
a
b
c
d
e
f
g
h
i
j
k

# Carcinization

NOT crabs ➡️

# Carcinization

**Carcinisation** is a form of convergent evolution in which non-crab crustaceans evolve a crab-like body plan.

NOT crabs ➡️

# Carcinization

**Carcinisation** is a form of convergent evolution in which non-crab crustaceans evolve a crab-like body plan.

NOT crabs ➡️

# Carcinization

**Carcinisation** is a form of convergent evolution in which non-crab crustaceans evolve a crab-like body plan.

The term was introduced into evolutionary biology by L.A. Borradaile, who described it as:

NOT crabs ➡️

# Carcinization

**Carcinisation** is a form of convergent evolution in which non-crab crustaceans evolve a crab-like body plan.

The term was introduced into evolutionary biology by L.A. Borradaile, who described it as:

*"the many attempts of Nature to evolve a crab"*

NOT crabs ➡️

# Who thinks interop is about... C FFI

# Who thinks interop is about...

C FFI

glue code

# Who thinks interop is about...

C FFI

glue code

coge generators

# Who thinks interop is about...

C FFI

glue code

coge generators

(fat) compilers

# Who thinks interop is about...

C FFI

glue code

coge generators

(fat) compilers

linkers

# Who thinks interop is about...

C FFI

glue code

coge generators

(fat) compilers

linkers

ABI compat

# When you ask about Rust interop

C ffi, autocxx, bindgen, cbindgen, diplomat,

capigen, cxx, zngur, crubit, wit-bindgen, etc.

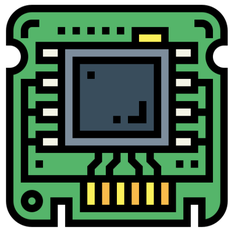
# When you ask about Rust interop

C ffi, autocxx, bindgen, cbindgen, diplomat,

capigen, cxx, zngur, crubit, wit-bindgen, etc.

glue code
source annotations
code generators
IDLs
mapping language semantics & constructs

# Rust extreme range of operation




# Rust @Microsoft

- Project Mu
- Pluton security processor
- SymCrypt - rustls
- Azure Integrated HSM
- Azure Boost Agents
- Open VMM / Open HCL
- Hyper-V

- Azure SDK for Rust
- Azure Data Explorer
- Drasi
- MIMIR
- Caliptra
- Hyperlight / WASM
- ... 🤫

TBD:

⚙️ Windows core components

🎛️ Drivers

☁️ Microservices (Azure, M365)

# Oxidation

## More oxidation 🦀 efforts in progress...

## C++ ➡️ Rust ⬅️ C#

- Assist developers making the *transition* from C, C++, C# to Rust
- Investing in Rust developer tooling

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust
- Costs of writing new Rust components

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust
- Costs of writing new Rust components
- Is the full pipeline of Rust tooling ready?

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust
- Costs of writing new Rust components
- Is the full pipeline of Rust tooling ready?
- Debugging hybrid binaries

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust
- Costs of writing new Rust components
- Is the full pipeline of Rust tooling ready?
- Debugging hybrid binaries
- Performance targets, x-language LTO

# Rust in Production

**Learn** by doing: **Exploration → Flighting → Production**

- Direct impact: improve security & reduce operation cost
- Gain experience with transitioning to Rust in production
- Costs of porting to Rust
- Costs of writing new Rust components
- Is the full pipeline of Rust tooling ready?
- Debugging hybrid binaries
- Performance targets, x-language LTO
- Costs of maintaining a hybrid C++/Rust codebase

# Rust ❤️ C++

They need to play nice together... for a looong time!

# Rust / C++ interoperability

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)
- Plays well with C++ ABI

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)
- Plays well with C++ ABI
- Easily automated

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)
- Plays well with C++ ABI
- Easily automated
- Debuggable

Choose... ~~none~~ some?

