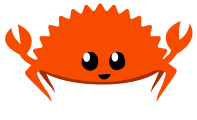
VISUALIZING OWNERSHIP AND BORROWING IN RUST PROGRAMS

Christian Schott

22.05.2024



Rust

A language empowering everyone
to build reliable and efficient software.

GET STARTED

Version 1.78.0

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.

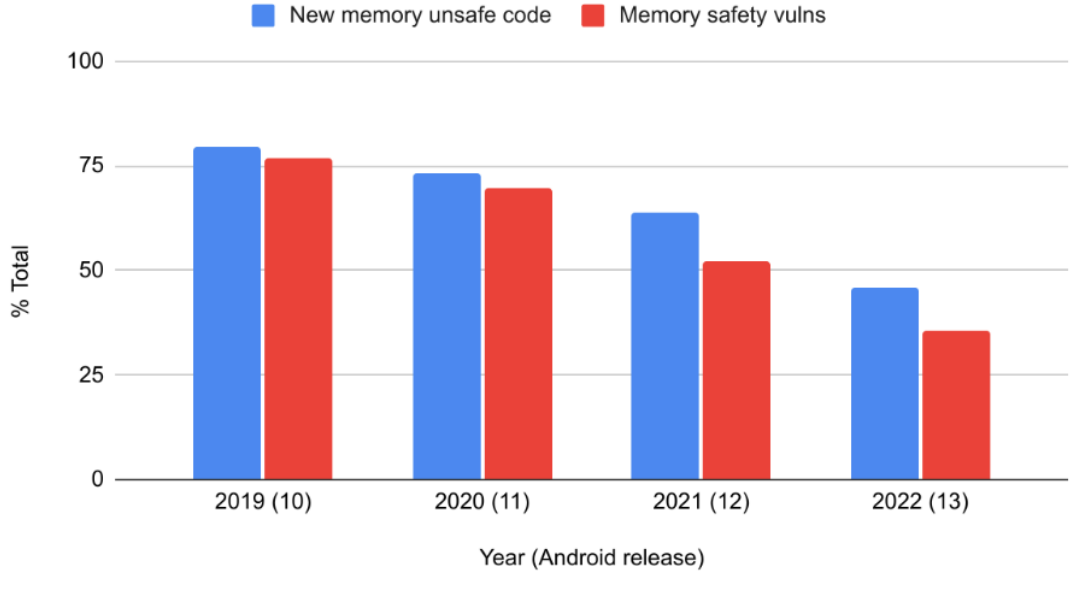
Productivity

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.

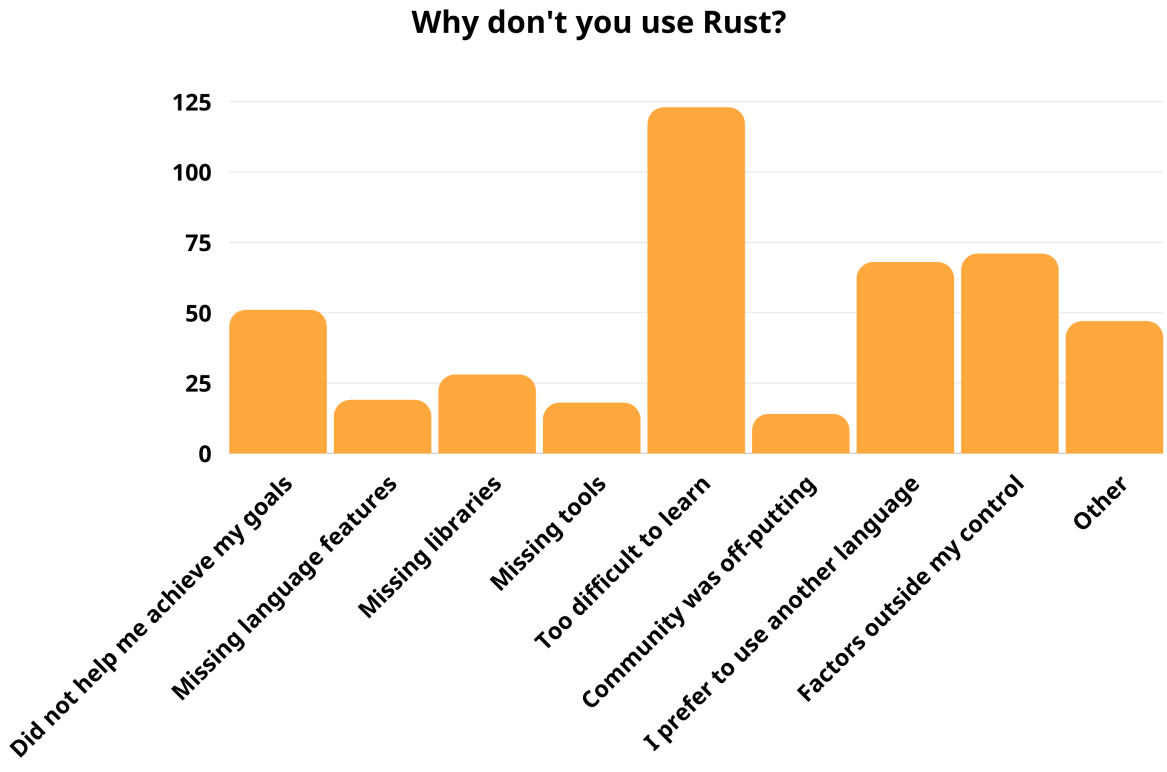
https://www.rust-lang.org/

SAFETY

- 2019 Microsoft: ~70% of CVEs memory related
- Memory vulnerabilities in Android

