iOS DeCal : Lecture 2

Structure of iOS Applications: MVC and Auto Layout

Overview : Today's Lecture

Model View Controller Design Pattern

Creating Views in Storyboard

Connecting your Views to Code

Auto Layout

Announcements

- Enroll in the Course!

- enroll through CalCentral (you will not be automatically enrolled in the course)
- CCN and more info can found on Piazza

- Sign in on Piazza!

- No Lecture next week (still meeting for Lab on 2/16)

- Lab 1 is due TONIGHT if you did not get checked off

- Submit via Gradescope
- We will not be able to post grades for Lab 1 on GradeScope, but will for future labs

- Attendance Google Sheet?

Model View Controller

Overview : Software Design Patterns

Software Design Patterns are reusable solutions to common problems in software design

Main Idea: Assign objects in your application distinct roles using well-defined patterns and object relationships

There are many different types of patterns (not just basic MVC)!

Model View Controller (MVC)

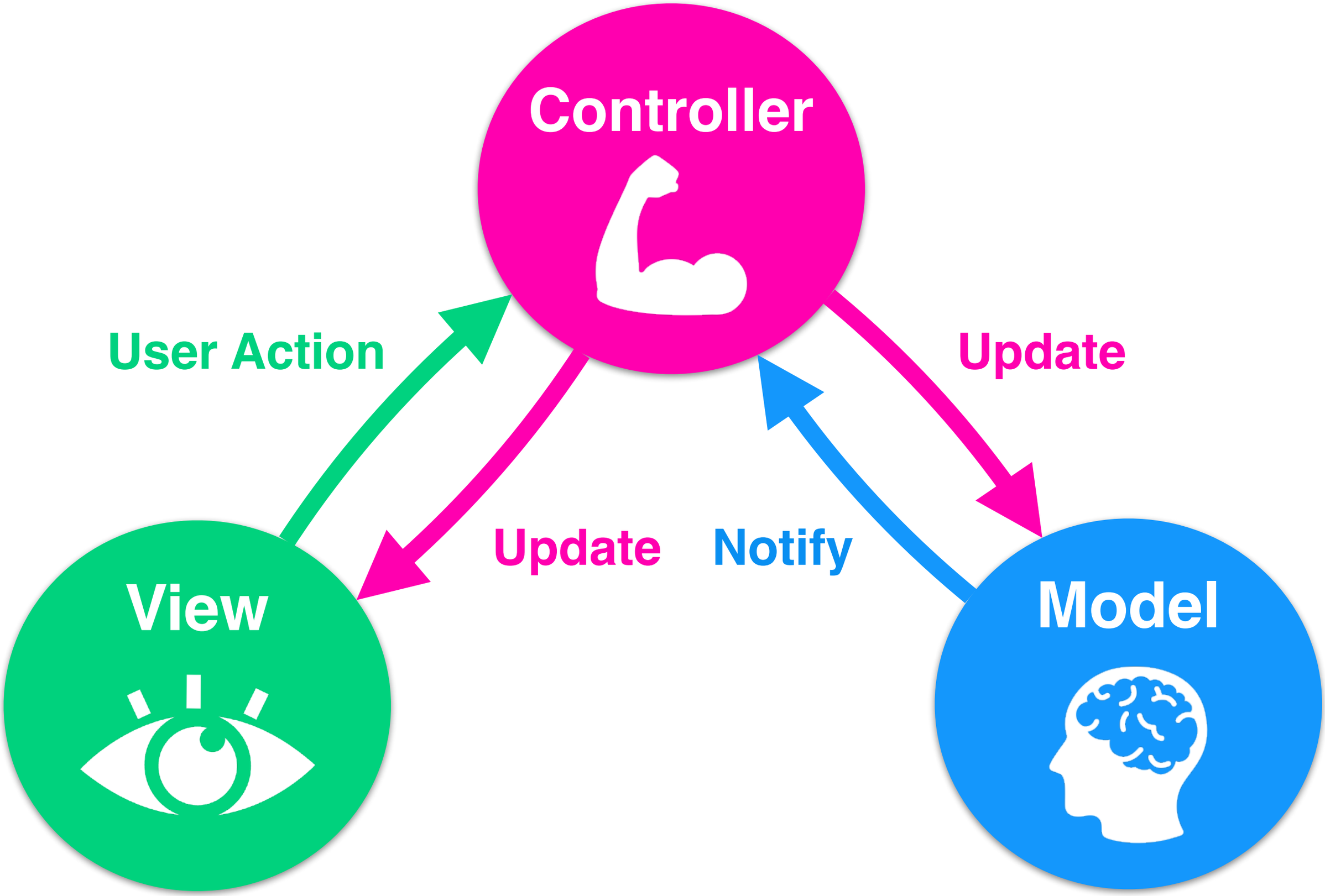
Common Design Pattern in Cocoa Applications

Assigns objects in your application one of the following distinct roles:

Model - encapsulates data and defines logic / computations

View - what the users see and interact with

Controller - intermediary between models and views



Model View Controller

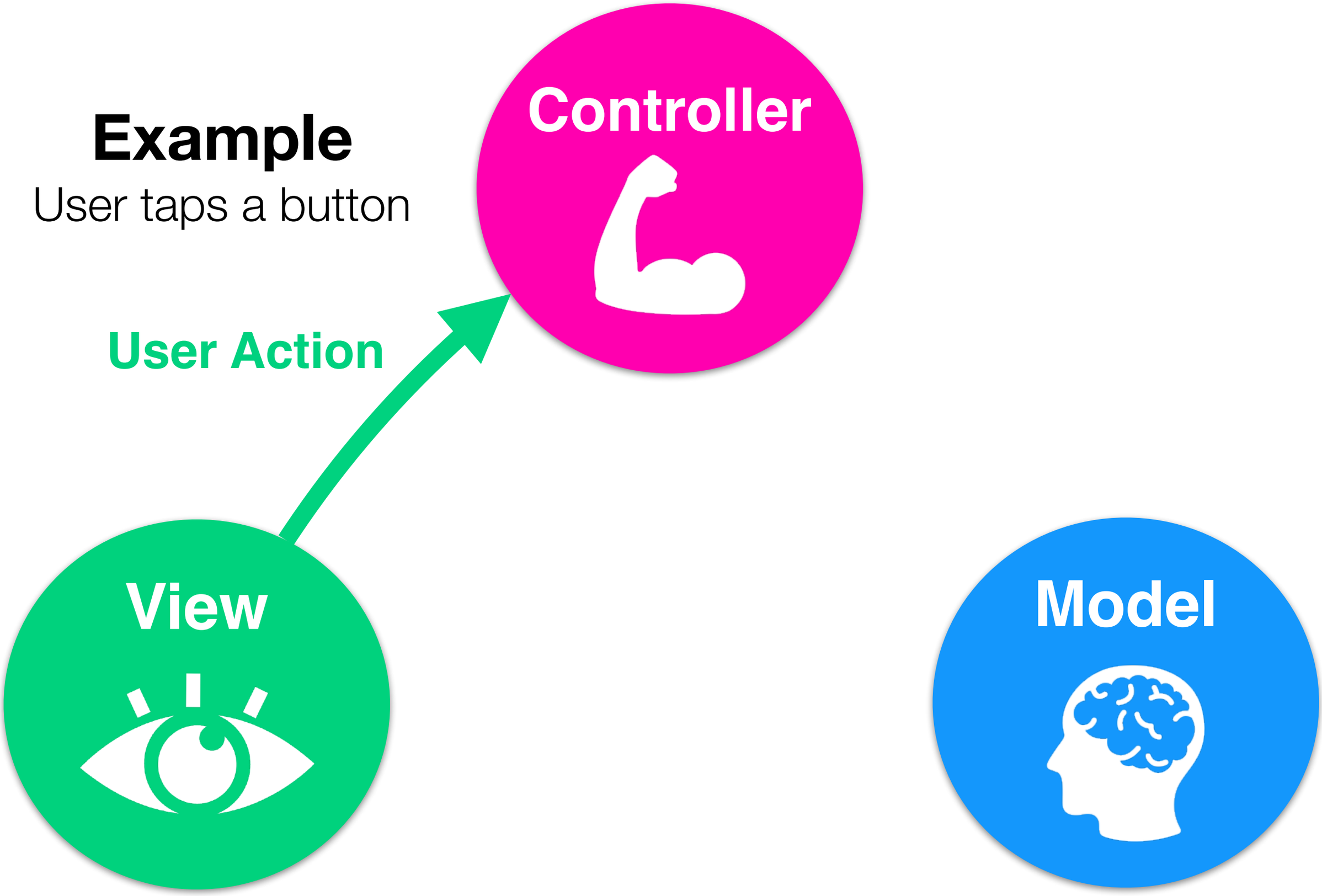
Example

User taps a button

User interacts with **View** (eg user taps “=“ button in calculator)

