

Rust: async

Hafta - 6

async nedir?

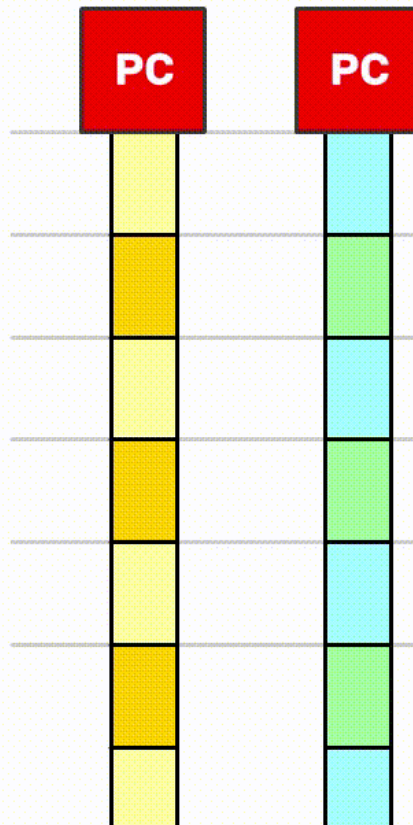
Parallel:

Birden fazla thread'in farklı işlemci çekirdeklerinde aynı anda yürütülmesi

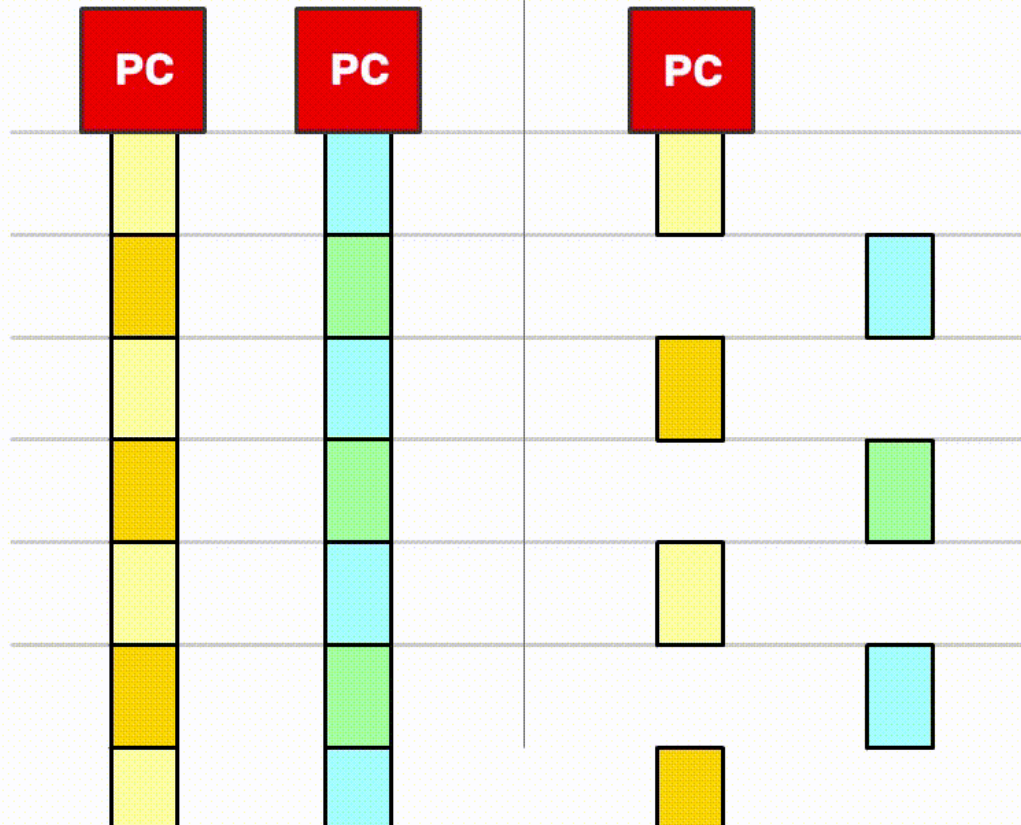
Concurrent (Eş Zamanlı):

