Arc in the Linux Kernel

Alice Ryhl

Rust in the Linux Kernel

- The Rust for Linux project started in 2020.
- First RFC to add Rust to Linux in April 2021.
- Actually merged in October 2022.
- I sent the Rust Binder RFC in November 2023.

What does it take for Rust to succeed in the kernel?

- Real Rust drivers used in the real world.
- Good first impressions for people coming from C.
- Do not require a specific version of rustc.
- Compile Rust with GCC.

What does it take for Rust to succeed in the kernel?

- **Real Rust drivers used in the real world.**
- Good first impressions for people coming from C.
- Do not require a specific version of rustc.
- Compile Rust with GCC.

My previous talks

GoLab + RustLab 2023

Alice Ryhl

KEYNOTE: Rust in the Linux kernel

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

Using Rust in the binder driver

Alice Ryhl et al.

What does it take for Rust to succeed in the kernel?

- Real Rust drivers used in the real world.
- **Good first impressions for people coming from C.**
- **Do not require a specific version of rustc.**
- Compile Rust with GCC.

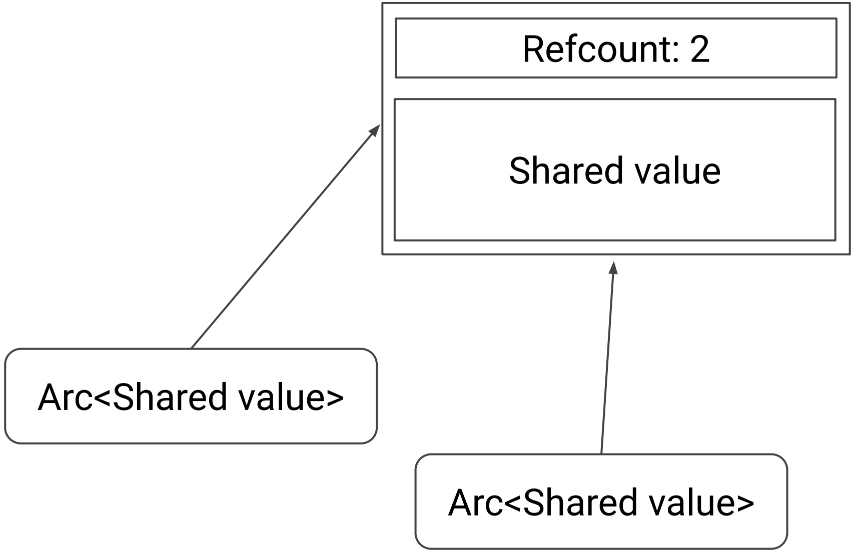
This talk

Goals for this talk

- The Linux Kernel needs unstable Rust features.
- This talk aims to explain why.
- Deep dive on unstable features related to custom Arc:
  - Arbitrary self types.
  - #[derive(SmartPointer)].

