Starting Rust
from a Scripting Background
5/10/2016

oreilly@edunham.net
edunham.net
github.com/edunham
@qedunham
talks.edunham.net/oscon-webcast2016

I...

- DevOps engineer for Mozilla Research
  - Rust
  - Servo
- ~6 years FOSS
- CS Education
- “How to learn Rust” talk at OSCON

You...

- Have coded a bit
- Interpereted language
  - Python
  - Ruby
  - JavaScript
  - Shell scripts
- Want to learn Rust!

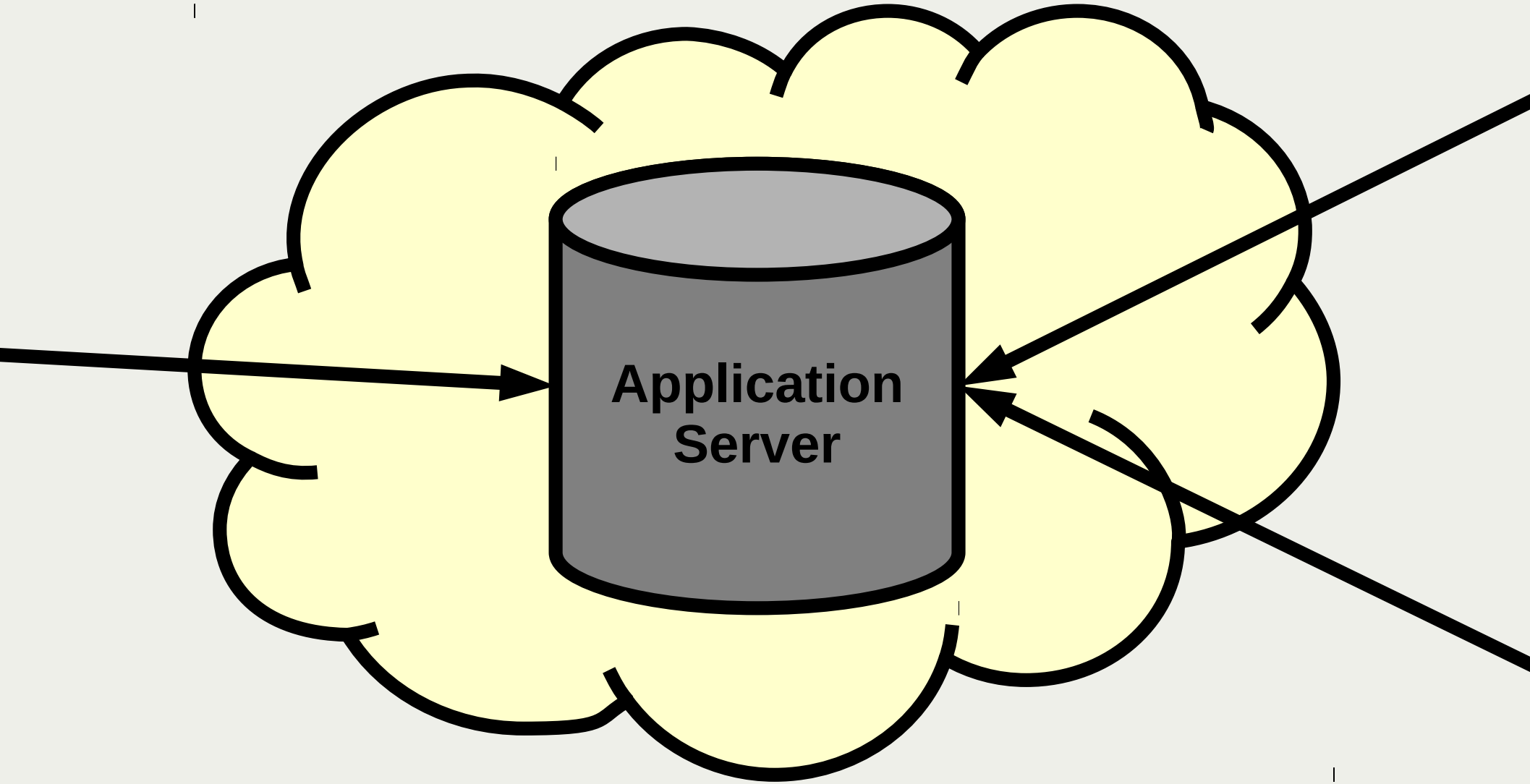
This Webcast

- Current as of May 10, 2016
- Rust 1.8.0 is the latest Stable version

- Systems Programming
- Installing & Using Rust
- The Compiler Is Your Friend

When should you use Rust?