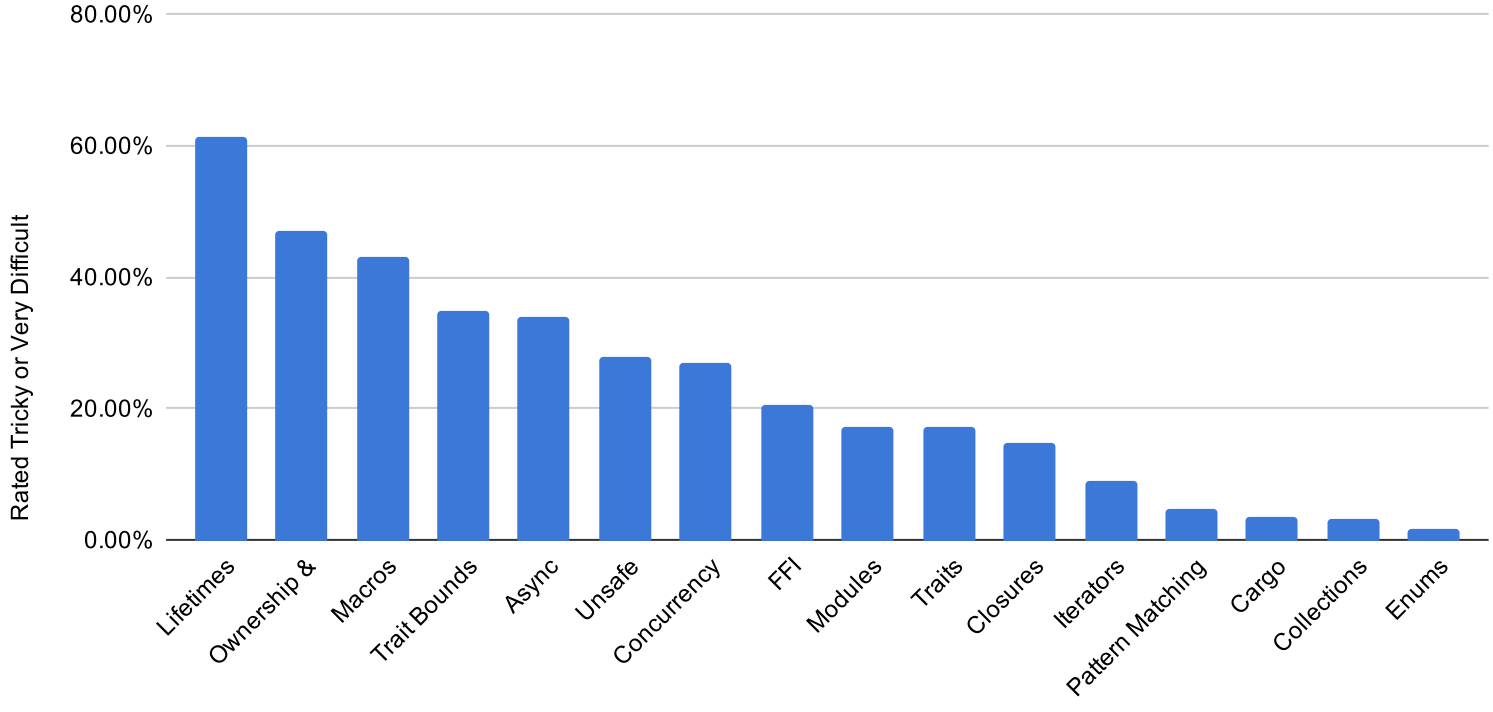


1. https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/
2. https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html



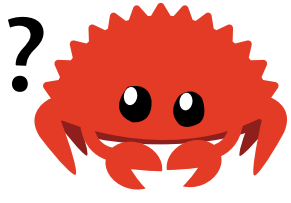
https://blog.rust-lang.org/2023/08/07/Rust-Survey-2023-Results.html

WHY IS IT DIFFICULT?



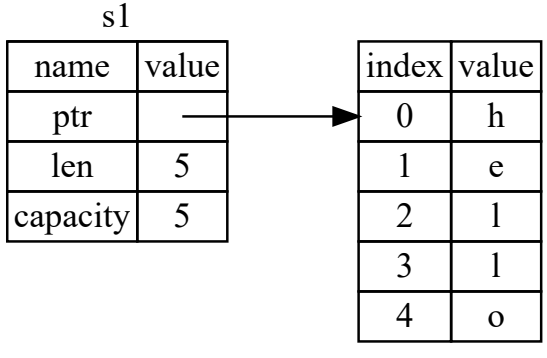
https://blog.rust-lang.org/2020/12/16/rust-survey-2020.html

OWNERSHIP AND LIFETIMES



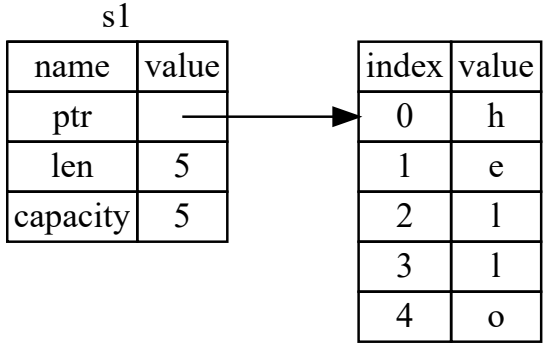
OWNERSHIP

```rust
let s1 = String::from("hello");
```



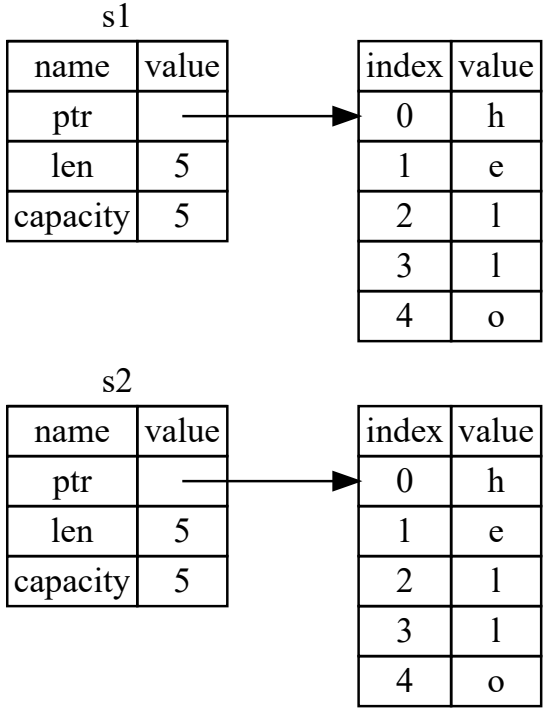
OWNERSHIP

- Each value has **one** *owner*
- Owner scope ends $\longrightarrow$ value is dropped (C++ RAII)



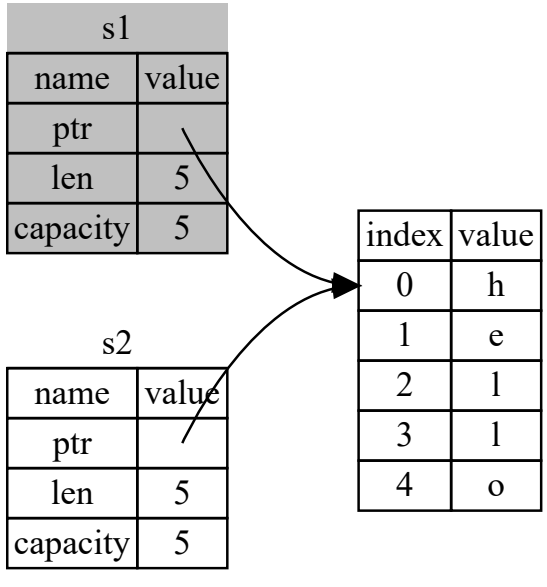
OWNERSHIP IN C++

```
auto s1 = std::string("hello");
auto s2 = s1;
```



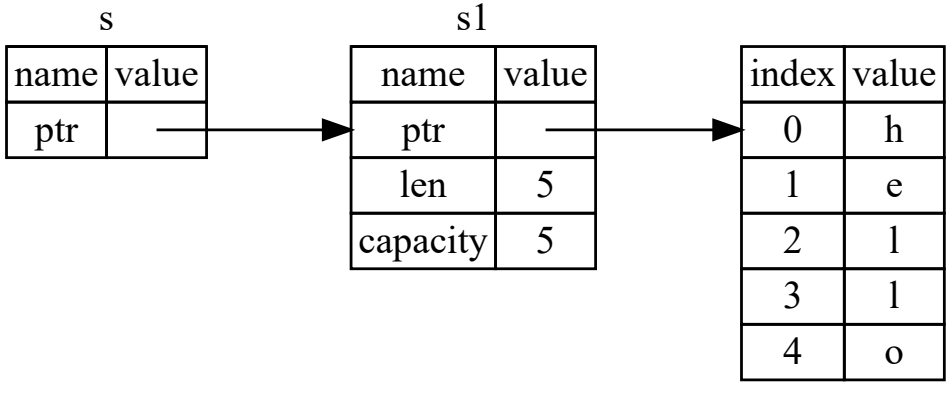
OWNERSHIP

```rust
let s1 = String::from("hello");
let s2 = s1;                        // std::move(s1) in C++
```



BORROWING

```rust
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```



Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

BORROWING RULES

- Values can be borrowed *mutably* or *immutably*
- The *owner* must **outlive** all *borrows*
- At each point a value may **either** be borrowed
  - *mutably* **once**
  - *immutably* **multiple times**

$\rightarrow$ but why?

LIFETIME ALIASING

