NICTA

Towards *Verified* Real-World 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

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 challenge: Complexity

Our Vision: Trustworthy Systems

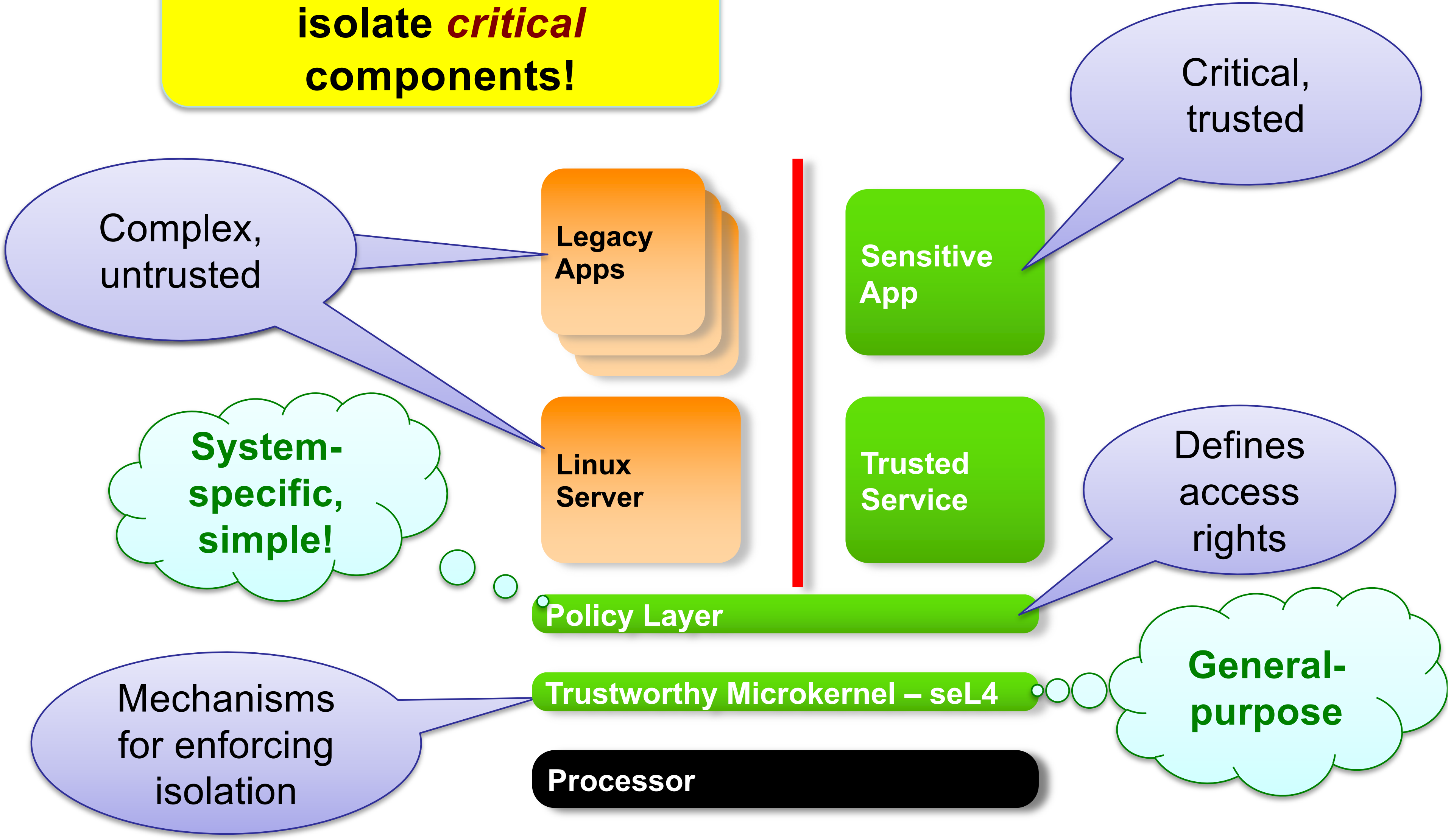
We will change the *practice* of designing and implementing critical systems, using rigorous approaches to achieve *true trustworthiness*

