Rust : Performance **et** Sécurité

Daniel Henry-Mantilla

12 juin 2019

Checks à la compilation ou à l'exécution ?

Exemple : type checking

basic.py

```python
from sys import argv, exit, stderr

try:
    arg = int(argv[1])
except (IndexError, ValueError):
    print(
        "Usage: {} <number>"
        .format(argv[0]),
        file=stderr,
    )
    exit(1)

if arg != 42:
    print("This is not a very interesting number...")
else:
    print("The answer is indeed " + arg + "!")
```

Exemple : type checking

basic.py

```python
from sys import argv, exit, stderr

try:
    arg = int(argv[1])
except (IndexError, ValueError):
    print(
        "Usage: {} <number>"
        .format(argv[0]),
        file=stderr,
    )
    exit(1)

if arg != 42:
    print("This is not a very interesting number...")
else:
    print("The answer is indeed " + arg + "!")
```

```
$ python3 basic.py 27
This is not a very interesting number...
```

Exemple : type checking

basic.py

```python
from sys import argv, exit, stderr

try:
    arg = int(argv[1])
except (IndexError, ValueError):
    print(
        "Usage: {} <number>"
        .format(argv[0]),
        file=stderr,
    )
    exit(1)

if arg != 42:
    print("This is not a very interesting number...")
else:
    print("The answer is indeed " + arg + "!")
```

```
$ python3 basic.py 27
This is not a very interesting number...


$ python3 basic.py 42
Traceback (most recent call last):
  File "basic.py", line 16, in <module>
    print("The answer is indeed " + arg + "!")
TypeError: Can't convert 'int' object to str implicitly
```

Exemple : type checking

basic.py

```python
from sys import argv, exit, stderr

try:
    arg = int(argv[1])
except (IndexError, ValueError):
    print(
        "Usage: {} <number>"
        .format(argv[0]),
        file=stderr,
    )
    exit(1)

if arg != 42:
    print("This is not a very interesting number...")
else:
    print("The answer is indeed " + arg + "!")
```

```
$ python3 basic.py 27
This is not a very interesting number...


$ python3 basic.py 42
Traceback (most recent call last):
  File "basic.py", line 16, in <module>
    print("The answer is indeed " + arg + "!")
TypeError: Can't convert 'int' object to str implicitly


$ python3 -m mypy basic.py  # (no input number)
basic.py:16: error: Unsupported operand types for + ...
```

Exemple : type checking en Rust

basic.rs

```rust
use ::std::env::args;

fn main ()
{
    let arg: String =
        args()
            // impl Iterator<Item = String>
            .nth(1)
            // Option<String>
            .expect("Missing parameter `<number>`")
            // String
    ;
    if arg != 42 {
        println!("This is not a very interesting number...");
    } else {
        println!("The answer is indeed {}!", arg);
    }
}
```

Exemple : type checking en Rust

cargo check

```
user@DESKTOP-DHB5CKG:~/Documents/git/presentation/example$ cargo check
    Checking example v0.1.0 (/mnt/c/Users/CakeIsAPie/Documents/git/presentation/example)
error[E0277]: can't compare `std::string::String` with `{integer}`
  --> src/main.rs:13:12
   |
13 |     if arg != 42 {
   |            ^^ no implementation for `std::string::String == {integer}`
   |
   = help: the trait `std::cmp::PartialEq<{integer}>` is not implemented for `std::string::String`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0277`.
error: Could not compile `example`.

To learn more, run the command again with --verbose.
```

- Playground

C'est tout ?

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)
    - "ownership"
    - "borrowing" & "lifetimes"

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)
    - "ownership"
    - "borrowing" & "lifetimes"
- Empêche les *Data races*

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)
    - "ownership"
    - "borrowing" & "lifetimes"
- Empêche les *Data races* !???

Et encore plus de checks !

L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)
    - "ownership"
    - "borrowing" & "lifetimes"
- Empêche les *Data races*
  - ↪ Plusieurs threads : "Fearless Concurrency"

Et encore plus de checks !

