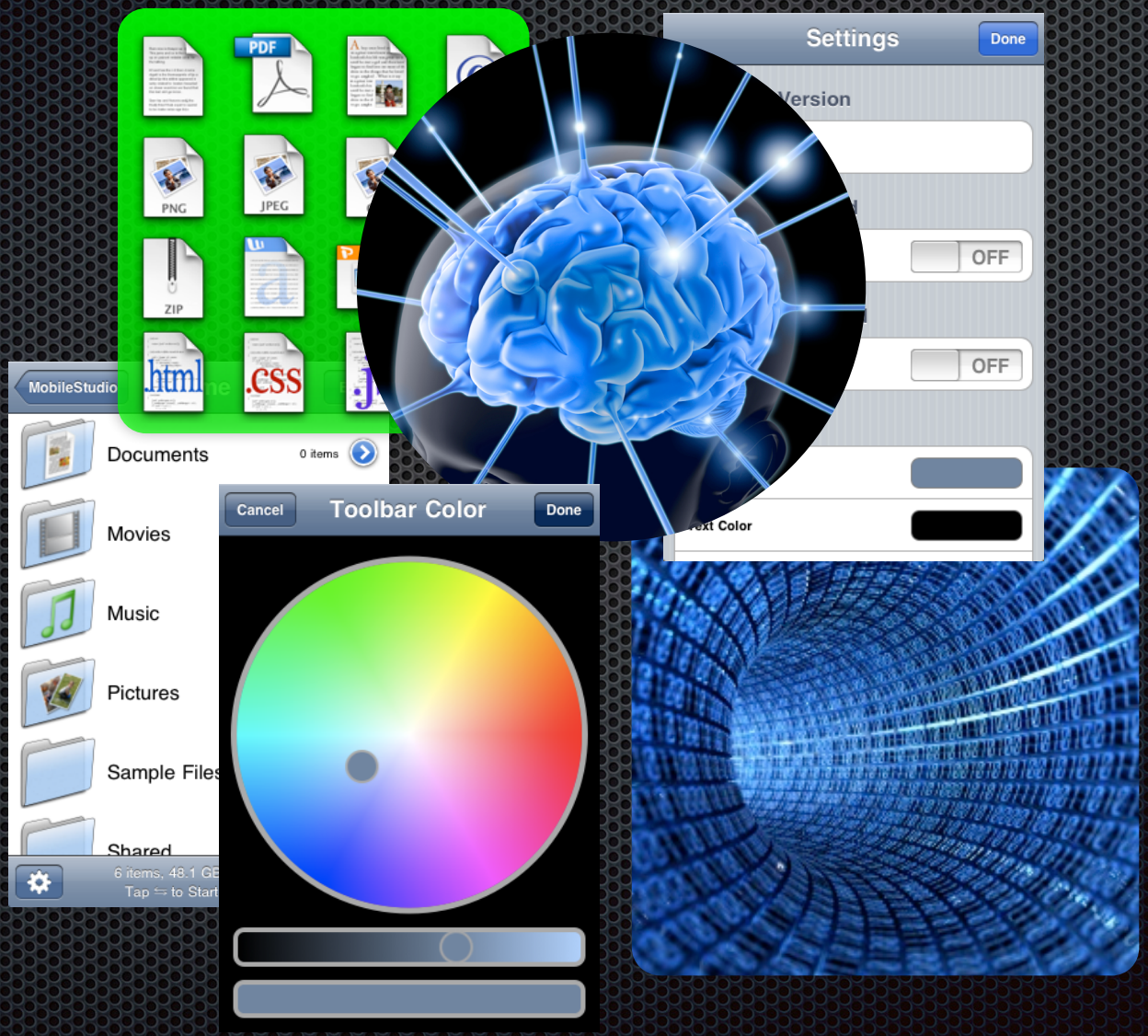
