Optimizing Swift Performance

Session 409

Nadav Rotem  Manager, Swift Performance Team
Michael Gottesman  Engineer, Swift Performance Team
Joe Grzywacz  Engineer, Performance Tools
Agenda

Swift 2.0 performance update
Understanding Swift performance
Using Instruments to analyze the performance of Swift programs
Swift is a Flexible, Safe Programming Language with ARC
Swift is a Flexible, Safe Programming Language with ARC

<table>
<thead>
<tr>
<th>Flexible</th>
<th>Safe</th>
<th>ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>function signature specializations</td>
<td>overflow checks removal</td>
<td>ARC optimizer</td>
</tr>
<tr>
<td>global variable optimizations</td>
<td>bounds checks elimination</td>
<td>copy forwarding</td>
</tr>
<tr>
<td>lock-less metadata caches</td>
<td>obj-c bridge optimizations</td>
<td>heap to stack</td>
</tr>
<tr>
<td>generics specializations</td>
<td>checked casting optimizations</td>
<td>code motion</td>
</tr>
<tr>
<td>class hierarchy analysis</td>
<td></td>
<td>alias analysis</td>
</tr>
<tr>
<td>closure optimizations</td>
<td></td>
<td>reference counting analysis</td>
</tr>
<tr>
<td>SSA optimizations</td>
<td></td>
<td>copy-on-write optimizations</td>
</tr>
<tr>
<td>call graph analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devirtualization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function inliner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Swift is a Flexible, Safe Programming Language with ARC

<table>
<thead>
<tr>
<th>Flexible</th>
<th>Safe</th>
<th>ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>function signature specializations</td>
<td>overflow checks removal</td>
<td>ARC optimizer</td>
</tr>
<tr>
<td>global variable optimizations</td>
<td>bounds checks elimination</td>
<td>copy forwarding</td>
</tr>
<tr>
<td>lock-less metadata caches</td>
<td>obj-c bridge optimizations</td>
<td>heap to stack</td>
</tr>
<tr>
<td>generics specializations</td>
<td>checked casting optimizations</td>
<td>code motion</td>
</tr>
<tr>
<td>class hierarchy analysis</td>
<td></td>
<td>alias analysis</td>
</tr>
<tr>
<td>closure optimizations</td>
<td></td>
<td>reference counting analysis</td>
</tr>
<tr>
<td>SSA optimizations</td>
<td></td>
<td>copy-on-write optimizations</td>
</tr>
<tr>
<td>call graph analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devirtualization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function inliner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds
Swift can lift checks out of loops
O(n) checks become O(1)

```swift
for i in 0..<n {
    A[i] ^= 13
}
```
Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds.
Swift can lift checks out of loops.
O(n) checks become O(1)

```swift
for i in 0..<n {
    precondition (i < length)
    A[i] ^= 13
}
```
Array Bounds Checks Optimizations

Swift ensures that array access happen in bounds

Swift can lift checks out of loops

O(n) checks become O(1)

```
precondition (n ≤ length)
for i in 0..<n {
    A[i] ^= 13
}
```
Swift is a Flexible, Safe Programming Language with ARC

<table>
<thead>
<tr>
<th>Flexible</th>
<th>Safe</th>
<th>ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>function signature specializations</td>
<td>overflow checks removal</td>
<td>ARC optimizer</td>
</tr>
<tr>
<td>global variable optimizations</td>
<td>bounds checks elimination</td>
<td>copy forwarding</td>
</tr>
<tr>
<td>lock-less metadata caches</td>
<td>obj-c bridge optimizations</td>
<td>heap to stack</td>
</tr>
<tr>
<td>generics specializations</td>
<td>checked casting optimizations</td>
<td>code motion</td>
</tr>
<tr>
<td>class hierarchy analysis</td>
<td></td>
<td>alias analysis</td>
</tr>
<tr>
<td>closure optimizations</td>
<td></td>
<td>reference counting analysis</td>
</tr>
<tr>
<td>SSA optimizations</td>
<td></td>
<td>copy-on-write optimizations</td>
</tr>
<tr>
<td>call graph analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>devirtualization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function inliner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Improvements Since 1.0

Optimized programs (higher is better)

<table>
<thead>
<tr>
<th>Program</th>
<th>1x</th>
<th>2x</th>
<th>3x</th>
<th>4x</th>
<th>5x</th>
<th>6x</th>
<th>7x</th>
<th>8x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richards</td>
<td>1.78</td>
<td>2.53</td>
<td>4.65</td>
<td>5.34</td>
<td>7.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GenericStack</td>
<td>1.78</td>
<td>2.53</td>
<td>4.65</td>
<td>5.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$.function</td>
<td>4.65</td>
<td>5.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LinkedList</td>
<td>2.53</td>
<td>4.65</td>
<td>5.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSStringConvert</td>
<td>1.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Improvements Since 1.0

