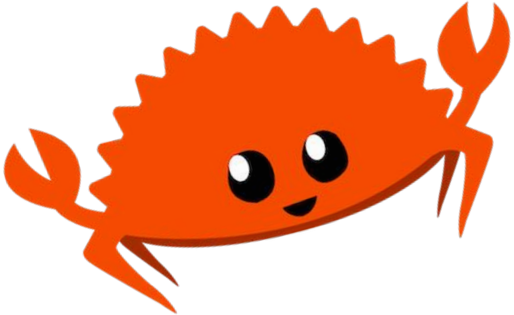
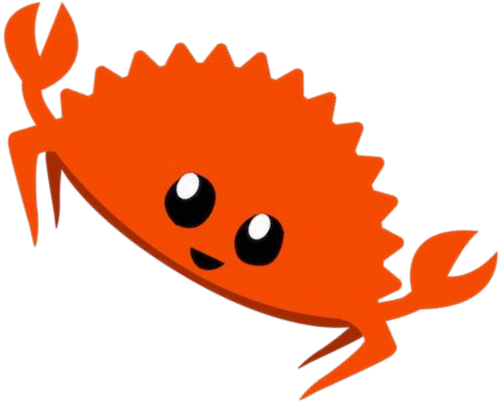
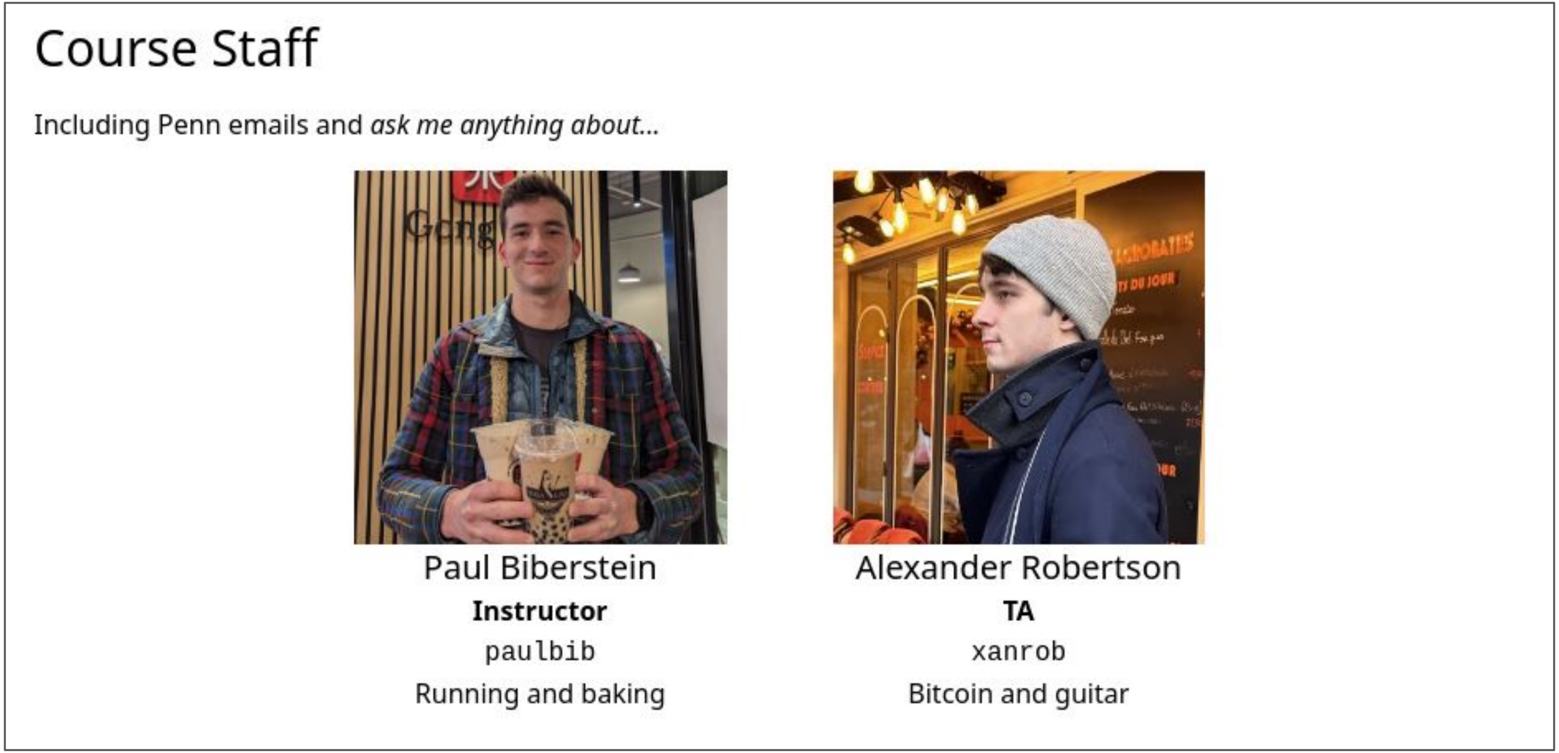


CIS1905



Welcome!

Who are we?



