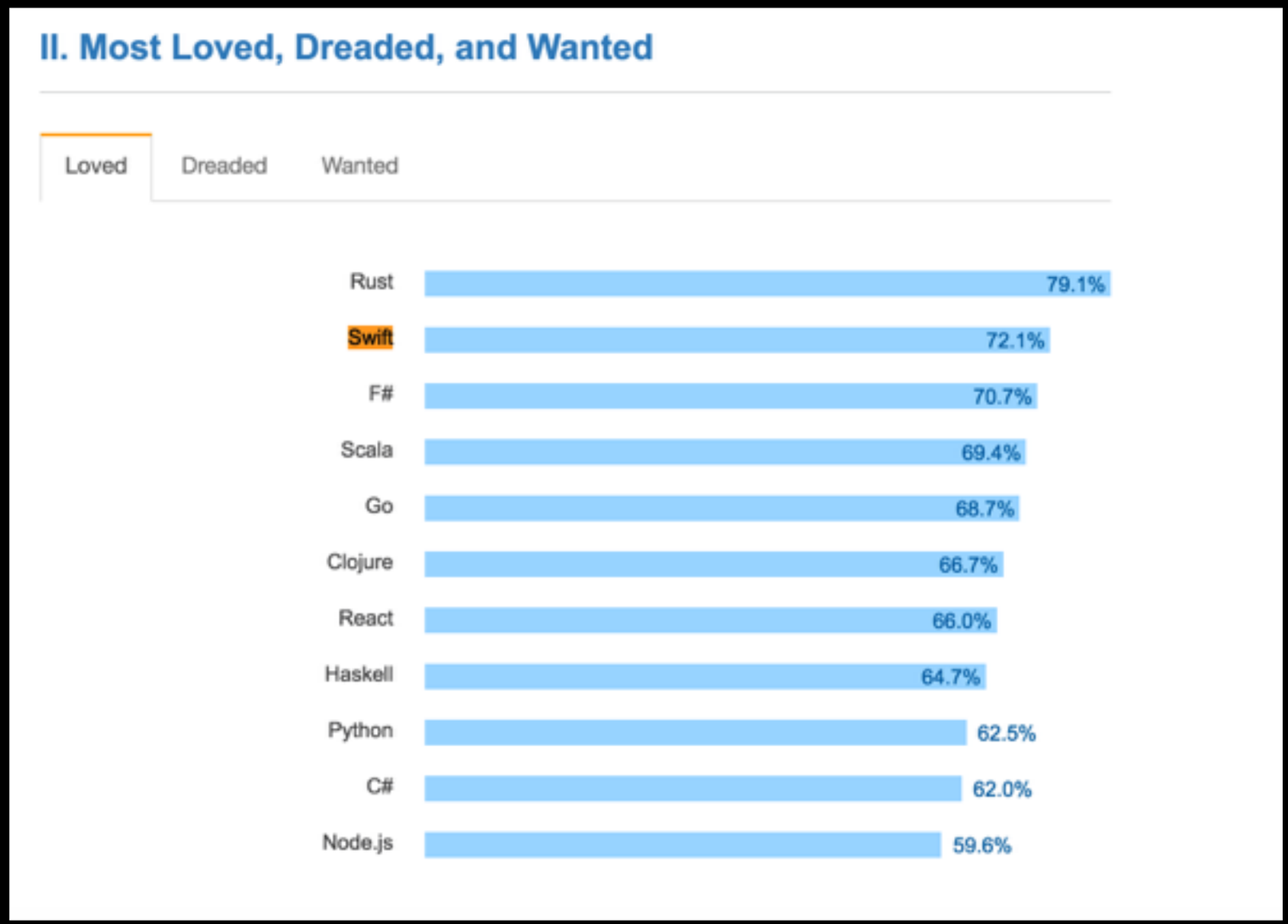
A Glance At Swift

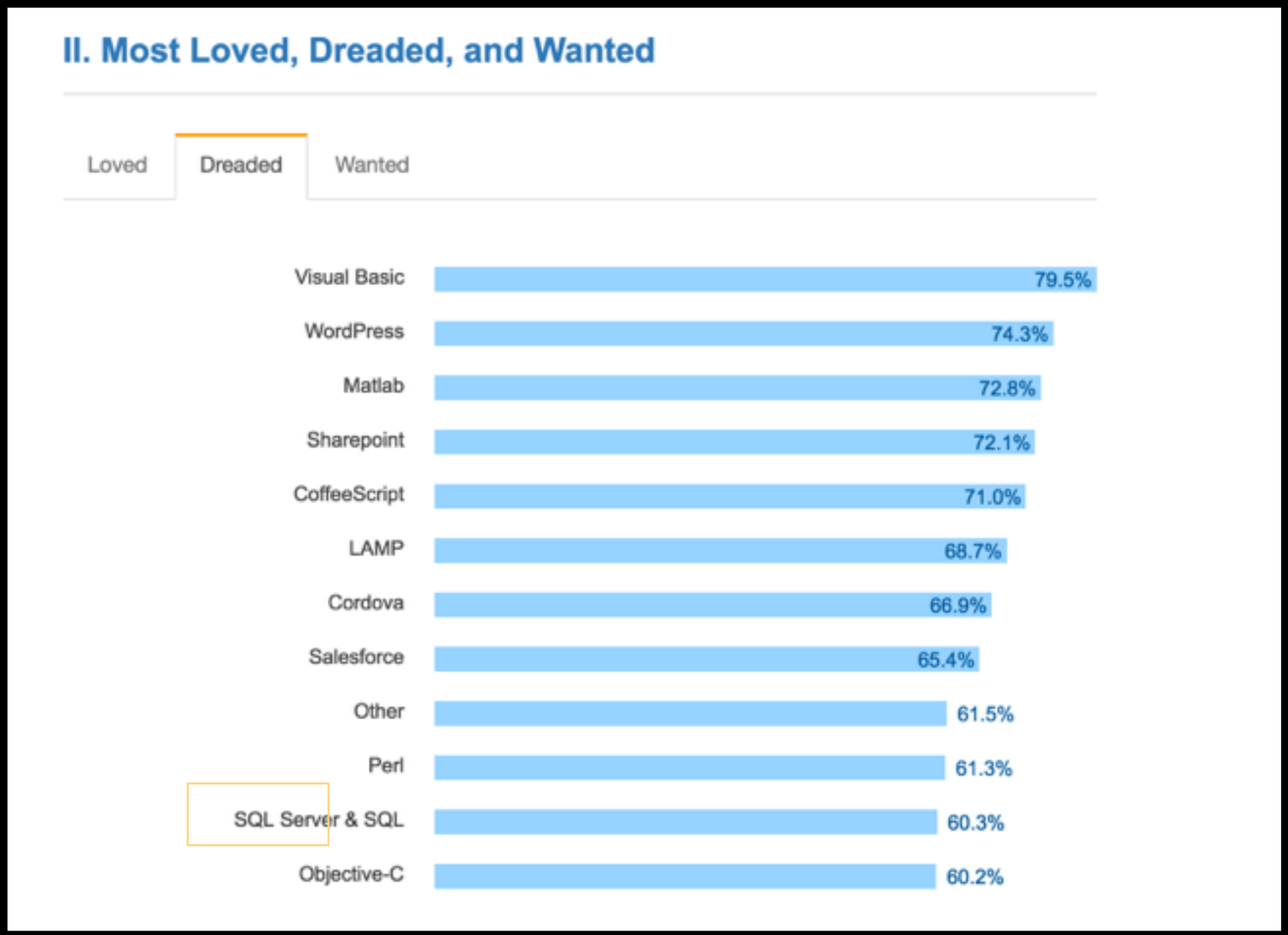
主讲人：黄智辉

Why Swift ?



