Making Trusted Systems Trustworthy

Gernot Heiser

NICTA and University of New South Wales

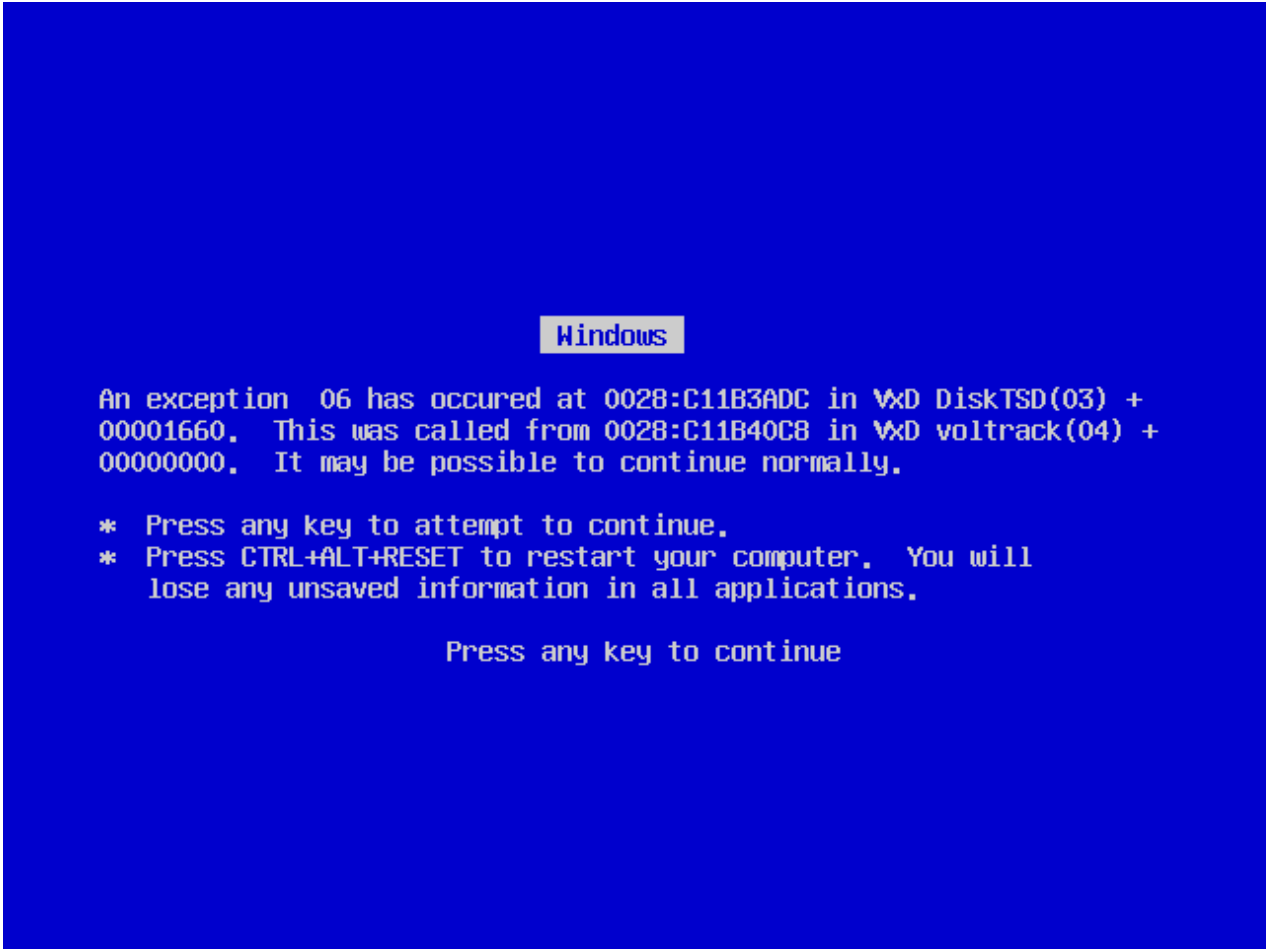
Sydney, Australia

Australian Government

Department of Broadband, Communications and the Digital Economy

Australian Research Council

NICTA Funding and Supporting Members and Partners



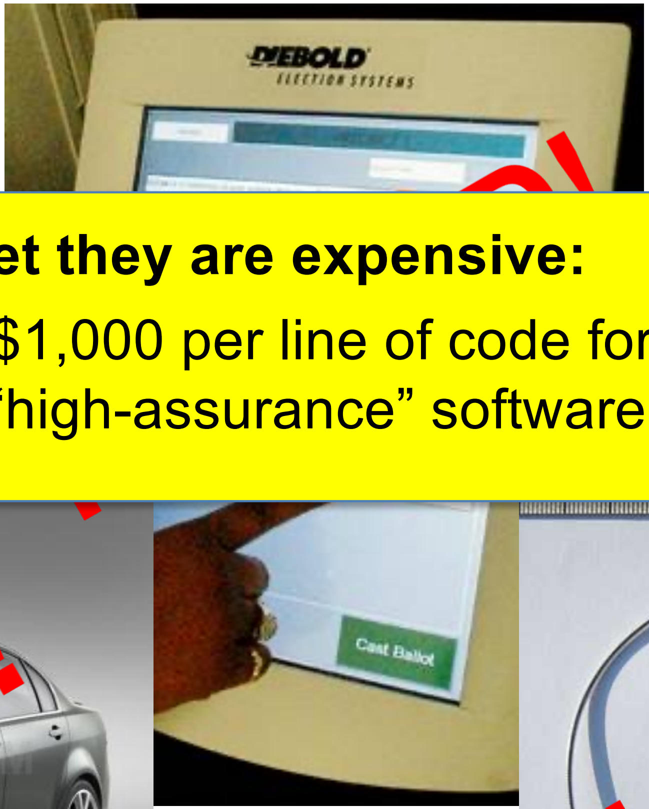
```
                           Windows

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!

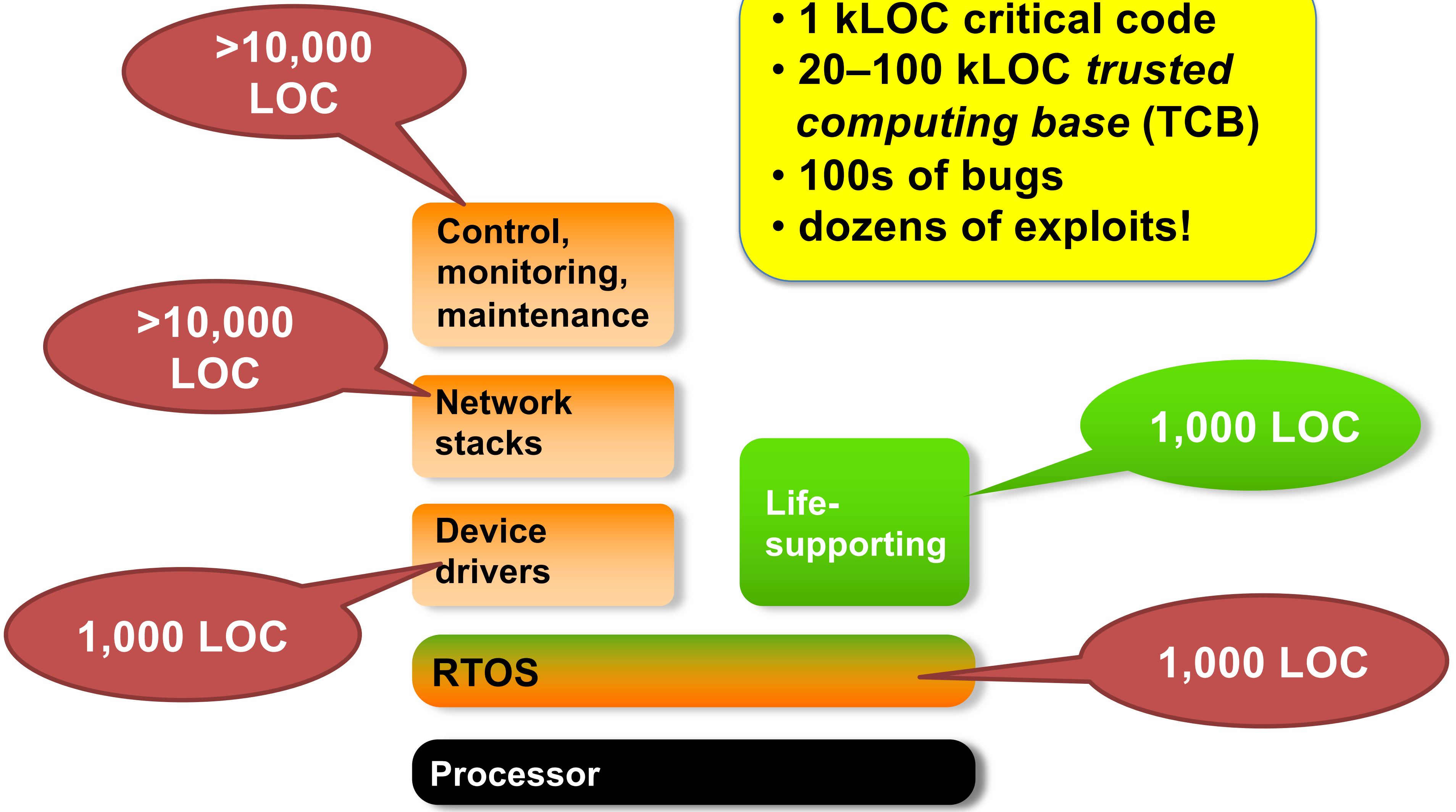


Yet they are expensive:

- $1,000 per line of code for "high-assurance" software!

Fundamental issue: large stacks, need isolation

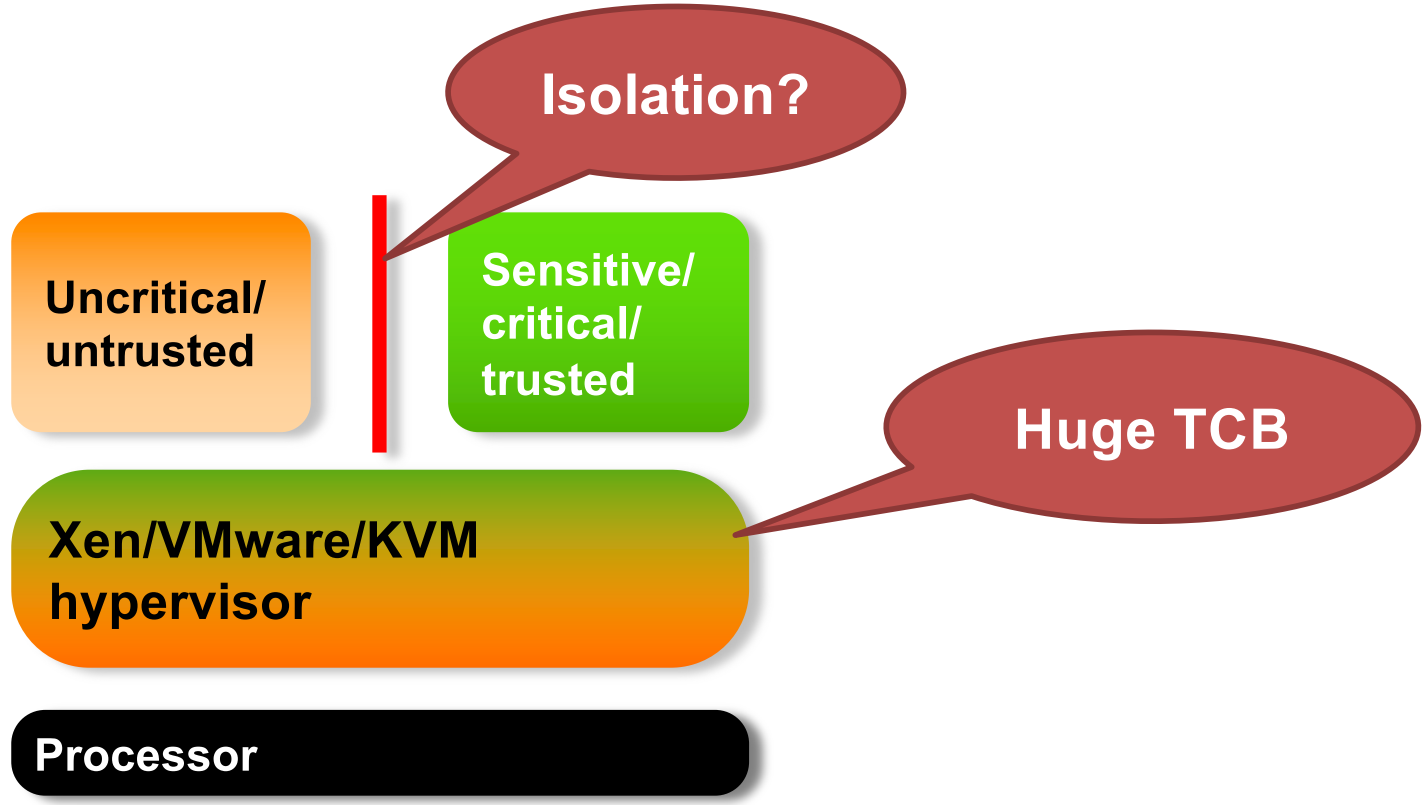
E.g. medical implant



- 1 kLOC critical code
- 20–100 kLOC *trusted computing base* (TCB)
- 100s of bugs
- dozens of exploits!

