Slides to follow

https://yakshav.es/oop-2019.pdf

Whoami

- https://twitter.com/argorak
- https://github.com/skade
- CEO https://asquera.de, https://ferrous-systems.com
- Rust Programmer and Trainer
- Mozillian

Whoami

- Started learning Rust in 2013
- Started the Berlin usergroup back then
- Project member since 2015
- Previously 10 years of Ruby community work
- https://twitter.com/argorak

Whoami

- Leadership events team
- Organizer of RustFest
- Part of the global Rust community team
- Project member since 2015

I’m in management of a programming language.

Whatever that means.

Three (and a half years) of Rust

Making a Programming Language Successful

Florian Gilcher

OOP 2019

CEO and Rust-Trainer
Ferrous Systems GmbH

The Language

What is it?

- new(ish) systems programming language
- powers and was developed in along with Servo, a new browser engine
- Providing an alternative to C/C++, but also higher-level languages

History - where does Rust come from?

- Started by Graydon Hoare in 2008
- An attempt to bring some research literature into production language
- Adopted by Mozilla Research 2010
- Graydon left the project in 2013
- Since then without nameable leadership

What are the goals of Rust?

- Safety:
  - No illegal memory accesses
  - Safe Collections-APIs as default
  - No Garbage Collector!
- Concurrent
  - The type system has a notion of concurrency
  - No unsafe concurrent accesses over concurrent boundaries
- Fast:
  - Speed comparable to C
  - Safety checks are a compile time concern

What are the goals of Rust?

- Safety:
  - No illegal memory accesses
  - Safe Collections-APIs as default
  - No Garbage Collector!
- Concurrent
  - The type system has a notion of concurrency
  - No unsafe concurrent accesses over concurrent boundaries
- Fast:
  - Speed comparable to C
  - Safety checks are a compile time concern

What are the goals of Rust?

- Safety:
  - No illegal memory accesses
  - Safe Collections-APIs as default
  - No Garbage Collector!
- Concurrent
  - The type system has a notion of concurrency
  - No unsafe concurrent accesses over concurrent boundaries
- Fast:
  - Speed comparable to C
  - Safety checks are a compile time concern

What are the goals of Rust?

- Stable, large code bases:
  - Communicates a lot of context locally
  - Sometimes a bit verbose
  - Good and detailled error reporting
- At full control
  - Rust allows control of memory layout
  - Works with values and references

What are the goals of Rust?

- Stable, large code bases:
  - Communicates a lot of context locally
  - Sometimes a bit verbose
  - Good and detailled error reporting
- At full control
  - Rust allows control of memory layout
  - Works with values and references

What are the goals of Rust?

- Stable, large code bases:
  - Communicates a lot of context locally
  - Sometimes a bit verbose
  - Good and detailled error reporting
- At full control
  - Rust allows control of memory layout
  - Works with values and references

It’s not ”just” a C Replacement!

What does Rust look like?

```
fn main() {
    println!("Hello, OOP!");
}
```

What does Rust look like?

```rust
use std::fs::File;
use std::io::Read;

fn main() -> Result<(), std::io::Error> {
    let mut contents = String::new();
    let mut file = File::open("hello_world.rs")?;

    file.read_to_string(&mut contents)?;

    println!("{}", contents);
    Ok(())
}
```

Rust has only a few basic concepts. These are new, though and will need getting used to.

Programming languages nowadays often differ in their semantics around mutability: where does Rust land?

Mutability

```
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let point = Point { x: 1, y: 2 };
    point.x = 2;
}
```

Mutability

```
error[E0594]: cannot assign to field `point.x` of immutable binding
 --> mutability_broken.rs:8:5
  |
7 |     let point = Point { x: 1, y: 2 };
  |         ----- help: make this binding mutable: `mut point`
8 |     point.x = 2;
  |     ^^^^^^^^^^ cannot mutably borrow field of immutable
binding

error: aborting due to previous error

For more information about this error, try `rustc --explain E0594`.
```

Mutability

```
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let mut point = Point { x: 1, y: 2 };
    point.x = 2;
}
```

- Mutability is a property of the variable, not the data!
- The mutability marker is used later on.

Ownership

All data in Rust has exactly one owner.

