

By Matthias Endler



# The Rust 2024 Edition and Beyond





# Agenda

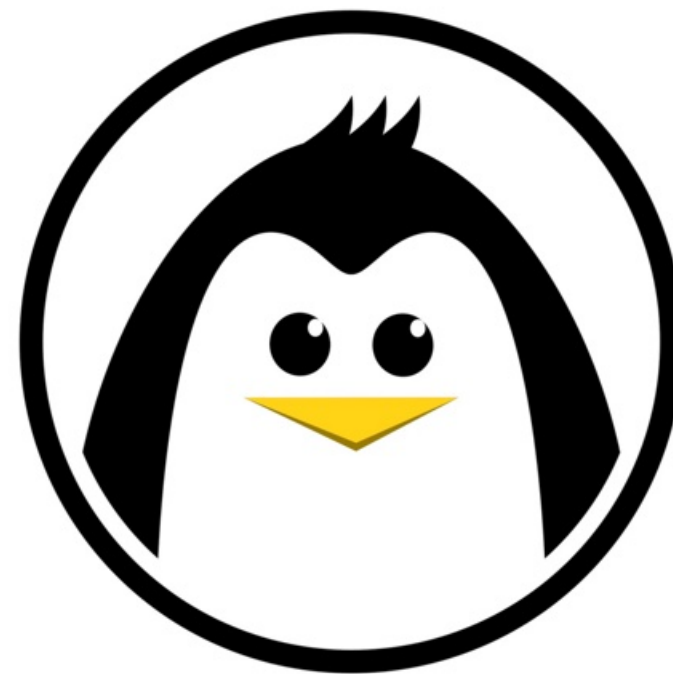
- 01. About Me
- 02. Rust's past
- 03. Editions
- 04. Rust 2024
- 05. Rust's future





# ~ ~ About me ~ ~

Rust consultant (corrode.dev)  
Living in Düsseldorf  
Open Source Maintainer (lychee)  
'Rust in Production' podcast





01



May 4, 2016



02

# Rust's Past



2006-2009: Personal project by Graydon Hoare

2009-2012: Mozilla Sponsorship

2012-2015: Massive change

15th of May 2015: Rust 1.0 release

119 Releases since then

(84 feature + 35 patch releases)





02

# Rust's Past



Steve Klabnik: The History of Rust





02

# Rust's Past



Garbage Collector

Runtime

Green Threads

Lots of different pointer types





02

# Rust's Past



```
let x = @5; // GC'd pointer  
let y = ~5; // unique pointer  
let z = &5; // borrowed pointer
```





# 02

```
obj counter(int i) {  
    fn incr() {  
        i += 1;  
    }  
    fn get() → int {  
        ret i;  
    }  
}  
  
fn main() {  
    auto c = counter(10);  
    c.incr();  
    log c.get();  
}
```





# 02

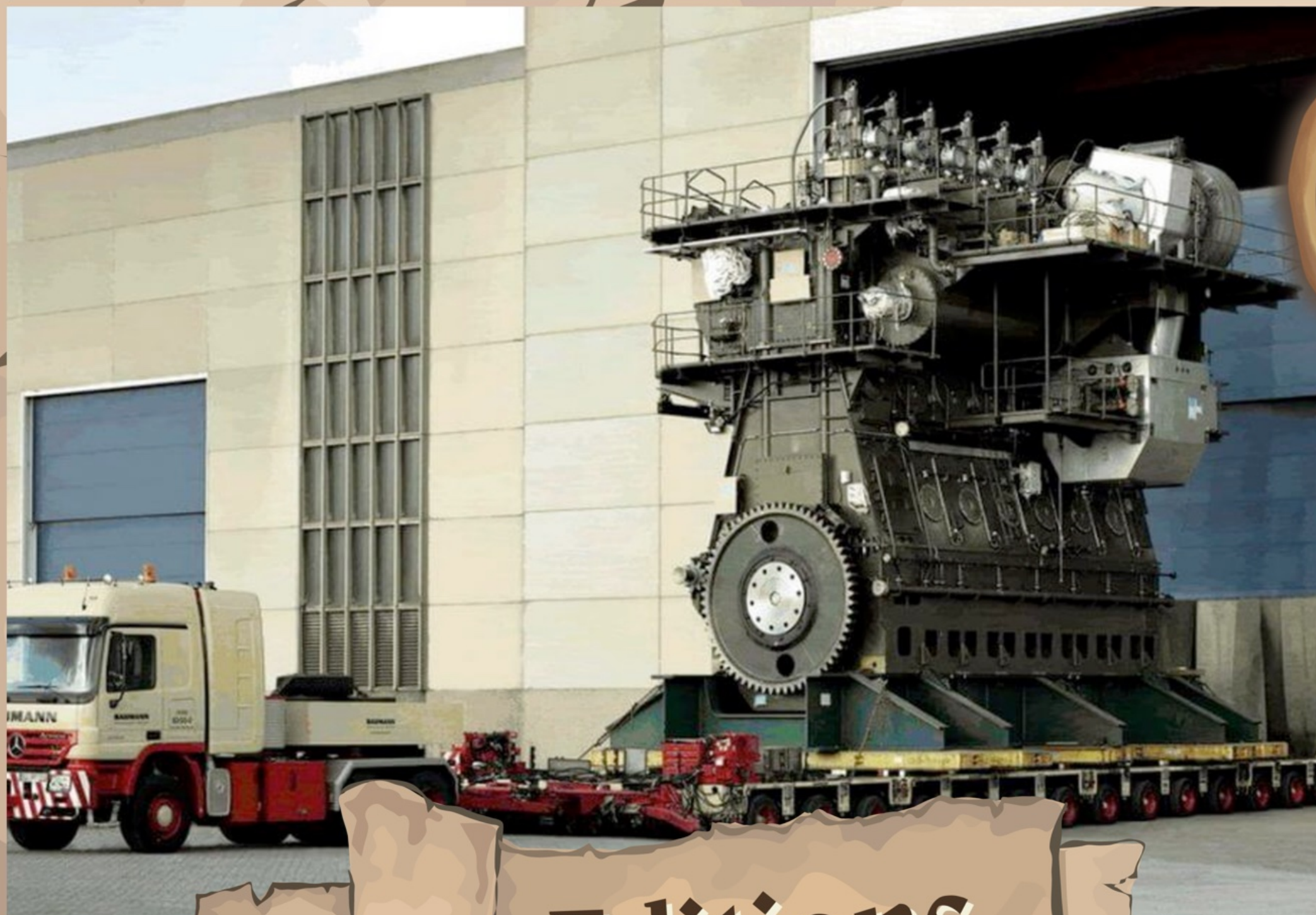
```
obj swap[T](tup(T,T) pair) → tup(T,T) {  
    ret tup(pair._1, pair._0);  
}
```

```
fn main() {  
    auto str_pair = tup("hi", "there");  
    auto int_pair = tup(10, 12);  
    str_pair = swap[str](str_pair);  
    int_pair = swap[int](int_pair);  
}
```





03



Editions



03

# Objectives

## Stability

Rust code keeps compiling on newer versions of the compiler.

