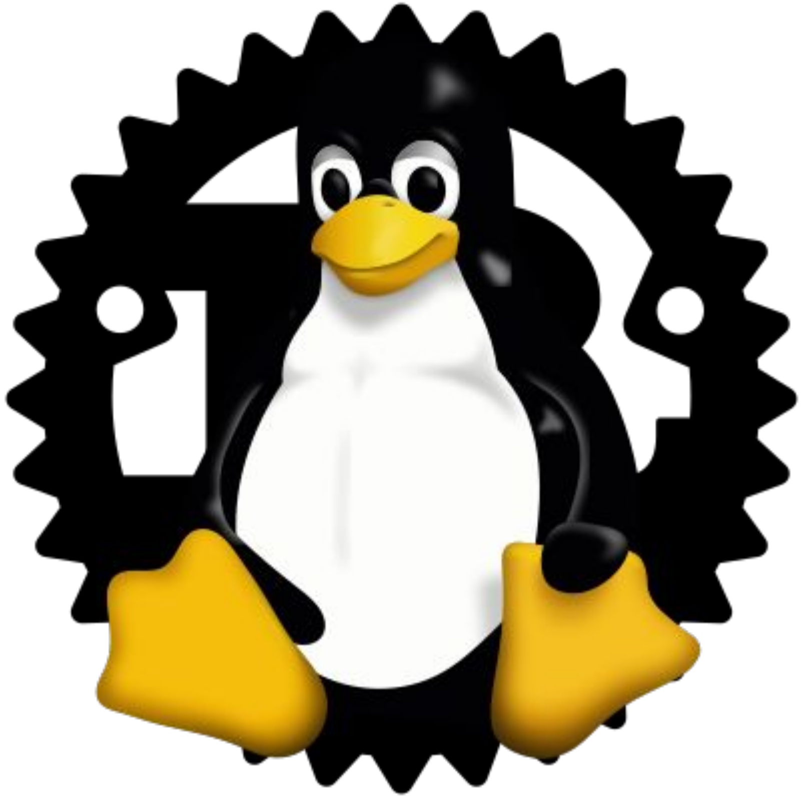


Rust for Linux
Overview and Status

Miguel Ojeda

ojeda@kernel.org

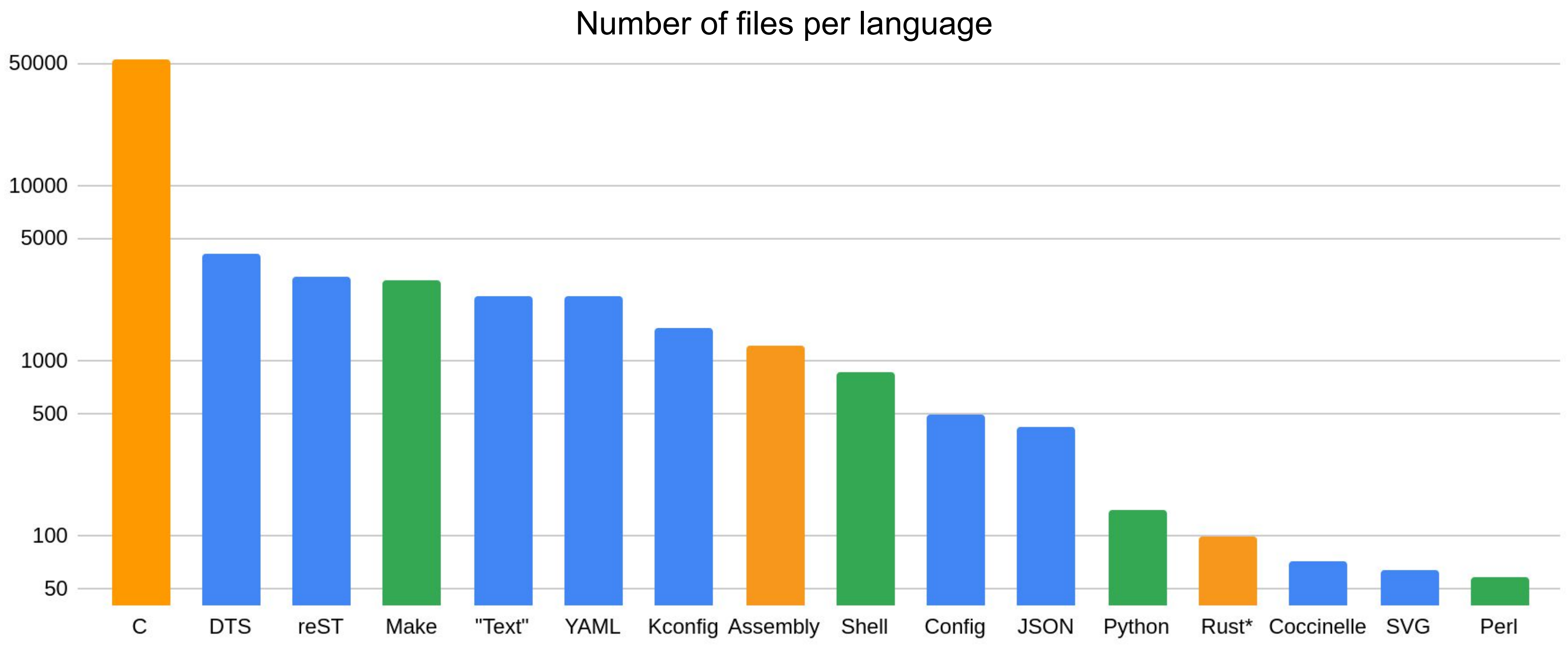
ojeda.dev

Rust for Linux

The project aims to bring Rust support to the Linux kernel as a first-class language.

This includes providing support for writing kernel modules in Rust, such as drivers or filesystems, with as little unsafe code as possible (potentially none).

Languages in the kernel



*if merged

Why Rust for the kernel?

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Why Rust for the kernel?

- Decreased chance of memory safety bugs.
    - In other words, bugs related to how memory is used:
        - Uses after free
        - Double frees
        - Out of bounds accesses
        - Uninitialized memory reads
        - Invalid inhabitants

Why Rust for the kernel?

- Decreased chance of memory safety bugs.
  - In other words, bugs related to how memory is used:
    - Uses after free
    - Double frees
    - Out of bounds accesses
    - Uninitialized memory reads
    - Invalid inhabitants
  - They are often security issues.

Why Rust for the kernel?

- Decreased chance of memory safety bugs.
  - In other words, bugs related to how memory is used:
    - Uses after free
    - Double frees
    - Out of bounds accesses
    - Uninitialized memory reads
    - Invalid inhabitants
  - They are often security issues.
  - It also prevents data races and other forms of Undefined Behavior.

