Rust: reliability, productivity, performance -- pick three

Dirkjan Ochtman
dirkjan.ochtman@ing.com
April 2019

Why talk about Rust?

- Building reliable performant software faster
- Used in Hyperledger Indy and Sawtooth
- Ethereum implementation in Rust
- Other blockchain frameworks like Exonum
- Can be used in the browser through WebAssembly

Agenda

- History
- Reliability
- Productivity
- Performance
- Social proof
- Challenges

History

- Initially funded by Mozilla (makers of Firefox) Research
  - Come up with a new language for performance and security (to build a better browser with)
- Worked in tandem with the Servo project
  - A new browser engine that makes better use of multi-core computers and the GPU
  - Parts of Servo have been incorporated into Firefox

Reliability

- Option type: no billion-dollar null mistake
- Result type: explicit ergonomic error handling
- Memory safety: no buffer overflows or use-after-free
- Privacy in modules helps build abstractions

Combinators

Option<T>

```
// To inner type
⇒ unwrap () → T
⇒ unwrap_or (T) → T
⇒ unwrap_or_else (() → T) → T
⇒ unwrap_or_default () → T where T: Default
⇒ expect (&str) → T

// Converting to another type
⇒ map ((T) → U) → Option<U>
⇒ map_or (U, (T) → U) → U
⇒ map_or_else (() → U, (T) → U) → U

// To Result
⇒ ok_or (E) → Result<T, E>
⇒ ok_or_else (() → E) → Result<T, E>

// Conditioning
⇒ filter ((&T) → bool) → Option<T>
⇒ and (Option<U>) → Option<U>
⇒ and_then ((T) → Option<U>) → Option<U>
⇒ or (Option<T>) → Option<T>
⇒ or_else (() → Option<T>) → Option<T>
```

Result<T, E>

```
// To inner type
⇒ unwrap () → T where E: Debug
⇒ unwrap_err () → E where T: Debug
⇒ unwrap_or (T) → T
⇒ unwrap_or_else ((E) → T) → T
⇒ unwrap_or_default () → T where T: Default
⇒ expect (&str) → T
⇒ expect_err (&str) → E
⇒ ok () → Option<T>
⇒ err () → Option<E>

// Mapping
⇒ map ((T) → U) → Result<U, E>
⇒ map_err ((E) → F) → Result<T, F>

// Conditioning
⇒ and (Result<U, E>) → Result<U, E>
⇒ and_then ((T) → Result<U, E>) → Result<U, E>
⇒ or (Result<T, F>) → Result<T, F>
⇒ or_else ((E) → Result<T, F>) → Result<T, F>
```

Built for concurrency

- Ownership and borrow checking (RAII)
- Immutable by default
- No data races

No traditional OOP

- Explicit self (&self, &mut self, self, mut self)
- No inheritance, structs (and enums) + methods
- Traits and impls for generics -> monomorphization
  - Dynamic dispatch needs explicit syntax

Macros

- Extensive macro system
  - Helps deduplicate boilerplate code
- Procedural macros
  - Write Rust code to generate Rust code from Rust code
  - Very powerful way to do aspect-oriented programming

Productivity: tooling

- Great error messages
- Linting tools help
  - Suggest idiomatic code style
  - Highlight potential errors

