From ASTs to LLVM

Dimitri Racordon (dimitri.racordon@unige.ch)

CoCoDo @ Everywhere
22nd Match, 2021

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES

From ASTs to LLVM

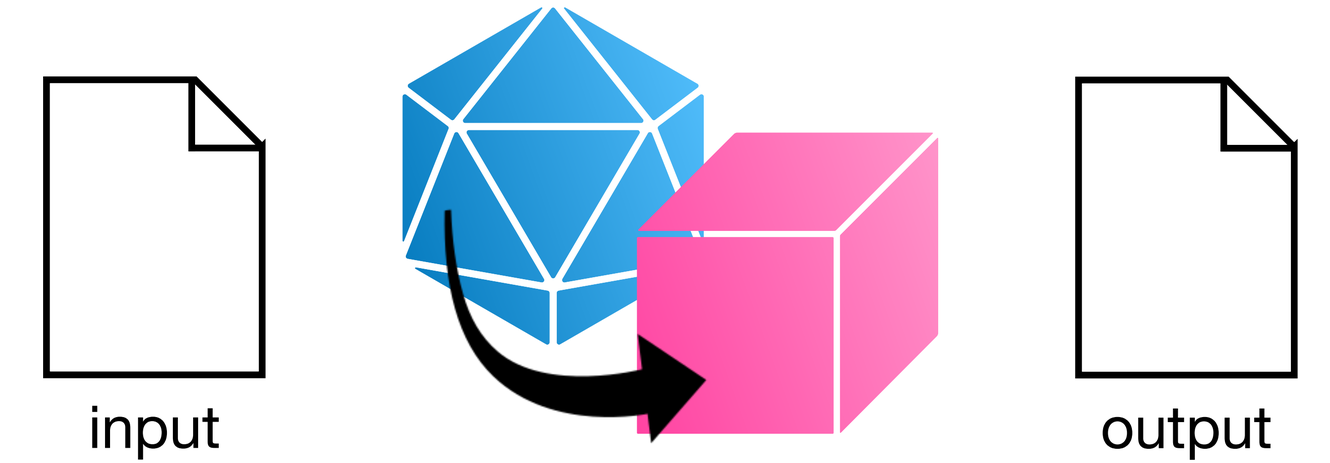
Agenda

- ASTs, code generation, optimizations, Oh My!
- LLVM in a nutshell
- Let's emit some IR
  - Simple arithmetics
  - Existential containers
  - Closures
- Final thoughts

Downloads!

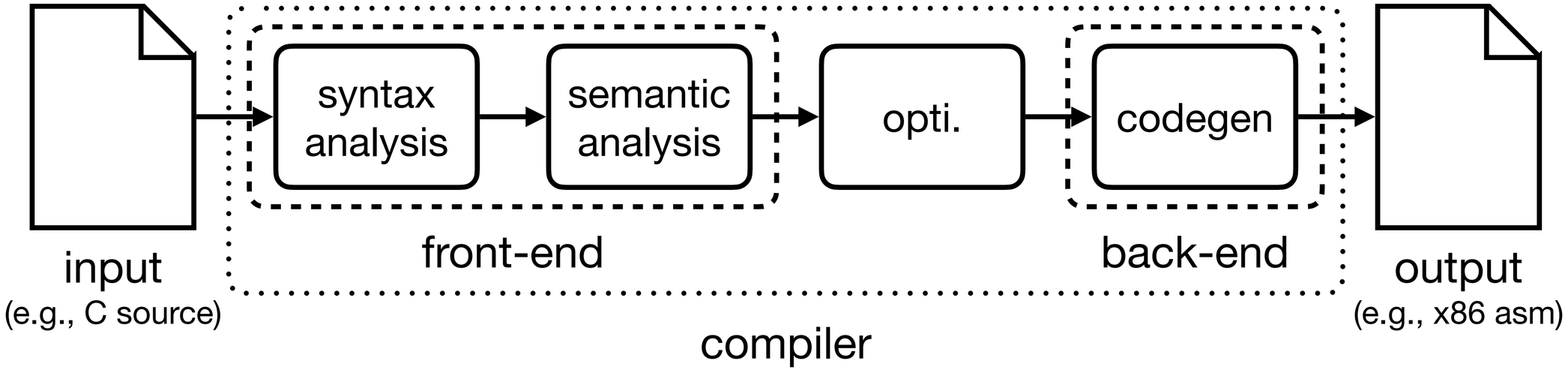
macOS Big Sur	Ubuntu/centOS	Others
 Xcode	 Swift	 virtualbox
 LLVM	 LLVM	
	 VS code	

What's a compiler?



