Can We Make Trusted Systems Trustworthy?

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!

What's Next?

So, why don't we prove trustworthiness?

***Claim*:**

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

Corollary [with apologies to Dijkstra]:

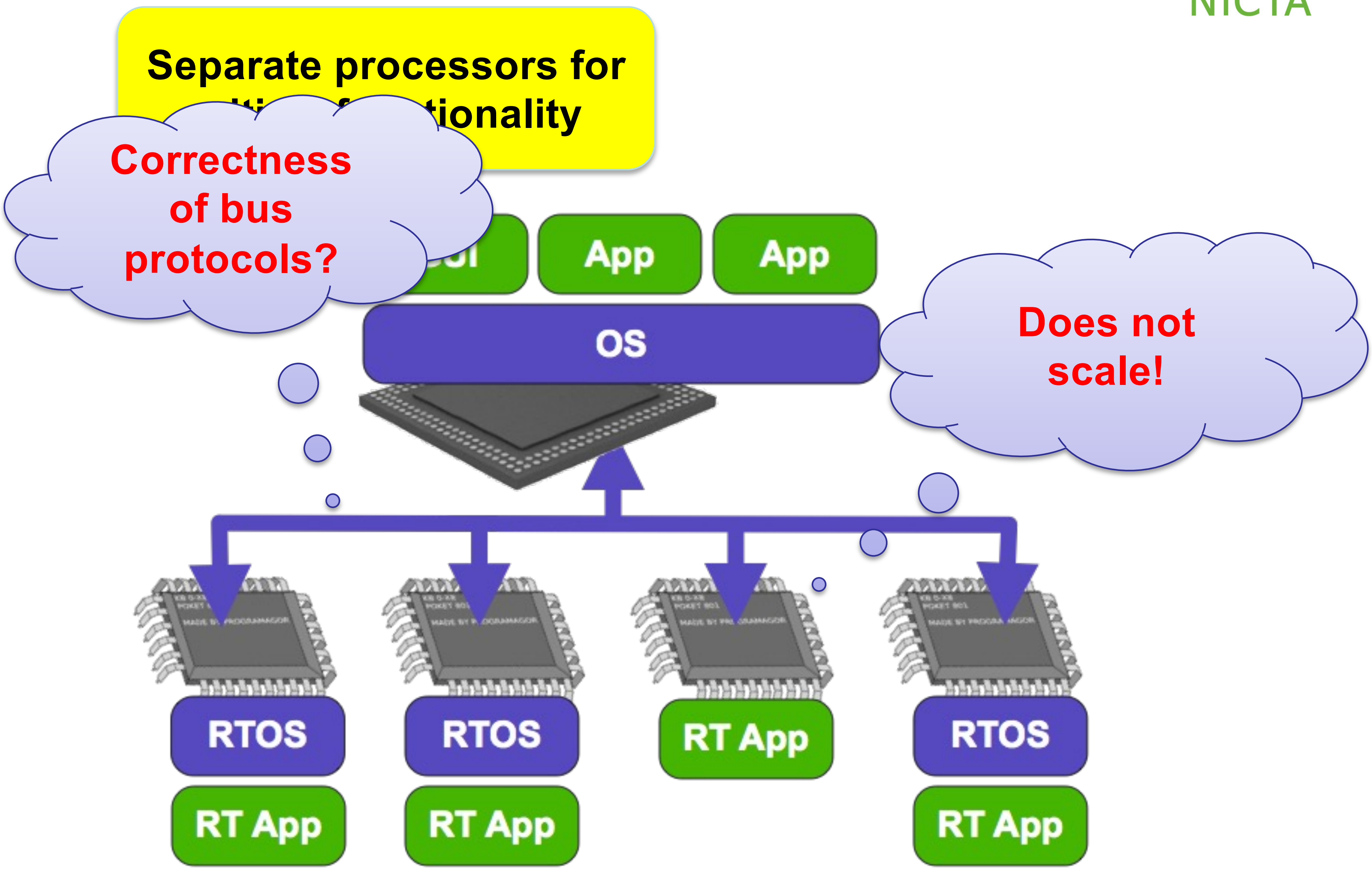
Testing, code inspection, etc. can only show *lack of trustworthiness*!

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…

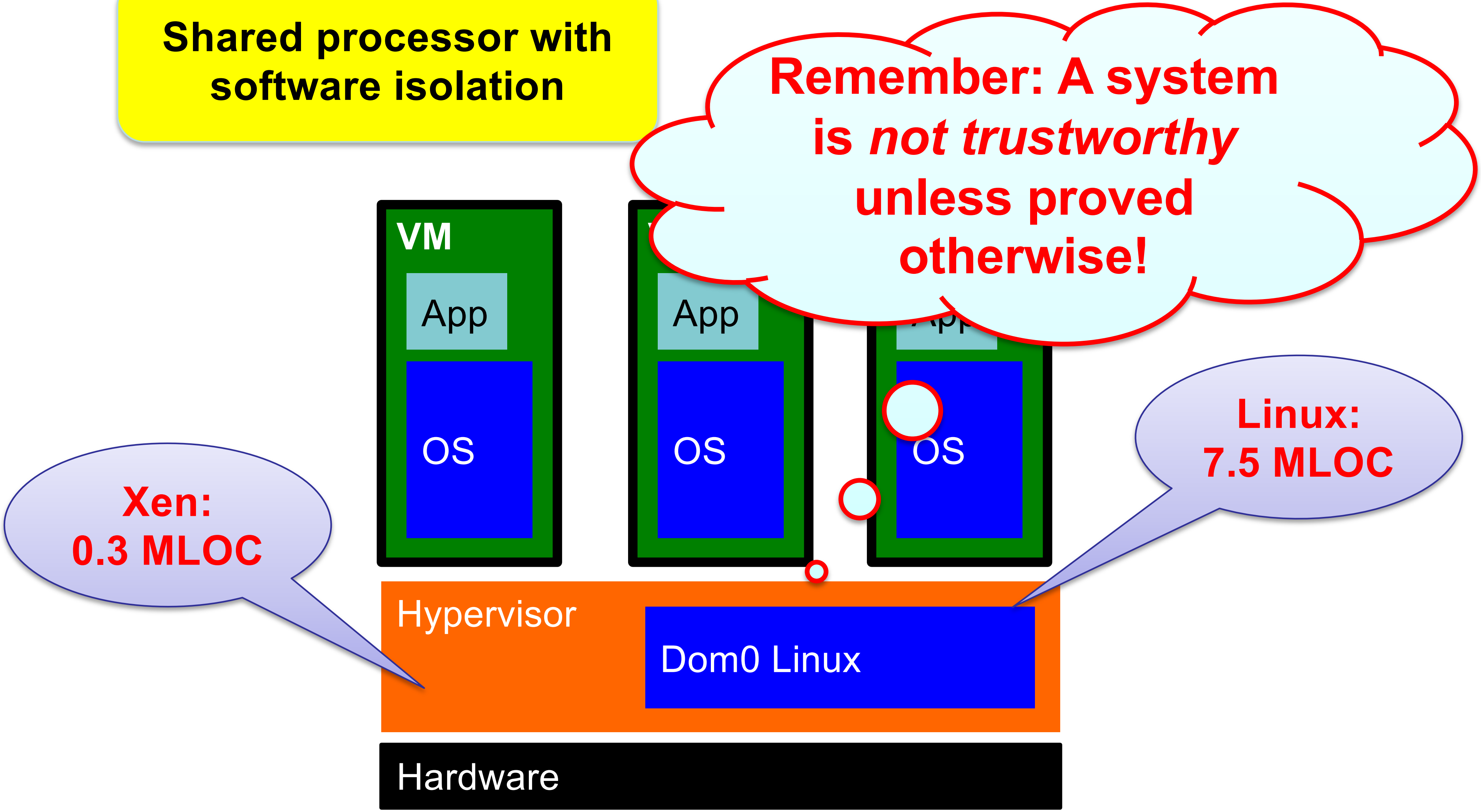
Systems far too complex to prove their trustworthiness!

Dealing with Complexity: Physical Isolation



How About Logical Isolation?

Shared processor with software isolation



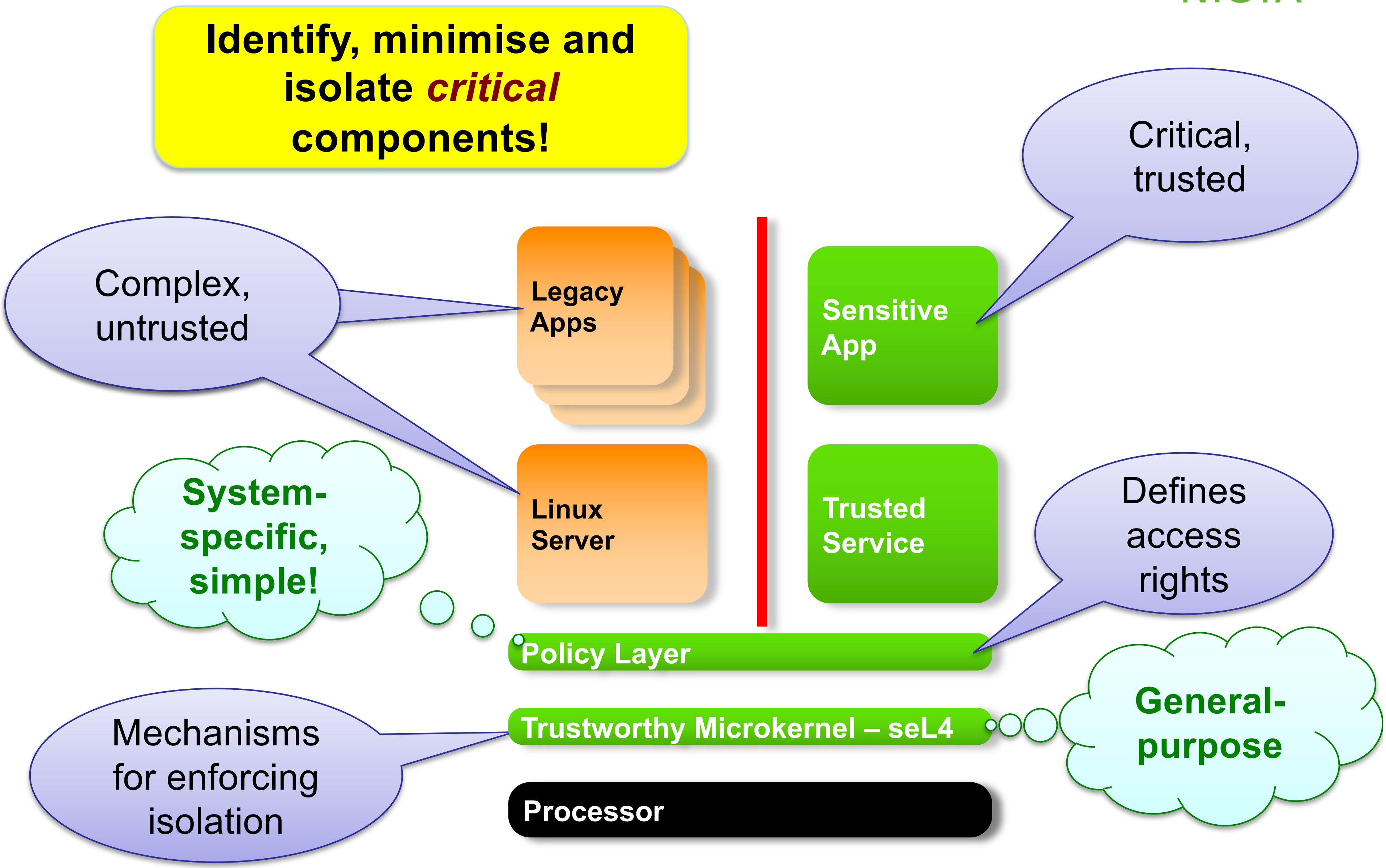
Our Vision: Trustworthy Systems

Suitable for real-world systems

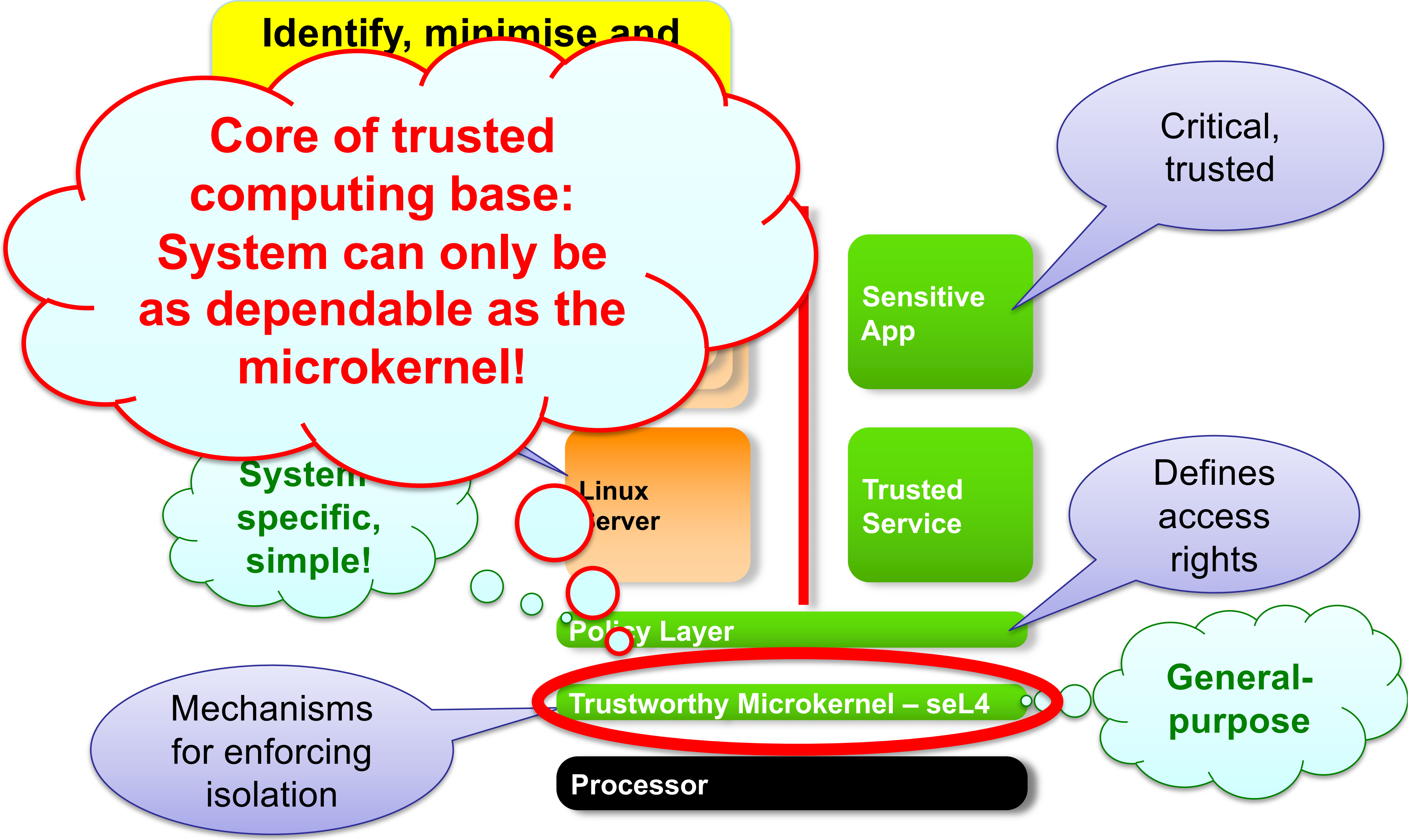
We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

Hard *guarantees* on safety/security/ reliability

Isolation is Key!



Isolation is Key!



NICTA Trustworthy Systems Agenda

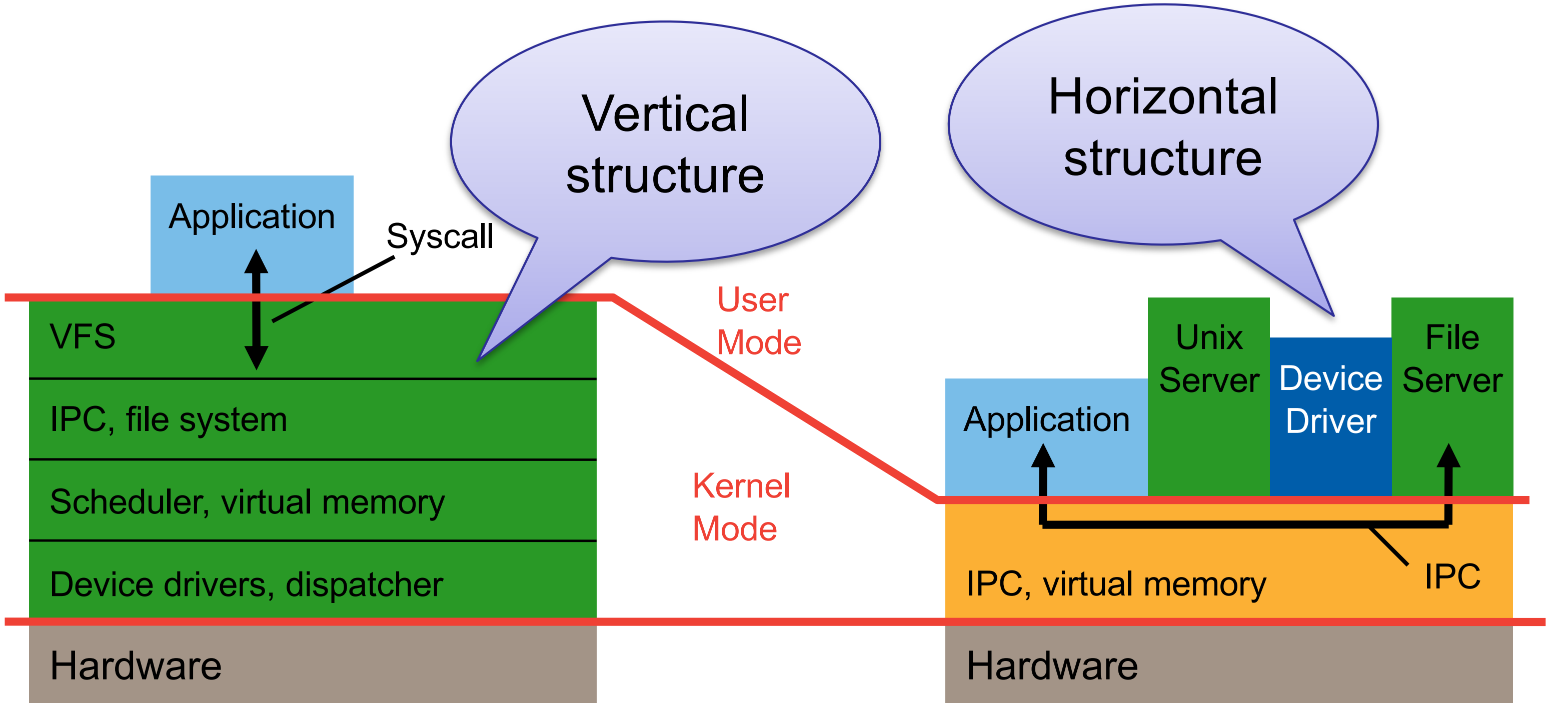
1. **Dependable microkernel (seL4) as a rock-solid base**
   - Formal specification of functionality
   - Proof of functional correctness of implementation
   - Proof of safety/security properties

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!

Agenda

- Motivation
- **Microkernels and seL4 design**
- Establishing trustworthiness
- From kernel to system
- Sample system: Secure access controller

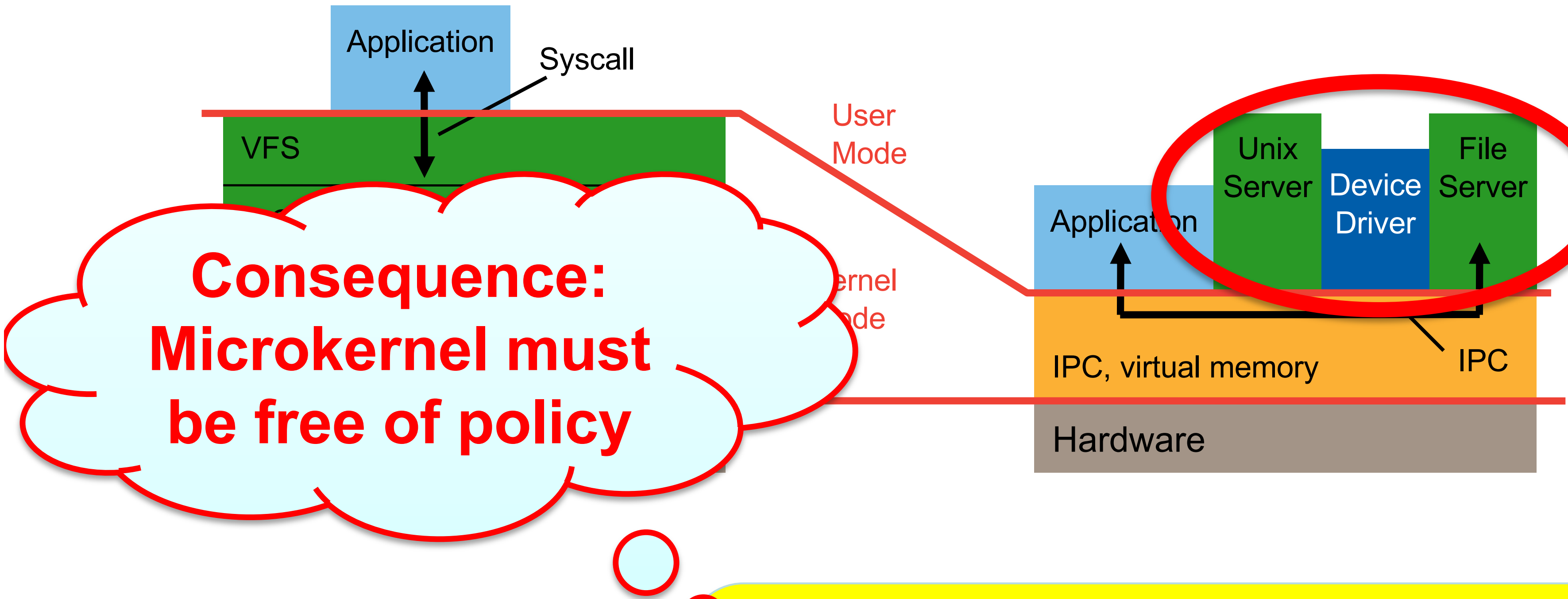
Monolithic Kernels vs Microkernels



Idea of microkernel:

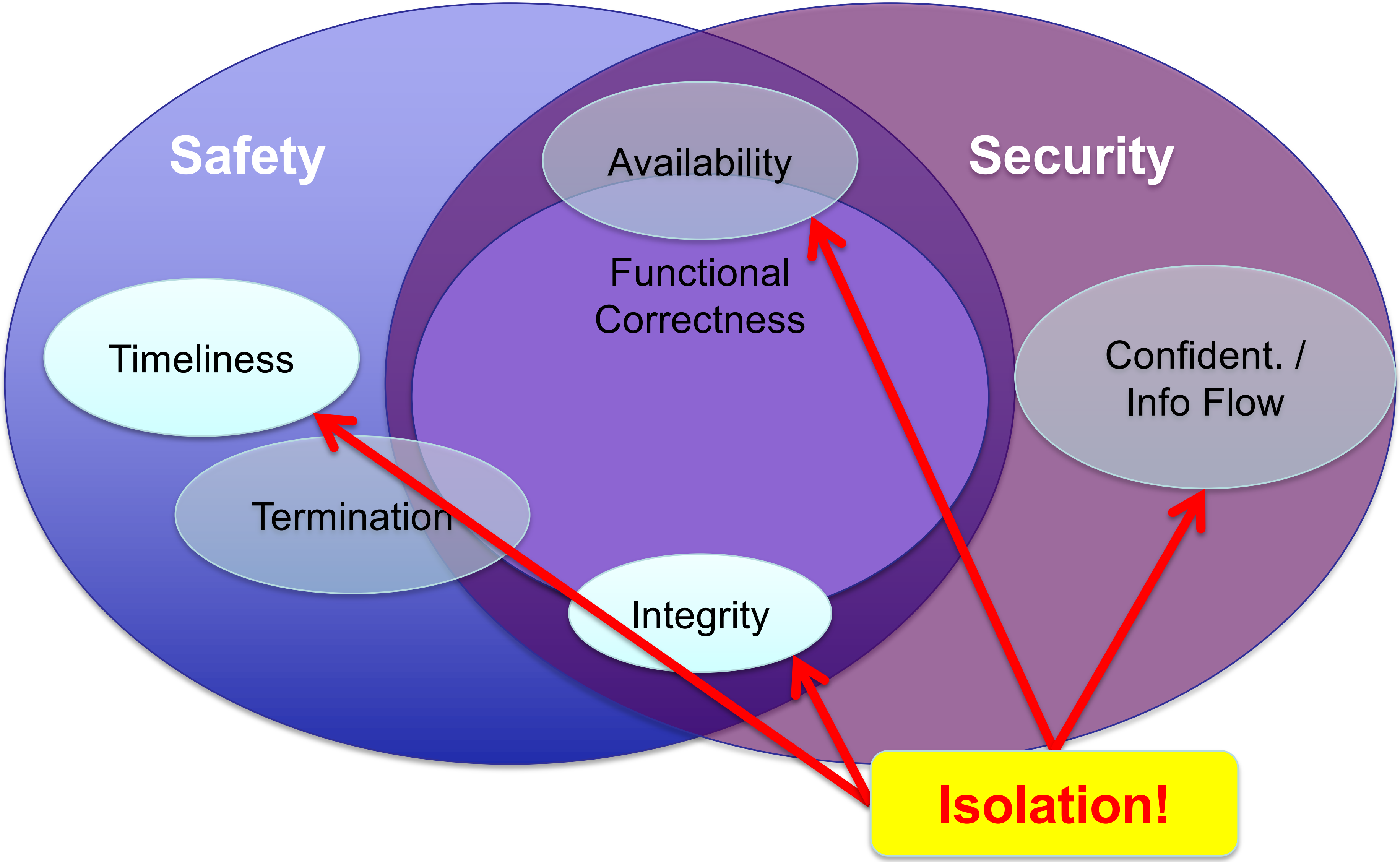
- Flexible, minimal platform, extensible
- Mechanisms, not policies
- Goes back to Nucleus [Brinch Hansen, CACM'70]

Consequence of Minimality: User-level Services

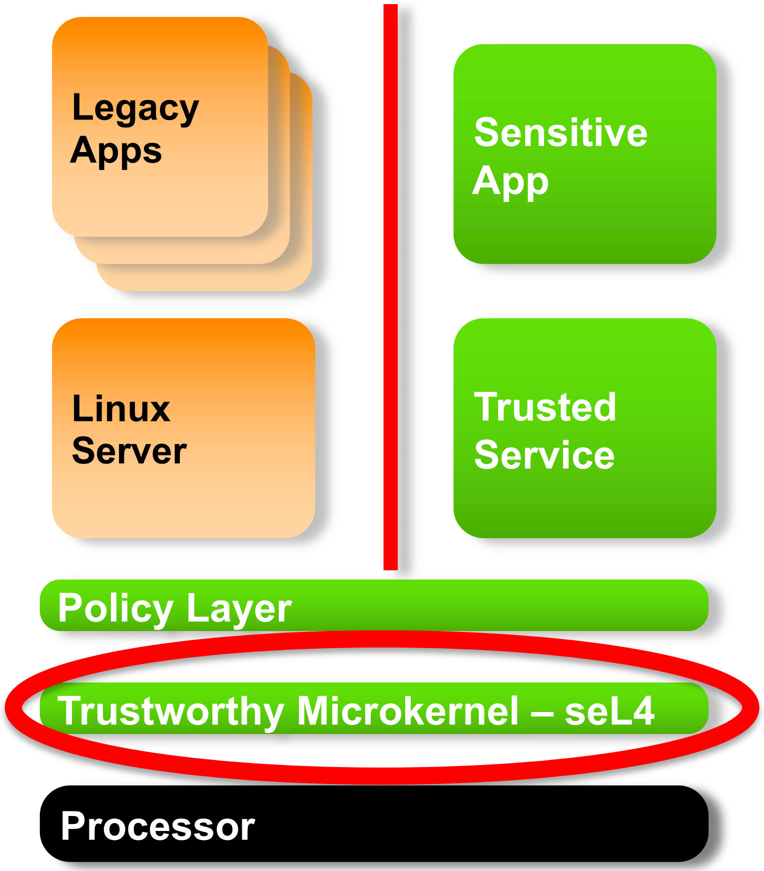


- Kernel provides no services, only mechanisms
- Strongly dependent on fast IPC and exception handling

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

NICTA

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

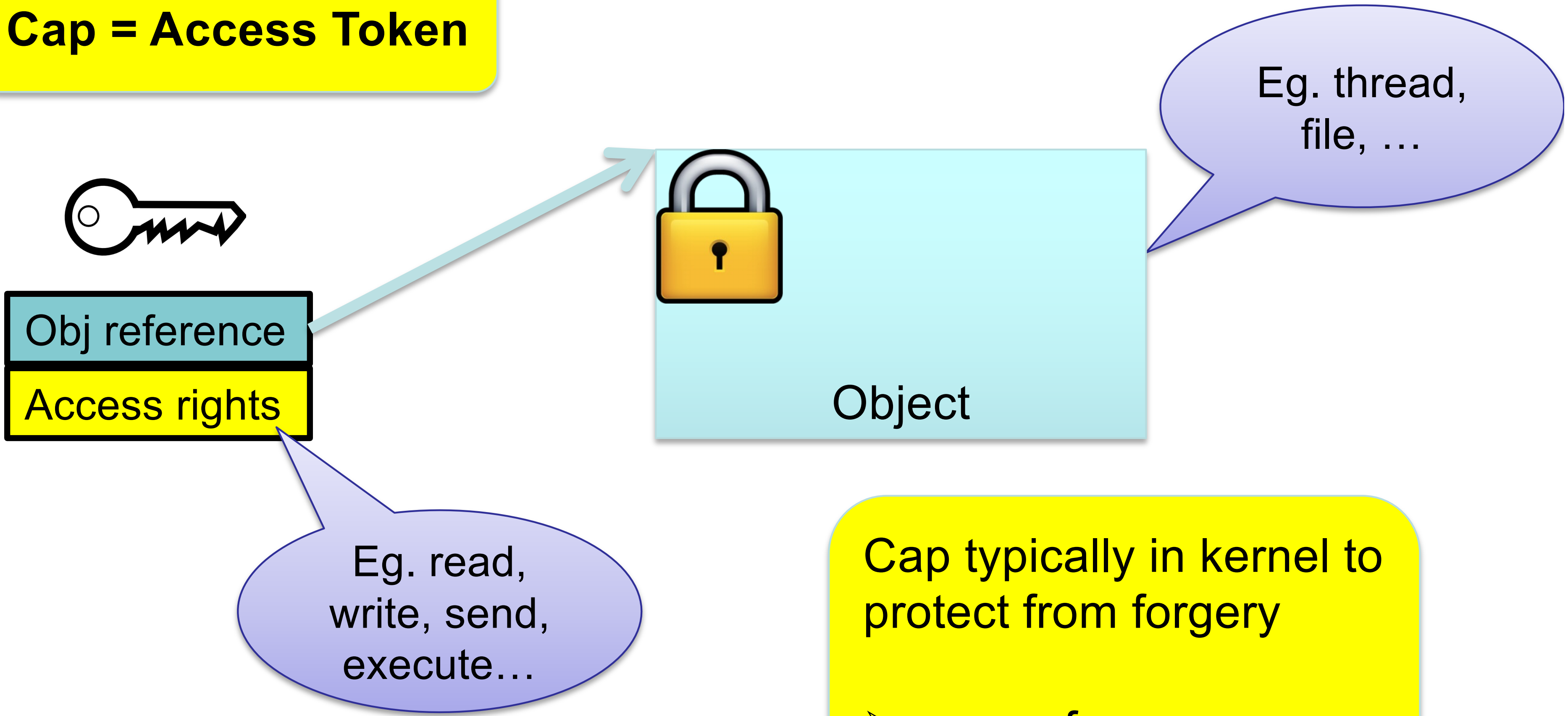
Real-time

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

Verification, Performance

What are Capabilities?

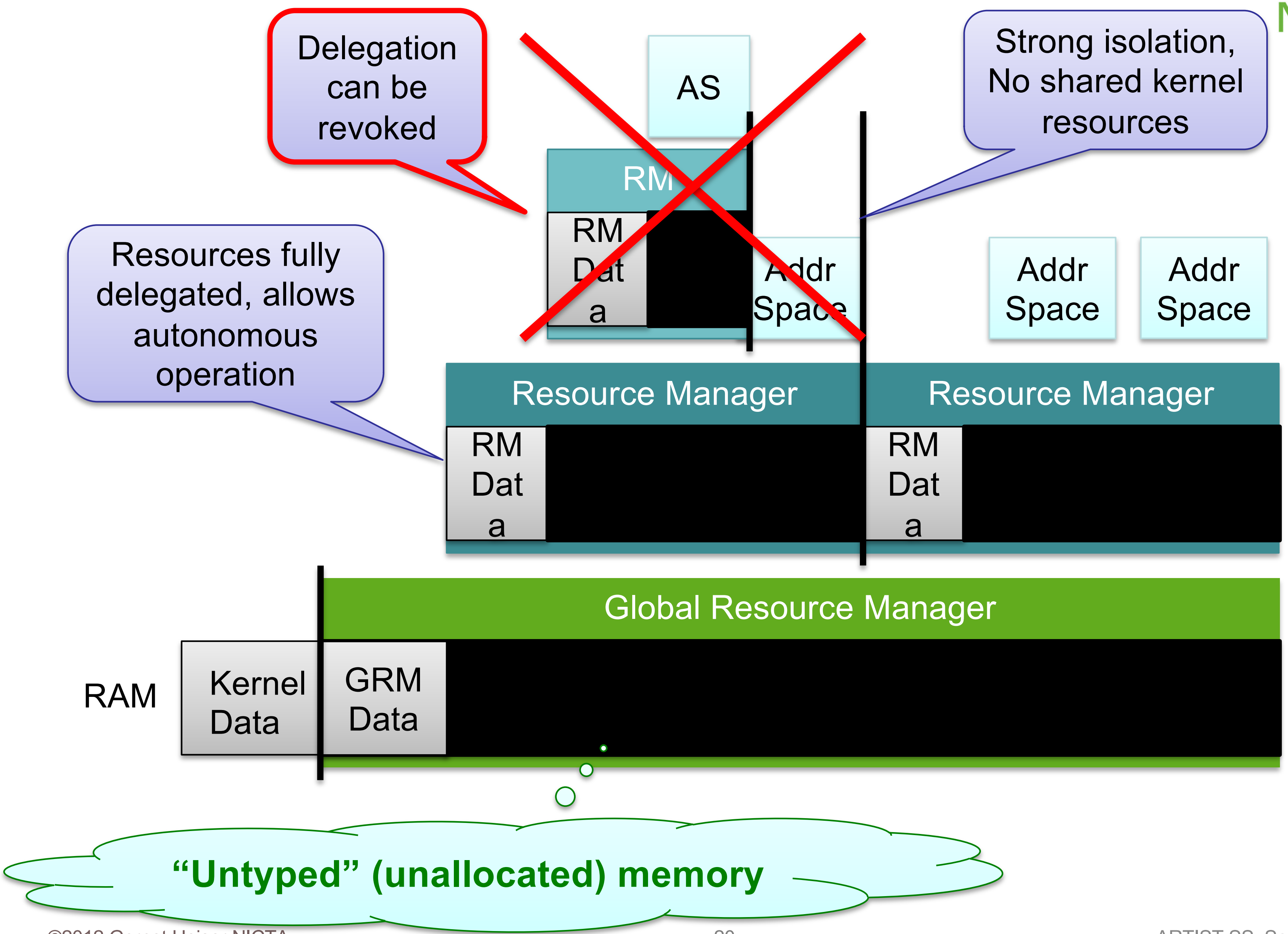
Cap = Access Token



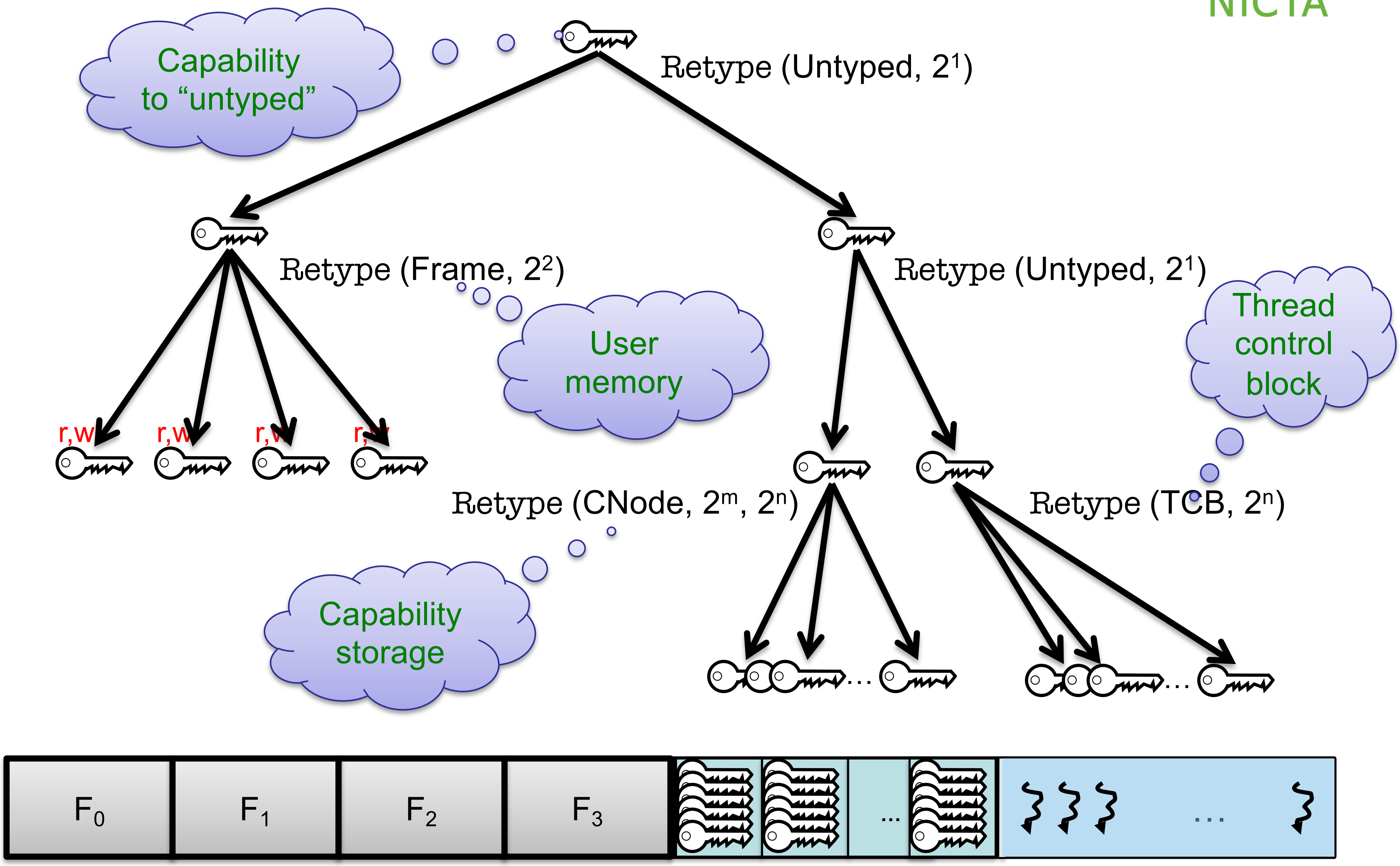
Cap typically in kernel to protect from forgery

- ➢ user references cap through handle

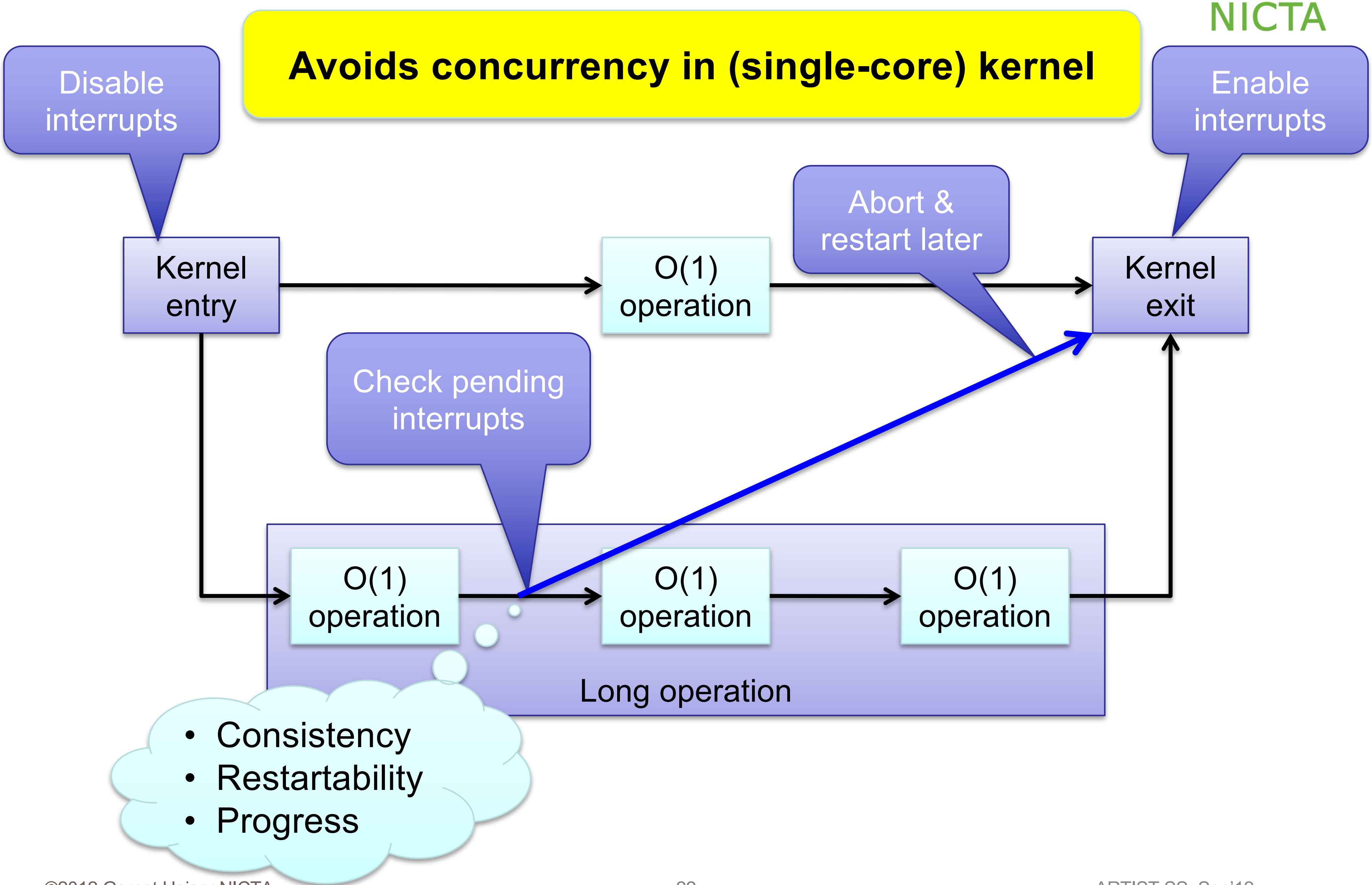
seL4 User-Level Memory Management



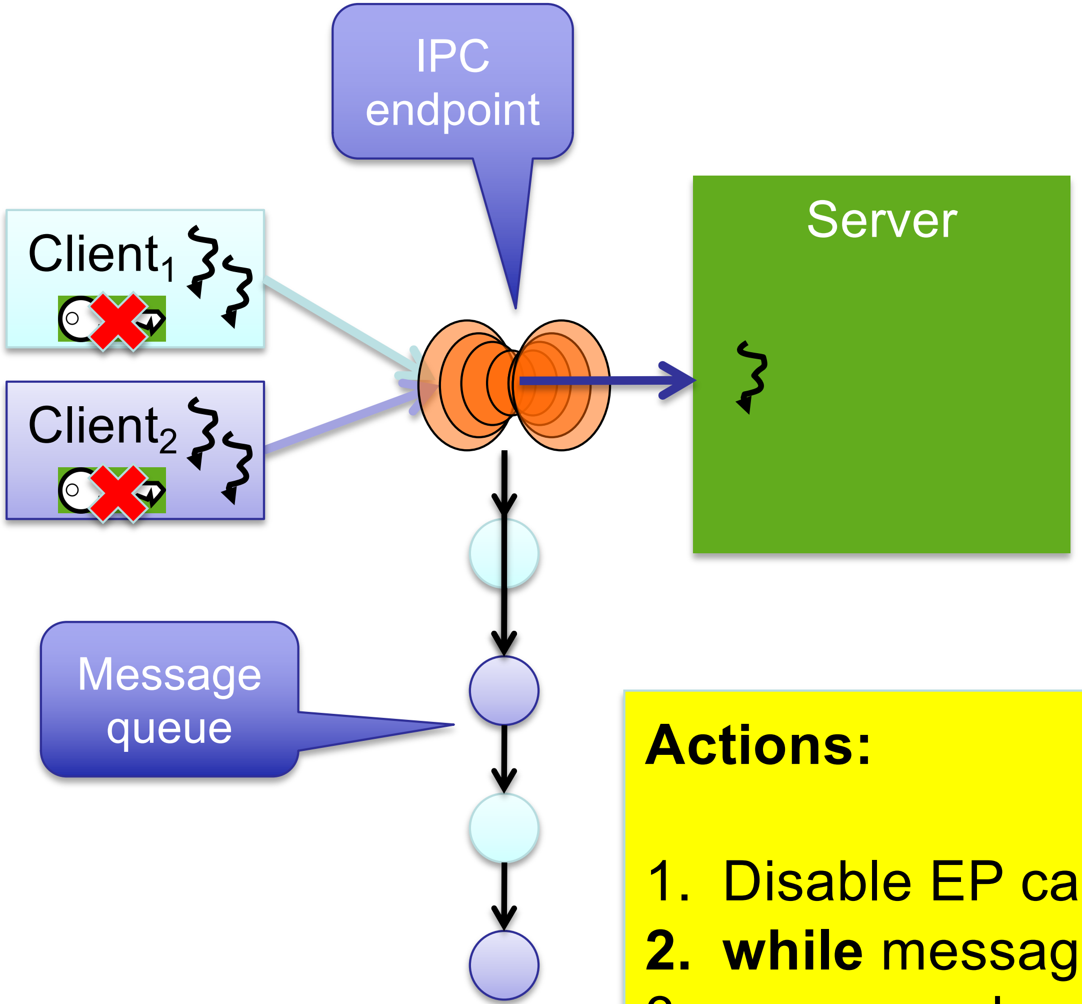
seL4 Memory Management Mechanics: Retype