# Rust / C++ interoperability

- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad types support - with safety
- Avoid lowering through C FFI
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)
- Plays well with C++ ABI
- Easily automated
- Debuggable
- Hybrid build systems (CMake, cargo, MSBuild, bazel, buck...)

Compiler

Debugger

Linker


ABI guarantees

Interop Library

Build system

RustWeek

May 2025


# Duck-Tape Chronicles
## Rust/C++ Interop

**Episode 1**

Victor Ciura

Principal ~~Engineer~~
Rambling Idiot
Rust Tooling @ Microsoft

@ciura_victor
@ciura_victor@hachyderm.io
@ciuravictor.bsky.social

# About me

**Advanced Installer**

**Clang Power Tools**

**Oxidizer SDK**

**Visual C++**

**Rust Tooling**
Microsoft

@ciura_victor
@ciura_victor@hachyderm.io
@ciuravictor.bsky.social

# Disclaimer

## I'm just an engineer, with some opinions on stuff...


# Today: A story in 3 pieces

- ABI / Layout
- Move Semantics
- Codegen & Compilers

# Let me start
# with a ~~sad story~~ cautionary tale

# C++ ABI - Now or Never

February 24, 2020

## The Day The Standard Library Died

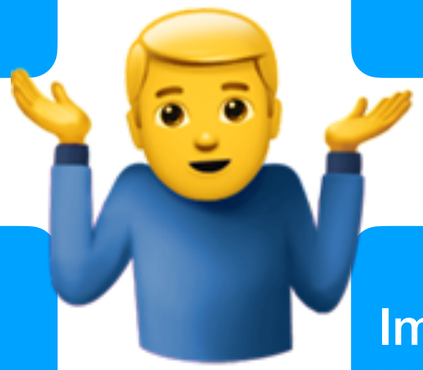
- wg21.link/P2028
- wg21.link/P1863

cor3ntin.github.io/posts/abi/

# C++ the king of mix signals and ambivalent behavior

C++ does not have an ABI resilience model (it's not stable)

The committee will reject* any proposal that could cause ABI breaks in existing STL components



C++ will not officially commit to guaranteeing ABI stability

Implementors* will not change/improve library components if it would cause an ABI break for clients

- wg21.link/P2028
- wg21.link/P1863

# What is ABI, anyway?

ABI isn't a property of a programming language

It's really a property of a system and its toolchain

ABI is something defined by the *platform*

Eg.
Compilers determine class layout: ❌ portable

- Layout of types
  - size & alignment (stride)
  - offsets & types of fields
  - v-table entries
  - closures
- Calling conventions
- Name mangling (symbols)
- Metadata (if applicable)

# ABI Stability - When?

- Don't shut the door on future compiler & library improvements

- Stabilizing the ABI (too early)™ might miss optimization opportunities
  - implement a faster custom calling convention
  - implement optimal structure layout
  - improve the way a std utility works
  - make changes affecting v-table
  - (re)use existing padding

# ABI Stability - Why?

# ABI Stability - Why?

- You don’t have to share the source code of your library

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version
- You don’t have to recompile everything (full project visibility)

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version
- You don’t have to recompile everything (full project visibility)
- Binaries can be shipped and updated independently (patches)

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version
- You don’t have to recompile everything (full project visibility)
- Binaries can be shipped and updated independently (patches)
- Multiple programs can share the same library (incl. std lib)

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version
- You don’t have to recompile everything (full project visibility)
- Binaries can be shipped and updated independently (patches)
- Multiple programs can share the same library (incl. std lib)
- Plugins/extensions (dynamically loaded)

# ABI Stability - Why?

- You don’t have to share the source code of your library
- You can use the most recent compiler for your library
- Use libraries compiled with a different compiler version
- You don’t have to recompile everything (full project visibility)
- Binaries can be shipped and updated independently (patches)
- Multiple programs can share the same library (incl. std lib)
- Plugins/extensions (dynamically loaded)
- Language interop (hybrid projects)

# The (early) 90s are calling...

- Old-school interop
- COM
  - MIDL for interop
  - metadata
  - ABI resilience

# Design for Library Evolution

Principles for ABI-stable library evolution:

- make all promises explicit
- delineate what can and cannot change in a stable ABI
- provide a performance model that indirects only when necessary
- let the authors of libraries & consumers be in control

Doug Gregor
*Implementing Language Support for*
*ABI-Stable Software Evolution in Swift and LLVM*
youtube.com/watch?v=MgPBetJWkmc

# Struct Layout

C++ compilers could provide a class' data members with layout metadata

=> allow representation of Rust struct fields in C++

Retrieve layout via the C++ AST and the rustc query API

# Layout

Type Layout should be as-if we had the whole program:

- *Widget* library should layout the type without indirection
- Expose metadata with layout information:
  - size/alignment of type
  - offsets of each of the public fields
  - overlapping sub-objects
  - padding tricks & vtables
- Attributes, annotations, or compiler synthesized

```
size_t Widget_size = 32;
size_t Widget_align = 8;
size_t Widget_field1_offset = 0;
size_t Widget_field2_offset = 8;
```

# Client/External Code

Client code (external) indirects through layout metadata

- Access a field:
  - read the metadata for the field offset
  - add that offset to the base object
  - cast the new pointer and load the field
- Store an instance on the stack:
  - read the metadata for instance size
  - emit alloca instruction, to setup as needed

# Library Code

Library code (internal) eliminates all ~~indirection~~

- performance: indirects only when necessary

- Access a field:
  - ~~read the metadata for the field offset~~
  - add that offset to the base object
  - cast the new pointer and load the field
- Store an instance on the stack:
  - ~~read the metadata for instance size~~
  - emit alloca instruction, to setup as needed

# Dynamically-sized

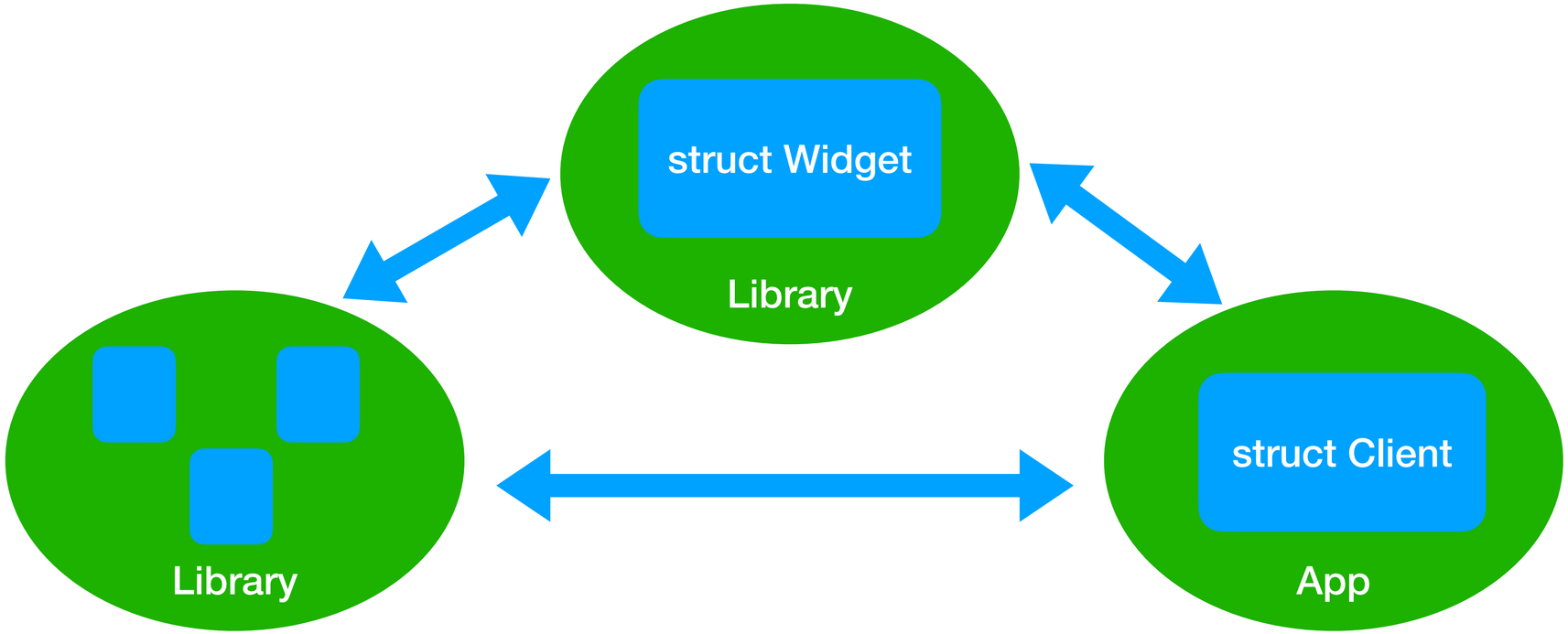
- Support for dynamically-sized things on the stack is key (eg. LLVM)
- Compilers can use of this for of ABI-stable value types:
  - you have local variable of some struct defined in an ABI-stable library
  - so you don’t know it’s size until load time
- Dynamic allocs can handle this nicely (with minimal perf impact)
- C++ desperately wants all objects to have compile-time-constant size
  - the notion of sizeof/alignof being runtime values clashes with the C++ model

