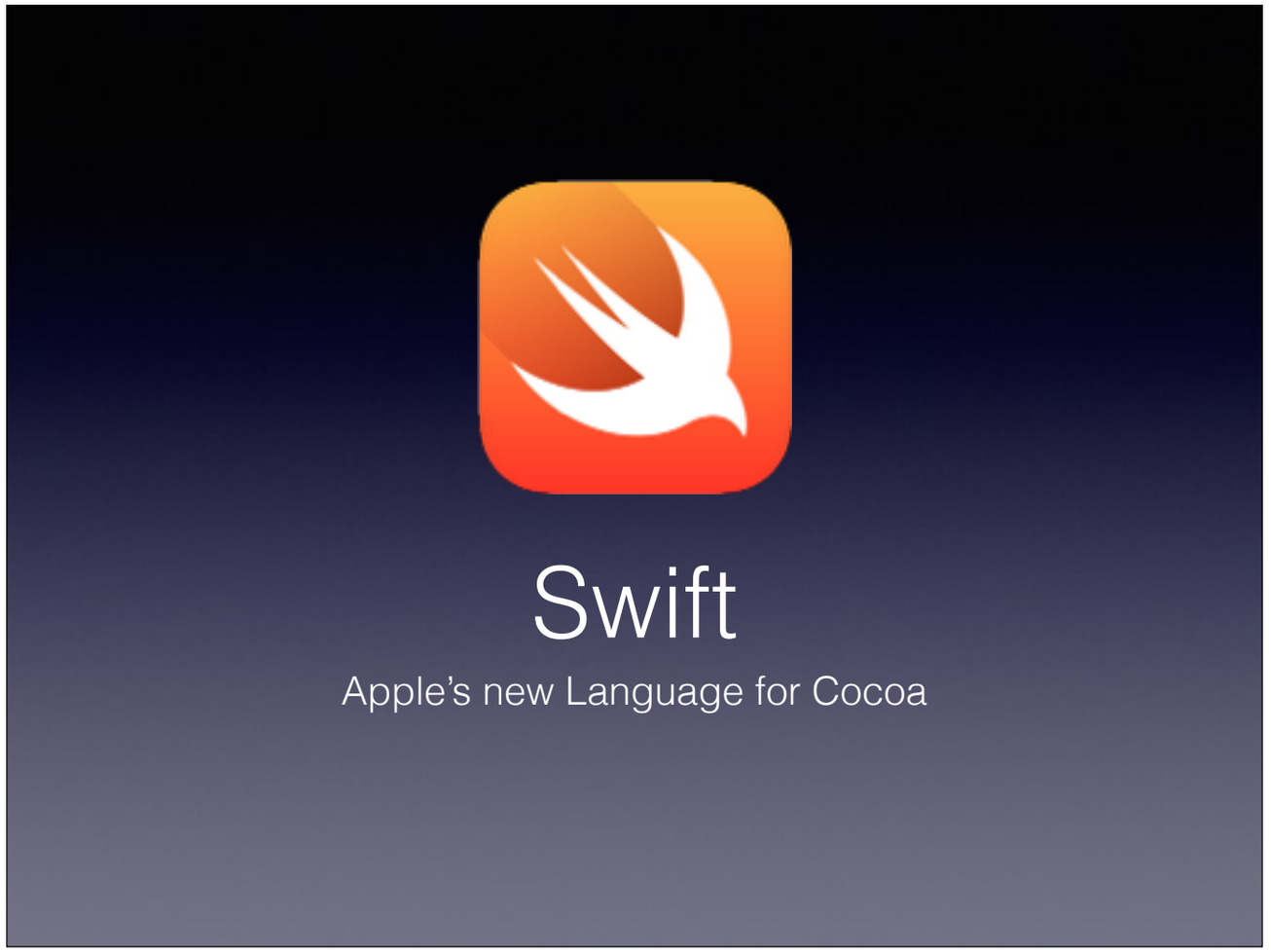
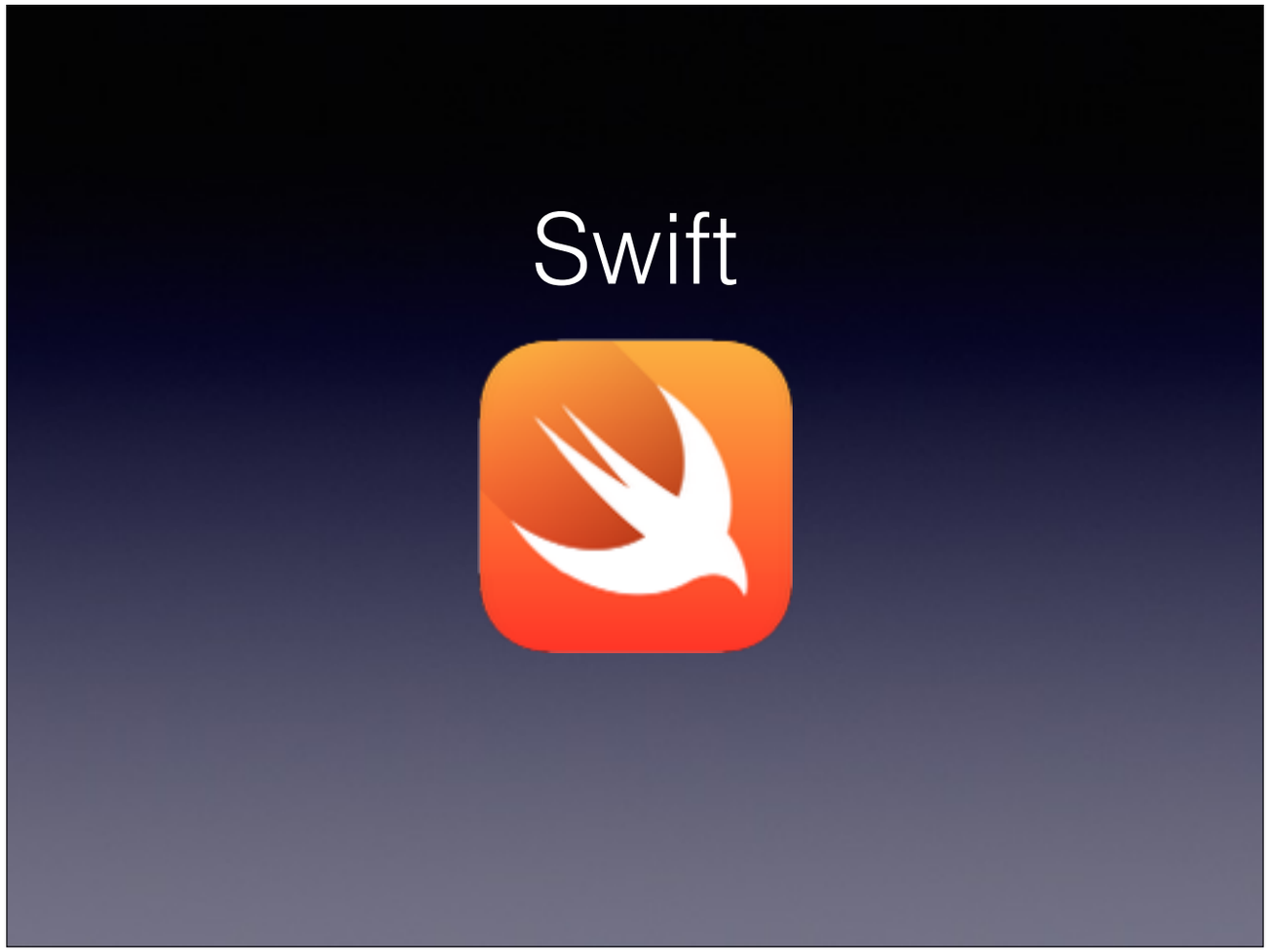


