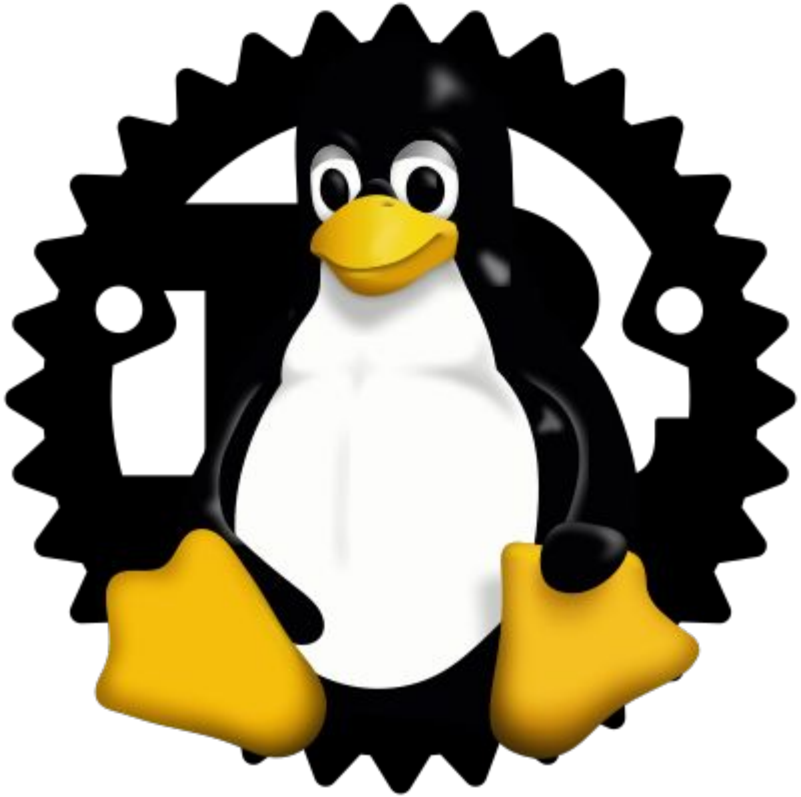


Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Credits & Acknowledgments

Rust

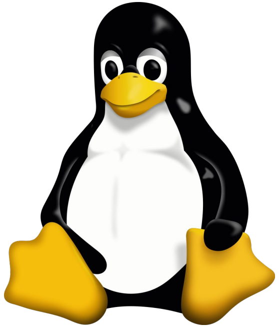
...for being a breath of fresh air

Kernel maintainers

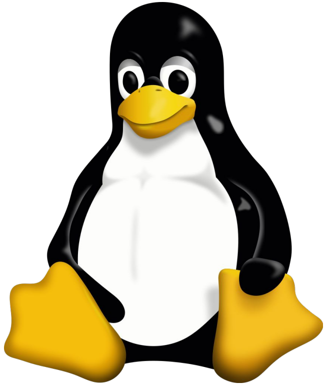
...for being open-minded

Everyone that has helped Rust for Linux

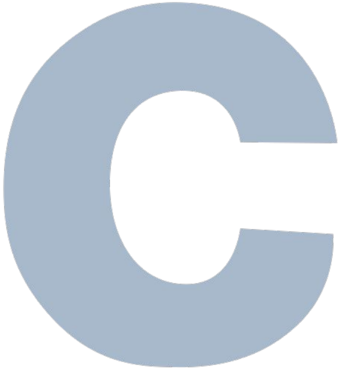
(see credits in the patch series)



History

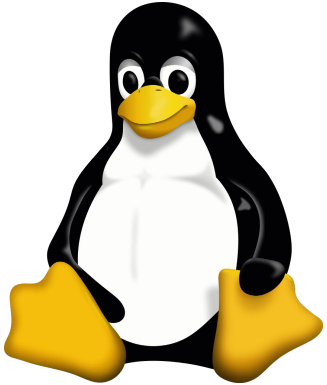


30 years of Linux

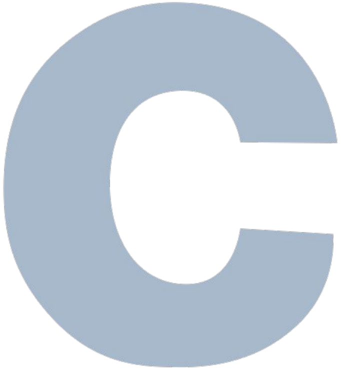


30 years of ISO C

Love story*



❤️*



30 years of Linux

30 years of ISO C

** Terms and Conditions Apply.*

An easy task

An easy task

> “Do you see any language except C which is suitable for development of operating systems?”

An easy task

> “Do you see any language except C which is suitable for development of operating systems?”

> “I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”
>
> — Linus Torvalds 2012

Why is C a good language for the kernel?

"You can use C to generate good code for hardware."

Fast

"When I read C, I know what the assembly language will look like."

Low-level

"The people that designed C ... designed it at a time when compilers had to be simple."

Simple

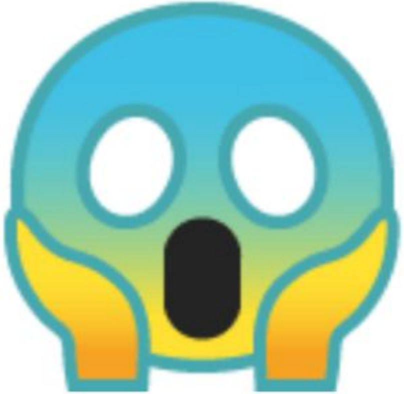
"If you think like a computer, writing C actually makes sense."

Fits the domain

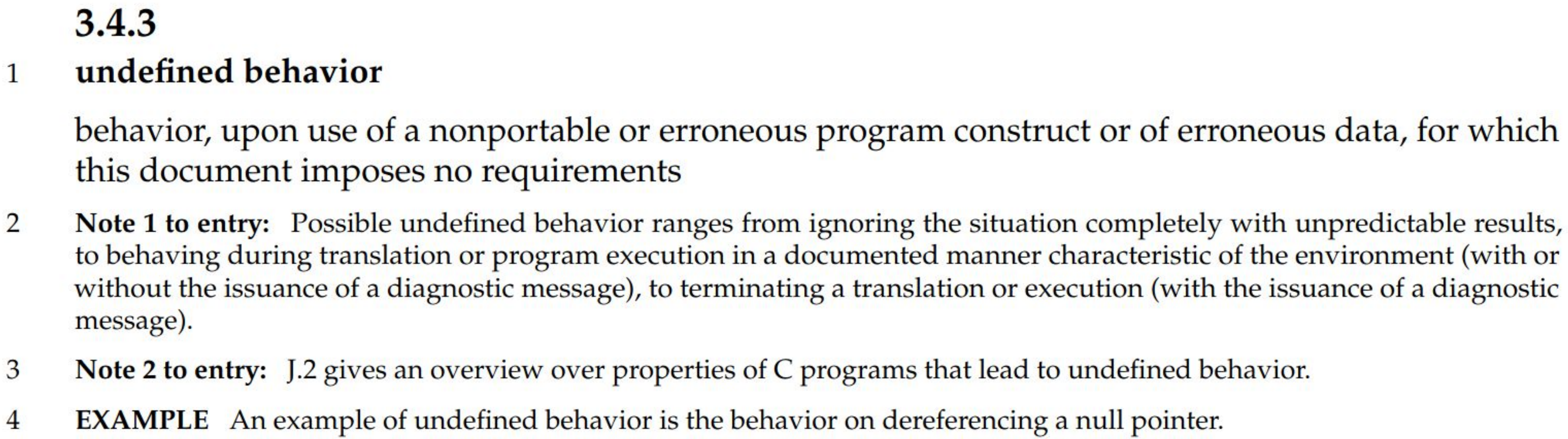
But...

But...

UB



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

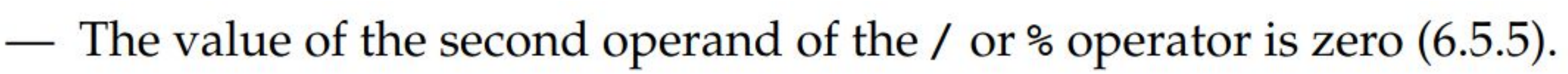
2 **Note 1 to entry:** Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating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

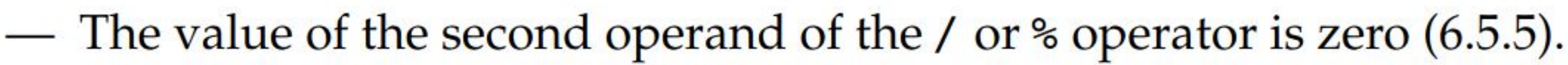
Examples of UB



— The value of the second operand of the / or % operator is zero (6.5.5).

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

Examples of UB



— The value of the second operand of the / or % operator is zero (6.5.5).

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

UB ∀x f(x, 0);

Examples of UB

Any other inputs that trigger UB?

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

Examples of UB

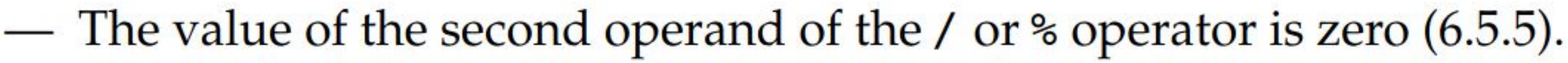
Any other inputs that trigger UB?

