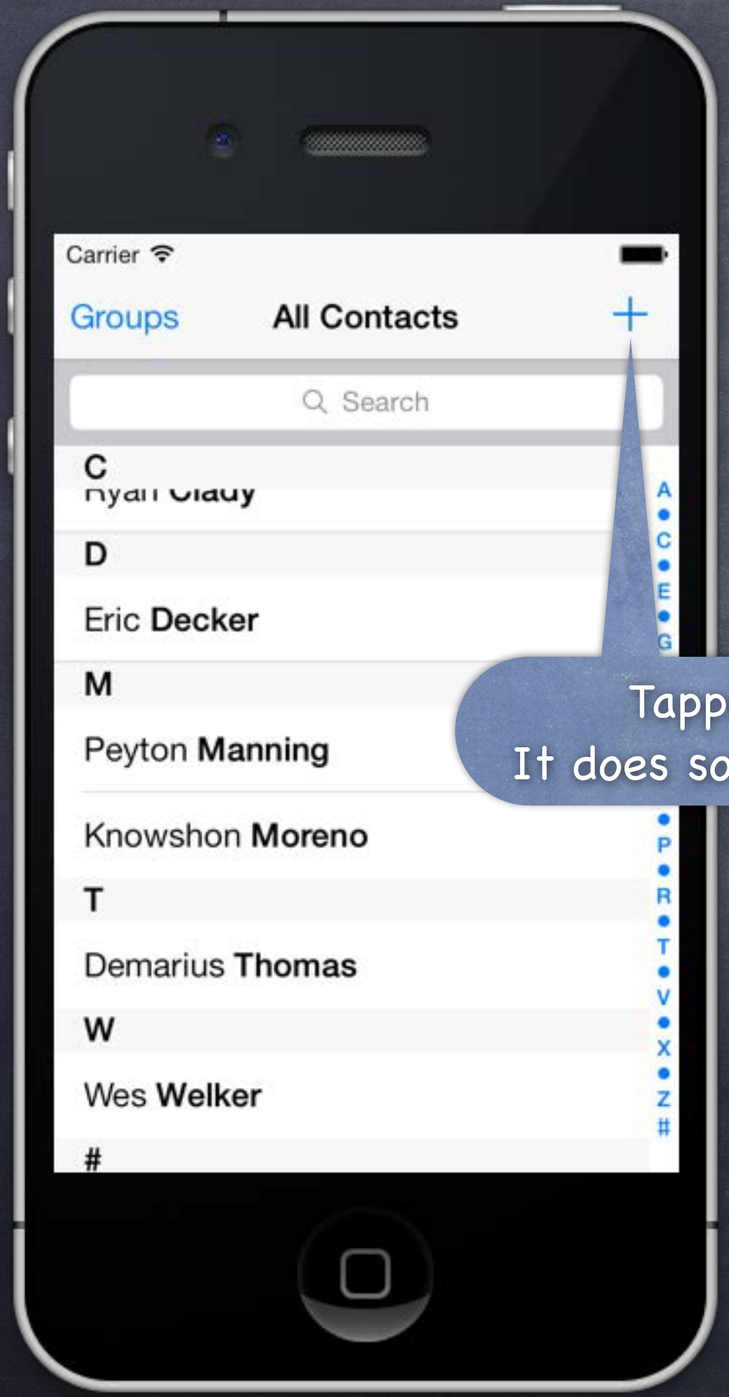
Stanford CS193p

Developing Applications for iOS
Spring 2016

Today

- Segues
  - Modal
  - Unwind
  - Popover
  - Embed
- Where am I?
  - Core Location
  - MapKit
- Trax Demo (time permitting)
  - Showing a Map
  - Putting waypoints on the Map
  - Segueing from a Map

Modal View Controllers

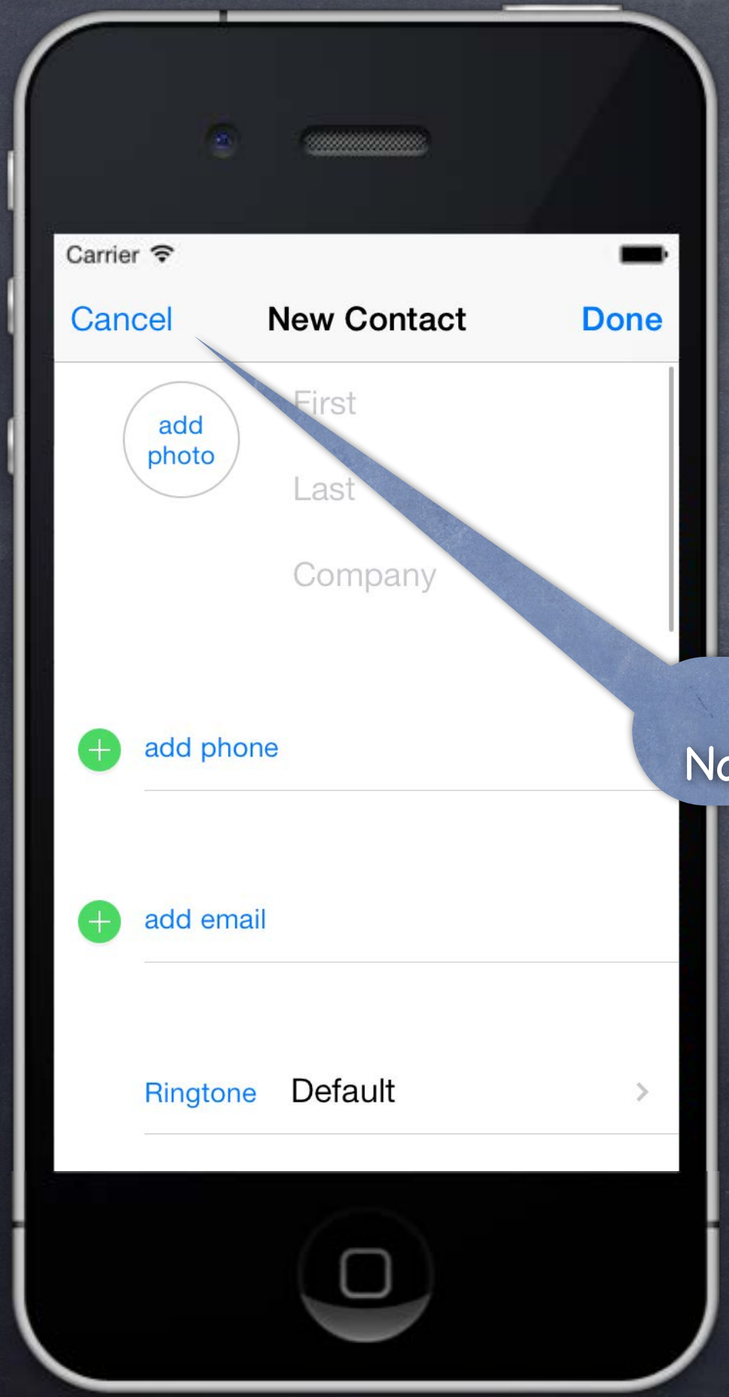


- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

Tapping here adds a new contact.
It does so by taking over the entire screen.

Modal View Controllers

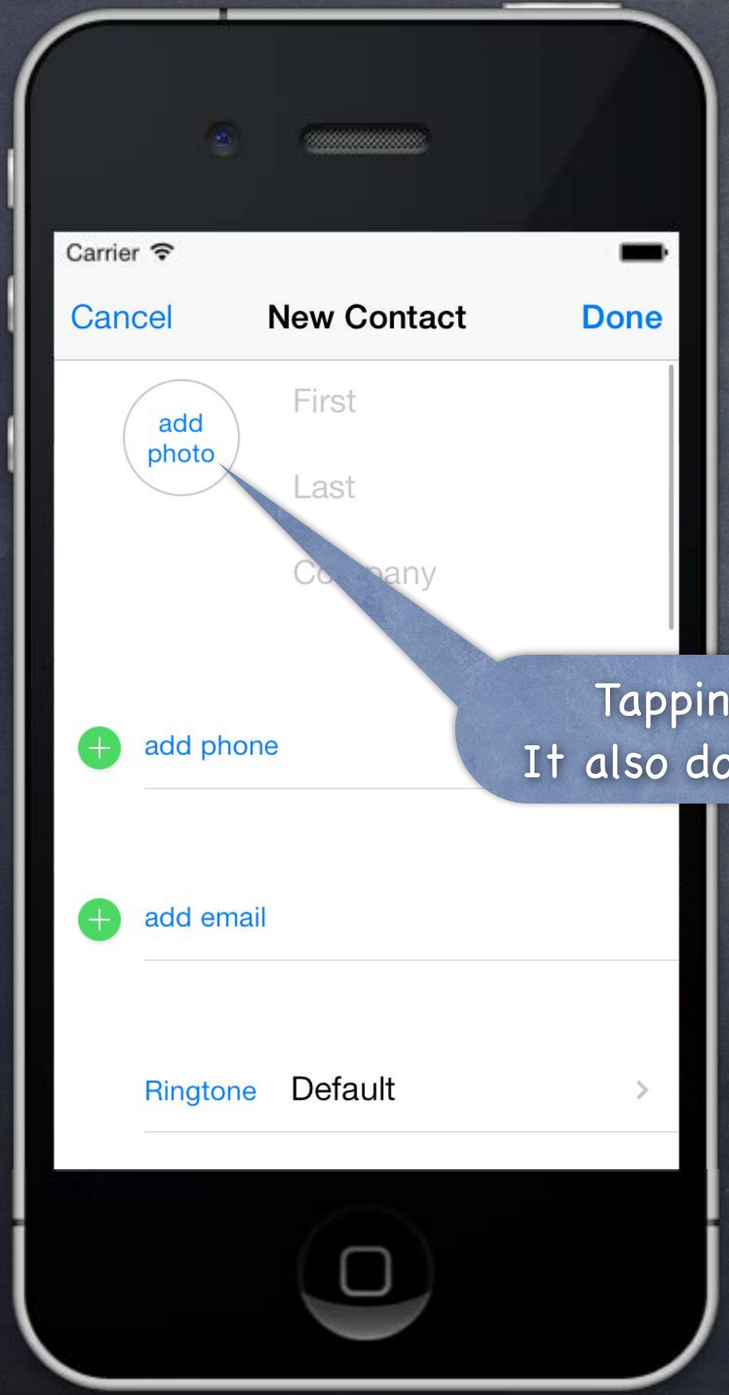
- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



This is not a push.
Notice, no back button (only Cancel).

