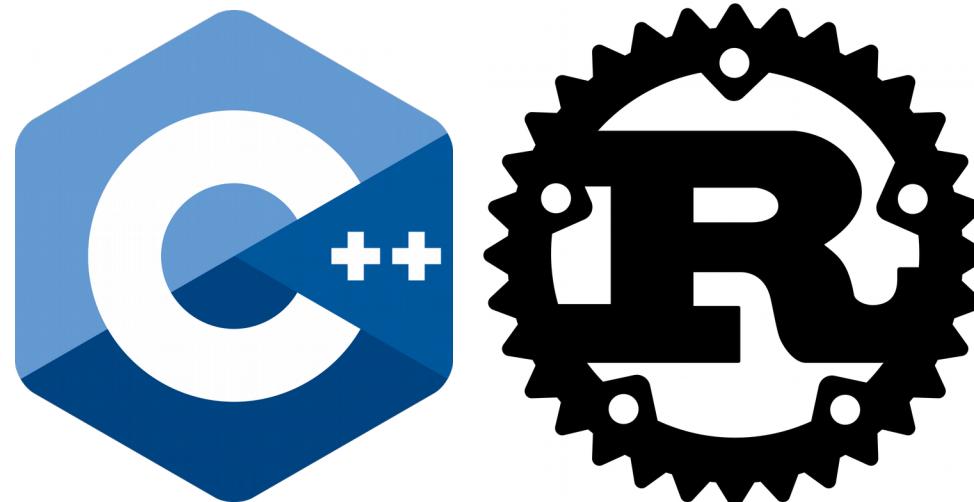


# C++ & Rust: A Comparison



by Robin Freyler

Date: November 15<sup>th</sup> 2017

```
std::cout << self;
```

Co-founder at



```
std::cout << self;
```

Co-founder at



B.Sc. in Computer Science



# `std::cout << self;`

Co-founder at



B.Sc. in Computer Science

~2 years experience with Rust (1.05)



# `std::cout << self;`

Co-founder at



B.Sc. in Computer Science

~2 years experience with Rust (1.05)

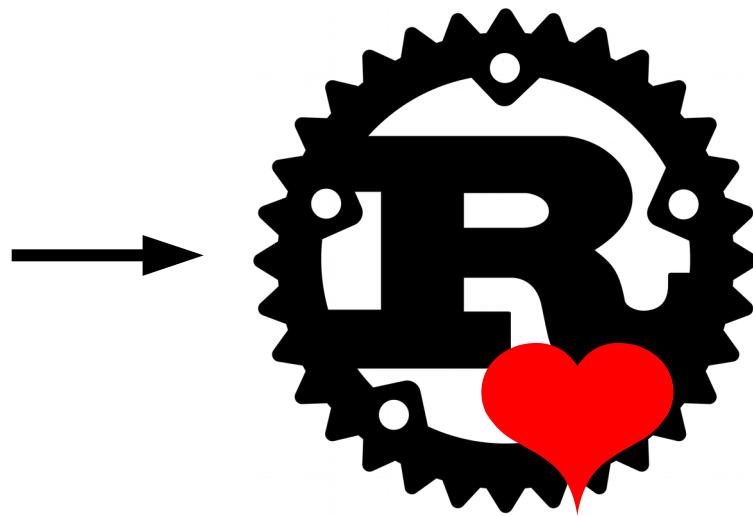
**Projects:** [github.com/robbepop](https://github.com/robbepop)



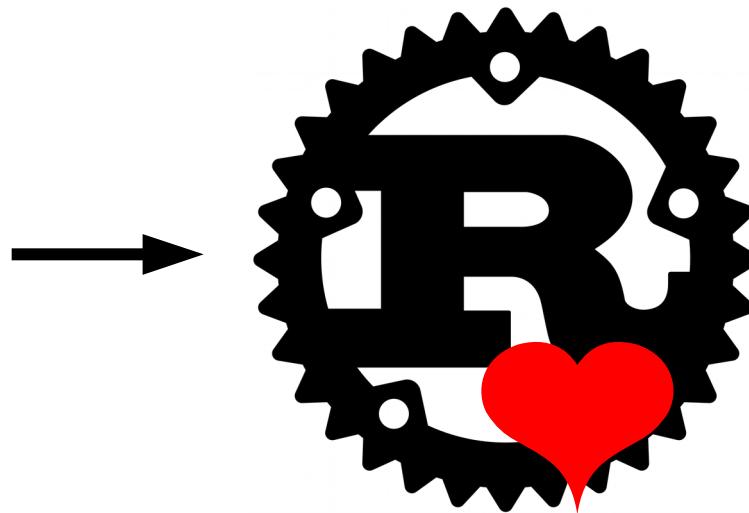
# Structure



# Structure

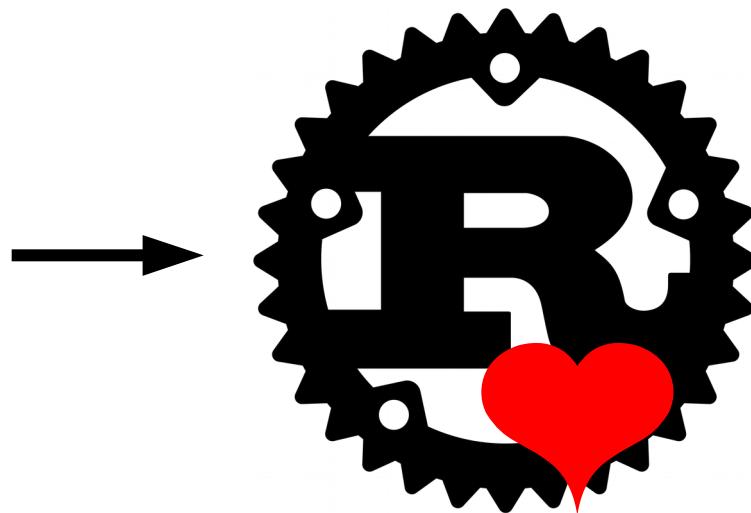


# Structure



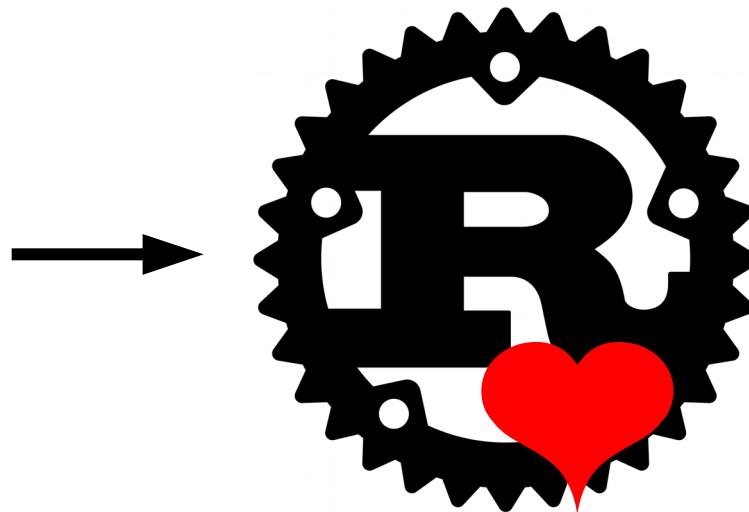
Ownership  
& Borrowing

# Structure



Ownership  
& Borrowing  
  
Lifetimes

# Structure

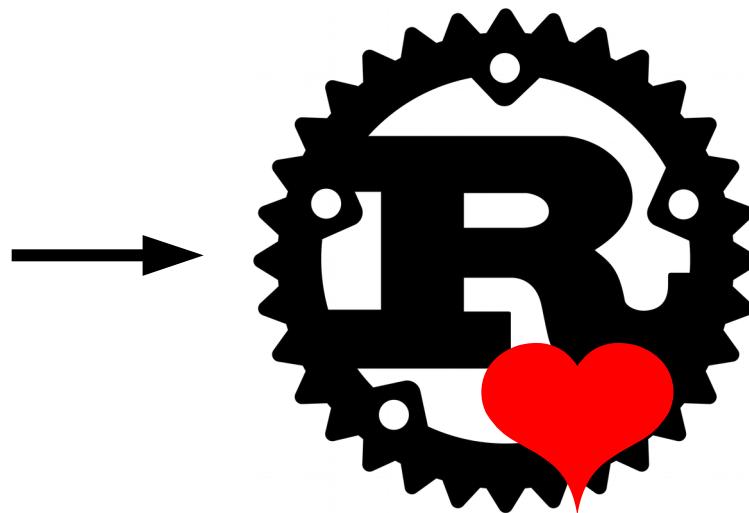


Ownership  
& Borrowing

Lifetimes

Traits

# Structure



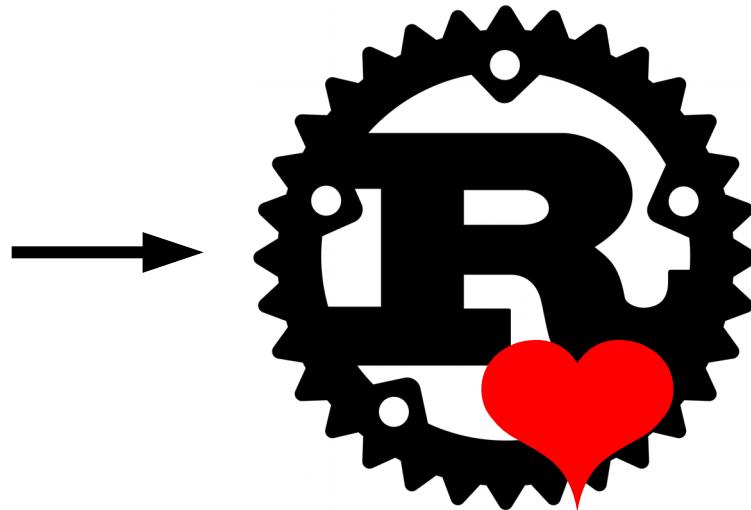
Ownership  
& Borrowing

Lifetimes

Traits

unsafe Rust

# Structure



Ownership  
& Borrowing

Lifetimes

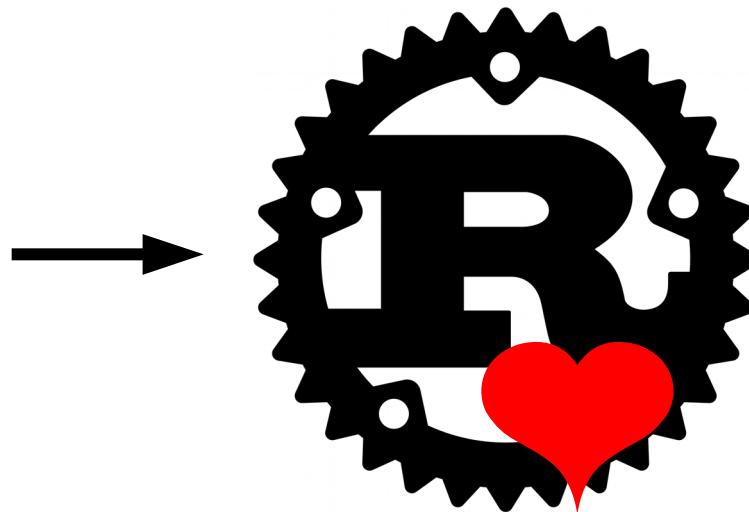
Traits

unsafe Rust



Community

# Structure



Ownership  
& Borrowing

Lifetimes

Traits

unsafe Rust

Livecoding ← Community

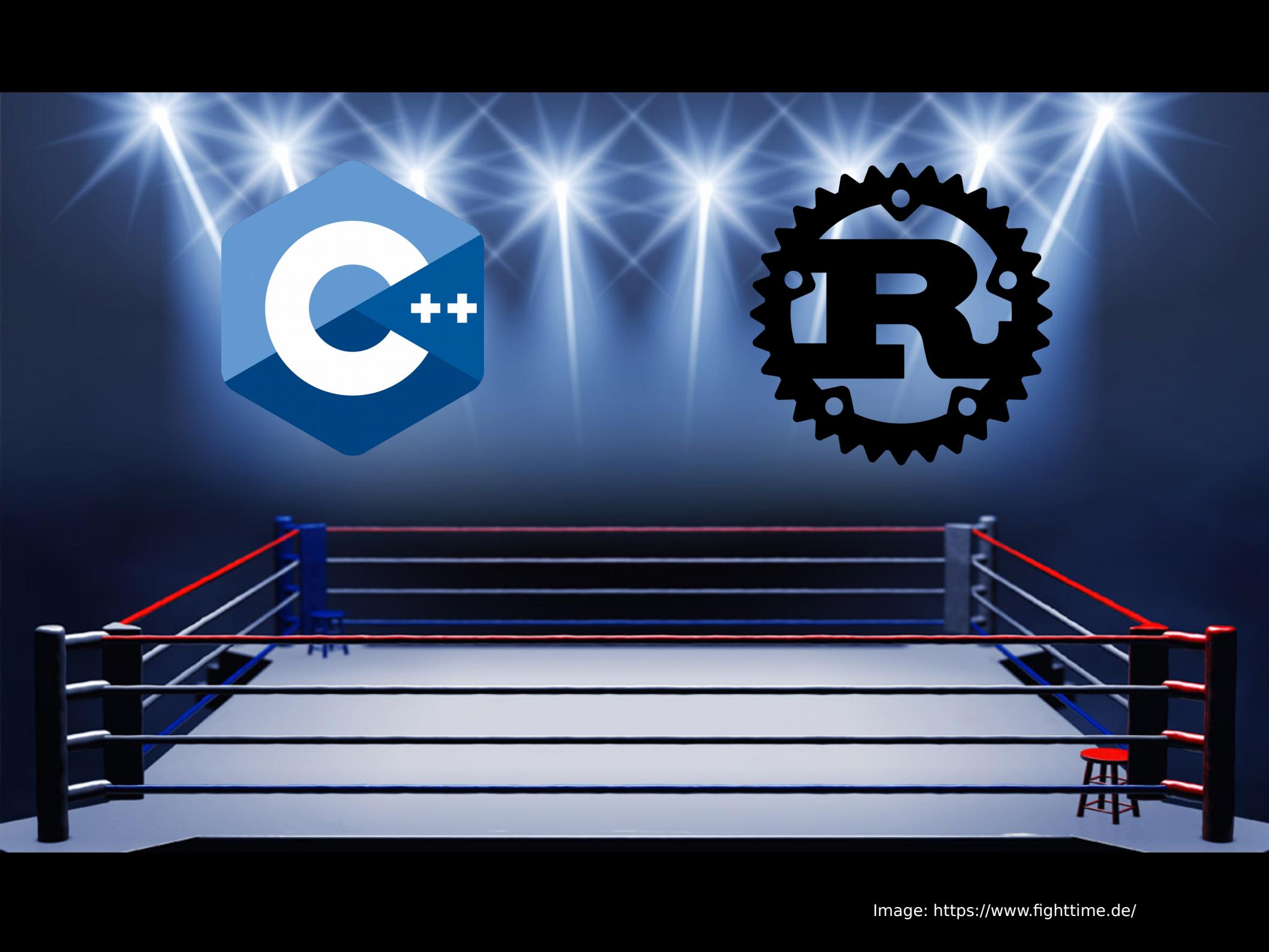


Image: <https://www.fighttime.de/>



Image: <https://www.fighttime.de/>





# Have you tried Rust before?



# Rust Friends

# Rust Friends



# Rust Friends



# Rust Friends



# Rust Friends



facebook



# Rust Friends



facebook



ORACLE®

# Rust Friends



facebook



ORACLE®

Google

# Rust Friends



facebook



ORACLE®

Google

mozilla

# Rust Friends



GNOME™



facebook

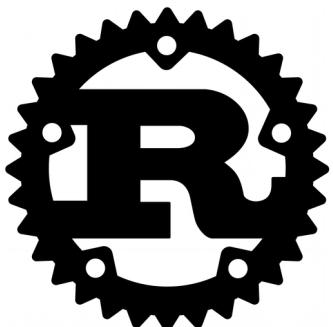


ORACLE®

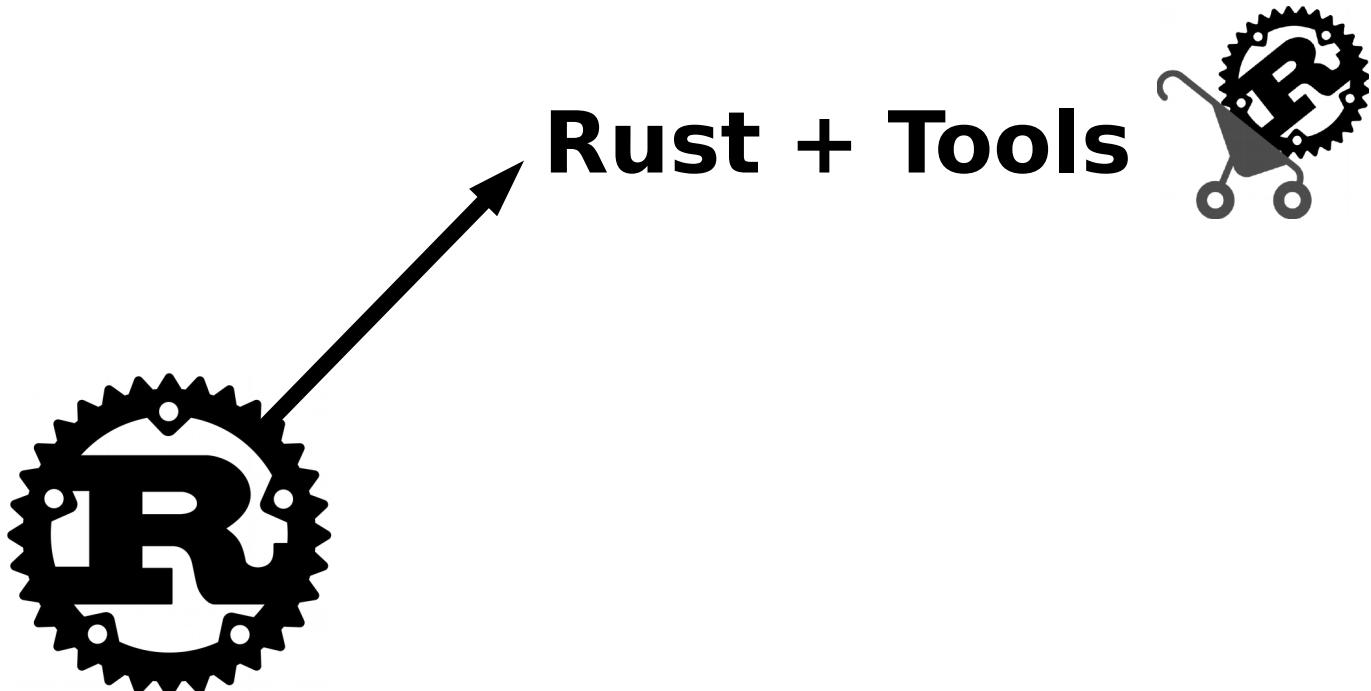
Google

mozilla

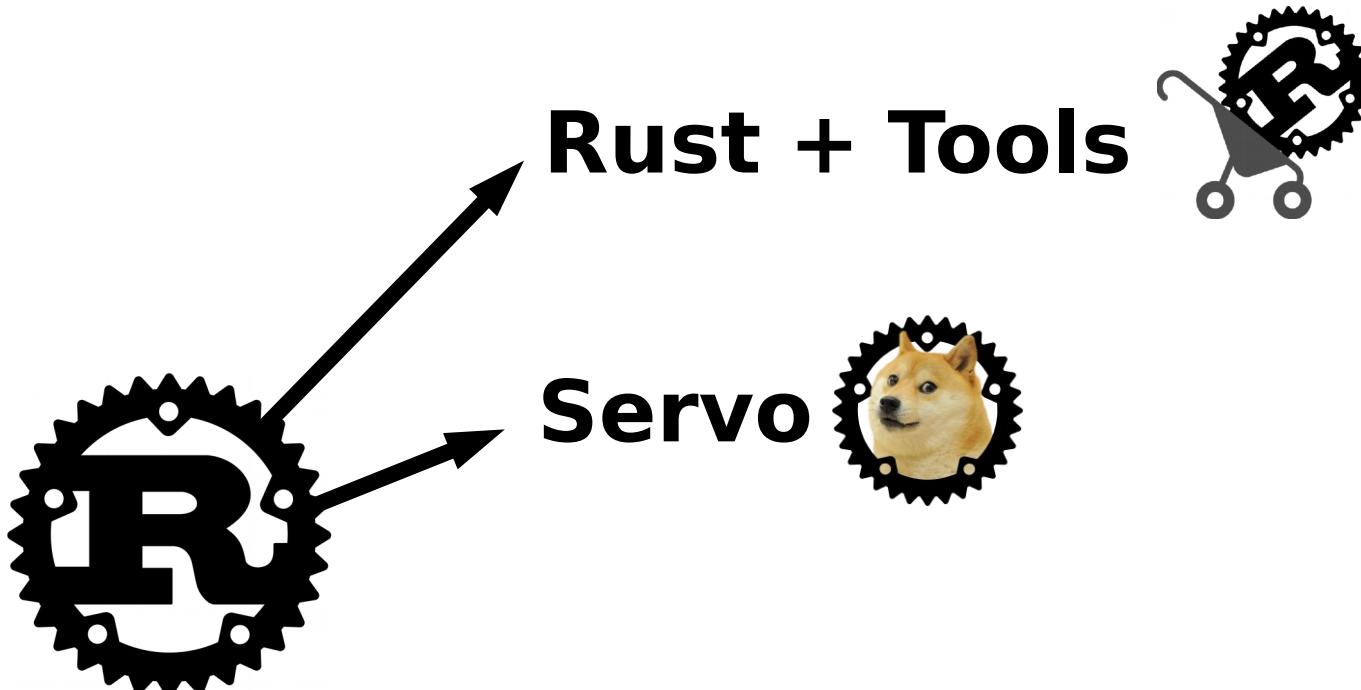
# Rust Projects



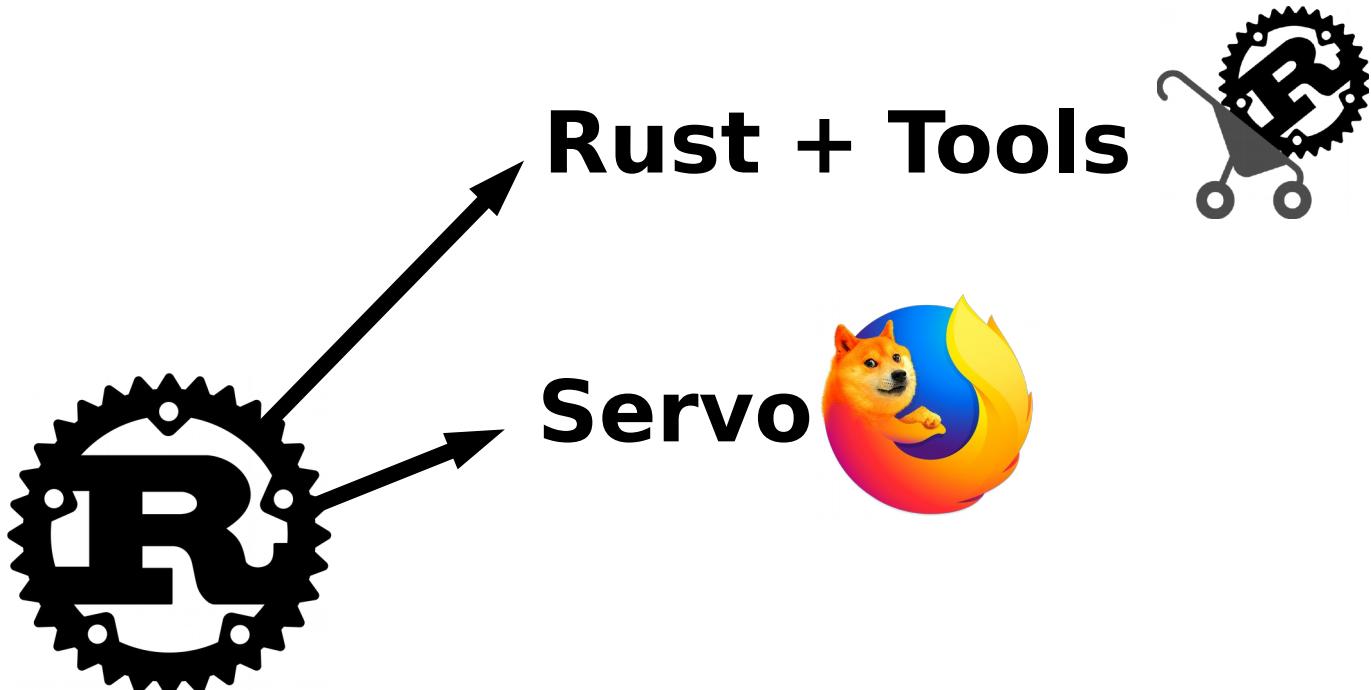
# Rust Projects



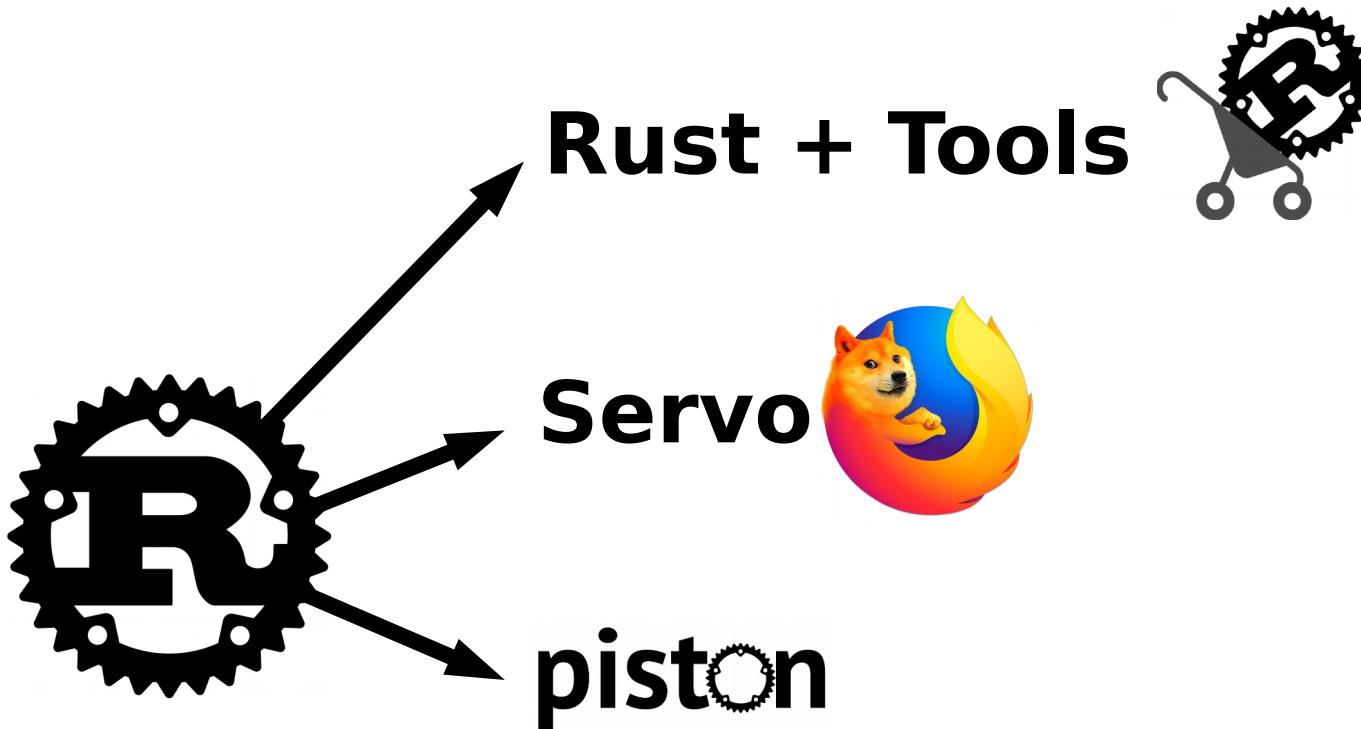
# Rust Projects



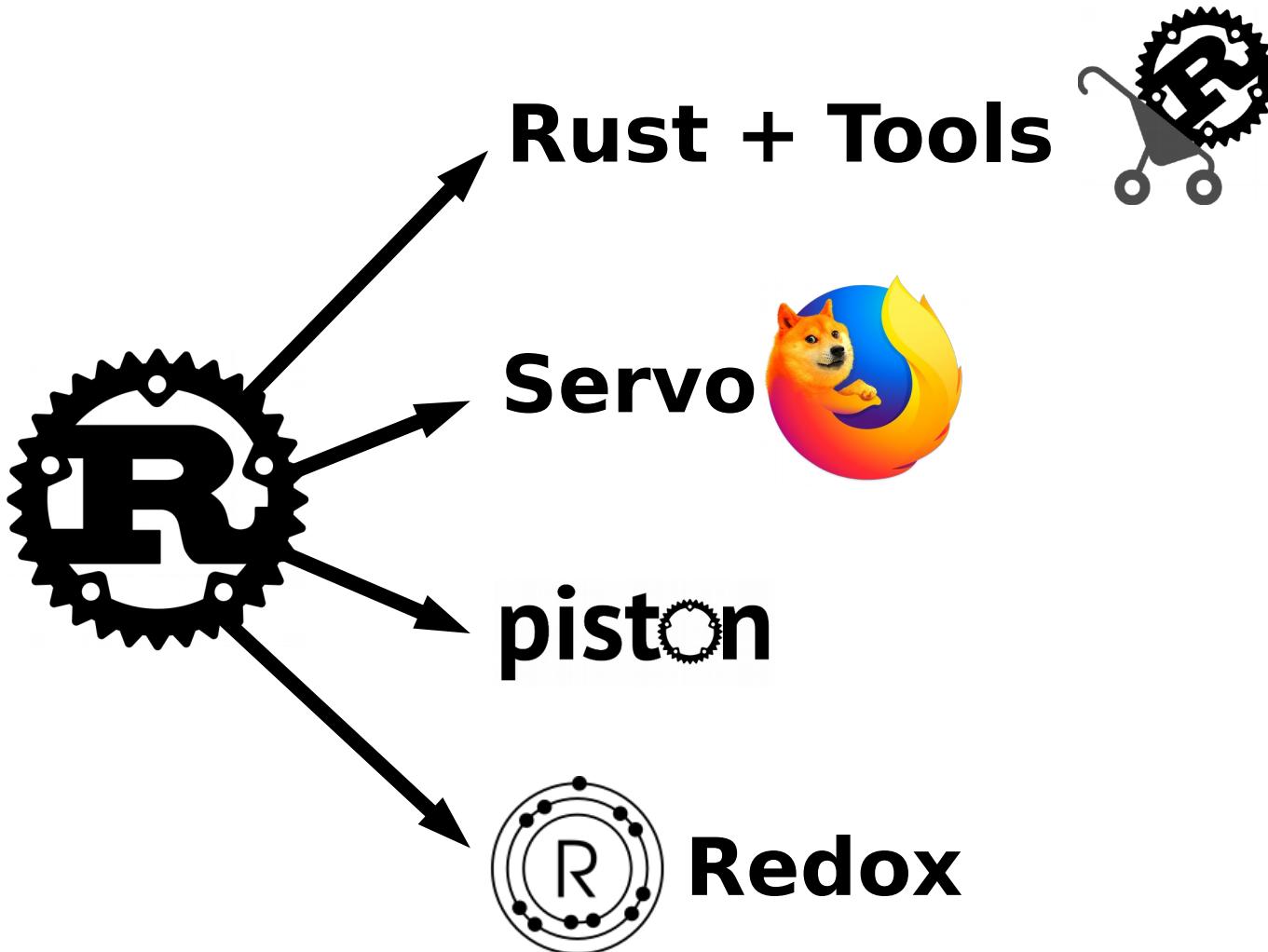
# Rust Projects



# Rust Projects



# Rust Projects

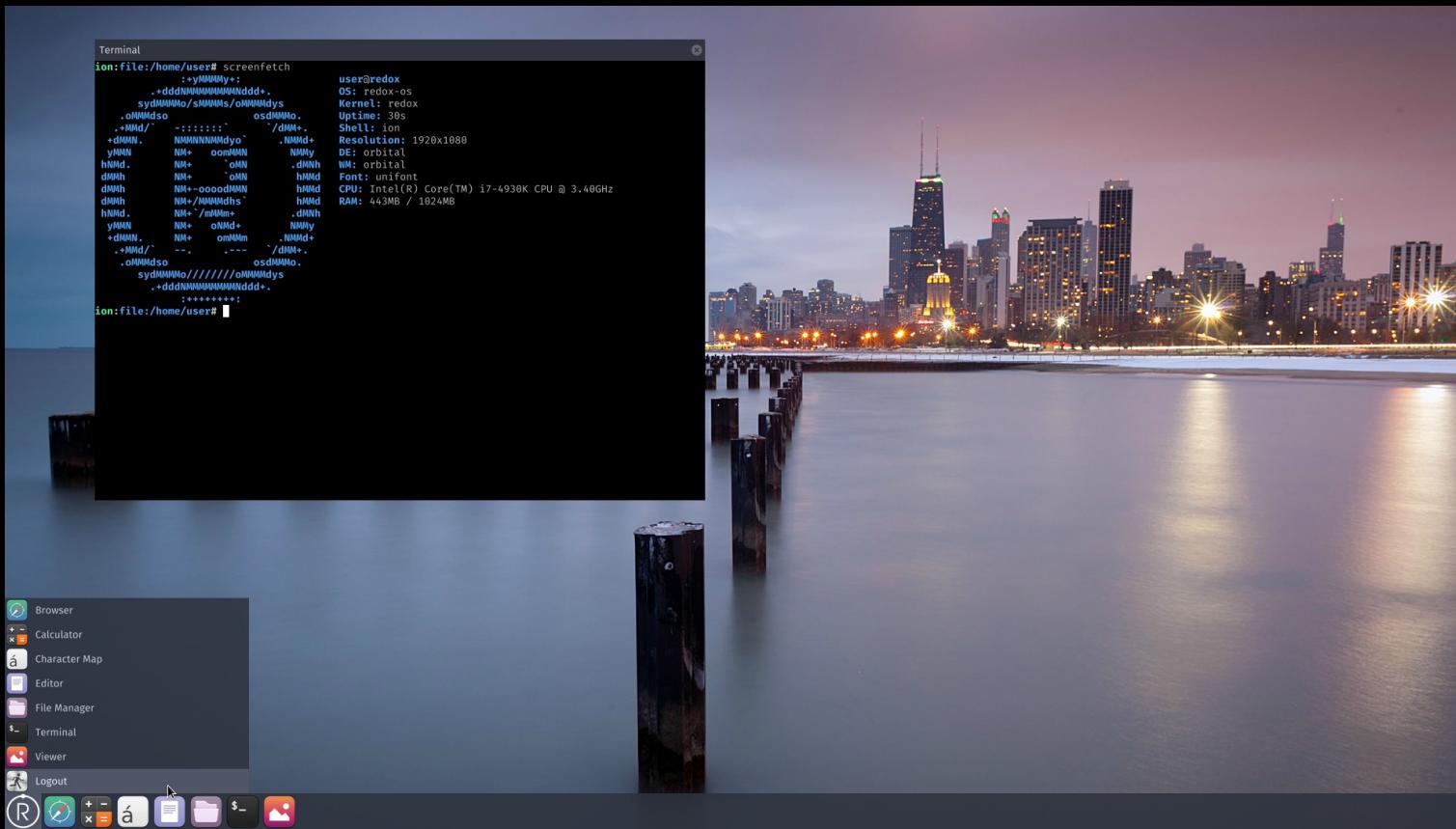


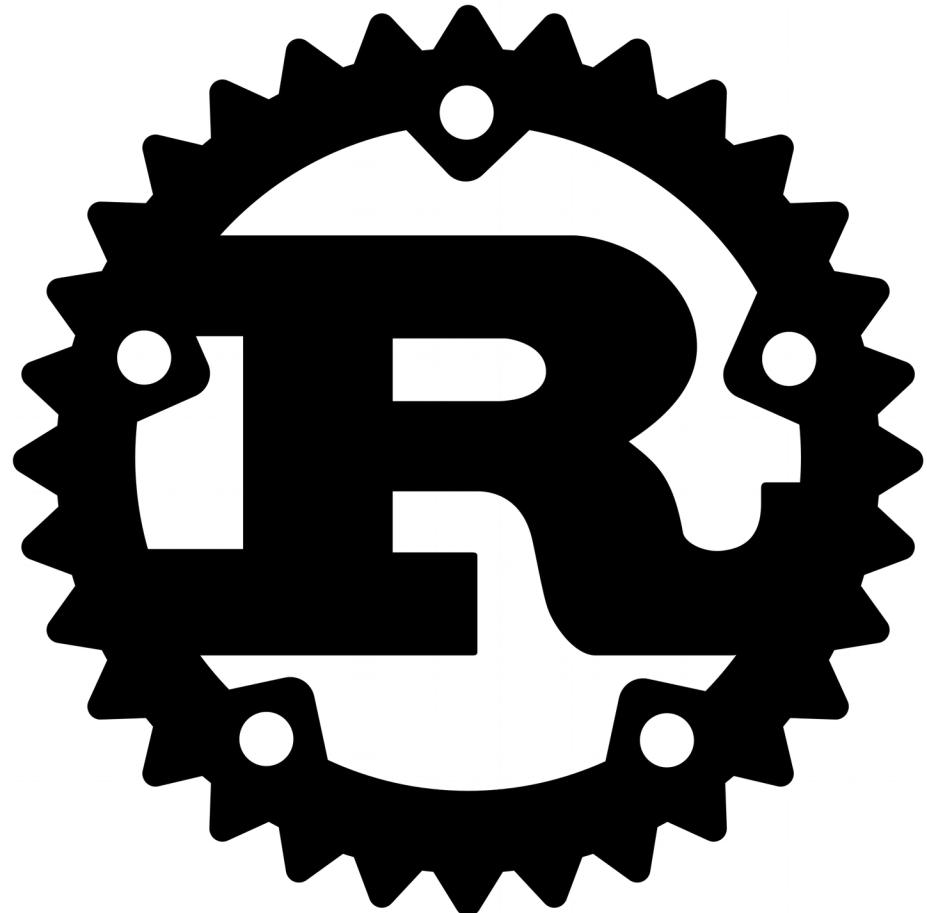
# Redox

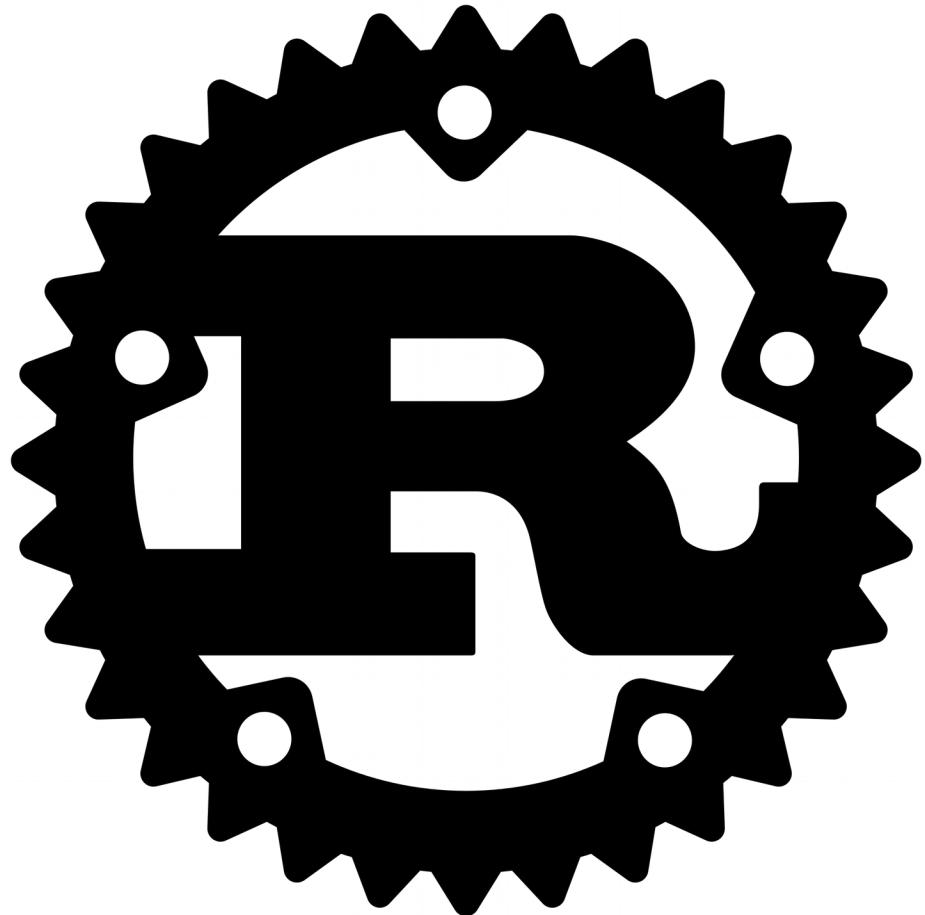
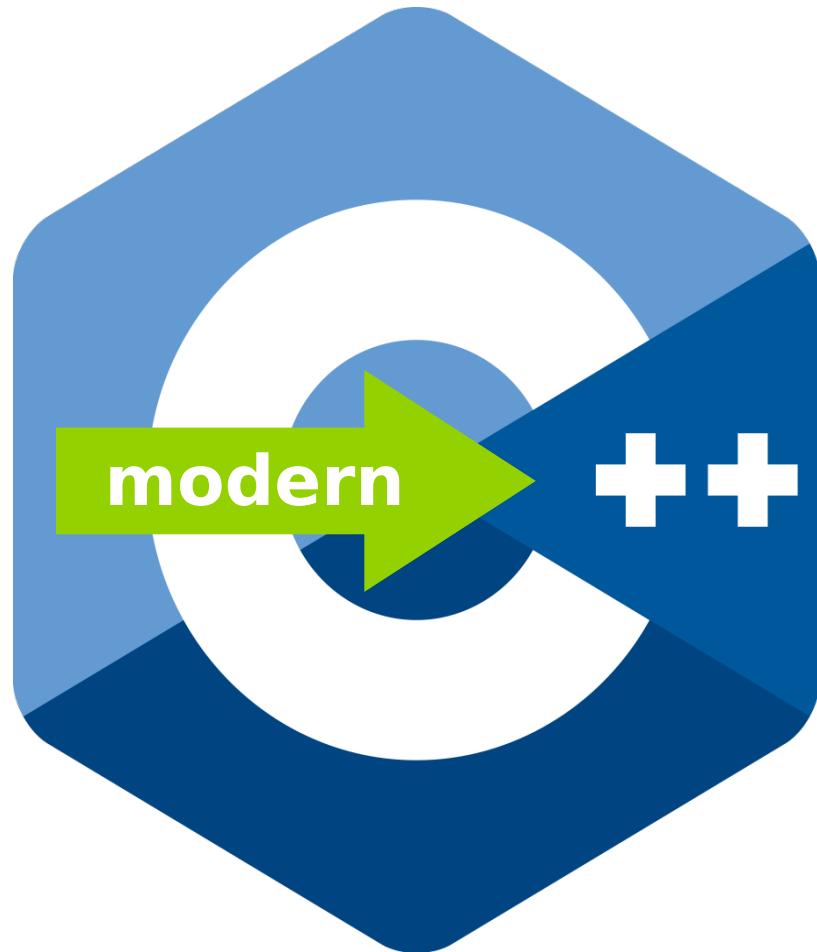
Microkernel design

Unix commands

Supports Rust Standard Library







# What is modern C++?

The best practices using the **current ISO C++ standard**

Aim for completely **type-safe** and **resource-safe** code

- Bjarne Stroustrup

# C++ Complexity

“C makes it easy to shoot yourself in the foot;  
C++ makes it harder, but when you do it blows your whole leg off”

- Bjarne Stroustrup

# C++ Complexity

“C makes it easy to shoot yourself in the foot;  
C++ makes it harder, but when you do it blows your whole leg off”

- Bjarne Stroustrup

**Goal**  
Make C++ simpler!

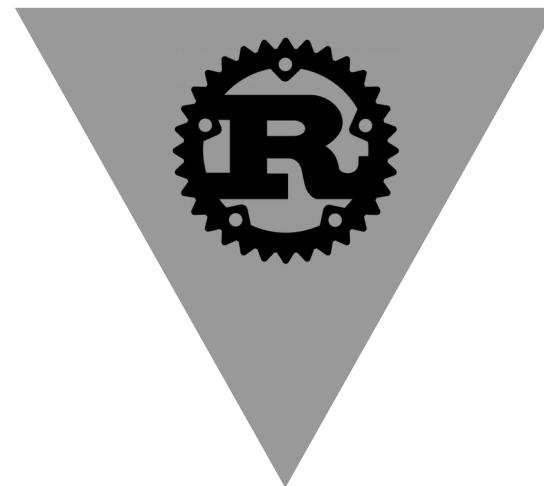
# Rust Focus

Rust is a programming language that's focused on **safety, speed, and concurrency.**

# PICK THREE

FAST

SAFE



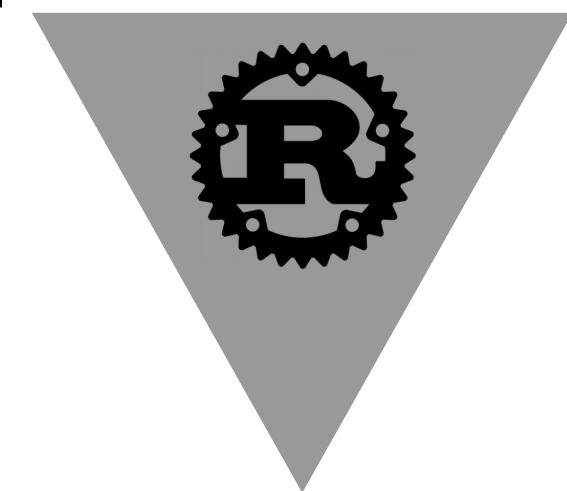
CONCURRENT

# PICK THREE

**FAST**

No Runtime  
Zero-Cost Abstraktionen  
Compiled

**SAFE**



**CONCURRENT**

# PICK THREE

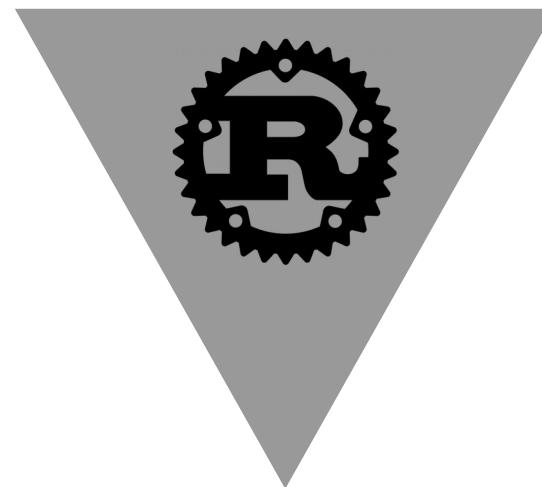
**FAST**

No Runtime  
Zero-Cost Abstraktionen  
Compiled

**SAFE**

Owners & Borrows  
Lifetimes  
Explizites unsafe

**CONCURRENT**



# PICK THREE

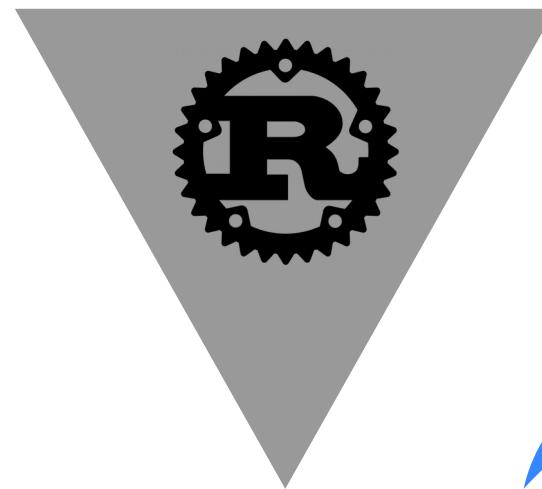
**FAST**

No Runtime  
Zero-Cost Abstraktionen  
Compiled

**SAFE**

Owners & Borrows  
Lifetimes  
Explizites unsafe

**CONCURRENT**



# PICK THREE

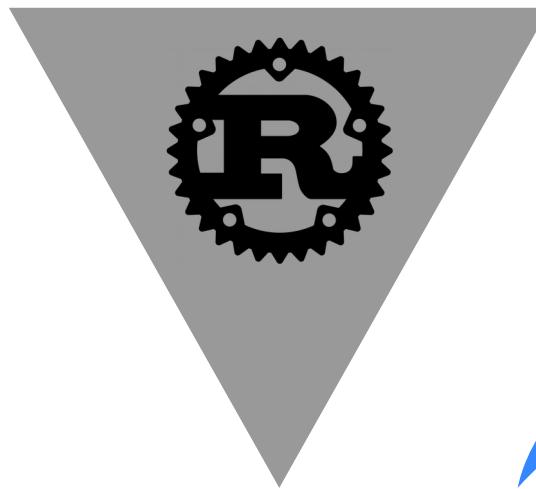
**FAST**

No Runtime  
Zero-Cost Abstraktionen  
Compiled

**SAFE**

Owners & Borrows  
Lifetimes  
Explizites unsafe

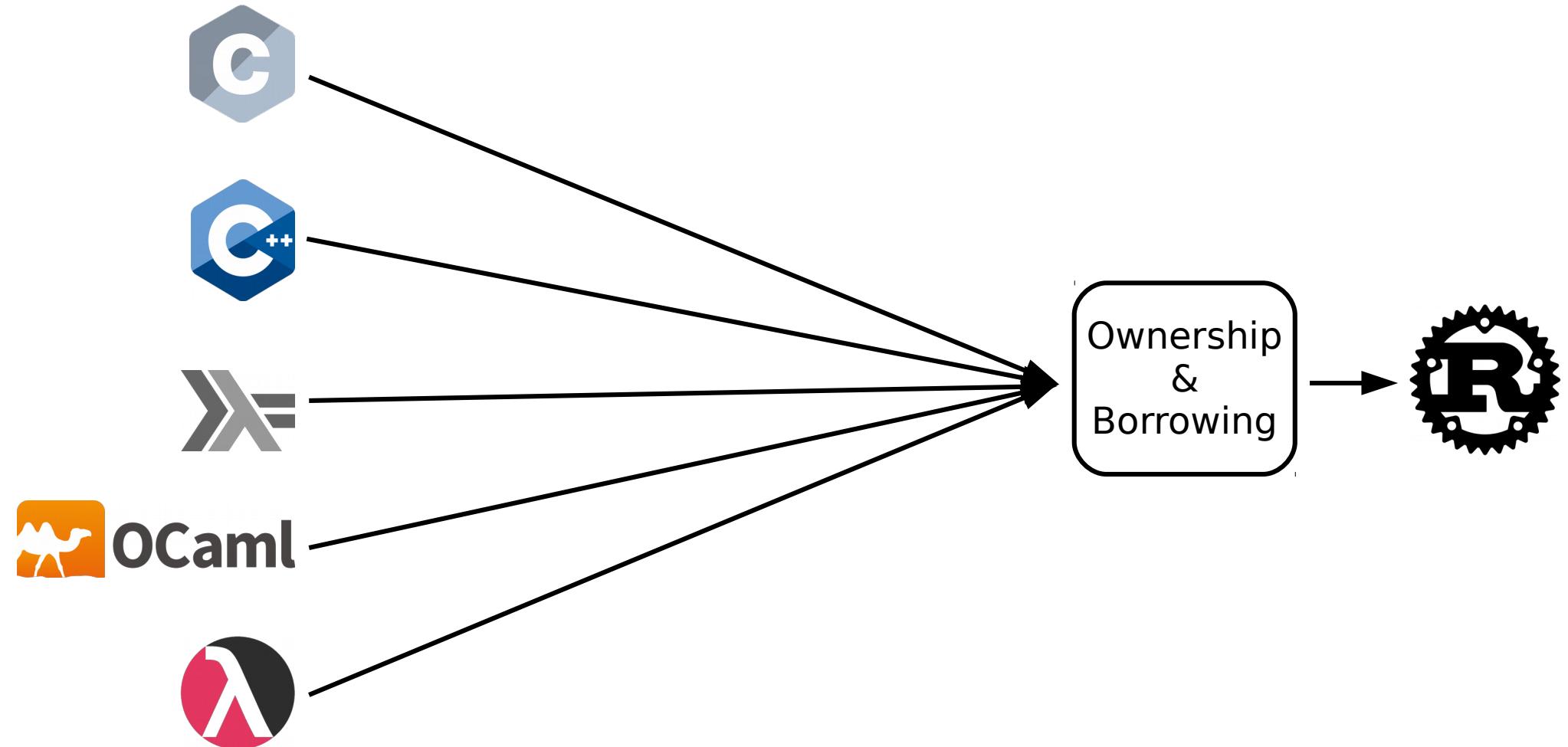
**CONCURRENT**



“the same tools that make Rust safe also help you tackle concurrency head-on.”

