RUST INTRODUCTION

INCEPTION

- "Most loved" language in 2016, 2017 and 2018
- Initiated in the "The free lunch is over" era
- 1.0 release in 2015
- New point releases every 6 weeks

DESIGN TRIFECTA

- Safe
- Concurrent
- Fast

Pick all three!

DESIGN CHOICES

- A systems programming language
- For "programming in the large"
- Main influences from C++, Haskell and ML
- Static type system with local type inference

DESIGN CHOICES CONTINUED...

- Return values instead of exceptions
- Static automatic, deterministic memory management
- RAII - no garbage collector

SAFETY FEATURES

- No null pointers
- No dangling pointers
- No data races (partially true)
- Everything is immutable by default
- Safe by default, "unsafe" optional
- Compile-time concurrency safety (static analysis)

LANGUAGE FEATURES

ROOT INFLUENCES

- ML-inspired match system
- Extremely helpful error messages
- Haskell-inspired type system

STANDARD LIBRARY FEATURES

- One standard library implementation on all platforms
  - Actually readable/debuggable!
- One compiler implementation
- Documented in code

BUILD SYSTEM FEATURES

- crate - a build system and package manager
- crate.io - the package repository
- Unit testing integrated
  - Tests executed in parallel by default

META-PROGRAMMING FEATURES

- Generics
  - Trait-constrained, not duck-typed
- Macros
- Procedural Macros
  - Rust Macros using Rust
- Compiler Plugins
  - Also managed as crates

FEARLESS CONCURRENCY WITH RUST

Rust was initiated to solve two thorny problems:

- How do you do safe systems programming?
- How do you make concurrency painless?

Central: Ownership/"Borrowing" → compile-time, not runtime (=static analysis)

No compromises in terms of performance - zero-cost abstractions

OWNERSHIP

- Every piece of data is uniquely owned
- Ownership can be passed
  - Passed by default!
- When owned data reaches the end of a scope, it is destructed
- Determined statically at compile time - even for references

BORROWING

- Access can be borrowed
- You can borrow mutably once
- You can borrow immutably multiple times
- Exclusivity: No mutable and immutable borrowing at the same time
- Boil down to pointers at runtime

THREADS

- Use "spawn" to create new threads
- Move data structures into closures to pass ownership
- Message passing similar to Go Caution: Can deadlock when waiting on messages

STRUCTURED PARALLEL PROGRAMMING

BASIC PRINCIPLES

- Composable parallel programming patterns
- Avoid using explicit threads to avoid oversubscription
- Define *potential* concurrency as discrete tasks
- Usually implemented using a form of "work stealing"
    - Pioneered by Cilk(MIT Project), used in TBB

RAYON

- Rayon = fake (C)Silk
- Work Stealing similar to Cilk
- Idiomatic Rust with all its safety guarantees

RAYON FEATURES

- par_iter
  - almost identical to the non-parallel version
- join
  - building block of par_iter implementation
- scope
  - more flexible than join to define parallel workload spans

PERFORMANCE IN EXAMPLES

- Walk through Mandelbrot example in Rust and C++
  - Rust wins (probably due to better/newer compiler back-end vs Ubuntu 16.0.4 gcc)
- Show geth, cpp-ethereum, parity comparison
  - Rust wins (C++ client instable, slower - Go slower, more memory)