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

```
UB f(INT_MIN, -1);
```

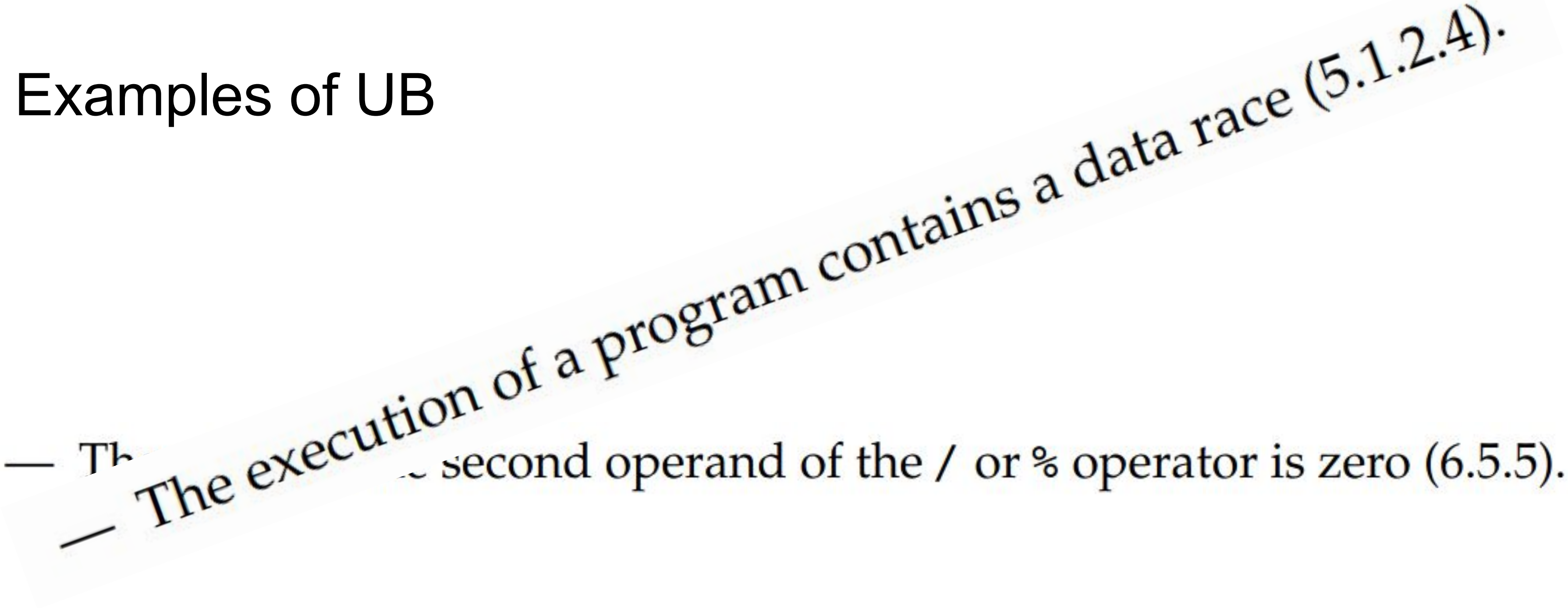
Examples of UB

Examples of UB



— The value of the second operand of the / or % operator is zero (6.5.5).

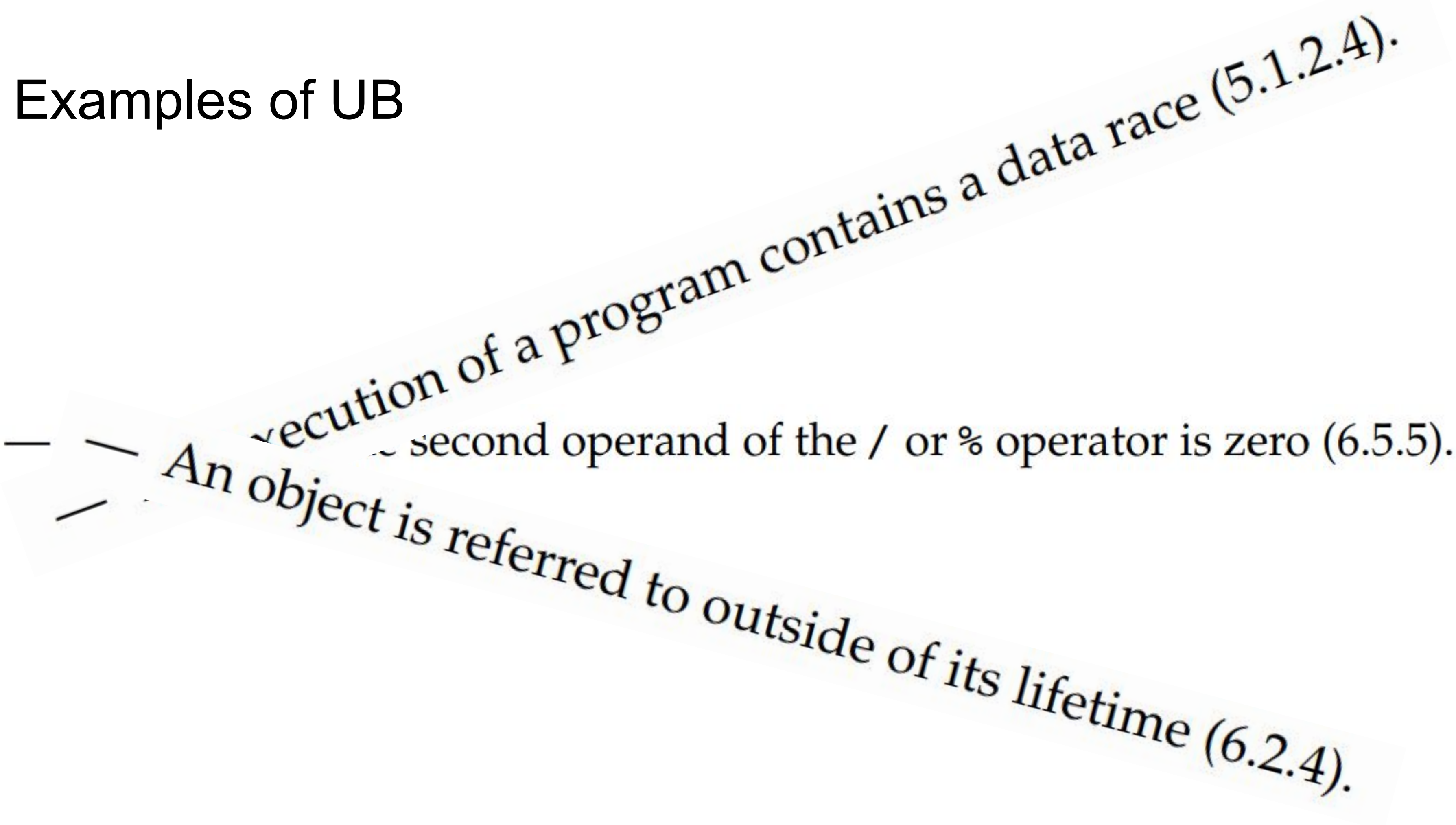
Examples of UB



— The execution of a program contains a data race (5.1.2.4).

— Th second operand of the / or % operator is zero (6.5.5).

Examples of UB

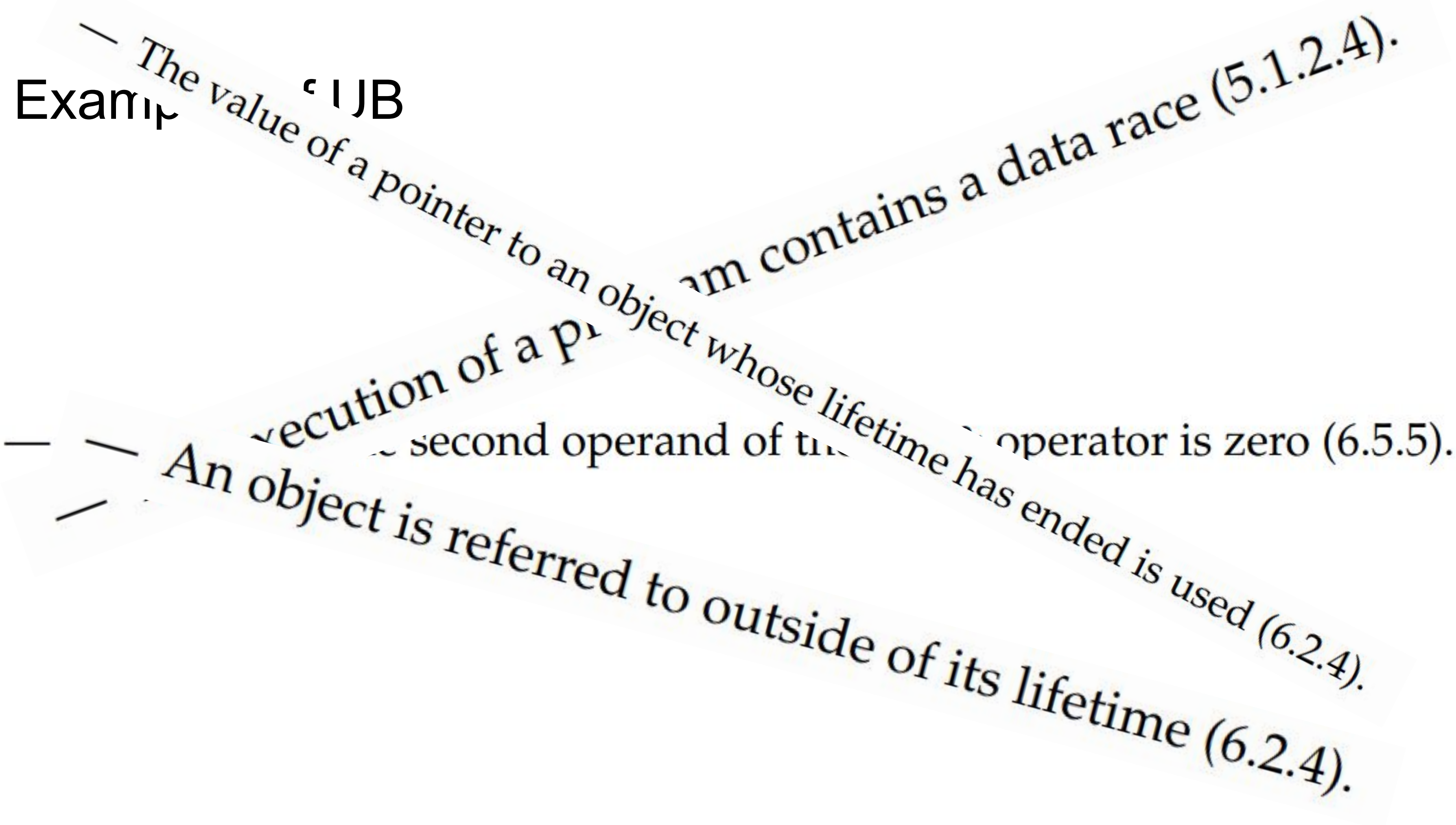


— ecution of a program contains a data race (5.1.2.4).

