netguru

Pedal to the Metal

Piotr Sochalewski
swift.map{$6}

Why?

netguru

What the heck?!

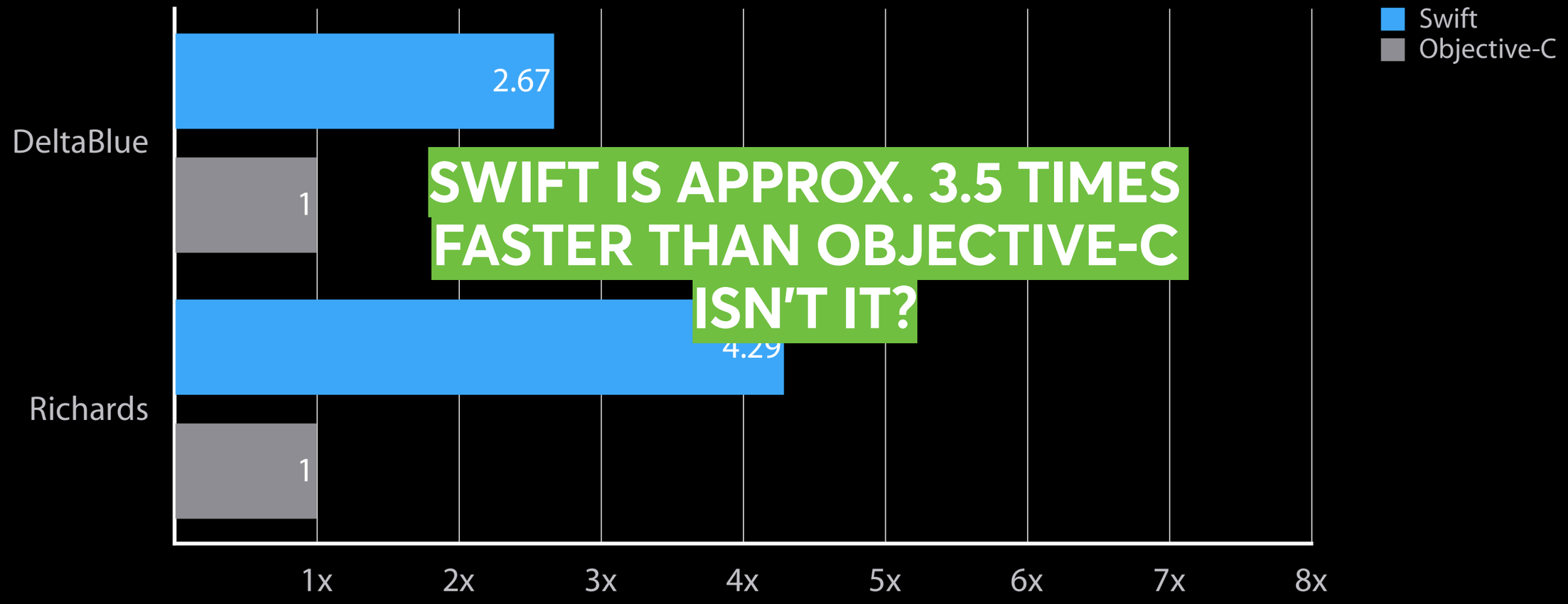
netguru

SWIFT IS A LOT FASTER THAN OBJECTIVE-C

OPTIMIZING SWIFT PERFORMANCE – WWDC 2015

Swift vs. Objective-C

Program speed (higher is better)



Source: Optimizing Swift Performance, Session 409, WWDC 15

iOS • 10 March 2016

Swift vs Obj-C Performance Comparison

Piotr Sochalewski

Is Swift faster than Objective-C? This question has been asked so many times and the answer is still unclear. So I took the recent Xcode 7.3 beta and ran some tests comparing Swift 2.2 and Objective-C. The results were surprising even for me.

I found many tests proving that Swift is faster than Objective-C and some saying old Obj-C is swifter than Swift. The truth is in the middle. Straight out in some cases Swift is much faster, but Obj-C still has some advantages.