High Assurance *Bad* Practice

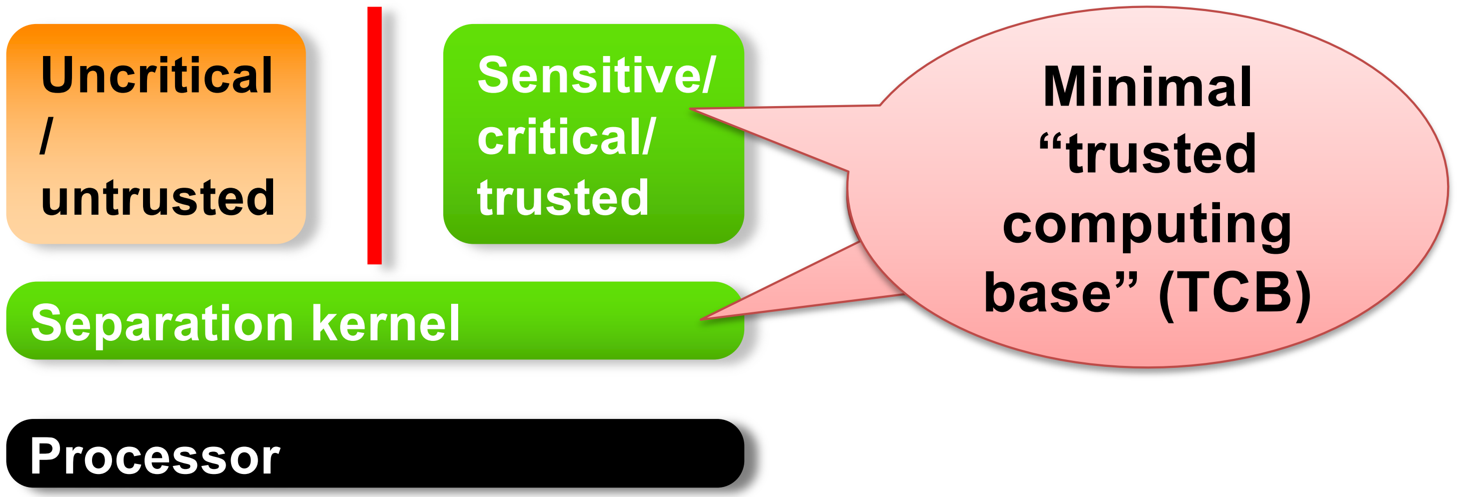
- TCB of millions of LOC
- Expect 1000s of bugs
- Expect 100s of vulnerabilities



High Assurance Best Practice

- Isolate
- Minimise the TCB
- Assure TCB by
    - testing
    - code inspection
    - bug-finding tools

Always incomplete!


Uncritical / untrusted
Sensitive/ critical/ trusted
Separation kernel
Processor
Minimal “trusted computing base” (TCB)

So, why don't we prove trustworthiness ?

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

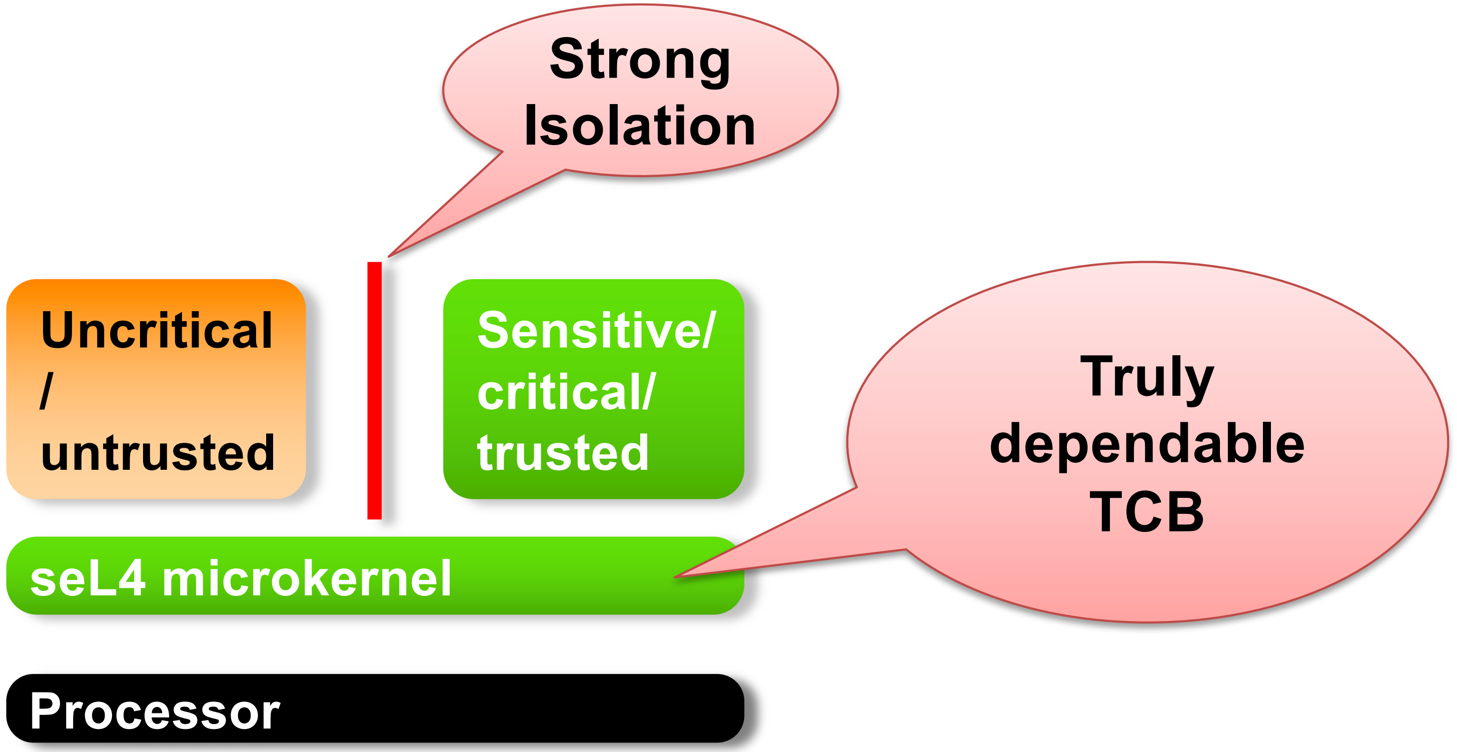
Corollary [with apologies to Dijkstra]:

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

State of the Art: NICTA's seL4 Microkernel

- Provable isolation!
- Provable assurance!

No place for bugs to hide!



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

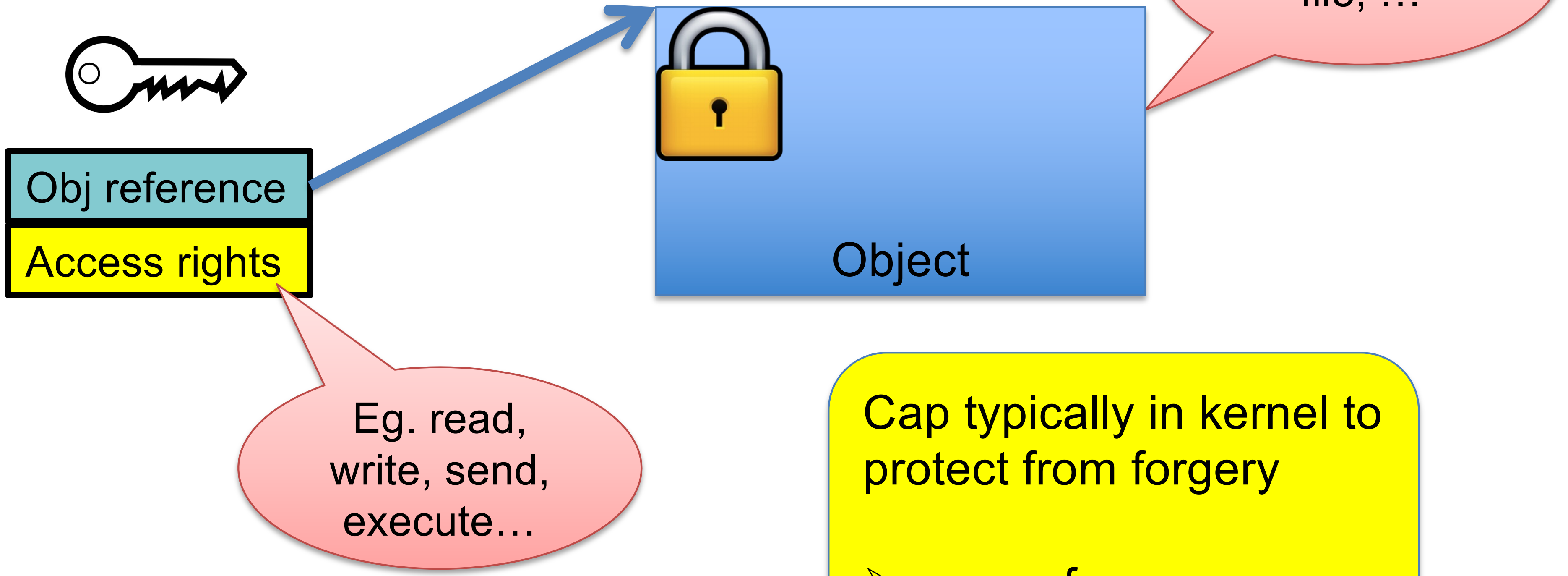
Real-time

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?

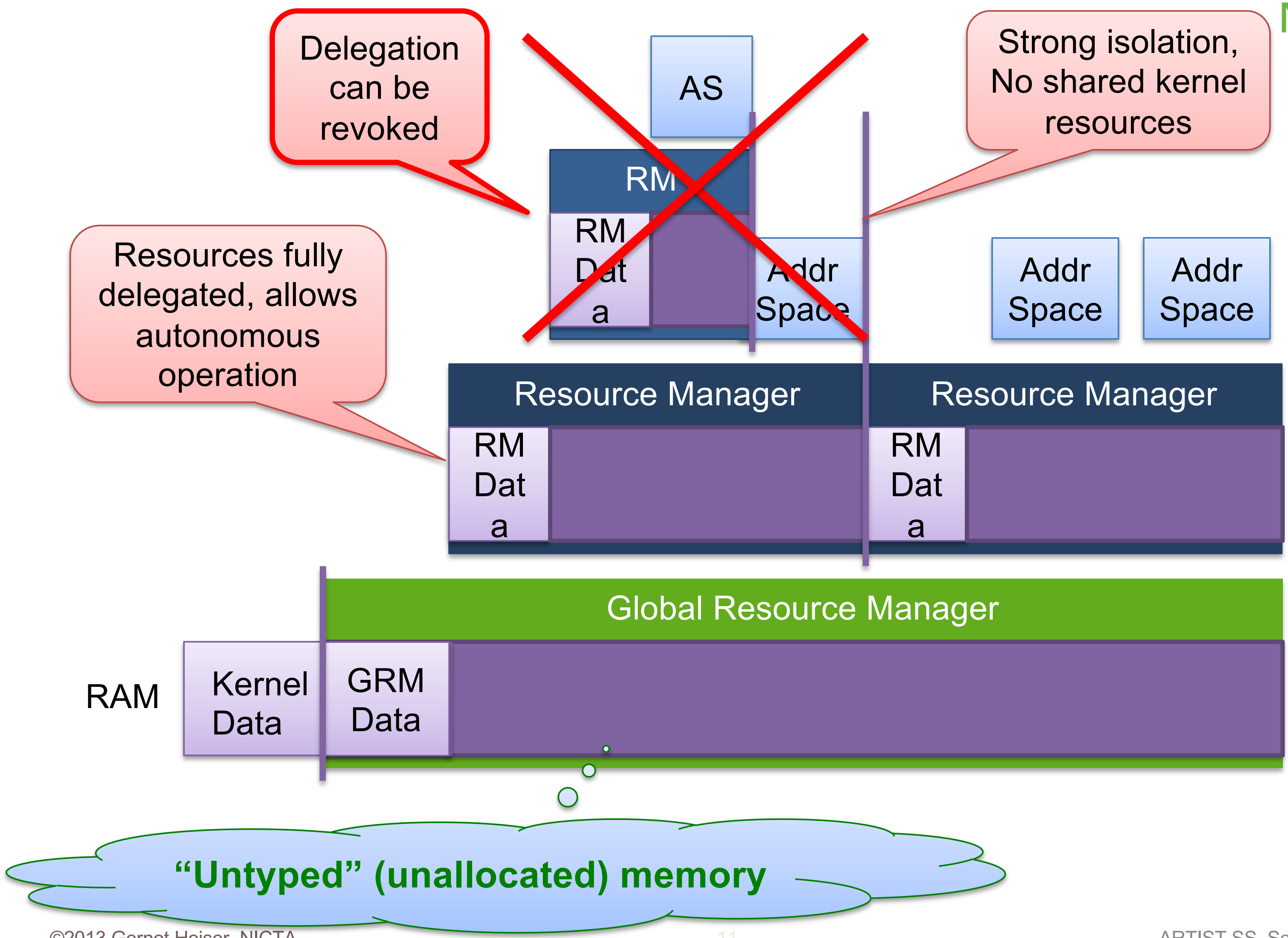
Cap = Access Token



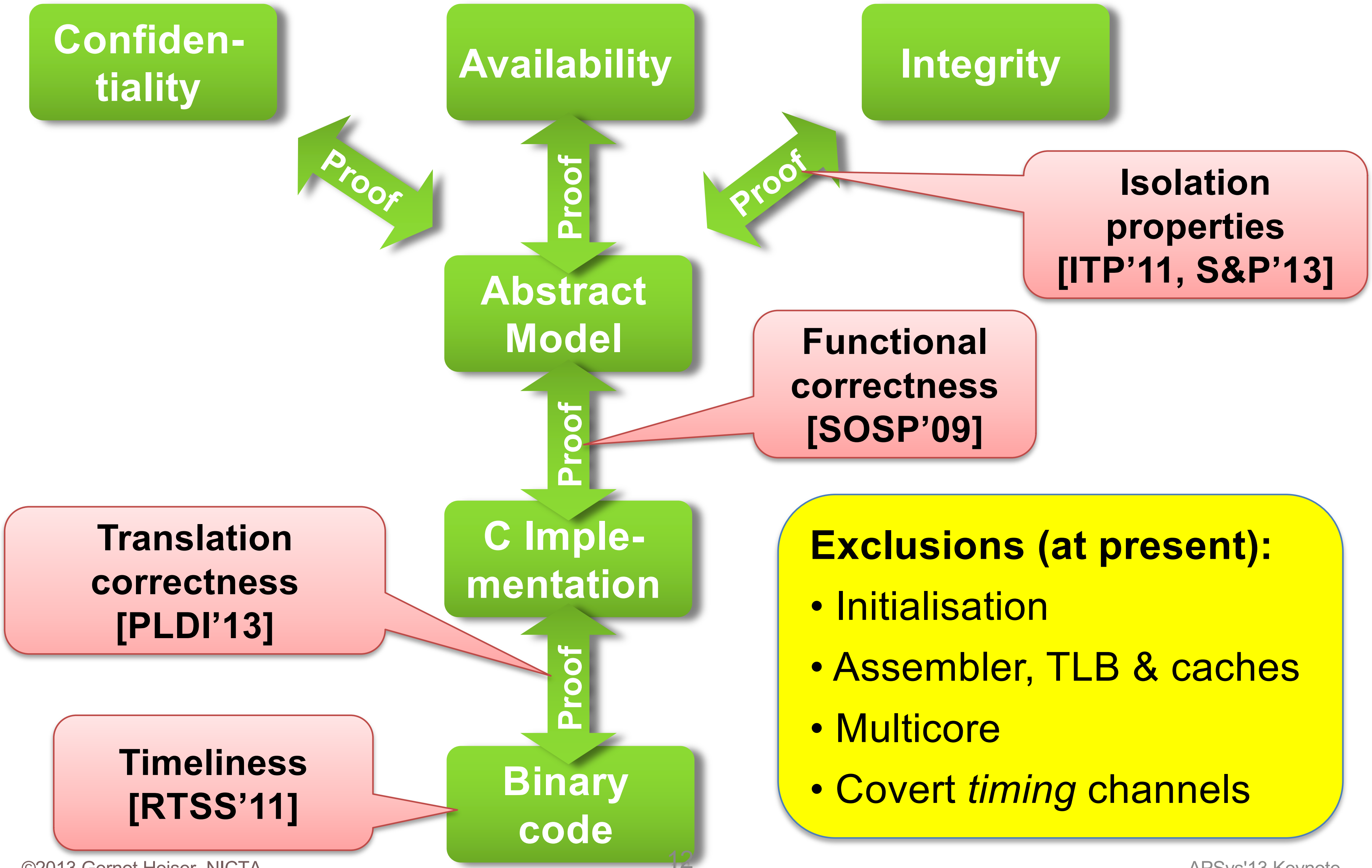
Cap typically in kernel to protect from forgery

- ➢ user references cap through handle

seL4 User-Level Memory Management


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
RM Dat a
Resource Manager
RM Dat a
Global Resource Manager
RAM