Course Staff

Including Penn emails and *ask me anything about...*

Paul Biberstein

Instructor

`paulbib`

Running and baking

Alexander Robertson

TA

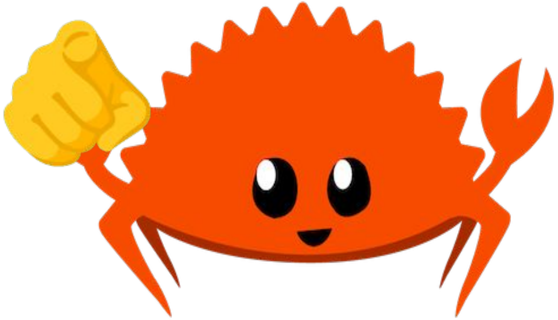
`xanrob`

Bitcoin and guitar

https://www.cis.upenn.edu/~cis1905/2024fall/#course-staff

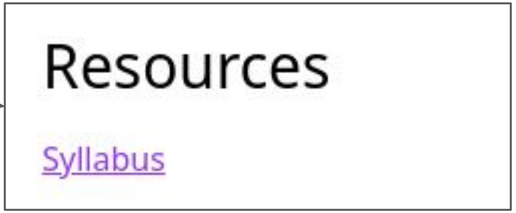
What about you?

- Name
- Year
- Why you're taking the course
- If you were a sea creature, which sea creature would you be?



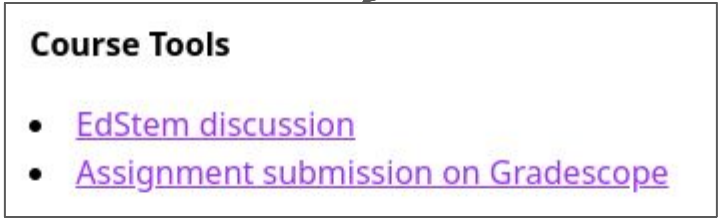
Logistics

https://www.cis.upenn.edu/~cis1905/2024fall/



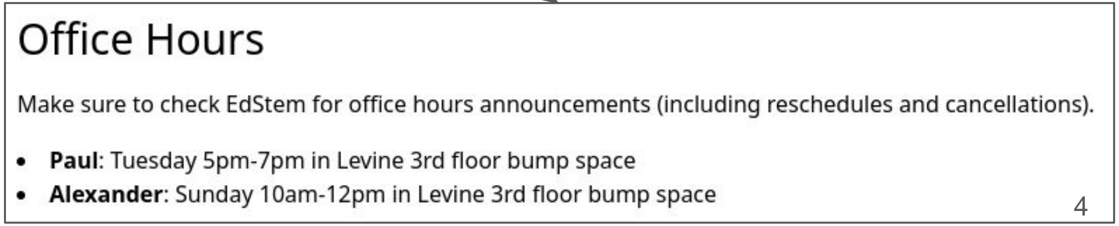
Resources

Syllabus



Course Tools

- EdStem discussion
- Assignment submission on Gradescope



Office Hours

Make sure to check EdStem for office hours announcements (including reschedules and cancellations).

- **Paul**: Tuesday 5pm-7pm in Levine 3rd floor bump space
- **Alexander**: Sunday 10am-12pm in Levine 3rd floor bump space

Assignments

3.5 coding assignments

Post lecture quizzes (⅗ credit for completion, ⅖ for correctness)

Open ended final project (done in groups)

No exams

Final Project

Native GUI applications

- Chat app
- Music player
- Code editor
- Video game

Challenging projects from other domains:

- Graphics: pathtracer, FEM simulation
- Networks: TCP/UDP/IP stack,
- PL: garbage collector, compiler
- DBs: relational database
- OS: filesystem, device driver
- Distributed systems: load balancer, consensus algorithm

Open source contributions

- Contribute a crate to the Rust ecosystem
- Make Rust bindings to a C/C++ library
- Make a fast scientfic computing library with…
    - Python bindings
    - or WebAssembly bindings
- Rewrite a CLI application

Anything else!

- (just check first)

A Brief Rusty History

Early history (2009-2012): personal project by Mozilla employee Graydon Hoare. Sponsored by Mozilla

Pre-release (2012-2015): larger open source community formed, underwent many feature changes trying to find a niche

May 15 2015: Rust 1.0 release, commitment stability

Adoption (2015-2020): Firefox code migrated to Rust, adoption elsewhere in industry

Modern era (2020-present):

- Mozilla lay offs include core Rust contributors
- Rust Foundation started by AWS, Huawei, Google, Microsoft, and Mozilla

Who’s using Rust Now?

Developers: StackOverflow’s most loved language eight years running

CLI tools: `grep` -> `ripgrep` (~5x faster)

Full rewrites: Dropbox core syncing code fully rewritten in Rust

Partial migrations: Firefox (20% of core codebase)

https://github.blog/developer-skills/programming-languages-and-frameworks/why-rust-is-the-most-admired-language-among-developers/

https://github.com/BurntSushi/ripgrep

https://dropbox.tech/infrastructure/rewriting-the-heart-of-our-sync-engine

https://4e6.github.io/firefox-lang-stats/

Rust is a *Systems Programming* Language

One definition: applications that require control of **memory layout** and access to **machine primitives**

Better definition: applications that have strong **correctness** and **performance** requirements.