Example

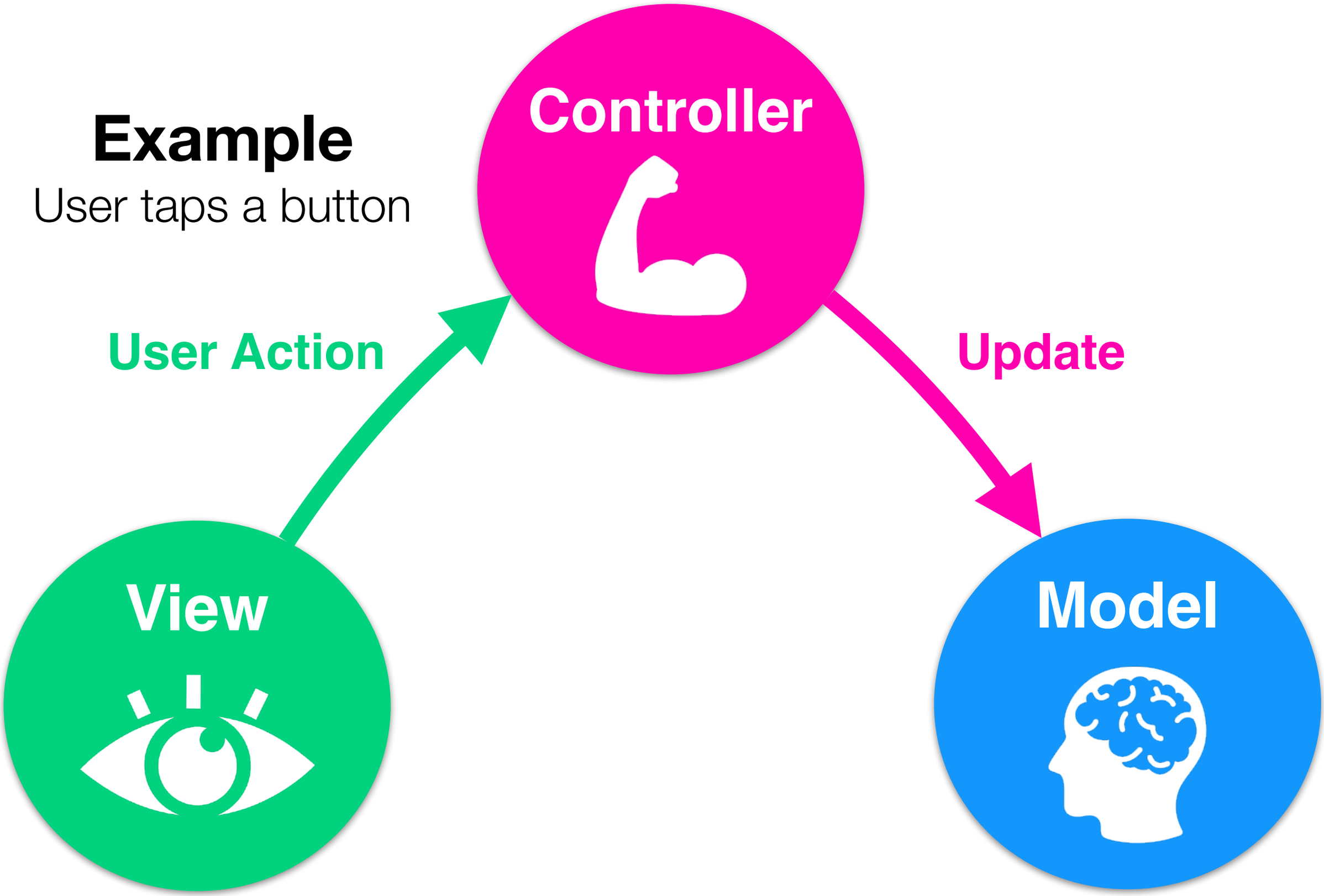
User taps a button



Controller is notified that the user has made an action

Example

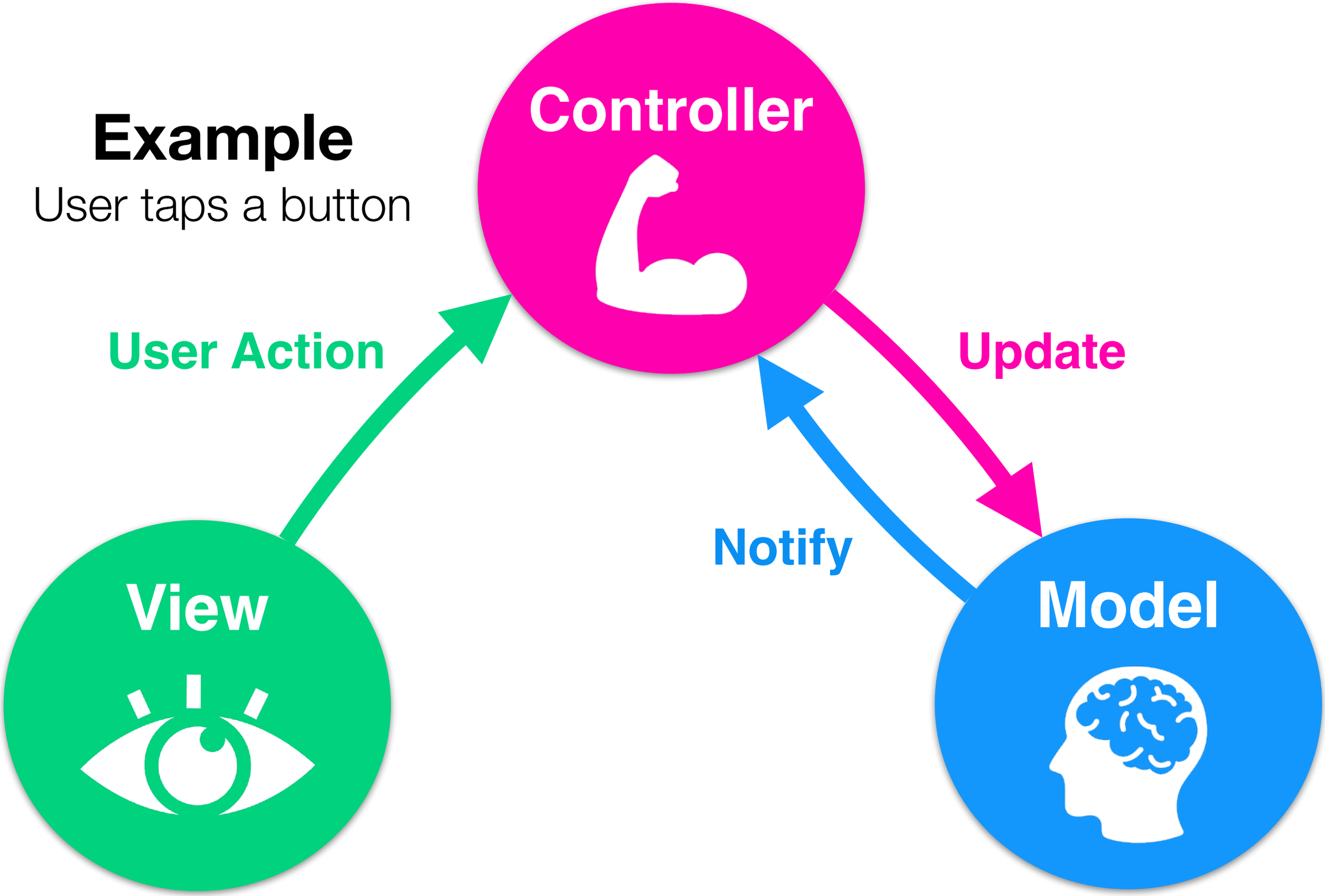
User taps a button



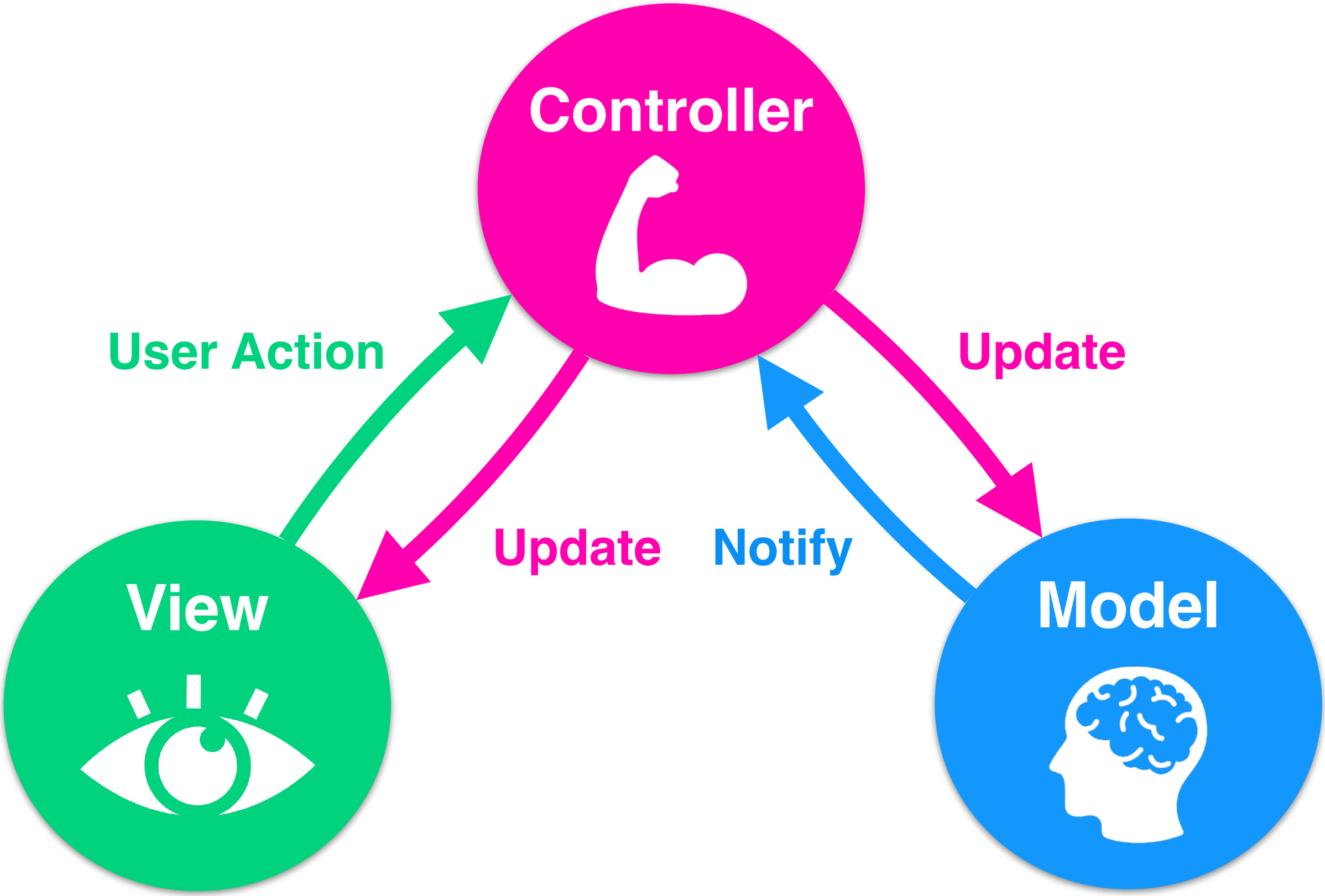
Controller updates the **Model** to reflect the users change

Example

User taps a button

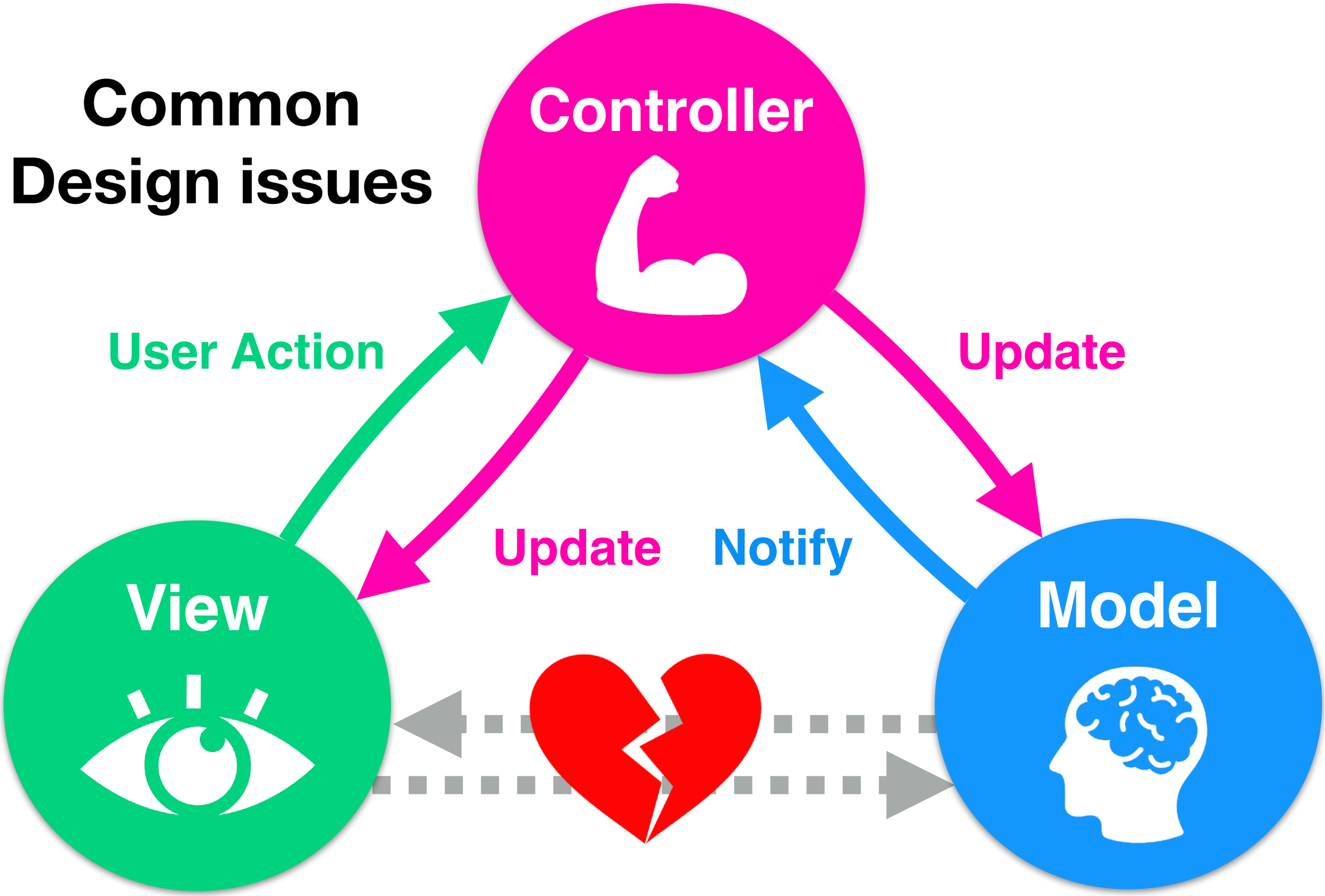


Model stores info / calculates then notifies controller of result



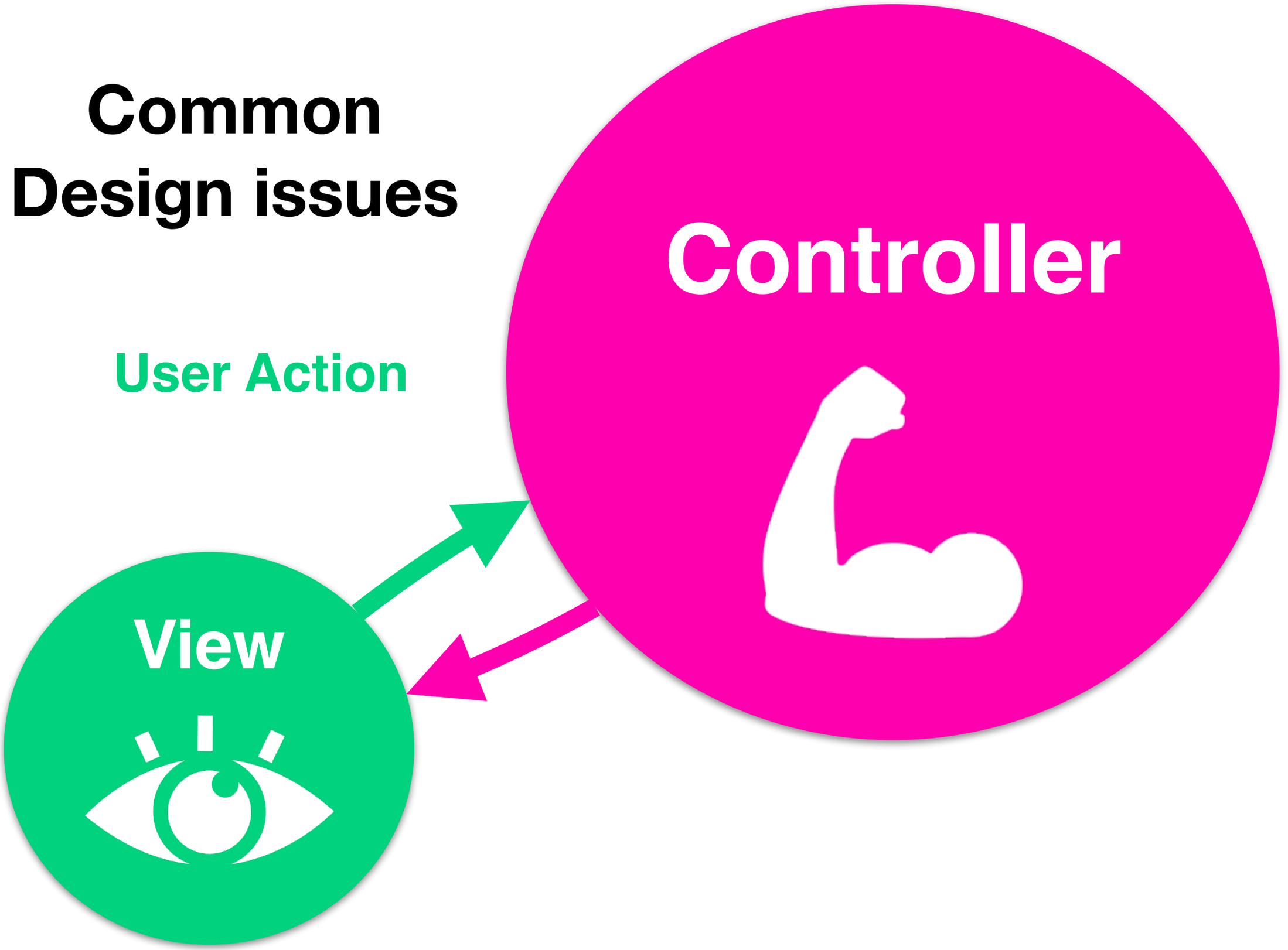
Notice how there is no direct connection from **Model** to **View**!