## Progress

New features can be introduced.



# 03

# Prior Editions

<https://doc.rust-lang.org/edition-guide>



## 2015

Rust 1.0

## 2018

- non-lexical lifetimes
- impl Trait
- Module system improvements

## 2021

- Intolter for arrays
- TryFrom in prelude
- Closures only capture individual fields





03

# Methodology



Editions do not split the ecosystem!

Different editions can be mixed!

You can use 2018 code in 2024 projects.

Secret sauce: the compiler!

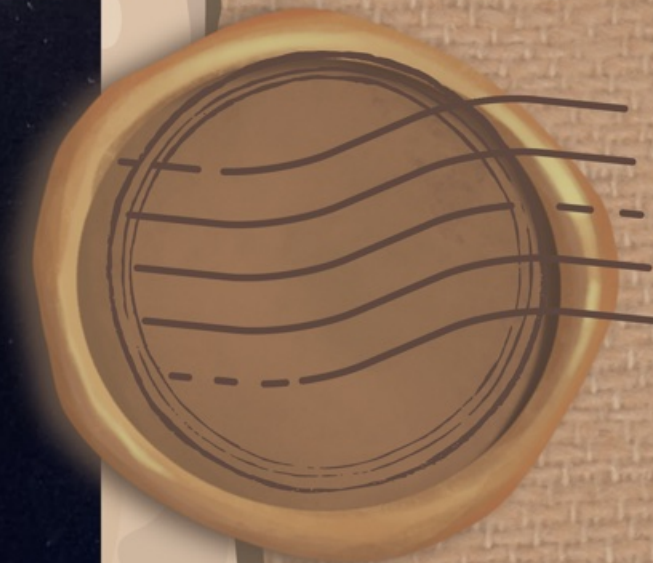
Updating is easy: change edition in

Cargo.toml and run ``cargo fix --edition``





04



Rust 2024





# Rust 2024

- The Future and IntoFuture traits are now part of the prelude.





# Rust 2024

- std::env::{set var,  
remove var} is now  
unsafe (in  
multithreaded  
environments)





# Rust 2024

- Cargo: Reject unused inherited default-features





# Rust 2024

```
[package]
name = "foo"
version = "1.0.0"
edition = "2024"

[dependencies]
regex = { workspace = true, default-features = false } # ERROR
```

The reason for this change is to avoid confusion when specifying `default-features = false` when the default feature is already enabled, since it has no effect.



# Rust 2024

reserve the gen keyword





# 04

## How the compiler works

```
79     let import_path = root
80         .iter()
81         .chain(&[name, sym::prelude])
82         .chain(&[match edition {
83             Edition2015 => sym::rust_2015,
84             Edition2018 => sym::rust_2018,
85             Edition2021 => sym::rust_2021,
86             Edition2024 => sym::rust_2024,
87         }])
88         .map(|&symbol| Ident::new(symbol, span))
89         .collect();
```

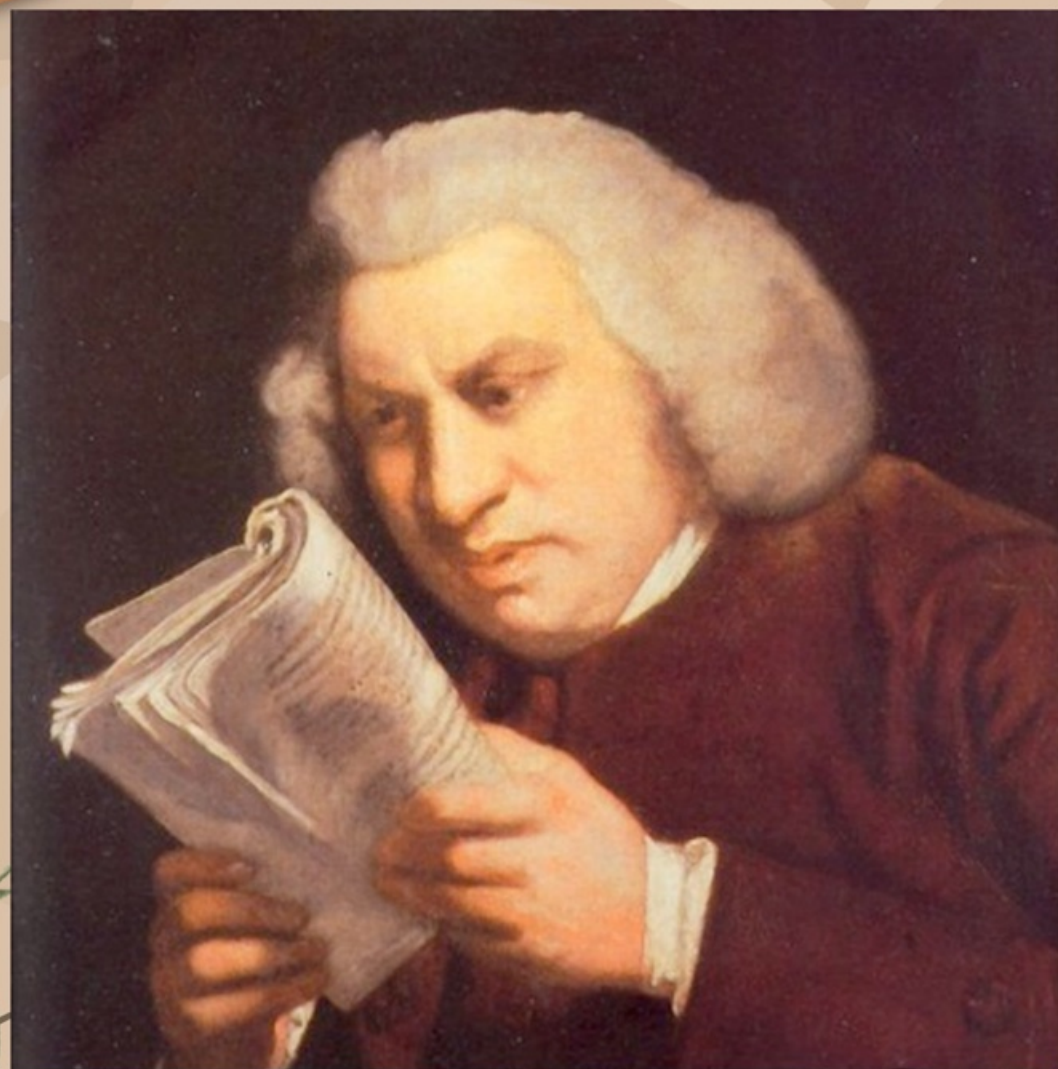
[Source Code](#)





