Hello World from Scratch

Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft

Hello World from Scratch

Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft

“If you wish to make an apple pie from scratch, you must first invent the universe.”

— Carl Sagan, Cosmos

Chirag Jha, B.Tech Electrical and Electronics Engineering, PES University (2018) Answered Jun 24, 2017

This phrase: “making an apple pie from scratch” has a really deep meaning if you look at it in a more general sense than just the words.

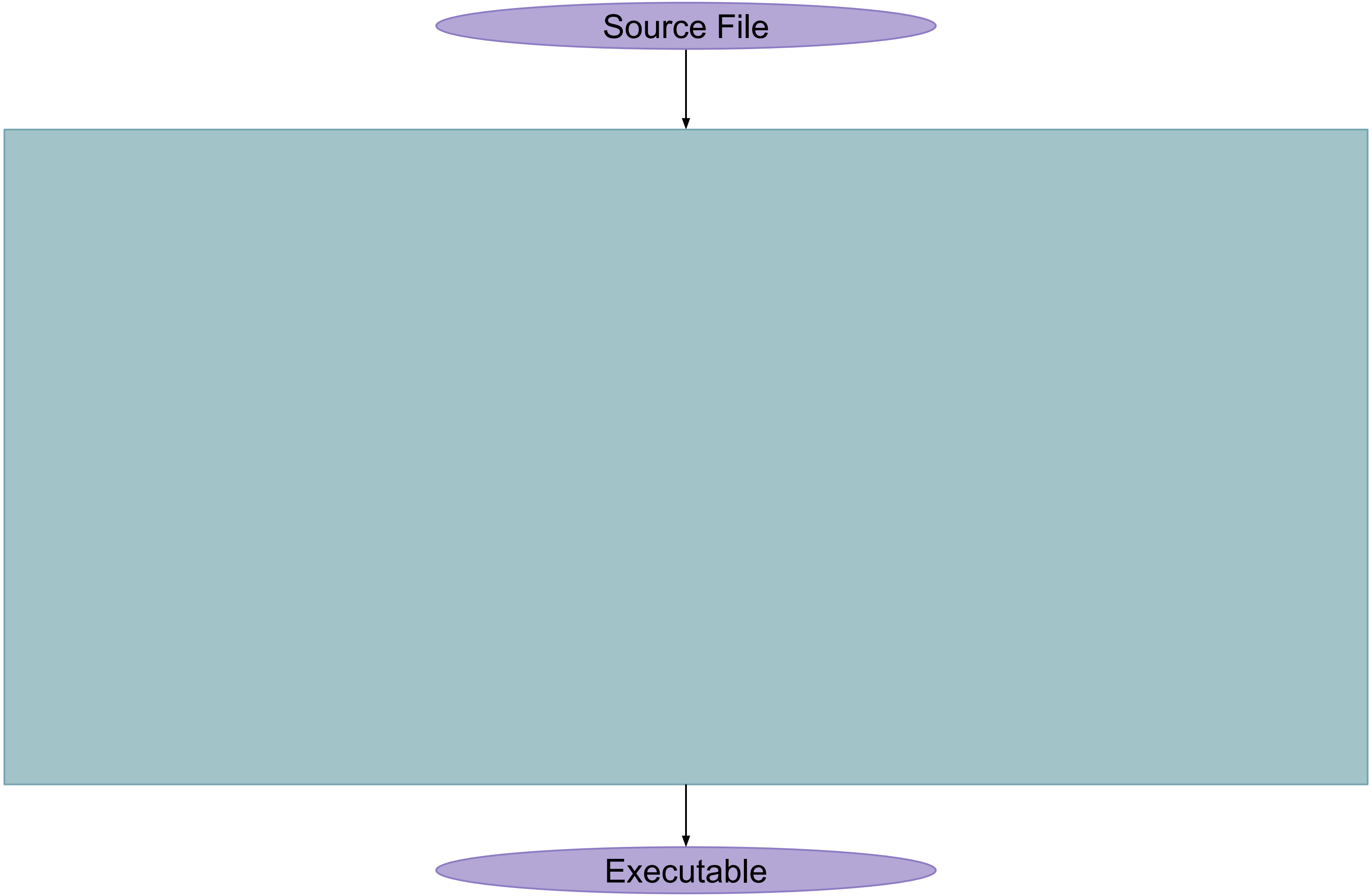
If you look at everything that exists in the universe, **we** tend to **forget how ‘complex’ things really are**, both the living and the non living. To arrive at a thing such as an apple pie, you need to go through all that ‘complexity’: from creating the universe, to the laws of nature: the physics and the chemistry that actually structure the apple pie.

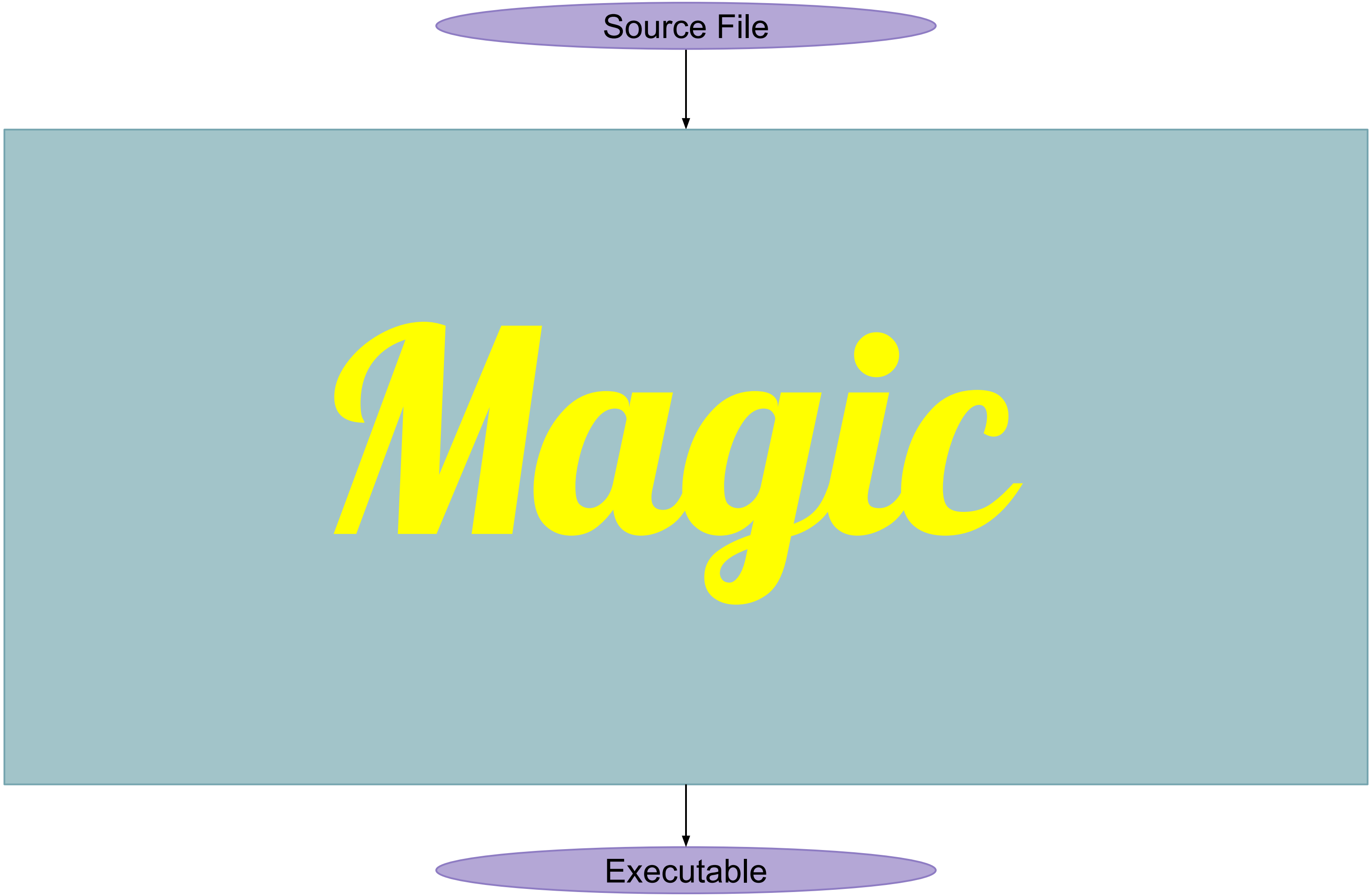
Agenda

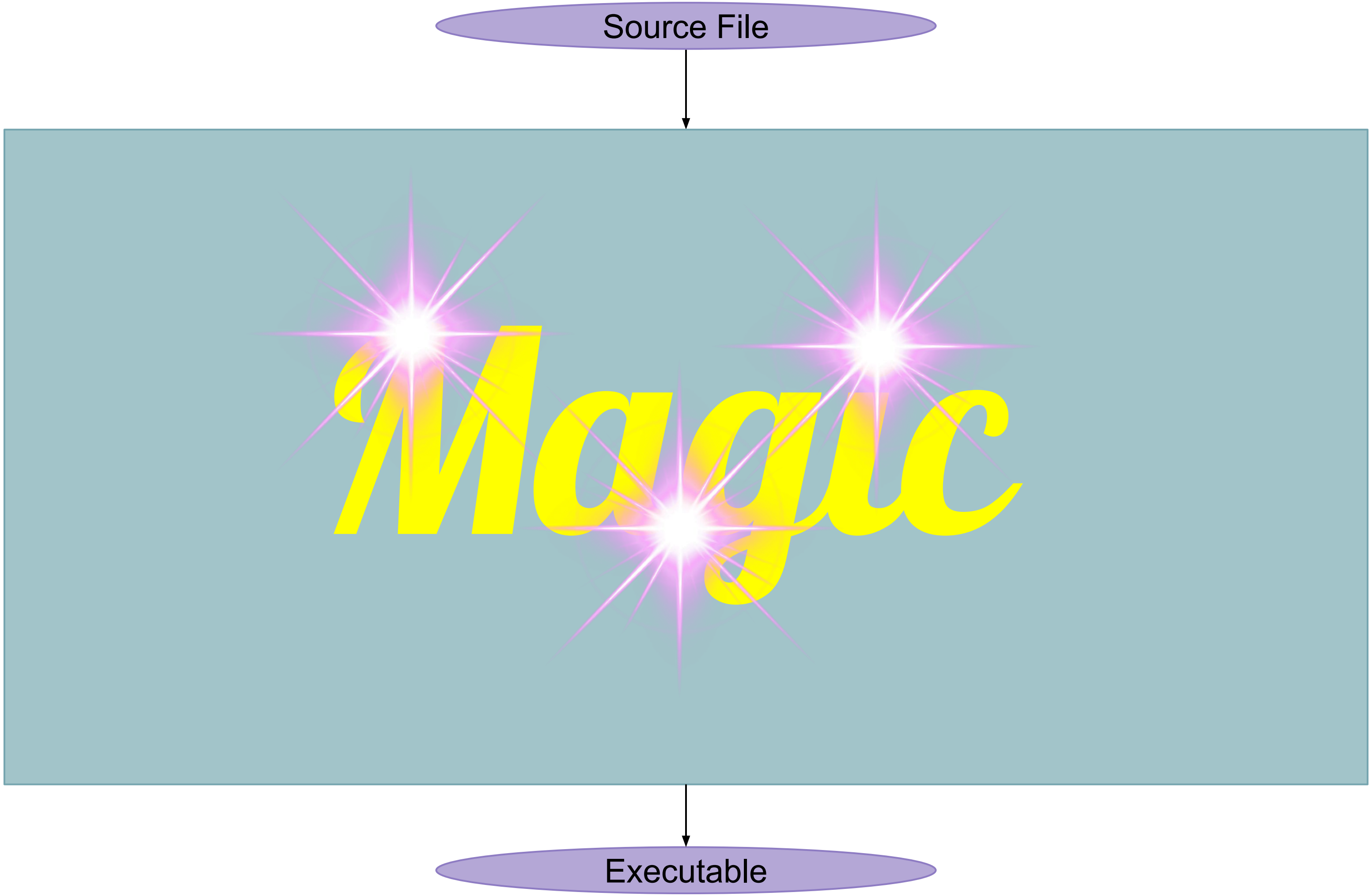
- Hello World in C
- Hello World in C++

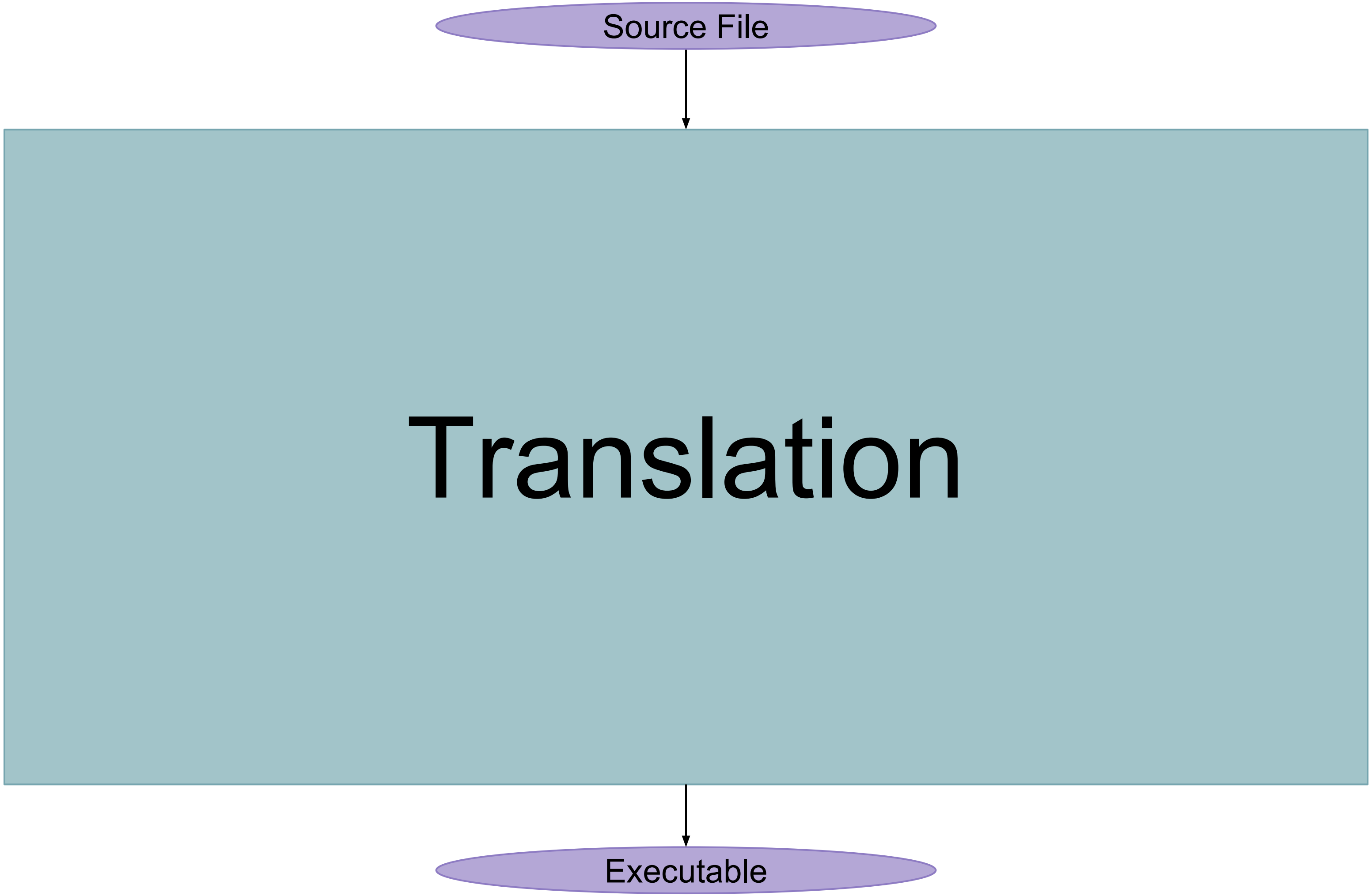
Hello World in C

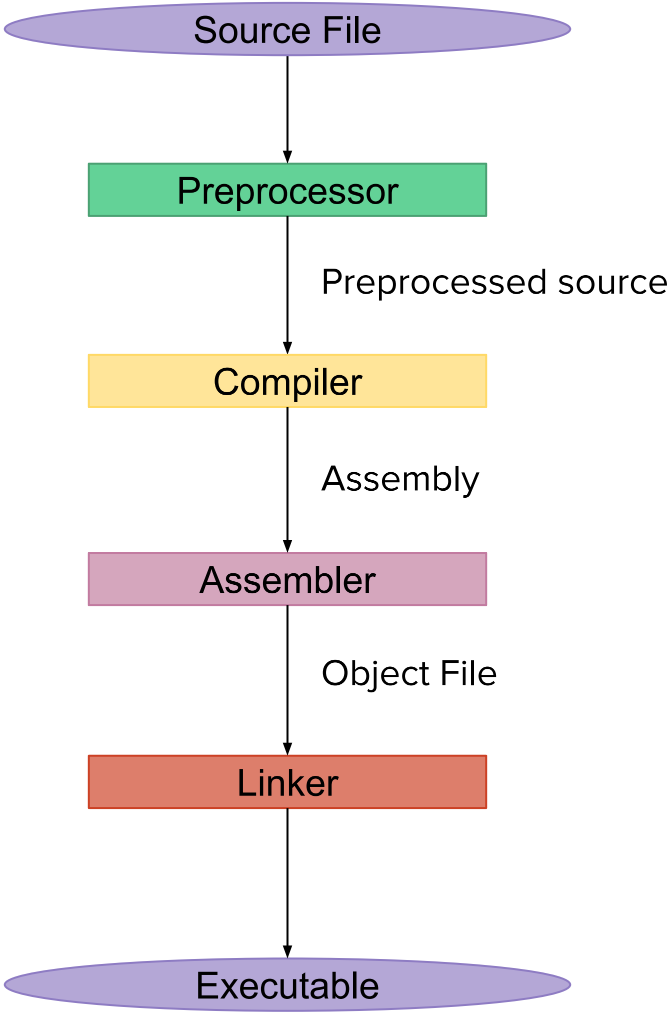
```c
#include <stdio.h>

int main() {
  puts("Hello World!");
}
```

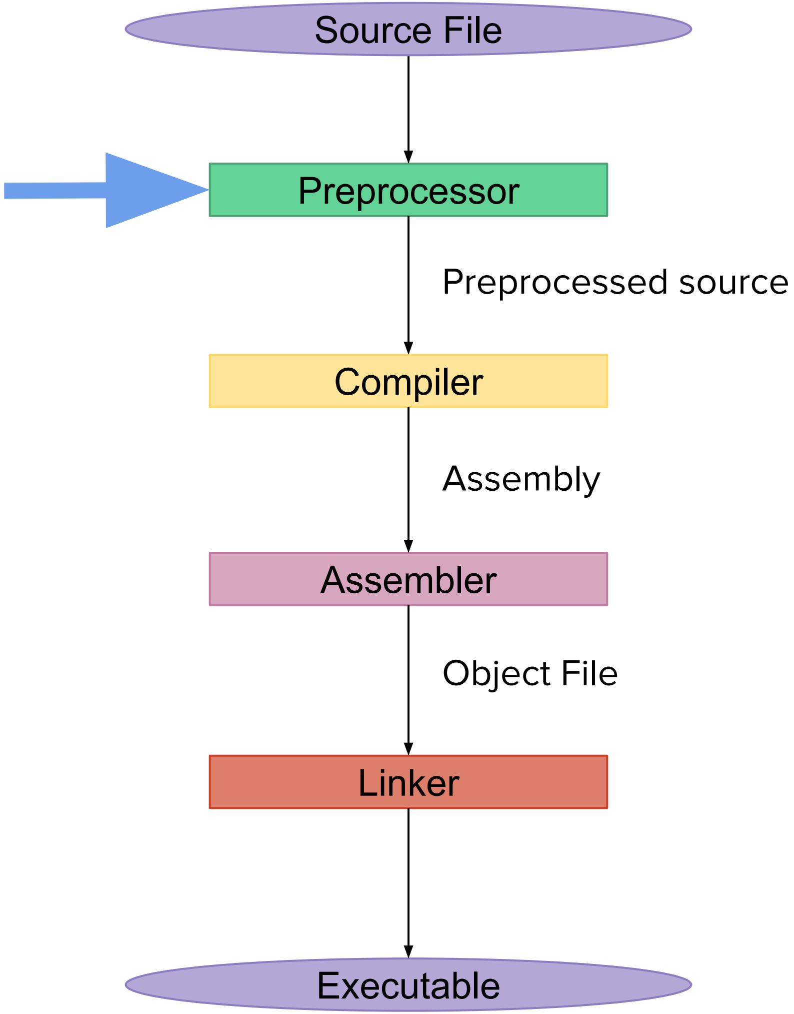

Source File
Executable


Source File
Magic
Executable


Source File
Magic
Executable


Source File
Translation
Executable


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

```
#define X Y
```

```
#define X(A) Y
```

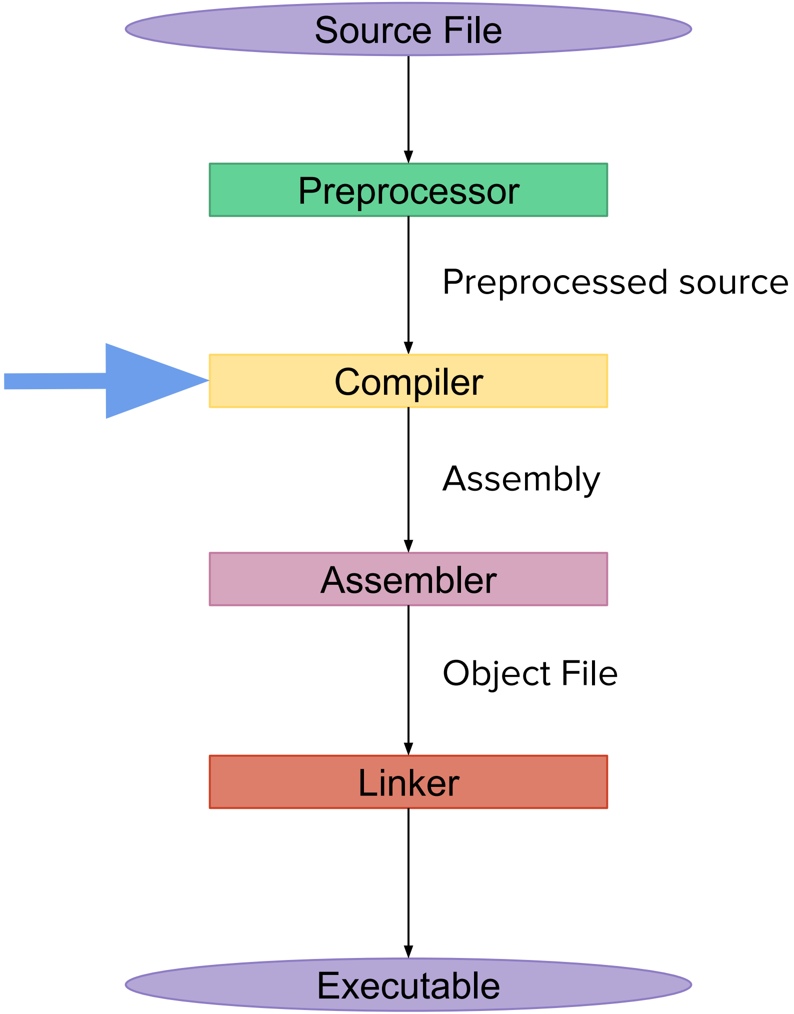
```
#define X(A) Y
X(array<int, 4>)
```

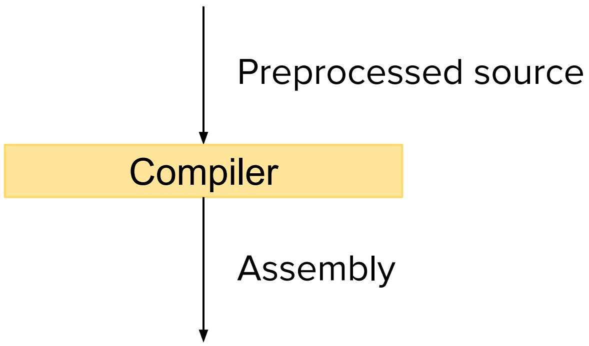
```
#define X(A) Y
X(array<int, 4>)
```

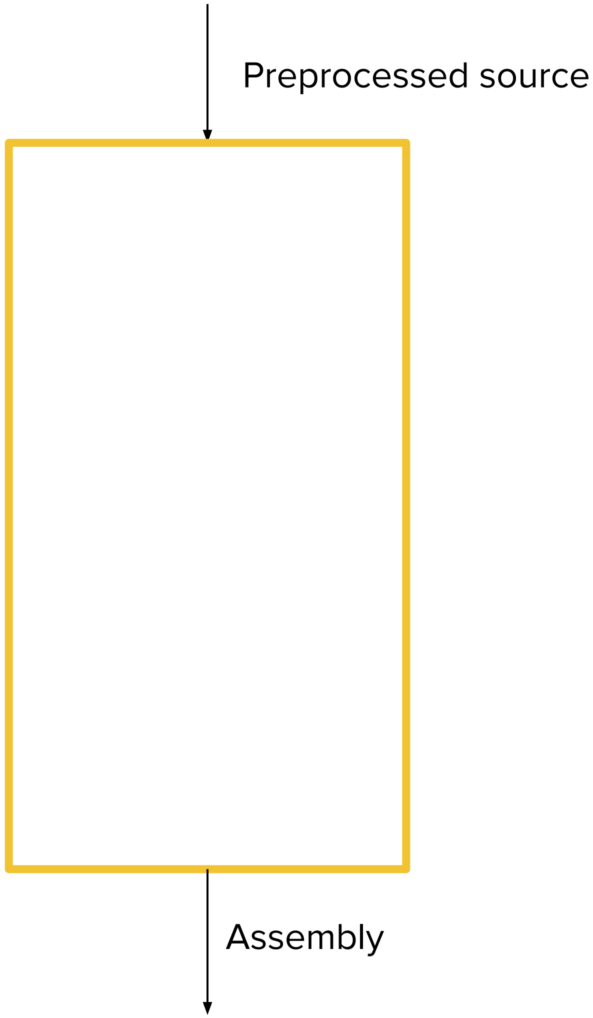
```
#ifdef ACCU
puts(“Hello ACCU!”);
#else
puts(“Hi person at home!”);
#endif
```

