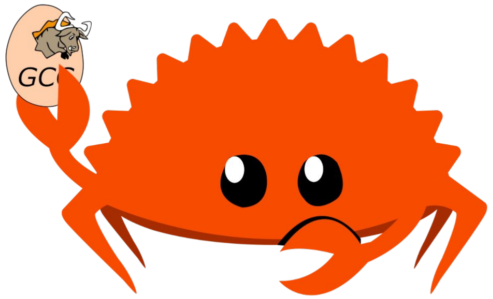
GCC Rust Update



Philip Herron
<herron.philip@googlemail.com>

Pierre-Emmanuel Patry
<pierre-emmanuel.patry@embecosm.com>

Summary

- Milestone progress
  - GCC 14
- Proc Macros Overview
- GSoC 2023
- Sized trait
- Iterators
- Missing features
- Questions

Current status

Milestone	Last Week	This Week	Delta	Start Date	Completion Date	Target
Data Structures 1 - Core	100%	100%	-	30th Nov 2020	27th Jan 2021	29th Jan 2021
Control Flow 1 - Core	100%	100%	-	28th Jan 2021	10th Feb 2021	26th Feb 2021
Data Structures 2 - Generics	100%	100%	-	11th Feb 2021	14th May 2021	28th May 2021
Data Structures 3 - Traits	100%	100%	-	20th May 2021	17th Sep 2021	27th Aug 2021
Control Flow 2 - Pattern Matching	100%	100%	-	20th Sep 2021	9th Dec 2021	29th Nov 2021
Macros and cfg expansion	100%	100%	-	1st Dec 2021	31st Mar 2022	28th Mar 2022
Imports and Visibility	100%	100%	-	29th Mar 2022	13th Jul 2022	27th May 2022
Const Generics	100%	100%	-	30th May 2022	10th Oct 2022	17th Oct 2022
Initial upstream patches	100%	100%	-	10th Oct 2022	13th Nov 2022	13th Nov 2022
Upstream initial patchset	100%	100%	-	13th Nov 2022	13th Dec 2022	19th Dec 2022

Current status

Update GCC’s master branch	100%	100%	-	1st Jan 2023	21st Feb 2023	3rd Mar 2023
Final set of upstream patches	100%	100%	-	16th Nov 2022	1st May 2023	30th Apr 2023
Borrow Checking 1	0%	0%	-	TBD	-	15th Aug 2023
AST Pipeline for libcore 1.49	78%	78%	-	13th Apr 2023	-	1st Jul 2023
HIR Pipeline for libcore 1.49	68%	69%	+1%	13th Apr 2023	-	TBD
Procedural Macros 1	100%	100%	-	13th Apr 2023	3rd Sep 2023	6th Aug 2023
GCC 13.2 Release	100%	100%	-	13th Apr 2023	18th Jul 2023	15th Jul 2023
GCC 14 Stage 3	0%	80%	+1%	TBD	-	1st Nov 2023
core 1.49 functionality [AST]	4%	4%	-	1st Jul 2023	-	1st Nov 2023
Rustc Testsuite Prerequisistes	84%	84%	-	TBD	-	1st Sep 2023
Intrinsics and builtins	18%	18%	-	6th Sep 2022	-	TBD
Const Generics 2	0%	0%	-	TBD	-	TBD
Rust-for-Linux compilation	0%	0%	-	TBD	-	TBD
Procedural Macros 2	46%	50%	+4%	3rd Sep 2023	-	TBD

Current status

- Short term (GCC14 ?)
  - Libcore
- Longer term
  - Liballoc
  - Libstd
  - Rust for linux

GCC 14

- We have 800+ commits out of sync to GCC master
  - Proc macros changes GCC Build system, needs GCC review
    - Adds new runtime library
    - Installed for end-users
    - Also needs to linked into the front-end
      - needs to be compiled for target machine and host
  - Some changes to gcc-diagnostics API
  - Unicode changes to libcpp