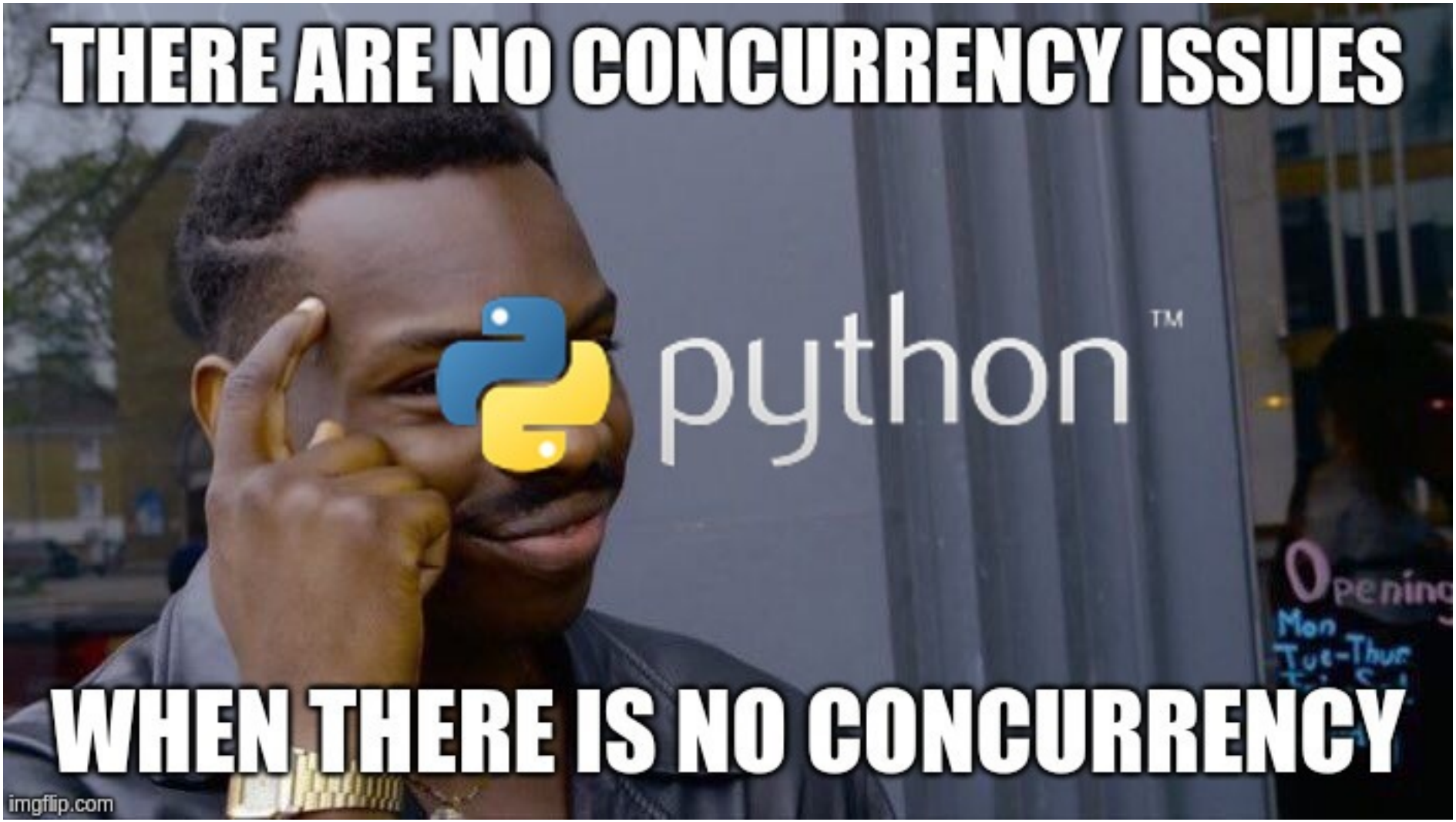
L'analyse statique de Rust peut faire bien plus que juste du "type checking" :

- Empêche les bugs / vulnerabilités de type *use-after-free* ou *double-free*
  - ↪ sans besoin de *garbage collector* ($\Longrightarrow$ WASM, performance **fiable**)
    - "ownership"
    - "borrowing" & "lifetimes"
- Empêche les *Data races*
  - ↪ Plusieurs threads : "Fearless Concurrency"
  - ↪ Un seul thread : plus de *dangling pointer* (par exemple : plus d'invalidation d'itérateur)

Data races en environnement multithreadé

Data races en environnement multithreadé : Python

Data races en environnement multithreadé : Python



Data races en environnement multithreadé : C

data-race.c

```c
void increment (int * at_x) {
    for (size_t i = 0; i < N; ++i) {
        *at_x += 1;
    }
}
int main (void) {
    int x = 0;
    pthread_t other_thread;
    // spawn a new thread that calls increment(&x)
    if (pthread_create(&other_thread, NULL, increment, &x) != 0)
        exit(EXIT_FAILURE);
    increment(&x);
```

Data races en environnement multithreadé : C

data-race.c

```c
void increment (int * at_x) {
    for (size_t i = 0; i < N; ++i) {
        *at_x += 1;
    }
}
int main (void) {
    int x = 0;
    pthread_t other_thread;
    // spawn a new thread that calls increment(&x)
    if (pthread_create(&other_thread, NULL, increment, &x) != 0)
        exit(EXIT_FAILURE);
    increment(&x);
```

Data races en environnement multithreadé : C

data-race.c

```c
void increment (int * at_x) {
    for (size_t i = 0; i < N; ++i) {
        *at_x += 1;
    }
}
int main (void) {
    int x = 0;
    pthread_t other_thread;
    // spawn a new thread that calls increment(&x)
    if (pthread_create(&other_thread, NULL, increment, &x) != 0)
        exit(EXIT_FAILURE);
    increment(&x);

    // wait for the other_thread to finish too
    pthread_join(other_thread, NULL);
    printf("x = %d (%.1f%% of expected value)\n",
            x, ((double) x) * 50. / ((double) N));
    return EXIT_SUCCESS;
}
```