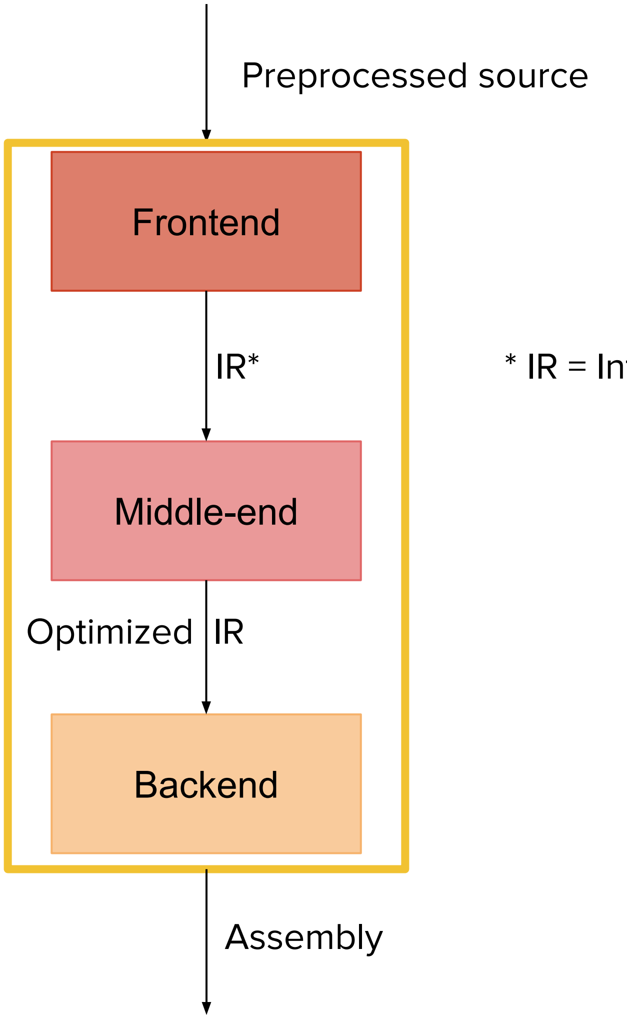
```
puts(“Hello ACCU!”);
```

```
#include <file>
#include “file”
```

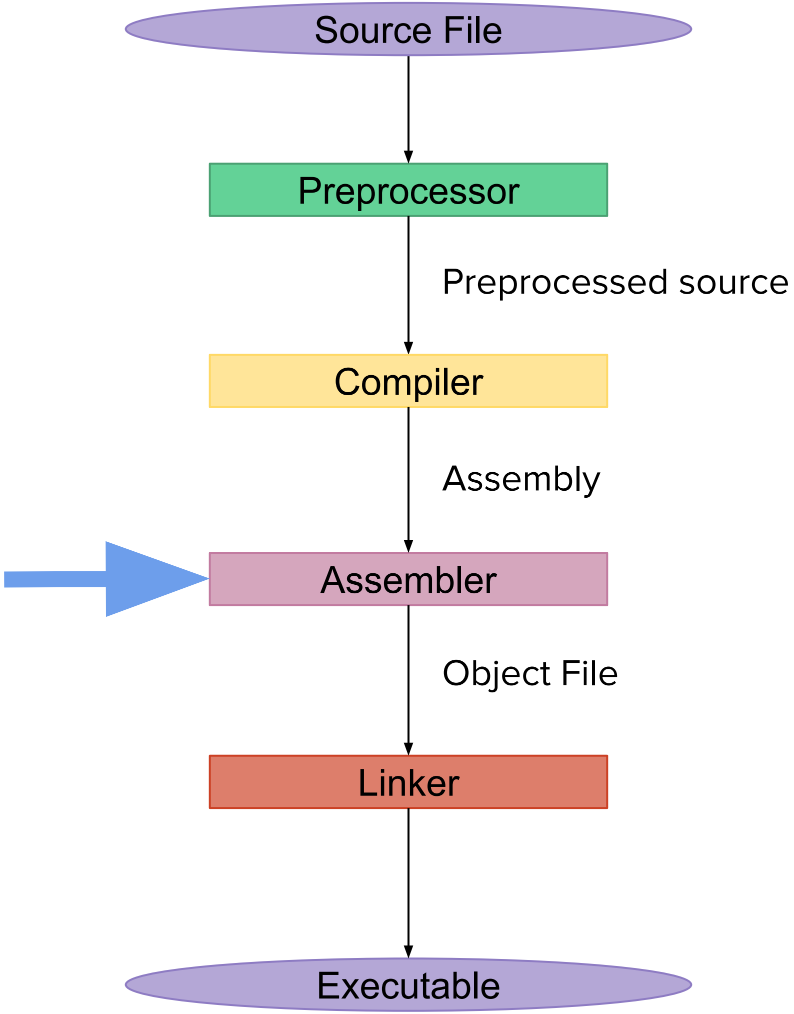

Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


Preprocessed source
Compiler
Assembly


Preprocessed source
Assembly



* IR = Intermediate Representation



Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

opcode	rs	rt	rd	shift	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

0	18	19	17	0	0x20

Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

000000	10010	10011	10001	00000	100000

Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

00000010 01010011 10001000 00100000

Assembler Directives

```
    .data
variable_name:
    .space 4
    .align 2
    .text
```

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                         34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

MINI

ELF HEADER

IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001

PROGRAM HEADER TABLE

EXECUTION INFORMATION

FIELDS	VALUES
p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

CODE

X86 ASSEMBLY	EQUIVALENT C CODE
mov ebx, 42 mov eax, SC_EXIT [1] int 80h	return 42;

ELF file types

Object File

Executable

Shared library

Core dump

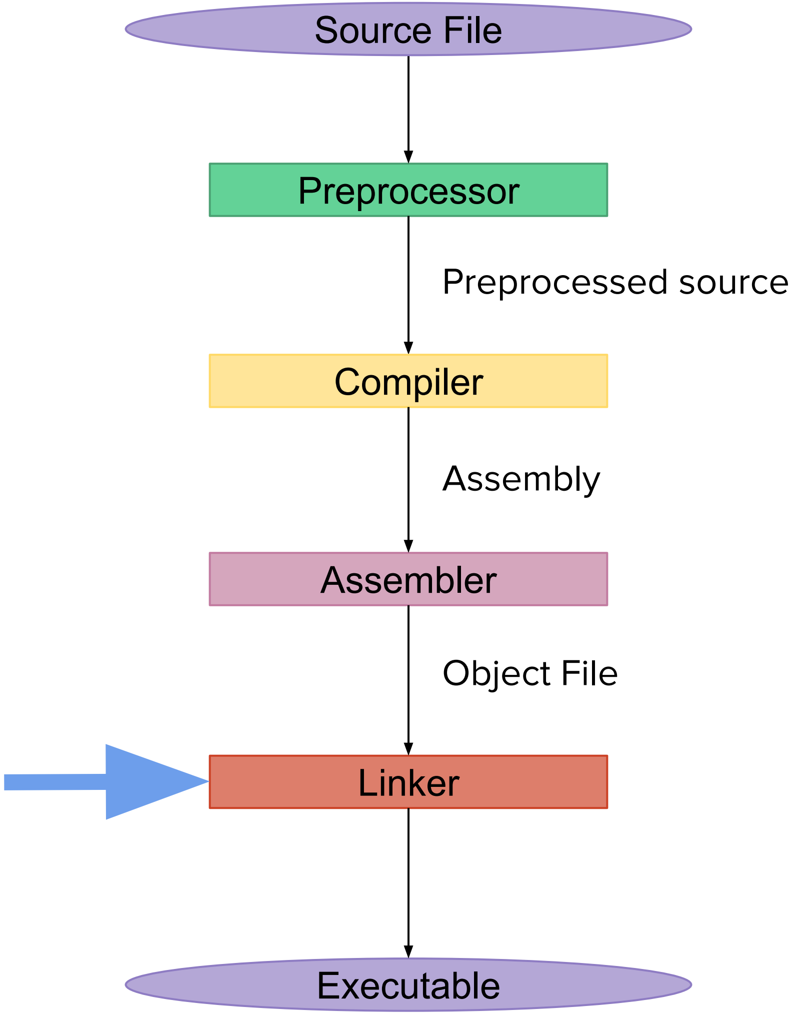
ELF file types

Object File - A part of your program in bits (sections)

Executable - Your whole program as a “whole”

Shared library - Shared bits between programs

Core dump - Your whole program as a crash dump


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

Linker

- Take all passed-in object files
- Create lookup table of symbols referenced

Linker

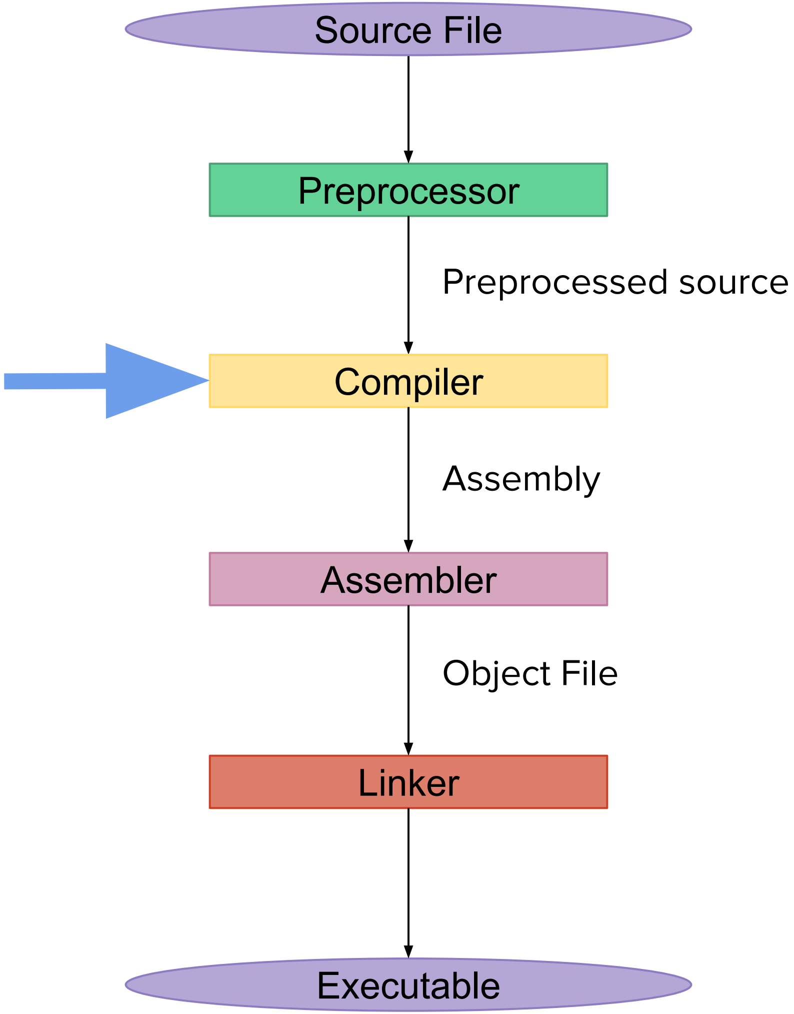
- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file

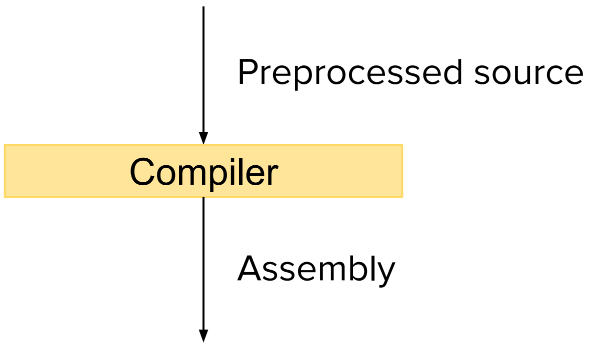
Linker

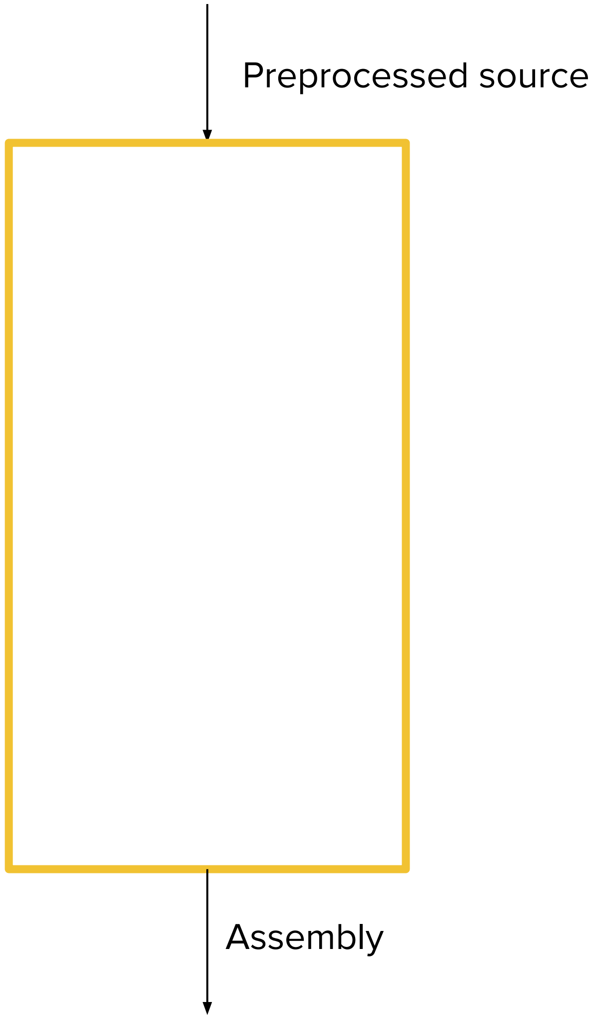
- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file
- Rewrite all references in the byte code to point to actual symbols

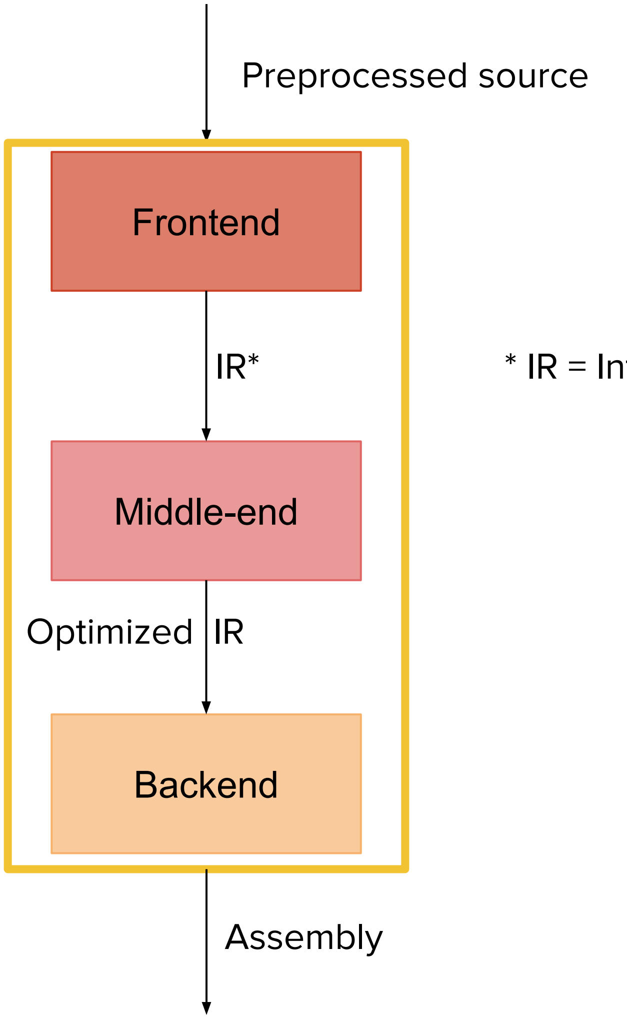
Linker

- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file
- Rewrite all references in the byte code to point to actual symbols
- Output all loaded symbols and their data to an executable