# Interop Domains

By explicitly modeling the boundaries between software modules that evolve separately vs. together:

- introduce appropriate indirections across separately-evolved software modules
- while optimizing away that indirection within software modules that are always compiled together

# Interop Domains




An interop domain contains code that will always be compiled together

Domains can control where the costs of interop are paid

# Interop Domains

**Optimization vs. Resilience**

- Across resilience domains => maintain stable ABI
- Within a resilience domain => all implementation details are fair game
  - no indirections (direct access, no computed metadata)
  - no guarantees made
- Optimizations need to be aware of resilience domain boundaries
- A program can have just 1 resilience domain

# Rust ABI Stability

Rust dev: "Can we have stable ABI?"

Rust dev: "We have stable ABI at home."

# Rust ABI Stability

Rust dev: "Can we have stable ABI?"

Rust dev: "We have stable ABI at home."

Stable ABI at home: `#[repr(C)]`

# Rust ABI

## Status quo: `repr(C)` - fake it, till you make it 😀

- Using the C calling convention for function definitions and calls `extern "C" fn`
- Using the C data layout for a type `#[repr(C)]`
- Definitions of C types like char, int, long, etc. `std::ffi::c_*`
- Exporting an item under a stable linking symbol `#[no_mangle]`
- Limited to C types, mostly
- No slices

```
u8, i64, c_int, c_char, ...
&T, &mut T
*const T, *mut T
struct
```

# Rust ABI

The Future™: calling convention and data layout

- Stable calling convention that supports common data types — `extern "crabi" fn`
  - `&str` `&[u8]` `etc.`
- Standard data layout that supports enums (with data), etc. — `#[repr(crabi)]`
  - `enum` `struct`
- Stable layout guarantees of common standard library types — `#[repr(crabi)] in std`
  - `Option` `Result` `etc.`

**crABI**
github.com/joshtriplett/rfcs/blob/text/3470-crabi.md

# Rust ABI

The Future™: mechanism for exporting/importing, naming symbols and working with dynamic libraries

- Exporting items under stable linking symbols, supporting crates, modules, methods `#[export]`
- Use a crate as dynamic library, only importing the exported items `extern dyn crate`
- Cargo features for dynamically linking to Rust libraries `cargo dynamic deps`

# Rust ABI

The Future™: trait objects/vtables and typeid

- A standard data layout for dynamic trait objects (v-tables)
  - `&dyn T`  `&mut dyn T`  `Box<dyn T>`
- A way of dealing with types that depend on global state (eg. allocated objects)
  - `Box`  `Vec`
- Stable typeid
  - `Any`  `catch_unwind`
- Access to std structures like maps through dynamic std trait objects
  - `&dyn HashMap`  `etc.`

