

Arc<Mutex<T>>



...and ways to avoid it


it can be a fuckin pain in the ass 22:28

but i am kinda sorta getting the hang of it 22:29

when in doubt just wrap everything in an `Arc<RwLock<...>>`
right 22:29

ezpz 22:29

20 July

 Jesus Higuera
when in doubt just wrap everything in an `Arc<RwLock<...>>` right

Noooooooooooo well sometimes

15:51 ✓✓

My website = 18K lines of Rust
(first commit in June 2020)

Lots of async/sync mixing
(weird tricks, not today's topic)

Typical stack: axum on tokio
(used to be tide, then warp)

tokio = thread pool
(tasks can be polled from any thread)

thread pool = Arc all the things
(tokio::spawn needs 'static, etc.)

But: do you ever free it?
If not: `&'static T` (via `Box::leak`)

Does it ever change?

If no: $\text{Arc}\langle T \rangle$, if yes: $\text{Arc}\langle \text{Mutex}\langle T \rangle \rangle$

Is it write-heavy?

If no: `Arc<RwLock<T>>`, if yes: `Arc<Mutex<T>>`

Does it fit in an atomic?

If yes: `AtomicU8` (dev vs prod environment), etc.

Is it *almost* a `lazy_static`?
But it needs a CLI arg or something? `AtomicPtr`

Does it change **while blocking?**

If not: listen up!

Implicit context is bad
Unless I'm doing it

tokio has implicit context
so does tracing-subscriber, etc.

Sometimes we must!

grass_compiler takes a function, not a closure

I used to do something awful
(process-wide locks)

Sometimes Arcs will do
(liquid templating engine, rebuilding filters)

But consider: thread-locals
You will certainly not regret it.

Easy case: owned type
`LocalKey<T>`

What if you want to share?

Several threads cannot own the same T

My terrible hack:

```
LocalKey<RefCell<Option<*const T>>>
```

Main insight:

Nobody can mess with your thread-local while blocking

Main takeaway:

You can lend a shared reference to arbitrary blocking code!

(Show code)