Kernel Data
GRM Data
"Untyped" (unallocated) memory

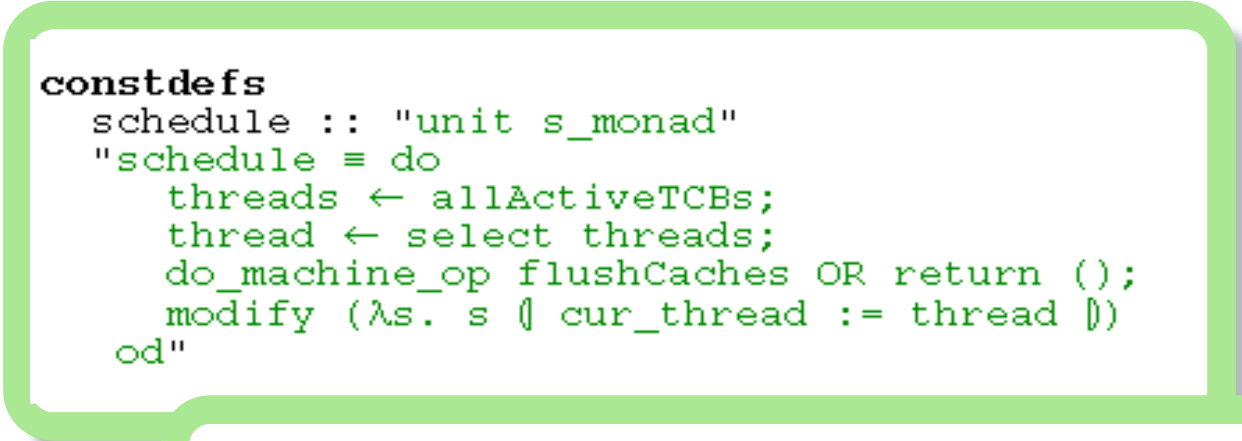
NICTA's seL4: Mathematical *Proof* of Isolation



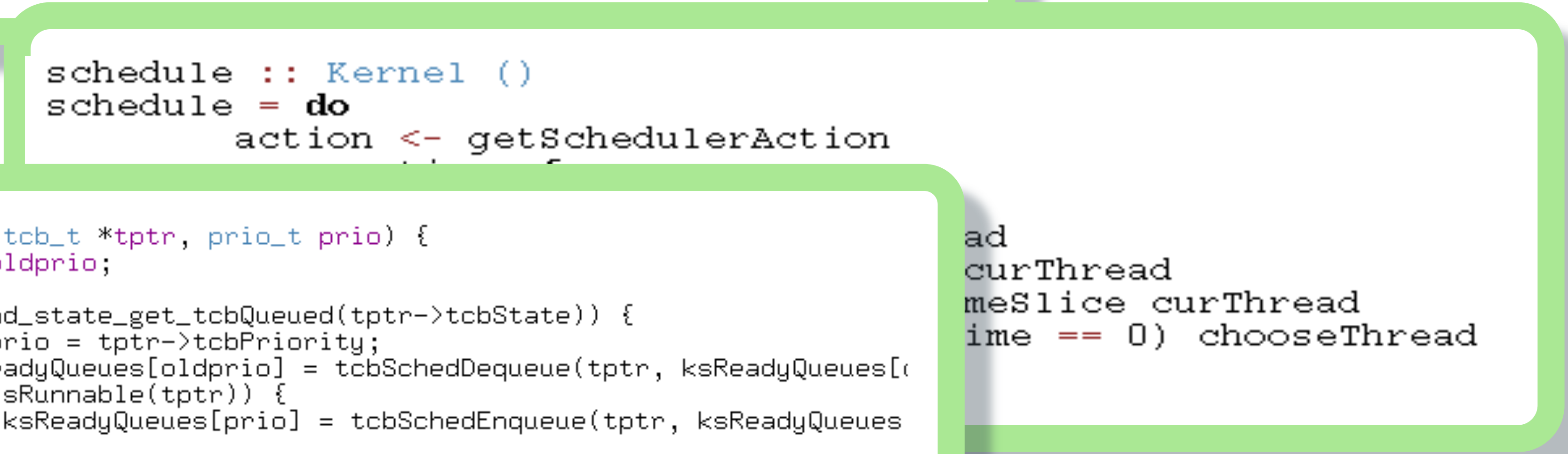
Exclusions (at present):

- Initialisation
- Assembler, TLB & caches
- Multicore
- Covert *timing* channels

Proving Functional Correctness

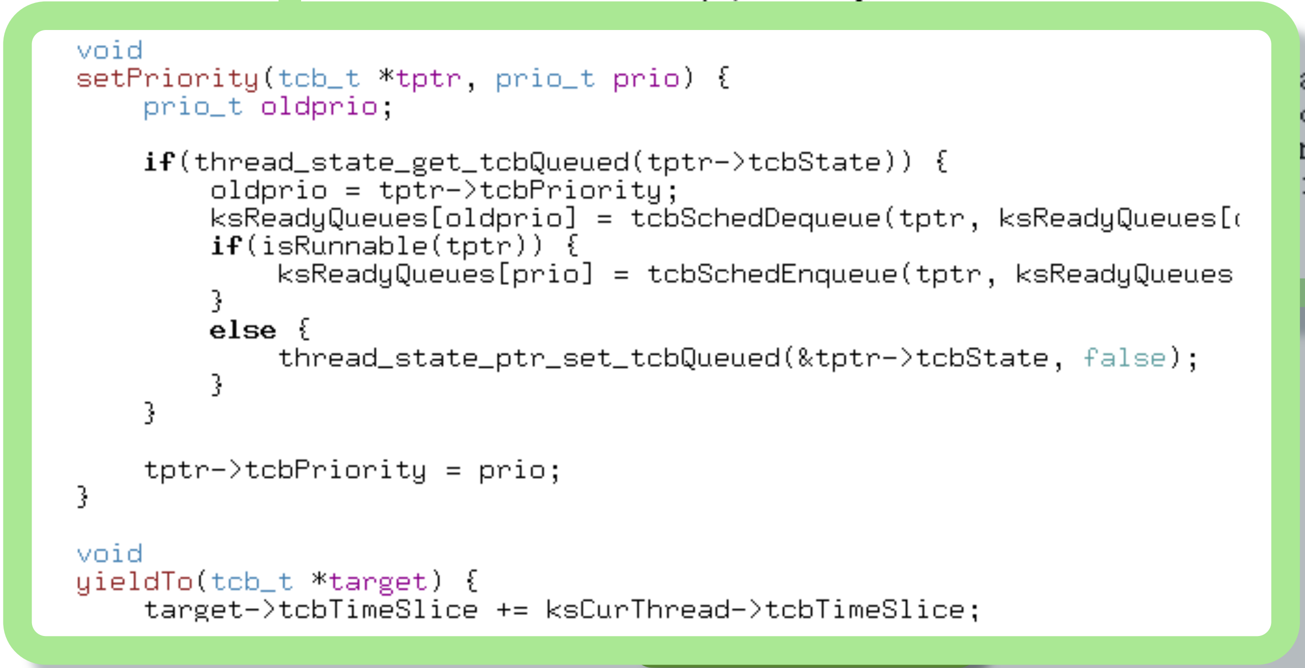


```
constdefs
  schedule :: "unit s_monad"
  "schedule ≡ do
     threads ← allActiveTCBs;
     thread ← select threads;
     do_machine_op flushCaches OR return ();
     modify (λs. s ⦇ cur_thread := thread ⦈)
   od"
```



```
schedule :: Kernel ()
schedule = do
        action <- getSchedulerAction
```

```
ad
curThread
meSlice curThread
ime == 0) chooseThread
```

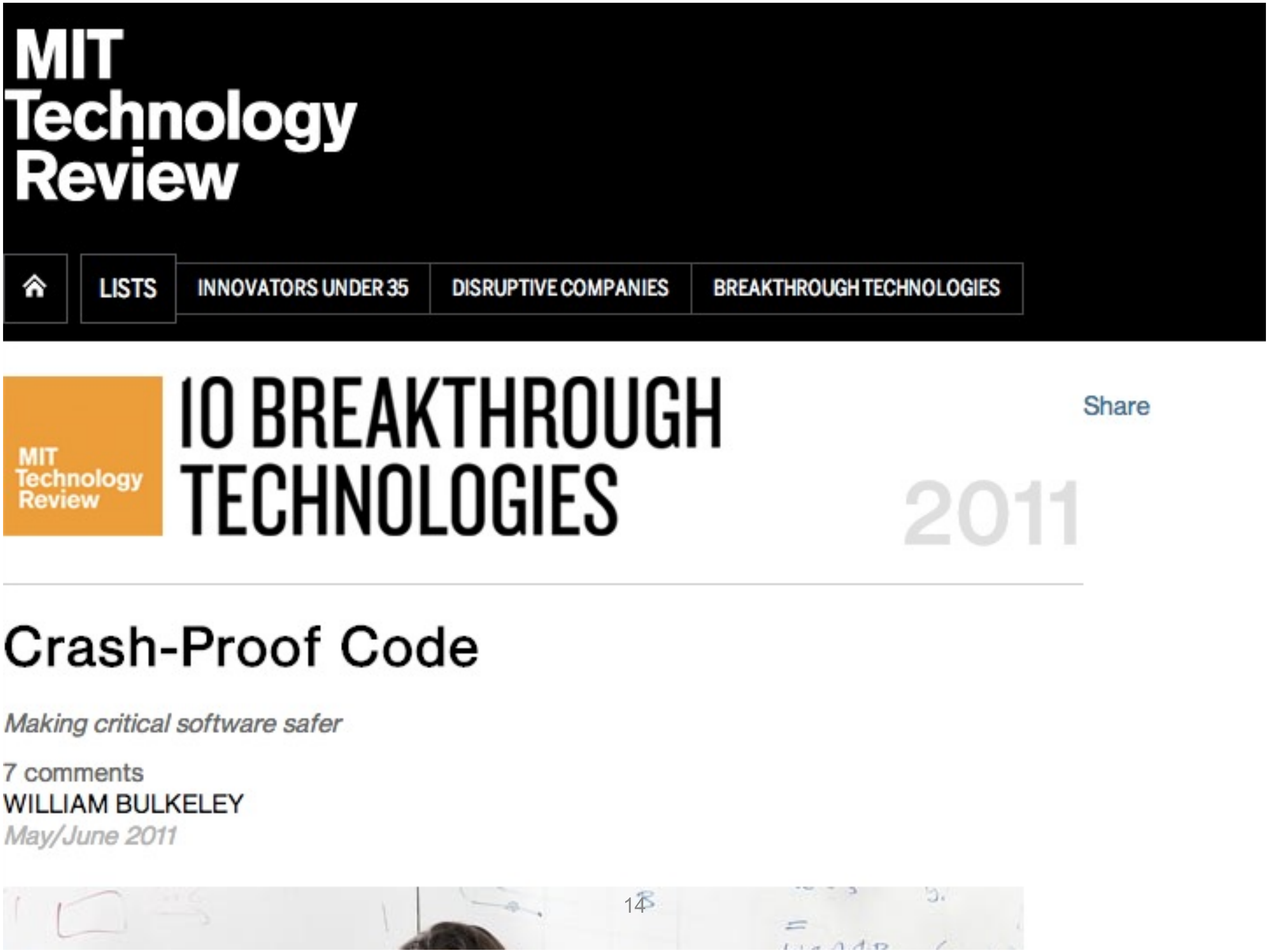


```
void
setPriority(tcb_t *tptr, prio_t prio) {
    prio_t oldprio;

    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;
```



MIT Technology Review

LISTS | INNOVATORS UNDER 35 | DISRUPTIVE COMPANIES | BREAKTHROUGH TECHNOLOGIES

10 BREAKTHROUGH TECHNOLOGIES 2011

Share

Crash-Proof Code

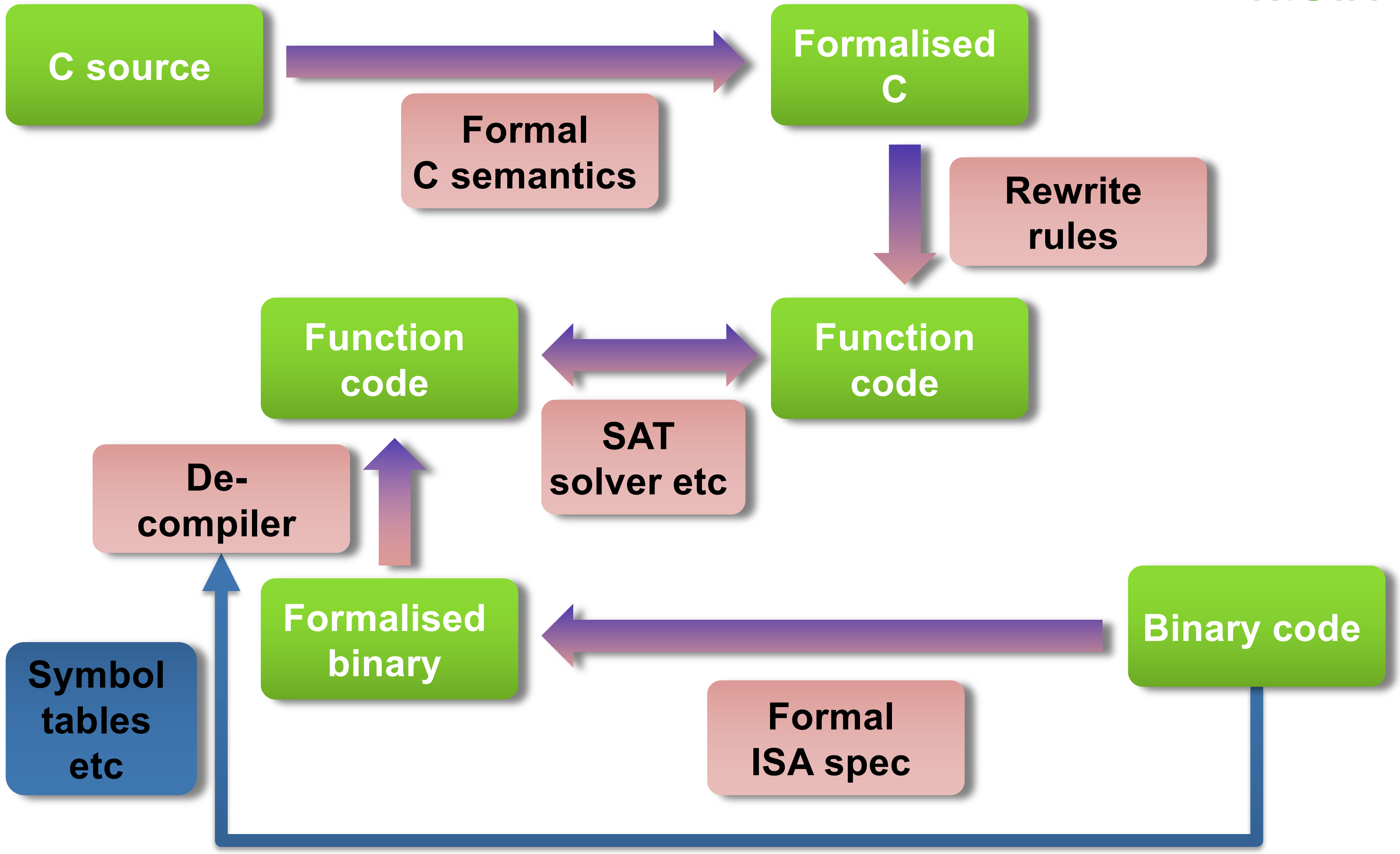
Making critical software safer

7 comments

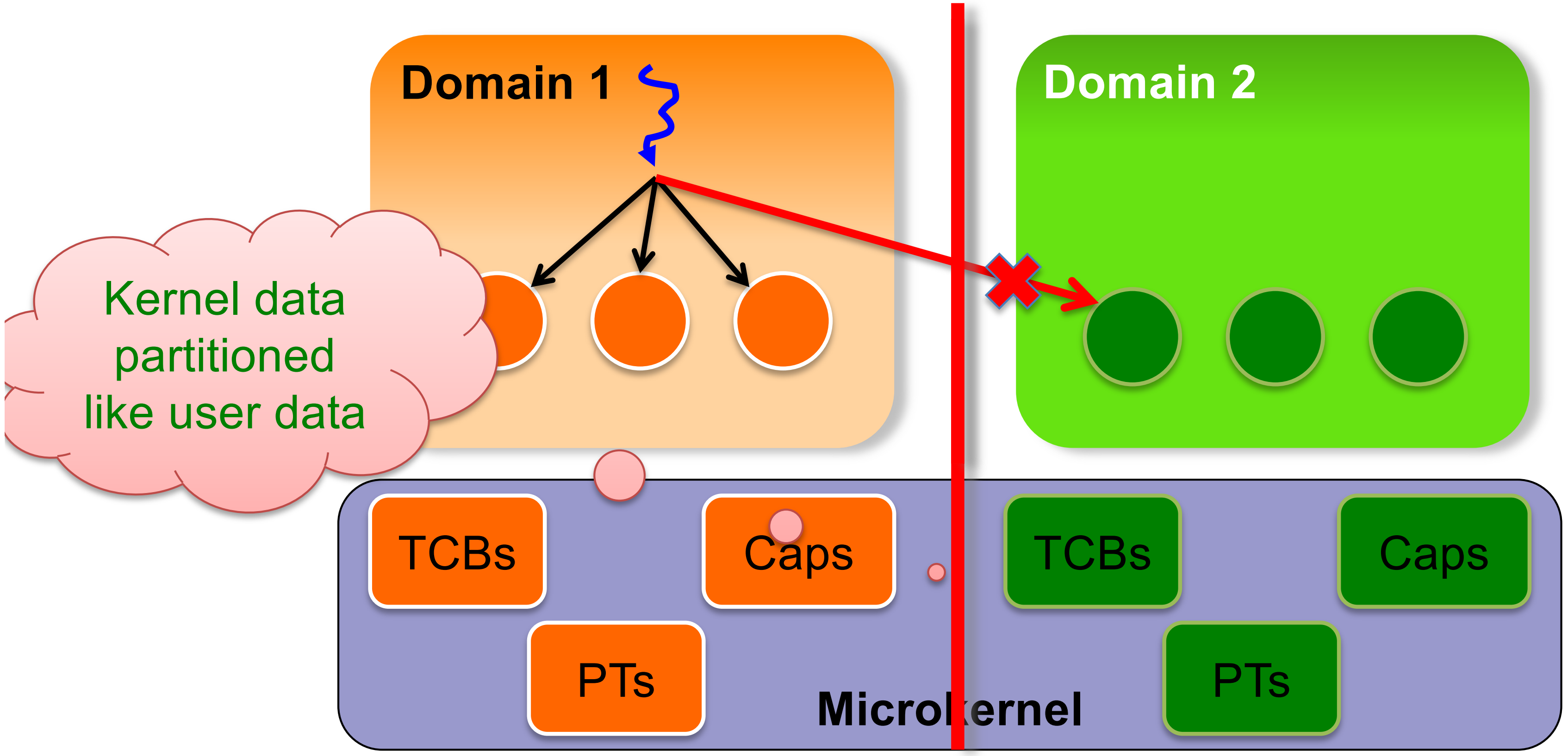
WILLIAM BULKELEY

May/June 2011

Binary Code Verification



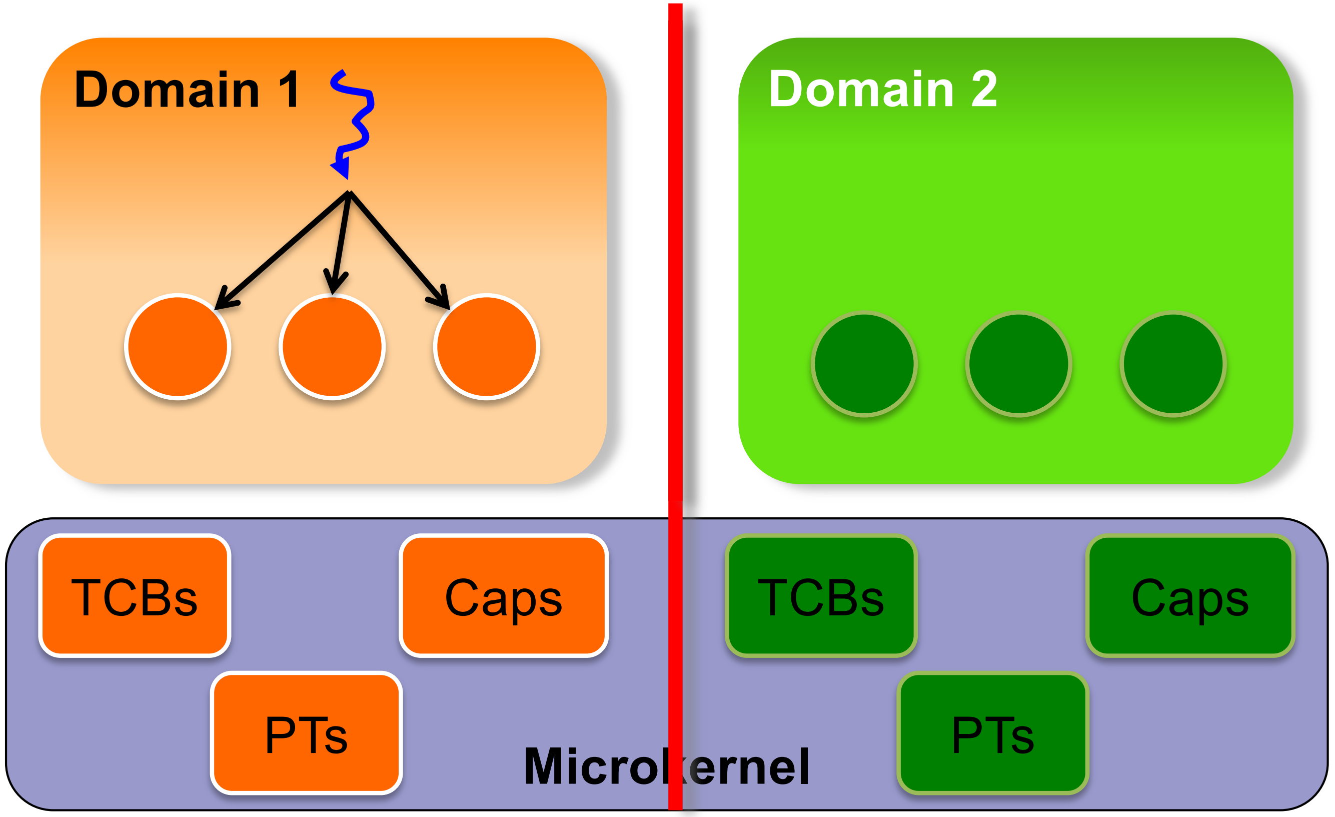
