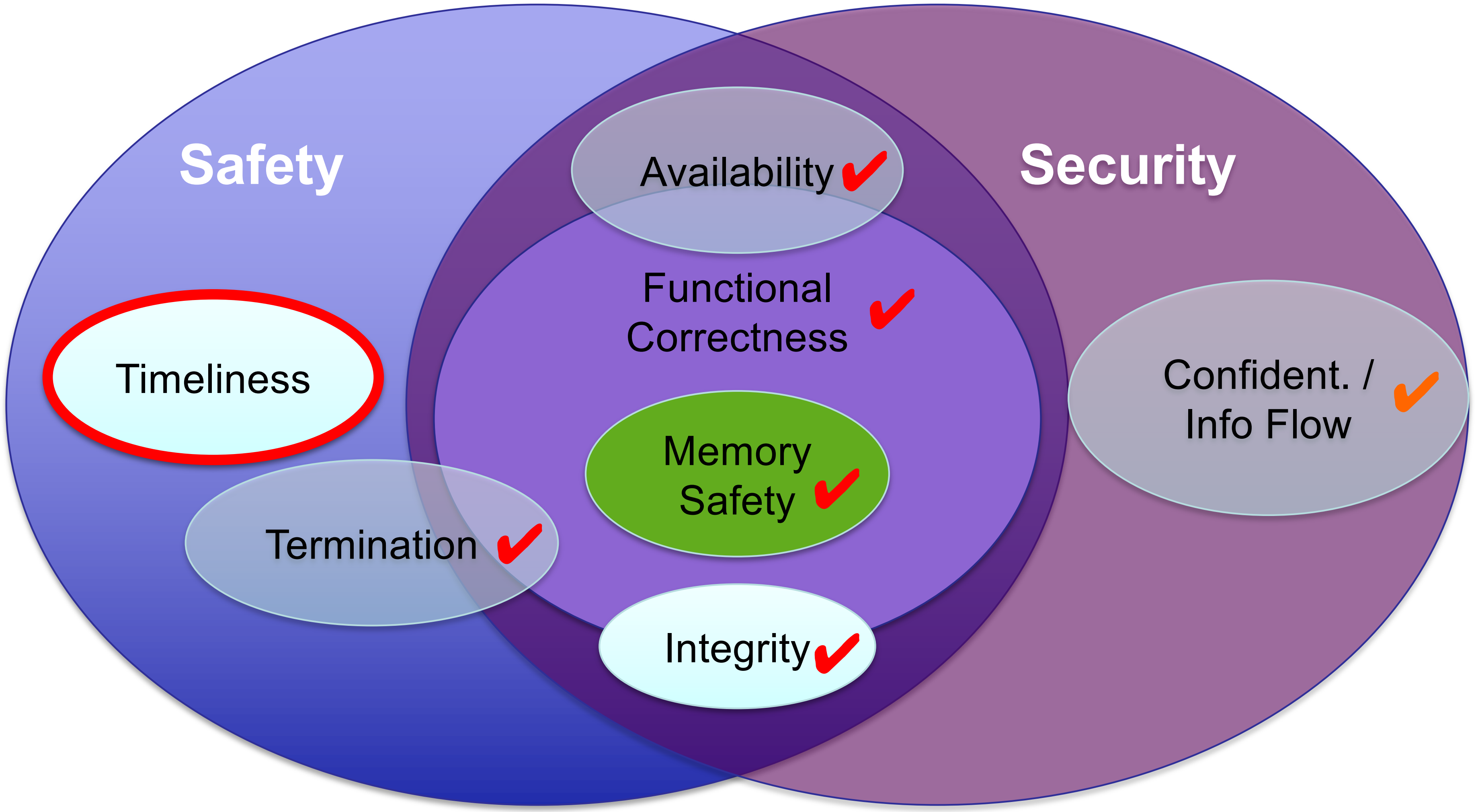
Availability is Trivially Ensured at Kernel Level



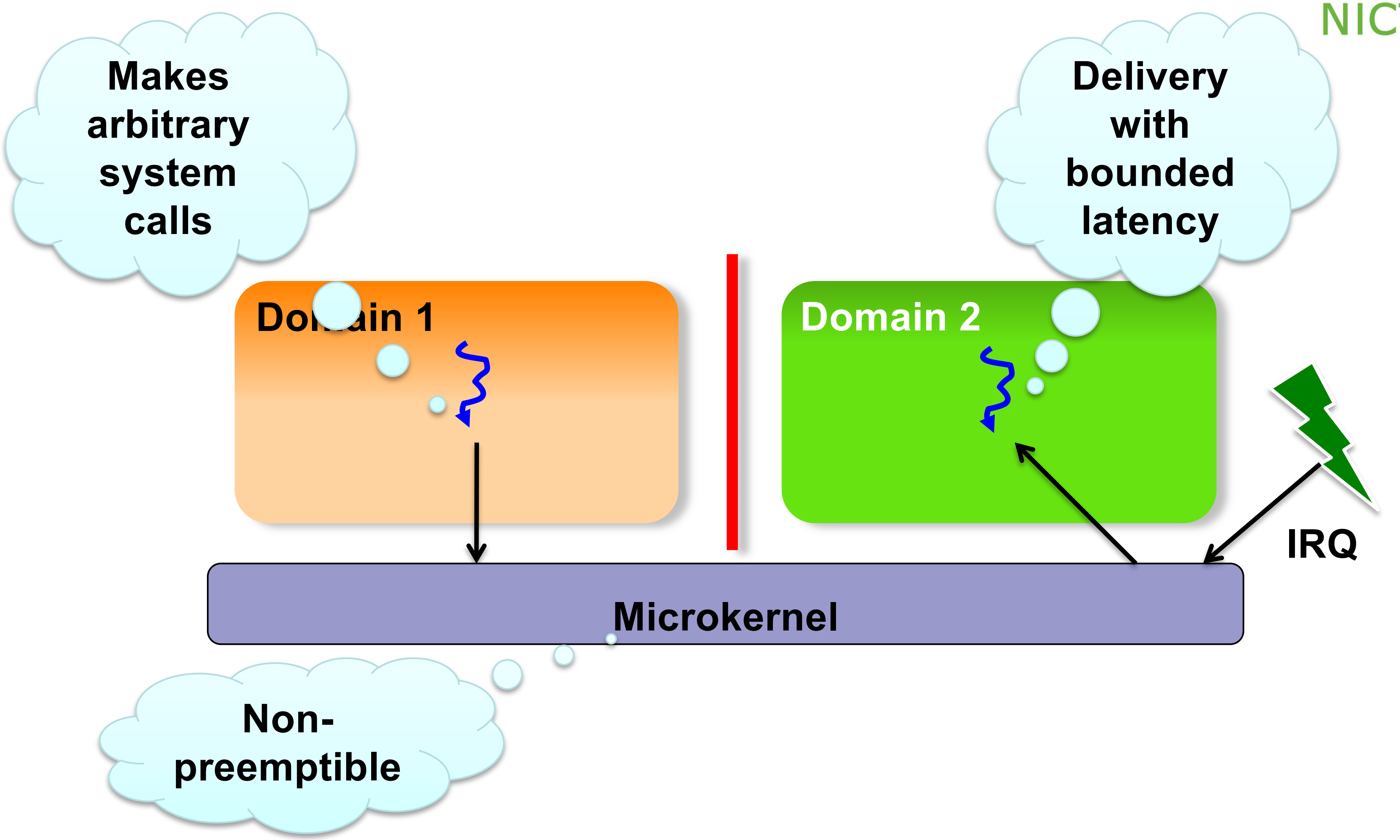
Managing resource availability is user-level issue!

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

seL4 as Basis for Trustworthy Systems

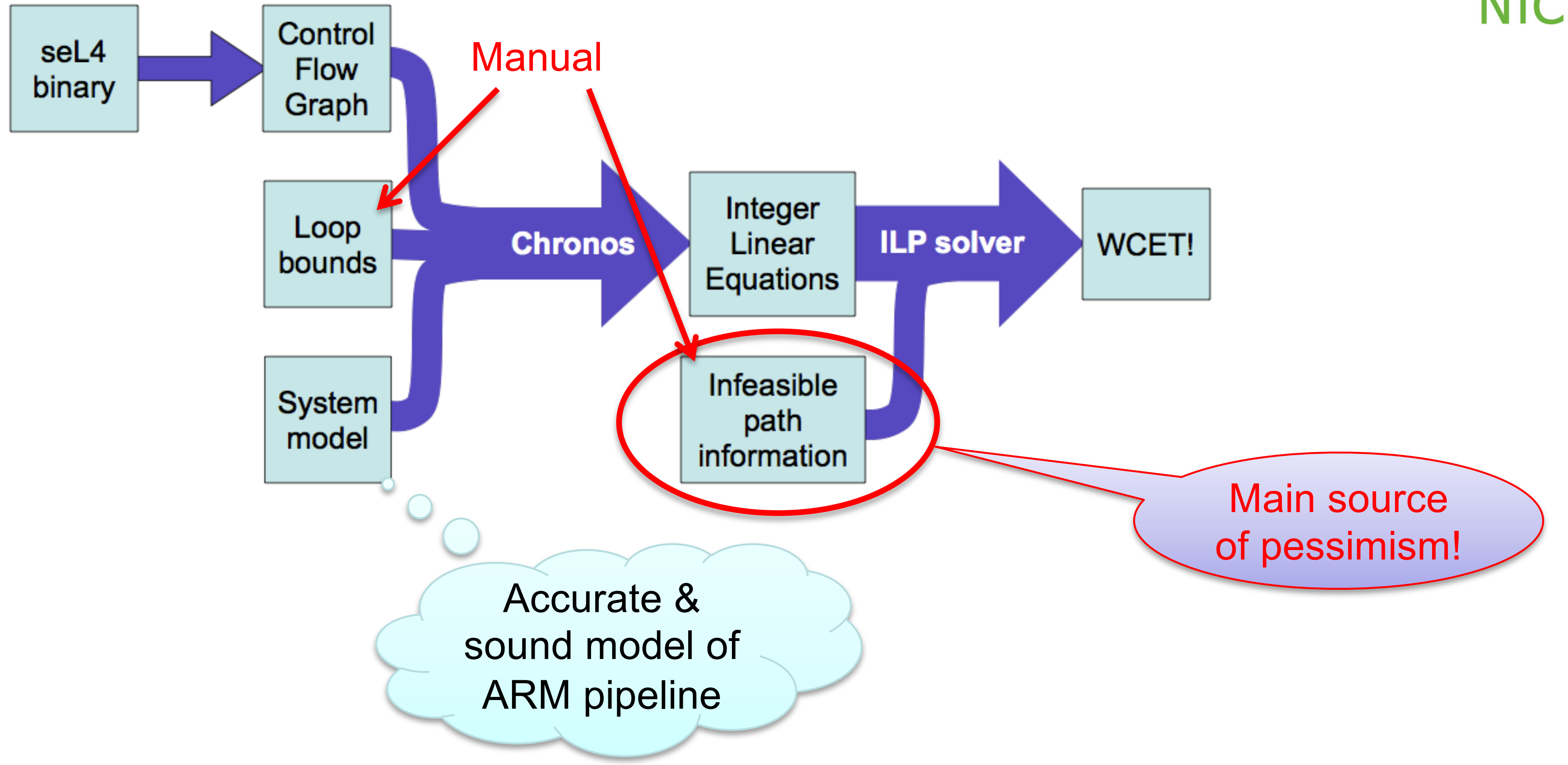


Timeliness



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

WCET Analysis Approach



Result: WCET >1 sec!

