Using Rust for Mission Critical Systems

Jonathan Pallant @ Ferrous Systems, September 2023

V5 2023-09-14

ferrous systems

Introductions

- Jonathan Pallant (@thejpster)
- Embedded systems development for ~20 years
  - Delphi, PHP, Perl, C, C++, C#, Bash, Ruby, Python, JavaScript, Rust
- Rust Embedded Working Group
- Rust Leadership Council
- Senior Engineer at Ferrous Systems

> “Ferrous Systems provides a one-stop-shop service for businesses that want to harness the power of Rust.”

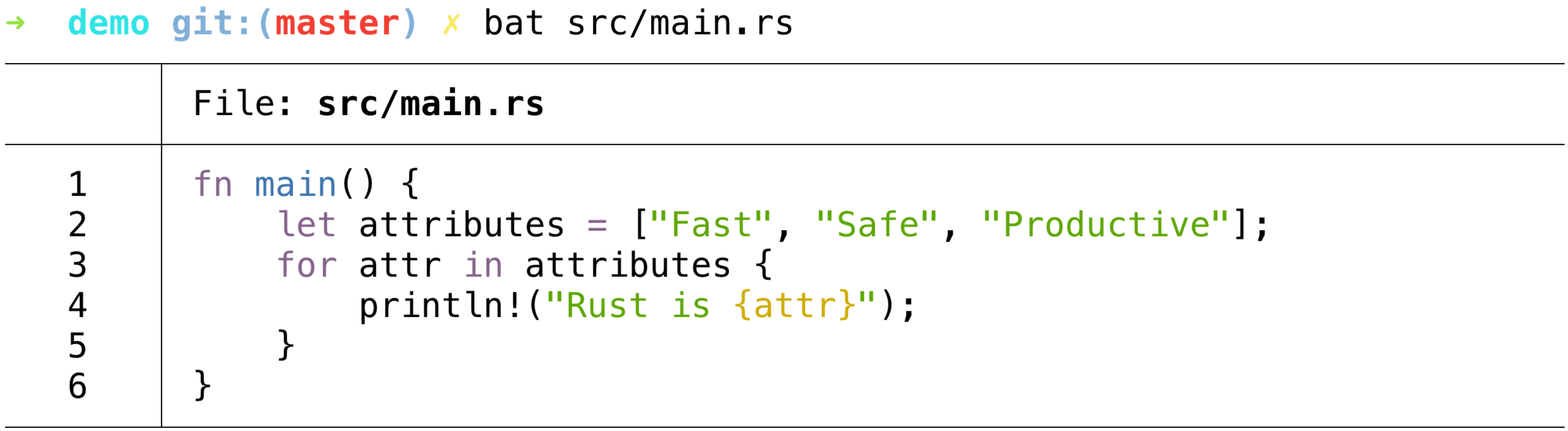
– *https://ferrous-systems.com*

Agenda

- Rust: An empathic systems programming language 🟥
- But what about when it’s Mission Critical? 🟩
- Case Study: Porting Rust to a new platform 🟦
- Questions?

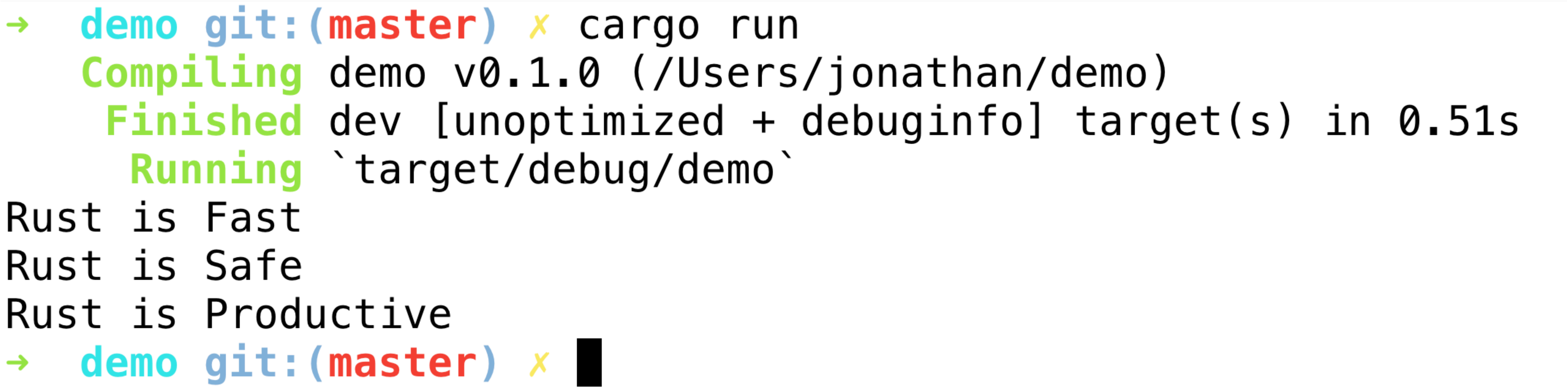
Rust: An empathic systems programming language

Rust’s Key Features



```
➜  demo git:(master) ✗ bat src/main.rs
File: src/main.rs
fn main() {
    let attributes = ["Fast", "Safe", "Productive"];
    for attr in attributes {
        println!("Rust is {attr}");
    }
}
```

Batteries are Included



```
➜  demo git:(master) ✗ cargo run
   Compiling demo v0.1.0 (/Users/jonathan/demo)
    Finished dev [unoptimized + debuginfo] target(s) in 0.51s
     Running `target/debug/demo`
Rust is Fast
Rust is Safe
Rust is Productive
➜  demo git:(master) ✗
```

What makes it special?