Suitable for real-world systems

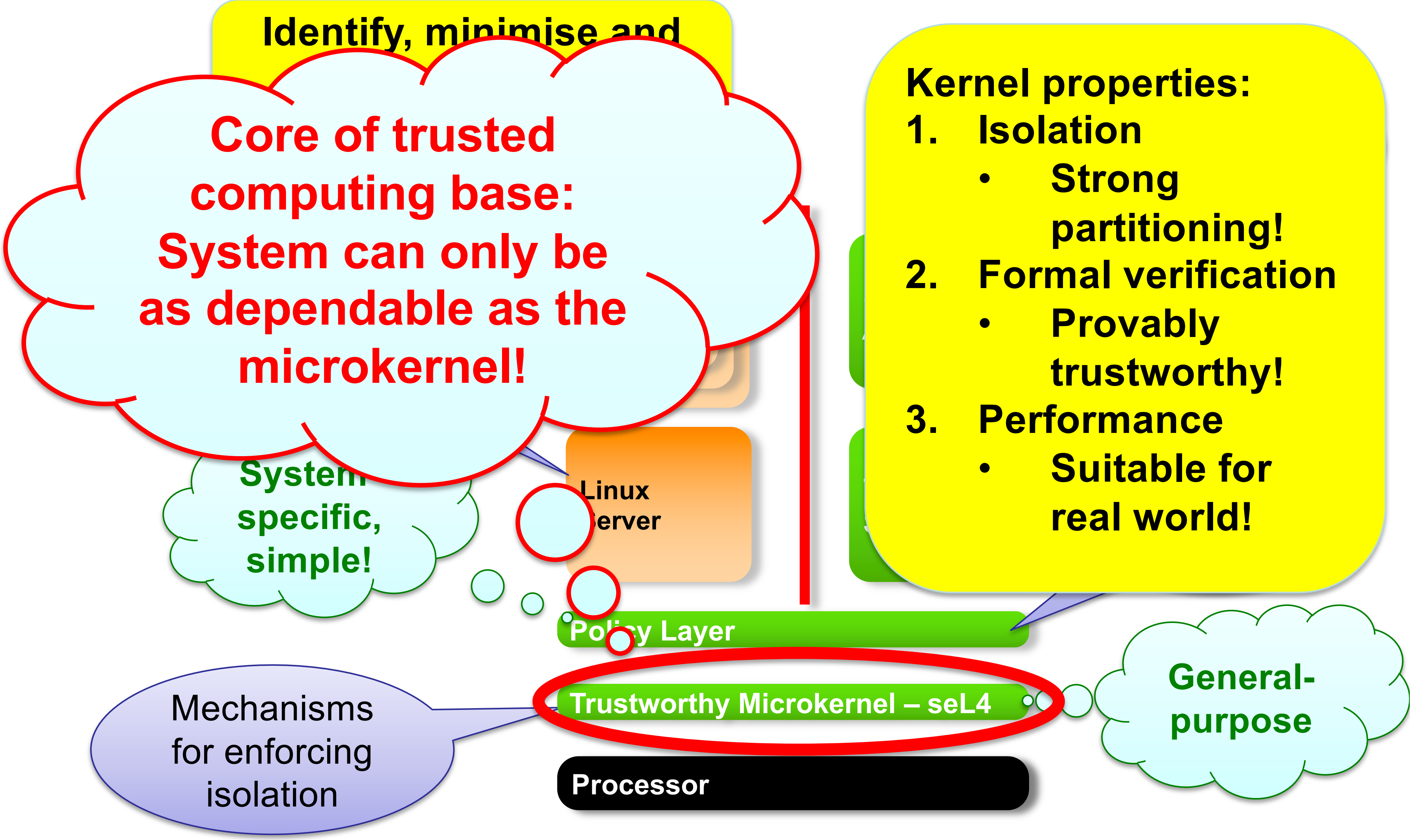
Hard *guarantees* on safety/security/ reliability

Isolation is Key!

Identify, minimise and isolate *critical* components!



Isolation is Key!


Identify, minimise and
Core of trusted computing base:
System can only be as dependable as the microkernel!
System specific, simple!
Linux Server
Policy Layer
Trustworthy Microkernel – seL4
Processor
Mechanisms for enforcing isolation
General-purpose

Kernel properties:

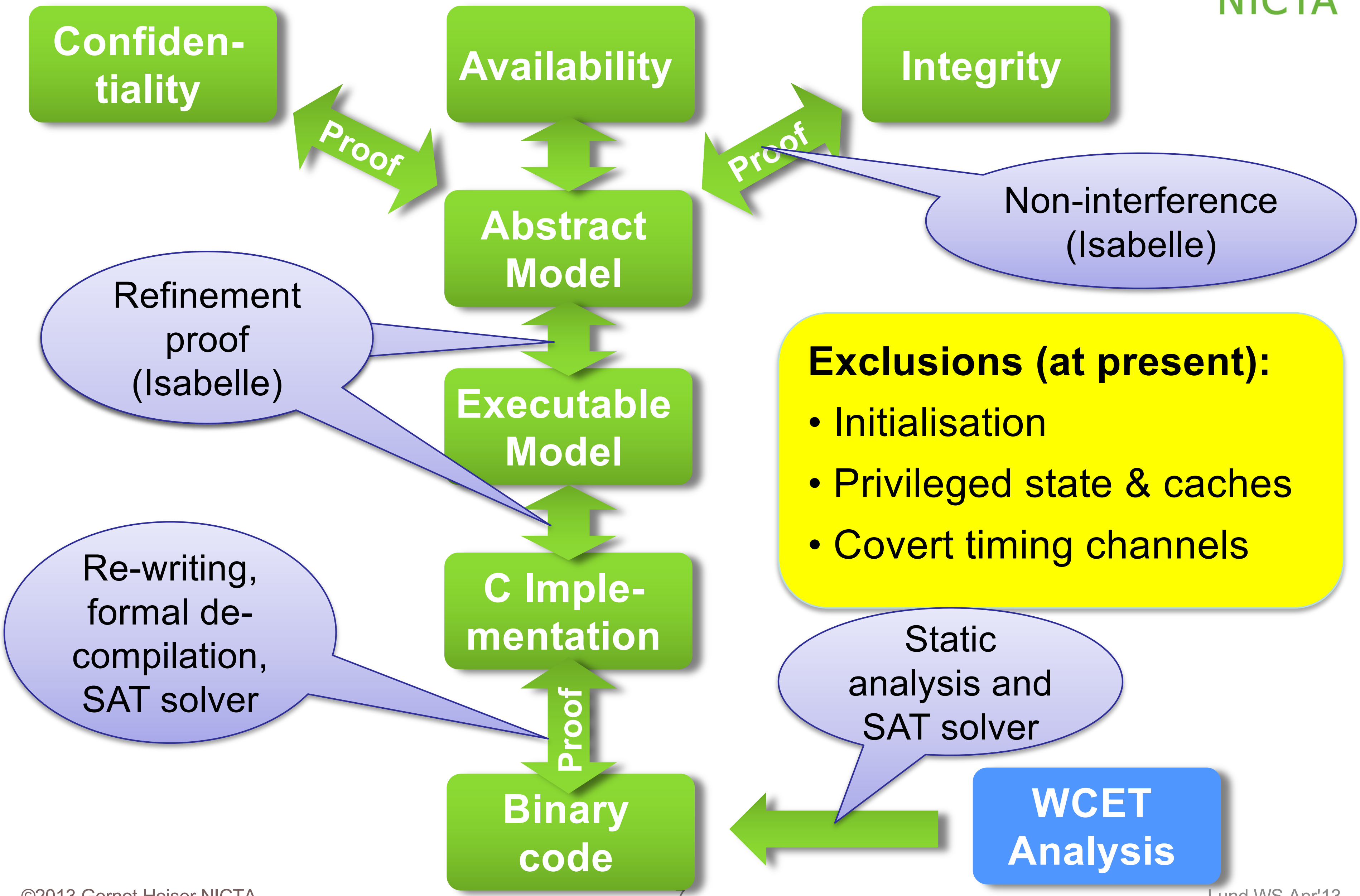
1. Isolation
   - Strong partitioning!
2. Formal verification
   - Provably trustworthy!
3. Performance
   - Suitable for real world!

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!

seL4: Proof Chain: From Requirements to Binary



Confidentiality

Availability

Integrity

Proof

Proof

Non-interference (Isabelle)

Abstract Model

Refinement proof (Isabelle)

Executable Model

C Implementation

Re-writing, formal decompilation, SAT solver

Proof

Binary code

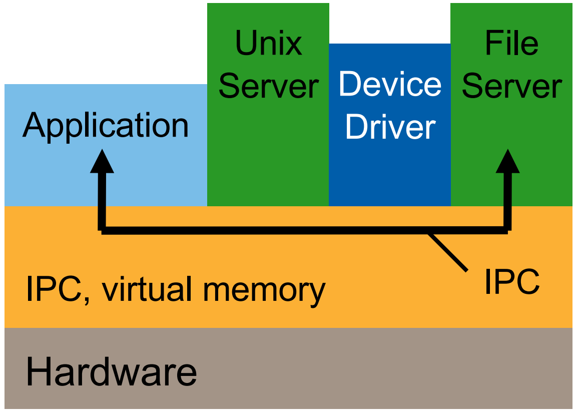
Static analysis and SAT solver

WCET Analysis

Exclusions (at present):

- Initialisation
- Privileged state & caches
- Covert timing channels

How About Performance?



seL4 is basically slow!

- C code quickly (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!

