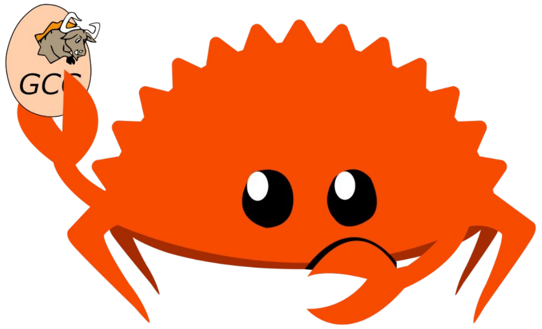
GCC Rust Update



Philip Herron
<herron.philip@googlemail.com>

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

Summary

- Milestone progress
  - What’s missing
  - GCC 14
- Proc Macros Overview
- GSoC 2023
- Roadmap
  - Sized trait
  - Iterators
  - Drop
  - Opaque Types
  - missing features
- Future Work
- 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

What’s Missing

- Metadata exports
- Drop
- Opaque Types
- Some Intrinsics
- format_args! macro (allows for println!)

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 compiled for target machine and host
    - Some changes to gcc-diagnostics API
    - Unicode changes to libcpp

Proc Macros Overview

- What are they
- How do they work
- Runtime libraries

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

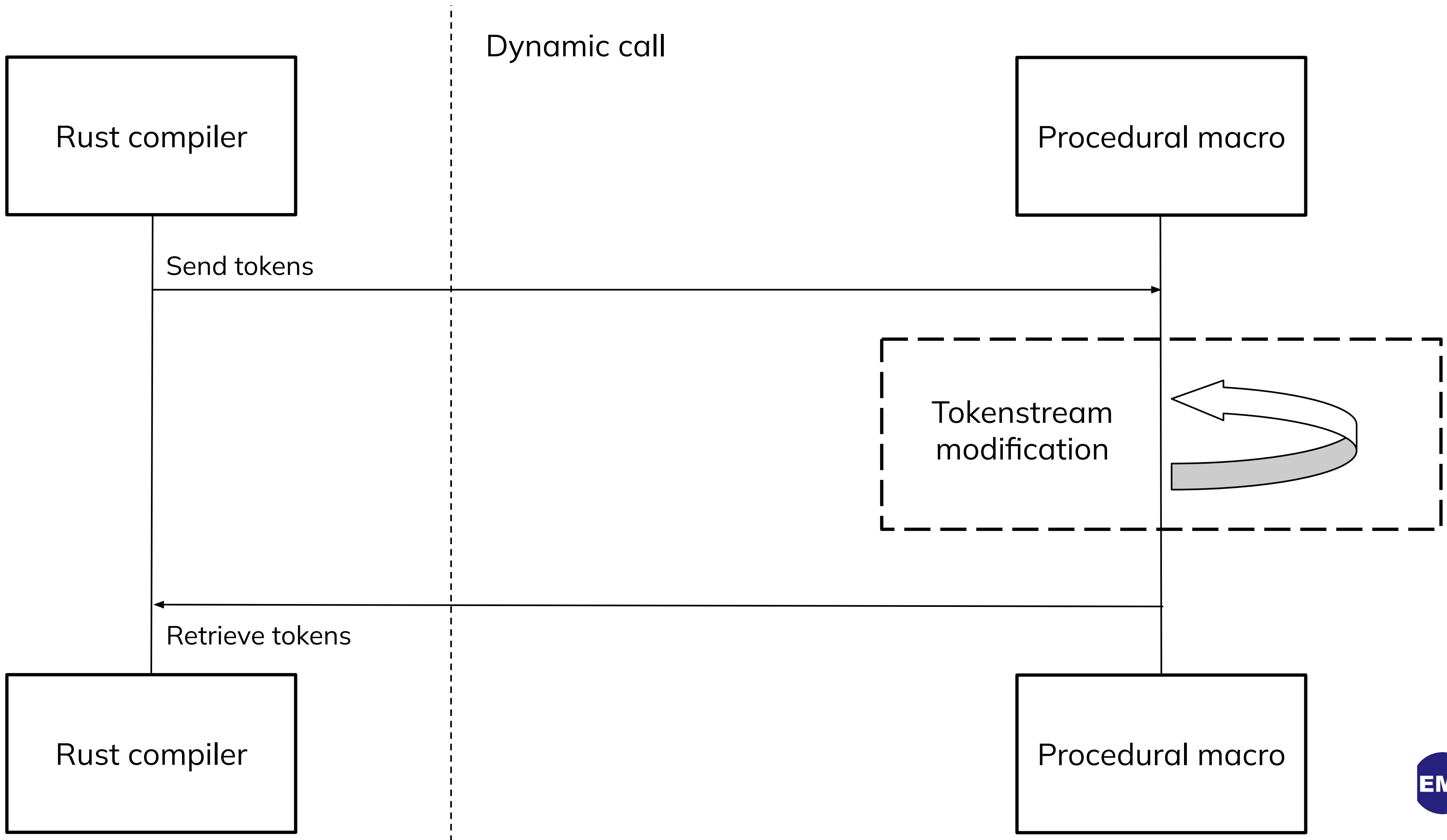
Procedural macros: Interface

```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

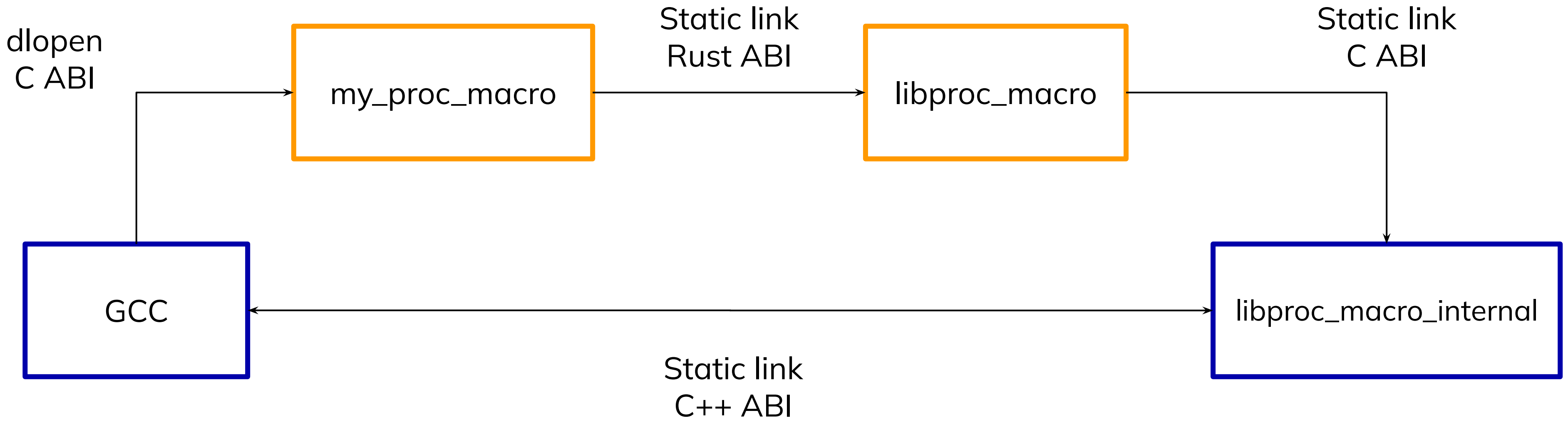
```
lazy_static! {
    static ref HASHMAP: HashMap<u32, &'static str> = {
        let mut m = HashMap::new();
        m.insert(0, "foo");
    }
}


#[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



● C++ ● Rust

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

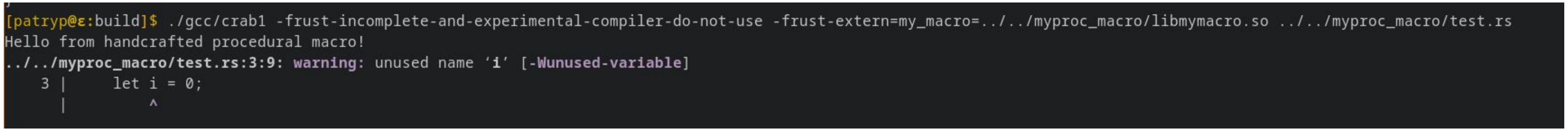
- Converting from tokens to tokenstream is easy
- Converting from tokens to text is even easier
- We have two enormous identical visitors

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 constraint relaxing operator “?”