Stack Over Flow,2016最受喜欢的语言

Why Swift ?



Stack Over Flow,2016最恐怖的语言

Agenda

1. Swift Style

    1.1 简约
    1.2 安全
    1.3 快速

2. Types

    2.1 Struct
    2.2 Protocol
    2.3 Enum
    2.4 Function

3. Sever Side

    3.1 Vapor

简约

Objective-C

```objc
#import "WBOTCalendarViewController.h"
#import "WBOTTabBarController.h"
#import "WBOTAddRecordViewController.h"
#import "WBOTDateCacheCenter.h"
#import "WBOTSettingSalaryViewModel.h"
#import "WBOTAddRecordViewModel.h"
#import "WBOTRecordDetialContainerViewModel.h"
#import "WBOTViewModelServices.h"
#import "WBOTNavigationSwitchTitleView.h"
#import "WBOTCalendarRootTopView.h"
#import "WBOTCalendarRootViewModel.h"
#import "ZHUlimitedScrollView.h"
#import "UIView+WBOTAddtion.h"
#import "WBOTDataBaseManager.h"
#import "WBOTuserSetting.h"
#import "WBOTCommonInfo.h"
#import "WBOToverworkRecord.h"
#import "WBOTCalendarDayNumberTool.h"
#import "UIView+WBOTAddtion.h"
#import "WBOTCommonDefine.h"
#import "UIColor+WBOTAddition.h"
#import "WBOTStatisticCalculateTool.h"
#import "WBOTLogEventManager.h"
#import "WBOTNewUserGuideView.h"
#import "WBCustomAlertView.h"
#import "ReactiveCocoa.h"
#import "RACEXTScope.h"
#import "UIView+Toast.h"
```

Swift

```swift
import RxSwift
```

e.g. ,Lyft 75000 -> 25000

简约

```
var addressNumber: Int?

if let home = paul.residence {
    if let postalAddress = home.address {
        if let building = postalAddress.buildingNumber {
            if let convertedNumber = building.toInt() {
                addressNumber = convertedNumber
            }
        }
    }
}
```

Optional Chain

```
addressNumber = paul.residence?.address?.buildingNumber?.toInt()
```

简约

取出1-100中开完平方后是偶数的值

```
let squares = (1..<10).map { $0 * $0 }.filter { $0 % 2 == 0 }
// [4, 16, 36, 64]
```

简约

```
self.statusLabel.text = "正在同步好友..."
self.syncBuddys {
    dispatch_async(dispatch_get_main_queue()) {
        self.statusLabel.text = "正在同步群..."
        self.statusProgressView.progress = 0.5
        self.tableView.reloadData()

        self.syncGroups {
            dispatch_async(dispatch_get_main_queue()) {
                self.statusLabel.text = "点击进行同步"
                self.statusProgressView.progress = 0.0
                self.tableView.reloadData()
            }
        }
    }
}
```

简约

```
func updateStatus(text: String, progress: Float = 0.0, reload: Bool = true) {
    self.statusLabel.text = text
    self.statusProgressView.progress = progress
    if reload { self.tableView.reloadData() }
}

updateStatus("正在同步好友...", reload: false)
self.syncBuddys {
    dispatch_async(dispatch_get_main_queue()) {
        updateStatus("正在同步群...", progress: 0.5)
        self.syncGroups {
            dispatch_async(dispatch_get_main_queue()) {
                updateStatus("点击进行同步")
            }
        }
    }
}
```

安全

Bug　　let age = response.toInt()

NULL　　NSIntegerMax

NSNotFound　　INT_MAX

-1

Nil　　nil　　0

如果有多层判断呢?

安全

Optional

```
var optionalNum: Int ?
var myObject: MyClass = MyClass()
myObjct = nil  // compile-time error
```

?

!

安全

```
var neighbors = ["Alex", "Anna", "Madison", "Dave"]
let index: Int? = findIndexOfString("Madison", neighbors)

if index {
    println("Hello, \(neighbors[index])")
} else {
    println("Must've moved away")
}
// error: value of optional type 'Int?' not unwrapped
```

安全

在编译时就明确告诉你可能存在的bug，而不是在运行时

++i or i++　　for (int i = 0; i < 10 ; i++)

