Introduction to Verification

Bryan Parno (and the Verus Team)

Carnegie Mellon University

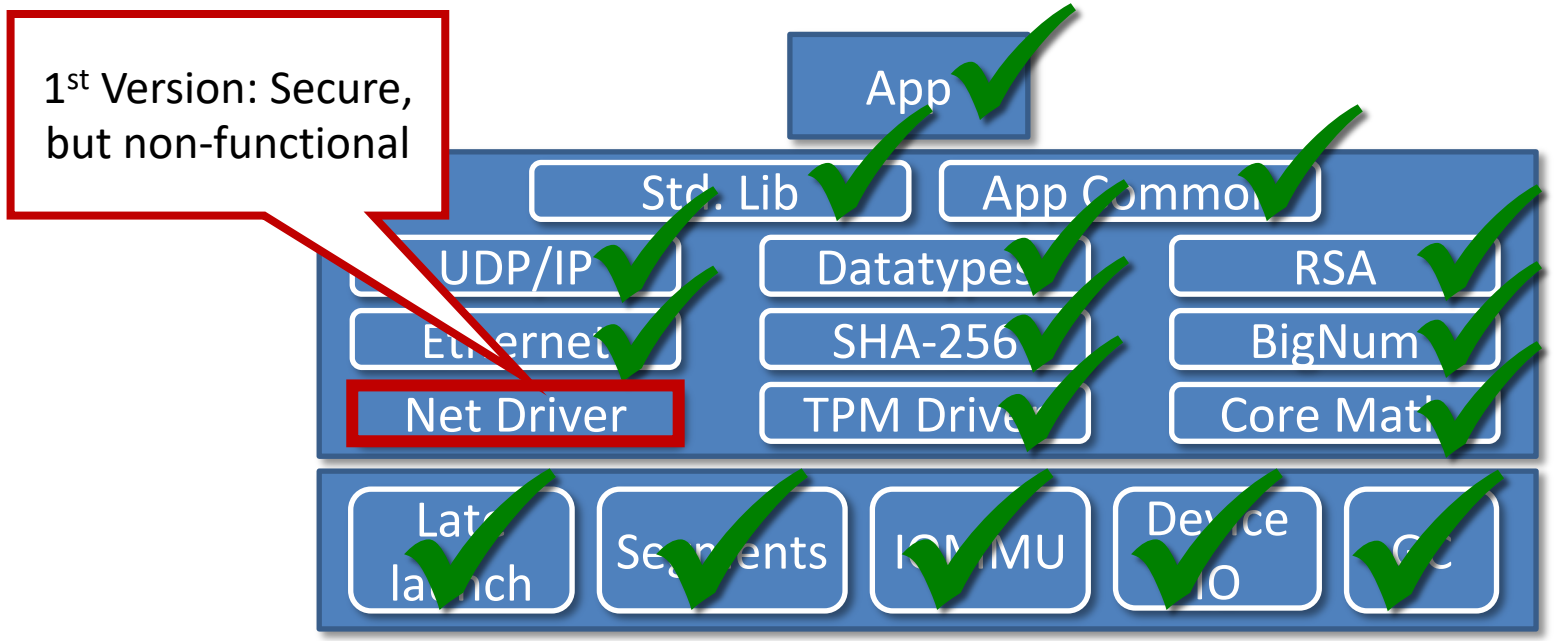
https://verus.rs/

Imagine a world…

- where code worked the 1st time you ran it
- where code was proven to be
  - correct
  - secure
  - reliable

  at compile time!

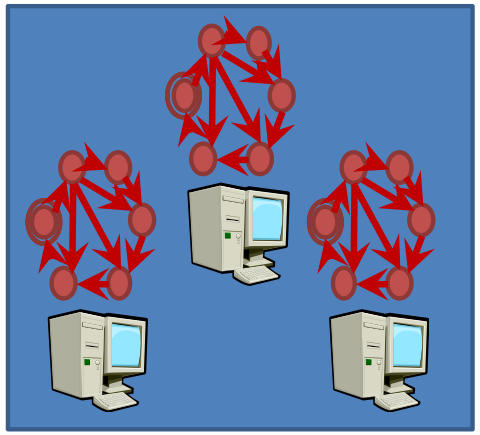
Verification can make it happen!



