BorrowSanitizer

Finding Ownership Bugs in Multilanguage Rust Applications

Ian McCormack

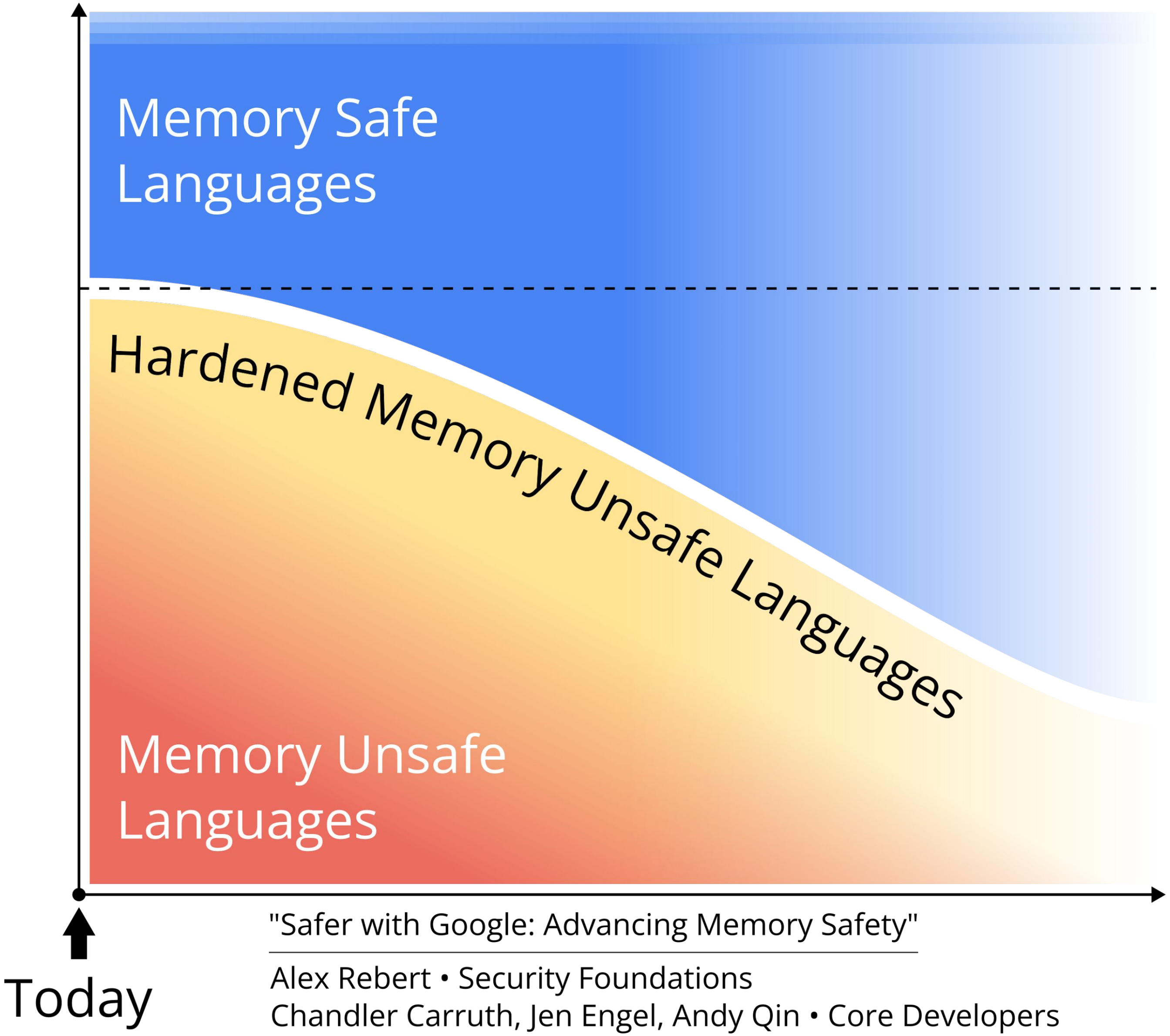
Carnegie Mellon University

Software & Societal Systems

Background & Motivation

Design Principles

Future Work

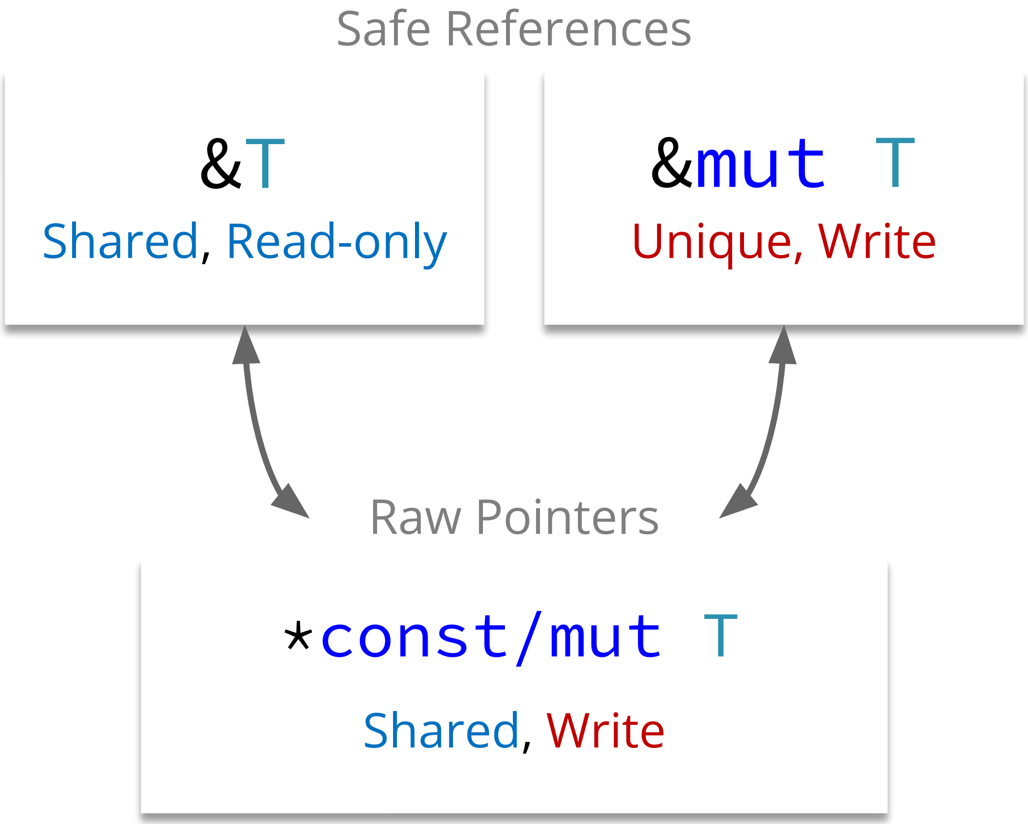


"Safer with Google: Advancing Memory Safety"

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

Rust restricts aliasing to provide static safety guarantees…

…but developers need to bypass these restrictions.