Tek thread ve tek çekirdek ile birden fazla programı parça parça yürütme

Paralel (parallel)



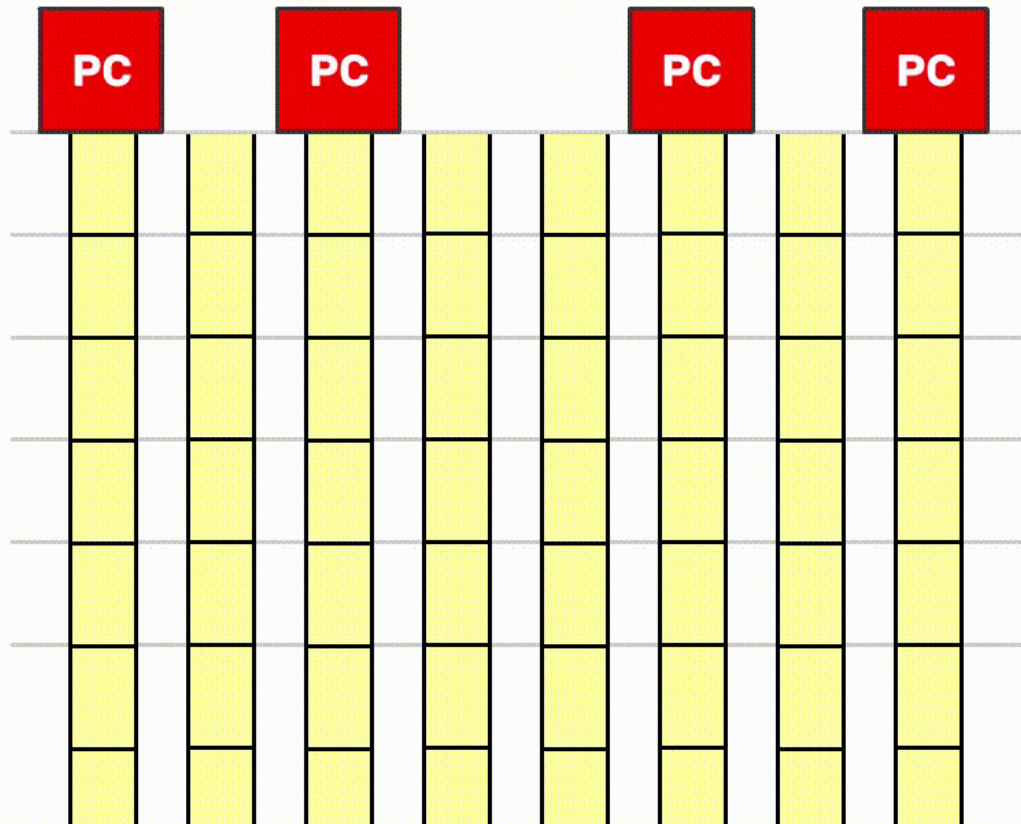
Eş Zamanlı (concurrent)

