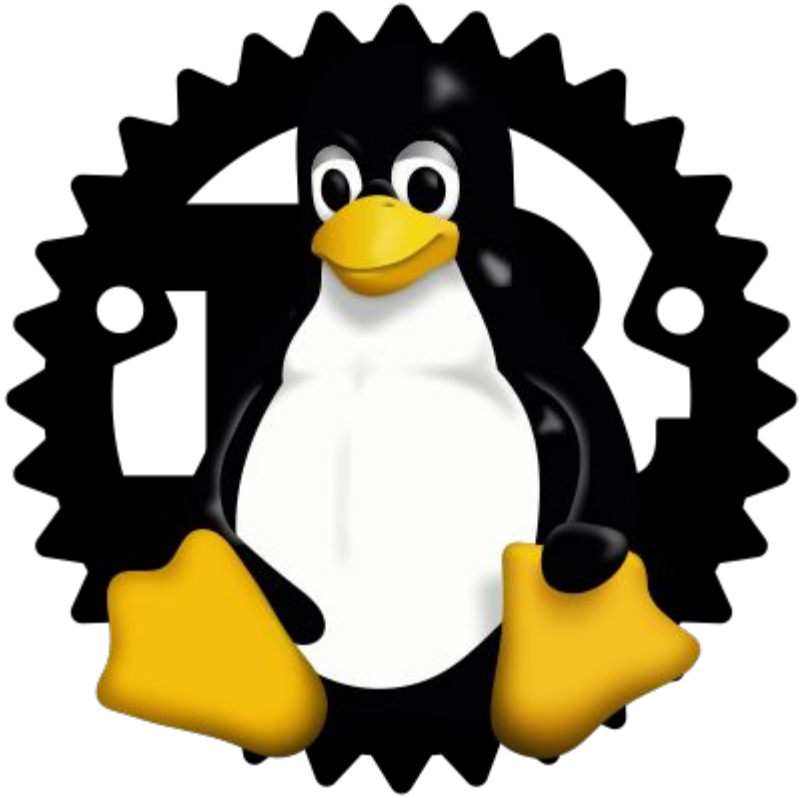


Rust for Linux

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.

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.

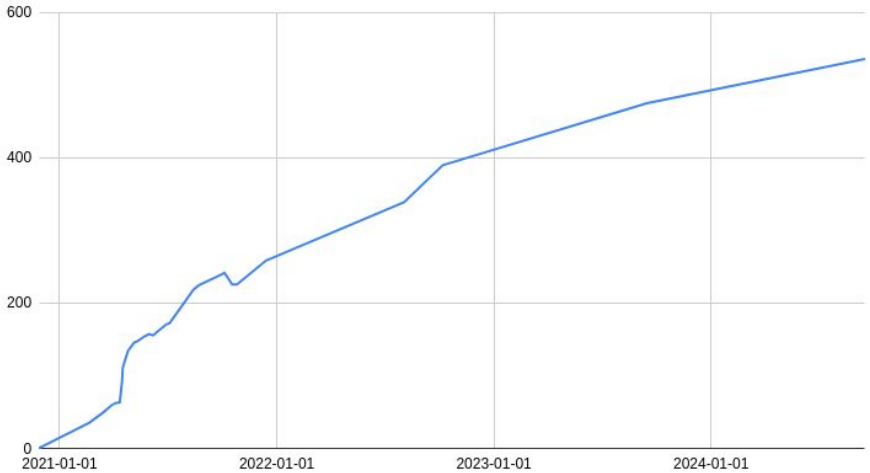
However, the project is not really only about kernel changes.

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

Growing Community

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

From ~460 last year.

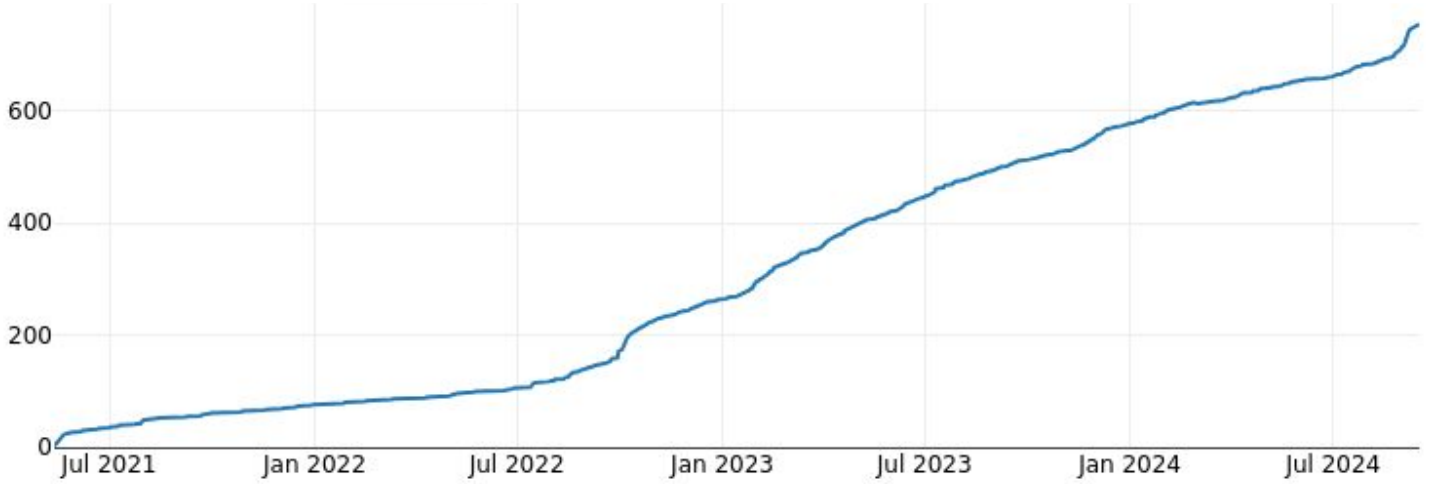


— https://subspace.kernel.org/vger.kernel.org.html

Growing Community

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

From ~530 last year.

