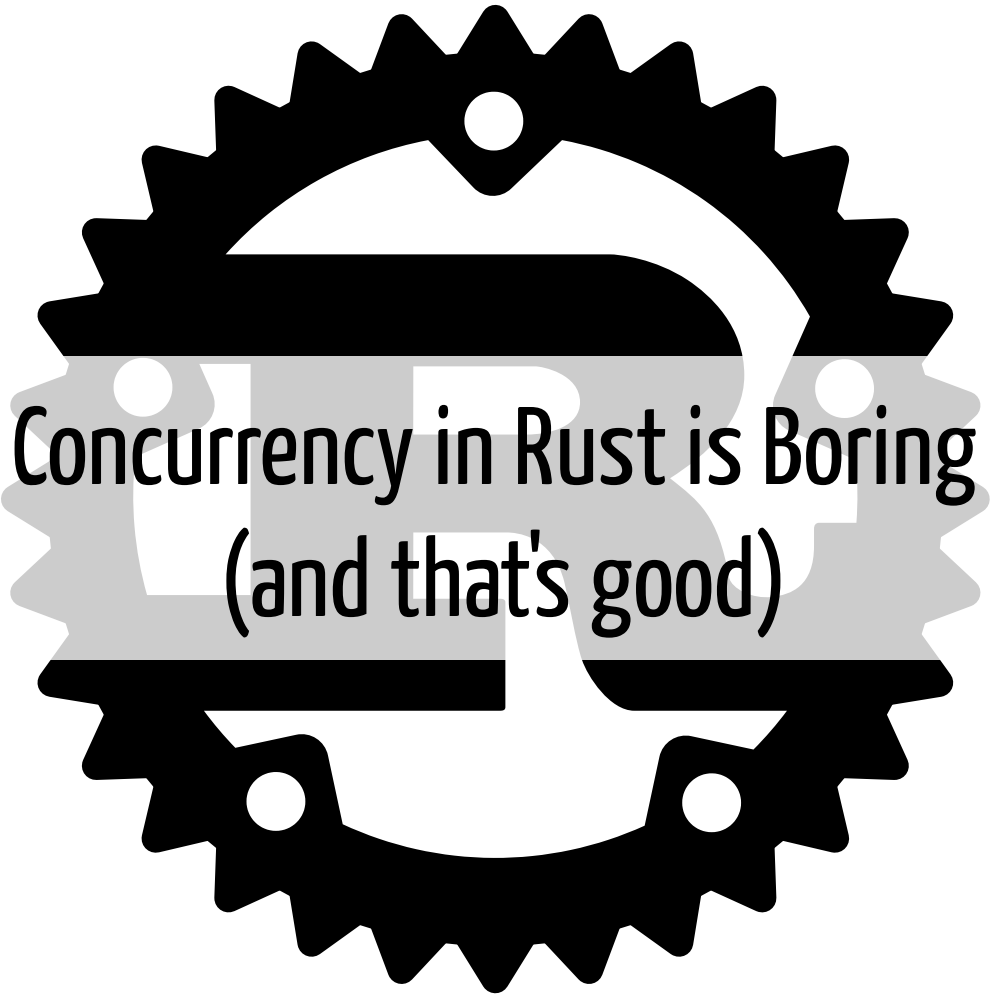
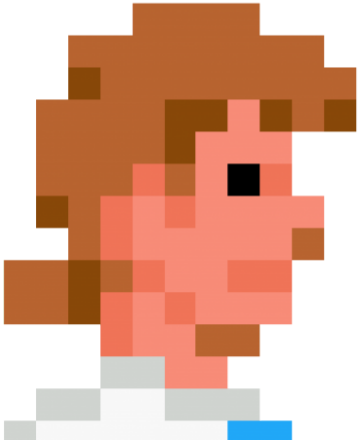

Concurrency in Rust is Boring
(and that's good)