What is the result of this program?

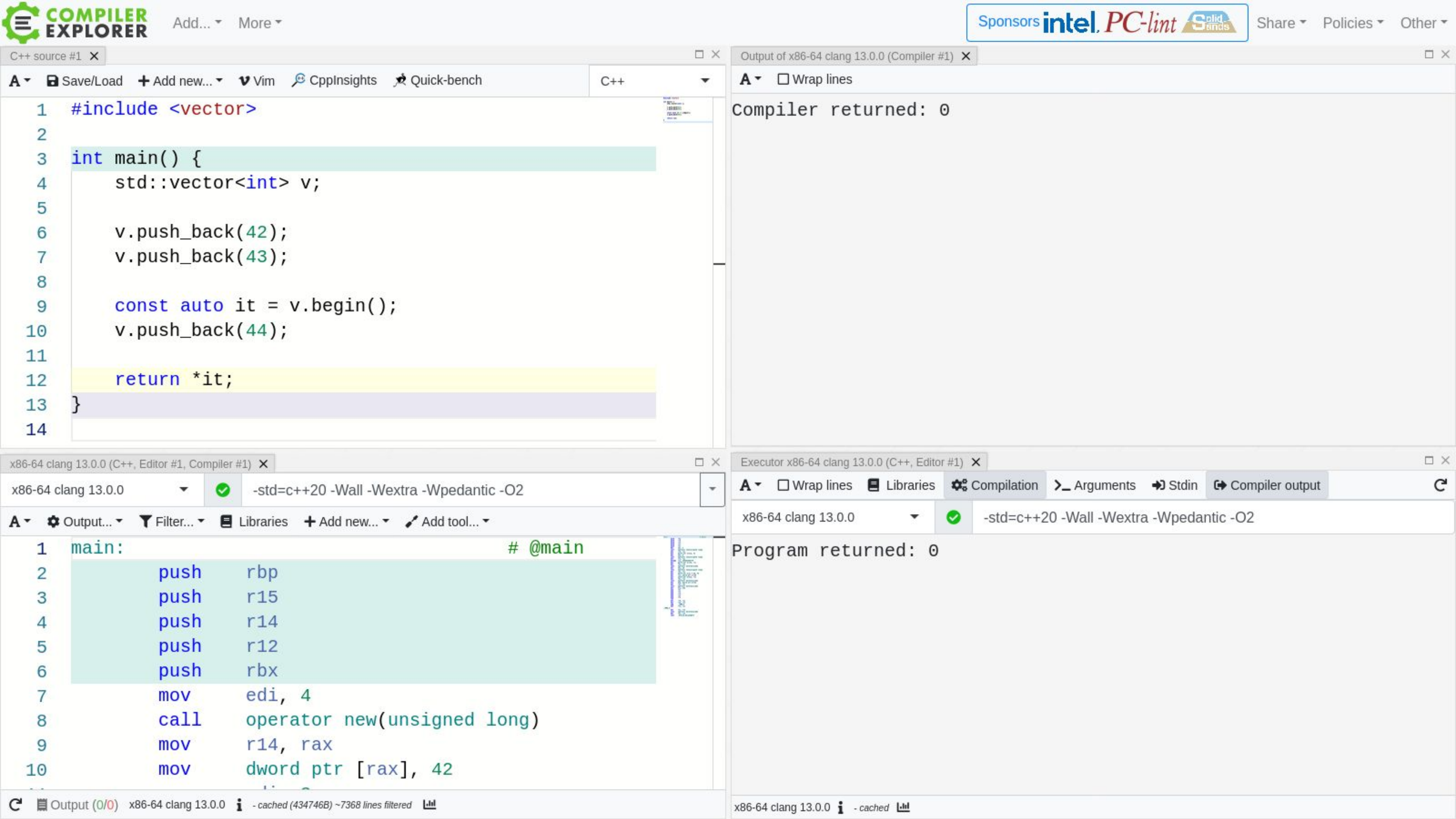
```
#include <vector>

int main() {
    std::vector<int> v;

    v.push_back(42);
    v.push_back(43);

    const auto it = v.cbegin();
    v.push_back(44);

    return *it;
}
```



```cpp
#include <vector>

int main() {
    std::vector<int> v;

    v.push_back(42);
    v.push_back(43);

    const auto it = v.begin();
    v.push_back(44);

    return *it;
}
```

