You came early!

We Appreciate that, Before we get started I want you to do the following

1. Get the slides from 10.1.82.20:[8000 - 8010]
2. Explore Google Summer of Code Organization page

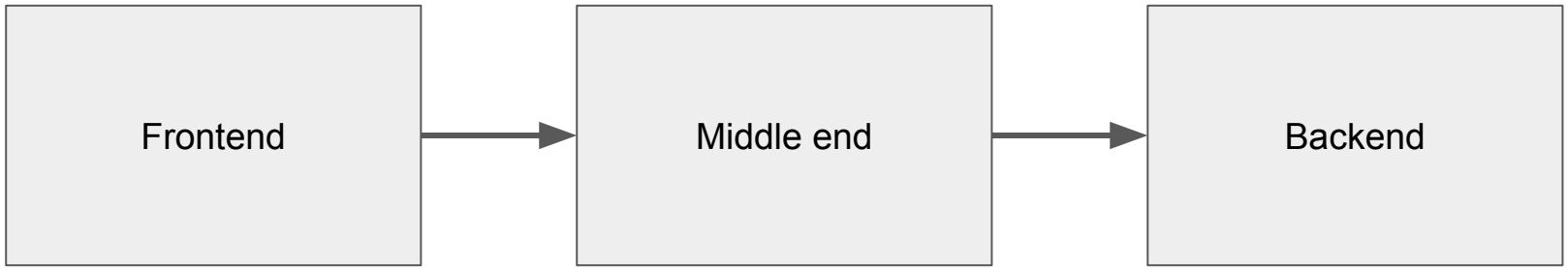
Compilers 101

What is a Compiler?

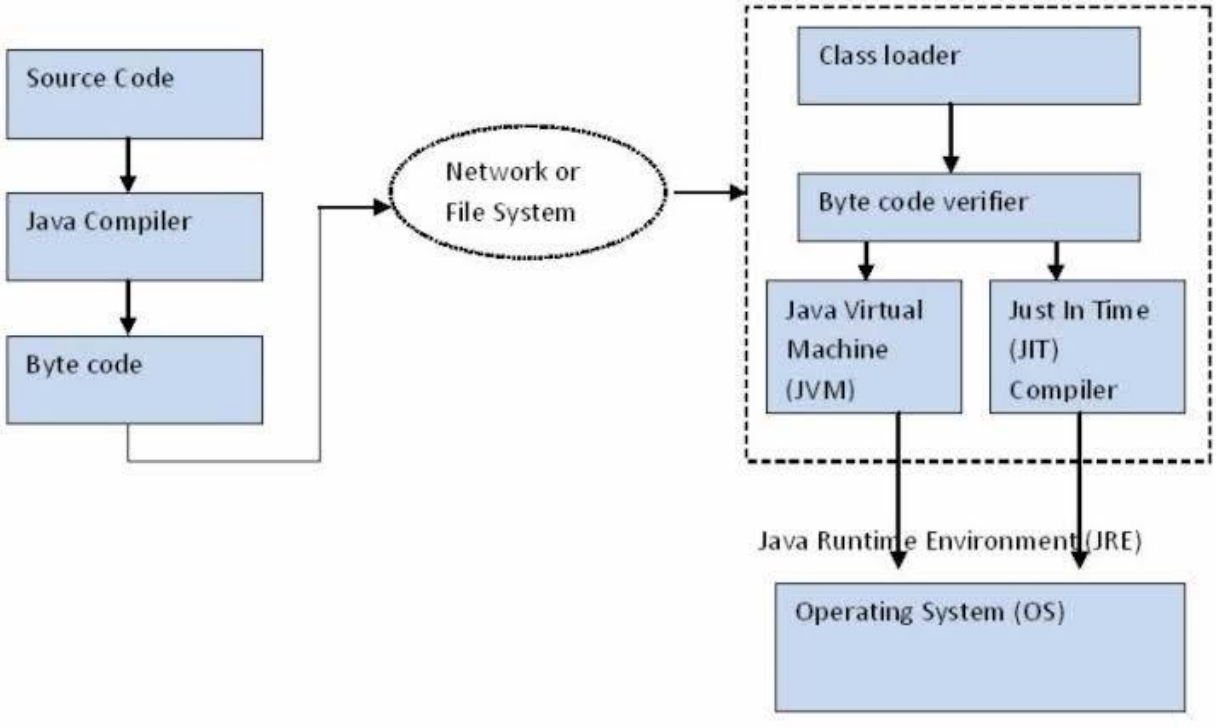
How to build a compiler for a new language?