— second operand of the / or % operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

Examp UB



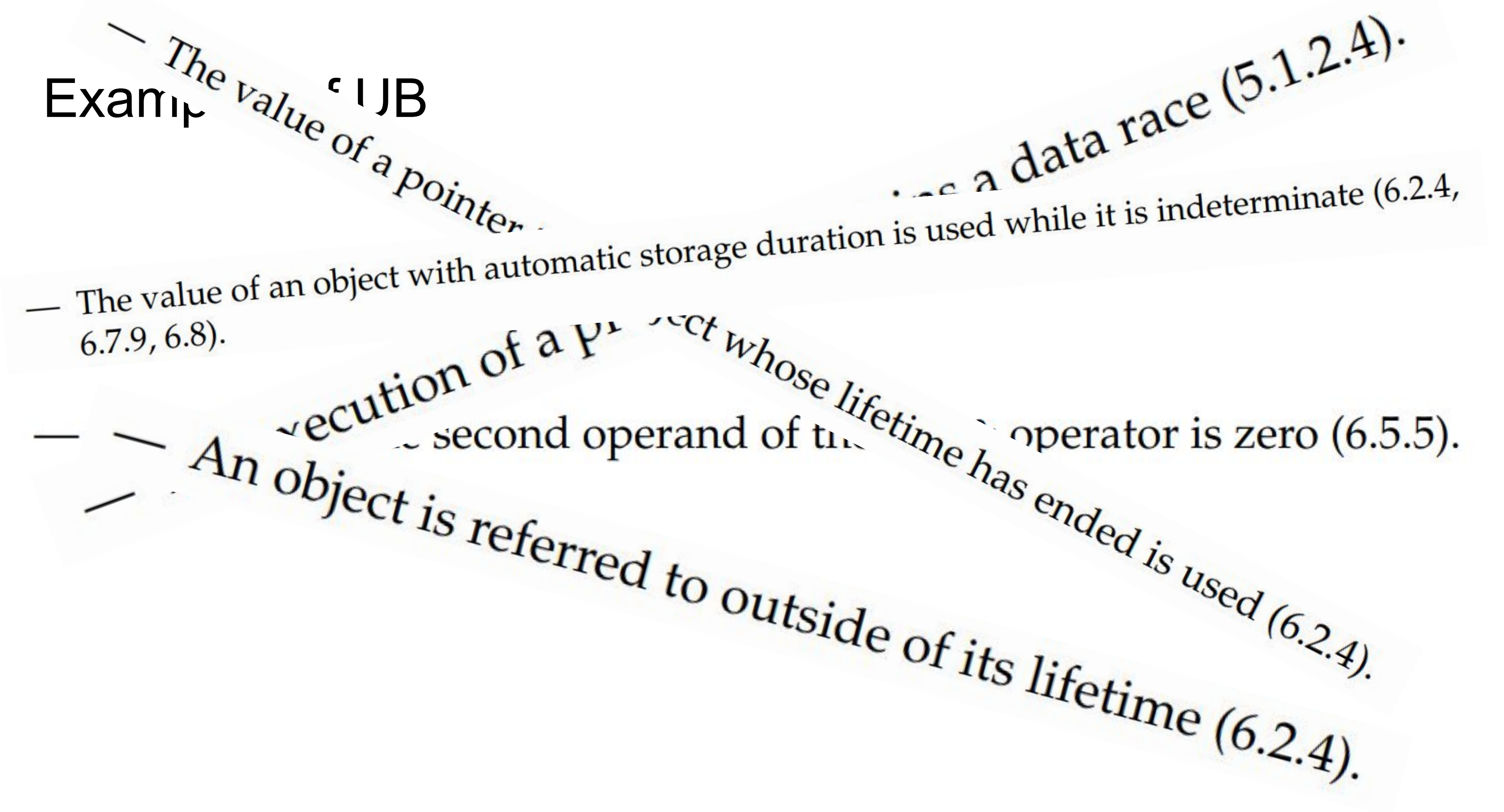
— The value of a pointer to an object whose lifetime has ended is used (6.2.4).

ecution of a pr am contains a data race (5.1.2.4).

— second operand of th operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

Examp UB



— The value of a pointer ect whose lifetime has ended is used (6.2.4).

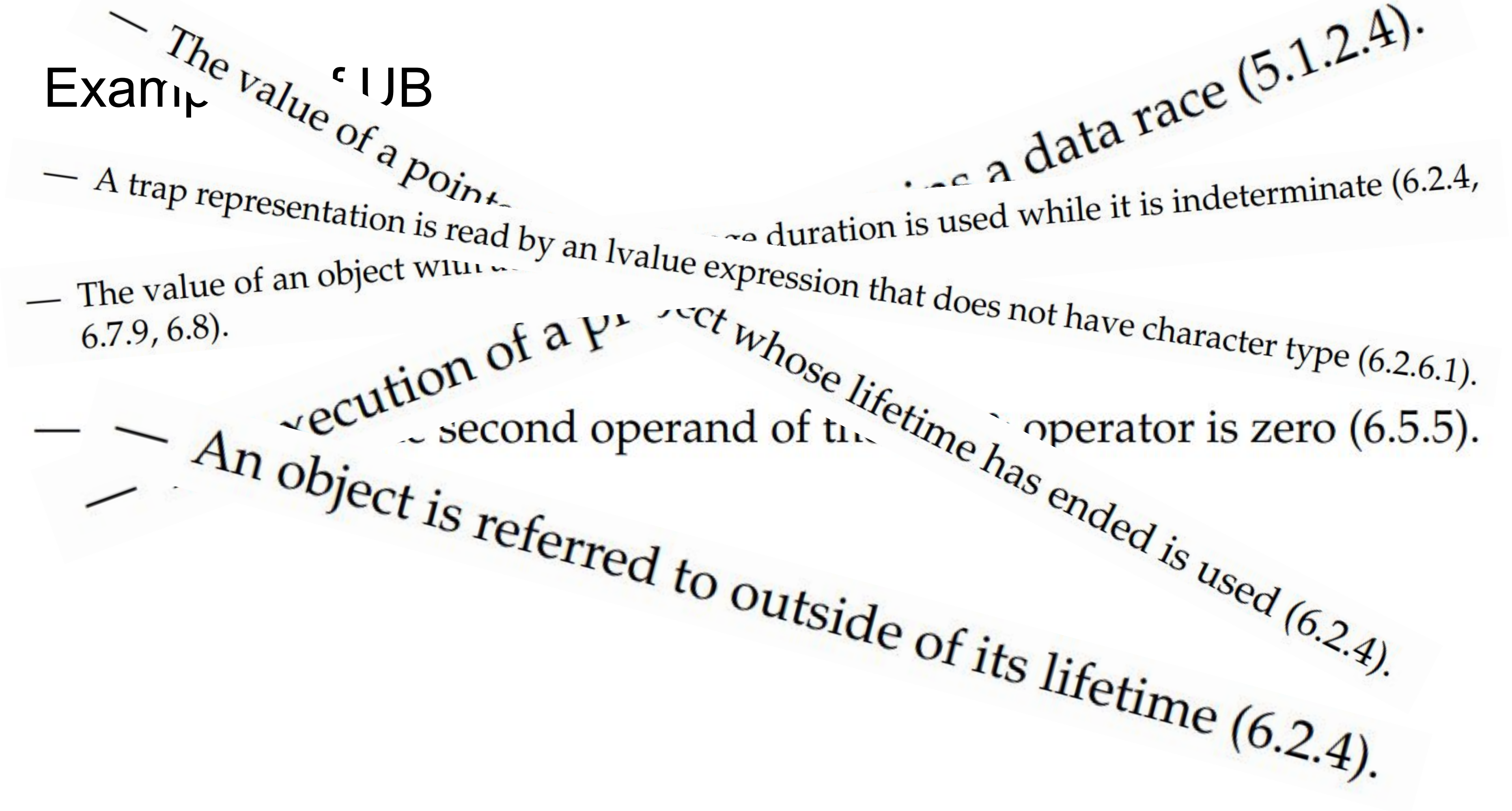
— ecution of a pr es a data race (5.1.2.4).

— The value of an object with automatic storage duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— second operand of th operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

Examp UB



— The value of a poin

a data race (5.1.2.4).