Iterators

Iterators are everywhere in rust

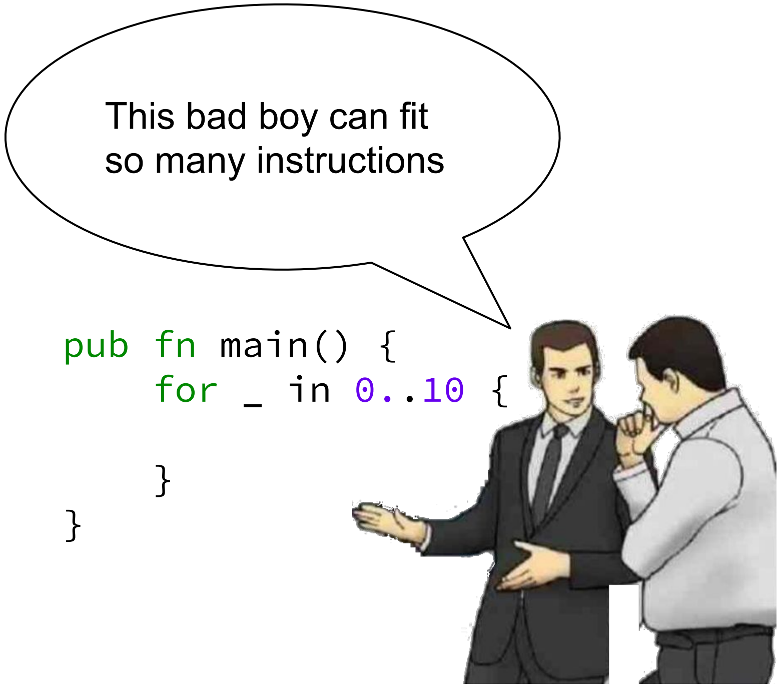
- Imperative and functional
- More than 40 types in the standard library implements into iterator!

```rust
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]
            cmp         rax, 0
            jne         .LBB4_1
            add         rsp, 24
            ret
```

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!");
}
```

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));
}
```

Community

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

Questions?