Verification produces better systems

[Yang et al. 2011]

CompCert *is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task.*



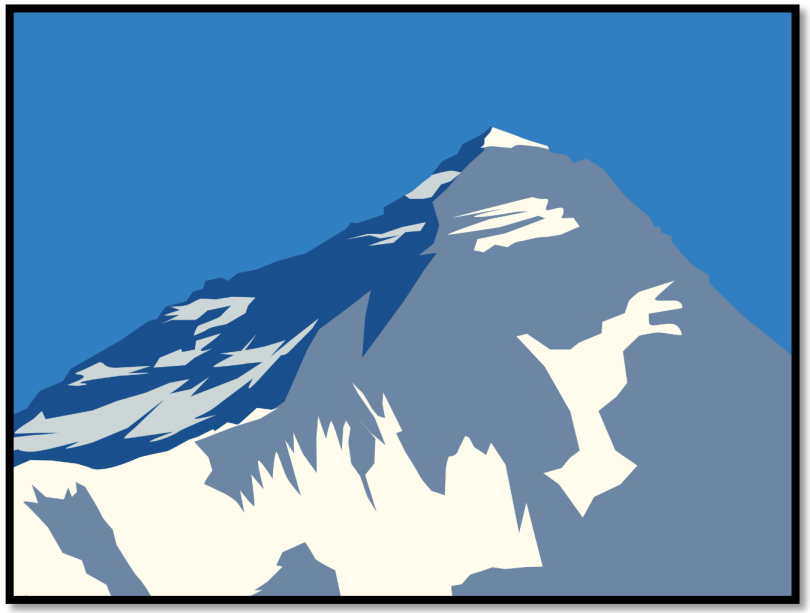
[Fonseca et al. 2017]

[N]one of these bugs were found in the distributed protocols of verified systems, despite that we specifically searched for protocol bugs and spent more than eight months in this process.

[Fisher et al. 2017]

A penetration testing expert at DARPA commented that the SMACCMCopter is probably 'the most secure UAV on the planet.'

Verification can produce faster systems

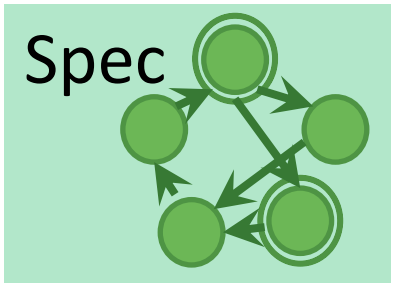


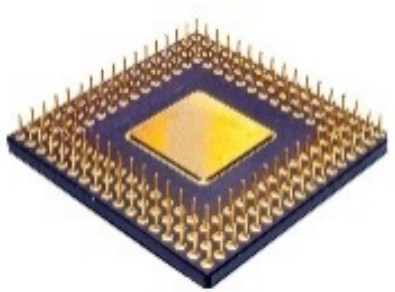
EverCrypt Crypto Provider

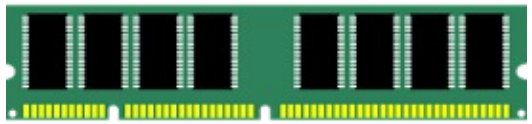
matches or exceeds the performance of state-of-the art (verified ***or unverified***) implementations!

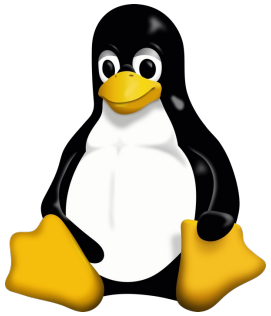
Verification limitations

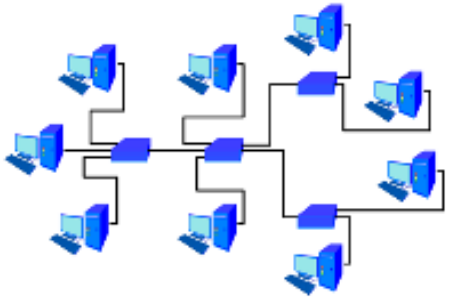
Proofs are subject to assumptions, not absolute!











What is Verification?

Is this program correct?

```
fn find_elt(elt: u64, elts: Vec<u64>) -> (result: Option<usize>)
{
    let mut i: usize = 0;
    while i < elts.len()
    {
        if elts[i] == elt {
            return Some(i);
        }
        i += 1;
    }
    None
}
```