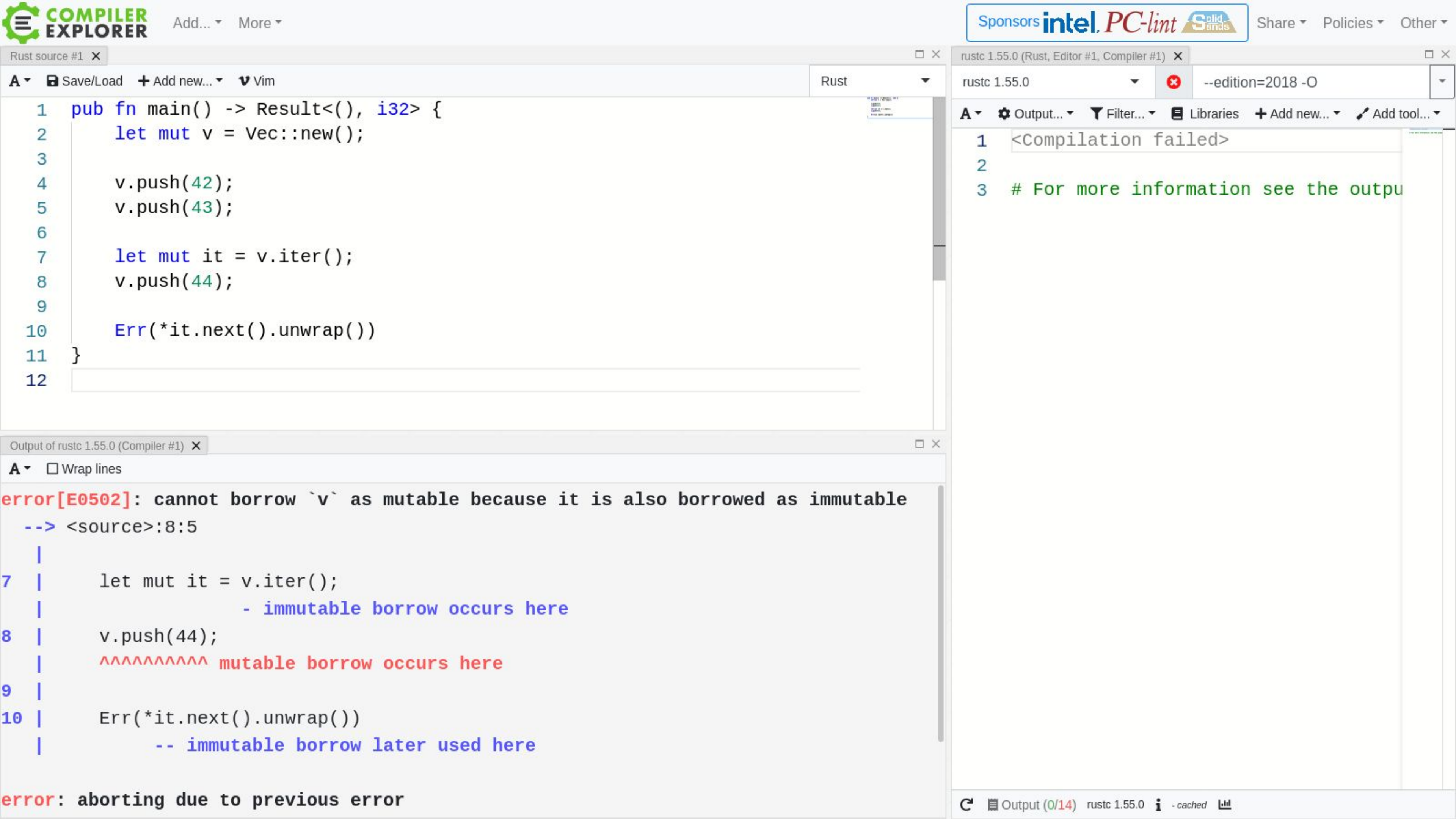
```
Compiler returned: 0
```

-std=c++20 -Wall -Wextra -Wpedantic -O2

```
main:                                   # @main
        push    rbp
        push    r15
        push    r14
        push    r12
        push    rbx
        mov     edi, 4
        call    operator new(unsigned long)
        mov     r14, rax
        mov     dword ptr [rax], 42
```

-std=c++20 -Wall -Wextra -Wpedantic -O2

```
Program returned: 0
```



```rust
pub fn main() -> Result<(), i32> {
    let mut v = Vec::new();

    v.push(42);
    v.push(43);

    let mut it = v.iter();
    v.push(44);

    Err(*it.next().unwrap())
}

```

```
<Compilation failed>

# For more information see the outpu
```

```
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
  --> <source>:8:5
   |
7  |     let mut it = v.iter();
   |                  - immutable borrow occurs here
8  |     v.push(44);
   |     ^^^^^^^^^^ mutable borrow occurs here
9  |
10 |     Err(*it.next().unwrap())
   |         -- immutable borrow later used here

error: aborting due to previous error
```

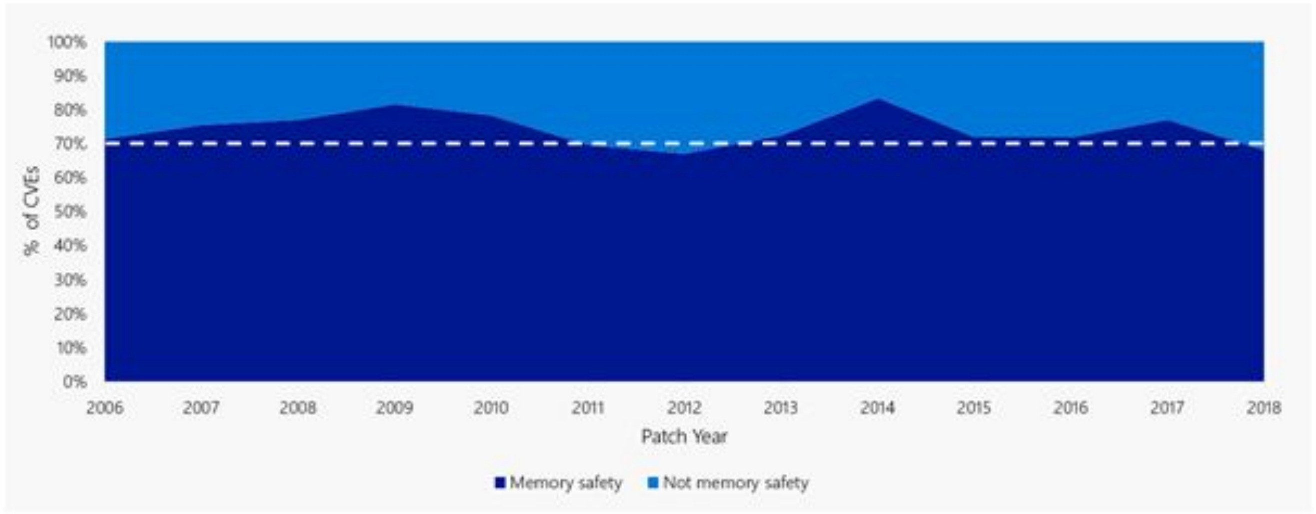
Is memory safety important?

