



**Rust**


**Intro & model pamięci**

**Łukasz Neumann**

lukasz.neumann@pw.edu.pl

[https://staff.elka.pw.edu.pl/~lneumann/rust\\_1.pdf](https://staff.elka.pw.edu.pl/~lneumann/rust_1.pdf)

# Anty-agenda

- unsafe
- makra 
- społeczność
- async

# O czym będą wykłady?

1. Model pamięci, struktury i cechy
2. Programowanie współbieżne
3. Zarządzanie projektem, łączenie z innymi językami oraz kiedy należy przepisywać stary kod

# This is **not** a language war talk



**Cpp** @cpp\_lang

· Feb 24, 2019



Am I the most elegant language out there ? Of course not. But do I have memory safety ? Well, no. But do I wake up every day and try to be the easiest language to learn ? Also no.

 815

 75.7K

 241.5K



# Hype driven development



# Rust

GET STARTED

[Version 1.52.1](#)

A language empowering everyone to build reliable and efficient software.

## Why Rust?

### Performance

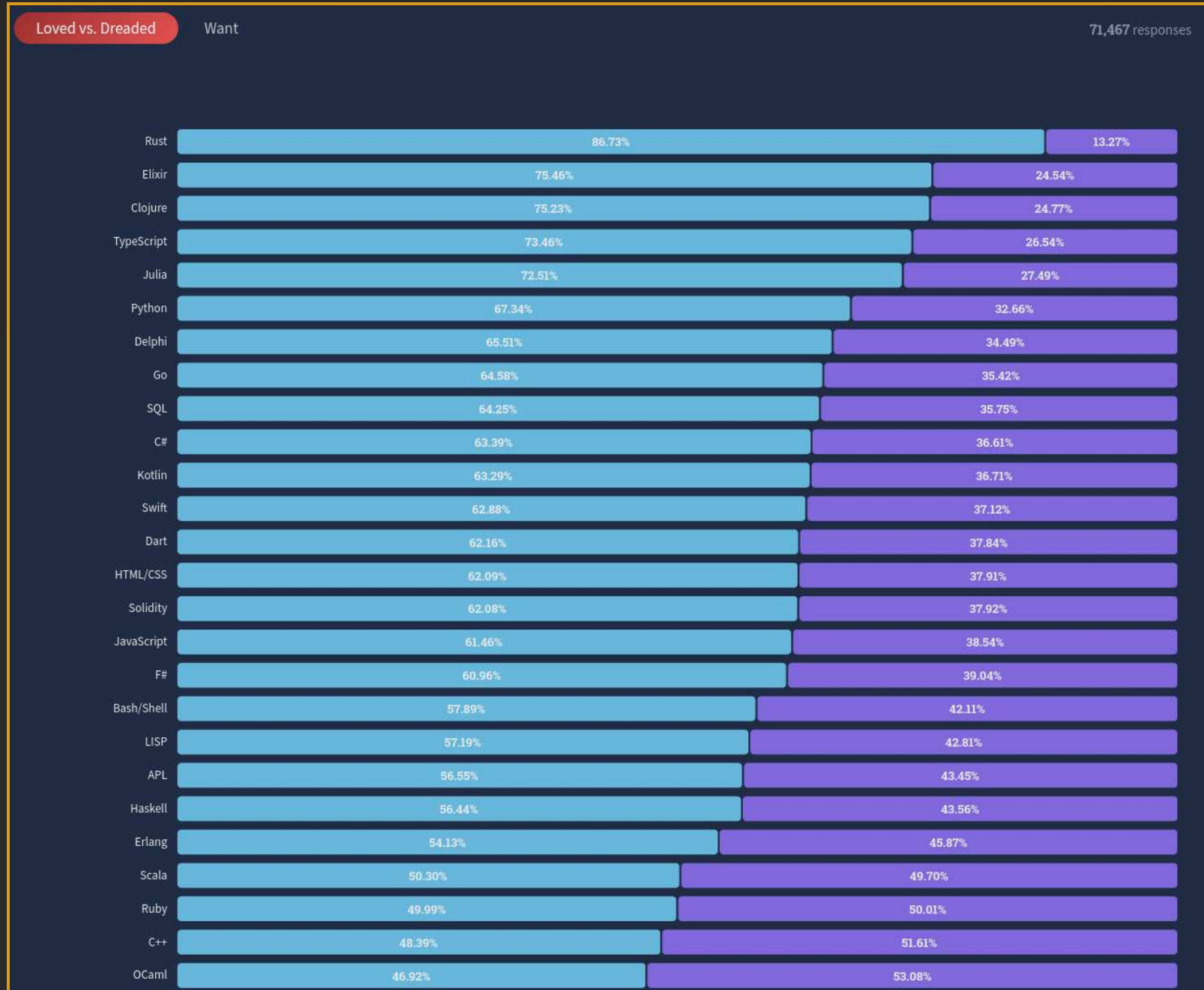
Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

### Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

### Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.



master 1 branch 810 tags

Go to file Code

About

Linux kernel source tree

- Readme
- View license
- Activity
- 159k stars
- 8.3k watching
- 50.7k forks

Report repository

Releases

810 tags

Packages

No packages published

Contributors 5,000+



+ 14,650 contributors

Languages



torvalds	Linux 6.6-rc6	5872080	12 hours ago	🕒 1,216,693 commits
Documentation	Merge tag 'char-misc-6.6-rc6' of git://git.kernel.org/pub/scm/linux/k...		17 hours ago	
LICENSES	LICENSES: Add the copyleft-next-0.3.1 license		last year	
arch	Revert "x86/smp: Put CPUs into INIT on shutdown if possible"		14 hours ago	
block	block: Don't invalidate pagecache for invalid falloc modes		4 days ago	
certs	certs: Reference revocation list for all keyrings		2 months ago	
crypto	crypto: sm2 - Fix crash caused by uninitialized context		last month	
drivers	virtio_net: fix the missing of the dma cpu sync		14 hours ago	
fs	Merge tag 'ovl-fixes-6.6-rc6' of git://git.kernel.org/pub/scm/linux/k...		17 hours ago	
include	Merge tag 'char-misc-6.6-rc6' of git://git.kernel.org/pub/scm/linux/k...		17 hours ago	
init	Merge tag 'wq-for-6.6' of git://git.kernel.org/pub/scm/linux/kernel/...		last month	
io_uring	io-wq: fully initialize wqe before calling cpuhp_state_add_instance_n...		2 weeks ago	
ipc	Merge tag 'x86_shstk_for_6.6-rc1' of git://git.kernel.org/pub/scm/lin...		2 months ago	
kernel	Merge tag 'sched-urgent-2023-10-14' of git://git.kernel.org/pub/sc...		yesterday	
lib	maple_tree: add MAS_UNDERFLOW and MAS_OVERFLOW states		2 weeks ago	
mm	Merge tag 'mm-hotfixes-stable-2023-10-01-08-34' of git://git.kernel....		2 weeks ago	
net	Merge tag 'ceph-for-6.6-rc6' of https://github.com/ceph/ceph-client		3 days ago	
rust	Merge tag 'docs-6.6' of git://git.lwn.net/linux		2 months ago	
samples	Merge tag 'vfio-v6.6-rc1' of https://github.com/awilliam/linux-vfio		2 months ago	
scripts	kbuild: remove stale code for 'source' symlink in packaging scripts		2 weeks ago	
security	KEYS: trusted: Remove redundant static calls usage		last week	



# Źródło bólu w dużych projektach

- Chrome: 70% of high/critical vulnerabilities are memory unsafety
- Firefox: 72% of vulnerabilities in 2019 are memory unsafety
- Odays: 81% of in the wild Odays (PO dataset) are memory unsafety
- Microsoft: 70% of all MSRC tracked vulnerabilities are memory unsafety
- Ubuntu: 65% of kernel CVEs in USNs in a 6-month sample are memory unsafety
- Android: More than 65% of high/critical vulnerabilities are memory unsafety
- macOS: 71.5% of Mojave CVEs are due to memory unsafety

# Memory safety problems

- Access errors
  - Buffer overflow
  - Buffer over-read
  - Data race condition
  - Use after free
- Uninitialized variables
  - Null pointer dereference
  - Wild pointers
- Memory leak
  - Stack exhaustion
  - Heap exhaustion
  - Double free
  - Invalid free

Todd Howard: "If you're running low on memory, you can reboot the original Xbox and the user can't tell. You can throw a screen up. When Morrowind loads sometimes you get a very long load. That's us rebooting the Xbox."



# Case study - Cloudflare

“ When crashes do occur an engineer needs to spend time to diagnose how it happened and what caused it. Since Pingora's inception we've served a few hundred trillion requests and have yet to crash due to our service code.