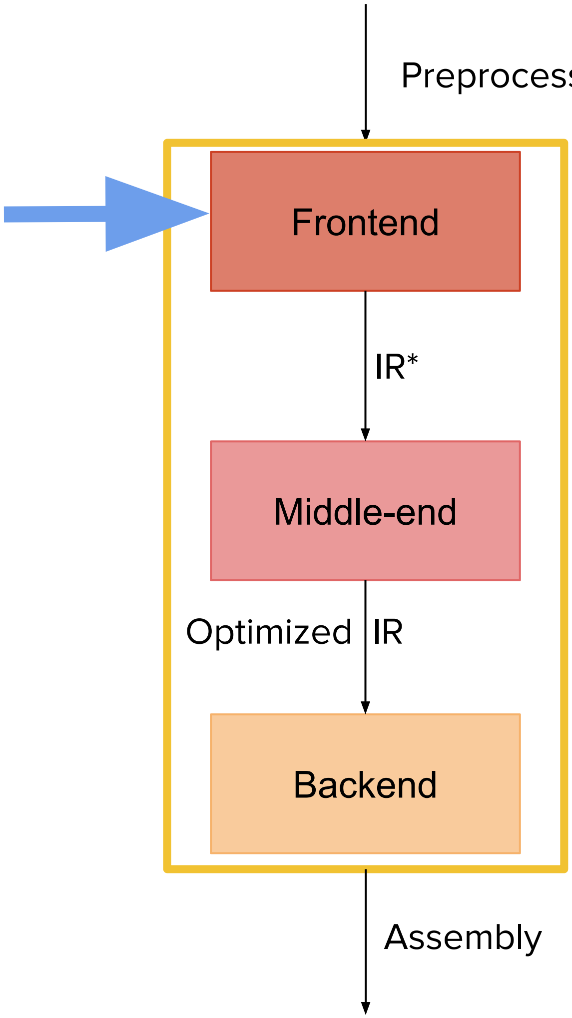

Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


Preprocessed source
Compiler
Assembly

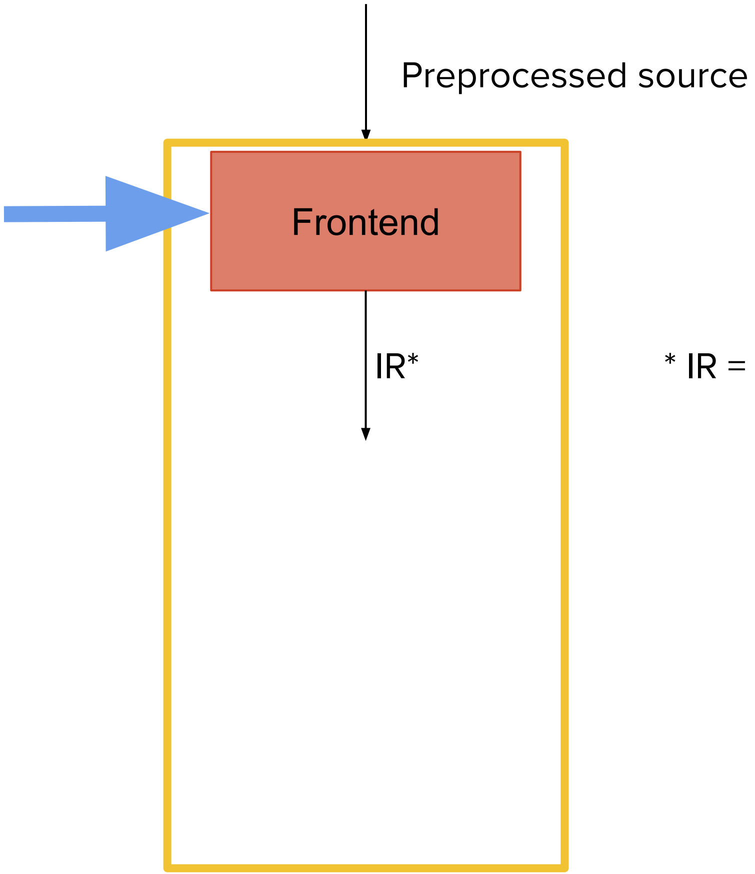

Preprocessed source
Assembly



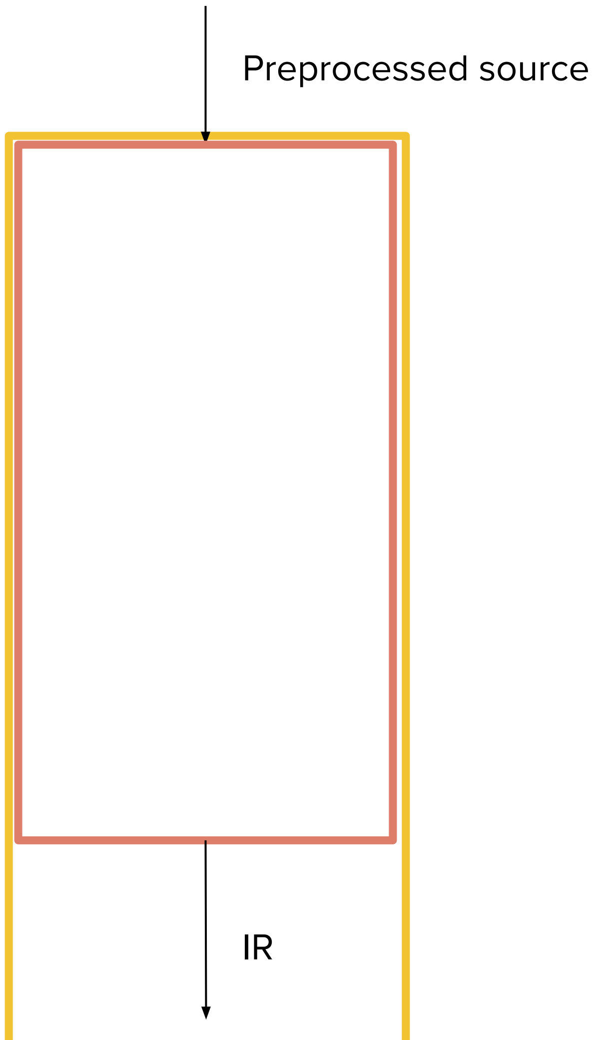
* IR = Intermediate Representation

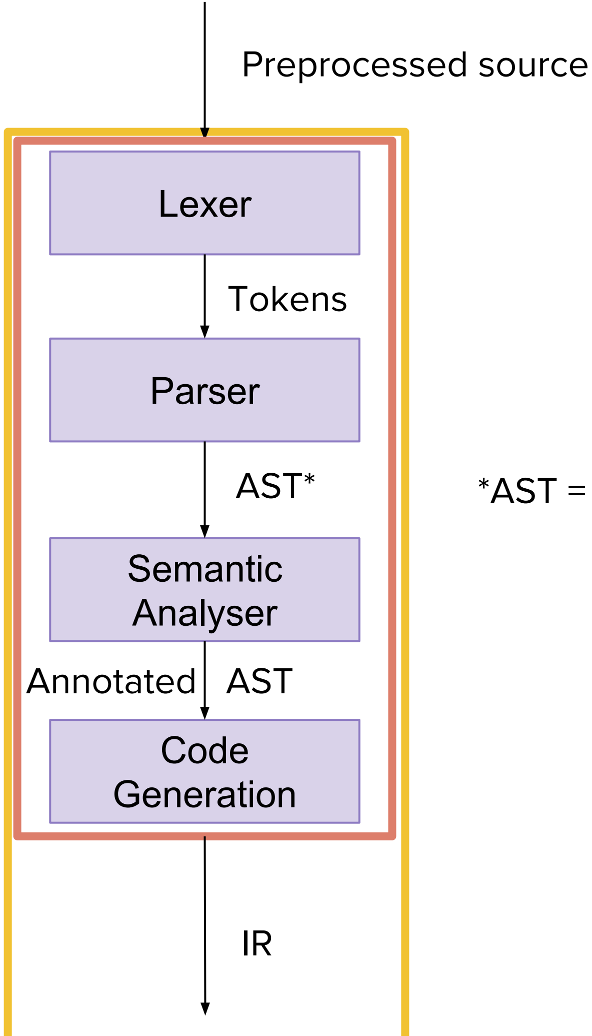


* IR = Intermediate Representation

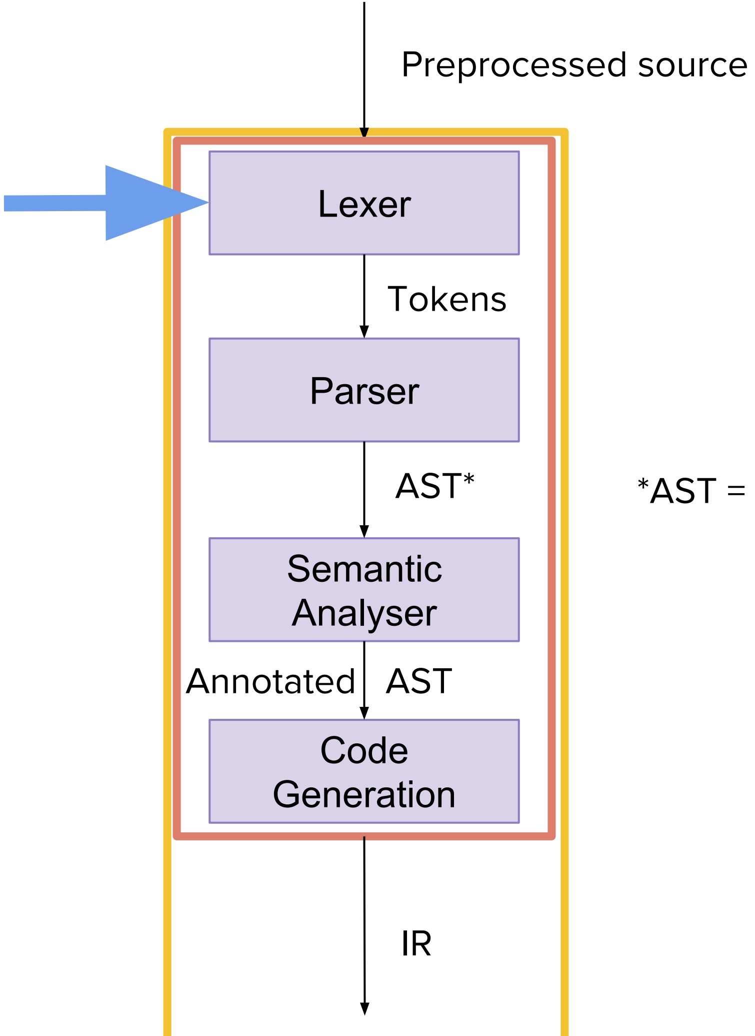


* IR = Intermediate Representation


Preprocessed source
IR



*AST = Abstract Syntax Tree



*AST = Abstract Syntax Tree

Tokens

```
int main() {
  puts("Hello world!");
}
```

Tokens

```
int main() {
  puts("Hello world!");
}
```

- Have to deal with:
  - Whitespace
  - Identifiers
  - Strings
  - Punctuation
  - Multi-char operators

ID(int)

Tokens

```
int main() {
  puts("Hello world!");
}
```

Tokens

ID(int)
ID(main)

```
int main() {
  puts("Hello world!");
}
```

Tokens

```
int main() {
  puts("Hello world!");
}
```

ID(int)
ID(main)
LPAREN

Tokens

```
int main() {
  puts("Hello world!");
}
```

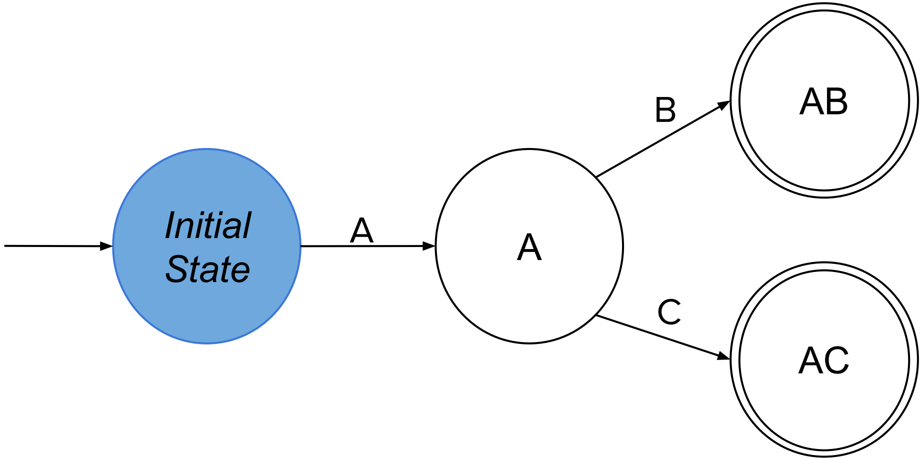
ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE

Lexer Implementation

Example: recognize the tokens AB and AC

Lexer Implementation

Example: recognize the tokens AB and AC



Lexer Implementation - Switch

```
while (keep_going) {
  switch(get_char()) {
  case 'A': {
    switch(get_char()) {
    case 'B':tokens.push_back(token::ab); break;
    case 'C':tokens.push_back(token::ac); break;
    }
  }
  }
}
```

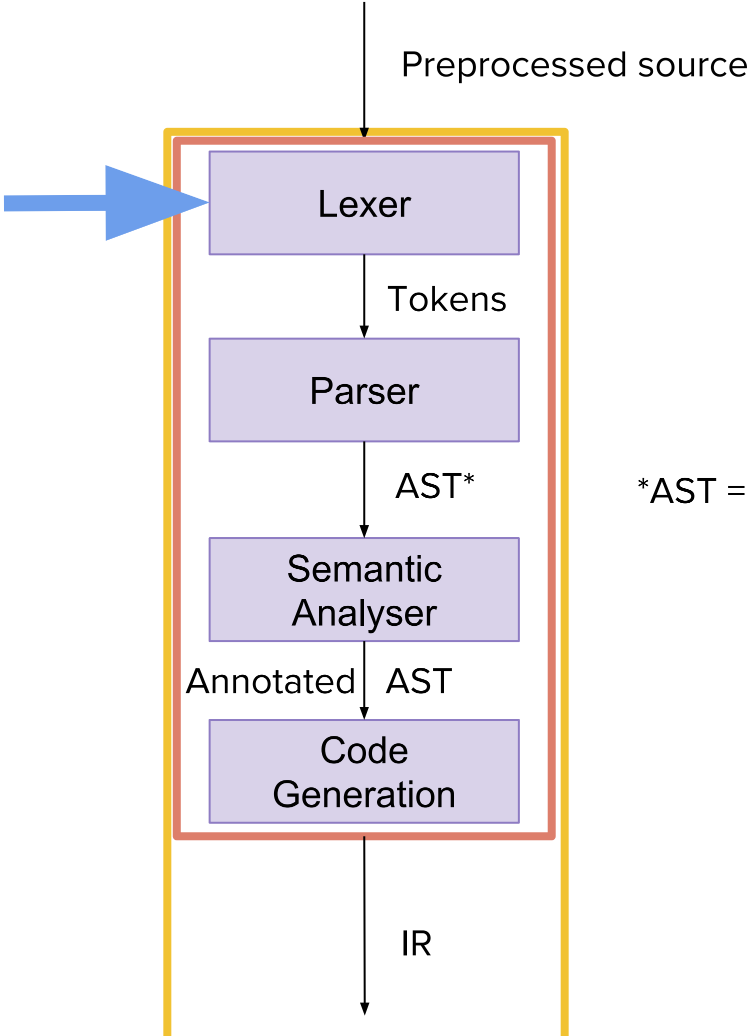
Lexer Implementation - Flex

```
%%
AB { return token::ab; }
AC { return token::ac; }
%%
```

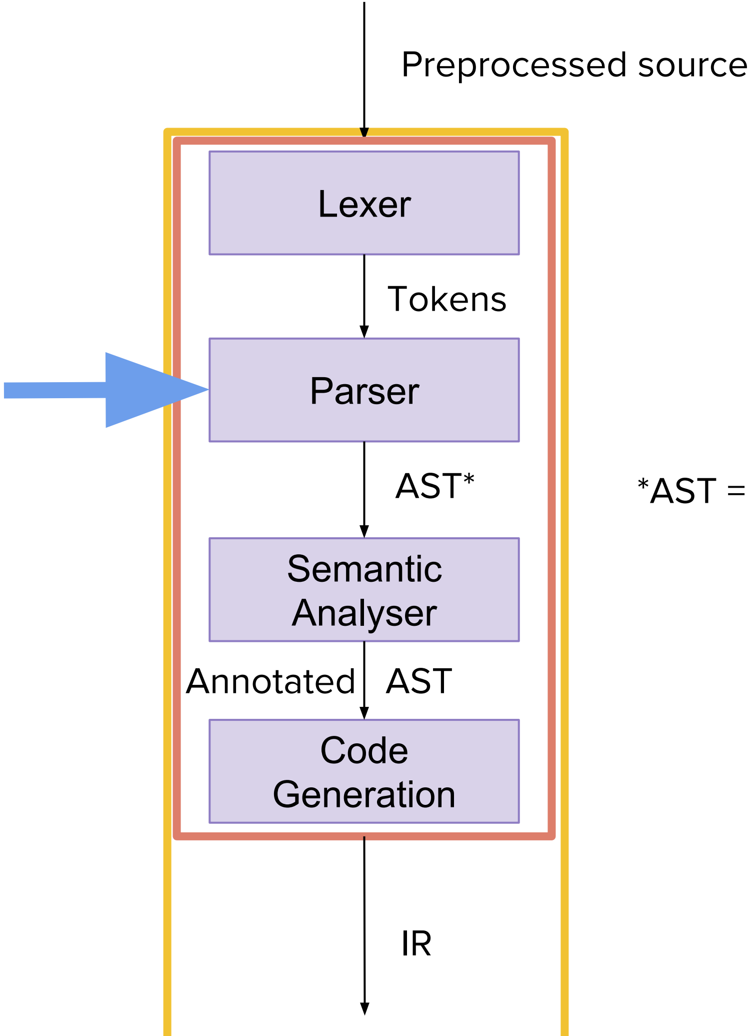
https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator)

Lexer Implementation - Flex

```
%%
{letter}({letter}|{digit})*
   { yylval.id = strdup(yytext); return IDENT; }

{digit}+
   { yylval.num = atoi(yytext); return NUMBER; }

[ \t\n\r] /* skip whitespace */

. { printf("Unknown char\n”); return UNKNOWN; }
%%
```



*AST = Abstract Syntax Tree



*AST = Abstract Syntax Tree

ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE

(Extended) Bachus-Naur Form

Definition ::= Name ‘::=’ Body

Something ::= Parts That Make It ‘.’

Repetition ::= { Something }

Optional ::= [Something]

Result ::= ‘a’ [{ Very } Flexible] Language [‘for’ Grammars]

Type ::= ID(int) | ...

Function ::= Type Name LPAREN [ArgDecl { ‘,’ ArgDecl }]
RPAREN LBRACE { Statement } RBRACE

Statement ::= Expression SEMI | ...

Expression ::= Name LPAREN [Expression { ‘,’ Expression }]
RPAREN | String | ...

Name ::= ID(...)

ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE

Type ::= ID(int) | …

Name ::= ID(…)

Type → ID(int)
Name → ID(main)
LPAREN
RPAREN
LBRACE
Name → ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE

Function ::= Type Name LPAREN [ArgDecl { ‘,’ ArgDecl }]
RPAREN LBRACE { Statement } RBRACE

Function → Type → ID(int)
Name → ID(main)
LPAREN
<empty>
RPAREN
LBRACE
Name → ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE

Statement ::= Expression SEMI | …

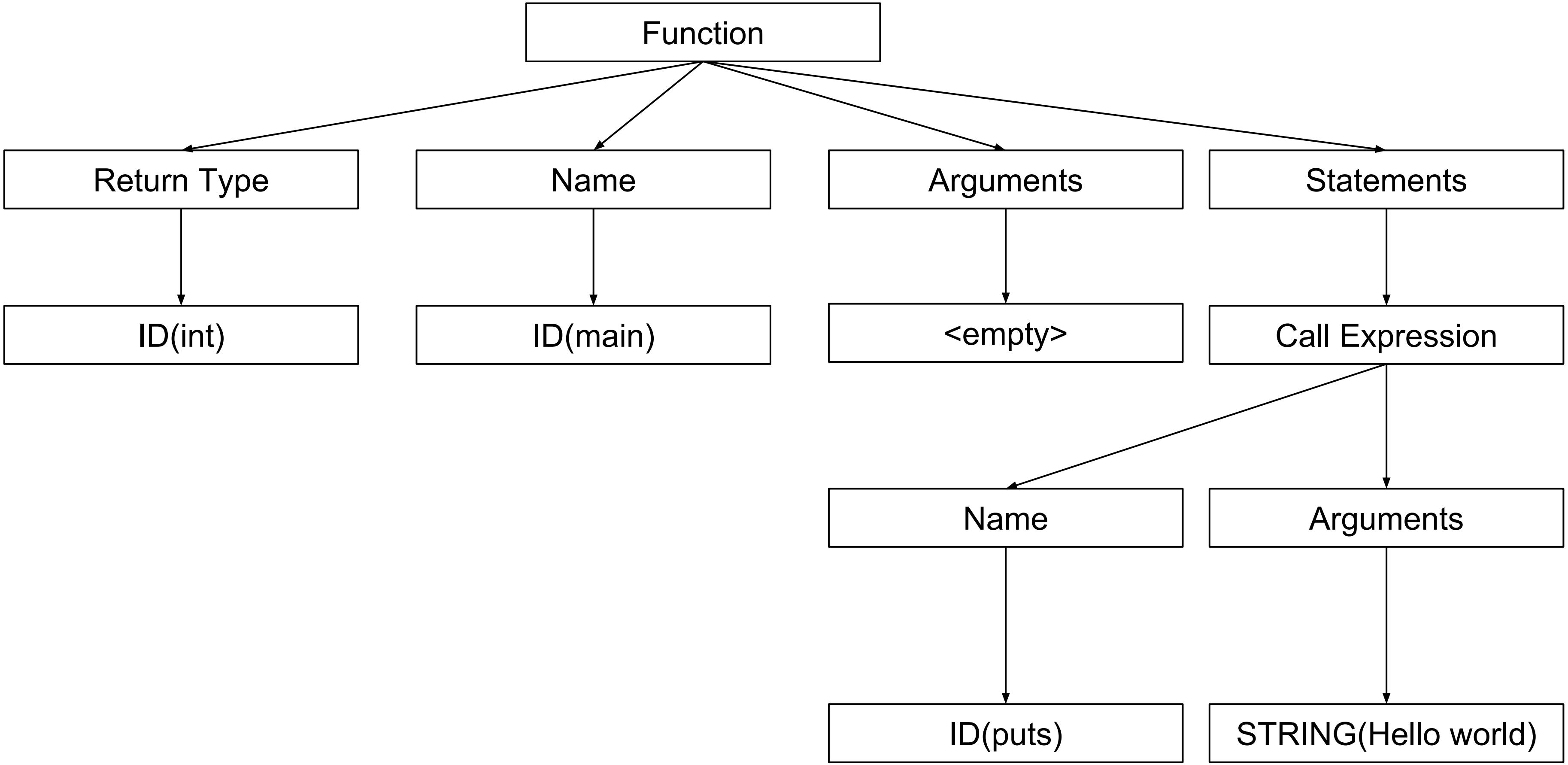
```
Function ➔ Type ➔ ID(int)
           Name ➔ ID(main)
           LPAREN
                <empty>
           RPAREN
           LBRACE
           Statement ➔ Name ➔ ID(puts)
                       LPAREN
                       STRING(Hello world!)
                       RPAREN
                       SEMI
           RBRACE
```

Expression ::= Name LPAREN [Arg { ‘,’ Arg }] RPAREN |
String | …

```
Function → Type → ID(int)
           Name → ID(main)
           LPAREN
                 &lt;empty&gt;
           RPAREN
           LBRACE
           Statement → Expression → Name → ID(puts)
                                    LPAREN
                                    STRING(Hello world!)
                                    RPAREN
                    SEMI
           RBRACE
```

Expression ::= Name LPAREN [Arg { ‘,’ Arg }] RPAREN |
String | …

```
Function → Type → ID(int)
           Name → ID(main)
           LPAREN
                 <empty>
           RPAREN
           LBRACE
           Statement → Expression → Name → ID(puts)
                                  LPAREN
                                  Expression → STRING(Hello world!)
                                  RPAREN
                       SEMI
           RBRACE
```


Function
Return Type
Name
Arguments
Statements
ID(int)
ID(main)
<empty>
Call Expression
Name
Arguments
ID(puts)
STRING(Hello world)

Parser Implementation

```
selection_statement
    : IF LPAREN expression RPAREN statement ELSE statement
    | IF LPAREN expression RPAREN statement
    | SWITCH LPAREN expression RPAREN statement
```

```
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {

}
```

```
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);




}
```

```cpp
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {




  }

}
```

```
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {
    auto cond = parse_expression(ctx);



  }

}
```

```
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {
    auto cond = parse_expression(ctx);
    auto if_stmt = parse_statement(ctx);



  }

}
```

```
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {
    auto cond = parse_expression(ctx);
    auto if_stmt = parse_statement(ctx);
    return std::make_unique<selection_statement>
              (cond, if_stmt);
  }

}
```

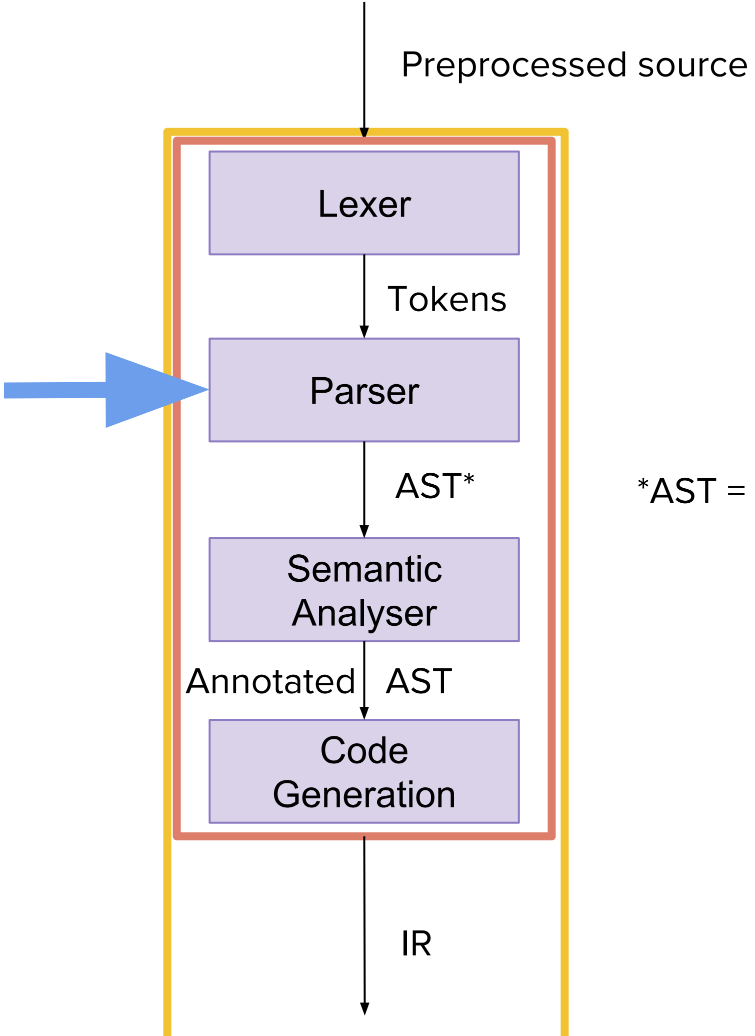
```cpp
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {
    auto cond = parse_expression(ctx);
    auto if_stmt = parse_statement(ctx);
    return std::make_unique<selection_statement>
              (cond, if_stmt);
  }
  //...
}
```

Generator vs. Hand-Written

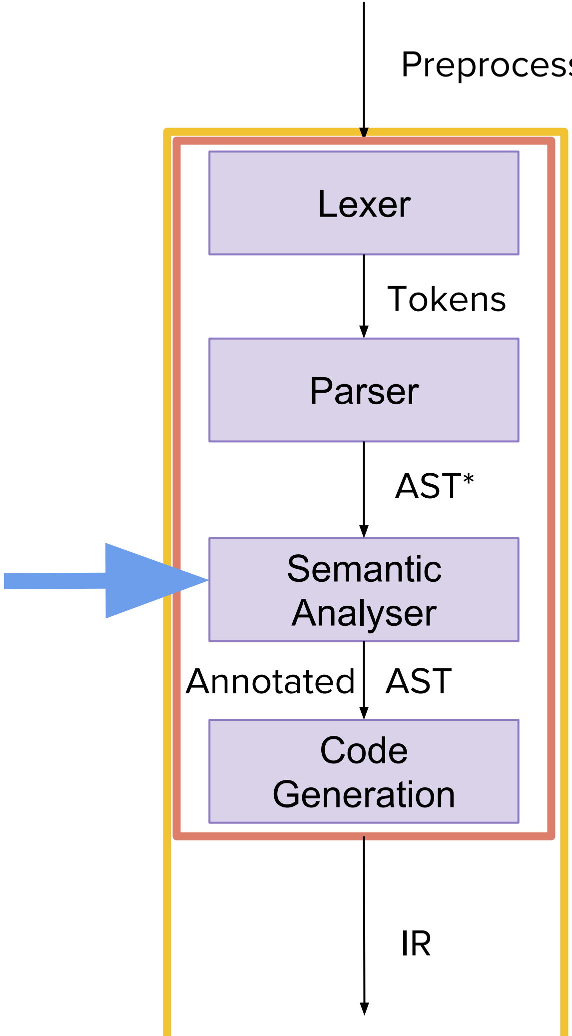
- Generator
  - Fast to get started
  - Can generate efficient parsers w/o much code
  - Grammar checker
- Hand-Written
  - Easier to handle and report errors
  - Easier to debug
  - Can write a faster, friendlier parser with enough work

Parser Implementation

```
selection_statement
    : IF LPAREN expression RPAREN statement ELSE statement
    | IF LPAREN expression RPAREN statement
    | SWITCH LPAREN expression RPAREN statement

if (a) if (b) puts(‘x’); else puts(‘y’);
```