That's  
basically  
it.



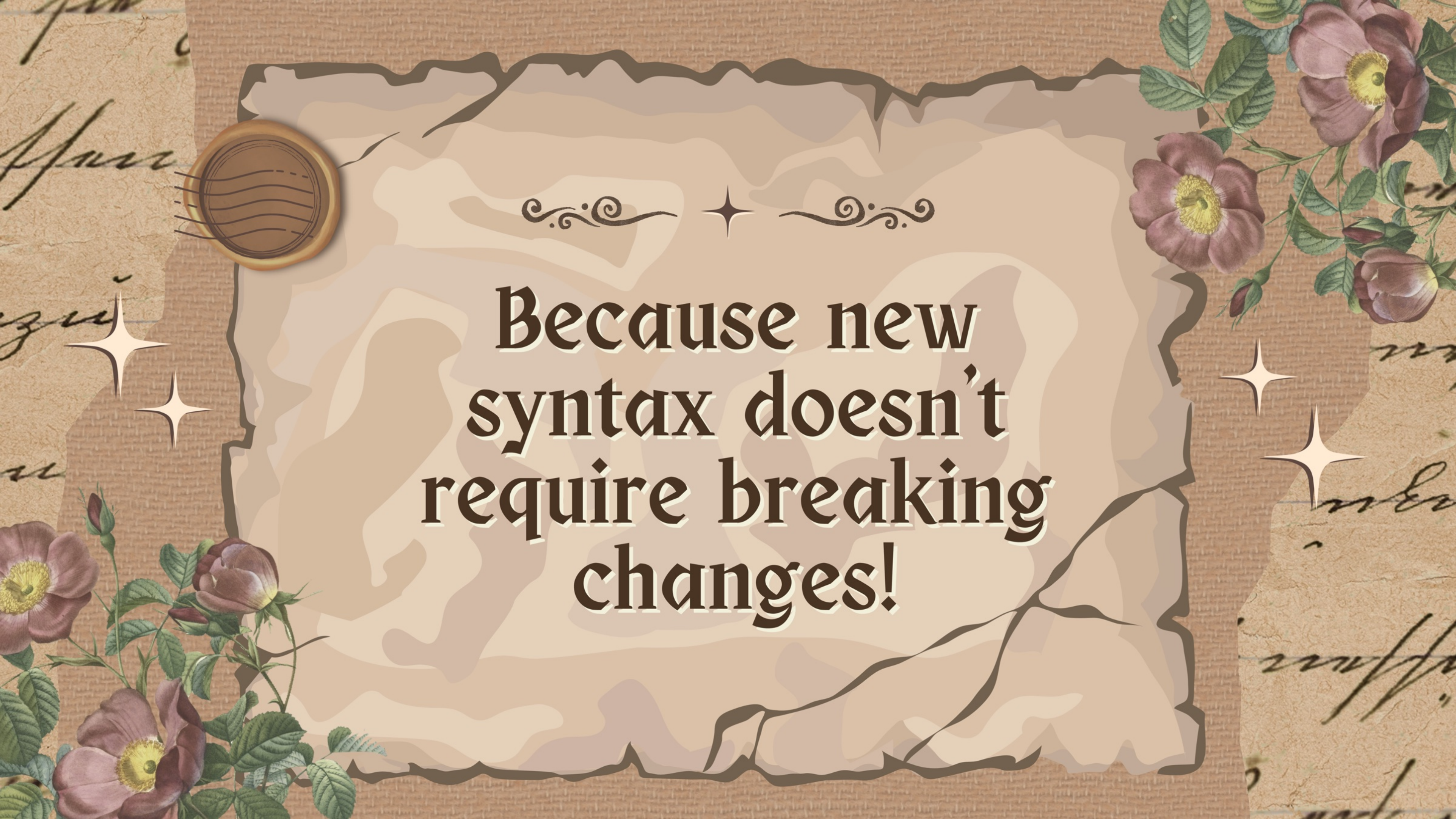






**Not much  
going on in  
editions**





Because new  
syntax doesn't  
require breaking  
changes!






# Rust 2024



Exception in 2024:  
reserve the gen keyword







Many big  
features were  
introduced in  
non-edition  
releases...



## Stabilize `async await` in Rust 1.39.0 #63209

Merged by bors `rust-lang:master` ← `Centril:stabilize-async-await` on Aug 21, 2019

Conversation 64

Commits 4

Checks 0

Files changed 181



Centril (Mazdak Farrokhzad) on Aug 2, 2019 • edited by sgrif

Contributor

Here we stabilize:

- free and inherent `async fn` s,
- the `<expr>.await` expression form,
- and the `async move? { ... }` block form.

Closes [#62149](#).

Closes [#50547](#).

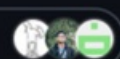
All the blockers are now closed.

