A functional view to Rust


WAIT,
WHAT?


Q&A
feel free to interrupt
Thanks
we're hiring
redhat.
For attending
Still here
@FLAPER87
FLAVIO@REDHAT.COM

What's up with this Rust-lang thingy?

Does it really buy me something?

What does it have to do with functional languages?

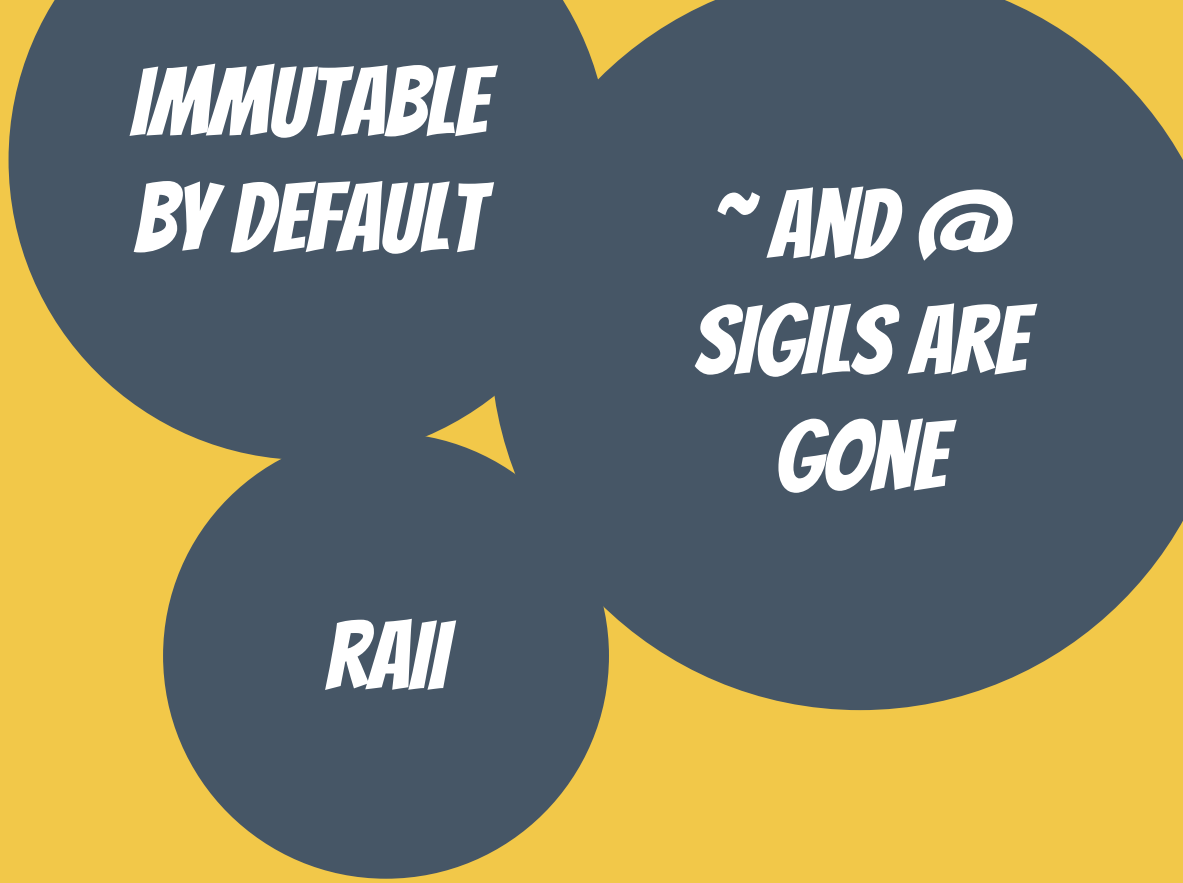
