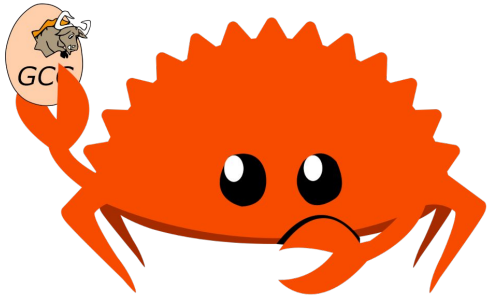
EMBECOSM®

OPEN SOURCE SECURITY

Kangrejos 2025
2025-09-17
Oviedo

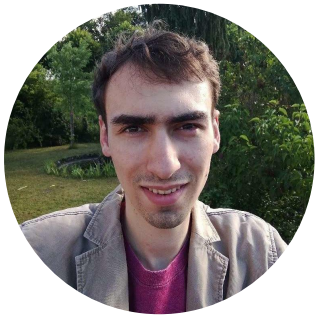
From GCC to RFL



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

CC BY

What is even GCCRS ? Who am I ?

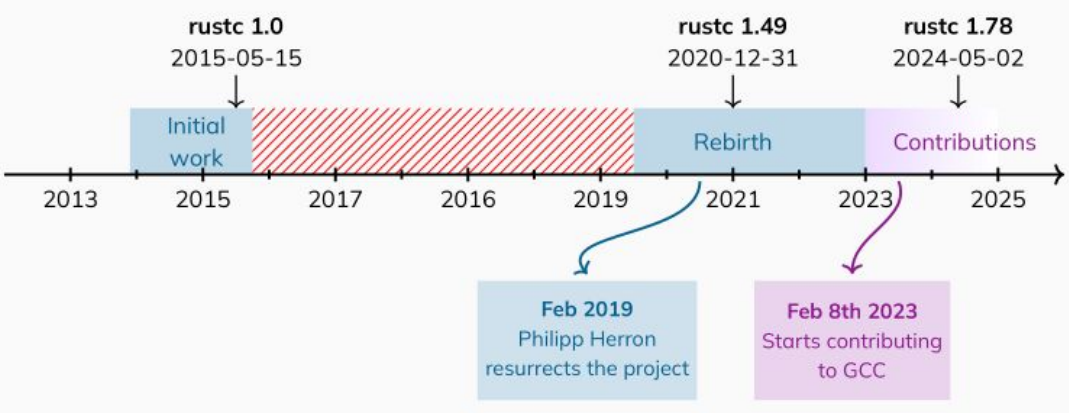


- Work at Embecosm SAS
  - Co-lead on GCCRS since 2023
  - Involved in RISCV toolchain
- Lecturer at EPITA
- Love kernel & microarchitecture stuff

EMBECOSM®

What is even GCCRS ?

- Alternative implementation of a rust compiler
- Different from rustc_codegen_gcc
- Rust v1.49
- Already included in your GCC



EMBECOSM®

When will GCC be able to compile rust ?

EMBECOSM®

We do compile (some) rust

```rust
#[lang = "sized"]
pub trait Sized {}

#[lang = "fn_once"]
pub trait FnOnce<Args> {
    #[lang = "fn_once_output"]
    type Output;

    extern "rust-call" fn call_once(self, args: Args) -> Self::Output;
}

fn main() -> i32 {
    let closure_annotated = |i: i32| -> i32 { i + 1 };

    let i = 1;
    closure_annotated(i) - 2
}
```

https://godbolt.org/z/jvjb5Exsr

EMBECOSM®

What’s the catch ?

A better example: Rust out of tree

```rust
// SPDX-License-Identifier: GPL-2.0
//! Rust out-of-tree sample

use kernel::prelude::*;

struct RustOutOfTree {
    numbers: KVec<i32>,
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

Use declaration

- Part of the name resolution algorithm
- **Complex**
- Reworked over the past two years
  - Famous “3 week project” (Arthur Cohen, 2023)
- Feature parity with old name resolution achieved
- Almost complete

EMBECOSM®

A better example: Rust out of tree

```rust
// SPDX-License-Identifier: GPL-2.0
//! Rust out-of-tree sample

use kernel::prelude::*;

struct RustOutOfTree {
    numbers: KVec<i32>,
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

Struct declaration

- Has been working for a long time
- Generic structs too

EMBECOSM®

A better example: Rust out of tree

```rust
impl kernel::Module for RustOutOfTree {
    fn init(_module: &'static ThisModule) -> Result<Self> {
        pr_info!("Rust out-of-tree sample (init)\n");

        let mut numbers = KVec::new();
        numbers.push(72, GFP_KERNEL)?;
        numbers.push(108, GFP_KERNEL)?;
        numbers.push(200, GFP_KERNEL)?;

        Ok(RustOutOfTree { numbers })
    }
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

Macros

- MBEs are properly expanded
- Still a few issues with metavars

A better example: Rust out of tree