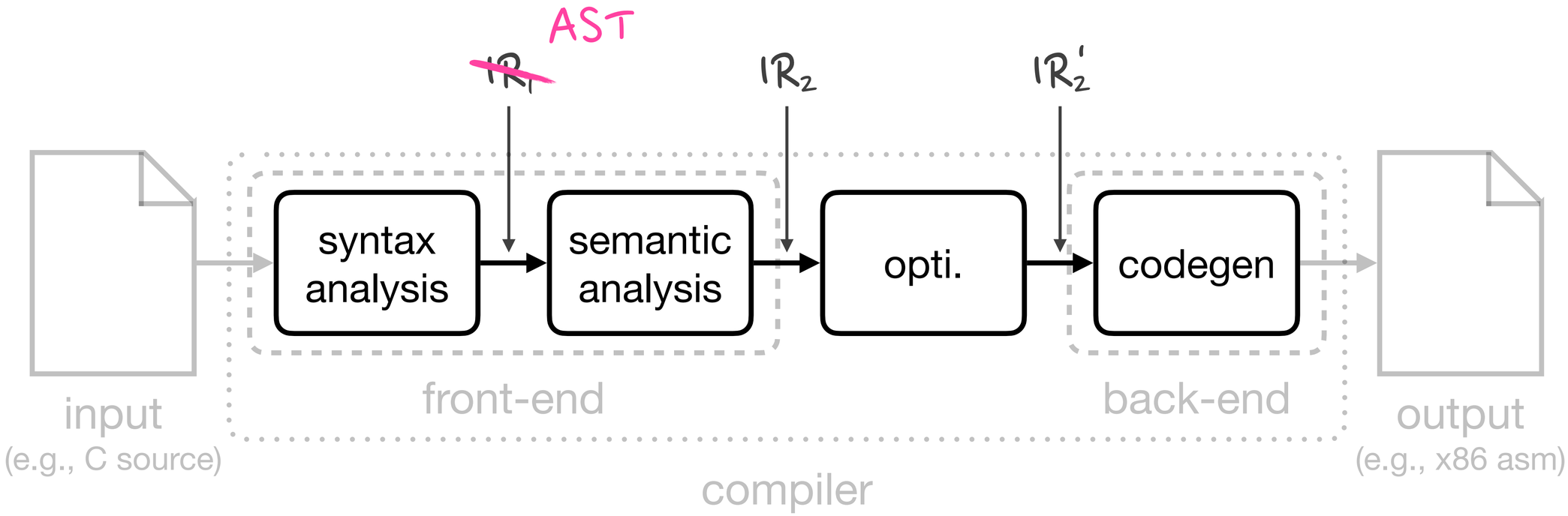
Structure of a modern compiler

The Three-Phase Model



Structure of a modern compiler

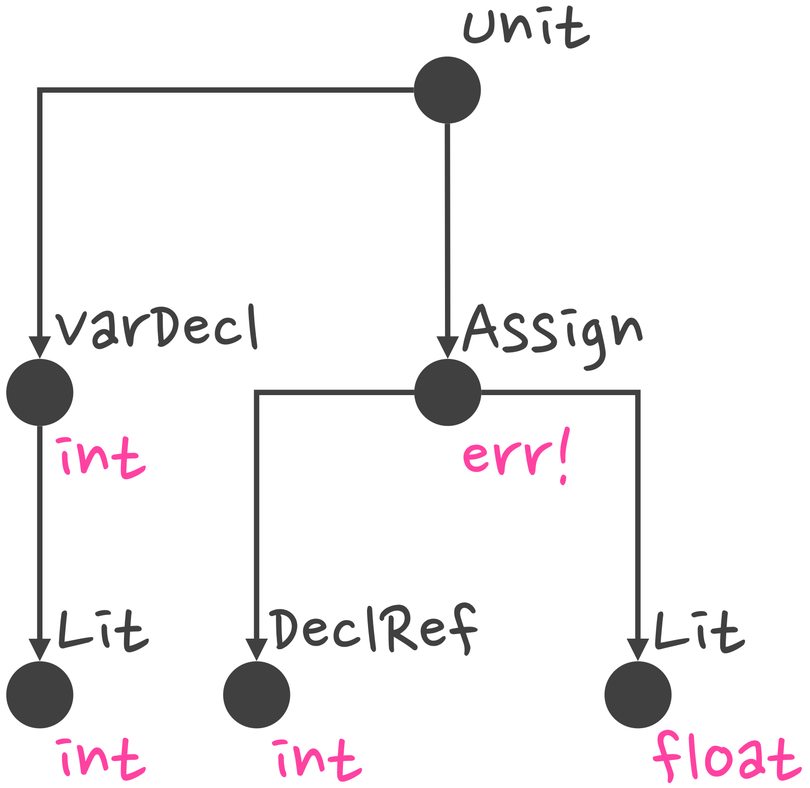
The Three-Phase Model



What's an Abstract Syntax Tree?

