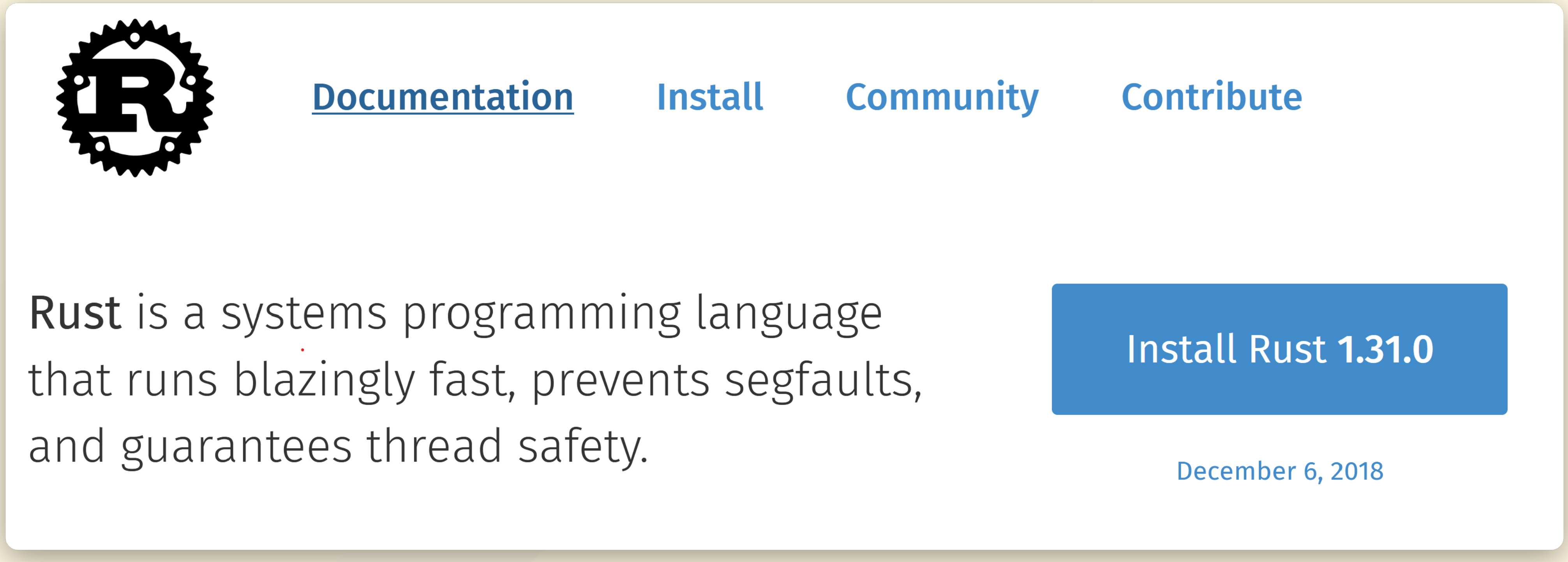
What do you even mean "Empowering Everyone"?!

Or: Give Embedded A Chance

Ye' Olden Days: prev.rust-lang.org



Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Pre-2019 Rust Website

What was Pre-2018 Rust Like?

2018 Edition was the first ever edition.

- Parallel Codegen, incremental compilation
- The `try!` operator was superseded by `?`
- The module system was funny (`extern crate ...`)
- `impl Trait`, `NLL`, Clippy
- `async` and `await` not keywords yet!

Occasionally, builds would break.