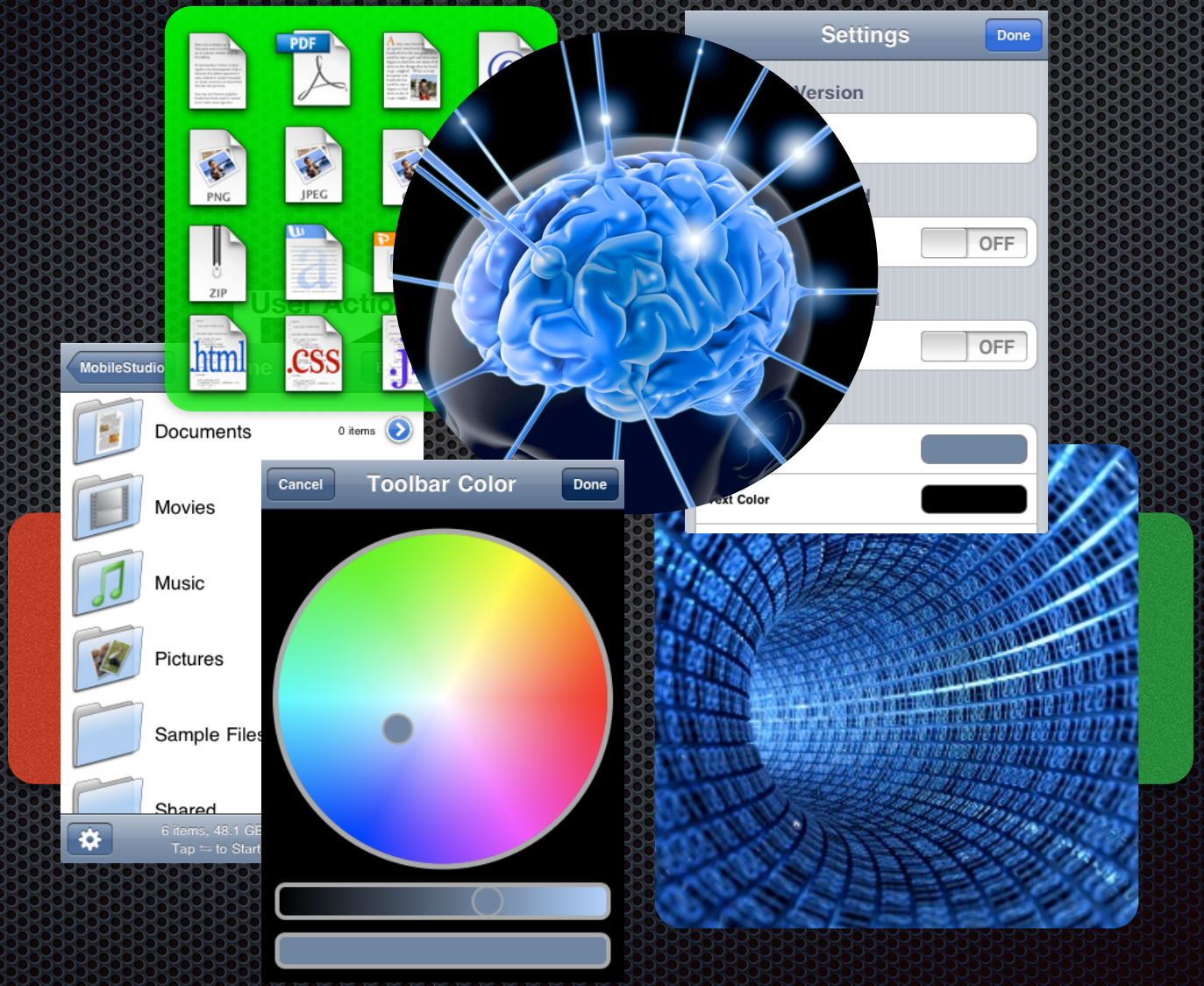
Mobile Application Programing: iOS

