# @cocaTalk #3

Jayson.xu
http://akadealloc.github.io/blog

# Agenda

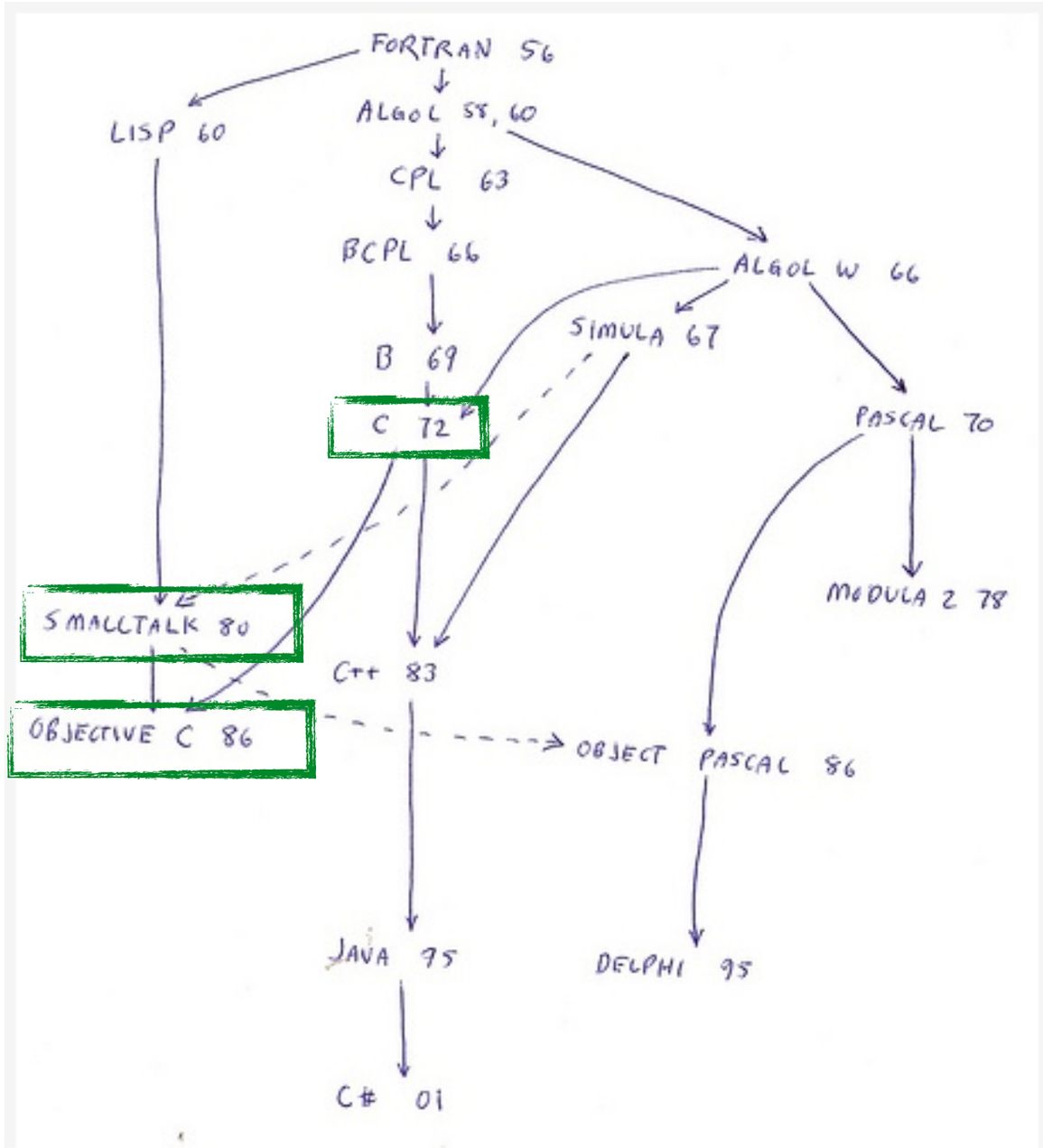
- PART 1 : Introduce to Swift
- PART 2 : Inside Swift

# PART 1

- Objective-C
- Swift Overview /  New Features
- Optional
- Tuples
- Functions / Closures
- Struct
- Enums
- Pattern Matching
- Generics
- How to Get Started
- Wrap up : My thoughts about Swift

# Objective-C

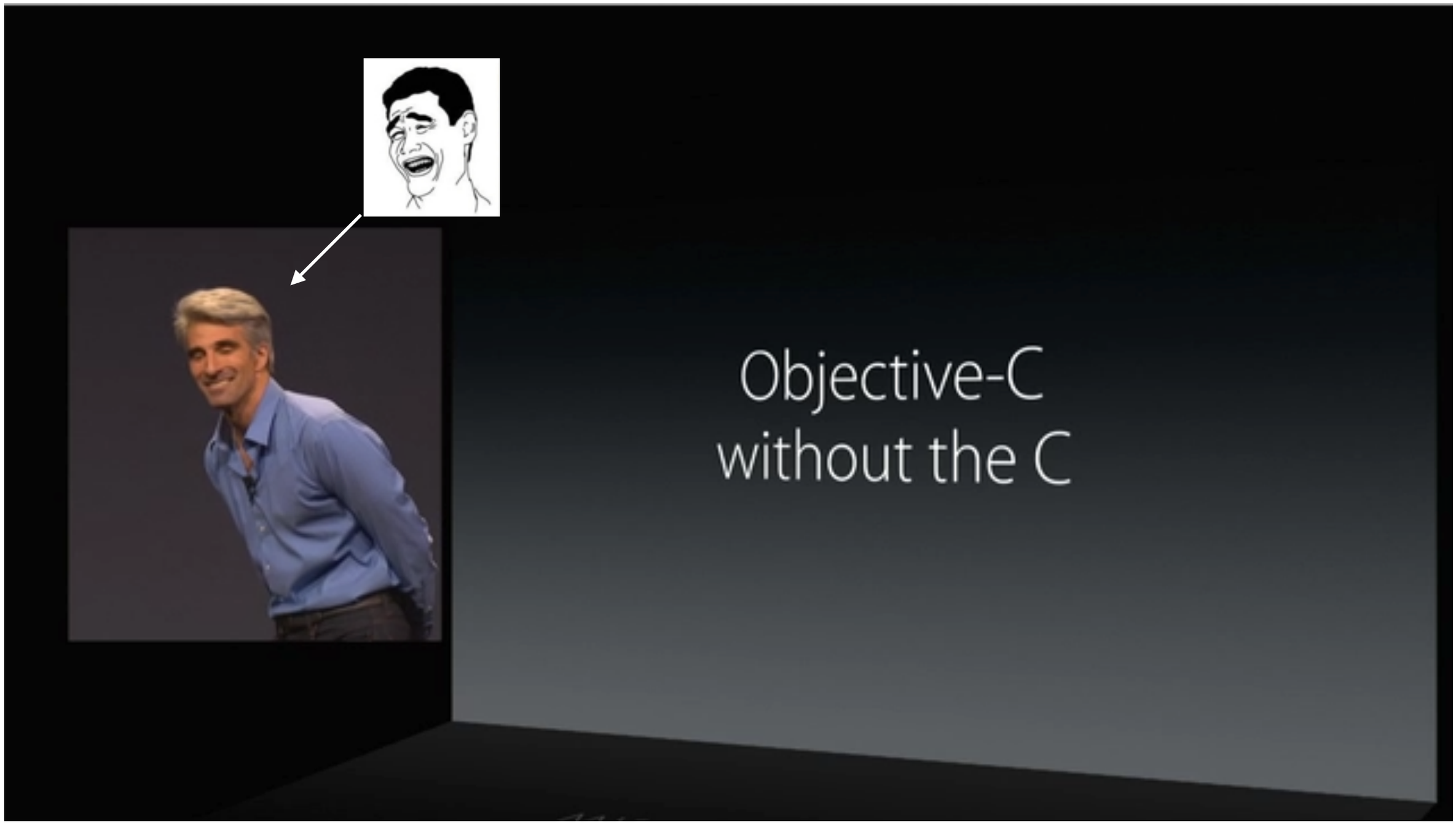
- Very old language
- C+SmallTalk => OOP + low level UNIX/C APIs
- Full of Design-patterns
- Powerful Runtime Features:
    - Static language with dynamic environment:
        - Create class,method,var… on the fly
        - dispatch method dynamically
        - whatever you can imagine….
    - MetaClass -> Meta Programming：
        - wax, NU, RubyMotion…
        - DSL



