

Real World Verification

Pratap Singh

some slides courtesy of Bryan Parno, Xavier Leroy, Tahina Ramananandro



This lecture's material
will NOT be on the final!

You should still pay attention because...

- Industrial/“practical” examples of program verification
- Applications of ideas from class
- Research ideas! Email me 😊

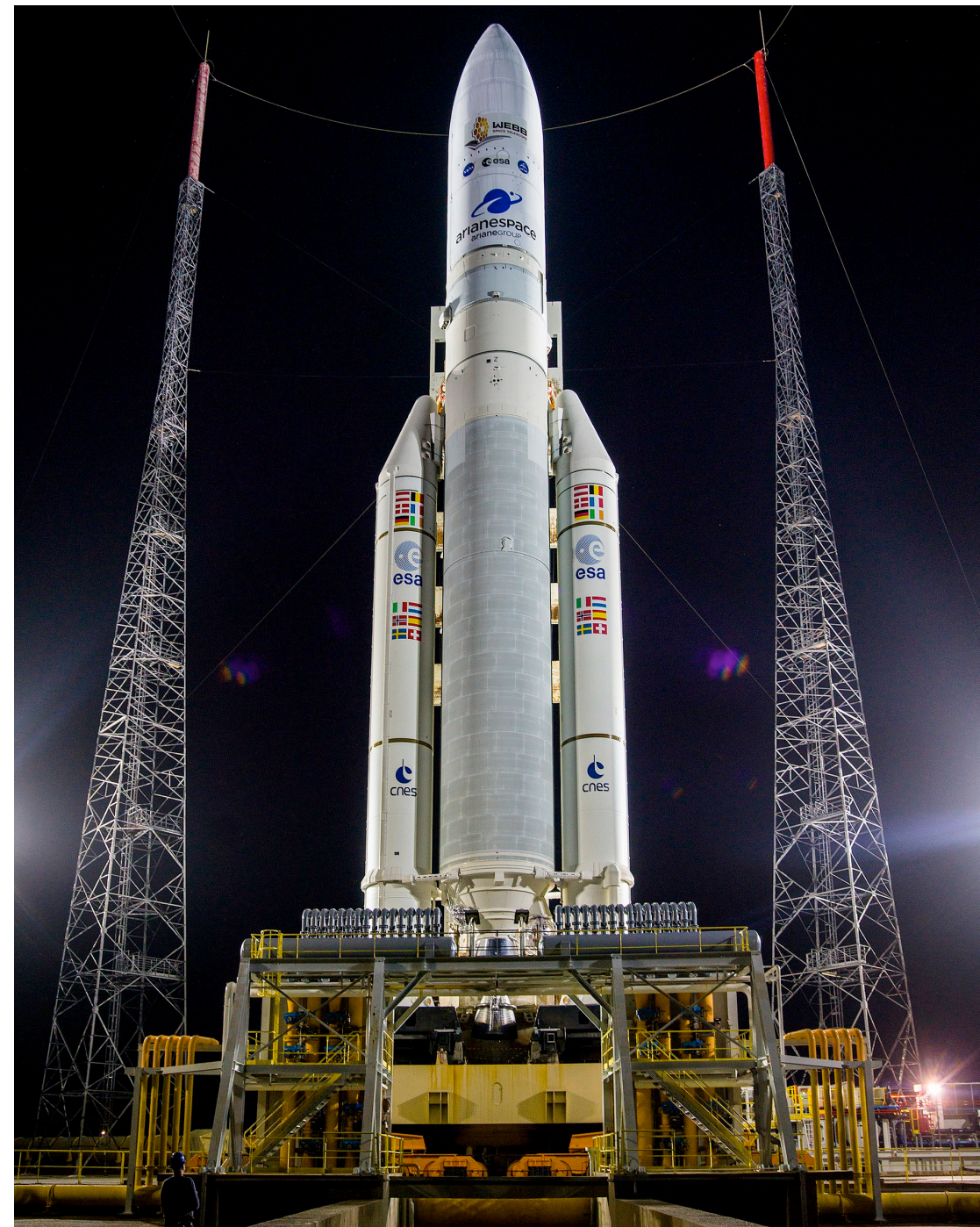
Outline

- Why verify systems software?
- What's hard about verifying systems software?
- Automated program verifiers
- Some projects!
 - Compilers
 - OS kernels
 - Distributed systems
 - Cryptography

Why verify systems software?

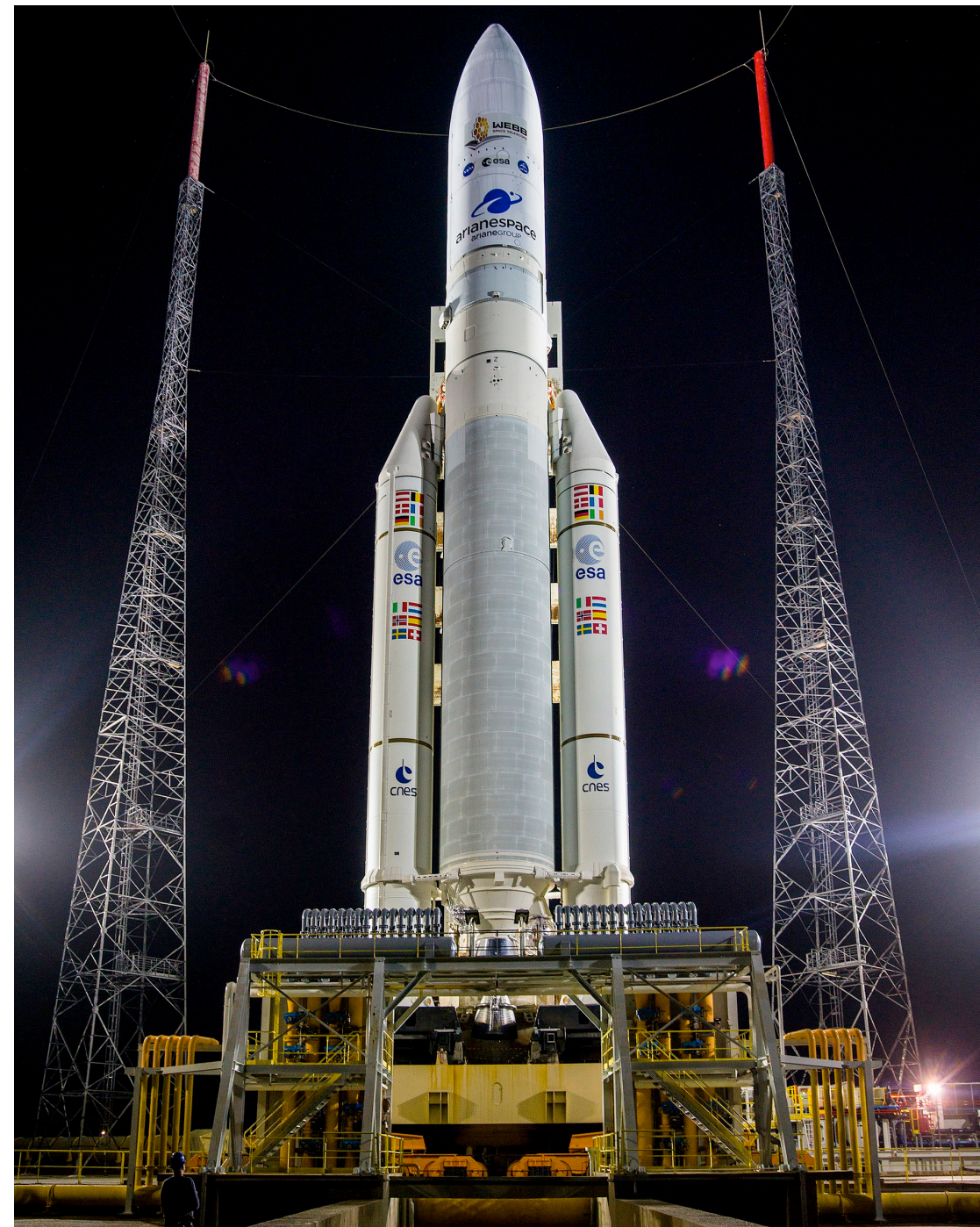
Why verify systems software?

- Ariane V rocket: June 1996




Why verify systems software?

- Ariane V rocket: June 1996



Why verify systems software?

- Ariane V rocket: June 1996



An incorrectly handled software exception resulted from a data conversion of a 64-bit floating point to a 16-bit signed integer value. The value of the floating point number that was converted was larger than what could be represented by a 16-bit integer, resulting in an operand error not anticipated by the Ada code.

Why verify systems software?