```rust
fn main() {
  let mut s1 = String::from("hello world");
  let (left, right) = s1.split_at(5);
  s1.push_str("!");            // -> compile error!
  println!("{}", left);
}
```

VISUALIZE *WHAT* RUST'S BORROW CHECKER *SEES*

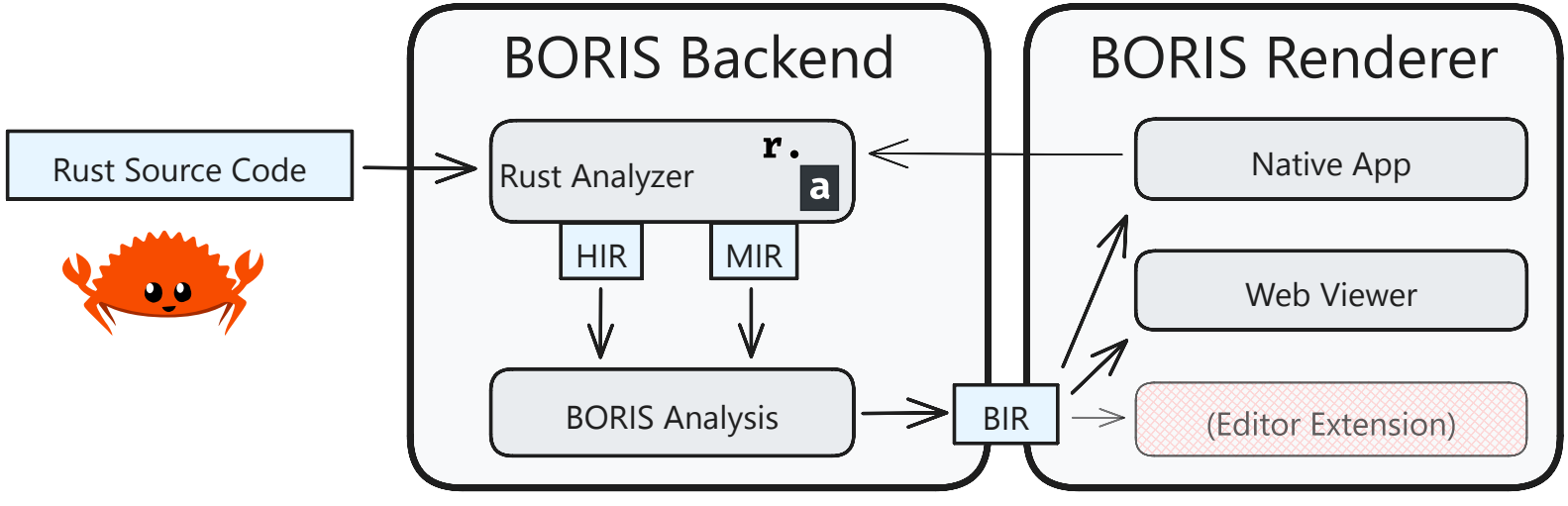
```rust
fn main() {
  let mut s1 = String::from("hello world");
  let (left, right) = s1.split_at(5);
  s1.push_str("!");
  println!("{}", left);
}
```

$\rightarrow$ less cognitive overhead!

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

BORIS

A BORrow vISualizer for Rust programs



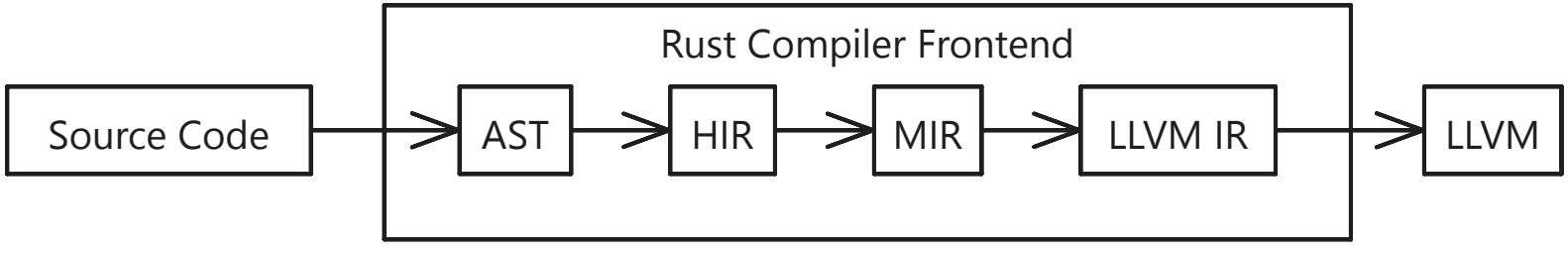
RUSTC

- ✅ ground truth
- ❌ unstable API
- ❌ nightly compiler
- ❌ slow response times

RUST ANALYZER

- ❌ no borrow checker
- ❌ unstable API
- ✅ already running
- ✅ fast response times

COMPILER FRONTEND



- IR ~ Intermediate Representation
- High- to low-level

SOURCE CODE

```
fn main() {
    let mut x = 0;
    for n in 1..10 {
        if n % 2 == 1 {
            x += n;
        }
    }
}
```

ABSTRACT SYNTAX TREE

```
SOURCE_FILE@0..114
  FN@0..113
    FN_KW@0..2 "fn"
    WHITESPACE@2..3 " "
    NAME@3..7
      IDENT@3..7 "main"
    PARAM_LIST@7..9
      L_PAREN@7..8 "("
      R_PAREN@8..9 ")"
    WHITESPACE@9..10 " "
    BLOCK_EXPR@10..113
      STMT_LIST@10..113
        L_CURLY@10..11 "{"
        WHITESPACE@11..16 "\n    "
        LET_STMT@16..30
```

HIGH-LEVEL INTERMEDIATE REPRESENTATION

```
fn main() {
    let mut x = 0;
    match builtin#lang(into_iter)(
        (1) ..(10) ,
    ) {
        mut <ra@gennew>18 => loop match builtin#lang(next)(
            &mut <ra@gennew>18,
        ) {
            builtin#lang(None) => break,
            builtin#lang(Some)(n) => {
                if ((n) % (2)) == (1) {
                    x += n;
                }
            },
        }
```

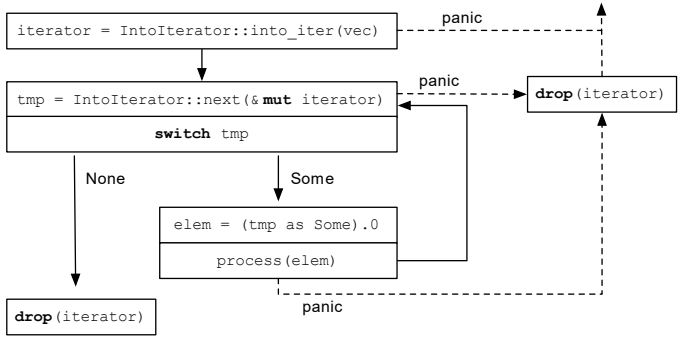
MID-LEVEL INTERMEDIATE REPRESENTATION

```
fn main() {
    let _0: ();
    let x_1: i32;
    let n_2: i32;
    let <ra@gennew>18_3: Range<i32>;
    let _4: i32;
    let _5: Range<i32>;
    let _6: Range<i32>;
    let _7: ();
    let _8: Option<i32>;
    let _9: &mut Range<i32>;
    let _10: &mut Range<i32>;
    let _11: i128;
    let _12: !;
    let _13: bool;
```

BORIS ANALYSIS

- Operates on MIR-level
- Track moves and references of variables

MIR STRUCTURE



MIR STRUCTURE

- Control flow graph of `BasicBlocks`
- `BasicBlock`:
  - sequential statements
  - ending with terminator
- no nested operations

