

Adventures in

Specification-Based Testing

John Hughes

Chalmers / Quviq

Rini Banerjee

Cambridge

Benjamin C. Pierce

Penn

Newton Institute workshop on Big Specification: Specification, Proof, and Testing at Scale
October 8, 2024

VERSE

Specification, Testing, and Verification for the
Working Software Engineer

Benjamin C. Pierce
Penn

Plan

- Property-based testing (John)
- Verse overview (Benjamin)
- A Taste of CN (Benjamin)
- CN Testing (Rini)
- Discussion
- Testing smart contracts (John)

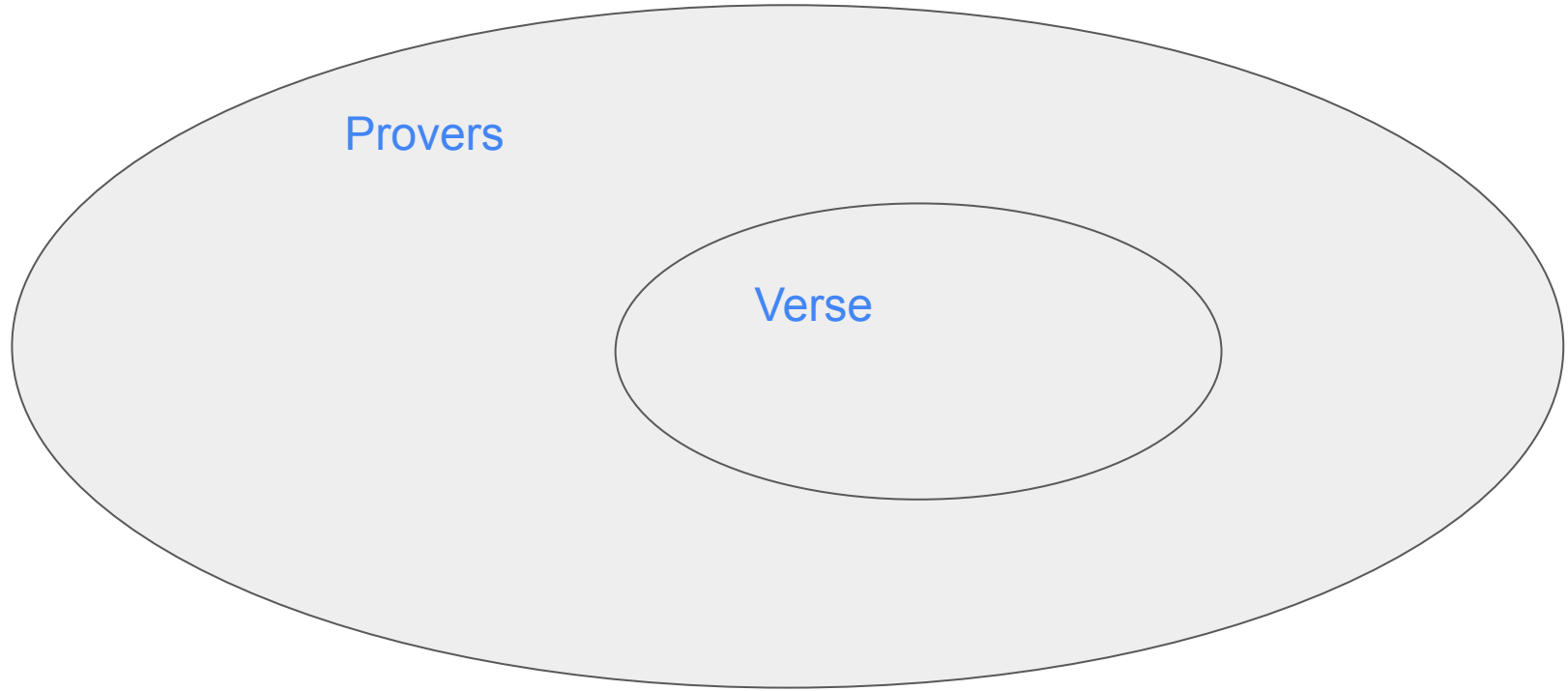
VERSE

Specification, Testing, and Verification for the
Working Software Engineer

Rini Banerjee
Cambridge

Benjamin C. Pierce
Penn

The DARPA Provers program



The DARPA PROVERS program

“The goal of PROVERS is to **make formal methods accessible to non-experts** (e.g., traditional software developers and systems engineers) while minimizing the impact on their existing processes and performance. Furthermore, the tooling would integrate into a development pipeline enabling a continuous flow of capabilities over time while maintaining high levels of assurance.”