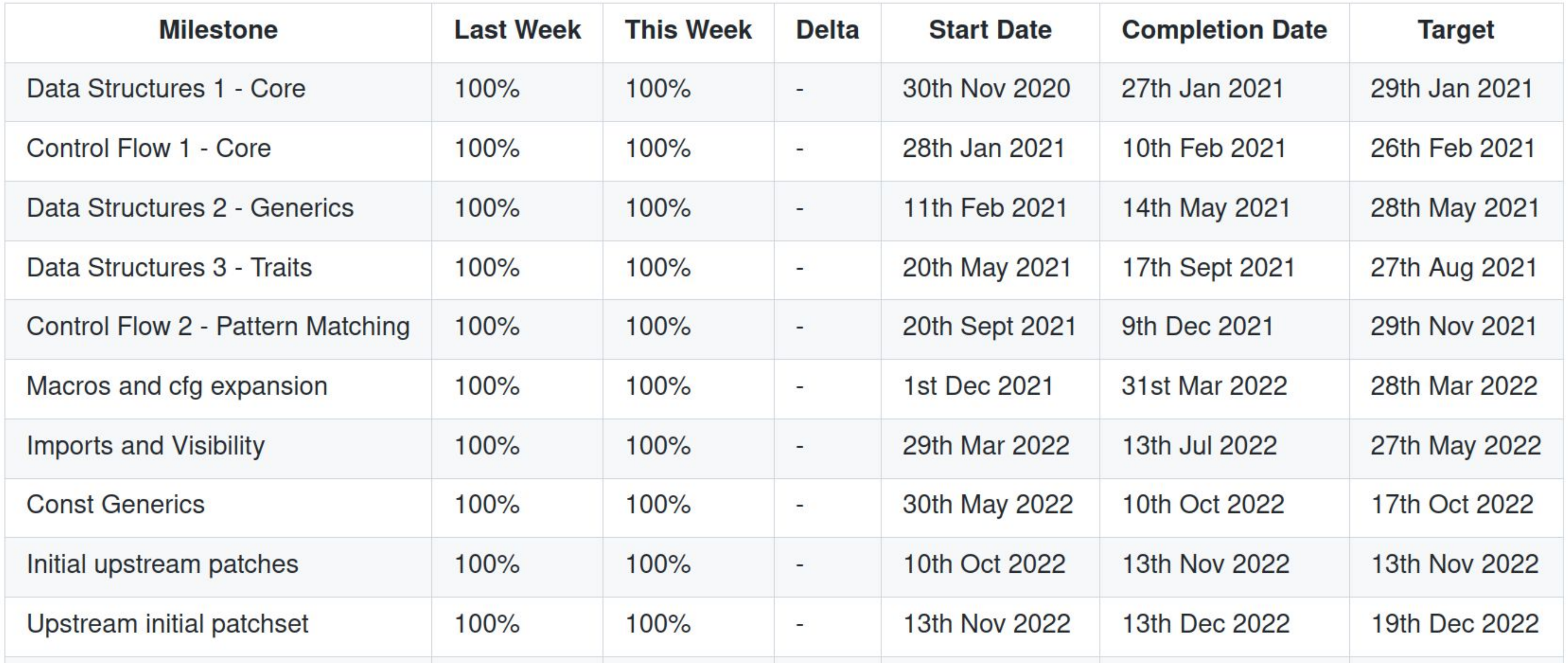
github.com/Rust-GCC/gccrs/

gcc-rust.zulipchat.com/

irc.oftc.net #gccrust

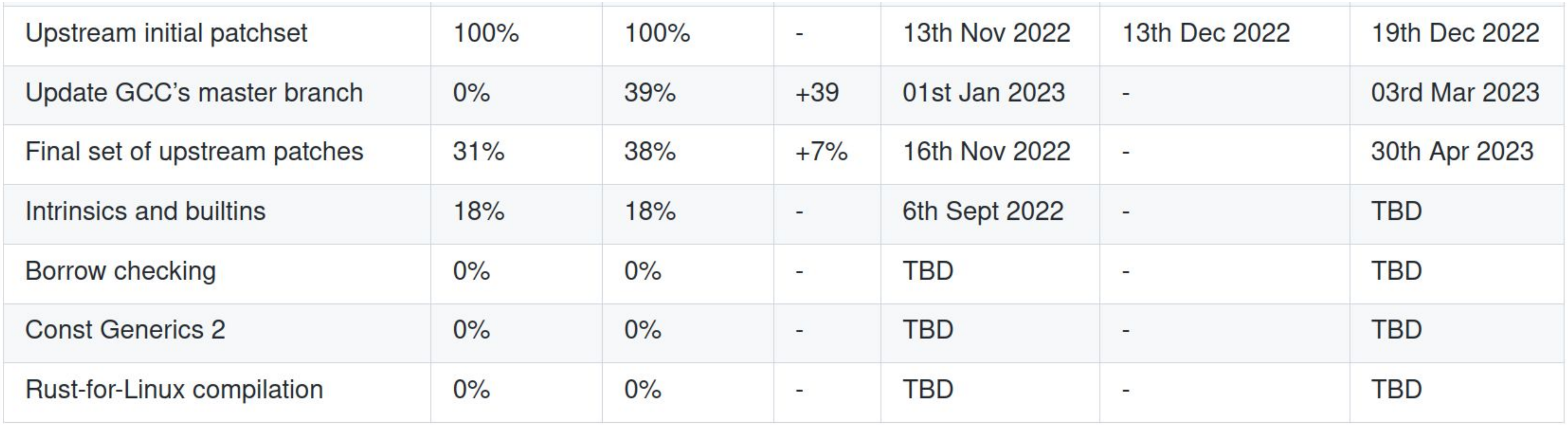
Backup - Slides

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 Sept 2021	27th Aug 2021
Control Flow 2 - Pattern Matching	100%	100%	-	20th Sept 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