- The Rust Compiler **statically analyses** the *ownership* of all of your variables
- First-class **slice types**, **iterators**, and Unicode **strings**
- **Compile-time code generation** (e.g. printing structs to the console…)
- *Static* or *Dynamic Dispatch* with **traits** - your choice
- **Performance on-par with C** (and easier to multi-thread safely)
- A commitment to **fix any unclear error messages**

Rust has been successful at:

- Network Services
- Command-line tools
- Operating System Components and Drivers
- Bootloaders
- Embedded Systems

Embedded Systems?

- Rust is a cross-compiler that supports target binaries either:
  - Running under an Operating System (Linux, macOS, Windows, etc)
  - Running on bare-metal or an unsupported OS
- Tier 1: Macs, PCs, Arm64 Linux
- Tier 2: PowerPC, MIPS, RISC-V, other Arm systems, …
- Tier 3: Motorola 68000, Sony PSP, SPARC, QNX, VxWorks, …

What’s in the box?

- `rustc` - converts Rust source code to object code (.o)
- `cargo` - build system, package manager and test runner
- `libcore`, `liballoc` and `libstd` - the Rust standard libraries
- `lld` - the LLVM linker*
- `rustdoc` - makes HTML documentation
- `rustfmt` - formats Rust source code
- `clippy` - suggests improvements to your source code
- `rust-analyser` - an IDE plugin for auto-complete, rename, annotations…
- `rustup` - downloads new versions of all of the above

Who’s in charge?

- The Rust Project produces the toolchain
- Teams and Working Groups, led by the Leadership Council
    - T-compiler, T-libs, T-lang, T-release, etc
    - wg-embedded, wg-cli, wg-async, etc

Who’s in charge?

- The Rust Foundation supports The Rust Project
- Companies join as members

aws

Google

HUAWEI

JFrog

Meta

Microsoft

“Based on our studies, more than 2/3 of respondents are confident in contributing to a Rust codebase within two months or less when learning Rust … Anecdotally, these ramp-up numbers are in line with the time we’ve seen for developers to adopt other languages, both inside and outside of Google.”

- *https://opensource.googleblog.com/2023/06/rust-fact-vs-fiction-5-insights-from-googles-rust-journey-2022.html*

But what about when it’s Mission Critical?

> “Ferrocene will provide a qualified Rust compiler tool chain. With this, Ferrous Systems will make Rust a first-class language for mission-critical and functional safety systems.”

– *https://ferrous-systems.com/ferrocene/*

Ferrocene…

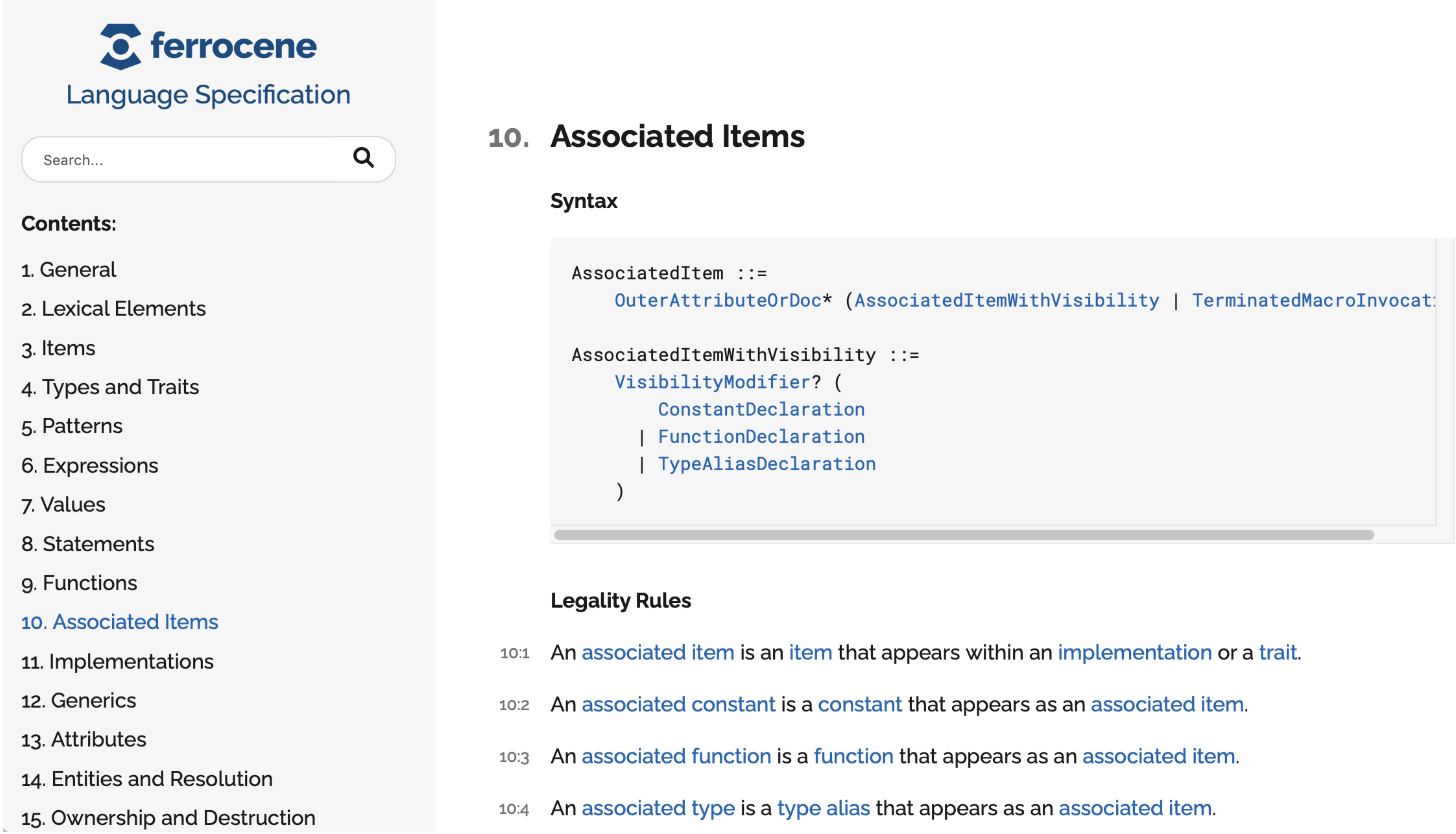
- sits downstream of The Rust Project
- is not a fork
- has sent all its bug-fixes upstream
- hosts some additional targets that upstream can’t host
- has a big announcement coming on 4 October