Common Design issues



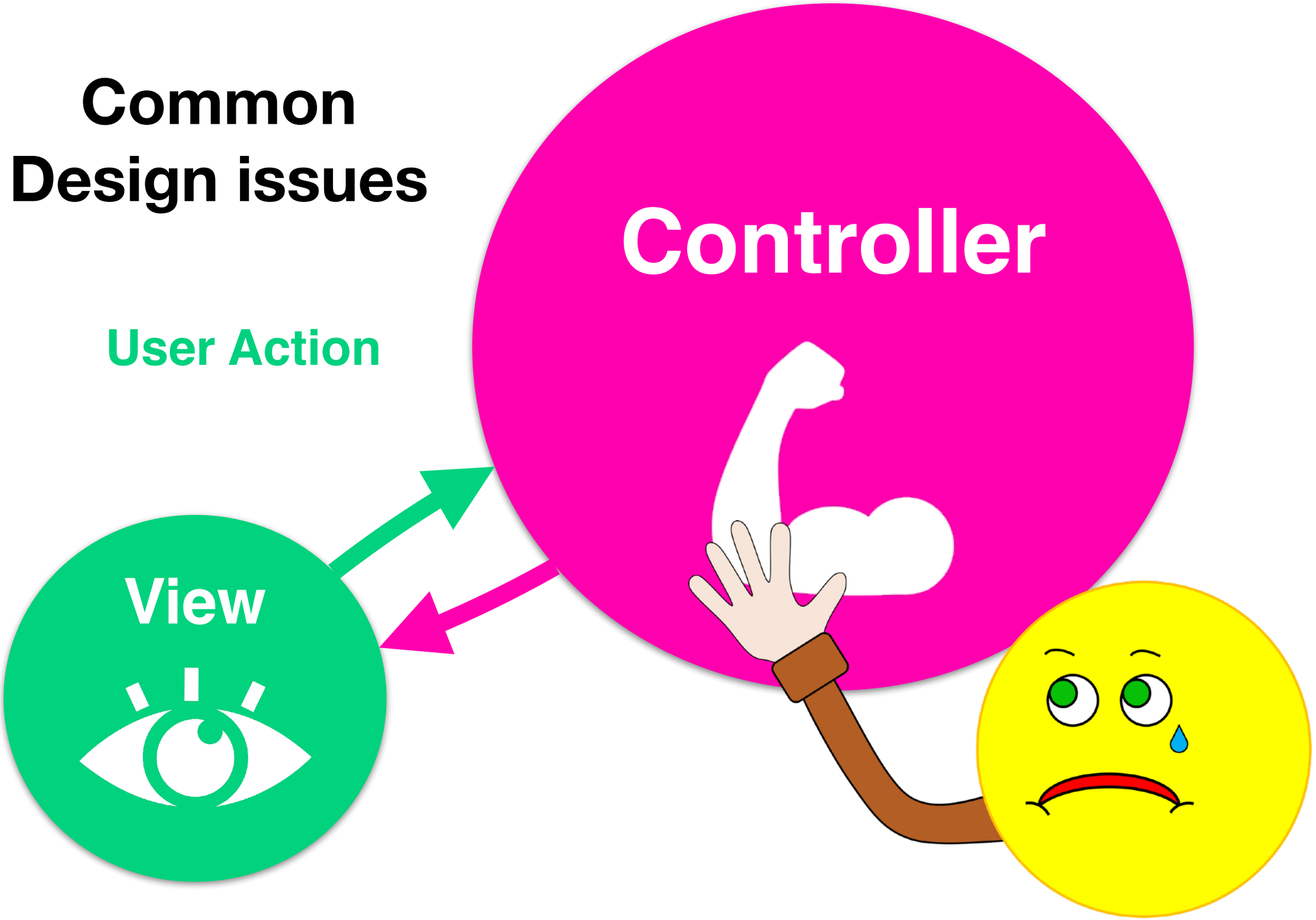
Issue #1 : Breaking Model / View Abstraction

Common Design issues



Issue #2 : Bloated Controllers

Common Design issues



Issue #2 : Bloated Controllers

MVC in iOS / Xcode

Model : data, logic, and computation

Puppy.swift — Edited

Puppy.swift 〉 No Selection

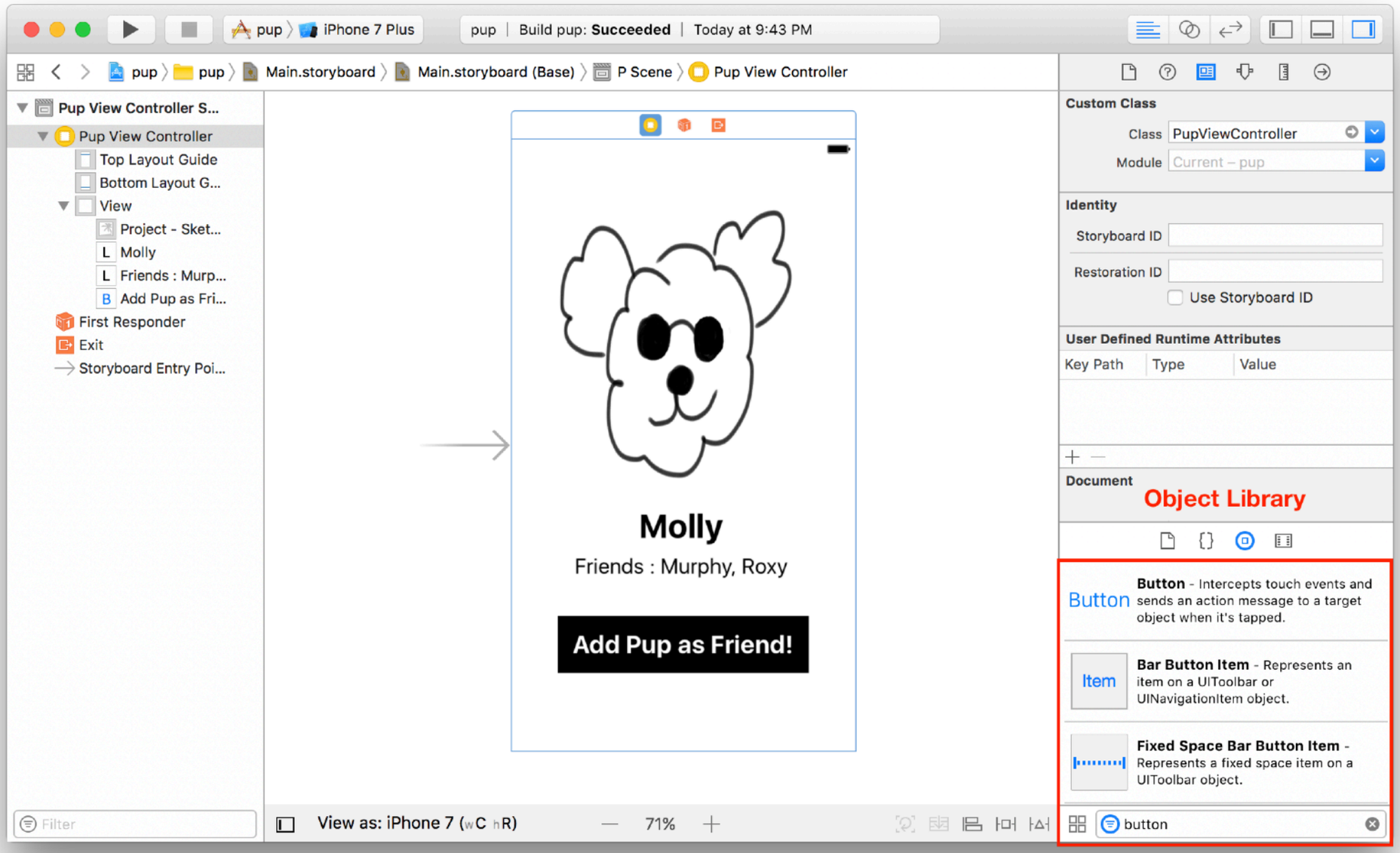
```swift

// Puppy.swift

import Foundation

/// Model Class for Pups
class Puppy {

    var name: String
    var friendList: [Puppy]?
    let location: Location?
    var superPower: SuperPower?

    init(pupName: String) {
        name = pupName
    }

    func bark(direction: Direction) {
        // ...
    }

    func makeNewFriend(withOtherPup pup: Puppy) {
        // ...
    }

    func activateSuperPower(superPower: SuperPower?) {
        // ...
    }
}

```

Swift File

To Create

New > File > Swift

View (Interface Builder) : What the user sees



In Storyboard - create views from the **Object Library**

View (Programmatic) : what the users see

PupView.swift — Edited

PupView.swift 〉 No Selection

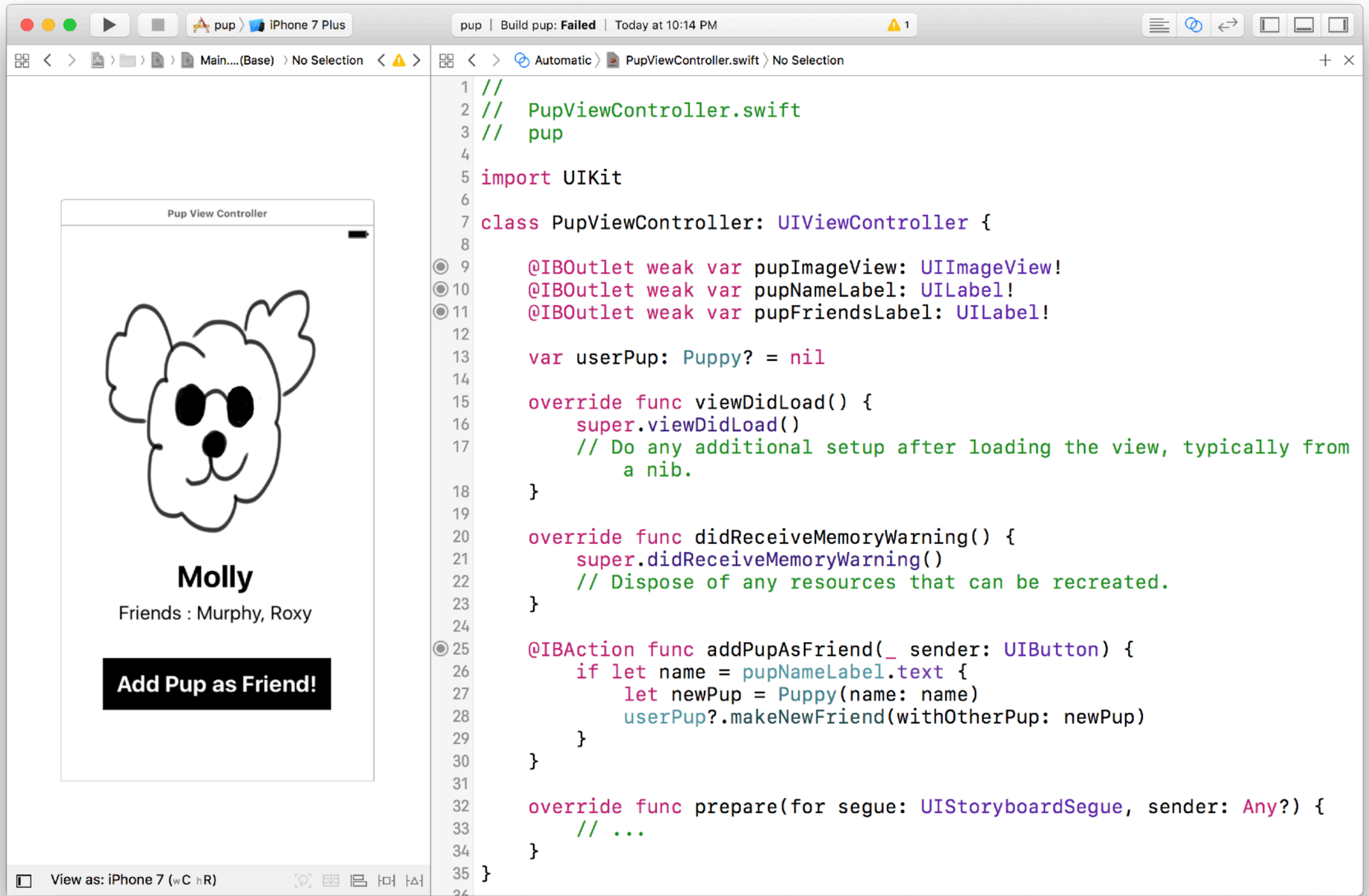
```swift
//  PupView.swift

import UIKit

class PupView: UIView {

    struct Constants {
        static let marginSize: CGFloat = 8
        static let buttonText: String = "Add Pup as Friend!"
    }

    let pupView = UIImageView(image: UIImage(named: "pup"))
    let friendsListLabel = UILabel()

    var addPupFriendButton: UIButton = {
        var button = UIButton()
        button.setTitle(Constants.buttonText, for: .normal)
        return button
    }()

    override init(frame: CGRect) {
        super.init(frame: CGRect.zero)
        addSubview(pupView)
        addSubview(friendsListLabel)
        addSubview(addPupFriendButton)
        setupConstraints()
    }

    func setupConstraints() {
        pupView.topAnchor.constraint(equalTo: topAnchor,
            constant: Constants.marginSize)
        // ...
    }
}
```

Cocoa Touch Class

Creating views programmatically

New > File > Cocoa Touch Class. Then subclass an existing type of View

Controllers in Xcode > **View Controllers**



Creating a Custom View Controller

(Interface Builder)

