The Rust Borrow Checker
A Deep Dive

Nell Shamrell-Harrington, Mozilla
@nellshamrell

Is the Borrow Checker a friend or a foe?

r/rust

r/rust

Posted by u/dr_eh 1 month ago

15

Newbie question re: the borrow checker

r/rust

15

Posted by u/dr_eh 1 month ago

Newbie question re: the borrow checker

6

Posted by u/speaced 21 days ago

help fighting the borrow checker

r/rust

Posted by u/dr_eh 1 month ago

15

Newbie question re: the borrow checker

Posted by u/speaced 21 days ago

6

help fighting the borrow checker

Posted by u/JeamBim 3 months ago

33

[Beginner] Does it ever get easier? - 'fighting with the borrow checker'

r/rust

117

Posted by u/RecallSingularity 5 months ago ✓

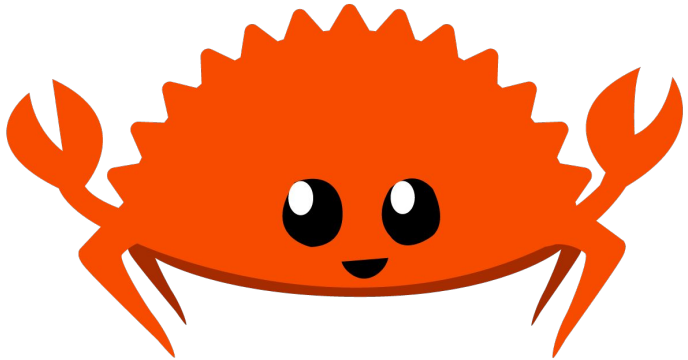
Two years in, the borrow checker exacts no extra cost on me. Coding in Rust is "cheaper" than the alternatives.

The Borrow Checker becomes **your friend** through experience...

...understanding how it works also helps.

Nell Shamrell-Harrington

- Sr. Staff Research Engineer at Mozilla
- Lead Editor of This Week in Rust
- Host of the This Week in Rust podcast on the Rustacean Station
- @nellshamrell



Agenda

- Rust Compiler Overview
- Borrow Checker Deep Dive

Overview of the Rust Compiler

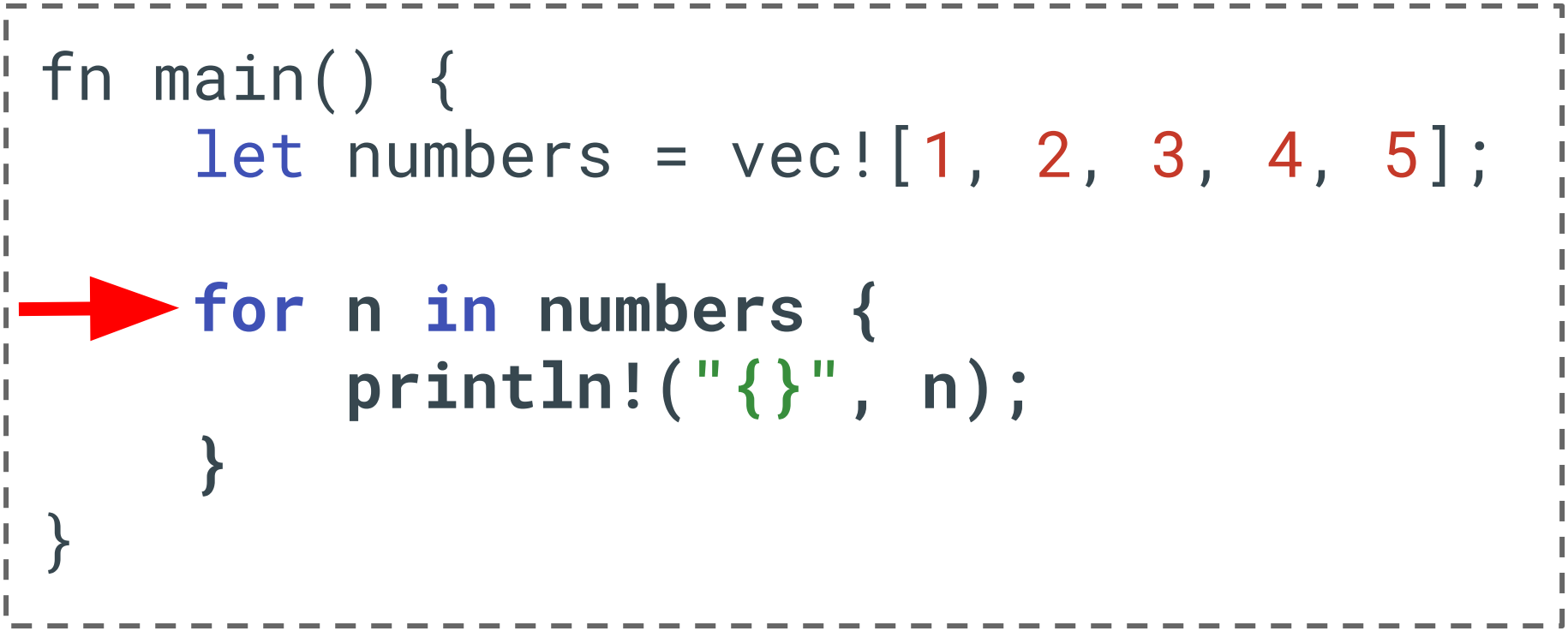
src/main.rs

```
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for n in numbers {
        println!("{}", n);
    }
}
```

src/main.rs

```rust
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for n in numbers {
        println!("{}", n);
    }
}
```

src/main.rs



```rust
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for n in numbers {
        println!("{}", n);
    }
}
```

$ cargo run

```
$ cargo run

1
2
3
4
5
```

Stages of Compilation

- **Lexical Analysis**

Source: "The Imposter's Handbook" by Rob Conery

Stages of Compilation

- Lexical Analysis
- **Parsing**

Source: "The Imposter's Handbook" by Rob Conery

Stages of Compilation

- Lexical Analysis
- Parsing
- **Semantic Analysis**

Source: "The Imposter's Handbook" by Rob Conery

Stages of Compilation

- Lexical Analysis
- Parsing
- Semantic Analysis
- **Optimization**

Source: "The Imposter's Handbook" by Rob Conery

Stages of Compilation

- Lexical Analysis
- Parsing
- Semantic Analysis
- Optimization
- **Code Generation**

Source: "The Imposter's Handbook" by Rob Conery

Wait a minute - isn't the Rust compiler query based?

Yes...but that is out of the scope of this talk.

