Provably Trustworthy Systems

seL4 and beyond

Gerwin Klein

Royal Society Meeting on Verified trustworthy software systems
April 2016

Formal verification of real systems is happening!

Formal verification of real systems

- **Increasingly many examples:**

Formal verification of real systems

- **Increasingly many examples:**

  - **seL4**
    - verified OS kernel implementation

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  - verified OS kernel implementation

- **CompCert**
  - verified compiler implementation

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  verified OS kernel implementation

- **CompCert**
  – verified compiler implementation

- **Ironfleet and Ironclad**
  – verified distributed system

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  verified OS kernel implementation

- **CompCert**
  – verified compiler implementation

- **Ironfleet and Ironclad**

- **CakeML**

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  - verified OS kernel implementation

- **CompCert**
  - verified compiler implementation

- **Ironfleet and Ironclad**

- **CakeML**

- **Candle**
  - verified interactive HOL theorem prover implementation

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  - verified OS kernel implementation

- **CompCert**
  - verified compiler implementation

- **Ironfleet and Ironclad**

CAKEML

A Verified Implementation of ML

- **PolarSSL**
  - verified SSL implementation

TRUST IN SOFT VERIFIED

le
- verified interactive HOL theorem prover implementation

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  - verified OS kernel implementation

- **CompCert**
  - verified compiler implementation

- **Ironfleet and Ironclad**

A Verified Implementation of ML

- **PolarSSL**
  - verifi

ve HOL theorem prover

- **CoCon**
  - verified conference system

Formal verification of real systems

- **Increasingly many examples:**

- **seL4**
  - verified OS kernel implementation

- **CompCert**
  - verified compiler implementation

- **Ironfleet and Ironclad**

CAKEML

A Verified Implementation of ML

- **PolarSSL**
  - verifi

POLARSSL

Straightforward, Secure Communication

ve HOL theorem prover

- **CoCon**
  - verified conference system

Formal verification of real systems

- Increasingly many examples:

- **seL4**
  verified OS kernel implementation

- **CompCert**
  – verified compiler implementation

- **Ironfleet and Ironclad**

CAKEML

A Verified Implementation of ML

- **PolarSSL**
  – verifi

- **CoCon**
  – verified conference system

ve HOL theorem prover

- **OpenSSL HMAC**
  – verified crypto implementation

Formal verification of real systems

- Increasingly many examples:

- seL4
  verified OS kernel implementation

- CompCert
  - verified compiler implementation

- Ironfleet and Ironclad

- PolarSSL
  - verifi

CAKEML

A Verified Implementation of ML

ve HOL theorem prover

- CoCon

- FSCQ
  - verified crash resistance file system

OpenSSL HMAC
– verified crypto implementation

But:

Still far from mainstream

Too Expensive

- **Such projects are still big research results**
  - Often break new ground
  - Multiple person years or person decades
  - Real, binary-level results still rare
  - Hard to maintain over long periods

Too Expensive

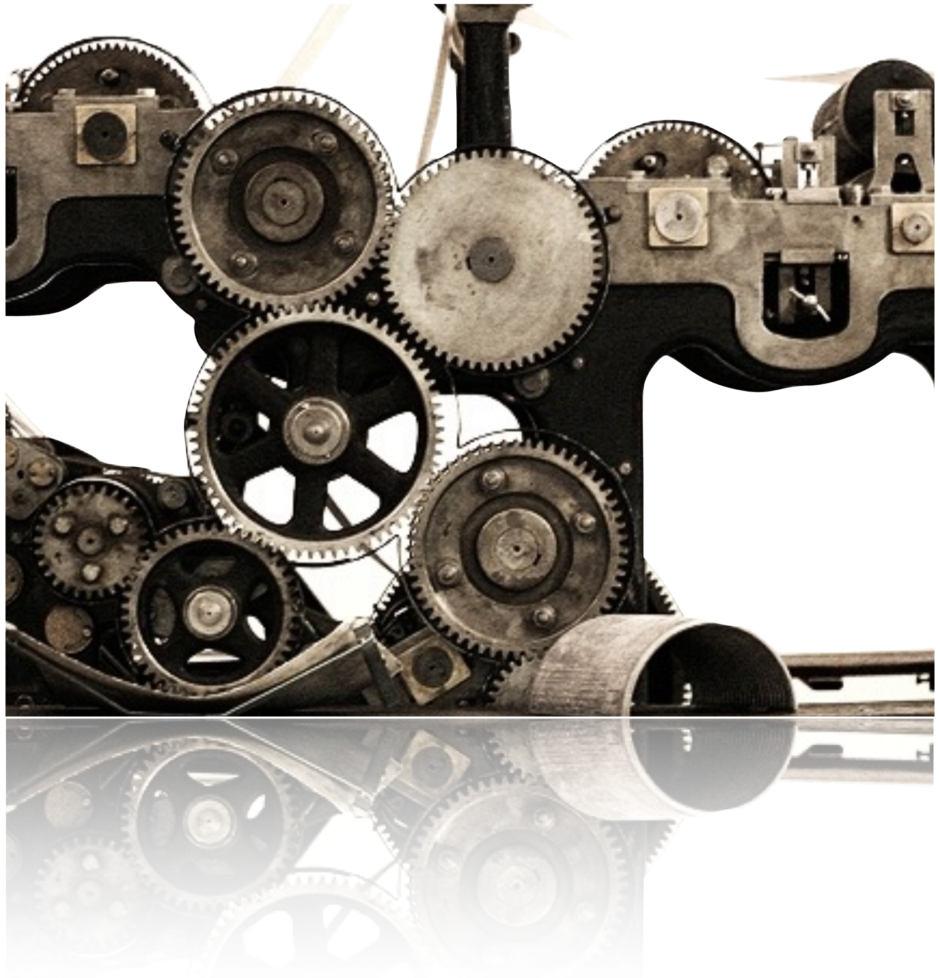
- **Such projects are still big research results**
  - Often break new ground
  - Multiple person years or person decades
  - Real, binary-level results still rare
  - Hard to maintain over long periods

- **Still too expensive**
  - But not that far off:
    - **cheaper than traditional high-assurance dev**
    - **factor 2-3 over high-quality traditional embedded systems dev**

What can be done?

Better, cheaper, faster.

- **Just needs to be cheaper:**
  - economic pressure wins over time
  - everything else follows



Better, cheaper, faster.

- **Just needs to be cheaper:**
  - economic pressure wins over time
  - everything else follows

- **Proof Productivity:**
  - Tools
    - more automation, deeper automation, built for scale
  - Proof Engineering
    - predictability, estimation, scale
  - Languages
    - design for verification, increase verification productivity
  - …