Macros in rust

- Declarative macros/Macros by example (MBE)

```
macro_rules! add {
    ($e:expr) => { $e };
    ($e:expr, $($es:expr),*) => { $e + add!($($es),*) };
}

add!(1); // 1
add!(1, 2, 4); // 7
add!(1, add!(2, 3), five(), b, 2 + 4);
```

- Procedural macros

Procedural macros

- Function like
  - Indistinguishable from a declarative macro invocation
  - Often used to create DSL
- Attribute
  - Accepts custom value parameters
- Derive
  - Shall refer to either one or multiple traits

Procedural macros: Function like

```
quote! {
    let value = <#field_type>::new();
}
```

Procedural macros: Derive

```rust
#[derive(Serialize, Deserialize, Debug)]
struct Point {
    x: i32,
    y: i32,
}
```

Procedural macros: Attribute

```
#[get("/cauldron")]
async fn hello() -> impl Responder {
    HttpResponse::Ok().body("Hello cauldron!")
}
```

Procedural macros: Interface

- Special functions
- Input and output types from `libproc_macro` library
- Access to other libraries (libstd, custom crates…)
- Compiled as a shared library

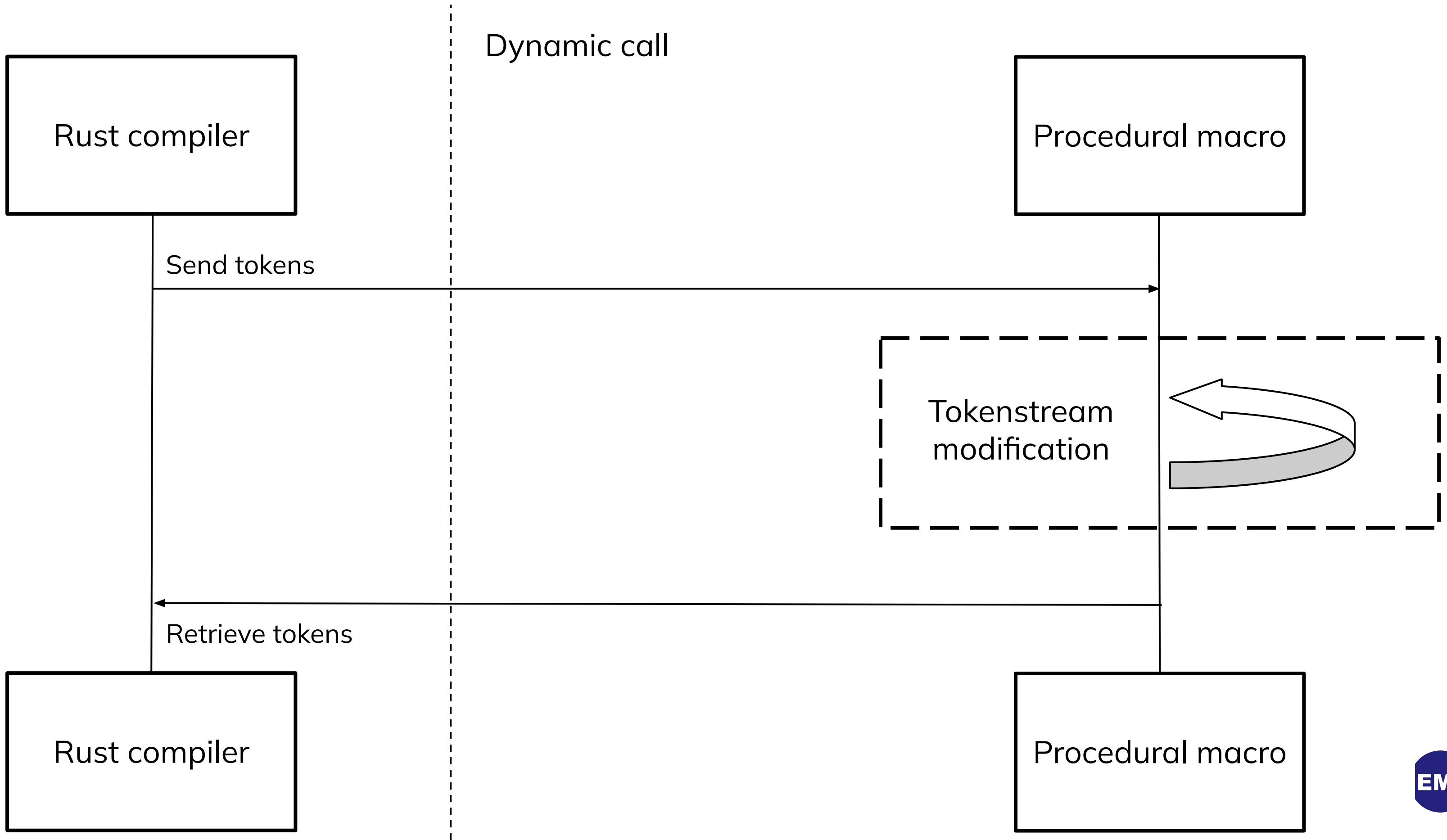
Procedural macros: Interface

```rust
use proc_macro::TokenStream;

#[proc_macro]
pub fn function_like_macro(items: TokenStream) -> TokenStream {
    "fn cauldron_year() -> u32 { 2023 }"
        .parse()
        .unwrap_or(items)
}

#[proc_macro_attribute]
pub fn attribute_macro(_attr: TokenStream, items: TokenStream) -> TokenStream {
    items
}

#[proc_macro_derive(DummyTrait)]
pub fn derive_macro(_items: TokenStream) -> TokenStream {
    TokenStream::new()
}
```

