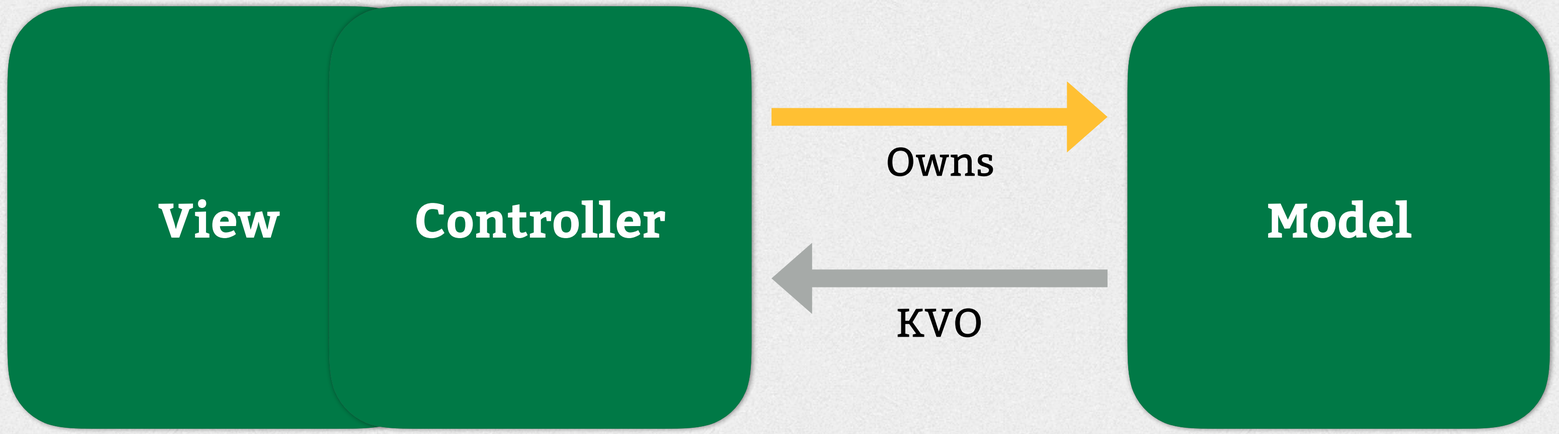
306: MVVM

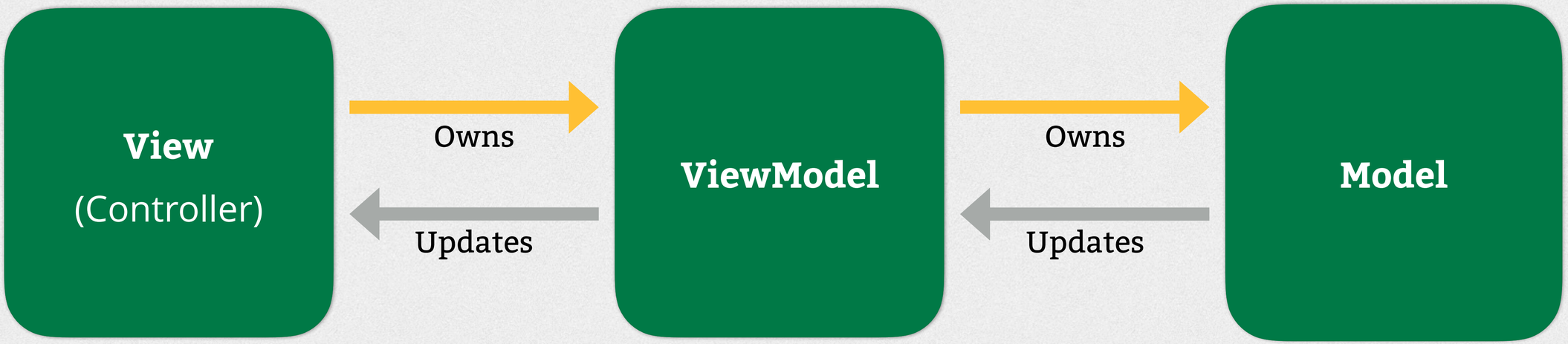
Model-View-ViewModel

MVC In Theory

MVC In Practice



Model-View-ViewModel



Responsibilities

- **View** - Presentation, user interaction
- **ViewModel** - Presentation logic
- **Model** - Business logic

What does it solve?

- MVC - *Massive* View Controller
- Testability
- Code organization
- Code reusability

Limitations / Cons

- Requires binding
- Potential for boilerplate code
- Overkill for simple views and logic
- Doesn’t cover every case