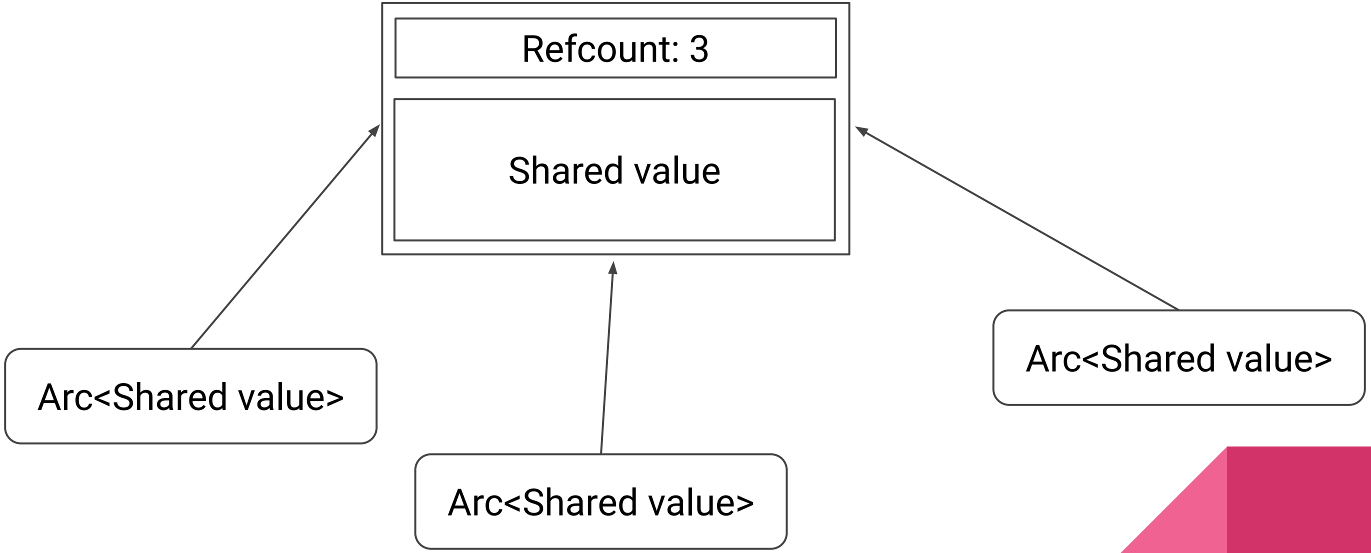
What is Arc?

- Reference counted container for sharing immutable values.



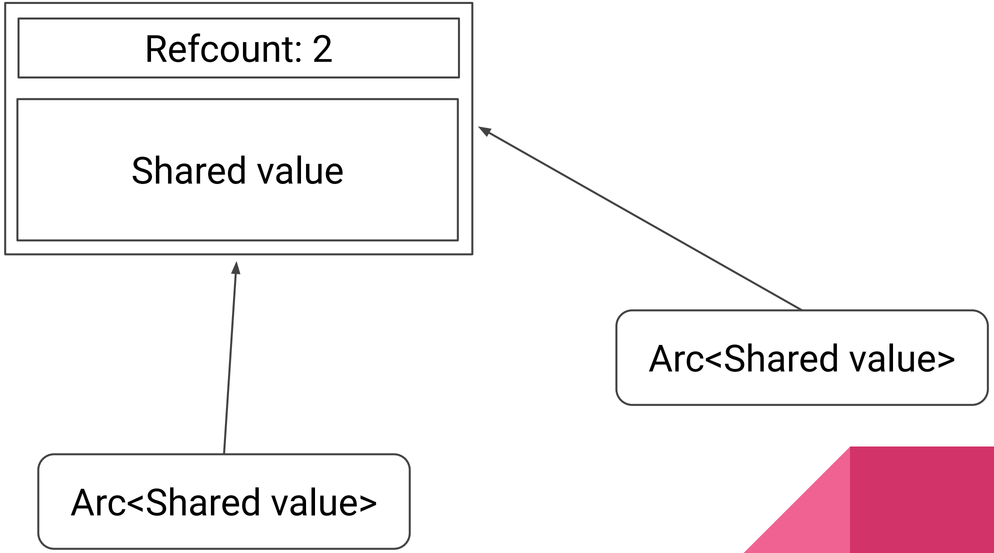
What is Arc?

- Reference counted container for sharing immutable values.



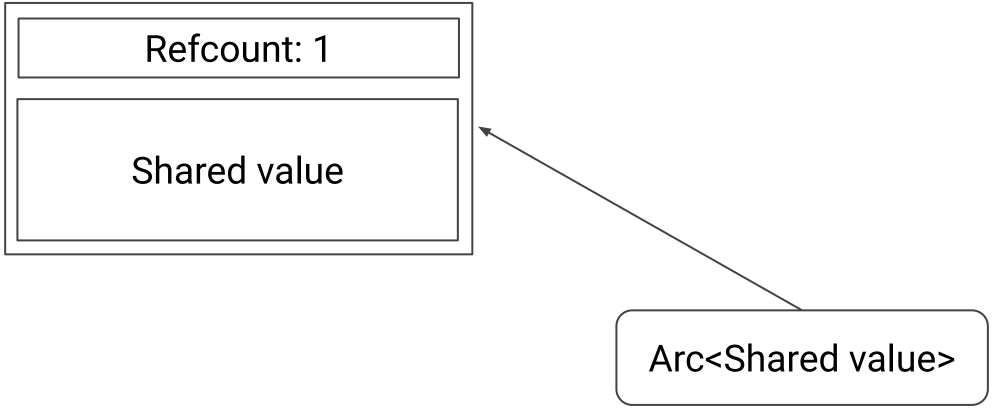
What is Arc?