Intuition Behind Compilers

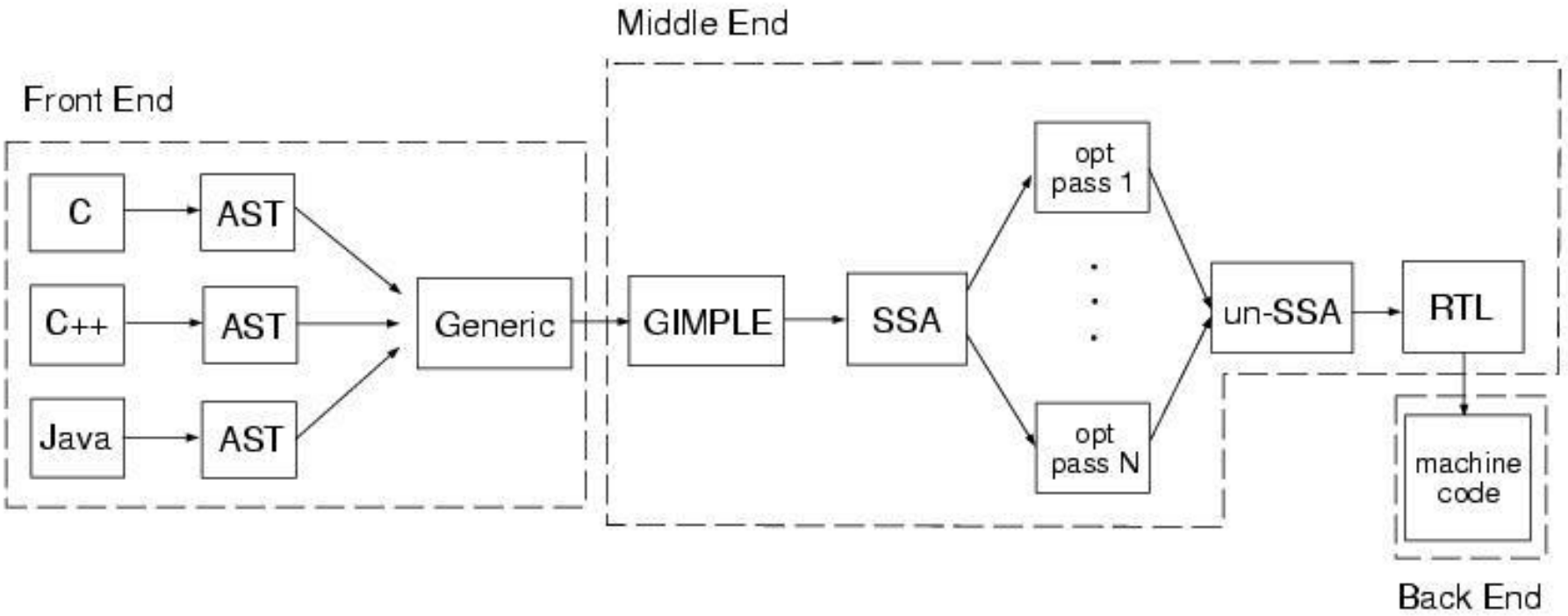
Architecture of a Compiler*



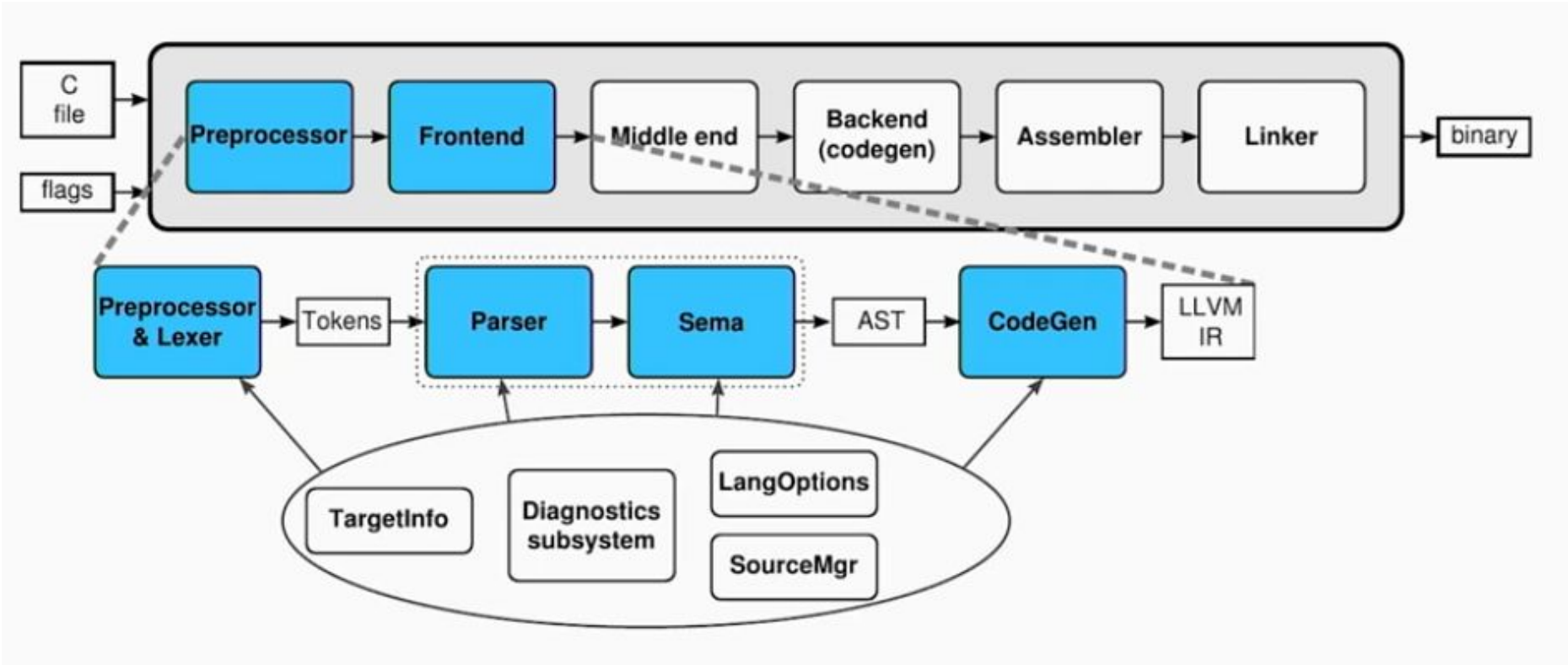
JVM Languages



