How to Build Truly 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

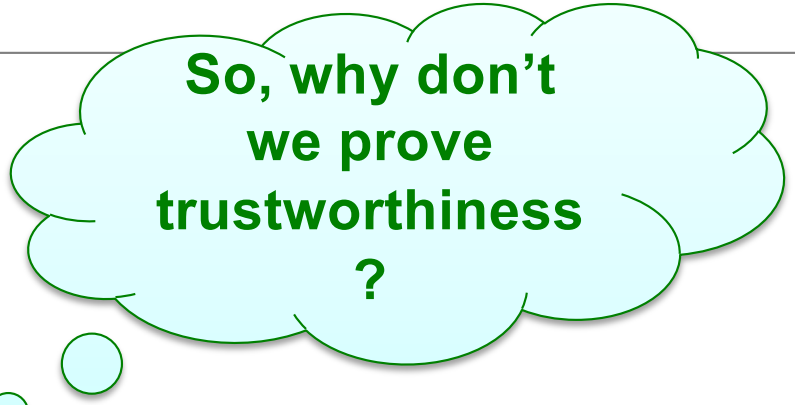

Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue

Present Systems are *NOT* Trustworthy!

NICTA

What's Next?

NICTA



***Claim*:**

A system must be considered *not trustworthy* unless *proved* otherwise!

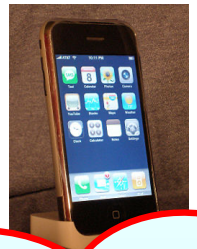
Corollary [with apologies to Dijkstra]:

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

©2012 Gernot Heiser NICTA

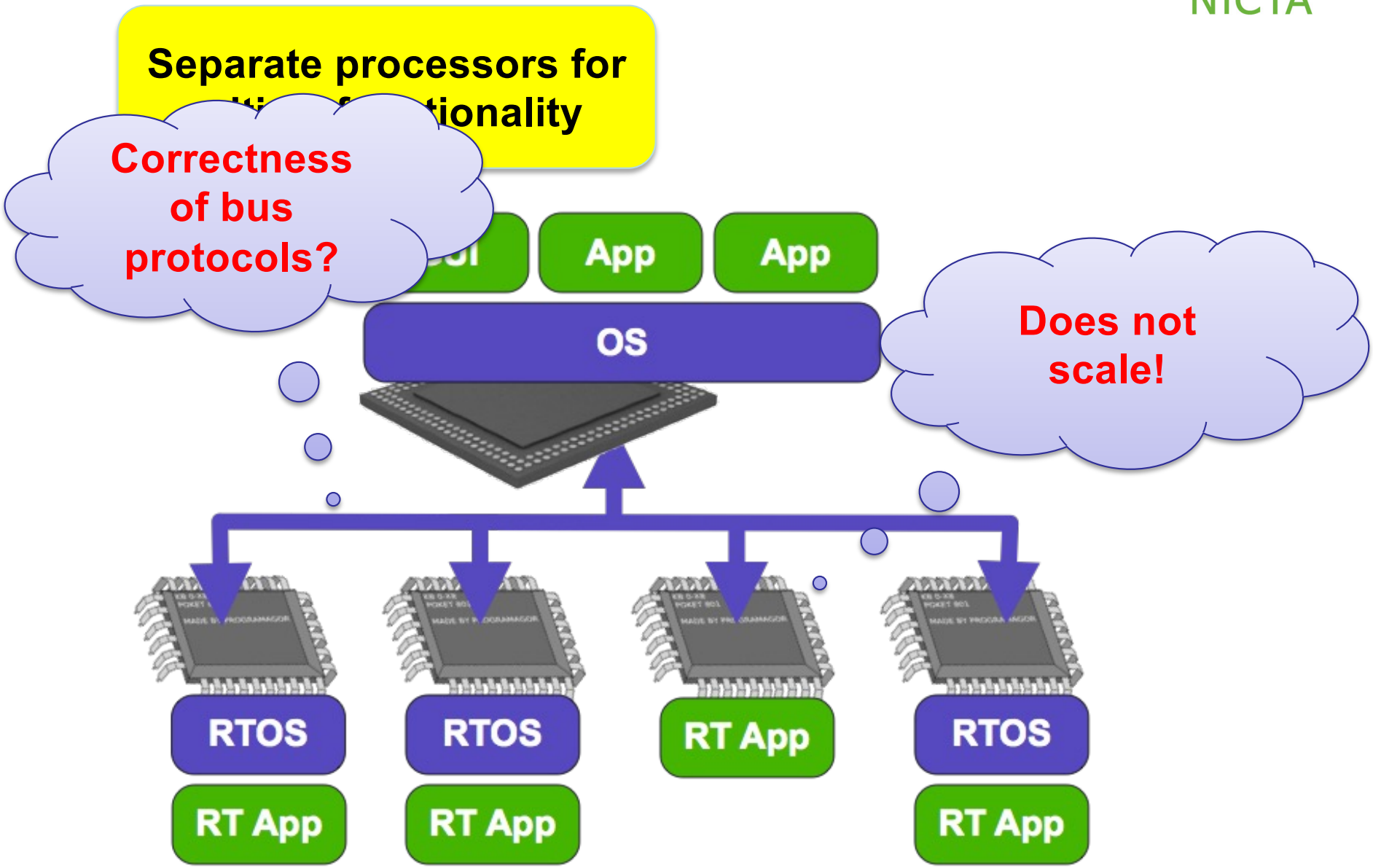
Core Issue: Complexity

- 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 c s
  - Automotive infotainment a
  - Gigabytes of software on 100 CPUs…

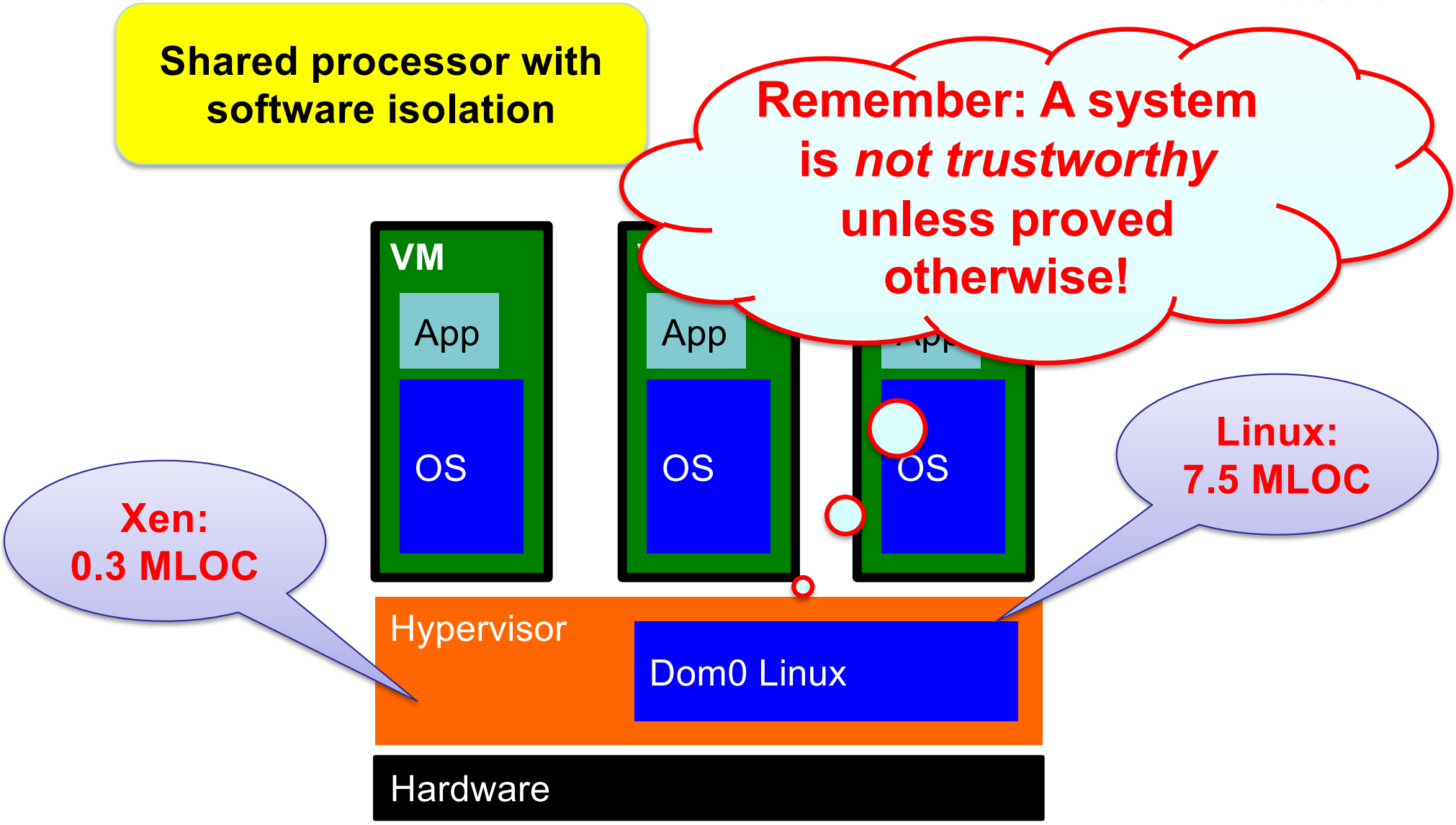


Systems far too complex to prove their trustworthiness!

Dealing with Complexity: Physical Isolation



How About Logical Isolation?



Our Vision: Trustworthy Systems

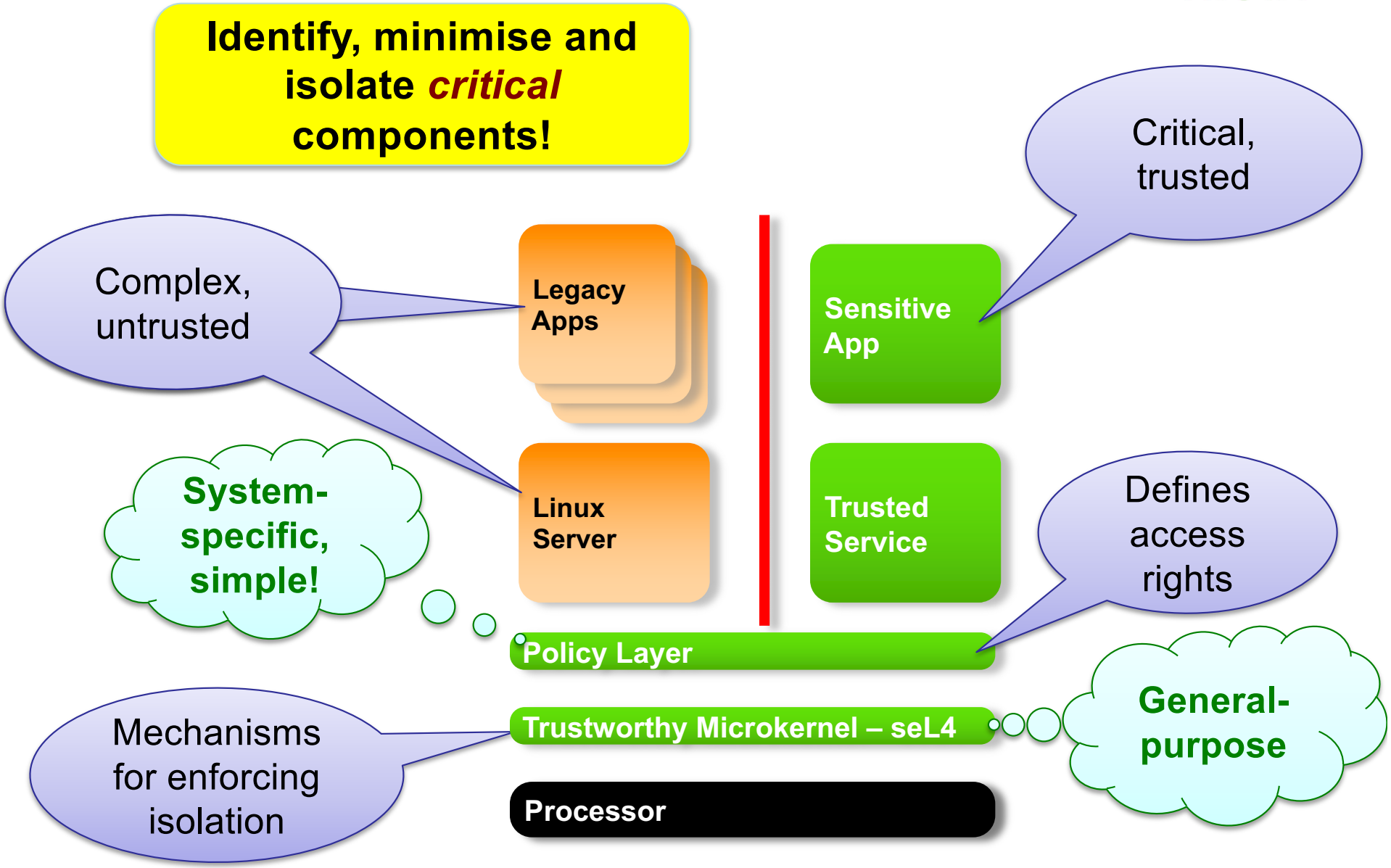
NICTA

Suitable for real-world systems

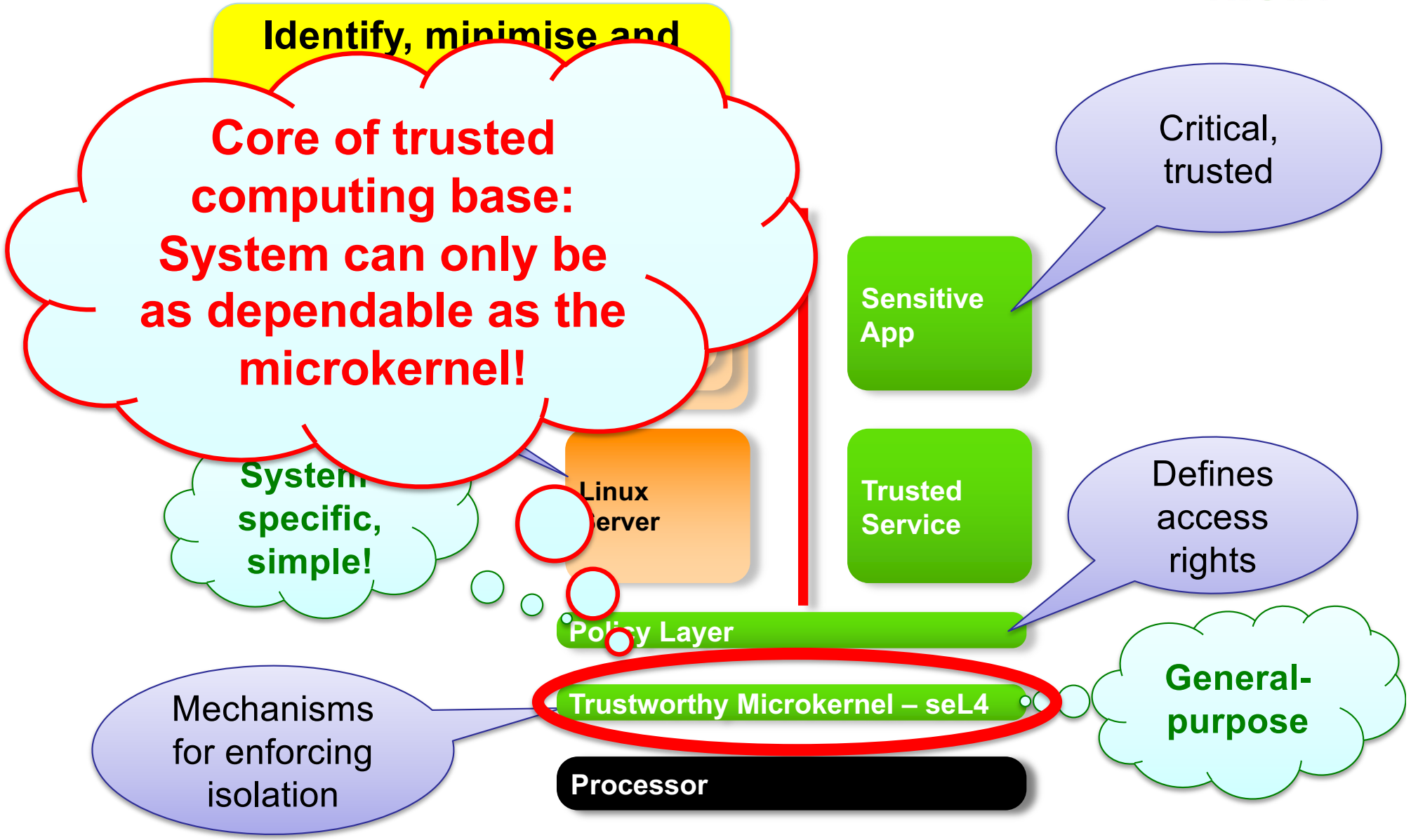
We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

Hard *guarantees* on safety/security/ reliability

Isolation is Key!



Isolation is Key!



NICTA Trustworthy Systems Agenda

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
     - despite 99 % of code untrusted!

©2012 Gernot Heiser NICTA

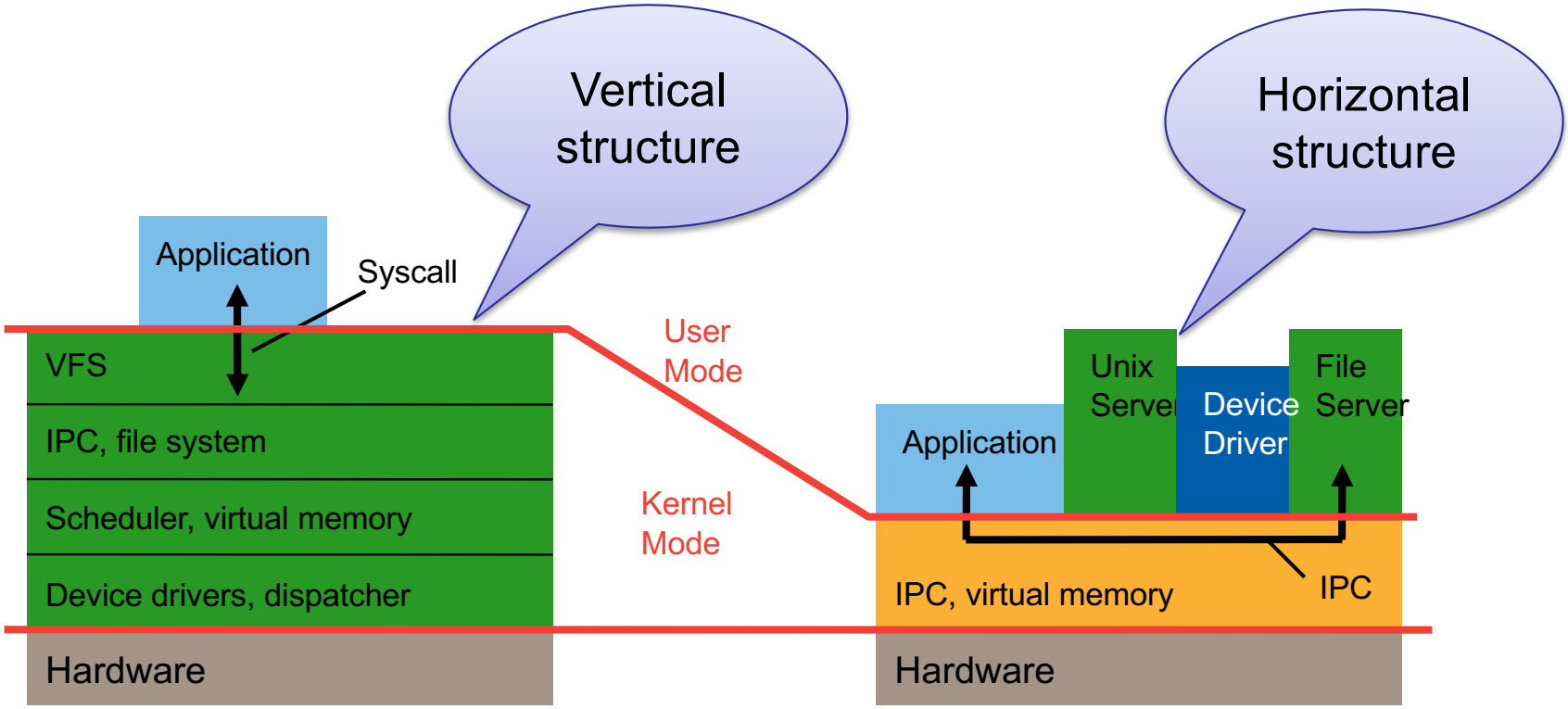
Agenda

- Motivation
- **What is a microkernel, and what is L4?**
- seL4 – designed for trustworthiness
- Establishing trustworthiness
- From kernel to system
- Sample system 1: Secure access controller
- Sample system 2: RapiLog

Monolithic Kernels vs Microkernels

NICTA

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

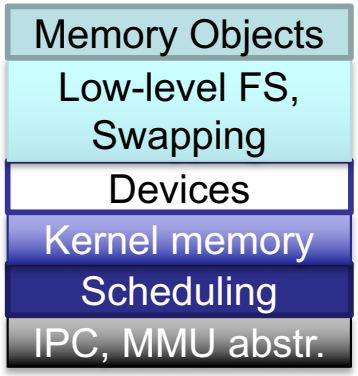


Microkernel Evolution

NICTA

First generation

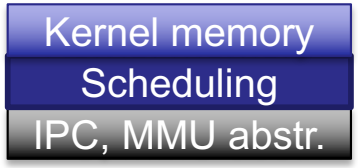
- Eg Mach (’87)



- 180 syscalls
- 100 kLOC
- 100 µs IPC

Second generation

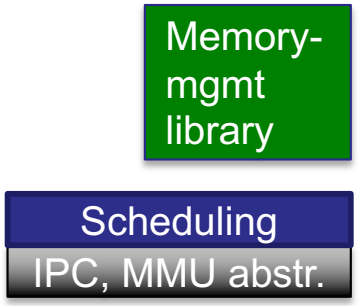
- Eg L4 (‘95)



- ~7 syscalls
- ~10 kLOC
- ~ 1 µs IPC

Third generation

- seL4 (‘09)



- ~3 syscalls
- 9 kLOC
- < 1 µs IPC

2nd-Generation Microkernels

- 1st-generation kernels (Mach, Chorus) were a failure
  - Complex, inflexible, slow
- L4 was first 2nd-G microkernel [Liedtke, SOSP'93, SOSP'95]
  - Radical simplification & manual micro-optimisation, fast IPC

A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system's required functionality

- Family of L4 kernels:
  - Original GMD assembler kernel ('95)
  - Fiasco (Dresden '98), Hazelnut (Karlsruhe '99), Pistachio (Karlsruhe/UNSW '02), L4-embedded (NICTA '04)
    - L4-embedded commercialised as OKL4 by Open Kernel Labs
    - Deployed in >1.5 billion phones
  - Commercial clones (PikeOS, P4, CodeZero, …)
  - Approach adopted e.g. in QNX ('82) and Green Hills Integrity ('90s)

Microkernel Principles: Minimality

Strict adherence to minimality leads to a very small kernel

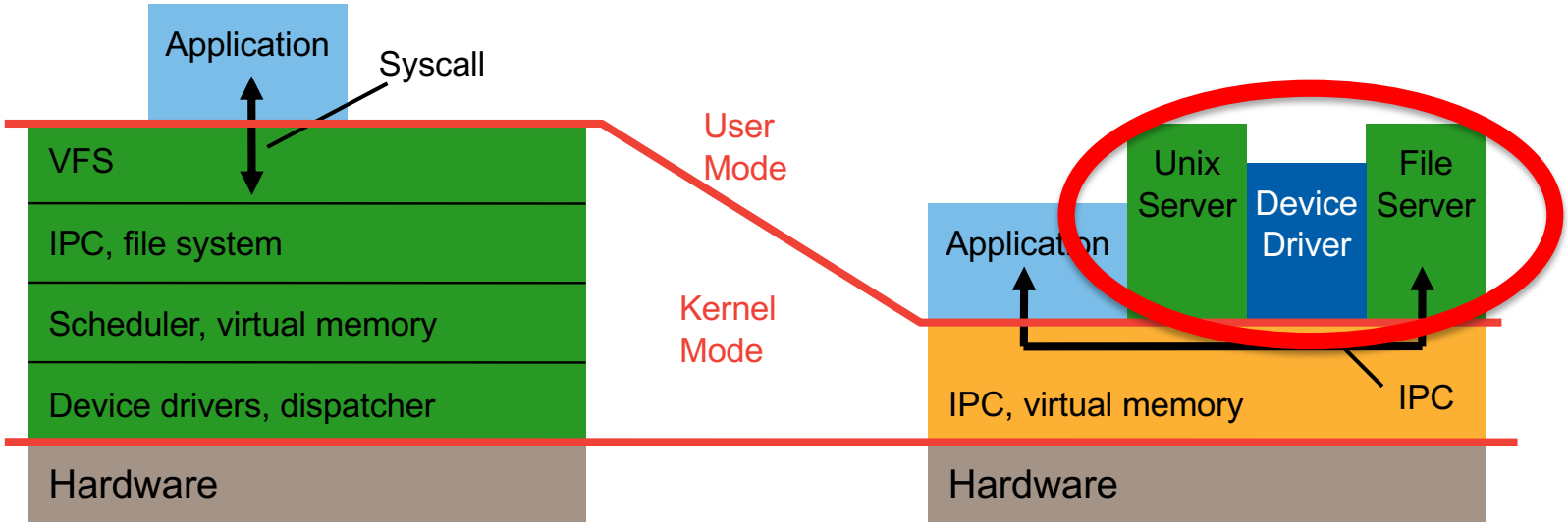
Advantages:

- Easy to implement, port?
  - in practice limited architecture-specific micro-optimization
- Less code to optimise
- Hopefully enables a minimal *trusted computing base* (TCB)
  - small attack surface, fewer failure modes
- Easier debug, maybe even *prove* correct?

Challenges:

- API design: generality with small code base
- Kernel design and implementation for high performance
  - … and correctness!

Consequence of Minimality: User-level Services



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

Microkernel Principles: Policy Freedom

A true microkernel must be free of policy!

Policies limit

- May be good for many cases, but always bad for some
- Example: disk pre-fetching

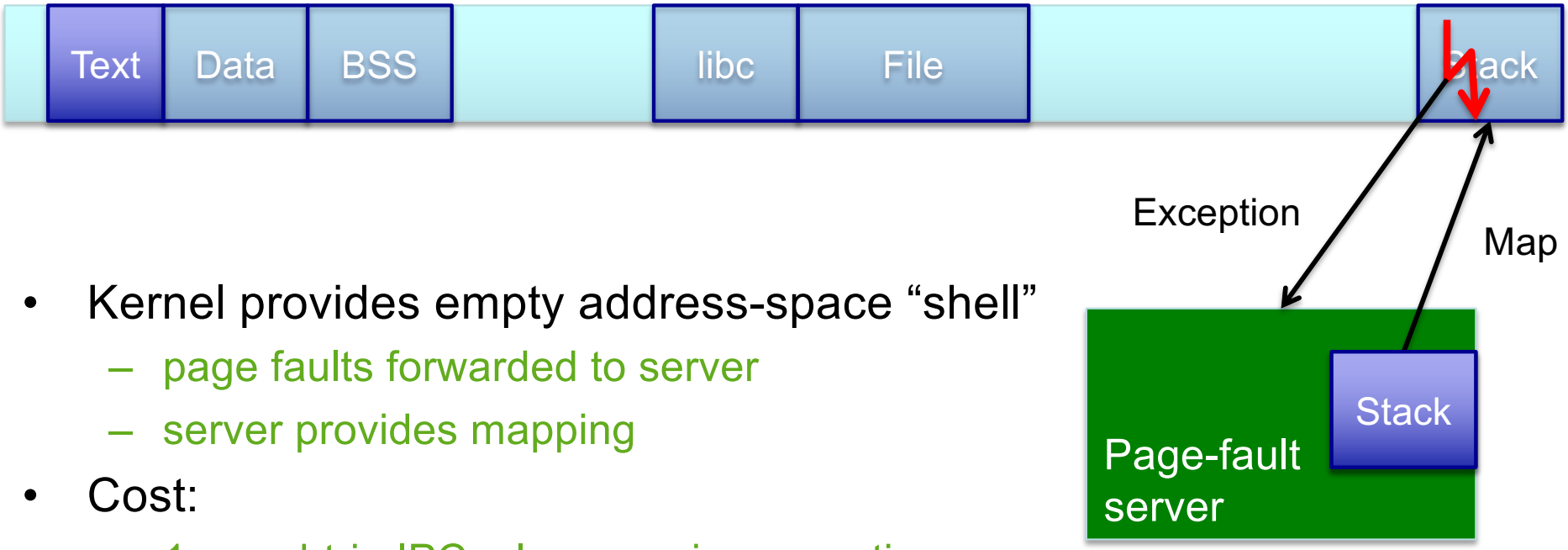
"General" policies lead to bloat

- Implementing combination of policies
- Try to determine most appropriate one at run-time

Policy Example: Address-Space Layout

- Kernel determines layout, knows executable format, allocates stack
  - limits ability to import from other OSes
  - cannot change layout
    - small non-overlapping address spaces beneficial on some archs
  - kernel loads apps, sets up mappings, allocates stack
    - requires file system in kernel or interfaced to kernel
    - bookkeeping for revokation & resource management
    - heavyweight processes
  - memory-mapped file API

Policy-Free Address-Space Management



- Kernel provides empty address-space "shell"
  - page faults forwarded to server
  - server provides mapping
- Cost:
  - 1 round-trip IPC, plus mapping operation
    - mapping may be side effect of IPC
    - kernel may expose data structure
  - kernel mechanism for forwarding page-fault exception
- "External pagers" first appeared in Mach [Rashid et al, '88]
  - … but were optional

What Mechanisms?

- Fundamentally, the microkernel must abstract
  - *Physical memory*
  - *CPU*
  - *Interrupts/Exceptions*
- Unfettered access to any of these bypasses security
  - No further abstraction needed for devices
    - memory-mapping device registers and interrupt abstraction suffices
    - …but some generalised memory abstraction needed for I/O space
- Above isolates execution units, hence microkernel must also provide
  - *Communication* (traditionally referred to as *IPC*)
  - *Synchronization*

What Mechanisms?