- Reference counted container for sharing immutable values.



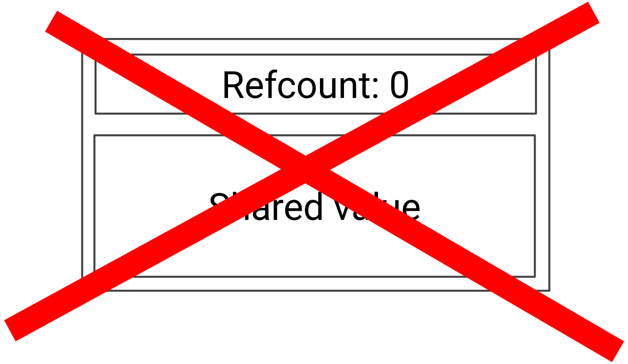
What is Arc?

- Reference counted container for sharing immutable values.



What is Arc?

- Reference counted container for sharing immutable values.



Linux has a custom Arc

- Linux can’t use the standard library Arc.
- But the standard library Arc is special.
- You can’t implement your own Arc.
- Linux uses unstable features to work around this.

Do not call abort on overflow

- When you call abort in the kernel, your device turns off.
  - Even if something goes wrong, it’s better to try and carry on.
- When the refcount hits `isize::MAX`, replace it with ∞.
  - (and print a warning)
- Once it hits ∞, the refcount stays there forever.

The Linux Kernel Memory Model

- Arc uses an atomic integer for the refcount.
- Rust uses the C++ memory model for atomics.
- Linux uses the Linux Kernel Memory Model for atomics.
- Rust in Linux must also use LKMM for atomics.
  - We use inline assembly for all atomic operations.

Nice to have: Pin every Arc

- Pretty much every C type is !Unpin.
    - This includes Mutex.
- Linux’s custom Arc will pin all values automatically.

Nice to have: No weak pointers

- The standard library Arc has weak references.
- Not needed in the Linux Kernel.
- Removing them simplifies our Arc.

Other custom types similar to Arc

Types that are similar to Arc. They need the same unstable features.

- ArcBorrow - similar to &Arc<T>
- UniqueArc - Arc with mutable access
- ListArc - Version of Arc for linked lists

Arbitrary self types

```
impl MyStruct {
    fn my_func1(&self) { ... }
    fn my_func2(&mut self) { ... }

    fn my_func3(self: Arc<Self>) {
        ...
    }
}
```

This is unstable.

Arbitrary self types

```
impl MyStruct {
    fn my_func1(&self) { ... }
    fn my_func2(&mut self) { ... }

    fn my_func3(self: Arc<Self>) {
        let arc = self.clone();
    }
}
```

Makes this possible.

Linked lists also pose a problem

- Linked lists are:
  - Hard to implement in Rust.
  - And perform worse than Vec.
- Yet, the Linux Kernel still uses them (with Arc!)

Atomic context

```
let guard = spin_lock.lock();

guard.list.insert(value);

drop(guard);
```

While spinlock is locked, you can’t allocate memory.

How do you implement `insert` without allocating memory?

