Building awesome tools in Rust on Linux

Kent Marete

@kentccs

maretekent

What is Rust

History

- **Graydon Hoare** dev 2006
- **Mozilla** starts sponsoring Rust in 2009
- Version 1.0.0 (2015-)
- **> 19,567** crates – libraries & **584,987,337** Downloads
- **>2,211** contributors on Github. Compiler
- **Big areas**: game dev, operating systems, web development, block chain

What is Rust?

- **Rust** is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety. **(safety, concurrency, and speed)**
- It’s a programming language founded by Mozilla research.

Javascript

Rust

D

C++

Java

Perl

Scala

Python

Haskell

Ruby

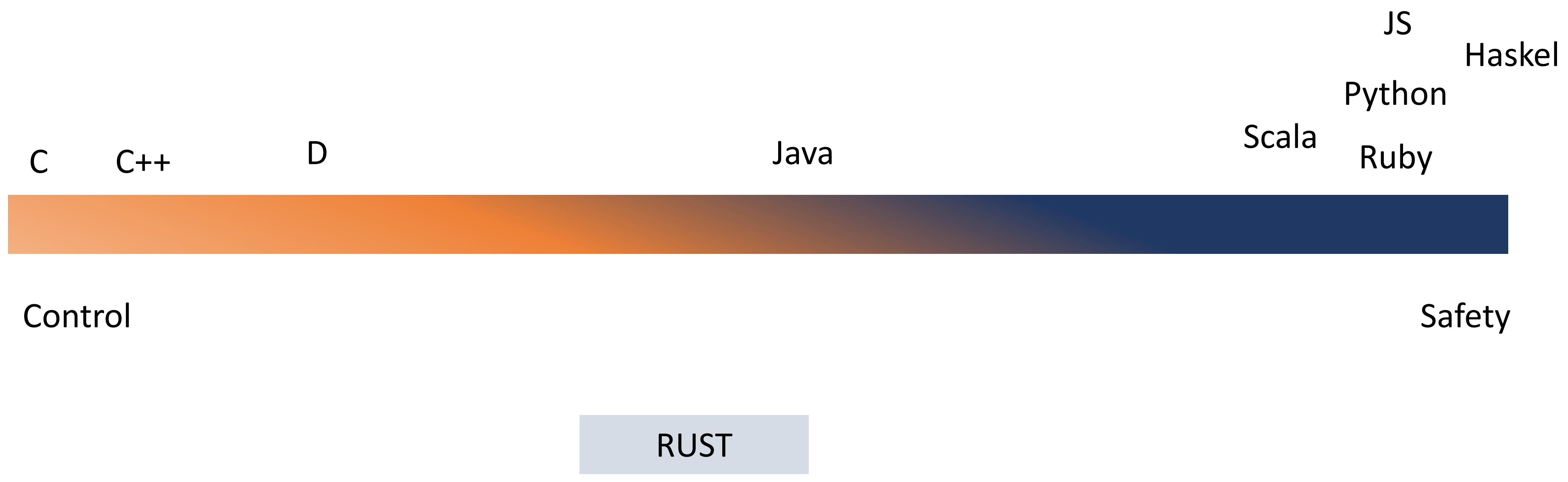
C#

C

PHP

What Rust has to offer

- We can organize these languages in a linear spectrum



Why Rust?

- ❑ Speed
- ❑ Rust has great functionality – raw binary data
- ❑ Reliability – rare to break, once it runs
- ❑ Control
- ❑ limited resources
- ❑ Concurrency
- ❑ Type inference - let b= 5u8; let a=5;
- ❑ High level abstraction – minimizes code

How Rust looks?

How Rust looks?

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

```
fn plus_one(a: i32) -> i32 {
    a + 1 //no ; means an expression, return a+1
}
```

```
// ⭐ Function pointers, Usage as a Data Type
let b = plus_one;
let c = b(5); //6
```

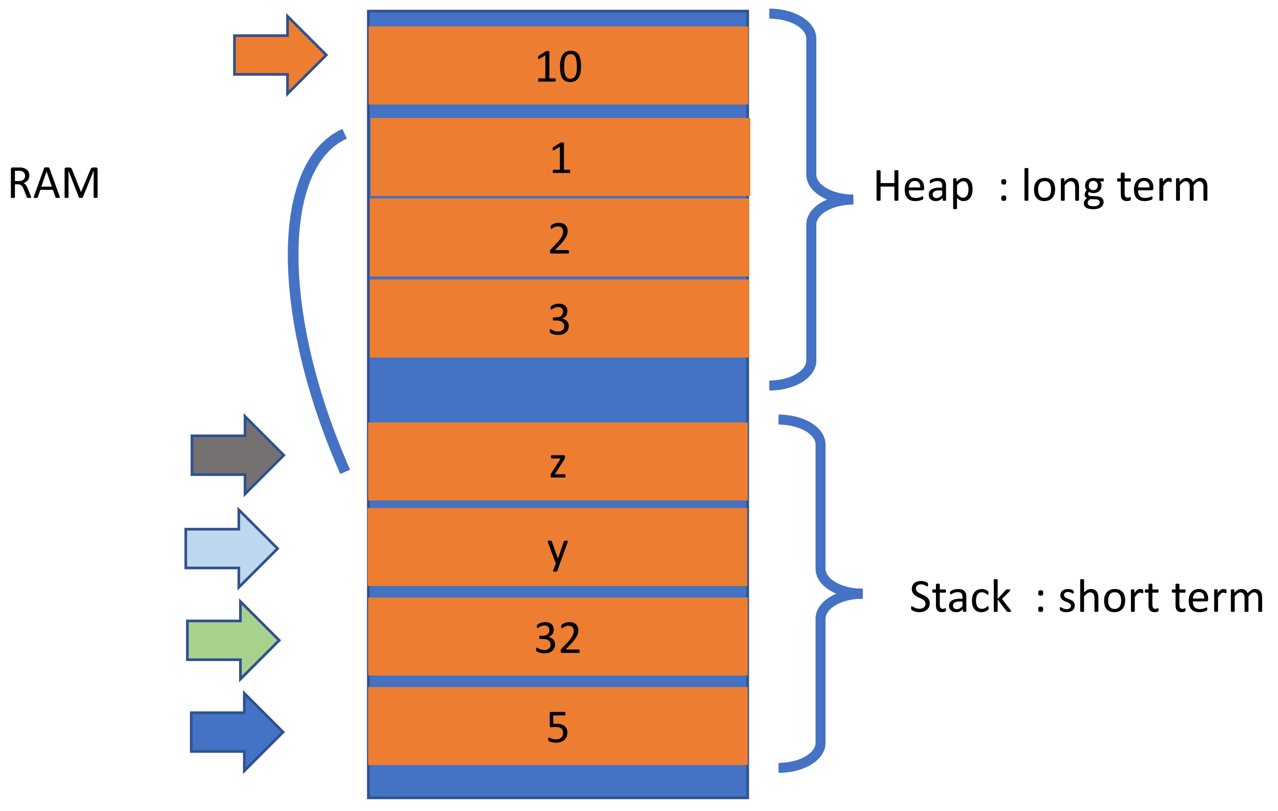
Speed?

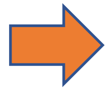
- No Garbage Collection - Rust uses the ***Resource Acquisition Is Initialization*** (RAII) technique - object lifetime
- LLVM - is a compiler infrastructure
- Zero Cost Abstractions - *What you don’t use, you don’t pay for*
- Minimal Runtime – No GC, can compile without stdlib

What is Control?