In fact, Pingora crashes are so rare we usually find unrelated issues when we do encounter one. Recently we discovered a kernel bug soon after our service started crashing. We've also discovered hardware issues on a few machines, **in the past ruling out rare memory bugs caused by our software even after significant debugging was nearly impossible.**

”

# Const-by-default

```
1 let a = 7;  
2 a += 1;
```



```
error[E0384]: cannot assign twice to immutable variable `a`  
  → const_by_default.rs:3:5  
2 |     let a = 7;  
  |     -  
  |     |  
  |     first assignment to `a`  
  |     help: consider making this binding mutable: `mut a`  
3 |     a += 1;  
  |     ^^^^^^ cannot assign twice to immutable variable
```

# Obowiązkowa inicjalizacja



```
1 let to_find: i32;  
2 let values: Vec<i32>;  
3 let found = values.iter().find(|&&x| x == to_find);  
4 println!("{found:?}");
```



```
1 int to_find;  
2 std::vector<int> values;  
3 auto found = std::find(values.begin(), values.end(), to_find);  
4 std::cout << *found << std::endl;
```

# Obowiązkowa inicjalizacja



```
> rustc lack_of_init.rs
error[E0381]: borrow of possibly-uninitialized variable: `values`
  → lack_of_init.rs:4:17
  |
4 |     let found = values.iter().find(|&&x| x == to_find);
  |                               ^^^^^^^^^^^^^^^^^ use of possibly-uninitialized `values`

error[E0381]: borrow of possibly-uninitialized variable: `to_find`
  → lack_of_init.rs:4:36
  |
4 |     let found = values.iter().find(|&&x| x == to_find);
  |                               ^^^^^^          borrow occurs due to use in closure
  |                               |
  |                               use of possibly-uninitialized `to_find`
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 lack_of_init.cpp && ./a.out
[1] 40941 segmentation fault (core dumped) ./a.out
```

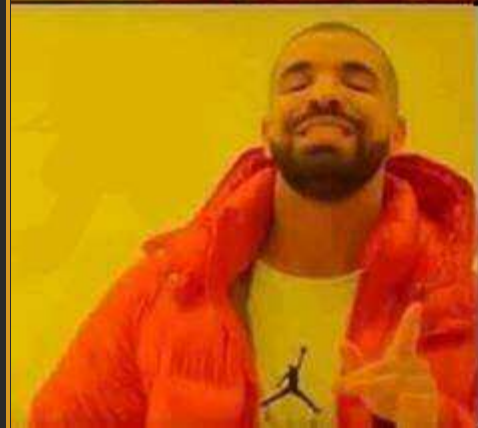
# Undefined Behavior

- Undefined behavior is the result of executing a program whose behavior is **prescribed** to be unpredictable.
- This is different from:
  - **unspecified** behavior, for which the language specification **does not prescribe** a result
  - **implementation-defined** behavior that defers to the documentation of another component of the platform (such as the ABI or the translator documentation).
- The only possible UB in safe Rust is a result of a bug in the compiler or in the standard library

# Zarządzanie pamięcią



**Segmentation Fault**



**Suprise Garbage  
Collection**



# Resource Acquisition Is Initialization



```
1 fn dummy() {  
2     let mut text = String::from("Hi");  
3     text += " world!";  
4     println!("{text}");  
5     println!("{}", text.len());  
6 }
```



```
1 void dummy() {  
2     auto text = std::string("Hi");  
3     text += " world!";  
4     std::cout << text << std::endl;  
5     std::cout << text.size() << std::endl;  
6 }
```



# Borrow checker - windykator

1. Each value in Rust has a variable that's called its **owner**.
2. There can only be **one** owner at a time.
3. When the owner goes **out of scope**, the value will be dropped.



# Właściciel - przykłady



```
1 let v1 = vec![1, 2, 3];  
2 let v2 = v1;  
3 println!("{}", v1.len(), v2.len());
```



```
1 std::vector<int> v1{1, 2, 3};  
2 auto v2 = std::move(v1);  
3 std::cout << v1.size() << " " << v2.size() << std::endl;
```

# Właściciel - przykłady



```
> rustc assignment.rs && ./assignment
error[E0382]: borrow of moved value: `v1`
  → assignment.rs:4:23
   |
2  |     let v1 = vec![1, 2, 3];
   |           -- move occurs because `v1` has type `Vec<i32>`, which does not implement the `Copy` trait
3  |     let v2 = v1;
   |           -- value moved here
4  |     println!("{}", v1.len(), v2.len());
   |                   ^^^^^^^^^ value borrowed here after move
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 assignment.cpp && ./a.out
0 3
```

# Referencje

1. At any given time, you can have **either**:
  - one mutable reference
  - any number of immutable references
2. References must **always** be valid.