The rest of this talk

- seL4
- Scale
- Proof Engineering
- Proof Effort
- Future

seL4

seL4: Isolation

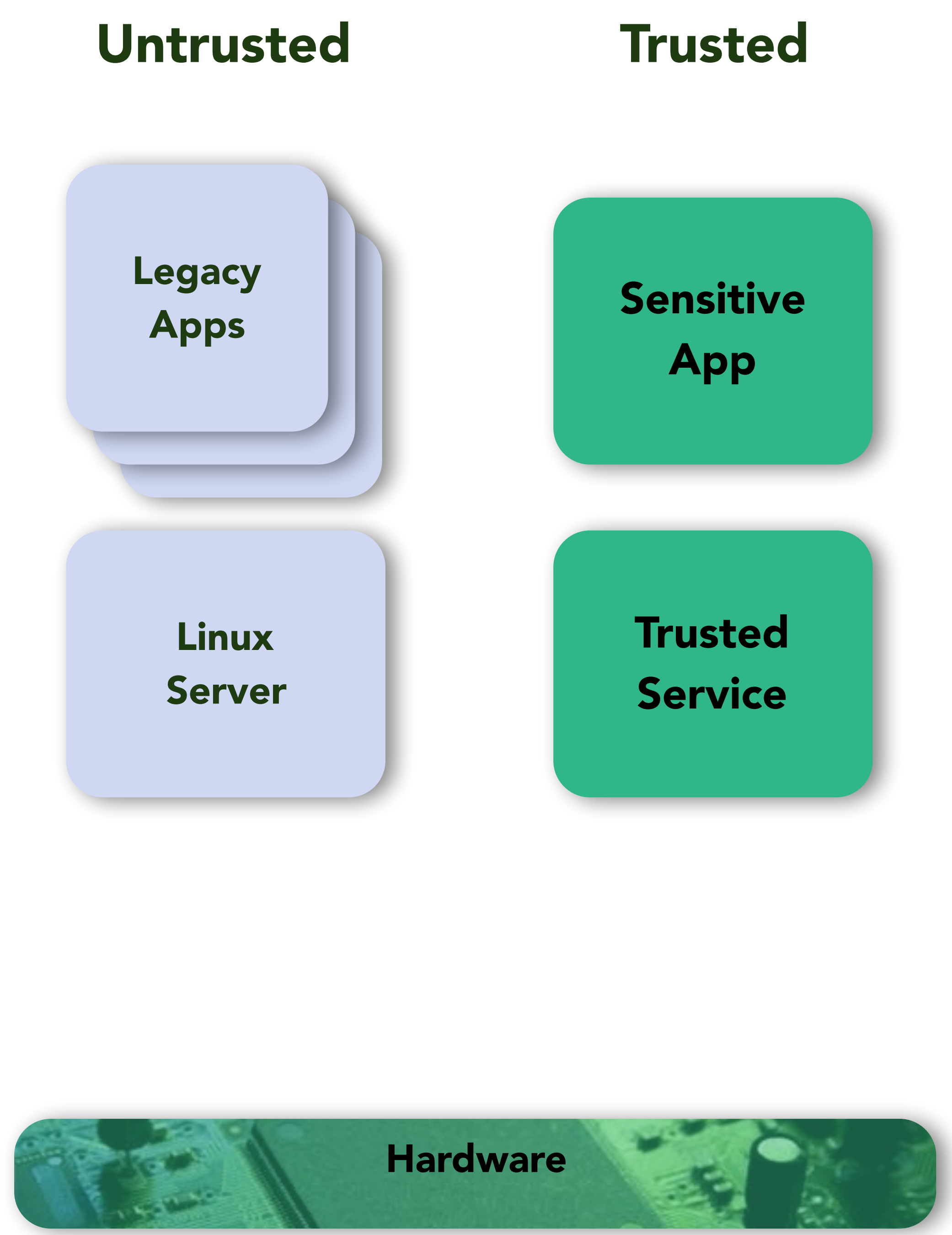
Trustworthy Computing Base

- message passing
- virtual memory
- interrupt handling
- access control

Applications

- fault isolation
- fault identification
- IP protection
- modularity

Trusted next to Untrusted



seL4: Isolation

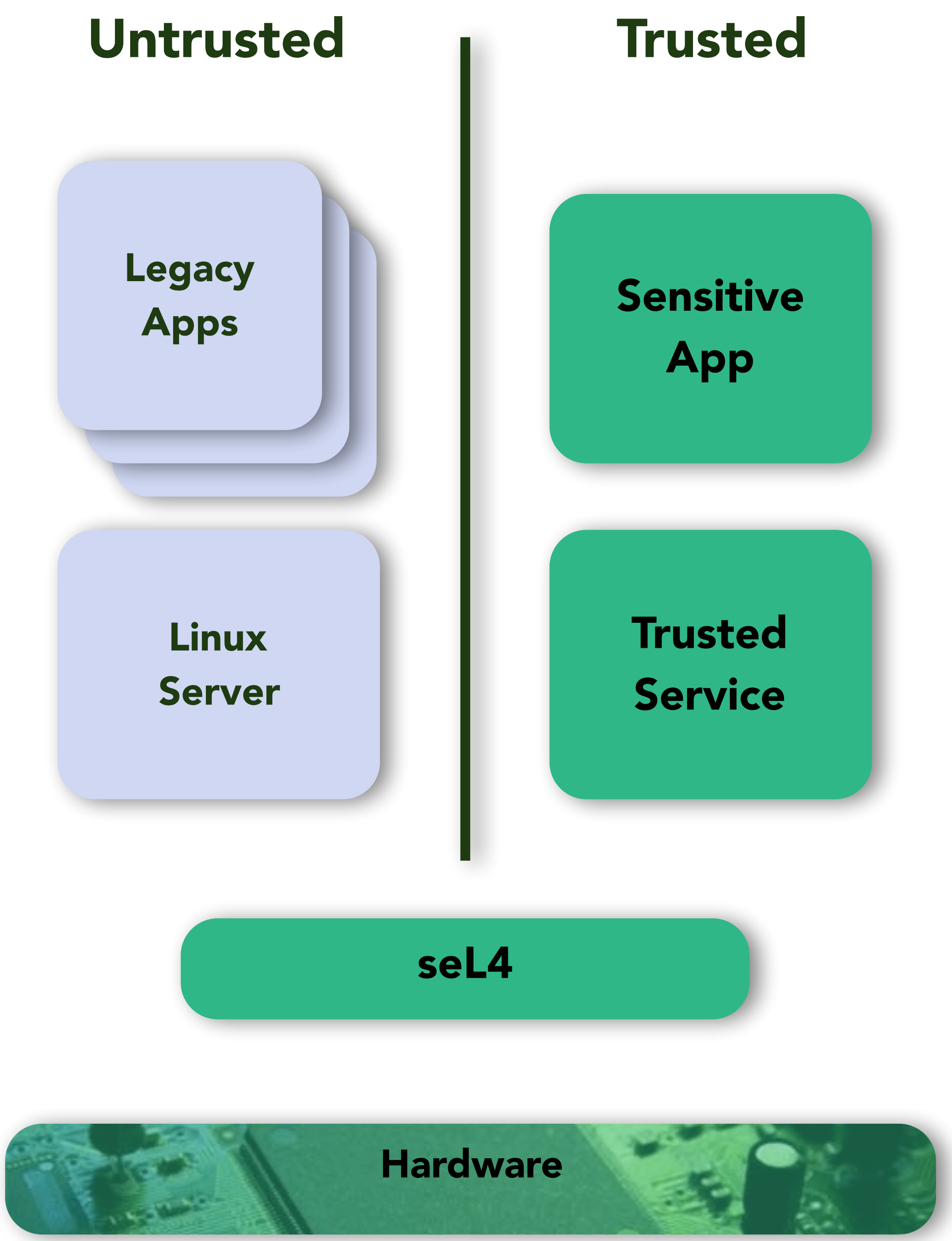
Trustworthy Computing Base

- message passing
- virtual memory
- interrupt handling
- access control

Applications

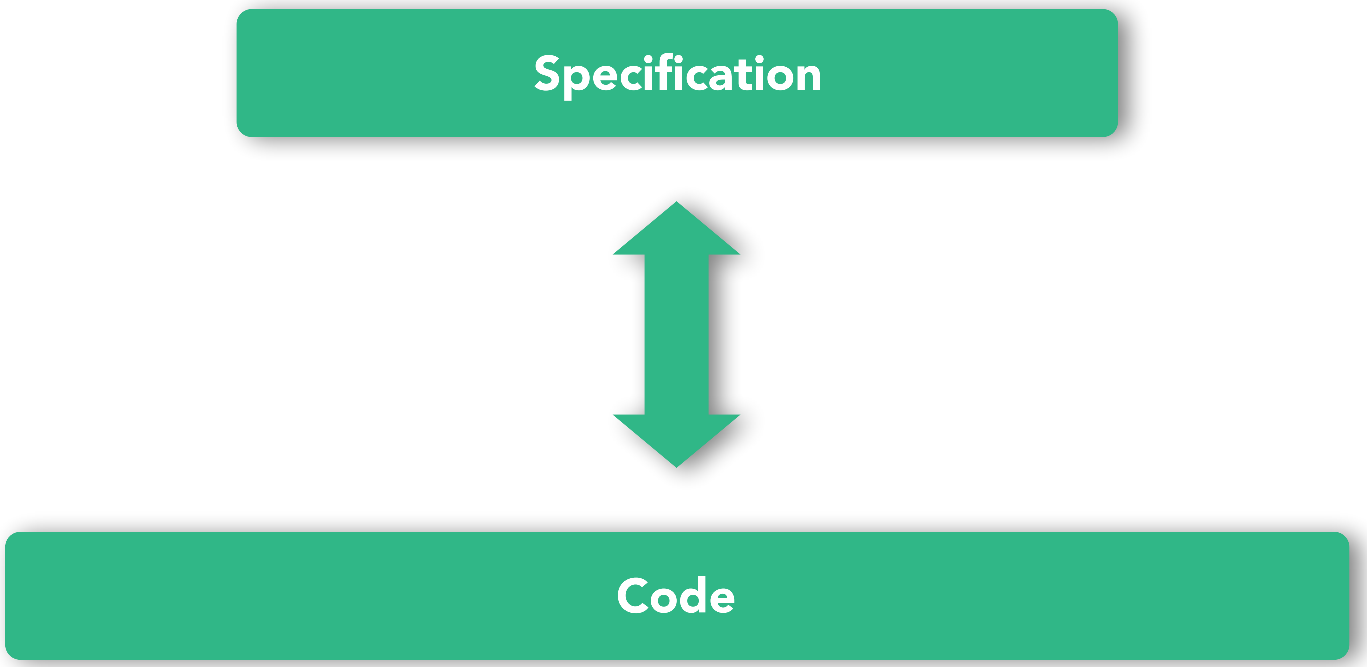
- fault isolation
- fault identification
- IP protection
- modularity