- Therac-25 radiation therapy machine

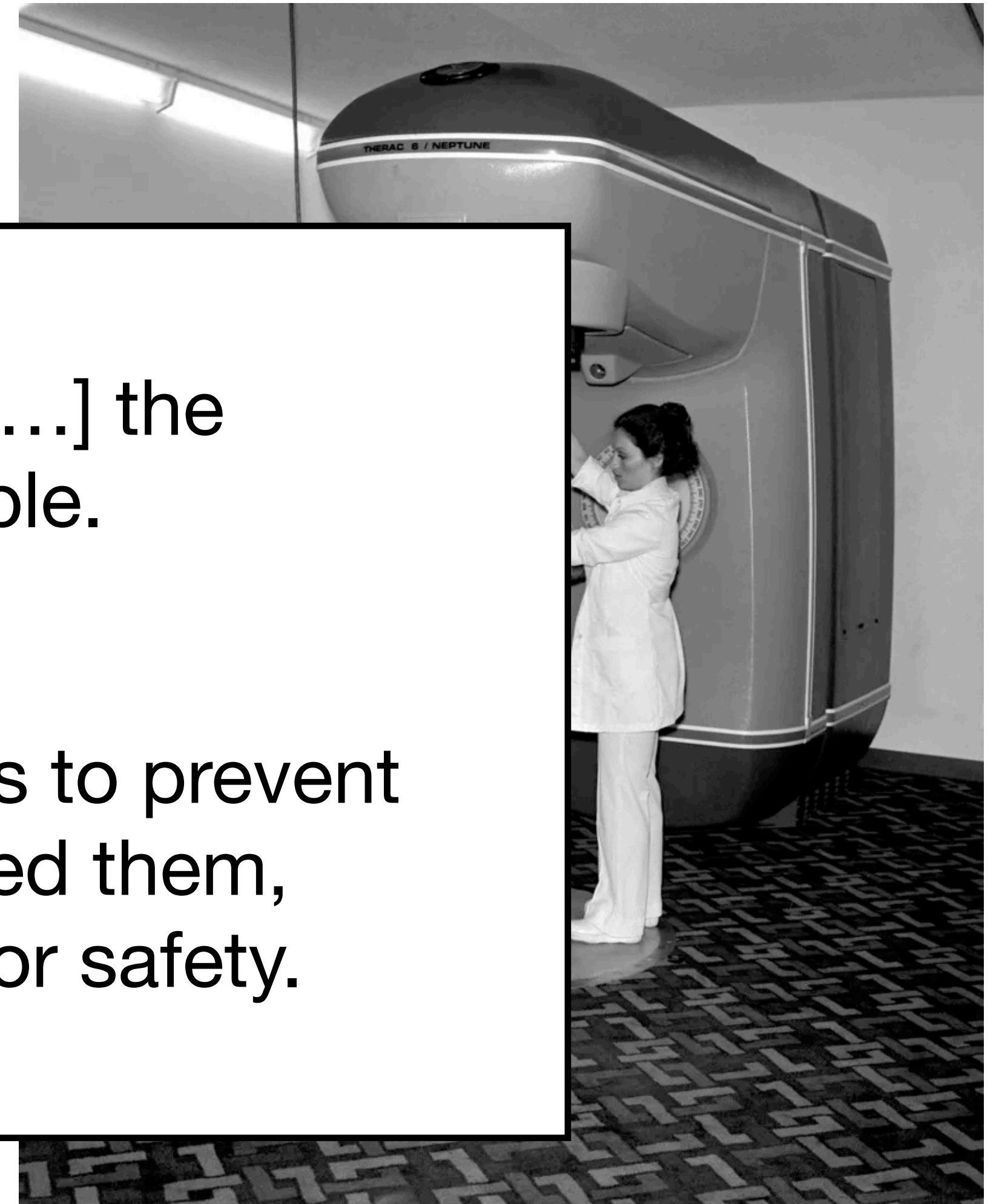


Why verify systems software?

- Therac-25 radiation therapy machine

Between June 1985 and January 1987, [...] the Therac-25 massively overdosed six people.

Previous models had hardware interlocks to prevent [...] faults, but the Therac-25 had removed them, depending instead on software checks for safety.



Why verify systems software?

Buggy code update triggers Bing, Yahoo outage

Jan 5, 2015 9:19 AM
Filed under [Software](#)

[Like](#) 9 [Tweet](#)



Tags

Google blames software bug for Friday night Gmail outage

Promoted Content



SHARE



TWITTER



WRITTEN BY
Caroline
Donnelly

Search giant
users unable

JUST IN With the launch of its Edge browser, Microsoft ends an a

Topic: [Amazon](#)

Follow via:

Amazon Web Services suffers outage, takes down Vine, Instagram, others with it

Summary: The cloud giant suffered an outage for about an hour on Sunday, showing once again the perils of an outsourced cloud service, as many AWS customers went down with it.



By [Zack Whittaker](#) for [Between the Lines](#) | August 26, 2013 -- 13:22 GMT (06:22 PDT)

[Follow @zackwhittaker](#)

10.6K followers

[Get the ZDNet Product Watch newsletter now](#)

Comments

12



Share on Facebook



Tweet

1



Share

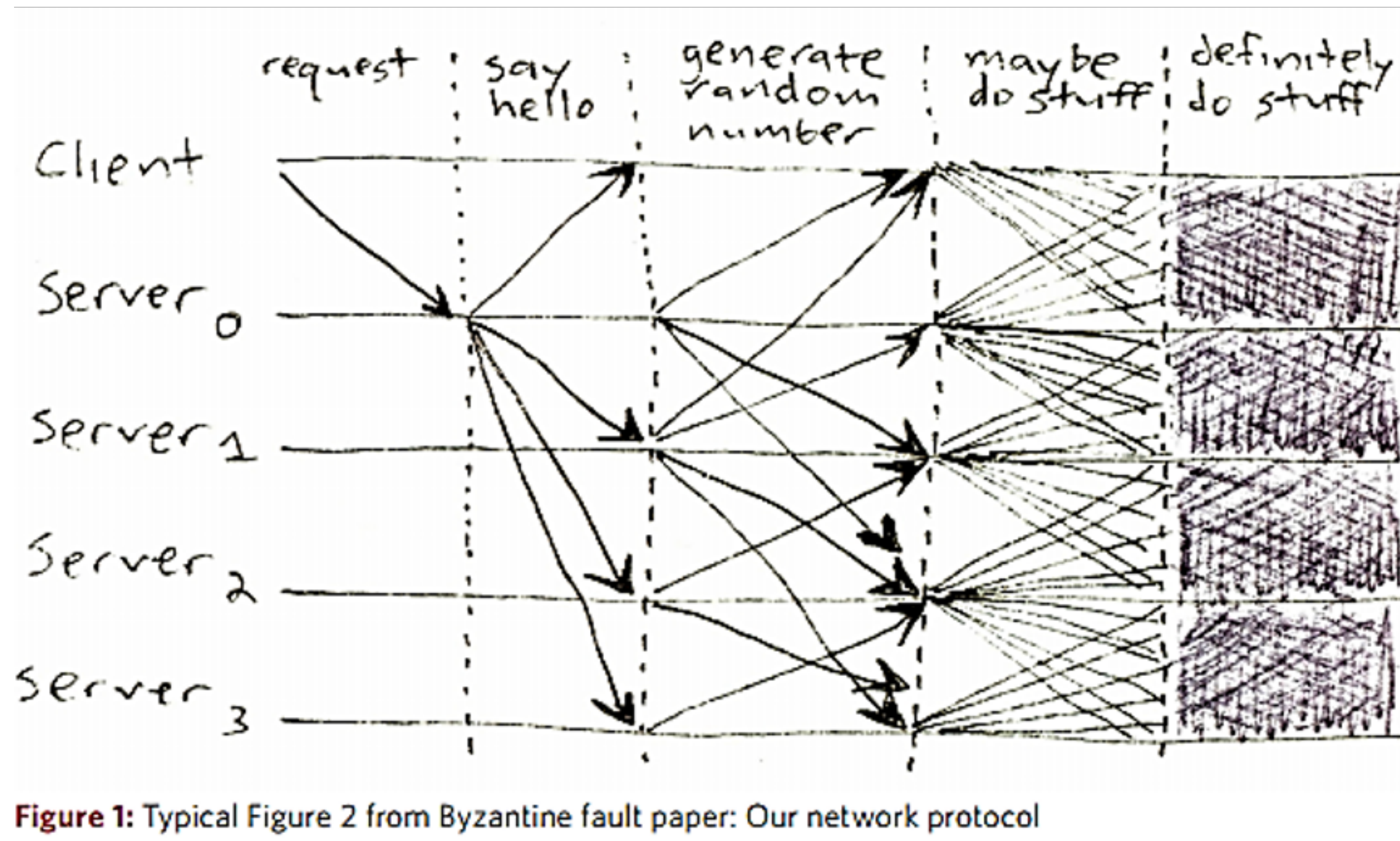
more +

Why verify systems software?