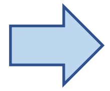
Rust gives the developer fine control over the use of memory

Control? - Stack and heap

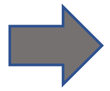



Use box construct

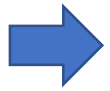
```
let y = Box::new(10);
```



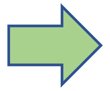
```
println!("y = {}", *y);
```



```
let z = vec![1,2,3];
```



```
let x = 5;    //i32

fn inc(x:i32){x+1}
```

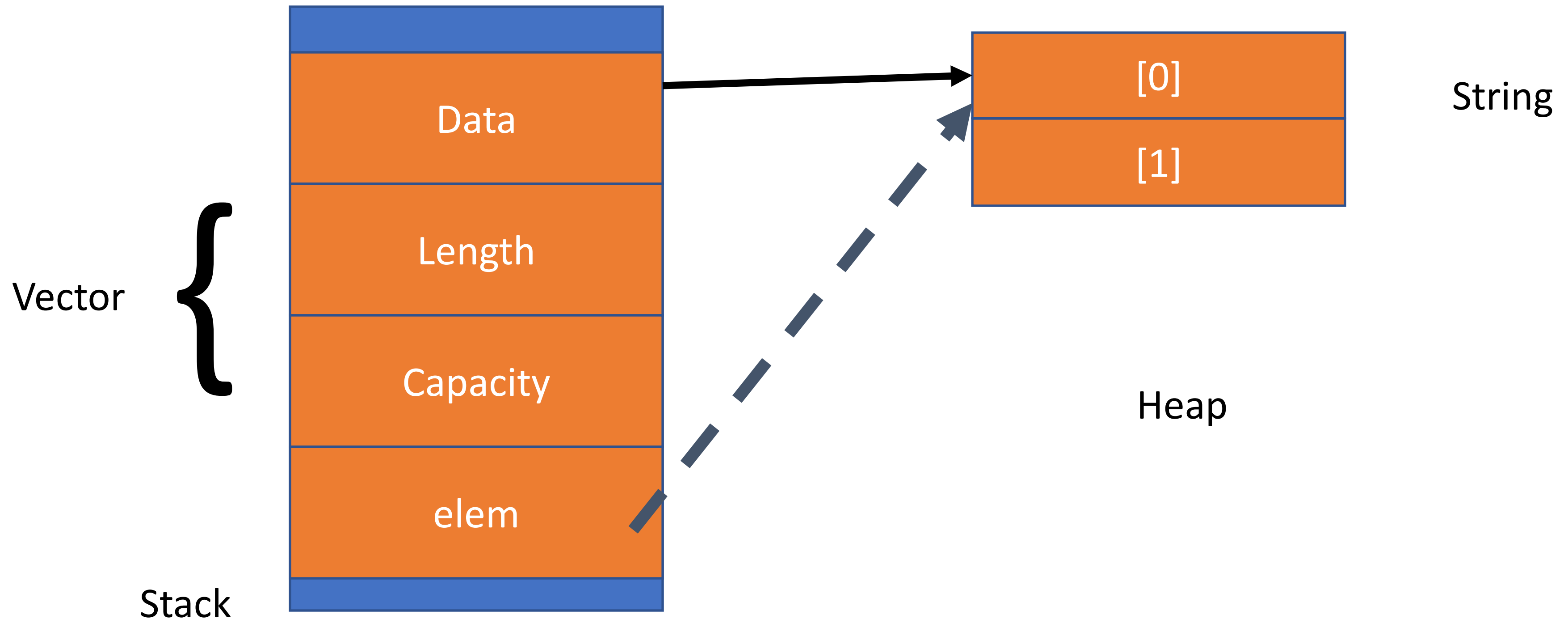


```
inc(32);
```

Playground link

What is Safety?

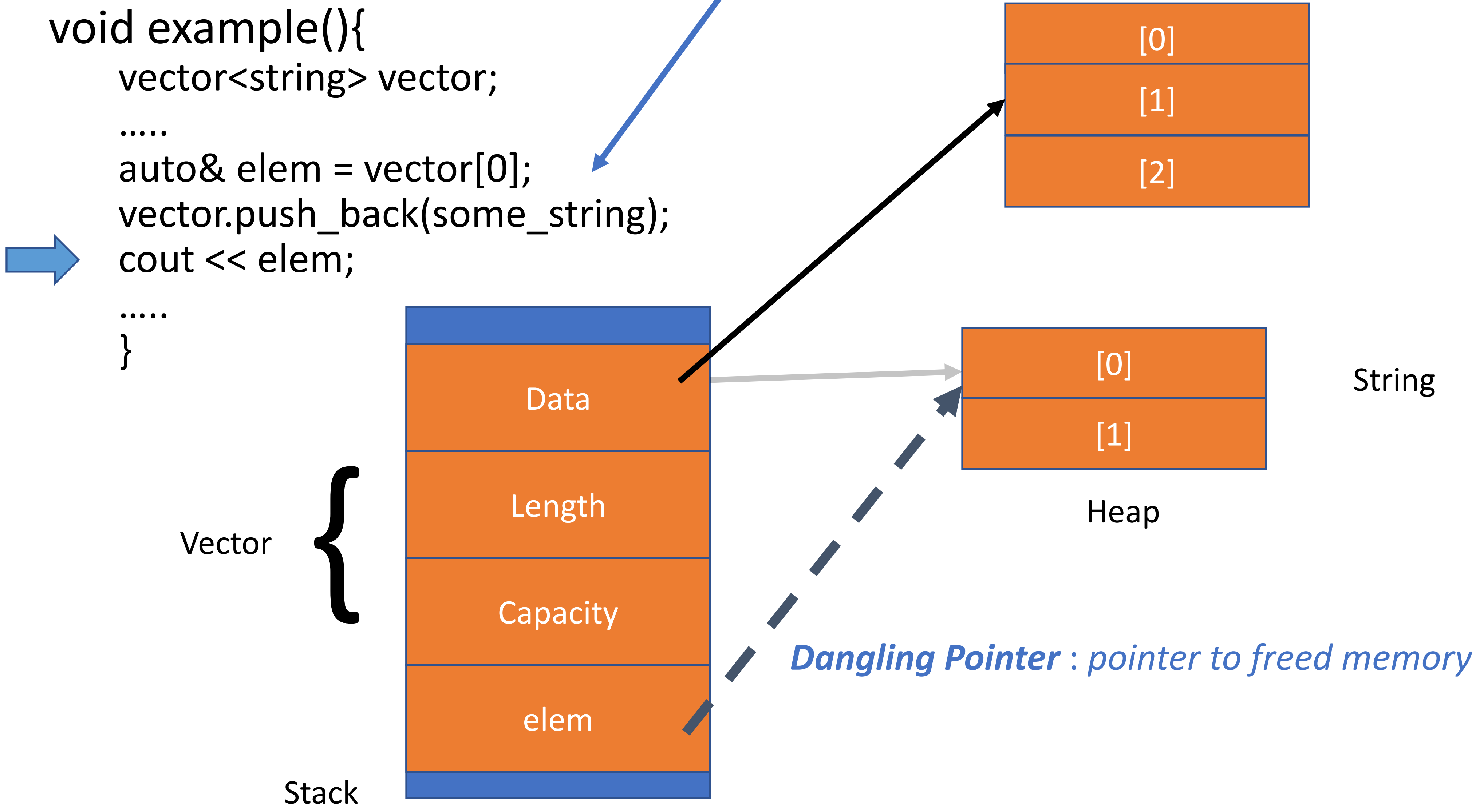
```
void example(){
	vector<string> vector;
	.....
	auto& elem = vector[0];
	vector.push_back(some_string);
	.....
	}
```



What is Safety?

```
void example(){
    vector<string> vector;
    .....
    auto& elem = vector[0];
    vector.push_back(some_string);
    cout << elem;
    .....
}
```