VERSE in a nutshell

An *integrated testing and verification tool* for production C code, with a *smooth onramp* from low to high assurance



A real-world system-under-test
and

A rigorous evaluation of VERSE technologies

Academia:

- UPenn - Benjamin Pierce, Stephanie Weirich, Andrew Head
- Cambridge [UK] - Peter Sewell, Neel Krishnaswami, Christopher Pulte
- UMD - Leonidas Lampropoulos
- UIUC - Talia Ringer
- UMass - Yuriy Brun
- EPFL - Clément Pit-Claudel

Industry:

- Galois (project lead) - Mike Dodds, Joe Kiniry, Raj Patra, Cole Schlesinger
- Lynx Software Technologies
- Lockheed Martin Aeronautics



VERSE Targets...

deployed systems software,

written in C,

developed by typical engineers,

with limited assurance budgets,

and mixed assurance needs

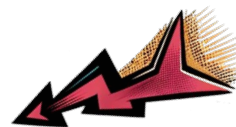
We are not the first to try this...

- FramaC
- JML
- Dafny
- Sirium
- VeriFast
- ...
- ...

What's the secret sauce?

Two things...

Test-based
assurance

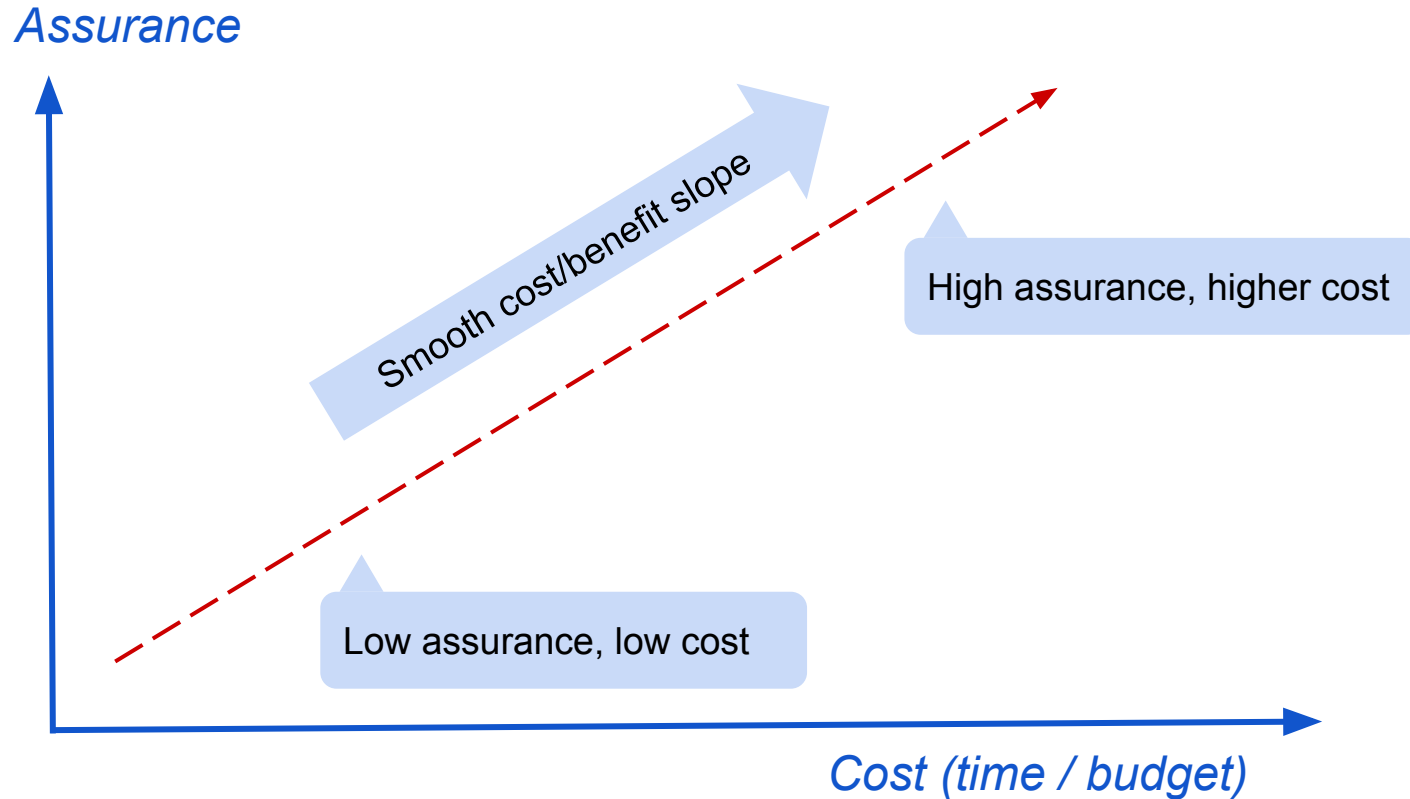


Formal
verification



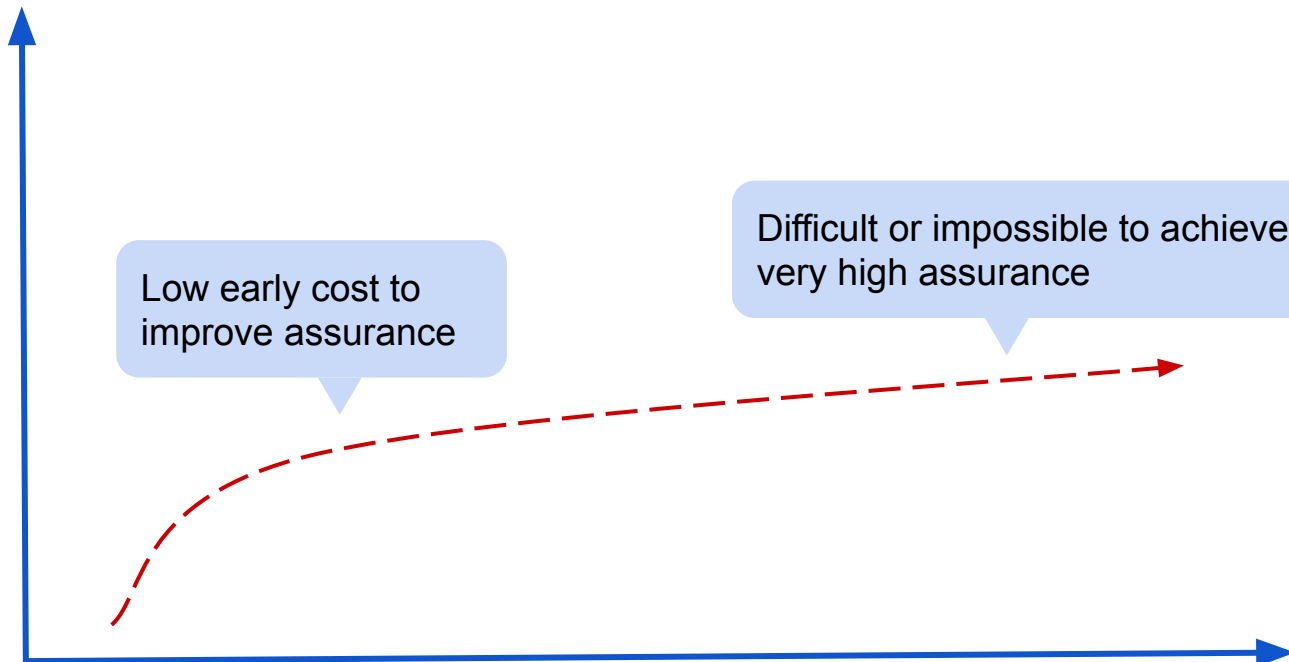
VERSE

VERSE objective: *the assurance onramp*



Compare: write-test-debug

Assurance



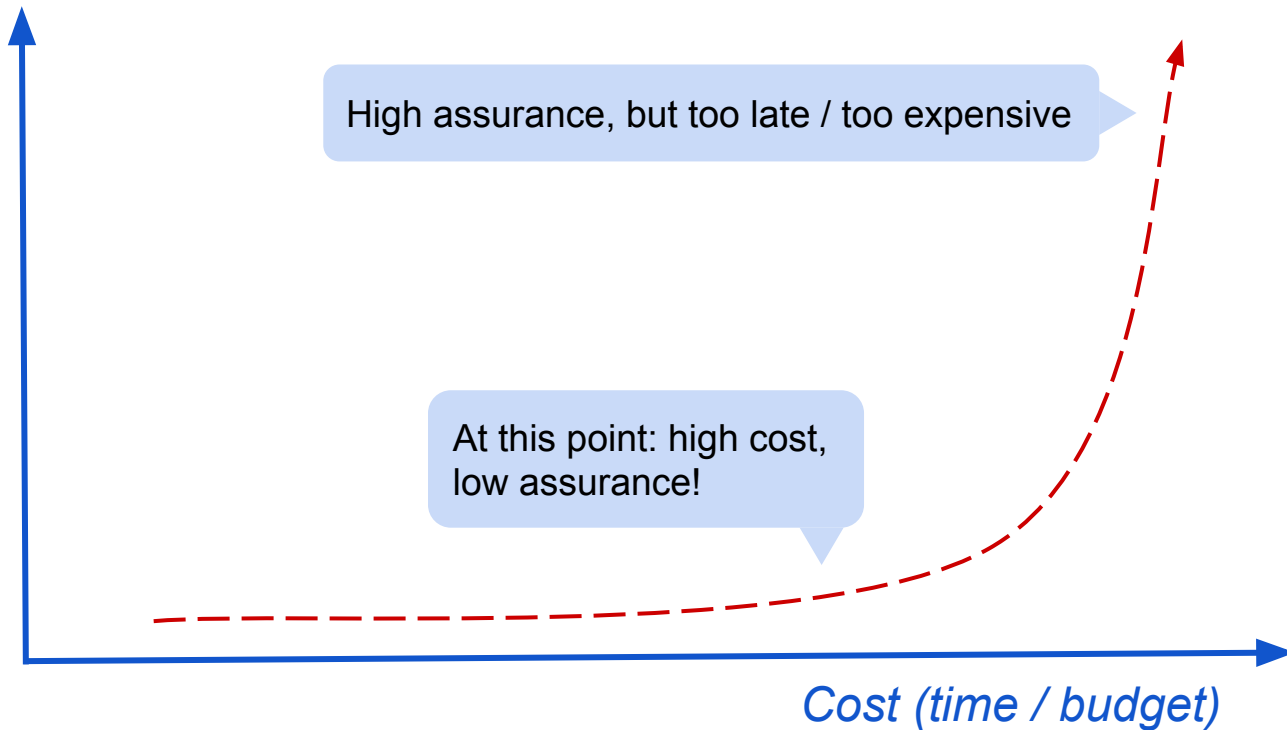
Low early cost to improve assurance

Difficult or impossible to achieve very high assurance

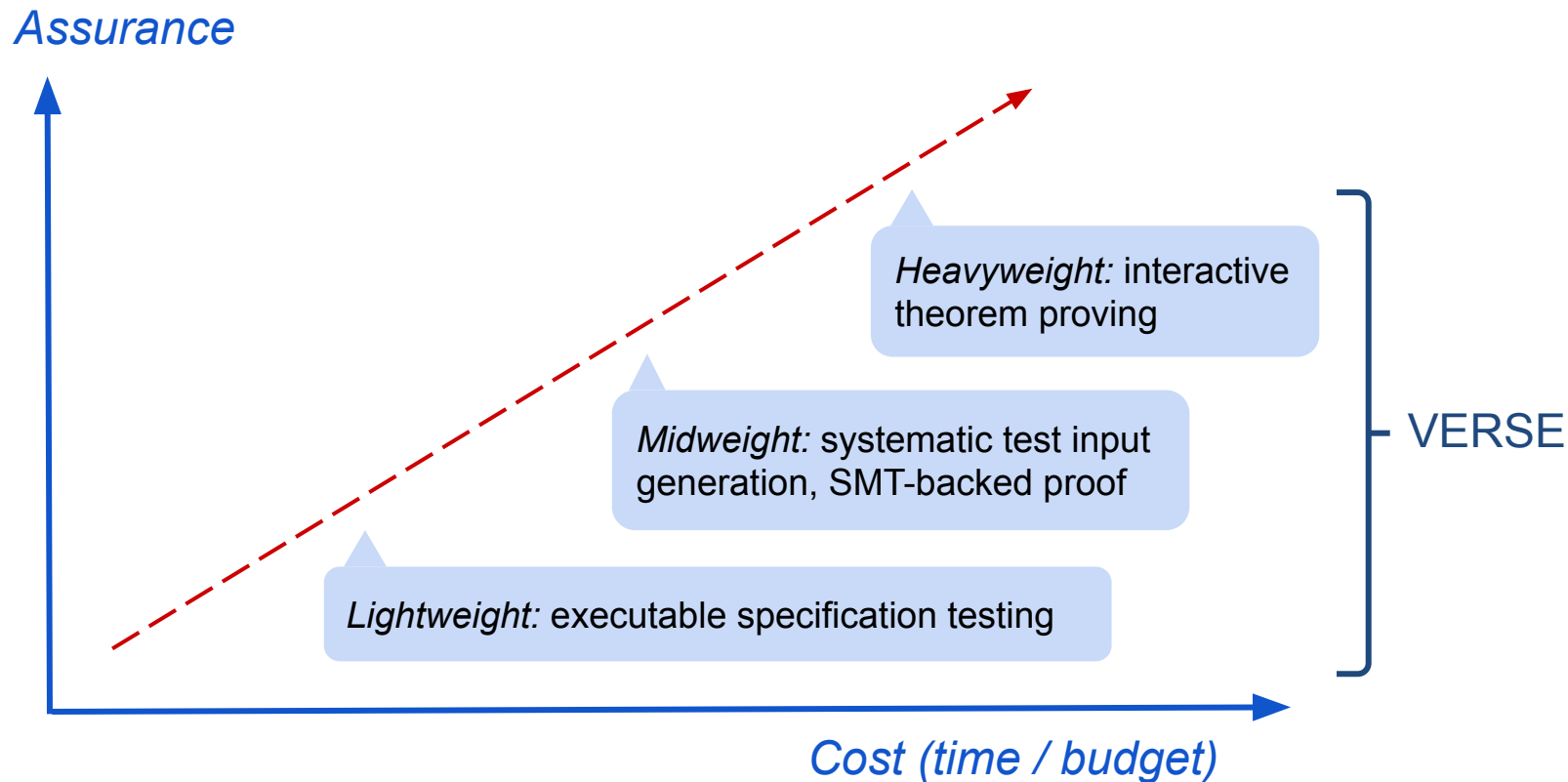
Cost (time / budget)