Integrity: Limiting Write Access



To prove:

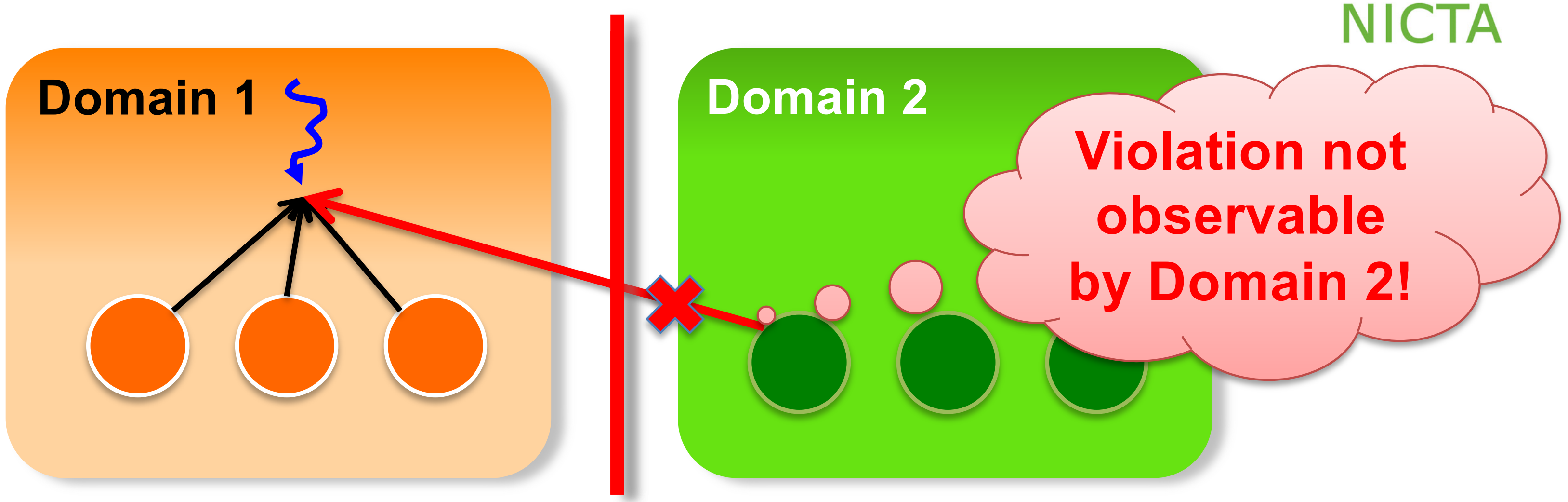
- Domain-1 doesn't have write *capabilities* to Domain-2 objects
  ⇒ no action of Domain-1 agents will modify Domain-2 state
- Specifically, *kernel does not modify on Domain-1's behalf!*
  - Event-based kernel operates on behalf of well-defined user thread
  - Prove kernel only allows write upon capability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

Availability: Ensuring Resource Access



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

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:

- Evolution of Domain 1 does not depend on Domain-2 state
- Also shows absence of covert storage channels

NICTA's seL4 Microkernel: Unique Assurance

First and only operating-system with functional-correctness proof: operation is always according to specification

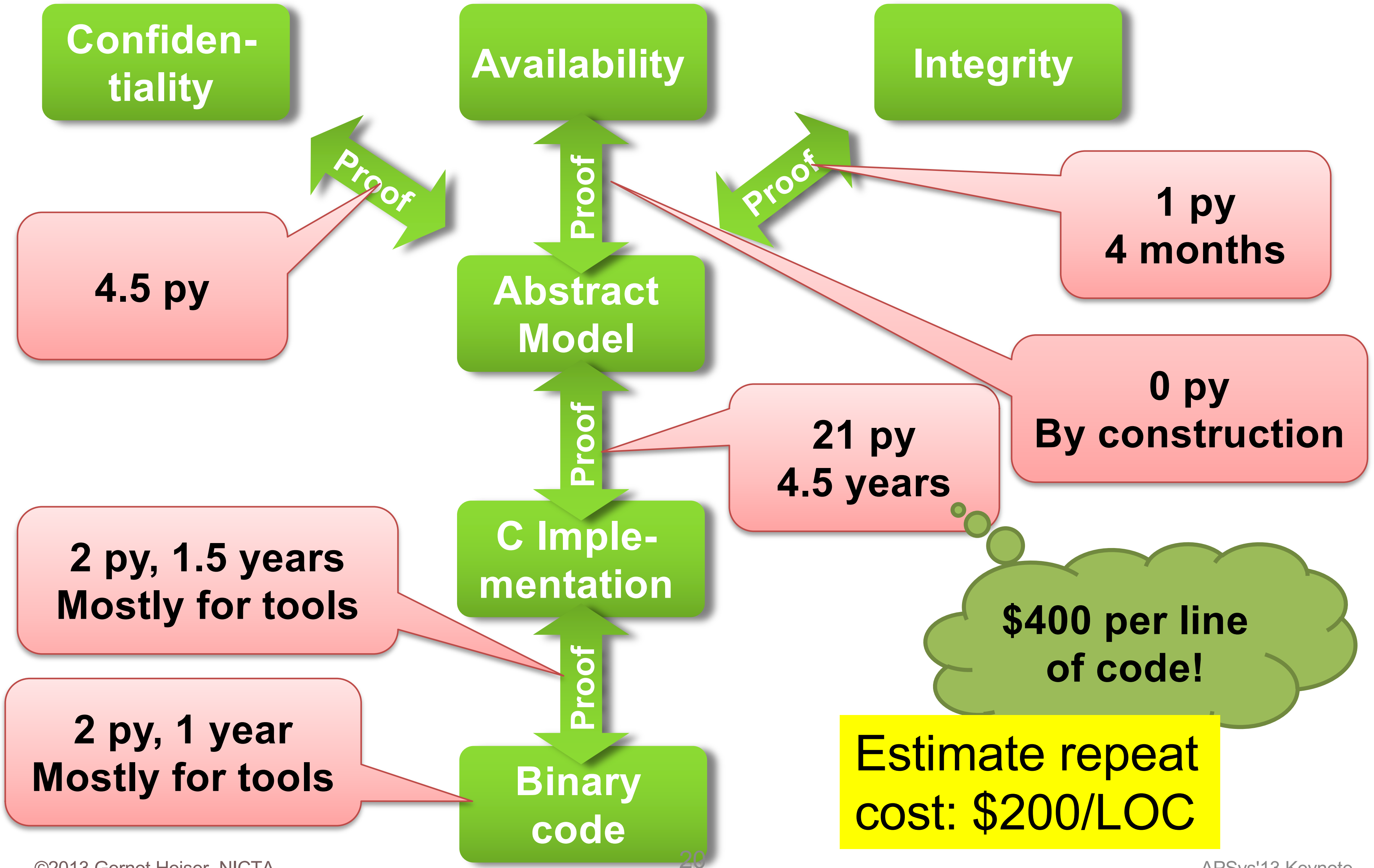
Predecessor deployed on 2 billion devices

First and only operating-system with *proof* of integrity and confidentiality enforcement – at the level of binary code!

World's fastest microkernel on ARM architecture

First and only protected-mode operating-system with complete and sound timing analysis

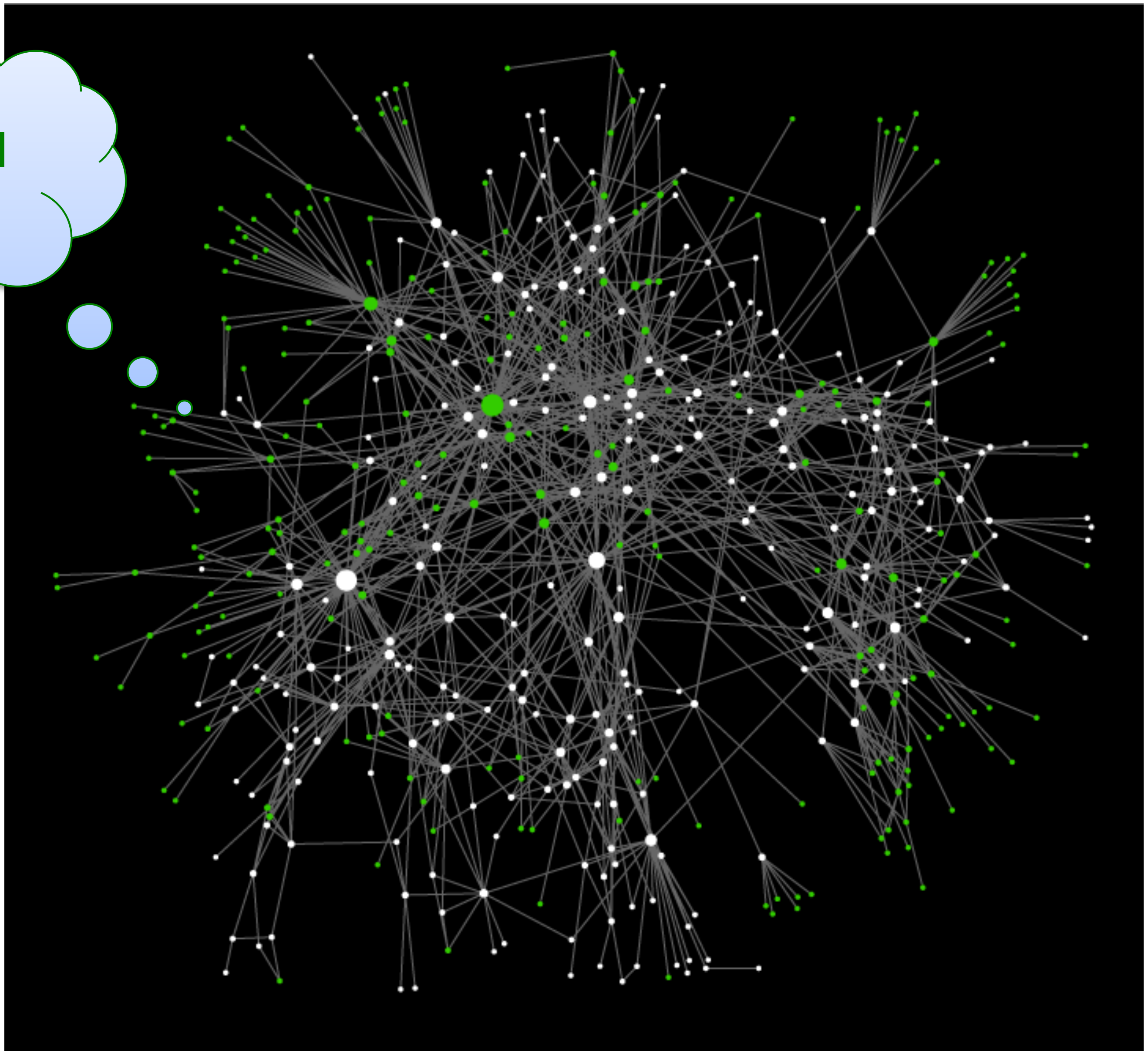
seL4: Cost of Assurance



Estimate repeat cost: $200/LOC