Swift

Apple's new Language for Cocoa

Hi I’m Paul Wood

I’m the Lead Developer @ Transmedia Creative Lab
where I make shopping apps

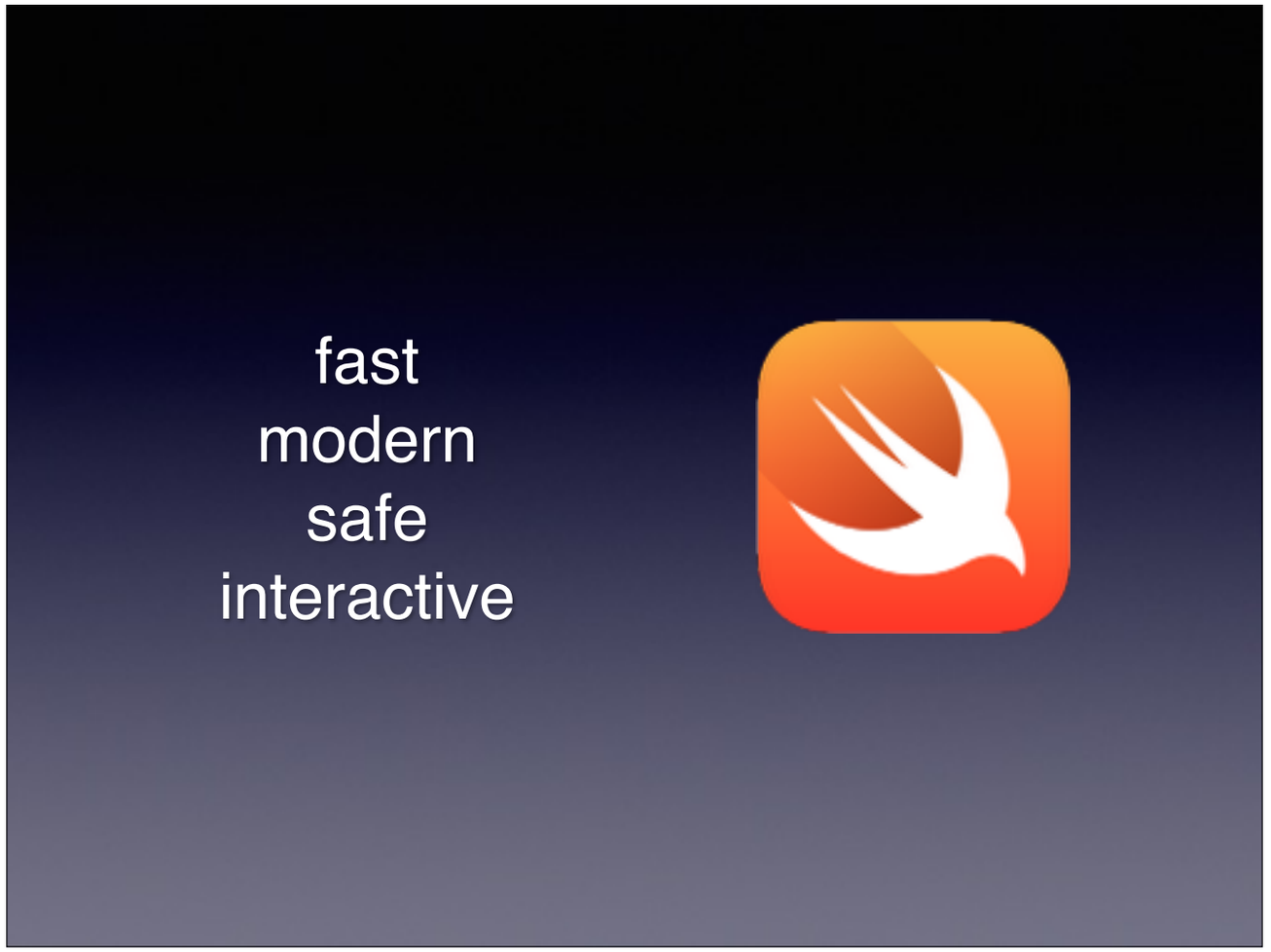