Safe References

&T

Shared, Read-only

&mut T

Unique, Write

Raw Pointers

*const/mut T

Shared, Write

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

Calling `unsafe` functions

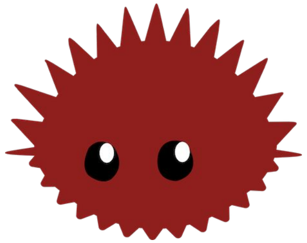
Intrinsics & inline assembly

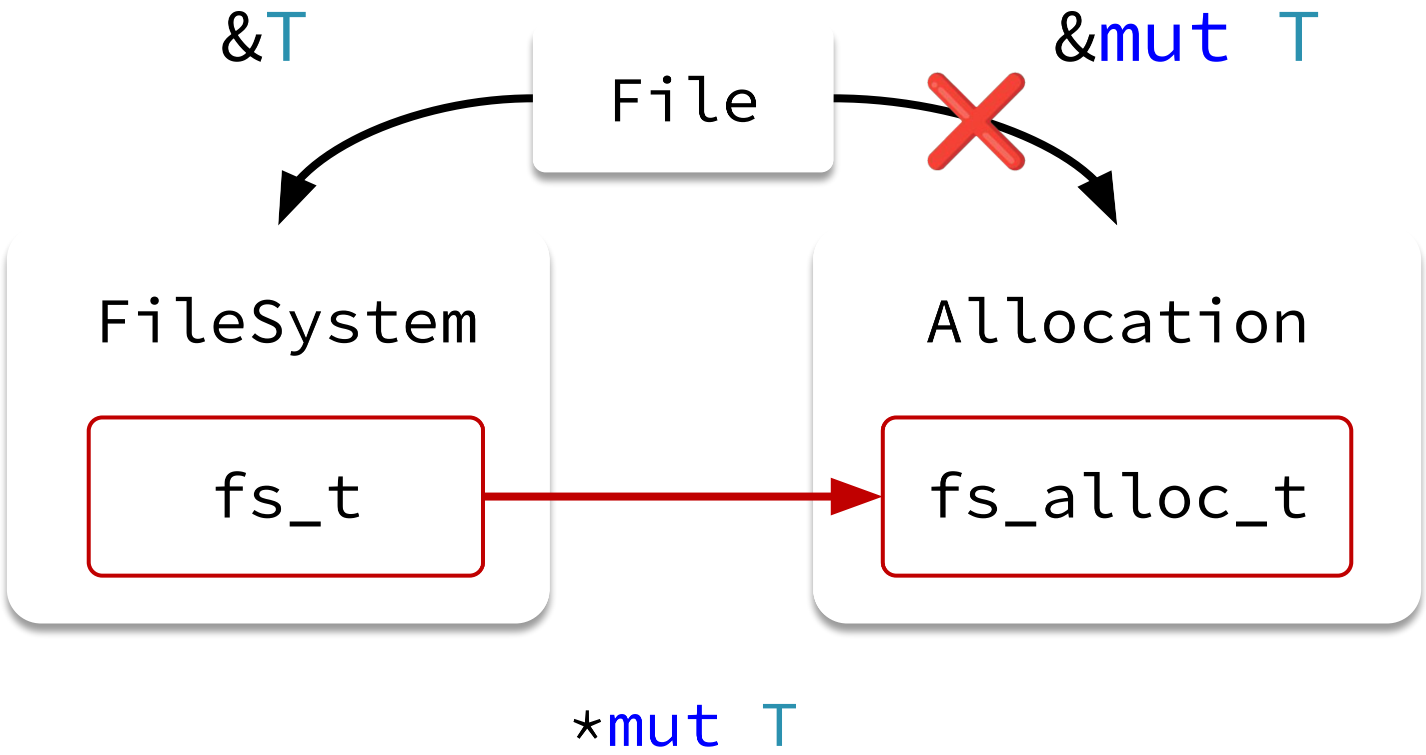
Manipulating uninitialized memory

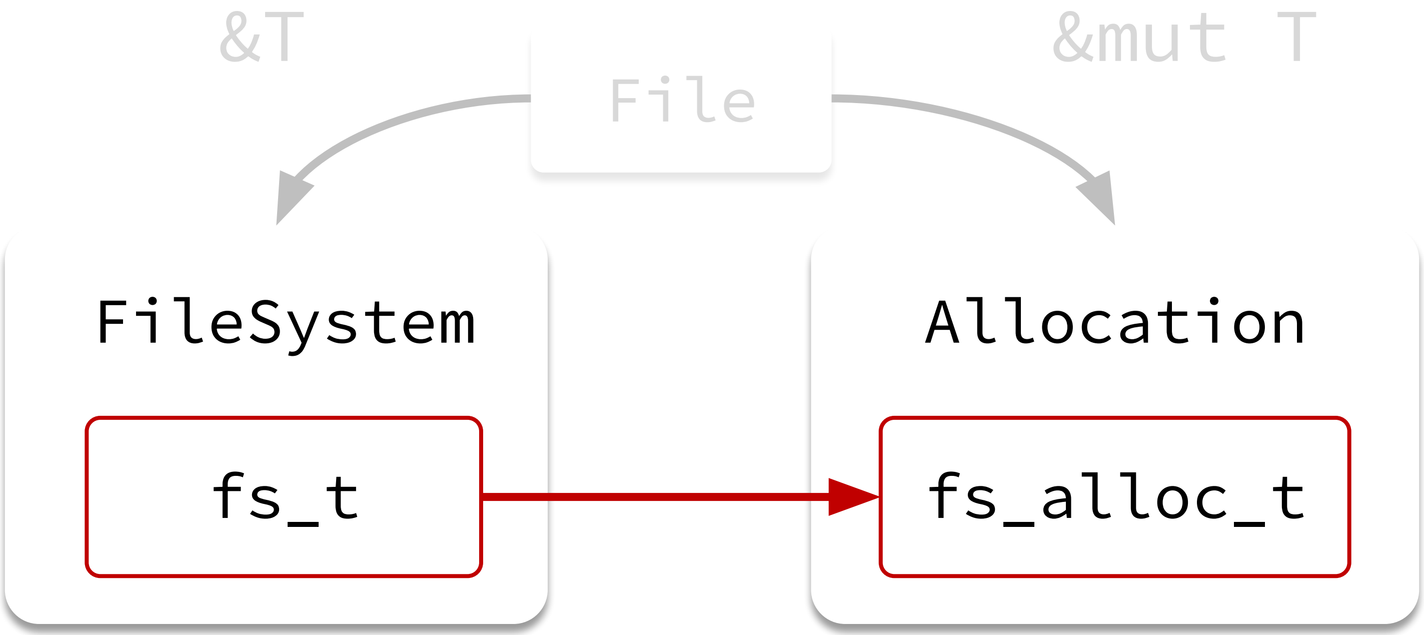
Accessing global, mutable state

Dereferencing raw pointers

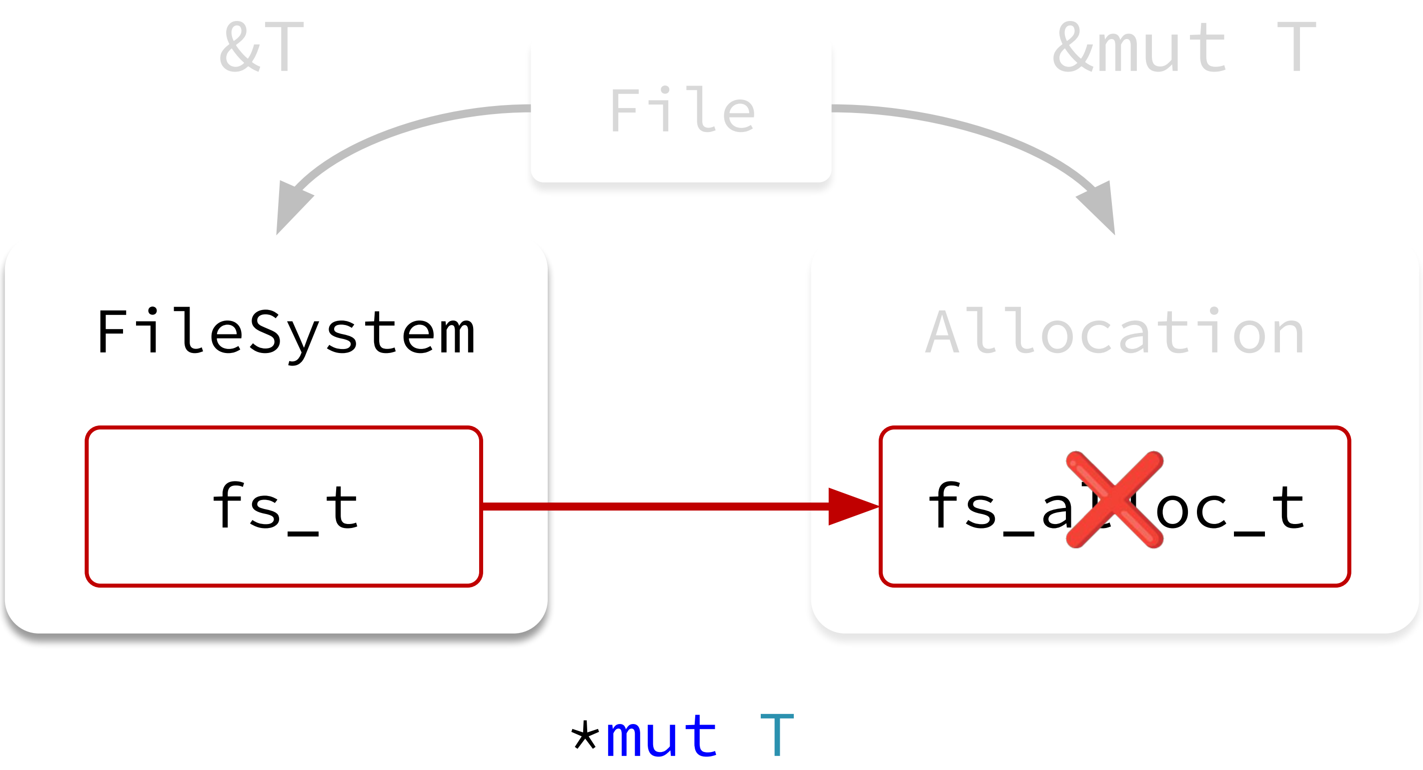
Implementing an `unsafe` trait



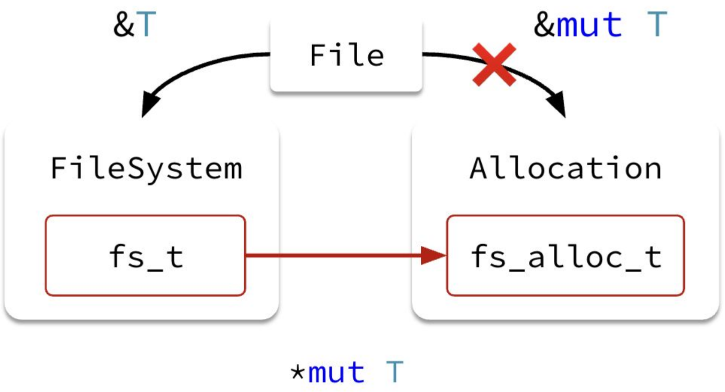

&T
File
&mut T
❌
FileSystem
fs_t
Allocation
fs_alloc_t
*mut T



*mut T


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

Aliasing violations are **both** a form of undefined behavior **and** an indication that other safety errors might exist.