- Not just to stop bugs...Systems are really hard to get right in the first place!

Why verify systems software?

- Not just to stop bugs...Systems are really hard to get right in the first place!



[Mickens 2013]

What's hard about verifying systems software?

Not exhaustive...

- Complexity
- Concurrency
- Performance
- Reasoning about memory/pointers

What's hard about verifying systems software?

Integration Verification across Software and Hardware for a Simple Embedded System

Andres Erbsen*
Samuel Gruetter*
Joonwon Choi
Clark Wood
Adam Chlipala
MIT CSAIL
USA

PLDI 2021

- Simple embedded system: turns a lightbulb on when it receives a particular network packet
- Fully verified hardware (RISC-V processor) and software (compiler, network/lightbulb driver)

Table 4. Lines of code

Excluded:	implementation m	interface n	interesting proof p	low-insight proof q	proof overhead $(m + n + p + q) / m$
unrelated					
library	10044				
imports	7301				
doc	1907				
Kami	354				
	48294				
lightbulb app	176	130	33	1443	10.1
program logic	0	208	552	1785	—
compiler	931	1114	1325	6654	10.8
SW/HW interface	0	2053	991	3804	—
end-to-end	0	254	74	539	—

“around four person-years of work”



Automated program verifiers

Cayden will talk about interactive
theorem proving on Thursday!

Dafny



- Developed at Microsoft Research by Rustan Leino (2009-present)
- Imperative and functional features, inspired by C#
- Compiles to C#, Java, JS, Python,...
- Recent significant investment from Amazon



Dafny vs Why3

```
mystery2.dfy > g
1
2 method g(n: int) returns (result: int) |
3   requires n >= 0
4   ensures result >= 0
5     && result * result <= n
6     && n < (result + 1) * (result + 1)
7 {
8   var a := 0;
9   var b := 0;
10  var c := 0;
11  while b <= n
12  ...
13    invariant 0 <= a
14    invariant b == a * a
15    invariant c == 2 * a
16    invariant a == 0 || (a-1) * (a-1) <= n
17  {
18    b := b + c + 1;
19    c := c + 2;
20    a := a + 1;
21  }
22  assert b > n;
23  result := a - 1;
24 }
```

```
mystery2.mlw
1 module Mystery2
2
3 use int.Int
4
5 let g (n : int) : int =
6   requires { n >= 0 }
7   ensures { result >= 0
8     /\ result * result <= n
9     /\ n < (result+1)*(result+1) }
10  let ref a = 0 in
11  let ref b = 0 in
12  let ref c = 0 in
13  while b <= n do
14    invariant { 0 <= a }
15    invariant { b = a * a }
16    invariant { c = 2 * a }
17    invariant { a = 0 /\ (a-1) * (a-1) <= n }
18    variant { n - b }
19    b <- b + c + 1 ;
20    c <- c + 2 ;
21    a <- a + 1
22  done ;
23  assert { b > n };
24  a-1
```

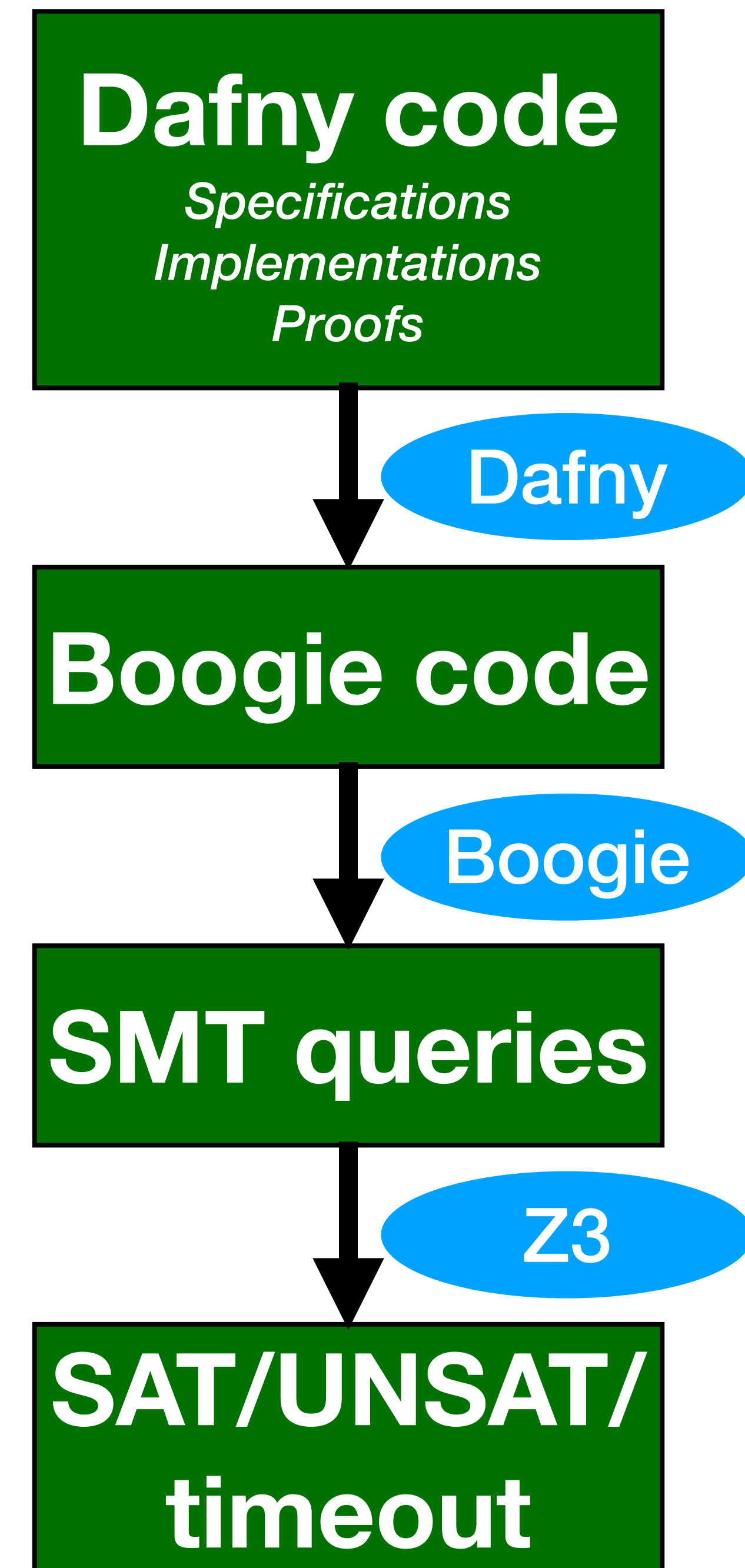
Dafny under the hood

**Boogie: An Intermediate
Verification Language**

Established: December 10, 2008

used by other tools
(VCC, Chalice, Spec#...)