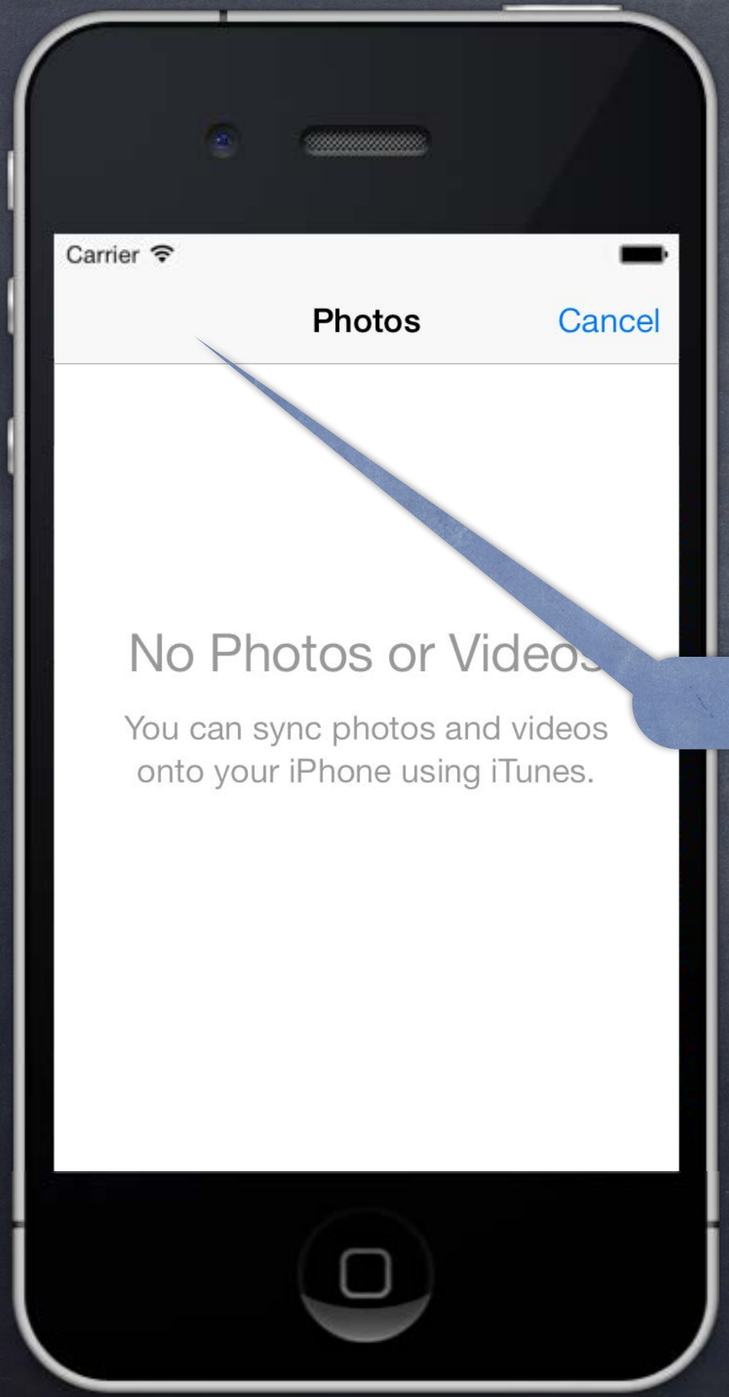
Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Tapping here adds a photo to this contact.
It also does so by taking over the entire screen.

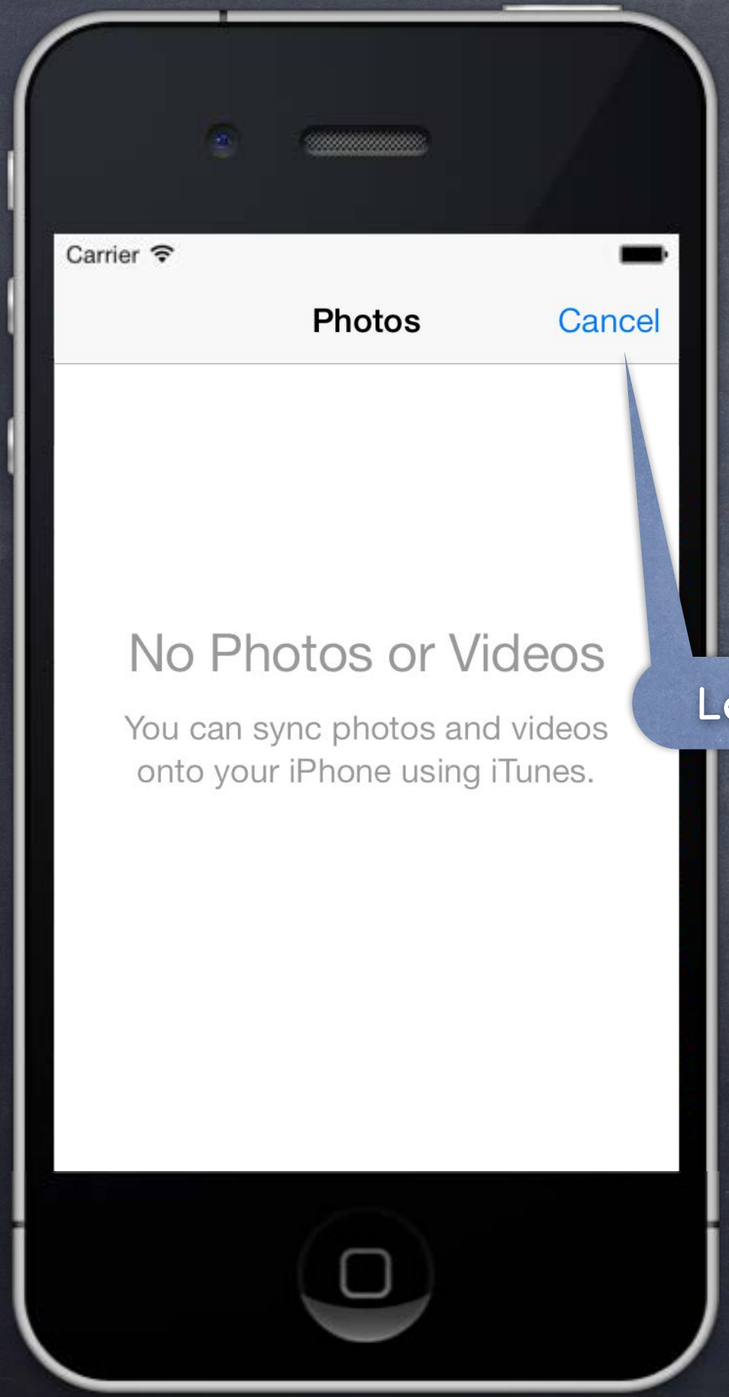
Modal View Controllers



- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

Again, no back button.

Modal View Controllers

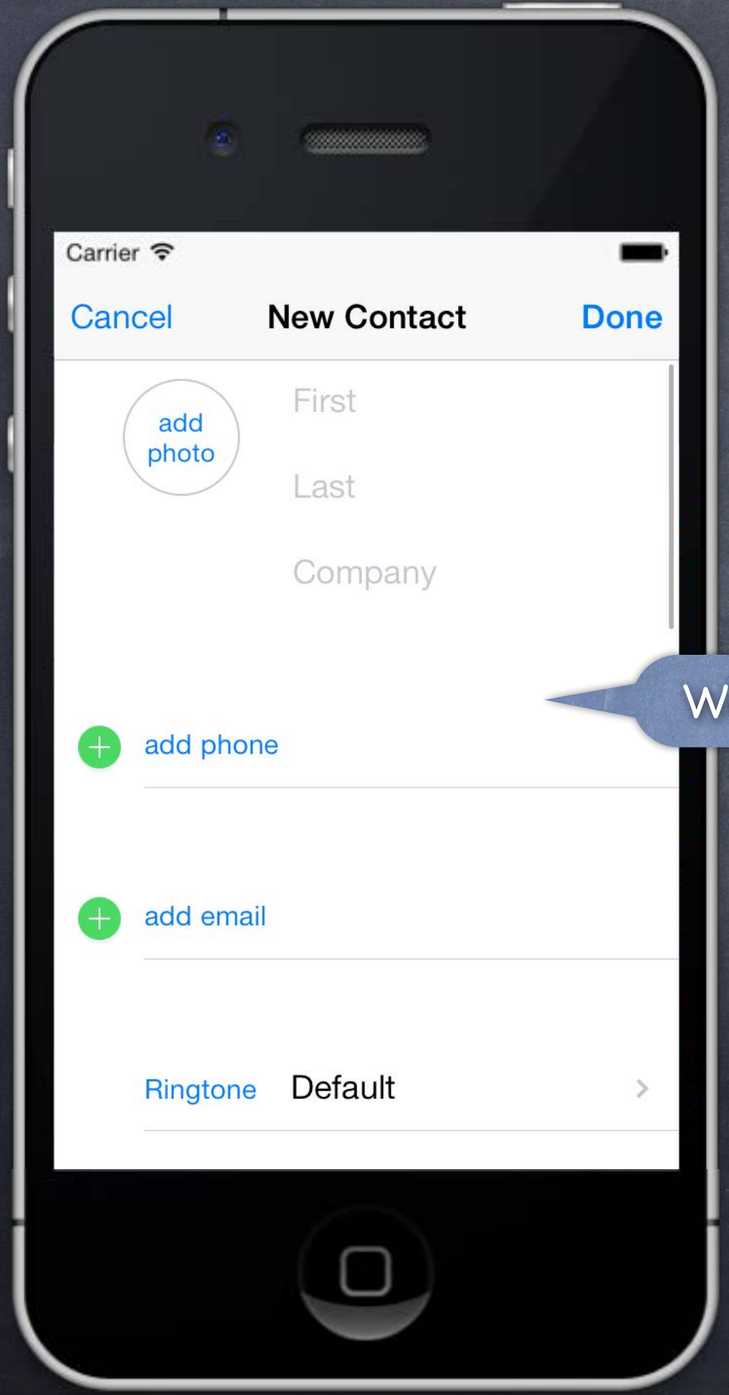


- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

Let's Cancel and see what happens.

Modal View Controllers

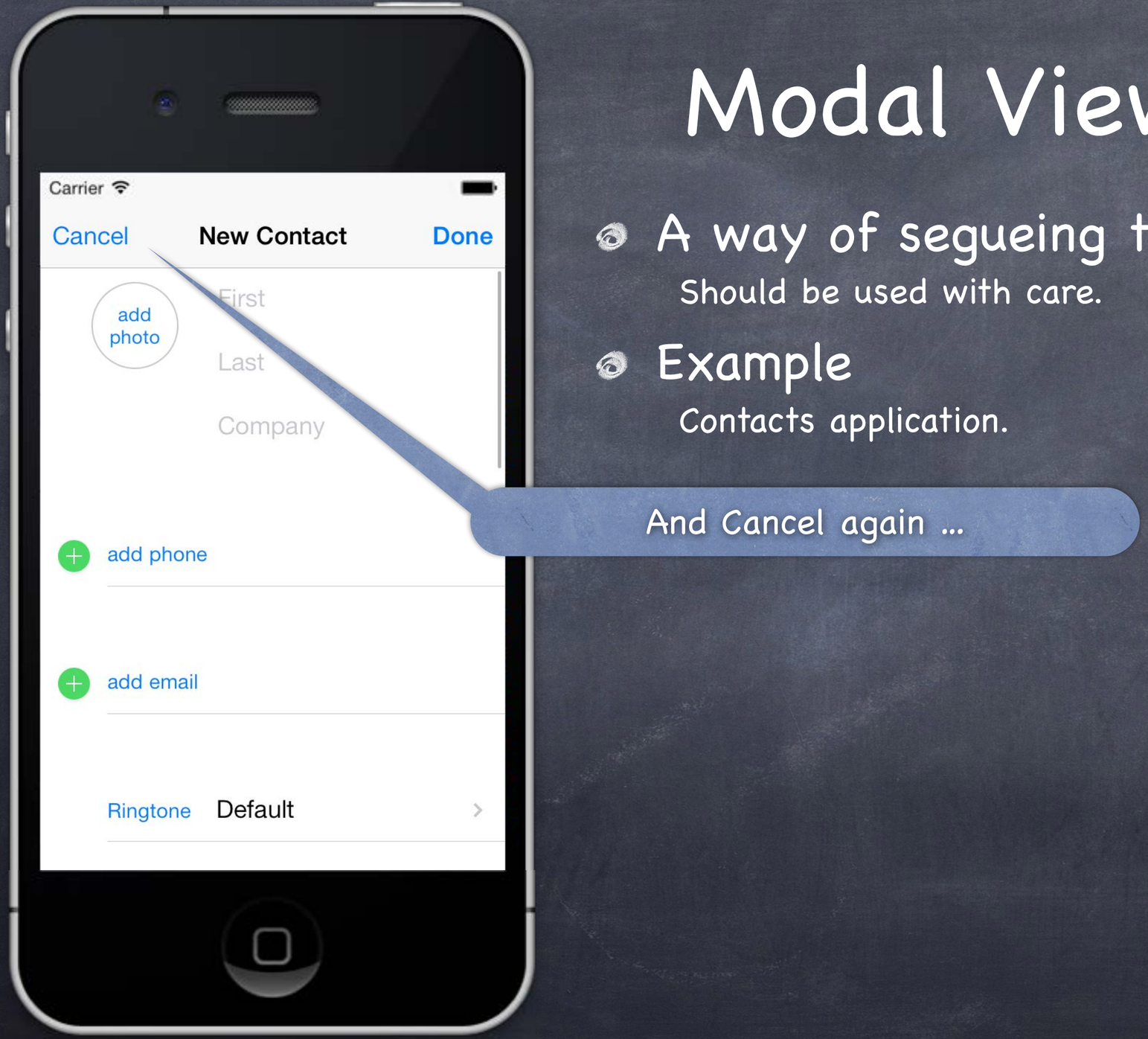
- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



We're back to the last Modal View Controller.