Traditional hypervisor vs microkernel abstractions

Resource	Hypervisor	Microkernel
Memory	Virtual MMU (vMMU)	Address space
CPU	Virtual CPU (vCPU)	Thread or scheduler activation
Interrupt	Virtual IRQ (vIRQ)	IPC message or signal
Communication	Virtual NIC	Message-passing IPC
Synchronization	Virtual IRQ	IPC message

Issues of 2G L4 Kernels

- L4 solved performance issue [Härtig et al, SOSP'97]
  … but left a number of security issues unsolved
- Problem: ad-hoc approach to protection and resource management
  - Global thread name space ⇒ covert channels
  - Threads as IPC targets ⇒ insufficient encapsulation
  - Single kernel memory pool ⇒ DoS attacks
  - Insufficient delegation of authority ⇒ limited flexibility, performance

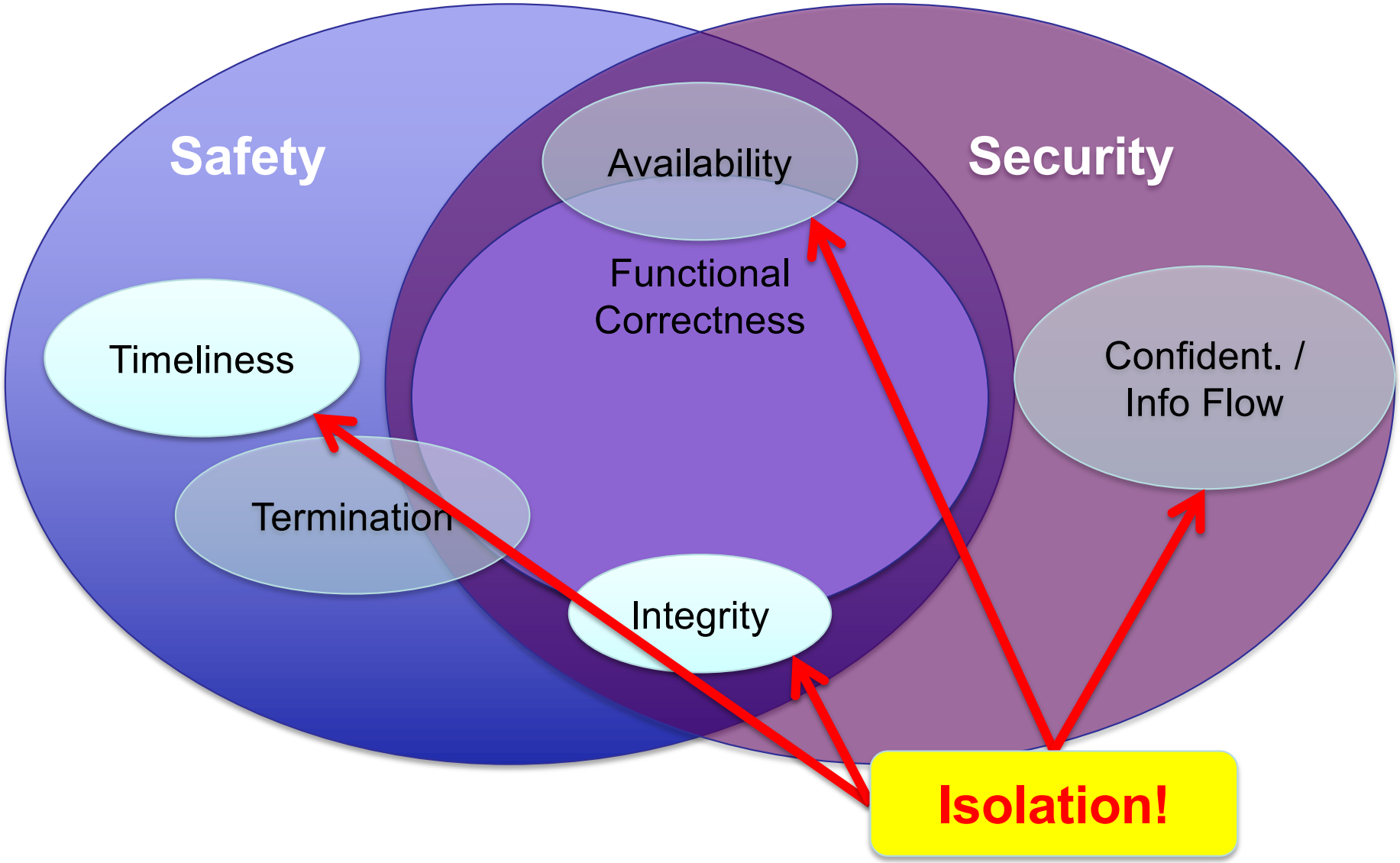
- Addressed by seL4
  - Designed to support safety- and security-critical systems

Agenda

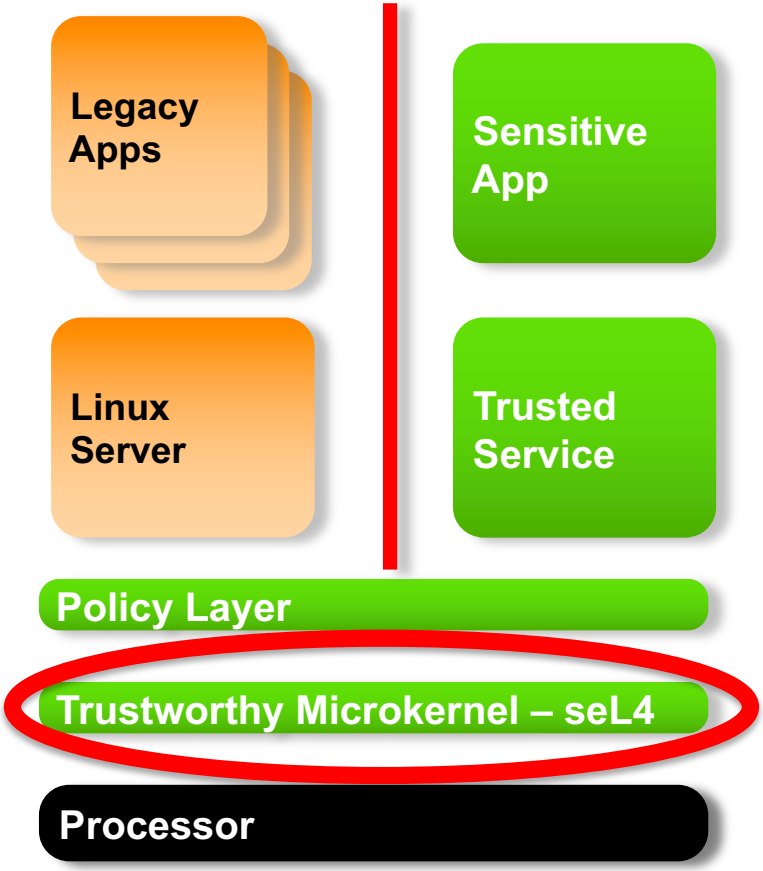
- Motivation
- What is a microkernel, and what is L4?
- **seL4 – designed for trustworthiness**
- Establishing trustworthiness
- From kernel to system
- Sample system 1: Secure access controller
- Sample system 2: RapiLog

Requirements for Trustworthy Systems

NICTA



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

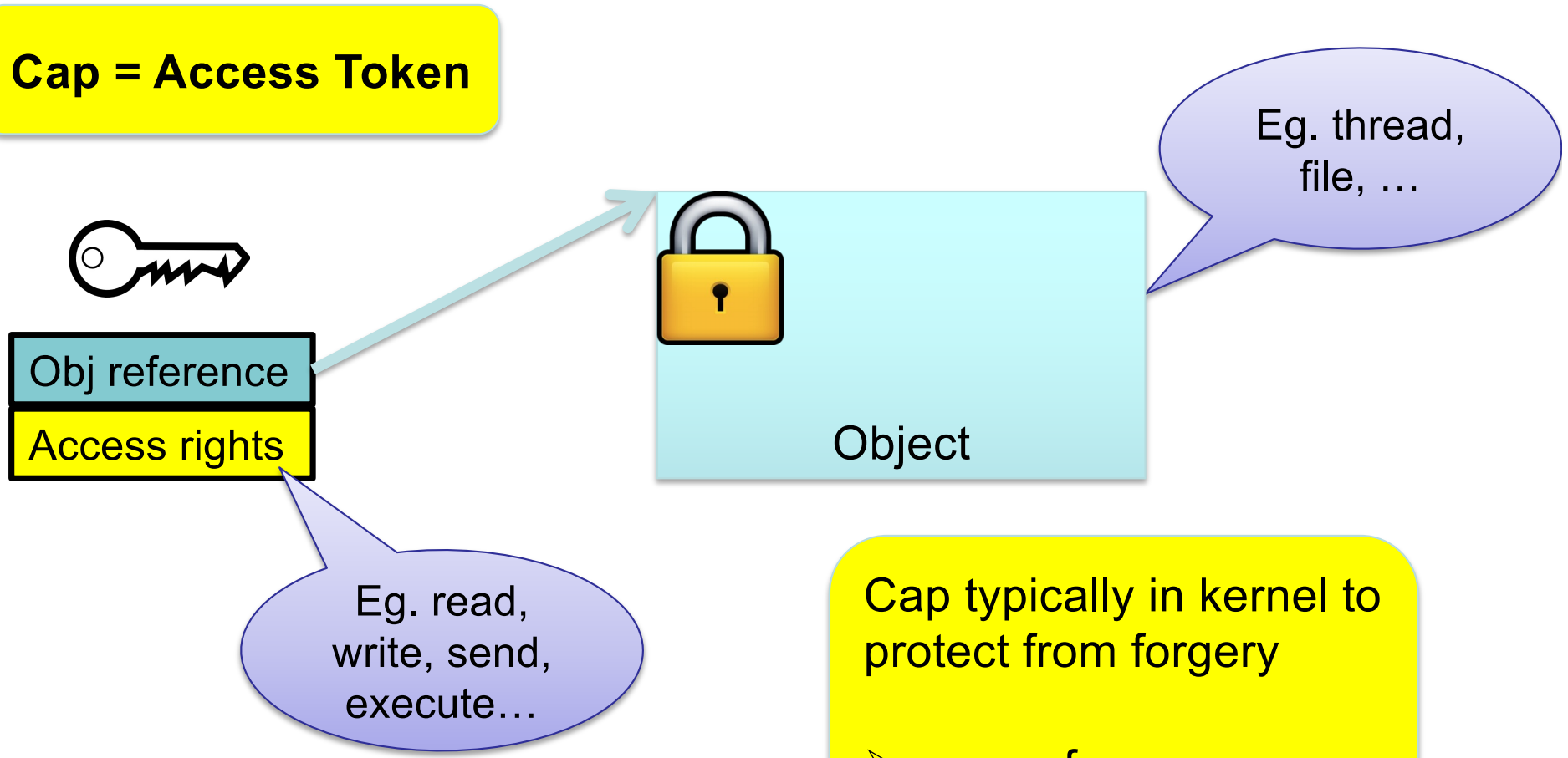
4. No concurrency in the kernel
   - Interrupts never enabled in kernel
   - Interruption points to bound latencies
   - Clustered multikernel design for multicores

Verification

What are Capabilities?

NICTA

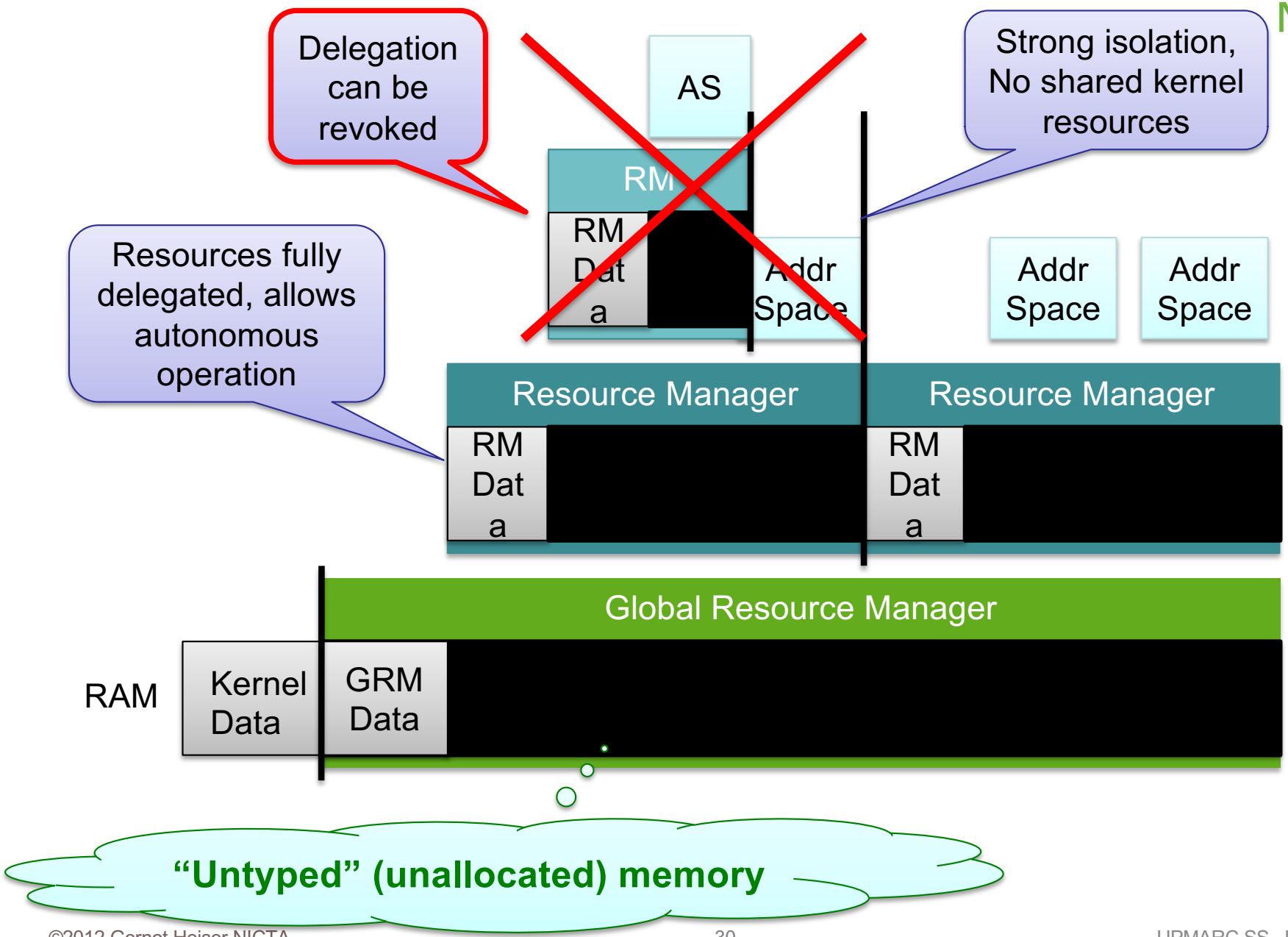
Cap = Access Token



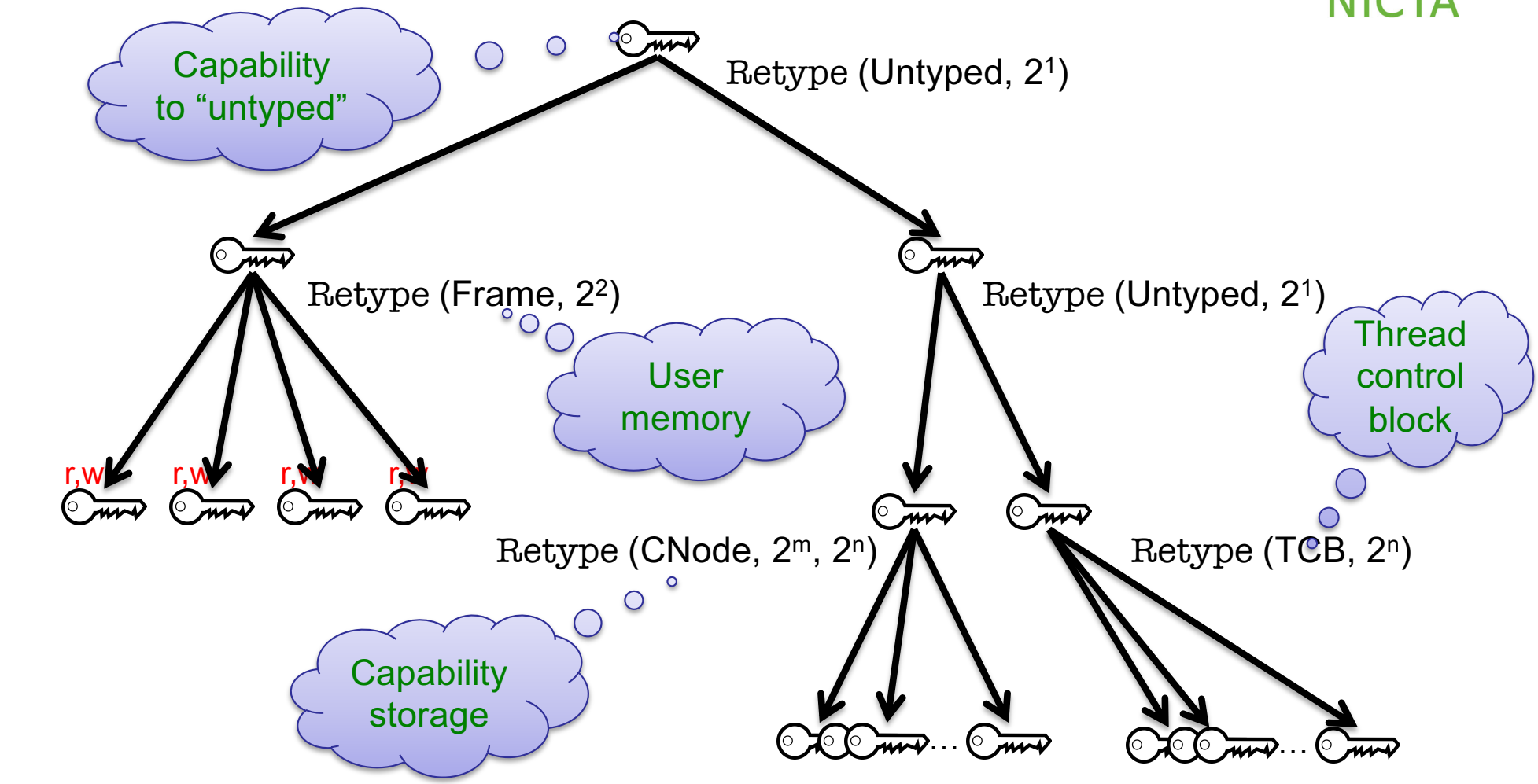
Cap typically in kernel to protect from forgery

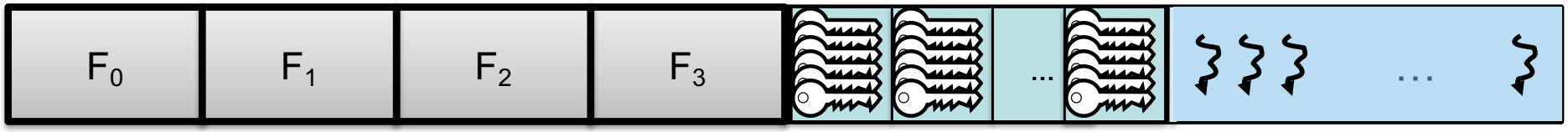
- user references cap through handle

seL4 User-Level Memory Management

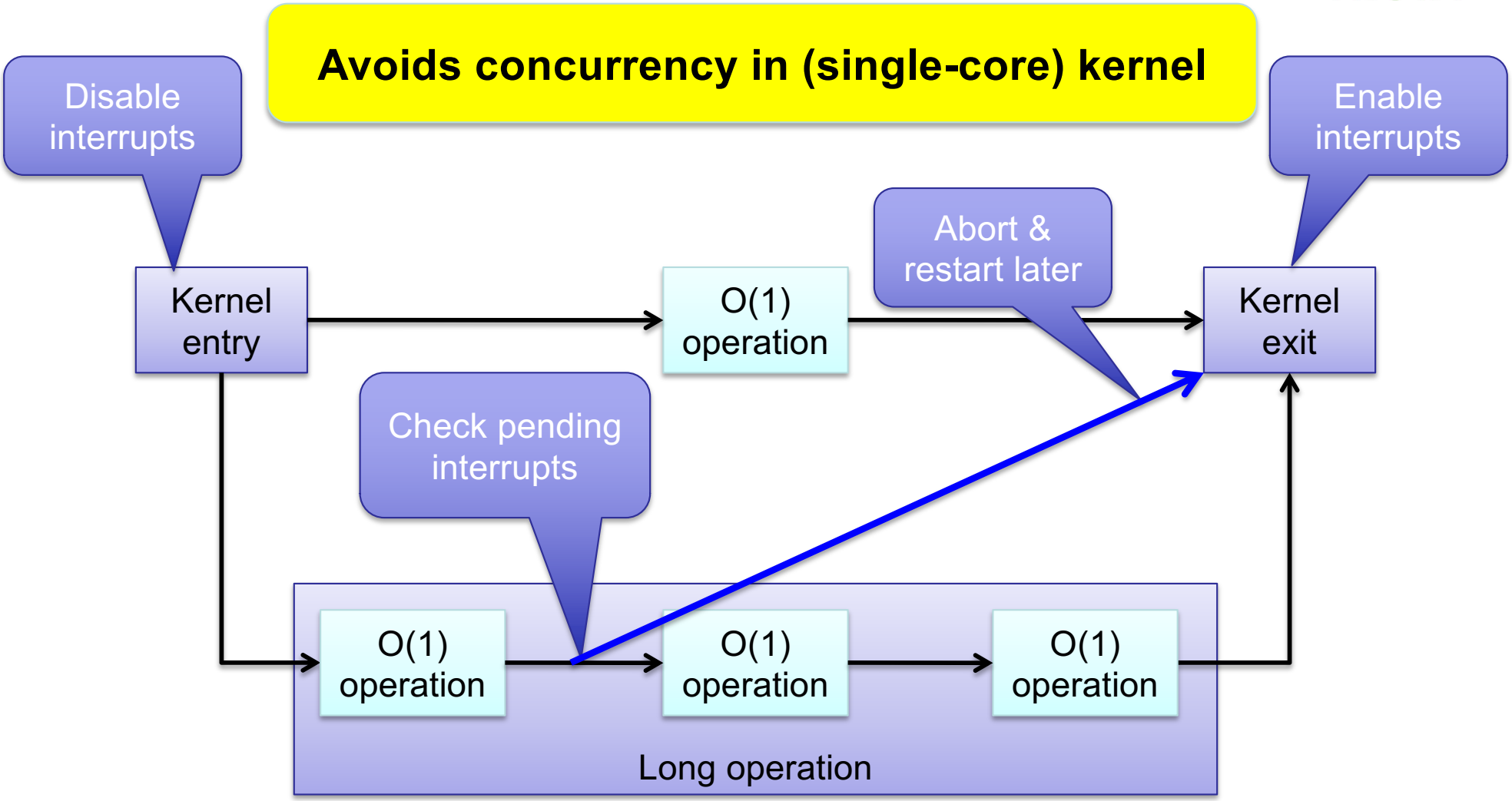


seL4 Memory Management Mechanics: Retype

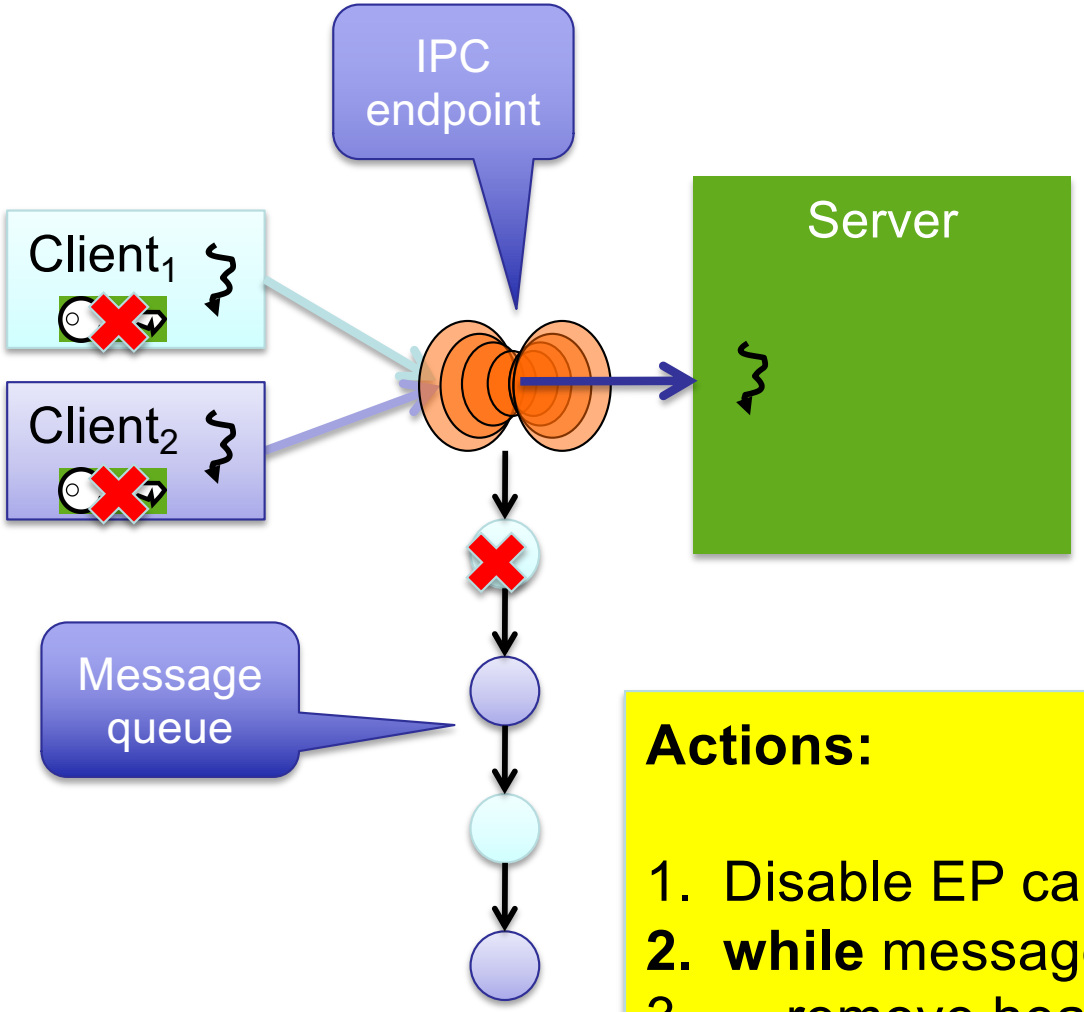




Incremental Consistency



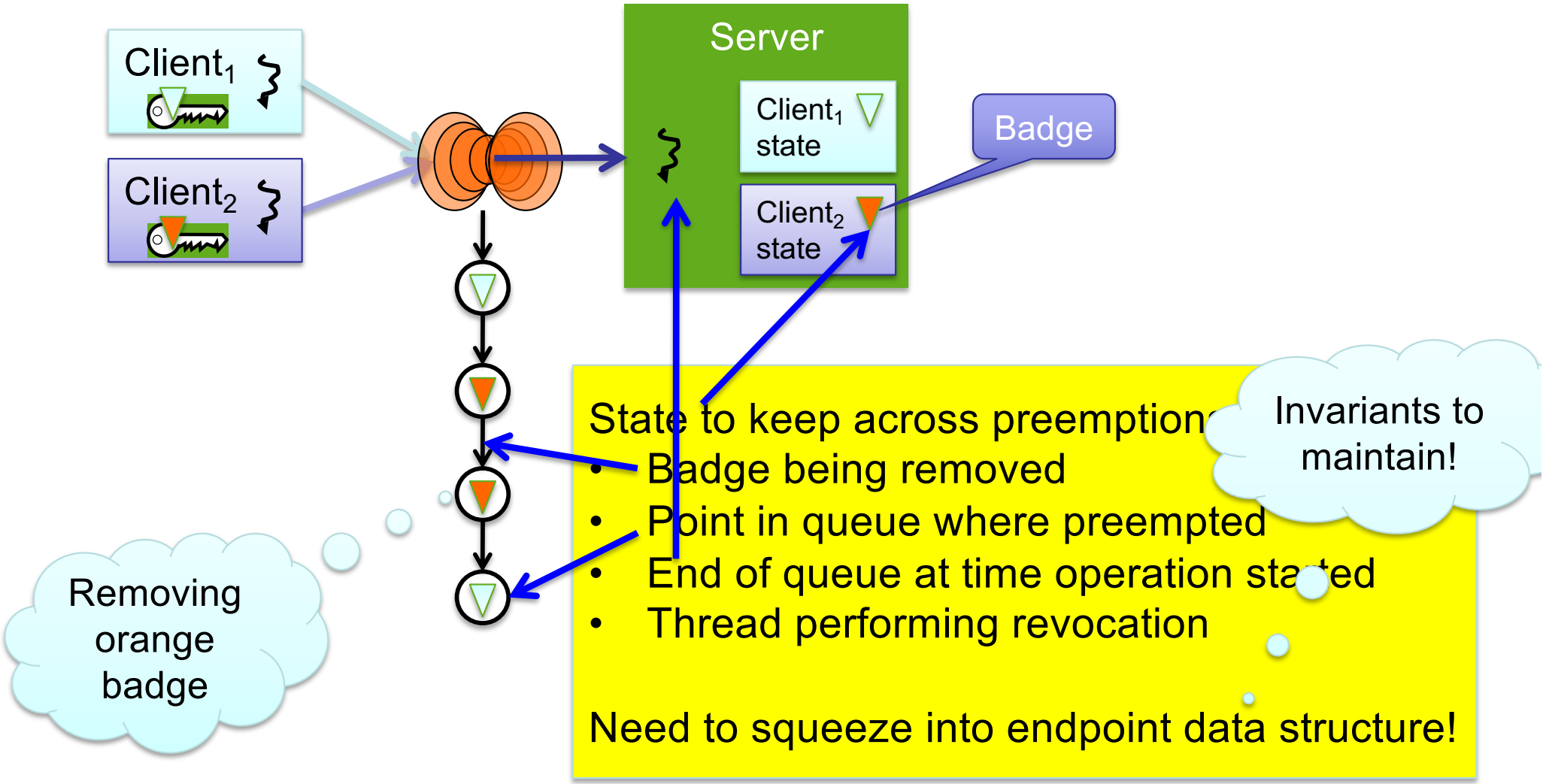
Example: Destroying IPC Endpoint



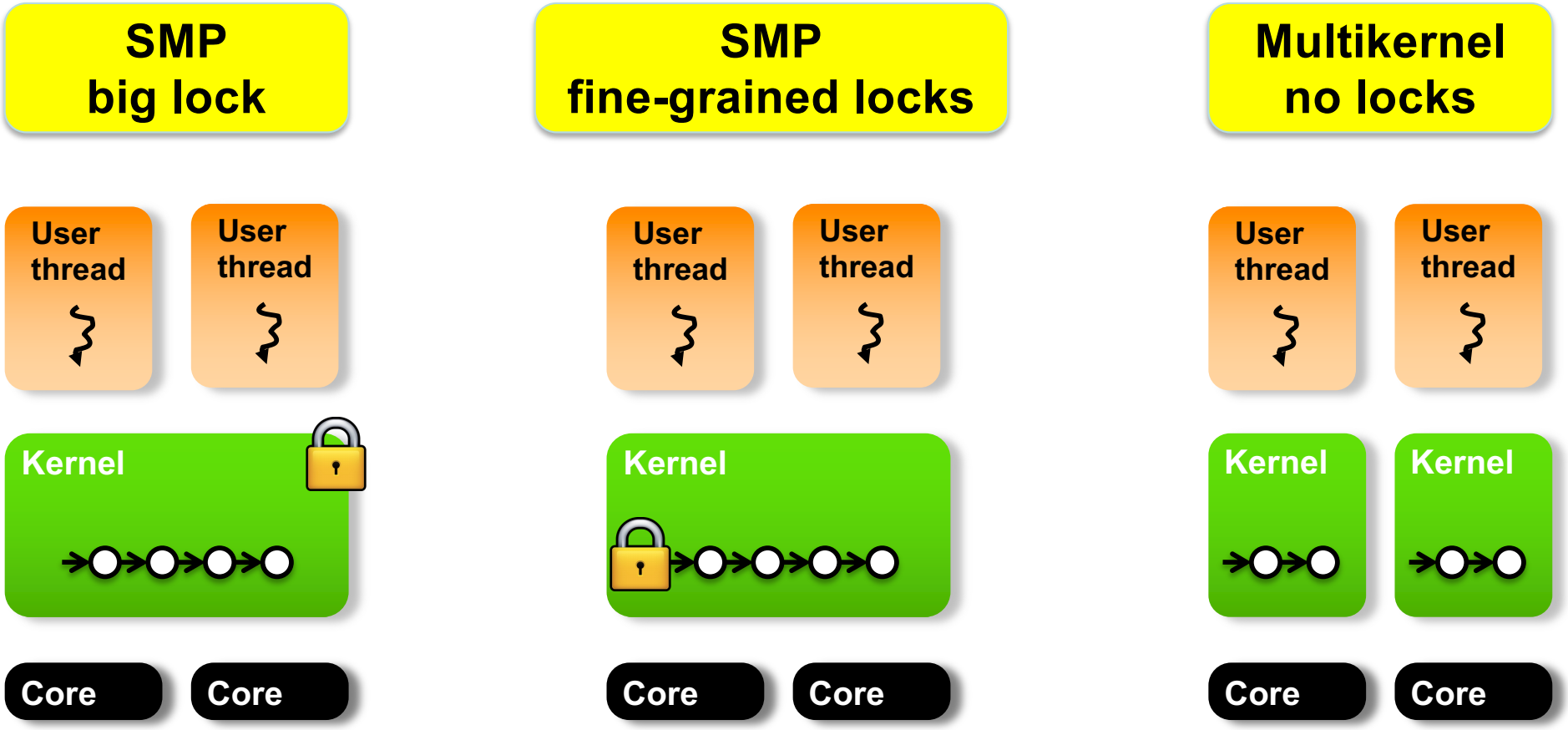
Actions:

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**

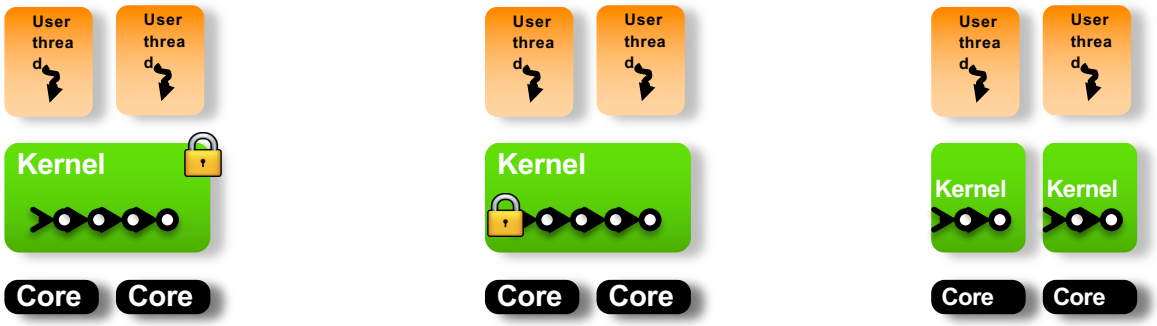
Difficult Example: Revoking IPC "Badge"



Approaches for Multicore Kernels

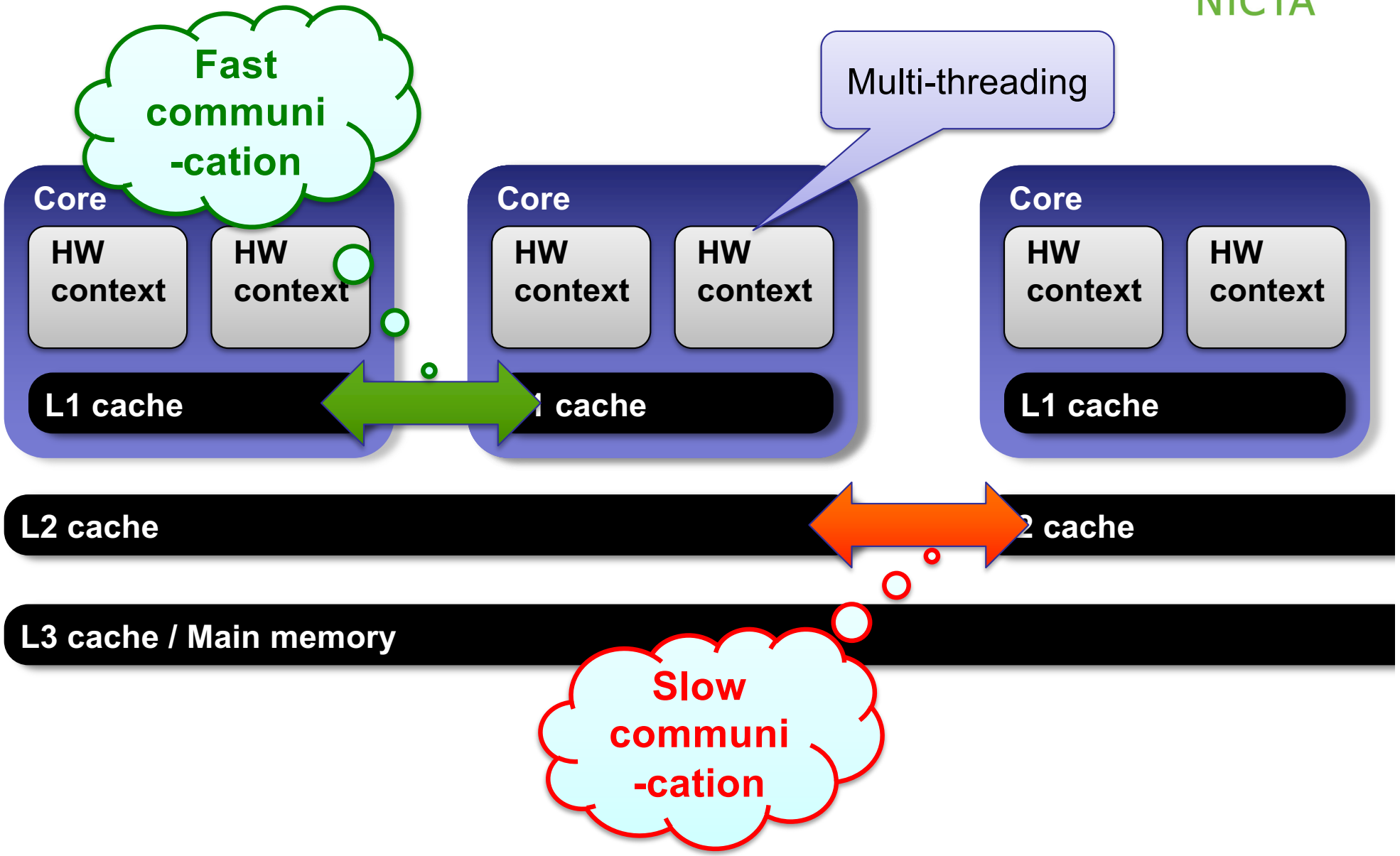


Multicore Kernel Trade-Offs

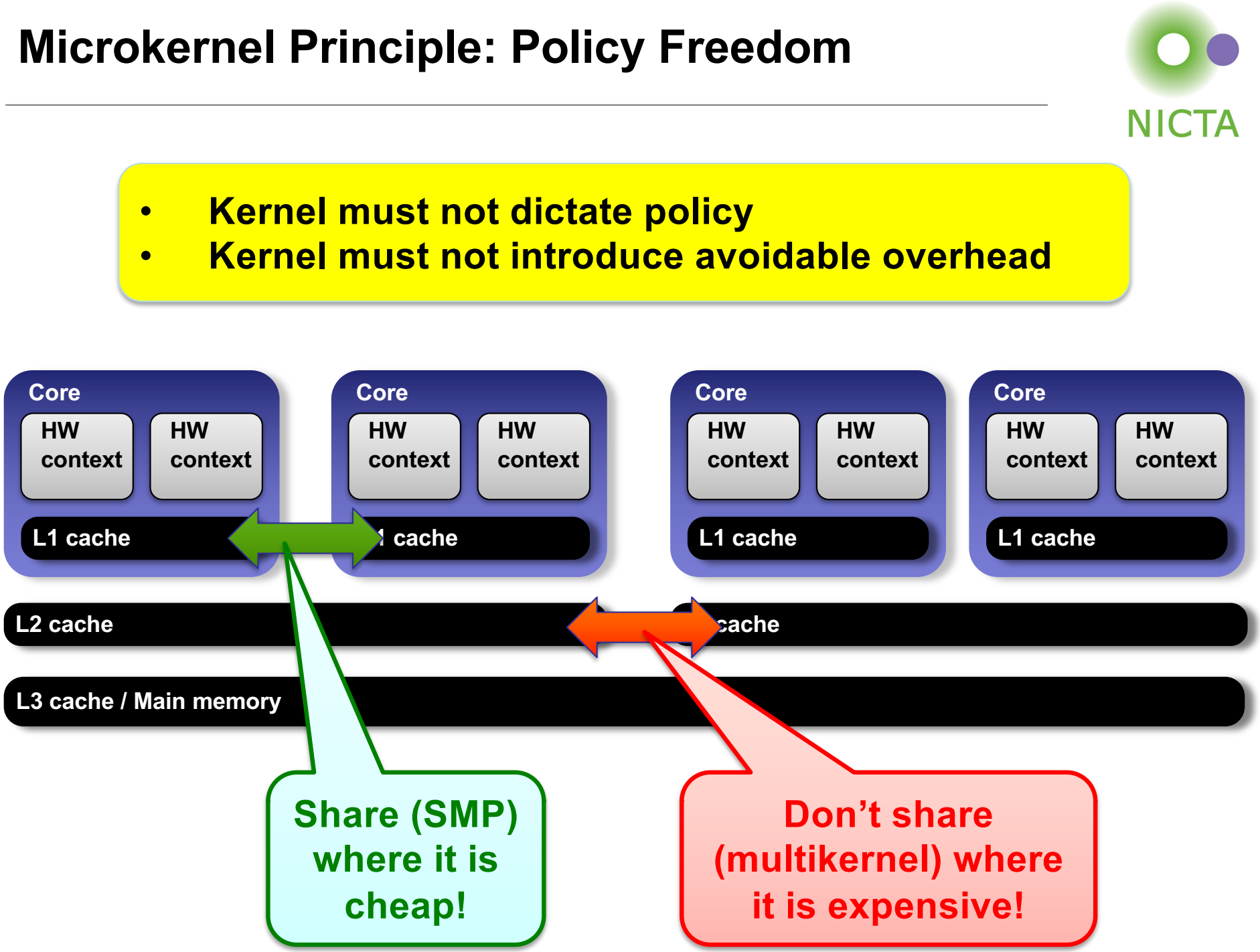


Property	Big Lock	Fine-grained Locking	Multikernel
Data structures	shared	shared	distributed
Scalability	poor	good	excellent
Concurrency in kernel	zero	high	zero
Kernel complexity	low	high	low
Resource management	centralised	centralised	distributed

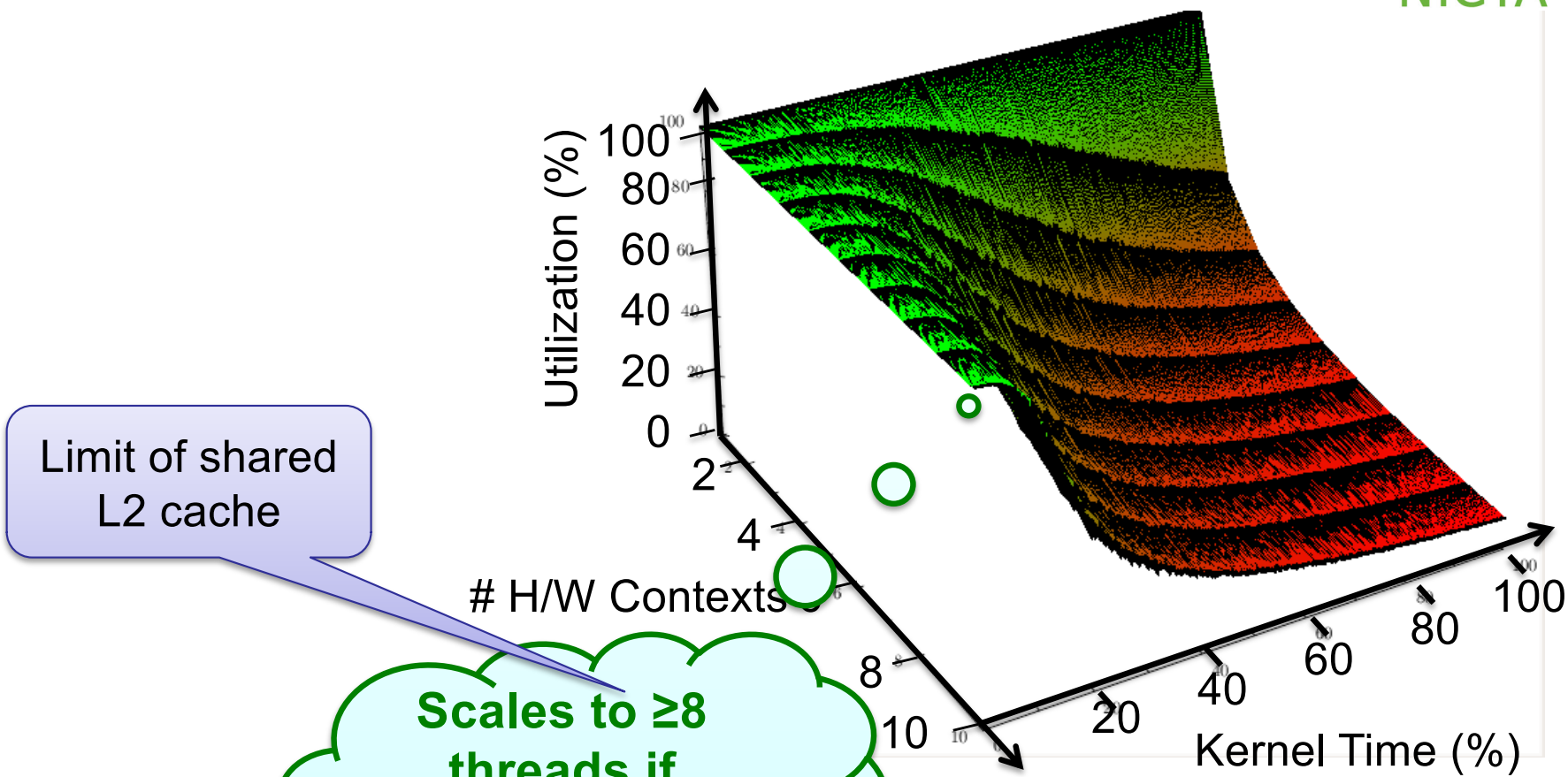
Reality of Multicore is NUMA!



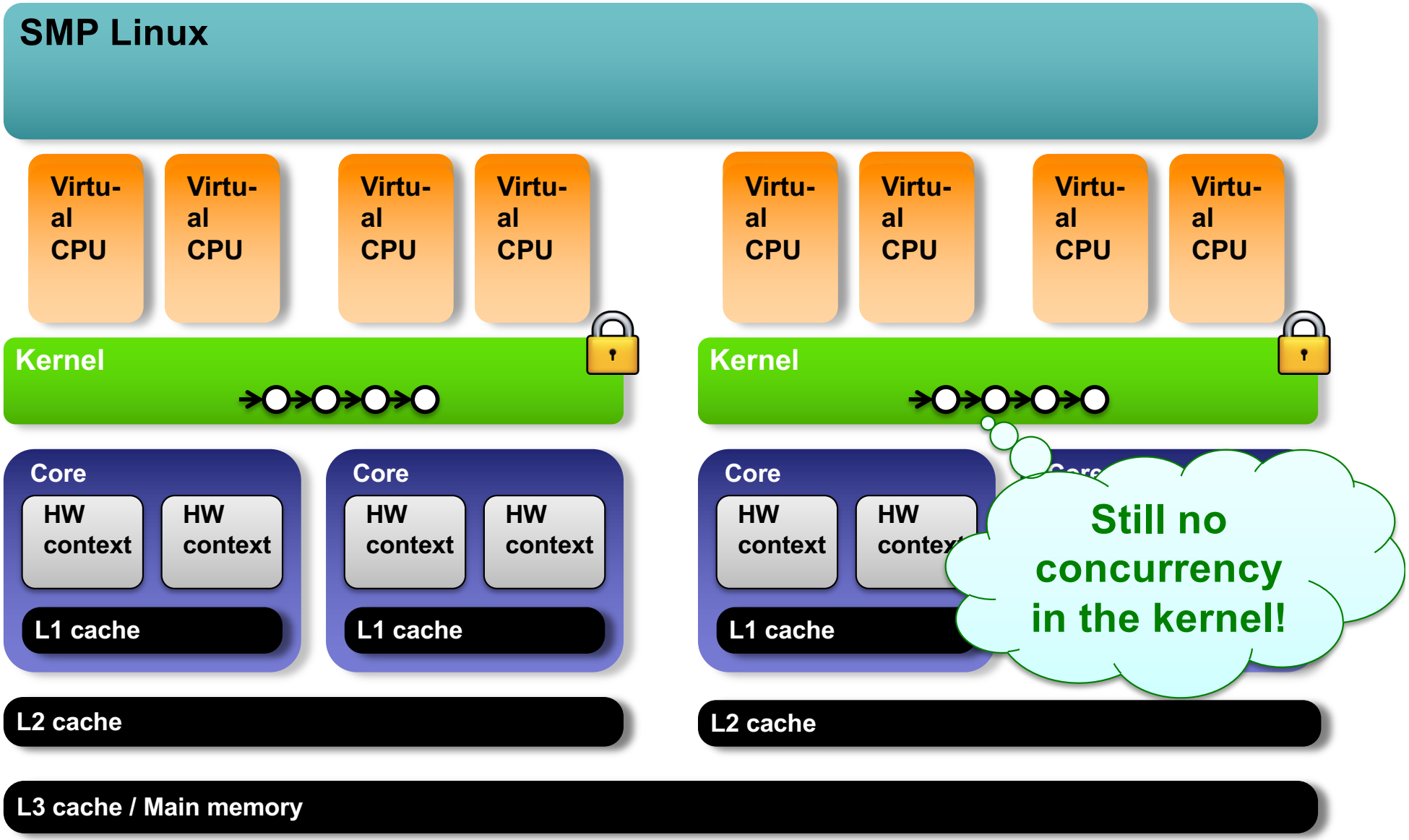
Microkernel Principle: Policy Freedom



Performance of Big Kernel Lock



Resulting Design: Clustered Multikernel

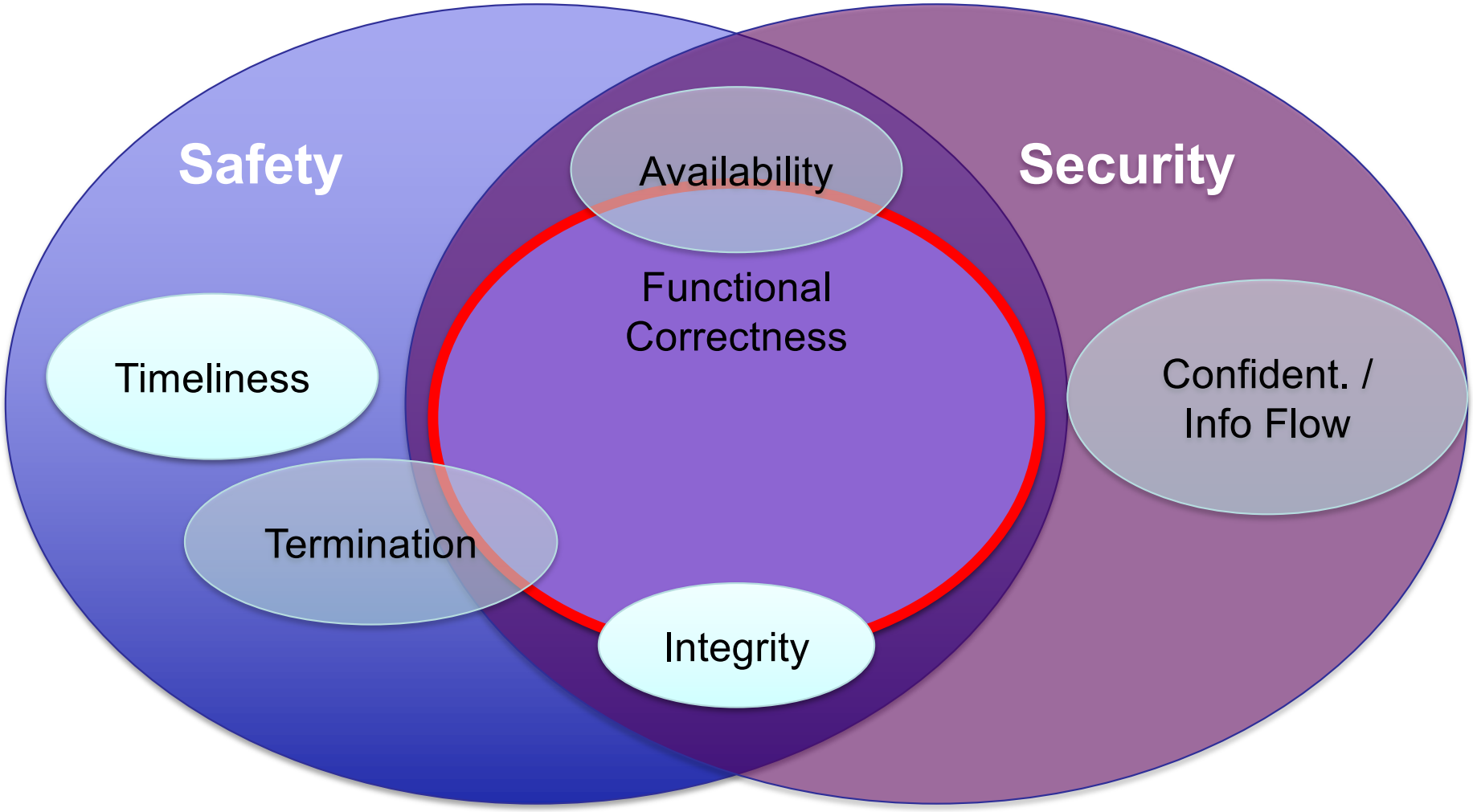


Agenda

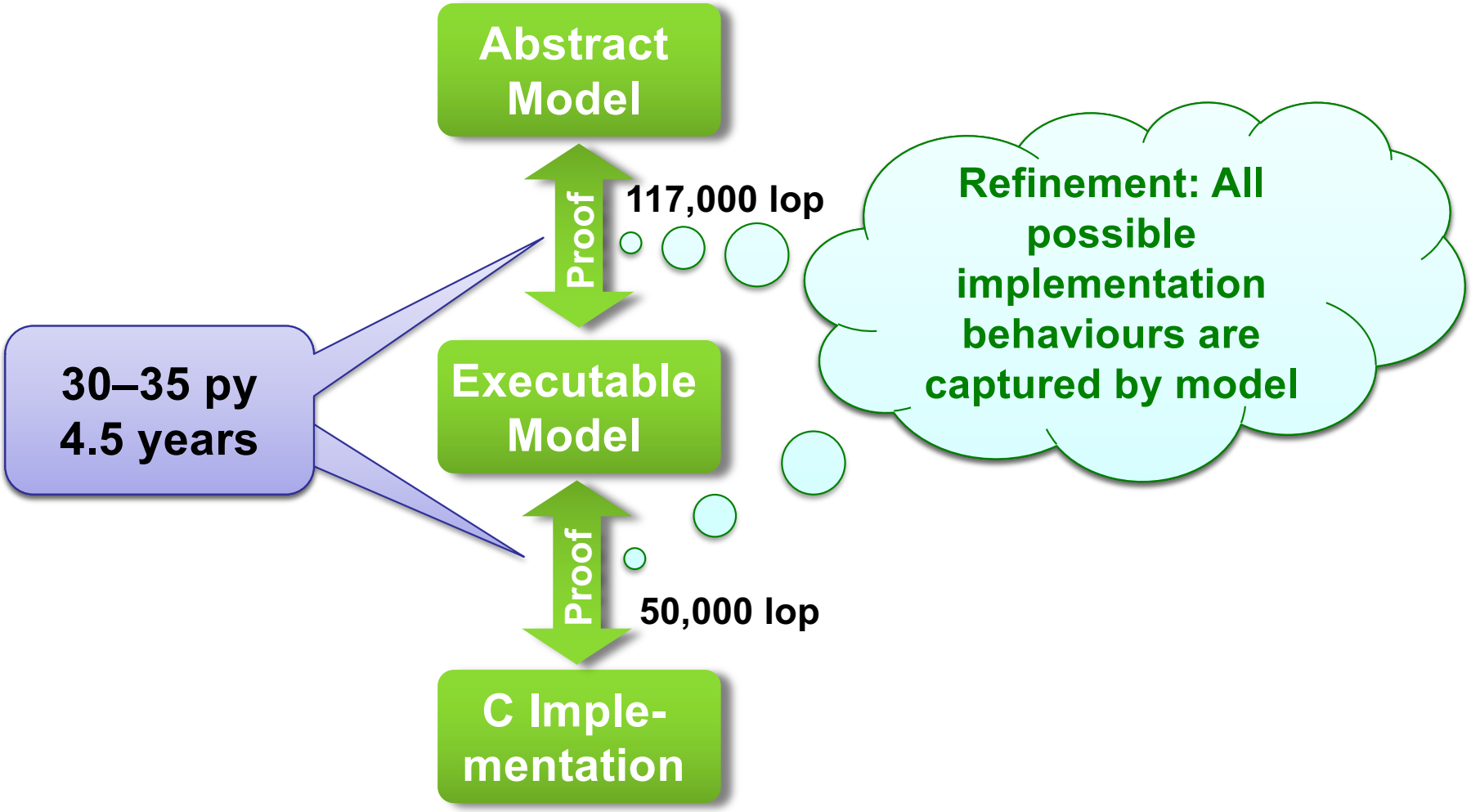
- Motivation
- What is a microkernel, and what is L4?
- seL4 – designed for trustworthiness
- **Establishing trustworthiness**
- From kernel to system
- Sample system 1: Secure access controller
- Sample system 2: RapiLog

seL4 as Basis for Trustworthy Systems

NICTA



Proving Functional Correctness



Correctness

NICTA

```
datatype
  rights = Read
         | Write
         | Grant
         | Create

record cap =
  entity :: entity_id
```

```
constdefs
  schedule :: "unit s_monad"
  "schedule ≡ do
```

```
lemma isolation:
  "⟦sane s;
    s' ∈ execute cmds s;
    isEntityOf s eS;
    sEntityOf s e;
    ntity c = e;
    :> subSysCaps s eS⟧
   ⟹ c :> subSysCaps s' eS"
```

```
schedule :: Kernel ()
schedule = do
```

```
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;
```

ecification

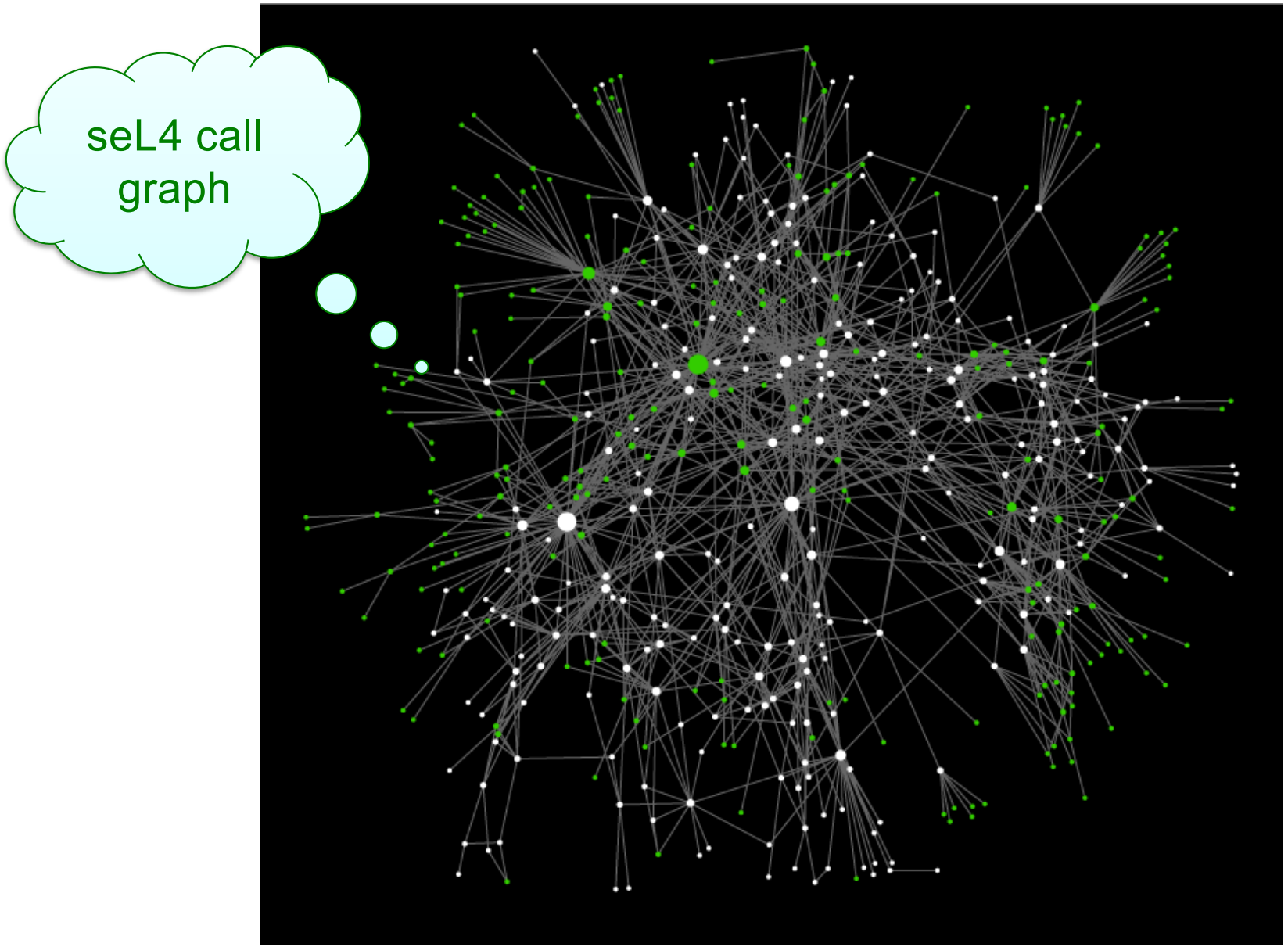
L)

d
Thread

be (5,700)

mance Implementation
(C/asm)
odel

Why So Long for 9,000 LOC?