Software Engineer at Tuenti

MadRust co-organizer

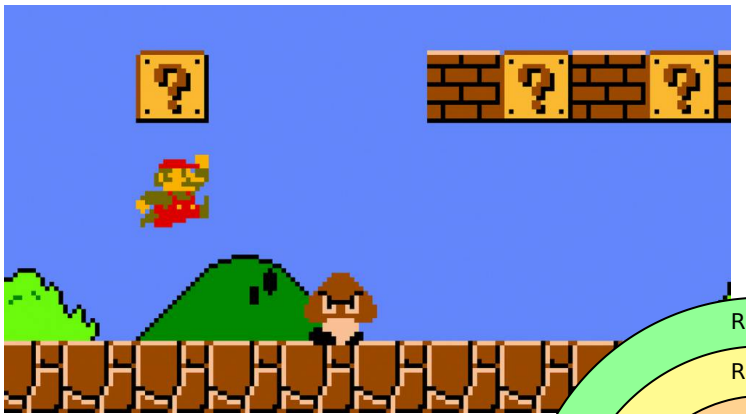
jrvidal

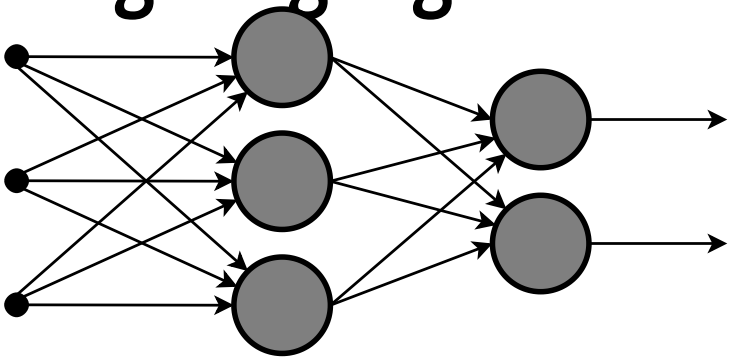
_rvidal

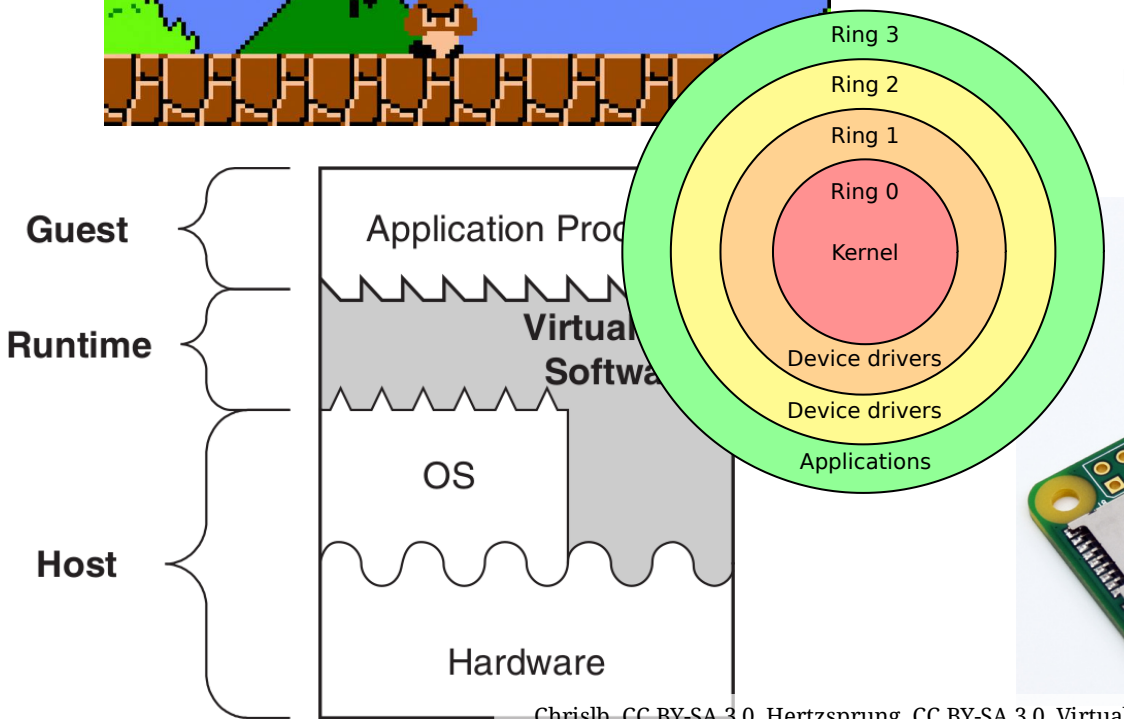

I HAVE NO
IDEA WHAT
I'M DOING

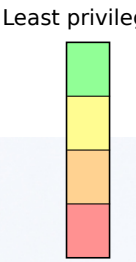
Concurrency: the End is Near

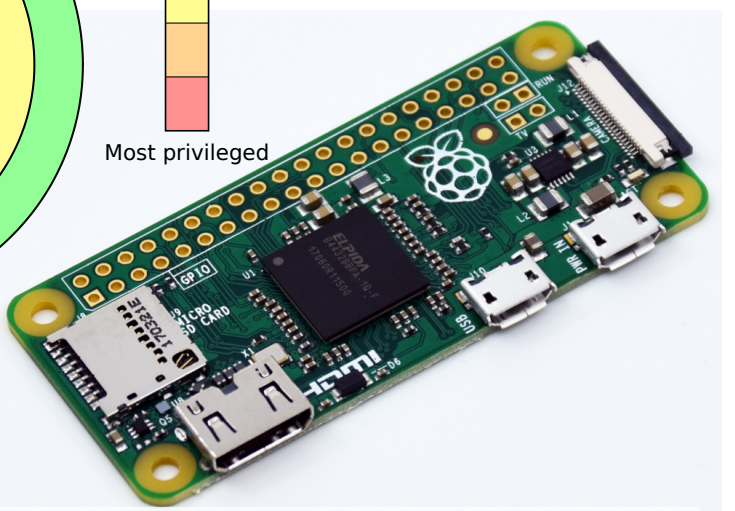
Rust is a Systems Programming Language











...and more

```
fn dot_product(
  vec1: &[u32],
  vec2: &[u32]
) -> u32 {
  vec1
    .iter()
    .zip(vec2)
    .map(|(a, b)| a * b)
    .sum()
}
```

Concurrency in Rust is *boring*

```
use std::{thread, sync};

let arc = sync::Arc::new("foo");
let mutex = sync::Mutex::new(1);

thread::spawn(|| {
  println!("Look ma, concurrency")
});

let (sender, receiver) = sync::mpsc::channel();
```

Concurrency in Rust is *boring* (luckily)

```
use std::{thread, sync};

let arc = sync::Arc::new("foo");
let mutex = sync::Mutex::new(1);

thread::spawn(|| {
  println!("Look ma, concurrency")
});

let (sender, receiver) = sync::mpsc::channel();
```

NO DATA RACES

Ownership and Borrowing

Single owner

```
let x = Box::new(7);

consume(x);

println!("x = {}", x);
```

Ownership and Borrowing

Single owner

```
let x = Box::new(7);

consume(x);
// ⚠ borrow of moved value: `x`
println!("x = {}", x);
```

Ownership and Borrowing

Borrowing prevents moves

```
let x = Box::new(7);

let pointer = &x;


consume(x);

println!("pointer = {}", pointer);
```

Ownership and Borrowing

Borrowing prevents moves

```
let x = Box::new(7);

let pointer = &x;
// ⚠ cannot move out of `x`
// because it is borrowed
consume(x);

println!("pointer = {}", pointer);
```

Ownership and Borrowing

Multiple read-only borrows *or* single mutable borrow

```
let mut x = 7;

let pointer = &x;


let another_pointer = &mut x;

*another_pointer += 1;

println!("pointer = {}", pointer);
```

Ownership and Borrowing

Multiple read-only borrows *or* single mutable borrow

```
let mut x = 7;

let pointer = &x;
// ⚠ cannot borrow `x` as mutable
