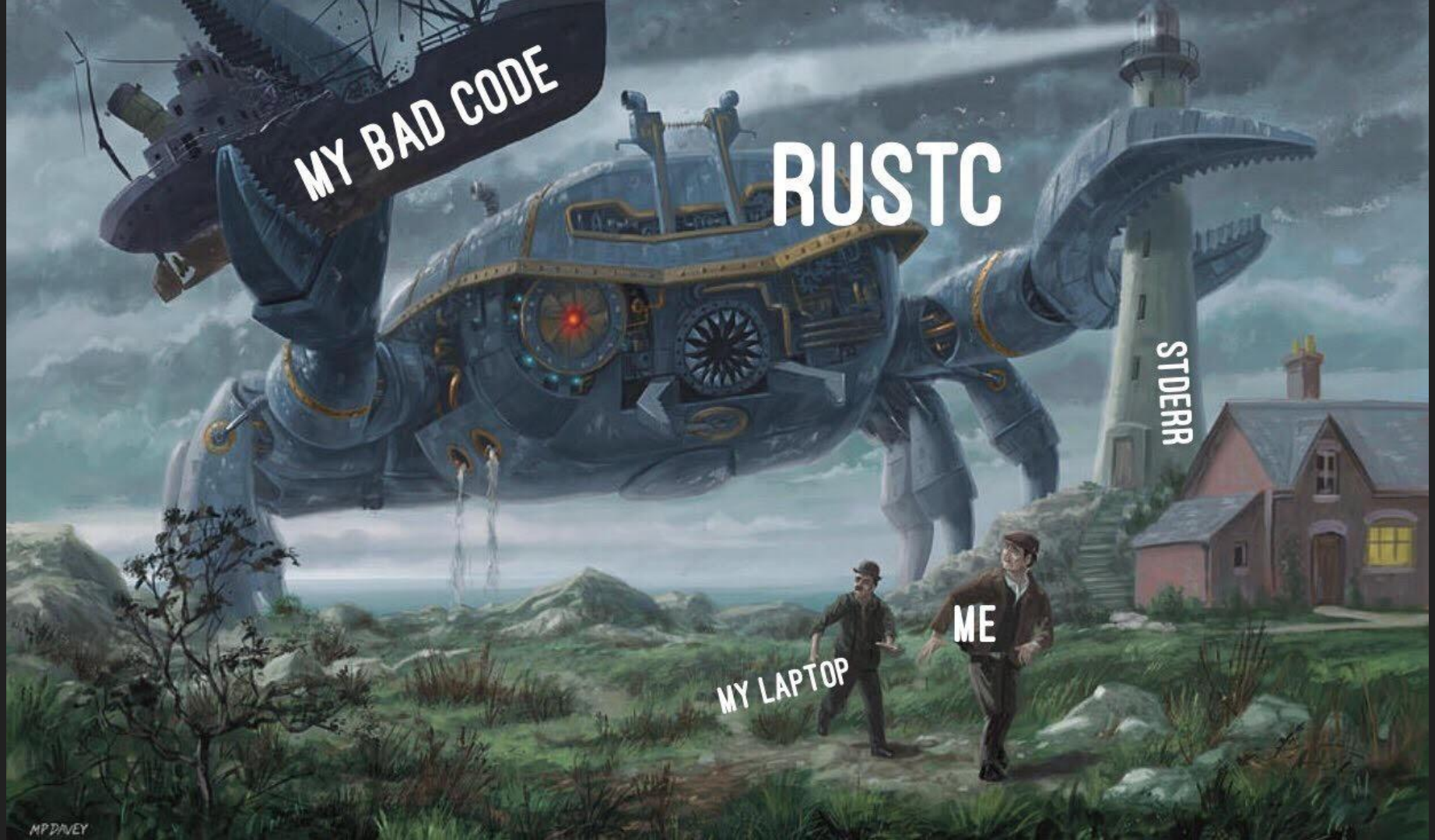
impl Trait!

Kevin Martin S1 - 2018
@6b766e

This talk assumes some previous familiarity with Rust, we unfortunately will not have time to cover some of the foundational concepts in play such as traits or closures, but I will try and make this easy to follow even if you are new to Rust.

These slides and the accompanying examples can be found at:

https://github.com/data-pup/impl-trait-talk


MY BAD CODE
RUSTC
STDERR
ME
MY LAPTOP
MP DAVEY

Rust Documentation Tip

Before we delve into `impl Trait`, I want to pass a helpful trick along that I wish I knew earlier on while learning Rust. `rustup` is the tool used to install and manage Rust toolchains, and it will also help you navigate documentation for Rust!