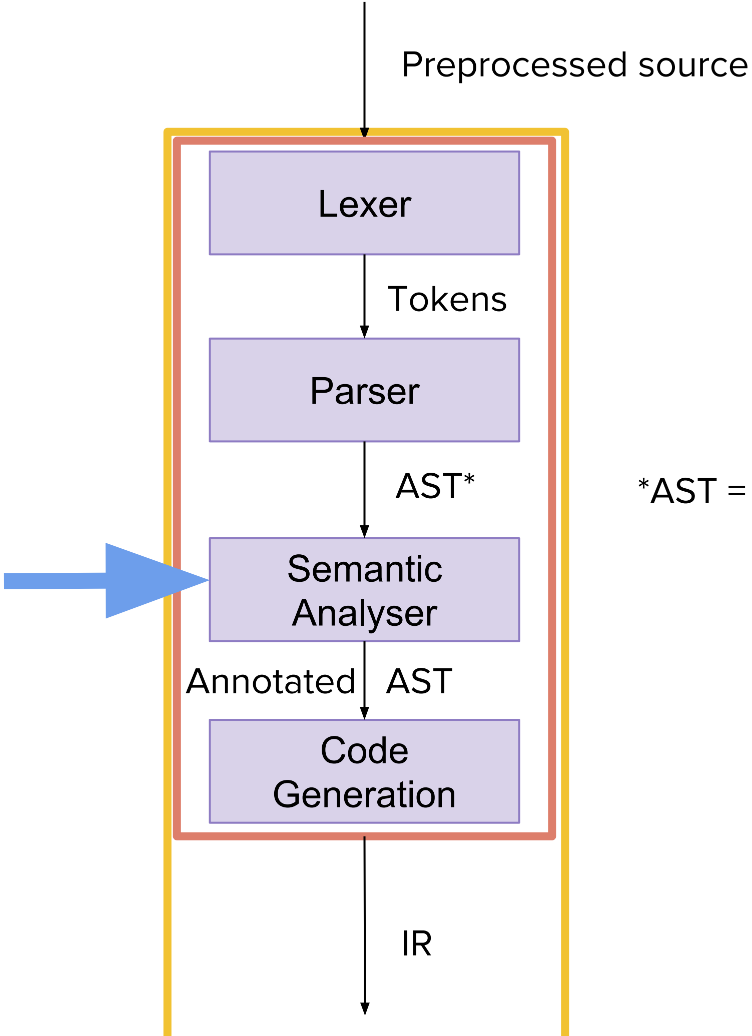
*AST = Abstract Syntax Tree



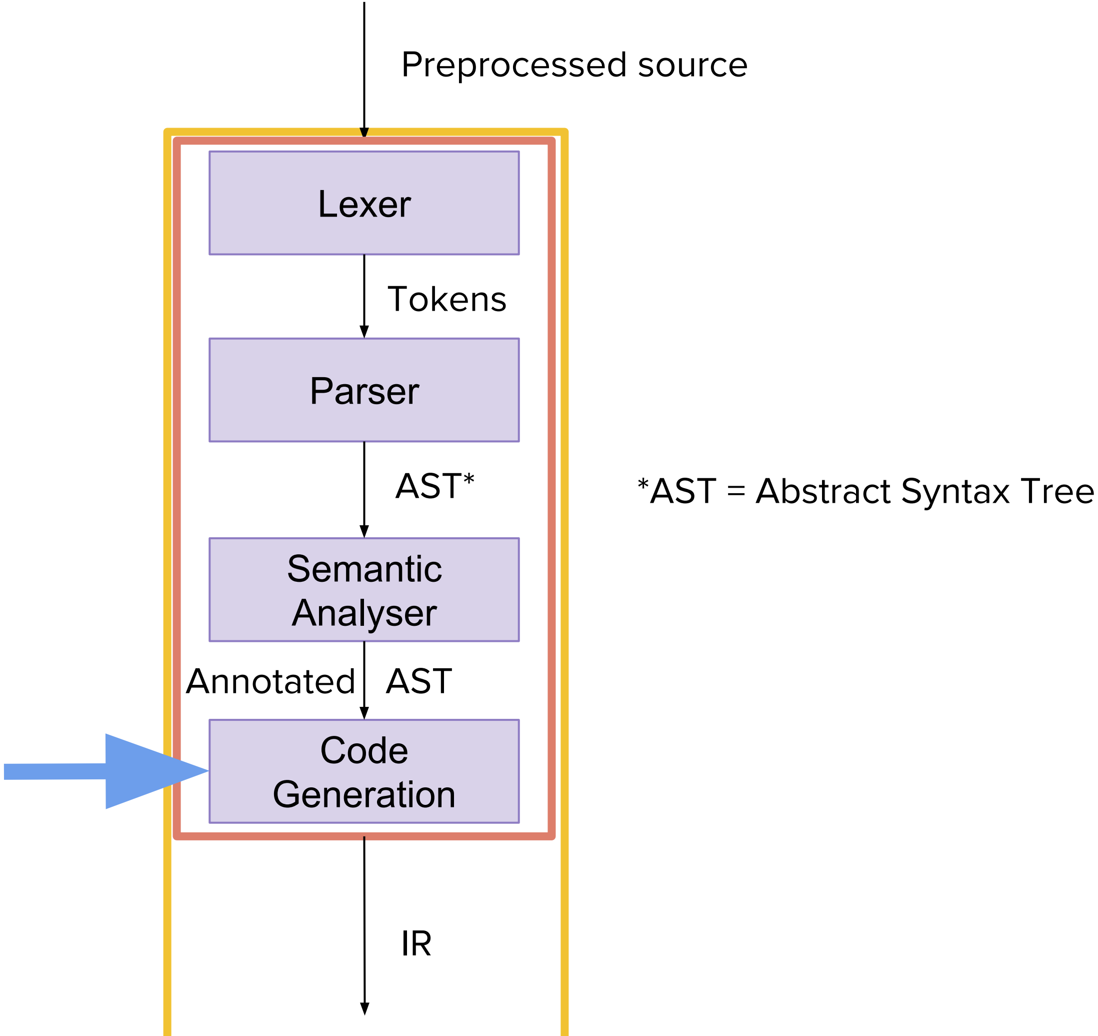
*AST = Abstract Syntax Tree

```cpp
int main() {
  auto s = "wat";
}
```

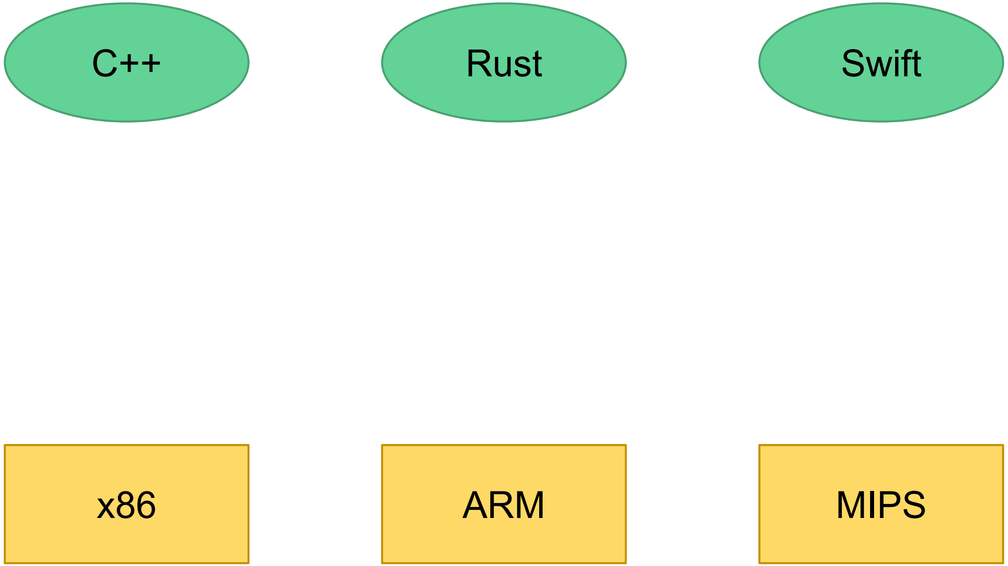
```
int main() {
  int i = "wat";
}
```



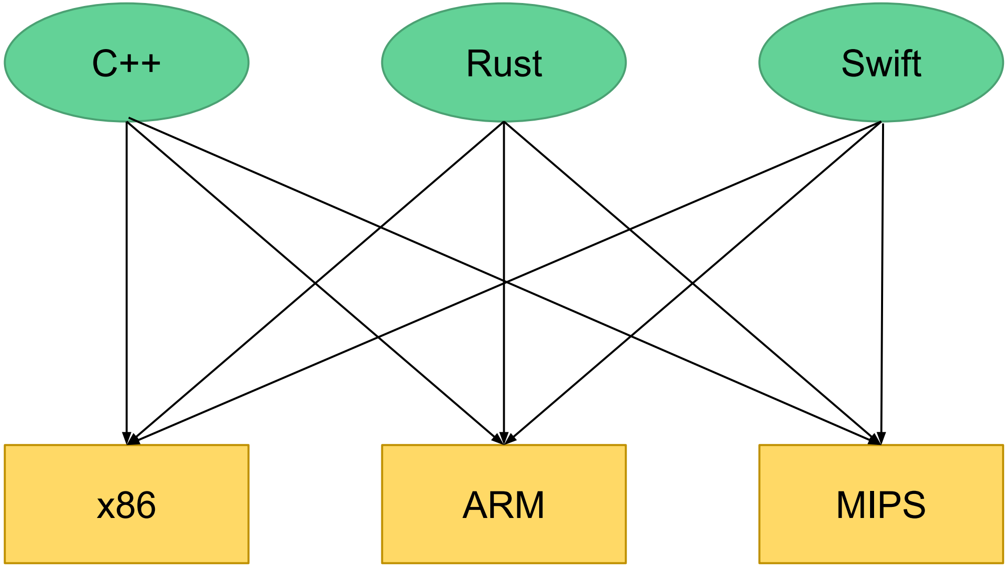
*AST = Abstract Syntax Tree


Preprocessed source
Lexer
Tokens
Parser
AST*
Semantic Analyser
Annotated AST
Code Generation
IR
*AST = Abstract Syntax Tree

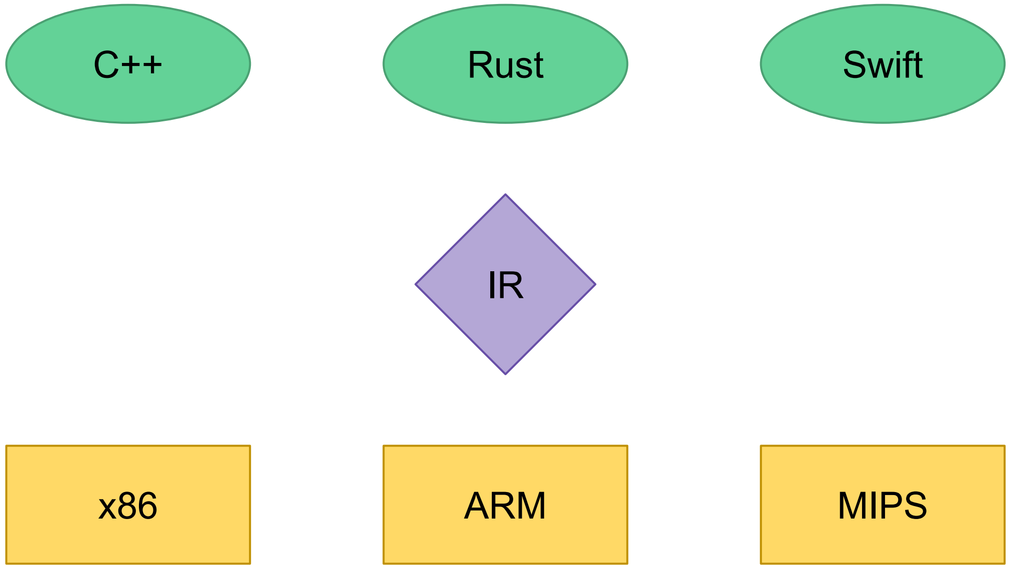
Intermediate Representation



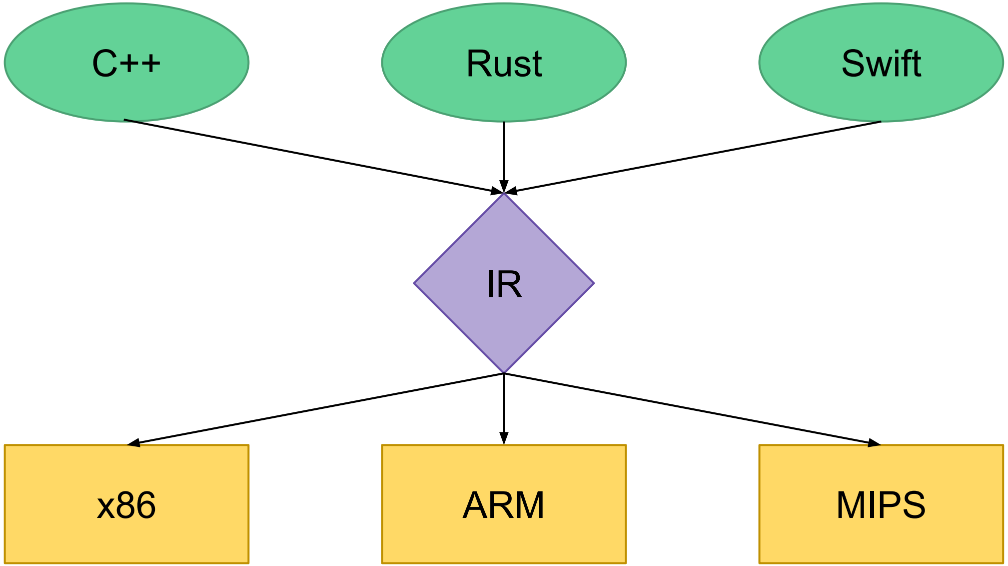
Intermediate Representation



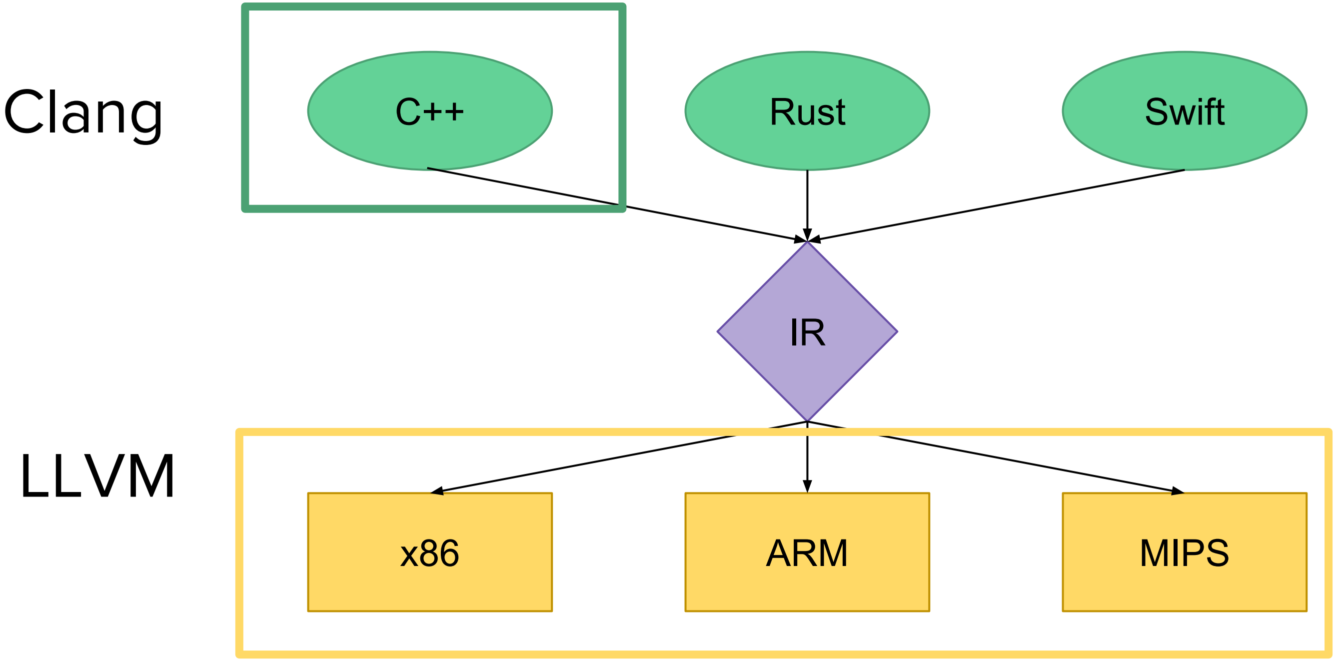
Intermediate Representation



Intermediate Representation



Intermediate Representation



```
Function → Type → ID(int)
           Name → ID(main)
           LPAREN
                 <empty>
           RPAREN
           LBRACE
           Statement → Expression → Name → ID(puts)
                                    LPAREN
                                    Expression →
                                 STRING(Hello world!)
                                    RPAREN
                 SEMI
           RBRACE
```

```
Function → Type → ID(int)
           Name → ID(main)
           LBRACE
           Statement → Expression → Name → ID(puts)
                                   LPAREN
                                   Expression →
                                 STRING(Hello world!)
                                   RPAREN
                       SEMI
           RBRACE
```

main:
LBRACE

Statement → Expression → Name → ID(puts)
LPAREN
Expression → STRING(Hello world!)
RPAREN
SEMI

RBRACE

main:
LBRACE

r0 = Expression → STRING(Hello world!)

Statement → Expression → Name → ID(puts)
LPAREN
r0
RPAREN
SEMI

RBRACE

main:
LBRACE

r0 = Expression ➔ STRING(Hello world!)

r1 = Expression ➔ Name ➔ ID(puts)
LPAREN
r0
RPAREN

Statement ➔ r1 SEMI

RBRACE

main:
LBRACE

r0 = STRING(Hello world!)

r1 = Expression → Name → ID(puts)
LPAREN
r0
RPAREN

Statement → r1 SEMI

RBRACE

main:
LBRACE

r0 = STRING(Hello world!)

r1 = call puts(r0)

Statement ➜ r1 SEMI

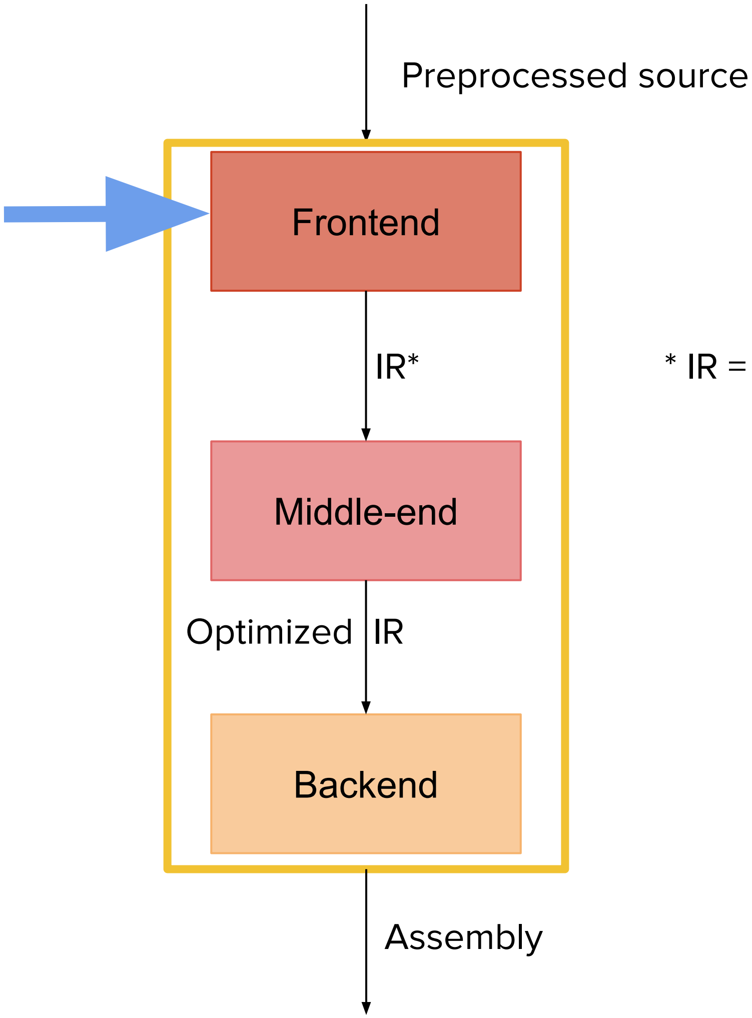
RBRACE

```
main:
LBRACE

        r0 = STRING(Hello world!)

        r1 = call puts(r0)

        (discard r1)

RBRACE
```

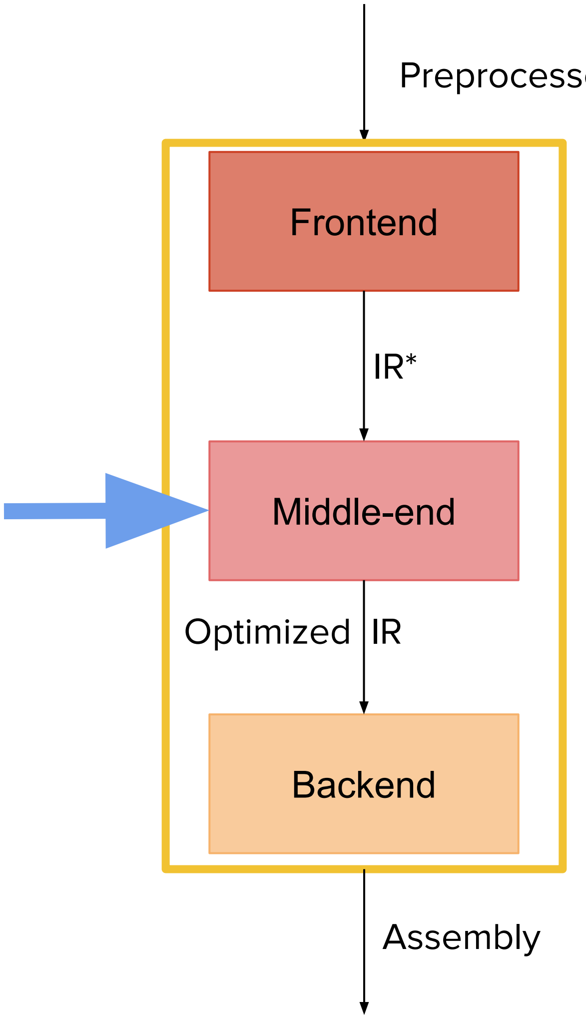
```
main:
LBRACE

        r0 = STRING(Hello world!)

        r1 = call puts(r0)

RBRACE
```

```
main:
        set up stack frame

        r0 = STRING(Hello world!)

        r1 = call puts(r0)

RBRACE
```

```
main:
        set up stack frame

        r0 = STRING(Hello world!)

        r1 = call puts(r0)

        remove stack frame
        return 0
```



* IR = Intermediate Representation



* IR = Intermediate Representation

When are variables “live?

