Rust and the Crazyflie Workshop

Arnaud (Bitcraze)
BAM days
October 20th 2021

BAM days
19 -21 Oct.
bitcraze Awesome Meetup Days

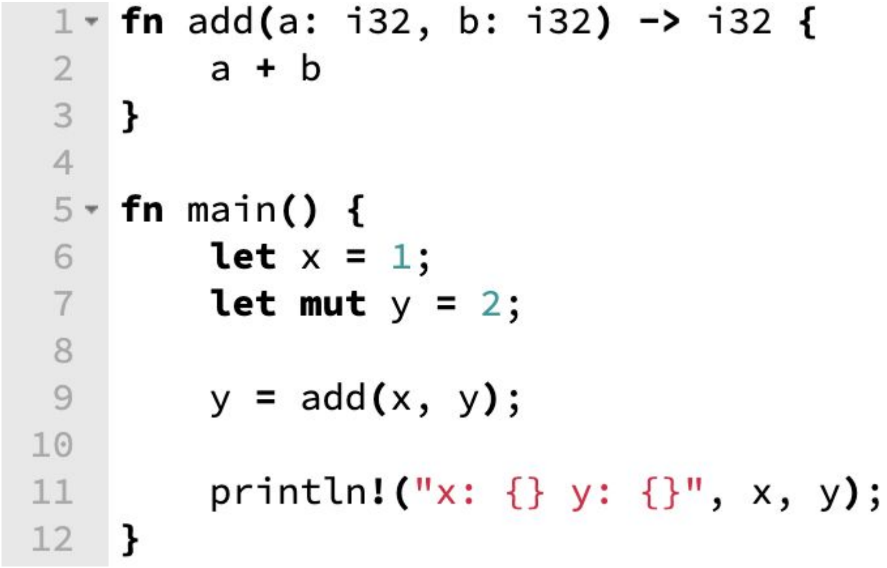
The Rust language

- Started in 2009 at Mozilla
- 1.0 in 2015
- Stable language
  - Stability guarantee since 1.0
  - Optional editions: 2015, 2018 and 2021. Improves the language without breaking compatibility.
  - Rust foundation started in 2021
- Performant
  - Compiles to machine code (using LLVM)
  - Allows low level access to the machine
- Reliable
  - Strongly typed with type inference
  - Memory safe: no possible data race
- Productive
  - Modern tooling, package manager, Convention over Configuration
  - Helpful compiler error message: a bad error message is considered a bug by the compiler team

Rust at Bitcraze

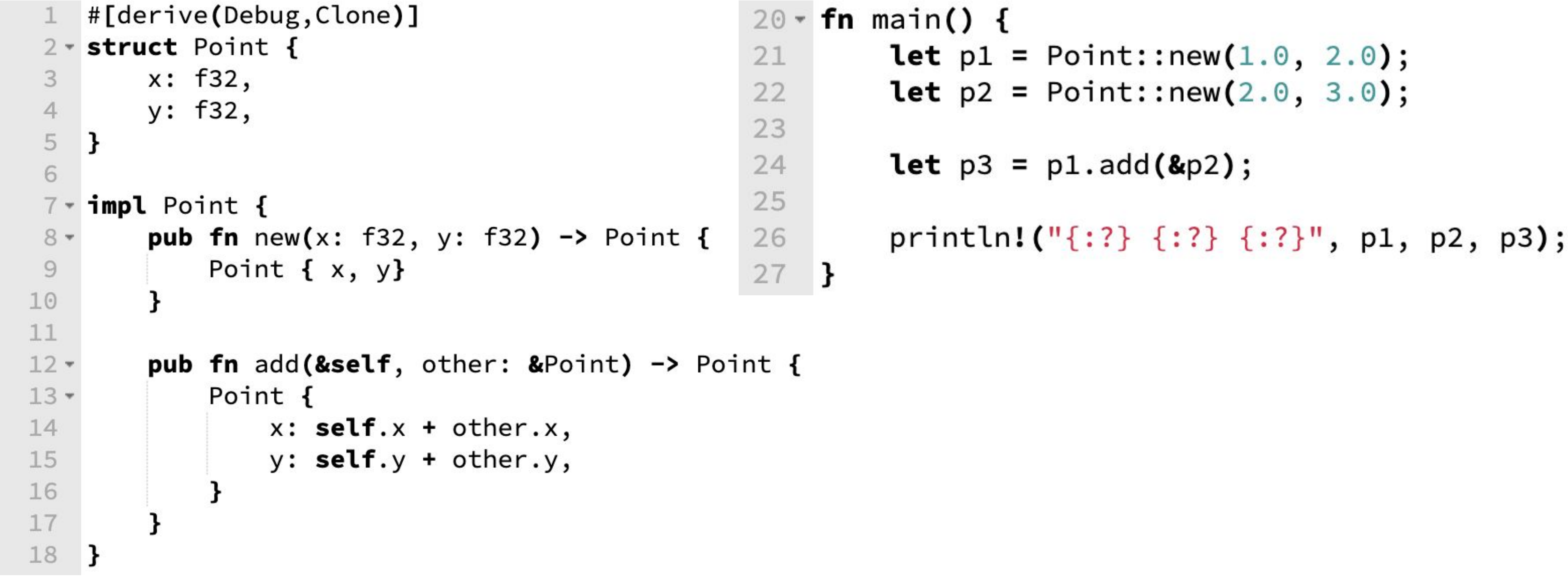
- Shipping printer
- Crazyradio/Crazyflie-link/Crazyflie-lib
  - Base for a web-client when compiled to Wasm
  - Binding possible to Python, C++, Ros, …
  - Crazy-mouse
- Rust in the firmware
  - Deck driver
  - Crazyflie APP <=- This talk
  - Crazyflie2-stm bootloader re-implentation

