Project context: Continuous Reasoning with Gradual Verification

- Continuous reasoning: the ability to reason about the security of software in an ongoing way as the software evolves
- Gradual verification: using a combination of static and dynamic analysis to reason in the presence of partial and evolving specifications
- Today: continuous reasoning about memory safety in multi-language Rust applications, using a combination of static and dynamic analysis

A Study of Undefined Behavior Across Foreign Function Boundaries in Rust Libraries

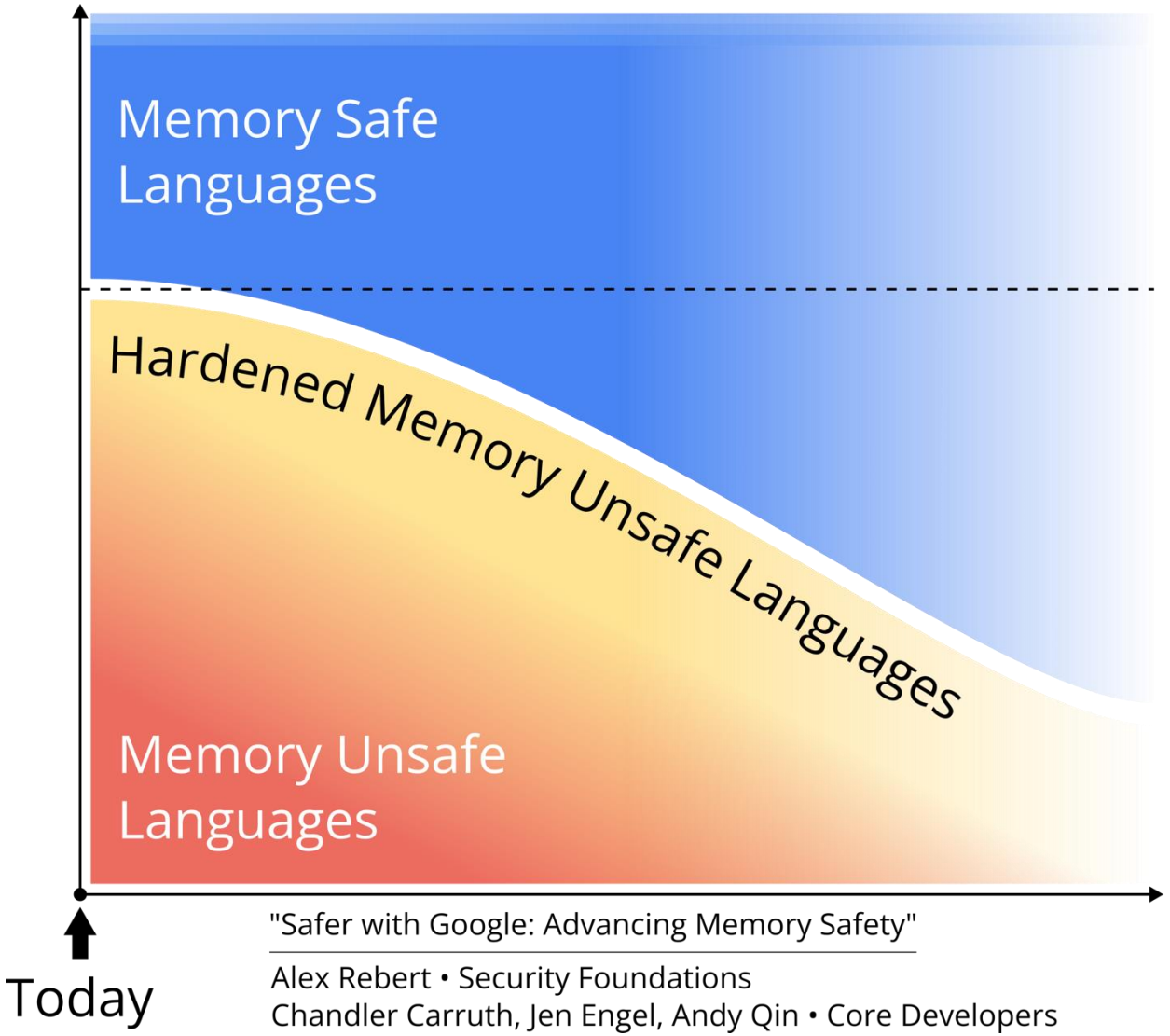
Ian McCormack
Carnegie Mellon University

Joshua Sunshine
Carnegie Mellon University

Jonathan Aldrich
Carnegie Mellon University

S3D Software and Societal Systems Department

Preprint



"Safer with Google: Advancing Memory Safety"

Alex Rebert • Security Foundations
Chandler Carruth, Jen Engel, Andy Qin • Core Developers

The Rust Programming Language

Rust can prevent safety errors
without runtime overhead.

Ownership

"The Usability of Ownership" • Will Crichton

Values have exactly one owner, *or none*.

A reference to a value cannot outlive the owner.

A value can have **one** *mutable* reference
or many *immutable* references

Safe References

`&T`	`&mut T`
Shared, Read-only	Unique, Write

Rust developers use a set of ”unsafe” features to interoperate with other languages.

Calling `unsafe` functions

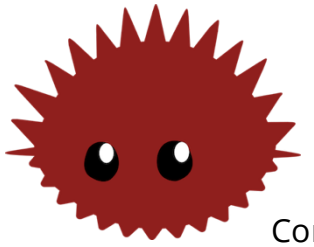
Dereferencing raw pointers

Intrinsics & inline assembly

Implementing an `unsafe` trait

Manipulating uninitialized memory

Accessing global, mutable state

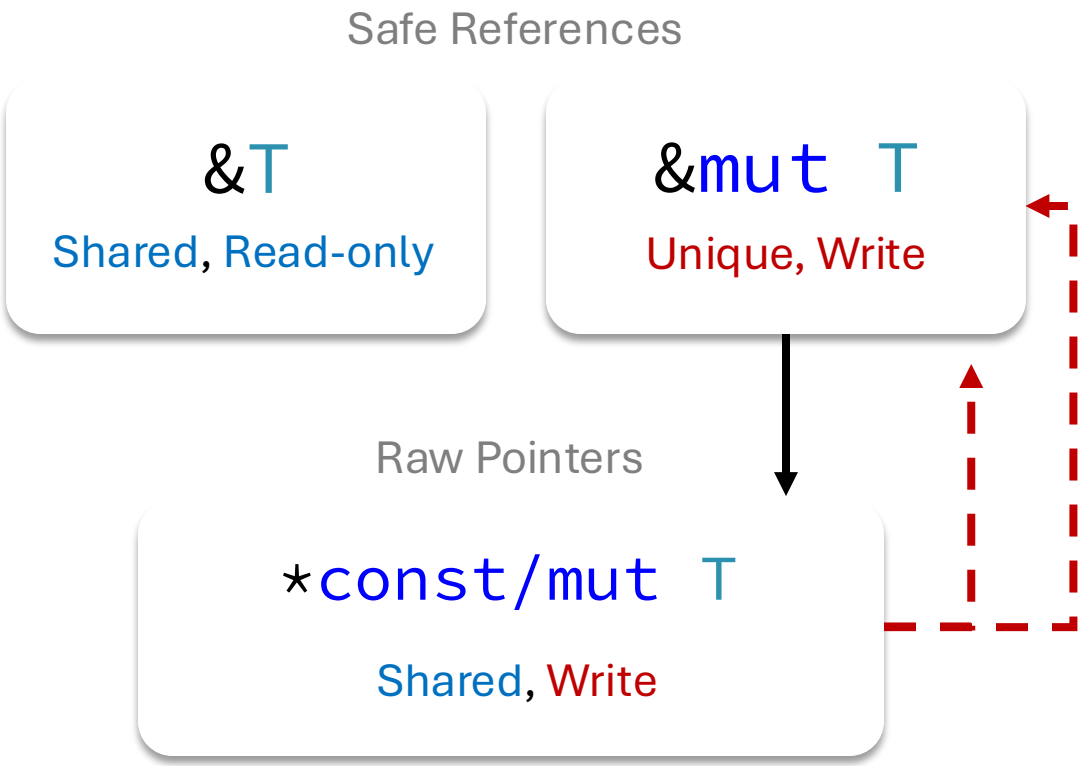


Corro

Research Questions

RQ1 What types of errors occur in Rust libraries that call foreign functions?