Confidence in your Tools

- What is the compiler supposed to do?
- Does it do what it is supposed to do?
- Does someone I trust believe it does what it is supposed to do?
- Can I get support and bug-fixes?

What is the compiler supposed to do?

- Rust doesn’t have a written specification (yet)
- So we wrote the Ferrocene Language Specification:
  - https://spec.ferrocene.dev/



ferrocene

Language Specification

Contents:

10. Associated Items

Syntax

```
AssociatedItem ::=
    OuterAttributeOrDoc* (AssociatedItemWithVisibility | TerminatedMacroInvocat

AssociatedItemWithVisibility ::=
    VisibilityModifier? (
        ConstantDeclaration
      | FunctionDeclaration
      | TypeAliasDeclaration
    )
```

Legality Rules

10:1 An associated item is an item that appears within an implementation or a trait.

10:2 An associated constant is a constant that appears as an associated item.

10:3 An associated function is a function that appears as an associated item.

10:4 An associated type is a type alias that appears as an associated item.

Does it do what it is supposed to do?

- Rust already had an **excellent** compiler test suite!
- Our work was mainly **joining the dots** between the tests and the specification, and **automating everything** (even the doc signing)
- **Nothing hits our main branch unless all the tests pass**
- We then documented everything in our new **Safety Manual**

Does someone I trust believe it does what it is supposed to do?

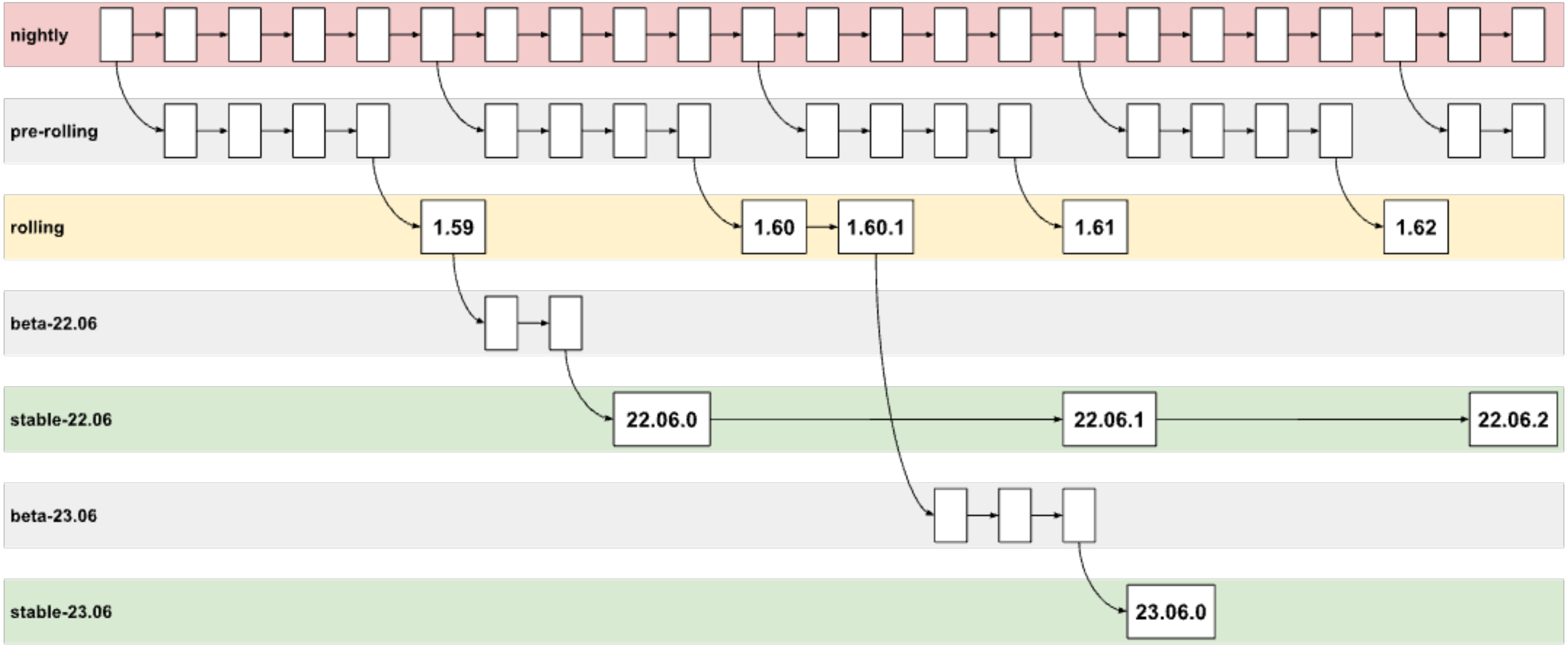
- We sent all our documents to TÜV SÜD for ISO 26262 approvals
- You might have your own approvals body…

