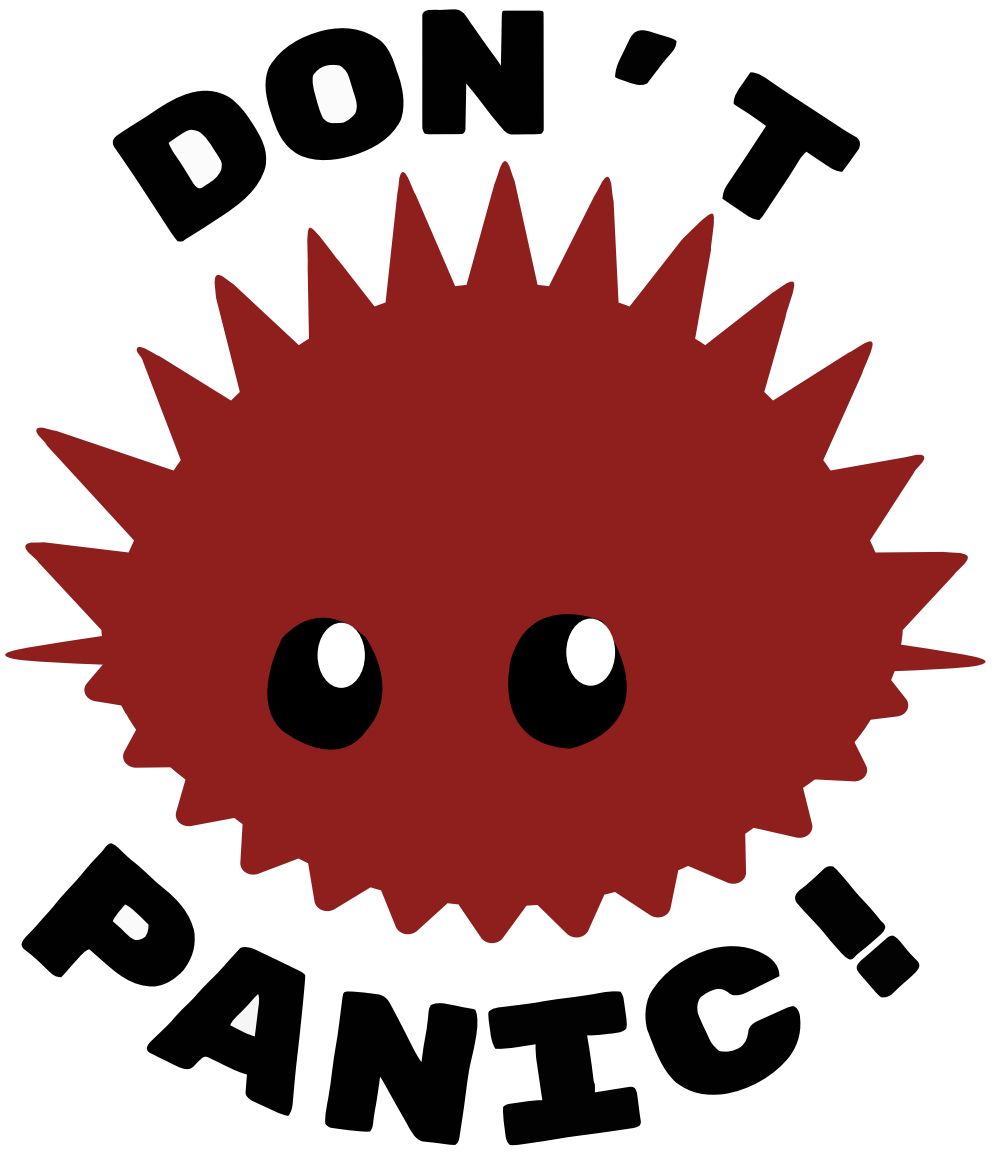