- Aaron Turon

# Rust Roots



# Zero-Cost Abstractions

**What you don't use, you don't pay for.**

# Zero-Cost Abstractions

**What you don't use, you don't pay for.**

What you do use, you couldn't hand code any better.

---

# Zero-Cost Abstractions

**What you don't use, you don't pay for.**

What you do use, you couldn't hand code any better.

---

```
fn is_whitespace(text: &str) -> bool {  
    text.chars()  
        .all(|c| c.is_whitespace())  
}
```

# Zero-Cost Abstractions

**What you don't use, you don't pay for.**

What you do use, you couldn't hand code any better.

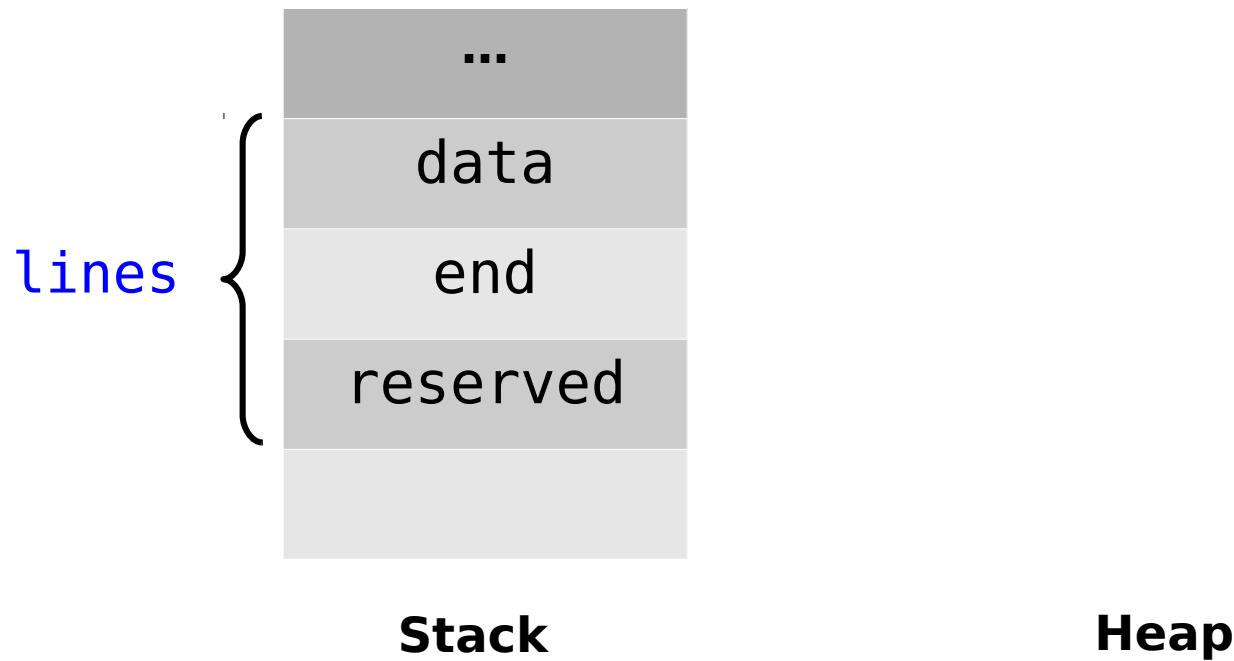
---

```
fn is_whitespace(text: &str) -> bool {  
    text.chars()  
        .all(|c| c.is_whitespace())  
}
```

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {  
    paths.par_iter()  
        .map(|path| Image::load(path))  
        .collect()  
}
```

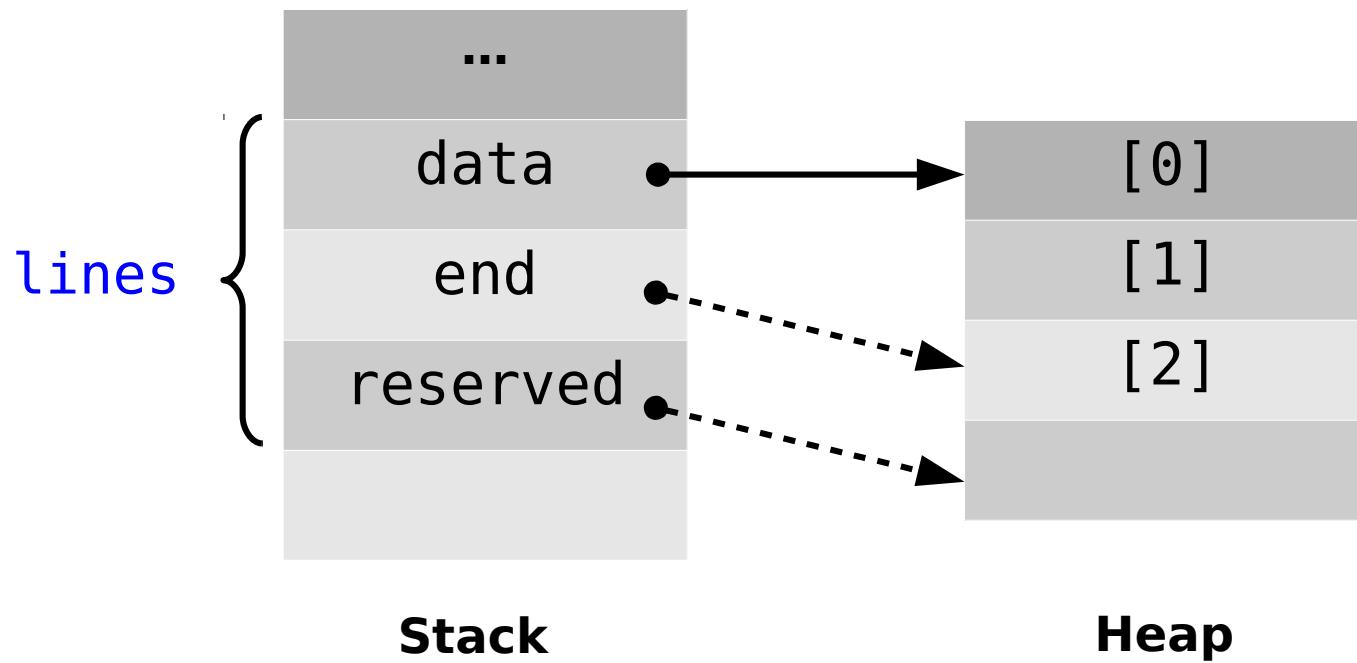
# Zero-Cost Abstractions

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    ...
}
```



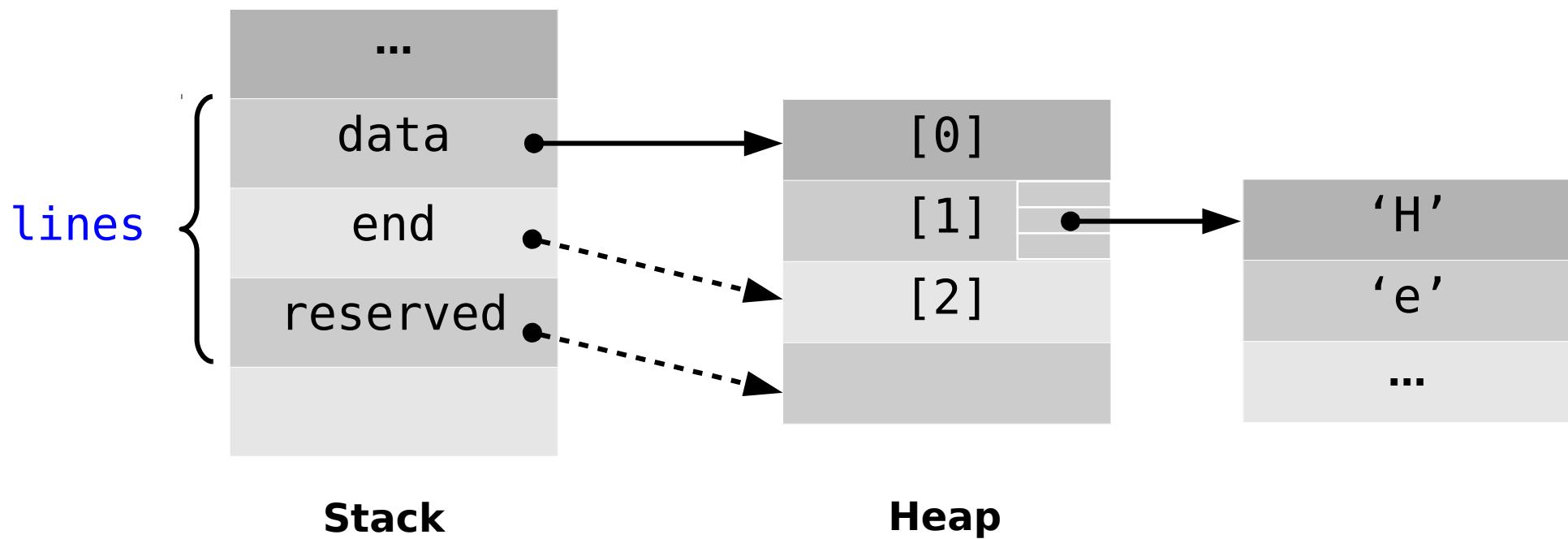
# Zero-Cost Abstractions

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    ...
}
```



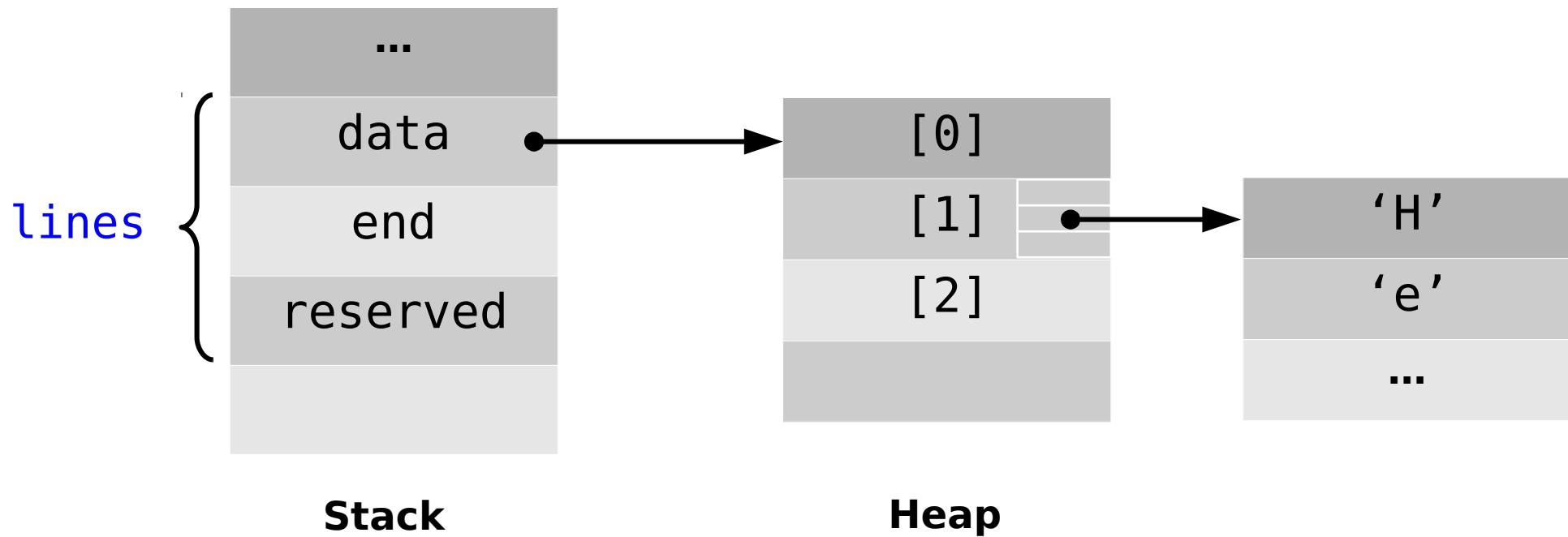
# Zero-Cost Abstractions

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    ...
}
```



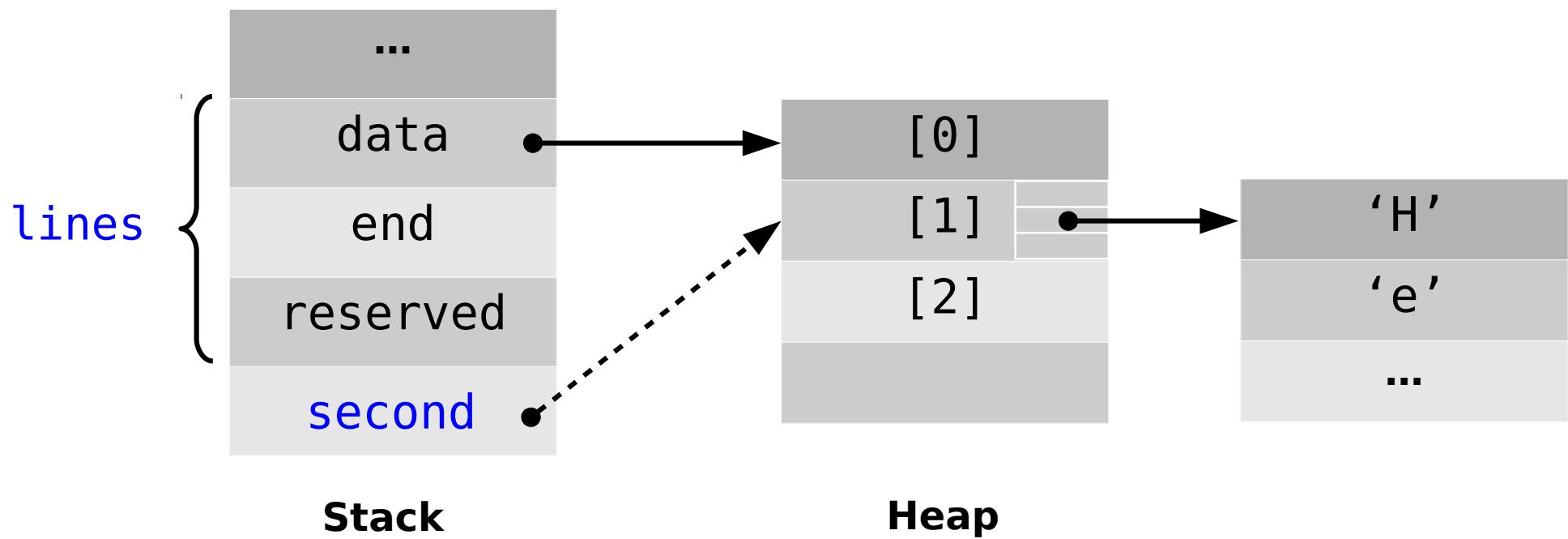
# Zero-Cost Abstractions

```
int main() {  
    vector<string> lines;  
    ...  
    auto& second = lines[1];  
    ...  
}
```



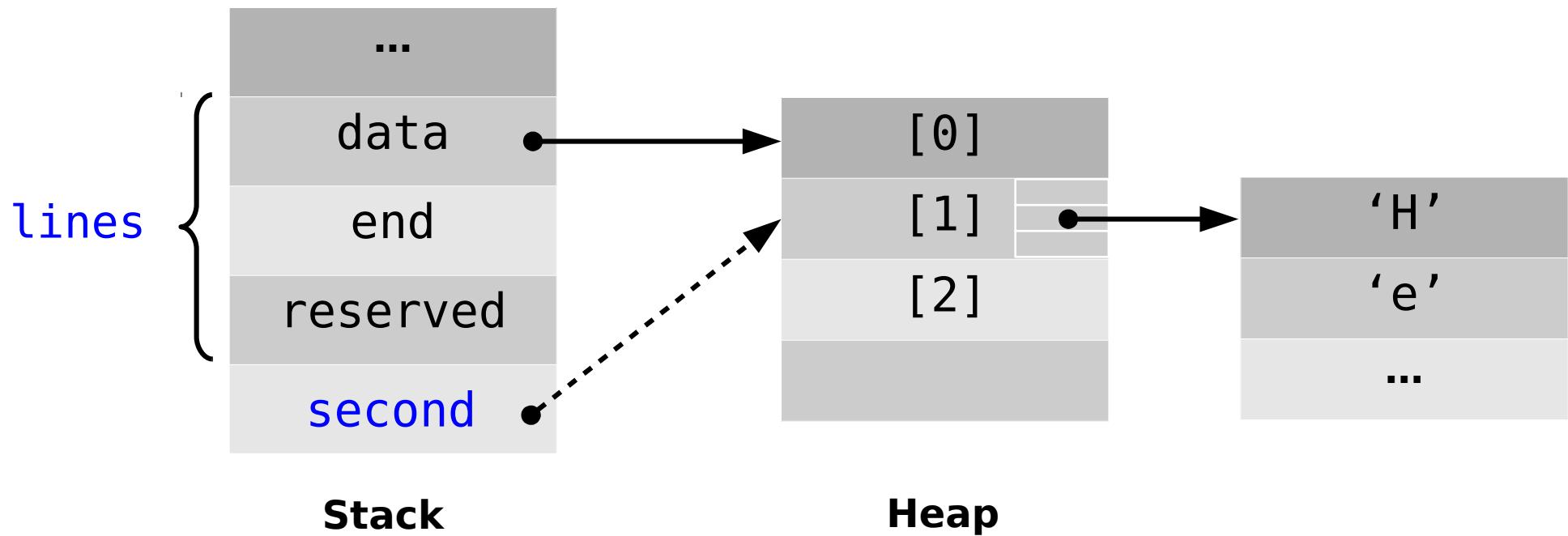
# Zero-Cost Abstractions

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    ...
}
```



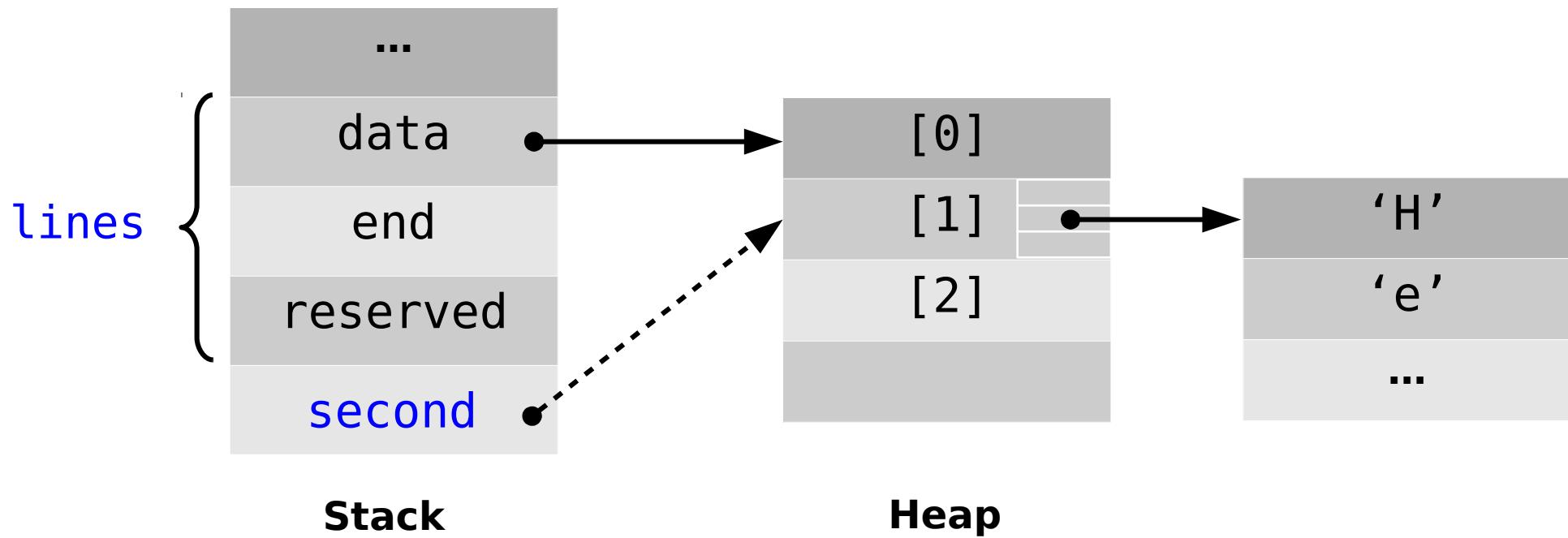
# Zero-Cost Abstractions

```
int main() {  
    vector<string> lines;      ← Memory Layout  
    ...  
    auto& second = lines[1];  
    ...  
}
```



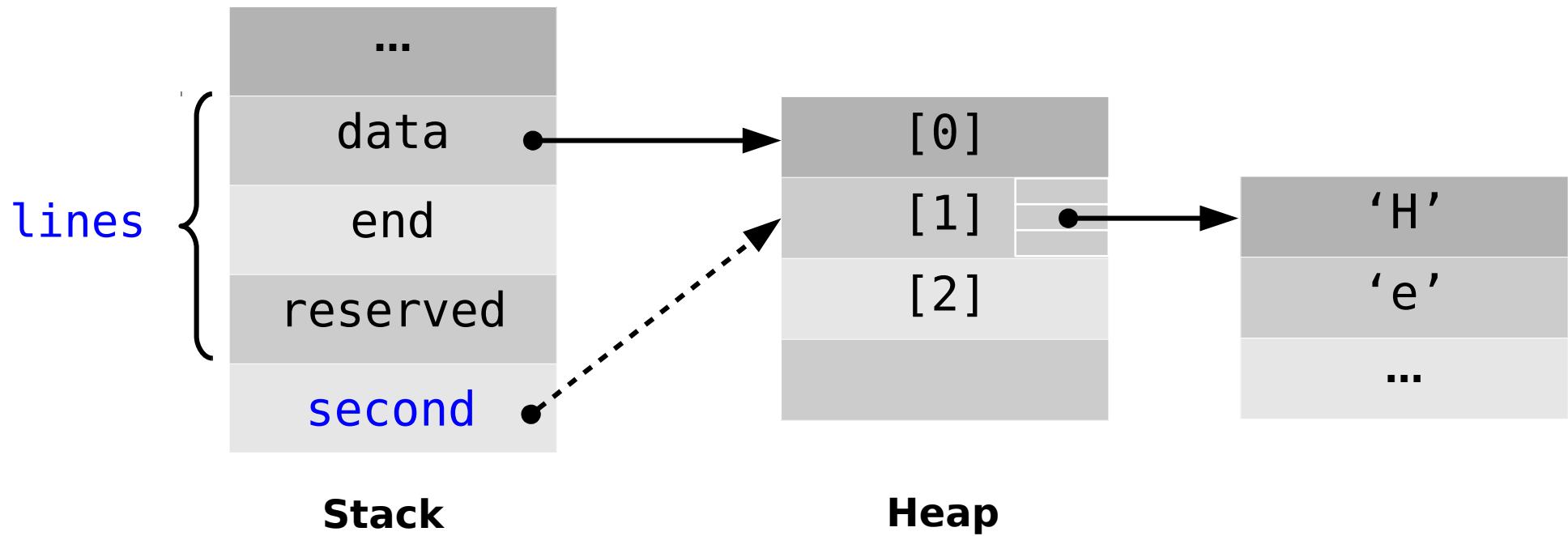
# Zero-Cost Abstractions

```
int main() {  
    vector<string> lines;      ← Memory Layout  
    ...  
    auto& second = lines[1];   ← Lightweight references  
    ...  
}
```



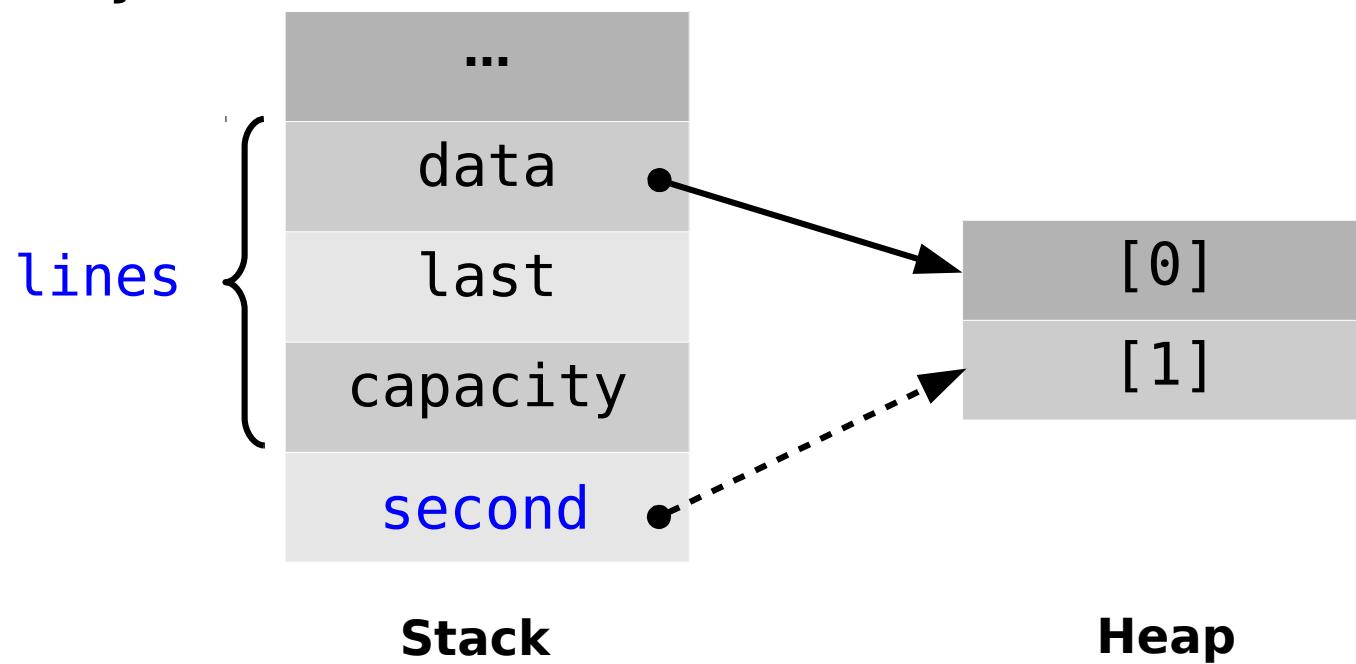
# Zero-Cost Abstractions

```
int main() {  
    vector<string> lines;           ← Memory Layout  
    ...  
    auto& second = lines[1];        ← Lightweight references  
    ...  
}  
                                ← Deterministic destruction
```



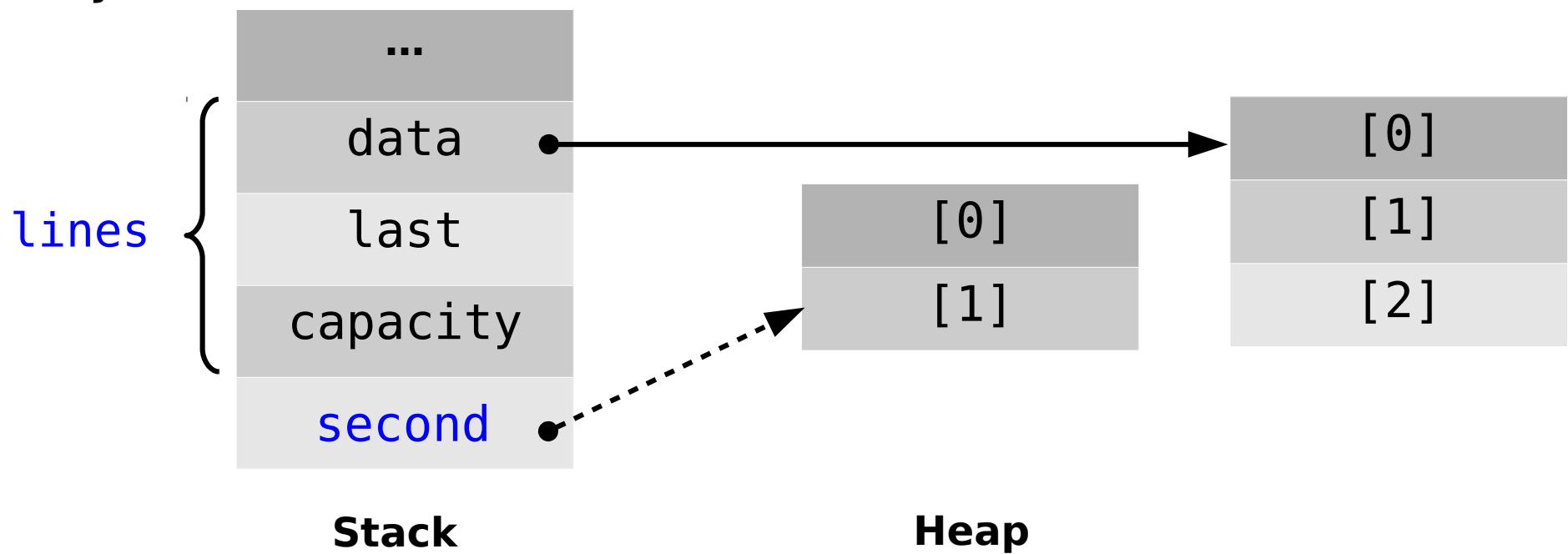
# Memory Unsafety

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    lines.push_back("World!");
    ...
}
```



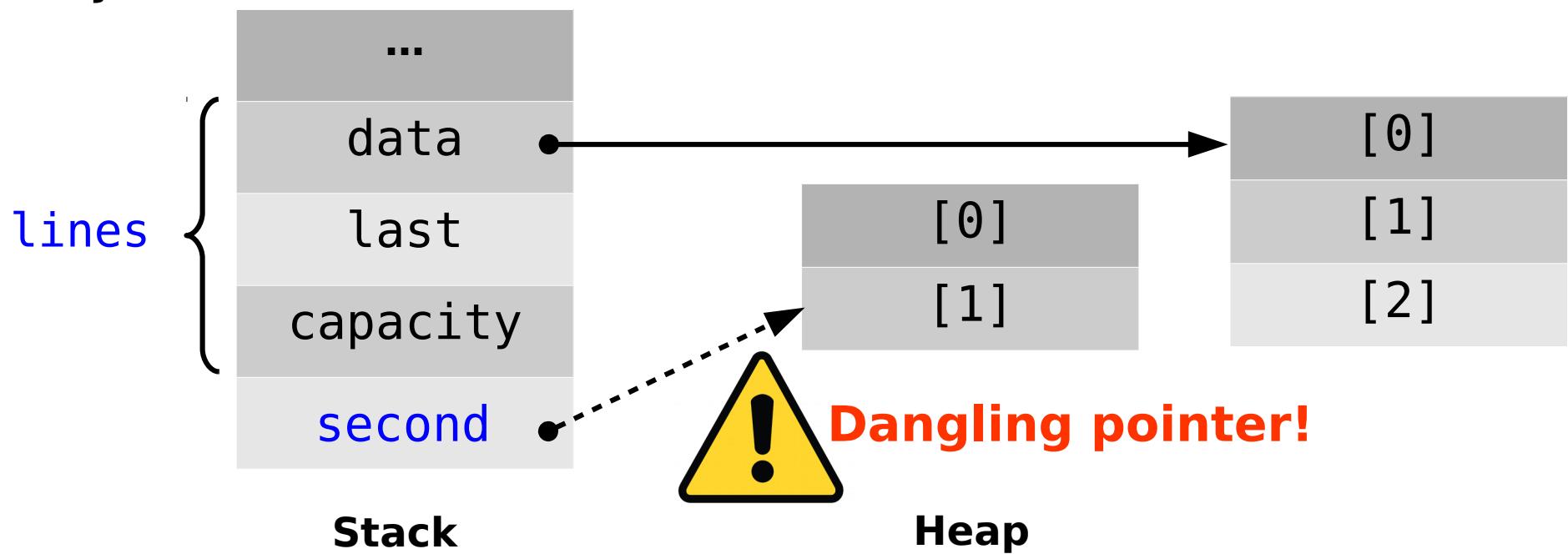
# Memory Unsafety

```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    lines.push_back("World!");
    ...
}
```



# Memory Unsafety

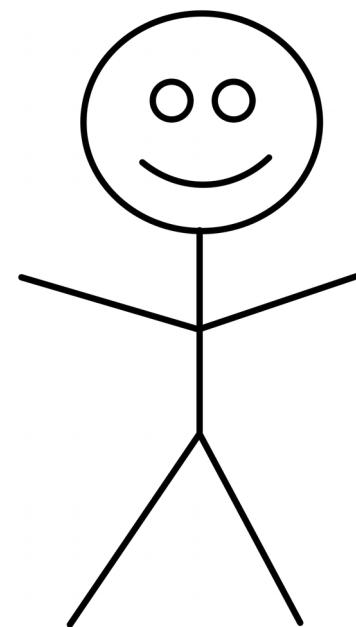
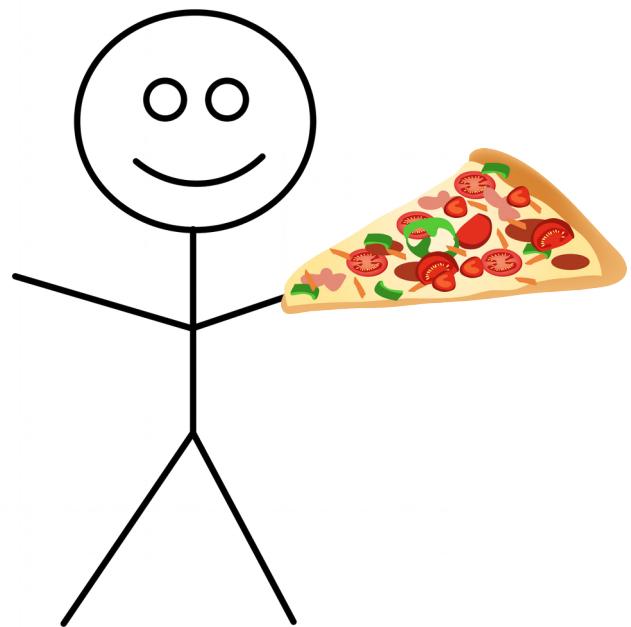
```
int main() {
    vector<string> lines;
    ...
    auto& second = lines[1];
    lines.push_back("World!");
    ...
}
```



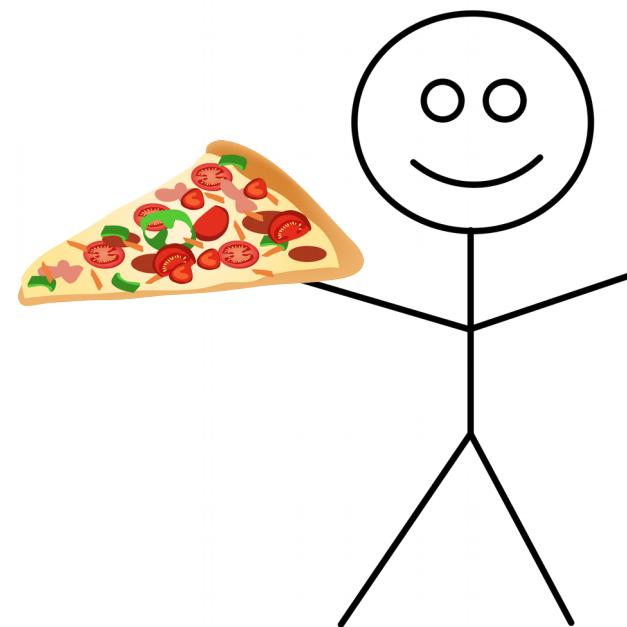
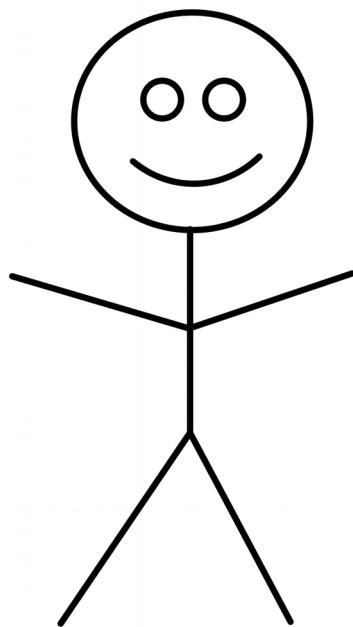
# Ownership

| Type | Owns | Can Alias | Can Mutate |
|------|------|-----------|------------|
| T    | ✓    |           | ✓          |

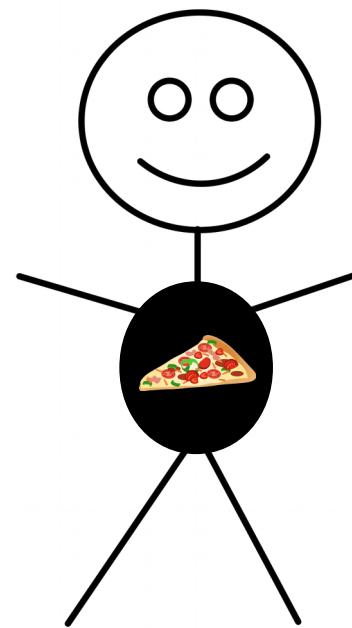
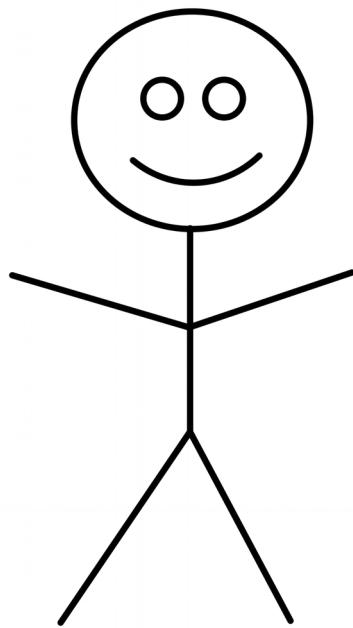
# Ownership Transfer



# Ownership Transfer



# Ownership Transfer





# Ownership Transfer

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
  
}
```



# Ownership Transfer

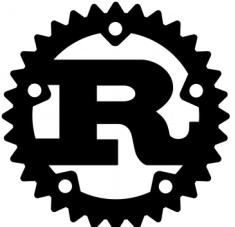
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
  
}  
  
fn feed(pizza: Vec<Zutat>) {  
    // yum yum  
}
```



# Ownership Transfer

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza); ← Give ownership.  
}  
~~~~~
```

```
fn feed(pizza: Vec<Zutat>) {  
    // yum yum  
}
```



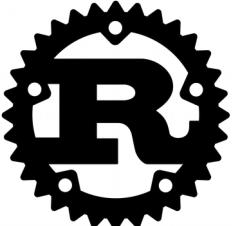
# Ownership Transfer

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza); ← Give ownership.  
}  
~~~~~
```

```
fn feed(pizza: Vec<Zutat>) {  
    // yum yum  
}
```



Take ownership.



# Ownership Transfer

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza); ← Give ownership.  
    feed(pizza);  
}
```

```
fn feed(pizza: Vec<Zutat>) {  
    // yum yum  
}
```

Take ownership.



# Ownership Transfer

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza); ← Give ownership.  
    ⚡feed(pizza);  
}
```

```
fn feed(Vec<Zutat>) {
```

error[E0382]: use of moved value: `pizza`

Take ownership.





# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(pizza);
    feed(pizza);
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(pizza); ← Kopie erstellen.  
    feed(pizza);  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(pizza); ← Kopie erstellen.  
    feed(pizza);  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```

Kopie nehmen.



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(pizza);
    feed(pizza);
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza); ← Nächste Kopie erstellen.  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza); ← Nächste Kopie erstellen.  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```

Kopie nehmen.



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza)); ← Ownership geben.
    feed(std::move(pizza));
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(std::move(pizza)); ← Ownership geben.  
    feed(std::move(pizza));  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```

Ownership nehmen.



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza)); ← Ownership geben.
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```



# Ownership Transfer

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza)); ← Ownership geben.
}
```

```
void feed(vector<Zutat> pizza) {
    // yum yum
}
```

Ownership nehmen.



# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(std::move(pizza));  
    feed(std::move(pizza)); ← Ownership geben.  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```

Ownership nehmen.

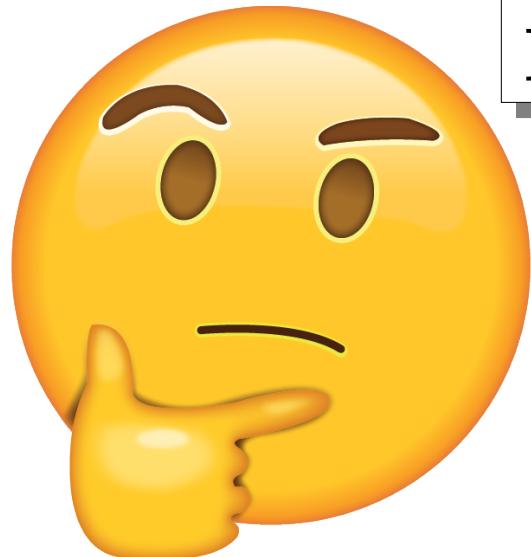




# Ownership Transfer

```
int main() {  
    vector<Zutat> pizza;  
    pizza.push_back(Zutat::Salami);  
    ...  
    feed(std::move(pizza));  
    feed(std::move(pizza)); ← Ownership geben.  
}
```

```
void feed(vector<Zutat> pizza) {  
    // yum yum  
}
```



Ownership nehmen.



# Non-destructive Moves

A moved object  
must be in a valid state afterwards.



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

**Stack**

**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

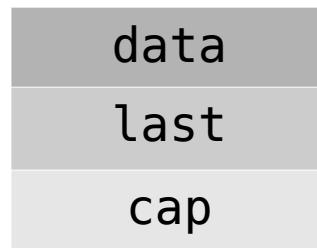
**Stack**

**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```



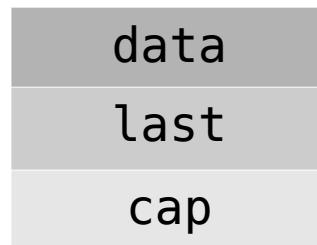
**Stack**

**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```



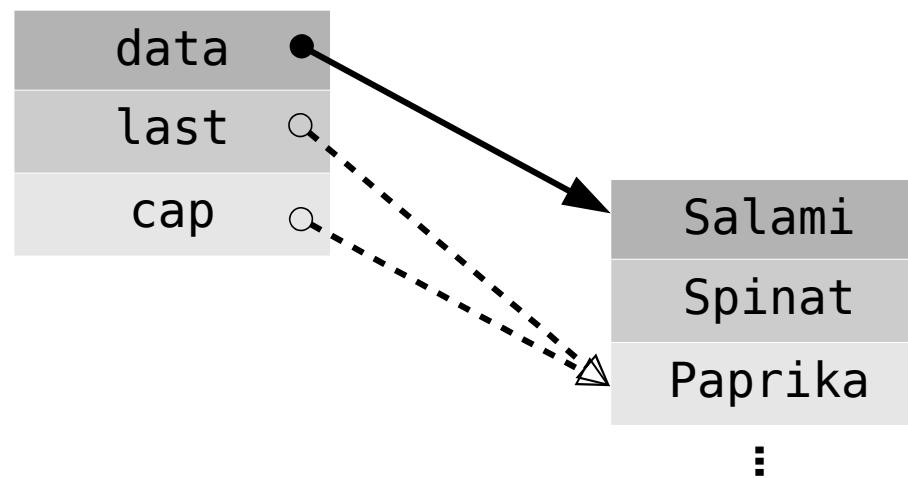
**Stack**

**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```



**Stack**

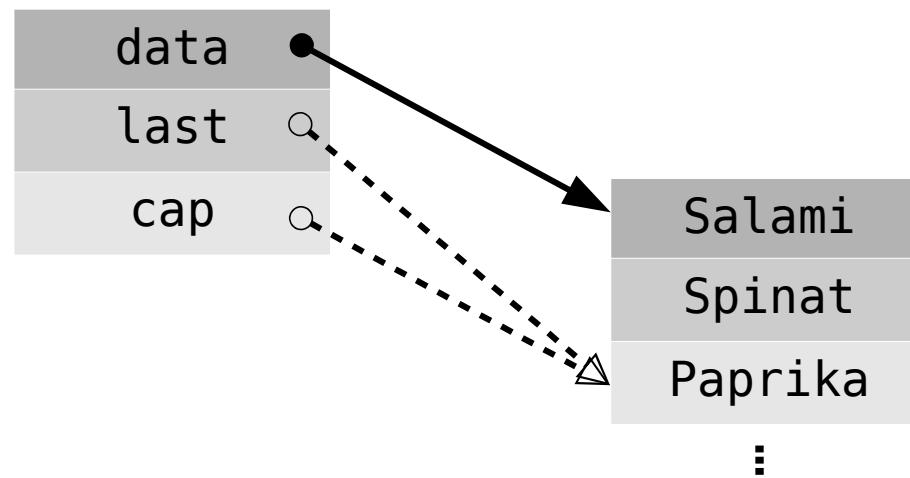
**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);

    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```



**Stack**

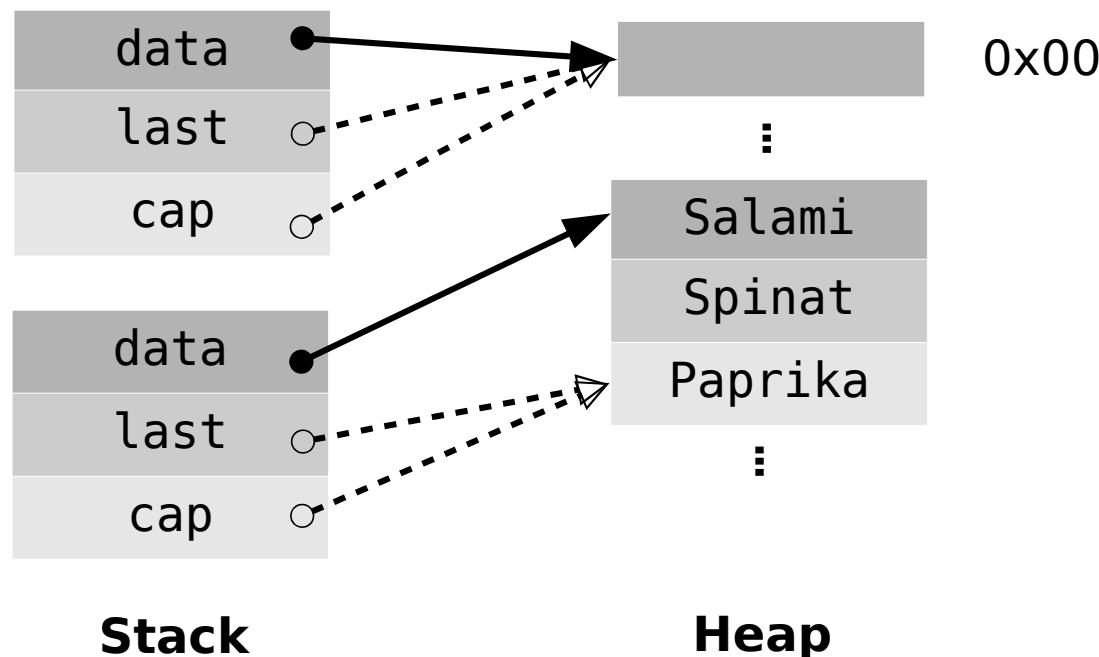
**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);

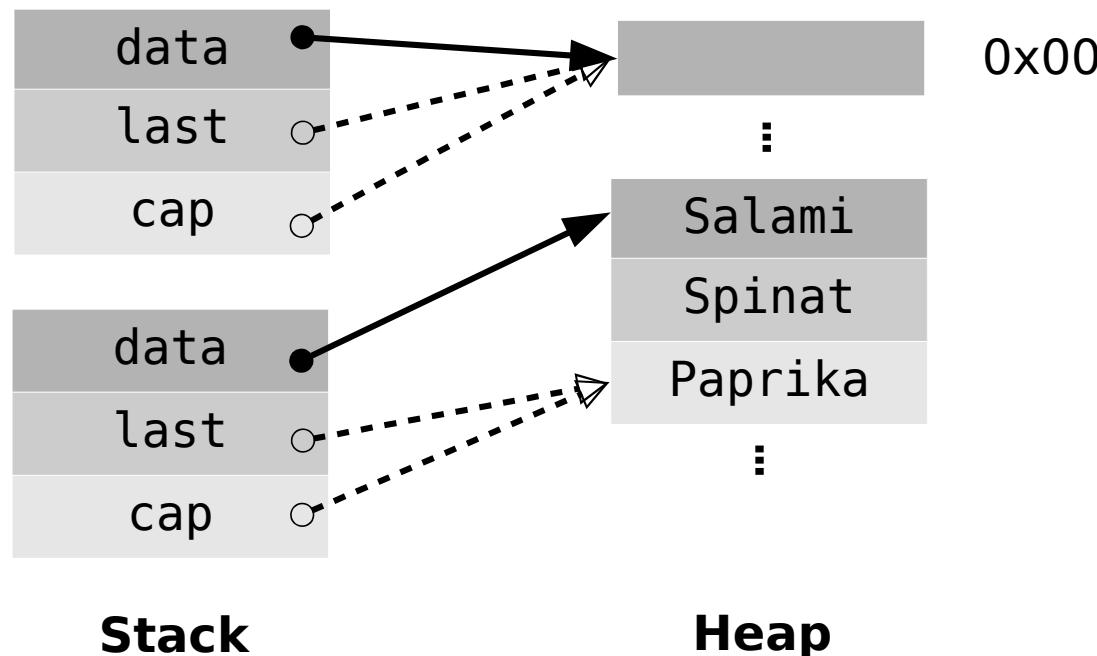
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

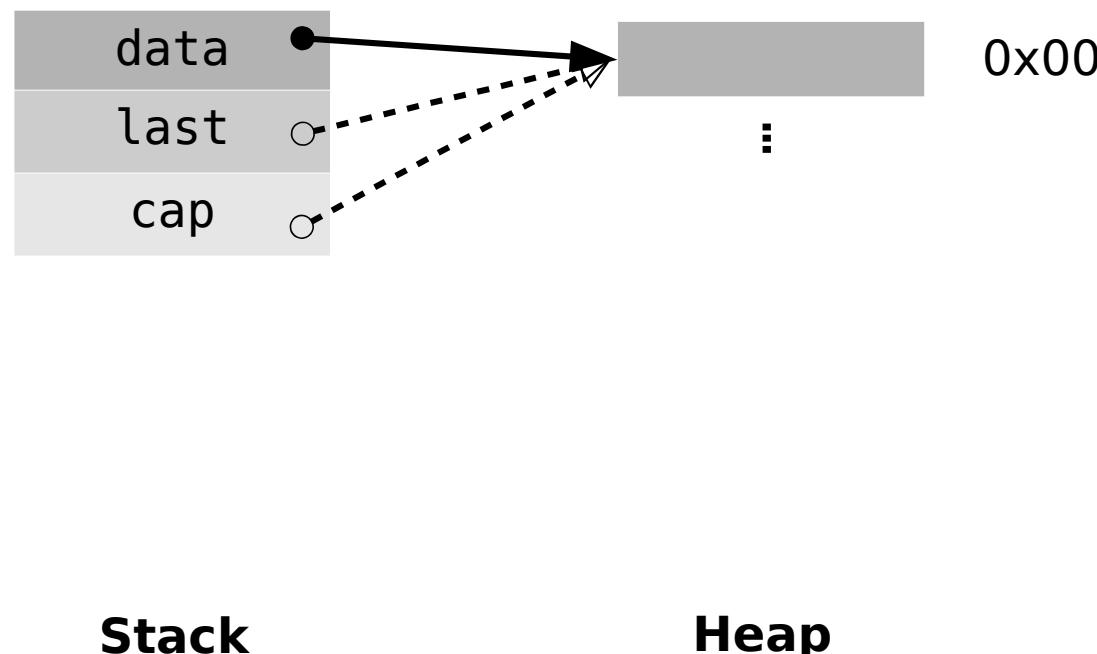
```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

