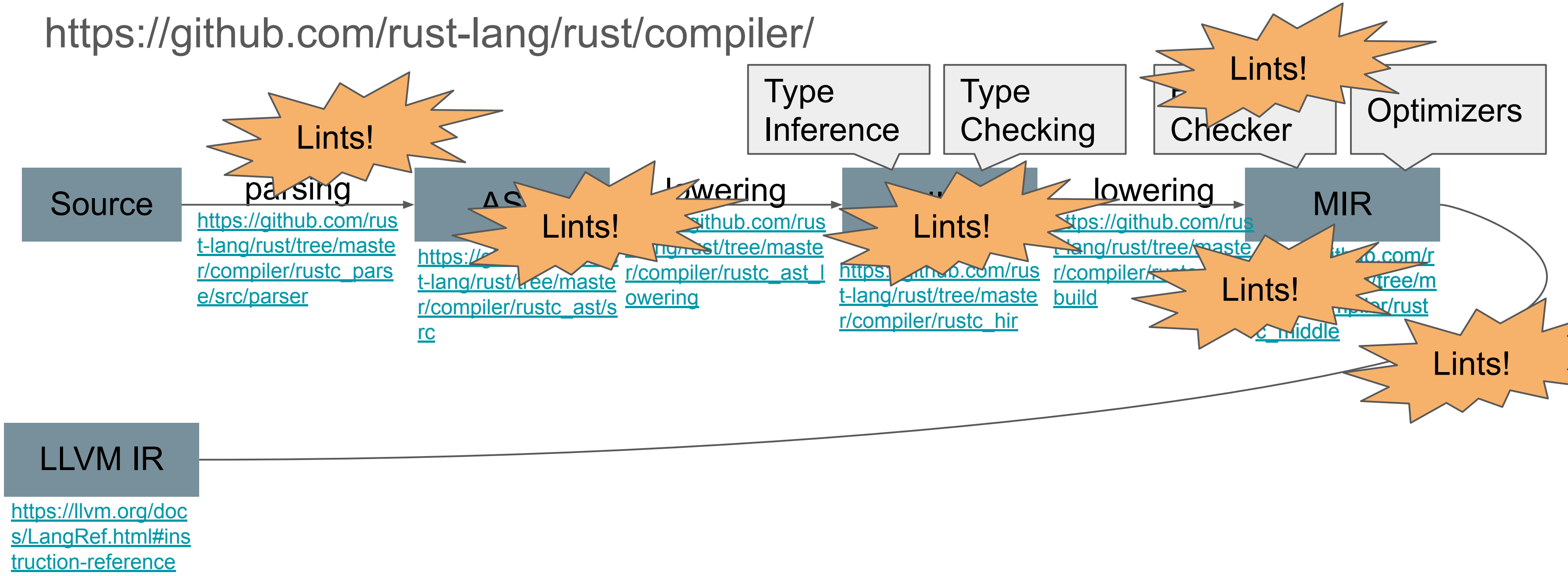
Something interesting about rustc

Polyvios Pratikakis
DevStaff, Oct 13

The Rust Compiler

- https://github.com/rust-lang/rust/compiler/



Source
parsing
https://github.com/rust-lang/rust/tree/master/compiler/rustc_parse/src/parser
AS
t-lang/rust/tree/master/compiler/rustc_ast/src
lowering
github.com/rust-lang/rust/tree/master/compiler/rustc_ast_lowering
Type Inference
Type Checking
Checker
Optimizers
https://github.com/rust-lang/rust/tree/master/compiler/rustc_hir
lowering
https://github.com/rust-lang/rust/tree/master/compiler/rustc_mir_build
MIR
c_middle
Lints!
Lints!
Lints!
Lints!
Lints!
Lints!
LLVM IR
https://llvm.org/docs/LangRef.html#instruction-reference

Compiler code base - Interesting stuff #1

- Mem Mgmt tricks (in Rust, so ok)
- Values interned in arenas (Hash-consing)
    - All values allocated in MIR are in the same arena
    - Compare on allocation
    - Pointer is the same ⇔ Value is the same
    - Arena is lifetime

Compiler code base - Interesting stuff #2

- Query system
  - Be lazy, look like an IDE (but only the good parts)
  - Faster compilation, avoid re-doing work
  - Incremental compilation (no, really)
  - “Do we have optimized MIR for this function?” “Do we have the live variables for this basic block?”
    - “Yes, here” (even for the same file, crate)
    - “No, get me HIR” (this may cascade to another query, and another)
  - Compile lazily
  - Save intermediate state for subsequent compilations
    - If you change only 1 function in a file, why recompile the rest?
- (OK, it’s a bit more complex than that.)

Compiler code base - Interesting stuff #3

- Is being parallelized
- Cargo already parallelizes building stuff, but not RustC
- Parallel LLVM code generation
- Other parts of the compiler are being parallelized
  - Not yet there
  - Need people to do the work
  - Good: In Rust, use “async”
  - Good: Borrow checker protects from races!
  - Less good: may need to rewrite lifetimes to partition arenas and then parallelize work

Compiler code base - Interesting stuff #4

- MIR is a control-flow graph (CFG)
- Includes data flow analysis engine
- Pretty easy to write a dataflow analysis
  - (Write your own lints!)
- Pretty easy to write an optimization
  - (Communication with LLVM, well, not so easy)
- Rust has polymorphic code
  - MIR does not
  - No type erasure
  - Instantiation-based monomorphization (like c++)
    - Ew

Historical stuff

- Safe memory management, but fast
  - For many years, science fiction
  - The main problem is free(), in C (from ALGOL etc.)
    - Garbage collection (1960)
- 1967 first “Arena”, 1990 in C
- 1988, first idea to get safe “arena”, for objects
- 1994, first theory for safe regions, Standard ML (inference, expensive)
- 2002, Cyclone (C-clone), safe regions (unique, borrowing, lots of annotations)
- Rust: safe region checking (ownership types, borrowed ownership, simpler)
  - Seems to be OK
  - Is there a simpler way? (Except infinite RAM)

Ref

- https://github.com/rust-lang/rust/tree/master/compiler
  https://rustc-dev-guide.rust-lang.org/
- https://llvm.org/docs/LangRef.html#instruction-reference