Check out the "Guide to Rustc Development" for more info!

https://rustc-dev-guide.rust-lang.org/

Stages of Compilation

- **Lexical Analysis**
- Parsing
- Semantic Analysis
- Optimization
- Code Generation

Source: "The Imposter's Handbook" by Rob Conery

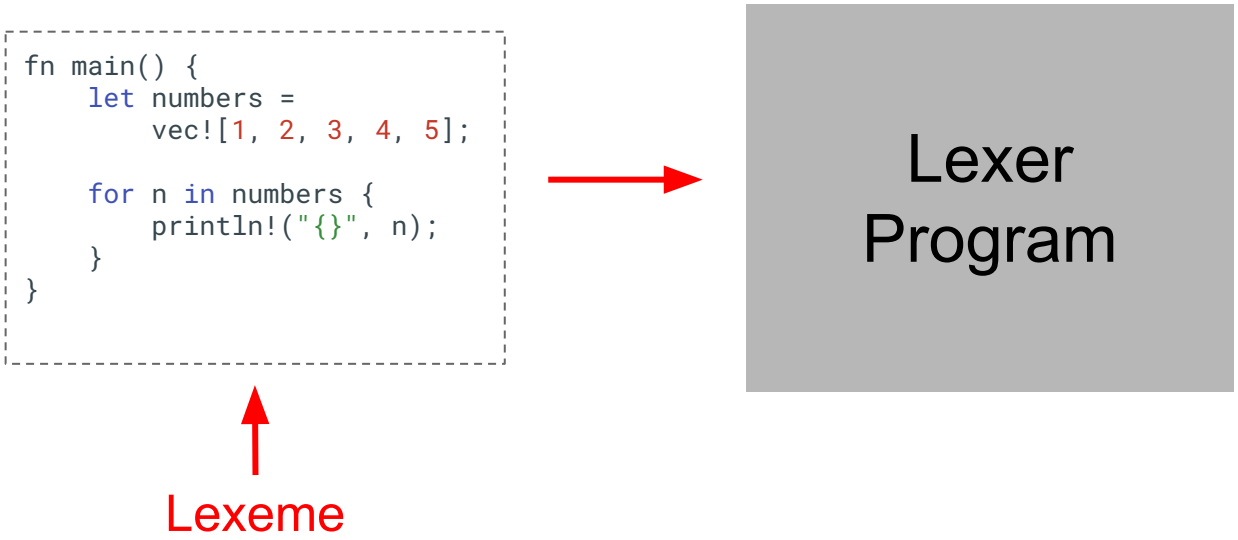
Lexical Analysis

A program called a **lexer**

Source: "The Imposter's Handbook" by Rob Conery

Lexical Analysis

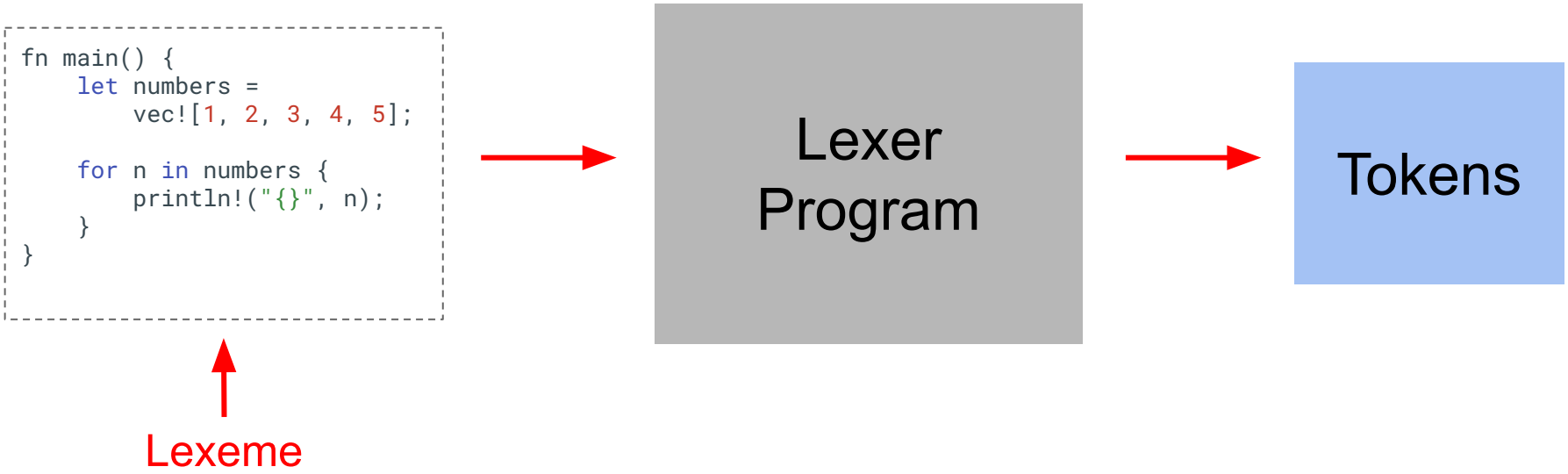
A program called a **lexer** takes the raw source code (called a **lexeme**)



Source: "The Imposter's Handbook" by Rob Conery

Lexical Analysis

A program called a **lexer** takes the raw source code (called a **lexeme**) and splits it into **tokens**



Source: "The Imposter's Handbook" by Rob Conery

Stages of Compilation

- Lexical Analysis
- **Parsing**
- Semantic Analysis
- Optimization
- Code Generation

Source: "The Imposter's Handbook" by Rob Conery

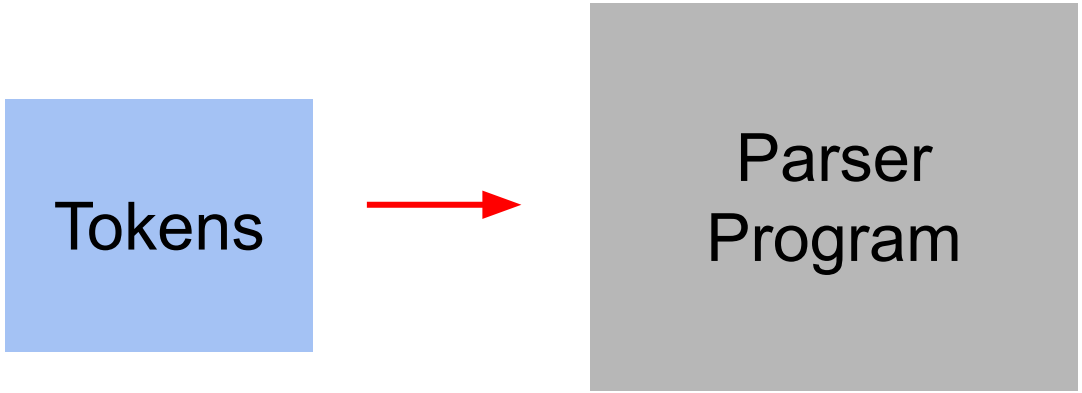
Parsing

A program called a **parser**

Source: "The Imposter's Handbook" by Rob Conery

Parsing

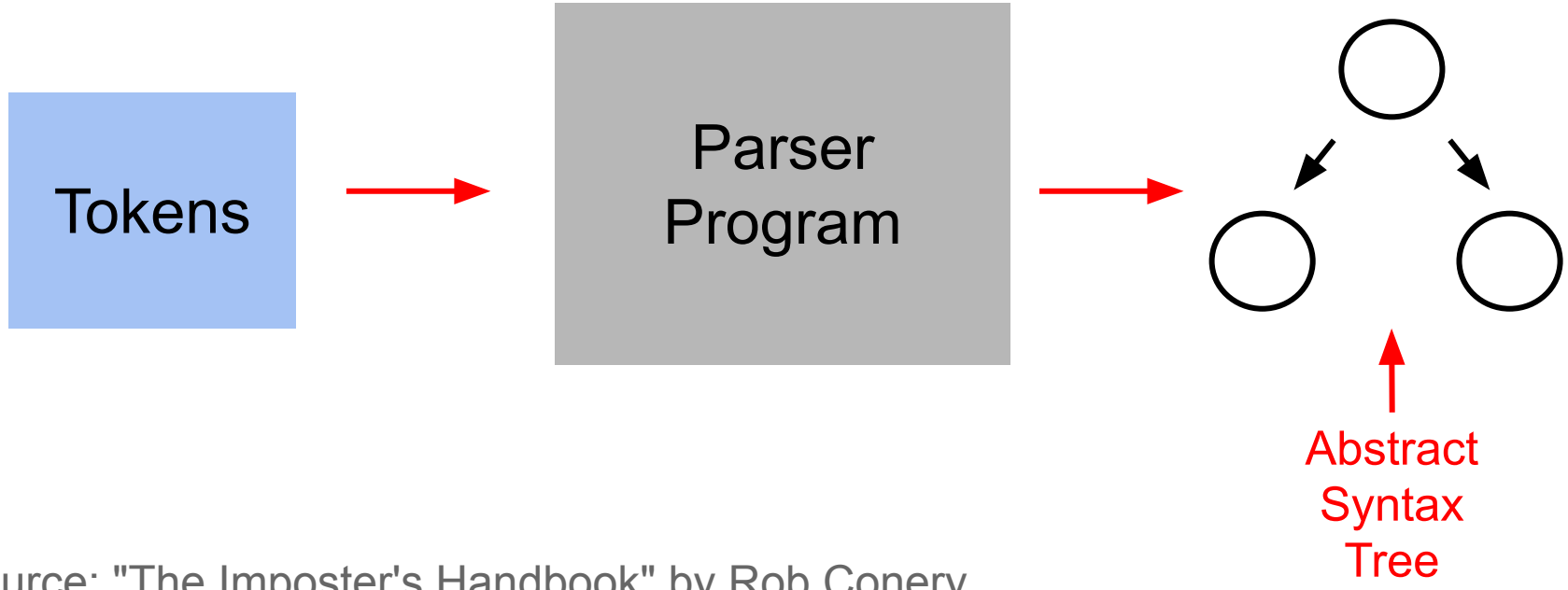
A program called a **parser** analyzes the tokens