***Mutation**: vector freed old contents*

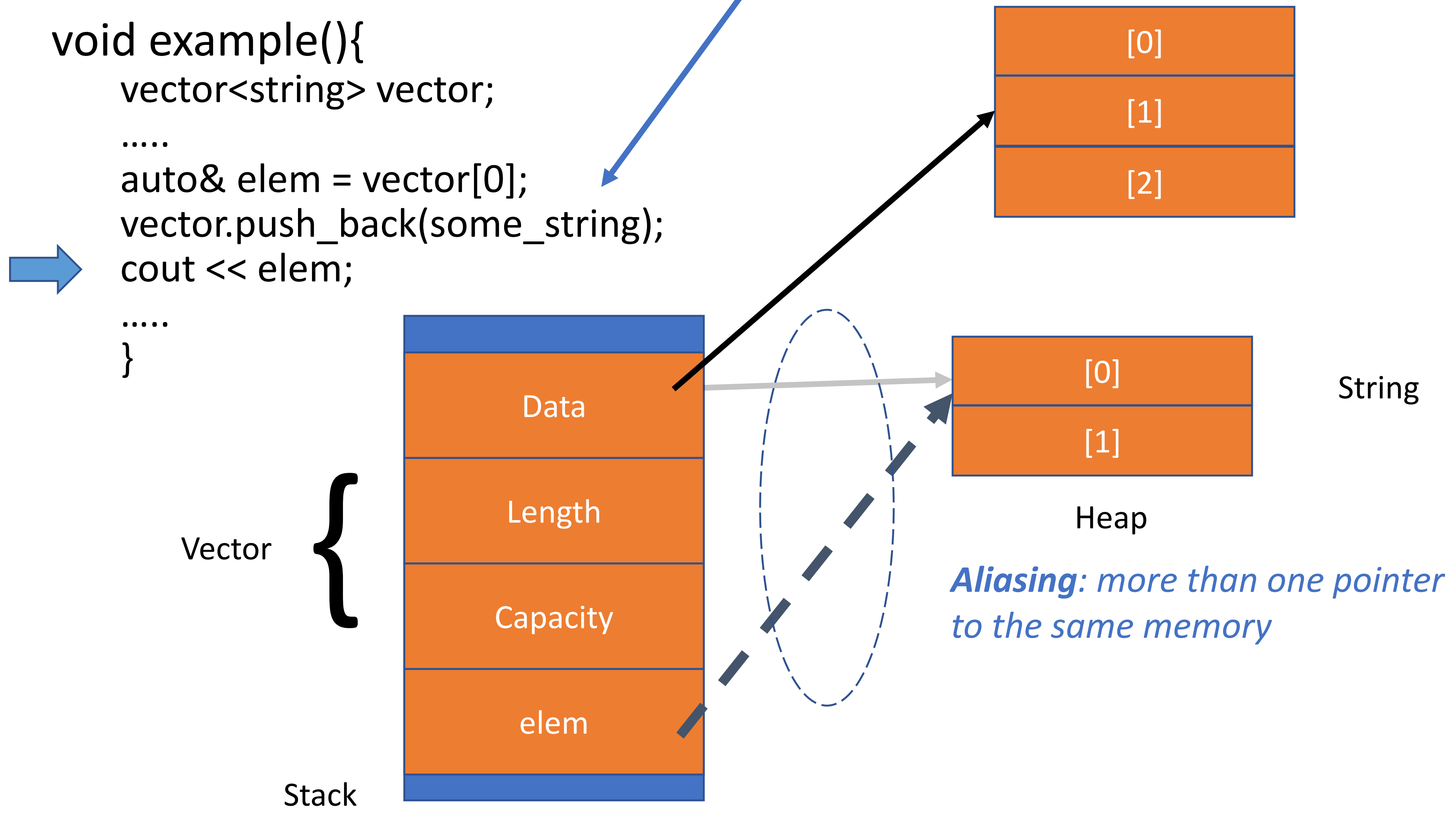


***Dangling Pointer** : pointer to freed memory*

What is Safety?

```
void example(){
      vector<string> vector;
      .....
      auto& elem = vector[0];
      vector.push_back(some_string);
      cout << elem;
      .....
      }
```

__Mutation__: vector freed old contents



__Aliasing__: more than one pointer to the same memory

Garbage Collection?

Downside:

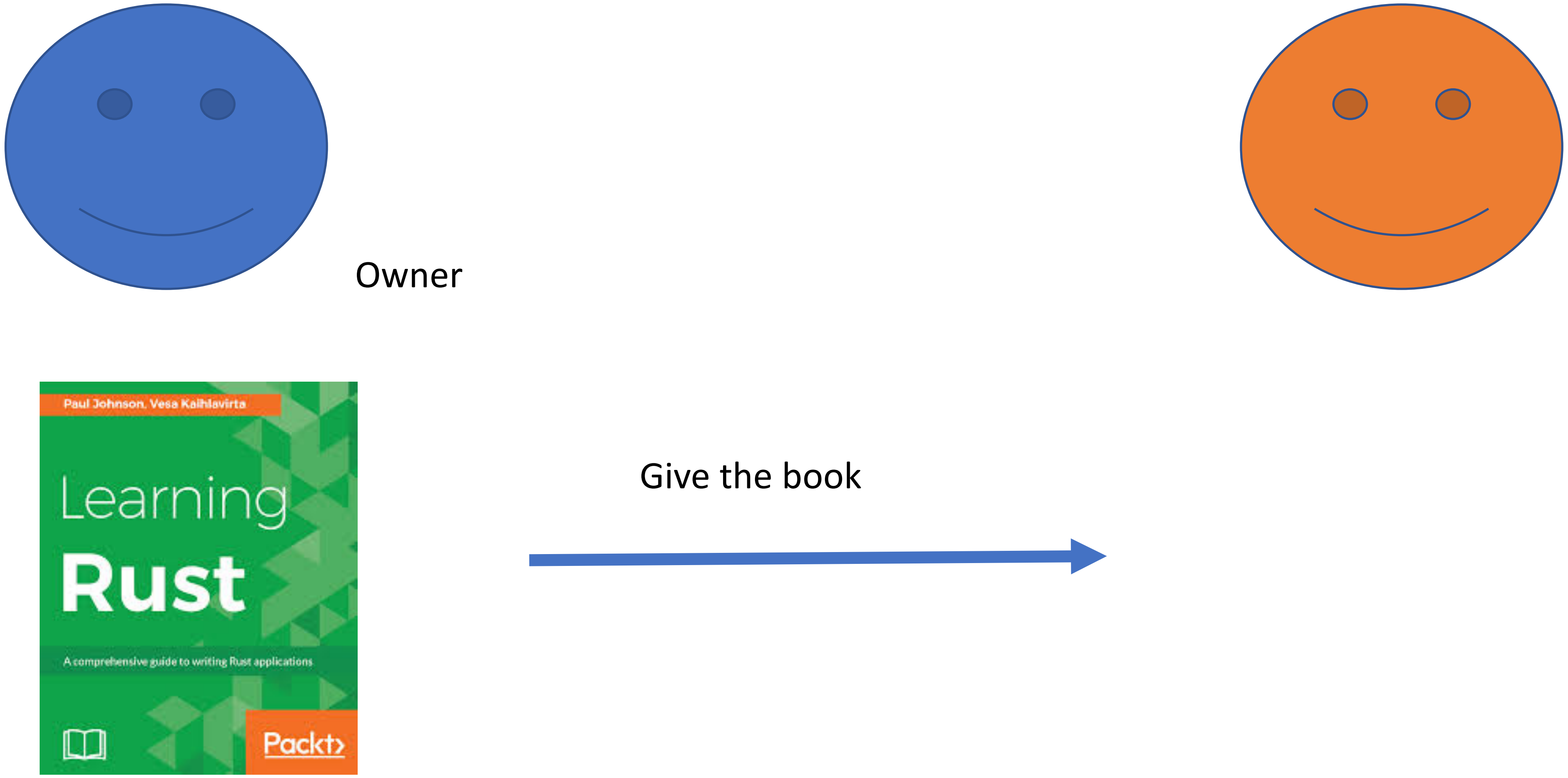
- No Low level control
- GC pauses -- suspension time
- requires runtime