~ AND @ SIGILS ARE GONE

BORING STUFF

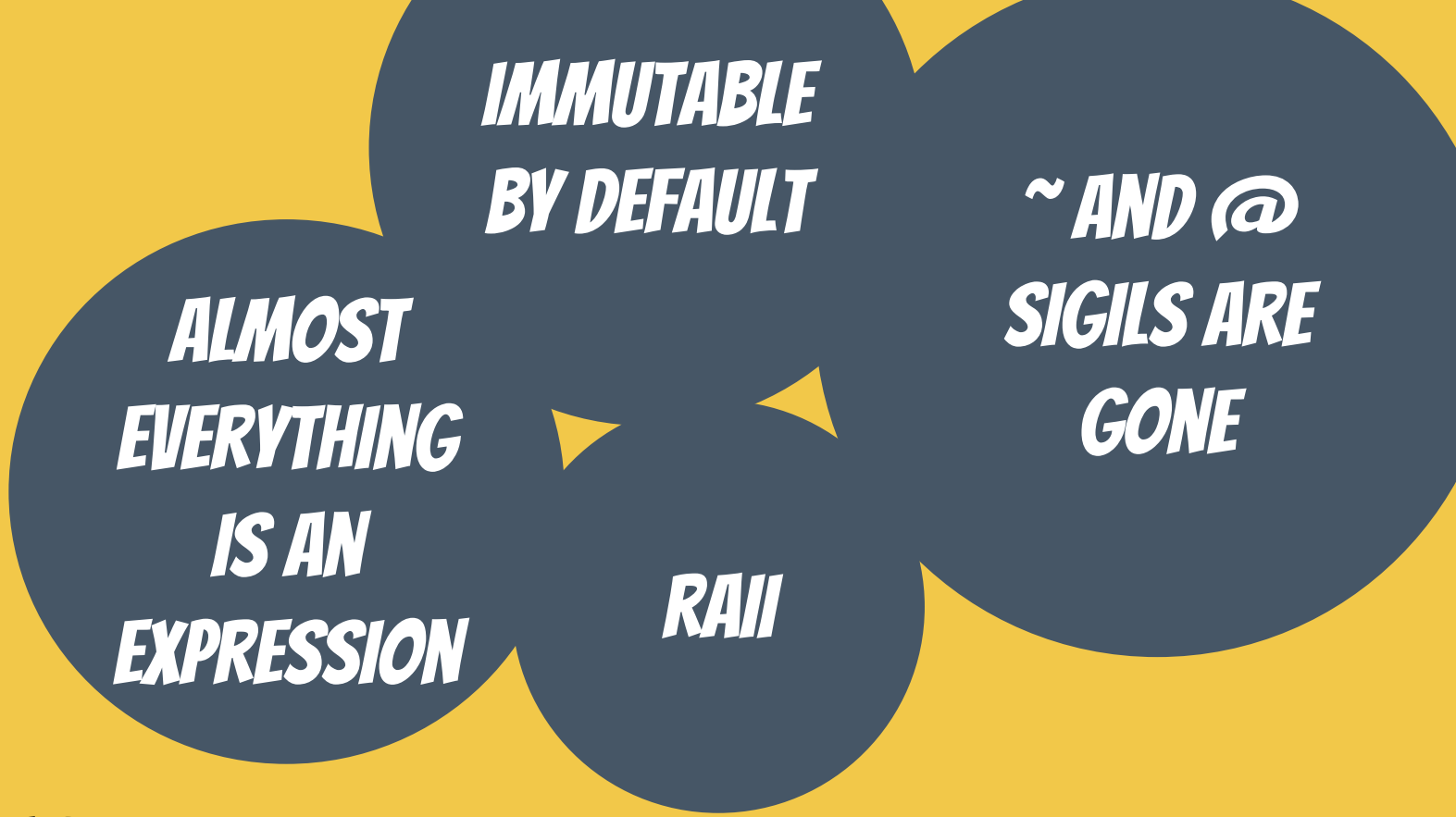
IMMUTABLE BY DEFAULT

~ AND @ SIGILS ARE GONE

BORING STUFF

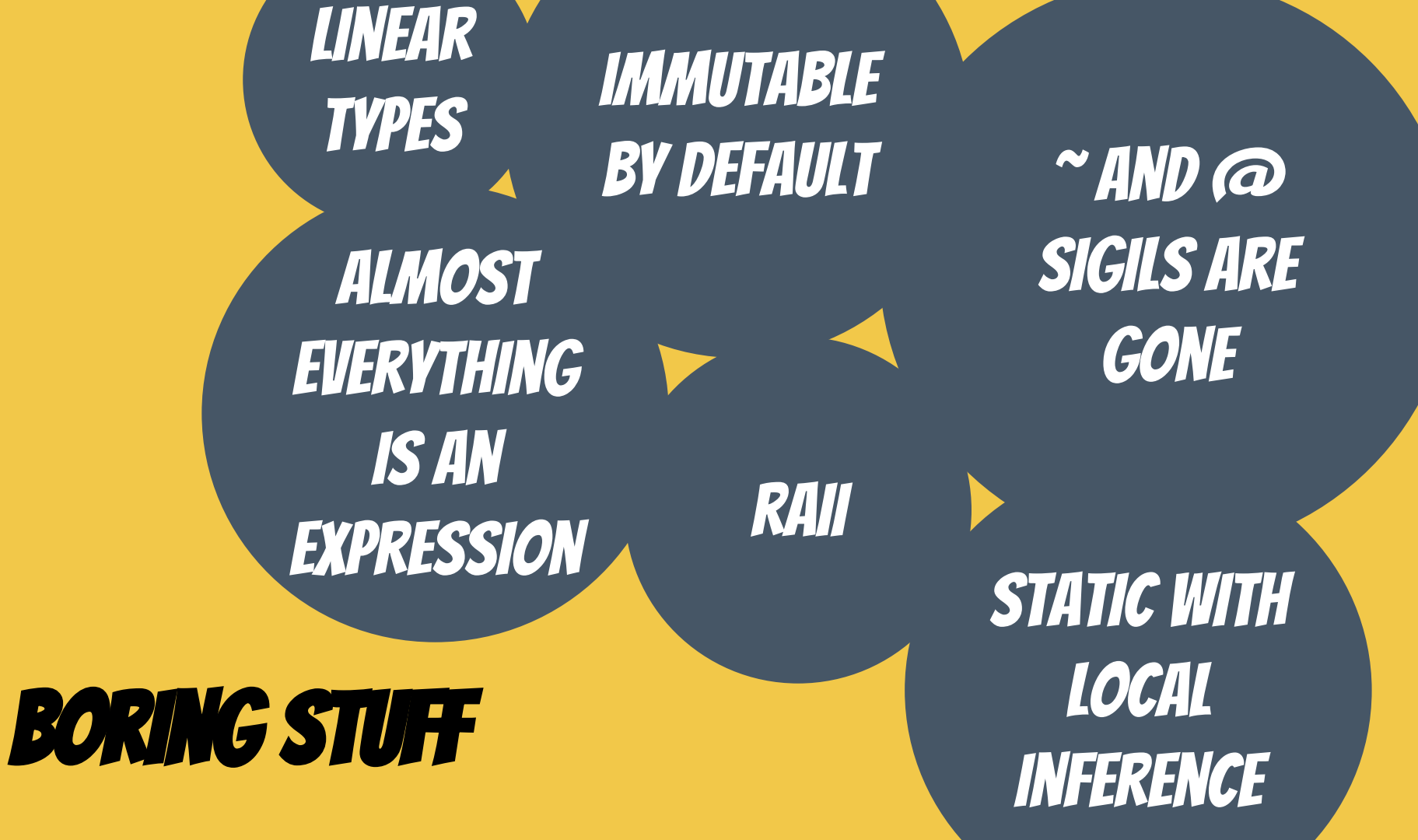


BORING STUFF



BORING STUFF


IMMUTABLE BY DEFAULT
~ AND @ SIGILS ARE GONE
ALMOST EVERYTHING IS AN EXPRESSION
RAII
STATIC WITH LOCAL INFERENCE
BORING STUFF


LINEAR TYPES
IMMUTABLE BY DEFAULT
~ AND @ SIGILS ARE GONE
ALMOST EVERYTHING IS AN EXPRESSION
RAII
STATIC WITH LOCAL INFERENCE
BORING STUFF


LINEAR TYPES
IMMUTABLE BY DEFAULT
~ AND @ SIGILS ARE GONE
...
ALMOST EVERYTHING IS AN EXPRESSION
RAII
STATIC WITH LOCAL INFERENCE
BORING STUFF