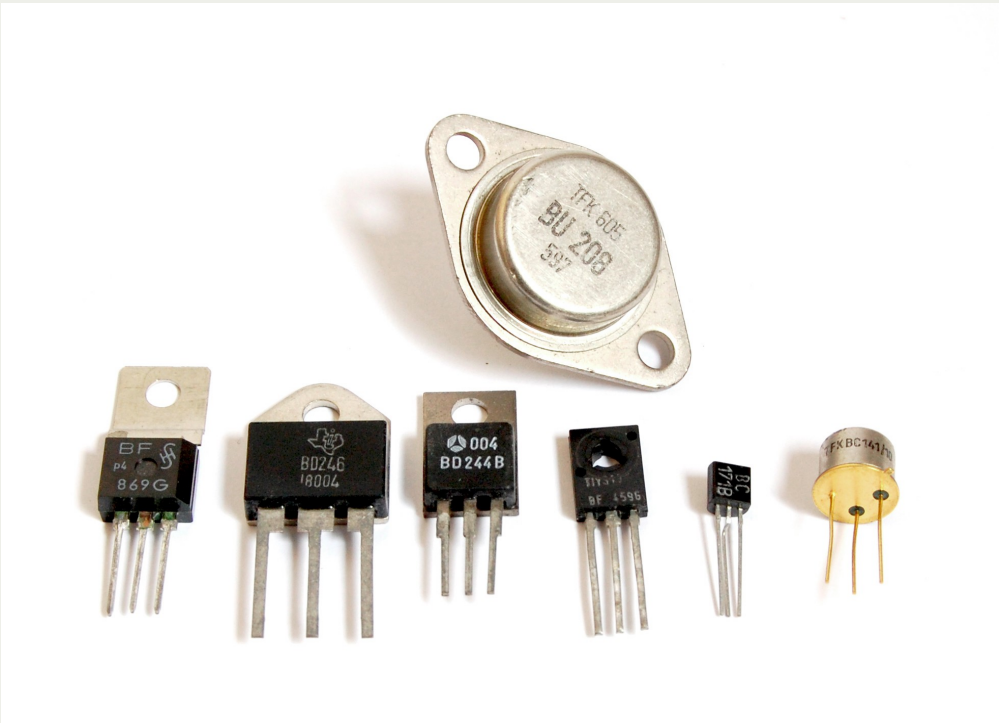
A System

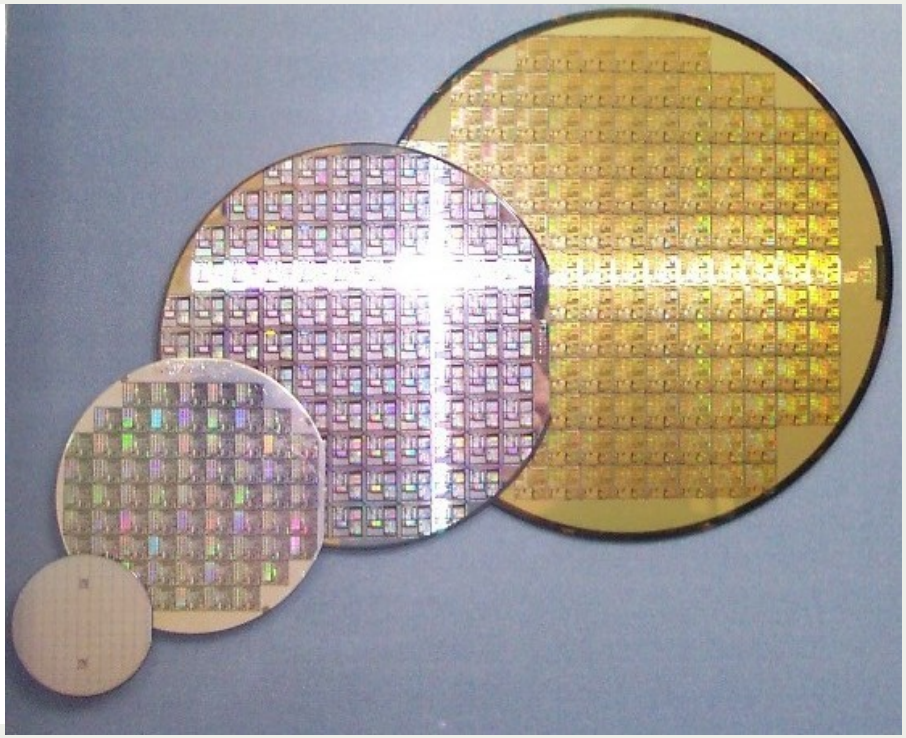


A System

https://upload.wikimedia.org/wikipedia/commons/a/ae/Rs161-e2-inside.jpg

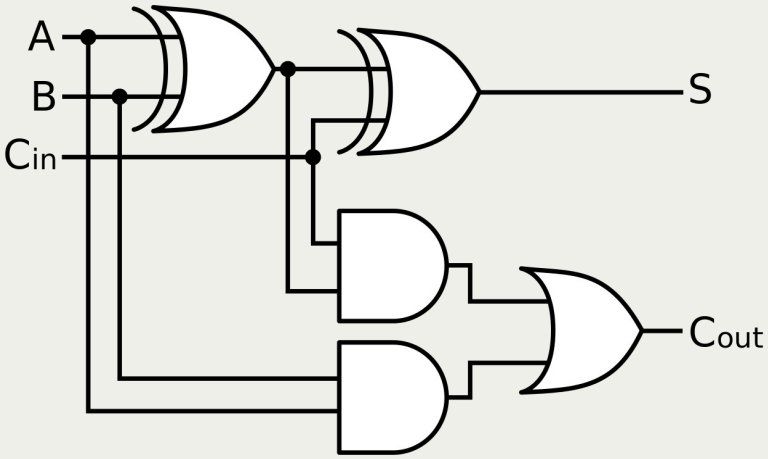
Transistors

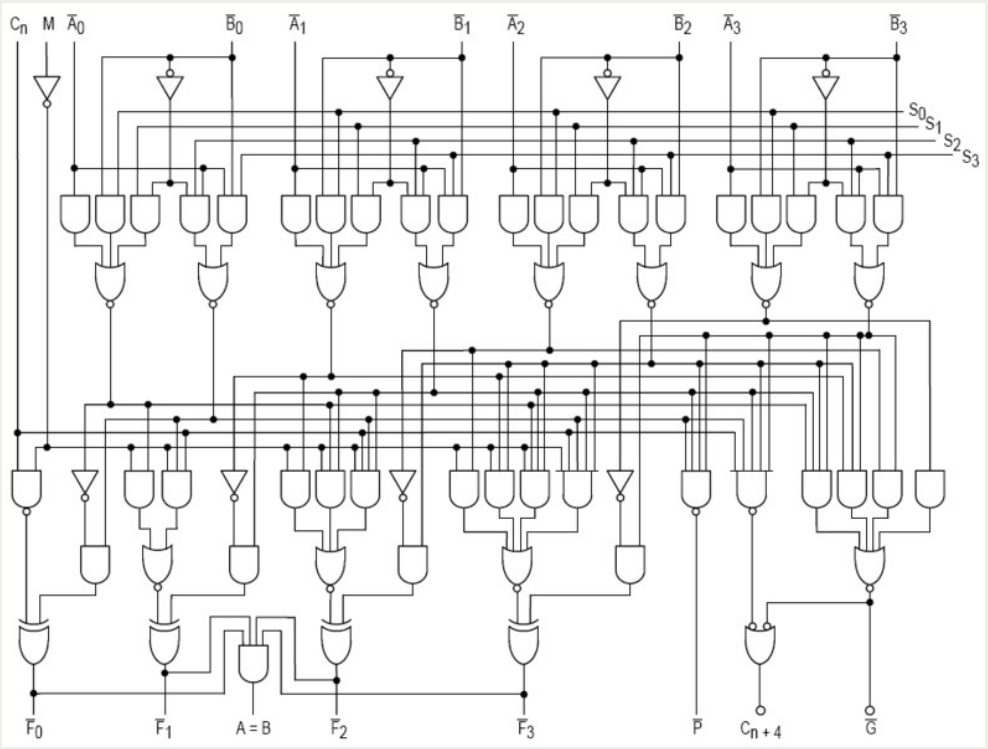




https://upload.wikimedia.org/wikipedia/commons/0/0e/Transistors-white.jpg
https://upload.wikimedia.org/wikipedia/commons/d/d7/Wafer_2_Zoll_bis_8_Zoll_2.jpg

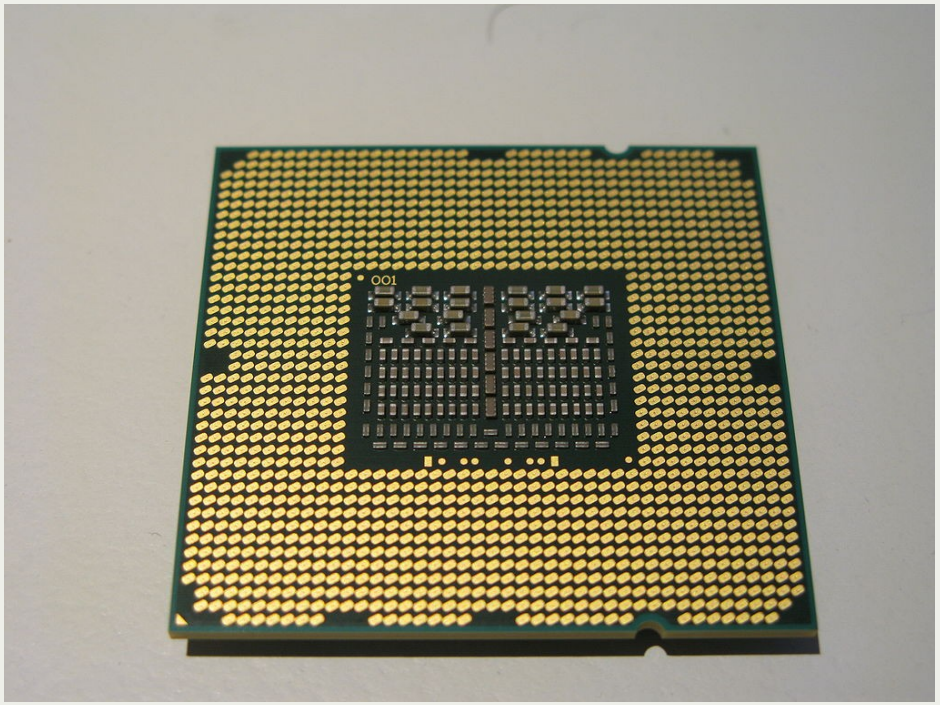
Gates & Components





https://commons.wikimedia.org/wiki/File:Full-adder.svg
https://commons.wikimedia.org/wiki/File:74181aluschematic.png

The CPU



- Get an instruction
- Read or store a number
- Add, multiply, or negate numbers
- Fast & parallel

https://en.wikipedia.org/wiki/List_of_Macintosh_models_grouped_by_CPU_type#/media/File:Intel_Core_i7-940_bottom.jpg

Assembly Language

```
main:
.Lfunc_begin1:
    .cfi_startproc
    sub rsp, 24
.Ltmp7:
    .cfi_def_cfa_offset 32
    lea rax, [rip + _ZN4main20h6da371e41eb34c3deaaE]
    mov rcx, qword ptr [rip + __rustc_debug_gdb_scripts_section__@GOTPCREL]
    mov dl, byte ptr [rcx]
    mov qword ptr [rsp + 16], rdi
    mov rdi, rax
    mov rax, qword ptr [rsp + 16]
    mov qword ptr [rsp + 8], rsi
    mov rsi, rax
    mov rcx, qword ptr [rsp + 8]
    mov byte ptr [rsp + 7], dl
    mov rdx, rcx
    call    _ZN2rt10lang_start20h0ba42f7a8c46a626rKzE@PLT
    add rsp, 24
    ret
```

For this code, run http://is.gd/d9xXAK and under settings choose Intel assembly. It's 219 lines.

Smaller, faster programs
take closer control of the hardware

and require greater
understanding of the hardware
to reason about them.