# Rust ABI

The Future™: "*Don't stop me now!*" 🎶

- Turning parts of std into an opt-in dynamic library with a stable ABI (std as dylib)
- Tools to help with detect/maintaining ABI compatibility and tools to debug ABI issues
- Store signatures, data layouts in binaries (introspection)

ABI Cafe 🧩☕

faultlore.com/abi-cafe/book/

Pair Your Compilers At The ABI Café:
faultlore.com/blah/abi-puns/

# I like to move it, move it...

## Object Relocation

One particularly sensitive topic about handling C++ **values**
is that they are all conservatively considered **non-relocatable**

## Object Relocation

In contrast, a **relocatable value** would preserve its invariant,
even if its bits were moved arbitrarily in memory

For example, an `int32` is relocatable because moving its 4 bytes
would preserve its actual value, so the address of that value does not matter to its integrity

# I like to move it, move it...

## Object Relocation

C++'s assumption of non-relocatable **values** hurts everybody

for the benefit of a few questionable designs

## Object Relocation

Only a *minority* of objects are genuinely non-relocatable:

Eg.
- objects that use internal pointers
- objects that need to update observers that store pointers to them

# I like to move it, move it...

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

wg21.link/P2786

# I like to move it, move it...

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

Many types in C++ cannot be trivially moved or destroyed, but do support trivially moving an object from one location to another by copying its bits — an operation known as trivial relocation

wg21.link/P2786

# I like to move it, move it...

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

Many types in C++ cannot be trivially moved or destroyed, but do support trivially moving an object from one location to another by copying its bits — an operation known as trivial relocation

wg21.link/P2786

# I like to move it, move it...

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

Many types in C++ cannot be trivially moved or destroyed, but do support trivially moving an object from one location to another by copying its bits — an operation known as trivial relocation

Some types even support bitwise swapping, which requires replacing the objects passed to the swap function, without violating any object invariants

wg21.link/P2786

# I like to move it, move it...

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

Many types in C++ cannot be trivially moved or destroyed, but do support trivially moving an object from one location to another by copying its bits — an operation known as trivial relocation

Some types even support bitwise swapping, which requires replacing the objects passed to the swap function, without violating any object invariants

wg21.link/P2786

## Trivial Relocatability For C++26

Proposal to safely relocate objects in memory

Many types in C++ cannot be trivially moved or destroyed, but do support trivially moving an object from one location to another by copying its bits — an operation known as trivial relocation

Some types even support bitwise swapping, which requires replacing the objects passed to the swap function, without violating any object invariants

Optimizing containers to take advantage of this property of a type is already in widespread use throughout the industry, but is undefined behavior as far as the language is concerned

wg21.link/P2786

# I like to move it, move it...

❌ place a C++ object on a Rust stack since it cannot be safely memcopy-moved (relocated)

C++26 proposal: Make C++ types trivially relocatable (annotate types)

Get standard library to be relocatable
=> allow most C++ types on the Rust stack (efficiency)

# I like to move it, move it...

Support for destructive moves in C++ would match the behavior of Rust drop mechanics

# I like to move it, move it...

Support for destructive moves in C++ would match the behavior of Rust drop mechanics

- Rust move: which is a blind memcpy
  - render the moved-from object inaccessible
- C++ move: where a move is really like a mutating Clone operation
  - leave the moved-from value accessible to be destroyed at the end of the scope

# I like to move it, move it...

Support for destructive moves in C++ would match the behavior of Rust drop mechanics

- Rust move: which is a blind memcpy
  - render the moved-from object inaccessible
- C++ move: where a move is really like a mutating Clone operation
  - leave the moved-from value accessible to be destroyed at the end of the scope

📦 moveit

- safe in-place construction of Rust and C++ objects
- mirrors Rust's drop semantics in its destructive moves
- moved-from values can no longer be used afterwards

# Compilers & Interop

Many of the tricks here require deep compiler involvement:

- on C++ side (pick your poison 🙂)
- on Rust side (easy: 1 instance?)

