Introduction to Rust

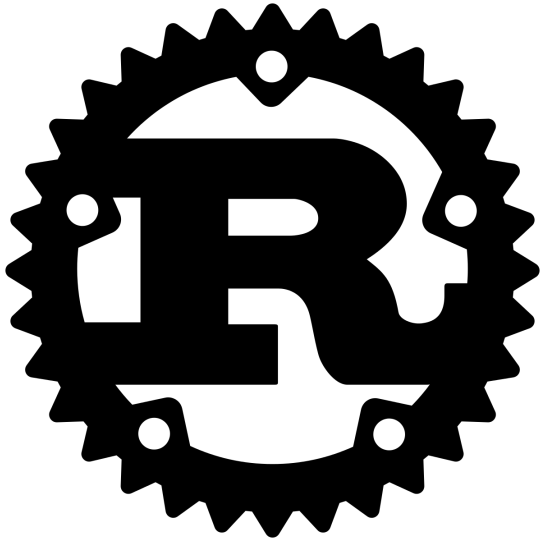
Jon Drobny

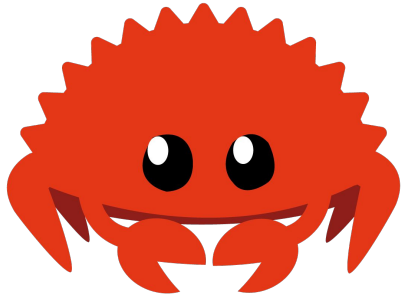
THW Illinois, April 8 2020

What is Rust?

Low-level, high-performance programming language, developed by Mozilla

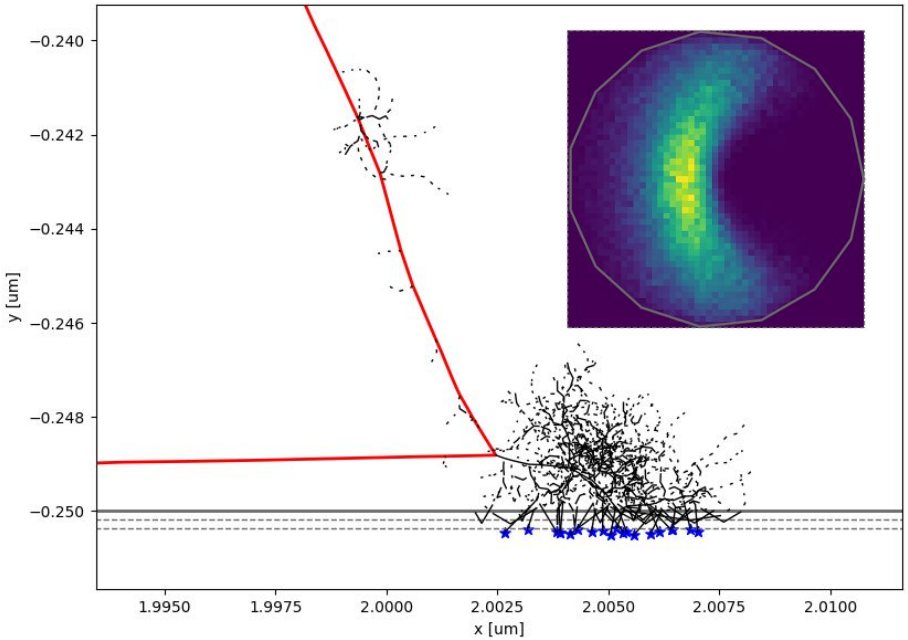
Documentation kept up date in The Rust Book:
https://doc.rust-lang.org/book/title-page.html

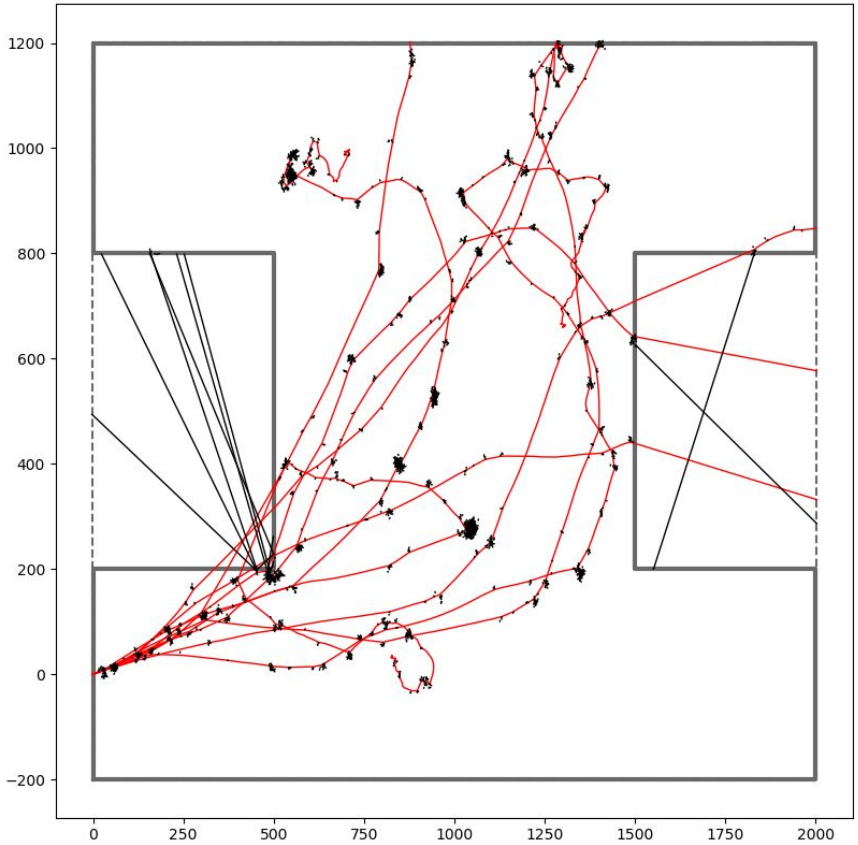




What have I written in Rust?

https://github.com/drobnyjt/rustbca





Why Rust?

