How does compiler see your app

What is a compiler

A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language) ...

What is a compiler

```
Compiler(source code) -> Binary
Compiler(source code) -> Object file .o
Compiler(source code) -> Other source code
Compiler(source code) -> YAML/JSON
```

What is a compiler

- Read the source code
- Analyse the source code
- Optimize
- Transform

Why should I learn it -

Fun 😄🤓

Why should I learn it -

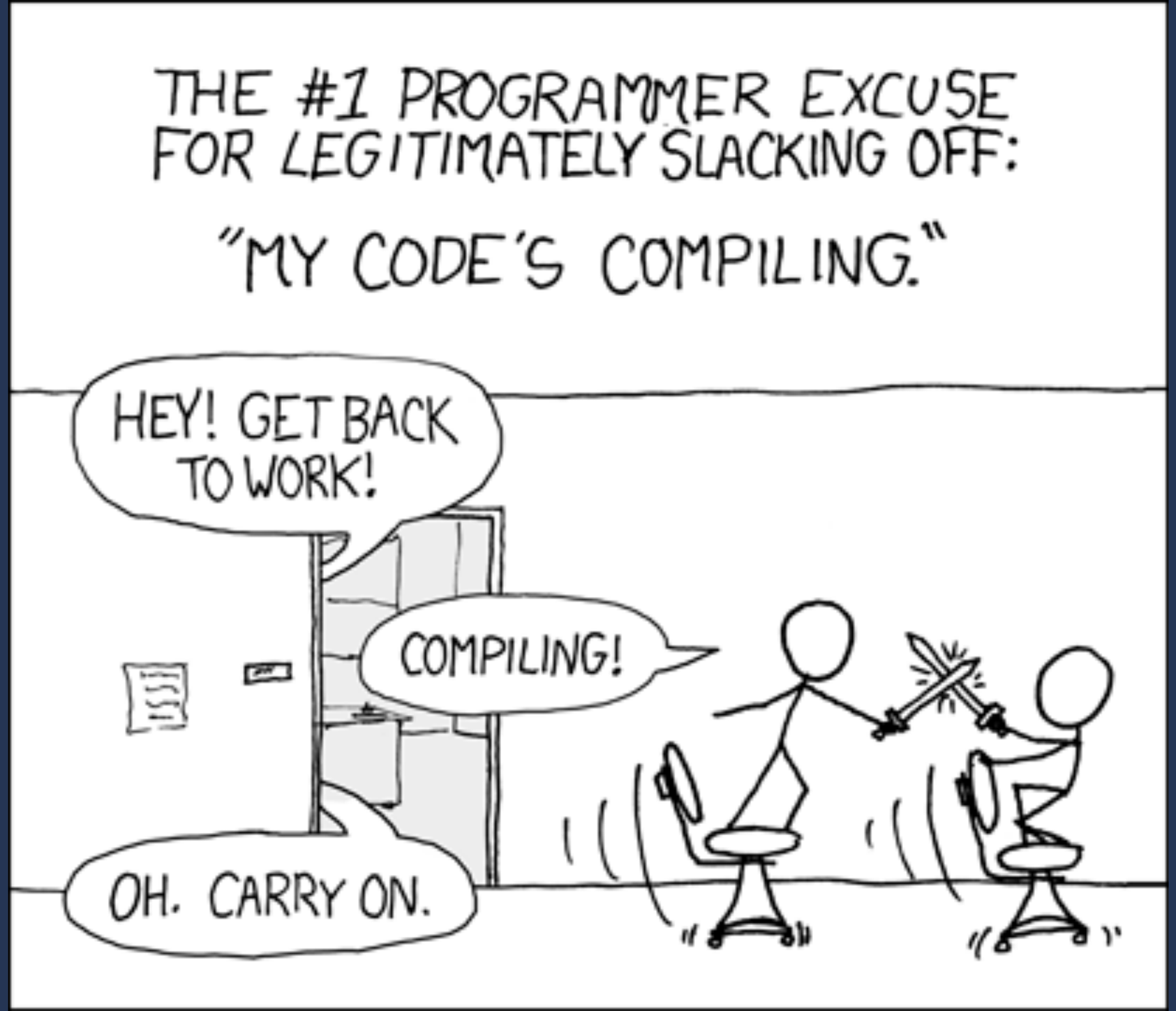
Profit

Why should I learn it - Profit

- New CS knowledge
- Build own compiler
- Building own tools:
  formaters, linters, code coverages, code generators ...