Incremental Consistency



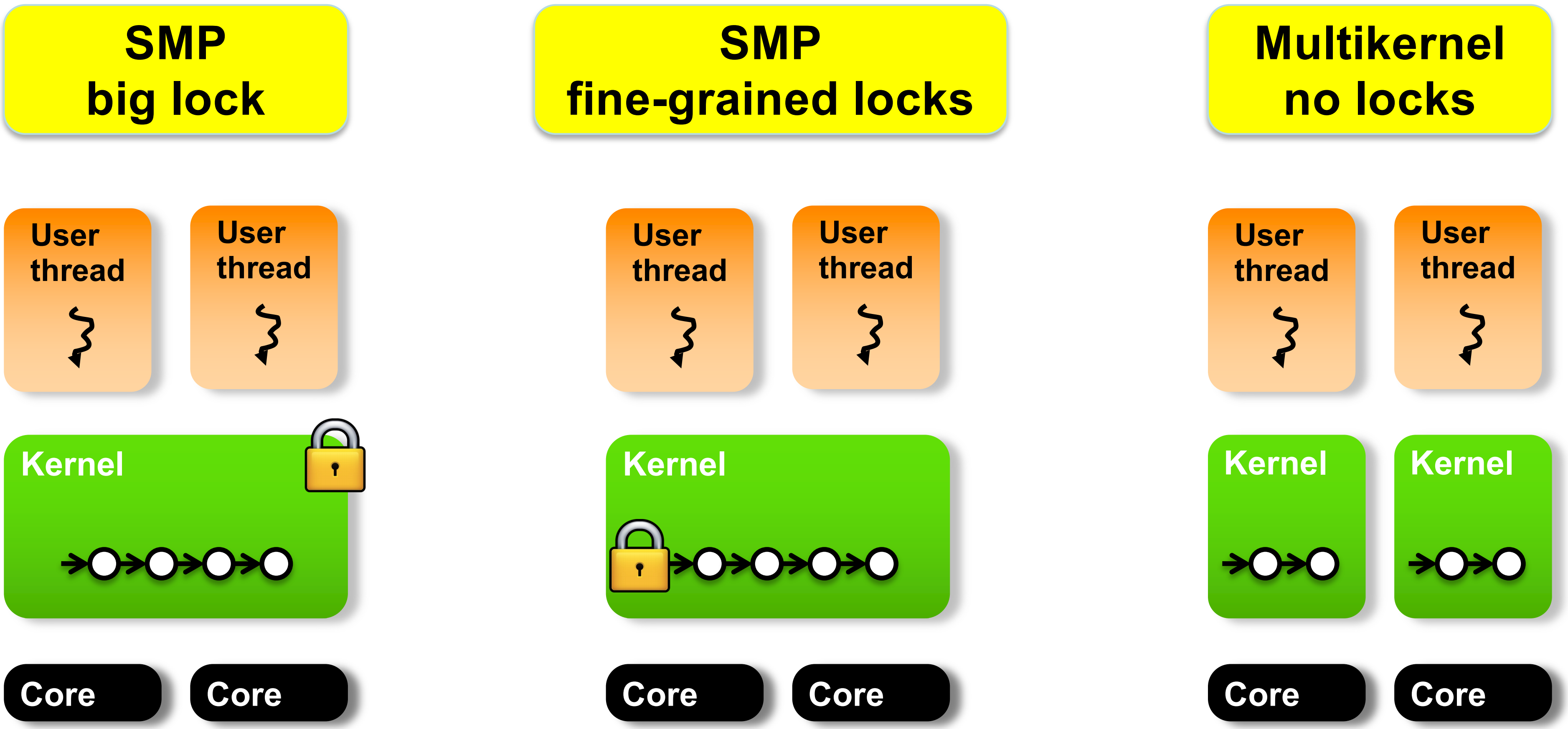
Example: Destroying IPC Endpoint



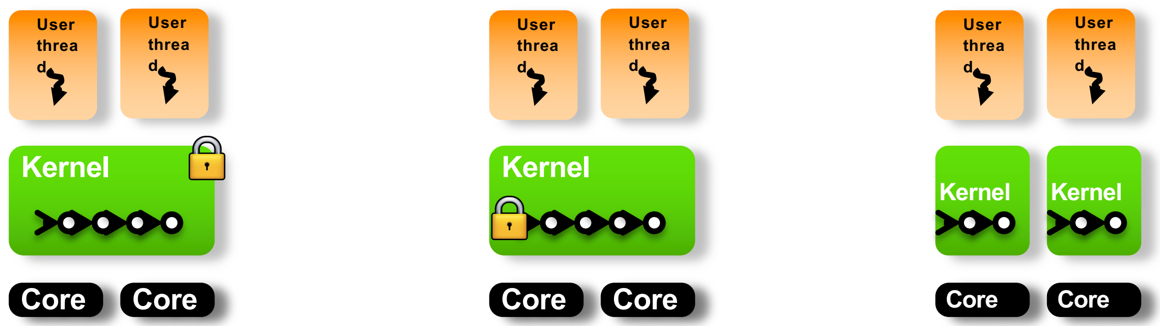
Actions:

1. Disable EP cap (prevent new messages)
2. while message queue not empty **do**
3. remove head of queue (abort message)
4. check for pending interrupts
5. done

Approaches for Multicore Kernels

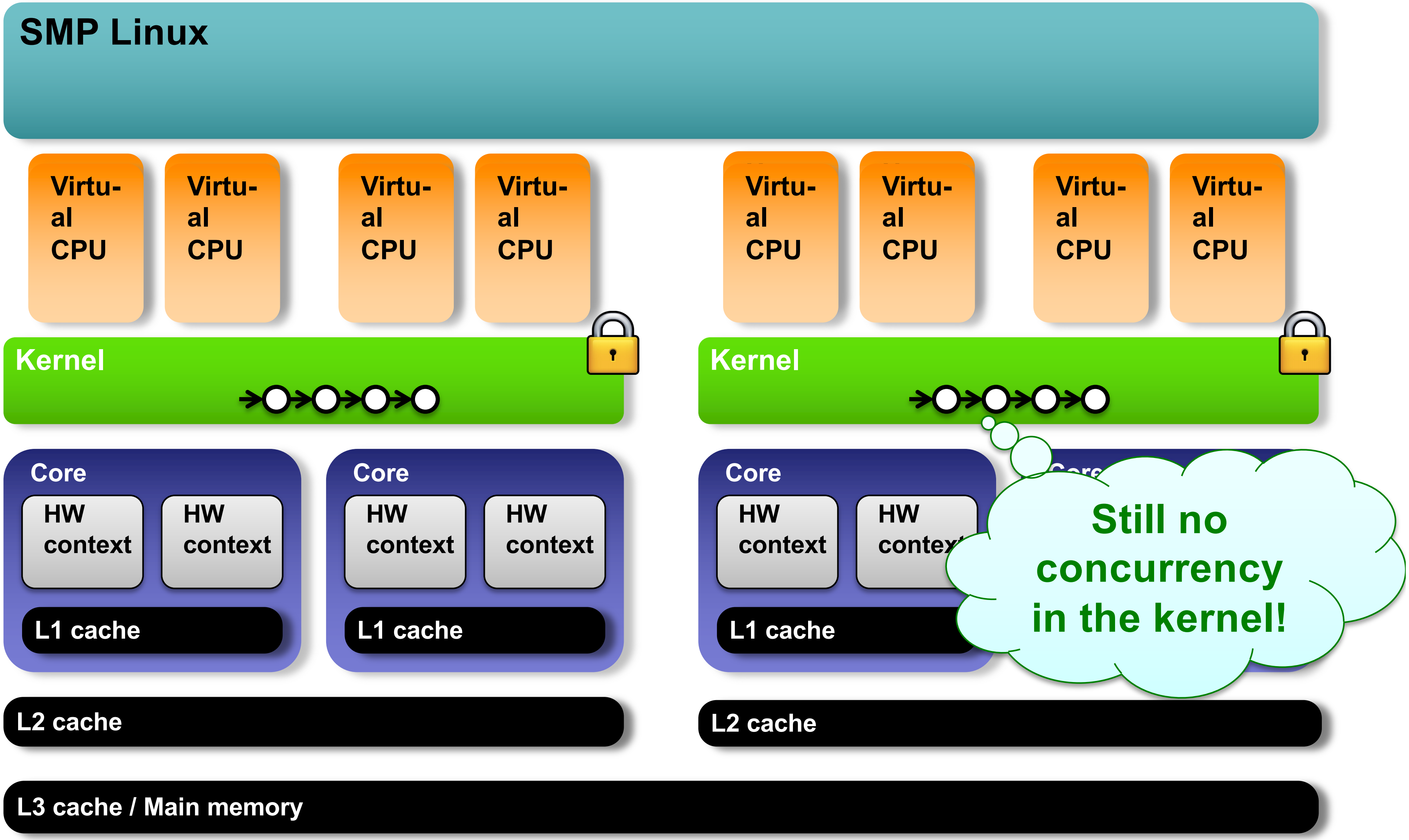


Multicore Kernel Trade-Offs

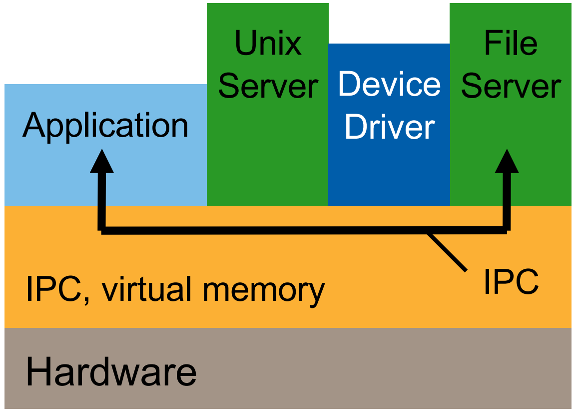


Property	Big Lock	Fine-grained Locking	Multikernel
Data structures	shared	shared	distributed
Scalability	poor	good	excellent
Concurrency in kernel	zero	high	zero
Kernel complexity	low	high	low
Resource management	centralised	centralised	distributed

seL4 Multicore Design: Clustered Multikernel



How About Performance?



Let's face it, seL4 is basically slow!

- C code (semi-blindly) translated from Haskell
- Many small functions, little regard for performance

Bare "pass" in Advanced Operating Systems course!

IPC: one-way, zero-length	
Standard C code:	1455 cycles
C fast path:	185 cycles

Fastest-ever IPC on ARM11!

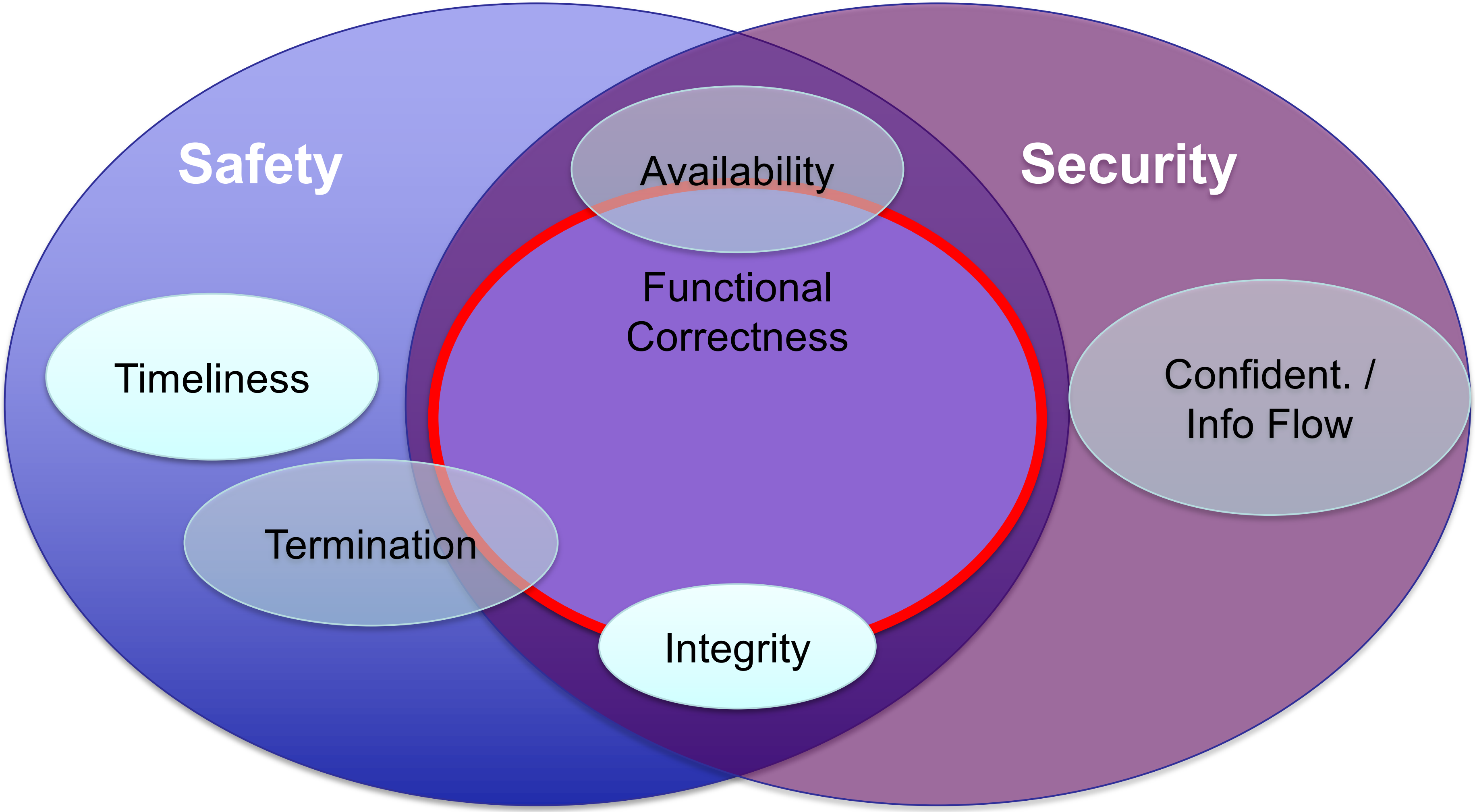
But can speed up critical operations by short-circuit "fast paths"

- … without resorting to assembler!

Agenda

- Motivation
- Microkernels and seL4 design
- **Establishing trustworthiness**
- From kernel to system
- Sample system: Secure access controller

seL4 as Basis for Trustworthy Systems



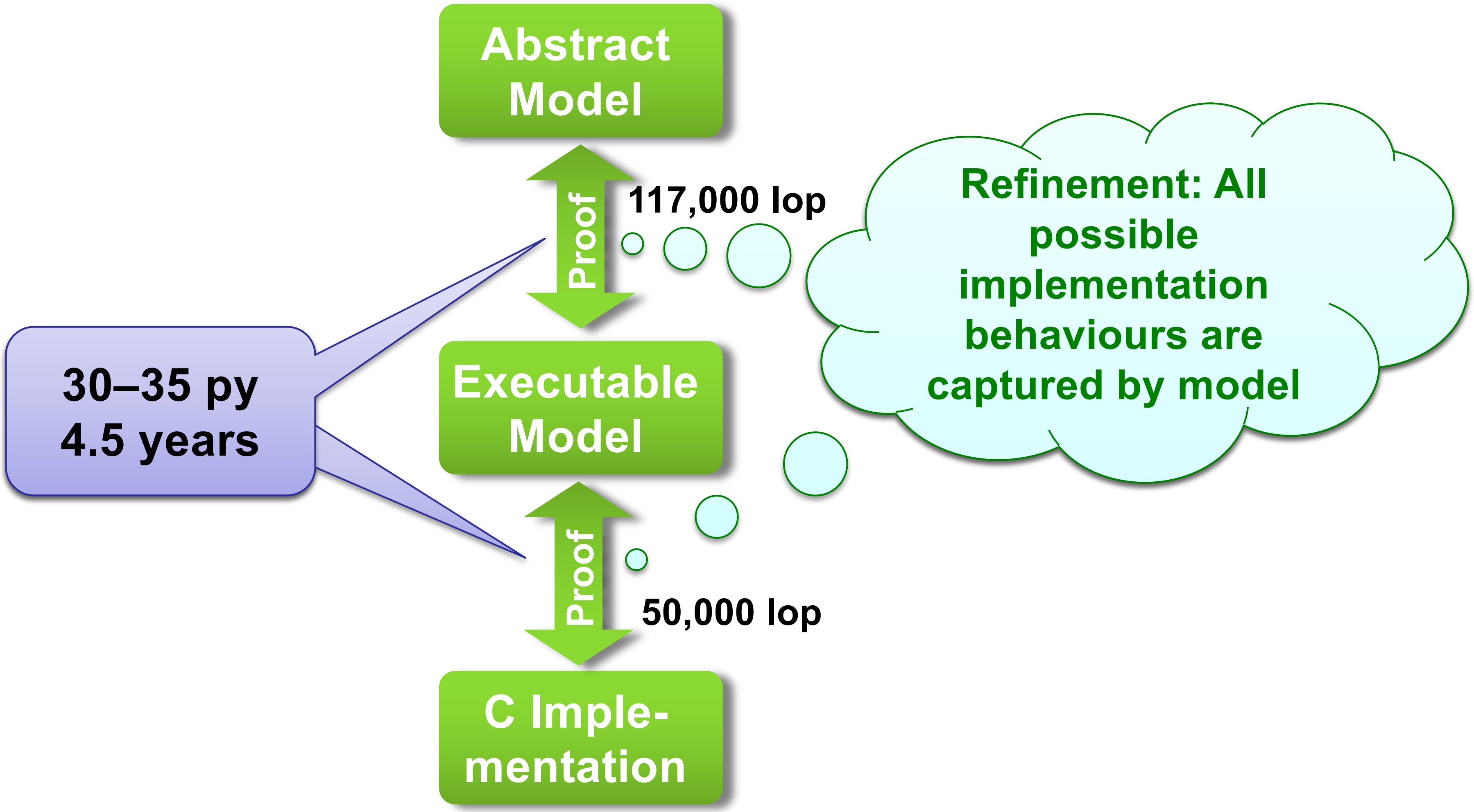
Proving Functional Correctness

```
constdefs
  schedule :: "unit s_monad"
  "schedule ≡ do
     threads ← allActiveTCBs;
     thread ← select threads;
     do_machine_op flushCaches OR return ();
     modify (λs. s ⦇ cur_thread := thread ⦈)
   od"
```

```
schedule :: Kernel ()
schedule = do
        action <- getSchedulerAction
ad
curThread
meSlice curThread
ime == 0) chooseThread
```

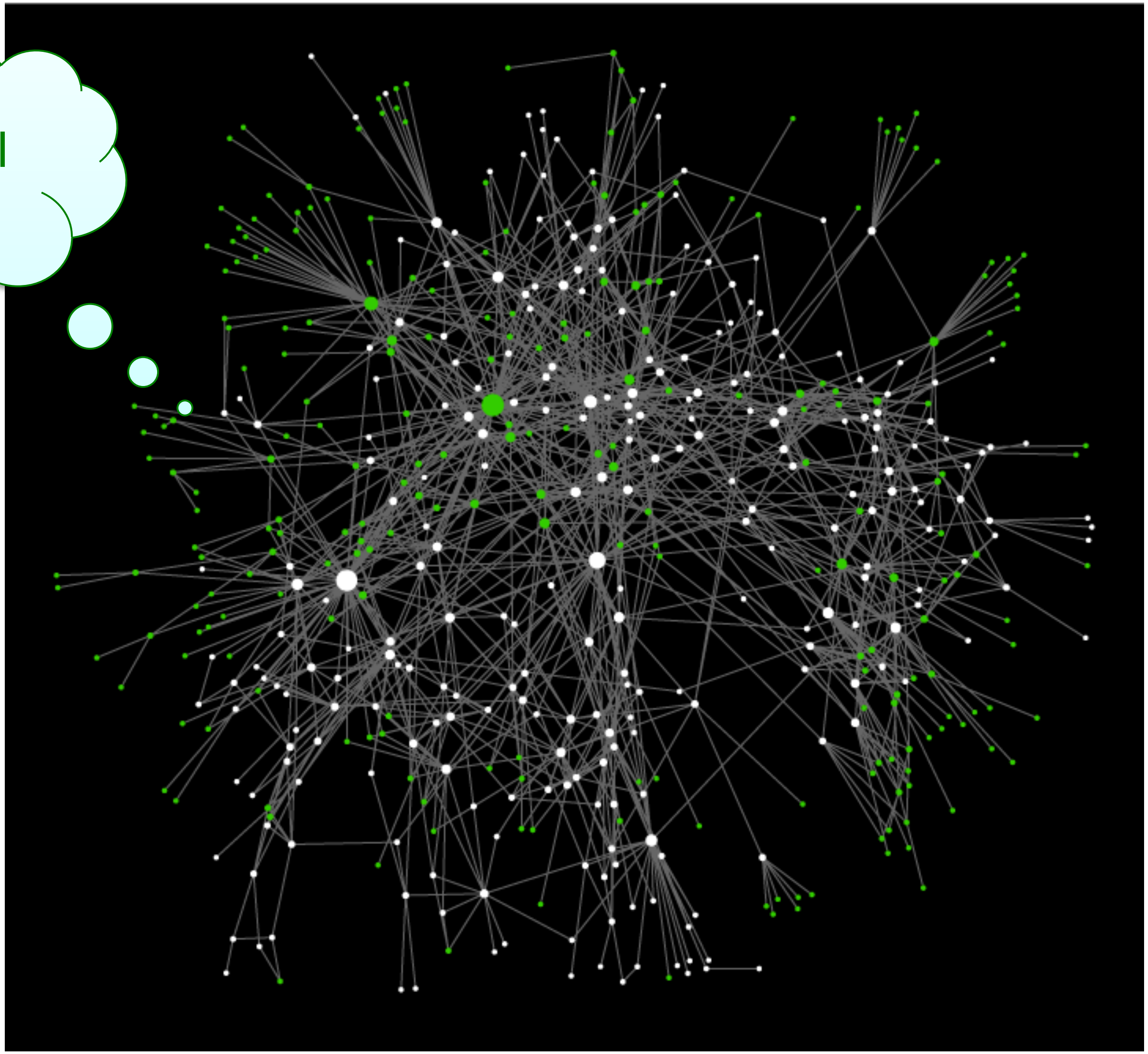
```
void
setPriority(tcb_t *tptr, prio_t prio) {
    prio_t oldprio;

    if(thread_state_get_tcbQueued(tptr->tcbState)) {
        oldprio = tptr->tcbPriority;
        ksReadyQueues[oldprio] = tcbSchedDequeue(tptr, ksReadyQueues[
        if(isRunnable(tptr)) {
            ksReadyQueues[prio] = tcbSchedEnqueue(tptr, ksReadyQueues
        }
        else {
            thread_state_ptr_set_tcbQueued(&tptr->tcbState, false);
        }
    }

    tptr->tcbPriority = prio;
}

void
yieldTo(tcb_t *target) {
    target->tcbTimeSlice += ksCurThread->tcbTimeSlice;
```

Proving Functional Correctness



Why So Long for 9,000 LOC?

seL4 call graph



Costs Breakdown

NICTA

Haskell design	2 py
C implementation	2 weeks
Debugging/Testing	2 months
Kernel verification	12 py
Formal frameworks	10 py
Total	**25 py**
Repeat (estimated)	6 py
Traditional engineering	4–6 py

Did you find bugs???