Rust Solution

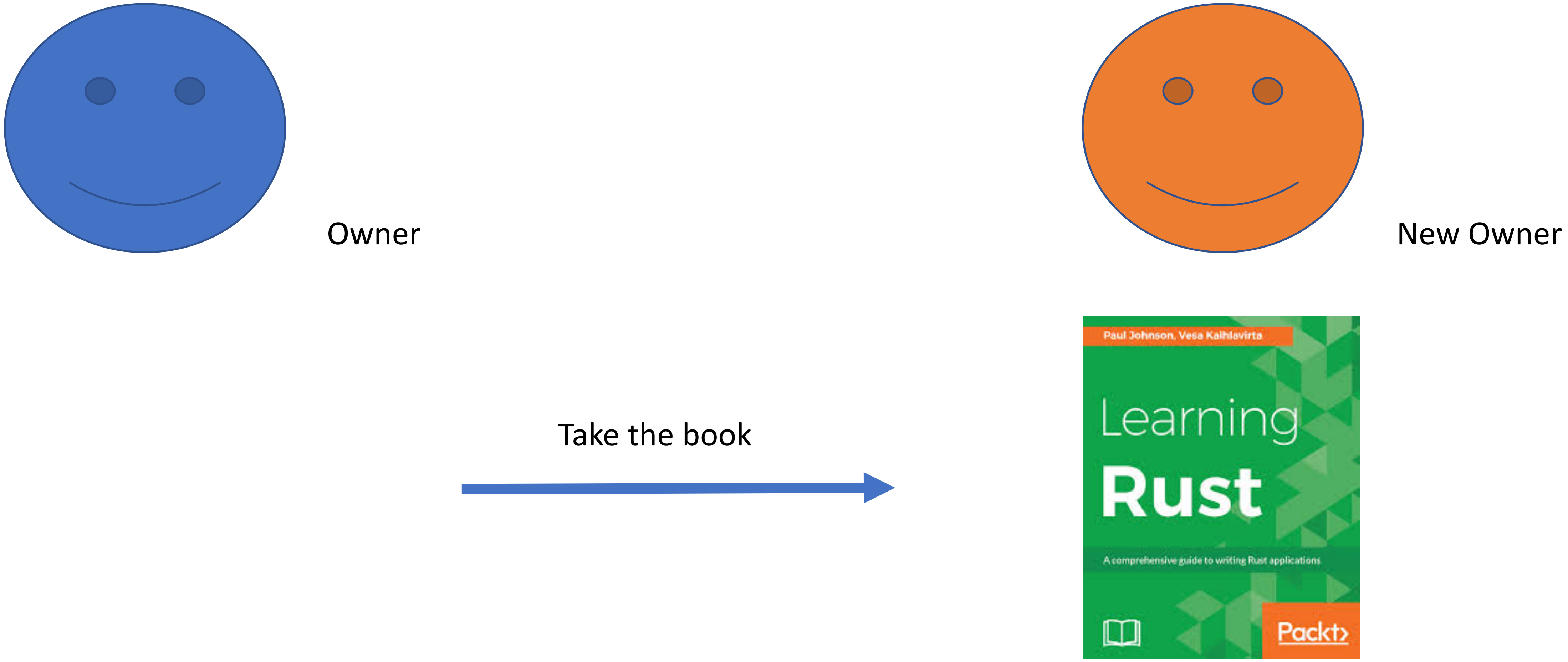
$$\begin{tikzcd}
 & \text{Ownership / borrowing} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{No need for runtime} & \text{Memory Safety} & \text{Data Races freedom} \\
\text{C++} \arrow[u] & \text{GC} \arrow[u] &
\end{tikzcd}$$

Ownership

~~Aliasing~~ + Mutation

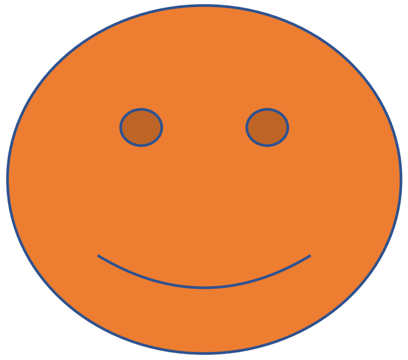

Owner
Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt
Give the book

Ownership


Owner
New Owner
Take the book
Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt›

Ownership

The owner decides to go away



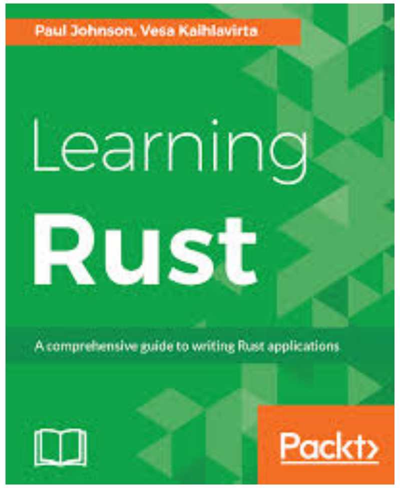
Take the book

Paul Johnson, Vesa Kaihlavirta
Learning
Rust
A comprehensive guide to writing Rust applications
Packt›

Ownership

The new owner goes away

Destroy the book

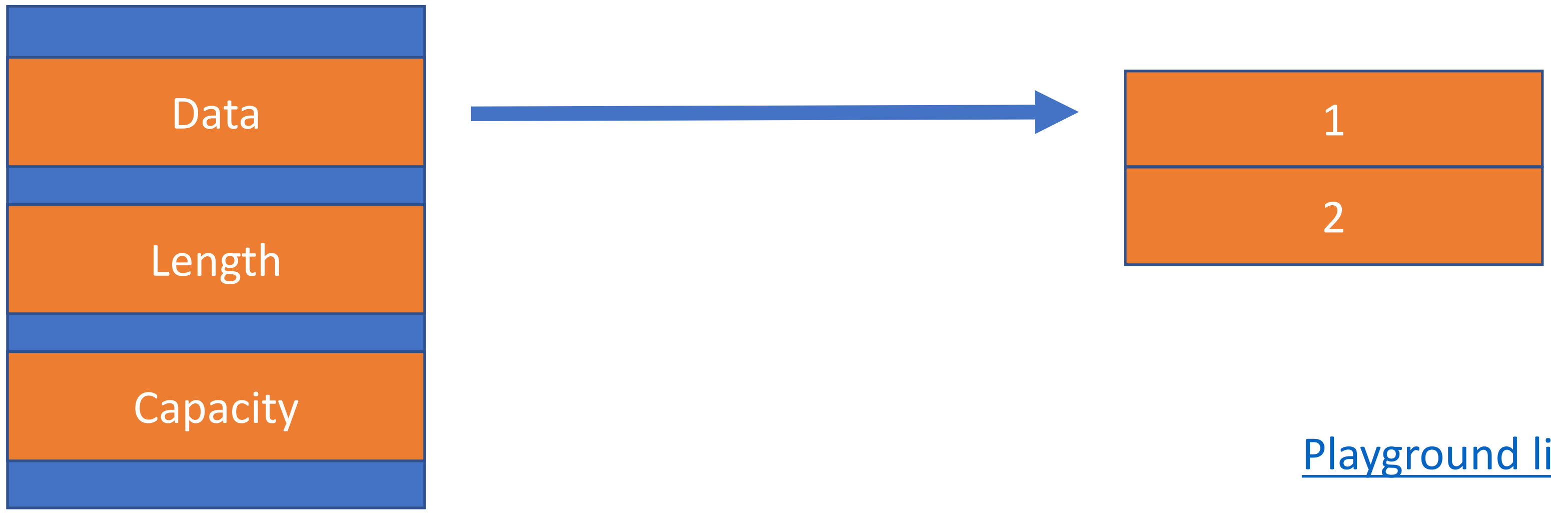

Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt

Compiler enforces ownership

```
fn  give(){
        let  mut  vec  =  Vec::new();
        vec.push(1);
        vec.push(2);
        take(vec);
        ~~vec.push(3);~~
}
```

```
fn take(vec: Vec<i32>) {
        println!("{:?}",  vec);
}
```