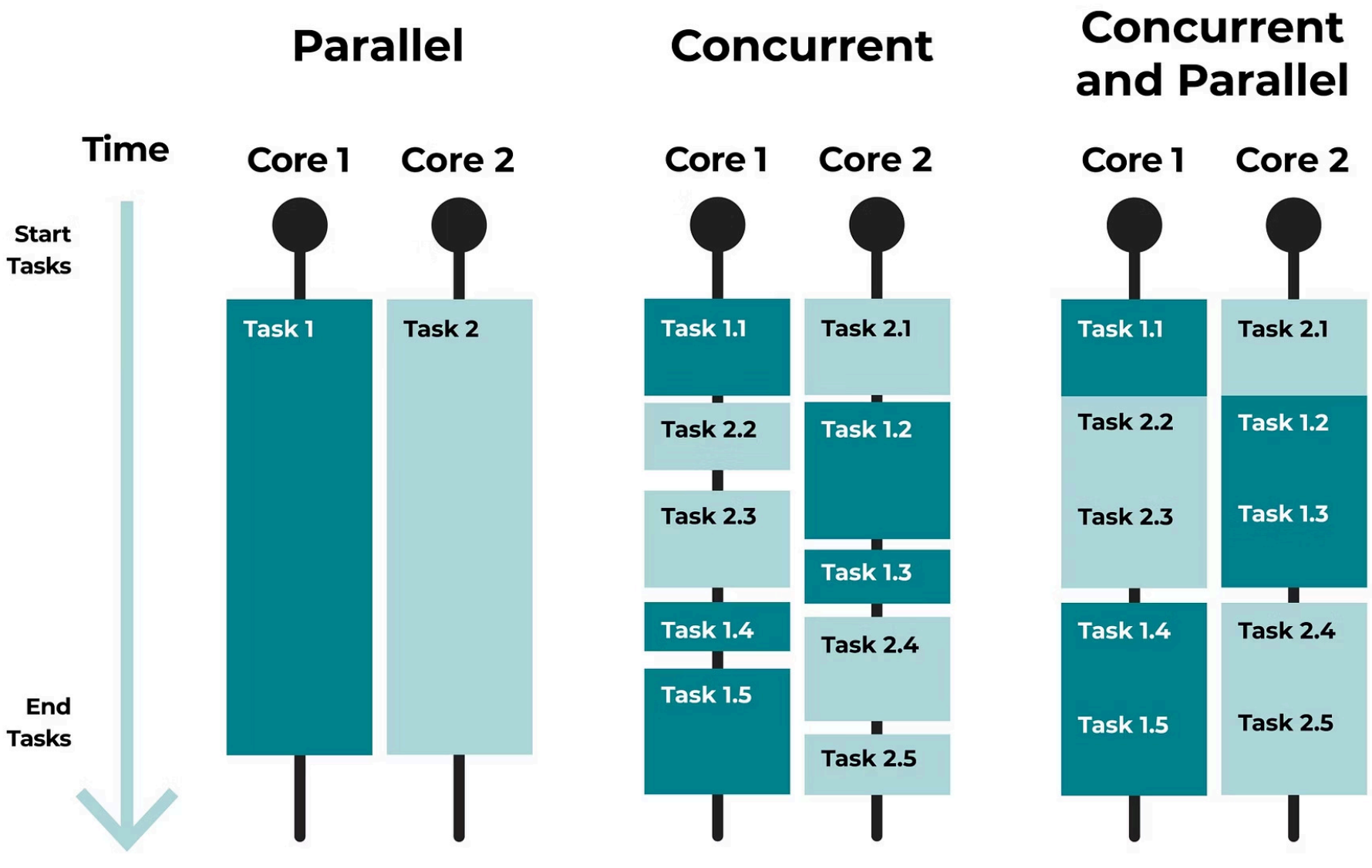