Unoptimized programs (higher is better)

- Richards: 5.80
- GenericStack: 7.80
- $.function: 4.70
- LinkedList: 9.10
- NSStringConvert: 2.10
Swift vs. Objective-C

Program speed (higher is better)

- DeltaBlue: 2.67 Swift, 1 Objective-C
- Richards: 4.29 Swift, 1 Objective-C
Swift Compilation

Xcode compiles files independently, in parallel
Re-compile only files that need to be updated
Optimizer is limited to scope of one file
Whole Module Optimizations

Compilation is not limited to the scope of one file
Analyzing the whole module allows better optimizations
Whole Module Optimization greatly improved in Swift 2.0
• Better optimizations
• Parallel code generation
Performance Improvements Due to WMO

Swift 2 vs Swift 2 + WMO (higher is better)

<table>
<thead>
<tr>
<th></th>
<th>2x</th>
<th>3x</th>
<th>4x</th>
<th>5x</th>
<th>6x</th>
<th>7x</th>
<th>8x</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
<td>4.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBody</td>
<td></td>
<td></td>
<td>3.60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# New Optimization Level Configurations

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyApp</th>
<th>MyApp</th>
<th>iOS Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable Safety Checks</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Install Objective-C Compatibility Header</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Objective-C Bridging Header</td>
<td>MyApp-Swift.h</td>
<td>MyApp-Swift.h</td>
<td></td>
</tr>
<tr>
<td>Objective-C Generated Interface Header Name</td>
<td>&lt;Multiple values&gt;</td>
<td>&lt;Multiple values&gt;</td>
<td>-Swift.h</td>
</tr>
<tr>
<td>Optimization Level - Debug</td>
<td>None [-Onone]</td>
<td>None [-Onone]</td>
<td>Fast [-O]</td>
</tr>
<tr>
<td></td>
<td>Fast [-O]</td>
<td>Fast [-O]</td>
<td></td>
</tr>
<tr>
<td>Optimization Level - Release</td>
<td>Fast, Whole Module Optimization</td>
<td>Fast, Whole Module Optimization</td>
<td>-Wholemodule</td>
</tr>
</tbody>
</table>

## Swift Compiler - Code Generation

- **Setting:**
  - Resolved
  - MyApp
  - MyApp
- **Disable Safety Checks:**
  - No
- **Install Objective-C Compatibility Header:**
  - Yes
- **Objective-C Bridging Header:**
  - MyApp-Swift.h
- **Objective-C Generated Interface Header Name:**
  - <Multiple values>
- **Optimization Level - Debug:**
  - None [-Onone]
  - Fast [-O]
- **Optimization Level - Release:**
  - Fast, Whole Module Optimization

## Swift Compiler - Custom Flags

- **Setting:**
  - Resolved
  - MyApp
  - MyApp
- **Other Swift Flags:**
New Optimization Level Configurations
Writing High Performance Swift Code

Michael Gottesman  Engineer, Swift Performance Team
Overview

Reference Counting
Generics
Dynamic Dispatch
Overview

Reference Counting
Generics
Dynamic Dispatch
How Reference Counting Works
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
How Reference Counting Works

class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
class C { ... }
func foo(c: C?) { ... }

var x: C? = C()
var y: C? = x
foo(y)

y = nil
x = nil
Classes That Do Not Contain References

class Point {
    var x, y: Float
}

class Point {
    var x, y: Float
}

/* Diagram */

/* Diagram */

/* Diagram */

/* Diagram */

/* Diagram */
class Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    ...
}
class Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    increment
    ...
}
class Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    increment
    ...
    decrement
}
struct Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    increment
    ...
    decrement
}
struct Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    increment
    ...
    decrement
}
Structs That Do Not Contain References

```swift
struct Point {
    var x, y: Float
}

var array: [Point] = ...
for p in array {
    ...
}
```

All reference counting operations eliminated
Structs Containing a Reference
A Struct requires reference counting if its properties require reference counting.
A Struct requires reference counting if its properties require reference counting.
A Struct requires reference counting if its properties require reference counting.
A Struct requires reference counting if its properties require reference counting
A Struct requires reference counting if its properties require reference counting.
A Struct requires reference counting if its properties require reference counting.
Structs Containing Many References
Structs Containing Many References