- Pessimism of analysis (loop bounds, infeasible paths)

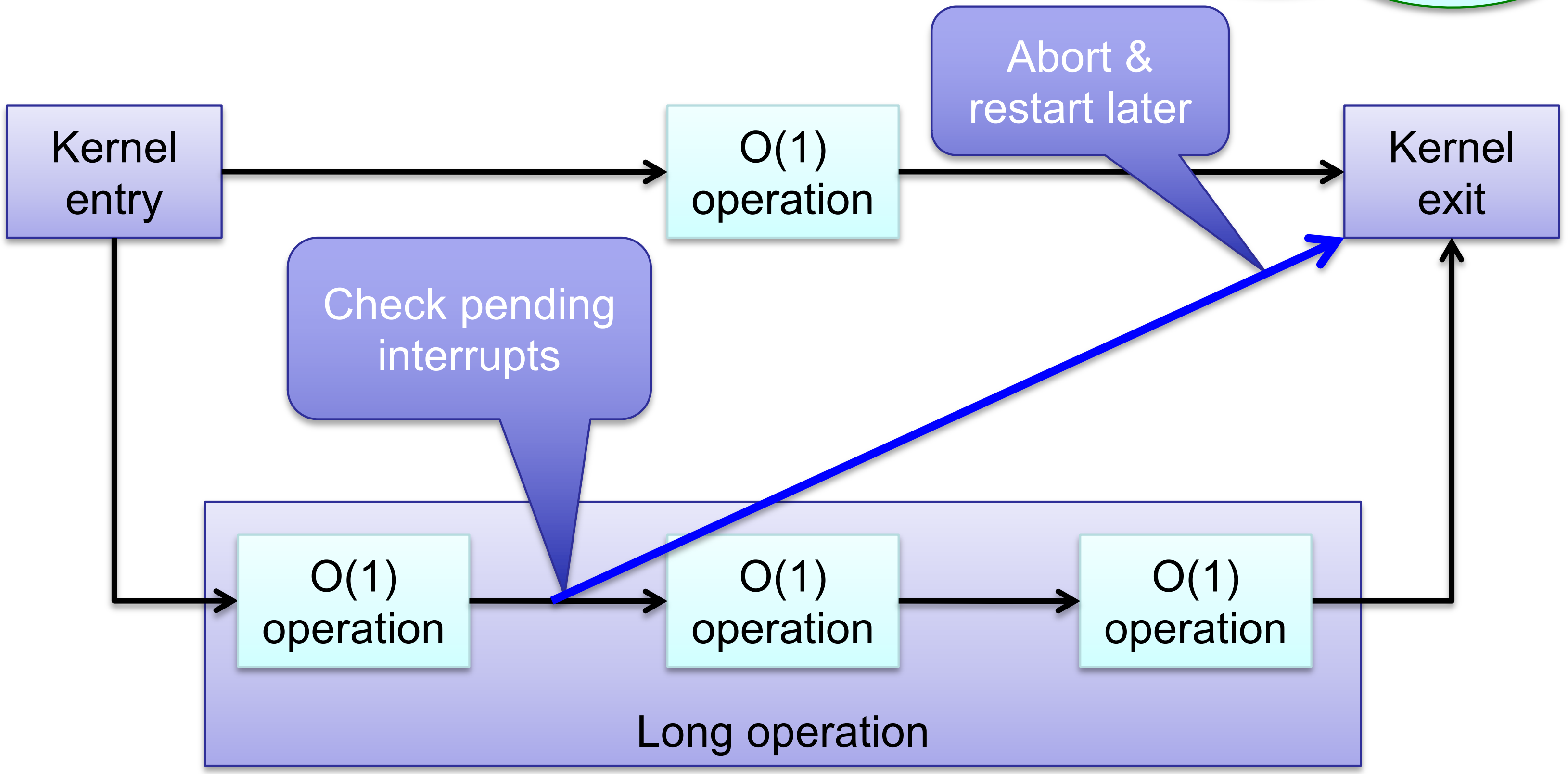
⇒ Manual elimination of infeasible paths

- Result: 600 ms ☹

Improving Real-Time Behaviour of seL4

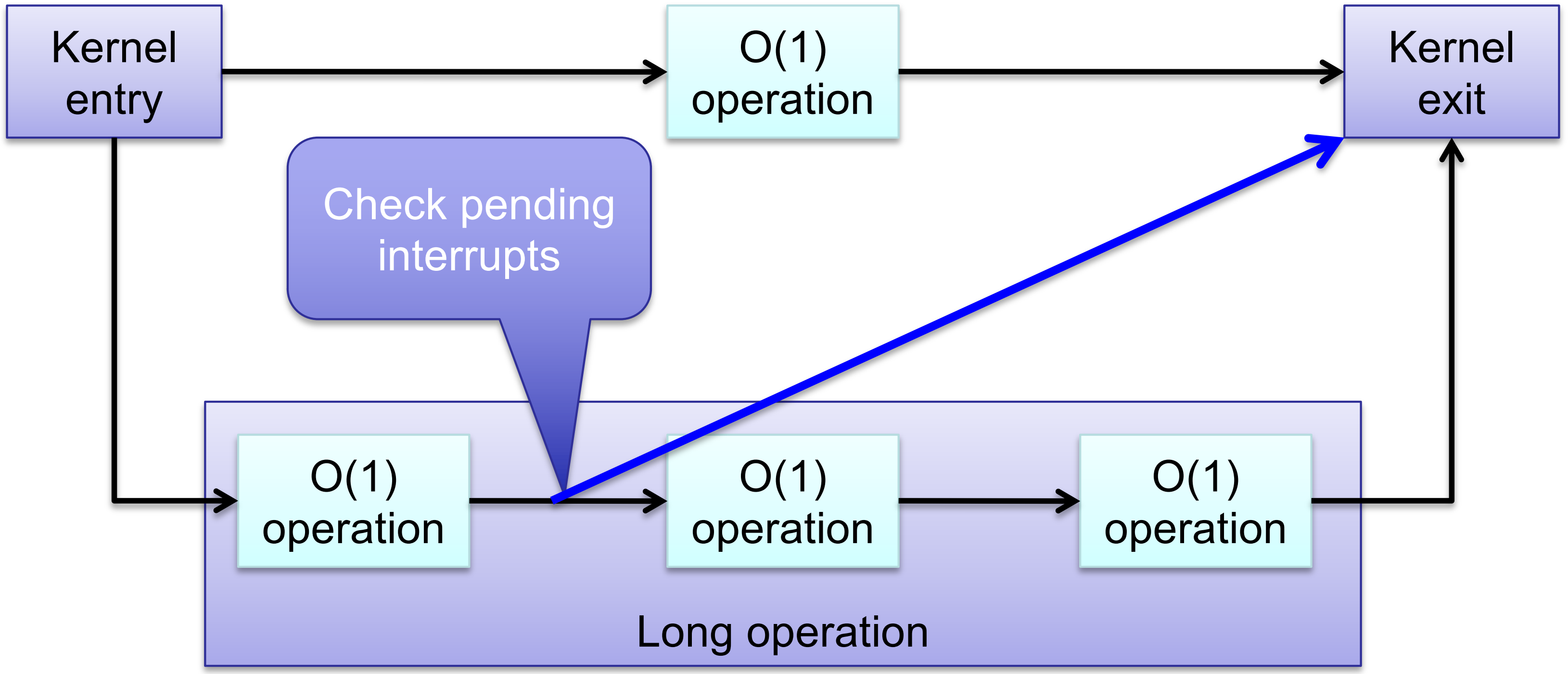
- Challenge: Improving WCET while
  - retaining ability to verify
  - maintaining high average-case performance

Event-oriented kernel running with interrupts disabled!

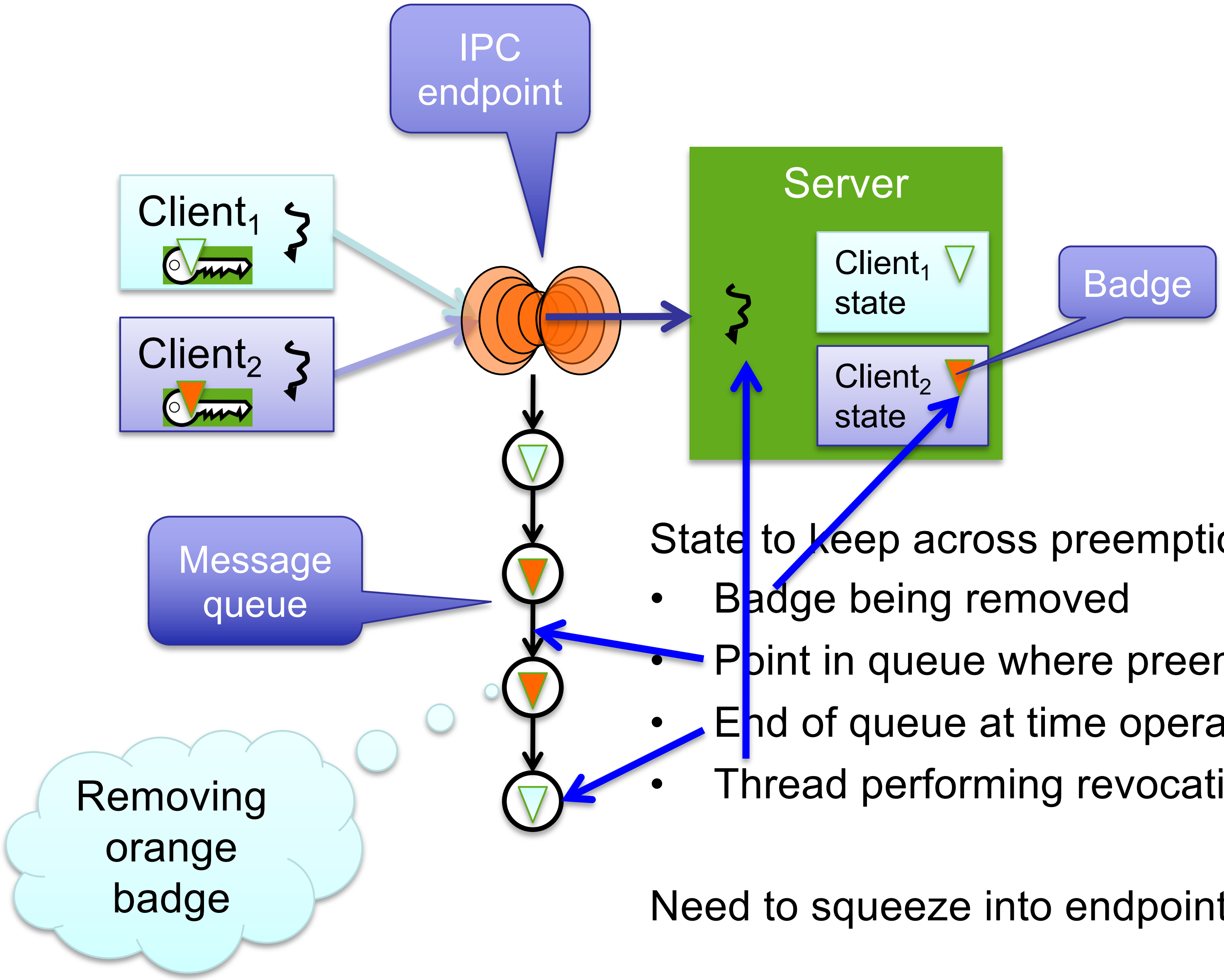


Placing Preemption Points

- 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



Example: Revoking IPC "Badge"



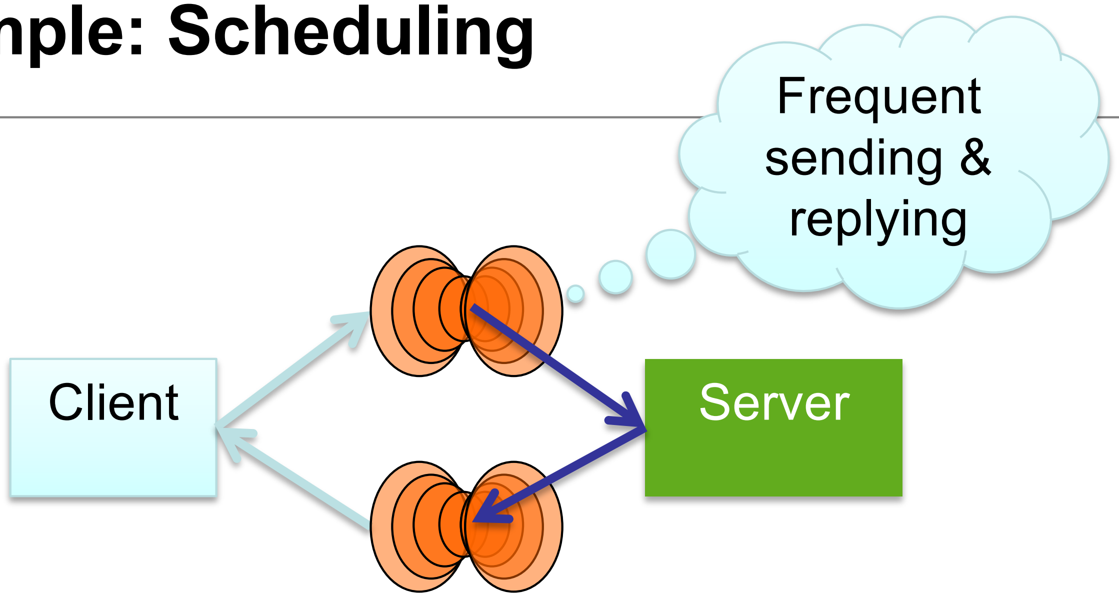
State to keep across preemptions

- Badge being removed
- Point in queue where preempted
- End of queue at time operation started
- Thread performing revocation

New invariants!

Need to squeeze into endpoint data structure!

Example: Scheduling



Blocking IPC: Each send/receive blocks a thread!

- Remove thread from ready queue
- Will be re-inserted in the reply!

Idea: Leave blocked threads in ready queue!