- Owned data can always be mutated
- Owned data is always exclusively owned
- The owner is responsible for removing the data from memory
- This happens at the end of a scope

Ownership

```
fn main() -> Result<(), std::io::Error> {
    let f = File::open("hello.txt")?;

    // The File will be closed here
    // and removed from memory
}
```

Ownership

Ownership can be passed on:

```
fn main() -> Result<(), std::io::Error> {
    let f = File::open("hello.txt")?;

    write_to_file_and_close(f);
}

fn write_to_file_and_close(mut f: File) {
    write!(f, "Hello!");

    // The File will be closed here
    // and removed from memory
}
```

Automatic memory reclamation without runtime component!

Ownership

```
fn main() -> Result<(), std::io::Error> {
    let mut f = File::open("hello.txt")?;

    write_to_file_and_close(f);

    write!(f, "Hello again!") ❶
}

fn write_to_file_and_close(mut f: File)
    -> Result<(), std::io::Error> {
    write!(f, "Hello!")

    // The File will be closed here
    // and removed from memory
}
```

❶ is illegal!

This effectively avoids reuse of deleted data.

Ownership passing can become unwieldy. But what you own, you can borrow.

Borrowing

```
fn main() {
    let mut f = File::open("hello.txt");
    print_len(&f);

    write!(f, "Done!");

    // The File will be closed here
    // and removed from memory
}

fn print_len(f: &File) {
    println!("file length: {}", f.metadata()
                                  .unwrap().length());
}
```

Lingo: & is a reference, references borrow

References never dangle.

Simple borrows cannot be used to mutate the data

Borrowing

```
fn main() {
    let mut f = File::open("hello.txt");
    print_len(&f);
    write_to_file(&mut f)
    print_len(&f);

    write!(f, "Done!");

    // The File will be closed here
    // and removed from memory
}

fn write_to_file(f: &mut File) {
    write!(f, "Written from write_to_file!");
}
```

Lingo: &mut is a mutable reference

Borrowing - Rules

The exact rules are:

- Simple (immutable) borrows are allowed multiple times.
- Mutable borrowing is only allowed once.
- Both rules are exclusive.

Borrowing - Regeln

Rust effectively guarantees that mutation of memory cannot be witnessed by another program part.

Data races when reading/writing to memory are not possible.

Example

```
fn main() {
    let mut vector = vec![1,2,3];❶
    let element = &vector[2]; ❷
    let last = vector.pop(); ❸
    println!("{}", element);
    // what's `element`?
}
```

❶ Initialisation of a new vector

❷ Accessing the third element

❸ Shortening the vector

Example

```
error[E0502]: cannot borrow `vector` as mutable because it is
also borrowed as immutable
 --> failing_borrow.rs:4:16
  |
3 |     let element = &vector[2];
  |                   ------ immutable borrow occurs here
4 |     let last = vector.pop();
  |                ^^^^^^ mutable borrow occurs here
5 | }
  | - immutable borrow ends here

error: aborting due to previous error
```

Shared mutable data errors are possible in single-threaded programs!

Further properties

- Rules are enforced in concurrent contexts (+ synchronisation)
- Rust provides generics like C++ and Java
- Has an unsafe sublanguage, e.g. for calling into C
- Zero-Cost abstractions
- Language by itself needs no memory allocation

2015: Finally Done

Rust 2010-2015

Rust was a very different language when it was announced in 2010, very much similar to Go!

The name stems from a kind of resilient fungus and means ”Rest” in Afrikaans.

For a early history lesson: Steve Klabnik - The History of Rust

Before Release

Removal of a lot of features:

- All active runtime
- stdlib trimming
- Garbage collector removal
- Removal of some error handling mechanics

Marijn Haverbeke - The Rust That Could Have Been

Before Release

Coming up with ergonomics and tooling around everything.

RFC process

Rust has an RFC process under rust-lang/rfcs on GitHub.

All language changes have to go through that.

May 15th

The Package

rustc The compiler, based on the LLVM.

cargo A build and dependency management tool, along with a registry.

rustup A toolchain manager, covering the ability to run things side by side.

rustdoc A documentation tool, complete with API search and doc testing.

A book The Rust programming language, first edition.

The whole toolchain is cross-compile aware. This is hard to retrofit.

Compatibility-tooling

- rust-gdb
- vim plugin
- VSCode Plugin

Release cycle

- Nightly releases
- All 6 weeks, nightly is promoted to beta
- After 6 weeks, beta is promoted to stable

Stability

Rust is backwards-compatible, the stable compiler only exposes the stable interfaces an features. CLI interfaces are considered interface.