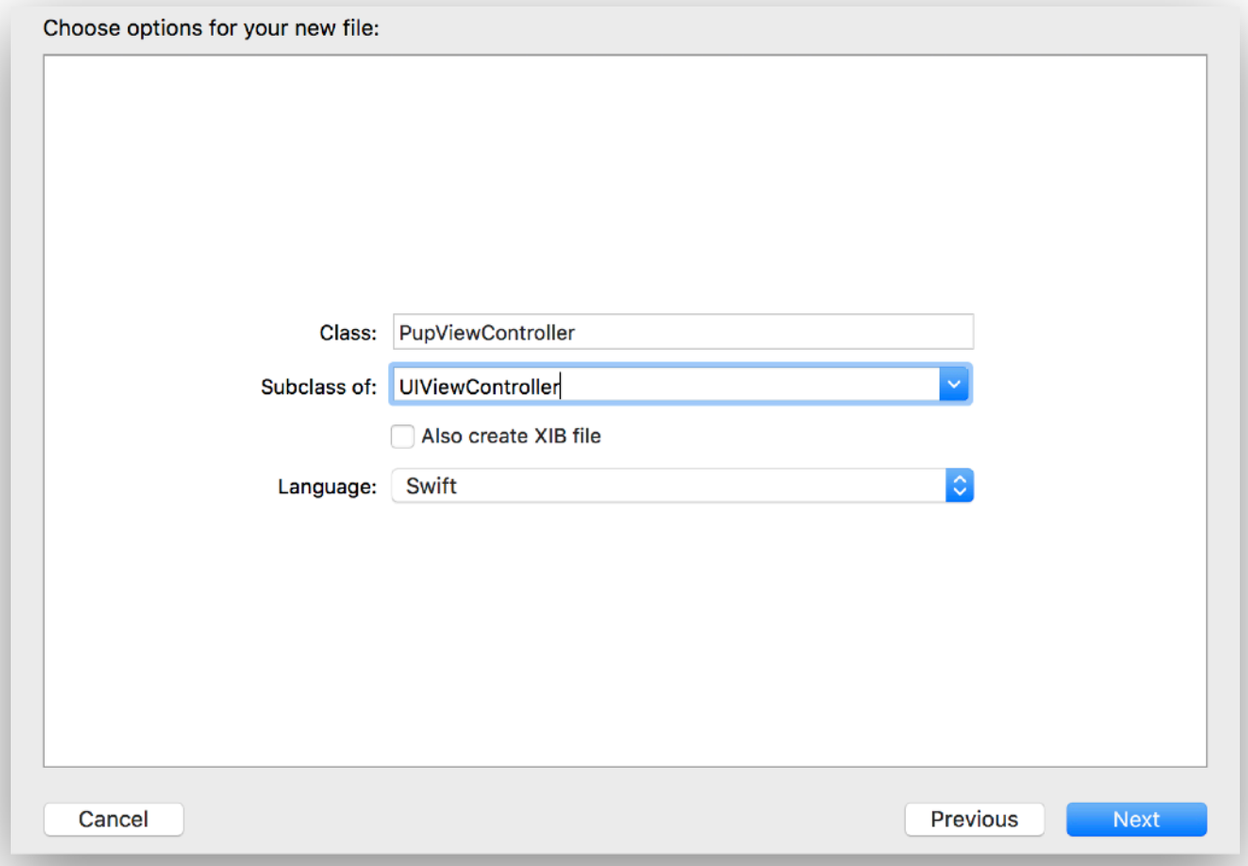
Creating View Controllers : Step 1

Create a View Controller Class for your custom View Controller

Cocoa Touch
Class

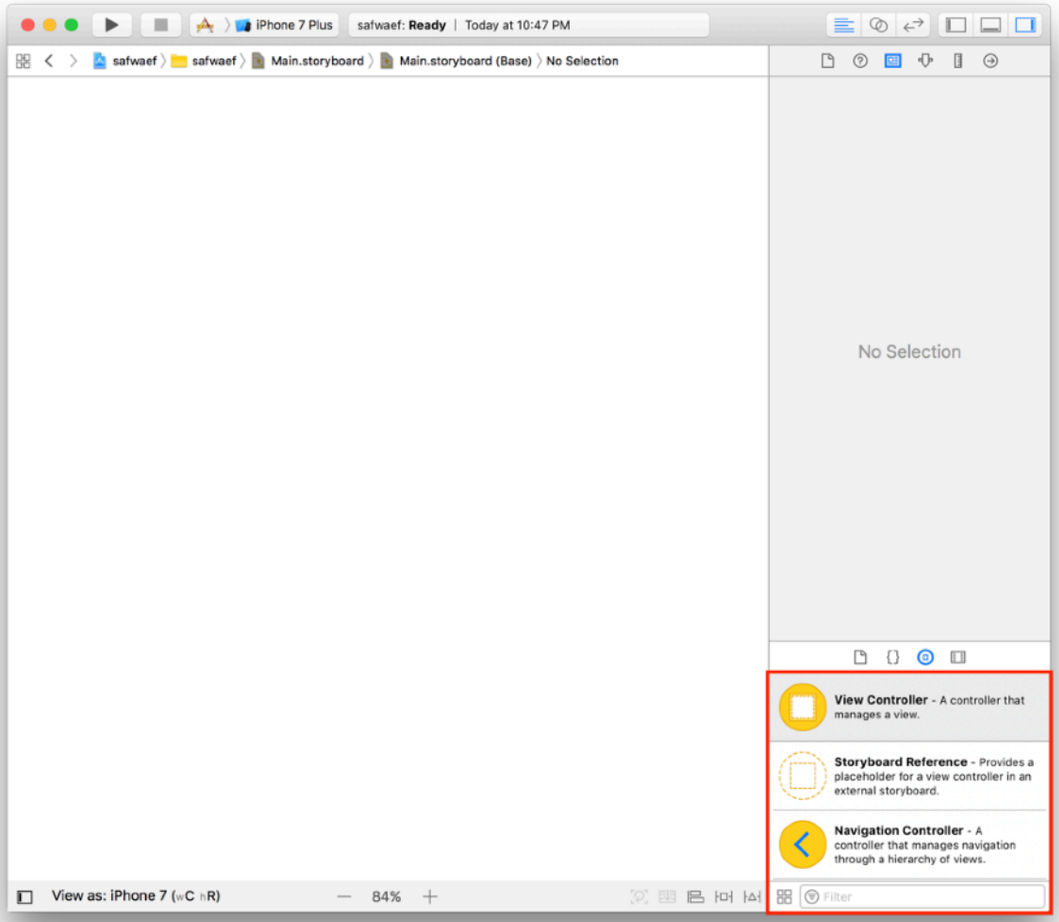
To create

New > File > Cocoa Touch Class. Then subclass an existing View Controller

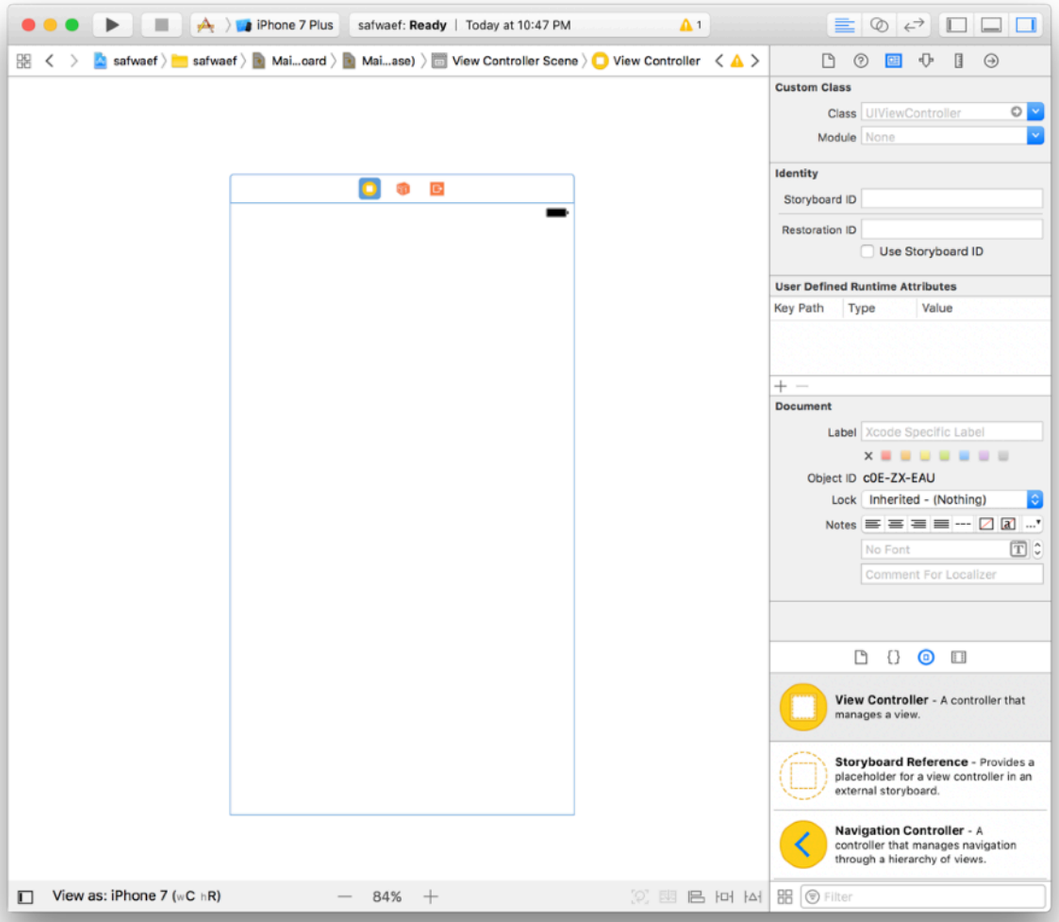


Creating View Controllers : Step 2

Drag a View Controller from your **Object Library** into your Storyboard



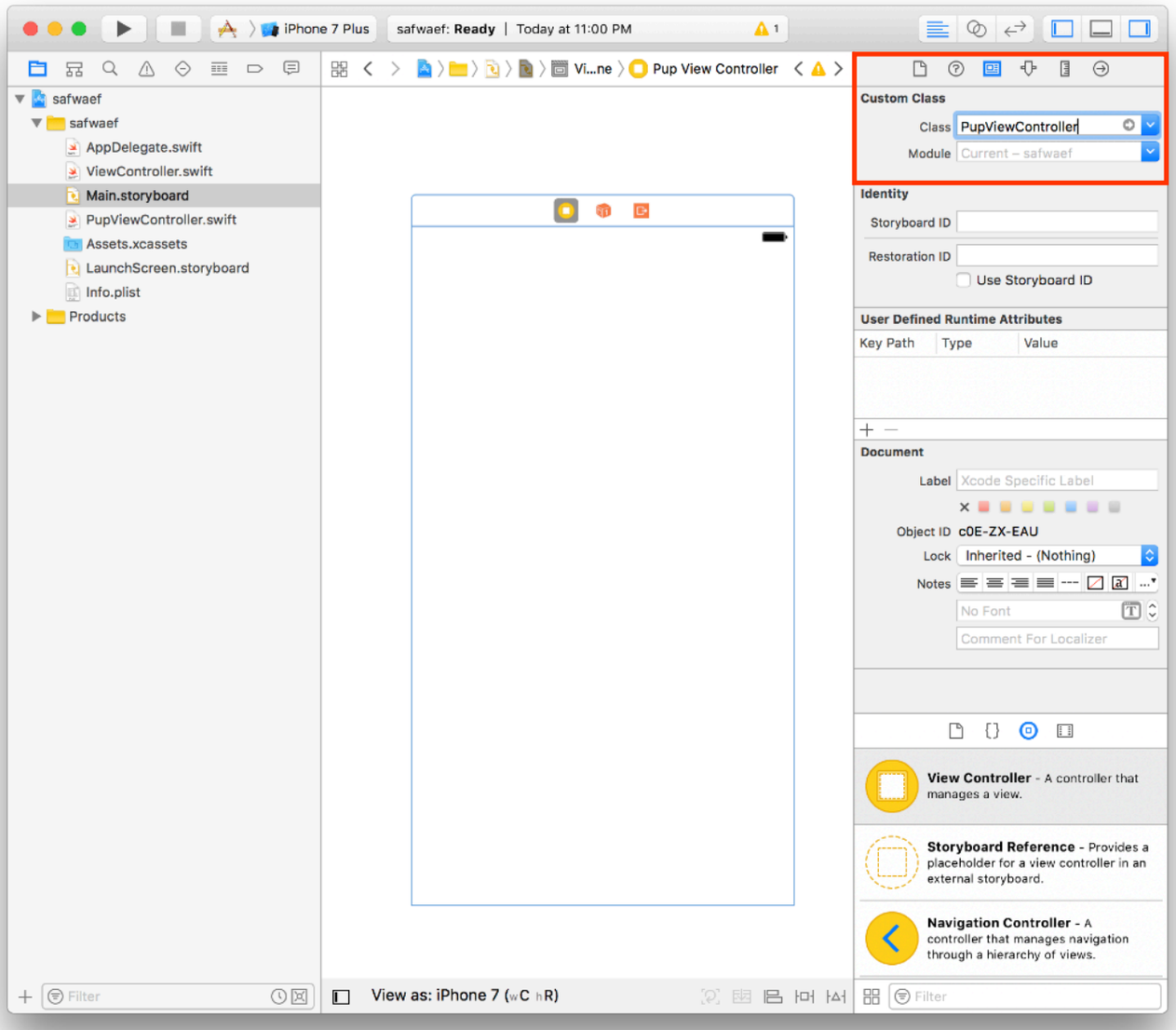
Open Main.storyboard, and navigate to the **Object Library**



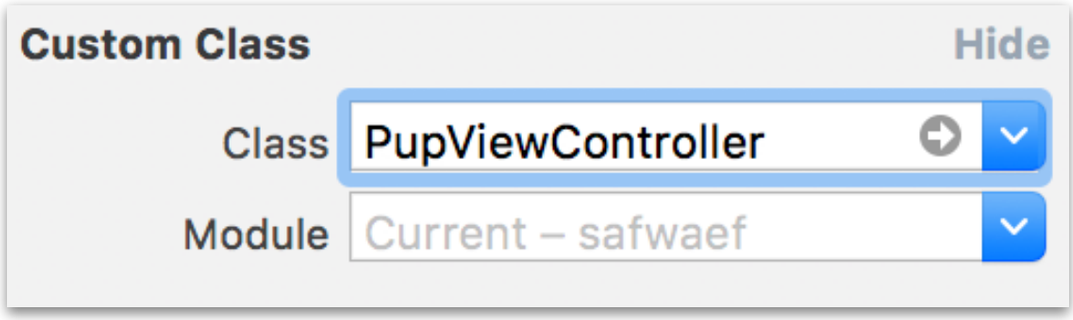
Drag a **View Controller** onto your Storyboard

Creating View Controllers : Step 3

Set the View Controller's class to the custom class you created in Step 1 **(don't forget this step)!**



Tap on your View Controller in the Storyboard, then open the **Identity Inspector**