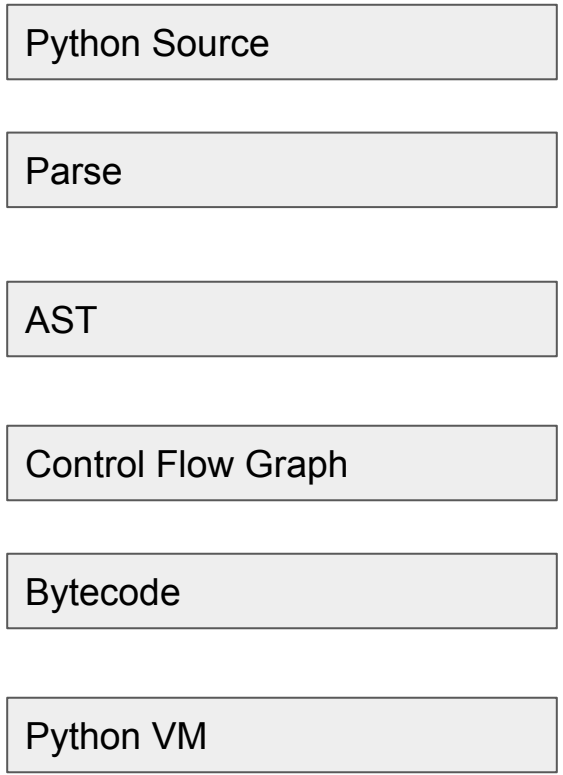
C++ Compiler Architecture (GCC)



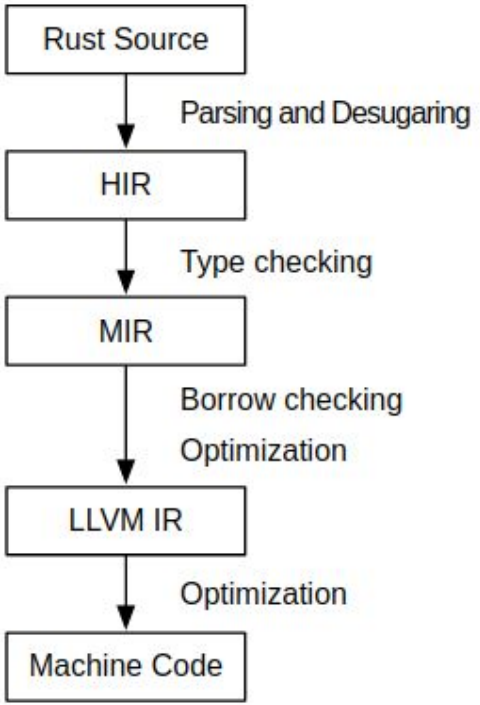
C++ Compiler Architecture (Clang)



