

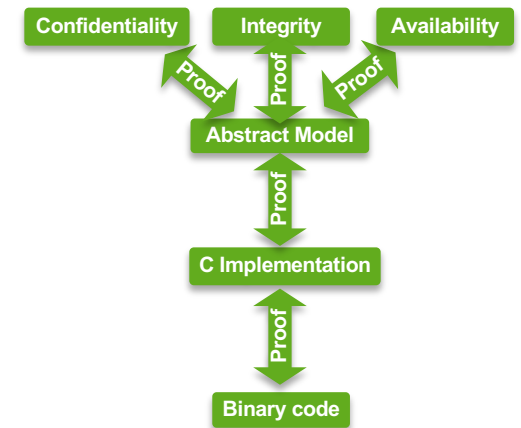


School of Computer Science & Engineering
COMP9242 Advanced Operating Systems

2024 T3 Week 7 Part 1

Formal Verification and seL4

@GernotHeiser



Copyright Notice

These slides are distributed under the Creative Commons Attribution 4.0 International (CC BY 4.0) License

- You are free:
 - to share—to copy, distribute and transmit the work
 - to remix—to adapt the work
- under the following conditions:
 - **Attribution:** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:

“Courtesy of Gernot Heiser, UNSW Sydney”

The complete license text can be found at
<http://creativecommons.org/licenses/by/4.0/legalcode>

Today's Lecture

- Assurance and verification
 - Common Criteria
 - Formal verification
- seL4
 - Functional correctness
 - Translation correctness
 - Security enforcement
 - Verification limitations
 - WCET analysis
 - Cost of verification
- Security impact of OS design

Assurance and Verification

Refresher: Assurance and Formal Verification

- **Assurance:**

- systematic evaluation and testing
- essentially an intensive and onerous form of quality assurance

- **Formal verification:**

- mathematical proof

Assurance and formal verification
aim to establish correctness of

- mechanism design
- mechanism implementation

- **Certification:** independent examination

- confirming that the assurance or verification was done right

Assurance: Substantiating Trust

- Specification
 - Unambiguous description of desired behaviour
- System design
 - Justification that it meets specification
- Implementation
 - Justification that it implements the design
- Maintenance
 - Justifies that system use meets assumptions

Informal (English)
or formal (maths)

Compelling argument
or formal proof

Code inspection,
rigorous testing,
proof

Common Criteria

- *Common Criteria for IT Security Evaluation* [ISO/IEC 15408, 99]
 - ISO standard, for general use
 - Evaluates QA used to ensure systems meet their requirements
 - Developed out of the famous US DOD “Orange Book”:
Trusted Computer System Evaluation Criteria [1985]
- Terminology:
 - *Target of evaluation* (TOE): Evaluated system
 - *Security target* (ST): Defines requirements
 - *Protection profile* (PP): Standardised ST template
 - *Evaluation assurance level* (EAL): Defines thoroughness of evaluation
 - PPs have maximum EAL they can be used for

CC: Evaluation Assurance Levels

Thoroughness, cost ↓

Level	Requirements	Specification	Design	Implementation
EAL1	not evaluated	Informal	not eval	not evaluated
EAL2	not evaluated	Informal	Informal	not evaluated
EAL3	not evaluated	Informal	Informal	not evaluated
EAL4	not evaluated	Informal	Informal	not evaluated
EAL5	not evaluated	Semi-Formal	Semi-Formal	Informal
EAL6	Formal	Semi-Formal	Semi-Formal	Informal
EAL7	Formal	Formal	Formal	Informal

Common Criteria: Protection Profiles (PPs)

- *Controlled Access PP* (CAPP)
 - standard OS security, up to EAL3
- *Single Level Operating System PP*
 - superset of CAPP, up to EAL4+
- *Labelled Security PP* (LSPP)
 - MAC for COTS OSeS
- *Multi-Level Operating System PP*
 - superset of CAPP, LSPP, up to EAL4+
- *Separation Kernel Protection Profile* (SKPP)
 - strict partitioning, for EAL6-7

COTS OS Certifications

- EAL3:
 - 2010 Mac OS X (10.6)
- EAL4:
 - 2003: Windows 2000
 - 2005: SuSE Enterprise Linux
 - 2006: Solaris 10 (EAL4+)
 - against CAPP (an EAL3 PP!)
 - 2007: Red Hat Linux (EAL4+)
- EAL6:
 - 2008: Green Hills INTEGRITY-178B (EAL6+)
 - against SKPP, relatively simple PPC-based hardware platform in TOE
- EAL7:
 - 2019: Prove & Run PROVENCORE
 - TEE OS for Arm TrustZone



Get regularly
hacked!

SKPP on Commodity Hardware

- SKPP: OS provides only separation
- One Box One Wire (OB1) Project
 - Use INTEGRITY-178B to isolate VMs on commodity desktop hardware
 - Leverage existing INTEGRITY certification
 - by “porting” it to commodity platform

NSA subsequently dis-endorsed SKPP,
discontinued certifying $\geq$ EAL5

Conclusion [NSA, March 2010]:

- SKPP validation for commodity hardware platforms infeasible due to their complexity
- SKPP has limited relevance for these platforms

Common Criteria Limitations

Effectively dead in
5-Eyes defence

- Very expensive
 - rule of thumb: EAL6+ costs \$1K/LOC [Green Hills] design-implementation-evaluation-certification
- Too much focus on development process
 - rather than the product that was delivered
 - “evaluating paperwork, not the product” [N Daughety, AFRL]
- Lower EALs of little practical use for OSEs
 - c.f. COTS OS EAL4 certifications
- Commercial Evaluation Facilities licenses rarely revoked
 - Leads to potential “race to the bottom” [Anderson & Fuloria, 2009]

Formal Verification

Prove properties about a mathematical model of a system

Automatic (“push-button”) techniques

- **Model checking / abstract interpretation / SMT**
- **Systematic exploration of system state space**
- ☐ Cannot generally prove code correct
 - Proves specific properties
 - Functional correctness in simple cases
- ☐ Generally have to
 - over-approximate (false positives), or
 - under-approximate (false negatives, unsound)
- ☐ Suffers state-space explosion
- ✓ Can scale to large code bases

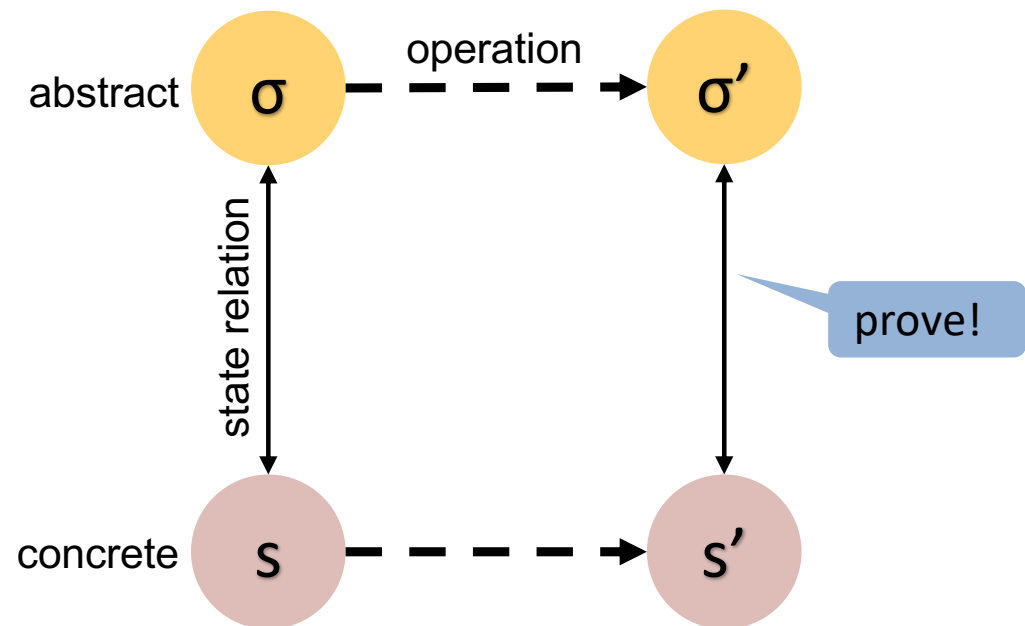
Interactive techniques:

- **Theorem proving**
- **Proofs about state spaces**
- ✓ Can deal with large (even infinite) state spaces
- ✓ Can prove functional correctness against a spec
- ☐ Very labour-intensive

Recent work automatically proved functional correctness of simple systems using SMT solvers [Hyperkernel, SOSp'17]

Theorem Proving

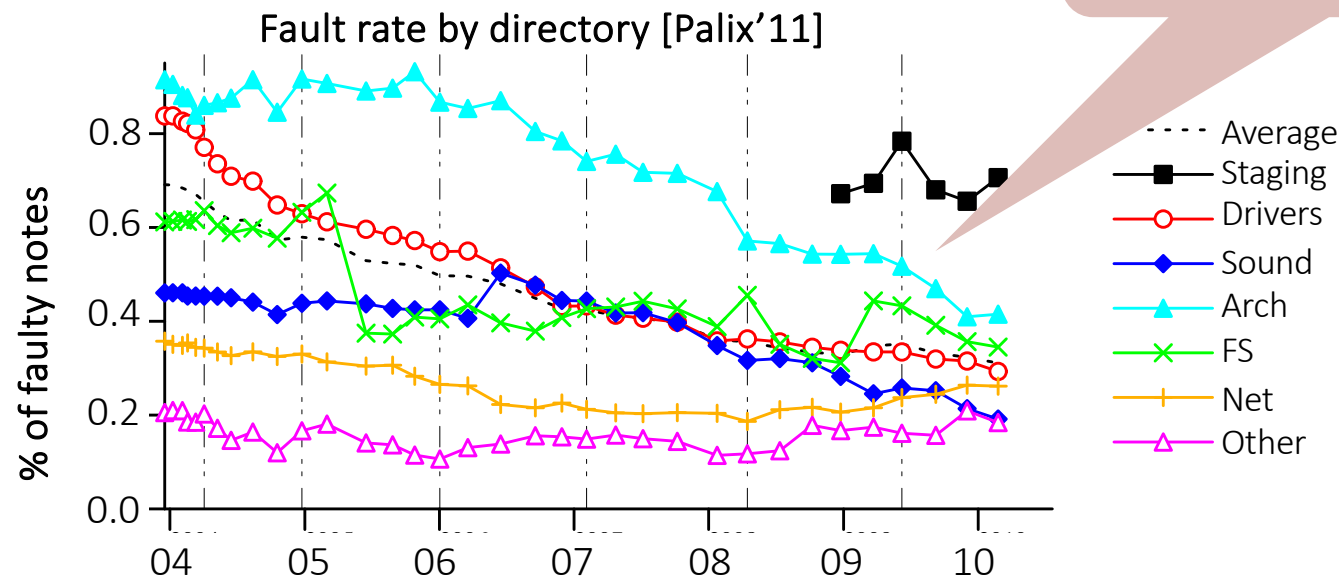
“Forward simulation”:
Prove state correspondence
of abstract and concrete levels



