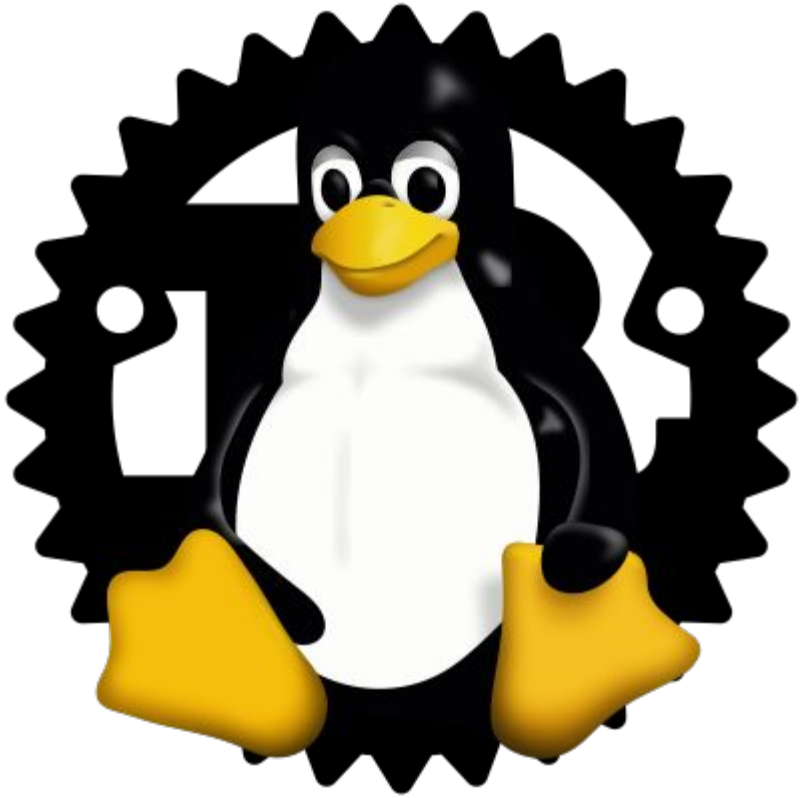


Rust for Linux

Miguel Ojeda

ojeda@kernel.org

What is Rust for Linux?

“Rust for Linux is the project adding support for the Rust language to the Linux kernel.”

What is Rust for Linux?

Our goal has always been:

- Full integration of Rust into the kernel as the second main programming language.
- First-class support for the language.
- Focused on in-tree, not out-of-tree.
- Not limited to loadable modules.
- Shared infrastructure, e.g. standard library.
- Not limited to drivers or “leaf modules”.
- Not limited to kernelspace code.
- Always with the aim to upstream it.

Who uses Rust for Linux?

Upstreamed users:

- PHY drivers: Asix PHYs (first “Rust reference driver”) and AMCC QT2025 PHY.
- Null Block driver.
- DRM panic screen QR code generator.

Users targeting upstream:

- Android Binder driver.
- Apple AGX GPU driver.
- NVMe driver.
- Nova GPU driver.
- ...and other efforts (e.g. tarfs, erofs, PuzzleFS, codec libraries, regulator driver, DSI panel driver...).

— https://rust-for-linux.com‘s “Users” section
— https://rust-for-linux.com/rust-reference-drivers

Why?

And why Rust?

Is this function correct?

```
/// Returns whether the integer pointed by `a`
/// is equal to the integer pointed by `b`.
bool f(int *a, int *b) {
   return *a == 42;
}
```

Is this function correct?

```
/// Returns whether the integer pointed by `a`
/// is equal to the integer pointed by `b`.
bool f(int *a, int *b) {
    return *a == 42;
}
```

```
/// Returns whether the integer pointed by `a`
/// is equal to the integer pointed by `b`.
bool f(int *a, int *b) {
    return *a == *b;
}
```

Incorrect

```
bool f(const int *a, const int *b) {
   return *a == 42;
}
```

Correct

```
bool f(const int *a, const int *b) {
   return *a == *b;
}
```

Incorrect

```
bool f(const int *a, const int *b) {
   return *a == 42;
}
```

```
fn f(a: &i32, b: &i32) -> bool {
   *a == 42
}
```

Correct

```
bool f(const int *a, const int *b) {
   return *a == *b;
}
```

```
fn f(a: &i32, b: &i32) -> bool {
   *a == *b
}
```

	Unsafe	Safe

Incorrect	`bool f(const int *a, const int *b) {`
	`  return *a == 42;`
	`}`
		`fn f(a: &i32, b: &i32) -> bool {`
		`  *a == 42`
		`}`

Correct	`bool f(const int *a, const int *b) {`
	`  return *a == *b;`
	`}`
		`fn f(a: &i32, b: &i32) -> bool {`
		`  *a == *b`
		`}`

	Unsafe	Safe
Incorrect	`bool f(const int *a, const int *b) { return *a == 42; }` `unsafe fn f(a: *const i32, b: *const i32) -> bool { unsafe { *a == 42 } }`	`fn f(a: &i32, b: &i32) -> bool { *a == 42 }`
Correct	`bool f(const int *a, const int *b) { return *a == *b; }` `unsafe fn f(a: *const i32, b: *const i32) -> bool { unsafe { *a == *b } }`	`fn f(a: &i32, b: &i32) -> bool { *a == *b }`

	Unsafe	Safe

Incorrect	`bool f(const int *a, const int *b) {`
	`  return *a == 42;`
	`}`
	`unsafe fn f(a: *const i32, b: *const i32) -> bool {`
	`  unsafe { *a == 42 }`
	`}`
		
		`fn f(a: &i32, b: &i32) -> bool {`
		`  *a == 42`
		`}`

Correct	`bool f(const int *a, const int *b) {`
	`  return *a == *b;`
	`}`
	`unsafe fn f(a: *const i32, b: *const i32) -> bool {`
	`  unsafe { *a == *b }`
	`}`
		
		`fn f(a: &i32, b: &i32) -> bool {`
		`  *a == *b`
		`}`

“**Goals**

By using Rust in the Linux kernel, our hope is that:

– New code written in Rust has a reduced risk of memory safety bugs, data races and **logic bugs** overall, thanks to the language properties mentioned below.

– **Maintainers are more confident** in refactoring and accepting patches for modules thanks to the safe subset of Rust.

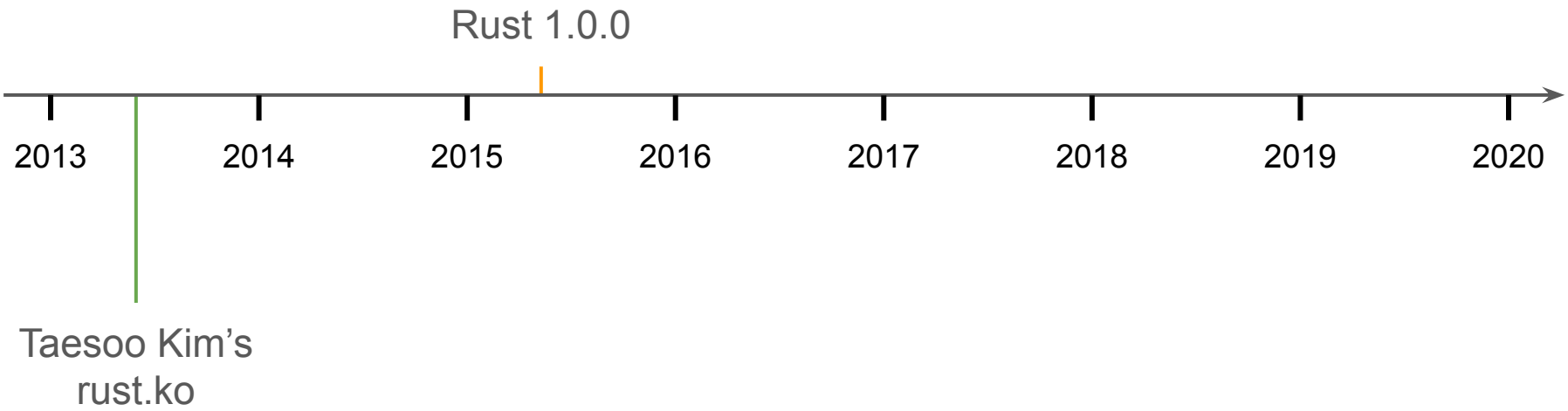
– New drivers and **modules become easier to write**, thanks to abstractions that are easier to reason about, based on modern language features, as well as backed by detailed documentation.

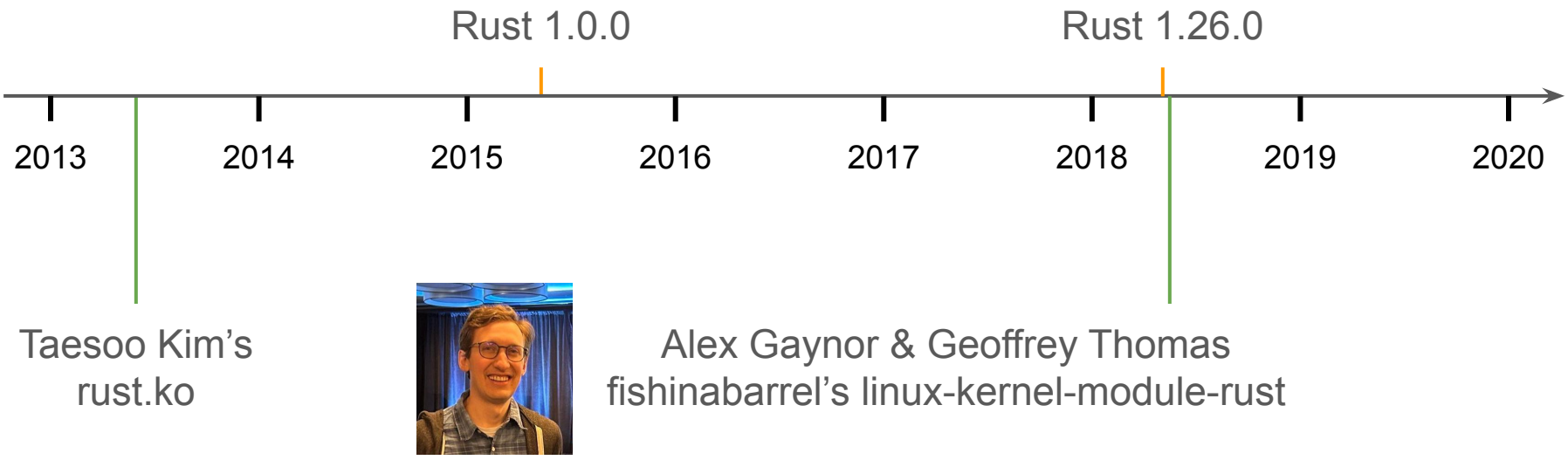
– **More people get involved** overall in developing the kernel thanks to the usage of a modern language.

– By taking advantage of Rust tooling, we keep enforcing the **documentation guidelines** we have established so far in the project. For instance, we require having all public APIs, safety preconditions, `unsafe` blocks and type invariants documented.”

Original Rust for Linux RFC
https://lore.kernel.org/lkml/20210414184604.23473-1-ojeda@kernel.org/


2013
2014
2015
2016
2017
2018
2019
2020


Rust 1.0.0
2013
2014
2015
2016
2017
2018
2019
2020
Taesoo Kim’s
rust.ko


Rust 1.0.0
Rust 1.26.0
2013
2014
2015
2016
2017
2018
2019
2020
Taesoo Kim’s
rust.ko
Alex Gaynor & Geoffrey Thomas
fishinabarrel’s linux-kernel-module-rust