Systems Programming vs Application Programming

- Direct hardware access
- Operating Systems
- Embedded
- Language needs:
  - Minimal overhead
  - Max performance
  - Sacrifice ease of use

- Userland
  - No direct hardware access
  - Runs on OS
- Language needs:
  - Easy to write
  - Easy to read
  - Sacrifice performance

Compiled vs Interpreted

- Turn into machine language then execute
- Platform-specific
- C, C++, Go, Rust, Swift

- Runs directly on VM or emulator
- Now, compiled to IR
  - Java
  - Python
  - Ruby
- Platform-agnostic

Rust

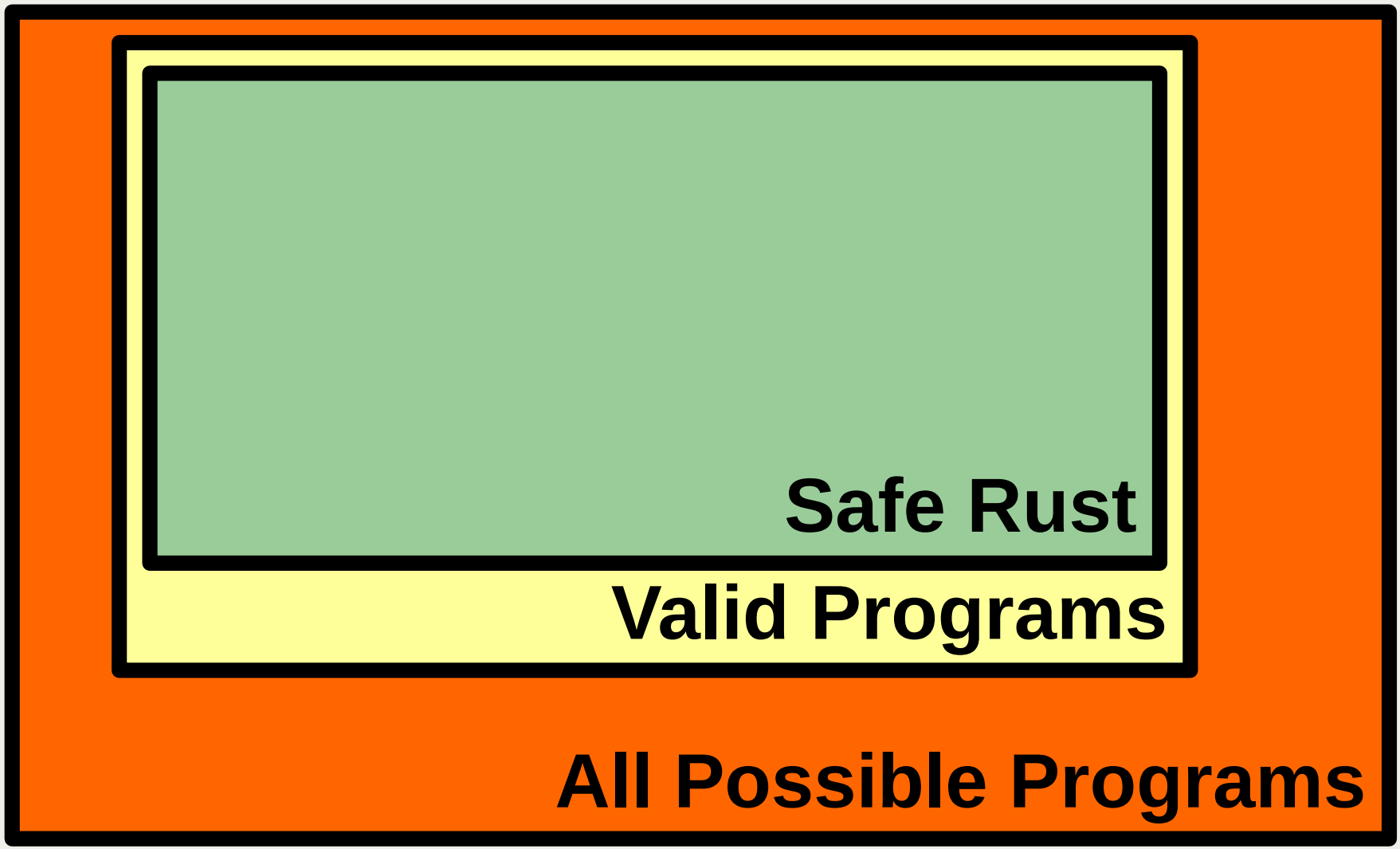
- Code which compiles has no memory errors
- Simpler concurrency
- Helpful compiler
- Younger language

vs

Other Systems Languages

- Manually prevent errors
- Concurrency as last resort
- Obscure platform support

Safe & Unsafe



Additional Resources

- Mixing Rust with scripting languages: http://callahad.github.io/pycon2015-rust/#/ & https://www.youtube.com/watch?v=3CwJ0MH-4MA
- Rust, Go, D, & C++ panel, 2014 https://www.youtube.com/watch?v=BBbv1ej0fFo
- OS written in Rust: http://www.redox-os.org/
- Browser engine in Rust: https://servo.org/

The Rust Ecosystem

Code of Conduct

- Friendly, safe, & welcoming environment for all
- Please be kind and courteous
- Please keep unstructured critique to a minimum.

www.rust-lang.org/conduct.html

"The Rust community seems to be populated entirely by human beings. I have no idea how this was done."

- http://scattered-thoughts.net/blog/2015/06/04/three-months-of-rust/

Channels

- Stable
- Beta
- Nightly

"The stable release channel will provide pain-free upgrades, and the nightly channel will give early adopters access to unfinished features as we work on them."

– http://blog.rust-lang.org/2014/10/30/Stability.html

Libraries

- Cargo
  - Pip
  - Gem
  - NPM
  - CPAN
- Crates.io
- Libraries are called crates
- http://doc.rust-lang.org/stable/book/crates-and-modules.html