- OS Kernels
- Databases
- Networking code

But also…

- Scientific computing
- Embedded systems
- Web programming

Why Rust?

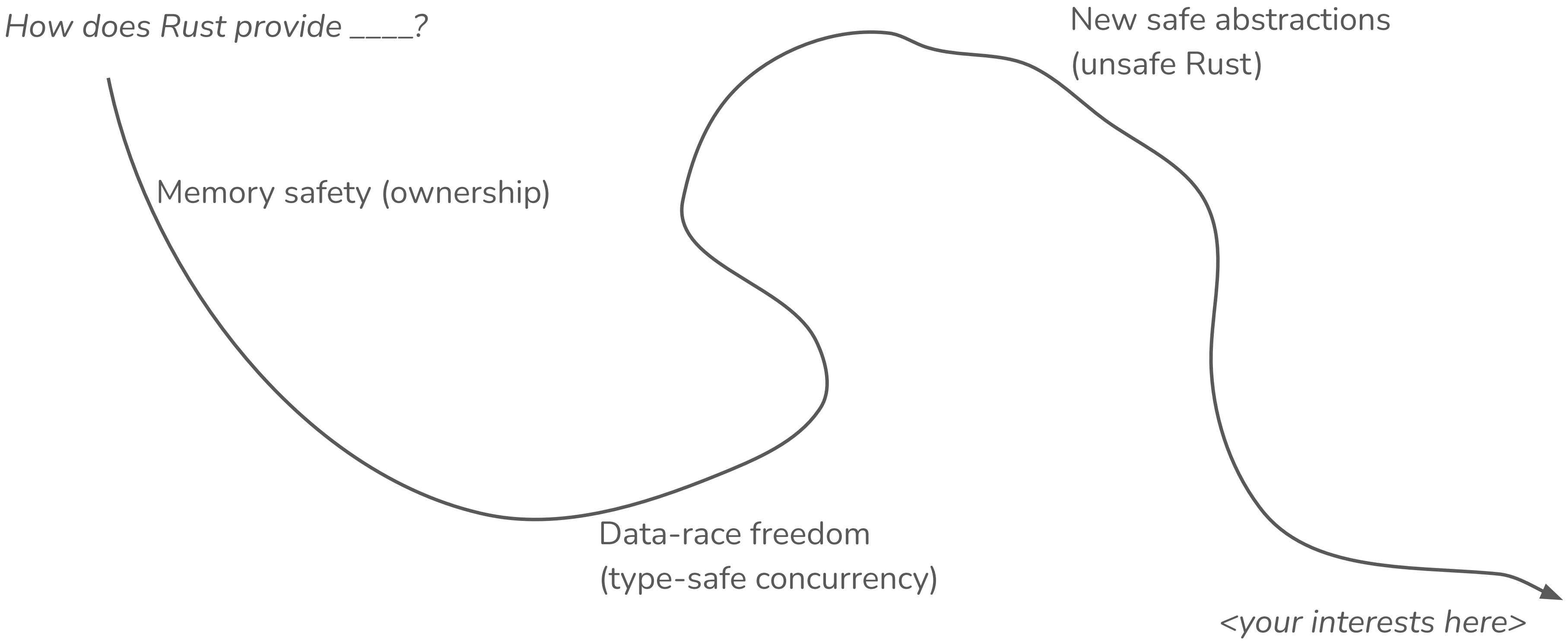
We'll primarily be comparing to C/C++, languages that give more control than nearly anything else out there.

Unfortunately, with great power comes great danger:

- read past buffer
- use-after-free
- double free
- memory leaks
- race conditions

Key question: can you keep the control of C/C++ while not having the dangers?

Course roadmap



How does Rust provide ____?

Memory safety (ownership)

Data-race freedom
(type-safe concurrency)

New safe abstractions
(unsafe Rust)

<your interests here>

Course roadmap

How does Rust provide ?

New safe abstractions

70% of vulnerabilities in Microsoft’s codebases are memory safety

Chrome, Firefox have similar numbers

The source? U.S. Homeland Security: *The Urgent Need for Memory Safety in Software Products*