```rust
pub struct MirBody {
    pub basic_blocks: Arena<BasicBlock>,
    pub locals: Arena<Local>,
    pub start_block: BasicBlockId,
    ...
}

pub struct BasicBlock {
    pub statements: Vec<Statement>,
    pub terminator: Terminator,
    ...
}

pub struct Local { pub ty: Ty }
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

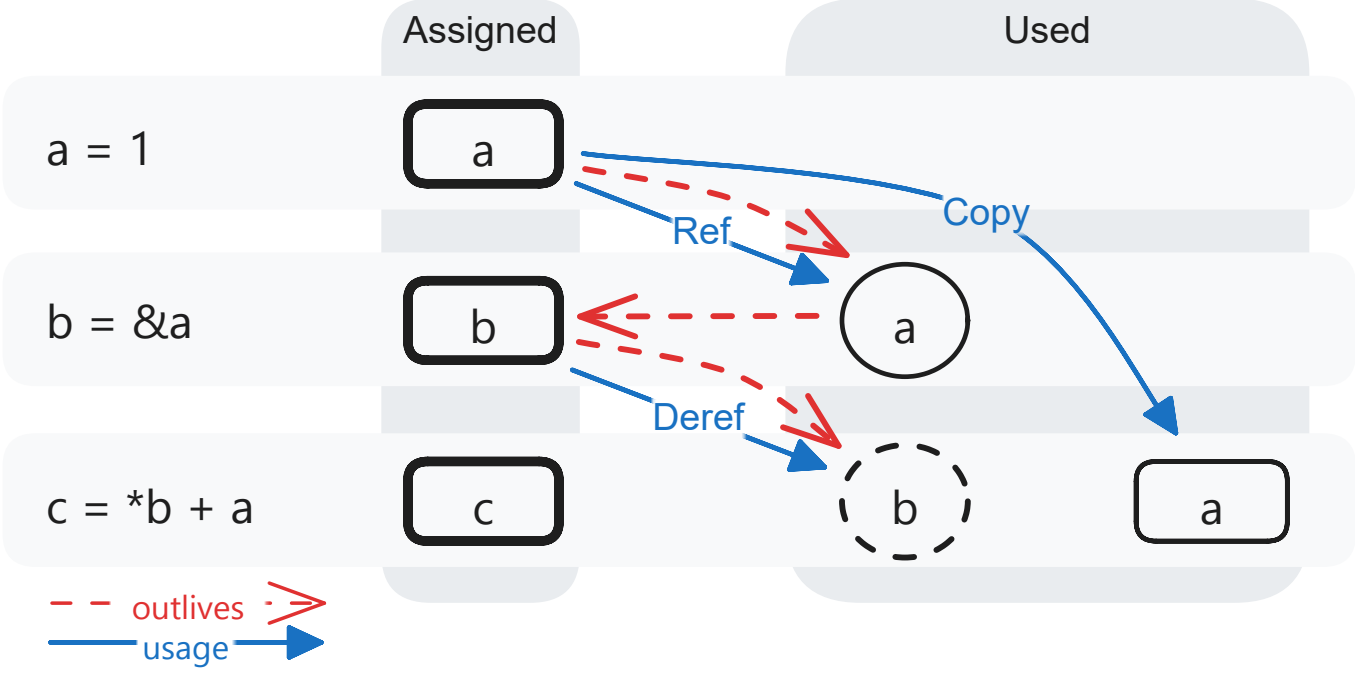
BORIS ANALYSIS

```
let a = 1;
let b = &a;
let c = *b + a;
```

- traverse `BasicBlocks`
- operators and assignees $\longrightarrow$ nodes
- add dependencies

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

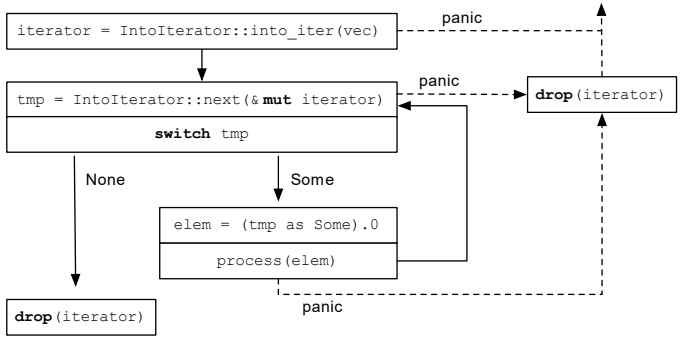
BORIS ANALYSIS



BORIS RENDERER 🖌️

- Lifetime ~ path in MIR graph
- Map MIR-spans to source code

```
let mut iterator = vec.into_iter();
while let Some(elem) = iterator.next() {
    process(elem);
}
```



Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

MAPPING PROBLEM ⚠️

```
let x = String::from("Hello");

if .. {
    take(x);                // x is moved here
    // can not access x anymore for the rest of the block
} else {
    print!("{}", x);        // x is back 'alive' here
}
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

MAPPING PROBLEM ⚠️

```
let mut y = String::from("Hello");
y = {                          // y is reassigned
    println!("{}", y);         // original value of y is accessed
    String::from("World")
};
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

SOLUTION 💡

- Evaluation-order-based rendering of the code
- Based on HIR $\rightarrow$ format independent

SOLUTION 💡

```
fn main() {
 let s = String::from("Hello");
 if condition {  else {
  drop(s);        println!("{s}");
 }               }
}
-> ()
```

```
fn main() {
 let mut x = 42;
     {
      println!("calculating..");
      x + 1
     }
 x =
}
-> ()
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

LIFETIME ANNOTATIONS

```
fn fizzBuzz(n: i32) {
  for i in 1..n + 1 {
    match (i % 3 == 0, i % 5 == 0) {
      (true, true) =>      (true, false) =>  (false, true) =>  _ =>
        println!("FizzBuzz")  println!("Fizz")  println!("Buzz")  println!("{i}")
    }
  }
}
-> ()
```

```rust
pub struct DrawCall {
    pub kind: DrawCallKind,
    pub size: Vec2,
}

pub enum DrawCallKind {
    // primitive
    Text(Arc<Galley>, Color32),
    Rect(Color32, Rounding),

    // compound
    Inline(Box<[RelativeDrawCallId]>),
    Branch(Box<[RelativeDrawCallId]>, ...),
    Sequential(Box<[RelativeDrawCallId]>),
    ...
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

```
fn add(a: i32, b: i32) {
  a + b
}
-> i32
```