Liveness Analysis

Live out

Live in

Liveness Analysis

Live out for I: live ins of all successors

Live in

Liveness Analysis

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

$$Live_{out}(I) = \cup_{S \in succ(I)} Live_{in}(S)$$
$$Live_{in}(I) = (Live_{out}(I) - DEF(I)) \cup USE(I)$$

Liveness Analysis

```
E = 42;
A = 12;
B = 32;
C = f(B);
D = f(A);
f(D);
f(C);
```

Liveness Analysis

```
E = …
A = …
B = …
… = f(B)
C = …
… = f(A)
D = …
… = f(D)
… = f(C)
```

Liveness Analysis

```
def(E)

def(A)

def(B)

use(B)

def(C)

use(A)

def(D)

use(D)

use(C)
```

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D)
use(C)
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D)
use(C) -            out = {}
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D)
use(C) - in = {C}, out = {}
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D) -                out = {C}
use(C) - in = {C}, out = {}
```

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D) -            out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

```
def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables

Liveness Analysis

```
def(E)

def(A)

def(B)

use(B)

def(C)

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

Liveness Analysis

```
def(E)

def(A)

def(B)

use(B)

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

Liveness Analysis

```
def(E)

def(A)

def(B)

use(B) - in = {A,B}, out = {A}

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

Liveness Analysis

```
def(E)

def(A)

def(B) - in = {A}, out = {A,B}

use(B) - in = {A,B}, out = {A}

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

Liveness Analysis

```
def(E)

def(A) - in = {}, out = {A}

def(B) - in = {A}, out = {A,B}

use(B) - in = {A,B}, out = {A}

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

Liveness Analysis

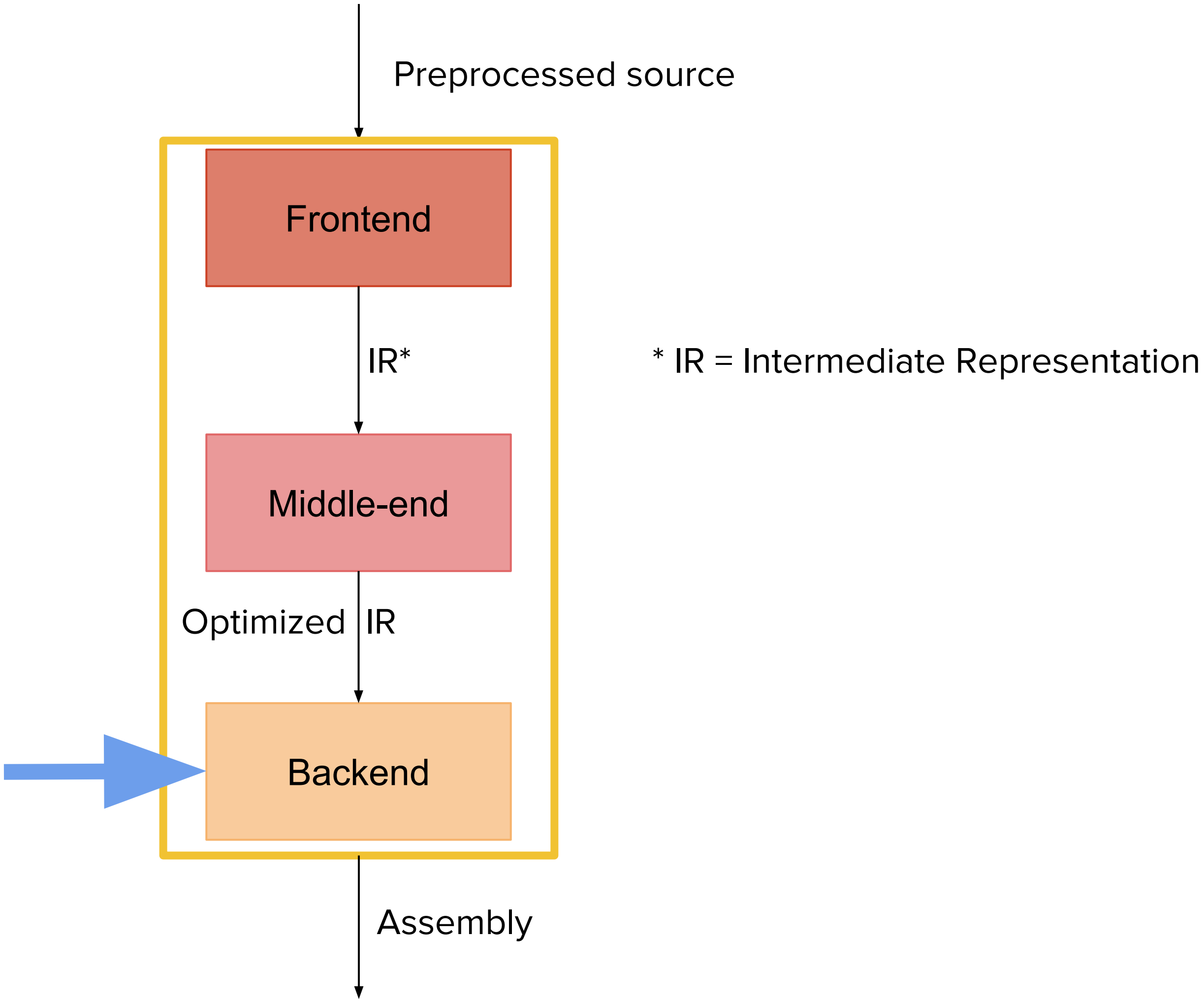
```
def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```

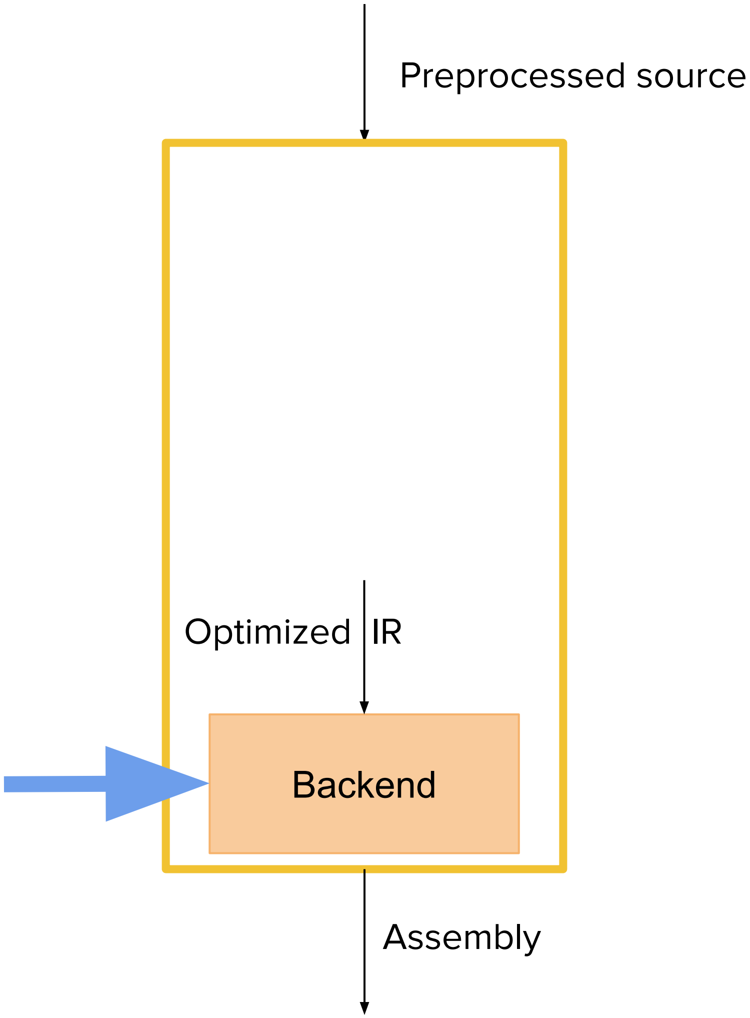
Dead Store Elimination

```
def(E) - in = {}, out = {}

def(A) - in = {}, out = {A}

def(B) - in = {A}, out = {A,B}

use(B) - in = {A,B}, out = {A}

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

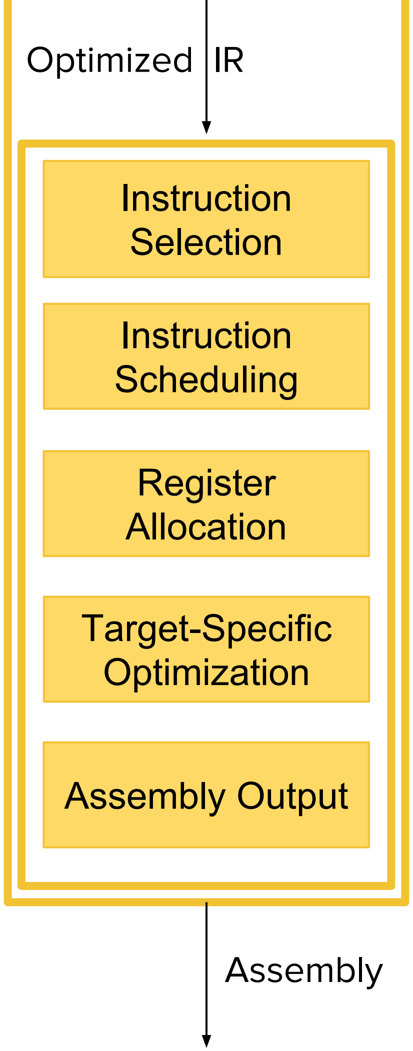
Dead Store Elimination

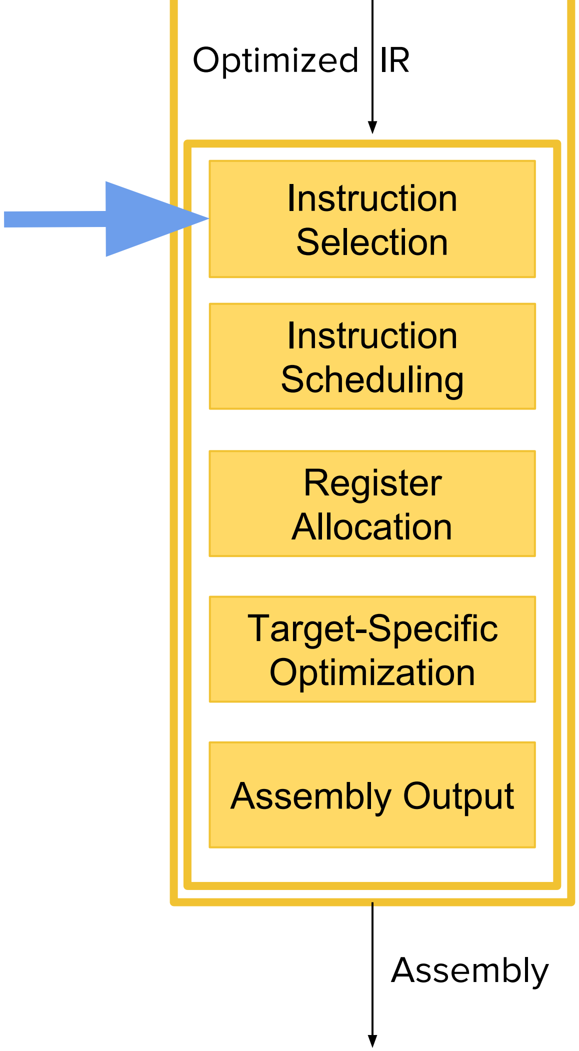
```
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```


Preprocessed source
Frontend
IR*
* IR = Intermediate Representation
Middle-end
Optimized IR
Backend
Assembly



Preprocessed source
Optimized IR
Backend
Assembly


Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly


Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly

Instruction Selection

```
int* v0, v1;

*v0 = *v0 + *v1;
```

Instruction Selection

```
r0 = load v0

r1 = load v1

r2 = add r0 r1

store v0 r2
```

Macro Expansion

```
r0 = load v0      ———— mov eax, [rsi]
r1 = load v1      ———— mov ebx, [rdi]
r2 = add r0 r1    ———— add eax, ebx
store v0 r2       ———— mov [rsi], eax
```

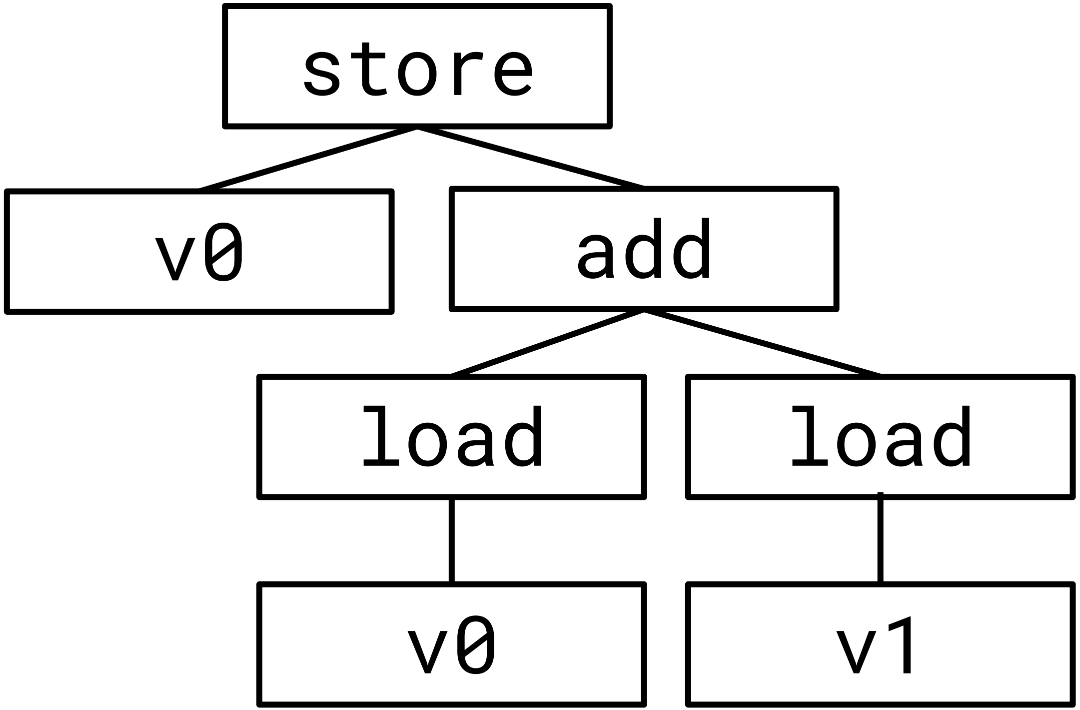
Macro Expansion

`r0 = load v0`	`mov eax, [rsi]`	`mov eax, [rsi]`
`r1 = load v1`	`mov ebx, [rdi]`	`add [rdi], eax`
`r2 = add r0 r1`	`add eax, ebx`	
`store v0 r2`	`mov [rsi], eax`	

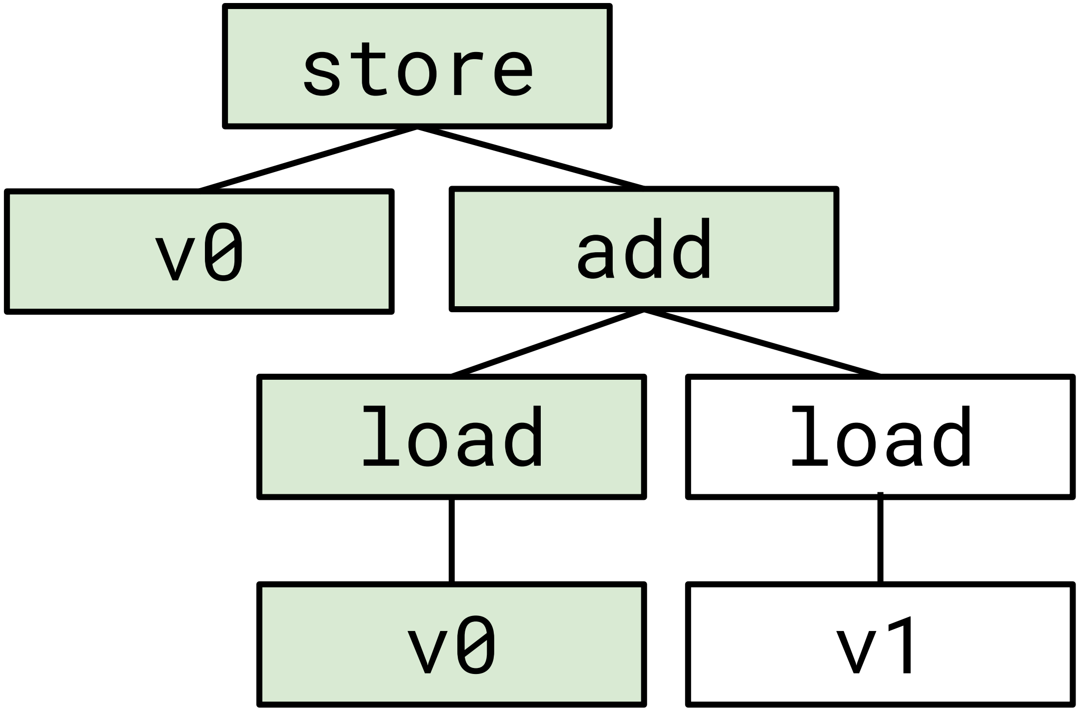
Selection DAG

```
r0 = load v0

r1 = load v1

r2 = add r0 r1

store v0 r2
```