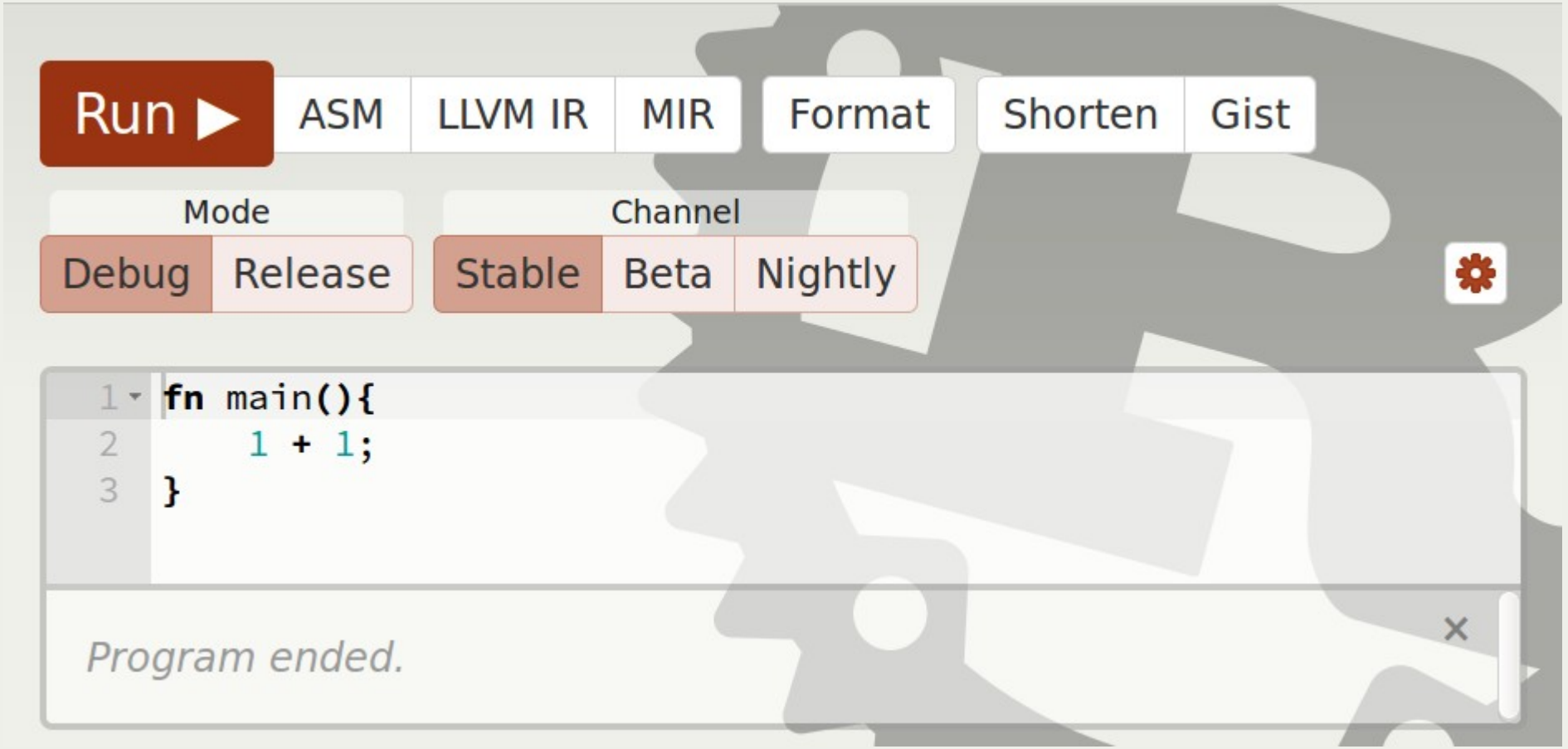
Find Rustaceans...

- users.rust-lang.org
- reddit.com/r/rust
- twitter.com/rustlang
- irc.mozilla.org (#rust, #rust-beginners)
- github.com/rust-lang
- stackoverflow.com/questions/tagged/rust
- This Week in Rust newsletter