Why 21 py for 9,000 LOC?

seL4 call graph



Costs Breakdown

Haskell design	2 py
C implementation	2 months
Debugging/Testing	2 months
Kernel verification	11.5 py
Formal frameworks	9 py
Total	**21 py**
Repeat (estimated)	6 py
Traditional engineering	4–6 py

Including subsequent fastpath verification

Did you find bugs???

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

Cost of Assurance

Industry Best Practice:

- "High assurance": $1,000/LOC, no guarantees, *unoptimised*
- Low assurance: $100–200/LOC, 1–5 faults/kLOC, *optimised*

State of the Art – seL4:

  - $400/LOC, 0 faults/kLOC, *optimised*

- Estimate repeat would cost half
  - that's about the development cost of the predecessor Pistachio!
- Aggressive optimisation [APSys'12]
  - much faster than traditional high-assurance kernels
  - as fast as best-performing low-assurance kernels

What Have We Learnt?

Formal verification *probably* didn't produce a more *secure* kernel

- In reality, traditional separation kernels are *probably* secure

But:

- We now have certainty
- We did it *probably* at less cost

Real achievement:

- Cost-competitive at a scale where traditional approaches still work
- Foundation for scaling beyond: **2 × cheaper, 10 × bigger!**

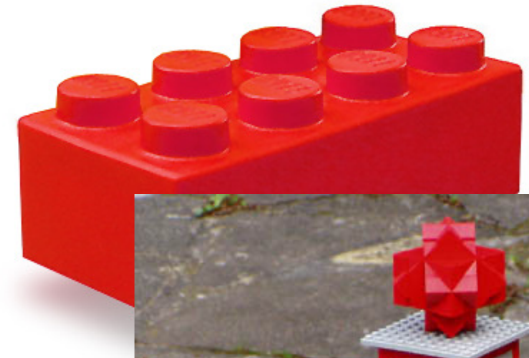
How?