- Struct User
  - String
  - String
  - String
  - Array
  - Dictionary

- Class 1
- Class 1
- Class 1
- Class 1
- Class 1
Structs Containing Many References

- Struct User
  - String
  - String
  - String
  - Array
  - Dictionary

- Class: 1

- Class
  - String: 1

- Class
  - String: 1

- Class
  - Array: 1

- Class
  - Dictionary: 1

- Struct User
Structs Containing Many References

- Struct User
  - String
  - String
  - Array
  - Dictionary

- Class
  - 2

- Class
  - 2

- Class
  - 2

- Class
  - 2

- Class
  - 2

- Struct User
  - String
  - String
  - Array
  - Dictionary
Use a Wrapper Class

Reference

Wrapper Class

Struct User

String

String

String

Array

Dictionary

Class

1

Class

1

Class

1

Class

1

Class

1
Use a Wrapper Class
Use a Wrapper Class

Building Better Apps with Value Types in Swift
Overview

Reference Counting
Generics
Dynamic Dispatch
Overview

Reference Counting
Generics
Dynamic Dispatch
How Generics Work

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```
How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}

func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```
How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```swift
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```
How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```swift
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```
How Generics Work

```swift
func min<T : Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

```swift
func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
```
func foo() {
    let x: Int = ...
    let y: Int = ...
    let r = min(x, y)
    ...
}

Generic Specialization
func foo() {
    let x: Int = ...
    let y: Int = ...
    let r = min(x, y)
    ...
}

Generic Specialization
func foo() {
    let x: Int = ...
    let y: Int = ...
    let r = min(x, y)
    ...
}

func min<T : Comparable>(x: T, y: T, FTable: FunctionTable) -> T {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
func foo() {
    let x: Int = ...
    let y: Int = ...
    let r = min(x, y)
    ...
}

func min<Int>(x: Int, y: Int, FTable: FunctionTable) -> Int {
    let xCopy = FTable.copy(x)
    let yCopy = FTable.copy(y)
    let m = FTable.lessThan(yCopy, xCopy) ? y : x
    FTable.release(x)
    FTable.release(y)
    return m
}
func foo() {
    let x: Int = ...  
    let y: Int = ...  
    let r = min(x, y)
    ...
}  

func min<Int>(x: Int, y: Int) -> Int {
    return y < x ? y : x
}
Generic Specialization

```swift
func foo() {
    let x: Int = ...
    let y: Int = ...
    let r = min<Int>(x, y)
    ...
}

func min<Int>(x: Int, y: Int) -> Int {
    return y < x ? y : x
}
```
Specialization is Limited by Visibility

Module A

File1.swift

```swift
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```
Specialization is Limited by Visibility

Module A

File1.swift

```swift
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Passing Int to min<T>
Specialization is Limited by Visibility

Module A

File1.swift

```swift
func compute(...) -> Int {
    ...  
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

- Passing Int to `min<T>`
- Definition not visible in File1
Specialization is Limited by Visibility

File1.swift

```swift
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Module A

- Passing `Int` to `min<T>`
- Must call `min<T>`
- Definition not visible in File1
Whole Module Optimization

Module A

File1.swift

```swift
func compute(...) -> Int {
    ...  
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```
Whole Module Optimization

Module A

File1.swift

```swift
func compute(...) -> Int {
    ...
    return min(x, y)
}
```

File2.swift

```swift
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}
```

Definition is visible in File1.
func min<T: Comparable>(x: T, y: T) -> T {
    return y < x ? y : x
}

func compute(...) -> Int {
    ...
    return min(x, y)
}

Can call min<Int>
Definition is visible in File1
Overview

- Reference Counting
- Generics
- Dynamic Dispatch
Overview

Reference Counting
Generics
Dynamic Dispatch
Dynamic Dispatch
Dynamic Dispatch

public class Pet

  func noise()
  var name
  func noiseImpl()

class Dog

override func noise()
func makeNoise(p: Pet) {
    print("My name is \(p.name)\n    \n    p.noise()
}
func makeNoise(p: Pet) {
    print("My name is \(p.name)\n    \n    p.noise()
func makeNoise(p: Pet) {
    print("My name is \(p.name)"")
    p.noise()
}

func makeNoise(p: Pet) {
    print("My name is \(p.name)"")
    p.noise()
}
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}

func makeNoise(p: Pet) {
}

func noiseImpl()

class Dog
    override func noise()
func makeNoise(p: Pet) {
    print("My name is \(p.name)")
    p.noise()
}

let nameGetter = Pet.nameGetter(p)
print("My name is \(nameGetter(p))")

func makeNoise(p: Pet) {
    let noiseMethod = Pet.noiseMethod(p)
    noiseMethod(p)
}
func makeNoise(p: Pet) {
    print("My name is \(p.name)"
    p.noise()
}

let noiseMethod = Pet.noiseMethod(p)
noiseMethod(p)

let nameGetter = Pet.nameGetter(p)
print("My name is \(nameGetter(p))")