Intro to Rust: Variables and Functions



```rust
fn add(a: i32, b: i32) -> i32 {
    a + b
}

fn main() {
    let x = 1;
    let mut y = 2;

    y = add(x, y);

    println!("x: {} y: {}", x, y);
}
```

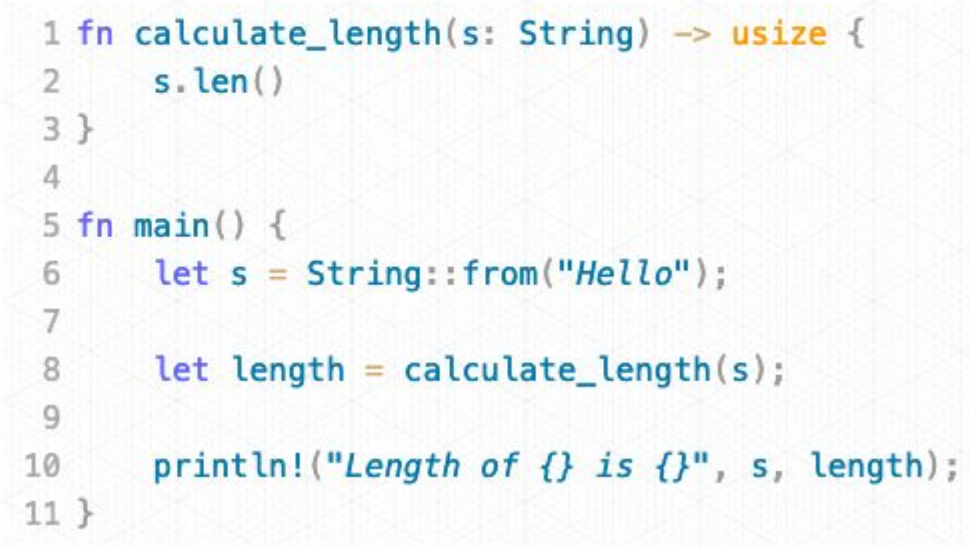
Intro to Rust, Struct and Impl



```rust
#[derive(Debug,Clone)]
struct Point {
    x: f32,
    y: f32,
}

impl Point {
    pub fn new(x: f32, y: f32) -> Point {
        Point { x, y}
    }

    pub fn add(&self, other: &Point) -> Point {
        Point {
            x: self.x + other.x,
            y: self.y + other.y,
        }
    }
}
```

```rust
fn main() {
    let p1 = Point::new(1.0, 2.0);
    let p2 = Point::new(2.0, 3.0);

    let p3 = p1.add(&p2);

    println!("{:?} {:?} {:?}", p1, p2, p3);
}
```