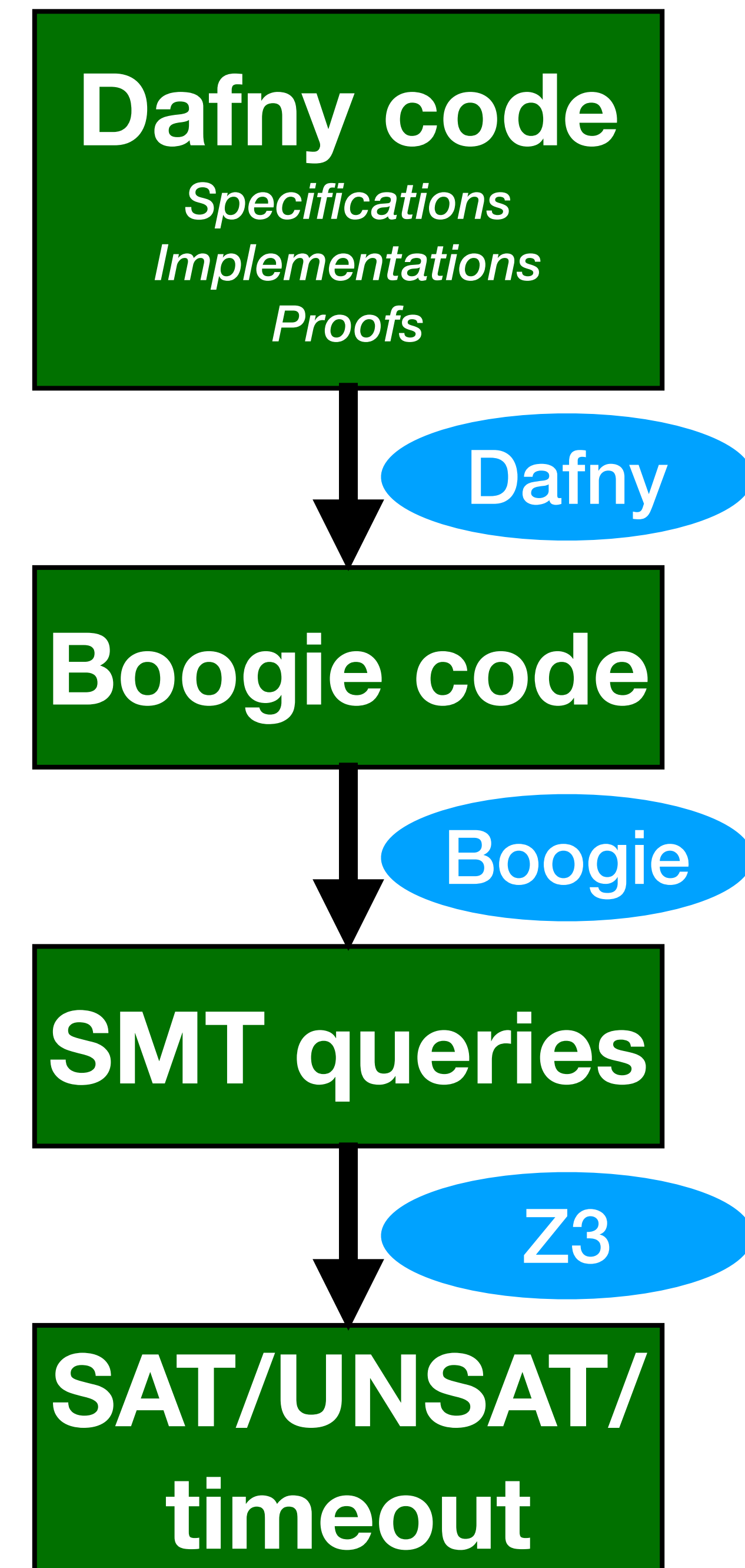
Z3



Dafny under the hood

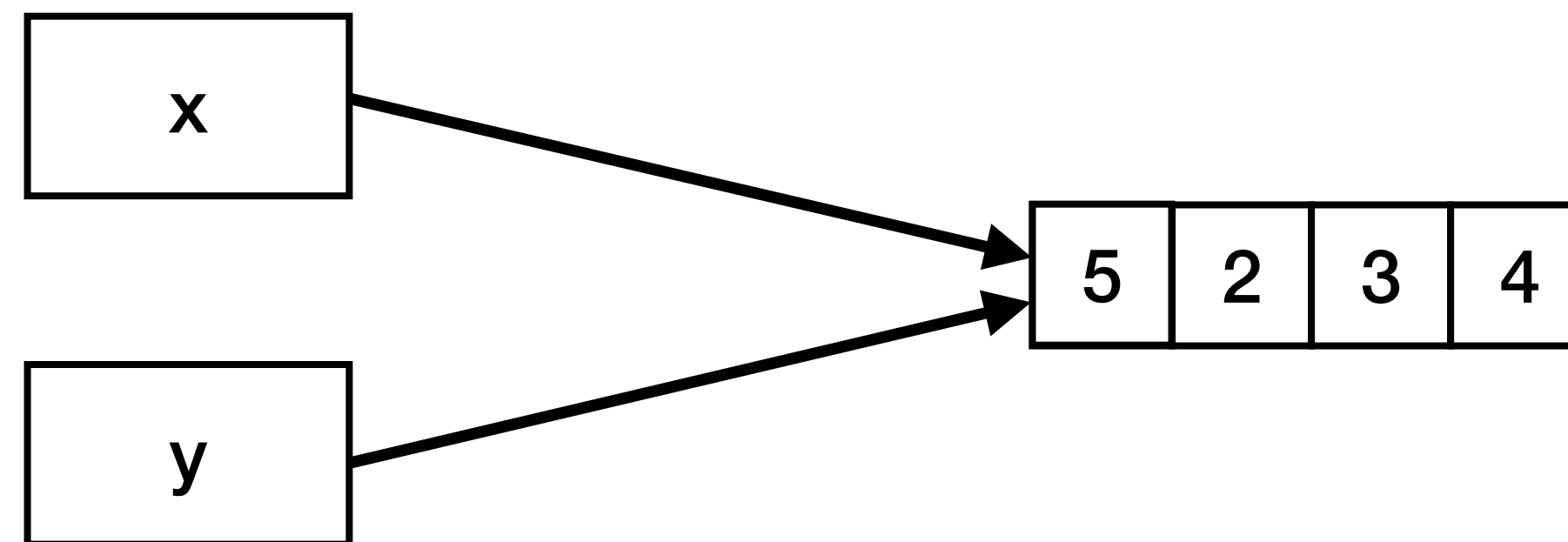
Compared to Why3, Dafny...

- is specialized to one prover backend
- can adapt to this backend's properties
- (arguably) produces “better” SMT queries
- provides less visibility into verification
 - (no “Task view”)



Memory reasoning

- Pointers are hard because of **aliasing**:



```
x[0] := 5;  
assert y[0] == 1; 🙄
```

- APV tools verify one function body at a time
- But all functions share the **same global mutable heap**
- Why3's solution: look at the function body to see what memory it modifies
 - Doesn't scale to large functions/programs!

Memory reasoning in Dafny: Framing

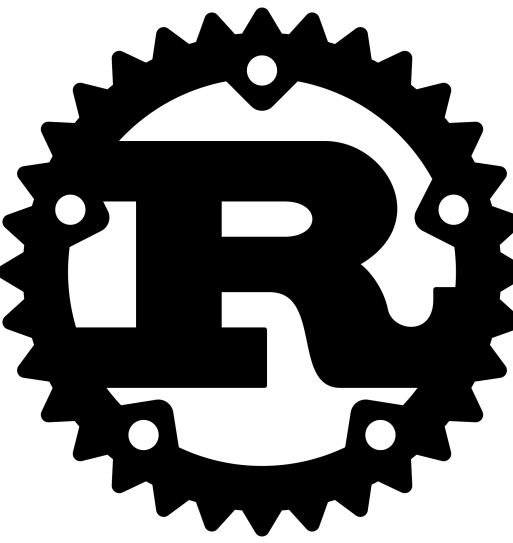
- Each method specifies which mutable memory it **modifies**
- Method callers know that other memory remains unchanged

```
method array_update(a: array<int>, b: array<int>)  
  requires a.Length > 0 && b.Length > 0  
  modifies a  
{  
  a[0] := 3;  
}
```

assignment might update an array element not in the enclosing context's modifies clause Verifier

```
b[0] := 5;  
array_update(a, b);  
assert b[0] == 5; ✓
```

Rust verification



- Rust promises **safety at zero cost**
 - Low-level memory control, without the footguns of C
- Rust's type system **prohibits aliasing!***
- Several verification tools built on top of Rust:

Creusot RustHorn
Prusti RustBelt Verus
Aeneas

**specifically, it only allows aliasing when the pointers are read-only*

Verus

Verus: Verifying Rust Programs using Linear Ghost Types

ANDREA LATTUADA*, VMware Research, Switzerland

TRAVIS HANCE, Carnegie Mellon University, USA

CHANHEE CHO, Carnegie Mellon University, USA

MATTHIAS BRUN, ETH Zurich, Switzerland

ISITHA SUBASINGHE[†], UNSW Sydney, Australia

YI ZHOU, Carnegie Mellon University, USA

JON HOWELL, VMware Research, USA

BRYAN PARNO, Carnegie Mellon University, USA

CHRIS HAWBLITZEL, Microsoft Research, USA

OOPSLA 2023

```
fn update_array(a: &mut [u8], b: &[u8])
  requires a.len() > 0 && b.len() > 0
{
  a[0] = 3;
  b[0] = 3;
}
```

← error[E0596]: cannot borrow `*b` as mutable, as it is behind a `&` reference

This is a Rust compiler error—no SMT query required!