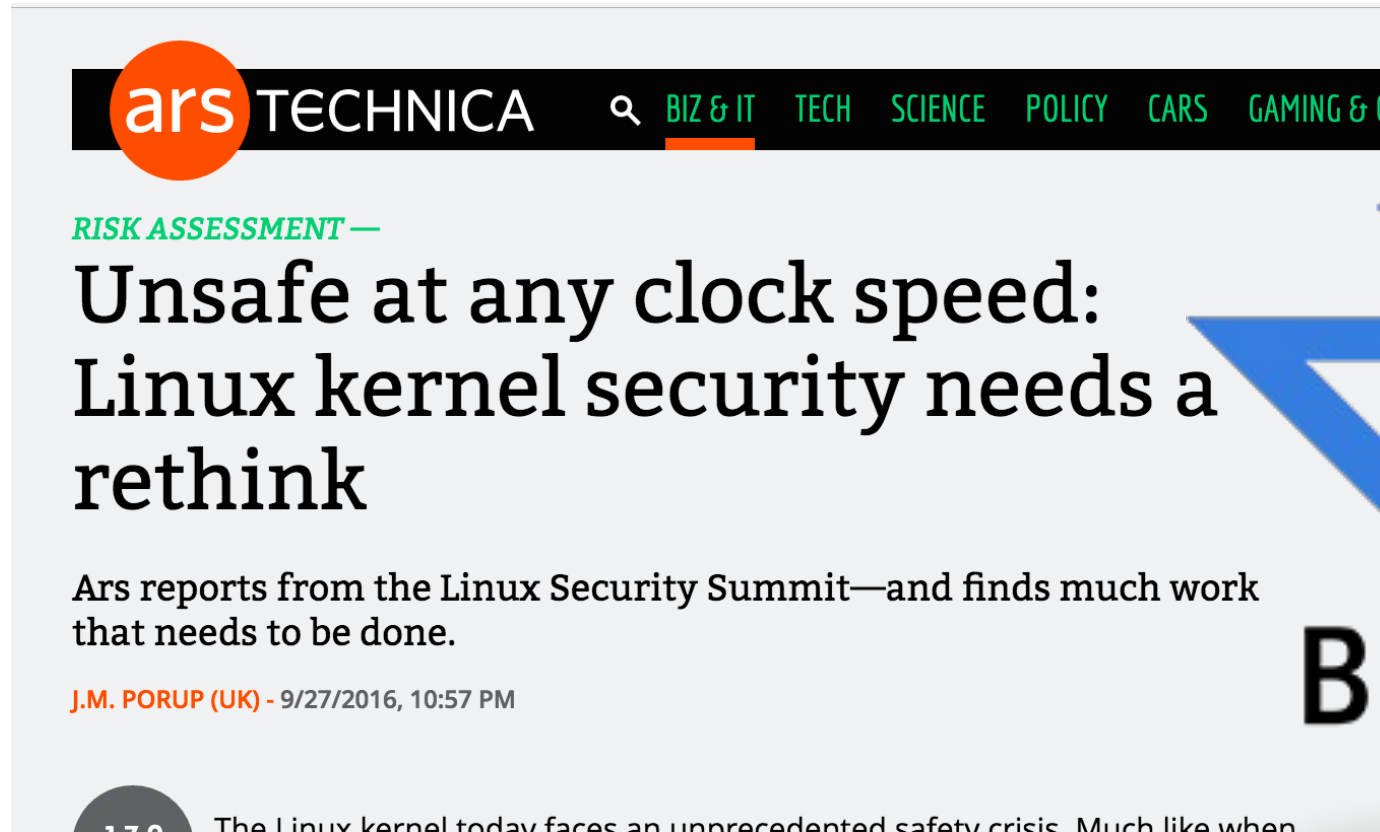
Model Checking and Linux: A Sad Story

- Static analysis of Linux source [Chou & al, 2001]
 - Found high density of bugs, especially in device drivers
- Re-analysis 10 years later [Palix & al, 2011]

Disappointing rate of improvement for bugs that are automatically detectable!



And the Result?





August 2009

A NICTA bejelentette a világ első, formális módszerekkel igazolt,



Slashdot is powered by your submissions

Technology: World's First

Posted by [Soulskill](#) on Thursday Aug
from the wait-for-it dept.

An anonymous reader writes

"Operating systems usually have and so forth are known by almost to prove that a particular OS kernel formally verified, and as such it researchers used an executable the Isabelle theorem prover to generate matches the executable and the

Does it run Linux? "We're pleased to say that it does." In other words, we have a para-virtualized version of Linux running on the microkernel. The microkernel is designed to be able to run on a wide range of hardware, and the Linux kernel is designed to be able to run on a wide range of hardware. The microkernel is designed to be able to run on a wide range of hardware, and the Linux kernel is designed to be able to run on a wide range of hardware. The microkernel is designed to be able to run on a wide range of hardware, and the Linux kernel is designed to be able to run on a wide range of hardware.



New Scientist

Saturday 29/8/2009

Page: 21

Section: General News

Region: National

Type: Magazines Science / Technology

Size: 196.31 sq.cms.

Published: ----S-

The ultimate way to keep your computer safe from harm

FLAWS in the code, or "kernel", that sits at the heart of modern computers leave them prone to occasional malfunction and vulnerable to attack by worms and viruses. So the development of a secure general-purpose microkernel could pave the

just mathematics, and you can reason about them mathematically," says Klein.

His team formulated a model with more than 200,000 logical steps which allowed them to prove that the program would always behave as its

IO BREAKTHROUGH TECHNOLOGIES

Crash-Proof Code

Making critical software safer

7 comments

VILLIAM BULKELEY

May/June 2011



ACM Software System Award 2022

PRESENTED TO

Gernot Heiser, Gerwin Klein, June Andronick, David Cock, Dhammika Elkaduwe, Kai Eng, Toby Murray, Rafal Kolanski, Thomas Sewell, Harvey Tuch,

For the development of the first industrial performance operating system kernel to have complete, mechanically checked proof of full

ACM PRESIDENT

AWARDS COMMITTEE CO-CHAIR

AWARDS COMMITTEE CO-CHAIR



The ACM SIGOPS 2019 Hall of Fame Award

is presented to

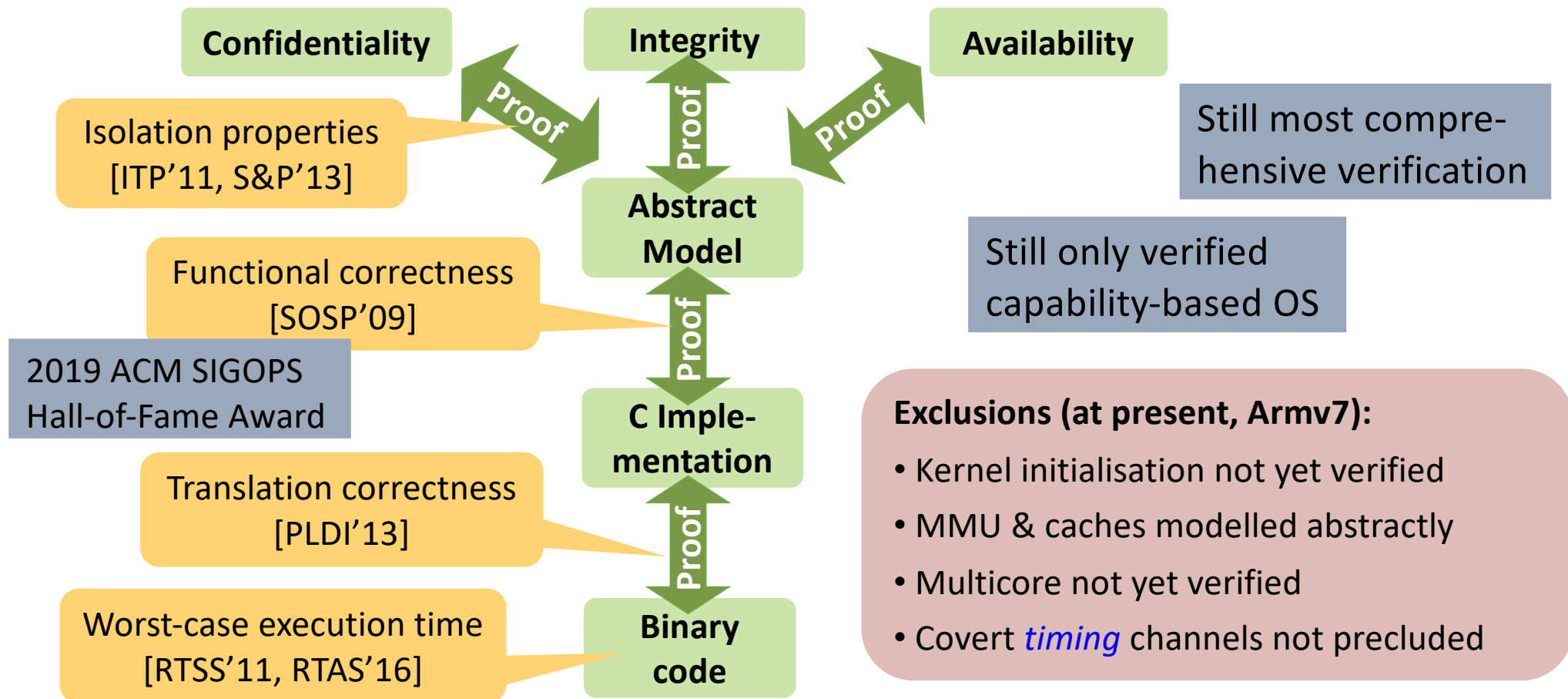
Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood

