LESSON 11

main.rs

```rust
let c = Arc::new(Mutex::new(0));
thread::spawn(move || {
  *c.lock().unwrap() += 1;
});
```

Rust

Fearless Concurrency

Learn it all @MyWebUniversity



A FREE MYWEBUNIVERSITY COURSE

RUST PROGRAMMING

LESSON 11

Smart Pointers and Fearless Concurrency

Box, Rc, RefCell, threads, move closures, channels, Arc and Mutex

Box, Rc and RefCell

Type	Gives you	Threads?
`Box<T>`	heap, one owner	✓
`Rc<T>`	shared owners	✗
`RefCell<T>`	mutate via &	✗
`Arc<T>`	shared, atomic	✓
`Mutex<T>`	locked mutation	✓

- Box puts a value on the heap with one owner, needed for recursive types like trees. Rc lets …
- RefCell allows changing data behind a shared reference, with the borrow rules checked at …

01

Spawning Threads

main.rs

```rust
use std::thread;

let files = vec!["a.log", "b.log", "c.log"];
let handles: Vec<_> = files.into_iter().map(|f| {
    thread::spawn(move || count_lines(f))
}).collect();

let total: usize = handles.into_iter().map(|h| h.join().unwrap()).sum();
```

- thread spawn runs a closure on a new operating system thread. The handle's join waits for …
- The closure needs move to own the data it uses, because the thread may outlive the current …

02

Channels

main.rs

```rust
use std::sync::mpsc;

let (tx, rx) = mpsc::channel();
for id in 0..3 {
    let tx = tx.clone();
    thread::spawn(move || tx.send(format!("worker {id} done")).unwrap());
}
drop(tx);
for msg in rx { println!("{msg}"); }
```

- mpsc channels send values between threads: many producers, single consumer. The sender …
- Sending a value moves it, so two threads can never use it at the same time.

03

Arc and Mutex

main.rs

```rust
use std::sync::{Arc, Mutex};

let counter = Arc::new(Mutex::new(0));
let mut hs = vec![];
for _ in 0..8 {
    let c = Arc::clone(&counter);
    hs.push(thread::spawn(move || { *c.lock().unwrap() += 1; }));
}
for h in hs { h.join().unwrap(); }
println!("{}", *counter.lock().unwrap());    // 8
```

- To share changeable state between threads, wrap it in Arc of Mutex. Arc shares ownership …
- lock returns a guard, and the lock is released automatically when the guard goes out of scope.

04

Why It Is Fearless

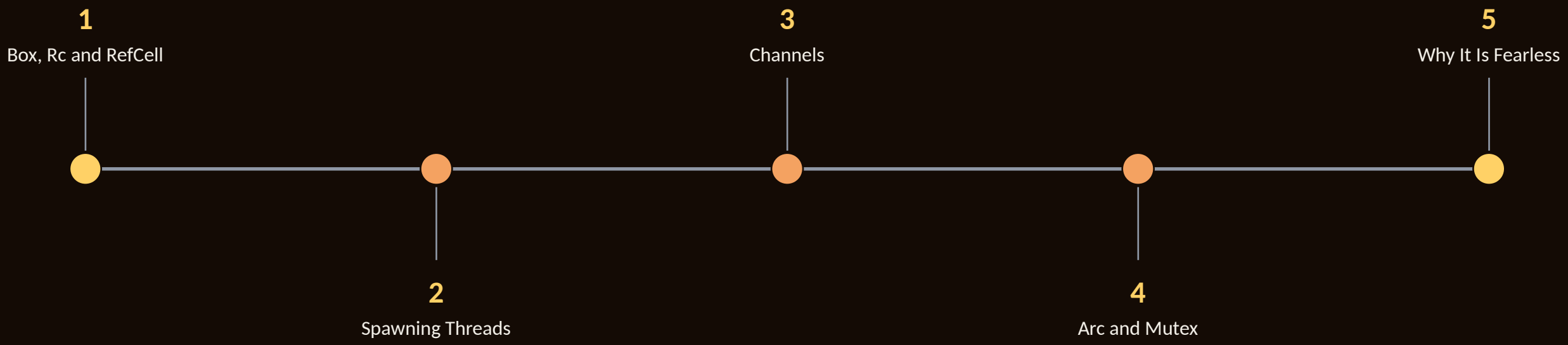
terminal

```
error[E0277]: `Rc<i32>` cannot be sent between threads safely
   |
8  |     thread::spawn(move || println!("{}", rc));
   |     ^^^^^^^^^^^^^ `Rc<i32>` cannot be sent between threads safely
   = help: use `Arc<i32>` instead
```

- The Send and Sync traits mark which types are safe to move or share between threads. Rc is …
- Data races become compile errors, not bugs you hunt for at three in the morning. For async …

05

Lesson Map


1
Box, Rc and RefCell
2
Spawning Threads
3
Channels
4
Arc and Mutex
5
Why It Is Fearless

Sources and Further Reading

Type	Gives you	Threads?
`Box<T>`	heap, one owner	✓
`Rc<T>`	shared owners	✗
`RefCell<T>`	mutate via &	✗
`Arc<T>`	shared, atomic	✓
`Mutex<T>`	locked mutation	✓

RUST PROGRAMMING · 11

main.rs

```rust
use std::sync::mpsc;

let (tx, rx) = mpsc::channel();
for id in 0..3 {
    let tx = tx.clone();
    thread::spawn(move || tx.send(format!("worker {id} done")).unwrap());
}
drop(tx);
for msg in rx { println!("{msg}"); }
```

Thank you for watching

MyWebUniversity.com

Free technical education since 2000 — English and Dari

@MyWebUniversity · @MyWebUniversity-Dari