# Mieszanie mutowalności



```
1 let mut greetings = vec!["czesc", "hi", "hello"];
2
3 let hello = &greetings[2];
4 println!("{hello}");
5
6 greetings.pop();
7 println!("{hello}");
```



```
1 std::vector<std::string> greetings = {"czesc", "hi", "hello"};
2
3 const auto& hello = greetings[2];
4 std::cout << hello << std::endl;
5
6 greetings.pop_back();
7 std::cout << hello << std::endl;
```

# Mieszanie mutowalności



```
error[E0502]: cannot borrow `greetings` as mutable because it is also borrowed as immutable
  -> borrow.rs:7:5
   |
4  |     let hello = &greetings[2];
   |                   _____ immutable borrow occurs here
   |
...
7  |     greetings.pop();
   |     ^^^^^^^^^^^^^^^^^ mutable borrow occurs here
8  |     println!("{hello}");
   |                   _____ immutable borrow later used here
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow.cpp && ./a.out
hello
hello
```

# Undefined Behavior i jego skutki



```
1 std::vector<std::string> greetings = {"czesc", "hi", "hello from the other side"};
2
3 const auto& hello = greetings[2];
4 std::cout << hello << std::endl;
5
6 greetings.pop_back();
7 std::cout << hello << std::endl;
```

```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow.cpp && ./a.out
hello from the other side
vYwUn
7other side
```



# Dwa 'typy' referencji - przykład wredny



```
1 fn foo(first_vec: &Vec<i32>, second_vec: &mut Vec<i32>) {
2     second_vec.clear();
3     println!("{}", first_vec[0]);
4 }
5
6 fn main() {
7     let mut v = vec![1, 2, 3];
8     foo(&v, &mut v);
9 }
```



```
1 void foo(const std::vector<int> &first_vec, std::vector<int> &second_vec) {
2     second_vec.clear();
3     std::cout << first_vec[0] << std::endl;
4 }
5
6 int main() {
7     std::vector<int> v = {1, 2, 3};
8     foo(v, v);
9     return 0;
10 }
```

# Dwa rodzaje referencji - przykład wredny



```
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
  → borrow_function.rs:8:13
   |
8  |     foo(&v, &mut v);
   |     --    ^^^^^^^ mutable borrow occurs here
   |     |
   |     | immutable borrow occurs here
   |     | immutable borrow later used by call
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined borrow_function.cpp && ./a.out
1
> clang-tidy borrow_function.cpp
> cppcheck borrow_function.cpp
Checking borrow_function.cpp ...
```

# Życie po życiu (obiektu)



```
1 fn get_ref() → &'static i32 {
2     let my_number = 7;
3     let ref_ = &my_number;
4     ref_
5 }
6
7 fn main() {
8     let ref_ = get_ref();
9     println!("{ref_}");
10 }
```



```
1 const int& get_ref() {
2     int my_number = 7;
3     const int& ref = my_number;
4     return ref;
5 }
6
7 int main() {
8     const auto& ref = get_ref();
9     std::cout << ref << std::endl;
10 }
```

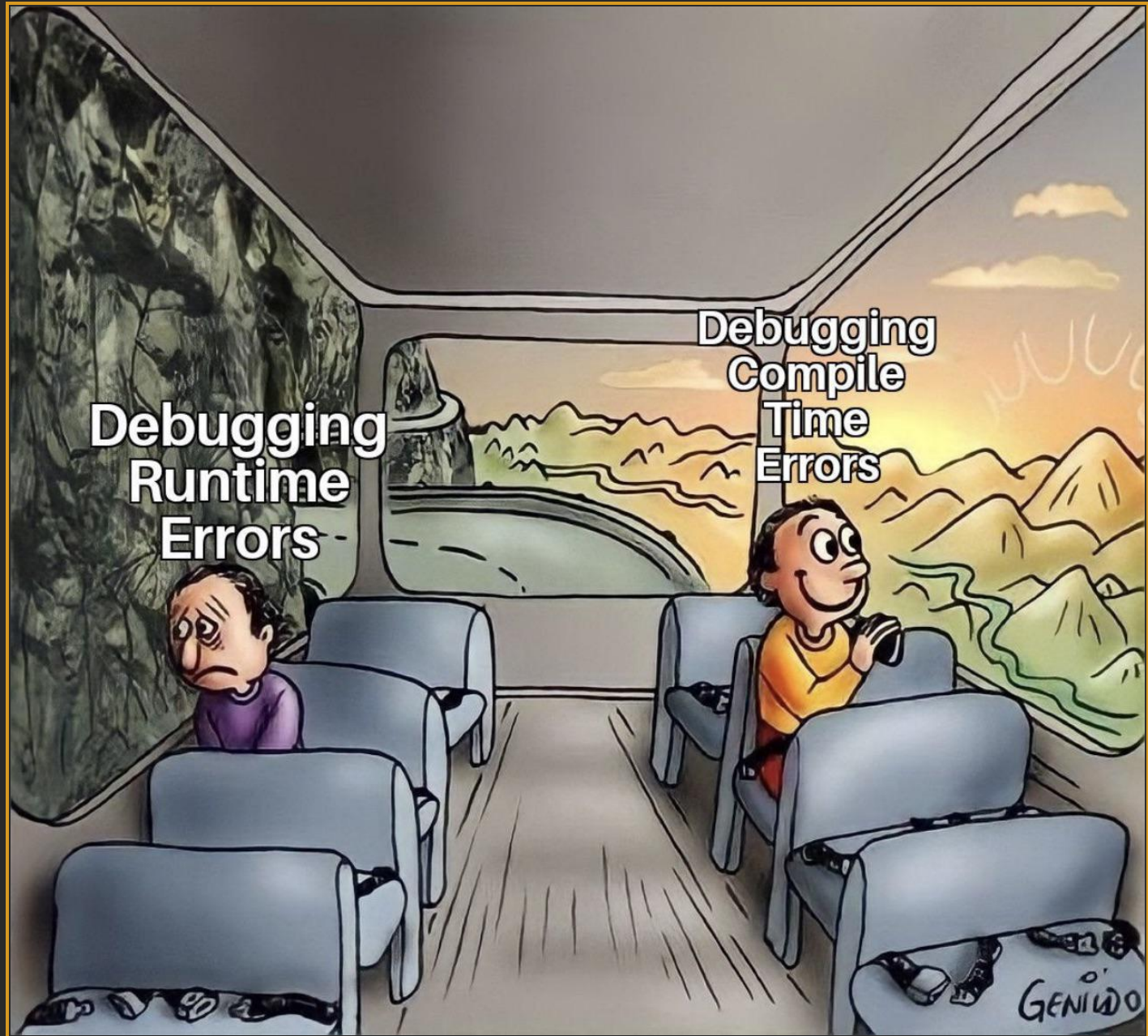
# Życie po życiu (obiektu)



```
> rustc out_of_scope.rs
error[E0515]: cannot return value referencing local variable `my_number`
  → out_of_scope.rs:4:5
   |
3  |     let ref_ = &my_number;
   |                _____ `my_number` is borrowed here
4  |     ref_
   |     ^^^^ returns a value referencing data owned by the current function
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -fsanitize=undefined out_of_scope.cpp && ./a.out
22031
```