for their paper

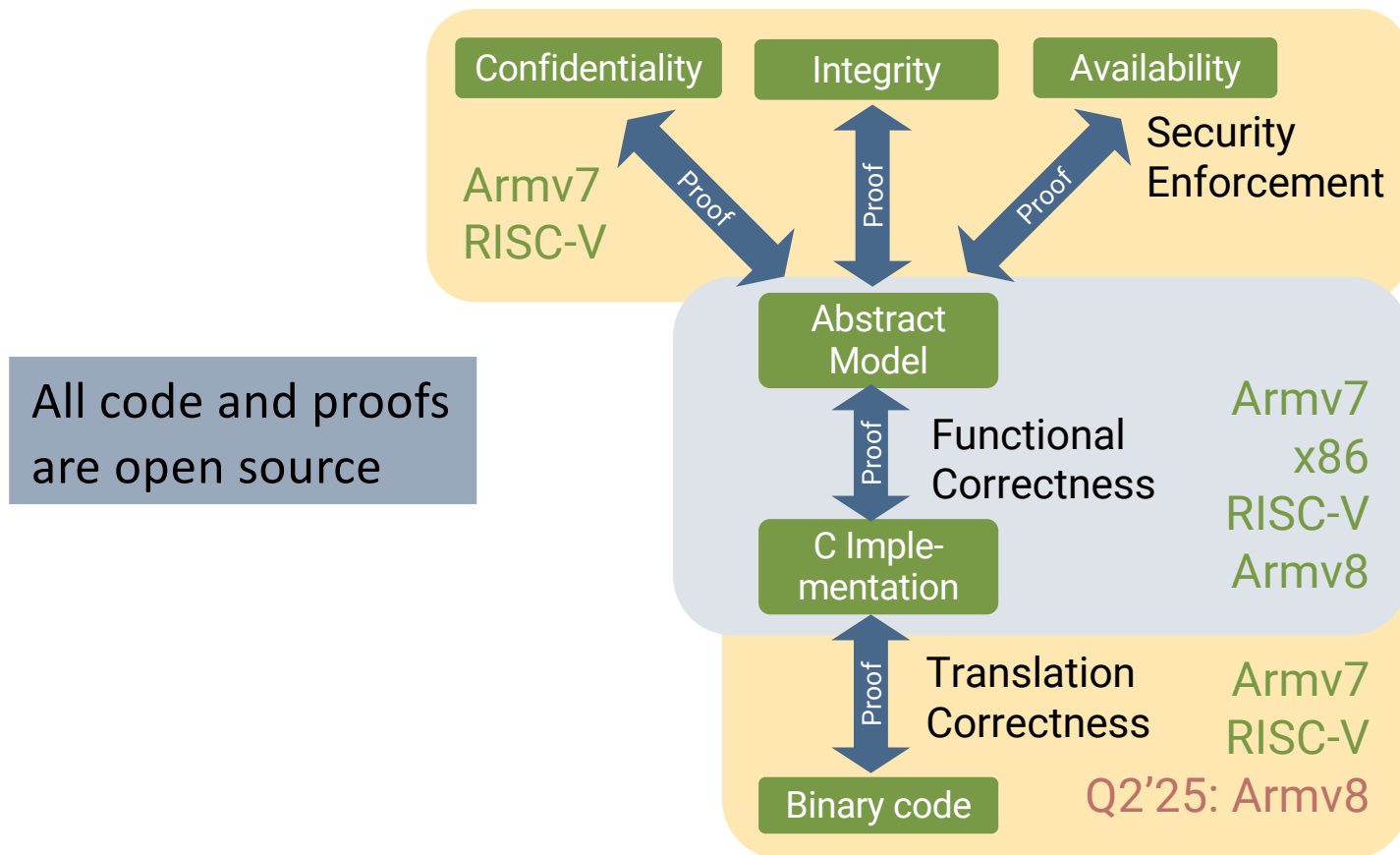
"seL4: Formal Verification of an OS Kernel"

2009 ACM Symposium on Operating Systems Principles

seL4 Proving Security and Safety (Armv6/7)



seL4 Other Architectures



Security Is No Excuse For Bad Performance!

Cost	seL4	Fiasco.OC	Zircon
IPC RT latency (cycles)	986	2717	8157
Mand. HW cost (cycles)	790	790	790
Abs. overhead (cycles)	196	1972	7367
Rel. overhead (%)	25	240	930

Hardware
cost dominates

SW overheads
dominate

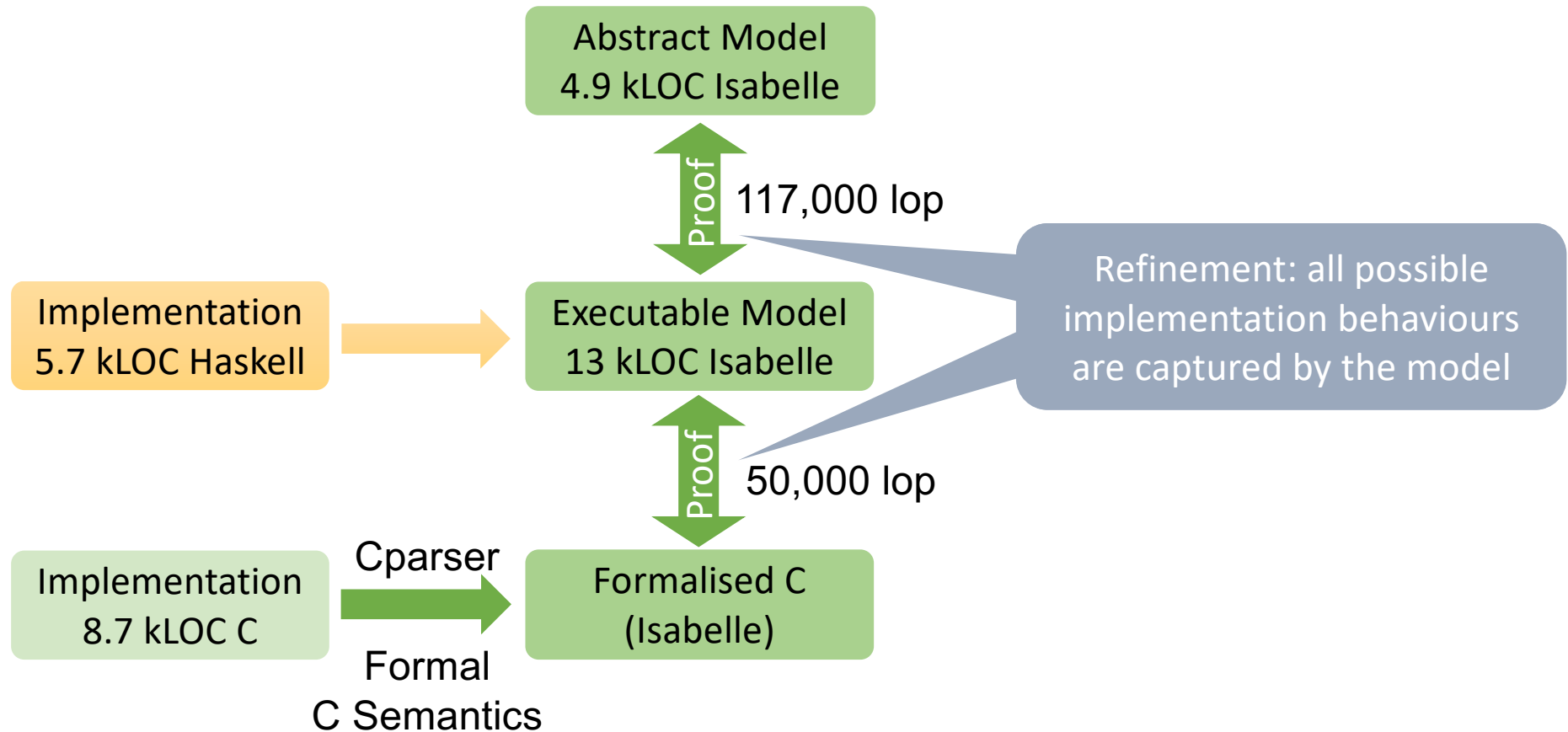
Round-trip, cross-address-space IPC on x64 (Intel Skylake)

Operation	1-way	RT
SYSCALL	82	164
SWAPGS	2×26	104
Switch PT	186	372
SYSRET	75	150
Total	395	790

Source: Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, Haibo Chen: "SkyBridge: Fast and Secure Inter-Process Communication for Microkernels", EuroSys, April 2019

Functional Correctness

seL4 Proving Functional Correctness



Functional Correctness 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
 - will never access array out of bounds
 - cannot be subverted by malformed input
- All syscalls terminate, reclaiming memory is safe, ...
- Well typed references, aligned objects, kernel always mapped...
- Access control is decidable