Trusted next to Untrusted



Functional Correctness

Proof



Functional Correctness

What

Specification

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Proof

Code

Functional Correctness

What

Specification

```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Proof

How

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```

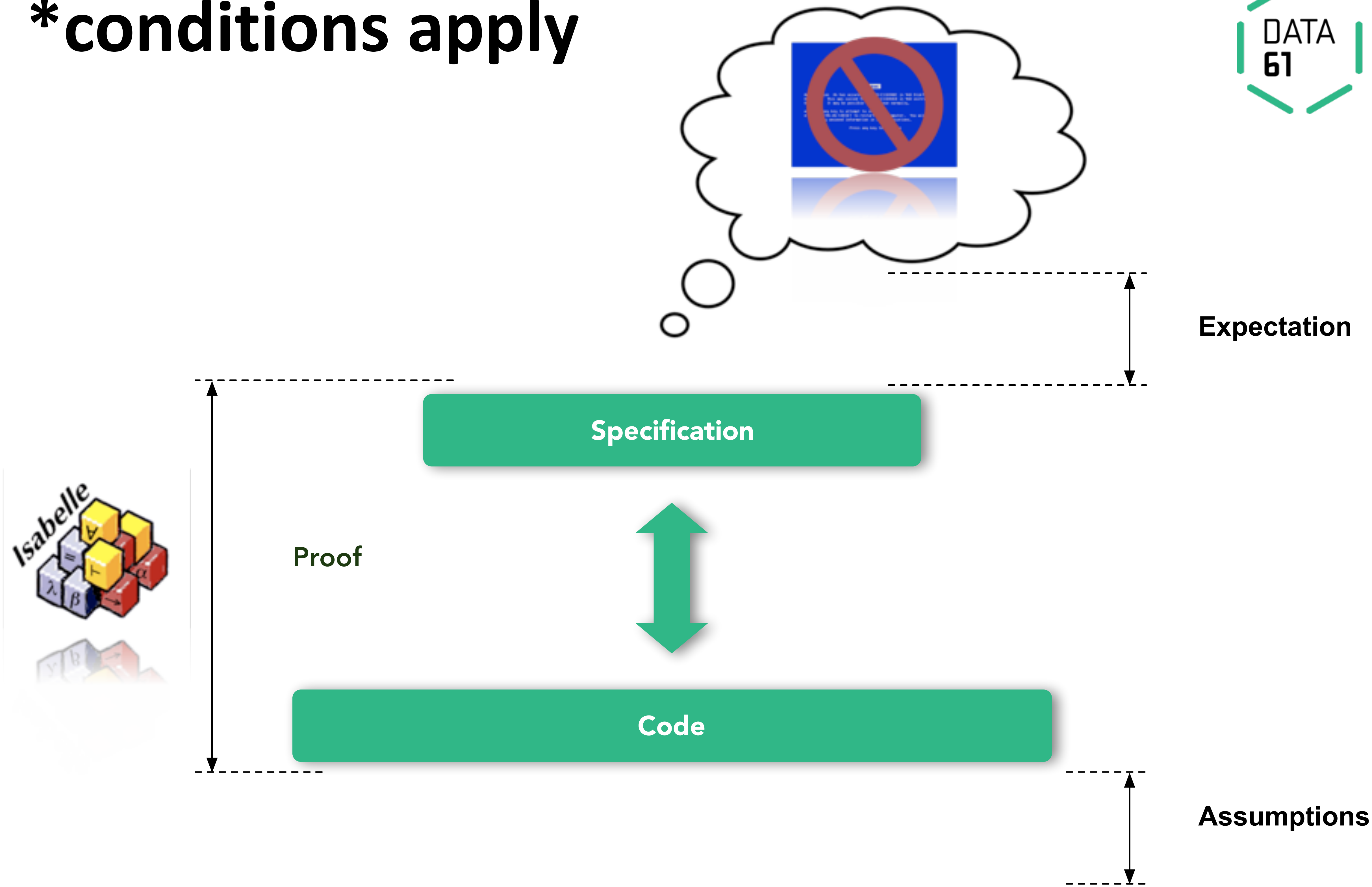
*conditions apply

Specification

Proof

Code

*conditions apply



*conditions apply

Assume correct:

- ~~compiler + linker (wrt. C op sem)~~
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

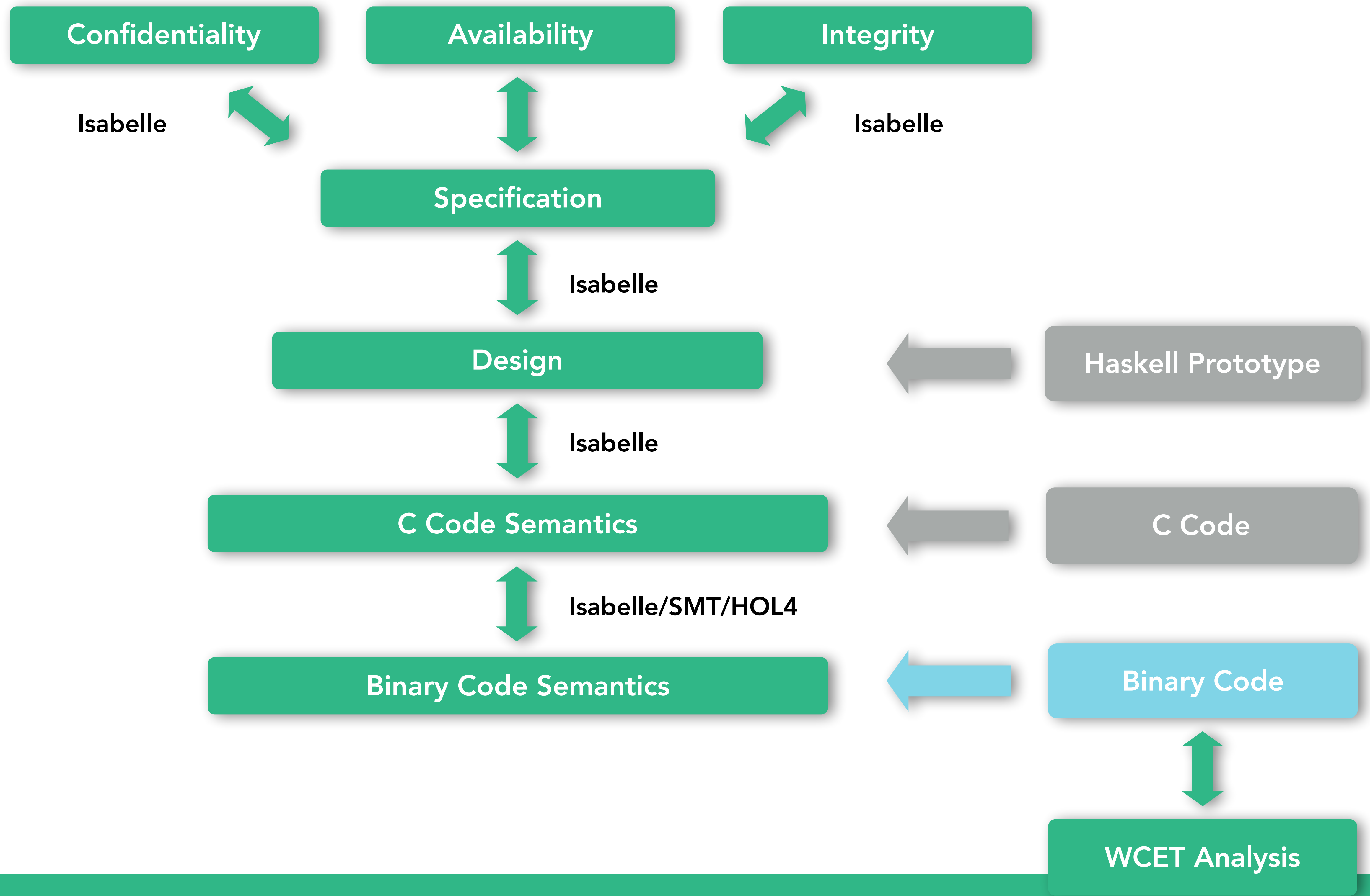
Specificatio

Proof

Code

Assumptions

Proof Architecture Now



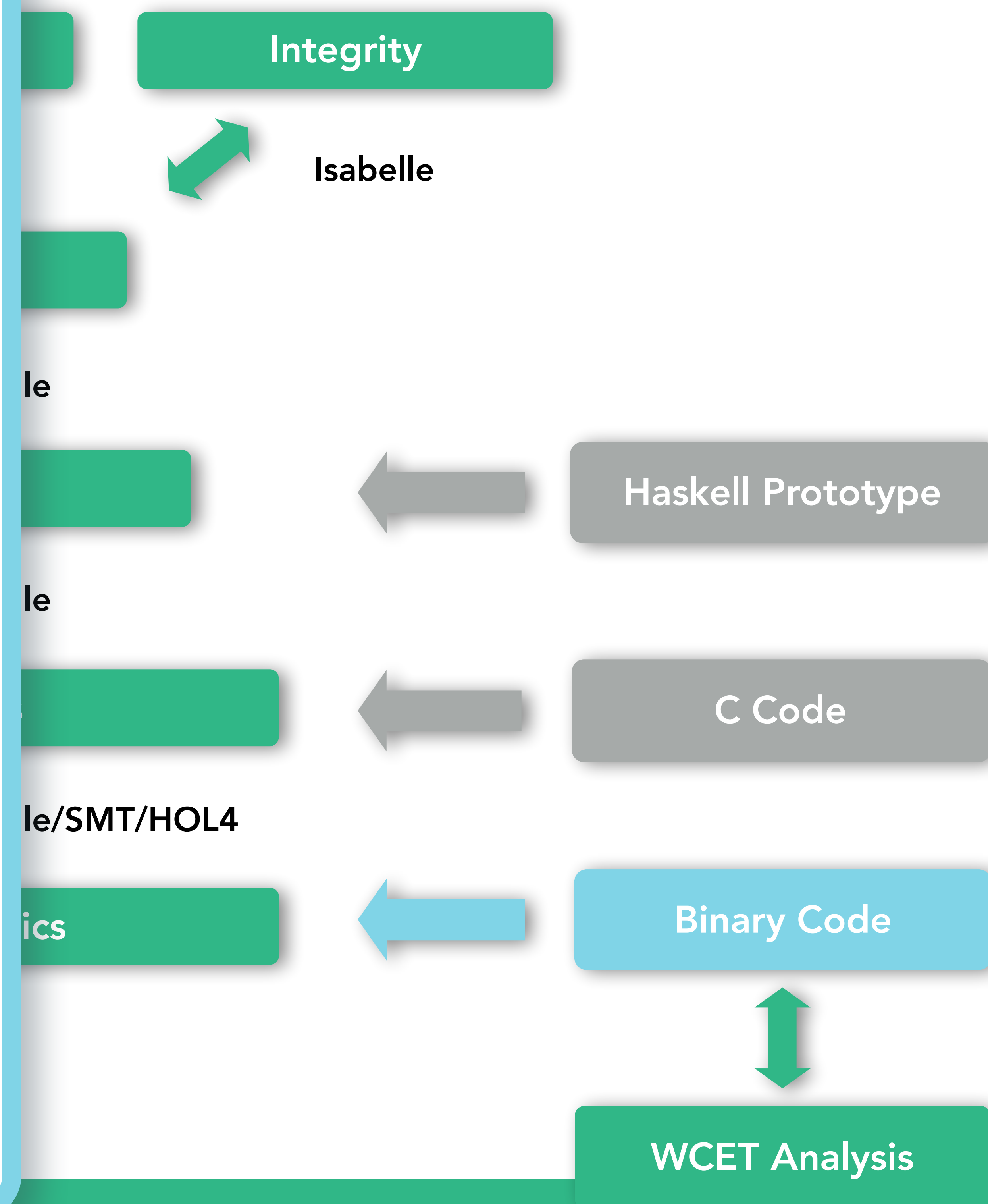
Proof Architecture Now

High-level properties:

- functional correctness
- integrity
- authority confinement
- non-interference
- termination
- user-level system initialisation
- verified component platform
- worst-case execution time (by static analysis)

Roadmap:

- verified x64 version
- virtualisation extensions
- mixed-criticality real-time
- timing side-channel elimination



Proof Architecture Now

High-level properties:

- functional correctness
- integrity
- authority confinement
- non-interference
- termination
- user-level system initialisatio
- verified component platform
- worst-case execution time (by static analysis)

Roadmap:

- verified x64 version
- virtualisation extensions
- mixed-criticality real-time
- timing side-channel elimination

Open Source

http://seL4.systems
https://github.com/seL4/

Integrity
le/SMT/HOL4
ics
Binary Code
WCET Analysis

As Real as it Gets

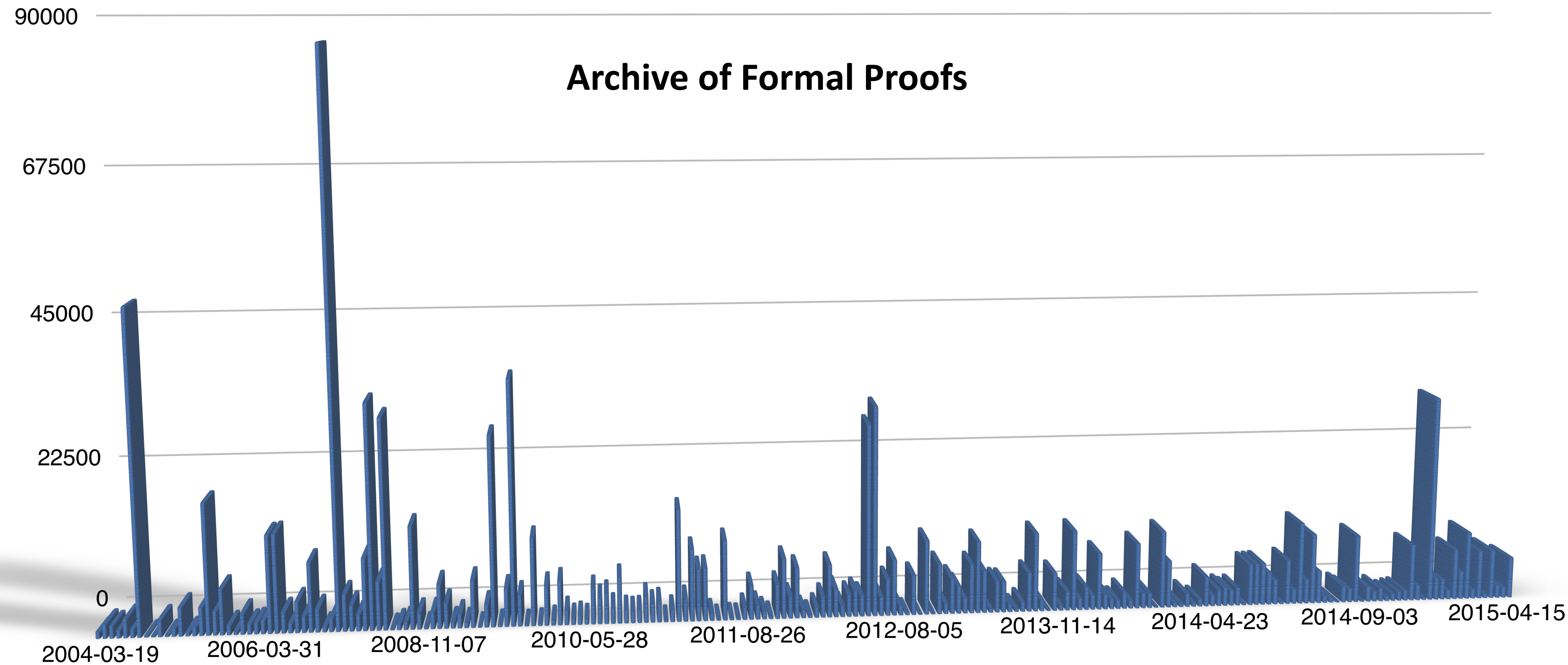
- **Autonomous in**

As Real as it Gets

- Autonomous in 3, 2, 1..