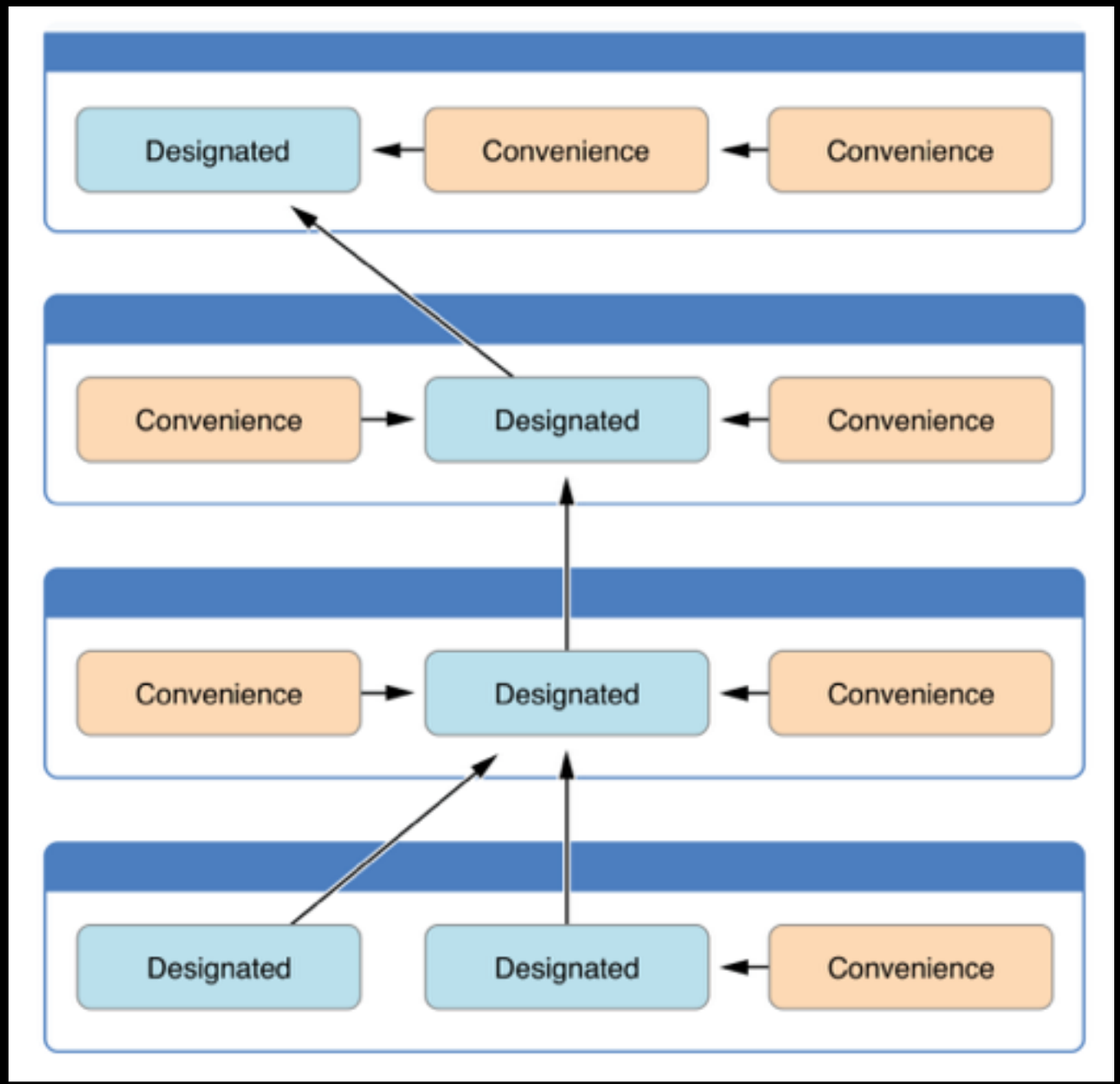
类型安全语言

非相同类型的数据不能运算

Enum必须遍历所有的类型

安全

初始化的流程


Designated
Convenience
Convenience
Convenience
Designated
Convenience
Convenience
Designated
Convenience
Designated
Designated
Convenience

安全

所有的变量使用前必须有明确的值　　OC呢?

```
struct Color {
    let red, green, blue: Double
    mutating func validateColor() { ... }
    init(grayScale: Double) {
        red = grayScale
        green = grayScale
        self.validateColor()
        blue = grayScale
    }
}
// error: 'self' used before being initialized
```

安全

```
class Car {
    var paintColor: Color
    init(color: Color) {
        paintColor = color
    }
}
```

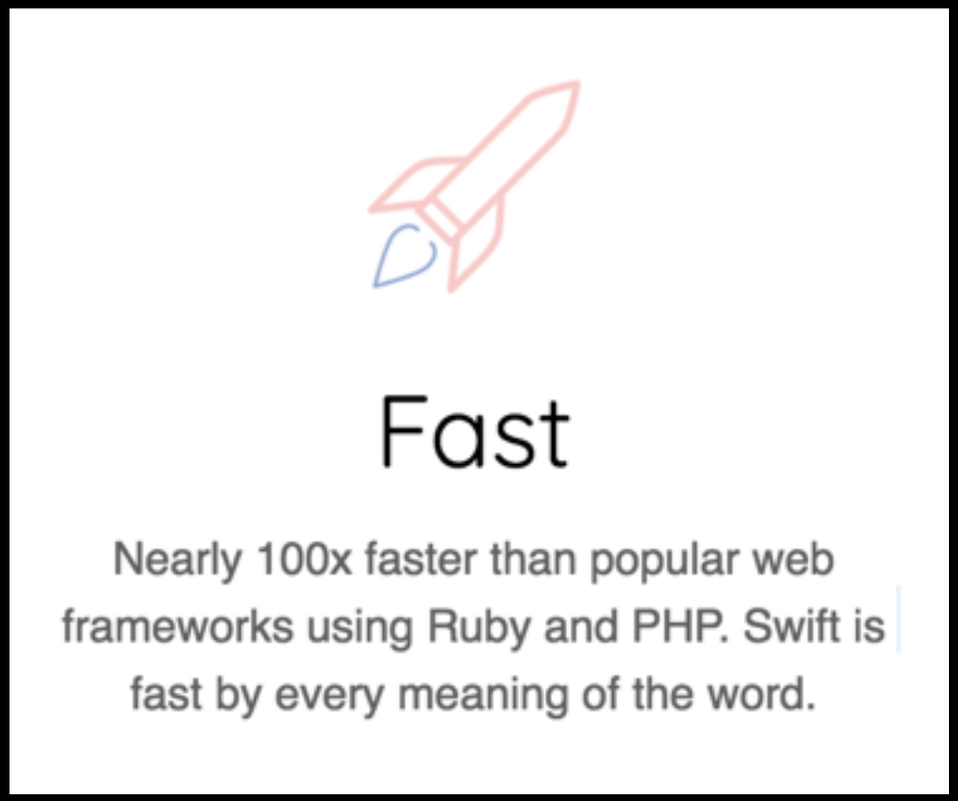
```
class RaceCar: Car {
    var hasTurbo: Bool

    init(color: Color, turbo: Bool) {
        super.init(color: color)
        hasTurbo = turbo
    }
}
```

安全

调用Super前必须所有数值属性都初始化

```
class Car {
    var paintColor: Color
    func fillGasTank() {...}
    init(color: Color) {
        paintColor = color
        fillGasTank()
    }
}

class RaceCar: Car {
    var hasTurbo: Bool
    override func fillGasTank() { ... }
    init(color: Color, turbo: Bool) {
        super.init(color: color)
        hasTurbo = turbo
    }
} // error: property 'hasTurbo' not initialized at super.init call
```

快速


Fast
Nearly 100x faster than popular web frameworks using Ruby and PHP. Swift is fast by every meaning of the word.