Classical L4 optimisation "lazy scheduling"

- Good average-case performance

Lazy Scheduling

Scheduler must clean up the mess:

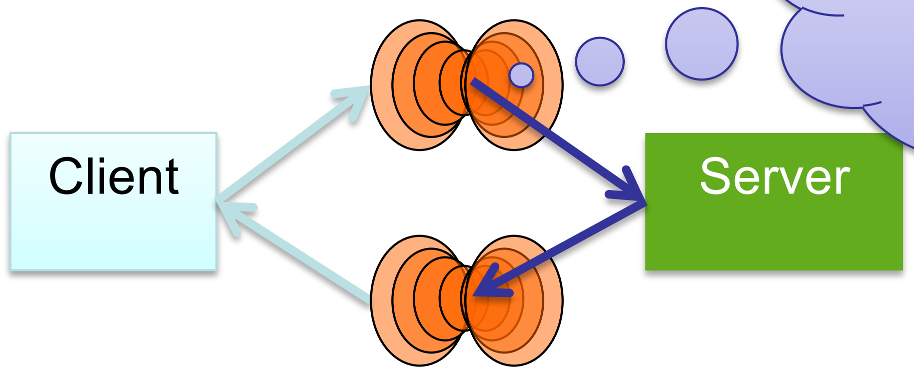
```
thread_t schedule() {
    foreach (prio in priorities) {
        foreach (thread in runQueue[prio]) {
            if (isRunnable(thread))
                return thread;
            else
                schedDequeue(thread);
        }
    }
    return idleThread;
}
```

Scheduling becomes unbounded!

But scheduling cannot be preempted!

Must re-factor code

Improving Scheduling



Benno scheduling: Do not insert newly unblocked thread in ready queue!

O(1) cleanup !

Blocking IPC: Each send/receive *unblocks* a thread!

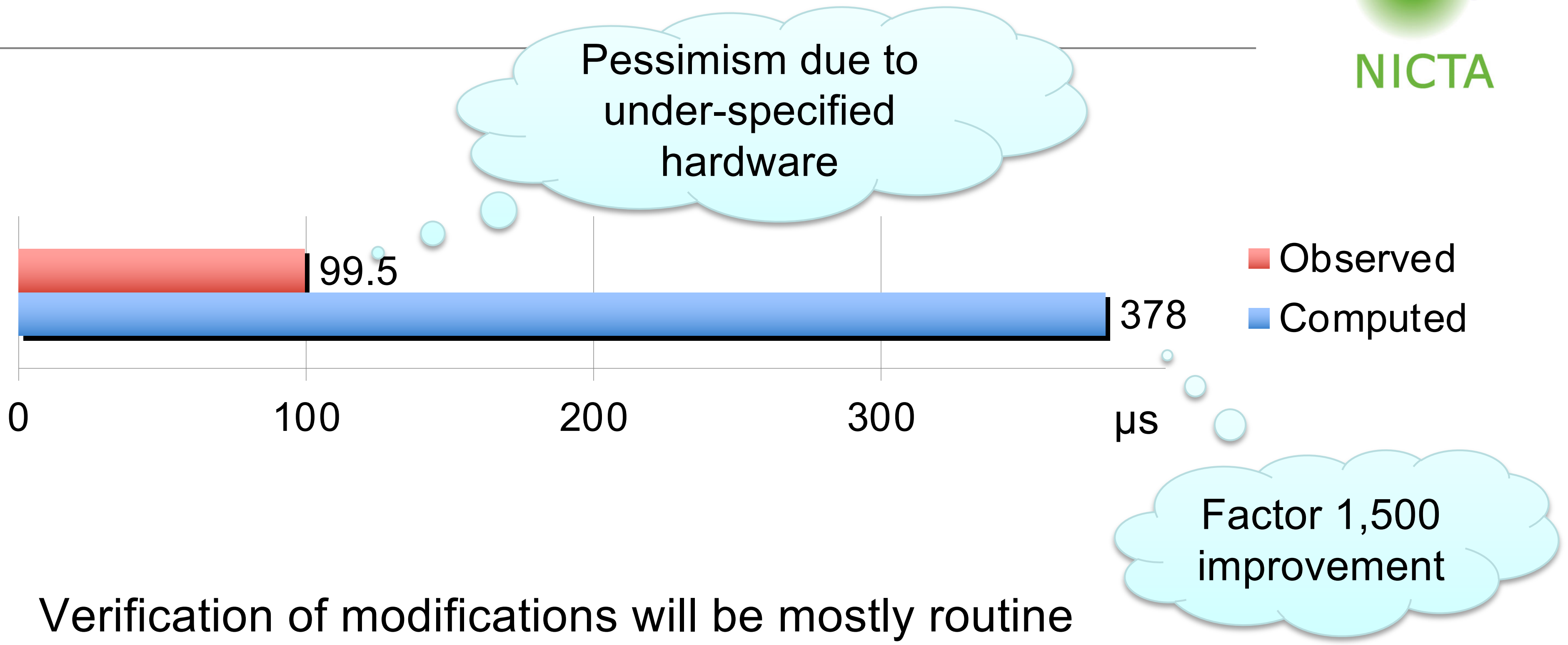
- At preemption time, insert presently running thread into ready queue

New scheduling invariant: *all threads in ready queue are runnable*

- Same average-case performance as lazy scheduling
- Scheduling WCET becomes O(1)

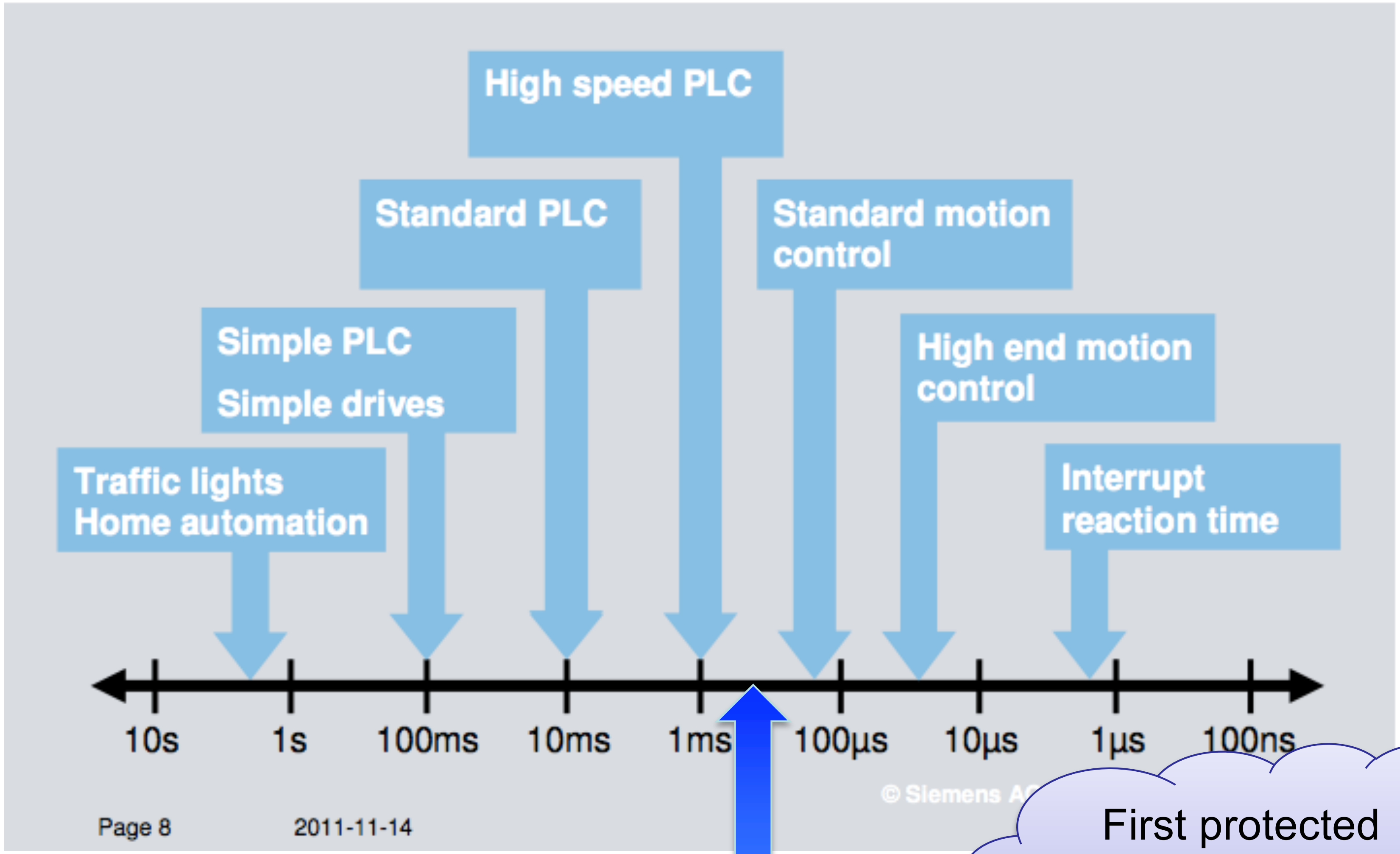
Use priority bitmap and CLZ instruction

Result



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

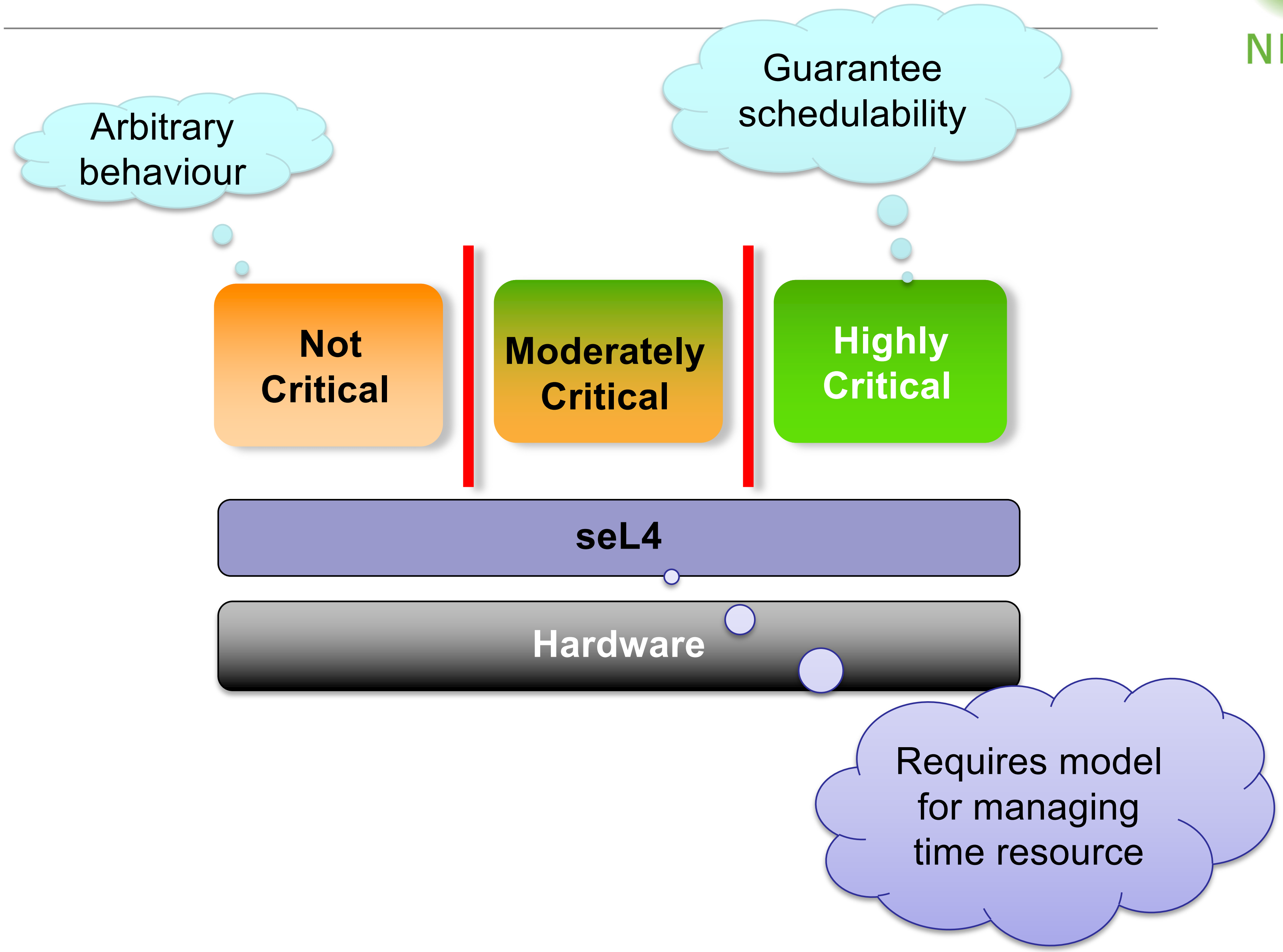
RT Requirements in Industrial Automation