Rust 1.0.0

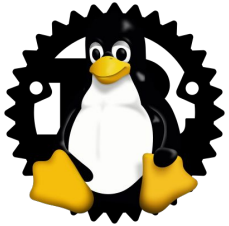
Rust 1.26.0

2013 2014 2015 2016 2017 2018 2019 2020

Taesoo Kim's
rust.ko

Alex Gaynor & Geoffrey Thomas
fishinabarrel's linux-kernel-module-rust

Out-of-tree
approaches



Rust for Linux
GitHub Organization

Rust 1.0.0

Rust 1.26.0

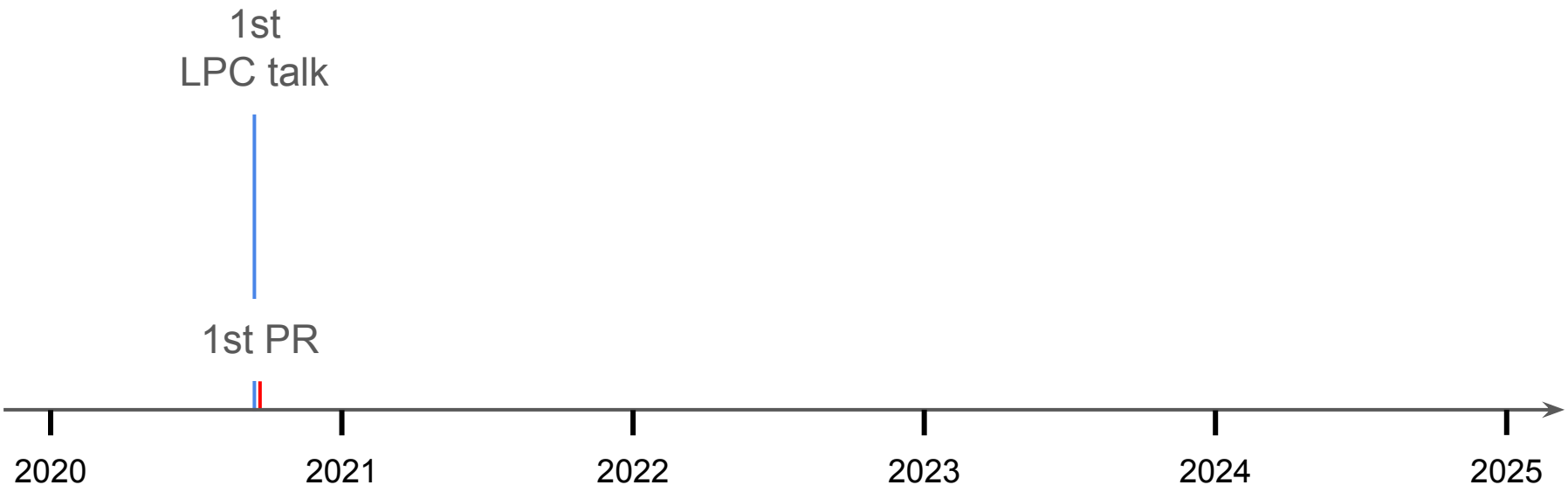
2013 2014 2015 2016 2017 2018 2019 2020

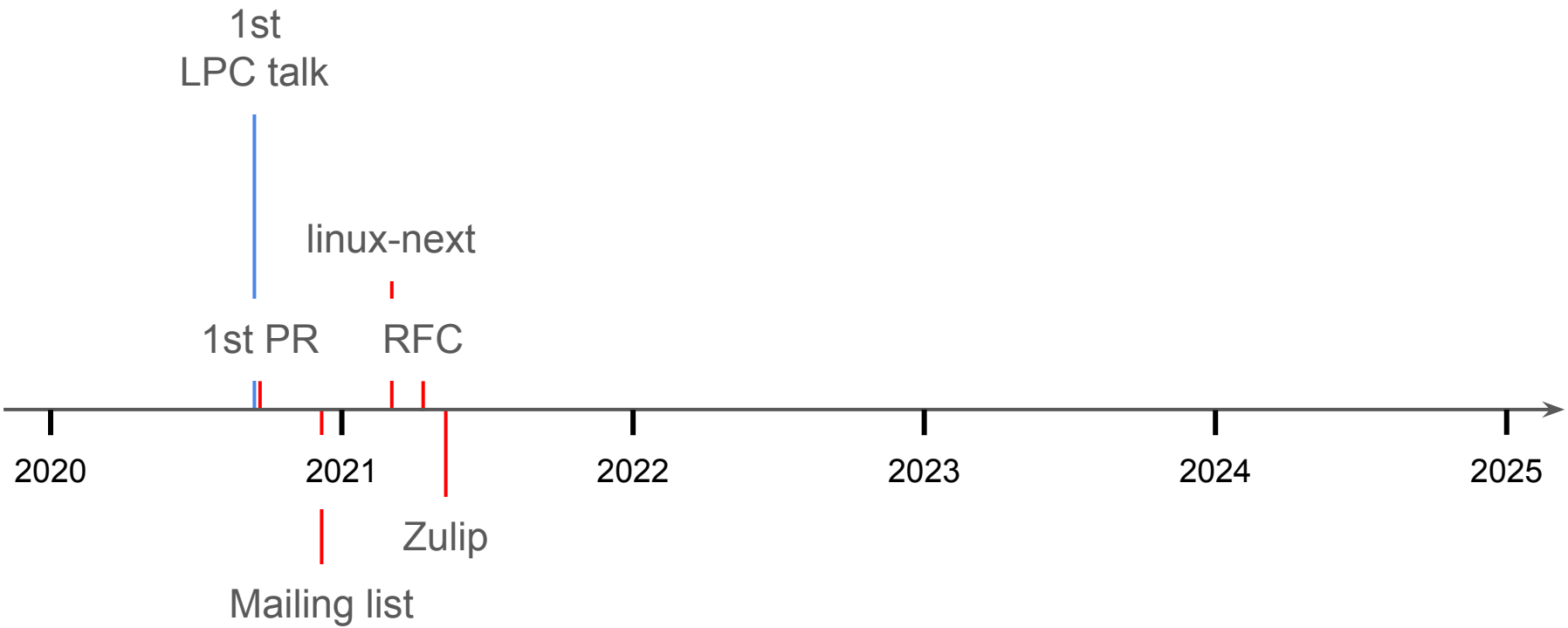
Taesoo Kim’s
rust.ko

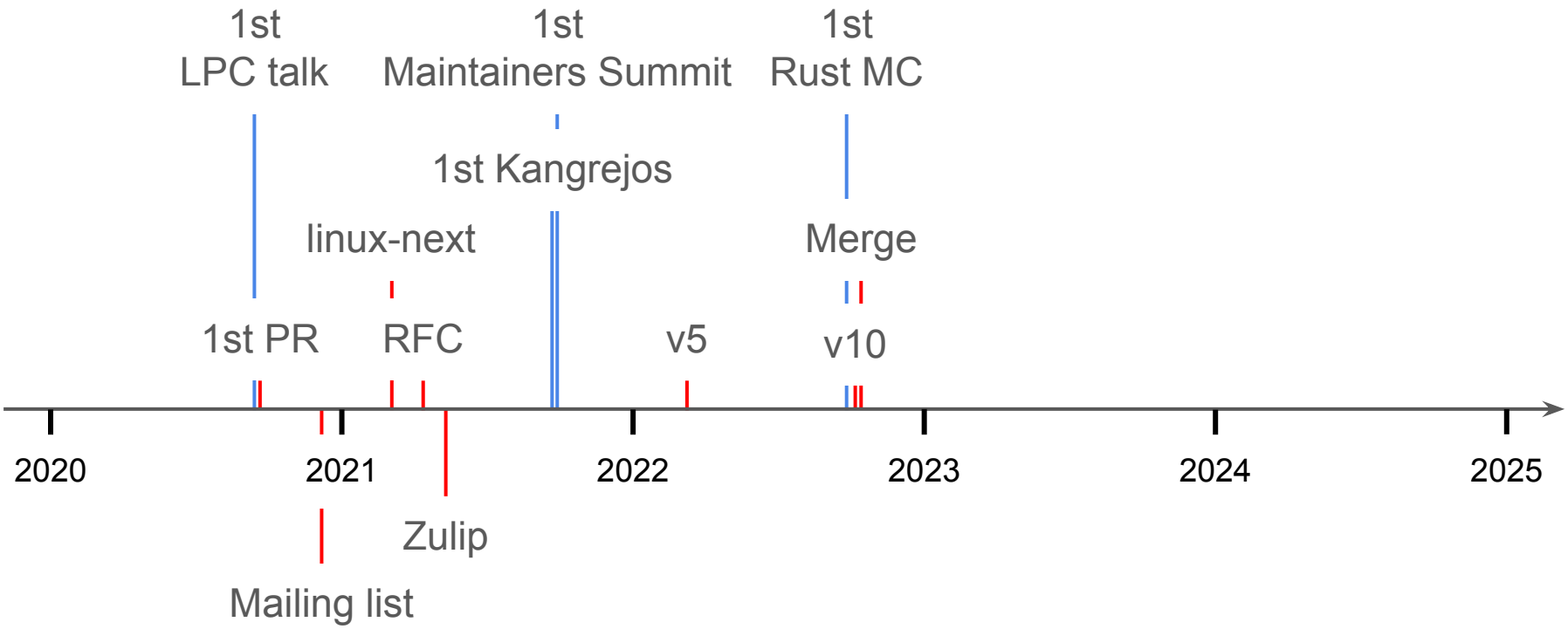
Alex Gaynor & Geoffrey Thomas
fishinabarrel’s linux-kernel-module-rust

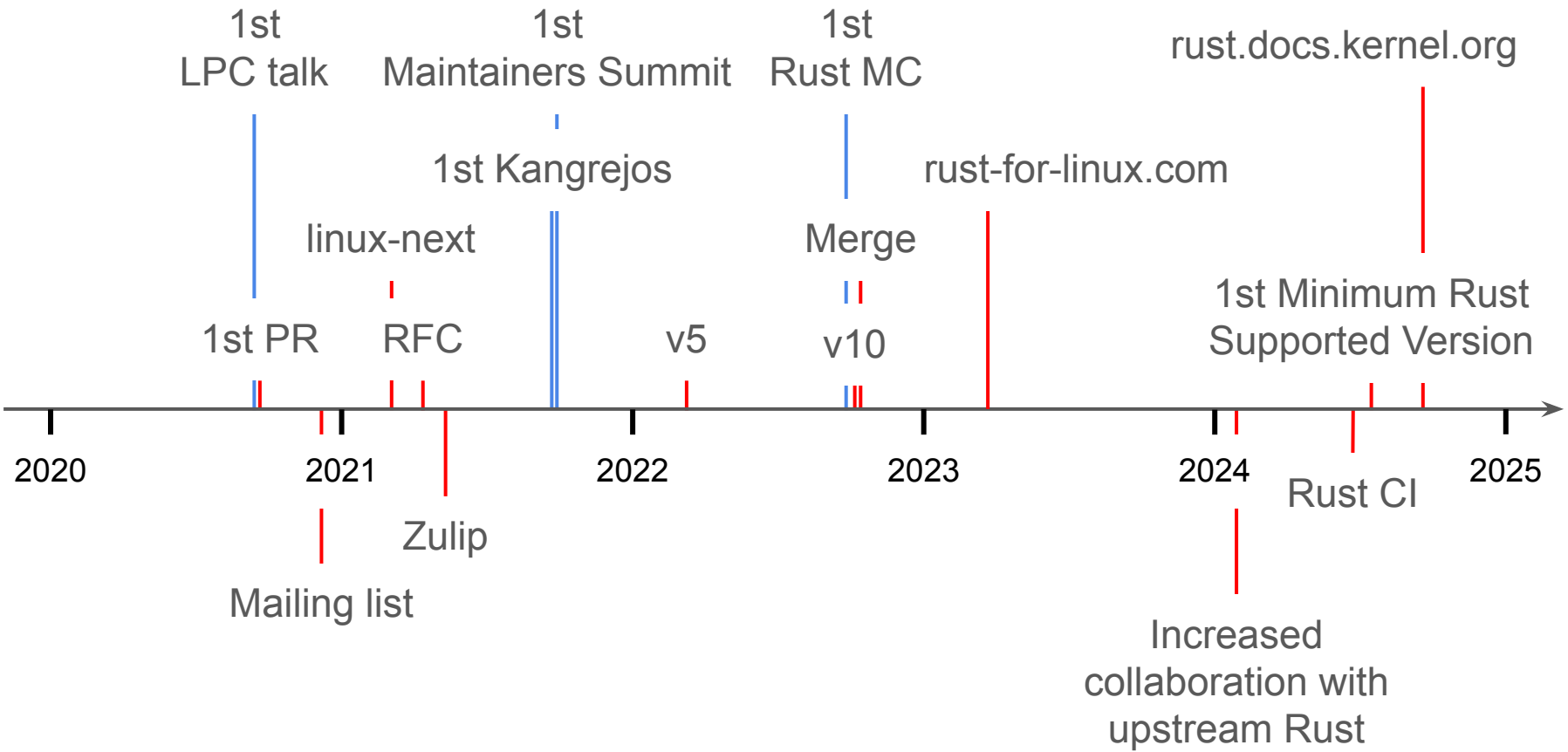
Out-of-tree
approaches


2020
2021
2022
2023
2024
2025


1st
LPC talk
1st PR
2020
2021
2022
2023
2024
2025


1st
LPC talk
linux-next
1st PR
RFC
2020
2021
2022
2023
2024
2025
Zulip
Mailing list


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
1st Kangrejos
linux-next
Merge
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
Zulip
Mailing list


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st Minimum Rust
Supported Version
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
Rust CI
Zulip
Mailing list
Increased
collaboration with
upstream Rust

Collaboration with Rust

Since February, **regular meetings** between Rust and Rust for Linux.

Thanks a lot to Josh, Niko and Sid for helping to set them up.

Rust for Linux was a **flagship Rust Project goal** for 2024H2.

Closing the largest gaps that block building Linux on stable Rust.

Including language, library, compiler, CI...

See also Niko's and my RustConf 2024 keynote.

— https://rustconf.com/schedule/
— https://blog.rust-lang.org/2024/08/12/Project-goals.html
— https://rust-lang.github.io/rust-project-goals/2024h2/rfl_stable.html

Linux in Rust's and `bindgen`'s CI

One result that happened very quickly was including Rust for Linux in the per-merge Rust CI.

That is, **every Rust PR now build-tests the Linux kernel**.

Both projects hope to avoid unintentional changes to Rust that break the kernel.

Thus, in general, apart from intentional changes, the upcoming Rust compiler versions should generally work.

`bindgen` will also include Linux in its CI.

— https://rust-for-linux.com/rust-version-policy
— https://rustc-dev-guide.rust-lang.org/tests/rust-for-linux.html

Declaring a minimum Rust version (unpinning)

Having Linux in Rust's and bindgen's CI helped us unpin the Rust version.

In Linux v6.11, a minimum Rust version was declared.

> Our "Minimum Supported Rust Version" is currently 1.78.0.

How often will we upgrade it?

> When there is a good reason for that.
>
> For instance, Debian Trixie has been requested to provide Rust 1.85 for Edition 2024. If it happens, we may migrate to it.

— https://rust-for-linux.com/rust-version-policy
— https://rustc-dev-guide.rust-lang.org/tests/rust-for-linux.html
— https://alioth-lists.debian.net/pipermail/pkg-rust-maintainers/2024-July/044870.html

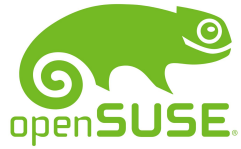
Distributions support

Declaring a minimum Rust version allowed us to start supporting distributions.