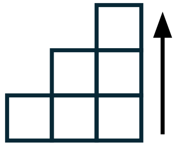


Miri, a Rust interpreter, can find these aliasing bugs

Stacked Borrows

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

POPL ‘20

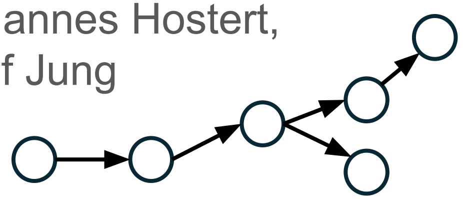


OR

Tree Borrows

Neven Villani, Johannes Hostert,
Derek Dreyer, Ralf Jung

PLDI ‘25



Bounds Checking

Liveness Checking

Data Race Detection

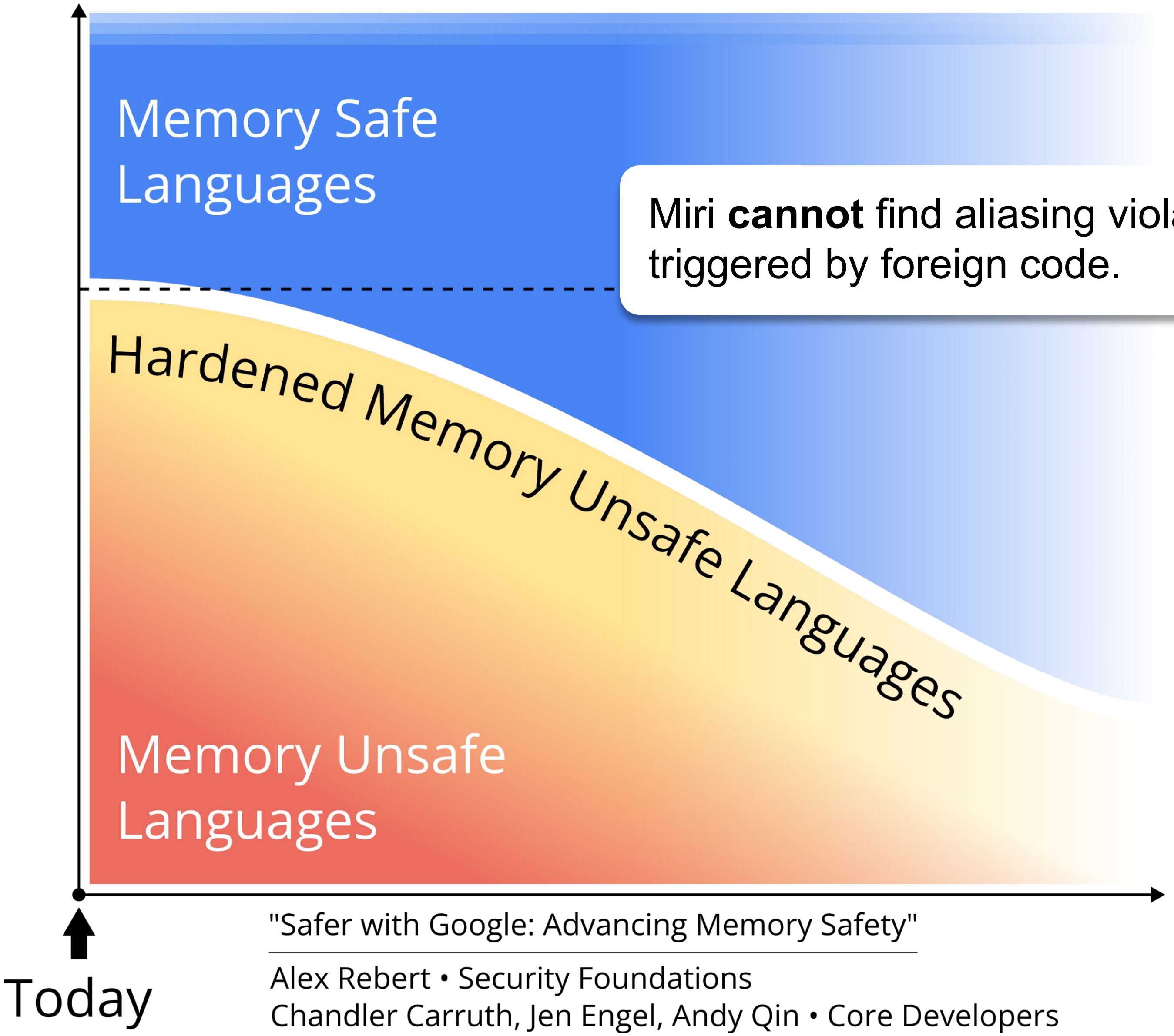
Pointer

$(\underset{\mathbb{N}}{\text{Address}}, \text{Provenance})$

→

Provenance

$(\underset{\mathbb{N}}{\text{Allocation ID}}, \underset{\mathbb{N}}{\text{Tag}})$



Miri **cannot** find aliasing violations triggered by foreign code.

"Safer with Google: Advancing Memory Safety"

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

Are aliasing violations hiding, undetected, in multilanguage Rust programs?