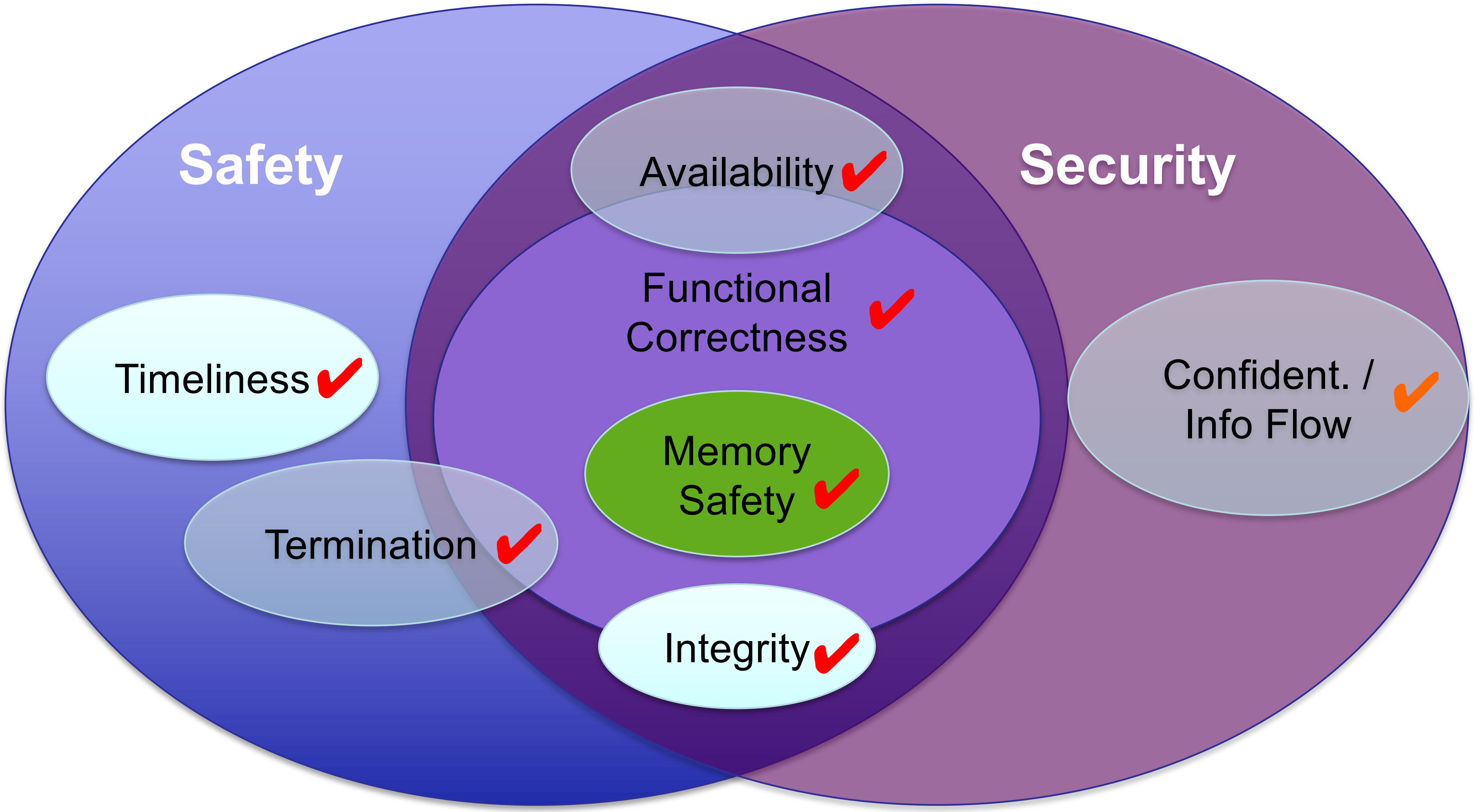
seL4 today

First protected RTOS with sound WCET analysis

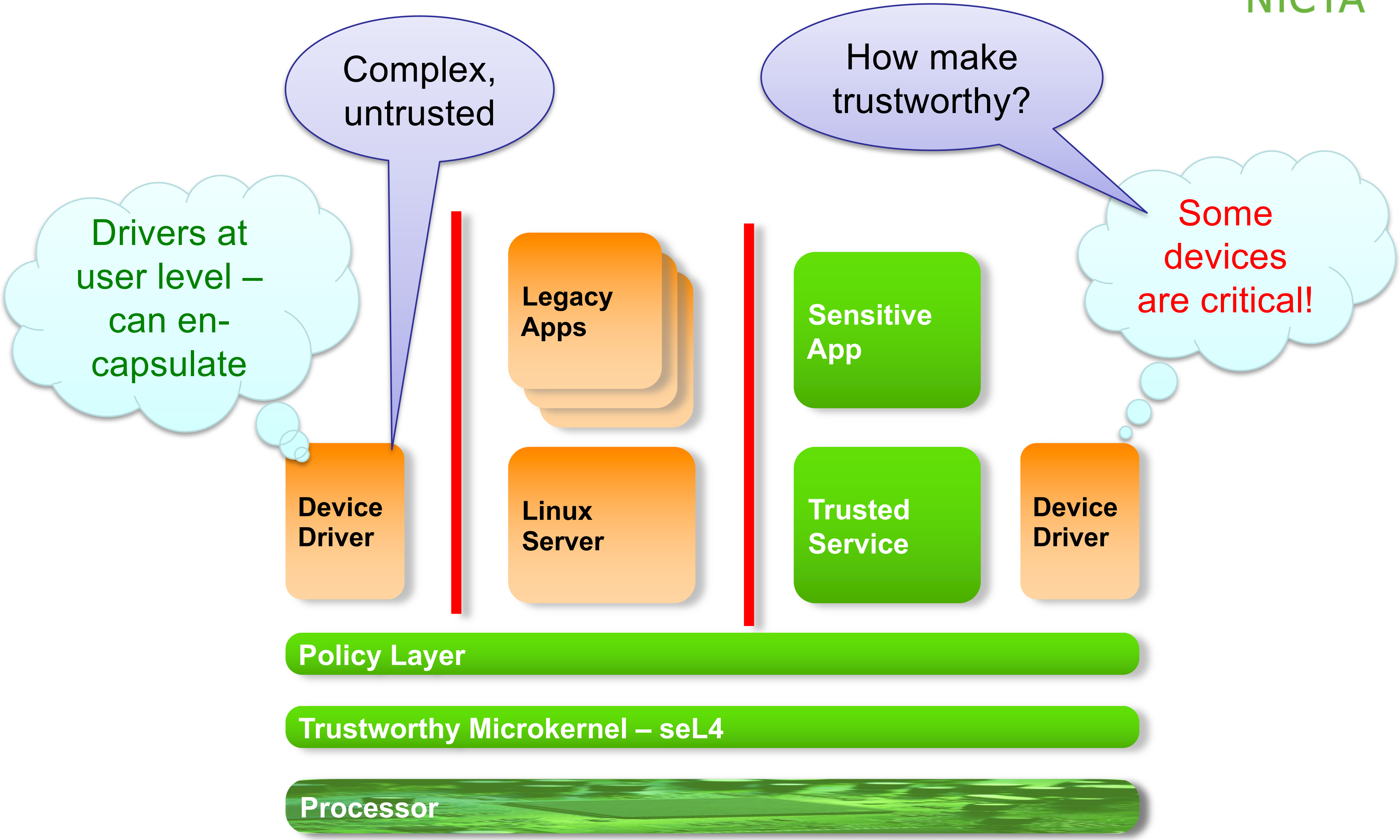
Future: Whole-System Schedulability



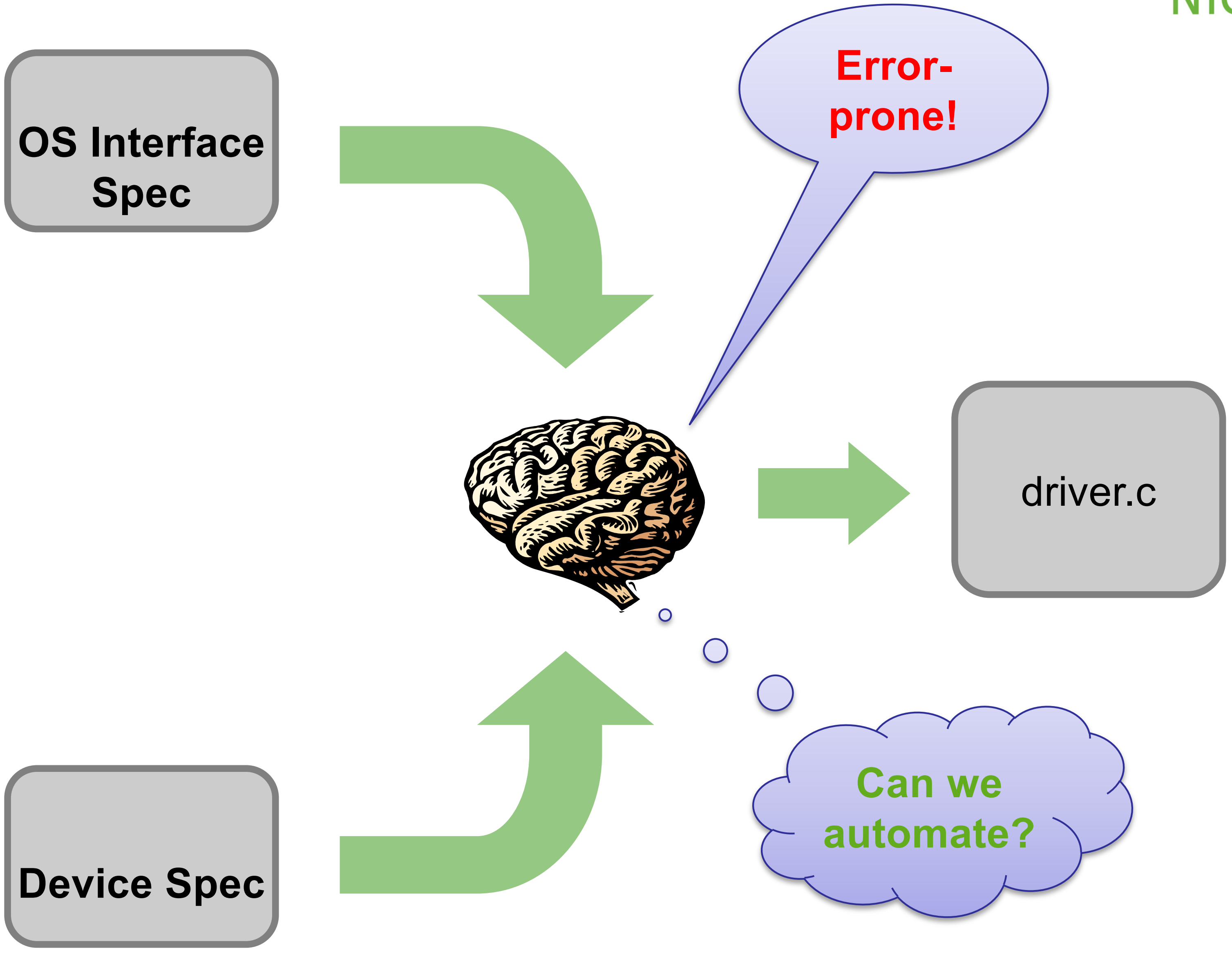
seL4 for Safety and Security



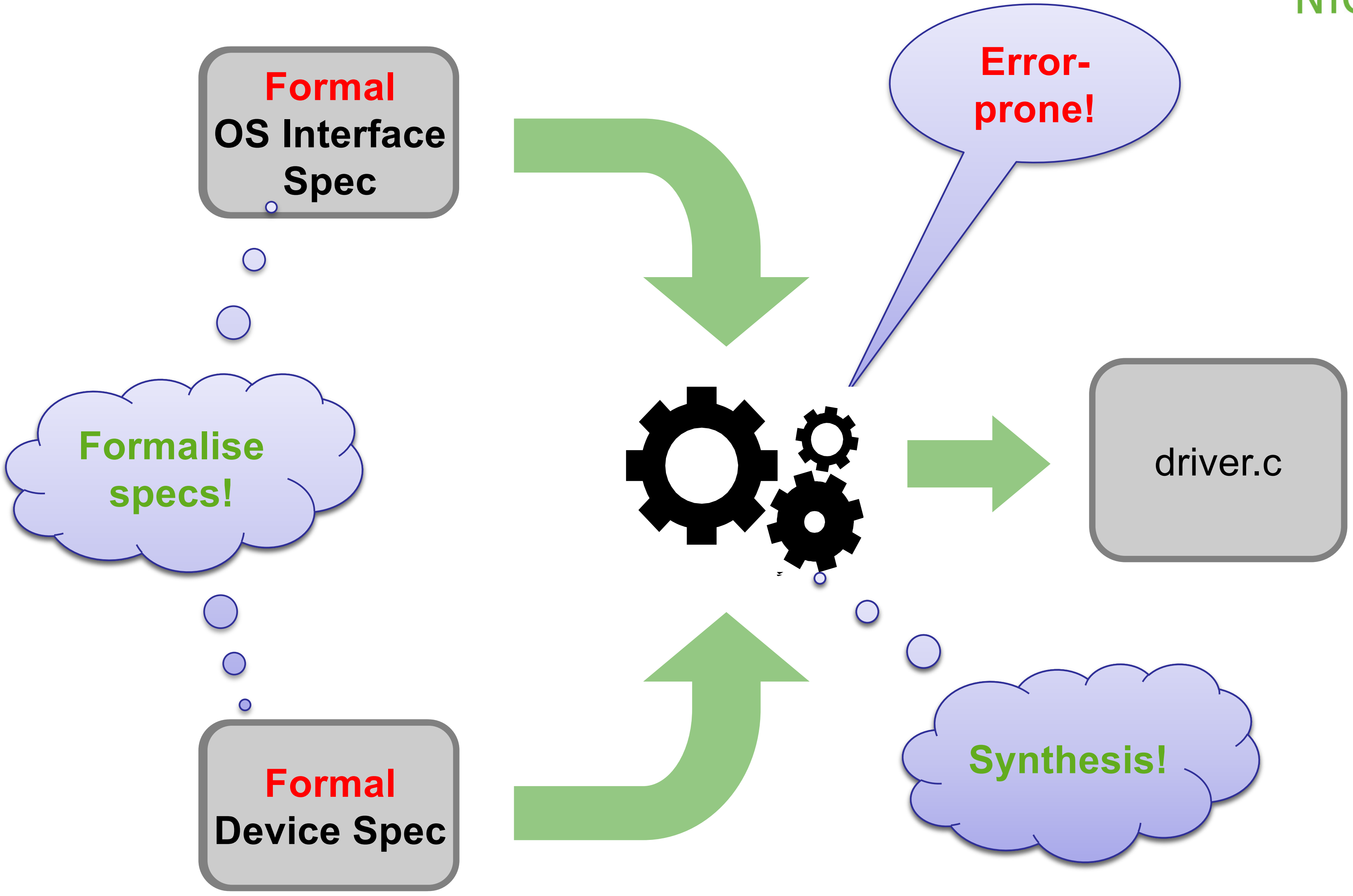
Device Drivers



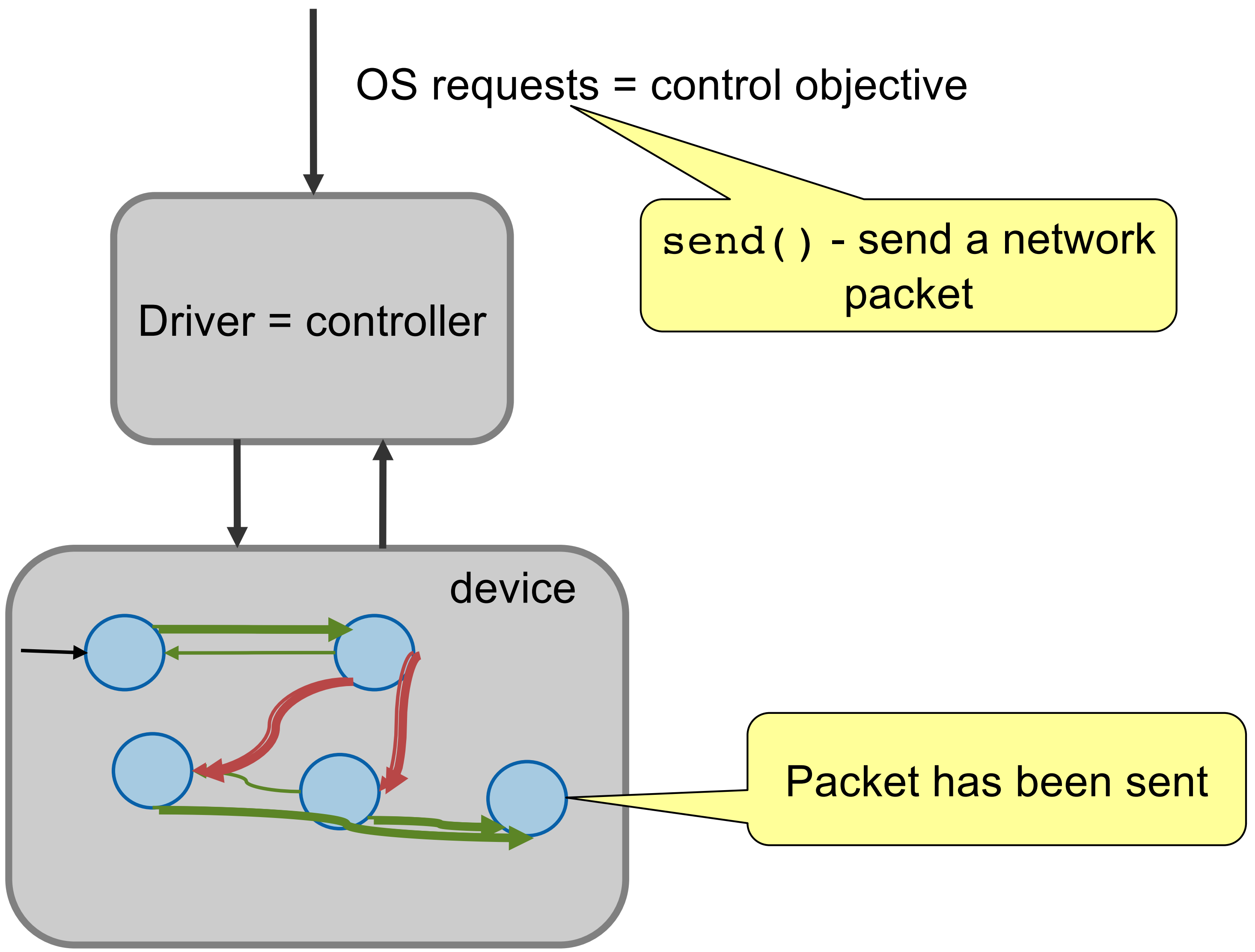
Driver Development



Driver Development

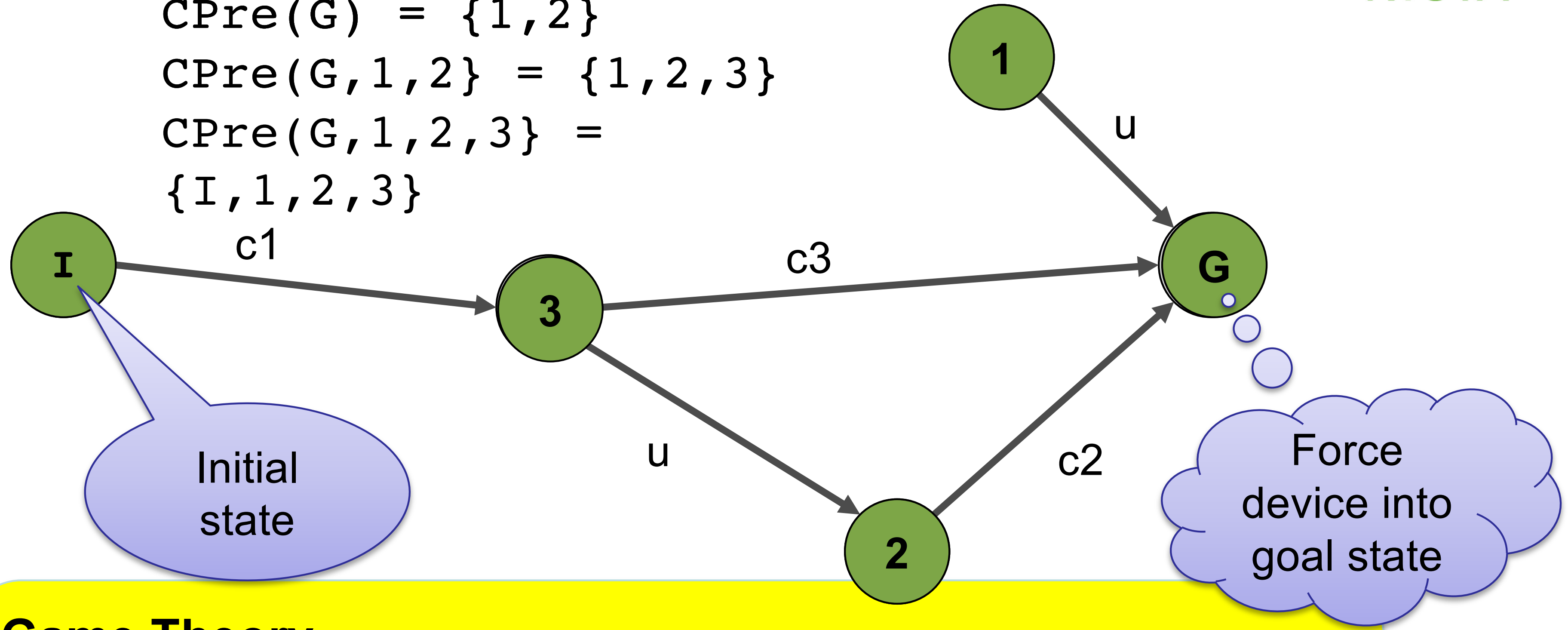


Driver Synthesis as Controller Synthesis


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}
```



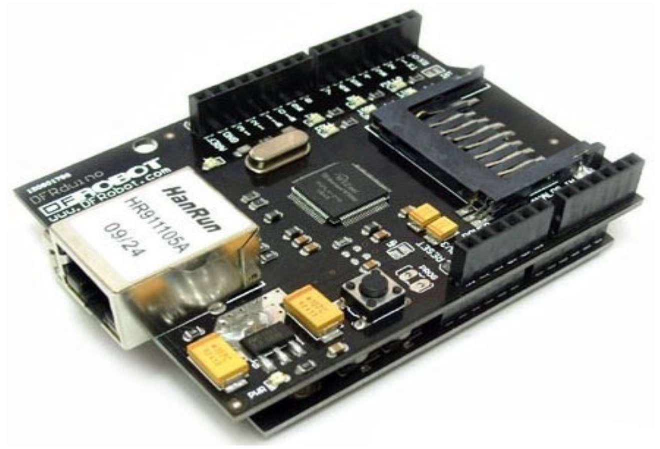
Game Theory

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

Device driver!

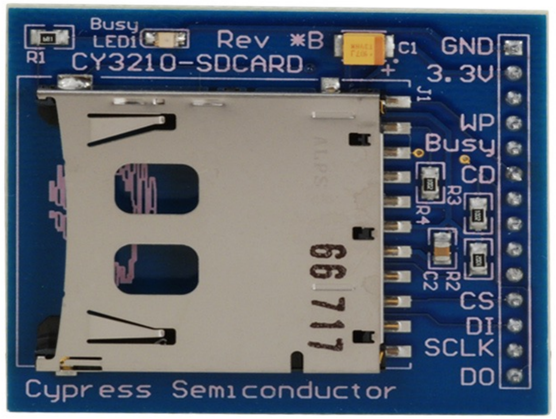
Drivers Synthesised (To Date)

