
MONTANA
STATE UNIVERSITY

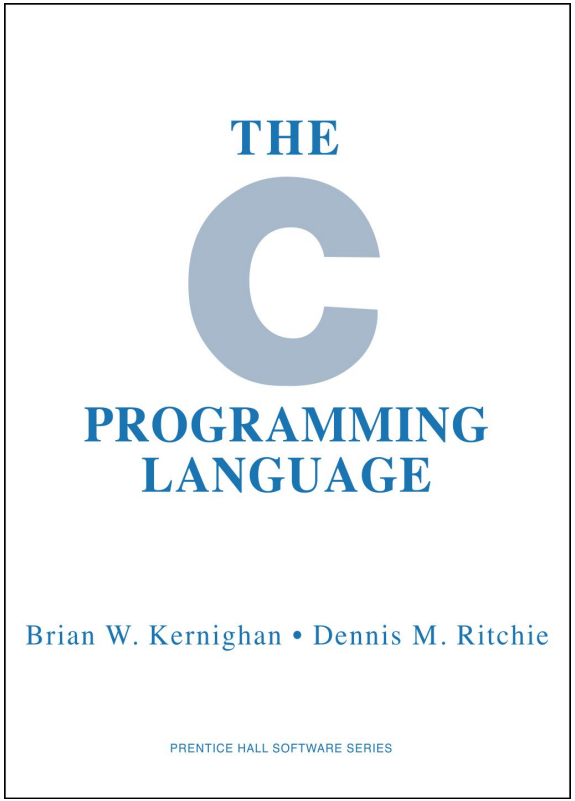
C++, Rust, etc.

...

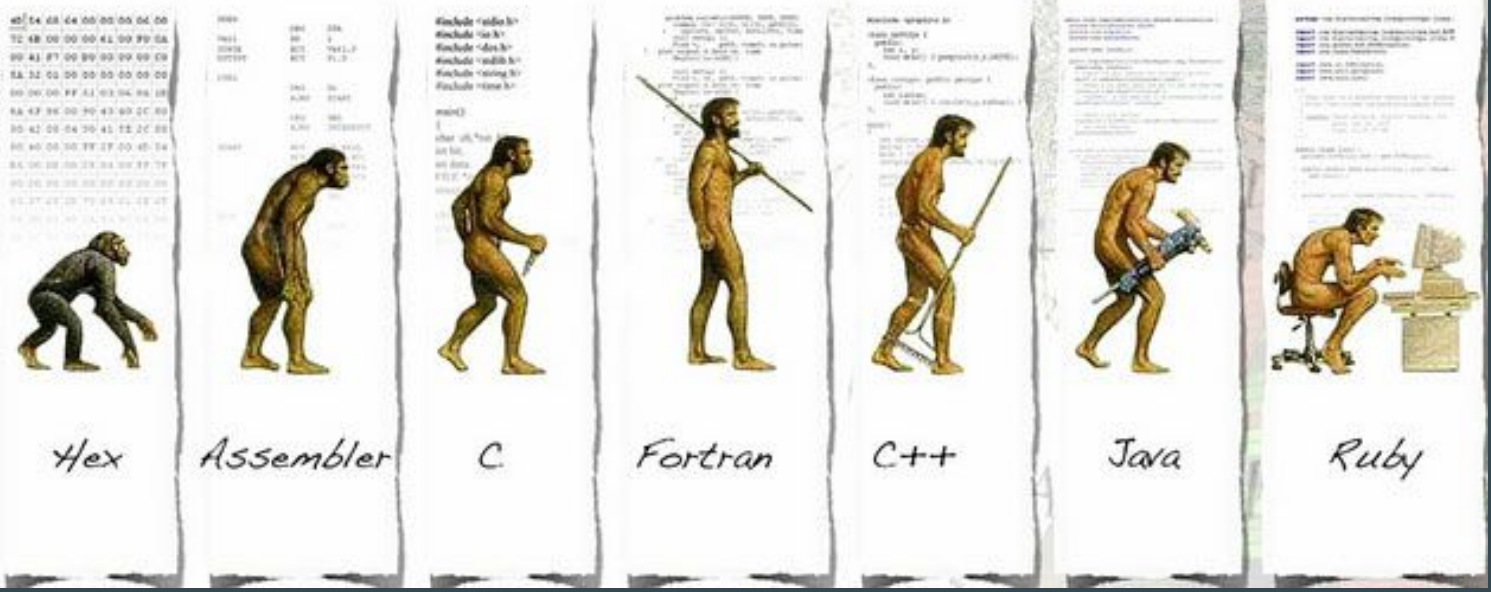
Is there any hope?