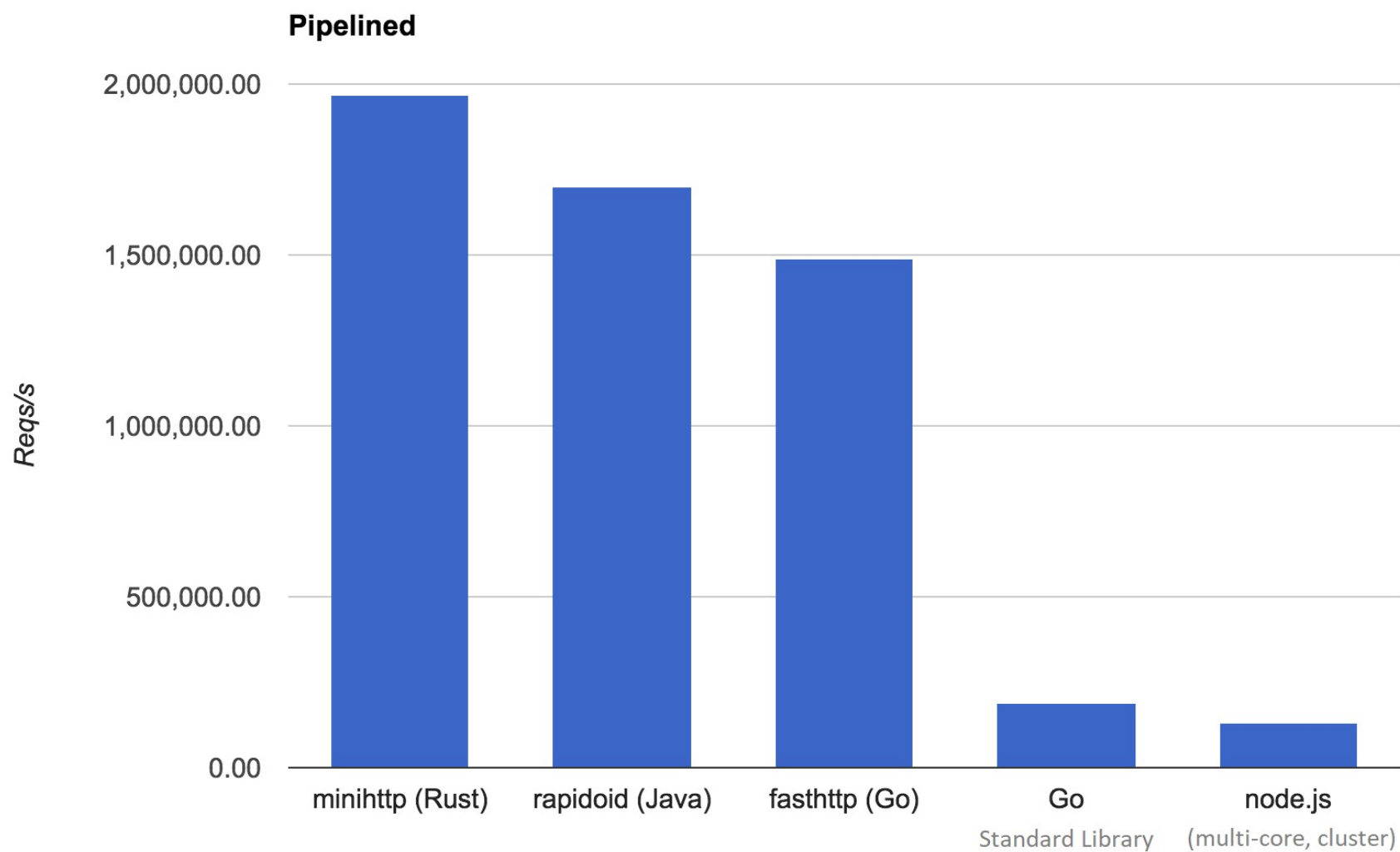


Asynchronous (Asenkron):

Programın Concurrent veya Parallel bir yapı sayesinde
bloklanmadan yürütülmesi

Asenkron (asynchronous)





Future

Asenkron bir iş.

Future

```
async fn ornek() → i32 {  
    42  
}
```



```
fn ornek() → impl Future<Output=i32> {  
    async {  
        42  
    }  
}
```

İkisi aynı

Executor

Future'ları alıp çalıştıran runtime.

Örneğin: async-std, tokio

Executor

```
use async_std::task;

async fn ornek() -> i32 {
    42
}

fn main() {
    task::block_on(async {
        println!("ornek() = {}", ornek().await);
    });
}
```

> ornek() = 42

Executor

```
use async_std::task;

async fn ornek() -> i32 {
    42
}

async fn async_main() {
    println!("ornek() = {}", ornek());
}

fn main() {
    task::block_on(async_main());
}
```

```
> --> src/main.rs:8:27
|
8 |     println!("ornek() = {}", ornek());
|                                     ^^^^^^^^ `impl Future`
cannot be formatted with the default formatter
|
```

Executor

Future'lar tek başına çalışmaz.