- During (very shallow) testing: 16
- During verification: 460
  - 160 in C, ~150 in design, ~150 in spec

Does not include subsequent fastpath verification

seL4 Formal Verification Summary

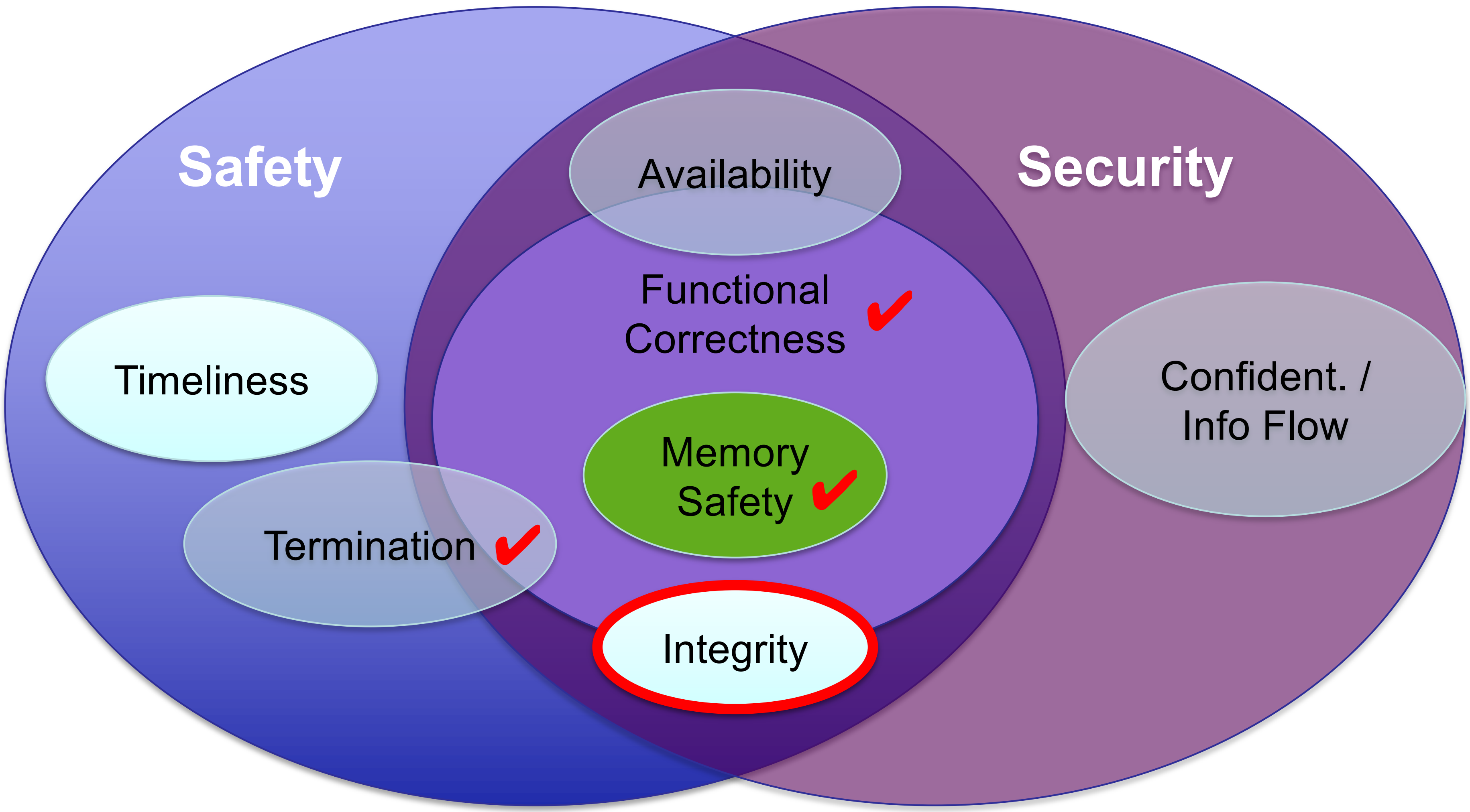
NICTA

Kinds of properties proved

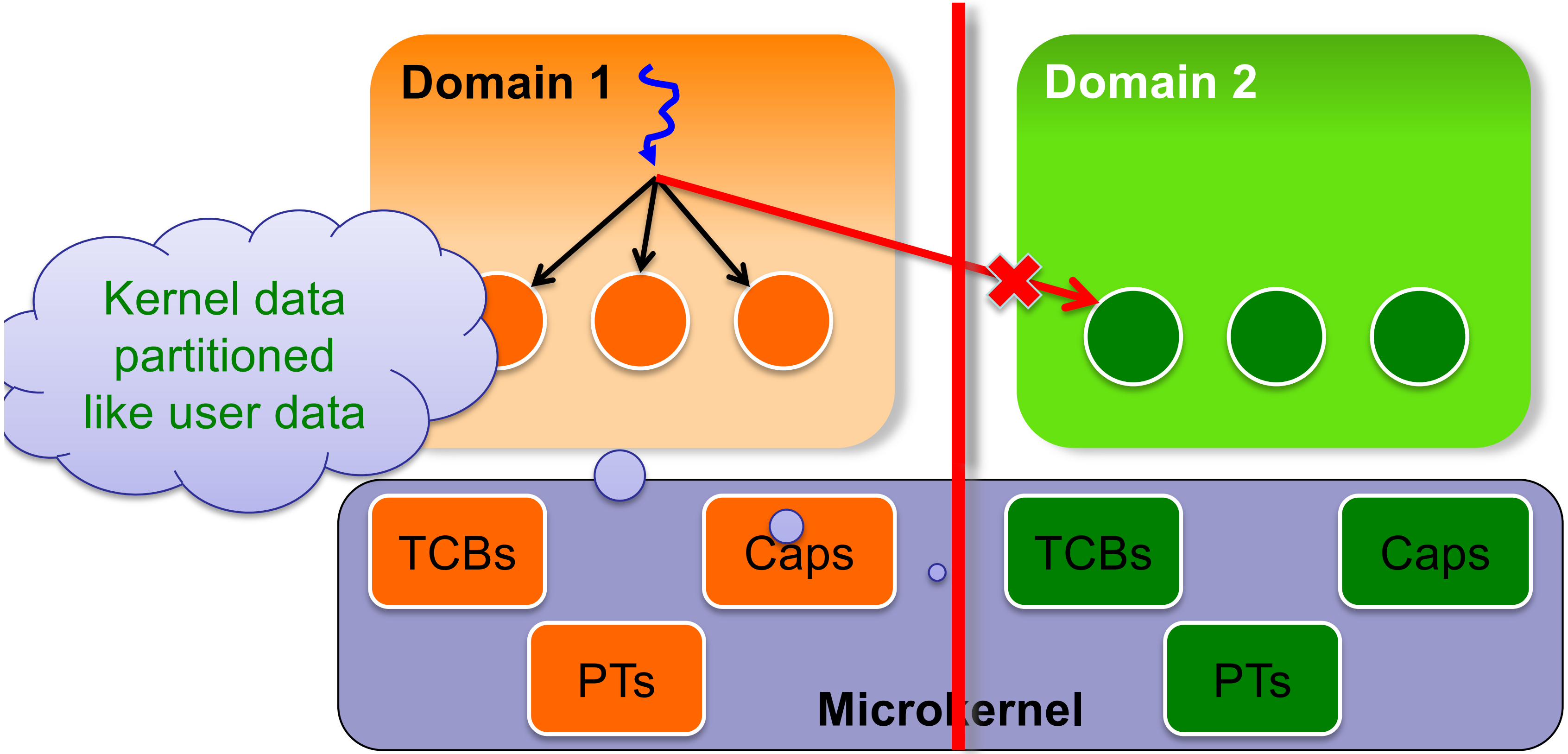
- Behaviour of C code is fully captured by abstract model
- Behaviour of C code is fully captured by executable model
- Kernel never fails, behaviour is always well-defined
  - assertions never fail
  - will never de-reference null pointer
  - cannot be subverted by misformed input
- All syscalls terminate, reclaiming memory is safe, ...
- Well typed references, aligned objects, kernel always mapped…
- Access control is decidable

Can prove further poperties on abstract level!

seL4 as Basis for Trustworthy Systems



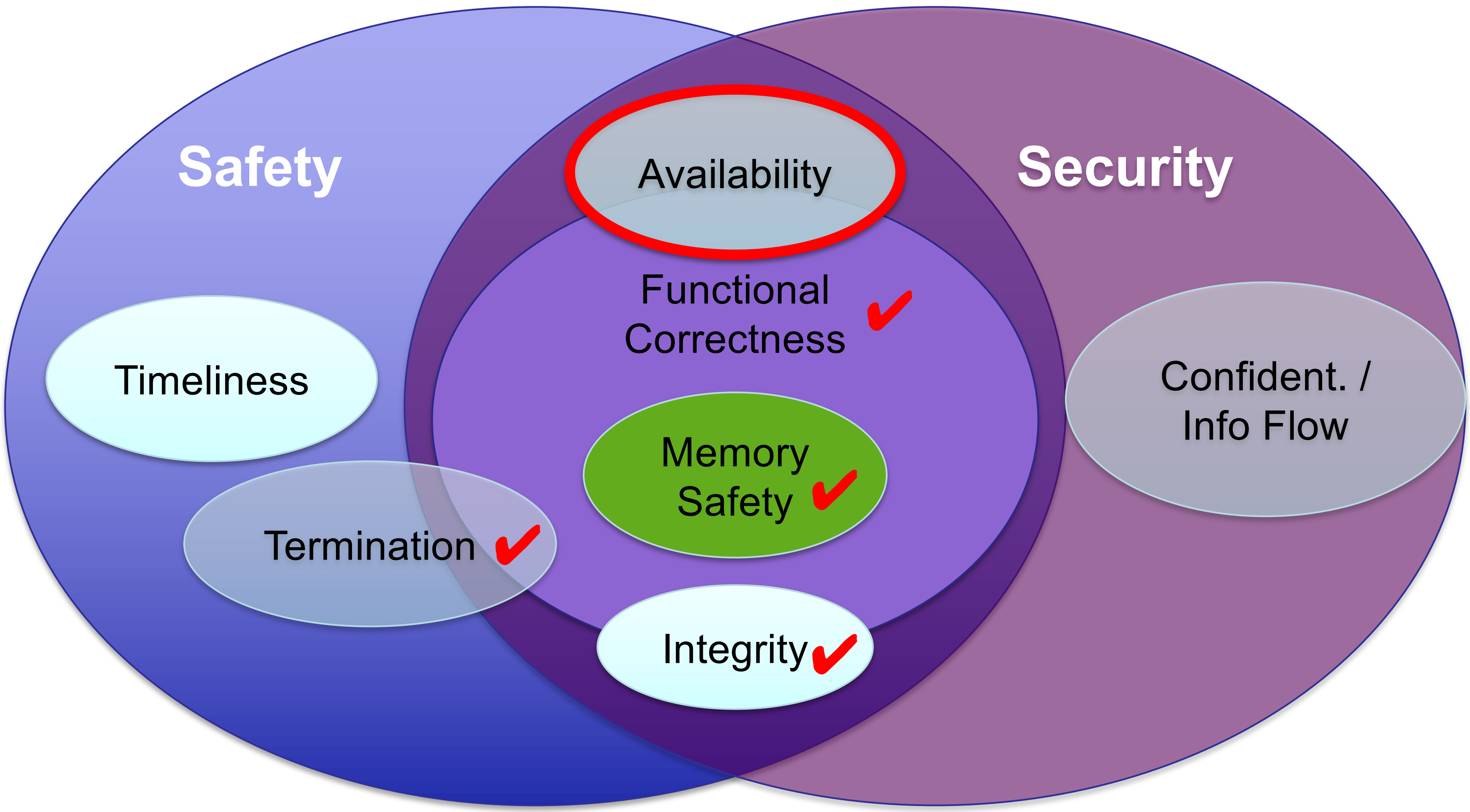
Integrity: Limiting Write Access



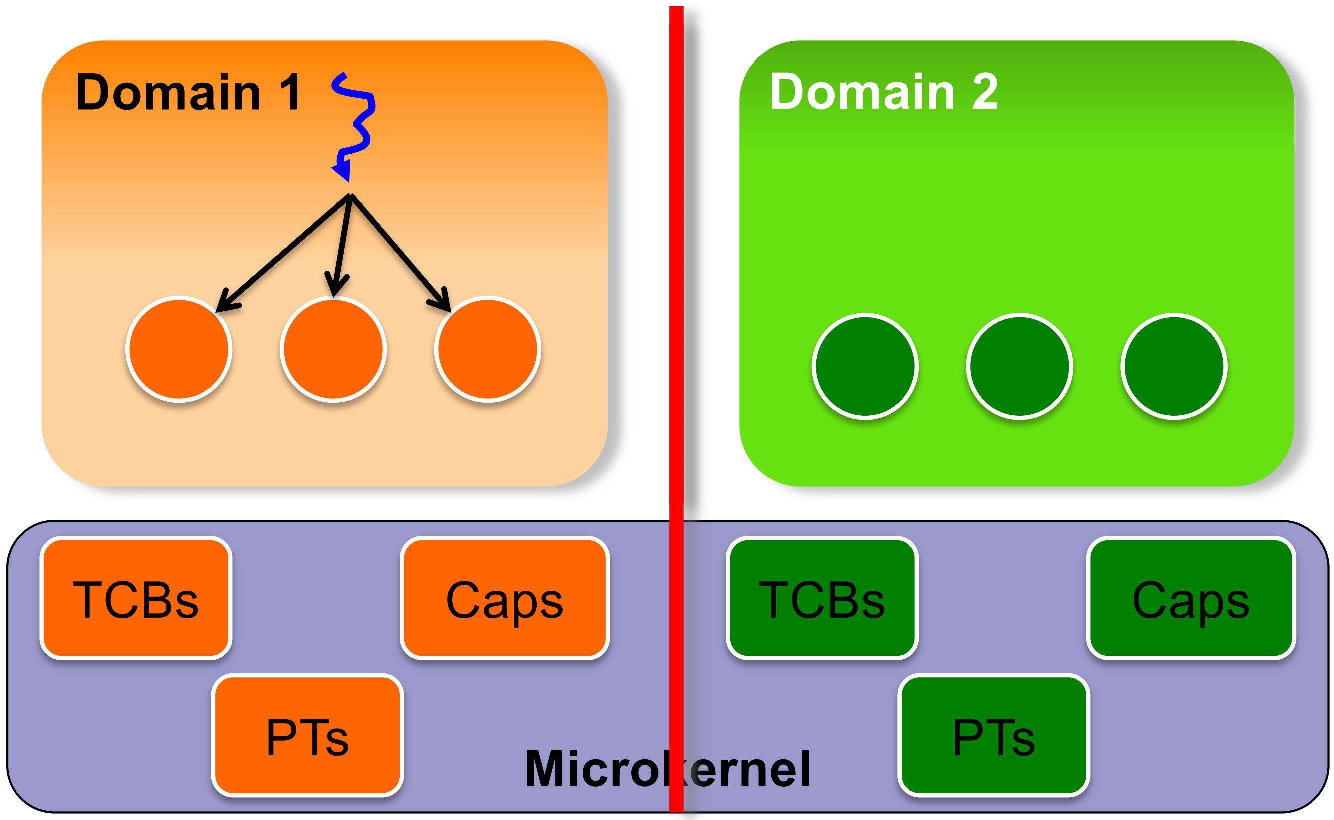
To prove:

- 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!*
  - Event-based kernel operates on behalf of well-defined user thread
  - Prove kernel only allows write upon capability presentation

seL4 as Basis for Trustworthy Systems

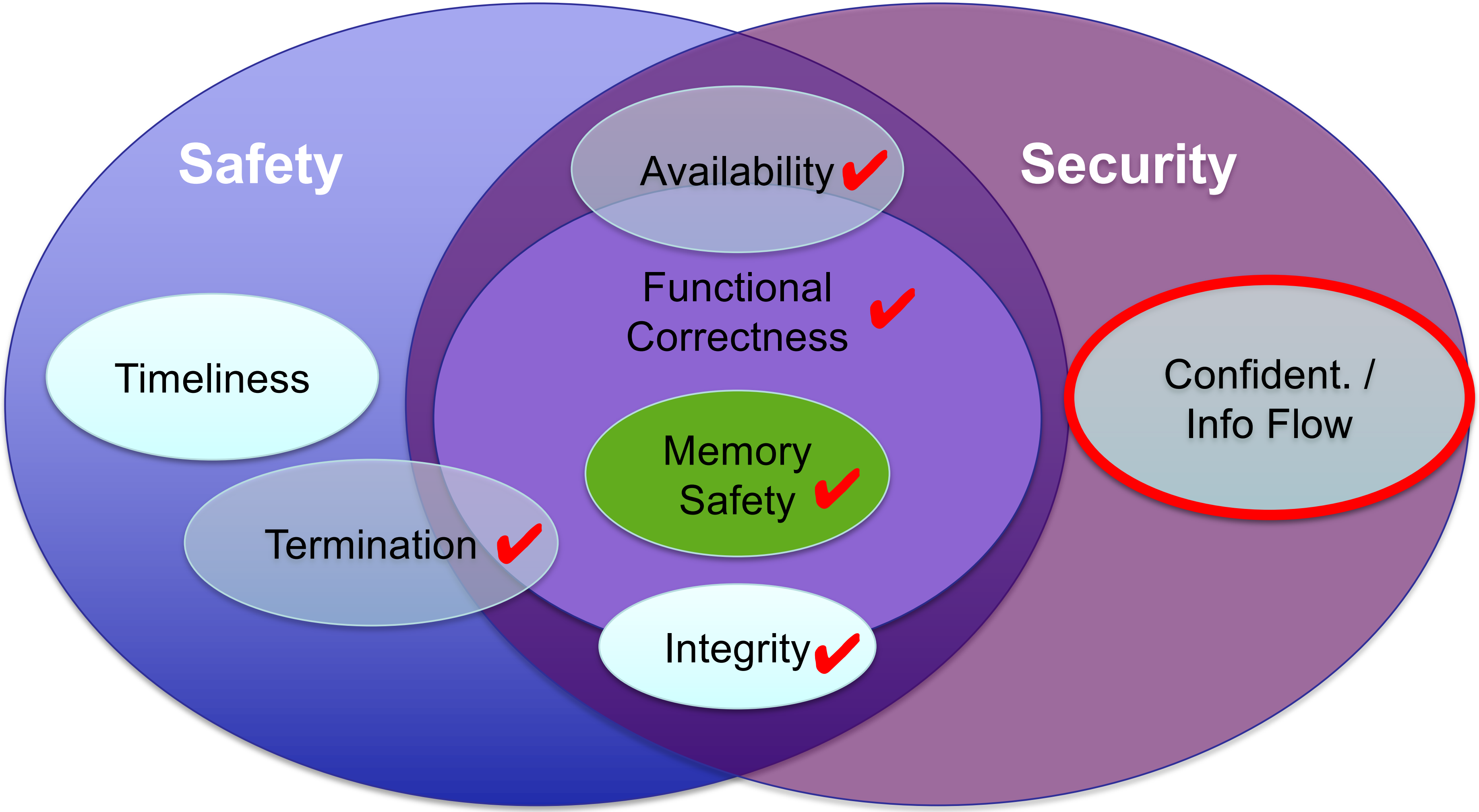


Availability: Ensuring Resource Access

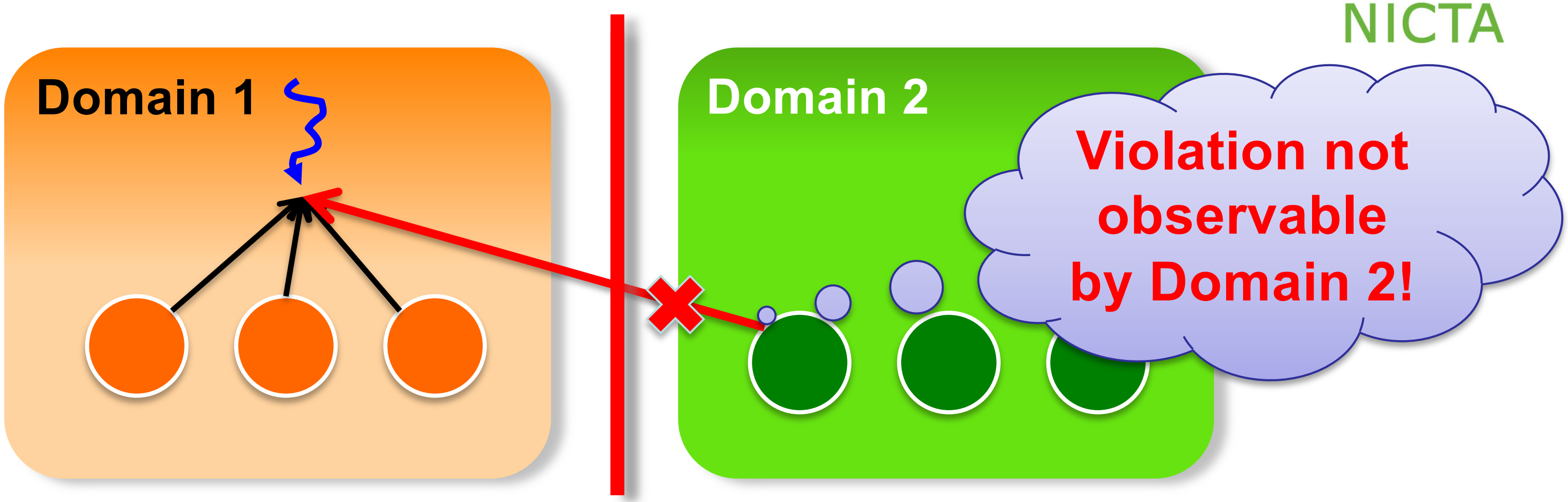


- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources

seL4 as Basis for Trustworthy Systems



Confidentiality: Limiting Read Accesses



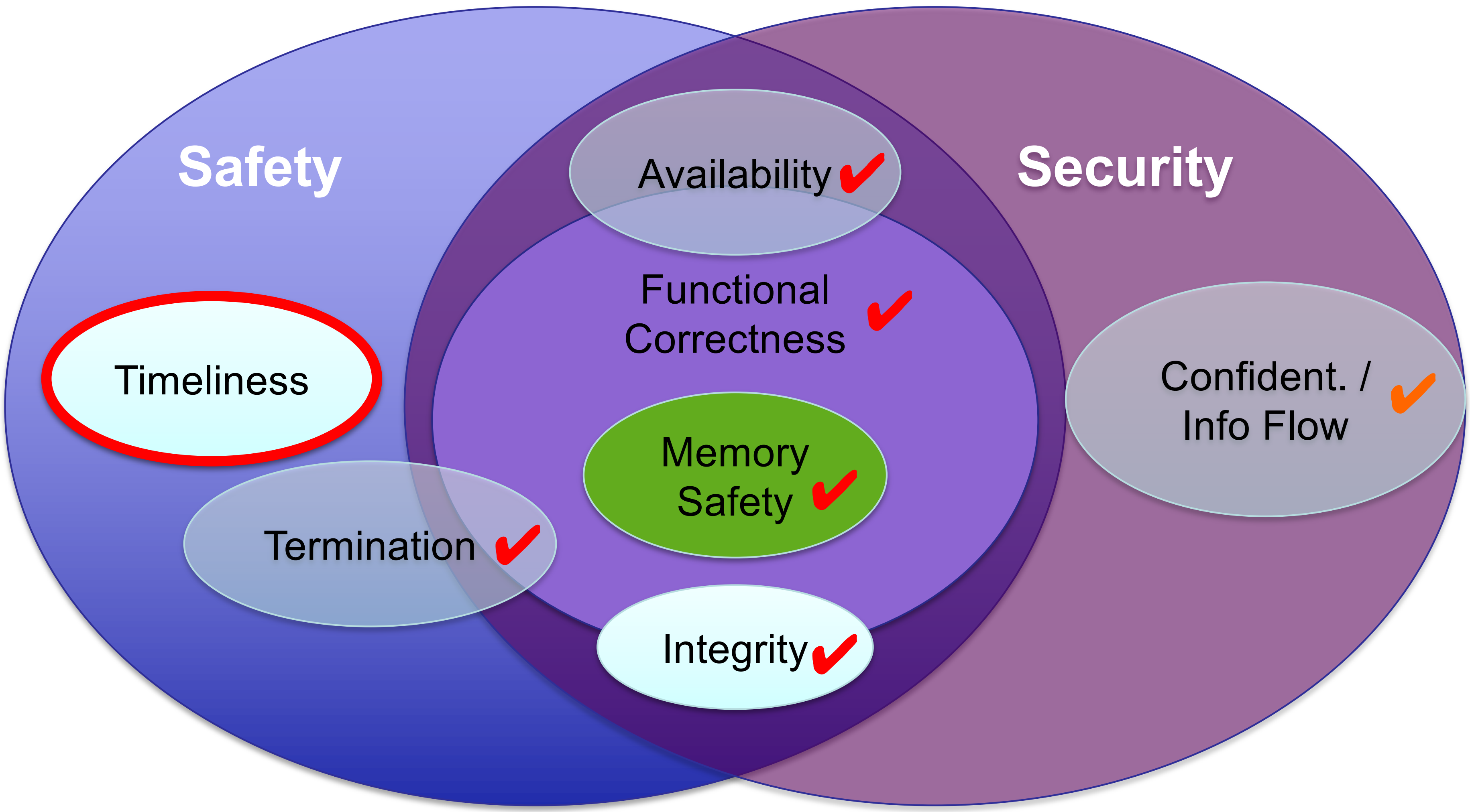
To prove:

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

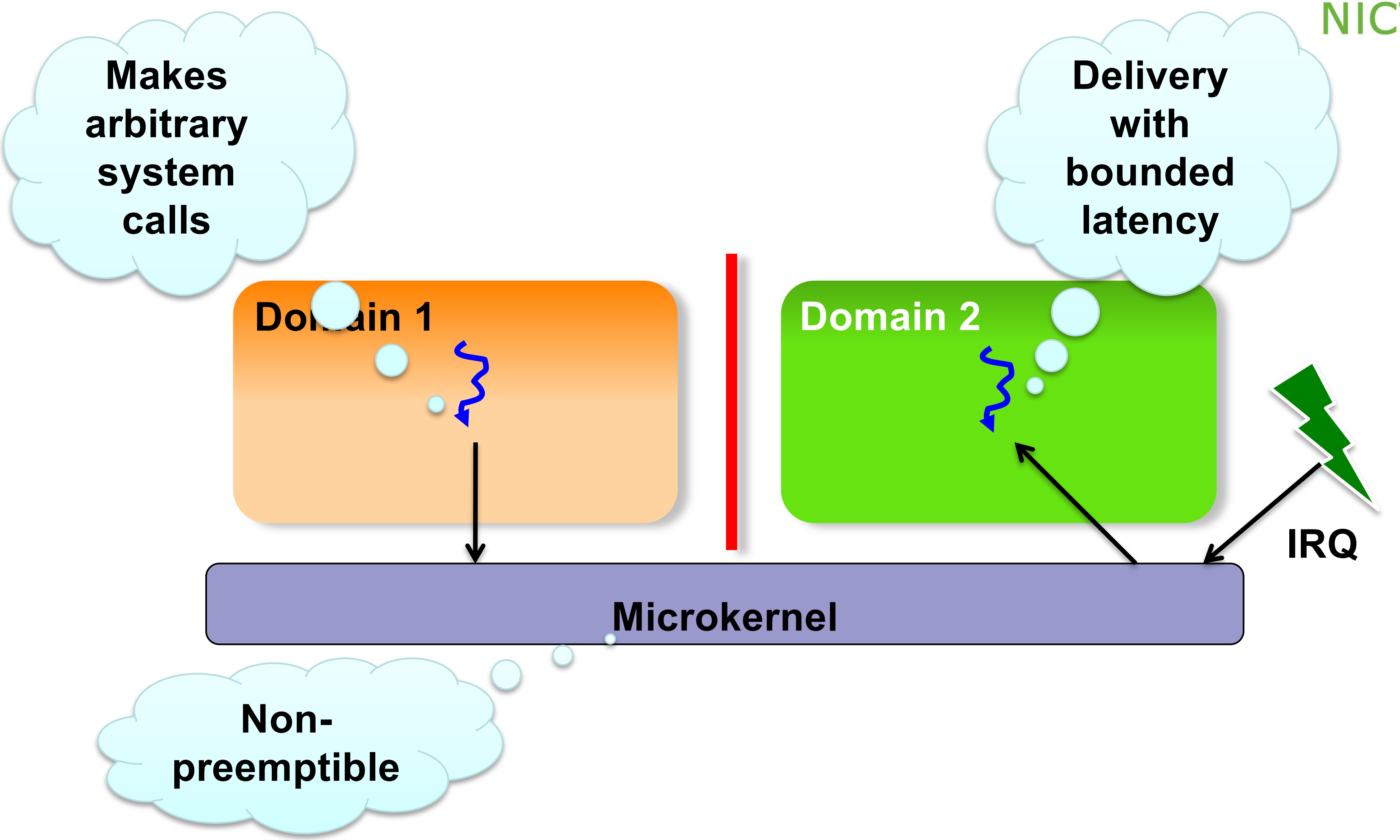
Non-interference proof in progress:

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

seL4 as Basis for Trustworthy Systems

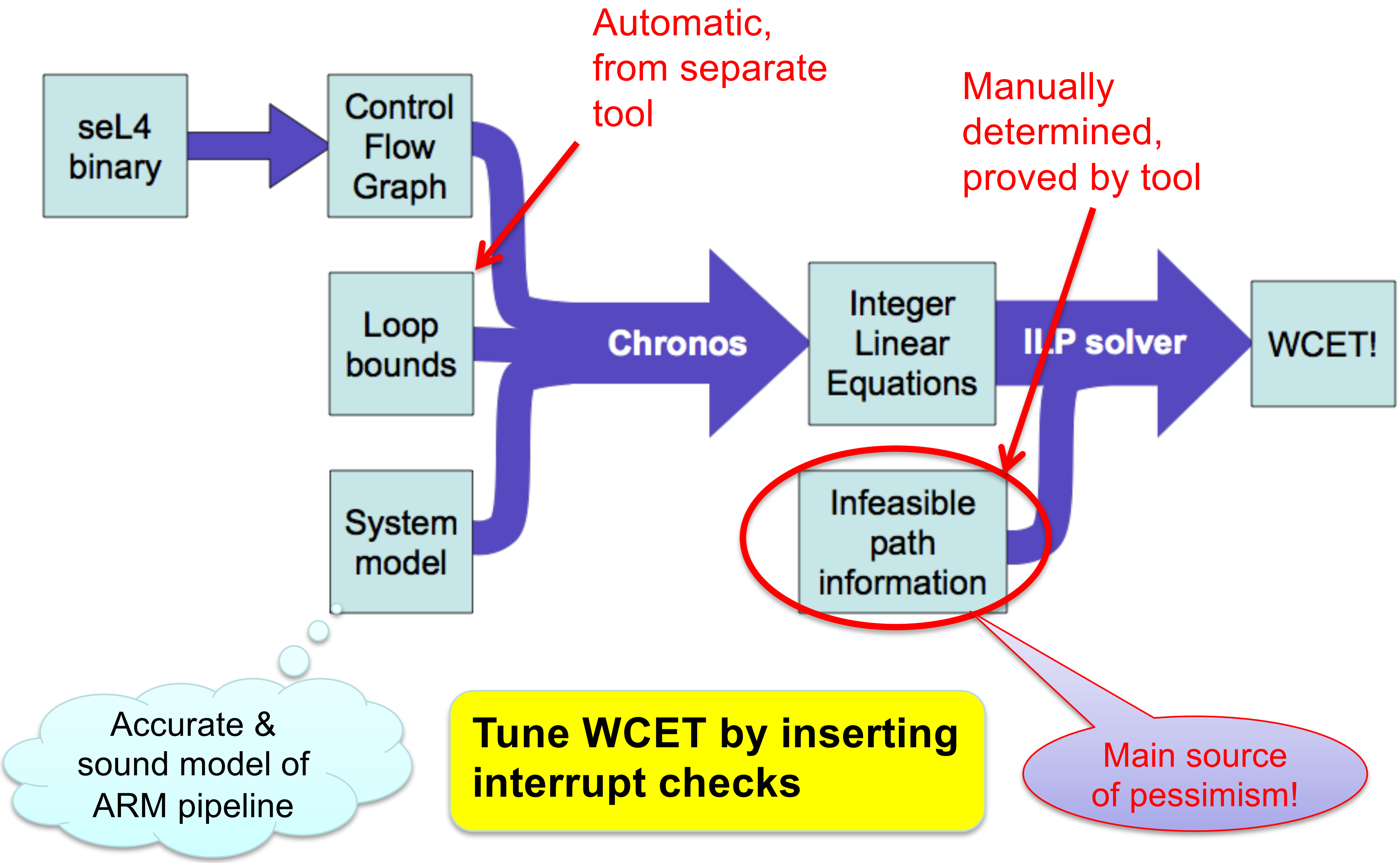


Timeliness

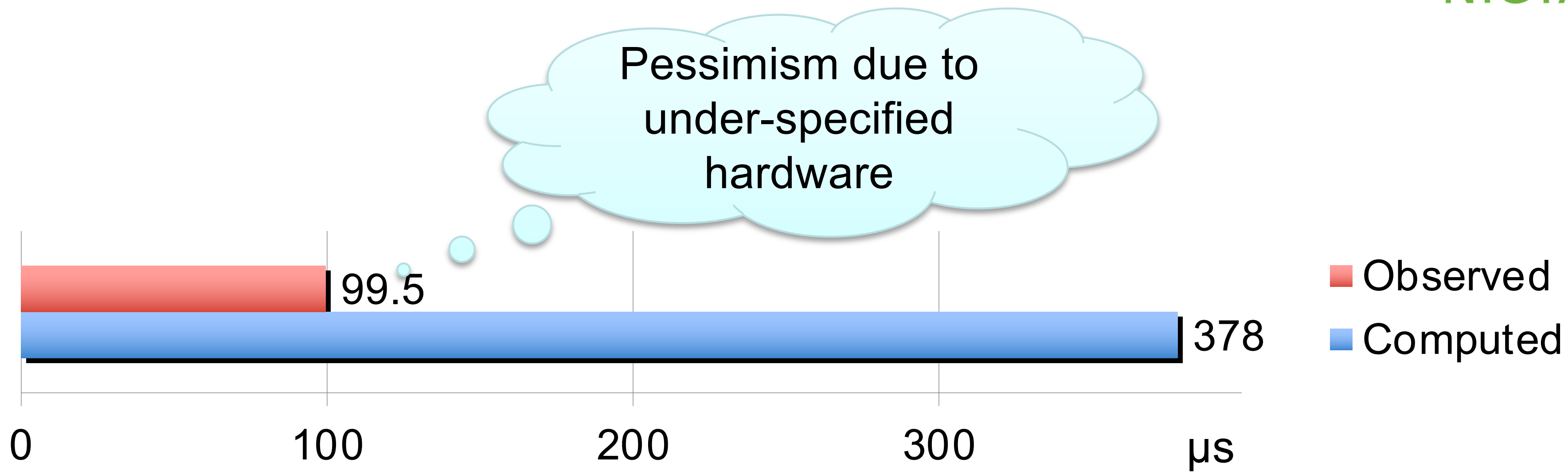


Need worst-case execution time (WCET) analysis of kernel

WCET Analysis Approach



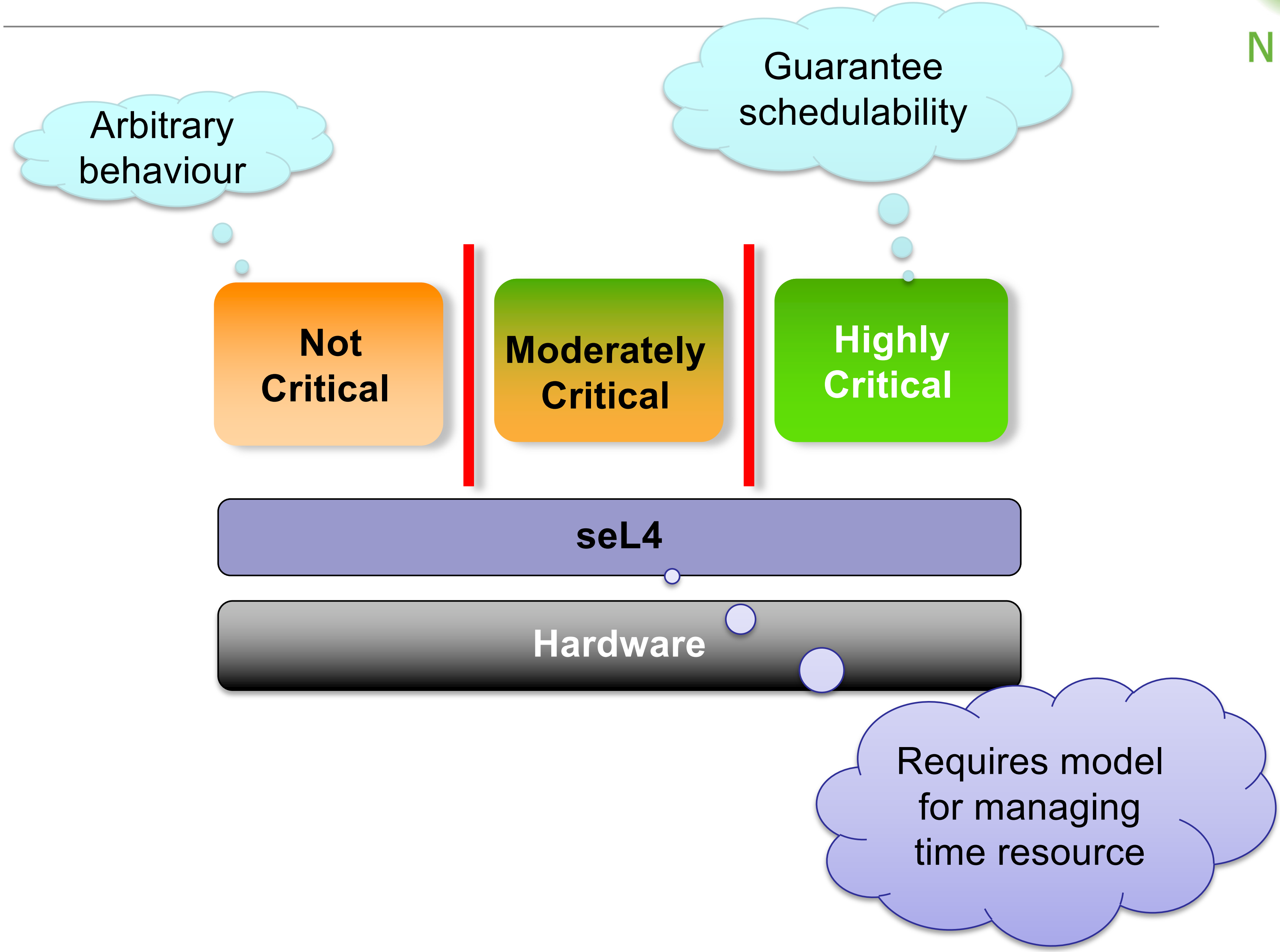
Result



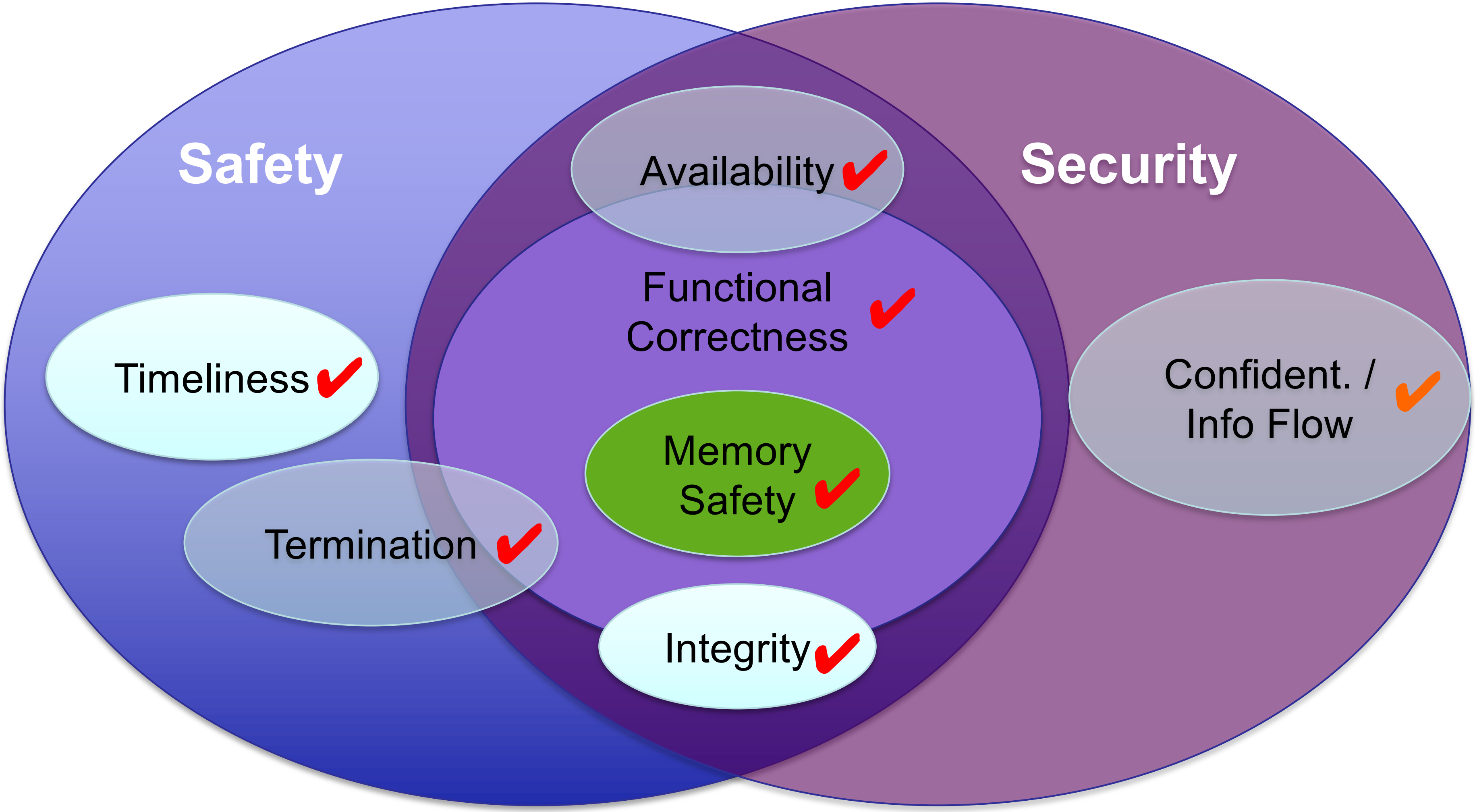
WCET presently limited by verification practicalities

- **10 µs seem achievable**

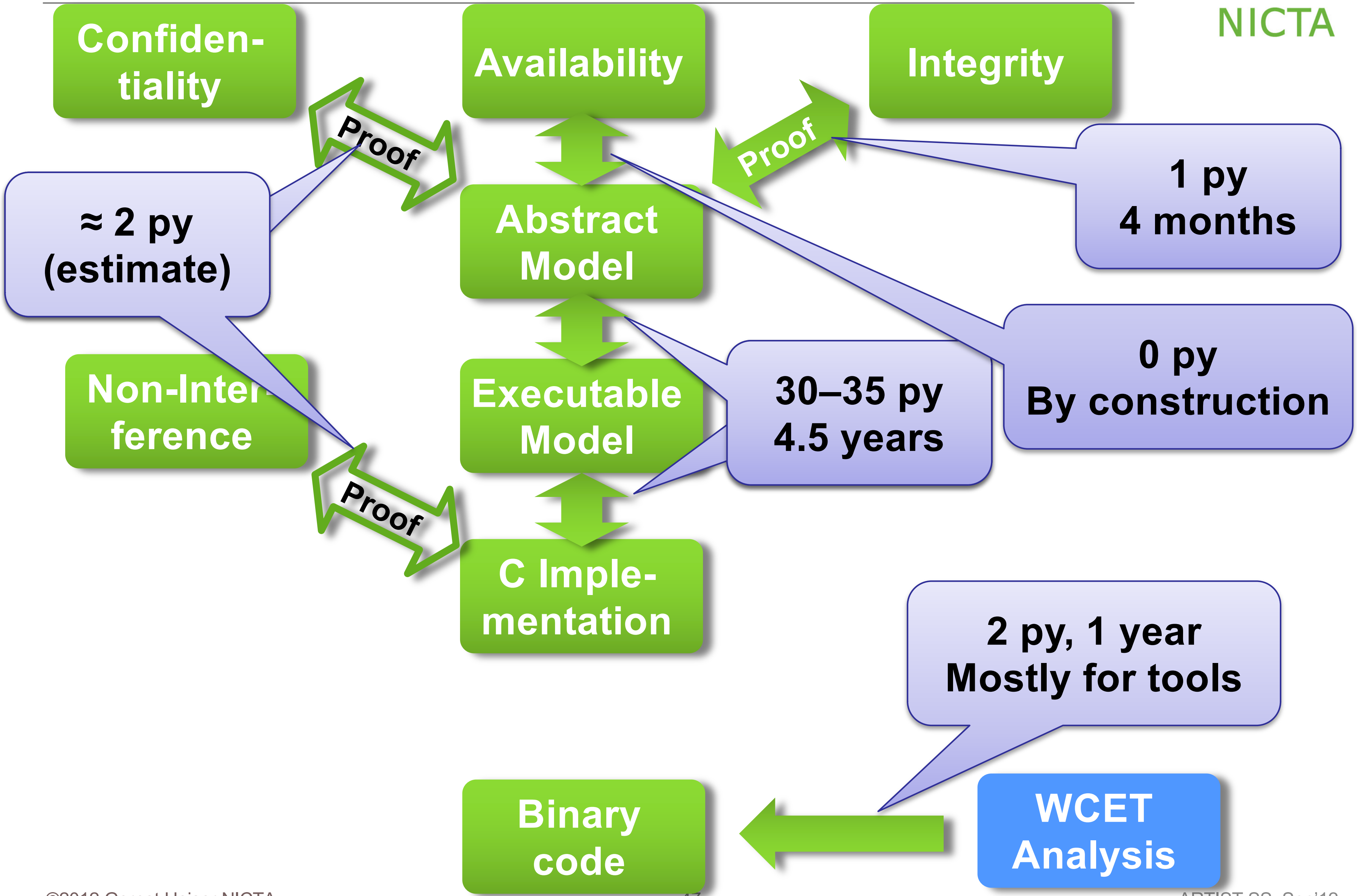
Future: Whole-System Schedulability



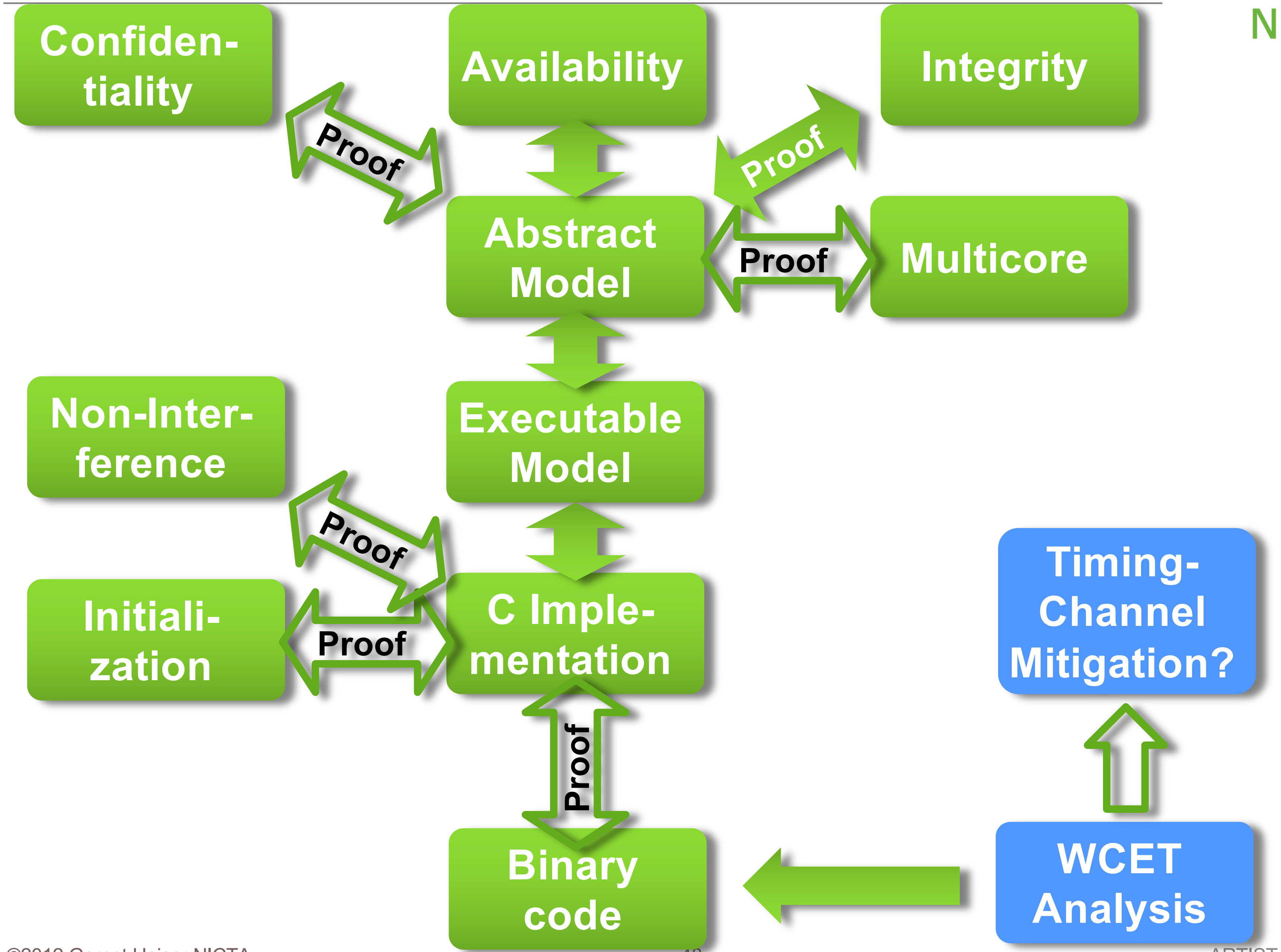
seL4 as Basis for Trustworthy Systems



Proving seL4 Trustworthiness



seL4 – the Next 24 Months

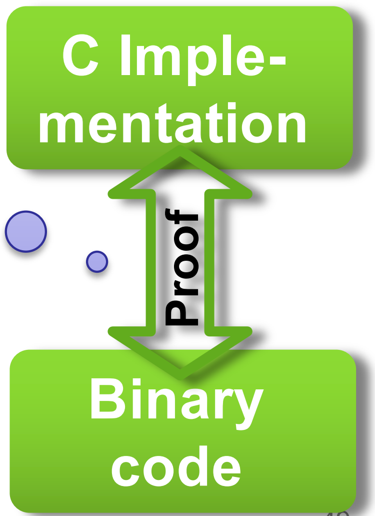


Binary Verification

IPC: one-way, zero-length		
Compiler	**gcc**	**Compcert**
Standard C code:	1455 cycles	3749 cycles
C fast path:	185 cycles	730 cycles

Uncompetitive performance!

