The Secret Project

Sophia Turner

My Background

Started at Mozilla at 2016 - helped create compiler errors, IDE support, and survey

Helped start the Rust ergonomics initiative

Worked as part of the Rust core team

Helped create the Rust Leadership Council

My Background

Started at Mozilla at 2016 - helped create compiler errors, IDE support, and survey

Helped start the Rust ergonomics initiative

Worked as part of the Rust core team

Helped create the Rust Leadership Council

Spent two years teaching Rust professionally

What's the secret project?

Is it possible to create a safe systems language that's

- Easy to read
- Easy to learn
- Easy to teach

Introducing: June

Introducing: June

Safe systems language with focus on:

- Readability
- Teachability
- Learn-ability
- Maintainability

A sister language to Rust, with a different set of trade-offs

Review (Rust)

```
struct Node {
    data1: &Data
    data2: &Data
    data3: &Data
}
```

Review (Rust)

```
struct Node<'a,'b,'c> {
    data1: &'a Data
    data2: &'b Data
    data3: &'c Data
}
```

Review (Rust)

```
struct Node<'a,'b,'c> {
    data1: &'a Data
    data2: &'b Data
    data3: &'c Data
}
```

- Lifetimes
- Lifetime annotations
- Lifetime parameters
- Ownership and borrowing
- Generics

Thinking in groups

Show me some code

Lifetimes, part 1

```
struct Employee {
    name: c_string,
    age: i64,
}

fun main() {
    let employee = new Employee(name: c"Bob", age: 123)
    println(employee.age)
}
```

Lifetimes, part 1

```
struct Employee {
    name: c_string,
    age: i64,
}

fun main() {
    let employee = new Employee(name: c"Bob", age: 123)
    println(employee.age)
}
```

Lifetimes, part 2

```
struct Stats {
    age: i64
}

struct Employee {
    name: c_string,
    stats: Stats,
}

fun set_stats(mut employee: Employee) {
    employee.stats = new Stats(age: 33)
}

fun main() {
    mut employee = new Employee(name: c"Sophia", stats: new Stats(age: 22))
    set_stats(employee)
    println(employee.stats.age)
}
```

Lifetimes, part 2

```
struct Stats {
    age: i64
}

struct Employee {
    name: c_string,
    stats: Stats,
}

fun set_stats(mut employee: Employee) {
    employee.stats = new Stats(age: 33)
}

fun main() {
    mut employee = new Employee(name: c"Sophia", stats: new Stats(age: 22))
    set_stats(employee)
    println(employee.stats.age)
}
```

A linked list

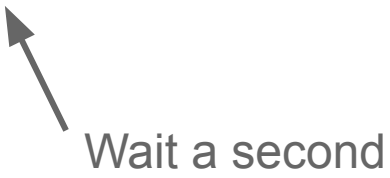
```
struct Node {
    data: i64
    next: Node?
}

fun main() {
    let node3 = new Node(data: 3, next: none)
    let node2 = new Node(data: 2, next: node3)
    let node1 = new Node(data: 1, next: node2)
}
```

A circular linked list

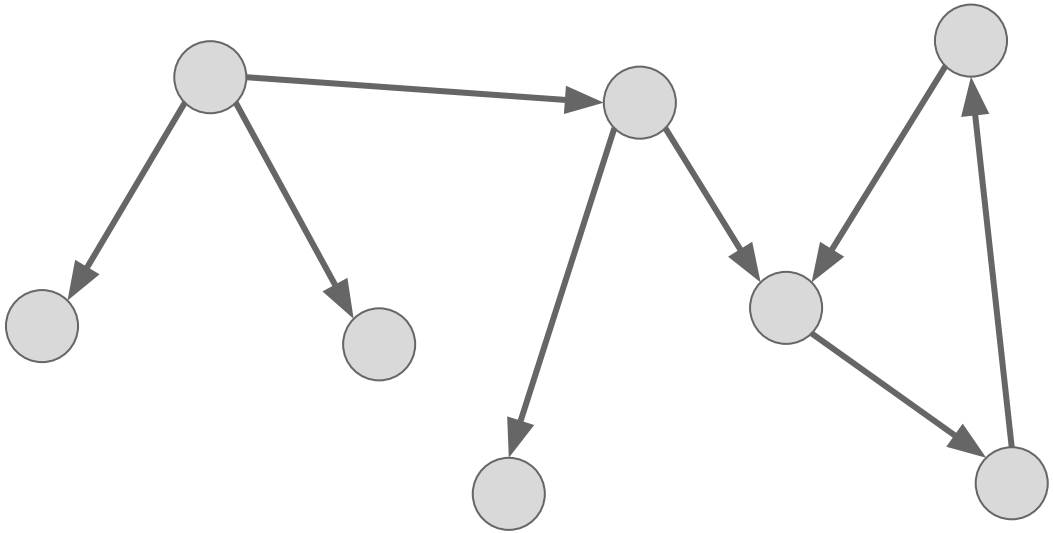
```
struct Node {
    data: i64
    next: Node?
}

fun main() {
    mut node3 = new Node(data: 3, next: none)
    let node2 = new Node(data: 2, next: node3)
    let node1 = new Node(data: 1, next: node2)

    node3.next = node1
}
```