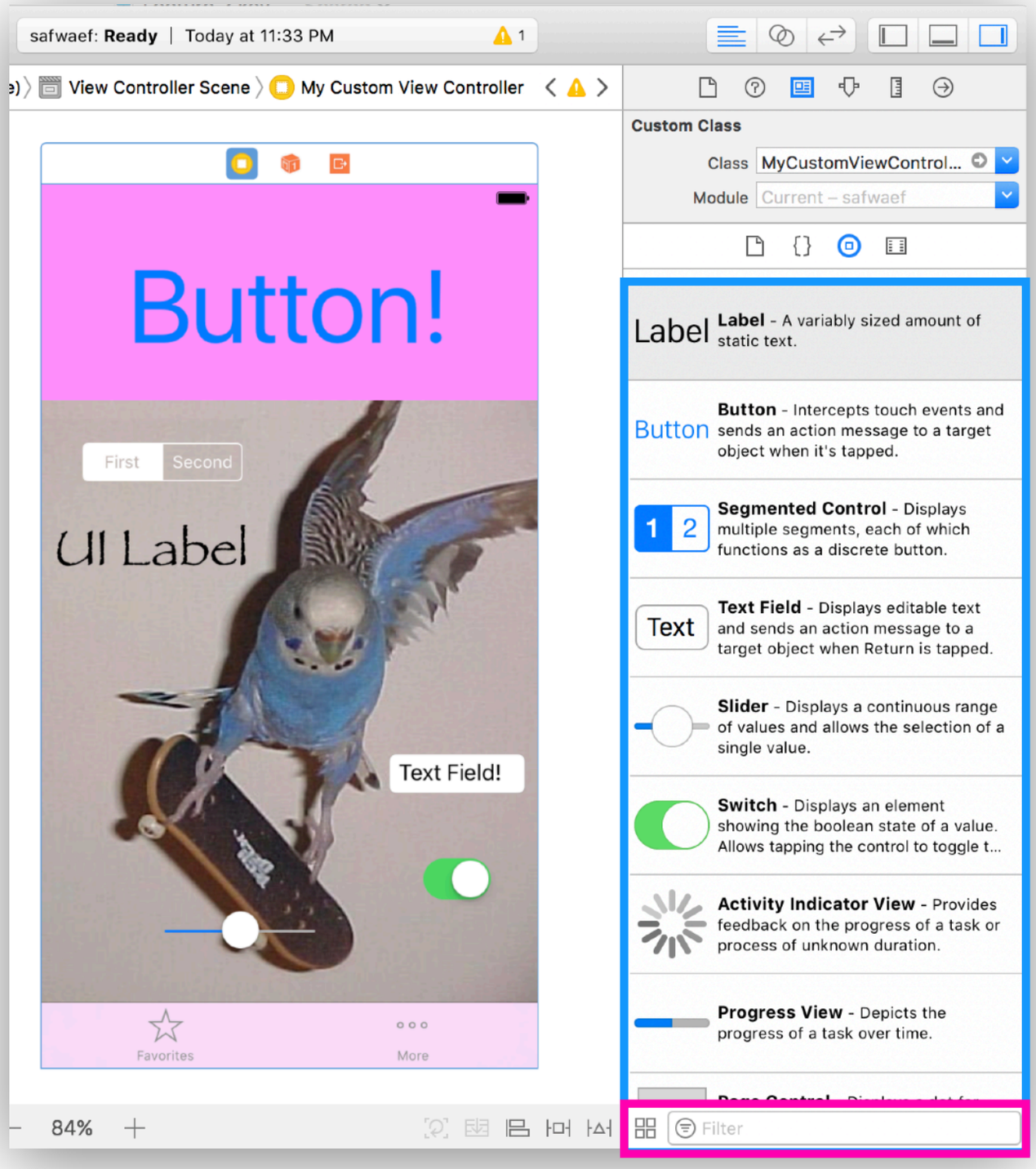
Change the Class field (found in the **Identity Inspector**) from *ViewController* to your custom View Controller's name

Creating View Controllers : Step 4

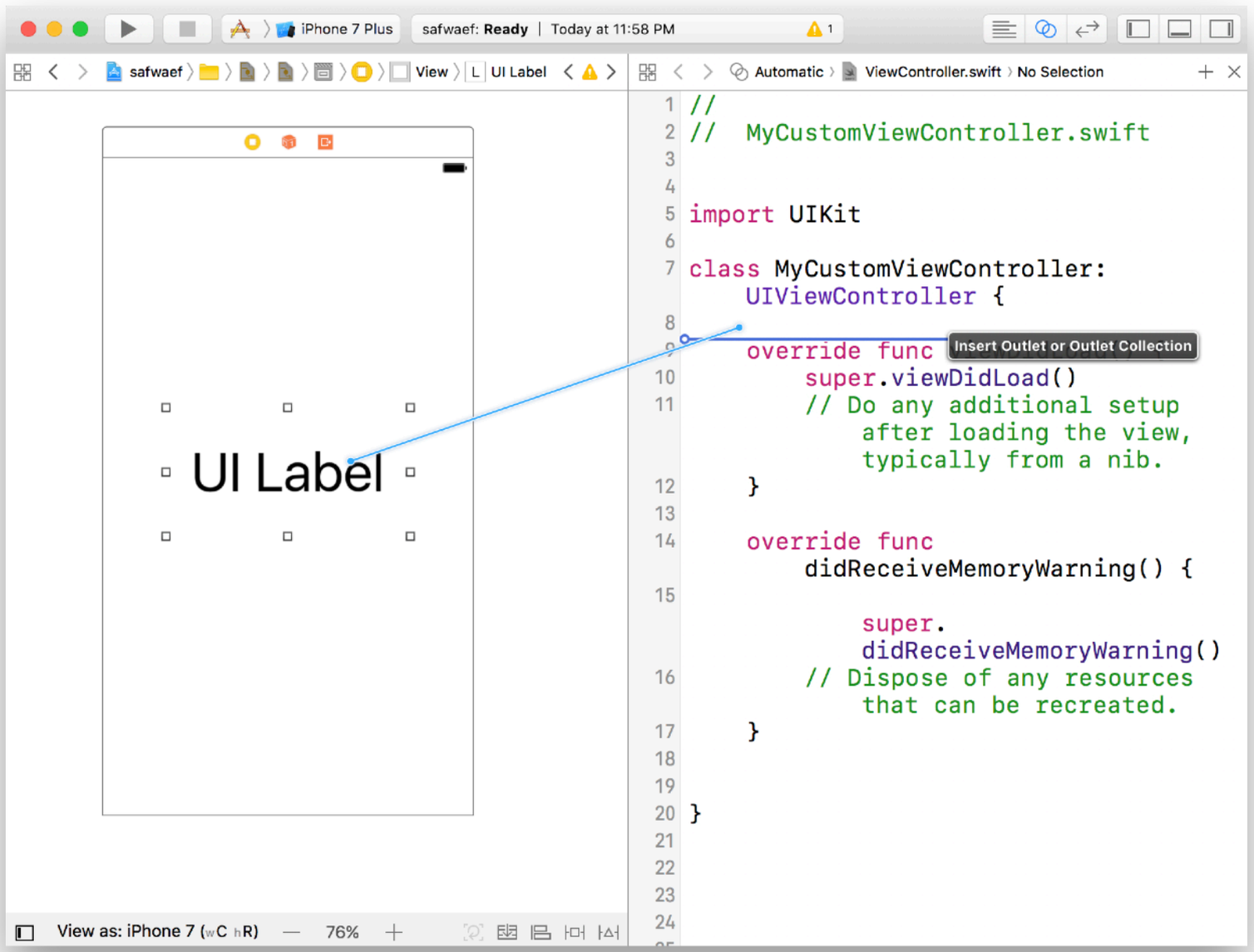
Your new View Controller is all set to go!

Now you can start adding views from the **Object Library** and customizing your View.

Search for specific objects in the **Filter Search Bar**

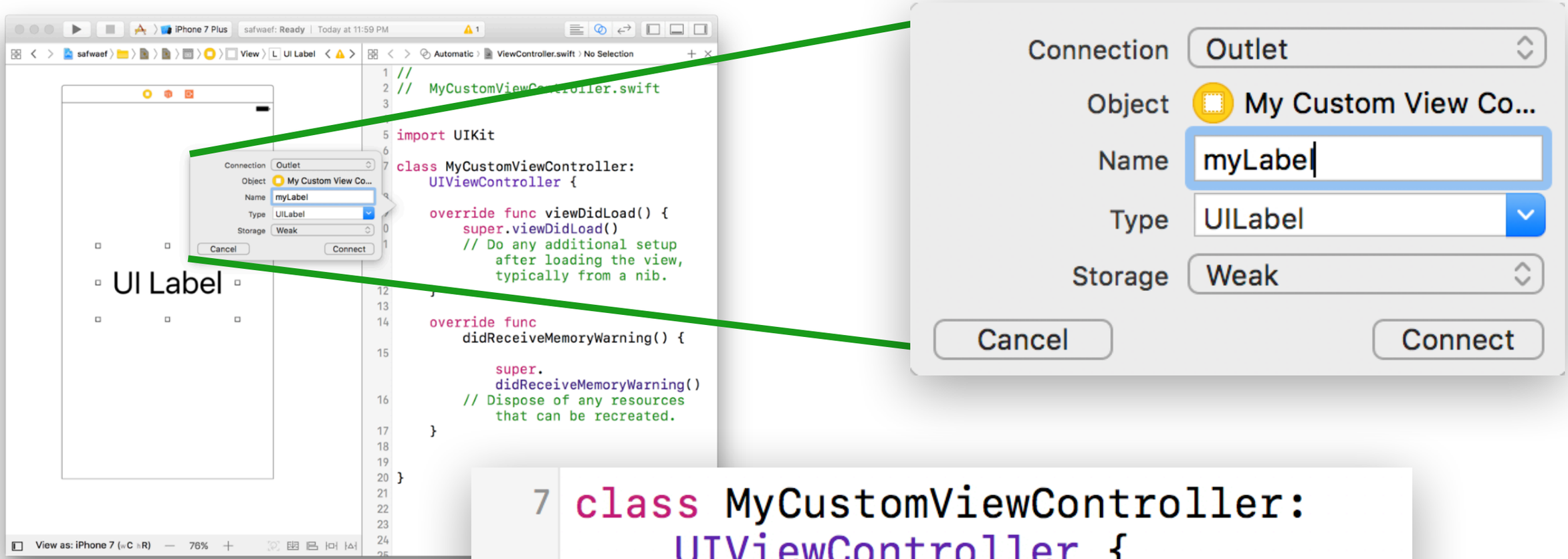


Control Drag in Interface Builder



Control + Drag to connect UI elements in Storyboard to your code

Control Drag : Outlets



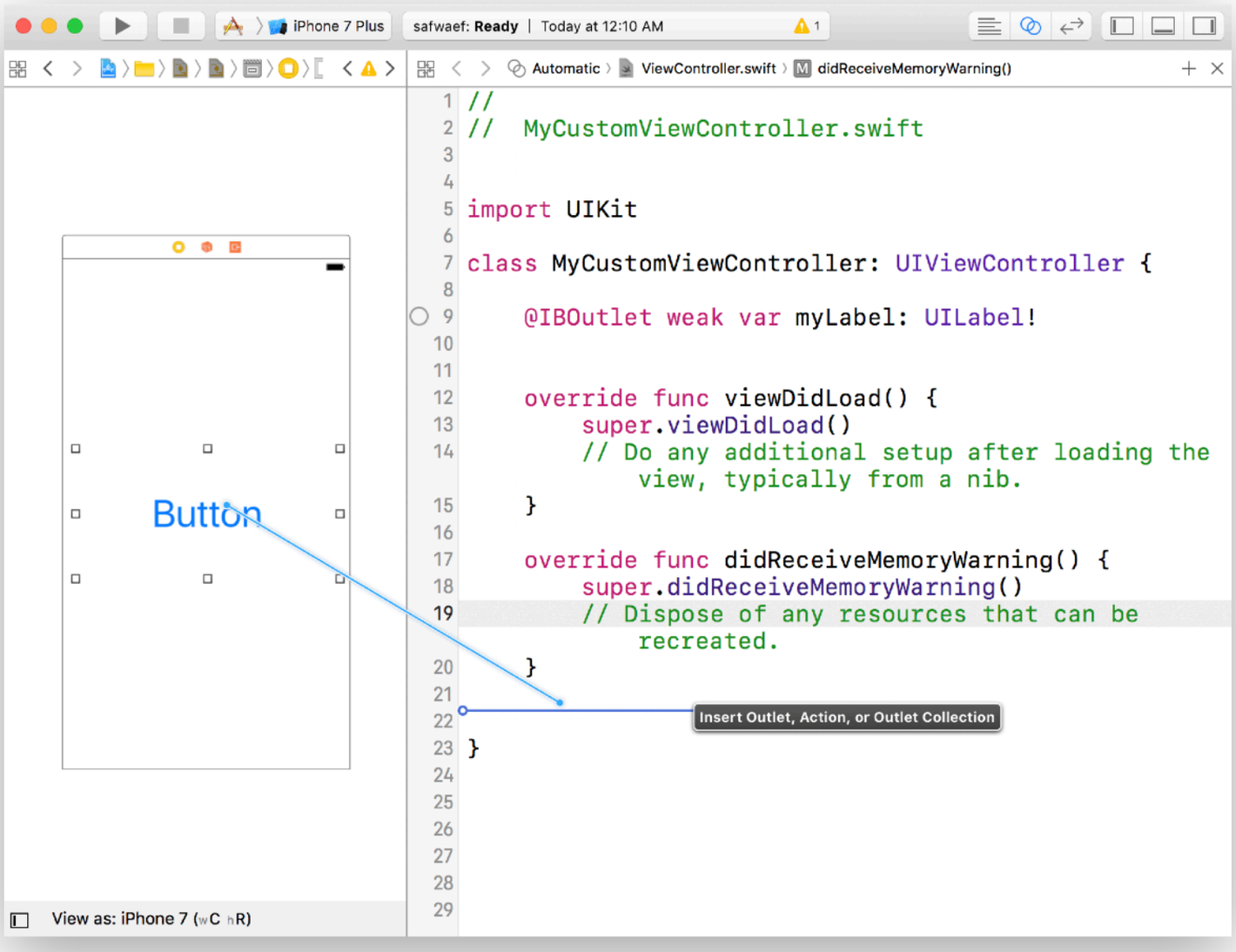

```
//
//  MyCustomViewController.swift

import UIKit

class MyCustomViewController:
    UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup
            after loading the view,
            typically from a nib.
    }

    override func
        didReceiveMemoryWarning() {

        super.
        didReceiveMemoryWarning()
        // Dispose of any resources
            that can be recreated.
    }

}
```

```
7  class MyCustomViewController:
       UIViewController {
8
9      @IBOutlet weak var myLabel:
           UILabel!
```