One IR to rule them all

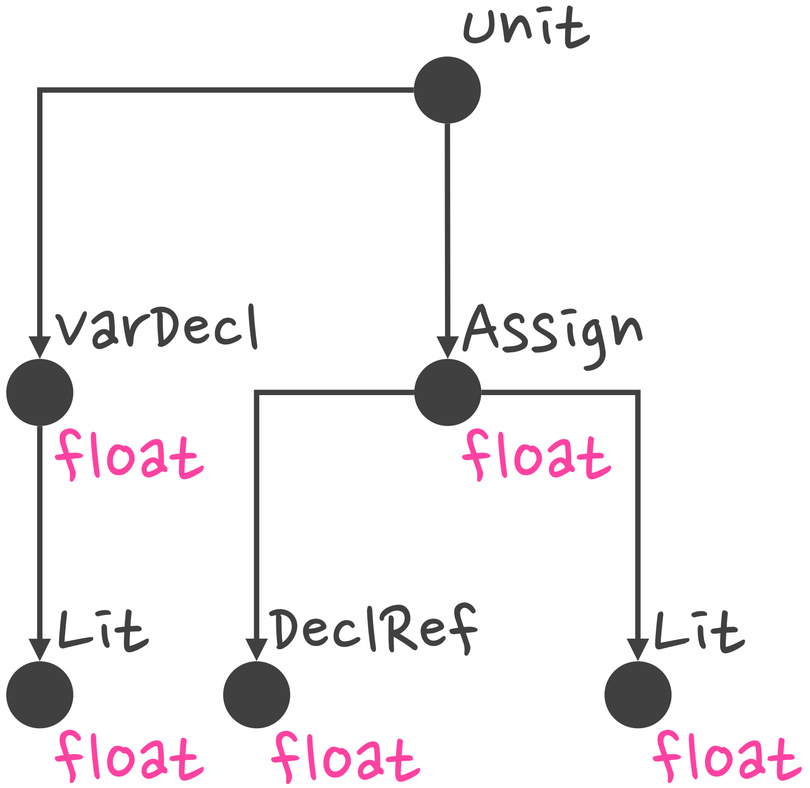
```c
1    int i = 0;
2    i = 4.2;
```



What's an Abstract Syntax Tree?

One IR to rule them all

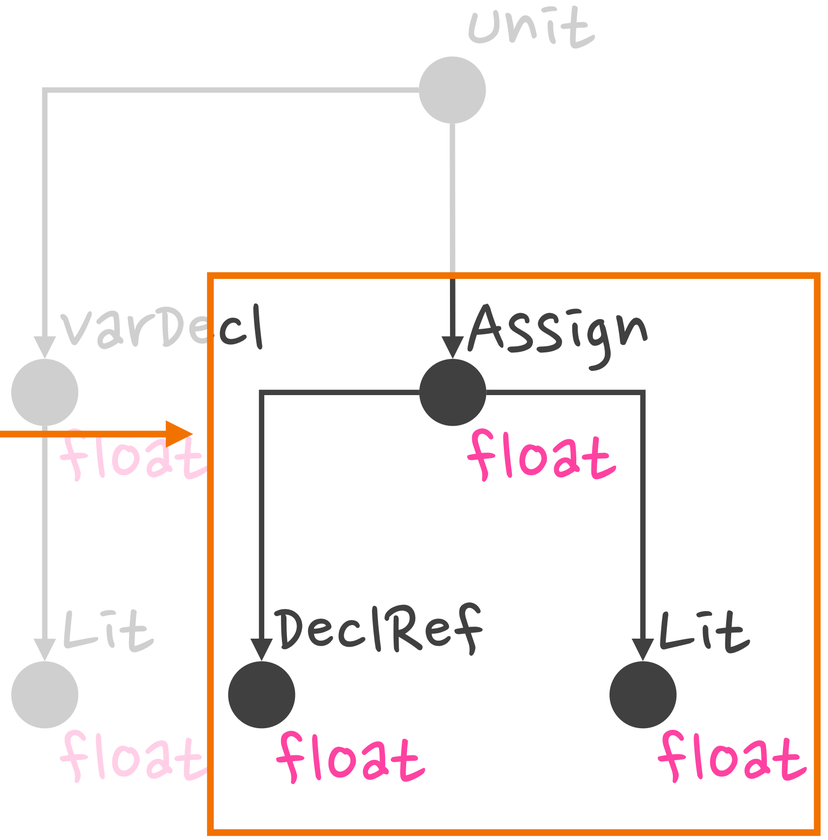
```
1    float i = 0;
2    i = 4.2;
```



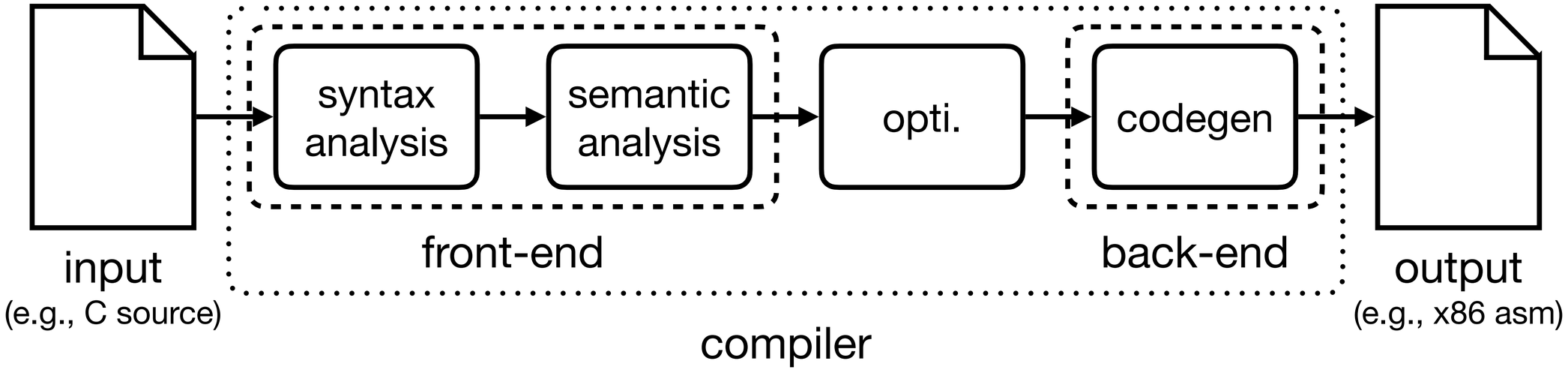
What's an Abstract Syntax Tree?