A circular linked list

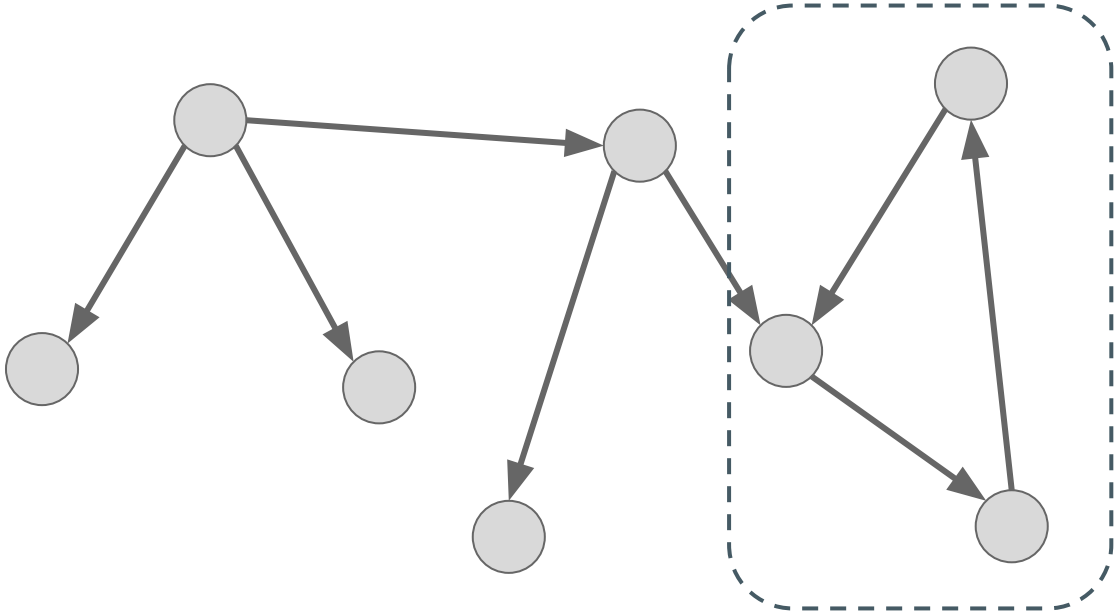
```
struct Node {
    data: i64
    next: Node?
}

fun main() {
    mut node3 = new Node(data: 3, next: none)
    let node2 = new Node(data: 2, next: node3)
    let node1 = new Node(data: 1, next: node2)

    node3.next = node1
}
```



The problem (as seen by graphs)



The problem (as seen by graphs)



Make encapsulation stronger

Same rules as regular encapsulation

-and-

Don’t allow copies of pointers to leak in or out

Let's build an abstraction

```
struct Node {
    data: i64
    next: Node?
}

class CircularList {
    node: Node?

    fun default() -> owned CircularList {
        return new owned CircularList(node: none)
    }

    // …
}
```

Fully encapsulated values

...can move between threads

...can create cleaner APIs

...can create a memory boundary for safe memory recycling

Safe memory recycling

How do we make free lists safe?

Need to know if a pointer has any additional copies

Need to give the user a way to recycle

Safe memory recycling

Limit safe memory recycling to full encapsulations

Create a `recycle` keyword

Check recycled pointers (and pointers they point to) are isolated, and if so, allow them to be reused

Linear time operation, so we make this manual

Lots of roads ahead

Continue building reference compiler

Better abstractions

Rust interop

More exploration of easy-to-use safety mechanisms

Follow for more info

Twitter: @sophiajturner

Mastodon: @sophiajt@fosstodon.org

Questions?