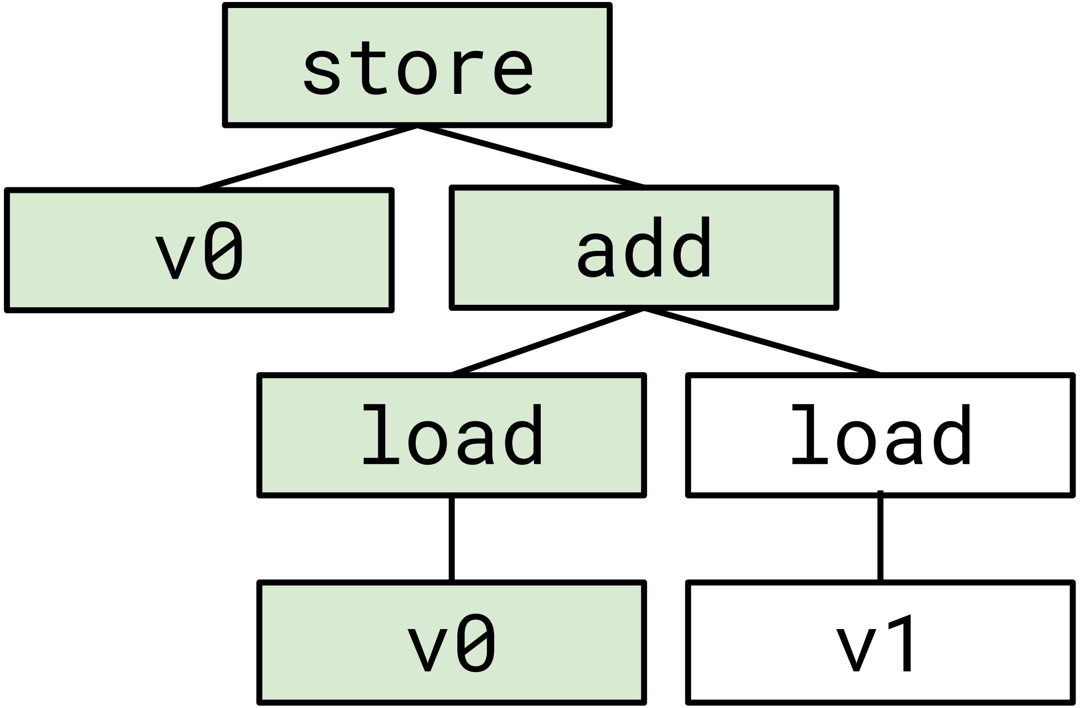
Selection DAG



Selection DAG

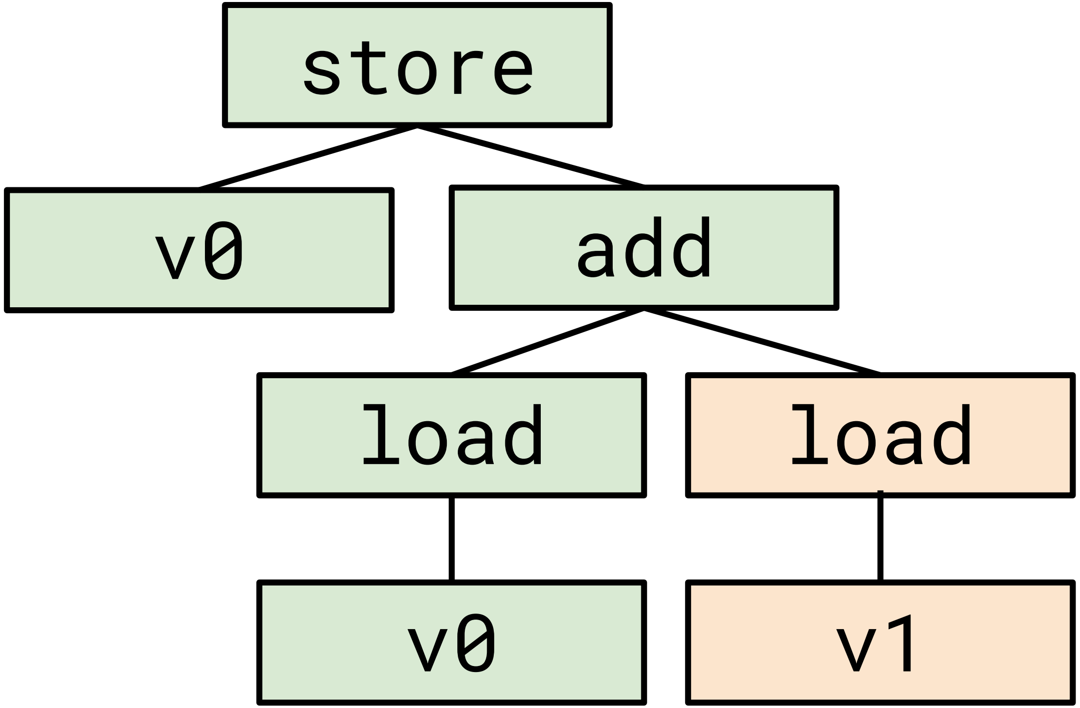


Selection DAG



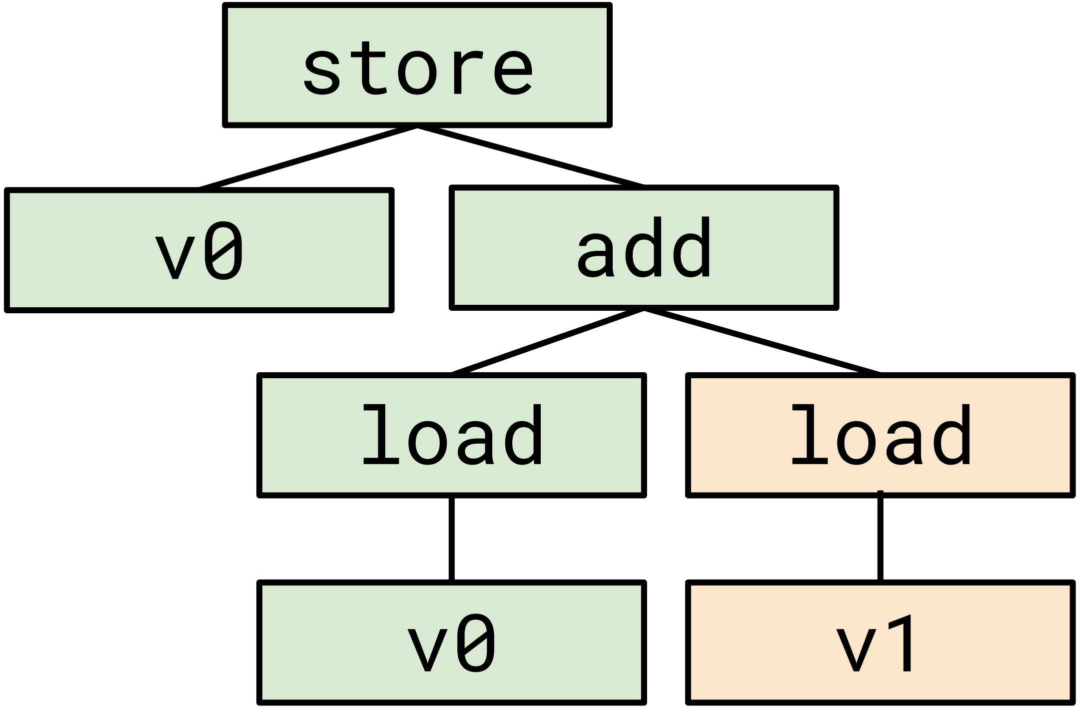

```
add [rdi], eax
```

Selection DAG

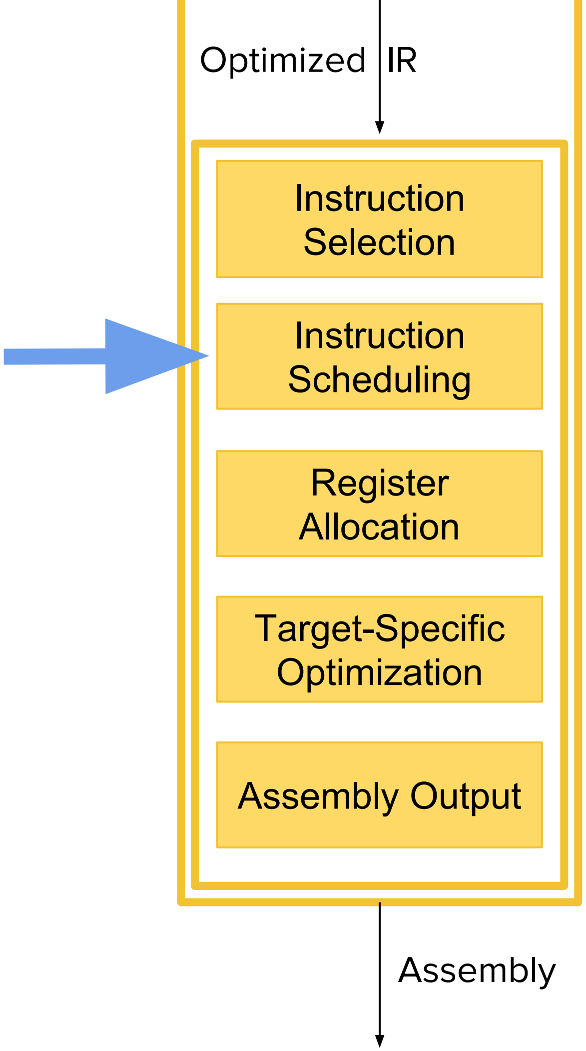



```
add [rdi], eax
```

Selection DAG




```
mov eax, [rsi]
add [rdi], eax
```

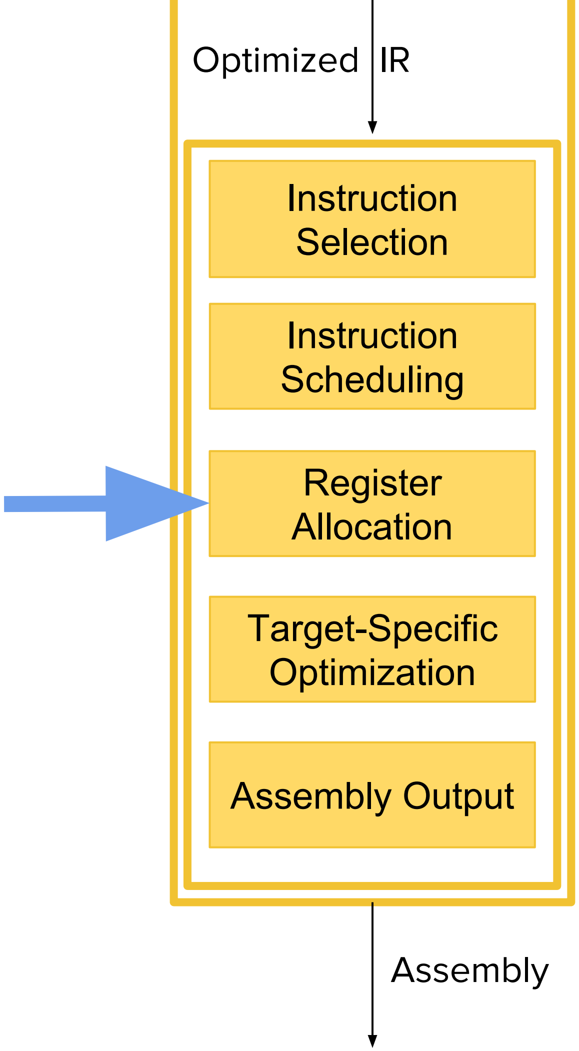

Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly

Instruction Scheduling

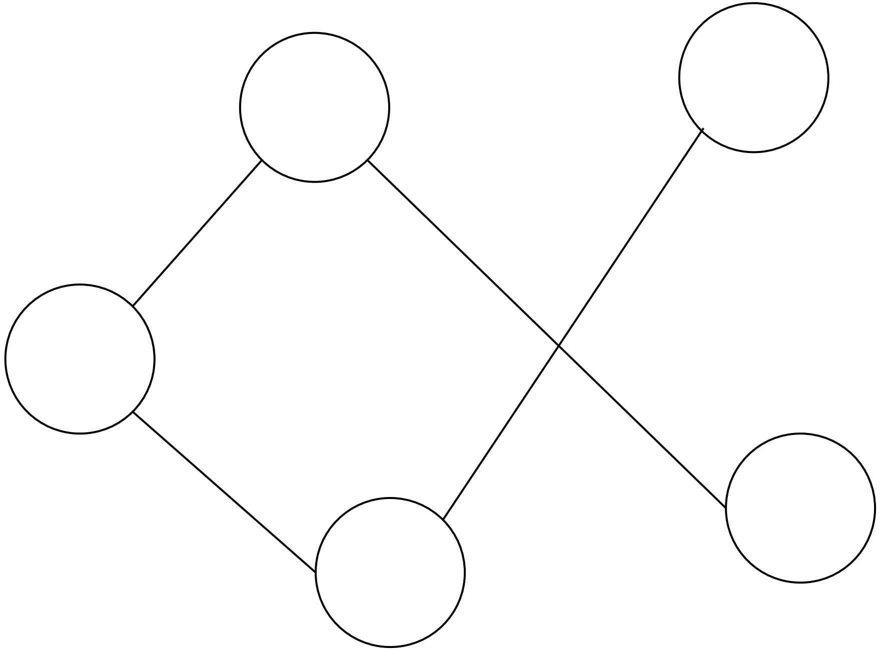
```
add r8, r9

add r8, r10

add r11, r12

add r11, r13
```

Instruction Scheduling

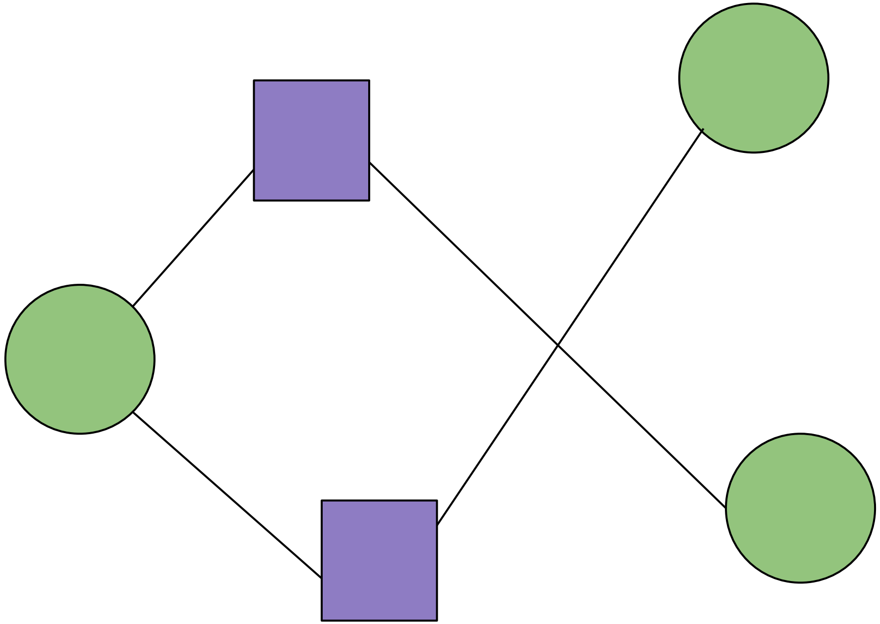
```
add r8, r9

add r11, r12

add r8, r10

add r11, r13
```


Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly

Graph Colouring



Graph Colouring

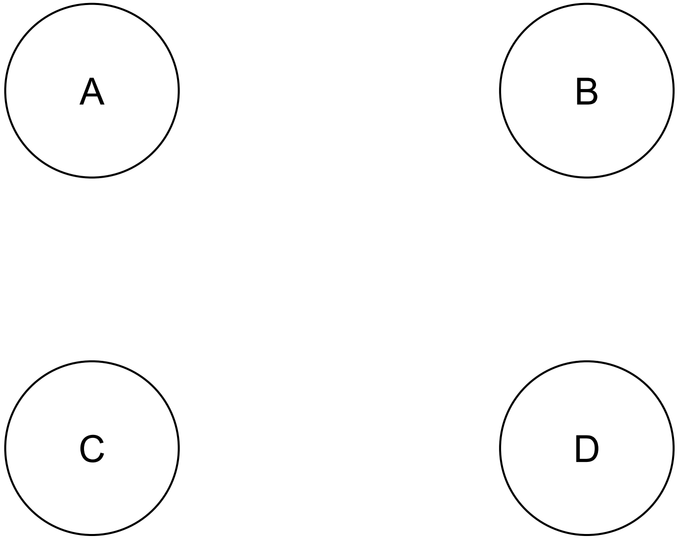


Interference Graph

```
def(E) - in = {}, out = {}

def(A) - in = {}, out = {A}

def(B) - in = {A}, out = {A,B}

use(B) - in = {A,B}, out = {A}

def(C) - in = {A}, out = {A,C}

use(A) - in = {A,C}, out = {C}

def(D) - in = {C}, out = {C,D}

use(D) - in = {C,D}, out = {C}

use(C) - in = {C}, out = {}
```

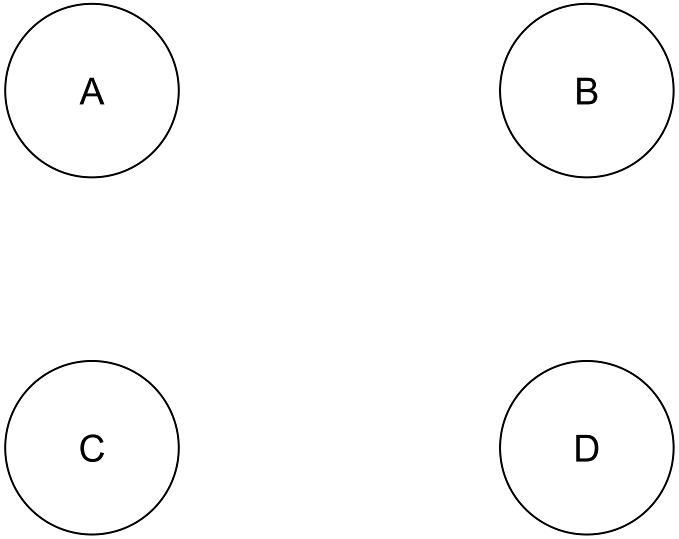
Interference Graph

```
def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```



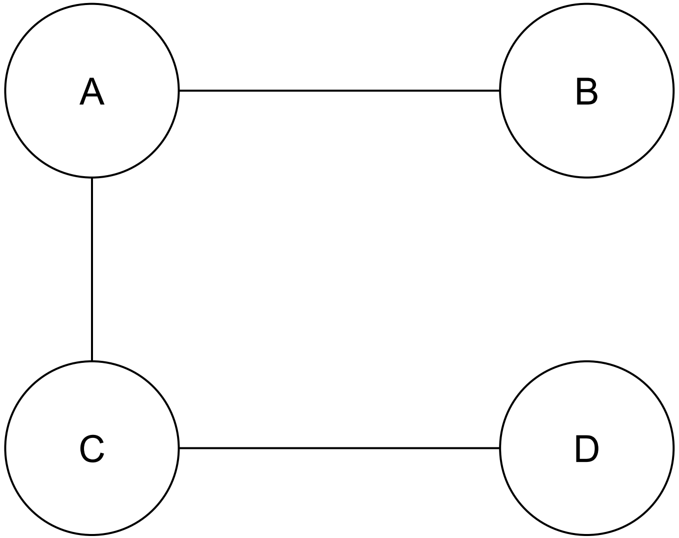
Interference Graph

```
def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```



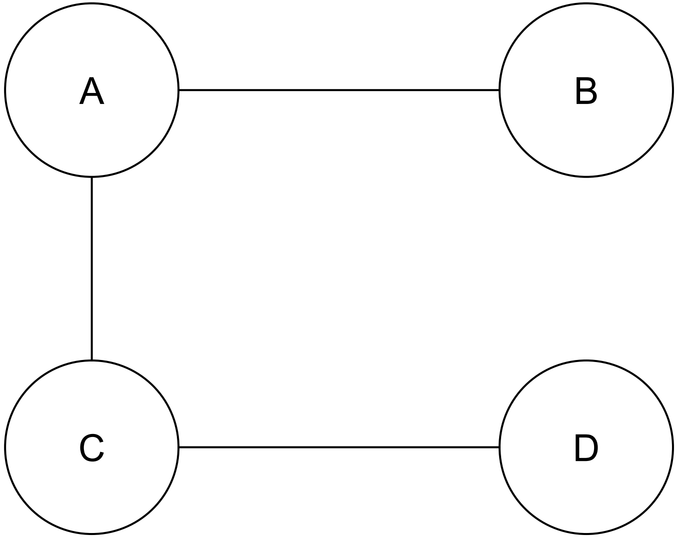
Interference Graph

```
def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
```



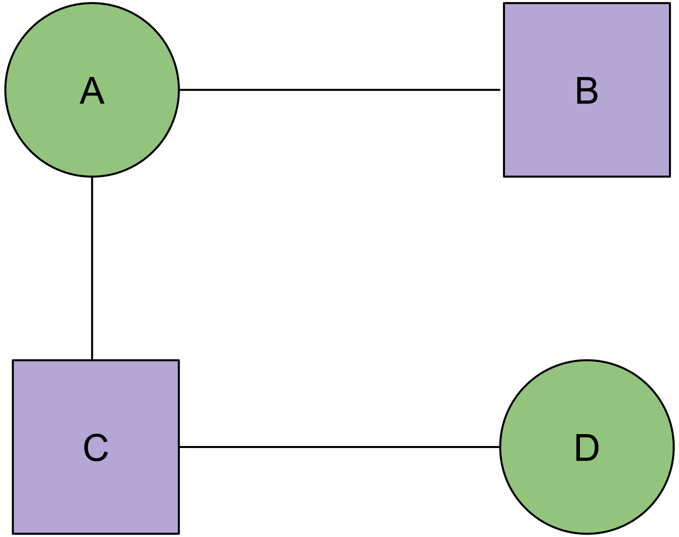
Interference Graph

```
E = …
A = …
B = …
… = f(B)
C = …
… = f(A)
D = …
… = f(D)
… = f(C)
```



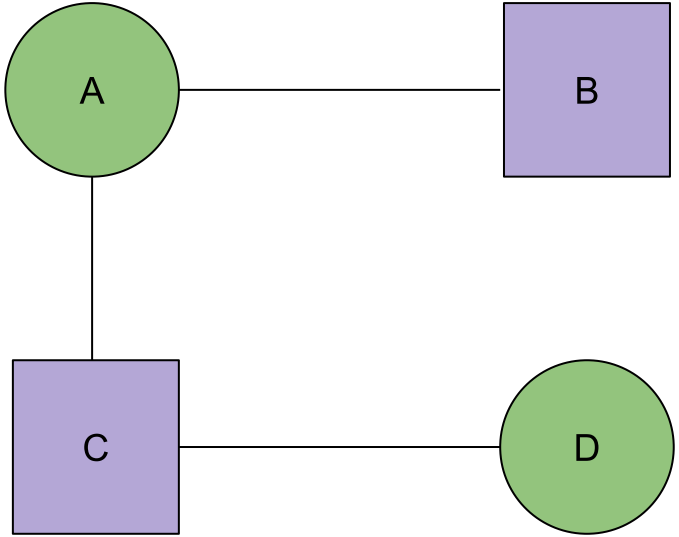
Interference Graph

```
E = …
A = …
B = …
… = f(B)
C = …
… = f(A)
D = …
… = f(D)
… = f(C)
```



Interference Graph

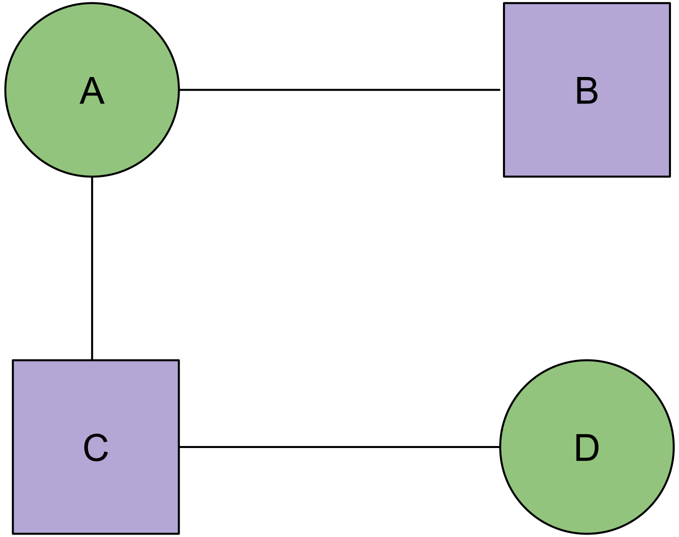
```
E = …
A = …
B = …
… = f(B)
C = …
… = f(A)
D = …
… = f(D)
… = f(C)
```




```
R0 = …
R1 = …
… = f(R1)
R1 = …
… = f(R0)
R0 = …
… = f(R0)
… = f(R1)
```

Interference Graph

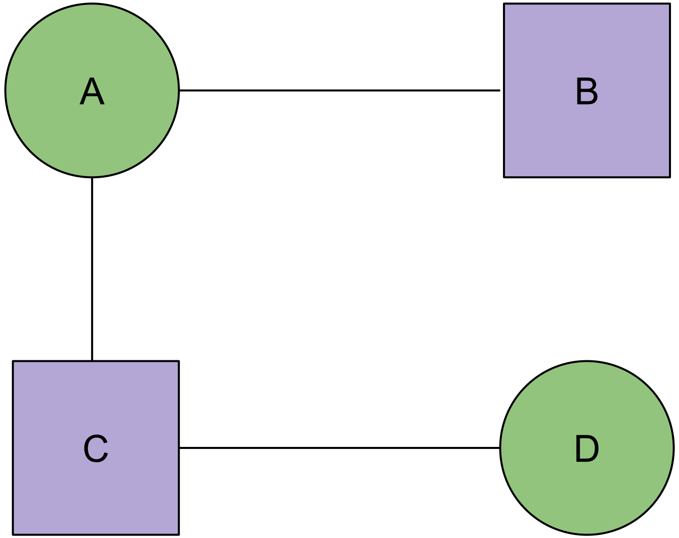
```
E = …
A = …
B = …
… = f(B)
C = …
… = f(A)
D = …
… = f(D)
… = f(C)
```




```
R0 = …
R1 = …
… = f(R1)
R1 = …
… = f(R0)
R0 = …
… = f(R0)
… = f(R1)
```

Interference Graph

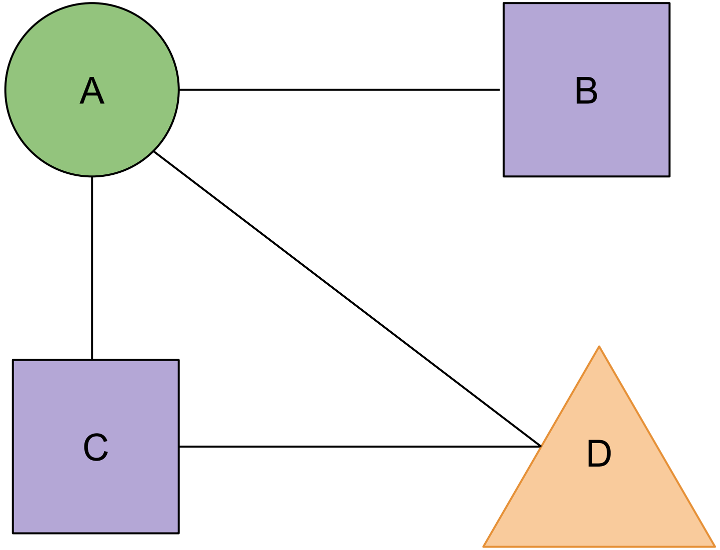
```
E = …
A = …
B = …
… = f(B)
C = …
D = …
… = f(A)
… = f(D)
… = f(C)
```

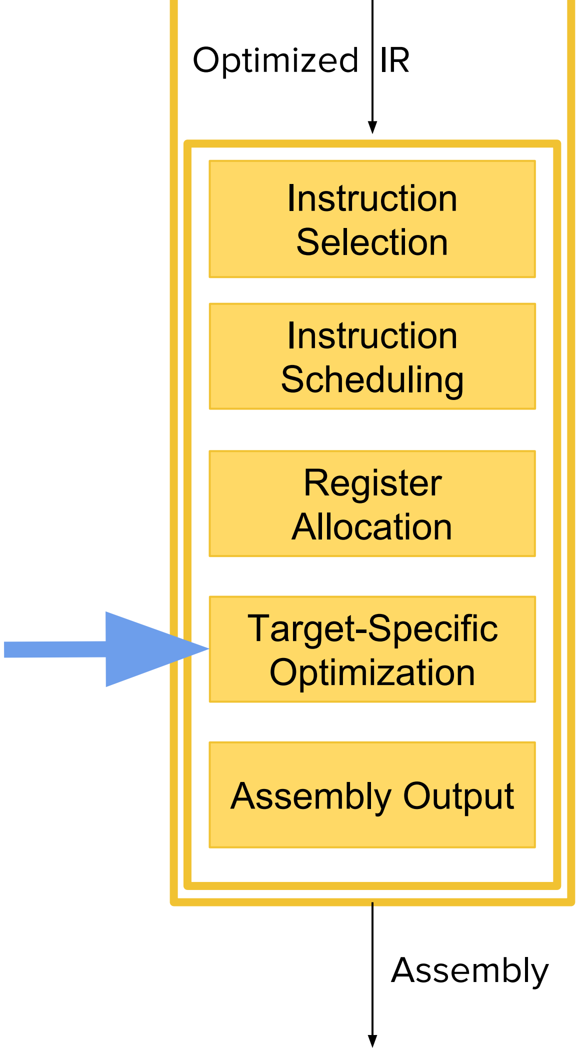



```
R0 = …
R1 = …
… = f(R1)
R1 = …
… = f(R0)
R0 = …
… = f(R0)
… = f(R1)
```

Interference Graph

```
E = …
A = …
B = …
… = f(B)
C = …
D = …
… = f(A)
… = f(D)
… = f(C)
```



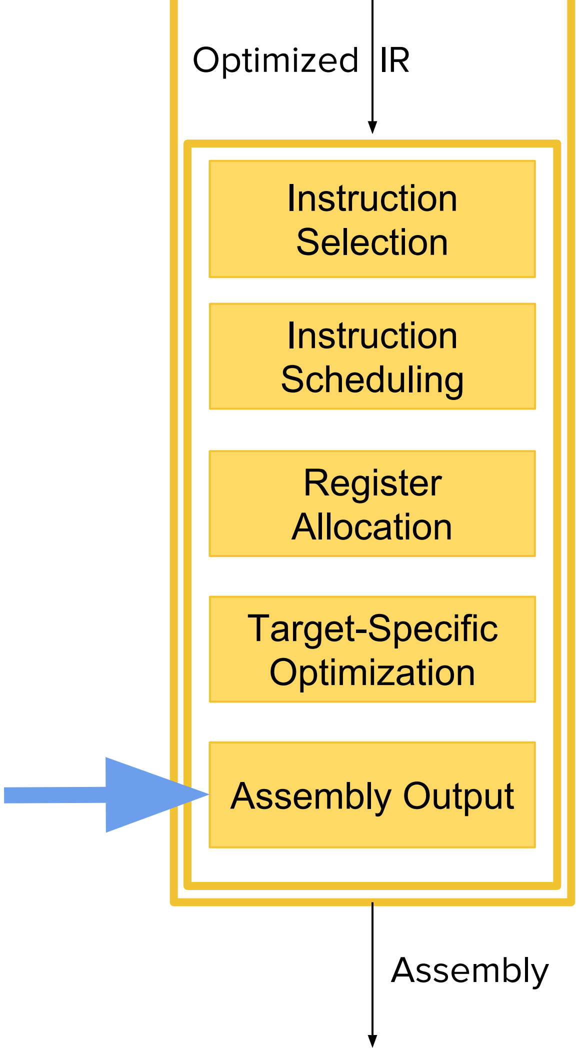

Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly

```
mov eax,0
```

```
mov  eax,0
b800000000
```

```
xor  rax,rax
4831c0
```

```
xor  eax,eax
31c0
```