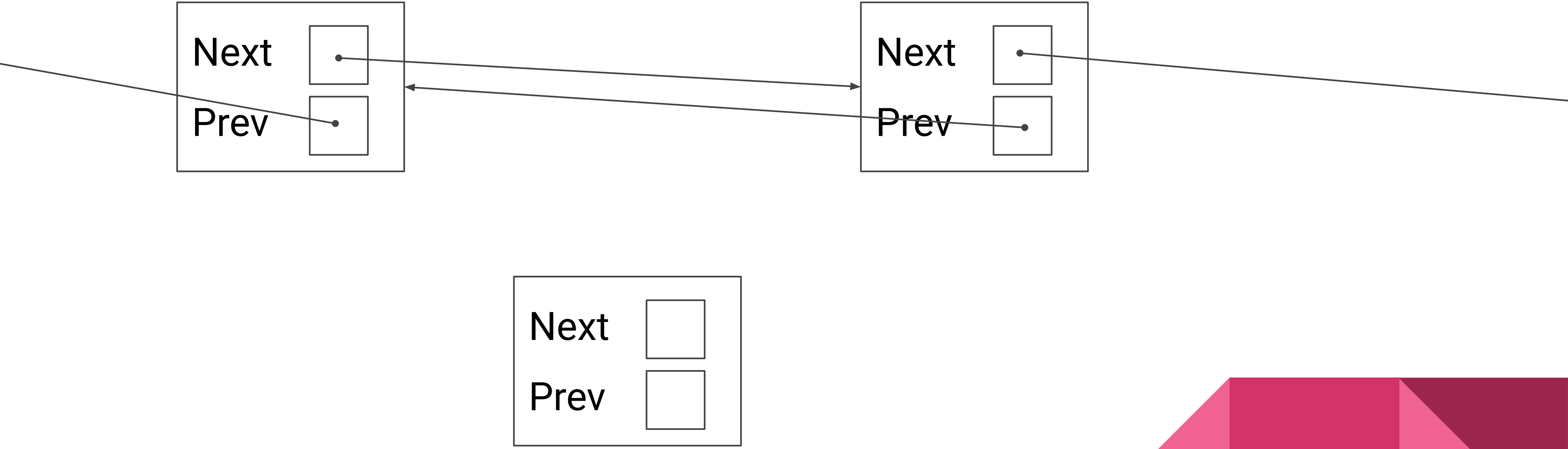
With a vector

- If there is no more space:
  a. Exit the spinlock.
  b. Allocate memory for a larger vector.
  c. Re-enter the spinlock.
  d. Move all elements to larger vector.