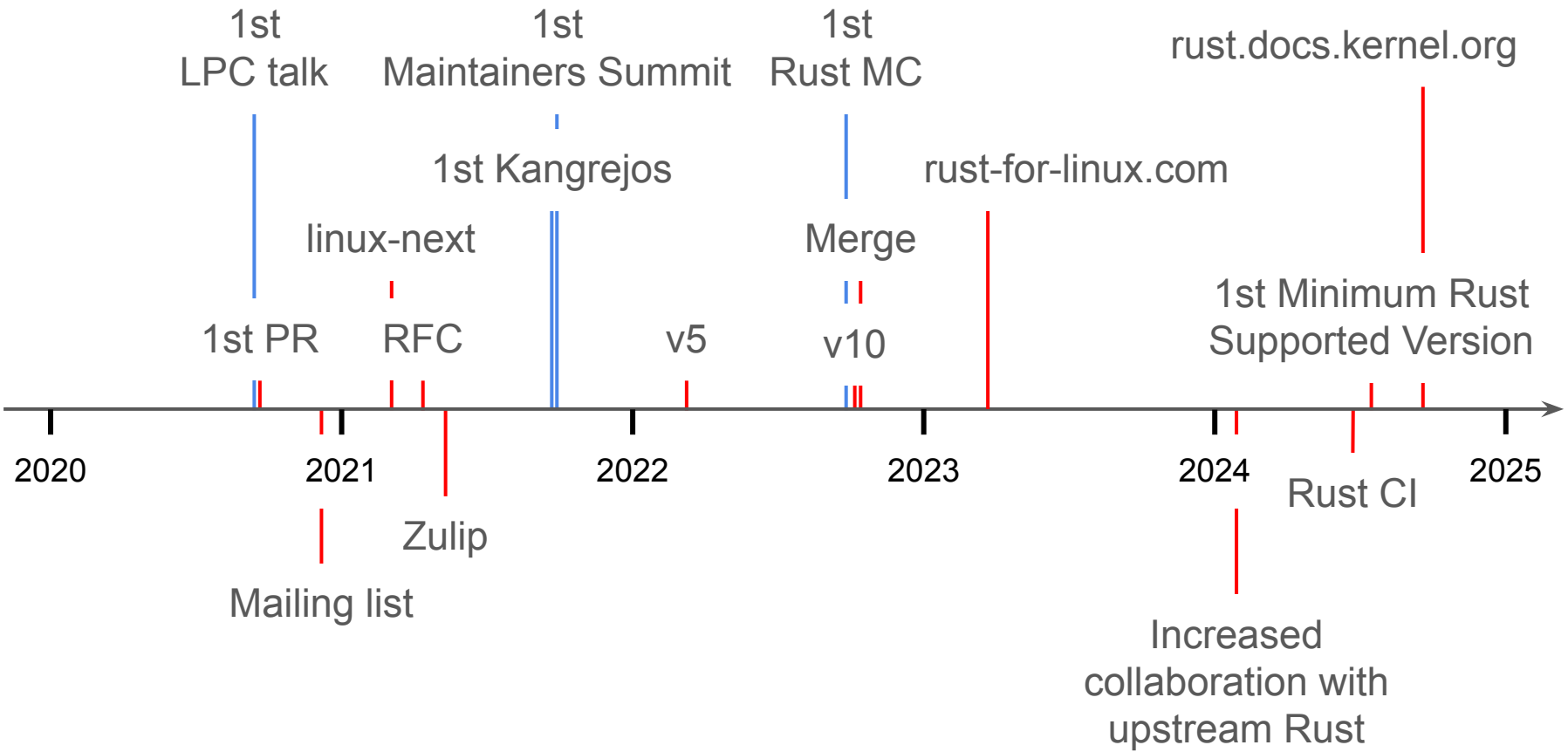
This was a top requirement.

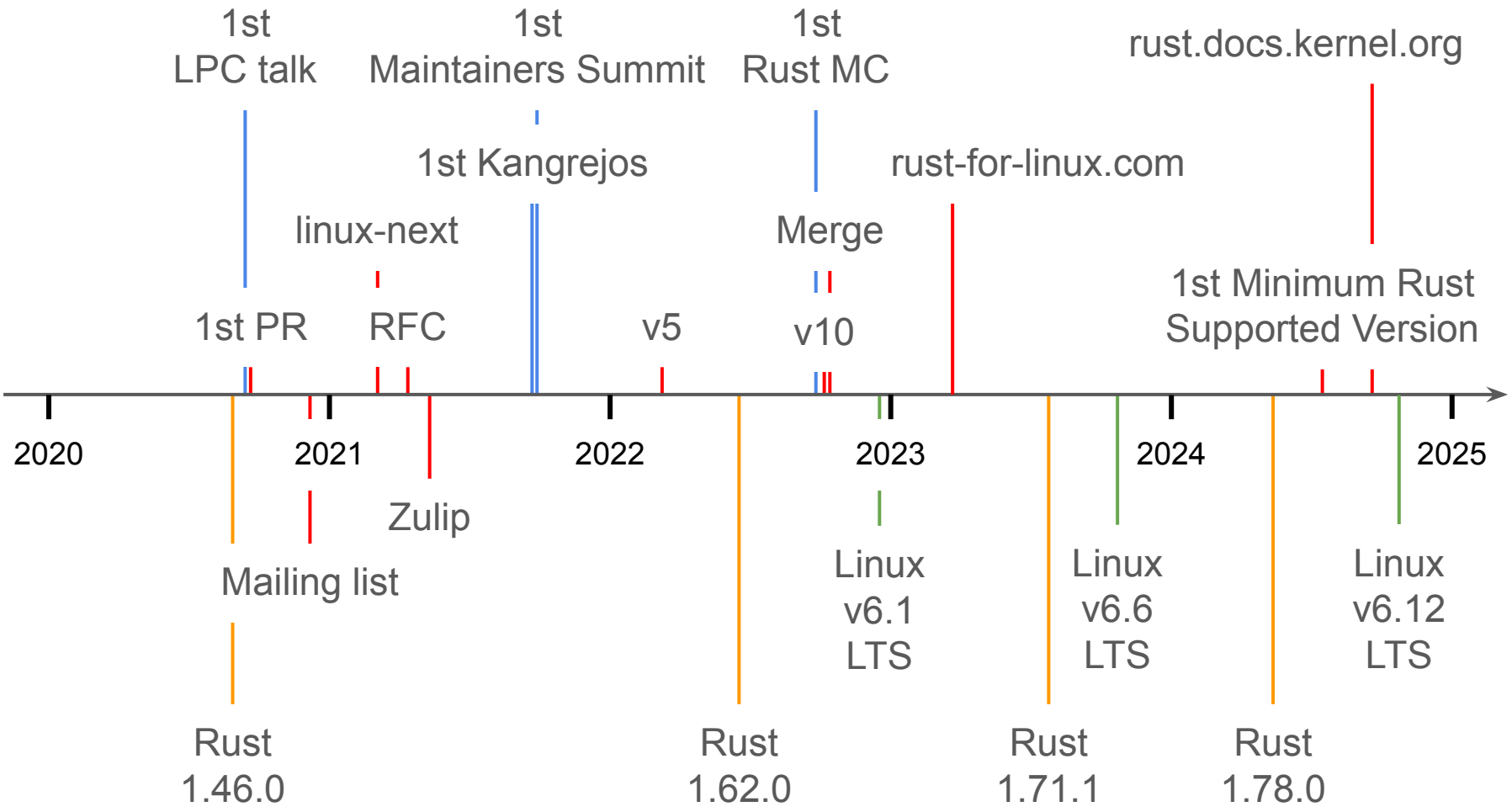
Distributions that should generally work out of the box:

- Arch Linux.
- Debian Testing and Unstable (outside the freeze period).
- Fedora Linux.
- Gentoo Linux.
- Nix (unstable channel).
- openSUSE Slowroll and Tumbleweed.
- Ubuntu 24.04 LTS and 24.10.

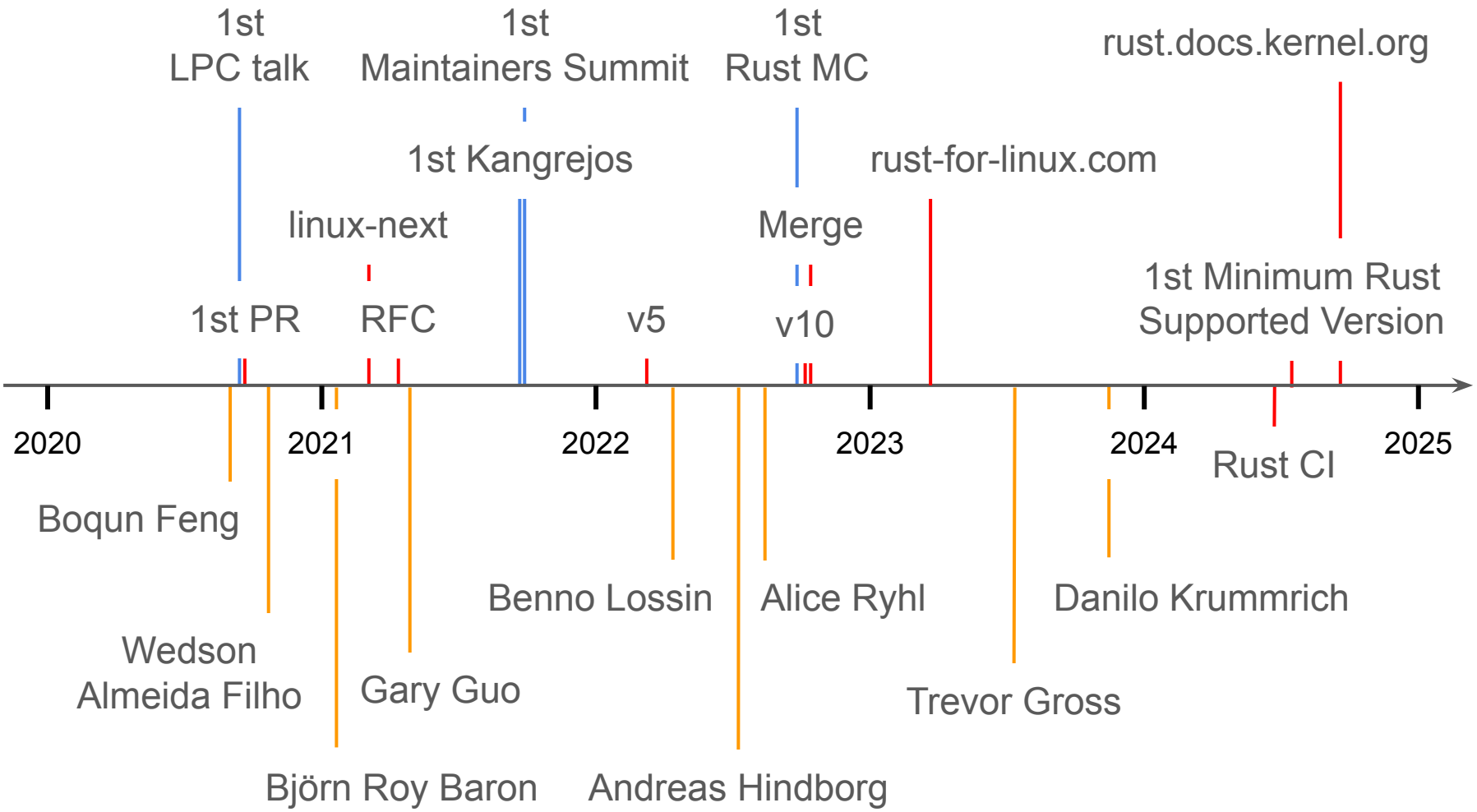


— https://docs.kernel.org/rust/quick-start.html#distributions
— https://rust-for-linux.com/rust-version-policy#supported-toolchains


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st Minimum Rust
Supported Version
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
Rust CI
Zulip
Mailing list
Increased
collaboration with
upstream Rust


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st PR
RFC
v5
v10
1st Minimum Rust
Supported Version
2020
2021
2022
2023
2024
2025
Zulip
Mailing list
Linux
v6.1
LTS
Linux
v6.6
LTS
Linux
v6.12
LTS
Rust
1.46.0
Rust
1.62.0
Rust
1.71.1
Rust
1.78.0

Who is the team?


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st Minimum Rust
Supported Version
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
Rust CI
Boqun Feng
Benno Lossin
Alice Ryhl
Danilo Krummrich
Wedson
Almeida Filho
Gary Guo
Trevor Gross
Björn Roy Baron
Andreas Hindborg

Wedson Almeida Filho is the original author of many of the initial abstractions, drivers and prototypes that others then went to work into, including:

- Android Binder driver.
- 9p server and Async Rust.
- NVMe driver.
- Read-only FS abstractions and tarfs.
- GPIO driver.

Without him, the project would have taken much longer to get bootstrapped.

It was a pleasure to work with him.

When you got interested/started in the project?

“I cannot remember the exact month, but I think it should be year 2020, I vaguely remember I was interested and knew the project even before the talk about Rust in LPC, so it should be before Aug, 2020 ;-)”

What got you interested/started in the project, i.e. why you joined?

“As some one who worked in Linux kernel and system programming for a few years, I was always interested to see how type system (and even formal proof) can help in the area, and Rust is just the perfect match for this purpose, so when people (mostly Miguel ;-)) figured out how to build the Linux kernel with Rust enabled, I felt it's the time for me to chime in ;-)”

What are you working on? Anything else that you may want to share?

“I'm currently working on:

- develop and review Rust code in my focus areas: core kernel, atomic/locking/RCU.
- support Rust-for-Linux community, such as a) follow up email/zulip discussion with GitHub issues, b) organize and host online meetings, c) organize zulip discussion.
- help more talents from Microsoft, my employer, find their way in Rust-for-Linux project.”

Boqun Feng
Microsoft

When you got interested/started in the project?

“That would have been all the way back in https://github.com/Rust-for-Linux/linux/pull/52#pullrequestreview-567163056, so January 13th 2021.”

What got you interested/started in the project, i.e. why you joined?

“As someone who has been working on the rust compiler I noticed that RfL has many dependencies on unstable features and some one rustc implementation details. I've been trying to reduce these dependencies and to steer away from using unstable features that are less likely to get stabilized. And doing unsafe code reviews has also been fun.”

Anything else that you may want to share?

“I'm not really active anymore as you may have noticed. Now that I've got a full time job, I don't have as much spare time anymore as I used to. I'm still available for answering (rustc) questions.”

Björn Roy Baron

When you got interested/started in the project?

“I think I started getting involved after the first RFC hits the list?”

What got you interested/started in the project, i.e. why you joined?

“I am interested because of prior experience of both the Rust and the Linux kernel.”

Anything else that you may want to share?

“I don't have any interesting status update to share, as what I was working on recently have very little to do with RfL. My involvement nowadays is quite limited, with just occasional code reviews and weekly meetings and I just don't have enough time...”

Gary Guo

“I got interested in the project at the end of 2021. One of my friends at university told me that Rust would be added to the Linux kernel. In that year, I just learned Rust and was very excited, talking a lot about it to my friends.

After my friend told me about it, I decided to take a look at the code in the github repository. I didn't know anything at all about the kernel back then (aside from "it's the kernel, it does hardware stuff") and I just opened random files until I found the `rust/kernel` folder. There I looked around for something familiar and I managed to find the synchronization primitives. It was with some shock that I discovered that the `new` functions were `unsafe`!

This discovery motivated me to try to find a solution, first I looked around for some issues and PR fixes, but I did not find anything that solved it in a nice way. The next few months I hacked away at the problem and eventually opened my first issue explaining my issue and my initial solution.

Then one thing lead to another and now my solution for this problem is upstream and ensuring that we don't have to write (as much) `unsafe` initialization code!

Nowadays, I am working on field projections, a general solution to many problems that have been coming up again and again in Rust for Linux. Additionally, I am maintaining the `pinned-init` library and I have a patch series on the back burner about untrusted data. Lastly I had an idea about abstracting the intrusive-data-pattern into a library similar to `pinned-init`, but I haven't had the time to do so.”

Benno Lossin

“I think I started looking at the project around 22Q1.