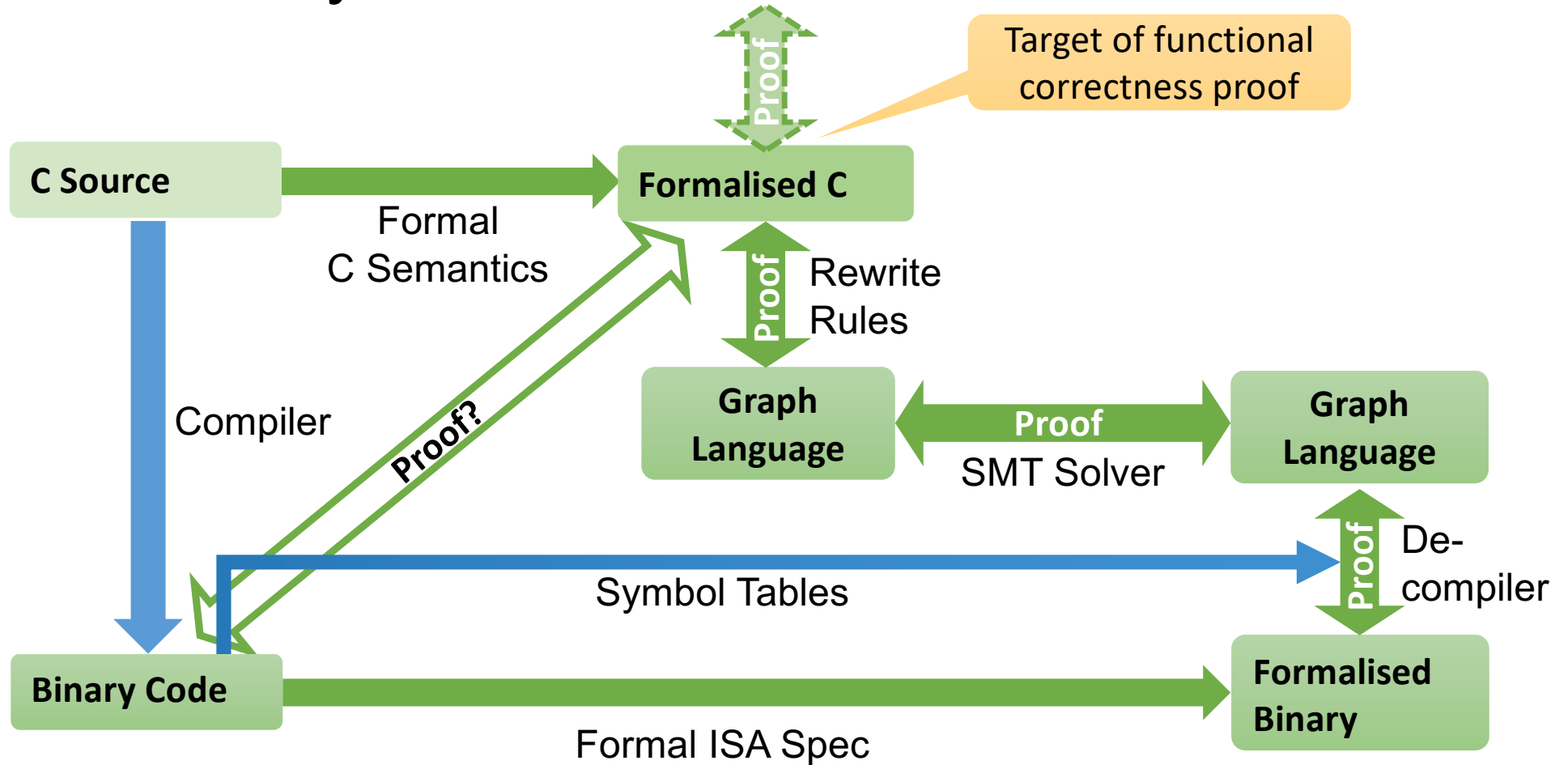
Can prove further properties on abstract level!

Bugs found:

- 16 in (shallow) testing
- 460 in verification
 - 150 in spec,
 - 150 in design,
 - 160 in C

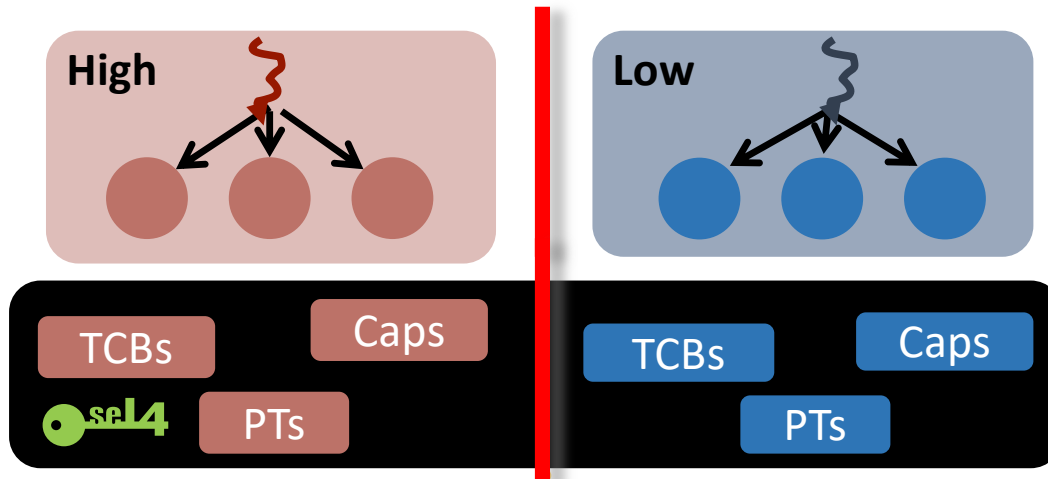
Translation Correctness

seL4 Binary Verification: Translation Validation



Security Enforcement

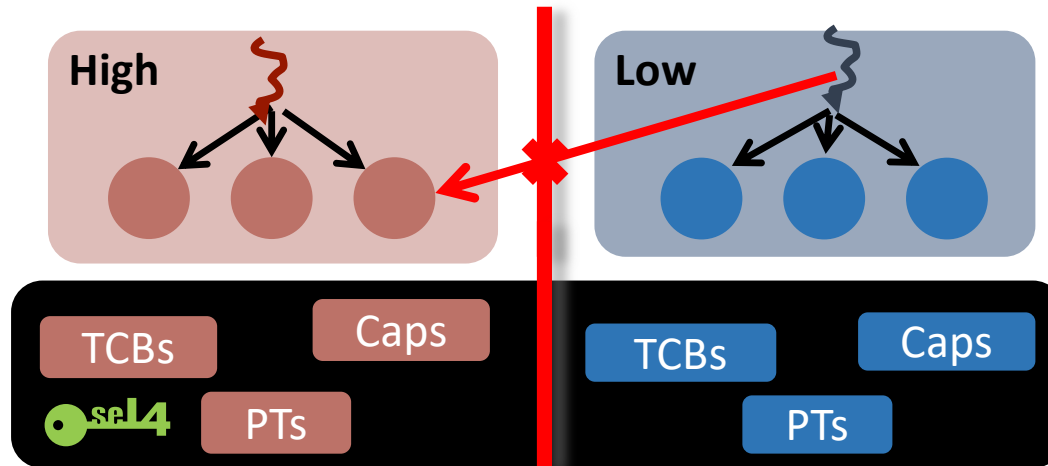
sel4 Isolation Goes Deep



Kernel data
partitioned
like user data



Integrity: Control Write Access



To prove:

Low has no *write* capabilities to High objects

⇒ no action of Low will modify High state

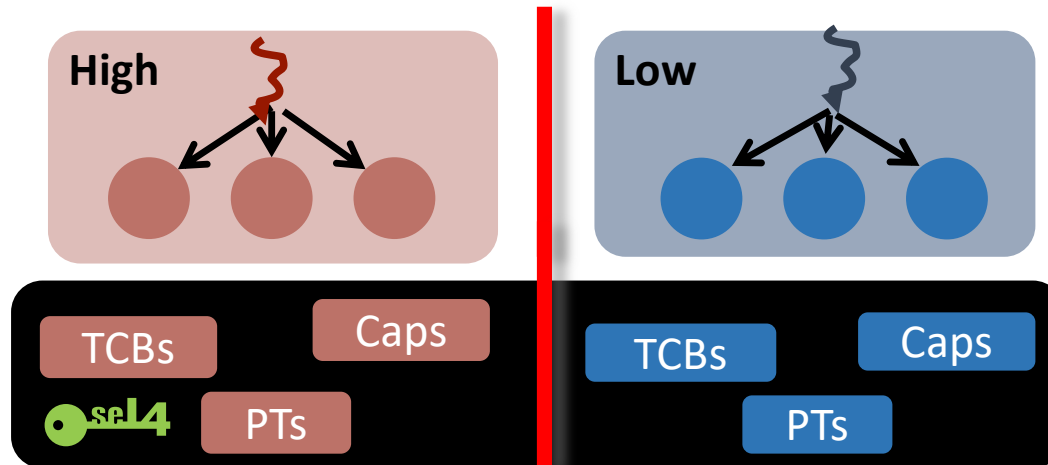
Specifically, *kernel does not modify on Low's behalf!*

Event-based kernel always operates on be-half of well-defined user:

- Prove kernel only modifies data if presented write cap



Availability: Ensuring Resource Access

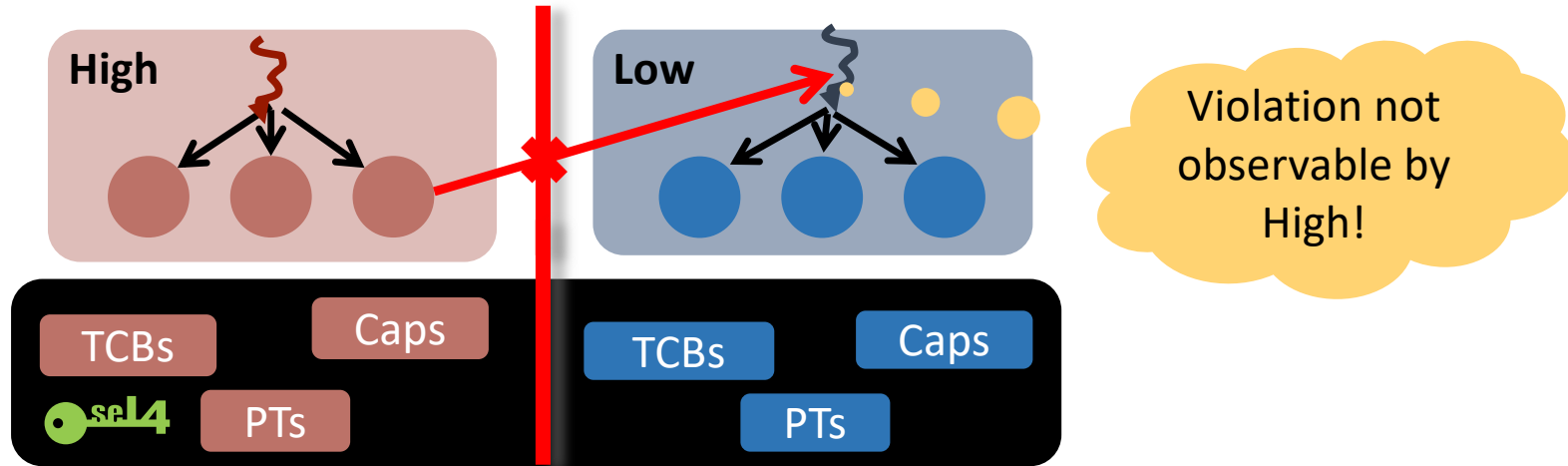


Nothing to do, implied by other properties!