Vapor

快速

Method Dispatch

Statistic Dispatch

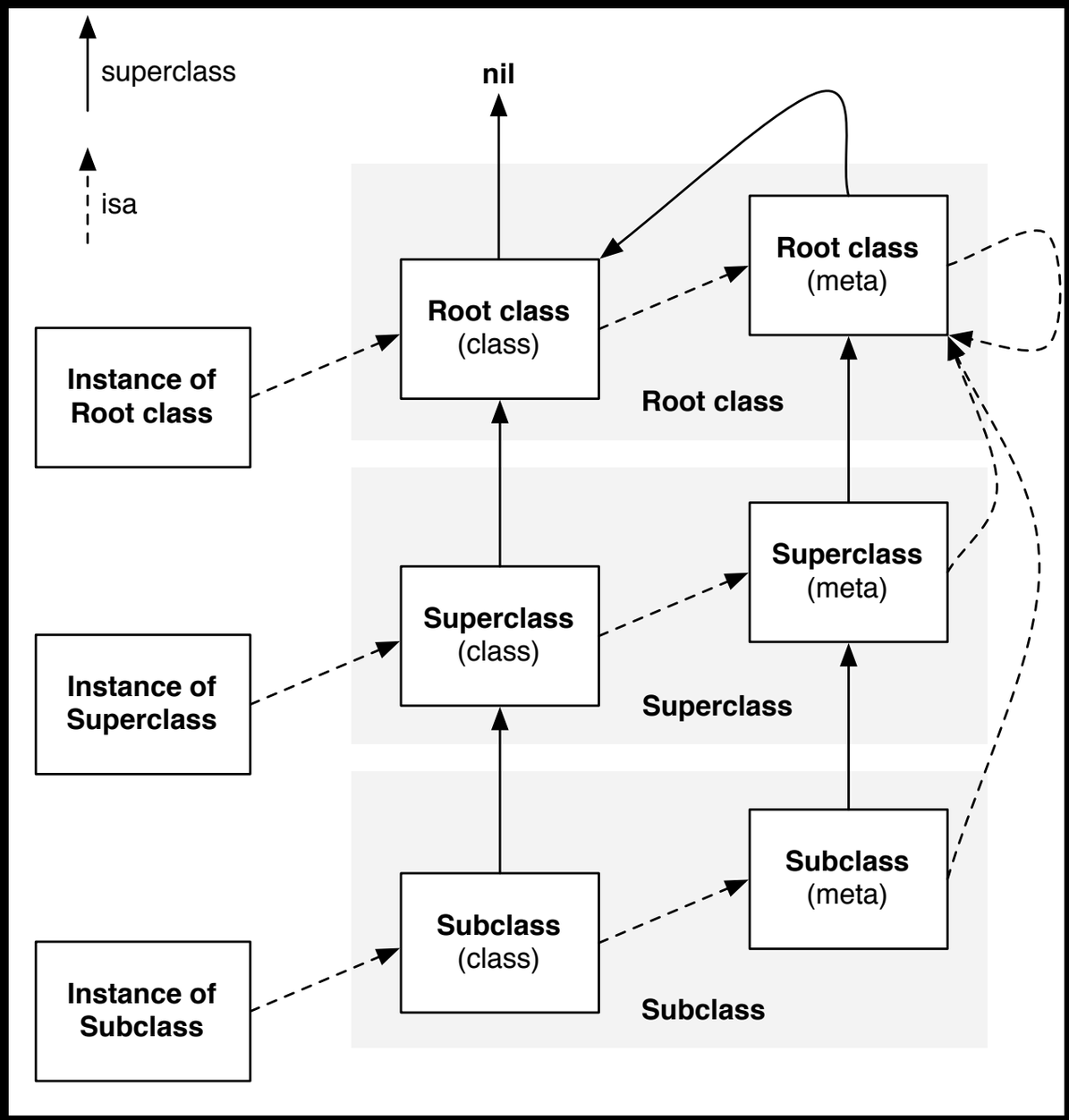
```
class Animal {
   func eat() {}
   func sleep() {}
}

class Dog: Animal {
verride func sleep() { }
}

class Rabbit: Animal {

   override func eat(){}
   override func sleep(){}

}
```

Dynamic Dispatch

```
var animal:Animal?
//执行很多业务逻辑
if somThingTrue {
    animal = Rabbit()
}else{
    animal = Dog()
}
animal?.eat()
```

快速

OC的Dynamic Dispatch



快速

Animal

Index0 eat 0x0001
Index1 sleep 0x0004

Dog

Index0 eat 0x0001 (copied)
Index1 sleep 0x0008 (overrideen)

Rabbit

Index0 eat 0x0002 (overrideen)
Index1 sleep 0x0003 (overrideen)