Participation Question

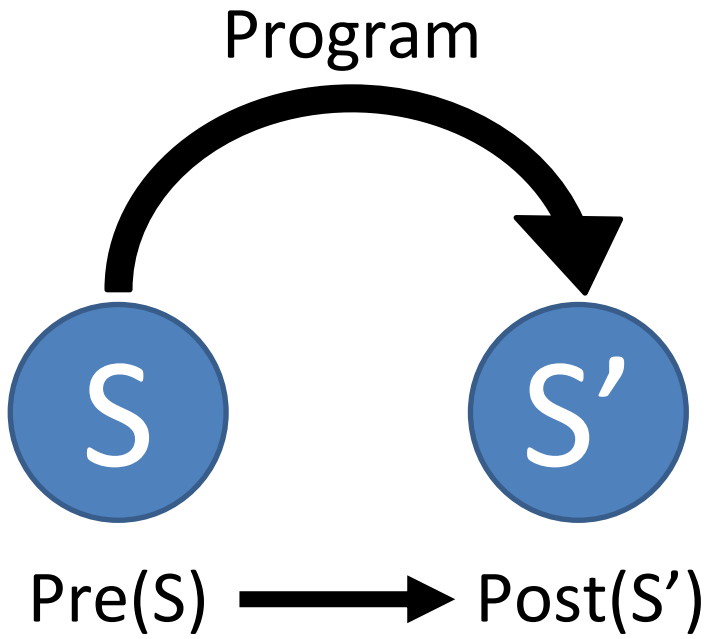
A. Yes

B. No

C. Unsure

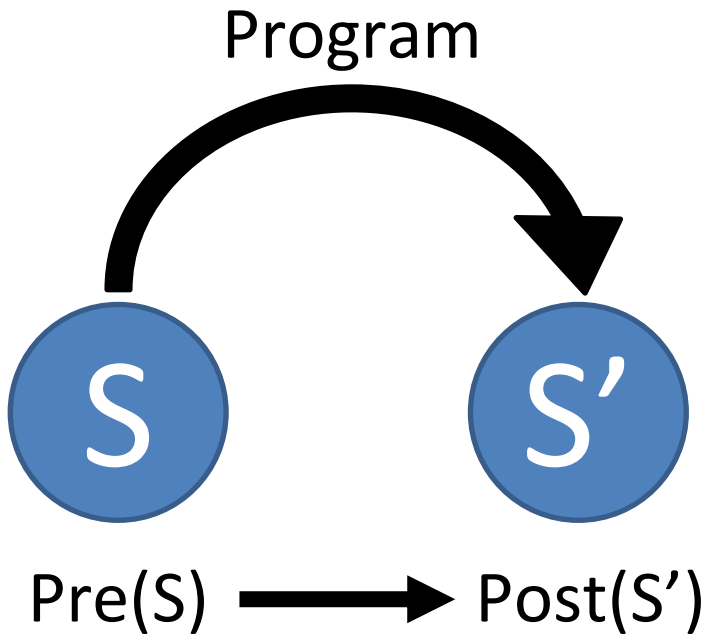
How would you argue this program is correct?

```
fn find_elt(elt: u64, elts: Vec<u64>) -> (result: Option<usize>)
{
    let mut i: usize = 0;
    while i < elts.len()
    {
        if elts[i] == elt {
            return Some(i);
        }
        i += 1;
    }
    None
}
```

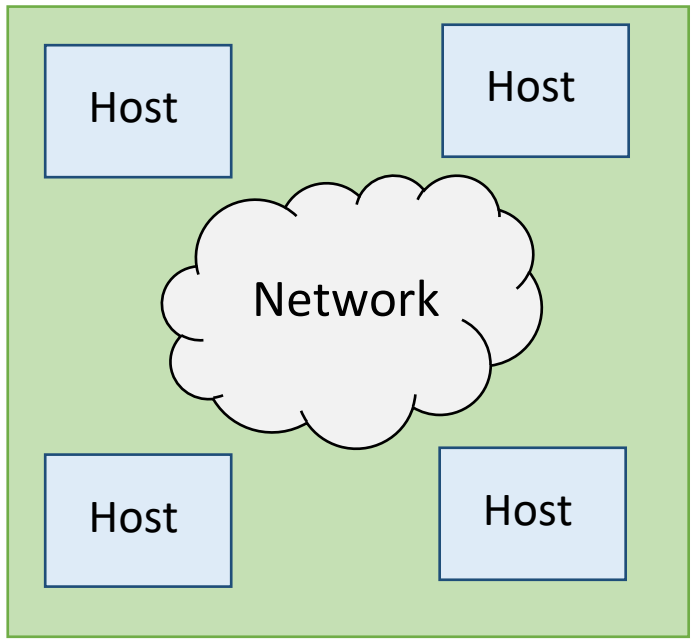


How would you argue this program is correct?

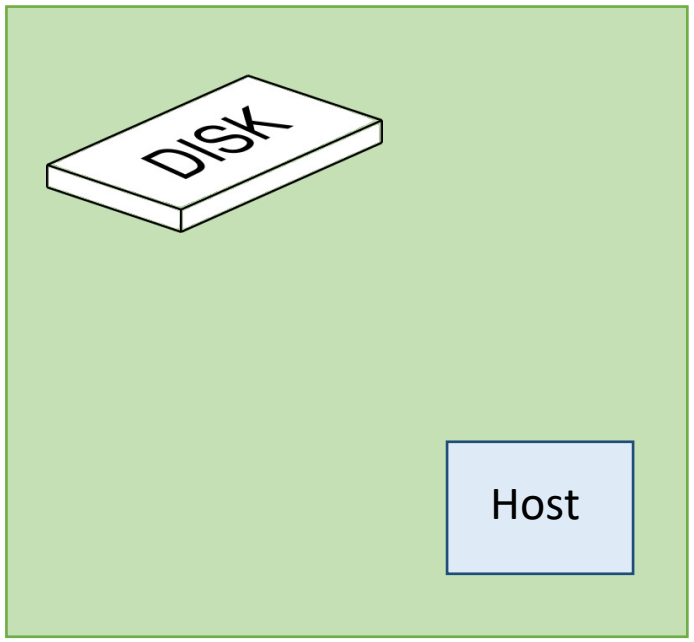
```
fn find_elt(elt: u64, elts: Vec<u64>) -> (result: Option<usize>)
    ensures




{
    let mut i: usize = 0;
    while i < elts.len()
    {
        if elts[i] == elt {
            return Some(i);
        }
        i += 1;
    }
    None
}
```