~70%

of vulnerabilities in C/C++ projects come from UB

See more at https://www.memorysafety.org/docs/memory-safety/

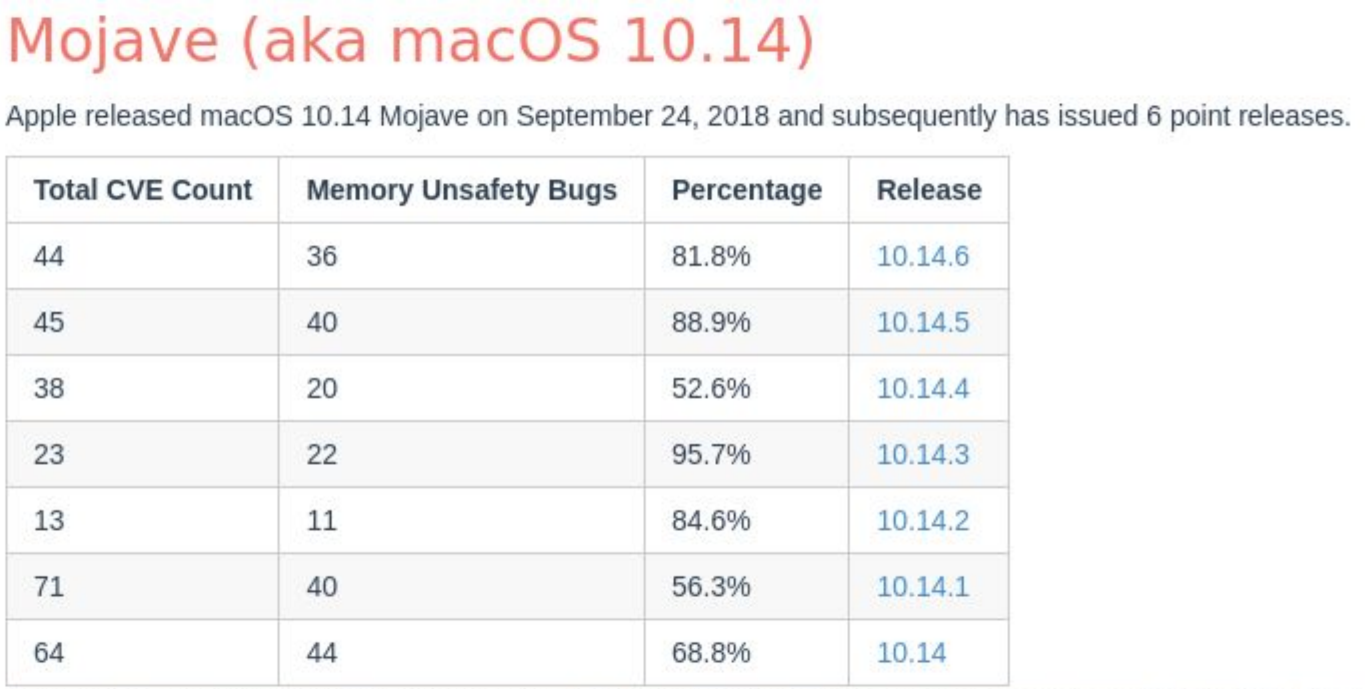
The cost of memory unsafety



~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

— https://msrc-blog.microsoft.com/2019/07/18/we-need-a-safer-systems-programming-language/

The cost of memory unsafety



Mojave (aka macOS 10.14)

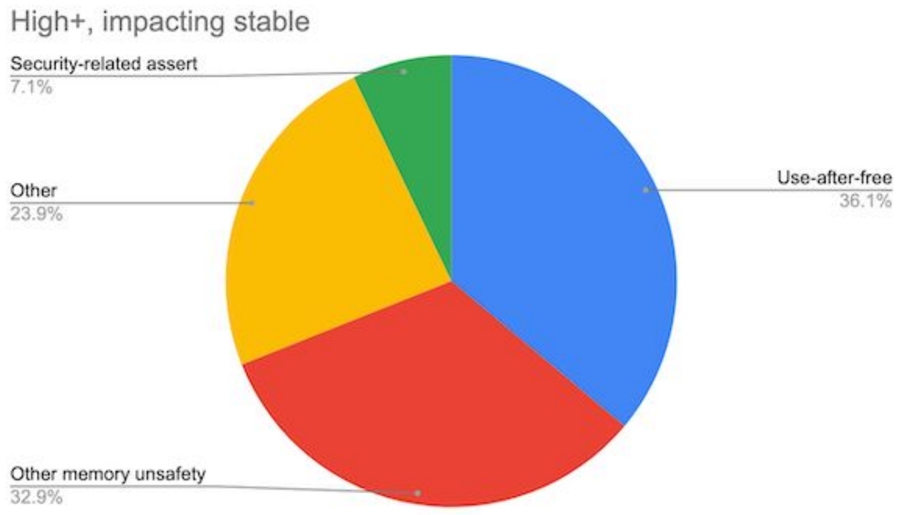
Apple released macOS 10.14 Mojave on September 24, 2018 and subsequently has issued 6 point releases.

Total CVE Count	Memory Unsafety Bugs	Percentage	Release
44	36	81.8%	10.14.6
45	40	88.9%	10.14.5
38	20	52.6%	10.14.4
23	22	95.7%	10.14.3
13	11	84.6%	10.14.2
71	40	56.3%	10.14.1
64	44	68.8%	10.14

— https://langui.sh/2019/07/23/apple-memory-safety/

The cost of memory unsafety

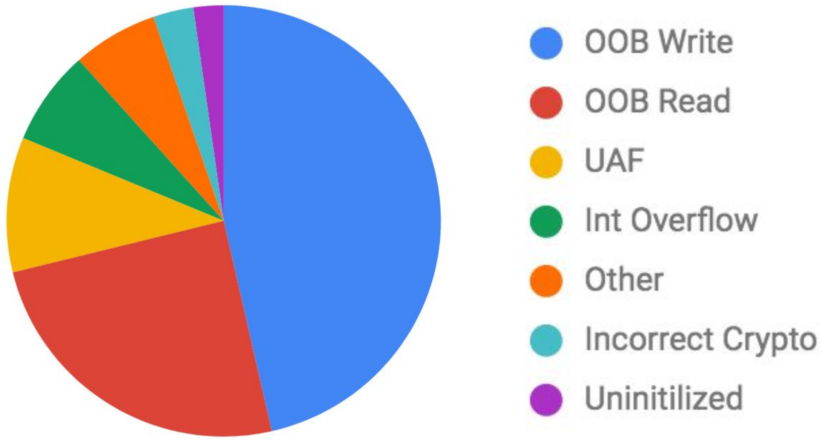
The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.



— https://www.chromium.org/Home/chromium-security/memory-safety

The cost of memory unsafety

Most of Android's vulnerabilities occur in the media and bluetooth components. Use-after-free (UAF), integer overflows, and out of bounds (OOB) reads/writes comprise 90% of vulnerabilities with OOB being the most common.



— https://security.googleblog.com/2019/05/queue-hardening-enhancements.html

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Help decrease logic bugs.

Why Rust for the kernel?

- Decreased chance of memory safety bugs.
- Help decrease logic bugs.
  - Stricter type system.
  - Language features that may improve the quality of kernel code.

Why Rust for the kernel?

- Decreased chance of memory safety bugs.
- Help decrease logic bugs.
  - Stricter type system.
  - Language features that may improve the quality of kernel code.
    - Pattern matching and sum types (variants, tagged unions)
    - Destructors and RAII (“Resource Acquisition Is Initialization”)
    - Traits
    - Generics
    - Asynchronous Rust (“async”)
    - ...

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Help decrease logic bugs.

Easier development and reviewing drivers.

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Help decrease logic bugs.

Easier development and reviewing drivers.

Less risky refactoring of drivers in the future.

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Help decrease logic bugs.

Easier development and reviewing drivers.

Less risky refactoring of drivers in the future.

A chance to raise the bar in other areas.

Why Rust for the kernel?

Decreased chance of memory safety bugs.