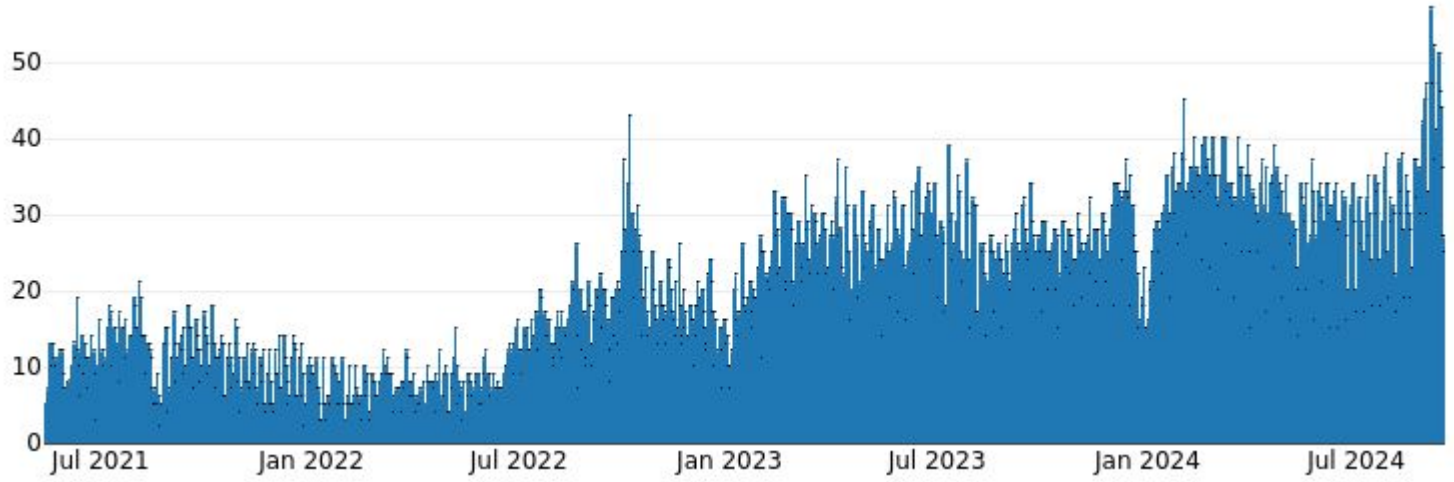


— https://rust-for-linux.zulipchat.com/stats

Growing Community

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

From ~25 last year.

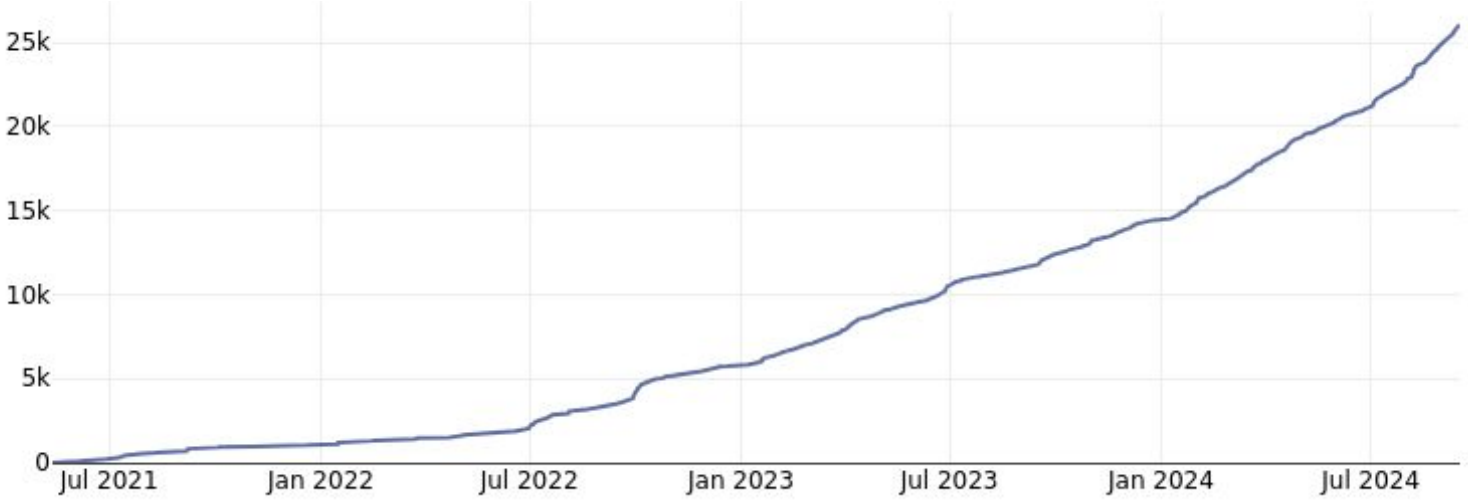


— https://rust-for-linux.zulipchat.com/stats

Growing Community

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

From ~12k last year.