Compare: all-or-nothing formal methods

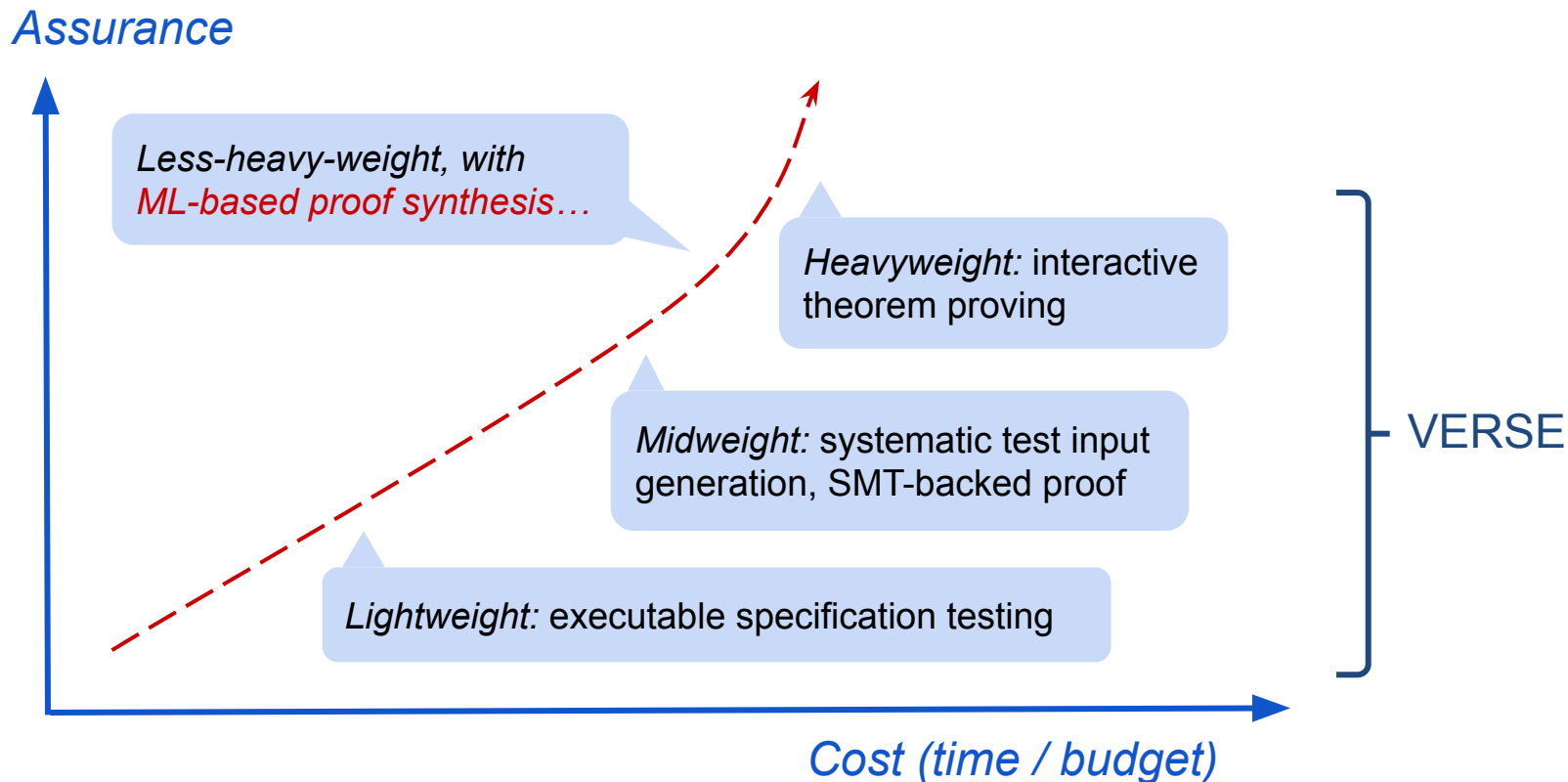
Assurance



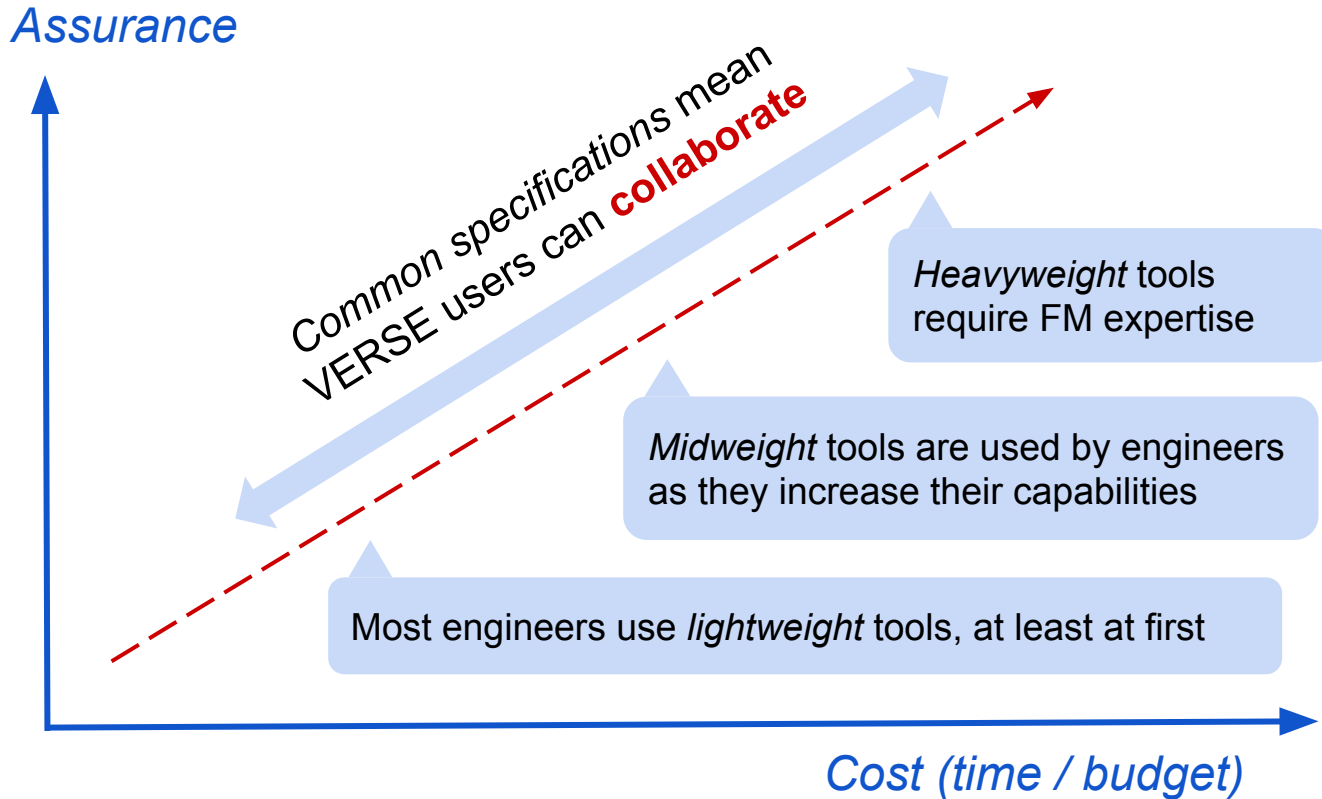
VERSE: integrated assurance technologies



VERSE: integrated assurance technologies



VERSE is designed for collaboration



Second ingredient (of the secret sauce)

Actual HCI experts!



Katrina Schleisman
(Galois)



Andrew Head
(Penn)

Second ingredient (of the secret sauce)

And actual users from (almost) day 1!



This talk:

- A taste of CN
- Property-based testing in CN

Christopher's talk:

- Deeper dive on CN

CN

CN combines...

- predictable SMT,
- separation logic,
- refinement type inference,
- export to Coq where needed, and
- executable specification