Source: "The Imposter's Handbook" by Rob Conery

Parsing

A program called a **parser** analyzes the tokens and translates them into an **abstract syntax tree (AST)**



Source: "The Imposter's Handbook" by Rob Conery

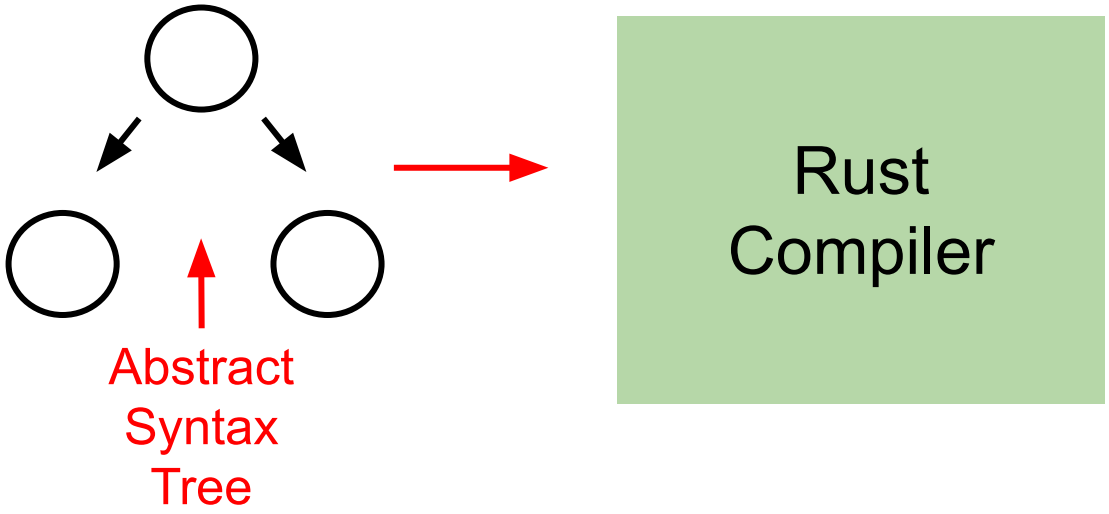
Lowering to HIR

The Rust compiler

Source: "Guide to Rustc Development"

Lowering to HIR

The Rust compiler takes the AST



Source: "Guide to Rustc Development"

And...

- **Expands any macros included in the code**

Source: "Guide to Rustc Development"

src/main.rs

```rust
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for n in numbers {
        println!("{}", n);
    }
}
```

src/main.rs

```
println!("{}", n);
```

src/main.rs

```
::std::io::_print(::core::fmt::Arguments::new_v1(
    &["", "\n"],
    &match (&n,) {
        (arg0,) => [::core::fmt::ArgumentV1::new(
            arg0,
            ::core::fmt::Display::fmt,
        )],
    },
));
```

And...

- Expands any macros included in the code
- **Desugars certain constructs**

Source: "Guide to Rustc Development"

src/main.rs

```
    for n in numbers {

    }
```

src/main.rs

```
    let result = match IntoIterator::into_iter(numbers) {
        mut iter => loop {
            let next;
            match iter.next() {
                Some(val) => next = val,
                None => break,
            };
            (...)
        },
    };
```

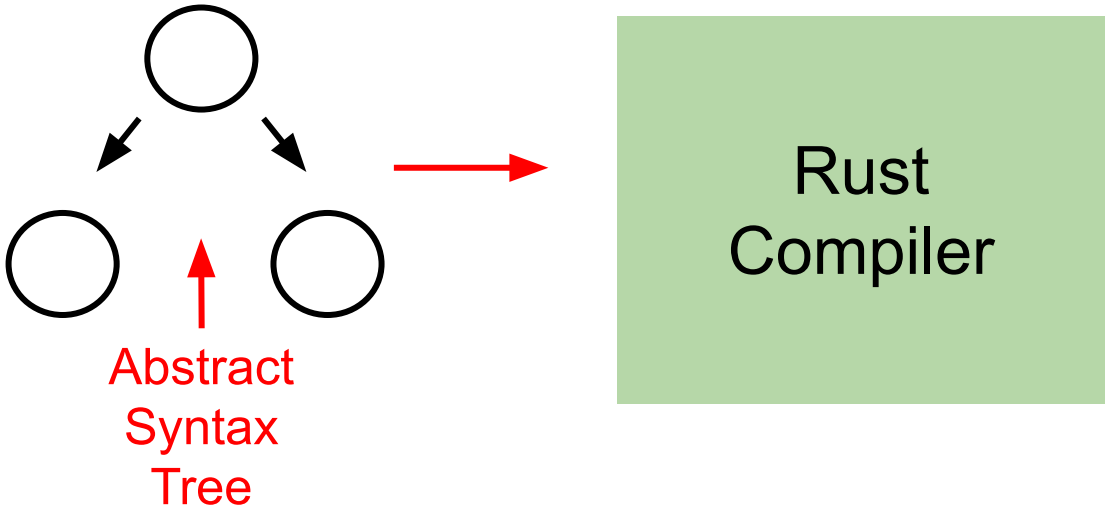
And...