# Struktury



```
1 struct Rectangle {
2     pub width: f32,
3     height: f32,
4 }
5
6 impl Rectangle {
7     fn diagonal_length(&self) → f32 {
8         (self.width.powf(2.) + self.height.powf(2.)).sqrt()
9     }
10
11     fn area(&self) → f32 {
12         self.width * self.height
13     }
14
15     fn perimeter(&self) → f32 {
16         2.0 * (self.width + self.height)
17     }
18 }
```

# Cechy



```
1 trait Shape {  
2  
3     fn area(&self) → f32;  
4  
5     fn perimeter(&self) → f32;  
6  
7     fn area_to_perimeter(&self) → f32 {  
8         self.area() / self.perimeter()  
9     }  
10 }
```

# Cechy - implementacja



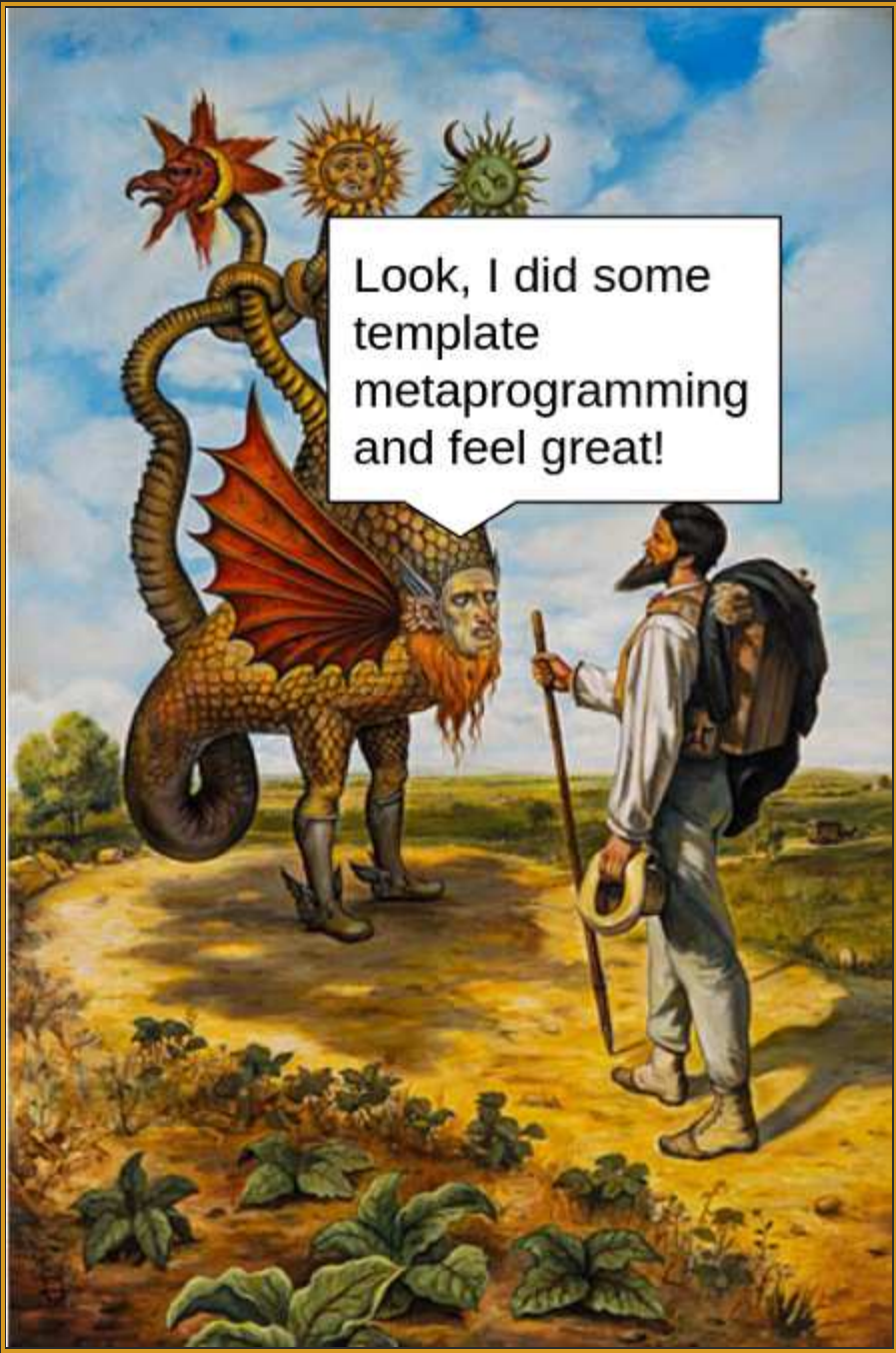
```
1 struct Rectangle {
2     pub width: f32,
3     height: f32,
4 }
5
6 impl Rectangle {
7     fn diagonal_length(&self) → f32 {
8         (self.width.powf(2.) + self.height.powf(2.)).sqrt()
9     }
10 }
11
12 impl Shape for Rectangle {
13     fn area(&self) → f32 {
14         self.width * self.height
15     }
16
17     fn perimeter(&self) → f32 {
18         2.0 * (self.width + self.height)
19     }
20 }
```



# Borrow checker quiz



```
1 struct Coords {
2     pub x: i64,
3     pub y: i64,
4 }
5
6 fn shift_x_twice(coords: &mut Coords, delta: &i64) {
7     coords.x += delta;
8     coords.x += delta;
9 }
10
11 fn main() {
12     let mut a = Coords{x: 10, y: 10};
13     let delta_a = 10;
14     shift_x_twice(&mut a, &delta_a);
15
16     let mut b = Coords{x: 10, y: 10};
17     let delta_b = &b.x;
18     shift_x_twice(&mut b, delta_b);
19 }
```



Look, I did some  