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object wi

e duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

ecution of a pr

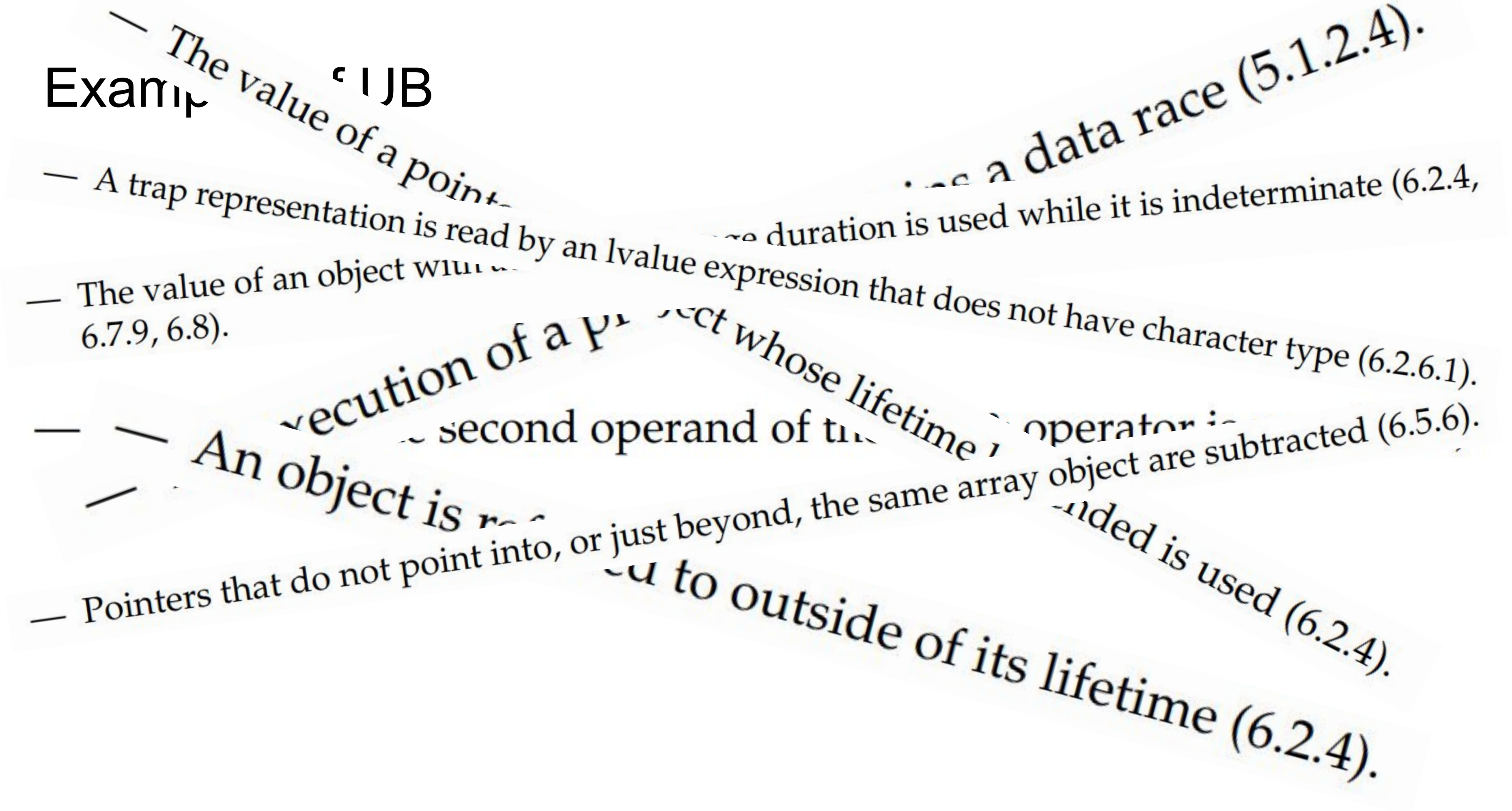
ect whose lifetime has ended is used (6.2.4).

— — second operand of t

operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

Examp UB



— The value of a point … ect whose lifetime … nded is used (6.2.4).

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object wit… …ge duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

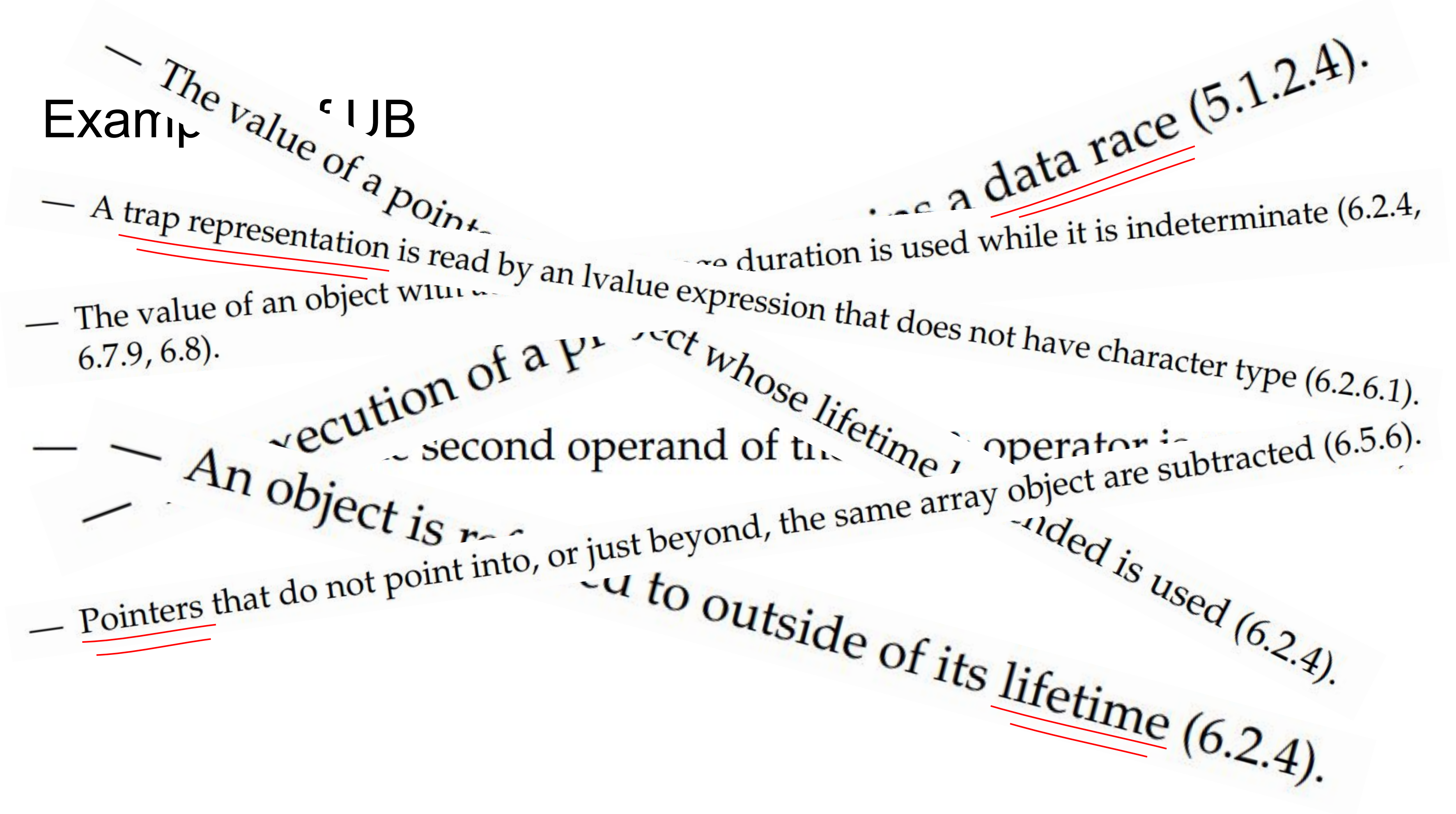
… ecution of a pr… …es a data race (5.1.2.4).

— … second operand of th… operator i…

— An object is r… …d to outside of its lifetime (6.2.4).

— Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.6).

Examp UB



— The value of a point

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object with duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— ecution of a p es a data race (5.1.2.4).

— second operand of th operator i

— An object is re d to outside of its lifetime (6.2.4).

ect whose lifetime nded is used (6.2.4).

— Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.6).

So, what does Rust offer?

So, what does Rust offer?

~~UB~~

Safety

Safety in Rust

=

No undefined behavior

similar to C (ISO/IEC 9899)

Safety

Safety in Rust

≠

Safety in “safety-critical”

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

✈ 🚕 🚆

Safety

⇒

`abort()`s in C

are

Rust-safe

Safety

⇒

`abort()`s in C

are

Rust-safe

Even if your company goes bankrupt.

Safety

⇒

`abort()`s in C

are

Rust-safe

Even if your company goes bankrupt.

Even if somebody is injured.

Avoiding UB

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

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

  return a / b;
}
```

Avoiding UB

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

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

  return a / b;
}
```

f is a safe function

Safety

⇒

Rust panics

are

Rust-safe

Safety

⇒

Kernel panics

are

Rust-safe

Safety

Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

⇒

are not

Rust-safe

Safety

Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

⇒ are not

Rust-safe

Even if your system still works.

What else does Rust offer?

Language

What else does Rust offer?

Shared & exclusive references

Modules & visibility

Generics

Lifetimes

Stricter type system

Language

Pattern matching

Safe/unsafe split

RAII

Sum types

Powerful hygienic and procedural macros

What else does Rust offer?

Freestanding standard library

What else does Rust offer?

Vocabulary types like `Result` and `Option`

Pinning

Formatting

Freestanding standard library

Checked, saturating & wrapping integer arithmetic primitives

Iterators

What else does Rust offer?

Tooling

What else does Rust offer?

Documentation generator

Unit & integration tests

Static analyzer

C ↔ Rust bindings generators

Linter

Tooling

Macro debugging

Formatter

IDE tooling

Great compiler error messages

UBSAN-like interpreter

What else does Rust offer?