Getting Rust

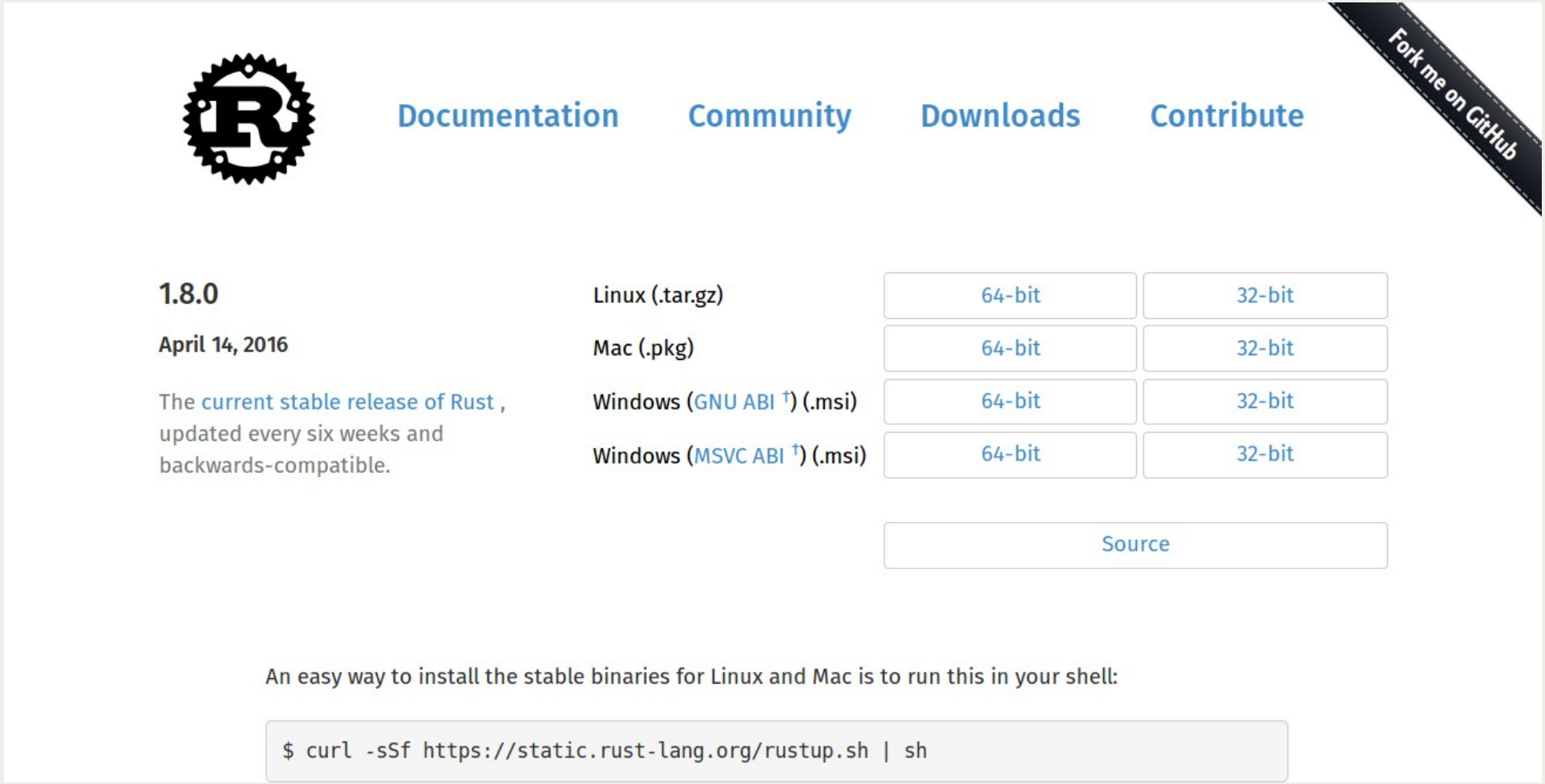
Try it Out

- play.rust-lang.org



One Version

- rust-lang.org/downloads.html



Need Several Versions?

- github.com/brson/multirust

```
Welcome to Rust.

This script will download, build, and install multirust as root, then
configure multirust with the most common options. It may prompt for
your password for installation via 'sudo'.

You may run /usr/local/lib/rustlib/uninstall.sh to uninstall multirust.

Ready? (y/N) y

blastoff: working in temporary directory /tmp/tmp.tIYuwdLG5Q
blastoff: cloning multirust git repo
Cloning into 'multirust'...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (20/20), done.
```

Your First Project

- `cargo new myproject`
  - OR
- `multirust run stable cargo new myproject`
  - THEN
- `vim myproject/src/lib.rs`

IDE Support

https://areweideyet.com/

	Syntax highlighting (.rs)	Snippets	Code Completion	Linting	Go-to Definition	Syntax Checking
Atom	✓[1]	✓[1]	✓[2]	✓[1]	✓[2]	
BBedit	✓[1]	✓[1]				
Emacs	✓[1]	✓[1]	✓[2]	✓[1]	✓[2]	✓[1]
Geany	✓					
Kate	✓		✓[2]			
Textadept	✓	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
Sublime	✓[1]	✓[1]	✓[2]	✓[1]	✓[2]	
Vim	✓[1]	✓[1]	✓[2]		✓[2]	✓[1]
VS Code	✓	✓	✓[2]		✓[2]	
Eclipse	✓[1]	✓[1]	✓[2]		✓[1]	
Visual Studio	✓		✓[2]		✓[1]	
SolidOak	✓		✓[2]			
IntelliJ IDEA	✓[1]					✓[1]

✓ = supported out-of-the-box, ✓[1] = supported via plugin, ✓[2] = supported via racer and plugin

Resources for Writing Rust

- http://rustbyexample.com/
- http://doc.rust-lang.org/stable/book/
- https://github.com/carols10cents/rustlings
- https://github.com/ctjhoa/rust-learning

Hello, World

Basic Syntax

```
1  // Main takes no arguments and returns nothing
2  fn main(){
3      // The function body is the *scope* inside these curly brace
4      // Create a variable. It owns a string.
5      let what_to_say = "Hello World";
6      // Meet print syntax
7      println!("This program says {}", what_to_say);
8  }
```

```
This program says Hello World

Program ended.
```

Scope Errors!

```
1  fn not_main(){
2      let what_to_say = "Hello World";
3  }
4  fn main(){
5      println!("This program says {}", what_to_say);
6  }
```

```
<anon>:5:38: 5:49 error: unresolved name `what_to_say` [E0425]
<anon>:5      println!("This program says {}", what_to_say);
                                              ^~~~~~~~~~~
```

Punctuation Errors

```
1  fn main(){
2      let what_to_say = "Hello World"
3      println!("This program says {}", what_to_say);
4  }
```

```
<anon>:3:5: 3:12 error: expected one of `.`, `;`, or an
operator, found `println`
<anon>:3      println!("This program says {}", what_to_say);
```

Unused Variables

```
1 fn main(){
2     let what_to_say = "Hello World";
3     println!("Hello");
4 }
```

```
<anon>:2:9: 2:20 warning: unused variable: `what_to_say`,
#[warn(unused_variables)] on by default
<anon>:2     let what_to_say = "Hello World";
```

Hey, Pythonistas!

```
1 fn main(){let what_to_say="Hello World";println!
2 ("This program says {}",what_to_say);}
```

```
This program says Hello World
```

Hey, Pythonistas!