template  
metaprogramming  
and feel great!

# Generics



```
1 fn print_shape_info<T>(shape: &T) {
2     println!("Area: {}", shape.area());
3     println!("Perimeter: {}", shape.perimeter());
4     println!("Ratio: {}", shape.area_to_perimeter());
5 }
```

```
> rustc generics_wrong.rs
error[E0599]: no method named `area` found for reference `&T` in the current scope
  → generics_wrong.rs:32:32
   |
32 |     println!("Area: {}", shape.area());
   |                                ^^^^^ method not found in `&T`
   |
   = help: items from traits can only be used if the type parameter is bounded by the trait
help: the following trait defines an item `area`, perhaps you need to restrict type parameter `T` with it:
31 | fn print_shape_info<T: Shape>(shape: &T) {
   |             ~~~~~
```

# Co mówi nam sygnatura o referencjach?



```
1 /// Get free beer
2 fn free_beer(name: &str) → &str {
3     // magic
4 }
```

# Co mówi nam sygnatura o referencjach?



```
1 /// Get free beer
2 fn free_beer(name: &str, surname: &str) → &str {
3     // magic
4 }
```

# Prosty ko[dt] w Ruście



```
1 struct Cat {
2     pub weight: u32,
3     pub name: String,
4 }
5
6 fn chonkiest(x: &Cat, y: &Cat) → &Cat {
7     if x.weight > y.weight {
8         x
9     } else {
10        y
11    }
12 }
13
14 fn main() {
15     let x = Cat {weight: 7, name: String::from("Kotlet")};
16     let chonky; // ← uninitialized
17     let y = Cat {weight: 1, name: String::from("Kotara")};
18     chonky = chonkiest(&x, &y);
19     println!("{}", chonky.name);
20 }
```

