ESCAPE 824064

LAPP
Laboratoire d'Annecy de Physique des Particules

Introduction to Rust

Pierre Aubert

ESCAPE
OSCARS
Open Science Clusters' Action for Research & Society
EURO
cnrs
UNIVERSITÉ SAVOIE MONT BLANC
eosc | EVERSE
CCFR CENTRE DE COMPÉTENCE HPC.HPDA.IA

History

History

1957

History

1957

History

1957

1959

History

1957

1959

History

1957

1959

1972

History

1957

1959

1972

History

1957

1959

1972

1985

History

1957

1959

1972

1985

History

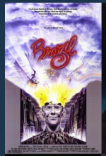
1957

1959

1972

1985

1991



History

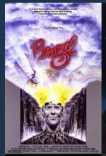
1957

1959

1972

1985

1991



History


1957

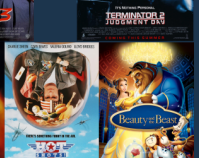
1959

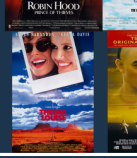

1972


1985


1991


2015





History

1957

1959

1972

1985

1991

2015

Rust

Rust

is a **system language** such as

Rust

is a **system language** such as

Cargo

Project builder

Rust

is a **system language** such as

Cargo **Project builder**

Rust-doc **Doc builder**

doxygen

Rust

is a **system language** such as

Cargo **Project builder**

Rust-doc **Doc builder**

doxygen

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

```cpp
#include <iostream>
int main(int argc, char** argv){
    std::cout << "Hello world !" << std::endl;
    return 0;
}
```

Rust

is a **system language** such as

Cargo **Project builder**

Rust-doc **Doc builder**


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

```cpp
#include <iostream>
int main(int argc, char** argv){
»       std::cout << "Hello world !" << std::endl;
»       return 0;
}
```

Project generator :
cargo new --name rusty_example RustyExample
Program Project

A lot of IDEs

Rust is Brilliant

Rust is Brilliant

Trait -> API contract

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Rust is Brilliant

Trait -> API contract

Abstract Trait

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

```
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```cpp
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```cpp
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

assert(**isSimpleType**(data));

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

assert(**isSimpleType**(data));

or for static usage

```
///Generic Implementation of "trait" isSimpleType
template<typename T>
bool isSimpleType(){return false;}
///Specialisation of "trait" isSimpleType for int
template<>
bool isSimpleType<int>(){return true;}
```

Rust is Brilliant

Trait -> API contract

```
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

assert(**isSimpleType**(data));

or for static usage

```
///Generic Implementation of "trait" isSimpleType
template<typename T>
bool isSimpleType(){return false;}
///Specialisation of "trait" isSimpleType for int
template<>
bool isSimpleType<int>(){return true;}
```

assert(**isSimpleType<**type>());

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```cpp
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

assert(**isSimpleType**(data));

or for static usage

```cpp
///Generic Implementation of "trait" isSimpleType
template<typename T>
bool isSimpleType(){return false;}
///Specialisation of "trait" isSimpleType for int
template<>
bool isSimpleType<int>(){return true;}
```

assert(**isSimpleType**<type>());

Shadok is implicit

Rust is Brilliant

Trait -> API contract

```rust
/// Defines if a type is primitive
trait IsSimpleType {
    fn is_simple_type(&self) -> bool;
}

impl IsSimpleType for i32{
    fn is_simple_type(&self) -> bool{true}
}

struct Shadok{}

impl IsSimpleType for Shadok{
    fn is_simple_type(&self) -> bool{false}
}
```

Abstract Trait

Trait for int 32

Trait for Shadok

Shadok is explicit

```rust
let value: i32 = 42;
assert_eq!(value.is_simple_type(), true);
let shadok: Shadok = Shadok {};
assert_eq!(shadok.is_simple_type(), false);
```

Trait **isSimpleType** not implemented for **Gibi**

Equivalent implementations in **C++**

Trait for int 32

```cpp
///Declaration of "trait" isSimpleType for int
bool isSimpleType(const int & data){return true;}
///Overload of "trait" isSimpleType for Shadok
bool isSimpleType(const Shadok & data){return false;}
```

Trait for Shadok

Shadok is explicit

assert(**isSimpleType**(data));

or for static usage

```cpp
///Generic Implementation of "trait" isSimpleType
template<typename T>
bool isSimpleType(){return false;}
///Specialisation of "trait" isSimpleType for int
template<>
bool isSimpleType<int>(){return true;}
```

assert(**isSimpleType**<type>());

Shadok is implicit

Can be used with **Gibi**

Rust is Brilliant

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
    - Even primitive type : **i32::shadok()**

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
    - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
    - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**
    - Introducing **Life Time** of variables

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
  - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**
  - Introducing **Life Time** of variables

- **unsafe** keyword -> when **compiler cannot prove** a correct behaviour

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
  - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**
  - Introducing **Life Time** of variables

- **unsafe** keyword -> when **compiler cannot prove** a correct behaviour

- The goal is to use **no unsafe** at all

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
  - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**
  - Introducing **Life Time** of variables

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

- **unsafe** keyword -> when **compiler cannot prove** a correct behaviour

- The goal is to use **no unsafe** at all
  - better to **give** the **compiler all** needed **information**

Rust is Brilliant

- Split between **data** (**struct**) and **methods** (**impl**)

- Any **type** can have any **trait**
  - Even primitive type : **i32::shadok()**

- **Compiler** can **prove** a lot of **behaviours**
  - Introducing **Life Time** of variables

```
trait Shadok{
»       ///An important static method
»       fn shadok();
}

impl Shadok for i32 {
»       ///Important statis method
»       fn shadok(){
»       »       println!("Shadok");
»       }
}
```

- **unsafe** keyword -> when **compiler cannot prove** a correct behaviour

- The goal is to use **no unsafe** at all
  - better to **give** the **compiler all** needed **information**

- Powerfull **error handling** : **Option<T>**, **Result<T, E>**, **panic!**

Rust is Light

Rust is Light

- No constant suffix/cast needed

Rust is Light

- No constant suffix/cast needed

```
size_t val = 0lu;
```

Rust is Light

- No constant suffix/cast needed

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

Rust is Light

- No constant suffix/cast needed

```cpp
size_t val = 0lu;
```

```rust
let val: u64 = 0;
let var: i32 = 0;
```

- No default parameter

Rust is Light

- **No constant suffix/cast needed**

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

- **No default parameter**
    - **error**: **parameter defaults** are not supported

Rust is Light

- **No constant suffix/cast needed**

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

- **No default parameter**
  - **error**: **parameter defaults** are not supported

- **No function/method overload**

Rust is Light

- **No constant suffix/cast needed**

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

- **No default parameter**
  - **error**: **parameter defaults** are not supported

- **No function/method overload**

- **No virtual method**

Rust is Light

- **No constant suffix/cast needed**

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

- **No default parameter**
  - **error**: **parameter defaults** are not supported
- **No function/method overload**
- **No virtual method**
- **No inheritage**

Rust is Light

- **No constant suffix/cast needed**

```
size_t val = 0lu;
```

```
let val: u64 = 0;
let var: i32 = 0;
```

- **No default parameter**
  - **error**: **parameter defaults** are not supported
- **No function/method overload**
- **No virtual method**
- **No inheritage**

One main **question** :
Does this **type** implement this **trait** ?

Rust is Light

Overload / Specialization (kind of)

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

Template type

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Rust is Light

Overload / Specialization (kind of)

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Template type

Printable

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
        where T: Display + 'static
{
        if TypeId::of::<i32>() == TypeId::of::<T>() {
                println!("overload_function : this is an i32, value = {}", value);
        }else{
                println!("overload_function : this is an other type");
        }
}

fn main() {
        overload_function(42 as i32);
        overload_function(3.14 as f64);
}
```

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Template type

Printable

Statically defined

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Template type

Printable

Statically defined

Specialization for **i32**

Rust is Light

Overload / Specialization (kind of)

```rust
use std::{any::TypeId, fmt::Display};

///Try to overload function
/// # Parameters
/// - `value` : value
fn overload_function<T>(value: T)
»       where T: Display + 'static
{
»       if TypeId::of::<i32>() == TypeId::of::<T>() {
»       »       println!("overload_function : this is an i32, value = {}", value);
»       }else{
»       »       println!("overload_function : this is an other type");
»       }
}

fn main() {
»       overload_function(42 as i32);
»       overload_function(3.14 as f64);
}
```

TypeId :
- **unique id** of any type (such as C++)
- OK for **local compilation check**

Template type

Printable

Statically defined

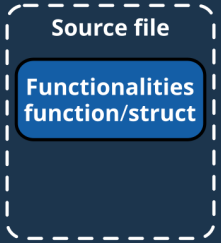
Specialization for **i32**

Call for **other types**

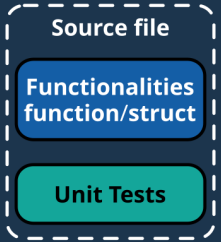
Rust is nice

Rust is nice

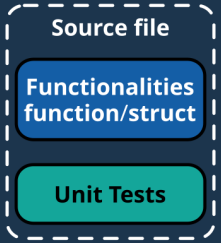
Rust is nice



Rust is nice

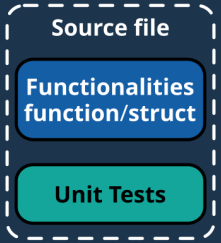


Rust is nice



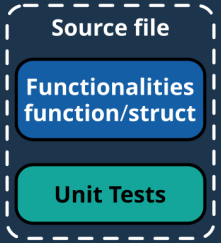
- **Default** is **constant**

Rust is nice



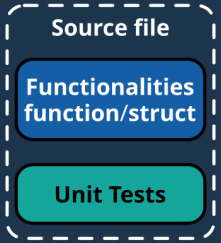
- **Default** is **constant**
    - No need to put **const** everywhere

Rust is nice



- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

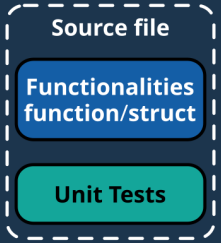
Rust is nice



- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

```
let value: i32 = 0;»     //const
let mut var: i32 = 0;»   //non-const
```

Rust is nice

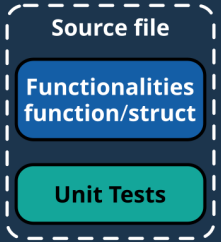


- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

```
let value: i32 = 0;»     //const
let mut var: i32 = 0;»   //non-const
```

But need to use **mut** for all **mutable** variables

Rust is nice



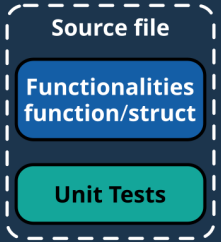
- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

```
let value: i32 = 0;»     //const
let mut var: i32 = 0;»   //non-const
```

But need to use **mut** for all **mutable** variables

todo! macro to mark **unimplemented** feature / case

Rust is nice



- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

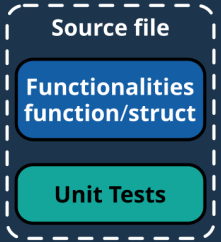
```
let value: i32 = 0;»     //const
let mut var: i32 = 0;»   //non-const
```

But need to use **mut** for all **mutable** variables

todo! macro to mark **unimplemented** feature / case

- **Rust** encourages **spliting** your **project** into few **crates** => **nice** for **reusability**
- **Cargo** : nice package manager
  - **Cargo.lock** : **exact** versions of all your **dependencies**

Rust is nice



- **Default** is **constant**
  - No need to put **const** everywhere
  - **Compiler** => **data** will **dive in cache** but never go up again

```
let value: i32 = 0;»     //const
let mut var: i32 = 0;»   //non-const
```

But need to use **mut** for all **mutable** variables

todo! macro to mark **unimplemented** feature / case

- **Rust** encourages **spliting** your **project** into few **crates** => **nice** for **reusability**
- **Cargo** : nice package manager
  - **Cargo.lock** : **exact** versions of all your **dependencies**

pixi.lock comes from **Cargo.lock**

Rust is Concise

Rust is Concise

Type :
- **char** : **i8**
- **unsigned char** : **u8**
- **short** : **i16**
- **unsigned short** : **u16**
- **int** : **i32**
- **unsigned int** : **u32**
- **float** : **f32**
- **double** : **f64**

Rust is Concise

Type :
- **char : i8**
- **unsigned char : u8**
- **short : i16**
- **unsigned short : u16**
- **int : i32**
- **unsigned int : u32**
- **float : f32**
- **double : f64**

- **const char * : str**
- **std::string : String**
- **std::vector : Vec**
- **function : fn**
- **public : pub**
- **implementation : impl**

Rust is Concise

Type :
- **char** : **i8**
- **unsigned char** : **u8**
- **short** : **i16**
- **unsigned short** : **u16**
- **int** : **i32**
- **unsigned int** : **u32**
- **float** : **f32**
- **double** : **f64**

- **const char *** : **str**
- **std::string** : **String**
- **std::vector** : **Vec**
- **function** : **fn**
- **public** : **pub**
- **implementation** : **impl**

Pointer and reference :
- ***** : **Box**
- **Reference** : **&**, **Ref**, **RefMut**, **RefCell**
- **Reference Counter** : **Rc**

Rust is Explicit

Rust is Explicit

Function call

Rust is Explicit

Function call

Rust

```rust
let mut var: u32 = 0;
function_call(&mut var);
```

C++

```cpp
int var = 0;
function_call(var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

C++

```
int var = 0;
function_call(var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

Rust is Explicit

Function call

Rust

```rust
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```rust
pub fn function_call(var: &mut u32);
```

C++

```cpp
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```cpp
void function_call(int var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```
void function_call(int var);
```

or var **could** be **modified**

```
void function_call(int & var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```
void function_call(int var);
```

or var **could** be **modified**

```
void function_call(int &var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

No implicit cast

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```
void function_call(int var);
```

or var **could** be **modified**

```
void function_call(int &var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

No implicit cast

Everything is **private** by **default** :
- **method**, **function**, **struct**, **trait**

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```
void function_call(int var);
```

or var **could** be **modified**

```
void function_call(int &var);
```

Rust is Explicit

Function call

Rust

```
let mut var: u32 = 0;
function_call(&mut var);
```

var **will be** modified by **function_call**

Only **one** possible call :

```
pub fn function_call(var: &mut u32);
```

No implicit cast

Everything is **private** by **default** :
- **method**, **function**, **struct**, **trait**

At **file** level

C++

```
int var = 0;
function_call(var);
```

var **can be** modified by **function_call**

var **could** remain **const**

```
void function_call(int var);
```

or var **could** be **modified**

```
void function_call(int &var);
```

Rust is Explicit

Macro

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (#define of C/C++)

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (**#define** of **C/C++**)

Macro Name

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (**#define** of **C/C++**)

Macro Name

Macro Parameters

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (**#define** of **C/C++**)

Macro Name

Macro Parameters

Macro Body

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (#define of C/C++)

Macro Name

Macro Parameters

Macro Body

Ugly bash-like syntax
Macro Variable Call

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}

create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (#define of C/C++)

Macro Name

Macro Parameters

Ugly bash-like syntax
Macro Variable Call

Macro Body

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}
```

Macro Call

```
create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (**#define** of **C/C++**)

Macro Name

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}
```

Macro Parameters

Macro Body

Ugly bash-like syntax

Macro Variable Call

Macro Parameters are typed !

