Swift Values

Francisco Díaz

@fco_diaz

Objective-C -> Swift

NSArray / NSMutableArray
-> Array

NSDictionary / NSMutableDictionary
-> Dictionary

NSString / NSMutableString
-> String

- Differences between Value / Reference Types
- Immutability in Swift
- Using Value Types

Value Type

Struct

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

Copying creates an independent instance with its own unique copy of its data

```
var a = Point(x: 1, y: 2)
var b = a // a: {1, 2}; b: {1, 2}
b.x = 3 // a: {1, 2}; b: {3, 2}
```

Reference Type

Class

```
class Person {
    var name: String

    init(name: String) {
        self.name = name
    }
}
```

Copying a reference, on the other hand, implicitly creates a shared instance

```
let pedro = Person(name: "Pedro")
var clon = pedro // pedro: {"Pedro"}; clon: {"Pedro"}
clon.name = "Pablo" // pedro: {"Pablo"}; clon: {"Pablo"}
```

Value Types are Immutable

What about variables? And *mutating* functions? Eh? Eh? 😠

```
struct Point {
    var x: Int, y: Int

    init(x: Int, y: Int) {
        self.x = x
        self.y = y
    }

    mutating func movePointBy(x: Int, y: Int) {
        self.x += x
        self.y += y
    }
}

var a = Point(x: 1, y: 2)
a.movePointBy(3, y: 3) // a: {4, 5}
a.x = 20 // a: {20, 5}
```

```swift
let a = Point(x: 1, y: 2)
a.movePointBy(3, y: 3) // Compilation error
a.x = 20 // Compilation error

// Immutable value of type 'Point' only has mutating members named 'movePointBy'
```

Using Value Types

The Value layer game

by Andy Matuschak

Object layer

Value layer

Prefer structs over classes

Constants by default

```swift
struct Point {
    let x: Int, y: Int
}
```

Use mutability carefully, where it makes sense

```
struct Meetup {
    let speakers: [String]
}

struct Meetup {
    var speakers: [String]
    mutating func addAwesomeSpeaker(speaker: String)
}

addAwesomeSpeaker("Francisco") ~== Meetup(speaker: speakers.append("Francisco"))
```

Every Value type should be Equatable

Values are inherently equatable

```
let a = "Hola "
let b = "Mundo"
a == b // false

"Hola Mundo" == a + b // true

1 == 2 - 1 // true
```