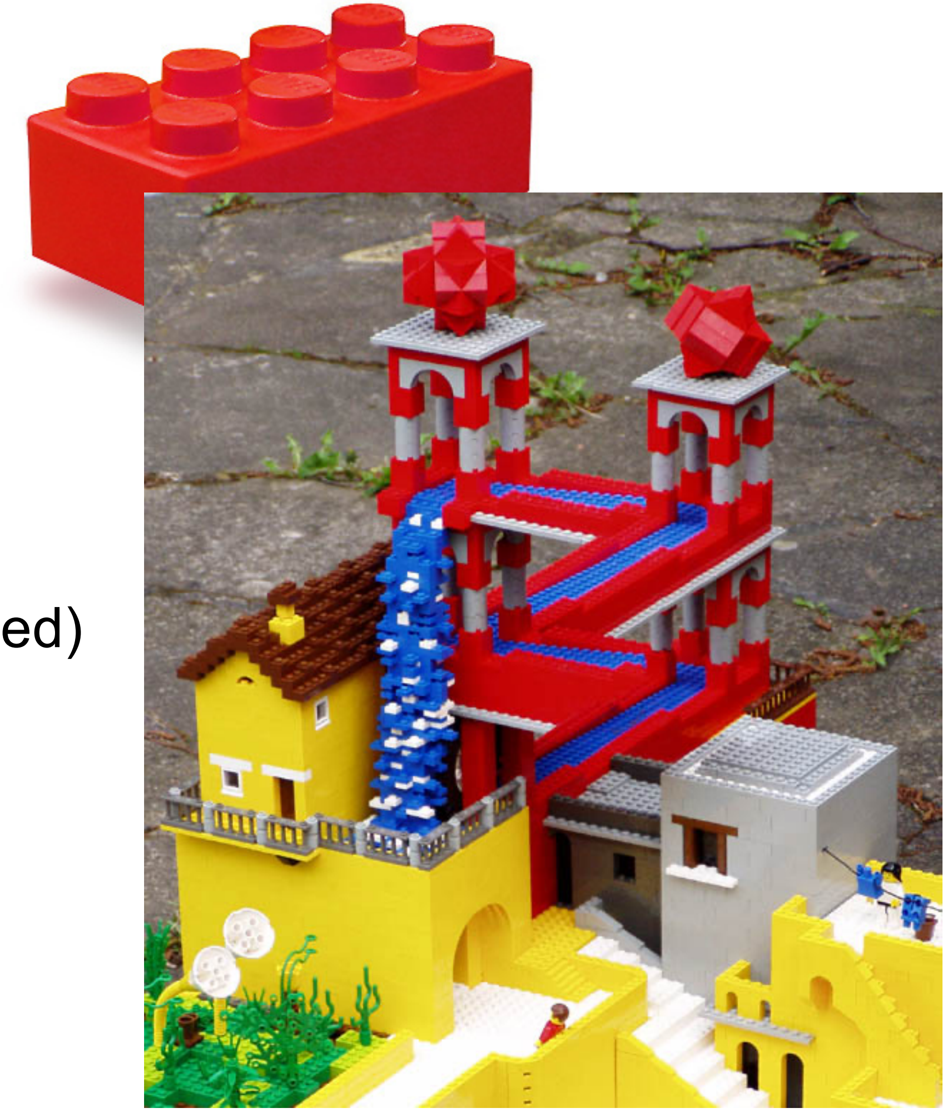
But can speed up critical operations by short-circuit “fast paths”

- … without resorting to assembler!

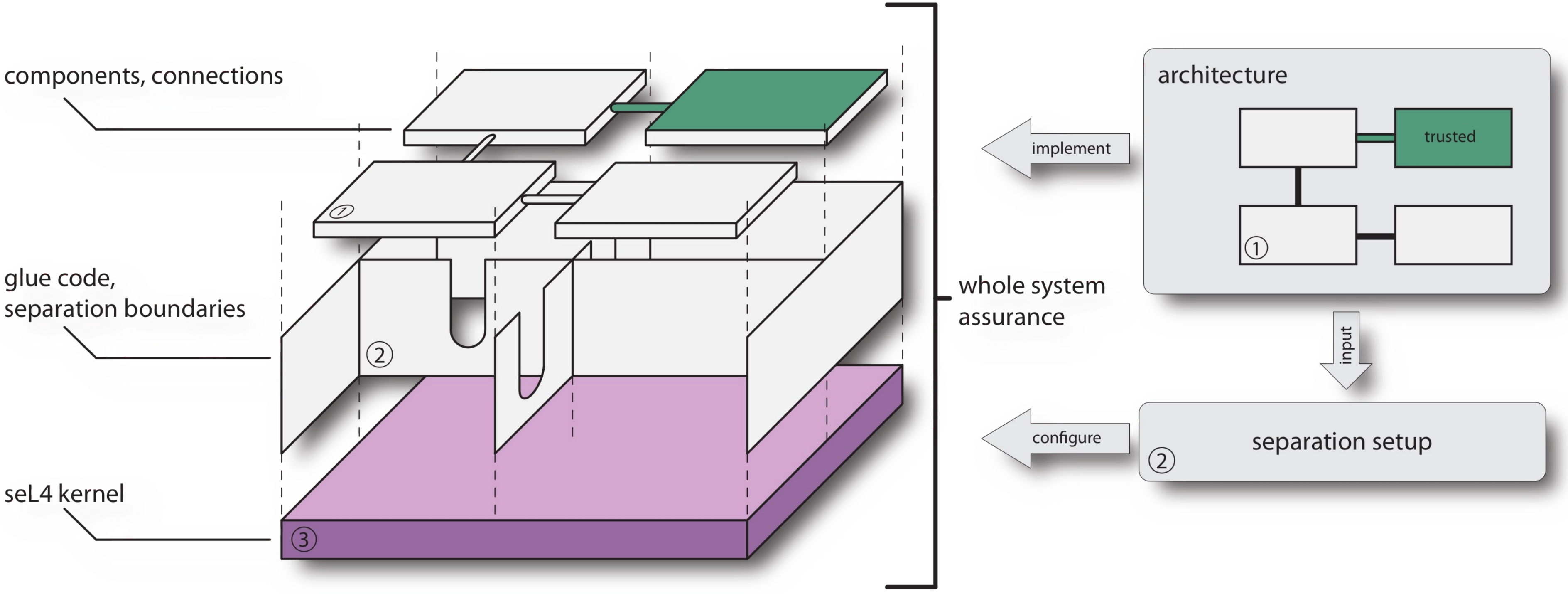
Full-System Guarantees

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

- Next step: Guarantees for real-world systems (1,000,000s LOC, 99% untrusted)