Data races en environnement multithreadé : C

data-race.c

```c
void increment (int * at_x) {
    for (size_t i = 0; i < N; ++i) {
        *at_x += 1;
    }
}
int main (void) {
    int x = 0;
    pthread_t other_thread;
    // spawn a new thread that calls increment(&x)
    if (pthread_create(&other_thread, NULL, increment, &x) != 0)
        exit(EXIT_FAILURE);
    increment(&x);

    // wait for the other_thread to finish too
    pthread_join(other_thread, NULL);
    printf("x = %d (%.1f%% of expected value)\n",
            x, ((double) x) * 50. / ((double) N));
    return EXIT_SUCCESS;
}
// x = 5488566 (54.9% of expected value)
```

Data races en environnement multithreadé : Rust

data-race.rs

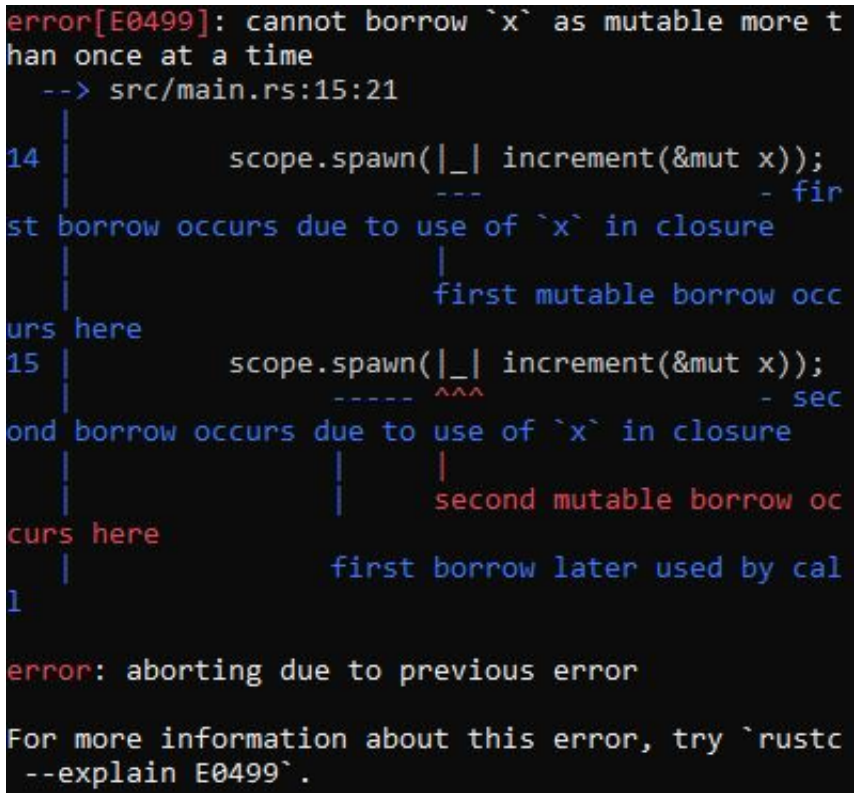
```rust
const N: usize = 50_000_000;

fn increment (at_x: &mut i32) {
    for i in 0 .. N {
        *at_x += 1;
    }
}

fn main () {
    let mut x: i32 = 0;
    ::crossbeam::scope(|scope| {
        scope.spawn(|_| increment(&mut x));
        scope.spawn(|_| increment(&mut x));
    }) // wait for all threads to finish
      .expect("Some thread panicked.");
    println!("x = {}", x);
}
```