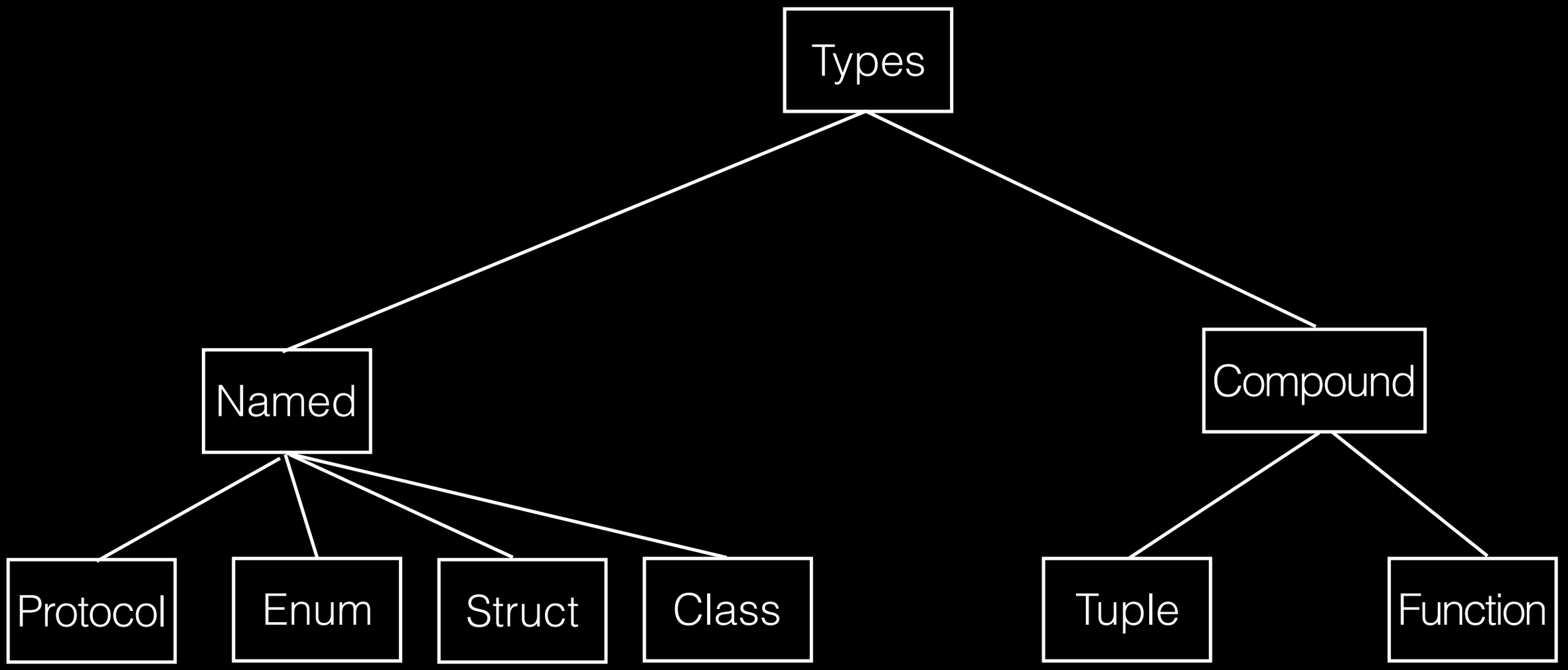
快速

全Module编译　使能Whole Module Optimization

多使用Access Control，让编译器理解Class的结构，

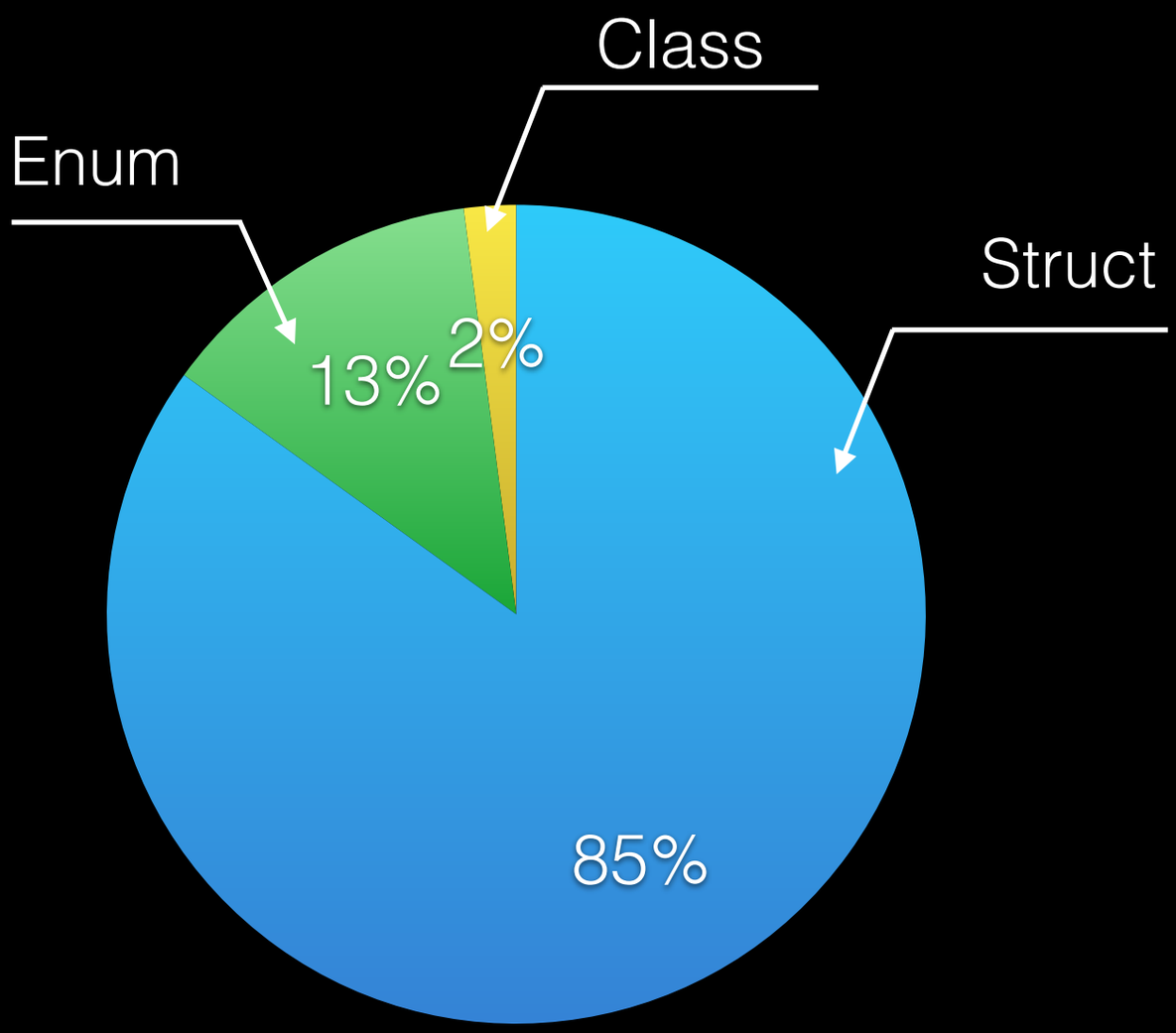
但是要避免破坏已有的调用

Swift Types


Types
Named
Protocol
Enum
Struct
Class
Compound
Tuple
Function

Swift Types

在Swift3标准库中，Struct， Enum，Class的使用率：



Reference Type

Engin *engin = [Engin alloc]init]; engin.speed = 0;

🚲 * bike = [🚲 alloc] init];　　bike.engin = engin; engin.speed = 10;
🚘 * car = [🚘 alloc] init];　　car.engin = engin; engin.speed = 100;
🚀 * rocket = [🚀 alloc] init];　roket.engin = 1000;
👴 * leyang = [👴 alloc] init];

leyang.rokect = rokect;
leyang.car = car;
leyang.bike = bike;

[leyang byBIke];

页面状态保存　　Cell的复用，享元模式

页面传值　　需要实现NSCopying协议并且Copy

Dictionary的key需要满足NSCopying协议

Swift Types

Value Type

```
int a = 20;
int b = a;
b = 30;
print(b);
print(a);
```

没有副作用
不会造成Race Condition

```
var numbers = [1, 2, 3, 4, 5]
scheduler.processNumbersAsynchronously(numbers)
for i in 0..<numbers.count { numbers[i] = numbers[i] * i }
scheduler.processNumbersAsynchronously(numbers)
```

Struct VS Class

Struct VS Class

共同点：

1. 定义属性并存值
2.定义方法
3.定义脚标
4. 定义初始化方法
5. 可以被扩展
6. 遵守协议

不同点：

1.值类型和引用类型
2.继承
3.强转
4.析构函数
5.ARC
6.Method Dispatch

Struct VS Class

选择Struct还是Class？

1. 封装简单的数据数值
2. 被赋值的时候被拷贝
3. 属性是也是值类型的，也可以被拷贝
4. 不需要从其它的结构体中继承属性
5. 全局只有一份，不需要被拷贝
6. Foundation

不能用Struct的时候
再考虑用Class

Protocol

```swift
protocol RandomNumberGenerator {
    func random() -> Double
}

protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

protocol Togglable {
    mutating func toggle()
}

class Dice {
    let sides: Int
    let generator: RandomNumberGenerator
    init(sides: Int, generator: RandomNumberGenerator) {
        self.sides = sides
        self.generator = generator
    }
    func roll() -> Int {
        return Int(generator.random() * Double(sides)) + 1
    }
}
```