► Details

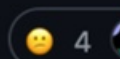
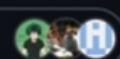
r? [@cramertj](#)



41



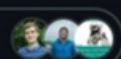
313



4



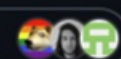
59



51





24








## Stabilize inclusive range (..<=) #47813

 Merged by bors `rust-lang:master` ← `kennytm:stable-incl-range` on Mar 15, 2018 1.26.0

 Conversation 58

 Commits 5

 Checks 0


 Files changed 39



kennytm on Jan 27, 2018 • edited

Member

Stabilize the followings:

- `inclusive_range` — The `std::ops::RangeInclusive` and `std::ops::RangeInclusiveTo` types, except its fields (tracked by  [Tracking issue for `start`, `end` and `new` methods of `RangeInclusive` #49022](#) separately).
- `inclusive_range_syntax` — The `a..=b` and `..=b` expression syntax
- `dotdoteq_in_patterns` — Using `a..=b` in a pattern

cc [#28237](#)

r? @rust-lang/lang



32



1







Sometimes we  
don't notice  
steady progress...



# Example: A lot more things can be const now



```
const fn factorial(n: u32) → u32 {  
    match n {  
        0 ⇒ 1,  
        n ⇒ n * factorial(n - 1)  
    }  
}  
const FACT_5: u32 = factorial(5); // Computed at compile time!
```



# Even more const things



```
const fn array_sum<const N: usize>(arr: &[i32; N]) → i32 {  
    let mut sum = 0;  
    let mut i = 0;  
    while i < N {  
        sum += arr[i];  
        i += 1;  
    }  
    sum  
}  
  
const NUMBERS: [i32; 4] = [1, 2, 3, 4];  
const TOTAL: i32 = array_sum(&NUMBERS);
```



# Moooar const



```
const MSG: &str = "Hello, Rust!";  
const IS_ASCII: bool = MSG.is_ascii();  
const BYTE_LEN: usize = MSG.as_bytes().len();
```



If you can't get enough of  
const...



[Check out more examples](#)



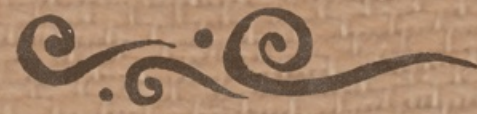


The background features a central rectangular area with a torn, layered paper effect in shades of beige and cream. This central area is framed by a dark brown, irregular border. In the top-left corner of the central area is a circular wax seal with a brown, textured appearance and concentric lines. The corners of the image are decorated with floral illustrations of pink roses with yellow centers and green leaves. Additionally, there are four white, four-pointed starburst or sparkles: two on the left side and two on the right side of the central text area. The entire composition is set against a background of aged, yellowed paper with faint, illegible cursive handwriting visible at the edges.

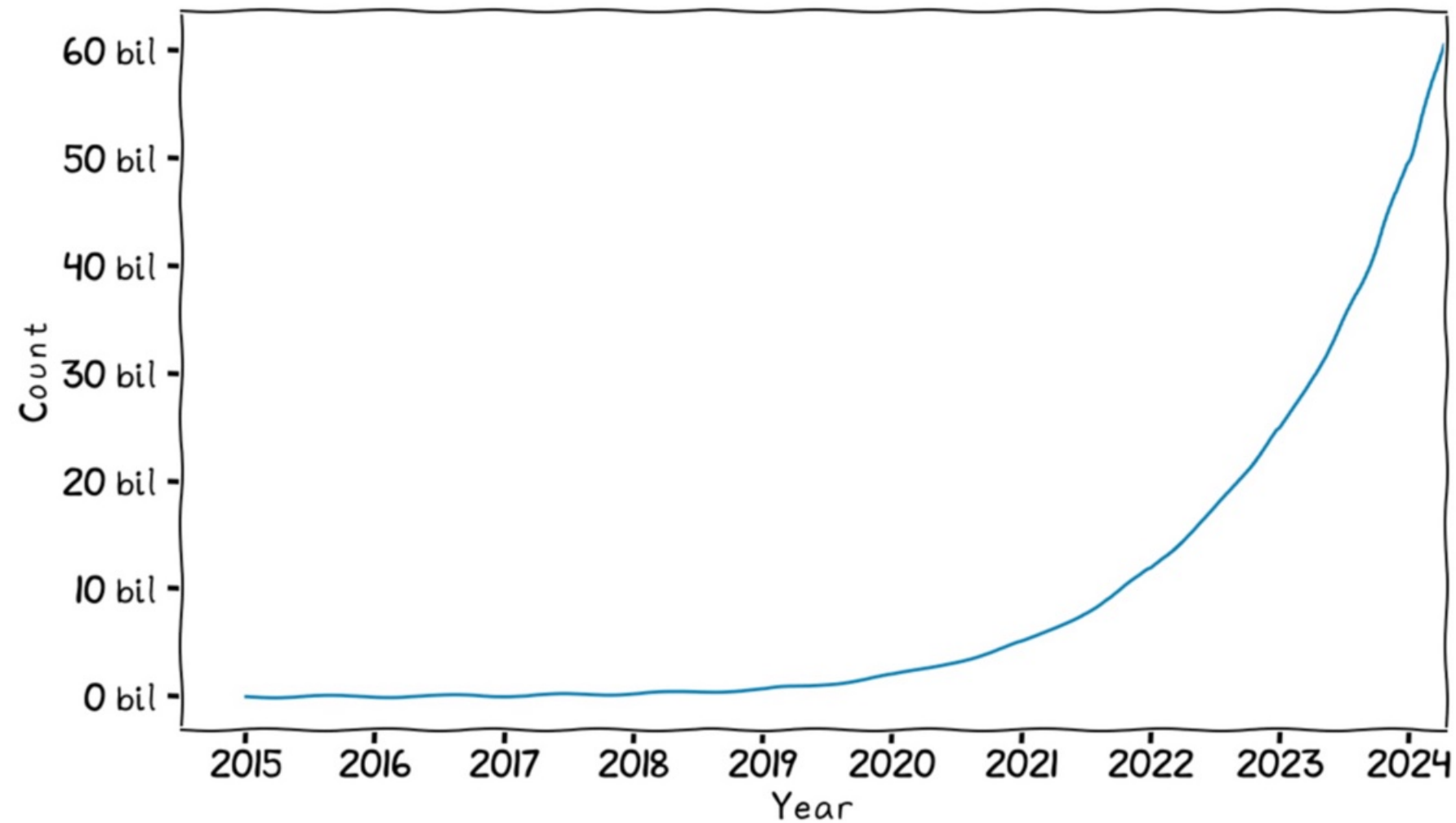
**Steady progress  
across  
the ecoystem**



# The ecosystem keeps growing



Rust package downloads





# The ecosystem keeps growing



**100,204,157,539**

Downloads



**165,831**

Crates in stock





05



**Steampunk Crab Brooch with Vintage Watch Mechanism - Unique Mechanical...**  
This stunning steampunk crab brooch features a vintage watch mechanism delicately embedded in...  
Etsy / 75.99 SGD



The Future



# How Rust ships features



## 1. RFC

Create an RFC, which gets discussed and maybe accepted.