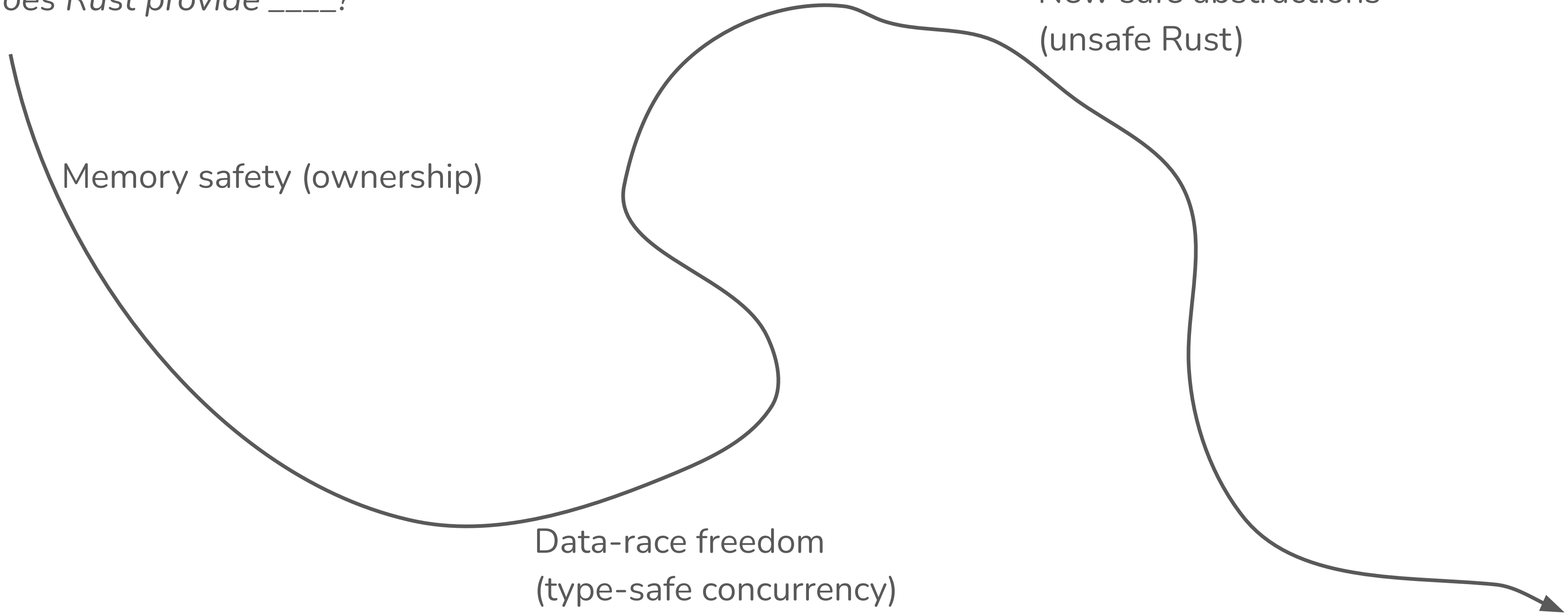
- How can languages address memory safety?

https://www.cisa.gov/news-events/news/urgent-need-memory-safety-software-products

Dennard scaling

Course roadmap

How does Rust provide ____?



Memory safety (ownership)

Data-race freedom
(type-safe concurrency)

New safe abstractions
(unsafe Rust)

<your interests here>

Course roadmap

How does Rust provide ?

New safe abstractions

Parallel Programming is inevitable
multi-core is what separates CPUs from 2005 and today

Unfortunately, parallel programming is hard. Can it be easier?

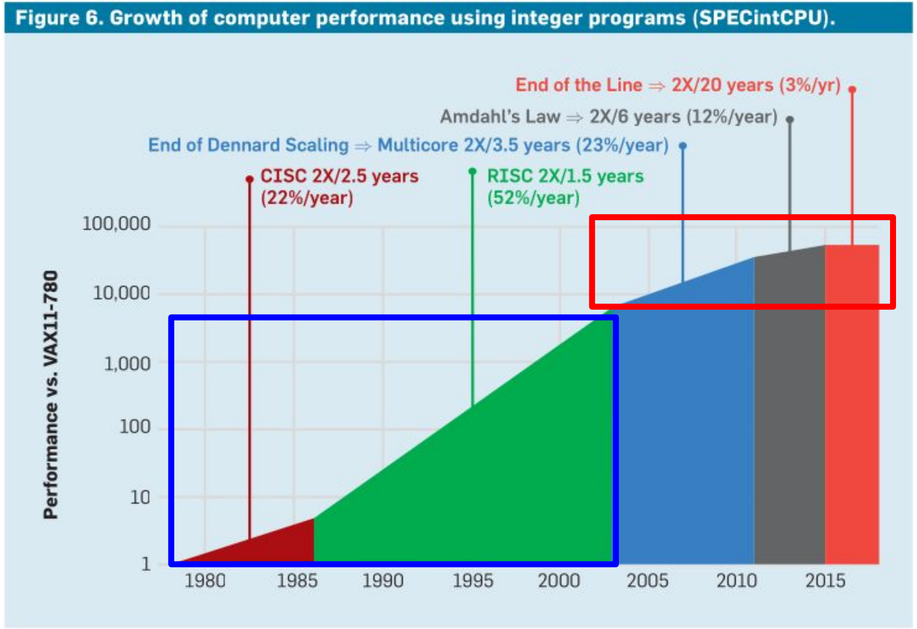
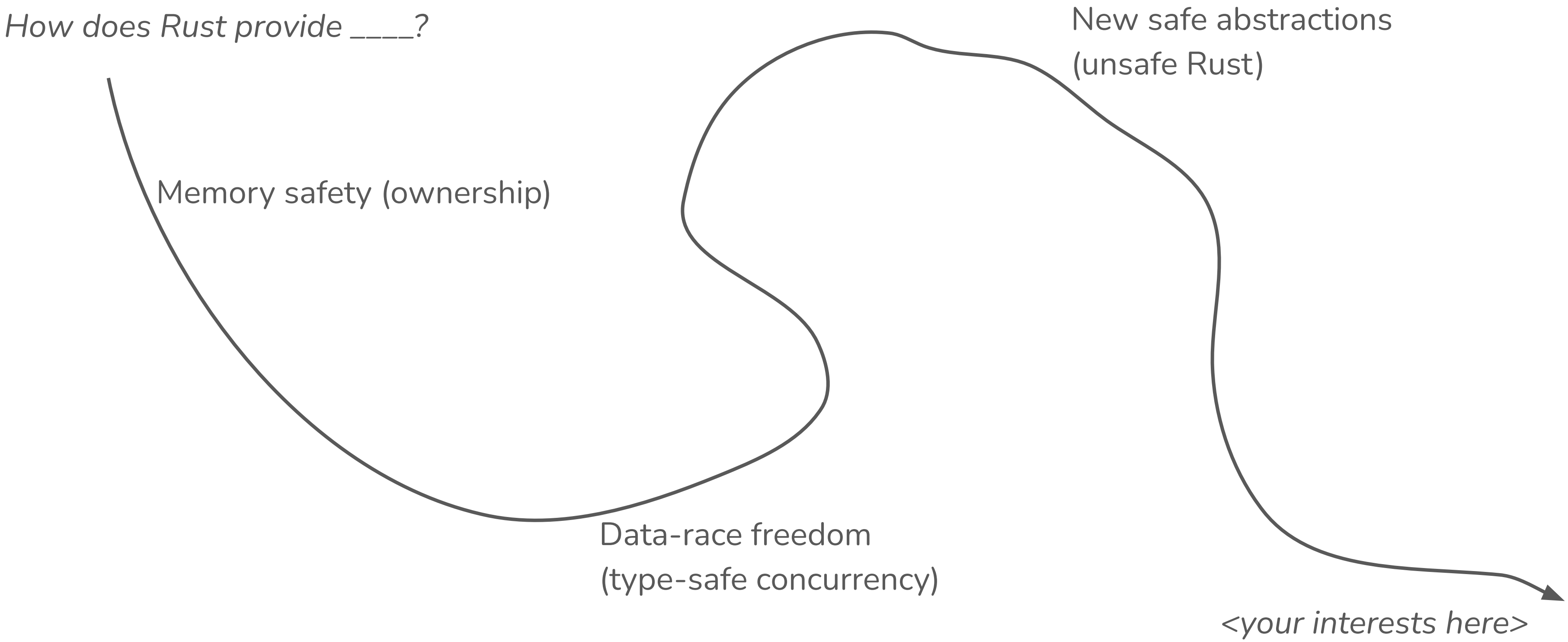