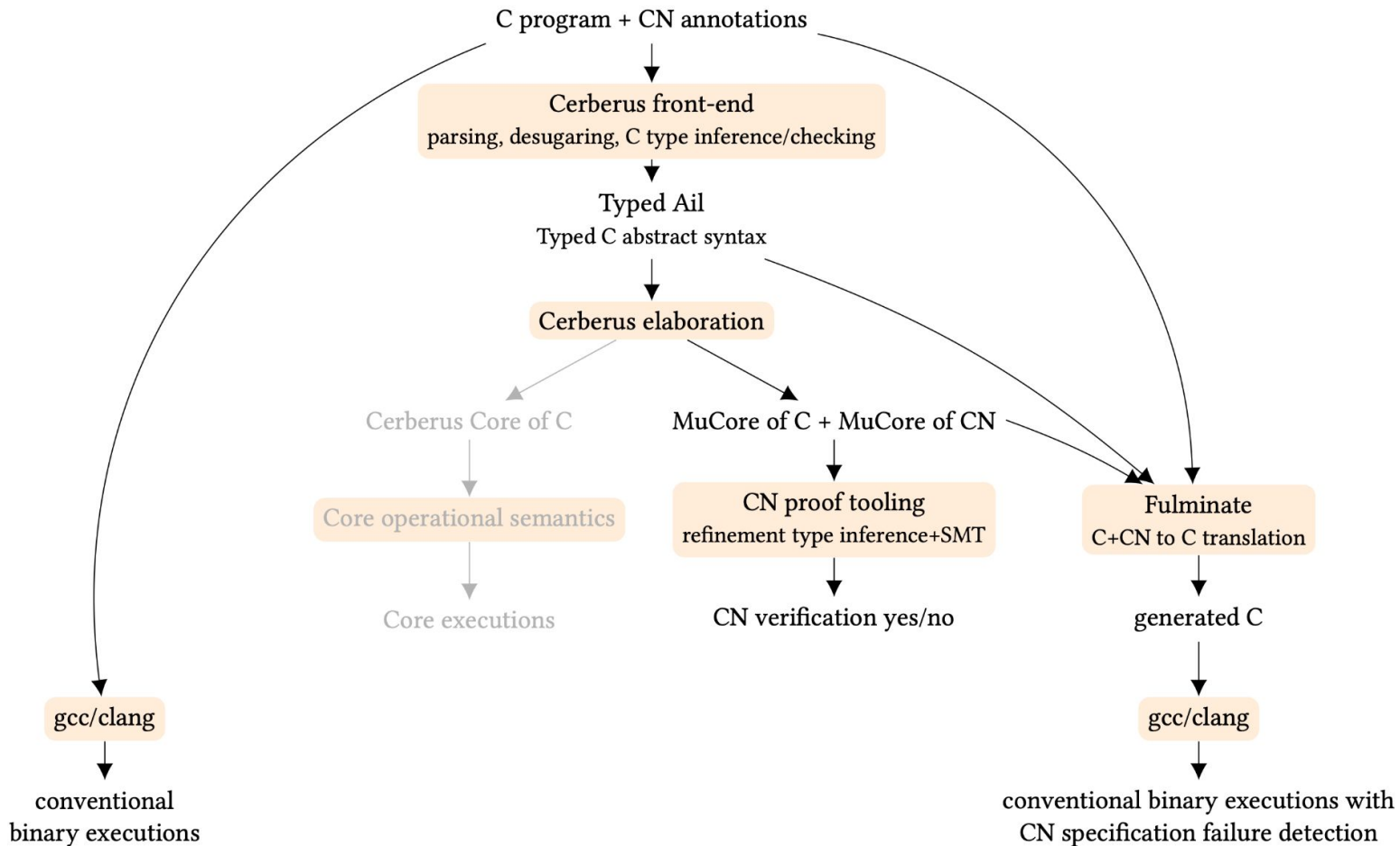


CN: Verifying Systems C Code with Separation-Logic Refinement Types

CHRISTOPHER PULTE, University of Cambridge, UK
DHRUV C. MAKWANA, University of Cambridge, UK
THOMAS SEWELL, University of Cambridge, UK
KAYVAN MEMARIAN, University of Cambridge, UK
PETER SEWELL, University of Cambridge, UK
NEEL KRISHNASWAMI, University of Cambridge, UK

Despite significant progress in the verification of hypervisors, operating systems, and compilers, and in verification tooling, there exists a wide gap between the approaches used in verification projects and conventional development of systems software. We see two main challenges in bringing these closer together: verification handling the complexity of code and semantics of conventional systems software, and verification usability.

We describe an experiment in verification tool design aimed at addressing some aspects of both: we design and implement CN, a separation-logic refinement type system for C systems software, aimed at predictable proof automation, based on a realistic semantics of ISO C. CN reduces refinement typing to



Precondition

```
signed int increment(signed int i)
/*@ requires i < power(2, 31) - 1 @*/
/*@ ensures return == i + 1 @*/
{
    return i + 1;
}
```

Postcondition

```
void increment ( signed int *p )
/*@ requires take i1 = Owned(p); @*/
/*@ requires i1 < power (2i32, 31i32) - 1i32; @*/
/*@ ensures take i2 = Owned(p); @*/
/*@ ensures i2 == i1 + 1i32; @*/
{*p = *p + 1;}
```

*Acquire exclusive
access to p*

Release access to p

Restricted format for ownership specifications

requires take $i1$ = Owned (p) ;

*(Deterministic)
output*



*(Known)
input*



exists $i1, p \mid \rightarrow i1$

VERSE status

- 6 months into a 3.5-year project
- Still pre-alpha
 - CN already working on lots of examples
 - E.g., pKVM buddy allocator
 - Many things still under construction
- Rudimentary VSCode-based programming environment

Testing CN

Runtime checking of CN specifications

- Dynamic checking of rich properties (including ownership)
- Both proof and testing from single annotation language
- Incremental ramp to formal verification
 - Not all-or-nothing
 - Guides spec-writing process