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

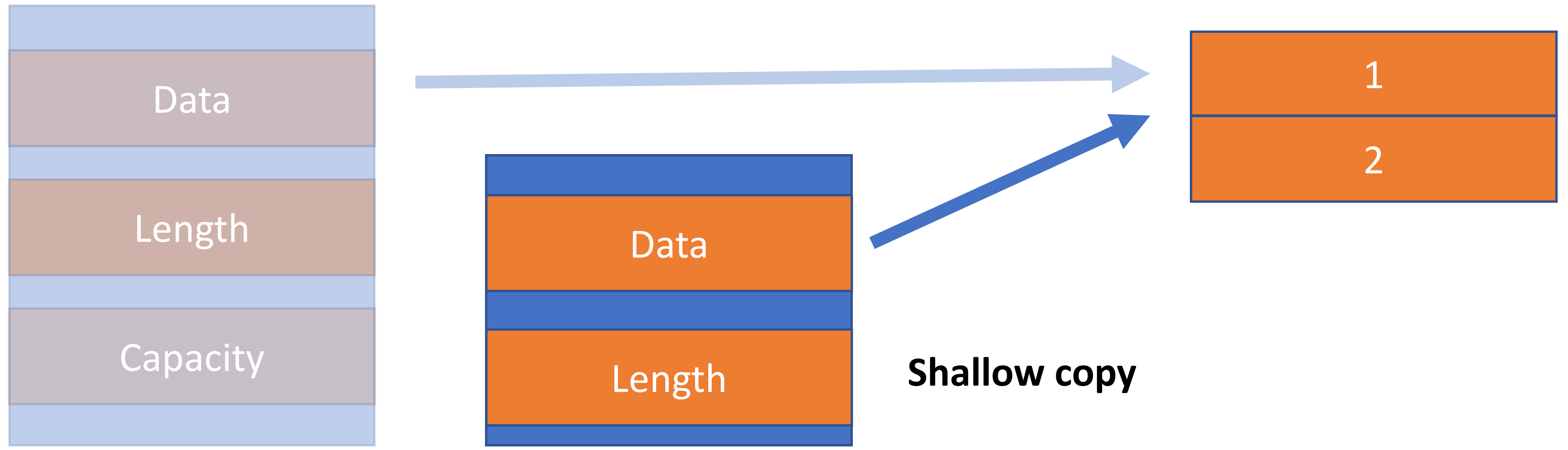


Playground link

Compiler enforces ownership

```
fn  give(){
        let  mut  vec  =  Vec::new();
        vec.push(1);
        vec.push(2);
        take(vec);
}
```

```
fn take(vec: vec<i32>) {
        println!("{:?}",  vec);
}
```



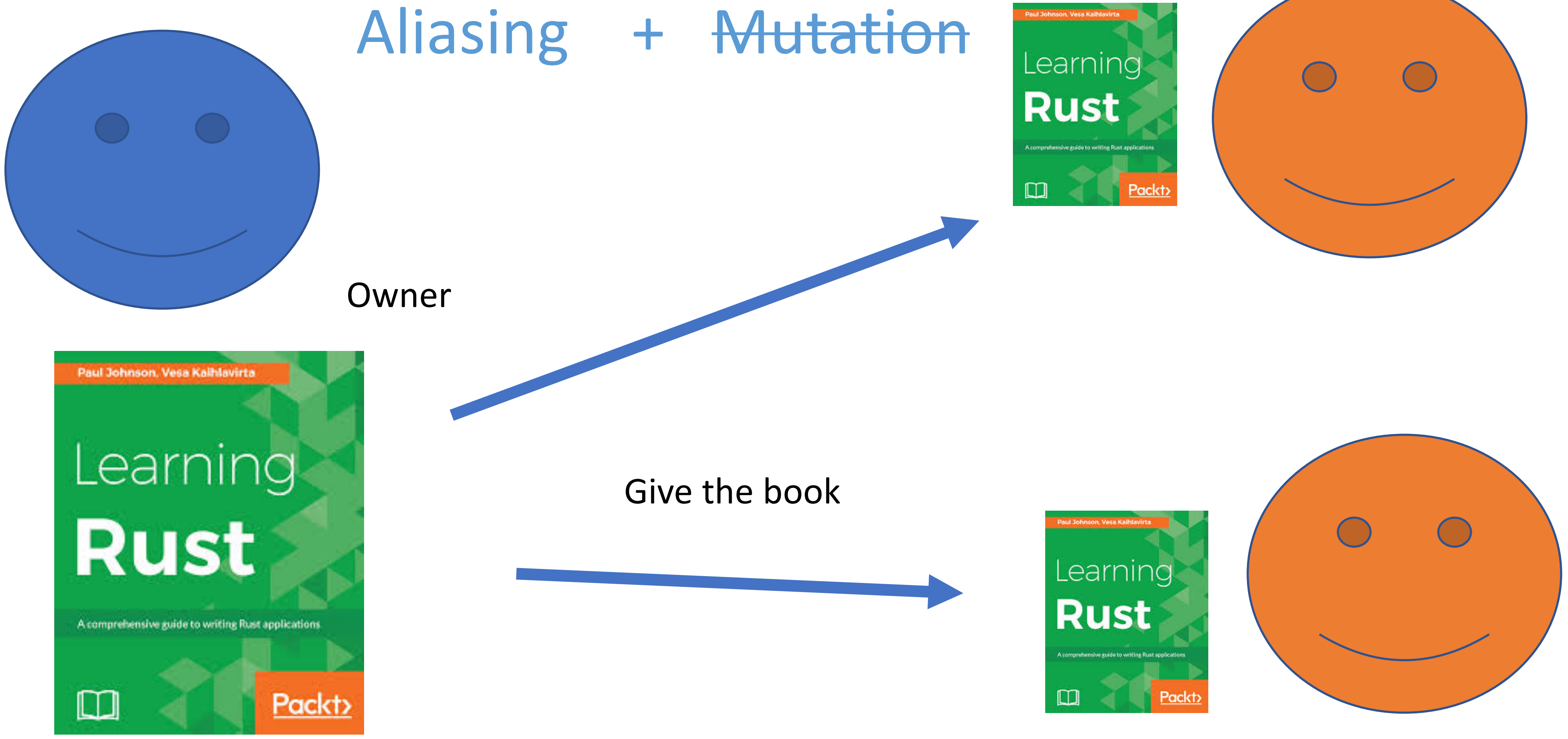
Shallow copy

Rules of Ownership

- Each value has its ***owner***.
- There can only be one **owner** at a time.
- When the owner goes out of scope, the value will be dropped.