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

Core Team

```
RUST
M:   Miguel Ojeda <ojeda@kernel.org>
M:   Alex Gaynor <alex.gaynor@gmail.com>
M:   Wedson Almeida Filho <wedsonaf@gmail.com>
R:   Boqun Feng <boqun.feng@gmail.com>
R:   Gary Guo <gary@garyguo.net>
R:   Björn Roy Baron <bjorn3_gh@protonmail.com>
R:   Benno Lossin <benno.lossin@proton.me>
R:   Andreas Hindborg <a.hindborg@samsung.com>
R:   Alice Ryhl <aliceryhl@google.com>
R:   Trevor Gross <tmgross@umich.edu>
L:   rust-for-linux@vger.kernel.org
S:   Supported
W:   https://rust-for-linux.com
B:   https://github.com/Rust-for-Linux/linux/issues
C:   zulip://rust-for-linux.zulipchat.com
P:   https://rust-for-linux.com/contributing
T:   git https://github.com/Rust-for-Linux/linux.git rust-next
...
```

Core Team

MAINTAINERS: add **Trevor Gross** as Rust reviewer

Trevor 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.

— Commit `68d3b6aa0870` ("MAINTAINERS: add Trevor Gross as Rust reviewer")

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)...

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 is 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 our 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

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!

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

`RUSTC_VERSION` support

- Supporting several versions implies conditional support sometimes.
  - Especially taking into account the unstable features in use.
  - For both compiler flags and source code.
- Thus the need for `RUSTC_VERSION`.
  - Including automatic reconfiguration and rebuild on `*_TEXT` changes.
  - And probing macros like `rustc-option`.
    - Commit `c42297438aee` ("kbuild: rust: Define probing macros for rustc")

— https://lore.kernel.org/rust-for-linux/20240902165535.1101978-1-ojeda@kernel.org/

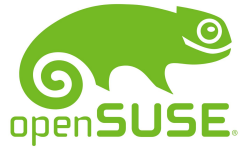
Distributions support

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

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

Each of these binaries is signed with the Rust signing key, which is available on keybase.io, by the Rust build infrastructure, with GPG. In the tables below, the `.asc` files are the signatures.

Past releases can be found in the archive.

platform	stable (1.81.0)	beta	nightly
`aarch64-apple-darwin`	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc
`aarch64-pc-windows-msvc`	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc
`aarch64-unknown-linux-gnu`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`aarch64-unknown-linux-musl`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`arm-unknown-linux-gnueabi`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`arm-unknown-linux-gnueabihf`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc

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

Each of these binaries is signed with the Rust signing key, which is available on keybase.io, by the Rust build infrastructure, with GPG. In the tables below, the `.asc` files are the signatures.

Past releases can be found in the archive.

platform	stable (1.81.0)	beta	nightly
`aarch64-apple-darwin`	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc
`aarch64-pc-windows-msvc`	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc
`aarch64-unknown-linux-gnu`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`aarch64-unknown-linux-musl`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`arm-unknown-linux-gnueabi`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc
`arm-unknown-linux-gnueabihf`	tar.xz tar.xz.asc	tar.xz tar.xz.asc	tar.xz tar.xz.asc

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

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

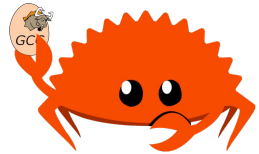
Toolchains with latest LLVM version

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

gccrs



We are in the process of fixing the bugs which prevent us from compiling `core`. Once `core` is done, we expect `alloc` to go smoothly which means the compiler should be able to be tested on the Rust parts of Linux.

One of biggest issues we are facing for compiling `core` is needing to re-engineer a big pass of our compiler pipeline (name-resolution). It was not powerful enough to handle the many complex imports, exports, glob imports, and re-exports used in `core`.

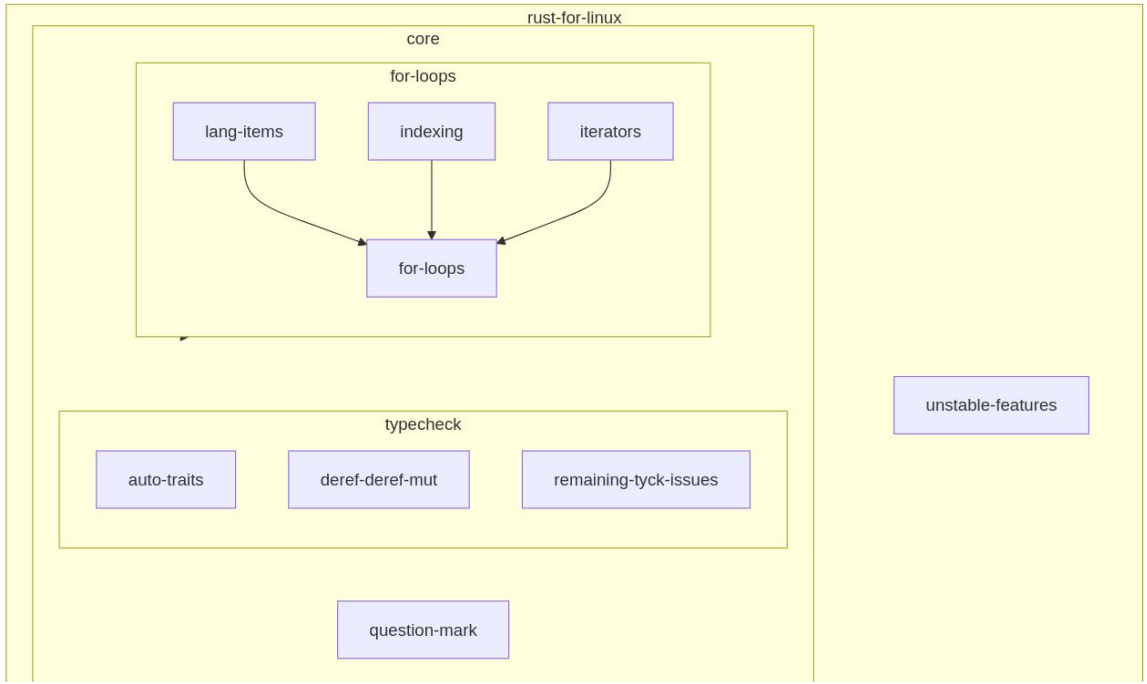
To keep working on other issues in parallel, we spent a massive amount of time "flattening" `core` so that it does not require any name-resolving to be compiled - all modules are laid out in the same file, without any imports or exports. This allowed us to expose bugs in our macro expansion, type system, and codegen, which we are now taking care of.