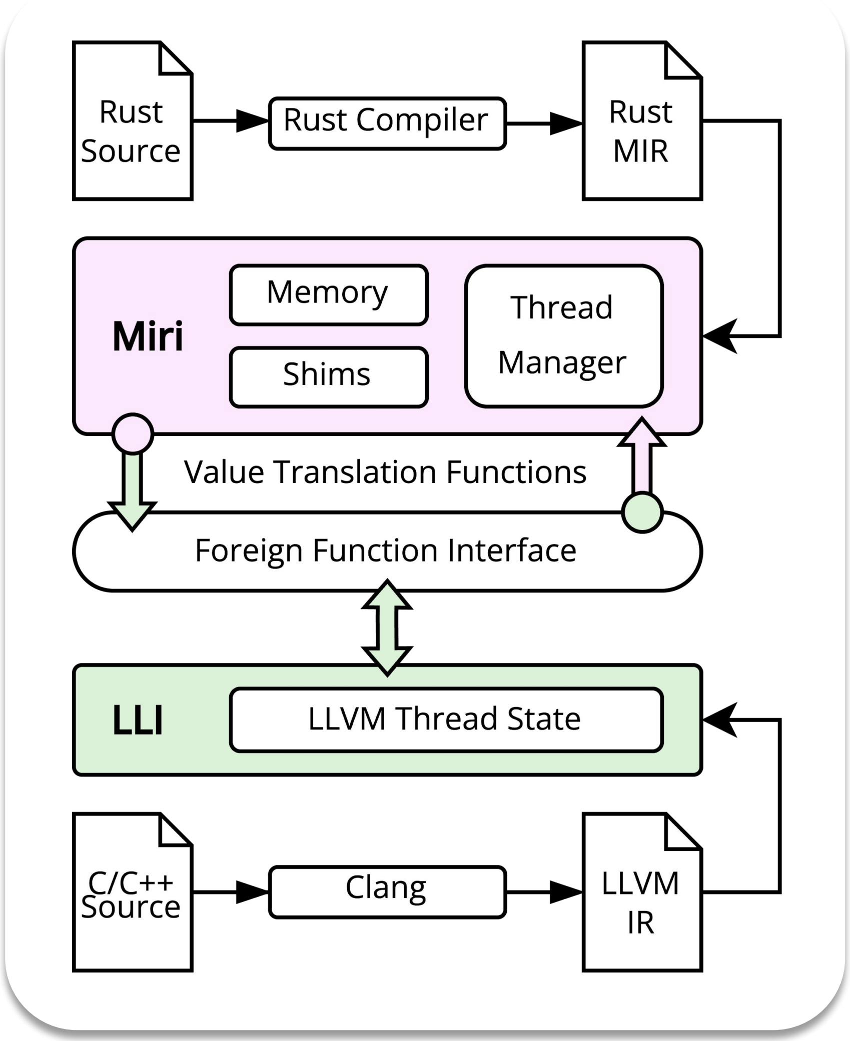
A Study of Undefined Behavior Across Foreign Function Boundaries in Rust Libraries

Ian McCormack, Jonathan Aldrich, Joshua Sunshine

ICSE ‘25

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**.



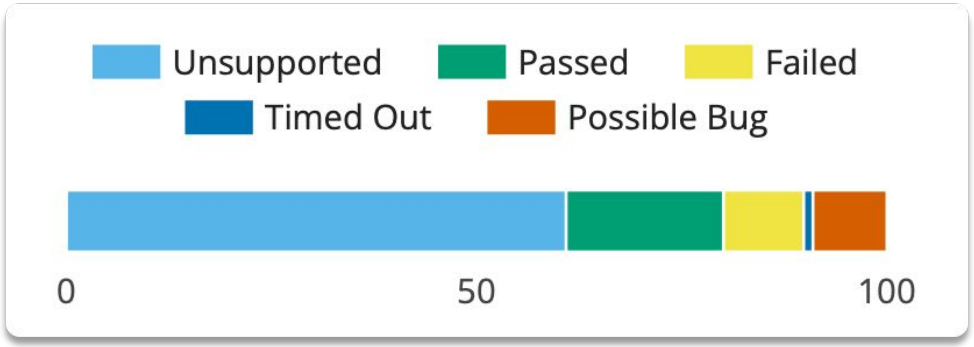
Miri is not enough for large-scale, multi-language applications.

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?

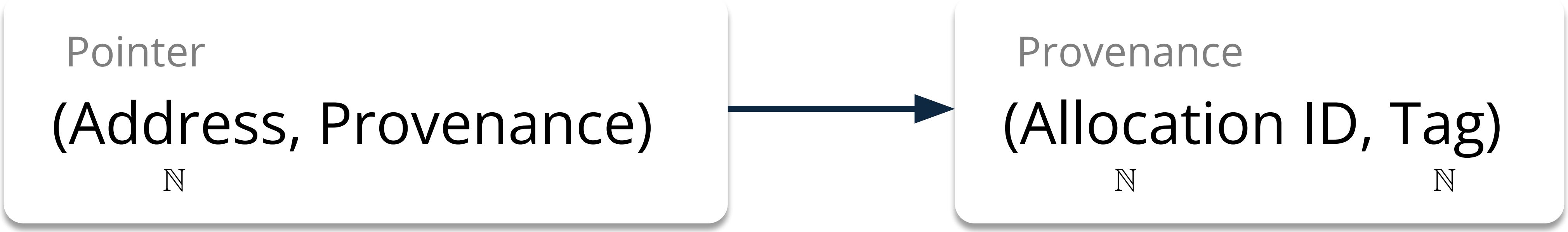
Fast

Native instrumentation…

C/C++ Support

…through a common format.

Pointer-Level Metadata



Pointer

(Address, Provenance)

$\mathbb{N}$

Provenance

(Allocation ID, Tag)

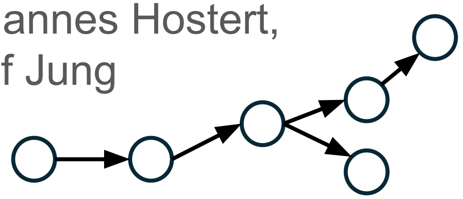
$\mathbb{N}$ $\mathbb{N}$

Allocation-Level Metadata

Tree Borrows

Neven Villani, Johannes Hostert,
Derek Dreyer, Ralf Jung

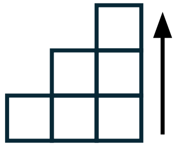
PLDI '25



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 ✅

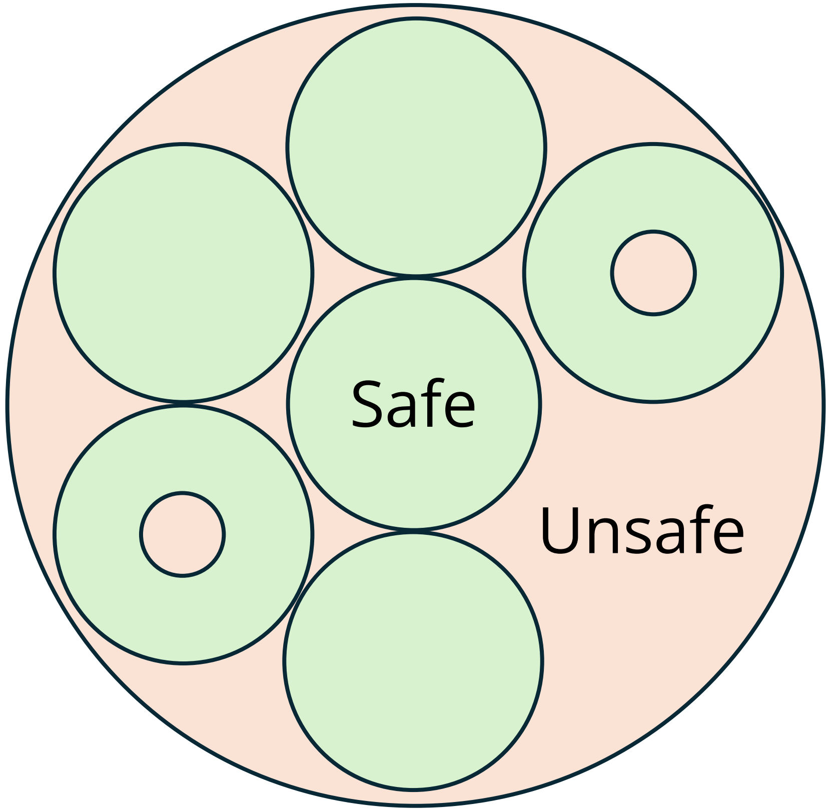
In 2023, the **Krabcake** project proposed extending Valgrind to support detecting Stacked Borrows violations.

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.