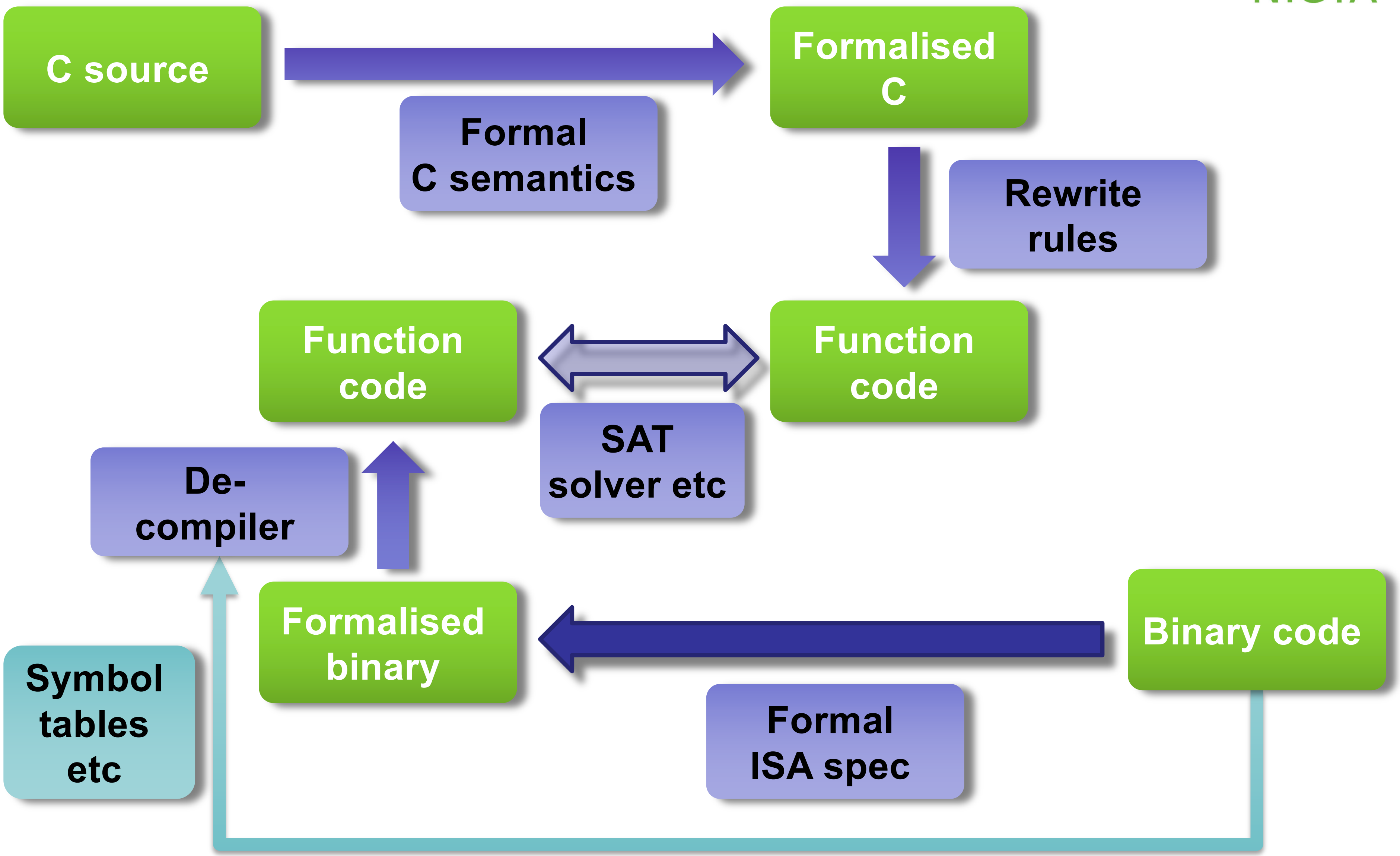
Use verified compiler (Compcert)?



Bigger problem:

- Our proofs are in Isabel/HOL, Compcert uses Coq
- We cannot prove that they use the same C semantics!

Binary Code Verification (In Progress)



Multikernel Verification

- By definition, multikernel images execute independently
  - except for explicit messaging

- To prove:
  - isolated images are initialised correctly
  - images maintain isolation at run time

Essentially non-interference

Agenda

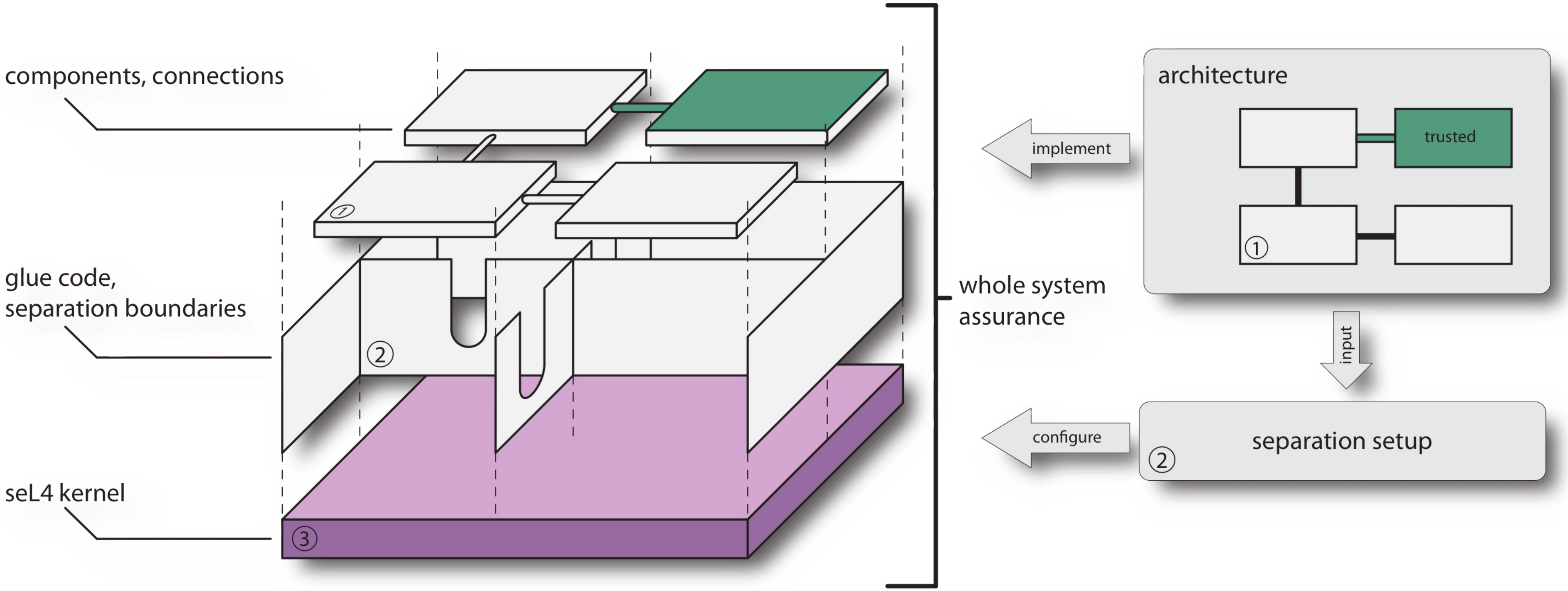
- Motivation
- Microkernels and seL4 design
- Establishing trustworthiness
- **From kernel to system**
- Sample system: Secure access controller

Phase Two: Full-System Guarantees

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

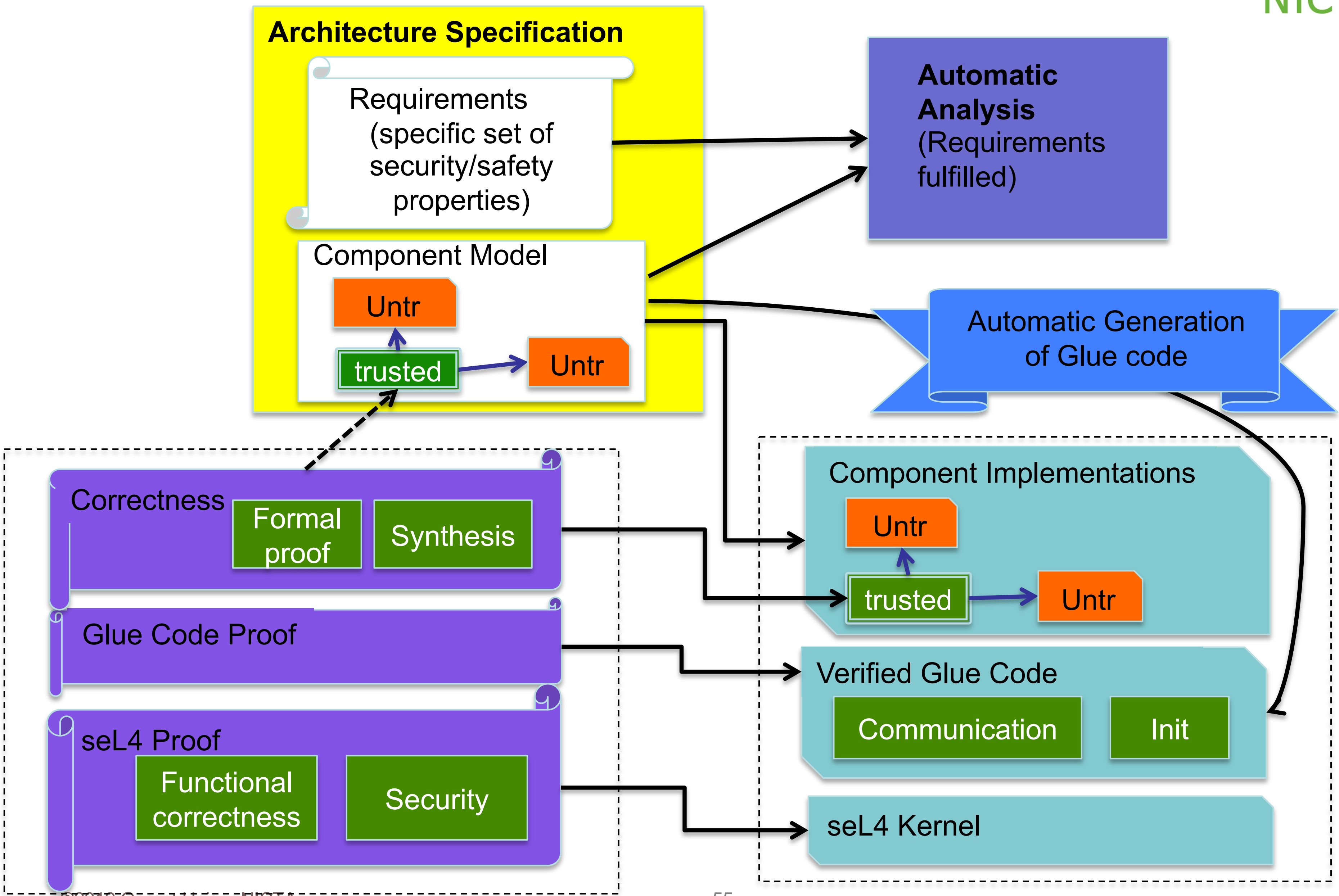
- Next step: Guarantees for real-world systems (1,000,000 LOC)