| Objective-C | Java |
| --- | --- |
| . class | . class |
| . methods | . methods |
| . protocols | . interface |
| . categories | . NO |
| . functions | . functions |
| . closures | . inner classes |
| . ARC(compile time) | . JVM(runtime) |

Every thing is Perfect in
Objective-C
world

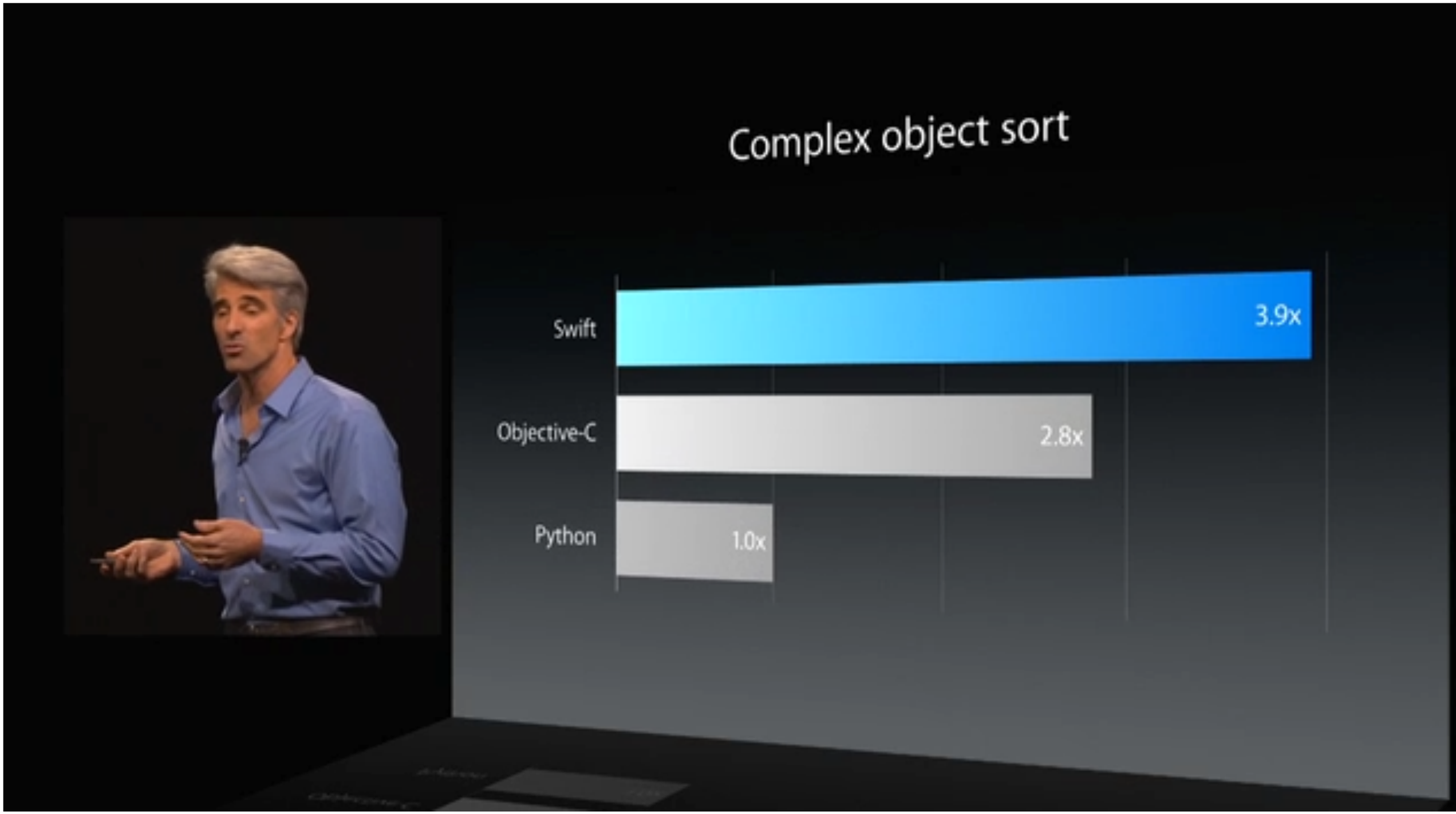
# Until This Guy Came Out



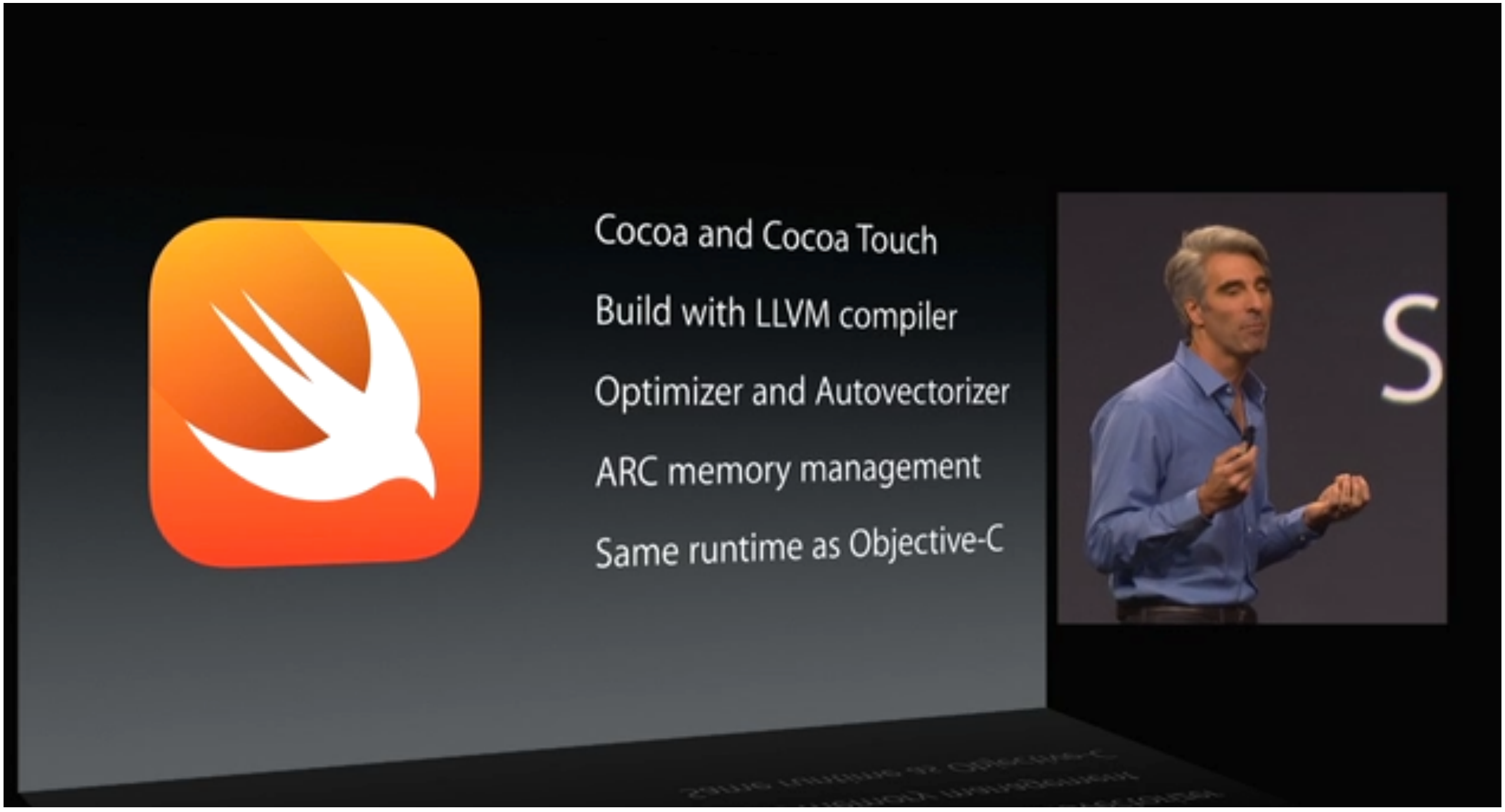
# First, let’s recap

New programming language

Fast
Modern
Safe
Interactive


Complex object sort
Swift
3.9x
Objective-C
2.8x
Python
1.0x

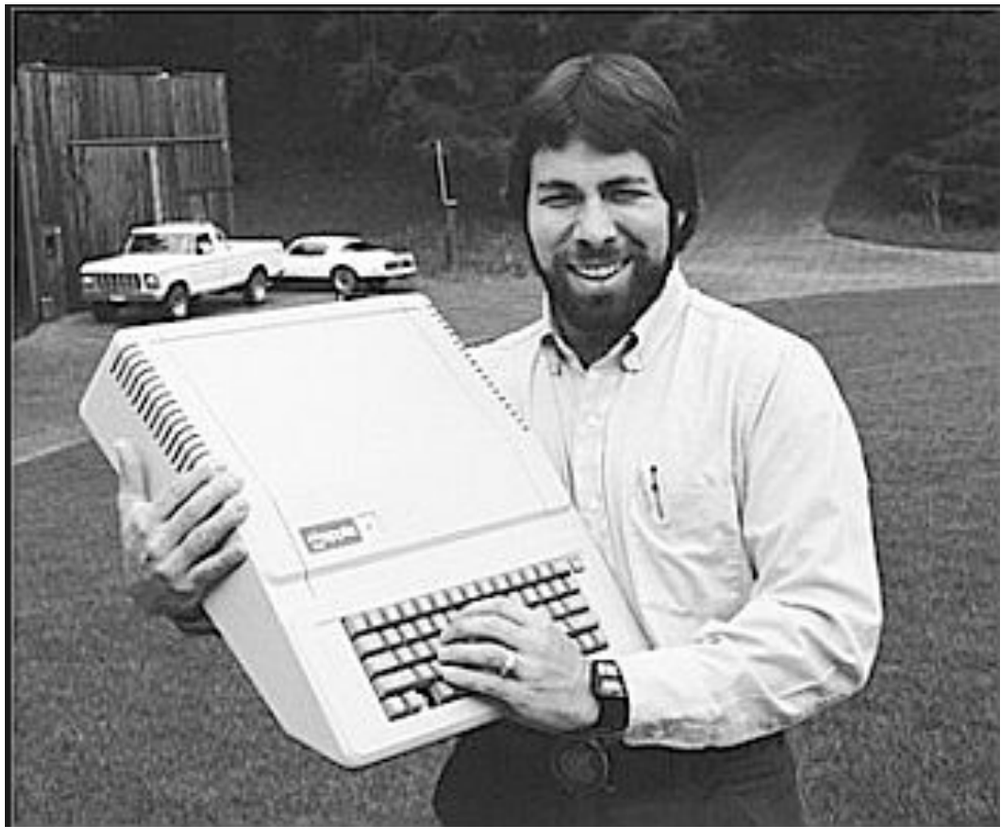
Functional patterns
Protocols and extensions on structs
Pattern matching
Concise syntax
Closures
Generics
Fast iteration
Native collections
Optional types
Operator overloading
Object orientation
Namespaces
Tuples
Type inference
Clear mutability syntax
Read-Eval-Print-Loop (REPL)
Interactive playground
Multiple return types
Compile to native code


Cocoa and Cocoa Touch
Build with LLVM compiler
Optimizer and Autovectorizer
ARC memory management
Same runtime as Objective-C
S

# Safety!

- “It is designed for safety.”
- “Swift makes your code safe”
- “…makes your code safer…”
- “…makes code safe…”
- “…much safer…”
- “…again, makes your code safer…”