<https://play.verus-lang.org>

Verified real-world systems

The software reliability landscape

Impact of bugs

Software kind

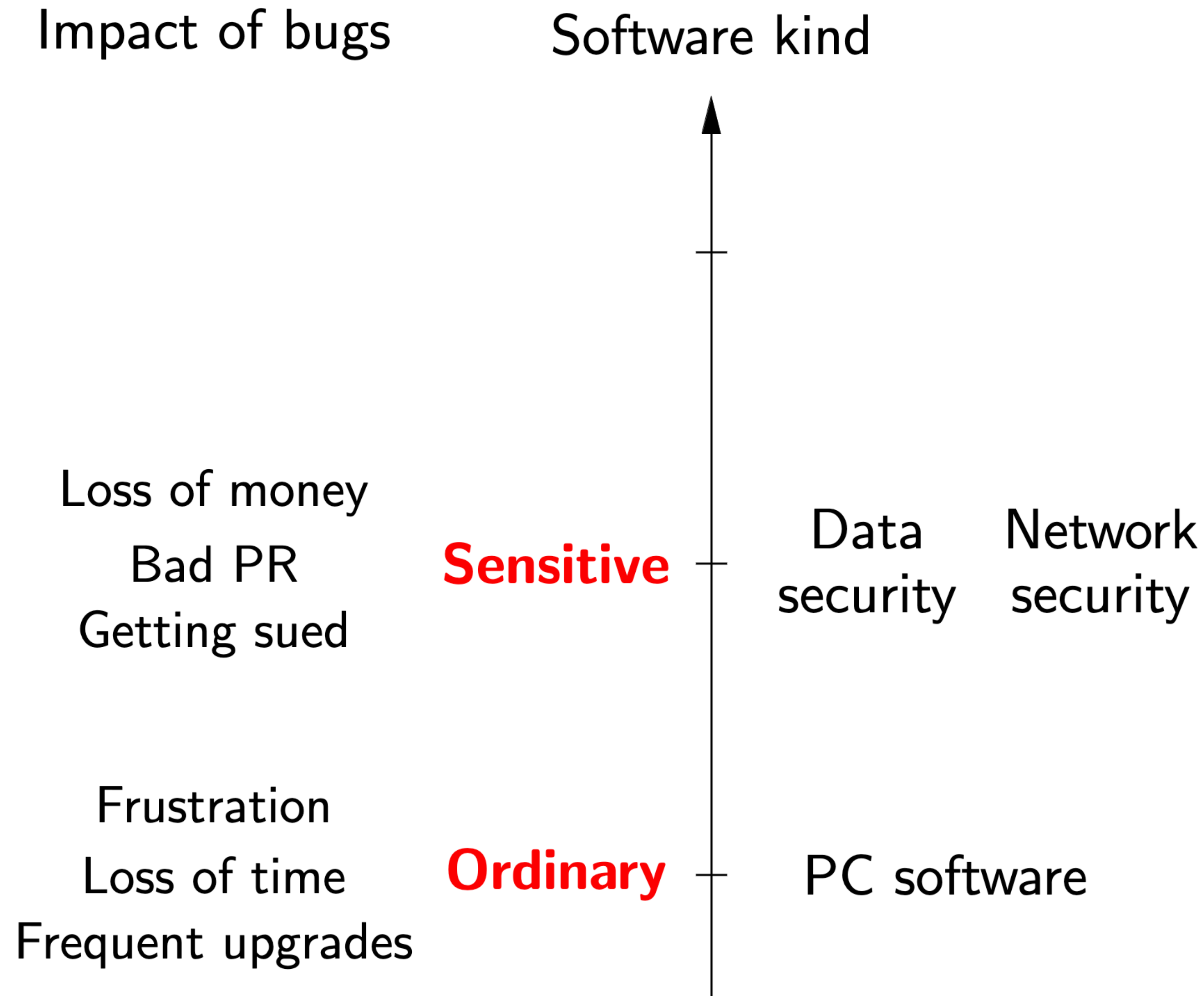
Frustration
Loss of time
Frequent upgrades

Ordinary

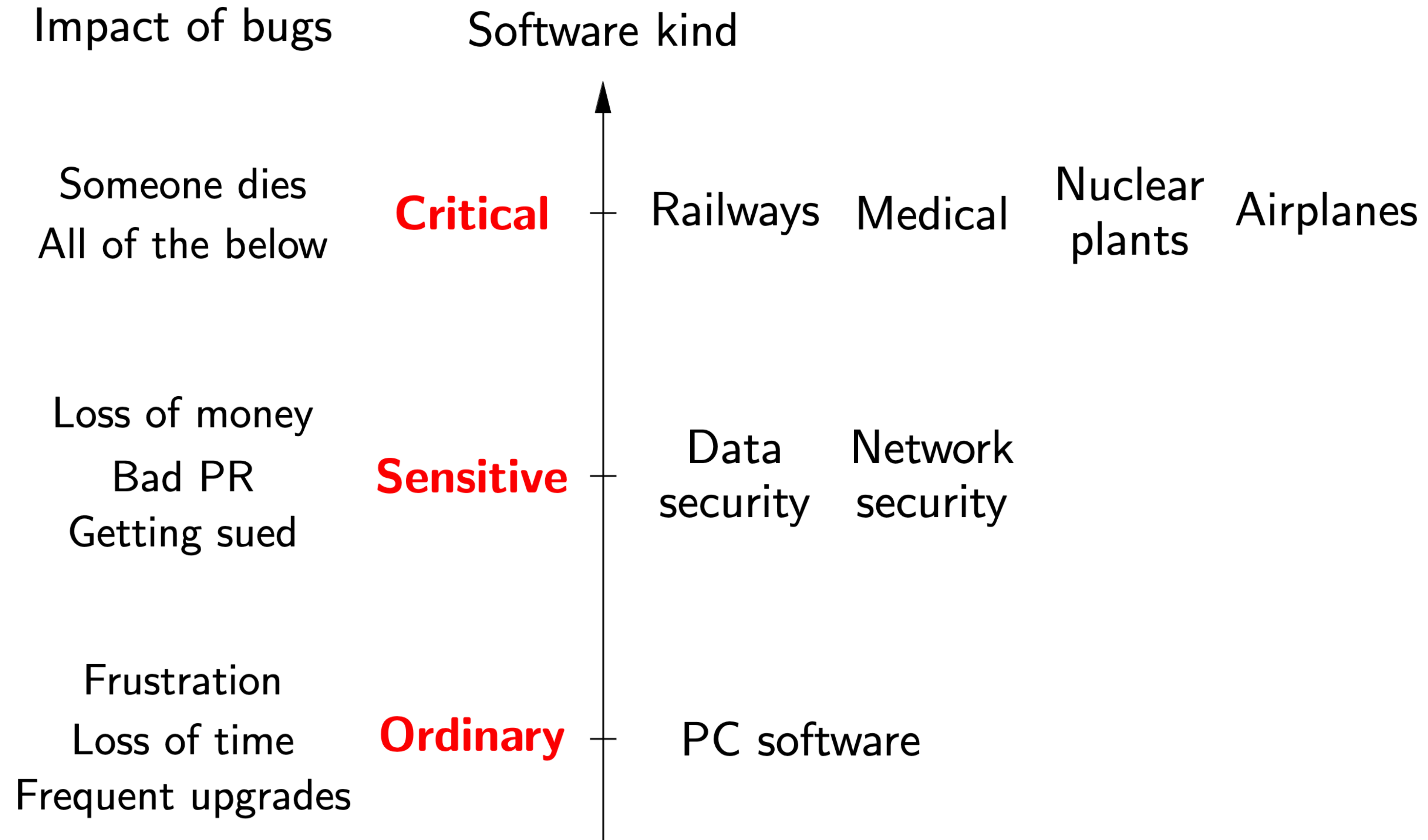
PC software



The software reliability landscape



The software reliability landscape



Verified compilers

- Every program goes through the compiler!
- Miscompilation bugs are rare but very costly
- Compiler verification is not new

John McCarthy
James Painter¹

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS²

1. Introduction. This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Mathematical Aspects of Computer Science, 1967

Verified compilers: CompCert

Formal Verification of a Realistic Compiler

By Xavier Leroy

CACM 2004



- Formally verified compiler for C, targeting ARM, x86, RISC-V, ...
- Includes complex compiler optimizations
- 50,000 lines in the Coq proof assistant