- High performance, on par with idiomatic C/C++
- Complete memory safety: Rust guarantees memory safety without a garbage collector through ownership and the borrow checker - no more pointer woes
- Effortless concurrency - no possibility of data races
- Useful documentation, an open official textbook including *helpful compiler error messages*!
- Standardized package manager (Cargo) and easy-to-use toolchain (Rustup)
- Modern programming paradigms

Mutability, ownership and the borrow-checker

Explicit mutability and distinguishing between declaration and assignment

```
let x: i32 = 5;
let mut y: f64 = 1.;
x = 1; //ERROR
y = 2.; //NO ERROR
```

Ownership - no garbage collector, no pointers

Rules of ownership, from the Rust Book:

- Each value in Rust has a variable that's called its *owner*.
- There can only be one owner at a time.
- When the owner goes out of scope, the value will be dropped.

Borrowing

Ownership can be borrowed temporarily (e.g., in a function), and it goes back once the borrower goes out of scope

Code Shepherding*: "It's not what programming languages do, it's what they shepherd you to"
Rust shepherds developers to write good code

*http://nibblestew.blogspot.com/2020/03/its-not-what-programming-languages-do.html

Ownership and borrowing examples

https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

Helpful compiler error messages!

Not only are the compiler error messages useful, but the rust compiler has an `--explain` command that provides examples and documentation explaining every compiler error!

```
C:\projects\rustbca>cargo build
warning: unused manifest key: profile.release.target-cpu
   Compiling rustBCA v0.1.0 (C:\projects\rustbca)
error[E0423]: expected function, found macro `println`
   --> src\main.rs:794:5
    |
794 |     println("");
    |     ^^^^^^^ help: use `!` to invoke the macro: `println!`
```

```
It is common to forget the trailing `!` on macro invocations, which would also
yield this error:

```
println("");
// error: expected function, tuple struct or tuple variant,
// found macro `println`
// did you mean `println!(...)`? (notice the trailing `!`)
```

Another case where this error is emitted is when a value is expected, but
something else is found:

```
pub mod a {
    pub const I: i32 = 1;
}

fn h1() -> i32 {
    a.I
    //~^ ERROR expected value, found module `a`
    // did you mean `a::I`?
}
```
```

Cargo, Rust’s package manager

Never worry about compiling libraries from source again -- cargo does everything for you!

```
[package]
name = "rustBCA"
version = "0.1.0"
authors = ["Jon Drobny <drobny2@illinois.edu>"]
edition = "2018"

[dependencies]
rand = "0.7"
geo = "0.12.2"
serde = { version = "1", features = ["derive"] }
toml = "0.5"

[profile.release]
lto = "fat"
codegen-units = 1
target-cpu = "native"
```

What does Rust look like?

Operator chains, I/O

```
//Open output files for streaming output
let mut reflected_file = OpenOptions::new()
    .write(true).
    create(true).
    open(format!("{}{}", options.name, "reflected.output")).
    unwrap();
let mut reflected_file_stream = BufWriter::with_capacity(options.stream_size, reflected_file);
```

Match statements, panic, macros, wildcard

```
//Determine the length, energy, and mass units for particle input
let length_unit: f64 = match particle_parameters.length_unit.as_str() {
    "MICRON" => MICRON,
    "CM" => CM,
    "ANGSTROM" => ANGSTROM,
    "NM" => NM,
    "M" => 1.,
    _ => panic!("Incorrect unit {} in input file. Choose one of: MICRON, CM, ANGSTROM, NM, M",
        particle_parameters.length_unit.as_str())
};
```

What does Rust look like?

OOP, adding methods to structs

```rust
pub struct Vector {
    x: f64,
    y: f64,
    z: f64,
}
impl Vector {
    pub fn new(x: f64, y: f64, z: f64) -> Vector {
        Vector {
            x: x,
            y: y,
            z: z
        }
    }
    fn magnitude(&self) -> f64 {
        return (self.x*self.x + self.y*self.y + self.z*self.z).sqrt();
    }
    fn assign(&mut self, other: &Vector) {
        self.x = other.x;
        self.y = other.y;
        self.z = other.z;
    }
    fn dot(&self, other: &Vector) -> f64 {
        return self.x*other.x + self.y*other.y + self.z*other.z;
    }

    fn normalize(&mut self) {
        let magnitude = self.magnitude();
        self.x /= magnitude;
        self.y /= magnitude;
        self.z /= magnitude;
    }
}
```

Downsides to Rust

“Fighting the borrow-checker”

Certain language features not completely mature (e.g., I/O)

Fewer and less developed libraries and packages available

Learning Rust can be more difficult than other languages - fewer tutorials for stable language versions, lack of settled best practices, especially in a scientific software context

Slow adoption introduces obstacles - rust toolchain not available on many HPC systems

Rewriting Legacy Software

When do you rewrite legacy software? When the cost of maintaining and/ior using the software exceeds the cost of rewriting it

Goals:

1. No new bugs
2. Discover and remove old bugs
3. Make it possible to maintain

Rust’s language features and code shepherding make 1. and 2. easier, especially compared to C++

Point 3 simply requires modern code development practices - easier with a modern language