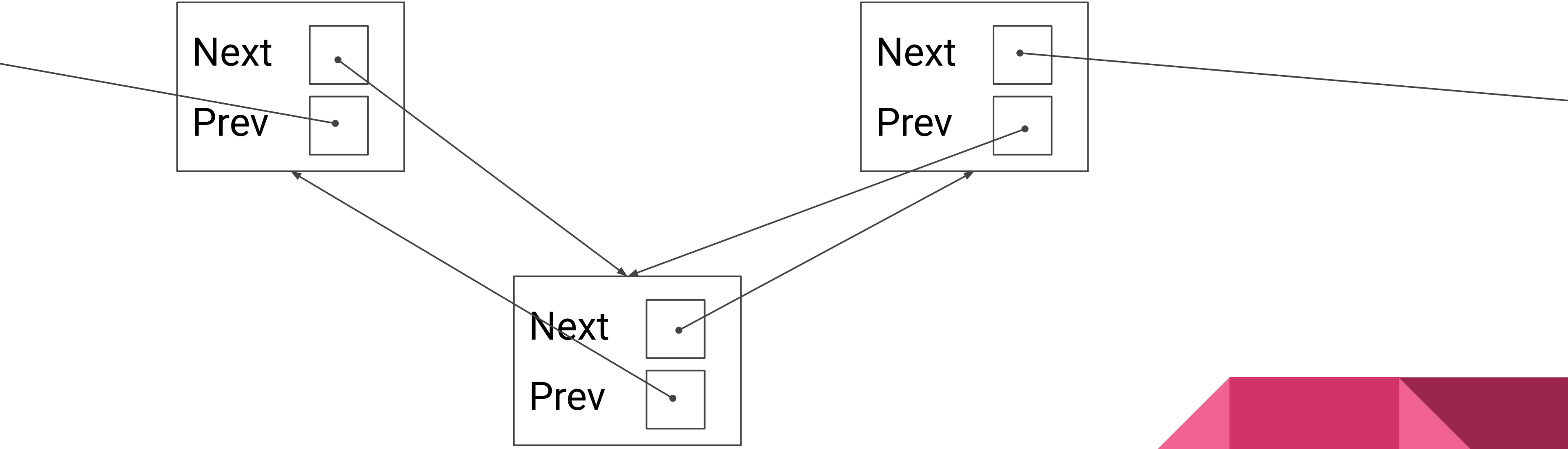
With a linked list

- To insert, modify prev/next pointers.
- That’s it!
- No allocations needed.

Linked list



Linked list



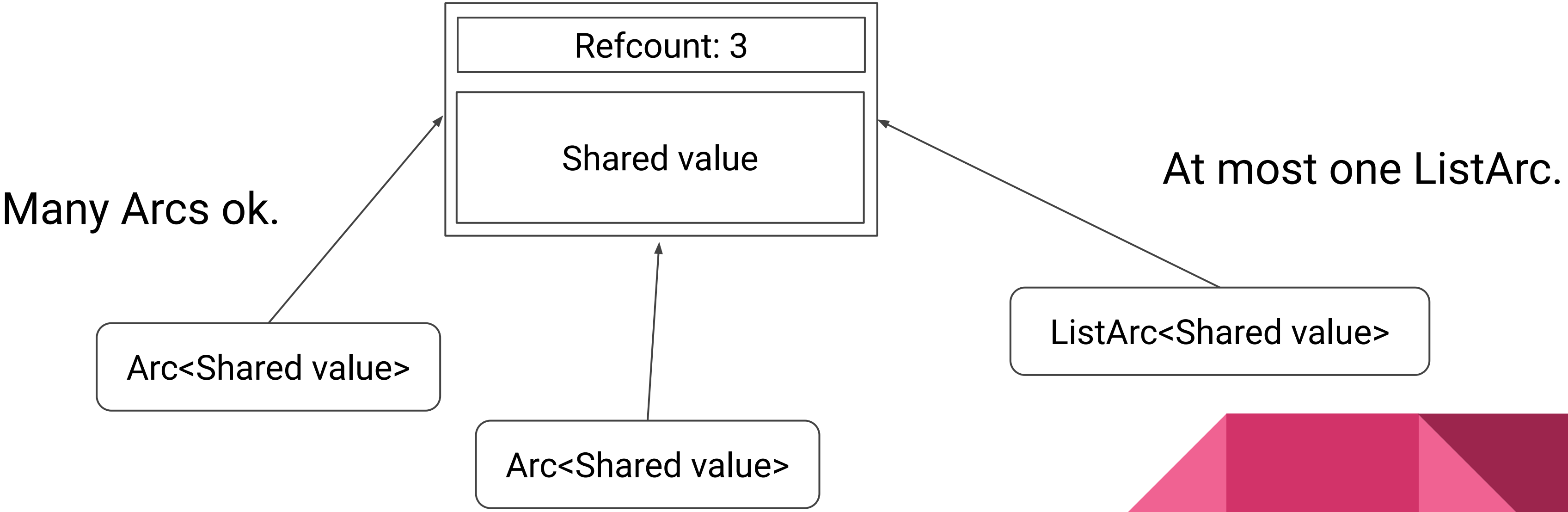
Linked list

```
struct MyValue {
    // In practice, these are wrapped into one field using a struct.
    next: *mut MyValue,
    prev: *mut MyValue,
    foo: Foo,
    bar: Bar,
}
```