Developers can use unsafe code to break Rust's aliasing rules.

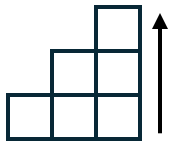



```
let mut x: i8 = 0;
let rx = &mut x;
let ptr = rx as *mut _;

example(rx, unsafe { &mut *ptr });

fn example(x: &mut i8, y: &mut i8) {
  *x = 0
  *y = 1;
  *x;
}
```

Credit: Ralf Jung, Hoang-Hai Dang, Jeehoon Kang, and Derek Dreyer

Miri, a Rust interpreter, can find these aliasing bugs.

Stacked Borrows: an Aliasing Model for Rust



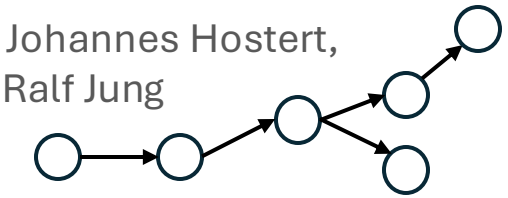
Ralf Jung, Hoang-Hai Dang, Jeehoon Kang, and Derek Dreyer

POPL, '20

OR

Tree Borrows

Neven Villani, Johannes Hostert, Derek Dreyer, Ralf Jung



PLDI, '25

Bounds Checking

Liveness Checking

Data Race Detection

Pointer

(Address, Provenance)

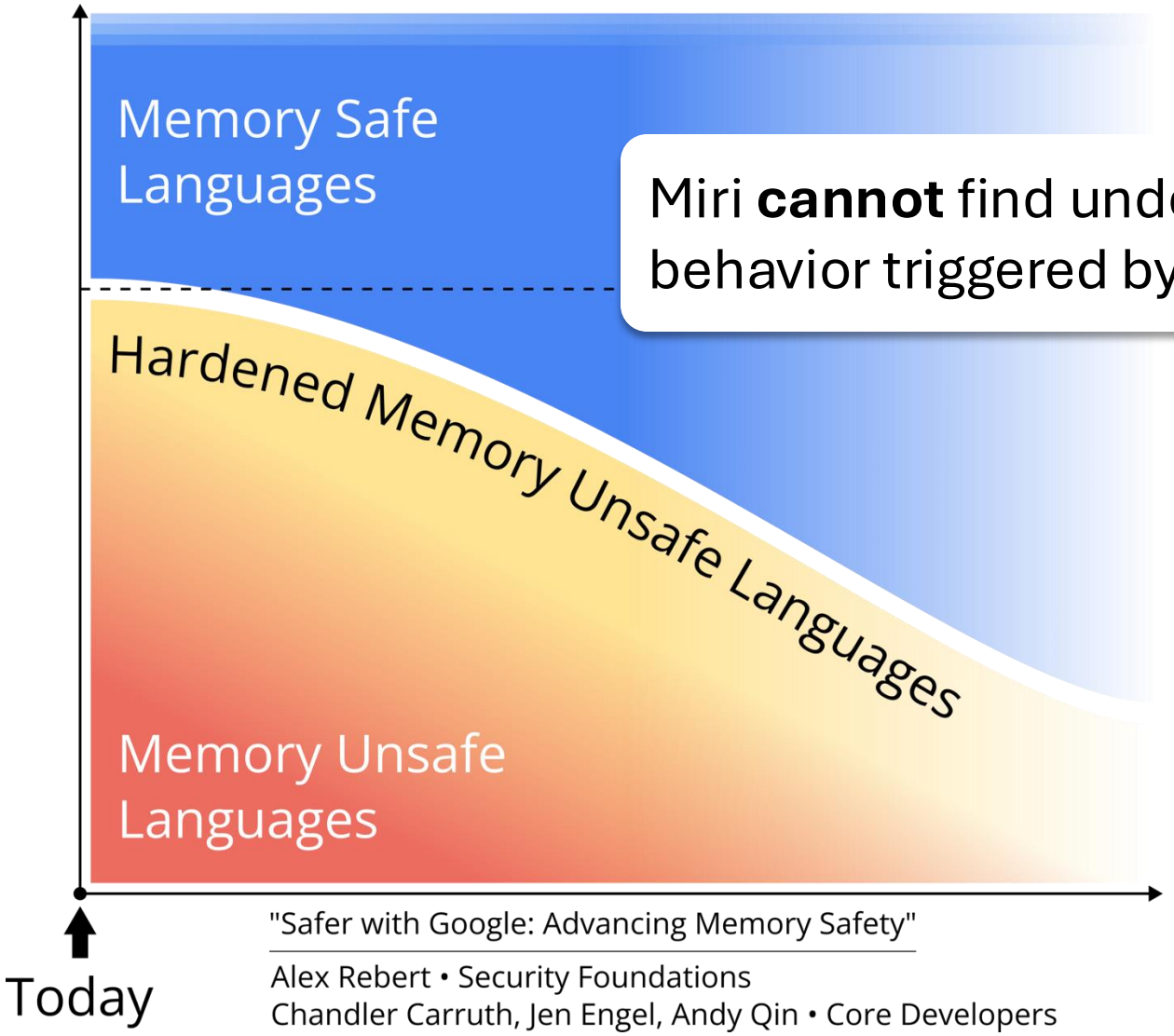
$\mathbb{N}$

→

Provenance

(Allocation ID, Tag)

$\mathbb{N}$ $\mathbb{N}$



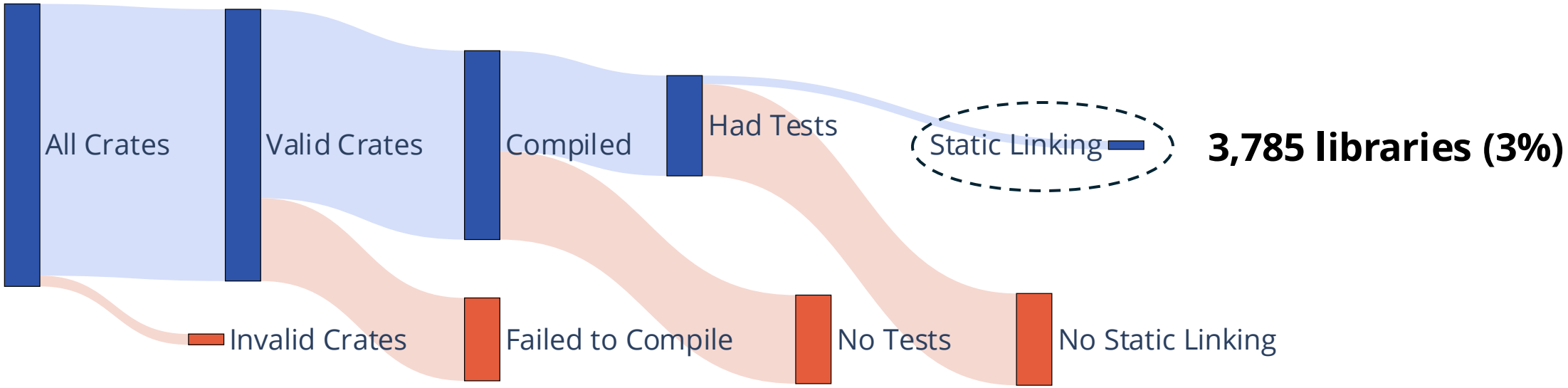
Miri **cannot** find undefined behavior triggered by foreign code!