Messaging

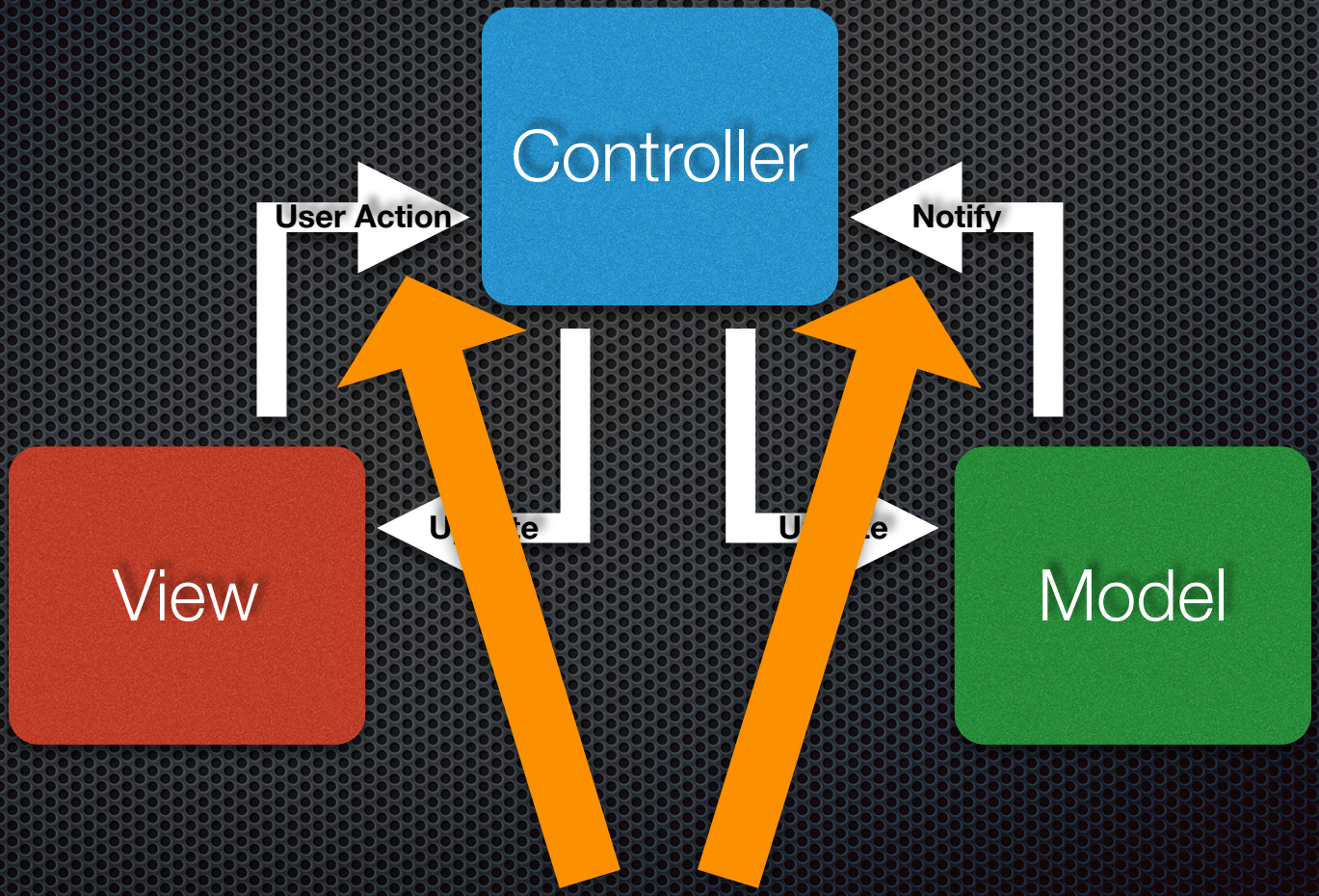
Application



Application Controller (MVC)