Swift was released last monday at WWDC

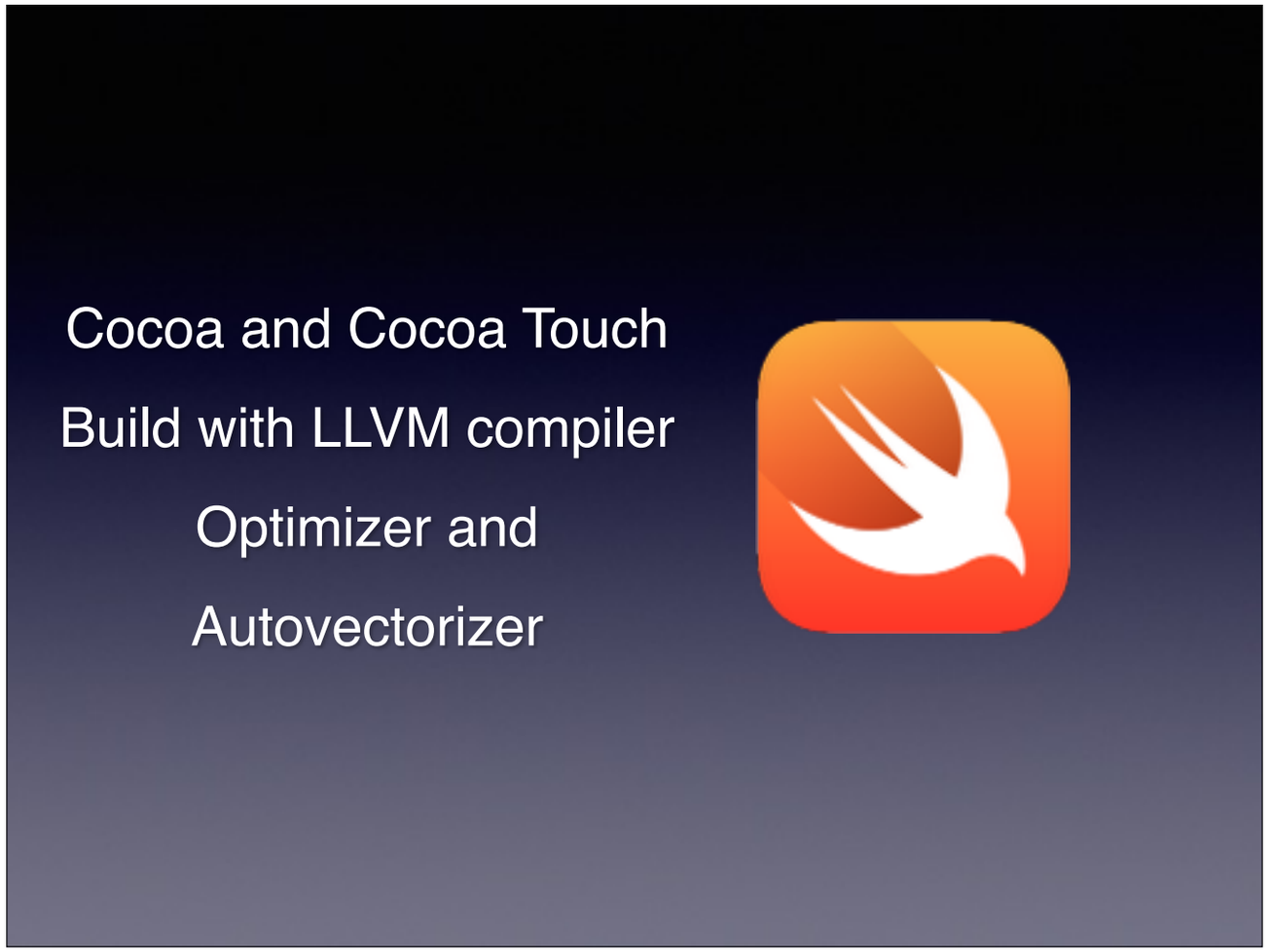
It’s really new and there is a lot more to be learned than what has been presented at WWDC