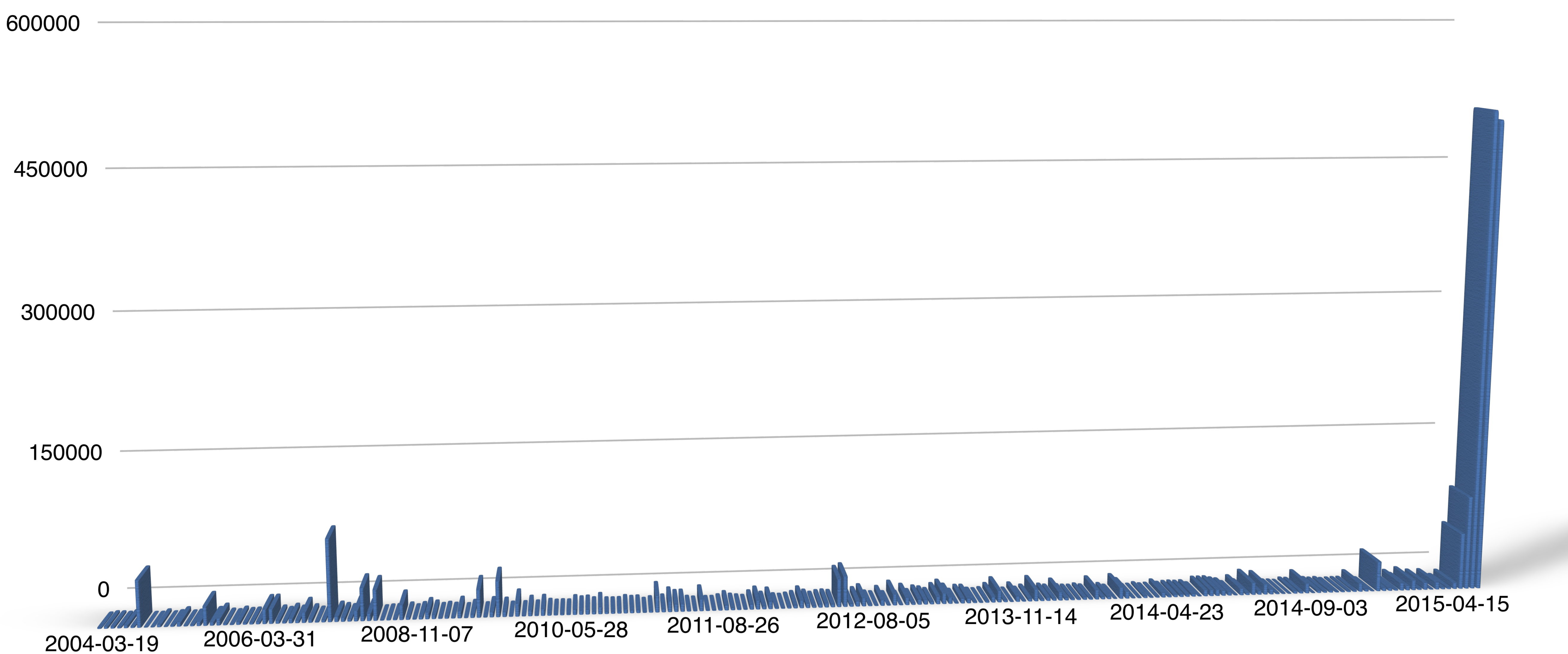
Scale

Scale



size of AFP entries by submission date

Scale

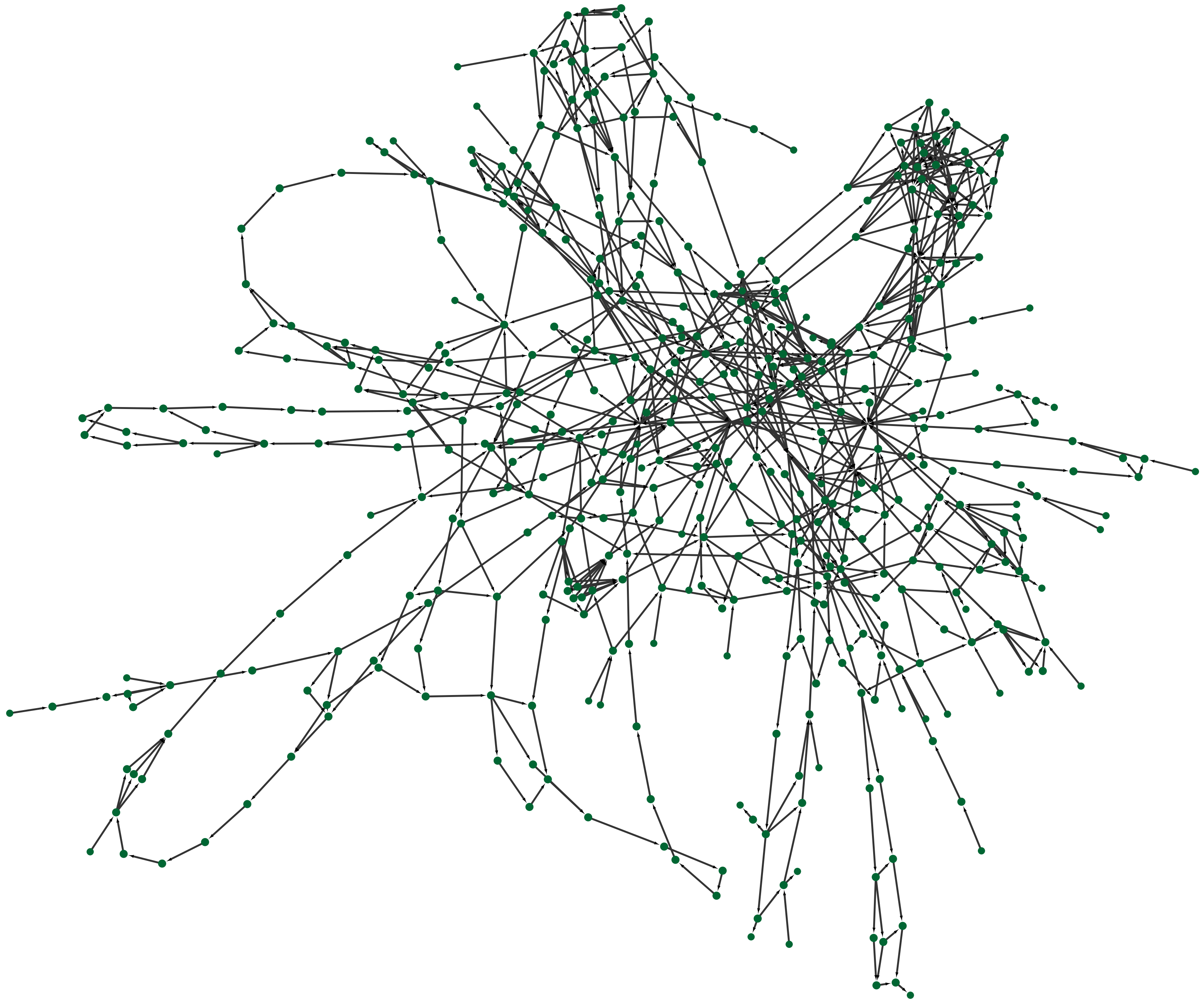


size of AFP entries by submission date

with Four-Colour theorem, Odd-Order theorem, Verisoft, seL4

Proof Introspection

- 500 files
- 22,000 lemmas stated
- 95,000 lemmas proved



Proof Introspection

- 500 files
- 22,000 lemmas stated
- 95,000 lemmas proved

Raf's Observation

The introspection of proof and theories is an essential part of working on a large-scale verification development.

- Learning Isabelle? **Easy.**
- Learning microkernels? **Not too bad.**
- Finding your way in the 500kloc proof jungle? **Hard!**

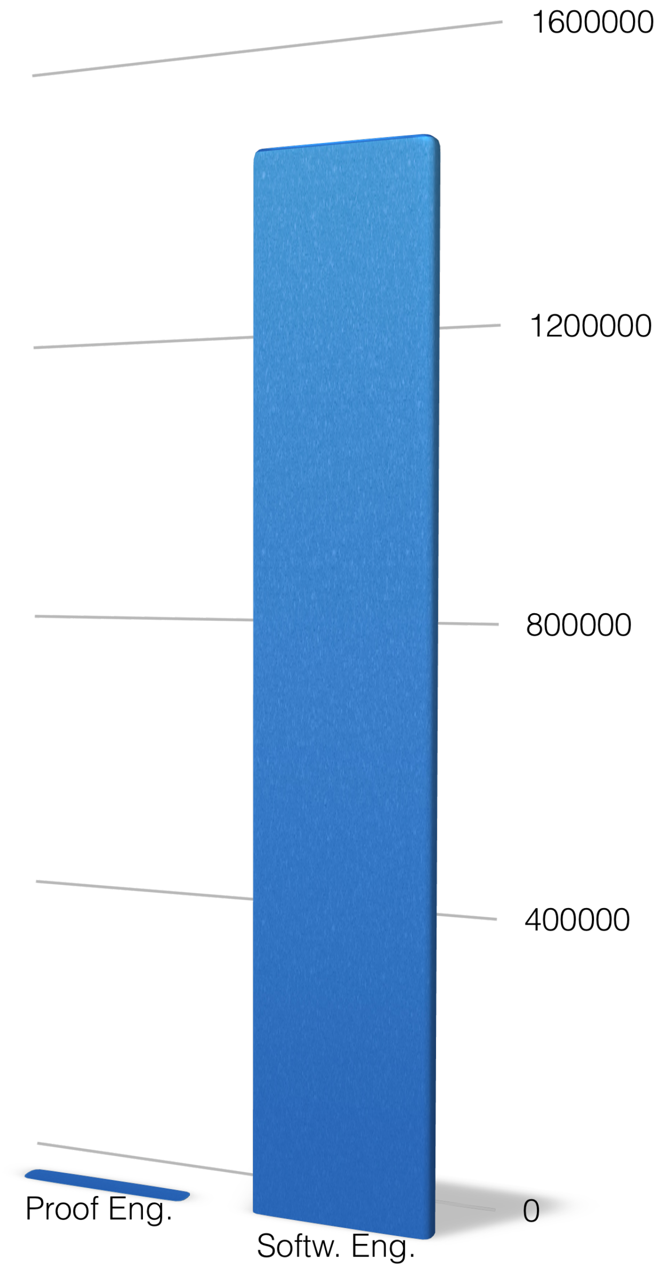
Proof Engineering

Software vs Proof Engineering

- **Is Proof Engineering a thing?**
  - Google Scholar:
    - "software engineering" 1,430,000 results

Software vs Proof Engineering

- **Is Proof Engineering a thing?**
  - Google Scholar:
    - "software engineering" 1,430,000 results
    - "proof engineering" 564 results



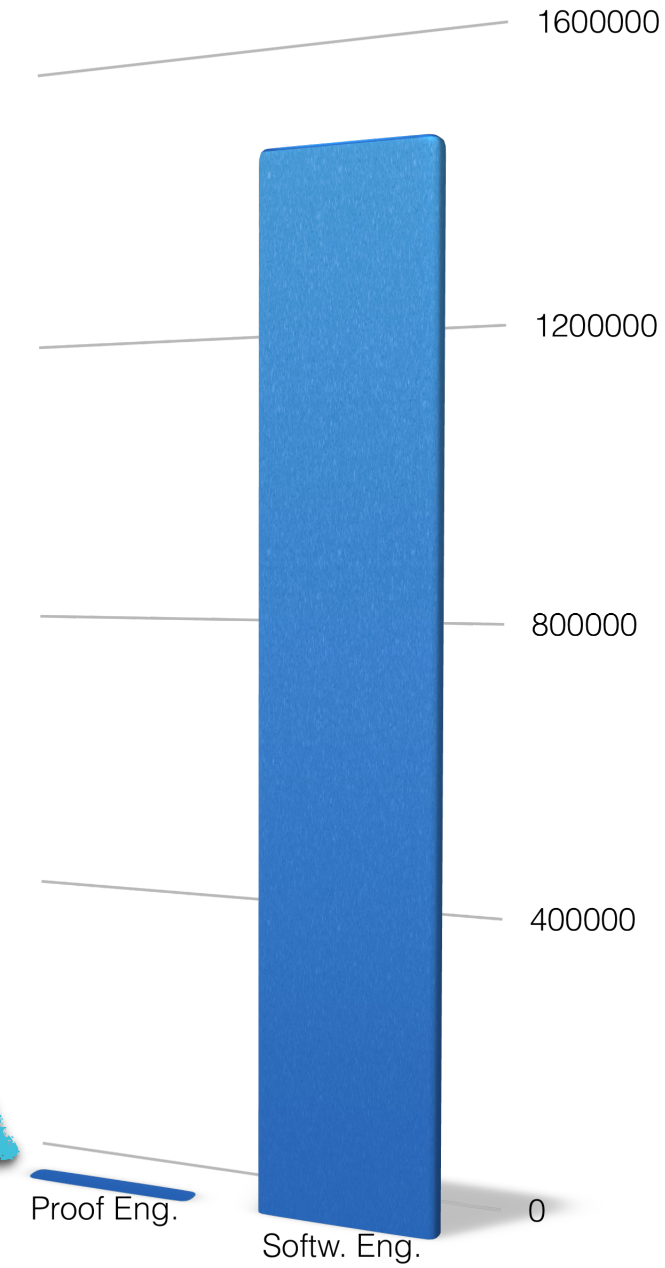
Software vs Proof Engineering

- **Is Proof Engineering a thing?**
  - Google Scholar:
    - "software engineering" 1,430,000 results
    - "proof engineering" 564 results

Includes

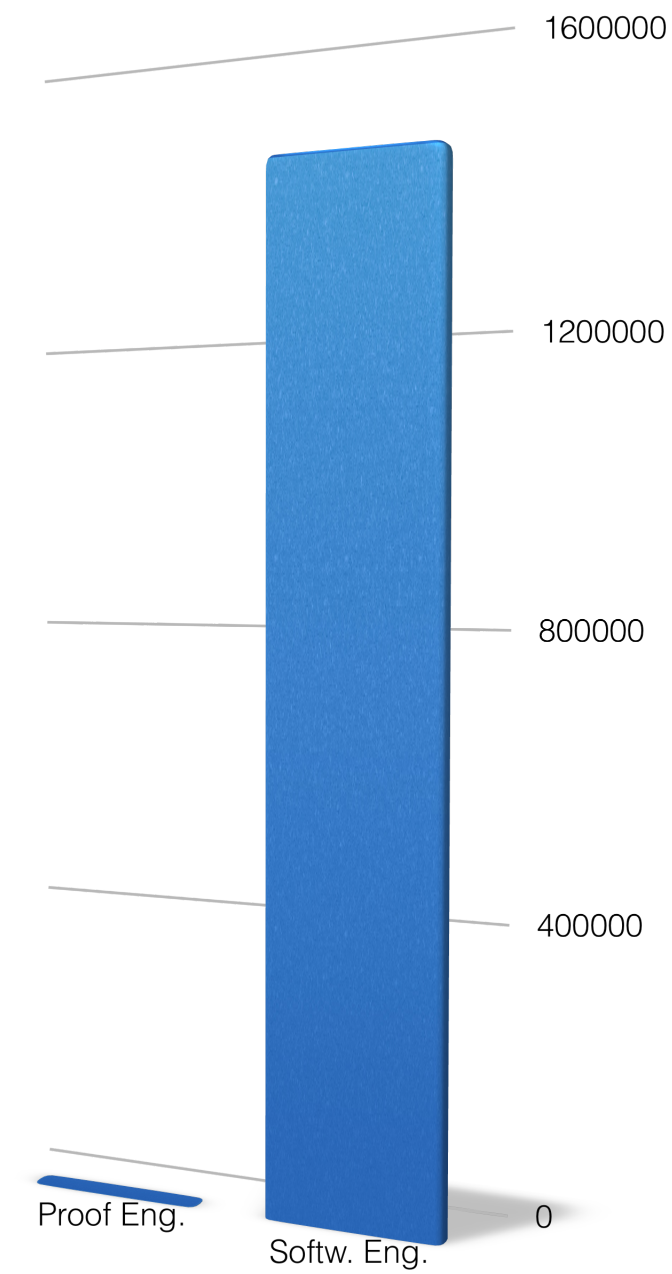
"The Fireproof Building" and

"Influence of water permeation and analysis of treatment for the Longmen Grottoes"



Proof Engineering is The Same

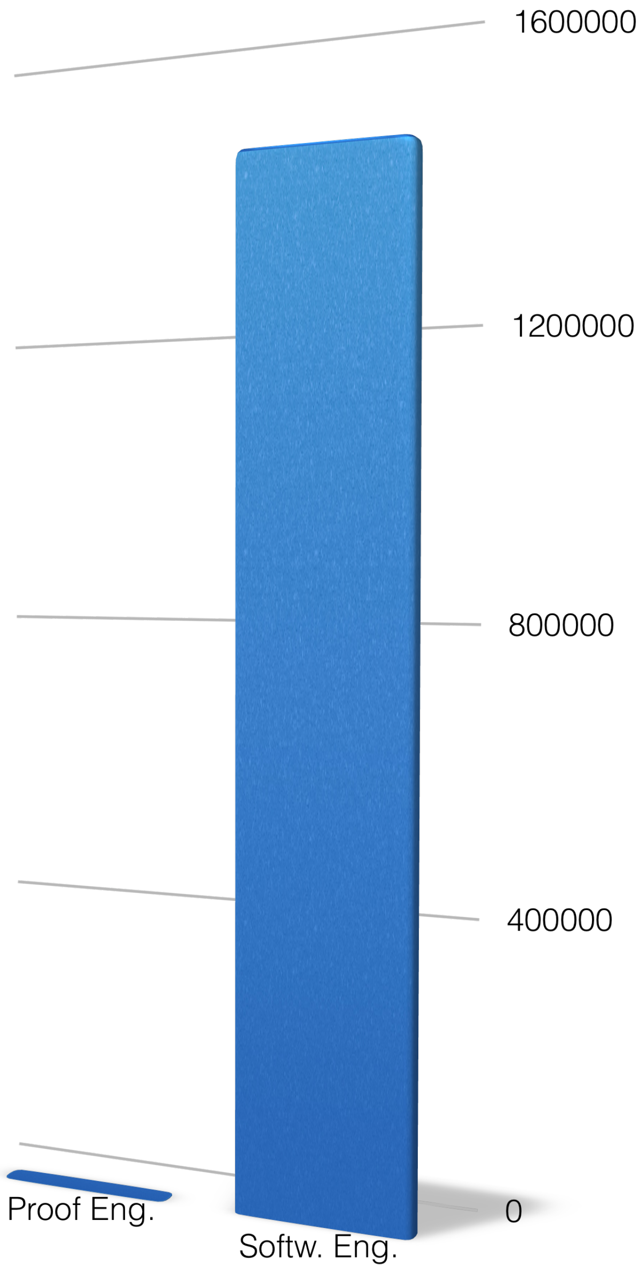
- **Same kind of artefacts:**
  - lemmas are functions, modules are modules
  - code gets big too
  - version control, regressions, refactoring and IDEs apply



Proof Engineering is The Same

- **Same kind of artefacts:**