One IR to rule them all

$$\frac{\mu \vdash e \Downarrow v'', \mu'' \qquad \mu''[x \mapsto v''] \vdash \overline{s} \Downarrow \mu', v'}{\mu \vdash \boxed{x = e;}\ \overline{s} \Downarrow \mu', v'}$$

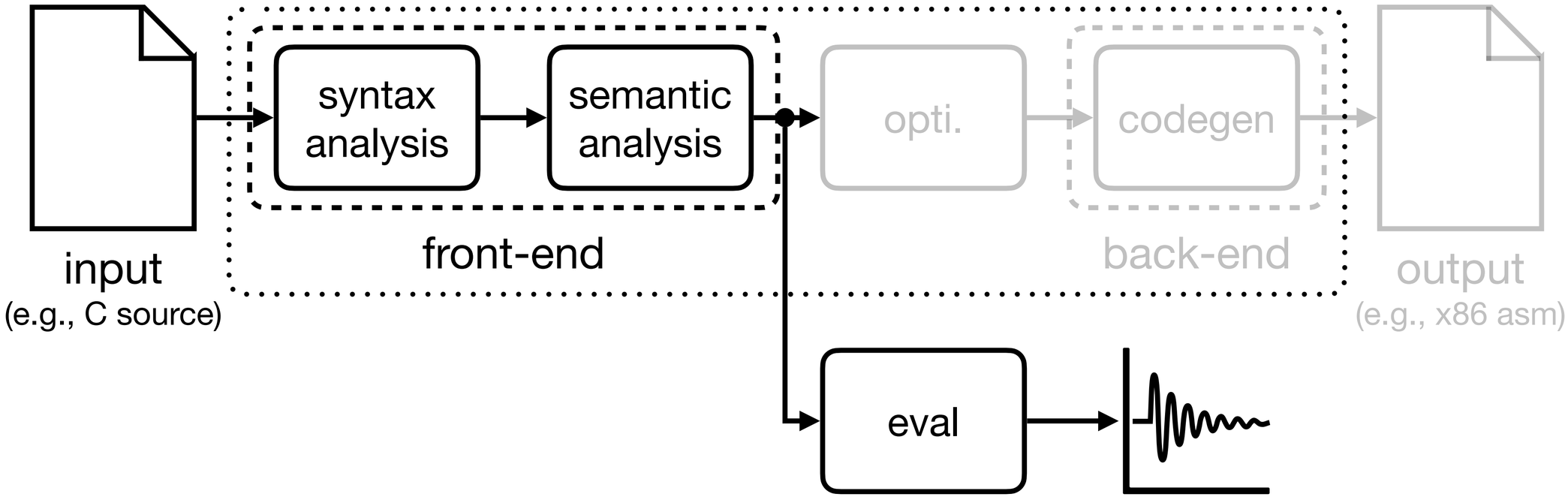


Structure of a modern compiler



Structure of a modern ~~compiler~~

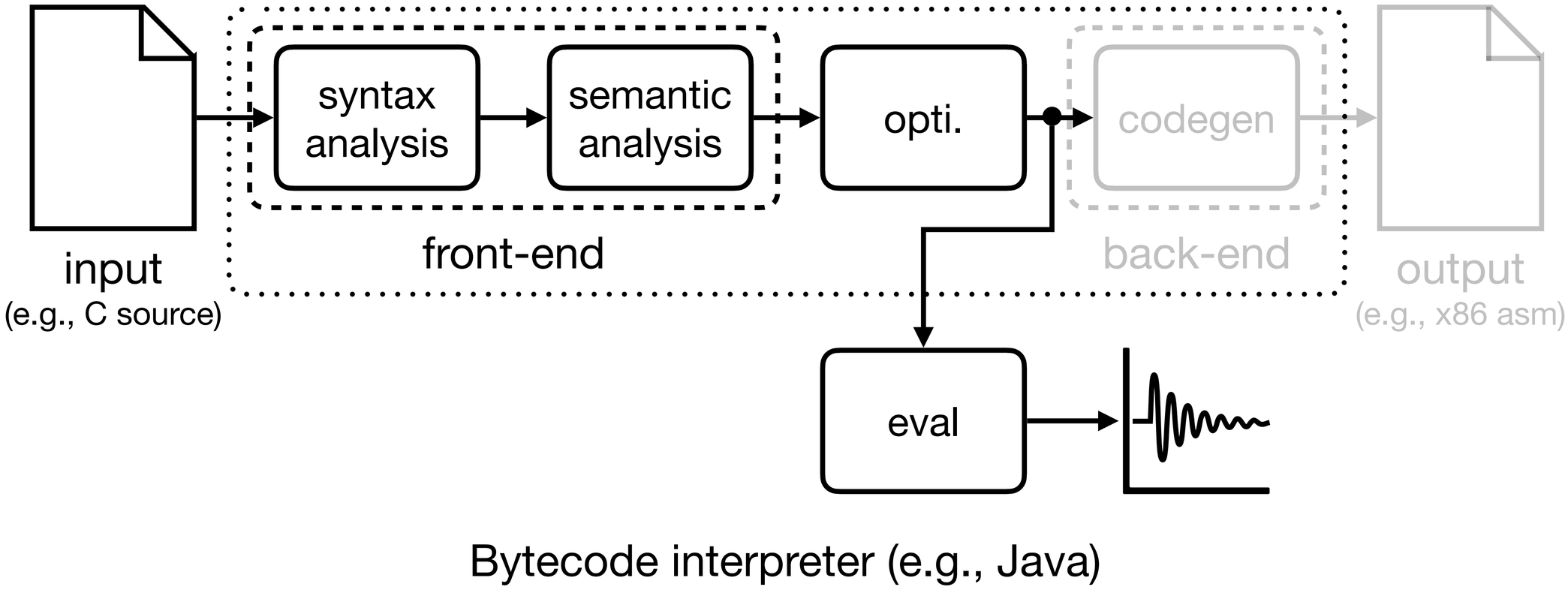
Interpreter



AST interpreter (a.k.a. tree walker)

Structure of a modern ~~compiler~~

Interpreter