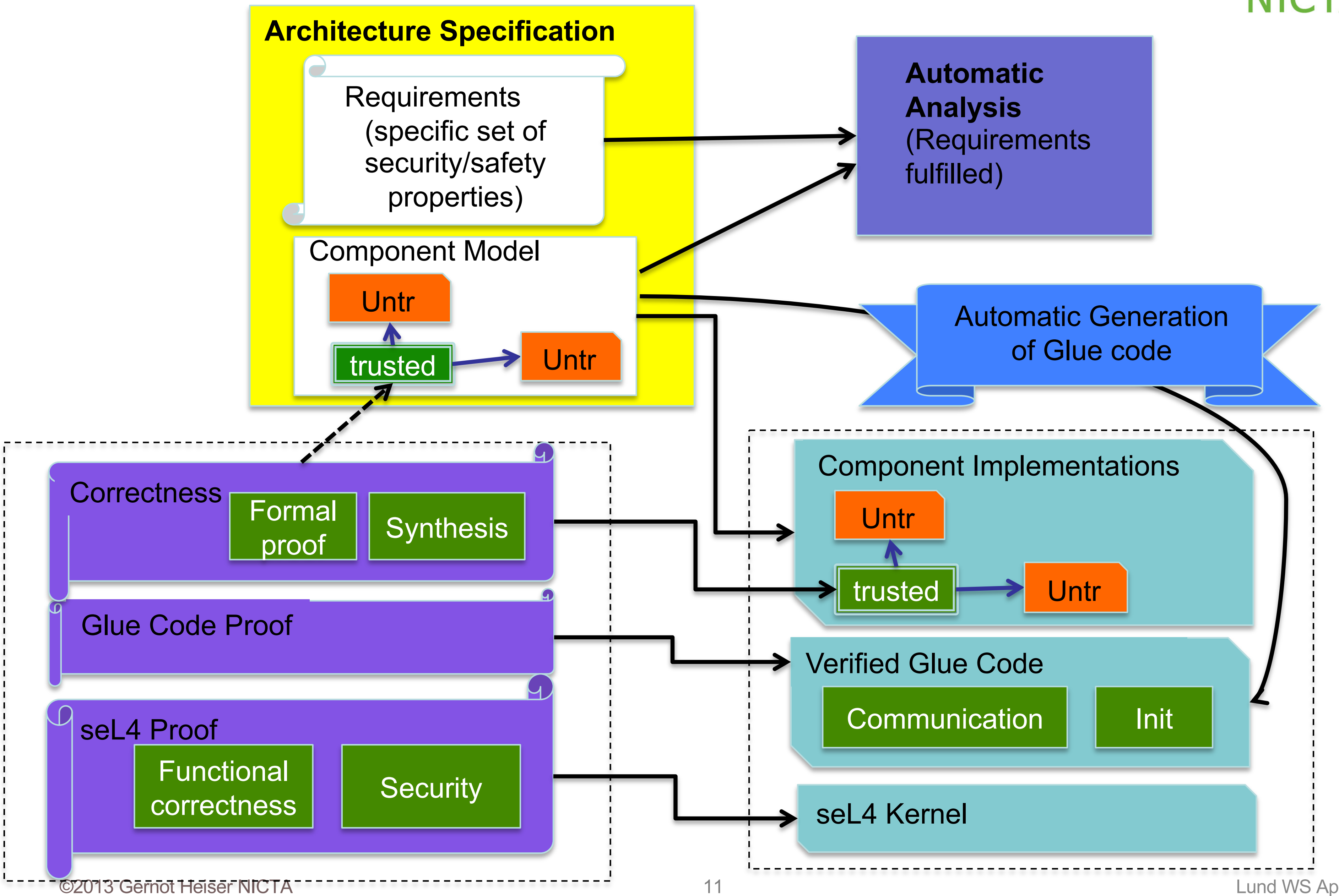


Overview of Approach



- 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 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