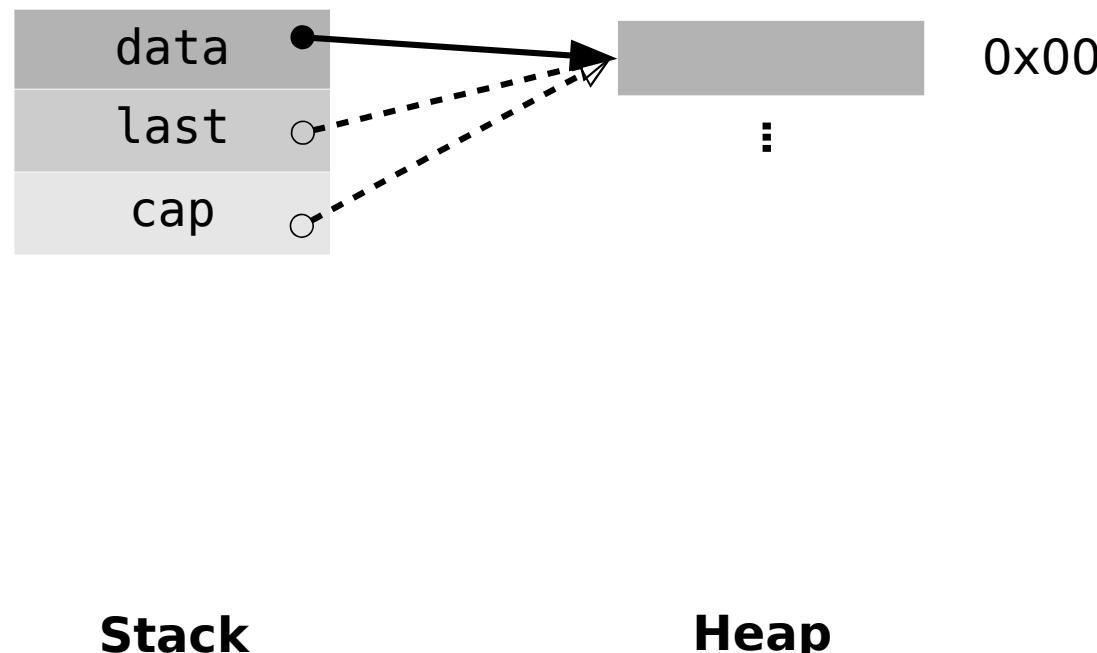
```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

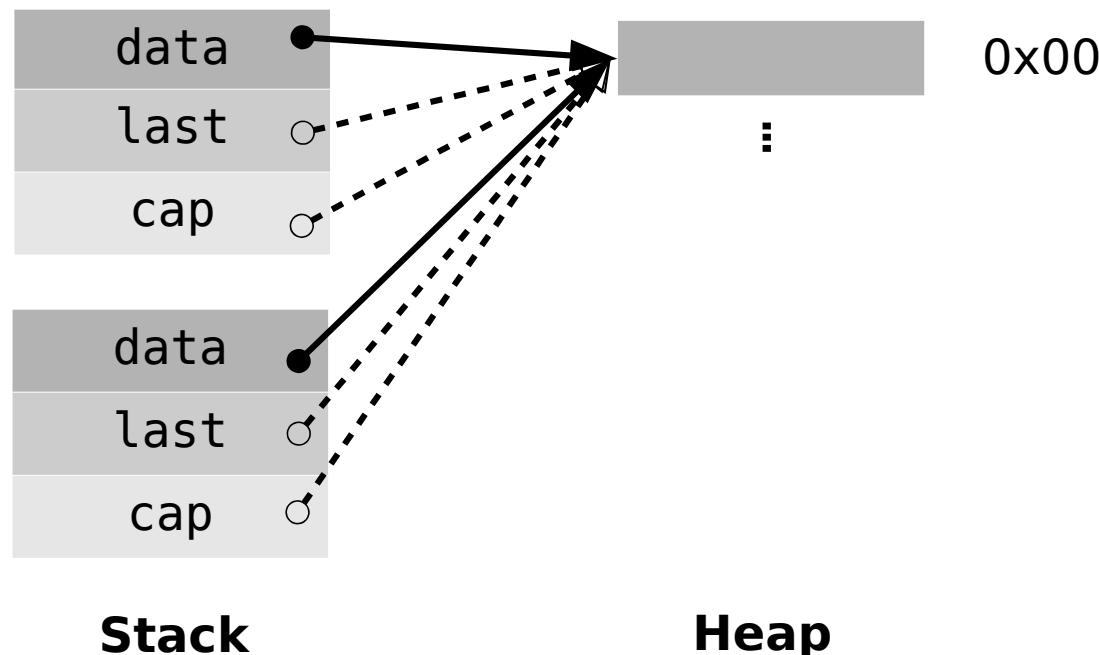




# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);

    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

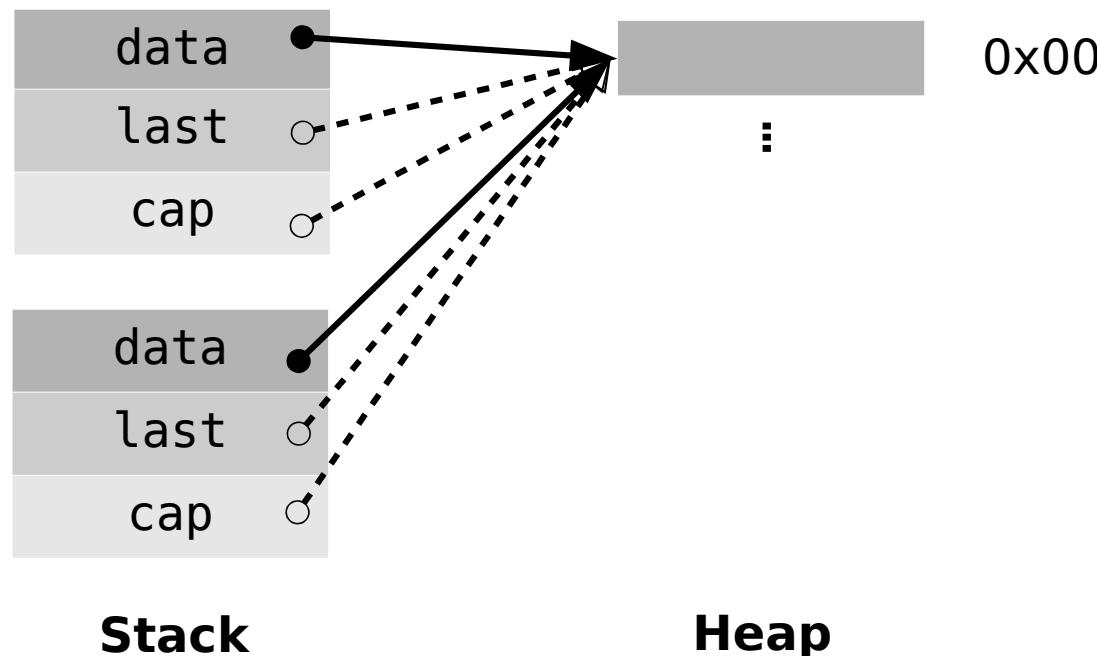




# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);

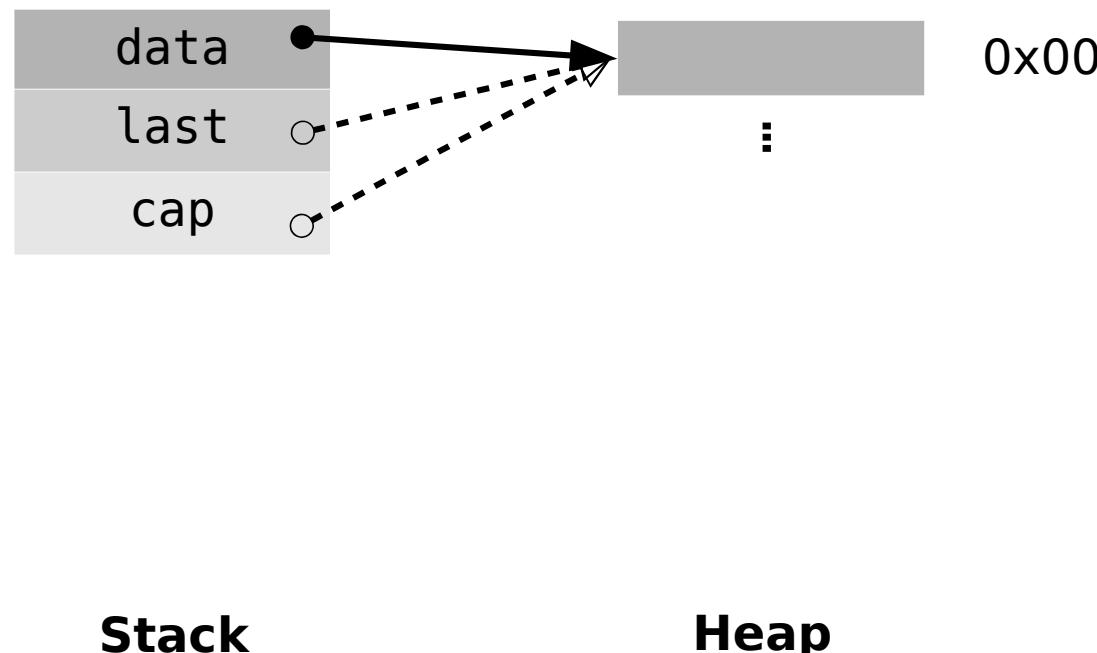
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

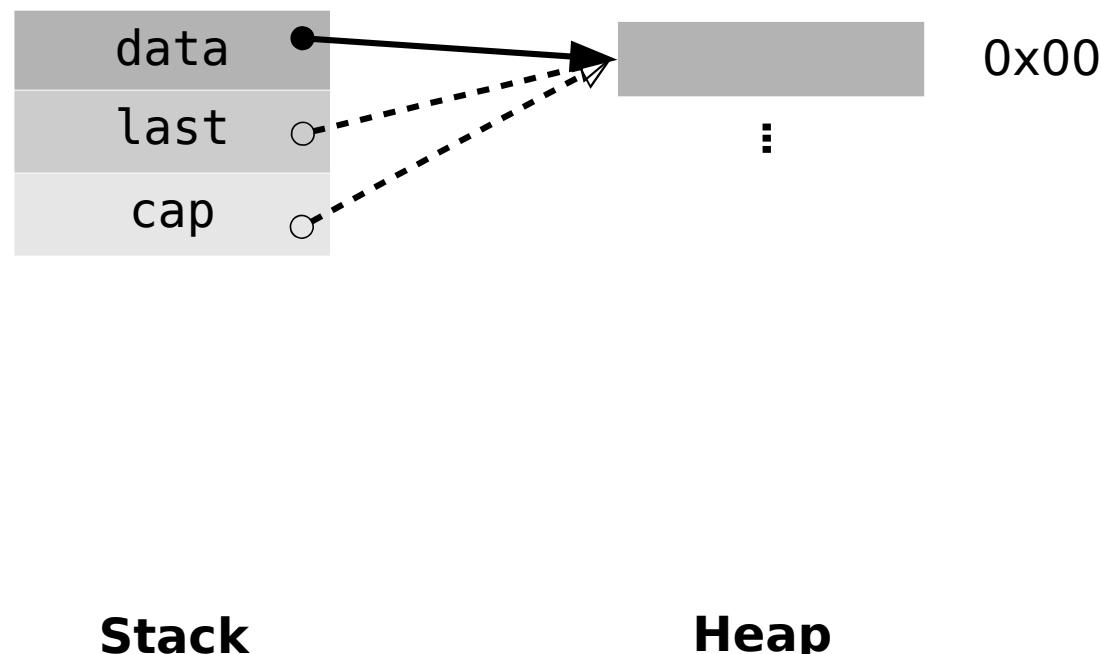
```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```





# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```



**Stack**

**Heap**



# Non-destructive Moves

```
int main() {
    vector<Zutat> pizza;
    pizza.push_back(Zutat::Salami);
    ...
    feed(std::move(pizza));
    feed(std::move(pizza));
}
```

IMPLEMENTATION DEFINED

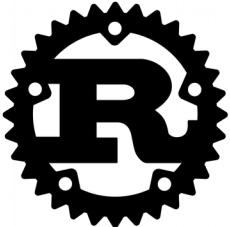
Stack

Heap



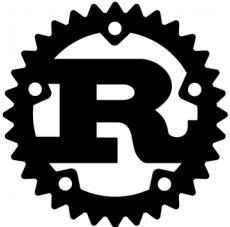
# Destructive Moves

A moved object  
must not be accessed afterwards.



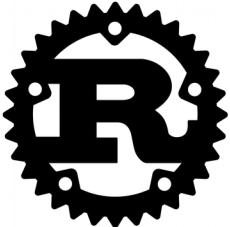
# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```



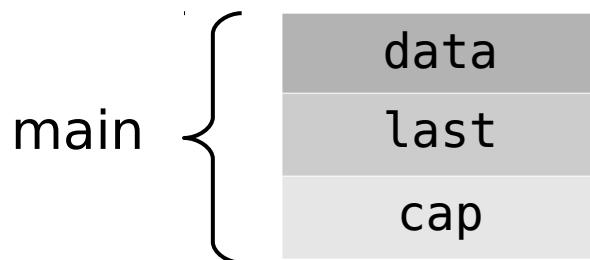
# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```

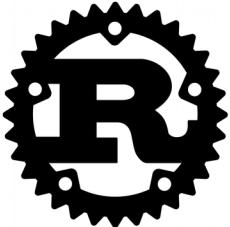


# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```

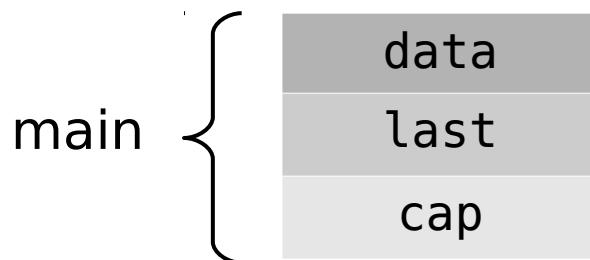


**Stack**



# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```

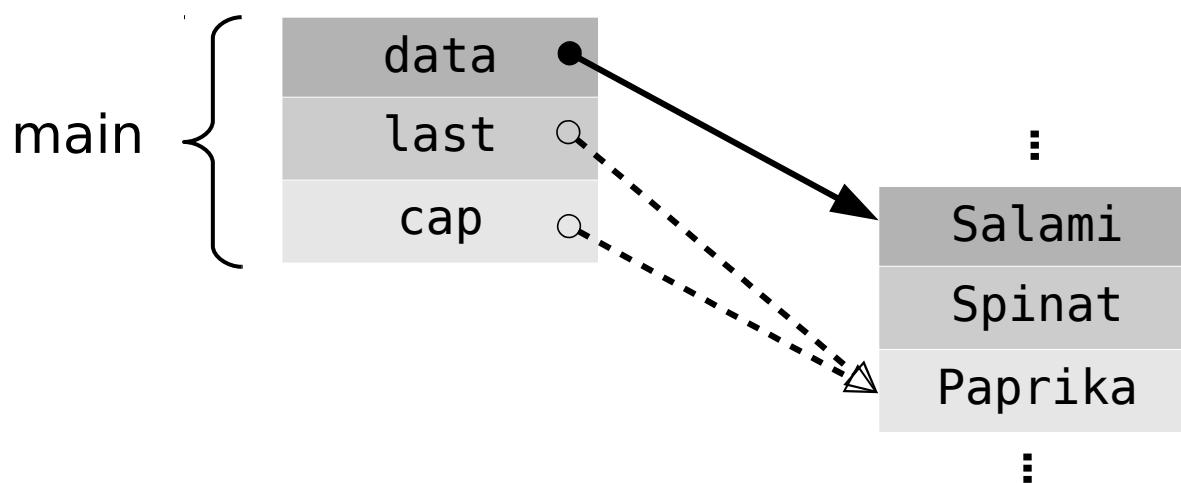


**Stack**



# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```



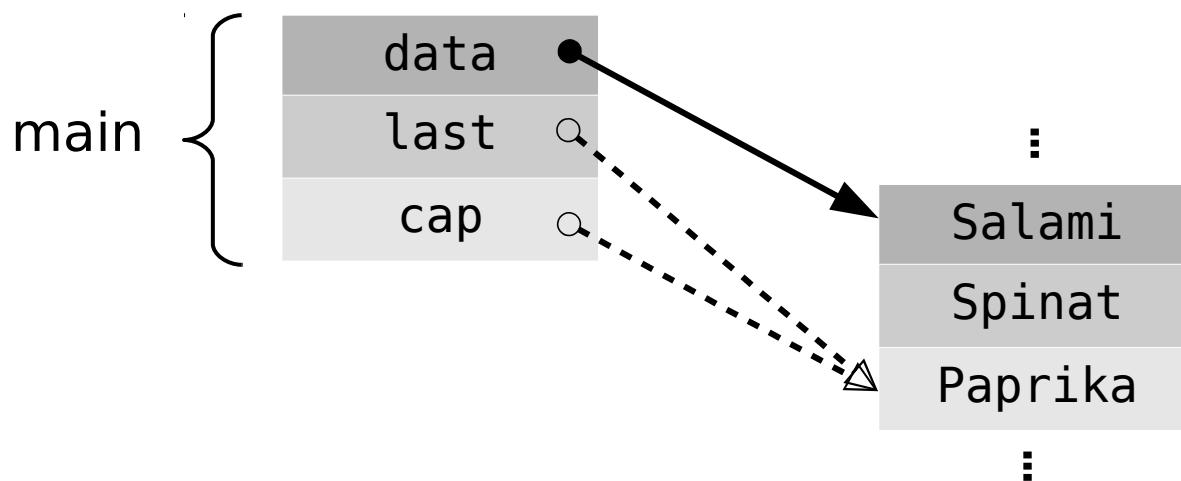
**Stack**

**Heap**



# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```



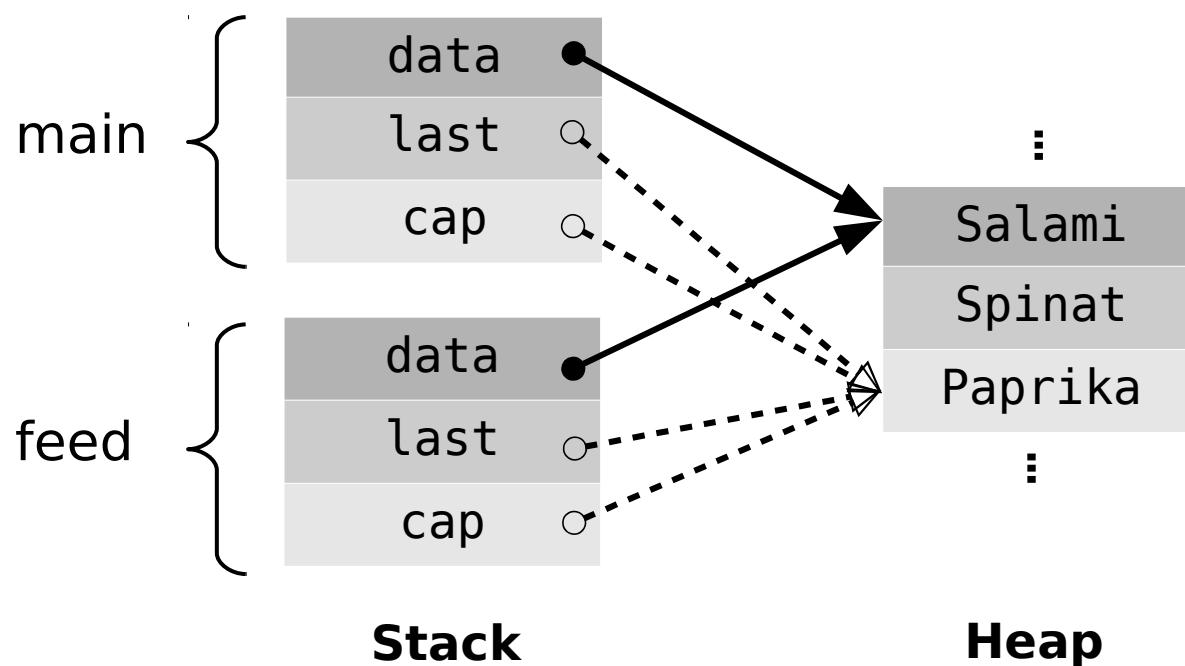
**Stack**

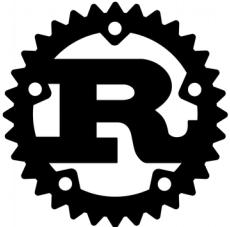
**Heap**



# Destructive Moves

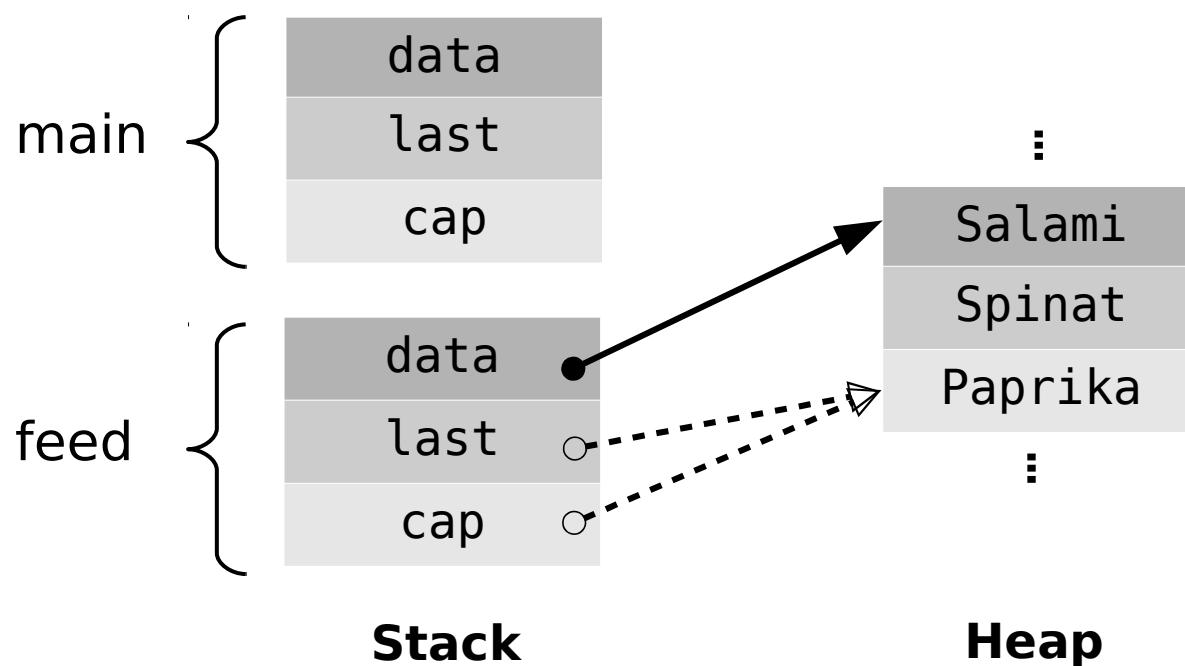
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```

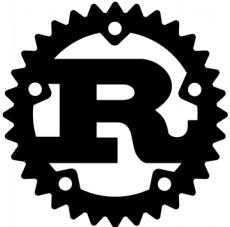




# Destructive Moves

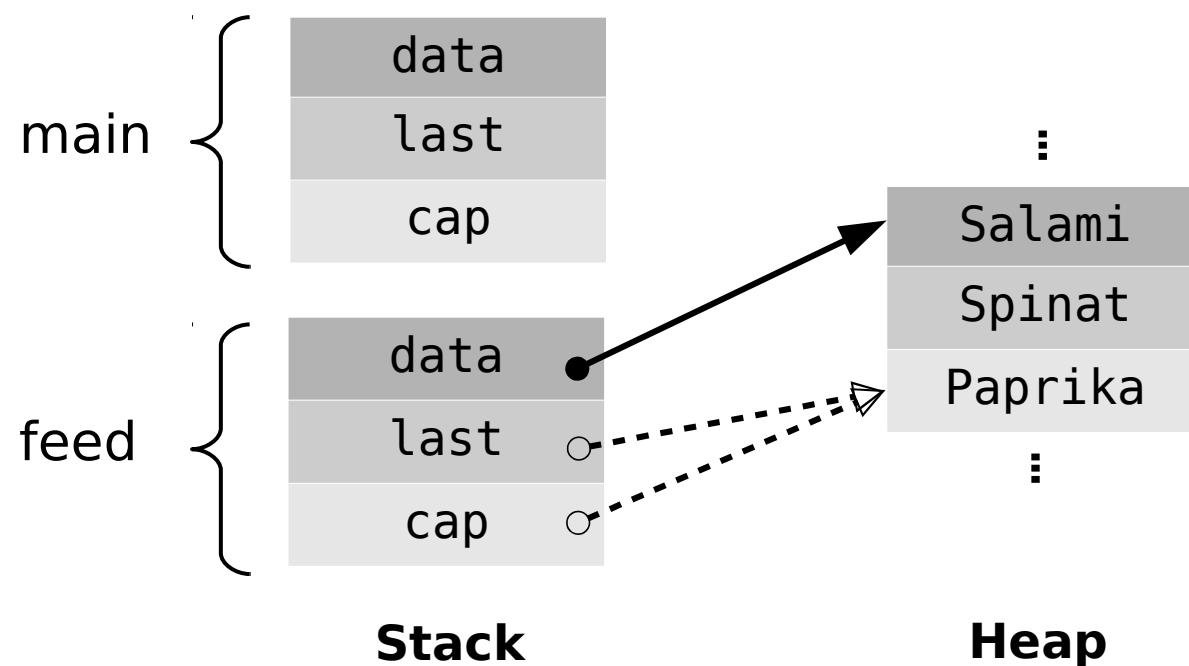
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```

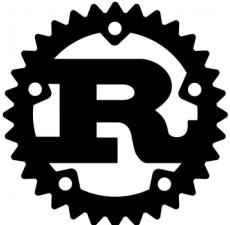




# Destructive Moves

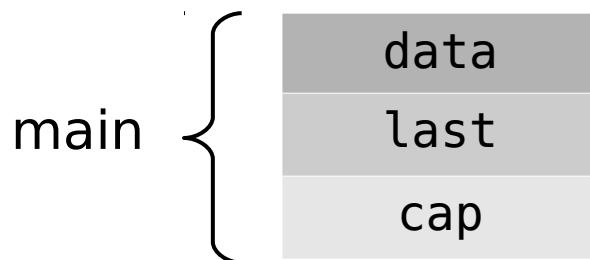
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```





# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza);  
}
```



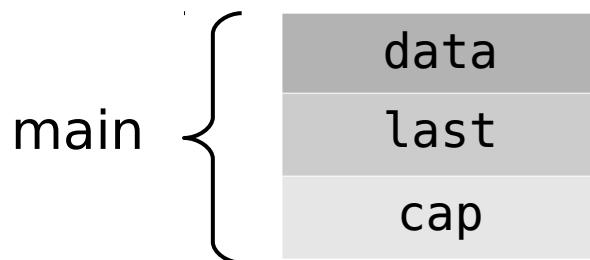
**Stack**

**Heap**



# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza);  
    feed(pizza) //error[E0382]: use of moved value: `pizza`  
}
```



Stack

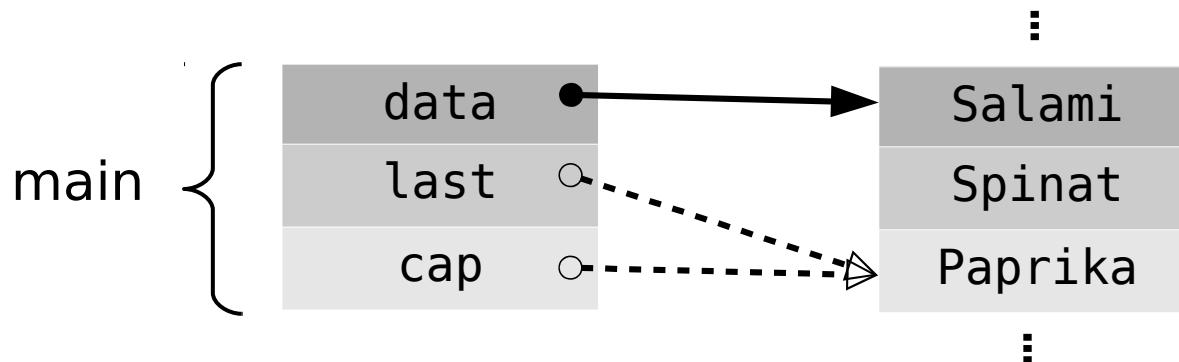
Heap

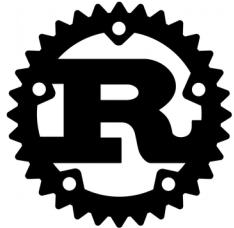




# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

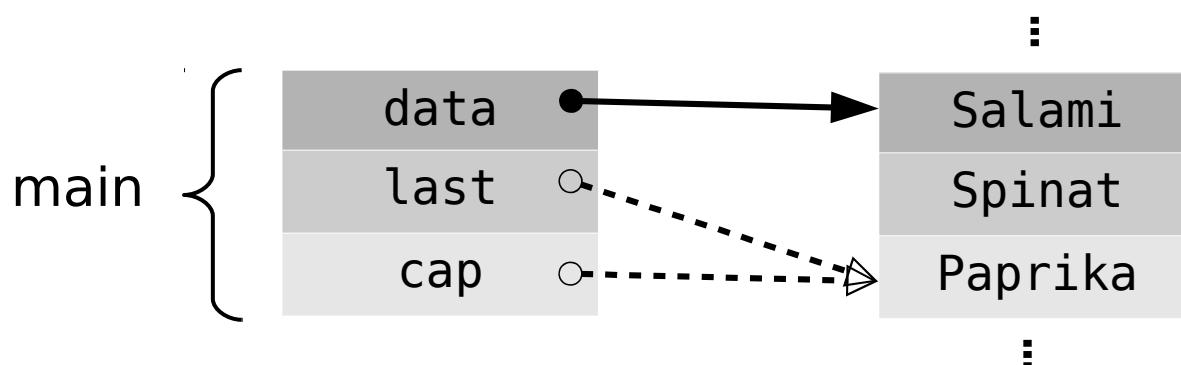




# Destructive Moves

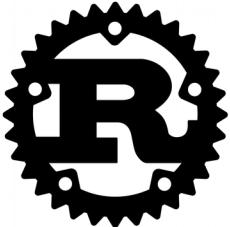
```
fn main() {
    let mut pizza = Vec::new();
    pizza.push(Zutat::Salami);

    ...
    feed(pizza.clone());
    feed(pizza);
}
```



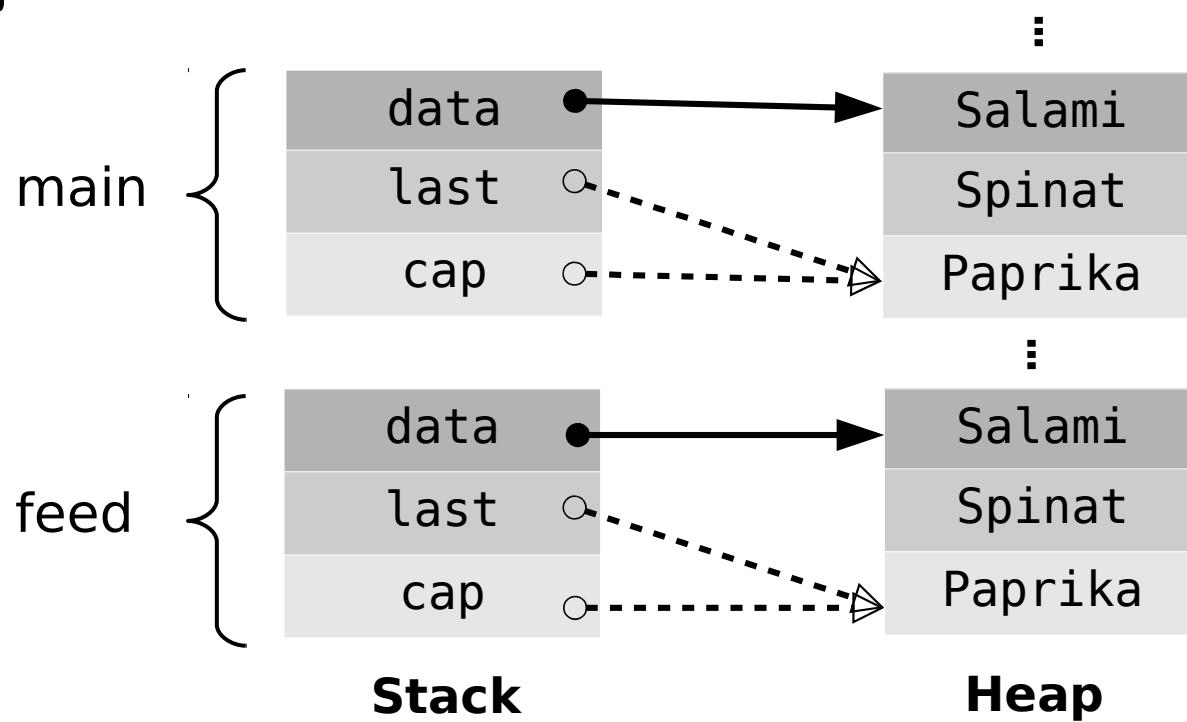
# Stack

## Heap



# Destructive Moves

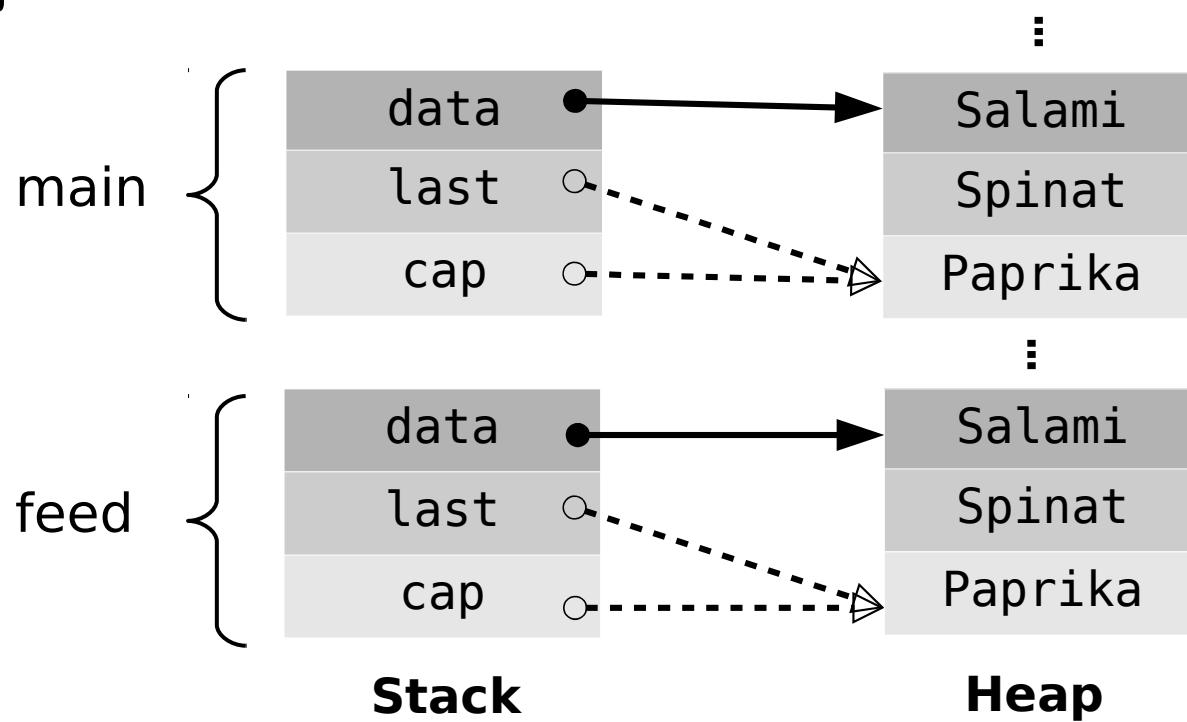
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

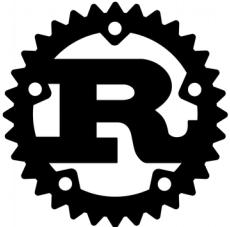




# Destructive Moves

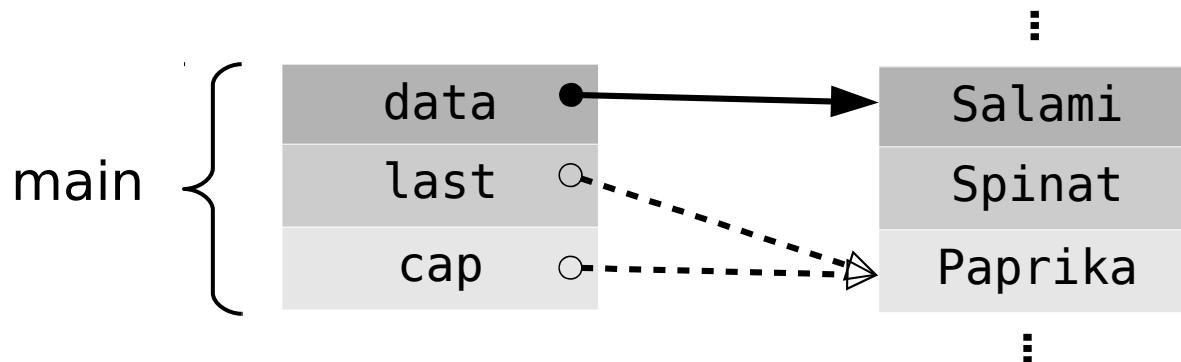
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```





# Destructive Moves

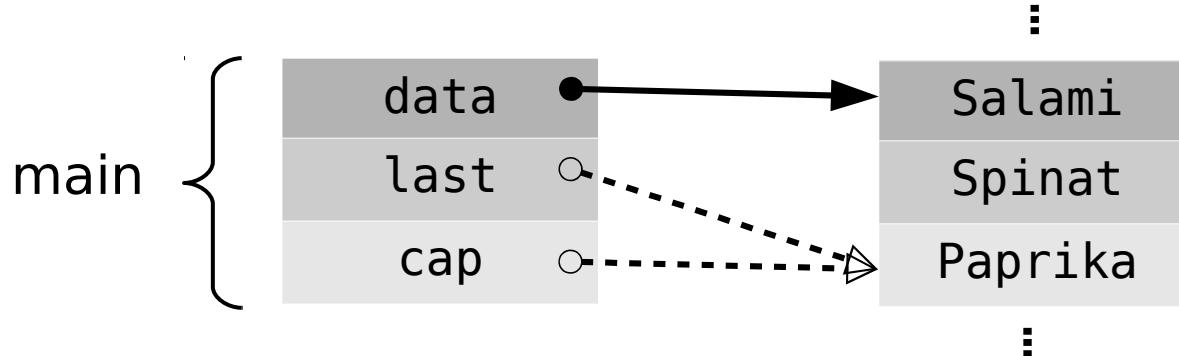
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```





# Destructive Moves

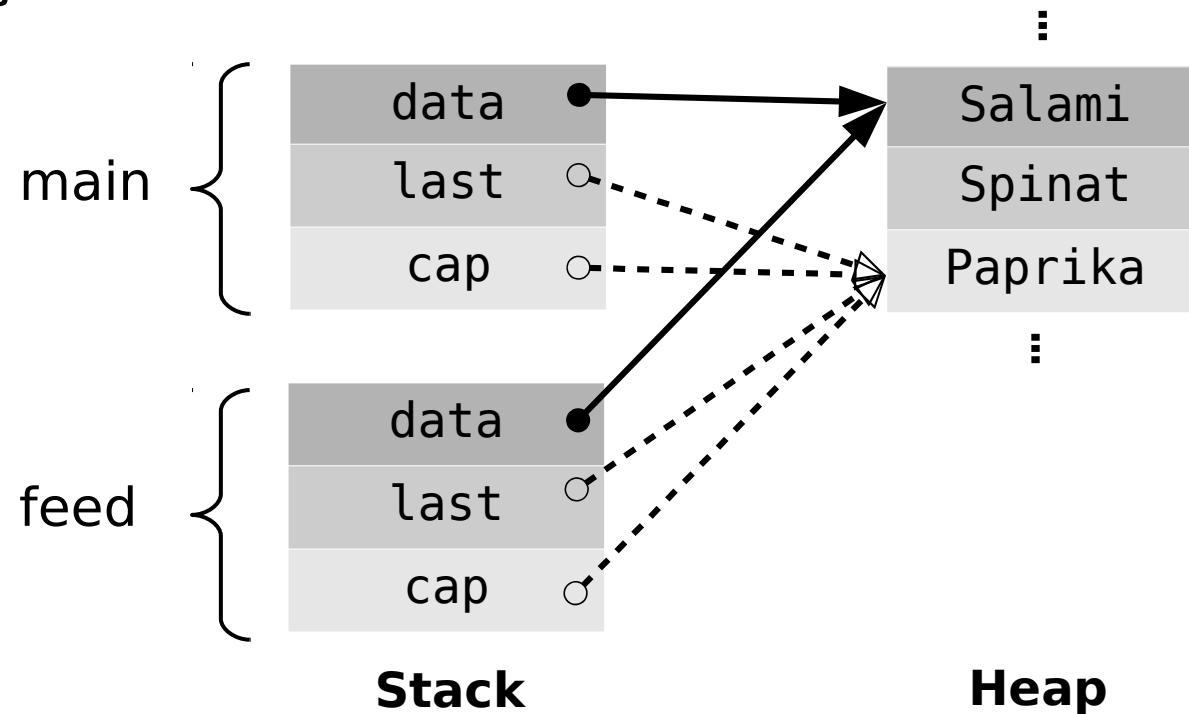
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

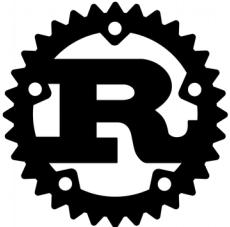




# Destructive Moves

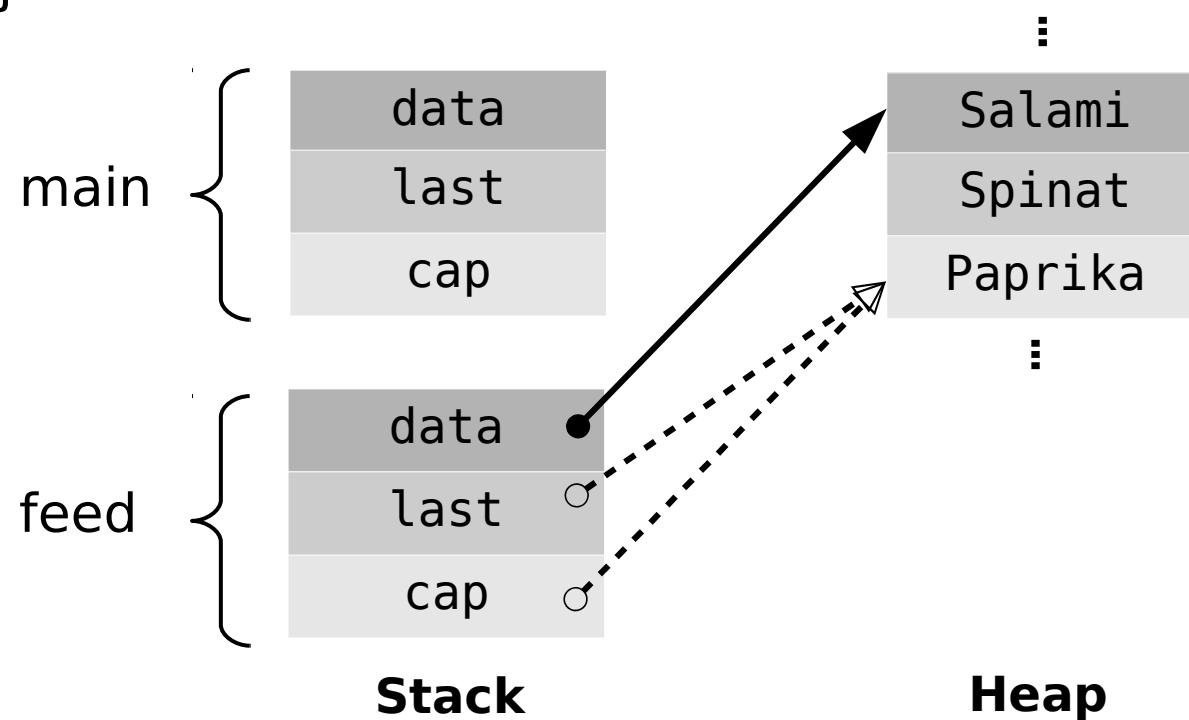
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

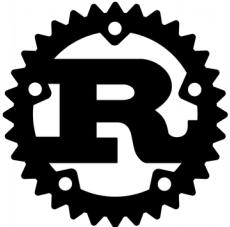




# Destructive Moves

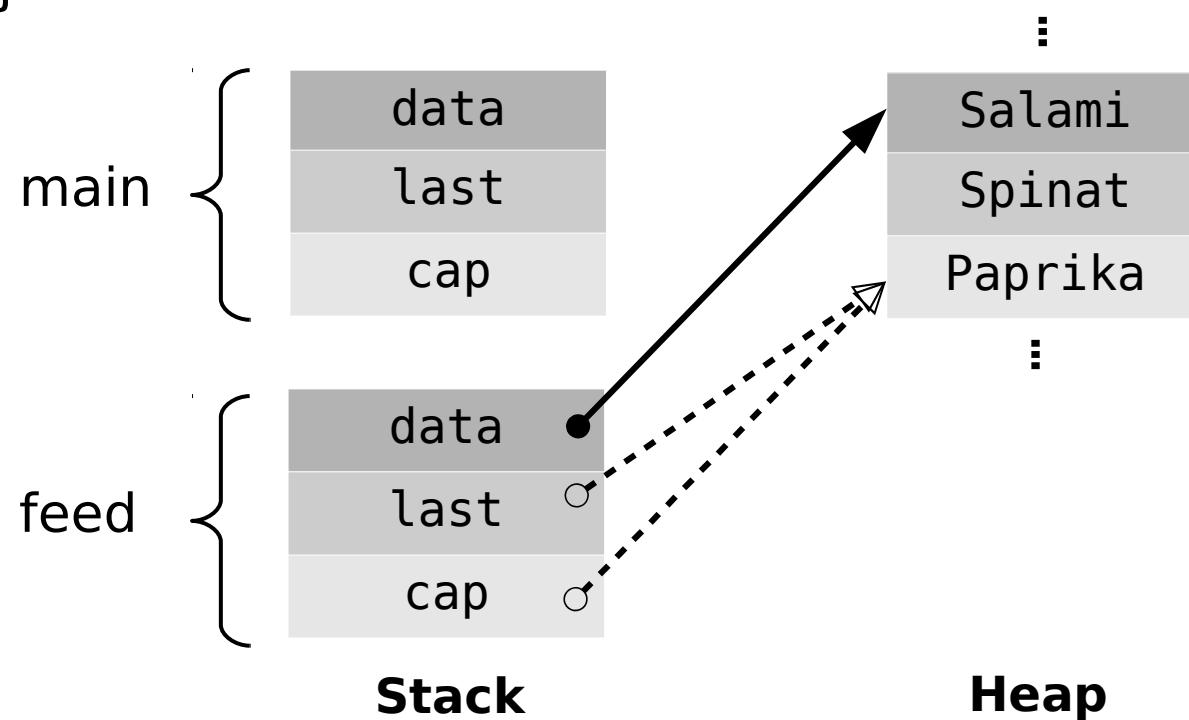
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

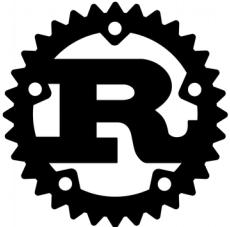




# Destructive Moves

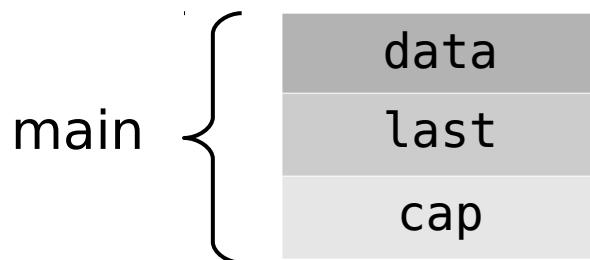
```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```





# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```



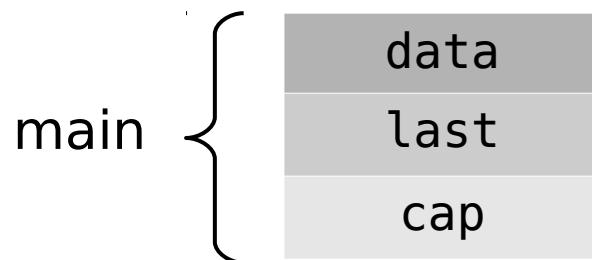
**Stack**

**Heap**



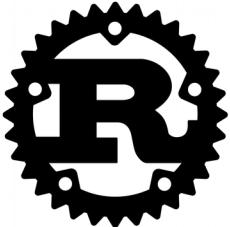
# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```



**Stack**

**Heap**



# Destructive Moves

```
fn main() {  
    let mut pizza = Vec::new();  
    pizza.push(Zutat::Salami);  
  