Source: Droids On Roids blog

Agenda

1. Dispatch
2. Objects
3. Protocols
4. Tips
5. Just don't do it

Dispatch

netguru

STATIC
vs
DYNAMIC

Reducing dynamic dispatch

- `final`
- `private`
- Whole Module Optimization
- Swift-only codebase

Final

```
class Awesome {

  var kitties: [UIImage]?

  func showRandomKitty(in imageView: UIImageView) {
    imageView.image = kitties?.random
  }
}
```

Final

```
class Awesome {

  final var kitties: [UIImage]?

  final func showRandomKitty(in imageView: UIImageView) {
    imageView.image = kitties?.random
  }
}
```

Final

```
final class Awesome {

  var kitties: [UIImage]?

  func showRandomKitty(in imageView: UIImageView) {
    imageView.image = kitties?.random
  }
}
```

Private

```
class Awesome {

  fileprivate var kitties: [UIImage]?

  fileprivate func showRandomKitty(in imageView: UIImageView) {
    imageView.image = kitties?.random
  }
}
```

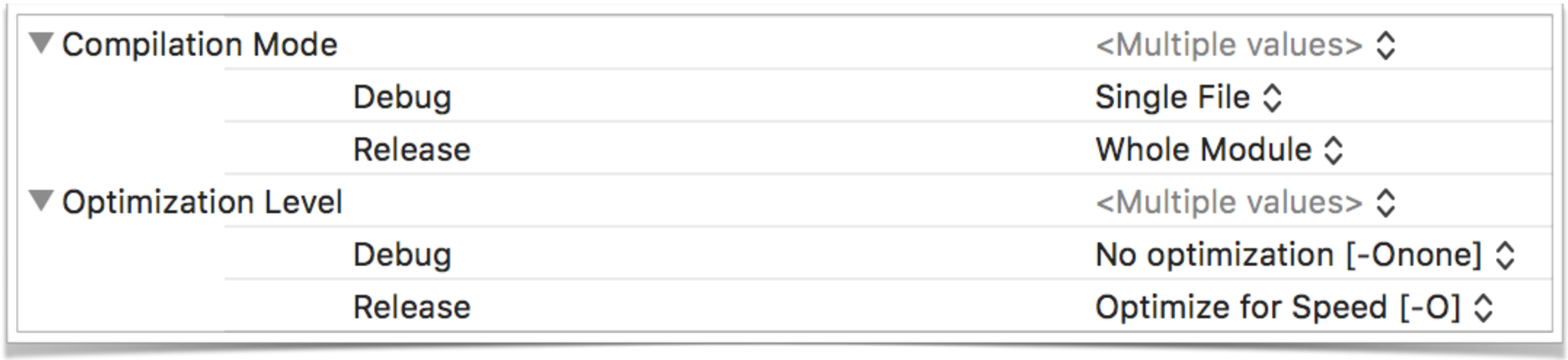
FINAL

NON-FINAL

```
final class EvenMoreAwesome: Awesome {

  override func showRandomKitty(in imageView: UIImageView) { … }
}
```

FINAL

Whole Module Optimization



Objects

CLASSES, STRUCTS

& ENUMS

Objects

netguru

struct, enum, tuple

VALUE TYPES

&

REFERENCE TYPES

class

Struct vs Class

```
struct Circle {
  let center: CGPoint
  let radius: Double

  func draw() {}
}
```

Struct vs Class

```
struct Circle {
  let center: CGPoint
  let radius: Double

  func draw() {}
}
```

```
let circles = (0..<n).map { _ in
  Circle(center: .zero, radius: 1.0)
}

circles.forEach { $0.draw() }
```

⏱ **0.14 s**

n = 15.000.000

netguru

Struct vs Class

```
final class Circle {
  let center: CGPoint
  let radius: Double

  init(center: CGPoint, radius: Double) {
    self.center = center
    self.radius = radius
  }

  func draw() {}
}
```

Struct vs Class

```
final class Circle {
  let center: CGPoint
  let radius: Double

  init(center: CGPoint, radius: Double) {
    self.center = center
    self.radius = radius
  }

  func draw() {}
}
```