# Jaki będzie zakres życia referencji?



```
1 struct Cat {
2     pub weight: u32,
3     pub name: String,
4 }
5
6 fn chonkiest_cat(x: &Cat, y: &Cat) → &Cat {
7     if x.weight > y.weight {
8         x
9     } else {
10        y
11    }
12 }
13
14 fn main() {
15     let x = Cat {weight: 7, name: String::from("Kotlet")};
16     let chonky; // ← uninitialized
17     {
18         let y = Cat {weight: 1, name: String::from("Kotara")};
19         chonky = chonkiest_cat(&x, &y);
20     }
21     println!("{}", chonky.name);
22 }
```

# Błąd kompilacji - brak lifetimes

```
> rustc cat.rs
error[E0106]: missing lifetime specifier
  → cat.rs:6:39
   |
6  | fn chonkiest_cat(x: &Cat, y: &Cat) → &Cat {
   |                        _____ ^ expected named lifetime parameter
   |
   = help: this function's return type contains a borrowed value,
         but the signature does not say whether it is borrowed from `x` or `y`
help: consider introducing a named lifetime parameter
6  | fn chonkiest_cat<'a>(x: &'a Cat, y: &'a Cat) → &'a Cat {
   |                        ++++  ++          ++          ++
error: aborting due to previous error

For more information about this error, try `rustc --explain E0106`.
```



# Lifetimes



```
1 {  
2     let r;           // -----+-- 'a  
3                     // |  
4     {               // |  
5         let x = 5;   // -+-- 'b |  
6         r = &x;      // | |  
7     }               // -+ |  
8                     // |  
9     println!("{r}"); // |  
10 }
```

```
1 {  
2     let x = 5;       // -----+-- 'b  
3                     // |  
4     let r = &x;      // --+-- 'a |  
5                     // | |  
6     println!("{r}"); // | |  
7                     // --+ |  
8 }
```

# Naprawiony ko[dt]



```
1 struct Cat {
2     pub weight: u32,
3     pub name: String,
4 }
5
6 fn chonkiest_cat<'a>(x: &'a Cat, y: &'a Cat) → &'a Cat {
7     if x.weight > y.weight {
8         x
9     } else {
10        y
11    }
12 }
13
14 fn main() {
15     let x = Cat {weight: 7, name: String::from("Kotlet")};
16     let chonky; // ← uninitialized
17     {
18         let y = Cat {weight: 1, name: String::from("Kotara")};
19         chonky = chonkiest_cat(&x, &y);
20     }
21     println!("{}", chonky.name);
22 }
```

# Wymuszenie zgodności życia zmiennych

```
> rustc cat_with_lifetimes.rs
error[E0597]: `y` does not live long enough
  → cat_with_lifetimes.rs:19:36
   |
19 |         chonky = chonkiest_cat(&x, &y);
   |                                ^^ borrowed value does not live long enough
20 |     }
   |     - `y` dropped here while still borrowed
21 |     println!("{}", chonky.name);
   |                       _____ borrow later used here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0597`.
```

# Arytmetyczna dogrzewka



```
1 let x1: u16 = 1;  
2 let x2 = 2_u16;  
3 println!("{}", x1 - x2);
```



```
1 uint16_t x1 = 1;  
2 uint16_t x2 = 2;  
3 std::cout << x1 - x2 << std::endl;
```

# Arytmetyczna dogrzewka - compiler screams

```
> rustc u16_subtraction_naive.rs
error: this arithmetic operation will overflow
   → u16_subtraction_naive.rs:4:20
   |
4  |     println!("{}", x1 - x2);
   |                        ^^^^^^^ attempt to compute `1_u16 - 2_u16`, which would overflow
   |
= note: `[deny(arithmetic_overflow)]` on by default
```

# Arytmetyczna dogrzewka - zaciemniony kod



```
1 fn subtract(val1: u16, val2: u16) {
2     println!("{}", val1 - val2);
3 }
4
5 fn main() {
6     let x1 = 1;
7     let x2 = 2;
8     subtract(x1, x2);
9 }
```



```
1 void subtract(uint16_t val1, uint16_t val2) {
2     std::cout << val1 - val2 << std::endl;
3 }
4
5 int main() {
6     uint16_t x1 = 1;
7     uint16_t x2 = 2;
8     subtract(x1, x2);
9 }
```

# Arytmetyczna dogrzewka - plusy i minusy rzutowania



```
> rustc u16_subtraction.rs && ./u16_subtraction
thread 'main' panicked at 'attempt to subtract with overflow', u16_subtraction.rs:2:20
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
> rustc -O u16_subtraction.rs && ./u16_subtraction
65535
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 u16_subtraction.cpp && ./a.out
-1
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 u16_subtraction.cpp && ./a.out
-1
```