> “Ferrocene has been qualified to be used in safety-related software development according to ISO 26262”

– TÜV Süd

Can I get support and bug-fixes?

- Ferrocene offers *long-term support*



ferrous systems

Case Study: Porting Rust to a new platform

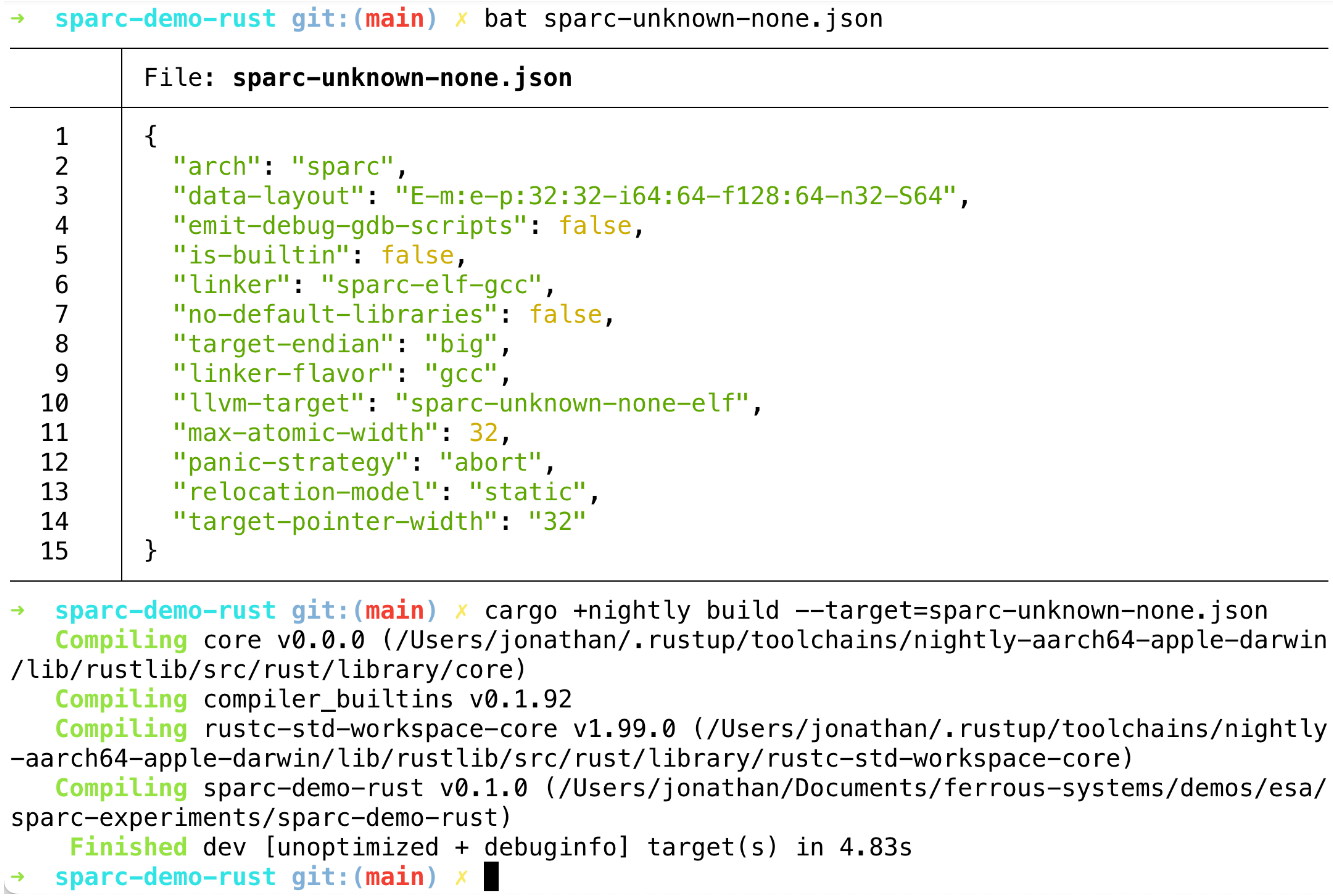
Case Study: Porting Rust to a new platform

- Rust works great with ARM Cortex-M
  - Lots of tools, libraries, sample projects
  - But that’s a boring demo
- Rust uses LLVM to generate machine code
- LLVM supports: Arm, Intel, PowerPC, MIPS … and SPARC?
- But Rust only supported SPARC64 on Linux…