I started working for a systems group at Western Digital. In a previous position I had learned Rust and used it for user space and embedded micro controller firmware. At WD I kept pushing for Rust as implementation language for new projects - so I quickly became "the Rust guy". One of my colleagues was organizing Lund Linux Conference, and he had heard about the Rust for Linux project. He thought it would be great to have a presentation about Rust for Linux at LLC. Since I was the Rust guy, he asked me if I could give such a presentation.

I thought that could be fun - but since I did not have any association with the Rust for Linux project, I decided to look into it to do some research for my presentation. I started interacting with the project via the GitHub issues list. This rather quickly became a full time thing.

I am currently working towards providing a fully featured API for writing block device drivers in Rust. To that end I am building a Rust version of the C null block driver. The C null block driver has quite a few dependencies on other kernel APIs, such as module parameters, timers, xarray, configfs. So to build this Rust version of configfs, I have to make all these APIs consumable from Rust. I am currently working on making configfs available.”

Andreas Hindborg
Samsung

“I think I started working on the project around August 2022. Maybe the month before, but that's the earliest evidence I can find. I think the history for how I ended up working on Rust for Linux goes like this:

– I was working on async Rust with Tokio.

– My first task at Google was to add async Rust support to Android's userspace Binder library.

– When I finished that, we wanted to improve the kernel driver to make the userspace async support better.

– We found that to be really difficult in the C driver, and that's when I decided to work on the Rust driver, since I thought it would be easier to adjust for better async support.”

Alice Ryhl
Google

“**Trevor Gross** has been involved with the Rust for Linux project for more than a year now. He has been active reviewing Rust code in the mailing list, and he already is a formal reviewer of the Rust PHY library and the two PHY drivers.

In addition, he is also part of several upstream Rust teams: compiler-contributors team (contributors to the Rust compiler on a regular basis), libs-contributors (contributors to the Rust standard library on a regular basis), crate-maintainers (maintainers of official Rust crates), the binary size working group and the Rust for Linux ping group.

His expertise with the language will be very useful to have around in the future if Rust keeps growing within the kernel, thus add him to the `RUST` entry as a reviewer.”

What got you interested/started in the project, i.e. why you joined?

“As mentioned in my talk at Kangrejos and LPC, Nouveau has a lot of problems (GSP firmware abstractions, very complicated and undocumented lifetime and locking architecture, missing documentation, etc.).

A fair number of those issues especially improve with Rust (GSP firmware abstraction with proc macros, lifetime management, memory safety).

Obviously, those problems are not specific to nouveau. Hence, the motivation is not only limited to solve Nouveau's issues with Nova in Rust, but also motivated by improving the whole kernel making use of Rust's capabilities.”

What are you working on? Anything else that you may want to share?

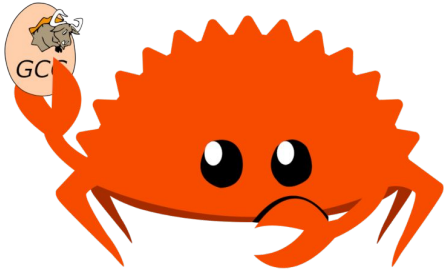
“I'm working on:

- leading the Nova driver efforts as a successor of Nouveau.
- driving and working on the enablement of the required Rust infrastructure for Nova (e.g. device / driver, PCI, I/O abstractions, ALLOC API, etc.).
- helping to enable and grow Rust infrastructure in the kernel in general as one of Nova's explicit goals.”

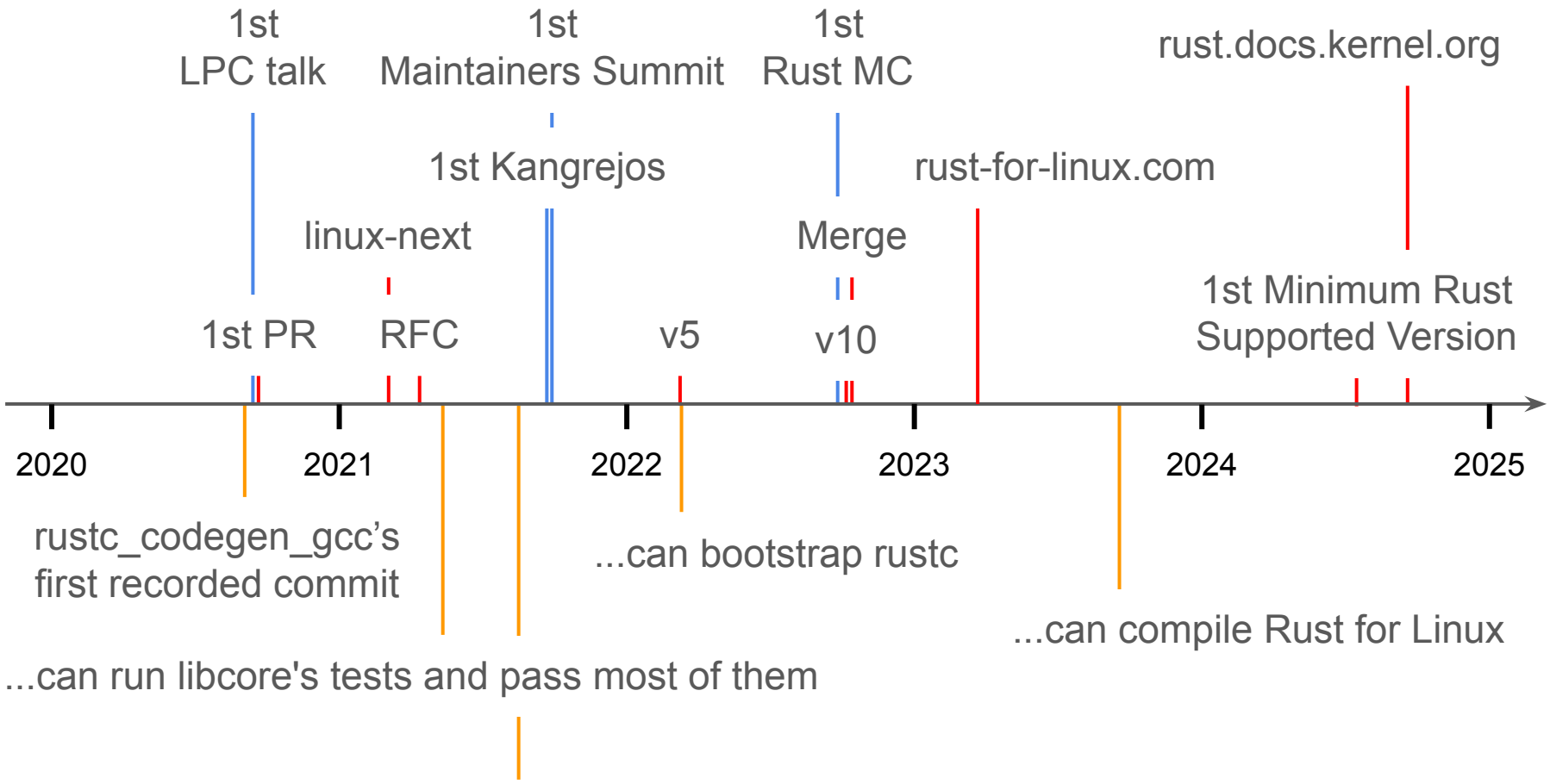
Danilo Krummrich
Red Hat

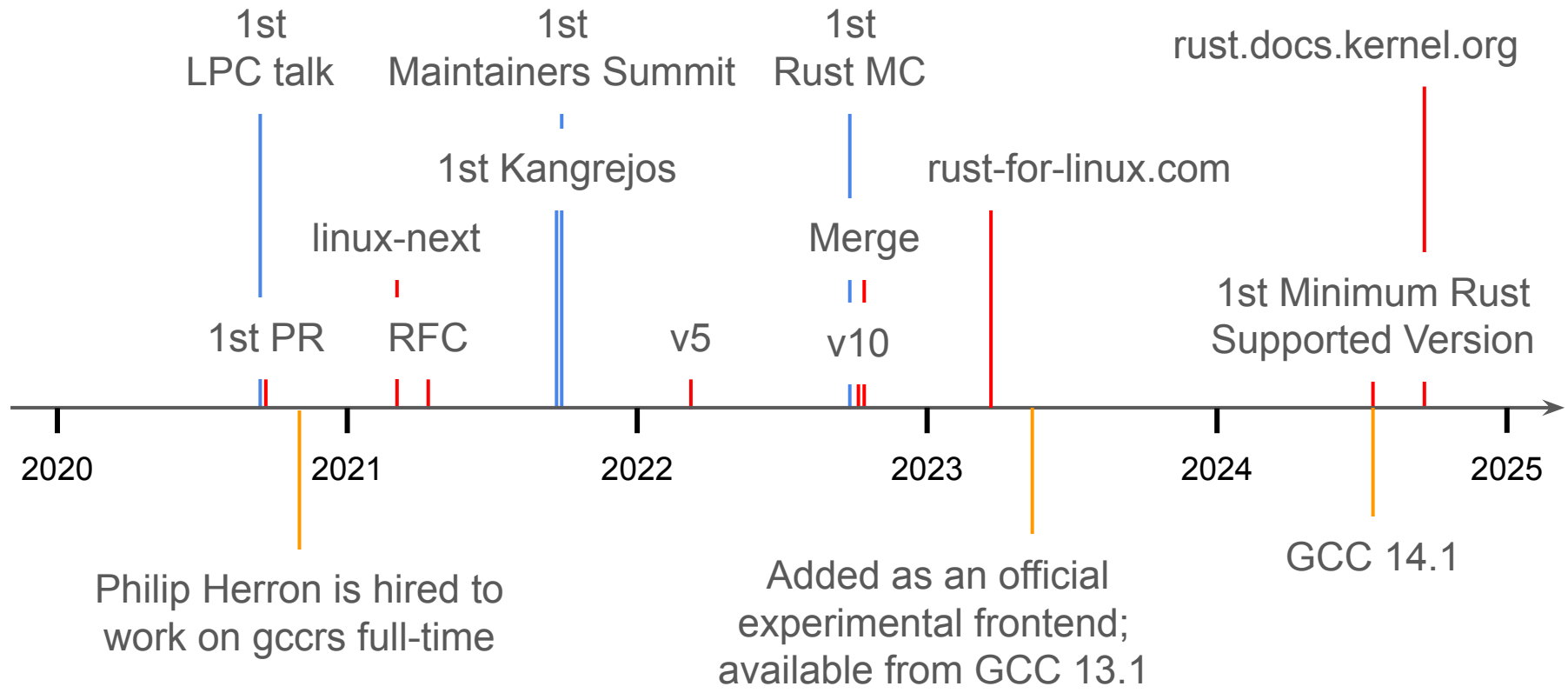
What about GCC?

`rustc_codegen_gcc`



gccrs


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st Minimum Rust
Supported Version
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
rustc_codegen_gcc's
first recorded commit
...can bootstrap rustc
...can compile Rust for Linux
...can run libcore's tests and pass most of them
...can compile Rust code on a target not supported by LLVM at the time (m68k)


1st
LPC talk
1st
Maintainers Summit
1st
Rust MC
rust.docs.kernel.org
1st Kangrejos
rust-for-linux.com
linux-next
Merge
1st Minimum Rust
Supported Version
1st PR
RFC
v5
v10
2020
2021
2022
2023
2024
2025
Philip Herron is hired to
work on gccrs full-time
Added as an official
experimental frontend;
available from GCC 13.1
GCC 14.1
gccrs started in 2014
Hopefully compiling
core in GCC 15.1

What kernel developers think about Rust?

Let’s take a look...

All mistakes and bolding are mine.

“**2025 will be the year of Rust GPU drivers.** I am confident that a lot will be achieved once the main abstractions are in place, and so far, progress has been steady, with **engineers from multiple companies joining together** to tackle each piece of the puzzle. DRM maintainers have been very receptive too, as they see a clear path for Rust and C to coexist in a way that doesn’t break their established maintainership processes. In fact, **given all the buy-in from maintainers, companies and engineers, I’d say that Rust is definitely here to stay in this part of the kernel**.”

Daniel Almeida
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“I think it's great that Linux is demonstrating how to integrate Rust into existing large C programs. The **challenge of writing correct and idiomatic bindings is nontrivial** and can indeed pose problems for developers that are less versed in Rust; but **Linux is showing that it's possible** and I'm sure that **other projects will follow**.”

Paolo Bonzini
Kernel maintainer

“I think the introduction of Rust in the kernel is **one of the most exciting development experiments** we've seen in a long time.

Beyond the well-known technical benefits, particularly around safety and memory management, one of the most interesting aspects, in my view, is **the potential to attract a new generation of developers to the kernel**, that could bring fresh ideas and new perspectives.

That said, many open questions remain, especially around tooling, integration with the existing kernel subsystems, long-term maintenance and how Rust will be adopted for core kernel development.

The enthusiasm from some parts of the kernel community is undeniable, so it seems almost inevitable that we will continue to see more Rust code merged upstream. However, like any big change in the kernel, **broader adoption will probably take time**.”

Andrea Righi
Kernel developer

“Rust for Linux (RfL) is on a good shape and makes quite good progress - each kernel cycle's pull summary gives a impressive overview of the details. Recently the RfL project made a big step with Danilo's device patches. And with Viresh's property_present() a first step for device tree handling is done. Which will be needed to give the ARM support a big push. Additionally, getting the support from some core kernel developers like Greg helps the project a lot!

On the other hand the RfL project needs to be careful to not become a victim of its own success. The interest and the number of people working on Rust for Linux is growing. And with this the number of discussions, proposals and patches increase. **The project needs to be careful to not overload the maintainers and distribute the increasing workload to more shoulders.**”

Dirk Behme
Kernel developer

“The Rust-for-Linux project is groundbreaking but built on solid ground. It's groundbreaking because Rust offers security guarantees that Linux has never enjoyed before, and solidly built because changes to the kernel itself are made slowly and with extreme care.

Still, the project faces unique challenges. **Rust's biggest weakness, as a language, is that relatively few people speak it**. Indeed, Rust is not a language for beginners, and systems-level development complicates things even more. That said, the Linux kernel project has historically attracted developers who love challenging software -- if there's an open source group willing to put the extra effort for a better OS, it's the kernel devs.”

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“Personally, I don't love the idea of trying to make 'standard' C completely memory-secure; a good knife should be sharp enough to cut you!”

Carlos Bilbao
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“Started with rust around 1-2 years prior that I started with the RfL work. At this time I would guess that **user space rust is helpful but also not too much when using both a custom alloc and no-std**. Kinda depends on the projects but the overall way of writing the language will of course apply.”