LINEAR TYPES
IMMUTABLE BY DEFAULT
~ AND @ SIGILS ARE GONE
...
ALMOST EVERYTHING IS AN EXPRESSION
RAII
...
STATIC WITH LOCAL INFERENCE
BORING STUFF

Memory Safety

```rust
fn main() {
    let mut x = 5;

    add_one(&mut x);

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

fn add_one(num: &mut i32) {
    *num += 1;
}
```

passing by-ref "lends" the value

```rust
fn main() {
    let x = Box::new(5);

    add_one(x);

    println!("{}", x); // error
}

fn add_one(mut num: Box<i32>) {
    *num += 1;
}
```

passing by-val "gives" the value

```rust
fn main() {
    let x = 5;

    add_one(x);

    println!("{}", x); // No error
}

fn add_one(num:i32) -> i32 {
    num + 1
}
```

wait, you said no-ref moves the value

Type & Trait system

Bounded Polymorphism

Bounds

T: U
T: Send

T: Sync

T: Sync+Copy

```
struct MyType<T, &> {...}
```

```
struct MyType<T: Copy> {...}
```

```
fn send<T: Show+Copy> {...}
```

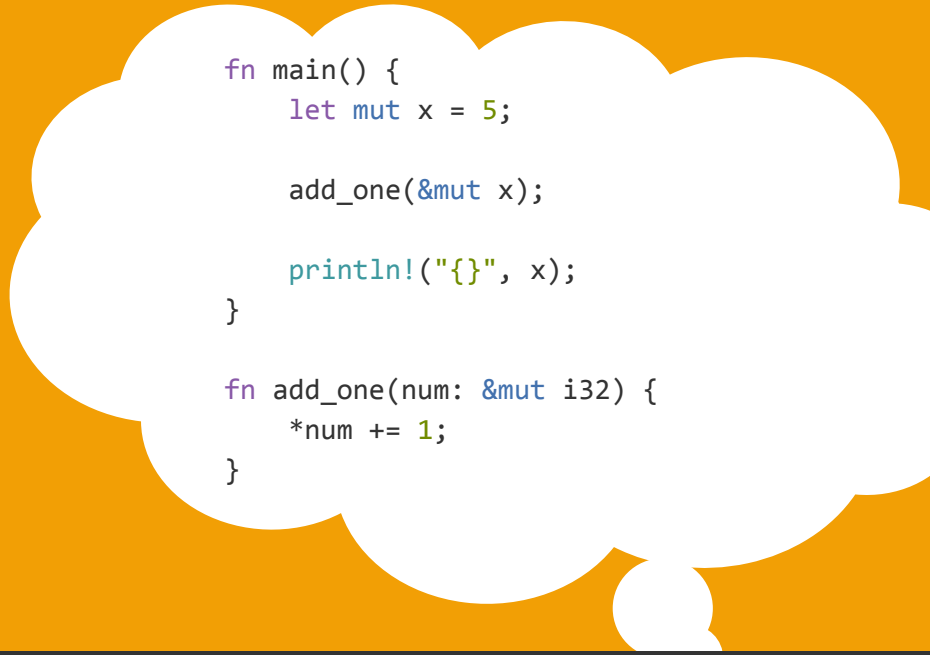
Lifetimes

```
struct Foo<'a> {
    x: &'a i32,
}

fn main() {
    let x;                      // -+ x goes into scope
                                //  |
    {                           //  |
        let y = &5;             // ---+ y goes into scope
        let f = Foo { x: y };   // ---+ f goes into scope
        x = &f.x;               //  | | error here
    }                           // ---+ f and y go out of scope
                                //  |
    println!("{}", x);          //  |
}
```

lifetimes, regions, scope...

Mutability



Is mutability limited to linearity?

