

COMP6991 23T3

Unsafe Rust

Recap: Memory safety

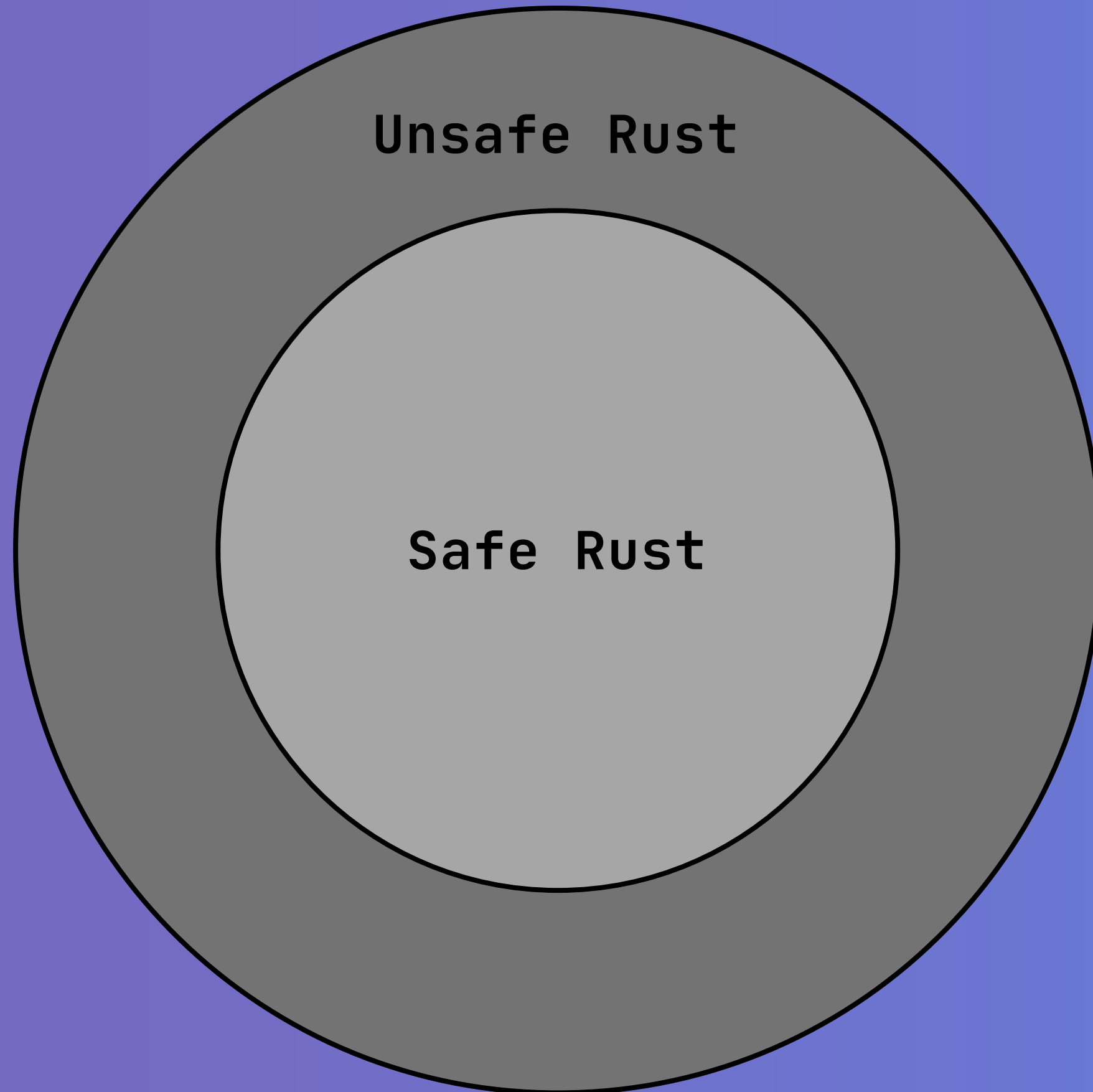
What if we
need low-level
access?

Unsafe Rust



Building blocks of unsafe Rust

- > The ``unsafe`` keyword
- > `*const T, *mut T`
- > `std::ptr::*`
- > `std::alloc::*`
- > `std::ffi::*`
- > (sometimes) ``PhantomData``



**Why segment
the language?**

The advantages of safe Rust

i.e. restricting the language

- > Memory safe by default!
- > No undefined behaviour
- > A relative low complexity
- > Quality diagnostics
- > Optimisation

The advantages of unsafe Rust

i.e. wizardly powers

- > Low-level control
- > Raw system access
- > Fine-grained optimisation in pathological cases
- > Reasoning that the compiler cannot

The disadvantages of safe Rust

i.e. restricting the language

- > Not flexible enough for all use cases
- > e.g. not many would write a (kernel) device driver in Python

The disadvantages of unsafe Rust

i.e. wizardly powers

- > Threatens most advantages of safe Rust
- > e.g.
 - > Memory unsafety
 - > Undefined behaviour
- > Very high complexity