Research Questions

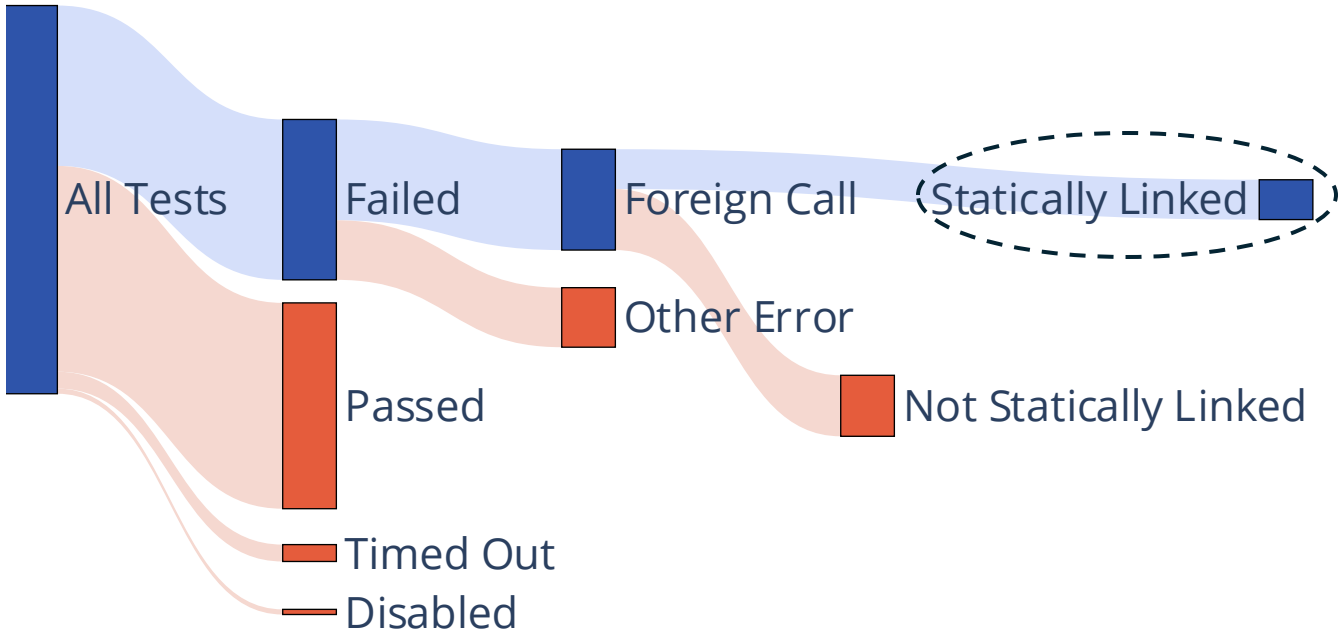
RQ1 What types of errors occur in Rust libraries that call foreign functions?

RQ2 Which of Rust's aliasing models permits more real-world programs with foreign function calls?

In September of 2023, we searched through all **125,804** Rust libraries published on crates.io to find test cases that statically linked to foreign code.



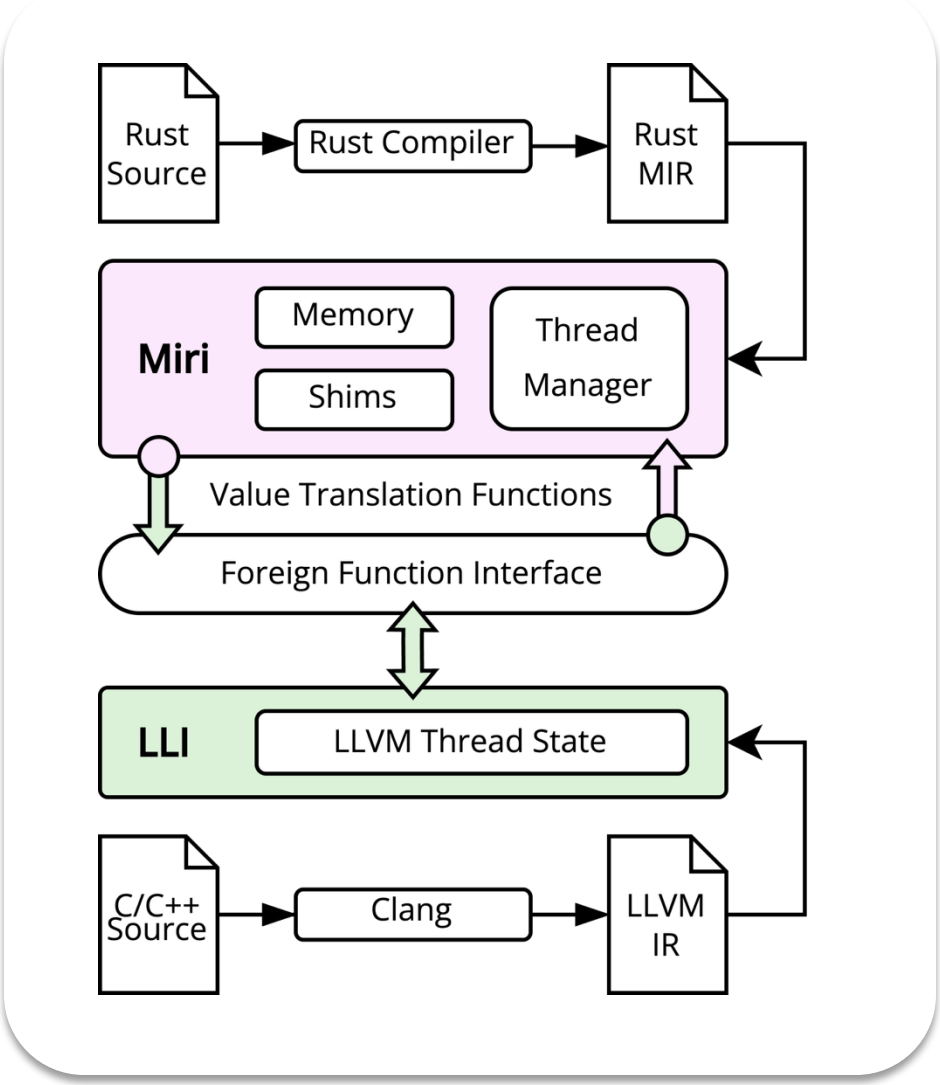
We executed all **88,637** tests from these libraries in Miri to find the subset of tests that called foreign functions which were statically linked.



9,130 tests (10%) from 957 crates

We combined **Miri** with **LLI**, an LLVM interpreter, to create **MiriLLI**.

Our tool uses each interpreter to jointly execute programs defined across **Rust** and **LLVM IR**.



We executed all **9,130** tests in **MiriLLI**.

61% encountered an unsupported operation,
but **9% had a potential bug**.

We deduplicated all errors into **394** “unique” test outcomes.

```
error: Undefined Behavior: write access through <984> at alloc594[0x0] is forbidden
 --> src/main.rs:9:5
  |
  |     *wx = 2;
  |     ^^^^^^^ write access through <984> at alloc594[0x0] is forbidden
  |
```

We found **46** instances of undefined or undesirable behavior from **37** libraries.

Location		Bug Type			Total
Fix	**Error**	**Allocation**	**Ownership**	**Typing**	
Binding	Binding	-	-	6	6
Binding	LLVM	-	3	-	3
LLVM	LLVM	-	3	-	3
Rust	LLVM	1	**16**	-	17
Rust	Rust	9	2	6	17
	Total:	10	**24**	12	**46**

We found **46** instances of undefined or undesirable behavior from **37** libraries.

Location		Bug Type			Total
Fix	**Error**	Allocation	Ownership	Typing	
Binding	Binding	-	-	6	6
Binding	LLVM	-	3	-	3
LLVM	LLVM	-	3	-	3
Rust	LLVM	1	**16**	-	17
Rust	Rust	9	2	6	17
	Total:	10	**24**	12	**46**

We found **46** instances of undefined or undesirable behavior from **37** libraries.