- 🐞 Aliasing Violations
- 🐜 Accesses out-of-bounds
- 🐛 Use-after-free

Our Team

Ian McCormack
Carnegie Mellon University

Rafayel Amirkhanyan
University of Pittsburgh

Krit Dass
Carnegie Mellon University

Jonathan Aldrich
Carnegie Mellon University

Molly MacLaren
Carnegie Mellon University

Oliver Braunsdorf
Ludwig Maximilian University

Johannes Kinder
Ludwig Maximilian University

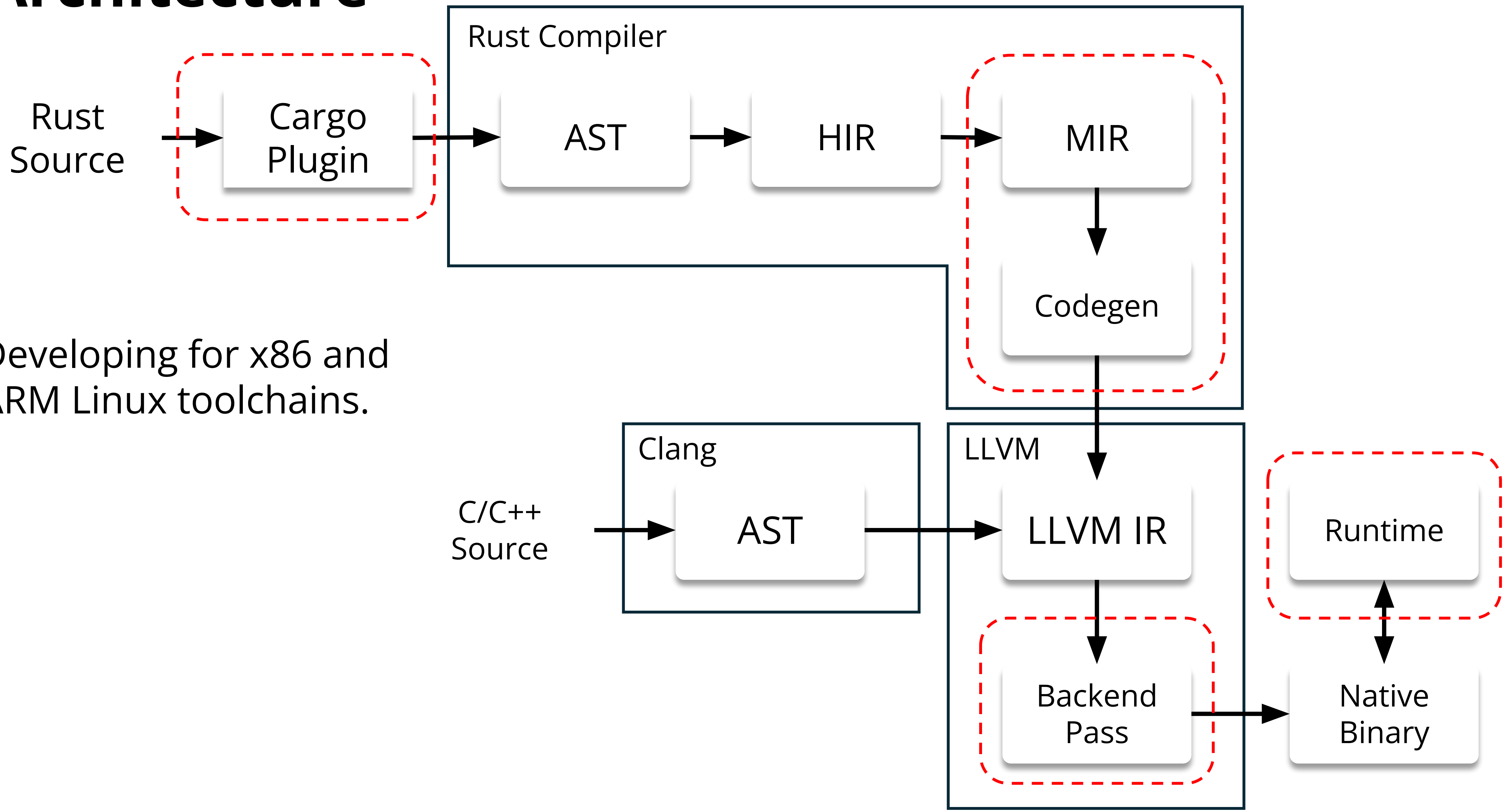
Joshua Sunshine
Carnegie Mellon University

Funded by

Rust Foundation

+

Architecture



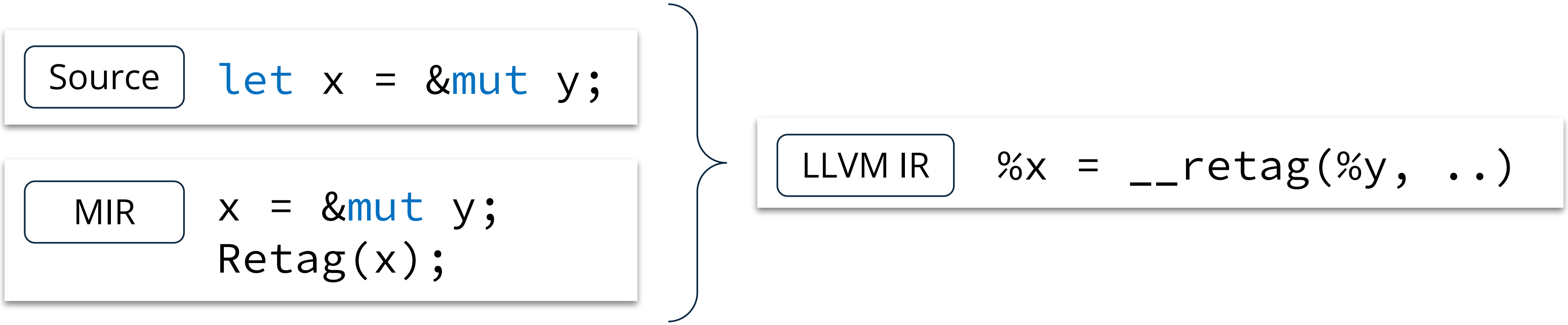
Developing for x86 and ARM Linux toolchains.

Frontend

Inside the Rust Compiler

Today, only some retags are explicit MIR statements.

Others are added implicitly when Miri interprets assignments.



Source

```
let x = &mut y;
```

MIR

```
x = &mut y;
Retag(x);
```

LLVM IR

```
%x = __retag(%y, ..)
```

Our modified compiler emits all retags as explicit statements.

Frontend

Inside the Rust Compiler

MIR Retags are “coarse-grained” and apply to entire places.

```
Retag(RetagKind, Box<Place<'tcx>>)
```

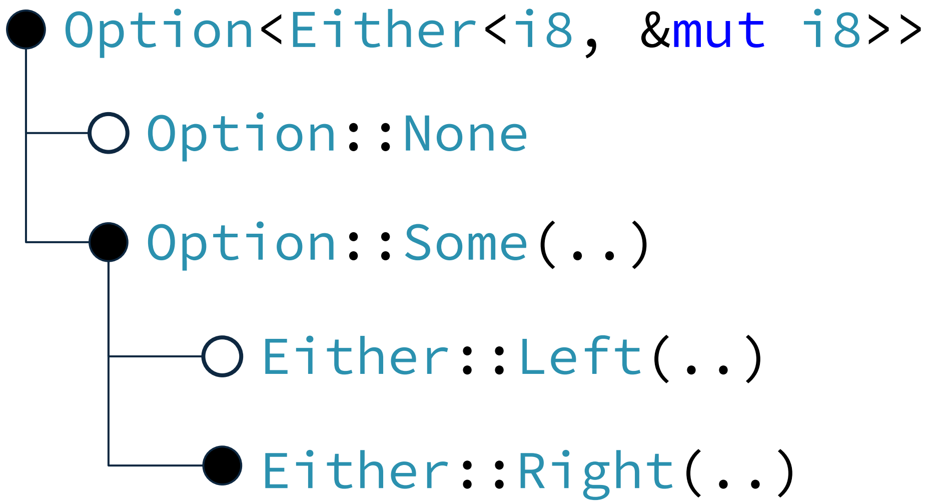
Frontend

Inside the Rust Compiler

MIR Retags are “coarse-grained” and apply to entire places.

```
Retag(RetagKind, Box<Place<'tcx>>)
```

ADTs containing references may need to be conditionally retagged.



Frontend

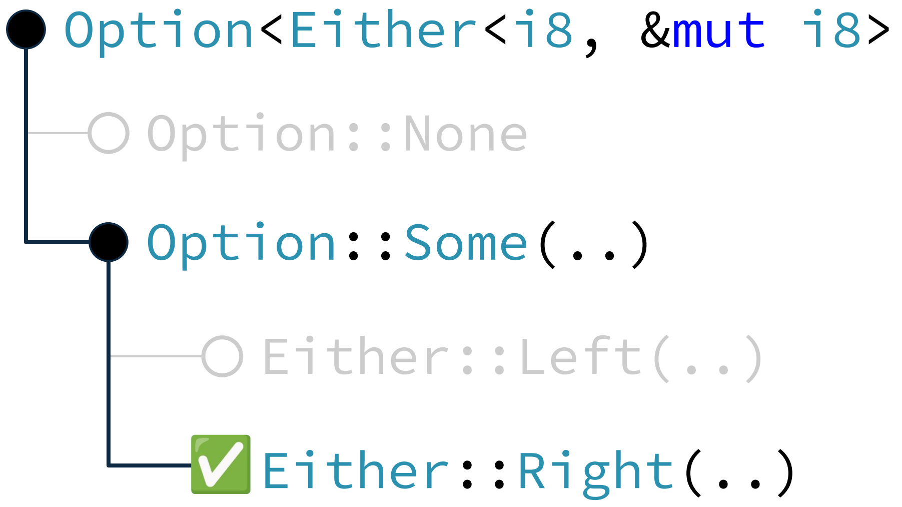
Inside the Rust Compiler

MIR Retags are “coarse-grained” and apply to entire places.