Help decrease logic bugs.

Easier development and reviewing drivers.

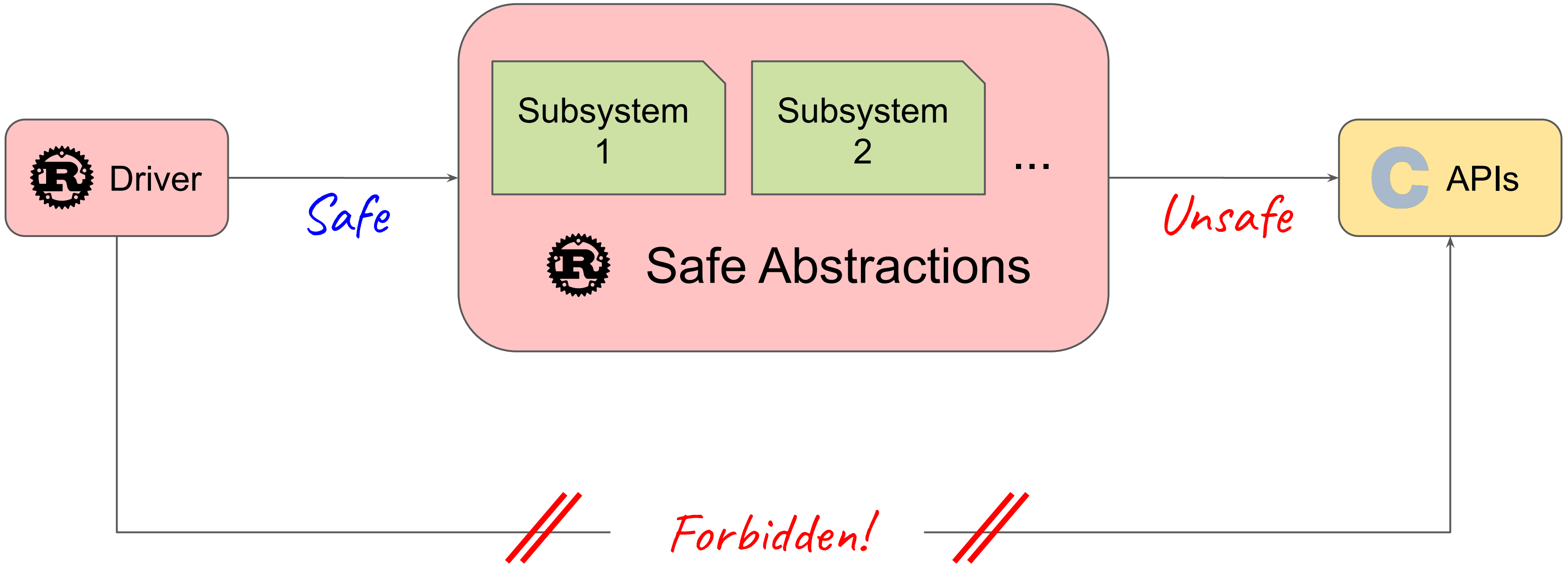
Less risky refactoring of drivers in the future.

A chance to raise the bar in other areas.

- Establish and enforce policies, e.g. “`SAFETY`” requirements, type invariants...
- Improved, closer-to-the-code documentation
- Automatically formatted code

How does Rust work in the kernel?

Encapsulating unsafety



Safe and unsafe code

Unsafe code: code inside an unsafe block.

It has access to all operations.

Safe code: code that is outside an unsafe block (i.e. the default).

It cannot perform a few operations (e.g. calling unsafe functions or dereferencing raw pointers).

Safe and unsafe APIs

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

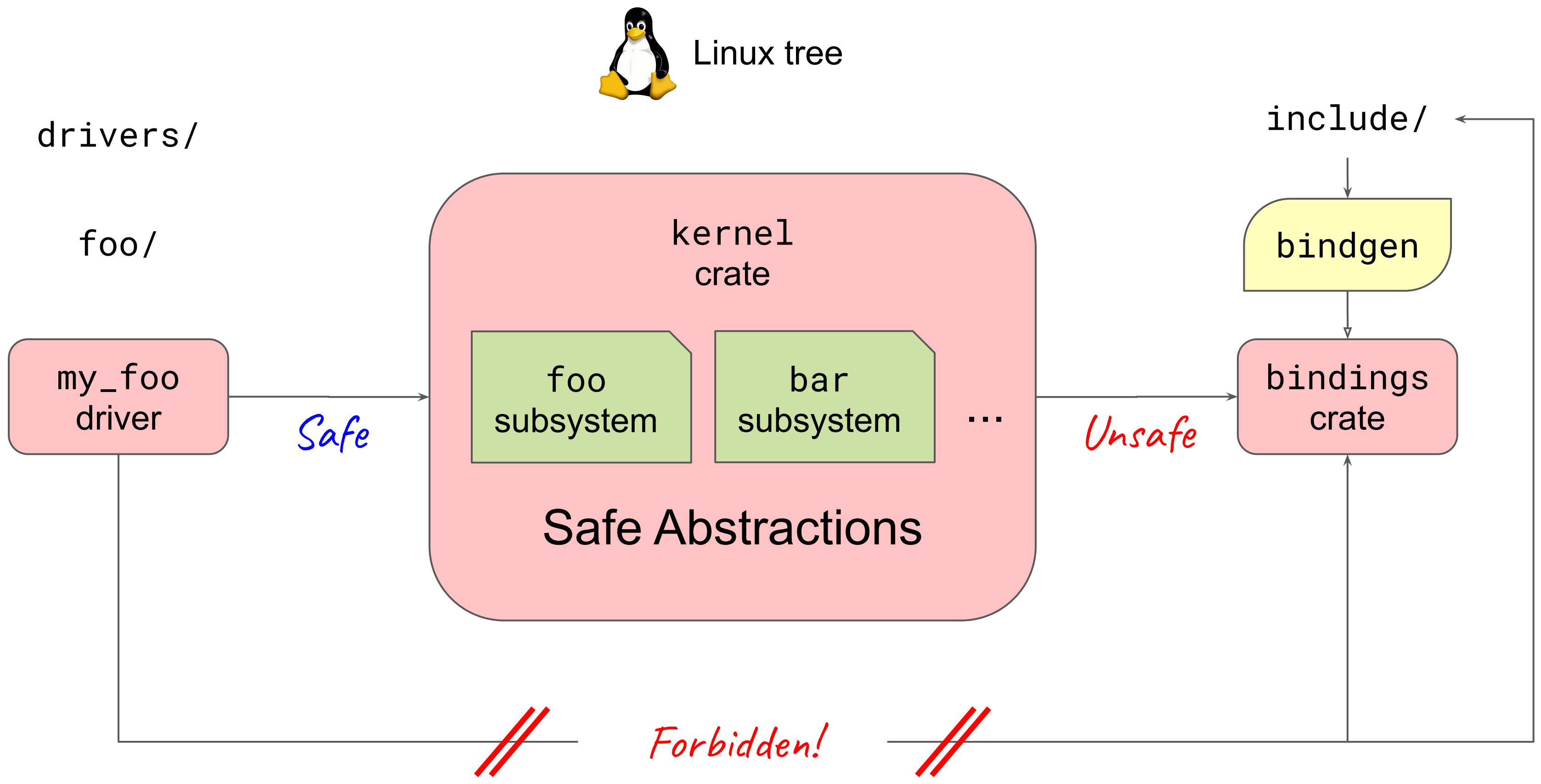
That is, it does not have any precondition (regarding undefined behavior).