## 2. Unstable Feature

After RFC got accepted, you can use the new feature on nightly. Changes are still expected.



## 3. Stabilization

The feature gets released to beta and later to stable. It goes "on the release train."  
Everyone can use the new feature.



# How Rust ships features



RFC Book



Unstable Book



Vision  
Documents



05



for compiler changes  
for language changes  
for library changes  
e-fields  
ocess  
ute-usage  
ntrinsics  
-attributes  
n-builtin-traits  
ve-priv  
ded-type-parameters  
-facade  
ps  
n-arm-attributes  
t  
ve-tilde  
ne-strbuf  
le-file-system-hierarchy  
r-temporary-lifetimes  
-unsafe-pointers  
iterals  
-block-expr  
fined-struct-layout  
rn-macros  
registers



## Rust RFCs - RFC Book - Active RFC List

The “RFC” (request for comments) process is intended to provide a consistent and controlled path for changes to Rust (such as new features) so that all stakeholders can be confident about the direction of the project.

Many changes, including bug fixes and documentation improvements can be implemented and reviewed via the normal GitHub pull request workflow.

Some changes though are “substantial”, and we ask that these be put through a bit of a design process and produce a consensus among the Rust community and the [sub-teams](#).

### Table of Contents

- [Opening](#)
- [Table of Contents](#)
- [When you need to follow this process](#)
- [Sub-team specific guidelines](#)
- [Before creating an RFC](#)
- [What the process is](#)
- [The RFC life-cycle](#)
- [Reviewing RFCs](#)
- [Implementing an RFC](#)
- [RFC Postponement](#)
- [Help this is all too informal!](#)

RFC Book







# RFC Book

[Go to the book](#)



05



The Rust Unstable Book

## The Unstable Book

Welcome to the Unstable Book! This book consists of a number of chapters, each one organized by "feature flag." That is, when using an unstable feature of Rust, you must use a flag, like this:

```
#![feature(coroutines, coroutine_trait, stmt_expr_attributes)]

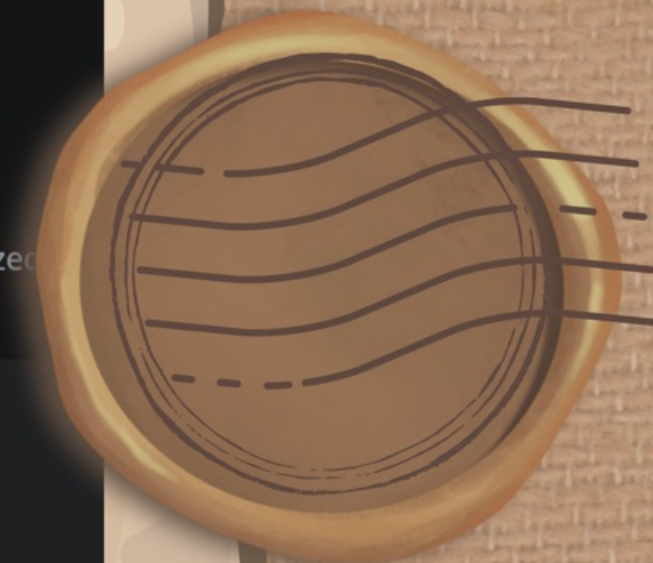
use std::ops::{Coroutine, CoroutineState};
use std::pin::Pin;

fn main() {
    let mut coroutine = #[coroutine] || {
        yield 1;
        return "foo"
    };

    match Pin::new(&mut coroutine).resume(()) {
        CoroutineState::Yielded(1) => {}
        _ => panic!("unexpected value from resume"),
    }
    match Pin::new(&mut coroutine).resume(()) {
        CoroutineState::Complete("foo") => {}
        _ => panic!("unexpected value from resume"),
    }
}
```

The `coroutines` feature [has a chapter](#) describing how to use it.

This documentation relates to unstable features. It is not guaranteed that what is here is accurate or up-to-date. It is provided on an "as-is" basis. Each page will have a "Warning" icon in the top right corner of those as well.



Unstable Book



05

# Bonus!

A personal list  
of exciting features  
I look forward to







# async iterations are painful...

```
async fn sum_with_next(mut stream: Pin<&mut dyn Stream<Item = i32>>) -> i32 {  
    use futures::stream::StreamExt; // for `next`  
    let mut sum = 0;  
    while let Some(item) = stream.next().await {  
        sum += item;  
    }  
    sum  
}
```