Macro Call

```
create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Rust is Explicit

Macro

Macro Declaration (**#define** of **C/C++**)

Macro Name

```
macro_rules! create_hello_print {
»       ($function_name:ident, $ty:ty) => {
»       »
»       »       fn $function_name(value: $ty){
»       »       »       println!("Print of type {} : value = {}", stringify!($ty), value);
»       »       }
»       }
}
```

Macro Body

Macro Parameters

Ugly bash-like syntax
Macro Variable Call

Macro Call

```
create_hello_print!(print_u32, u32);
create_hello_print!(print_f32, f32);

fn main() {
»       println!("Hello, world macro!");
»       print_u32(23);
»       print_f32(23.0f32);
}
```

Macro Parameters are typed !

With very inspired types names :
- **ty** : type
- **tt** : token tree
- **pat** : pattern
- **ident** : identifier (function name, variable name, etc)
- **expr** : expression
- **block** : a block expression
- **path** : a TypePath style path
- etc

Rust is Subtle

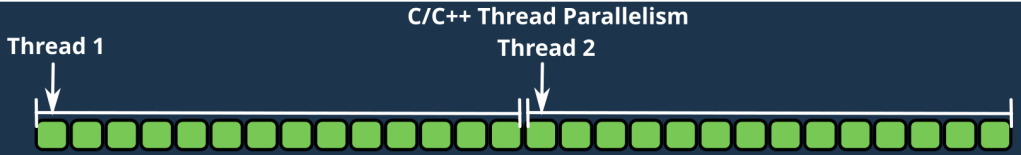
Rust is Subtle

C/C++ Thread Parallelism

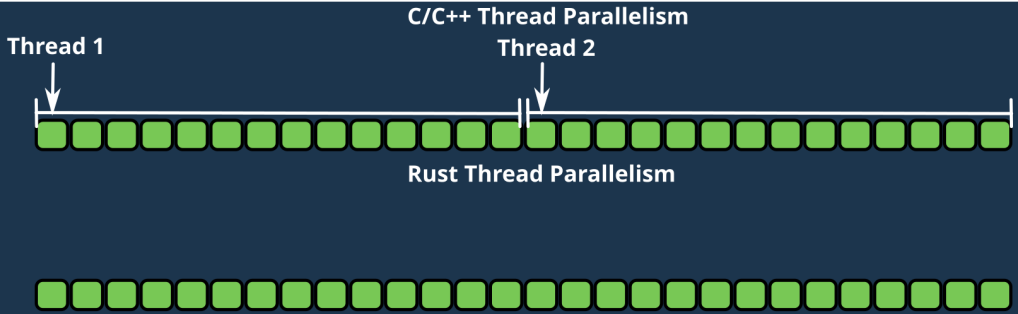
Rust is Subtle

C/C++ Thread Parallelism

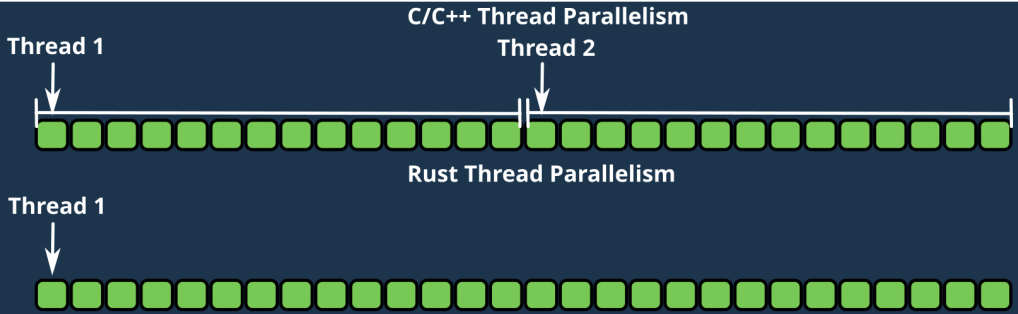
Rust is Subtle



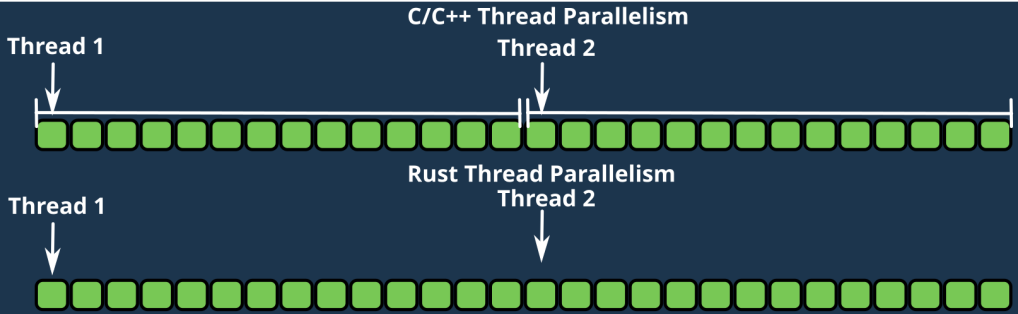
Rust is Subtle



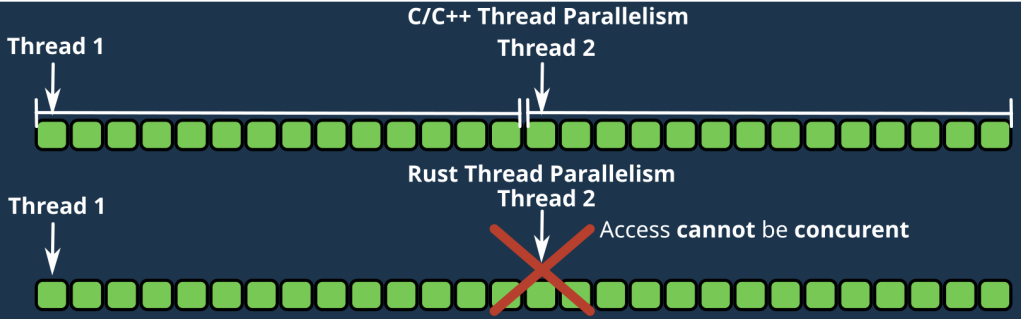
Rust is Subtle



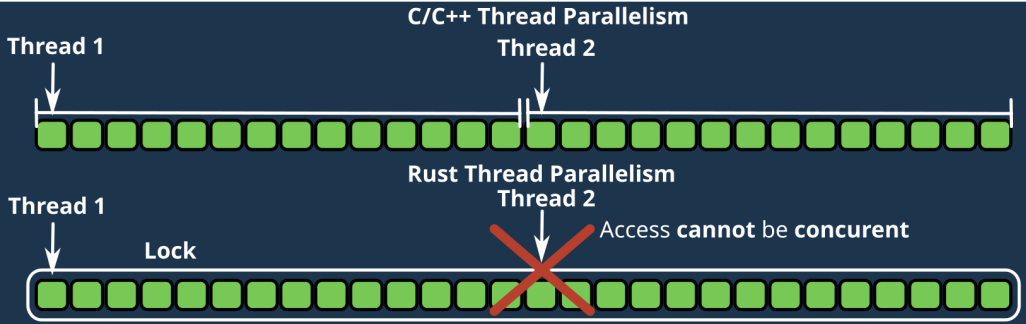
Rust is Subtle



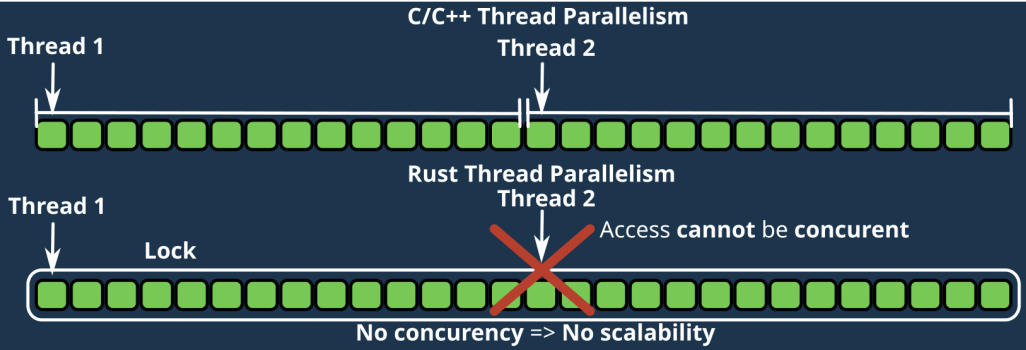
Rust is Subtle



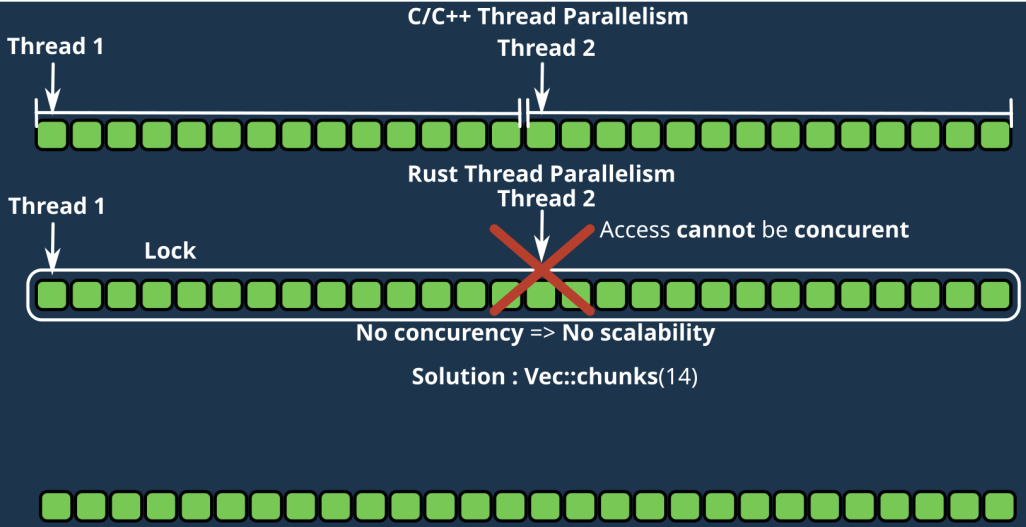
Rust is Subtle



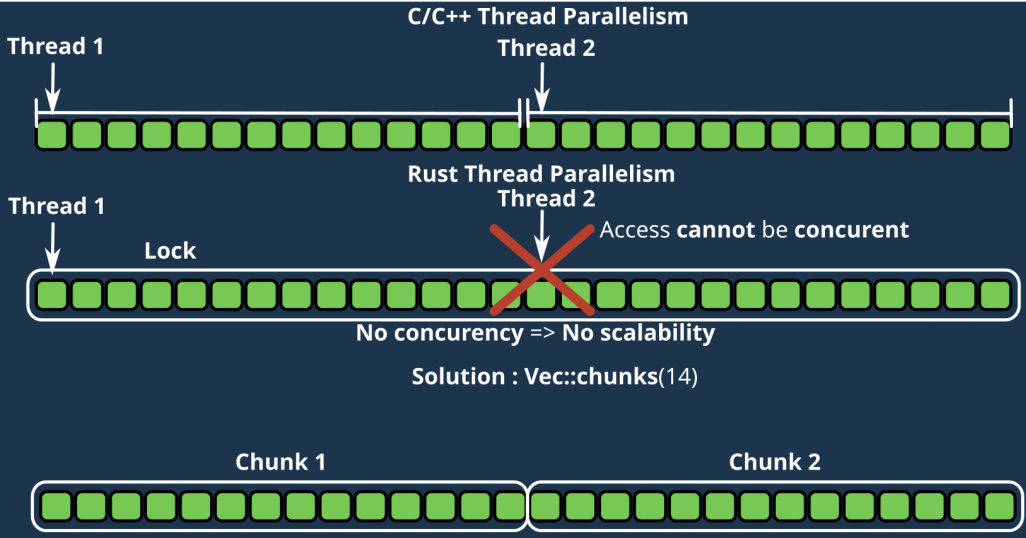
Rust is Subtle



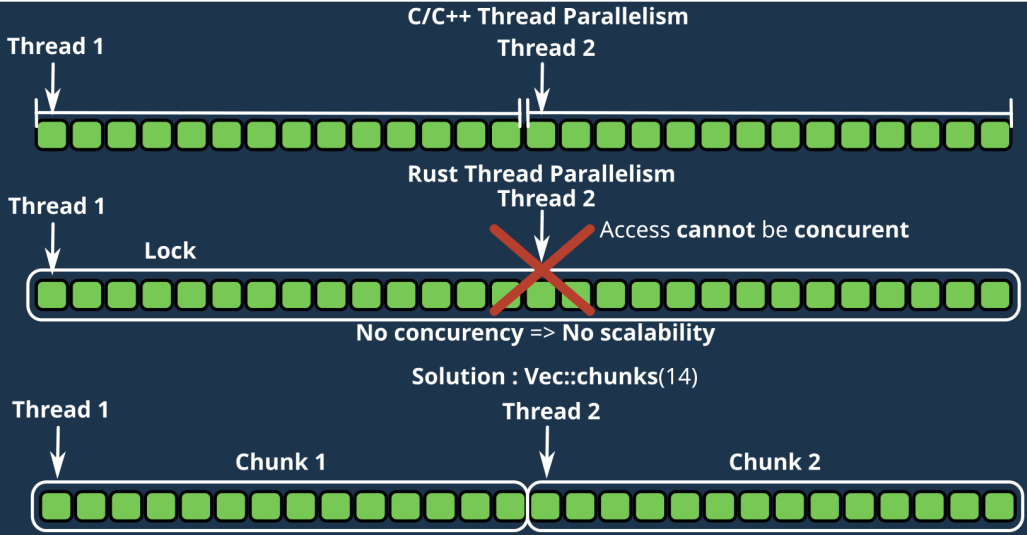
Rust is Subtle



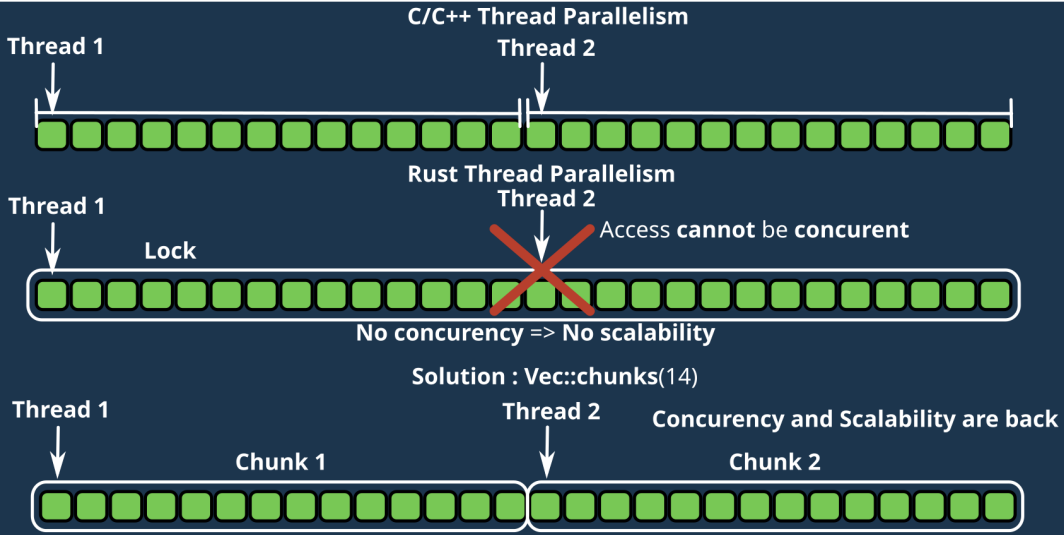
Rust is Subtle



Rust is Subtle



Rust is Subtle



Rust is Powerfull

Rust is Powerfull

- **Iterators** are very powerfull

Rust is Powerfull

- **Iterators** are very powerfull
    - **enumerate**
    - **step_by**
    - **chain**
    - **take**
    - ...

Rust is Powerfull

- **Iterators** are very powerfull
  - **enumerate**
  - **step_by**
  - **chain**
  - **take**
  - ...

Lazy evaluation of the compiler

Rust is Powerfull

- **Iterators** are very powerfull
  - **enumerate**
  - **step_by**
  - **chain**
  - **take**
  - ...

Lazy evaluation of the compiler

Such as **C++ 23** : **std::views::transform**

Rust is Powerfull

- **Iterators** are very powerfull
  - **enumerate**
  - **step_by**
  - **chain**
  - **take**
  - ...

- **zip** is working well

Lazy evaluation of the compiler

Such as **C++ 23** : **std::views::transform**

Rust is Powerfull

- **Iterators** are very powerfull
  - **enumerate**
  - **step_by**
  - **chain**
  - **take**
  - ...

Lazy evaluation of the compiler

Such as **C++ 23** : **std::views::transform**

- **zip** is working well