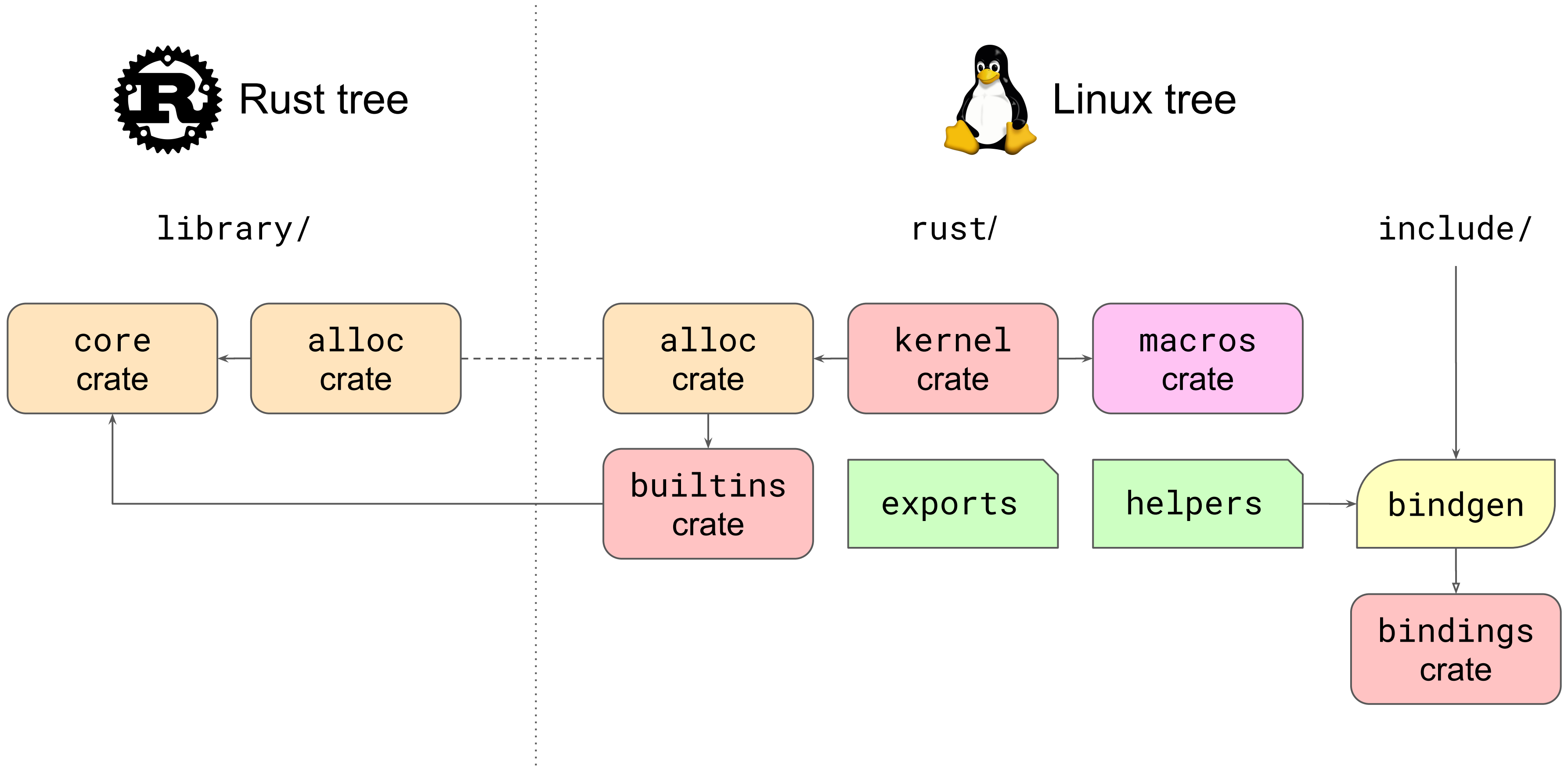
In other words, whatever a caller does, it does not produce undefined behavior.

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

This means it has safety preconditions.

Callers have to declare they are upholding the contract.


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

The last year

The last year – Infrastructure

Removed panicking allocations.

Moved to Edition 2021 of the Rust language.

Moved to stable releases of the Rust compiler.

- And started to track the latest version.

Added `arm` (32-bit) and `riscv` architecture support.

Testing support.

- Including running documentation tests inside the kernel as KUnit tests.

Support for “hostprogs” written in Rust.

On-the-fly generation of target specification files based on the kernel configuration.

The last year – Abstractions

PrimeCell PL061 GPIO example driver.

Functionality for platform and AMBA drivers, red-black trees, file descriptors, efficient bit iterators, tasks, files, IO vectors, power management callbacks, IO memory, IRQ chips, credentials, VMA, Hardware Random Number Generators, networking...

Synchronization features such as RW semaphores, revocable mutexes, raw spinlocks, a no-wait lock, sequence locks...

Replaced `Arc` and `Rc` from the `alloc` crate with a simplified kernel-based `Ref`.

Better panic diagnostics and simplified pointer wrappers.

The beginning of Rust `async` support.

The last year – Related projects

Rust stabilized a few unstable features we used.

Improvements on the Rust compiler, standard library and tooling.

> e.g. `rustc_parse_format` compile on stable, the addition of the `no_global_oom_handling` and `no_fp_fmt_parse` gates...

`binutils`/`gdb`/`libiberty` got support for Rust v0 demangling.

Intel's 0Day/LKP kernel test robot started testing a build with Rust support enabled.

Linaro's TuxSuite added Rust support.