On nightly, these features must be activated through opt-in.

Project management

- Anti-Benevolent Dictator For Life
- Core is a tie-breaker, not much more
- Most contributions are from non-Mozillians (75%)
- Rust had close to 1000 contributors in 2015

Project management: Infrastructure

- Rust runs on stable master
- We run CI 24/7
- There's a merge queue managed by BORS
- PRs are not allowed to be unassigned
- We use a lot of bots for support

Mozilla pays a lot of the bills.

Teams

Rust grows teams on need.

Initial Teams

Core, Infra, Language, Library, Compiler, Documentation, Community, Moderation

Initial Teams

This raises awareness of peoples work and makes them addressable.

Low-friction community interaction

- A meetup calendar
- This week in Rust Newsletter
- This week in x newsletters
- community@rust-lang.org
- Forums/IRC channels

Low-friction contributions

People want to contribute! But most projects have no tasks for people that just have 5 minutes at a time!

Low-friction contributions

Avoid lingering tasks.

We track time-to-merge very closely.

Conversational tone

Friendly and focused, not very bikesheddy, non-insulting.

Conversational tone

Stating issues is great, just be realistic about potential of quick fixes.

Conversational tone

Being grumpy or in general stern is completely fine!

Usability

Rust was a usable contenter in the systems programming space back then, but a lot of things were missing, notably: libraries.

Usability

- Good tooling early helps
- Establishing a productive environment is work
- Non-programming work is a substantial amount
- Credit it!

2016: Finding adoption

Gathering Data

We ran a users survey with 3000 respondents.

- 20 percent of all users were using Rust at work
- Half of them in projects over 10k LoC

Demographics

Rustaceans are coming in equal parts from:

- Dynamic languages
- Classic systems prgoramming
- Functional programming

Demographics

Rusts community is based in Europe! Other huge groups include China, Russia and India.

Notable Mention

Our reference usergroup is Nairobi, Kenia.

Production user survey

We conducted a production users survey. Feedback was good, especially for reliability. Pain points were as expected.

Production uses

- Alternative to plain C
- Language to write shared libraries in
- For heavily concurrent code, such as servers

Projects are rarely fully Rust, but important parts are!

Often stability-critical and power-critical components.

Production user problems

- Library maturity
- Tooling maturity
- Some missing features

People having problems with what’s not there is better then problems with things that are there!

Especially businesses are very aware of context!

Production user liaison

Production users get support on compiler-related things. You can approach us in trust.

Users

- Mozilla
- Dropbox
- Chucklefish
- Chef
- NPM
- Schauspiel Dortmund

and over 100 other companies.

How do we compare to others?

We don’t know. Programming language adoption is terribly documented.

Integrating people

- Our default meetup type is ”Hack and Learn”
- Quick review times for patches
- Helping out where we can

Conferences

- RustFest (Europe, every half year)
- RustConf (US, every year)
- Rust Belt Rust (US, every year)

Community-run, community sponsored.

Support people online

youtube: /c/rustvideos

Twitter: twitter.com/rustlang

- We tried running a global Hackfest
- The Rust community loves to write

Conclusions

- Social media isn’t a bad word
- Actively speaking ans working for feedback is better then waiting
- Your bugtracker is only half of the story
- A solid base is better then a lot of half-solutions

2017: Growing pains

Survey

5500 Respondents.

Stability

- Over 90 percent never had upgrade problems
- Almost everyone uses our tooling
- People rate the tooling favourably (4 or 5 out of 5)

Community

- 99 percent of people have no issue with the community
- 75 percent see the community favourably

Are those numbers good? We don’t know. There’s no comparisons.

Language progess

Some important language features keeping people on nightly are finally stabilised.

Especially Rust 1.15, which brings methods of pluggable code generation needed for convenient serialization.

We don’t know Rust

- Producing a language doesn’t mean you won’t discover things yourself.
- New API practices are beginning to form.
- How do include Rust into other projects becomes a main question!

New patterns coming up

Ownership is useful beyond pure memory safety:

- Used to model statemachines
- Used to model unique device driver memory access
- Extremely useful in concurrency
- ”Who’s owning that?” is a common programming problem

Realisation

Ownership is a vastly more important feature then borrowing in Rust.