Example

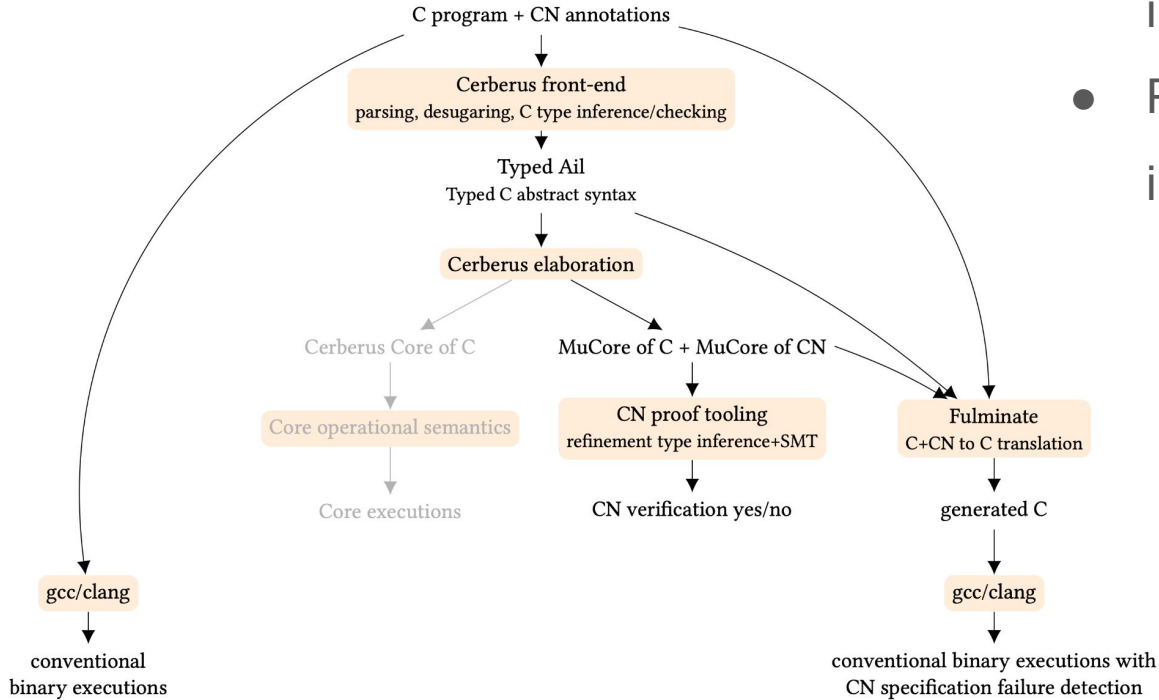
```
signed int increment(signed int i)
/*@ requires i < power(2, 31) - 1 @*/
/*@ ensures return == i + 1 @*/
{
    return i + 1;
}
```



Source-to-source
translation

```
signed int increment(signed int i)
{
    check_increment_precondition(i);
    signed int ret_value = i + 1;
    check_increment_postcondition(i, ret_value);
    return ret_value;
}
```

How does this work?

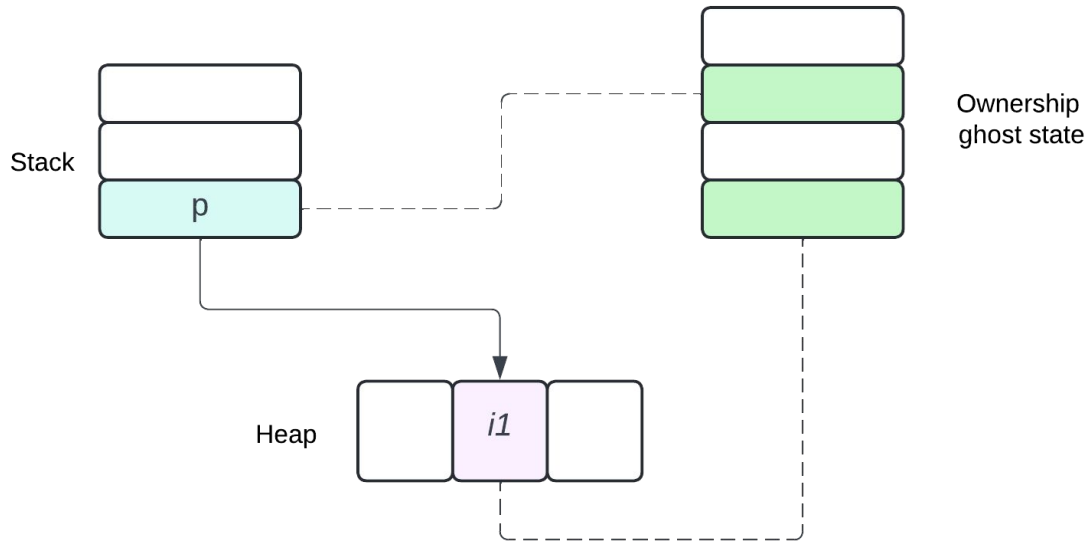


- Takes C+CN input
- Produces Ail, a Cerberus internal representation of C
- Pretty-prints Ail and injects into C source

Dynamic ownership checking

- Global ownership ghost state that maps addresses to function stack depth
- Translates CN's `owned` resource predicate into a runtime check
 - This “**gets**” or “**puts back**” ownership in the global ghost state
- Also involves mapping + unmapping addresses of C stack-local variables
 - Checks every load and store

```
void increment ( signed int *p )
/*@ requires take i1 = Owned(p); @*/
/*@ requires i1 < power (2i32, 31i32) - 1i32; @*/
/*@ ensures take i2 = Owned(p); @*/
/*@ ensures i2 == i1 + 1i32;@*/
{*p = *p + 1;}
```



Technical bits

- Dealing with the C
 - Struct plumbing
 - Header files + multiple includes
 - Injection
- Destination passing
 - CN (expression language) to C (statement language)
 - Destination of expr could be: **Assert, Return, AssignVar** Or **PassBack**
- Pattern matching and datatypes in C
- Control flow: **return, break** (**goto** and **continue** TBD)

Demo

Discussion

Specification

Testing

Proof

Three great tastes that go great together!

But what about **static analysis**?

Restricted expressiveness



Improved automation

??

Thank you!

More discussion??

Want to play?

Leave us your email here
(or send it to me)



<https://forms.gle/xgumxfZNpTbSHFA79>