If you have watched all the WWDC video on Swift and already read the Language book then you may have a head start on me. If you’ve gotten in to writing code you are also doing better than me

I’m trying to make this presentation similar to the lectures I received in College about new languages, I took a programming languages course and had to break down nearly every popular language since ADA



To Apple Says Swift is these 4 things, and after reading all the published goodies out there I’m going to back up their claim

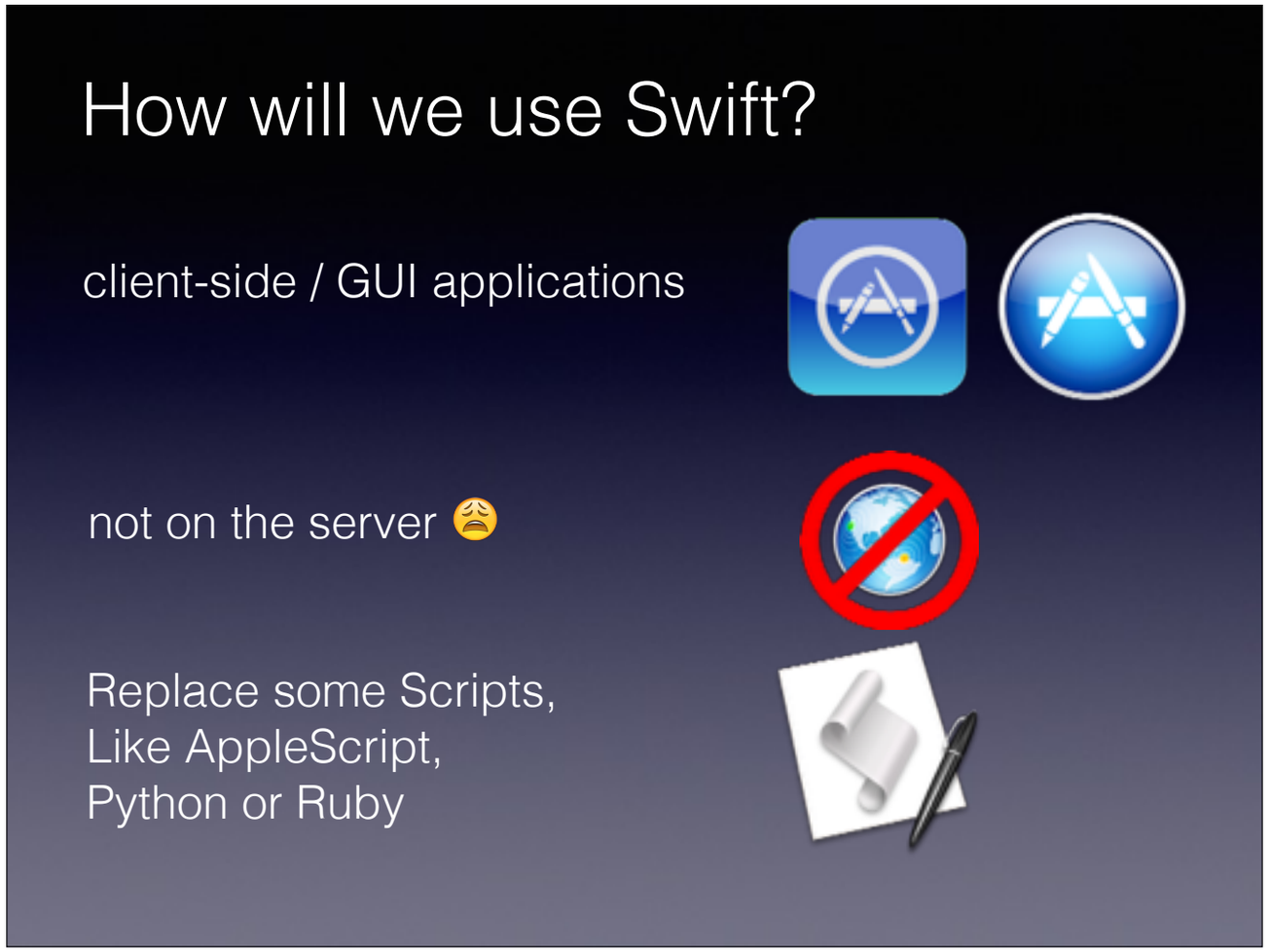


We as Cocoaheads need to know how its going to be useful in our current projects and Apple has built with with Cocoa development in mind. Almost purely Cocoa development in mind.

I’m going to talk about LLVM a little later

But incase you forgot a optimizer takes your readable code and puts in better algorithms on your behalf and Autovectorizer helps you with multiple threading, parallel computing.

Its complicated stuff but just remember that it is there to make Swift fast while also being readable and concise



Swift is really all about Cocoa

For right now I expect that developer will only work in Swift for client side Graphical user interface based applications

Its not a scripting language, it doesn't have an interpreter, it isn't going to be replacing PHP or Ruby, There are very few tools in the language or the Cocoa libraries to get this on the server.

