

Rust @ Code4Fun

Stefan Kaestle

June 18, 2019



About me



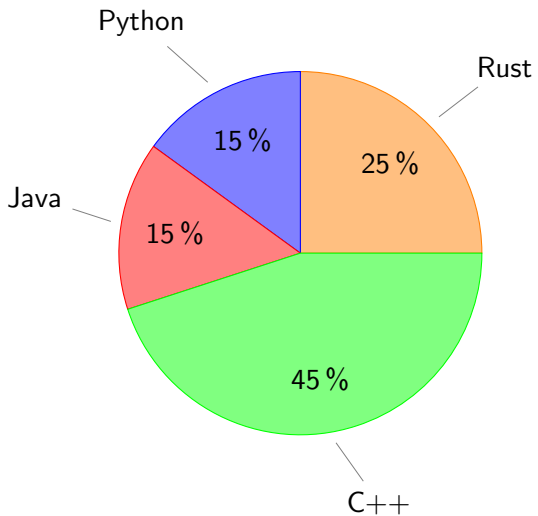
ETH zürich

- ▶ Stefan
- ▶ Since 2016: Oracle Labs, Distributed Graph Processing
- ▶ Background: ETH Systems Group, Barrelfish OS
- ▶ Rust since ~9 months
 - ▶ So absolutely no expert :-)

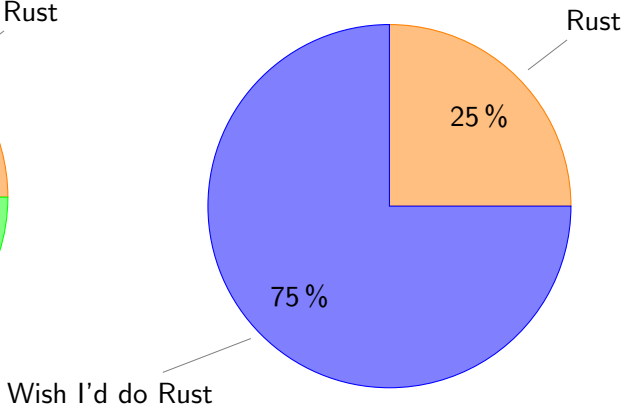
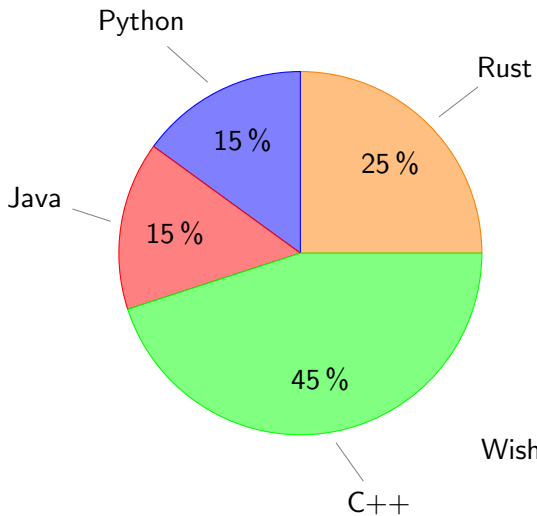


ORACLE®

About me: Programming languages



About me: Programming languages





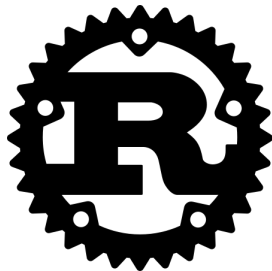
Section 1

Intro to Rust

Short history of Rust



- ▶ Developed by Mozilla Research
- ▶ Goal: better memory safety, but retain high-performance
- ▶ Announced in 2010
- ▶ 2011: First successfully compiled `rustc`

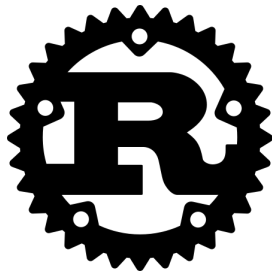


Short history of Rust



Ohh, and there's this little fella
Meet ferries, the crab

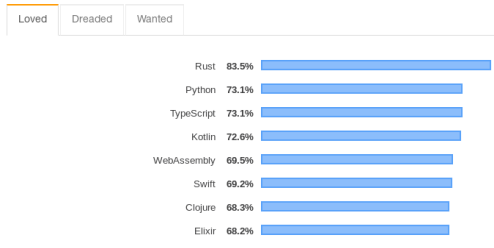
- ▶ Developed by Mozilla Research
- ▶ Goal: better memory safety, but retain high-performance
- ▶ Announced in 2010
- ▶ 2011: First successfully compiled `rustc`



Why Rust?