Review

- Run code on play.rust-lang.org
- Variables live in their {scopes}
- Compiler enforces the rules

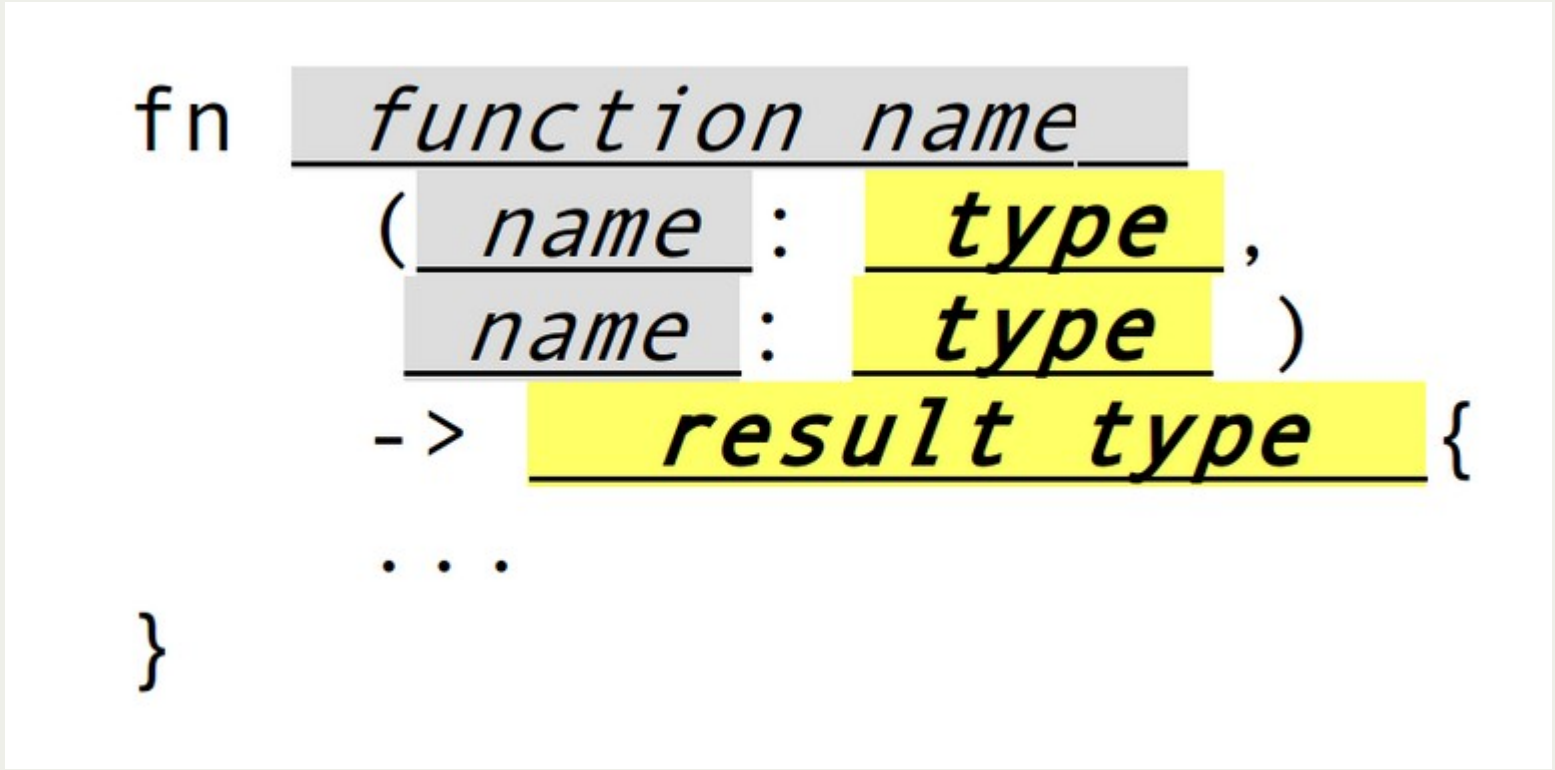
Types

Primitive types (built-in)

- `bool`
- `char`
- `i8, i16, i32, i64`
- `u8, u16, u32, u64`
- `f32, f64`
- `isize, usize`
- `str`
- `tuple`

https://doc.rust-lang.org/book/primitive-types.html

Function Type Signatures



Function with a type signature

```
1  fn sum(a: i32, b: i32, c: i32) -> i32{
2      a + b + c
3  }
4
5  fn main(){
6      println!("A sum is {}", sum(1, 2, 3))
7  }
```

```
A sum is 6

Program ended.
```

Synonymous return

```
1  fn sum(a: i32, b: i32, c: i32) -> i32{
2      return a + b + c;
3  }
4
5  fn main(){
6      println!("A sum is {}", sum(1, 2, 3))
7  }
```

```
A sum is 6

Program ended.
```