LIFETIME ANNOTATIONS

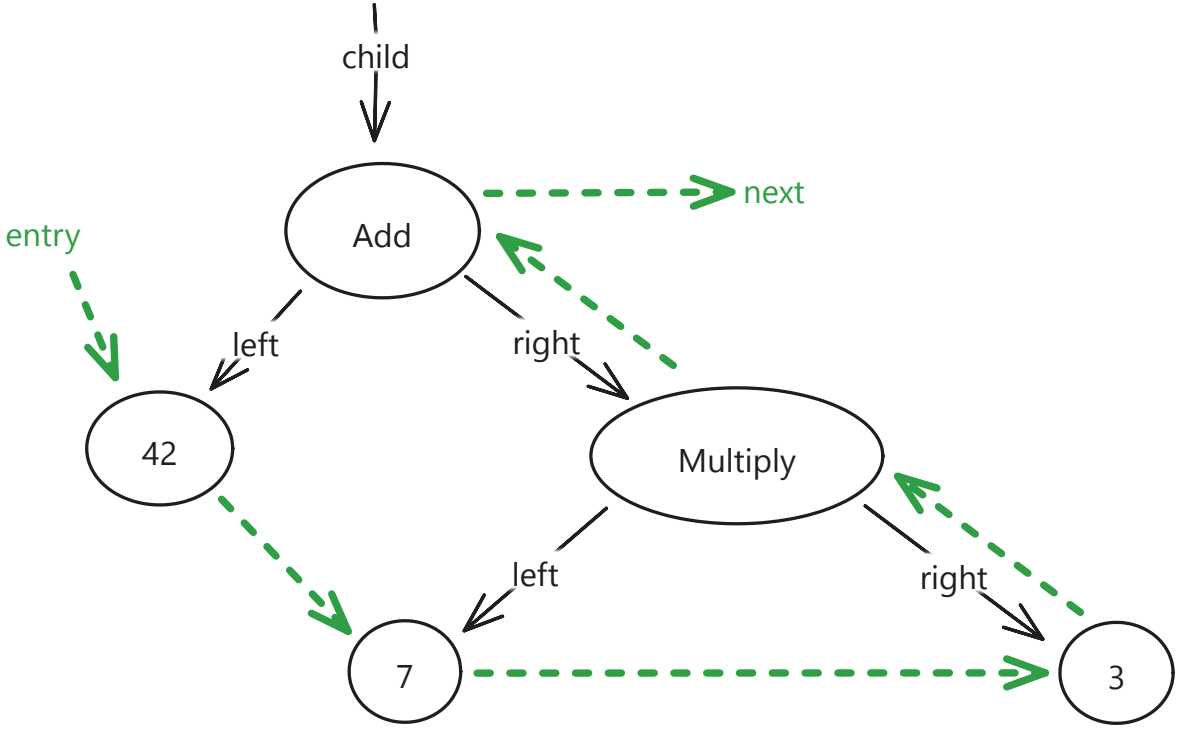
LIFETIME ANNOTATIONS

```
fn main(z: i32) {
  let mut x = String::from("Hello world!");
  let y = 42;
  if y != z {
    println!("{}", x);
  }
  else {
    x = x.to_lowercase();
  }
  drop(x);
}
-> ()
```

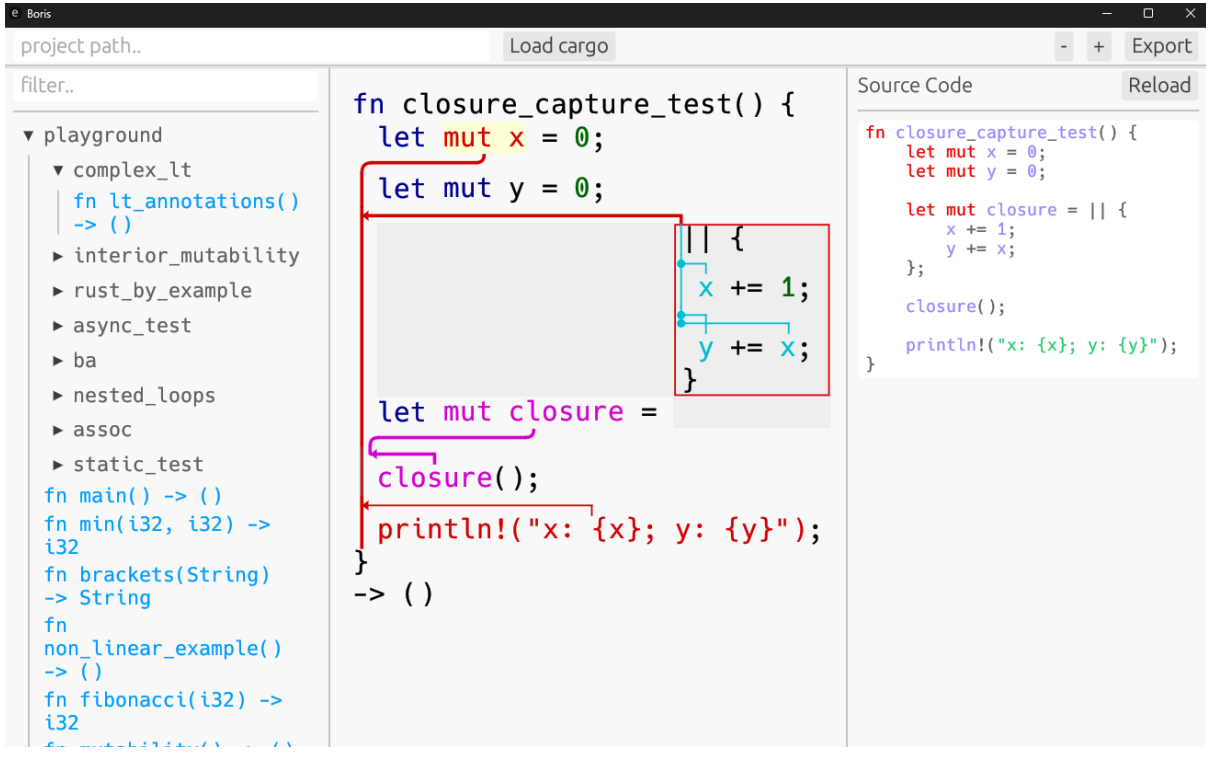
LIFETIME ANNOTATIONS

LIFETIME ANNOTATIONS

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

  let y = x;

  let ref_mut_x = &mut x;

  *ref_mut_x = 1337;
}
-> ()
```


child
next
entry
Add
left
right
42
Multiply
left
right
7
3

BORIS



https://github.com/ChristianSchott/boris

🚧 LIMITATIONS 🚧

- Complex lifetime annotations
- Closure captures
- `unsafe` code (Interior mutability)
- `async` code

COMPLEX LT ANNOTATIONS

```rust
struct Container<'a, 'b> {
    a: &'a str,
    b: &'b str,
}
```

```rust
fn lt_annotations() {
  let hello = String::from("Hello");
  let world = String::from("World");

  let container = Container{a: &hello, b: &world};

  let a_ref = container.a;

  println!("{a_ref}");
}
-> ()
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

CLOSURES

```
let mut count = 0;
let mut inc = || {
    count += 1;
    println!("{}", count);
};
inc();  // 1
inc();  // 2
```

```
struct inc_closure<'a> {
    capture_0: &'a mut i32,
}
impl<'a> std::ops::FnMut<()> for inc_closure<'a> {
    fn call_mut(&mut self, args: ()) -> () {
        *self.capture_0 += 1;
        println!("{}", self.capture_0);
    }
}
```

CLOSURES

```
fn closure_capture_ref() {
  let mut x = 0;
  let mut y = 0;
                        || {
                          x += 1;
                          y += x;
                        }
  let mut closure =
  closure();
  println!("x: {x}; y: {y}");
}
-> ()
```

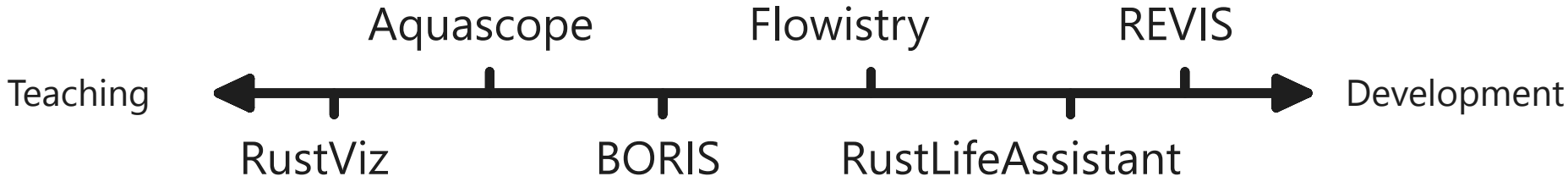
INTERIOR MUTABILITY

```
impl<T> Cell<T> {
    pub fn set(&self, val: T) { .. }
    ..
}
```

```
fn interior_mutability() {
  let ref_cell = RefCell::new(5);

  let shared_ref = ref_cell.borrow();

  *ref_cell.borrow_mut() = 42;

  println!("{}", shared_ref);
}
-> ()
```

