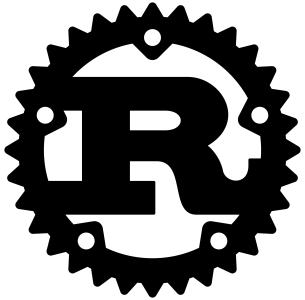
Introduction to Rust

Ian McCormack



What is a memory error?

```c
char *get_hello_world() {
 char buffer[11];
 strcpy(buffer, "hello world");
 return buffer;
}
```

C

"...an object accessed using a pointer expression is different from the one intended."
— van der Veen et al., 2012

Nearly 70% of security vulnerabilities found by Google (2015 - 2020) and Microsoft (2006 - 2018) were caused by memory errors.

What is a memory error?

```
char *get_hello_world() {
 char buffer[11];
 strcpy(buffer, "hello world");   Spatial
 return buffer;   Temporal
}
```

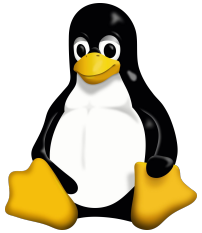
"...an object accessed using a pointer expression is different from the one intended."
— van der Veen et al., 2012

Nearly 70% of security vulnerabilities found by Google (2015 - 2020) and Microsoft (2006 - 2018) were caused by memory errors.

Rust is popular and widely used in production.

Chosen as **the "most loved" language** in StackOverflow's annual developer survey for **the last eight years.**






Garbage collection supports **dynamic memory safety**.

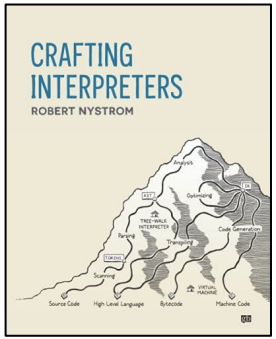
Tracing garbage collection treats memory as a reachability graph, and periodically eliminates nodes that are unreachable.

Reference counting frees memory when the count of references to an allocation in scope reaches zero.

No use-after free!

+ No pointers
+ Bounds checks on array accesses

Check out *Crafting Interpreters*!

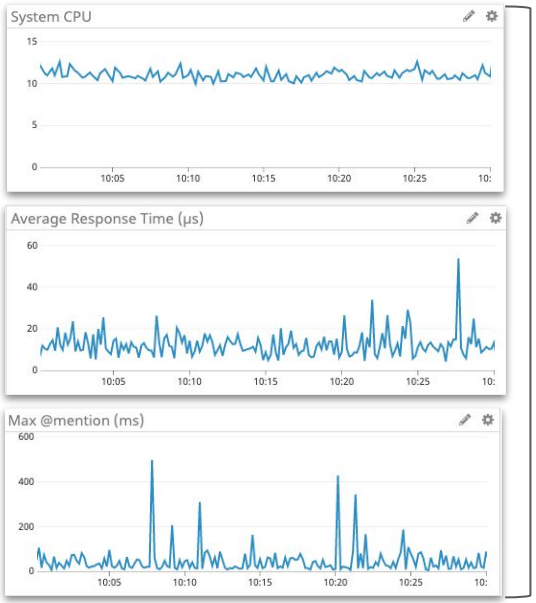


Total					
	Energy (J)		Time (ms)		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Energy Efficiency across Programming Languages, Pereira et al. 2021

