Concurrency testing with Loom

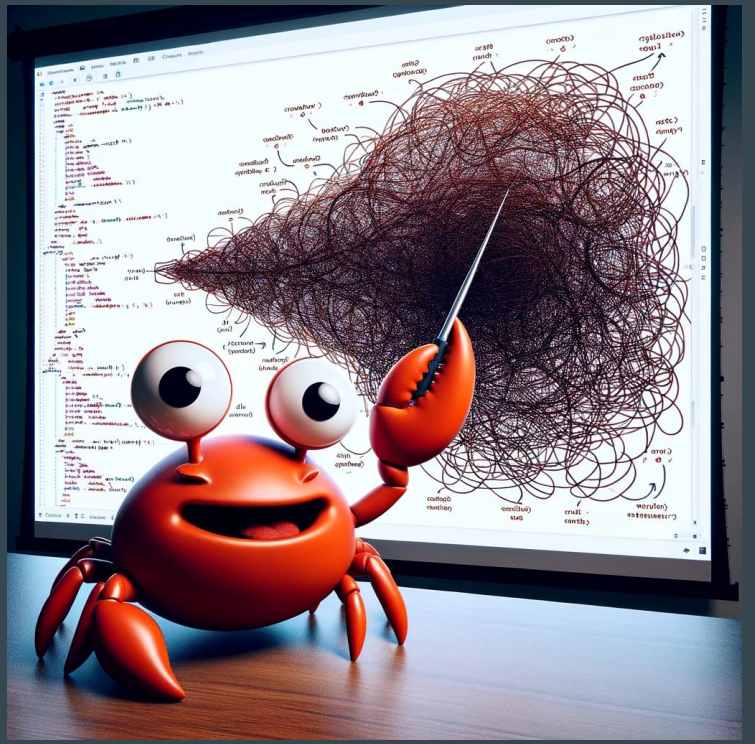
Anton Velikanov

whoami

- using Rust in production since 2018
- studying databases internals in practice
- previously security researcher (audits/bug hunting/ctfs participant)
- ❤️ Linux & OpenSource:
  - https://github.com/bondifuzz
  - I use  btw
  - ...

Agenda

1. Fearless concurrency in Rust
2. Concurrency fundamentals recap
3. Example of tricky concurrent code
4. What is Loom? How to use?
5. Loom usage example
6. Loom downsides
7. Conclusion & references



Fearless concurrency

- Send
- Sync
- std::sync::{Mutex, RwLock, Arc, mpsc, atomic}

Concurrency fundamentals recap

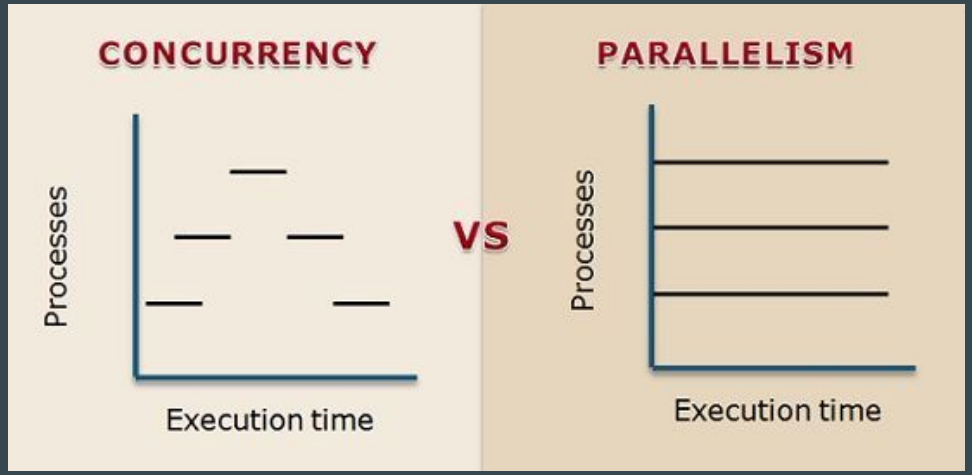
- why ?
- concurrency != parallelism
- compiler reordering
- hardware reordering

Concurrency fundamentals recap

- why ?