Do you think Rust will continue growing in the kernel? Do you think it should?

“I sure hope so, but also genuinely think it will, not in all subsystems for now but it feels helpful to have rust manage lifetimes and also scope some things when written in rust.”

Fiona Behrens
Kernel developer

If your company is using Rust for kernel-related tasks (perhaps internally), could you share any thoughts about it? If not, have you had any conversations internally about it?

“I’m the only person in my team at my company that does kernel development at all. There, **I usually write C drivers** (LED drivers for hardware that don’t have mainline support). **But now, I am currently translating one of those drivers to rust with the idea to also mainline it** and if that is successful just bin the C one (which is at this point less complete then the rust one).”

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“Well, it mostly attracted me. I did some C patches before but the work I do in the kernel is mostly rust. It just feels easier to use as the documentation is IMHO a bit better (though that also **makes writing abstractions harder** because you have to come up with documentation for C functions that are abstracted in rust, where **the C function often is lacking documentation**).”

Fiona Behrens
Kernel developer

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“Could be fun, but also don’t see to much reason it in. For me the two languages just have different concepts. I still use both languages as some other projects (zephyr) don’t have rust support, and stuff like how zephyr handles the device tree would for now just be a pain in rust. **Adding memory safety would be rather nice, but I fear that it would more likely destroy the language and make it less usable than actually help.**”

Fiona Behrens
Kernel developer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language?

“**The language is interesting**, and I have no real opinion about it the syntax. I played a little with it in user space, and **I just absolutely hate the cargo concept**. It is like pip and to me I hate having to pull down other code that I do not trust. At least with shared libraries, I can trust a third party to have done the build and all that. Also, if you have a large library it does bloat the applications that use it.”

What features do you think will be useful for kernel development, especially around your area?

“**The obvious one is the memory safety it gives. That's really the downfall of C.** And probably some of the template like features of C++ that it brings.”

Steven Rostedt
Kernel maintainer

Do you think it will be hard to learn for other kernel developers?

“**Yes ;-)**

It requires thinking differently, and some of the syntax is a little counter intuitive. Especially the use of '!' for macros, but I did get use to it after a while.”

Do you think Rust will continue growing in the kernel? Do you think it should?

“**Yes and yes**. Although, I also see things like cleanup.h making C more robust, and **if a subset of C becomes as safe as rust, it may make rust obsolete. But it requires the thread of rust to push that**. So even if rust doesn't become the defacto language and something else takes over, I believe rust would have played a large role in moving that way.”

Steven Rostedt
Kernel maintainer

If your company is using Rust for kernel-related tasks (perhaps internally), could you share any thoughts about it? If not, have you had any conversations internally about it?

“**I'm sure my company uses it**, as I see patches from people in my company. But not in my department (chromebooks) is more about just making the kernel stable and not really about new development. Although, we do new development for the scheduler and other core code, but **core code is not yet a candidate for rust code**.”

Steven Rostedt
Kernel maintainer

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“I think you can answer that better than I can. I'm too old school to know ;-)

But **I feel rust is more of a language that younger developers want to learn**, and C is their dad's language.”

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“Heh, I wrote the above before seeing this. And yes, I would love C to get better features. This is why I'm now a big user of the cleanup.h code. **I hope that C gets even more memory safety features!** And as I said previously, I do believe that push is due to the introduction of rust.”

Steven Rostedt
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“In my opinion, **Rust is the biggest advance in systems programming languages in decades**, perhaps since C.

To put this in some historical and future context, my dream for a far off future is for a systems programming language where embedded, compiler checked correctness proofs are practical and ergonomic.

We know, from Rice's theorem, that practical correctness checking has to come from type system improvements, and the future that's evolving now in research languages is looking like dependent types.

But **the first thing that had to be solved, for systems languages, was effective and practical checking of memory references**: hence the borrow checker, part of the type system in Rust.

That immediately solves the biggest issue with C, and it even improves upon other memory safe languages by constraining side effects (mutable ^ shared references) in a way that gets us some of the desirable properties of pure functional languages.”

Kent Overstreet
Kernel maintainer

Do you think Rust will continue growing in the kernel? Do you think it should?

“Yes, and I very much look forward to that.”

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“I believe so. It's getting harder and harder to find good C programmers among the younger generation, much the same as with assembly a few decades ago. And **C, and even more so kernel C, have an enormous learning curve which Rust helps a great deal with**.”

Kent Overstreet
Kernel maintainer

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“I would much rather see those engineering resources put towards increased adoption of Rust.

The security story with C has actually gotten dramatically better over the past decade or so, with the most recent advance being widespread practical effective fuzz testing. We're well into the "long tail" of how secure we can make C, further improvements will require bigger investments for smaller gains.

But we're potentially not that far from being able to write most new code in Rust instead of C, especially if we can get past some of the process issues that have been slowing things down.

The kernel, by default, tends to be rather slow and nitpicky: dot every i and cross every t. But there's a time and a place for that, and **in the early stages of a project's lifecycle - like Rust in the kernel - we should be prioritizing experimentation and rapid iteration**. Also, Rust is dramatically easier and safer to refactor than C - meaning it's easier to go back and fix mistakes.”

Kent Overstreet
Kernel maintainer

Any impressions you had over the course of the history of the project...

“Rust people, in my experience, have been universally great to work with. **It's a wide open field, with lots of interesting technical problems still to be solved and cool work waiting to be done.** There's a lot to be optimistic about :)”

Kent Overstreet
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“On the kernel side, I've been mostly just watching but I've been using rust quite a bit (as much as I can actually) in userspace projects including sched_ext schedulers. **I think memory safety and more structured design and programming which are encouraged and enforced by the language would be beneficial for kernel (while also being an obstacle to adoption that is)**. The standard containers and other crates add a lot of convenience but I'm unsure how well they can translate to kernel due to control over memory allocation and locking.”

Tejun Heo
Kernel maintainer

Do you think Rust will continue growing in the kernel? Do you think it should?

“**I sure hope so**.”

If your company is using Rust for kernel-related tasks (perhaps internally), could you share any thoughts about it? If not, have you had any conversations internally about it?

“Nothing directly on kernel but **a lot of, if not most, new things around kernel are switching to rust.**”

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“I don't know. Both have pretty steep learning curves after all, but it seems clear that **there will be more programmers who are familiar with rust going forward**.”

Tejun Heo
Kernel maintainer

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“I don't think I know enough to have a strong opinion here and maybe smarter people can come up with a clean-ish way to integrate memory safety into C but **I'm skeptical that the end result would be able to maintain the positive aspects of both languages** especially given that how thorough memory safety's impact on rust language itself and code written on it seems.”

Any impressions you had over the course of the history of the project...

“Looks like an **uphill battle** but I hope you guys persevere as there are substantial **long-term gains** to be had here.”

Tejun Heo
Kernel maintainer

“I can't yet write a single Rust program, yet **I'll be considering it for anything new and serious for the kernel provided we get gcc support**. I recently learned that the way in which we can leverage Coccinelle rules for APIs in the kernel for example are not needed for Rust -- the rules to follow APIs are provided by the compiler for us. Similarly **formal verification may actually be easier with Rust than in C**, and in the future we may be able to embed preconditions and postconditions as part of routines which may include unsafe functions to assist with formal verification. All this makes Rust extremely appealing towards the future.

I'd recommend however for Rust *kernel* folks to work on public OKRs to more easily and collaboratively prioritize work and requirements by the community. Not being able to use gcc with Rust makes it currently a deal breaker for built-in code. To me that's O1 for getting Rust into any serious part of the Linux kernel.”

Luis Chamberlain
Kernel maintainer

“Although I support the idea of introducing Rust to the kernel, I still have mixed feelings about it. I believe Rust is the language we need to make low-level development safer, including kernel development. I enjoy programming in Rust, as its structure allows us to build more powerful abstractions. However, I still have doubts about the best way to move the kernel in this direction. **With more than 30 years of history, the kernel has deeply rooted values and structures, making it difficult to change its course**. While some might suggest developing a new kernel, I still believe in RfL. Our community has the maturity to discuss, find compromises, and work toward a shared goal. **For Rust to succeed in the kernel, it cannot be a second-class citizen; it must be a first-class citizen, the gold standard for developing new drivers.**”

Maíra Canal
Kernel maintainer

“Using rust code in the kernel is an interesting development and I expect we will see more use over time. **If/when I can make the time for it I definitely want to look into using rust for e.g. device-drivers myself.**”

Hans de Goede
Kernel maintainer

Do you think it will be hard to learn for other kernel developers?

“In my case, it is. I am really open to including Rust and trying out what benefits it brings. Yet personally, **I have zero bandwidth to learn it and no customer I have will pay me for learning it.** I watched some high level talks about Rust in Linux and am positive about it. But I still have no experience with the language.”

Do you think Rust will continue growing in the kernel? Do you think it should?

“**Yes and yes**. Like Linus said at the Maintainers Summit 2023, we can always revert it if it fails ;) But frankly, I don't think this will happen.”

Wolfram Sang
Kernel maintainer

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“I think it has to, otherwise it will fail. **The biggest problem I see is with reviewing Rust code. I simply cannot review a driver written in Rust**. Especially given that a reviewer should ideally have larger experience with the language than the contributor. **So, I need assistance from someone who is able to do that. I can assist then with my subsystem specific knowledge** and, hopefully, it will work out. Probably learning Rust bit-for-bit this way. It kinda worked for DT bindings in YAML. We will have to see if this can be a blueprint for Rust.

I know you guys are aware of the problem. Maybe it is not a problem now because the amount of Rust code can easily be handled. I don't know.”

Any impressions you had over the course of the history of the project...

“I like your constant effort of getting feedback from all of the Kernel community. Making sure everybody gets heard and can provide input. Kudos!”

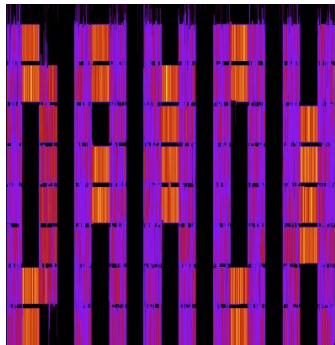
Wolfram Sang
Kernel maintainer

“I should confess that I'm not a big fan of Rust. **I once tried to read an online book for Rust years ago. My impression was that it feels more complicated than I wanted**, so I stopped reading the book, and haven't tried Rust at all so far.

I'm optimistic about Rust for Linux in long term, though. I believe the project is continuously getting attraction from many developers, and it is well maintained with good culture. I believe it will make Linux not only memory-safer, but also helps attract newer and better contributors. I'm particularly convinced with Miguel's announcement of the Rust for Linux experiment success, which was made at LPC 2024.

After the announcement, I started planning to use Rust on DAMON, to make it memory safer and attract more developers. My plan is to start from writing DAMON sample modules. Also, a developer recently reached out to me with their plan to write Rust bindings for DAMON. No real progress is made so far, but **I'm optimistic about the future of Rust for Linux and DAMON**.”

SeongJae Park
Kernel maintainer



If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“I've used Rust a little bit in personal userspace projects, but the kernel has been where I've spent most of my time. This is not the easiest path into Rust, as it hits all of the nasty bits of unsafe code, and complex behaviour. **Spending a bunch of time writing userspace projects first would probably help, though this can be hard to justify for a busy maintainer**, especially since some of the userspace Rust (particularly 3rd-part crates) won't apply in the kernel.

Favourite feature: the enum type.
Scariest feature: proc macros.”

Do you think Rust will continue growing in the kernel? Do you think it should?

“Yes, I think Rust will continue to eke out a bigger place in the kernel. **I suspect there are some parts of the kernel which will adopt Rust more quickly and completely, and some which will need to remain C-only for a long time**, in order to support systems (or people) who are not ready for Rust yet.”

David Gow
Kernel maintainer

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“Probably yes to both, but it's a bit of a frustrating path: the projects have some significant differences in style and opinion, and working on the kernel involves using some of the most difficult bits of Rust, and working with Rust bindings involves touching some of the more subtle bits of the kernel. This will hopefully get easier with time, but **everyone involved will still need to make compromises**.”

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“**I'd definitely love to see C pick up more memory safety features**: both as core parts of the language, and as implementation features and tools like sanitizers. Kees' work in the kernel has been a really good example of how to do this well, I think.”

David Gow
Kernel maintainer

Any impressions you had over the course of the history of the project...

“Some random impressions:

- **Kernel Rust has a different flavour to idiomatic userspace Rust** (different APIs, different build system, failable allocations, no dependencies, etc). This isn't necessarily a _bad_ thing (and kernel C is different to standard C as well), and we shouldn't be too afraid to let kernel Rust diverge a bit.

- **Rust as a language is evolving very quickly**. This is great, because blockers which are making kernel code difficult/impossible are being resolved quickly, but also really annoying because it's necessary to update tools all of the time. The wider compiler version support introduced recently has been a great help here, but it's not a perfect solution, as new versions are regularly introducing new warnings which need fixing, so testing on lots of versions is required anyway. This will hopefully settle down a bit over time.

- **Support for more architectures is important**, and it's making great progress. The Rust roadmap is exciting here, as is work on gccrs and rustc-backend-gcc.

- Making these structural changes in the kernel project has a very high latency, but moves quickly once it really gets going. I think Rust-for-Linux is starting to pick up momentum, and the fact that progress can seem slow (particularly for people used to the Rust project's fast pace) may seem discouraging, but **I think we're starting to see some really tangible progress, and it'll only get faster from here**.”

David Gow
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“I find **any sufficiently complex Rust codebase is largely unreadable without rust-analyzer**. This makes reviewing patches more painful since I need to import them into an enabled editor, even for patches touching code I'm familiar with, instead of reviewing inline with the patch thread. So reviews from my phone when I'm on the go are not always possible.”

Do you think Rust will continue growing in the kernel?

“**Yes**.”

Do you think it should?

“**Neutral**.”

Anonymous kernel maintainer

If your company is using Rust for kernel-related tasks (perhaps internally), could you share any thoughts about it? If not, have you had any conversations internally about it?

“Not really kernel, but we use Rust as the user space component with the vfio kernel driver. They're generally pretty good. I know many people have said this, but it's also my experience that **developing new features in Rust tends to require more time fighting the compiler, but much less time debugging runtime failures**. Sometimes crazy complex stuff runs correctly the very first time, and that's a pretty interesting experience compared to developing similarly complex implementations in C.”