Documentation generator

Unit & integration tests

Static analyzer

C ↔ Rust bindings generators

Linter

Tooling

Macro debugging

Formatter

IDE tooling

Great compiler error messages

UBSAN-like interpreter

plus the usual friends: `gdb`, `lldb`, `perf`, `valgrind`...

Where is the catch?

Where is the catch?

Cannot model everything ⇒ Unsafe code required

Where is the catch?

Cannot model everything ⇒ Unsafe code required

More information to provide ⇒ More complex language

Where is the catch?

Cannot model everything ⇒ Unsafe code required

More information to provide ⇒ More complex language

Extra runtime checks ⇒ Potentially expensive

Where is the catch?

Cannot model everything ⇒ Unsafe code required

More information to provide ⇒ More complex language

Extra runtime checks ⇒ Potentially expensive

An extra language to learn ⇒ Logistics & maintenance burden

Why is C a good language for the kernel?

"You can use C to generate good code for hardware."

Fast

"When I read C, I know what the assembly language will look like."

Low-level

"The people that designed C ... designed it at a time when compilers had to be simple."

Simple

"If you think like a computer, writing C actually makes sense."

Fits the domain

Rust

Why is C a good language for the kernel?

"You can use C to generate good code for hardware."

"When I read C, I know what the assembly language will look like."

"The people that designed C ... designed it at a time when compilers had to be simple."

"If you think like a computer, writing C actually makes sense."

Fast
Yes

Low-level
Sometimes

Simple
Not really

Fits the domain
...

An easy task

> “Do you see any language except C which is suitable for development of operating systems?”

> “I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”
>
> — Linus Torvalds 2012

An easy task maybe?

“Do you see any language except C which is suitable for development of operating systems?”

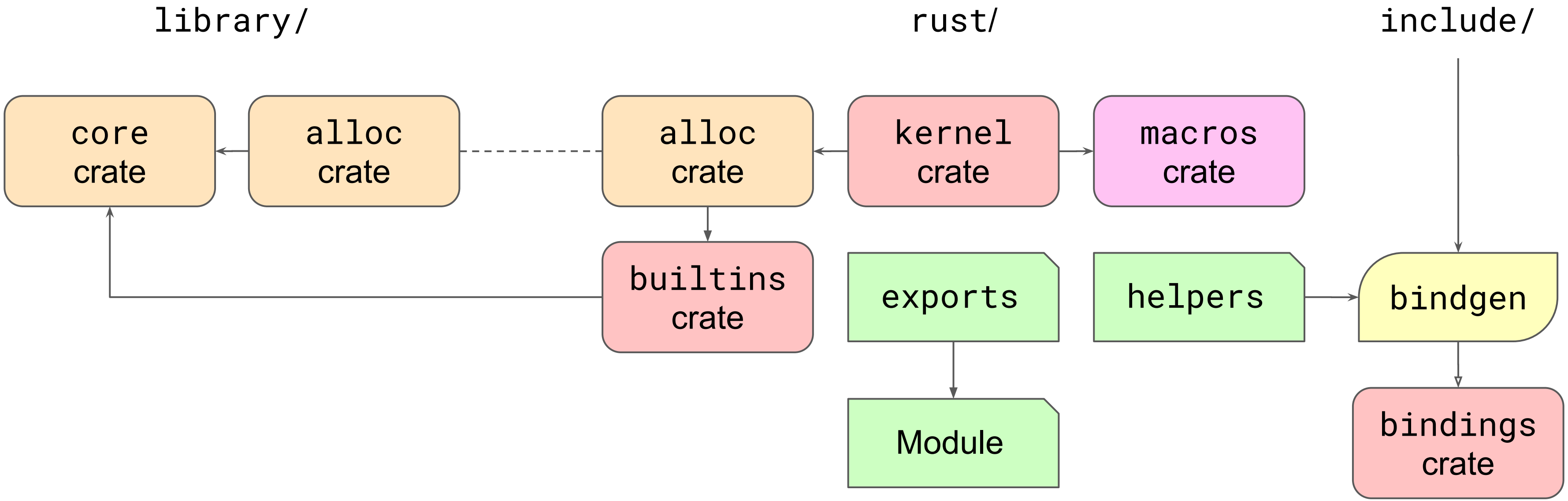
“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012

Rust support in the kernel

Rust tree

Linux tree



library/

rust/

include/

core
crate

alloc
crate

alloc
crate

kernel
crate

macros
crate

builtins
crate

exports

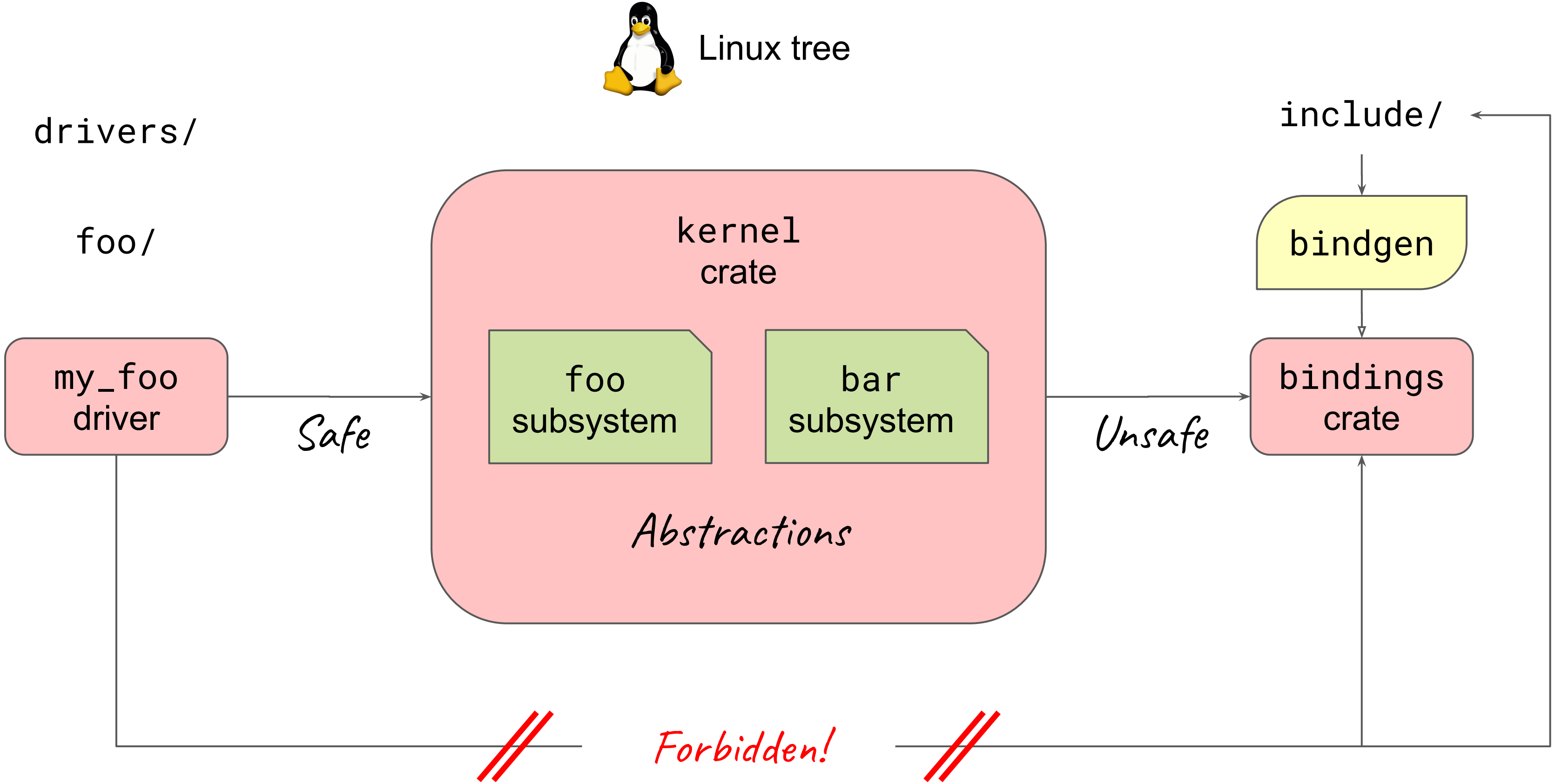
helpers

bindgen

Module

bindings
crate

Driver point of view


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

Supported architectures

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

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

Supported architectures

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

...so far!