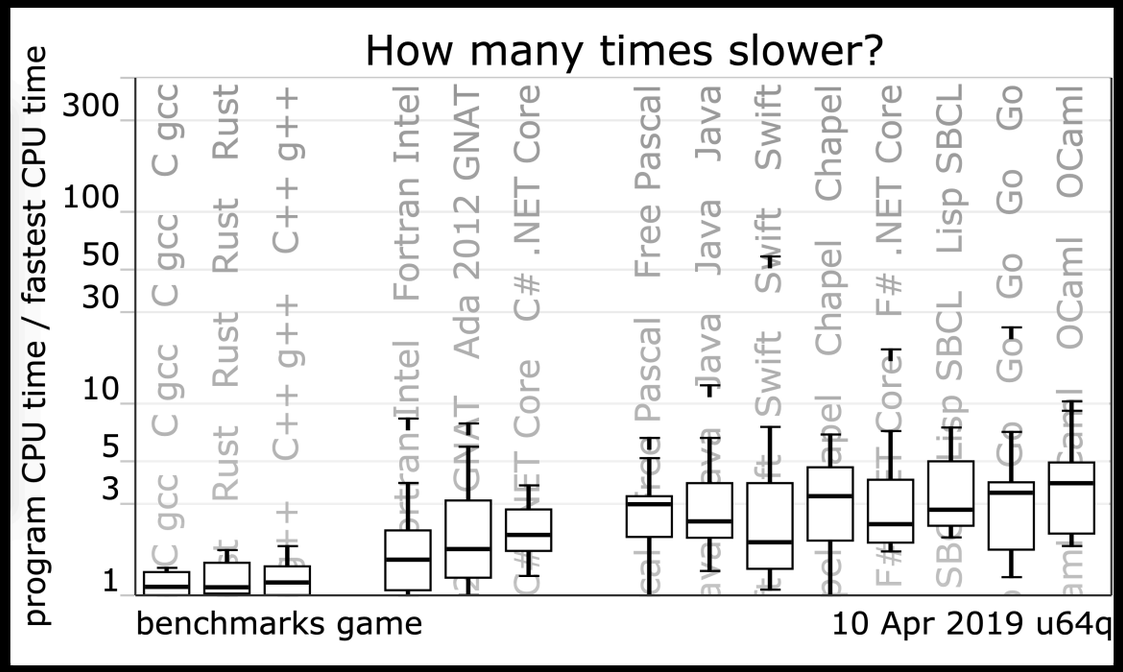
Productivity: Cargo

- Dependency management
- Build system
- Built-in support for:
  - Debug and release build profiles
  - Unit tests, integration tests and benchmarks

Productivity: platforms

- Straightforward to use in projects that can call C functions
- Viable on microprocessors (ARM Cortex-M0, RISC V)
- Broad platform support
  - Share code between Android and iOS
  - Target WebAssembly to run fast in the browser

Performance (1)



https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fast.html

Performance (2)

These results quantify the time-space tradeoff of garbage collection:

with five times as much memory, an Appel-style generational collector with a non-copying mature space matches the performance of reachability-based explicit memory management.

With only three times as much memory, the collector runs on average 17% slower than explicit memory management.

However, with only twice as much memory, garbage collection degrades performance by nearly 70%.

Hertz and Berger, OOPSLA 2005

https://people.cs.umass.edu/~emery/pubs/gcvsmalloc.pdf

Most loved language

Stack Overflow developer survey:

"For the fourth year in a row, Rust is the most loved programming language among our respondents"

(% of developers who are developing with the language or technology and have expressed interest in continuing to develop with it)

Ranked #13 in 30-day GitHub PRs

https://insights.stackoverflow.com/survey/2019/
https://twitter.com/jntrnr/status/1082024512701456384

Other Rust users

- Dropbox: components of the core file-storage system
- Google: Fuchsia, new mobile OS
- Amazon: Firecracker, lightweight virtualization solution
- Deliveroo, Zalando, Atlassian, Parity, Cloudflare, Mozilla...

https://www.rust-lang.org/production
https://firecracker-microvm.github.io/
https://www.rust-lang.org/production
https://jobs.zalando.com/tech/blog/getting-started-with-rust/
https://deliveroo.engineering/2019/02/14/moving-from-ruby-to-rust.html

npm

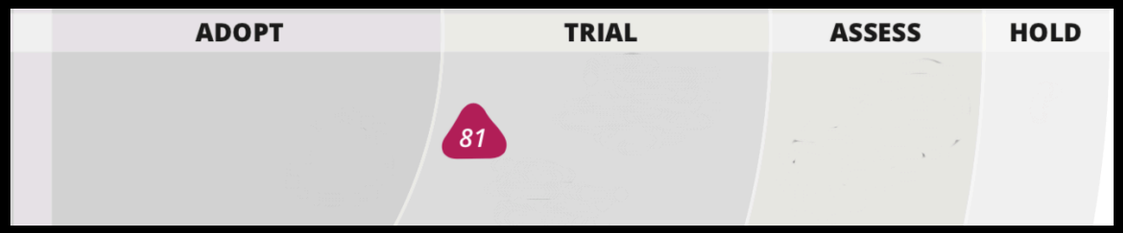
npm’s first Rust program hasn't caused any alerts in its year and a half in production. [..] The process of deploying the new Rust service was straightforward, and soon they were able to forget about the Rust service because it caused so few operational issues.

https://www.rust-lang.org/static/pdfs/Rust-npm-Whitepaper.pdf

Challenges

- Learning curve can be challenging
  - C/C++ programmers vs functional vs dynamic language
- Compile times
  - Often similar to large C++ projects
  - Code generation bottleneck, type checking much faster
- IDE support
  - IntelliJ/CLion is the best option right now
  - Not as great as Java/Scala/Kotlin, Swift or C++
- Ecosystem maturity (rapidly improving)

ThoughtWorks Tech Radar



April 2019

https://assets.thoughtworks.com/assets/technology-radar-vol-20-en.pdf