Bonus: You will be good at working with `String` in Swift :D

Where we use it ?


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Syntax highlighting

```
//: Playground - noun: a place where people can play

import Cocoa

class Person {
    let name :String

    init(name: String) {
        self.name = name
    }
}
```

```
1  //: Playground - noun: a place where people can play
2
3  import Cocoa
4
5  class Person {
6      let name :String
7
8      init(name: String) {
9          self.name = name
10     }
11 }
12
13
```

Code completion

```
1  //: Playground - noun: a place where people can play
2
3  import Cocoa
4
5  class Person {
6      let name :String
7
8      init(name: String) {
9          self.name = name
10         name.
```

```
M   String? addingPercentEncoding(withAllowedCharacters: CharacterSet)
M   String? ~~addingPercentEscapes(using: String.Encoding)~~
M      Void append(c: Character)
M      Void append(contentsOf: S)
M      Void append(other: String)
M    String appending(aString: String)
M    String appendingFormat(format: String, arguments: CVarArg...)
M   String? applyingTransform(transform: StringTransform, reverse: Bool)
```

SourceKitService Terminated

Editor functionality temporarily limited.

Error + Warning + Fix suggestions

```
41 class Person {
42     let name :String
43
44     init(name: String) {
45         _ = 10
46         self.name = name
```

Initialization of variable 'age' was never used; consider replacing with assignment to '_' or removing it

Fix-it Replace "var age" with "_"

```
40
41 class Person {
42     let name :String
43
44     init(name: String) {
45         var age = 10
46         self.name = name + 10
47
48     }
49 }
```

It's not IDE

```
+ ~/My_Projects/Presentations/How does Compiler see your app  master±  swiftc warning.swift
warning.swift:5:13: warning: initialization of variable 'age' was never used; consider replacing with assignment to '_' or removing it
        var age = 10
        ~~~~^~~
        _
```

+ Great example of **Business Logic** & **UI** separation

😀

Let's look inside a compiler

Source code

↓

↓

App

Frondend & Backend

Frondend

- Lexical analysis
- Syntax analysis
- Semantic analysis
- Generates errors and warnings
- Generates IR

Backend

- Code independence optimizations
- Generates target-dependent assembly code
- Hardware specific optimizations
- Machine code

Example

```
let a = 10
```

1 - Language grammar

- Finite automata (state machine)
- Regular expression

`let`

GRAMMAR OF A CONSTANT DECLARATION

constant-declaration → *attributes*$_{opt}$ *declaration-modifiers*$_{opt}$ **let** *pattern-initializer-list*

pattern-initializer-list → *pattern-initializer* | *pattern-initializer* , *pattern-initializer-list*

pattern-initializer → *pattern* *initializer*$_{opt}$

initializer → = *expression*

2 - Lexical Analysis

```
var a = 10
<keyword var> <identifier a> <operator => <value 10>

+ + +
<operator +> <operator +> <operator +>
```