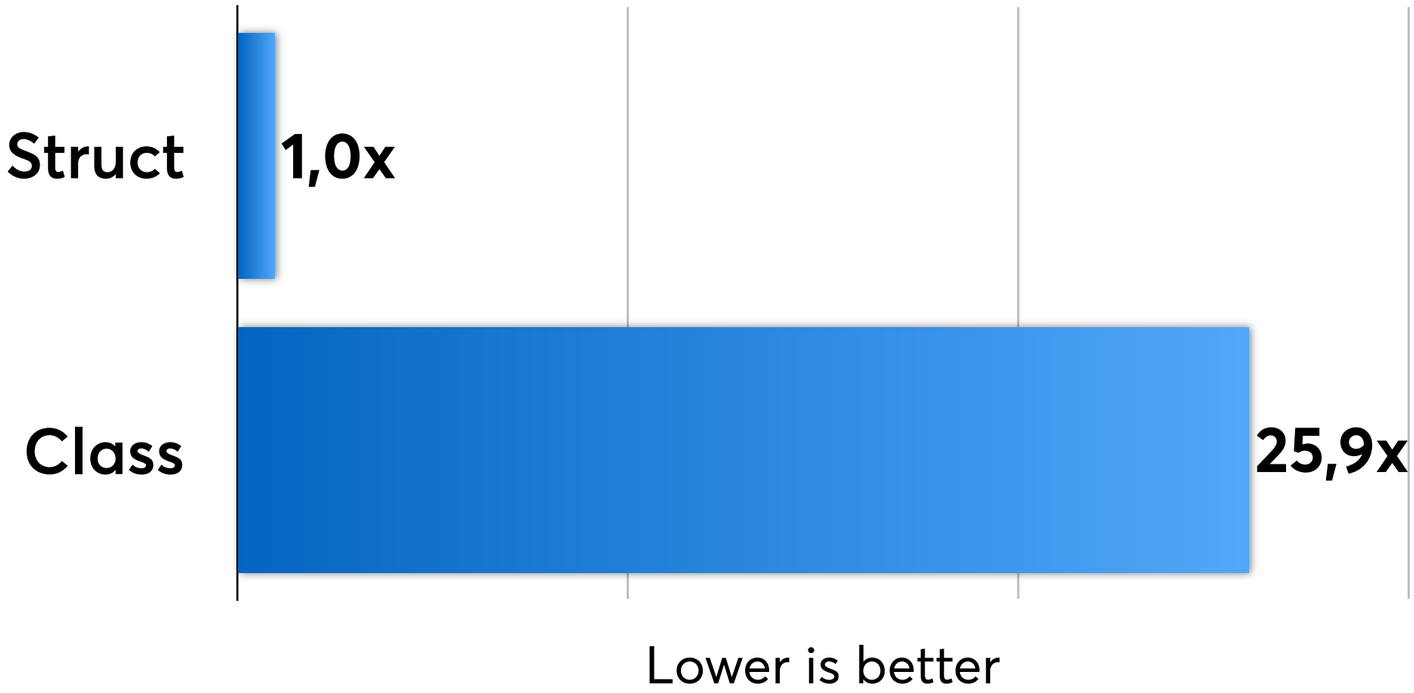
```
let circles = (0..<n).map { _ in
  Circle(center: .zero, radius: 1.0)
}

circles.forEach { $0.draw() }
```

3.68 s

n = 15.000.000

netguru

Struct vs Class



Struct and Class vs Protocol Conformance

```
protocol Drawable {
  func draw()
}
```

```
struct Circle: Drawable {
  let center: CGPoint
  let radius: Double

  func draw() {}
}
```

Struct and Class vs Protocol Conformance

```
protocol Drawable {
  func draw()
}

struct Circle: Drawable {
  let center: CGPoint
  let radius: Double

  func draw() {}
}
```

```
let circles: [Drawable] = (0..<n).map { _ in
  Circle(center: .zero, radius: 1.0)
}

circles.forEach { $0.draw() }
```

0.4 s

n = 15.000.000

netguru

Struct and Class vs Protocol Conformance

```
protocol Drawable {
  func draw()
}

final class Circle: Drawable {
  let center: CGPoint
  let radius: Double

  init(center: CGPoint, radius: Double) {
    self.center = center
    self.radius = radius
  }

  func draw() {}
}
```