Figure 6. Growth of computer performance using integer programs (SPECintCPU).

Course roadmap



How does Rust provide ____?

Memory safety (ownership)

Data-race freedom
(type-safe concurrency)

New safe abstractions
(unsafe Rust)

<your interests here>

Anatomy of a Rust Program

```
fn main() {
    println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

Anatomy of a Rust Program

```rust
fn main() {
    println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

`println!(…)`

Like C-style printf formatting but…

- Type-inferred
- Type-safe
- No run-time cost

Implemented via macros (we’ll see more later)

Anatomy of a Rust Program

```
fn main() {
   println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
   match n {
       0 | 1 => n,
       _ => fib(n - 1) + fib(n - 2)
    }
}
```

Function Declaration

- Argument types and return types must be annotated

Numeric types

	signedness	
size	i8	u8
	i16	u16
	i32	u32
	i64	u64

f32
f64

Anatomy of a Rust Program

```rust
fn main() {
    println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

Pattern Matching

- Very similar to OCaml
- Very flexible, we’ll see more in future lectures and homework

Anatomy of a Rust Program

```rust
fn main() {
   println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
   match n {
       0 | 1 => n,
       _ => fib(n - 1) + fib(n - 2)
   }
}
```

Anatomy of a Rust Program

```
fn main() {
    println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

Hang on, where are the semicolons?
What about `return`??

Anatomy of a Rust Program

```
fn main() {
    println!("fib(6) = {}", fib(6));
}

fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

```
fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

Anatomy of a Rust Program

```rust
fn main() {
    println!("fib(6) = {}", fib(6));
}
```

```rust
fn fib(n: u64) -> u64 {
    match n {
        0 | 1 => n,
        _ => fib(n - 1) + fib(n - 2)
    }
}
```

```rust
fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

Rust borrows ideas from declarative and imperative programming.
Allows you to balance reasoning about code and performance
(In this case, the left is preferable)

Anatomy of a Rust Program

```rust
fn main() {
    println!("fib(6) = {}", fib(6));
}


fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

Anatomy of a Rust Program

```
fn main() {
    println!("fib(6) = {}", fib(6));
}
```

```
fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

```
fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        n
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        result
    }
}
```

```
fn fib_imperative(n: u64) -> u64 {
    return if n <= 1 {
        n
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        result
    };
}
```

equivalent

Anatomy of a Rust Program

```
fn main() {

}

fr

}
```

Statements and expressions

Semicolon ➡️ sequence statements

```
fn baz() -> u32 {
    100
}
```

⇩

```
fn baz() -> u32 {
    let a = qux();
    let b = buzz();
    a + b
}
```

Braces ➡️ statements in expression context

```
let x = foo();
```

⇩

```
let x = { println!("Calling foo..."); foo() };
```

Anatomy of a Rust Program