Procedural macros: Interface

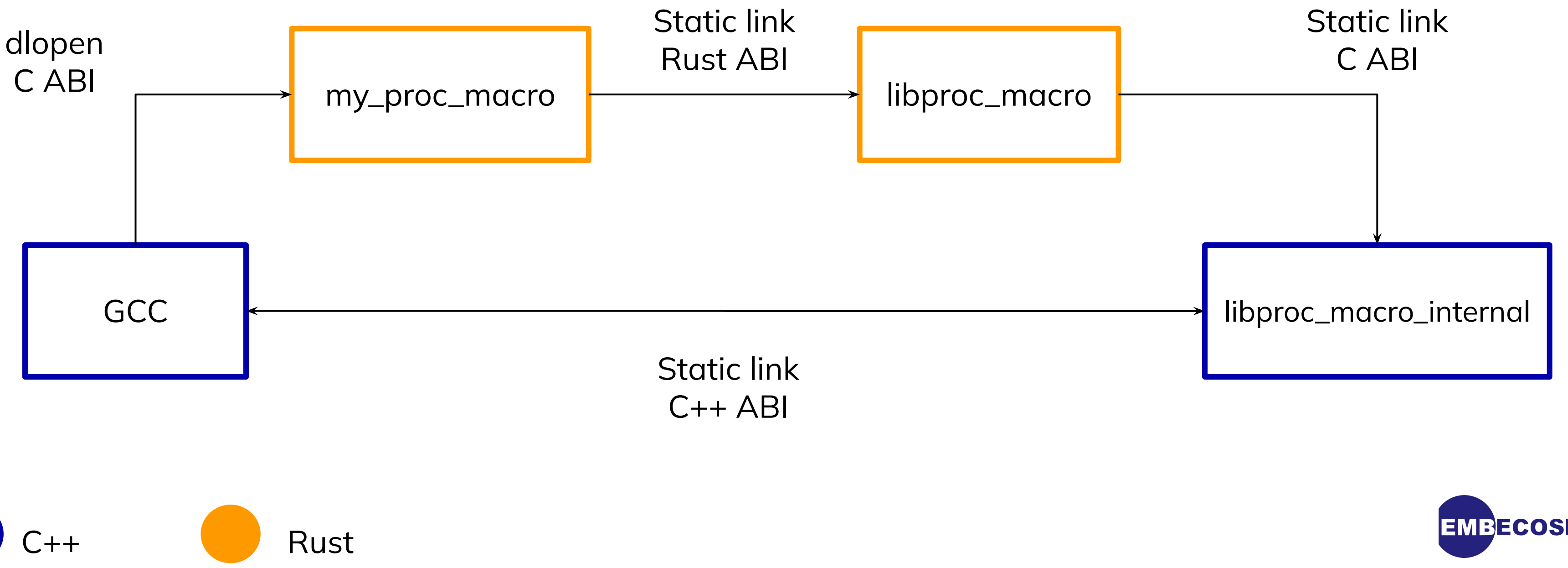
```
lazy_static! {
    static ref HASHMAP: HashMap<u32, &'static str> = {
        let mut m = HashMap::new();
        m.insert(0, "foo");
    }
}

#[get("/cauldron")]
#[other_inner_macro]
async fn hello() -> impl Responder {
    HttpResponse::Ok().body("Hello cauldron!")
}

#[derive(Serialize, Deserialize, Debug)]
struct Point {
    x: i32,
    y: i32,
}
```