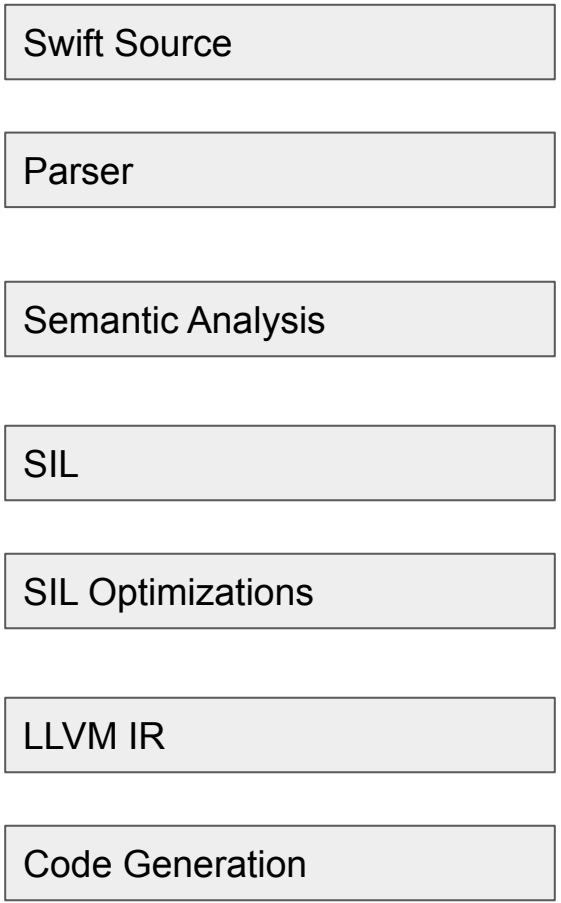
Python Compiler Architecture



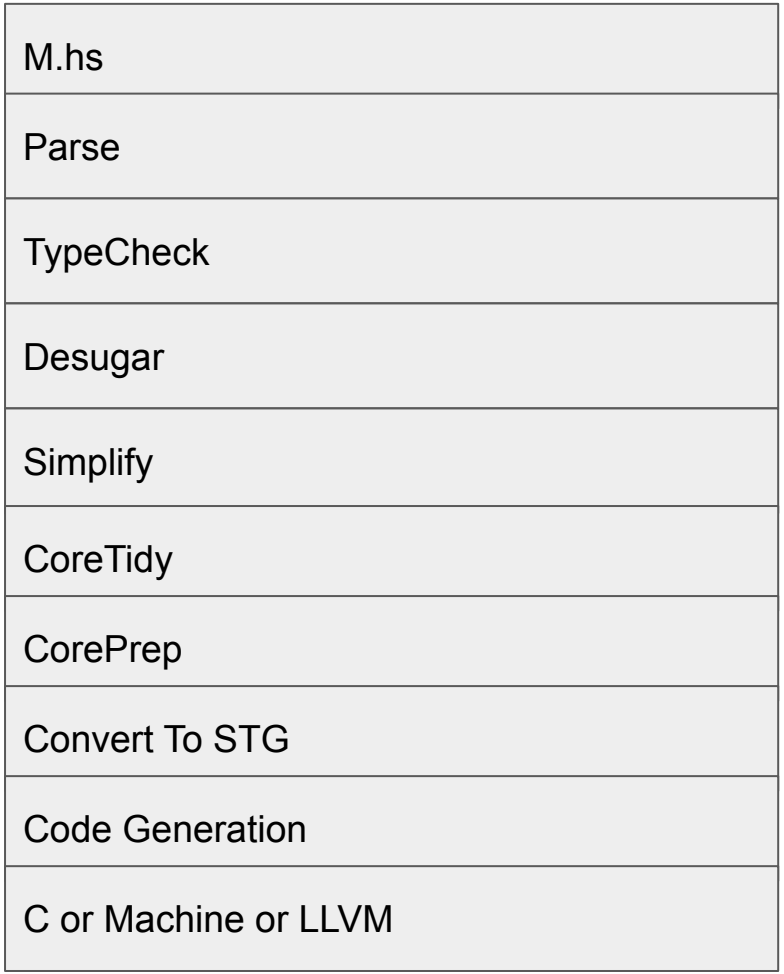
Architecture of Rust Compiler



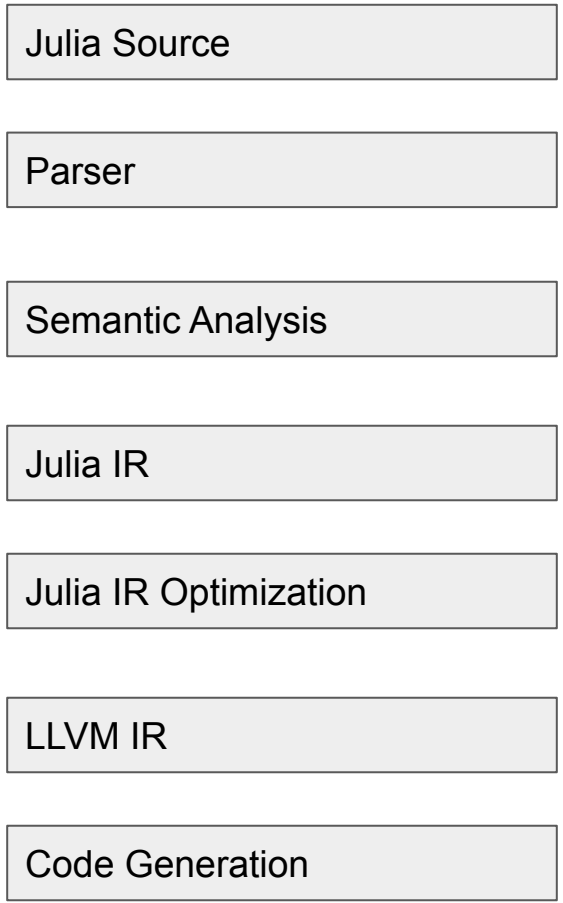
Architecture of Swift Compiler