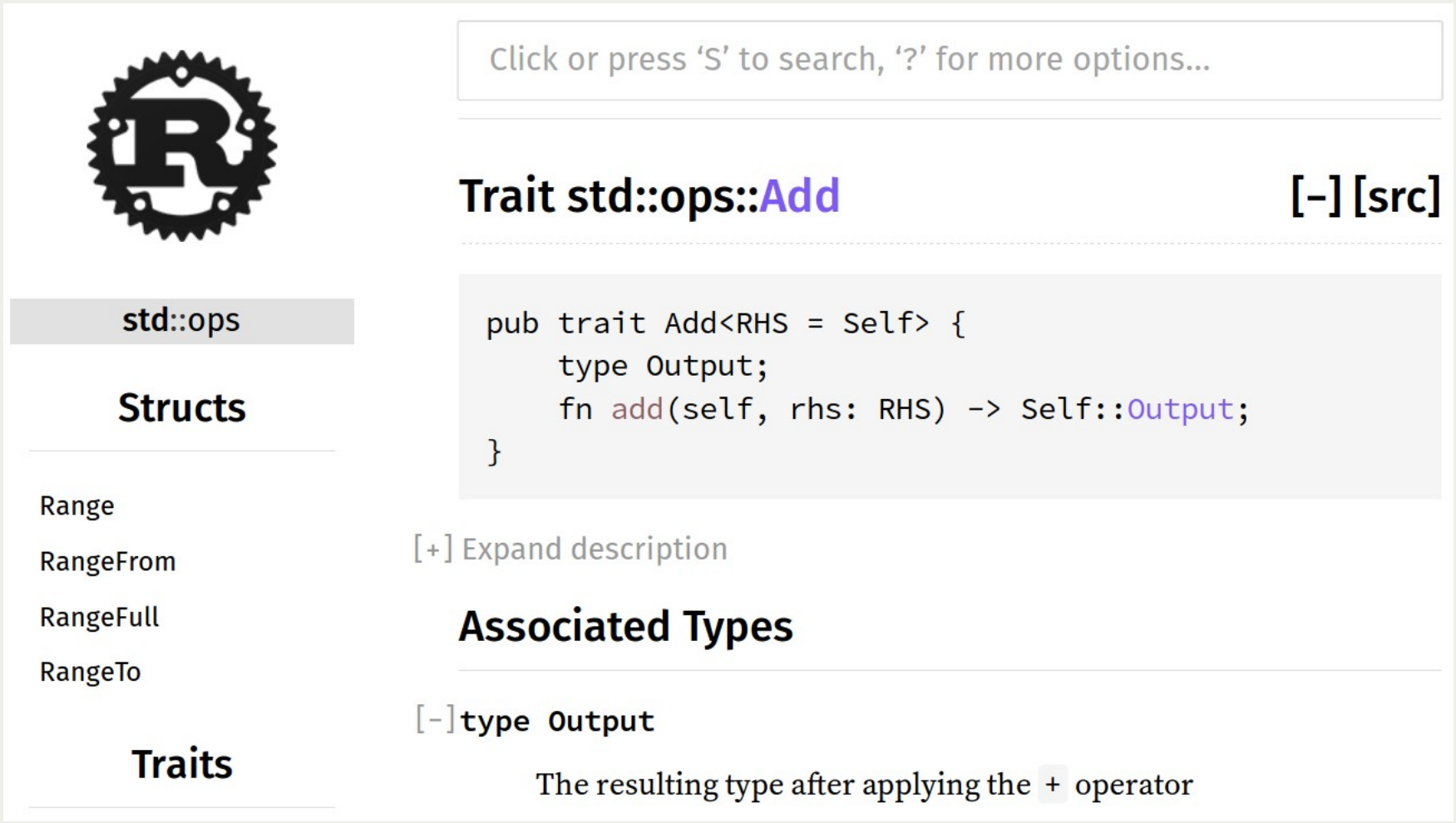
Type Errors

```
1  fn sum(a: i32, b: i32, c: i32) -> i32{
2      return a + b + c;
3  }
4
5  fn main(){
6      println!("A sum is {}", sum(1.1, 2.3, 3.2))
7  }
```

```
<anon>:6:33: 6:36 error: mismatched types:
 expected `i32`,
    found `_`
(expected i32,
    found floating-point variable) [E0308]
```

Anything you can add...

https://doc.rust-lang.org/std/ops/trait.Add.html



Traits

```
1  use std::ops::Add;
2
3  fn sum<T: Add<Output=T>>(a: T, b: T, c: T) -> T{
4      return a + b + c;
5  }
6
7  fn main(){
8      println!("A sum is {}", sum(1.1, 2.3, 3.2))
9  }
```

```
A sum is 6.6

Program ended.
```

Review

- Functions have type signatures
- You must provide the promised inputs & outputs
- Types have traits
- You can create custom types & traits

Additional Resources

- https://doc.rust-lang.org/book/traits.html
- http://blog.rust-lang.org/2015/05/11/traits.html
- http://rustbyexample.com/trait.html
- http://pcwalton.github.io/blog/2012/08/08/a-gentle-introduction-to-traits-in-rust/

Ownership

The Rules

- No borrow may outlive value's owner
- Exactly 1 mutable reference (&mut T)
  - OR
- As many read-only references (&T) as you want

Giving Away a Value

```
1  fn main() {
2      let mystring = "I'm immutable!".to_string();
3      println!("{}", mystring);
4      let mut yarn = mystring; //yarn now owns the value
5      println!("{}", yarn);
6      yarn = "I have been mutated".to_string();
7      println!("{}", yarn);
8  }
9
```

```
I'm immutable!
I'm immutable!
I have been mutated

Program ended.
```

Given away means gone

```
1  fn main() {
2      let mystring = "I'm immutable!".to_string();
3      println!("{}", mystring);
4      let mut yarn = mystring; //yarn now owns the value
5      println!("{}", mystring);
6  }
7
```

```
<anon>:5:20: 5:28 error: use of moved value: `mystring` [E0382]
<anon>:5      println!("{}", mystring);
                             ^~~~~~~~
<std macros>:2:25: 2:56 note: in this expansion of format_args!
<std macros>:3:1: 3:54 note: in this expansion of print!
```

Borrow the value

```
1  fn main() {
2      let mystring = "I'm immutable!".to_string();
3      println!("{}", mystring);
4      let yarn = &mystring;
5      println!("{}", mystring);
6      println!("{}", yarn);
7  }
8
```

```
I'm immutable!
I'm immutable!
I'm immutable!

Program ended.
```

Make a mutable copy

```
1  fn main() {
2      let mystring = "I'm immutable!".to_string();
3      println!("{}", mystring);
4      let mut yarn = mystring.clone();
5      println!("{}", mystring);
6      println!("{}", yarn);
7      yarn = "I have been mutated".to_string();
8      println!("{}", yarn);
9  }
```

```
I'm immutable!
I'm immutable!
I'm immutable!
I have been mutated
```

Review

- Only owner can access value
- 1 mutable reference or unlimited read-only references to each value
- Borrow may not outlive owner

Additional Resources

- http://doc.rust-lang.org/stable/book/ownership.html
- http://doc.rust-lang.org/stable/book/references-and-borrowing.html
- Why the to_string()? http://hermanradtke.com/2015/05/03/string-vs-str-in-rust-functions.html
- http://rustbyexample.com/trait/clone.html
- https://www.reddit.com/r/rust/comments/2xxjda/when_should_my_type_be_copy/

Questions?

Starting Rust
from a Scripting Background
5/10/2016

oreilly@edunham.net
edunham.net
github.com/edunham
@qedunham
talks.edunham.net/oscon-webcast2016