Strict separation of kernel resources
⇒ Low cannot deny High access to resources



Confidentiality: Control Information Flow



Non-interference proof:

- Evolution of Low does not depend on High state
- Also shows absence of covert *storage channels*

To prove:

Low has no *read* capabilities to High objects
⇒ no action will reveal High state to Low



Confidentiality Proof Challenge

Spec

```
bool a();
```

Idiotic but valid refinement

Implementation

```
bool a() {  
    return !secret;  
}
```

Solution:

- Remove non-determinism where it affects confidentiality
- Eg: scheduler strictly round-robin

Non-determinism
breaks confidentiality
under refinement!

Infowflow is very strong
property, requiring
restrictions rarely met
in real world

Limitations



Verification Assumptions

1. Hardware behaves as expected

- Formalised hardware-software contract (ISA)
- Hardware implementation free of bugs, Trojans, ...

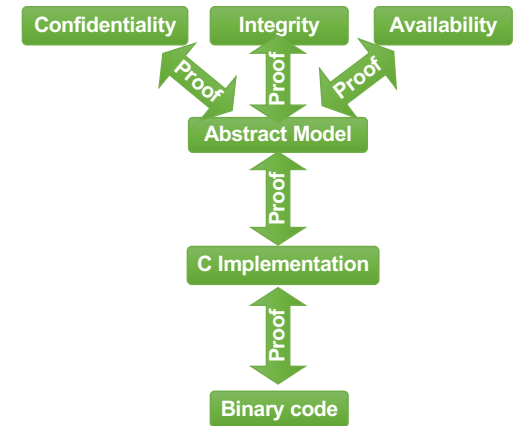
2. Spec matches expectations

- Can only prove “security” if specify what “security” means
- Spec may not be what we think it is

3. Proof checker is correct

- Isabel/HOL checking core that validates proofs against logic

With binary verification do
not need to trust C compiler!

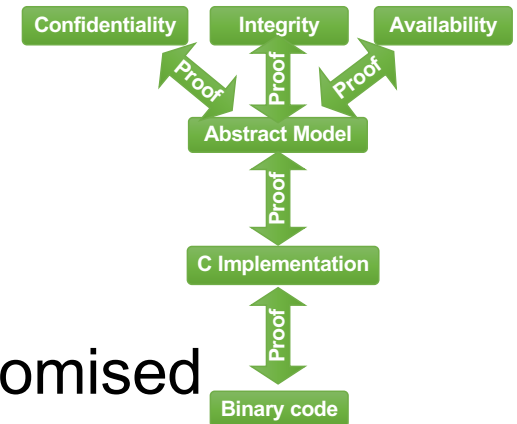




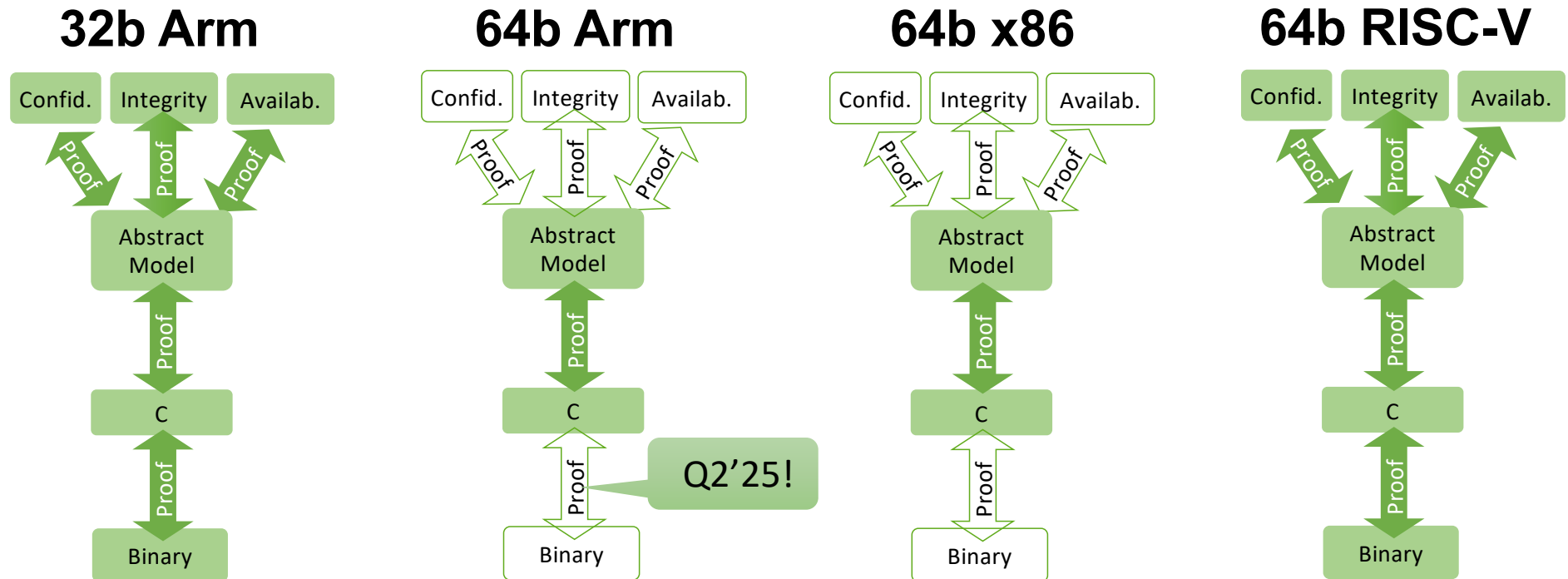
Present Verification Limitations

- Not verified boot code
 - **Assume** it leaves kernel in safe state
- Caches/MMU presently modeled at high level / axiomised
 - MMU model finished by PhD but not integrated
- Not proved any temporal properties
 - Presently not proved scheduler observes priorities, properties needed for RT
 - WCET analysis applies only to outdated ARM11/A8 cores
 - No proofs about timing channels (yet)

Present research!



seL4 Present Status



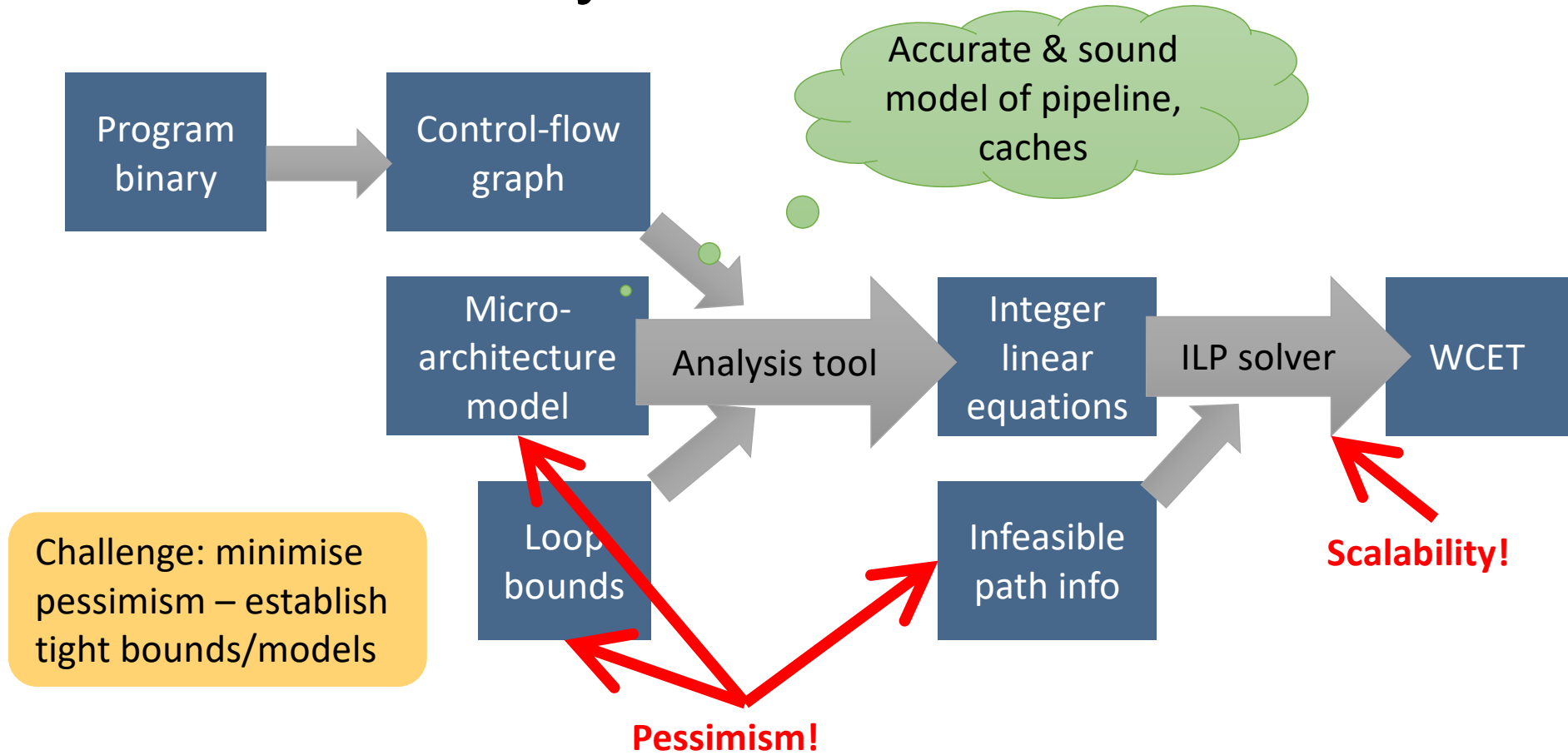


Common Criteria?

