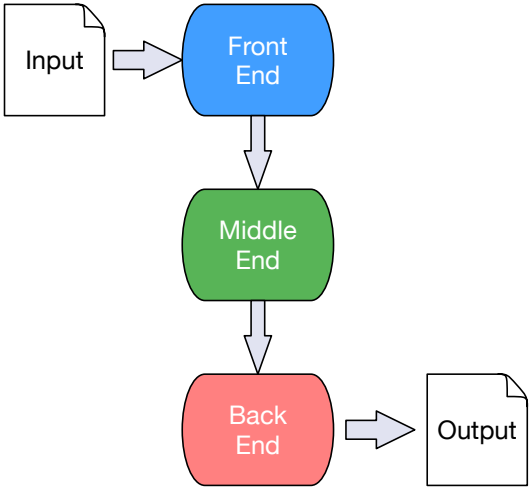
Compiler Construction

A Compiler Architecture

General layout – Compiler structure



Front end

Analysis of the input language

- Lexical analysis
- Syntactic analysis
- Static semantic analysis:
  - type checking
  - context sensitive check
  - ...
- Source language specific optimizations

Middle end

Generic synthesis and optimizations

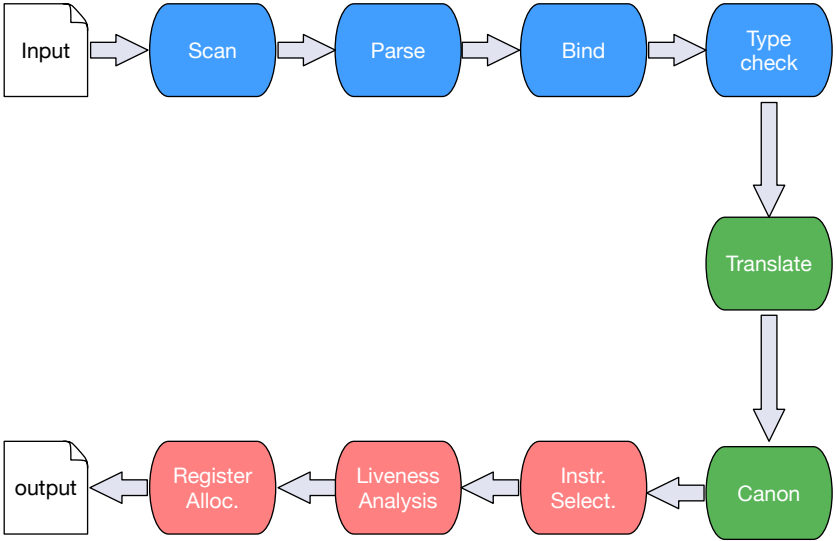
- Stepwise refinement of the intermediate representation
- Generic optimizations

Back end

Synthesis of the output

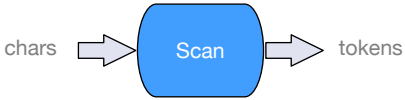
- Instruction selection
- Register allocation
- Assembly specific optimizations
  - SIMD
  - Caches
  - ...

Classical Architecture: a long pipe



Scanner

Breaks the source file into individual words called *tokens*

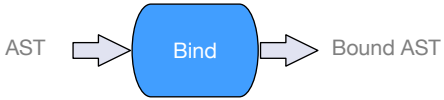


Parser

Analyze the phrase structure of the program

Binder

Relate uses of the variables to their definitions

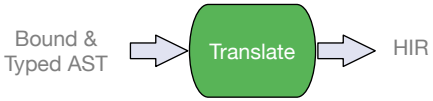


Type Checker

Check types of expressions

Translate

Place variables into activation records and generate Intermediate Representation.

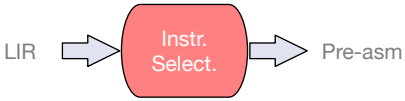


Canonicalize

Hoist side effects, cleanup for convenient of the next phases

Instruction Selection

Group IR elements into clumps that correspond to the actual target-machine instructions



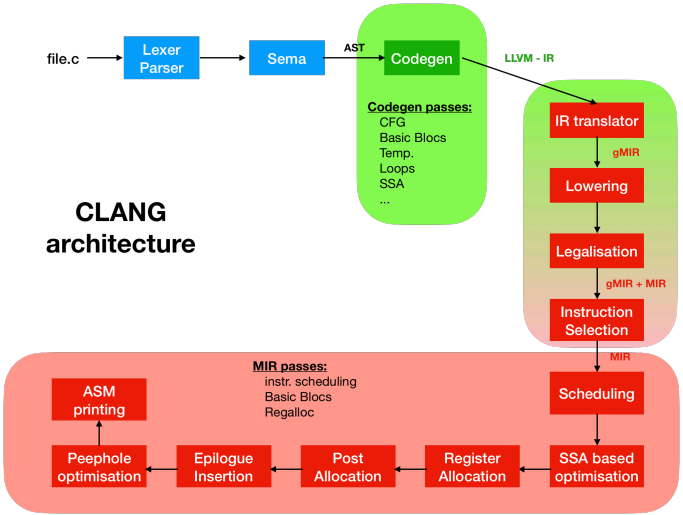
Liveness Analysis

Calculates places where each variable holds

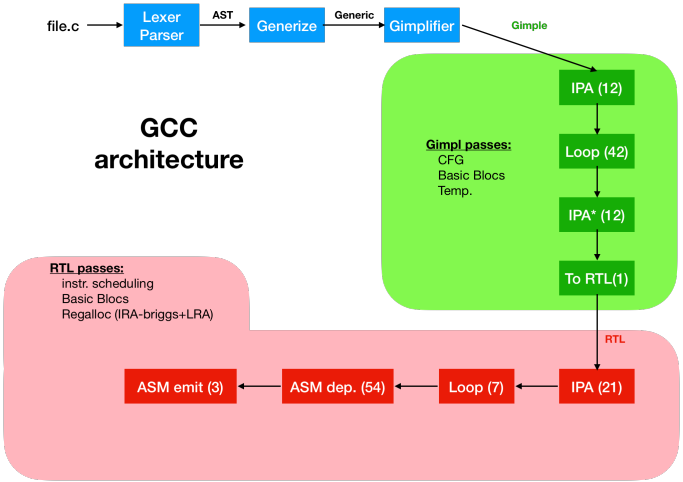
Register Allocation

Calculates places where each variable holds

Overview of Clang



Overview of GCC

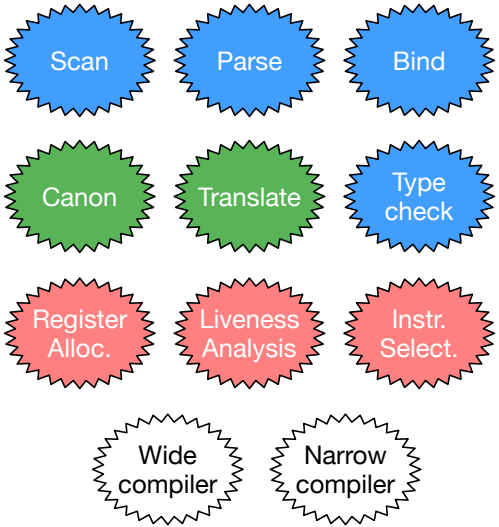


Wide compilers vs narrow compilers

- A *narrow compiler* is a compiler that only read a few chars from the input, typically one line or one function.

- A *wide compiler* is a compiler that works on the whole source file.

Summary


Scan
Parse
Bind
Canon
Translate
Type check
Register Alloc.
Liveness Analysis
Instr. Select.
Wide compiler
Narrow compiler