Verified compilers: CompCert

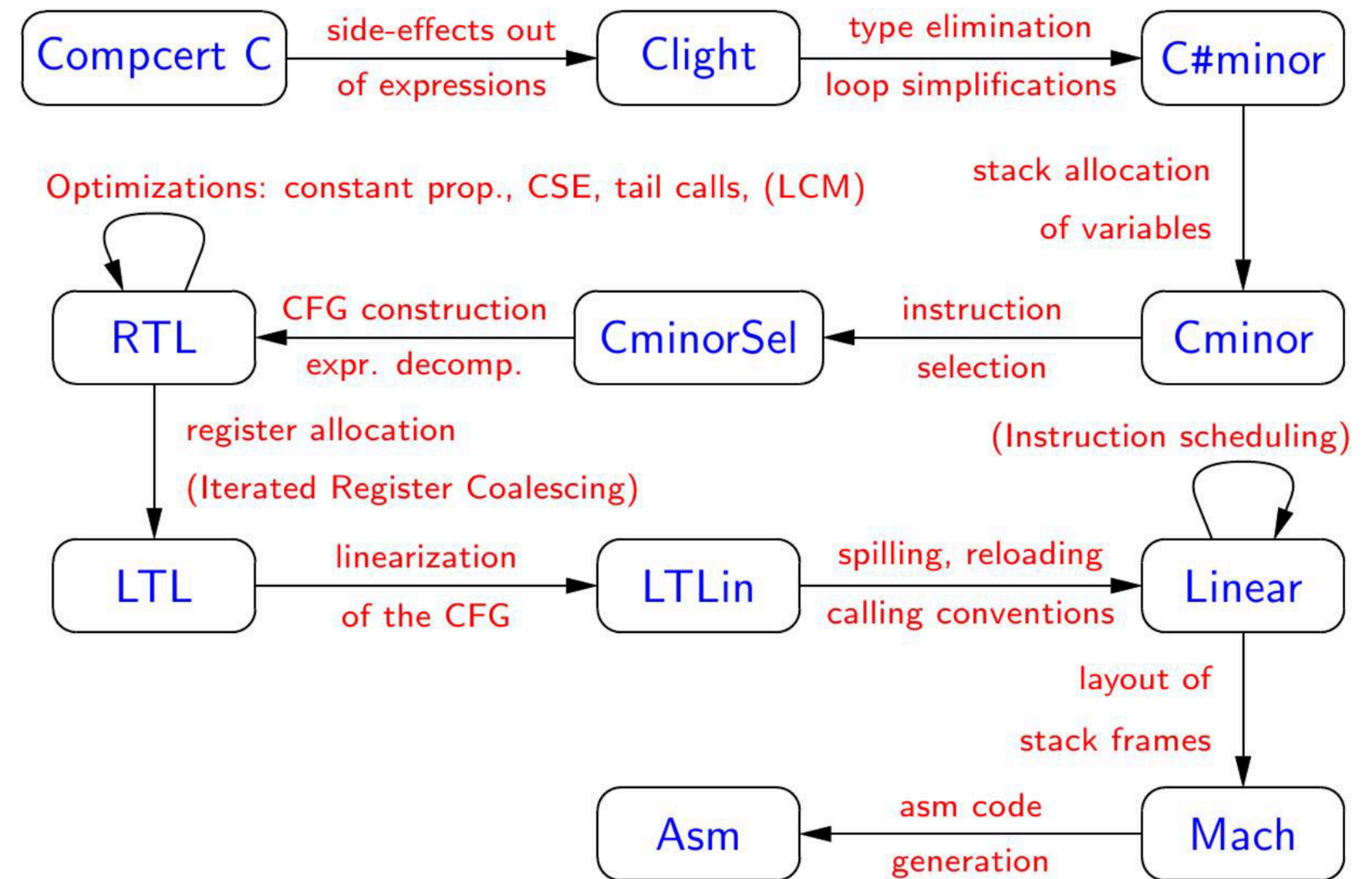
Theorem (Semantic preservation)

*For all source code S ,
if the compiler generates machine code C from source S without reporting
any compilation error,
then C behaves like S .*

- “Behaves like” requires mechanized semantics for C and assembly

Verified compilers: CompCert

- Many optimization/lowering passes
- Easier to verify many small transformations



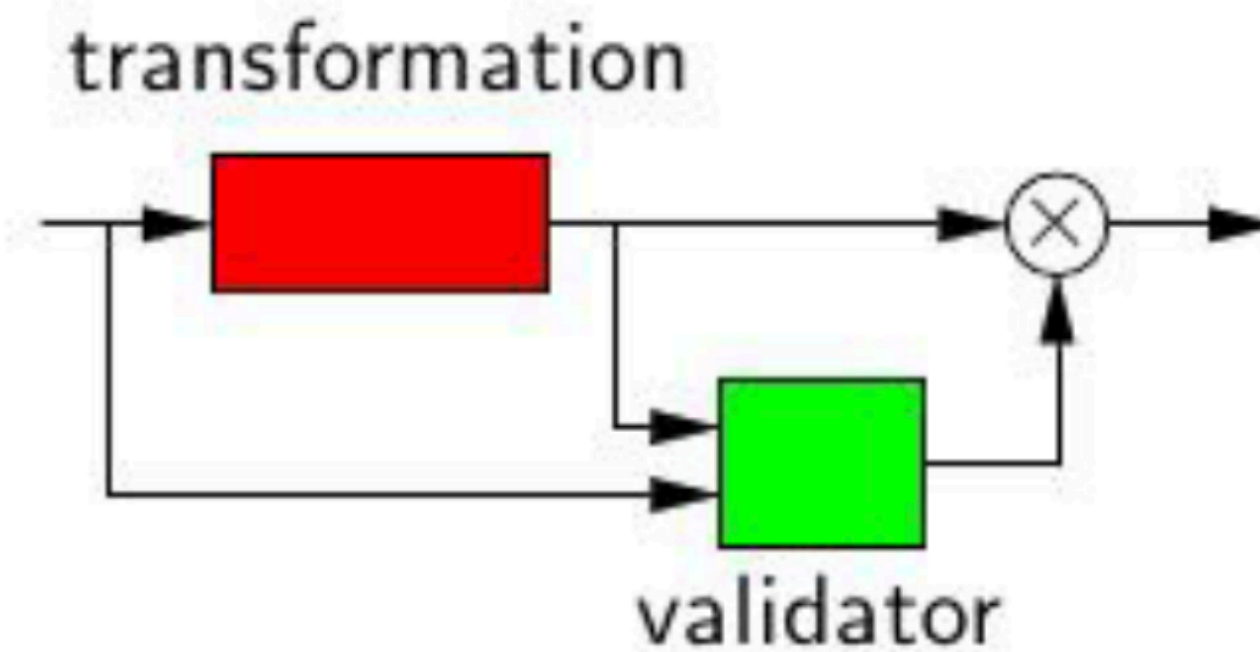
Verified compilers: CompCert

- Some passes (e.g., register allocation) are just too hard to prove
- Solution: **translation validation**

Verified transformation



Verified translation validation



Verified compilers: CompCert

- Used in industry!

Who uses CompCert?



The Institute of Flight System Dynamics at the Technical University of Munich uses CompCert in the [development of flight control and navigation algorithms](#).



In 2017, CompCert was successfully qualified by MTU Friedrichshafen according to IEC 60880, category A, and IEC 61508-3:2010, SCL 3 for a [certification project in the nuclear energy domain](#). The use of CompCert reduced development time and costs.



Airbus France is deploying CompCert at the Toulouse plant in a number of currently undisclosed projects.



In the civil-aviation research project [QSMA](#) by the German Federal Ministry for Economic Affairs and Energy, CompCert is being used to develop a TSO-C151b Terrain Avoidance and Warning System in accordance with DAL-C. The project is carried out by emmtrix, the German Aerospace Center DLR, Validas, TU Clausthal, and AbsInt.



- Formally verified operating system microkernel
- Proved using Isabelle/HOL, an interactive theorem prover
- Properties:
 - **Integrity**: data is only changed as requested
 - **Confidentiality**: data can only be read with appropriate permission



seL4



Today, seL4 is part of an ecosystem supporting active use in various domains including **automotive, aviation, infrastructure, medical, and defence**. A key highlight demonstrating its fit for real-world deployment was in the DARPA-funded HACMS program, where seL4 was used to protect an autonomous helicopter against cyber-attacks.

- Distributed systems are very difficult to design and implement:
 - Protocols must handle concurrent execution on separate machines
 - Implementation concerns can break protocol assumptions
- Hard to uncover race conditions with simple testing
- Can verification help?

IronFleet

IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

Microsoft Research

SOSP 2015

Proofs are
machine-
checked
in Dafny

“We show how to build complex, efficient distributed systems whose implementations are provably safe and live.”