```
let vec_a: Vec<i32> = vec![1, 2, 3, 4];
let vec_b: Vec<i32> = vec![10, 20, 30, 40];
for (a, b) in vec_a.iter().zip(vec_b.iter()) {
»       println!("a = {}, b = {}", a, b);
}
```

Rust is Powerfull

- **Iterators** are very powerfull
  - **enumerate**
  - **step_by**
  - **chain**
  - **take**
  - ...

Lazy evaluation of the compiler

Such as **C++ 23** : **std::views::transform**

- **zip** is working well

```
let vec_a: Vec<i32> = vec![1, 2, 3, 4];
let vec_b: Vec<i32> = vec![10, 20, 30, 40];
for (a, b) in vec_a.iter().zip(vec_b.iter()) {
»       println!("a = {}, b = {}", a, b);
}
```

Such as **C++ 23** : **std::views::zip**

Rust is Complex

Rust is Complex

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

Rust is Complex

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

The **enum** is **not just** an **enum**...

Rust is Complex

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

The **enum** is **not just** an **enum**...

```
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

Rust is Complex

Debug print

```
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

The **enum** is **not just** an **enum**...

Rust is Complex

Debug print

```
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

The **enum** is **not just** an **enum**...

```
///Gibi
#[derive(Debug)]
struct Gibi{
»       /// Value of the Gibi
»       value: i32
}
```

Rust is Complex

Debug print

```rust
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

Classic **enum** example :

```rust
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

The **enum** is **not just** an **enum**...

```rust
///Gibi
#[derive(Debug)]
struct Gibi{
»       /// Value of the Gibi
»       value: i32
}
```

```rust
///Contains a Shadok, a Gibi or nothing
enum Rouxel{
»       Shadok(Shadok),
»       Gibi(Gibi),
»       Empty
}
```

Rust is Complex

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

Debug print

```
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

The **enum** is **not just** an **enum**...

```
///Gibi
#[derive(Debug)]
struct Gibi{
»       /// Value of the Gibi
»       value: i32
}
```

```
///Contains a Shadok, a Gibi or nothing
enum Rouxel{
»       Shadok(Shadok),
»       Gibi(Gibi),
»       Empty
}
```

```
///Print a Rouxel
/// # Parameters
/// - `rouxel` : Rouxel to be printed
fn print_rouxel(rouxel: Rouxel){
»       match rouxel {
»       »       Rouxel::Shadok(shadok) => println!("Shadok {:?}", shadok),
»       »       Rouxel::Gibi(gibi) => println!("Gibi {:?}", gibi),
»       »       Rouxel::Empty => println!("Empty")
»       }
}
```

Rust is Complex

Classic **enum** example :

```
pub enum PClockMode {
»       NoMock,
»       Mock,
»       MockRecord,
}
```

Debug print

```
///Shadok
#[derive(Debug)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64
}
```

The **enum** is **not just** an **enum**...

```
///Gibi
#[derive(Debug)]
struct Gibi{
»       /// Value of the Gibi
»       value: i32
}
```

```
///Contains a Shadok, a Gibi or nothing
enum Rouxel{
»       Shadok(Shadok),
»       Gibi(Gibi),
»       Empty
}
```

Rust **enum** are **trees**

```
///Print a Rouxel
/// # Parameters
/// - `rouxel` : Rouxel to be printed
fn print_rouxel(rouxel: Rouxel){
»       match rouxel {
»       »       Rouxel::Shadok(shadok) => println!("Shadok {:?}", shadok),
»       »       Rouxel::Gibi(gibi) => println!("Gibi {:?}", gibi),
»       »       Rouxel::Empty => println!("Empty")
»       }
}
```

Rust is too Powerfull

Rust is too Powerfull

Add trait to a **struct** by calling a **proc-macro**

```
#[derive(Default, Debug, MyCustomMacro)]
struct Shadok{}
```

Rust is too Powerfull

Add trait to a **struct** by calling a **proc-macro**

```
#[derive(Default, Debug, MyCustomMacro)]
struct Shadok{}
```

Also for **comparison**, **equality**, **display**, **argument parsing**, **data format**, ...

Rust is too Powerfull

Add trait to a **struct** by calling a **proc-macro**

```
#[derive(Default, Debug, MyCustomMacro)]
struct Shadok{}
```

Also for **comparison**, **equality**, **display**, **argument parsing**, **data format**, ...

Proc-macro can **change** the **compiler's intermediate representation** on the fly

Rust is too Powerfull

Add trait to a **struct** by calling a **proc-macro**

```
#[derive(Default, Debug, MyCustomMacro)]
struct Shadok{}
```

Also for **comparison**, **equality**, **display**, **argument parsing**, **data format**, ...

Proc-macro can **change** the **compiler's intermediate representation** on the fly

This **intermediate representation** is an **enum**

Rust is too Powerfull

Add trait to a **struct** by calling a **proc-macro**

```
#[derive(Default, Debug, MyCustomMacro)]
struct Shadok{}
```

Also for **comparison**, **equality**, **display**, **argument parsing**, **data format**, ...

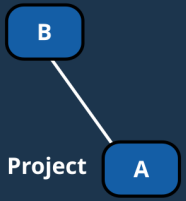
Proc-macro can **change** the **compiler's intermediate representation** on the fly

This **intermediate representation** is an **enum**

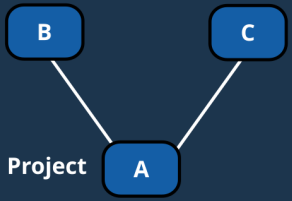
Only for **very specific use** :
- **Complexity quickly increases**
- A **proc-macro cannot** be used in the **crate** which **defines it**
  - So **0% coverage** for this project

Rust is too Powerfull

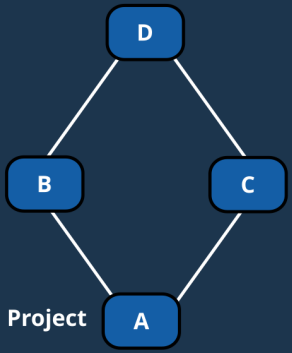
Rust is too Powerfull



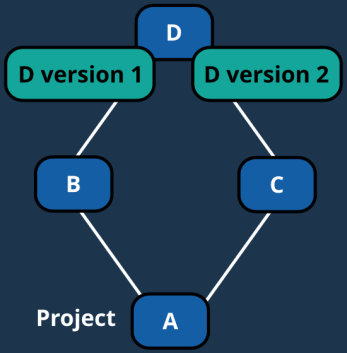
Rust is too Powerfull



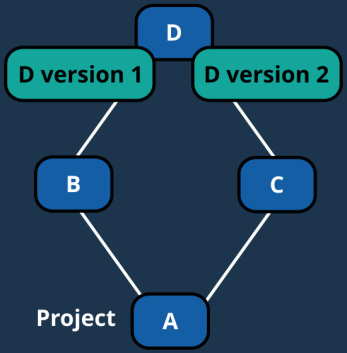
Rust is too Powerfull



Rust is too Powerfull



Rust is too Powerfull



cargo can **mix** different versions of different **crates** to compile

Rust crates are various : Serde

SERialization / **DE**serialization

Rust crates are various : Serde

SERialization / **DE**serialization

json, **toml**, **yml**, etc

```rust
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64,
}
```

Rust crates are various : Serde

SERialization / **DE**serialization

json, **toml**, **yml**, etc

```rust
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64,
}
```

Toml Example

```toml
name = "Shadoko"
age = 42
```

Rust crates are various : Serde

SERialization / **DE**serialization

json, **toml**, **yml**, etc

```rust
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64,
}
```

Load **toml** config :
toml::from_str(config_string)

Toml Example

```toml
name = "Shadoko"
age = 42
```

Rust crates are various : Serde

SERialization / **DE**serialization

json, **toml**, **yml**, etc

```rust
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64,
}
```

Load **toml** config :
toml::from_str(config_string)

Handles :
- **Parsing errors**
- **Missing Values**

Toml Example

```toml
name = "Shadoko"
age = 42
```

Rust crates are various : Serde

SERialization / **DE**serialization

json, **toml**, **yml**, etc

```rust
use serde::Deserialize;

#[derive(Debug, Deserialize)]
struct Shadok{
»       ///Name of the Shadok
»       name: String,
»       ///Age of the Shadok
»       age: u64,
}
```

Load **toml** config :
toml::from_str(config_string)

Handles :
- **Parsing errors**
- **Missing Values**

Toml Example

```toml
name = "Shadoko"
age = 42
```

- C++ **PhoenixFileGenerator**
  - **DataStream** : binary file/message/size (C++, Rust)
  - **TypeStream** : type names (C++, Rust)
  - **CheckStream** : value check (C++, Rust)
  - **ConfigStream** : config parsing (C++, Rust)

Rust crates are various : Clap

Arguments Parsing

Rust crates are various : Clap

Arguments Parsing

```rust
use clap::Parser;

#[derive(Debug, Parser)]
#[command(version, about)]
struct Shadok {
    ///Name of the Shadok
    #[arg(short, long)]
    name: String,
    ///Age of the Shadok
    #[arg(short, long, default_value = "42")]
    age: u64,
}
```

Rust crates are various : Clap

```rust
use clap::Parser;

#[derive(Debug, Parser)]
#[command(version, about)]
struct Shadok {
    ///Name of the Shadok
    #[arg(short, long)]
    name: String,
    ///Age of the Shadok
    #[arg(short, long, default_value = "42")]
    age: u64,
}
```

Arguments Parsing

Usage :

```rust
let shadok = Shadok::parse();
```

Rust crates are various : Clap

Arguments Parsing

```rust
use clap::Parser;

#[derive(Debug, Parser)]
#[command(version, about)]
struct Shadok {
    ///Name of the Shadok
    #[arg(short, long)]
    name: String,
    ///Age of the Shadok
    #[arg(short, long, default_value = "42")]
    age: u64,
}
```

Usage :

```rust
let shadok = Shadok::parse();
```

C++ PhoenixOptionParser

```cpp
#include "OptionParser.h"
OptionParser createOptionParser(){
    OptionParser parser(true, "1.0.0");
    parser.setExampleLongOption("program --name \"shadoko\" --age 42");
    parser.setExampleShortOption("program -n \"shadoko\" -a 42");
    parser.addOption("name", "n", OptionType::STRING, true, false, "Name of the Shadok");
    size_t age(42lu);
    parser.addOption("age", "a", age, "Age of the Shadok");
    return parser;
}
```

Rust crates are various : Clap

Arguments Parsing

Bash Completion

```rust
use clap::Parser;

#[derive(Debug, Parser)]
#[command(version, about)]
struct Shadok {
    ///Name of the Shadok
    #[arg(short, long)]
    name: String,
    ///Age of the Shadok
    #[arg(short, long, default_value = "42")]
    age: u64,
}
```

Usage :

```rust
let shadok = Shadok::parse();
```

C++ PhoenixOptionParser

Bash Completion

```cpp
#include "OptionParser.h"
OptionParser createOptionParser(){
    OptionParser parser(true, "1.0.0");
    parser.setExampleLongOption("program --name \"shadoko\" --age 42");
    parser.setExampleShortOption("program -n \"shadoko\" -a 42");
    parser.addOption("name", "n", OptionType::STRING, true, false, "Name of the Shadok");
    size_t age(42lu);
    parser.addOption("age", "a", age, "Age of the Shadok");
    return parser;
}
```

Rust crates are various : Clap

Laboratoire d'Annecy de Physique des Particules

Arguments Parsing

Bash Completion

```rust
use clap::Parser;

#[derive(Debug, Parser)]
#[command(version, about)]
struct Shadok {
    ///Name of the Shadok
    #[arg(short, long)]
    name: String,
    ///Age of the Shadok
    #[arg(short, long, default_value = "42")]
    age: u64,
}
```

Python ArgParse

```python
parser = argparse.ArgumentParser(description="Program description")
parser.add_argument("--name", "-n", action="store",
        type=str, help="Name of the shadok", required=True)
parser.add_argument("--age", "-a", action="store",
        type=int, help="Age of the Shadok", default=42)
```

Usage :

```rust
let shadok = Shadok::parse();
```

C++ PhoenixOptionParser

Bash Completion

