
AWS
re:Invent

OPN205

Next-gen networking infrastructure with Rust and Tokio

Carl Lerche
Principal Software Engineer
AWS

Sean McArthur
Software Development Engineer
AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Build reliable network applications without compromising speed



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Java, Go, or other language with a runtime?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

C or C++?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

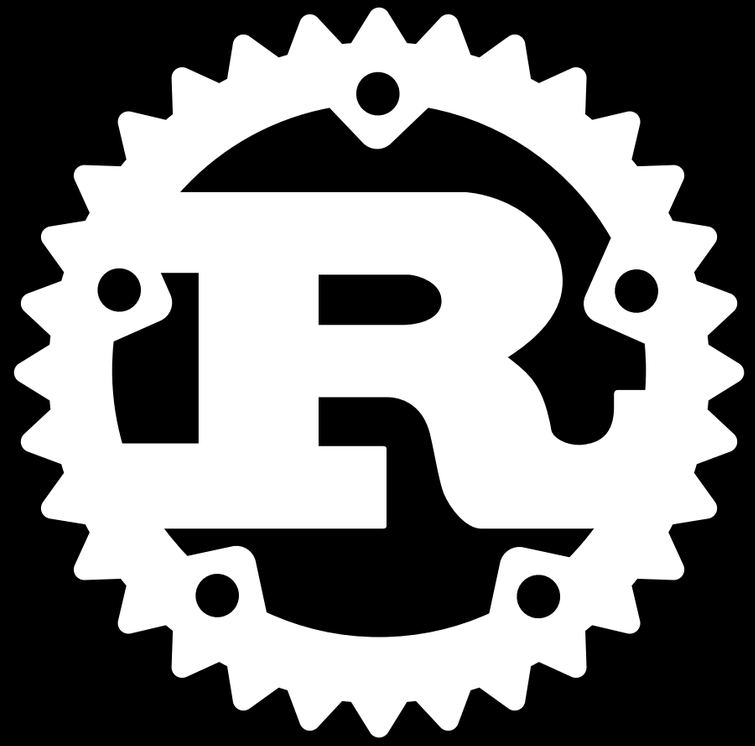
SEGV

```
$ ./carldb
Segmentation fault (core dumped)
```

Safety vs. performance



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Ownership

- Data is owned by a variable
- There can be only one owner at a time
- When the owner goes out of scope, the data is freed

The compiler prevents memory errors

```
fn take_ownership(s: String) {
    println!("{}", s);
    // `s` goes out of scope and the string is
    // freed.
}

let foo = String::from("hello"); // foo comes in scope
take_ownership(foo);             // ownership is passed into the function

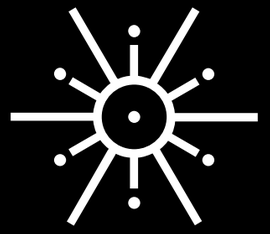

// Trying to use `foo` again does not compile. `foo` no longer owns the
// string. The string is freed inside `take_ownership`, reusing it would
// be a use-after-free bug.
//
// take_ownership(foo);   -> boom
```

Fast, reliable, productive: Pick three



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Tokio

```
#[tokio::main]
async fn main() -> io::Result<()> {
    let listener = TcpListener::bind("localhost:8000").await?;

    loop {
        let (mut socket, _) = listener.accept().await?;

        tokio::spawn(async move {
            // `socket` is moved into the spawned task.
            socket.write_all(b"hello from Tokio").await


            // `socket` goes out of scope, the TCP socket is closed.
        });
    }
}
```

Tokio

- TCP, UDP, Unix sockets, pipes
- Timers
- Channels, mutex, semaphore
- Subprocesses
- Signal handlers
- Multithreaded, work-stealing, task scheduler

async / await

```
async fn hello_world() {
    time::sleep(Duration::from_millis(10)).await;
    println!("done sleeping");
}

println!("starting");
hello_world().await;
println!("done");
```