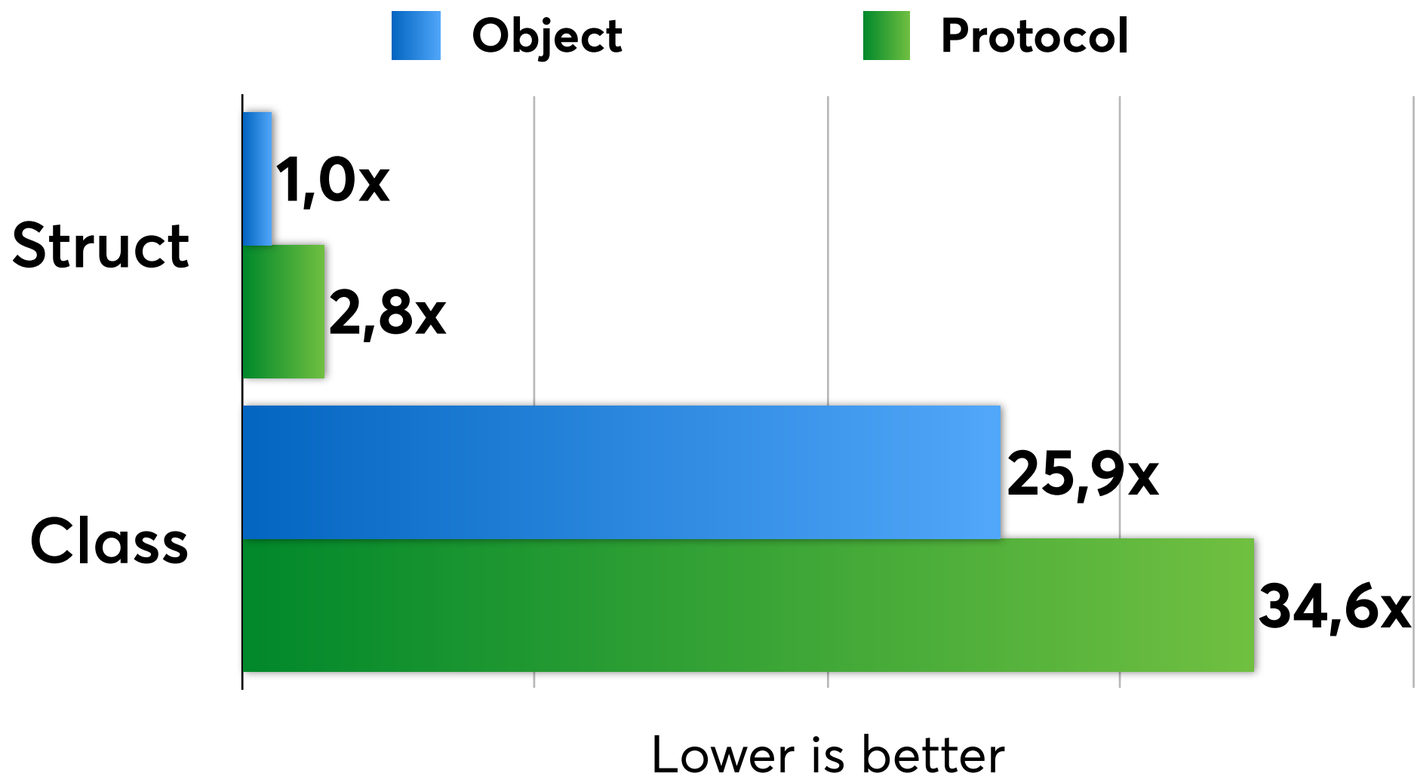
```
let circles: [Drawable] = (0..<n).map { _ in
  Circle(center: .zero, radius: 1.0)
}

circles.forEach { $0.draw() }
```