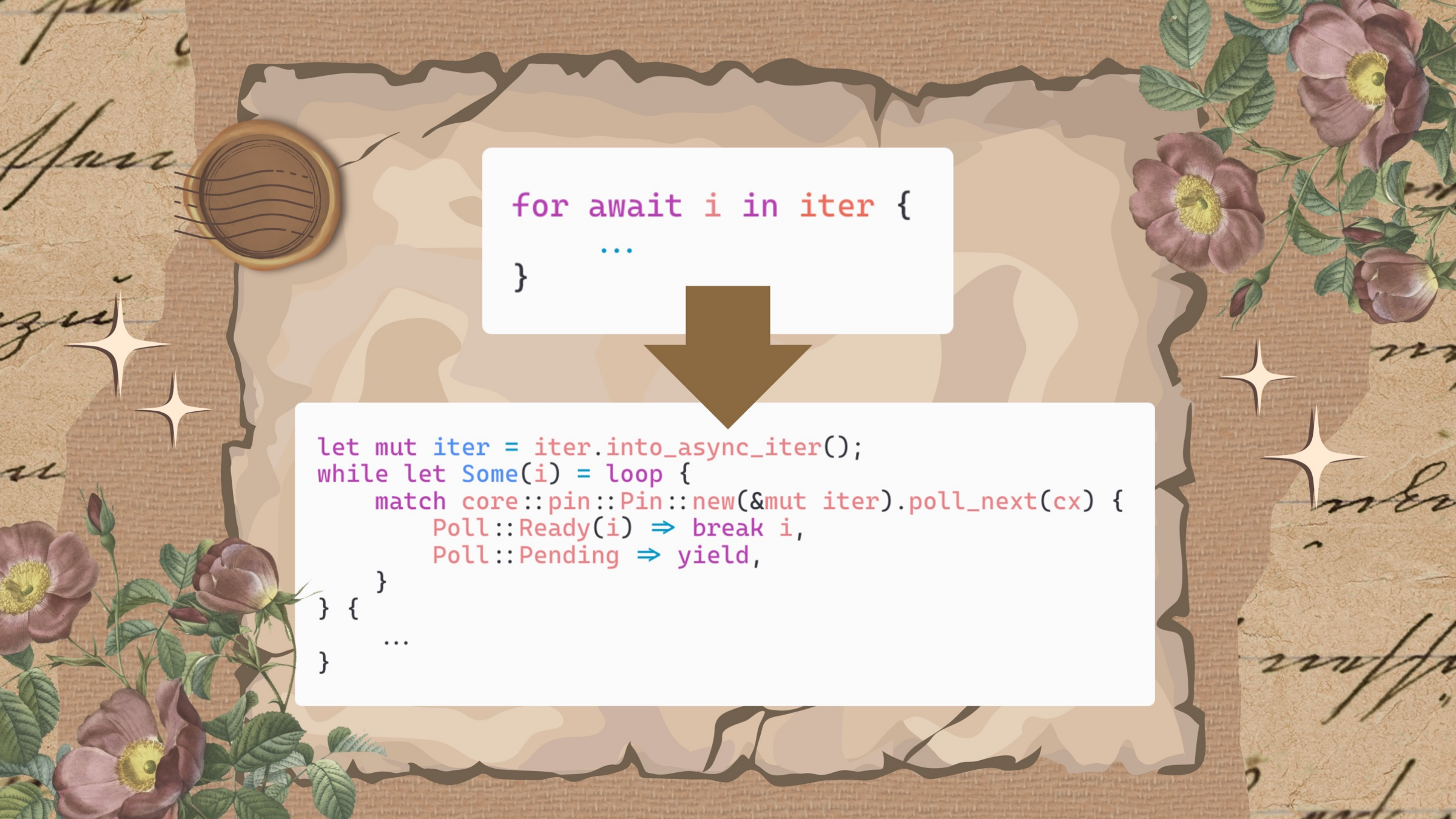




# Enter `async_for_loop`

```
for await i in iter {  
    ...  
}
```





```
for await i in iter {  
    ...  
}
```



```
let mut iter = iter.into_async_iter();  
while let Some(i) = loop {  
    match core::pin::Pin::new(&mut iter).poll_next(cx) {  
        Poll::Ready(i) => break i,  
        Poll::Pending => yield,  
    }  
} {  
    ...  
}
```