Data races en environnement multithreadé : Rust

data-race.rs

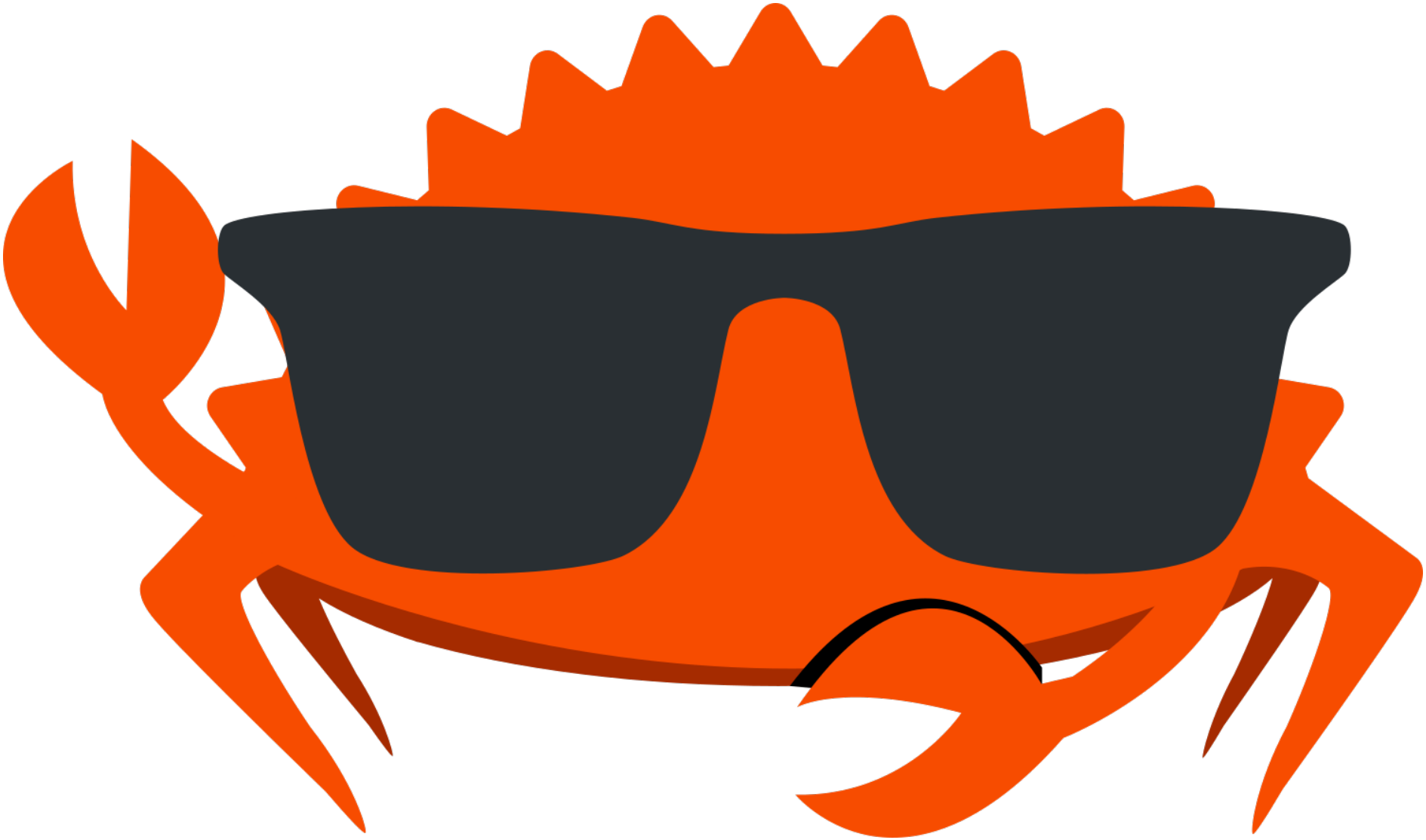
```rust
const N: usize = 50_000_000;

fn increment (at_x: &mut i32) {
    for i in 0 .. N {
        *at_x += 1;
    }
}

fn main () {
    let mut x: i32 = 0;
    ::crossbeam::scope(|scope| {
        scope.spawn(|_| increment(&mut x));
        scope.spawn(|_| increment(&mut x));
    }) // wait for all threads to finish
      .expect("Some thread panicked.");
    println!("x = {}", x);
}
```

$ cargo check



- Playground

Data races en environnement multithreadé : Rust



by TheZoq2

Data races en environnement multithreadé : Rust

no-data-race.rs

```rust
fn increment (at_x: & AtomicCell<i32>) {
    for i in 0 .. N {
        at_x.fetch_add(1);
    }
}

fn main () {
    let x = <AtomicCell<i32>>::new(0);
    ::crossbeam::scope(|scope| {
        scope.spawn(|_| increment(& x));
        scope.spawn(|_| increment(& x));
    }) // wait for all threads to finish
      .expect("Some thread panicked.");
    println!("x = {}", x.load());
}
```

Data races en environnement multithreadé : Rust

no-data-race.rs

```rust
fn increment (at_x: & AtomicCell<i32>) {
    for i in 0 .. N {
        at_x.fetch_add(1);
    }
}

fn main () {
    let x = <AtomicCell<i32>>::new(0);
    ::crossbeam::scope(|scope| {
        scope.spawn(|_| increment(& x));
        scope.spawn(|_| increment(& x));
    }) // wait for all threads to finish
      .expect("Some thread panicked.");
    println!("x = {}", x.load());
}
```

data-race.rs

```rust
fn increment (at_x: &mut i32) {
    for i in 0 .. N {
        *at_x += 1;
    }
}

fn main () {
    let mut x: i32 = 0;
    ::crossbeam::scope(|scope| {
        scope.spawn(|_| increment(&mut x));
        scope.spawn(|_| increment(&mut x));
    }) // wait for all threads to finish
      .expect("Some thread panicked.");
    println!("x = {}", x);
}
```

- Playground

Nul besoin de parallélisme pour avoir des "data races"