```
fn main() {
   println!("fib(6) = {}", fib(6));
}


fn fib_imperative(n: u64) -> u64 {
   if n <= 1 {
       return n;
   } else {
       let mut result = fib(n - 1);
       result += fib(n - 2);
       return result;
   }
}
```

Anatomy of a Rust Program

```
fn main() {
    println!("fib(6) = {}", fib(6));
}


fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

Type inference

- local variables are statically typed, but type inference allows omitting type annotations

Could write

```
let mut result: u64 = fib(n - 1);
```

Anatomy of a Rust Program

```
fn main() {
  println!("fib(6) = {}", fib(6));
}


fn fib_imperative(n: u64) -> u64 {
  if n <= 1 {
      return n;
  } else {
      let ~~mut~~ result = fib(n - 1);
      result += fib(n - 2);
      return result;
  }
}
```

```
error[E0384]: cannot assign twice to immutable variable `result`
 --> fib.rs:17:9
    |
16  |         let result = fib(n - 1);
    |             ------
    |             |
    |             first assignment to `result`
    |             help: consider making this binding mutable: `mut result`
17  |         result += fib(n - 2);
    |         ^^^^^^^^^^^^^^^^^^^^ cannot assign twice to immutable variable

error: aborting due to 1 previous error

For more information about this error, try `rustc --explain E0384`.
```

`mut` keyword

- bindings are immutable by default
- Reverse of C/C++ `const` keyword

Anatomy of a Rust Program

```rust
fn main() {
    println!("fib(6) = {}", fib(6));
}


fn fib_imperative(n: u64) -> u64 {
    if n <= 1 {
        return n;
    } else {
        let mut result = fib(n - 1);
        result += fib(n - 2);
        return result;
    }
}
```

Rapid fire time

Rapid fire time

```
fn main() {
   let s = "foobar"; // string literals

   let x = 1.0 + 2.0 / 3.0 * 4.0; // arithmetic

   let b = true || false; // bools

   let s: bool = 1 < 2; // explicit type annotations

   let c = '😻'; // unicode

   let tup = ('🦀', "Ferris"); // tuples

   while false {
        println!("Uh-oh");
   } // looping
}
```

But how do I run it?

Other languages have many build/package systems to choose from

- C/C++: `make`, `CMake`, `Bazel`, `Ninja`
- Python: `pip`, `poetry`, `setuptools`
- Javascript: `npm`, `yarn`, `webpack`

In Rust, we'll just use `cargo` :)

```
cargo init my-project
cd my-project
cargo add a-cool-dependency
cargo run # or cargo run --release
```

Philosophical Takeaways

Rust emphasizes ~safety~

- immutable by default

Rust emphasizes ~control~

- declarative code for pure functions, imperative code for procedural algorithms

Rust emphasizes ~productivity~

- type inference
- helpful error messages

Quiz time

Does it compile? Should it?

```
fn main() {
    x = 5;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```
fn main() {
    let x = 5;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```
fn main() {
    let mut x = 5;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```
fn main() {
    let x = 5;
    let x = 6;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```rust
fn main() {
    let mut x = 5;
    let x = 6;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```rust
fn main() {
    let mut x = 5;
    x = 6;
    println!("{}", x + 1);
}
```

Does it compile? Should it?

```rust
fn main() {
    let mut x = 5;
    x = ”🦀🦀🦀”;
    println!("{}", x);
}
```

Does it compile? Should it?

```
fn main() {
    x = 5;
    println!("{}", x + 1);
}
```

```
error[E0425]: cannot find value `x` in this scope
--> shadow.rs:2:5
  |
2 |     x = 5;
  |     ^
  |
help: you might have meant to introduce a new binding
  |
2 |    let x = 5;
  |     +++
```

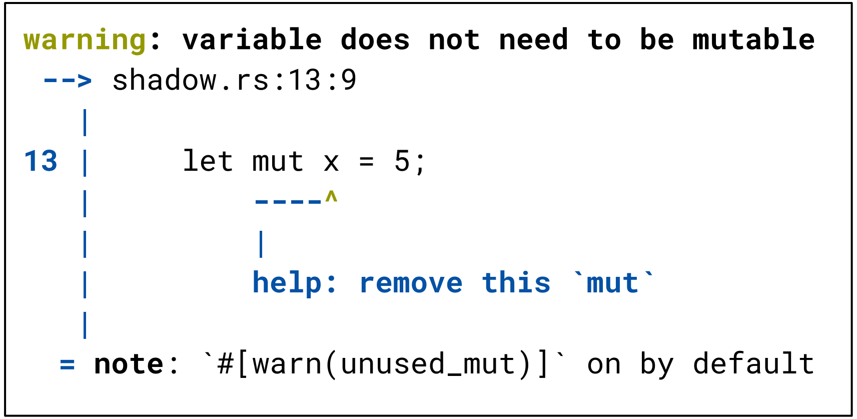
Does it compile? Should it?

```
fn main() {
    let x = 5;
    println!("{}", x + 1);
}
```

✅

Does it compile? Should it?

```rust
fn main() {
    let mut x = 5;
    println!("{}", x + 1);
}
```



```
warning: variable does not need to be mutable
 --> shadow.rs:13:9
   |
13 |     let mut x = 5;
   |         ----^
   |         |
   |         help: remove this `mut`
   |
  = note: `#[warn(unused_mut)]` on by default
```