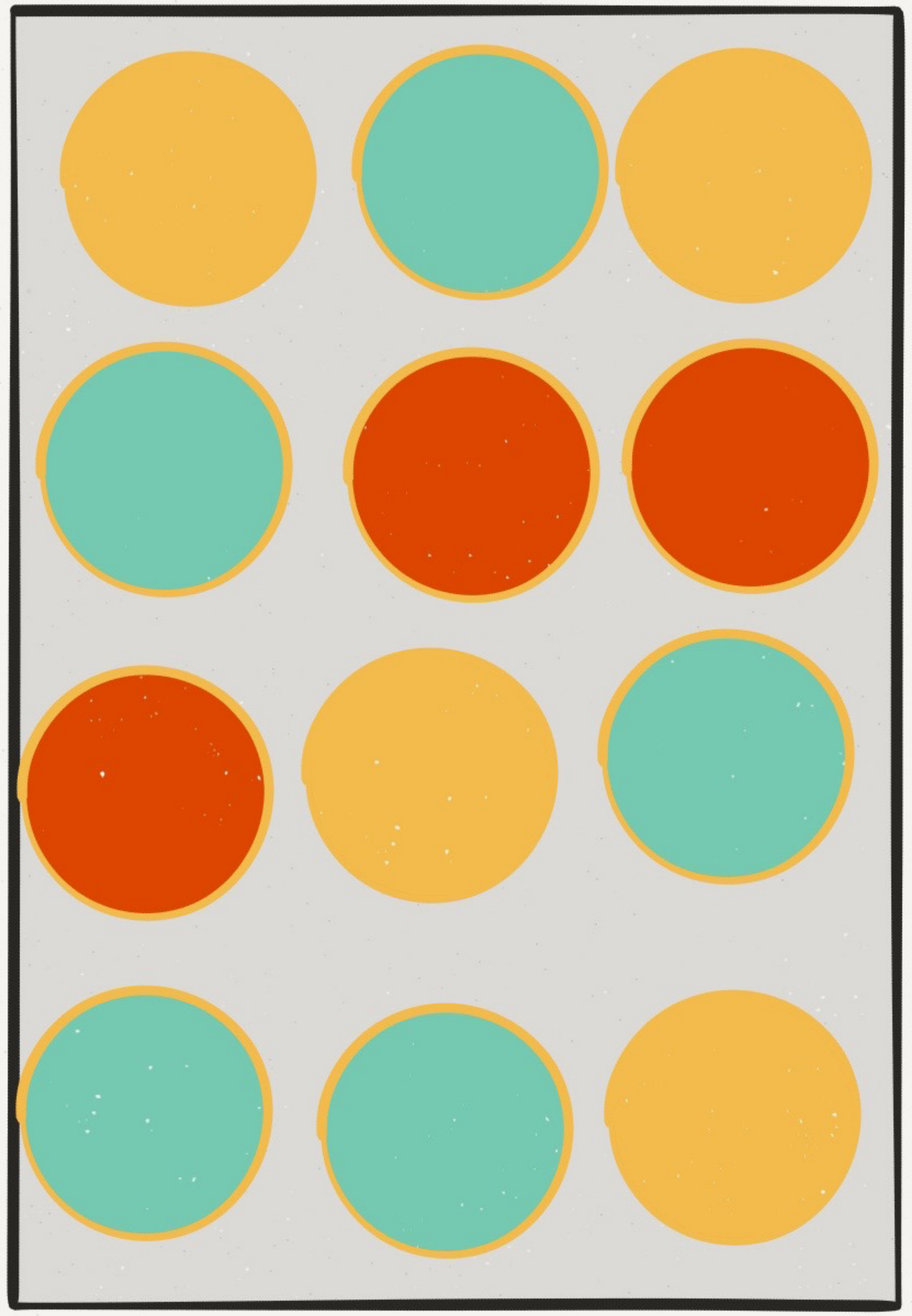
How?