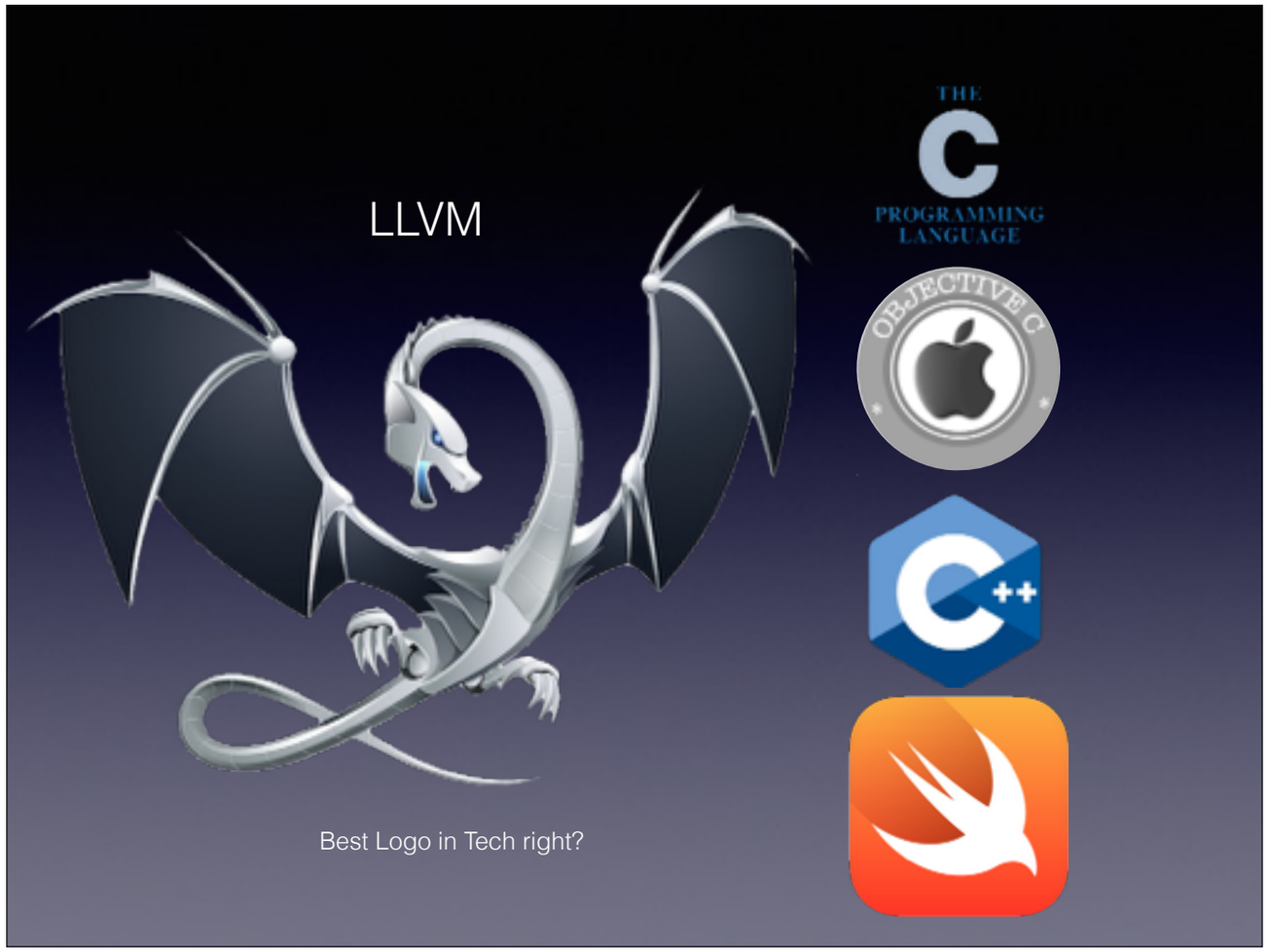
But you can replace scripts with Swift. Since its readable its a good first language. We will get into that more later.

Apple's Language Ecosystem

So why did Apple bother with creating Swift? Objective-C is currently the number 3 language in the world according to Tiobe a tracker of such things.

(http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

I think its understandable when you look at the stack of languages Apple was working with