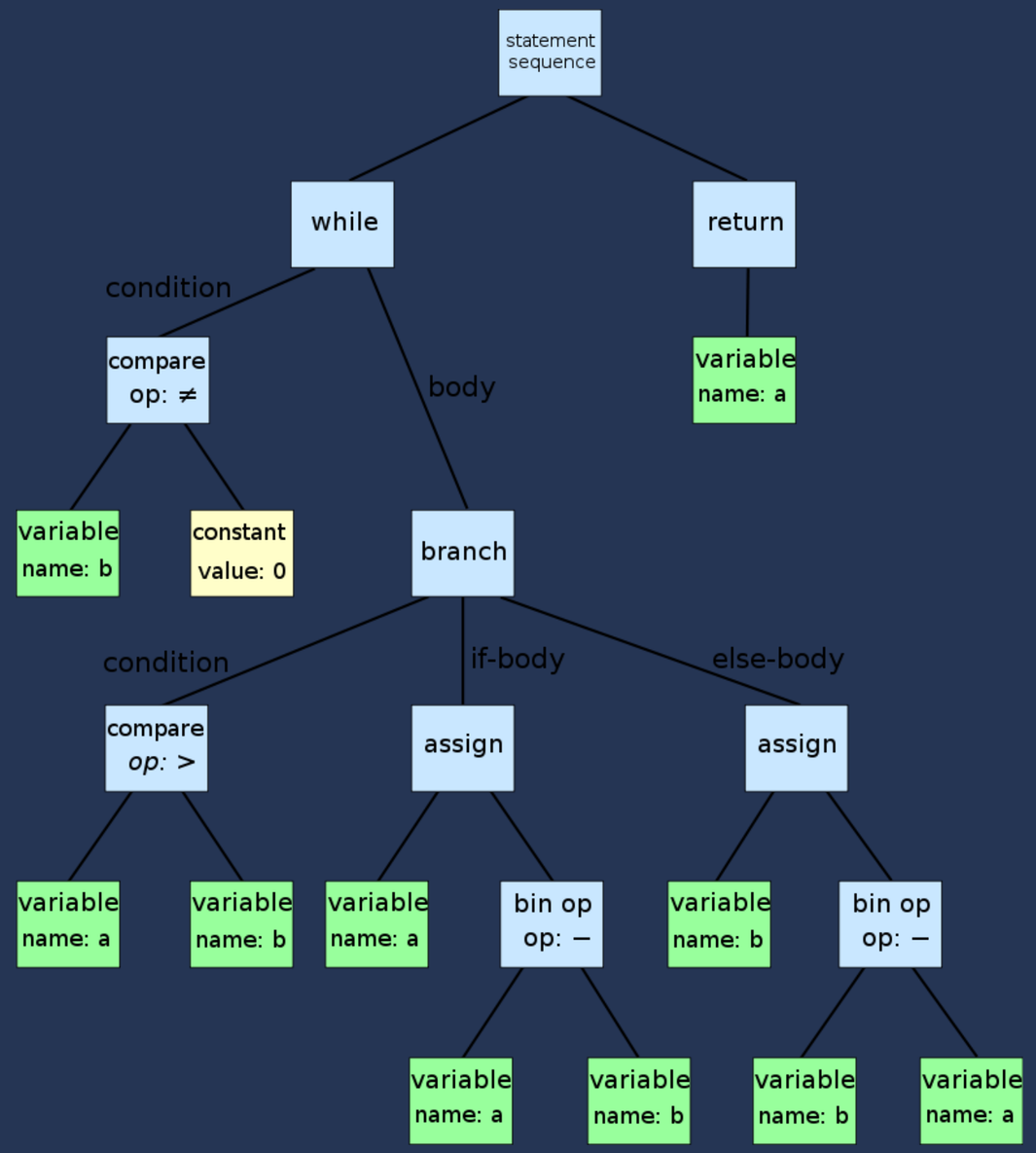
3 - Syntax Analysis

- Combines tokens together
- Produces AST

```swift
var a = 10
//swiftc -dump-parse hello.swift
```

AST

```
source_file
  (top_level_code_decl
    (brace_stmt
      (pattern_binding_decl
        (pattern_named 'a')
        (integer_literal_expr type='<null>' value=10))
))
  (var_decl "a" type='<null type>' storage_kind=stored))
```


statement sequence
while
return
condition
body
compare
op: ≠
variable
name: a
variable
name: b
constant
value: 0
branch
condition
if-body
else-body
compare
op: >
assign
assign
variable
name: a
variable
name: b
variable
name: a
bin op
op: −
variable
name: b
bin op
op: −
variable
name: a
variable
name: b
variable
name: b
variable
name: a