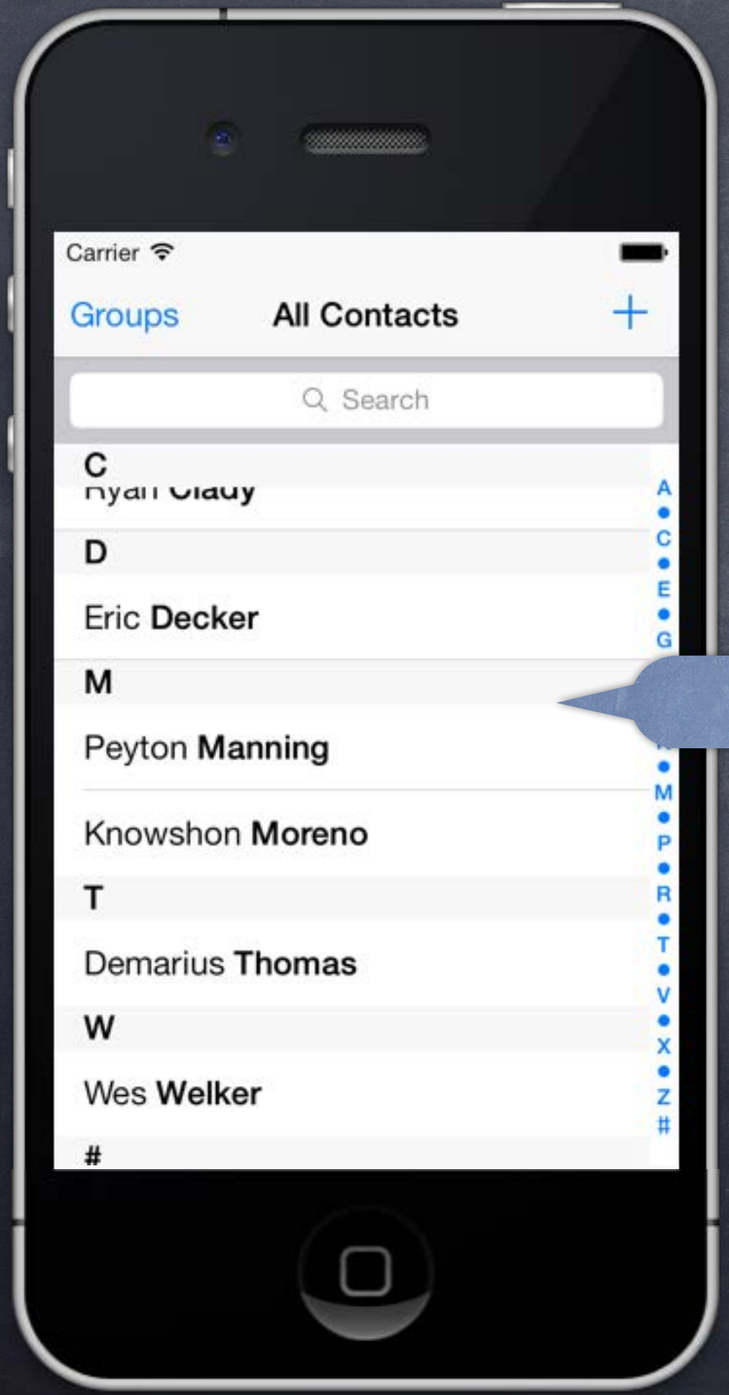
Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Back to where we started.

Modal View Controllers

- Considerations
  - The view controller we segue to using a Modal segue will take over the entire screen
  - This can be rather disconcerting to the user, so use this carefully
- How do we set a Modal segue up?
  - Just ctrl-drag from a button to another View Controller & pick segue type "Modal"
  - Inspect the segue to set the style of presentation

  If you need to present a Modal VC not from a button, use a manual segue …

  ```
  func performSegueWithIdentifier(String, sender: AnyObject?)
  ```

  … or, if you have the view controller itself (e.g. Alerts or from `instantiateViewController`…) …

  ```
  func presentViewController(UIViewController, animated: Bool, completion: () -> Void)
  ```

  In horizontally regular environments, `modalPresentationStyle` will determine how it appears …

  `.FullScreen`, `.OverFullScreen` (presenter left underneath), `.Popover`, `.FormSheet`, etc.

  In horizontally compact environments, this will adapt to always be full screen!

Modal View Controllers

- Preparing for a Modal segue

  You prepare for a Modal segue just like any other segue …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
    if segue.identifier == “GoToMyModalVC” {
        let vc = segue.destinationViewController as MyModalVC
        // set up the vc to run here
    }
}
```

- Hearing back from a Modally segue-to View Controller

  When the Modal View Controller is “done”, how does it communicate results back to presenter?

  If there’s nothing to be said, just dismiss the segued-to MVC (next slide).

  To communicate results, generally you would Unwind (though delegation possible too).

Modal View Controllers

- How to dismiss a view controller

  The presenting view controller is responsible for dismissing (not the presented).
  You do this by sending the presenting view controller this message …

```
func dismissViewControllerAnimated(Bool, completion: () -> Void)
```

  … which will dismiss whatever MVC it has presented (if any).

  If you send this to a presented view controller, for convenience, it will forward to its presenter
  (unless it itself has presented an MVC, in which case it will dismiss that MVC).
  But to reduce confusion in your code, only send `dismiss` to the presenting controller.

  Unwind Segues (coming up soon) automatically dismiss (you needn't call the above method).

Modal View Controllers

- How is the modal view controller animated onto the screen?

  Depends on this property in the view controller that is being presented …