Anonymous kernel maintainer

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“We need younger people interested and involved with Linux kernel, and in my experience, **fewer college grads are learning C**. The Rust language supposedly adds to the pool of talent that would otherwise not readily be able or want to contribute, so I guess that's good.”

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“I think it would be difficult to get Standard C to do such things. More likely would get **GNU and LLVM to make extensions targeting specific cases**. The auto cleanup seems like a step in the right direction.”

Any impressions you had over the course of the history of the project...

“**It seems unnecessarily controversial**.”

Anonymous kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

“I've been using Rust mostly in userspace so far. Using it seriously inside the kernel is for a while one my TODO, but at my dayjob there was so far no opportunity and in my spare time I'm busy with other things.

On learning Rust: If you know OCaml or Haskell, Rust feels more or less familiar. On the other hand, **from a pure C background thinking in Rust can be a tough job**.”

Do you think Rust will continue growing in the kernel? Do you think it should?

“Yes, it will and I deserves a fair chance. **Resistance from some subsystems or individuals is expected, but that's okay**.”

Anonymous kernel maintainer #2

If your company is using Rust for kernel-related tasks (perhaps internally), could you share any thoughts about it? If not, have you had any conversations internally about it?

“Sadly at my dayjob I've touched Rust only in userspace.”

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

“For sure! It important for a project like Linux to gain new people and new ideas. On the other hand, newbies need to understand that kernel maintainers are super careful and often don't see the benefit of maintaining a second language. But **time will resolve all these issues**.”

Anonymous kernel maintainer #2

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

“Yes! **I really hope that C will gain sooner or later such features**. The __counted_by attribute is such an example, I hope to see more soon.”

Any impressions you had over the course of the history of the project...

“I think **Rust guys don't really know that they have already won. Both Linus and GregKH want to give Rust a chance, that a *huge* win**. Sure, the progress is slow, much slower than some guys expected. But Linux is a huge and complicated project with many individuals involved.”

Anonymous kernel maintainer #2

“**I'm a strong supporter of Rust in the kernel, although I do worry that it will make it even more difficult to make pervasive treewide changes** in the future. I have been meaning to get more involved with Rust for Linux, and I am subscribed to the mailing list, and hopefully I will be able to engage more in the near future.

I am reasonably familiar with Rust. I wrote and maintain a Rust implementation of EFI for arm64 virtual machines in QEMU [0], which is based on an efiloader crate which I wrote for this purpose. I also co-maintain the aarch64-paging crate with a fellow googler, which is another effort that originated in my firmware implementation.

One of the things on my personal to-do list is actually to run this EFI code inside the kernel, which will be useful for securely booting EFI payloads such as systemd UKI images. **Integrating a Rust crate in that manner is going to create all kinds of challenges, I expect**, so I'll surely come and find you and the team for advice.

[0] https://github.com/ardbiesheuvel/efilite

I wrote this as a proof-of-concept, to prove that -on arm64- booting a QEMU/kvm VM with minimal firmware is actually faster than booting a VM with no firmware at all. I won't bore you with the details, but some people love to hate EFI for being slow, and so having a fast (and safe!) implementation that is actually faster than nothing was quite useful.”

Ard Biesheuvel
Kernel maintainer

If you have tried Rust so far, perhaps in userspace, what are your impressions of the language? What features do you think will be useful for kernel development, especially around your area? Do you think it will be hard to learn for other kernel developers?

"The jury is still out, but **I suspect that the safety features will be useful -- and also hard to learn** for many developers."

Do you think Rust will continue growing in the kernel? Do you think it should?

"Slowly at first, with later results depending on experience."

Will Rust in the kernel attract more contributors (to Rust or to the kernel)?

"**Rust appears to be the new hotness in universities**, so perhaps it will attract more new-college graduates to the kernel. Success stories would of course attract existing kernel developers to Rust, especially in parts of the kernel that have suffered from memory-safety issues."

Paul E. McKenney
Kernel maintainer

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones?

"**I believe that there will be memory-safety features in C++, maybe later in C**, but I would not be surprised if they ended up differing significantly from those of Rust in order to (1) support legacy C++/C code, (2) to better address specific existing use cases, and (3) to also handle other safety issues. **I do not expect any of this to be free from spirited discussions and other forms of controversy. ;-)**"

Any impressions you had over the course of the history of the project...

"**The patience and persistence has been good, and will likely be required going forward**. (Hey, do you want to change the world or don't you?)"

Paul E. McKenney
Kernel maintainer

“I've been working exclusively in Rust for the last 3 months, and **I don't ever want to go back to C based development again**.

Rust makes so many of the things I think about when writing C a non-issue. I spend way less time dealing with stupid bugs, I just have to get it to compile.

The lifetime rules make everything so nice, there's way less details to get wrong. I've been writing C code in the linux kernel for 20 years and I still make the same stupid mistakes during development.

Rust enables me to be faster, more accurate, more bugfree, so I can focus on the actual important part of my job, solving problems.

I have enjoyed some of the RAII things that Peter has added to give us some more modern lifetime features in C, but I think that unless C makes some pretty fundamental, radical changes it will never be able to compete with Rust when it comes to safety and ease of use.

I wish Rust were more successful in the linux kernel, and it will be eventually. Unfortunately I do not have the patience to wait that long, I will be working on other projects where I can utilize Rust. I think Rust will make the whole system better and hopefully will attract more developers.”

Josef Bacik
Kernel maintainer

On 2eff01ee2881 ("Merge tag 'char-misc-6.13-rc1' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/char-misc"):

"...

- rust misc driver bindings and other rust changes to make misc drivers actually possible.

  I think this is the tipping point, expect to see way more rust drivers going forward now that these bindings are present. Next merge window hopefully we will have pci and platform drivers working, which will fully enable almost all driver subsystems to start accepting (or at least getting) rust drivers.

  This is the end result of a lot of work from a lot of people, congrats to all of them for getting this far, you've proved many of us wrong in the best way possible, working code :)

..."

Greg Kroah-Hartman
Kernel maintainer

“In my mind, the Rust for Linux project has already achieved an important goal: **proving that Rust is indeed a viable and desirable language for kernel development**. I think that the discussion on whether we should go that way has run its course; as they say, it's all over but the shouting. Of course, this is the kernel community, so **we should expect a fair amount more shouting still**. **This work is important for the long-term viability of Linux, and I am glad that it is succeeding.**”

Jonathan Corbet
LWN editor and kernel maintainer

What other stakeholders think?

Let’s also take a look...

All mistakes and bolding are mine.

“Personally, I quite like Rust as a language. I'm a programming language enthusiast, and **seeing affine types used in a production language has been really interesting**. Integrating it into a large existing project like the kernel is a ton of work, though. **I think it will be many years until Rust is fully integrated, if it ever is.**”

Daroc Alden
LWN editor

“I am very happy and proud to see Rust used in the Linux kernel. :) I think **R4L has pushed Rust towards stabilizing features that will also be useful elsewhere** and that would have taken a lot longer to materialize otherwise (e.g. the smart pointer work), which is good, though we have to be a bit careful to avoid an unhealthily narrow focus here. **It's also been a great challenge for Rust as the kernel has quite special needs**, stretching Rust's desire to provide safety by default into new domains (I am thinking of the entire "target modifier" project here) -- but I think this, too, will benefit Rust in other low-level / embedded domains. I didn't mean to become an expert in "terrible flags that can break your code if you use them incorrectly", but I am grateful that the R4L folks are taking this concern seriously and investing the time and effort to find Rust-y solutions to these challenges.”

Ralf Jung
Rust t-opsem & t-miri lead

“**Rust for Linux is a big deal for Rust**. It helps us validate the bare metal and kernel space use case and bring it to maturity. It also helps us focus our efforts on features that people have been wanting for a long time. The working group has felt very collaborative and I've enjoyed working with everyone in it.”

Would you like to see standard C or its implementations get memory-safety-related features, especially following Rust's ones? Or better interop?

“Yes! I think **adding memory safety and interop features would be an extremely positive direction for C**. It's still the lingua franca ABI, and the more we can safely express across that ABI, the better off everyone will be.”

Tyler Mandry
Rust t-lang lead

“For me, RfL is the proof of Rust potential. It's very biased but **GCC support is one of the last key blockers for having Rust everywhere**. In addition to that, a lot of (old/esoteric) platforms are not supported. **Once the GCC backend is done, hopefully RfL will get a (big) step closer to succeed**.

I don't think adding Rust support in the Linux kernel will increase the number of contributors, but it will hopefully bring new blood.”

Guillaume Gomez
Rust t-rustdoc lead

“**In the past year, collaboration between RfL and the Rust project has increased substantially, enabling stabilisation of parts of the language and toolchain that would otherwise not have seen much progress**. Given RfL's low-level requirements, these interactions are particularly meaningful and mutually beneficial, ultimately strengthening both projects.”

Urgau
Rust t-compiler member

“I believe the integration of Rust into the Linux kernel represents a significant milestone for both Rust and the kernel community. **Rust's emphasis on memory safety and concurrency could not only enhance the kernel's reliability but also set a precedent for how other systems programming languages might evolve**. The collaboration between Rust developers and kernel maintainers has been exemplary, showcasing how open-source communities can work together to push technological boundaries. This partnership could serve as a model for future integrations of innovative technologies into established systems.

Regarding GCC support, it will be really important to have because this can drive the community to have another group of people to implement the Rust language, which helps to resolve hidden bugs that we may not have discovered yet, simply because no one has tried digging inside the implementation of the compiler.

As for C, incorporating memory-safety features inspired by Rust would be really nice to see what could be built on an existing language where the 'memory-safety features' are hidden inside the expertise of the developer, and try to make these features more accessible to everyone!”

Vincenzo Palazzo
Rust wg-macros lead

“My experience with voluntary work in between Rust-for-Linux and Rust team has still been rather short-termed. In spite of this, from a personal perspective of mine, **a healthy interaction between the two organisations is being developed** under the stewardship of both the organisations and individual team members, through the aid of which I have had the chance to gather support and assistance, wherever and whenever possible.

In addition, this journey has provided a healthy amount of assurance and courage that has **greatly kindled my interest in exploring areas of the Linux kernel**. I would say that the feeling is not particularly technically driven but rather culturally motivated. It is about exchange of knowledge and ideas for the betterment. I firmly believe that **the outlook of attracting contributors onto both of the projects is becoming ever more promising**.

Somehow it reminds me of what is written in Go West from Village Boys. Together we will love the beach, learn and teach, change our pace of life and work and strive.”

Xiang Fei Ding
Rust contributor

“I think Rust in the kernel is a net positive for both the Linux project and the Rust programming language. I think it will help show unconvinced people that Rust can and should be used for serious engineering and low-level programming. **Features adapted to the language in order to create Rust-for-Linux will definitely be useful for other projects**, I think particularly in areas with important restrictions on memory.

I believe that **GCC support is an important step for Rust, be that through rustc_codegen_gcc, Rust to C backends, or gccrs**. GCC supports a lot of targets, some very old but still in use. Users of these targets deserve to be able to enjoy writing Rust, and having wider reach is a net positive for the language in my opinion.

I would like for C to get safer, but I'm not sure that it can realistically be done. If you look at Sean Baxter's efforts to bring a borrow-checker to the C++ programming language, you'll see that extra syntax is required, and that the existing C++ standard library would not work with borrow-checking. The cost to add safety features to languages which weren't designed with them in mind seems really high to me. On the other hand, hands-on studies have shown that vulnerabilities will often come from new code, not old one - **focusing on FFI and inter-operability between Rust and C/C++ seems like the correct path to me**. It would enable slowly adding Rust components to existing C/C++ programs, which would improve safety on newer code.”

Arthur Cohen
gccrs

“**Rust has an extremely extremely(2x) welcoming community, especially for beginners**. Though starting it is daunting with the borrow checker and lifetimes, it definitely allows people with low experience in C such as myself make less errors. In my opinion such things accumulate over time leading to a safer codebase in general.

The amazing Rust for Linux team is working very efficiently to make Rust a reality in the Linux kernel, and it would only be a matter of time before we start seeing it in more complex, critical systems. There is plenty of information on the internet about how great the build system and community is.

We have been working on Coccinelle For Rust for more than 2 years now and started with very little knowledge in Rust, but the communities on Zulip and the mailing lists were very helpful with its development and very clear about the problems they are facing currently.

In my opinion Rust could potentially lower the bar for entry into memory and safety critical systems while still maintaining the quality of code, as one does not have to make as many considerations as one does with C.”

Tathagata Roy
Coccinelle for Rust

What about companies?

“In the context of the Nova project Red Hat dedicates three fulltime positions to the Rust for Linux project.”

“Samsung employs a full-time engineer to work on Rust for Linux.”

“The Linux kernel provides the security foundation for many of Google's products such as Android. More than just about any other component, it's important that the kernel provide robust security, and itself be robust against security vulnerabilities. Using the memory safe language Rust is a crucial step in fortifying the kernel against vulnerabilities and bolstering its overall security posture. We are already working on multiple Rust drivers.”

Lars Bergstrom
Director of Engineering
Google Android

“Rust Community’s contribution to Linux stands tall as a hopeful example of a true inclusive global open-source collaboration that is paving the way toward a future in which safe, open and secure computing can serve all of humanity. We are grateful to Miguel and the rest of ‘Rust for Linux' team for their dedication and relentless pursuit of building a better tomorrow.”

Sid Askary
Technical Director of Open-Source
Futurewei Technologies

Sponsors & Industry support

— https://rust-for-linux.com/sponsors
— https://rust-for-linux.com/industry-and-academia-support
— https://www.memorysafety.org/initiative/linux-kernel/
— https://www.memorysafety.org/blog/rustls-and-rust-for-linux-funding-openssf/

How to contribute?

