Async

Técnicas de Programación Concurrente

Cuando usar async

- Calcular un factorial
- Calcular un producto de matrices
- Implementar el problema de los filósofos
- Estado interno compartido

- Consultar servicios externos
- Leer de un archivo
- Servir requests HTTP

Cómo funciona async

```
impl Future for Main {
    fn poll(mut self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<()> {
        loop {
            match self.state {
                State::AwaitingHello => match self.hello.poll(cx) {
                    Poll::Pending => return Poll::Pending,
                    Poll::Ready(r) => {
                        self.state = State::AwaitingWorld;
                        self.hello_result = Poll::Ready(r)
                    },
                }
                State::AwaitingWorld => match self.world.poll(cx) {...}
                State::Done => {
                    ...
                    return Poll::Ready(())
                },
            }
        }
    }
}
```

Estilo funcional

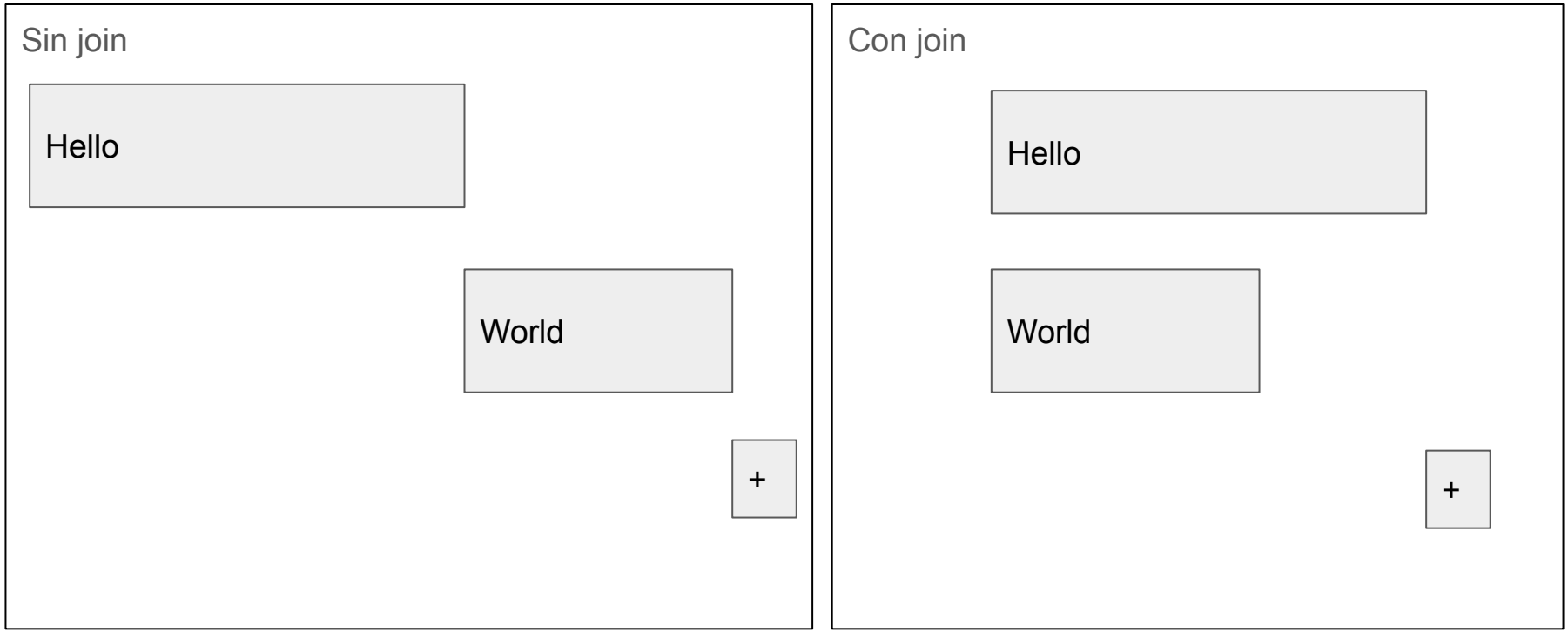
Los futures son functors (se pueden map) y monads (se pueden flatten)

```
hello()
    .map(|h| world().map(|w| h + w.as_str()))
    .flatten()
```