```
var modalTransitionStyle: UIModalTransitionStyle
.CoverVertical   // slides the presented modal VC up from bottom of screen (the default)
.FlipHorizontal  // flips the presenting view controller over to show the presented modal VC
.CrossDissolve   // presenting VC fades out as the presented VC fades in
.PartialCurl     // only if presenting VC is full screen (& no more modal presentations coming)
```

  You can also set this in the storyboard by inspecting the modal segue.

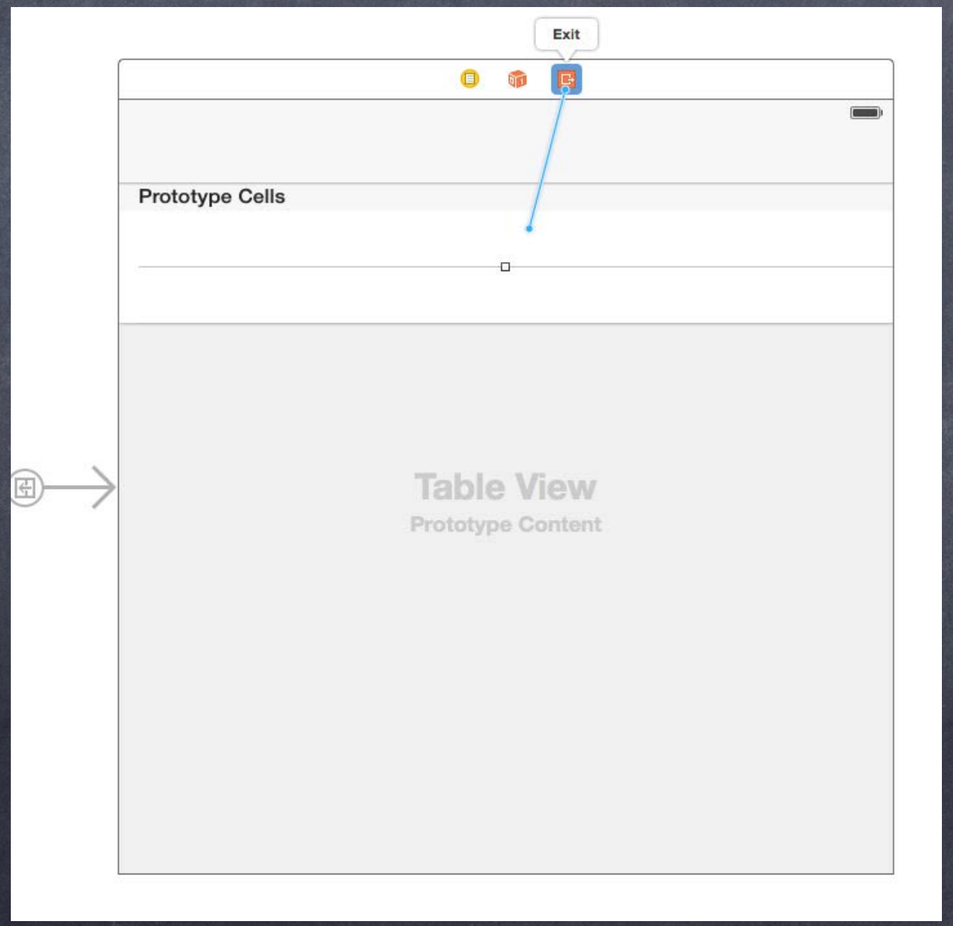
Unwind Segue

- The only segue that does NOT create a new MVC
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- What's it good for?
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter

Unwind Segue

- How does it work?

  Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC

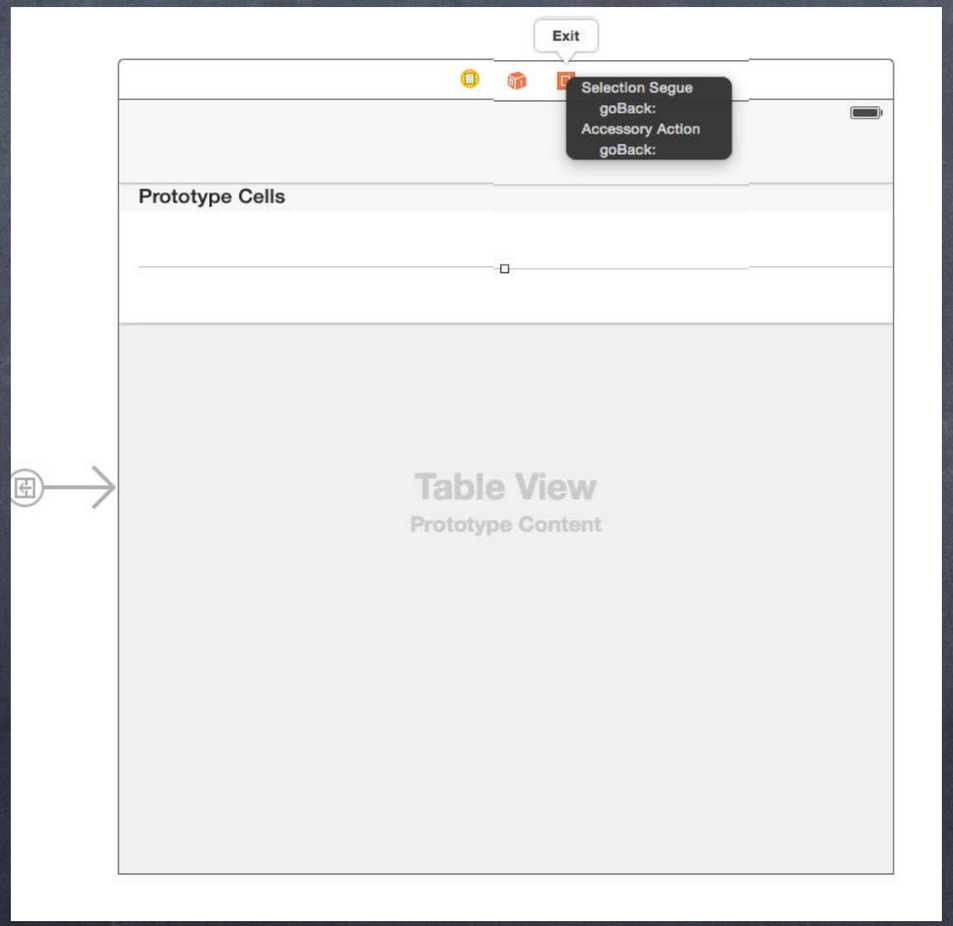


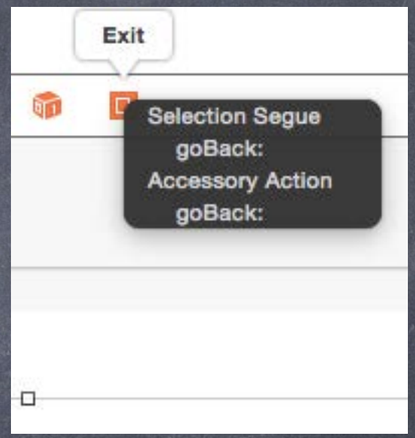
Unwind Segue

- How does it work?

  Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC

  Then you can choose a special `@IBAction` method you've created in another MVC



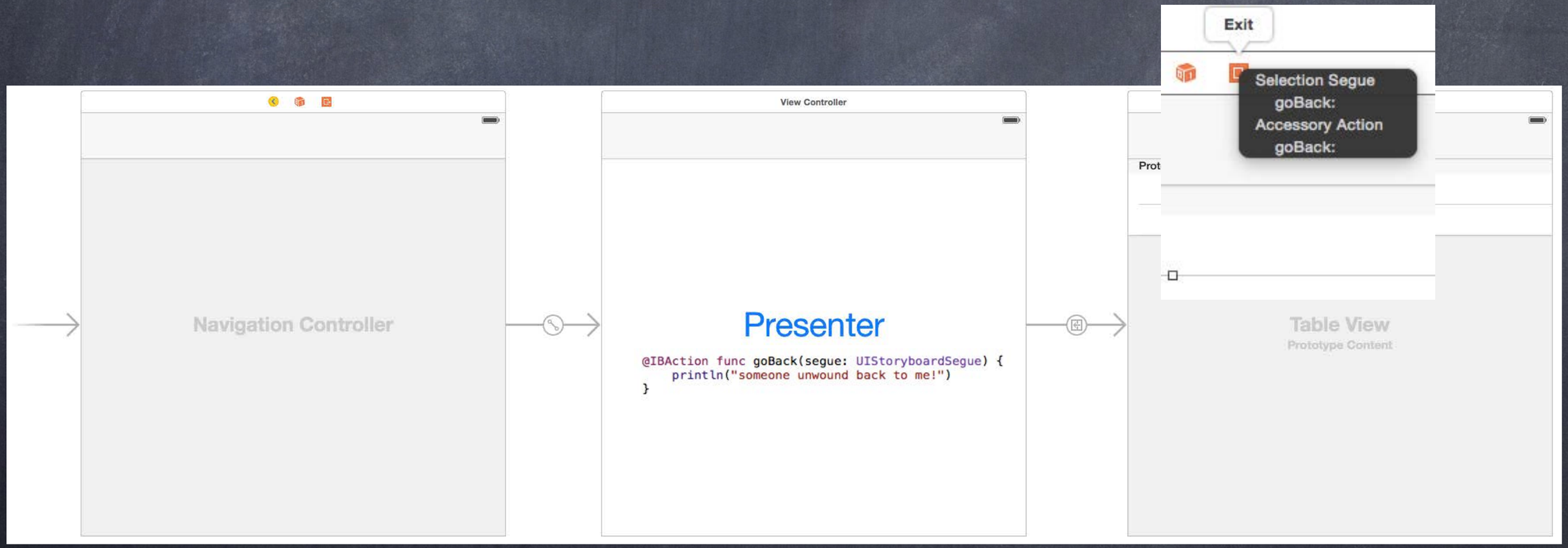


Unwind Segue

- How does it work?

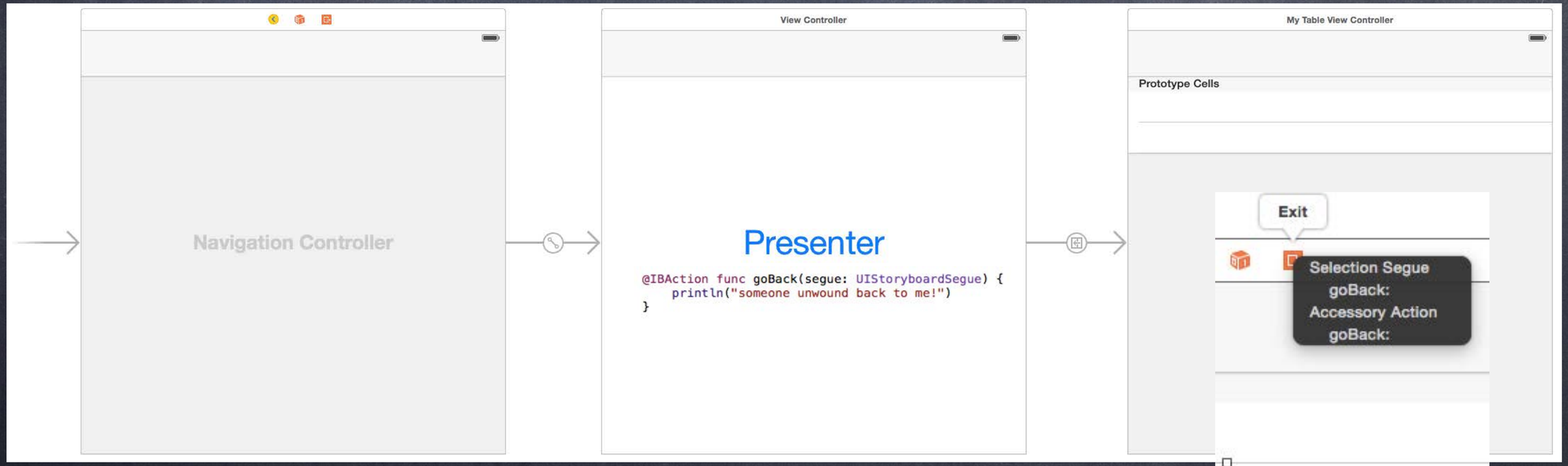
  Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC

  Then you can choose a special `@IBAction` method you've created in another MVC



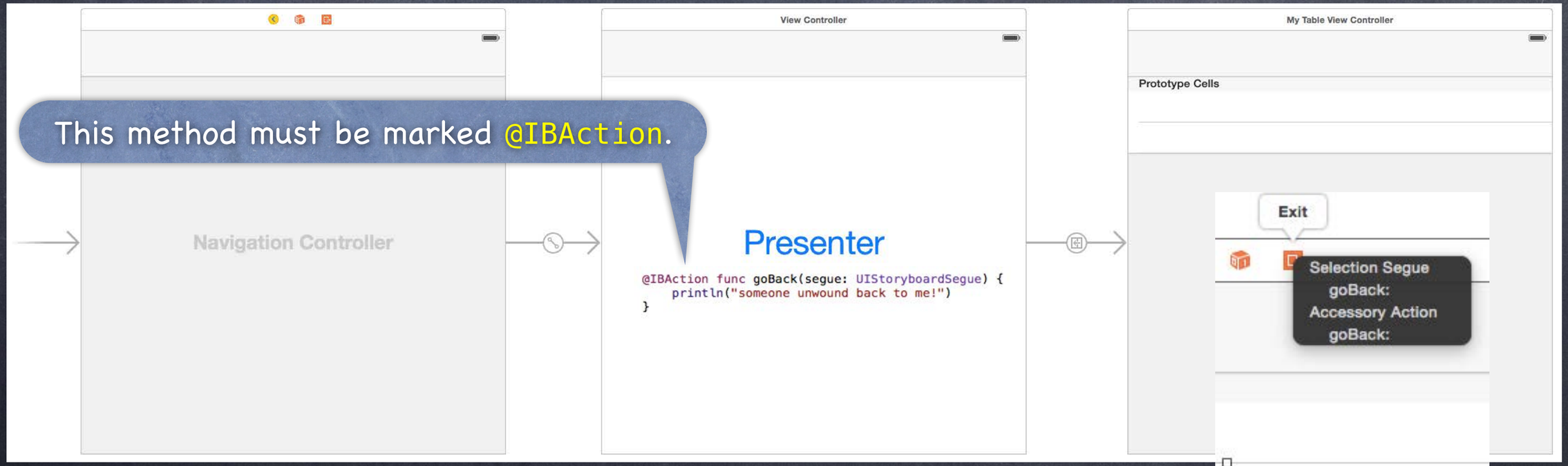
Unwind Segue

- How does it work?
  - Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC
  - Then you can choose a special `@IBAction` method you've created in another MVC
  - This means "segue by exiting me and finding a presenter who implements that method"
  - If no presenter (directly or indirectly) implements that method, the segue will not happen



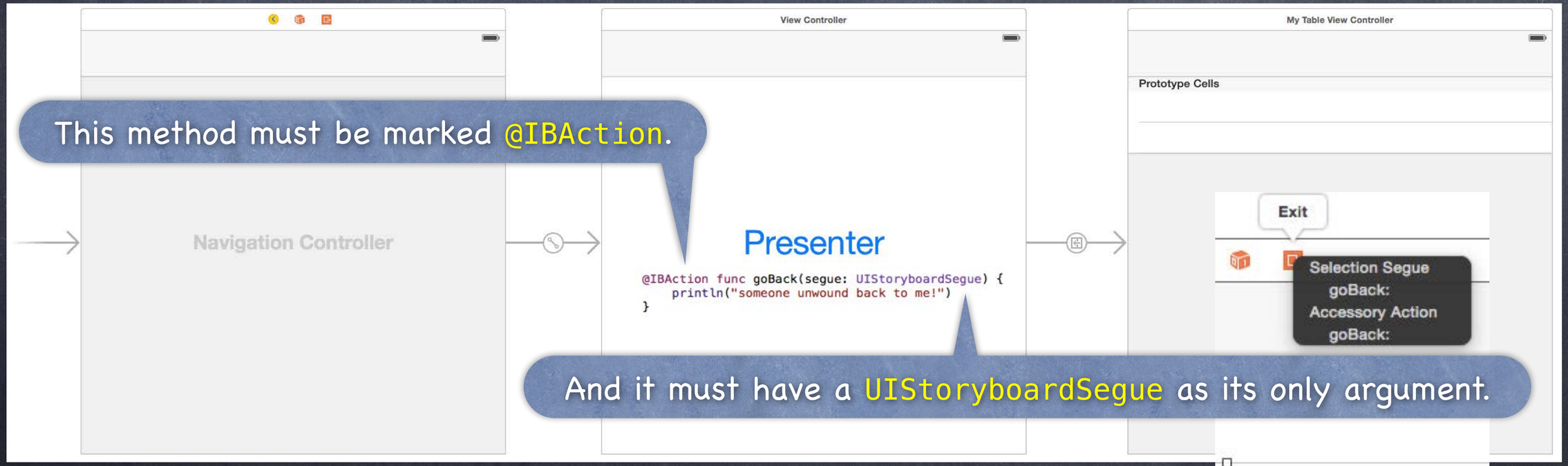
Unwind Segue

- How does it work?
  - Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC
  - Then you can choose a special `@IBAction` method you've created in another MVC
  - This means "segue by exiting me and finding a presenter who implements that method"
  - If no presenter (directly or indirectly) implements that method, the segue will not happen



Unwind Segue

- How does it work?
  - Instead of ctrl-dragging to another MVC, you ctrl-drag to the "Exit" button in the same MVC
  - Then you can choose a special `@IBAction` method you've created in another MVC
  - This means "segue by exiting me and finding a presenter who implements that method"
  - If no presenter (directly or indirectly) implements that method, the segue will not happen

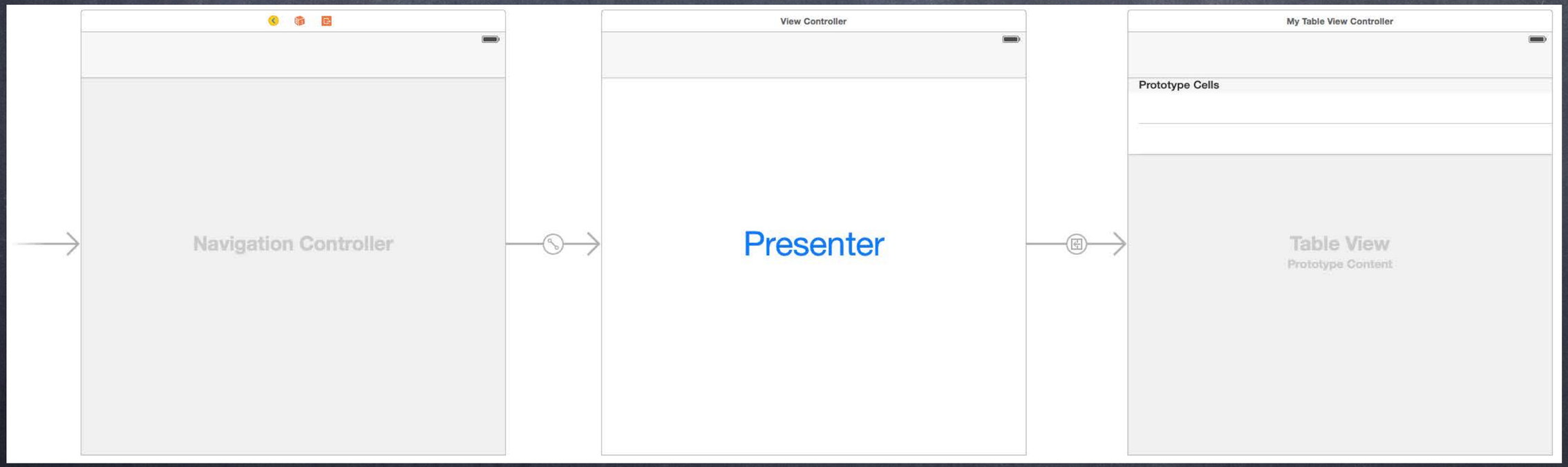


Unwind Segue

- How does it work?

  If the `@IBAction` can be found, you (i.e. the presented MVC) will get to `prepareForSegue` as normal

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
{
    if segue.identifier == "GoBack Unwind Segue" {
        if let unwoundToMVC = segue.destinationViewController as? MyPresentingViewController {
            // prepare unwoundToMVC
        }
    }
}
```