```cpp
#include "OptionParser.h"
OptionParser createOptionParser(){
        OptionParser parser(true, "1.0.0");
        parser.setExampleLongOption("program --name \"shadoko\" --age 42");
        parser.setExampleShortOption("program -n \"shadoko\" -a 42");
        parser.addOption("name", "n", OptionType::STRING, true, false, "Name of the Shadok");
        size_t age(42lu);
        parser.addOption("age", "a", age, "Age of the Shadok");
        return parser;
}
```

Rust is Meh

Rust is Meh

The **most anoying** compiler ever !

Rust is Meh

The **most anoying** compiler ever !

Conventions are rules :
- Namming Convention :
  - Variables / Parameters / Functions / Methods / Source Files / Directories : **snake_case**
  - Structs / Enums / Traits : **CamelCase**
  - Constants : **RIHNO_CASE**

Rust is Meh

The **most anoying** compiler ever !

Conventions are rules :
- Namming Convention :
  - Variables / Parameters / Functions / Methods / Source Files / Directories : **snake_case**
  - Structs / Enums / Traits : **CamelCase**
  - Constants : **RIHNO_CASE**

Technically you can change convention but you will have **1000s of warnings** !

Rust is Meh

The **most anoying** compiler ever !

Conventions are rules :
- Namming Convention :
  - Variables / Parameters / Functions / Methods / Source Files / Directories : **snake_case**
  - Structs / Enums / Traits : **CamelCase**
  - Constants : **RIHNO_CASE**

Technically you can change convention but you will have **1000s of warnings** !

Cargo's convention :
- sources are in **src** dir
  - main is in **main.rs**
  - library is in **libs.rs**
- unit tests are in **tests** dir

If you **do not respect cargo**'s convention, **nothing will work**

Rust is Meh

The **most anoying** compiler ever !

Conventions are rules :
- Namming Convention :
  - Variables / Parameters / Functions / Methods / Source Files / Directories : **snake_case**
  - Structs / Enums / Traits : **CamelCase**
  - Constants : **RIHNO_CASE**

Technically you can change convention but you will have **1000s of warnings** !

Cargo's convention :
- sources are in **src** dir
  - main is in **main.rs**
  - library is in **libs.rs**
- unit tests are in **tests** dir

If you **do not respect cargo**'s convention, **nothing will work**

mod keyword -> **cmake add_subdirectory**

Rust is Meh

The **most anoying** compiler ever !

Conventions are rules :
- Namming Convention :
  - Variables / Parameters / Functions / Methods / Source Files / Directories : **snake_case**
  - Structs / Enums / Traits : **CamelCase**
  - Constants : **RIHNO_CASE**

Technically you can change convention but you will have **1000s of warnings** !

Cargo's convention :
- sources are in **src** dir
  - main is in **main.rs**
  - library is in **libs.rs**
- unit tests are in **tests** dir

If you **do not respect cargo**'s convention, **nothing will work**

mod keyword -> **cmake add_subdirectory**

Mix of **build instructions inside sources**

Rust is Meh

The **return** keyword **is not** mandatory

Rust is Meh

The **return** keyword **is not** mandatory

```
fn function() -> u32{
»       42
}
```

Rust is Meh

The **return** keyword **is not** mandatory

```
fn function() -> u32{
»       42
}
```

```
fn function() -> u32{
»       return 42;
}
```

Rust is Meh

The **return** keyword **is not** mandatory

```
fn function() -> u32{
»       42
}
```



```
fn function() -> u32{
»       return 42;
}
```

Nice, because you cannot search anymore for a **return** keyword, but for a missing **;** because of matlab !!!

Rust is Meh

The **return** keyword **is not** mandatory

```
fn function() -> u32{
»       42
}
```



```
fn function() -> u32{
»       return 42;
}
```

Nice, because you cannot search anymore for a **return** keyword, but for a missing **;** because of matlab !!!

Very confusing on complex functions

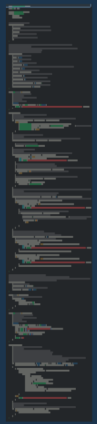
Rust is Meh

No Header

Rust is Meh

No Header

Need to **dive into** a **full source file** to get **exposed structs/traits** and **functions**



Rust is Meh

No Header

Need to **dive into** a **full source file** to get **exposed structs/traits** and **functions**

Searching **pub** keyword

Rust is Meh

No Header

Need to **dive into** a **full source file** to get **exposed structs/traits** and **functions**

Searching **pub** keyword

Possibility to **refactor** :
- only **public** members source
- only **private** members source

Rust is Meh

No Header

Need to **dive into** a **full source file** to get **exposed structs/traits** and **functions**

Searching **pub** keyword

Possibility to **refactor** :
- only **public** members source
- only **private** members source

Prelude module to **expose** essentials of the crate

Rust is Meh

No Header

Need to **dive into** a **full source file** to get **exposed structs/traits** and **functions**

Searching **pub** keyword

Possibility to **refactor** :
- only **public** members source
- only **private** members source

Prelude module to **expose** essentials of the crate

=> **Solved** with a **good IDE**

Rust is NOT C++

Rust is NOT C++

```
int var = 0;
```

Variables

Rust is NOT C++

```int var = 0;```

Variables

```let mut var: u32 = 0;```

Rust is NOT C++

```cpp
int var = 0;
```

Variables

```rust
let mut var: u32 = 0;
```

```cpp
///@brief Shadok
struct Shadok{
»       ///Age of the shadok
»       size_t age;
»       ///Name of the Shadok
»       std::string name;
};
```

Struct

Rust is NOT C++

Variables

```cpp
int var = 0;
```

```rust
let mut var: u32 = 0;
```

Struct

```cpp
///@brief Shadok
struct Shadok{
»       ///Age of the shadok
»       size_t age;
»       ///Name of the Shadok
»       std::string name;
};
```

```rust
///Shadok
struct Shadok{
»       ///Age of the shadok
»       age: u64,
»       ///Name of the Shadok
»       name: String
}
```

Rust is NOT C++

```
int var = 0;
```

Variables

```
let mut var: u32 = 0;
```

```
///@brief Shadok
struct Shadok{
»       ///Age of the shadok
»       size_t age;
»       ///Name of the Shadok
»       std::string name;
};
```

Struct

```
///Shadok
struct Shadok{
»       ///Age of the shadok
»       age: u64,
»       ///Name of the Shadok
»       name: String
}
```

You **cannot copy-paste C++ from/to Rust**

Rust is NOT C++

```int var = 0;```

Variables

```let mut var: u32 = 0;```

```
///@brief Shadok
struct Shadok{
»       ///Age of the shadok
»       size_t age;
»       ///Name of the Shadok
»       std::string name;
};
```

Struct

```
///Shadok
struct Shadok{
»       ///Age of the shadok
»       age: u64,
»       ///Name of the Shadok
»       name: String
}
```

You **cannot copy-paste C++ from/to Rust**

Avoid a lot of mistakes

Rust is Pretentious

People present **rust-doc** as a better tool that **doxygen** but :

- doc

doxygen

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...

doxygen

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**
- **no documentation example** generated by **Cargo**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**
- **no documentation example** generated by **Cargo**
- code **documentation** is just **markdown**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**
- **no documentation example** generated by **Cargo**
- code **documentation** is just **markdown** with **nice urls** to link **crates content**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**
- **no documentation example** generated by **Cargo**
- code **documentation** is just **markdown** with **nice urls** to link **crates content**

- no exception but **Option<T>** and **Result<T, E>** instead
  - you are **forced** to **handle correctly the returned types**

Rust is Pretentious

- doc

People present **rust-doc** as a better tool that **doxygen** but :

doxygen

- **rust-doc** does not do **1/100** of what **doxygen** does
  - **no** call graph, callee, caller, include, trait use, ...
- **cryptic error** if you document your code with tabulation because why not
- creating doc **can crash** your **CI** (**doc tests** run during **unit tests**)
- **not even a main page index.html** generated for the doc
  - you have to **do the redirection yourself !**
- documentation of **rust-doc** is realy bad
  - good luck to find how to **format doc properly**
- **no documentation example** generated by **Cargo**
- code **documentation** is just **markdown** with **nice urls** to link **crates content**

- no exception but **Option<T>** and **Result<T, E>** instead
  - you are **forced** to **handle correctly the returned types**
  - a lot of developpers use **'?'** to foreward the error handling to the caller function
    - **lazy** but very **usefull** in **I/O implementation**

Rust is a Liar

Rust is a Liar

- **Rust** is **safe** :

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

  But **mdbook**'s (rust doc book compiler) **dependencies compiles** with **autotools**

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

  But **mdbook**'s (rust doc book compiler) **dependencies compiles** with **autotools**

- **Rust Developers always deal with errors well**

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

  But **mdbook**'s (rust doc book compiler) **dependencies compiles** with **autotools**

- **Rust Developers always deal with errors well**

  - Practically all **examples** in the **Rust Book do not have error handling** but only **unwrap** to skip it

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

  But **mdbook**'s (rust doc book compiler) **dependencies compiles** with **autotools**

- **Rust Developers always deal with errors well**

  - Practically all **examples** in the **Rust Book do not have error handling** but only **unwrap** to skip it

- **Cargo** eases packages **sharing** :

Rust is a Liar

- **Rust** is **safe** :

  But a lot of **crates** are **wrappers of C**, **C++** or **js libraries** because nobody wants to develop them again

  - **Socket**
  - **ZMQ**
  - **HDF5**
  - **...**

- **Cargo** is the **only builder** you need for **Rust** :

  But **mdbook**'s (rust doc book compiler) **dependencies compiles** with **autotools**

- **Rust Developers always deal with errors well**

  - Practically all **examples** in the **Rust Book do not have error handling** but only **unwrap** to skip it

- **Cargo** eases packages **sharing** :

  - But some **projects** already have **way to much dependencies** : **191** for **cargo-tarpaulin** (for coverage)
    - To much foreign dependencies -> **big technical dept**
    - And **very long compilation time**

Rust is Horrible

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

```
let out: String = format!("{} some string {} other string {}", s1, s1, s2);
```

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

```
let out: String = format!("{} some string {} other string {}", s1, s1, s2);
```

Close to **sprintf C** concatenation

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

```
let out: String = format!("{} some string {} other string {}", s1, s1, s2);
```

Close to **sprintf C** concatenation

- Try to **debug** a **unit test** with **cargo test**
  - You have to use **ugly environment variables** to activate trace and you **still not have** the **verbose output** of your test !!!!
  - No color, no highlighting and very confusing output

Rust is Horrible

- Rust is **explicit** => so no **= operator** for **String** and **Vec<T>**
  - But **clone()**
- **Borrowing** can be a **nightmare**
  - **Unique** usage for **= operator**
  - **Reference borrowing** similar to **stolen ownership**

```
std::string out = s1 + "some string" + s2 + " other string " + s2;
```

```
let out: String = s1 + " some string " + &s2 + " other string " + &s2;
```

First string **cannot be** a **reference**

```
let out: String = s1.clone() + " some string " + &s1 + " other string " + &s2;
```

```
let out: String = format!("{} some string {} other string {}", s1, s1, s2);
```

Close to **sprintf C** concatenation

- Try to **debug** a **unit test** with **cargo test**
  - You have to use **ugly environment variables** to activate trace and you **still not have** the **verbose output** of your test !!!!
  - No color, no highlighting and very confusing output
- **Coverage report** is **ugly**

Rust is Horrible

Error stack

Rust is Horrible

Error stack

Ugly **RUST_BACKTRACE=1** to get error stack

```
pierre@pierre-Precision-7670:~/projects/PHOENIX2/RustyPhoenixGenerator$ RUST_BACKTRACE=1 ./target/debug/rusty_phoenix_generator -i tests/generator.toml -o generated

thread 'main' (96637) panicked at src/main.rs:33:21:
rusty_phoenix_generator: crate creation error : No such file or directory (os error 2)
stack backtrace:
   0: __rustc::rust_begin_unwind
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
   1: core::panicking::panic_fmt
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
   2: rusty_phoenix_generator::main
             at ./src/main.rs:33:15
   3: core::ops::function::FnOnce::call_once
             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
```

Rust is Horrible

Error stack

Ugly **RUST_BACKTRACE=1** to get error stack

```
pierre@pierre-Precision-7670:~/projects/PHOENIX2/RustyPhoenixGenerator$ RUST_BACKTRACE=1 ./target/debug/rusty_phoenix_generator -i tests/generator.toml -o generated

thread 'main' (96637) panicked at src/main.rs:33:21:
rusty_phoenix_generator: crate creation error "No such file or directory (os error 2)
stack backtrace:
   0: __rustc::rust_begin_unwind
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
   1: core::panicking::panic_fmt
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
   2: rusty_phoenix_generator::main
             at ./src/main.rs:33:15
   3: core::ops::function::FnOnce::call_once
             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
```

Rust is Horrible

Error stack

Ugly **RUST_BACKTRACE=1** to get error stack

```
pierre@pierre-Precision-7670:~/projects/PHOENIX2/RustyPhoenixGenerator$ RUST_BACKTRACE=1 ./target/debug/rusty_phoenix_generator -i tests/generator.toml -o generated

thread 'main' (96637) panicked at src/main.rs:33:21:
rusty_phoenix_generator: crate creation error "No such file or directory" (os error 2)
stack backtrace:
   0: __rustc::rust_begin_unwind
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
   1: core::panicking::panic_fmt
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
   2: rusty_phoenix_generator::main
             at ./src/main.rs:33:15
   3: core::ops::function::FnOnce::call_once
             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
```

OK but which path ???!!!!

Rust is Horrible

Error stack

Ugly **RUST_BACKTRACE=1** to get error stack

```
pierre@pierre-Precision-7670:~/projects/PHOENIX2/RustyPhoenixGenerator$ RUST_BACKTRACE=1 ./target/debug/rusty_phoenix_generator -i tests/generator.toml -o generated

thread 'main' (96637) panicked at src/main.rs:33:21:
rusty_phoenix_generator: crate creation error "No such file or directory (os error 2)"
stack backtrace:
   0: __rustc::rust_begin_unwind
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
   1: core::panicking::panic_fmt
             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
   2: rusty_phoenix_generator::main
             at ./src/main.rs:33:15
   3: core::ops::function::FnOnce::call_once
             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
```

OK but which path ???!!!!

Rust is Horrible

```
pierre@pierre-Precision-7670

thread 'main' (96637) panick
rusty_phoenix_generator: cra
stack backtrace:
   0: __rustc::rust_begin_un
             at /rustc/ed61e
   1: core::panicking::panic
             at /rustc/ed61e
   2: rusty_phoenix_generato
             at ./src/main.r
   3: core::ops::function::F
             at /home/pierre
note: Some details are omitt
```

```
   8:     0x57a55a5700f2 - std::sys::backtrace::BacktraceLock::print::hf67a46baa62198e0
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:42:9
   9:     0x57a55a571f5c - std::panicking::default_hook::{{closure}}::h391aa815d5e47ec8
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:301:27
  10:     0x57a55a571db6 - std::panicking::default_hook::hd6fdcf2489bb807d
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:328:9
  11:     0x57a55a5725e5 - std::panicking::panic_with_hook::h185ddfb86bf14d73
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:834:13
  12:     0x57a55a57247a - std::panicking::panic_handler::{{closure}}::had89ddd01b6112c9
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:707:13
  13:     0x57a55a570229 - std::sys::backtrace::__rust_end_short_backtrace::h5d0fc36eef7265ea
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:174:18
  14:     0x57a55a55dc2d - __rustc[eb8946e36839644a]::rust_begin_unwind
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
  15:     0x57a55a5a7880 - core::panicking::panic_fmt::h92c8e5abe71dd8d1
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
  16:     0x57a55a4b5417 - rusty_phoenix_generator::main::hd164d4b9c84e7c94
                               at /home/pierre/projects/PHOENIX2/RustyPhoenixGenerator/src/main.rs:33:15
  17:     0x57a55a4bbae3 - core::ops::function::FnOnce::call_once::h39f21887f05d4088
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
  18:     0x57a55a4bbae3 - std::sys::backtrace::__rust_begin_short_backtrace::h277653590cab7736
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/sys/backtrace.rs:158:18
  19:     0x57a55a4bb4d9 - std::rt::lang_start::{{closure}}::h9d12810b45177177
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/rt.rs:206:18
  20:     0x57a55a56b730 - core::ops::function::impls::<impl core::ops::function::FnOnce<A> for &F>::call_once::h6bd8af1873a4fcba
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/ops/function.rs:287:21
  21:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h8502df911ab1d854
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
  22:     0x57a55a56b730 - std::panicking::catch_unwind::hf8590bd54a09e620
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
  23:     0x57a55a56b730 - std::panic::catch_unwind::hef0d4e02e13f8ea6
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
  24:     0x57a55a56b730 - std::rt::lang_start_internal::{{closure}}::h813a94d4aa317bce
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:175:24
  25:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h0a1ac6ae8aee102d
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
  26:     0x57a55a56b730 - std::panicking::catch_unwind::hddd02d0cb4cfa799
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
  27:     0x57a55a56b730 - std::panic::catch_unwind::hbe5a97997afb140b
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
  28:     0x57a55a56b730 - std::rt::lang_start_internal::h6ba36b077a531782
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:171:5
  29:     0x57a55a4b647c - main
  30:     0x714030e2a1ca - __libc_start_call_main
                               at ./csu/../sysdeps/nptl/libc_start_call_main.h:58:16
  31:     0x714030e2a28b - __libc_start_main_impl
                               at ./csu/../csu/libc-start.c:360:3
  32:     0x57a55a4ac6ce - _start
  33:                0x0 - <unknown>
```