Can only emit direct calls if it is known that the method is not overridden
Communicate API Constraints
Communicate API Constraints

Inheritance
Communicate API Constraints

Inheritance
Access Control
Inheritance

```swift
public class Pet {
    func noise() {
        var name
        func noiseImpl()
    }
}

class Dog {
    override func noise() {
    }
}

func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    print("My name is \(p.name)")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```
func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}

Inheritance

public class Pet

    var name
    func noise()
    func noiseImpl()

class Dog

    override func noise()


Inheritance

```
func makeNoise(p: Pet) {
    let nameGetter = Pet.getNameGetter(p)
    print("My name is \(nameGetter(p))")
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
```
func makeNoise(p: Pet) {
    print("My name is \(p.name)\)
    let noiseMethod = Pet.getNoiseMethod(p)
    noiseMethod(p)
}
Access Control

public class Pet

  func noise()
  final var name
  func noiseImpl()

override func noiseImpl()

class Dog

override func noise()
Access Control

Module A

Pet.swift
- public class Pet
  - func noise()
  - final var name
  - func noiseImpl()

Dog.swift
- class Dog
  - override func noise()
Module A

Pet.swift

public class Pet

func noise()
final var name
func noiseImpl()

Dog.swift

class Dog

override func noise()

Module B

Cat.swift

class Cat

override func noise()
Access Control

Module A
- Pet.swift
  - public class Pet
    - func noise()
    - final var name
    - func noiseImpl()

Dog.swift
- class Dog
  - override func noise()

Module B
- Cat.swift
  - class Cat
    - override func noise()
Access Control

Module A

Pet.swift

public class Pet

func noise()  
final var name

private func noiseImpl()  

Dog.swift

class Dog

override func noise()  
override func noiseImpl()  

Module B

Cat.swift

class Cat

override func noise()  
override func noiseImpl()
Access Control

Module A
- Pet.swift
  - public class Pet
    - func noise()
    - final var name
    - private func noiseImpl()

Dog.swift
- class Dog
  - override func noise()

Module B
- Cat.swift
  - class Cat
    - override func noise()
Whole Module Optimization

Module A

Pet.swift

public class Pet

func noise()
final var name
private func noiseImpl()

Dog.swift

class Dog
override func noise()
Whole Module Optimization

```swift
func bark(d: Dog) {
    d.noise()
}
```
Whole Module Optimization

```
class Dog
  override func noise()

func noiseImpl()

public class Pet
  func noise()
  final var name
  private func noiseImpl()

func bark(d: Dog) {
  d.noise()
}

func bark(d: Dog) {
  let noiseMethod = Dog.getNoiseMethod()
  noiseMethod(d)
}
```
func bark(d: Dog) {
    d.noise()
}

func bark(d: Dog) {
    let noiseMethod = Dog.getNoiseMethod()
    noiseMethod(d)
}
module A

Pet.swift

public class Pet

final var name

private func noiseImpl()

Dog.swift

class Dog

override func noise()
```swift
func bark(d: Dog) {
    d.noise()
}
```

```
class Dog {
    override func noise() {
        final var name
        private func noiseImpl() {
        }
    }
}
```
Swift vs. Objective-C

Program speed (higher is better)

- **DeltaBlue**
  - Swift: 2.67
  - Objective-C: 1

- **Richards**
  - Swift: 4.29
  - Objective-C: 1
Communicate your API Intent

Use the final keyword and access control
- Help the compiler understand your class hierarchy
- Be aware of breaking existing clients

Enable Whole Module Optimization
Demo

Joe Grzywacz
Engineer, Performance Tools
Swift is a flexible, safe programming language with ARC
Write your APIs and code with performance in mind
Profile your application with Instruments
More Information

Swift Language Documentation
http://developer.apple.com/swift

Apple Developer Forums
http://developer.apple.com/forums

Stefan Lesser
Developer Tools Evangelist
slesser@apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling in Depth</td>
<td>Mission</td>
<td>Thursday 3:30PM</td>
</tr>
<tr>
<td>Building Better Apps with Value Types in Swift</td>
<td>Mission</td>
<td>Friday 2:30PM</td>
</tr>
</tbody>
</table>