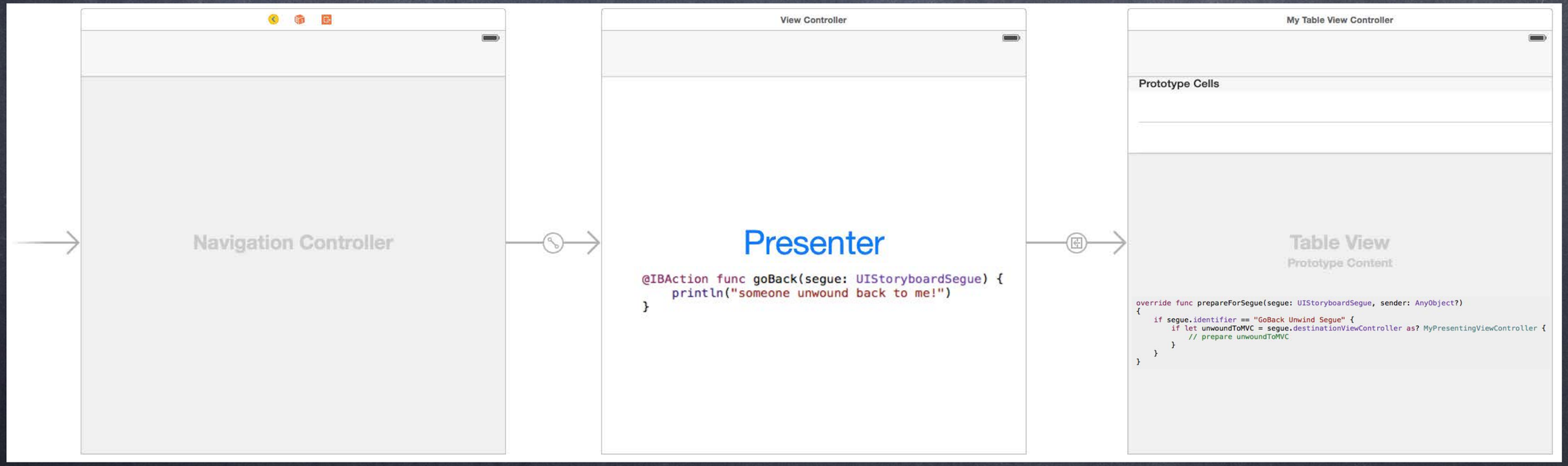


Unwind Segue

- How does it work?

  If the `@IBAction` can be found, you (i.e. the presented MVC) will get to `prepareForSegue` as normal

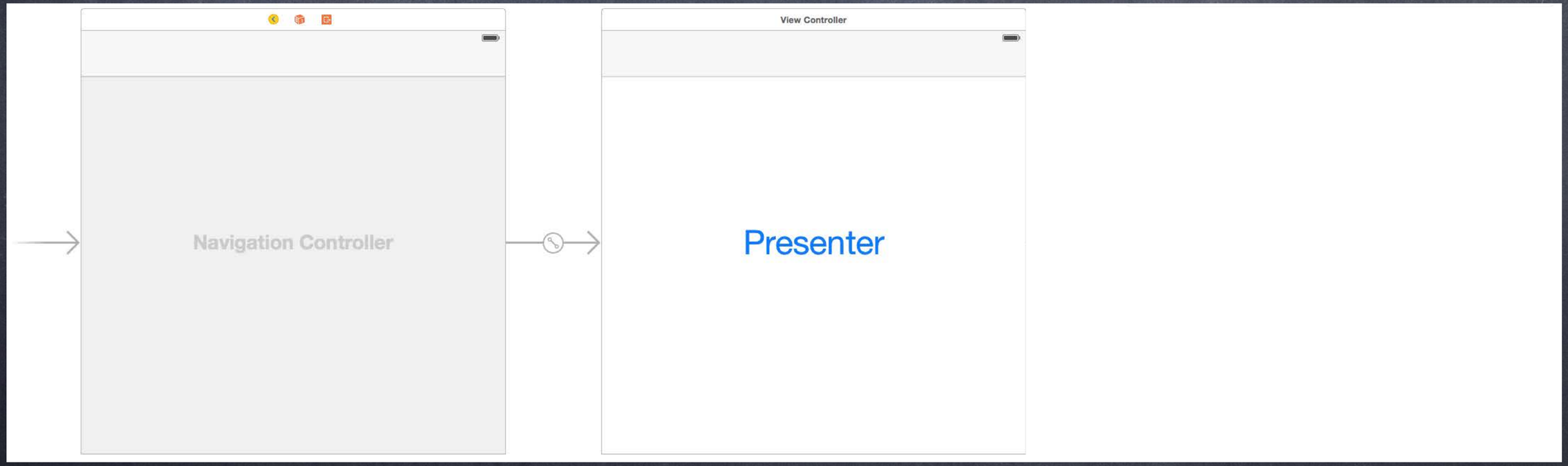
  Then the special `@IBAction` will be called in the other MVC and that MVC will be shown on screen



Unwind Segue

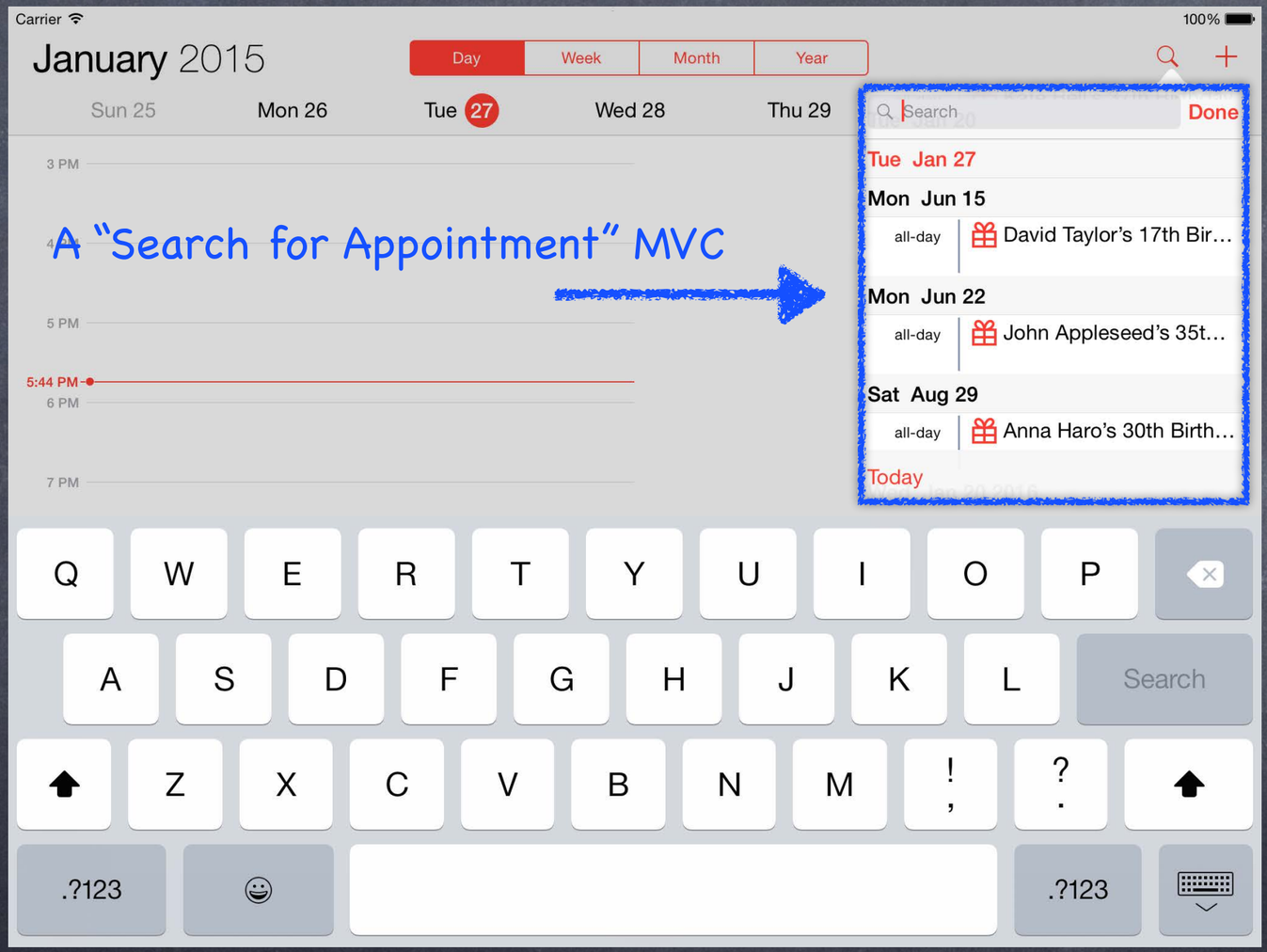
- How does it work?