Specifying more complex properties



- Network delivering packets
- Packet reordering
- Packet duplication



- Disk
- IO queue
- Command reordering
- Host failure
- Host **reinitialization**
- (Limited) spontaneous data corruption

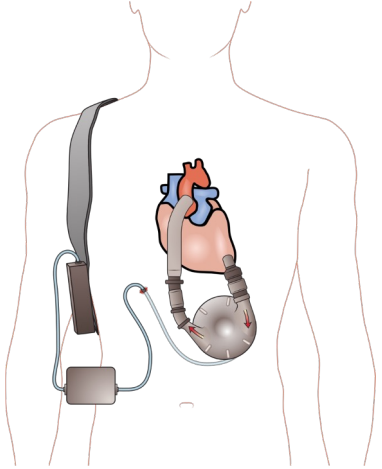
Verification in Practice

Lots of academic examples

- Compilers (CompCert, CakeML, …)
- Microkernels (seL4, Verve, Atmosphere, …)
- Secure applications (Ironclad, Knox, …)
- File systems (FSCQ, Yggdrasil, …)
- Distributed systems (Ironfleet, Verdi, Ivy,…)
- Cryptographic libraries (Evercrypt, SAW, …)
- Database optimizations (Cosette, …)

Commercial Applications, safety

- aviation
- traffic management
- medical
- space



LifeFlow ventricular assist device

(University of Virginia)

EuroFighter Typhoon

(Eurofighter Jagdflugzeug GmbH)

(Airbus, BAE Systems, Alenia Aermacchi)

NATS iFACTS system (UK)

CubeSat

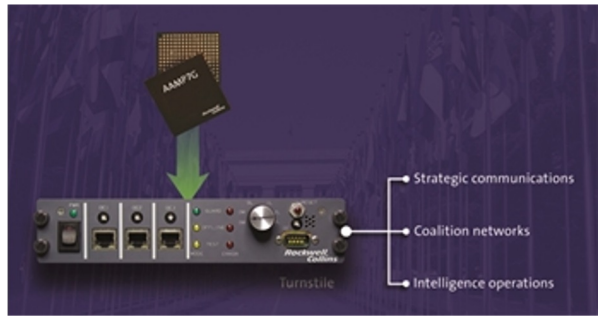
(Vermont Technical College)

ARINC

ACAMS system

Commercial Applications, security

- cross-domain guard
- smart card OS
- multi-level workstation



Turnstile, and SecureOne,
by Rockwell Collins

MULTOS,
open standard by
MULTOS Consortium

by Secunet

Commercial
Applications, software services

amazon | science

Feedback

AUTOMATED REASONING

How we b
automate
differentia

The new developme
Cedar authorization-

By Mike Hicks

May 10, 2023

Proving the correctness of AWS authorization

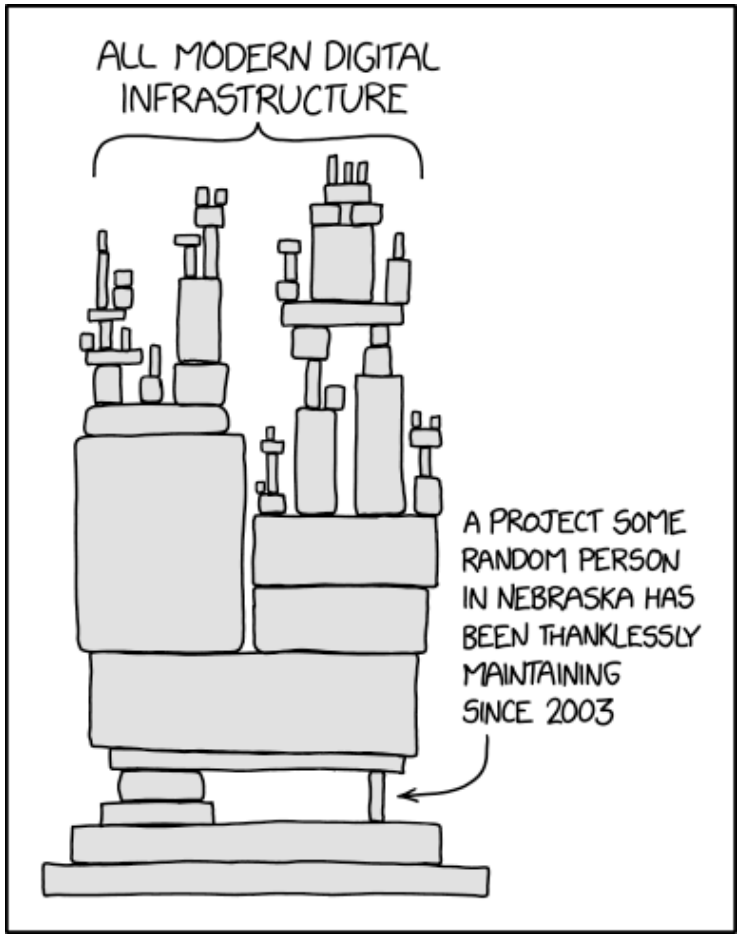
Lucas Wagner
(he/him)
Sr. Applied Science Manager
Amazon Web Services