Procedural macros



Procedural macros

- The compiler loads the macro as a shared library
- Collect procedural macros
- Call them during expansion



Procedural macros: Last problem

- A string could be converted to a tokenstream
- The conversion code already exists in the compiler

- Split the lexer and converter from GCC ?
- dlopen GCC ?
- Install a callback function on macro load

Procedural macros

- Load the procedural macro and initialize it (callback, bridge value)
- Visit the AST and search for procedural macro call
- Collect nodes of a designated area
- Convert those nodes back to tokens
- Convert those tokens to rust’s tokenstream
- Send those tokenstream to the macro
- Get a tokenstream back from the macro
- Convert them back to tokens
- Parse the resulting tokens back to an AST fragment
- Attach back the fragment to the AST

Procedural macros

- Load the procedural macro and initialize it
- Visit the AST and search for procedural macro call
- Collect nodes of a designated area
- **Convert those nodes back to tokens**
- Convert those tokens to rust’s tokenstream
- Send those tokenstream to the macro
- Get a tokenstream back from the macro
- **Convert them back to tokens**
- Parse the resulting tokens back to an AST fragment
- Attach back the fragment to the AST

Procedural macros

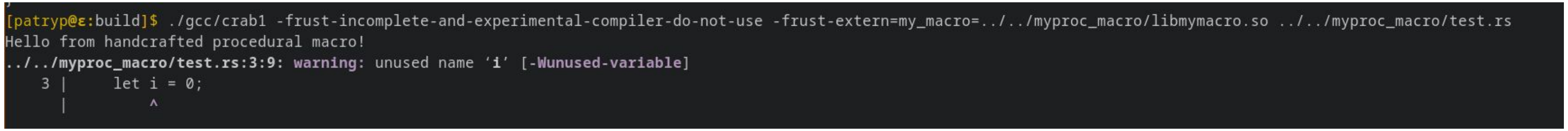
- Converting from tokens to tokenstream is easy
- Converting from tokens to text is even easier
- We have two enormous identical visitors

Let’s merge them!

- Less maintenance
- AST dump becomes reliable and accurate
- Syntax update requires less work

Procedural macros

- Procedural macros can now be expanded…
- …but not yet generated*



```
[patryp@ε:build]$ ./gcc/crab1 -frust-incomplete-and-experimental-compiler-do-not-use -frust-extern=my_macro=../../myproc_macro/libmymacro.so ../../myproc_macro/test.rs
Hello from handcrafted procedural macro!
../../myproc_macro/test.rs:3:9: warning: unused name ‘i’ [-Wunused-variable]
    3 |     let i = 0;
      |         ^
```

* from rust

GSoC 2023

Two GSOC students this year

- Raiki Tamura: Unicode support
  - lexer modification
  - mangling

- Mahad Muhammad: Error Codes
  - 49 Error Codes
  - https://github.com/Rust-GCC/gccrs/issues/2553

Sized

- Trait to denote whether a type has a size
- This means some types can be zero sized
- Requires the introduction of a trait bound relaxing syntax

