Async

Técnicas de Programación Concurrente

Cuando usar async

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

- 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()
```

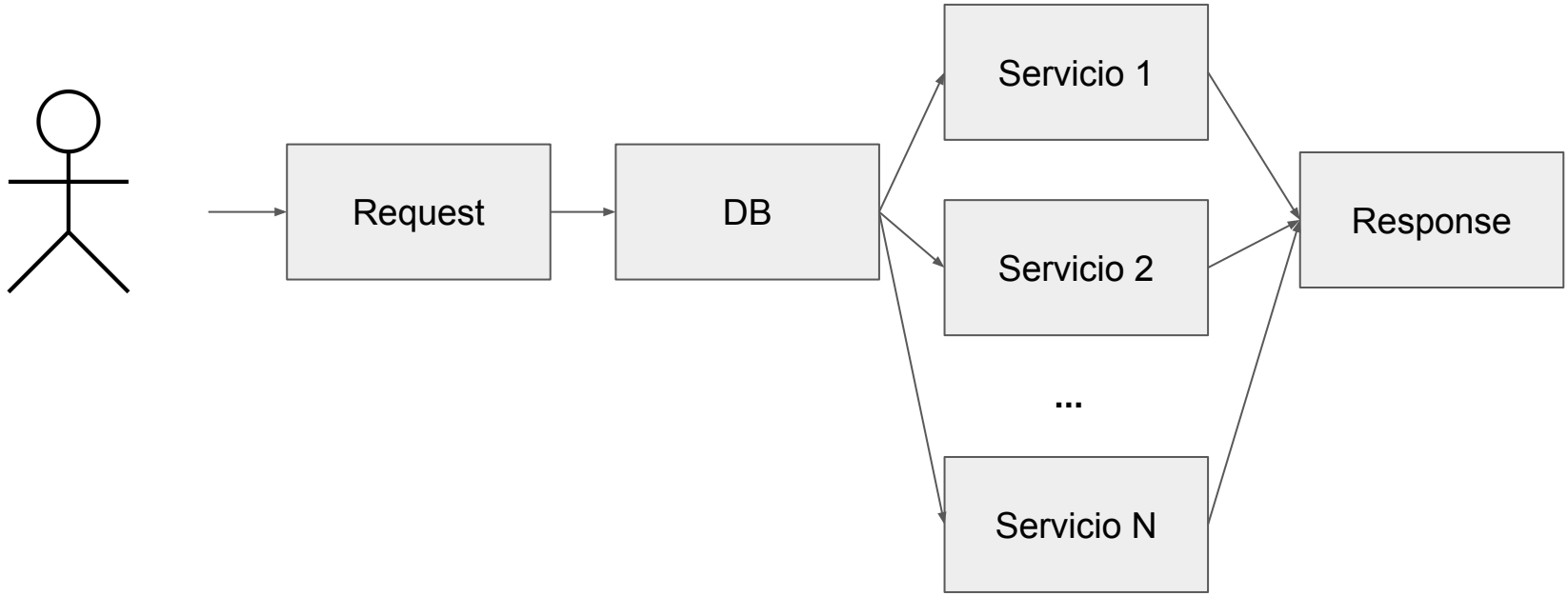
Encadenar futures

```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()
}
```

Ejemplo del mundo real



Pin

En Rust los objetos pueden estar allocados en el stack, es por ello que se "mueven".

Referencias

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