  If the `@IBAction` can be found, you (i.e. the presented MVC) will get to `prepareForSegue` as normal
  Then the special `@IBAction` will be called in the other MVC and that MVC will be shown on screen
  You will be dismissed in the process (i.e. you'll be "unpresented" and thrown away)



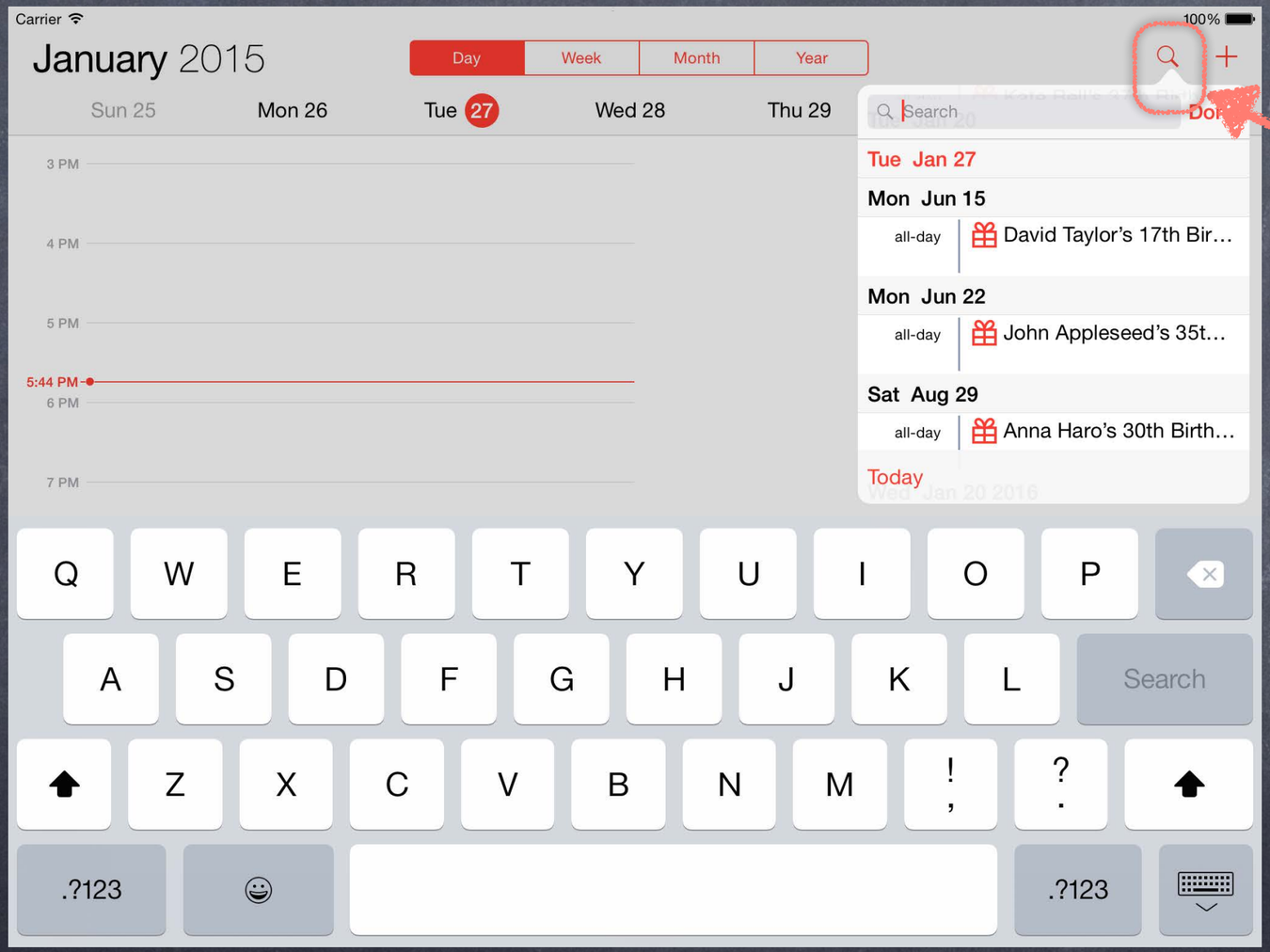
Popover

- Popovers pop an entire MVC over the rest of the screen



Popover

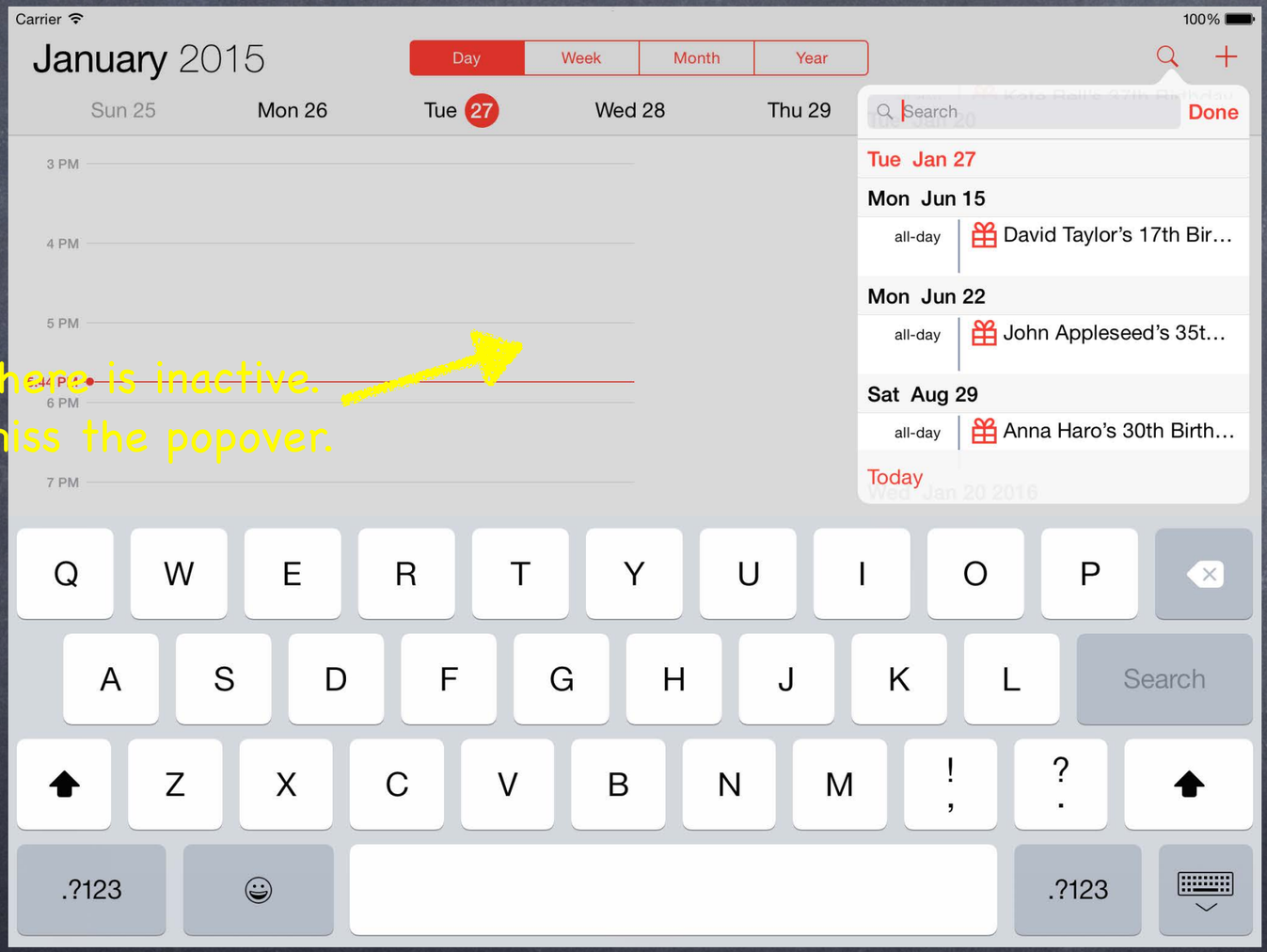
- Popovers pop an entire MVC over the rest of the screen



Popover's arrow pointing to what caused it to appear

Popover

- Popovers pop an entire MVC over the rest of the screen



The grayed out area here is inactive.
Touching in it will dismiss the popover.

Popover

- Popovers are not quite the same as other segued-to MVCs
  - Tab Bar, Split View and Navigation Controllers are `UIViewControllers`, popovers are not.
- Seguing to a popover works is set up the same way though
  - You still ctrl-drag, you still have an identifier, you still get to prepare
- Things to note when preparing for a popover segue
  - All segues are managed via a `UIPresentationController` (but we're not going to cover that)
  - But we are going to talk about a popover's `UIPopoverPresentationController`
  - It notes what caused the popover to appear (a bar button item or just a rectangle in a view)
  - You can also control what direction the popover's arrow is allowed to point
  - Or you can control how a popover adapts to different sizes classes (e.g. iPad vs iPhone)

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Do Something in a Popover Segue”:
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

One thing that is different is that we are retrieving the popover's presentation controller

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "Do Something in a Popover Segue":
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

We can use it to set some properties that will control how the popover pops up

Popover Prepare

- Here's a prepareForSegue that prepares for a Popover segue

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Do Something in a Popover Segue”:
                if let vc = segue.destinationViewController as? MyController {
                    if let ppc = vc.popoverPresentationController {
                        ppc.permittedArrowDirections = UIPopoverArrowDirection.Any
                        ppc.delegate = self
                    }
                    // more preparation here
                }
            default: break
        }
    }
}
```

And we can control the presentation by setting ourself (the Controller) as the delegate

Popover Presentation Controller

- Adaptation to different size classes

  One very interesting thing is how a popover presentation can "adapt" to different size classes.
  When a popover is presenting itself in a horizontally compact environment (e.g. iPhone),
  there might not be enough room to show a popover window comfortably,
  so by default it "adapts" and shows the MVC in full screen modal instead.

  But the popover presentation controller's `delegate` can control this "adaptation" behavior.
  Either by preventing it entirely ...