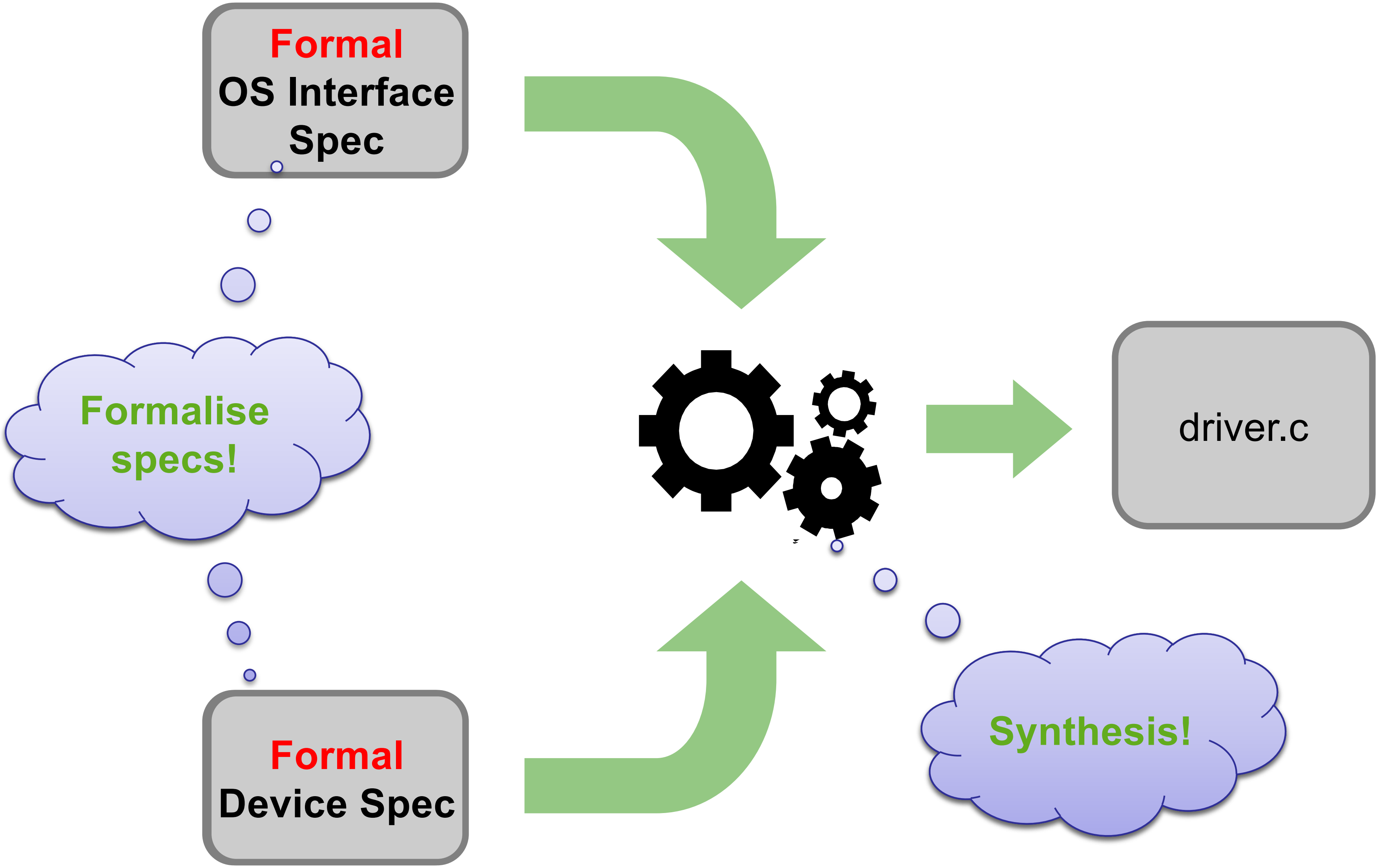
Verified Glue Code

Communication

Init

seL4 Kernel

Synthesis 1: Device Drivers



Formal OS Interface Spec

Formalise specs!

driver.c

Synthesis!

Formal Device Spec

Actually works!