ADTs containing references may need to be conditionally retagged.

We create a “retag plan” based on the structure of each type.

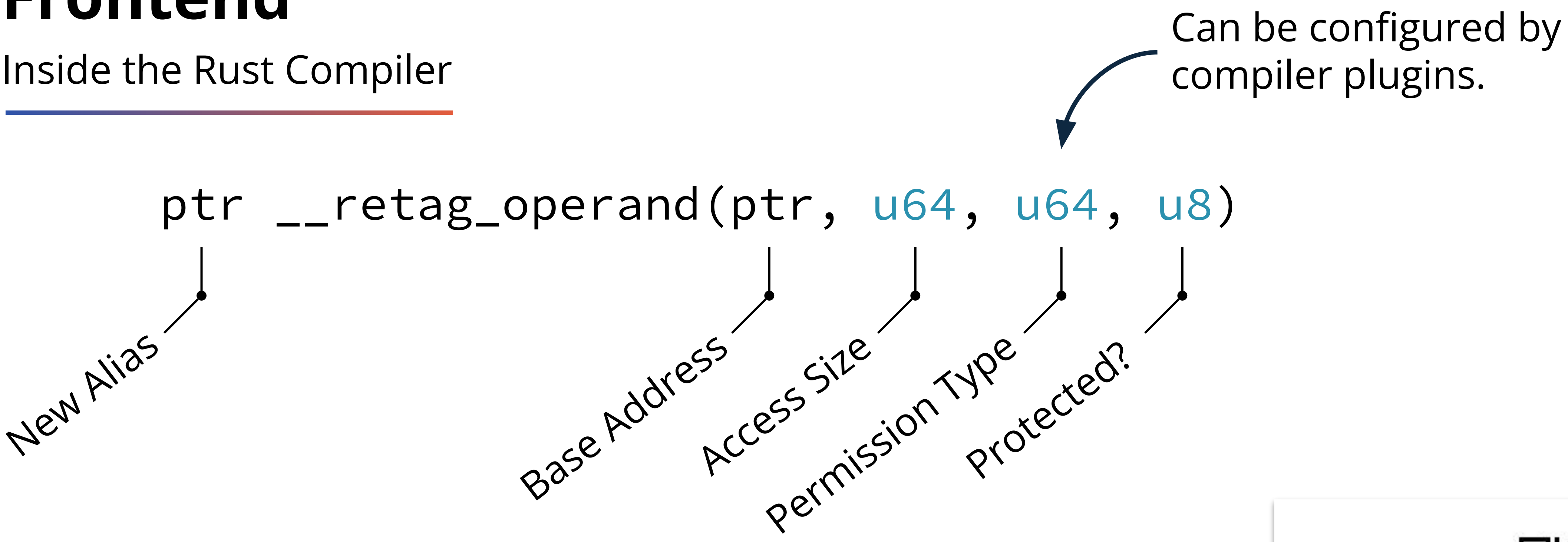
```
Retag(RetagKind, Box<Place<'tcx>>)
```



```
● Option<Either<i8, &mut i8>
├─○ Option::None
└─● Option::Some(..)
    ├─○ Either::Left(..)
    └─✓ Either::Right(..)
```

Frontend

Inside the Rust Compiler



Can be configured by compiler plugins.

```
ptr __retag_operand(ptr, u64, u64, u8)
```

All parameters are standard between aliasing models except for the “permission type”.

Status updates:

Backend Pass

Out-of-Tree LLVM Plugin

Associates each pointer with “provenance”.

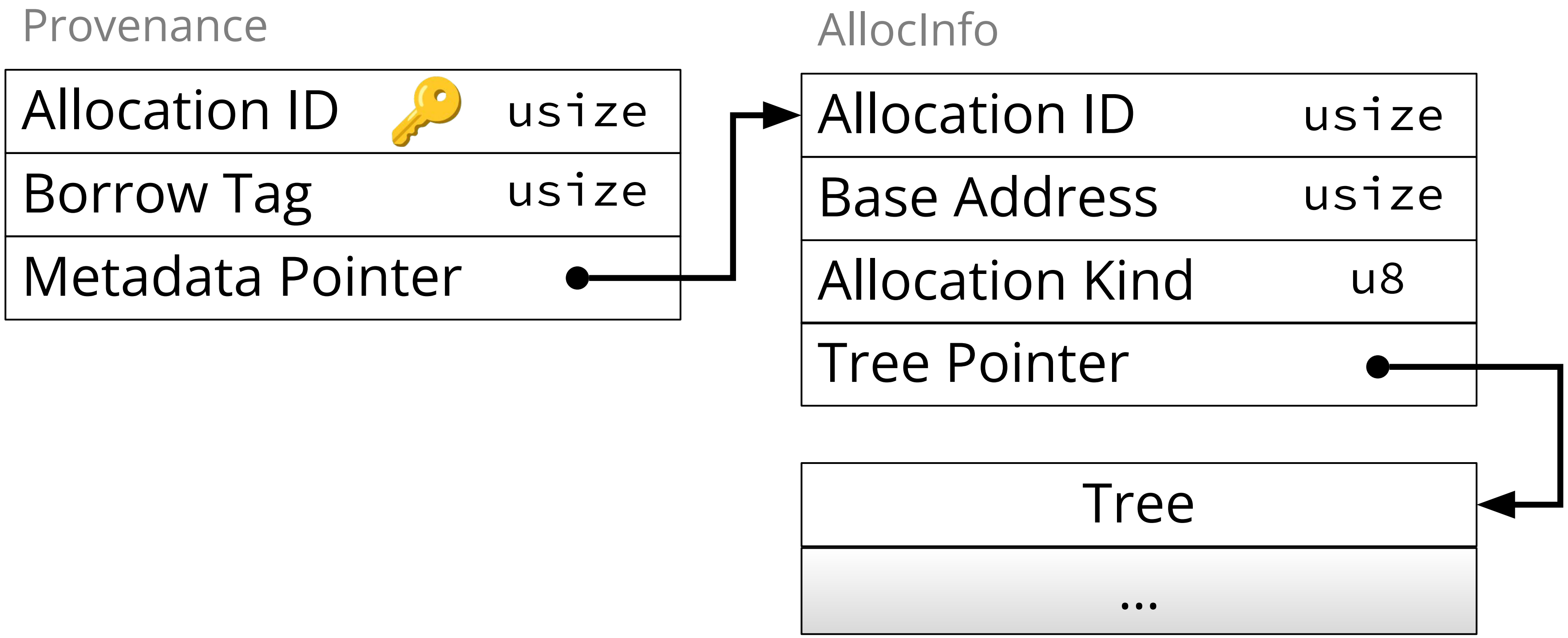
Allocation ID + Borrow Tag + Metadata Pointer

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



Provenance

Allocation ID 🔑	usize
Borrow Tag	usize
Metadata Pointer	●

AllocInfo

Allocation ID	usize
Base Address	usize
Allocation Kind	u8
Tree Pointer	●

Tree
...