Objective-C



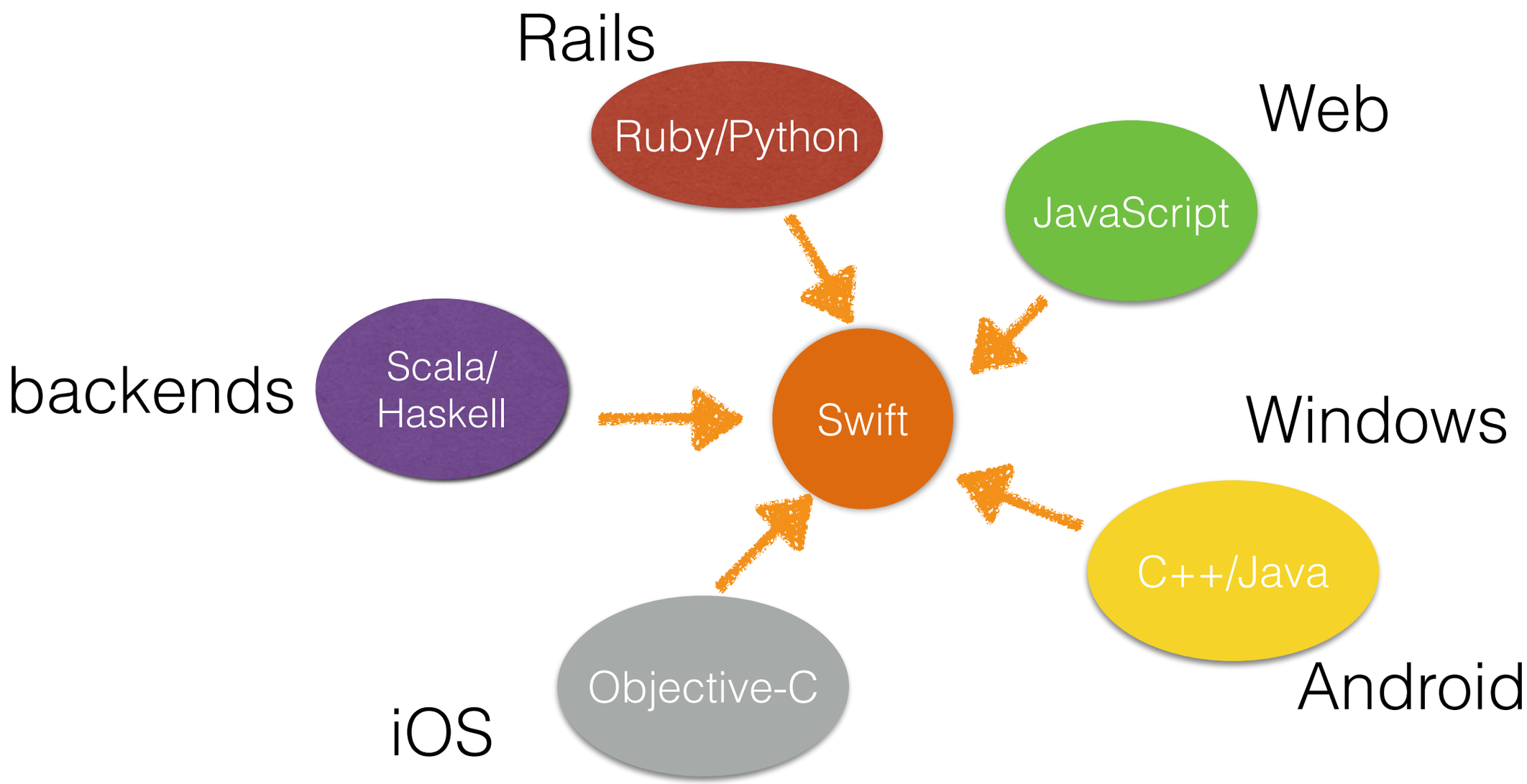
old,classic,geek…

Swift

young, pretty, fashion ….

| Objective-C | Swift |
| --- | --- |
| . class | . class |
| . methods | . methods |
| . protocols | . protocols |
| . categories | . categories |
| . functions | . functions |
| . ARC | . ARC |
| . blocks | . blocks |
| . Cocoa API | . Cocoa API |

# Swift


Rails
Ruby/Python
Web
JavaScript
backends
Scala/
Haskell
Swift
Windows
C++/Java
Android
Objective-C
iOS

Objective-C

Swift

. class
. methods
. protocols
. categories
. functions
. ARC
. closures
. Cocoa API

+

*a bit new*

Swift

. new swift values
. struct
. generics
. namespaces
. access control
. operator overloading
. Objc inter-operation

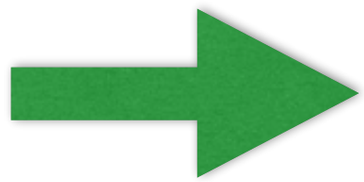
Objective-C

| Swift | + | *a bit new* Swift | + | *really new* Swift |
|---|---|---|---|---|
| . class | | . new swift values | | . enums ( new ) |
| . methods | | . struct ( new ) | | . optional type |
| . protocols | | . generics | | . pattern matching |
| . categories | | . namespaces | | . type inference |
| . functions | | . access control | | . immutability |
| . ARC | | . operator overloading | | . tuples |
| . closures | | . Objc inter-operation | | |
| . Cocoa API | | | | |

*really new*

Swift

. enums ( new )
. optional type
. pattern matching
. type inference
. immutability
. tuples



# FP

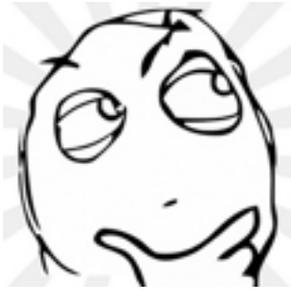
SINCE 1980 or earlier

http://www.raywenderlich.com/88707/swift-functional-programming-matter-tech-talk-video

*“Objective-C without the C” implies*
*something subtractive,*
*but*
*Swift dramatically expands the design space through*
*the*
*introduction of generics and functional programming concepts.”*
*— Chris Lattner*

It’s really cool

# Nil



```
NSString* ret = [@"abc" stringByAppendingString:obj];

Bool ret = [@“abc” isEqualToString nil];  //Does this one crash?
```

```
var ret = "abc".stringByAppendingString(obj);
```

API Doc:

```
// Returns a new string made by appending a given string to the `String`.
func stringByAppendingString(aString: String) -> String
```

# optional

```
func someFunction(v:String) -> Int? { …… }

let a = someFunction(“hello”)

if a != nil
{
    let b = a! //unwrap a
    //do something…
}

//or
if let a = a
{
    //do something…
}
```

# optional

```
var a:UInt32 //error!

var a:UInt32? //b might be nil

var a:UInt32! //force unwrapping
```

# Tuples

```
func myMethod(name:String) -> (Bool,Int?)
{
   //. . .

   return (result, errorCode)

}


let t1 = myMethod(“swift”)

if (t1.0)
{
   …
}

//binding
let (r,_) = myMethod(“swift”)

if (r)
{
   …
}
```

# Functions

```
// (x:int, y:int) -> int
func sum(x:int, y:int) -> int {return x+y}

// int(sum*)(int,int)
int sum(int x, int y){ return (x+y); }

// int(sum^)(int,int)
int sum(int, int) = ^(int x, int y){ return (x+y); }

(* int*int -> int *)
func sum(x:int, y:int) = x+y

sum::Int -> Int -> Int
sum x y = x + y
```

# Closure

```
//function:
func greetingFunc()->()
{
   println("greeting func is called!”)
}


//closure is just an unnamed function:
let greetingBlock:()->() =
{
   println("greeting block is called!”)
}
```

# Function is First-Class Object

```
func repeat(count:Int, task:()->())
{
    for i in 0...count
    {
        task()
    }
}
```

```
//defining a function
func print()->(){ println(“hello")}

//defining a block
let proc:()->() = { println("hello")}

//pass closure as parameter
repeat(2,proc)

//pass function as parameter
repeat(2,print)
```

# Advanced Topics on Function/Closure

- External Parameters
- Variadic parameters
- #,inout….
- Multiple return values
- Map-Filter-Reduce
- Trailing closures

# Struct

C:

```
typedef struct{

    float x;
    float y;
    float w;
    float h;

 }Rect;
```

Swift:

```
struct Rect{

    var x:float;
    var y:float;
    var w:float;
    var h:float;
};
```

```
Rect t = (Rect){1.0f,2.0f,3.0f,4.0f};
```

```
//struct is a Swift Object
var t:Rect = Rect(x:1.0f, y:2.0f, w:3.0, h:4.0)
```

# Struct

```
struct Point
{
  var x,y :Double
}

struct Size
{
    var width,height :Int
}
```

```
struct Rect
{

  var origin:Point;
  var sz:Size
  var taggedPoints:[Point]

  var area:Double
  {
      return (Double)(sz.height * sz.height)
  }

  func isBiggerThanRect(other:Rect) -> Bool
  {
      return self.area > other.area
  }

  mutating func addTaggedPoint(point:Point)
  {
        self.taggedPoints.append(point)
  }

};
```

# Enum

```
typedef NS_ENUM(NSInteger, UIDocumentSaveOperation){

        UIDocumentSaveForCreating,
        UIDocumentSaveForOverwriting
};
```

```
 enum UIDocumentSaveOperation:Int{

     case ForCreating,
     case ForOverwriting

 };

let ext = doc.filenameForType(“test”,op:.ForCreating)
```