32-bit and other restrictions should be easy to remove

Kernel LLVM builds work for `mips` and `s390`

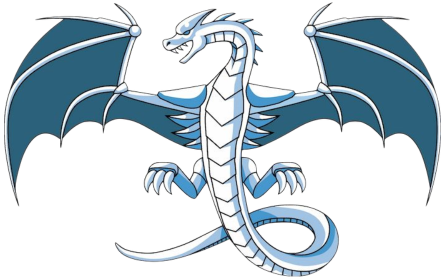
GCC codegen paths should open up more

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

Rust codegen paths for the kernel

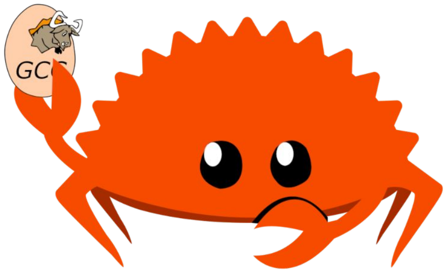
`rustc_codegen_gcc`

Already passes most rustc tests



`rustc_codegen_llvm`

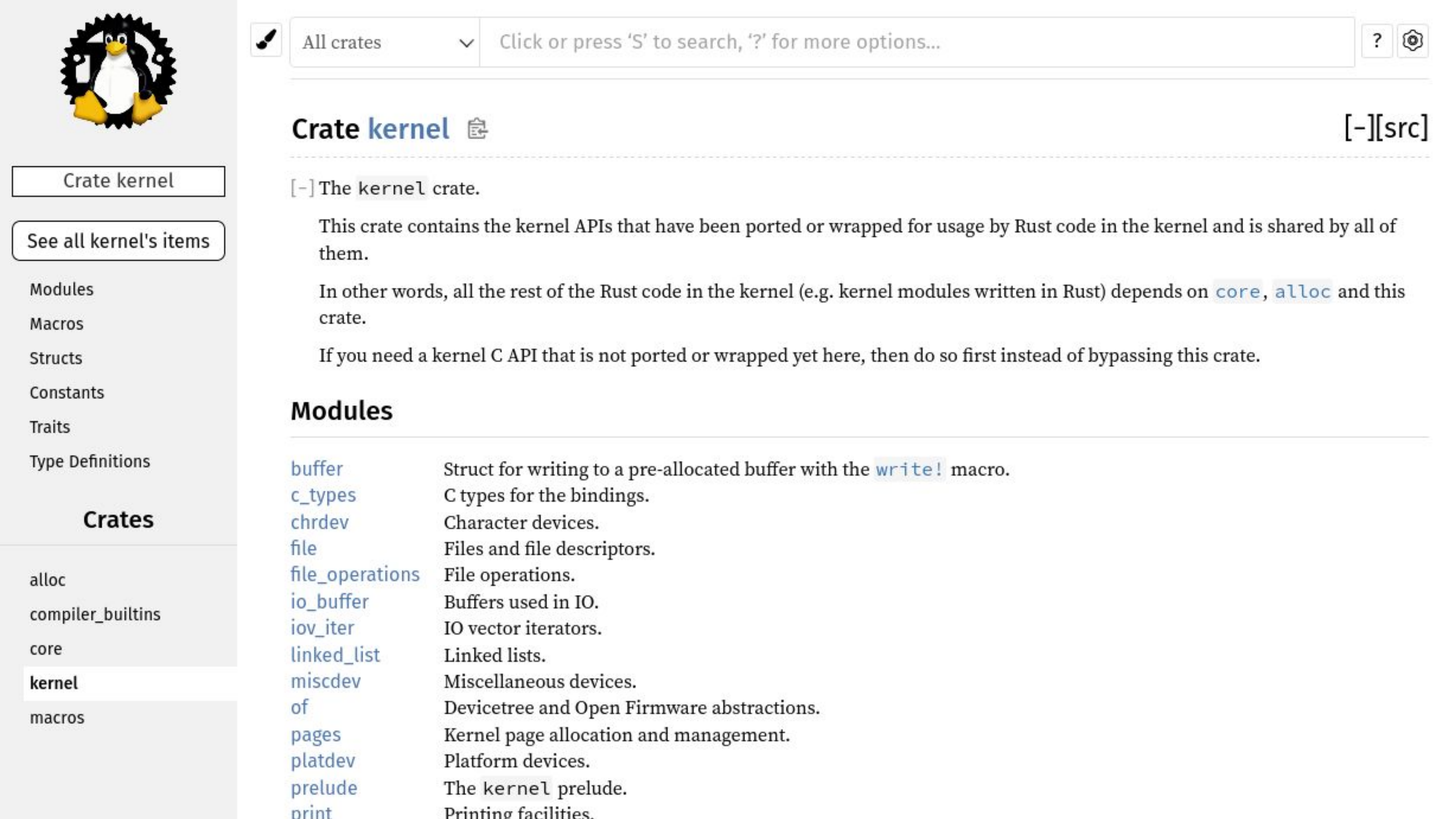
Main one



Rust GCC

Expected in 1-2 years (rough estimate)

Documentation



Crate kernel

[-] The `kernel` crate.

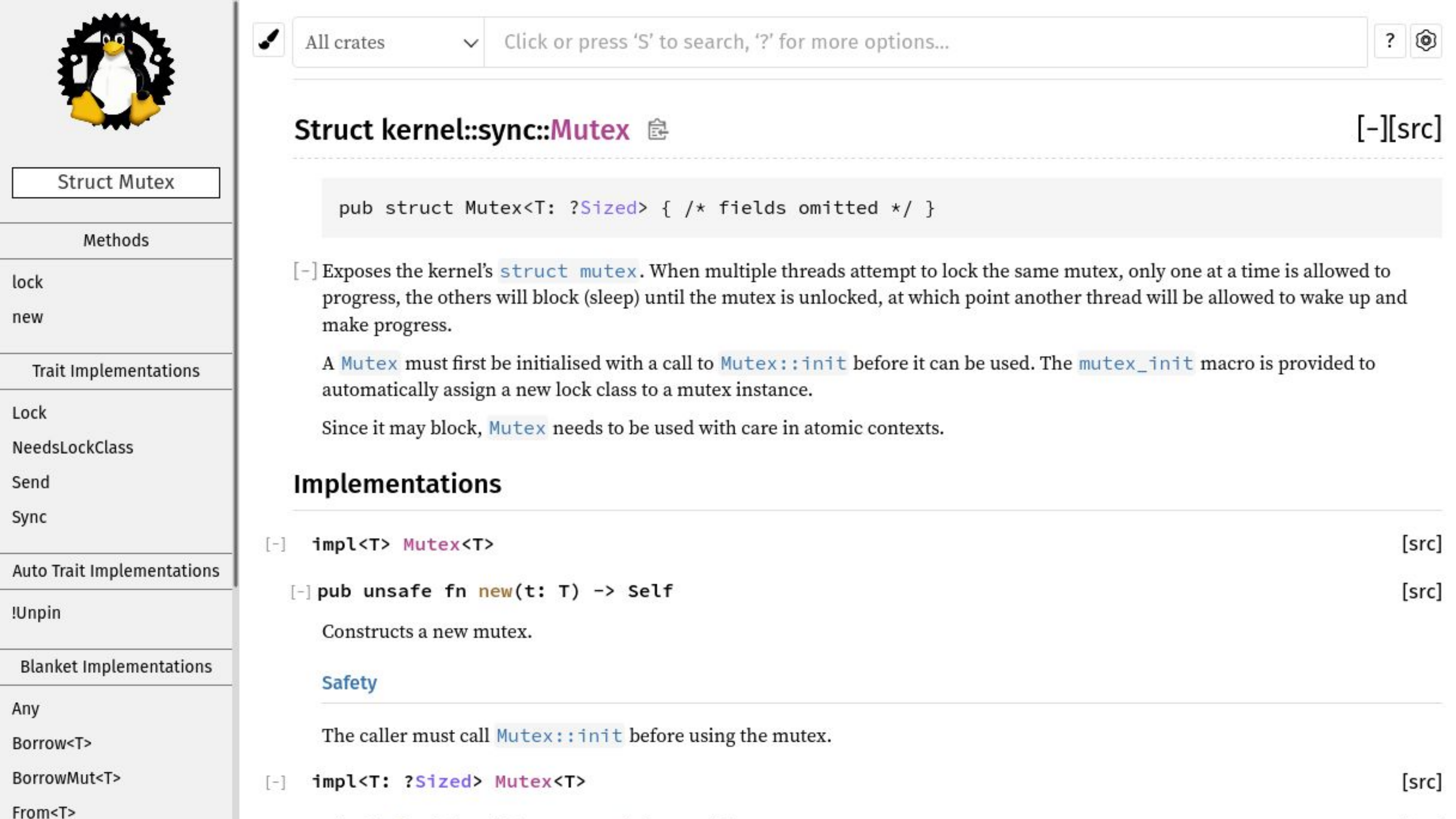
This crate contains the kernel APIs that have been ported or wrapped for usage by Rust code in the kernel and is shared by all of them.

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.

Modules

buffer	Struct for writing to a pre-allocated buffer with the `write!` macro.
c_types	C types for the bindings.
chrdev	Character devices.
file	Files and file descriptors.
file_operations	File operations.
io_buffer	Buffers used in IO.
iov_iter	IO vector iterators.
linked_list	Linked lists.
miscdev	Miscellaneous devices.
of	Devicetree and Open Firmware abstractions.
pages	Kernel page allocation and management.
platdev	Platform devices.
prelude	The `kernel` prelude.
print	Printing facilities.



Struct kernel::sync::Mutex

```
pub struct Mutex<T: ?Sized> { /* fields omitted */ }
```

[−] Exposes the kernel’s `struct mutex`. When multiple threads attempt to lock the same mutex, only one at a time is allowed to progress, the others will block (sleep) until the mutex is unlocked, at which point another thread will be allowed to wake up and make progress.

A `Mutex` must first be initialised with a call to `Mutex::init` before it can be used. The `mutex_init` macro is provided to automatically assign a new lock class to a mutex instance.

Since it may block, `Mutex` needs to be used with care in atomic contexts.