Pressing **Connect** generates an **Outlet** (linked to your storyboard)

Control Drag : Actions

If you want your UI element to **DO** something when tapped, highlighted, changed, etc. create an **Action**

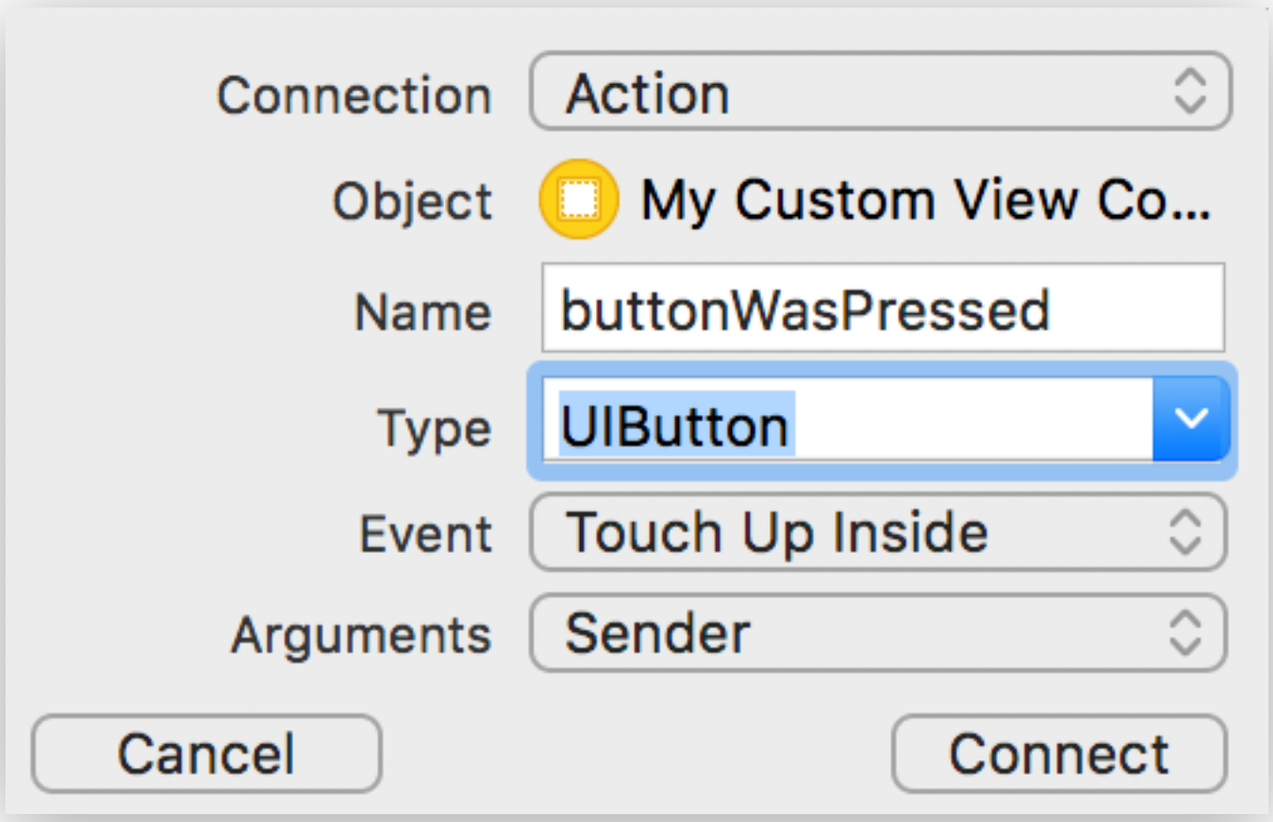



```
//
//  MyCustomViewController.swift

import UIKit

class MyCustomViewController: UIViewController {

    @IBOutlet weak var myLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be
            recreated.
    }

}
```

Example: UIButton

Start by **Control + Dragging** from button your code (just as you would with an Outlet)

Control Drag : Actions

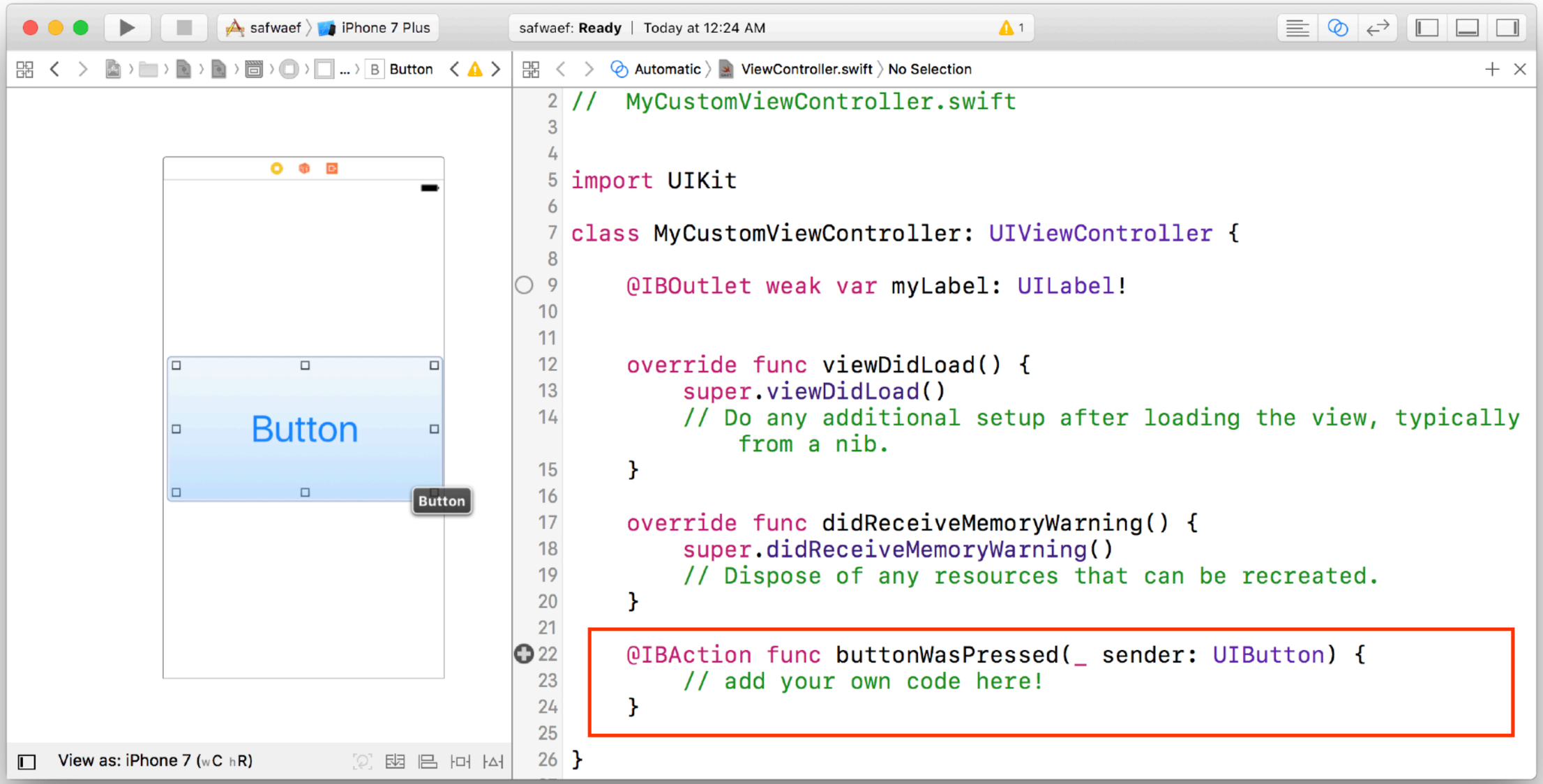


Set the Connection type to **Action** and Type (it's referring to Sender Type) to **UIButton**

Other values should default to the ones shown above.

Control Drag : Actions

Pressing *connect* will generate a method for you in your file that will be called every time the user taps your button.



Control Drag : Actions

Pressing *connect* will generate a method for you in your file that will be called every time the user taps your button.

You can access the button itself by modifying `sender`

```
@IBAction func buttonWasPressed(_ sender: UIButton) {
    // add your own code here!

}
```

Control Drag : Actions

Pressing done will generate a method for you in your file that will be called every time the user taps your button.

You can access the button itself by modifying `sender`

```
@IBAction func buttonWasPressed(_ sender: UIButton) {
    // add your own code here!
    sender.setTitle("Button was tapped!”,for: .normal)
}
```

Avoid adding too much to View Controllers!

With more complicated UI's, you'll end up having lots of outlets, actions, and view customization code

Instead of sticking this all in your View Controller, subclass views as well (i.e. make your own CustomButton rather than doing all your UI work in your view controllers)

TilesViewController.swift — Edited

TilesViewController.swift 〉 M collectionView(collectionView:layout:sizeForItemAtIndexPath:)

```
108
109     func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
            NSIndexPath) -> UICollectionViewCell {
110         setTempData()
111         let cell = collectionView.dequeueReusableCellWithReuseIdentifier("basic1", forIndexPath:
                indexPath) as! BasicCollectionCell
112         cell.imageView?.image = tempData.getImage(indexPath.row, folder: folder)
113         cell.textLabel.text = tempData.getLabel(indexPath.row, folder: folder)
114         print(cell.textLabel.text)
115         cell.contentView.layer.cornerRadius = 5
116         cell.contentView.layer.masksToBounds = true
117         cell.textLabel.layer.cornerRadius = 5
118         cell.textLabel.layer.masksToBounds = true
119
120         return cell
121     }
122
123     func collectionView(collectionView: UICollectionView,
124         layout collectionViewLayout: UICollectionViewLayout,
125         sizeForItemAtIndexPath indexPath: NSIndexPath) -> CGSize {
126             let width = mainCollection.frame.width
127             var itemsPerRow = 2
128             var addedWidth = CGFloat(0.0)
129             var leftover: CGFloat
130             while (addedWidth == 0) {
131                 leftover = width - (CGFloat(itemsPerRow * 100) + CGFloat(itemsPerRow + 1) *
                        spaceBetweenCells)
132                 if leftover < CGFloat(100) + spaceBetweenCells {
133                     addedWidth = leftover / CGFloat(itemsPerRow)
134                 } else {
135                     itemsPerRow += 1
136                 }
137             }
138             let finalWidth = 100 + addedWidth
139             let finalHeight = finalWidth * 1.2
140             let size = CGSize(width: finalWidth, height: finalHeight)
141             sizeOfCells = size
142             return size
143     }
144
145     func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
            NSIndexPath) {
146         setTempData()
147         tempData.speakAtIndex(indexPath.row, folder: folder)
148         collectionView.deselectItemAtIndexPath(indexPath, animated: true)
149     }
150
151
152     func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
            UICollectionViewLayout, minimumLineSpacingForSectionAtIndex section: Int) -> CGFloat {
153         let sectionInset = UIEdgeInsets(top: spaceBetweenCells, left: spaceBetweenCells, bottom:
                spaceBetweenCells, right: spaceBetweenCells)
154         mainCollection.contentInset = sectionInset
155         return spaceBetweenCells
156     }
157
158     func collectionView(collectionView: UICollectionView,
159         layout collectionViewLayout: UICollectionViewLayout,
160         minimumInteritemSpacingForSectionAtIndex section: Int) -> CGFloat {
161             return spaceBetweenCells
162     }
163
164     //MARK: Segues
165
166     override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
167         if let identifier = segue.identifier {
168             switch identifier {
169                 case "MainToFilter":
170                     let destination = segue.destinationViewController as! FilterViewController
171                     destination.data = data
172                     destination.dataToFilter = dataToFilter
```

Avoid adding too much to View Controllers!