Overview of Approach

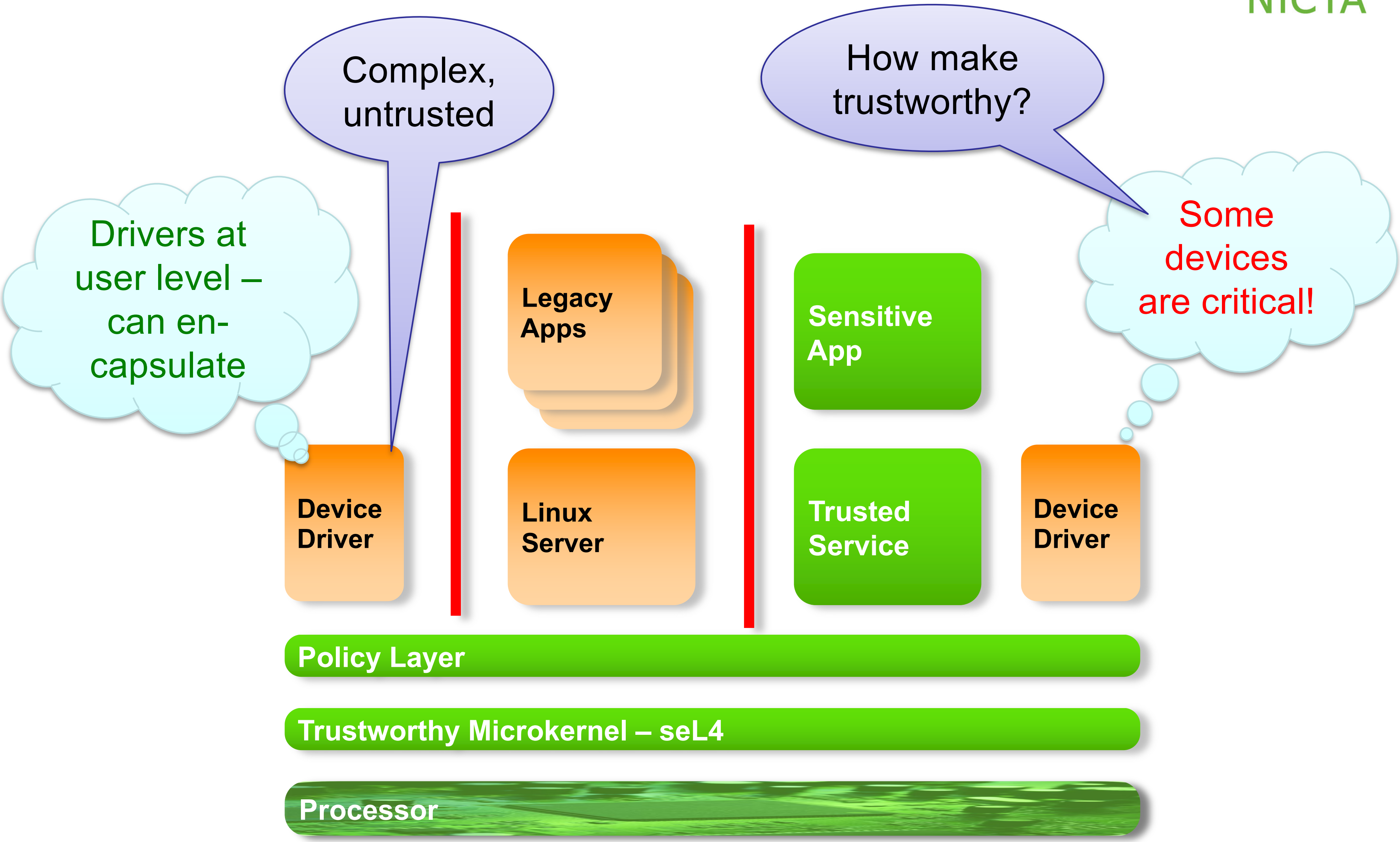


- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance

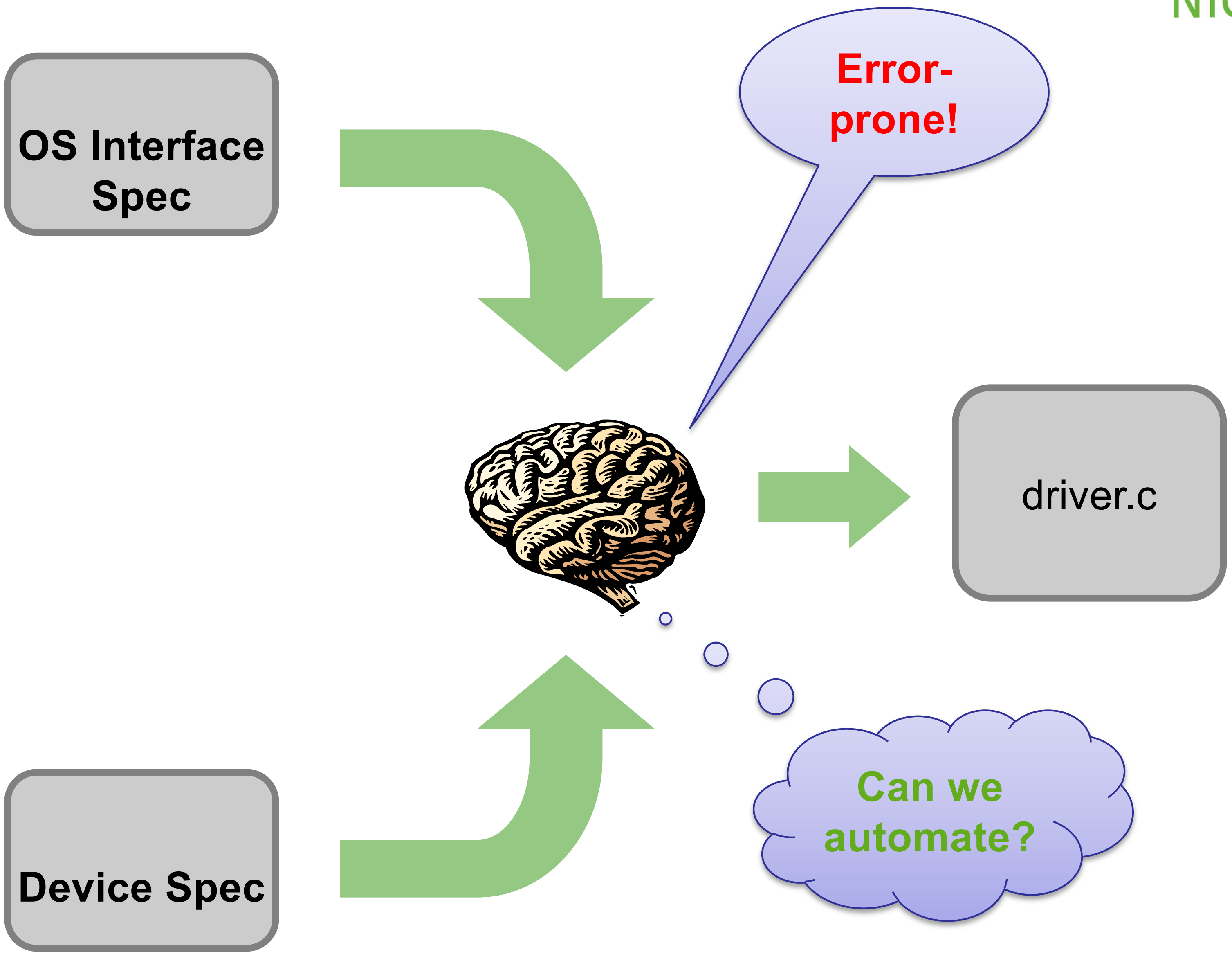
Architecting Security/Safety



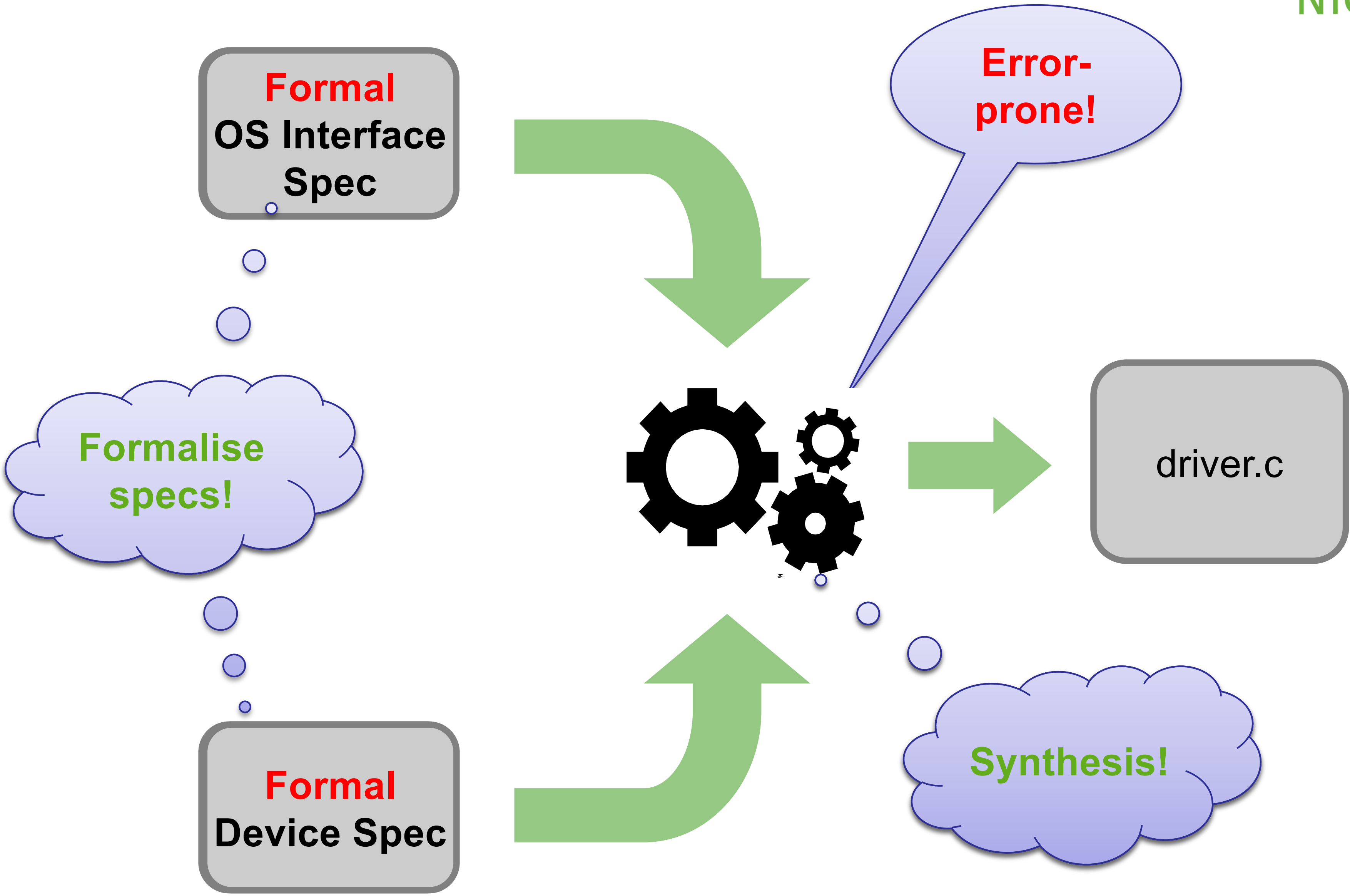
Device Drivers



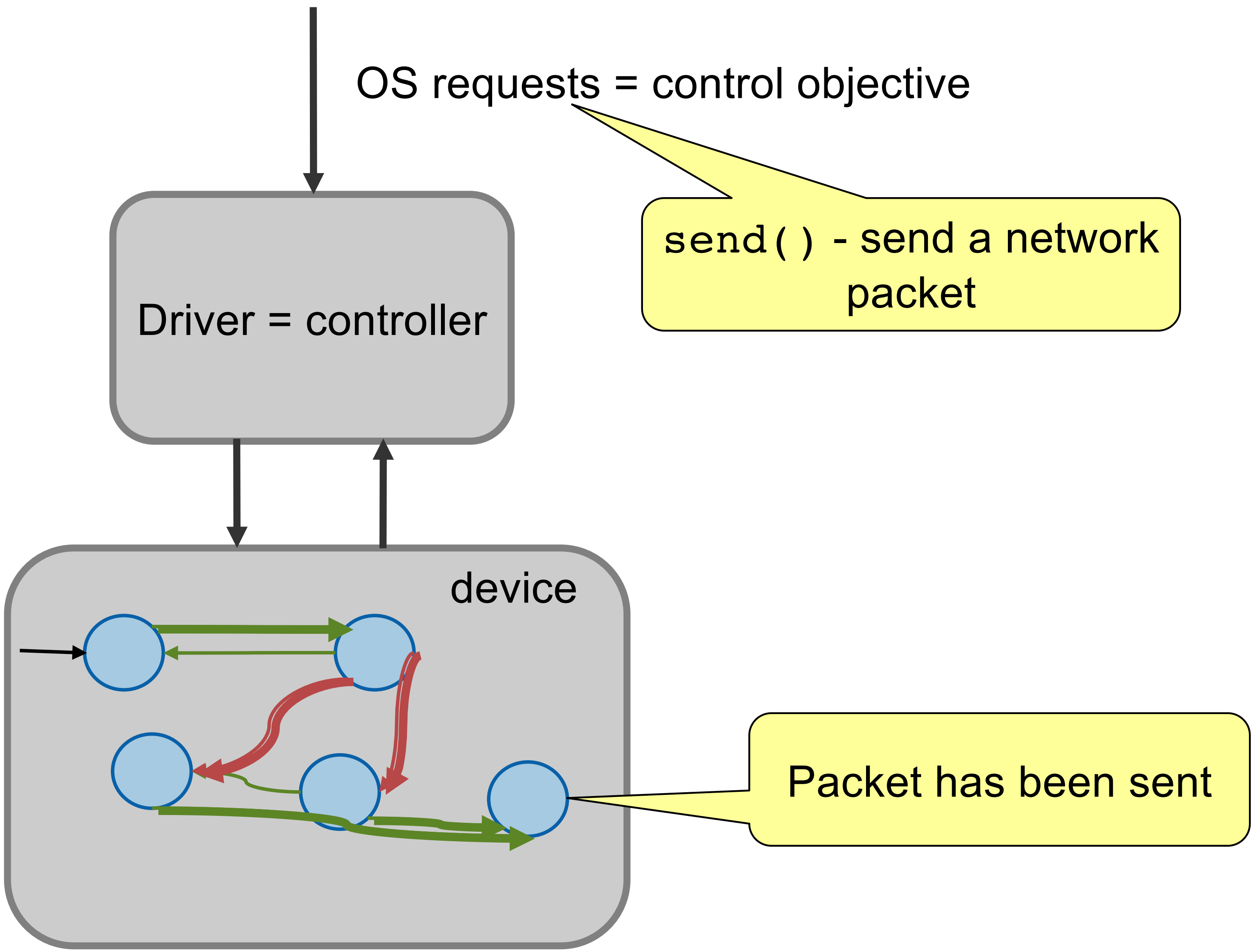
Driver Development



Driver Development

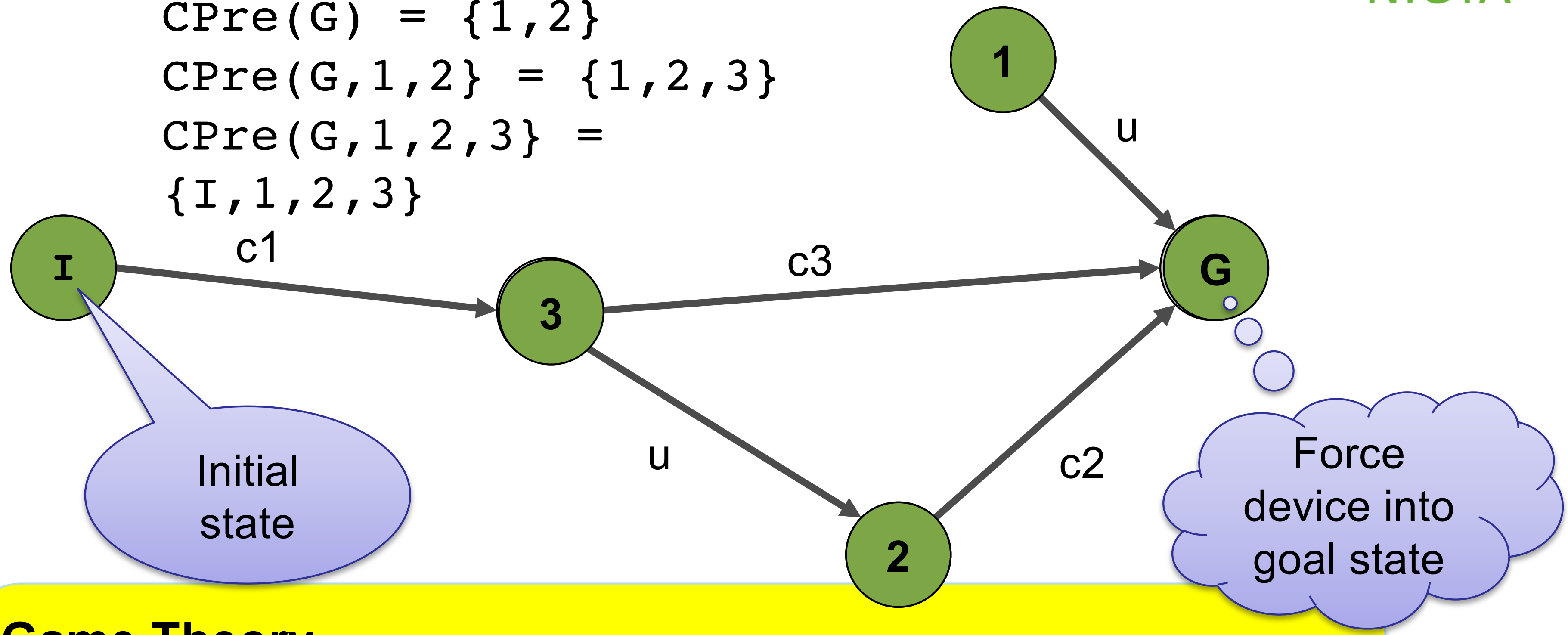


Driver Synthesis as Controller Synthesis



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!

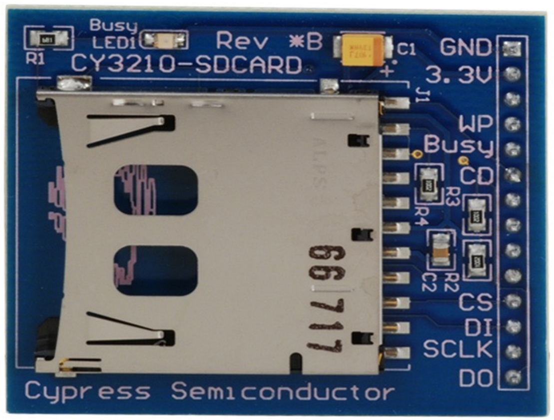
Device driver!

Drivers Synthesised (To Date)

IDE disk controller

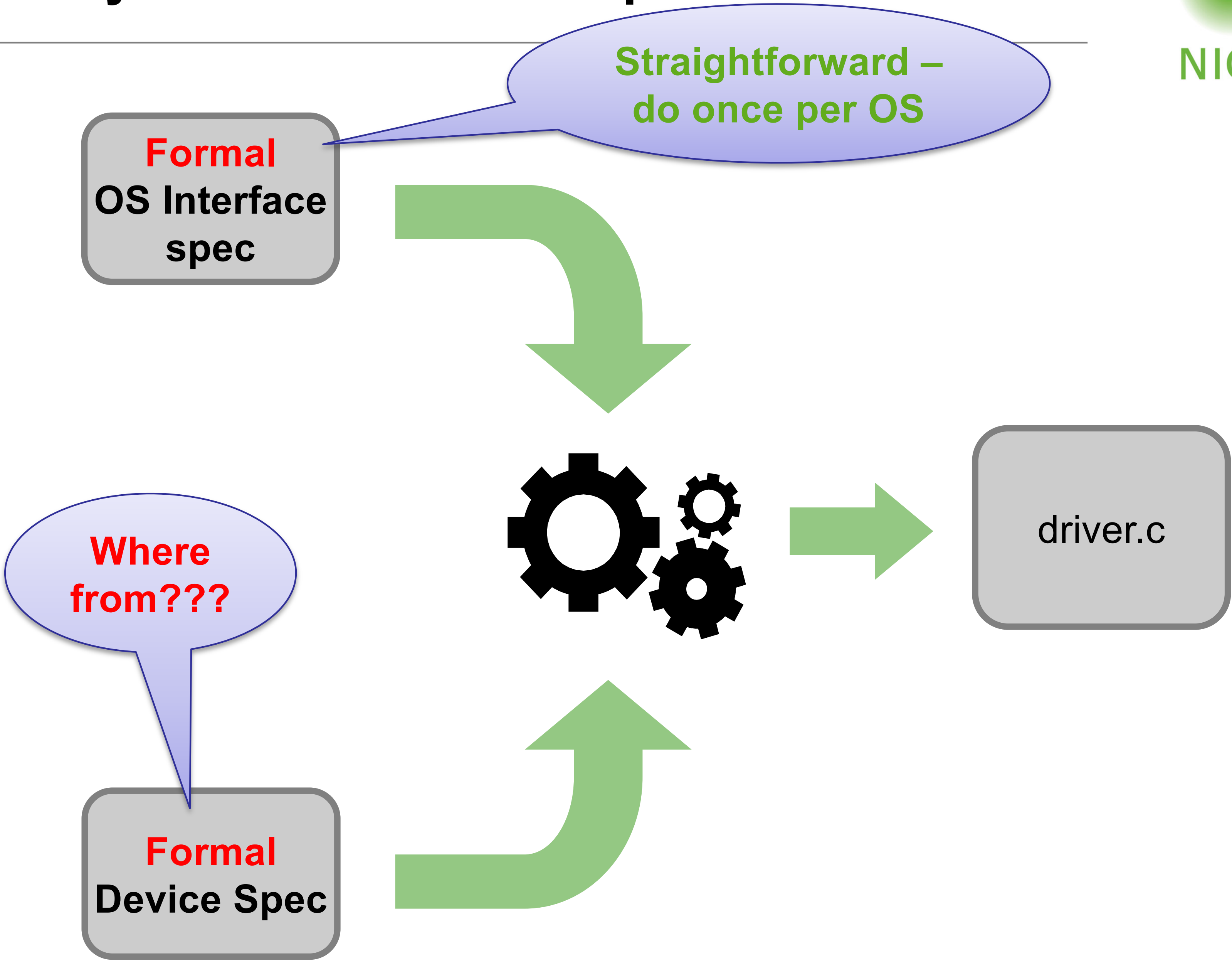
W5100 Eth shield

Asix AX88772
USB-to-Eth adapter



SD host controller

Driver Synthesis: Interface Specs



Hardware Design Workflow

NICTA

Informal specification

⇩

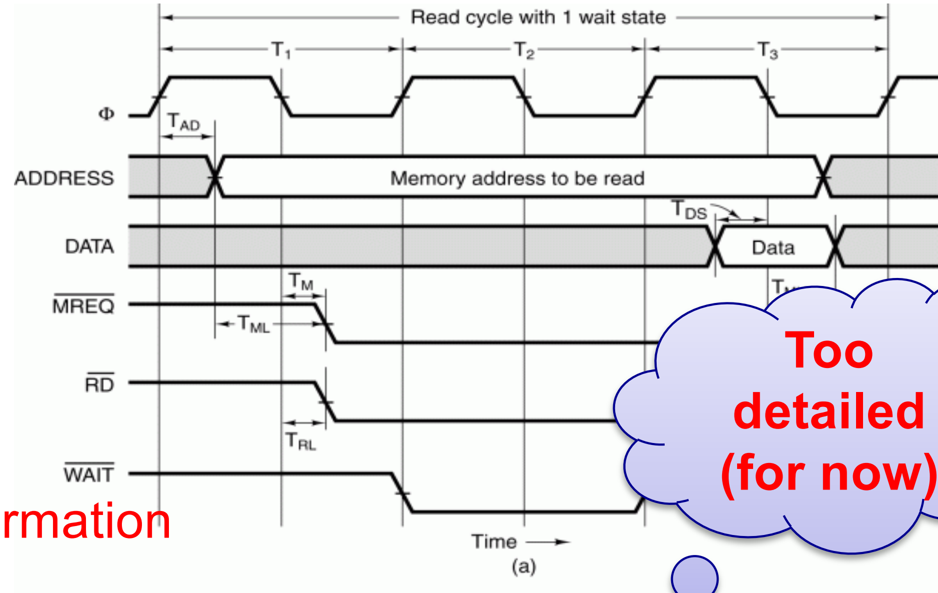
High-level model

⇩ 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

Informal specification

⇩

High-level model

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

⇩ Manual transform

Use for now

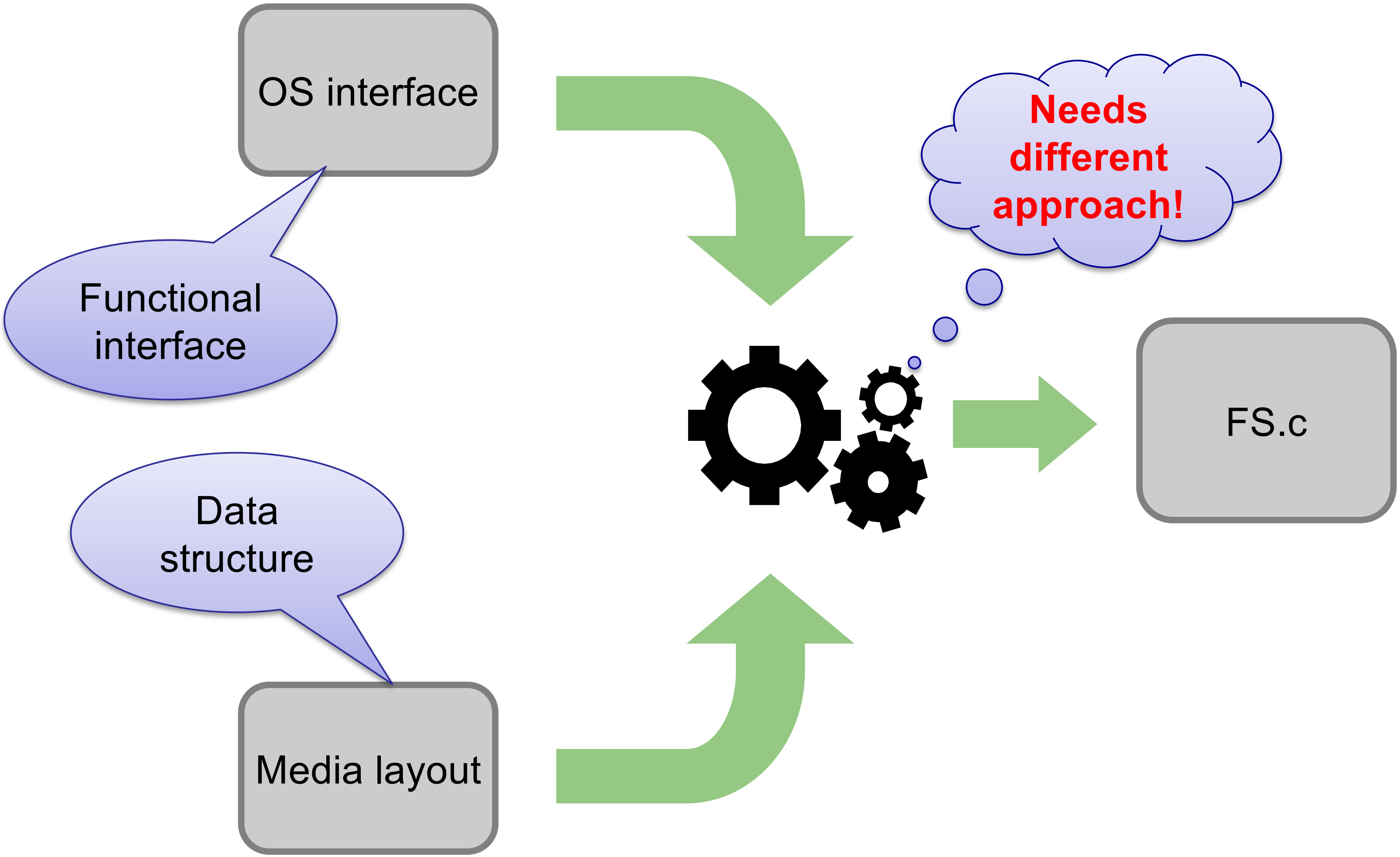
Register-transfer-level description

⇩

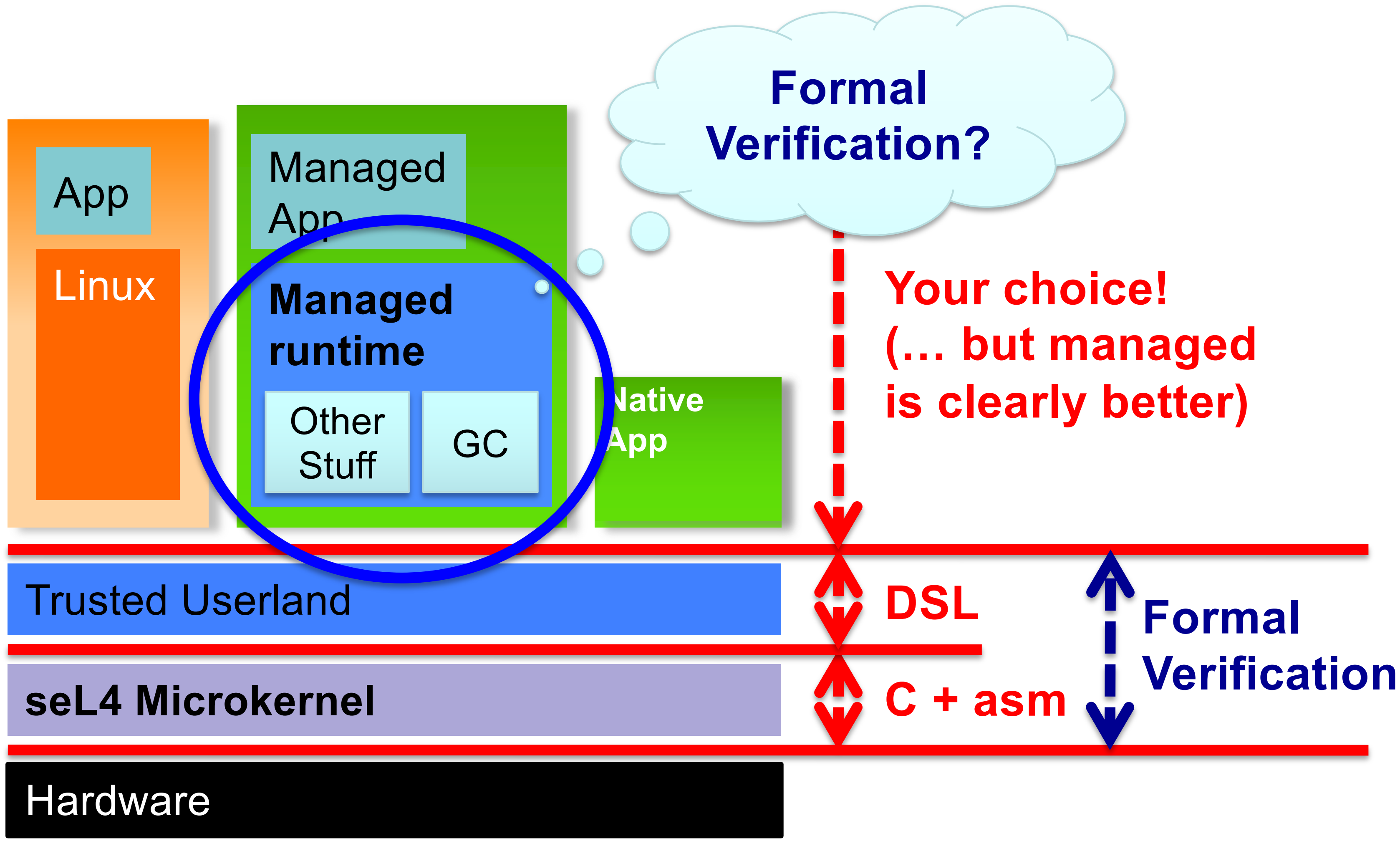
netlist

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

From Drivers to File Systems?