Intro to rust: Ownership

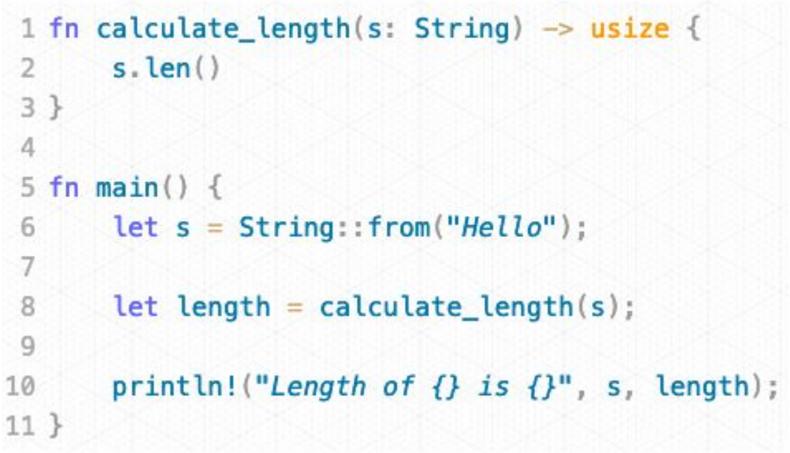


```rust
fn calculate_length(s: String) -> usize {
    s.len()
}

fn main() {
    let s = String::from("Hello");

    let length = calculate_length(s);

    println!("Length of {} is {}", s, length);
}
```

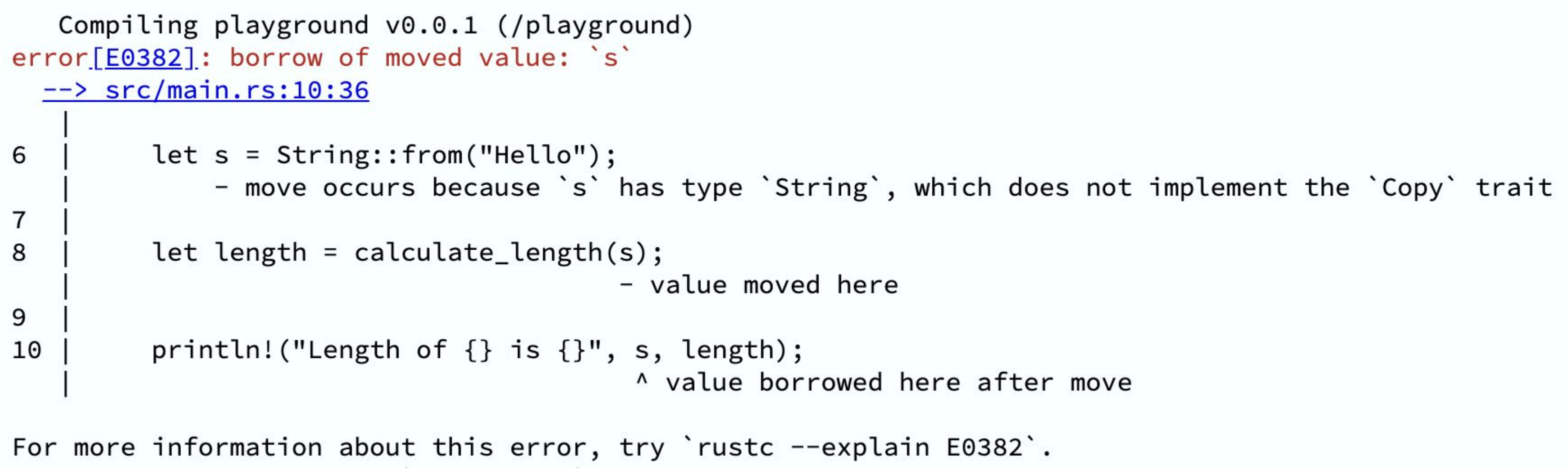
- Simple enough code but…
- This will not compile!