Sean McLaughlin
(he/him)
Principal Applied Scientist
Amazon Web Services

Brief Verus Overview

Verus's Goal

Verify components at the **foundation** of modern computing



File Systems

Databases

Operating Systems

Distributed Systems

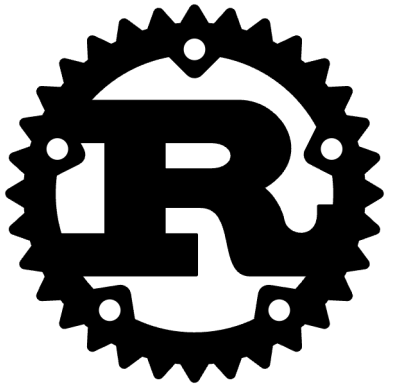
Memory allocators

…

Writing Systems Code is Hard!

- Few abstractions
- Memory layout matters
- No garbage collection
- Performance critical
  - Concurrency!

Rust Ensures Fast, Safe Systems Code



- Rust is like C/C++:
  - Low-level pointer manipulation
  - No garbage collector
  - Structs inside structs
  - In-place mutation of fields
- Rust is like Haskell/ML:
  - Type-safe & thread-safe
  - Algebraic datatypes
  - Pattern matching
  - Side effects are restricted

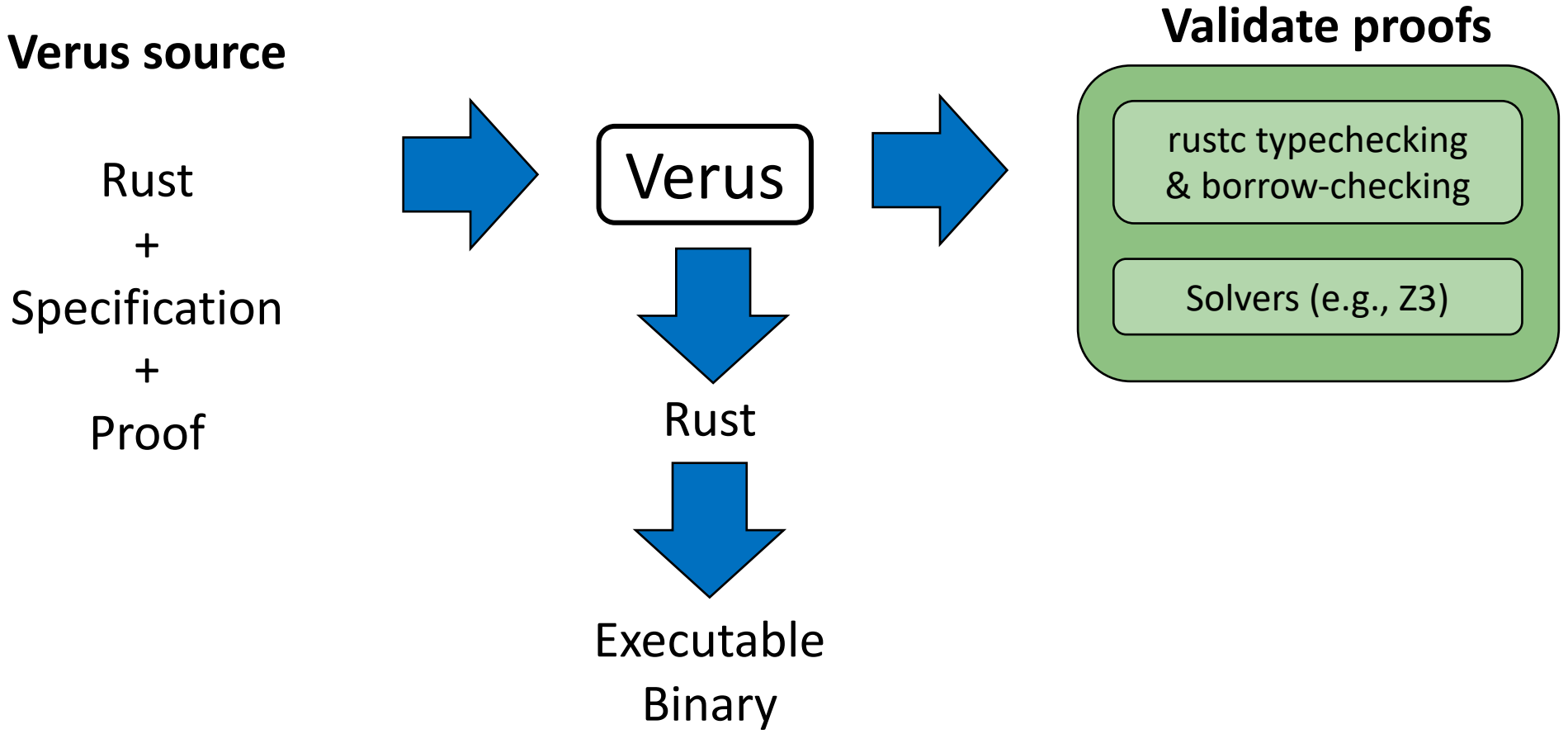
- Rust is unique:
  - Linear types
  - Borrowing