  - lemmas are functions, modules are modules
  - code gets big too
  - version control, regressions, refactoring and IDEs apply

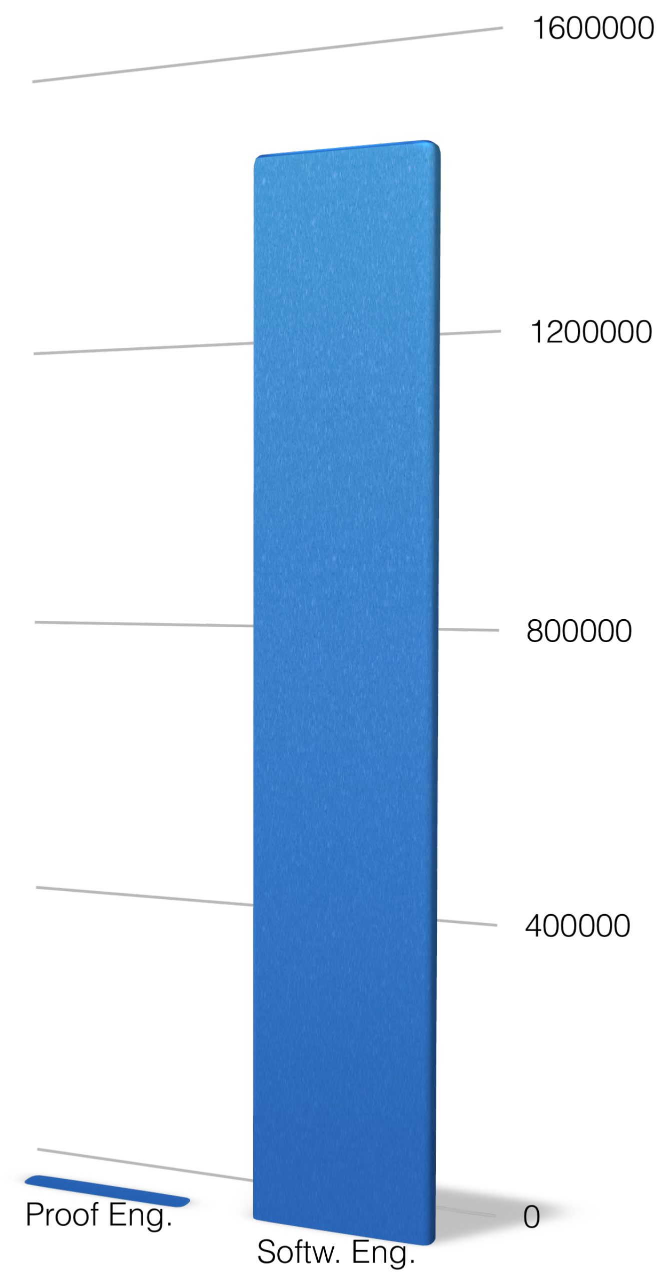
- **Same kind of problems**
  - managing a large proof base over time
  - deliver a proof on time within budget
  - dependencies, interfaces, abstraction, etc



Proof Engineering is Different

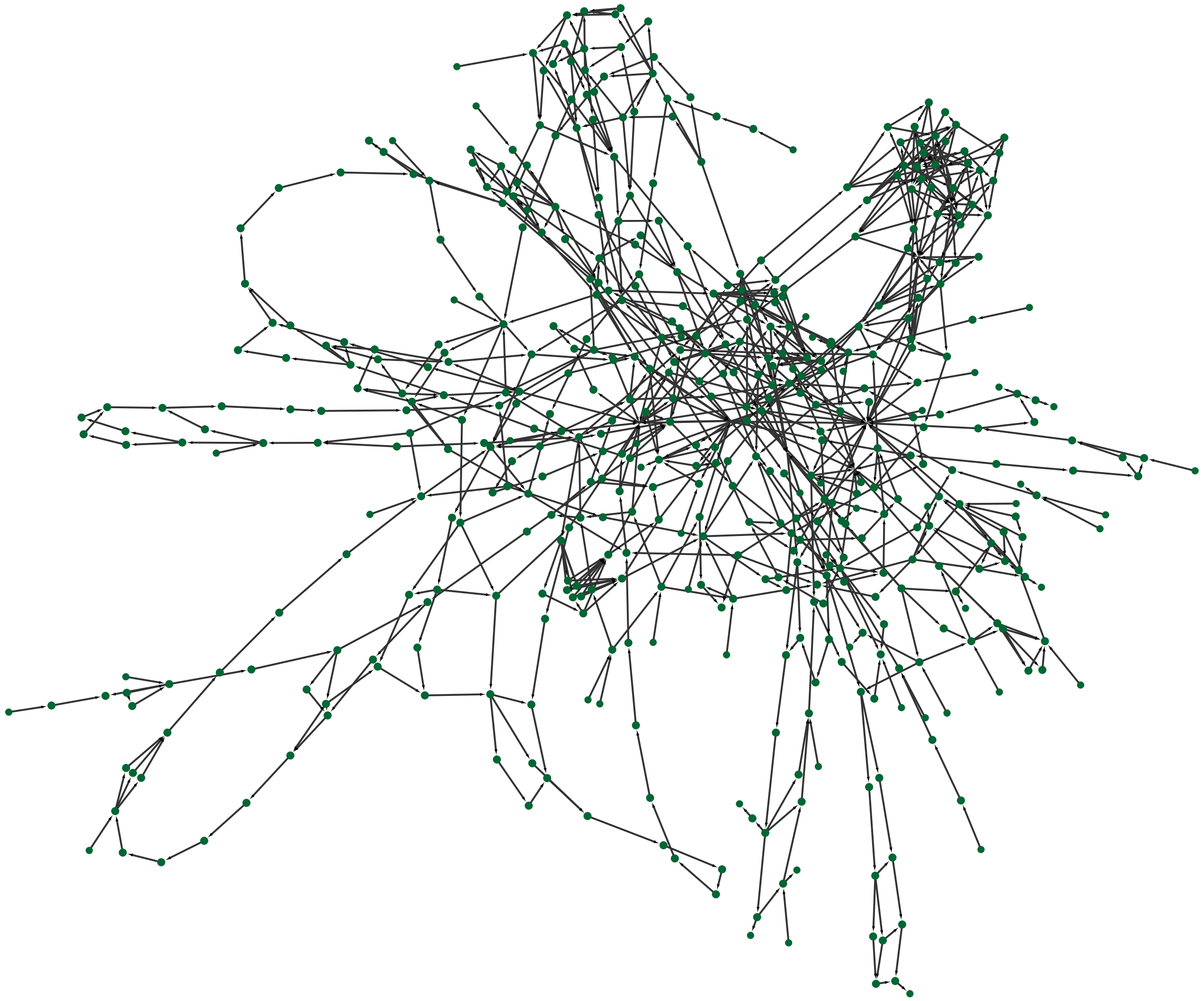
But: New Properties and Problems

- Results are checkable
  - You know when you are done!
  - No testing
  - 95% proof: no such thing
- More dead ends and iteration
- 2nd order artefact
  - Performance less critical
  - Quality less critical
  - Proof Irrelevance
- More semantic context
  - Much more scope for automation



Proof Development

- **Proof development**
  - decomposition of proofs over people,
  - custom proof calculus,
  - automating mechanical tasks, custom tactics
  - proof craft

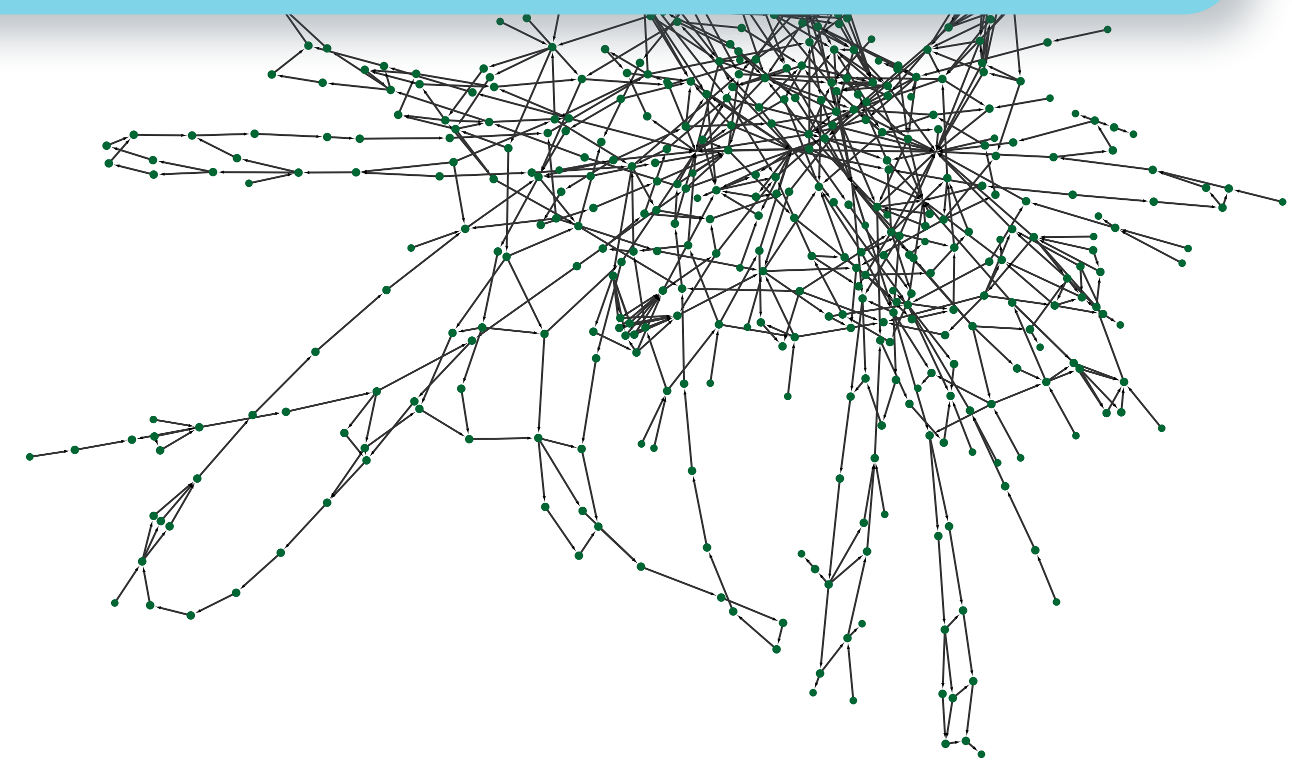


Proof Development

- **Proof development**
  - decomposition of proofs over peop
  - custom proof calculus,
  - automating mechanical tasks, cust
  - proof craft

Tim's Statement

Automating "donkey work" allows attention and effort to be focussed where most needed – but it must be done judiciously.



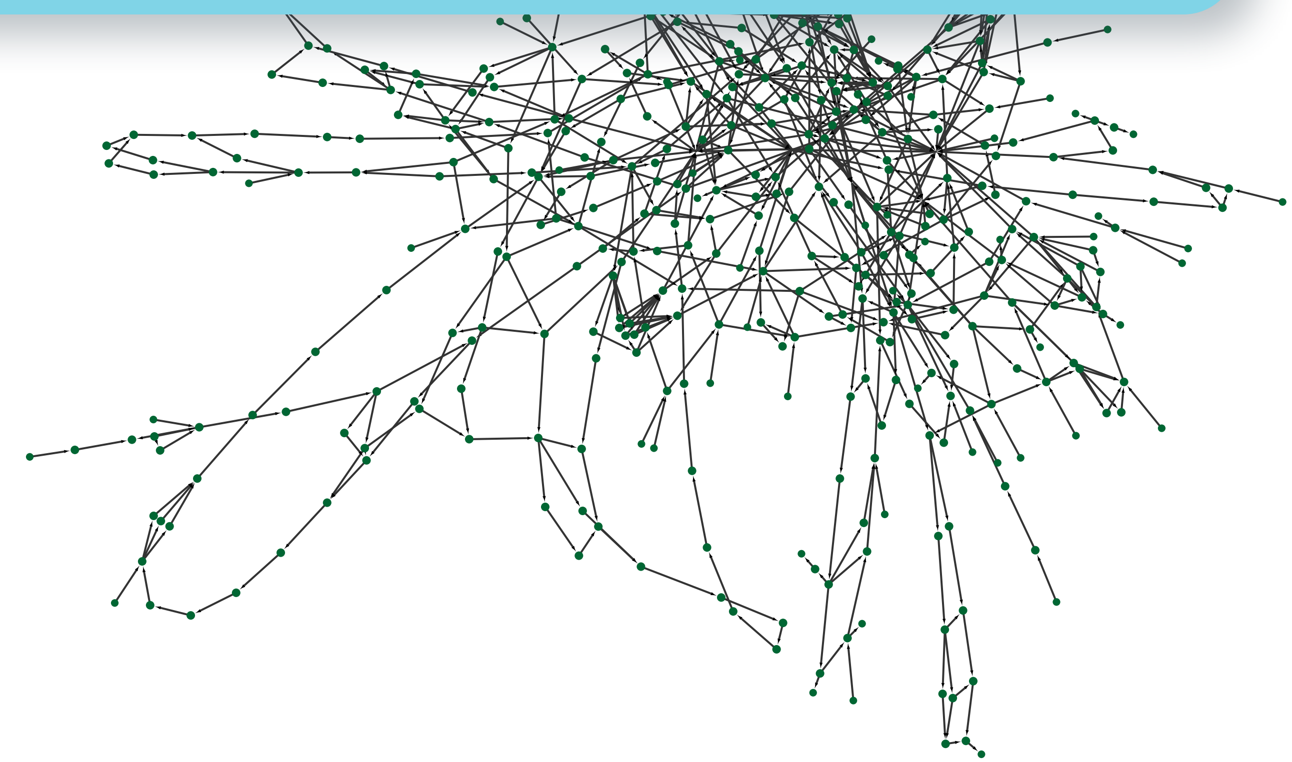
Proof Development

- **Proof development**
  - decomposition of proofs over peop
  - custom proof calculus,
  - automating mechanical tasks, cust
  - proof craft

- **Challenges**
  - non-local change,
  - speculative change,
  - distributed development

Tim’s Statement

Automating “donkey work” allows attention and effort to be focussed where most needed – but it must be done judiciously.



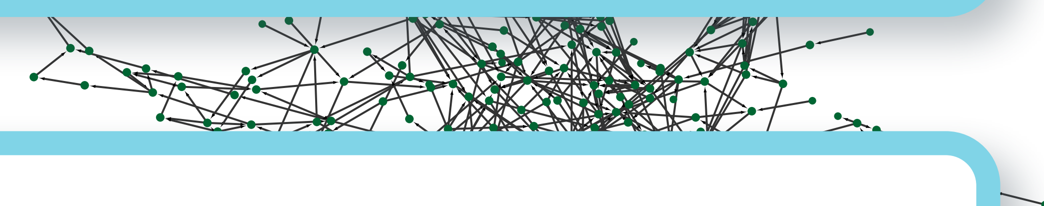