Most Loved, Dreaded, and Wanted Languages

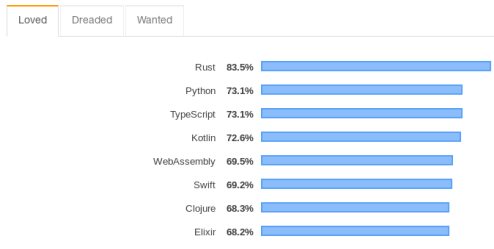


► People love it

Why Rust?



Most Loved, Dreaded, and Wanted Languages



► People love it

Most Loved, Dreaded, and Wanted Languages

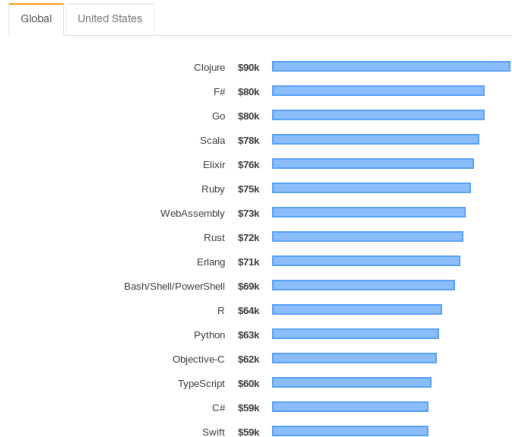


► Interest to learn it

Why Rust?



What Languages Are Associated with the Highest Salaries Worldwide?



► Rustaceans get paid well

Why now?

Industry puts Rust to production



- ▶ Mozilla: sytlo, soon WebRender
- ▶ Dropbox: storage system
- ▶ Facebook: Mercurial rewrite
- ▶ Google: Fuschia OS
- ▶ Amazon: Firecracker VMM
- ▶ Microsoft: Azure
- ▶ Oracle: railcar

Why Rust - better programs



- ▶ Robustness + Reliability
 - ▶ Memory safety
 - ▶ Thread safety
 - ▶ Result error types (`Result<T, E>`)

Why Rust - better programs



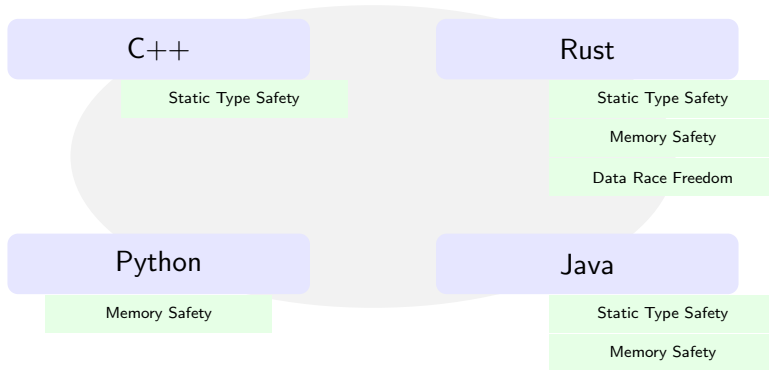
► Robustness + Reliability

- Memory safety
- Thread safety
- Result error types (`Result<T, E>`)

► Performance

- Compiled language
- *No garbage collection*
- Lightweight standard lib

Overview: Safety in programming languages



Why Rust - benefits for programmers



```
projects/rust/hello_cargo cargo build
Compiling hello_cargo v0.1.0 (/home/skaestle/projects/rust/hello_cargo)
error[E0425]: cannot find function `foo` in this scope
--> src/main.rs:29:5
29 |     foo();
   |     ^^^ help: a function with a similar name exists: `fooo`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0425`.
error: Could not compile `hello_cargo`.
To learn more, run the command again with --verbose.
```

- ▶ Extremely helpful compiler messages
- ▶ Push complexity to compilation, rather than runtime
- ▶ Very good tooling
- ▶ Good IDE integration
- ▶ Good documentation: `rustup doc`

Dependency management



The screenshot shows the crates.io website. At the top, there's a green header with the crates.io logo (a yellow cube) and the text "crates.io Rust Package Registry". To the right of the logo is a search bar with the placeholder text "Click or press 'S' to search...", and links for "Browse All Crates", "Docs", and "Log In with GitHub". Below the header, the main content area has a large heading "The Rust community's crate registry". Under this heading are two orange buttons: "Install Cargo" and "Getting Started". Below these buttons, there's a paragraph of text: "Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work." To the right of this text are two statistics: "1,124,677,049 Downloads" and "26,619 Crates in stock". At the bottom, there are three columns of crates: "New Crates", "Most Downloaded", and "Just Updated". Each column lists a crate name and version, with a green arrow icon next to it.