C++ : Dangling pointer

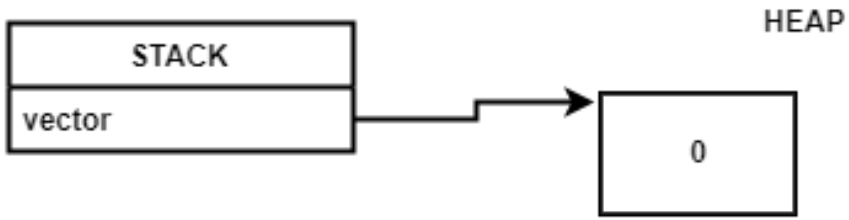
dangling-pointer.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> vector(1);
    int * at_first = &vector[0];
    vector.push_back(42);
    cout << *at_first;
}
```

C++ : Dangling pointer

dangling-pointer.cpp

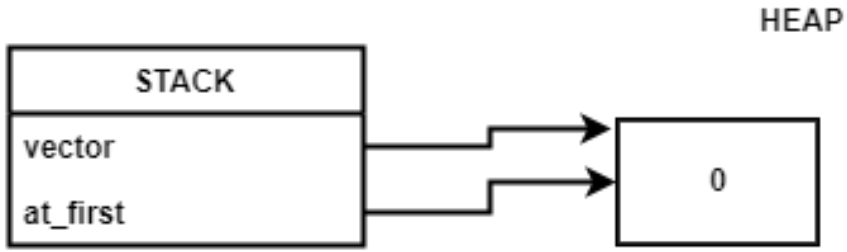
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> vector(1);
    int * at_first = &vector[0];
    vector.push_back(42);
    cout << *at_first;
}
```



C++ : Dangling pointer

dangling-pointer.cpp

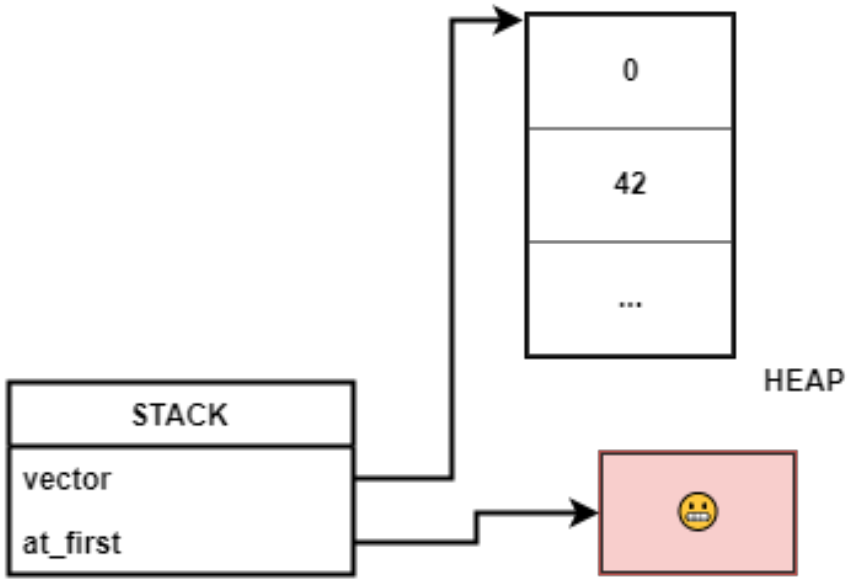
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> vector(1);
    int * at_first = &vector[0];
    vector.push_back(42);
    cout << *at_first;
}
```



C++ : Dangling pointer

dangling-pointer.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
    vector<int> vector(1);
    int * at_first = &vector[0];
    vector.push_back(42);
    cout << *at_first;
}
```



Rust : Dangling pointer ?

dangling-pointer.cpp

```
5  int main ()
6  {
7      vector<int> vector(1);
8      int * at_first = &vector[0];
9      vector.push_back(42);
10     cout << *at_first;
11 }
```

Rust : Dangling pointer ?

dangling-pointer.cpp

```cpp
int main ()
{
    vector<int> vector(1);
    int * at_first = &vector[0];
    vector.push_back(42);
    cout << *at_first;
}
```

dangling-pointer.rs

```rust
fn main ()
{
    let mut vector: Vec<i32> = vec![0];
    let at_first: &i32 = &vector[0];
    vector.push(42);
    println!("{}", *at_first);
}
```

Rust : Dangling pointer ?