# Enum

```
enum Rank2
{
    case Ace
    case Num(Int)
    case Jack,Queen,King
    func description() ->String
    {
        switch(self)
        {
        case .Ace:
            return "ACE"
        case .Jack:
            return "JACK"
        case .Queen:
            return "Queue"
        case .King :
            return "King"

        case .Num(let val):
            return String(val)
        }
    }
}
let rank4 = Rank2.Num(3)
rank4.description()
```

# Pattern Matching

```
//pattern matching and binding
let point = (10,10)

switch point
{
case(0,_): //this comes from Haskell
    println("\(point)")

case(0,let y):
    println("\(y)")//this comes from Ruby#{}

case let (x,y) where x==y:
    println("(\(x),\(y))")

default:
   println("default!")
}
```

# Generic

```
func SWAP<T>(inout x:T, inout y:T )
{
    let tmp = x
    x = y
    y = tmp
}

var a = 42
var b = 43

SWAP(&a, &b)
```

# Other things…

- getter/setter
- lazy var
- initialization
- Operator overloading
- Swift <- -> Objective-C
- Swift -> C
- Cocoa APIs
- …

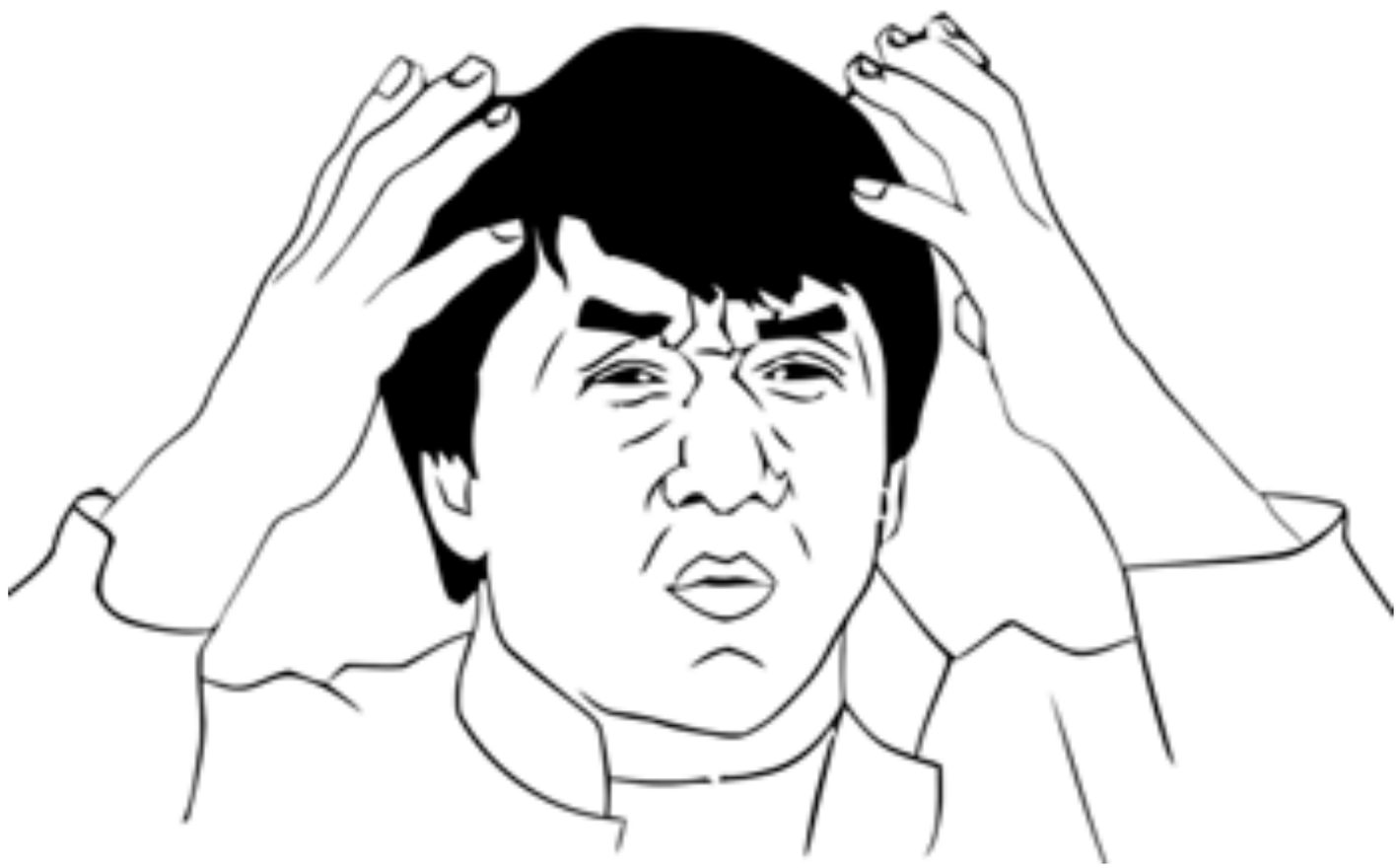
# Evil Part

- @objc
- AnyObject, AnyClass……
- class_getSuperClass()
- __stdlib
- COpaquePointer
- Swift.print()
- var a:UInt!
- …

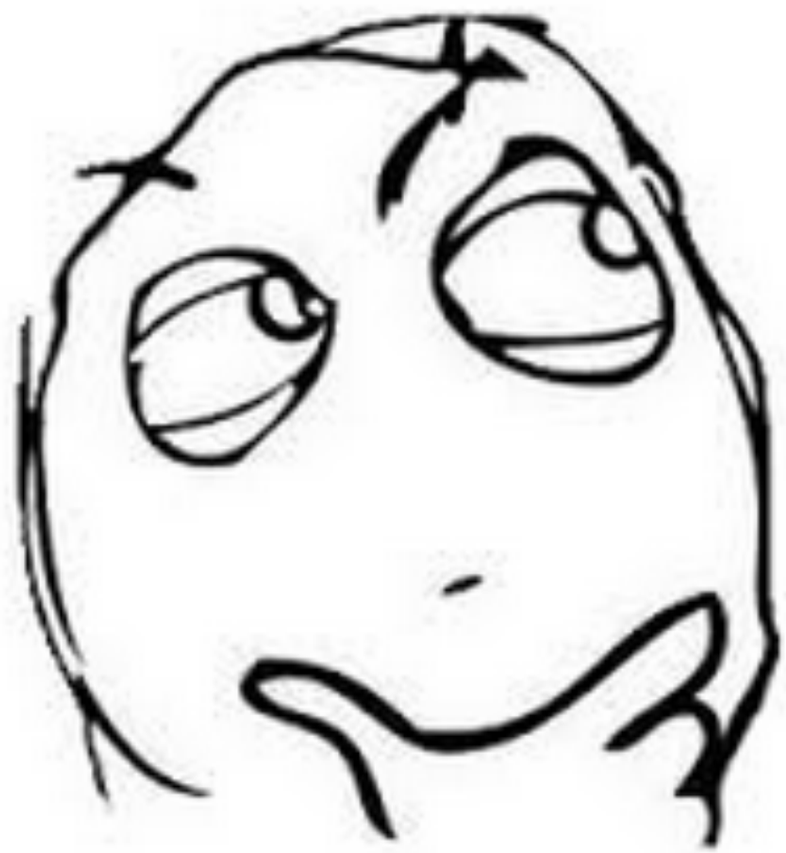
# Wrap up: my thoughts

- Swift API is concise, modern, powerful and safer…
- Quite a strict, safe and strongly-typed language.
- It looks like it's a dynamic language, but It is not!
- Optional type is not a good idea…
- No pointers…
- Functional programming might become popular in Swift.
- Transition from OC to Swift might take a little while…
- Full of dark magic, evil part，undocumented APIs…
- It's a good chance to step into iOS/Mac development…