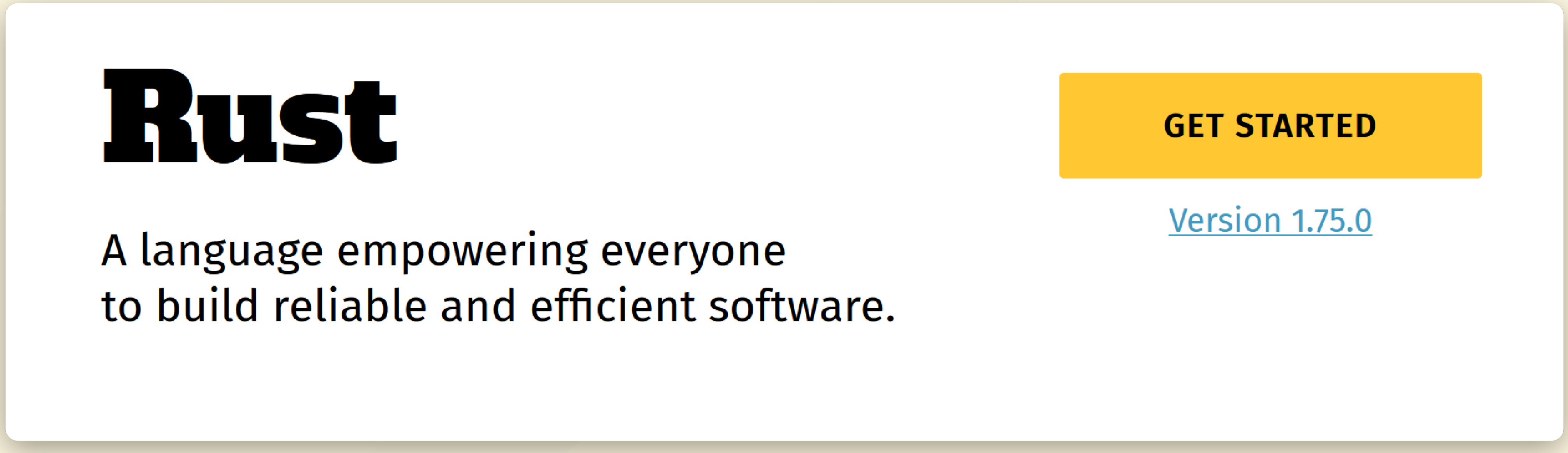
https://blog.rust-lang.org/2020/05/15/five-years-of-rust.html

`"prevents segfaults"`

- What does it even mean `SEGFAULT`?
  Does the *absence* of something define a language?!
- "🚀🚀🚀 blazingly fast 🚀🚀🚀" has become a meme
  Don't put this in your project `README.md`
- Guaranteed Thread Safety is also pretty technical

Overall, yes these are a big deal, but this slogan had to evolve.

Current rust-lang.org



Rust

A language empowering everyone
to build reliable and efficient software.

Current Rust Language website

"Empowering Everyone"

It's not just some inclusivity statement
(though the community is pretty diverse-friendly).

This is the **long-term goal** (as in, not yet fulfilled).

> Rust will erase the boundary between system and application development

-- Somebody on Reddit

The system vs. application development boundary

Perceived boundary:

- Systems care about
  - rigorous design invariants (thread/memory safety)
  - performance
- Applications tend to care more about:
  - fast development
  - maintainability
  - simplicity

It's mostly a false dichotomy

There is no meaningful distinction (anymore).

Only slightly more interesting:
Are you working on a `lib.rs` or a `main.rs`?