Iterators

Iterators are everywhere in rust

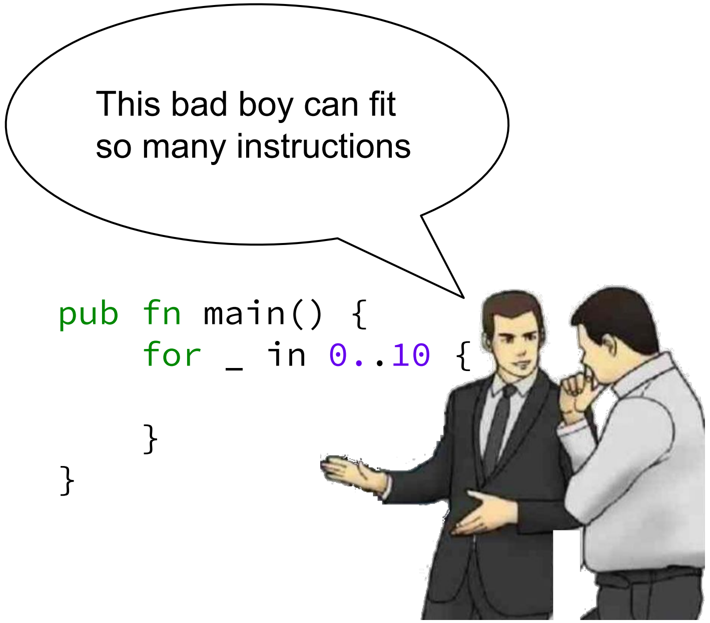
- Imperative and functional
- More than 40 types in the standard library implements IntoIterator!

```
for _ in 0..10 {
   println!("Hello cauldron!") ;
}

Some("Hello cauldron").iter().for_each(|e| println!("{e}"));
```

Iterators

- Why is it so hard ?
  - leverage many functions…
  - …which in turn leverage even more intrinsics
  - those functions are constrained by some traits requiring other traits
- Several month going down the rabbit hole

Iterators



```rust
pub fn main() {
    for _ in 0..10 {

    }
}
```

```asm
<i32 as core::iter::range::Step>::forward_unchecked:
            mov         eax, edi
            mov         ecx, esi
            add         eax, ecx
            ret

core::iter::range::<impl core::iter::traits::iterator::Iterator for core::ops::range::Range<A>>::next:
            push        rax
            mov         rax, qword ptr [rip + <core::ops::range::Range<T> as
core::iter::range::RangeIteratorImpl>::spec_next@GOTPCREL]
            call        rax
            pop         rcx
            ret

<I as core::iter::traits::collect::IntoIterator>::into_iter:
            mov         edx, esi
            mov         eax, edi
            ret

<core::ops::range::Range<T> as core::iter::range::RangeIteratorImpl>::spec_next:
            sub         rsp, 24
            mov         qword ptr [rsp + 8], rdi
            mov         eax, dword ptr [rdi]
            cmp         eax, dword ptr [rdi + 4]
            jl          .LBB3_2
            mov         dword ptr [rsp + 16], 0
            jmp         .LBB3_3
.LBB3_2:
            mov         rax, qword ptr [rsp + 8]
            mov         edi, dword ptr [rax]
            mov         dword ptr [rsp + 4], edi
            mov         esi, 1
            call        <i32 as core::iter::range::Step>::forward_unchecked
            mov         rcx, qword ptr [rsp + 8]
            mov         edx, eax
            mov         eax, dword ptr [rsp + 4]
            mov         dword ptr [rcx], edx
            mov         dword ptr [rsp + 20], eax
            mov         dword ptr [rsp + 16], 1
.LBB3_3:
            mov         eax, dword ptr [rsp + 16]
            mov         edx, dword ptr [rsp + 20]
            add         rsp, 24
            ret

example::main:
            sub         rsp, 24
            mov         dword ptr [rsp], 0
            mov         dword ptr [rsp + 4], 10
            mov         edi, dword ptr [rsp]
            mov         esi, dword ptr [rsp + 4]
            call        qword ptr [rip + <I as core::iter::traits::collect::IntoIterator>::into_iter@GOTPCREL]
            mov         dword ptr [rsp + 8], eax
            mov         dword ptr [rsp + 12], edx
.LBB4_1:
            mov         rax, qword ptr [rip + core::iter::range::<impl core::iter::traits::iterator::Iterator for
core::ops::range::Range<A>>::next@GOTPCREL]
            lea         rdi, [rsp + 8]
            call        rax
            mov         dword ptr [rsp + 20], edx
            mov         dword ptr [rsp + 16], eax
            mov         eax, dword ptr [rsp + 16]
```