Performance of Discord's "Read States" Service

February 2020



The Rust Book

- Interactive examples
- Quizzes

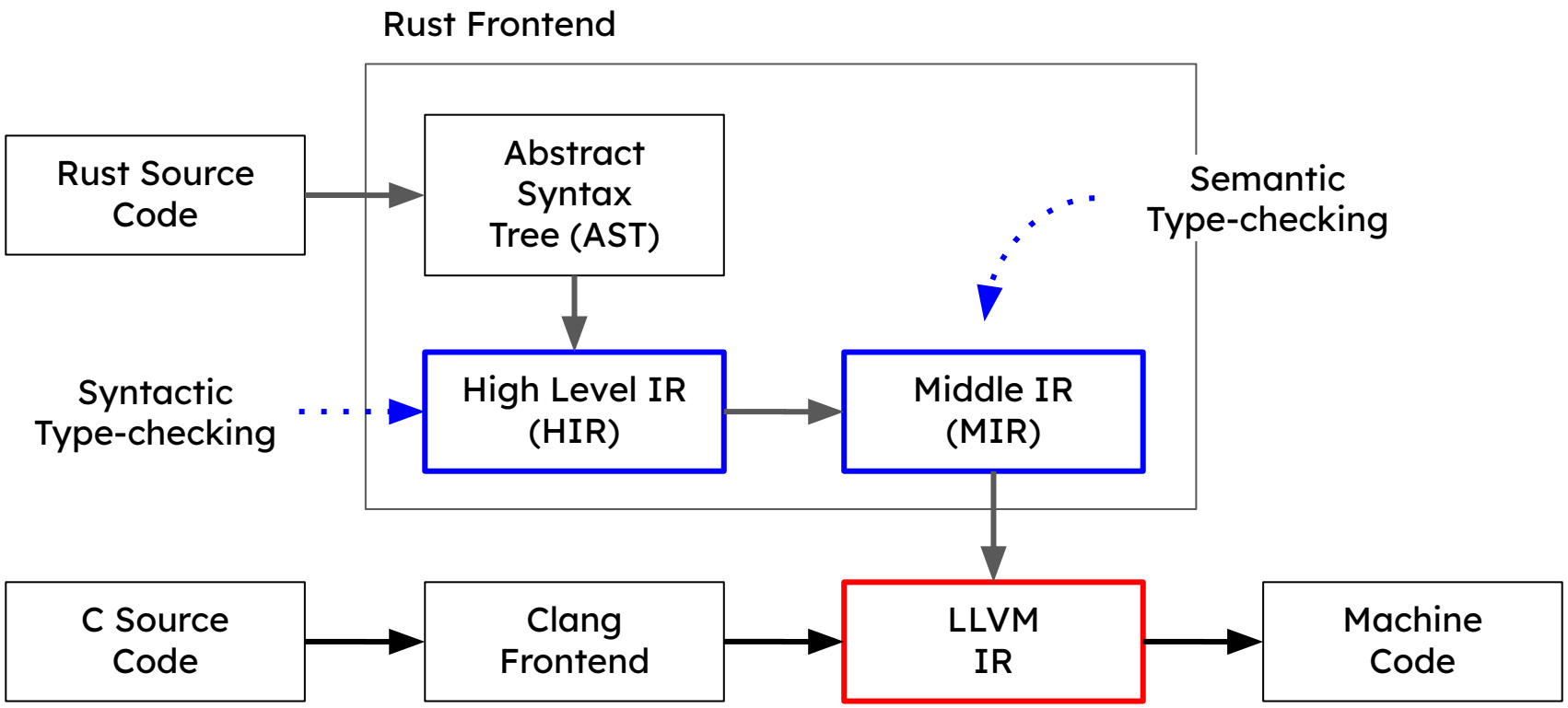
Steve Klabnik and Carol Nichols
+ Will Crichton & Shriram Krishnamurthi