Costs Breakdown

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

seL4 Formal Verification Summary

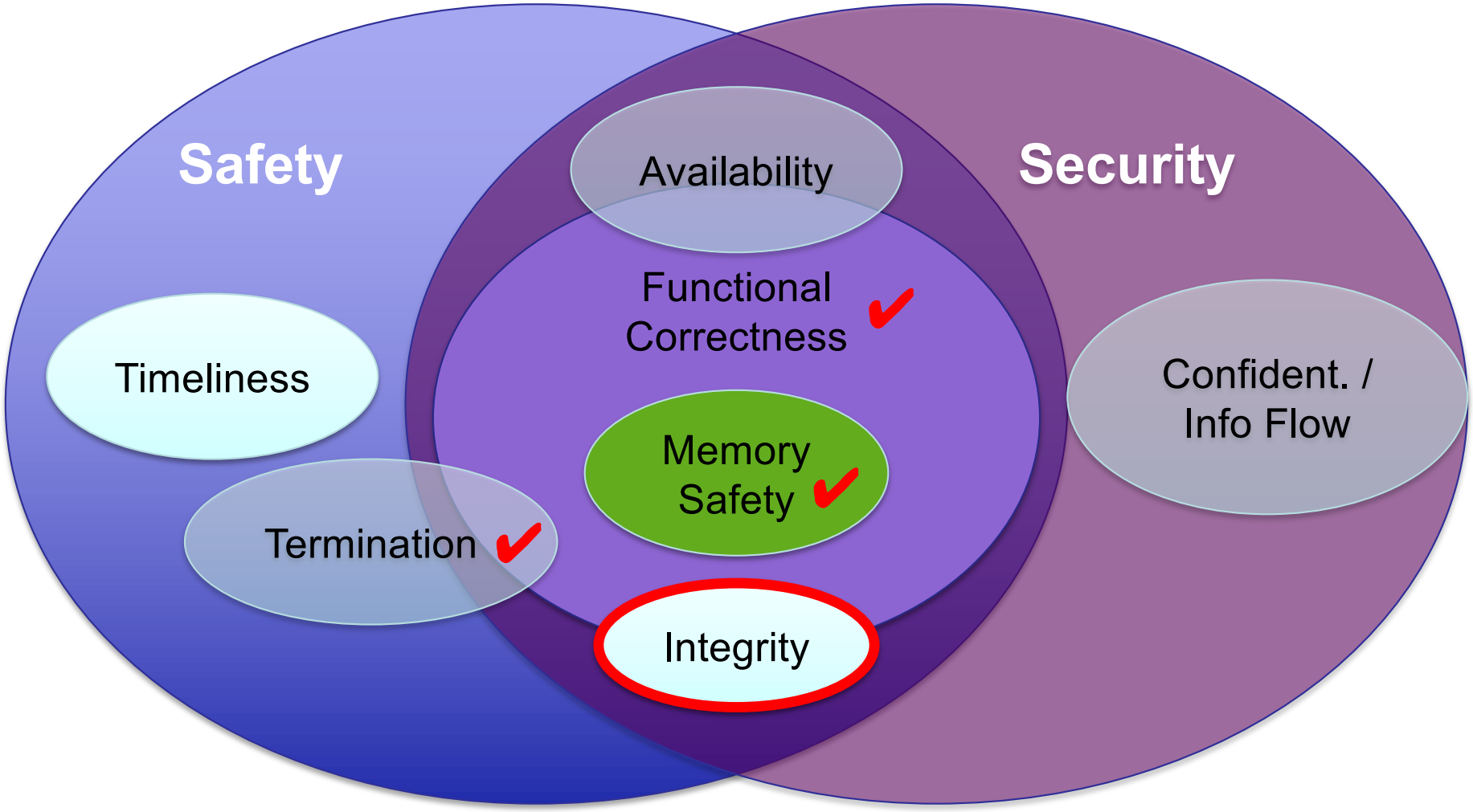
Kinds of properties proved

- 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

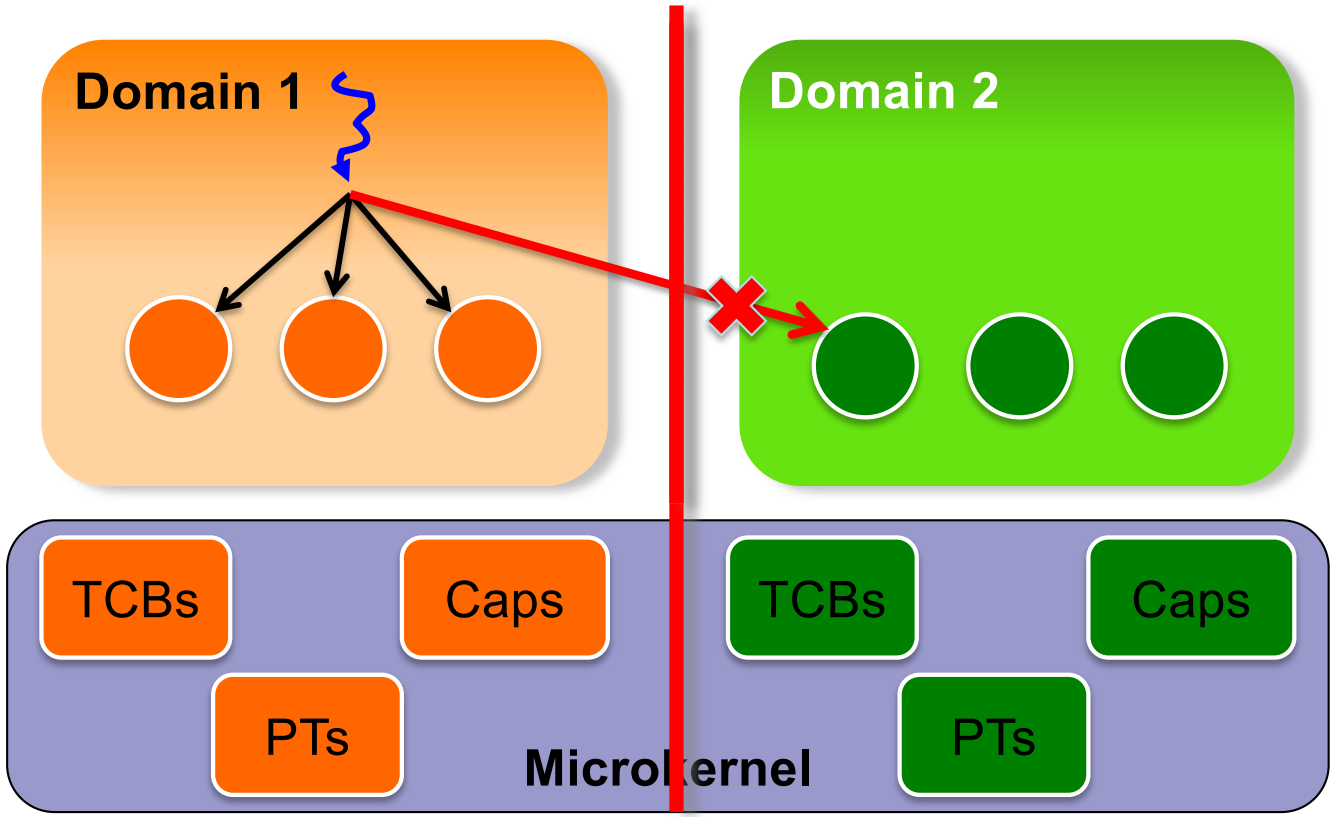
Can prove further poperties on abstract level!

seL4 as Basis for Trustworthy Systems

NICTA



Integrity: Limiting Write Access

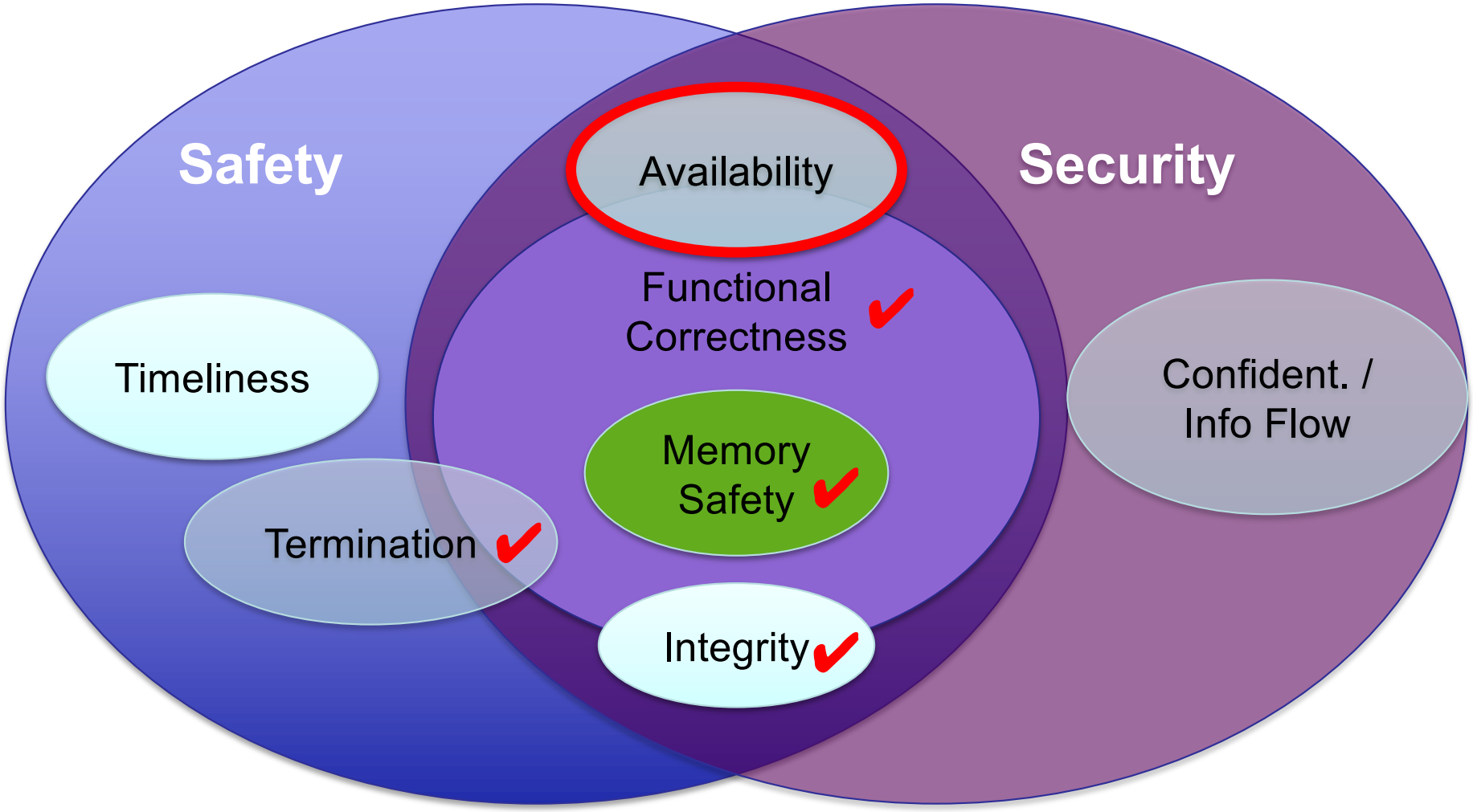


To prove:

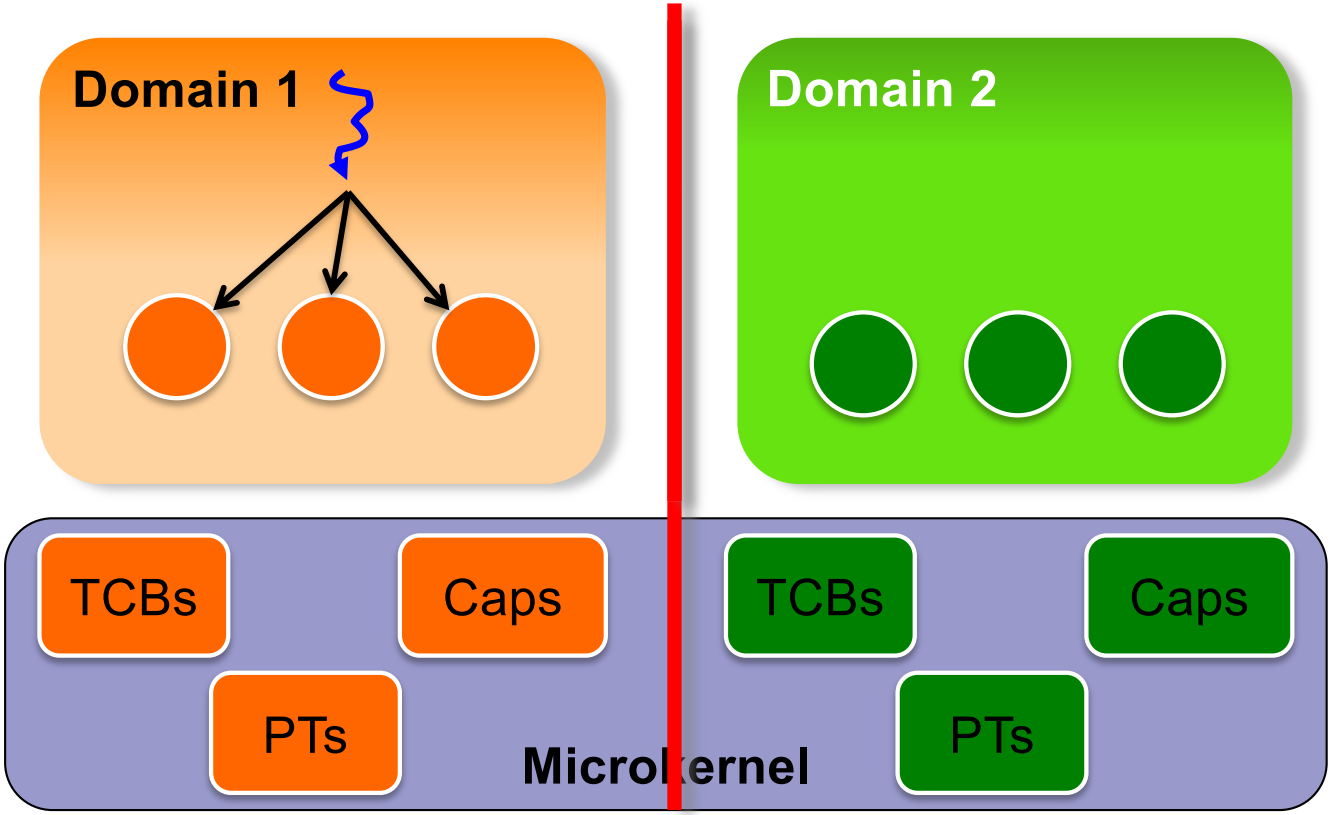
- Domain-1 doesn't have write *capabilities* to Domain-2 objects
  ⇒ no action of Domain-1 agents will modify Domain-2 state
- Specifically, *kernel does not modify on Domain-1's behalf!*
  - Prove kernel only allows write upon capability presentation

seL4 as Basis for Trustworthy Systems

NICTA



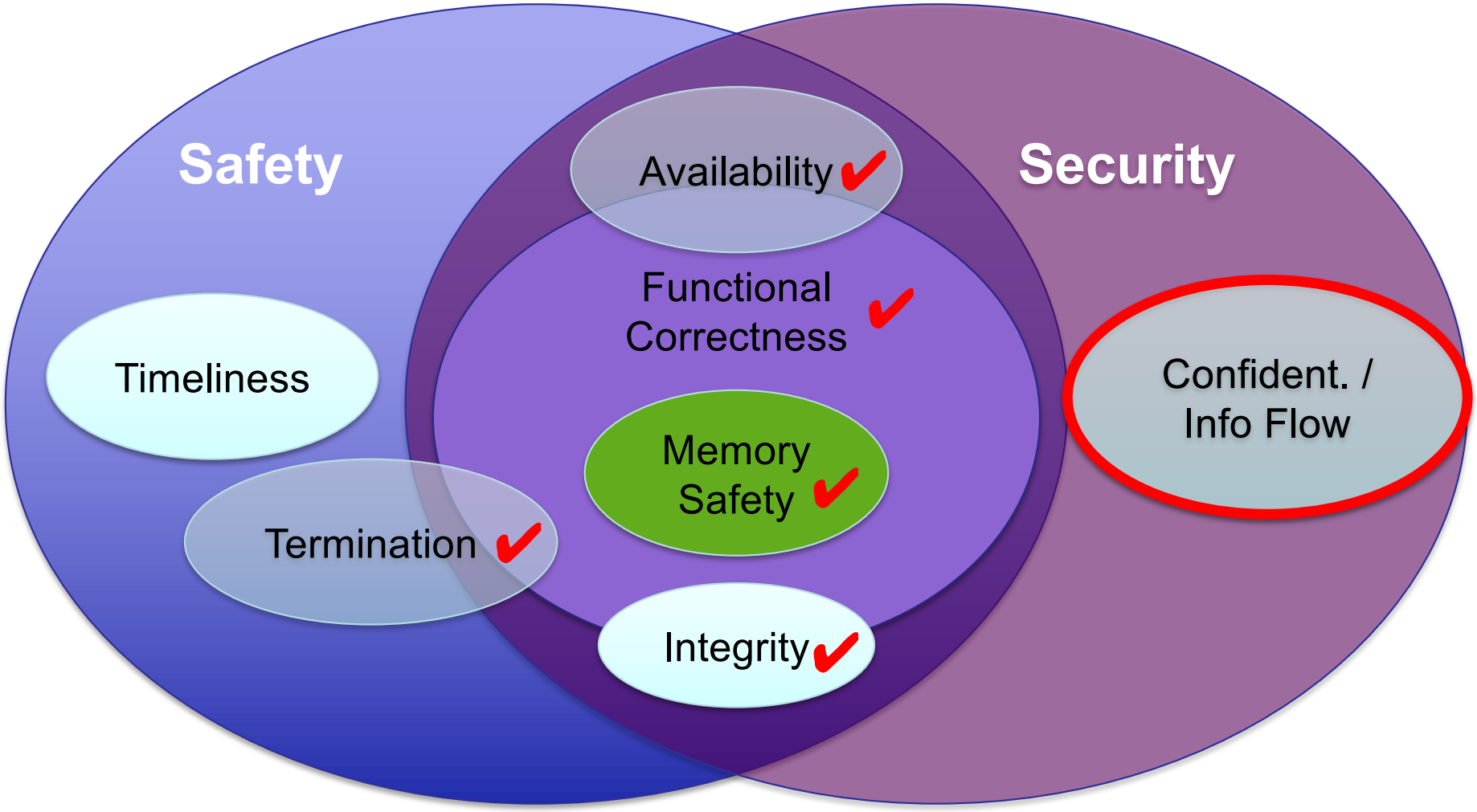
Availability: Ensuring Resource Access



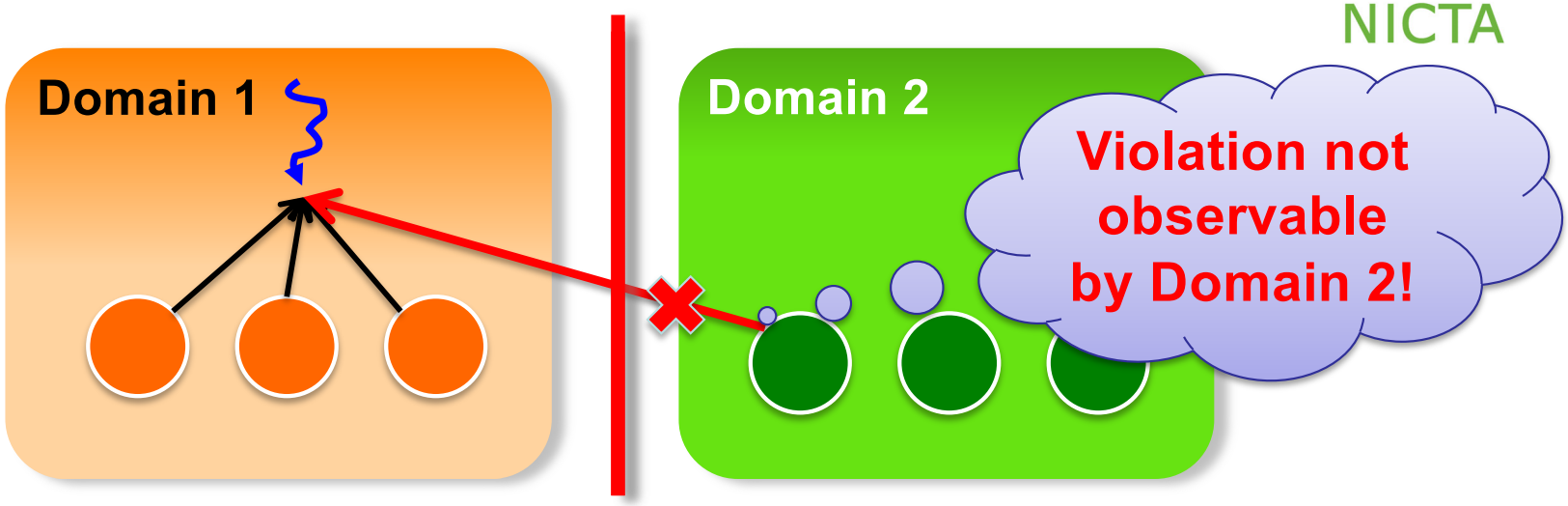
- Strict separation of kernel resources
  ⇒ agent cannot deny access to another domain's resources

seL4 as Basis for Trustworthy Systems

NICTA



Confidentiality: Limiting Read Accesses



To prove:

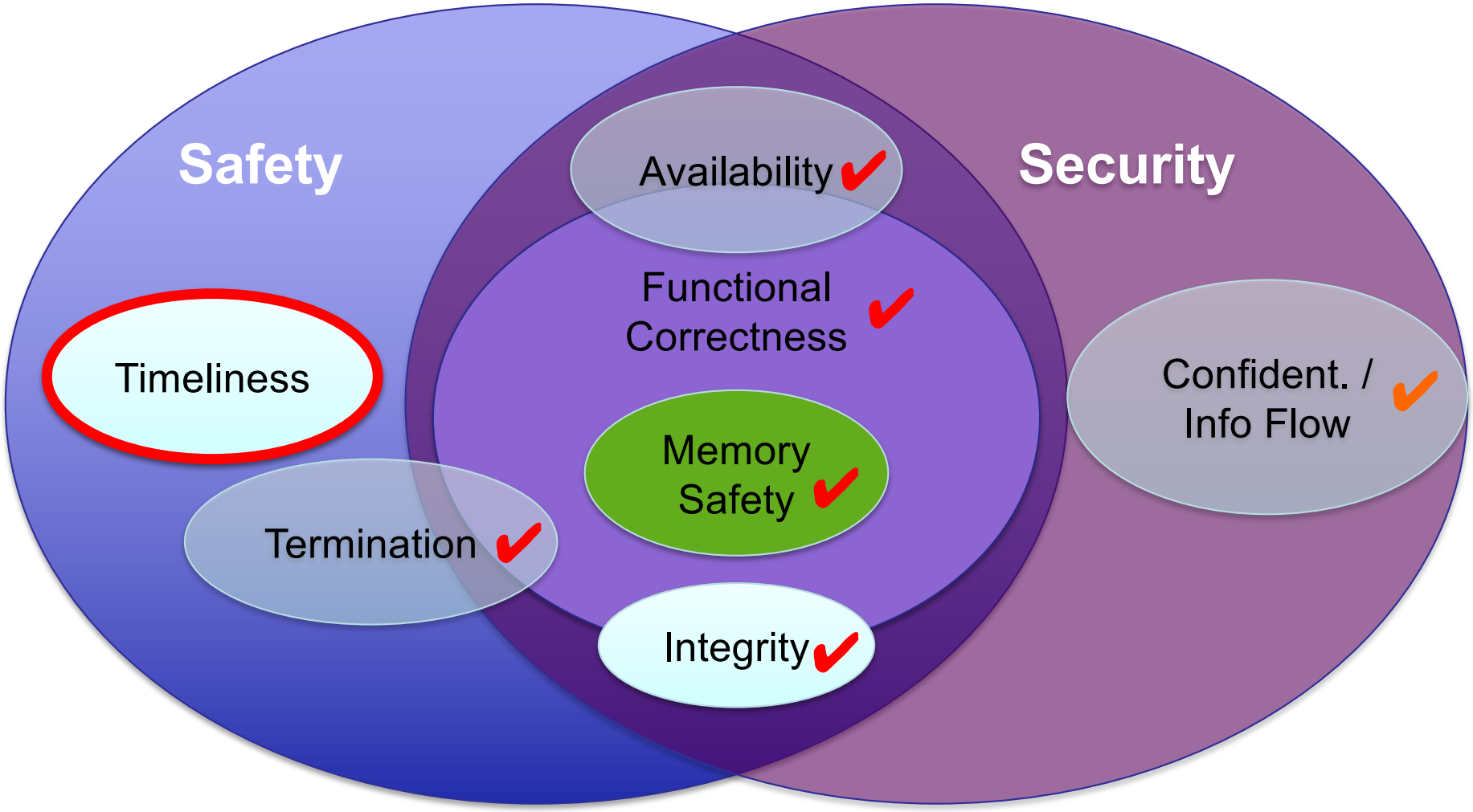
- Domain-1 doesn't have read capabilities to Domain-2 objects
  ⇒ no action of any agents will reveal Domain-2 state to Domain-1

Non-interference proof in progress:

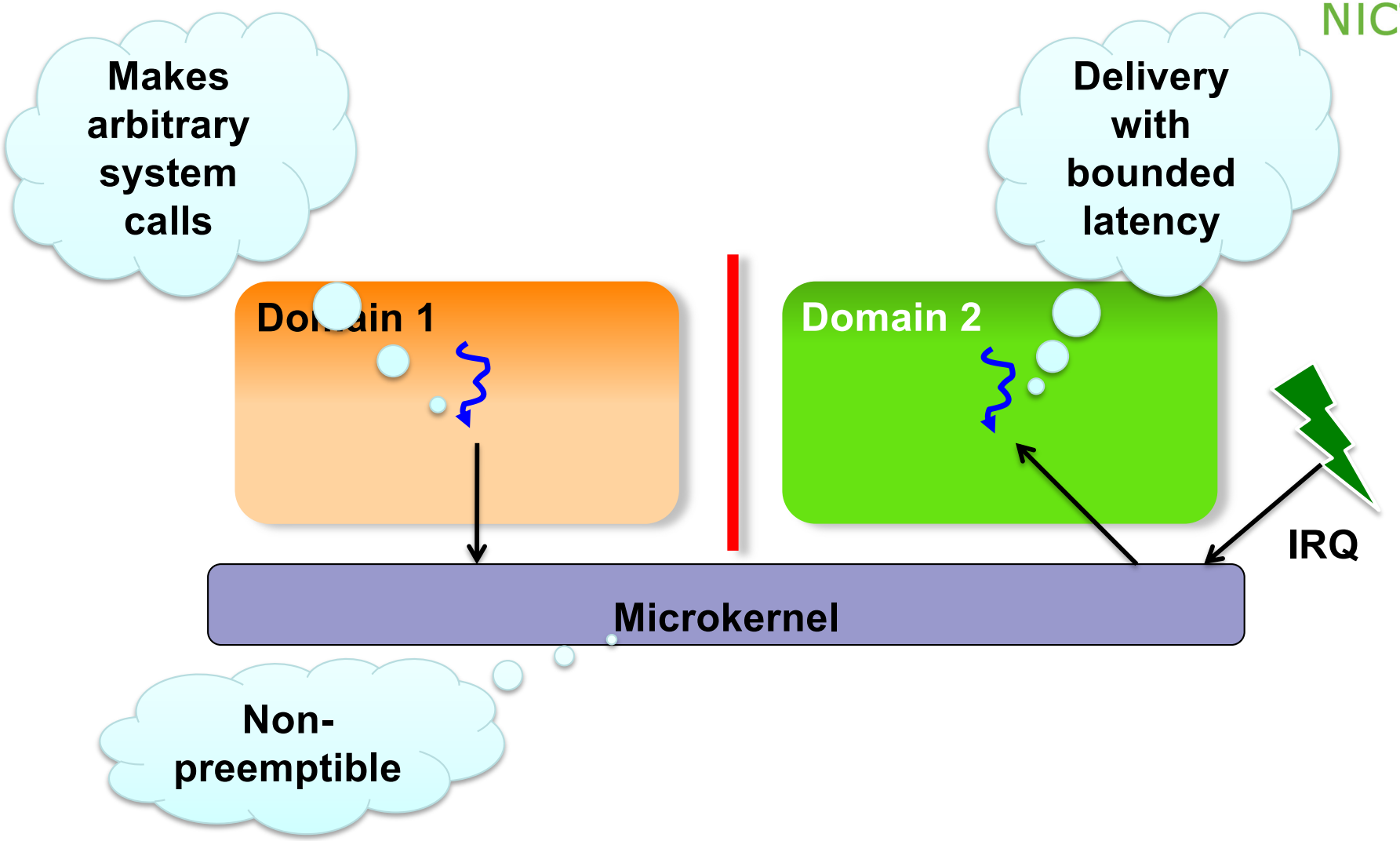
- Evolution of Domain 1 does not depend on Domain-2 state
- Presently cover only overt information flow

seL4 as Basis for Trustworthy Systems

NICTA



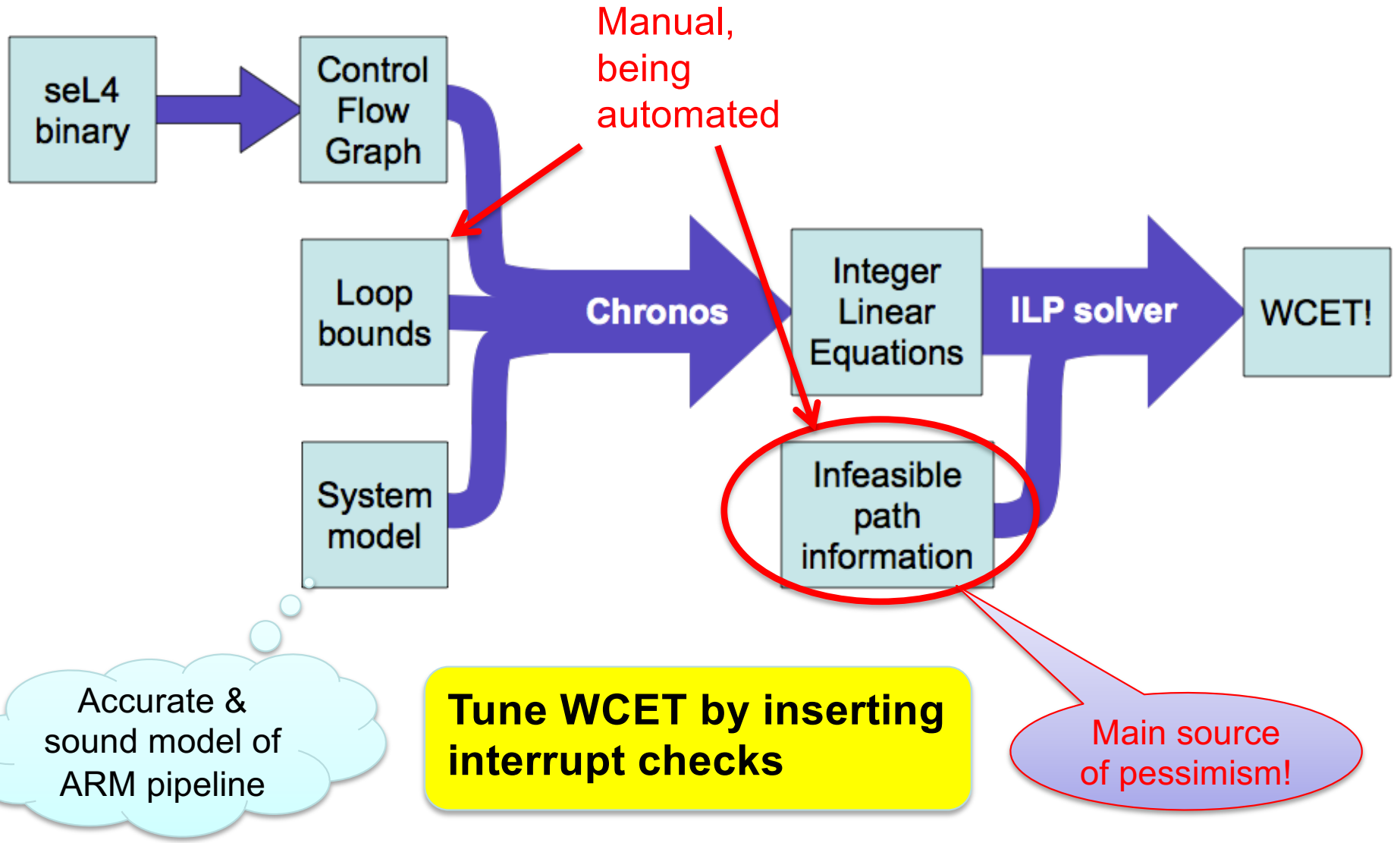
Timeliness



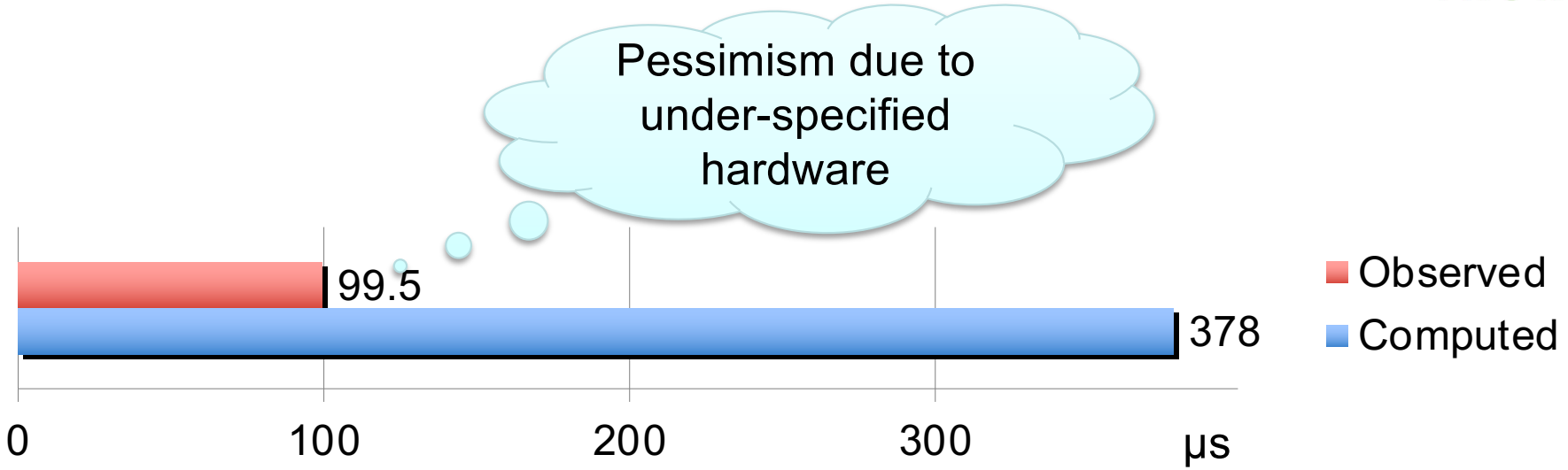
Need worst-case execution time (WCET) analysis of kernel

WCET Analysis Approach

NICTA



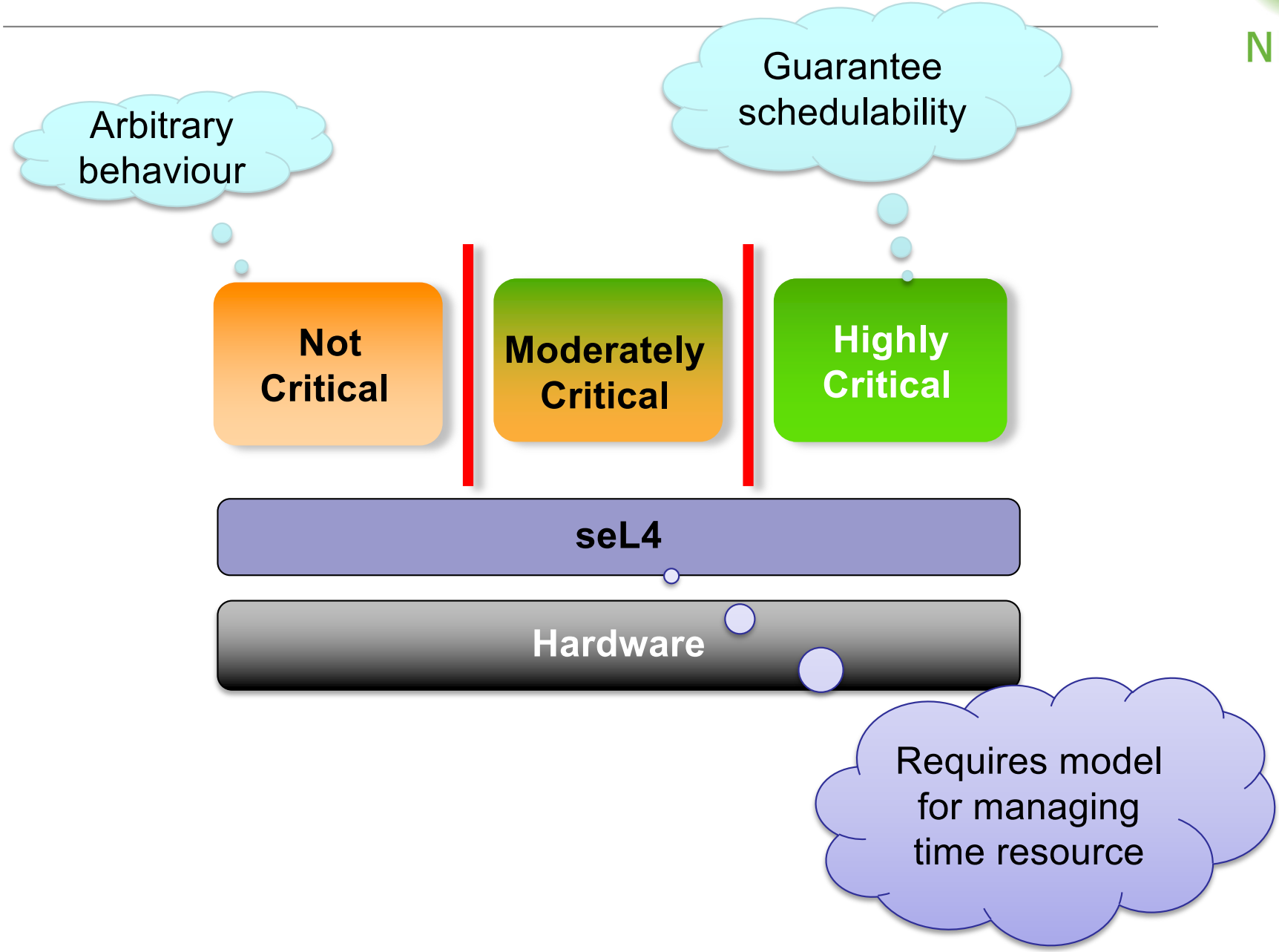
Result



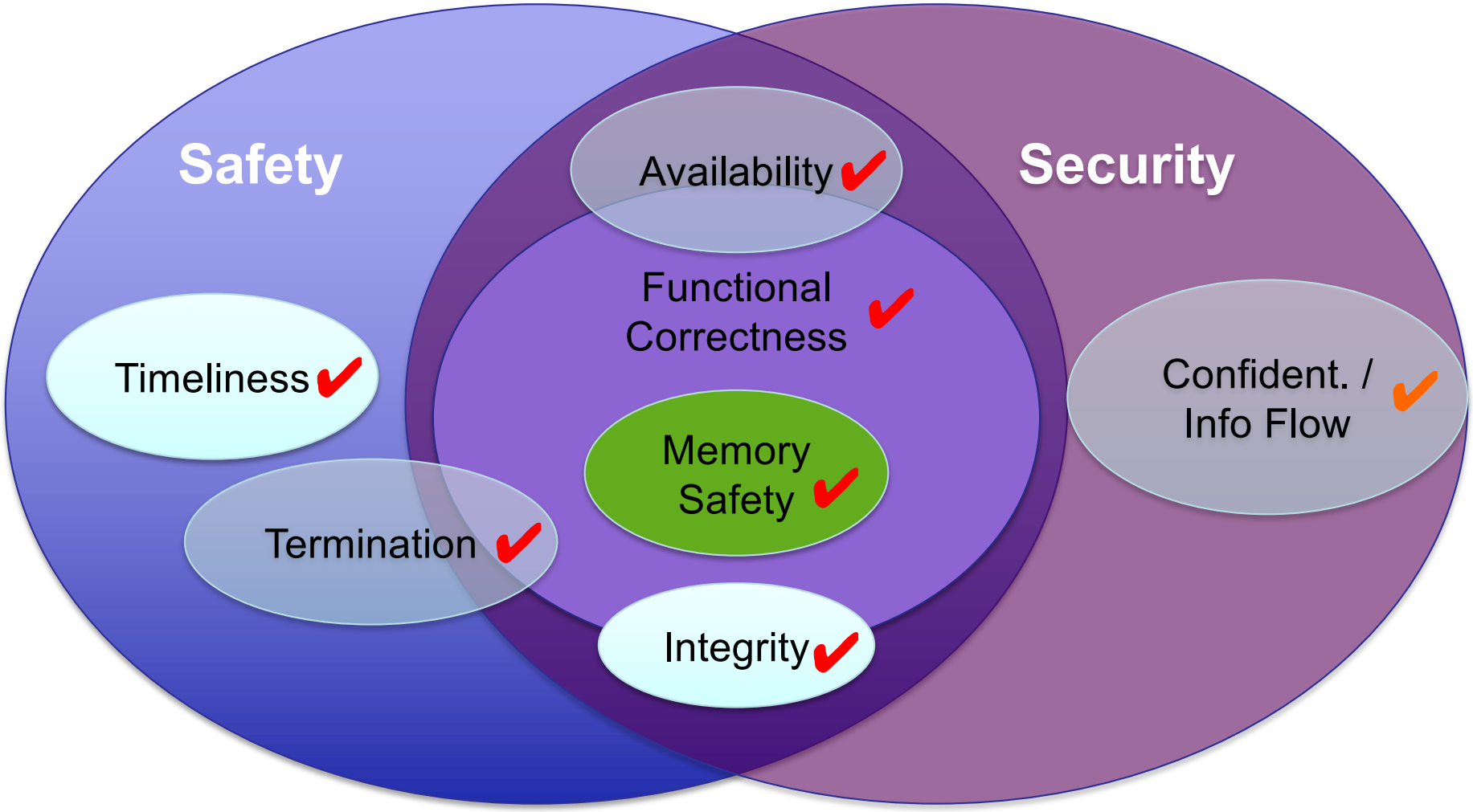
WCET presently limited by verification practicalities

- **10 µs seem achievable**

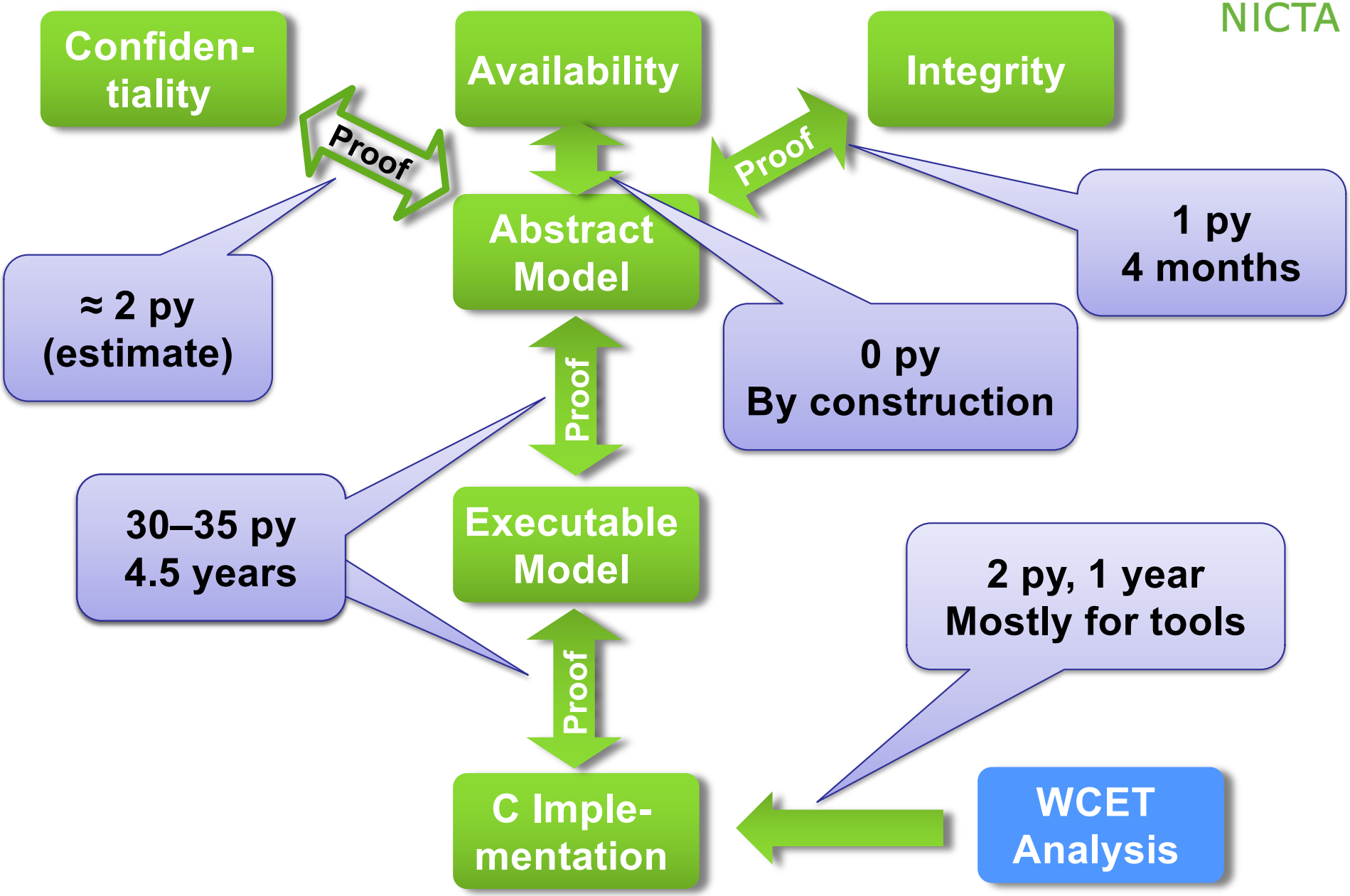
Future: Whole-System Schedulability



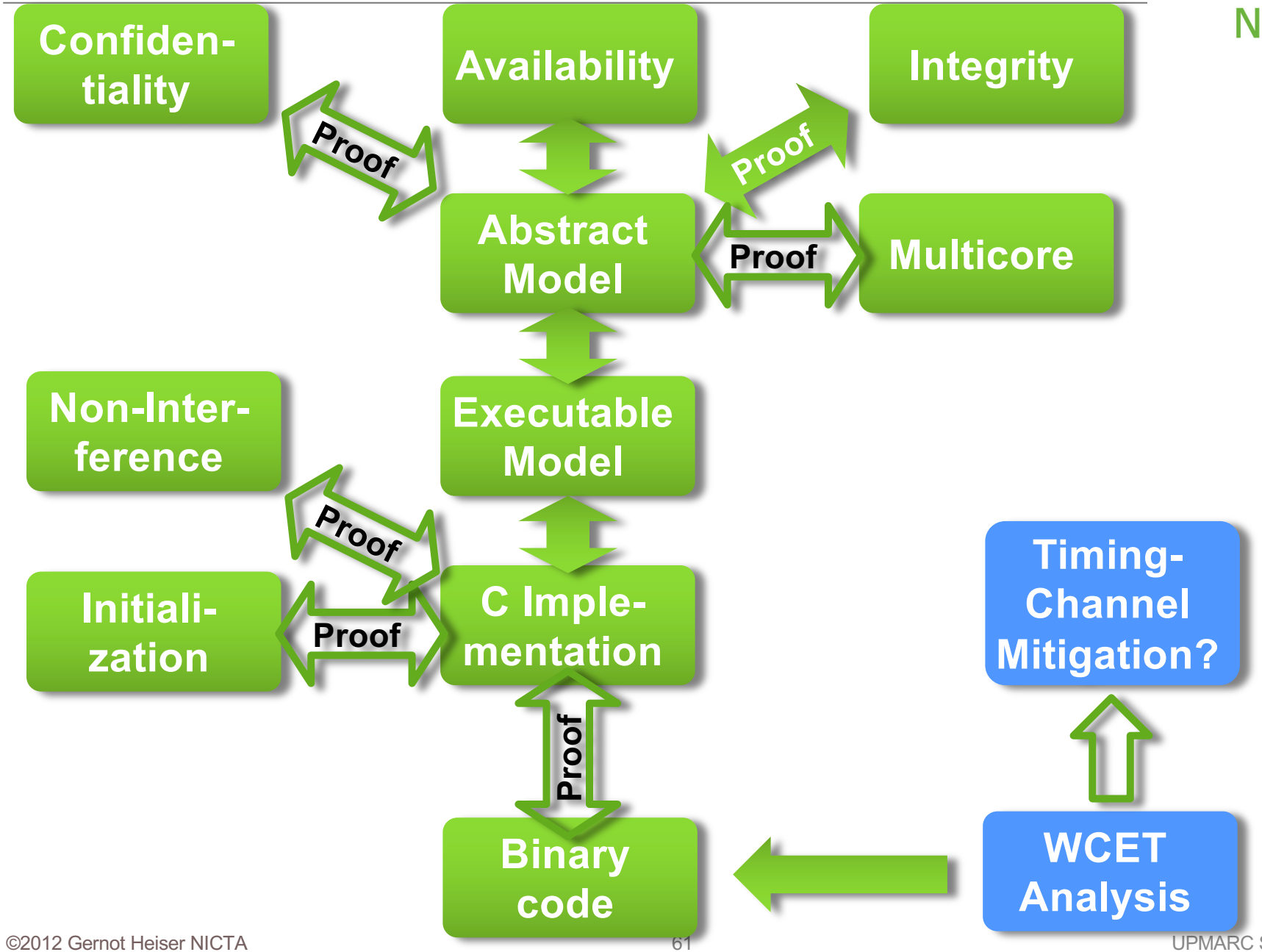
seL4 as Basis for Trustworthy Systems



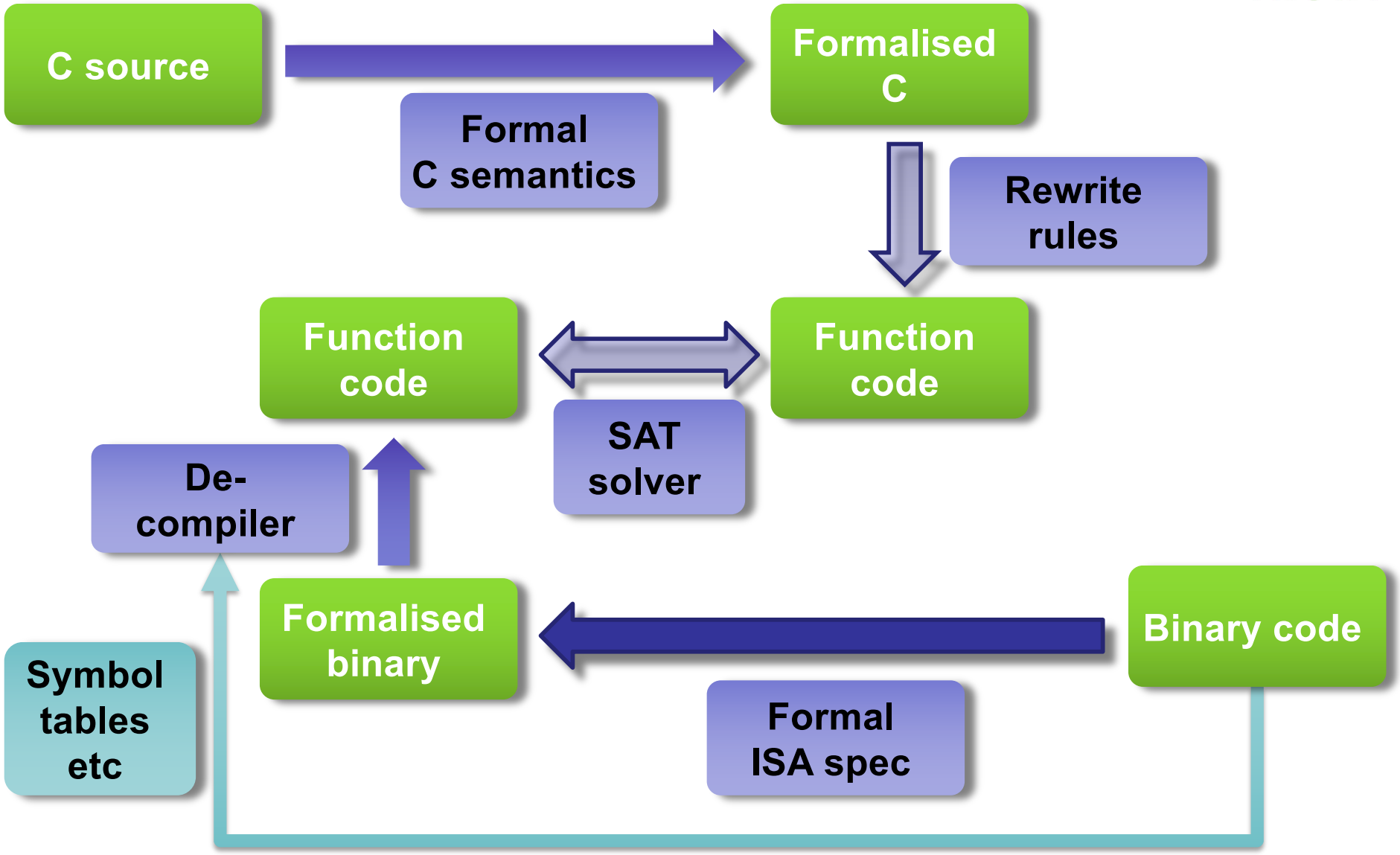
Proving seL4 Trustworthiness



seL4 – the Next 24 Months



Binary Code Verification (In Progress)



Multikernel Verification

NICTA

- 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
- What is a microkernel, and what is L4?
- seL4 – designed for trustworthiness
- Establishing trustworthiness
- **From kernel to system**
- Sample system 1: Secure access controller
- Sample system 2: RapiLog

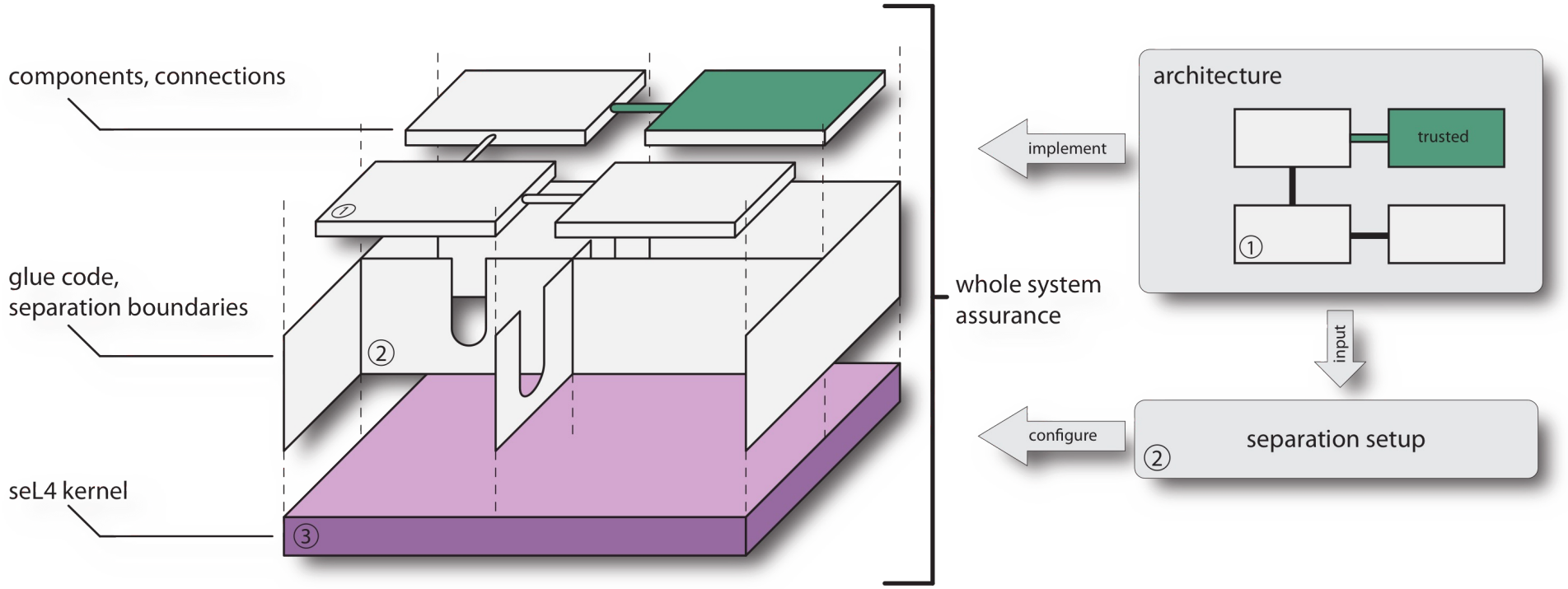
Phase Two: Full-System Guarantees

NICTA

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

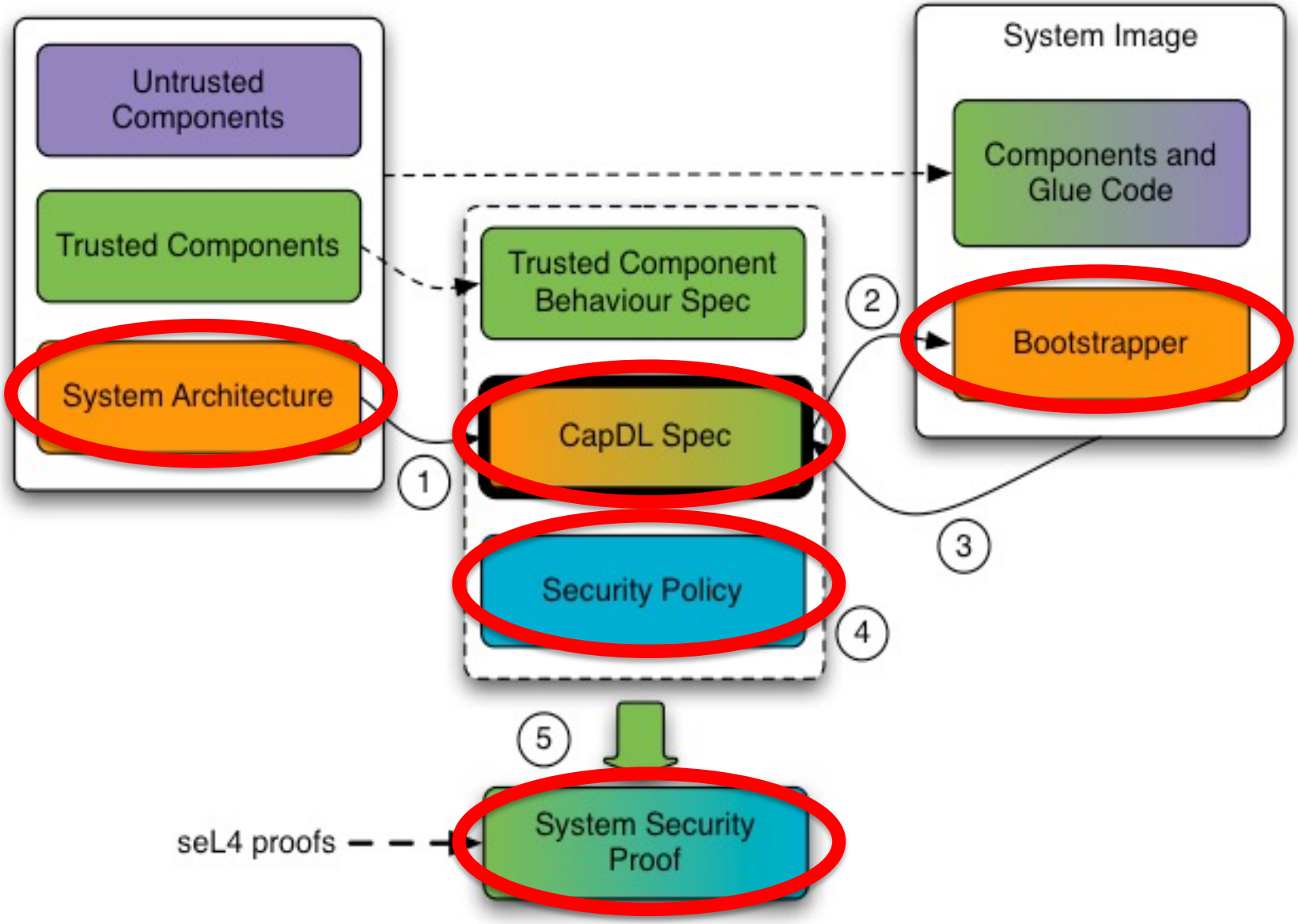
- Next step: Guarantees for real-world systems (1,000,000 LOC)