How to contribute?

https://rust-for-linux.com/contributing

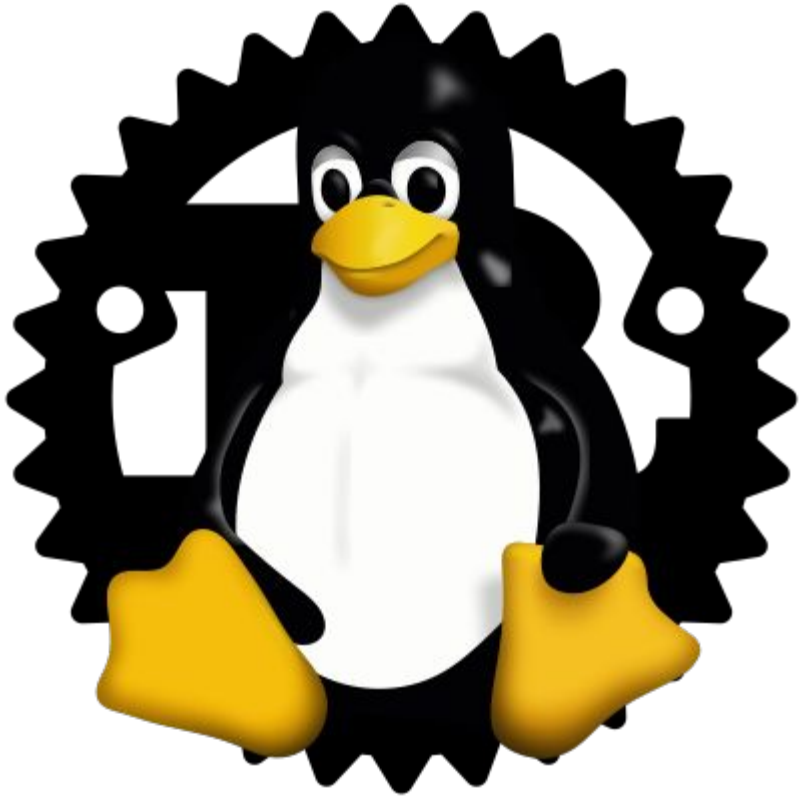
Kangrejos 2022, Oviedo, Spain

Kangrejos 2023, Gijón, Spain

— https://kangrejos.com

Kangrejos 2024, Copenhagen, Denmark

— https://kangrejos.com



Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Backup slides

Linux in perspective

One can think of Linux as a project with lots of subprojects.

In the v6.10 cycle (9 weeks), around:

13,312 non-merge commits were pulled.

1,918 developers contributed.

242 made their first kernel contribution.

203 companies supported work in the kernel.

1,800 unique kernel maintainers are listed.

— https://lwn.net/Articles/981559/

What is Rust for Linux?

Is Rust for Linux a Rust project?

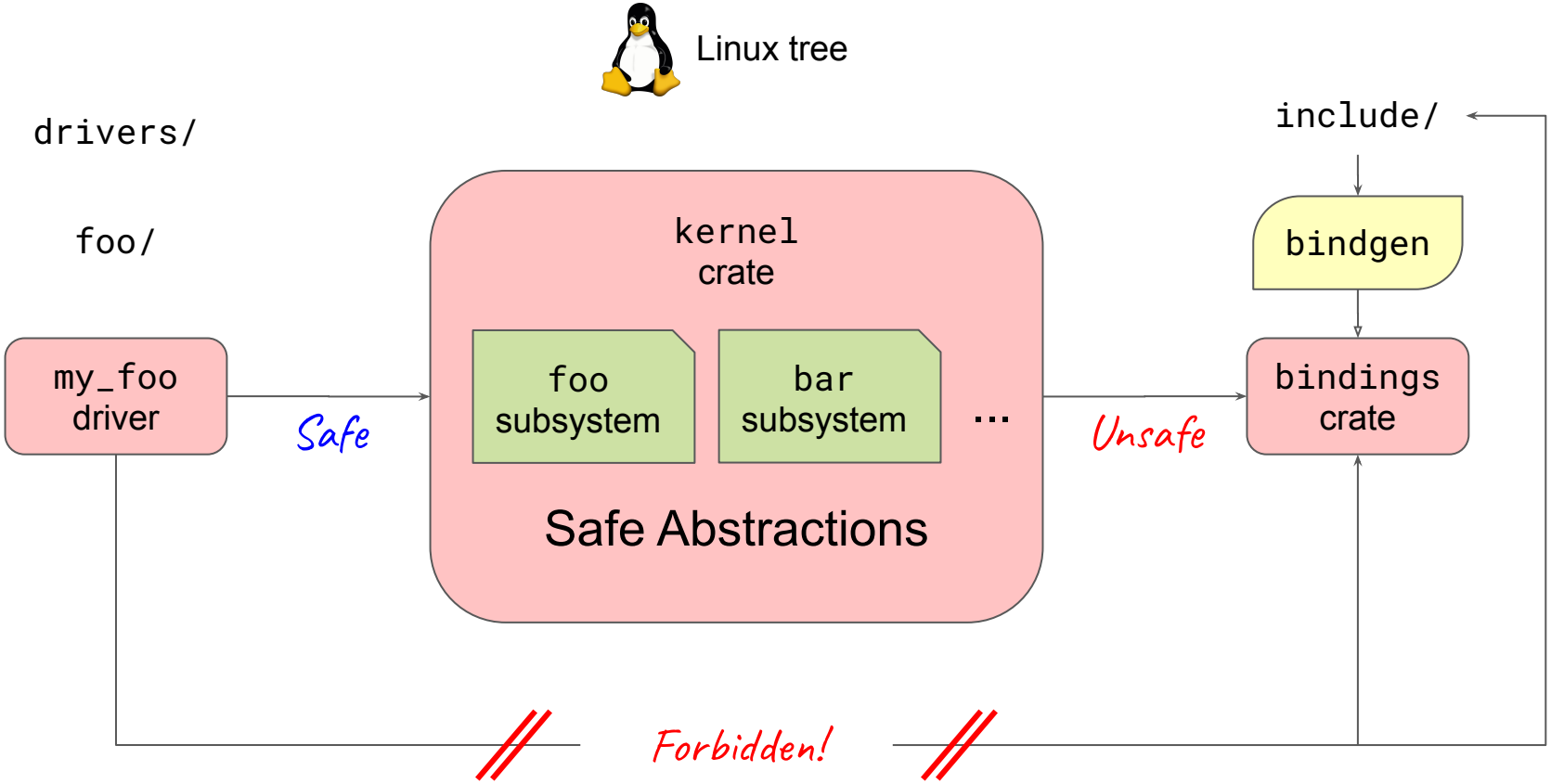
No, although some of us collaborate in Rust or are part of teams there.

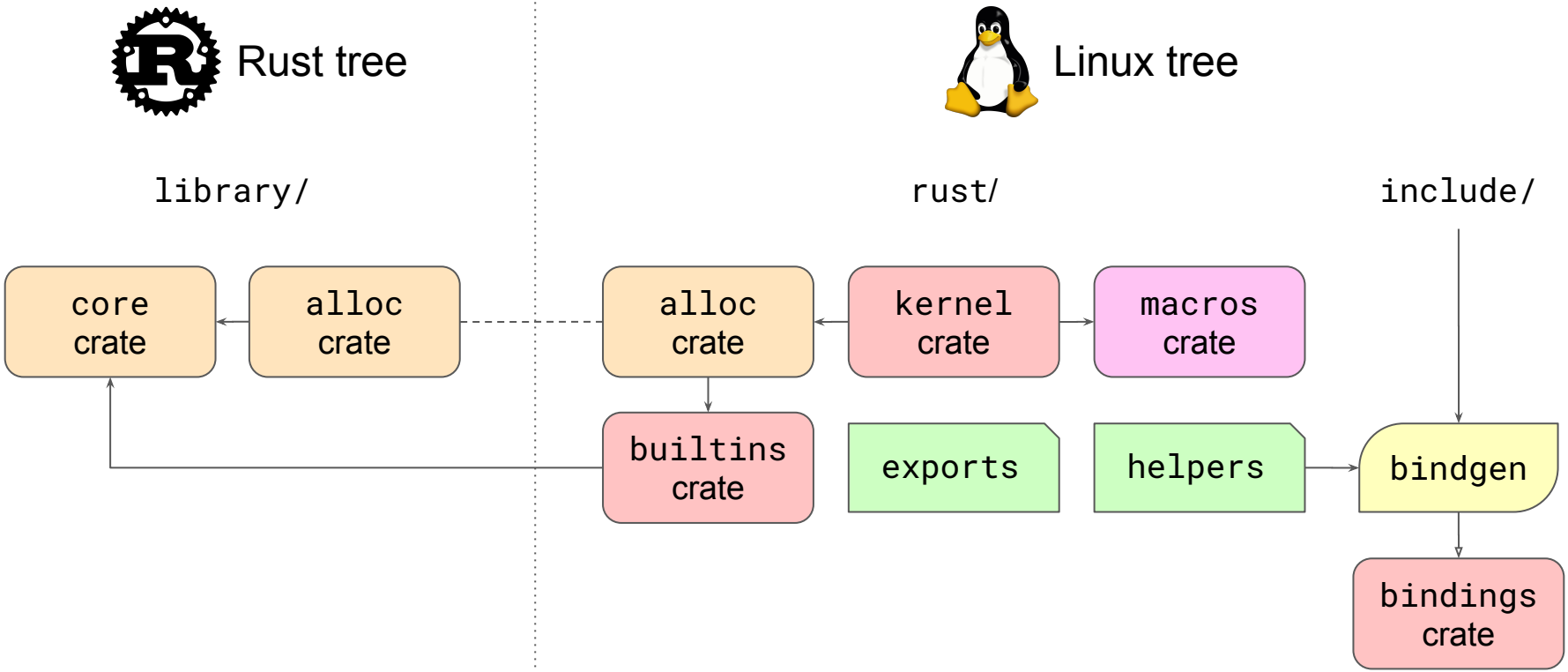
Is Rust for Linux a kernel project?

Yes, we are part of the kernel.

However, the project is not really only about kernel changes.

Rust for Linux is really a project involving a few other projects.

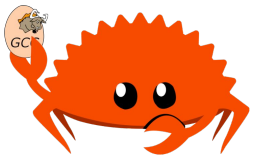

Linux tree
drivers/
foo/
my_foo
driver
Safe
kernel
crate
foo
subsystem
bar
subsystem
...
Safe Abstractions
Unsafe
include/
bindgen
bindings
crate
Forbidden!


Rust tree
Linux tree
library/
rust/
include/
core crate
alloc crate
alloc crate
kernel crate
macros crate
builtins crate
exports
helpers
bindgen
bindings crate

“We are still focused on getting the Rust standard library to compile with gccrs. To that end, we undertook a massive rewrite of our name resolution pass which is paying off and enabling us to handle the complex import and export structure within the Rust `std` crate. We are currently working on missing language features for handling the standard library, such as compiling the question-mark operator (`<expr>? `) and for-loops. This required a big re-engineering of our AST and HIR data structures in order to accept "lang item paths", paths that refer to a specific, compiler-known item which is used for desugaring and codegen - things like the `Sized` trait, or `Result::Err`, or the `IntoIterator::into_iter` function. We also added support for auto-traits, which are required for properly handling the well-known `Send` and `Sync` traits in the standard library, among other auto traits.

We continue to fix bugs in our codegen pass and in our type system, which we are discovering as time passes. **We feel very close to compiling the Rust `core` library, and are hoping to be able to do so with the release of GCC 15.1 this spring. This will enable us to start experimenting with Rust-for-Linux, which we would like to be able to test during the summer 2025**.

On the social side of things, we got a [blogpost published on the official Rust blog] with the intent of clarifying the relationship between the official Rust project, the Rust community, and gccrs. It was very well received, and mentions some of the technical changes we are making to the compiler to permit reusing rustc components such as the polonius borrow-checker, or, in the future, the next-gen trait solver.”



Arthur Cohen
gccrs

“One big update since February last year is that we added **initial support for debug information** in rustc_codegen_gcc. [1]”

“There's some progress towards **rustup distribution**.”

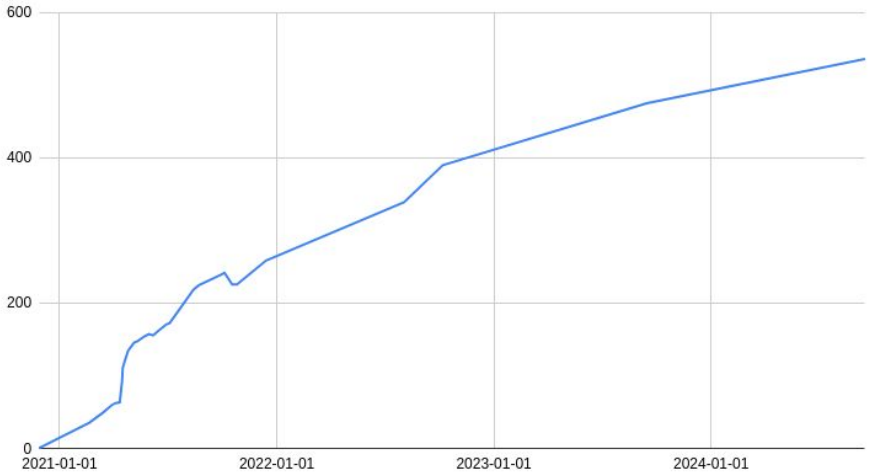
[1] https://blog.antoyo.xyz/rustc_codegen_gcc-progress-report-31

Antoni Boucher
rustc_codegen_gcc

Growing Community

536 subscribers in the `rust-for-linux` mailing list.

From ~460 last year.

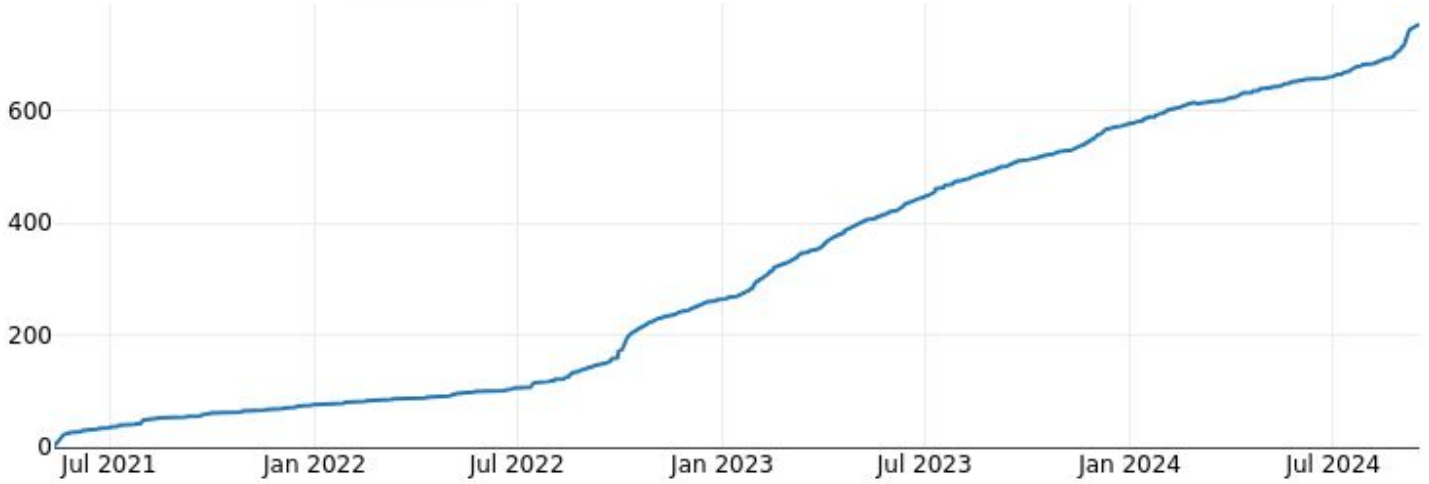


— https://subspace.kernel.org/vger.kernel.org.html

Growing Community

754 users in the Zulip instance (i.e. chat).