We have started the work to integrate the polonius borrow-checker into gccrs. We now have nice looking borrow-errors, with a good integration with the library. There are still some classes of errors that we are missing, but we are making good progress.

We are close to being able to handle inline assembly, which is necessary for `core` and presumably for the Linux kernel.

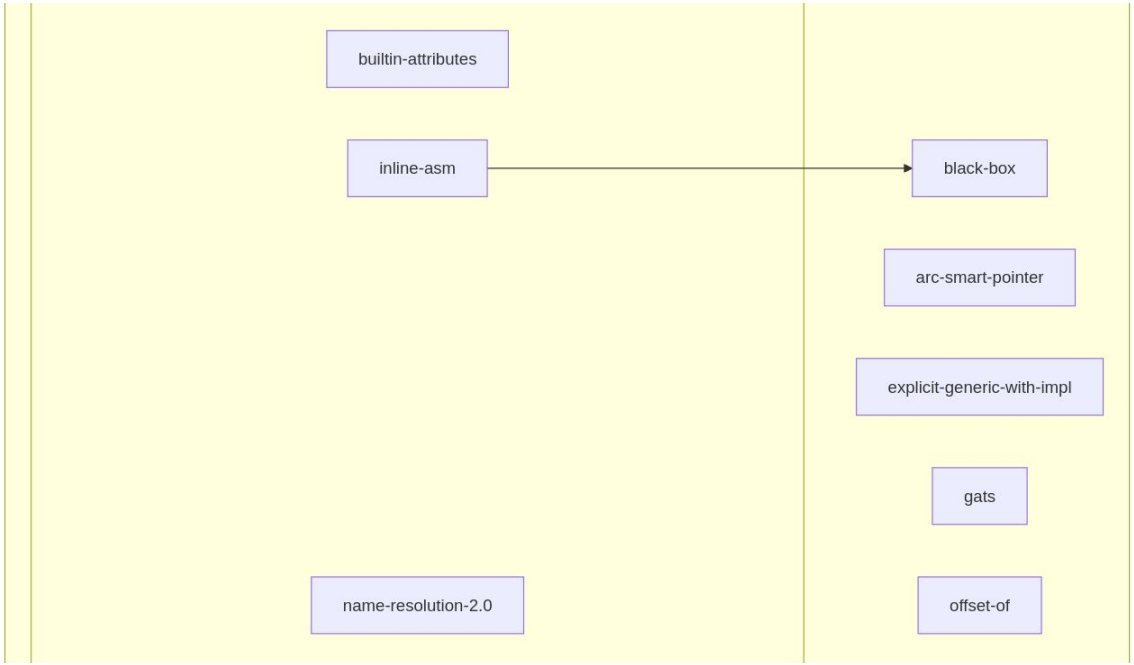
— Arthur Cohen
— https://rust-for-linux.com/gccrs
— https://rust-gcc.github.io

gccrs



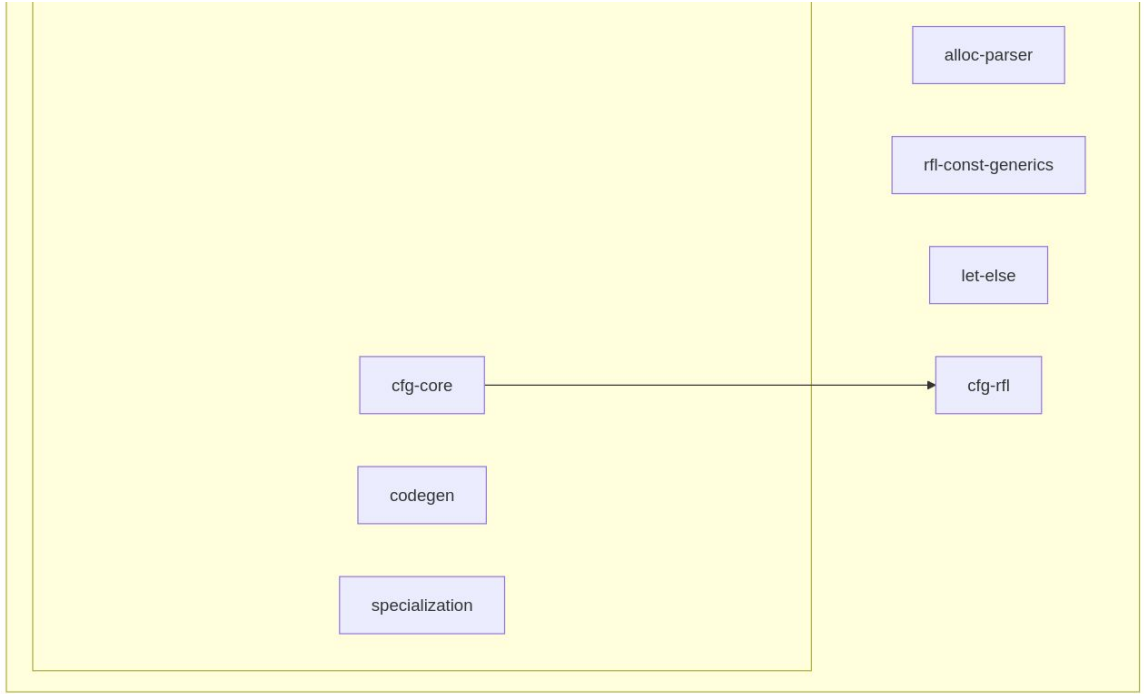
— Arthur Cohen
— https://rust-gcc.github.io/2024/09/03/towards-gcc15.1.html

gccrs



— Arthur Cohen
— https://rust-gcc.github.io/2024/09/03/towards-gcc15.1.html

gccrs



— Arthur Cohen
— https://rust-gcc.github.io/2024/09/03/towards-gcc15.1.html

`rustc_codegen_gcc`

Support for LTO (Link-Time optimization).