Borrowing (Shared Borrowing(**&T**))

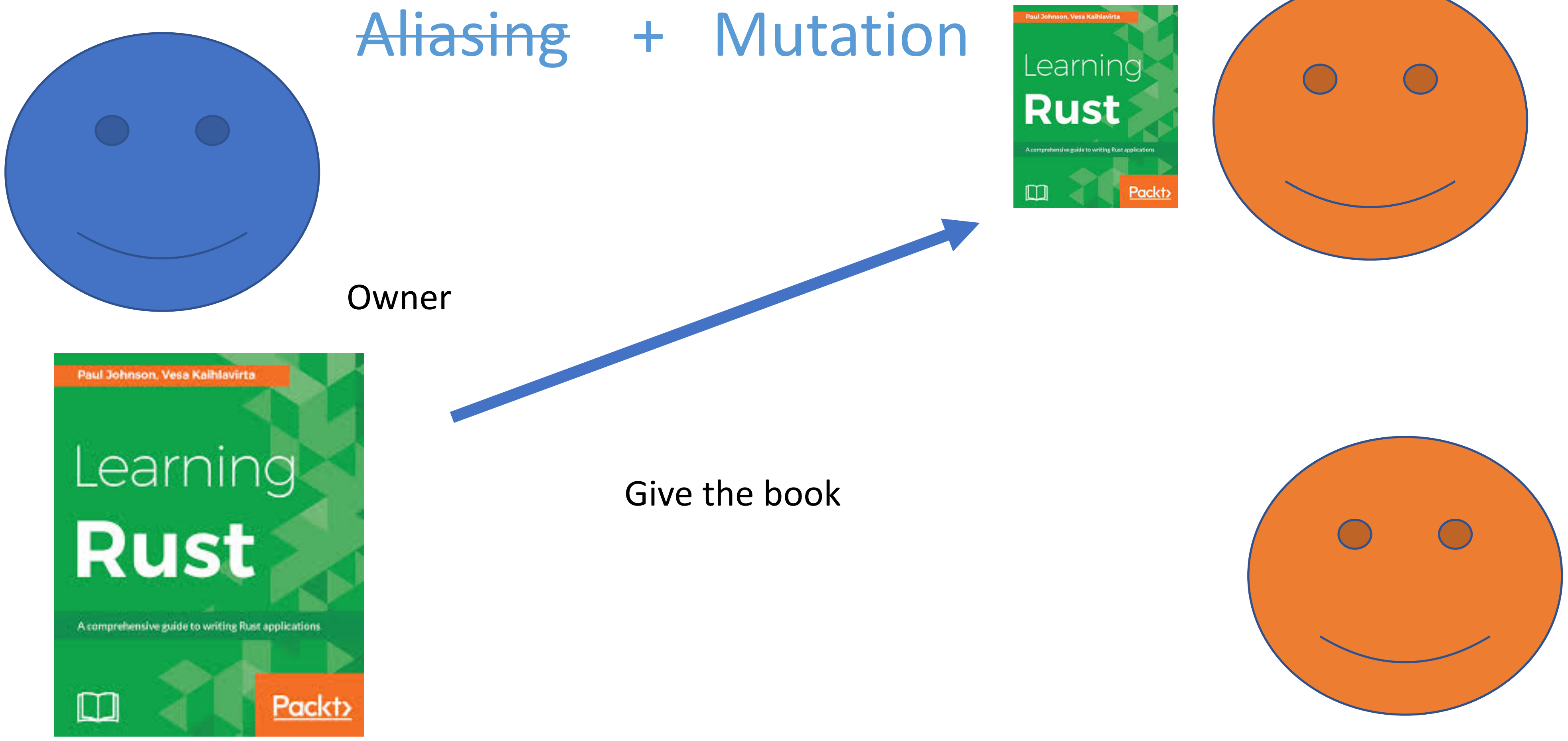
Aliasing + ~~Mutation~~



Owner
Give the book
Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt

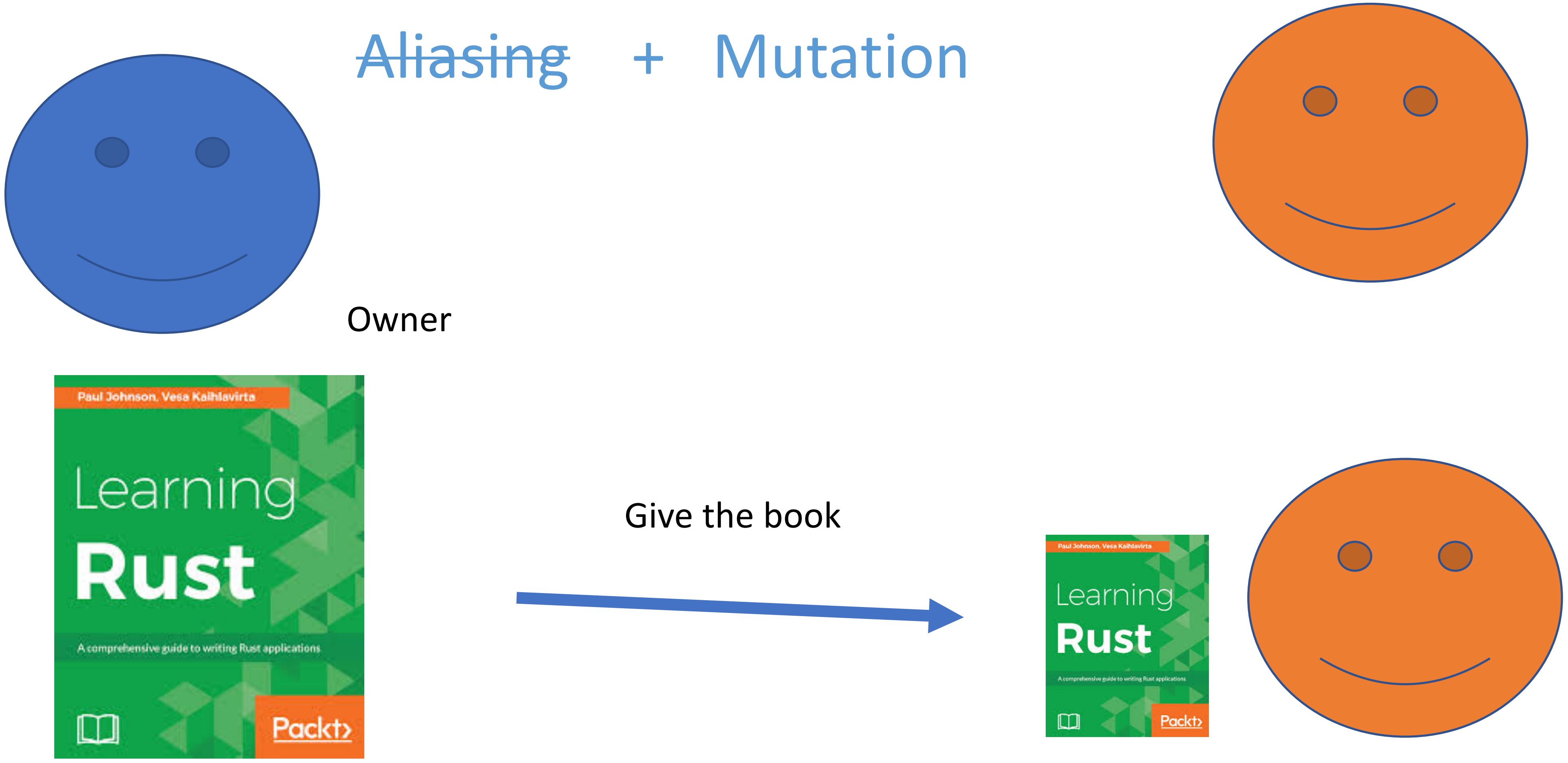
Borrowing (Mutable Borrowing(**&mut T**))

~~Aliasing~~ + Mutation


Owner
Give the book
Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt

Borrowing (Mutable Borrowing(**&mut T**))

~~Aliasing~~ + Mutation


Owner
Paul Johnson, Vesa Kaihlavirta
Learning Rust
A comprehensive guide to writing Rust applications
Packt>
Give the book