Location		Bug Type			Total
Fix	Error	Allocation	Ownership	Typing	
Binding	Binding	-	-	6	6
Binding	LLVM	-	3	-	3
LLVM	LLVM	-	3	-	3
Rust	LLVM	1	**16**	-	17
Rust	Rust	9	2	6	17
	Total:	10	**24**	12	**46**

We found **46** instances of undefined or undesirable behavior from **37** libraries.

Location		Bug Type			Total
Fix	**Error**	**Allocation**	**Ownership**	**Typing**	
Binding	Binding	-	-	6	6
Binding	LLVM	-	3	-	3
LLVM	LLVM	-	3	-	3
Rust	LLVM	1	**16**	-	17
Rust	Rust	9	2	6	17
	Total:	10	**24**	12	**46**

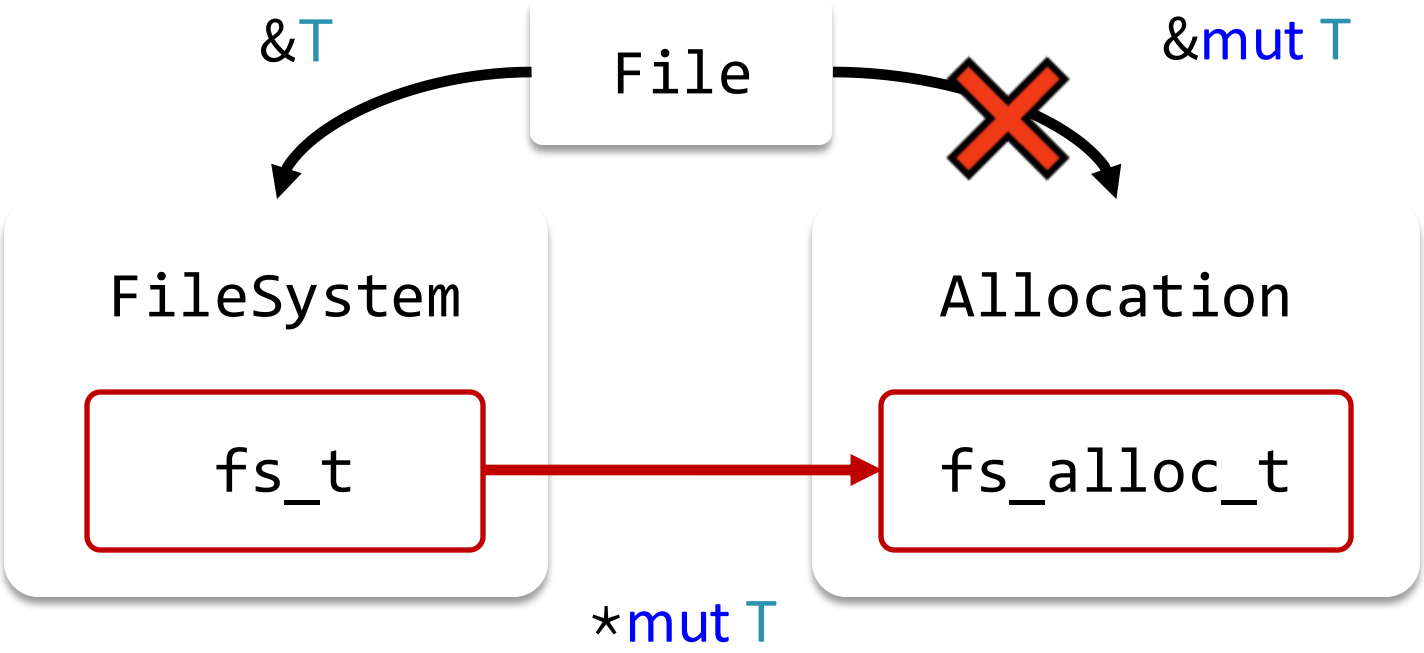
90 tests that had a Stacked Borrows violation
66% were accepted by Tree Borrows.