1. 作为方法函数的参数和返回值
2. 作为常量变量和属性类型
3.作为数组，字典等储存值的类型

Protocol

实现多态不必非要继承了

```
protocol TextRepresentable {
    var textualDescription: String { get }
}

extension Dice: TextRepresentable {
    var textualDescription: String {
        return "A \(sides)-sided dice"
    }

}

struct LinearCongruentialGenerator: RandomNumberGenerator{
    func random() -> Double {
        return 12.3
    }

}

let d12 = Dice(sides: 12, generator: LinearCongruentialGenerator(
print(d12.textualDescription)
```

```
protocol Drawable { func draw() }

struct Point : Drawable {
   var x, y: Double
   func draw() { … }
}

struct Line : Drawable {
   var x1, y1, x2, y2: Double
   func draw() { … }
}

var drawables: [Drawable]
for d in drawables {
   d.draw()
}
```

Enum

```swift
enum ColorName: String {
    case black, silver, gray, white, maroon, red, purple, fuchsia
}
```

```swift
enum CSSColor {
    case named(ColorName)
    case rgb(UInt8, UInt8, UInt8)
}
```

```swift
enum TrainStatus {
   case OnTime, Delayed(Int)
   init() {
      self = OnTime
   }
   var description: String {
      switch self {
         case OnTime:
            return "on time"
         case Delayed(let minutes):
            return "delayed by \(minutes) minute(s)"
      }
   }
}
```

Function

多个返回值的方法，有默认值

```swift
func minMax(array: [Int]) -> (min: Int, max: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}



let bounds = minMax(array: [8, -6, 2, 109, 3, 71])
print("min is \(bounds.min) and max is \(bounds.max)")
// Prints "min is -6 and max is 109"


func someFunction(parameterWithoutDefault: Int, parameterWithDefault: Int = 12) {

}
someFunction(parameterWithoutDefault: 3, parameterWithDefault: 6)

someFunction(parameterWithoutDefault: 4)
```

Function

不定参函数

```swift
func arithmeticMean(_ numbers: Double...) -> Double {
    var total: Double = 0
    for number in numbers {
        total += number
    }
    return total / Double(numbers.count)
}
arithmeticMean(1, 2, 3, 4, 5)
// returns 3.0, which is the arithmetic mean of these five numbers
arithmeticMean(3, 8.25, 18.75)
```

Function

Function可以做参数，返回值

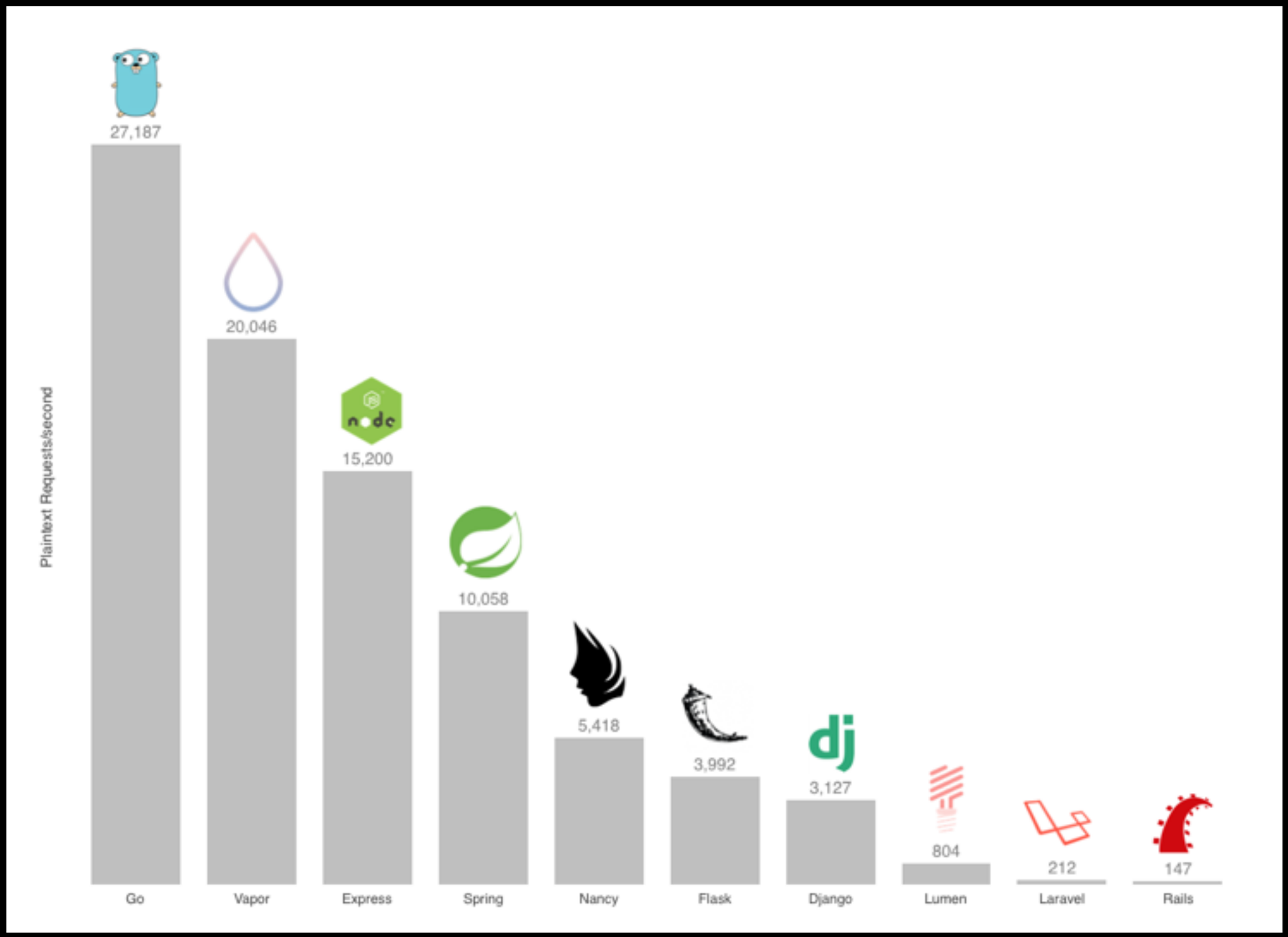
```swift
func addTwoInts(_ a: Int, _ b: Int) -> Int {
    return a + b
}
func multiplyTwoInts(_ a: Int, _ b: Int) -> Int {
    return a * b
}

var mathFunction: (Int, Int) -> Int = addTwoInts

print("Result: \(mathFunction(2, 3))")
// Prints "Result: 5"

mathFunction = multiplyTwoInts
print("Result: \(mathFunction(2, 3))")
// Prints "Result: 6"

func printMathResult(_ mathFunction: (Int, Int) -> Int, _ a: Int, _ b: Int) {
    print("Result: \(mathFunction(a, b))")
}

printMathResult(addTwoInts, 3, 5)
// Prints "Result: 8"
func stepForward(_ input: Int) -> Int {
    return input + 1
}
func stepBackward(_ input: Int) -> Int {
    return input - 1
}

func chooseStepFunction(backward: Bool) -> (Int) -> Int {
    return backward ? stepBackward : stepForward
}
```