Level	Requirements	Specification	Design	Implementation
EAL1	not evaluated	Informal	not eval	not evaluated
EAL2	not evaluated	Informal	Informal	not evaluated
EAL3	not evaluated	Informal	Informal	not evaluated
EAL4	not evaluated	Informal	Informal	not evaluated
EAL5	not evaluated	Semi-Formal	Semi-Formal	Informal
EAL6	Formal	Semi-Formal	Semi-Formal	Informal
EAL7	Formal	Formal	Formal	Informal
 seL4	Formal	Formal	Formal	Formal

WCET Analysis

sel4 WCET Analysis

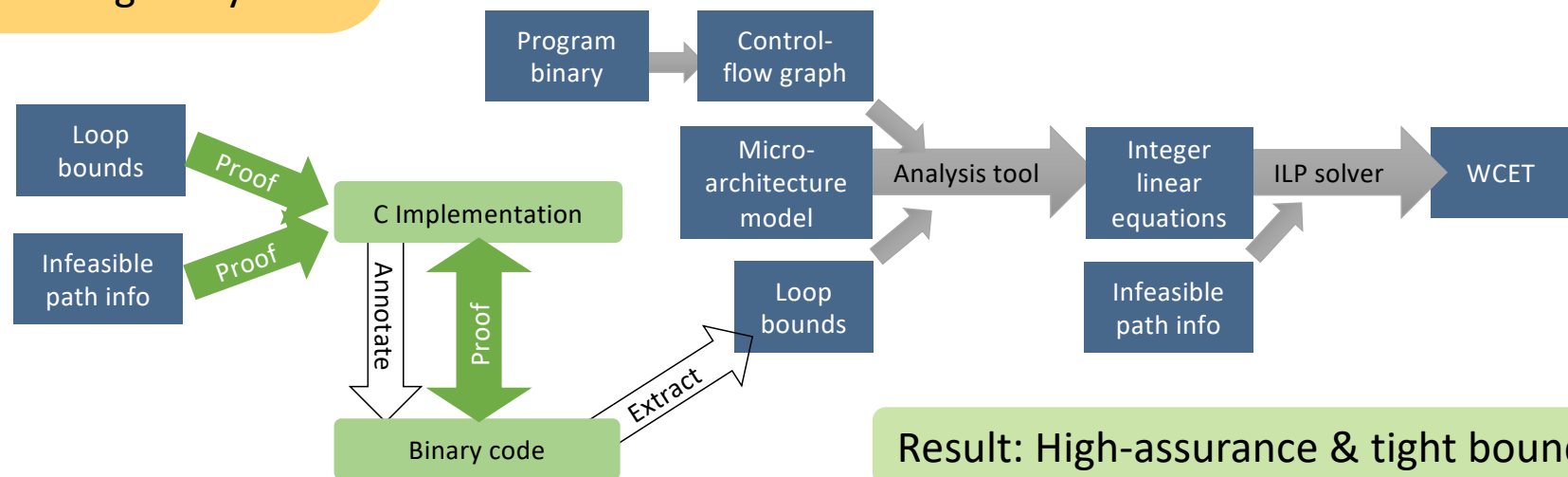


seL4 Loop Bounds & Infeasible Paths

Tight loop bounds and infeasible path refutations infeasible to obtain from binary – lack of semantic information, especially pointer aliasing analysis.

Idea:

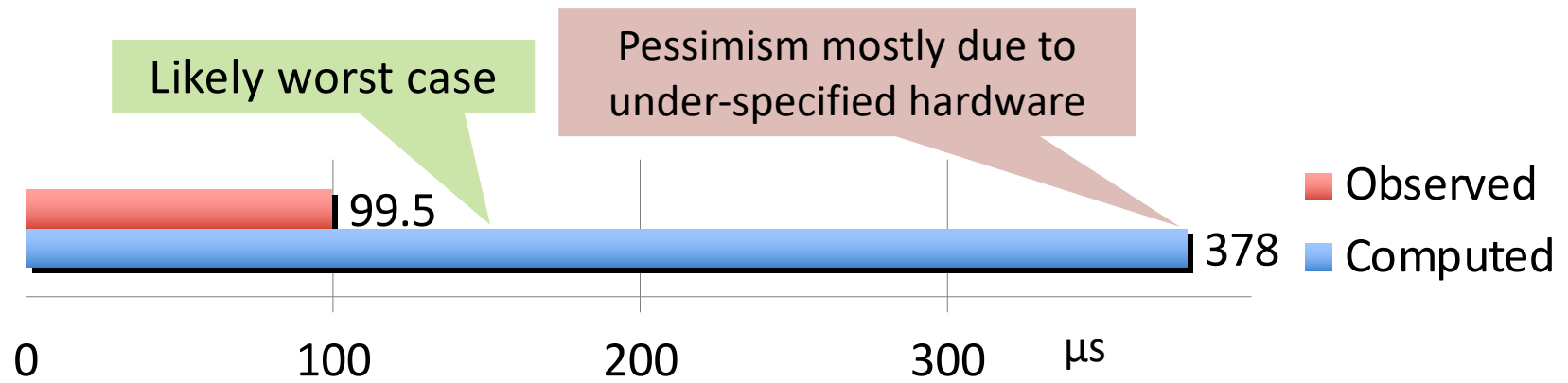
- prove on C level
- transfer to binary using translation-validation toolchain



Result: High-assurance & tight bounds!



WCET Analysis on ARM11



WCET presently limited by verification practicalities

- without regard to verification achieved 50 μs
- 10 μs seem achievable
- BCET ~ 1μs
- [Blackham'11, '12] [Sewell'16]

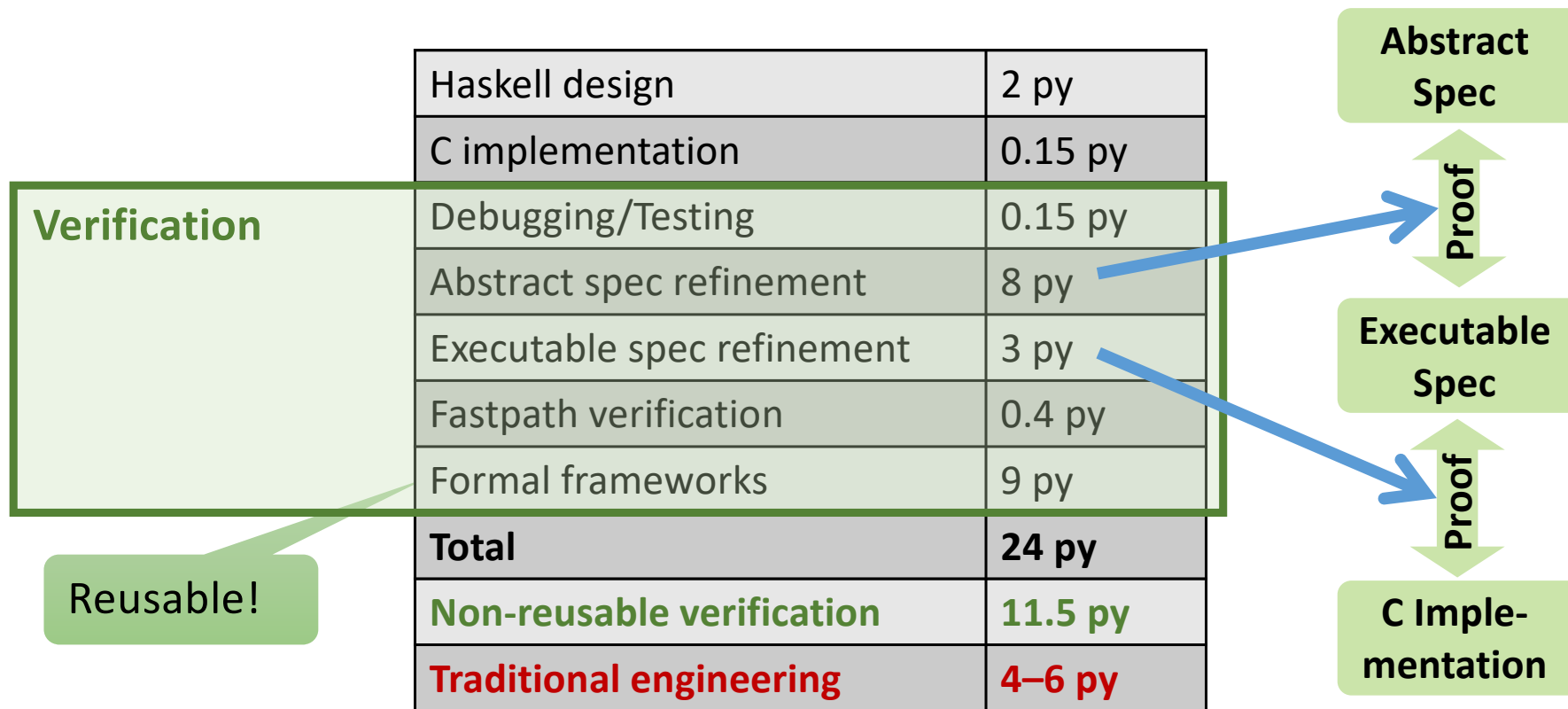
Problem: Latency information no longer published by Arm!