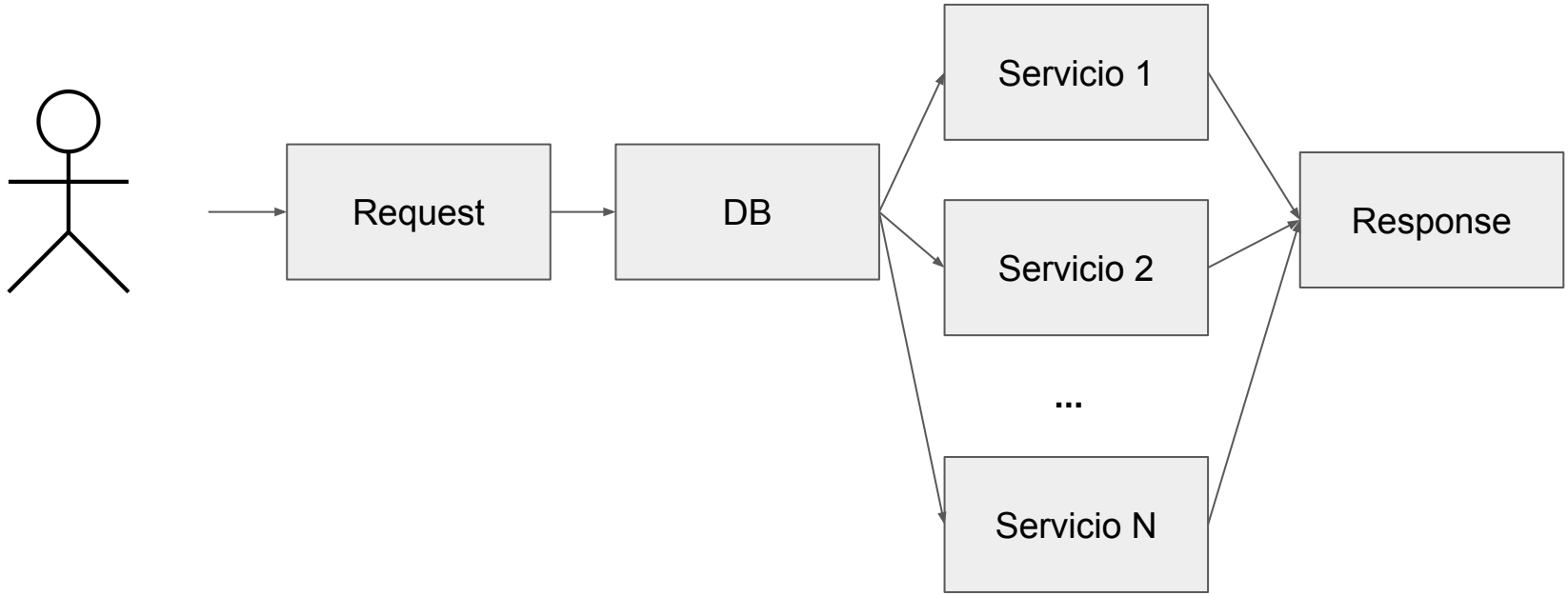
Encadenar futures

```rust
async fn hello() -> String {
    println!("before hello");
    task::sleep(Duration::from_secs(2)).await;
    println!("after hello");
    String::from("Hello")
}

async fn world() -> String {
    println!("before world");
    task::sleep(Duration::from_secs(1)).await;
    println!("after world");
    String::from(" World!")
}

async fn async_main() -> String {
    hello().await + world().await.as_str()
}
```

Encadenar futures



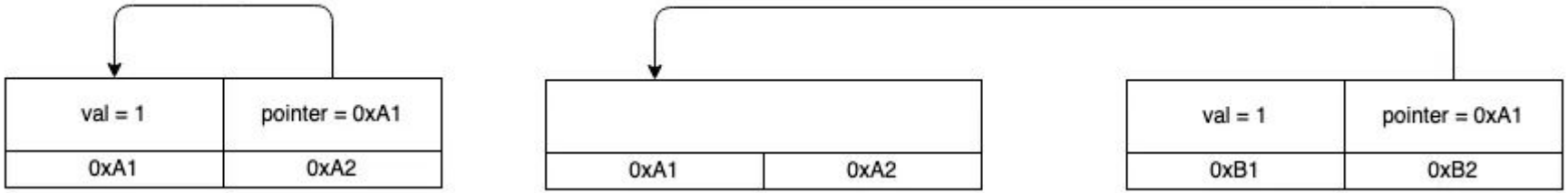
Ejemplo del mundo real



Pin

Por qué?

- Los tipos autogenerados de async que implementan Future guardan referencias a sí mismos
- Si ellos fueran movidos (por ejemplo por estar en el stack) sus referencias internas no se actualizarían



Pin

Cómo?

- Todos los tipos por defecto implementan el autotrait Unpin (como Send)
  - Salvo que específicamente se marquen como !Unpin

- Las self-references se encierran en un tipo Pin (ej Pin<Box<T>>)
- Si T es !Unpin, Pin "evita" que se mueva haciendo imposible llamar métodos que requieran &mut T como mem::swap

Runtimes

Los runtimes como Tokio y async-std incluyen:

- Executors: "Traducen" await en llamadas a poll.
  - Con qué frecuencia?
  - Sobre qué thread(s)?

- Bibliotecas
  - async IO
  - timers
  - locks
  - channels
  - etc

Ejercicio

Realizar un pequeño utilitario que utilizando el crate reqwest imprima por pantalla los 10 tweets mas destacados de cada uno de los trending topics de Argentina usando las APIs de Twitter:
https://developer.twitter.com/en/docs/twitter-api/v1/trends/trends-for-location/api-reference/get-trends-place
https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/guides/build-standard-query

Referencias

https://rust-lang.github.io/async-book/

https://www.ncameron.org/blog/what-is-an-async-runtime/

https://blog.adamchalmers.com/pin-unpin/

https://doc.rust-lang.org/stable/std/pin/index.html

https://cfsamson.github.io/books-futures-explained/