# Rozmiar typów i jego wpływ na zachowanie



```
1 fn subtract(val1: u32, val2: u32) {
2     println!("{}", val1 - val2);
3 }
4
5 fn main() {
6     let x1 = 1;
7     let x2 = 2;
8     subtract(x1, x2);
9 }
```



```
1 void subtract(uint32_t val1, uint32_t val2) {
2     std::cout << val1 - val2 << std::endl;
3 }
4
5 int main() {
6     uint32_t x1 = 1;
7     uint32_t x2 = 2;
8     subtract(x1, x2);
9 }
```



# Rozmiar typów i jego wpływ na zachowanie



```
> rustc u32_subtraction.rs && ./u32_subtraction
thread 'main' panicked at 'attempt to subtract with overflow', u32_subtraction.rs:2:20
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
> rustc -O u32_subtraction.rs && ./u32_subtraction
4294967295
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 u32_subtraction.cpp && ./a.out
4294967295
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 u32_subtraction.cpp && ./a.out
4294967295
```

???



```
1 fn multiply(val1: u16, val2: u16) {
2     println!("{}", val1 * val2);
3 }
4
5 fn main() {
6     let x1 = 0xffff;
7     let x2 = 0xffff;
8     multiply(x1, x2);
9 }
```



```
1 void multiply(uint16_t val1, uint16_t val2) {
2     std::cout << val1 * val2 << std::endl;
3 }
4
5 int main() {
6     uint16_t x1 = 0xffff;
7     uint16_t x2 = 0xffff;
8     multiply(x1, x2);
9 }
```

???



```
> rustc u16_multiplication.rs && ./u16_multiplication
thread 'main' panicked at 'attempt to multiply with overflow', u16_multiplication.rs:2:20
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
> rustc -O u16_multiplication.rs && ./u16_multiplication
1
```



```
> g++ -Wall -Wextra -pedantic -Werror -std=c++20 -O2 -fsanitize=undefined u16_multiplication.cpp && ./a.out
u16_multiplication.cpp:5:25: runtime error: signed integer overflow: 65535 * 65535 cannot be represented in type 'int'
-131071
```

# Usual arithmetic conversions - C++

Many binary operators that expect operands of arithmetic or enumeration type cause conversions and yield result types in a similar way. The purpose is to yield a common type, which is also the type of the result. This pattern is called the usual arithmetic conversions, which are defined as follows:

- If either operand is of scoped enumeration type, no conversions are performed; if the other operand does not have the same type, the expression is ill-formed.
- If either operand is of type `long double`, the other shall be converted to `long double`.
- Otherwise, if either operand is `double`, the other shall be converted to `double`.
- Otherwise, if either operand is `float`, the other shall be converted to `float`.
- Otherwise, the **integral promotions** shall be performed on both operands.

# Integral promotion rules - excerpt - C++

A prvalue of an integer type other than `bool`, `char8_t`, `char16_t`, `char32_t`, or `wchar_t` whose integer conversion rank is less than the rank of `int` can be converted to a prvalue of type `int` if `int` can represent all the values of the source type; otherwise, the source prvalue can be converted to a prvalue of type `unsigned int`.

**GONNA TELL MY KIDS THIS IS BJARNE STROUSTRUP WITH THE ORIGINAL VERSION OF THE C++ STANDARD**



# Jak to wygląda w "rzeczywistości"?



```
1 fn main() {
2     let buffer = ["x"; 50];
3     for j in 0..9 {
4         println!("{}", j * 0x20000001);
5         if buffer[0] == "y" {break;}
6     }
7 }
```



```
1 #include <iostream>
2
3 int main() {
4     char buffer[50] = "y";
5     for (int j = 0; j < 9; ++j) {
6         std::cout << (j * 0x20000001) << std::endl;
7         if (buffer[0] == 'x') break;
8     }
9 }
```

# Do czego kompilatorowi UB?

- Kompilator zakłada, że przekroczenie licznika nigdy nie nastąpi:

```
1 for (int p = 0; p < 9 * 0x20000001; p += 0x20000001) {  
2     std::cout << p << std::endl;  
3     if (buf[0] == 'x') break;  
4 }
```

- `p < 9 * 0x20000001` jest zamienione na `true`, ponieważ `INT_MAX < 9 * 0x20000001`, a `p` nie może być większe niż `INT_MAX`

# Referencje/materiały

- <https://doc.rust-lang.org/book/>
- <https://doc.rust-lang.org/std/>
- <https://medium.com/nearprotocol/understanding-rust-lifetimes-e813bcd405fa>
- [https://shafik.github.io/c++/2021/12/30/usual\\_arithmetic\\_confusions.html](https://shafik.github.io/c++/2021/12/30/usual_arithmetic_confusions.html)