So lets start off with C

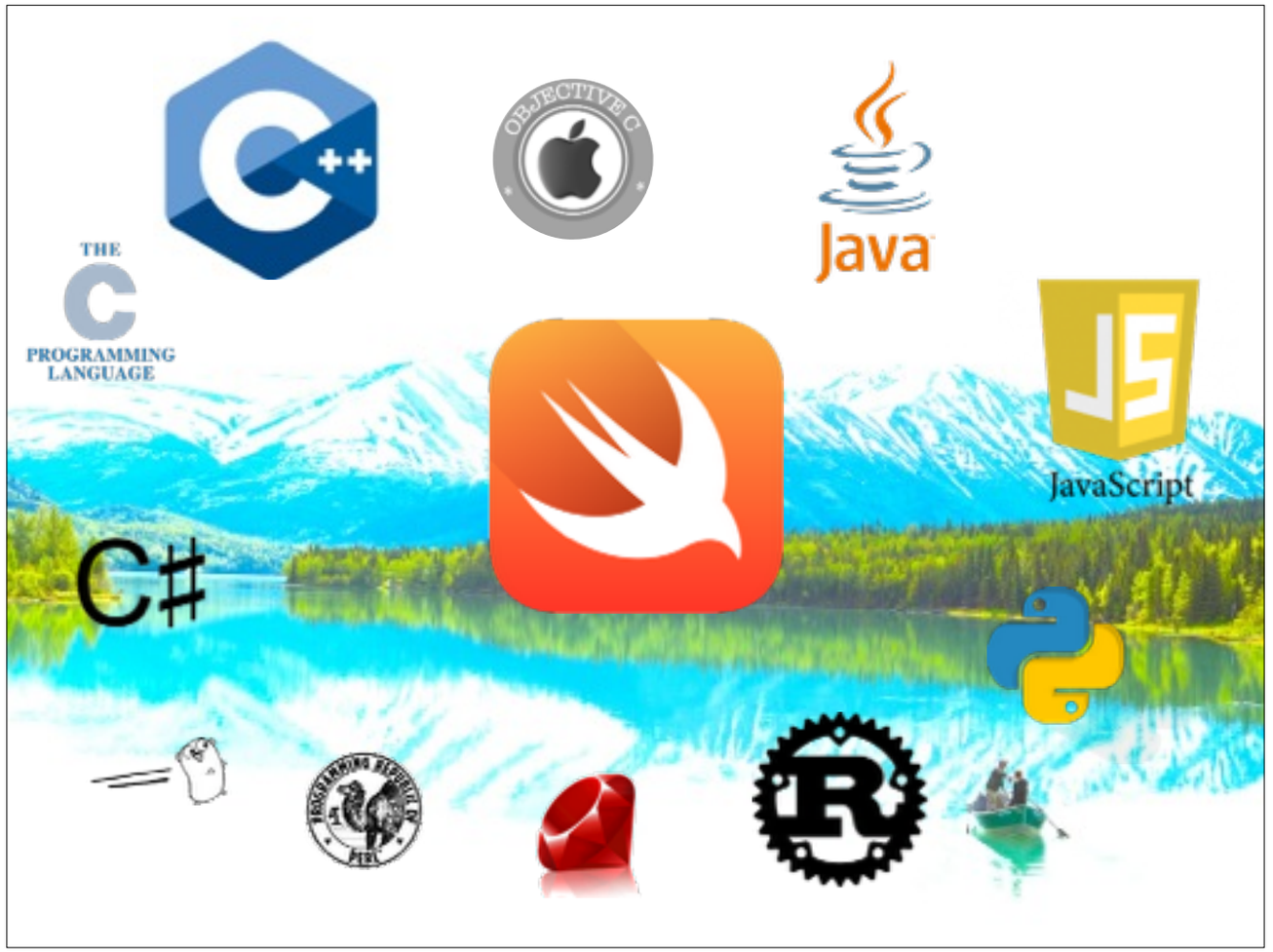
No Objects, highly functional, Pointers everywhere, very close to the metal, you can throw in assembly whenever you want to

Then Objective- C which is a superscript of C with Objects. Its getting better but still has all the problems of C like pointers. its also Old, not as old as C but still old. This language is all Apple as well and was brought into the company when Steve Jobs came back to the company after Apple bought NextStep.

C++ isn't used by many of us but its important for gaming and lets face it, C++ is a great language. It was my preference in College. We know this isn't Apple's go to language, Obj-C is, just a language they support because some very important industries rely on C++

Now there is Swift and no pointers and a whole lot more.

All of this is being compiled in Xcode with the LLVM compiler.



Influences!

C family with the parens

java with that dot syntax

Scripting languages like javascript or ruby with some of the iteration syntax, also Swift has no main function

I've heard Rust, and I've heard Go! but I don't know these well enough to explain

LLVM is Awesome

- GCC compiled code was faster only a year ago (and sometimes still is)
- LLVM is about modules that make up a compiler
- LLVM compiles things faster than GCC
  - Which means Swift can look like a scripting language, examples being XCode Playgrounds and REPL
- Just-In-Time Compilation
  - Compile things only when you need them, Its why Safari javascript is now faster than Google Chrome's implementation of js

Devs out there, you really should browse http://llvm.org sometime

So Back to Apple's baby LLVM

Sprang out as optimizations of the GCC compiler, thats the GNU C compiler project.

GCC was a great compiler and everyone was using it, things looked good for this open software.

LLVM is about modules that make up a compiler, you can pick and choose what and where you want to optimize.

Because of the modules you can perform some tricks at compile time (which when you think about it is also run at someones runtime) and things get compiled faster.

Compiling faster is one thing, but the performance of that compiled code matters as well

LLVM now produces code that is slightly faster than GCC, but that wasn't true for a while

Chris Lattner

Created LLVM
Apple's Director of Developer Tools
Worked 4 years in creating Swift
He makes your IDE, Xcode!

All of this is the brainchild of Chris Lattner,

As a student he created the LLVM compiler, he stuck with it and got hired by Apple

He has risen in the ranks is now the Director of Developer Tools

He created Swift because of his work the the LLVM compiler the two work very closely together.

He is in charge of your Xcode so when your Xcode 6 Beta crashes this summer feel free to curse his name.