IDE disk controller



W5100 Eth shield

Asix AX88772
USB-to-Eth adapter



SD host controller

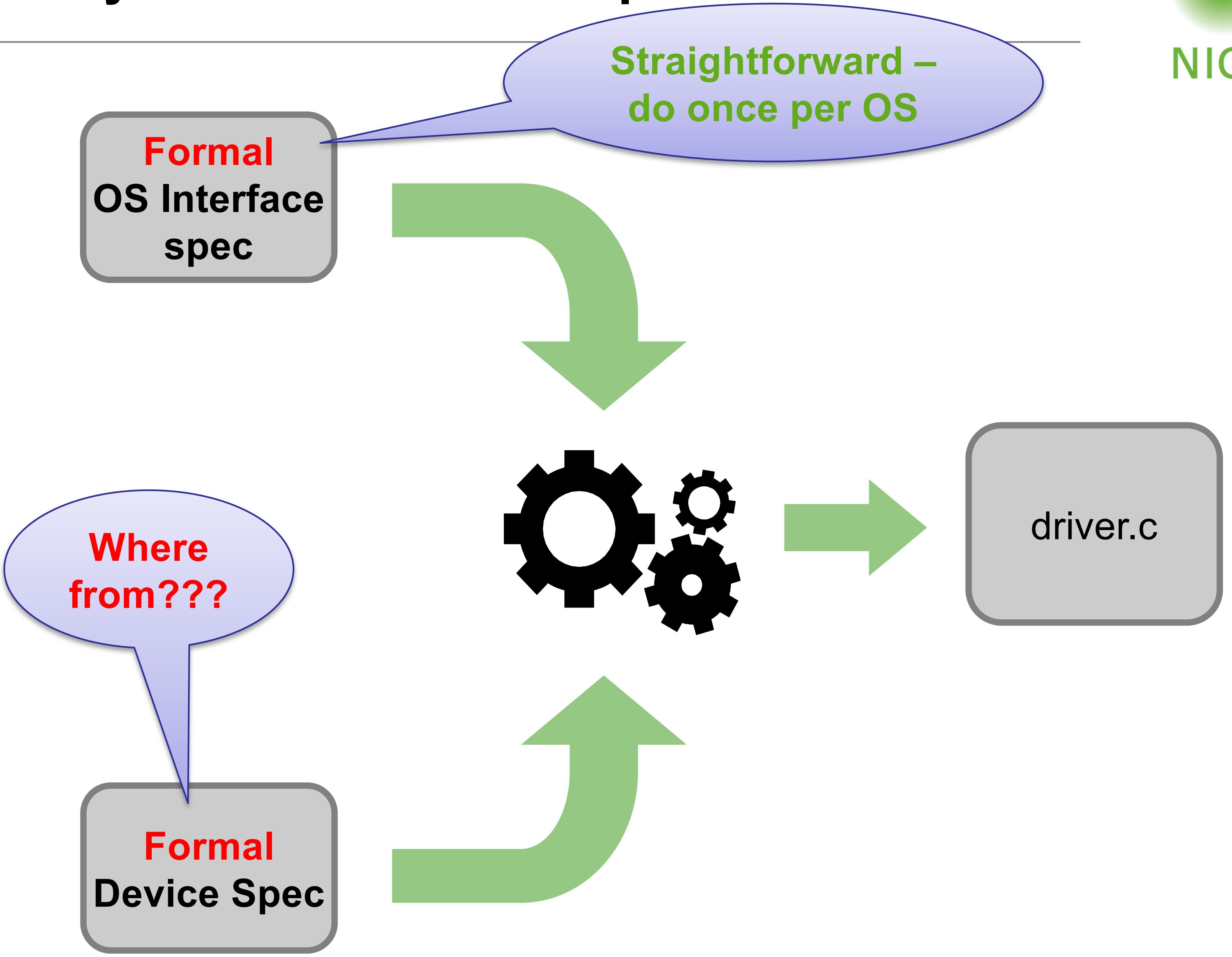
Driver Synthesis Challenges

- State explosion
  - Symbolic state space representation, predicate abstraction
    - **done**
- Synthesis with imperfect information
  - Generalisation of a perfect information strategy
    - **work in progress**

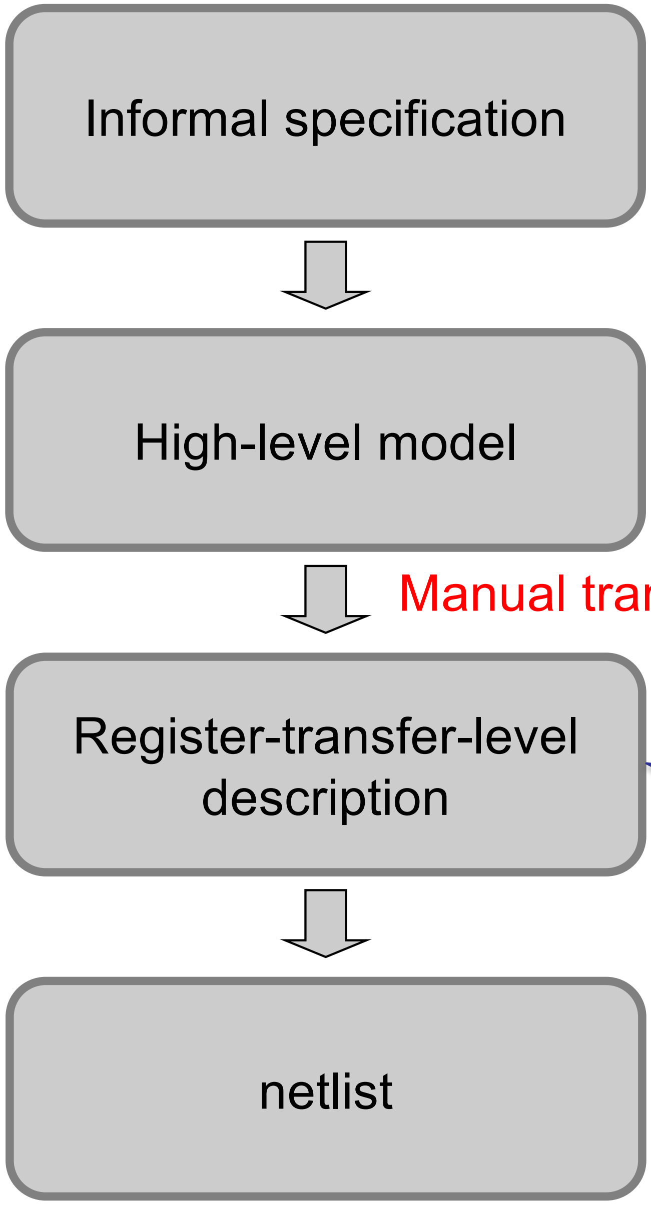
Un-observable state transitions

- Efficient C code generation
  - Avoid code bloat
    - **work in progress**
- Support for DMA
    - **this year**
- Verification: is the synthesised driver correct?
  - Errors in the synthesis tool
    - **future work**

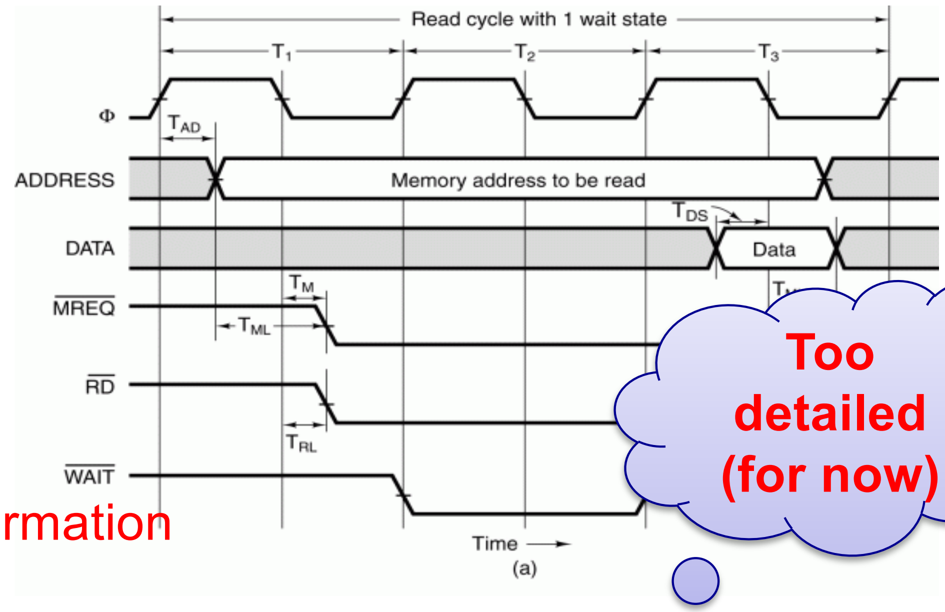