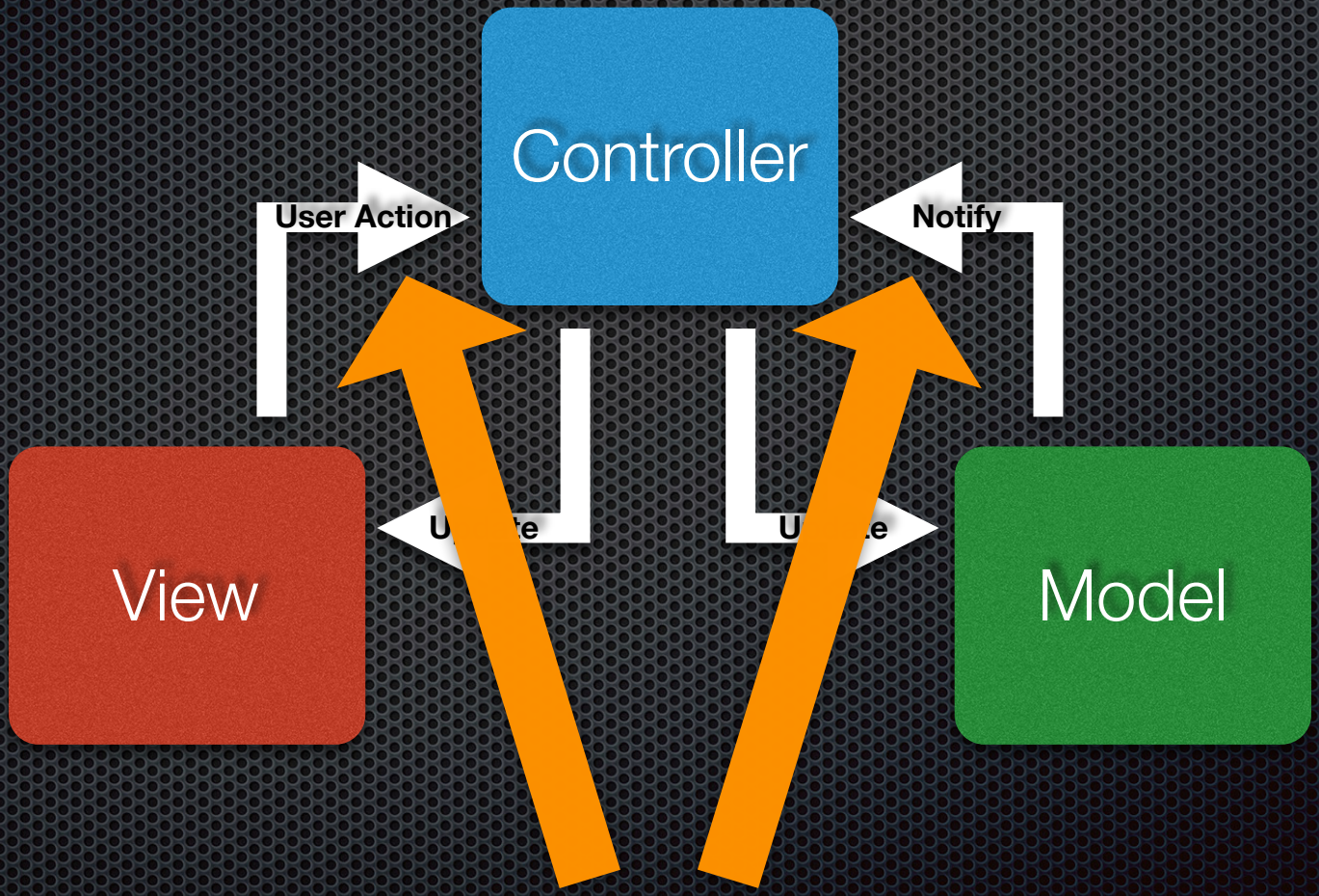


Messaging



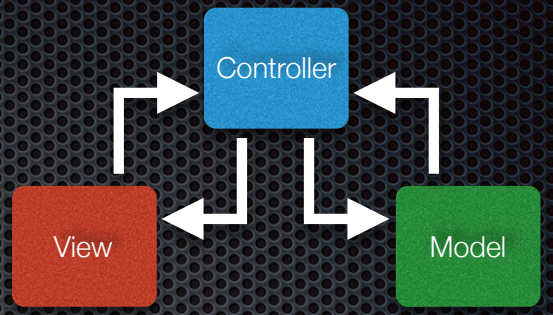
How do these happen?