So

From here on out I'll be giving you guys some code samples of Swift instead of a history lesson

```swift
// this is a single line comment using two slashes

/* this is also a comment,
but written over multiple lines */

// Swift variables are declared with "var" followed by a name, a type, and a value
var explicitDouble: Double = 70

/// if the type is omitted, Swift will infer it from the variable's initial value
var implicitInteger = 70
var implicitDouble = 70.0

// the "let" keyword defines a constant, which is also implicitly typed
let numberOfApples = 3
let numberOfOranges = 5
let appleSummary = "I have \(numberOfApples) apples."
let fruitSummary = "I have \(numberOfApples + numberOfOranges) pieces of fruit."

// code can be placed anywhere, making it global within the namespace
println("Hello, world")

// Swift cleanly hides NSArray and NSDictionary collection classes within internal language syntax.
// in this case we define a dictionary with four items, each with a person's name and age
let people = ["Anna": 67, "Beto": 8, "Jack": 33, "Sam": 25]

// now we use Swift's flexible enumerator system to extract both values in a single loop
for (name, age) in people {
    println("\(name) is \(age) years old.")
}

// methods and functions are declared with the "func" syntax
// note the use of parameter naming, and the return type specified with ->
func sayHello(personName: String) -> String {
    let greeting = "Hello, " + personName + "!"
    return greeting
}

// prints "Hello, Jane!"
println(sayHello("Jane"))
```

Heres a quick example of some Swift

From a purely cursary look things look like Objective-C

What I notice are the things it doesn’t have, the things Apple trimmed from Objective-C

notice no Main function

no semi-colons

No @ infront of your strings !

Solves Major Problems of C

- No Pointers (and we all know pointers suck)
- Assignments do not return a value
- No need to use break statements in switch blocks
- Overflows are trapped as a run-time errors

```
1 var f = 0
2 if (f=0){
3     //Doesn't Work😢!
4 }
```

```
1 var f = 0
2 if (f==0){
3     //Works😃!
4 }
```

```
let vegetable = "red pepper"
switch vegetable {
case "celery":
    let vegetableComment = "Add some raisins and
        make ants on a log."
case "cucumber", "watercress":
    let vegetableComment = "That would make a good
        tea sandwich."
case let x where x.hasSuffix("pepper"):
    let vegetableComment = "Is it a spicy \(x)?"
default:
    let vegetableComment = "Everything tastes good
        in soup."
}
```

```
var potentialOverflow = Int16.max
// potentialOverflow equals 32767, which is the
        largest value an Int16 can hold
potentialOverflow += 1
// this causes an error
```

Why did Chris make Swift, if You like John Siracusa you'd know pointers Suck and thats why Apple needed this language

Assignments in C do really funny things like returning true and therefore we had some lines of code that wrecked our app and took us hours to find

C is word, and has semicolons everywhere and isn't natural to read

And then there are overflows which I don't want to get into but when this happens in your Augmented reality code you'll hate yourself

Concise Readable Code

- Static Things and dynamic things are easy to differentiate
- Type Inference
- But you can declare types if you really want to
- Collections have great ‘for-in’ syntax using tuples

```
let foo = 1
let bar = 1.0
let baz = "one"
```

```
let foo :Int = 1
let bar :Double = 1.0
let baz :String = "one"
```

```

let people = ["Anna": 67,
   "Beto": 8, "Jack": 33,
   "Sam": 25]

for (name, age) in people {
    println("\(name) is \(age)
      years old.")
}
```

Apple said concise, readable code, this isn’t Ruby or Wolfarm Alpha speak while writing but it is pretty good looking as a Cocoa developer.

Types don’t needs to be declared but can be

We keep our prettier colon

Tuples make life easier for readability as well

Multiple Return Types

```swift
// Return the # of character types in a string
func count(string: String) ->
    (vowels: Int,
    consonants: Int,
    others: Int)
    // Return value is a tuple
{
    var vowels = 0, consonants = 0, others = 0
    // Logic Here
    return (vowels, consonants, others)
}
```

Objective-C & C didn’t have this and it made for some really awkward code, Think about the structs and dictionaries we’ve created over the years just to return more than one thing

A major problem with Obj-C was you didn’t have multiple return types

Swift adds this

Think about your hacks

First Class Functions

- Functions are a first-class type. This means that a function can return another function as its value.
- Closures are just like your Obj-C blocks, in fact you can use your old Obj-C block based APIs with closures

```swift
func makeIncrementer() -> (Int -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

http://en.wikipedia.org/wiki/First-class_functions

Functions can be wrapped up and taken on the go in Swift.

If you maintain open source code, don't go crazy with this. I don't still don't understand how most Javascript is read. I think I speak for a lot of developers that FuckingBlocksyntax.com is one of my go to sites

Then again maybe we can learn and if you watch the WWDC videos you can create some great algorithms with this syntactic spice. In Advanced Swift using the Memoable function was mind blowing.

Generics

```swift
func swapTwoInts(inout a: Int, inout b: Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}

func swapTwoValues<T>(inout a: T, inout b: T) {
    let temporaryA = a
    a = b
    b = temporaryA
}

var someInt = 3
var anotherInt = 107
swapTwoValues(&someInt, &anotherInt)
// someInt is now 107, and anotherInt is now 3

var someString = "hello"
var anotherString = "world"
swapTwoValues(&someString, &anotherString)
// someString is now "world", and anotherString is now
"hello"
```

Ever wanted to make a category on all those NS Containers like NSArray and NSDictionary
Generics make it happen.

look at that T there for the type, you can also add that T must conform to a protocol so that should help your code clarity as well

This should save a lot twice written code that was in a lot of our Cocoapods

Extensions