- Combine theorem proving with
  - synthesis
  - domain–specific languages (DSLs)

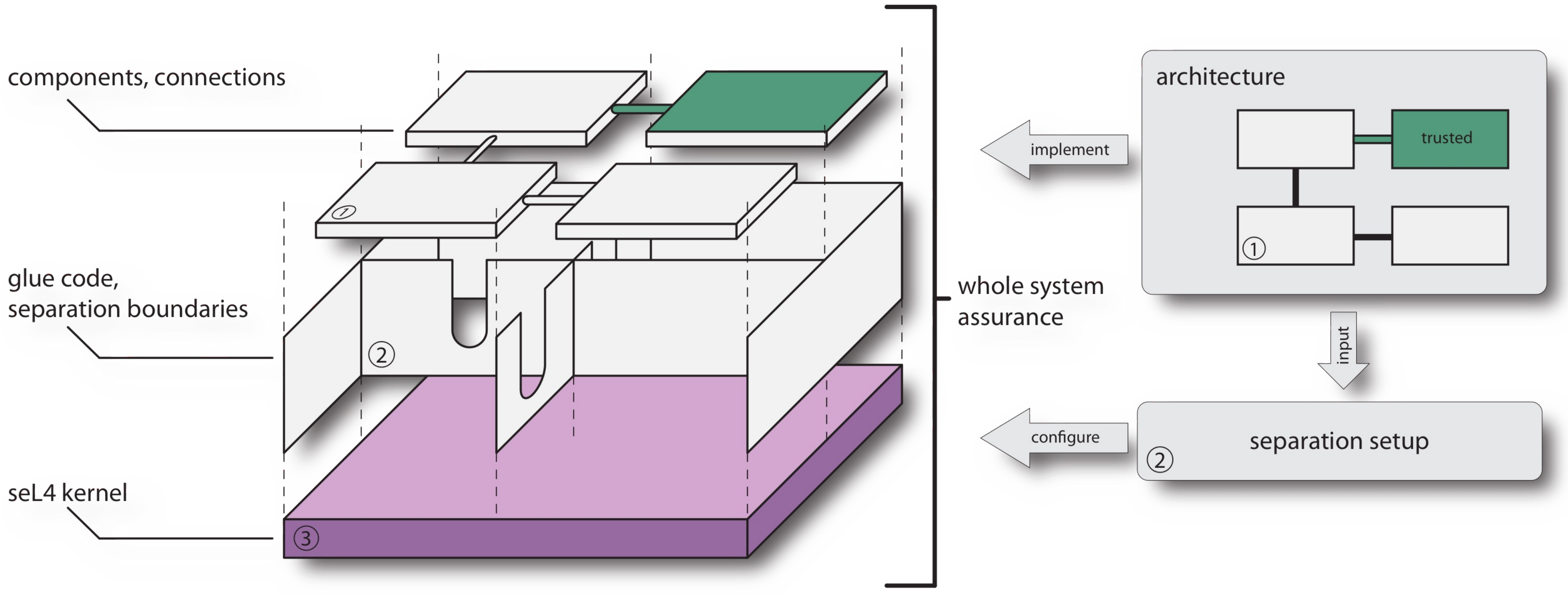
Phase Two: Full-System Guarantees

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

- Next step: Guarantees for real-world systems (10,000,000 LOC, <100,000 verified)



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

Next Step: Full System Assurance

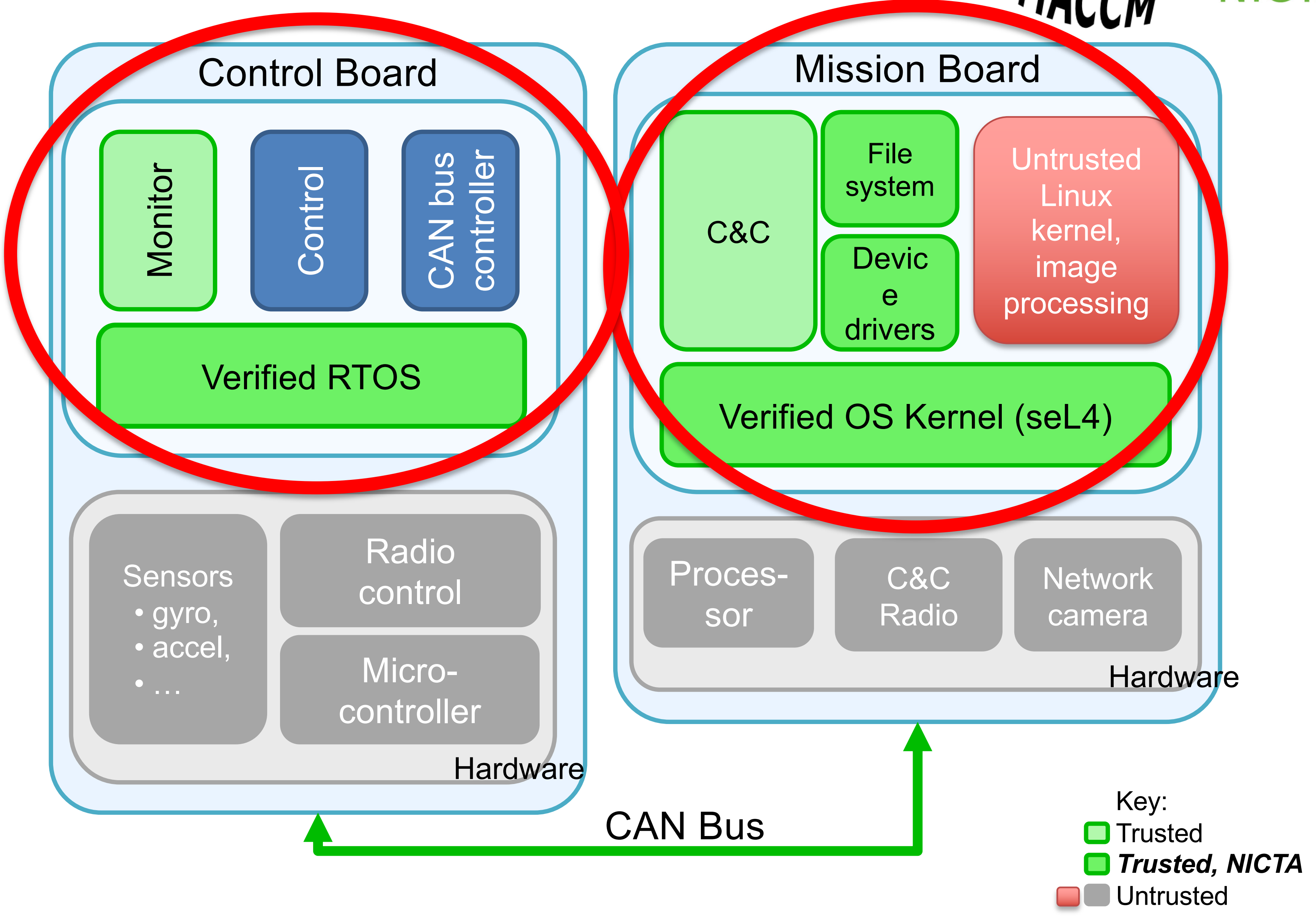
DARPA HACMS Program:

- Provable vehicle safety
- "Red Team" must not be able to divert vehicle

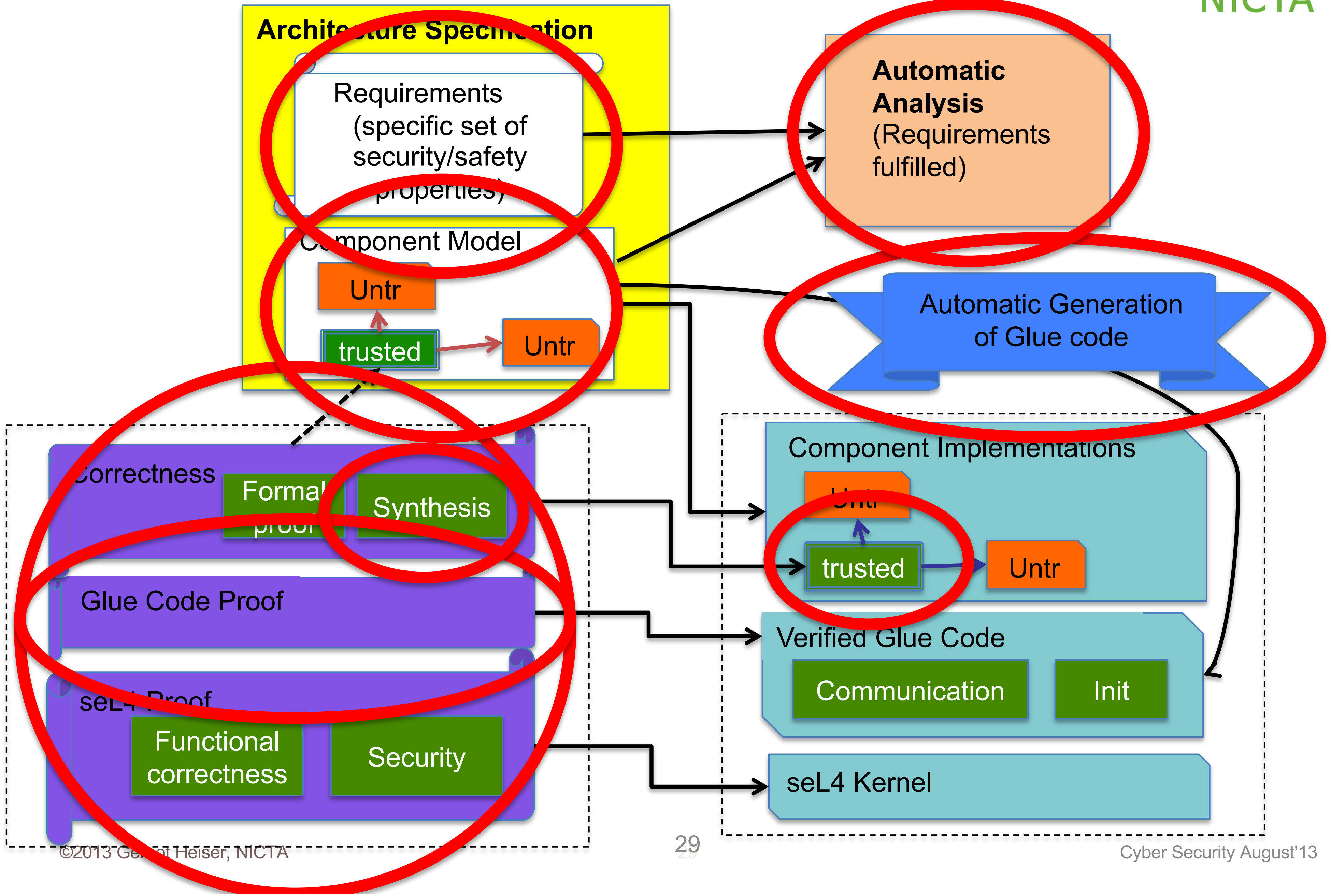
Boeing Unmanned Little Bird (AH-6) Deployment Vehicle

SMACCMcopter Research Vehicle

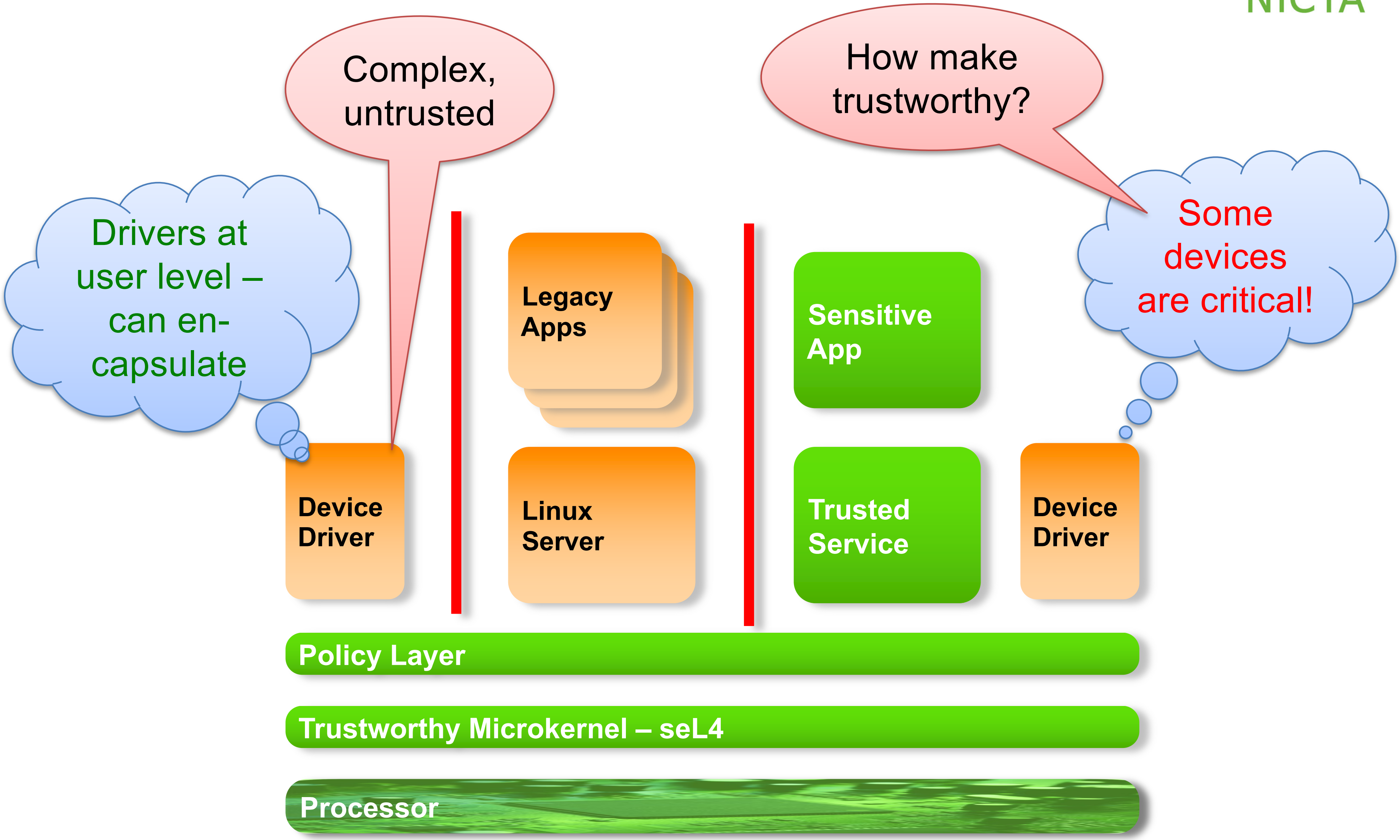
SMACCMcopter System Structure



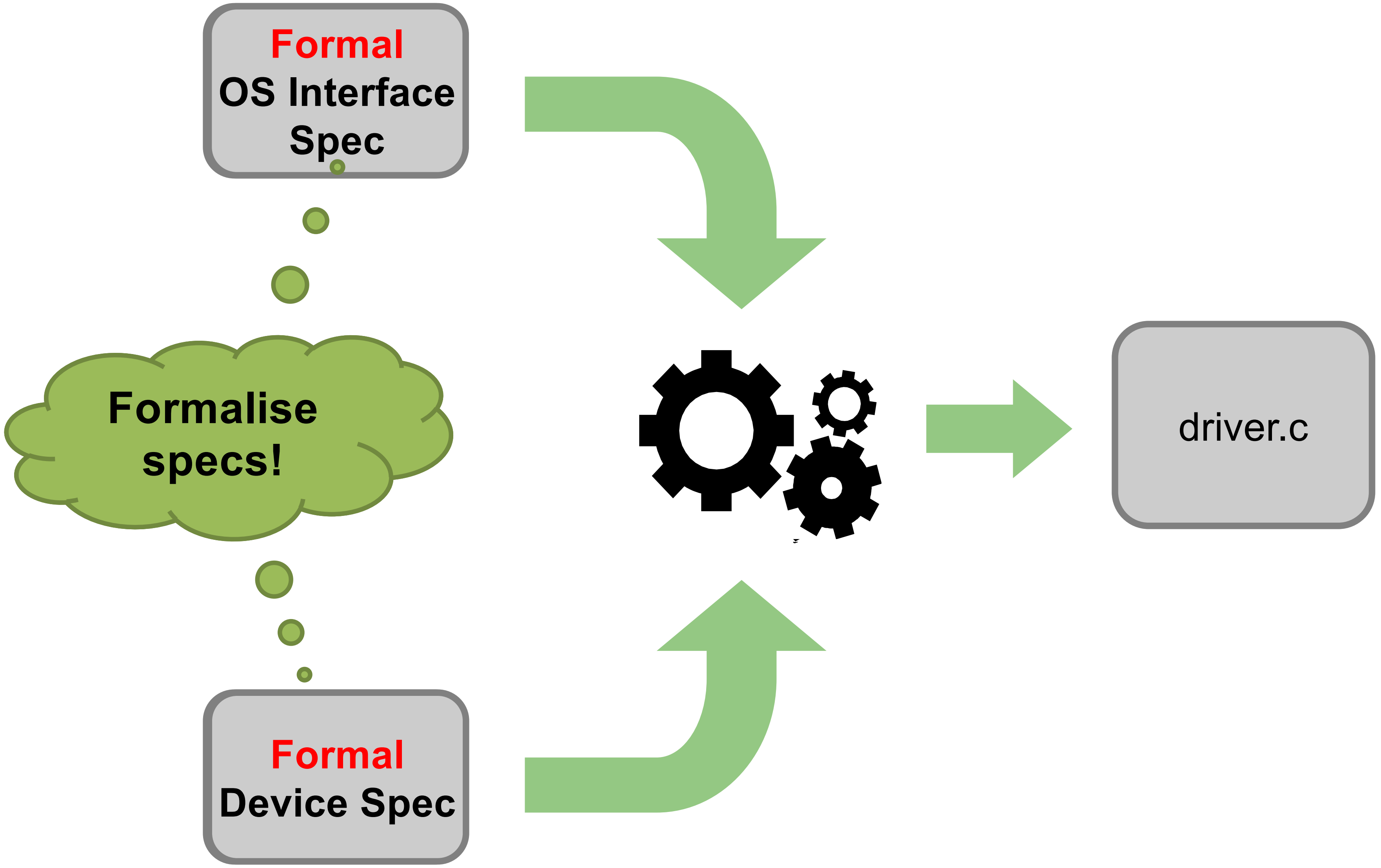
Architecting System-Level Security/Safety


Architecture Specification
Requirements (specific set of security/safety properties)
Component Model
Untr
trusted
Untr
Automatic Analysis (Requirements fulfilled)
Automatic Generation of Glue code
Correctness