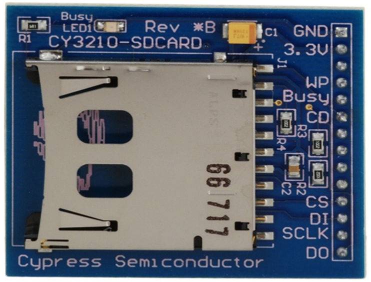
IDE disk controller

W5100 Eth shield

Intel PRO/1000 Ethernet

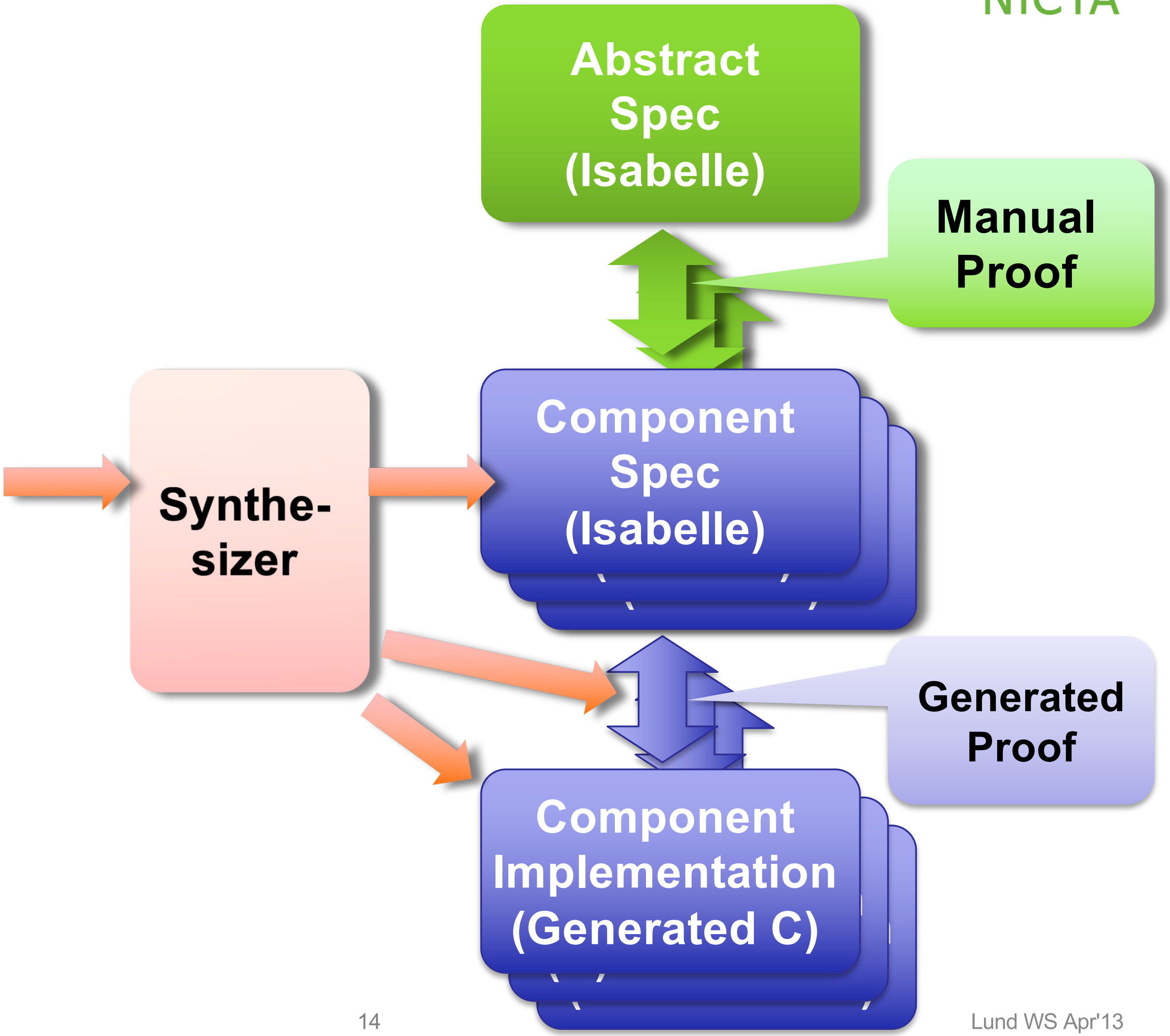
UART controller

Asix AX88772 USB-to-Eth adapter



SD host controller

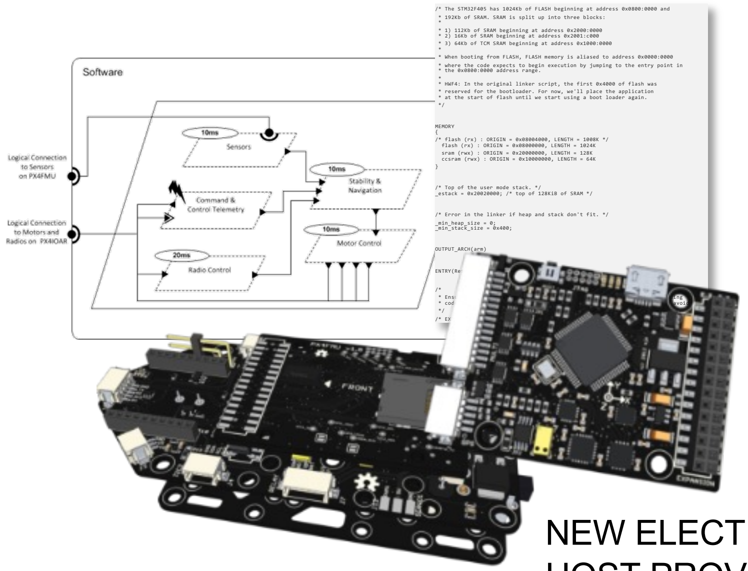
Synthesis 2: Domain-Specific Language (DSL)


Abstract Spec (Isabelle)
Manual Proof
Synthesizer
Component Spec (Isabelle)
Generated Proof
Component Implementation (Generated C)

Testbed: SMACCM Project (DARPA)

AR.DRONE
QUADCOPTER
(RESEARCH VEHICLE)

BOEING UNMANNED LITTLE BIRD (AH-6)



Software
Logical Connection to Sensors on PX4FMU
Logical Connection to Motors and Radios on PX4IOAR
10ms
Sensors
10ms
Stability & Navigation
Command & Control Telemetry
10ms
Motor Control
20ms
Radio Control
FRONT