Proof Development

- **Proof development**
  - decomposition of proofs over peop
  - custom proof calculus,
  - automating mechanical tasks, cust
  - proof craft

- **Challenges**
  - non-local change,
  - speculative change,
  - distributed development

Tim's Statement

Automating "donkey work" allows attention and effort to be focussed where most needed – but it must be done judiciously.



Matthias' Conjecture

Over the years, I must have waited weeks for Isabelle. Productivity hinges on a short edit-check cycle; for that, I am even willing to (temporarily) sacrifice soundness.

Problems of Scale

- **Proof maintenance**
  - changes, updates, new proofs, new features
  - automated regression, keep code in sync
  - refactoring
  - simplification

- **Original proof:** 2005-2009
- **Maintenance:** 2009-2016 and counting

Problems of Scale

- **Proof maintenance**
  - changes, updates, new proofs, new features
  - automated regression, keep code in sync
  - refactoring
  - simplification

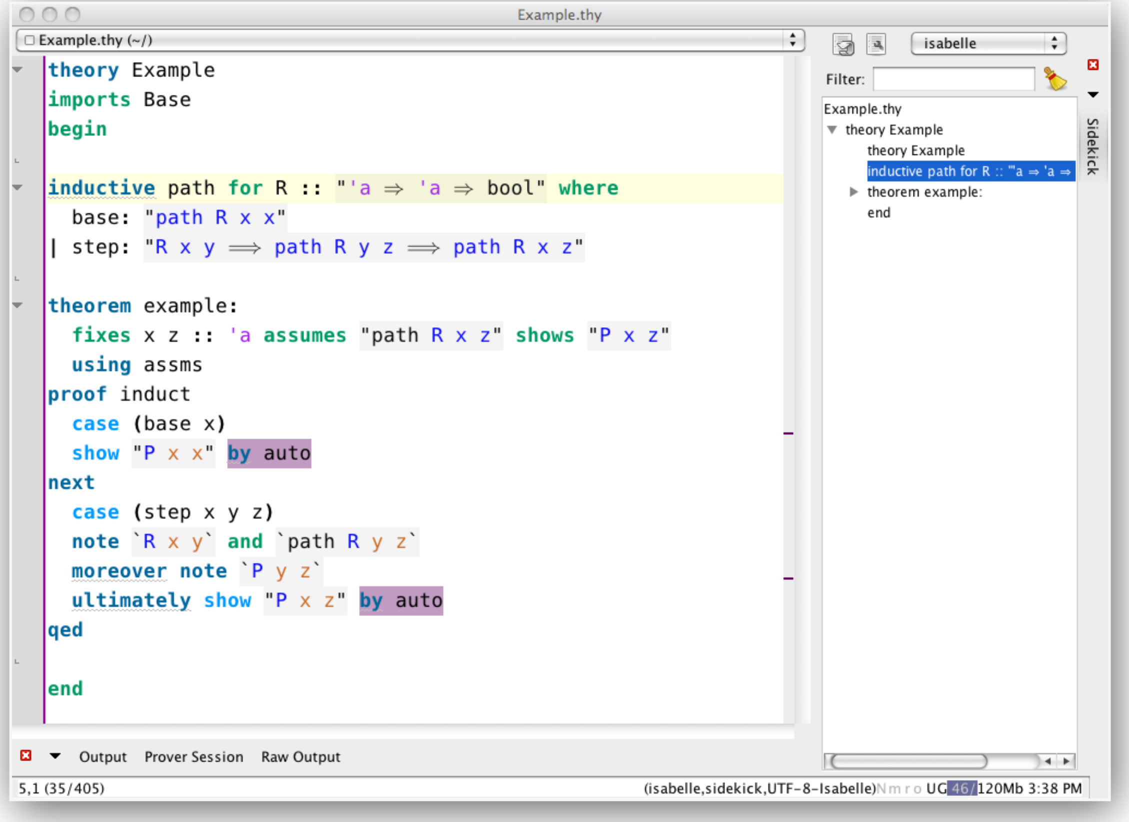
- **Original proof:** 2005-2009
- **Maintenance:** 2009-2016 and counting

Dan's Conclusion

Verification is fast, maintenance is forever.

Proof Engineering Tools

- **User Interface**
  - could proof IDEs be more powerful than code IDEs?
  - more semantic information
  - proof completion and suggestion?



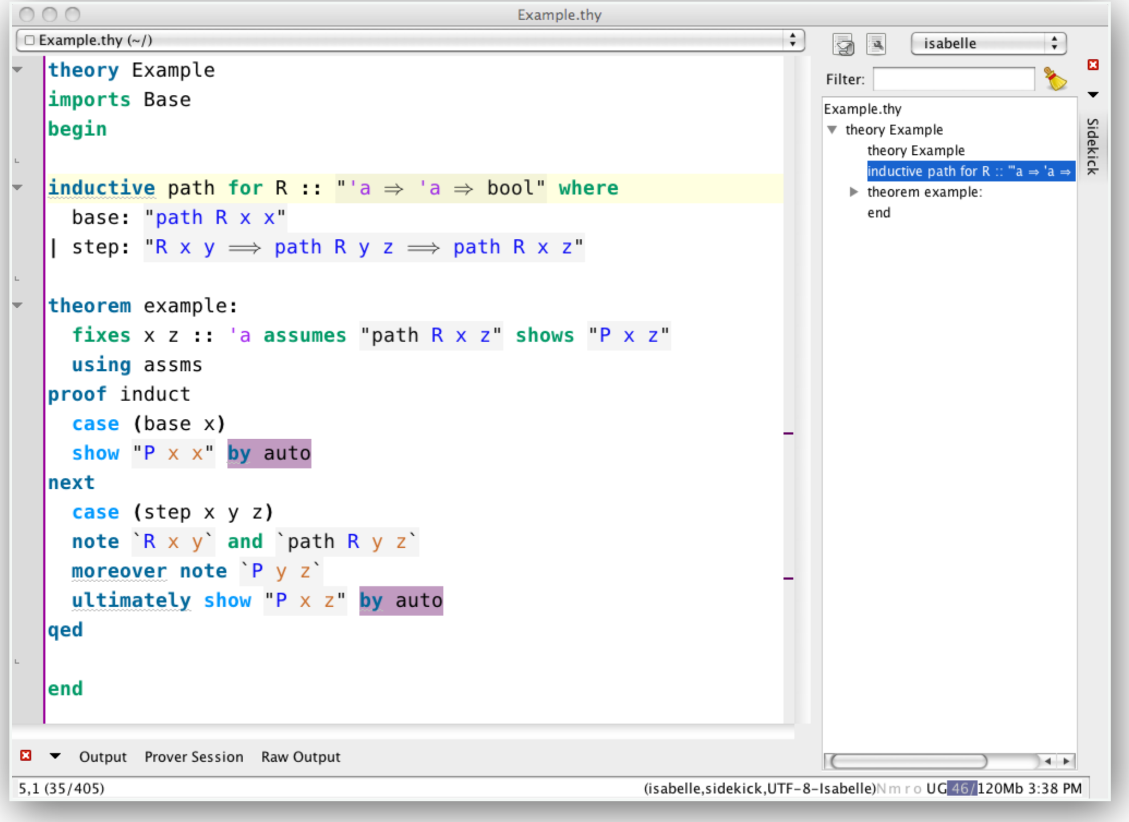
Proof Engineering Tools

User Interface

- could proof IDEs be more powerful than code IDEs?
- more semantic information
- proof completion and suggestion?

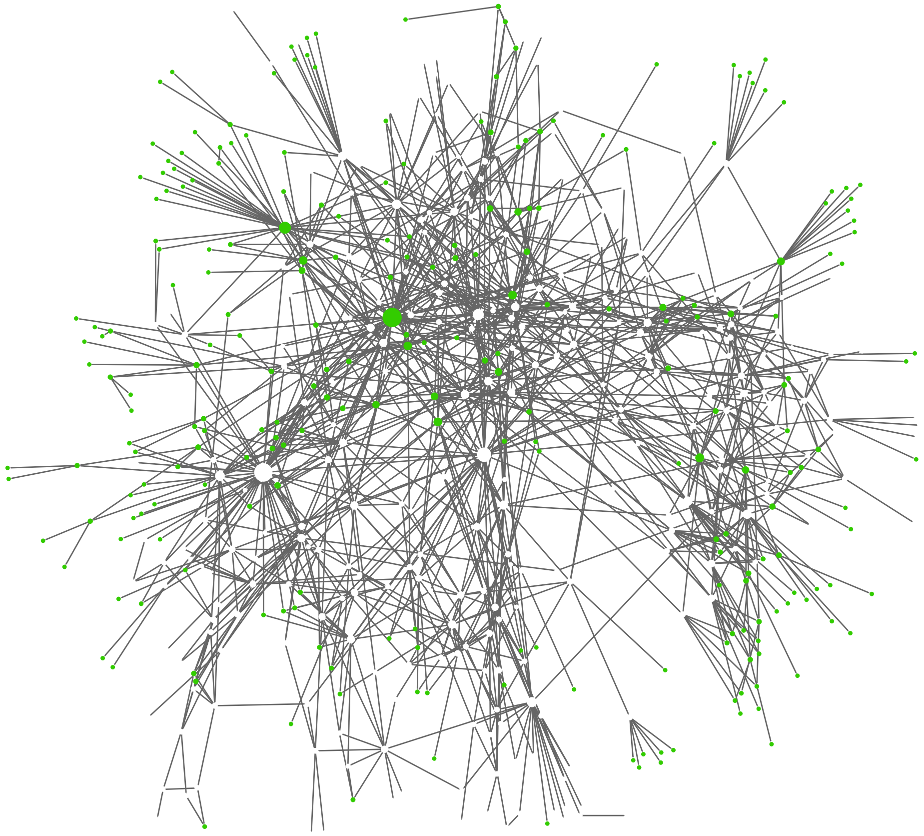
Refactoring

- less constrained,
  new kinds of refactoring possible, e.g.
  - move to best position in library
  - generalise lemma