Formal proof
Synthesis
Glue Code Proof
seL4 Proof
Functional correctness
Security
Component Implementations
Untr
trusted
Untr
Verified Glue Code
Communication
Init
seL4 Kernel

Device Drivers



Synthesis: Device Drivers [SOSP'09]



Actually works! (On Linux & seL4)

IDE disk controller

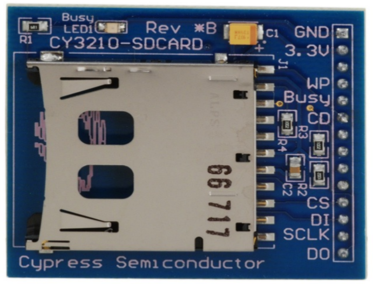
W5100 Eth shield

Intel PRO/1000 Ethernet

Working on proving correctness

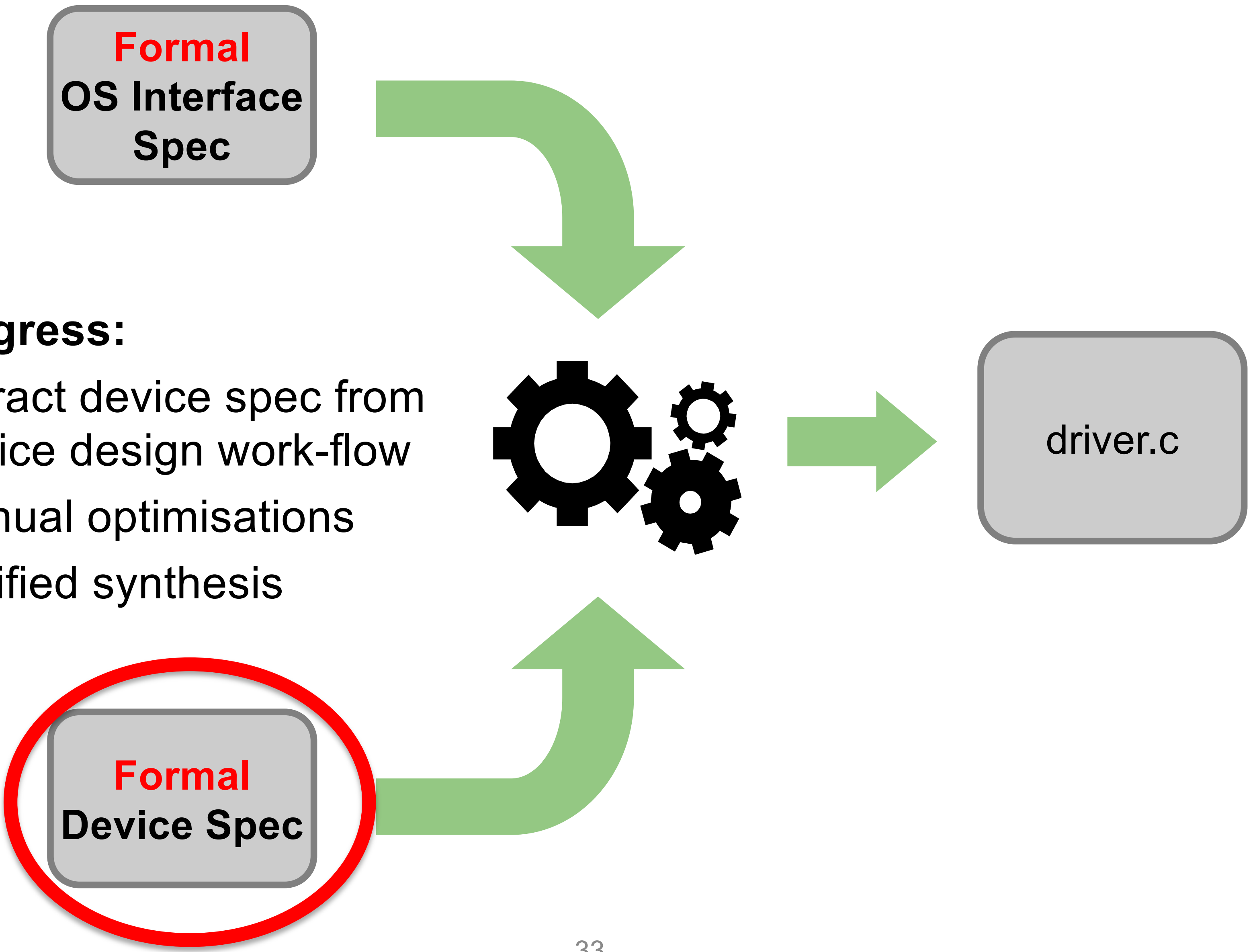
UART controller

Asix AX88772 USB-to-Eth adapter



SD host controller

Synthesis: Device Drivers



In progress:

- Extract device spec from device design work-flow
- Manual optimisations
- Verified synthesis

Hardware Design Workflow

Informal specification

⇩

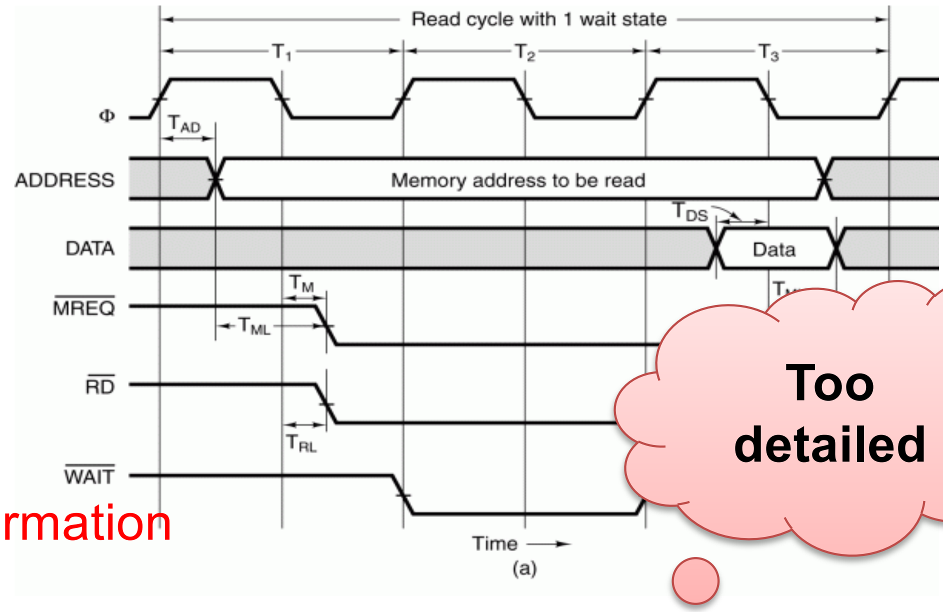
High-level model

⇩ Manual transformation

Register-transfer-level description

⇩

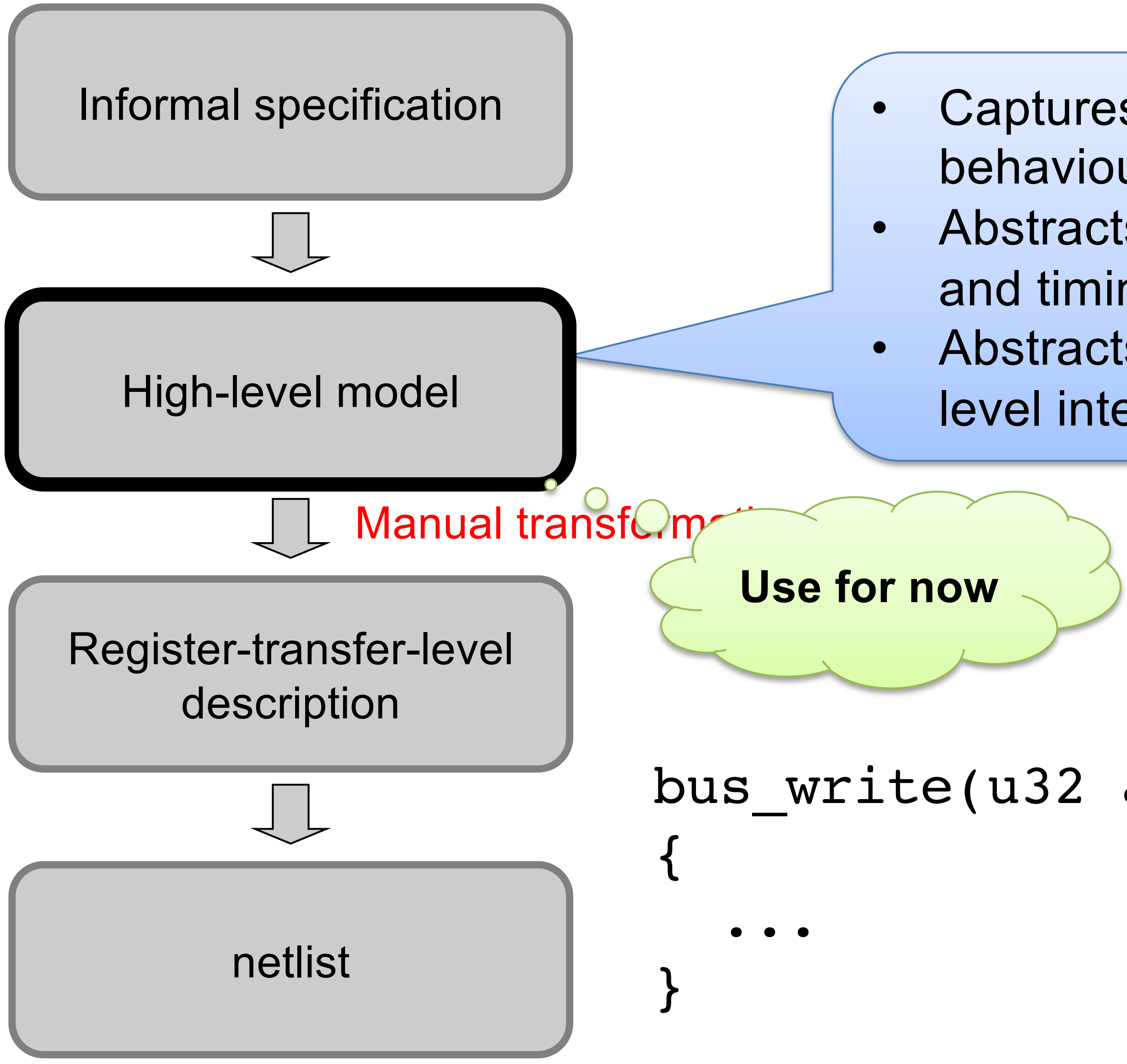
netlist



Too detailed

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

Hardware Design Workflow



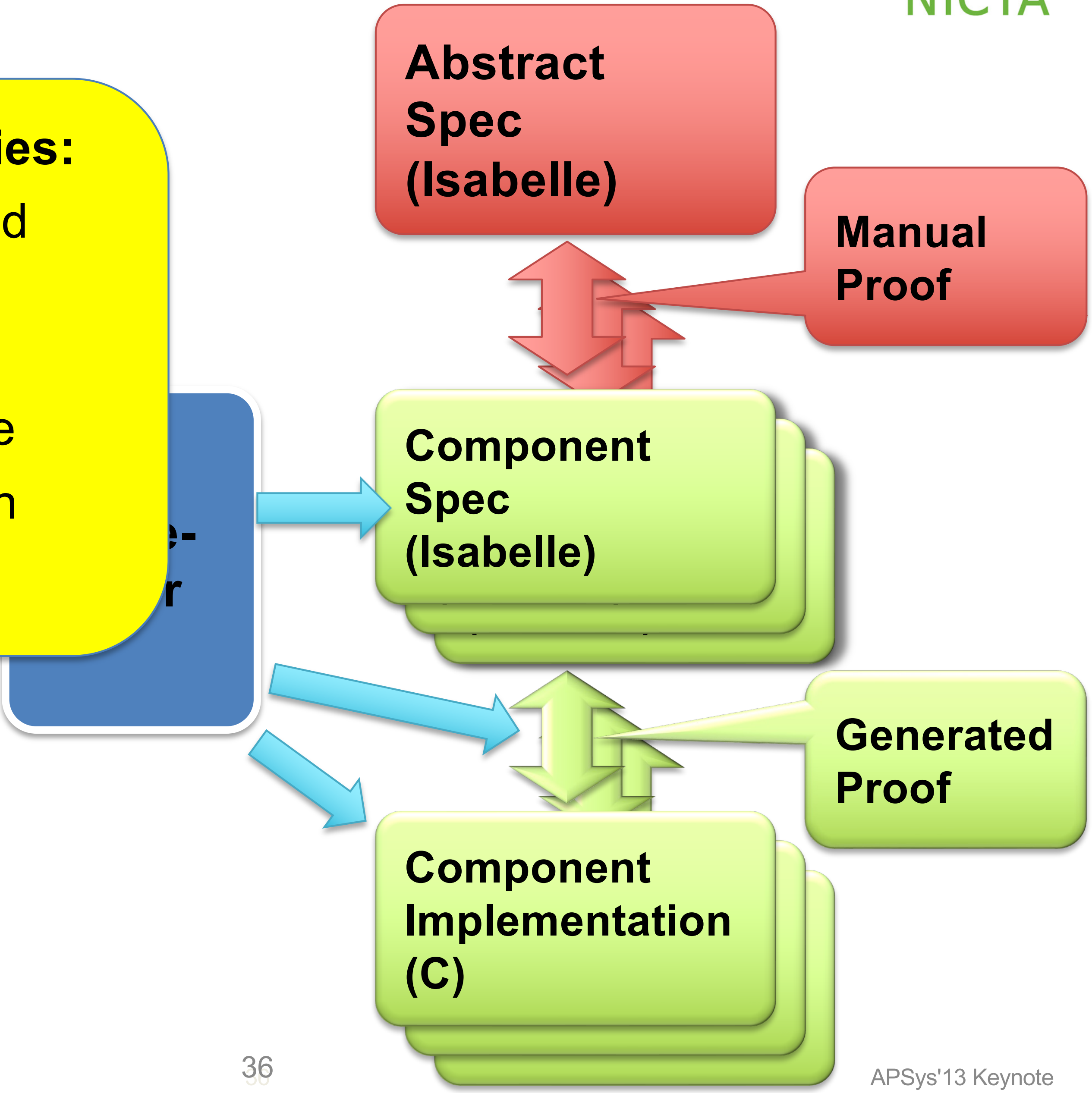