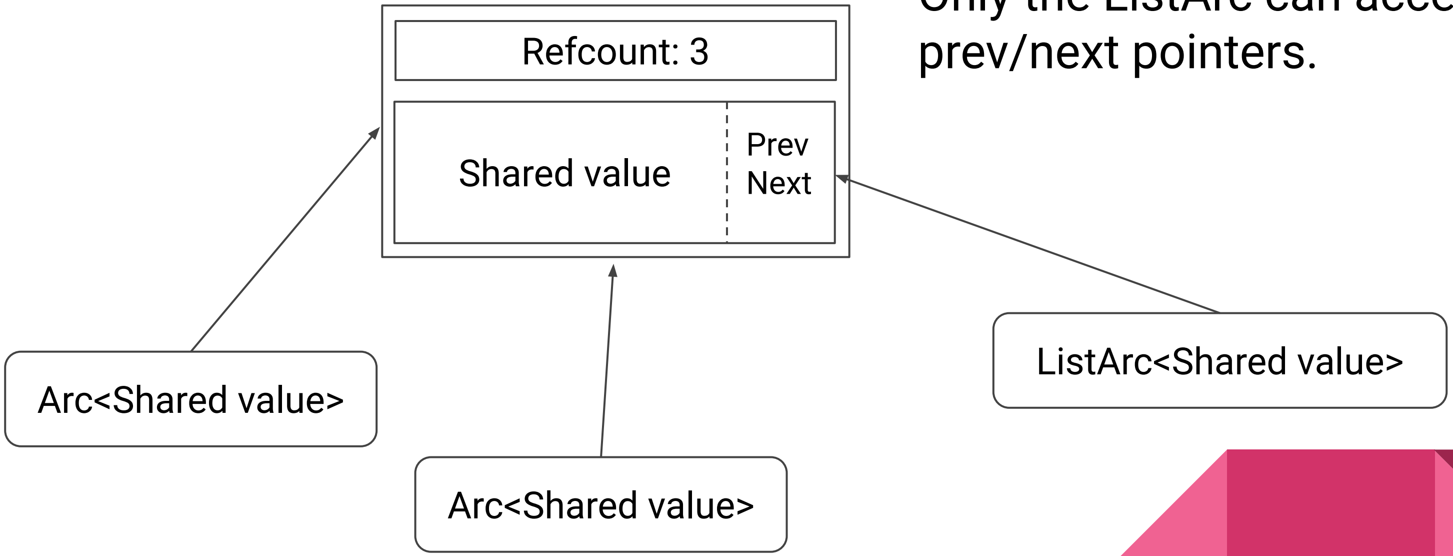
Using Arc for linked list elements

- There is a problem:
- You can have multiple Arcs to the same shared value.
- But you only have one pair of prev/next pointers.
- What if you insert into two different lists in parallel?

Solution: the ListArc



Solution: the ListArc



Only the ListArc can access prev/next pointers.

Queue of events

- Linked lists are used for queues of events.
- Many different types of events.
    - LinkedList<dyn EventTrait>

Trait for events

```
trait EventTrait {
    fn run_event(self: ListArc<Self>);
}
```

Can’t just be &self.

Traits with custom Arc

- Using `dyn Trait` with a custom Arc is also unstable.
- Requires many unstable features.

Traits with Arc

```
impl<T, U> CoerceUnsized<Arc<U>> for Arc<T>
where
    T: Unsize<U>,
    T: ?Sized,
    U: ?Sized,
{}
```

Unstable

Traits with Arc

```
impl<T, U> DispatchFromDyn<Arc<U>> for Arc<T>
where
    T: Unsize<U>,
    T: ?Sized,
    U: ?Sized,
{}
```