    ...  
    feed(pizza.clone());  
    feed(pizza);  
}
```

**Stack**

**Heap**

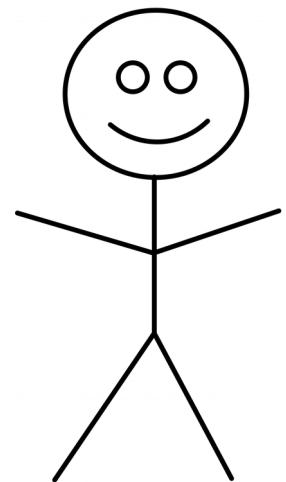
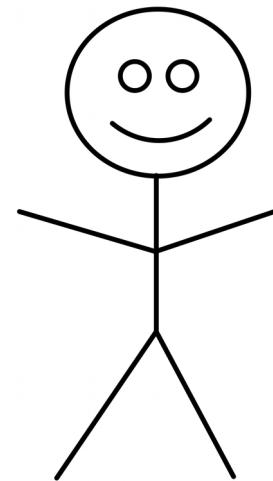
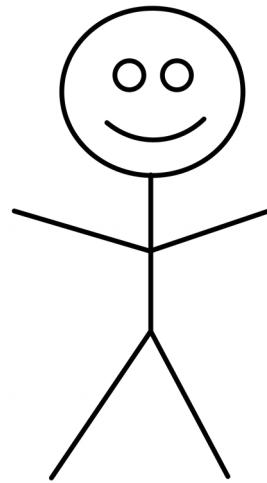
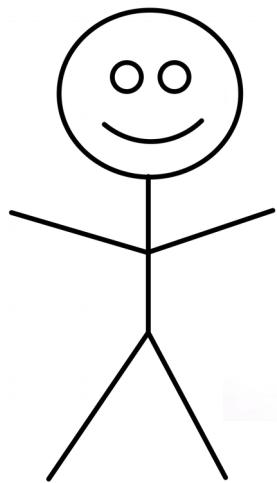
# Shared Borrows

| Type | Owns | May alias | Can mutate |
|------|------|-----------|------------|
| T    | ✓    |           | ✓          |

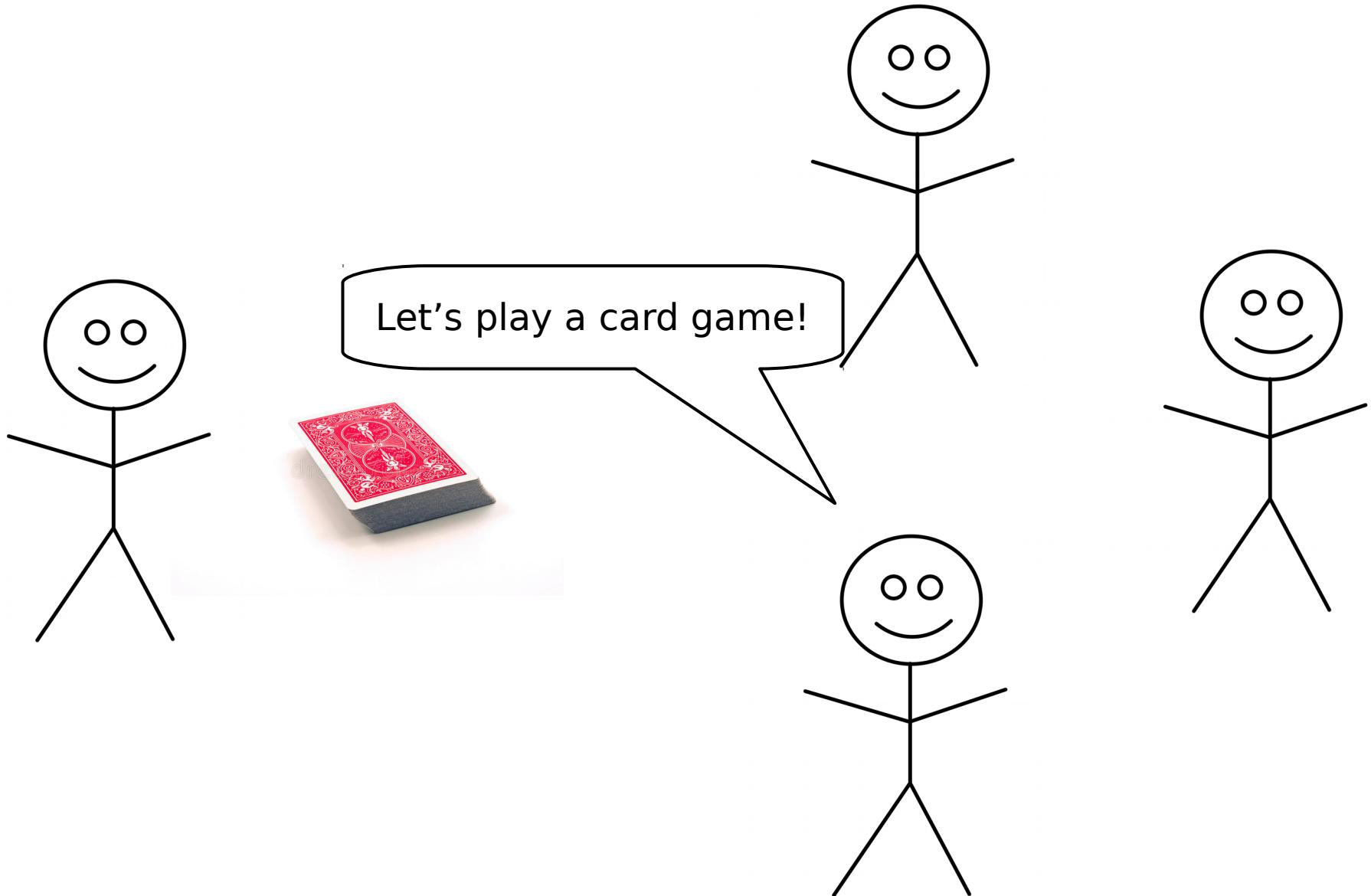
# Shared Borrows

| Type | Owns | May alias | Can mutate |
|------|------|-----------|------------|
| T    | ✓    |           | ✓          |
| &T   |      | ✓         |            |

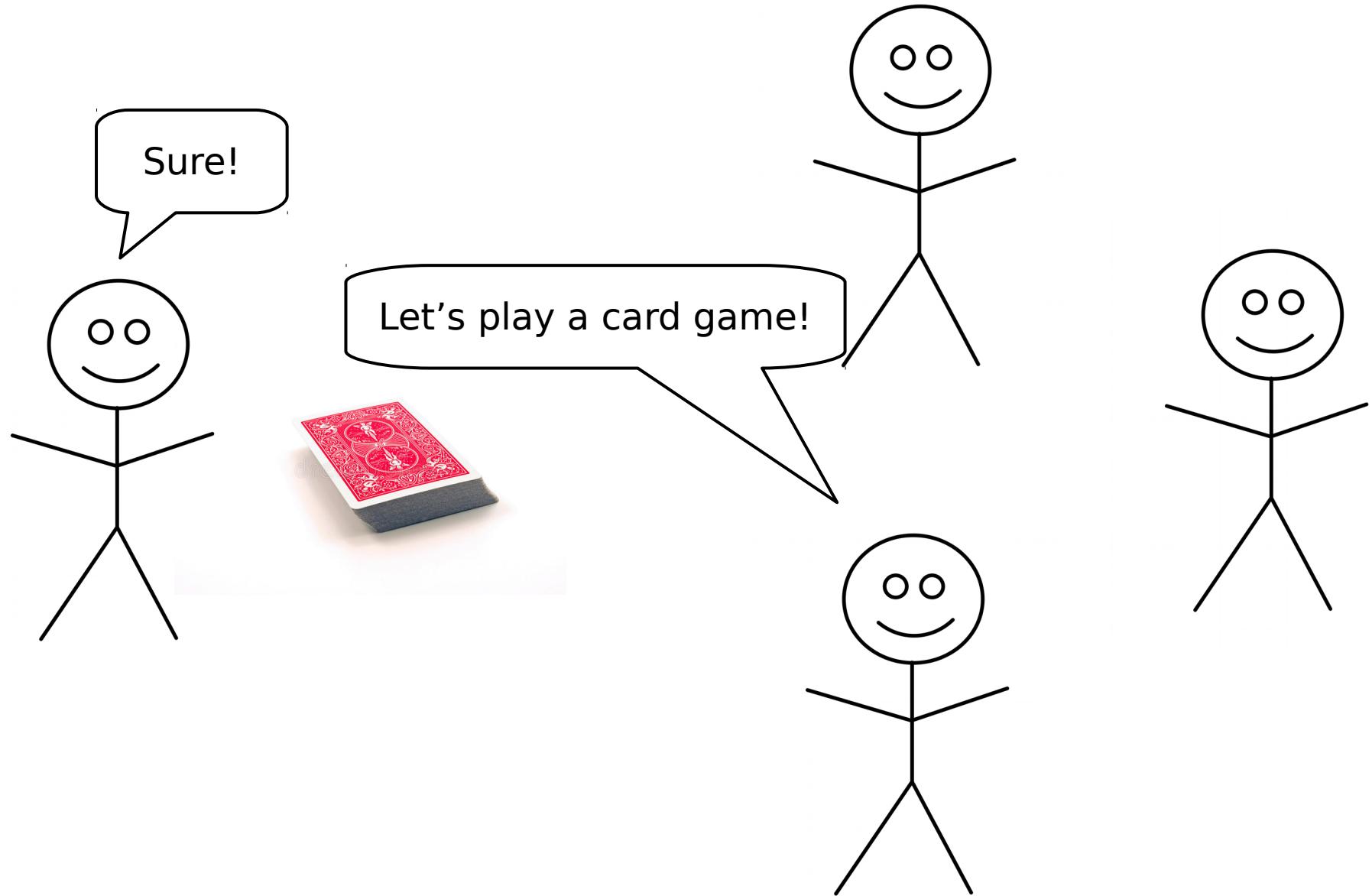
# Shared Borrows



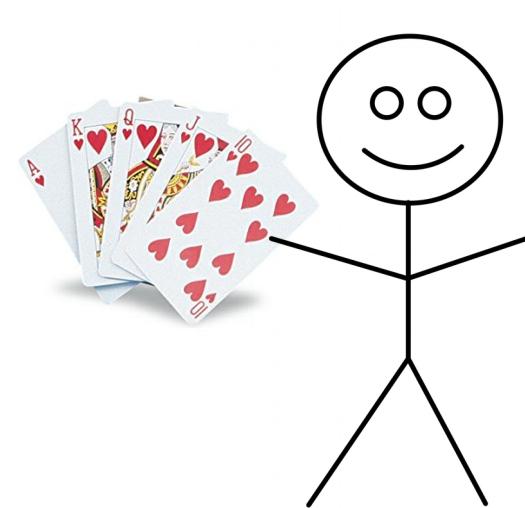
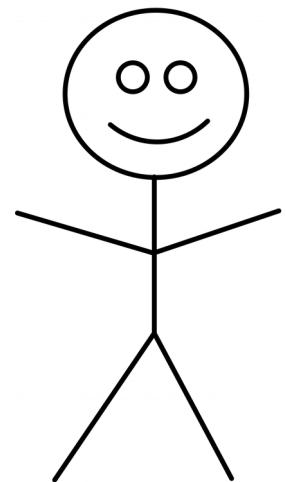
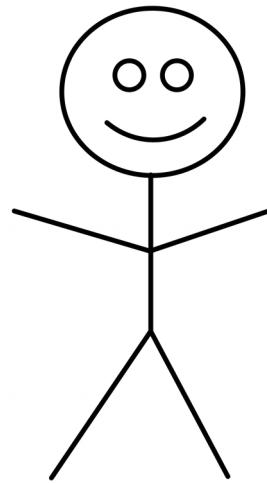
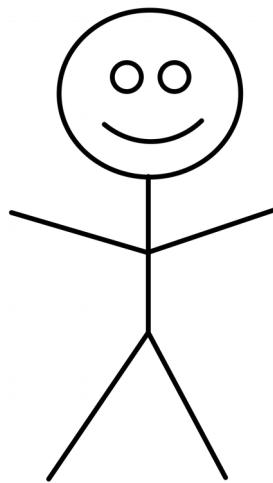
# Shared Borrows



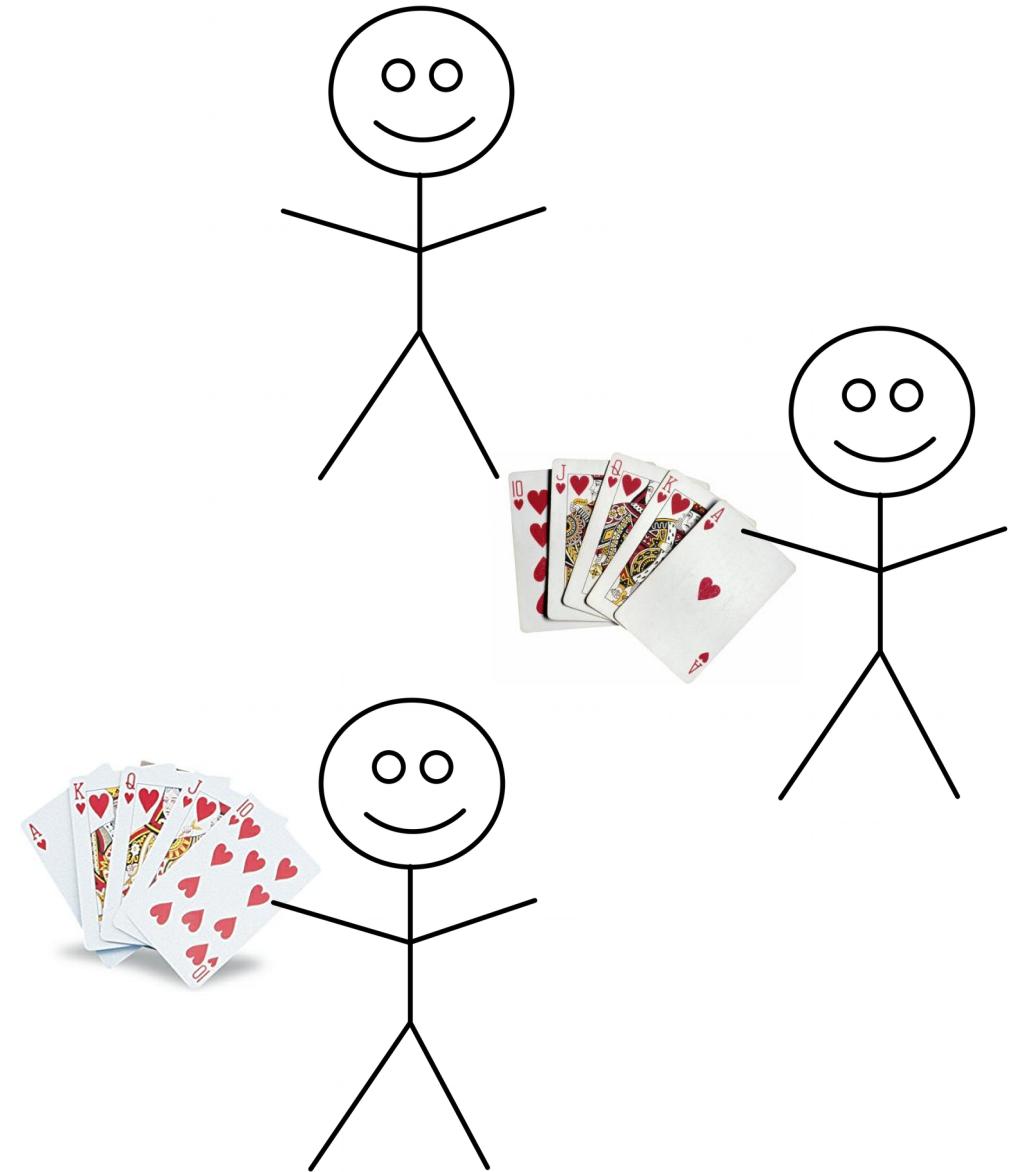
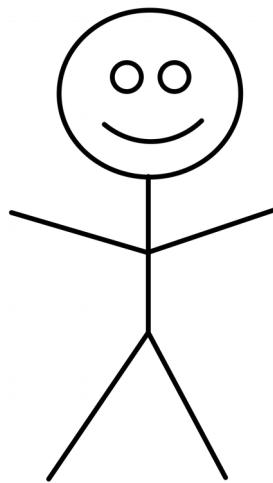
# Shared Borrows



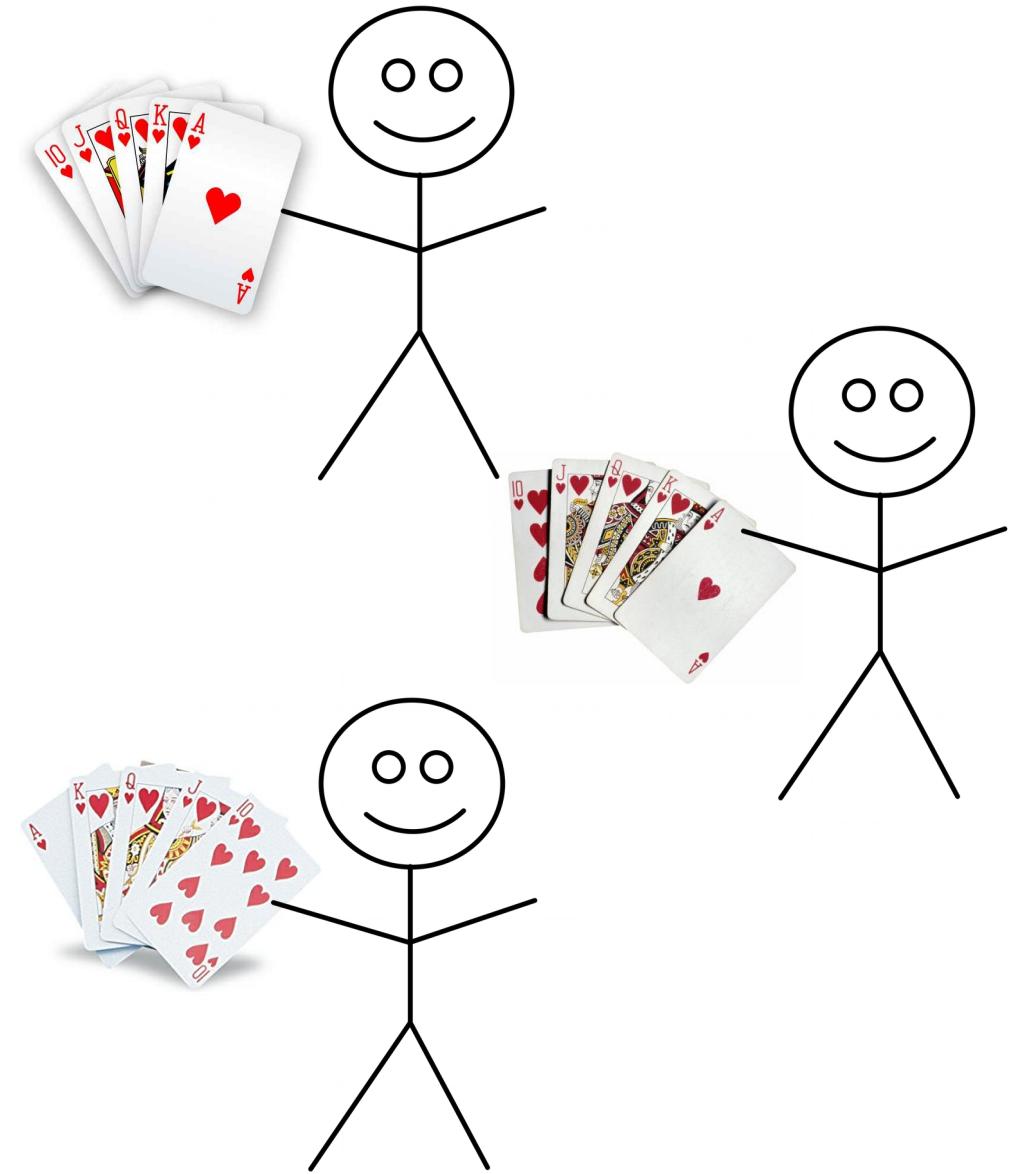
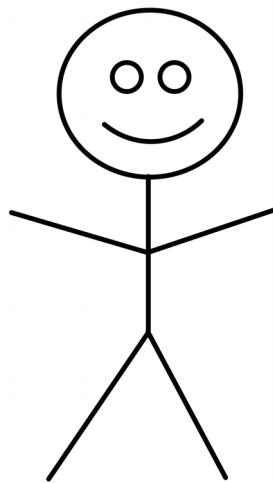
# Shared Borrows



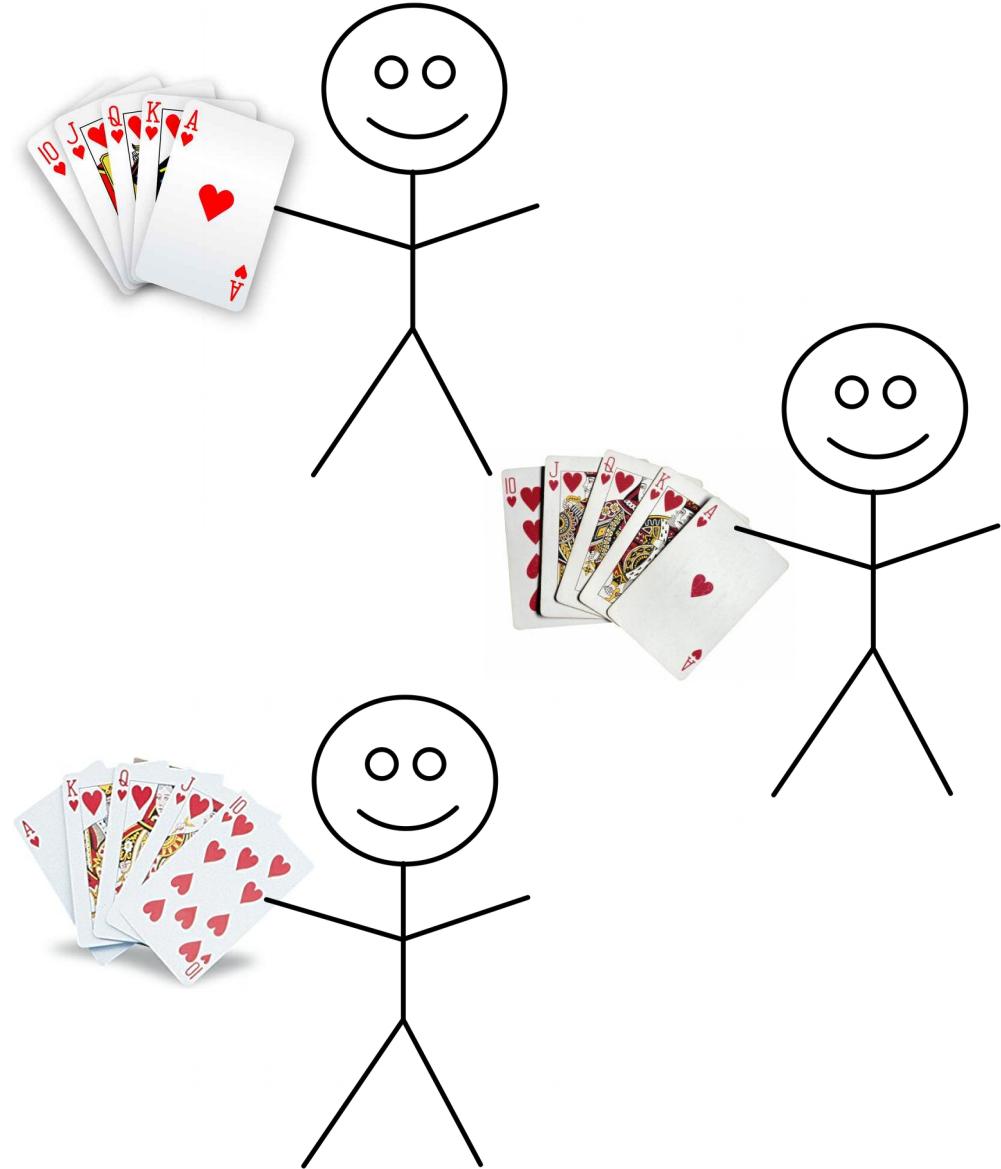
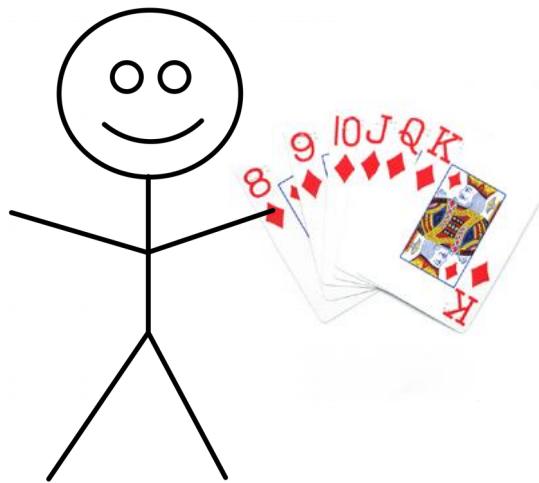
# Shared Borrows



# Shared Borrows



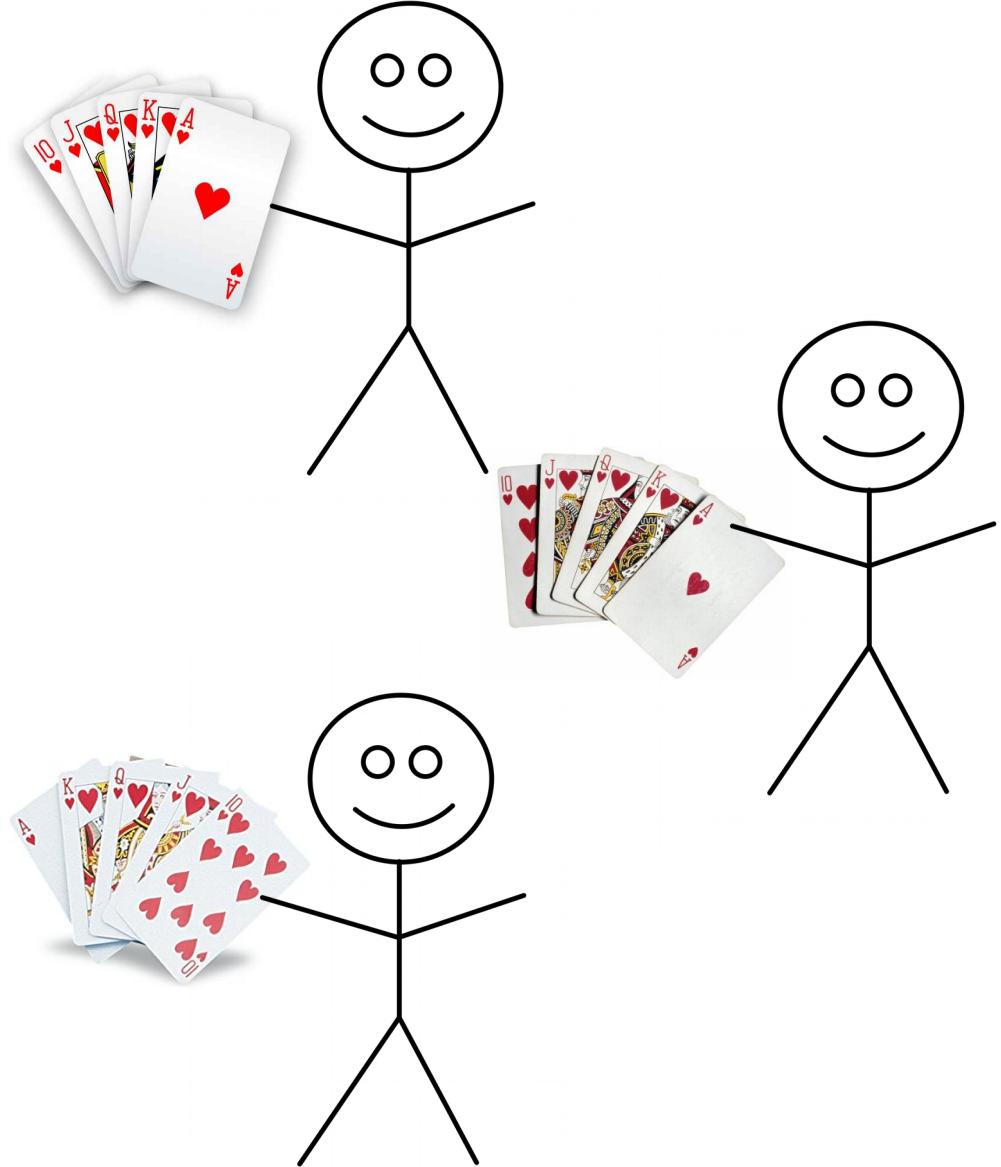
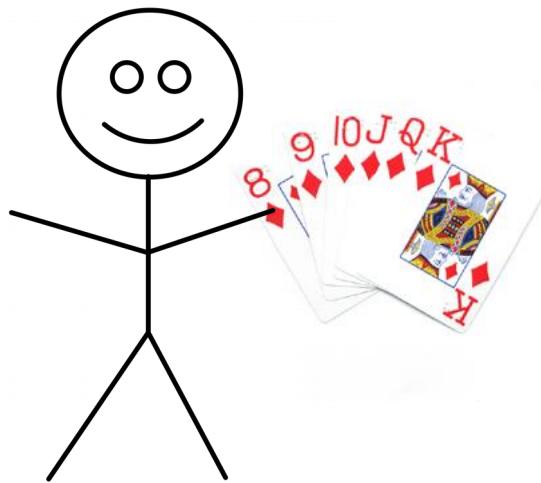
# Shared Borrows



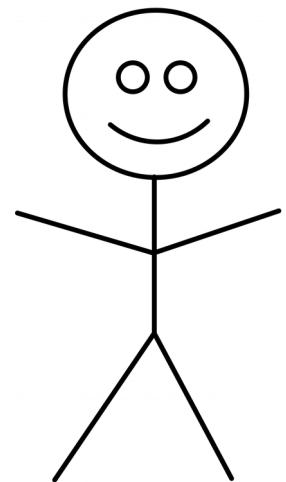
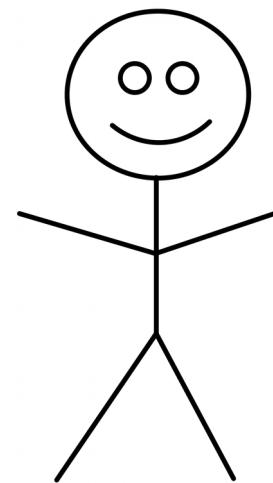
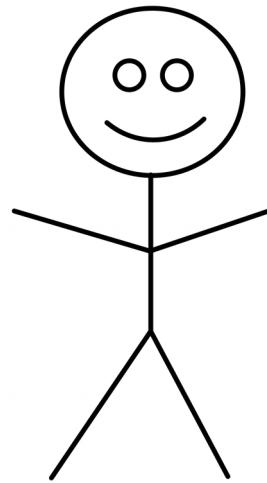
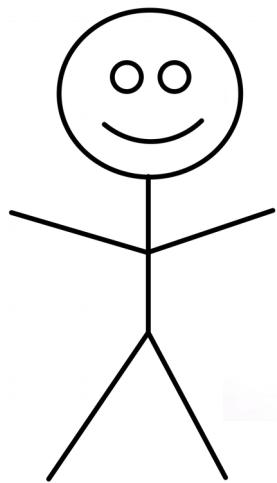
# Shared Borrows



# Shared Borrows



# Shared Borrows



# Shared Borrows

# Shared Borrows

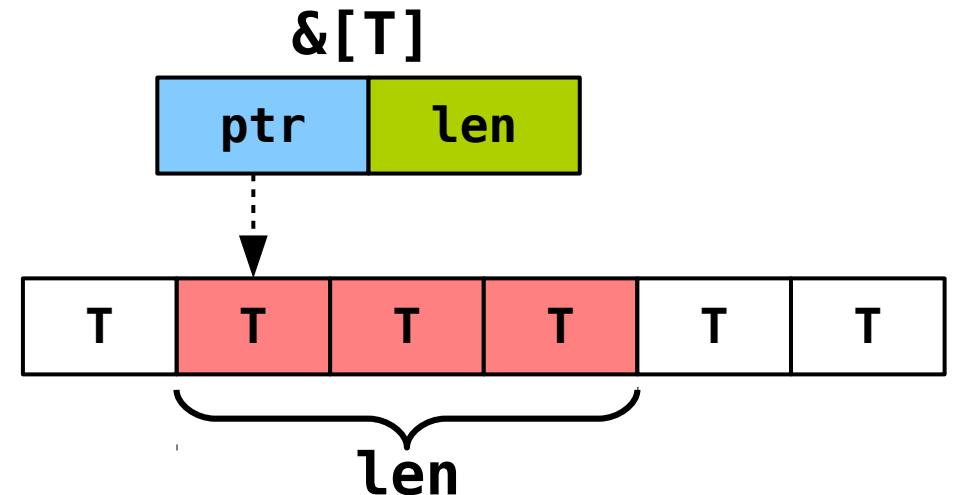
```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
```

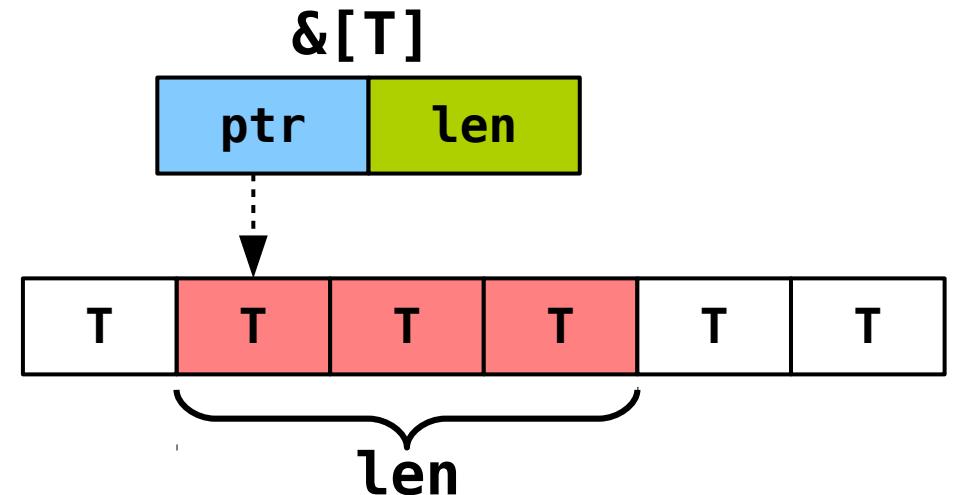
# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
```



# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
```



# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {  
    let mut r = Vec::new();
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
}
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
    let z1 = add_slices(&x, &y);
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
    let z1 = add_slices(&x, &y);
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
    let z1 = add_slices(&x, &y);
    let z2 = add_slices(&x, &x);
}
```

# Shared Borrows

```
fn add_slices(a: &[u32], b: &[u32]) -> Vec<u32> {
    let mut r = Vec::new();
    for (ai, bi) in a.iter().zip(b.iter()) {
        r.push(ai + bi)
    }
    return r;
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
    let z1 = add_slices(&x, &y);
    let z2 = add_slices(&x, &x); ← Aliasing
}
```

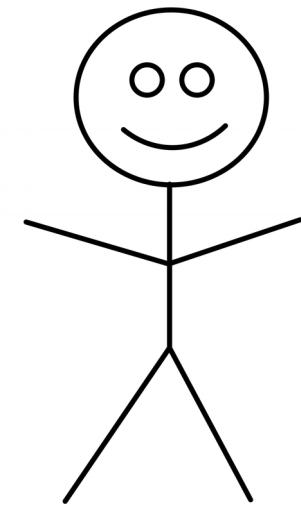
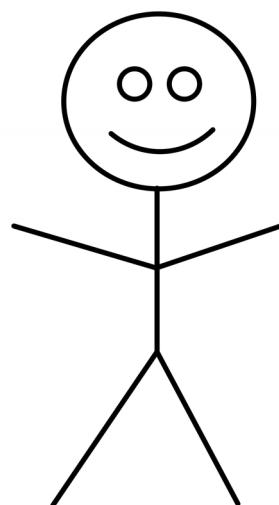
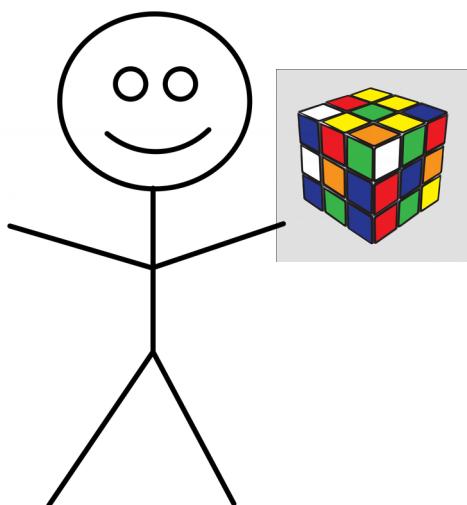
# Mutable Borrows

| Type | Owns | May alias | Can mutate |
|------|------|-----------|------------|
| T    | ✓    |           | ✓          |
| &T   |      | ✓         |            |

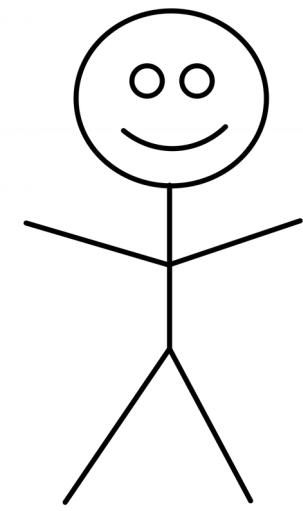
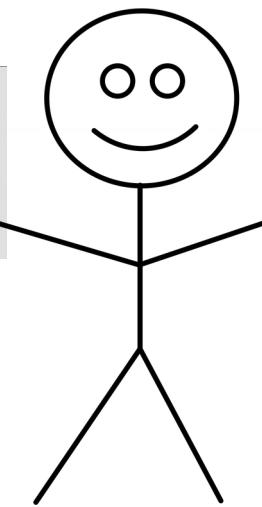
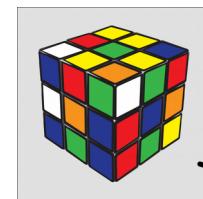
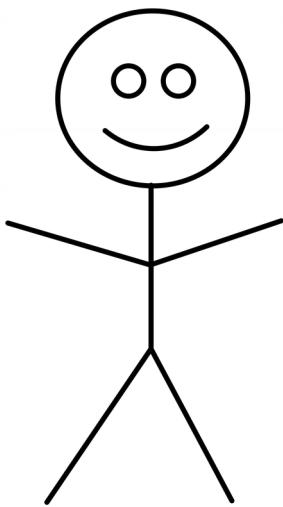
# Mutable Borrows

| Type   | Owns | May alias | Can mutate |
|--------|------|-----------|------------|
| T      | ✓    |           | ✓          |
| &T     |      | ✓         |            |
| &mut T |      |           | ✓          |

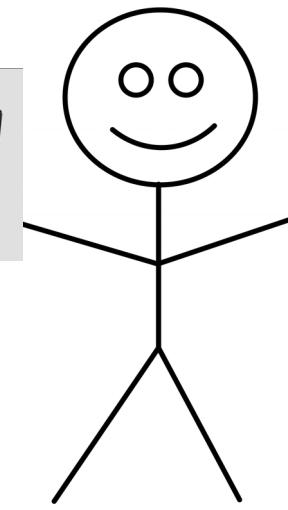
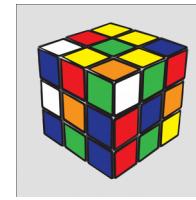
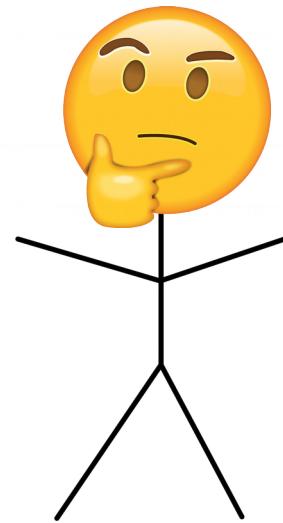
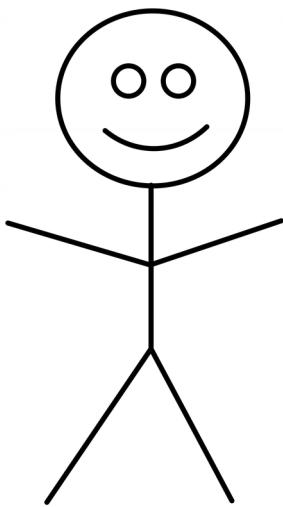
# Mutable Borrow



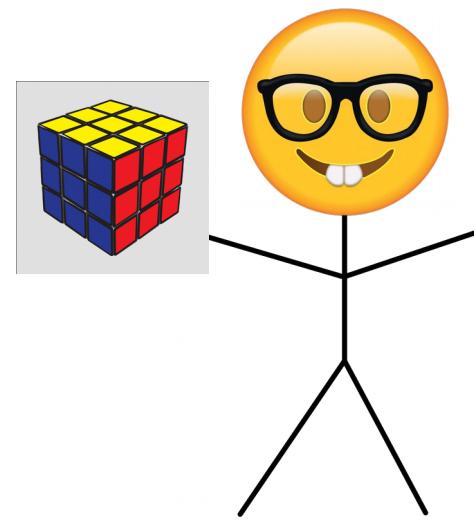
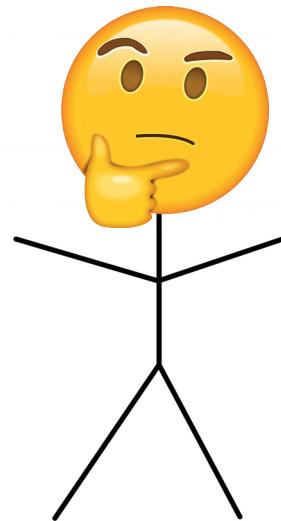
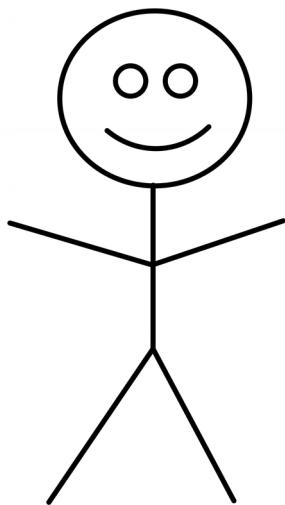
# Mutable Borrow



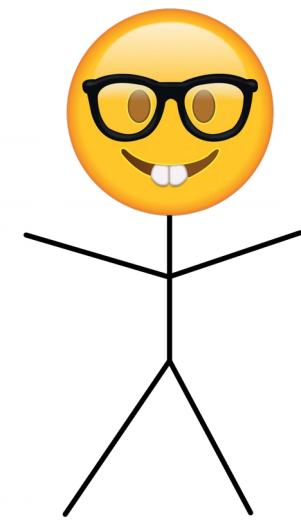
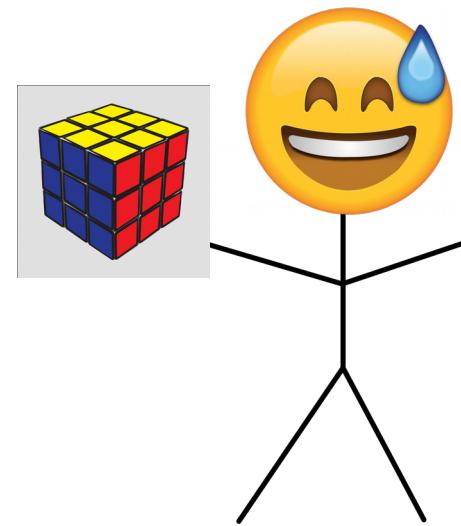
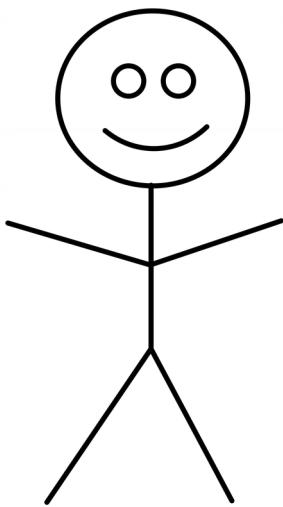
# Mutable Borrow



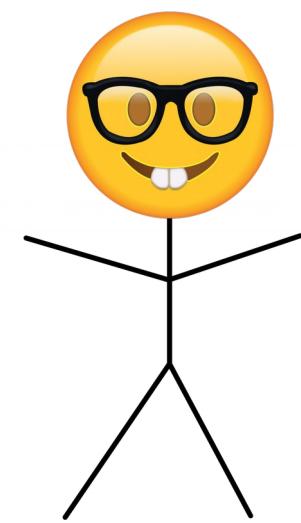
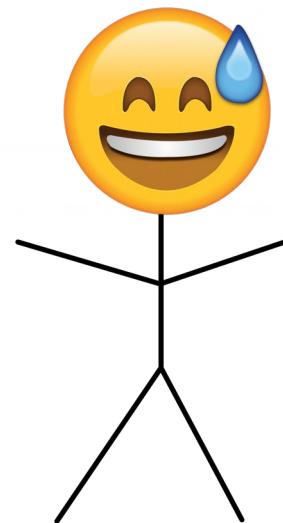
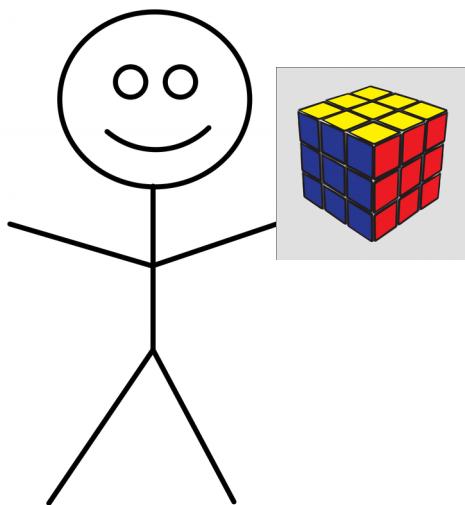
# Mutable Borrow



# Mutable Borrow



# Mutable Borrow



# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}  
  
fn main() {  
    let x = vec![10; 3];  
    let y = vec![42, 5, 1337];
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}  
  
fn main() {  
    let x = vec![10; 3];  
    let y = vec![42, 5, 1337];  
    add_slices_inplace(&mut x, &y);  
}
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}  
  
fn main() {  
    let x = vec![10; 3];  
    let y = vec![42, 5, 1337];  
    add_slices_inplace(&mut x, &y);  
}
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {
    for (ai, bi) in a.iter_mut().zip(b.iter()) {
        *ai += bi
    }
}

fn main() {
    let x = vec![10; 3];
    let y = vec![42, 5, 1337];
    add_slices_inplace(&mut x, &y);
    add_slices_inplace(&mut x, &x);
}
```

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}  
  
fn main() {  
    let x = vec![10; 3];  
    let y = vec![42, 5, 1337];  
    add_slices_inplace(&mut x, &y);  
    add_slices_inplace(&mut x, &x);  
}
```

↑  
Aliasing

# Mutable Borrows

```
fn add_slices_inplace(a: &mut [u32], b: &[u32]) {  
    for (ai, bi) in a.iter_mut().zip(b.iter()) {  
        *ai += bi  
    }  
}  
  
fn main() {  
    let x = vec![10; 3];  
    let y = vec![42, 5, 1337];  
    add_slices_inplace(&mut x, &y);  
    add_slices_inplace(&mut x, &x);  
}
```

error[E0502]: cannot borrow `x` as immutable  
because it is also borrowed as mutable

Aliasing



# References

```
fn main() {  
    let mut a = 42;
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;  
    *x = 3;
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;  
    *x = 3;  
    assert_eq!(x.count_ones(), 2);
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;  
    *x = 3;  
    assert_eq!(x.count_ones(), 2);
```

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;  
    *x = 3;  
    assert_eq!(x.count_ones(), 2);  
    a = 7;
```

# References

```
fn main() {
    let mut a = 42;
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
    a = 7;
```

# References

```
fn main() {
    let mut a = 42;
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
    a = 7;
```

error: cannot assign to `a` because it is borrowed

# References

```
fn main() {
    let mut a = 42;
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
    a = 7;
    assert_eq!(a.count_ones(), 3);
}
```

**error: cannot assign to `a` because it is borrowed**

# References

```
fn main() {
    let mut a = 42;
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
    a = 7;
    assert_eq!(a.count_ones(), 3);
}
```

error: cannot assign to `a` because it is borrowed

# References

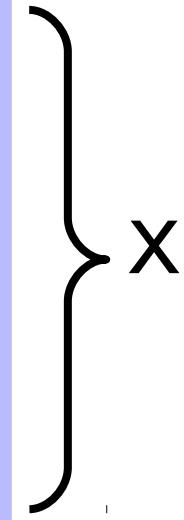
```
fn main() {
    let mut a = 42;
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
    a = 7;
    assert_eq!(a.count_ones(), 3);
}
```

error: cannot assign to `a` because it is borrowed

error: cannot use `a` because it was mutably borrowed

# References

```
fn main() {  
    let mut a = 42;  
    let x = &mut a;  
    *x = 3;  
    assert_eq!(x.count_ones(), 2);  
    a = 7;  
    assert_eq!(a.count_ones(), 3);  
}
```



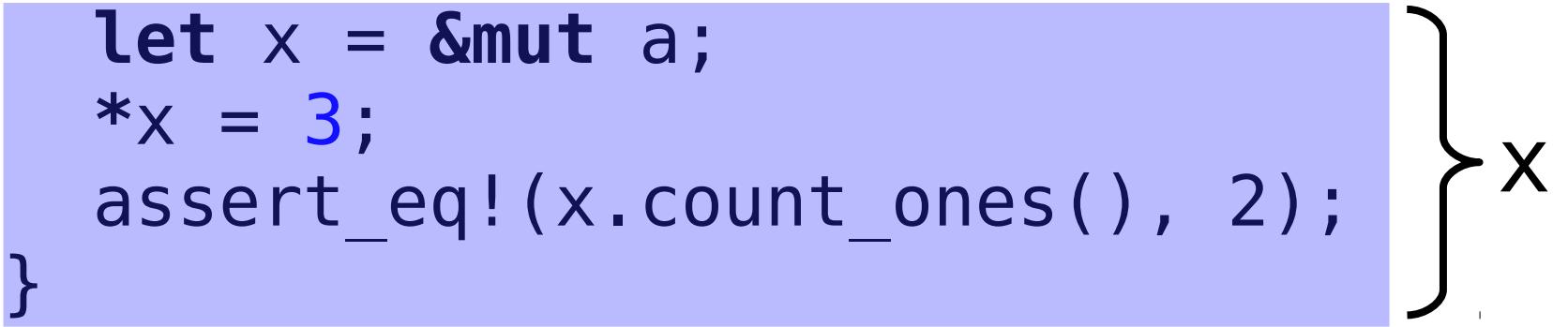
error: cannot assign to `a` because it is borrowed  
error: cannot use `a` because it was mutably borrowed

# References

```
fn main() {
    let mut a = 42;
{
    let x = &mut a;
    *x = 3;
    assert_eq!(x.count_ones(), 2);
}
a = 7;
assert_eq!(a.count_ones(), 3);
}
```

# References

```
fn main() {
    let mut a = 42;
    {
        let x = &mut a;
        *x = 3;
        assert_eq!(x.count_ones(), 2);
    }
    a = 7;
    assert_eq!(a.count_ones(), 3);
}
```



The code demonstrates mutable references. It declares a variable `a` with a value of 42. Inside a block, it creates a mutable reference `x` to `a`, changes its value to 3, and asserts that the count of ones in `x` is 2. After exiting the block, it changes the value of `a` to 7 and asserts that the count of ones in `a` is 3. The brace indicates that the entire block body is associated with the reference `x`.

# References

```
fn main() {
    let mut a = 42;
    {
        let x = &mut a;
        *x = 3;
        assert_eq!(x.count_ones(), 2);
    }
    a = 7;
    assert_eq!(a.count_ones(), 3);
}
```



# Lifetime Tracking

```
fn first<'a>(v: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &v[0];  
}
```

# Lifetime Tracking

```
fn first<'a>(v: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &v[0];  
}
```

Returns a reference with the same lifetime of v.

# Lifetime Tracking

```
fn first<'a>(v: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &v[0];  
}
```

Returns a reference with the same lifetime of v.