Compiles to

```
// Enumeration of states
enum MyTask {                       // Enumeration of all states
    Init,                           //    Initial task state
    AwaitSleep(Sleep),              //    Task is waiting for `sleep` to complete
}

fn poll(&mut self, cx: &mut Context) -> Poll<()> {
    match self {
        Init => {                                 // Initial state
            let sleep = time::sleep(ms_10);       //   Register timeout
            sleep.poll(cx);                       //   Advance `sleep`' state machine
            self = AwaitSleep(sleep);             //   Update our state
            return Poll::Pending;                 //   Yield back to Tokio
        }
        AwaitSleep(sleep) => {                    // Tokio called the task again
            return sleep.poll(cx);                //   Advance `sleep` state machine
        }
    }
}
```

Hello World

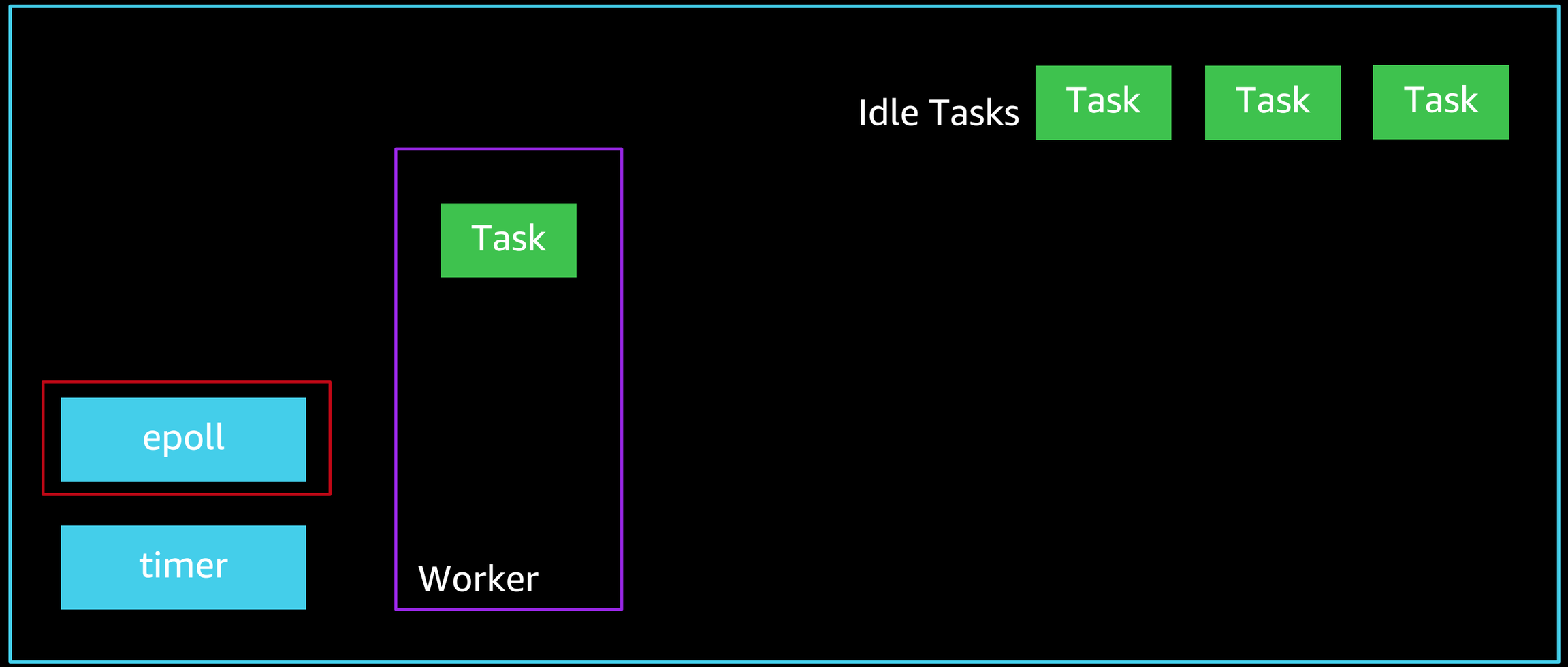
```
#[tokio::main]
async fn main() -> io::Result<()> {
    let listener = TcpListener::bind("localhost:8000").await?;

    loop {
        let (mut socket, _) = listener.accept().await?;

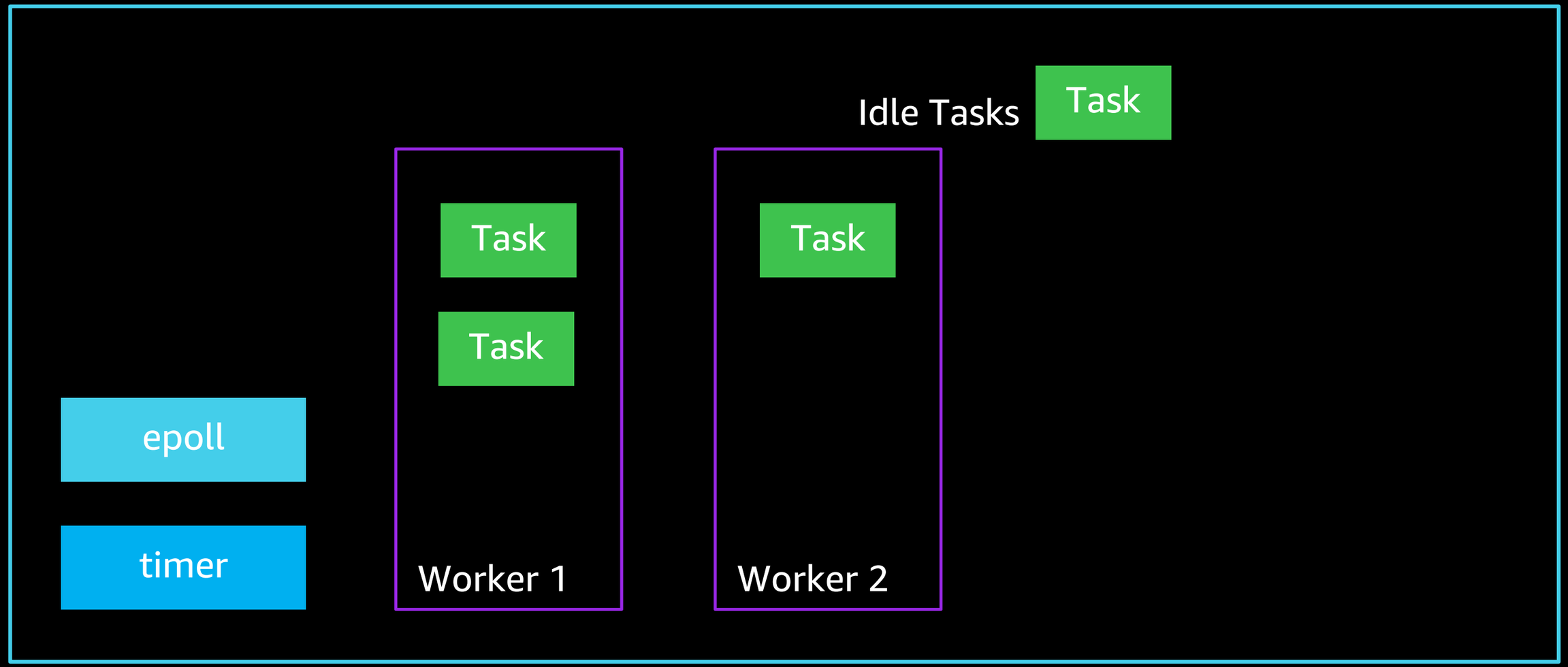
        tokio::spawn(async move {
            // `socket` is moved into the spawned task.
            socket.write_all(b"hello from Tokio").await


            // `socket` goes out of scope, the TCP socket is closed.
        });
    }
}
```