```
extension UIBezierPath {
    convenience init(triangleSideLength: Float, origin: CGPoint) {
        self.init()
        let squareRoot = Float(sqrt(3))
        let altitude = (squareRoot * triangleSideLength) / 2
        moveToPoint(origin)
        addLineToPoint(CGPoint(triangleSideLength, origin.x))
        addLineToPoint(CGPoint(triangleSideLength / 2, altitude))
        closePath()
    }
}
```

Need your Cocoa to do something unexpected?

Categories was how you did it in Obj-C now you have extensions and Extensions do a lot more, you can put in more places than just classes in Swift

? and !
Optional and Unwrap

```
    var optionalInteger: Int?
    optionalInteger = 42
    optionalInteger! // 42

     var window: UIWindow?

     func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: NSDictionary?) -> Bool {
         // Override point for customization after application launch.
         var vc = ViewController()
         var nav   = UINavigationController(rootViewController:vc);
         var rect = UIScreen.mainScreen().bounds;
         self.window = UIWindow(frame: rect);
         self.window!.rootViewController = nav;
         self.window?.makeKeyAndVisible();
         return true
    }
```

Heres a common CocoaTouch function, app did finish launching with Options!

These two characters are going to be everywhere in your Swift Code when working with Cocoa

Now Look at how you set up the window, the ! checks the property to make sure the window responds to this selector and the question mark checks for a function that should exist on makeKeyVisible.

Using the ! operator to unwrap an optional that has a value of nil results in a runtime error. so you gotta be sure about things

A lot of selectors in Obj-C has a syntax that allows one of the parameters to be nil, this ? mark does the check for it it is nil AND ask if the object passed responds to the selector, a

Anonymous _

```
let (justTheStatusCode, _) = http404Error
println("The status code is \(justTheStatusCode)")
// prints "The status code is 404"
```

For when you just don’t care you can tell the compiler that, this means that autovectorizer and code optimizer can make your code run faster

Subscripts

```
struct TimesTable {
    let multiplier: Int
    subscript(index: Int) -> Int {
        return multiplier * index
    }
}
let threeTimesTable = TimesTable(multiplier: 3)
println("six times three is \(threeTimesTable[6])")
// prints "six times three is 18"
```

We oftentimes program our own collection objects. If you know how these objects should be described as if in a list then use Subscripts.

This can lead to great weirdness if you are not careful, but also great clarity to you the developer.

Protocols

```
protocol SomeProtocol {
    // protocol definition goes here
    class var someTypeProperty: Int { get set }
    class func someTypeMethod()
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

struct SomeStructure: FirstProtocol, AnotherProtocol {
    // structure definition goes here
}

class SomeClass: SomeSuperclass, FirstProtocol, AnotherProtocol {
    // class definition goes here
}

protocol Togglable {
    mutating func toggle()
}

enum OnOffSwitch: Togglable {
    case Off, On
    mutating func toggle() {
        switch self {
        case Off:
            self = On
        case On:
            self = Off
        }
    }
}
var lightSwitch = OnOffSwitch.Off
lightSwitch.toggle()
```

Cocoa has so many protocols that we use every day, I'de bet all my view controllers conformed to at least one, its a great was to keep your code generic and modular.

Swift brings the greatness of protocols to more than classes, add it to a struct or even and enumeration!

Look at that toggle on the enumeration!

Adding New Operators

```swift
// Basic Operator overriding the plus sign
struct Vector2D {
    var x = 0.0, y = 0.0
}

@infix func + (left: Vector2D, right: Vector2D) -> Vector2D {
    return Vector2D(x: left.x + right.x, y: left.y + right.y)
}

let vector = Vector2D(x: 3.0, y: 1.0)
let anotherVector = Vector2D(x: 2.0, y: 4.0)
let combinedVector = vector + anotherVector
// combinedVector is a Vector2D instance with values of (5.0, 5.0)

// Advanced Custom Defined Operator
operator prefix +++ {}

@prefix @assignment func +++ (inout vector: Vector2D) -> Vector2D {
    vector += vector
    return vector
}

var toBeDoubled = Vector2D(x: 1.0, y: 4.0)
let afterDoubling = +++toBeDoubled
// toBeDoubled now has values of (2.0, 8.0)
// afterDoubling also has values of (2.0, 8.0)
```

I didn’t think you could do this in any language, because I’m so use to using functions with parens to do things to my objects

now swift can override an operator to make my code more readable

Also look at the triple plus

Lets use it with Cocoa

```swift
import UIKit
class MyViewController: UIViewController {
    let myButton = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 50))

    init(nibName nibNameOrNil: String!, bundle nibBundleOrNil: NSBundle!) {
        super.init(nibName: nibName, bundle: nibBundle)
        myButton.targetForAction("tappedButton:", withSender: self)
    }

    func tappedButton(sender: UIButton!) {
        println("tapped button")
    }
}
```

your sift classes can be a subclass of objective-c classes.

This first blew my mind but when you think about why Apple wrote Swift then its an obvious feature of the language.

Swift runs in the Objective-C runtime after all

Theres also toll free bridging meaning a swift array is the same as a NSArray, can thats down to the memory!

Why you need to start working with Swift now

Objective-C has been Apple's 'main' programming language for 17 years! So I expect Swift to last the same length of time

Its in Alpha now, so your feedback now will affect the language you work with for the rest of your career as a developer

Swift already works well with your current Objective-C code

Time!

- 2048 (a threes clone)
- Flappy Bird
- Cocoapods is getting ready

- The Swift Programming Language iBook
  - or on the web
- Using Swift with Cocoa and Objective-C