C... Really?

- Why are we still using C for systems software development in the year 2020?
- What other options are there?
- Why didn't they take over?
- We will look at 3 alternatives
  - C++
  - Rust
  - zig



The Evolution Of Computer Programming Languages



C++

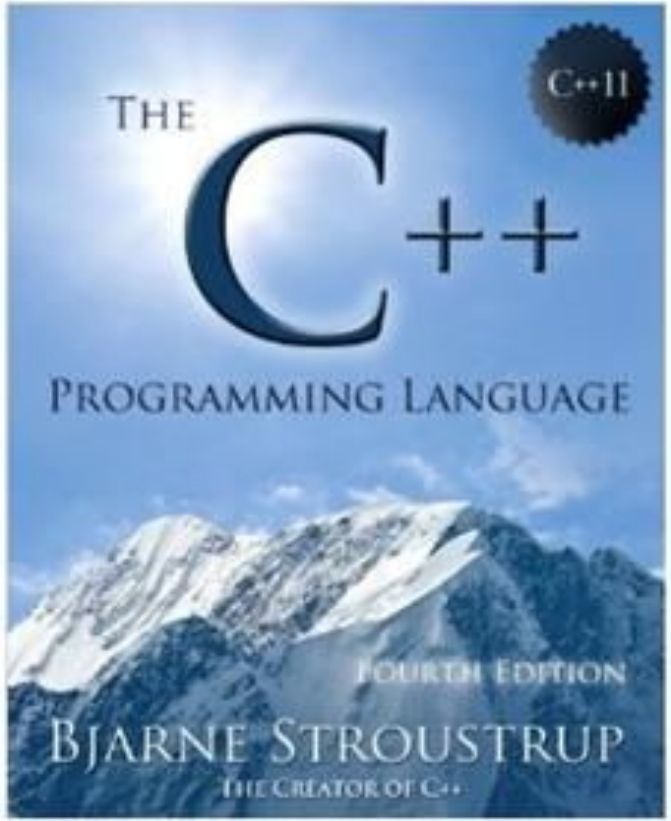
- Invented by Bjarne Stroustrup
- Began as “C with Classes” in 1979
- Generally available in 1985
- Heavily influenced by C

C++

- Features of C++
  - A full static type system with user defined classes
  - Value and reference semantics
  - A resource management mechanism (RAII)
  - Generic programming
  - Template programming
  - Operator overloading

C++

- Akin to the K&R book, there is Stroustrup's book on C++
- Stroustrup is now a managing director at Morgan Stanley in New York
- Smart guy...



C++ - Classes

- User defined classes
- Syntax should look vaguely familiar if you know Java
- Access control is specified with modifiers, as in Java
- Where are the implementations?

```
class CPPDemo {
public:
    void doIt();
    unsigned long long int addrOfCalled();
private:
    int _called = 0;
};
```

C++ - Classes

- Implementations are typically, although not required to be, defined elsewhere
- The .h file contains public definitions
- The .cpp file contains implementation details
- No File←→ Class correspondence, as in Java

```
void CPPDemo::doIt() {
    cout << "Did it... " << _called++ << "\n";
}

unsigned long long CPPDemo::addrOfCalled() {
    return reinterpret_cast<unsigned long long>(&_called);
}
```

C++ - Classes

- Classes can be created on the Stack like a struct
- Or can be created on the heap, using the *new* Keyword
- Address Demo

```
CPPDemo demo = CPPDemo();

CPPDemo * demo2 = new CPPDemo();
```

C++ - Constructors

- Classes can have constructors and destructors for setting up and cleaning up class data
- As with methods, the implementation is usually elsewhere

```
class ConstructorDemo {
public:
    ConstructorDemo(int);
    ~ConstructorDemo();
private:
    int i = 0;
    char * buffer;
};
```