New Crates	Most Downloaded	Just Updated
amux (0.0.1)	rand (0.7.0-pre.0)	trie-db (0.12.3)
statechain (0.0.0)	lib (0.2.0)	trie-mem (0.12.3)

- ▶ Large selection of external modules (called *crates*)
- ▶ Generally very good code quality



Okay, so what are the cons



- ▶ Steep learning curve
 - ▶ Some have reported up to a month to become productive

Okay, so what are the cons



- ▶ Steep learning curve
 - ▶ Some have reported up to a month to become productive
- ▶ Can be frustrating at times
 - ▶ Don't give up, simplify code, ask community
 - ▶ The compiler is (almost) always right, if it isn't happy, neither should you

Okay, so what are the cons




- ▶ Steep learning curve
 - ▶ Some have reported up to a month to become productive
- ▶ Can be frustrating at times
 - ▶ Don't give up, simplify code, ask community
 - ▶ The compiler is (almost) always right, if it isn't happy, neither should you
- ▶ Compilation times can be long
 - ▶ Rust compiles all code, even from dependencies

Okay, so what are the cons



- ▶ Steep learning curve
 - ▶ Some have reported up to a month to become productive
- ▶ Can be frustrating at times
 - ▶ Don't give up, simplify code, ask community
 - ▶ The compiler is (almost) always right, if it isn't happy, neither should you
- ▶ Compilation times can be long
 - ▶ Rust compiles all code, even from dependencies
- ▶ Some things need getting used to:
 - ▶ No OOP as in the traditional sense
 - ▶ Traits




Not today,
→ Rust book

Okay, so what are the cons



- ▶ Steep learning curve
 - ▶ Some have reported up to a month to become productive
- ▶ Can be frustrating at times
 - ▶ Don't give up, simplify code, ask community
 - ▶ The compiler is (almost) always right, if it isn't happy, neither should you
- ▶ Compilation times can be long
 - ▶ Rust compiles all code, even from dependencies
- ▶ Some things need getting used to:
 - ▶ No OOP as in the traditional sense
 - ▶ Traits
- ▶ Goal today: help you get over the initial fear
- ▶ Plus: Rust community generally very helpful



Not today,
→ Rust book



Section 2

Okay, let's get started

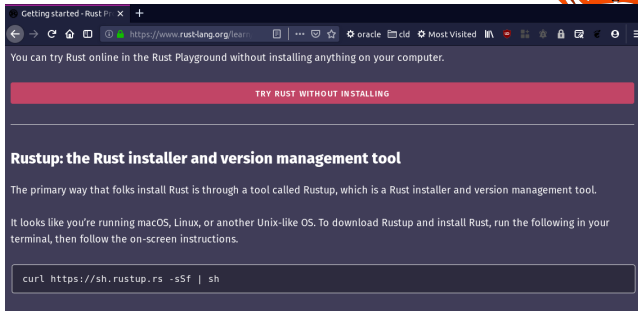
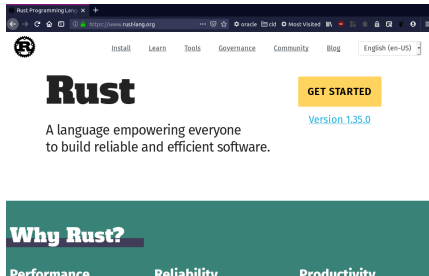
Let's get started



- ▶ `fn` keyword for functions
- ▶ Relatively similar to C/C++

```
fn main() {  
    println!("Hello world");  
}
```

Installing Rust



- ▶ Download from rust-lang.org → Get started
- ▶ Also: online editor: play.rust-lang.org
- ▶ Default settings in installer work well
- ▶ User installation works, no root needed
- ▶ Need build-essentials

Rust tooling



`rustc`

- ▶ The Rust compiler
- ▶ Normally not used directly

`rustup`

- ▶ Used to manage and update tool-chains
- ▶ Documentation and help with error handling