  - recognise proof patterns



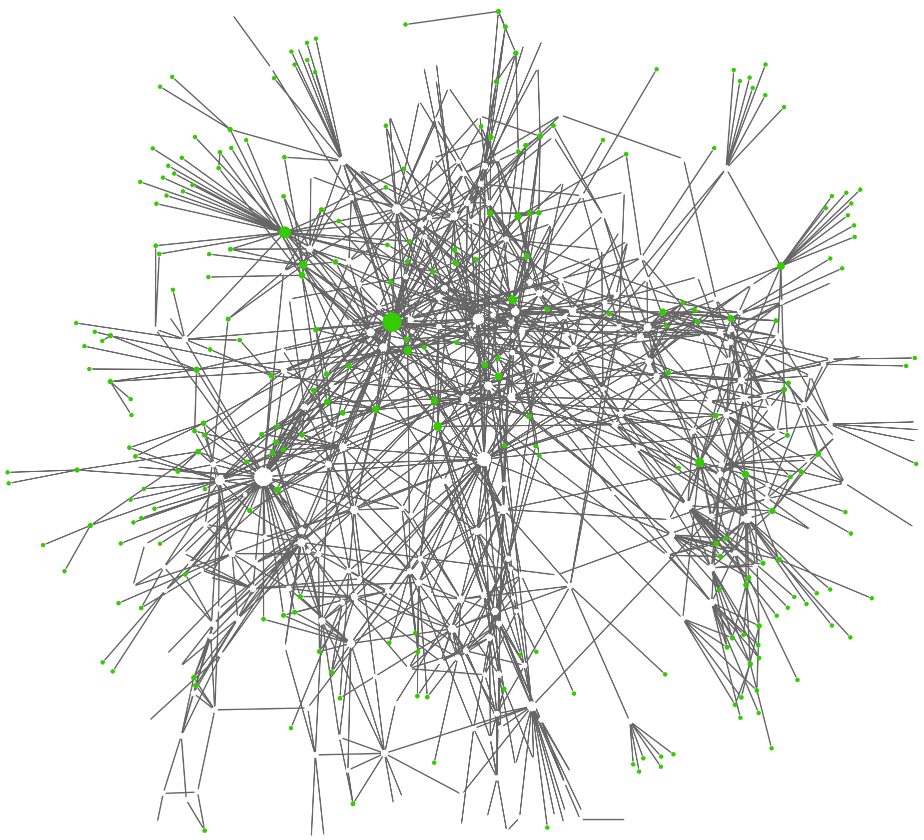
Proof Patterns

- **Large-scale Libraries**
  - architecture:
    - layers, modules, components, abstractions, genericity
  - proof interfaces
  - proof patterns



Proof Patterns

- **Large-scale Libraries**
  - architecture:
    - layers, modules, components, abstractions, genericity
  - proof interfaces
  - proof patterns



- **Technical Debt**
  - what does a clean, maintainable proof look like?
  - which techniques will make future change easier?
  - readability important? is documentation?

Proof Effort

Predictions

Can we predict for proofs:

- **how large will it be?**
- **how long will it take?**
- **how much will it cost?**

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of course not.

Many hard problems look deceptively easy.

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of course not.

Many hard problems look deceptively easy.

But maybe for program verification?

At least statistically, some of the time?

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of cours

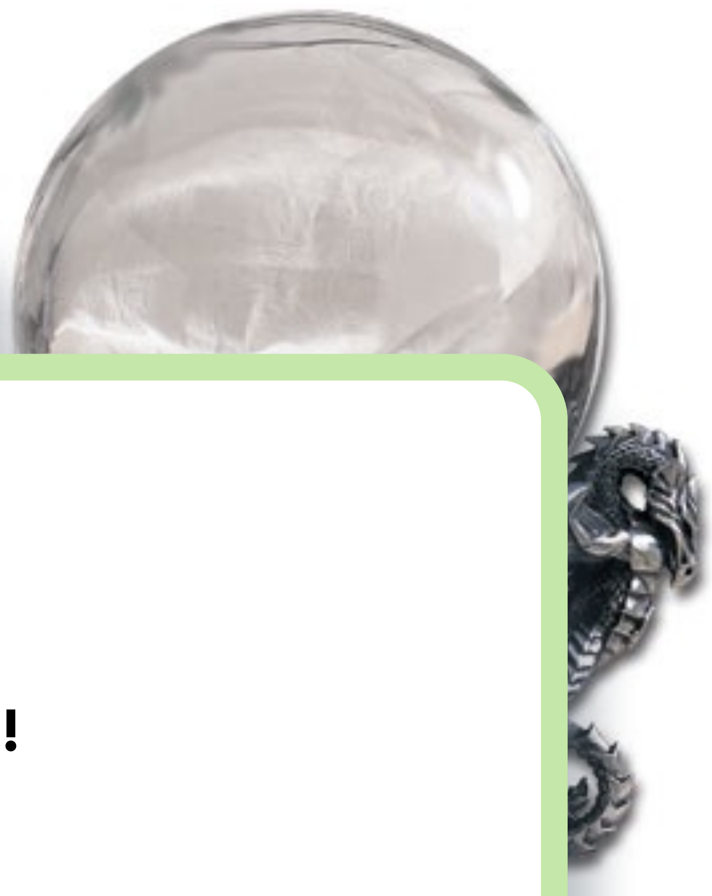
Many ha

But ma

At least

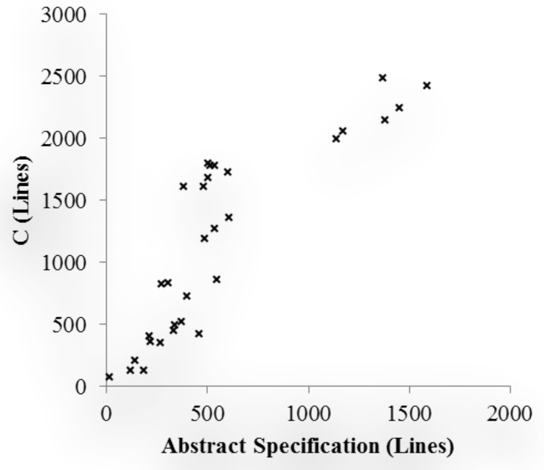
We have large proofs.

Let's crunch some data!



Some Hope

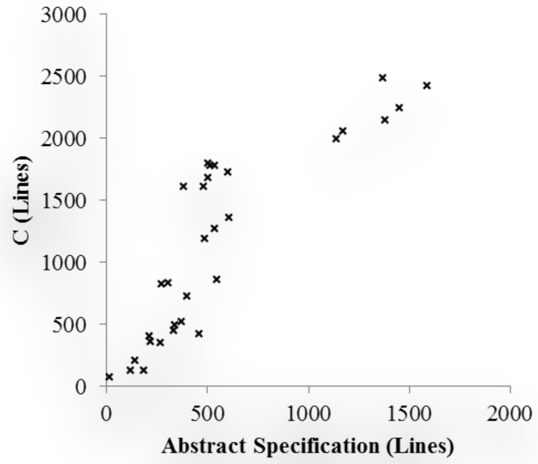
Code Size is correlated with Spec Size

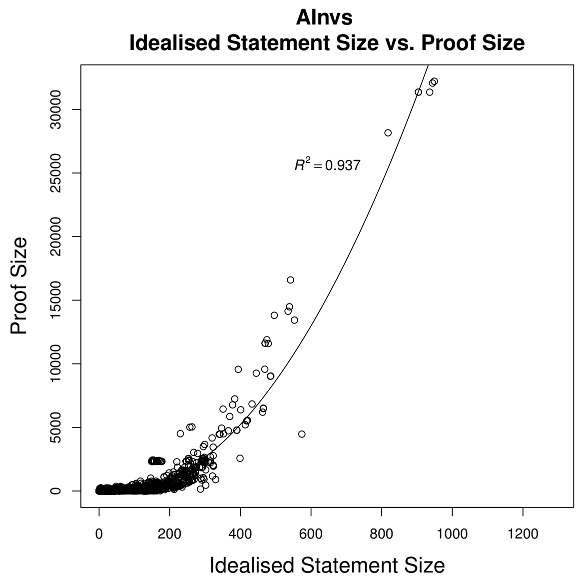


Some Hope

Code Size is correlated with Spec Size

Spec Size is correlated with Proof Size



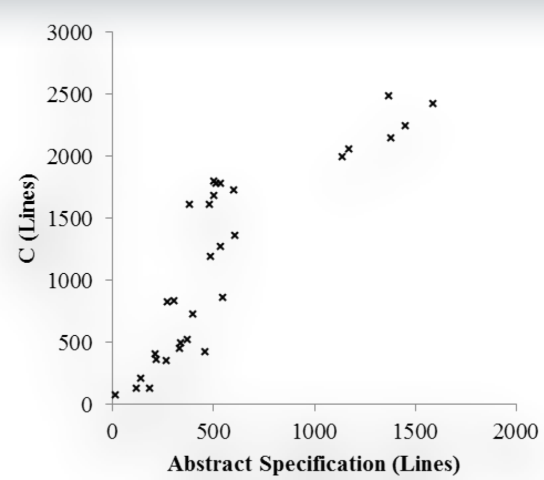


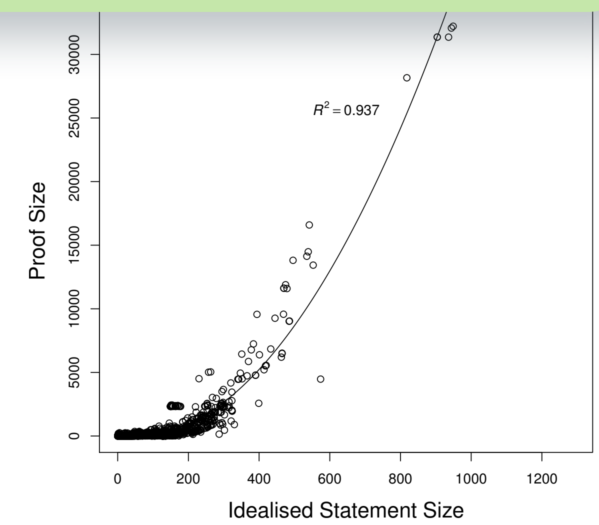
Some Hope

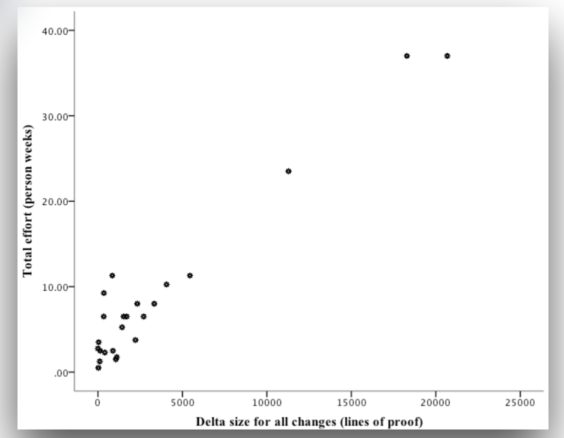
Code Size is correlated with Spec Size

Spec Size is correlated with Proof Size

Proof Size is correlated with Effort