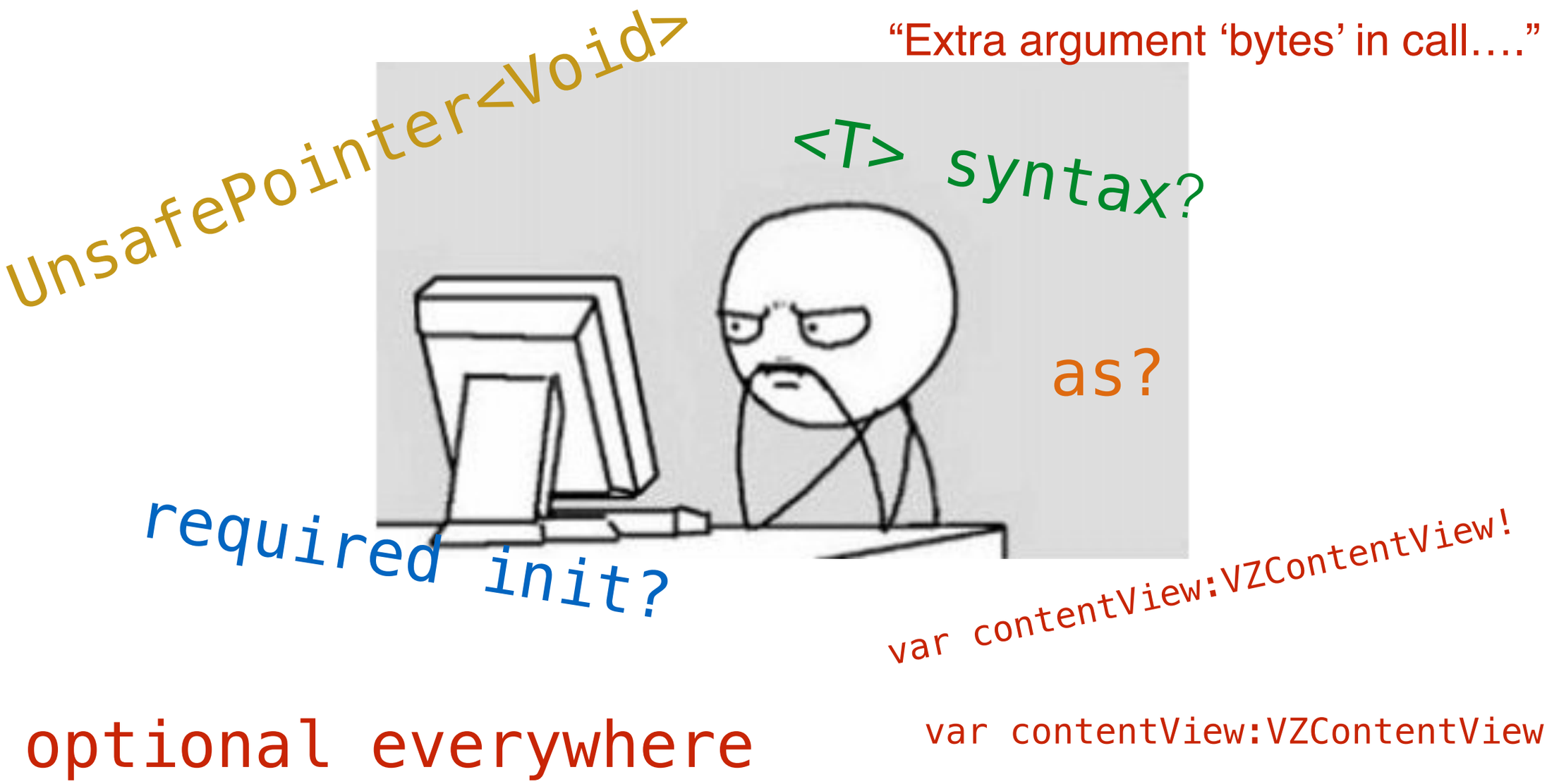
So many stuff…



Let’s try some demo…



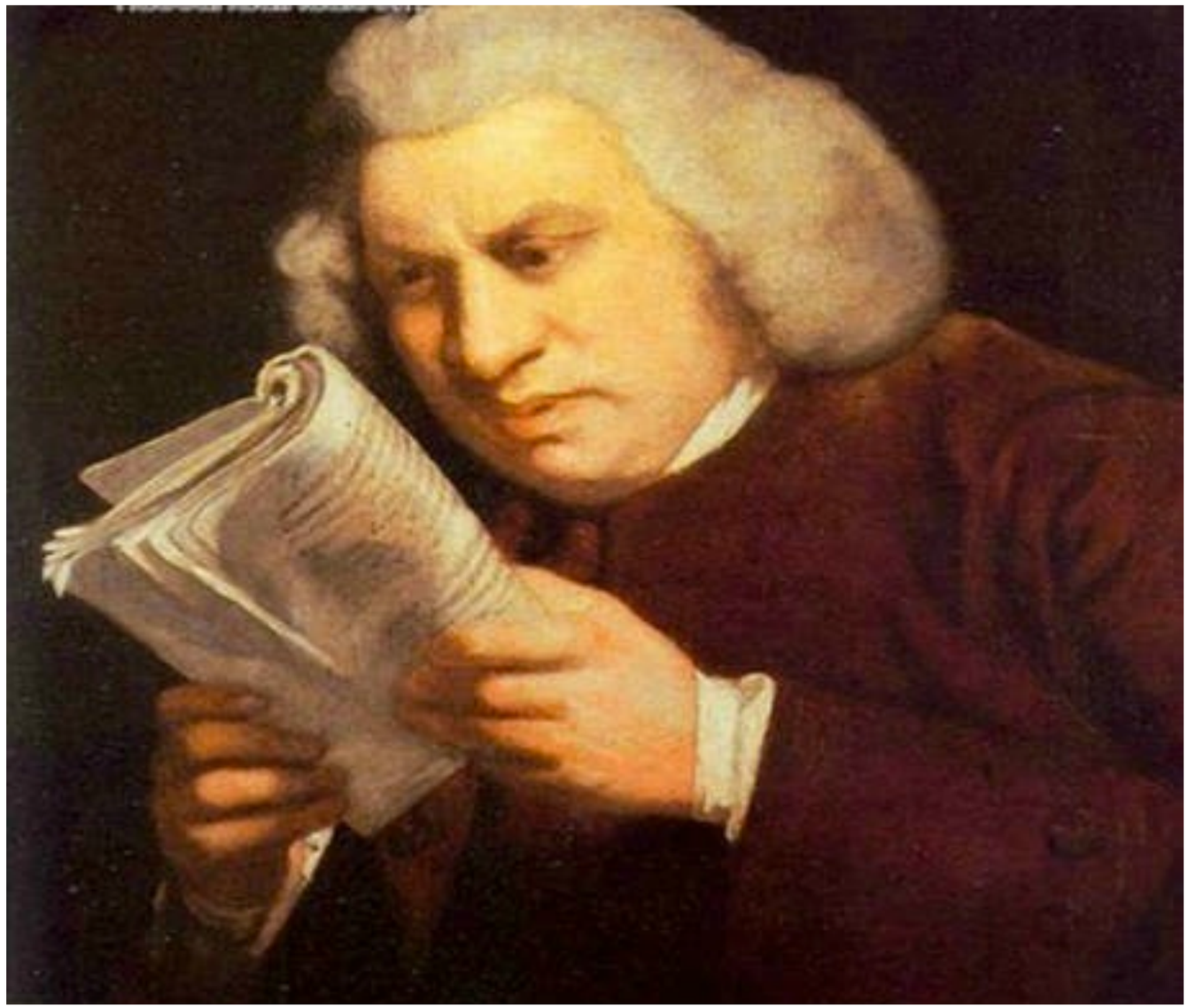
10 minutes…
the compiler just won't let me pass…
why?