5.1 s

n = 15.000.000

netguru

Struct and Class vs Protocol Conformance



netguru

PROTOCOLS

Protocols

```
protocol SolarSystem {
  func loadPartOfSolarSystem()
}
```

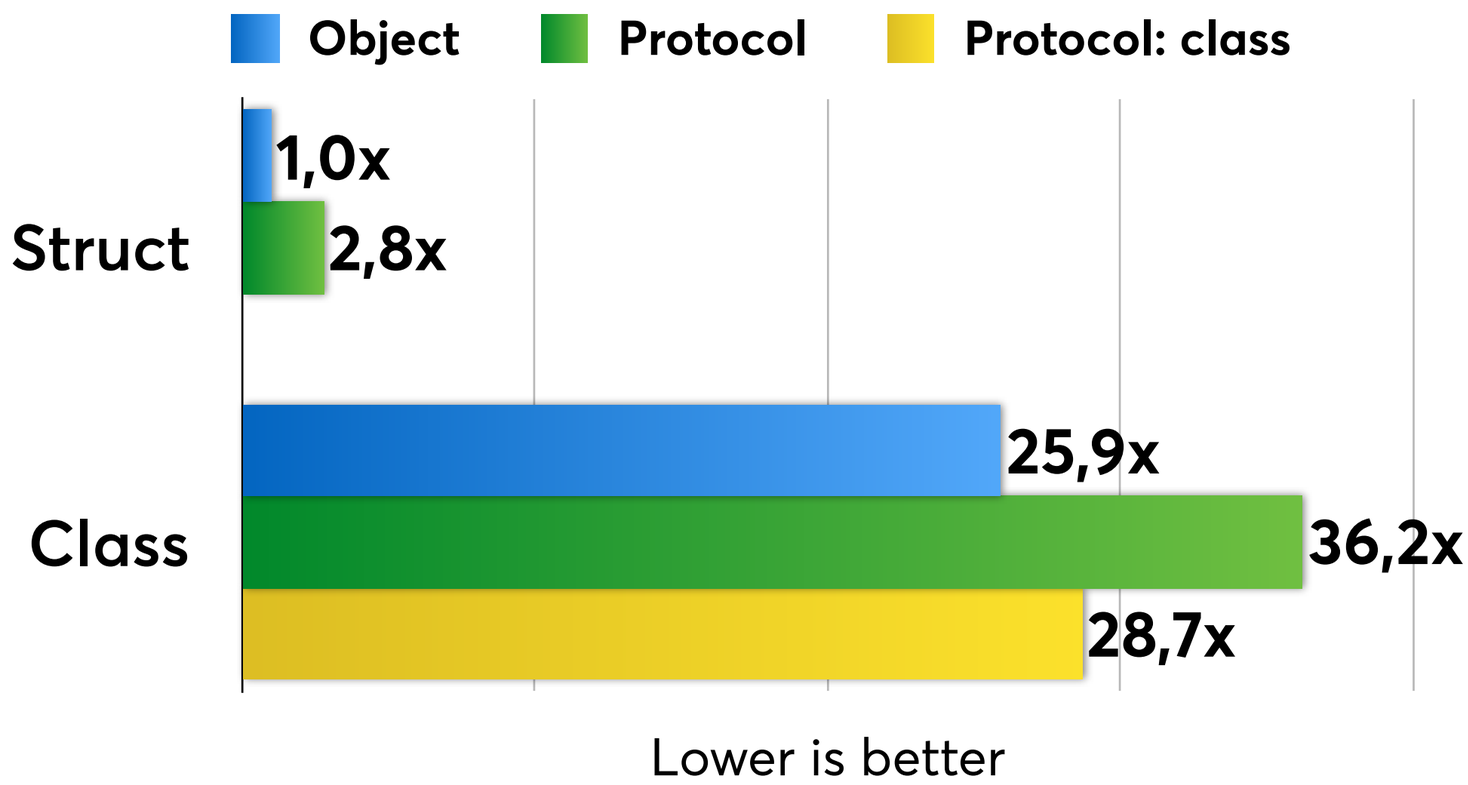
```
final class SunView: UIView, SolarSystem {

  func loadPartOfSolarSystem() { … }
}
```

Protocols

```
protocol SolarSystem: class {
  func loadPartOfSolarSystem()
}
```

```
final class SunView: UIView, SolarSystem {

  func loadPartOfSolarSystem() { … }
}
```

Protocols


Object
Protocol
Protocol: class
Struct
1,0x
2,8x
Class
25,9x
36,2x
28,7x
Lower is better

netguru

TIPS

Reuse DateFormatters

n = 10.000

```swift
let date = Date()
let dateFormatter = DateFormatter()
dateFormatter.dateStyle = .long
dateFormatter.timeStyle = .short
(0...n).forEach { _ in
  _ = dateFormatter.string(from: date)
}
```

⏱ **0.04 s**

```swift
let date = Date()
(0...n).forEach { _ in
  let dateFormatter = DateFormatter()
  dateFormatter.dateStyle = .long
  dateFormatter.timeStyle = .short
  _ = dateFormatter.string(from: date)
}
```

⏱ **0.3 s**

(7.5 times slower)

netguru

Dangerous optimization: Operators

```
&+, &-, &* // these operators don't do overflow checks
```

```
let x = UInt8.max // 255 🙋‍♀️
let y = x &+ 1 // 0 🤦‍♀️
```

ARC

```
func doSomethingFancy(with object: Object) {
  object.flipTheTable()
}
```