Driver Synthesis: Interface Specs



Hardware Design Workflow



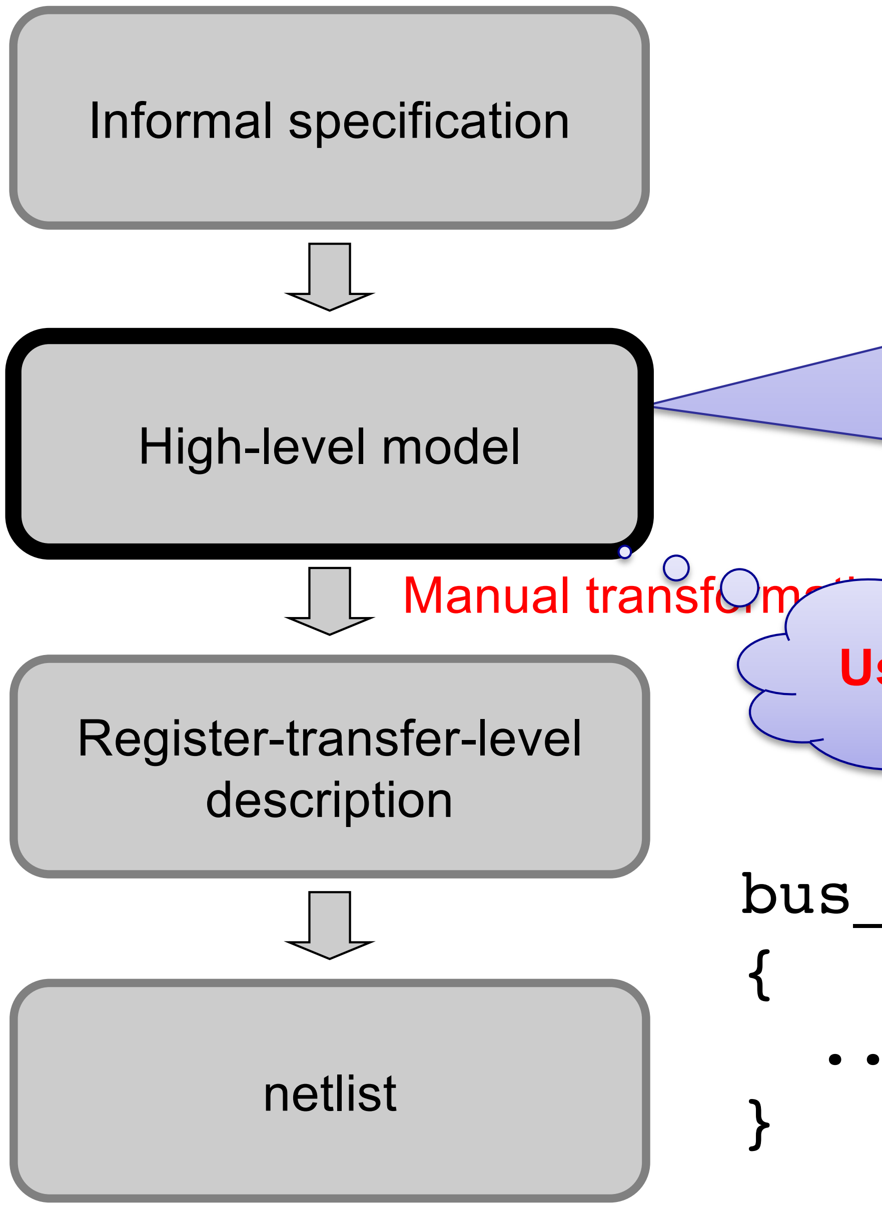
Manual transformation



Too detailed (for now)

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

Hardware Design Workflow

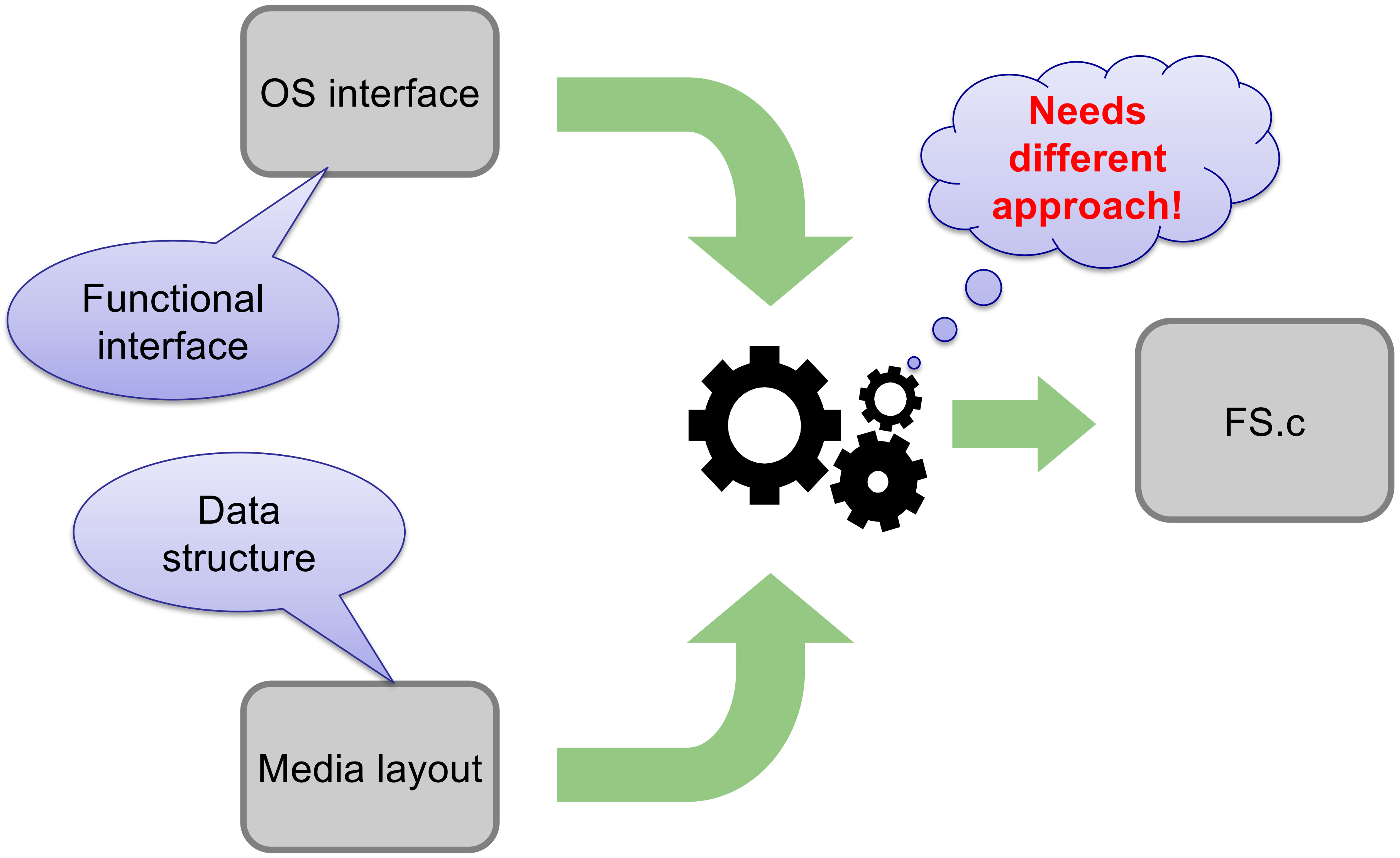


- Captures external behaviour
- Abstracts away structure and timing
- Abstracts away the low-level interface

Use for now

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

From Drivers to File Systems?



Does it Work in the Real World?

Must build realistic systems!

- 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

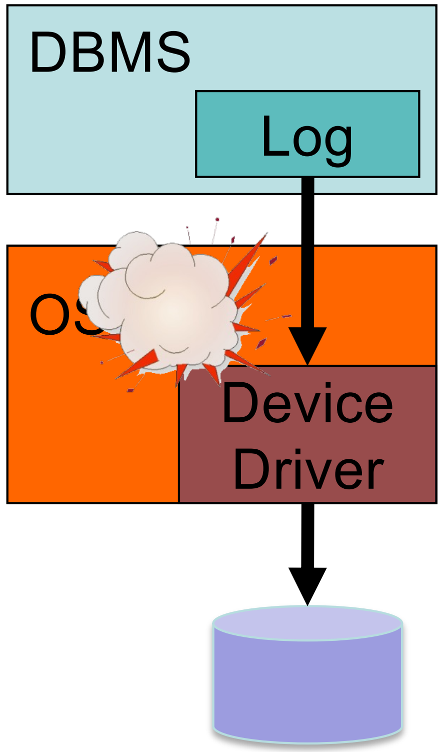
Example: RapiLog – Fast DBMS without sync()

Databases require durability guarantees