- Expands any macros included in the code
- Desugars certain constructs
- **Resolves any imports in the code**

Source: "Guide to Rustc Development"

Lowering to HIR

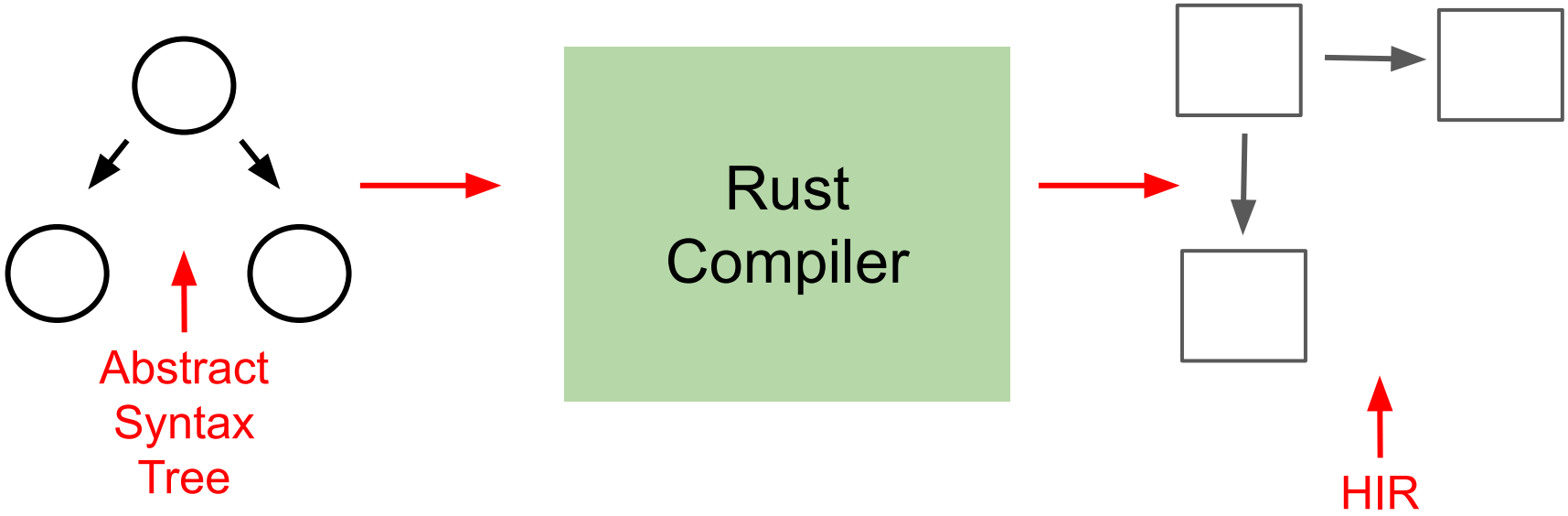
The Rust compiler takes the AST



Source: "Guide to Rustc Development"

Lowering to HIR

The Rust compiler takes the AST and converts it into a **Higher-level Intermediate Representation (HIR)**

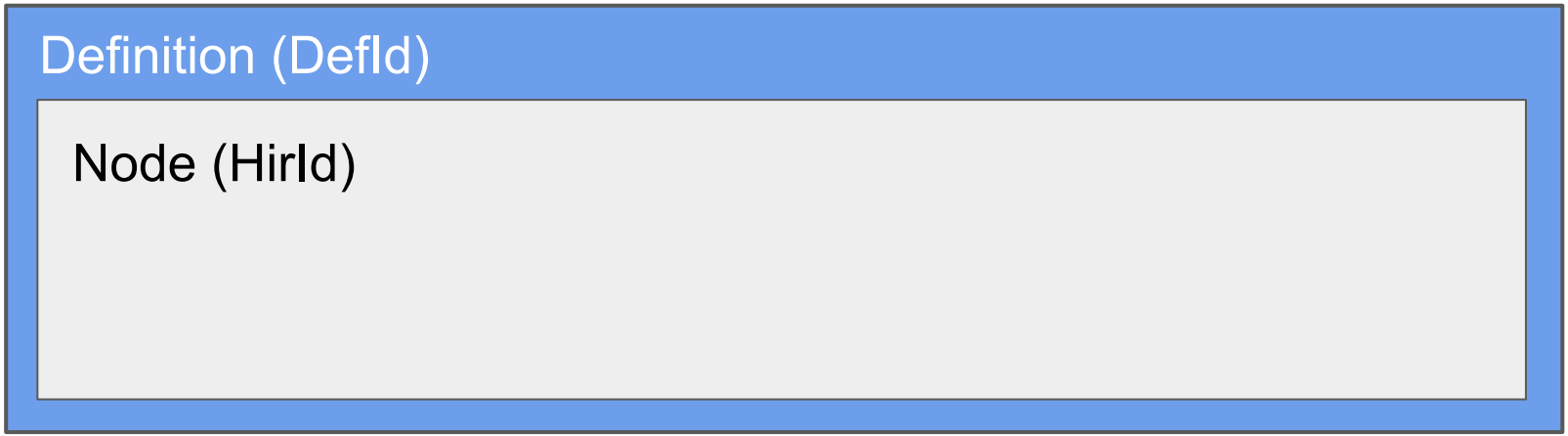


Source: "Guide to Rustc Development"

HIR Data Structures

Source: "Guide to Rustc Development"

HIR Data Structures



Source: "Guide to Rustc Development"

HIR Data Structures

Source: "Guide to Rustc Development"

src/main.rs

```rust
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for n in numbers {
        println!("{}", n);
    }
}
```

src/main.rs

```
    for n in numbers {

    }
```

src/main.rs

```
    let result = match IntoIterator::into_iter(numbers) {
        mut iter => loop {
            let next;
            match iter.next() {
                Some(val) => next = val,
                None => break,
            };
            (...)
        },
    };
```

higher_level_intermediate_representation

```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```

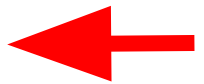
higher_level_intermediate_representation

```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```

Represents an arm of the match expression

higher_level_intermediate_representation

```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```



higher_level_intermediate_representation