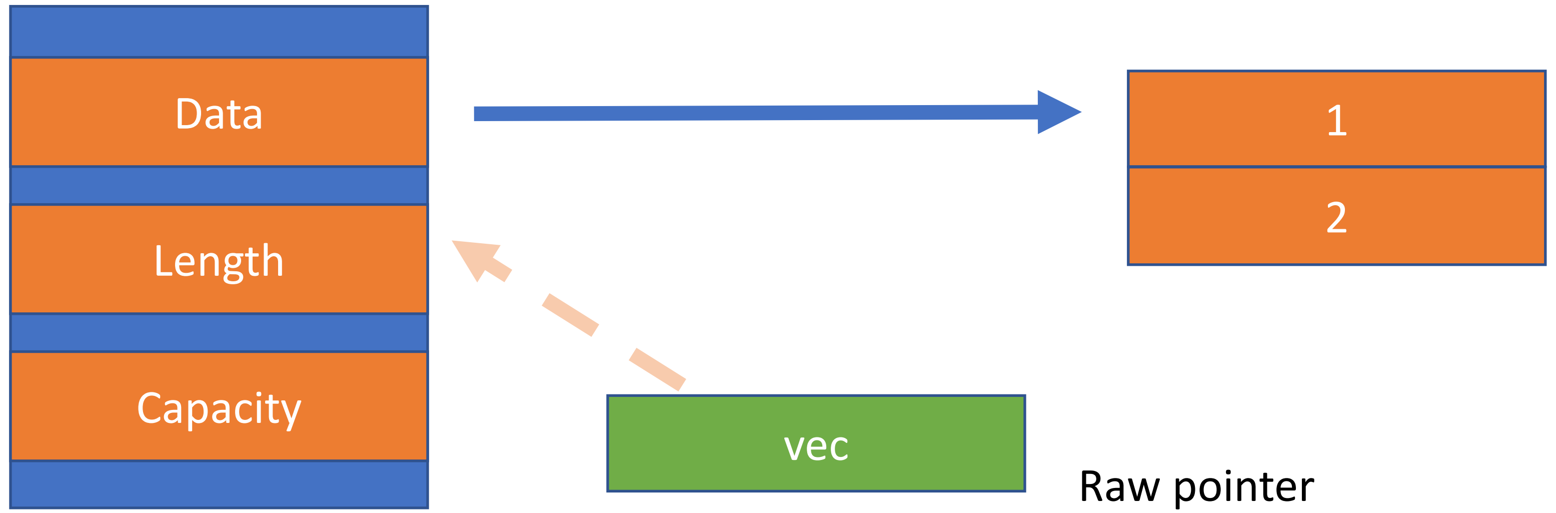
Shared borrow (&T)

Shared reference to &vec<i32>

```
fn  lender(){
        let  mut  vec  =  Vec::new();
        vec.push(1);
        vec.push(2);
        user(&vec);}
```

Loan out the vec

```
fn user(vec: &vec<i32>) {
        println!("{:?}",  vec);
}
```



Raw pointer

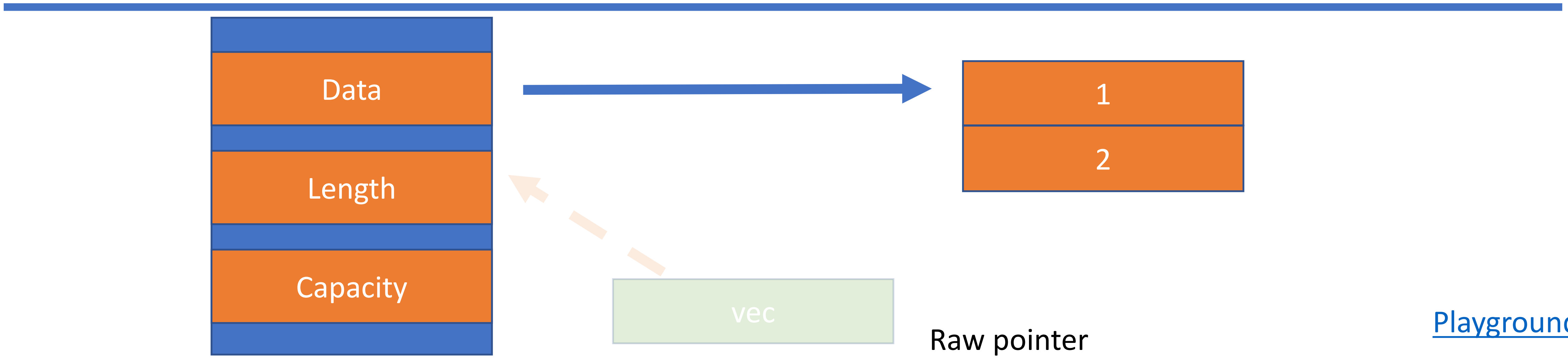
Playground link

Shared borrow

```
fn  lender(){
        let  mut  vec  =  Vec::new();
        vec.push(1);
        vec.push(2);
        user(&vec);}
```

```
fn user(vec: &vec<i32>) {
        println!("{:?}",  vec);
}
```

End forget about the vec



Playground link

Shared reference are immutable

```
fn user(vec: &vec<i32>) {
        ~~vec.push(1);~~
        ~~vec.push(2);~~
}
```