```
/generator.toml -o generated
```

!!

Rust is Horrible

```
pierre@pierre-Precision-7670

thread 'main' (96637) panick
rusty_phoenix_generator: cra
stack backtrace:
   0: __rustc::rust_begin_un
             at /rustc/ed61e
   1: core::panicking::panic
             at /rustc/ed61e
   2: rusty_phoenix_generato
             at ./src/main.r
   3: core::ops::function::F
             at /home/pierre
note: Some details are omitt
```

```
 8:     0x57a55a5700f2 - std::sys::backtrace::BacktraceLock::print::hf67a46baa62198e
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:42:9
 9:     0x57a55a571f5c - std::panicking::default_hook::{{closure}}::h391aa815d5e47ec8
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:301:27
10:     0x57a55a571db6 - std::panicking::default_hook::hd6fdcf2489bb807d
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:328:9
11:     0x57a55a5725e5 - std::panicking::panic_with_hook::h185ddfb86bf14d73
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:834:13
12:     0x57a55a57247a - std::panicking::panic_handler::{{closure}}::had89ddd01b6112c9
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:707:13
13:     0x57a55a570229 - std::sys::backtrace::__rust_end_short_backtrace::h5d0fc36eef7265ea
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:174:18
14:     0x57a55a55dc2d - __rustc[eb8946e36839644a]::rust_begin_unwind
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
15:     0x57a55a5a7880 - core::panicking::panic_fmt::h92c8e5abe71dd8d1
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
16:     0x57a55a4b5417 - rusty_phoenix_generator::main::hd164d4b9c84e7c94
                           at /home/pierre/projects/PHOENIX2/RustyPhoenixGenerator/src/main.rs:33:15
17:     0x57a55a4bbae3 - core::ops::function::FnOnce::call_once::h39f21887f05d4088
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
18:     0x57a55a4bbae3 - std::sys::backtrace::__rust_begin_short_backtrace::h277653590cab7736
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/sys/backtrace.rs:158:18
19:     0x57a55a4bb4d9 - std::rt::lang_start::{{closure}}::h9d12810b45177177
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/rt.rs:206:18
20:     0x57a55a56b730 - core::ops::function::impls::<impl core::ops::function::FnOnce<A> for &F>::call_once::h6bd8af1873a4fcba
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/ops/function.rs:287:21
21:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h8502df911ab1d854
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
22:     0x57a55a56b730 - std::panicking::catch_unwind::hf8590bd54a09e620
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
23:     0x57a55a56b730 - std::panic::catch_unwind::hef0d4e02e13f8ea6
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
24:     0x57a55a56b730 - std::rt::lang_start_internal::{{closure}}::h813a94d4aa317bce
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:175:24
25:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h0a1ac6ae8aee102d
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
26:     0x57a55a56b730 - std::panicking::catch_unwind::hddd02d0cb4cfa799
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
27:     0x57a55a56b730 - std::panic::catch_unwind::hbe5a97997afb140b
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
28:     0x57a55a56b730 - std::rt::lang_start_internal::h6ba36b077a531782
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:171:5
29:     0x57a55a4b647c - main
30:     0x714030e2a1ca - __libc_start_call_main
                           at ./csu/../sysdeps/nptl/libc_start_call_main.h:58:16
31:     0x714030e2a28b - __libc_start_main_impl
                           at ./csu/../csu/libc-start.c:360:3
32:     0x57a55a4ac6ce - _start
33:                0x0 - <unknown>
```

Not your Buisness

/generator.toml -o generated

!!

Rust is Horrible

Background terminal (partly covered):

```
pierre@pierre-Precision-7670                                   /generator.toml -o generated

thread 'main' (96637) panick
rusty_phoenix_generator: cra
stack backtrace:
   0: __rustc::rust_begin_un
             at /rustc/ed61e
   1: core::panicking::panic
             at /rustc/ed61e
   2: rusty_phoenix_generato
             at ./src/main.r
   3: core::ops::function::F
             at /home/pierre
note: Some details are omitt
```

Full backtrace:

```
 8:     0x57a55a5700f2 - std::sys::backtrace::BacktraceLock::print::hf67a46baa621998e
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:42:9
 9:     0x57a55a571f5c - std::panicking::default_hook::{{closure}}::h391aa815d5e47ec8
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:301:27
10:     0x57a55a571db6 - std::panicking::default_hook::hd6fdcf2489bb807d
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:328:9
11:     0x57a55a5725e5 - std::panicking::panic_with_hook::h185ddfb86bf14d73
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:834:13
12:     0x57a55a57247a - std::panicking::panic_handler::{{closure}}::had89ddd01b6112c9
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:707:13
13:     0x57a55a570229 - std::sys::backtrace::__rust_end_short_backtrace::h5d0fc36eef7265ea
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:174:18
14:     0x57a55a55dc2d - __rustc[eb8946e36839644a]::rust_begin_unwind
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
15:     0x57a55a5a7880 - core::panicking::panic_fmt::h92c8e5abe71dd8d1
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
16:     0x57a55a4b5417 - rusty_phoenix_generator::main::hd164d4b9c84e7c94
                             at /home/pierre/projects/PHOENIX2/RustyPhoenixGenerator/src/main.rs:33:15
17:     0x57a55a4bbae3 - core::ops::function::FnOnce::call_once::h39f21887f05d4088
                             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
18:     0x57a55a4bbae3 - std::sys::backtrace::__rust_begin_short_backtrace::h277653590cab7736
                             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/sys/backtrace.rs:158:18
19:     0x57a55a4bb4d9 - std::rt::lang_start::{{closure}}::h9d12810b45177177
                             at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/rt.rs:206:18
20:     0x57a55a56b730 - core::ops::function::impls::<impl core::ops::function::FnOnce<A> for &F>::call_once::h6bd8af1873a4fcba
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/ops/function.rs:287:21
21:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h8502df911ab1d854
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
22:     0x57a55a56b730 - std::panicking::catch_unwind::hf8590bd54a09e620
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
23:     0x57a55a56b730 - std::panic::catch_unwind::hef0d4e02e13f8ea6
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
24:     0x57a55a56b730 - std::rt::lang_start_internal::{{closure}}::h013a94d4ae317bce
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:175:24
25:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h0a1ac6ae8aee102d
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
26:     0x57a55a56b730 - std::panicking::catch_unwind::hddd02d0cb4cfa799
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
27:     0x57a55a56b730 - std::panic::catch_unwind::hbe5a97997afb148b
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
28:     0x57a55a56b730 - std::rt::lang_start_internal::h6ba36b077a531782
                             at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:171:5
29:     0x57a55a4b647c - main
30:     0x714030e2a1ca - __libc_start_call_main
                             at ./csu/../sysdeps/nptl/libc_start_call_main.h:58:16
31:     0x714030e2a28b - __libc_start_main_impl
                             at ./csu/../csu/libc-start.c:360:3
32:     0x57a55a4ac6ce - _start
33:                0x0 - <unknown>
```

Not your Buisness (frames 8–15)

!! (frame 16)

Not your Buisness (frames 17–33)

Rust is Horrible

```
pierre@pierre-Precision-7670

thread 'main' (96637) panick
rusty_phoenix_generator: cra
stack backtrace:
   0: __rustc::rust_begin_un
             at /rustc/ed61e
   1: core::panicking::panic
             at /rustc/ed61e
   2: rusty_phoenix_generato
             at ./src/main.r
   3: core::ops::function::F
             at /home/pierre
note: Some details are omitt
```

```
   8:     0x57a55a5700f2 - std::sys::backtrace::BacktraceLock::print::hf67a46baa62198e0
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:42:9
   9:     0x57a55a571f5c - std::panicking::default_hook::{{closure}}::h391aa815d5e47ec8
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:301:27
  10:     0x57a55a571db6 - std::panicking::default_hook::hd6fdcf2489bb807d
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:328:9
  11:     0x57a55a5725e5 - std::panicking::panic_with_hook::h185ddfb86bf14d73
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:834:13
  12:     0x57a55a57247a - std::panicking::panic_handler::{{closure}}::had89ddd01b6112c9
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:707:13
  13:     0x57a55a570229 - std::sys::backtrace::__rust_end_short_backtrace::h5d0fc36eef7265ea
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:174:18
  14:     0x57a55a55dc2d - __rustc[eb8946e36839644a]::rust_begin_unwind
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
  15:     0x57a55a5a7880 - core::panicking::panic_fmt::h92c8e5abe71dd8d1
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
  16:     0x57a55a4b5417 - rusty_phoenix_generator::main::hd164dd9c84e7c94
                               at /home/pierre/projects/PHOENIX2/RustyPhoenixGenerator/src/main.rs:33:15
  17:     0x57a55a4bbae3 - core::ops::function::FnOnce::call_once::h39f21887f05d4088
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
  18:     0x57a55a4bbae3 - std::sys::backtrace::__rust_begin_short_backtrace::h277653590cab7736
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/sys/backtrace.rs:158:18
  19:     0x57a55a4bb4d9 - std::rt::lang_start::{{closure}}::h9d128104b45177177
                               at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/rt.rs:206:18
  20:     0x57a55a56b730 - core::ops::function::impls::<impl core::ops::function::FnOnce<A> for &F>::call_once::h6bd8af1873a4fcba
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/ops/function.rs:287:21
  21:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h8502df911ab1d854
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
  22:     0x57a55a56b730 - std::panicking::catch_unwind::hf8590bd54a09e620
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
  23:     0x57a55a56b730 - std::panic::catch_unwind::hef0d4e02e13f8ea6
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
  24:     0x57a55a56b730 - std::rt::lang_start_internal::{{closure}}::h813a94d4aa317bce
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:175:24
  25:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h0a1ac6ae8aee102d
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
  26:     0x57a55a56b730 - std::panicking::catch_unwind::hddd02d0cb4cfa799
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
  27:     0x57a55a56b730 - std::panic::catch_unwind::hbe5a97997afb148b
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
  28:     0x57a55a56b730 - std::rt::lang_start_internal::h6ba36b077a531782
                               at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:171:5
  29:     0x57a55a4b6647c - main
  30:     0x714030e2a1ca - __libc_start_call_main
                               at ./csu/../sysdeps/nptl/libc_start_call_main.h:58:16
  31:     0x714030e2a28b - __libc_start_main_impl
                               at ./csu/../csu/libc-start.c:360:3
  32:     0x57a55a4ac6ce - _start
  33:                0x0 - <unknown>
```

/generator.toml -o generated

Not your Buisness

!!

Your buisness

Not your Buisness

Rust is Horrible

```
pierre@pierre-Precision-7670

thread 'main' (96637) panick
rusty_phoenix_generator: cra
stack backtrace:
   0: __rustc::rust_begin_un
             at /rustc/ed61e
   1: core::panicking::panic
             at /rustc/ed61e
   2: rusty_phoenix_generato
             at ./src/main.r
   3: core::ops::function::F
             at /home/pierre
note: Some details are omitt
```

```
 8:     0x57a55a5700f2 - std::sys::backtrace::BacktraceLock::print::hf67a46baa62198e0
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:42:9
 9:     0x57a55a571f5c - std::panicking::default_hook::{{closure}}::h391aa815d5e47ec8
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:301:27
10:     0x57a55a571db6 - std::panicking::default_hook::hd6fdcf2489bb807d
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:328:9
11:     0x57a55a5725e5 - std::panicking::panic_with_hook::h185ddfb86bf14d73
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:834:13
12:     0x57a55a57247a - std::panicking::panic_handler::{{closure}}::had89ddd01b6112c9
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:707:13
13:     0x57a55a570229 - std::sys::backtrace::__rust_end_short_backtrace::h5d0fc36eef7265ea
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/sys/backtrace.rs:174:18
14:     0x57a55a55dc2d - __rustc[eb8946e36839644a]::rust_begin_unwind
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:698:5
15:     0x57a55a5a7880 - core::panicking::panic_fmt::h92c8e5abe71dd8d1
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/panicking.rs:75:14
16:     0x57a55a4b5417 - rusty_phoenix_generator::main::hd164d4b9c84e7c94
                           at /home/pierre/projects/PHOENIX2/RustyPhoenixGenerator/src/main.rs:33:15
17:     0x57a55a4bbae3 - core::ops::function::FnOnce::call_once::h39f21887f05d4088
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/core/src/ops/function.rs:250:5
18:     0x57a55a4bbae3 - std::sys::backtrace::__rust_begin_short_backtrace::h277653590cab7736
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/sys/backtrace.rs:158:18
19:     0x57a55a4bb4d9 - std::rt::lang_start::{{closure}}::h9d128104b4577177
                           at /home/pierre/projects/RUST/PixiRust/.pixi/envs/default/lib/rustlib/src/rust/library/std/src/rt.rs:206:18
20:     0x57a55a56b730 - core::ops::function::impls::<impl core::ops::function::FnOnce<A> for &F>::call_once::h6bd8af1873a4fcba
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/core/src/ops/function.rs:287:21
21:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h8502df911ab1d854
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
22:     0x57a55a56b730 - std::panicking::catch_unwind::hf8590bd54a09e620
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
23:     0x57a55a56b730 - std::panic::catch_unwind::hef0d4e02e13f8ea6
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
24:     0x57a55a56b730 - std::rt::lang_start_internal::{{closure}}::h813a94d4aa317bce
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:175:24
25:     0x57a55a56b730 - std::panicking::catch_unwind::do_call::h0a1ac6ae8aee102d
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:590:40
26:     0x57a55a56b730 - std::panicking::catch_unwind::hddd02d0cb4cfa799
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panicking.rs:553:19
27:     0x57a55a56b730 - std::panic::catch_unwind::hbe5a97997afb148b
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/panic.rs:359:14
28:     0x57a55a56b730 - std::rt::lang_start_internal::h6ba36b077a531782
                           at /rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb/library/std/src/rt.rs:171:5
29:     0x57a55a4b647c - main
30:     0x714030e2a1ca - __libc_start_call_main
                           at ./csu/../sysdeps/nptl/libc_start_call_main.h:58:16
31:     0x714030e2a28b - __libc_start_main_impl
                           at ./csu/../csu/libc-start.c:360:3
32:     0x57a55a4ac6ce - _start
33:                0x0 - <unknown>
```

/generator.toml -o generated

Not your Buisness

!!

Your buisness

Not your Buisness

Not reversed Stack as Python but **still confusing** enough

Rust redefines Stupidity

Rust redefines Stupidity

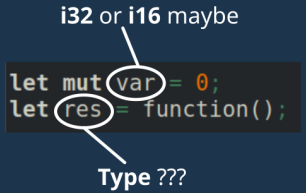
```
let mut var = 0;
let res = function();
```

Rust redefines Stupidity

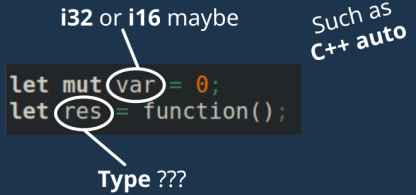
i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Rust redefines Stupidity



Rust redefines Stupidity



Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as
C++ auto

By the way

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