Cost of Verification



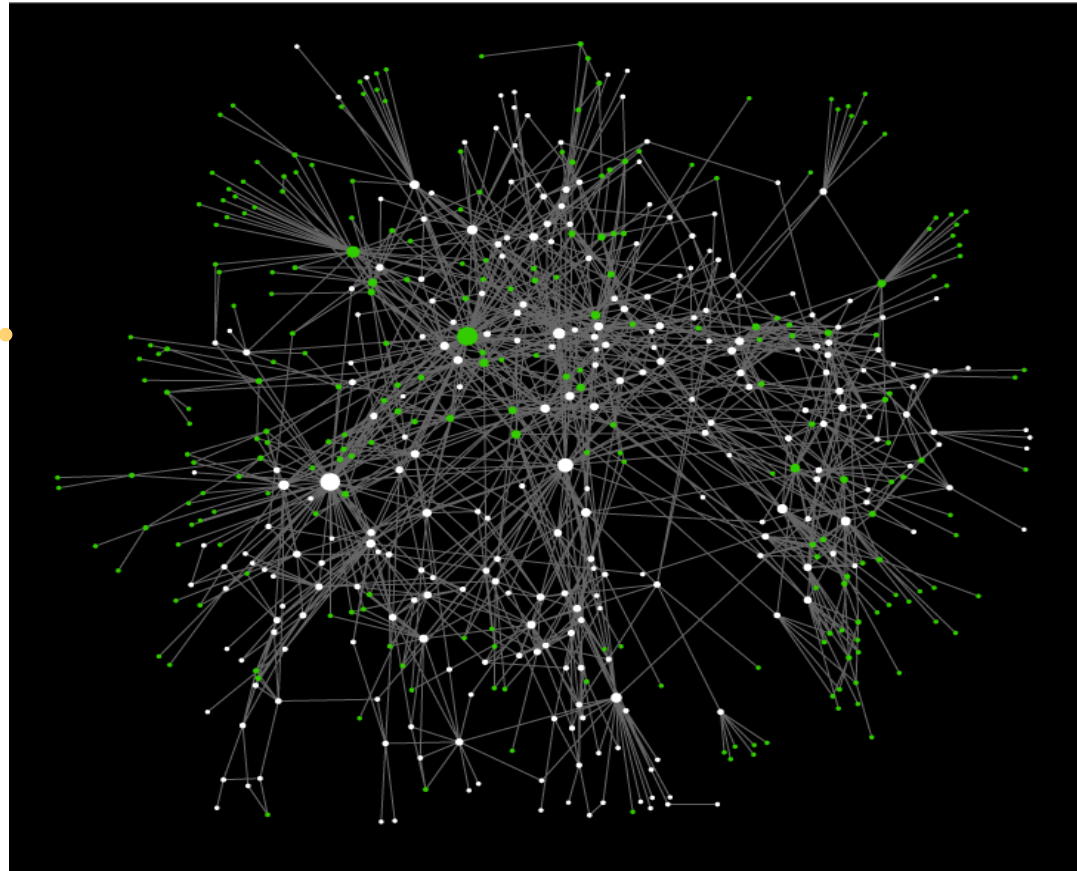
Verification Cost Breakdown



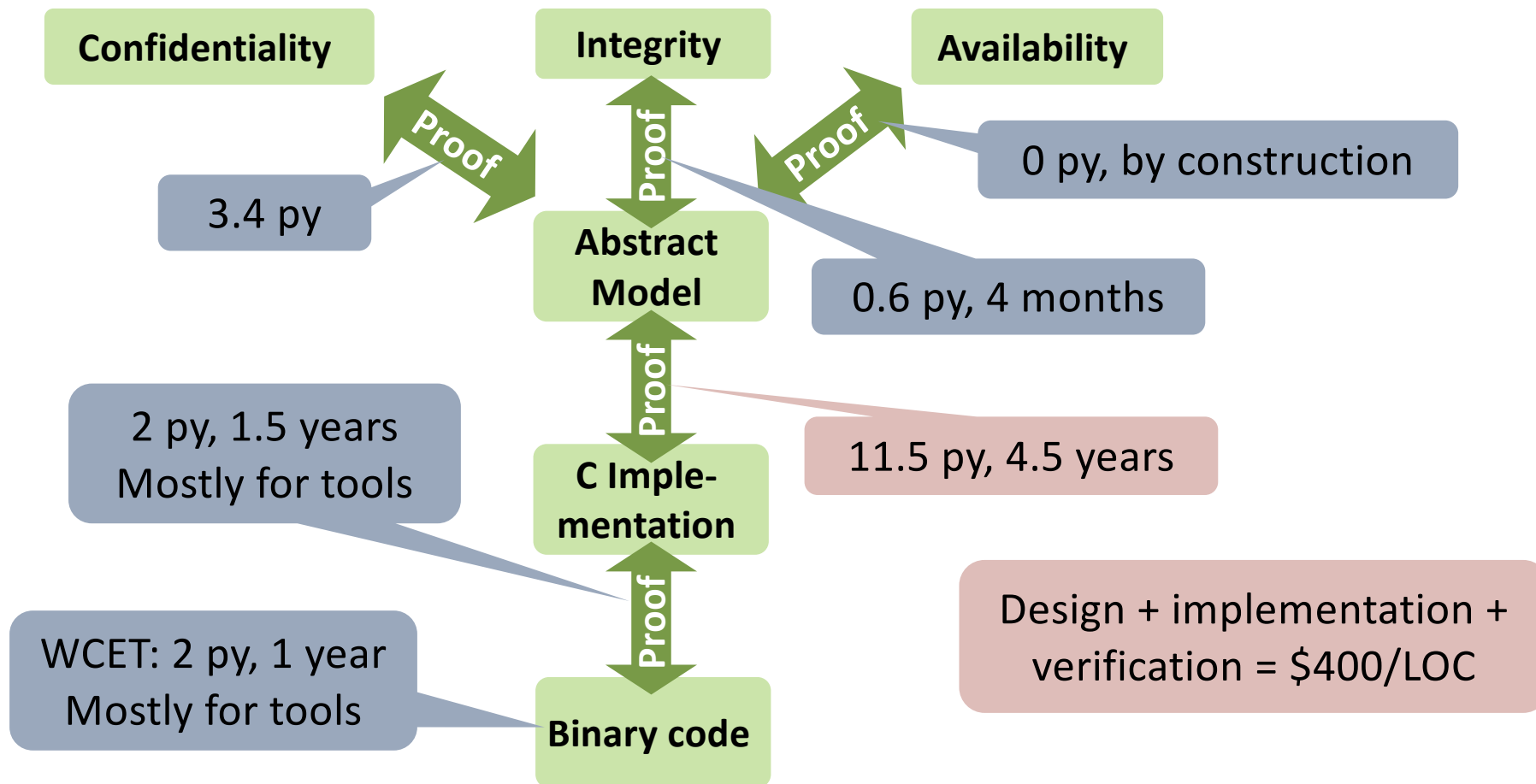


Why So Hard for 9,000 LOC?

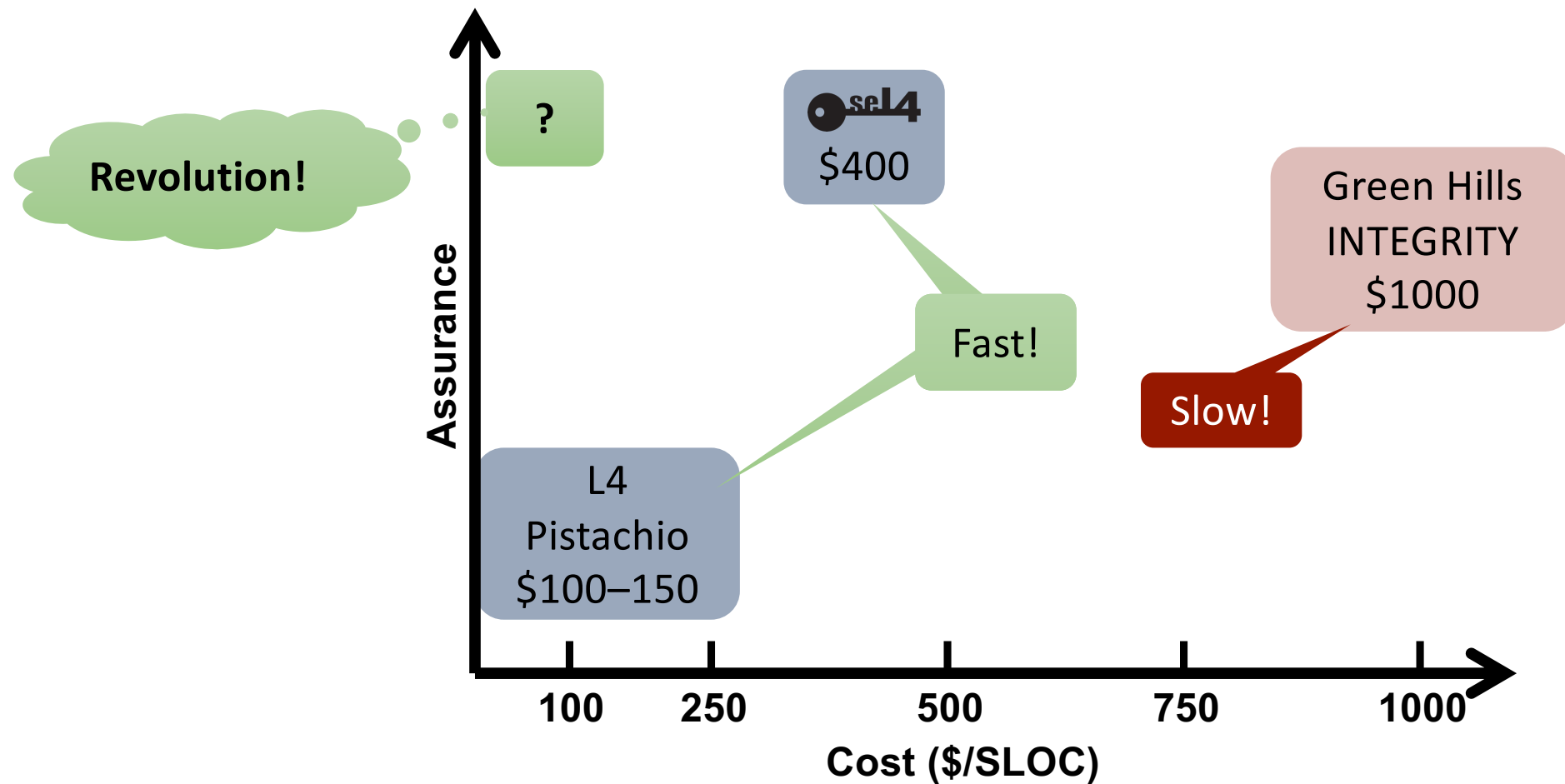
seL4 call
graph



seL4 Verification Cost



seL4 Microkernel Life-Cycle Cost in Context



Security Impact of OS Design

Quantifying OS-Design Security Impact

Approach:

- Examine all **critical** Linux CVEs (vulnerabilities & exploits database)