Aliasing + ~~Mutation~~

Error : cannot mutate shared references

Linux Tools in Rust

What we will talk about today

grep

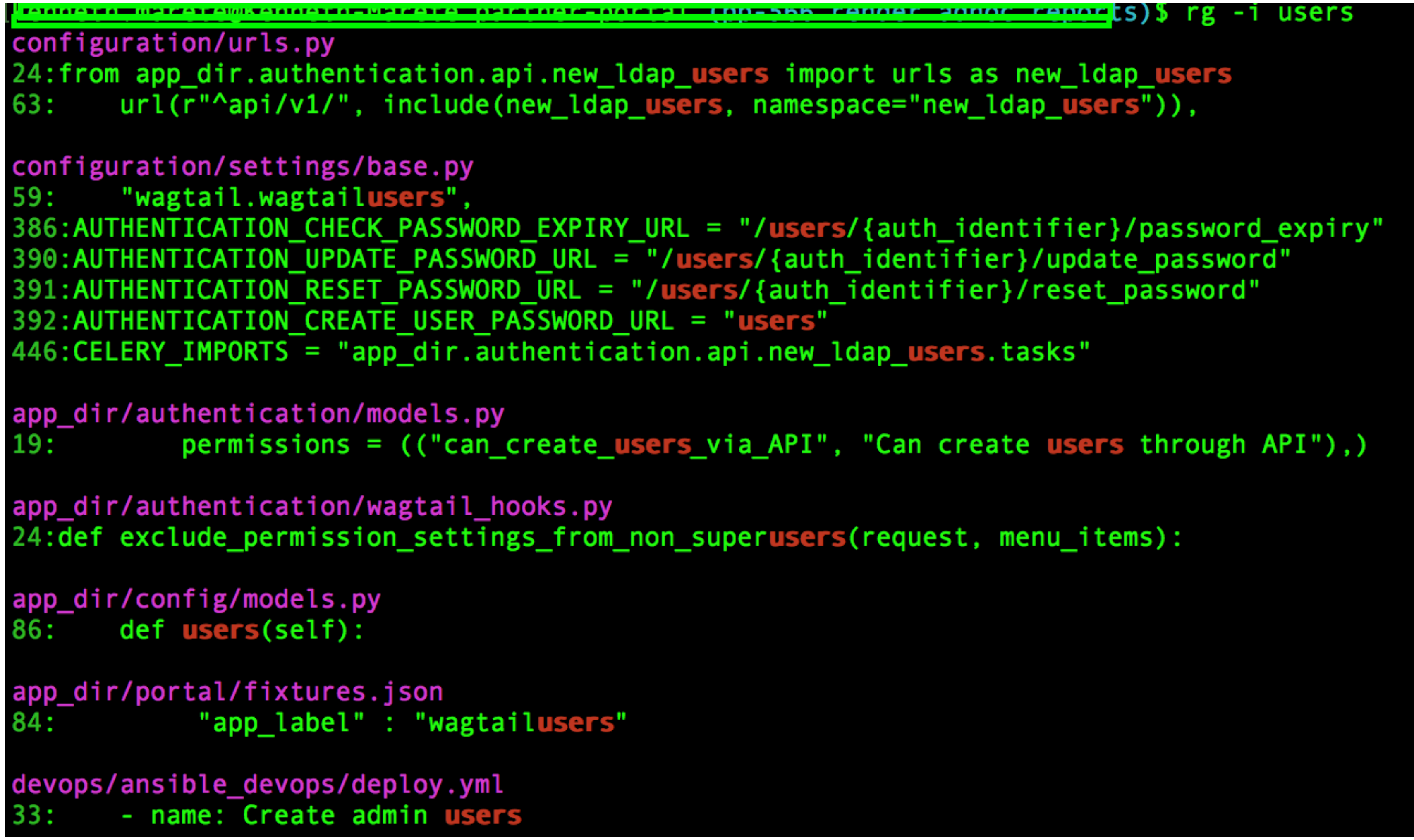
global regular expression print

grep ”<word>" <file>

```
grep -i "license" GPL-3
grep "^GNU" GPL-3
```

ripgrep (rg)

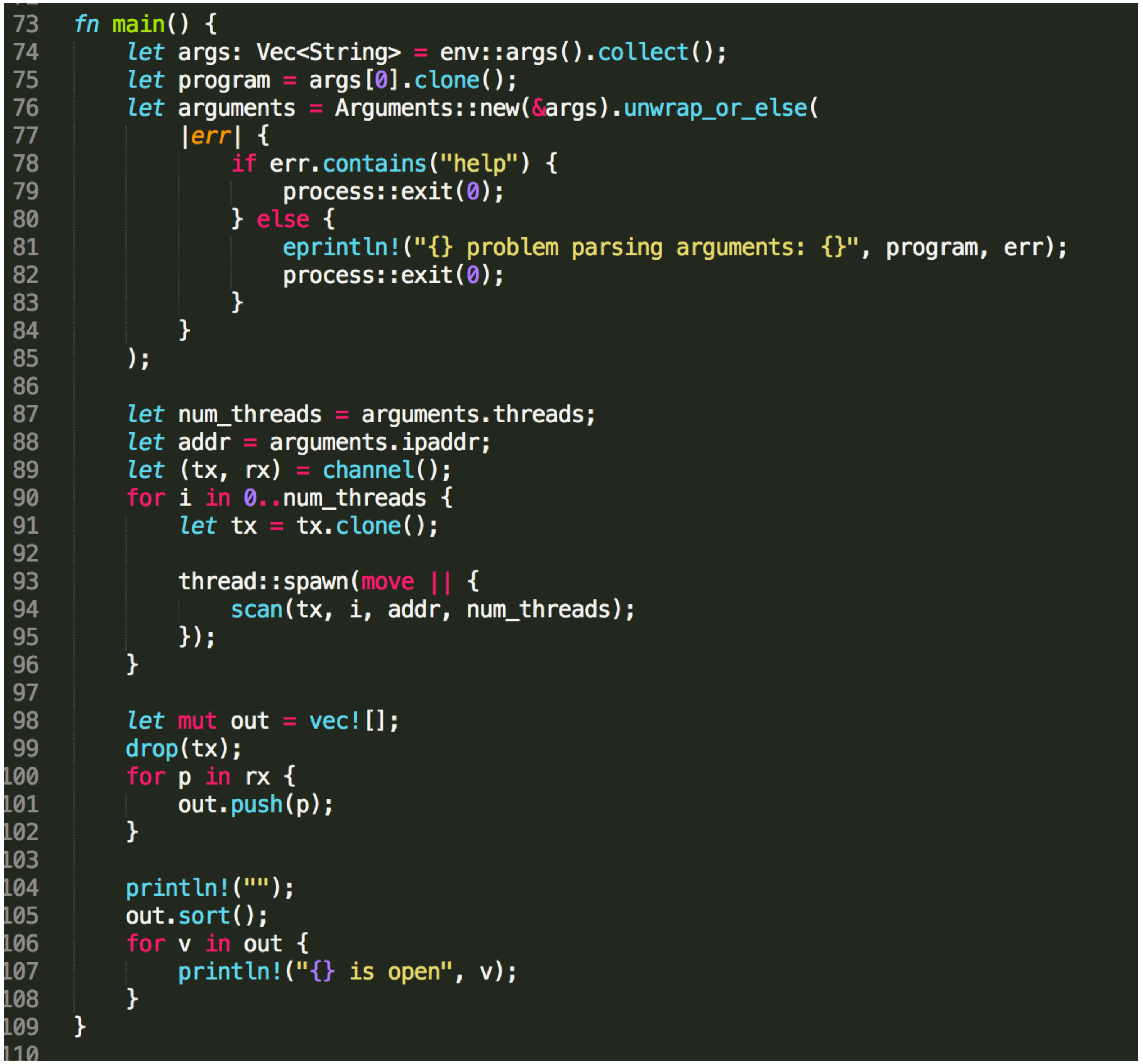
ripgrep is a line-oriented search tool that recursively searches your current directory for a regex pattern while respecting your gitignore rules.



```
[redacted]ts)$ rg -i users
configuration/urls.py
24:from app_dir.authentication.api.new_ldap_users import urls as new_ldap_users
63:    url(r"^api/v1/", include(new_ldap_users, namespace="new_ldap_users")),

configuration/settings/base.py
59:    "wagtail.wagtailusers",
386:AUTHENTICATION_CHECK_PASSWORD_EXPIRY_URL = "/users/{auth_identifier}/password_expiry"
390:AUTHENTICATION_UPDATE_PASSWORD_URL = "/users/{auth_identifier}/update_password"
391:AUTHENTICATION_RESET_PASSWORD_URL = "/users/{auth_identifier}/reset_password"
392:AUTHENTICATION_CREATE_USER_PASSWORD_URL = "users"
446:CELERY_IMPORTS = "app_dir.authentication.api.new_ldap_users.tasks"

app_dir/authentication/models.py
19:        permissions = (("can_create_users_via_API", "Can create users through API"),)

app_dir/authentication/wagtail_hooks.py
24:def exclude_permission_settings_from_non_superusers(request, menu_items):

app_dir/config/models.py
86:    def users(self):

app_dir/portal/fixtures.json
84:         "app_label" : "wagtailusers"

devops/ansible_devops/deploy.yml
33:    - name: Create admin users
```

Port Sniffer

https://github.com/tensor-programming/Rust_Port_Sniffer.git



```rust
fn main() {
    let args: Vec<String> = env::args().collect();
    let program = args[0].clone();
    let arguments = Arguments::new(&args).unwrap_or_else(
        |err| {
            if err.contains("help") {
                process::exit(0);
            } else {
                eprintln!("{} problem parsing arguments: {}", program, err);
                process::exit(0);
            }
        }
    );

    let num_threads = arguments.threads;
    let addr = arguments.ipaddr;
    let (tx, rx) = channel();
    for i in 0..num_threads {
        let tx = tx.clone();

        thread::spawn(move || {
            scan(tx, i, addr, num_threads);
        });
    }

    let mut out = vec![];
    drop(tx);
    for p in rx {
        out.push(p);
    }

    println!("");
    out.sort();
    for v in out {
        println!("{} is open", v);
    }
}
```

Rust Nairobi User group Meetup

https://www.meetup.com/Rust-Nairobi/

https://twitter.com/RustNairobi

https://github.com/rust-nairobi/

Q & A

Thank You