Messaging

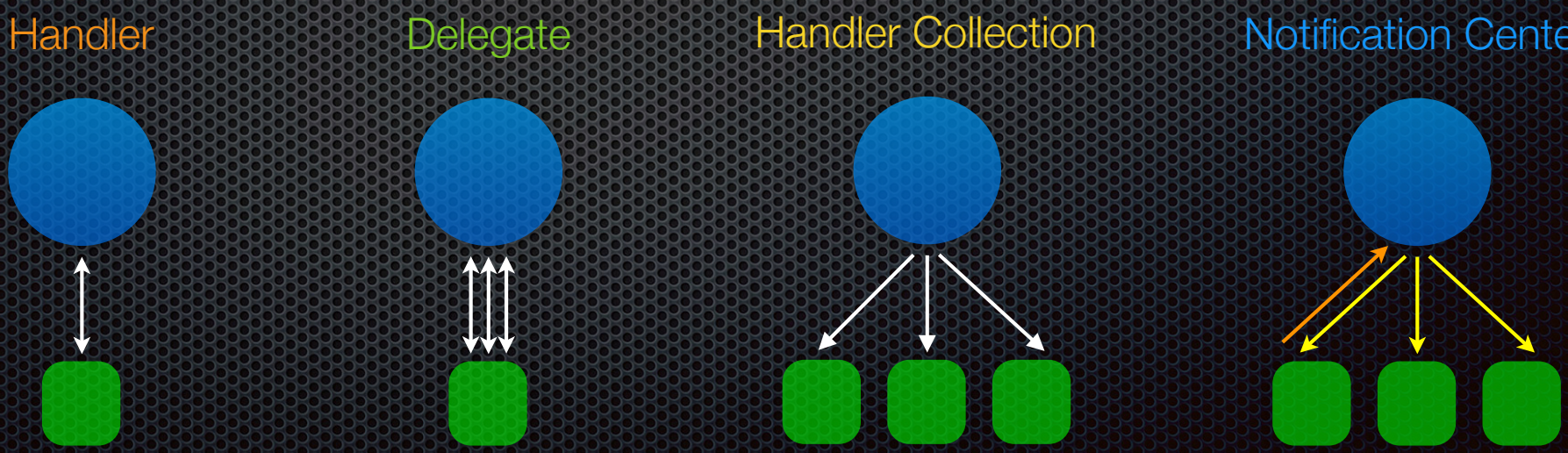


How do these happen? Delegation

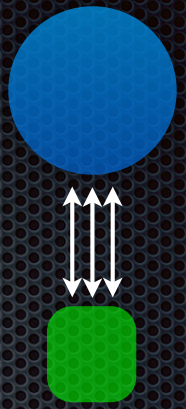
Messaging Options



- Delegates - a delegate property & delegation protocol
- Handlers - like single-method delegates but using a closure
- Handler Collection - a collection of handlers notified on events
- NSNotificationCenter - centralized notification dissemination



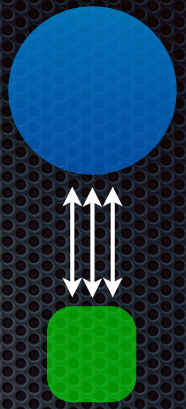
Delegates



- A delegate is an object that performs actions on the behalf of another object
- A common use is a data model object alerting a controller of changes to its data, which then tells view objects about the change
- Another use of them is a view object having a controller object interact with the program data model on its behalf when the user triggers events
- 6 bits of code are required to properly set up both sides of a delegate connection between two objects

```swift
import UIKit

protocol KnobDelegate: class
{
    func knob(knob: Knob, rotatedToAngle angle: Float)
}

class Knob : UIView
{
    private var _knobRect: CGRect = CGRectZero
    private var _angle: Float = 3.0 * Float(M_PI) / 2.0

    var angle: Float
    {
        get { return _angle }
        set
        {
            _angle = newValue
            setNeedsDisplay()
        }
    }

    weak var delegate: KnobDelegate? = nil

    override func touchesMoved(touches: NSSet, withEvent event: UIEvent)
    {
        let touch: UITouch = touches.anyObject() as UITouch
        let touchPoint: CGPoint = touch.locationInView(self)
        let touchAngle: Float = atan2f(
            Float(touchPoint.y - _knobRect.midY),
            Float(touchPoint.x - _knobRect.midX))

        angle = touchAngle
        delegate?.knob(self, rotatedToAngle: angle)
    }

    override func drawRect(rect: CGRect)
    {
    }
}
```

1. Delegate Protocol
2. Delegate Property
3. Delegate Invocation



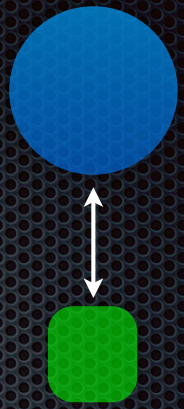
The method invocation here...