Upstream initial patchset	100%	100%	-	13th Nov 2022	13th Dec 2022	19th Dec 2022
Update GCC’s master branch	0%	39%	+39	01st Jan 2023	-	03rd Mar 2023
Final set of upstream patches	31%	38%	+7%	16th Nov 2022	-	30th Apr 2023
Intrinsics and builtins	18%	18%	-	6th Sept 2022	-	TBD
Borrow checking	0%	0%	-	TBD	-	TBD
Const Generics 2	0%	0%	-	TBD	-	TBD
Rust-for-Linux compilation	0%	0%	-	TBD	-	TBD

Frontend Representations

- AST (Abstract Syntax Tree)
  - Raw AST (Structured C++ class hierarchy)
- HIR (high level IR)
  - Desugared AST
    - remove distinction between functions/methods
    - macros don’t exist anymore
    - much much more....
- Generic (GCC IR)

Macro Expansion

- Macro arguments are typed
  - `expr, stmt, path, pat, vis...`
- Repetitions
- Mathematical logic
  - Kleene Operators
    - `* ? +`
  - Follow-set Ambiguity Restrictions
  - That we need to implement!

Macro Expansion

```
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);
```

Macro Expansion

```
macro_rules! tuplomatron {
  ($($e:expr),* ; $($f:expr),*) => { ( $( ( $e, $f ) ),* ) };
}

let tuple = tuplomatron!(1, 2, 3; 4, 5, 6); // valid
let tuple = tuplomatron!(1, 2, 3; 4, 5); // invalid
```

Macro Expansion

```
macro_rules! invalid {
  ($e:expr forbidden) => {};
  //  Forbidden by the follow-set ambiguity restriction

  ($e:expr $(,)? $(;)* $(=>)* forbidden) => {};
  //  1      2     3     4        5              (matches)
}
```

Extra HIR checks

- Privacy pass
  - Privacy in Rust is very different from C++
  - `pub(in path)`, `pub(super)`, `pub(crate)`…
- Unsafe
  - Some actions are only allowed in `unsafe` contexts
    - Dereferencing raw pointers, calling unsafe or extern functions, use of mutable or extern static variables, inline assembly…

Other Rust specific shenanigans

- Macros are lazy
    - No they're not
- Code sharing between crates
    - Headers like C/C++?
    - Dark `ELF` magic?
        - AST Serializing/Deserializing
- Type system
    - Extremely complex and powerful
    - Never type, GATs…
    - Sum types
    - Not a lot of GCC-languages have that!
- Inline assembly
    - Different from GCC's
    - Translation required

EMBECOSM®

Contributing | Reviewing | Merging | Upstreaming

Inspired from `rustc`'s workflow

- Github
- Zulip
- `bors r+`

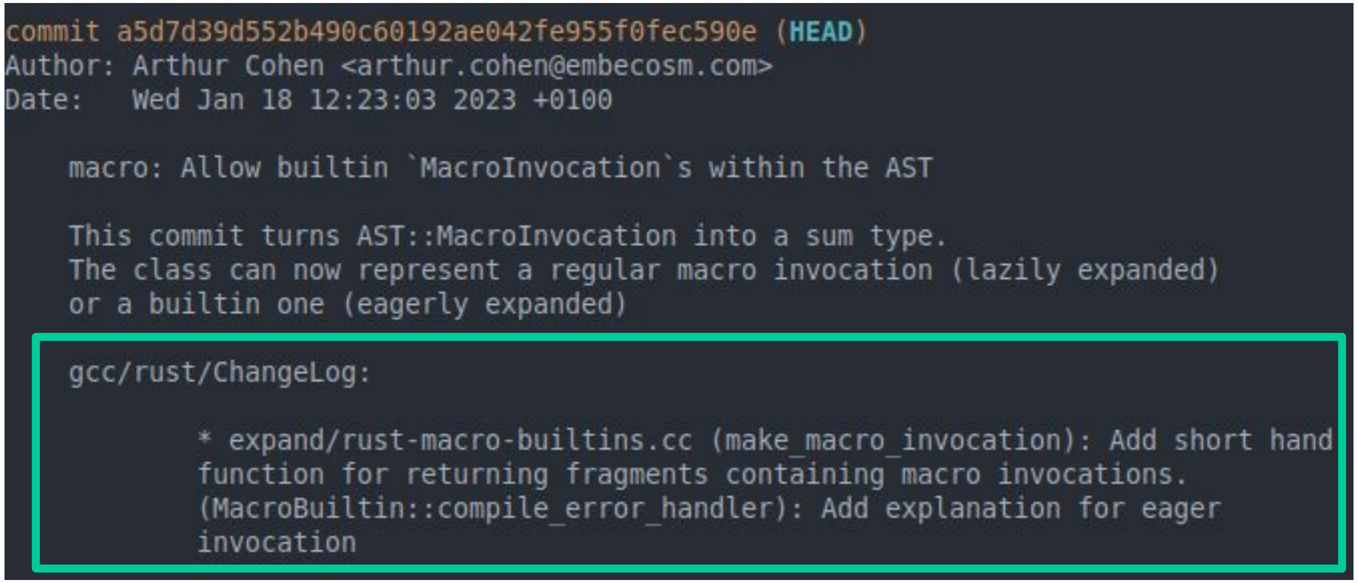
But also...