ARC

```
func doSomethingFancy(with object: Object) {
  __swift_retain(object)
  object.flipTheTable()
  __swift_release(object)
}
```

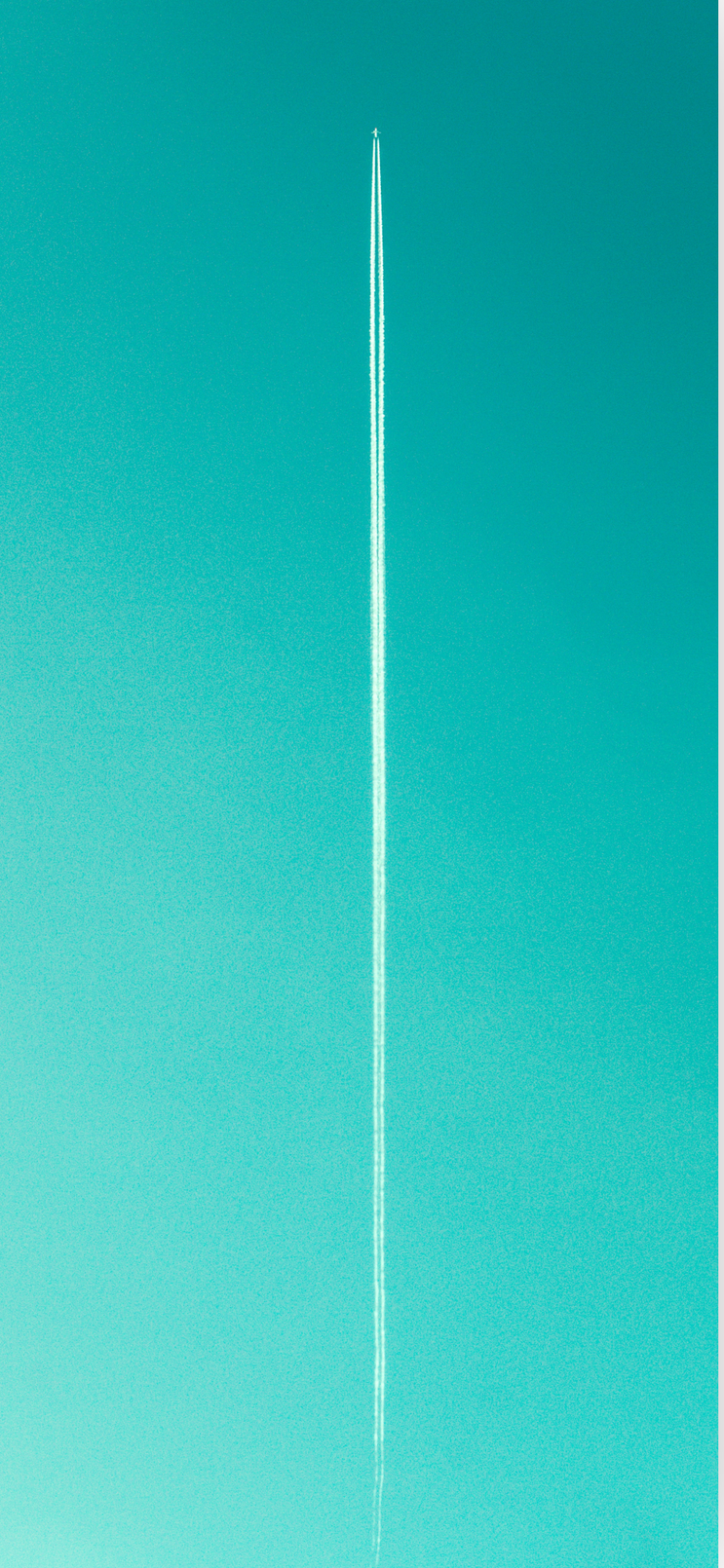
STOP ARC

STOP IT NOW

No.

Dangerous optimization: Unmanaged

```
public struct Unmanaged<Instance> where Instance : AnyObject { … }

// 👆 To avoid ARC calls
// Remember to have at least one strong reference
// to the object during execution
```



Optimization ways

- Swift-only codebase
- Static dispatch whenever possible
- Enable Whole Module Optimization
- Use structs instead of classes
- Avoid Protocol-Oriented Programming
  when you need extremely high performance
- Reuse Date Formatters

Piotr Sochalewski

Senior iOS Developer

piotr.sochalewski@netguru.co

netguru