```
func adaptivePresentationStyleForPresentationController(
        controller: UIPresentationController,
    traitCollection: UITraitCollection
) -> UIModalPresentationStyle {
    return UIModalPresentationStyle.None // don't adapt
    // the default in horizontally compact environments (iPhone) is .FullScreen
}
```

Popover Presentation Controller

- Adaptation to different size classes

  One very interesting thing is how a popover presentation can "adapt" to different size classes.
  When a popover is presenting itself in a horizontally compact environment (e.g. iPhone),
  there might not be enough room to show a popover window comfortably,
  so by default it "adapts" and shows the MVC in full screen modal instead.

  But the popover presentation controller's `delegate` can control this "adaptation" behavior.
  ... or by modifying the adaptation ...
  You can control the view controller that is used to present in the adapted environment
  Best example: wrapping a `UINavigationController` around the MVC that is presented

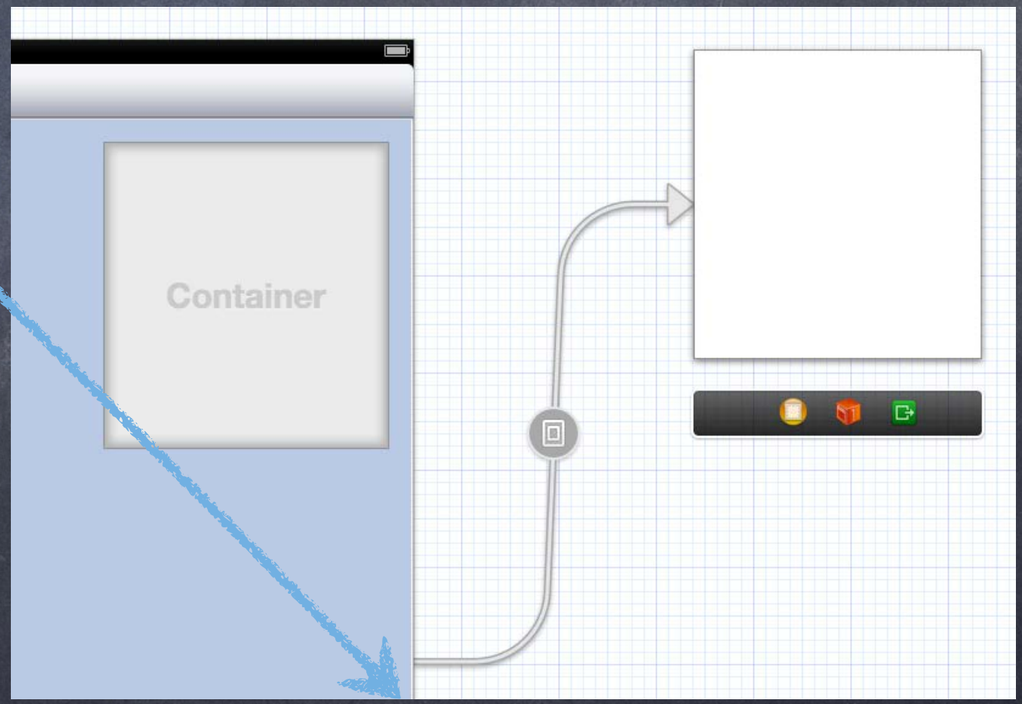
```
func presentationController(controller: UIPresentationController,
   viewControllerForAdaptivePresentationStyle: UIModalPresentationStyle)
    -> UIViewController?
{
    // return a UIViewController to use (e.g. wrap a Navigation Controller around your MVC)
}
```

Popover Size

- Important Popover Issue: Size
  - A popover will be made pretty large unless someone tells it otherwise.
  - The MVC being presented knows best what it's "preferred" size inside a popover would be.
  - It expresses that via this property in itself (i.e. in the Controller of the MVC being presented) ...
  - `var preferredContentSize: CGSize`
  - The MVC is not guaranteed to be that size, but the system will try its best.
  - You can set or override the var to always return an appropriate size.

Embed Segues

- Putting a VC's `self.view` in another VC's view hierarchy!
  - This can be a very powerful encapsulation technique.
- Xcode makes this easy
  - Drag out a Container View from the object palette into the scene you want to embed it in.
  - Automatically sets up an "Embed Segue" from container VC to the contained VC.
- Embed Segue
  - Works just like other segues.
  - `prepareForSegue(sender:)`, et. al.



Embed Segues

- Putting a VC's `self.view` in another VC's view hierarchy!
  - This can be a very powerful encapsulation technique.
- Xcode makes this easy
  - Drag out a Container View from the object palette into the scene you want to embed it in.
  - Automatically sets up an "Embed Segue" from container VC to the contained VC.
- Embed Segue
  - Works just like other segues.
  - `prepareForSegue(sender:)`, et. al.
- View Loading Timing
  - Don't forget, though, that just like other segued-to VCs,
    the embedded VC's outlets are not set at the time `prepareForSegue(sender:)` is called.

Core Location

- Framework for managing location and heading

  No user-interface.

- Basic object is `CLLocation`

  Properties: `coordinate`, `altitude`, `horizontal/verticalAccuracy`, `timestamp`, `speed`, `course`

- Where (approximately) is this location?

```
var coordinate: CLLocationCoordinate2D
struct CLLocationCoordinate2D {
    CLLocationDegrees latitude   // a Double
    CLLocationDegrees longitude  // a Double
}

var altitude: CLLocationDistance  // meters
```

A negative value means "below sea level"

Core Location

- How close to that latitude/longitude is the actual location?

```
var horizontalAccuracy: CLLocationAccuracy    // in meters
var verticalAccuracy: CLLocationAccuracy      // in meters
```

A negative value means the coordinate or altitude (respectively) is invalid. Other values ...

```
kCLLocationAccuracyBestForNavigation  // phone should be plugged in to power source
kCLLocationAccuracyBest
kCLLocationAccuracyNearestTenMeters
kCLLocationAccuracyHundredMeters
kCLLocationAccuracyKilometer
kCLLocationAccuracyThreeKilometers
```

- The more accuracy you request, the more battery will be used
  - Device "does its best" given a specified accuracy request
  - Cellular tower triangulation (not very accurate, but low power)
  - WiFi node database lookup (more accurate, more power)
  - GPS (very accurate, lots of power)

Core Location

- Speed

  `var speed: CLLocationSpeed` // meters/second

  Note that the speed is instantaneous (not average speed).

  Generally it's useful as "advisory information" when you are in a vehicle.

  A negative value means "speed is invalid."

- Course

  `var course: CLLocationDirection` // degrees, 0 is north, clockwise

  Not all devices can deliver this information.

  A negative value means "course is invalid."

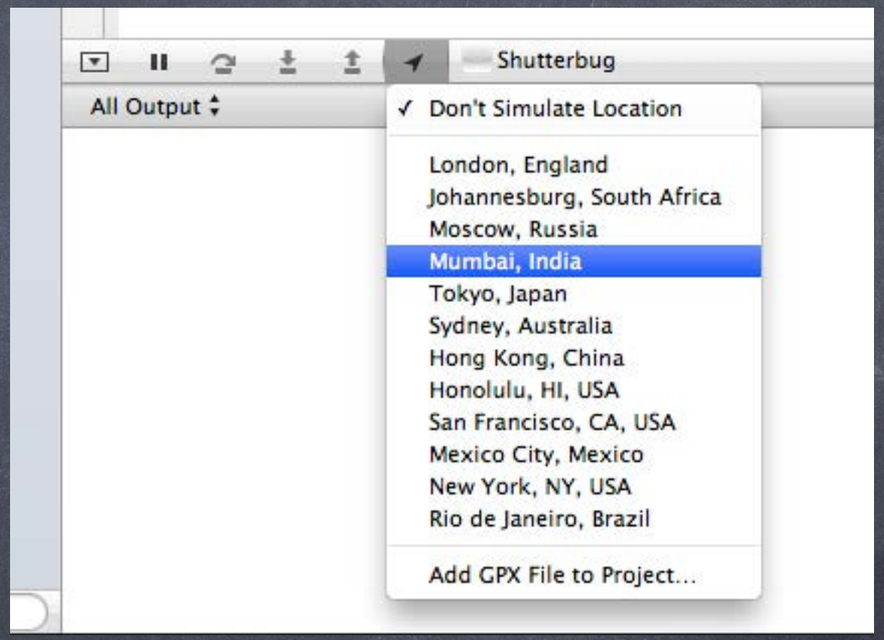
- Time stamp

  `var timestamp: NSDate`

  Pay attention to these since locations will be delivered on an inconsistent time basis.

Core Location

- How do you get a `CLLocation`?
  - Almost always from a `CLLocationManager` (sent to you via its `delegate`).
  - Can be tested in the simulator from Xcode.



Core Location

- How do you get a `CLLocation`?

  Almost always from a `CLLocationManager` (sent to you via its `delegate`).

  Can be tested in the simulator from Xcode.

- `CLLocationManager`

  General approach to using it:

  1. Check if the hardware you are on/user supports the kind of location updating you want.
  2. Create a `CLLocationManager` instance and set the `delegate` to receive updates.
  3. Configure the manager according to what kind of location updating you want.
  4. Start the manager monitoring for location changes.

Core Location

- Kinds of location monitoring
  - Accuracy-based continual updates.
  - Updates only when "significant" changes in location occur.
  - Region-based updates.
  - Heading monitoring.

Core Location

- Asking `CLLocationManager` what your hardware can do

```
class func authorizationStatus() -> CLAuthorizationStatus // Authorized, Denied or Restricted
class func locationServicesEnabled() -> Bool // user enabled (or not) locations for your app
class func significantLocationChangeMonitoringAvailable() -> Bool
class func isMonitoringAvailableForClass(AnyClass!) -> Bool // CLBeacon/CLCircularRegion
class func isRangingAvailable() -> Bool // device can tell how far it is from beacons
```

  Other tests for other location capabilities too.

- In addition, you must request authorization to get location info

  Because users won't necessarily trust your application to be monitoring their location ...

Core Location

- Asking the user if you can monitor their location
  - You do this by either …
  - `func requestWhenInUseAuthorization()` // you only want location data when you're active
  - `func requestAlwaysAuthorization()` // you want location data when you're not active too
  - These obtain authorization for you (from the user) asynchronously.
  - You can find out when authorization has been granted via a `delegate` method.
  - Until authorization is granted (if ever), your `authorizationStatus` will be `NotDetermined`.
- You must add an `Info.plist` entry for this to work …
  - `NSLocationWhenInUseUsageDescription`
  - `NSLocationAlwaysUsageDescription`
  - Right click in some open space in your `Info.plist`, then click Add Row to insert one

Core Location

- Getting the location information from the `CLLocationManager`
  - You can just ask (poll) the `CLLocationManager` for the location or heading, but usually we don't.
  - Instead, we let it update us when the location changes (enough) via its `delegate` ...

Core Location

- Accuracy-based continuous location monitoring

  ```
  var desiredAccuracy: CLLocationAccuracy // always set this as low as will work for you
  ```

  Can also limit updates to only occurring if the change in location exceeds a certain distance …

  ```
  var distanceFilter: CLLocationDistance
  ```

- Starting and stopping normal position monitoring

  ```
  func startUpdatingLocation()
  func stopUpdatingLocation()
  ```

  Be sure to turn updating off when your application is not going to consume the changes!

- Now get notified via the `CLLocationManager`'s `delegate`

  ```
  func locationManager(CLLocationManager, didUpdateLocations: [CLLocation])
  ```

- Similar API for heading (`CLHeading`, et. al.)

Core Location

- Error reporting to the `delegate`

```
func locationManager(CLLocationManager, didFailWithError: NSError)
```

Not always a fatal thing, so pay attention to this `delegate` method. Some examples ...