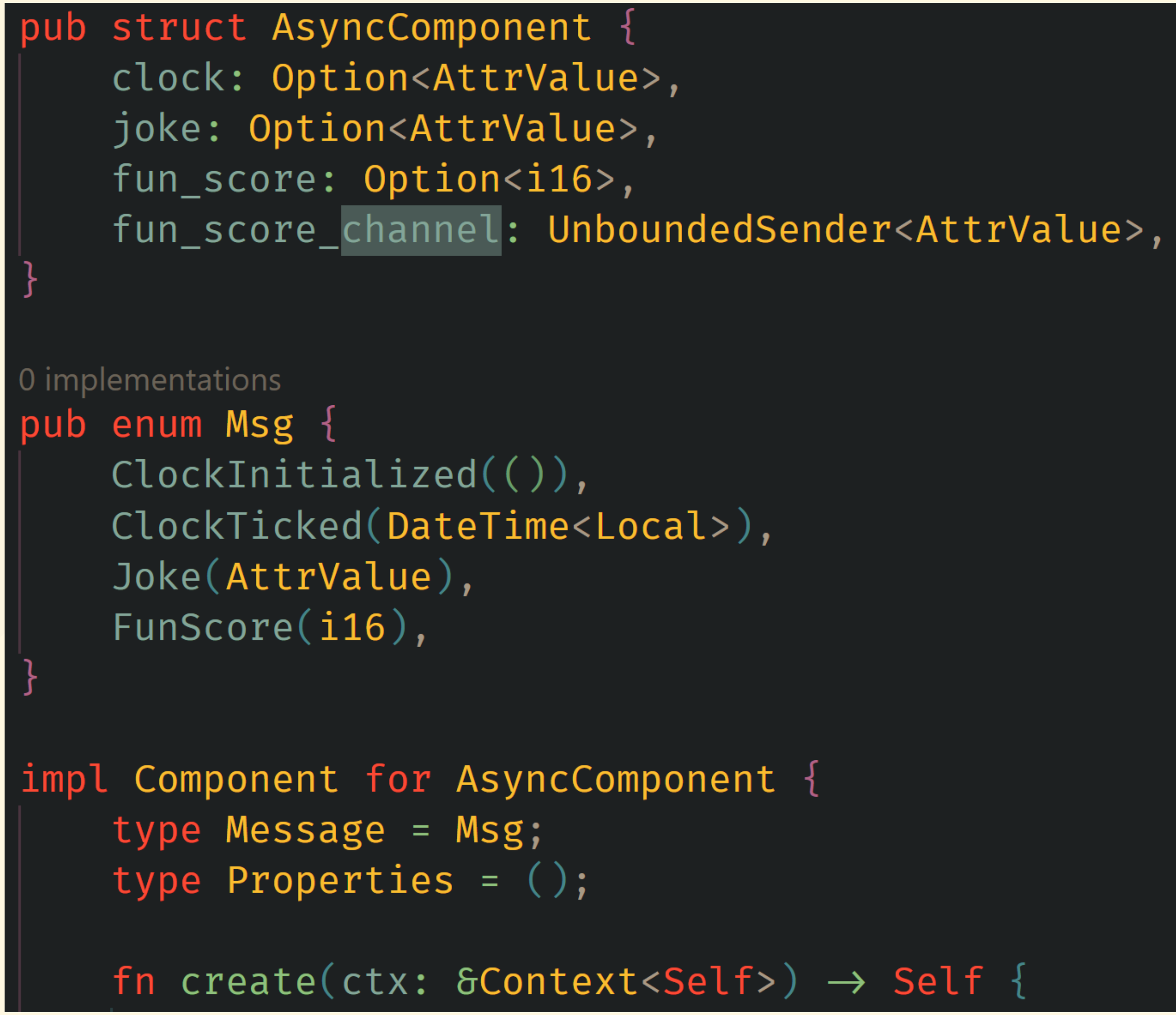
Realistically: both.

Aside: Yew

No boundary, because:
Frontend looks like
System Development!

That's maybe
not the goal.
Or is it?

Not dunking on yew,
it's cool



```
pub struct AsyncComponent {
    clock: Option<AttrValue>,
    joke: Option<AttrValue>,
    fun_score: Option<i16>,
    fun_score_channel: UnboundedSender<AttrValue>,
}

0 implementations
pub enum Msg {
    ClockInitialized(()),
    ClockTicked(DateTime<Local>),
    Joke(AttrValue),
    FunScore(i16),
}

impl Component for AsyncComponent {
    type Message = Msg;
    type Properties = ();

    fn create(ctx: &Context<Self>) → Self {
```

async-clock example for yew on github

If there is no boundary: everybody can do embedded programming

The public "image" of embedded has false admiration/despise.

It's not exactly easy, but can be rewarding!

Recently, it's gotten easier because of two events.

Embedded HAL 1.0 Release (Jan. 9th 2024)

The Hardware Abstraction Layer allows for writing hardware-agnostic drivers.

Previously, the ecosystem was furiously changing. 1.0 was bikeshedded for 4 years.

Now:
So many drivers

So many BSPs

Async on Microcontrollers: embassy.dev

Imagine `tokio`, but on microcontrollers.

Why does this make sense? `async` is for I/O, and on a microcontroller EVERYTHING is I/O!
Even the passing of time.

FINALLY embassy is on crates.io! (Jan. 22nd 2024)

Embassy Example

Simple things are simple:

```
let mut button = Input::new(p.PIN_16, Pull::None);
loop {
    button.wait_for_high().await;

    info!("Toggle LED");
    led.toggle();

    Timer::after_millis(500).await;
}
```

Embassy Example 2

Complicated things are possible:

```rust
loop {
    match control.join_wpa2(WIFI_NETWORK, WIFI_PASSWORD).await {
        Ok(_) => break,
        Err(err) => info!("join failed with status={}", err.status),
    }
}
```

Takeaways

- Rust has come a long way
- The "systems" aspects are converging with the "approachability" aspects
- The ecosystem is glorious, but it is bleeding edge
- Embedded programming can be painless and enjoyable.

Thanks :) and have fun