Verus: Verifying Rust Code

- Mathematically prove code is correct/reliable/secure
- Focus on fast, deterministic proof automation for systems code
- Ergonomic reasoning about (highly complex) concurrent code

Result: Scaling verification to large, complex systems

How Does Verus Work?



Verification Perf: Milli Benchmarks

	Time (seconds)
	Single
Verus	
Creusot	
Dafny	
F*	
Prusti	

Milli Benchmark 1

Verify that a singly-linked list refines a sequence

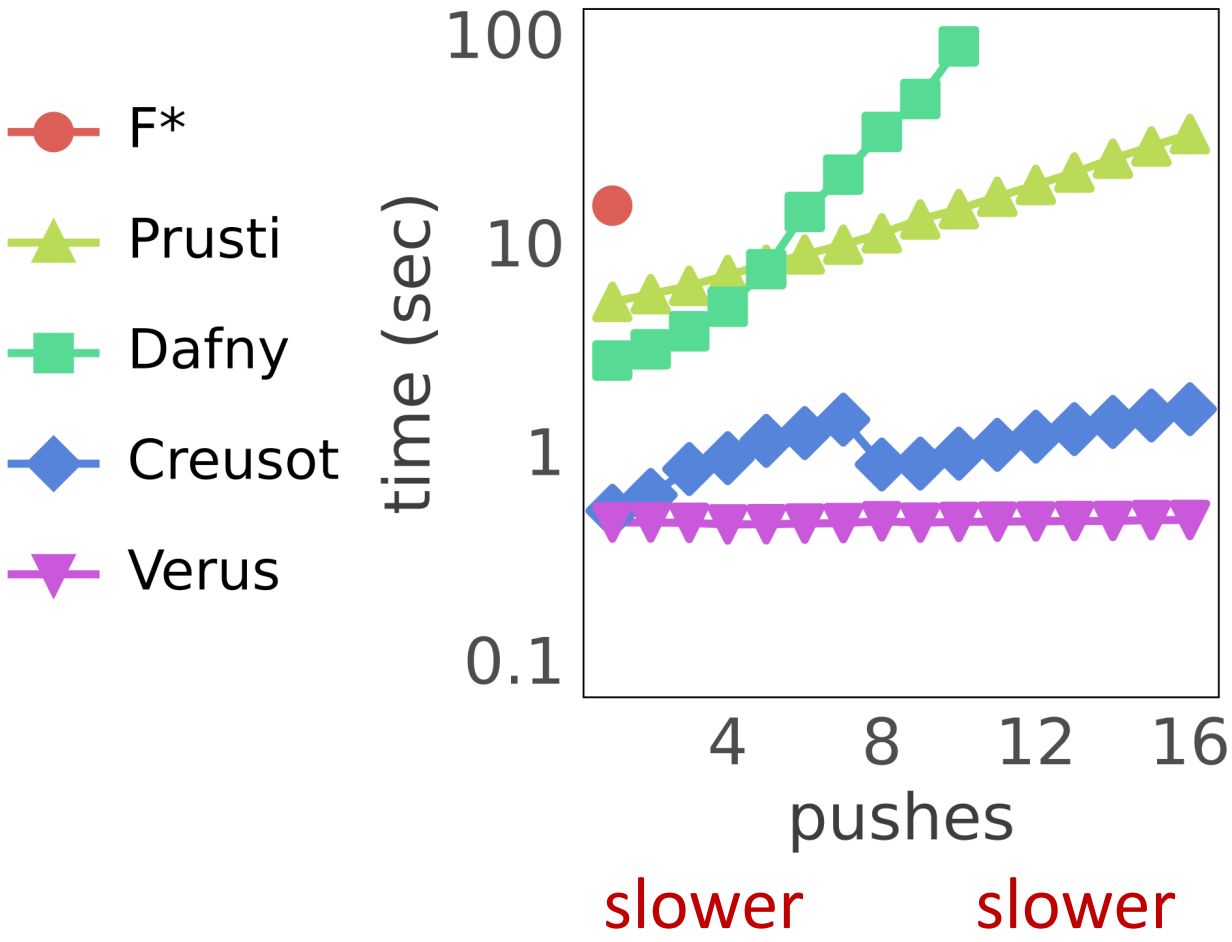