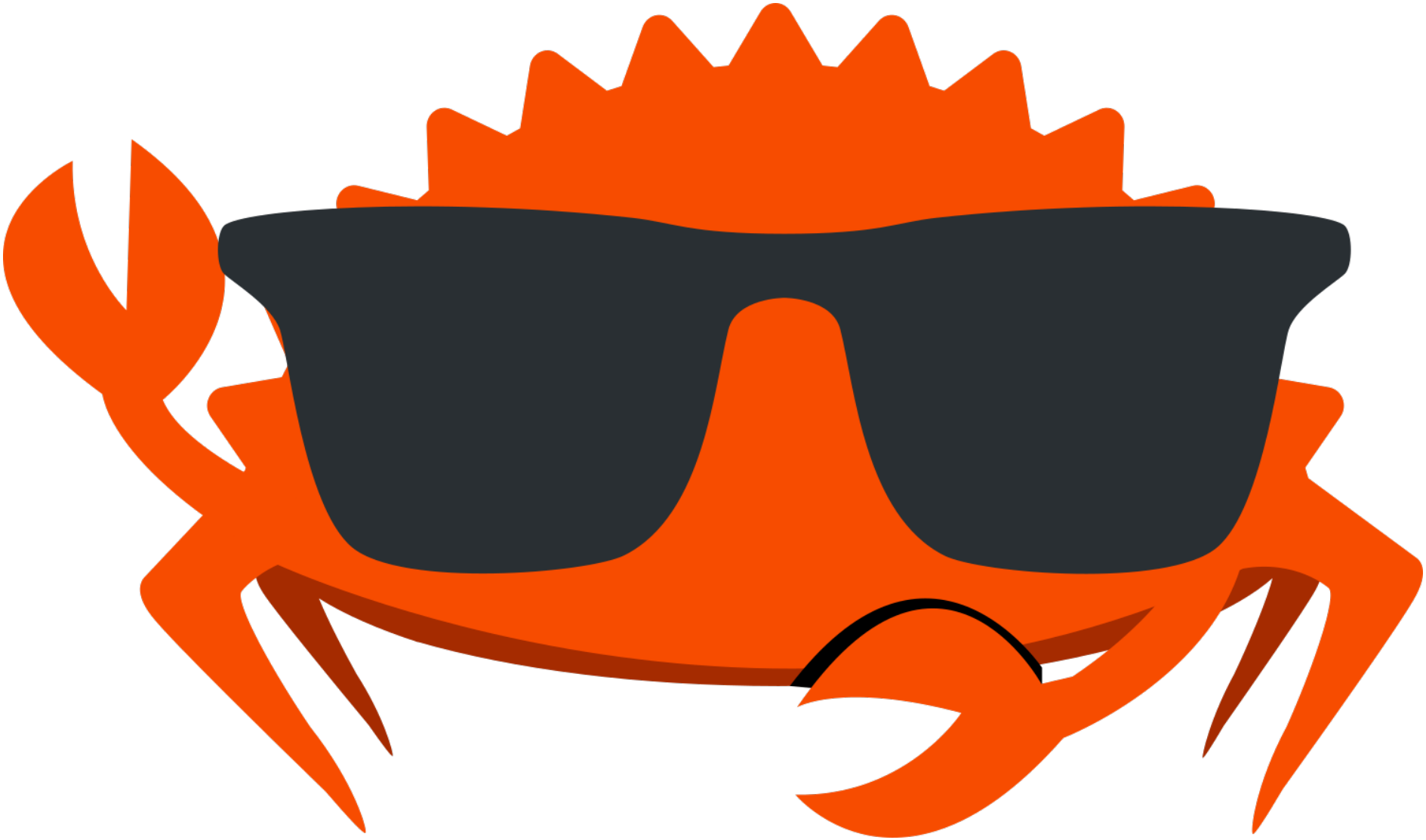
dangling-pointer.rs

```rust
fn main ()
{
    let mut vector: Vec<i32> = vec![0];
    let at_first: &i32 = &vector[0];
    vector.push(42);
    println!("{}", *at_first);
}
```

```
error[E0502]: cannot borrow `vector` as mutable because it is also borrowed as immutable
 --> /home/user/Documents/git/presentation/dangling-pointer.rs:5:5
  |
4 |     let at_first: &i32 = &vector[0];
  |                          ------ immutable borrow occurs here
5 |     vector.push(42);
  |     ^^^^^^ mutable borrow occurs here
6 |     println!("{}", *at_first);
7 | }
  | - immutable borrow ends here
```

- Playground

Rust : Dangling pointer ?



by TheZoq2

Python : invalidation d'itérateur

iterator-bug.py

```
2   def append_all(
3       out_list,
4       elements,
5   ):
6       for x in elements:
7           out_list.append(x)
8       return
```

Python : invalidation d'itérateur

iterator-bug.py

```
2   def append_all(
3       out_list,
4       elements,
5   ):
6       for x in elements:
7           out_list.append(x)
8       return
9
10  l = [1, 2, 3]
11  append_all(l, l)
```

Python : invalidation d'itérateur

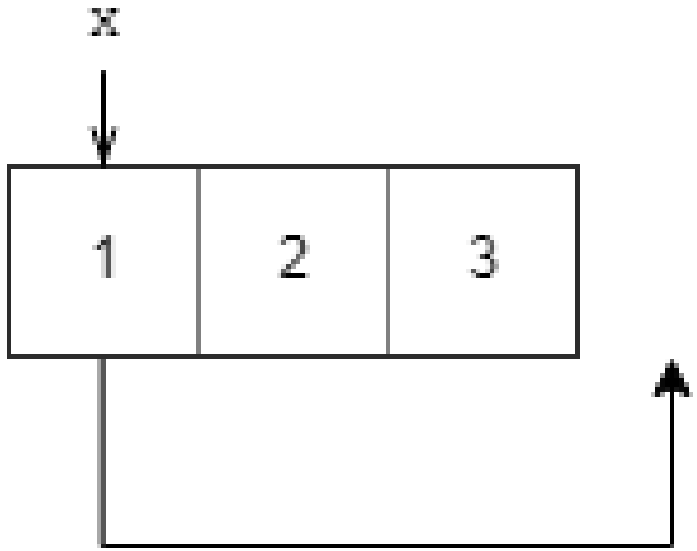
iterator-bug.py

```
2     def append_all(
3         out_list,
4         elements,
5     ):
6         for x in elements:
7             out_list.append(x)
8         return
9
10    l = [1, 2, 3]
11    append_all(l, l)
```