Bytecode interpreter (e.g., Java)

(Automated) code optimizations

Time: run the program faster

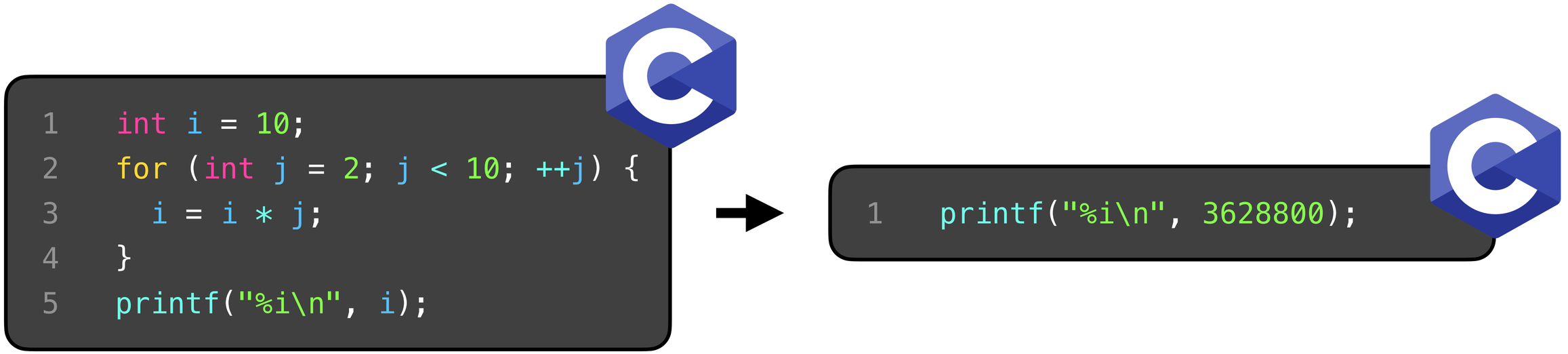
Space: use less memory

Energy: consume less power

without altering the program's semantics

(Automated) code optimizations

Platform independent



(Automated) code optimizations

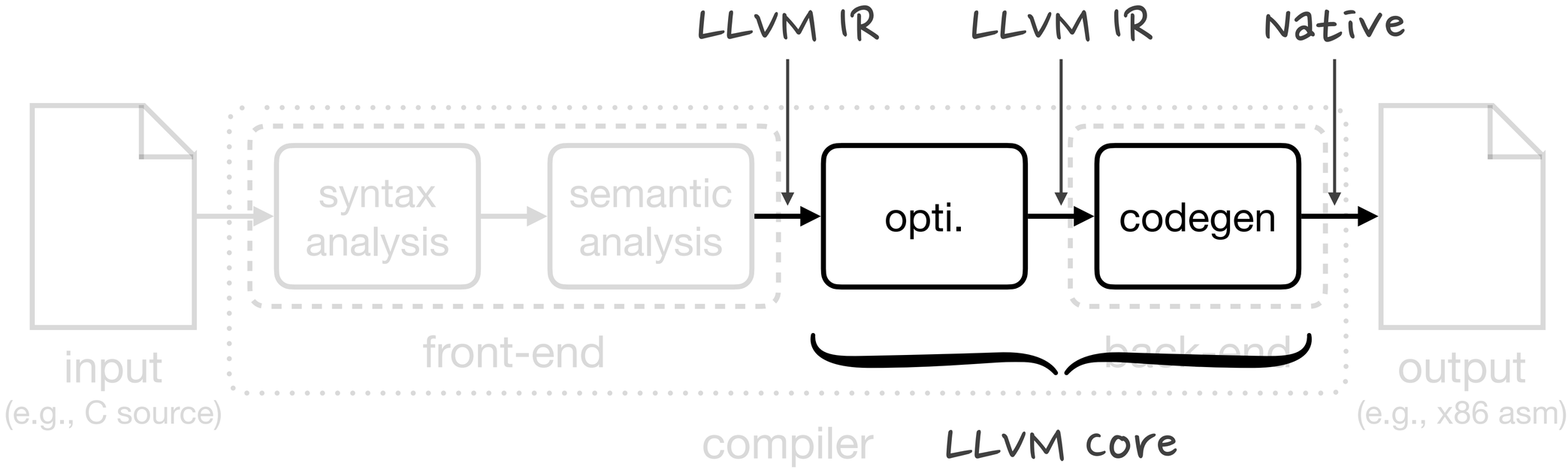
Platform dependent

```c
int c[4] = { 0 };
for (int i = 0; i < 4; ++i) {
  c[i] = a[i] * b[i];
}
```