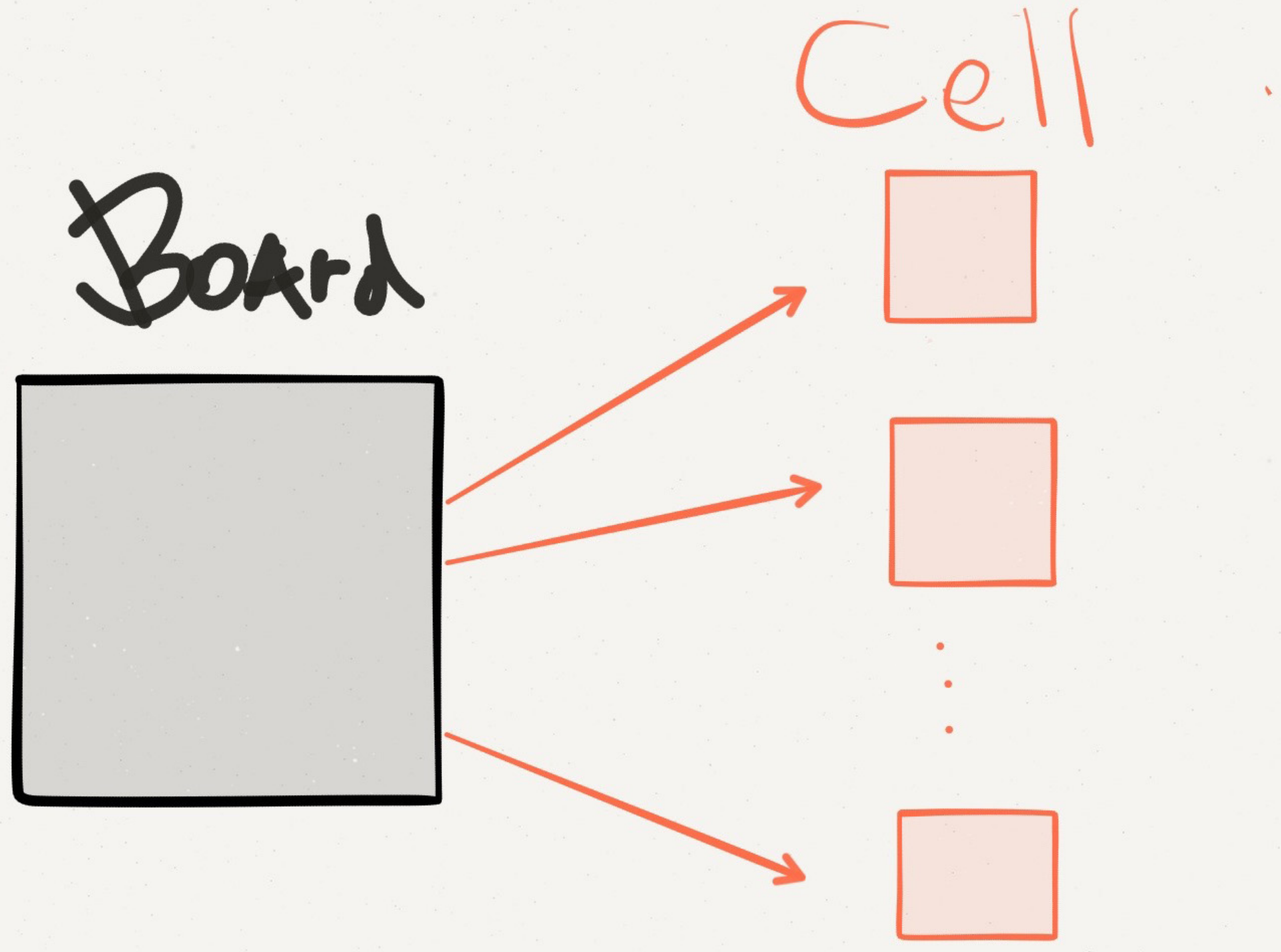
```
struct Point: Equatable {
    let x: Int, y: Int
}

func ==(lhs: Point, rhs: Point) -> Bool {
    return lhs.x == rhs.x && lhs.y == rhs.y
}
```

When to use Classes?

