what are you syncing about?

I want to talk about one of my favorite async libraries:

maitake-sync

comes from `mycelium`

maitake

the executor

`maitake-sync`

"synchronization primitives"

async code is all about

waiting

sync primitives are all about *notifications*

they are useful for building data structures or drivers or network stacks or...

`maitake-sync`

has three sync primitives I love:

`WaitCell`,
`WaitQueue`,
and `WaitMap`

`WaitCell`

holds zero or one

`Waker`s

Sort of like

`Mutex<Option<Waker>>`

limitation:
doesn't work with >1 tasks

`WaitQueue`

holds 0..∞ `Wakers`

but where do you store those `Waker`s?

solution:
doubly linked lists

specifically:
intrusive doubly linked lists
(from the `cordyceps`
crate)

once you've `poll`'d a `Future`,
you get two things:

```rust
pub trait Future {
    type Output;

    // Required method
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}
```

a `pin`ned `self` means we have a stable pointer we can (ab)use

upside: very flexible
especially on embedded!

downside: it's a doubly linked list

`Waitmap`
is the fun oddball of the group

instead of JUST a waker,
we also have
a Key and Value

example: async mailbox

upside: very flexible,
we don't need a bunch of
oneshot channels

downside: it's a doubly linked list (but worse)