`rustc_codegen_gcc` (the `rustc` backend for GCC) got merged into the Rust repository.

GCC Rust (a Rust frontend for GCC) gained a second full time developer.

Compiler Explorer added the alternative compilers for Rust, as well as other features such as MIR and macro expansion views.

The last year – Events

Linaro Virtual Connect Fall

Clang Built Linux Meetup

Linux Plumbers Conference (LPC)

Samsung Engineering Summit

Open Source Summit Japan

Rust Cross Team Collaboration Fun Times (CTCFT)

Rust Linz

Open Source Summit North America

Linux Foundation Live Mentorship Series

The last year – Industry support

“**Google** supports and contributes directly to the Rust for Linux project.

Our Android team is evaluating a new Binder implementation and considering other drivers where Rust could be adopted.”

— https://lore.kernel.org/lkml/20210704202756.29107-1-ojeda@kernel.org/

The last year – Industry support

"**Arm** recognises the Rust value proposition and is actively working with the Rust community to improve Rust for Arm based systems.

A good example is Arm's RFC contribution to the Rust language which made Linux on 64-bit Arm systems a Tier-1 Rust supported platform.

Rustaceans at Arm are excited about the Rust for Linux initiative and look forward to assisting in this effort."

— https://lore.kernel.org/lkml/20210704202756.29107-1-ojeda@kernel.org/

The last year – Industry support

“**Microsoft**'s Linux Systems Group is interested in contributing to getting Rust into Linux kernel.

Hopefully we will be able to submit select Hyper-V drivers written in Rust in the coming months.”

— https://lore.kernel.org/lkml/20210704202756.29107-1-ojeda@kernel.org/

The last year – Industry support

“There is interest in using Rust for kernel work that **Red Hat** is considering.”

— https://lore.kernel.org/lkml/20211206140313.5653-1-ojeda@kernel.org/

The last year – Academia

“Rust for Linux is a key step towards reducing security-critical kernel bugs, and on the path towards our ultimate goal of making Linux free of security-critical bugs. We are using Rust in our OS research, and adoption is easier with an existing Rust in the Linux kernel framework in place”

— Researchers at the University of Washington

They recently published “An Incremental Path Towards a Safer OS Kernel”
https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s09-li.pdf

The last year – Academia

“We used Rust for Linux because are convinced that Rust is changing the landscape of system programming by applying the research done on programming languages in the last decades. We wanted to see how the language was able to help us write code we are really comfortable with thanks to the extensive static checking.”

— Members of LSE (Systems Research Laboratory) at EPITA
(École pour l'informatique et les techniques avancées)

The last year – Prossimo project

PROSSIMO

Memory safety
for the Internet's most critical infrastructure

ISRG

https://www.abetterinternet.org/
https://www.memorysafety.org/blog/memory-safety-in-linux-kernel/

Today

Status

Experimental, but usable for:

- Working on new abstractions for subsystems.
- Writing drivers and other modules.
- Writing new subsystems from scratch.

Example modules: GPIO and Binder.

- Plus a small assortment of small samples.
- As well as a Rust out-of-tree template.

Supported architectures

`arm`	(`armv6` only)
`arm64`	
`powerpc`	(`ppc64le` only)
`riscv`	(`riscv64` only)
`x86`	(`x86_64` only)

The target specification file is generated from the kernel configuration.

GCC codegen paths should open up more.

See `Documentation/rust/arch-support.rst`

Rust unstable features

feature(allocator_api)
~~feature(alloc_error_handler)~~
feature(associated_type_defaults)
feature(bench_black_box)
feature(coerce_unsized)
feature(concat_idents)
~~feature(const_fn_trait_bound)~~
~~feature(const_fn_transmute)~~
feature(const_mut_refs)
~~feature(const_panic)~~
feature(const_ptr_offset_from)
~~feature(const_raw_ptr_deref)~~
feature(const_refs_to_cell)

feature(const_trait_impl)
~~feature(const_unreachable_unchecked)~~
~~feature(core_panic)~~
feature(dispatch_from_dyn)
feature(doc_cfg)
feature(generic_associated_types)
~~feature(global_asm)~~
~~feature(maybe_uninit_extra)~~
feature(more_fallible_allocation_methods)
feature(ptr_metadata)
feature(receiver_trait)
~~feature(try_reserve)~~
feature(unsize)

cfg(no_fp_fmt_parse)
cfg(no_global_oom_handling)
cfg(no_rc)
cfg(no_sync)

~~-Zallow-features~~
-Zbinary_dep_depinfo=y
-Zbuild-std
-Zcrate-attr
-Zunpretty=expanded
-Zfunction-sections
~~-Zsymbol-mangling-version=v0~~

https://github.com/Rust-for-Linux/linux/issues/2

Rust codegen paths for the kernel

`rustc_codegen_gcc`

Already passes most `rustc` tests and can bootstrap it

`rustc_codegen_llvm`

Main one

Rust GCC

Looking into compiling `core`, maybe getting merged next release

The next year

Next milestones

More users or use cases inside the kernel, including example drivers.

Splitting the `kernel` crate and managing dependencies to allow better development.

Extending the current integration of the kernel documentation, testing and other tools.

Getting more subsystem maintainers, companies and researchers involved.

Seeing most of the remaining Rust features stabilized.

Possibly start compiling the Rust code in the kernel with GCC.

And, of course, getting merged into the mainline kernel!

Upcoming events

Kangrejos, the Rust for Linux Workshop, face-to-face this time

https://kangrejos.com

Linux Plumbers Conference 2022

The Rust MC (microconference) will cover talks and discussions on both Rust for Linux as well as other non-kernel Rust topics.