```
fn first(v: &Vec<Vec<i32>>) -> &Vec<i32> {  
    return &v[0];  
}
```

Shorthand

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &m[0];  
}
```

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {
    return &m[0];
}

fn main() {
    let mut matrix: Vec::new();
    {
        let first_row = first(&matrix);
        matrix.push(...);
    }
}
```

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {
    return &m[0];
}

fn main() {
    let mut matrix: Vec::new();
    {
        let first_row = first(&matrix);
        matrix.push(...);
    }
}
```

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &m[0];  
}
```

```
fn main() {  
    let mut matrix: Vec::new();  
    {  
        let first_row = first(&matrix);  
        matrix.push(...);  
    }  
}
```



# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {  
    return &m[0];  
}
```

```
fn main() {  
    let mut matrix: Vec::new();  
    {  
        let first_row = first(&matrix);  
        matrix.push(...);  
    }  
}
```

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {
    return &m[0];
}

fn main() {
    let mut matrix: Vec::new();
    {
        let first_row = first(&matrix);
        matrix.push(...);
    }
}
```

The code illustrates lifetime tracking in Rust. The `first` function takes a reference to a vector of vectors and returns a reference to the first row. The `main` function creates a mutable vector `matrix`, initializes it with `Vec::new()`, and then pushes a new row onto it. Inside the block where `matrix` is mutable, a local variable `first\_row` is created by calling `first(&matrix)`. This call is highlighted with a blue box and a brace on the right labeled "'a", indicating that the returned reference has the same lifetime as the input reference `&matrix`.

# Lifetime Tracking

```
fn first<'a>(m: &'a Vec<Vec<i32>>) -> &'a Vec<i32> {
    return &m[0];
}

fn main() {
    let mut matrix: Vec::new();
    {
        let first_row = first(&matrix);
        matrix.push(...); } } 'a
}
```

# Traits

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
    fn clone(&self) -> Self {
        let mut v = Vec::new();
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
    fn clone(&self) -> Self {
        let mut v = Vec::new();
        for elem in self {
            v.push(elem.clone())
        }
    }
}
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
    fn clone(&self) -> Self {
        let mut v = Vec::new();
        for elem in self {
            v.push(elem.clone())
        }
    }
}
```

# Traits

```
let v = vec![1, 42, 7];
let y = v.clone();
```

```
trait Clone {
    fn clone(&self) -> Self;
}
```

```
impl<T: Clone> Clone for Vec<T> {
    fn clone(&self) -> Self {
        let mut v = Vec::new();
        for elem in self {
            v.push(elem.clone())
        }
        return v;
    }
}
```

# Static Polymorphism

```
trait Evaluable {  
    fn eval(&self) -> i32;  
}
```

# Static Polymorphism

```
trait Evaluable {  
    fn eval(&self) -> i32;  
}
```

# Static Polymorphism

```
trait Evaluable {  
    fn eval(&self) -> i32;  
}
```

```
struct Add<L, R> {  
    lhs: L,  
    rhs: R  
}
```

# Static Polymorphism

```
trait Evaluable {           impl<L, R> Evaluable for Add<L, R>
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}
```

# Static Polymorphism

```
trait Evaluable {           impl<L, R> Evaluable for Add<L, R>
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}
```

# Static Polymorphism

```
trait Evaluable {  
    fn eval(&self) -> i32;  
}  
  
struct Add<L, R> {  
    lhs: L,  
    rhs: R  
}
```

impl<L, R> Evaluable for Add<L, R>



# Static Polymorphism

```
trait Evaluable {           impl<L, R> Evaluable for Add<L, R>
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}
```

# Static Polymorphism

```
trait Evaluable {          impl<L, R> Evaluable for Add<L, R>
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}

impl Evaluable for i32 {
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}

impl Evaluable for i32 {
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}

impl Evaluable for i32 {
    fn eval(&self) -> i32 { *self }
}
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}

impl Evaluable for i32 {
    fn eval(&self) -> i32 { *self }
}
```

---

```
fn main() {
    let add = Add{ lhs: 5, rhs: 42 };
```

# Static Polymorphism

```
trait Evaluable {
    fn eval(&self) -> i32;
}

struct Add<L, R> {
    lhs: L,
    rhs: R
}

impl<L, R> Evaluable for Add<L, R>
    where L: Evaluable, R: Evaluable
{
    fn eval(&self) -> i32 {
        self.lhs.eval() + self.rhs.eval()
    }
}

impl Evaluable for i32 {
    fn eval(&self) -> i32 { *self }
}
```

---

```
fn main() {
    let add = Add{ lhs: 5, rhs: 42 };
    assert!(add.eval() == 47);
}
```

# Traits



Overloading

Templates

Concepts

Virtual functions

Virtual concepts



Traits

Traits

Traits

Trait objects

Trait objects

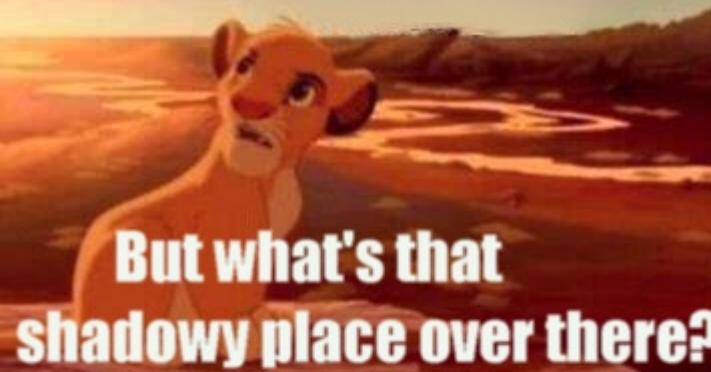
# Unsafe Rust

# Unsafe Rust

# Unsafe Rust

# Unsafe Rust

**THESE ARE THE RUST LANDS**



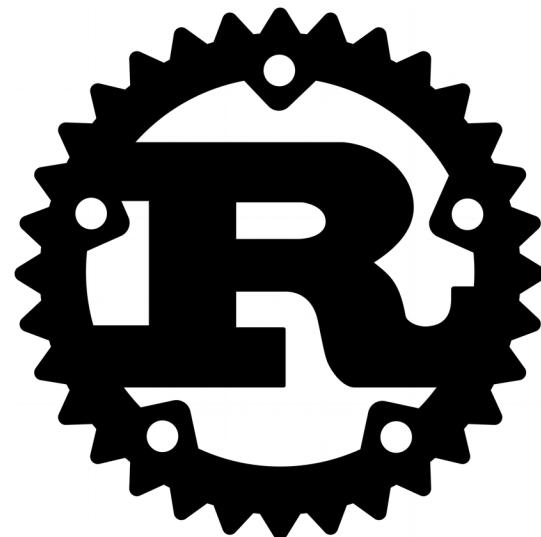
**But what's that  
shadowy place over there?**

A scene from The Lion King showing Simba looking towards a dark, shadowed area in the distance across a vast savanna.

**UNSAFE RUST**

Mufasa and Simba looking towards the same dark, shadowed area in the distance.

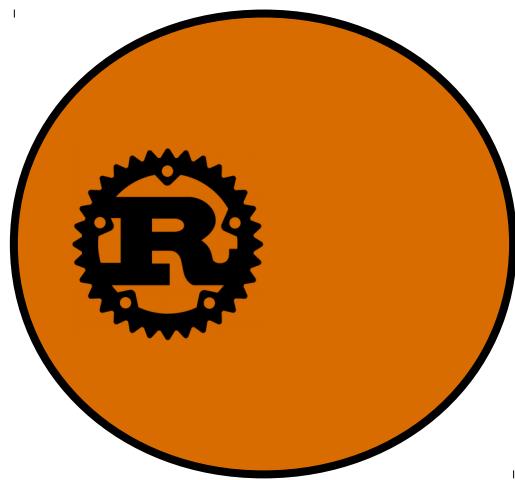
# Unsafe Rust



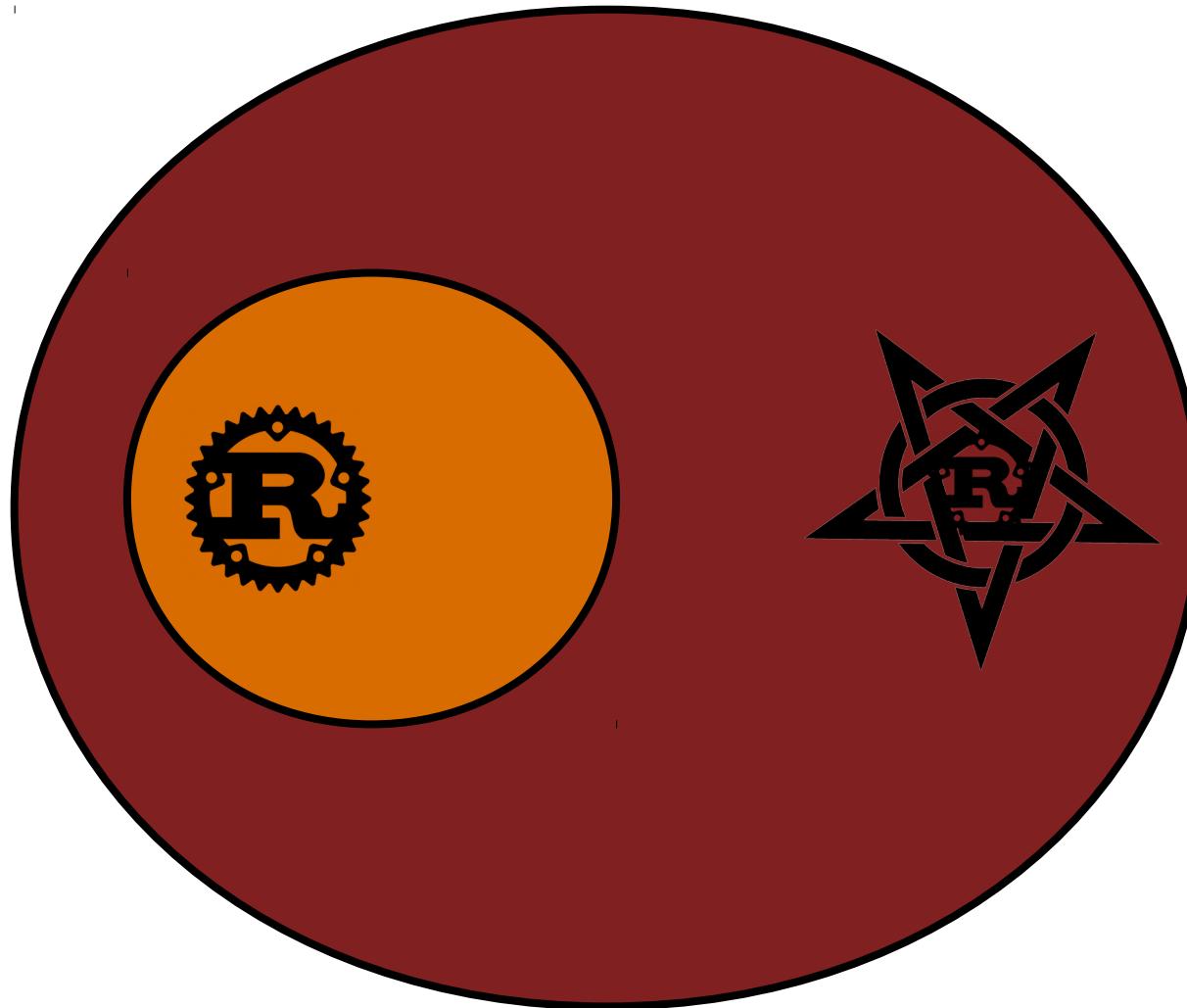
$\neq$



# Unsafe Rust



# Unsafe Rust



```
fn foo( ) {
```

**fn** foo( ) {

**unsafe** {

}

}

```
fn foo() {  
    unsafe {  
        }  
    }  
}
```



# Guarantees

```
impl<T> Vec<T> {
```

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }  
  
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }  
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }  
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }  
  
    ➔ n ∈ 0..N => Some  
    else           => None  
  
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

→ n ∈ 0..N => Some  
else => None

```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

→ n ∈ 0..N => Some  
else => None

```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

→ n ∈ 0..N => Some  
else => None

```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```

→ n ∈ 0..N => &T  
else => undefined

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

→  $n \in 0..N \Rightarrow \text{Some}$   
 $\text{else} \Rightarrow \text{None}$

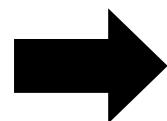
```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```

→  $n \in 0..N \Rightarrow \&T$   
 $\text{else} \Rightarrow \text{undefined}$

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

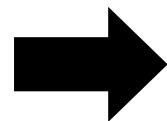


$n \in 0..N \Rightarrow$  Some  
**else**  $\Rightarrow$  None



**Callee**

```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```

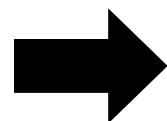


$n \in 0..N \Rightarrow$  &T  
**else**  $\Rightarrow$  undefined

```
}
```

# Guarantees

```
impl<T> Vec<T> {  
    fn get(&self, n: usize) -> Option<&T> { ... }
```

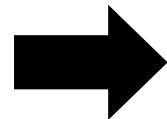


$n \in 0..N \Rightarrow$  Some  
 $\text{else} \Rightarrow$  None



Callee

```
unsafe fn get_unchecked(&self, n: usize) -> &T { ... }
```



$n \in 0..N \Rightarrow$  &T  
 $\text{else} \Rightarrow$  undefined



Caller

```
}
```

# Borrow Limitations

2

3

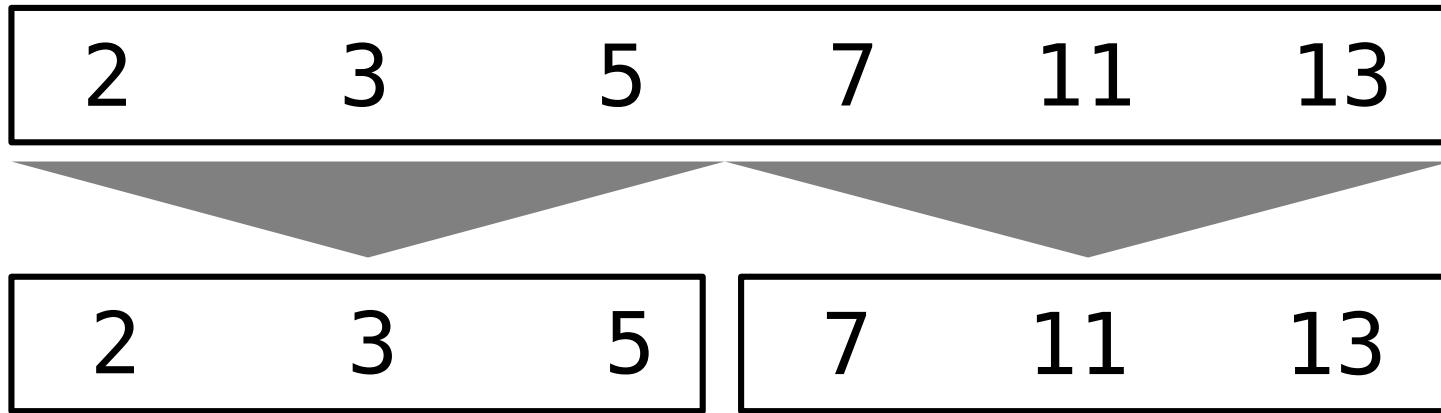
5

7

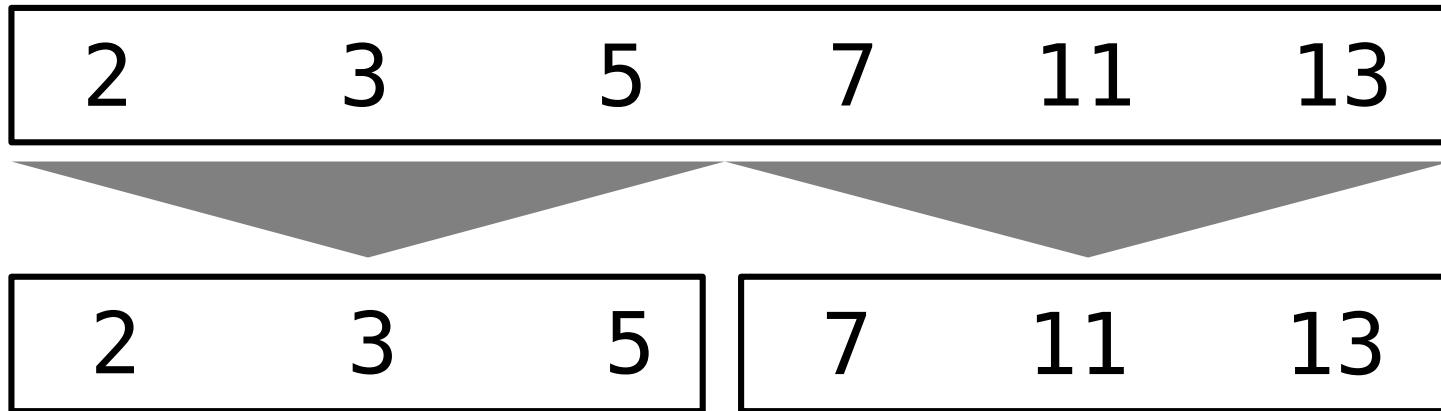
11

13

# Borrow Limitations

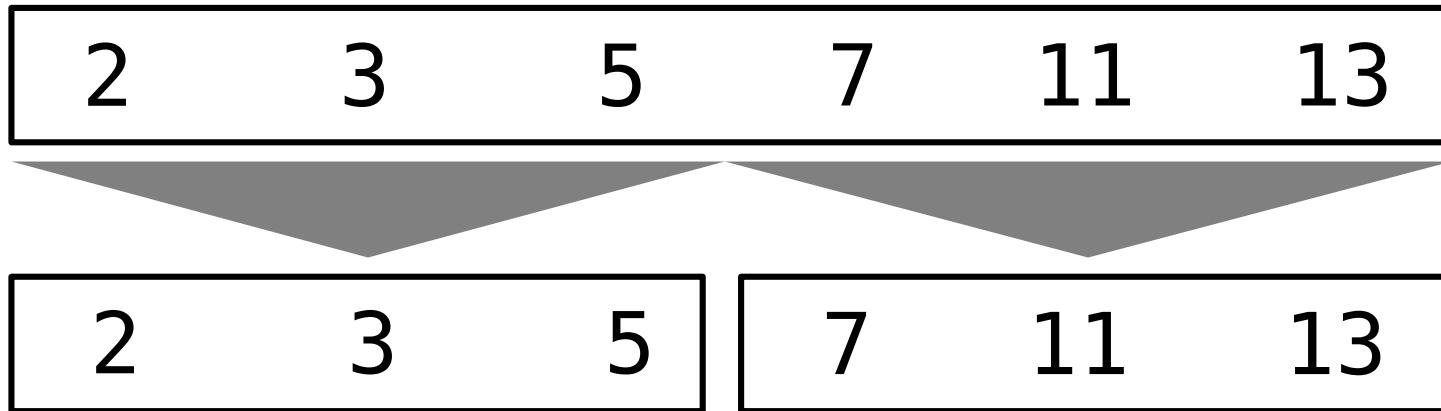


# Borrow Limitations



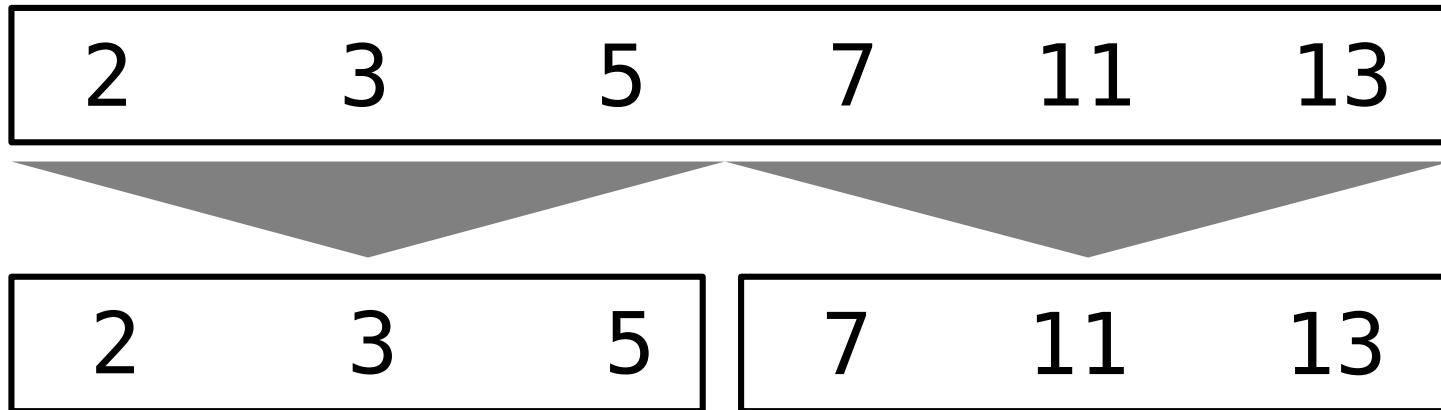
```
let mut v = vec![2, 3, 5, 7, 11, 13];
```

# Borrow Limitations



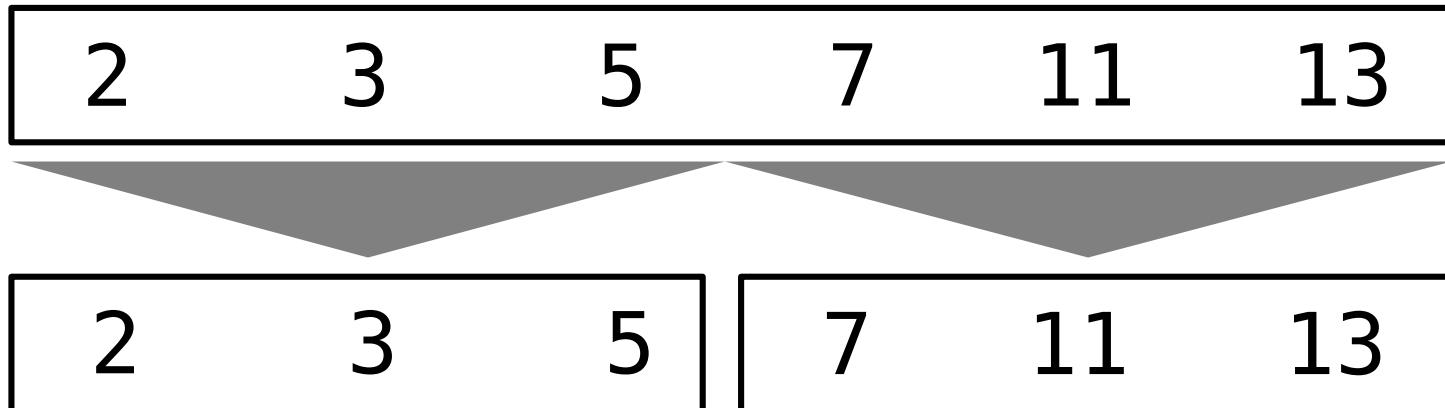
```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
```

# Borrow Limitations



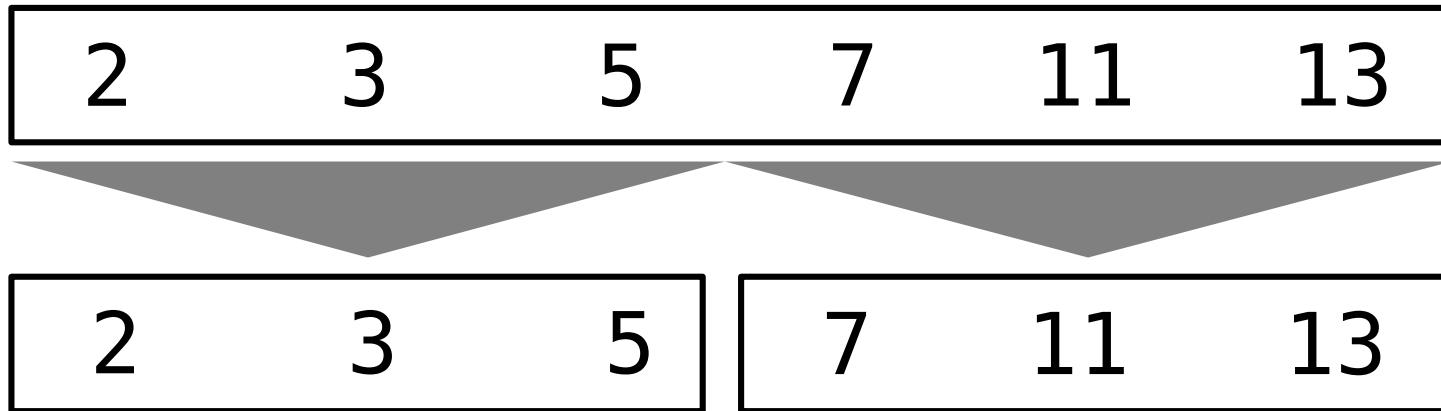
```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
let h1 = &mut v[ ..m];
```

# Borrow Limitations



```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
let h1 = &mut v[ ..m];
let h2 = &mut v[m..];
```

# Borrow Limitations



```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
let h1 = &mut v[..m];
let h2 = &mut v[m..];
```



error[E0499]: cannot borrow `v` as mutable  
more than once at a time

# Safe Abstractions

```
fn split_at_mut(&mut self, mid: usize) -> (&mut [T], &mut [T])
```

Divides one `&mut` into two at an index.

The first will contain all indices from `[0, mid)` (excluding the index `mid` itself) and the second will contain all indices from `[mid, len)` (excluding the index `len` itself).

## Panics

---

Panics if `mid > len`.

# Safe Abstractions

```
fn split_at_mut(&mut self, mid: usize) -> (&mut [T], &mut [T])
```

Divides one `&mut` into two at an index.

The first will contain all indices from `[0, mid)` (excluding the index `mid` itself) and the second will contain all indices from `[mid, len)` (excluding the index `len` itself).

## Panics

---

Panics if `mid > len`.

```
fn split_at_mut<'a>(&'a mut self, ...) -> (&'a mut [T], &'a mut [T])
```

# Safe Abstractions

```
fn split_at_mut(&mut self, mid: usize) -> (&mut [T], &mut [T])
```

Divides one `&mut` into two at an index.

The first will contain all indices from `[0, mid)` (excluding the index `mid` itself) and the second will contain all indices from `[mid, len)` (excluding the index `len` itself).

## Panics

---

Panics if `mid > len`.

```
fn split_at_mut<'a>(&'a mut self, ...) -> (&'a mut [T], &'a mut [T])
```

```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
```

# Safe Abstractions

```
fn split_at_mut(&mut self, mid: usize) -> (&mut [T], &mut [T])
```

Divides one `&mut` into two at an index.

The first will contain all indices from `[0, mid)` (excluding the index `mid` itself) and the second will contain all indices from `[mid, len)` (excluding the index `len` itself).

## Panics

---

Panics if `mid > len`.

```
fn split_at_mut<'a>(&'a mut self, ...) -> (&'a mut [T], &'a mut [T])
```

```
let mut v = vec![2, 3, 5, 7, 11, 13];
let      m = v.len() / 2;
let (h1, h2) = v.split_at_mut(m);
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {  
    let len = v.len();  
    if len <= 1 { return }  
}
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {  
    let len = v.len();  
    if len <= 1 { return }  
    let mid = len / 2;
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {  
    let len = v.len();  
    if len <= 1 { return }  
    let mid = len / 2;
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {  
    let len = v.len();  
    if len <= 1 { return }  
    let mid = len / 2;
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {  
    let len = v.len();  
    if len <= 1 { return }  
    let mid = len / 2;
```

# Mergesort

```
fn mergesort(v: &mut [i32]) {
    let len = v.len();
    if len <= 1 { return }
    let mid = len / 2;
    {
        let (hi, lo) = v.split_at_mut(mid);
        mergesort(hi);
        mergesort(lo);
    }
    merge(v, mid)
}
```

# Parallel Mergesort

```
fn mergesort(v: &mut [i32]) {
    let len = v.len();
    if len <= 1 { return }
    let mid = len / 2;
    {
        let (hi, lo) = v.split_at_mut(mid);
        rayon::join(|| mergesort(hi),
                    || mergesort(lo));
    }
    merge(v, mid)
}
```

# Parallel Mergesort

```
fn mergesort(v: &mut [i32]) {
    let len = v.len();
    if len <= 1 { return }
    let mid = len / 2;
    {
        let (hi, lo) = v.split_at_mut(mid);
        rayon::join(|| mergesort(hi),
                    || mergesort(lo));
    }
    merge(v, mid)
}
```

**&mut** can safely go  
between threads  
because it is unaliased

# Parallel Mergesort

```
fn mergesort(v: &mut [i32]) {
    let len = v.len();
    if len <= 1 { return }
    let mid = len / 2;
    {
        let (hi, lo) = v.split_at_mut(mid);
        rayon::join(|| mergesort(hi));
        rayon::join(|| mergesort(lo));
    }
}
```



**crates.io**

Rust Package Registry



rayon

Browse All Crates

Docs ▾

Log in with GitHub



**rayon** 0.9.0

[Documentation](#) [Repository](#) [Dependent crates](#)

Cargo.toml