    Though not yet tested on Rust for Linux.

Now runs a part of `rustc_codegen_gcc`'s CI in the Rust repo.

Support for `f16`/`f128`.

Preparation work to eventually get `rustup` distribution.

(A year ago, it was the first time it could compile a vanilla kernel).

— Antoni Boucher
— https://rust-for-linux.com/rustc_codegen_gcc
— https://blog.antoyo.xyz

rust.docs.kernel.org

The code documentation is available since August 2024 at:

https://rust.docs.kernel.org

Built with `--generate-link-to-definition`.

Thanks for the support, Konstantin!

Main releases at e.g.:

https://rust.docs.kernel.org/6.11/

linux-next at:

https://rust.docs.kernel.org/next/

The future:

More crates rendered? `uapi`, `bindings`.

More tags? Deduplication for storage? Tag selector?

Crate kernel

source · [-]

[-] The `kernel` crate.

This crate contains the kernel APIs that have been ported or wrapped for usage by Rust code in the kernel and is shared by all of them.

In other words, all the rest of the Rust code in the kernel (e.g. kernel modules written in Rust) depends on `core`, `alloc` and this crate.

If you need a kernel C API that is not ported or wrapped yet here, then do so first instead of bypassing this crate.

Re-exports

```
pub use macros;
pub use uapi;
```

Modules

alloc	Extensions to the `alloc` crate.
block	Types for working with the block layer.
device	Generic devices that are part of the kernel's driver model.
error	Kernel errors.
init	API to safely and fallibly initialize pinned `struct`s using in-place constructors.
ioctl	`ioctl()` number definitions.
kunit	KUnit-based macros for Rust unit tests.
net	Networking.

RBTree

Methods

Trait

Struct kernel::rbtree::RBTree

source · [-]

```
pub struct RBTree<K, V> { /* private fields */ }
```

[–] A red-black tree with owned nodes.

It is backed by the kernel C red-black trees.

Examples

In the example below we do several operations on a tree. We note that insertions may fail if the system is out of memory.

```
use kernel::{alloc::flags, rbtree::{RBTree, RBTreeNode, RBTreeNodeReservation}};

// Create a new tree.
let mut tree = RBTree::new();

// Insert three elements.
tree.try_create_and_insert(20, 200, flags::GFP_KERNEL)?;
tree.try_create_and_insert(10, 100, flags::GFP_KERNEL)?;
tree.try_create_and_insert(30, 300, flags::GFP_KERNEL)?;

// Check the nodes we just inserted.
{
    assert_eq!(tree.get(&10).unwrap(), &100);
    assert_eq!(tree.get(&20).unwrap(), &200);
```

```
26  }
27
28  /// A Rust wrapper around a `ktime_t`.
29  #[repr(transparent)]
30  #[derive(Copy, Clone)]
31  pub struct Ktime {
32      inner: bindings::ktime_t,
33  }
34
35  impl Ktime {
36      /// Create a `Ktime` from a raw `ktime_t`.
37      #[inline]
38      pub fn from_raw(inner: bindings::ktime_t) -> Self {
39          Self { inner }
40      }
41
42      /// Get the current time using `CLOCK_MONOTONIC`.
43      #[inline]
44      pub fn ktime_get() -> Self {
45          // SAFETY: It is always safe to call `ktime_get` outside of NMI context.
46          Self::from_raw(unsafe { bindings::ktime_get() })
47      }
48
49      /// Divide the number of nanoseconds by a compile-time constant.
50      #[inline]
51      fn divns_constant<const DIV: i64>(self) -> i64 {
52          self.to_ns() / DIV
53      }
54
55      /// Returns the number of nanoseconds.
```

`MITIGATIONS_*` support

Compiler support merged into `rustc`, patch series queued for v6.12.

With both pieces, now we have x86_64 `objtool`-enabled & clean builds.

Commit `c4d7f546dd9a` ("objtool/kbuild/rust: enable objtool for Rust")
Commit `fc582dfc1f20` ("x86/rust: support MITIGATION_SLS")
Commit `d7868550d573` ("x86/rust: support MITIGATION_RETHUNK")
Commit `284a3ac4a96c` ("x86/rust: support MITIGATION_RETPOLINE")

KCFI, KASAN, SCS

KCFI support queued for v6.12.

Commit `ca627e636551` ("rust: cfi: add support for CFI_CLANG with Rust")
Commit `ce4a2620985c` ("cfi: add CONFIG_CFI_ICALL_NORMALIZE_INTEGERS")

KASAN support queued for v6.12.

Commit `e3117404b411` ("kbuild: rust: Enable KASAN support")
...and related ones

SCS support queued for v6.12.

Commit `d077242d68a3` ("rust: support for shadow call stack sanitizer")

Enforcing safety docs/comments and `#[expect]`

Enablement of some safety lints to enforce that `// SAFETY` comments and `# Safety` sections are written and only where expected.

It has been a common theme in reviews.

`#[expect]` support.

Makes the compiler warn if the diagnostic was **not** produced.

```
#[expect(dead_code)]
fn f() {}
```

Expected for v6.13.

— https://lore.kernel.org/rust-for-linux/20240904204347.168520-1-ojeda@kernel.org/

`--check-cfg`

Checks conditional compilation names and values.