```
fn function() -> Shadok{
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
»       Shadok{
»       »       name: String::from("Shadoko"),
»       »       age: 42
»       }
}
```

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Using **Variable**

```
let mut var = 0;
function(&mut var);
```

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Using **Variable**

```
let mut var = 0;
function(&mut var);
```

As we saw before **Rust is explicit**

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Using **Variable**

```
let mut var = 0;
function(&mut var);
```

As we saw before **Rust is explicit**

Using **Parameter**

```
fn function(var: &mut u32){
    other_function(var);
}
```

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Using **Variable**

```
let mut var = 0;
function(&mut var);
```

As we saw before **Rust is explicit**

Using **Parameter**

```
fn function(var: &mut u32){
    other_function(var);
}
```

But **not** in **this** case ??!!

Rust redefines Stupidity

i32 or **i16** maybe

```
let mut var = 0;
let res = function();
```

Type ???

Such as **C++ auto**

By the way

Of course a **Shadok**

```
fn function() -> Shadok
    Shadok{
        name: String::from("Shadoko"),
        age: 42
    }
}
```

Using **parameters** or **variable** is **not coherent**

Using **Variable**

```
let mut var = 0;
function(&mut var);
```

As we saw before **Rust is explicit**

Using **Parameter**

```
fn function(var: &mut u32){
    other_function(var);
}
```

But **not** in **this** case ??!!

Does **other_function modifies var** or not ?

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

NUMA Node 1

NUMA Node 2

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

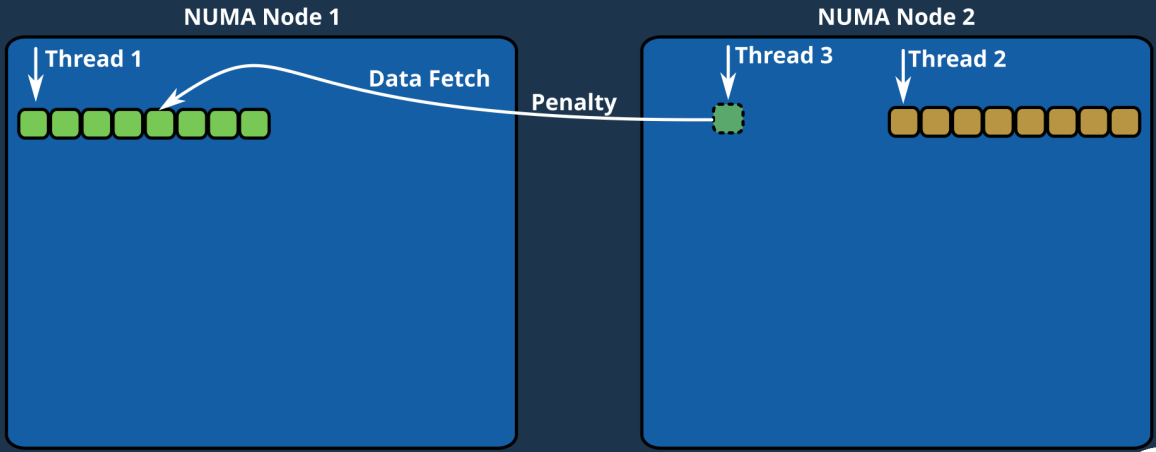
Globaly a **good point** to **prevent mistakes**

Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

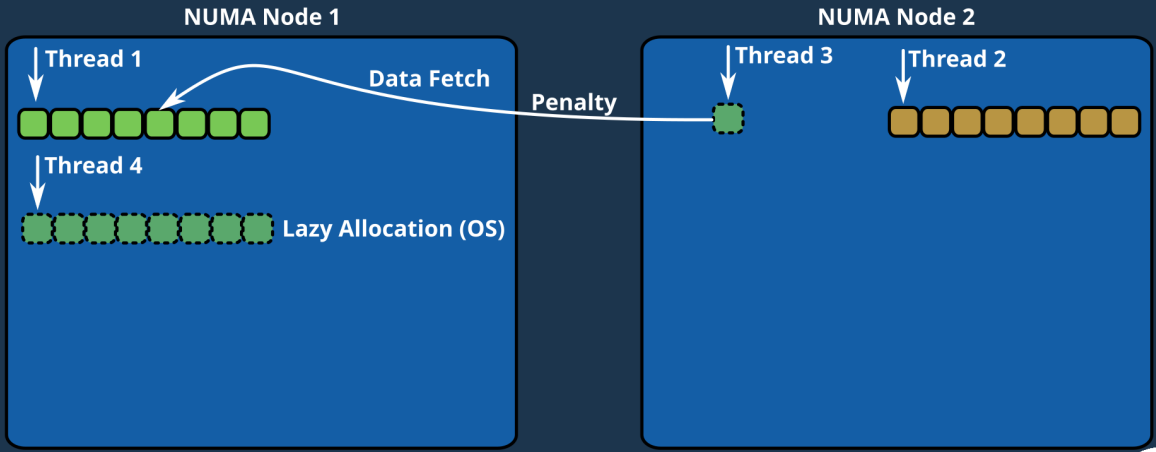


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

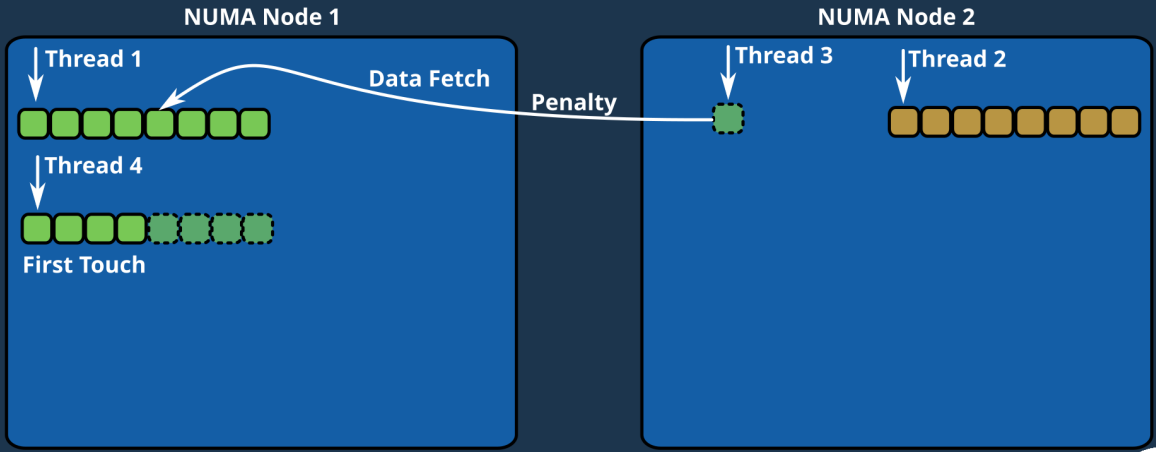


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

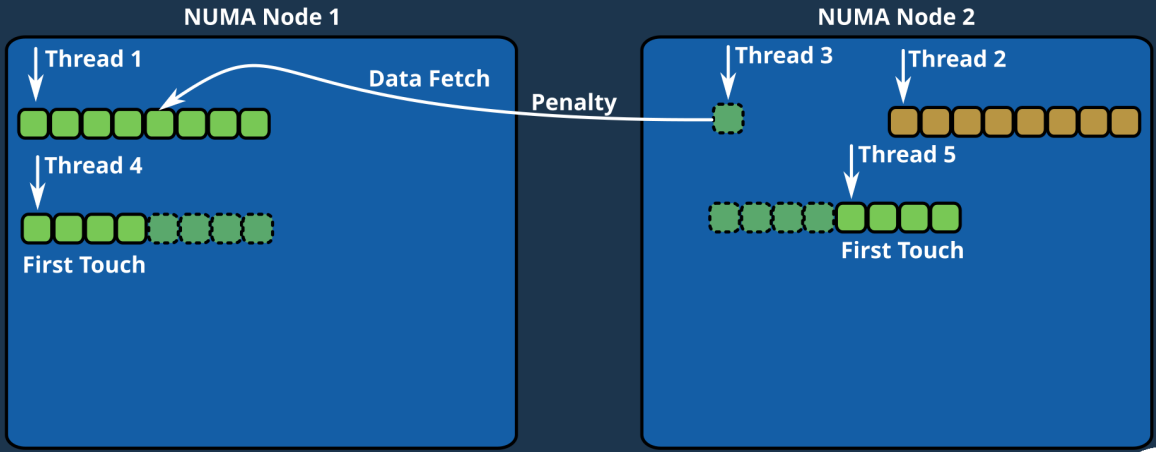
Globaly a **good point** to **prevent mistakes**



Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables** Globaly a **good point** to **prevent mistakes**

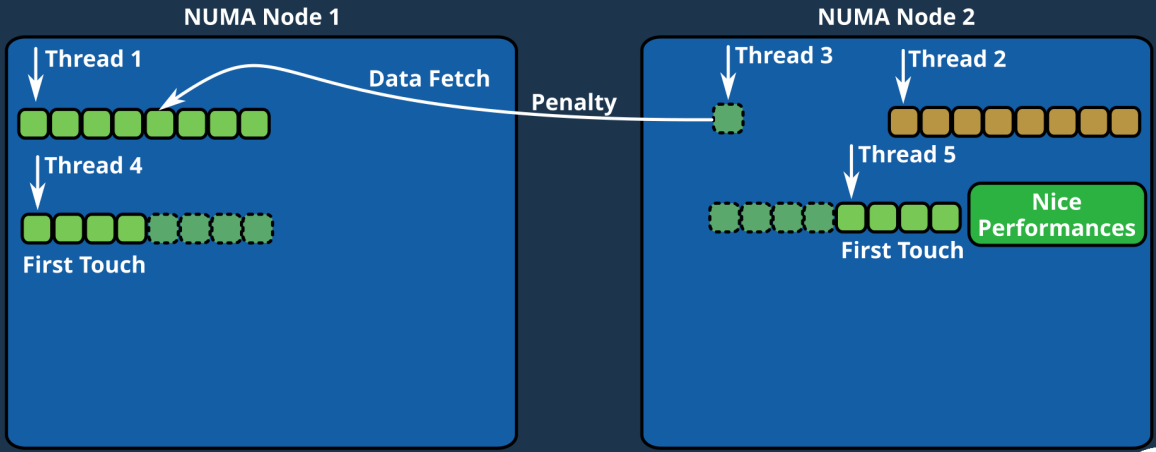


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

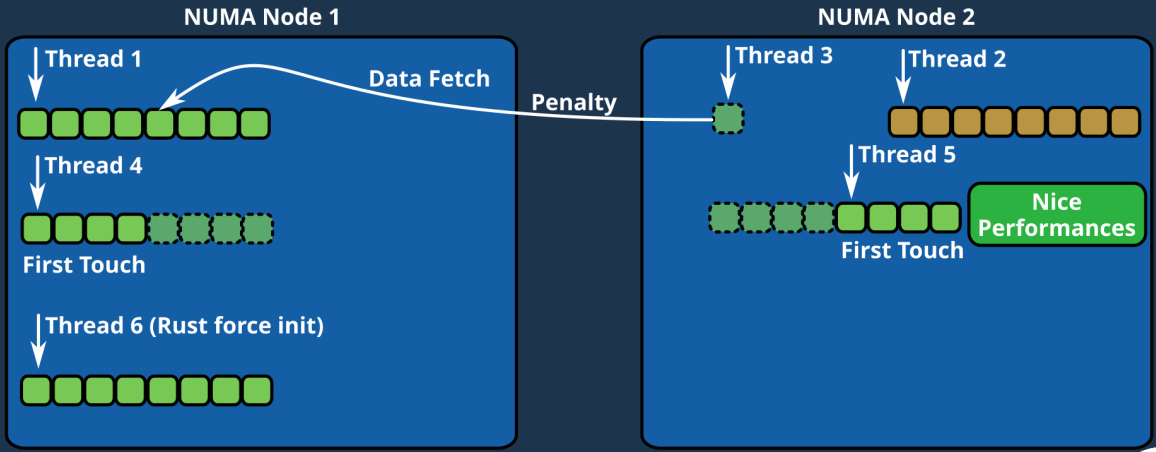


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

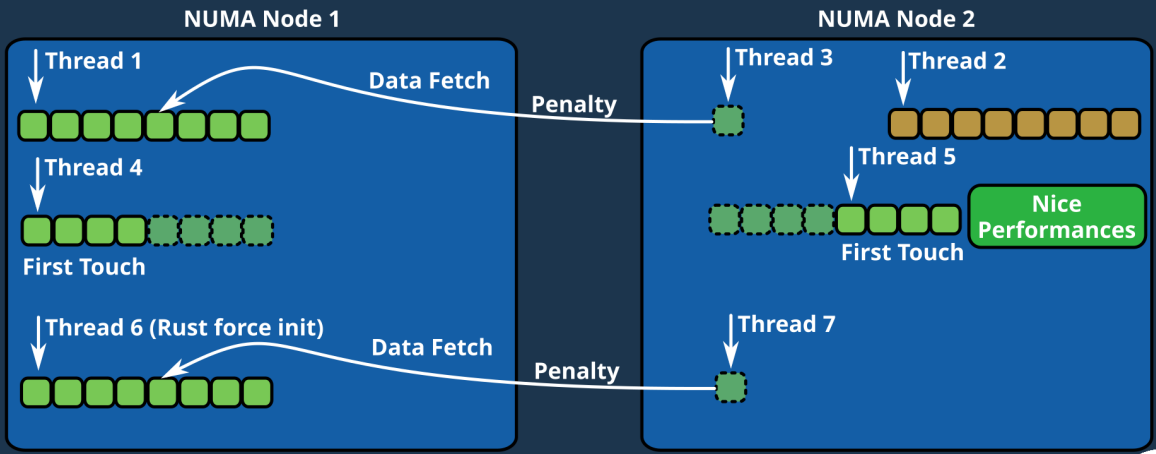


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

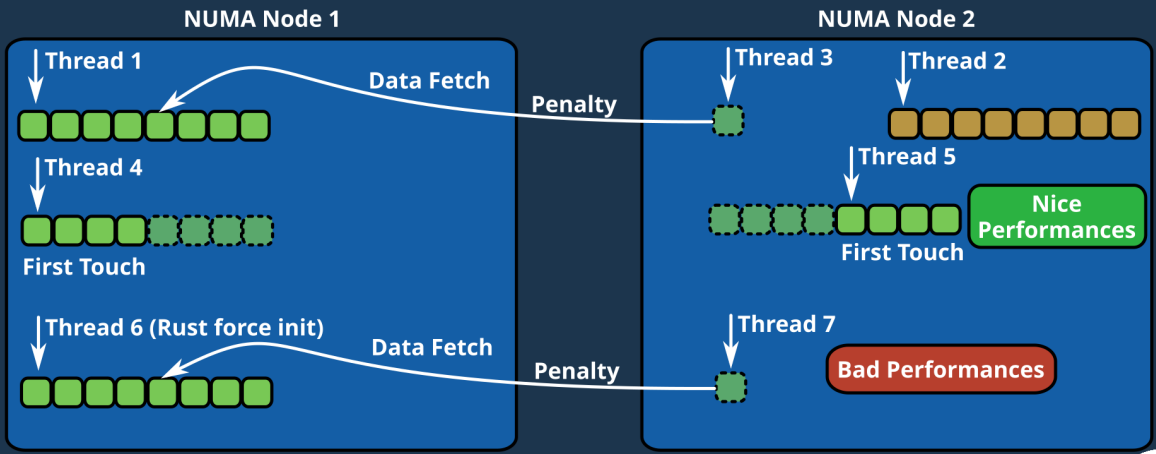


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**

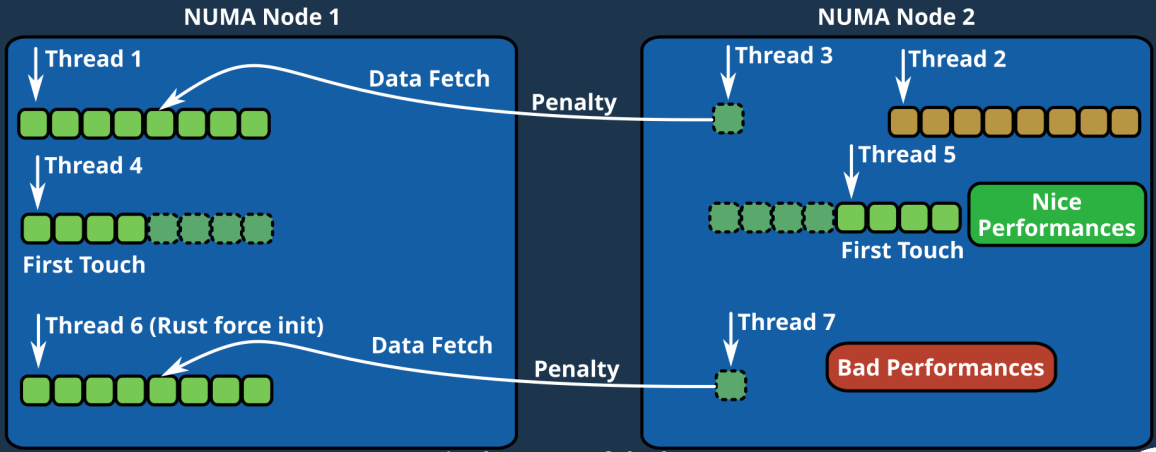


Rust redefines Stupidity

The hilarious example of the **NUMA nodes**

Rust forces to **initialise variables**

Globaly a **good point** to **prevent mistakes**



Rust is not finished yet

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

- Compilers :
  - **C++** : g++, clang++, dpc++, nvc++, nvcc, ...
  - **Rust** : rustc

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

- Compilers :
  - **C++** : g++, clang++, dpc++, nvc++, nvcc, ...
  - **Rust** : rustc

- **CPU Computing** not very mature
- **GPU Computing** not mature at all

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

- Compilers :
  - **C++** : g++, clang++, dpc++, nvc++, nvcc, ...
  - **Rust** : rustc

- **CPU Computing** not very mature
- **GPU Computing** not mature at all

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

- **13 MB** with Cargo 1.75 (debug)
- **13 MB** with Cargo 1.75 (release)

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

- Compilers :
  - **C++** : g++, clang++, dpc++, nvc++, nvcc, ...
  - **Rust** : rustc

- **CPU Computing** not very mature
- **GPU Computing** not mature at all

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

- **13 MB** with Cargo 1.75 (debug)
- **13 MB** with Cargo 1.75 (release)