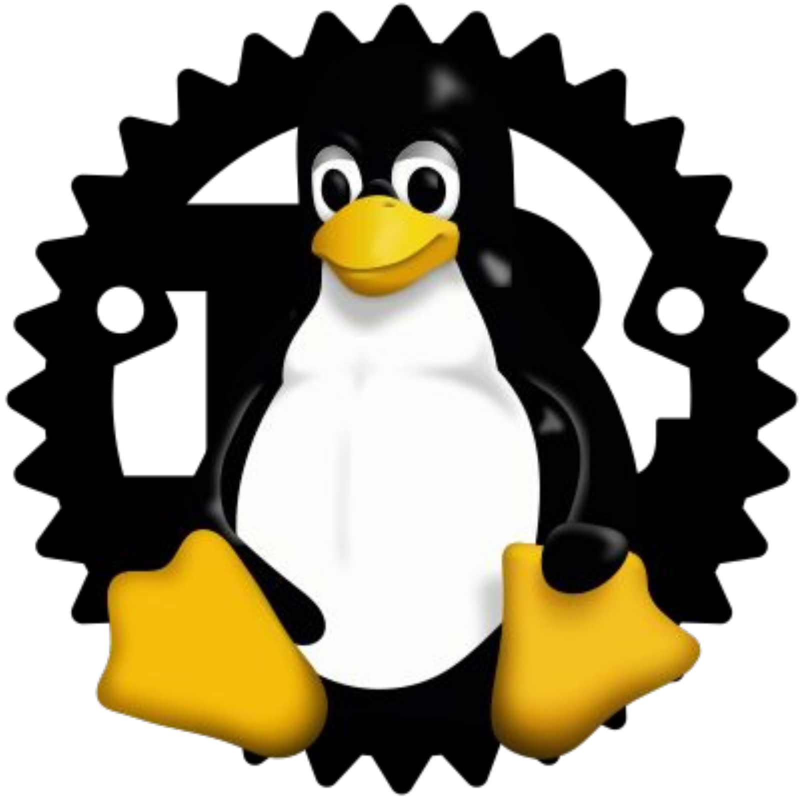
The Call for Proposals is open! https://lpc.events/event/16/contributions/1159/

Three more Linux Foundation Live Mentorship Series are coming

https://events.linuxfoundation.org/lf-live-mentorship-series/

Thank you!

Questions?



Rust for Linux
Overview and Status

Miguel Ojeda

ojeda@kernel.org

ojeda.dev

Backup slides

Safety

Safety in Rust

=

No Undefined Behavior

i.e. memory safe, data race free, etc.

Safety

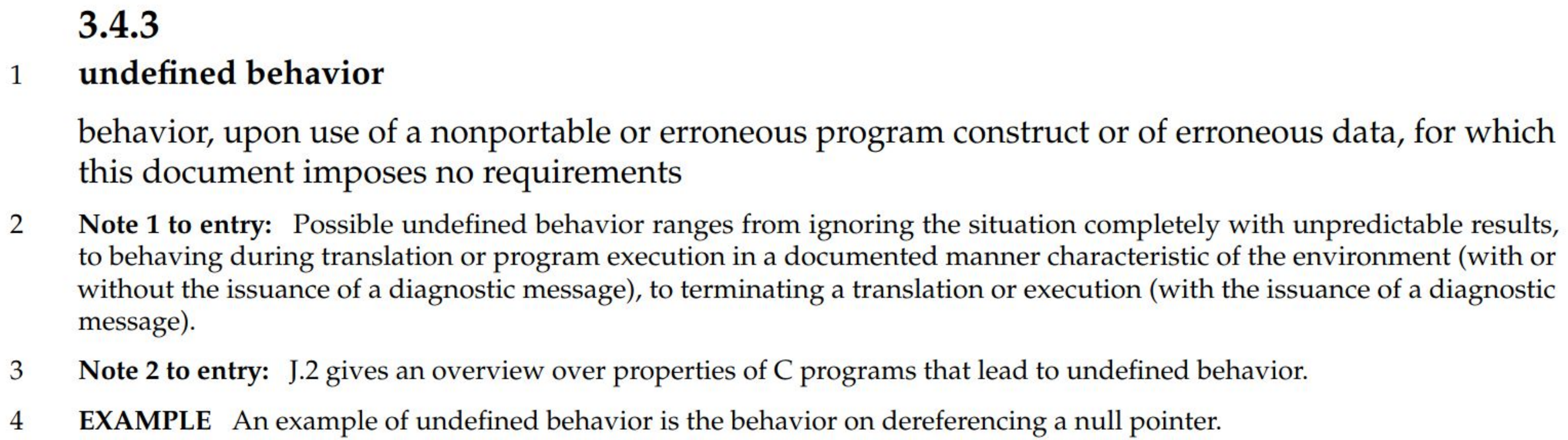
Safety in Rust

≠

Safety in safety-critical

as in functional safety (DO-178B/C, ISO 26262, EN 50128…)

What is Undefined Behavior?



3.4.3

1 **undefined behavior**

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this document imposes no requirements

2 **Note 1 to entry:** Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 **Note 2 to entry:** J.2 gives an overview over properties of C programs that lead to undefined behavior.

4 **EXAMPLE** An example of undefined behavior is the behavior on dereferencing a null pointer.

— N2596 C2x Working Draft

Can Undefined Behavior happen in this function?

```c
int f(int a, int b) {
    return a / b;
}
```

Unsafe function

```
int f(int a, int b) {
    return a / b;
}
```

UB $\forall x$ `f(x, 0);`

Unsafe function

```c
int f(int a, int b) {
    return a / b;
}
```

```
UB  ∀x f(x, 0);
UB     f(INT_MIN, -1);
```

Safe function

```
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

An example from the kernel

```rust
/// An owned string that is guaranteed to have exactly one `NUL` byte, which is at the end.
///
/// Used for interoperability with kernel APIs that take C strings.
///
/// # Invariants
///
/// The string is always `NUL`-terminated and contains no other `NUL` bytes.
pub struct CString {
    buf: Vec<u8>,
}

impl CString {
    /// Creates an instance of [`CString`] from the given formatted arguments.
    pub fn try_from_fmt(args: fmt::Arguments<'_>) -> Result<Self, Error> {
        // ...

        // INVARIANT: We wrote the `NUL` terminator and checked above that no
        // other `NUL` bytes exist in the buffer.
        Ok(Self { buf })
    }
}
```

An example from the kernel

```rust
impl Deref for CString {
    type Target = CStr;

    fn deref(&self) -> &Self::Target {
        // SAFETY: The type invariants guarantee that the string is
        // `NUL`-terminated and that no other `NUL` bytes exist.
        unsafe { CStr::from_bytes_with_nul_unchecked(self.buf.as_slice()) }
    }
}
```

Writing examples

```rust
/// A red-black tree with owned nodes.
///
/// It is backed by the kernel C red-black trees.
///
/// # Invariants
///
/// Non-null parent/children pointers stored in instances of the `rb_node`
/// C struct are always valid, and pointing to a field of our internal
/// representation of a node.
pub struct RBTree<K, V> {
    // ...
}
```

Writing examples

```
/// ...
///
/// # 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::prelude::*;
/// use kernel::rbtree::RBTree;
///
/// fn rbtest() -> Result {
///     // Create a new tree.
///     let mut tree = RBTree::new();
///
///     // Insert three elements.
///     tree.try_insert(20, 200)?;
///     tree.try_insert(10, 100)?;
///     tree.try_insert(30, 300)?;
///
/// ...
```