```
impl kernel::Module for RustOutOfTree {
    fn init(_module: &'static ThisModule) -> Result<Self> {
        pr_info!("Rust out-of-tree sample (init)\n");

        let mut numbers = KVec::new();
        numbers.push(72, GFP_KERNEL)?;
        numbers.push(108, GFP_KERNEL)?;
        numbers.push(200, GFP_KERNEL)?;

        Ok(RustOutOfTree { numbers })
    }
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

A better example: Rust out of tree

```
impl kernel::Module for RustOutOfTree {
    fn init(_module: &'static ThisModule) -> Result<Self> {
        pr_info!("Rust out-of-tree sample (init)\n");

        let mut numbers = KVec::new();
        numbers.push(72, GFP_KERNEL)?;
        numbers.push(108, GFP_KERNEL)?;
        numbers.push(200, GFP_KERNEL)?;

        Ok(RustOutOfTree { numbers })
    }
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

Try operator

- Requires a lot of lang item and intrinsics
  - Sized, From, Try
  - Result Ok/Err
- Implemented this year

EMBECOSM®

A better example: Rust out of tree

```
module! {
    type: RustOutOfTree,
    name: "rust_out_of_tree",
    authors: ["Rust for Linux Contributors"],
    description: "Rust out-of-tree sample",
    license: "GPL",
}
```

https://github.com/Rust-for-Linux/rust-out-of-tree-module/blob/main/rust_out_of_tree.rs

EMBECOSM®

Another macro ?

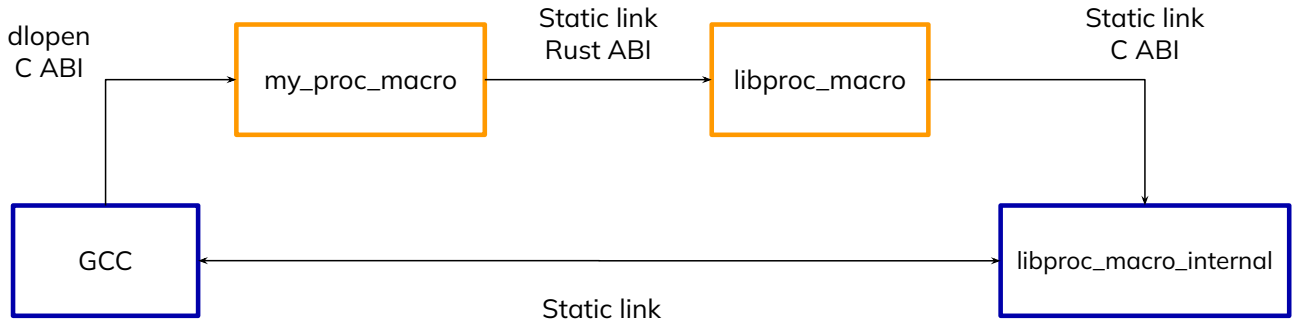
- $LINUX/rust/macro/lib.rs

```
use proc_macro::TokenStream;

// [...]

#[proc_macro]
pub fn module(ts: TokenStream) -> TokenStream {
    module::module(ts)
}
```

EMBECOSM®

Procedural macros

- Compiled procedural macros are compiler specific
- We can load and expand our own procedural macros
  - The TokenStream structure must be C++ compatible
  - We have our own proc_macro crate
  - The procedural macro requires the standard library
- Cross compilation issues



C++ Rust

EMBECOSM®

Dependencies

EMBECOSM®

Dependencies

- The kernel is compiled using `no_std`.

```
// SPDX-License-Identifier: GPL-2.0

//! The `kernel` crate.
//!
//! This crate contains the kernel APIs that have been ported or wrapped for
//! usage by Rust code in the kernel and is shared by all of them.
//!
//! In other words, all the rest of the Rust code in the kernel (e.g. kernel
//! modules written in Rust) depends on [`core`] and this crate.
//!
//! If you need a kernel C API that is not ported or wrapped yet here, then
//! do so first instead of bypassing this crate.

#![no_std]
```

https://github.com/Rust-for-Linux/linux/blob/rust-next/rust/kernel/lib.rs#L14

EMBECOSM®

Dependencies

- Even if we can compile a module we **need** to compile `core` first.
- Do we compile `core` ?

EMBECOSM®

Core

- Core is made of 56,386 lines of rust
- With nightly features
- Complex import and exports
    - Lang items should be resolved and matched to their compiler counterpart

Future improvements

- Compile libcore
- Compile it correctly
- Close the gap between rust 1.49 and 1.78
  - RFL features are high priority
    - offset_of! builtin macro
  - Shouldn't be too long
    - Most changes were made in the standard library
    - Some features were stabilized and previously used in `core`
- Performance ?

EMBECOSM®

EMBECOSM®

OPEN SOURCE SECURITY

Questions?

github.com/Rust-GCC/gccrs/

gcc-rust.zulipchat.com/

irc.oftc.net #gccrust

CC BY