Does it compile? Should it?

```
fn main() {
    let x = 5;
    let x = 6;
    println!("{}", x + 1);
}
```

✅

Does it compile? Should it?

```
fn main() {
    let mut x = 5;
    let x = 6;
    println!("{}", x + 1);
}
```

✅

Does it compile? Should it?

```
fn main() {
    let mut x = 5;
    x = 6;
    println!("{}", x + 1);
}
```

✅

Does it compile? Should it?

```
fn main() {
    let mut x = 5;
    x = ”🦀🦀🦀”;
    println!("{}", x);
}
```

```
error[E0308]: mismatched types
 --> shadow.rs:37:9
   |
36 |     let mut x = 5;
   |                 - expected due to this value
37 |     x = "🦀🦀🦀";
   |         ^^^^^^^^ expected integer, found `&str`
```

Does it compile? Should it?

```
fn foo() -> u32 {
    let x = 5;
    x
}
```

✅

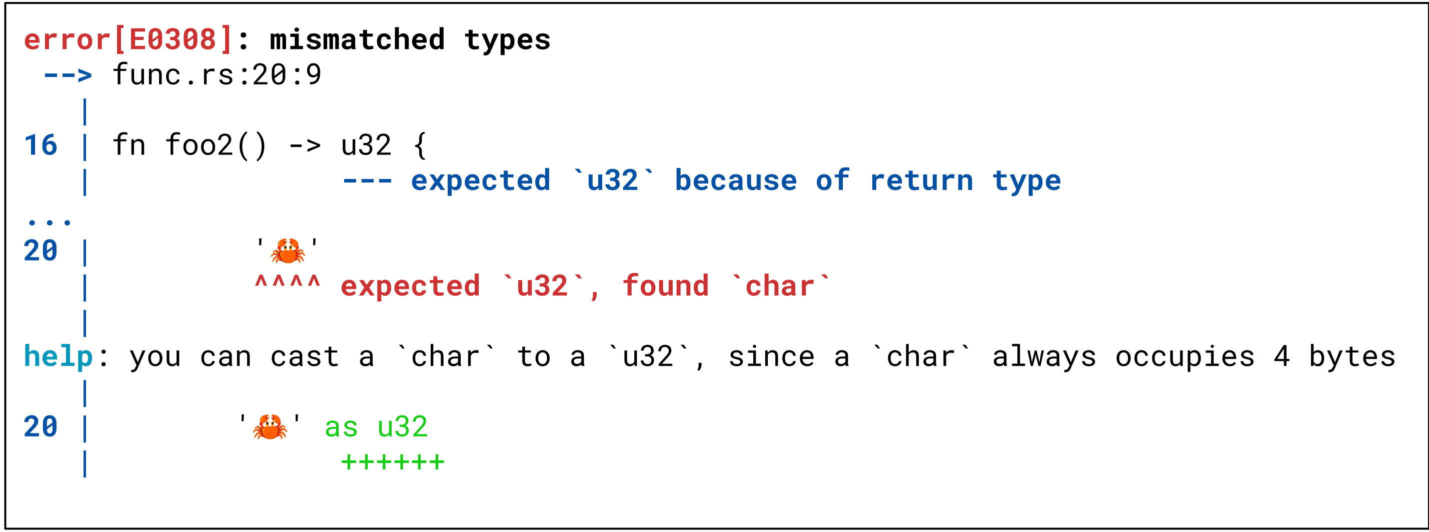
Does it compile? Should it?

```
fn foo() -> u32 {
    if true {
        1
    } else {
        2
    }
}
```

✅

Does it compile? Should it?

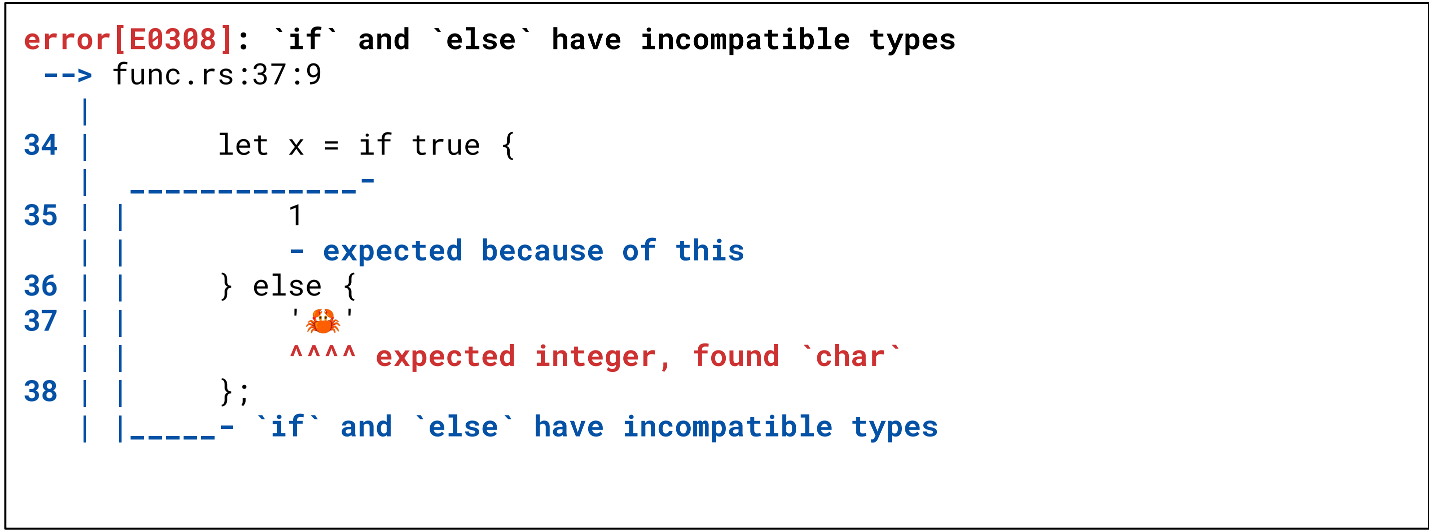
```
fn foo() -> u32 {
    if true {
        1
    } else {
        '🦀'
    }
}
```



```
error[E0308]: mismatched types
 --> func.rs:20:9
   |
16 | fn foo2() -> u32 {
   |              --- expected `u32` because of return type
...
20 |         '🦀'
   |         ^^^^ expected `u32`, found `char`
   |
help: you can cast a `char` to a `u32`, since a `char` always occupies 4 bytes
   |
20 |        '🦀' as u32
   |              ++++++
```