An example of simple, effective collaboration between the kernel and Rust.

We tested the feature early on when they requested feedback.

We found an ergonomics issue with the kernel's ~20k `cfg`'s.

They made sure it worked for the kernel case.

The feature was stabilized in Rust 1.79.

We plan to use it soon!

— https://github.com/rust-lang/rust/pull/121202

Coccinelle for Rust

Major Changes

- Added the `...` construct, allowing matching of arbitrary control flow paths.
- Disjunctions can now be much more complicated, and not restricted to only expressions.
- Addition of the more powerful CTL-VW engine which standardizes the matching process into “CTL formulas”.

Minor Changes

- Better error reporting and handling.
- Post-transformation formatting improved.
- Visualization of Control Flow and CTL formulas added.
- More compact representation of ASTs.

— Tathagata Roy
— https://rust-for-linux.com/coccinelle-for-rust
— https://lpc.events/event/18/contributions/1787/

Coccinelle for Rust

Challenges

- Parsing problems due to the internal representation of `...` and disjunctions.
- Macro formatting.
- Complex CFG representation.
- Limited parallelization capabilities due to the thread-unsafe structure of rowan syntax nodes.

— Tathagata Roy
— https://rust-for-linux.com/coccinelle-for-rust
— https://lpc.events/event/18/contributions/1787/

bindgen

John Baublitz's `bindgen` improvements for Rust for Linux:

- Functional C macros expansion:

  https://github.com/rust-lang/rust-bindgen/issues/753
  https://github.com/rust-lang/rust-bindgen/pull/2779
  https://github.com/rust-lang/rust-bindgen/pull/2823

  Released in 0.70.0.

- Raw pointer access for bitfields:

  https://github.com/rust-lang/rust-bindgen/issues/2674
  https://github.com/rust-lang/rust-bindgen/pull/2876

- New mapping for C enums — Rust enums, but sound (with safe and unsafe conversions):

  https://github.com/rust-lang/rust-bindgen/issues/2646
  https://github.com/rust-lang/rust-bindgen/pull/2908

In a similar way to Rust, `bindgen` will include the kernel in its CI.

— https://kangrejos.com

Sponsors & Industry support

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/

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

Topic branches

Focused on a particular topic and meant to enable collaboration on code that is targeted for upstreaming but has not reached mainline yet.

Some of these branches may contain work-in-progress code (similar to staging trees) that may not be suitable for upstreaming or general usage yet.

staging/rust-pci	Danilo Krummrich
staging/rust-net	Trevor Gross and Valentin Obst
staging/rust-device	Danilo Krummrich and Philipp Stanner
staging/dev	Danilo Krummrich and Philipp Stanner

— https://rust-for-linux.com/branches

“Rust reference drivers”

Some kernel subsystems maintainers are open to the idea of experimenting with Rust, but they may want to start simple with a driver they are familiar with. But such a driver would violate the "***no duplicate drivers***" rule.

Similarly, external people have expressed an interest in writing Rust drivers, but given the required abstractions are not there, they may decide to wait. But if nobody writes a first use case, the abstractions cannot be merged without breaking the "***no code without an expected in-tree user***" rule.

Rust reference drivers are a solution to these deadlocks: they are drivers that subsystem maintainers are allowed to introduce in their subsystem without dropping the existing C driver.

— https://rust-for-linux.com/rust-reference-drivers
— [MAINTAINERS SUMMIT] The Rust Experiment

“Rust reference drivers”

To bootstrap abstractions for new drivers, i.e. not the "duplicate"/"rewritten" one, but future new drivers that would use those abstractions; while avoiding breaking the "no dead code" rule.

To serve as a reference for existing C maintainers on how such drivers would look like in Rust, as "live" documentation, e.g. like how LWN featured a 1:1 comparison between a C and Rust driver. And it has to be buildable at all times.

To use all the in-tree kernel infrastructure and **to prepare their subsystem** for Rust over time, e.g. setting up tests and CI.

To learn over time, especially for subsystems that have several maintainers where not everybody may have time for it at a given moment. Reading Rust patches from time to time for APIs one is familiar with can help a lot.

And, most importantly, **to evaluate if the effort is worth it** for their subsystem.

— https://rust-for-linux.com/rust-reference-drivers
— [MAINTAINERS SUMMIT] The Rust Experiment

More maintainers & subsystems getting involved

DRM.

Netdev.

Block (“Stage 1”, i.e. breakage allowed).

Timekeeping, hrtimer.

Driver core.

Workqueue.

Kbuild.

Module support.

KUnit, kselftest.

Architectures: LoongArch, arm64, RISC-V, MIPS.

pahole.

...

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

Watch Live (Free)
Topics
My Conference
My Sessions
My Contributions
Graphic Resources
Anti-harassment policy
Contact
FAQs
LPC 2023 site
2024
contact@linuxplumbersconf....

"Room 1.85 - 1.86", Austria Center 10:45 - 11:30

Break
"Room 1.85 - 1.86", Austria Center 11:30 - 12:00

12:00 **Rust for Linux** *Miguel Ojeda et al.*
"Room 1.85 - 1.86", Austria Center 12:00 - 12:45

An Investigation of Patch Porting Practices of the Linux Kernel Ecosystem *Mr Chengyu Song et al.*

13:00
"Room 1.85 - 1.86", Austria Center 12:45 - 13:30

Lunch

14:00
"Room 1.85 - 1.86", Austria Center 13:30 - 15:00

15:00 **Graceful Under Pressure: Prioritizing Shutdown to Protect Your Data in Embedded Systems (Even When the Power Flickers)**
Oleksij Rempel
"Room 1.85 - 1.86", Austria Center 15:00 - 15:45