C++ - Constructors

- Note that we allocate a dynamic char buffer in the constructor and clean it up in the destructor
- *Constructor Demo*

```
ConstructorDemo::ConstructorDemo(int pi) {
    this->i = pi;
    cout << "Constructing a ConstructorDemo " << i << "\n";
    buffer = static_cast<char *>(malloc(i));
}

ConstructorDemo::~ConstructorDemo() {
    cout << "Destructing a ConstructorDemo " << i << "\n";
    free(buffer);
}
```

C++ - RIAA

- As you can see, Stack-based classes automatically invoke their destructor
- This is tied to the idea: Resource Acquisition Is Initialization (RIAA)

```
ConstructorDemo::ConstructorDemo(int pi) {
    this->i = pi;
    cout << "Constructing a ConstructorDemo " << i << "\n";
    buffer = static_cast<char *>(malloc(i));
}

ConstructorDemo::~ConstructorDemo() {
    cout << "Destructing a ConstructorDemo " << i << "\n";
    free(buffer);
}
```

C++ - RIAA

- C++ runtime *guarantees* that the destructor will run, even in the presence of exceptions
- For stack allocated classes the destructor will be run when the function exits, no matter what

```
ConstructorDemo::ConstructorDemo(int pi) {
    this->i = pi;
    cout << "Constructing a ConstructorDemo " << i << "\n";
    buffer = static_cast<char *>(malloc(i));
}

ConstructorDemo::~ConstructorDemo() {
    cout << "Destructing a ConstructorDemo " << i << "\n";
    free(buffer);
}
```

C++ - RIAA

- This can be used to for *Scope-based Resource Management*
  - e.g. An output stream allocated on the stack that opens a file for writing will guarantee that it closes the file when the stack frame exits

```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

C++ - RIAA

- Powerful idea
- To an extent, addresses the lack of garbage collection in C++
- Rust takes this idea even further...

```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

C++ - Other Stuff

- C++ has pretty extensive OO capabilities:
  - Interfaces
  - Polymorphism
- Namespaces for organizing code
  - C has a flat namespace, hence the crazy method names
- Templates
  - Allow for generic programming

```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

C++ - Other Stuff

- C++ has a good standard library
  - Vectors and other data structures
  - Pretty good I/O
  - A string class (!!!)
- OK, so, why didn't we use it?

```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

C++ - Other Stuff

- C++ is a very large and complex language compared with C
- Features like operator overloading and templates can make C++ code very difficult to read
- The reality is that most systems code written today is written in C

```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

C++ - Other Stuff

- That said, there would be some stuff that would be *a lot* easier in C++
- Testing!
- What do you guys think?

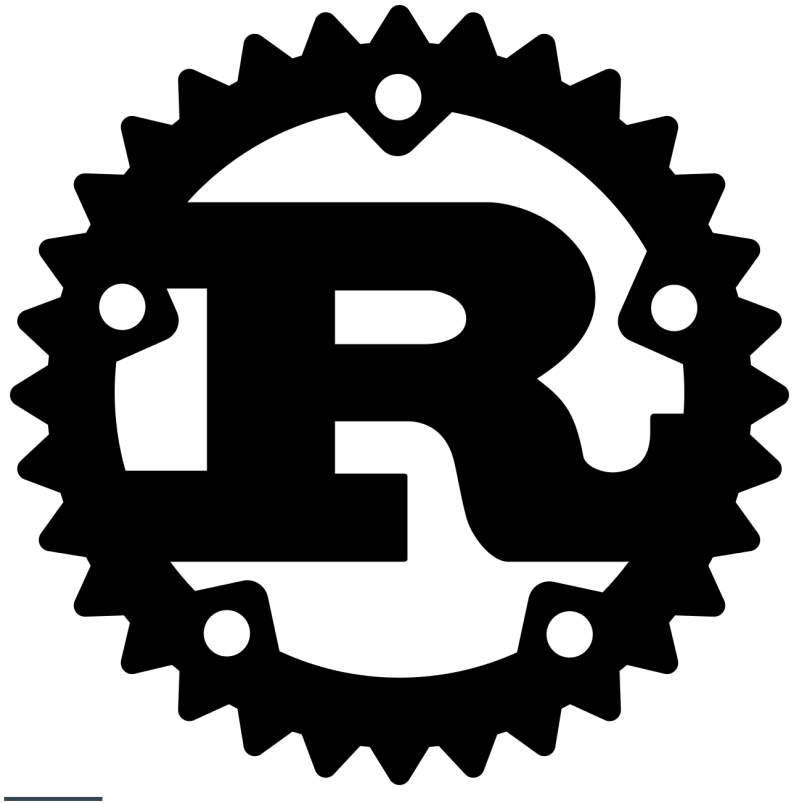
```
// Try to open file.
std::ofstream file( s: "example.txt");