Aquascope

Rust visualizations

Rust Playground

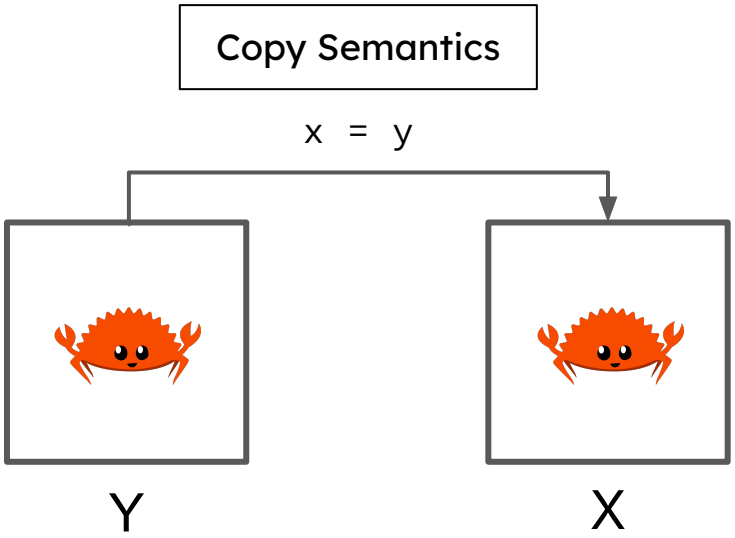
Rust in your browser

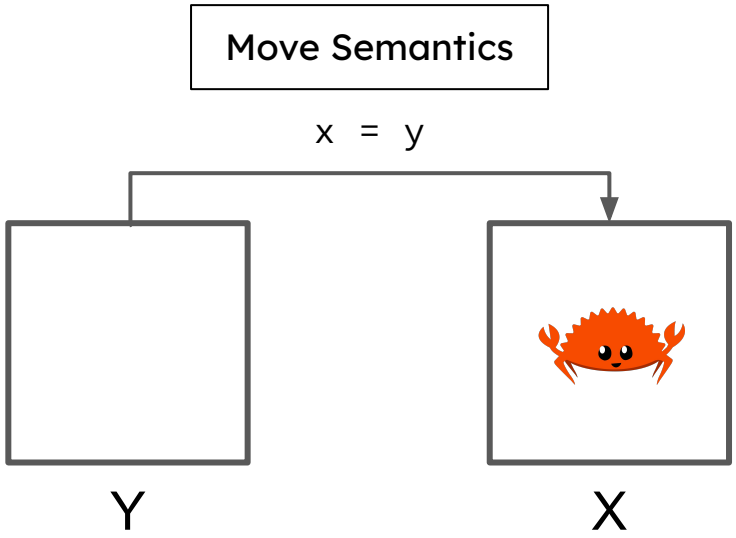

Rust Frontend
Rust Source Code
Abstract Syntax Tree (AST)
Semantic Type-checking
Syntactic Type-checking
High Level IR (HIR)
Middle IR (MIR)
C Source Code
Clang Frontend
LLVM IR
Machine Code

The Borrow Checker

1. All values have exactly one owner.

— Crichton 2021


Copy Semantics
x = y
Y
X


Move Semantics
x = y
Y
X

The Borrow Checker

1. All values have exactly one owner.

2. A value can have **one mutable** reference (`&mut T`) **or many immutable** references (`&T`), but not both.

— Crichton 2021

Copy

`&T`

Shared, read-only

Move

`&mut T`

Unique, read & write

Rust's borrow checker reasons using **lifetimes**.

The **scope of a value** is the duration for which it is **allocated.**

The **lifetime of a reference** is the *duration* for which it is **used.**

```rust
fn main() {
    let mut x = 5;
    let read_x: &i32 = &x;
    print!("{}", read_x);
    let write_x: &mut i32 = &mut x;

    *write_x = 10;

    print!("{}", write_x);

}
```

```rust
fn main() {
    let mut x = 5;

    let read_x: &i32 = &x;

    let write_x: &mut i32 = &mut x;

    *write_x = 10;

    print!("{}", write_x);

    print!("{}", read_x);
}
```

x ↑ +R +W +O

Read

Write

Own

Aquascope

```rust
fn main() {
    let mut x = 5;

    let read_x: &i32 = &x;

    let write_x: &mut i32 = &mut x;

    *write_x = 10;

    print!("{}", write_x);

    print!("{}", read_x);
}
```

x	↑	+R	+W	+O

x	→	R	~~W~~	~~O~~
read_x	↑	+R	–	+O
*read_x	↑	+R	–	–

Aquascope

Read

Write

Own

The Borrow Checker

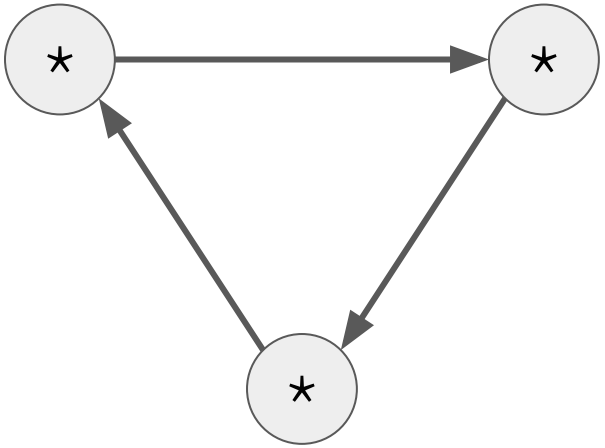
1. All values have exactly one owner.

2. A value can have **one mutable** reference (`&mut T`) **or many immutable** references (`&T`), but not both.

3. A reference to a value cannot outlive the owner.

— Crichton 2021

The Borrow Checker rejects “valid” programs.



- ❌ Doubly-linked lists
- ❌ Trees with parent and child pointers
- ❌ ***Any*** self-referential struct

The Rust Book

- Interactive examples
- Quizzes

Steve Klabnik and Carol Nichols
+ Will Crichton & Shriram Krishnamurthi

Aquascope

Rust visualizations

Rust Playground

Rust in your browser