KUBOS

Corroding Space

Considering Rust as a Flight Software Language

— Ryan Plauche (ryan@kubos.com)

KUBOS

This Presentation is about

KUBOS

This Presentation is about

- What is Rust?

KUBOS

This Presentation is about

- What is Rust?
- Why do we use Rust?

KUBOS

This Presentation is about

- What is Rust?
- Why do we use Rust?
- Why do we like Rust?

KUBOS

This Presentation is about

- What is Rust?
- Why do we use Rust?
- Why do we like Rust?
- Why hesitate about Rust?

KUBOS

This Presentation is not about

KUBOS

This Presentation is not about

- Why you should only use Rust

KUBOS

This Presentation is not about

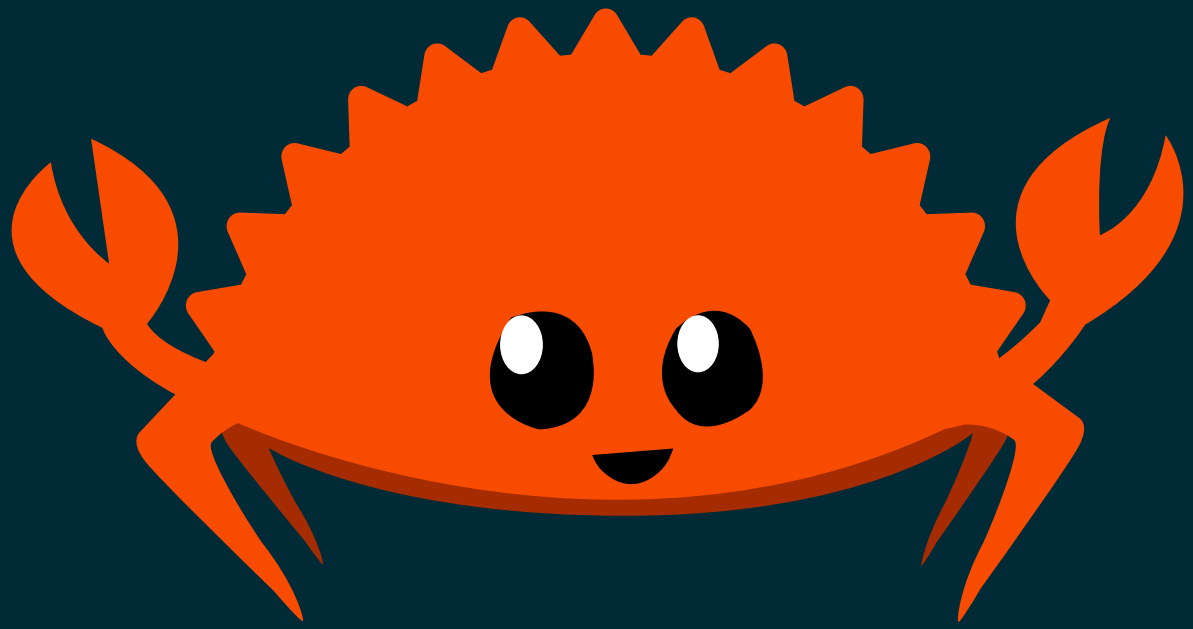
- Why you should only use Rust
- Why other languages are terrible

KUBOS

This Presentation is not about

- Why you should only use Rust
- Why other languages are terrible
- Which language has the best mascot

KUBOS



Background

Background

- Software Engineer for Kubos

KUBOS

Background

- Software Engineer for Kubos
- Rust-based Flight Software

KUBOS

Background

- Software Engineer for Kubos
- Rust-based Flight Software
- github.com/kubos/kubos

KUBOS

What is Rust?

KUBOS

Rust in 16 words

Rust is a systems programming language that runs blazingly fast, prevents segfaults and guarantees thread safety.

From - https://www.rust-lang.org/

KUBOS

Rust in 6 points

KUBOS

Rust in 6 points

- Statically Compiled

KUBOS

Rust in 6 points

- Statically Compiled
- LLVM Based

KUBOS

Rust in 6 points

- Statically Compiled
- LLVM Based
- Strong & Expressive Types

KUBOS

Rust in 6 points

- Statically Compiled
- LLVM Based
- Strong & Expressive Types
- Compile Time Ownership Checks

KUBOS

Rust in 6 points

- Statically Compiled
- LLVM Based
- Strong & Expressive Types
- Compile Time Ownership Checks
- Static Garbage Collector

KUBOS

Rust in 6 points

- Statically Compiled
- LLVM Based
- Strong & Expressive Types
- Compile Time Ownership Checks
- Static Garbage Collector
- Rich Standard Library

KUBOS

Why do we use Rust?