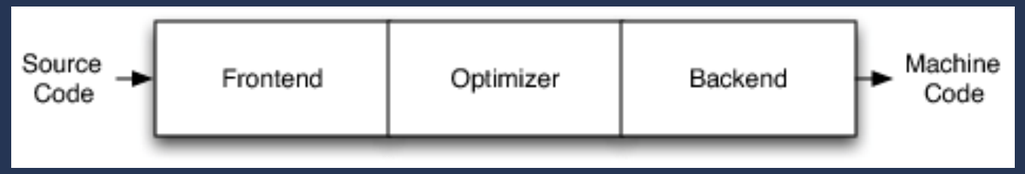
4 - Semantic Analysis

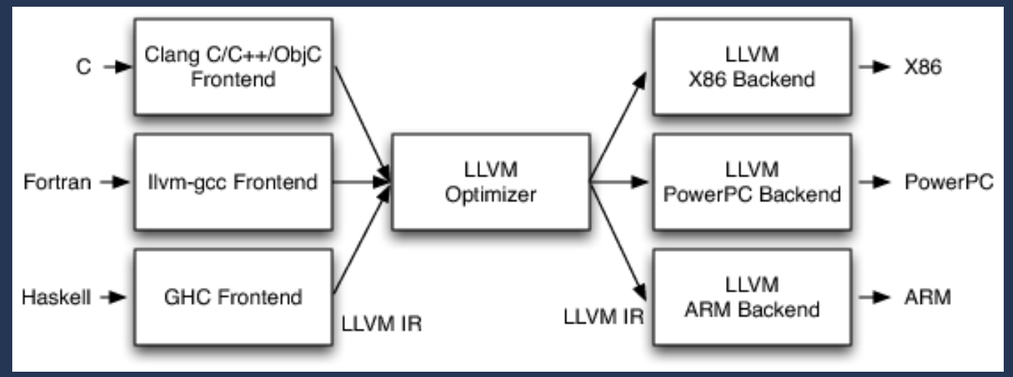
```
var a: String = 10
```

- Check for correctness
- Type mismatches, method call and arguments, variable refernce, violating access

5 - Generate IR

- Assembly like
- Language & independent independent





Use LLVM IR & Backend

C, C++, ActionScript, Ada, C#, Common Lisp, Crystal, D, Delphi, Fortran, OpenGL Shading Language, Halide, Haskell, Java bytecode, Julia, Lua, Objective-C, Pony, Python, R, Ruby, Rust, CUDA, Scala, and ...

Swift :)

5 - Generate IR

```
func square(a: Int) -> Int {
    return a*a;
}

//IR
define i32 @square(i32 %a) {
  %1 = mul i32 %a, %a
  ret i32 %1
}
```