20 minutes…

This might be work ?

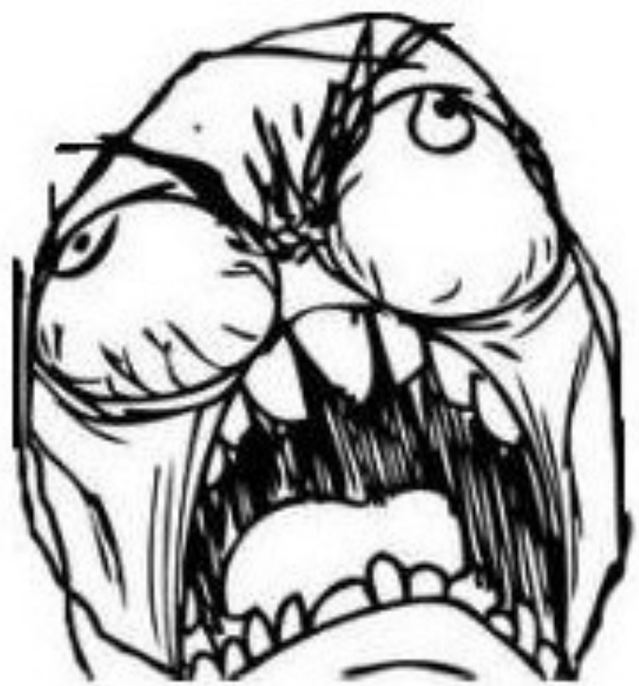
What’s the difference between …?



# 30 minutes…

Use of undeclared type 'CString'

Type 'Pointer' does not conform to protocol 'Equatable'



Expressions are not allowed at the top level

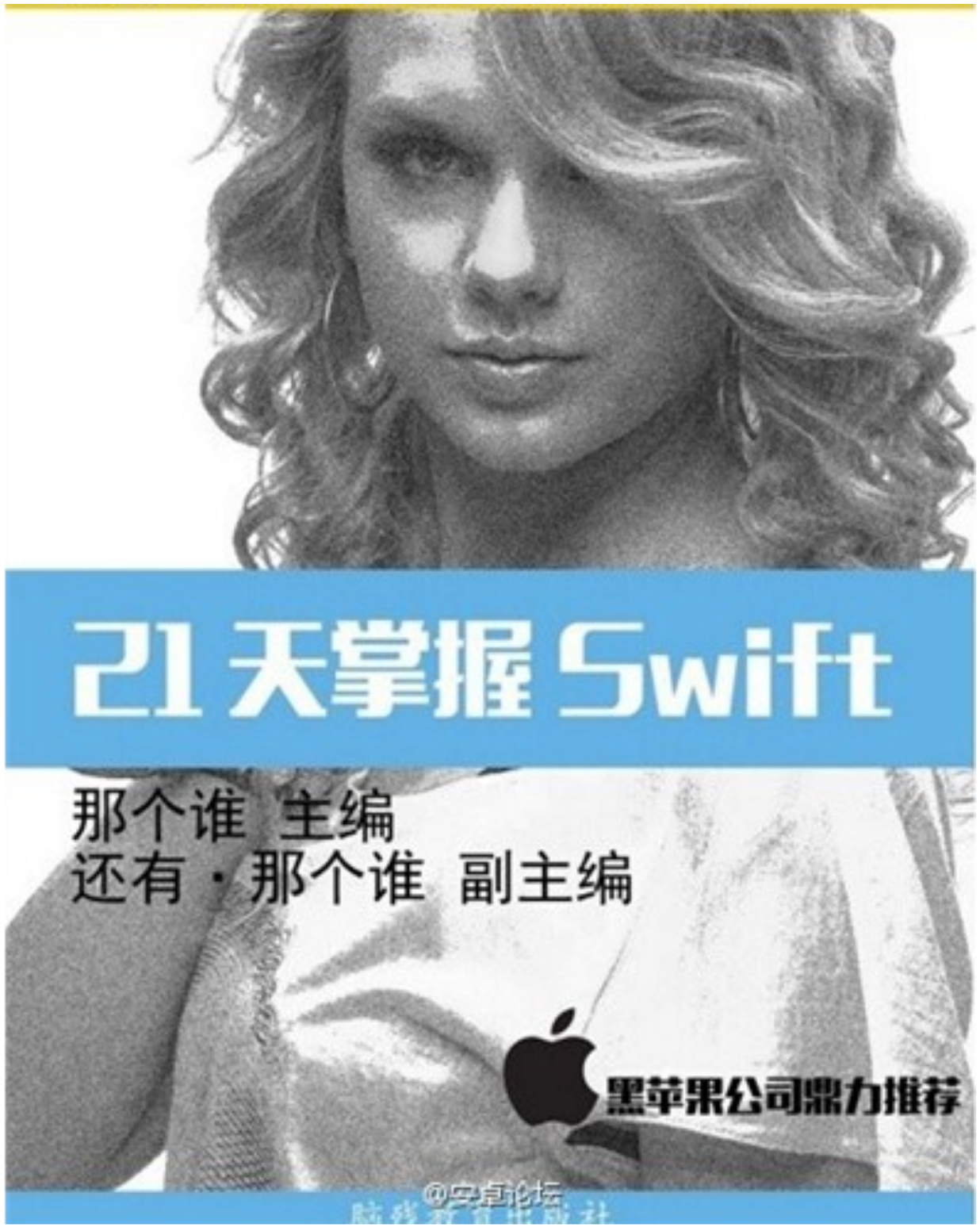
Cannot invoke 'init' with an argument list of type '(AnyClass!, $T3)'

Why there is no pointers?!!!!

What's happening!

# How to get started?

# We wish we could have…



# Getting Started

- WWDC: Swift overview
- Raywenderlich  => for beginers
- Youtube/Tiwtter  =>  tech talks, articles
- blogs: Ash Furrow, Scott …
- Objc.io => swift  under the hood
- Mike Ash => some low level stuff

# End of PART 1

- We’re all new to Swift. Everyone is a beginner
- No one is an expert in Swift
- Don’t be afraid of making mistakes
- The community is growing fast, feel free to ask question
- We should all share what we learn
- You teach , you learn

# PART 2

“These are things I think are useful for iOS programming in general, even if you program in high-level Swift/Objective-C. You could live without them, but knowing them will make you a better iOS programmer.”