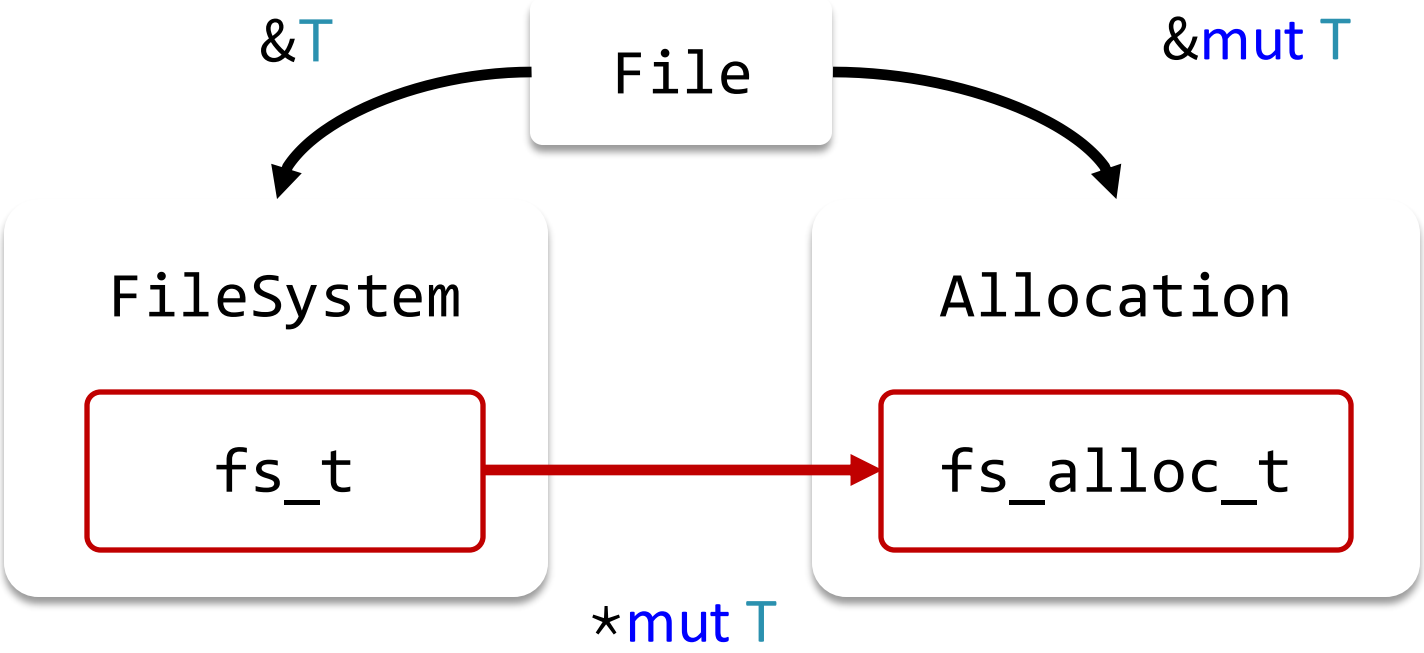
Rust

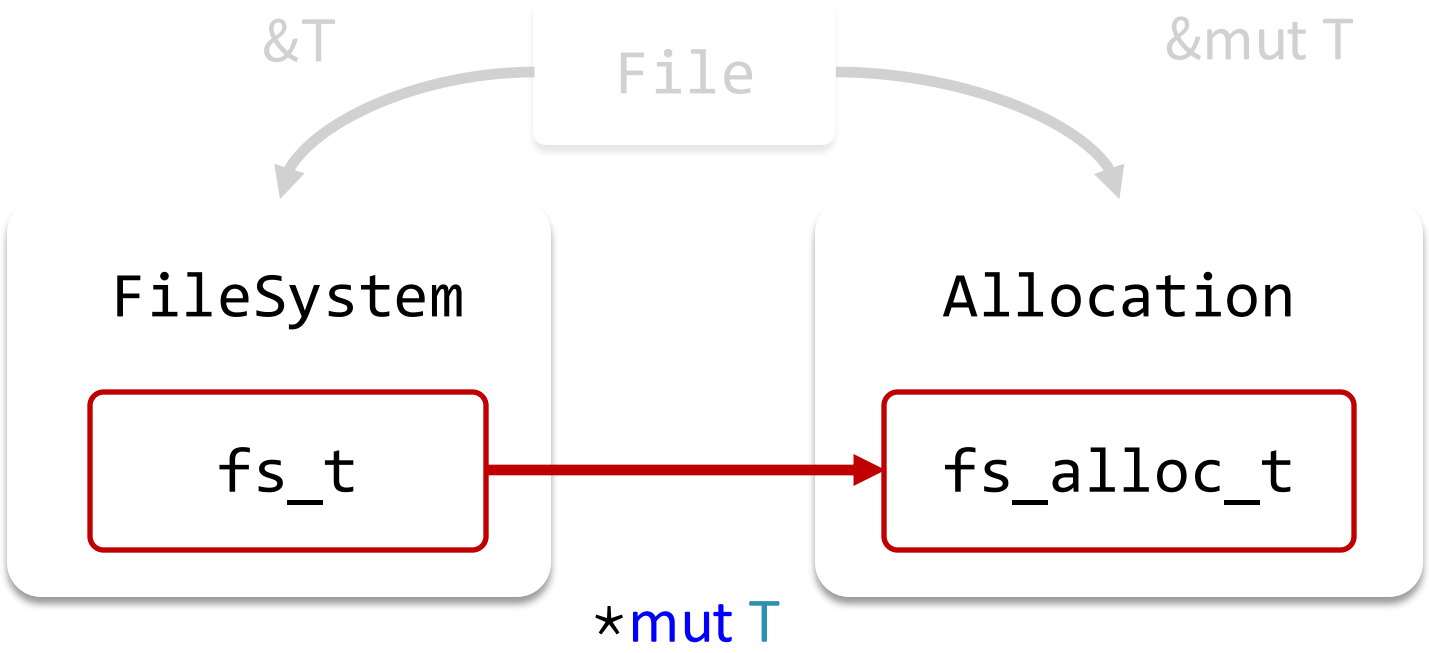
```rust
fn write_buffer(buffer: &mut [u8]) {
    write(&mut *buffer[0], buffer.len());
}
```

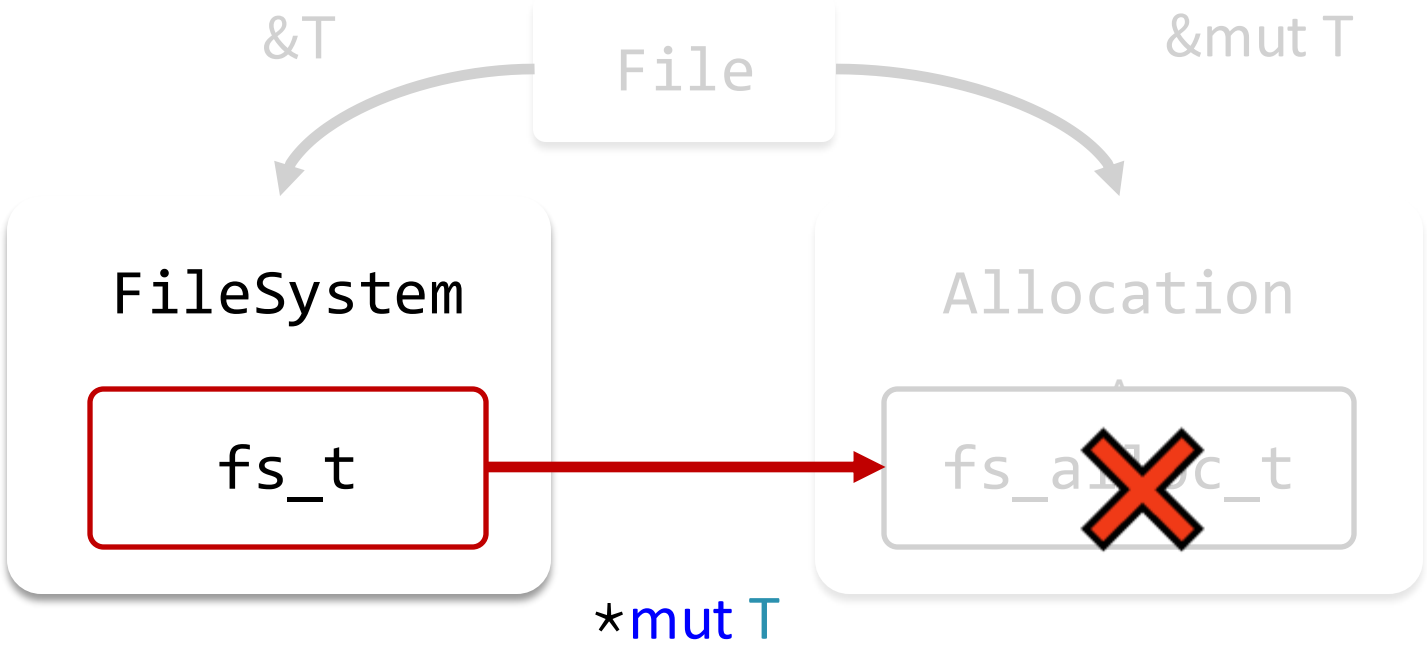
C

```c
void write(uint8_t *buffer, size_t len) {
    *(buffer + len - 1) = 1;
}
```


&T
File
&mut T
FileSystem
Allocation
fs_t
fs_alloc_t
*mut T


&T
File
&mut T
FileSystem
Allocation
fs_t
fs_alloc_t
*mut T


&T
File
&mut T
FileSystem
Allocation
fs_t
fs_alloc_t
*mut T


&T
File
&mut T
FileSystem
fs_t
Allocation
fs_alloc_t
*mut T

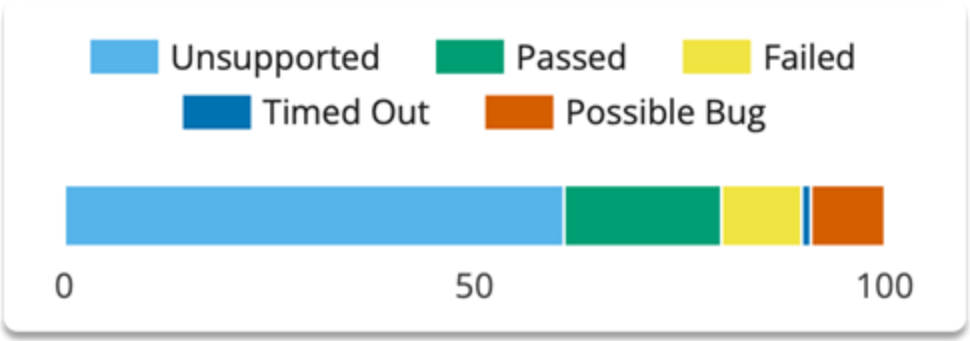
Miri is not enough for large-scale, multi-language applications.

Compatibility

We evaluated MiriLLI on every compatible crate.

There were **9,130** compatible tests from 957 crates.

61% encountered an unsupported operation.



Performance

Anecdotally, Miri is several orders of magnitude slower than native execution

What should a new tool look like?

Fast

Native instrumentation...

C/C++ Support

...through a common format.

Pointer-Level Metadata

Pointer

(Address, Provenance)

$\mathbb{N}$

Provenance

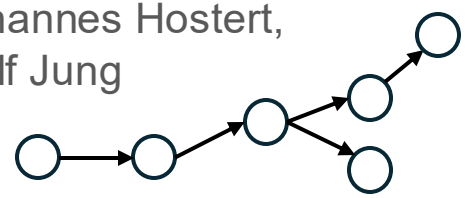
(Allocation ID, Tag)

$\mathbb{N}$ $\mathbb{N}$

Allocation-Level Metadata

Tree Borrows

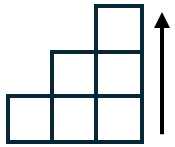
Neven Villani, Johannes Hostert, Derek Dreyer, Ralf Jung



PLDI '25

Stacked Borrows

Ralf Jung, Hong-Hai Dang, Jeehon Kang, and Derek Dreyer



POPL '20

"Identity-Based Access Checking"

SoK: Sanitizing for Security • Song et al., 2019

Valgrind injects instrumentation into compiled programs.