A bit more detail on Rust…

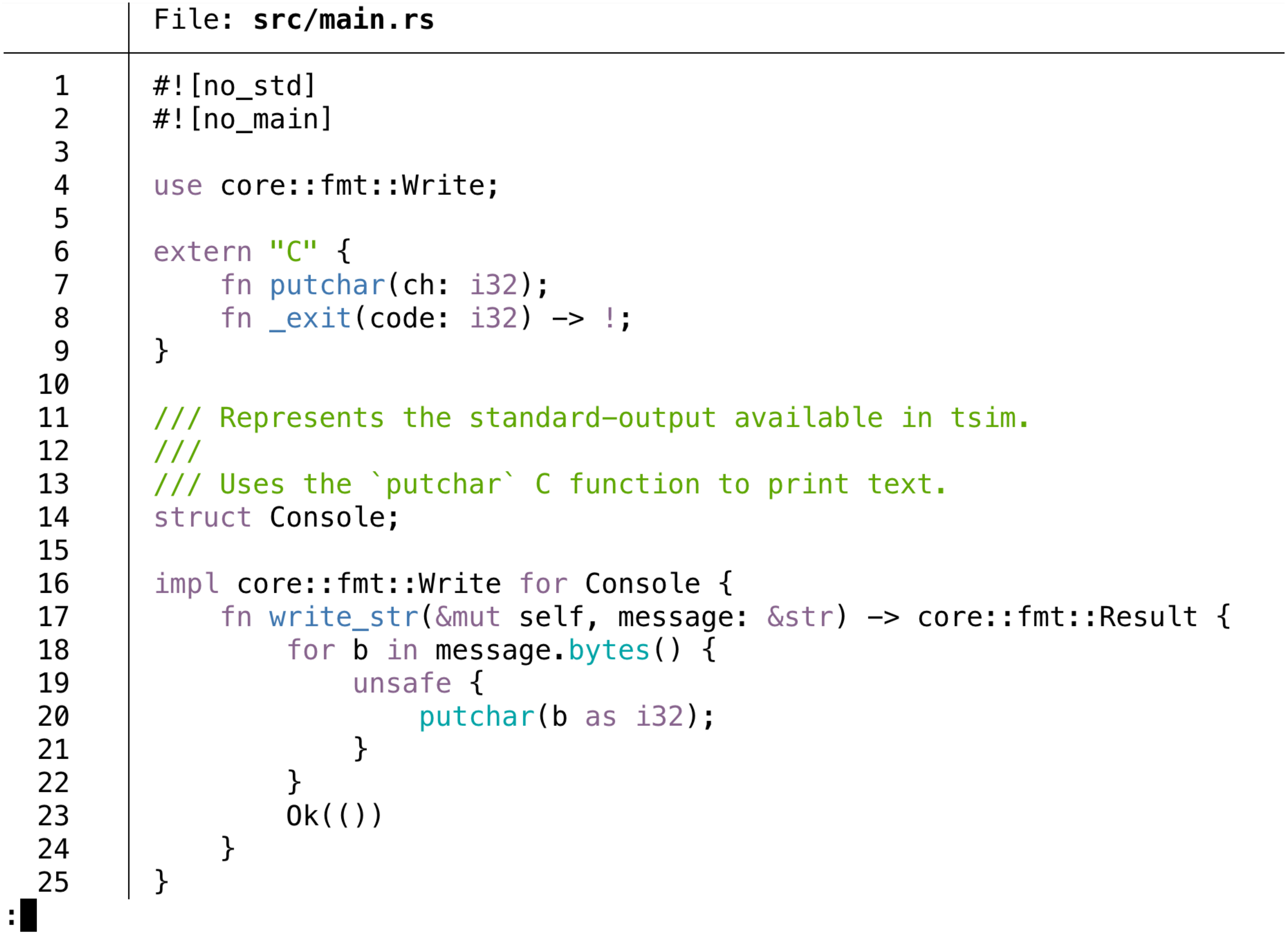
- Rust has **targets** - which describe the linker and CPU architecture to use:
  - Some targets are **built-in**
  - But **new targets** can be loaded at compile time
- Rust has both **libstd**, which needs an OS, and the smaller **libcore**, which does not

Teaching Rust bare-metal SPARC



```
➜  sparc-demo-rust git:(main) ✗ bat sparc-unknown-none.json
───────┬──────────────────────────────────────────────────────
       │ File: sparc-unknown-none.json
───────┼──────────────────────────────────────────────────────
       │ {
       │   "arch": "sparc",
       │   "data-layout": "E-m:e-p:32:32-i64:64-f128:64-n32-S64",
       │   "emit-debug-gdb-scripts": false,
       │   "is-builtin": false,
       │   "linker": "sparc-elf-gcc",
       │   "no-default-libraries": false,
       │   "target-endian": "big",
       │   "linker-flavor": "gcc",
       │   "llvm-target": "sparc-unknown-none-elf",
       │   "max-atomic-width": 32,
       │   "panic-strategy": "abort",
       │   "relocation-model": "static",
       │   "target-pointer-width": "32"
       │ }
───────┴──────────────────────────────────────────────────────
➜  sparc-demo-rust git:(main) ✗ cargo +nightly build --target=sparc-unknown-none.json
   Compiling core v0.0.0 (/Users/jonathan/.rustup/toolchains/nightly-aarch64-apple-darwin
/lib/rustlib/src/rust/library/core)
   Compiling compiler_builtins v0.1.92
   Compiling rustc-std-workspace-core v1.99.0 (/Users/jonathan/.rustup/toolchains/nightly
-aarch64-apple-darwin/lib/rustlib/src/rust/library/rustc-std-workspace-core)
   Compiling sparc-demo-rust v0.1.0 (/Users/jonathan/Documents/ferrous-systems/demos/esa/
sparc-experiments/sparc-demo-rust)
    Finished dev [unoptimized + debuginfo] target(s) in 4.83s
➜  sparc-demo-rust git:(main) ✗ █
```