```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```

Owned by a definition

higher_level_intermediate_representation

```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```

higher_level_intermediate_representation

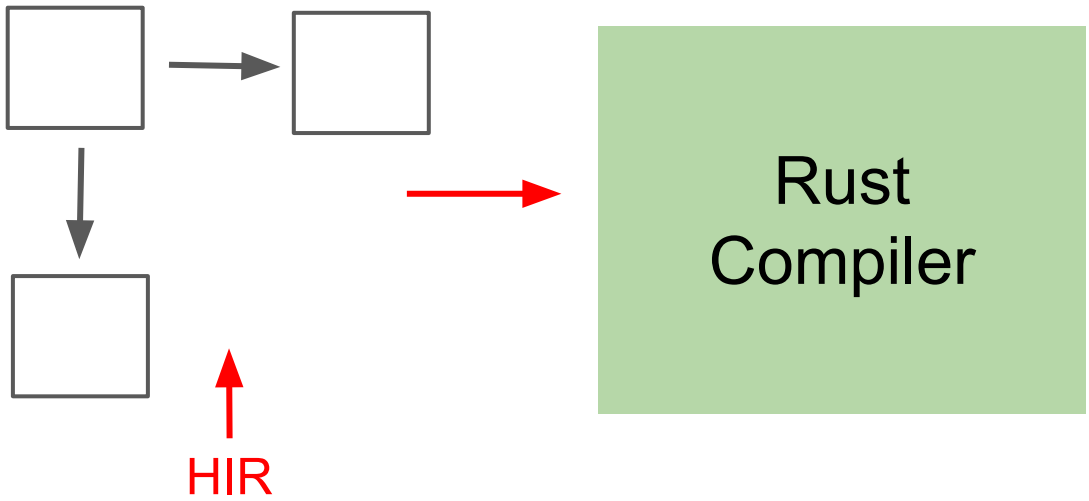
```
Arm {
    hir_id: HirId {
        owner: DefId(0:3 ~
            sample_code_1[24c3]::main[0]),
        local_id: 107,
    },
    span: src/main.rs:4:5: 6:6,
```

Lowering to MIR

Source: "Guide to Rustc Development"

Lowering to MIR

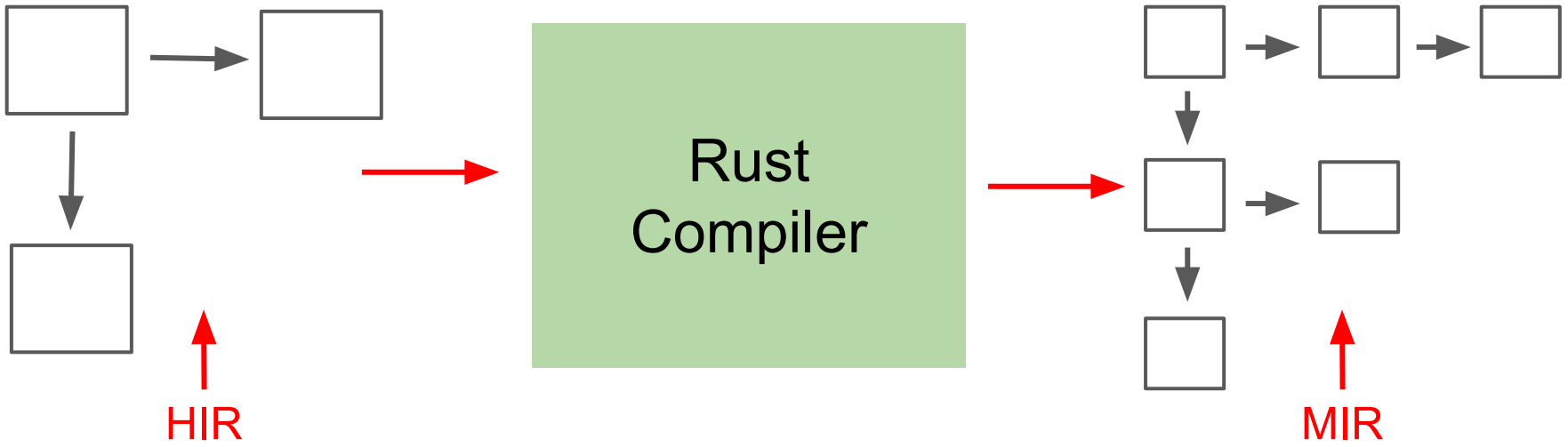
Compiler then lowers the HIR



Source: "Guide to Rustc Development"

Lowering to MIR

Compiler then lowers the HIR into **Mid-level Intermediate Representation (MIR)**



Source: "Guide to Rustc Development"

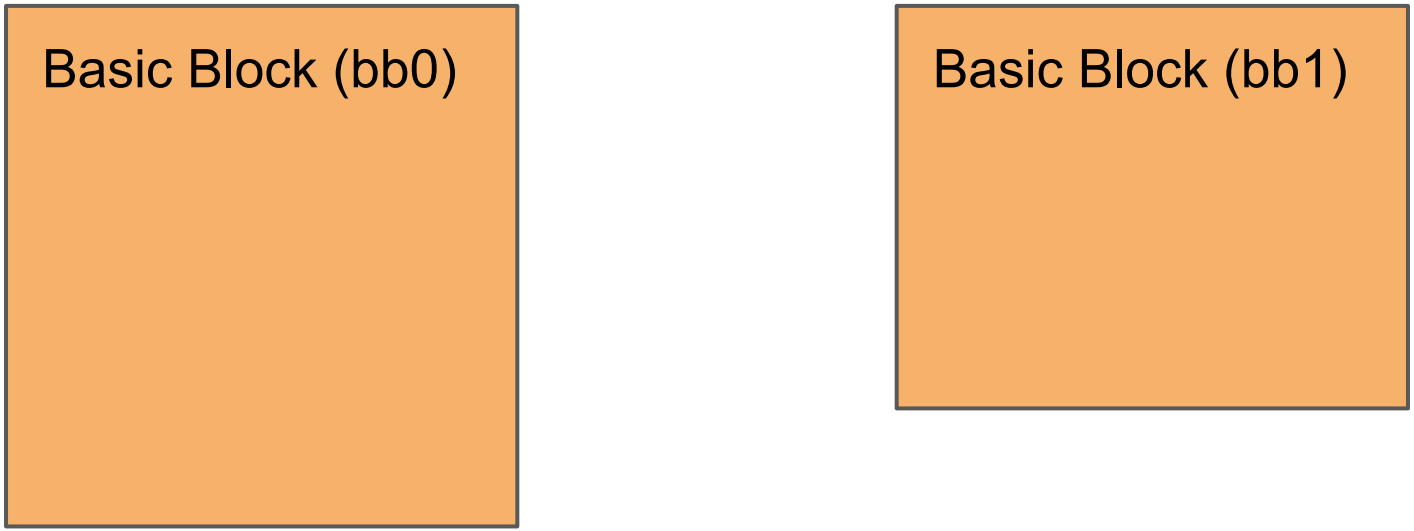
MIR Data Structures

Control Flow Graph

Source: "Guide to Rustc Development"

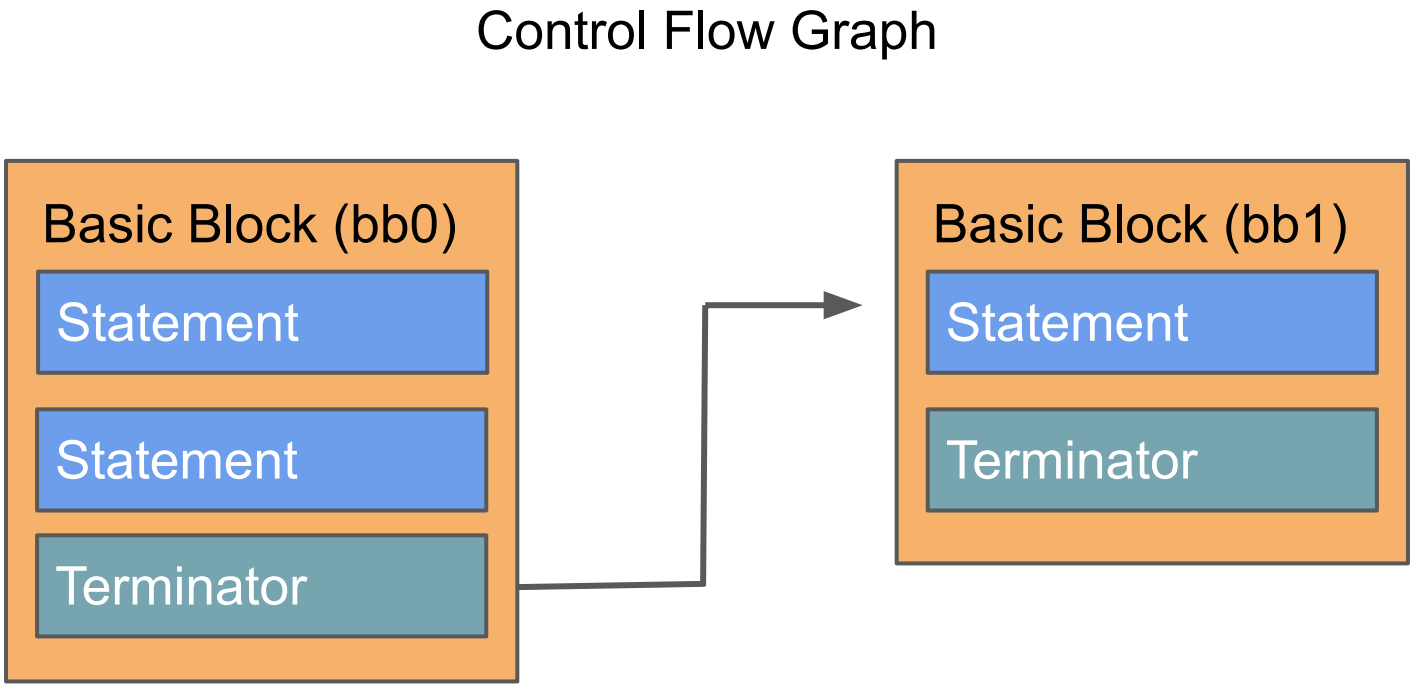
MIR Data Structures

Control Flow Graph



Source: "Guide to Rustc Development"

MIR Data Structures

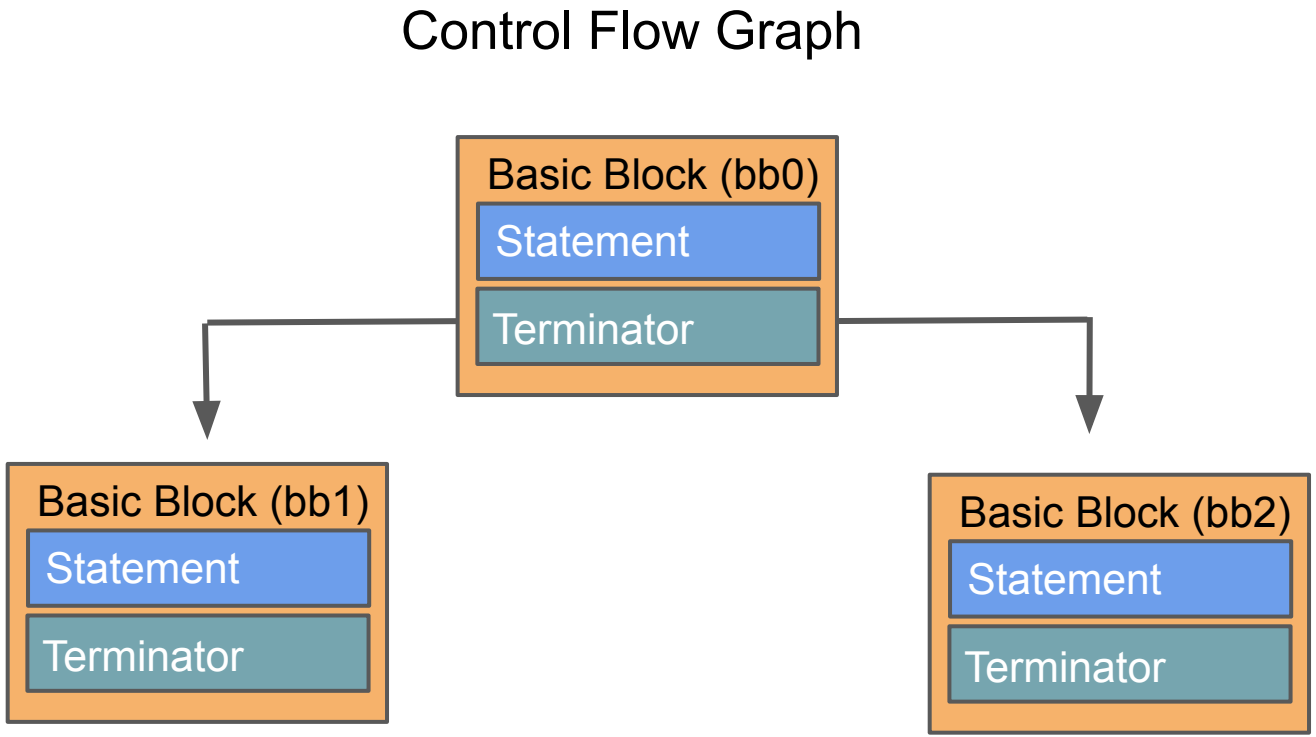


Source: "Guide to Rustc Development"

example.rs