Usable ✅

Fast ✅

C/C++ Support ✅

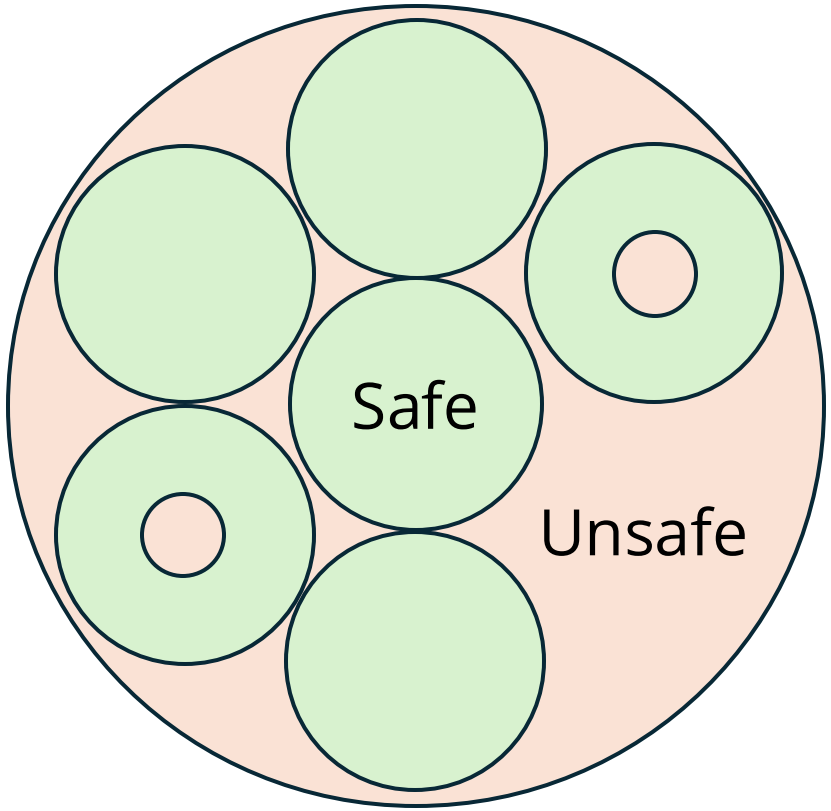
In 2023, the **Krabcake** project proposed extending Valgrind to support detecting Stacked Borrows violations.

Felix Klock, Bryan Garza • AWS

RW2023!

Valgrind's baseline overhead is still **4x.**

Components written in safe Rust *can* be provably **free of undefined behavior**



BorrowSanitizer

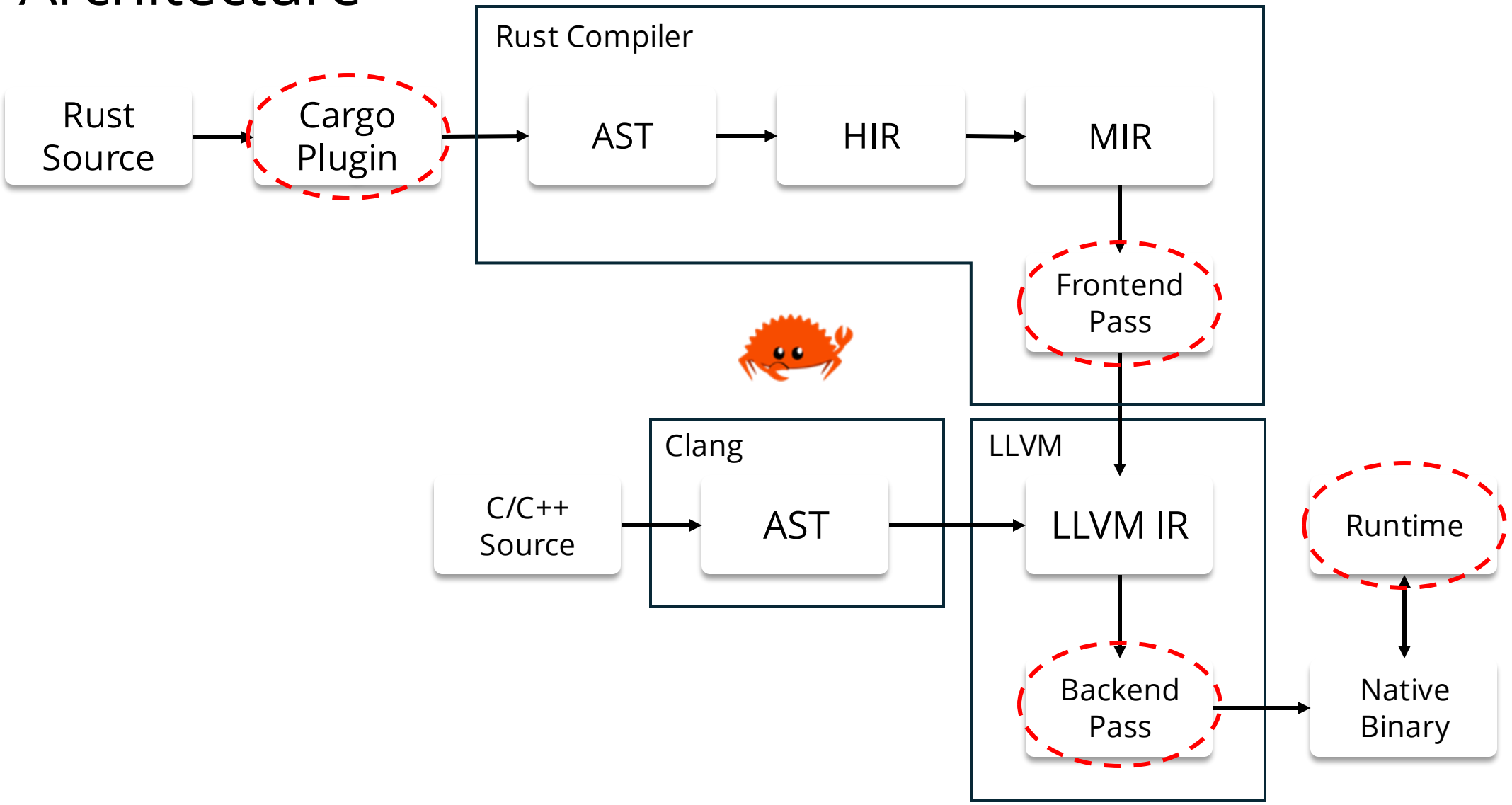
Finding aliasing bugs at-scale

borrowsanitizer.com

An LLVM-based dynamic analysis tool.

- Tree Borrows Violations
- Access out-of-bounds
- Use-after-free

Architecture



Frontend Pass

Inside the Rust Compiler

Our modified LLVM codegen backend inserts “retag” intrinsics.

```
@llvm.retag(ptr, usize, u16, u8, u8)
```

Base Address

Access Size

Permission Type

Protector Kind

Side-effect

Backend Pass

Out-of-Tree LLVM Plugin

Associates each pointer with “provenance”.