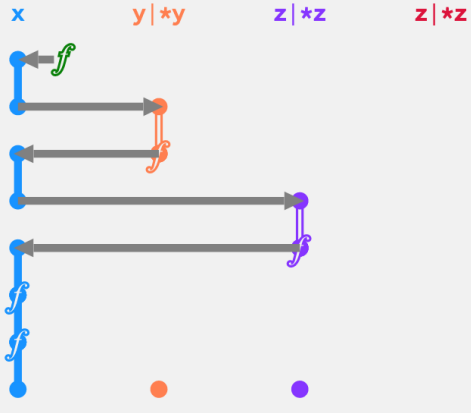
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

RELATED WORKS



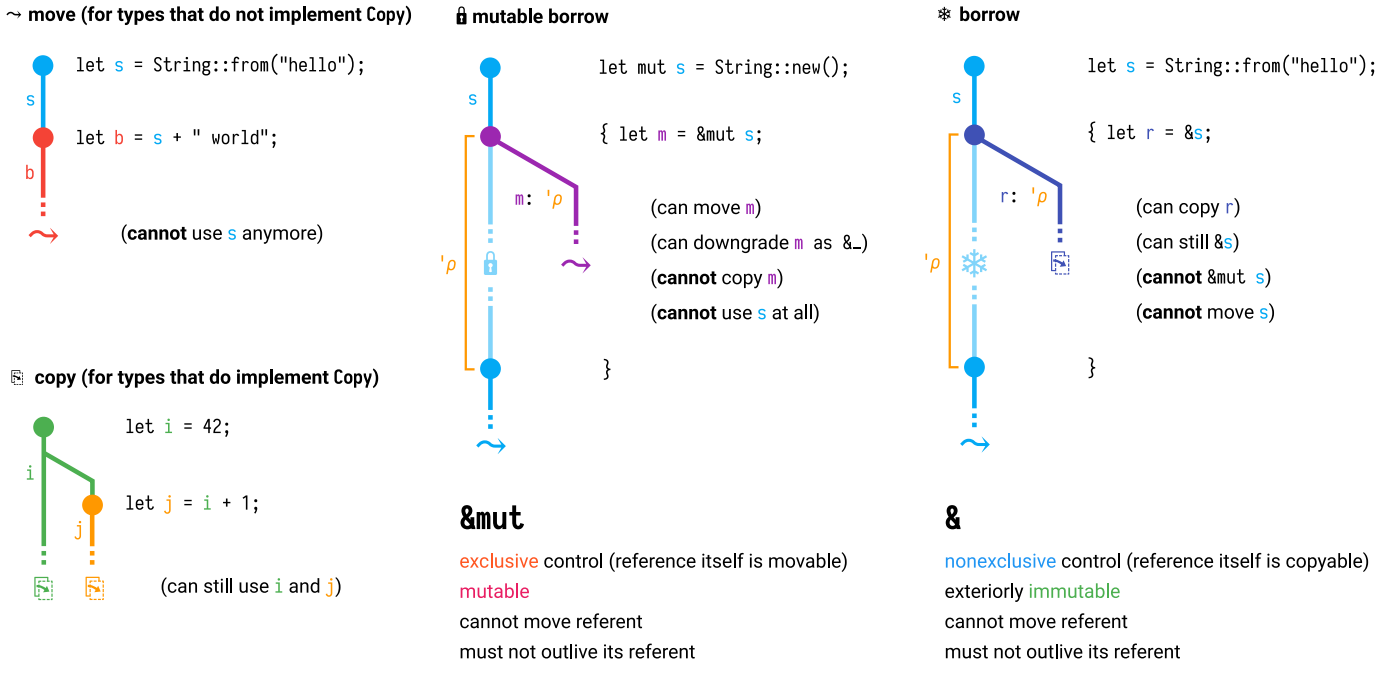
RUSTVIZ

```
1  fn main() {
2      let mut x = String::from("Hello");
3      let y = &mut x;
4      world(y);
5      let z = &mut x; // OK, because y's lifetime has ended (last use was
6      world(z);
7      x.push_str("!!"); // Also OK, because y and z's lifetimes have ended
8      println!("{}", x)
9  }
```



```
/* --- BEGIN Variable Definitions ---
Owner x; Owner y;
Function String::from();
 --- END Variable Definitions --- */
fn main() {
    let x = String::from("hello"); // !{ Move(String::from()-
    let y = x; // !{ Move(x->y) }
    println!("{}", y); // print to stdout!
} /* !{
    GoOutOfScope(x),
    GoOutOfScope(y)
} */
```

VISUAL INSPIRATION

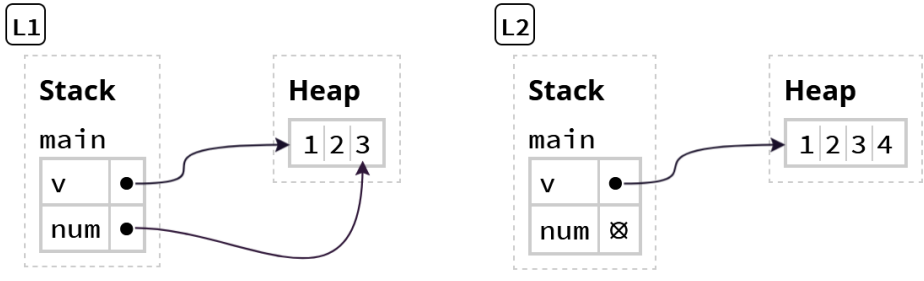


https://rufflewind.com/2017-02-15/rust-move-copy-borrow

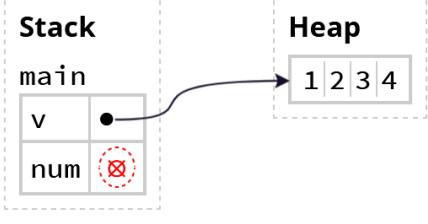
```
fn main() {
  let mut x = String::from("Hello");
  let y = &mut x;
  world(y);
  let z = &mut x;
  world(z);
  x.push_str("!!");
  println!("{}", x)
}
-> ()
```

AQUASCOPE

```rust
let mut v: Vec<i32> = vec![1, 2, 3];
let num: &i32 = &v[2]; L1
v.push(4); L2
println!("Third element is {}", *num); L3
```