# PART 2

- Object Memory Layout
- Where is “objc_class”
- Communication Between Swift Objects
- The Amazing SwiftObject
- Find “vtable”
- Name mangling Rule
- Wrap up: Swift = Objective-C WITHOUT message

# Memory Layout

```
class MySwiftClass
{
    let a : UInt64 = 10
    func method(){ }
}
```

X86_64:

<100d8719 01000000 0c000000 01000000
0a000000 00000000 00000000 00000000>

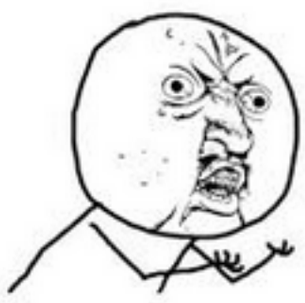
# id is gone

```
typedef struct objc_class *Class;

struct objc_class {
    Class isa  OBJC_ISA_AVAILABILITY;

#if !__OBJC2__
    Class super_class                          OBJC2_UNAVAILABLE;
    const char *name                           OBJC2_UNAVAILABLE;
    long version                               OBJC2_UNAVAILABLE;
    long info                                  OBJC2_UNAVAILABLE;
    long instance_size                         OBJC2_UNAVAILABLE;
    struct objc_ivar_list *ivars               OBJC2_UNAVAILABLE;
    struct objc_method_list **methodLists      OBJC2_UNAVAILABLE;
    struct objc_cache *cache                   OBJC2_UNAVAILABLE;
    struct objc_protocol_list *protocols       OBJC2_UNAVAILABLE;
#endif

} OBJC2_UNAVAILABLE;
```



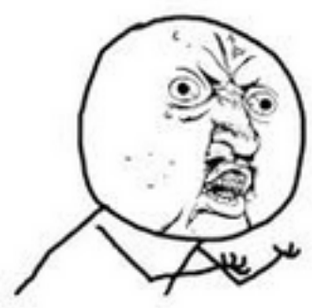
# Instead We have…

# New objc_class

```
struct objc_class
{
    uint64_t isa;
    uint64_t superclass;
    uint64_t cache;
    uint64_t vtable;
    uint64_t data;
};
```

# Message is Dead

~~id objc_msgSend(id self, SEL op, …)~~



```
SEL sel = @selector(onScrollViewClicked:);

IMP imp = [self.delegate methodForSelector:sel];

((void(*)(id,SEL,...))imp)(self.delegate,sel,nil);
```

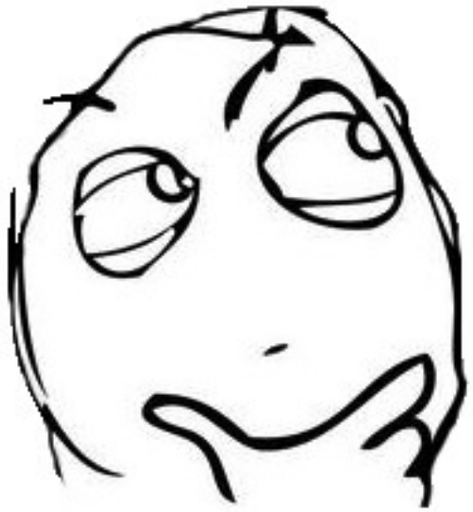
# Instead We have…

# vtable

```
method_imp = obj -> class.vtable[0]

method_imp()
```

Is this faster than `objc_msgSend` ?



# How about this?

```
MySwiftClass* s = [MySwiftClass new];

if([s responseToSelector(@selector(method))])
{
    [s method];
}
```

Where is responseToSelector in MySwiftClass ?

# SwiftObject

```
class:Optional("TestSwiftRuntime.MySwiftClass")
superclass:Optional("SwiftObject")
ivar:Optional("magic")  type:Optional("{SwiftObject_s=\"isa\"^v\"refCount\"q}")
ivar:Optional("a") type:Optional("")
property:Optional("hash") type:Optional("TQ,R")
property:Optional("superclass") type:Optional("T#,R")
property:Optional("description") type:Optional("T@\"NSString\",R,C")
property:Optional("debugDescription") type:Optional("T@\"NSString\",R,C")
method:Optional("zone") type:Optional("^{_NSZone=}16@0:8")
method:Optional("doesNotRecognizeSelector:") type:Optional("v24@0:8:16")
method:Optional("description") type:Optional("@16@0:8")
method:Optional(".cxx_construct") type:Optional("@16@0:8")
method:Optional("retain") type:Optional("@16@0:8")
method:Optional("release") type:Optional("v16@0:8")
method:Optional("autorelease") type:Optional("@16@0:8")
method:Optional("retainCount") type:Optional("Q16@0:8")
method:Optional("dealloc") type:Optional("v16@0:8")
method:Optional("isKindOfClass:") type:Optional("B24@0:8#16")
method:Optional("hash") type:Optional("Q16@0:8")
method:Optional("isEqual:") type:Optional("B24@0:8@16")
method:Optional("_cfTypeID") type:Optional("Q16@0:8")
method:Optional("respondsToSelector:") type:Optional("B24@0:8:16")
method:Optional("self") type:Optional("@16@0:8")
method:Optional("performSelector:") type:Optional("@24@0:8:16")
method:Optional("performSelector:withObject:") type:Optional("@32@0:8:16@24")
method:Optional("conformsToProtocol:") type:Optional("B24@0:8@16")
method:Optional("performSelector:withObject:withObject:") type:Optional("@40@0:8:16@24@32")
method:Optional("isProxy") type:Optional("B16@0:8")
method:Optional("isMemberOfClass:") type:Optional("B24@0:8#16")
method:Optional("superclass") type:Optional("#16@0:8")
method:Optional("class") type:Optional("#16@0:8")
method:Optional("debugDescription") type:Optional("@16@0:8")
```

# vtable

xcrun swiftc -emit-library -o test -

xcrun nm -g test

```
__TFC16TestSwiftRuntime12MySwiftClass6methodfS0_FT_T_
__TFC16TestSwiftRuntime12MySwiftClassCfMS0_FT_S0_
__TFC16TestSwiftRuntime12MySwiftClassD
__TFC16TestSwiftRuntime12MySwiftClasscfMS0_FT_S0_
__TFC16TestSwiftRuntime12MySwiftClassd
__TFC16TestSwiftRuntime12MySwiftClassg1aVSs6UInt64
__TFC16TestSwiftRuntime12MySwiftClassm1aVSs6UInt64
__TFC16TestSwiftRuntime12MySwiftClasss1aVSs6UInt64
```

# Name mangling

`__TFC16TestSwiftRuntime12MySwiftClass6methodfS0_FT_T_`

- Swift中所有的符号都以_开头。
- _T 用来标识这个符号是全局的。
- F 用来标识这个符号代表的是一个函数。
- C 代表它是从属于一个类的方法。
- 16TestSwiftRuntime代表module name，16为字符长度。
- 12MySwiftClass代表类名，12为字符长度。
- 6method代表方法名称，6为字符长度。
- f代表这个方法是uncurried function，第一个参数是一个隐式参数self。
- S0_FT_T_不确定，按照规律来看应该是用来标识入参和返回值的。

# Further Topic

- Swift in Cocoa/Cocoa Touch
- Functional programming in Swift
- Why Swift is faster?

We should all share
what we learn

# Resource

- PPT:https://github.com/akaDealloc/blog/tree/gh-pages/pdf
- Swift demo code:https://github.com/akaDealloc/blog/tree/gh-pages/code/swift/swift-basic
- Blog:http://akadealloc.github.io/blog/

@end

# Q&A