EMBECOSM®

What’s Missing for libcore

- Metadata exports
- Drop
- Opaque Types
- Some Intrinsics
- format_args! macro

Missing bits: format_args

- Builtin macro
- Brings support for `println!`
- Various edge case
  - Named parameters
    - Inline
    - From environment
  - Formatting
    - Width
    - Fill
    - Alignment
    - Sign
    - Hexadecimal / Binary / Octal
    - Precision
  - Escaping

```
pub fn main() {
    let var = 3;
    format_args!(
        r"
{}
{:?}
{var}
{:#?}
{:04}
{:<5}
{:-<5}
{:^5}
{:>5}
{:#010x}
{:8$}
{value}
{{}}
",
        1, 2, (4, 5), 6, 7, 8, 9, 10, 11, 12,
value = 13
    );
}
```

EMBECOSM®

Missing bits: format_args

```
format_string := text [ maybe_format text ] *
maybe_format := '{' '{' | '}' '}' | format
format := '{' [ argument ] [ ':' format_spec ] [ ws ] * '}'
argument := integer | identifier
format_spec := [[fill]align][sign]['#']['0'][width]['.' precision]type
fill := character
align := '<' | '^' | '>'
sign := '+' | '-'
width := count
precision := count | '*'
type := '' | '?' | 'x?' | 'X?' | identifier
count := parameter | integer
parameter := argument '$'
```

Missing bits: Drop

```rust
struct HasDrop;

impl Drop for HasDrop {
    fn drop(&mut self) {
        println!("Dropping HasDrop!");
    }
}

struct HasTwoDrops {
    one: HasDrop,
    two: HasDrop,
}

impl Drop for HasTwoDrops {
    fn drop(&mut self) {
        println!("Dropping HasTwoDrops!");
    }
}

fn main() {
    let _x = HasTwoDrops { one: HasDrop, two: HasDrop };
    println!("Running!");
}
```

Missing bits: Opaque Types

```rust
use std::fmt;

trait Human {
    fn name(&self) -> &str;
}

fn cauldron(person : &impl Human) -> impl fmt::Display + '_ {
    person.name()
}

struct Maintainer;

impl Human for Maintainer {
   fn name(&self) -> &str {
       "John Doe"
   }
}

fn main() {
    let maintainer = Maintainer;

    println!("{}", cauldron(&maintainer));
}
```

Links

- Github: https://rust-gcc.github.io/
- Reports: https://github.com/Rust-GCC/Reporting
- Zulip: https://gcc-rust.zulipchat.com/
- IRC: irc.oftc.net #gccrust
- https://gcc.gnu.org/mailman/listinfo/gcc-rust

Get Involved

- Goal is to make working on compilers fun
  - Lots of `good-first-pr` issues to work through
    - Refactoring work
    - Bugs
  - Lots of scope to make your mark on the compiler
- Google Summer of Code 2021, 2022 and 2023
- Status reporting
  - Weekly and Monthly
  - Shout out to contributors
  - Open and transparent
- Monthly Community Call and Weekly Syncup
  - In our calendar and Zulip
  - Open to everyone who is interested
  - Hosted on Jitsi

EMBECOSM®

OPEN SOURCE SECURITY

Questions?

github.com/Rust-GCC/gccrs/

gcc-rust.zulipchat.com/

irc.oftc.net #gccrust