```
    if condition {
        do_one_thing
    } else {
        do_something_else
    }
```

MIR Data Structures



Source: "Guide to Rustc Development"

Are there more MIR data structures?

Check out the "Guide to Rustc Development" for more info!

https://rustc-dev-guide.rust-lang.org/

src/main.rs

```
    let result = match IntoIterator::into_iter(numbers) {
        mut iter => loop {
            let next;
            match iter.next() {
                Some(val) => next = val,
                None => break,
            };
            (...)
        },
    };
```

src/main.rs

```
    let result = match IntoIterator::into_iter(numbers) {




    };
```

mid_level_intermediate_representation

```
bb2: {
    _5 = const <std::vec::Vec<i32> as
std::iter::IntoIterator>::into_iter(move _6)
        span: src/main.rs:4:14: 4:21
```

mid_level_intermediate_representation

```
bb2: {
    _5 = const <std::vec::Vec<i32> as
std::iter::IntoIterator>::into_iter(move _6)
        span: src/main.rs:4:14: 4:21
```

Basic Block

mid_level_intermediate_representation

```
bb2: {
    _5 = const <std::vec::Vec<i32> as
std::iter::IntoIterator>::into_iter(move _6)
                    span: src/main.rs:4:14: 4:21
```

Local (result)

mid_level_intermediate_representation

```
bb2: {
    _5 = const <std::vec::Vec<i32> as
std::iter::IntoIterator>::into_iter(move _6)
        span: src/main.rs:4:14: 4:21
```

Stages of Compilation

- Lexical Analysis
- Parsing
- **Semantic Analysis**
- Optimization
- Code Generation

Source: "The Imposter's Handbook" by Rob Conery

Semantic Analysis

Compiler tries to figure out what the programmer is trying to do in a way the compiler can understand it

Source: "The Imposter's Handbook" by Rob Conery

Semantic Analysis

At this point the Rust compiler will run several checks - including the **borrow checker**

Source: "Guide to Rustc Development"

We'll come back to the borrow checker shortly...

Stages of Compilation

- Lexical Analysis
- Parsing
- Semantic Analysis
- **Optimization**
- **Code Generation**

Source: "Guide to Rustc Development"

Optimization & Code Generation

This is where the code is transformed into an executable binary

Source: "Guide to Rustc Development"

LLVM

LLVM is a collection of modular and re-usable compiler and toolchain technologies

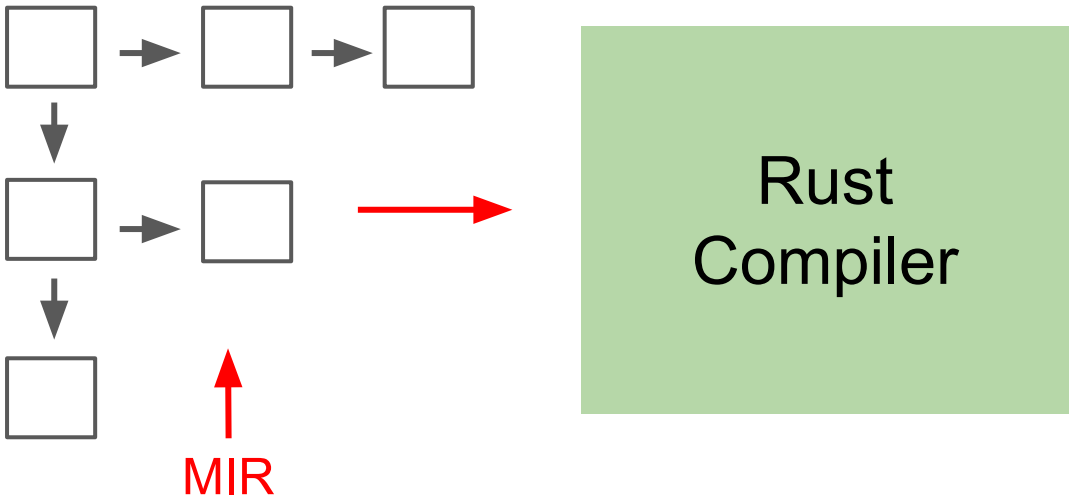
Source: https://llvm.org

Optimization & Code Generation

Source: "Guide to Rustc Development"

Optimization & Code Generation

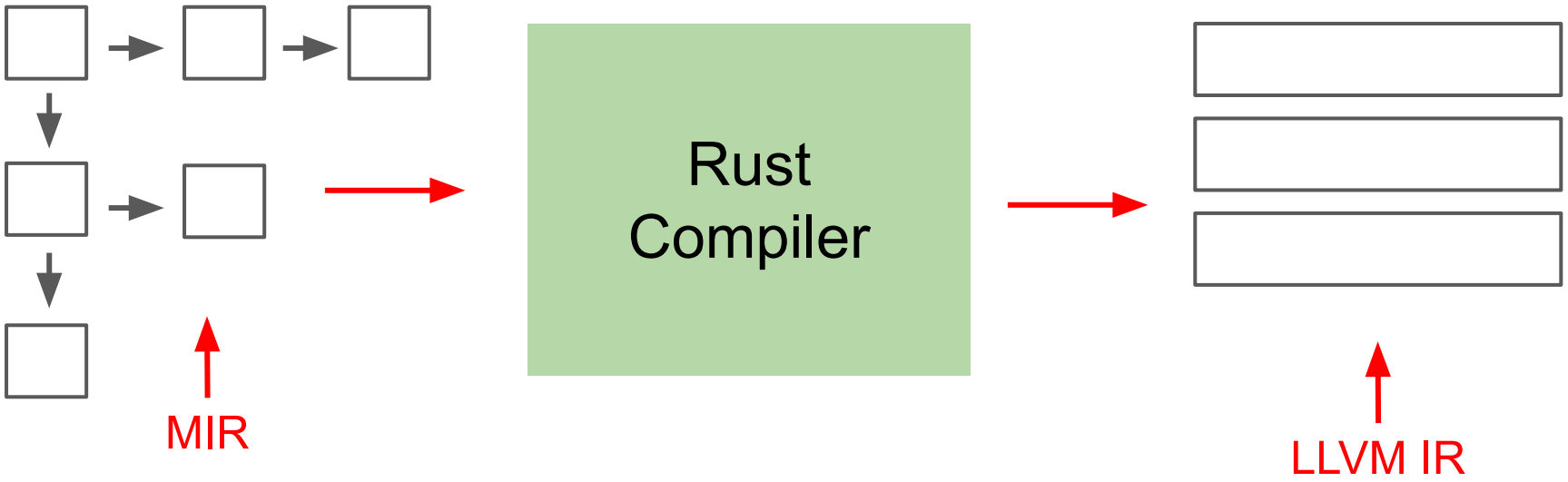
Compiler lowers the MIR



Source: "Guide to Rustc Development"

Optimization & Code Generation

Compiler lowers the MIR into LLVM Intermediate Representation



Source: "Guide to Rustc Development"

llvm_intermediate_representation