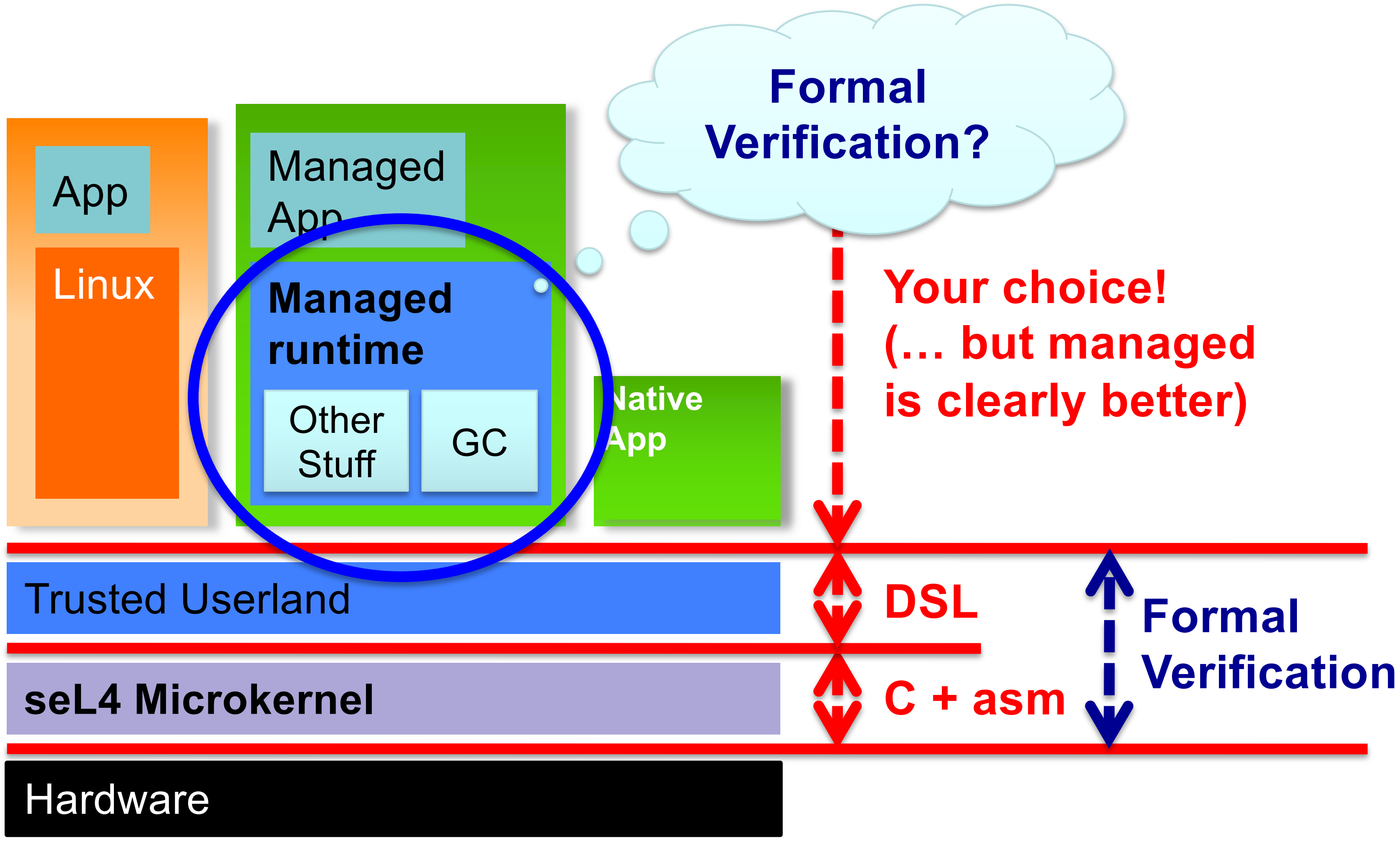
```
/* The STM32F405 has 1024Kb of FLASH beginning at address 0x0800:0000 and
 * 192Kb of SRAM. SRAM is split up into three blocks:
 *
 * 1) 112Kb of SRAM beginning at address 0x2000:0000
 * 2) 16Kb of SRAM beginning at address 0x2001:c000
 * 3) 64Kb of TCM SRAM beginning at address 0x1000:0000
 *
 * When booting from FLASH, FLASH memory is aliased to address 0x0000:0000
 * where the code expects to begin execution by jumping to the entry point in
 * the 0x0800:0000 address range.
 *
 * HWF4: In the original linker script, the first 0x4000 of flash was
 * reserved for the bootloader. For now, we'll place the application
 * at the start of flash until we start using a boot loader again.
 */


MEMORY
{
/* flash (rx) : ORIGIN = 0x08004000, LENGTH = 1008K */
  flash (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
  sram (rwx) : ORIGIN = 0x20000000, LENGTH = 128K
  ccsram (rwx) : ORIGIN = 0x10000000, LENGTH = 64K
}


/* Top of the user mode stack. */
_estack = 0x20020000; /* top of 128KiB of SRAM */


/* Error in the linker if heap and stack don't fit. */
_min_heap_size = 0;
_min_stack_size = 0x400;


OUTPUT_ARCH(arm)


ENTRY(Re


/*
 * Ens                                                      ing
 * cod                                                      avoid
 */
/* EX
```

NEW ELECTRONICS TO
HOST PROVABLY SECURE
SOFTWARE

Partners:

- Rockwell Collins
- NICTA
- Galois
- Boeing

Building Trustworthy Systems: Long-Term View


Formal Verification?
App
Linux
Managed App
Managed runtime
Other Stuff
GC
Native App
Your choice!
(… but managed is clearly better)
Trusted Userland
DSL
seL4 Microkernel
C + asm
Formal Verification
Hardware