# Compilers & Interop

Many of the tricks here require deep compiler involvement:

- on C++ side (pick your poison 🙂)
- on Rust side (easy: 1 instance?)

High-fidelity language semantics & mapping of vocabulary types:

- front-ends (C++, rustc)
- toolchain independent IR
- support libs?

# Compilers & Interop

Many of the tricks here require deep compiler involvement:

- on C++ side (pick your poison 🙂)
- on Rust side (easy: 1 instance?)

High-fidelity language semantics & mapping of vocabulary types:

- front-ends (C++, rustc)
- toolchain independent IR
- support libs?

Binary-level fidelity, ABI, linking, dylib, etc.

- platform integration
- post-build tooling
- codegen / back-end

# Active Effort

This year, there have been effervescent talks in the Rust Project & community about this topic (in the broader interop context, not just C++)

# Active Effort

This year, there have been effervescent talks in the Rust Project & community about this topic (in the broader interop context, not just C++)

This is also part of the Rust25H1 Project Goals:

- Evaluate approaches for seamless interop between C++ and Rust
  - Tyler Mandry is the point-of-contact for project goal
  - Tracking issue: rust-lang/rust-project-goals#253

# Active Effort

This year, there have been effervescent talks in the Rust Project & community about this topic (in the broader interop context, not just C++)

This is also part of the Rust25H1 Project Goals:

- Evaluate approaches for seamless interop between C++ and Rust
  - Tyler Mandry is the point-of-contact for project goal
  - Tracking issue: rust-lang/rust-project-goals#253

Rust Foundation joined INCITS in order to participate in the C++ ISO standards process

Jon Bauman attended the February WG 21 meeting in Austria, where he outlined some the Rust/C++ interop strategy, as seen from the Rust side

# Active Effort

## Short term:

- Contribute engineering time to some of the key interop crates
- Gain perspective on what sort of challenges need solutions external to those crates

# Active Effort

## Short term:

- Contribute engineering time to some of the key interop crates
- Gain perspective on what sort of challenges need solutions external to those crates

## Long term:

- Evaluate approaches for seamless interop between C++ and Rust
- Document the problem space of current interop challenges (identify the gaps)
- Facilitate top-down discussions about priorities and tradeoffs

# Active Effort

## Short term:

- Contribute engineering time to some of the key interop crates
- Gain perspective on what sort of challenges need solutions external to those crates

## Long term:

- Evaluate approaches for seamless interop between C++ and Rust
- Document the problem space of current interop challenges (identify the gaps)
- Facilitate top-down discussions about priorities and tradeoffs

## Meetings:

- (Feb 26) We held our first lang-team design meeting on the topic
- (Apr 23) Short-sync on interop interest in industry (attendees)
- Notes: Enabling seamless interop 

# Join the conversation

Anyone who is interested in the topic, please join the Rust Project Zulip server and start engaging on the **#t-lang/interop** channel

You'll find there some familiar Rust and C++ names 🙂

rust-lang.zulipchat.com

# Join the conversation

Anyone who is interested in the topic, please join the Rust Project Zulip server and start engaging on the **#t-lang/interop** channel

You'll find there some familiar Rust and C++ names 🙂

The next *meetings* on the interop will be 🇳🇱HERE, on **May 15-17** at **Rust All-Hands**

rust-lang.zulipchat.com

# Open Discussion

What does interop mean for you?

What are the interop requirements/challenges of your project?

RustWeek

May 2025


# Duck-Tape Chronicles
## Rust/C++ Interop

**Episode 1 – The ABI Menace**


**Episode 2 – Attack of the Codegen**

**Victor Ciura**

Principal ~~Engineer~~

Rambling Idiot

Rust Tooling @ Microsoft

@ciura_victor

@ciura_victor@hachyderm.io

@ciuravictor.bsky.social