```
bb2:
; preds = %bb1
  ret i32 %1, !dbg !194
}
```

llvm_intermediate_representation

```
bb2:
; preds = %bb1
  ret i32 %1, !dbg !194
}
```

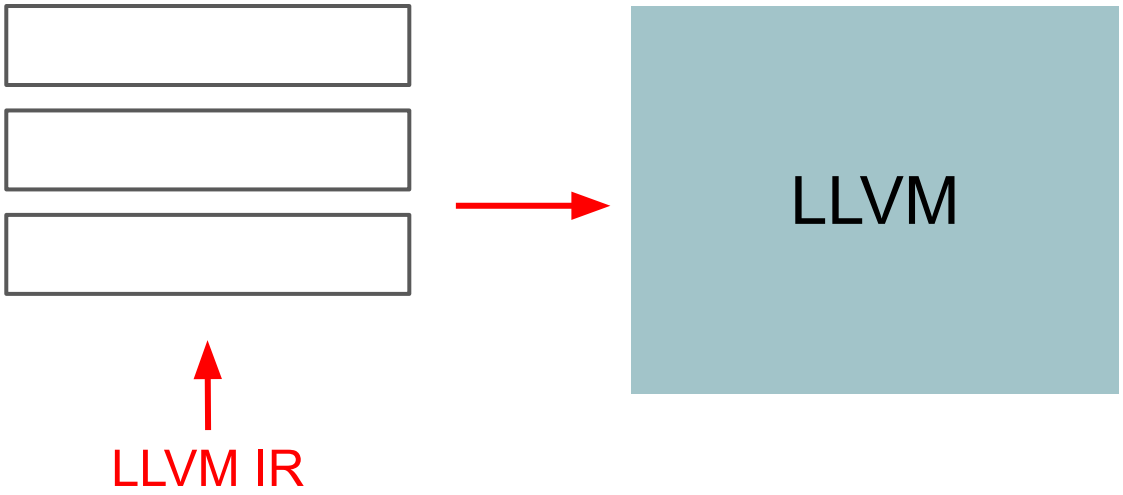
Basic Block

Optimization & Code Generation

Source: "Guide to Rustc Development"

Optimization & Code Generation

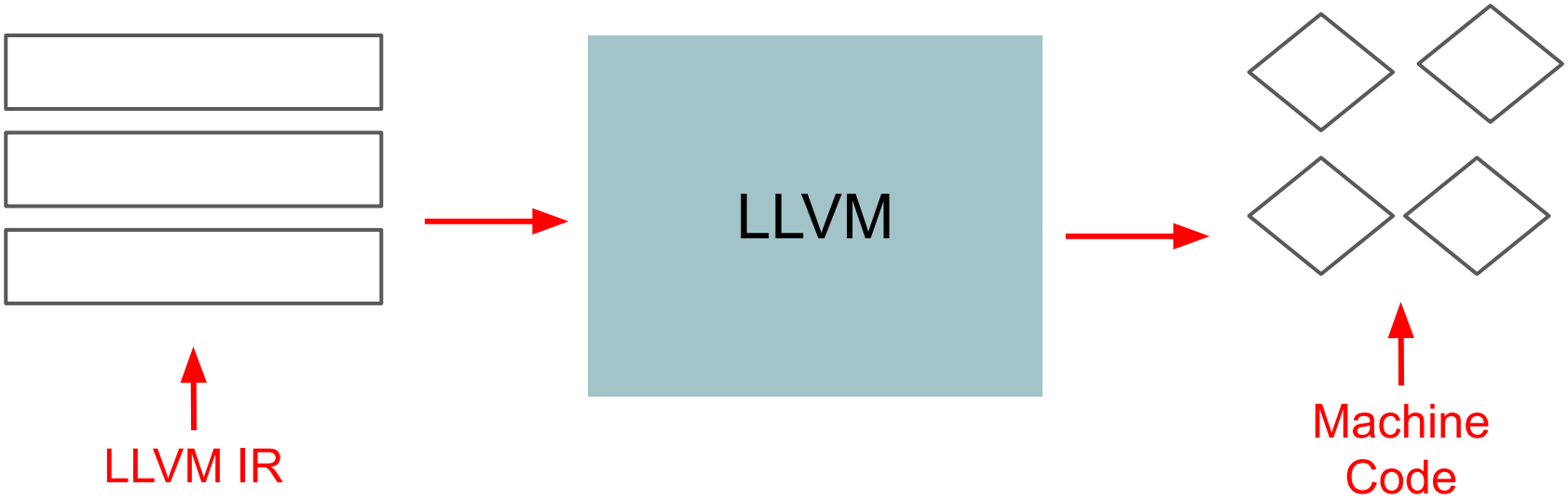
And LLVM takes the LLVM IR



Source: "Guide to Rustc Development"

Optimization & Code Generation

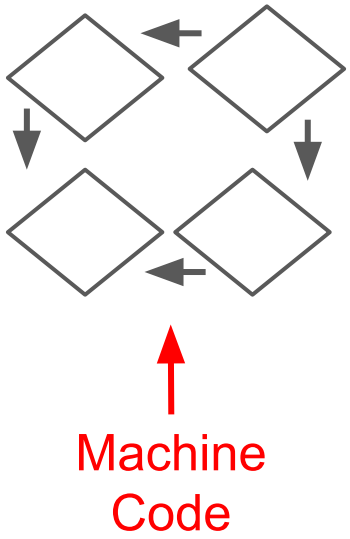
And LLVM takes the LLVM IR, which runs more optimizations on it and emits machine code



Source: "Guide to Rustc Development"

Optimization & Code Generation

LLVM then links the machine code files together to produce the final binary



Source: "Guide to Rustc Development"

$ cargo run