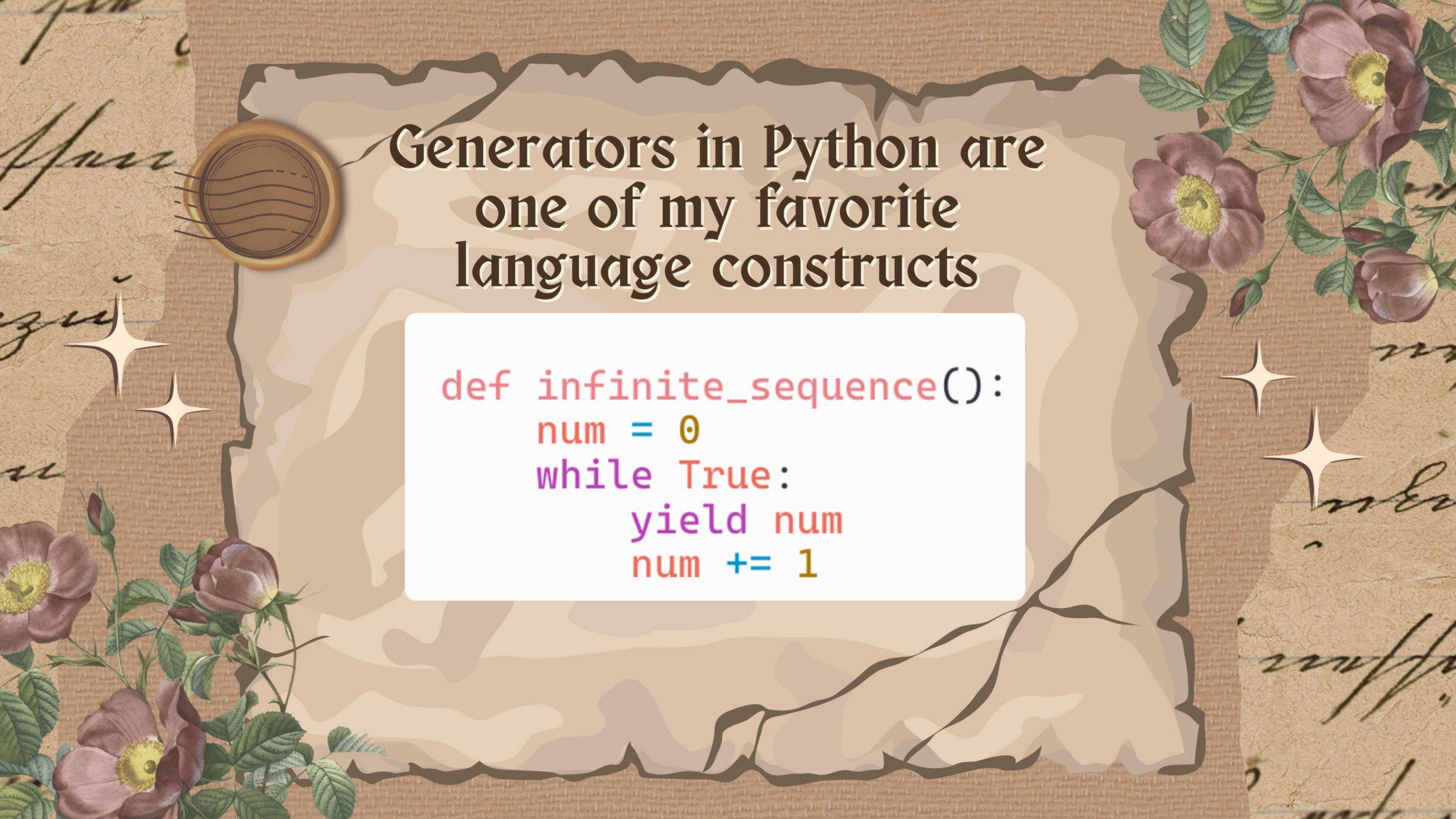


# Variadic functions with c\_variadic

```
#![feature(c_variadic)]

pub unsafe extern "C" fn add(n: usize, mut args: ...) -> usize {
    let mut sum = 0;
    for _ in 0..n {
        sum += args.arg::<usize>();
    }
    sum
}
```





# Generators in Python are one of my favorite language constructs

```
def infinite_sequence():  
    num = 0  
    while True:  
        yield num  
        num += 1
```



# Enter coroutines

```
#![feature(coroutines, coroutine_trait, stmt_expr_attributes)]

use std::ops::{Coroutine, CoroutineState};
use std::pin::Pin;

fn main() {
    let mut coroutine = #[coroutine] || {
        yield 1;
        return "foo"
    };

    match Pin::new(&mut coroutine).resume(()) {
        CoroutineState::Yielded(1) => {}
        _ => panic!("unexpected value from resume"),
    }
    match Pin::new(&mut coroutine).resume(()) {
        CoroutineState::Complete("foo") => {}
        _ => panic!("unexpected value from resume"),
    }
}
```



# Multiple chained if-lets are awkward...

```
if let Some(i) = func1() {  
    if let Some(j) = func2(i) {  
        if let Some(k) = func3(j) {  
            if let Some(result) = func4(k) {  
                // Do something with result  
            } else {  
                println!("func 4 returned None");  
            }  
        } else {  
            println!("func 3 returned None");  
        }  
    } else {  
        println!("func 2 returned None");  
    }  
} else {  
    println!("func 1 returned None");  
}
```



# Enter if-let-chains

```
fn param_env<'a, 'tcx>(tcx: TyCtxt<'a, 'tcx, 'tcx>, def_id: DefId) → ParamEnv<'tcx> {  
    if let Some(Def::Existential(_)) = tcx.describe_def(def_id)  
        && let Some(node_id) = tcx.hir.as_local_node_id(def_id)  
        && let hir::map::NodeItem(item) = tcx.hir.get(node_id)  
        && let hir::ItemExistential(ref exist_ty) = item.node  
        && let Some(parent) = exist_ty.impl_trait_fn  
    {  
        return param_env(tcx, parent);  
    }  
    ...  
}
```





# Much more in the Unstable Book!

Really, go to the book





05



tion  
goal process  
v  
t accepted  
notes  
goal process  
v  
d goals  
g the Async Rust experience  
o parity with sync Rust  
anize Rust All-Hands 2025  
obilize tooling needed by Rust for  
bilizable" prototype for  
ed const generics  
tinue resolving `cargo-semver-  
blockers for merging into cargo  
larative (`macro\_rules!`) macro  
ements  
uate approaches for seamless  
between C++ and Rust  
eriment with ergonomic ref-  
3  
ose experimental LLVM  
; for GPU offloading  
end pubgrub to match cargo's  
ency resolution



# Rust Vision Document

Metadata	
Point of contact	<a href="#">Niko Matsakis</a>
Teams	<a href="#">leadership-council</a>
Task owners	<a href="#">Niko Matsakis</a> , vision team
Status	Proposed

## Summary

Present a first draft of a "Rust Vision Doc" at the Rust All Hands in May.

The Rust Vision Doc will summarize the state of Rust adoption -- where is Rust adding value? what works well? what doesn't? -- based on conversations with individual Rust users from different communities, major Rust projects, and companies large and small that are adopting Rust. It will use that raw data to make recommendations on what problems Rust should be attacking and what constraints we should be trying to meet. The document will not include specific features or recommendations, which ought to be legislated through RFCs.

## Motivation

Goal is to author a longer term "vision doc" that identifies opportunities for Rust over the next few years. The document will identify problems that move the