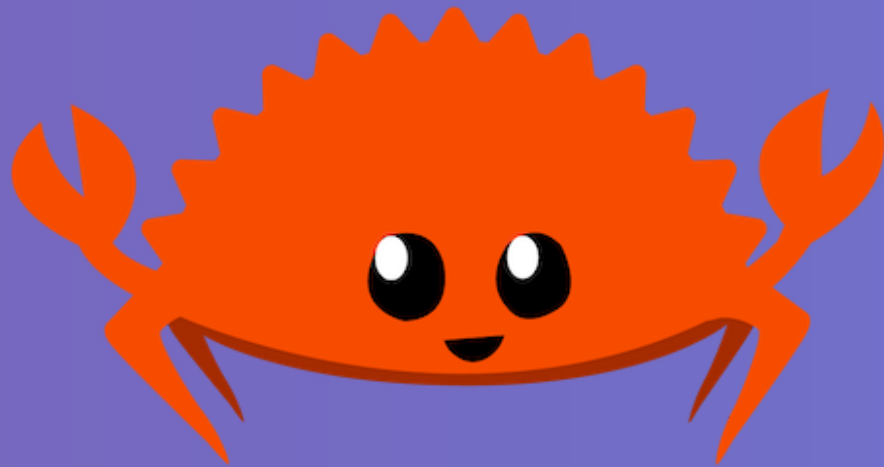
**Maintain guarantees
of safe Rust...**

**...without limiting
language capability?**

**Unsafe implementations
behind safe abstractions**



Safe Rust code



Safety checks pass!

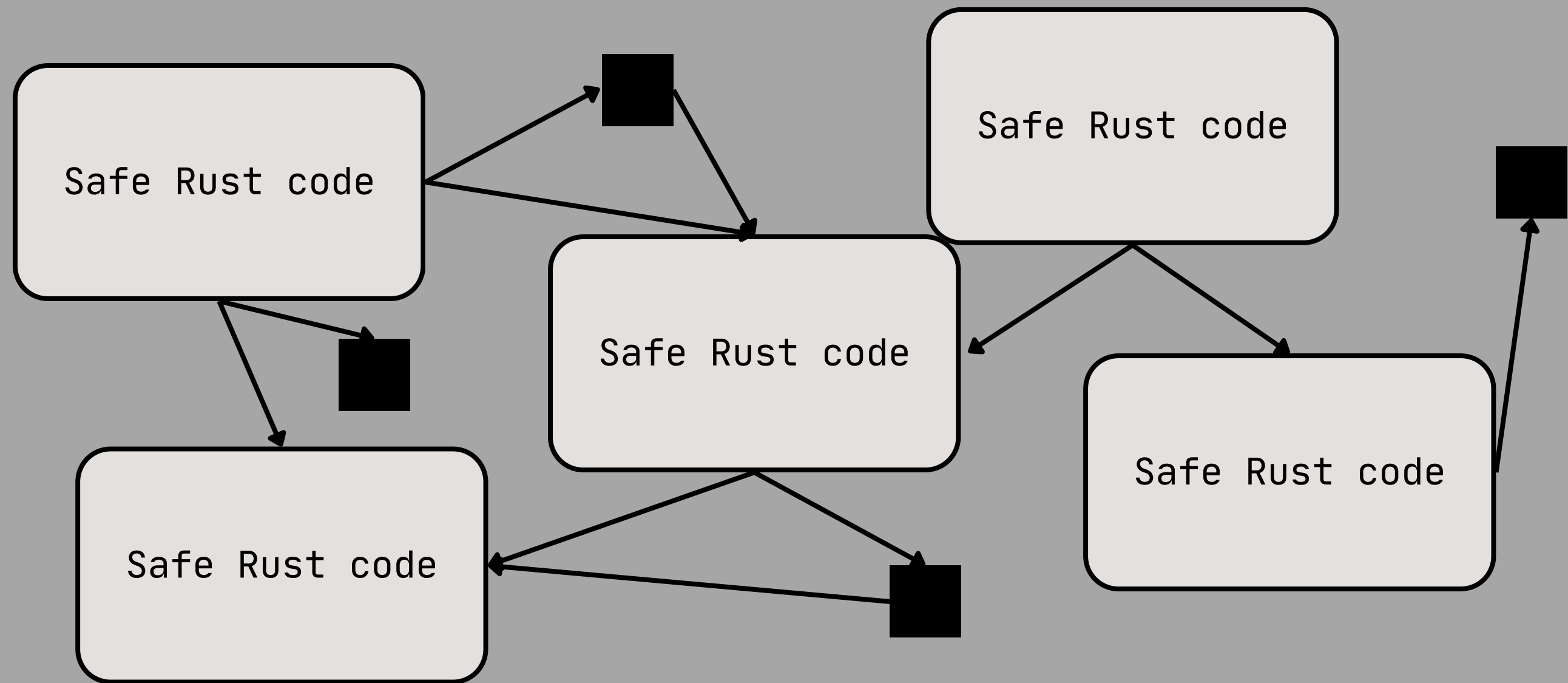
- > type inference
- > type checking
- > trait solving
- > borrowck
- > etc.



Safety checks fail!

Give diagnostic to user

Anatomy of a Rust program



Inside the black box

Safe box

unsafe { crimes }



The diagram features a central dark gray rounded rectangle with a black border. Above the rectangle, the text 'Safe box' is underlined. Inside the rectangle, the text 'unsafe { crimes }' is displayed. A horizontal arrow points from the left edge of the rectangle to the right edge, passing through the center of the box.

Unsafe superpowers

1. Dereference raw pointers
2. Call unsafe functions
3. Implement unsafe traits
4. Read/write a mutable or external static variable (???)
5. Accessing a field of a union, other than to assign to it (???)

**With great power
comes great
responsibility**

**`https://doc.rust-
lang.org/reference/behavior-
considered-undefined.html`**

Aside: UB in C

<https://blog.regehr.org/archives/1520>

<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf#page=571>

Tomorrow

Let's make a (mostly sound) Vec!

Checking unsafe code

> **Miri**

> <https://github.com/rust-lang/miri>

> **Sanitizers**

> <https://doc.rust-lang.org/nightly/unstable-book/compiler-flags/sanitizer.html>

> **With concurrency: Loom**

> <https://docs.rs/loom/latest/loom/>