Inherited Mutability

```
fn main() {

    let x = MyStruct {a: 5};
    let mut z = x; // z.a is mutable

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

Internal Mutability

```
fn main() {

    let x = Cell(5);

    x.set(6);
}
```

Soundness

```
struct Foo<'a> {
    x: &'a mut i32, // ERROR: &mut T is not copyable
}

impl<’a> Copy for Foo {}

fn is_copy<T: Copy>() {}


fn main() {

    is_copy::<Foo>();
}
```

unaliased mutable references

```
struct Foo<'a> {
    x: &'a i32,
}

impl<’a> Copy for Foo {}

fn is_copy<T: Copy>() {}


fn main() {

    is_copy::<Foo>();
}
```

Enough with the type system

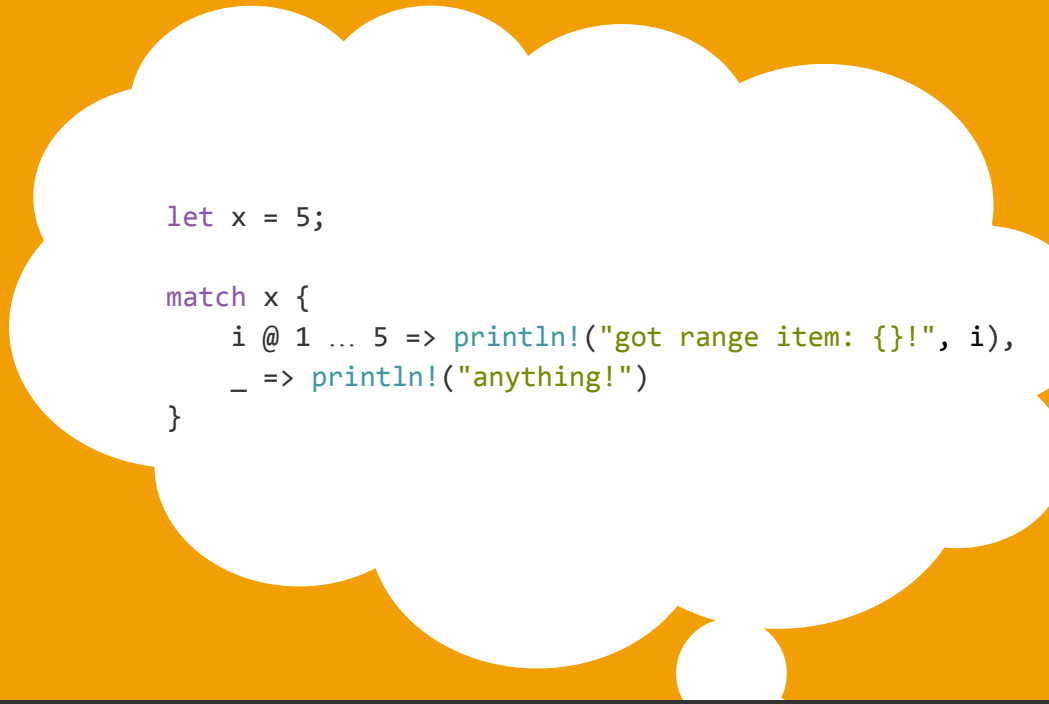
Pattern Matching

Pattern Matching

```
enum OptInt {
    Value(i32),
    Missing, // Warning on unused variant
}

match OptInt::Value(5) {

    OptInt::Value(a) if a > 5 => println!("{}", a),
    OptInt::Value(a) => println!("Got an int: {}!",
a),
    OptInt::Missing => println!("No such luck."),
}
```



Pattern Matching

Iterators

Iterator trait: fold

```
let sum = (1..4).fold(0, |sum, x| sum + x);
```

```
for i in (1..100).filter(|&x| x % 2 == 0) {
    println!("{}", i);
}
```

Iterator trait: filter

Iterator trait: map

```
for i in (1..100).map(|x| x + 1) {
    println!("{}", i);
}
```

Thread, Mutex, Channel

Channels

Multi-producer
Single-consumer

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

fn main() {
    let data = Arc::new(Mutex::new(0u32));

    let (tx, rx) = mpsc::channel();

    for _ in 0..10 {
        let (data, tx) = (data.clone(), tx.clone());

        thread::spawn(move || {
            let mut data = data.lock().unwrap();
            *data += 1;

            tx.send(());
        });
    }

    for _ in 0..10 {
        rx.recv();
    }
}
```


QUESTIONS?