- In the presence of failures (OS crash, power)
- Ensured typically by write-ahead logging
  - Flush log before continuing processing
  - Disk writes on critical path

Idea: Avoid synchronous I/O

- using guaranteed dependability of seL4

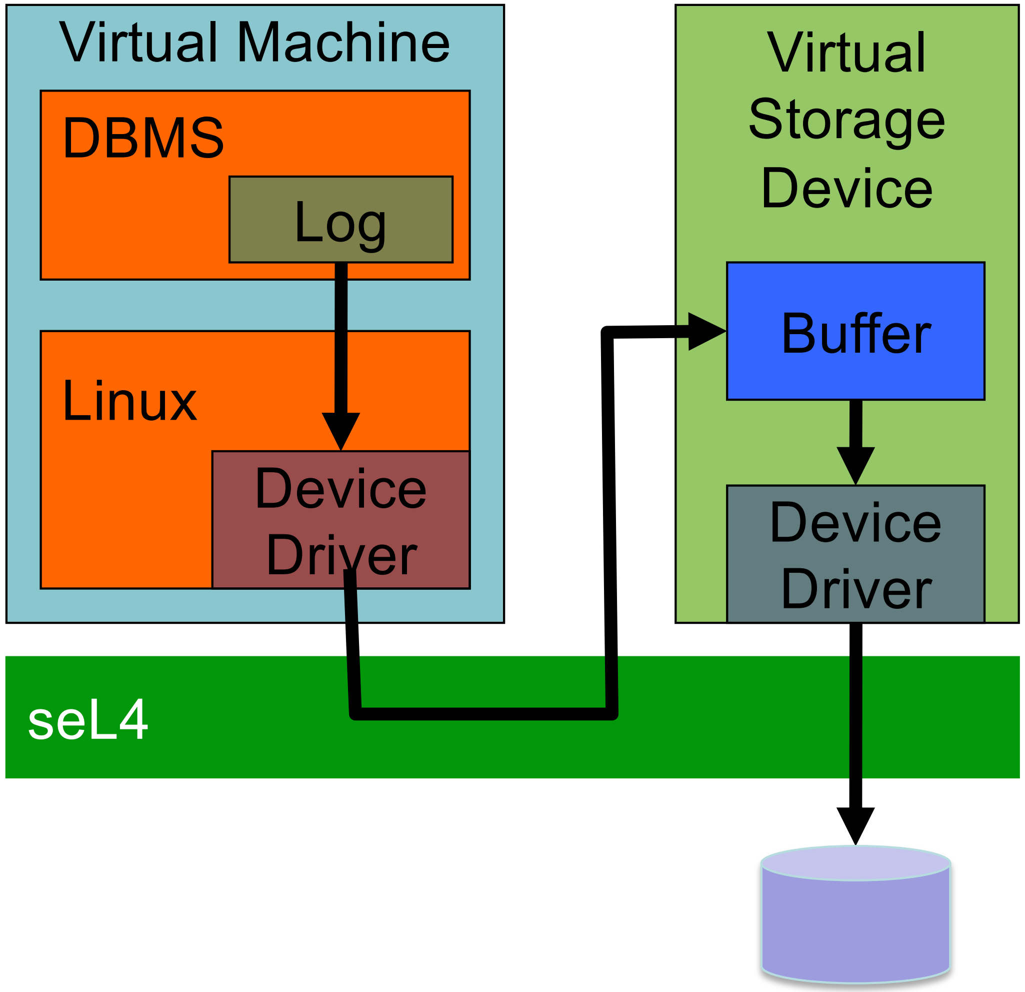


Example: RapiLog – Fast DBMS without sync()

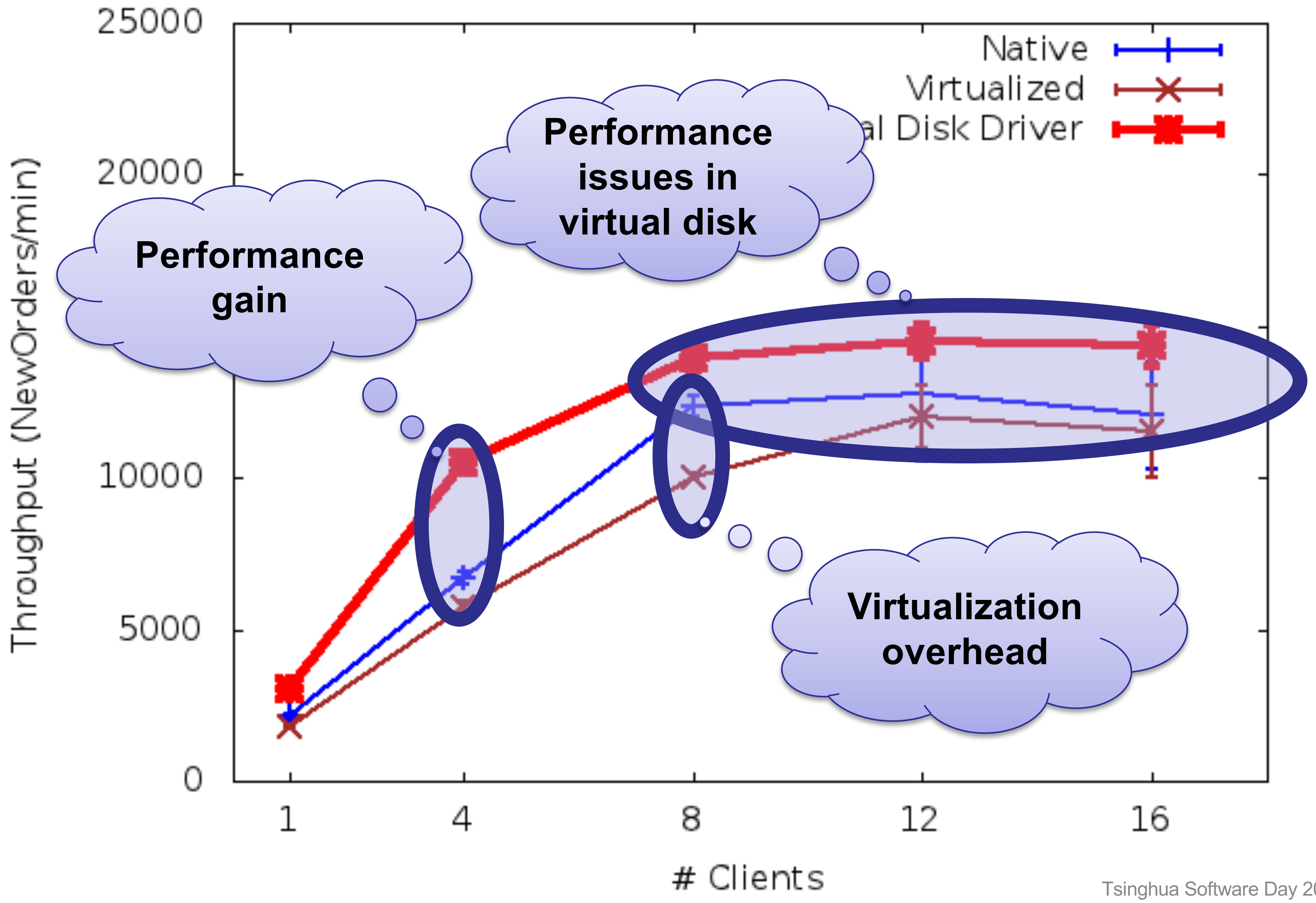
DBMS on seL4

- Using virtualized Linux

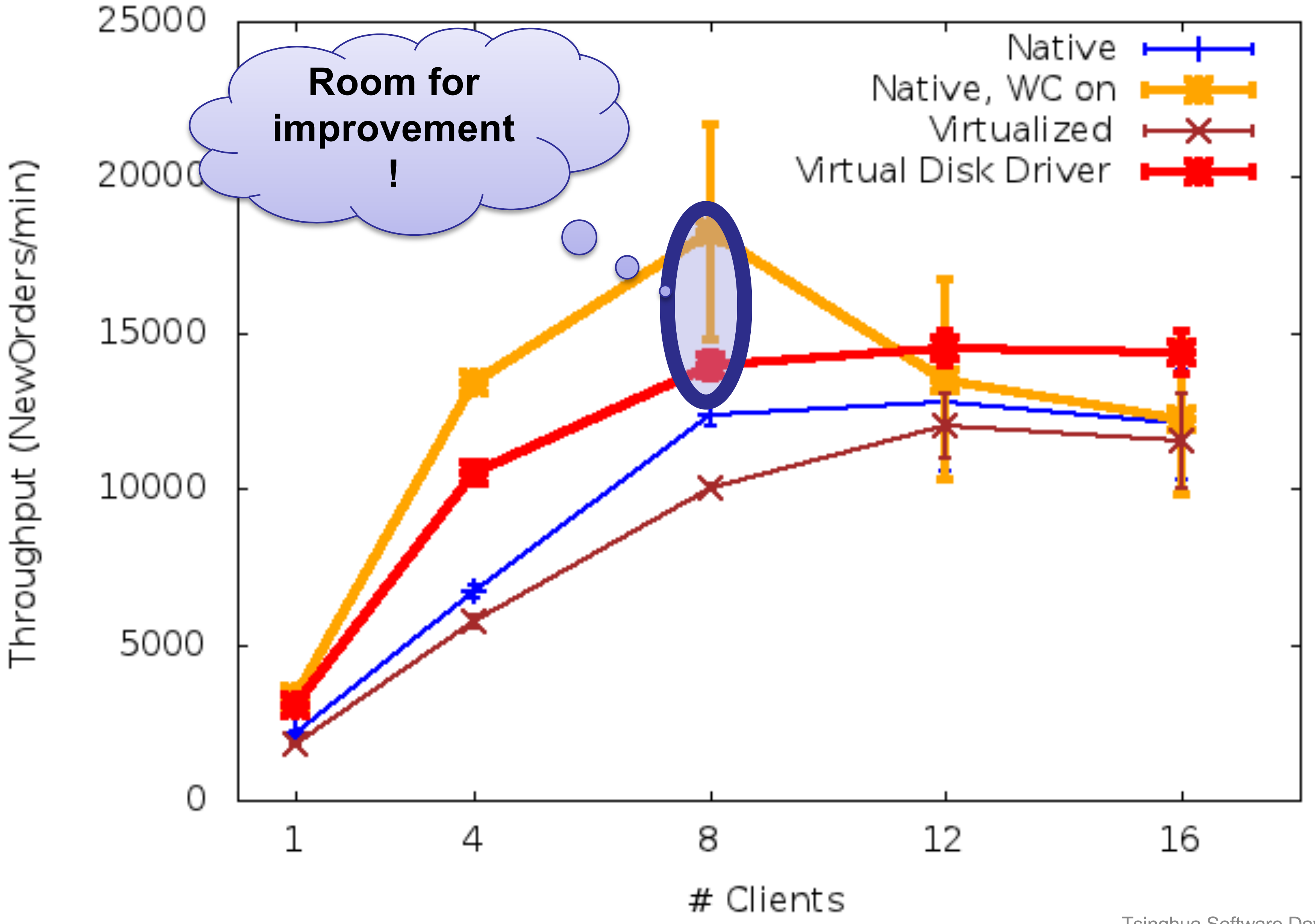
- Performance should matches unsafe (no-sync) operation on native Linux
- Benefits from driver synthesis



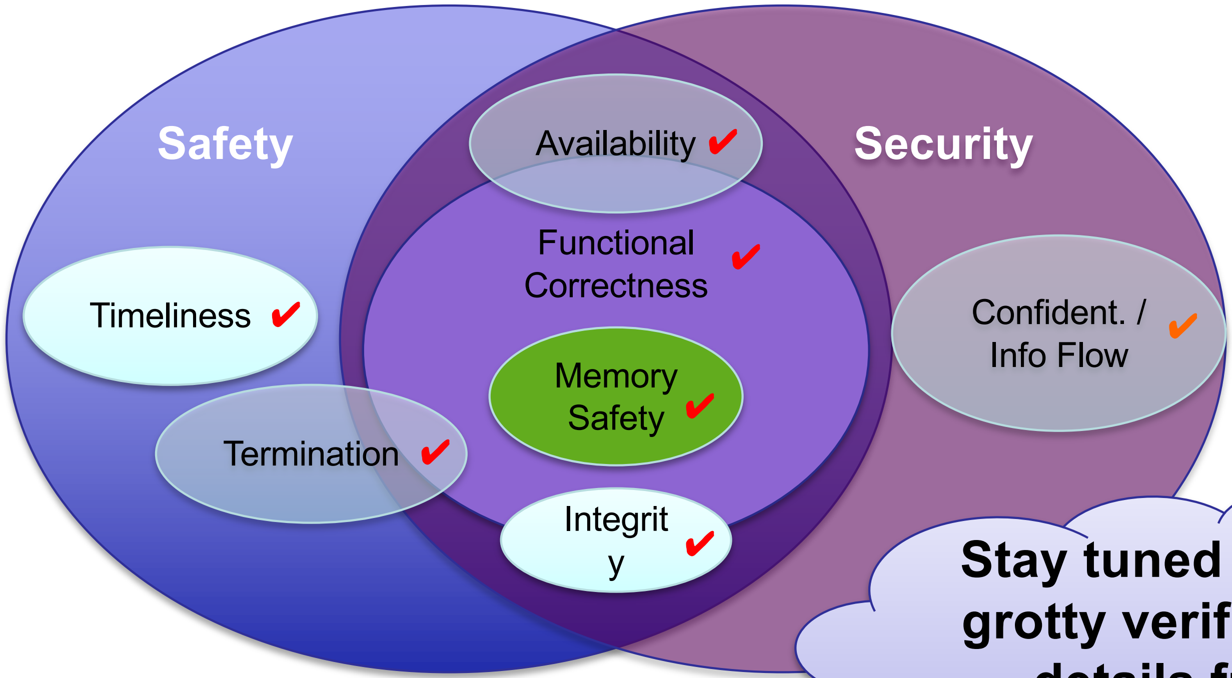
Initial Results: PostgreSQL Throughput



Initial Results: PostgreSQL Throughput



Trustworthy Systems Platform: Almost There!