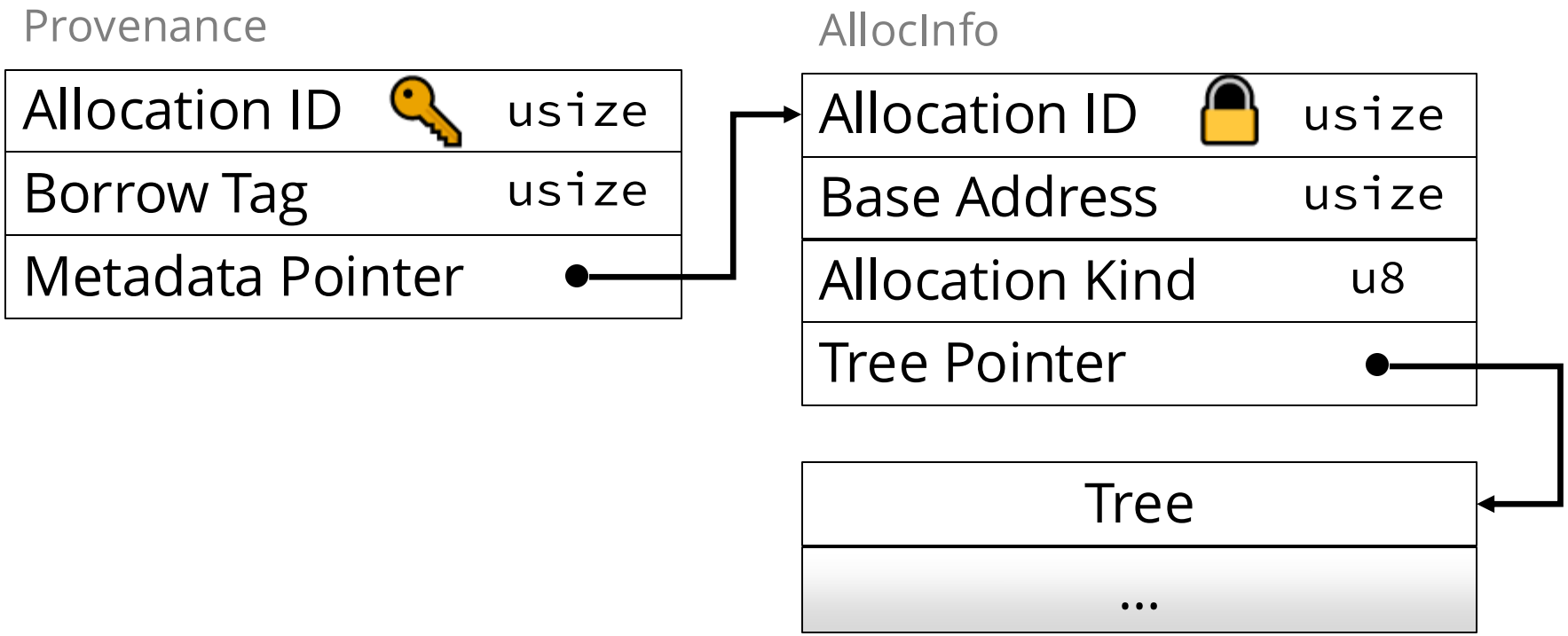
Allocation ID	+	Borrow Tag	+	Metadata Pointer

Uses *Thread-Local Storage* and *Shadow Memory* for storing and propagating provenance across the stack and heap.

Replaces “retag” intrinsics with calls into the runtime, and instruments all memory access operations.

Runtime

Static Rust Library



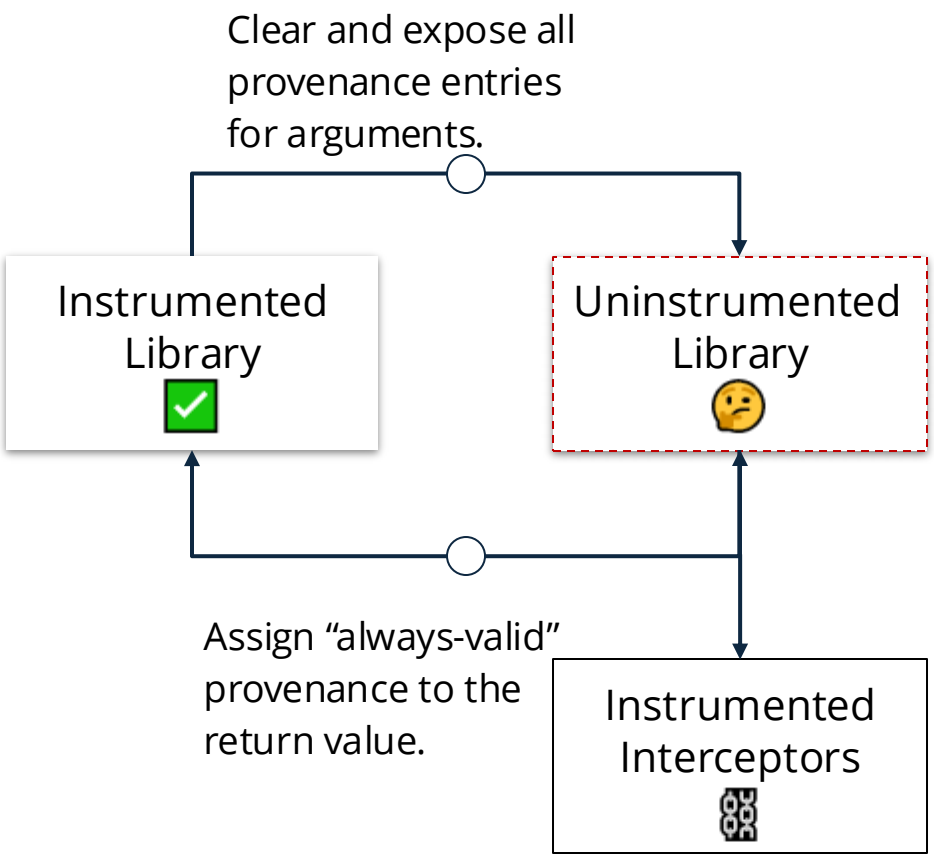
Handling Uninstrumented Libraries

Our default policy will match Miri's behavior for native library calls.

- Expose all provenance entries for pointer arguments.
- Overwrite shadow provenance entries in their underlying allocation with "wildcard" values.

Maintaining metadata integrity requires knowing whether the caller is instrumented.

Can we detect *some* violations in 3rd party code using interception?



Add indirection to shadow memory. Compress the Tree.

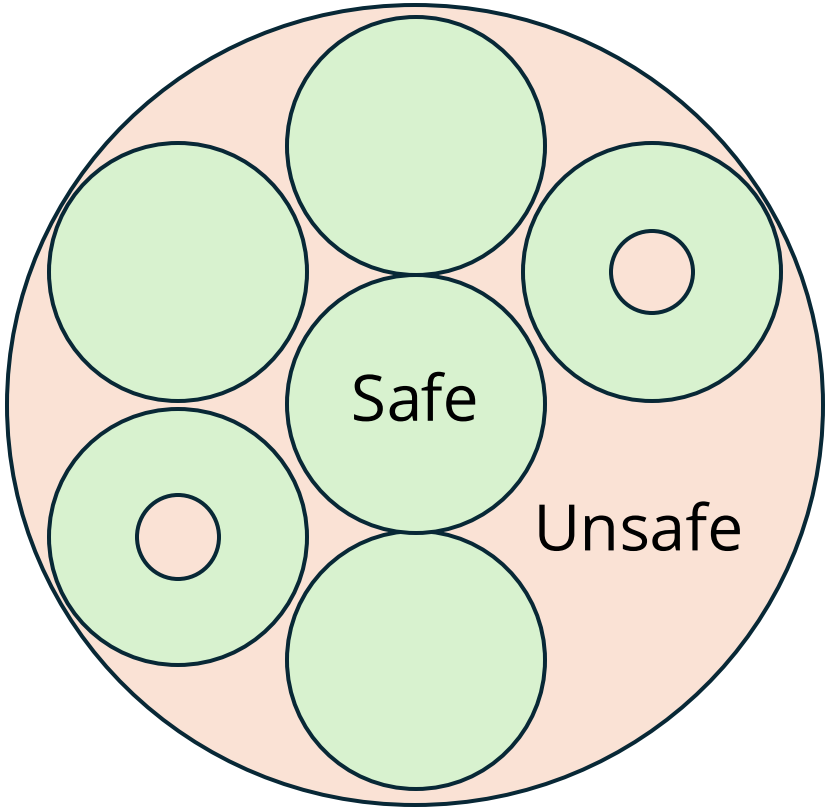
Defer initializing the Tree. LLVM temporaries. Inlining.