Architecture of Haskell Compiler

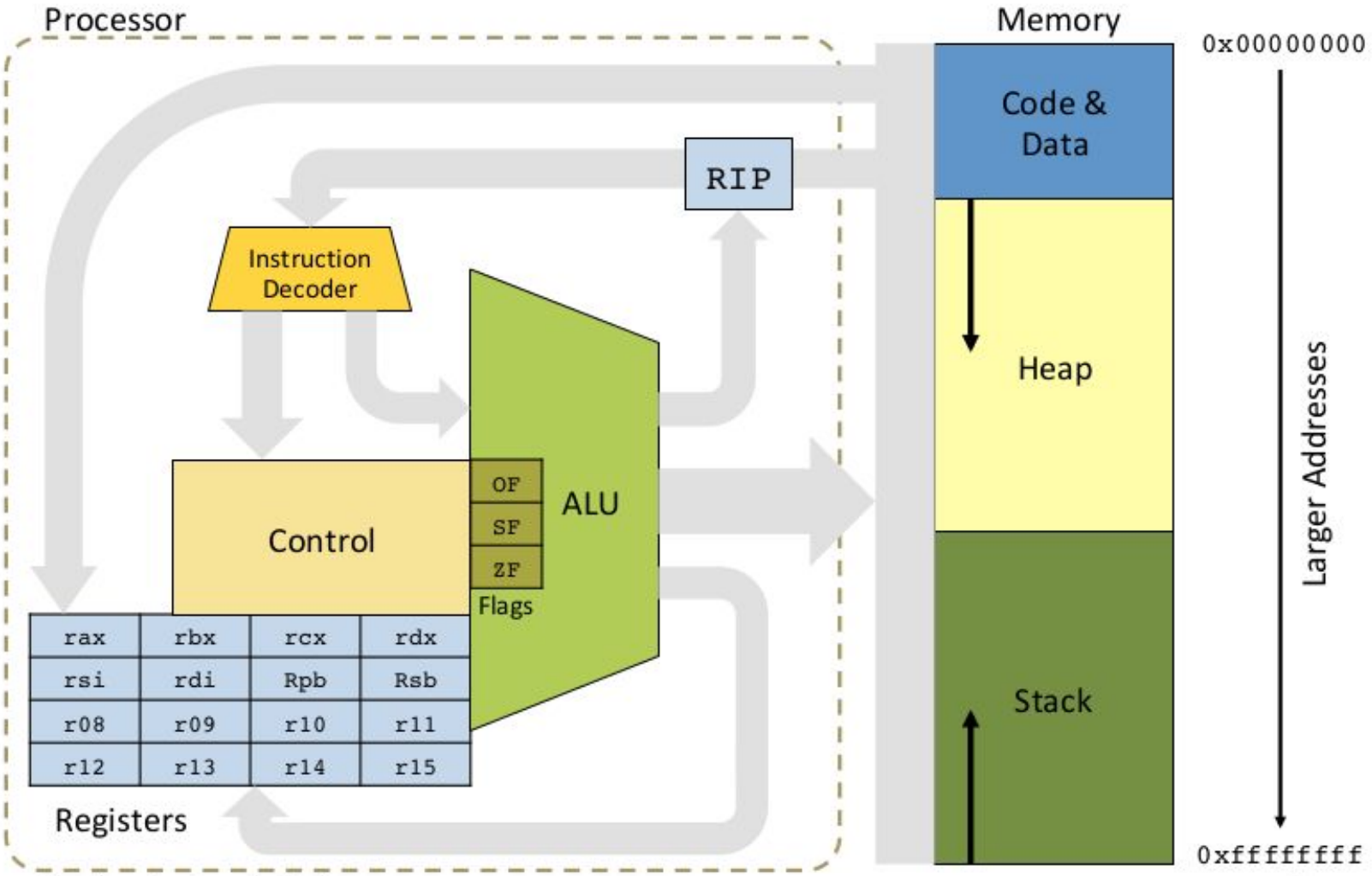


Architecture of Julia Compiler



Assumption Before We Start

Machine Model

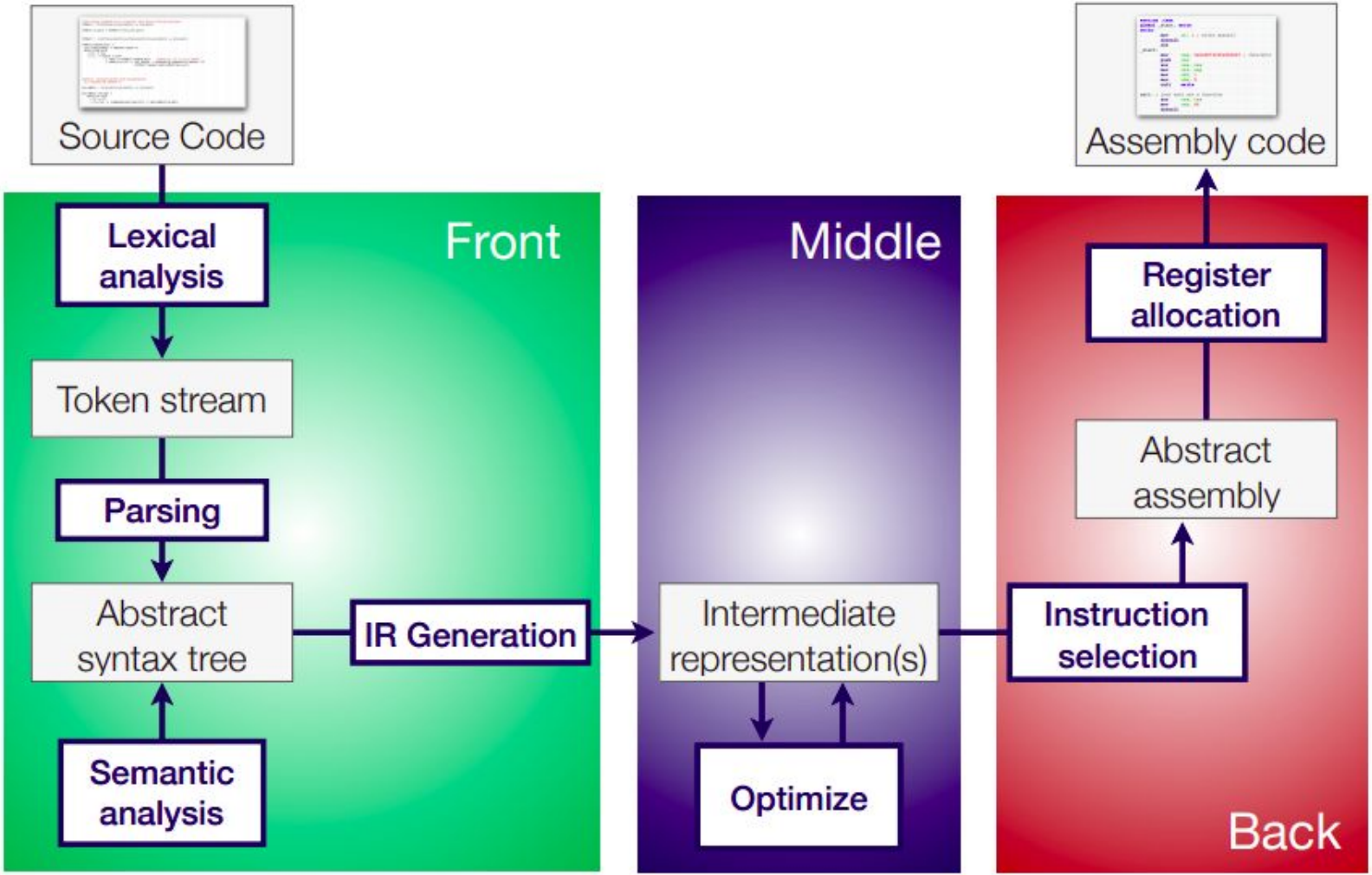

Processor
Memory
0x00000000
Code & Data
RIP
Instruction Decoder
Heap
Larger Addresses
OF
SF
ZF
ALU
Control
Flags
rax
rbx
rcx
rdx
rsi
rdi
Rpb
Rsb
r08
r09
r10
r11
r12
r13
r14
r15
Registers
Stack
0xffffffff