Milli Benchmark 2

Verify that a doubly-linked list refines a sequence

3-30x slower

22-56x slower

Verification Perf: Milli Benchmarks



Milli Benchmark 1

Verify that a singly-linked list refines a sequence

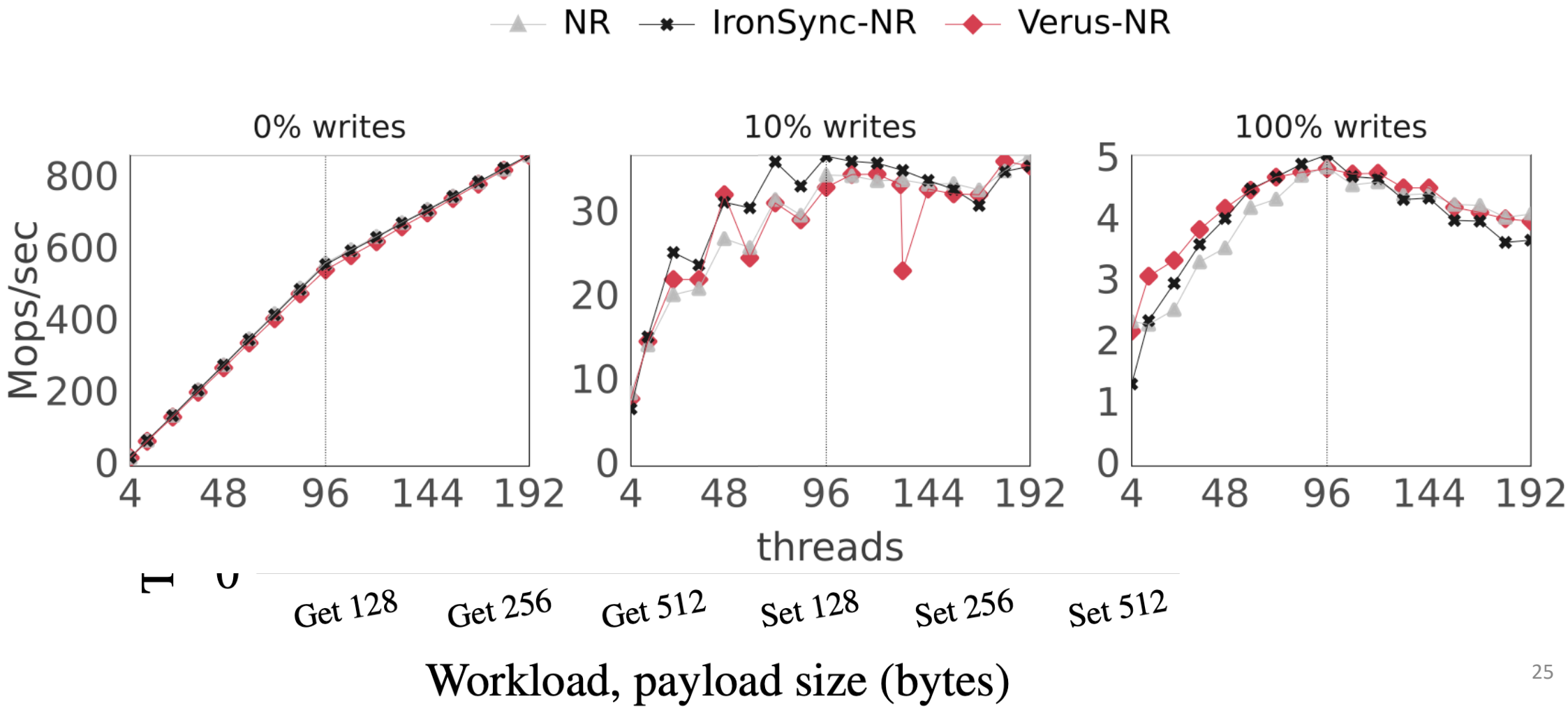
Milli Benchmark 2

Verify that a doubly-linked list refines a sequence

Milli Benchmark 3

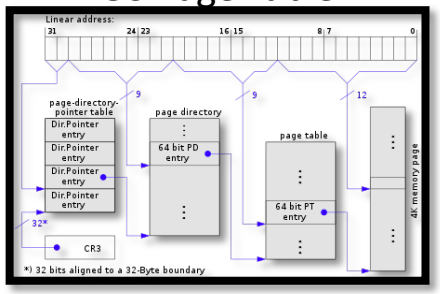
Push values onto multiple singly-linked lists and verify the resulting lists have the right values