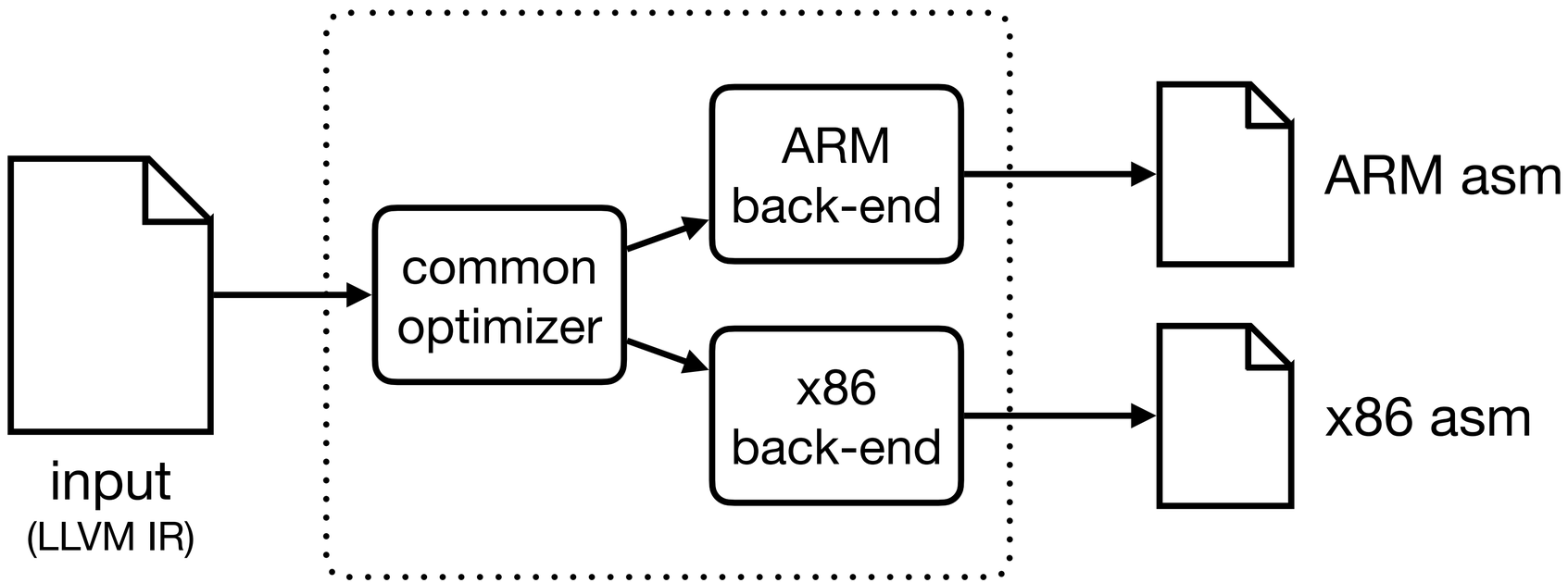
```c
int32x4_t c;
c = vmulq_s32(a, b);
```

What's LLVM (Core)



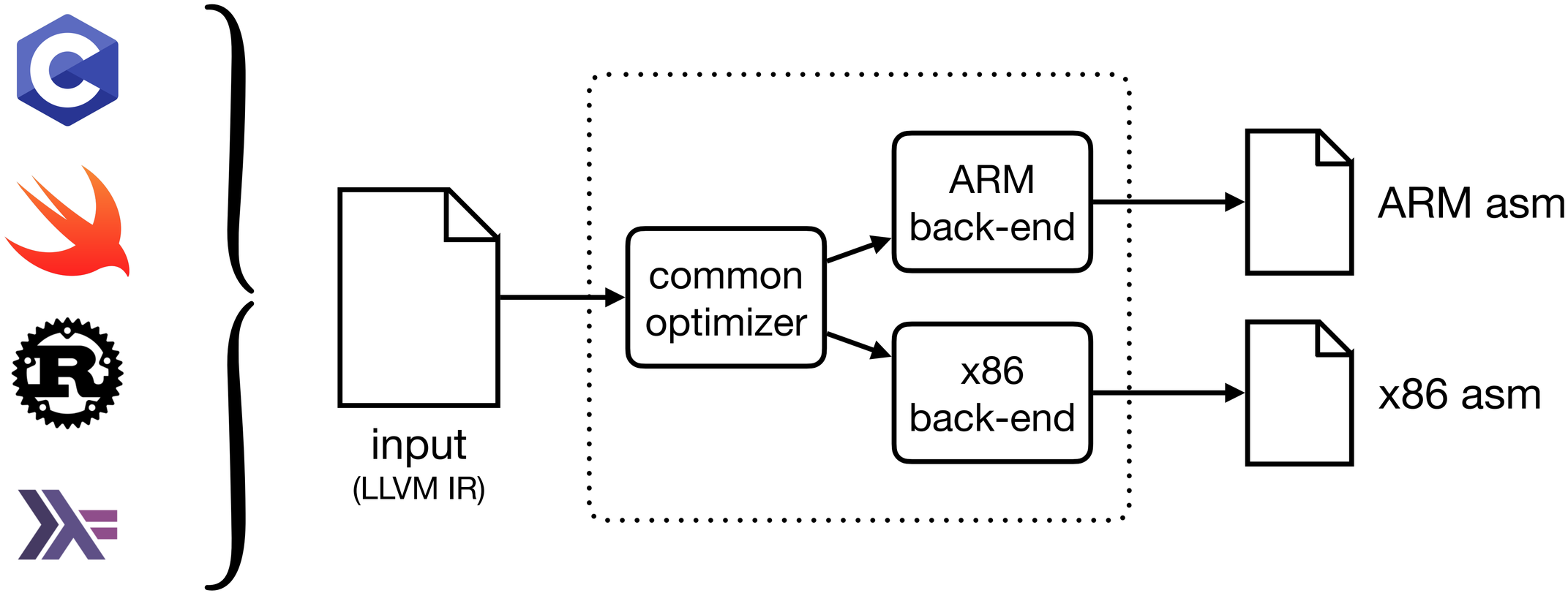
Why LLVM?