- `NetworkController1 == NetworkController2` ???
- `UIKit`

Example




Board
Cell

MVVM

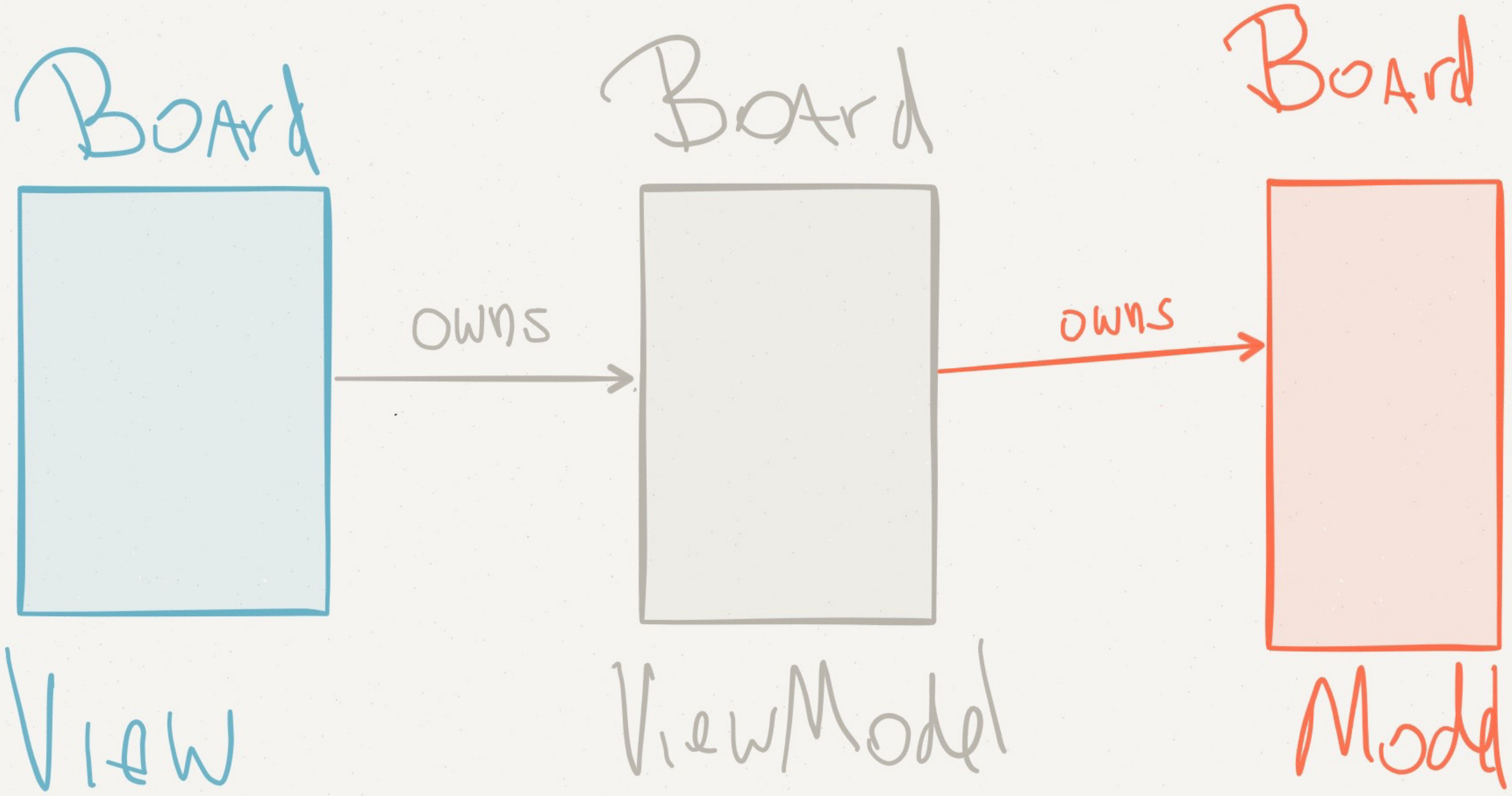
Board Model: Contains data representing a board

Board VM: Communicates between Model and View

- Converts model data to be displayed
- Takes user input and acts on model

Board View: Displays a board to the user

Game Scene: Puts it all together.


Board
Board
Board
owns
owns
View
ViewModel
Model

Choose immutability and see where it takes you.