Intro to rust: Ownership

Link to Rust playground

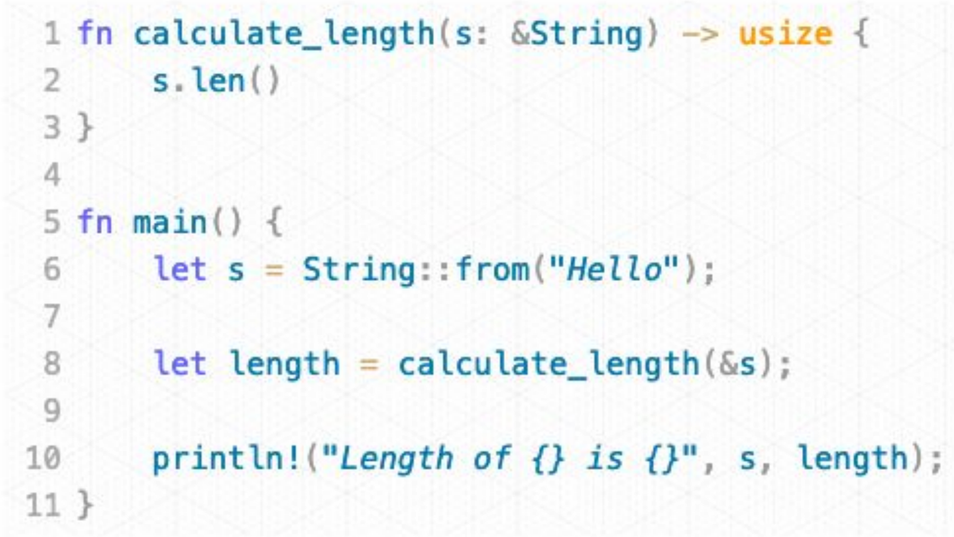


```
1 fn calculate_length(s: String) -> usize {
2     s.len()
3 }
4
5 fn main() {
6     let s = String::from("Hello");
7
8     let length = calculate_length(s);
9
10    println!("Length of {} is {}", s, length);
11 }
```

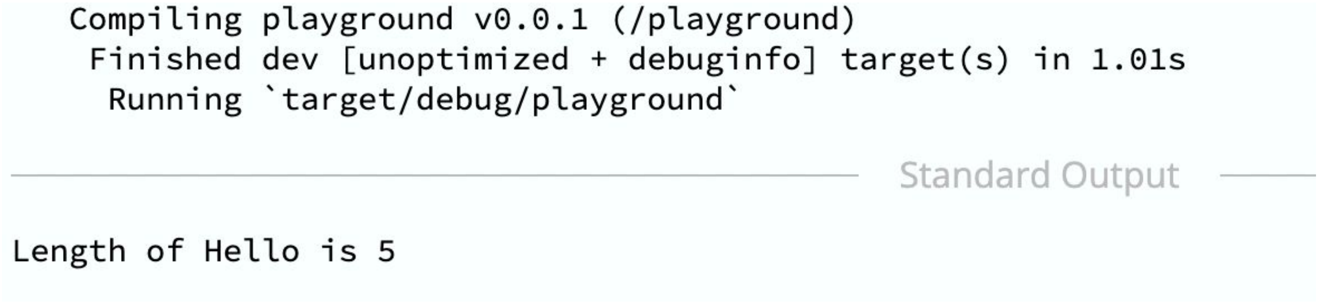


```
   Compiling playground v0.0.1 (/playground)
error[E0382]: borrow of moved value: `s`
  --> src/main.rs:10:36
   |
6  |     let s = String::from("Hello");
   |         - move occurs because `s` has type `String`, which does not implement the `Copy` trait
7  |
8  |     let length = calculate_length(s);
   |                                   - value moved here
9  |
10 |     println!("Length of {} is {}", s, length);
   |                                    ^ value borrowed here after move

For more information about this error, try `rustc --explain E0382`.
```

Intro to rust: Ownership



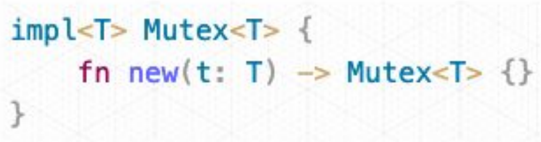
```rust
fn calculate_length(s: &String) -> usize {
    s.len()
}

fn main() {
    let s = String::from("Hello");

    let length = calculate_length(&s);

    println!("Length of {} is {}", s, length);
}
```



```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 1.01s
     Running `target/debug/playground`
```

```
Length of Hello is 5
```