Compiler vs Compiler Driver

clang vs cc1

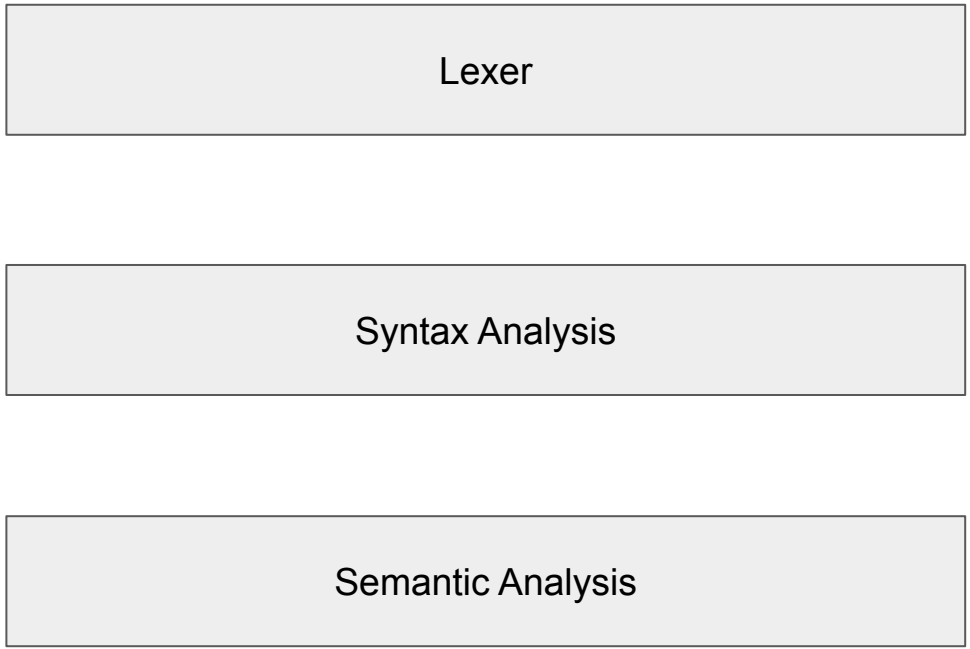
clang -ccc-print-phases <source.cpp>

clang <source.cpp> -v


Source Code
Lexical analysis
Front
Token stream
Parsing
Abstract syntax tree
IR Generation
Semantic analysis
Middle
Intermediate representation(s)
Optimize
Assembly code
Register allocation
Abstract assembly
Instruction selection
Back

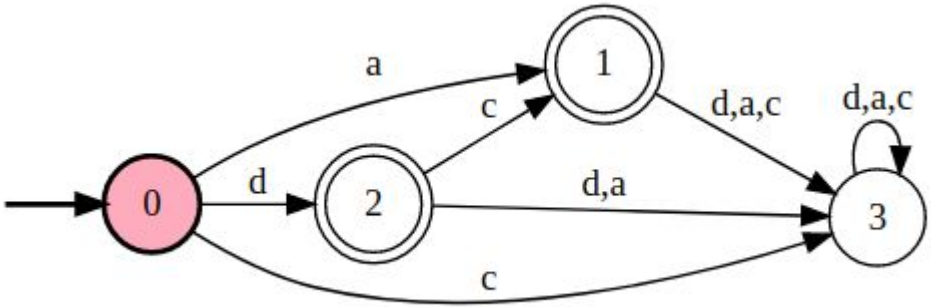
Frontend

Frontend



Lexer or Lexical Analysis

- **Lexer** reads the source code and locates all the tokens from the source
- The process is done using State Machines or Finite State Machines (FSM)
- The tokens are represented as patterns and every character is matched against a set of patterns
- Such patterns are called **Regular Expressions**
- e.g. d+a+dc



clang -c -Xclang -dump-tokens <source.cpp>

https://godbolt.org/z/SXtXW2

Before that we need to know how a language is represented

Any language has set of rules to form constructs

Grammar

Grammar

- The grammar of a programming language is represented by **Context Free Grammar**
- CFG contains set of rules on how to develop a construct
- e.g