Bir Executor'a "al bunu çalıştır" diyerek verilmeli.

Bunu Future'u `.await` ile bekleyerek yapabiliriz.

Executor

```
use async_std::task;

async fn ornek() -> i32 {
    42
}

async fn async_main() {
    println!("ornek() = {}", ornek().await);
}

fn main() {
    task::block_on(async_main());
}
```

> ornek() = 42

Executor

.await ile beklenen future'lar sonuçlanmadan program bir aşağıdaki satırı çalıştırmaz.

Executor, .await ile beklenen future'u yield edip başka bir future'u çalıştırmaya devam edebilir. Bu da asenkronluğu sağlar.

Örneğin siz bir dosya okuma işlemini .await ederken, dosya hazır olana kadar Executor başka bir .await ile beklenen future'u çalıştırmaya geçebilir.

Executor

```
use std::time::Duration;
use async_std::task;

async fn bekle() {
    task::sleep(Duration::from_millis(1000)).await;
    println!("Merhaba");
}

async fn async_main() {
    bekle().await;
    bekle().await;
}

fn main() {
    task::block_on(async_main());
}
```

```
- 1 saniye bekler -
> Merhaba
- 1 saniye bekler -
> Merhaba
```

Örnek

Terminal'den okuduğunu terminal'e yazma

```
use async_std::io::{WriteExt, self}; // std::io değil, async_std::io kullanılmalı.
use async_std::task;

async fn terminalden_oku() -> io::Result<String> {
    let mut buffer = String::new();

    io::stdin().read_line(&mut buffer).await?;

    Ok(buffer)
}

async fn async_main() -> io::Result<()> {
    let okunan = terminalden_oku().await?;
    io::stdout().write_all(okunan.as_bytes()).await?;

    Ok(())
}

fn main() -> io::Result<()>{
    task::block_on(async_main())
}
```

< selam

> selam

Örnek

Terminal'den okuduğunu terminal'e yazma

```
use async_std::io::{WriteExt, self}; // std::io değil, async_std::io kullanılmalı.
use async_std::task;

fn main() -> io::Result<()> {
    task::block_on(async {
        let stdin = io::stdin();
        let mut stdout = io::stdout();
        let mut line = String::new();

        loop {
            let n = stdin.read_line(&mut line).await?;

            // EOL
            if n == 0 {
                return Ok(());
            }

            stdout.write_all(line.as_bytes()).await?;
            stdout.flush().await?;
            line.clear();
        }
    })
}
```

< selam

> selam

Örnek

Async TCP Server

```
use async_std::io::{WriteExt, ReadExt, self};
use async_std::task;
use async_std::net::{TcpListener, TcpStream};

async fn on_connection(mut stream: TcpStream) -> io::Result<()> {
    println!("New Connection: {}", stream.peer_addr()?);

    let mut buffer = [0u8; 1024];
    loop {
        let len = stream.read(&mut buffer).await?;
        if len > 0 {
            println!("{}", String::from_utf8_lossy(&buffer[..len]));

            stream.write(&buffer).await?;
        } else {
            println!("Disconnected: {}", stream.peer_addr()?);
            break
        }
    }

    Ok(())
}

async fn async_main() -> io::Result<()> {
    let listener = TcpListener::bind("127.0.0.1:8080").await?;
    println!("Listening on {}", listener.local_addr()?);

    loop {
        let (stream, addr) = listener.accept().await?;
        task::spawn(on_connection(stream));
    }
}

fn main() -> io::Result<()> {
    task::block_on(async_main())
}
```

Homeworks

1. Write async tcp chat server with "async-std"

- Send each message to other connections.
- Use Arc<Mutex<>> to share "user list" between the tasks.
- Use 22222 port to make us test it easily.