Journey of a C kernel engineer starting a Rust driver project *Danilo Krummrich*

16:00
"Room 1.85 - 1.86", Austria Center 15:45 - 16:30

Break
"Room 1.85 - 1.86", Austria Center 16:30 - 17:00

17:00 **Introducing the power sequencing subsystem** *Bartosz Golaszewski*
"Room 1.85 - 1.86", Austria Center 17:00 - 17:45

Wed 18/09

Print | PDF | Full screen | Detailed view | Filter

10:00	**Coccinelle for Rust**	*Tathagata Roy et al.*	
	"Room 1.31-1.32", Austria Center		10:00 - 10:30
	Introduction to Rust: Quality of Life Beyond Memory Safety	*Benno Lossin*	
	"Room 1.31-1.32", Austria Center		10:30 - 11:00
11:00	**Giving Rust a chance for in-kernel codecs**	*Daniel Almeida*	
	"Room 1.31-1.32", Austria Center		11:00 - 11:30
	Break		
	"Room 1.31-1.32", Austria Center		11:30 - 12:00
12:00	**`hrtimer` Rust Abstractions**	*Mr Andreas Hindborg*	
	"Room 1.31-1.32", Austria Center		12:00 - 12:30
	Atomics and memory model for Rust code in kernel	*Boqun Feng*	
	"Room 1.31-1.32", Austria Center		12:30 - 13:00
13:00	**Birds of a feather**		
	"Room 1.31-1.32", Austria Center		13:00 - 13:30

Maintainers Summit

My takeaway is:

Encourage maintainers to experiment and merge code.

Do not worry too much about mistakes until one has an actual user.

Iterating in-tree and/or seeing early approaches may help maintainers.

Different subsystems may want to approach things differently.

Breaking Rust code may be a fine approach for a subsystem.

It is OK if it takes a couple years for everybody to get comfortable.