- Captures external behaviour
- Abstracts away structure and timing
- Abstracts away the low-level interface

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

DSLs: File System

File-system properties:

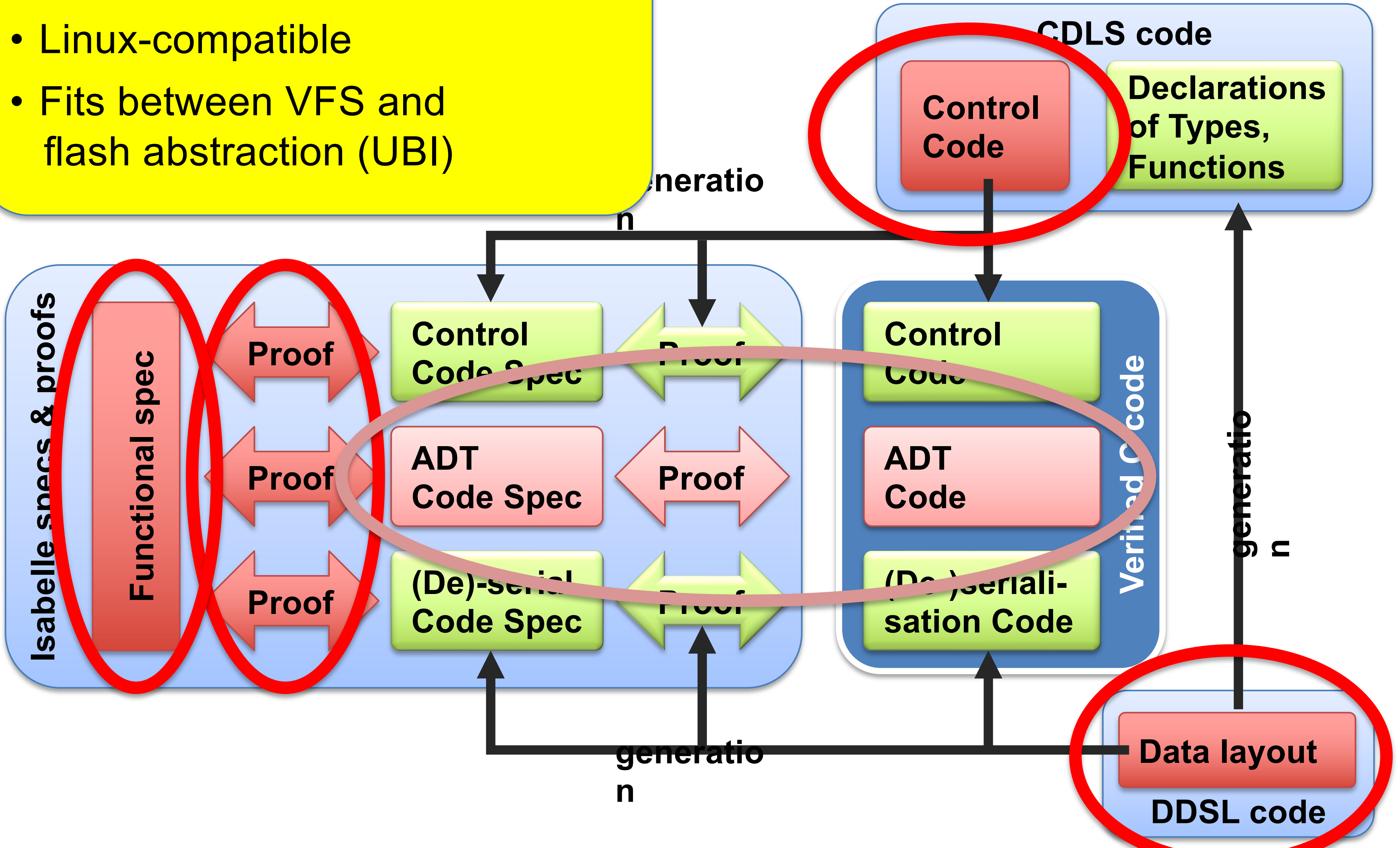
- Multiple, pre-defined abstraction levels
- Naturally modular
- Lots of “boring” code
    - (de-)serialisation
    - error handling



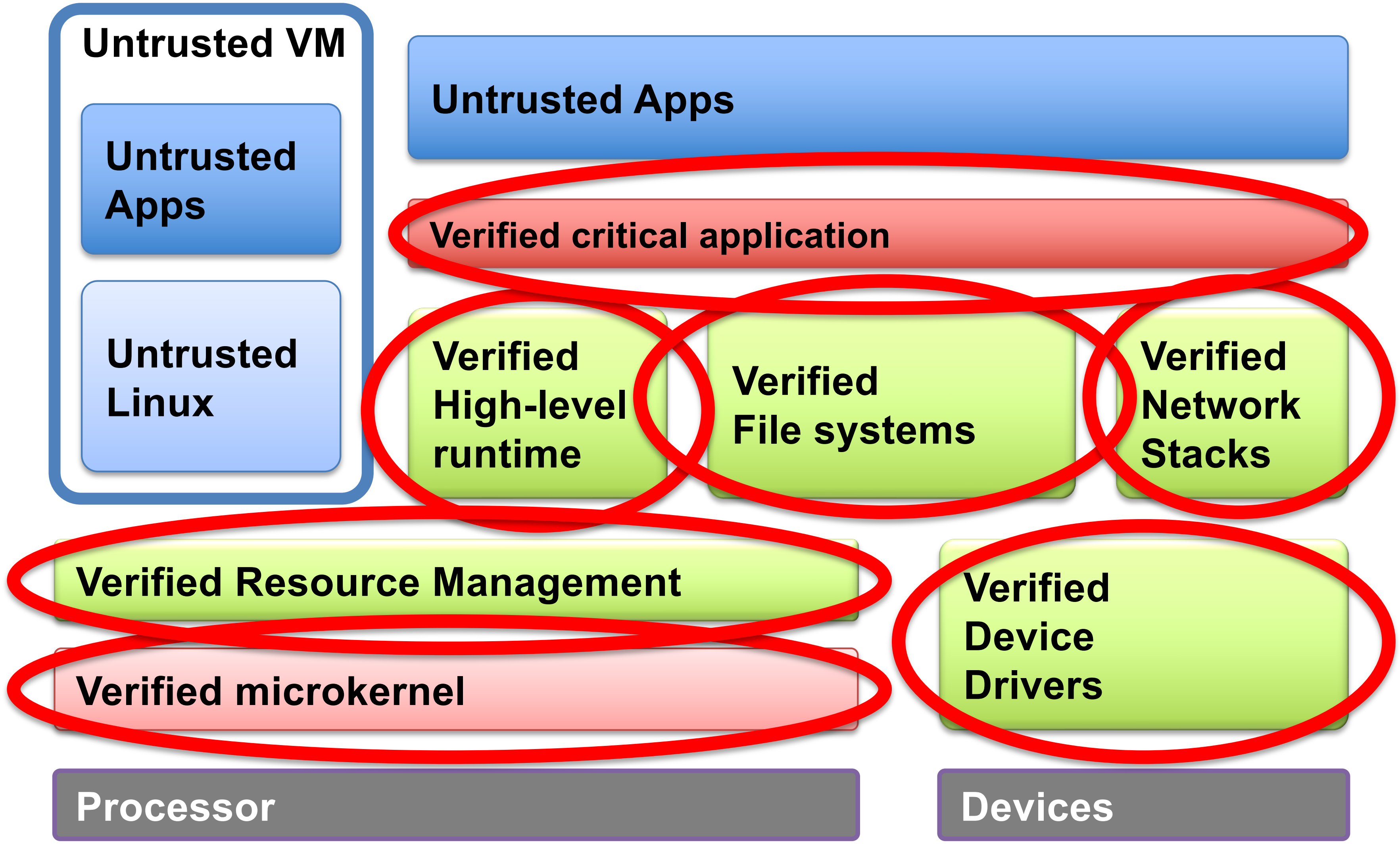
File System Code and Proof Co-Generation

Case study: Flash file system

- Linux-compatible
- Fits between VFS and flash abstraction (UBI)



Future: Full-Scale Trustworthy System



Lessons Learnt So Far

Formal methods are expensive?

- Cost-effective for high assurance on small to moderate scale
- $200-400/LOC for 10kLOC

We think we can scale bigger and cheaper:

- Componentisation
  - verify components in isolation – enabled by seL4 guarantees
  - cost – performance tradeoff
- Synthesis
- Abstraction: DSLs, HLLs increase productivity

google: “NICTA trustworthy”

mailto: gernot@nicta.com.au