Backends modernos em rust

- Concorrência e paralelismo
- Async await
- Async rust com tokio
- Arquitetura web moderna
- Prática
  - Todo list com axum

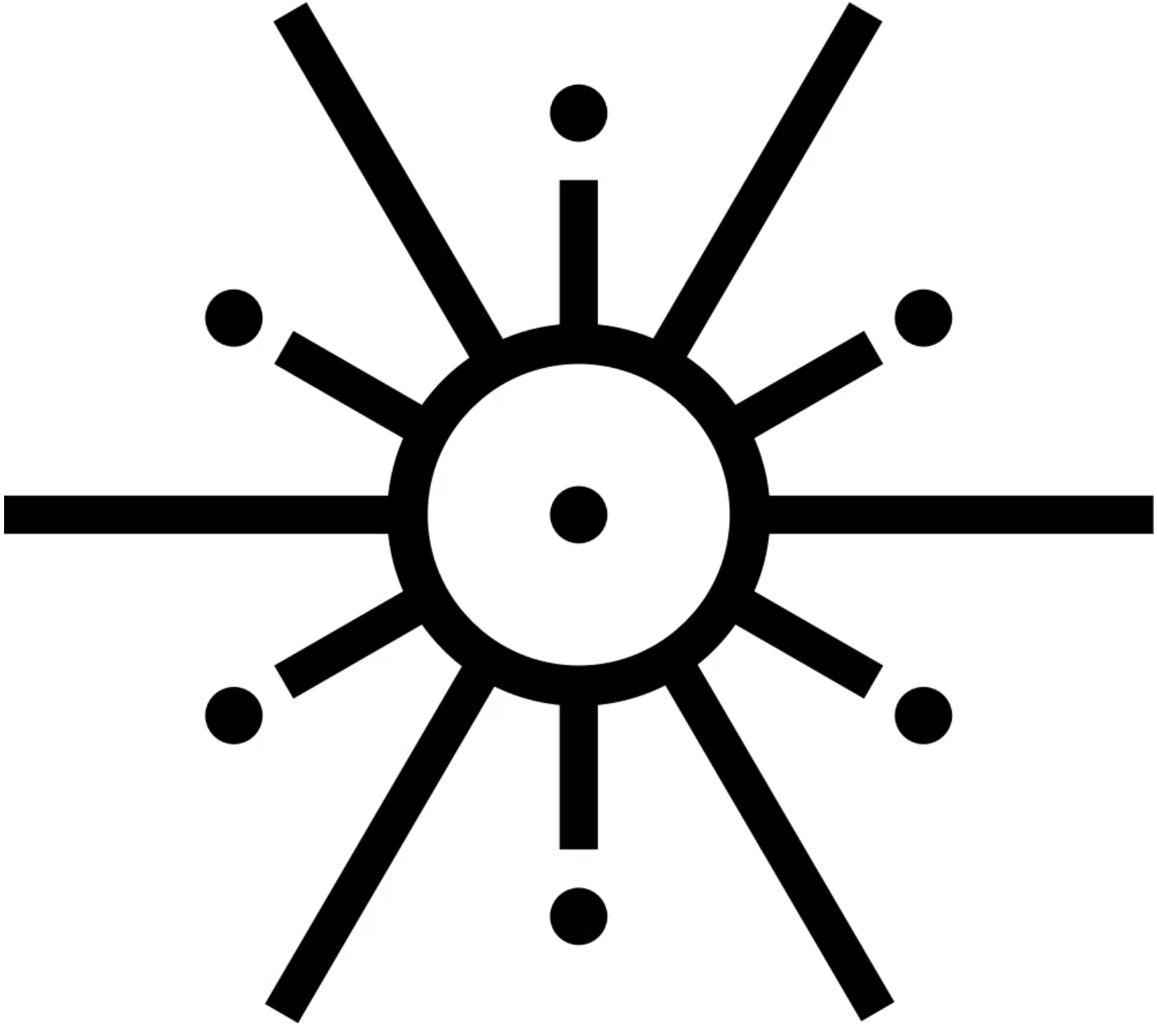
Concorrência e paralelismo



Async Await

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

Async Rust com tokio



O que é o tokio e por que precisamos dele?

```rust
use std::time::Duration;
use tokio::time;

async fn fetch_data_from_db() -> String {
    println!("Starting database query...");
    // Simulando uma chamada de rede
    time::sleep(Duration::from_secs(1)).await;
    "Here is your data.".to_string()
}

#[tokio::main]
async fn main() {
    // Função assincrona que retorna um future
    let data_future = fetch_data_from_db();

    println!("We can do other work here while waiting...");
    let data = data_future.await;

    println!("Received: {}", data);
}
```

```
async_rust> cargo run
    Finished `dev` profile [unoptimized + d
     Running `target/debug/async_rust`
We can do other work here while waiting...
Starting database query...
Received: Here is your data.
async_rust>
```