```
$ cargo run

1
2
3
4
5
```

```
$ cargo run

1
2
3
4
5
```

Yay!!!

Back to the borrow checker!

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

$ cargo build

$ cargo build

```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:7:20
5 |     let y = x;
  |             - value moved here
6 |
7 |     println!("{}", x);
  |                    ^ value borrowed here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain
E0382`.
```

Borrow Checker

Does several things including:

- **Tracking initializations and moves**

Source: "Guide to Rustc Development"

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```
let x: String; // x is not initialized here
```

src/main.rs

```
let x: String; // x is not initialized here
```

mid_level_intermediate_representation

```
debug x => _1;
let _1: std::string::String
```

src/main.rs

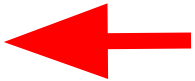
```
let x: String; // x is not initialized here
```

mid_level_intermediate_representation

```
debug x => _1;
let _1: std::string::String
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```



src/main.rs

```
x = String::from("Hi Rusty Days!"); // x is initialized here
```

src/main.rs

```
x = String::from("Hi Rusty Days!"); // x is initialized here
```

mid_level_intermediate_representation

```
_2 = const <String as
        From<&str>>::from(const "Hi Rusty Days!")
_1 = move _2;
```

src/main.rs

```
x = String::from("Hi Rusty Days!"); // x is initialized here
```

mid_level_intermediate_representation

```
_2 = const <String as
          From<&str>>::from(const "Hi Rusty Days!")
_1 = move _2;
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```
let y = x; // x is moved here
```

src/main.rs

```
let y = x; // x is moved here
```

mid_level_intermediate_representation

```
debug y => _3;
let _3: &std::string::String;
_3 = move _1;
```

src/main.rs

```
let y = x; // x is moved here
```

mid_level_intermediate_representation

```
debug y => _3;
let _3: &std::string::String;
_3 = move _1;
```

src/main.rs

```
let y = x; // x is moved here
```

mid_level_intermediate_representation

```
debug y => _3;

let _3: &std::string::String;

_3 = move _1;
```

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```
println!("{}", x); // x is NOT initialized here
```

$ cargo build

```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:7:20
5 |     let y = x;
  |             - value moved here
6 |
7 |     println!("{}", x);
  |                    ^ value borrowed here after move

error: aborting due to previous error

For more information about this error,
try `rustc --explain E0382`.
```

$ cargo build


```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:7:20
5 |     let y = x;
  |             - value moved here
6 |
7 |     println!("{}", x);
  |                    ^ value borrowed here after move

error: aborting due to previous error

For more information about this error,
try `rustc --explain E0382`.
```

$ cargo build

```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:7:20
5 |     let y = x;
  |             - value moved here
6 |
7 |     println!("{}", x);
  |                    ^ value borrowed here after move

error: aborting due to previous error

For more information about this error,
try `rustc --explain E0382`.
```

$ rustc --explain E0382

A variable was used after its contents have been moved elsewhere.

```
fn main() {
    let mut x = MyStruct{ s: 5u32 };
    let y = x;
    x.s = 6;
    println!("{}", x.s);
}
```

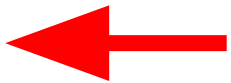
Since `MyStruct` is a type that is not marked `Copy`, the data gets moved out
of `x` when we set `y`.

$ rustc --explain E0382

Sometimes we don't need to move the value. Using a reference, we can let another function borrow the value without changing its ownership.

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    println!("{}", x);
    println!("{}", y);
}
```



$ cargo run

```
$ cargo run

Hi Rusty Days!
Hi Rusty Days!
```

Borrow Checker

Does several things including:

- Tracking initializations and moves
- **Lifetime inference**

Source: "Guide to Rustc Development"

Rust uses "lifetime" in two distinct ways

- **Lifetime of a variable**
  - Span of time before the value of the variable gets freed
  - Also known as "scope"

Source: "Rust RFC #2094: NLL"

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");


    let y = x;


    println!("{}", x);
    println!("{}", y);
}
```

x is initialized

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

x is moved

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = x;

    println!("{}", x);
    println!("{}", y);
}
```

x is not initialized here

Rust uses "lifetime" in two distinct ways

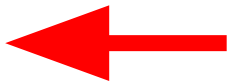
- Lifetime of a variable
  - Span of time before the value of the variable gets freed
  - Also known as "scope"

- **Lifetime of a reference**
  - **Span of time in which the reference can be used**

Source: "Rust RFC #2094: NLL"

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    println!("{}", x);
    println!("{}", y);
}
```



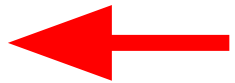
src/main.rs

```
let y = &x;
```

src/main.rs

```
let y = &x;
```

mid_level_intermediate_representation



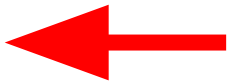
```
debug x => _1;
debug y => _3;
_3 = &_1;
```

src/main.rs

```
let y = &x;
```

mid_level_intermediate_representation

```
debug x => _1;
debug y => _3;
_3 = &_1;
```



src/main.rs

```
let y = &x;
```

mid_level_intermediate_representation

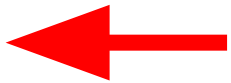
```
debug x => _1;
debug y => _3;
_3 = &_1;
```

src/main.rs

```rust
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    println!("{}", x);
    println!("{}", y);
}
```

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    drop(x);
    println!("{}", y);
}
```



$ cargo build

$ cargo build

```
error[E0505]: cannot move out of `x` because it is borrowed
 --> src/main.rs:7:10
  |
5 |     let y = &x;
  |             -- borrow of `x` occurs here
6 |
7 |     drop(x);
  |          ^ move out of `x` occurs here
8 |
9 |     println!("{}", y);
  |                    - borrow later used here
```

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    drop(x);
    println!("{}", y);
}
```

Lifetime of x

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    drop(x);
    println!("{}", y);
}
```

Needed lifetime of y

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    drop(x);
    println!("{}", y);
}
```

y can no longer reference x

src/main.rs

```
fn main() {
    let x: String;
    x = String::from("Hi Rusty Days!");

    let y = &x;

    drop(x);
    println!("{}", y);
}
```

y is dead here

Rust uses "lifetime" in two distinct ways

- Lifetime of a variable
  - Span of time before the value of the variable gets freed
  - Also known as "scope"

- Lifetime of a reference
  - Span of time in which the reference can be used

- **If you make a reference to a value, the lifetime of that reference cannot outlive the scope of the value**

Source: "Rust RFC #2094: NLL"

There is SO much more to both the compiler and the borrow checker!

Check out the "Guide to Rustc Development" for more info!

https://rustc-dev-guide.rust-lang.org/

Is the Borrow Checker a friend or a foe?

It is a friend...though a strict one

but one that will not only tell you when something is wrong...

...it will also tell you how to fix it.

Nell Shamrell-Harrington

- Sr. Staff Research Engineer at Mozilla
- Lead Editor of This Week in Rust
- Host of the This Week in Rust podcast on the Rustacean Station
- @nellshamrell