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, KnobDelegate
{
    var window: UIWindow?

    func application(application: UIApplication,
       didFinishLaunchingWithOptions l: [NSObject: AnyObject]?) -> Bool
    {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        window?.makeKeyAndVisible()

        var knob: Knob = Knob(frame: window!.frame)
        knob.backgroundColor = UIColor.darkGrayColor()
        knob.delegate = self
        window?.addSubview(knob)

        return true
    }

    func knob(knob: Knob, rotatedToAngle angle: Float)
    {
        println("Knob rotated to angle: \(angle)")
    }
}
```

goes here.

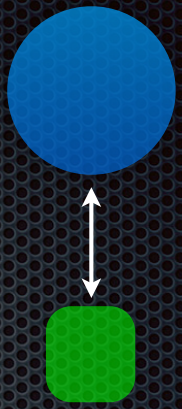
4. Delegate Protocol Conformity
5. Delegate Assignment
6. Delegate Protocol Method(s)

Handlers



- Implemented similarly to delegates but use a closure
- Keep event assignment and event code in the same location in the code file spatially
- Require 3 pieces of code instead of 6
  - 2 on the sending side
  - 1 on the receiving side
- Closure capture relationships need to be carefully considered to prevent memory leaks!

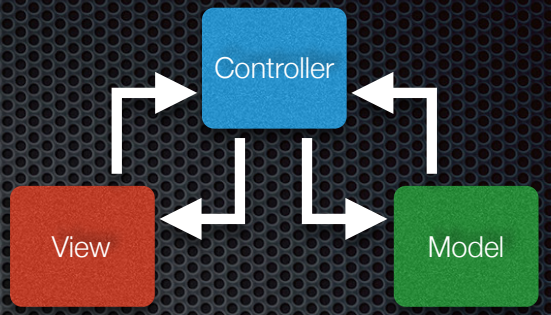
Handlers



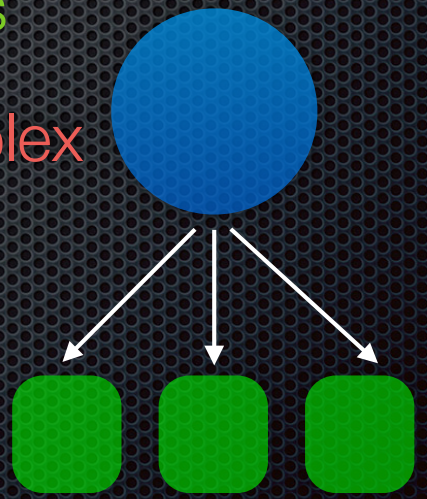
```
class PaintingCollection {
    private var _paintings: [Painting] = []

    // MARK: Indexing
    var paintingCount: Int {
        return _paintings.count
    }

    // MARK: Element Access
    func paintingWithIndex(paintingIndex: Int) -> Painting {
        return _paintings[paintingIndex]
    }

    func addPainting(painting: Painting) {
        // ...
    }

    func removePaintingWidthIndex(paintingIndex: Int) {
        // ...
    }

    func addStroke(stroke: Stroke, toPainting paintingIndex: Int) {
        // ...
        paintingChangedHandler?(paintingIndex)
    }

    // MARK: Events
    var paintingChangedHandler: ((_ paintingIndex: Int) -> Void)?
}
```