Optimized IR
Instruction Selection
Instruction Scheduling
Register Allocation
Target-Specific Optimization
Assembly Output
Assembly

```
main:
        set up stack frame

        r0 = STRING(Hello world!)

        r1 = call puts(r0)

        remove stack frame
        return 0
```

```
hello:
        .string "Hello World!"
main:
        set up stack frame

        r0 = OFFSET FLAT :hello

        r1 = call puts(r0)

        remove stack frame
        return 0
```

```
hello:
        .string “Hello World!”
main:
        set up stack frame

        mov rdi, OFFSET FLAT :hello

        r1 = call puts (implicit: rdi)

        remove stack frame
        return 0
```

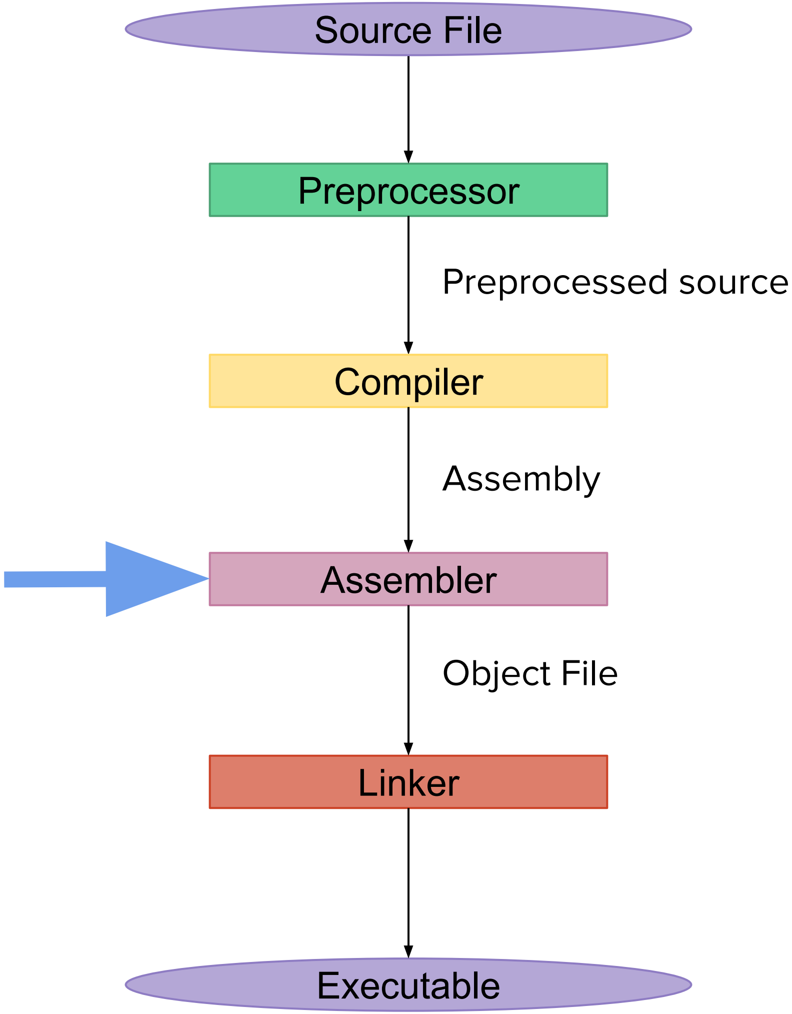
```
hello:
        .string "Hello World!"
main:
        set up stack frame

        mov rdi, OFFSET FLAT :hello

        call puts

        remove stack frame
        return 0
```

```
hello:
        .string “Hello World!”
main:
        sub rsp, 8

        mov rdi, OFFSET FLAT :hello

        call puts

        add rsp, 8
        return 0
```

```
hello:
        .string "Hello World!"
main:
        sub rsp, 8

        mov rdi, OFFSET FLAT :hello

        call puts

        add rsp, 8
        xor eax, eax
        ret
```

```
hello:
        .string “Hello World!”
main:
        sub rsp, 8
        mov rdi, OFFSET FLAT :hello
        call puts
        add rsp, 8
        xor eax, eax
        ret
```

https://godbolt.org/z/AejcvA


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

```
hello:
        .string "Hello World!"
main:
        sub rsp, 8
        mov rdi, OFFSET FLAT :hello
        call puts
        add rsp, 8
        xor eax, eax
        ret
```

```
48 65 6c 6c 6f 20 57 6f
72 6c 64 21 00
```

https://godbolt.org/z/AejcvA

```
hello:
        .string "Hello World!"
main:
        sub rsp, 8
        mov rdi, OFFSET FLAT :hello
        call puts
        add rsp, 8
        xor eax, eax
        ret
```

https://godbolt.org/z/AejcvA

```
48 65 6c 6c 6f 20 57 6f
72 6c 64 21 00

48 83 ec 08
bf ?? ?? ?? ??
e8 ?? ?? ?? ??
48 83 c4 08
31 c0
cb
```

hello:	48 65 6c 6c 6f 20 57 6f
.string “Hello World!”	72 6c 64 21 00
main:	
sub rsp, 8	48 83 ec 08
mov rdi, OFFSET FLAT :hello	bf 00 00 00 00 (hello)
call puts	e8 00 00 00 00 (puts)
add rsp, 8	48 83 c4 08
xor eax, eax	31 c0
ret	cb

https://godbolt.org/z/AejcvA

```
48 65 6c 6c 6f 20 57 6f
72 6c 64 21 00

48 83 ec 08
bf 00 00 00 00
e8 00 00 00 00
48 83 c4 08
31 c0
cb
```

```
48 65 6c 6c 6f 20 57 6f
72 6c 64 21 00

48 83 ec 08
bf 00 00 00 00        ← Relocation
e8 00 00 00 00
48 83 c4 08
31 c0
cb
```

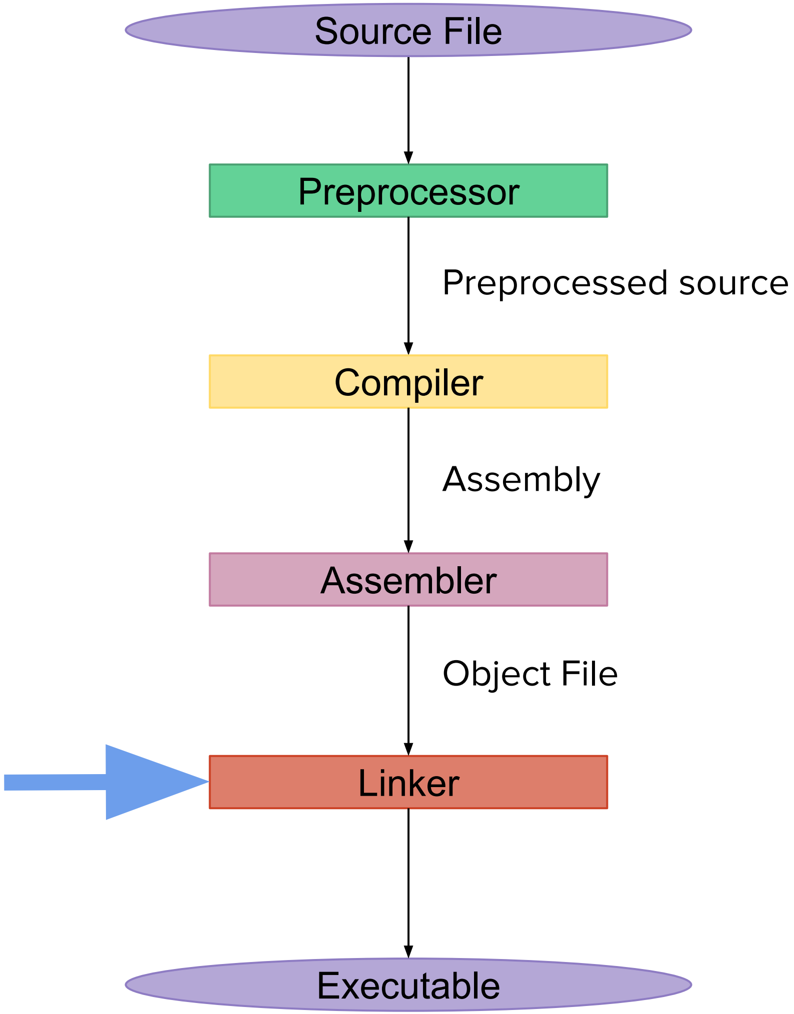
```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83
c4 08 31 c0 cb
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb

symtab:
   hello: .rodata + 0
   main: .text + 0
   puts: ?
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb

symtab:
   hello: .rodata + 0
   main: .text + 0
   puts: ?

.rel.text:
   +5: 4 bytes signed offset, point to hello
   +10: 4 bytes signed offset, point to puts
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb

symtab:
   hello: .rodata + 0
   main: .text + 0
   puts: ?

.rel.text:
   +5: R_X86_64_PC32, point to hello
   +10: R_X86_64_PC32, point to puts
```


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb

symtab:
   hello: .rodata + 0
   main: .text + 0
   puts: ?

.rel.text:
   +5: R_X86_64_PC32, point to hello
   +10: R_X86_64_PC32, point to puts
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb

(puts.o’s) .text: 41 55 41 54 49 89 fc 55 53 48
83 ec 08 …

(puts.o) symtab: puts: .text + 0x0
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb
```

(puts.o's) .text: `41 55 41 54 49 89 fc 55 53 48`
`83 ec 08 …`

```
(puts.o) symtab: puts: .text + 0x0
```

```
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00
00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55
53 48 83 ec 08 …

symtab:
   hello: .rodata + 0
   main: .text + 0
   puts: .text + 0x15
.rel.text:
   +5: R_X86_64_PC32, point to hello
   +10: R_X86_64_PC32, point to puts
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   hello: 8048000 + 0x0
   main: 804800d + 0
   puts: 804800d + 0x15
.rel.text:
   +5: R_X86_64_PC32, point to hello
   +10: R_X86_64_PC32, point to puts
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   hello: 8048000
   main: 804800d
   puts: 8048022
.rel.text:
   8048012: R_X86_64_PC32, point to hello
   8048017: R_X86_64_PC32, point to puts
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   hello: 8048000
   main: 804800d
   puts: 8048022
.rel.text:
   8048012: R_X86_64_PC32, point to hello
   8048017: R_X86_64_PC32, point to puts
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   hello: 8048000
   main: 804800d
   puts: 8048022
.rel.text:
   8048012: R_X86_64_PC32, 0xffffffef
   8048017: R_X86_64_PC32, 0x0000000c
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   hello: 8048000
   main: 804800d
   puts: 8048022
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   main: 804800d

Program table:
   Load these bytes at 0x8048000
Elf header:
   Entry point is 0x804800d
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …
```

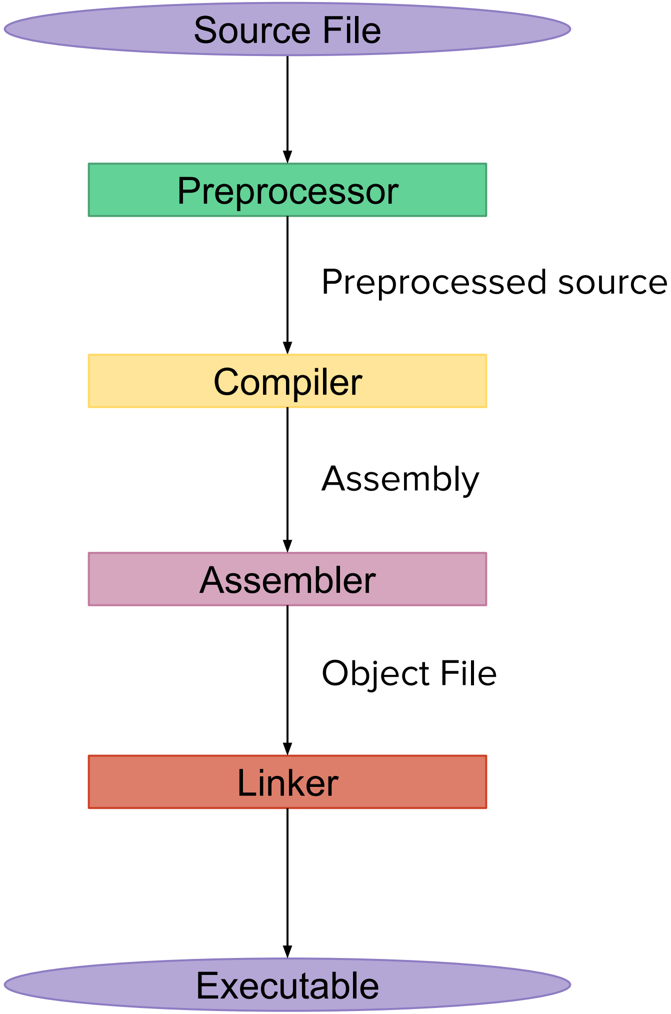
```
Program table:
   Load these bytes at 0x8048000
Elf header:
   Entry point is 0x804800d
```

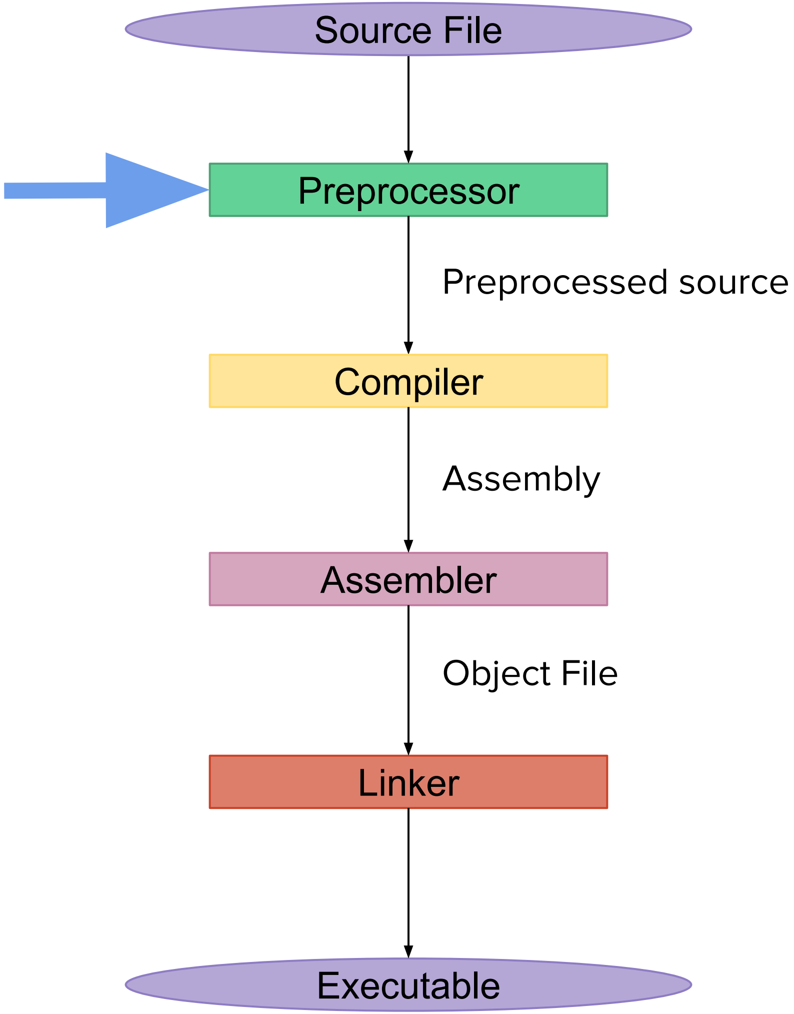
./hello

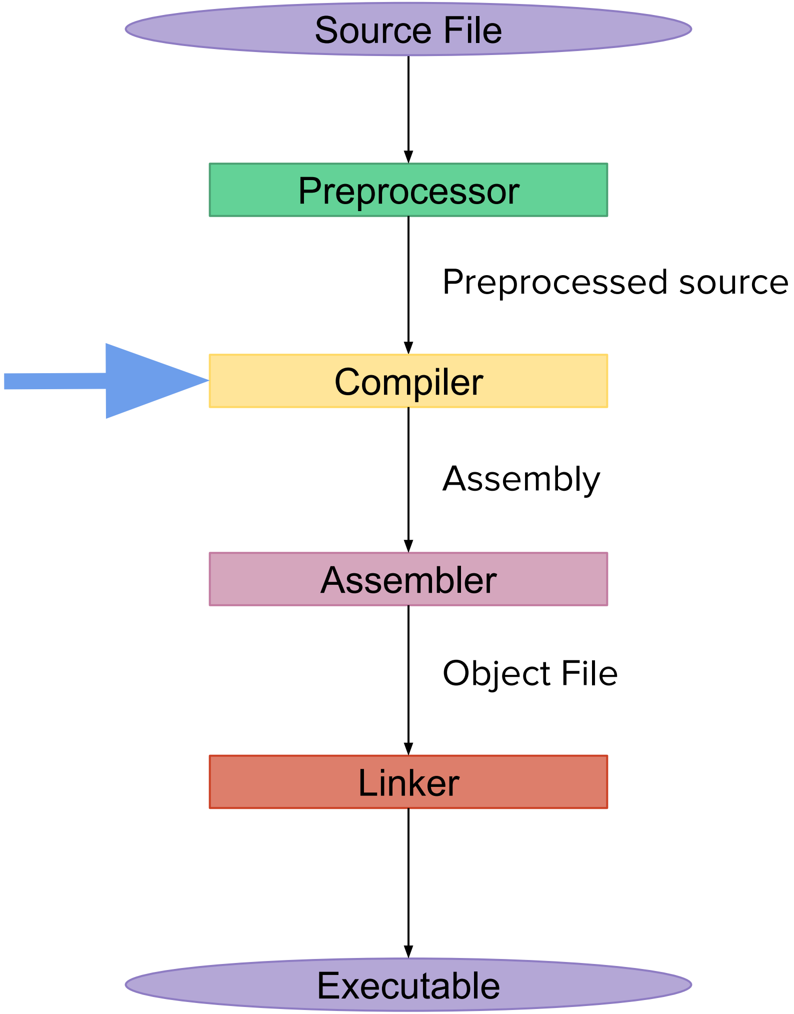
Hello world!

Hello World in C++

```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!\n";
}
```


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

```
a<b<x>>y;
```

```
a<b    <    x>>y;
```

```
a<    b<x>    >y;
```

```
a<b<x>>        y;
```

```
std::cout
```

```
0 | class std::basic_ostream<char>
0 |   (basic_ostream vtable pointer)
8 |   class std::basic_ios<char> (virtual base)
8 |       class std::ios_base (primary base)
8 |       (ios_base vtable pointer)
16 |     std::streamsize _M_precision
24 |     std::streamsize _M_width
32 |     std::ios_base::fmtflags _M_flags
36 |     std::ios_base::iostate _M_exception
40 |     std::ios_base::iostate _M_streambuf_state
48 |     struct std::ios_base::_Callback_list * _M_callbacks
56 |     struct std::ios_base::_Words _M_word_zero
56 |               void * _M_pword
64 |               long _M_iword
72 |     struct std::ios_base::_Words [8] _M_local_word
200 |             int _M_word_size
208 |             struct std::ios_base::_Words * _M_word
216 |             class std::locale _M_ios_locale
216 |             class std::locale::_Impl * _M_impl
224 |   basic_ostream<char, struct std::char_traits<char> > * _M_tie
232 |   std::basic_ios<char, struct std::char_traits<char> >::char_type _M_fill
233 |   _Bool _M_fill_init
240 |   basic_streambuf<char, struct std::char_traits<char> > * _M_streambuf
248 |   const std::basic_ios<char, struct std::char_traits<char> >::__ctype_type * _M_ctype
256 |   const std::basic_ios<char, struct std::char_traits<char> >::__num_put_type * _M_num_put
264 |   const std::basic_ios<char, struct std::char_traits<char> >::__num_get_type * _M_num_get
```

Use fmtlib

Use fmtlib
(P0645, C++20)

```
struct weird_int {
  int i;
  virtual void get_value(){}
};
```

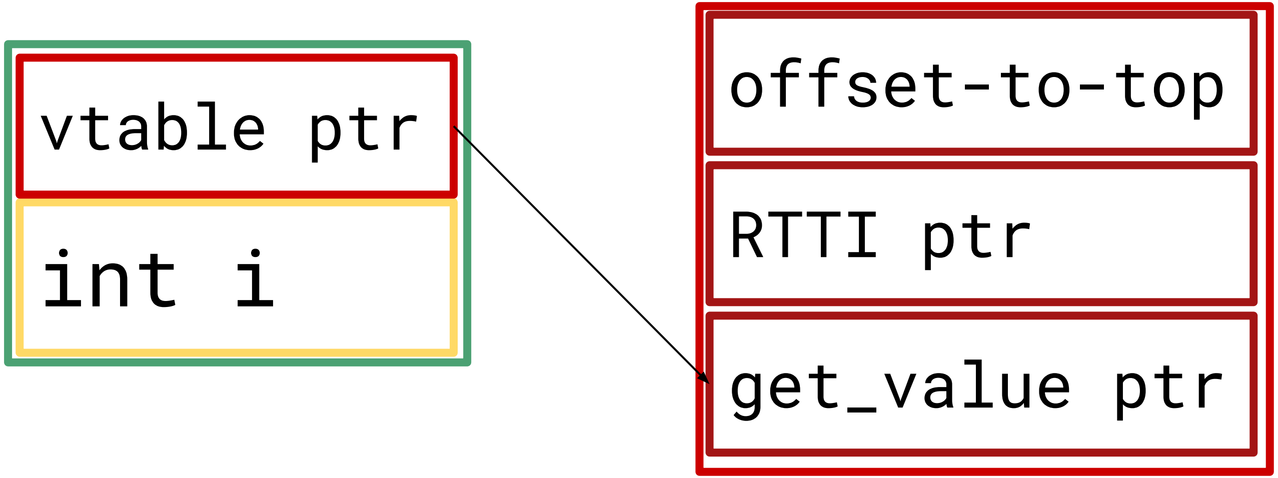
```
struct weird_int {
  int i;
  virtual void get_value(){}
};
```

```
0 | struct weird_int
0 |   (weird_int vtable pointer)
8 |   int i
```

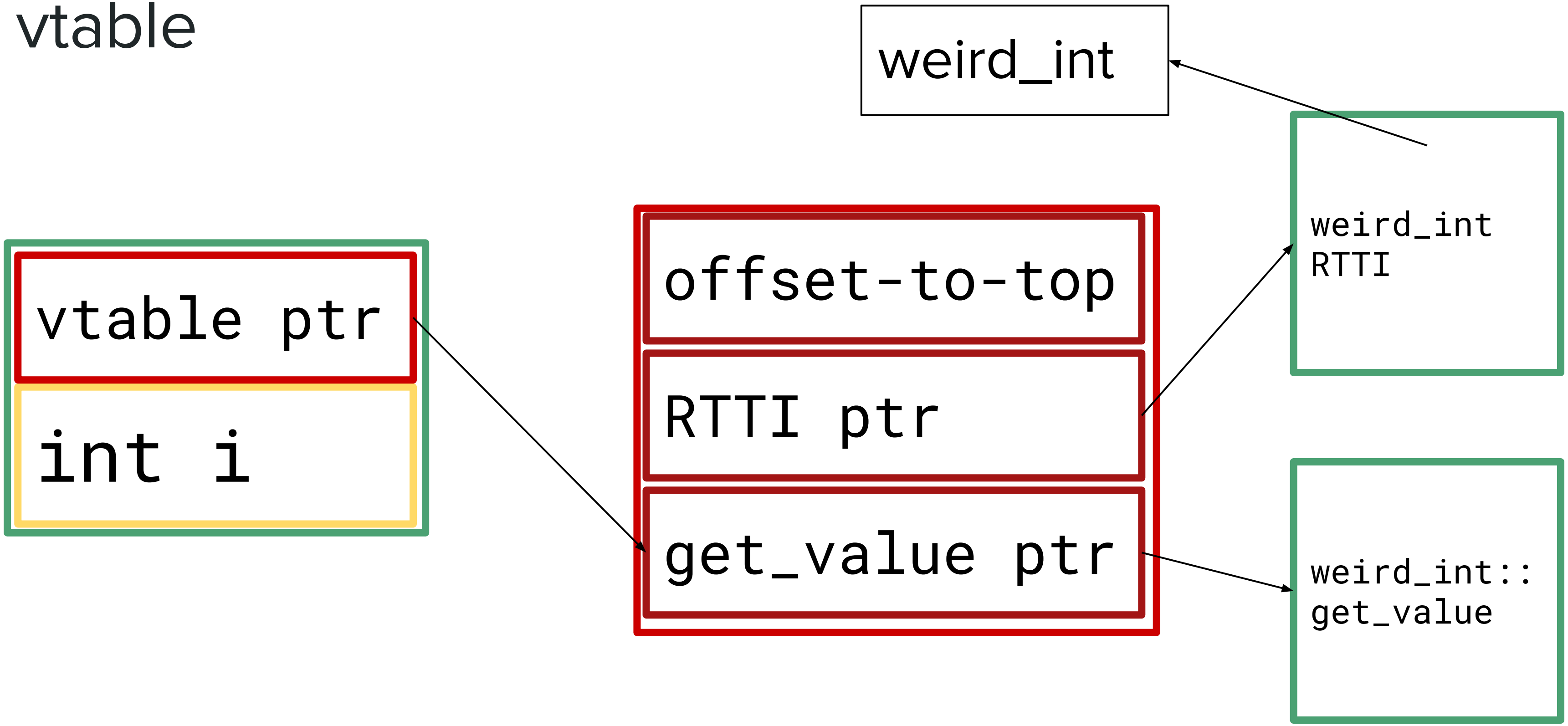
vtable