With more complicated UI's, you'll end up having lots of outlets, actions, and view customization code

Instead of sticking this all in your View Controller, subclass views as well (i.e. make your own CustomButton rather than doing all your UI work in your view controllers)

Demo

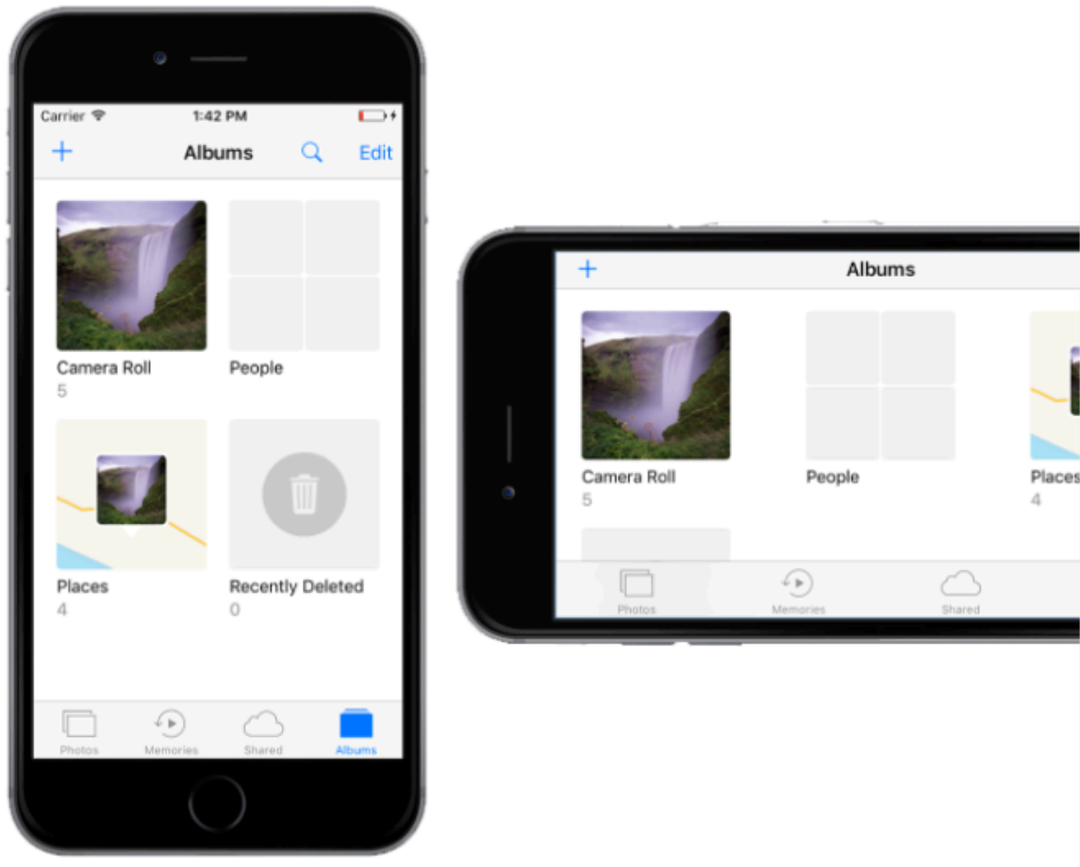
Views, View Controllers, Outlets, Actions

Check-in

Autolayout

Adaptive UI

- Display your app on **different screen sizes**
- Optimize for **different resolutions**
- Internationalize your UI (**Localization**)
- Resize/layout elements for **device rotations**

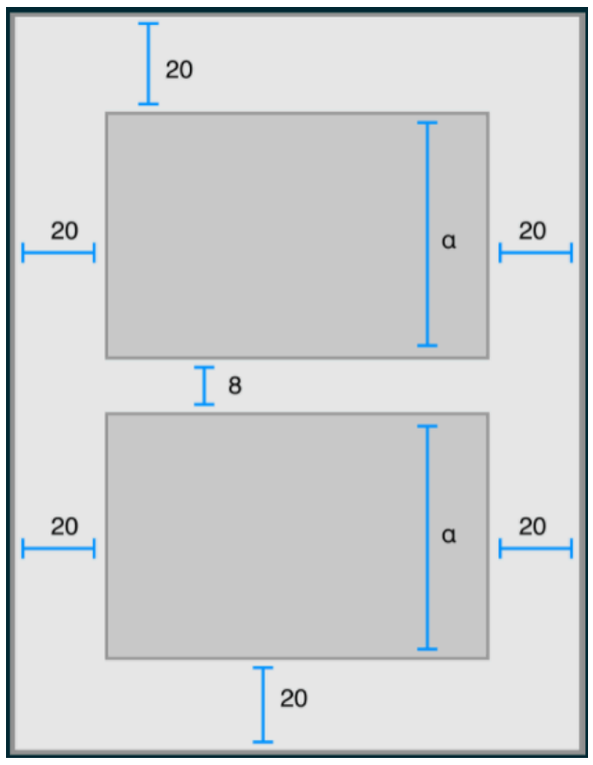


What is Auto Layout

- Constraint based, descriptive layout system
- Creating an adaptive interface that responds to changes in screen size and device orientation

What is a Constraint

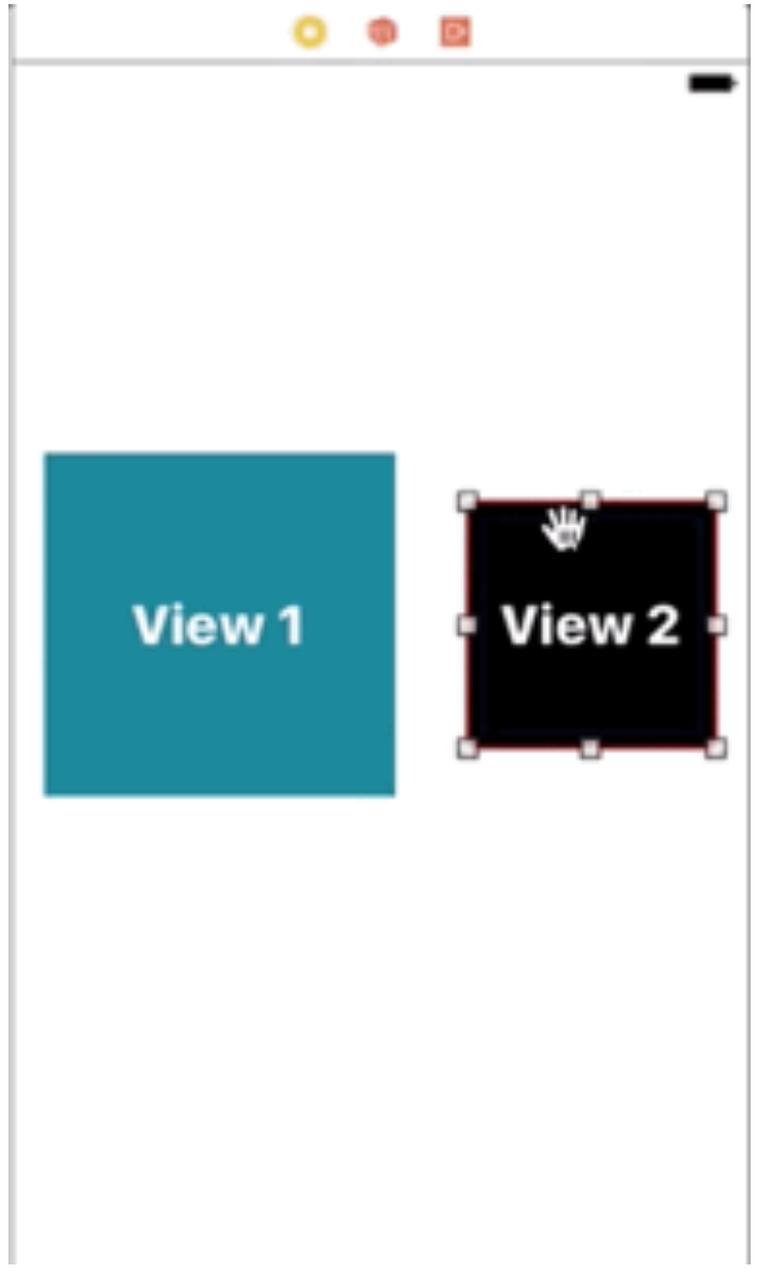
- Linear equations that relate different objects parts with one another.



Creating Constraints

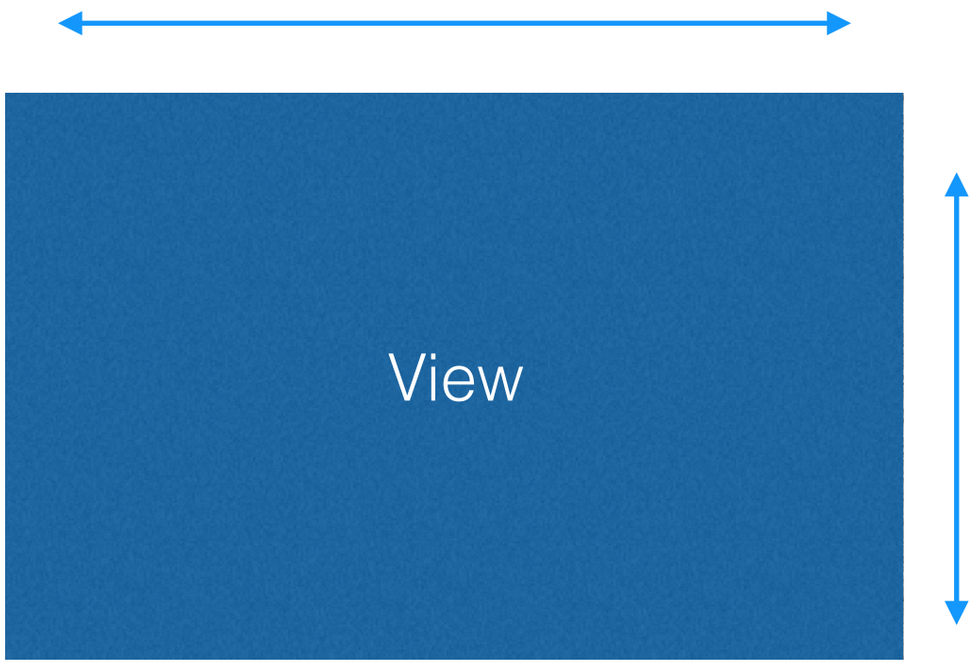
- Control + Drag from view to other view

- In Code:
  - **Layout anchors:**
    - NSLayoutAnchor
    - NSLayoutConstraint




```
let constraint =
    view1.leadingAnchor.constraint(
        equalTo: view2.trailingAnchor,
        constant: 8)

constraint.isActive = true
```