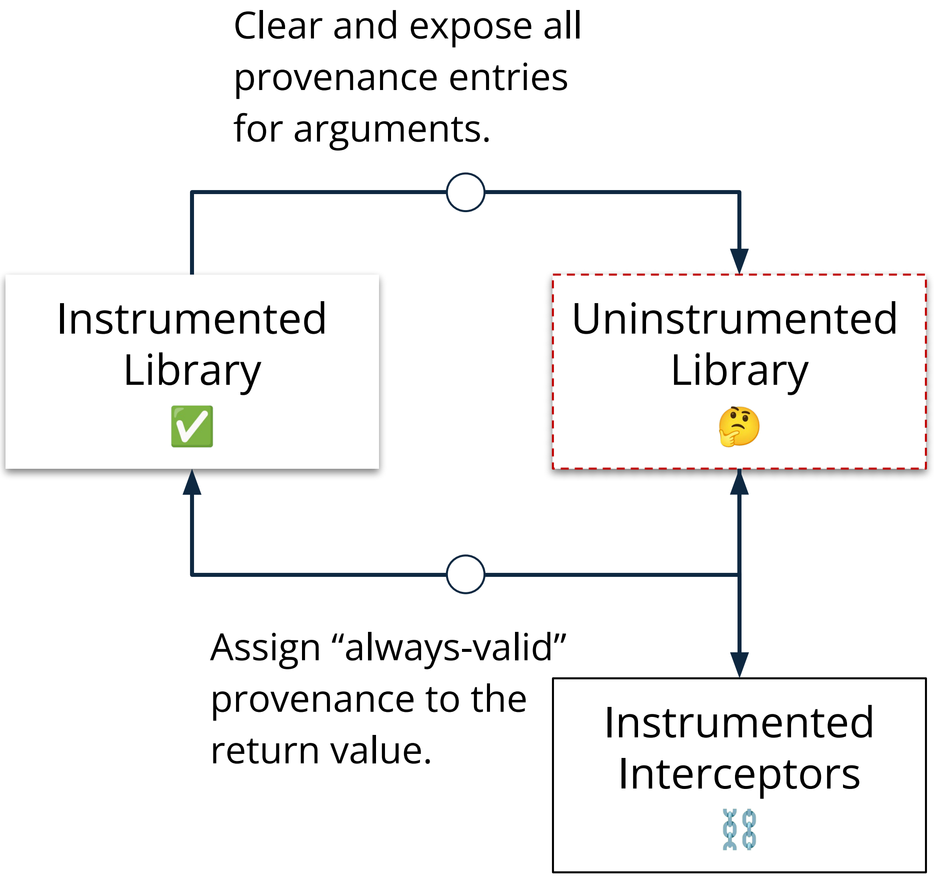
Runtime

Static Rust Library

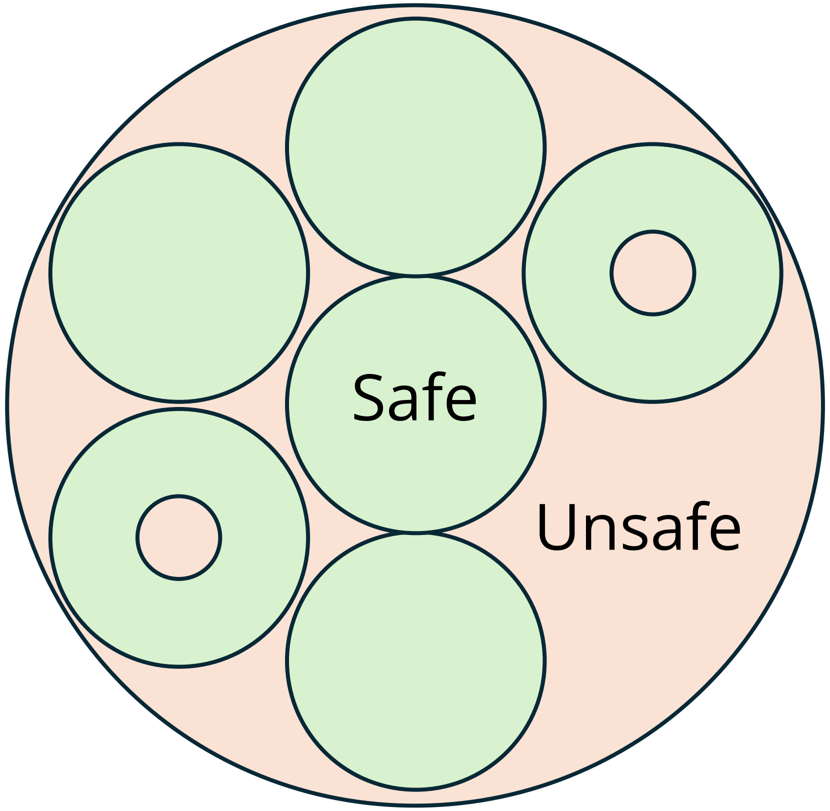
We will match Miri's behavior for uninstrumented function calls.

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

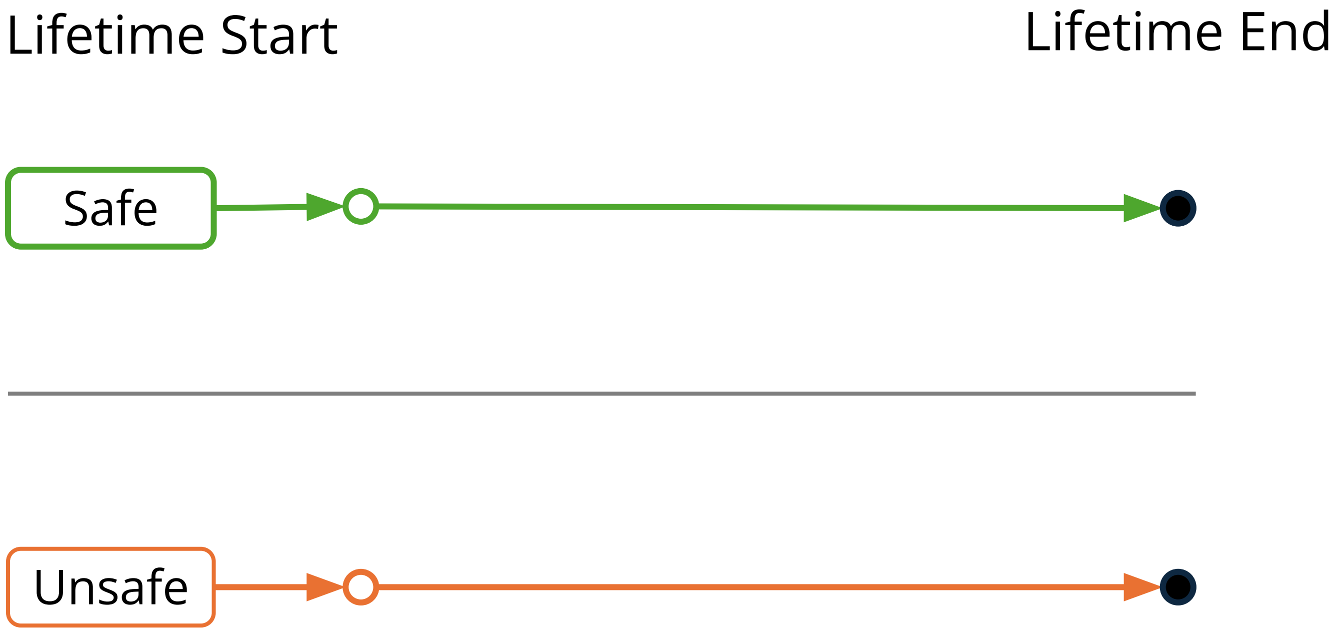
Maintaining metadata integrity requires knowing whether the caller is instrumented.