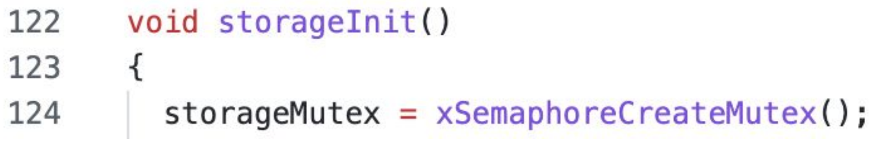
Ownership for better API: Mutex



```
impl<T> Mutex<T> {
    fn new(t: T) -> Mutex<T> {}
}
```

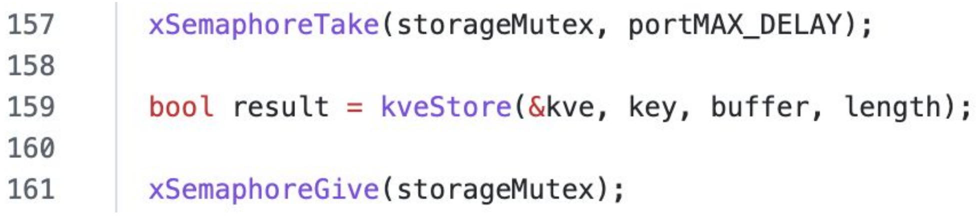
Mutex usage in storage.c

Mutex init is decoupled from what it protects



```
void storageInit()
{
  storageMutex = xSemaphoreCreateMutex();
```

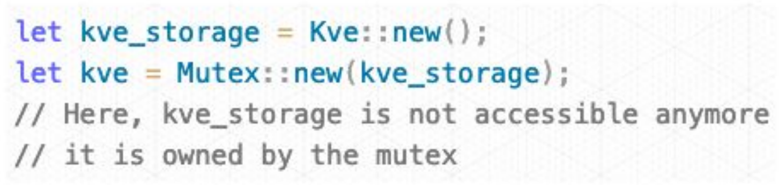
Locking and unlocking the mutex is manual:



```
xSemaphoreTake(storageMutex, portMAX_DELAY);

bool result = kveStore(&kve, key, buffer, length);

xSemaphoreGive(storageMutex);
```

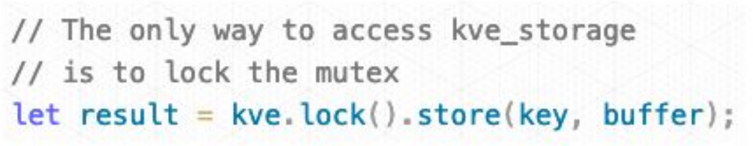
Possible implementation in Rust

Mutex takes ownership of what it protects



```
let kve_storage = Kve::new();
let kve = Mutex::new(kve_storage);
// Here, kve_storage is not accessible anymore
// it is owned by the mutex
```

Impossible to use the protected object without locking the mutex:



```
// The only way to access kve_storage
// is to lock the mutex
let result = kve.lock().store(key, buffer);
```

Rust in embedded

- Little to no runtime
- Performant, compile to machine code
- Standard library optional: no_std
- Lots of common crates supports no_std
    - Data serialization/deserialization
    - Cryptography
    - A growing ecosystem of embedded-specific crate (eg. heapless)
- Embedded-hal: interface standardisation to allow for hardware abstracted programs and drivers
- Type-safe hardware drivers!
- Great tooling (eg. probe-run, defmt)

Lets code!

Future?

- Finishing the Rust Crazyflie-lib
- Experimenting with Rust in the firmware:
    - Crazyflie-sys and Crazyflie-app crate in crates.io
    - Would allow to “just” add *crazyflie-app=”2021.02”* to cargo.toml to get started
- Some future utility firmware might be written in Rust (ie. Bootloader, Crazyradio or Crazyflie’s radio nRF firmware would be good candidates)
- No current plan to (re)write any major firmwares in Rust

Questions?