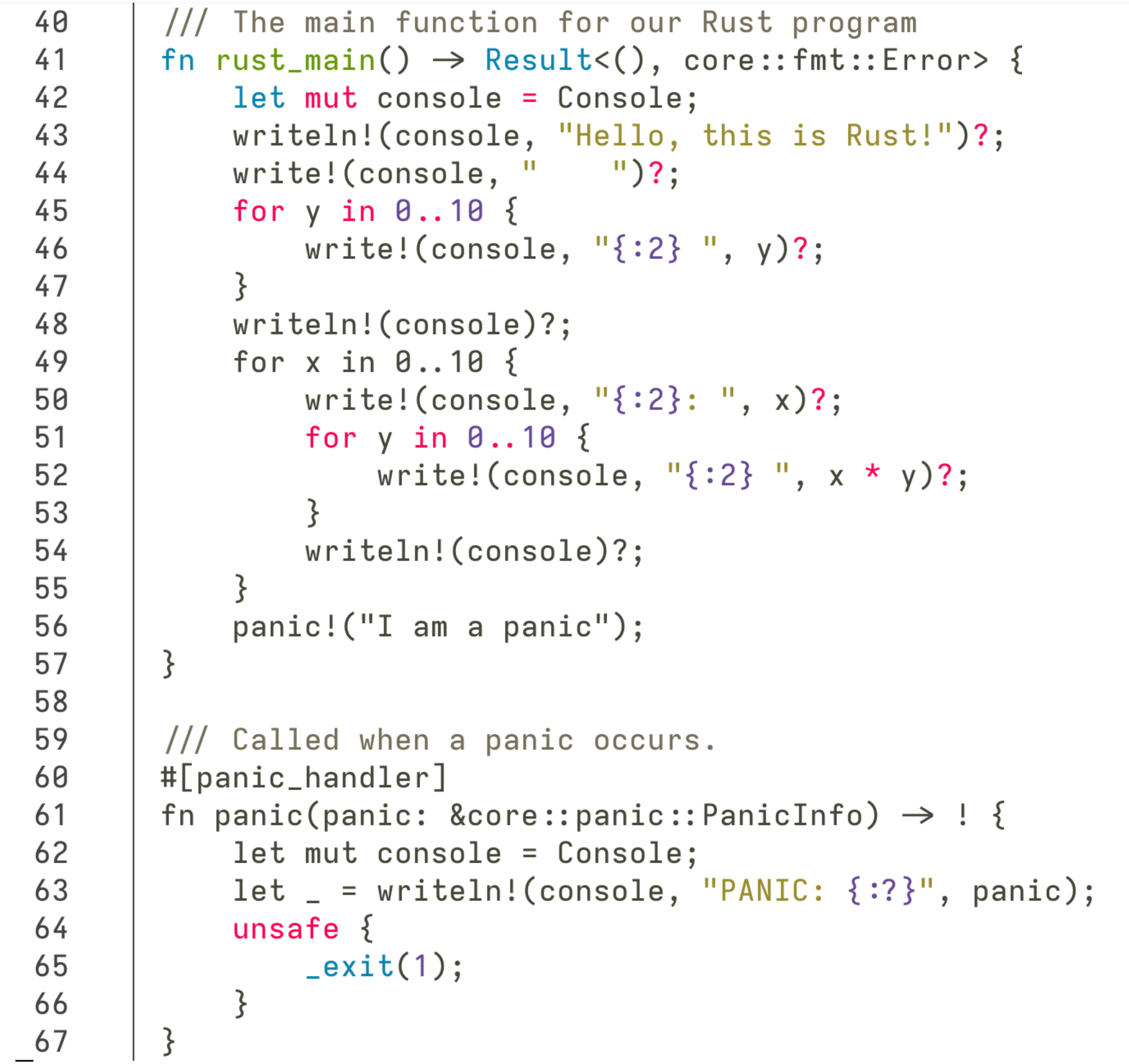
ferrous systems



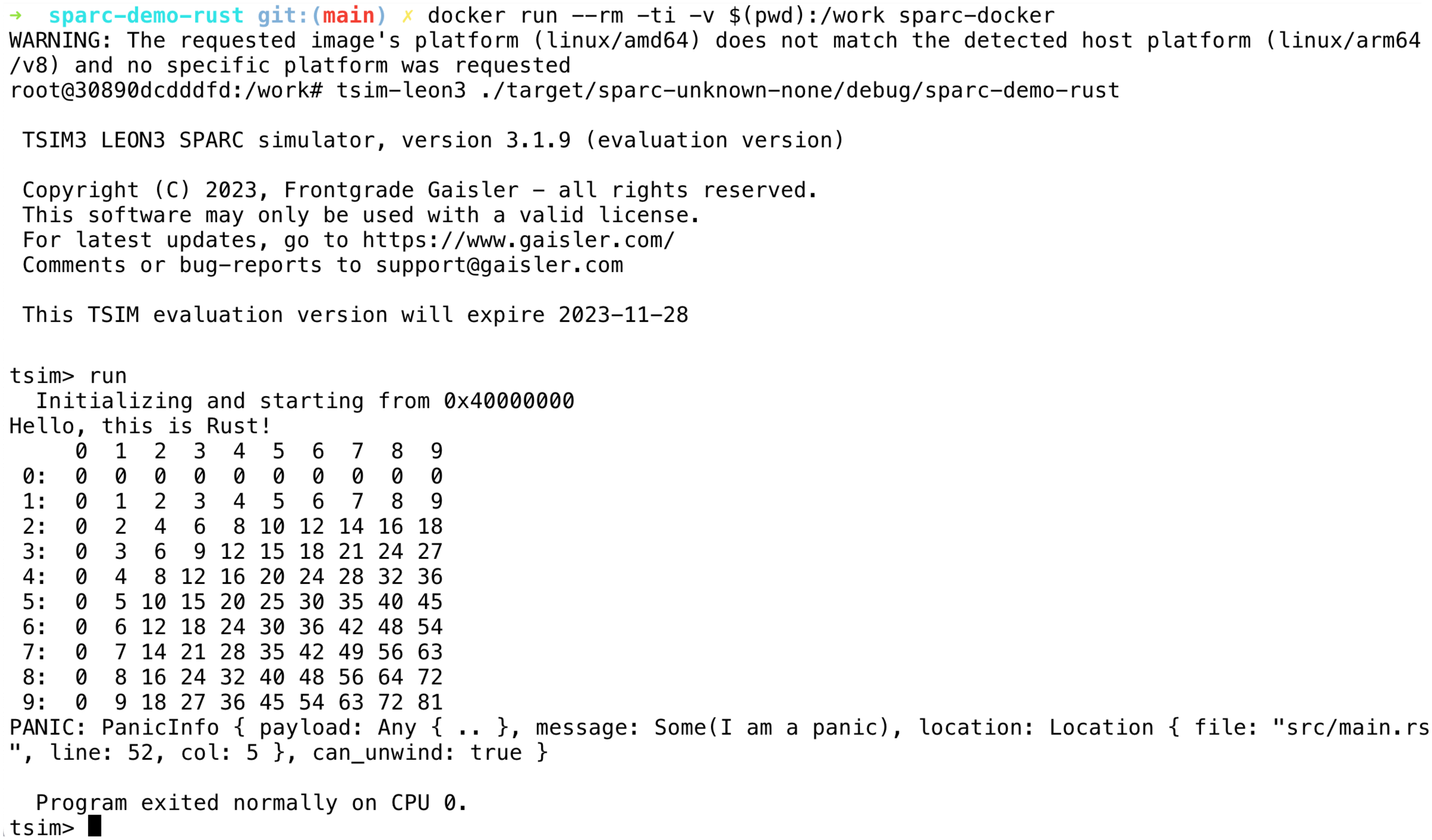
```
File: src/main.rs
```

```rust
#![no_std]
#![no_main]

use core::fmt::Write;

extern "C" {
    fn putchar(ch: i32);
    fn _exit(code: i32) -> !;
}

/// Represents the standard-output available in tsim.
///
/// Uses the `putchar` C function to print text.
struct Console;

impl core::fmt::Write for Console {
    fn write_str(&mut self, message: &str) -> core::fmt::Result {
        for b in message.bytes() {
            unsafe {
                putchar(b as i32);
            }
        }
        Ok(())
    }
}
```

```
:
```