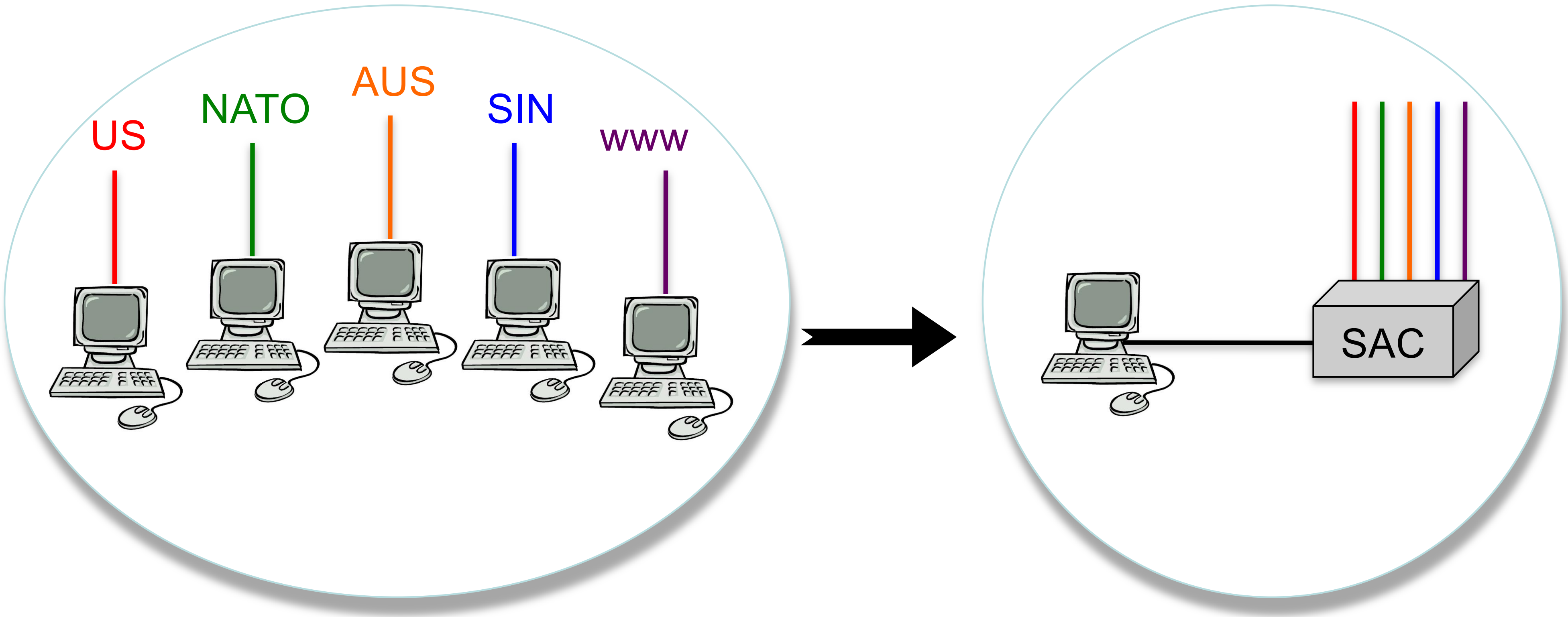
Building Secure Systems: Long-Term View



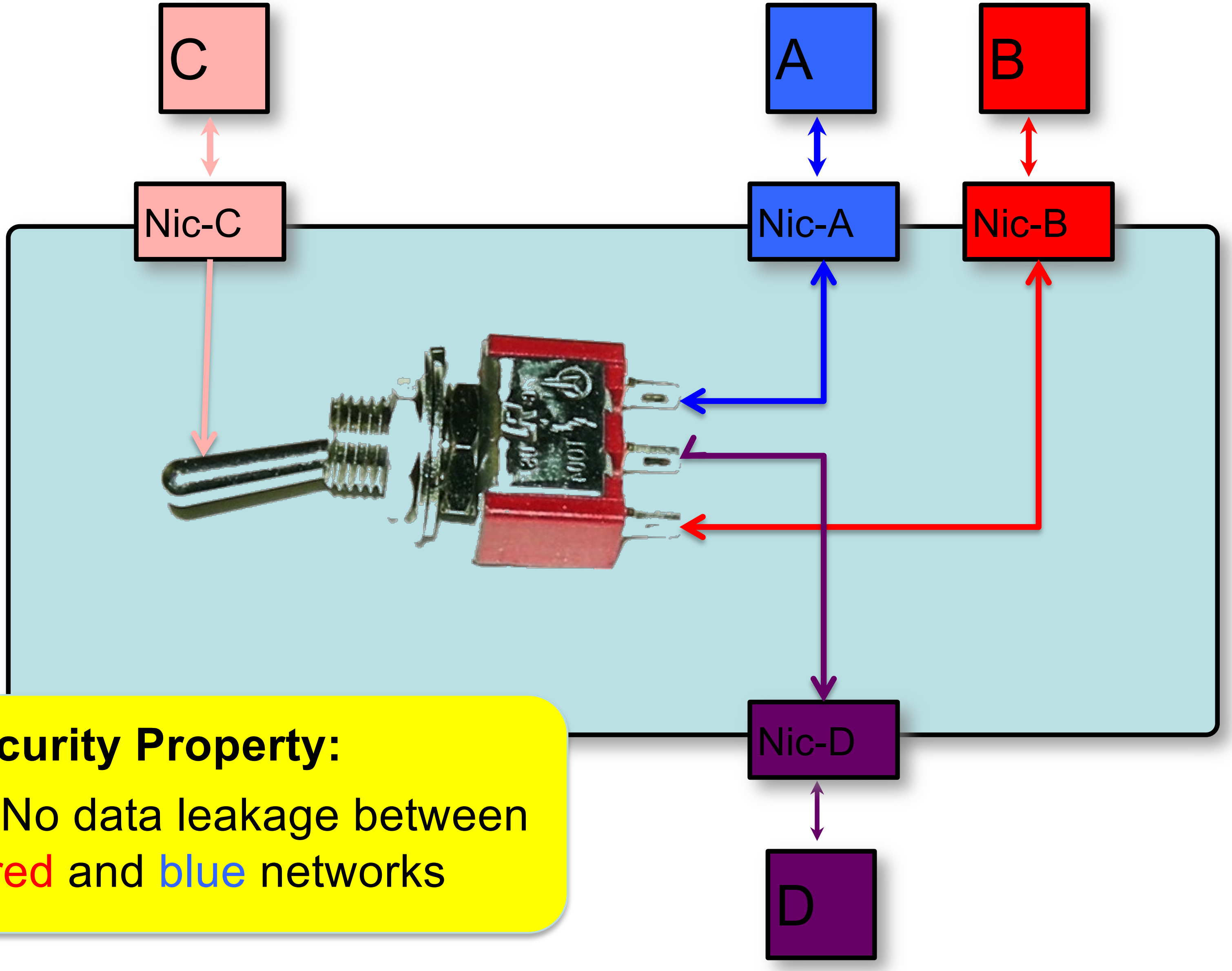
Agenda

- Motivation
- Microkernels and seL4 design
- Establishing trustworthiness
- From kernel to system
- **Sample system: Secure access controller**

Proof of Concept: Secure Access Controller



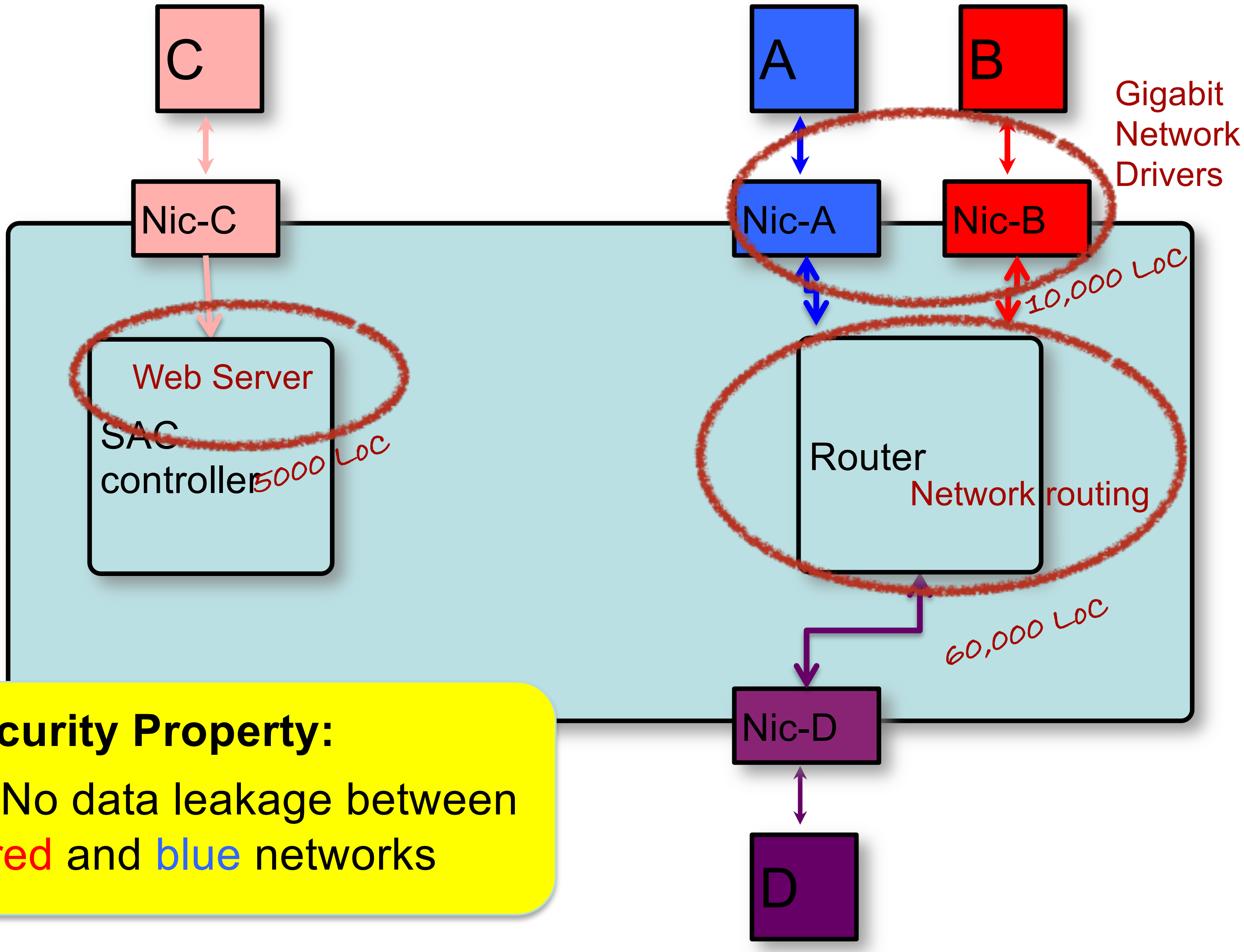
Logical Function



Security Property:

- No data leakage between red and blue networks

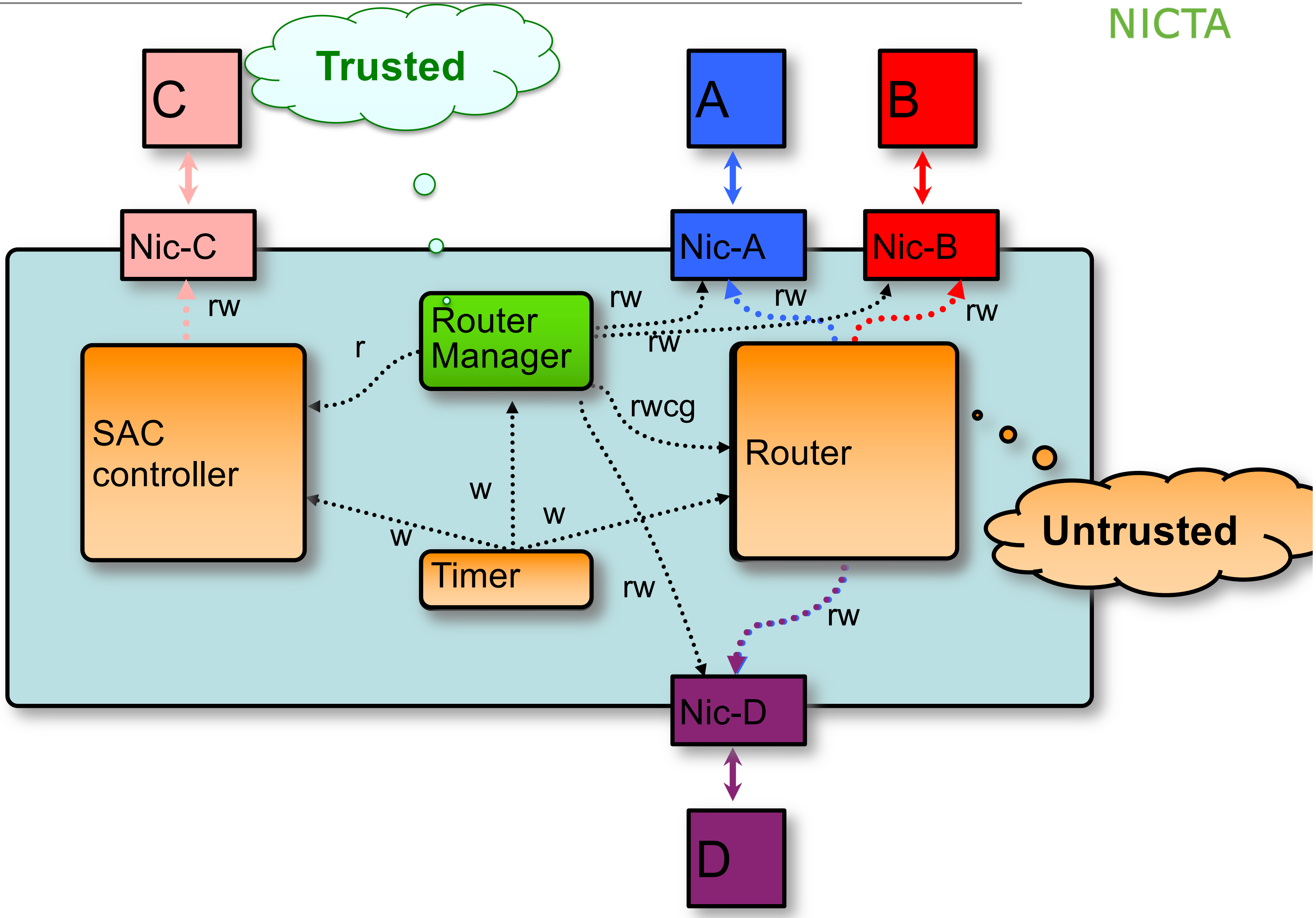
Logical Function



Security Property:

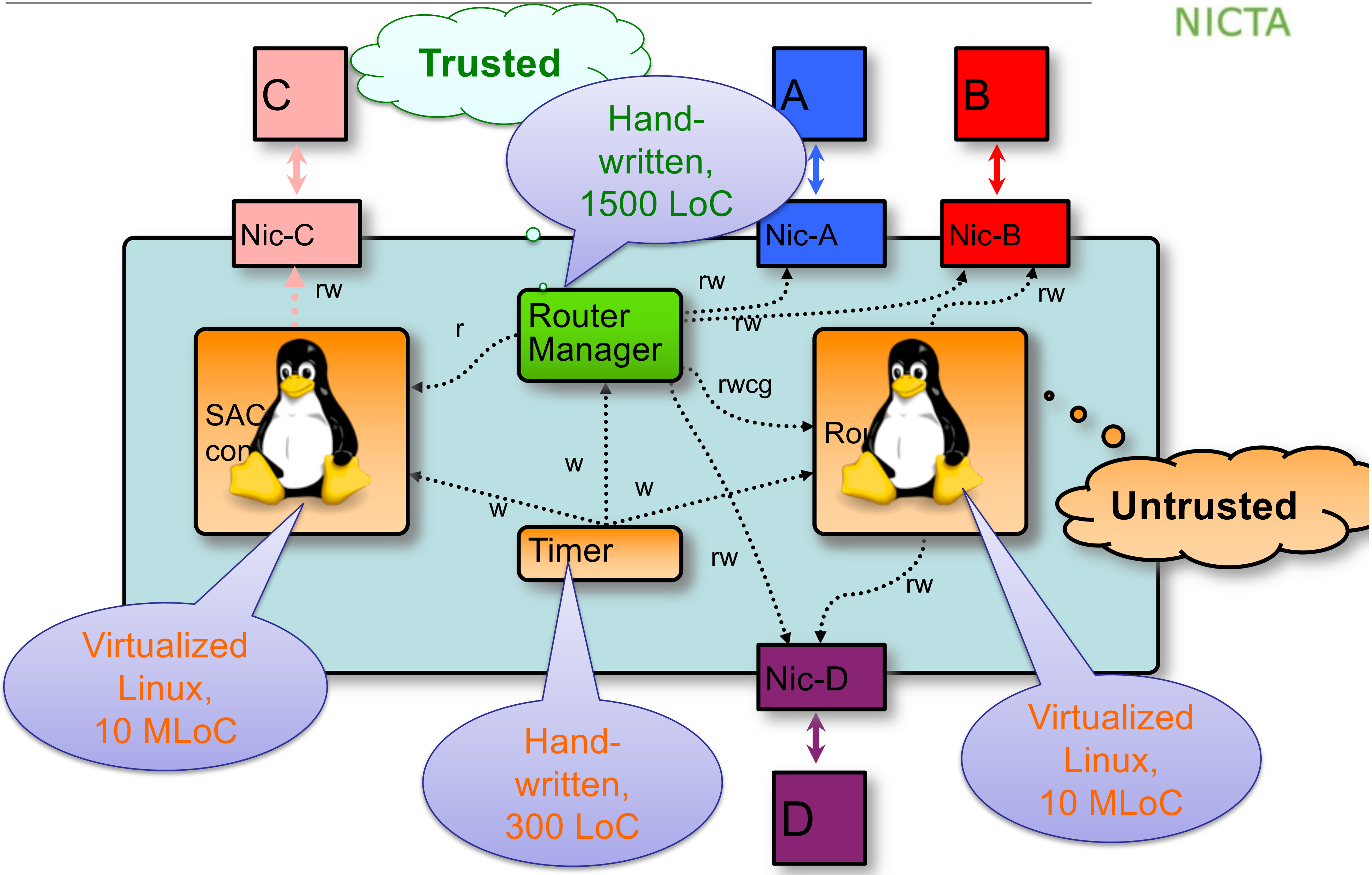
- No data leakage between red and blue networks

Minimal TCB

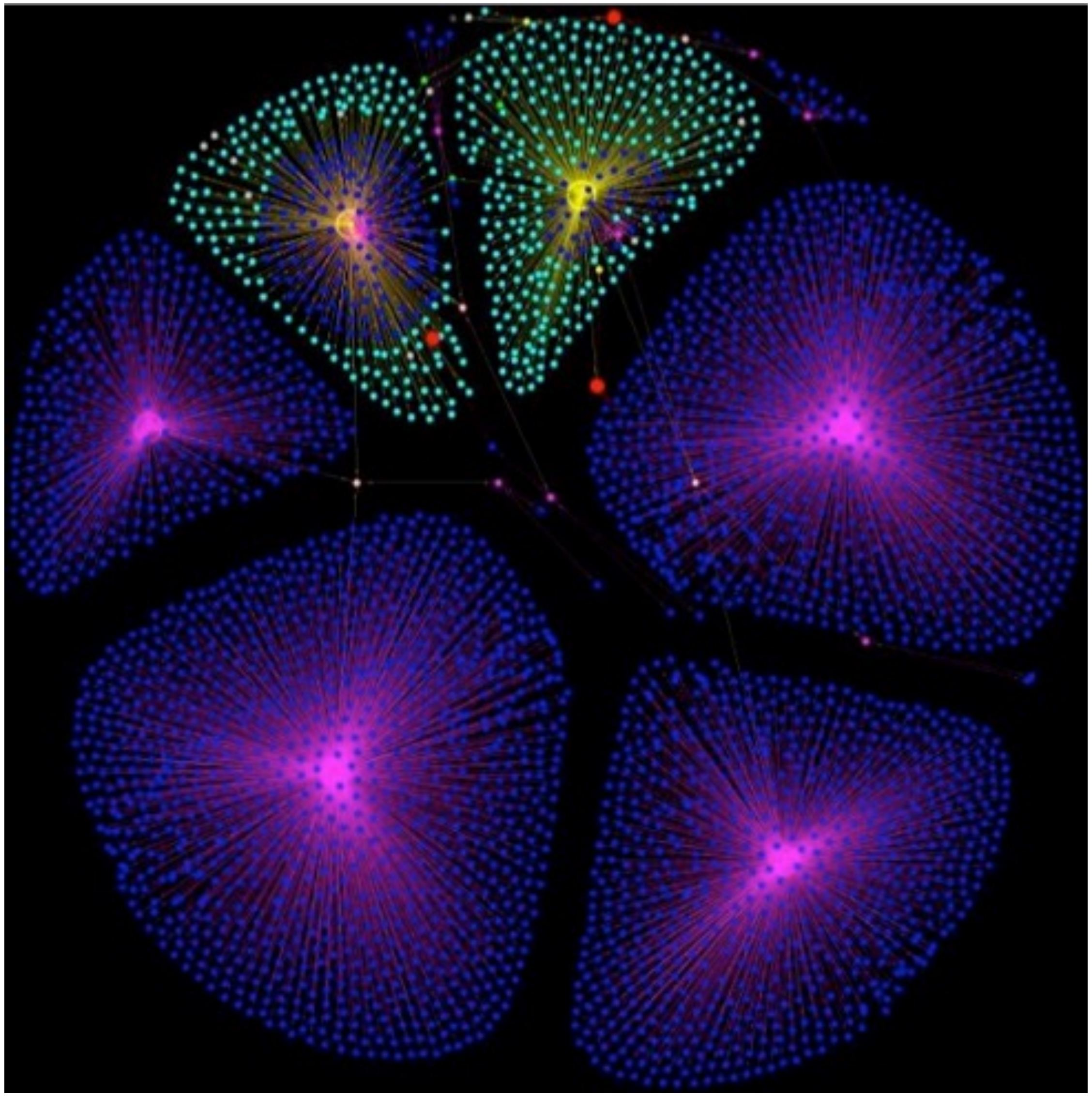


C
Trusted
A
B
Nic-C
Nic-A
Nic-B
rw
rw
rw
rw
Router Manager
r
rw
SAC controller
rwcg
Router
w
w
w
Untrusted
Timer
rw
rw
Nic-D
D

Implementation

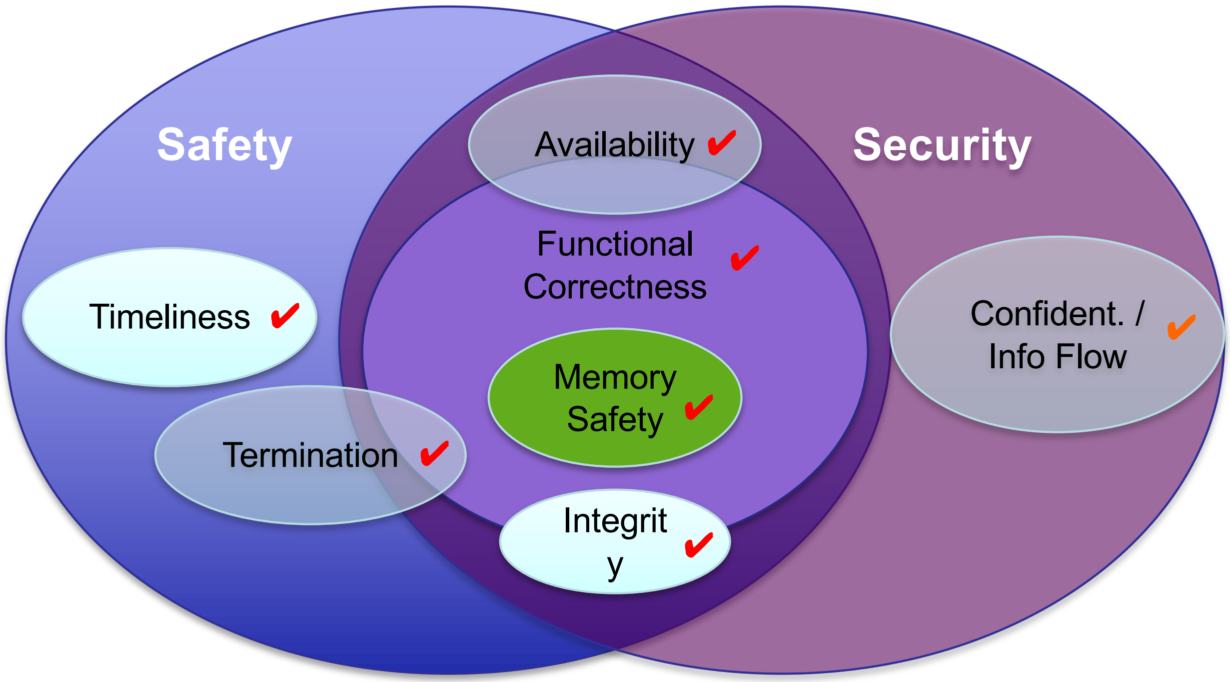


Access Rights



Trustworthy Systems – We've Made a Start!

NICTA



Thank You!

mailto:gernot@nicta.com.au

Twitter @GernotHeiser

Google: "nicta trustworthy systems"