Does it compile? Should it?

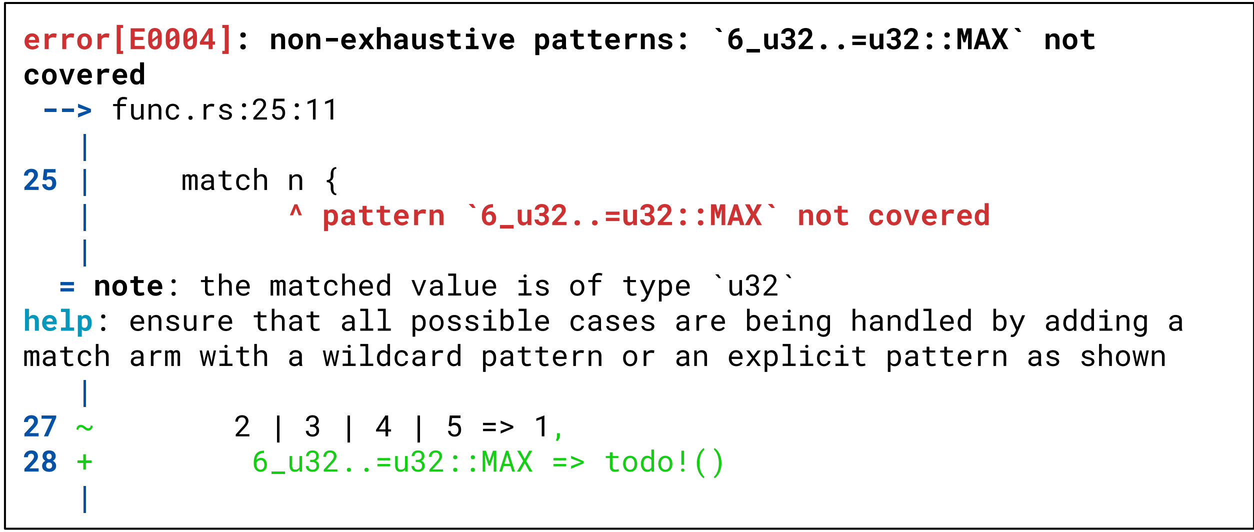
```
fn foo() {
    let x = if true
        1
    } else {
        '🦀'
    };
}
```



```
error[E0308]: `if` and `else` have incompatible types
 --> func.rs:37:9
   |
34 |       let x = if true {
   |  _____________-
35 | |         1
   | |         - expected because of this
36 | |     } else {
37 | |         '🦀'
   | |         ^^^^ expected integer, found `char`
38 | |     };
   | |_____- `if` and `else` have incompatible types
```

Does it compile? Should it?

```
fn foo(n: u32) -> u32 {
    match n {
        0 | 1 => 0,
        2 | 3 | 4 | 5 => 1
    }
}
```



```
error[E0004]: non-exhaustive patterns: `6_u32..=u32::MAX` not
covered
 --> func.rs:25:11
   |
25 |     match n {
   |           ^ pattern `6_u32..=u32::MAX` not covered
   |
  = note: the matched value is of type `u32`
help: ensure that all possible cases are being handled by adding a
match arm with a wildcard pattern or an explicit pattern as shown
   |
27 ~         2 | 3 | 4 | 5 => 1,
28 +          6_u32..=u32::MAX => todo!()
   |
```

Final notes

- Make sure you're on EdStem and Gradescope
- Bookmark course website
- Project 0 released soon: exercises to get you more familiar with Rust
- Complete post-lecture quiz

First class attendance form
(if still on waitlist)

Slide Credits

Inspiration from:

https://www.cis.upenn.edu/~cis1905/2024spring/

https://github.com/trifectatechfoundation/teach-rs

https://www.cs.umd.edu/class/fall2021/cmsc388Z/