KUBOS

“How do you account for the high error rate of C?”

KUBOS

Customer Centric

KUBOS

GraphQL

KUBOS

Why do we like Rust?

- Error Handling
- Traits & Generics
- Ownership
- Testing
- Tooling

Error Handling

KUBOS

Enums for Errors

```
pub enum DeviceError {
    ReadTimeout,
    WriteTimeout,
    ErrorReturned,
    CommandFailed,
}
```

KUBOS

Expressive Errors

```
pub enum DeviceCommand {
    EnableLED(u16),
    DisableLED(u16),
}

pub enum DeviceError {
    ReadTimeout,
    WriteTimeout,
    ErrorReturned(String),
    CommandFailed(DeviceCommand),
}
```

The Result Type

```
enum Result<T, E> {
   Ok(T),
   Err(E),
}

fn read_memory(addr: u32) -> Result<u32, DeviceError> {
    ...
    if read_ok {
        Ok(0x100)
    } else {
        Err(DeviceError::ReadTimeout)
    }
}
```

KUBOS

Returning Errors

```
pub fn read_memory(addr: u32) -> Result<u32, DeviceError> { ... }

pub fn read_status() -> Result<bool, DeviceError> {
    let mem_value = match read_memory(0x100) {
        Ok(num) => num,
        Err(e) => return Err(e),
    };

    return validate_status(mem_value);
}
```

Returning Errors

```
pub fn read_memory(addr: u32) -> Result<u16, DeviceError> { ... }

pub fn read_status() -> Result<bool, DeviceError> {
    let mem_value = read_memory(0x100)?;

    return validate_status(mem_value);
}
```

KUBOS

Traits & Generics

KUBOS

Defining Behavior with Traits

```
pub trait DeviceLogger {
    fn name(&self) -> String;
    fn status(&self) -> DeviceStatus;
    fn error(&self) -> DeviceError;
}
```

Defining Behavior with Traits

```
pub struct StarTracker {}

impl DeviceLogger for StarTracker {
    fn name(&self) -> String {
        "StarTracker".to_owned()
    }
    fn status(&self) -> DeviceStatus {
        self.read_status()
    }

    fn error(&self) -> DeviceError {
        self.read_error()
    }
}
```

KUBOS

Generics + Traits

```
fn log_device_status<T: DeviceLogger>(device: &T) -> () {
    println!("{}:status:{}", device.name(), device.status());
    println!("{}:error:{}", device.name(), device.error());
}
```

KUBOS

Ownership & References

```
pub struct Device { status: bool }

pub fn trust_me_or_not(dev: &Device) -> () {
    ...
}
```

KUBOS

Ownership & References

```
pub struct Device { status: bool }

pub fn trust_me_really(dev: &mut Device) -> () {
    ...
}
```

Testing

```rust
#[cfg(test)]
mod tests {

    #[test]
    fn simple_test() {
        assert_eq!(
            10, 10
        );
    }
}
```

KUBOS

Tooling

Cargo

```
$ cargo
```

Manage Dependencies

```
[package]
name = "kubos-file-client"
version = "0.1.0"
authors = ["Ryan Plauche <ryan@kubos.co>"]

[dependencies]
clap = "2.32"
simplelog = "^0.5.0"
log = "^0.4.0"
file-protocol = { path = "../../libs/file-protocol" }
failure = "0.1.2"
```

KUBOS

Build Your Code

```
$ cargo build
```

KUBOS

Test Your Code

```
$ cargo test
```

Run Your Code

```
$ cargo run
```

KUBOS

Cross Compile Your Code

```
$ cargo build --target arm-unknown-linux-gnueabi
```

Generate Your Docs

```
$ cargo doc
```

KUBOS

Why hesitate about Rust?

KUBOS

Why hesitate about Rust?

- Overall Maturity

KUBOS

Why hesitate about Rust?

- Overall Maturity
- Toolchain Availability

KUBOS

Why hesitate about Rust?

- Overall Maturity
- Toolchain Availability
- Young Ecosystem

KUBOS

Why hesitate about Rust?

- Overall Maturity
- Toolchain Availability
- Young Ecosystem
- RTOS Support

KUBOS

Thank You