`cargo`

- ▶ Rust's build system
- ▶ Used to create projects
- ▶ Dependency management
- ▶ Compile your programs `cargo build`
- ▶ Run your programs `cargo run`

More in Rust book

Meet cargo



```
> which cargo
```

```
> which rustc
```

```
> cargo new hello_world
```

```
> cd hello_world
```

```
> ls -R
```

```
..
```

```
Cargo.toml  src
```

```
./src:
```

```
main.rs
```

- ▶ Creates an empty project

- ▶ Creates a git repository

- ▶ Two files

- ▶ Cargo.toml: Dependency management and project info

- ▶ main.rs: Main Rust file

Lets run it



cargo run

```
/tmp/hello_world$ cargo run
Compiling hello_world v0.1.0 (/tmp/hello_world)
Finished dev [unoptimized + debuginfo] target(s) in 0.40s
Running `target/debug/hello_world`
Hello, world!
```

- ▶ That wasn't all too hard!
- ▶ Also:
 - ▶ `cargo run --release`
 - ▶ `cargo build [--release]`
 - ▶ Compiled binaries are in `target/{release,debug}/{app}`
 - ▶ Rust compiler uses clang



Section 3

Playtime

Playtime — recap



- ▶ Functions
- ▶ Error handling
- ▶ Variables, default: immutable
- ▶ Typed w/ inference

```
fn hello(s: String) {  
    println!(s);  
}
```

```
fn main() {  
    let mut s = "Hello".to_string();  
    s += " world";  
    hello(s);  
}
```



Section 4

Memory safety

Memory Safety



- ▶ Like Java:
 - ▶ Automatic memory dealloc
 - ▶ Checks index out of bound

Memory Safety



- ▶ Like Java:
 - ▶ Automatic memory dealloc
 - ▶ Checks index out of bound
- ▶ *But:* No garbage collection (GC) needed
 - ▶ → No overhead from running GC

Memory Safety



- ▶ Like Java:
 - ▶ Automatic memory dealloc
 - ▶ Checks index out of bound
- ▶ *But*: No garbage collection (GC) needed
 - ▶ → No overhead from running GC
- ▶ Instead: ownership + borrow checker
- ▶ Sharing: wrap data in synchronization primitives, checked by compiler

Garbage collection: memory management in Java



Short review of garbage collected languages, here: Java
Slides from: RustConf17

```
void main() {  
➡ Vector name = ...;  
  helper(name);  
  helper(name);  
}
```

```
void helper(Vector name) {  
  ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    ➔ helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    → helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    ...  
}
```

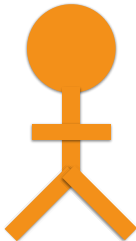
↑
Take **reference**
to Vector



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    ➔ helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

➔

```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```



```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java


```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

➔

```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```



```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```



```
void helper(Vector name) {  
    ...  
}
```

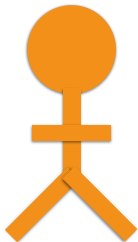


“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```



```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```



```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    ...  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    new Thread(...);  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    new Thread(...);  
}
```



“Ownership” in Java


```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    new Thread(...);  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    new Thread(...);  
}
```



“Ownership” in Java

```
void main() {  
    Vector name = ...;  
    helper(name);  
    helper(name);  
}
```

```
void helper(Vector name) {  
    new Thread(...);  
}
```

Ownership in Rust



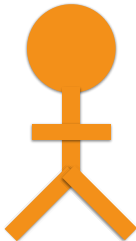
Hold on tight: this is going to be a bit more tricky (but perhaps easier than GC).
Slides from: RustConf17



Ownership

```
fn main() {  
➔ let name = format!("{}",  
  helper(name);  
  helper(name);  
}
```

```
fn helper(name: String) {  
  println!("{}",  
}  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    → helper(name);  
    helper(name);  
}
```

```
fn helper(name: String) {  
    println!("{}", ..);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!("{}", ...);  
}
```



Take ownership
of a String



Ownership


```
fn main() {  
    let name = format!("{}", ...);  
    → helper(name);  
    helper(name);  
}
```

```
fn helper(name: String) {  
    println!("{}", ..);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!("{}", ...);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!("{}", ..);  
}
```

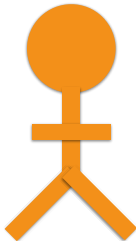


Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!("{}", ..);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}",);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!(..);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```



```
fn helper(name: String) {  
    println!("{}", ..);  
}
```



Ownership

```
fn main() {  
    let name = format!("{}", ...);  
    helper(name);  
    helper(name);  
}
```

```
fn helper(name: String) {  
    println!("{}", ..);  
}
```

Error: use of moved value: `name`



Ownership

Ownership in Rust: Borrowing



What about references?