The Tokio runtime



The Tokio runtime



Fearless concurrency



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Sharing data between tasks

```
#[tokio::main]
async fn main() -> io::Result<()> {
    let listener = TcpListener::bind("localhost:8000").await?;
    // Get the message from the arguments
    let msg: String = env::args().nth(1).unwrap();

    loop {
        let (mut socket, _) = listener.accept().await?;

        tokio::spawn(async move {
            socket.write_all(msg.as_bytes()).await
        });
    }
}
```

Sharing data between tasks

```
error[E0382]: use of moved value: `msg`
  --> src/main.rs:15:33
   |
10 |         let msg: String = env::args().nth(1).unwrap();
   |             --- move occurs because `msg` has type
   |                 `std::string::String`, which does not
   |                 implement the `Copy` trait
...
15 |             tokio::spawn(async move {
   |  ________________________________^
16 | |               socket.write_all(msg.as_bytes()).await
   | |                                    --- use occurs due to use in
   | |                                        generator
17 | |             });
   | |_____________^ value moved here, in previous iteration of loop
```

Clone data

```
#[tokio::main]
async fn main() -> io::Result<()> {
    let listener = TcpListener::bind("localhost:8000").await?;
    // Get the message from the arguments
    let msg: String = env::args().nth(1).unwrap();

    loop {
        let (mut socket, _) = listener.accept().await?;
        let msg = msg.clone(); // dup data

        tokio::spawn(async move {
            socket.write_all(msg.as_bytes()).await
        });
    }
}
```

Reliable without sacrificing performance

- Safe
- Misuse resistance APIs
- Compile-time verification
- Fast at runtime
- Batteries included

Tour of mini-redis



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Redis

- In-memory database
- Often used as a shared cache
- Supports many concurrent connections
- Written in C

mini-redis

- Written in Rust, using Tokio
- Implements a subset of Redis
  - GET
  - SET
  - SUBSCRIBE
  - PUBLISH
- For educational purposes

Handling Connections

```
#[tokio::main]
async fn main() -> io::Result<()> {
    let listener = Listener::bind("127.0.0.1:6379").await?;
    // Create our in-memory DB instance
    let database = Db::new();

    // Accept new connections
    loop {
        let (conn, _) = listener.accept().await?;
        let db = database.clone();

        // Spawn a new task per connection
        task::spawn(async move {
            loop {
                let cmd = conn.recv_frame().await?;
                // Apply the command to the database
                let response = cmd.apply(&db);
                conn.write_frame(&response).await?;
            }
        });
    }
}
```

```
struct Db(HashMap<String, Bytes>);
```

```
struct Db(Arc<HashMap<String, Bytes>>);
```

```
struct Db(Arc<Mutex<HashMap<String, Bytes>>>);
```

`std::sync::Mutex`

- RAII
  - Resource Acquisition Is Initialization
- `lock()`
  - Acquires an exclusive read/write lock
  - Returns a `MutexLockGuard` value
- `MutexLockGuard`
  - A “smart pointer”
    - Provides mutable dereferencing to the data inside
  - Destructor releases the lock

```
struct Db(Arc<Mutex<HashMap<String, Bytes>>>);
```

GET

```
fn get(db: &Db, key: String) -> Frame {
    // Lock the db while we use it
    let locked = db.lock();

    // Get the value from the shared database state
    if let Some(value) = locked.find(&key) {
        // If a value is present, it is
        // written to the client in "bulk“ format.
        Frame::Bulk(value)
    } else {
        // If there is no value, `Null` is written.
        Frame::Null
    }
}
```

SET

```
fn set(db: &Db, key: String, value: Bytes) -> Frame {
    // Lock the db while we use it
    let locked = db.lock();

    // Set the value in the shared database state.
    locked.insert(key, value);

    Frame::Simple("OK")
}
```

Benchmarks

	redis	mini-redis
SET	140845 reqs/s	140252 reqs/s
GET	141043 reqs/s	141643 reqs/s

Rust. Tokio.
Fast. Reliable.



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

tokio.rs/tokio/tutorial



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Carl Lerche

pdxcarl@amazon.com
Github: @carllerche
Twitter: @carllerche

Sean McArthur

@seanmonstar

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.