Implementations are correct,
not just abstract protocols

The system does not crash
or otherwise go wrong

The system will make progress
given sufficient time
(no livelock or deadlock)

- Methodology:
 - Specify the whole protocol as a **global state machine**
 - Prove that **individual role specs obey the global state machine**
 - Prove that **role implementations obey their specifications**
- All implemented in Dafny

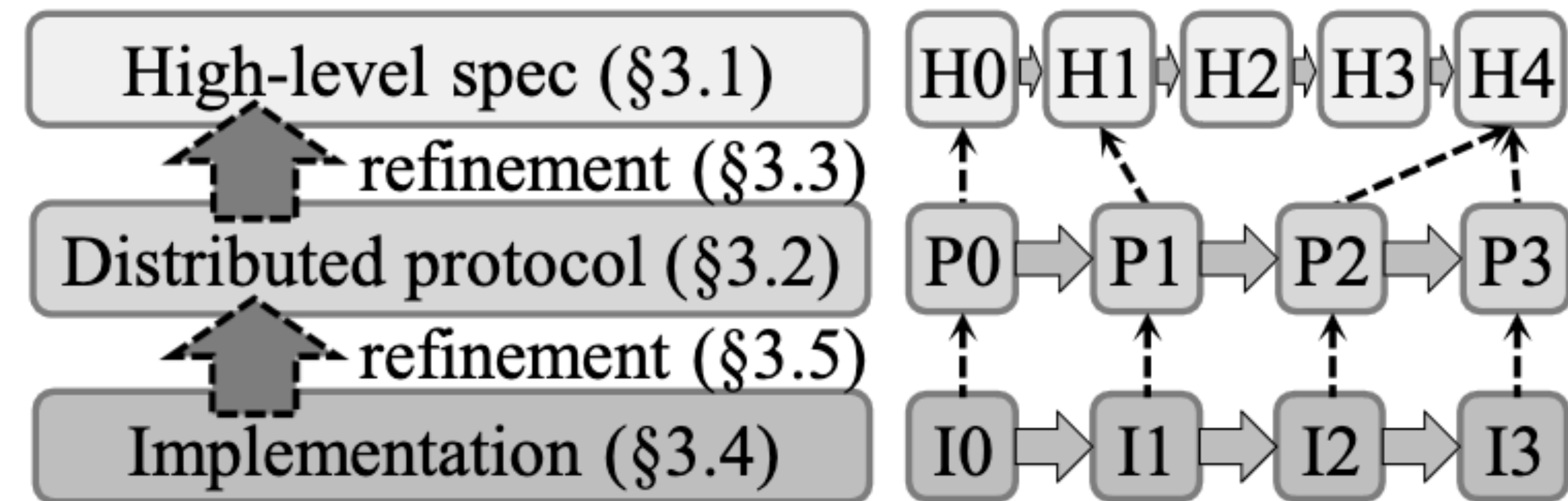


Figure 3. Verification Overview.