$$S \rightarrow T \mid T + S \mid T - S$$

$$T \rightarrow \texttt{int} \mid \texttt{int} * T \mid \texttt{int} / T$$

Grammar of a Programming Language

```
<prog> ::= (<gdecl> | <gdefn>)*

<gdecl> ::= struct <vid> ;
          | <tp> <vid> ( [<tp> <vid> (, <tp> <vid>)*] ) ;

<gdefn> ::= struct <sid> { (<tp> <fid> ;)* } ;
          | <tp> <vid> ( [<tp> <vid> (, <tp> <vid>)*] ) { <body> }
          | typedef <tp> <aid> ;

<body> ::= <decl>* <stmt>*

<decl> ::= <tp> <vid> [= <exp>] ;

<stmt> ::= <simple> ;
         | if ( <exp> ) <stmt> [ else <stmt> ]
         | while ( <exp> ) <stmt>
         | for ( [simple] ; <exp> ; [simple] ) <stmt>
         | continue ;
         | break ;
         | return [<exp>] ;
         | { <body> }
```

Syntax Analysis

- Through Parsing we construct an Abstract Syntax Tree (AST)
- AST represents the syntactic structure of a programming language
- The AST is used for further optimization at the source level
  - Constexpr evaluation
  - Constant Folding
  - etc.

clang -Xclang -ast-dump -fsyntax-ony <source.cpp>

https://godbolt.org/z/pmH4ak

The Most Vexing Parse

https://godbolt.org/z/xgG-p8

Middle end

Middle End

- This is one of the interesting phases in compilation
- The AST is then lowered to a Machine and Language Independent representation
- The idea behind an Intermediate representation is to progressively lower without loosing a lot of information along the way
- The representation is updated to optimize for the target architecture

LLVM Intermediate Representation
https://godbolt.org/z/ZwkG9x

LLVM Intermediate Representation

- LLVM IR is an SSA based IR
- LLVM also provides a set of tools to work on the intermediate representation
  - Query
  - Transform
  - Convert
- LLVM IR represents a pseudo assembly with an infinite register file

What is SSA?

What is SSA?

- SSA stands for Static Single Assignment
- Each assignment to a temporary is given a unique name
- A register is assigned only once and any subsequent assignments leads to a create of new virtual register
- SSA based IRs are more common now-a-days
- GCC uses SSA based IR
- SSA simplifies a lot of optimization like
    - Value Numbering
    - Constant Propagation
    - Common Subexpression Elimination
    - Partial-Redundancy Elimination

Example for SSA

```
int a = 10;
b = a + 100
c = b + 20
a = a + 1
b = b + 1
d = a1 + 2
e = b1 + 100
```

```
int a = 10;
b = a + 100
c = b + 20
a1 = a + 1
b1 = b + 1
d = a1 + 2
e = b1 + 100
```

Example for SSA

```
int a = 10;
b = a + 100
c = b + 20
a = a + 1
b = b + 1
d = a1 + 2
e = b1 + 100
```

```
int a = 10;
b = 10 + 100
c = 110 + 20
a1 = 10 + 1
b1 = 110 + 1
d = 11 + 2
e = 110 + 100
```

But that doesn’t end there

But that doesn’t end there
LLVM provides an infra. for even more optimizations

https://godbolt.org/z/Tm73FS

Interesting things to do

- Device your own optimizations
- Derive Optimizations automatically (super optimizer)
- Polyhedral Optimization
- MLIR (Multi-Level Intermediate Representation)

List of LLVM Optimization Passes

Backend

Backend

- The Backend is one of the complex part of the compiler phases
- The phase involves
  - Removing SSA representation
  - Selecting the best instruction for the target machine (Instruction Selection)
  - Register Allocation
  - Instruction Scheduling
- The Backend also has a representation
- Finally Object code is generated depending on the platform
- We forgot the symbol table

We are not done yet

How do we get to the main

objdump -t <executable>

objdump -t <executable> | grep main

ldd <executable>

nm <executable>

How do we get to the main

- There are lot of things after compilation
  - Linking
  - Loading
  - Runtime System
  - Memory Management like Garbage Collection

Kaleidoscope

Summary

Building a Compiler is a great engineering effort and cannot be done overnight.

Ideas

Ideas

- Build a compiler for Simple language
- Write Optimization passes for a programming language
- Converting from one language to another (Transpiling)

Resources

- CMU Compiler Course
- ETH Compiler Course
- Writing LLVM Passes 101
- LLVM Dev Meeting - Tutorial section
- Using Clang frontend

Google

Summer
of
Code