Python : invalidation d'itérateur

iterator-bug.py

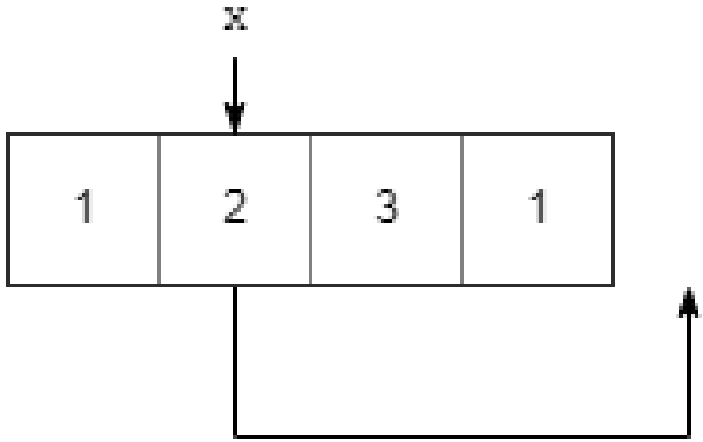
```
2     def append_all(
3         out_list,
4         elements,
5     ):
6         for x in elements:
7             out_list.append(x)
8         return
9
10    l = [1, 2, 3]
11    append_all(l, l)
```



Python : invalidation d'itérateur

iterator-bug.py

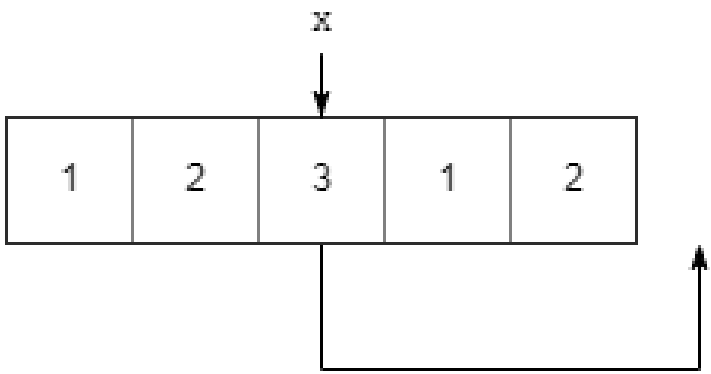
```
2     def append_all(
3         out_list,
4         elements,
5     ):
6         for x in elements:
7             out_list.append(x)
8         return
9
10    l = [1, 2, 3]
11    append_all(l, l)
```



Python : invalidation d'itérateur

iterator-bug.py

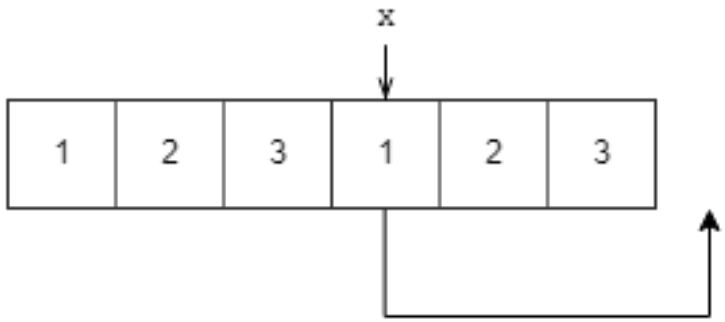
```
2   def append_all(
3       out_list,
4       elements,
5   ):
6       for x in elements:
7           out_list.append(x)
8       return
9
10  l = [1, 2, 3]
11  append_all(l, l)
```



Python : invalidation d'itérateur

iterator-bug.py

```
2     def append_all(
3         out_list,
4         elements,
5     ):
6         for x in elements:
7             out_list.append(x)
8         return
9
10    l = [1, 2, 3]
11    append_all(l, l)
```



Rust : Invalidation d'itérateur ?

iterator-bug.py

```
def main():
    def append_all(
        out_list,
        elements,
    ):
        for x in elements:
            out_list.append(x)
        return

    l = [1, 2, 3]
    append_all(l, l)
main()
```

Rust : Invalidation d'itérateur ?

iterator-bug.py

```python
def main():
    def append_all(
        out_list,
        elements,
    ):
        for x in elements:
            out_list.append(x)
        return

    l = [1, 2, 3]
    append_all(l, l)
main()
```

iterator-bug.rs

```rust
fn main () {
    fn append_all (
        out_vec: &mut Vec<i32>,
        elements: &[i32],
    ) {
        for &x in elements {
            out_vec.push(x)
        }
    }
    let mut l = vec![1, 2, 3];
    append_all(&mut l, &l);
}
```

Rust : Invalidation d'itérateur ?