```
rayon = "0.9.0"
```



Last Updated

3 days ago

crates.io v0.9.0

## About This Package

Simple work-stealing parallelism for Rust

## Authors

# Communities

# C++ Community



# C++ Community



# C++ Community



# C++ Community



**SUTTER'S MILL**  
Herb Sutter on software development



# C++ Community



**SUTTER'S MILL**  
Herb Sutter on software development



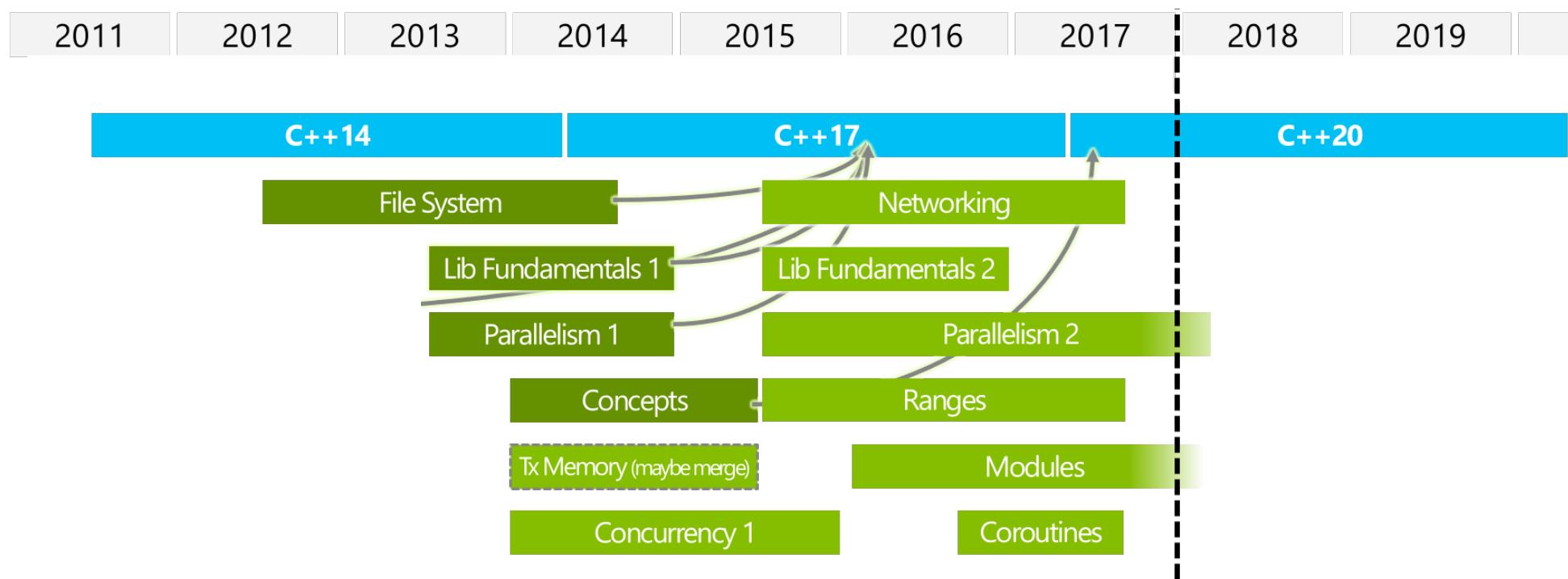
# C++ Community



# C++ Community

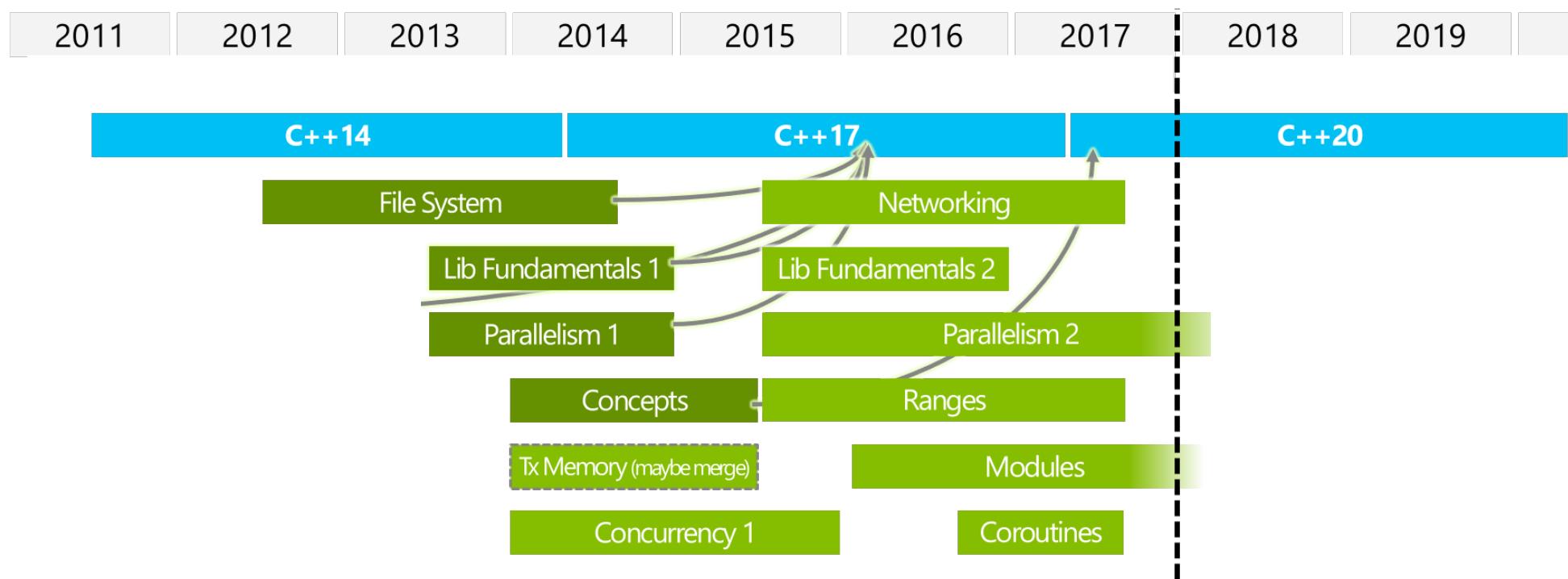


# C++ Pipeline



# C++ Pipeline

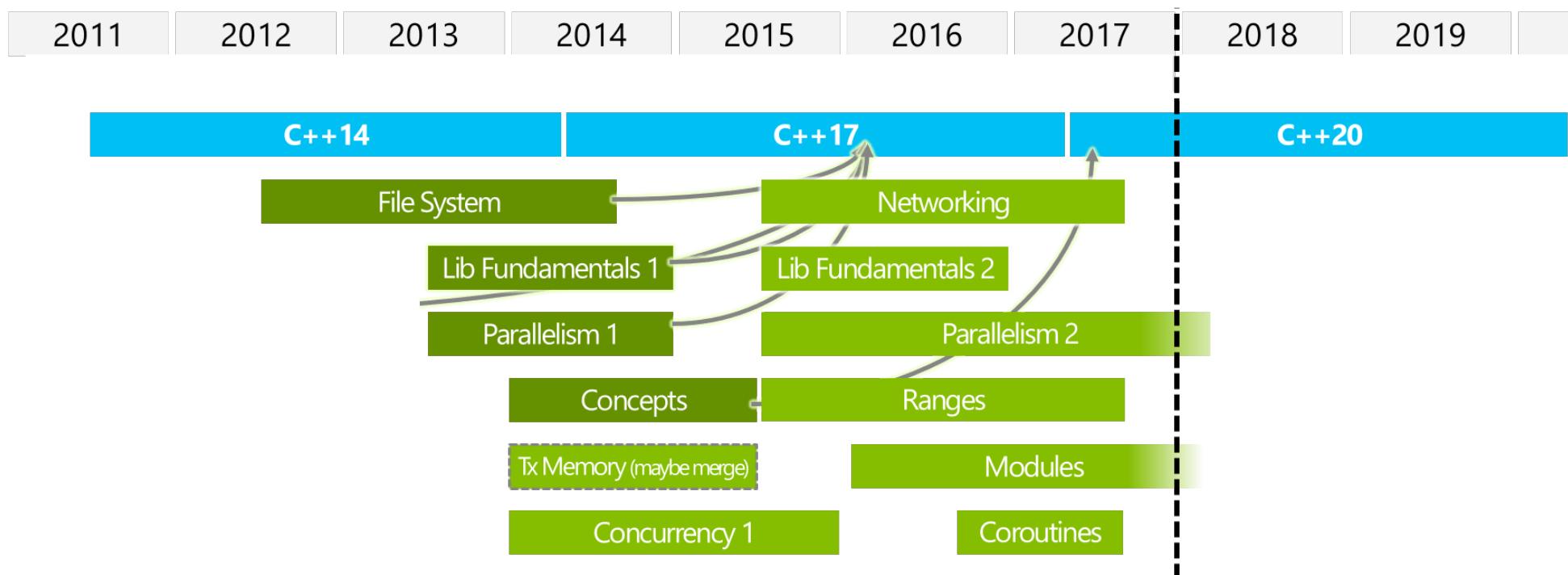
3 years cycle



# C++ Pipeline

3 years cycle

## Working Groups

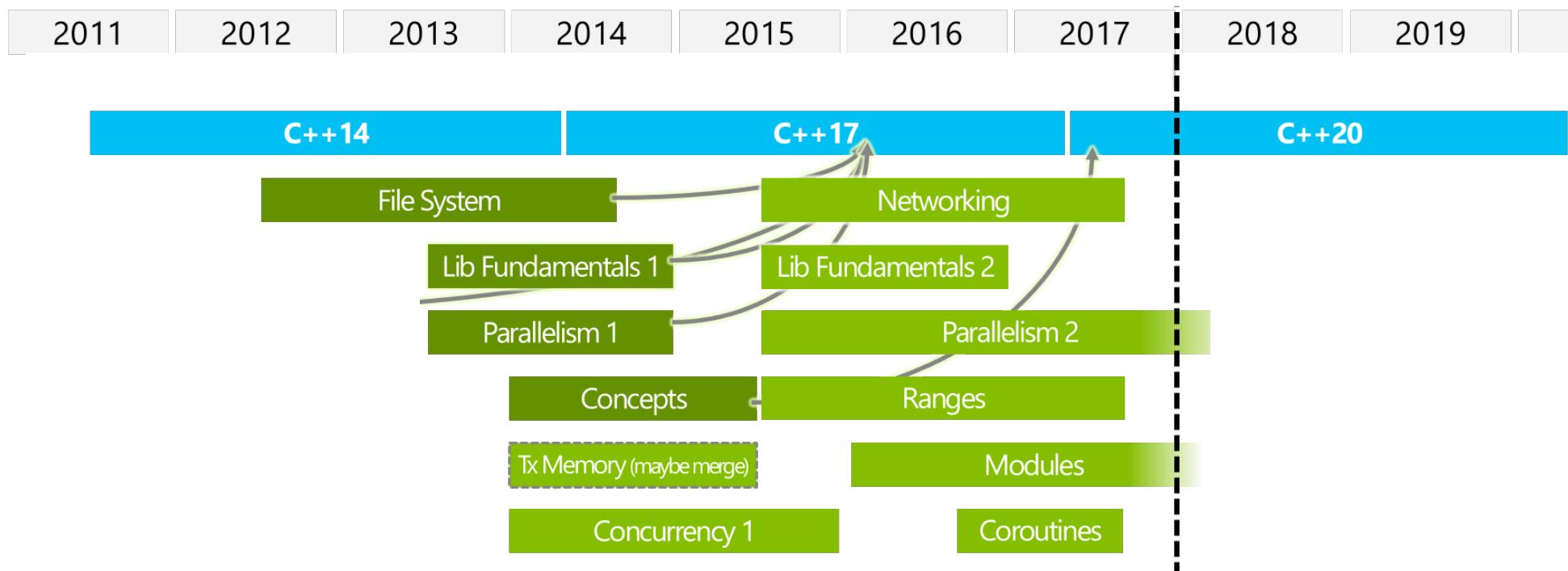


# C++ Pipeline

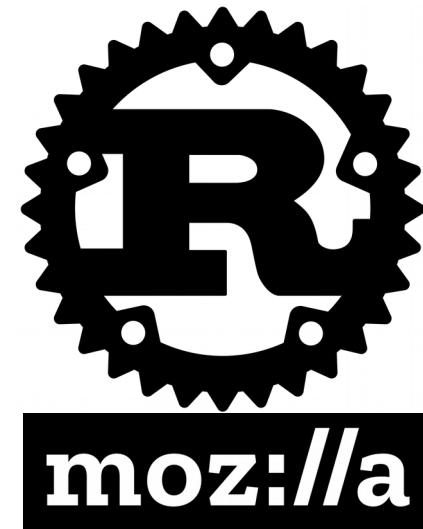
3 years cycle

Working Groups

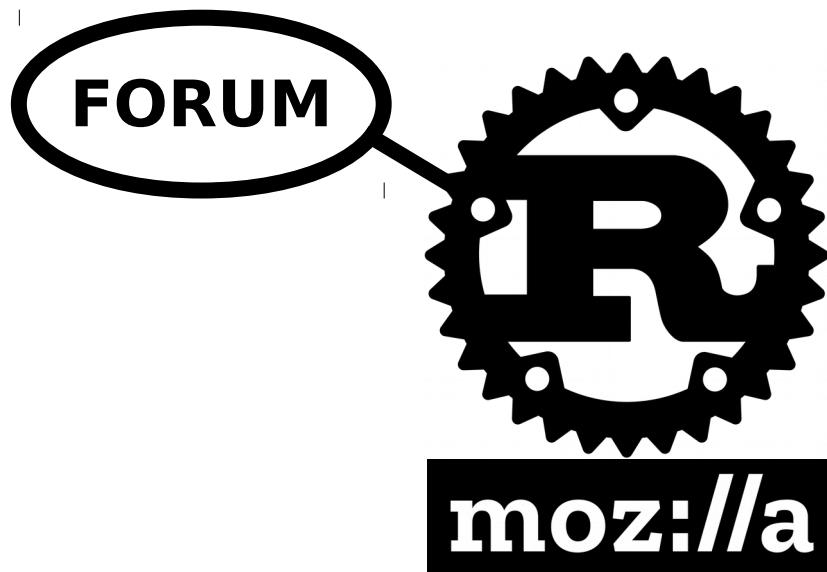
Technical Specifications (**TS**)



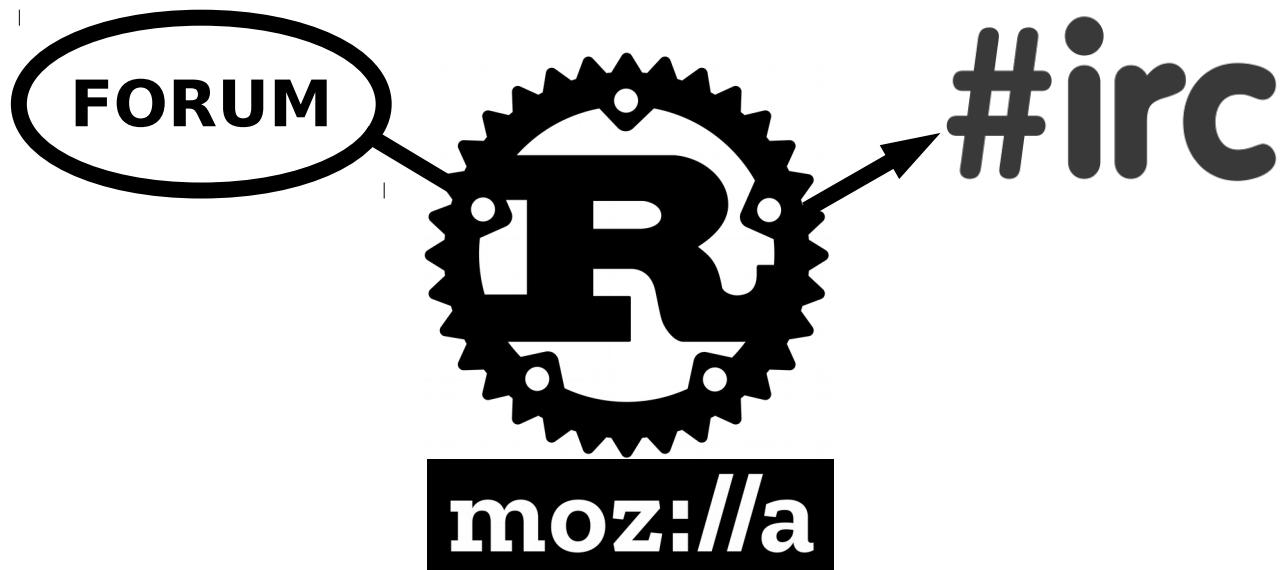
# Rust Community



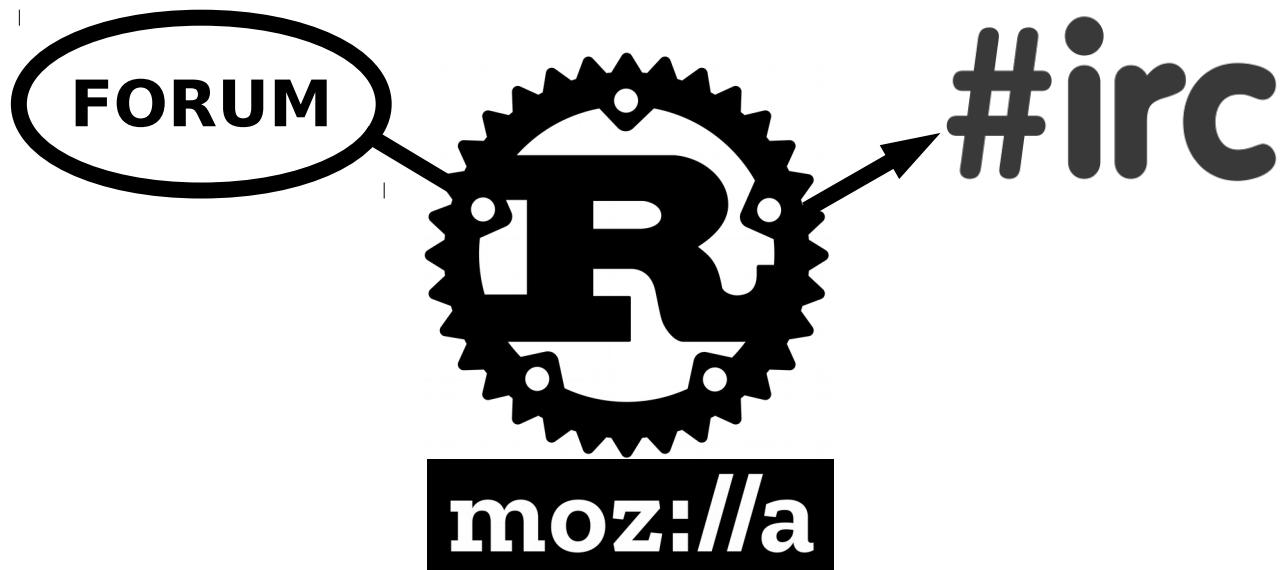
# Rust Community



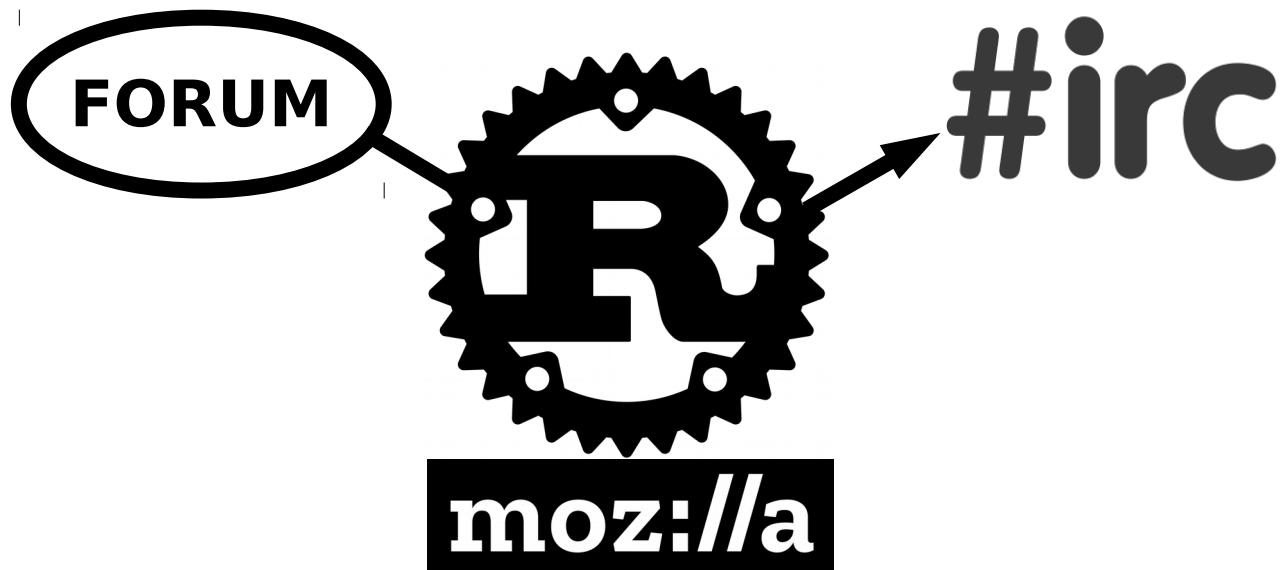
# Rust Community



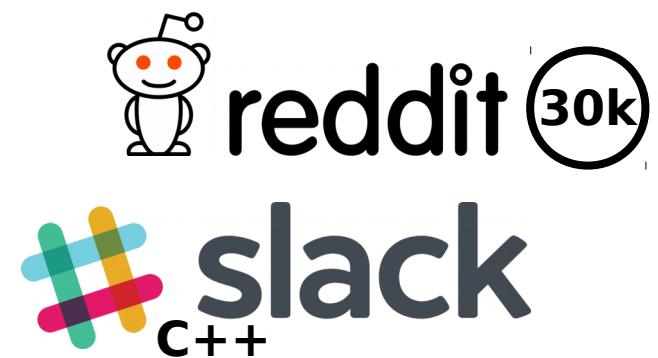
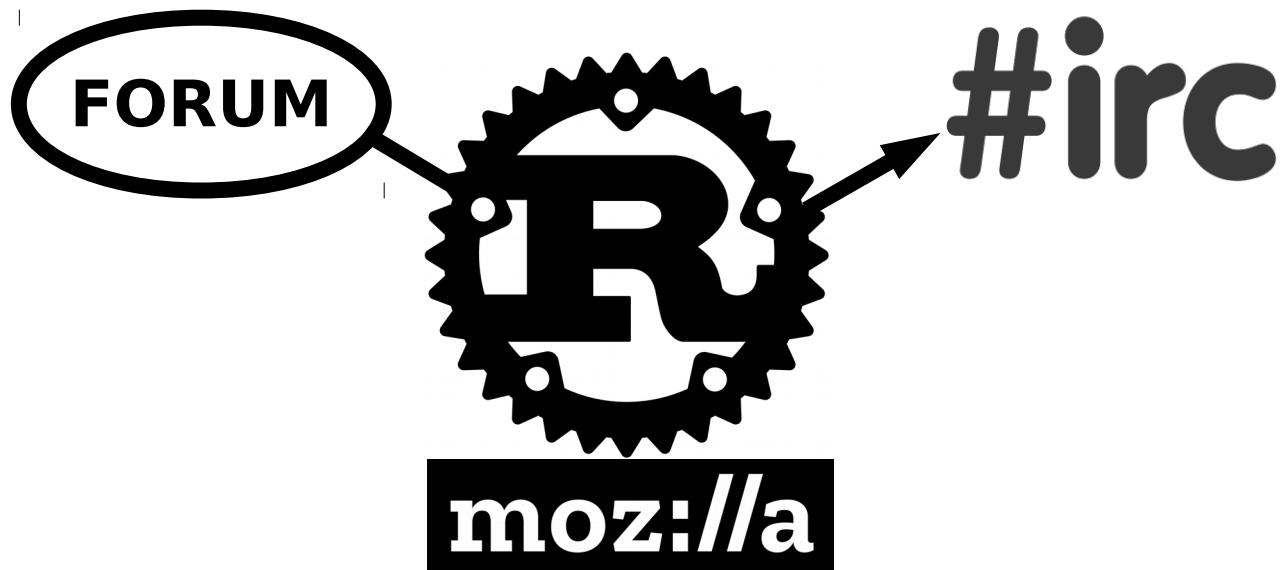
# Rust Community



# Rust Community



# Rust Community



# Rust Community



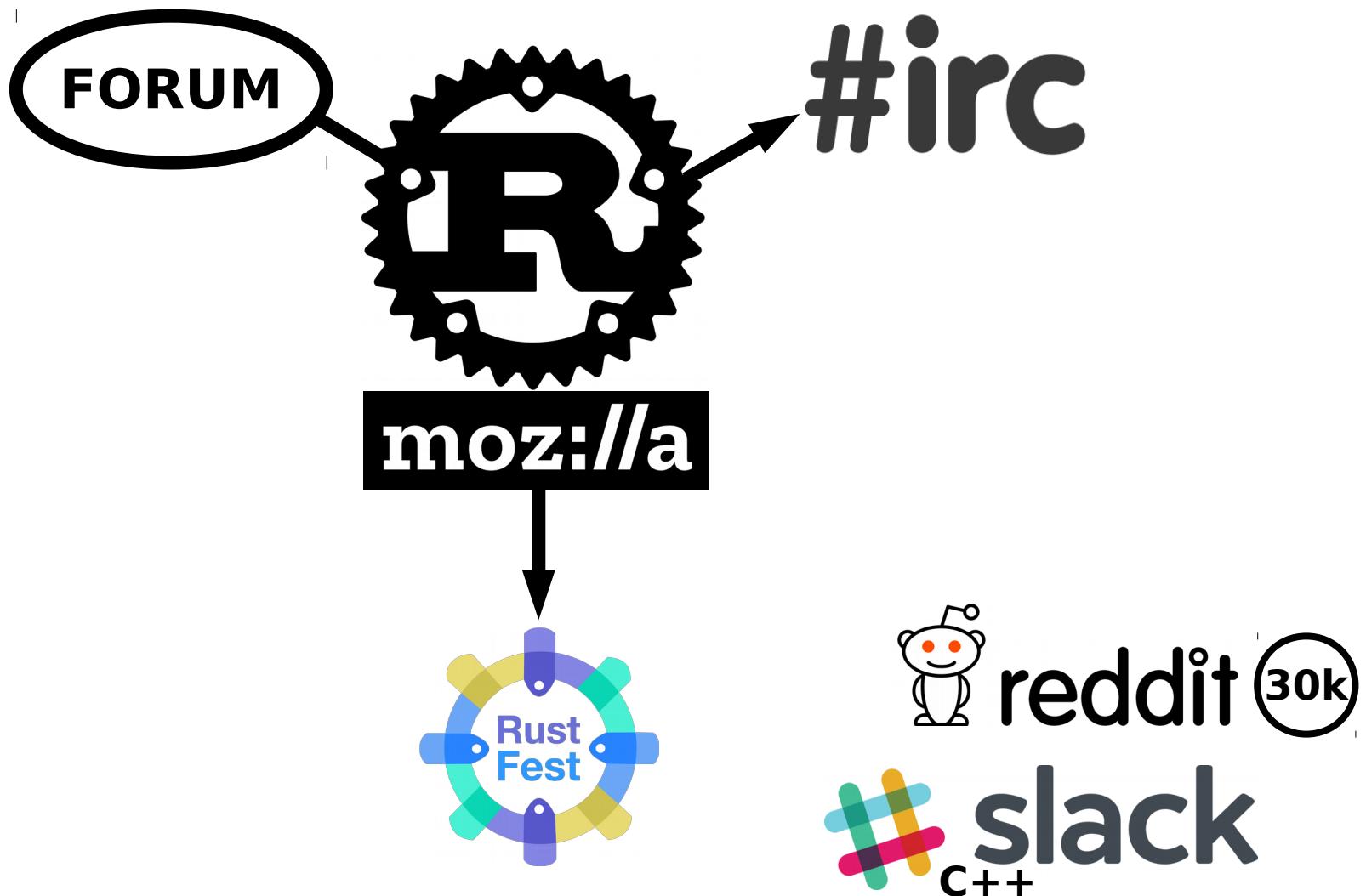
Welcome to the C++ Slack.  
[...]

Discuss anything somewhat related to C++  
**but beware of the Rust ~~trolls~~ evangelists ...**

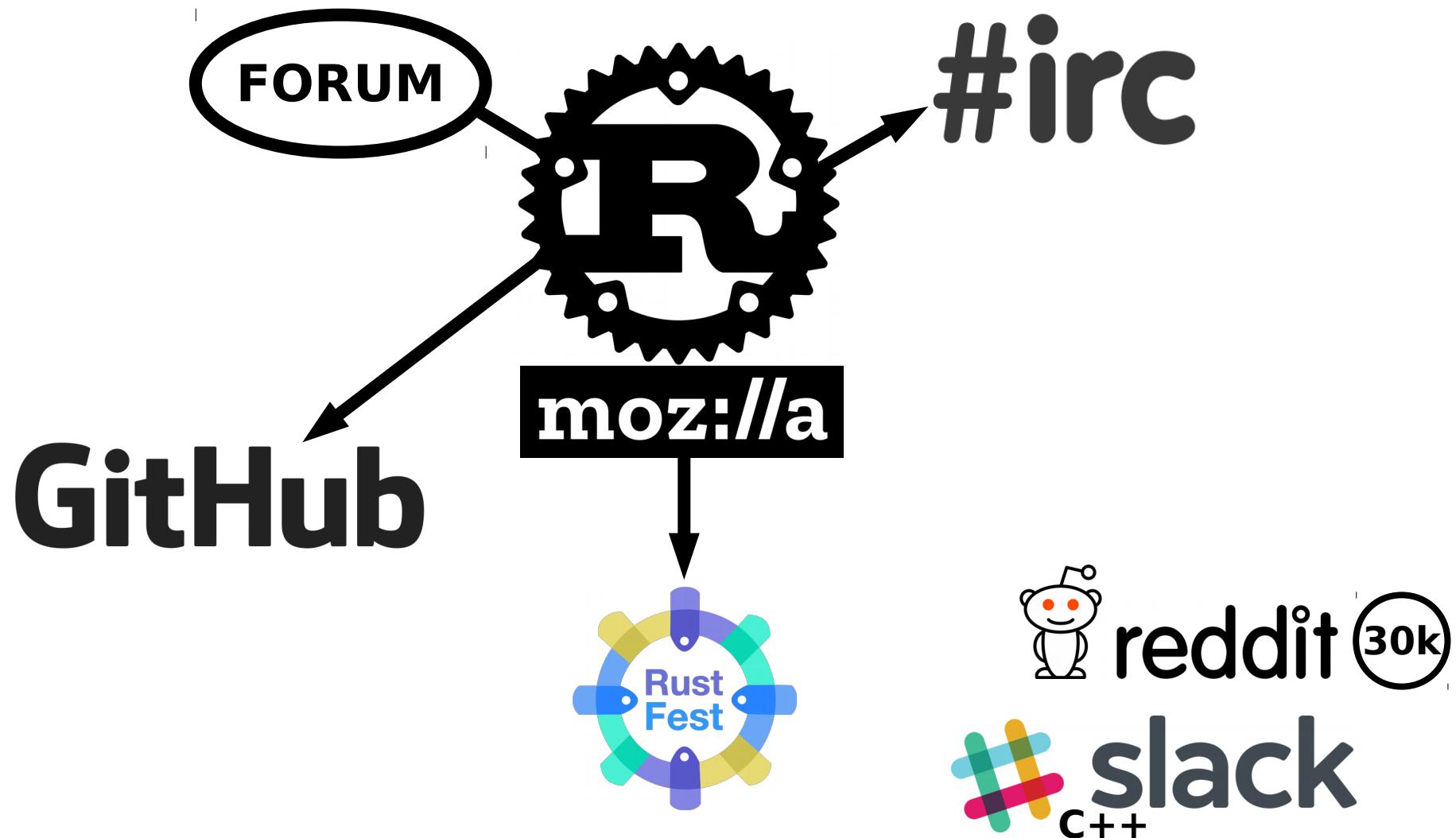
moz://a



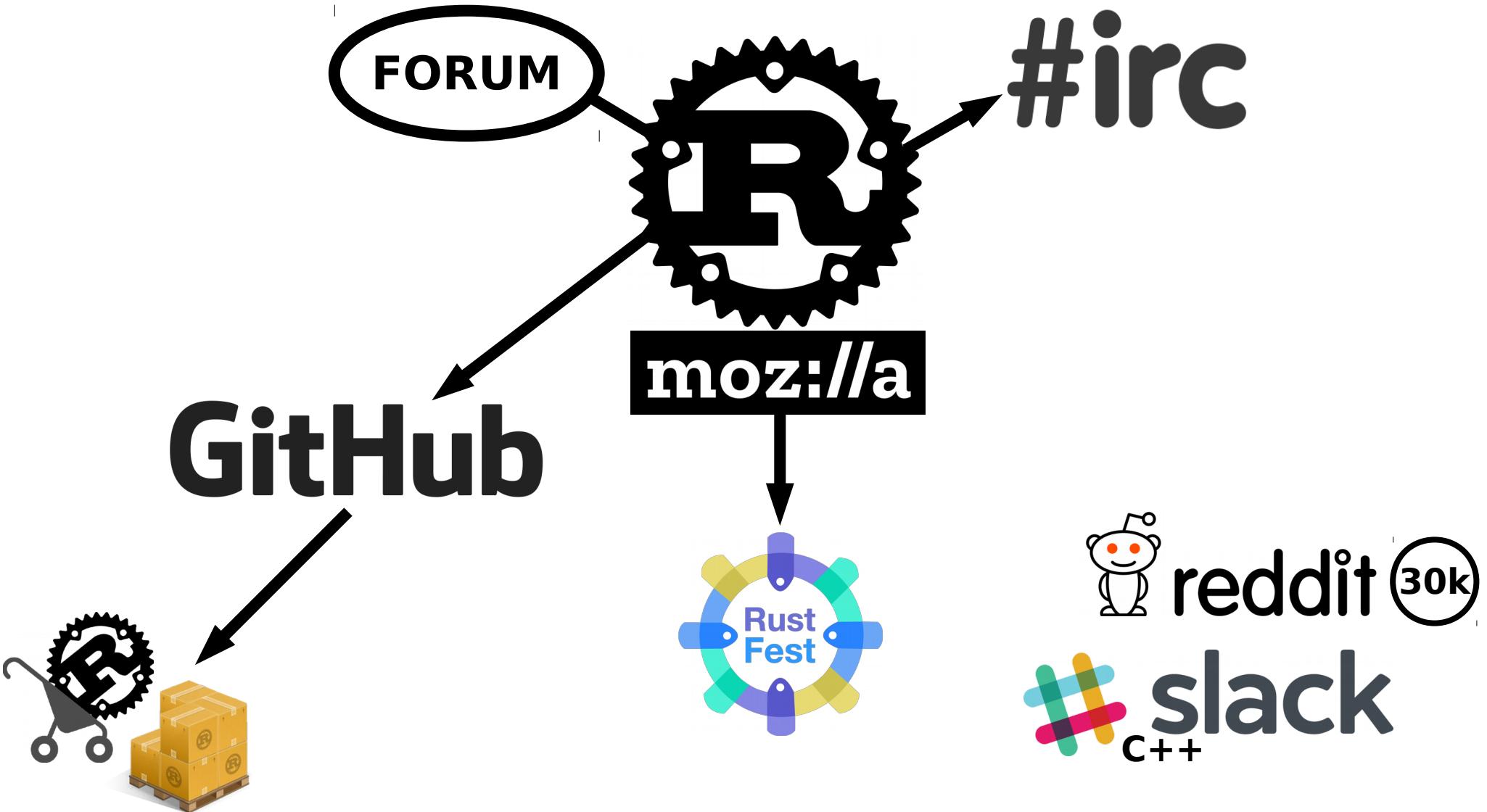
# Rust Community



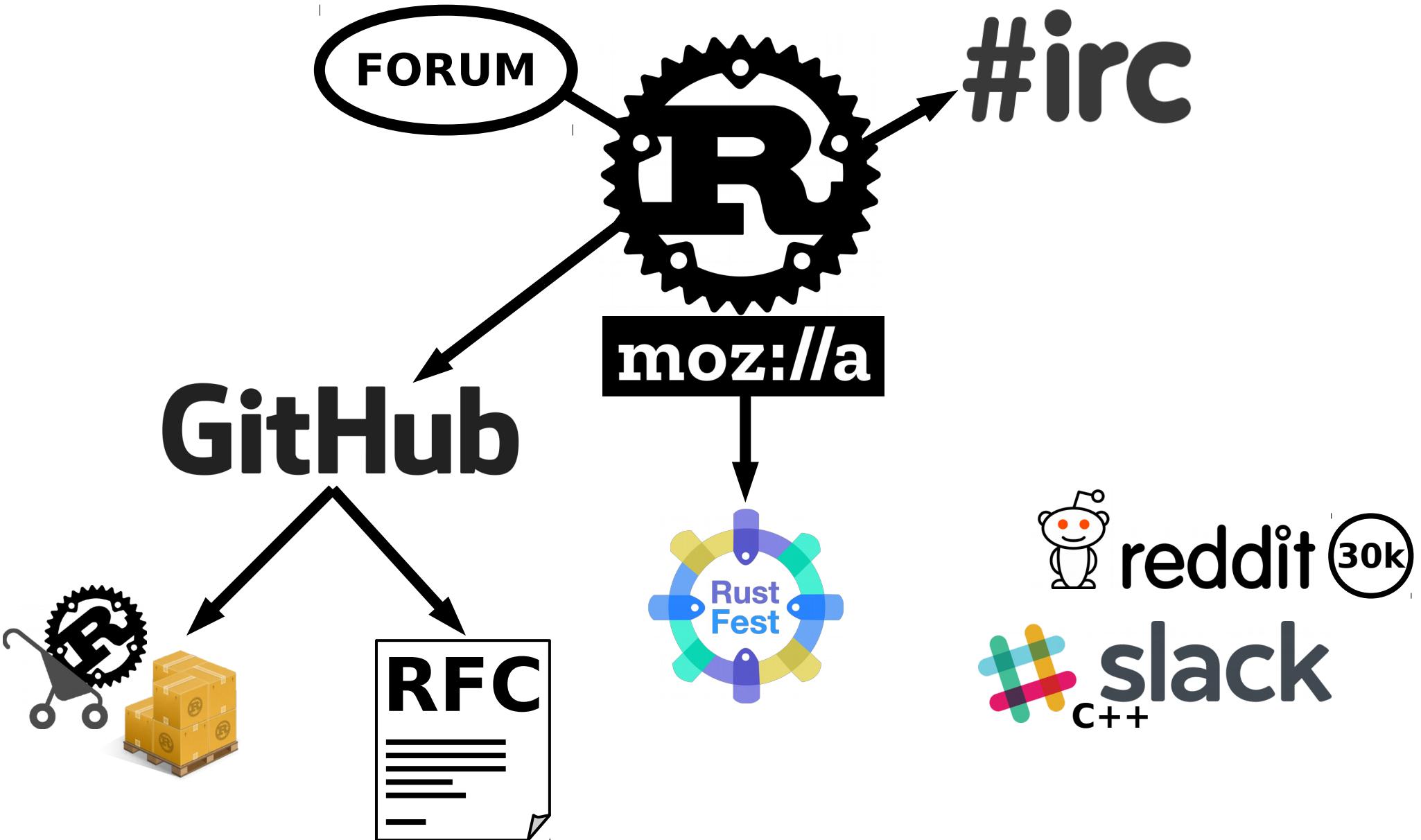
# Rust Community



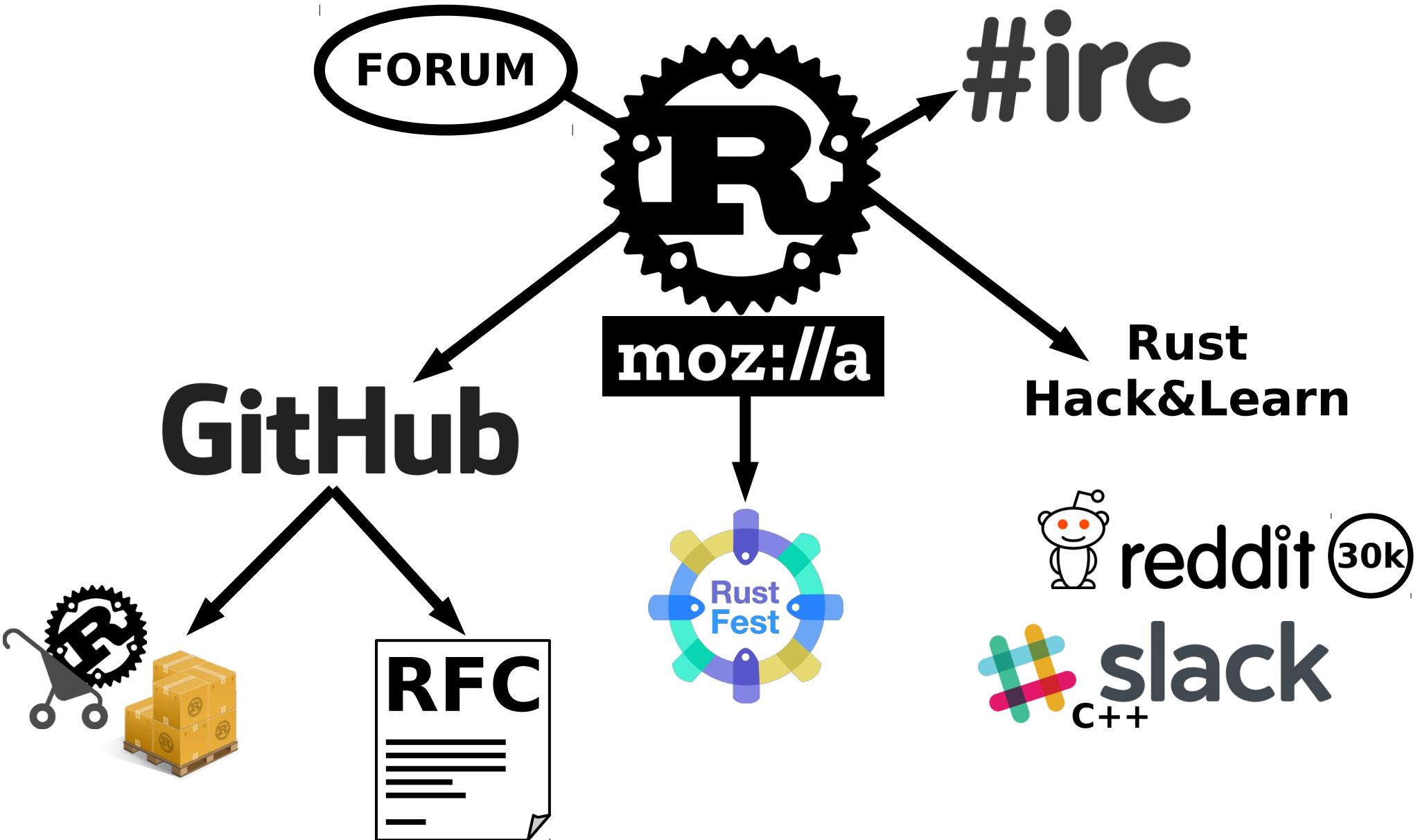
# Rust Community



# Rust Community



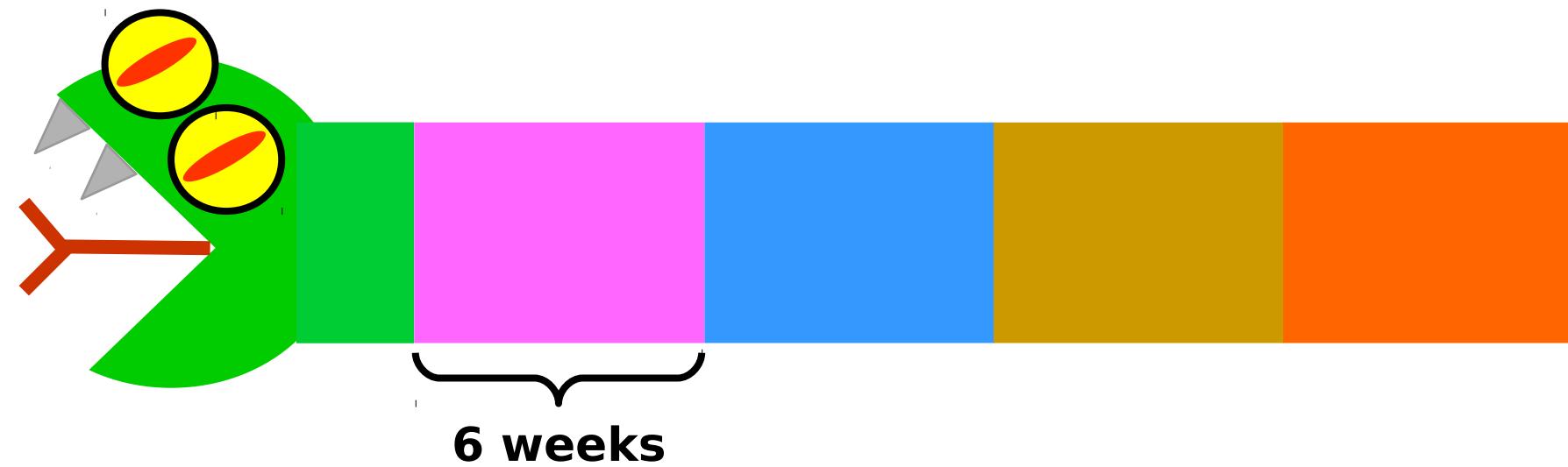
# Rust Community



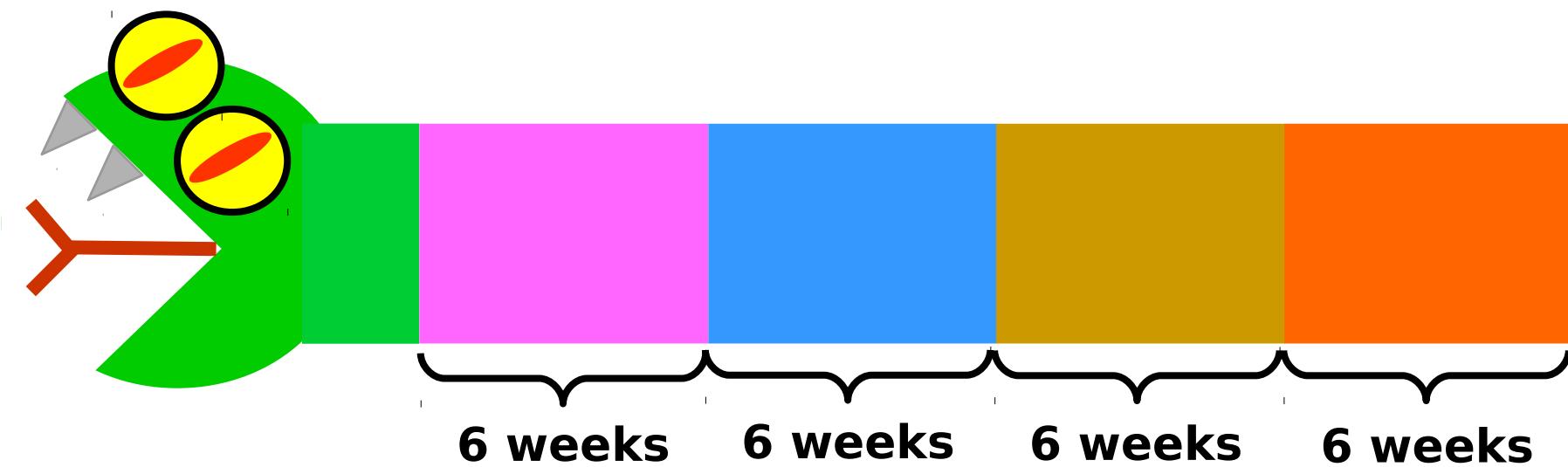
# Rust Pipeline



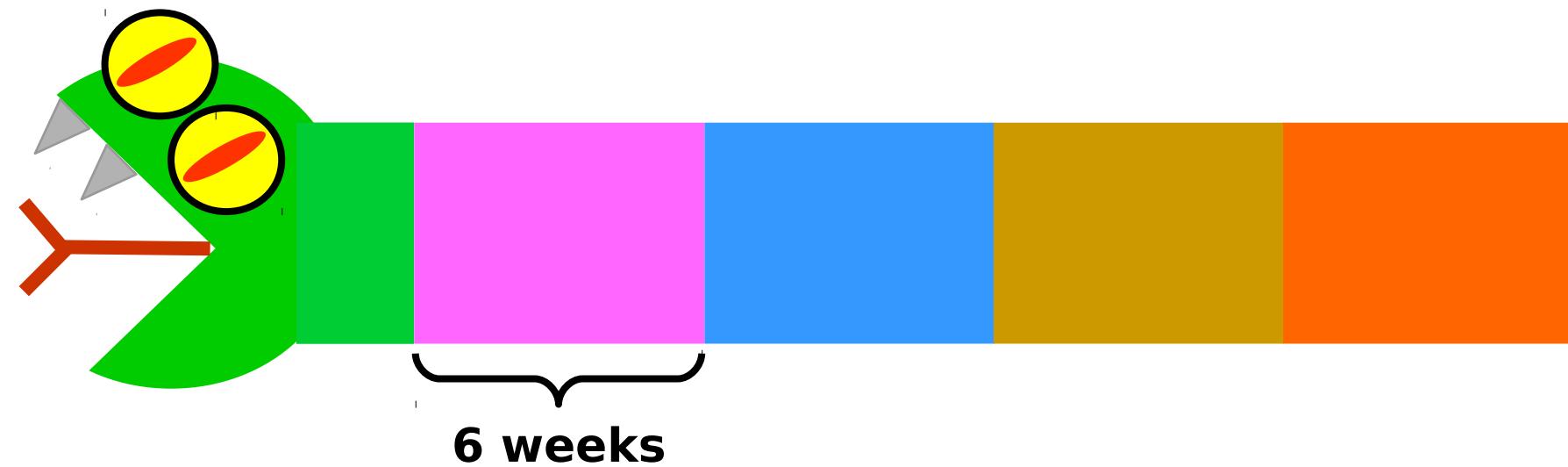
# Rust Pipeline



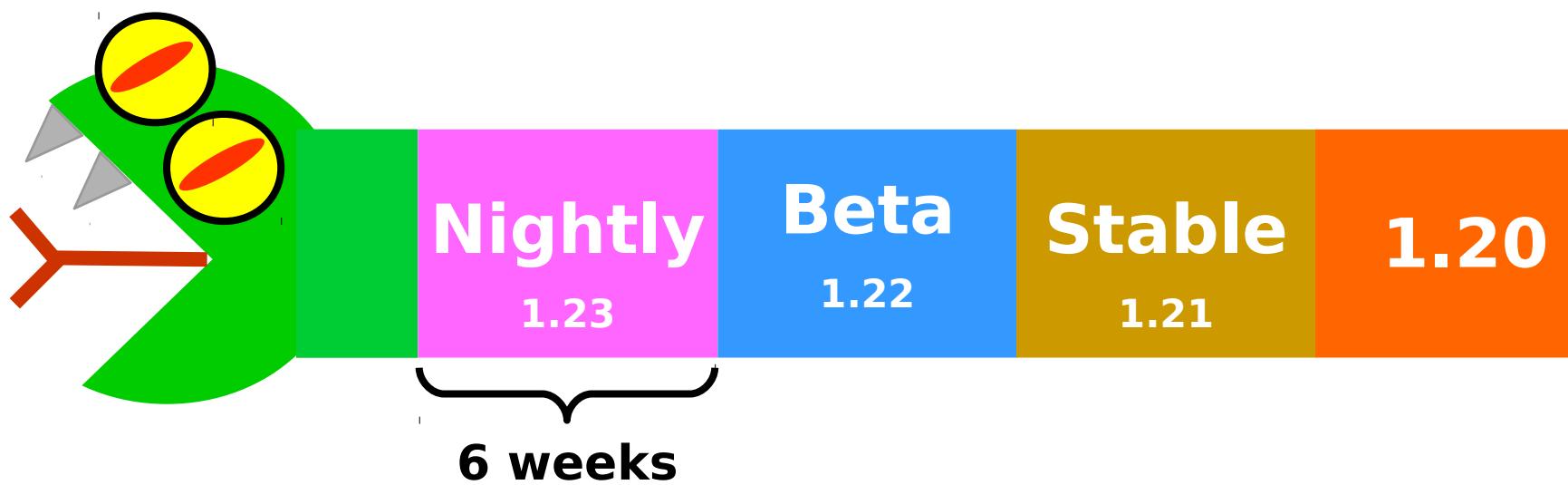
# Rust Pipeline



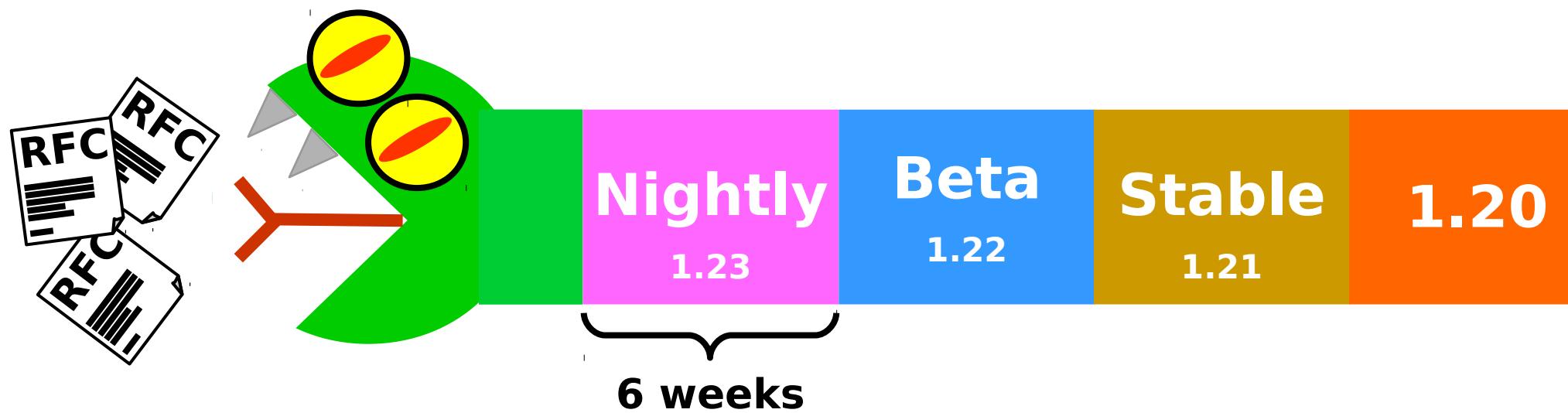
# Rust Pipeline



# Rust Pipeline



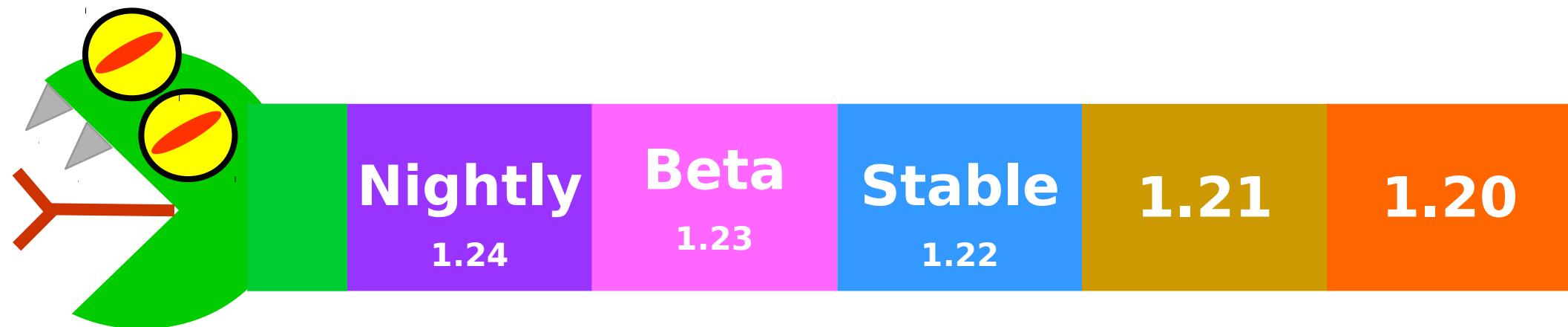
# Rust Pipeline



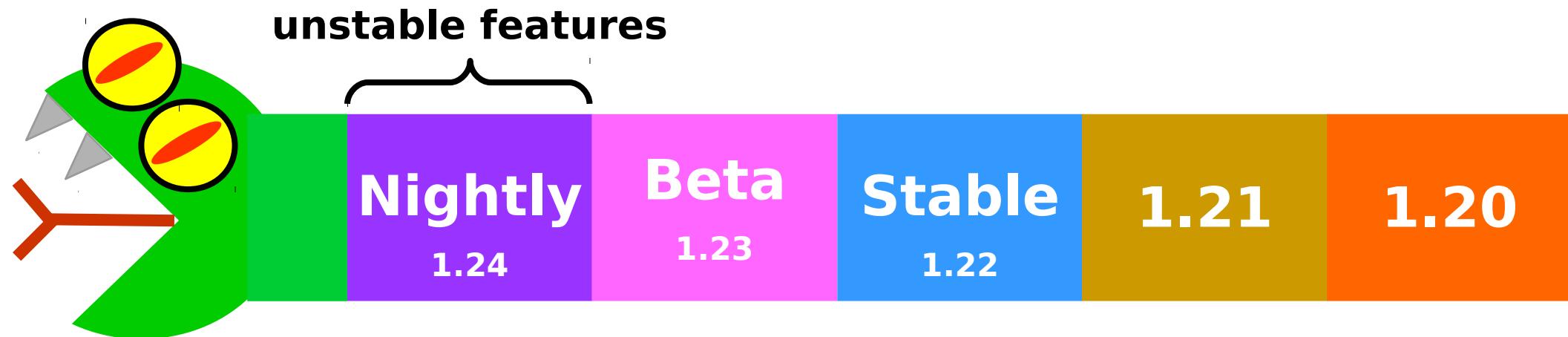
# Rust Pipeline



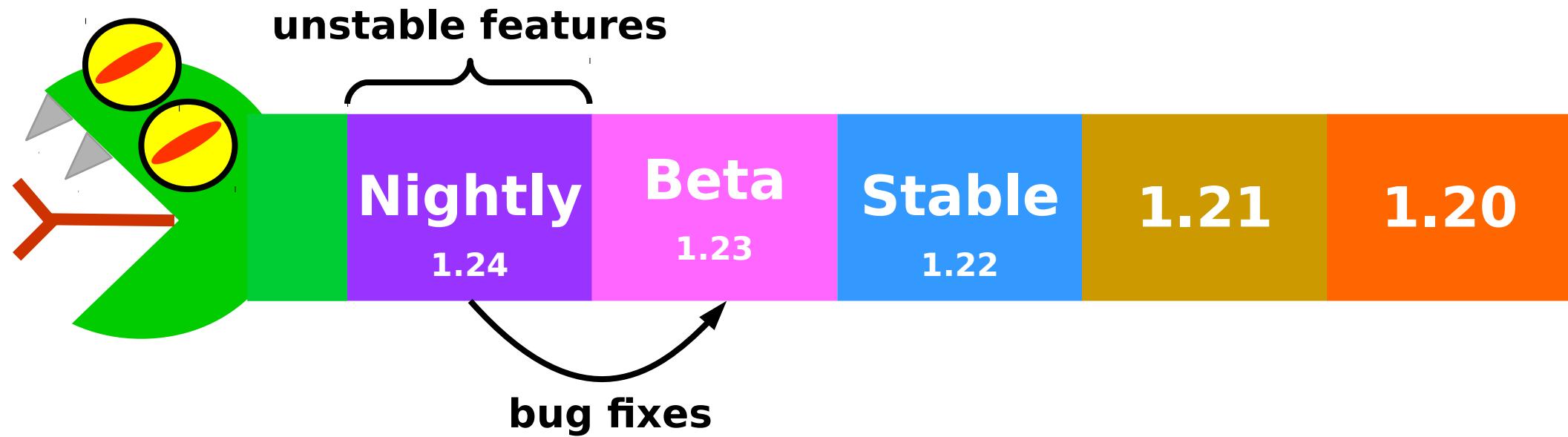
# Rust Pipeline



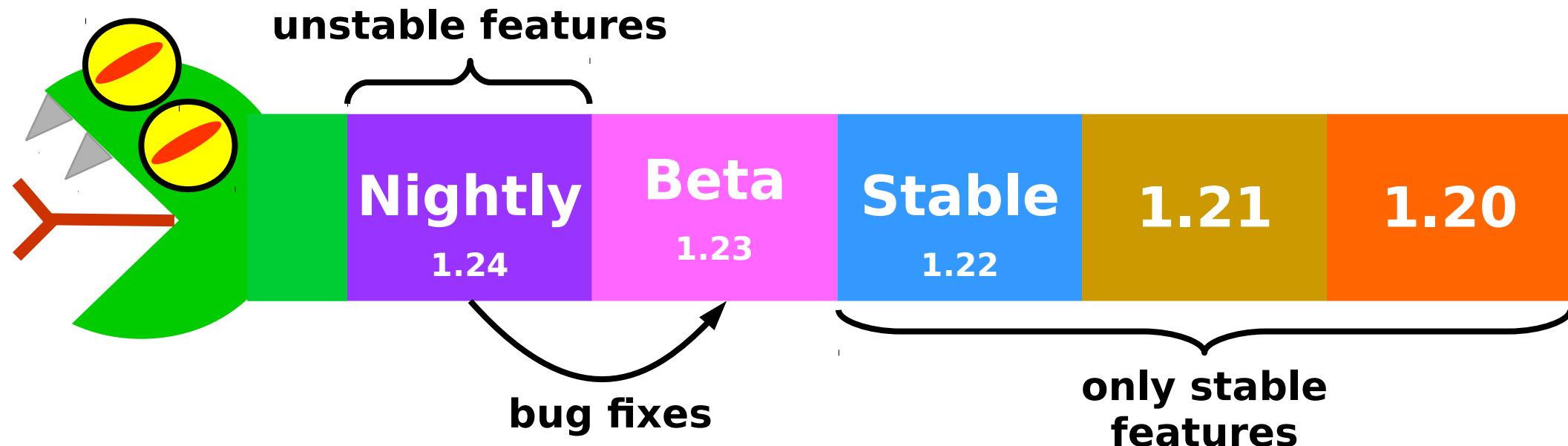
# Rust Pipeline



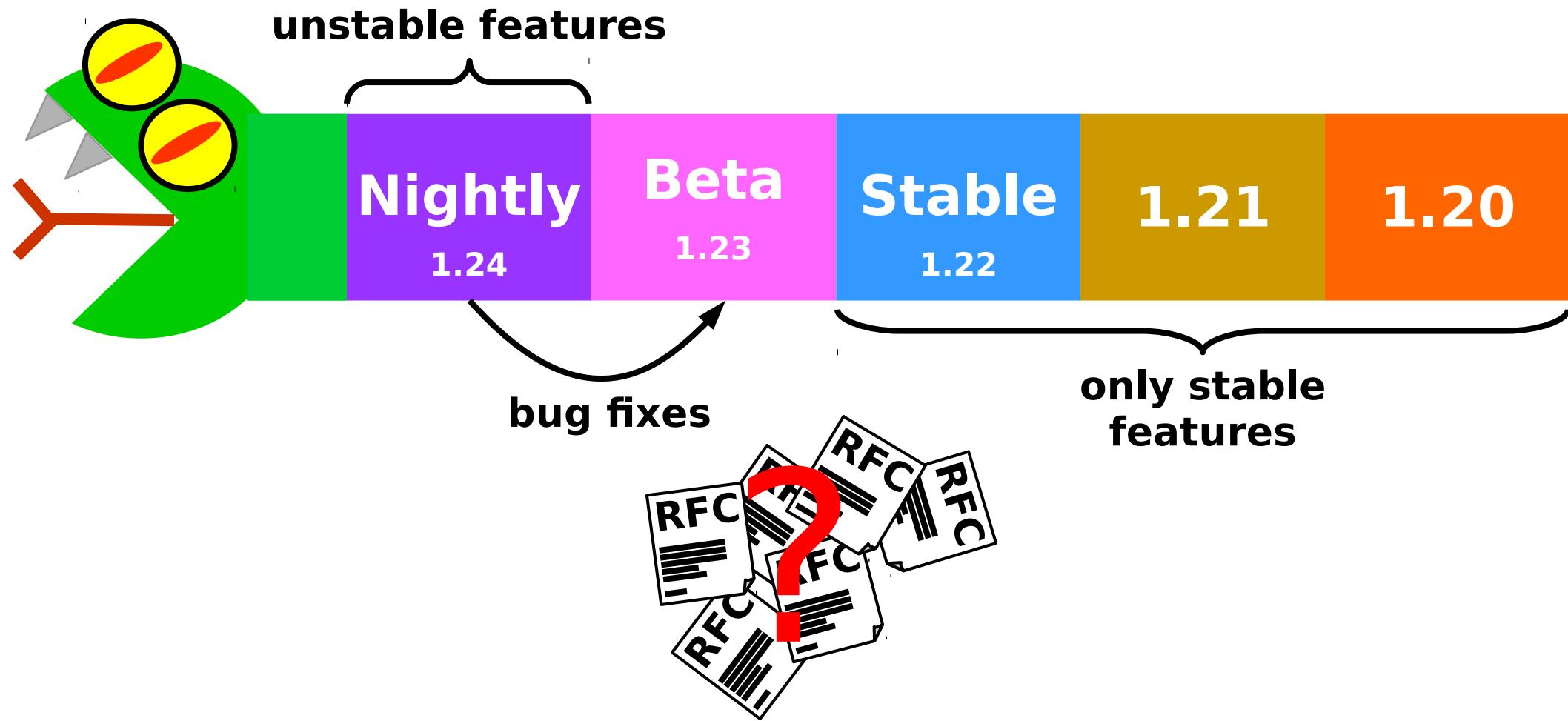
# Rust Pipeline



# Rust Pipeline



# Rust Pipeline



# RFC Process

# GitHub

non-lexical lifetimes #2094

 Merged aturon merged 2 commits into rust-lang:master from nikomatsakis:nll on Sep 29

 Conversation 53     Commits 2     Files changed 1



nikomatsakis commented on Aug 2 • edited by mbrubbeck

Contributor + 😊

Extend Rust's borrow system to support **non-lexical lifetimes** -- these are lifetimes that are based on the control-flow graph, rather than lexical scopes. The RFC describes in detail how to infer these new, more flexible regions, and also describes how to adjust our error messages. The RFC also describes a few other extensions to the borrow checker, the total effect of which is to eliminate many common cases where small, function-local requires would be required to pass the borrow check. (The appendix describes some of the remaining borrow-checker limitations that are not addressed by this RFC.)

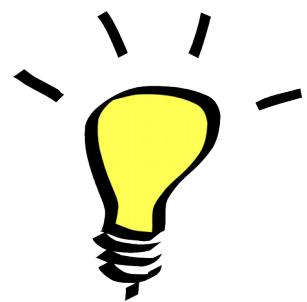
Due to its size and complexity, this RFC is being run through an experimental process. The text of the RFC itself is not in this file -- rather, it can be found at the [non-lexical lifetimes repository](#). Prior to merging, the final version of the text will be added to this PR directly; until then, hosting the RFC at the repository allows for us to track (using open issues or pending PRs) important points of conversation and so forth. Feel free to leave ordinary comments on the PR, or to open issues -- important points will be elevated to issues for further discussion.

The ideas in this RFC have been implemented in [prototype form](#). This prototype includes a simplified control-flow graph that allows one to create the various kinds of region constraints that can arise and implements the region inference algorithm which then solves those constraints.

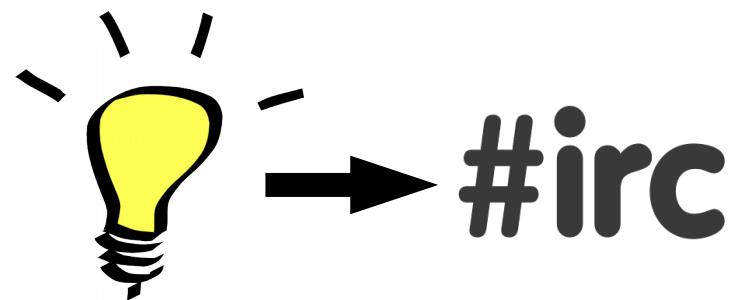
Rendered.

196 24 249 2 104

# RFC Process



# RFC Process

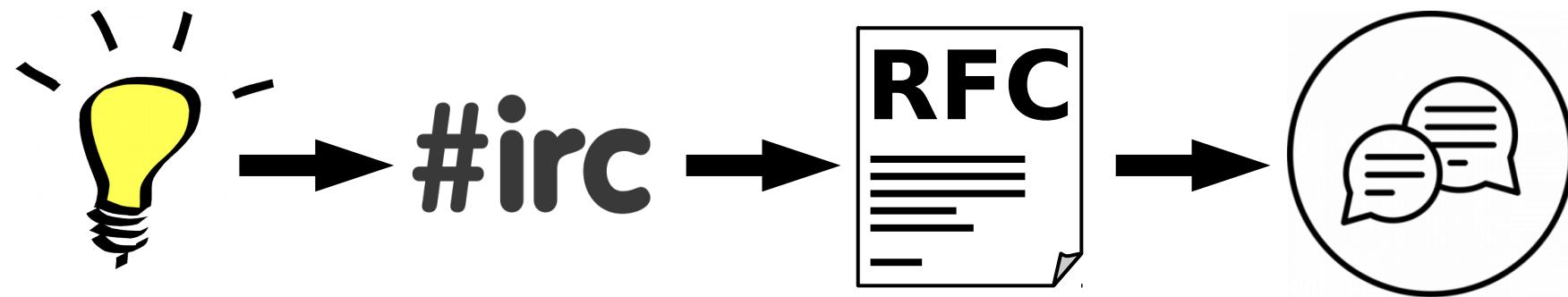


#irc

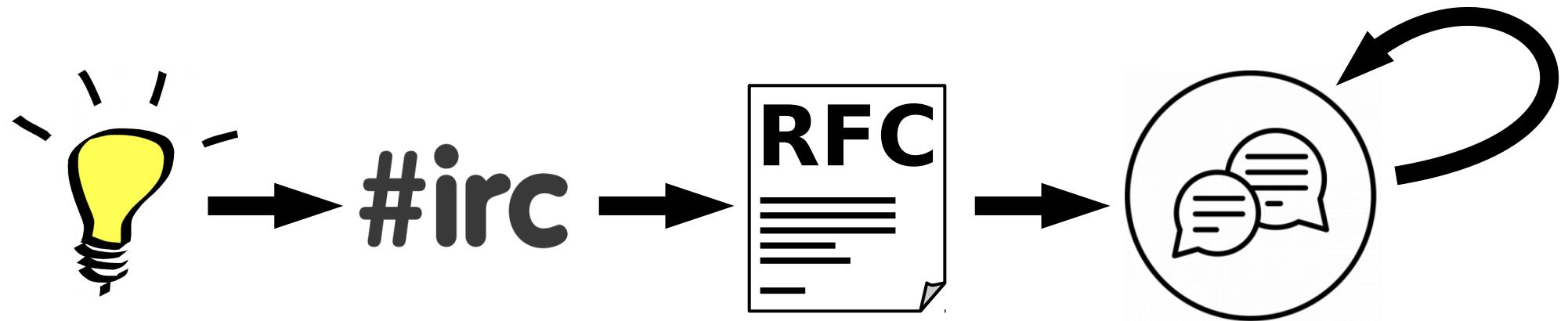
# RFC Process



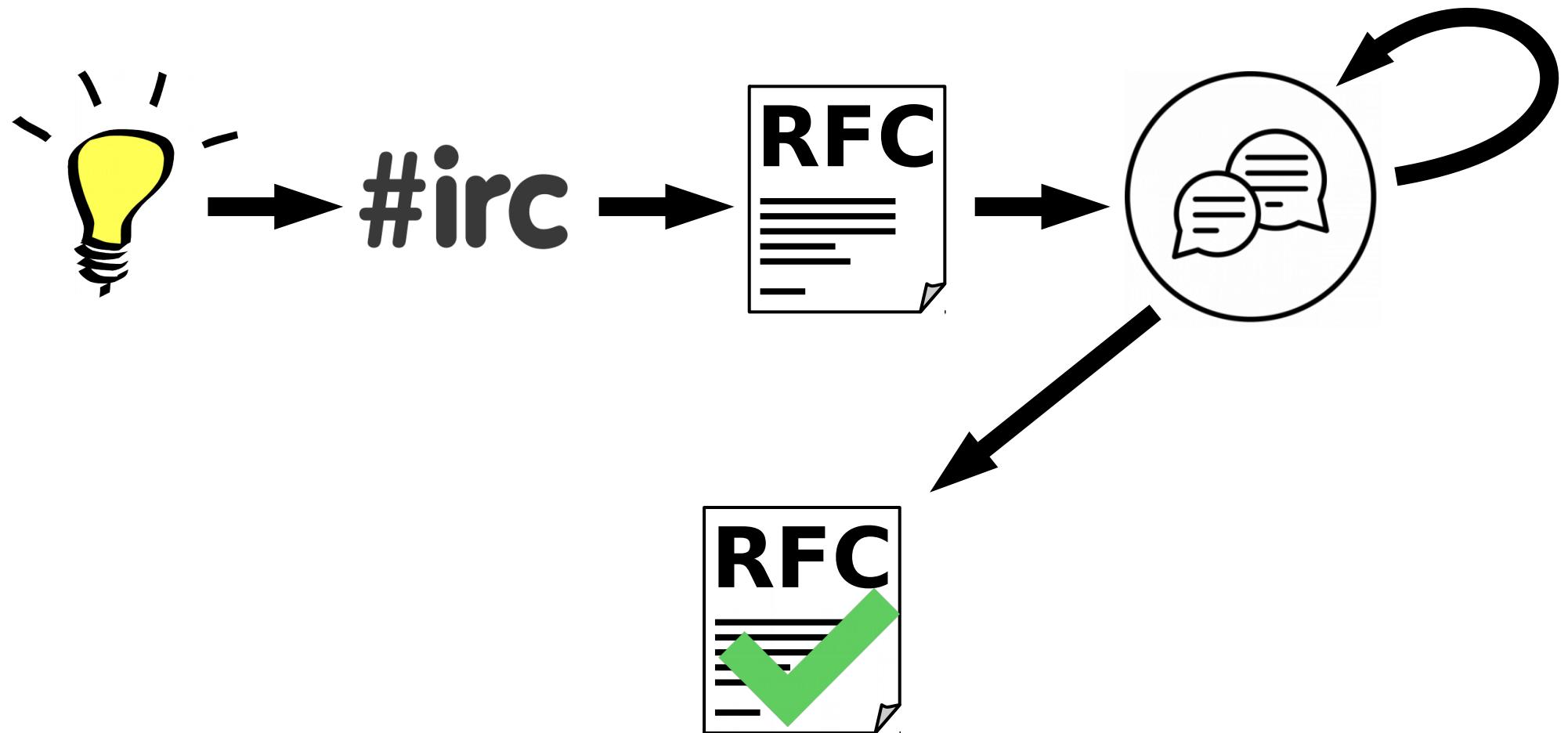
# RFC Process



# RFC Process



# RFC Process





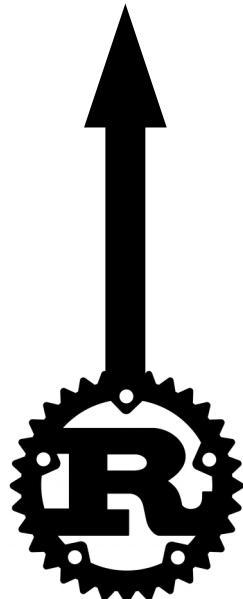
Sea of Hands | Wat Rung Khun , Chiang Mai , Thailand



rustup

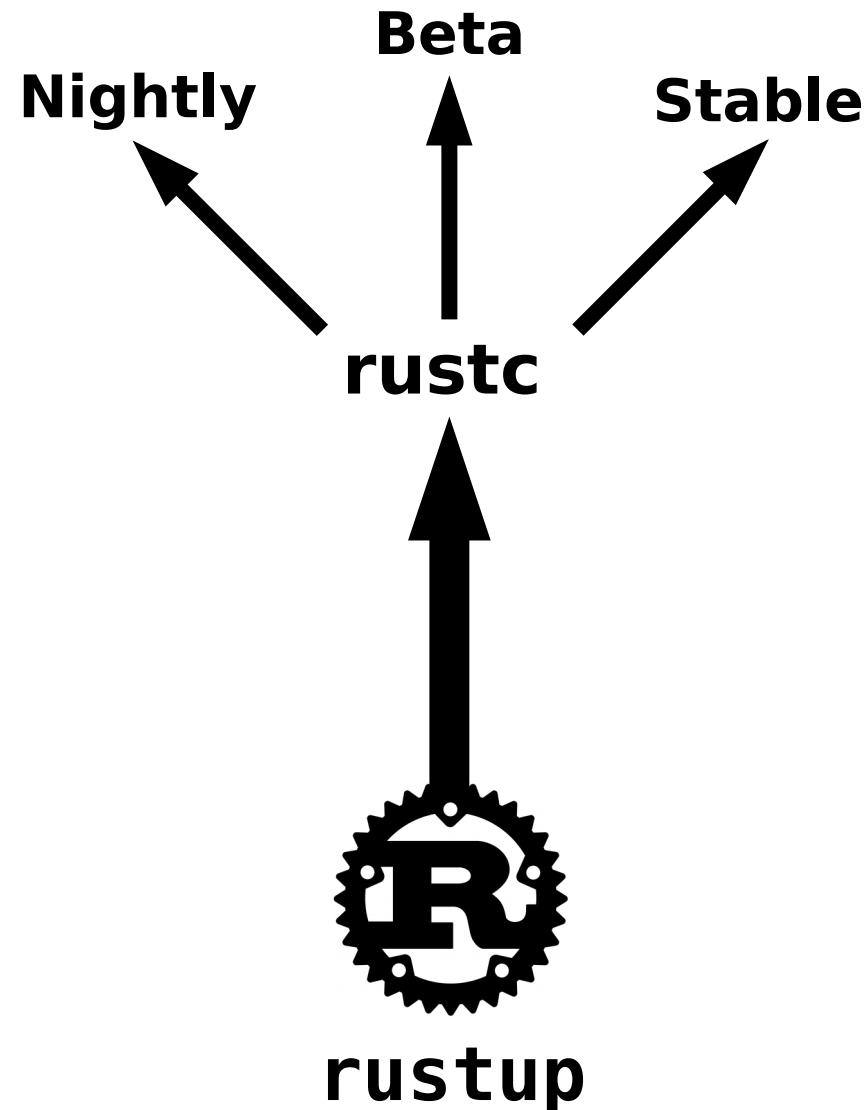
[www.rustup.rs](http://www.rustup.rs)

**rustc**

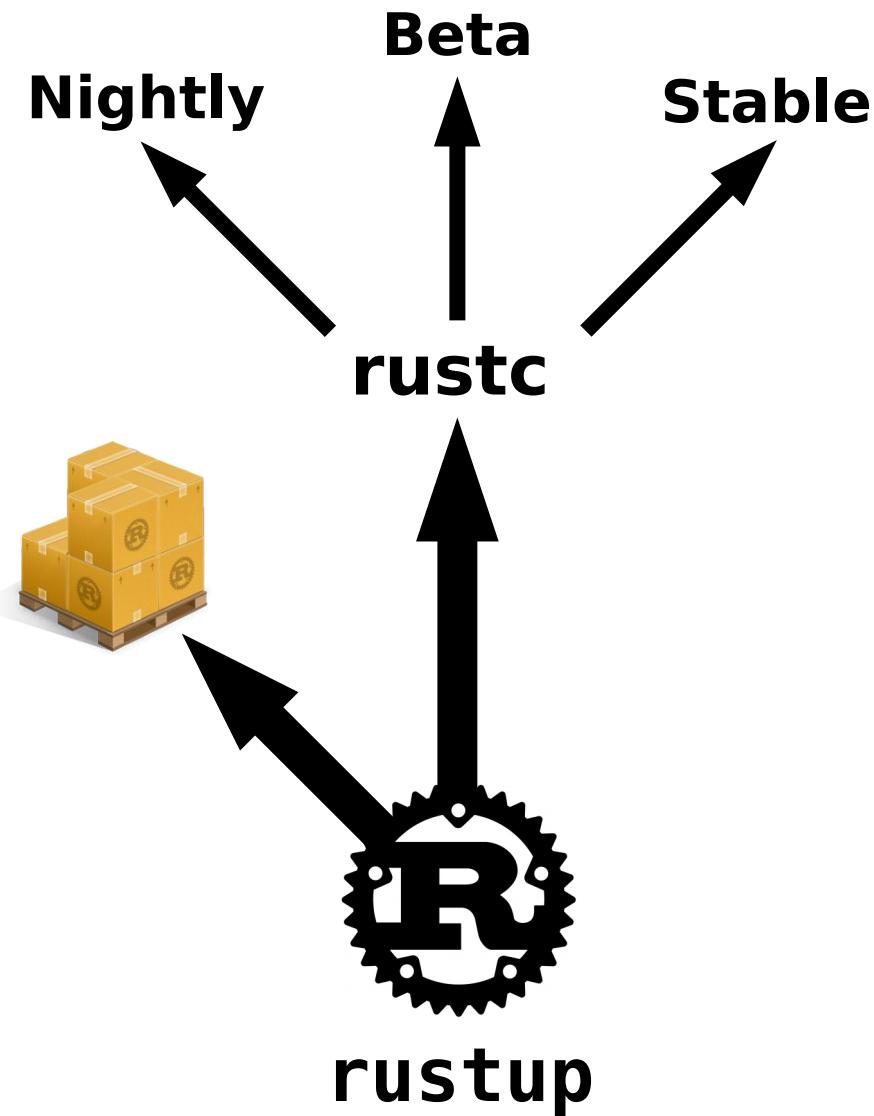


**rustup**

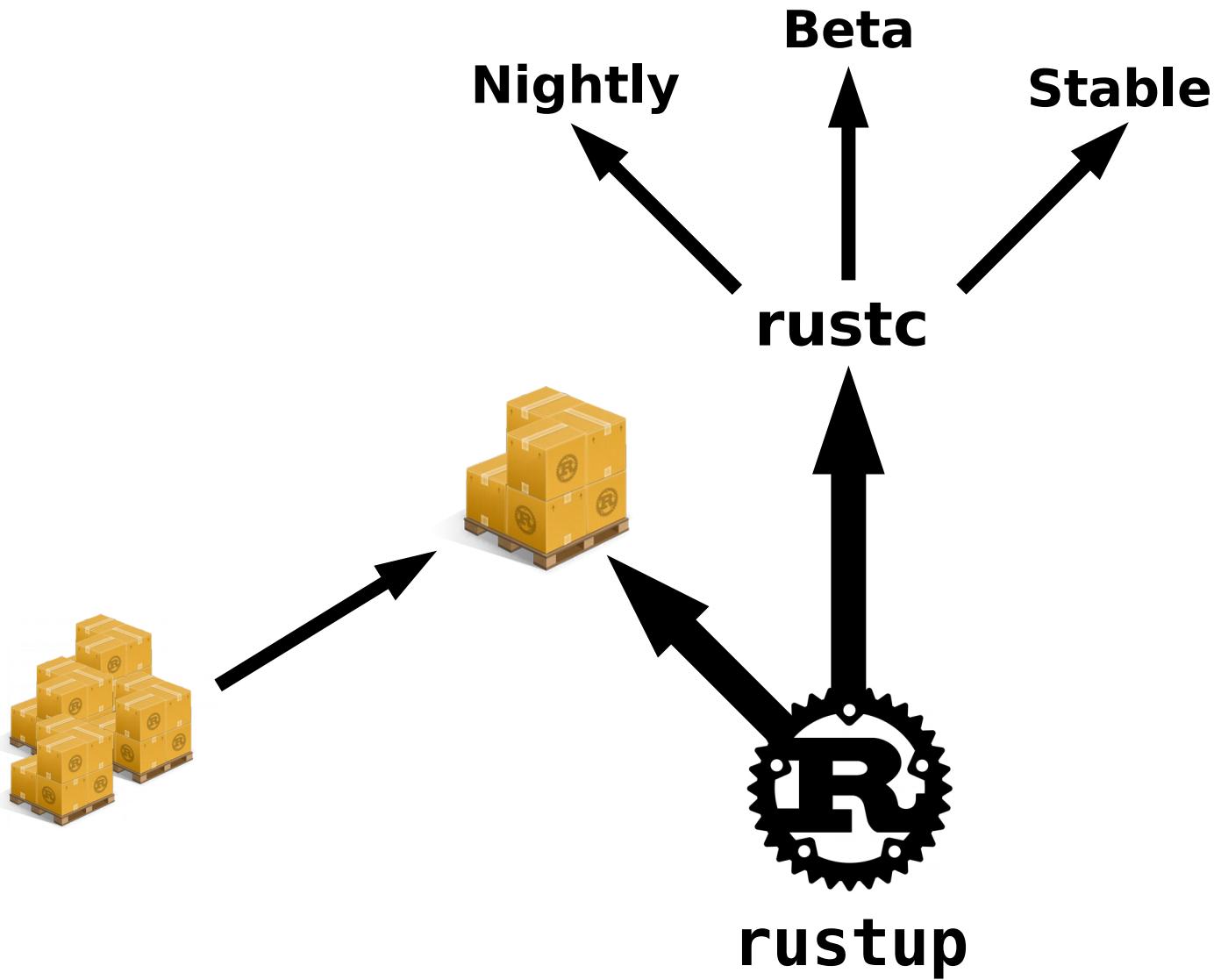
**[www.rustup.rs](http://www.rustup.rs)**



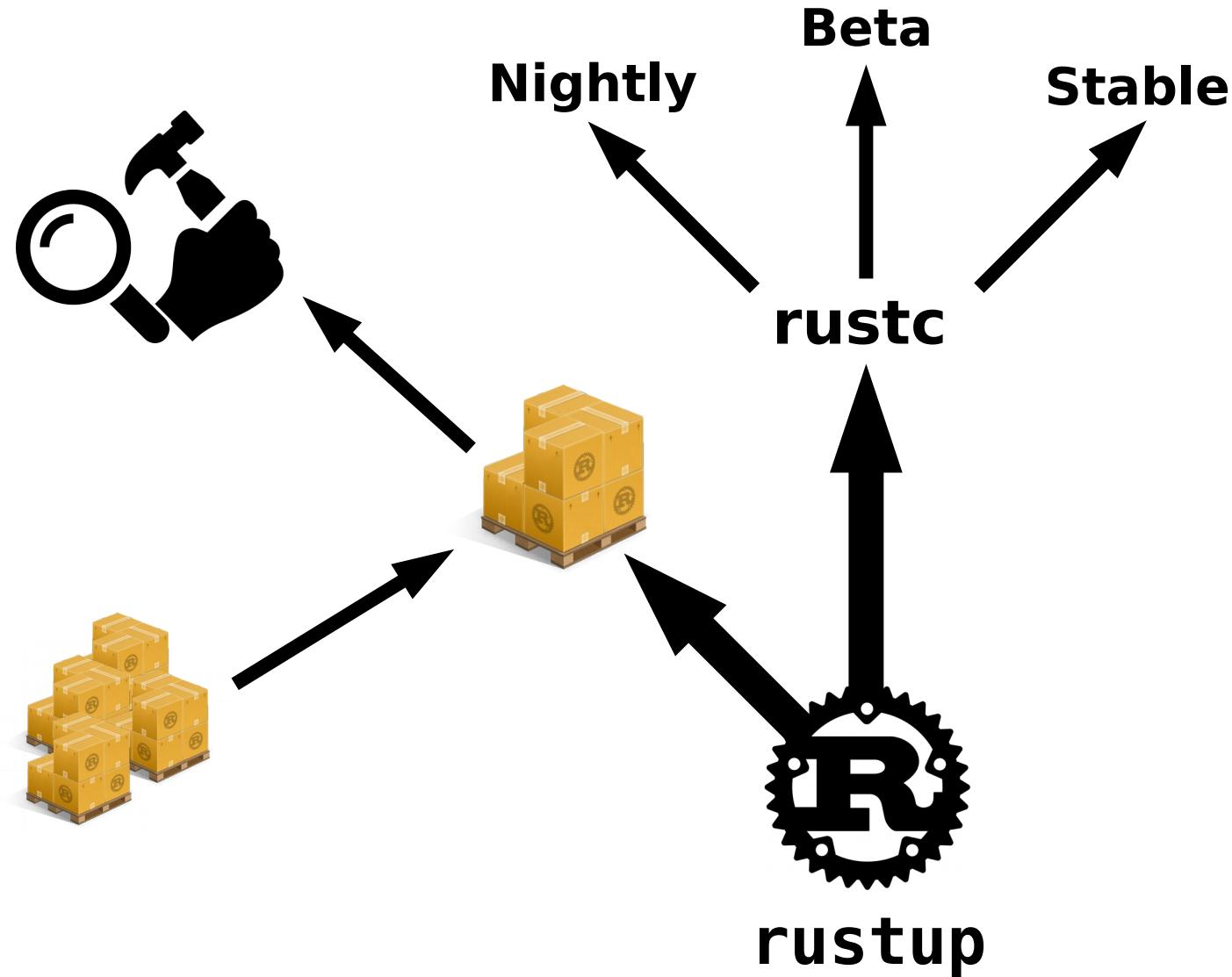
[www.rustup.rs](http://www.rustup.rs)



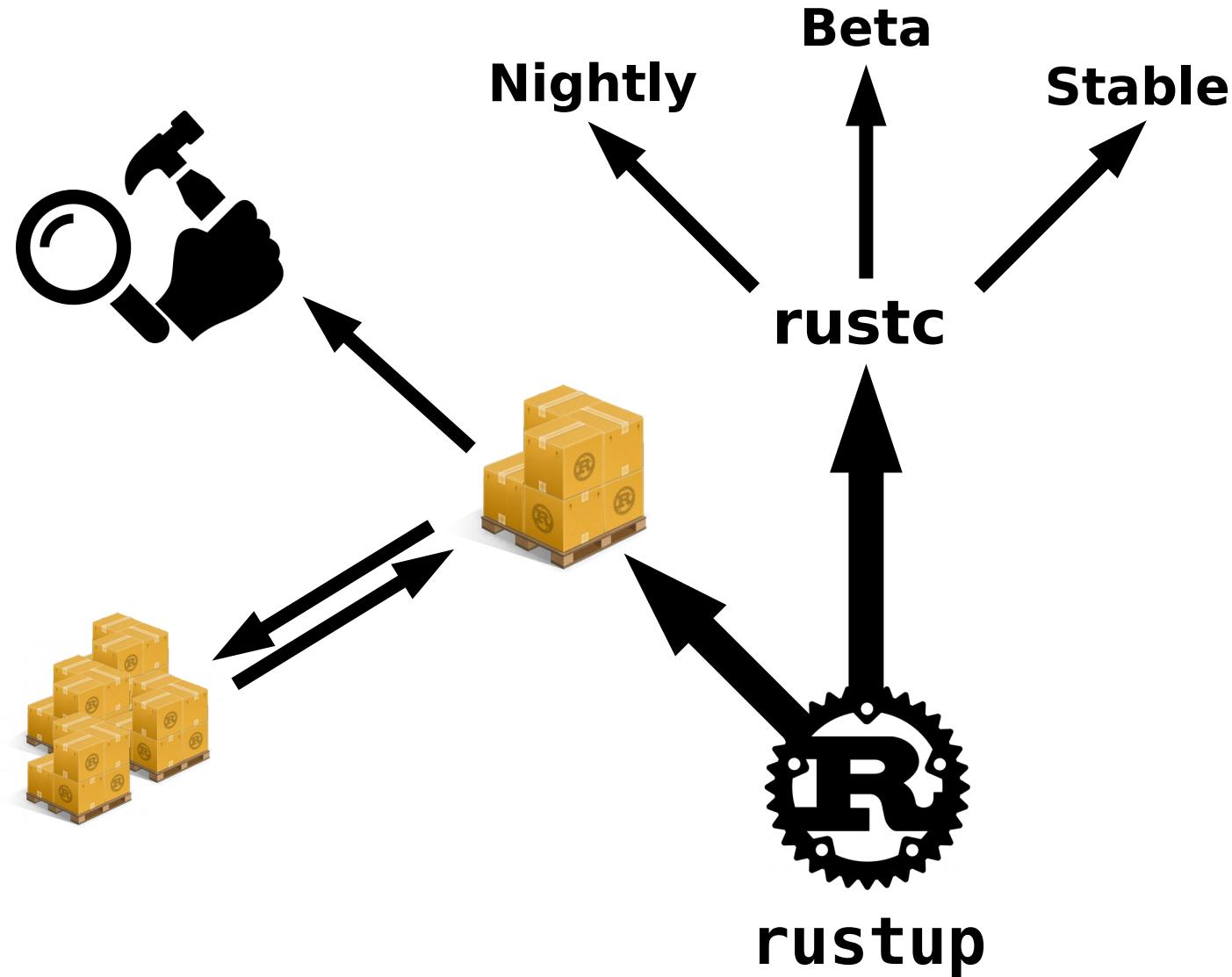
[www.rustup.rs](http://www.rustup.rs)



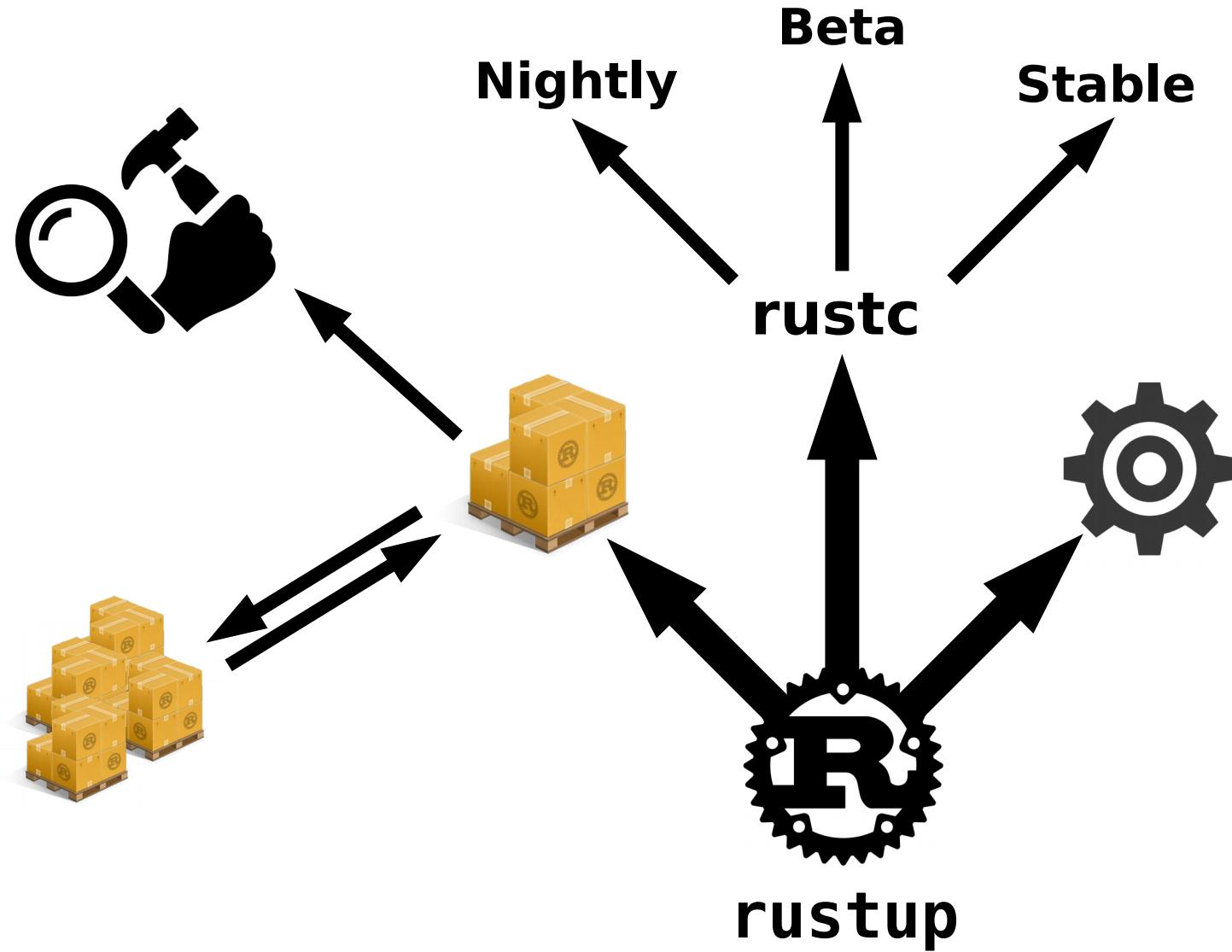
[www.rustup.rs](http://www.rustup.rs)



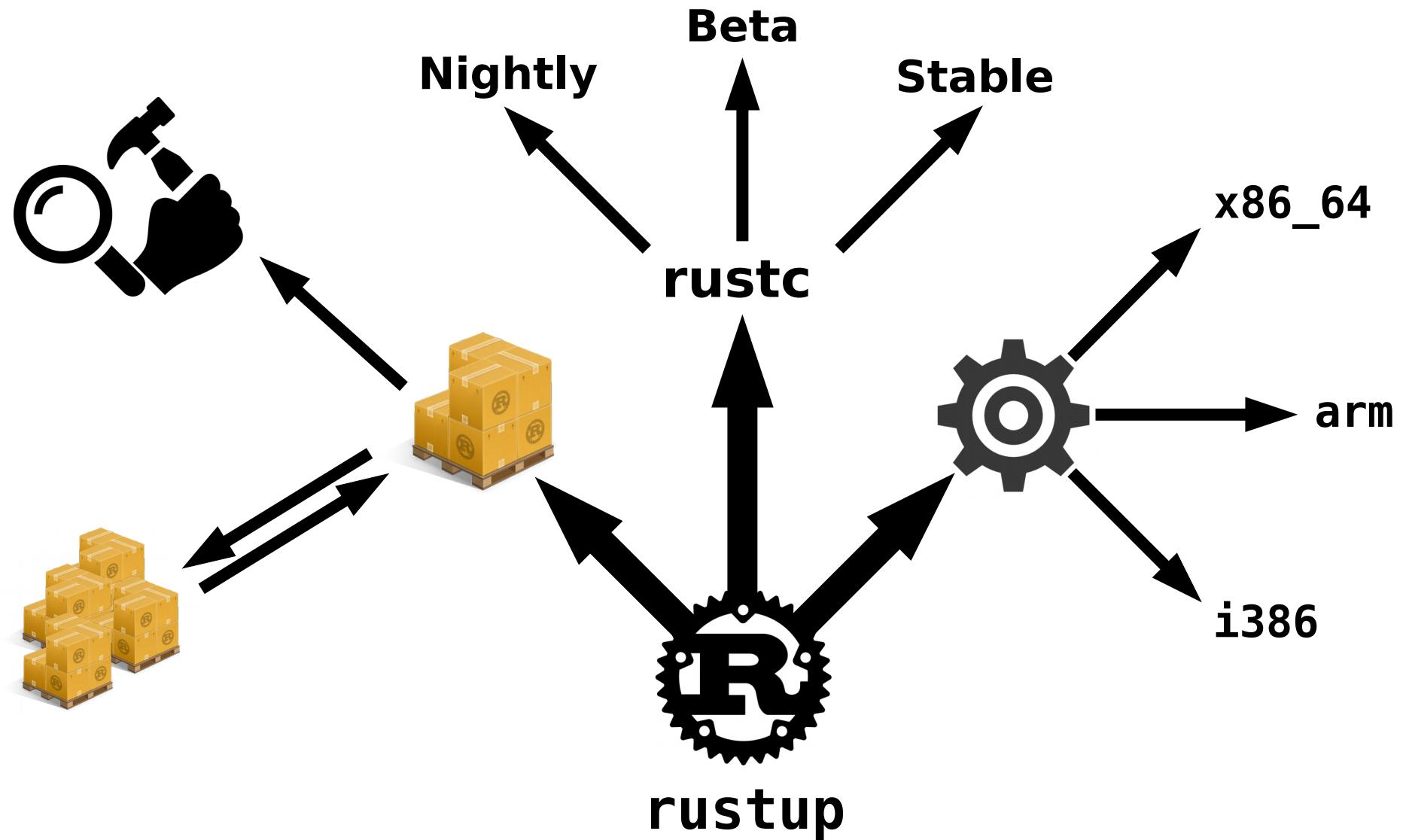
[www.rustup.rs](http://www.rustup.rs)



[www.rustup.rs](http://www.rustup.rs)



[www.rustup.rs](http://www.rustup.rs)



[www.rustup.rs](http://www.rustup.rs)

or ....



[play.rust-lang.org](https://play.rust-lang.org)

Run ► ASM LLVM IR MIR Tools Format Clippy Share Mode Debug Release Channel Stable Beta Nightly Config ?