- IRC
- gcc-rust@gcc.gnu.org
- Mailing list and patches

- No matter your background, you can contribute

GCC development is hard

- Email based code submitting/reviewing is difficult
- GCC Changelogs are hard to write
- Pushing directly to GCC's main branch
- `git send-email`



```
commit a5d7d39d552b490c60192ae042fe955f0fec590e (HEAD)
Author: Arthur Cohen <arthur.cohen@embecosm.com>
Date:   Wed Jan 18 12:23:03 2023 +0100

    macro: Allow builtin `MacroInvocation`s within the AST

    This commit turns AST::MacroInvocation into a sum type.
    The class can now represent a regular macro invocation (lazily expanded)
    or a builtin one (eagerly expanded)

    gcc/rust/ChangeLog:

            * expand/rust-macro-builtins.cc (make_macro_invocation): Add short hand
            function for returning fragments containing macro invocations.
            (MacroBuiltin::compile_error_handler): Add explanation for eager
            invocation
```

GCC development is hard

- We submit patches/commits to GCC's mailing list for your contributions
- Lots of CI
- Lots more machines building and bootstrapping `gccrs`
- Commit format checkers
- Working on a bot to post the Changelog template

GCC development is hard

- GCC development stages
  - Some files cannot be edited from November to May
- We keep track of that
  - Maintaining a “GCC-ready” branch
  - As well as our main development branch

EMBECOSM®

Is it working?

- More than 50 contributors in 2022 overall
- Multiple students
    - Multiple internships
- GCC developers
- Rust core team

Status | Future Work | Open Questions

When is it ready?

- Can compile `libcore` and actually works
  - Implements all necessary `lang` items
  - Unstable APIs, macros, attributes...
  - Passes the `rustc 1.49` testsuite!
- `libcore, liballoc...`
- `libproc_macro`
  - Powerful procedural macros
  - Requires an RPC server in the front-end
- Borrow checking
  - Polonius project
    - Having it optional is a no go for the community
- We are part of this year's GSoC!

EMBECOSM®

GSoC

- GSoC student Faisal Abbas ported large portions of C++ constexpr evaluation

```
const A: i32 = 1;

const fn test(a: i32) -> i32 {
    let b = A + a;
    if b == 2 {
        return b + 2;
    }

    a
}

const B: i32 = test(1);
const C: i32 = test(12);
```