if (!file.is_open()) {
    throw std::runtime_error("unable to open file");
}

// Write |message| to |file|.
file << "C++ isn't that bad..." << "\n";
```

Rust

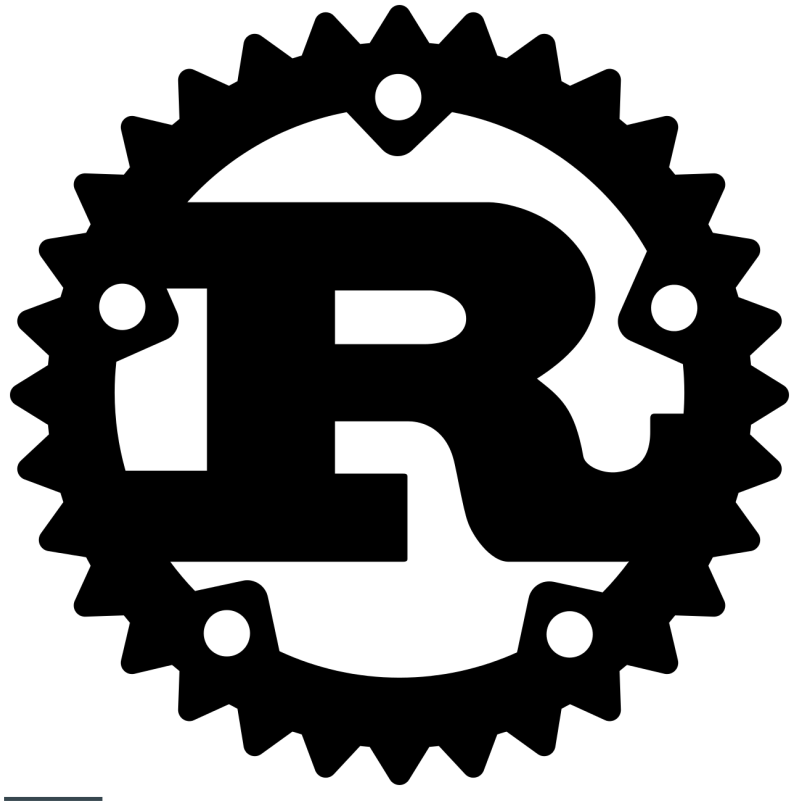
- Recently the Rust language has been making some headway in systems areas
- Some hints that it might be a supported language for parts of linux kernel development



Rust

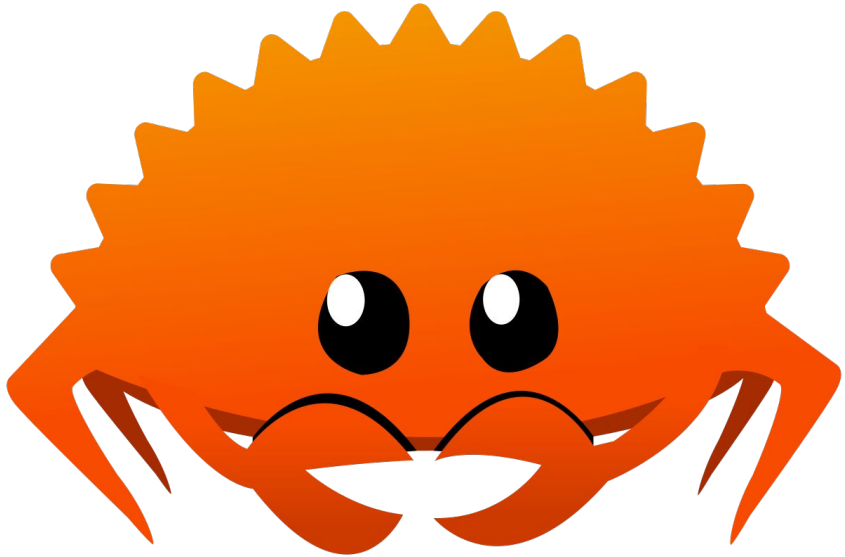
“Rust Is the Industry’s ‘Best Chance’ at Safe Systems Programming”

-- Ryan Levick (Microsoft)



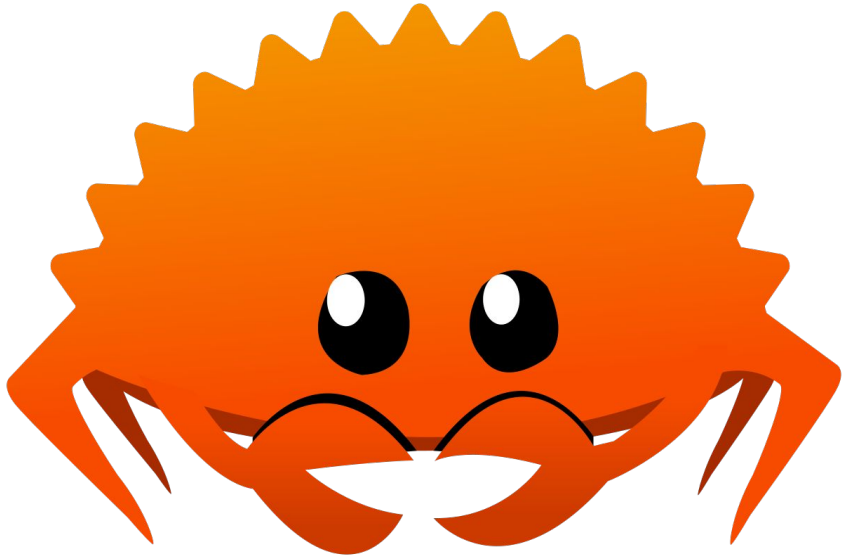
Rust

- History
  - Developed out a personal project begun at Mozilla in 2006
  - 1.0 was released in 2016
  - 2016-2020 named the Most Loved Programming Language on Stack Overflow
  - Used in Firefox development
  - Other projects are adopting it
    - OpenDNS
    - Discord



Rust Features

- Type inference
- Macros
  - Well thought out compared with C and C++
- C like control flow



Rust

- Hello Rust
- functions are declared with the *fn* keyword
- Functions declare parameter types *name : type*
- Note that there are distinct types for a string literal (&str or a *reference* to a str) and a String