Overview of Approach

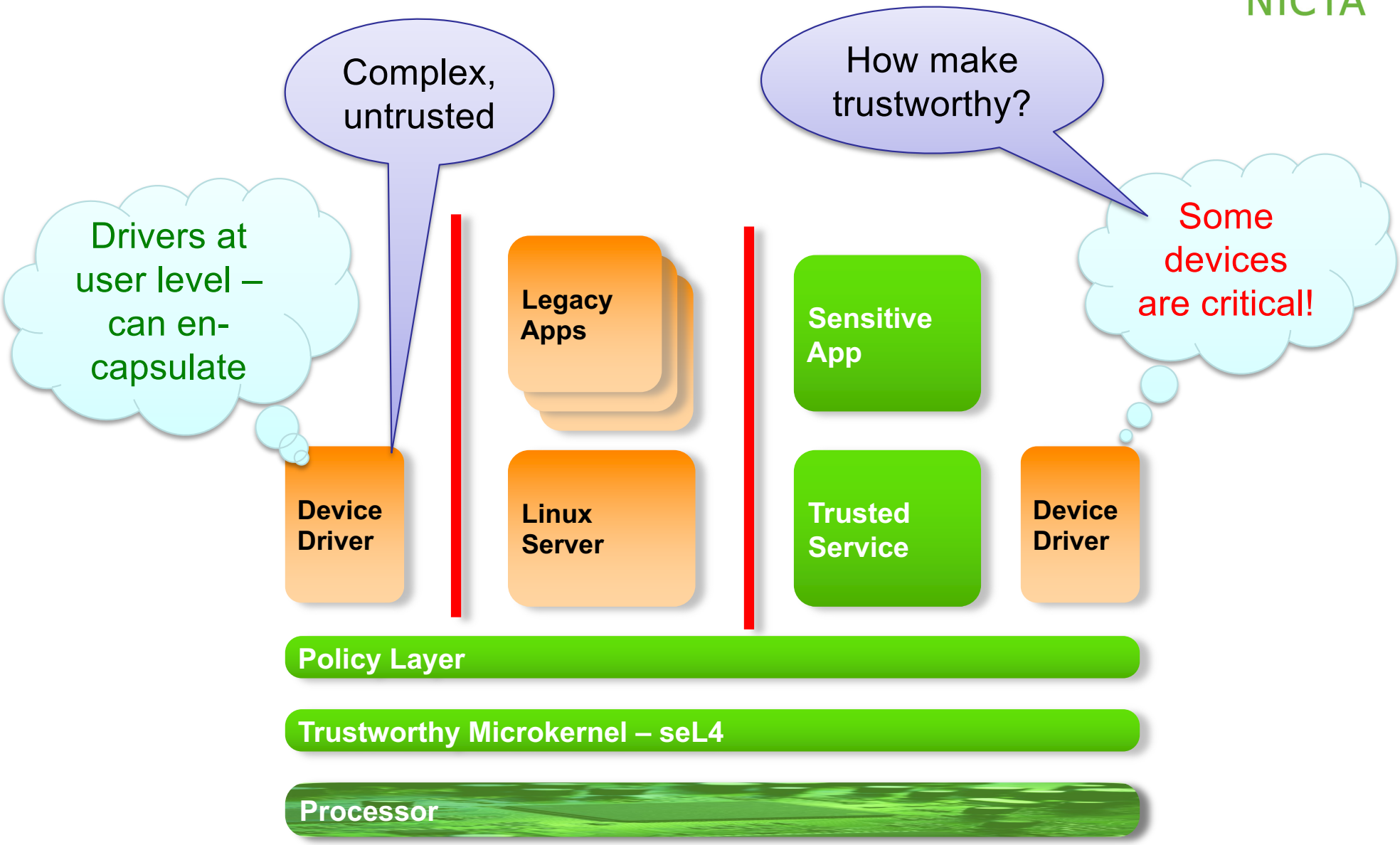


- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance

Specifying Security Architecture

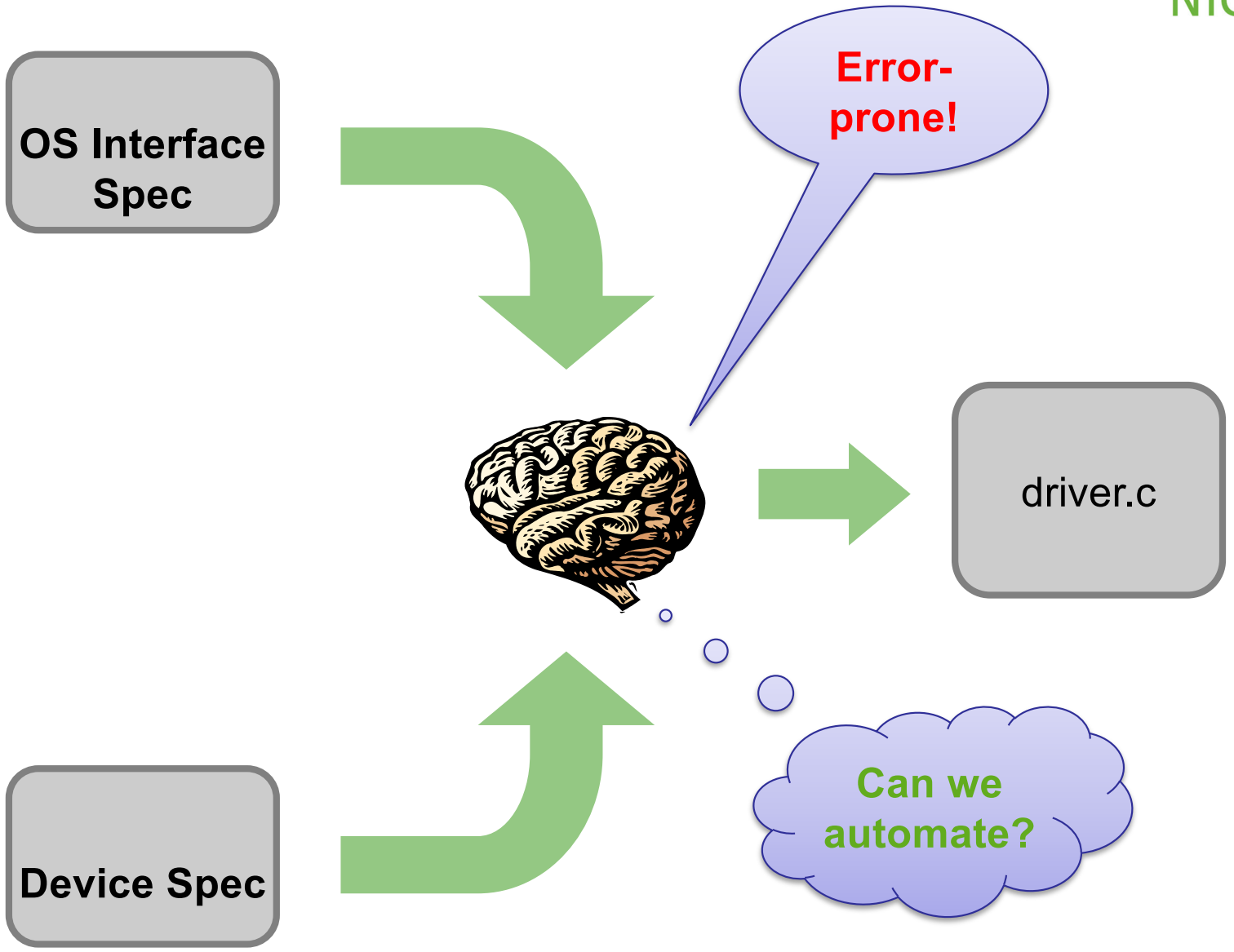


Device Drivers

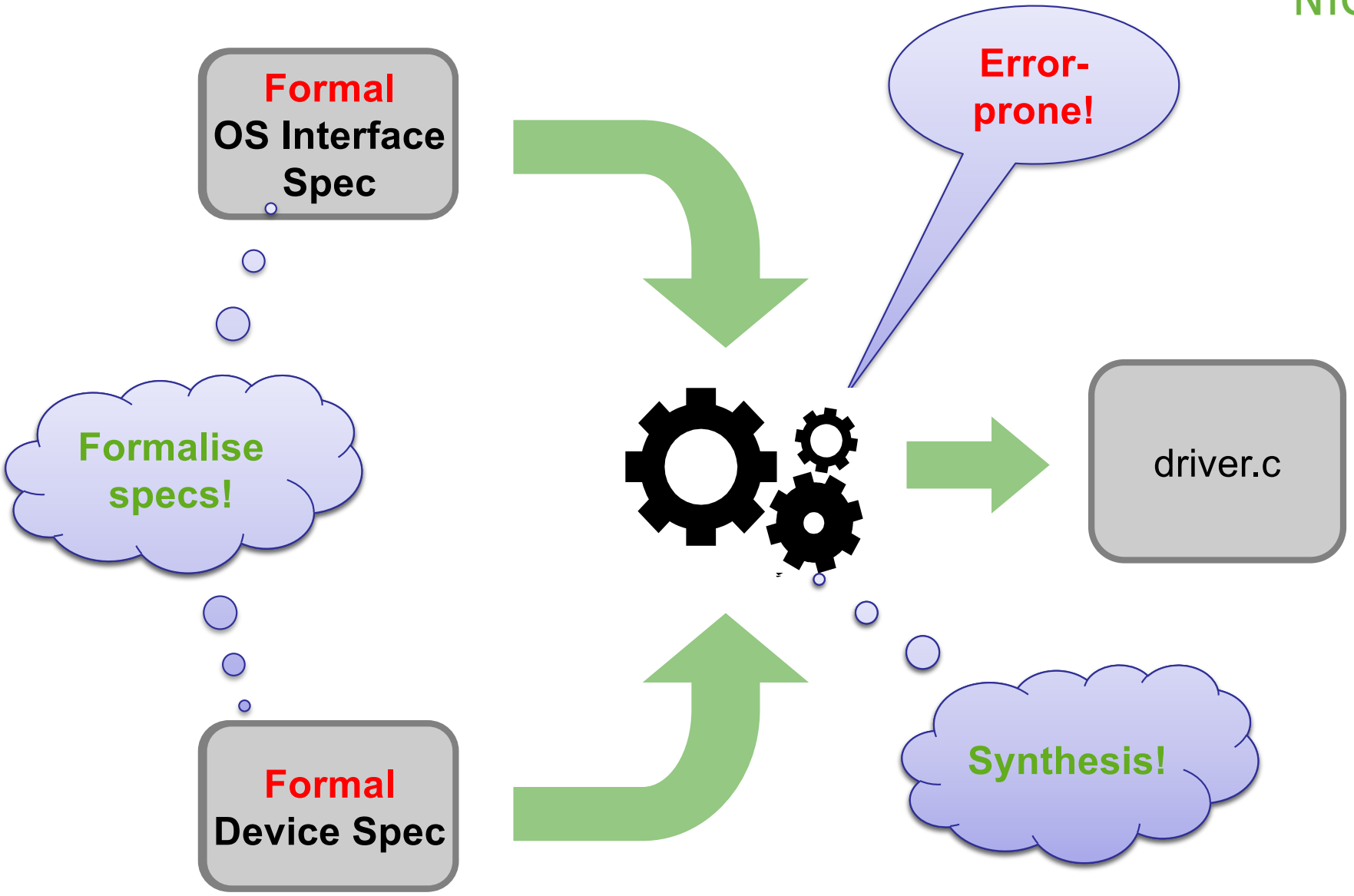


Driver Development

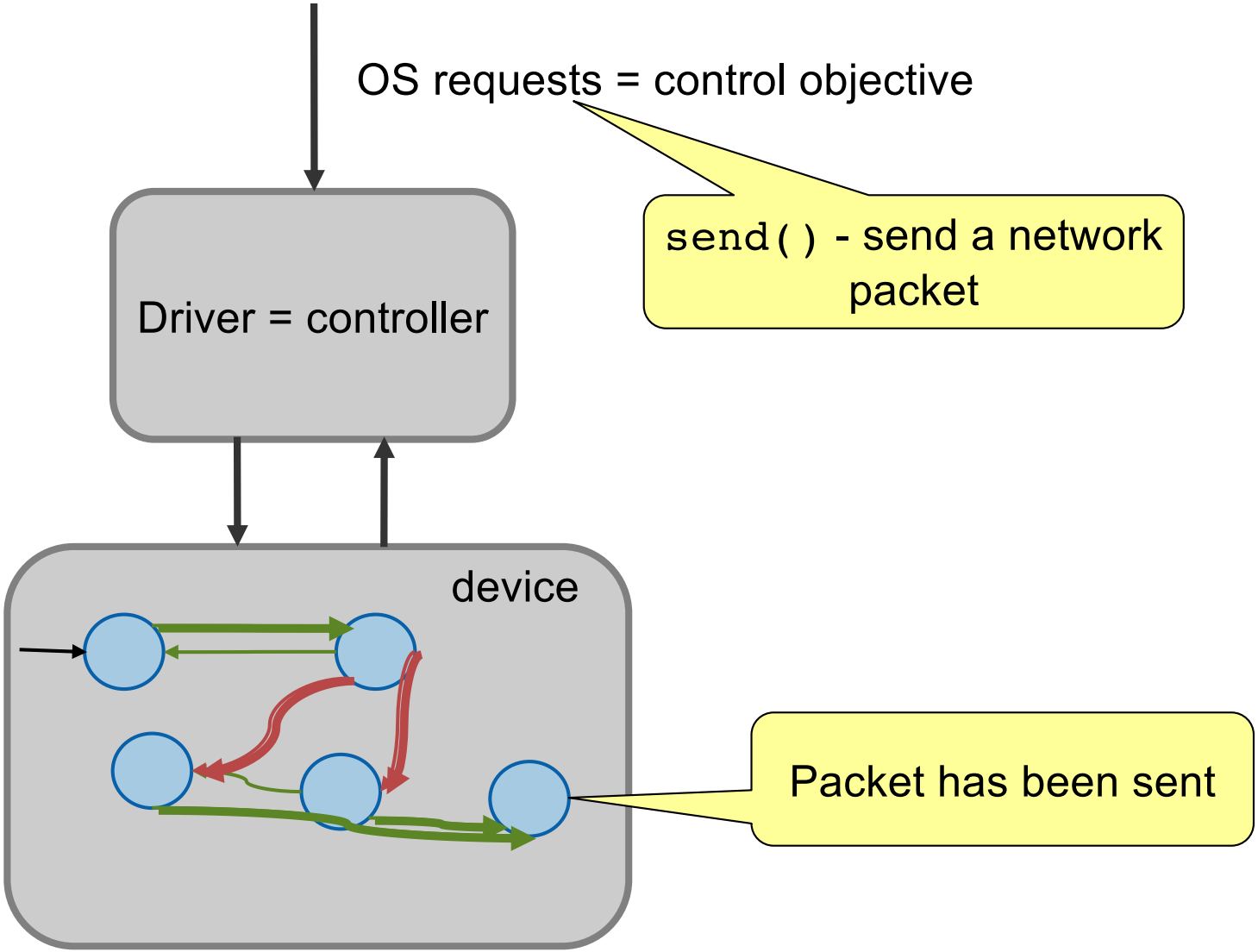
NICTA



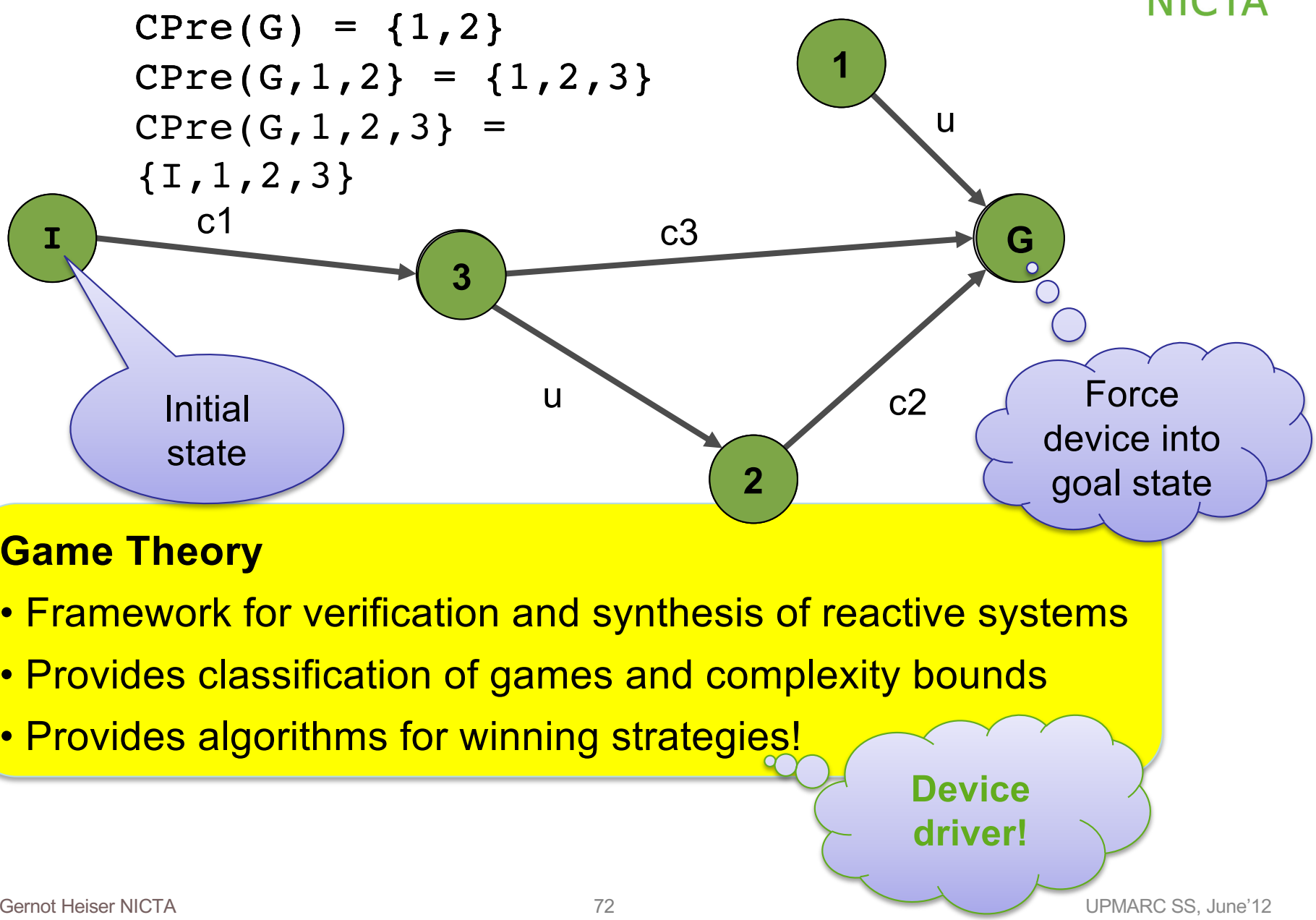
Driver Development



Driver Synthesis as Controller Synthesis



Synthesis Algorithm (Main Idea)



Game Theory

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

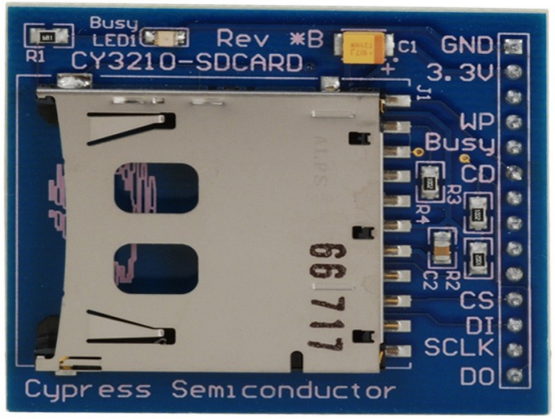
Device driver!

Drivers Synthesised (To Date)

IDE disk controller

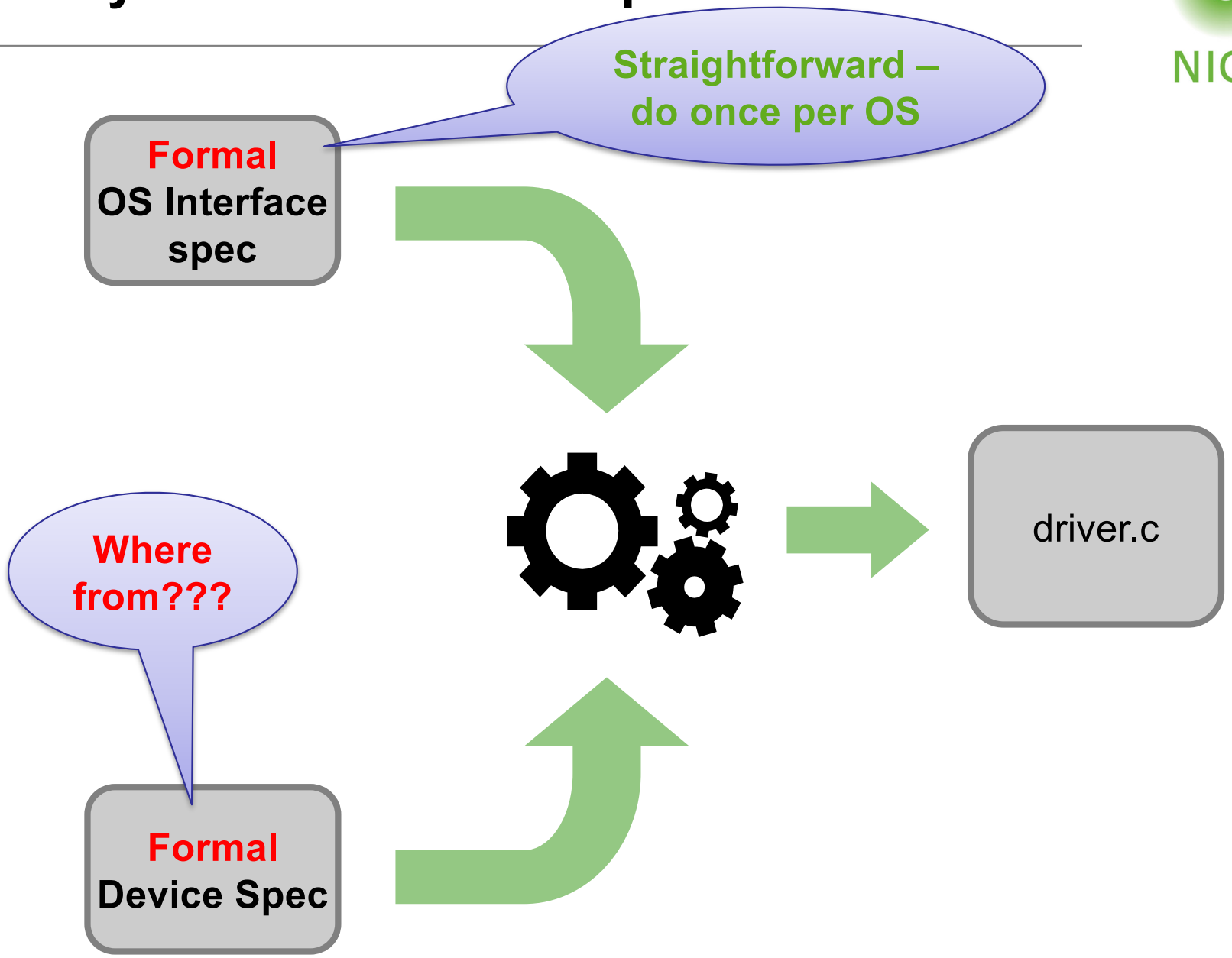
W5100 Eth shield

Asix AX88772
USB-to-Eth adapter



SD host controller

Driver Synthesis: Interface Specs



Hardware Design Workflow

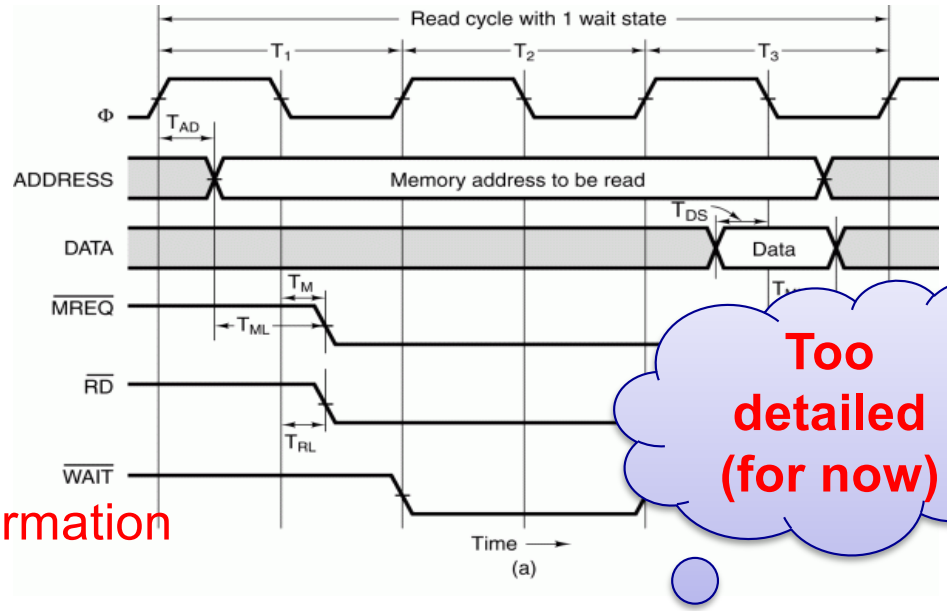
Informal specification

High-level model

Manual transformation

Register-transfer-level description

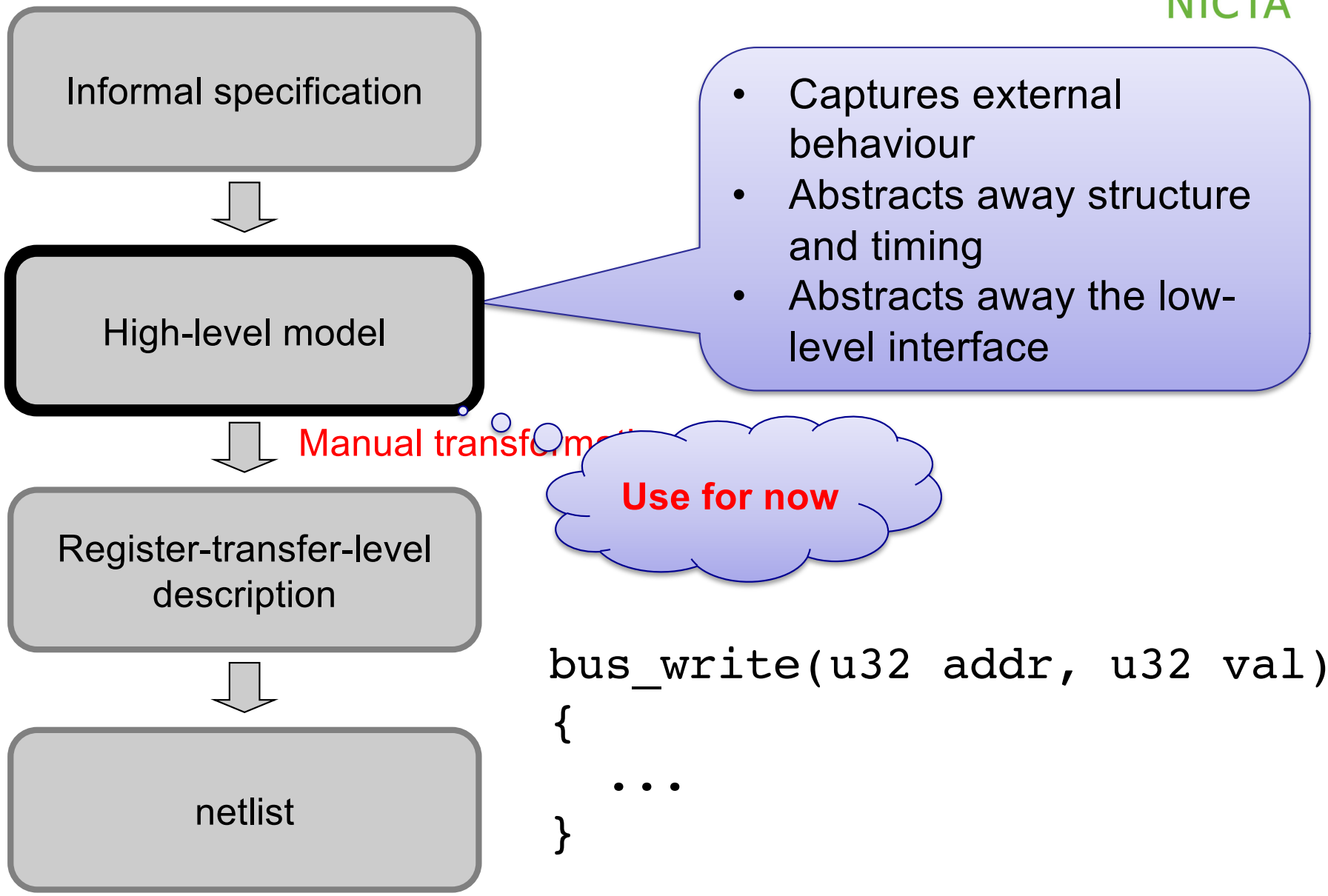
netlist



Too detailed (for now)