5 - Generate IR

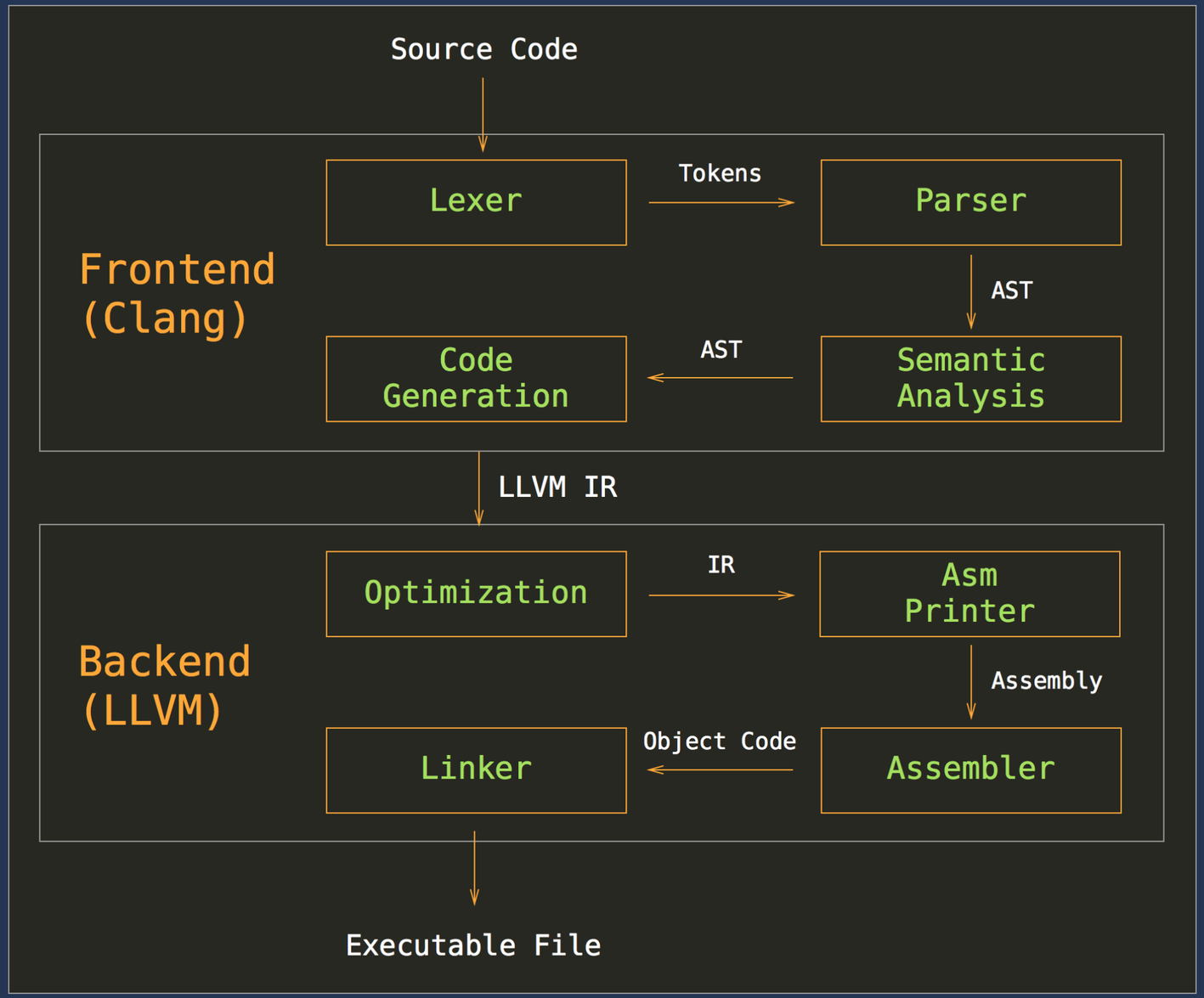
```
var a = 10
//swiftc -emit-ir hello.swift
```

```
; ModuleID = 'a'
source_filename = "a"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.9"

%Si = type <{ i64 }>
%Vs5Int32 = type <{ i32 }>
%Sp = type <{ i8* }>

@_Tv1a1aSi = hidden global %Si zeroinitializer, align 8
@_TZvOs11CommandLine5_argcVs5Int32 = external global %Vs5Int32, align 4
@globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_token4 = external global i64, align 8
@_TZvOs11CommandLine11_unsafeArgvGSpGSqGSpVs4Int8___ = external global %Sp, align 8
@__swift_reflection_version = linkonce_odr hidden constant i16 1
@llvm.used = appending global [1 x i8*] [i8* bitcast (i16* @__swift_reflection_version to i8*)], section "llvm.metadata", align 8

define i32 @main(i32, i8**) #0 {
entry:
  %2 = bitcast i8** %1 to i8*
  store i32 %0, i32* getelementptr inbounds (%Vs5Int32, %Vs5Int32* @_TZvOs11CommandLine5_argcVs5Int32, i32 0, i32 0), align 4
  %3 = load i64, i64* @globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_token4, align 8
  %4 = icmp eq i64 %3, -1
  br i1 %4, label %once_done, label %once_not_done

once_not_done:                                    ; preds = %entry
  call void @swift_once(i64* @globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_token4, i8* bitcast (void ()* @globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_func4 to i8*))
  br label %once_done

once_done:                                        ; preds = %once_not_done, %entry
  %5 = load i64, i64* @globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_token4, align 8
  %6 = icmp eq i64 %5, -1
  call void @llvm.assume(i1 %6)
  store i8* %2, i8** getelementptr inbounds (%Sp, %Sp* @_TZvOs11CommandLine11_unsafeArgvGSpGSqGSpVs4Int8___, i32 0, i32 0), align 8
  store i64 10, i64* getelementptr inbounds (%Si, %Si* @_Tv1a1aSi, i32 0, i32 0), align 8
  ret i32 0
}

declare void @globalinit_33_FD9A49A256BEB6AF7C48013347ADC3BA_func4() #0

declare void @swift_once(i64*, i8*)

; Function Attrs: nounwind
declare void @llvm.assume(i1) #1

attributes #0 = { "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "target-cpu"="core2" "target-features"="+ssse3,+cx16,+fxsr,+mmx,+sse,+sse2,+sse3" }
attributes #1 = { nounwind }

!llvm.module.flags = !{!0, !1, !2, !3, !4, !5, !9, !10}

!0 = !{i32 1, !"Objective-C Version", i32 2}
!1 = !{i32 1, !"Objective-C Image Info Version", i32 0}
!2 = !{i32 1, !"Objective-C Image Info Section", !"__DATA, __objc_imageinfo, regular, no_dead_strip"}
!3 = !{i32 4, !"Objective-C Garbage Collection", i32 1024}
!4 = !{i32 1, !"Objective-C Class Properties", i32 64}
!5 = !{i32 6, !"Linker Options", !6}
!6 = !{!7, !8}
!7 = !{!"-lswiftCore"}
!8 = !{!"-lobjc"}
!9 = !{i32 1, !"PIC Level", i32 2}
!10 = !{i32 1, !"Swift Version", i32 4}
```