```cpp
#include <iostream>
int main(int argc, char** argv){
»       std::cout << "Hello world !" << std::endl;
»       return 0;
}
```

- **33kB** in C++ (-O0 -g)
- **17kB** in C++ (-O3)

Rust is not finished yet

- **No library** (supposed to be an ongoing work)
  - **Only static linking** for now except for the **libgcc_s** and **libc**
  - **Crates** are just **configuration** without documentation, **not binary packages**
  - Need to **recompile the whole world** for any program

- Compilers :
  - **C++** : g++, clang++, dpc++, nvc++, nvcc, ...
  - **Rust** : rustc

- **CPU Computing** not very mature
- **GPU Computing** not mature at all

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

- **13 MB** with Cargo 1.75 (debug)
- **13 MB** with Cargo 1.75 (release)

```cpp
#include <iostream>
int main(int argc, char** argv){
»       std::cout << "Hello world !" << std::endl;
»       return 0;
}
```

- **33kB** in C++ (-O0 -g)
- **17kB** in C++ (-O3)

- After a **15 km long command line** we still get a binary **28x times bigger** than C++

Rust compiler is not mature yet

Rust compiler is not mature yet

```
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Rust compiler is not mature yet

```
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

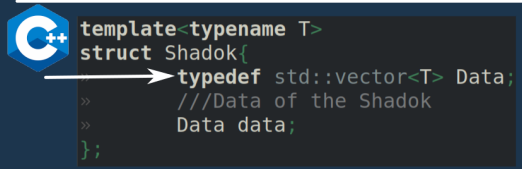
Rust compiler is not mature yet

```
template<typename T>
struct Shadok{
»       typedef std::vector<T> Data;
»       ///Data of the Shadok
»       Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Rust compiler is not mature yet



```
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

typedef -> **type**
Only defined in a **trait**, **forbidden** in a **struct**

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
»       typedef std::vector<T> Data;
»       ///Data of the Shadok
»       Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

typedef -> type
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
»       ///Data of the trait
»       type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
»       ///Data of the Shadok
»       data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
»       type Data = Vec<T>;
}
```

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Even **better** to **use** a **trait**

typedef -> type
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
    ///Data of the trait
    type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
    ///Data of the Shadok
    data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
    type Data = Vec<T>;
}
```

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Even **better** to **use** a **trait**

typedef -> type
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
    ///Data of the trait
    type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
    ///Data of the Shadok
    data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
    type Data = Vec<T>;
}
```

Typedef to **simplify** future calls

```rust
type Shadok = PShadok<i32>;
```

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Even **better** to **use** a **trait**

typedef -> type
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
    ///Data of the trait
    type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
    ///Data of the Shadok
    data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
    type Data = Vec<T>;
}
```

Typedef to **simplify** future calls

```rust
type Shadok = PShadok<i32>;
```

Impossible to call **Shadok::Data** yet

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Even **better** to **use** a **trait**

typedef -> **type**
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
    ///Data of the trait
    type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
    ///Data of the Shadok
    data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
    type Data = Vec<T>;
}
```

Typedef to **simplify** future calls

```rust
type Shadok = PShadok<i32>;
```

Impossible to call **Shadok::Data** yet

Use **<Shadok as ShadokTrait>::Data** for now

Rust compiler is not mature yet

```cpp
template<typename T>
struct Shadok{
    typedef std::vector<T> Data;
    ///Data of the Shadok
    Data data;
};
```

Design of the **std::vector<T>** :
- call **Shadok<T>::Data**

This is **great** and this **prevents** a lot of **errors** because there is **no duplication**.

Flexibility to replace **std::vector** by other container :
- **std::vector** -> **thrust::universal_vector** for **GPU** computing

Even **better** to **use** a **trait**

typedef -> type
Only defined in a **trait**, **forbidden** in a **struct**

```rust
///Trait of a Shadok
trait ShadokTrait{
    ///Data of the trait
    type Data;
}
///Test Shadok
#[derive(Debug)]
struct PShadok<T>{
    ///Data of the Shadok
    data: Vec<T>,
}
impl<T> ShadokTrait for PShadok<T>{
    type Data = Vec<T>;
}
```

Typedef to **simplify** future calls

```rust
type Shadok = PShadok<i32>;
```

Impossible to call **Shadok::Data** yet

Use **<Shadok as ShadokTrait>::Data** for now

Rust **Developers** are **aware** of this **issue** and **work on it**

OK, so why Rust ?

The **two** languages **development**

OK, so why Rust ?

The **two** languages **development**

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

OK, so why Rust ?

The **two** languages **development**

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

- **Experienced**
- **Implicit**
- **Plenty** of Code Generators
- **Production Ready**
- **Production Language**

OK, so why Rust ?

Quality :
- Design
- Implementation

The **two** languages **development**

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

- **Experienced**
- **Implicit**
- **Plenty** of Code Generators
- **Production Ready**
- **Production Language**

Development Time

OK, so why Rust ?

The **two** languages **development**

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

- **Experienced**
- **Implicit**
- **Plenty** of Code Generators
- **Production Ready**
- **Production Language**

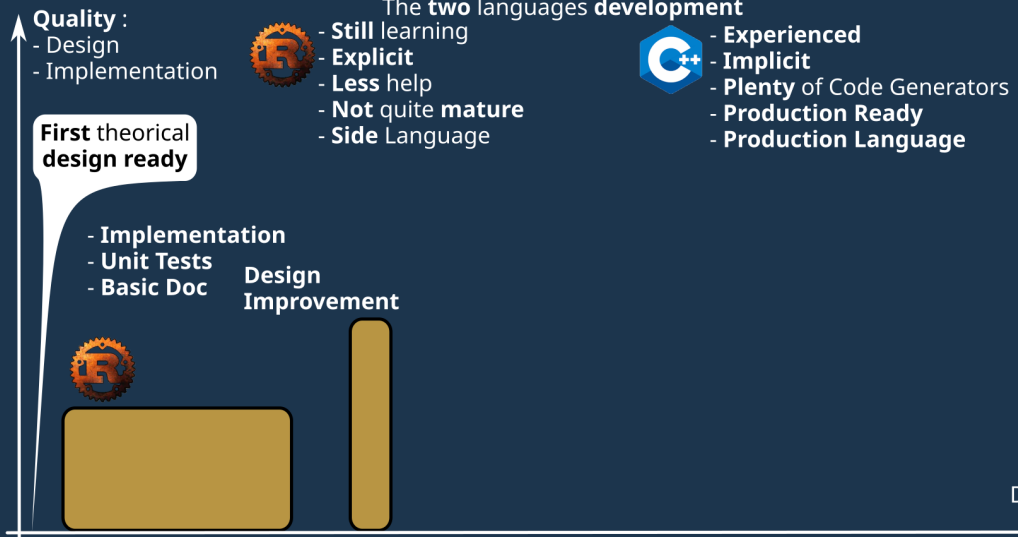
OK, so why Rust ?

The **two** languages **development**

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

- **Experienced**
- **Implicit**
- **Plenty** of Code Generators
- **Production Ready**
- **Production Language**

OK, so why Rust ?



OK, so why Rust ?

The **two** languages **development**

Quality :
- Design
- Implementation

- **Still** learning
- **Explicit**
- **Less** help
- **Not** quite **mature**
- **Side** Language

- **Experienced**
- **Implicit**
- **Plenty** of Code Generators
- **Production Ready**
- **Production Language**

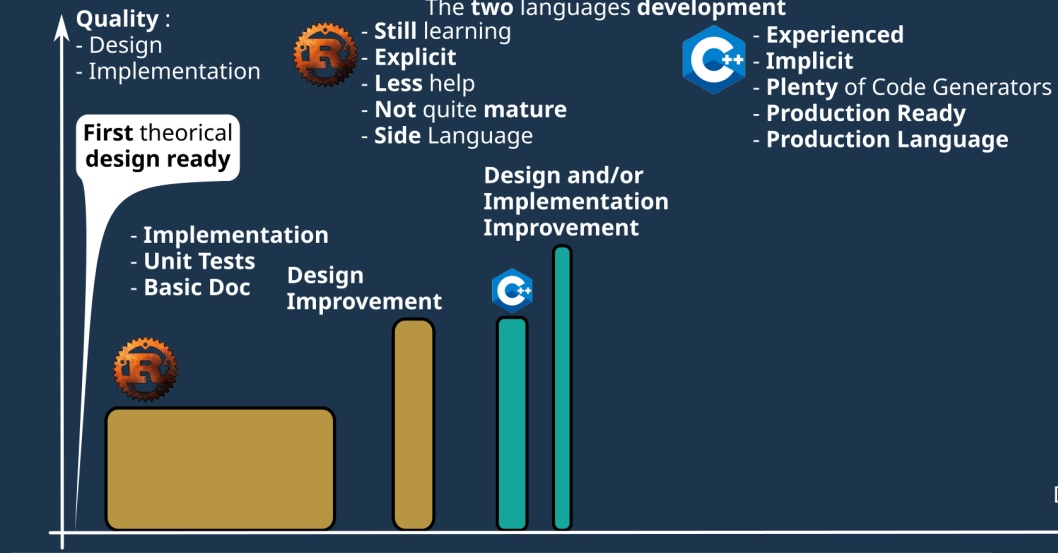
First theorical **design ready**

- **Implementation**
- **Unit Tests**
- **Basic Doc**

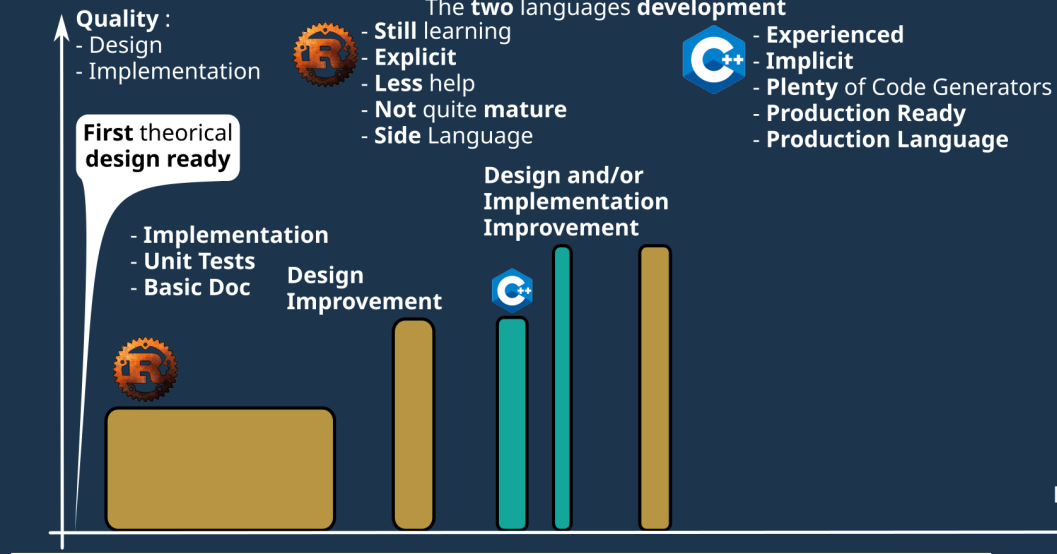
Design Improvement

Development Time

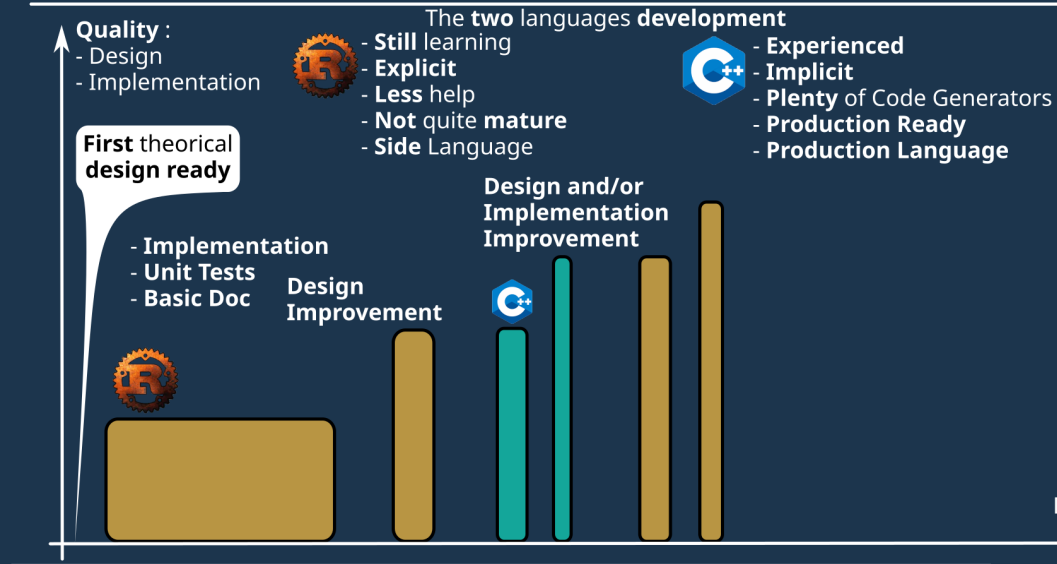
OK, so why Rust ?