Rust Vision



05

# Planned Flagship Goals



## Async

Remove papercuts

## Rust in Linux

Tools for making Rust in the Linux Kernel work better

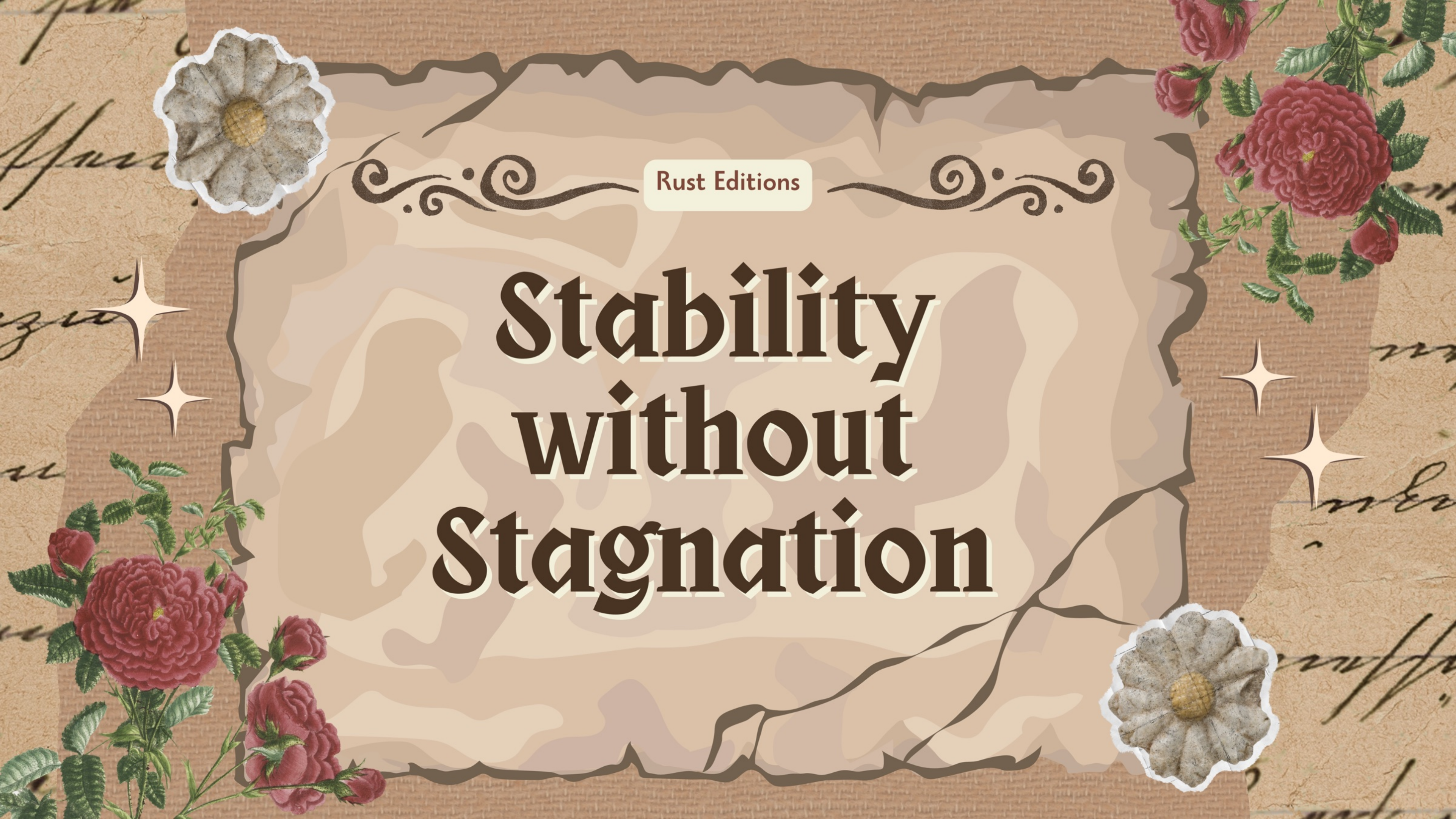
## All Hands

Rust All Hands at RustWeek NL 2025



Rust Editions

# Stability without Stagnation





05



corrode



[Home](#)

[Services](#)

[Podcast](#)

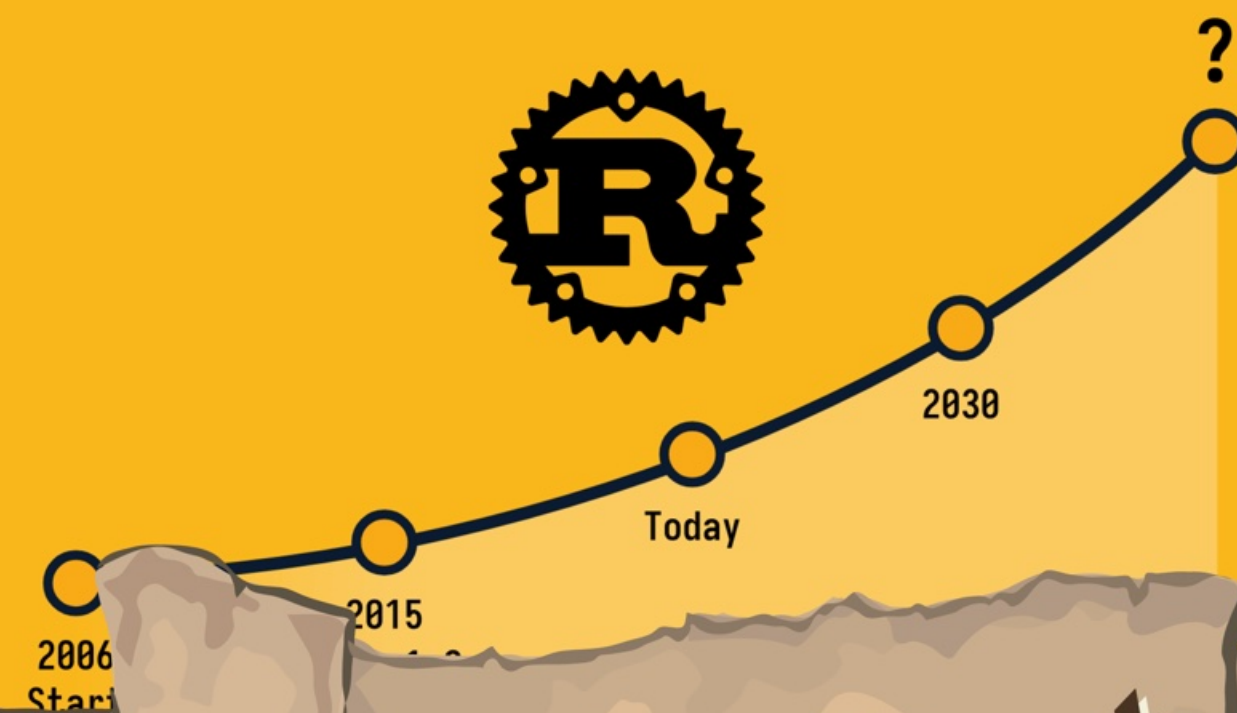
[Blog](#)

[About](#)

RUST INSIGHTS

# WILL RUST BE ALIVE IN 10 YEARS?

📅 Last updated 2024-07-10



Blog\_Post






Subscribe on



SEASON 03

 Brave with Anton Lazarev	2025-01-09 S03 E07	>
 Holiday Episode	2024-12-26 S03 E06	>
 Zoo with Jessie Frazelle	2024-12-12 S03 E05	>
 GitButler with Scott Chacon and Kiril Videlov	2024-11-28 S03 E04	>
 Oxide with Steve Klabnik	2024-11-14 S03 E03	>
 InfinyOn with Deb Roy Chowdhury	2024-10-31 S03 E02	>
 Zoo with win		

Podcast



[www.corrode.dev](http://www.corrode.dev)

**I brought  
stickers!**



[www.corrode.dev](http://www.corrode.dev)

**Thank  
you**