Solving the n langages × m backends problem



Why LLVM?

Solving the n langages × m backends problem



What does it look like?

```c
int factorial(int n) {
  return n > 1
    ? n * factorial(n - 1)
    : 1;
}
```

```llvm
define i32 @factorial(i32 %0) {
  %2 = icmp sgt i32 %0, 1
  br i1 %2, label %3, label %7
3:
  %4 = add nsw i32 %0, -1
  %5 = call i32 @factorial(i32 %4)
  %6 = mul nsw i32 %5, %0
  br label %7
7:
  %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
  ret i32 %8
}
```

What does it look like?

What does it look like?

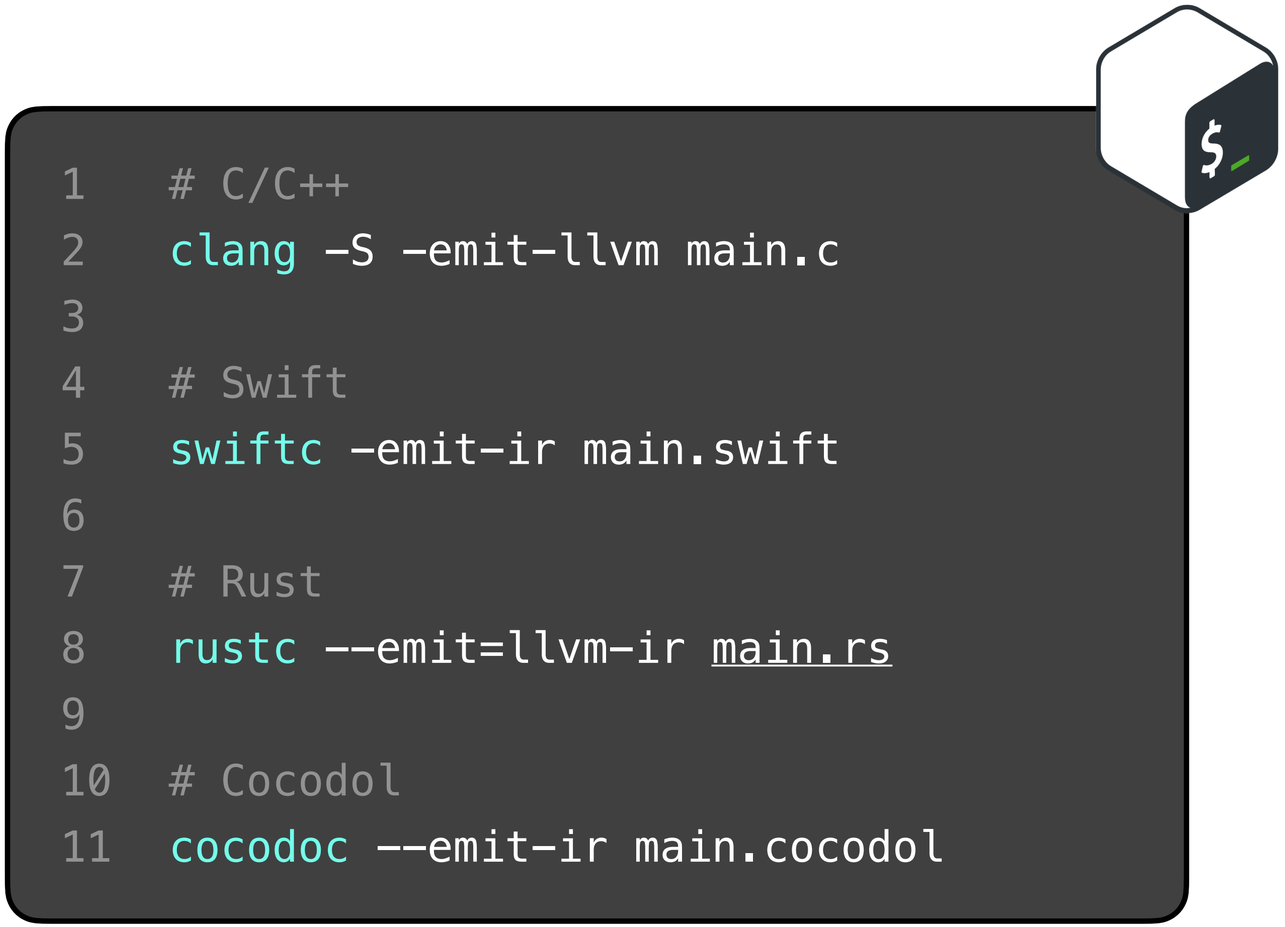
```
1   define i32 @factorial(i32 %0) {
2     %2 = icmp sgt i32 %0, 1
3     br i1 %2, label %3, label %7
4   3:
5     %4 = add nsw i32 %0, -1
6     %5 = call i32 @factorial(i32 %4)
7     %6 = mul nsw i32 %5, %0
8     br label %7
9   7:
10    %8 = phi i32 [ %6, %3 ], [ 1, %1 ]
11    ret i32 %8
12  }
```