GSoC

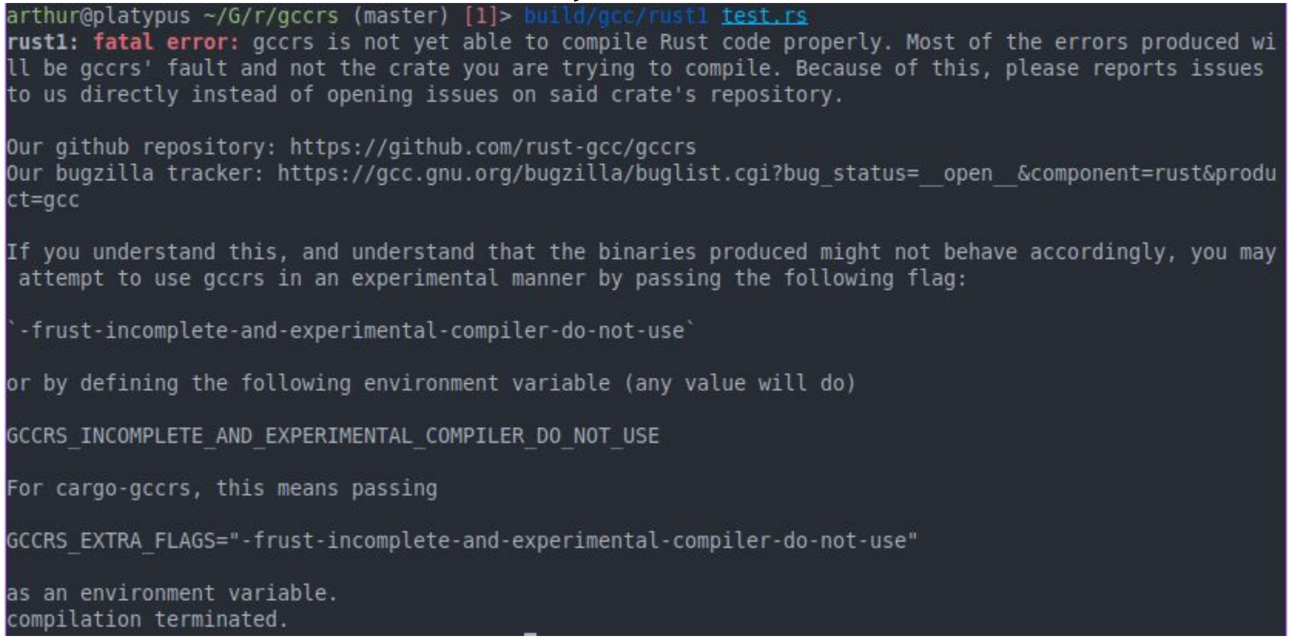
- HIR debugging dump
- Unicode support
- Metadata exports
- Better user error handling + Rust error codes

Tooling

- Testing project
  - Tries compiling various projects using `gccrs`
    - `blake3` cryptography library
    - `libcore` 1.49
    - All the valid cases from the `rustc` testsuite
      - in `#[no_std]` mode
      - in `#[no_core]` mode
  - Eventually add RfL to it!
- Testsuite generator
- Website
- Report generator and tooling
- `cargo-gccrs`
- Web dashboard

Finally...

- RiiR ?
  - Limited to Rust 1.49 for bootstrapping purposes
    - `gccrs` "1.0" will be able to compile `gccrs` "2.0"
  - Still a ways off :)
- The goal is NOT to break the ecosystem



```
arthur@platypus ~/G/r/gccrs (master) [1]> build/gcc/rust1 test.rs
rust1: fatal error: gccrs is not yet able to compile Rust code properly. Most of the errors produced wi
ll be gccrs' fault and not the crate you are trying to compile. Because of this, please reports issues
to us directly instead of opening issues on said crate's repository.

Our github repository: https://github.com/rust-gcc/gccrs
Our bugzilla tracker: https://gcc.gnu.org/bugzilla/buglist.cgi?bug_status=__open__&component=rust&produ
ct=gcc

If you understand this, and understand that the binaries produced might not behave accordingly, you may
 attempt to use gccrs in an experimental manner by passing the following flag:

`-frust-incomplete-and-experimental-compiler-do-not-use`

or by defining the following environment variable (any value will do)

GCCRS_INCOMPLETE_AND_EXPERIMENTAL_COMPILER_DO_NOT_USE

For cargo-gccrs, this means passing

GCCRS_EXTRA_FLAGS="-frust-incomplete-and-experimental-compiler-do-not-use"

as an environment variable.
compilation terminated.
```