```
1 fn main() {
2     println!("Hello, World!")
3 }
4 |
```

# Livecoding

# **EXPRESSIVE C++17**

---

## **CODING CHALLENGE**

# EXPRESSIVE C++17

## CODING CHALLENGE

```
First Name,Last Name,Age,City,Eyes color,Species  
John,Doe,32,Tokyo,Blue,Human  
Flip,Helm,12,Canberra,Red,Unknown  
Terdos,Bendarian,165,Cracow,Blue,Magic tree  
Dominik,Elpos,33,Paris,Purple,Orc  
Brad,Doe,42,Dublin,Blue,Human  
Ewan,Grath,51,New Delhi,Green,Human
```



City -> London

```
First Name,Last Name,Age,City,Eyes color,Species  
John,Doe,32,London,Blue,Human  
Flip,Helm,12, London,Red,Unknown  
Terdos,Bendarian,165,London,Blue,Magic tree  
Dominik,Elpos,33,London,Purple,Orc  
Brad,Doe,42,London,Blue,Human  
Ewan,Grath,51,London,Green,Human
```

# EXPRESSIVE C++17

## CODING CHALLENGE

```
First Name,Last Name,Age,City,Eyes color,Species  
John,Doe,32,Tokyo,Blue,Human  
Flip,Helm,12,Canberra,Red,Unknown  
Terdos,Bendarian,165,Cracow,Blue,Magic tree  
Dominik,Elpos,33,Paris,Purple,Orc  
Brad,Doe,42,Dublin,Blue,Human  
Ewan,Grath,51,New Delhi,Green,Human
```

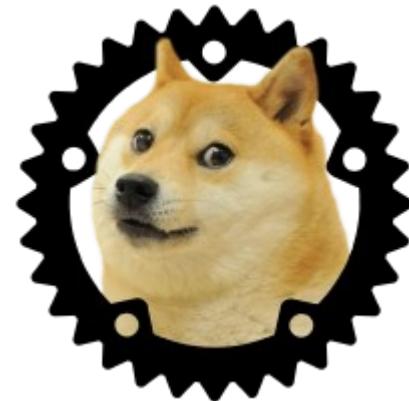


City -> London