Concurrency fundamentals recap

- why ?
- concurrency != parallelism
  - single-thread concurrency
  - single-core multithreaded concurrency
  - multicore concurrency



Concurrency fundamentals recap

- why ?
- concurrency != parallelism
- compiler reordering:
  - result stays the same
  - does not take other threads into account
  - require ordering for operations with atomics

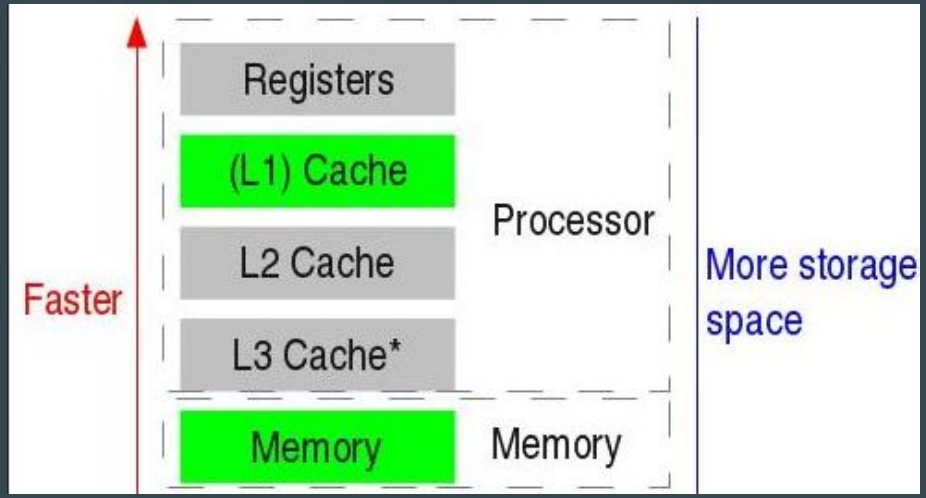
```
fn f(a: &mut i32, b: &mut i32)
{
    *a += 1;
    *b += 1;
    *a += 1;
}
```



```
fn f(a: &mut i32, b: &mut i32)
{
    *a += 2;
    *b += 1;
}
```

Concurrency fundamentals recap

- why ?
- concurrency != parallelism
- compiler reordering
- hardware reordering:
  - CPU registers
  - CPU caches
  - RAM



Concurrency fundamentals recap

- memory model
  - architecture agnostic
  - happens-before relationship
  - ignore machine instructions, caches, buffers, timing, instruction reordering, compiler optimizations, etc...
- Rust memory model copied from C++20 (almost)

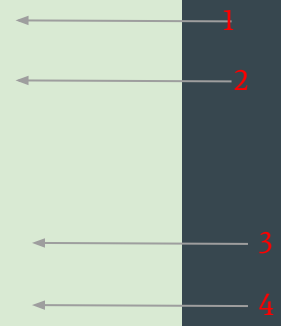
```
pub enum Ordering {
    Relaxed,
    Release,
    Acquire,
    AcqRel,
    SeqCst,
}
```

How reordering looks like?

```
static X: AtomicBool = AtomicBool::new(false);

static Y: AtomicBool = AtomicBool::new(false);


let t1 = spawn(|| {
    let r1 = Y.load(Ordering::Relaxed);
    X.store(r1, Ordering::Relaxed);
});
let t2 = spawn(|| {
    let r2 = X.load(Ordering::Relaxed);
    Y.store(true, Ordering::Relaxed);
});
```



r2 == true ?

How reordering looks like?

```rust
static X: AtomicBool = AtomicBool::new(false);

static Y: AtomicBool = AtomicBool::new(false);


let t1 = spawn(|| {
    let r1 = Y.load(Ordering::Relaxed);
    X.store(r1, Ordering::Relaxed);
});
let t2 = spawn(|| {
    let r2 = X.load(Ordering::Relaxed);
    Y.store(true, Ordering::Relaxed);
});
```

1

2

3

4

r2 == true ?

Valid execution order: 4 1 2 3

What is Loom?

- concurrency testing tool
- part of Tokio project
- model based testing
- track of all cross-thread interactions
- provides replacement for std::sync::*
  and std::thread::*

How to use Loom?

```rust
use std::sync::Arc;
use std::sync::atomic::AtomicUsize;
use std::sync::atomic::Ordering::SeqCst;
use std::thread;

#[test]
fn test_concurrent_logic() {
    let v1 = Arc::new(AtomicUsize::new(0));
    let v2 = v1.clone();

    thread::spawn(move || {
        v1.store(1, SeqCst);
    });

    assert_eq!(0, v2.load(SeqCst));
}
```

```rust
use loom::sync::atomic::AtomicUsize;
use loom::thread;

use std::sync::Arc;
use std::sync::atomic::Ordering::SeqCst;

#[test]
fn test_concurrent_logic() {
    loom::model(|| {
        let v1 =
Arc::new(AtomicUsize::new(0));
        let v2 = v1.clone();

        thread::spawn(move || {
            v1.store(1, SeqCst);
        });

        assert_eq!(0, v2.load(SeqCst));
    });
}
```