From ~530 last year.

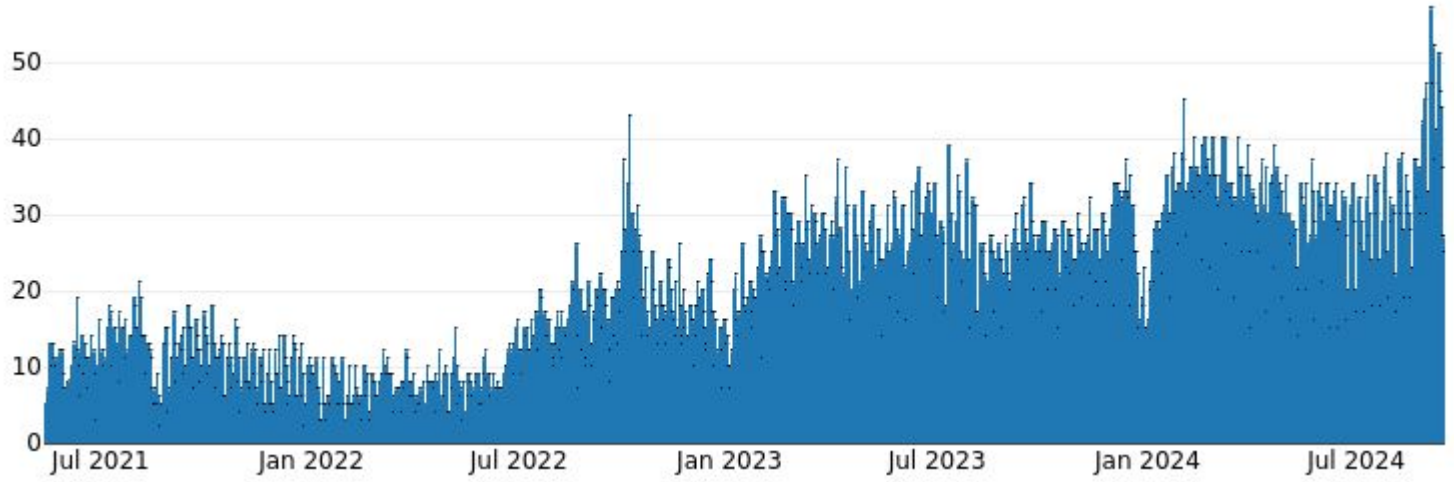


— https://rust-for-linux.zulipchat.com/stats

Growing Community

~30 daily active users in the Zulip instance (i.e. chat).

From ~25 last year.

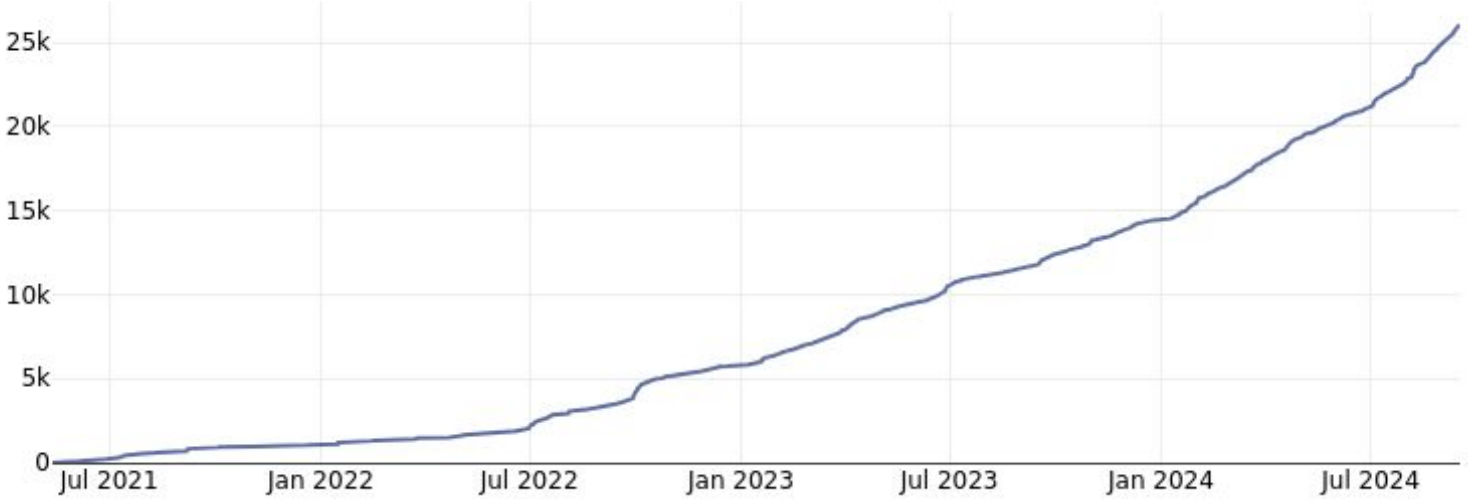


— https://rust-for-linux.zulipchat.com/stats

Growing Community

~25k messages sent in the Zulip instance (i.e. chat).

From ~12k last year.



— https://rust-for-linux.zulipchat.com/stats

Doubled the patch series submitters

Aakash Sen Sharma
Alexander Pantyukhin
Alexey Dobriyan
Alex Mantel
Alice Ryhl
Anders Roxell
Andrea Righi
Andreas Hindborg
Andrew Ballance
Andrey Konovalov
Antonio Hickey
Ariel Miculas
Arnaldo Carvalho de Melo
Asahi Lina
Aswin Unnikrishnan
Ayush Singh
Bagas Sanjaya
Ben Gooding
Benno Lossin
Björn Roy Baron
Boqun Feng
Bo-Wei Chen
Breno Leitao
Carlos Bilbao
Charalampos Mitrodimas
Christian Marangi
Christian Schrefl
Christina Quast
Conor Dooley
Costa Shulyupin
Daniel Almeida
Danilo Krummrich
David Gow
David Rheinsberg
Dirk Behme
Ethan D. Twardy
Felipe Alves
Filipe Xavier
Fiona Behrens
Francesco Zardi
FUJITA Tomonori
Gary Guo
Guillaume Plourde
Helen Koike
Hridesh MG
Ian Rogers
Jamie Cunliffe
Jiapeng Chong
Jiaxun Yang
Jiri Olsa
Jocelyn Falempe
John Hubbard
Jon Mulder
Jubilee Young
Laine Taffin Altman
Laura Nao
Lyude Paul
Maíra Canal
Manmohan Shukla
Martin Rodriguez Reboredo
Masahiro Yamada
Mathys-Gasnier
Matteo Croce
Matt Gilbride
Matthew Leach
Matthew Maurer
Michael Ellerman
Michael Vetter
Michal Rostecki
Michele Dalle Rive
Miguel Ojeda
Mika Westerberg
Mitchell Levy
Neal Gompa
Nell Shamrell-Harrington
Nick Desaulniers
Obei Sideg
Olof Johansson
Paran Lee
Patrick Blass
Patrick Miller
Pierre Gondois
Qingsong Chen
Roland Xu
Roy Matero
Sami Tolvanen
Sarthak Singh
SeongJae Park
Sergio González Collado
Siddharth Menon
Suren Baghdasaryan
Thomas Bamelis
Thomas Bertschinger
Thorsten Blum
Timo Grautstück
Trevor Gross
TruongSinh Tran-Nguyen
Valentin Obst
Vinay Varma
Vincent Woltmann
Vincenzo Palazzo
Viresh Kumar
Vlastimil Babka
WANG Rui
Wedson Almeida Filho
Wei Liu
Wu XiangCheng
Yang Yingliang
Yanteng Si
Yiyang Wu
Yutaro Ohno
Zehui Xu
Zheng Yejian
Zigit Zo

Latest developments

6.8: Rust 1.74.1, LoongArch, `srctree`-relative links, Kbuild improvements, Rust PHY abstractions and Asix PHY "Rust reference driver" (first one)...

6.9: Rust 1.76.0 (2 less unstable features), arm64, `container_of!` macro, `time` module, `CondVar` methods, documentation cleanup series, first Rust Kselftest...

6.10: Rust 1.78.0 (1 less unstable feature), RISC-V, dropped `alloc` in-tree fork (~30 language and ~60 library less unstable features), DWARFv5 and zlib/zstd support, GFP allocation flags support in `Box`/`Vec`/`Arc`... (1 less unstable feature), `Ktime` abstraction, methods for `CStr`/`CString`/`Arc`/`ArcBorrow`, `#[pin_data]` support for default values...

6.11: Support for multiple Rust and bindgen versions (thus support for distribution toolchains), `uaccess` module, `page` module, `device` module, `firmware` module, LLVM+Rust toolchains...

6.12: KCFI, KASAN and SCS support, `MITIGATION_*` and `objtool` support, `RUSTC_VERSION`, helpers split, `list` module (`ListArc`, `AtomicTracker`, `ListLinks`, `List`, `Iter`, `Cursor`, `ListArcField`), `rbtree` module (`RBTree`, `RBTreeNode`, `RBTreeNodeReservation`, `Iter`, `IterMut`, `Cursor`), https://rust.docs.kernel.org, Trevor joins, AMCC QT2025 PHY driver...

6.13/RFCs/WIP: generic `Allocator` (custom `alloc` crate, `KBox`/`VBox`/`KVBox`, `KVec`/`VVec`/`KVVec`), File abstractions, lints improvements and `#[expect]`, MIPS, shrinker abstraction, global lock support, `Untrusted`, custom FFI integer types, kernel (generic?) atomics, safety standard, hrtimer, codecs, tracepoints, third-party proc macro support (e.g. `syn`), `#[test]` KUnit support, new build system (`kernel` split, visibility, declarative)...

The Linux Kernel

6.11.0

Quick search

Go

Contents

Arch Linux

Arch Linux provides recent Rust releases and thus it should generally work out of the box, e.g.:

```
pacman -S rust rust-src rust-bindgen
```

Debian

Debian Unstable (Sid), outside of the freeze period, provides recent Rust releases and thus it should generally work out of the box, e.g.:

```
apt install rustc rust-src bindgen rustfmt rust-clippy
```

Fedora Linux

Fedora Linux provides recent Rust releases and thus it should generally work out of the box, e.g.:

```
dnf install rust rust-src bindgen-cli rustfmt clippy
```

Gentoo Linux

Gentoo Linux (and especially the testing branch) provides recent Rust releases and thus it should generally work out of the box, e.g.:

```
USE='rust-src rustfmt clippy' emerge dev-lang/rust dev-util/bindgen
```

`LIBCLANG_PATH` may need to be set.

— https://docs.kernel.org/rust/quick-start.html

Other toolchains support

In addition, of course, we still support `rustup` toolchains.

- Including `beta` and `nightly`.
- Very useful for development.

And the official Rust standalone installers too.

- https://forge.rust-lang.org/infra/other-installation-methods.html

— https://docs.kernel.org/rust/quick-start.html
— https://rust-for-linux.com/rust-version-policy#supported-toolchains

Other toolchains support

Nathan Chancellor kindly set up LLVM+Rust toolchains too.

https://mirrors.edge.kernel.org/pub/tools/llvm/rust/

These are based on the slim and fast LLVM builds provided in kernel.org.

Two sets are provided:

Latest LLVM: latest stable version of the major version of LLVM that Rust uses under the hood.

Matching LLVM: a matching version of LLVM that Rust uses under the hood, so that features such as cross-language LTO that may have subtle issues without the same LLVM version can be experimented with.

— https://docs.kernel.org/rust/quick-start.html
— https://rust-for-linux.com/rust-version-policy#supported-toolchains

Toolchains with latest LLVM version

Tree	Toolchain versions	aarch64	x86_64
6.10 (mainline)	LLVM 18.1.8, Rust 1.78.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.9 (stable)	LLVM 17.0.6, Rust 1.76.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.8 (old stable)	LLVM 17.0.6, Rust 1.74.1	.tar.gz .tar.xz	.tar.gz .tar.xz
6.6 (LTS)	LLVM 17.0.6, Rust 1.73.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.1 (LTS)	LLVM 14.0.6, Rust 1.62.0	.tar.gz .tar.xz	.tar.gz .tar.xz

Toolchains with matching LLVM version

Tree	Toolchain versions	aarch64	x86_64
6.10 (mainline)	LLVM 18.1.4, Rust 1.78.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.9 (stable)	LLVM 17.0.6, Rust 1.76.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.8 (old stable)	LLVM 17.0.4, Rust 1.74.1	.tar.gz .tar.xz	.tar.gz .tar.xz
6.6 (LTS)	LLVM 17.0.2, Rust 1.73.0	.tar.gz .tar.xz	.tar.gz .tar.xz
6.1 (LTS)	LLVM 14.0.5, Rust 1.62.0	.tar.gz .tar.xz	.tar.gz .tar.xz

Collaboration with Rust

Adrian Taylor
Alona Enraght-Moony
Amanieu d'Antras
Antoni Boucher
Arthur Cohen
Boxy
Christian Poveda Ruiz
Ding Xiang Fei
Ed Page
Emilio Cobos Álvarez
Erik Jonkers
Guillaume Gomez
Jakub Beránek
Josh Triplett
Jubilee
Jynn Nelson
Krishna Sundarram
Lukas Wirth
Mara Bos
Mark Rousskov
Michael Goulet
Nell Shamrell-Harrington
Nikita Popov
Niko Matsakis
Pietro Albini
Ralf Jung
Rémy Rakic
Santiago Pastorino
Serial-ATA
Sid Askary
Travis Cross
Tyler Mandry
Urgau
Vincenzo Palazzo
Waffle Maybe
Weihang Lo
Wesley Wiser

...and more!

Kangrejos

The Rust for Linux Workshop

An event where people involved in the Rust for Linux discussions can meet in a single place before LPC.

https://kangrejos.com

https://lwn.net/Archives/ConferenceIndex/#Kangrejos