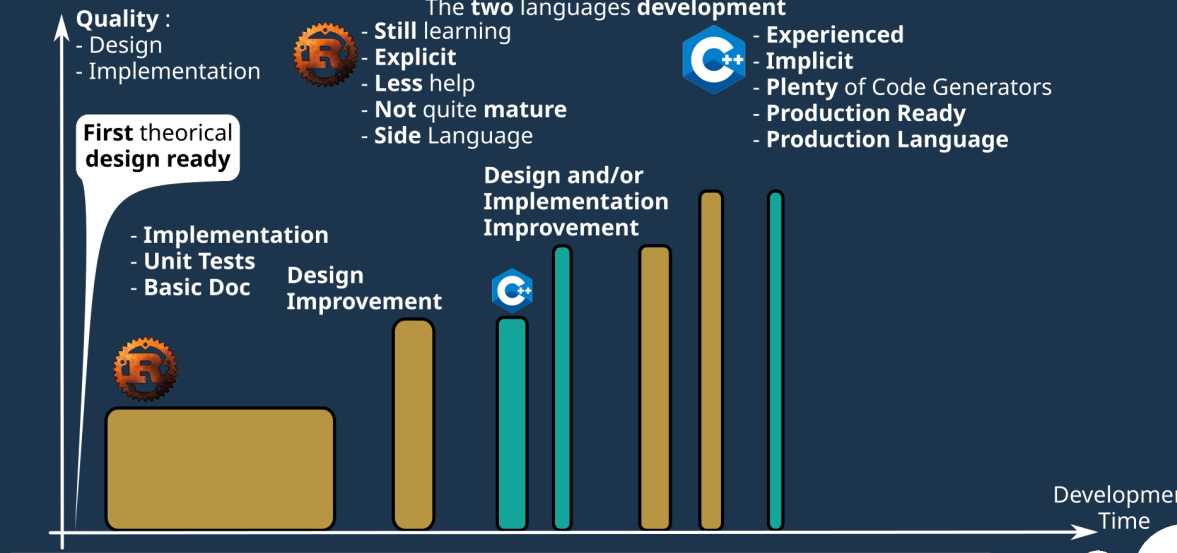
OK, so why Rust ?



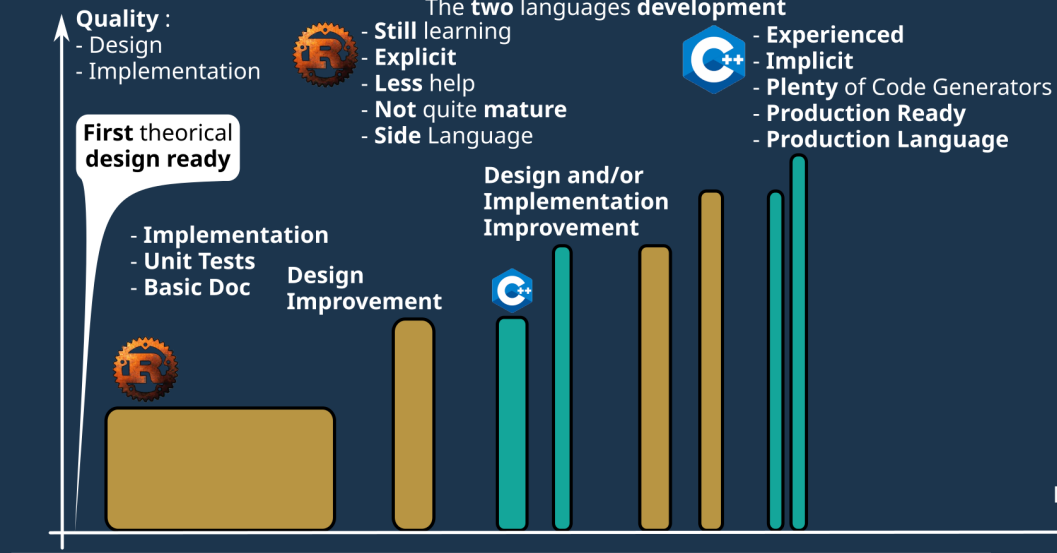
OK, so why Rust ?



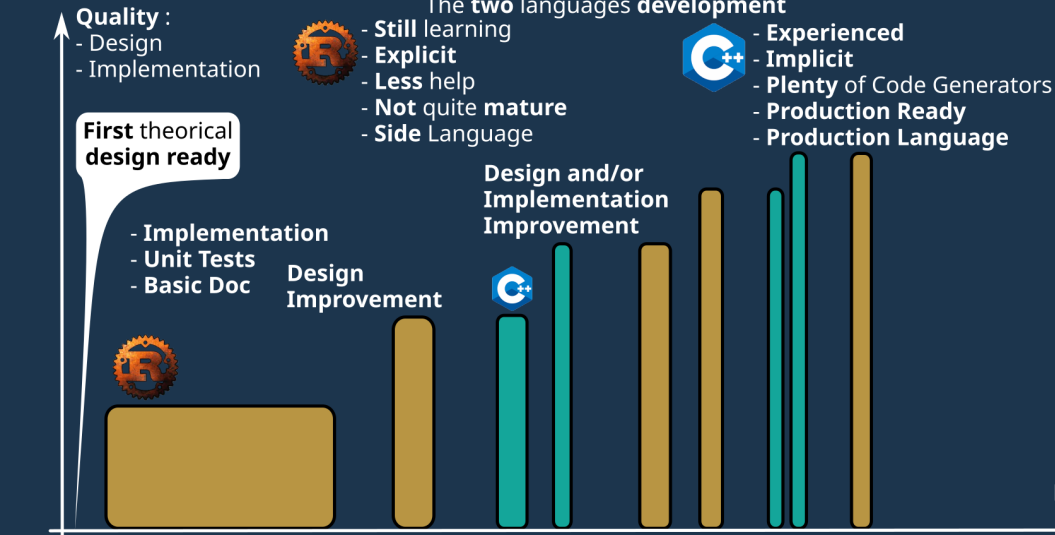
OK, so why Rust ?



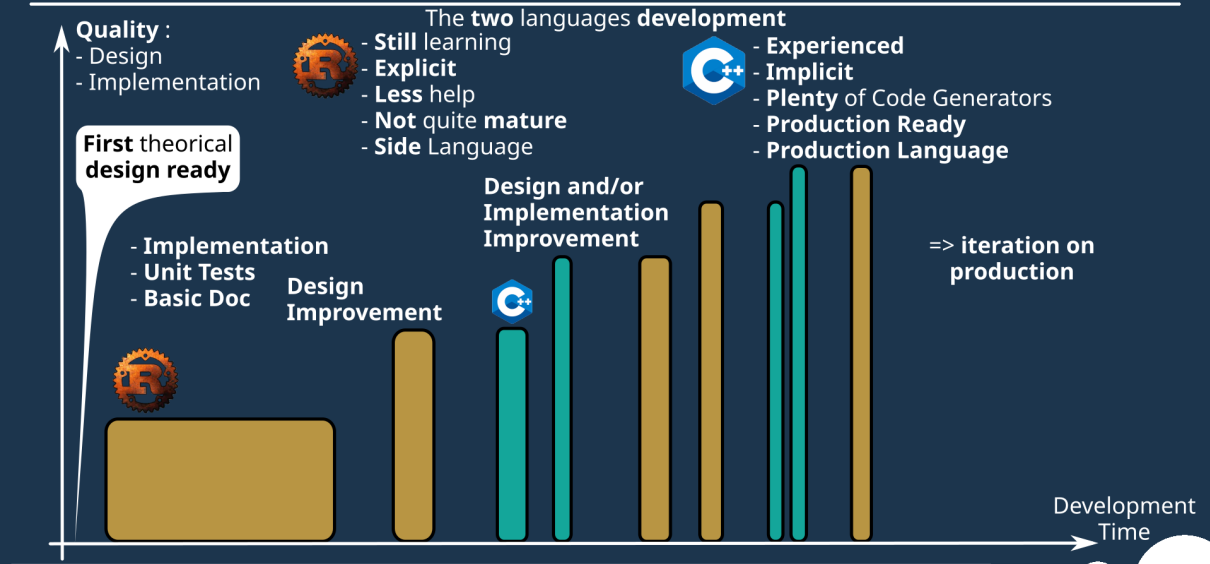
OK, so why Rust ?



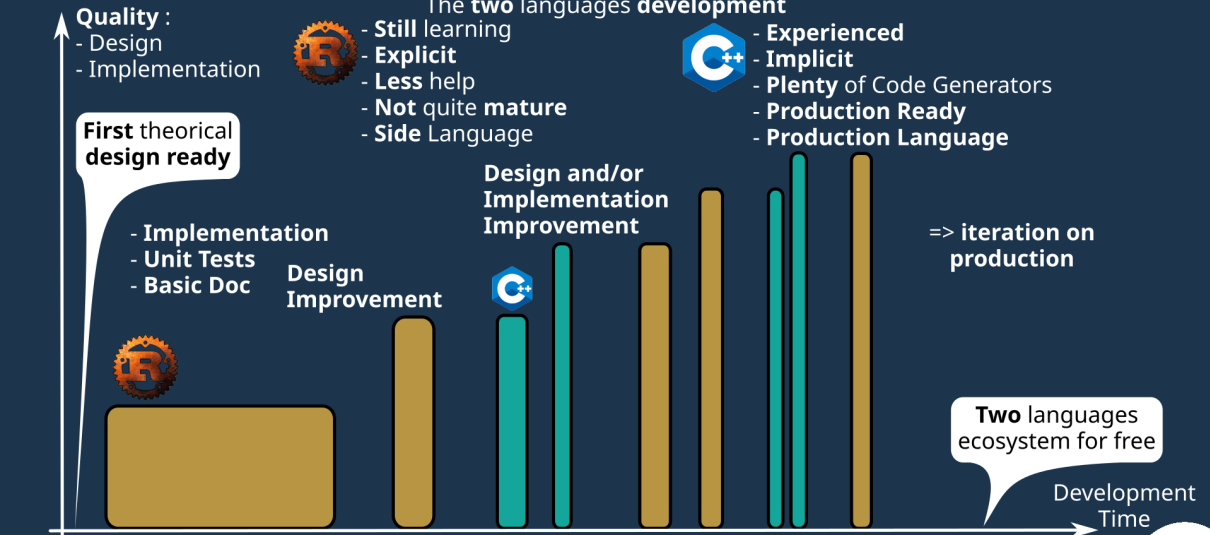
OK, so why Rust ?



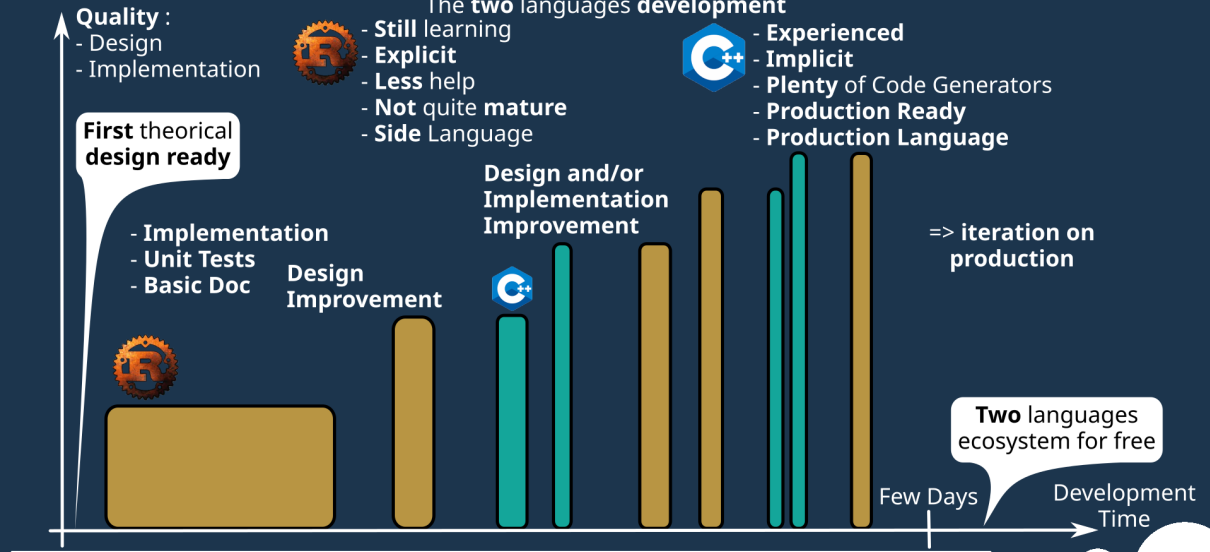
OK, so why Rust ?



OK, so why Rust ?



OK, so why Rust ?



Conclusion

Conclusion

Conclusion

Conclusion

Conclusion

Conclusion

Conclusion

Conclusion

Conclusion

In **C++**, you can do **everything** !

Conclusion

In **C++**, you can do **everything** !

Even **stupid** things

Conclusion

In **C++**, you can do **everything** !

With **great power** comes **great responsabilities**

Even **stupid** things

Conclusion

In **C++**, you can do **everything** !

With **great power** comes **great responsabilities**

Even **stupid** things

In **Rust**, you can do **nothing** !

Conclusion

In **C++**, you can do **everything** !

With **great power** comes **great responsabilities**

Even **stupid** things

In **Rust**, you can do **nothing** !

Even **smart** things

Conclusion

In **C++**, you can do **everything** !

With **great power** comes **great responsabilities**

Even **stupid** things

In **Rust**, you can do **nothing** !

Even **smart** things

The **Rust** learning curve is slow but **your design will improve** a lot.
Give it a try.