Stay tuned for the grotty verification details from Gerwin!

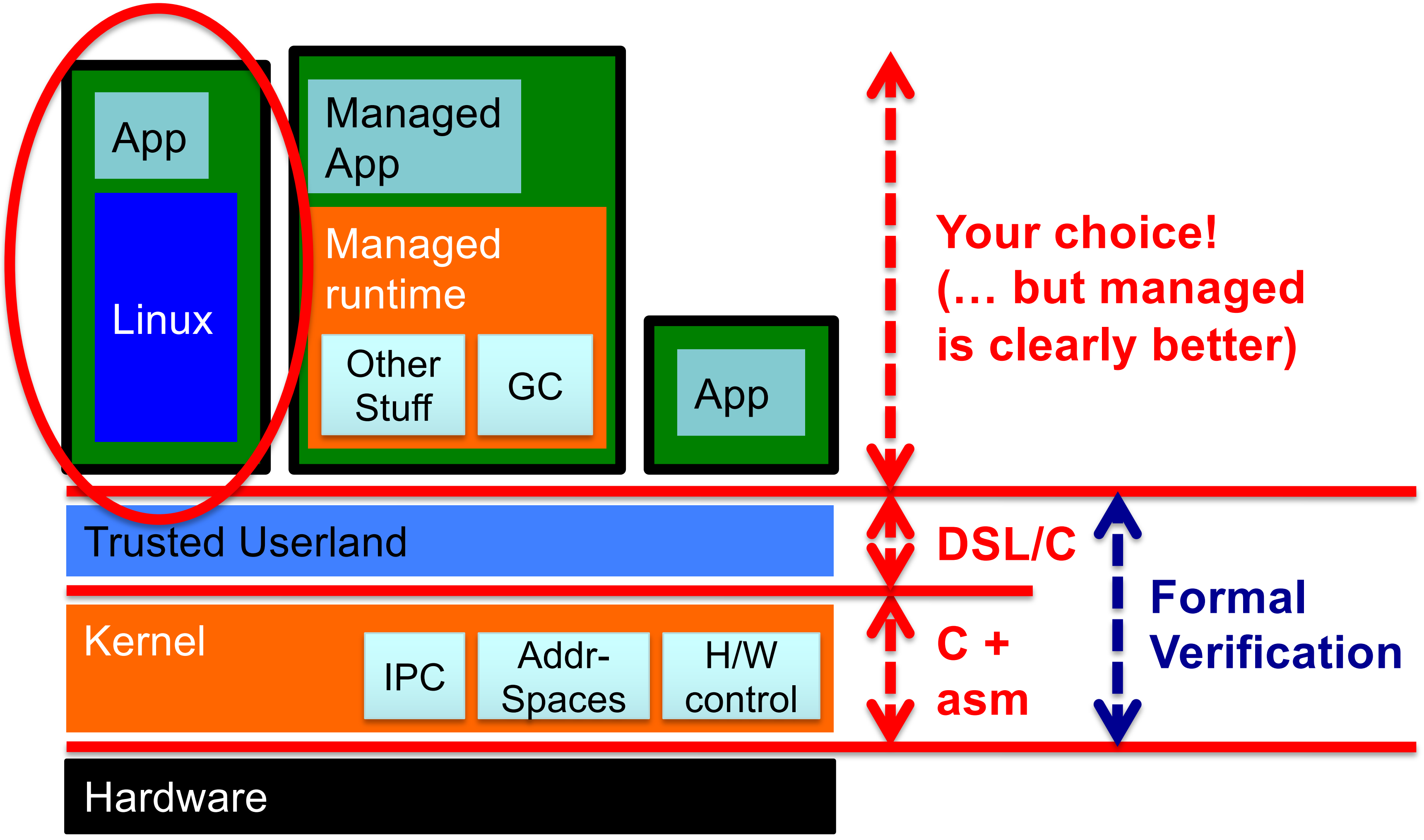
Thank You!

mailto:gernot@nicta.com.au

@GernotHeiser

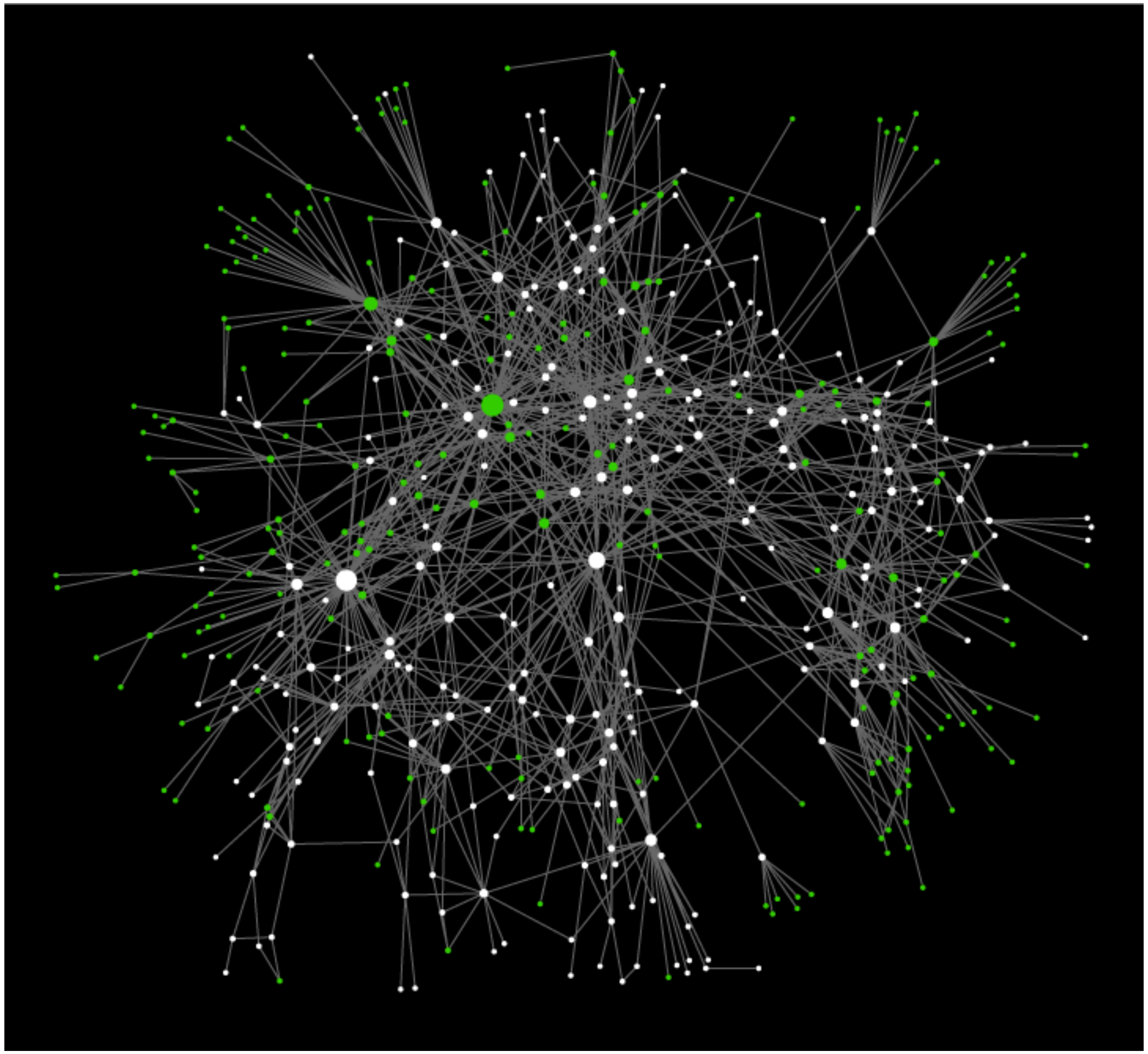
Google: “ertos”

Our View of Implementation Languages



seL4 Call Graph

NICTA



Verification vs Certification

Common Criteria: Military-Strength Security

Evaluation Level	Requirements	Functional Specification	Top Down Design	Imple-mentation	Cost
EAL1		Informal			
EAL2		Informal	Informal		
EAL3		Informal	Informal		
EAL4		Informal	Informal	Informal	
EAL5		Semi-formal	Semi-formal	Informal	
EAL6	Formal	Semi-formal	Semi-formal	Informal	1K/LoC
EAL7	Formal	Formal	Formal	Informal	
seL4	Formal	Formal	Formal	Formal	0.6K/LoC