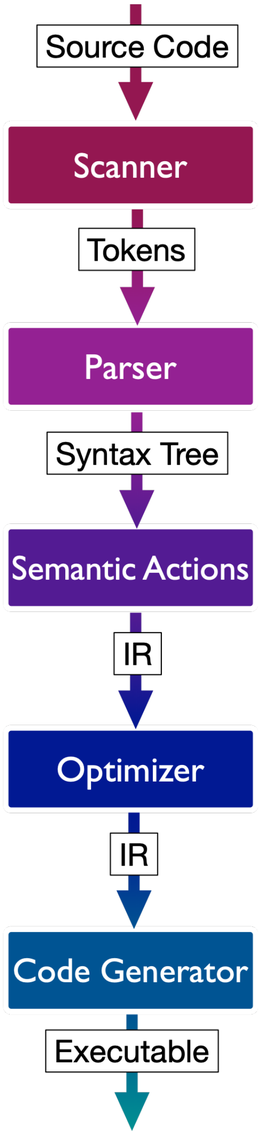
Structure of a Compiler

overall structure of a compiler



Use *regular expressions* to define tokens. Can then use scanner generators like flex or ANTLR

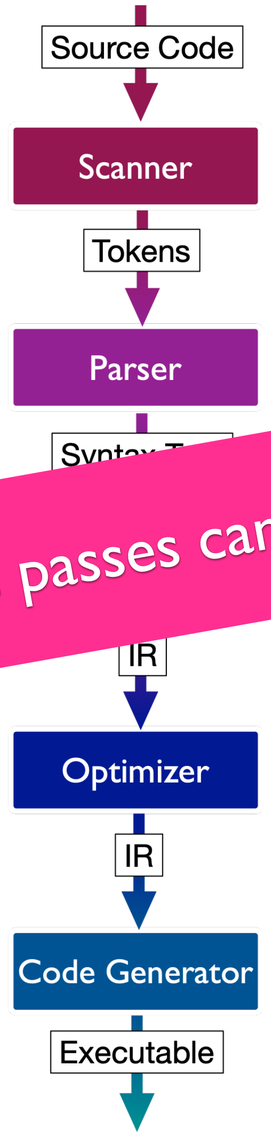
Define language using *context free grammar*. Can then use parser generators like bison or ANTLR

Typically written by hand, but can be formalized

Written manually. Optimization is an active research area

Typically written manually.

overall structure of a compiler



Many of these passes can be combined

Use *regular expressions* to define tokens. Can then use scanner generators like flex or ANTLR

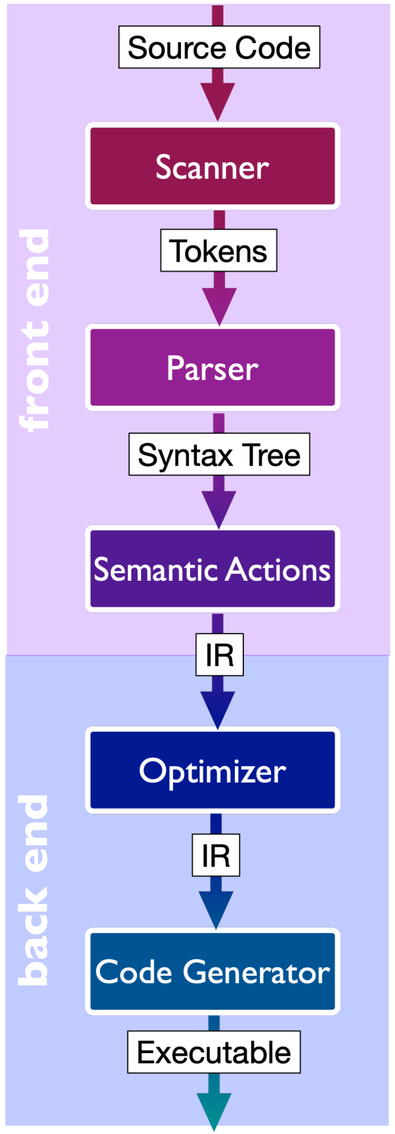
Define language using *context free grammar*. Can th rser generators like bison or ANTLR

ypically written by hand, but can be formalized

Written manually. Optimization is an active research area

Typically written manually.

front-end vs back-end



- Scanner + Parser + Semantic actions + (high level) optimizations called the front-end of a compiler
- IR-level optimizations and code generation (instruction selection, scheduling, register allocation) called the back-end of a compiler
- Can build multiple front-ends for a particular back-end
  - e.g., both Java and Scala target Java bytecode
- Can build multiple back-ends for a particular front-end
  - e.g., llvm allows targeting different architectures

next: how do scanners work?

Or: Wait, regexes are involved?