// because it is also borrowed as immutable
let another_pointer = &mut x;

*another_pointer += 1;

println!("pointer = {}", pointer);
```

Ownership and Borrowing *and* Concurrency

```
let mut x = 1;

let pointer = &mut x;

thread::spawn(|| {
  *pointer = 7;
});


thread::spawn(|| {
  println!("x = {}", x);
});
```

Ownership and Borrowing *and* Concurrency

```
let mut x = 1;

let pointer = &mut x;

thread::spawn(|| {
  *pointer = 7;
});
// ⚠ cannot borrow `x` as immutable
// because it is also borrowed as mutable
thread::spawn(|| {
  println!("x = {}", x);
});
```

Send and Sync

For everything else, the type system.

```
let original = Rc::new(1);
let copy = original.clone();


thread::spawn(|| {
  println!("original = {}", original)
});

let copy2 = copy.clone();
```

Send and Sync

For everything else, the type system.

```
let original = Rc::new(1);
let copy = original.clone();
// ⚠ `Rc<i32>` cannot be shared
// between threads safely
thread::spawn(|| {
  println!("original = {}", original)
});

let copy2 = copy.clone();
```

```
pub fn spawn<F>(f: F) where F: FnOnce + Send
```

Userland tour

Userland tour

- `rayon`: data parallelism
- `crossbeam`: concurrent data structures
- `lazy_static`: lazy-initialized static values
- `parking_lot`: better impls of std primitives
- `actix`: Actors
- `threadpool`: basic threadpool

What about the future?

What about the future?

```
async fn get_user(uid: u32) -> Option<User> {
  let db_user = db.get_user(uid).await;

  let metadata = external_service
    .get_metadata(uid).await;

  db_user.map(|u| User::from(u, metadata))
}
```

What about the future?

Withoutblogs

Tempus fugit

- 1st class documentation
- Error messages
- cargo & crates.io
- Enums for error handling
- Servo
- WebAssembly
- Serde
- Zero-cost closures
- Community 💞
- Redox
- libcore and embedded
- Editions, release channels
- Macros: declarative and procedural

- https://www.meetup.com/MadRust/
- https://www.rust-lang.org/learn


Concurrency,
Do Not Fear