The book

The book needed to be rewritten. Welcome second edition!

The book has kids

Having ”a book” as a core value makes people write books:

- Rustonomicon
- The little book of macros
- The FFI book

Directional questions coming up

Which areas should we focus on? Are we even solving useful problems?

Hobbyist problems aren't production user problems!

Release cycle review

The release cycle worked well. We only botched one release (1.14), hit no larger issues. It's a benefit to users, changes are consumable.

Release cycle review

Also, patch releases get more frequent, because of better tooling.

Rust is becoming a practices leader

- Other languages adopt the community interaction model (NodeJS)
- Other products and languages adopt our RFC process (Ember, Swift)
- Our management model is taken interest in

RFC process problems

Volume brings issues

- RFC Process can be swamped
- People with no time can’t follow along and feel excluded
- Non-english speakers have issues to follow

Usage in Firefox

Rust finally got adopted into Firefox (Quantum) on large scale, the CSS styling engine is now powered by Rust!

Conclusions

- Have a sense of accomplishment sometimes!
- Programming languages are extremely complex, even to the producers
- Identify issues early and move to fix them

2018: More structure!

Rust2018 campaign

A blogging campaign over January.

People were encouraged write down their wishes for next year.

Results

100 blog posts or essays.

readrust.net

StackOverflow most loved language

Third time in a row. Why?

We generate and manage Buzz

Buzz is a great thing if you can back it with actual technology.

Rust has a focus on things people value:

- Attention to detail
- Stability with progress
- Not reinventing the wheel
- Being around and approachable

Community team split

- RustBridge
- Events
- Content
- Switchboard (moved back in)

Forming a Roadmap

- Stable story for bare metal embedded
- Promoting our rock-solid networking story
- WASM
- Shipping language ergonomics and fixes (”Rust 2018”)
- async/await-Syntax for concurrency
- stable concurrency abstraction interfaces in the language (futures)

Introducing Working Groups

Working Groups are less global then teams. They are meant for being potentially temporary and cover aspects of the language.

- Embeeded Working Group
- Networking Working Group
- WASM Working Group
- CLI Working Group

Rust edition 2018

The compiler now knows 2 language profiles: 2015 and 2018.
Some 2015 features are syntactically illegal in 2018.

- rustc is still version 1.x and understands both
- Mixing 2015 libraries in 2018 libraries works A+
- libstd works on both
- 2015 is a committed backwards compatibility feature
- A new book!

Released on December 6, 2018.

Has that worked?

- Stable story for bare metal embedded! (Reached in August, with book!)
- Stable story for WASM (With book!)
- CLI WG produced a lot of tools and libraries to make writing CLI tools a breeze! (with small book)
- Language ergonomics and fixes are in 2018, especially more relaxed safety checking
- async-await syntax and interfaces needed to be pushed back (along with the book!)

Happy events

We had a couple of high-profile adoptions this year.

- Amazon Firecracker now powers lambda and is written in Rust
- Some game studios started working in Rust
- Some large companies now employ people to specifically work with us

We managed to go from 0 to being adoped by tons of large companies at once.

Project size

Rust now has over 100 team members and over 4800 contributors on core projects alone.

As a comparison: Go has 2400, with 3 years more lead time.

Project size: phew

Rust currently has a moratorium on forming new teams and groups. Our process of forming them needs to be structured.

We want to resolve that soon.

Surprising developments

Public perception of our adoption and feasibility for workloads is confusing.

- People misjudge our project size and speed to deliver.
- Embedded Rust is a surprisingly viable commercial market already.

Onwards

This years blogging compaign is in the processes of being evaluated. Ongoing themes:

- Stability and reduction of pace
- Sustainability
- Wishes for a governance reform
- Filling of documentation gaps

Stress release

Hitting the big ”Rust 2018” mark has also served as a stress release.

- Some people are now leaving
- A phase of refocus begins
- Some tensions show again

Conclusion

- Rework your organisation regularly
- Change your approaches along with it!
- Large goals are important, be ready for reaching them
- See what it unlocks!

Conclusion

Large, quality projects without a benevolent dictator are there and work. They just don’t make much noise!

Conclusion

The LLVM project, the Gnome Foundation, the Ruby, Python, Mozilla and JavaScript communities. We apply your work.

Thank you!