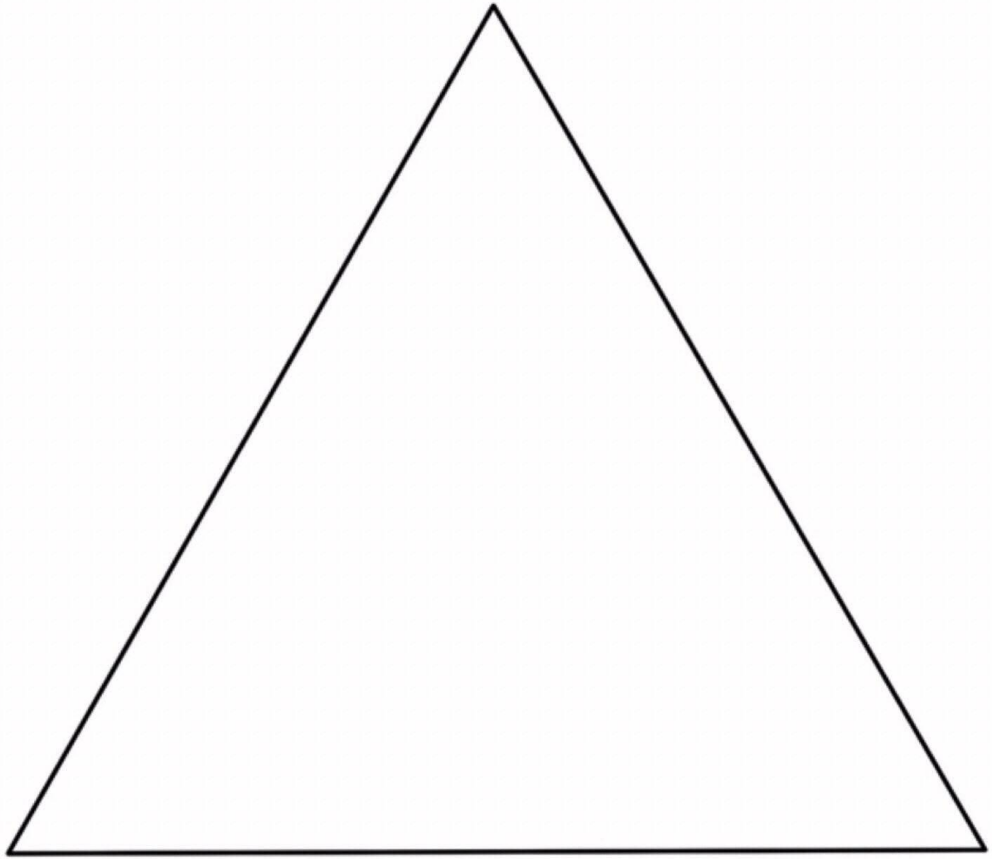
If you'd like to read the Rust Programming Language book yourself, you can use this command:

rustup doc --book

Documentation for the standard library can also be found with this command:

rustup doc --std

we use stable



runs “rustup update” as the devs cheer, adds a single inclusive range, commits

impl Trait! impl Trait!

impl Trait RFC’s

The details of `impl Trait` are divided into a few different RFC's. If you would like to browse Rust language RFC's you can find them at https://github.com/rust-lang/rfcs. It is a nice look into the process of how substantial changes are made to a language, tooling, etc., when the discussion required outgrows the typical pull request workflow offered by GitHub.

‘impl Trait’ landed in the stable build of Rust in v1.26, in May 2018. This work goes back to June 2016, so this feature has been in development for a long time!

The specific RFC related to this feature are:

- RFC 1522 - The original RFC, only covered `impl Trait` in the return position for inherent (free-standing) functions.
- RFC 1951 - Syntax design, added `impl Trait` to the argument position.
- RFC 2071 - Use of `impl Trait` in `let`, `const`, and `static`
- RFC 2250 - Finalizing syntax of `impl Trait`

What is impl Trait?

The ‘impl Trait’ feature provides the idea of “existential types,” which sounds complex but this makes more sense when it is demonstrated. Essentially, it means that we might not know the specific type of an object, but we do know a trait that it implements.

If you are familiar with interfaces in C#, Go, or TypeScript, this might look familiar when you see an example. We will briefly cover some of the use cases for this, and then move on to a quick demo!

When is impl Trait useful?

Previously, we could work around the lack of this feature by using Boxes. This is essentially Rust's term for a pointer. The downside to this however, is that it requires allocating space on the heap, which is much slower than using the stack.

Aside from performance benefits, this is useful when we are working with code that uses iterator adaptors such as 'map' or 'filter', especially when the adaptors might change for different conditions.

'impl Trait' also makes it much easier to return closures!