```
fn add_rust(arg : &str) -> String {
    let mut str :String  = "Rust ".to_owned();
    str.push_str( string: arg);
    str
}

fn main() {
    let x :&str  = "Rocks!";
    println!("Hello, world! {}",
             add_rust( arg: x));
}
```

Rust

- *let* introduces a new variable
- Must include the *mut* modifier if you intend to *mutate* the object
- No explicit return statement
  - Although it is allowed
- That funny *.to_owned()* function call converts a &str to an *owned* String

```
fn add_rust(arg : &str) -> String {
    let mut str :String  = "Rust ".to_owned();
    str.push_str( string: arg);
    str
}

fn main() {
    let x :&str  = "Rocks!";
    println!("Hello, world! {}",
             add_rust( arg: x));
}
```

Rust

- Borrowing & Owning
  - Rust has a notion of *ownership*
  - A scope owns a piece of data
  - When that scope exits, the data is reclaimed
    - **One and only one** symbol owns the data
  - & *borrows* a reference value
  - Data that is returned from a function is owned by the calling scope

```
fn add_rust(arg : &str) -> String {
    let mut str :String  = "Rust ".to_owned();
    str.push_str( string: arg);
    str
}

fn main() {
    let x :&str  = "Rocks!";
    println!("Hello, world! {}",
             add_rust( arg: x));
}
```

Rust

- Object Oriented Features
  - Rust does not have classes
  - Instead it has *Structs* and *Traits*
  - A Struct is a collection of data
  - A Trait is similar to an Interface
  - An implementation of a given trait is provided by an *impl* declaration
- This is very different than java or python

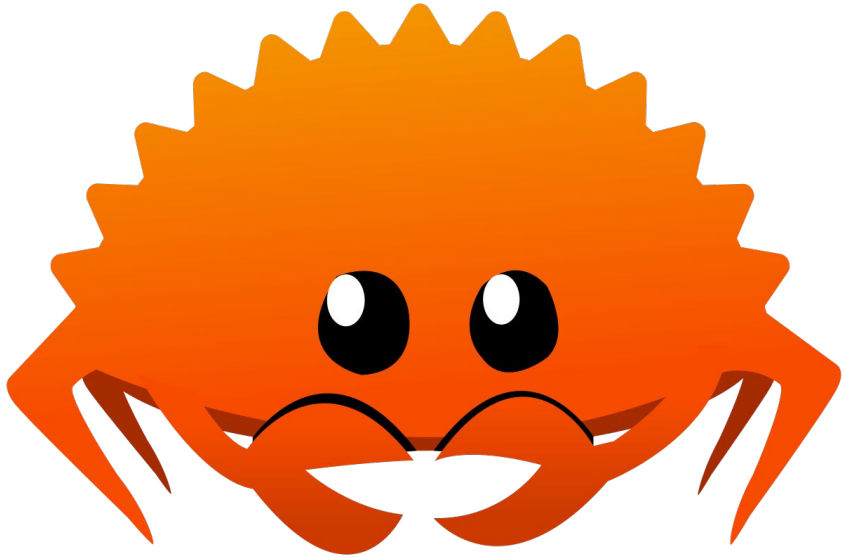
```rust
pub trait Summary {
    fn summarize(&self) -> String;
}