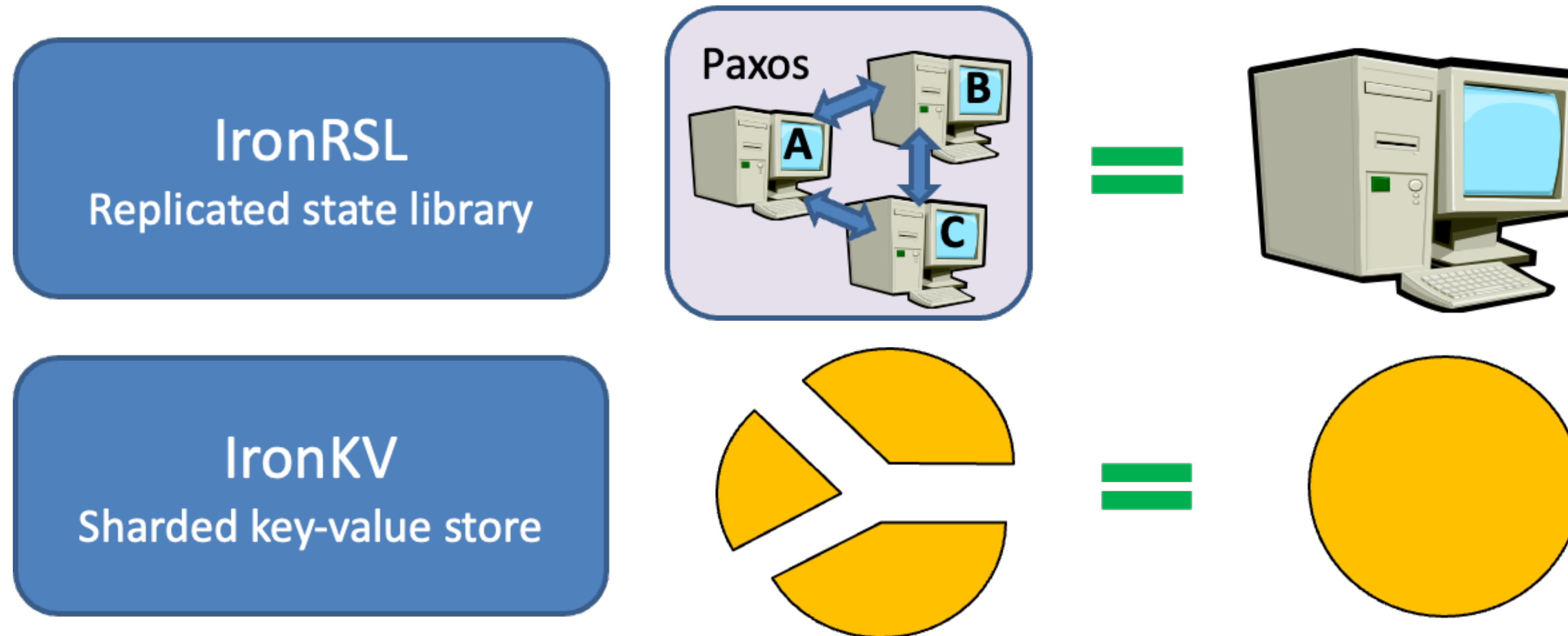
IronFleet

IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

Microsoft Research

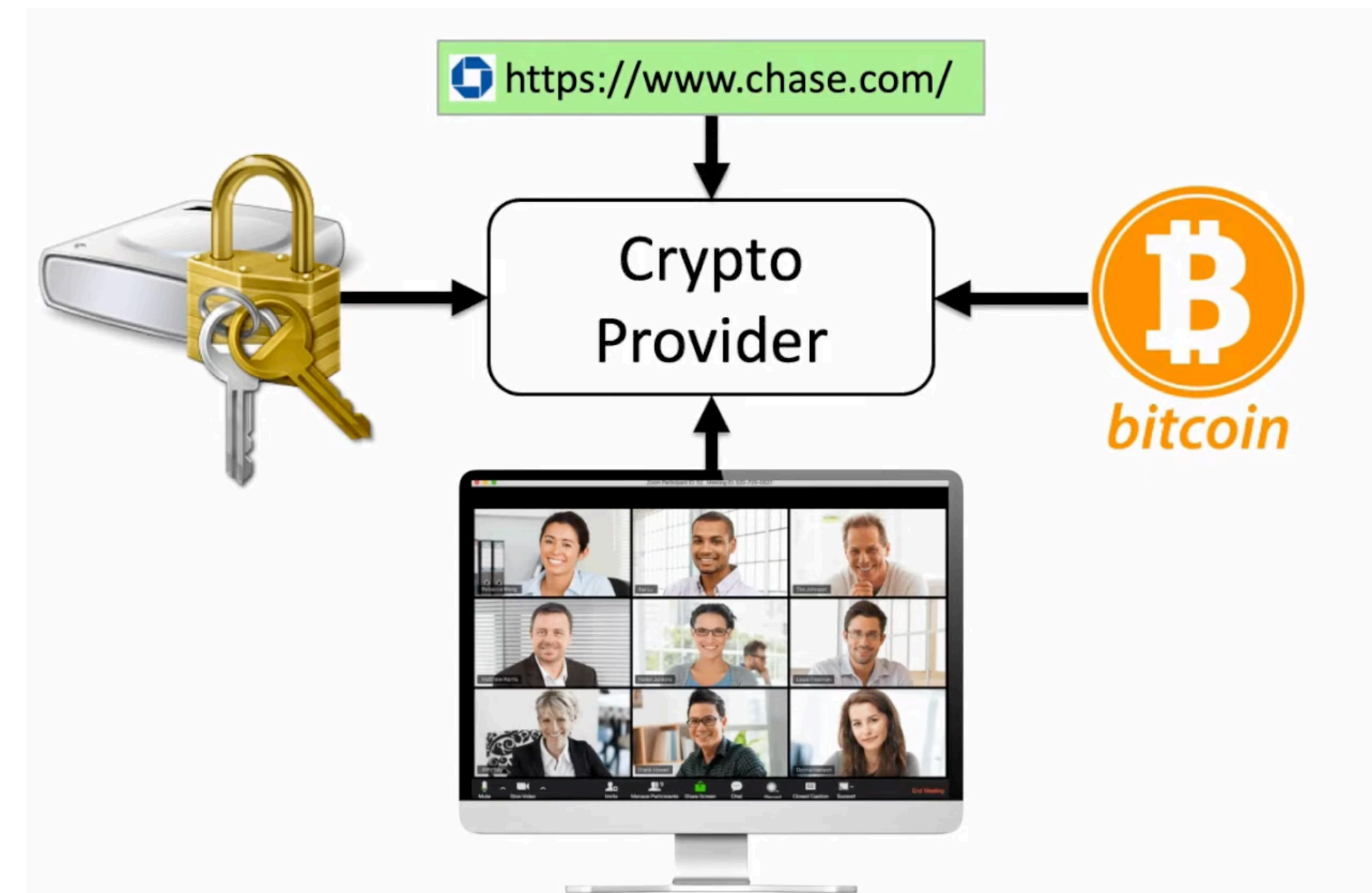
SOSP 2015



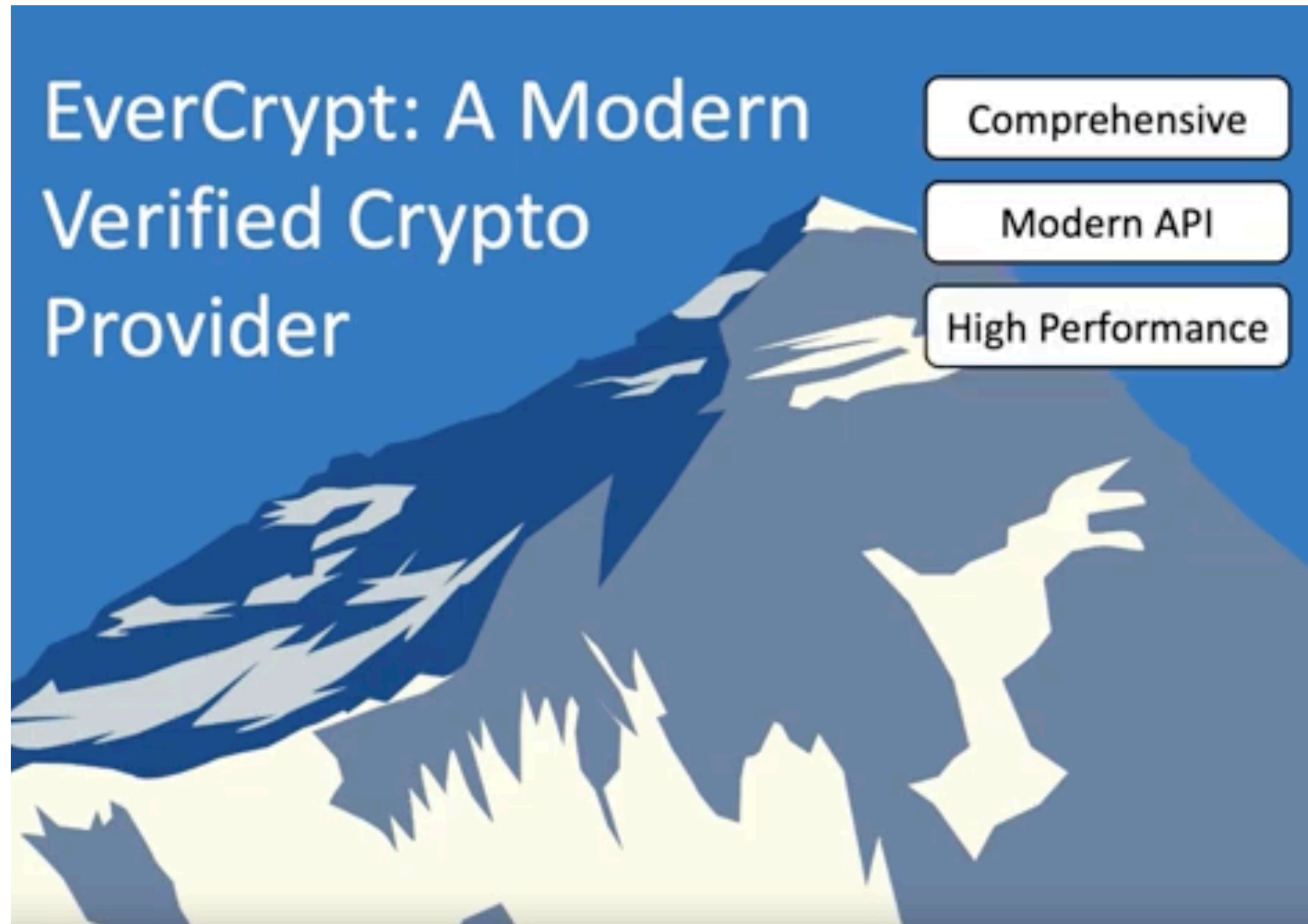
- ~50,000 lines of Dafny, 3.7 person-years of work
- Performance competitive with unverified versions!

Cryptography

- Cryptography is complicated
- Cryptography needs to be fast (<1 cycle per byte for modern encryption)
- Combination of C and hand-optimized assembly
- Bugs in cryptography libraries are extremely bad!



Cryptography: EverCrypt




EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

Jonathan Protzenko*, Bryan Parno[‡], Aymeric Fromherz[‡], Chris Hawblitzel*, Marina Polubelova[†], Karthikeyan Bhargavan[†]
Benjamin Beurdouche[†], Joonwon Choi^{*§}, Antoine Delignat-Lavaud*, Cédric Fournet*, Natalia Kulatova[†],
Tahina Ramananandro*, Aseem Rastogi*, Nikhil Swamy*, Christoph M. Wintersteiger*, Santiago Zanella-Beguelin*
*Microsoft Research [‡]Carnegie Mellon University [†]Inria [§]MIT S&P 2020

- **Symmetric-key ciphers** (AES, Chacha-Poly)
- **Hashes** (SHA, Blake2)
- **MACs** (HMAC, Poly1305)
- **KDFs** (HKDF)
- **Elliptic curves** (curve25519, ed25519, p256)
- ...

Cryptography: EverCrypt

- 124K lines of verified code and proofs in F^*
- (Sometimes) faster than unverified code!

 **CyLab** Carnegie Mellon University
Security and Privacy Institute

Research Education Partnerships News Events About Directory More ▾

Subscribe ✉ 🔍

HOME / NEWS / 2020 /

Provably-secure code incorporated into Linux kernel

Daniel Tkacik

APR 29, 2020

Mozilla Security Blog

SEARCH 🔍

SECURITY

Performance Improvements via Formally-Verified Cryptography in Firefox

Kevin Jacobs and Benjamin Beurdouche | July 6, 2020

Cryptography: Fiat

Simple High-Level Code For Cryptographic Arithmetic – With Proofs, Without Compromises

Andres Erbsen Jade Philipoom Jason Gross Robert Sloan Adam Chlipala

MIT CSAIL,

Cambridge, MA, USA

{andreser, jadep, jgross}@mit.edu, rob.sloan@alum.mit.edu, adamc@csail.mit.edu

S&P 2019

- Elliptic-curve crypto depends on **arithmetic over finite fields**
 - Simple in math, gnarly in code!
- Idea: **automatically search for programs** to implement finite-field arithmetic, along with a **proof of correctness**
- Relies on automatic proof search and type theory of Coq

Automated cryptocode generator is helping secure the web

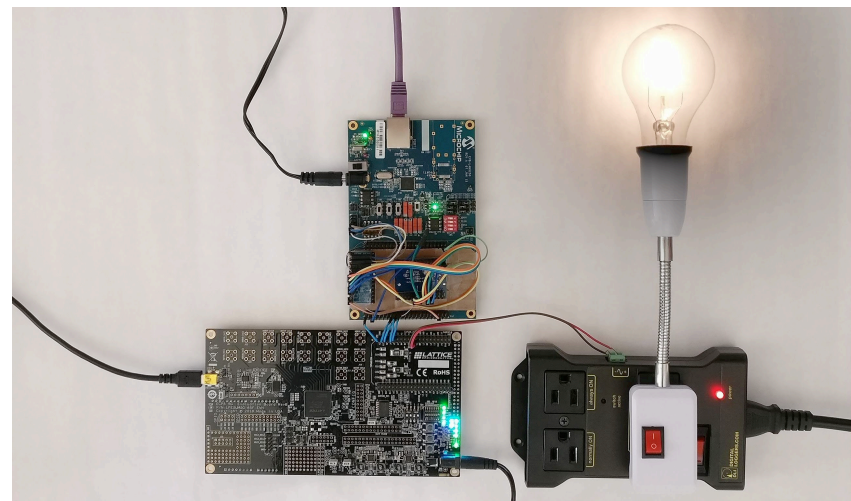
System automatically writes optimized algorithms to encrypt data in Google Chrome browsers and web applications.

Rob Matheson | MIT News Office

June 17, 2019

Takeaways

- Verification can be practical!
- Automated and manual tools can scale up to “production-grade” systems
- A lot of work remains to make verification commonplace



Z3

Formal Verification
of a Realistic Compiler
By Xavier Leroy