Sever Side

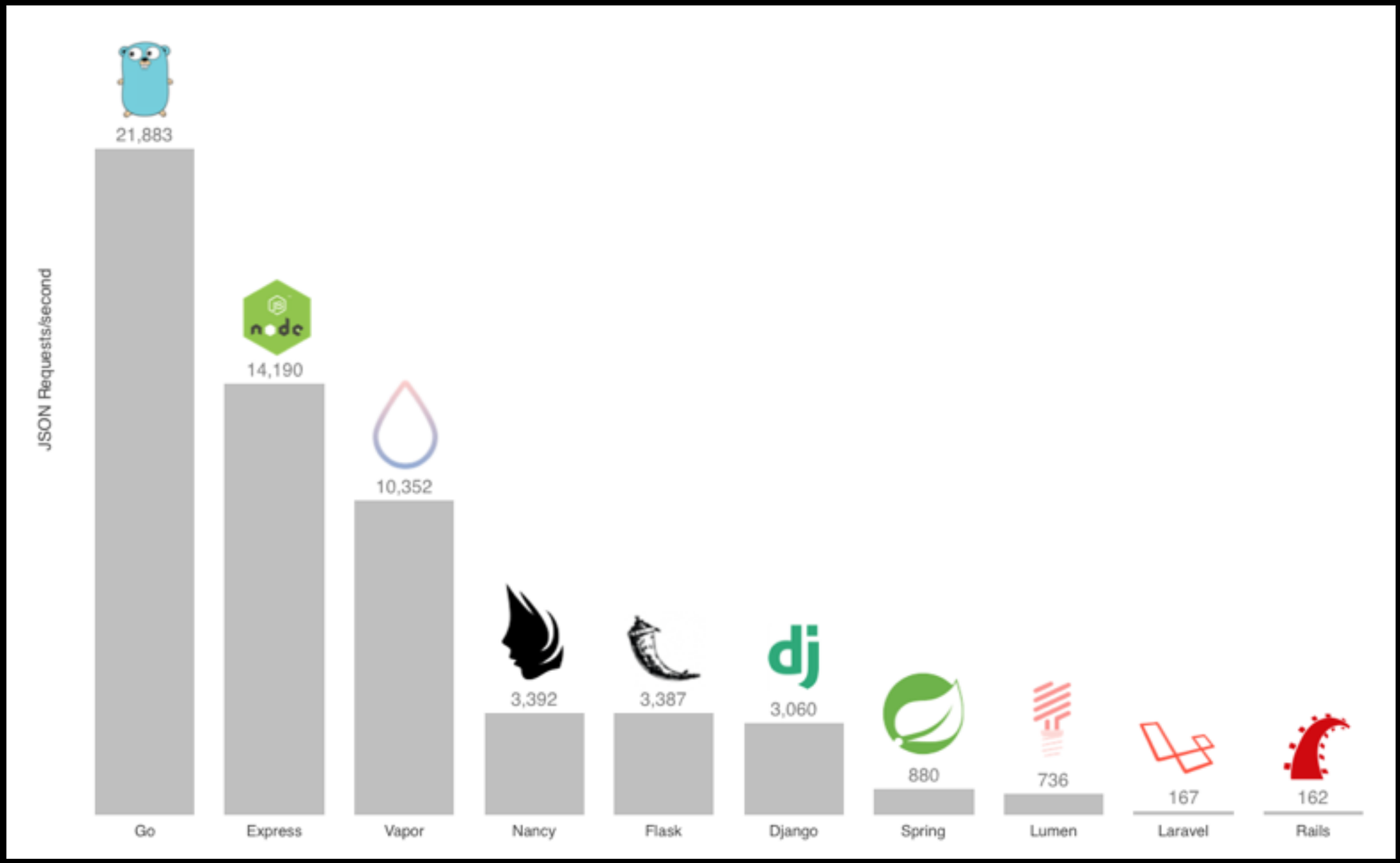
Frameworks

- Vapor (Swift)
- Ruby on Rails (Ruby)
- Laravel (PHP)
- Lumen (PHP)
- Express (JavaScript)
- Django (Python)
- Flask (Python)
- Spring (Java)
- Nancy (C#)
- Go (Pure Go, no framework)