— https://lwn.net/Articles/990740/

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

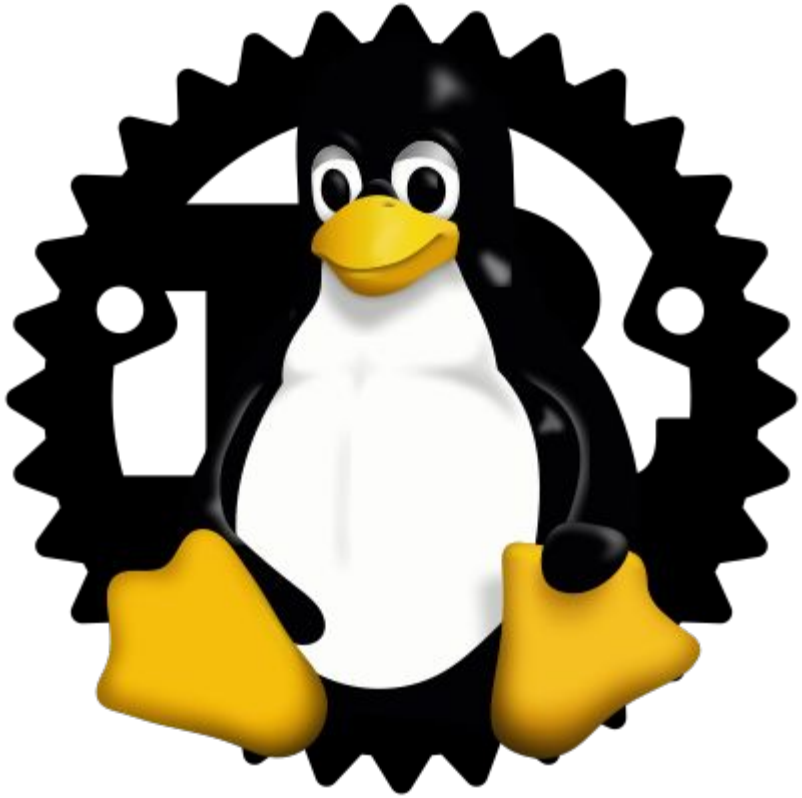
Kangrejos 2022, Oviedo, Spain

Kangrejos 2023, Gijón, Spain

— https://kangrejos.com

Kangrejos 2024, Copenhagen, Denmark

— https://kangrejos.com

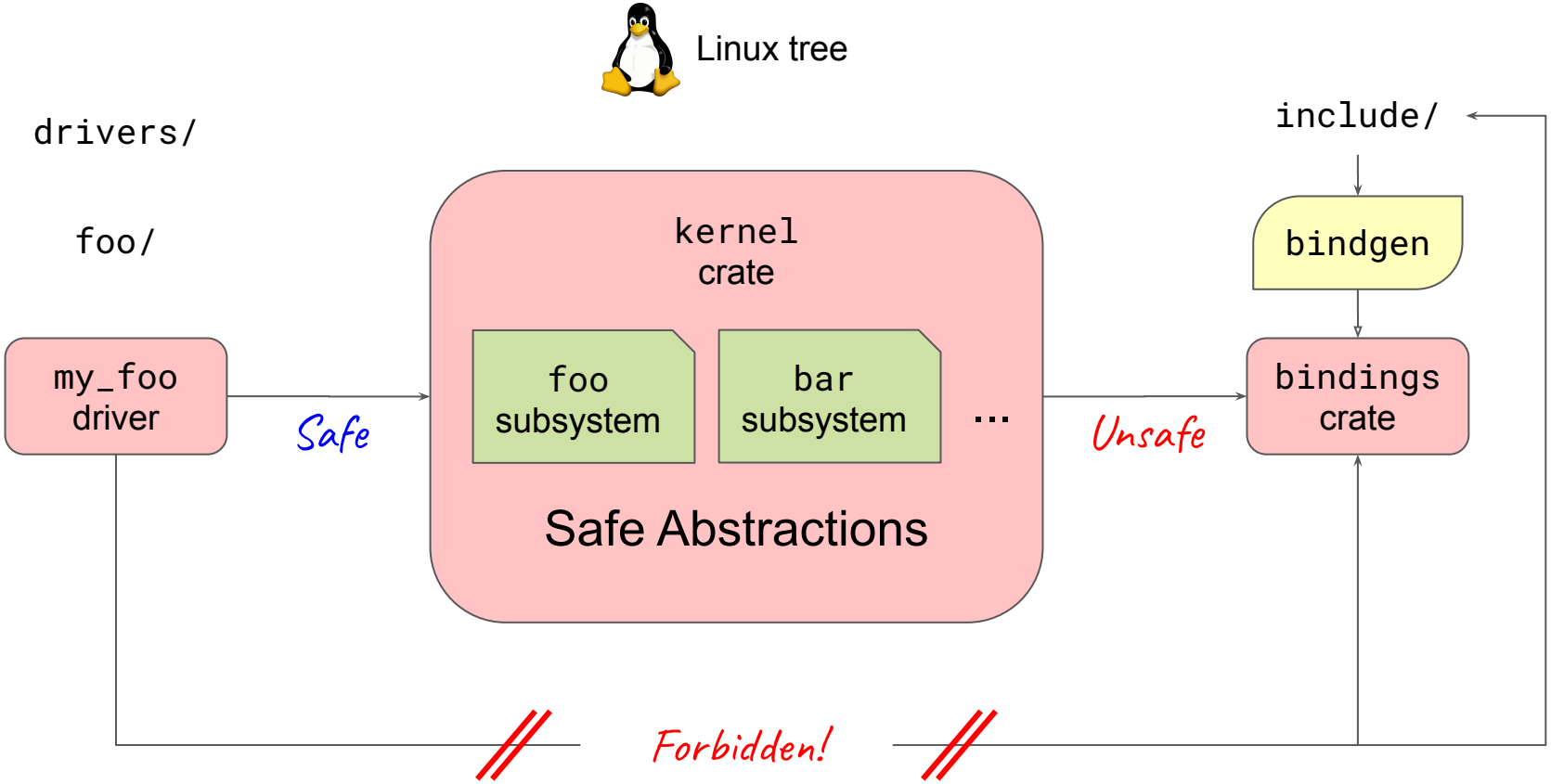


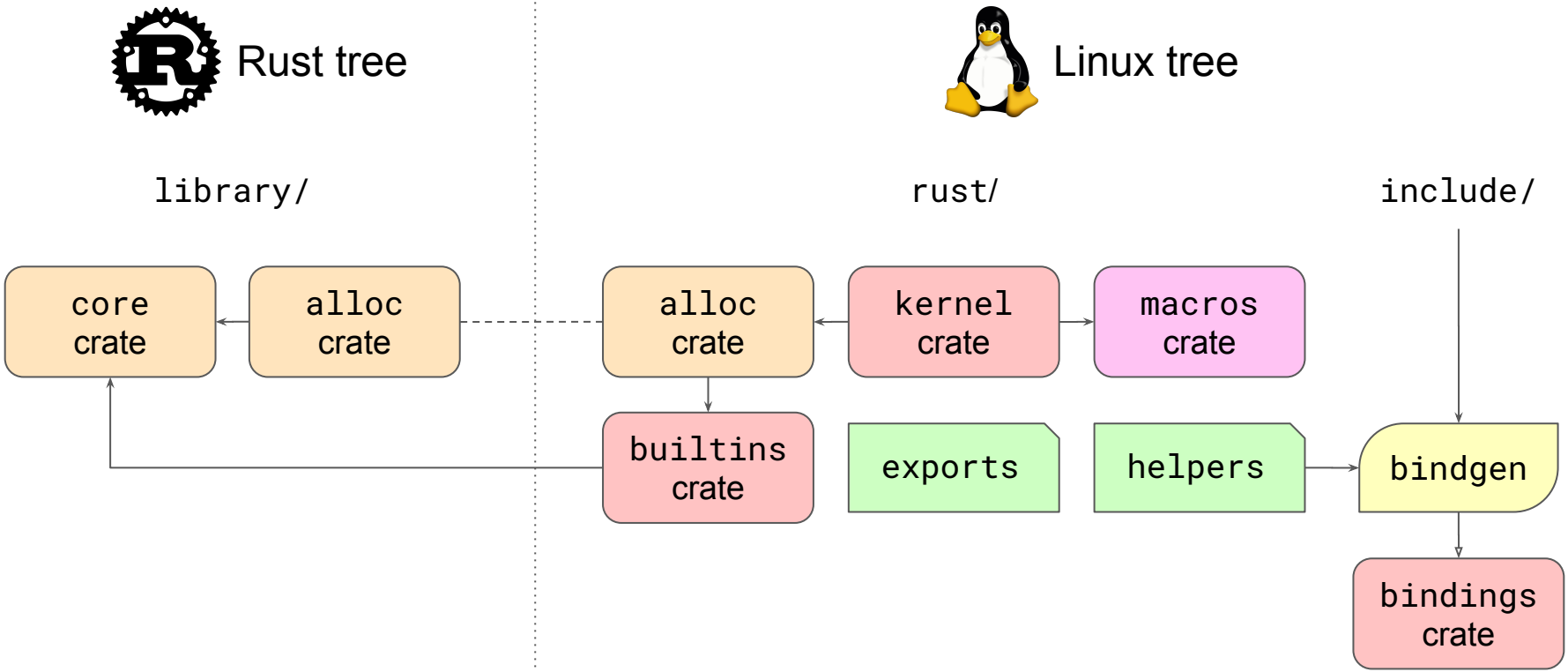
Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Backup slides


Linux 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