```
First Name,Last Name,Age,City,Eyes color,Species  
John,Doe,32,London,Blue,Human  
Flip,Helm,12, London,Red,Unknown  
Terdos,Bendarian,165,London,Blue,Magic tree  
Dominik,Elpos,33,London,Purple,Orc  
Brad,Doe,42,London,Blue,Human  
Ewan,Grath,51,London,Green,Human
```

If the input file is empty => “input file missing”  
column  $\notin$  table => “column name doesn’t exists in the input file”

# Want more Rust?



[rust-lang.github.io/book](https://rust-lang.github.io/book)



[users.rust-lang.org](https://users.rust-lang.org)

#irc [#rust-beginners  
on  
irc.mozilla.org](https://irc.mozilla.org)



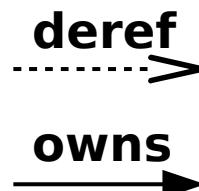
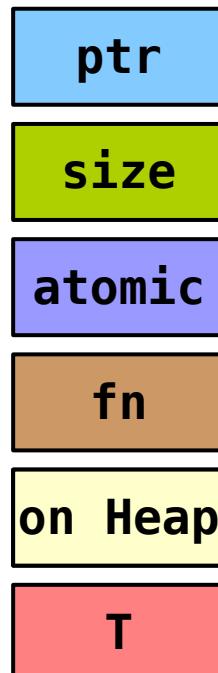
[intorust.com](https://intorust.com)

# Optional Slides

# Standard Data Structures

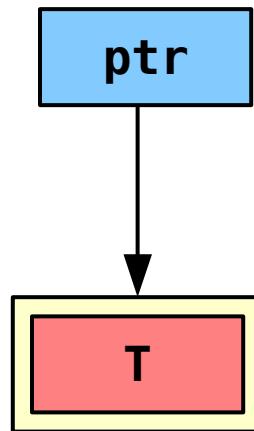
Inspired by “Rust container cheat sheet” from Raph Levinen

## Legend



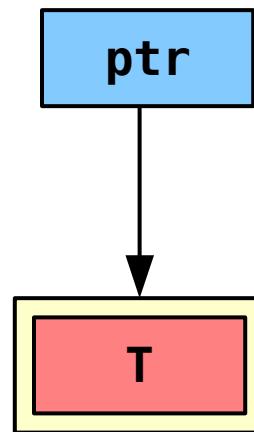
# Rust's Box

`Box<T>`

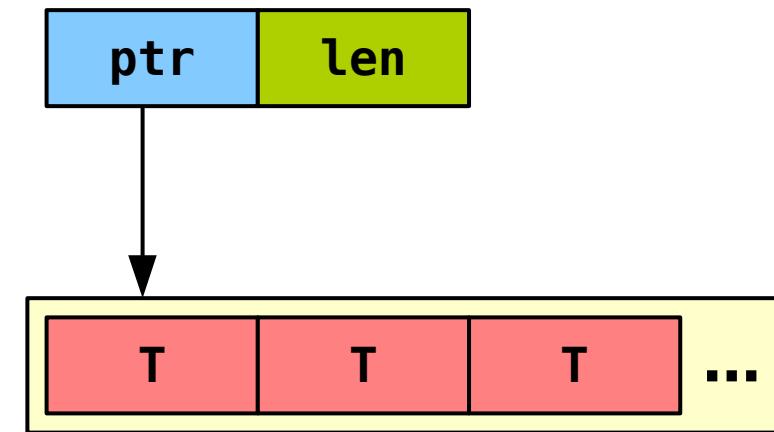


# Rust's Box

**Box<T>**

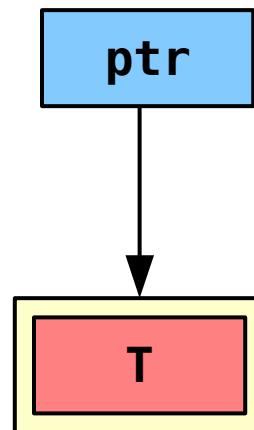


**Box<[T]>**

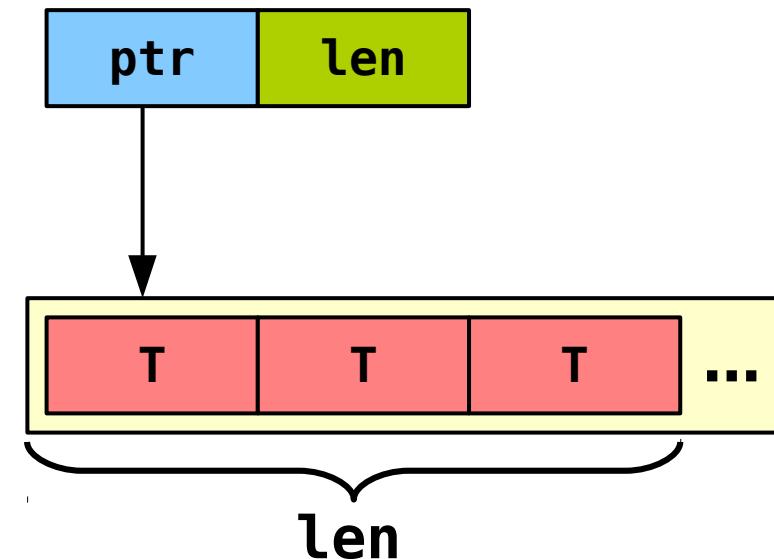


# Rust's Box

**Box<T>**

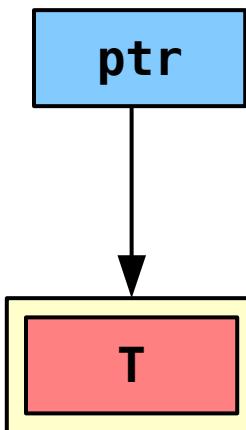


**Box<[T]>**

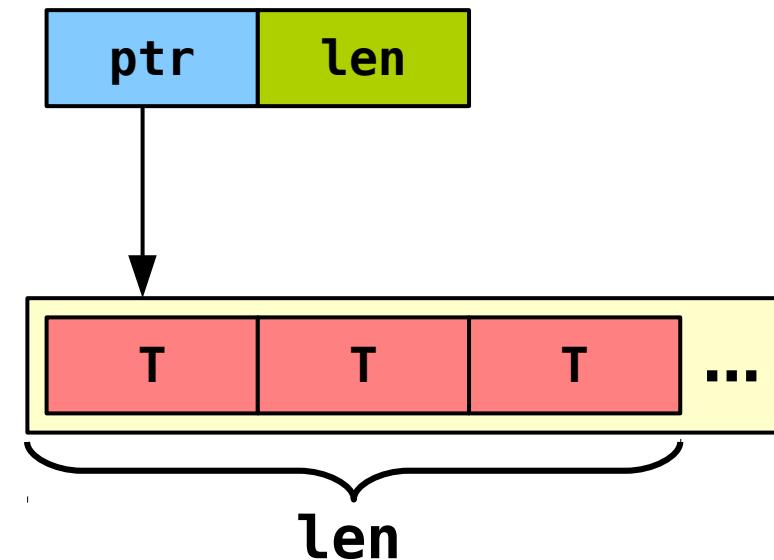


# Rust's Box

**Box<T>**



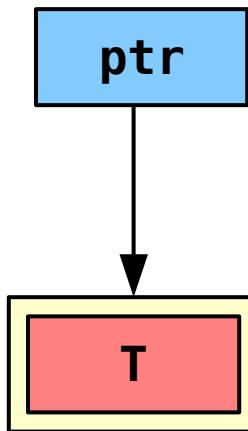
**Box<[T]>**



`T: Sized`

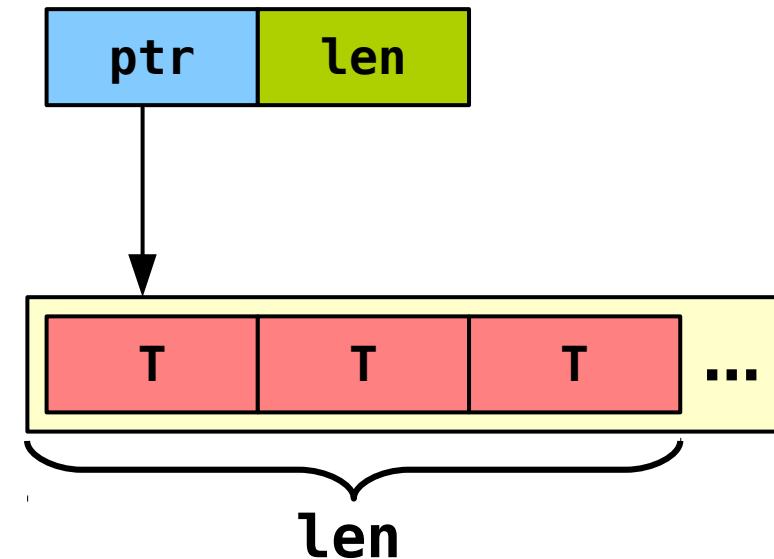
# Rust's Box

**Box<T>**



**T: Sized**

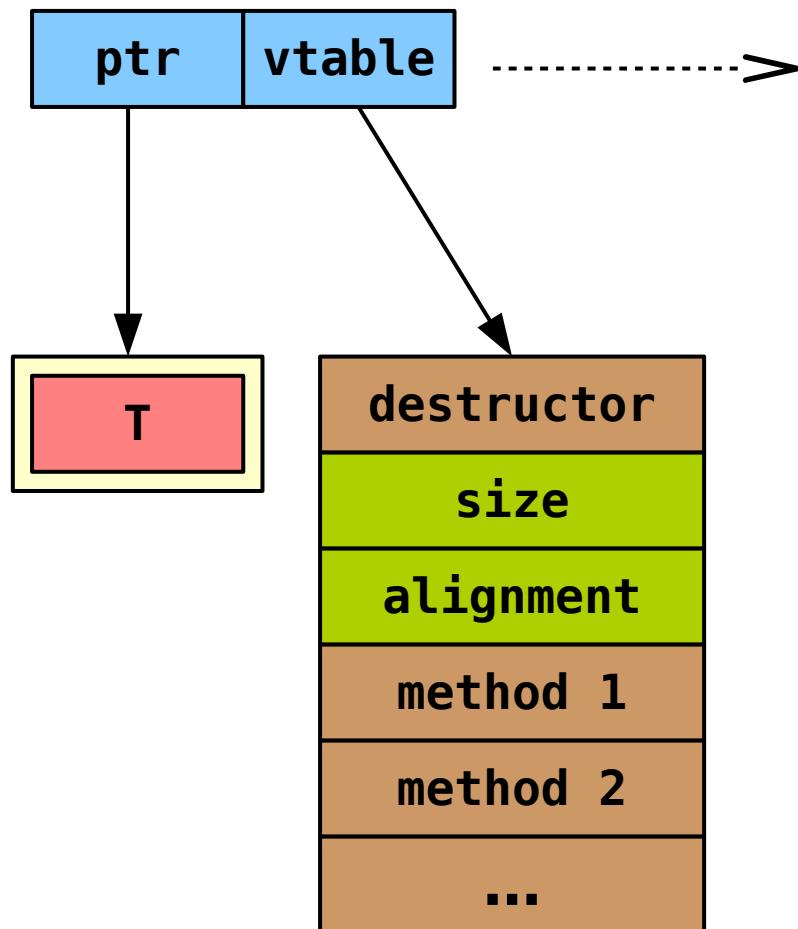
**Box<[T]>**



**Unsized types**

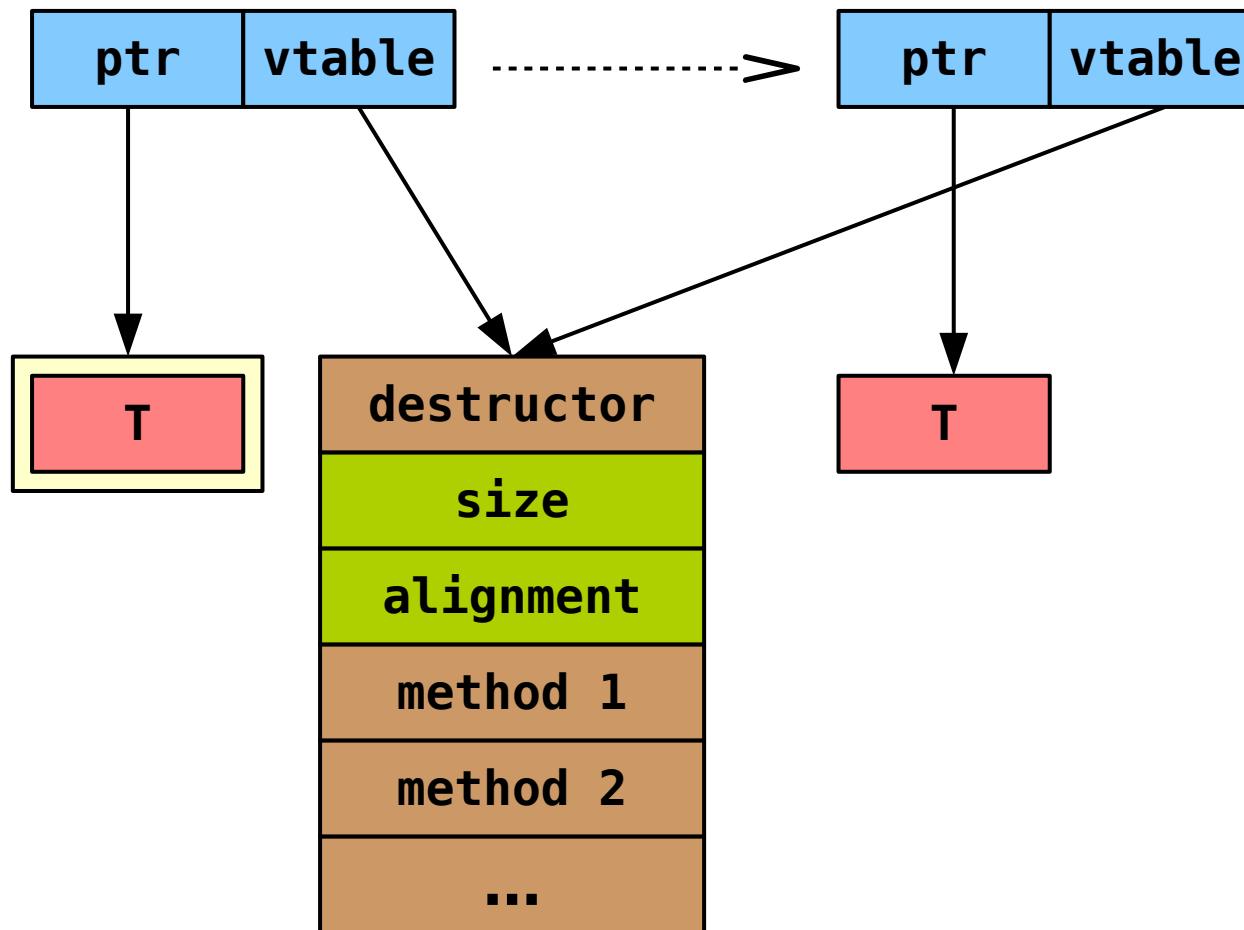
# Box and Traits

**Box<Trait>**



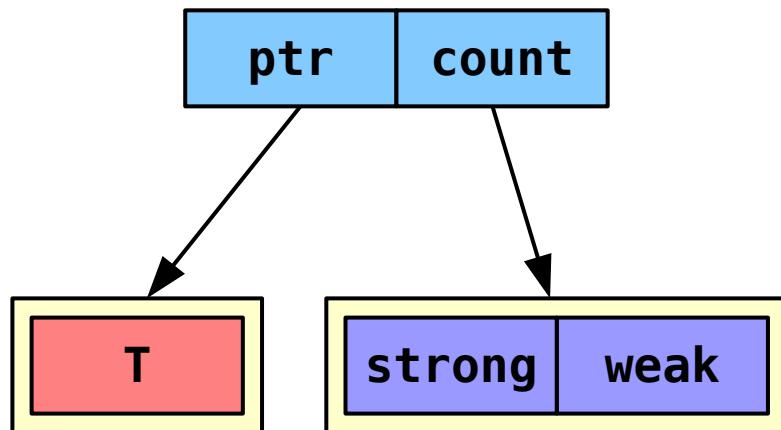
# Box and Traits

**Box<Trait>**      **&Trait**



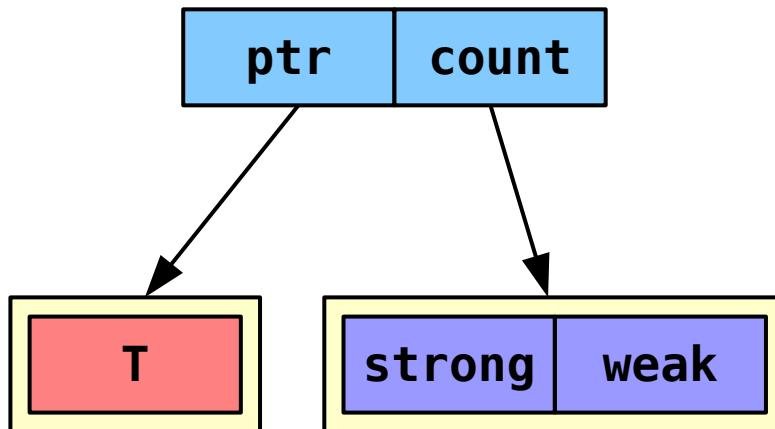


## **shared\_ptr<T>**

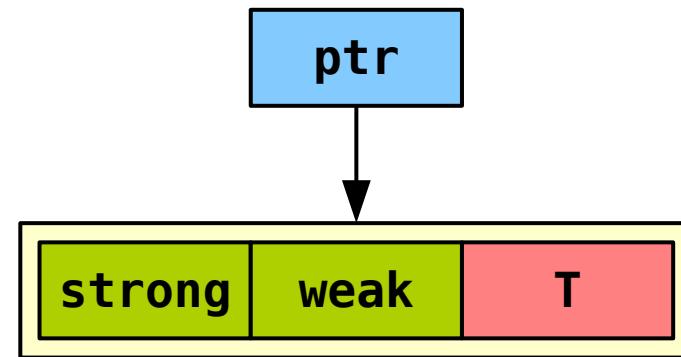


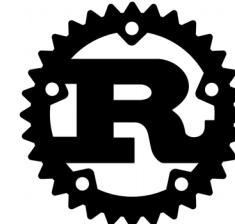


**shared\_ptr<T>**

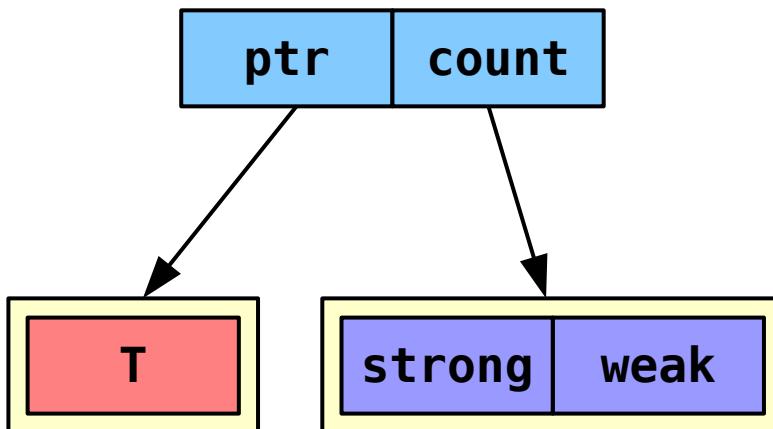


**Rc<T>**

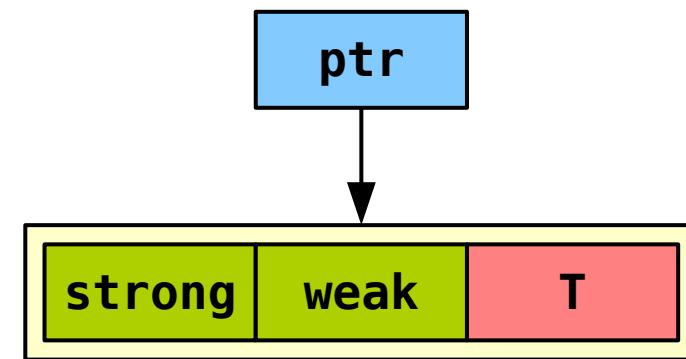




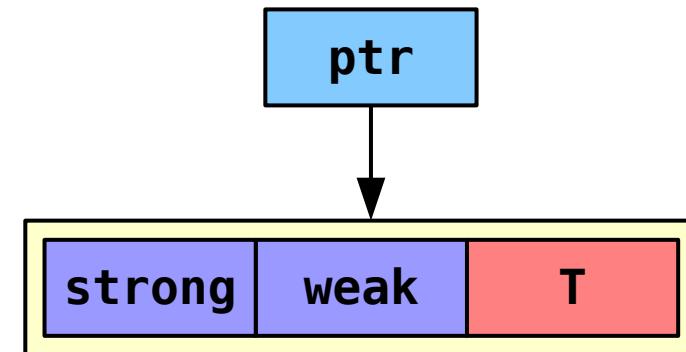
## shared\_ptr<T>

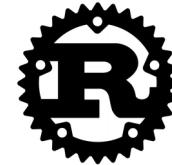


## Rc<T>

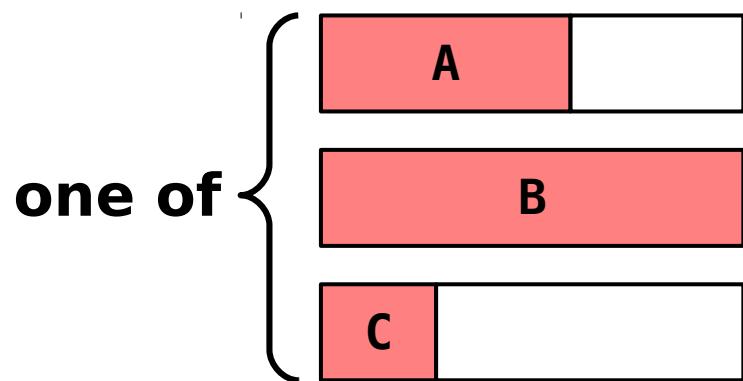


## Arc<T>



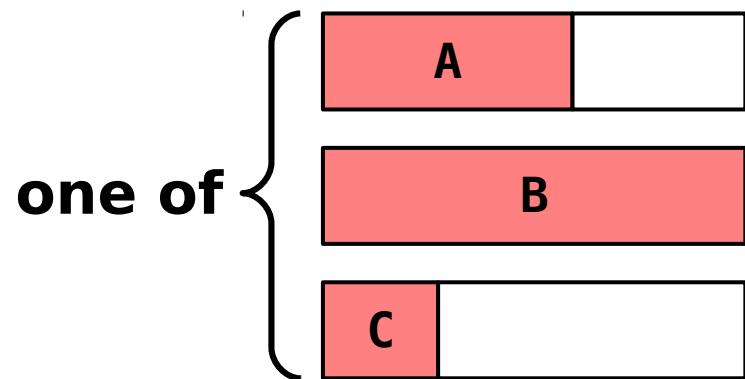


**union { A, B, C }**

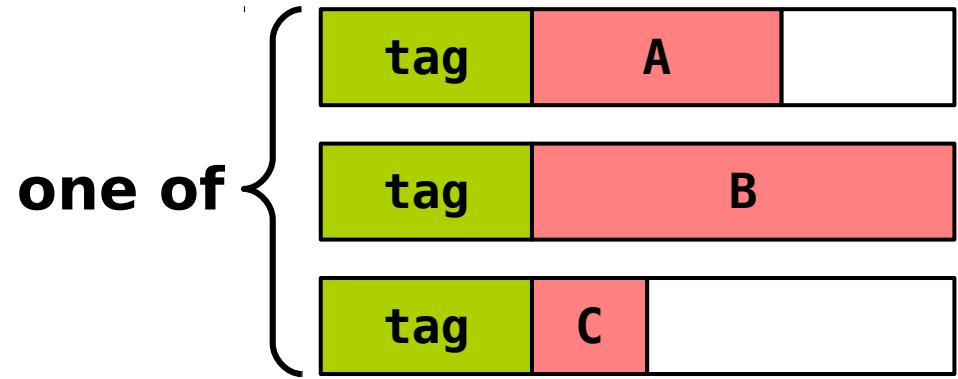




**union { A, B, C }**

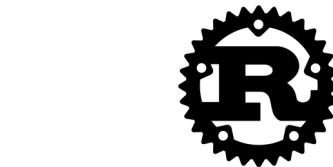
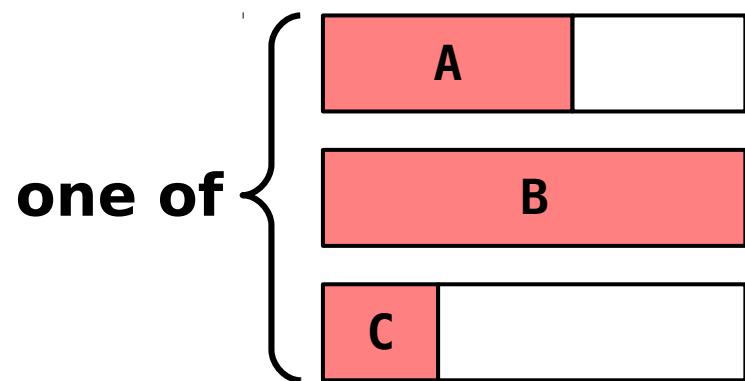


**enum { A, B, C }**

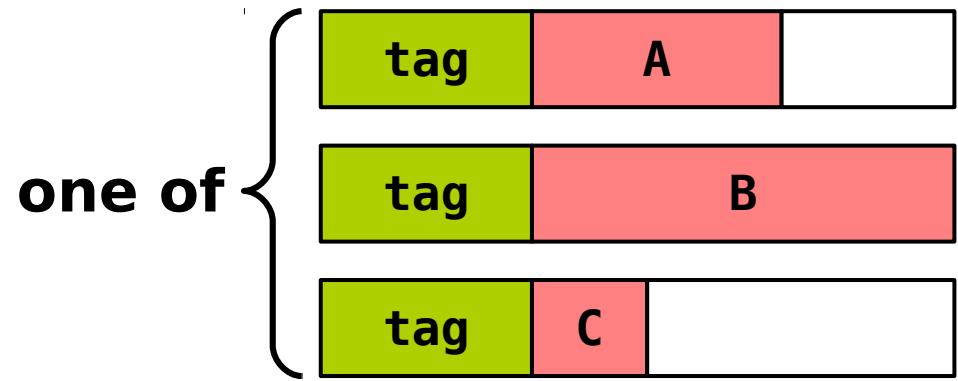




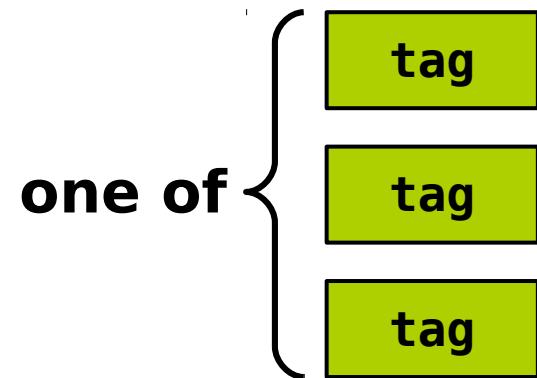
**union { A, B, C }**



**enum { A, B, C }**

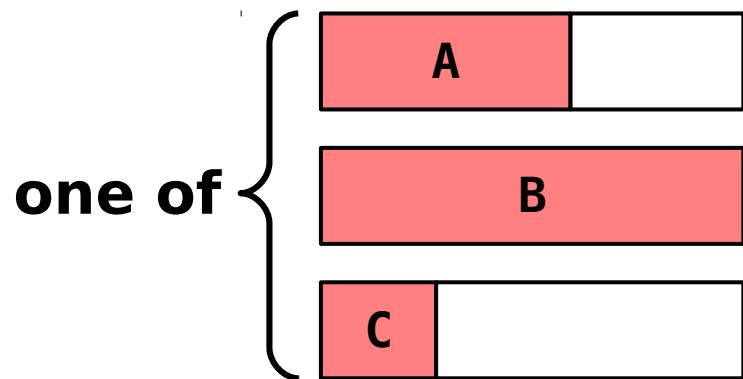


**enum class { A, B, C }**

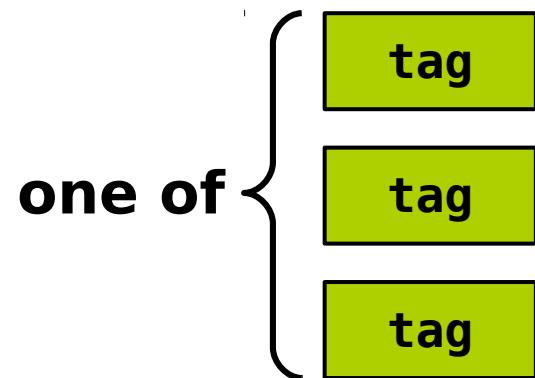




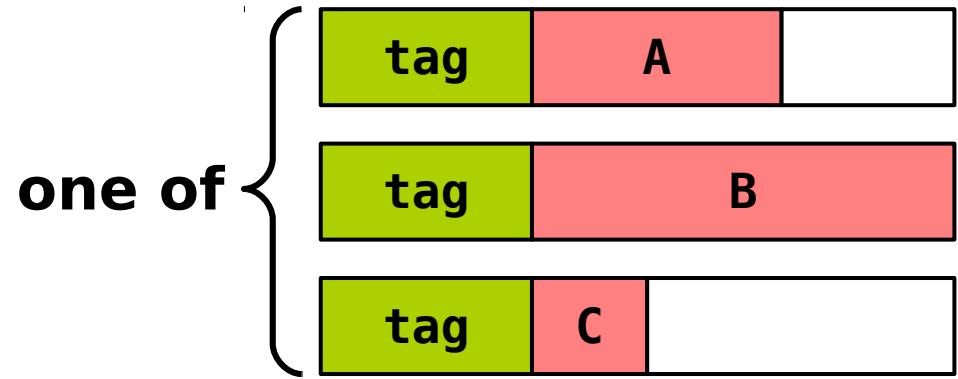
**union { A, B, C }**



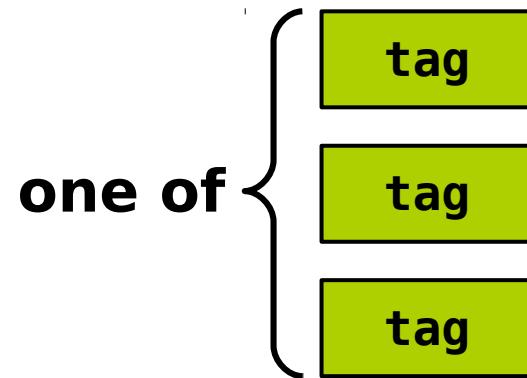
**enum class { A, B, C }**



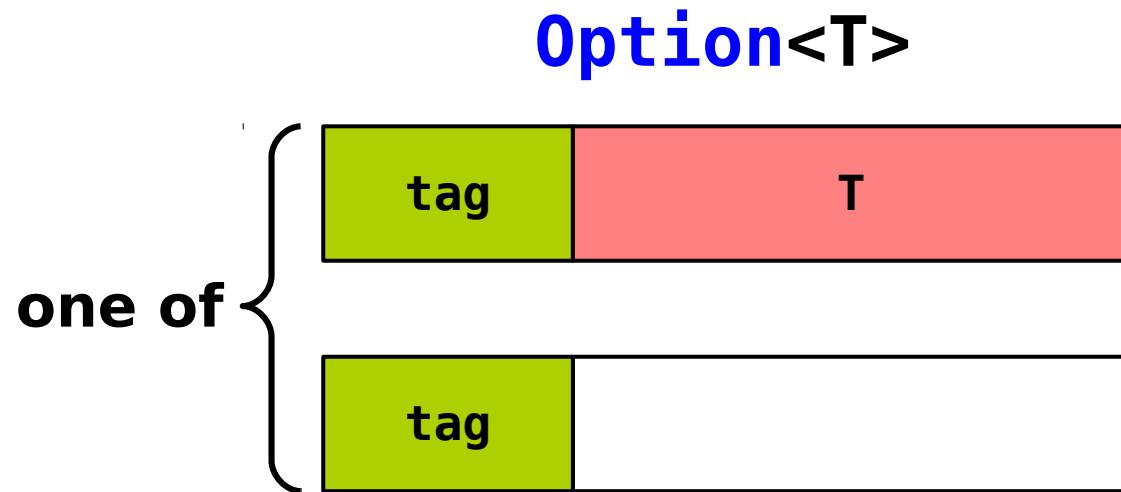
**enum { A, B, C }**



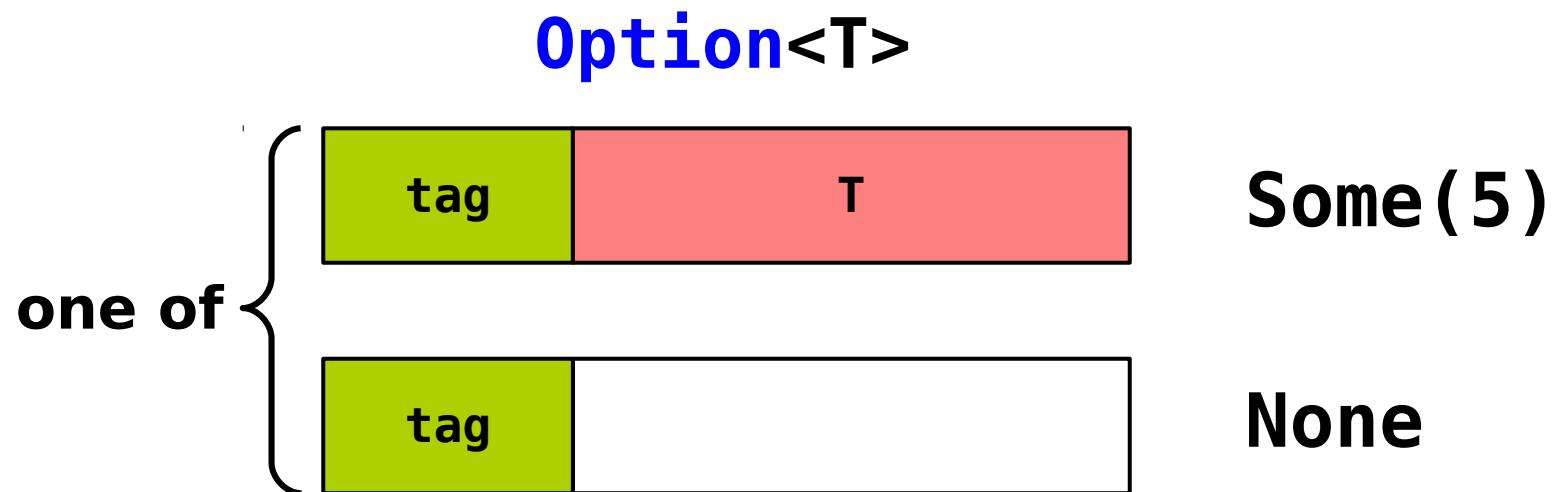
**enum { A, B, C }**



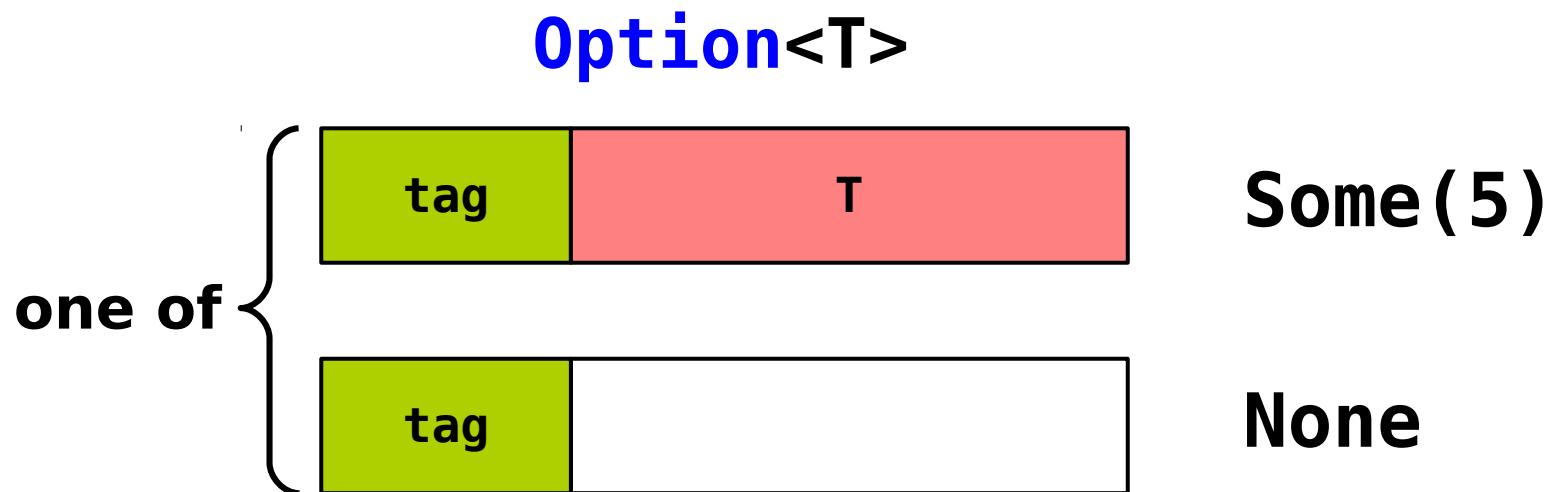
# Rust's Option



# Rust's Option



# Rust's Option

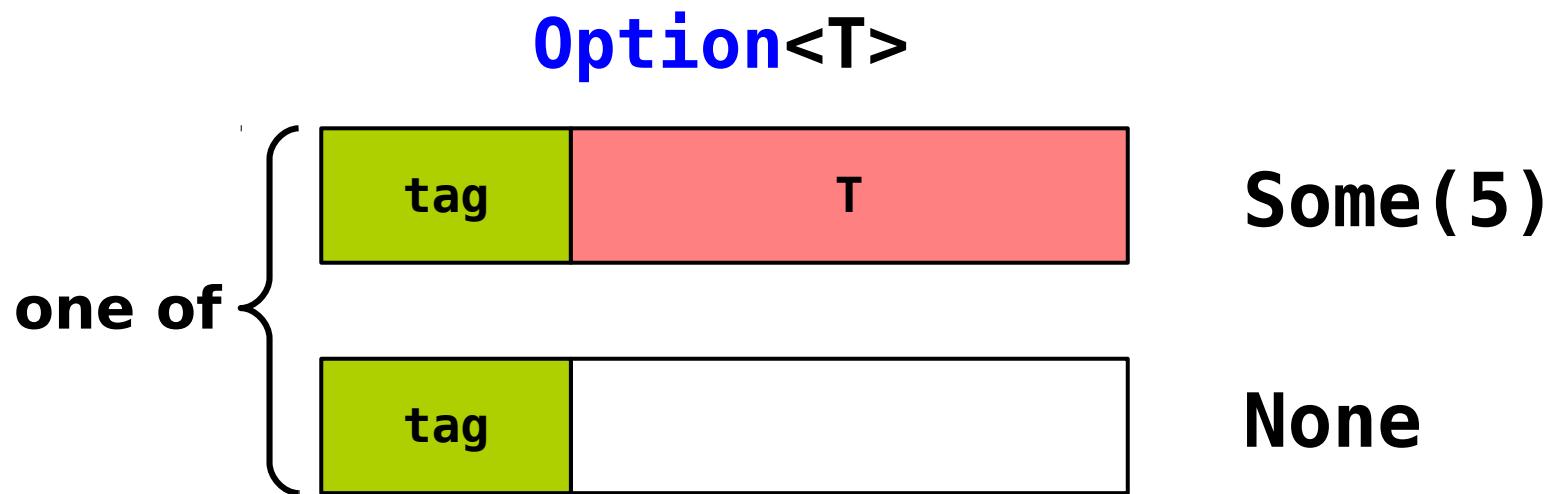


**Option<NonZero<T>>**



**Alle Ts die nicht null sein können.**

# Rust's Option



**Option<NonZero<T>>**



**Alle Ts die nicht null sein können.**

**&T**

**&mut T**

**Box<T>**

**Rc<T>**

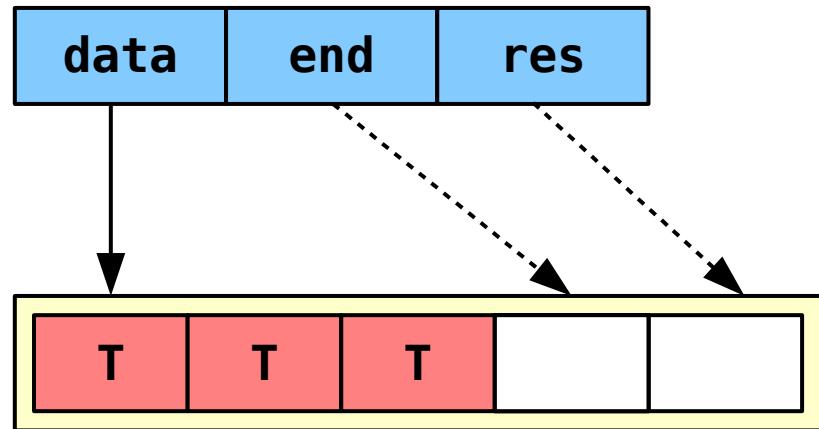
**Arc<T>**



**std::vector<T>**

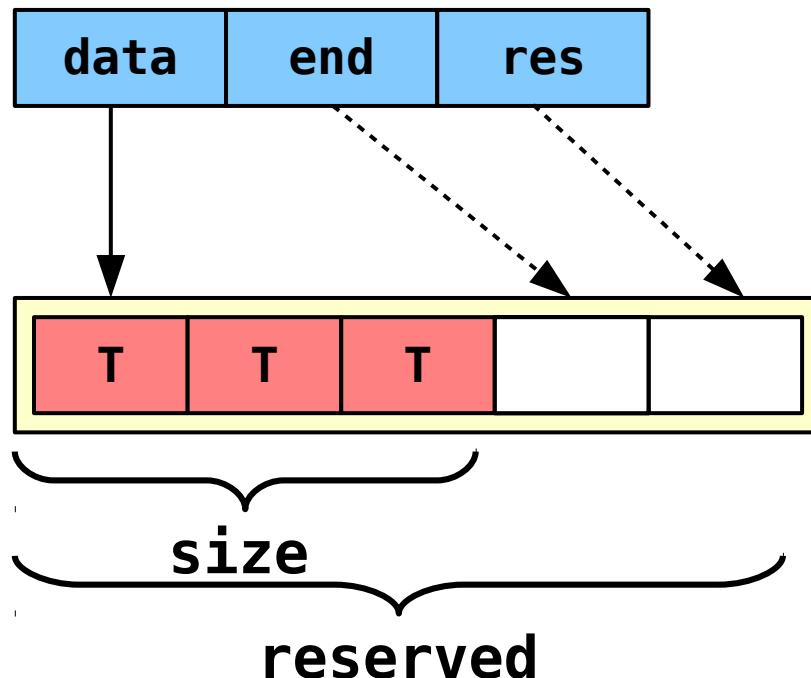


**std::vector<T>**



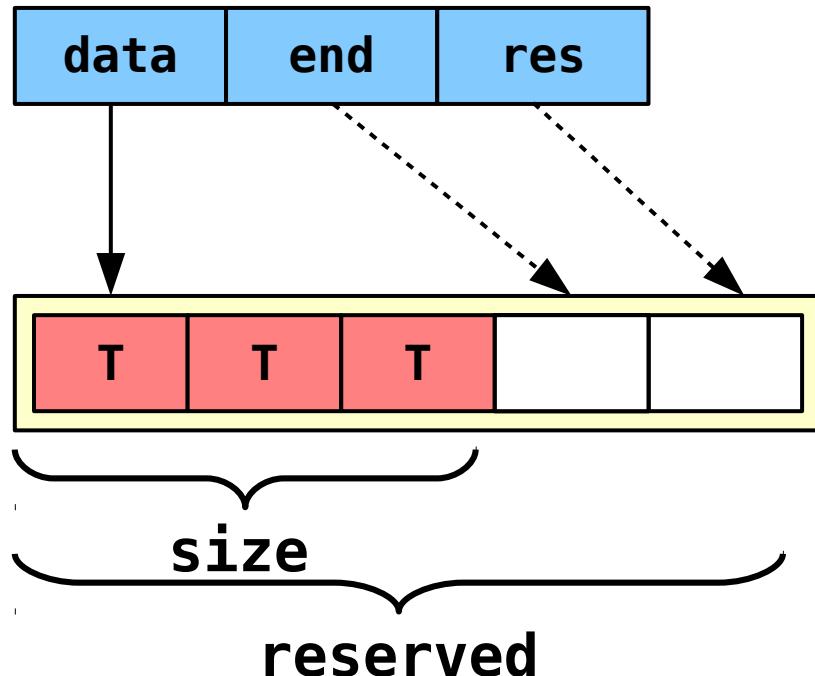


## `std::vector<T>`

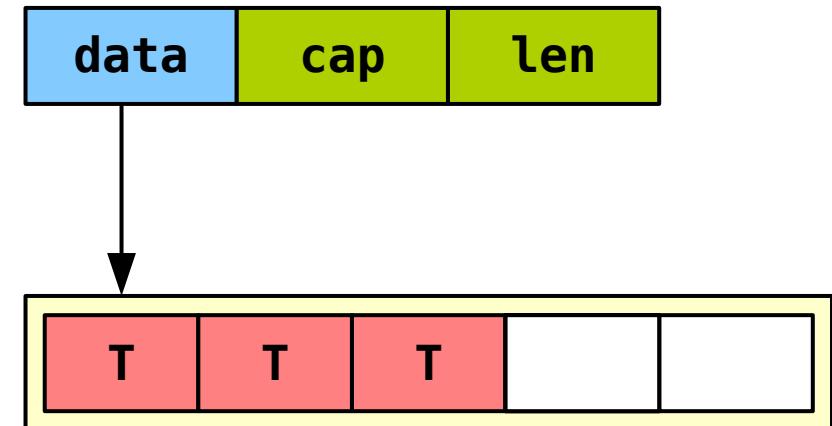




`std::vector<T>`

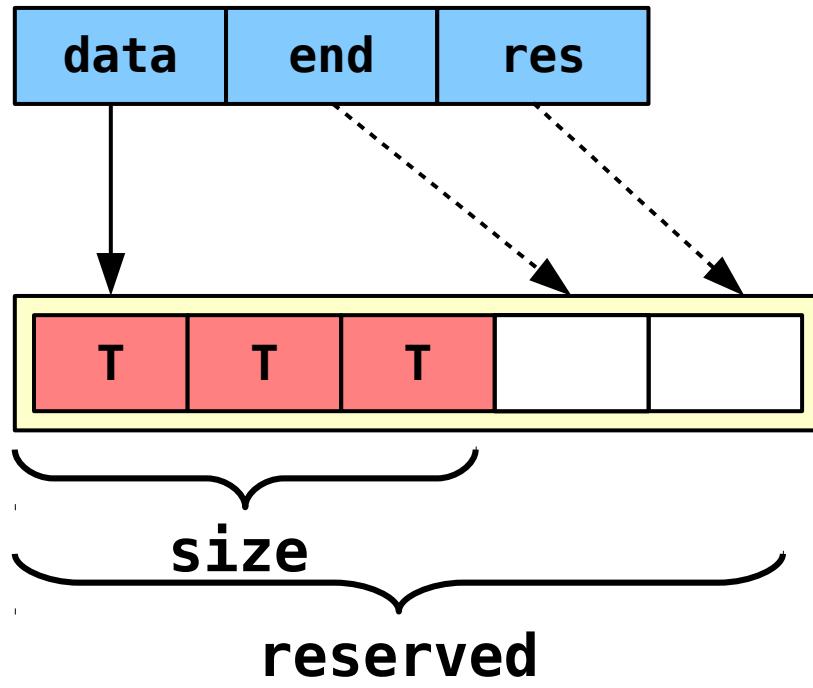


`Vec<T>`

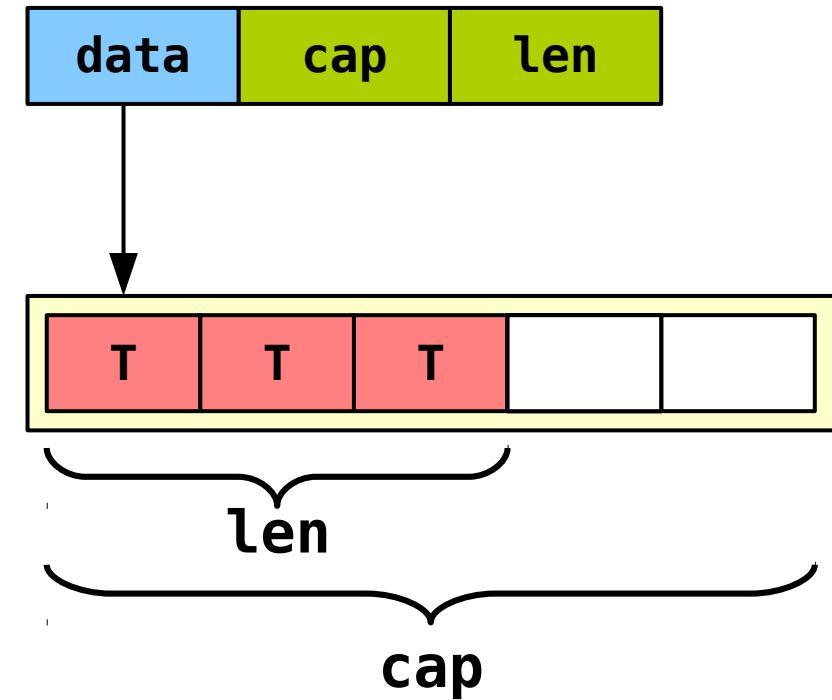




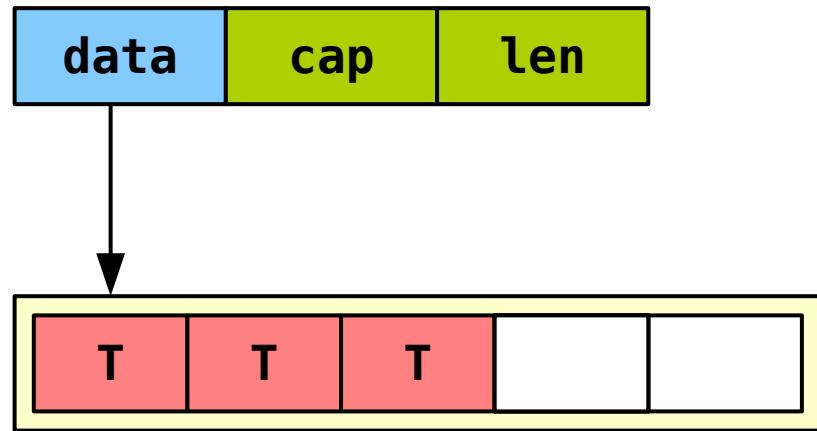
## `std::vector<T>`



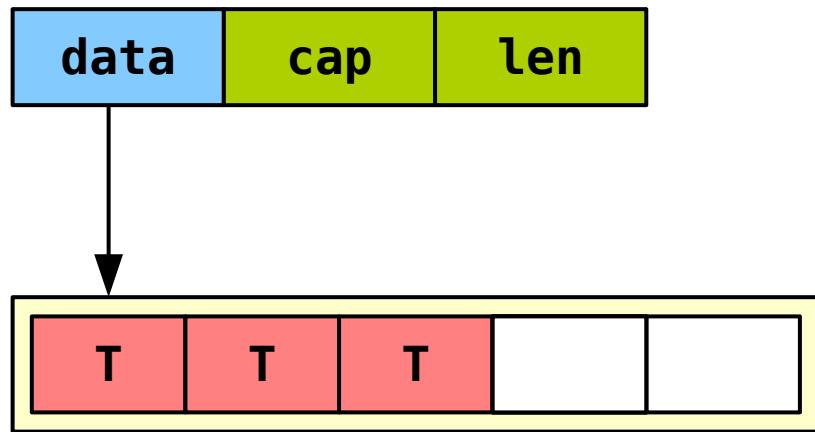
## `Vec<T>`



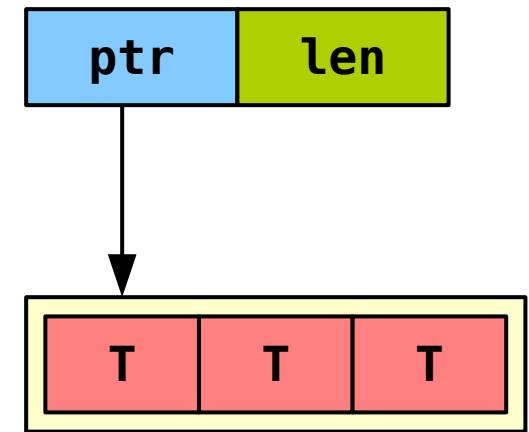
# **Vec<T>**



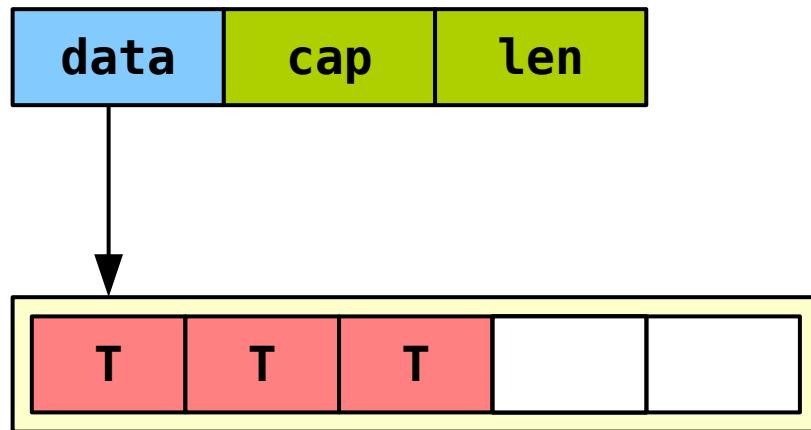
**Vec<T>**



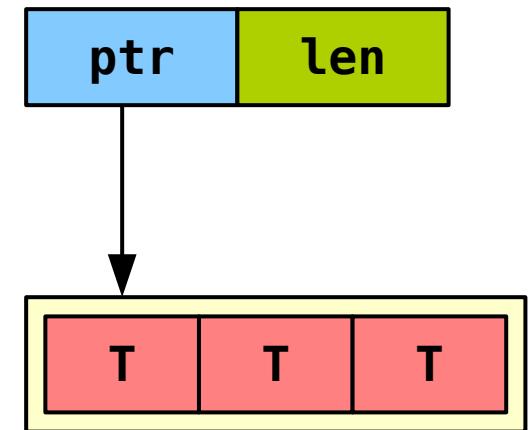
**Box<[T]>**



**Vec<T>**



**Box<[T]>**



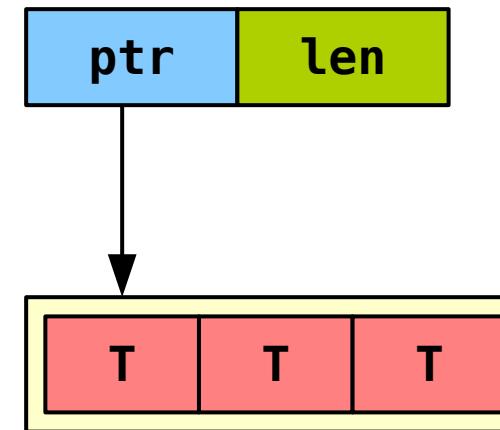
`into_boxed_slice`



**Vec<T>**



**Box<[T]>**



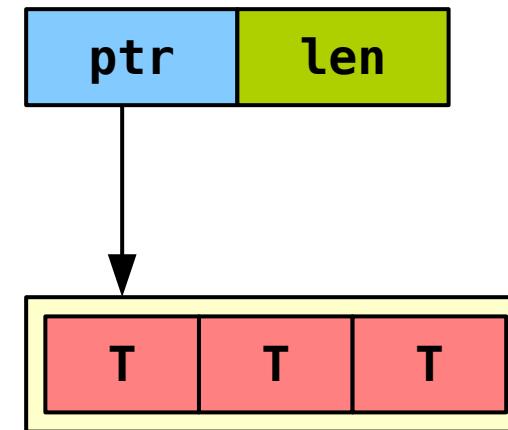
`into_boxed_slice`

`into_vec`

**Vec<T>**



**Box<[T]>**



**into\_boxed\_slice**

**into\_vec**

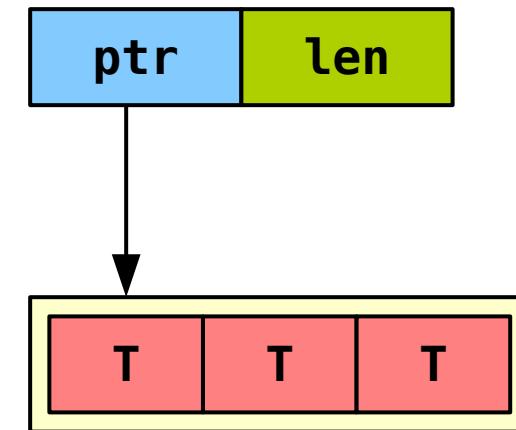
**&[T]**



**Vec<T>**



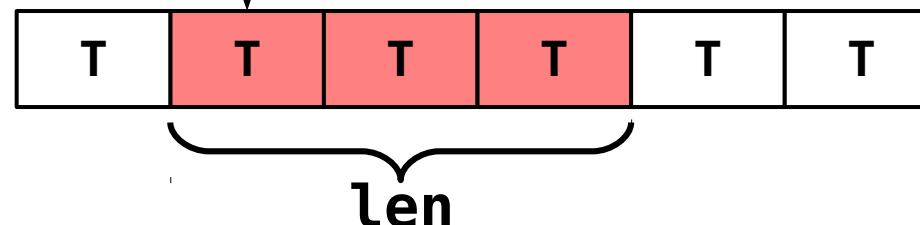
**Box<[T]>**



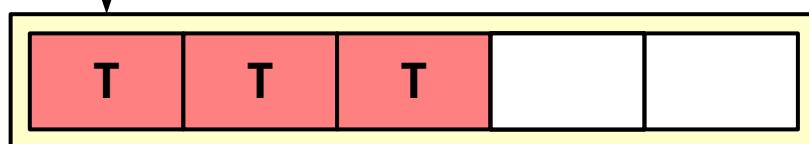
**into\_boxed\_slice**

**into\_vec**

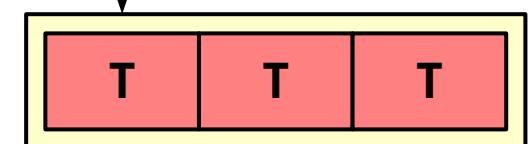
**&[T]**



**Vec<T>**



**Box<[T]>**



`into_boxed_slice`

`into_vec`

`deref`

`&[T]`

`deref`



`len`

**Vec<T>**



**Box<[T]>**



**into\_boxed\_slice**

**into\_vec**

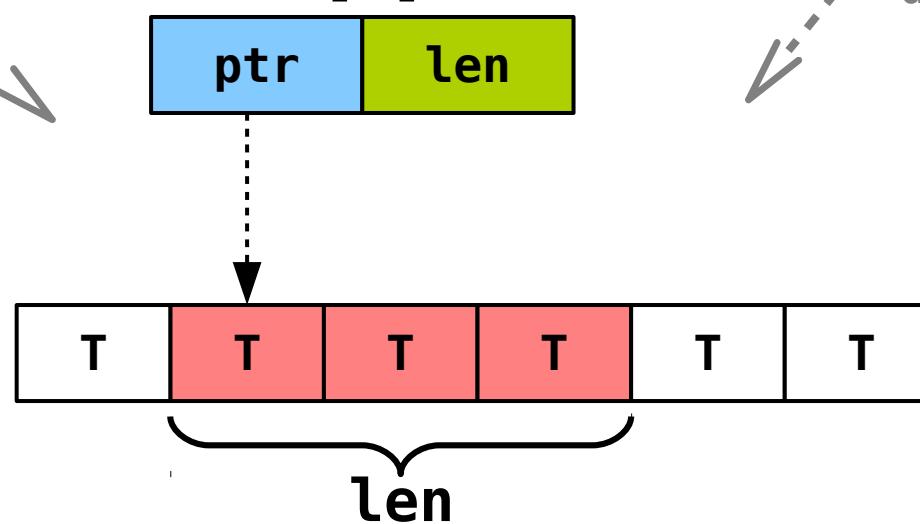
**to\_vec**

**deref**

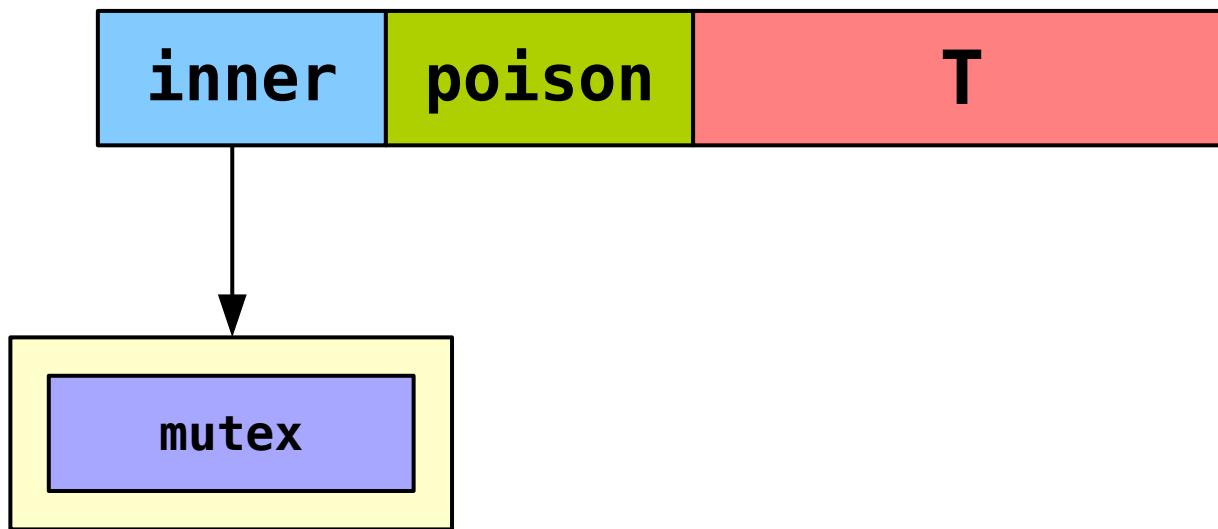
**&[T]**

**deref**

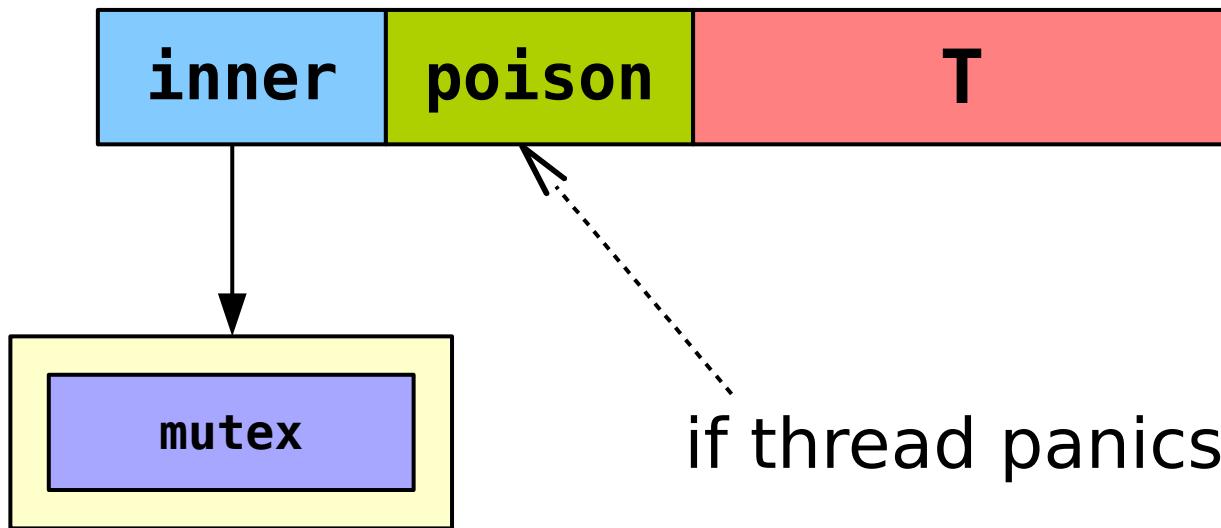
**len**



# **Mutex<T>**



# **Mutex<T>**



# Example: Mutex

```
let data = Arc::new(Mutex::new(0));
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));  
let (sndr, recv) = channel();
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));  
  
let (sndr, recv) = channel();  
  
for _ in 0..10 {  
    let (data, sndr) = (data.clone(), sndr.clone());
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));  
  
let (sndr, recv) = channel();  
  
for _ in 0..10 {  
    let (data, sndr) = (data.clone(), sndr.clone());  
    thread::spawn(move || {
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));

let (sndr, recv) = channel();

for _ in 0..10 {
    let (data, sndr) = (data.clone(), sndr.clone());
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));

let (sndr, recv) = channel();

for _ in 0..10 {
    let (data, sndr) = (data.clone(), sndr.clone());
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        *data += 1;
    });
}
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));

let (sndr, recv) = channel();

for _ in 0..10 {
    let (data, sndr) = (data.clone(), sndr.clone());
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        *data += 1;
        if *data == 10 {
            sndr.send(()).unwrap();
        }
    });
}
```

# Example: Mutex

```
let data = Arc::new(Mutex::new(0));

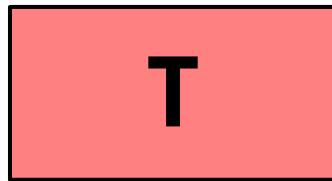
let (sndr, recv) = channel();

for _ in 0..10 {
    let (data, sndr) = (data.clone(), sndr.clone());
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        *data += 1;
        if *data == 10 {
            sndr.send(()).unwrap();
        }
    });
}

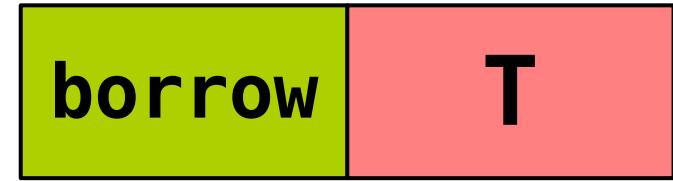
recv.recv().unwrap();
```

# Cell & RefCell

**Cell<T>**

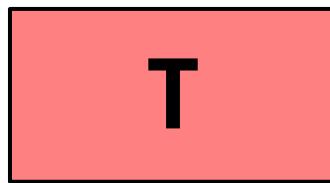


**RefCell<T>**



# Cell & RefCell

**Cell<T>**



Replacement

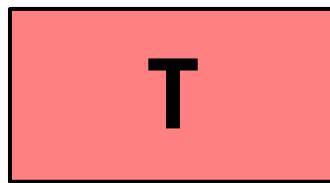
**RefCell<T>**



Runtime tracking

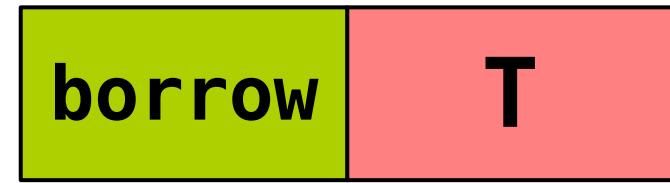
# Cell & RefCell

**Cell<T>**



Replacement  
**value**-like

**RefCell<T>**



Runtime tracking  
**reference**-like

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {
```



# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;  
}
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;   
}
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;   
}
```



```
struct Foo {  
    a: Cell<i32>,  
    b: f32  
}
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;   
}
```



```
struct Foo {  
    a: Cell<i32>,  
    b: f32  
}  
  
fn main() {  
    let foo = Foo{  
        a: Cell::new(42),  
        b: 13.37 };
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;   
}
```



```
struct Foo {  
    a: Cell<i32>,  
    b: f32  
}  
  
fn main() {  
    let foo = Foo{  
        a: Cell::new(42),  
        b: 13.37 };  
    foo.a.set(5);
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;    
}
```



```
struct Foo {  
    a: Cell<i32>,  
    b: f32  
}  
  
fn main() {  
    let foo = Foo{  
        a: Cell::new(42),  
        b: 13.37 };  
    foo.a.set(5);  
    foo.b = 7.7;  
}
```

# Example: mutable & Cell



```
struct Foo {  
    mutable int a;  
    float b;  
};  
  
int main() {  
    const auto foo  
        = Foo{42, 1337};  
    foo.a = 5;  
    foo.b = 7.7;   
}
```



```
struct Foo {  
    a: Cell<i32>,  
    b: f32  
}  
  
fn main() {  
    let foo = Foo{  
        a: Cell::new(42),  
        b: 13.37 };  
    foo.a.set(5);   
    foo.b = 7.7;   
}
```

# C++ & Rust

# Variable Declarations

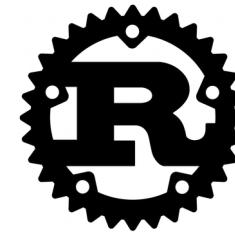


---

# Variable Declarations



```
int a = 42;
```



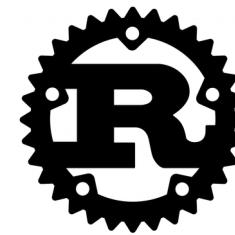
```
let mut a: i32 = 42;
```

# Variable Declarations



```
int a = 42;
```

```
int const a = 42;
```



```
let mut a: i32 = 42;
```

```
let a: i32 = 42;
```

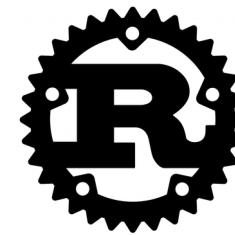
# Variable Declarations



```
int a = 42;
```

```
int const a = 42;
```

```
auto a = 42;
```



```
let mut a: i32 = 42;
```

```
let a: i32 = 42;
```

```
let mut a = 42;
```

# Variable Declarations



```
int a = 42;
```

```
int const a = 42;
```

```
auto a = 42;
```

```
auto const a = 42;
```



```
let mut a: i32 = 42;
```

```
let a: i32 = 42;
```

```
let mut a = 42;
```

```
let a = 42;
```

# Primitive Datatypes



# Primitive Datatypes



bool



bool

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t



bool

iN

uN

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t

float, double



bool

iN

uN

f32, f64

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t

float, double

std::array<T, N>



bool

iN

uN

f32, f64

[T; N]

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t

float, double

std::array<T, N>

-



bool

iN

uN

f32, f64

[T; N]

[T]

# Primitive Datatypes



bool  
std::intN\_t  
std::uintN\_t  
float, double  
std::array<T, N>  
-  
gsl::span<T>



bool  
iN  
uN  
f32, f64  
[T; N]  
[T]  
&[T]

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t

float, double

std::array<T, N>

-

gsl::span<T>

std::tuple<A, B, ...>



bool

iN

uN

f32, f64

[T; N]

[T]

&[T]

(A, B, ...)

# Primitive Datatypes



bool

std::intN\_t

std::uintN\_t

float, double

std::array<T, N>

-

gsl::span<T>

std::tuple<A, B, ...>

std::string\_view



bool

iN

uN

f32, f64

[T; N]

[T]

&[T]

(A, B, ...)

&str

# Dynamic Polymorphism

```
struct Button {
```

# Dynamic Polymorphism

```
struct Button {
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {  
    fn add_listener(&mut self, l: Box<Fn()>) {
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {  
    fn add_listener(&mut self, l: Box<Fn()>) {
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {  
    fn add_listener(&mut self, l: Box<Fn()>) {
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {  
    fn add_listener(&mut self, l: Box<Fn()>) {  
        self.listeners.push(l)  
    }  
}
```

# Dynamic Polymorphism

```
struct Button {  
    listeners: Vec<Box<Fn()>>  
}  
  
impl Button {  
    fn add_listener(&mut self, l: Box<Fn()>) {  
        self.listeners.push(l)  
    }  
  
    fn click(&self) {
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}

fn main() {
    let mut b = Button{ listeners: Vec::new() };
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}

fn main() {
    let mut b = Button{ listeners: Vec::new() };
    b.add_listener(Box::new(|| { println!("*clicked*") }));
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}

fn main() {
    let mut b = Button{ listeners: Vec::new() };
    b.add_listener(Box::new(|| { println!("*clicked*") }));
}
```

# Dynamic Polymorphism

```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}

fn main() {
    let mut b = Button{ listeners: Vec::new() };
    b.add_listener(Box::new(|| { println!("*clicked*") }));
}
```

# Dynamic Polymorphism

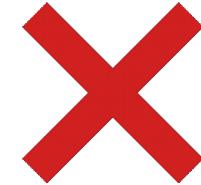
```
struct Button {
    listeners: Vec<Box<Fn()>>
}

impl Button {
    fn add_listener(&mut self, l: Box<Fn()>) {
        self.listeners.push(l)
    }

    fn click(&self) {
        for listener in &self.listeners {
            listener()
        }
    }
}

fn main() {
    let mut b = Button{ listeners: Vec::new() };
    b.add_listener(Box::new(|| { println!("*clicked*") }));
    b.click();
}
```

# Marker Traits



**trait Send { }**

Safe to send between threads.

---

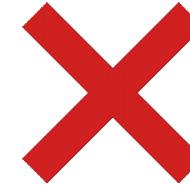
# Marker Traits

**trait Send { }**

Safe to send between threads.



i32



# Marker Traits

**trait Send { }**

Safe to send between threads.



i32

String



# Marker Traits

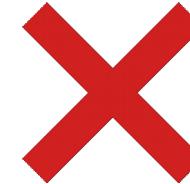
**trait Send { }**

Safe to send between threads.



i32

String



Rc<String>

# Marker Traits

**trait Send { }**

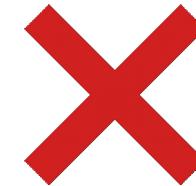
Safe to send between threads.



i32

String

Arc<String>

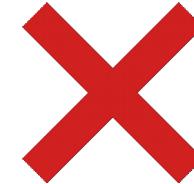


Rc<String>

# Marker Traits

**trait Send { }**

Safe to send between threads.



i32

String

Arc<String>

Rc<String>

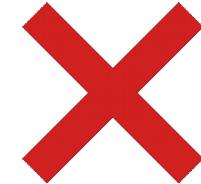
**trait Copy { }**

Safe to `memcpy`.

# Marker Traits

**trait Send { }**

Safe to send between threads.



i32

String

Arc<String>

Rc<String>

**trait Copy { }**

Safe to `memcpy`.

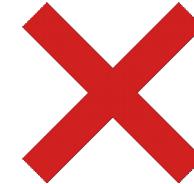
i32

[f64; 4]

# Marker Traits

**trait Send { }**

Safe to send between threads.



i32

String

Arc<String>

Rc<String>

**trait Copy { }**

Safe to `memcpy`.

i32

[f64; 4]

String

Rc<String>

# Marker Traits

| <b>trait Send { }</b><br>Safe to send between threads. |  | Rc<String>           |
|--|---|----------------------|
| <b>trait Copy { }</b><br>Safe to <code>memcpy</code> . | i32<br>String<br>Arc<String>  | i32<br>[f64; 4]      |
|  |   | String<br>Rc<String> |

**trait Sync { }**

Safe to share between threads as &T.

# Common Cargo Commands

|         |                            |
|---------|----------------------------|
| build   | Compiles the project       |
| doc     | Generates documentation    |
| new     | Creates new project        |
| run     | Runs binary                |
| test    | Runs tests                 |
| bench   | Runs benchmarks            |
| update  | Updates dependencies       |
| search  | Search registry for crates |
| publish | Upload crate to registry   |
| install | Install a Rust binary      |

The End