Verification Perf: Macro Benchmarks

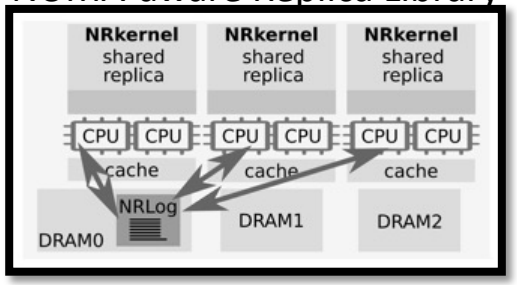


Verus Verified Systems

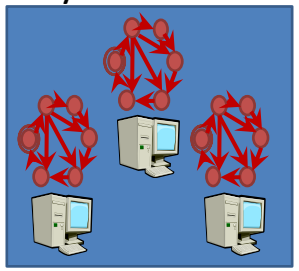
OS Page Table



NUMA-aware Replica Library

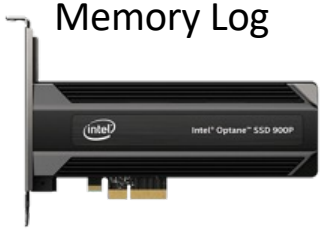


Distributed Key-Value Store



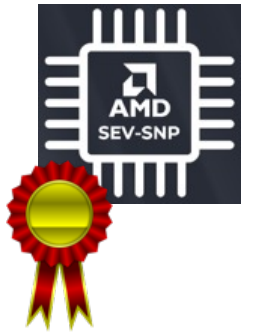
Concurrent Memory Allocator

Persistent-Memory Log

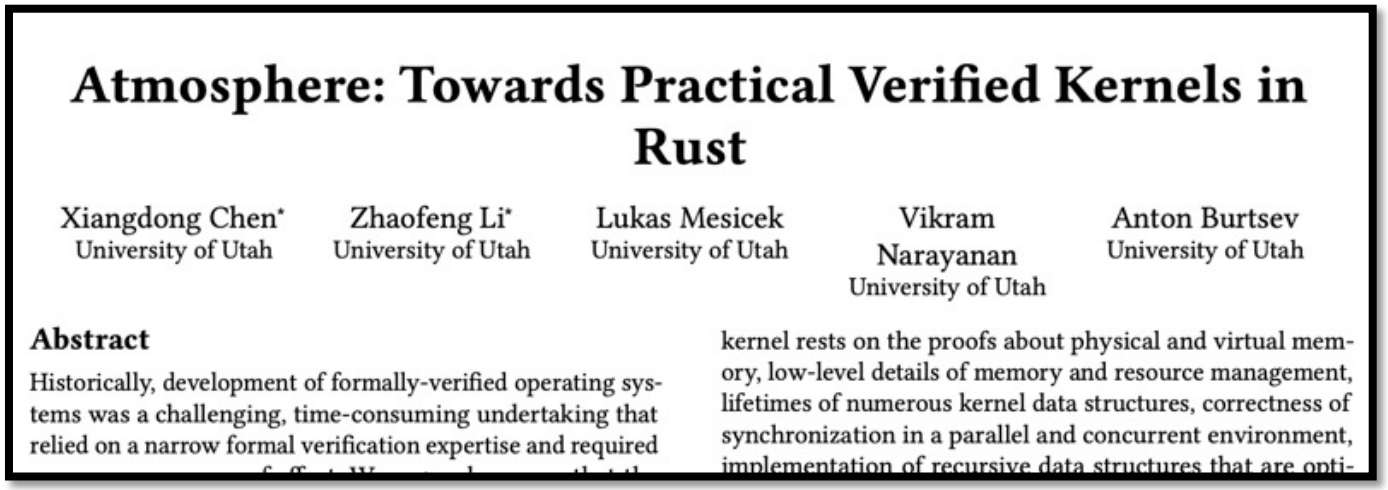


Kubernetes Controllers

Security Module for Confidential VMs



Proofs are becoming cheaper!



Atmosphere: Towards Practical Verified Kernels in Rust

Xiangdong Chen* University of Utah
Zhaofeng Li* University of Utah
Lukas Mesicek University of Utah
Vikram Narayanan University of Utah
Anton Burtsev University of Utah

Abstract

Historically, development of formally-verified operating systems was a challenging, time-consuming undertaking that relied on a narrow formal verification expertise and required

kernel rests on the proofs about physical and virtual memory, low-level details of memory and resource management, lifetimes of numerous kernel data structures, correctness of synchronization in a parallel and concurrent environment, implementation of recursive data structures that are opti-

Name	Language	Spec Lang.	Proof-to-Code Ratio
seL4	C+Asm	Isabelle/HOL	20:1 [28]
CertiKOS	C+Asm	Coq	14.9:1 [20]
SeKVM	C+Asm	Coq	6.9:1 [33]
Ironclad	Dafny	Dafny	4.8:1 [24]
NrOS	Rust	Verus (Rust eDSL)	10:1 [7]
Atmosphere	Rust	Verus (Rust eDSL)	2.3:1

Trying Verus