Implementations

```
impl<T> Mutex<T>
```

[−] [src]

```
pub unsafe fn new(t: T) -> Self
```

[src]

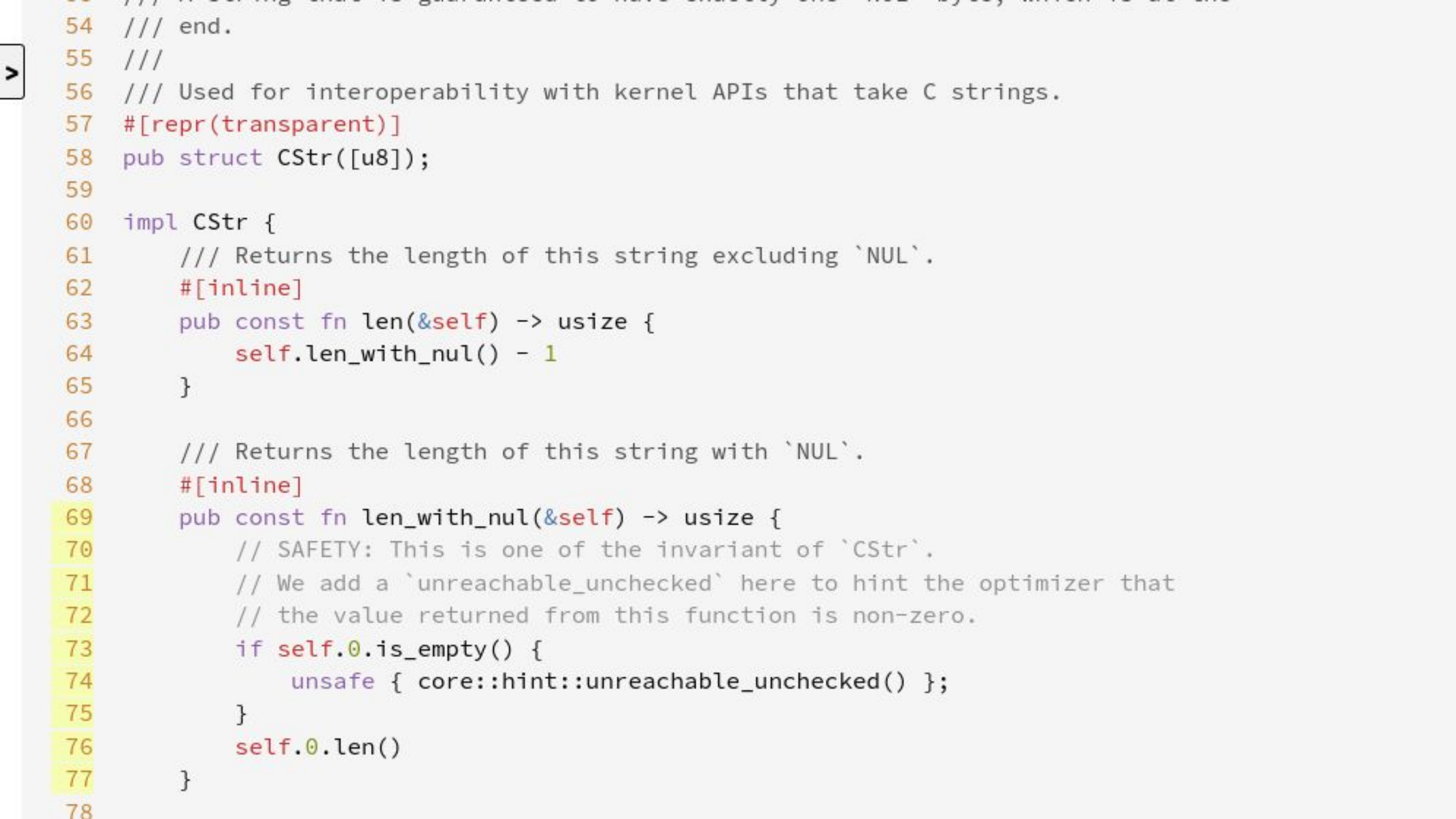
Constructs a new mutex.

Safety

The caller must call `Mutex::init` before using the mutex.

```
impl<T: ?Sized> Mutex<T>
```

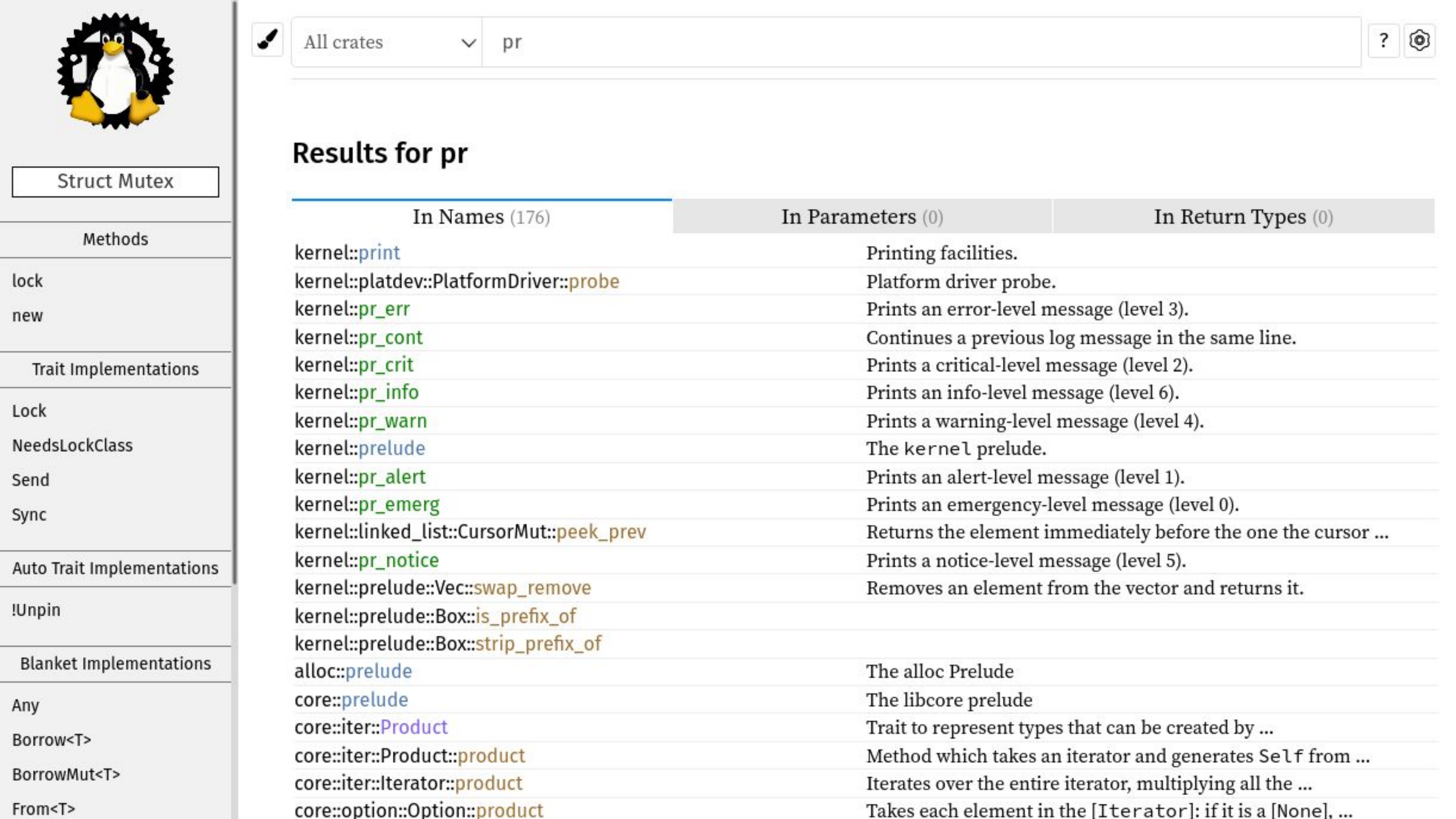
[−] [src]



```rust
/// A 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.
#[repr(transparent)]
pub struct CStr([u8]);

impl CStr {
    /// Returns the length of this string excluding `NUL`.
    #[inline]
    pub const fn len(&self) -> usize {
        self.len_with_nul() - 1
    }

    /// Returns the length of this string with `NUL`.
    #[inline]
    pub const fn len_with_nul(&self) -> usize {
        // SAFETY: This is one of the invariant of `CStr`.
        // We add a `unreachable_unchecked` here to hint the optimizer that
        // the value returned from this function is non-zero.
        if self.0.is_empty() {
            unsafe { core::hint::unreachable_unchecked() };
        }
        self.0.len()
    }
```



Results for pr

In Names (176) | In Parameters (0) | In Return Types (0)

kernel::print	Printing facilities.
kernel::platdev::PlatformDriver::probe	Platform driver probe.
kernel::pr_err	Prints an error-level message (level 3).
kernel::pr_cont	Continues a previous log message in the same line.
kernel::pr_crit	Prints a critical-level message (level 2).
kernel::pr_info	Prints an info-level message (level 6).
kernel::pr_warn	Prints a warning-level message (level 4).
kernel::prelude	The `kernel` prelude.
kernel::pr_alert	Prints an alert-level message (level 1).
kernel::pr_emerg	Prints an emergency-level message (level 0).
kernel::linked_list::CursorMut::peek_prev	Returns the element immediately before the one the cursor ...
kernel::pr_notice	Prints a notice-level message (level 5).
kernel::prelude::Vec::swap_remove	Removes an element from the vector and returns it.
kernel::prelude::Box::is_prefix_of	
kernel::prelude::Box::strip_prefix_of	
alloc::prelude	The alloc Prelude
core::prelude	The libcore prelude
core::iter::Product	Trait to represent types that can be created by ...
core::iter::Product::product	Method which takes an iterator and generates `Self` from ...
core::iter::Iterator::product	Iterates over the entire iterator, multiplying all the ...
core::option::Option::product	Takes each element in the [`Iterator`]: if it is a [None], ...

Documentation code

```rust
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # }
/// ```
pub struct Task {
   pub(crate) ptr: *mut bindings::task_struct,
}
```

Conditional compilation

Rust code has access to conditional compilation based on the kernel config

```
#[cfg(CONFIG_X)]      // `CONFIG_X` is enabled (`y` or `m`)
#[cfg(CONFIG_X="y")]  // `CONFIG_X` is enabled as a built-in (`y`)
#[cfg(CONFIG_X="m")]  // `CONFIG_X` is enabled as a module   (`m`)
#[cfg(not(CONFIG_X))] // `CONFIG_X` is disabled
```

Coding guidelines

No direct access to C bindings

No undocumented public APIs

No implicit `unsafe` block

Docs follows Rust standard library style

`// SAFETY` proofs for all `unsafe` blocks

Clippy linting enabled

Automatic formatting enforced

Rust 2018 edition & idioms

No unneeded panics

No infallible allocations

...