Some Hope

Code Size is correlated with Spec Size

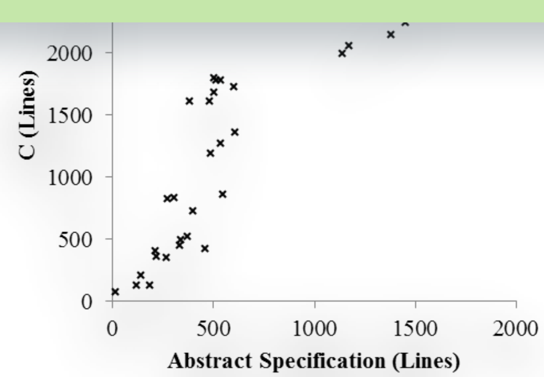
Spec

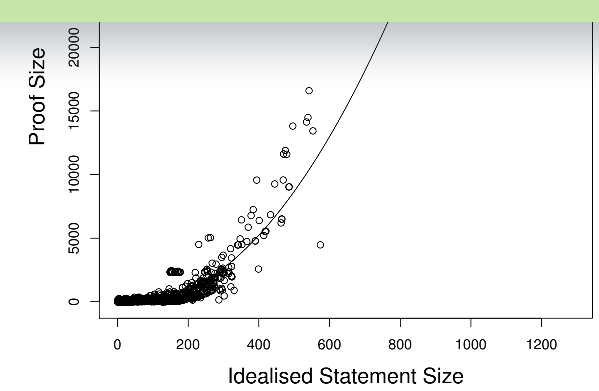
Pro

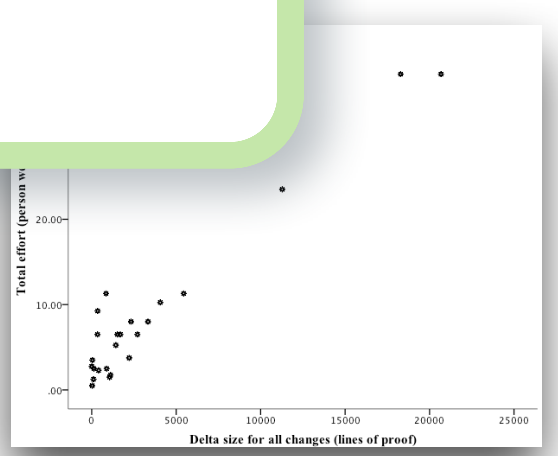
There may be hope for a prediction model.

Probably applies to verification of non-modular code.

Unlikely to work for other kinds of proofs, but likely to transfer to other interactive provers.







The Future

The Future: Integration

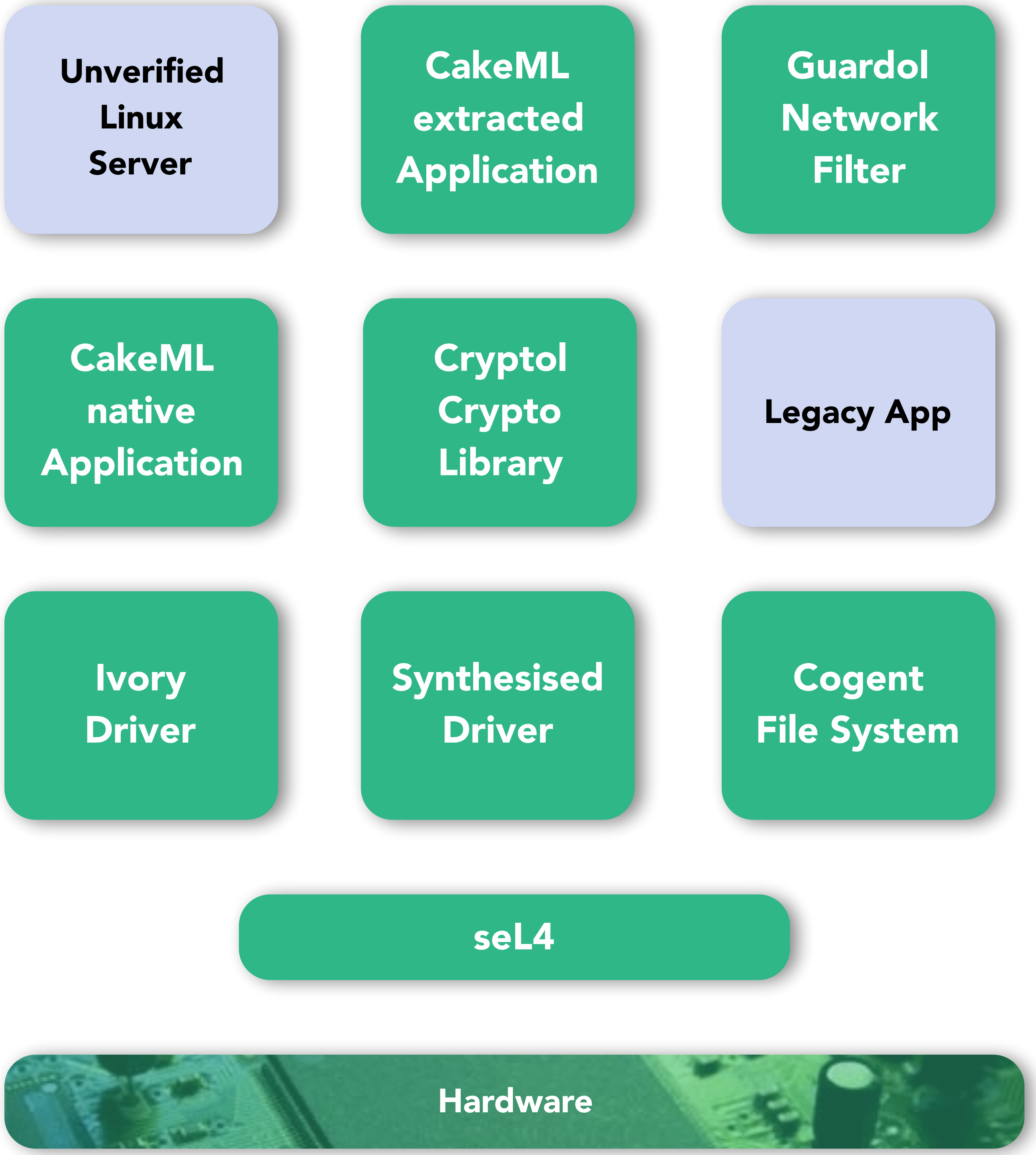
- **No method fits all**

- **Use seL4 isolation!**
  - don't verify all components
  - mix verification approaches

Hardware

The Future: Integration

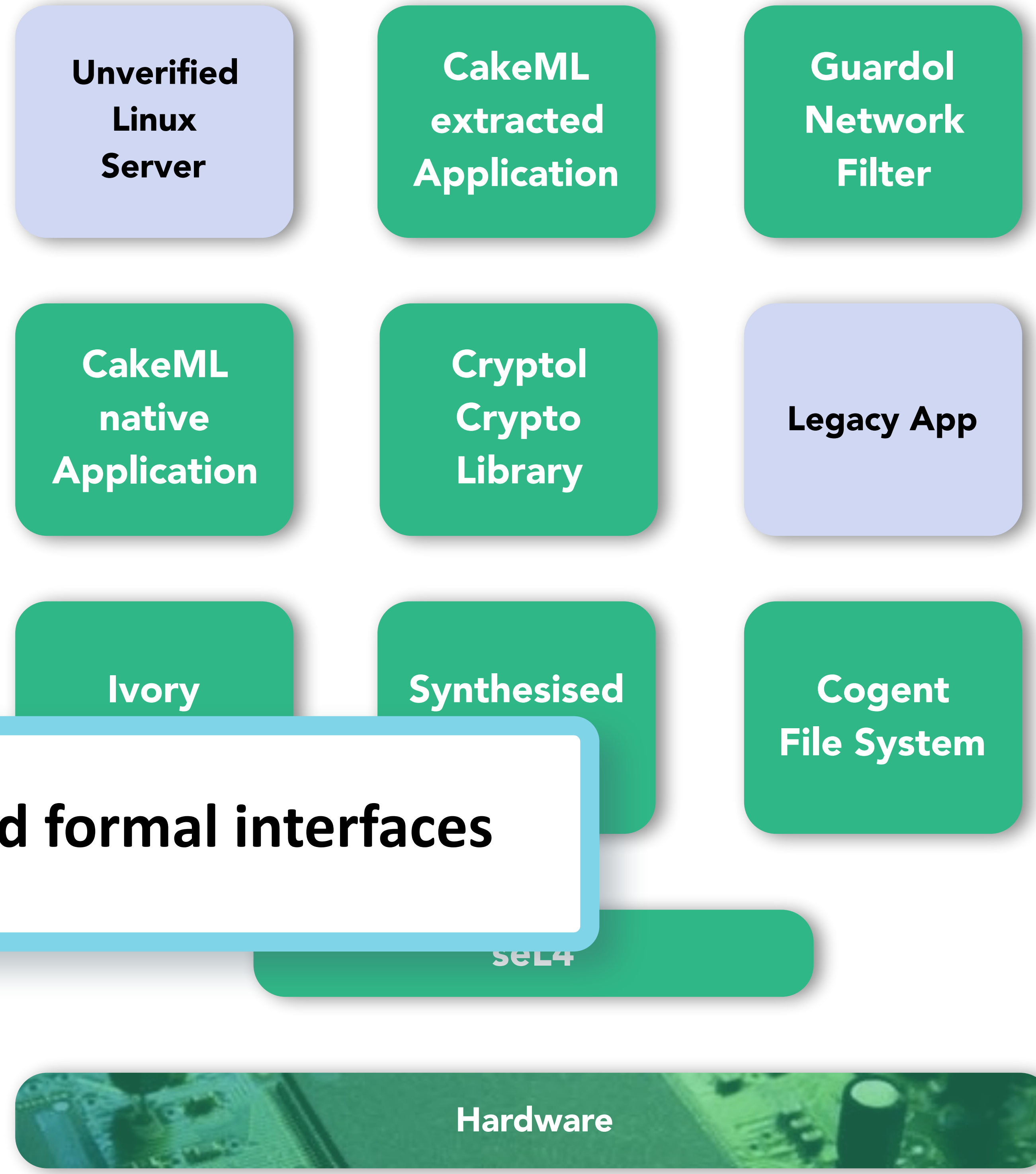
- **No method fits all**
- **Use seL4 isolation!**
  - don’t verify all components
  - mix verification approaches



The Future: Integration

- No method fits all

- Use seL4 isolation!
  - don't verify all components
  - mix verification approaches



Will need formal interfaces

Summary

- **Verification of real systems is happening**

Summary

- Verification of real systems is happening

- It's still too expensive

Summary

- Verification of real systems is happening
- It’s still too expensive
- There is hope

Summary

- Verification of real systems is happening
- It's still too expensive
- There is hope
- Ongoing work on
  - Proof Engineering
  - Languages for verification productivity
  - Increased Automation

Summary

- Verification of real systems is happening
- It's still too expensive
- There is hope
- Ongoing work on
  - Proof Engineering
  - Languages for verification productivity
  - Increased Automation
- Integration will be key

Thank You

Trustworthy Systems
Gerwin Klein

t +61 2 8306 0578
e gerwin.klein@nicta.com.au
w http://trustworthy.systems