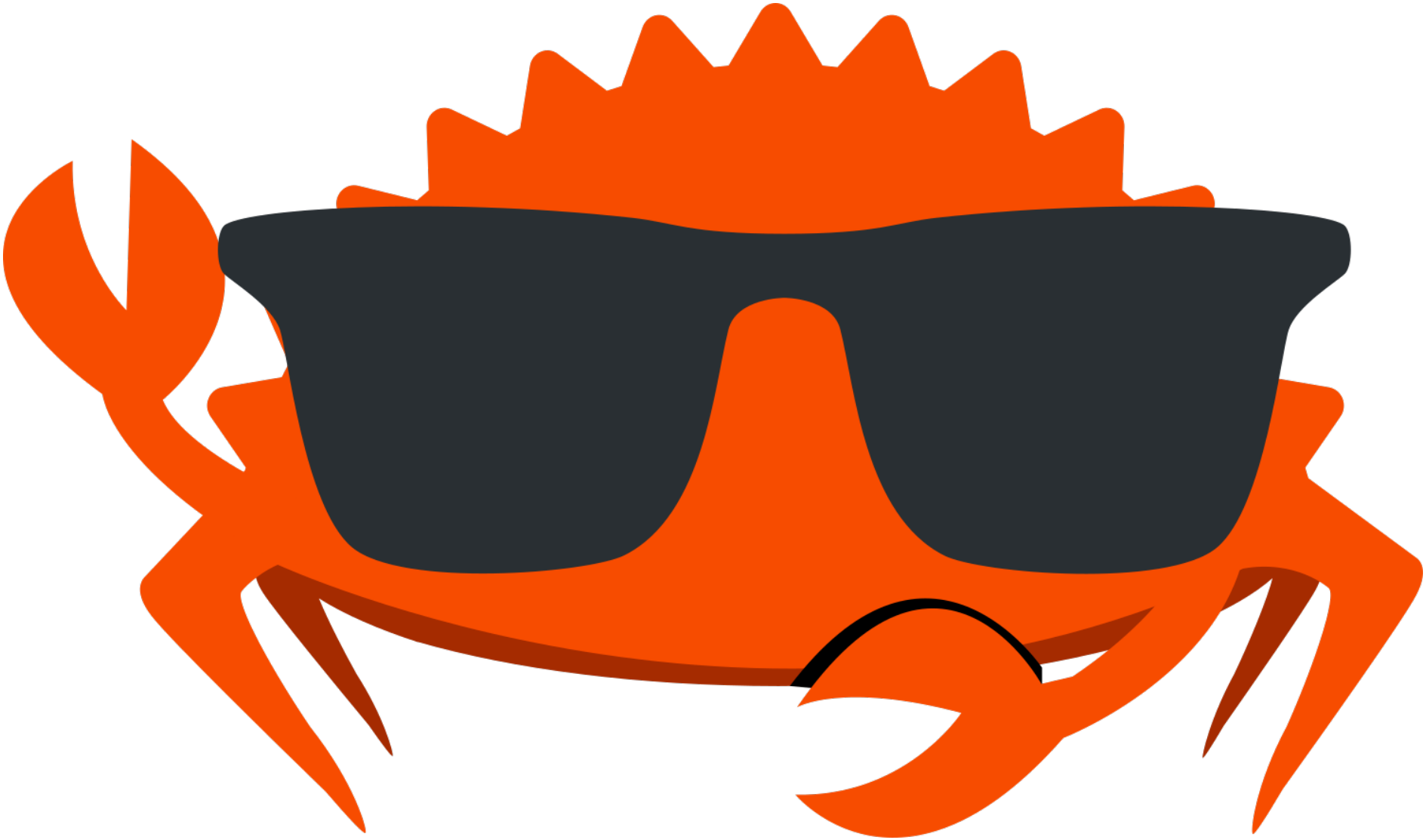
- Playground

iterator-bug.rs

```rust
fn main () {
    fn append_all (
        out_vec: &mut Vec<i32>,
        elements: &[i32],
    ) {
        for &x in elements {
            out_vec.push(x)
        }
    }
    let mut l = vec![1, 2, 3];
    append_all(&mut l, &l);
}
```

```
error[E0502]: cannot borrow `l` as immutable because it is also borrowed as mutable
  --> /home/user/Documents/git/presentation/iterator-bug.rs:11:25
   |
11 |     append_all(&mut l, &l);
   |                -    ^- mutable borrow ends here
   |                |    |
   |                |    immutable borrow occurs here
   |                mutable borrow occurs here
```

Rust : Invalidation d'itérateur ?



by TheZoq2

Conclusion

Questions ?