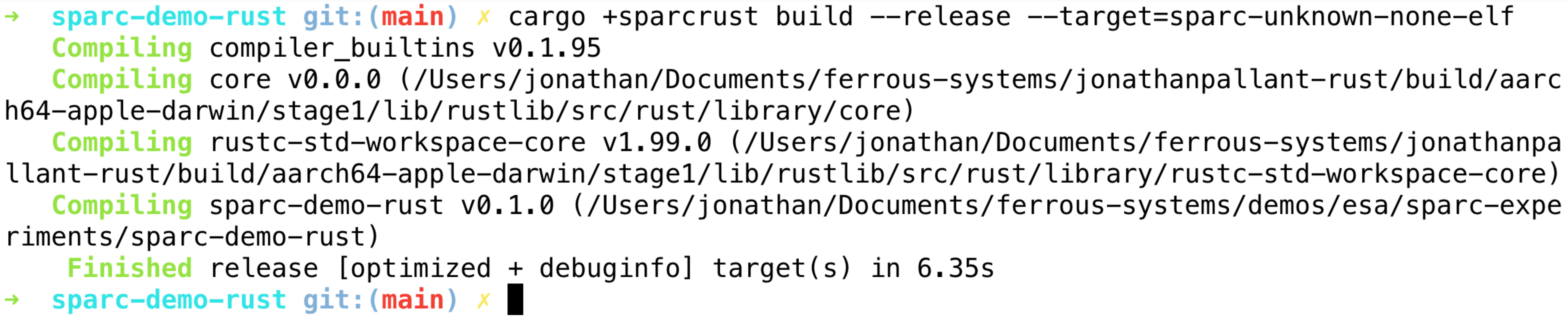


```rust
/// The main function for our Rust program
fn rust_main() -> Result<(), core::fmt::Error> {
    let mut console = Console;
    writeln!(console, "Hello, this is Rust!")?;
    write!(console, "    ")?;
    for y in 0..10 {
        write!(console, "{:2} ", y)?;
    }
    writeln!(console)?;
    for x in 0..10 {
        write!(console, "{:2}: ", x)?;
        for y in 0..10 {
            write!(console, "{:2} ", x * y)?;
        }
        writeln!(console)?;
    }
    panic!("I am a panic");
}

/// Called when a panic occurs.
#[panic_handler]
fn panic(panic: &core::panic::PanicInfo) -> ! {
    let mut console = Console;
    let _ = writeln!(console, "PANIC: {:?}", panic);
    unsafe {
        _exit(1);
    }
}
```



```
➜  sparc-demo-rust git:(main) ✗ docker run --rm -ti -v $(pwd):/work sparc-docker
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64
/v8) and no specific platform was requested
root@30890dcdddfd:/work# tsim-leon3 ./target/sparc-unknown-none/debug/sparc-demo-rust

 TSIM3 LEON3 SPARC simulator, version 3.1.9 (evaluation version)

 Copyright (C) 2023, Frontgrade Gaisler - all rights reserved.
 This software may only be used with a valid license.
 For latest updates, go to https://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 This TSIM evaluation version will expire 2023-11-28


tsim> run
  Initializing and starting from 0x40000000
Hello, this is Rust!
     0  1  2  3  4  5  6  7  8  9
 0:  0  0  0  0  0  0  0  0  0  0
 1:  0  1  2  3  4  5  6  7  8  9
 2:  0  2  4  6  8 10 12 14 16 18
 3:  0  3  6  9 12 15 18 21 24 27
 4:  0  4  8 12 16 20 24 28 32 36
 5:  0  5 10 15 20 25 30 35 40 45
 6:  0  6 12 18 24 30 36 42 48 54
 7:  0  7 14 21 28 35 42 49 56 63
 8:  0  8 16 24 32 40 48 56 64 72
 9:  0  9 18 27 36 45 54 63 72 81
PANIC: PanicInfo { payload: Any { .. }, message: Some(I am a panic), location: Location { file: "src/main.rs
", line: 52, col: 5 }, can_unwind: true }

  Program exited normally on CPU 0.
tsim> █
```

Can we make this target a built-in?



```
➜  sparc-demo-rust git:(main) ✗ cargo +sparcrust build --release --target=sparc-unknown-none-elf
   Compiling compiler_builtins v0.1.95
   Compiling core v0.0.0 (/Users/jonathan/Documents/ferrous-systems/jonathanpallant-rust/build/aarc
h64-apple-darwin/stage1/lib/rustlib/src/rust/library/core)
   Compiling rustc-std-workspace-core v1.99.0 (/Users/jonathan/Documents/ferrous-systems/jonathanpa
llant-rust/build/aarch64-apple-darwin/stage1/lib/rustlib/src/rust/library/rustc-std-workspace-core)
   Compiling sparc-demo-rust v0.1.0 (/Users/jonathan/Documents/ferrous-systems/demos/esa/sparc-expe
riments/sparc-demo-rust)
    Finished release [optimized + debuginfo] target(s) in 6.35s
➜  sparc-demo-rust git:(main) ✗ █
```

Yes we can. Upstreaming complete!

https://github.com/rust-lang/rust/pull/113535

ferrous systems

Bare-metal SPARC for everyone

`cargo --target sparc-unknown-none-elf` now works on nightly.

See https://doc.rust-lang.org/nightly/rustc/platform-support.html

sparc-unknown-none-elf	*	Bare 32-bit SPARC V7+

(It also works on the GR765 LEON 5 prototype, and in RTEMS)

If you want it in Ferrocene, let's talk!

Any Questions?

(https://github.com/ferrous-systems/sparc-experiments/)

Dead Code and Coverage

- Dead Code within a crate is a warning (can be an error)
- Dead Code in a binary (i.e. pub export from a library but unused) is removed by the LLVM optimiser (and we can do LTO)
- cargo-tarpaulin can do code coverage
  - Uses LLVM tooling
  - MC/DC is work in progress

Training and Support

- Ferrocene from Ferrous Systems
- GNAT Pro for Rust from AdaCore
- Several other training providers and consultancies around
- Many excellent on-line training courses too

Is there a MISRA for Rust?

- You’d have to ask MISRA (but I don’t think so)
- The language defaults are so good, most people don’t need to tie it down any further
- But if you do, we have `#[deny(rule)]` (+ `allow`, `warn`, and `forbid`) with a large number of built-in rules … e.g. `#[forbid(unsafe-code)]`
- https://doc.rust-lang.org/rustc/lints/index.html

Testing

- Unit Tests are compiled into your crate (can see private API)
- Integration Tests are compiled outside your crate (can only see public API)
- Documentation Tests compile and run the ``` code blocks in your doc comments
- Ferrous System has a tool for running tests on bare-metal targets

C and C++

- Rust can call C compatible functions (we saw this in the demo)
- Rust can generate C compatible functions
- Tools are available to auto-generate matching pairs of C++/Rust objects, and the appropriate, safe, C compatible conversion code for each side (https://crates.io/crates/cxx)

RTOS Support

- RTIC - a real-time framework written in Rust with guaranteed WCET
- FreeRTOS - bindings available (e.g. Espressif IDF)
- LynxOS-178 - we wrote the bindings for Lynx
- QNX and VxWorks - supported upstream
- RTEMS - I wrote a C binary with RCC and linked a Rust example to it