SSA variable

operand

instruction

Watch real compilers at work



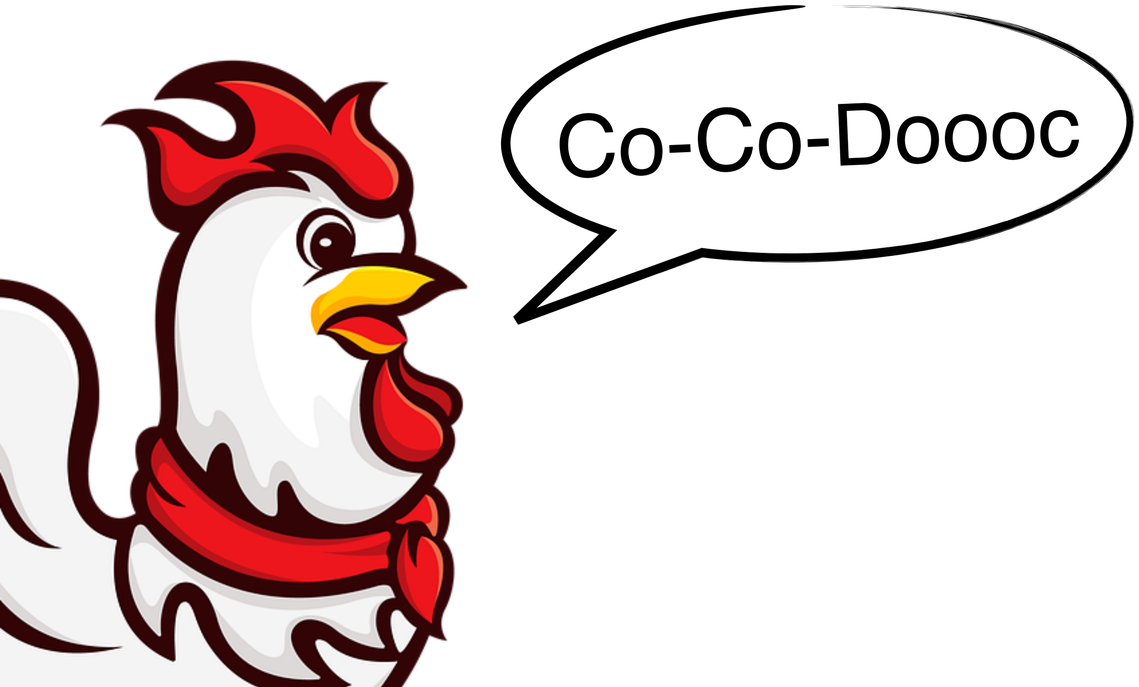
```
# C/C++
clang -S -emit-llvm main.c

# Swift
swiftc -emit-ir main.swift

# Rust
rustc --emit=llvm-ir main.rs

# Cocodol
cocodoc --emit-ir main.cocodol
```

Enough theory, let's emit some IR!



Cocodol

```
1   fun fac(n) {
2     if n > 1 {
3       ret n * fac(n - 1)
4     } else {
5      ret 1
6     }
7   }
```

https://github.com/kyouko-taiga/Cocodol

Existential containers

To be or not to be an integer

```
1   // How can we make this work?
2   {
3     var foo = 10
4     foo = (foo == 10)
5     print(foo)
6     // Prints "true"
7   }
```

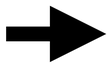
Let's look at the IR

Existential containers

To be or not to be an integer

```
we make this work?

 10
o == 10)
)
 "true"
```



```
1   %_Any = type { i64, i64 }
2   ; ...
3   %foo = alloca %_Any, align 8
4   store %_Any { i64 15, i64 10 }, %_Any* %foo, align 4
```

Closures

When variables develop Stockholm syndrome

```
1   // How can we make this work?
2   fun add(x) {
3     fun _add(y) {
4       ret x + y
5     }
6     ret _add
7   }
8   print(add(1)(2))
9   // Prints 3
```

Let's look at the IR

Final thoughts

Go further

- Help LLVM's optimizer
  - Type inference
  - Language-specific optimizations
- Implement debugging