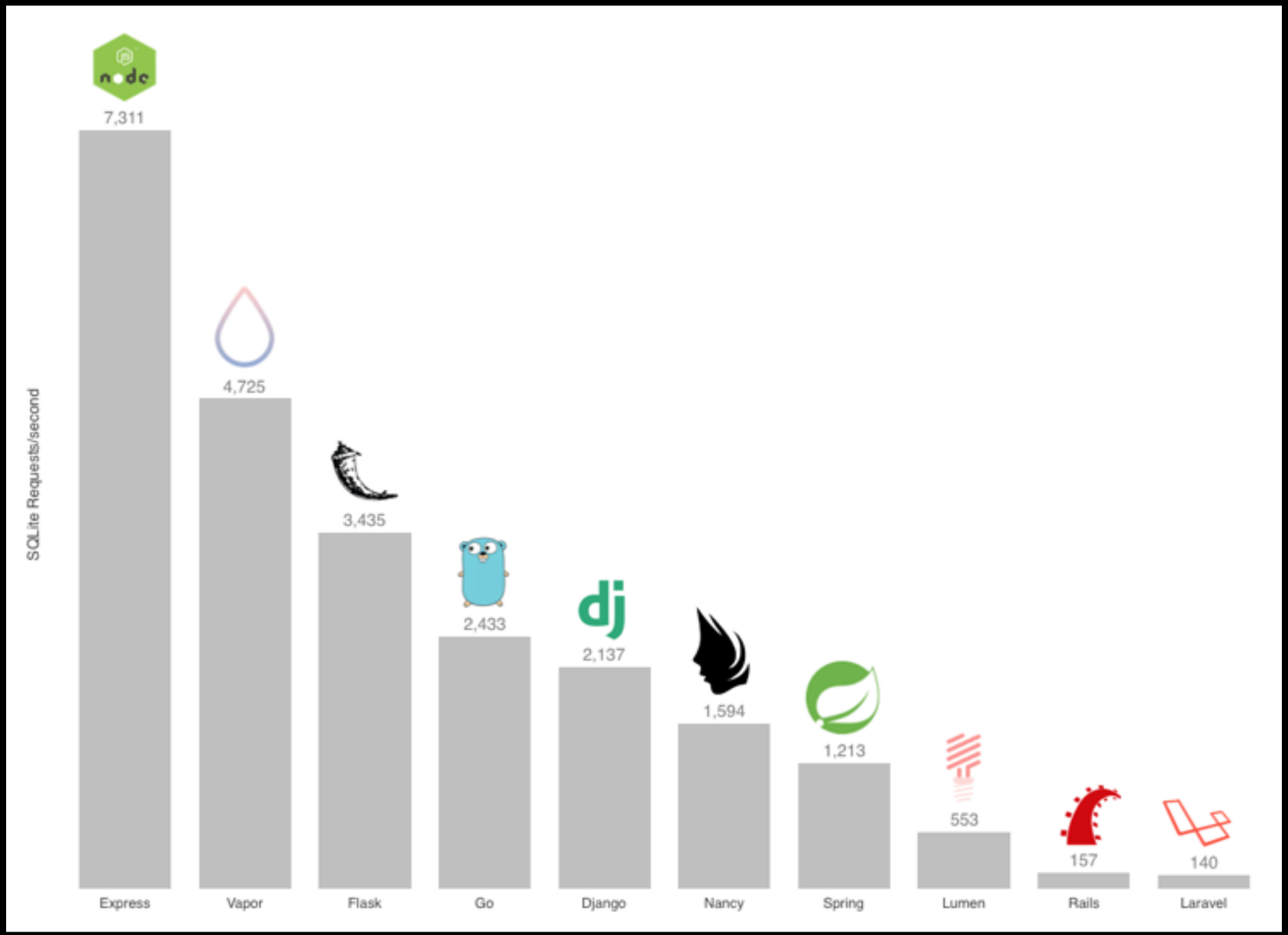
Sever Side


Plaintext Requests/second
27,187
20,046
15,200
10,058
5,418
3,992
3,127
804
212
147
Go
Vapor
Express
Spring
Nancy
Flask
Django
Lumen
Laravel
Rails

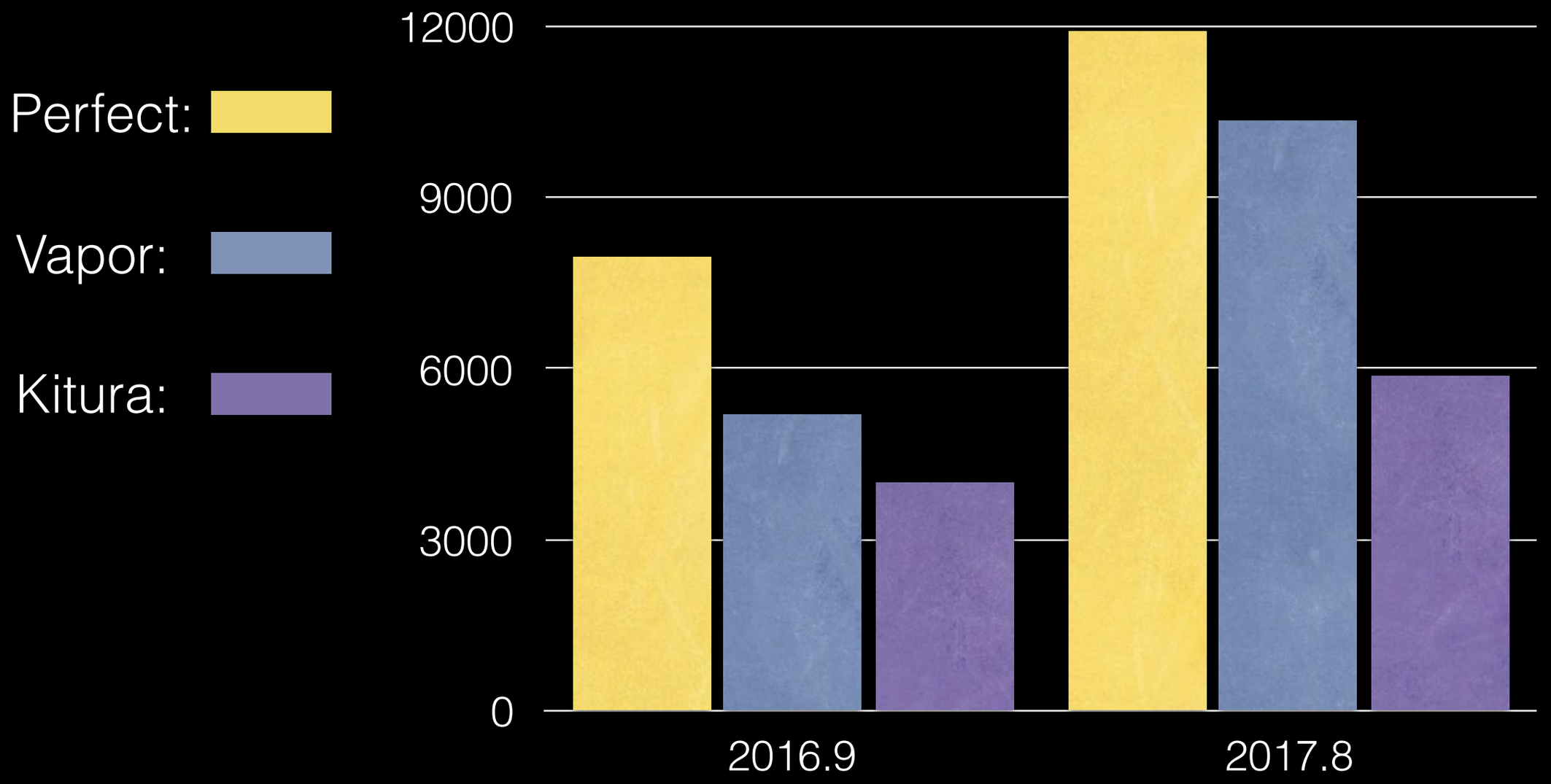
Sever Side


Requests/second
JSON
21,883
14,190
10,352
3,392
3,387
3,060
880
736
167
162
Go
Express
Vapor
Nancy
Flask
Django
Spring
Lumen
Laravel
Rails

Sever Side


SQLite Requests/second
7,311
4,725
3,435
2,433
2,137
1,594
1,213
553
157
140
Express
Vapor
Flask
Go
Django
Nancy
Spring
Lumen
Rails
Laravel

Sever Side



GitHub Stars

Sever Side

Prefect VS Vapor

Slack double

Contributors double

More active

More Concise

Sever Side

简洁

```
import PerfectLib

public func PerfectServerModuleInit() {
    Routing.Handler.registerGlobally()

    Routing.Routes["/"] = { _ in return HelloWorldHandler() }
}

class HelloWorldHandler: RequestHandler {
    func handleRequest(request: WebRequest, response: WebResponse) {
      response.appendBodyString("Hello, World!")
      response.requestCompletedCallback()
    }
}
```

Prefect

```
import Vapor

let app = Application()

app.get("/") { request in
    return "Hello, World!"
}

app.start()
```

Vapor

Sever Side

1. brew tap vapor/homebrew-tap
2. brew install vapor
3. vapor new HelloVapor —template=api
4. vapor build —release
5. vapor run serve

https://mikefighting.github.io/2017/07/26/VaporBasicUse/

Sever Side

Heroku

https://devcenter.heroku.com

注册账号
brew install heroku
heroku login
heroku create
git push heroku master

Q & A