```
collection.paintingChangedHandler = {
    [weak self] (paintingIndex: Int) in
    self?.thingsListView.reloadData()
}
```

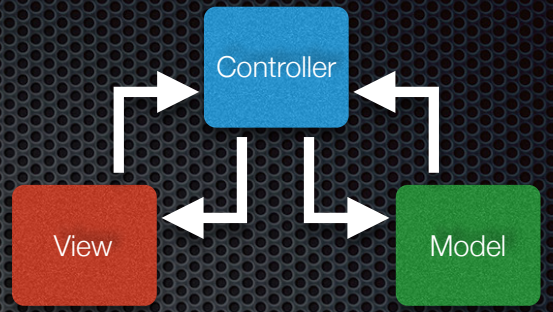
Handler Collection



- Create a collection of handlers
- When notifying a single handler, notify all handlers
- Note that this makes asking for information complex because all received data must be considered
- Example: Voting for president
  - Each voter asked for vote
  - Voter returns preference
  - Accounting of votes determines returned value

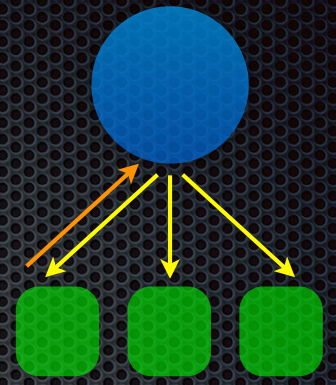


NSNotificationCenter



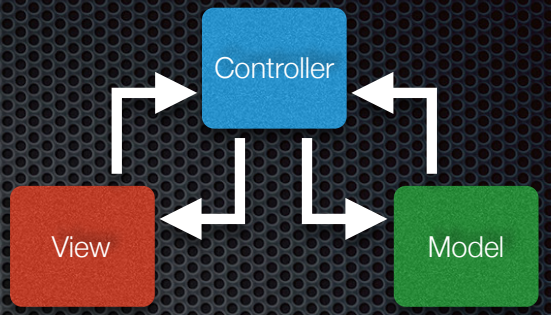
- Centralized system to register observers for named notifications and allow other objects to post notifications to the system
- Receivers may register / un-register as observers for receiving notifications at any time
- Sender can’t ask for information from receivers!

Cleanup: Essential!

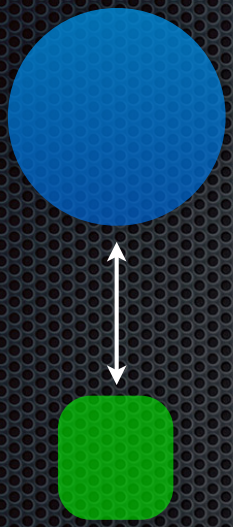


- When an object is no longer needed and should be deallocated, ensure it has un-registered itself as an observer. Otherwise, it will never be deallocated!
- The observer relationship is a strong reference to the object. When the object's other connections are removed, it will remain as NSNotificationCenter's reference is still active.
- E.g. a view controller that has been removed from a navigation controller will still be in memory if it is still an observer for a notification in NSNotificationCenter

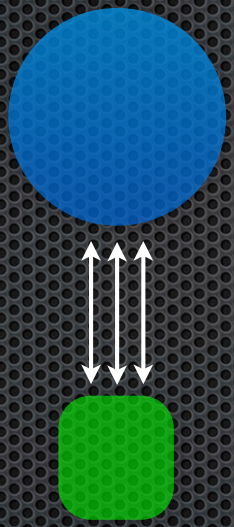
Messaging Options



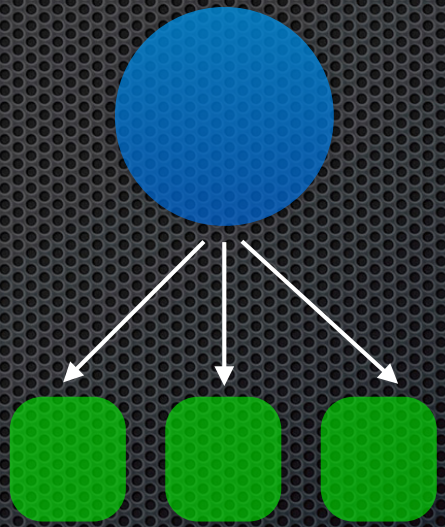
Handler



Delegate



Handler Collection



Notification Center