Rich Hickey

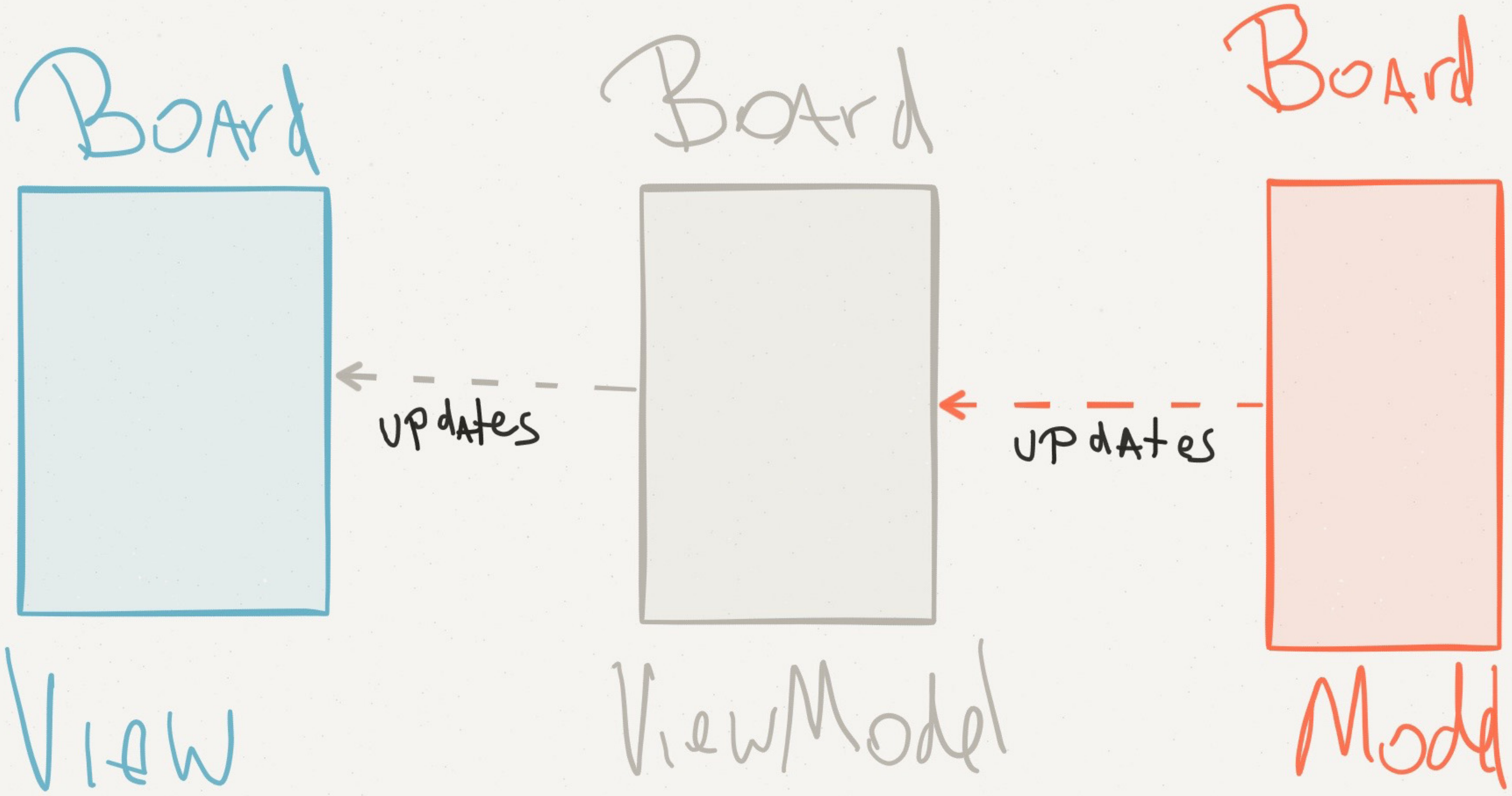
```swift
struct Board {
    let cells: [Cell]
}

struct Cell {
    let value: Int
}
```

```swift
struct BoardViewModel {
    let board: Board

    func cellViewModelAtIndex(index: Int) -> CellViewModel
}


struct CellViewModel {
    let cell: Cell
    let index: Int

    func attributes() -> (color: UIColor, texture: SKTexture, ...)
}
```

```swift
class BoardView: SKSpriteNode {
    var cellViews: [CellView] = []

    init(size: CGSize)

    func configure(boardViewModel: BoardViewModel)
}

class CellView: SKSpriteNode {

    init(size: CGSize, cellViewModel: CellViewModel)

    func configure(cellViewModel: CellViewModel)

    func scaleBy(scale: CGFloat)

    func animateTouch()

    ...
}
```

Every time something changes, create a new BoardViewModel and pass it to the BoardView

```
func configure(boardViewModel: BoardViewModel)
```