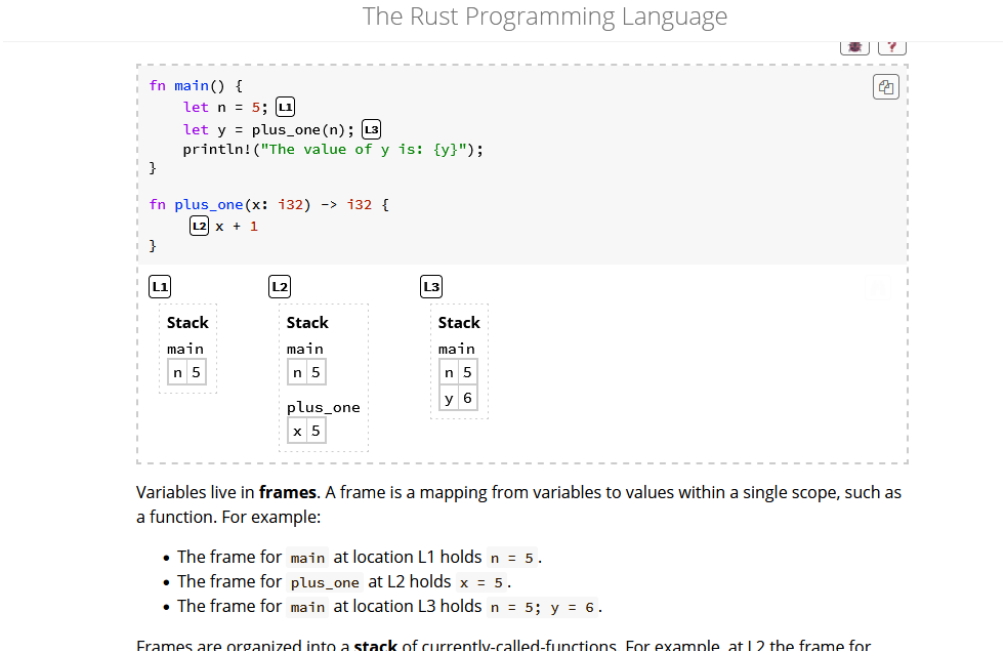
L3 undefined behavior: pointer used after its pointee is freed



```rust
fn ref_aliasing() {
  let mut v: Vec<i32> = vec![1, 2, 3];

  let num: &i32 = &v[2];

  v.push(4);

  println!("Third element is {}", *num);
}
-> ()
```

AQUASCOPE

- improved Ownership chapter in the Rust book
- +9% quiz score (N = 342, d = 0.56)



https://rust-book.cs.brown.edu/ch04-01-what-is-ownership.html

RUST LIFE ASSISTANT

```rust
let mut x = 4;
let y = &x;
let d = &x;
let y2 = move || {
    println("{}", y);
};
let y3 = y2;
let e = &d;
let mut g = 5;
let z = bar(&y3);
let f = &mut g;
let w = foobar(&z);
let mut a = 32;
let b = 42;
let s = &w;
```

Rust compiler error (basically stderr of rustc):

```
error[E0506]: cannot assign to `x` because it is borrowed
  --> /media/david/Daten/Daten/ETH_Studium_Informatik/BSc_Thesis_Rus
   |
3  |     let y = &x;
   |             -- borrow of `x` occurs here
...
20 |     x = 5;
   |     ^^^^^ assignment to borrowed `x` occurs here
...
23 |     take(w);
   |          - borrow later used here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0506`.
```

Possible explanation for "Why is w still borrowing the initial variable?"

1. "y" borrows the initial variable, due to line 3: 'let y = &x;'
2. "y2" borrows "y", due to line 5: 'let y2 = move || {'
3. "y3" borrows "y2", due to line 8: 'let y3 = y2;'
4. "z" borrows "y3", due to line 11: 'let z = bar(&y3);'
5. "w" borrows "z", due to line 13: 'let w = foobar(&z);'
6. "w" is later used

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

```
fn main() {
  let mut x = 4;

  let y = &x;

  let d = &x;

                move || {
                  println!("{}", y);
                }
  let y2 =

  let y3 = y2;

  let e = &d;
  let mut g = 5;

  let z = bar(&y3);

  let f = &mut g;

  let w = foobar(&z);

  let mut a = 32;
  let b = 42;

  let s = &w;

  let r = s;
  x = 5;

  *f = 42;
  take(g);

  take(w);
}
-> ()
```

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

REVIS

- *Rust Error VISualizer*
- Visual annotations of borrow checker messages

```
fn bindings_move(x: [String; 4]) {   -> lifetime of `x`
    match x {
        a @ [.., _] => (),           -> `x` moved to another variable
        _ => (),
    };
    &x;                              --> use of `x` after being moved
}                                       tip: value cannot be used after being moved
```

REVIS

```
fn binding_move(x: [String; 4]) {
 match x {
  a @ [.., _] =>  _ =>
   ()              ()
 }
 &x
}
-> &'static [String; 4]
```

SURVEY RESULTS

Ownership

In Rust you can bind a value to an owner using the statement:

*let **owner** = value;*

Rules:

- Each value in Rust has an ***owner***.
- There can only be **one** owner at a time.
- When the *owner* goes out of scope, the value will be *dropped* (and its memory freed).

Click on variables to visualize an ownership/ borrowing graph.

```
fn ownership() {
 let x = String::from("Hello world");

 let mine_now = x;
```

https://opnform.com/forms/visualizing-ownership-and-borrowing-in-rust-programs-nseo4z

SURVEY RESULTS

- 94 submissions
  - `users.rust-lang.org`
  - `r/rust`
  - `r/learnprogramming`
  - Advanced Programming WS23/24

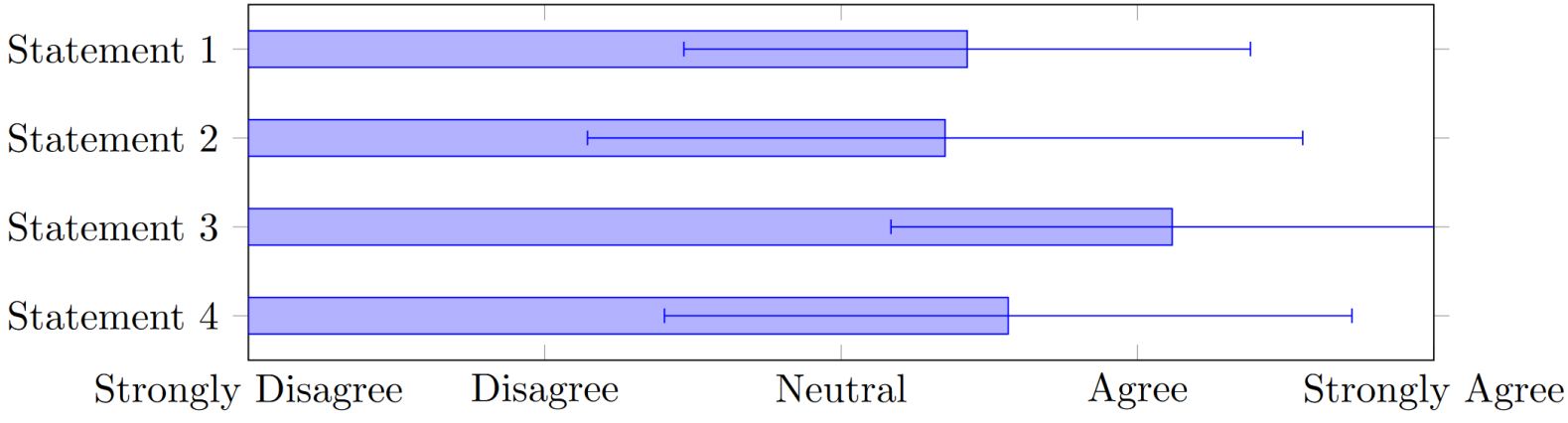
THREATS TO VALIDITY

Name	Total Members	Post Views	#Submissions[1]	Rate[2]
`r/rust`	283.599	26.000	~ 75	$\sim 0.29\%$
`r/learnprogramming`	4.100.000	8.700	~ 5	$\sim 0.057\%$