```
vtable ptr
int i
```

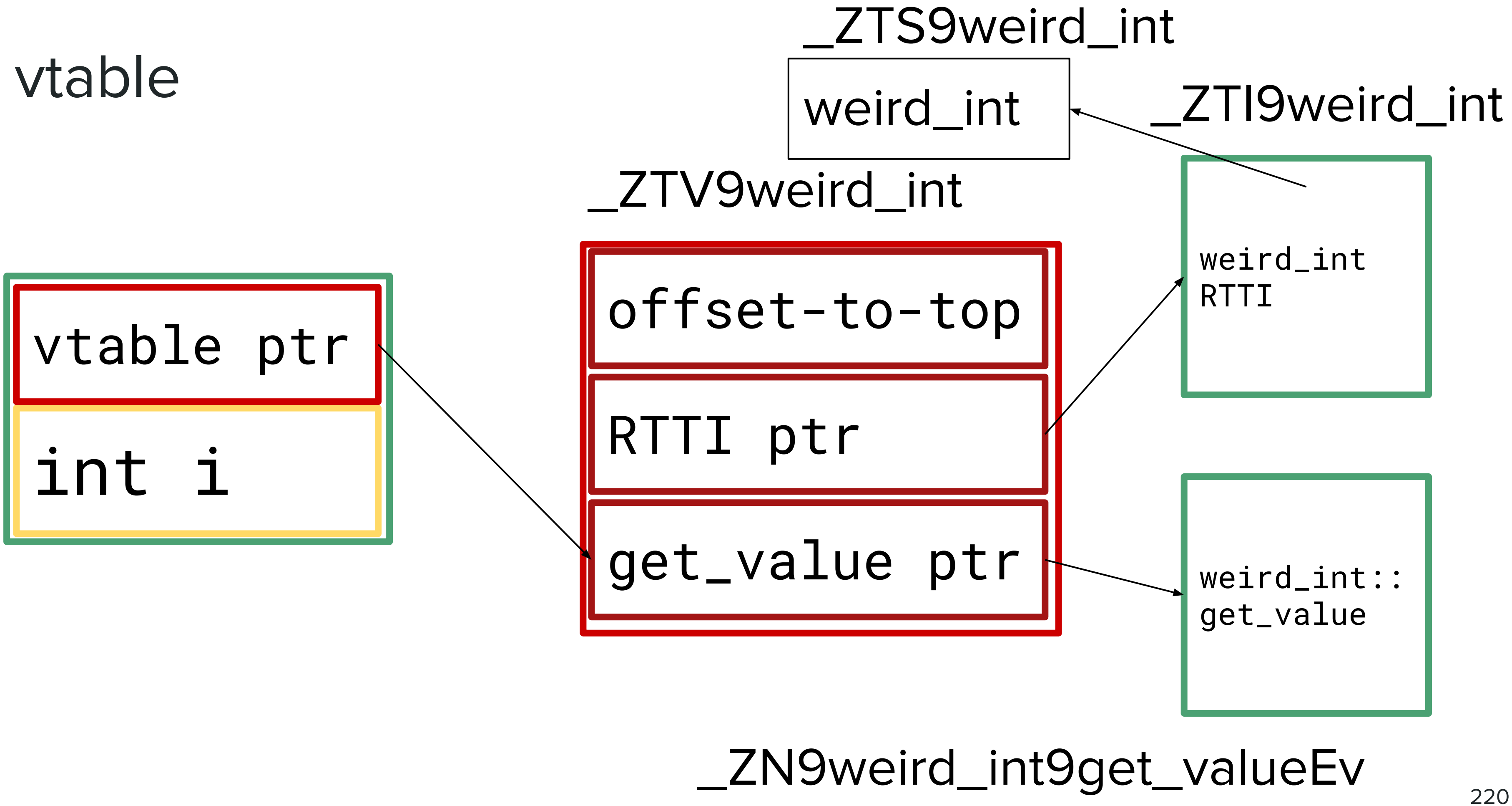
vtable



vtable


weird_int
vtable ptr
int i
offset-to-top
RTTI ptr
get_value ptr
weird_int
RTTI
weird_int::
get_value

vtable

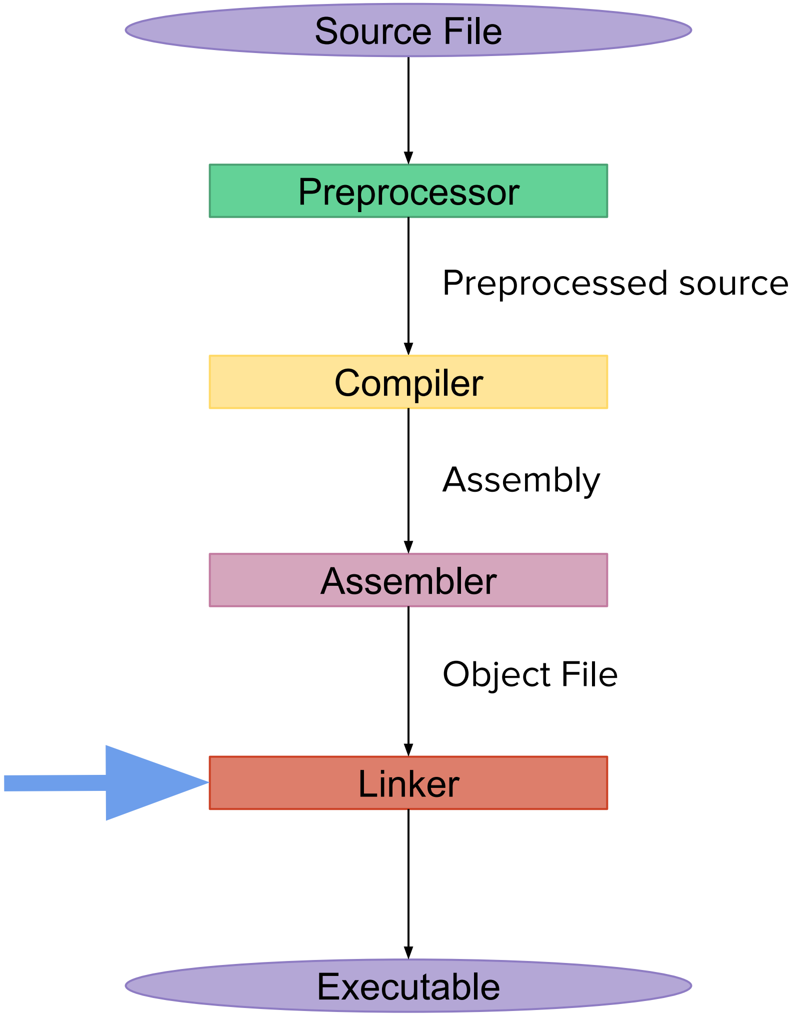

_ZTS9weird_int
weird_int
_ZTI9weird_int
_ZTV9weird_int
vtable ptr
int i
offset-to-top
RTTI ptr
get_value ptr
weird_int
RTTI
weird_int::
get_value
_ZN9weird_int9get_valueEv

```
_ZTV9weird_int
  dq 0
  dq _ZTI9weird_int
  dq _ZN9weird_int9get_valueEv

_ZTI9weird_int
  dq _ZTVN10__cxxabiv117
           __class_type_infoE
  dq _ZTS9weird_int
```

```
_ZTS9weird_int
  .string “weird_int”

_ZN9weird_int9get_valueEv:
  mov eax, DWORD PTR [edi+8]
  ret
```

Object Lifetimes

- Static
  - Global scope
  - Function scope
- Thread local
- Dynamic (heap)
- Automatic (stack)


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable

ELF file types

Object File - A part of your program in bits (sections)

Executable - Your whole program as a “whole”

Shared library - Shared bits between programs

Core dump - Your whole program as a crash dump

Shared library

- Contains a ton of functions you can import
- Read-only and code are shared between all processes
- Allows security patches without full recompile
- Origin of DLL Hell

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

puts

puts@plt (points to puts)

main (points to puts@plt)

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

puts	***puts***
puts@plt (points to puts)	***puts@plt***
main (points to puts@plt)	***fputs*** (points to ***puts@plt***)

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

puts

puts@plt (points to puts)

main (points to puts@plt)

puts

puts@plt (points to puts)

fputs (points to ***puts@plt***)

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

puts

puts@plt (points to ***puts***)

main (points to puts@plt)

puts@plt (points to ***puts***)

fputs (points to ***puts@plt***)

Shared library

- What if symbol names collide?
  - Use from first loaded executable/library
- PLT (or equivalent) for functions
  - Procedure Linkage Table
- GOT (or equivalent) for objects
  - Global Offset Table
- Often, the PLT uses the GOT.

Global initializer(s)

```
struct T {
  T() {}
  ~T() {}
};
T t;
```

Global initializer(s)

```
_GLOBAL__sub_I_t:
    mov edi,0x601044                (address of t)
    call 400626 <T::T()>            (construct T)

    mov edx,0x601038                (address of __dso_handle)
    mov esi,0x601044                (address of t)
    mov edi,0x4005f2                (address of T::~T)
    jmp 4004e0 <__cxa_atexit@plt>
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   main: 804800d

Program table:
   Load these bytes at 0x8048000
Elf header:
   Entry point is 0x804800d
```

```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec
08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31
c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 …

symtab:
   main: 804800d
   _start: 8048432

Program table:
   Load these bytes at 0x8048000
Elf header:
   Entry point is 0x8048432
```

- Crtbegin.o
  - __dso_handle in .data
  - __frame_dummy_init_array_entry in .init_array
- Crtend.o
- Linker trick
  - creates an array of global initializers
  - creates global variables pointing to the start and end
- Link in _start to call these global initializers

```
_start:
    xor    %ebp,%ebp
    pop    %rsi
    mov    %rsp,%r9
    and    $0xfffffffffffffff0,%rsp
    push   %rax
    push   %rsp
    lea    0x16a(%rip),%r8     # 11d0 <__libc_csu_fini>
    lea    0x103(%rip),%rcx    # 1170 <__libc_csu_init>
    lea    0xc3(%rip),%rdi     # 1137 <main>
    callq  *0x2f66(%rip)    # 3fe0 <__libc_start_main>
    hlt
```

Loader

ELF Loader

- Loads your ELF executable from disk
  - Loads any required shared libraries first
  - Causes globals to be initialized for shared libraries

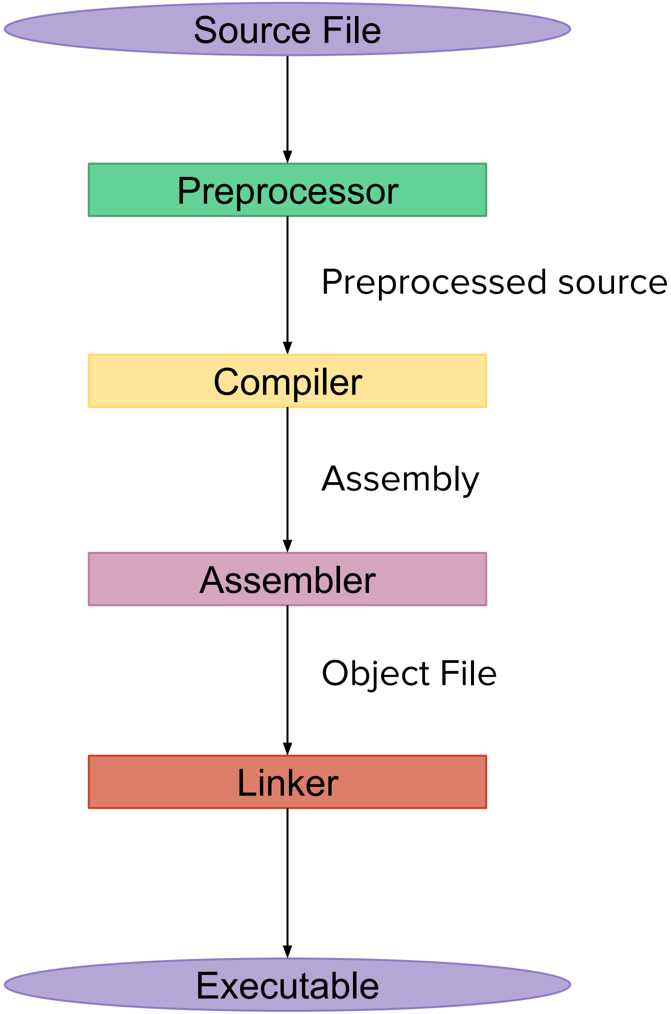
ELF Loader

- Loads your ELF executable from disk
  - Loads any required shared libraries first
  - Causes globals to be initialized for shared libraries
- Handles “where is our `puts` now”
- Handles __cxa_atexit and __dso_handle complexities
- PLT entries are all empty, pointing to the loader

./hello

Hello world!

Summary / conclusion


Source File
Preprocessor
Preprocessed source
Compiler
Assembly
Assembler
Object File
Linker
Executable


C++
Magic

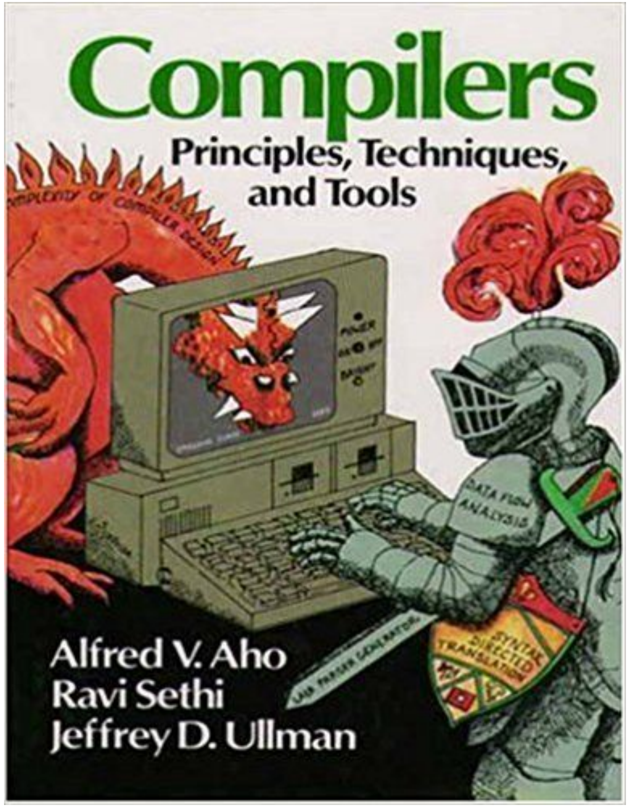
This was

Hello World from Scratch

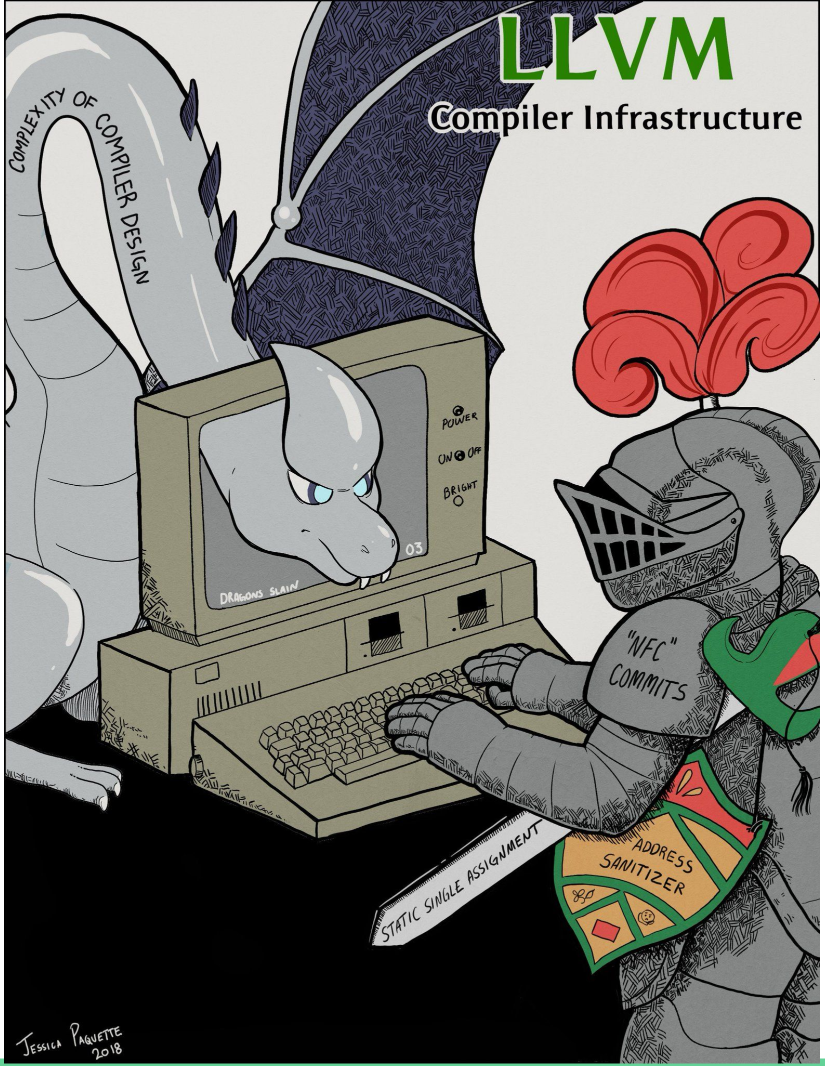
Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft

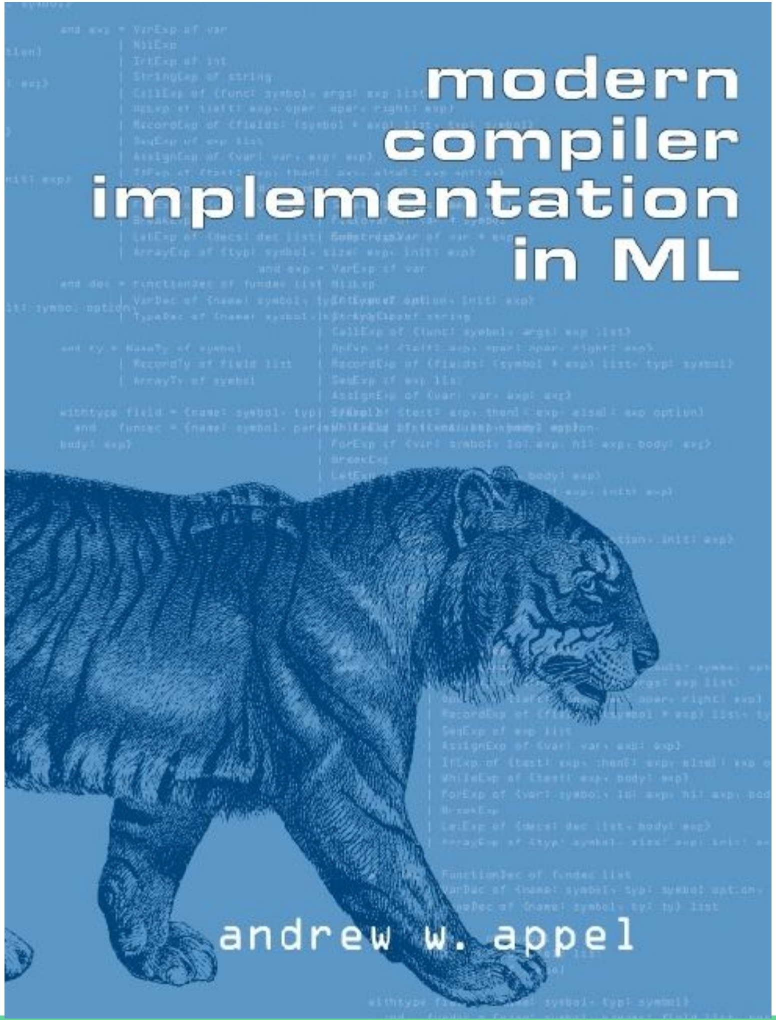
Dragon Book

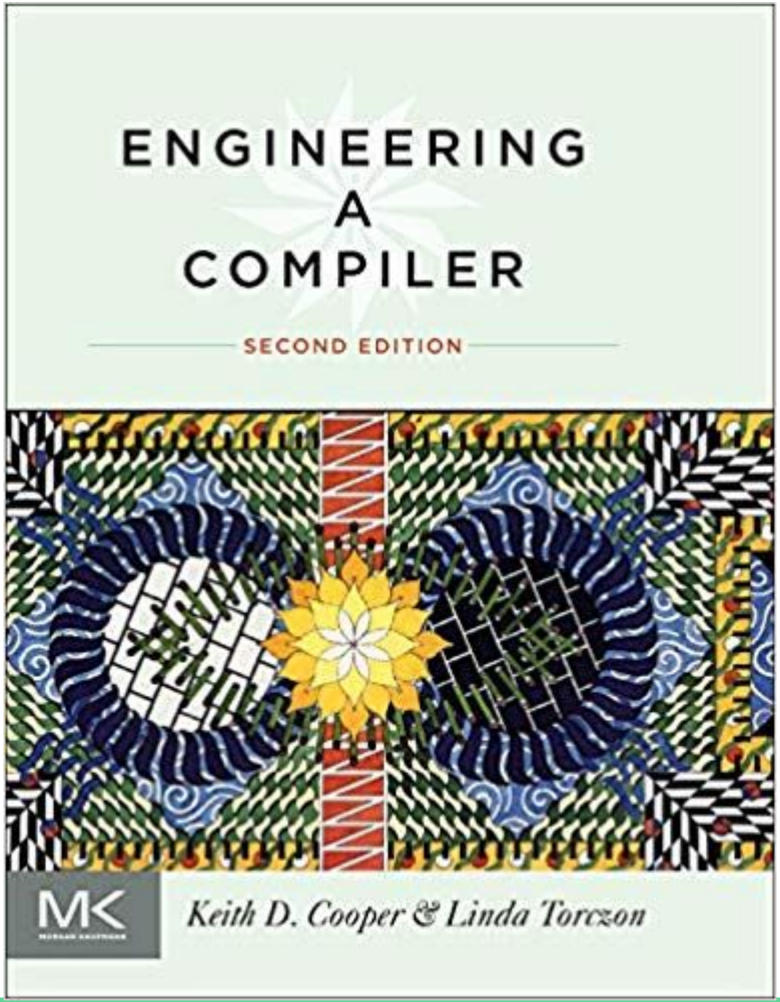

Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman
DATA FLOW ANALYSIS
SYNTAX DIRECTED TRANSLATION

Dragon Book


LLVM
Compiler Infrastructure
COMPLEXITY OF COMPILER DESIGN
POWER
ON OFF
BRIGHT
03
DRAGONS SLAIN
"NFC" COMMITS
ADDRESS SANITIZER
STATIC SINGLE ASSIGNMENT
JESSICA PAQUETTE 2018


Jessica Paquette
@barrelshifter Follows you


modern
compiler
implementation
in ML
andrew w. appel


ENGINEERING
A
COMPILER
SECOND EDITION
Keith D. Cooper & Linda Torczon

https://llvm.org/docs/tutorial/
https://llvm.org/docs/WritingAnLLVMBackend.html