Demo 1

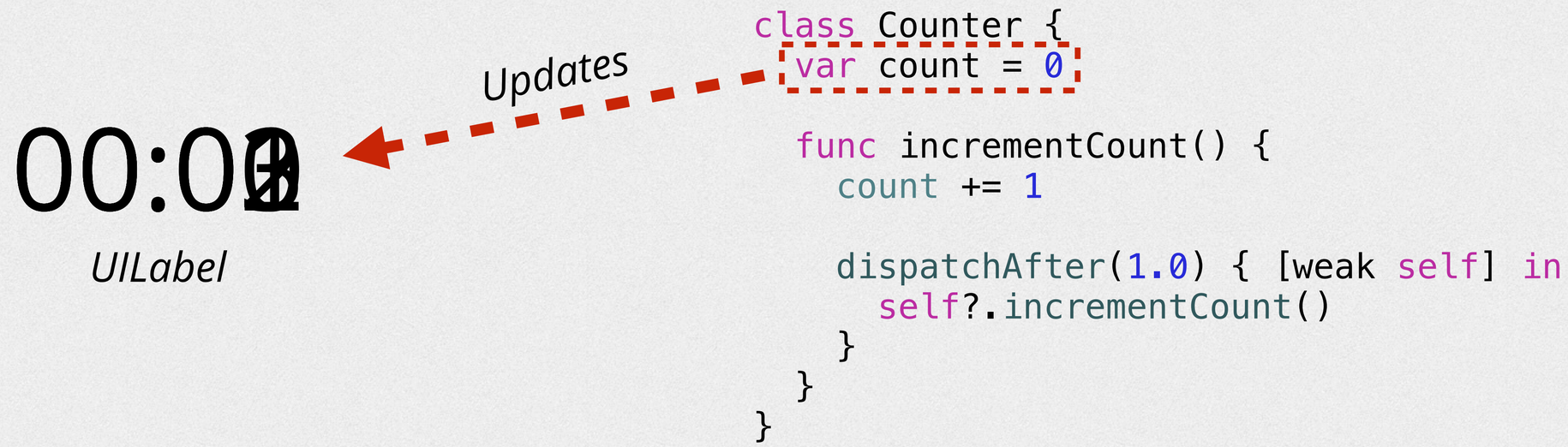
Username: ecerney
Password: swift

306: MVVM

Data Binding

What is data binding?

- Connection between UI and Business logic



Two-way Binding

- Changes to model update UI
- User input updates model

00:03


```
class Counter {
  var count = 0

  func incrementCount() {
    count += 1

    dispatchAfter(1.0) { [weak self] in
      self?.incrementCount()
    }
  }
}
```

How do we do this in Swift?

KVO

```
class ObjectToObserve: NSObject {
  dynamic var foo = 0
}

class MyObserver: NSObject {
  var objectToObserve = ObjectToObserve()

  override init() {
    super.init()
    objectToObserve.addObserver(self, forKeyPath: "foo", options: .New, context: nil)
  }

  override func observeValueForKeyPath(keyPath: String?, ofObject object: AnyObject?,
      change: [String : AnyObject]?, context: UnsafeMutablePointer<Void>) {
    if let newValue = change?[NSKeyValueChangeNewKey] {
      print("Value changed: \(newValue)")
    }
  }

  deinit {
    objectToObserve.removeObserver(self, forKeyPath: "foo", context: nil)
  }
}

let observer = MyObserver()
observer.objectToObserve.foo = 1 // Value changed: 1
```

How do we do this in Swift?

Delegation

```
protocol ObserveDelegate: class {
  func propertyChanged(newValue: Int)
}

class ObjectToObserve {
  private var foo = 0

  weak var delegate: ObserveDelegate?

  func changeFoo() {
    foo += 1
    delegate?.propertyChanged(foo)
  }
}
```

```
class MyObserver: ObserveDelegate {
  var objectToObserve = ObjectToObserve()

  init() {
    objectToObserve.delegate = self
  }

  func propertyChanged(newValue: Int) {
    print("Value changed: \(newValue)")
  }
}

let observer = MyObserver()
observer.objectToObserve.changeFoo() // Value changed: 1
```

How do we do this in Swift?

- Functional Reactive Programming (FRP)

函数式编程超出范围的这次谈话，我不觉得像创建它的一个例子。

How do we do this in Swift?

- Property Observers

```
class ObjectToObserve {
  var foo = 0 {
    didSet {
      print("Value changed: \(foo)")
    }
  }
}


let objectToObserver = ObjectToObserve()
objectToObserver.foo = 1 // Value changed: 1
```

Boxing

```
class Box<T> {
  var value: T {
    didSet {
      // Notify Listener(s)
    }
  }

  init(_ value: T) {
    self.value = value
  }
}

let boxedInt = Box(42)
boxedInt.value = 100
```

Adding Binding

```
class Box<T> {
  typealias Listener = T -> Void
  var listener: Listener?

  var value: T {
    didSet {
      listener?(value)
    }
  }

  init(_ value: T) {
    self.value = value
  }

  func bind(listener: Listener?) {
    self.listener = listener
    listener?(value)
  }
}

let boxedInt = Box(42)
boxedInt.bind {
  print("Value changed: \($0)")
}
// Value changed: 42
boxedInt.value = 100 // "Value changed: 100"
```

Demo 2

Lab

306: MVVM

Conclusion

What You Learned

- **Demo 1**: Refactoring MVC to MVVM
- **Demo 2**: Data Binding
- **Lab**: MVVM with Delegation

Benefits and Limitations

- **Benefits**
  - Breaks up code by responsibility
  - Simplifies Testing
  - Code reusability
- **Limitations**
  - Requires binding (or other workarounds)
  - Potential for boilerplate code
  - Sometimes feels like overkill
  - Doesn’t cover every case

Where To Go From Here?

- Ash Furrow's Blog: ashfurrow.com
- Reactive Cocoa (if you dare)
- Twitter: @ecerney