Reduce the size of the borrow tag. Only stack allocations.

Reduce the size of the Allocation ID. Tag-check for Frozen.

Tree Borrows is inherently expensive.

Components written in safe Rust *can* be provably **free of undefined behavior**



Borrow *Checking*

`&T`

Shared, Read-only

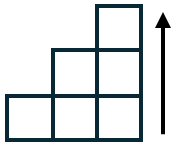
`&mut T`

Unique, Write

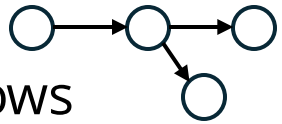
Static

Borrow *Tracking*

Stacked Borrows



Tree Borrows

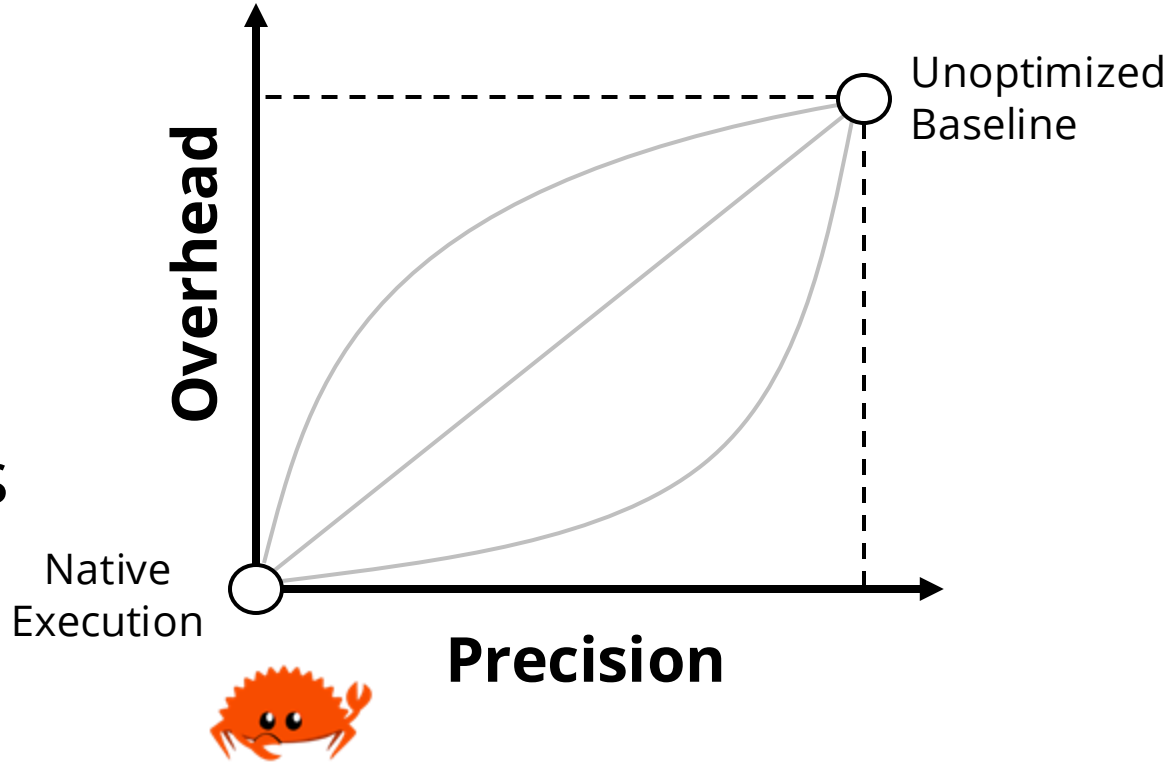


Dynamic

Gradual Typing

`*mut ? T`

We need to find a compromise between the ***eager*** **static** semantics of Polonius and the ***lazy*** **dynamic** semantics of Tree Borrows.



0.1.0

Phase 1

September 2025

1.0.0

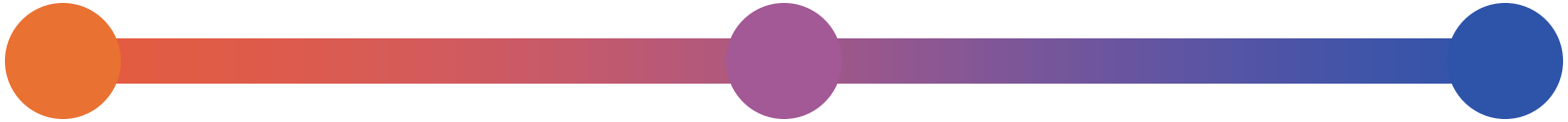
Phase 2

December 2025

1.1.0

Phase 3

September 2026



Single-threaded
+
Static Proof Sketch

Multi-threaded
+
Dynamic Optimizations

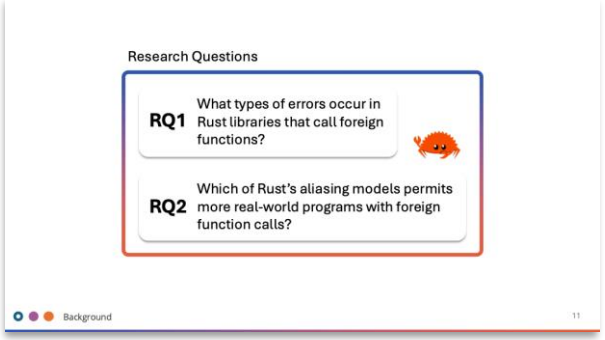
Static Optimizations

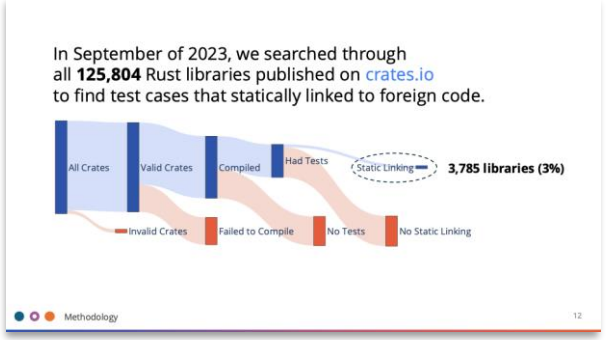
A Study of Undefined Behavior Across Foreign Function Boundaries in Rust Libraries

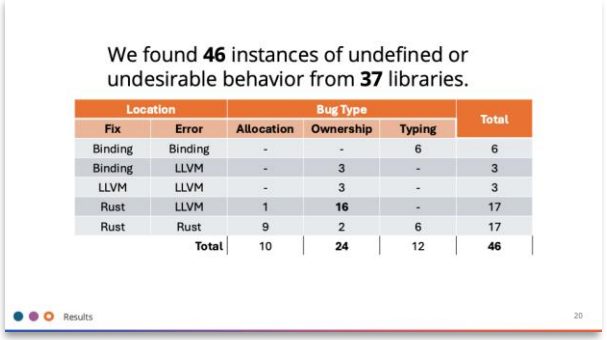
Ian McCormack
Carnegie Mellon University

Joshua Sunshine
Carnegie Mellon University

Jonathan Aldrich
Carnegie Mellon University







Location		Bug Type			Total
Fix	Error	Allocation	Ownership	Typing	
Binding	Binding	-	-	6	6
Binding	LLVM	-	3	-	3
LLVM	LLVM	-	3	-	3
Rust	LLVM	1	16	-	17
Rust	Rust	9	2	6	17
	Total	10	24	12	46

Preprint

Artifact

S3D Software and Societal Systems Department

This material is based on work supported by the Department of Defense and the National Science Foundation under Grant Nos. CCF-1901033, DGE1745016, and DGE2140739. Our results were obtained using CloudBank, which is supported by the National Science Foundation under award #1925001.

BorrowSanitizer

Finding aliasing bugs at-scale

borrowsanitizer.com