Board
View
updates
Board
ViewModel
updates
Board
Model

Only update what changed

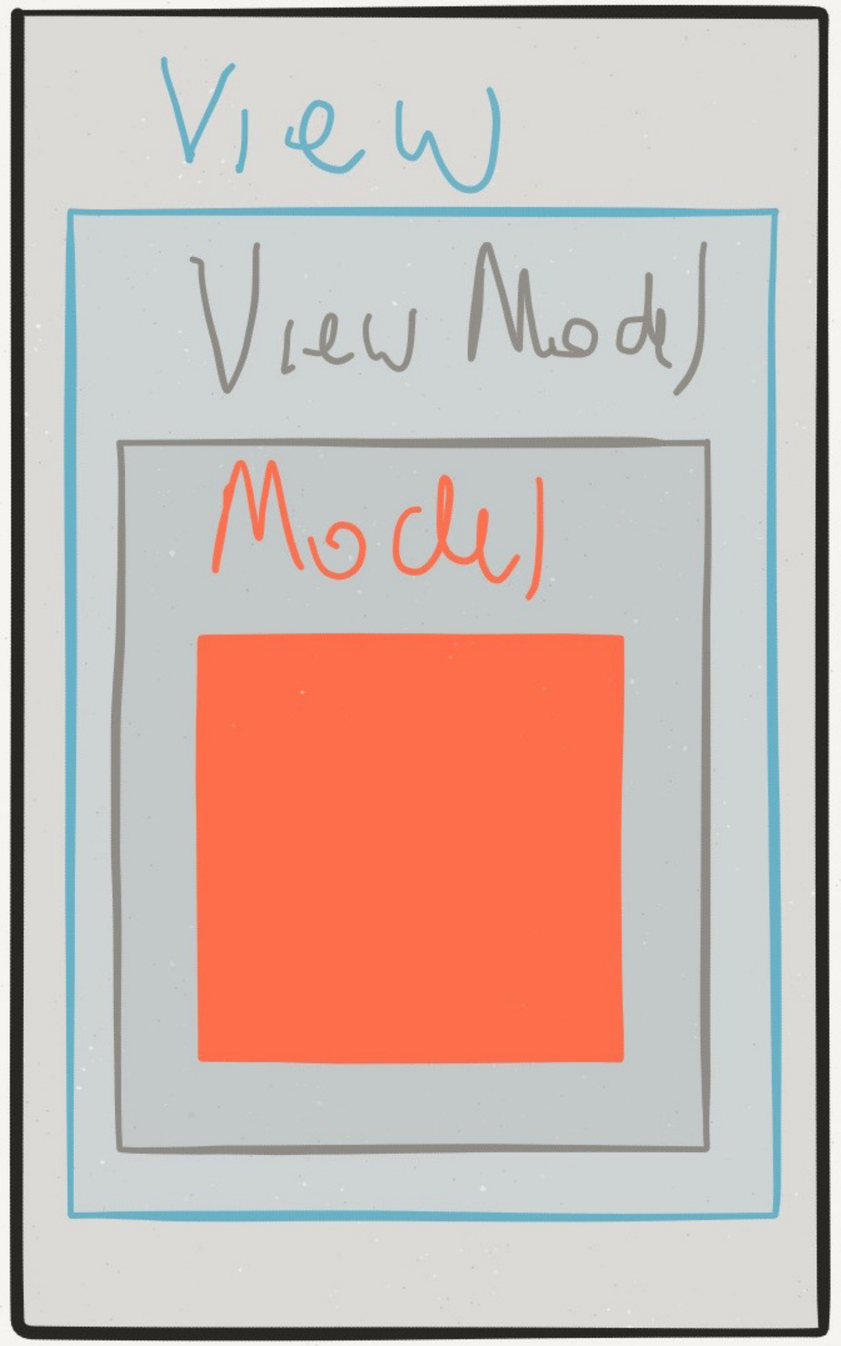
But how do I know what changed?

They're Values

...and you can compare values easily

```
func configure(cellViewModel: CellViewModel) {
    if oldCellViewModel != cellViewModel {
        // update
    }
}
```

Game Scene



- Differences between Value / Reference Types
- Immutability in Swift
- Using Value Types

Resources:

Swift Blog: Value and Reference Types

Should I use a Swift struct or a class?

WWDC 2015: Session 408

WWDC 2015: Session 414

The Value of Values by Rich Hickey

Enemy of the State by Justin Spahr-Summers

Functioning as a Functionalist by Andy Matuschak

Immutable Data and React by Lee Byron

Thanks!

Slides available at:

https://github.com/fdiaz/swift-values-talk