Unstable

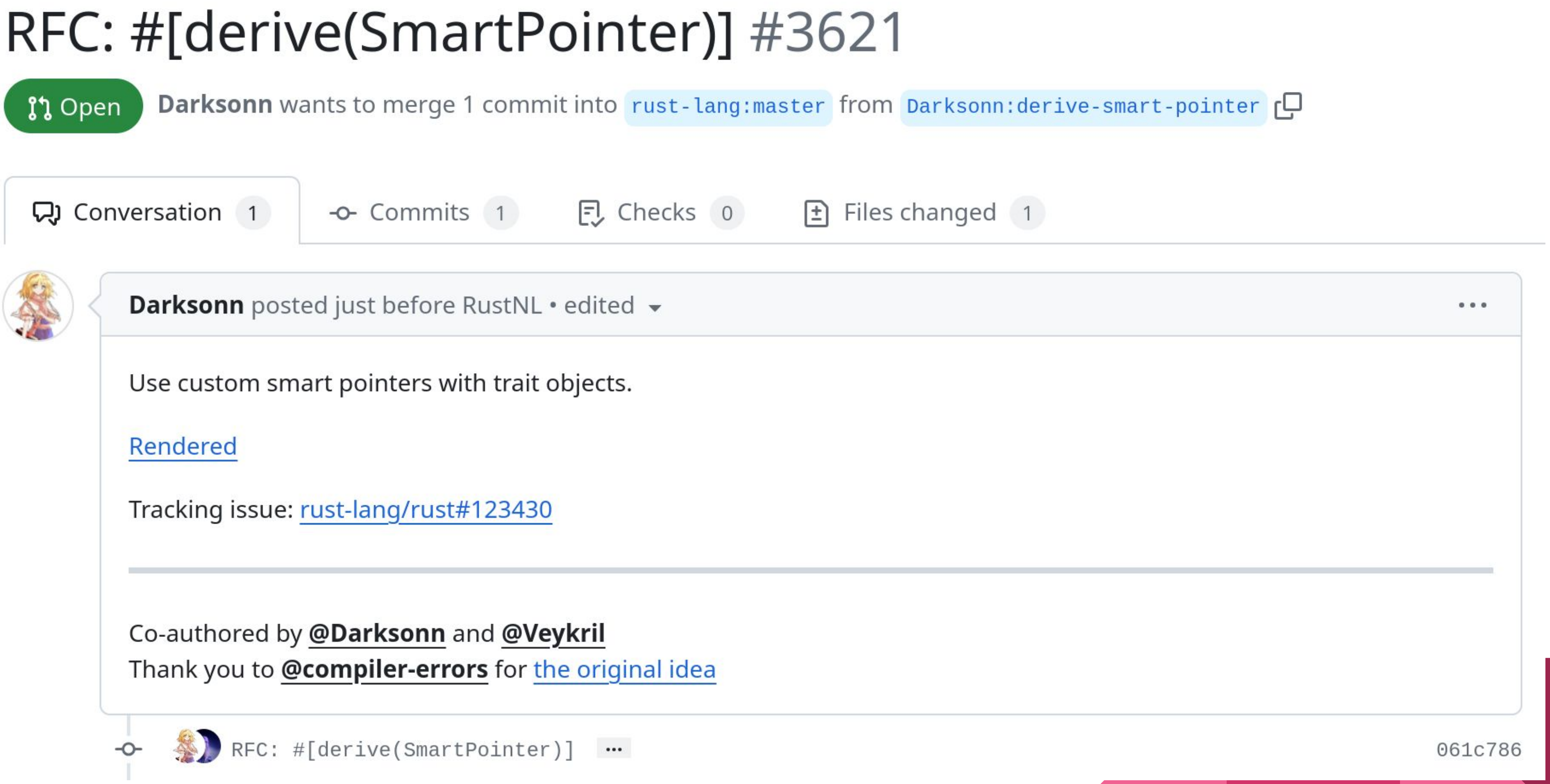
Traits with Arc

```
impl<T, U> DispatchFromDyn<Arc<U>> for Arc<T>
where
    T: Unsize<U>,
    T: ?Sized,
    U: ?Sized,
{}
```

Unstable

Solution: #[derive(SmartPointer)]

- We stabilize a derive macro.
- We do *not* stabilize what it expands to.
- This allows us to change the underlying traits in the future.



RFC: #[derive(SmartPointer)] #3621

Darksonn wants to merge 1 commit into `rust-lang:master` from `Darksonn:derive-smart-pointer`

Darksonn posted just before RustNL • edited

Use custom smart pointers with trait objects.

Rendered

Tracking issue: rust-lang/rust#123430

Co-authored by **@Darksonn** and **@Veykril**
Thank you to **@compiler-errors** for the original idea

`RFC: #[derive(SmartPointer)]` ··· `061c786`