- easy to exploit
- high impact
- no defence available
- confirmed

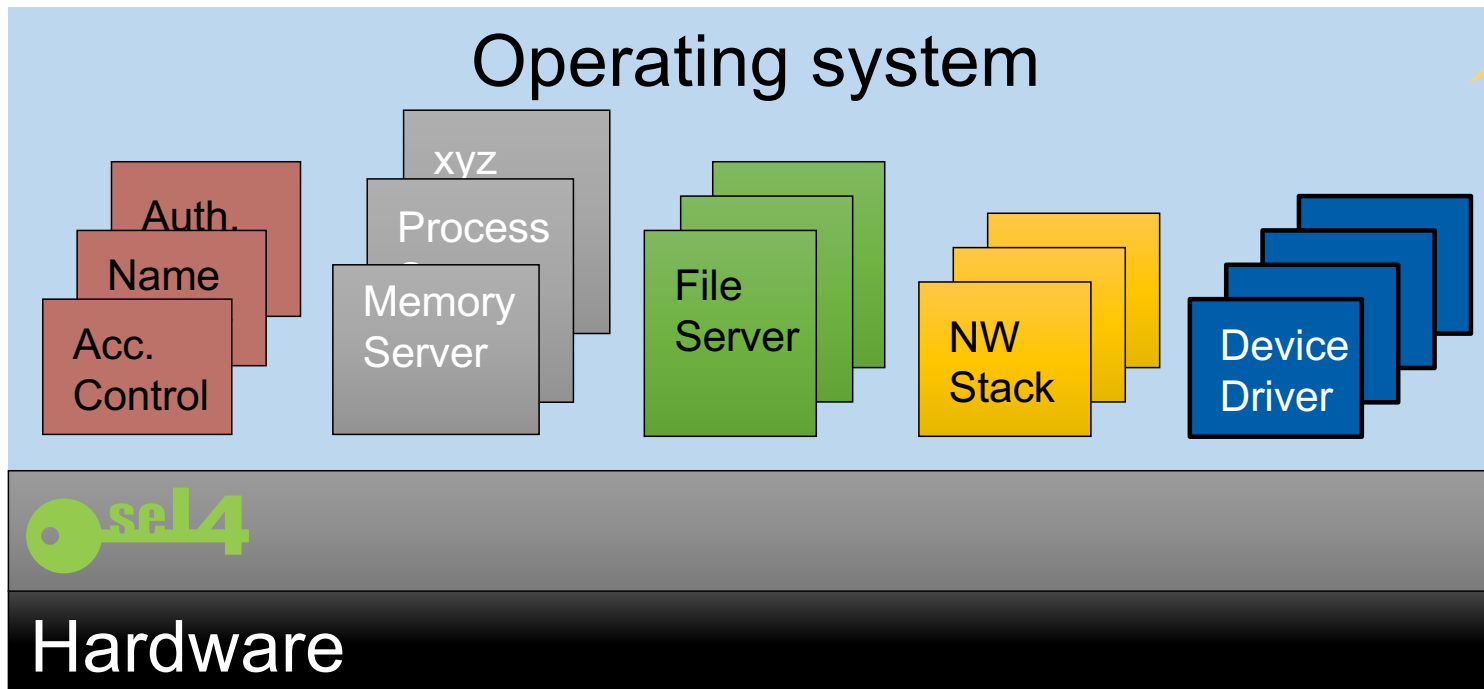
115 critical
Linux CVEs to
Nov'17

- For each establish how microkernel-based design would change impact

seL4 Hypothetical seL4-based OS

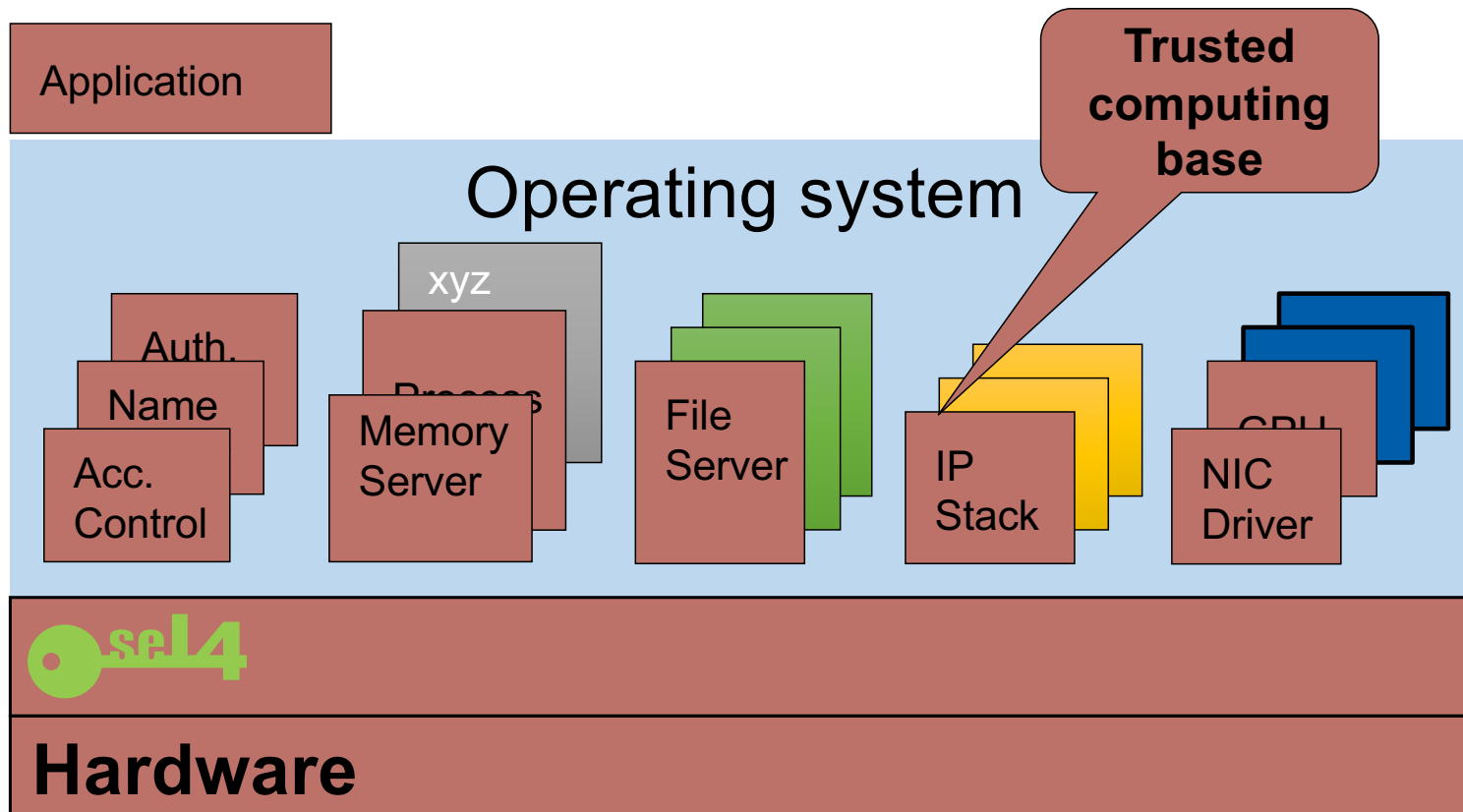
OS structured in *isolated* components, minimal inter-component dependencies, *least privilege*

Functionality comparable to Linux





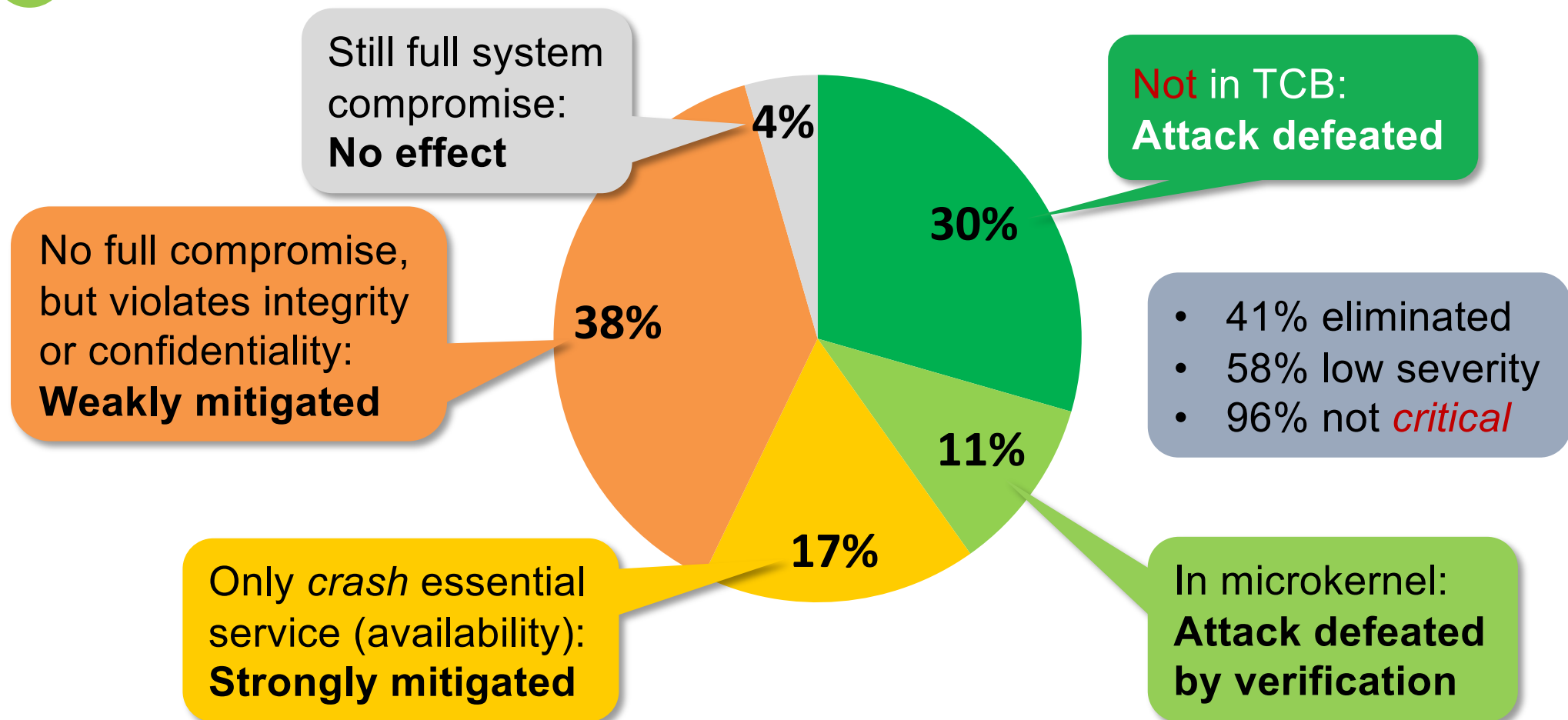
Hypothetical Security-Critical App



- App requires:
- IP networking
 - File storage
 - Display output



All Critical Linux CVEs to 2017



Conclusion: OS Structure Matters

- Microkernels definitely improve security
- Microkernel verification improves further
- Monolithic OS design is *fundamentally flawed from security point of view*

[Biggs et al., APSys'18]

Use of a monolithic OS in security- or safety-critical scenarios is professional malpractice!

Problem:
Where's the
seL4 OS?