Slides from: RustConf17



Borrowing: Shared Borrows

```
fn main() {  
    → let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    → let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```

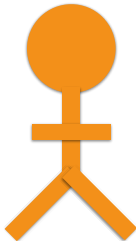


Shared borrow

```
fn main() {  
  ➔ let name = format!("{}",);  
  let reference = &name;  
  helper(reference);  
  helper(reference);  
}
```

```
fn helper(name: &String) {  
  println!(..);  
}
```

Change type to a
reference to a String



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```

Change type to a
reference to a String



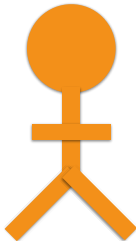
Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

Lend the string,
creating a reference

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```

Change type to a
reference to a String



Shared borrow

```
fn main() {
  let name = format!("{}",);
  let reference = &name;
  helper(reference);
  helper(reference);
}
```



Lend the string,
creating a reference

```
fn helper(name: &String) {
  println!("{}", ..);
}
```



Change type to a
reference to a String



Shared borrow

```
fn main() {
  let name = format!("{}",);
  let reference = &name;
  helper(reference);
  helper(reference);
}
```



Lend the string,
creating a reference

```
fn helper(name: &String) {
  println!(..);
}
```



Change type to a
reference to a String



Shared borrow


```
fn main() {
  let name = format!("{}", ...);
  let reference = &name;
  helper(reference);
  helper(reference);
}
```



Lend the string,
creating a reference

```
fn helper(name: &String) {
  println!("{}", ..);
}
```



Change type to a
reference to a String



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}",);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```



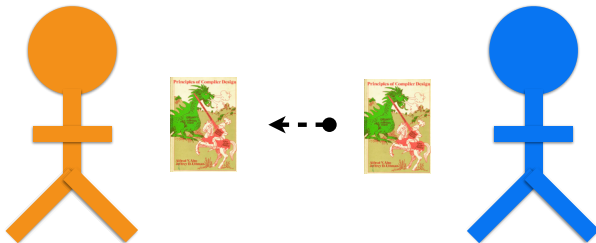
```
fn helper(name: &String) {  
    println!("{}", name);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```



```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```



```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}", ...);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```

```
fn helper(name: &String) {  
    println!("{}", ..);  
}
```



Shared borrow

```
fn main() {  
    let name = format!("{}",);  
    let reference = &name;  
    helper(reference);  
    helper(reference);  
}
```



```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

Shared borrow

Shared == Immutable

```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

```
fn helper(name: &String) {  
    name.push_str("foo");  
}
```

Shared == Immutable

```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

← OK. Just reads.

```
fn helper(name: &String) {  
    name.push_str("foo");  
}
```

Shared == Immutable

```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

← **OK.** Just reads.

```
fn helper(name: &String) {  
    name.push_str("foo");  
}
```

← **Error.** Writes.

Shared == Immutable

```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

← **OK.** Just reads.

```
fn helper(name: &String) {  
    name.push_str("foo");  
}
```

← **Error.** Writes.

```
error: cannot borrow immutable borrowed content `*name`  
      as mutable  
      name.push_str("s");  
      ^~~~~
```

Shared == Immutable^{*}

```
fn helper(name: &String) {  
    println!("{}", name);  
}
```

← **OK.** Just reads.

```
fn helper(name: &String) {  
    name.push_str("foo");  
}
```

← **Error.** Writes.

```
error: cannot borrow immutable borrowed content `*name`  
      as mutable  
      name.push_str("s");  
      ^~~~~
```

^{*} **Actually:** mutation only in **controlled circumstances**.



Section 5

Thread safety

Thread safety



- ▶ Sharing possible by means of reference, as with C/C++ and Java
- ▶ One writable reference to data only
- ▶ *Or* arbitrarily many read-only references



Section 6

A proper example: ORM + webserver

Setup dependencies

Diesel: ORM for Rust



- ▶ `cargo install diesel_cli --no-default-features --features mysql`
- ▶ `echo DATABASE_URL="mysql://rustuser:%23Code4Fun@130.61.92.85/"\"
"rustdemo" > .env`
- ▶ `diesel setup`
- ▶ `diesel print-schema`

Where to go from here?



- ▶ Look at the Rust book
- ▶ Join meetup “Rust Zurich”