Clear and expose all provenance entries for arguments.
Instrumented Library ✅
Uninstrumented Library 🤔
Assign "always-valid" provenance to the return value.
Instrumented Interceptors ⛓️

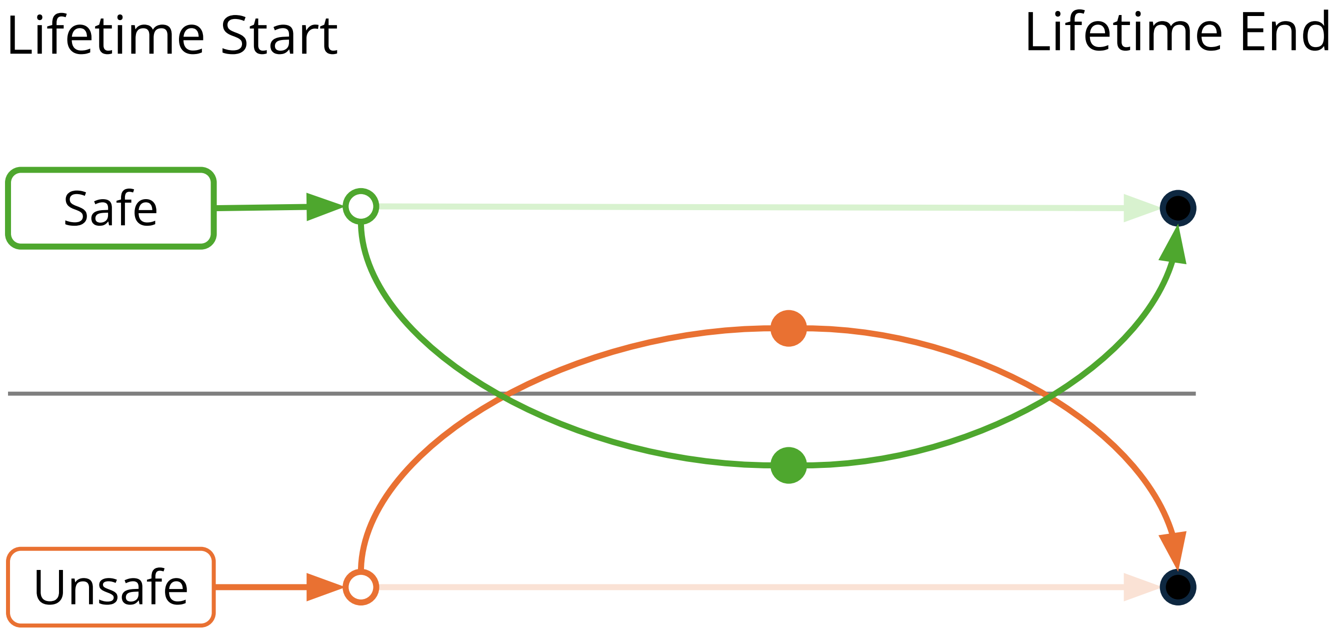
Components written in safe Rust *can* be provably **free of undefined behavior**



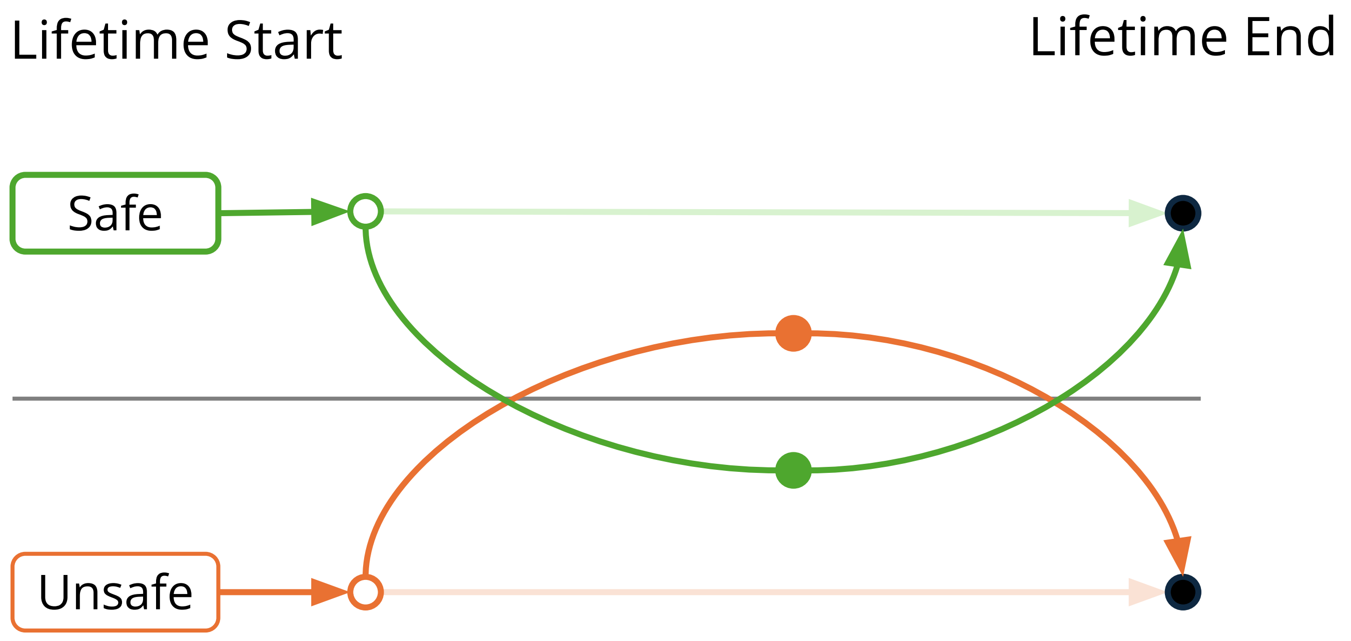
We only need to instrument allocations that are “tainted” by both safe and unsafe contexts.



We only need to instrument allocations that are “tainted” by both safe and unsafe contexts.

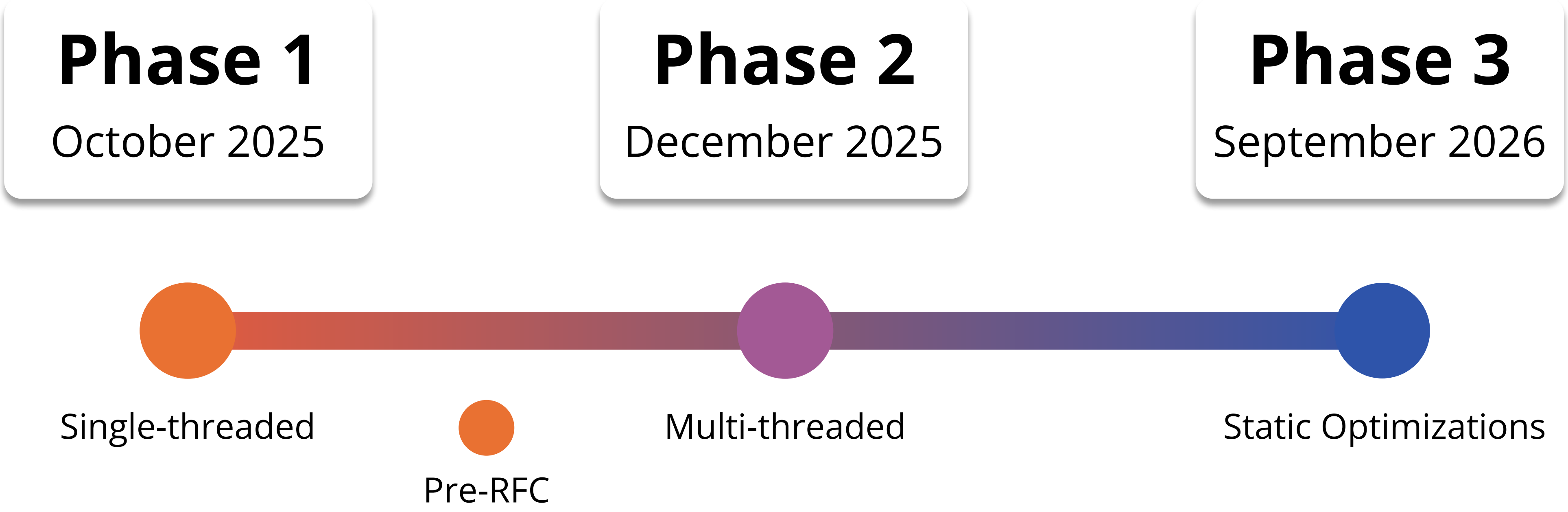


We only need to instrument allocations that are “tainted” by both safe and unsafe contexts.



LiteRSan:
Lightweight Memory Safety Via Rust-specific Program Analysis and Selective Instrumentation

Xia et al.



Phase 1

October 2025

Single-threaded

Pre-RFC

Phase 2

December 2025

Multi-threaded

Phase 3

September 2026

Static Optimizations

BorrowSanitizer

Finding Ownership Bugs in Multilanguage Rust Applications

Project Site
borrowsanitizer.com

Ian McCormack
Carnegie Mellon University

Molly MacLaren
Carnegie Mellon University

Rafayel Amirkhanyan
University of Pittsburgh

Oliver Braunsdorf
Ludwig Maximilian University

Krit Dass
Carnegie Mellon University

Johannes Kinder
Ludwig Maximilian University

Jonathan Aldrich
Carnegie Mellon University

Joshua Sunshine
Carnegie Mellon University

Rust Foundation