Atomic Reference Counter (Arc)

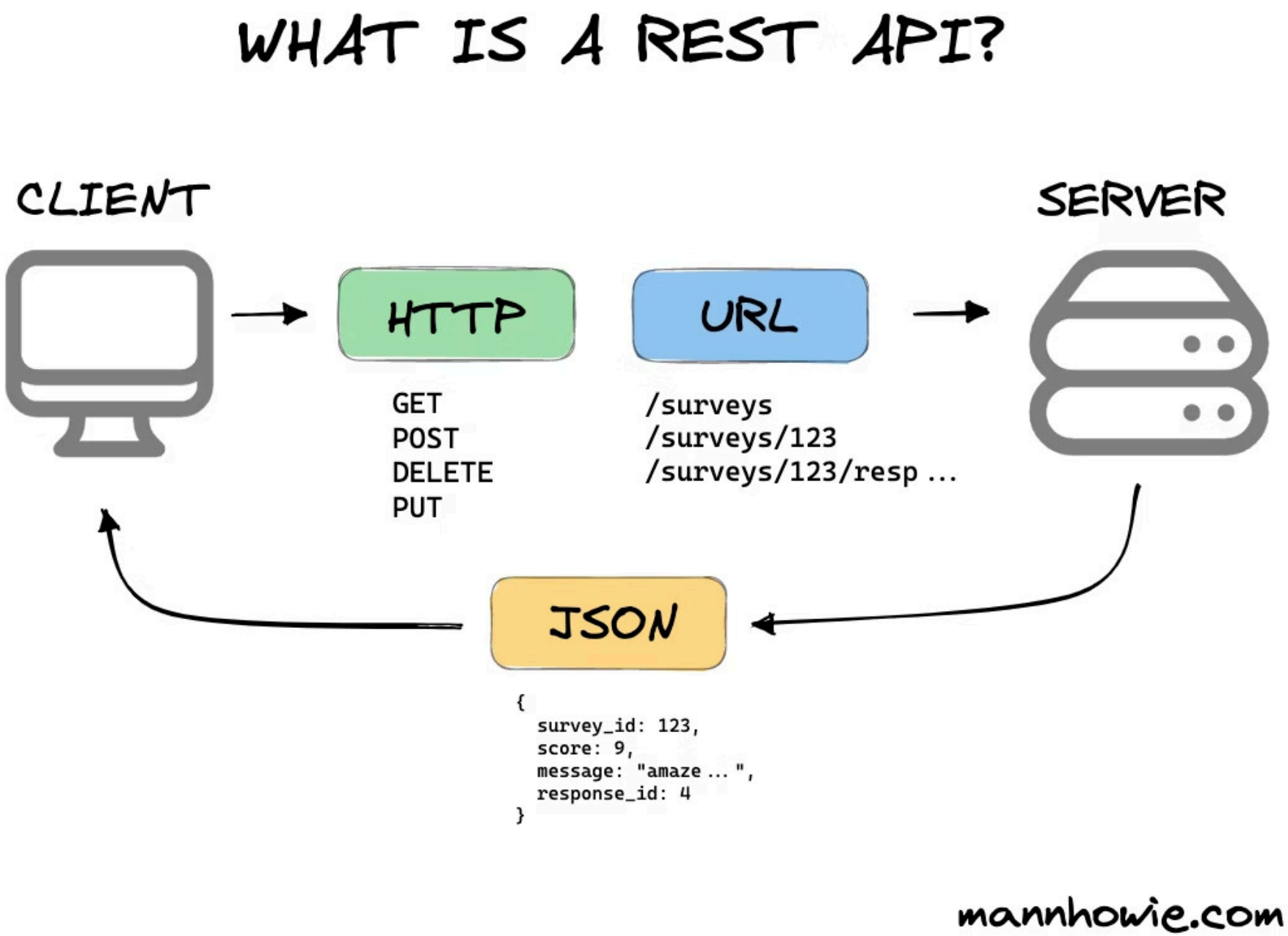
Thread safe shared ownership

Mutual Exclusion (Mutex)

Thread safe mutable access

Arquitetura web moderna

API REST



Axum

Bonus: SQLX