Coding guidelines

No direct access to C bindings

No undocumented public APIs

No implicit `unsafe` block

Docs follows Rust standard library style

`// SAFETY` proofs for all `unsafe` blocks

Clippy linting enabled

Automatic formatting enforced

Rust 2018 edition & idioms

No unneeded panics

No infallible allocations

...

Aiming to be as strict as possible

Abstractions code

```rust
/// Wraps the kernel's `struct file`.
///
/// # Invariants
///
/// The pointer `File::ptr` is non-null and valid.
/// Its reference count is also non-zero.
pub struct File {
   pub(crate) ptr: *mut bindings::file,
}
```

```
impl File {
    /// Constructs a new [`struct file`] wrapper from a file descriptor.
    ///
    /// The file descriptor belongs to the current process.
    pub fn from_fd(fd: u32) -> Result<Self> {
        // SAFETY: FFI call, there are no requirements on `fd`.
        let ptr = unsafe { bindings::fget(fd) };
        if ptr.is_null() {
            return Err(Error::EBADF);
        }

        // INVARIANTS: We checked that `ptr` is non-null, so it is valid.
        // `fget` increments the ref count before returning.
        Ok(Self { ptr })
    }

    // ...
}
```

Driver code

```c
static int pl061_resume(struct device *dev)
{
    int offset;

    struct pl061 *pl061 = dev_get_drvdata(dev);

    for (offset = 0; offset < PL061_GPIO_NR; offset++) {
        if (pl061->csave_regs.gpio_dir & (BIT(offset)))
            pl061_direction_output(&pl061->gc, offset,
                    pl061->csave_regs.gpio_data &
                    (BIT(offset)));
        else
            pl061_direction_input(&pl061->gc, offset);


    }

    writeb(pl061->csave_regs.gpio_is, pl061->base + GPIOIS);
    writeb(pl061->csave_regs.gpio_ibe, pl061->base + GPIOIBE);
    writeb(pl061->csave_regs.gpio_iev, pl061->base + GPIOIEV);
    writeb(pl061->csave_regs.gpio_ie, pl061->base + GPIOIE);

    return 0;
}
```

```rust
fn resume(data: &Ref<DeviceData>) -> Result {


    let inner = data.lock();
    let pl061 = data.resources().ok_or(Error::ENXIO)?;

    for offset in 0..PL061_GPIO_NR {
        if inner.csave_regs.gpio_dir & bit(offset) != 0 {
            let v = inner.csave_regs.gpio_data & bit(offset) != 0;
            let _ = <Self as gpio::Chip>::direction_output(
                data, offset.into(), v);
        } else {
            let _ = <Self as gpio::Chip>::direction_input(
                data, offset.into());
        }
    }

    pl061.base.writeb(inner.csave_regs.gpio_is, GPIOIS);
    pl061.base.writeb(inner.csave_regs.gpio_ibe, GPIOIBE);
    pl061.base.writeb(inner.csave_regs.gpio_iev, GPIOIEV);
    pl061.base.writeb(inner.csave_regs.gpio_ie, GPIOIE);

    Ok(())
}
```

Testing code

```rust
fn trim_whitespace(mut data: &[u8]) -> &[u8] {
    // ...
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_trim_whitespace() {
        assert_eq!(trim_whitespace(b"foo    "), b"foo");
        assert_eq!(trim_whitespace(b"    foo"), b"foo");
        assert_eq!(trim_whitespace(b"  foo  "), b"foo");
    }
}
```

```
/// Getting the current task and storing it in some struct. The reference count is automatically
/// incremented when creating `State` and decremented when it is dropped:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// struct State {
///    creator: Task,
///    index: u32,
/// }
///
/// impl State {
///    fn new() -> Self {
///        Self {
///            creator: Task::current().clone(),
///            index: 0,
///        }
///    }
/// }
/// ```
```

More details in...

Kangrejos Workshop

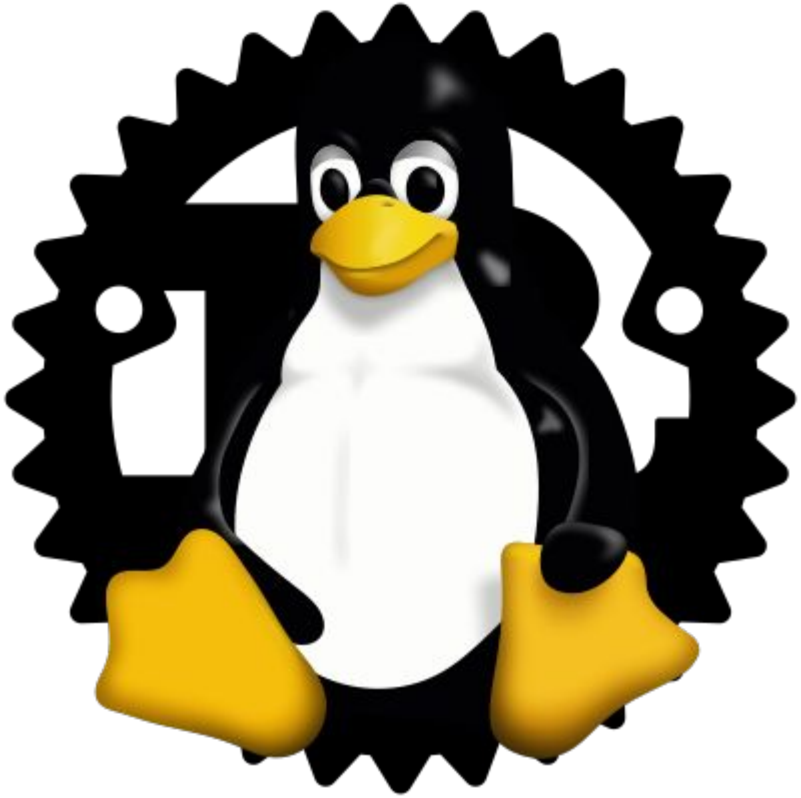
13-15 September

kangrejos.com

Linux Plumbers Conference

20-25 September

linuxplumbersconf.org



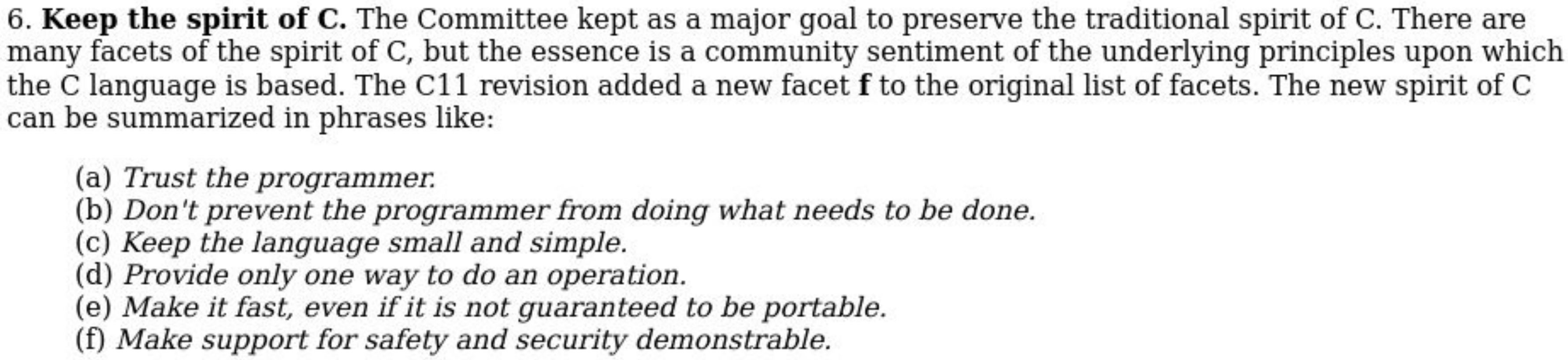
Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Backup slides

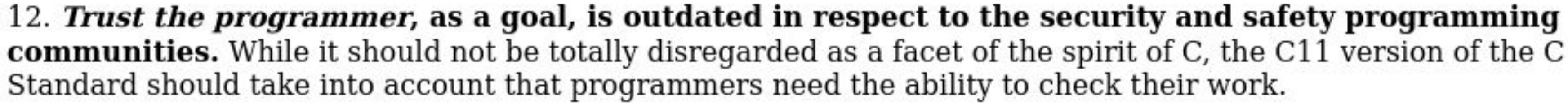
C Charter



6. **Keep the spirit of C.** The Committee kept as a major goal to preserve the traditional spirit of C. There are many facets of the spirit of C, but the essence is a community sentiment of the underlying principles upon which the C language is based. The C11 revision added a new facet **f** to the original list of facets. The new spirit of C can be summarized in phrases like:

(a) *Trust the programmer.*
(b) *Don't prevent the programmer from doing what needs to be done.*
(c) *Keep the language small and simple.*
(d) *Provide only one way to do an operation.*
(e) *Make it fast, even if it is not guaranteed to be portable.*
(f) *Make support for safety and security demonstrable.*

— N2086 C2x Charter - Original Principles



12. ***Trust the programmer*, as a goal, is outdated in respect to the security and safety programming communities.** While it should not be totally disregarded as a facet of the spirit of C, the C11 version of the C Standard should take into account that programmers need the ability to check their work.

— N2086 C2x Charter - Additional Principles for C11