6 - Dive into darkness Backend

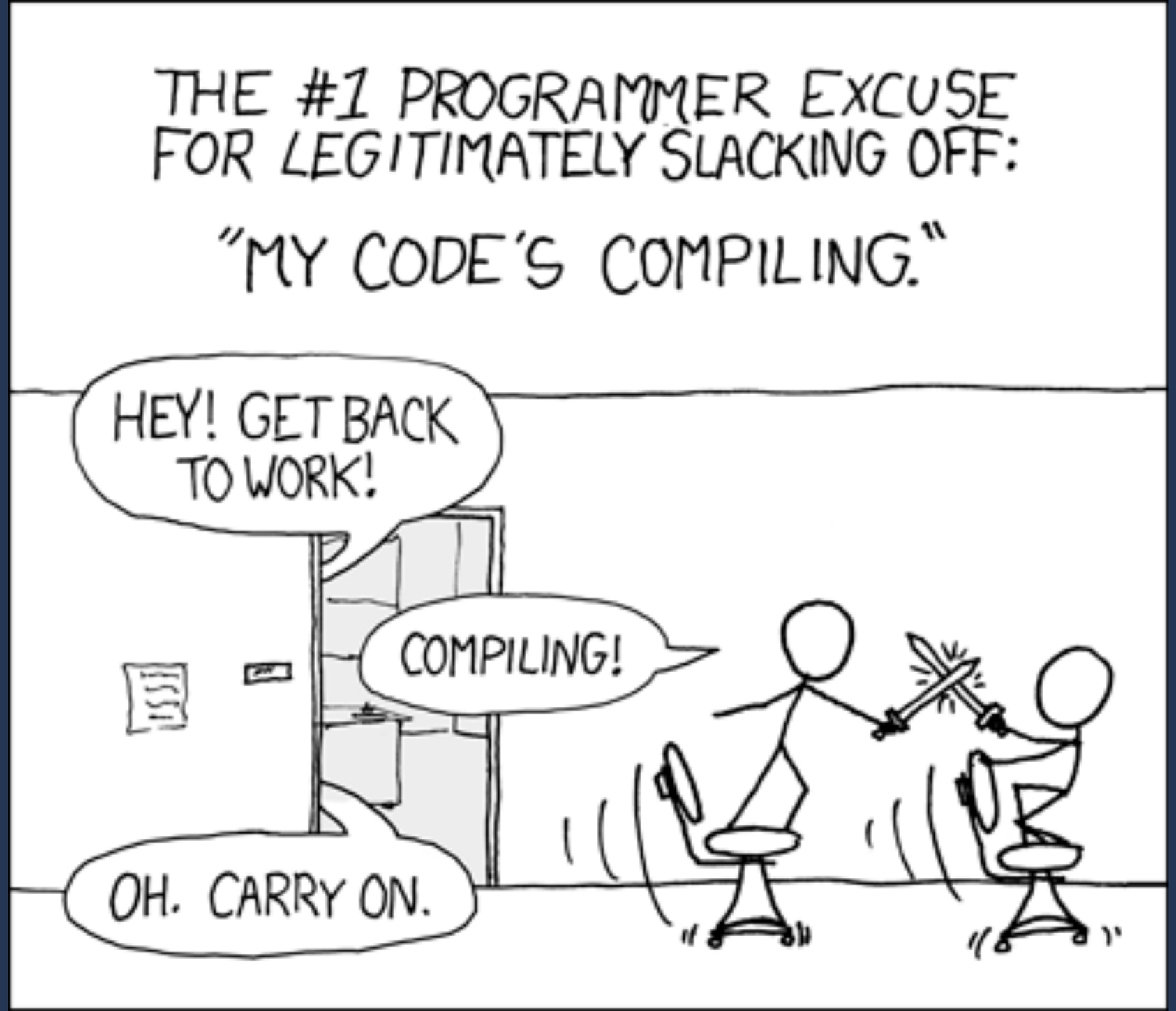
Backend - Still a lot of Magic!

- **Analysis and Transform** 🔮
  http://llvm.org/docs/Passes.html
- **Generic optimizations**
  Dead code elimination, Loop optimizations, etc.
- **Hardware specific optimizations**
  Use of Registers and architecture specific commands, example - armv7s
- Assembly code


Source Code
Frontend
(Clang)
Lexer
Tokens
Parser
AST
Semantic
Analysis
AST
Code
Generation
LLVM IR
Backend
(LLVM)
Optimization
IR
Asm
Printer
Assembly
Assembler
Object Code
Linker
Executable File

The End!

- swiftc --help
- http://llvm.org
- Youtube - LLVM
- Compiler for dummies :D


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

@KostiaKoval