pub struct NewsArticle {
    pub headline: String,
    pub location: String,
    pub author: String,
    pub content: String,
}

impl Summary for NewsArticle {
    fn summarize(&self) -> String {
        format!("{}, by {} ({})", self.headline, self.author, self.location)
    }
}
```

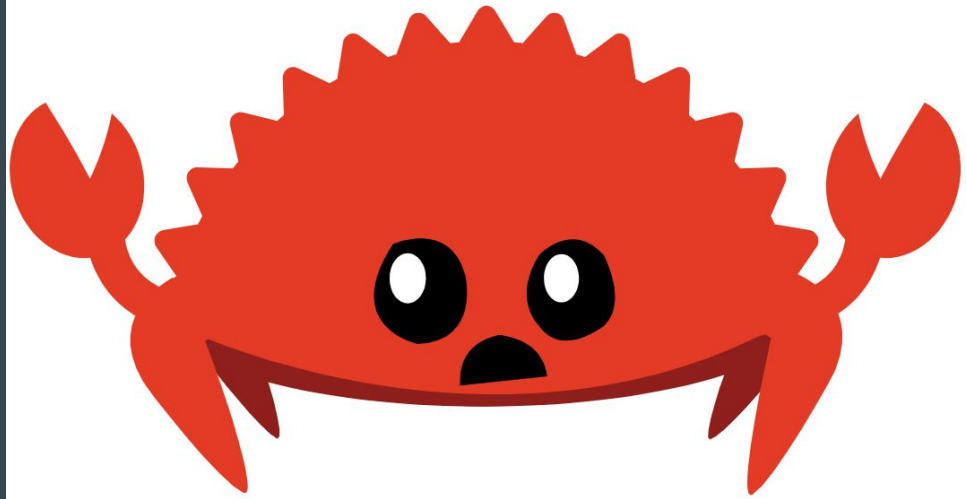
Rust - The Great Hope?

- Maybe
- Certainly has some very nice language features
- Much easier to reason about in most cases
- However
  - The ownership/borrow concept is difficult at first
  - Sometimes you have to explicitly state lifetimes for compiler



Rust - The Great Hope?

- I think Rust has a shot
- But if I were to place a bet, I'd bet that the linux kernel is still being written in C in 20 years...



Other Languages

- Zig
  - Nice design, small language compared with Rust
- D
  - A reengineering of C++
  - Has garbage collection
- Go
  - ...

Other Languages

- Go
  - Designed by Ken Thompson & Rob Pike at google
  - Has garbage collection
  - Compiles to a binary executable
  - Very fast compilation times
  - Excellent standard library
  - Especially good for networking

Other Languages

- All of them are good languages to play around with
- As of right now, I don’t anticipate any of them displacing C
  - Go & Rust may displace C for some networking-related software

Summary

- We looked in depth at two potential alternatives / successors to C
- C++ was the presumed successor to C, but has not replaced it in most systems-level programming
- Rust is the current potential challenger, but is a complex language compared with C
- There are other smaller possibilities
- As of right now, it looks like C is what we've got folks...


MONTANA
STATE UNIVERSITY