QUALITATIVE RATING

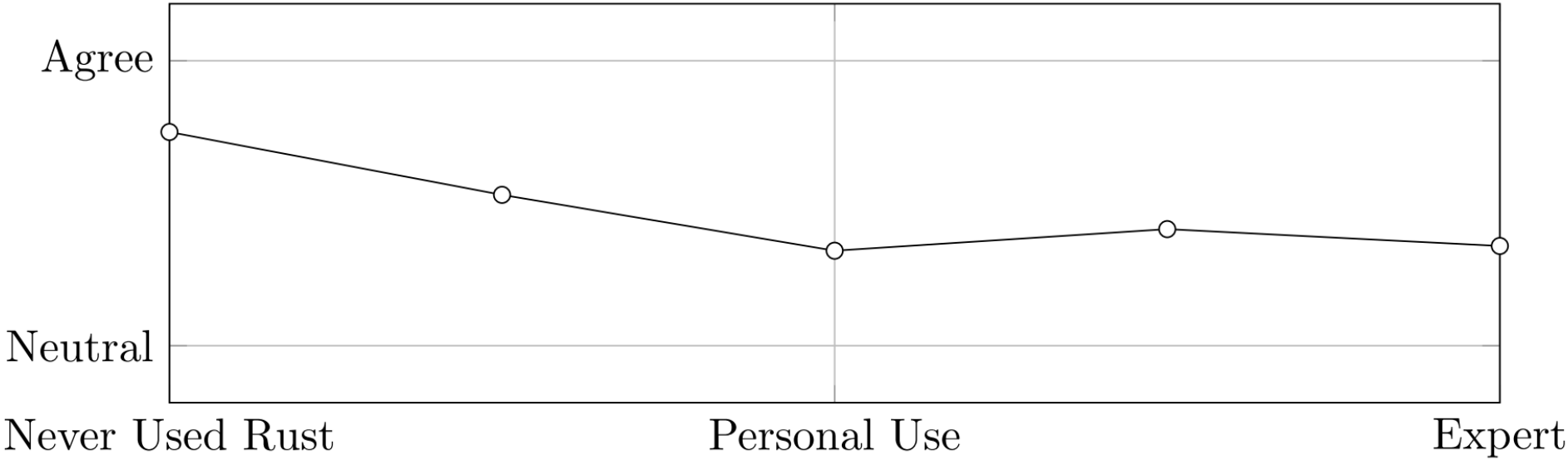
1. They are intuitive to use and grasp
2. They helped you to understand Ownership and Borrowing
3. They could help explain these concepts to beginners
4. You would use such Visualizations during development

QUALITATIVE RATING



1. They are intuitive to use and grasp
2. They helped you to understand Ownership and Borrowing
3. They could help explain these concepts to beginners
4. You would use such Visualizations during development

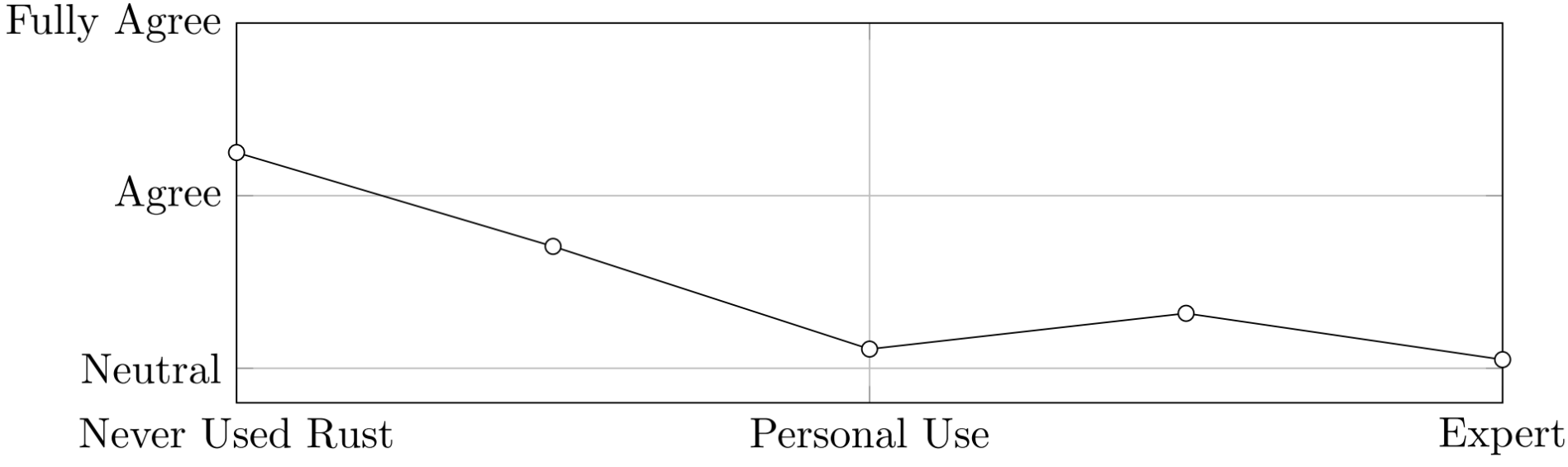
STATEMENT 1: BASED ON RUST EXPERIENCE



They are intuitive to use and grasp

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

STATEMENT 2: BASED ON RUST EXPERIENCE



They helped you to understand Ownership and Borrowing

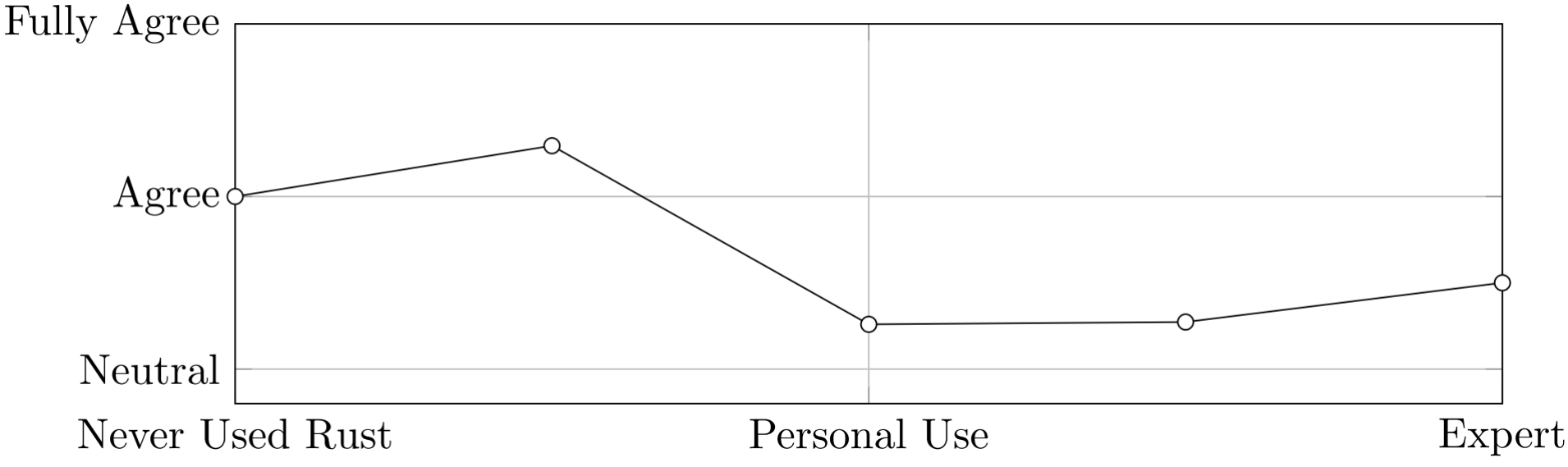
STATEMENT 3

They could help explain these concepts to beginners

- highest overall rating (~4.1)
- no significant impact based on Rust experience

$\rightarrow$ generally perceived as useful for beginners

STATEMENT 4: BASED ON RUST EXPERIENCE



You would use such visualizations during development

CONCLUSION

- improves on previous works
  - branching code
  - automatic generation
- IDE integration needed
- improve readability of annotations
- fix limitations of analysis

THANK YOU FOR LISTENING!

ANY QUESTIONS?