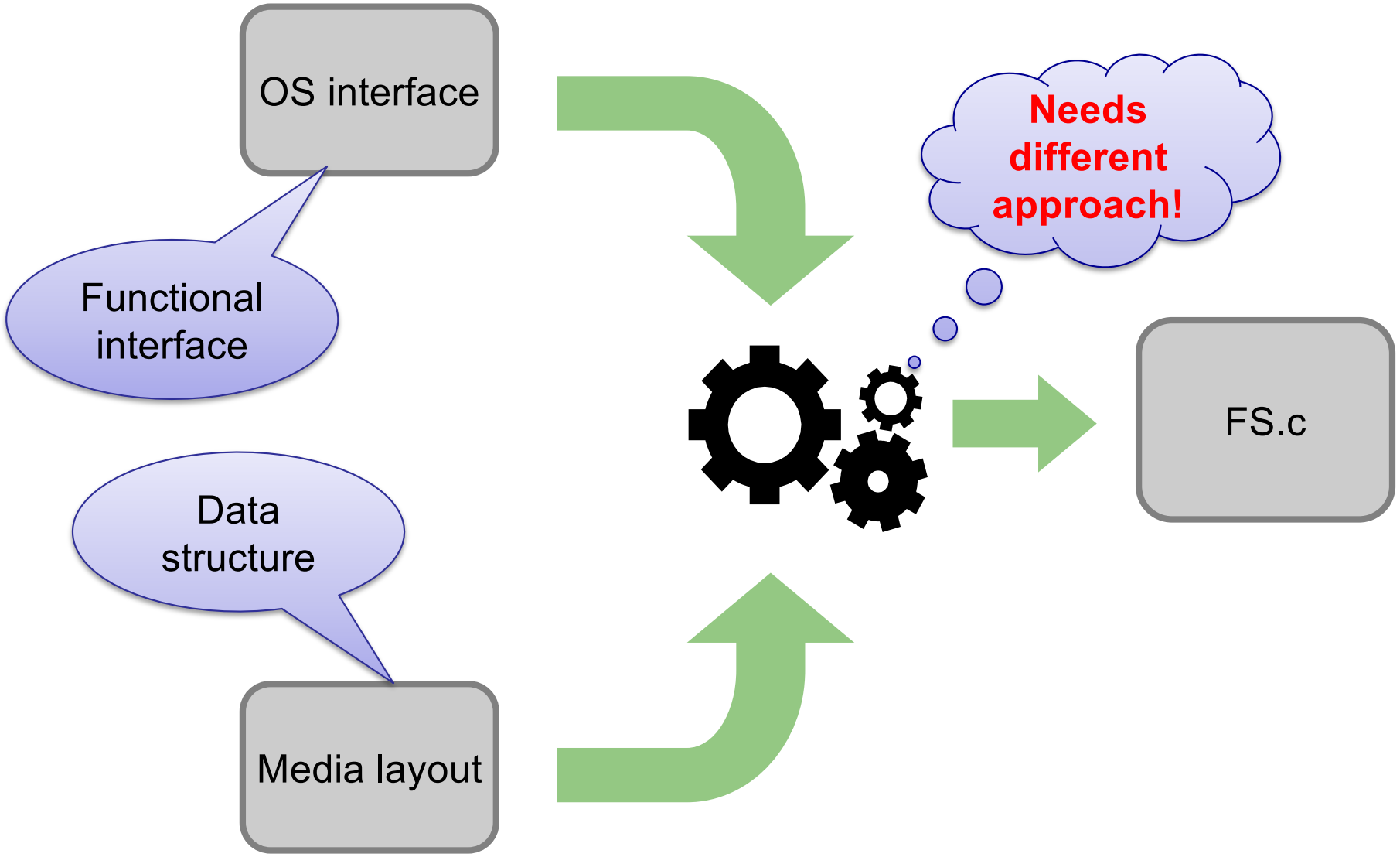
- Low-level description: registers, gates, wires.
- Cycle-accurate
- Precisely models internal device architecture and interfaces
- "Gold reference"

Hardware Design Workflow



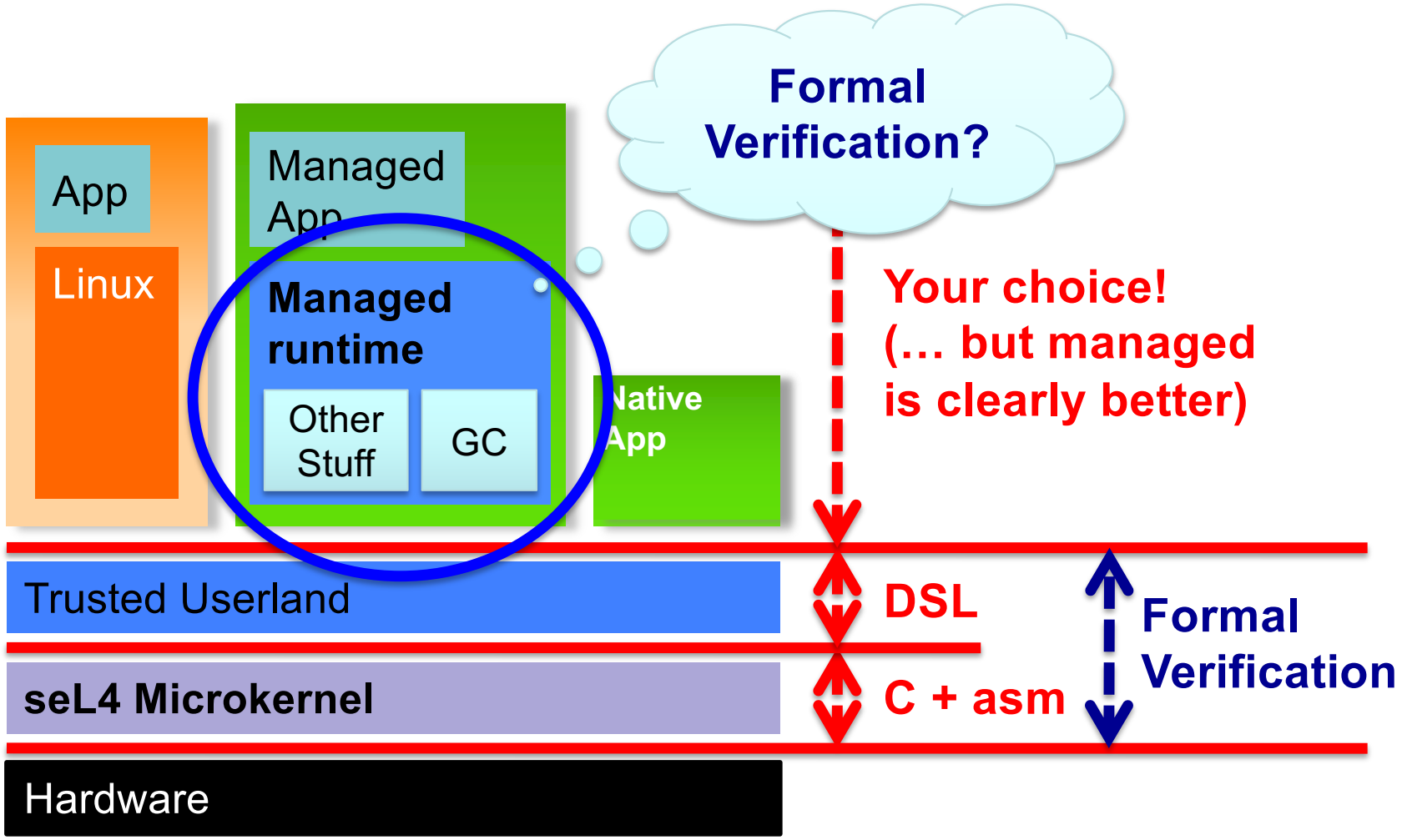
From Drivers to File Systems?

NICTA



Building Secure Systems: Long-Term View

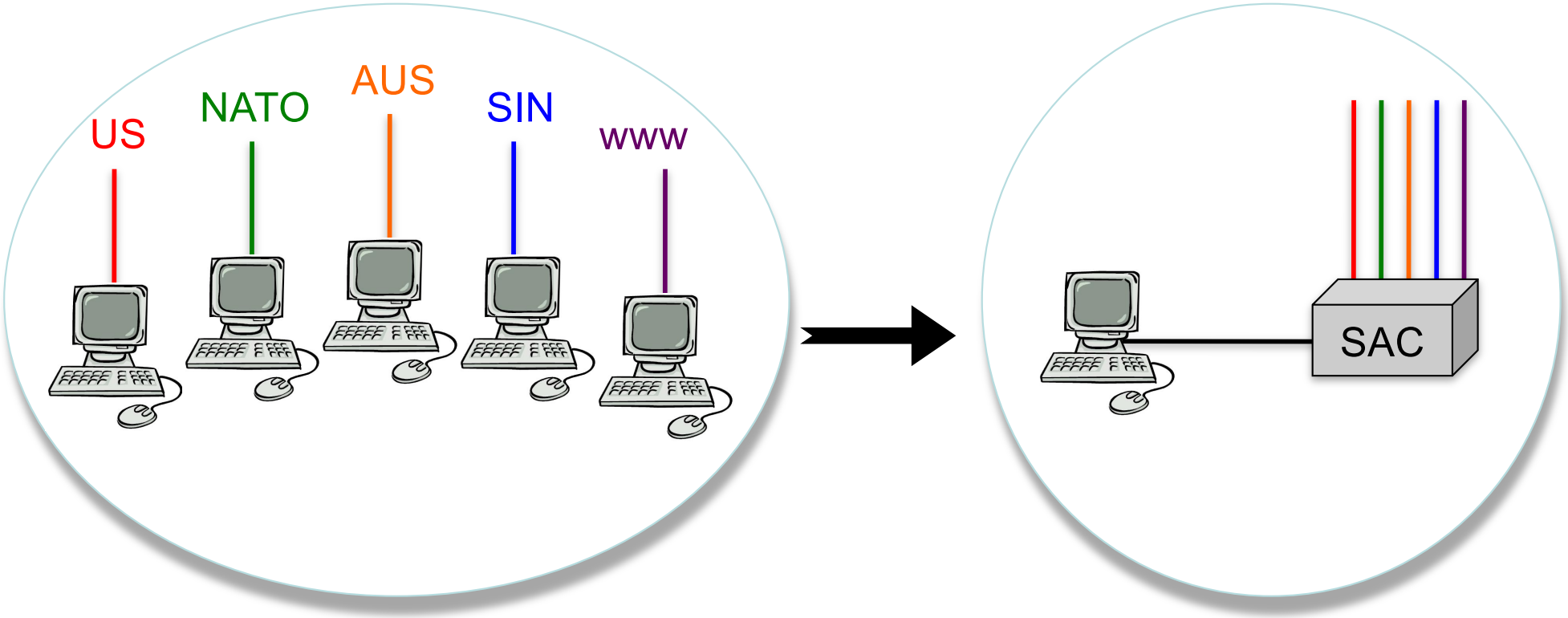
NICTA



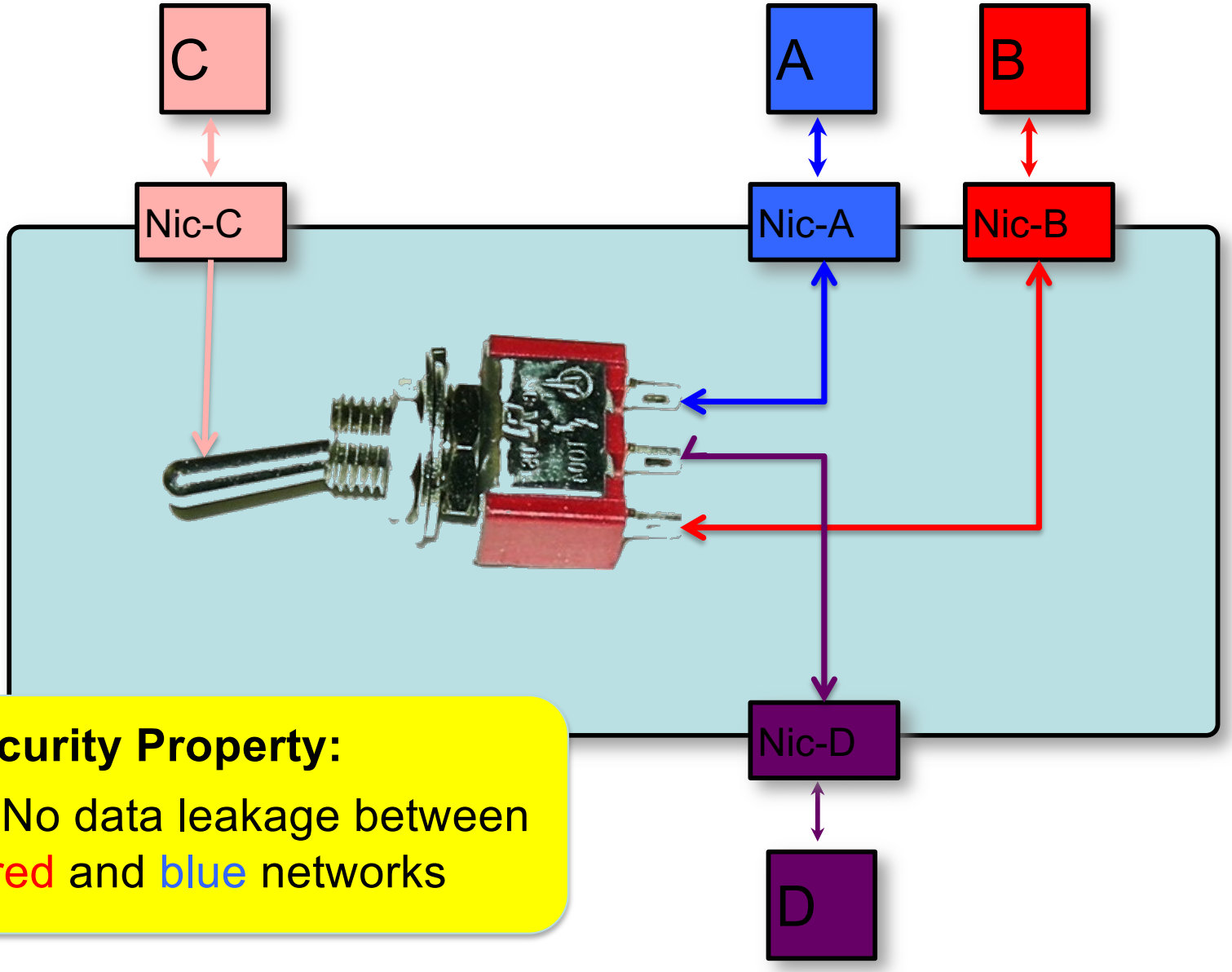
Agenda

- Motivation
- What is a microkernel, and what is L4?
- seL4 – designed for trustworthiness
- Establishing trustworthiness
- From kernel to system
- **Sample system 1: Secure access controller**
- Sample system 2: RapiLog

Proof of Concept: Secure Access Controller



Logical Function

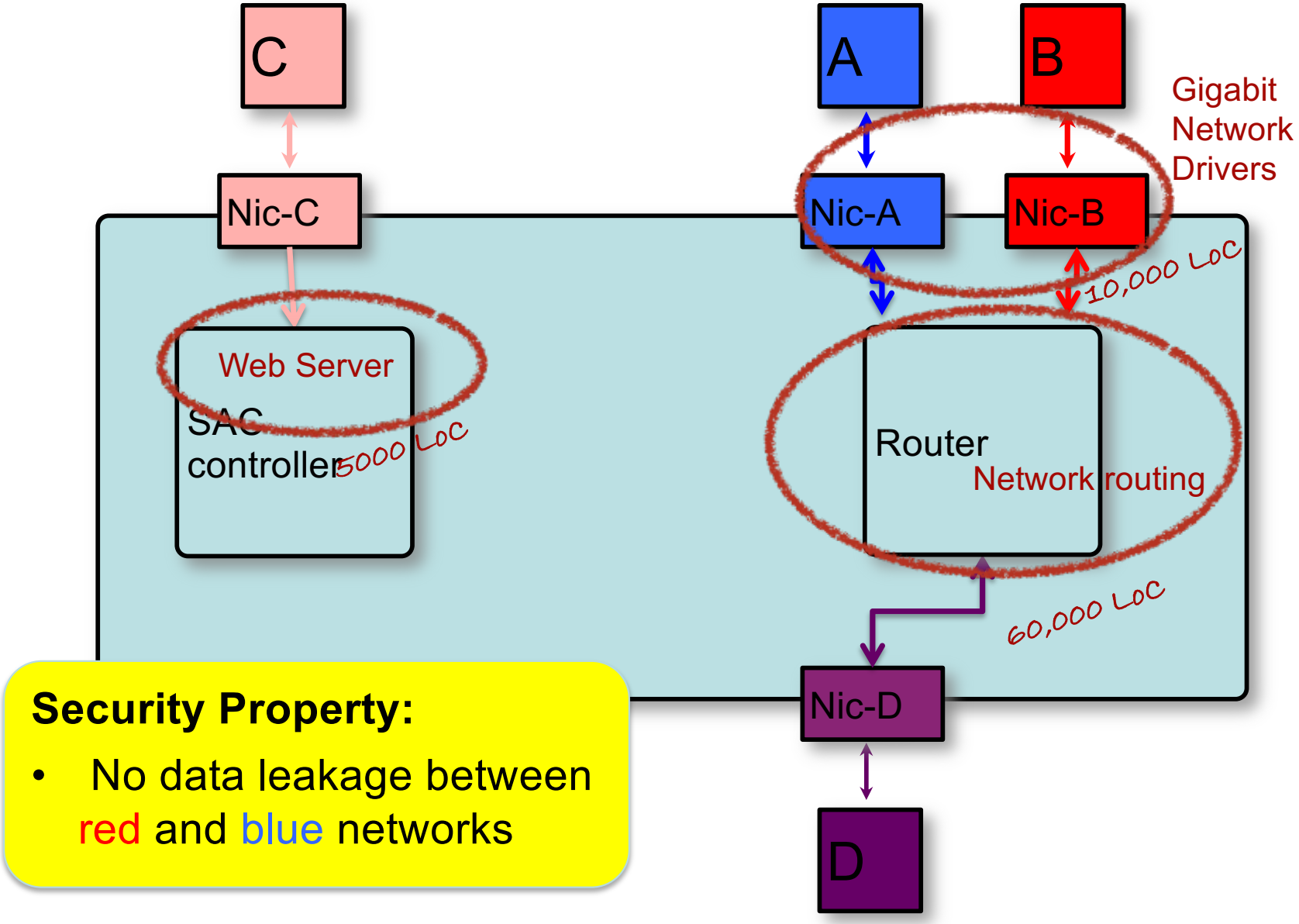


Security Property:

- No data leakage between red and blue networks

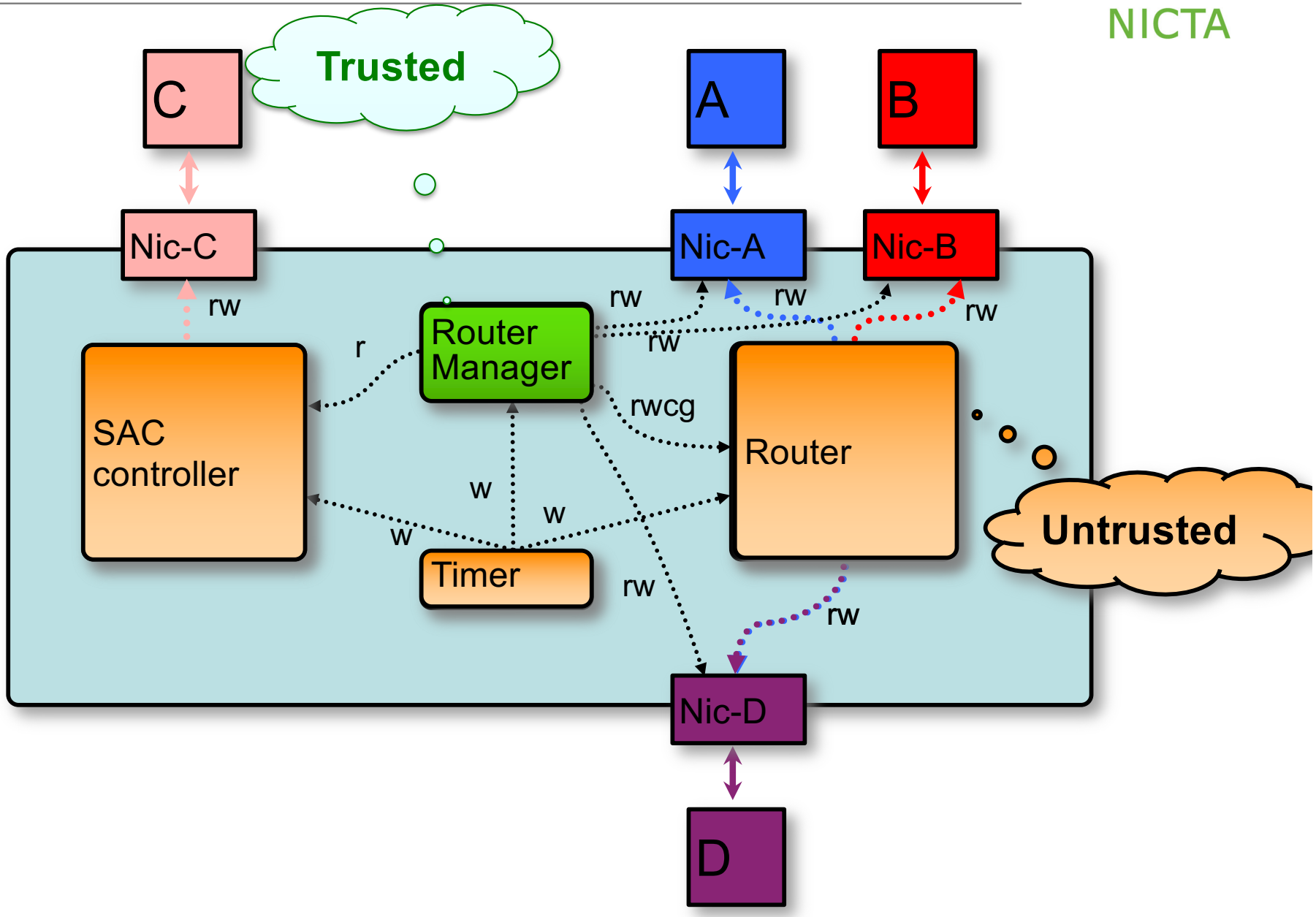
©NICTA 2010

Logical Function



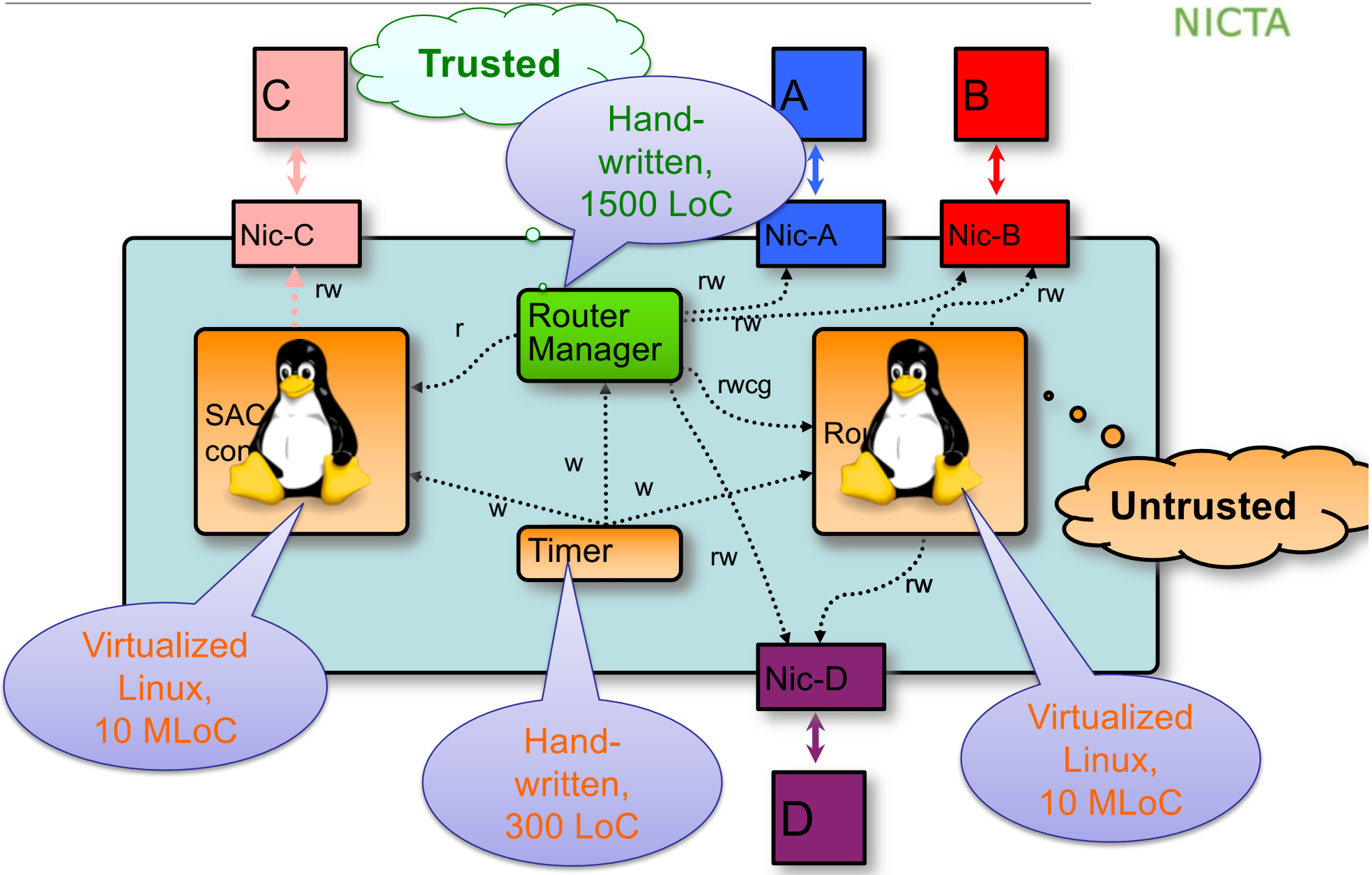
©2012 Gernot Heiser NICTA

Minimal TCB



©2012 Gernot Heiser NICTA

Implementation



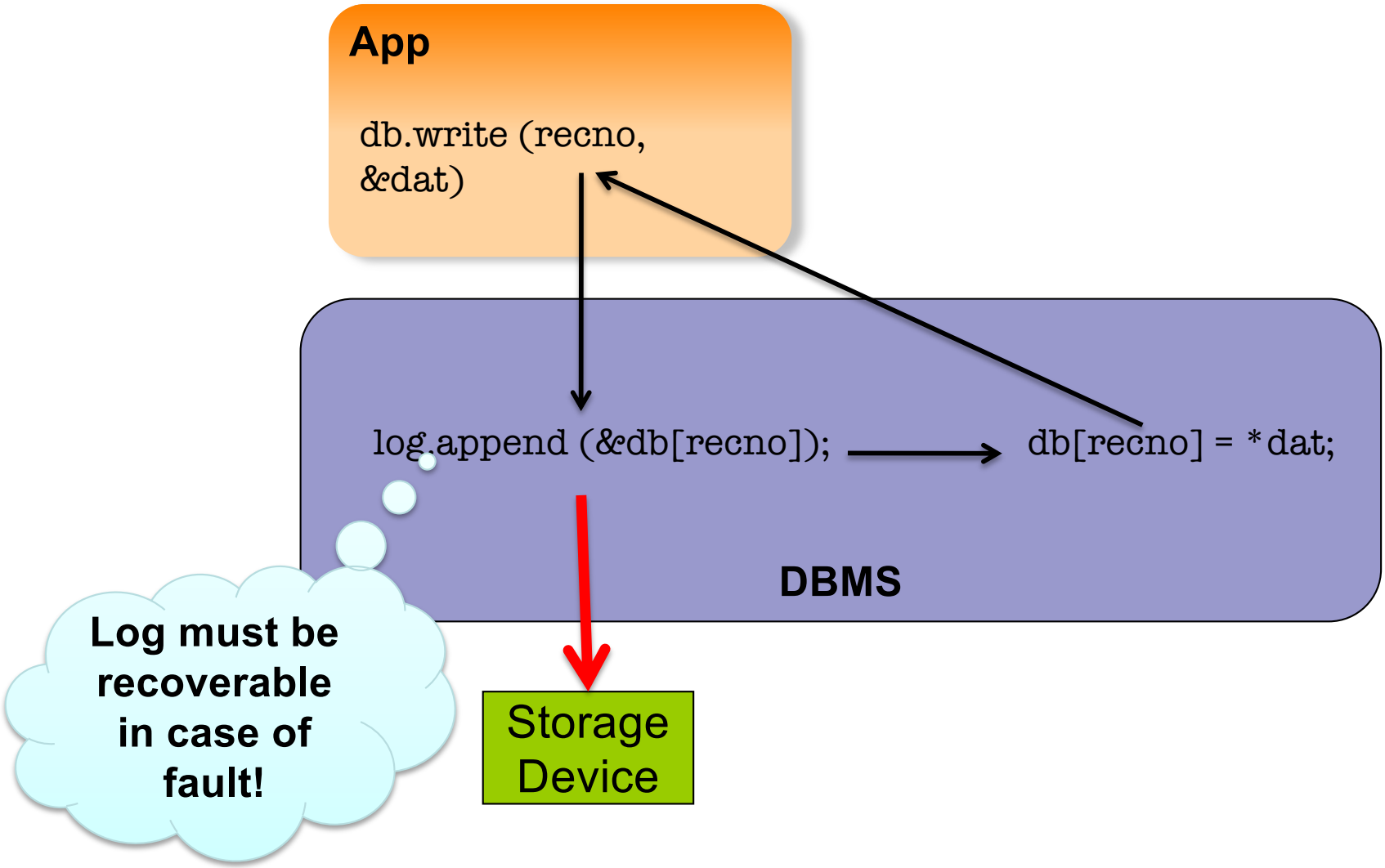
© NICTA 2010

Agenda

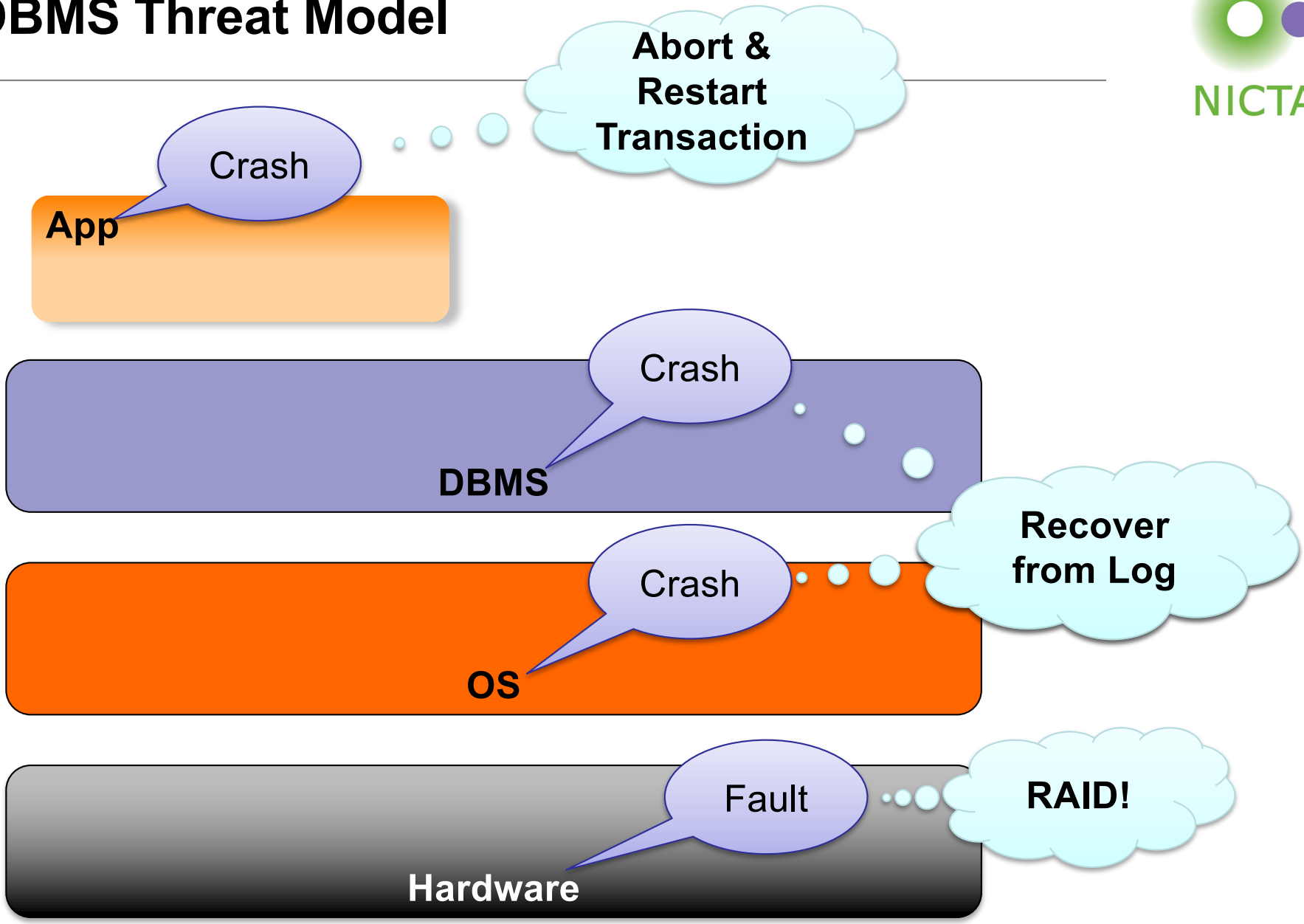
- Motivation
- What is a microkernel, and what is L4?
- seL4 – designed for trustworthiness
- Establishing trustworthiness
- From kernel to system
- Sample system 1: Secure access controller
- **Sample system 2: RapiLog**

Database Transactions

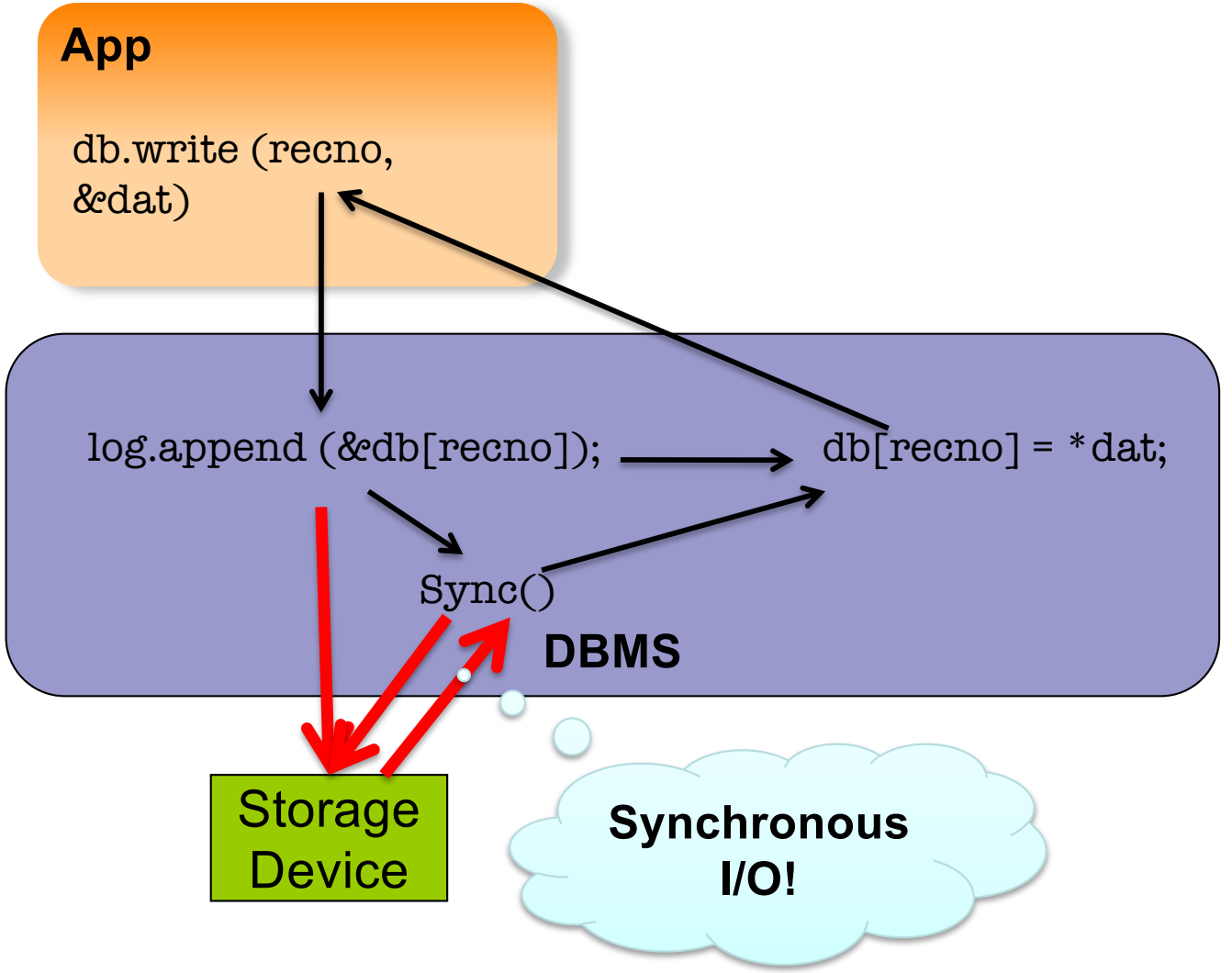
Various approaches, but today usually *write-ahead logging*:



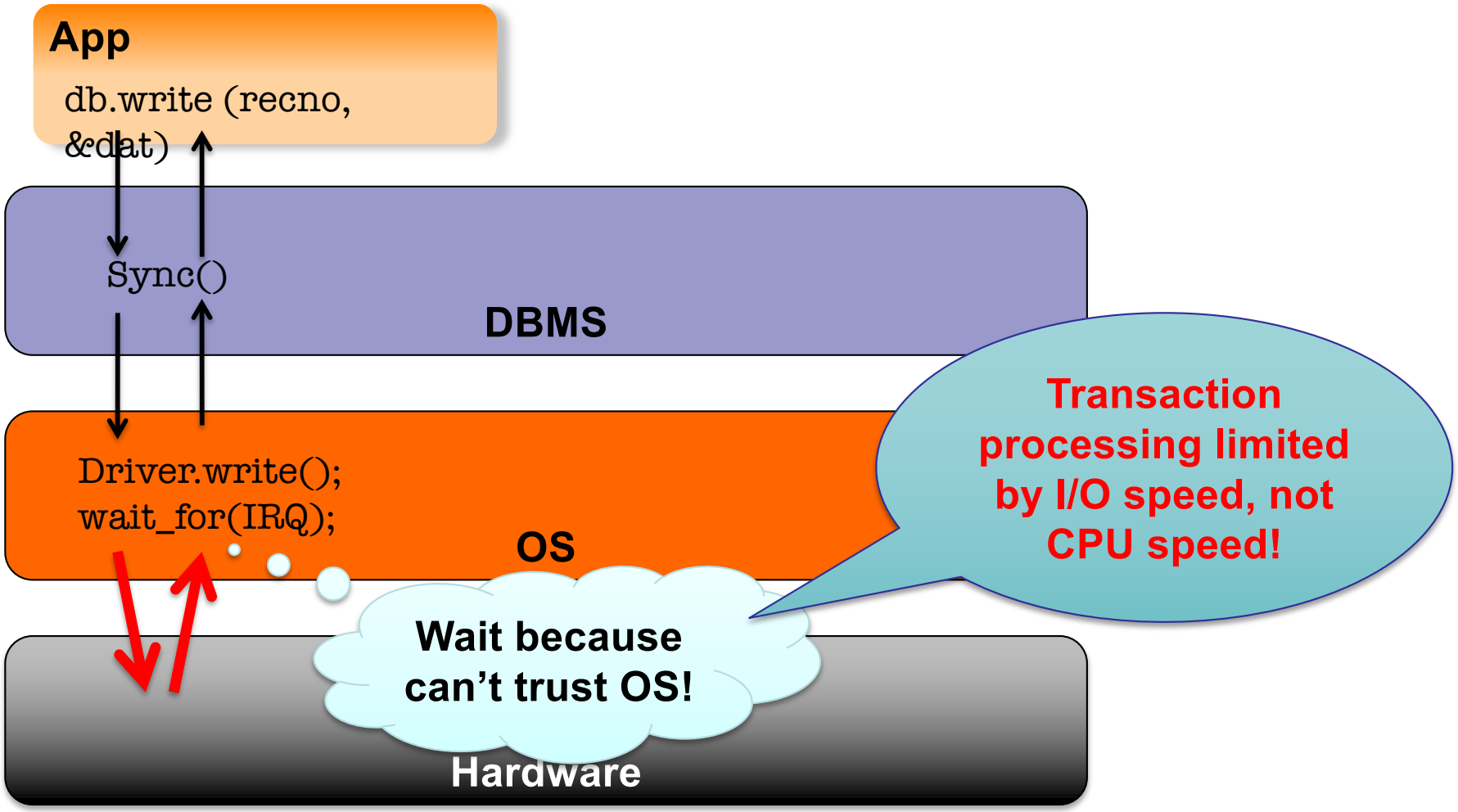
DBMS Threat Model



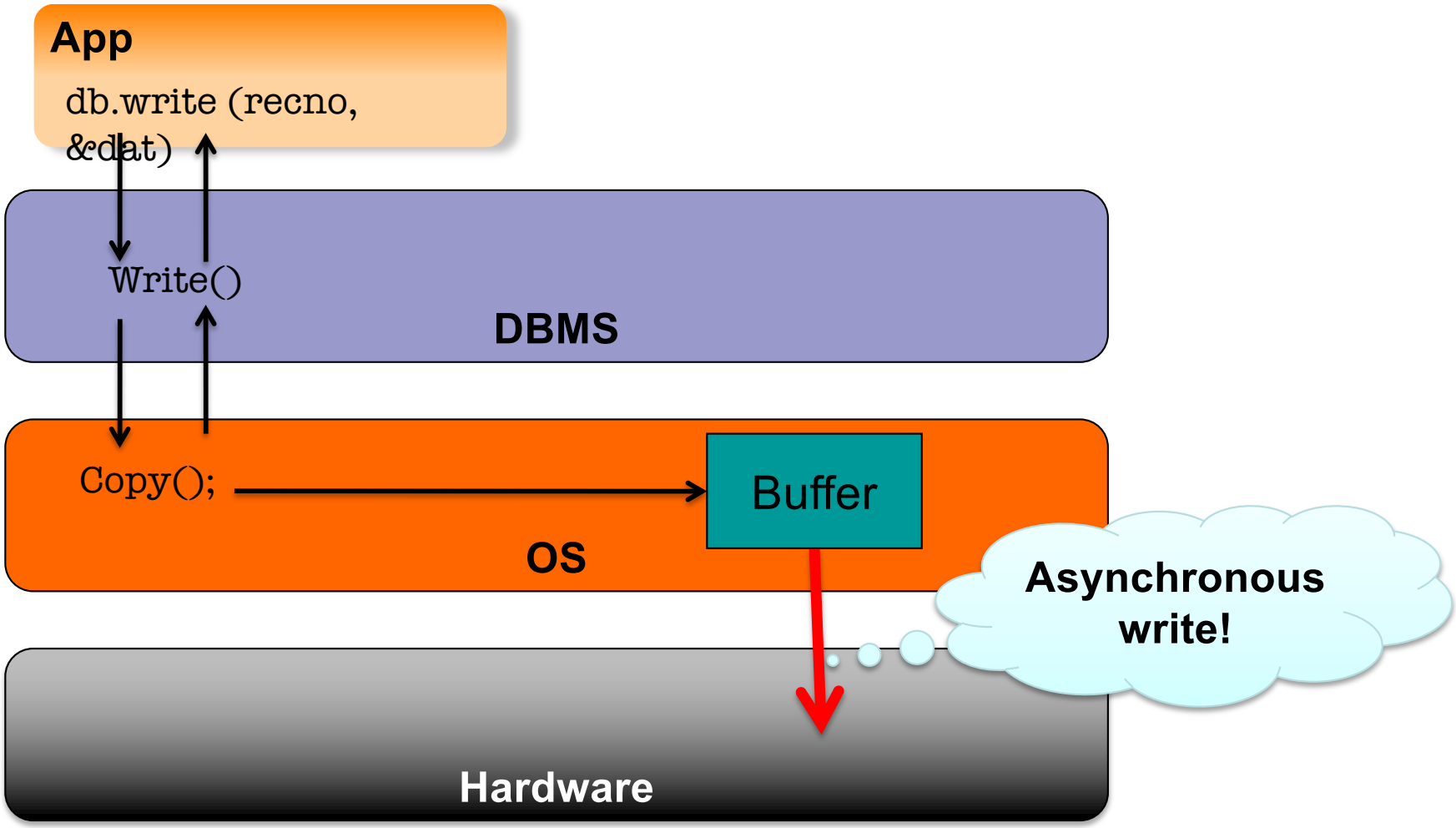
Log Data Must Be Recoverable!



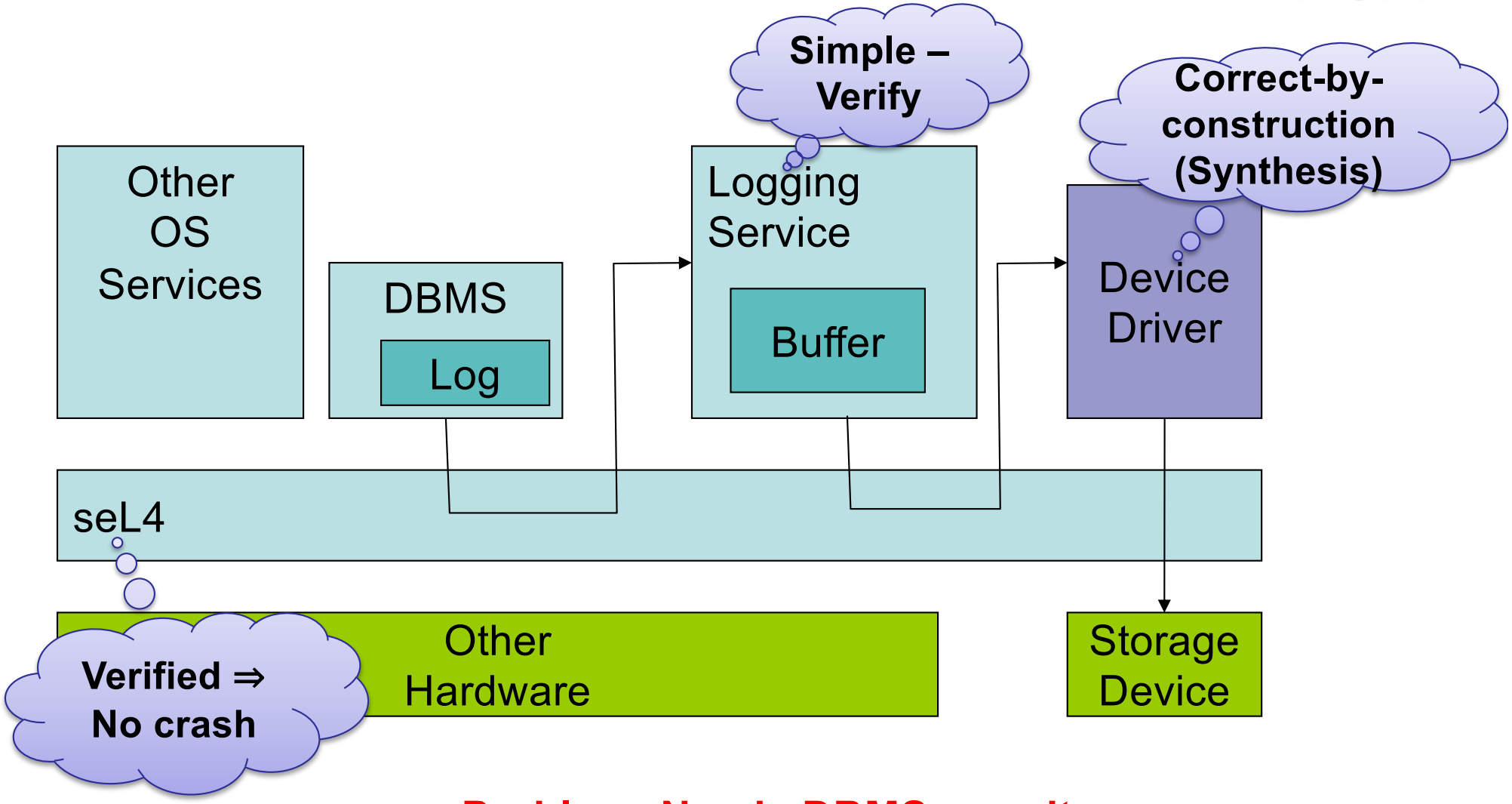
Log Data Must Be Recoverable!



What If We Could Trust the OS?

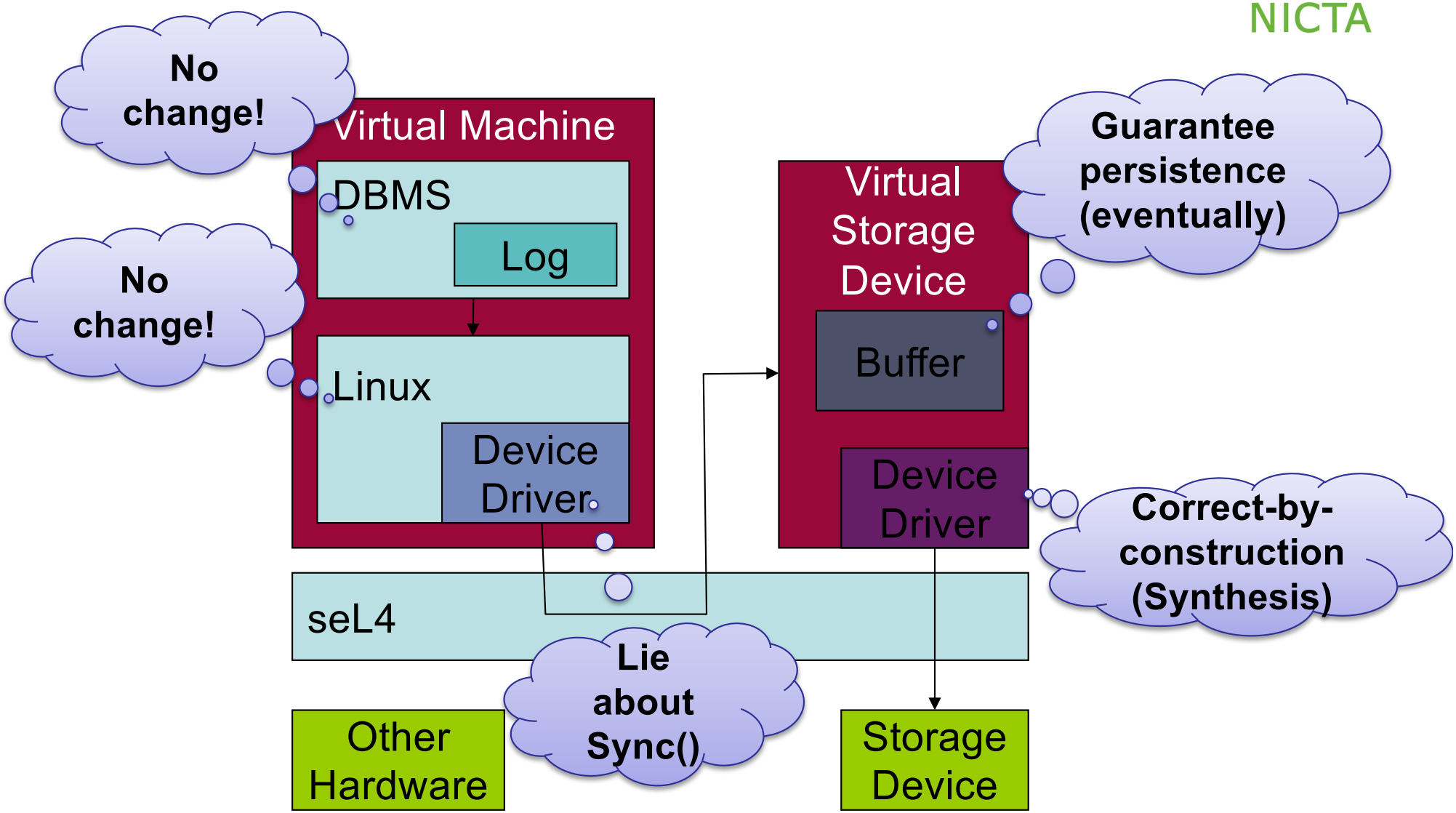


But We Can Trust seL4!

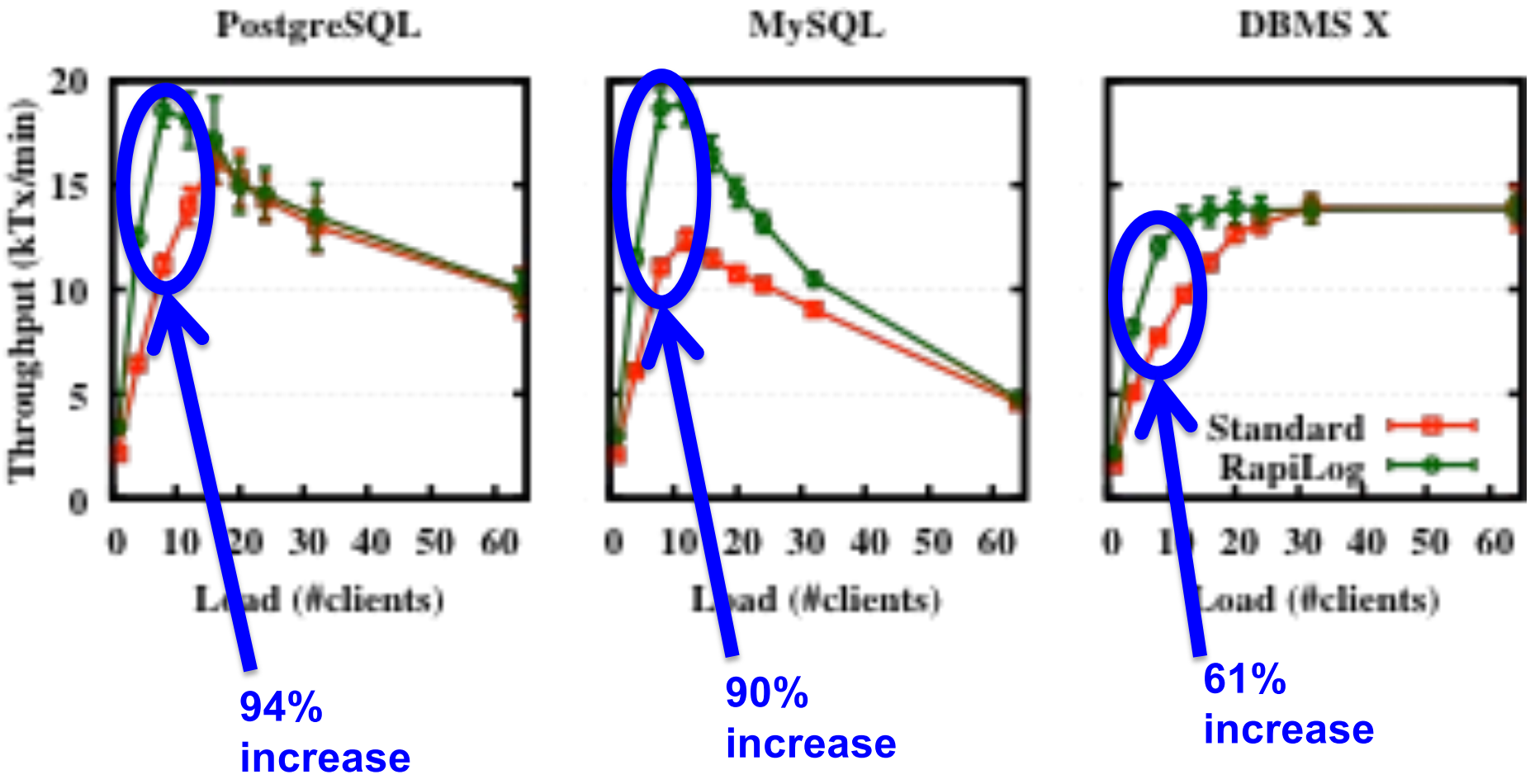


Problem: Needs DBMS re-write

RapiLog: Use Virtualization

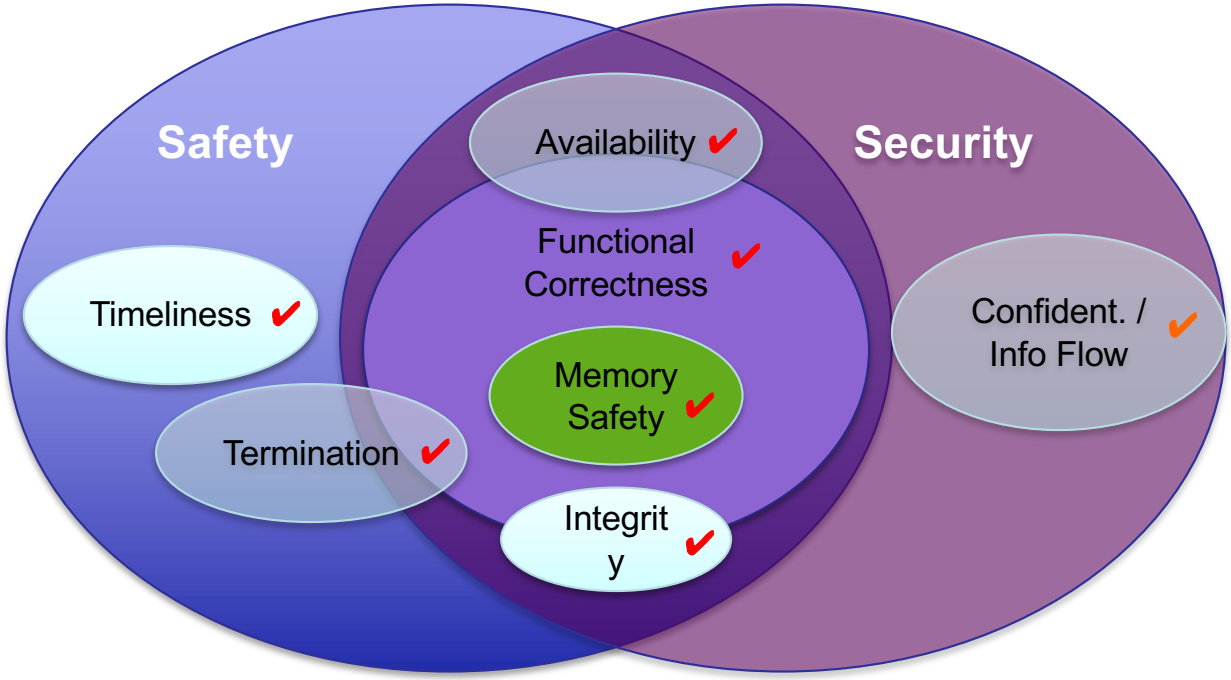


Performance



Also maintain durability on power failure!

Trustworthy Systems – We've Made a Start!



Thank You!

mailto:gernot@nicta.com.au
Twitter @GernotHeiser
Google: "nicta trustworthy systems"