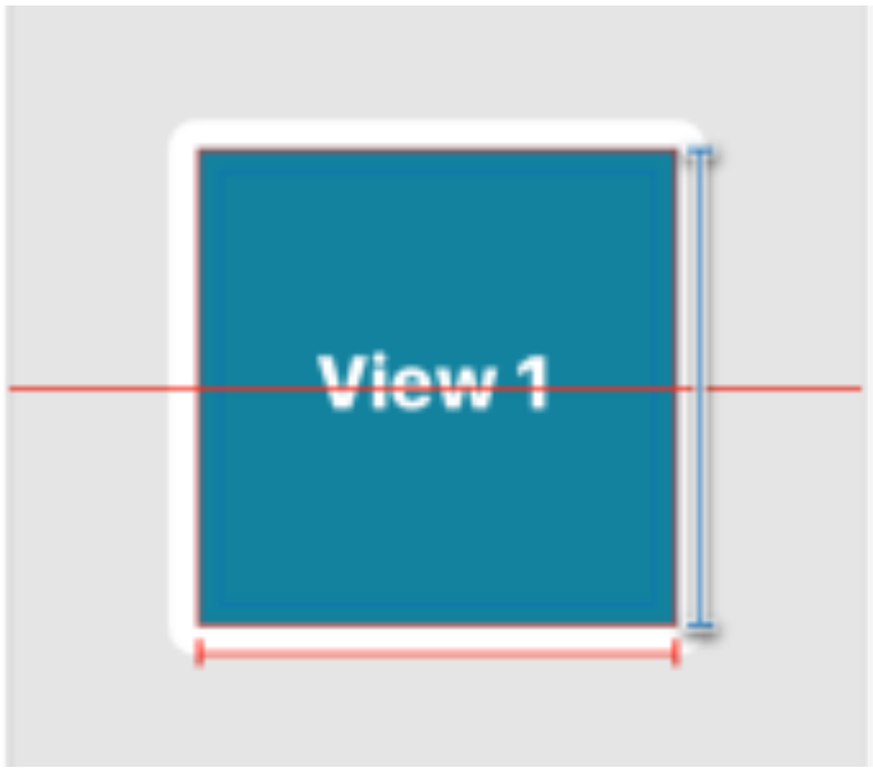
Creating Constraints : The Philosophy

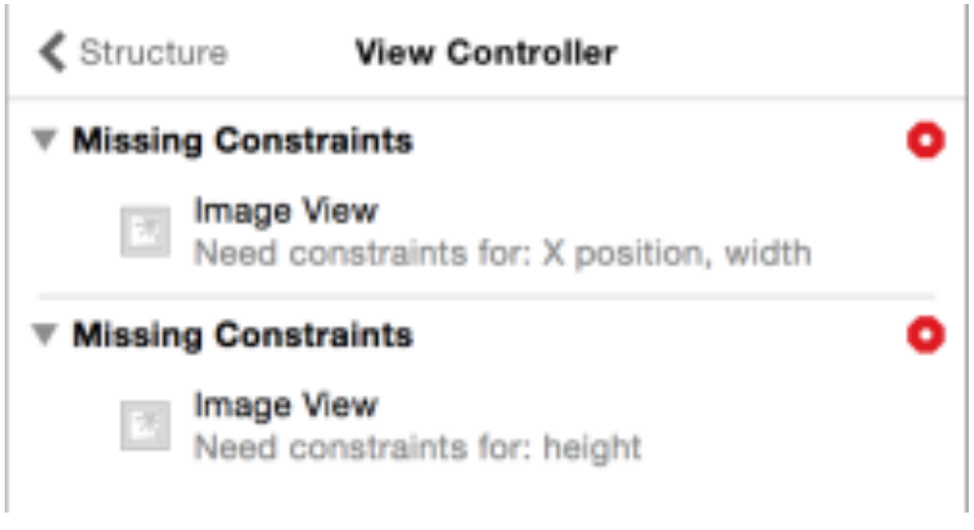


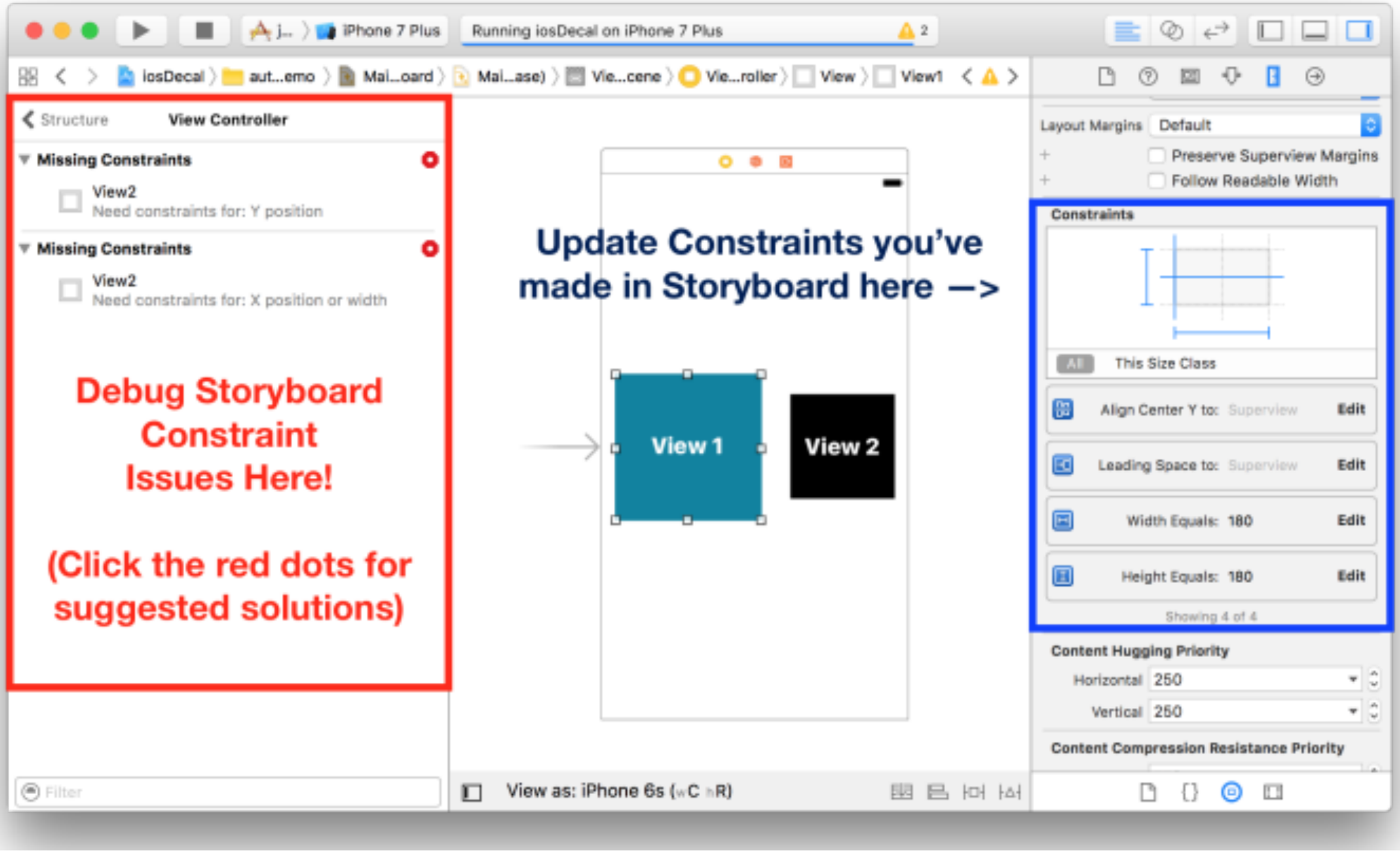
1. X Position
2. Y Position
3. Height
4. Width

Missing Constraints

- Xcode will let you know if you're missing a constraint
- Constraints will turn red
- Preview feature let's you test your work

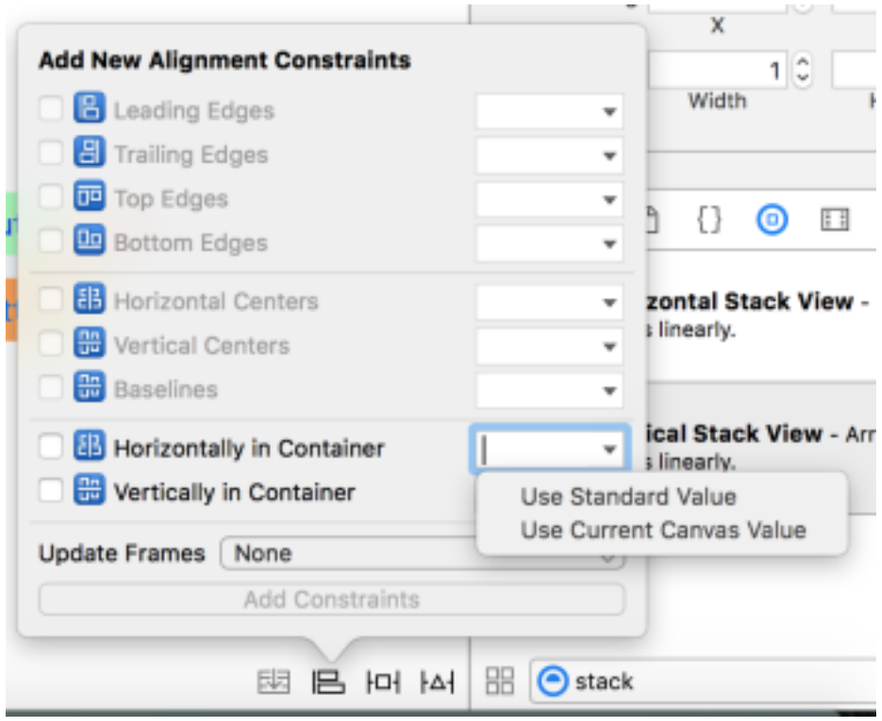




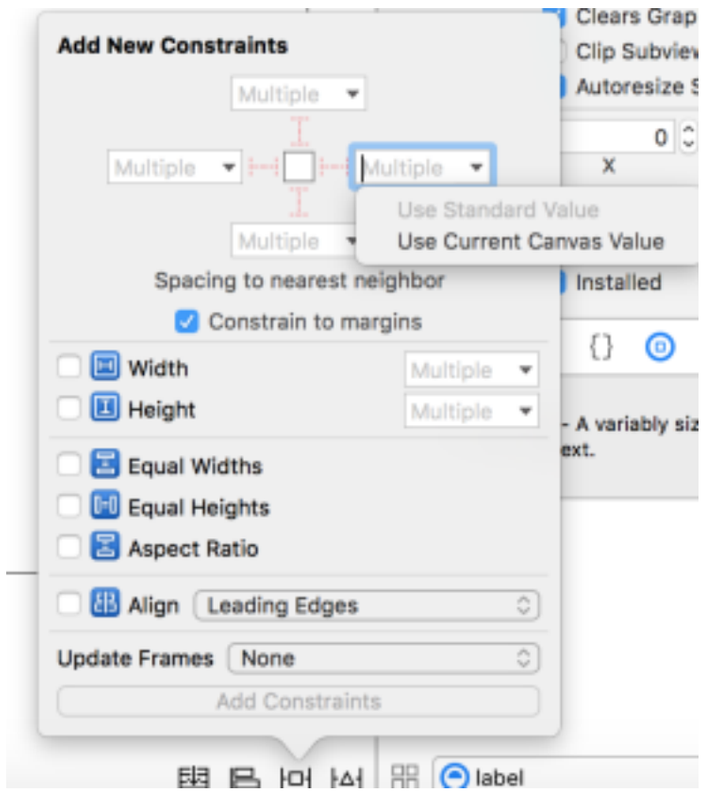

iPhone 7 Plus
Running iosDecal on iPhone 7 Plus
2
iosDecal
aut...emo
Mai...oard
Mai...ase)
Vie...cene
Vie...roller
View
View1
Structure
View Controller
Missing Constraints
View2
Need constraints for: Y position
Missing Constraints
View2
Need constraints for: X position or width
Debug Storyboard Constraint Issues Here!
(Click the red dots for suggested solutions)
Update Constraints you've made in Storyboard here —>
View 1
View 2
Filter
View as: iPhone 6s (wC hR)
Layout Margins
Default
Preserve Superview Margins
Follow Readable Width
Constraints
All
This Size Class
Align Center Y to: Superview
Edit
Leading Space to: Superview
Edit
Width Equals: 180
Edit
Height Equals: 180
Edit
Showing 4 of 4
Content Hugging Priority
Horizontal
250
Vertical
250
Content Compression Resistance Priority

Types of Constraints

- Alignment
  - Align Objects with each other



- Pin
  - Adds space to nearest neighbor (Can be a superview or itself)

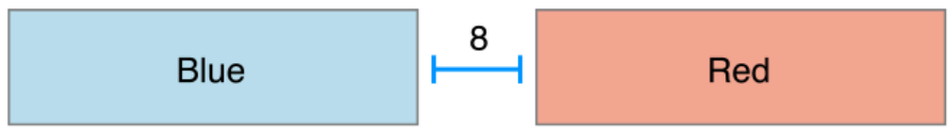


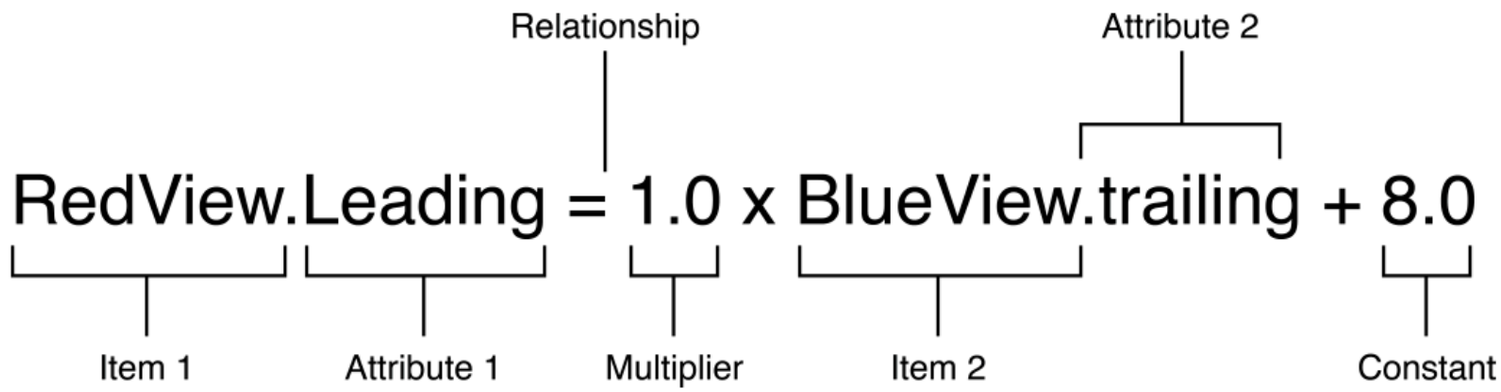
List of Constraint Types

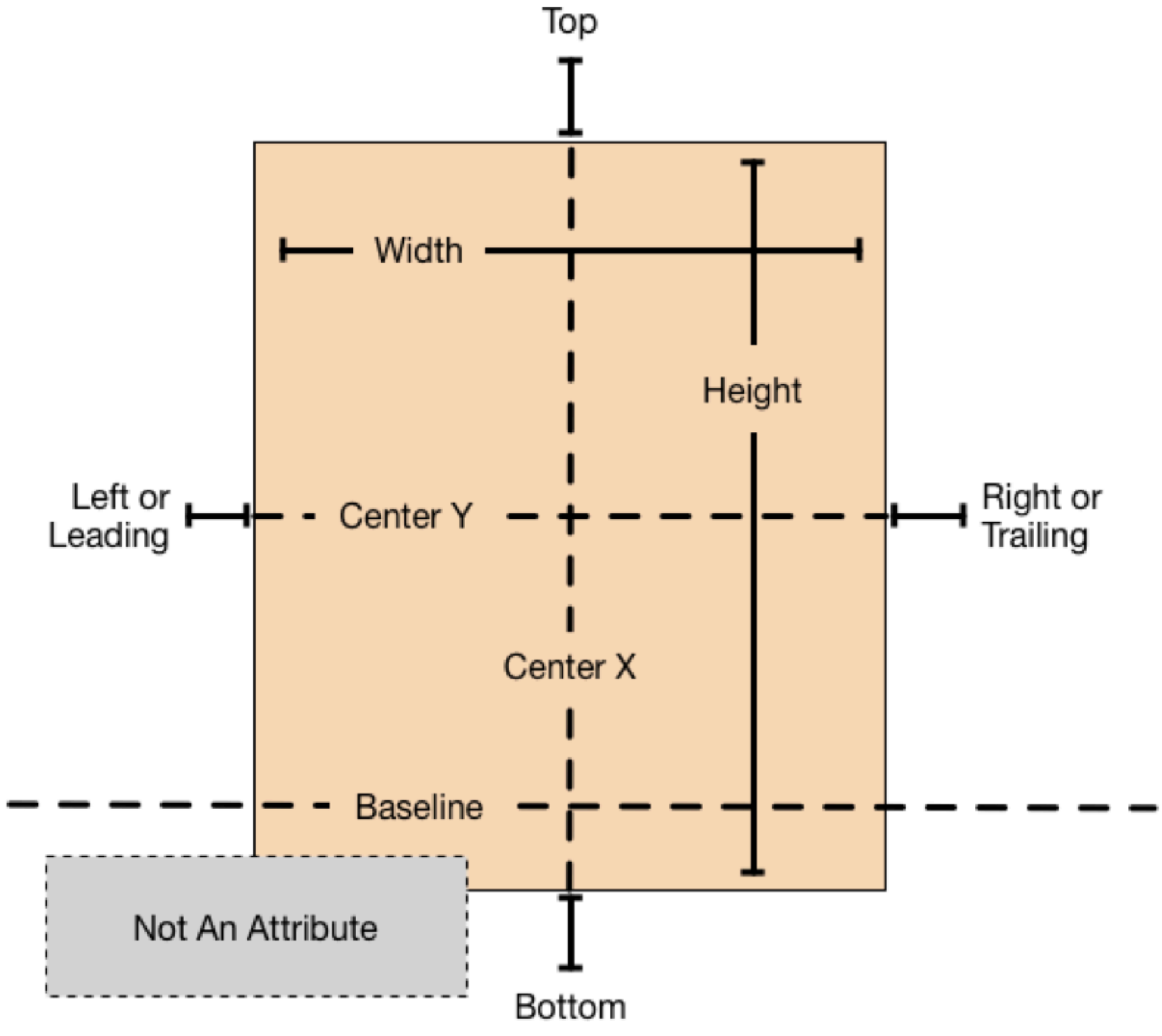
- **Height -** Height of View
- **Width -** Width of View
- **Top -** Vertical Spacing to Top View
- **Bottom -** Vertical Spacing to Bottom View
- **Baseline -** Align Baseline
- **Leading -** Spacing to Left View
- **Trailing -** Spacing to Right View
- **Center X -** Center Align Horizontally
- **Center Y -** Center Align Vertically

Formal Constraint Properties

- Item 1
- Attribute 1
- Relationship
- Multiplier
- Item 2
- Attribute 2
- Constant






Top
Width
Height
Left or
Leading
Center Y
Right or
Trailing
Center X
Baseline
Not An Attribute
Bottom

Demo

Lab 1 : Xcode Tutorial

Due **Tonight** at 11:59pm if you did not check off during lab

Next Lab : Auto Layout
Confused on AutoLayout? More Info Here
No Lecture next week (will have lab)