```
kCLErrorLocationUnknown   // likely temporary, keep waiting (for a while at least)
kCLErrorDenied            // user refused to allow your application to receive updates
kCLErrorHeadingFailure    // too much local magnetic interference, keep waiting
```

Core Location

- Background

  It is possible to receive these kinds of updates while you are in the background.
  Apps that do this have to be very careful (because these updates can be power hungry).
  There are very cool ways to, for example, coalesce and defer location update reporting.
  Have to enable backgrounding (in the same area of your project settings as background fetch).

  But there are 2 ways to get location notifications (on a coarser scale) without doing that ...

Core Location

- Significant location change monitoring in `CLLocationManager`
  - "Significant" is not strictly defined. Think vehicles, not walking. Likely uses cell towers.
  - `func startMonitoringSignificantLocationChanges()`
  - `func stopMonitoringSignificantLocationChanges()`
  - Be sure to turn updating off when your application is not going to consume the changes!
- Get notified via the `CLLocationManager`'s `delegate`
  - Same as for accuracy-based updating if your application is running.

Core Location

- And this works even if your application is not running!
  (Or is in the background)
  You will get launched and your Application Delegate's
  `func application(UIApplication, didFinishLaunchingWithOptions: [NSObject:AnyObject])`
  will have a dictionary entry for `UIApplicationLaunchOptionsLocationKey`
  Create a `CLLocationManager` (if you don't have one), then get the latest location via ...
  `var location: CLLocation`
  If you are running in the background, don't take too long (a few seconds)!

Core Location

- Region-based location monitoring in CLLocationManager

```
func startMonitoringForRegion(CLRegion) // CLCircularRegion/CLBeaconRegion
func stopMonitoringForRegion(CLRegion)
let cr = CLCircularRegion(center: CLLocationCoordinate2D,
                          radius: CLLocationDistance,
                      identifier: String) … to monitor an area
```

  CLBeaconRegion is for detecting when you are near another device.

- Now get notified via the CLLocationManager's delegate

```
func locationManager(CLLocationManager, didEnterRegion: CLRegion)
func locationManager(CLLocationManager, didExitRegion: CLRegion)
func locationManager(CLLocationManager, monitoringDidFailForRegion: CLRegion,
          withError: NSError)
```

Core Location

- Region-monitoring also works if your application is not running
  - In exactly the same way as "significant location change" monitoring.
  - The set of monitored regions persists across application termination/launch.
  - `var monitoredRegions: Set<CLRegion>` // this is a property in `CLLocationManager`
- CLRegions are tracked by name
  - Because they survive application termination/relaunch.
- Circular region monitoring size limit
  - `var maximumRegionMonitoringDistance: CLLocationDistance { get }`
  - Attempting to monitor a region larger than this (radius in meters) will generate an error (which will be sent via the `delegate` method mentioned on previous slide).
  - If this property returns a negative value, then region monitoring is not working.

Core Location

- Beacon regions can also detect range from a beacon

  `func startRangingBeaconsInRegion(CLBeaconRegion)`

  Delegate method `locationManager(didRangeBeacons:inRegion:)` gives you `CLBeacon` objects.

  `CLBeacon` objects will tell you proximity (e.g. `CLProximityImmediate/Near/Far`).

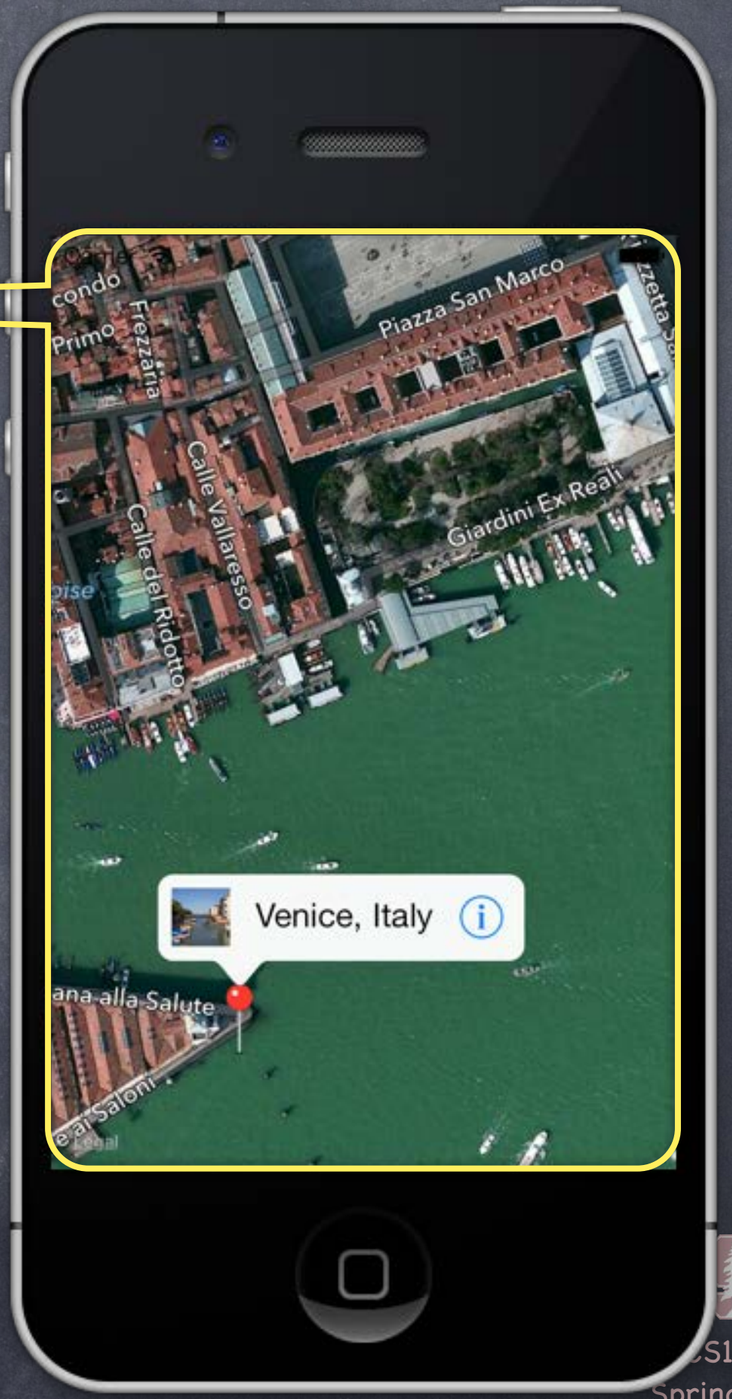
- To be a beacon is a bit more involved

  Beacons are identified by a globally unique UUID (that you generate).

  Check out `CBPeripheralManager` (Core Bluetooth Framework).

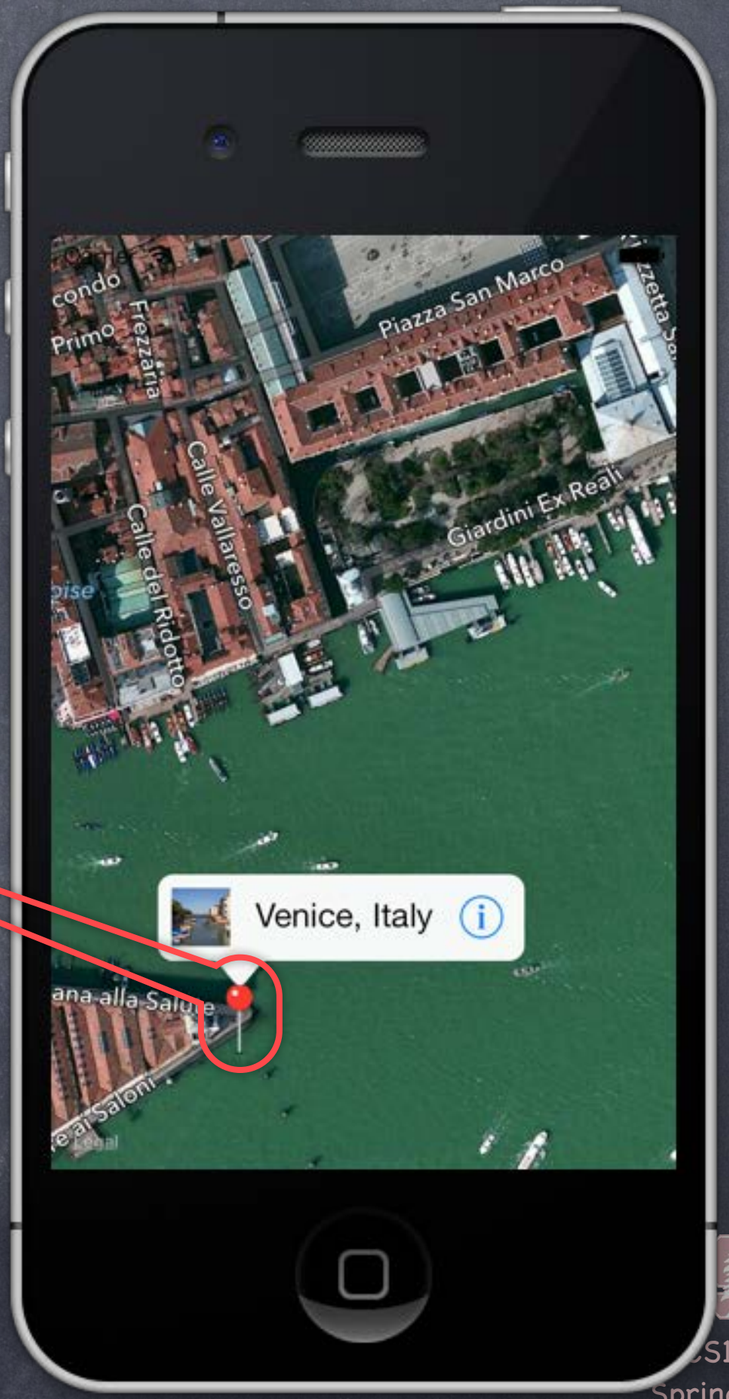
Map Kit

- `MKMapView` displays a map



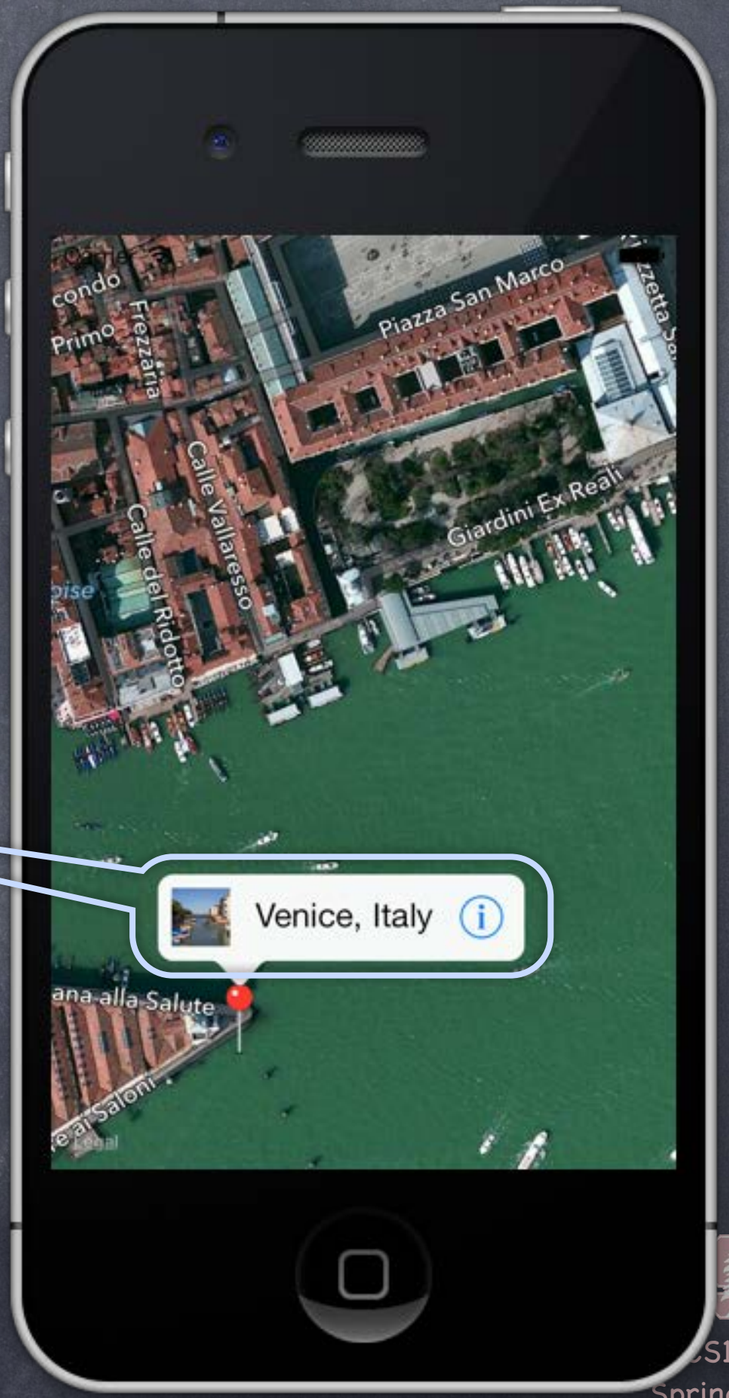
Map Kit

- `MKMapView` displays a map
- The map can have annotations on it
  Each annotation is a `coordinate`, a `title` and a `subtitle`.
  They are displayed using an `MKAnnotationView`
  (`MKPinAnnotationView` shown here).



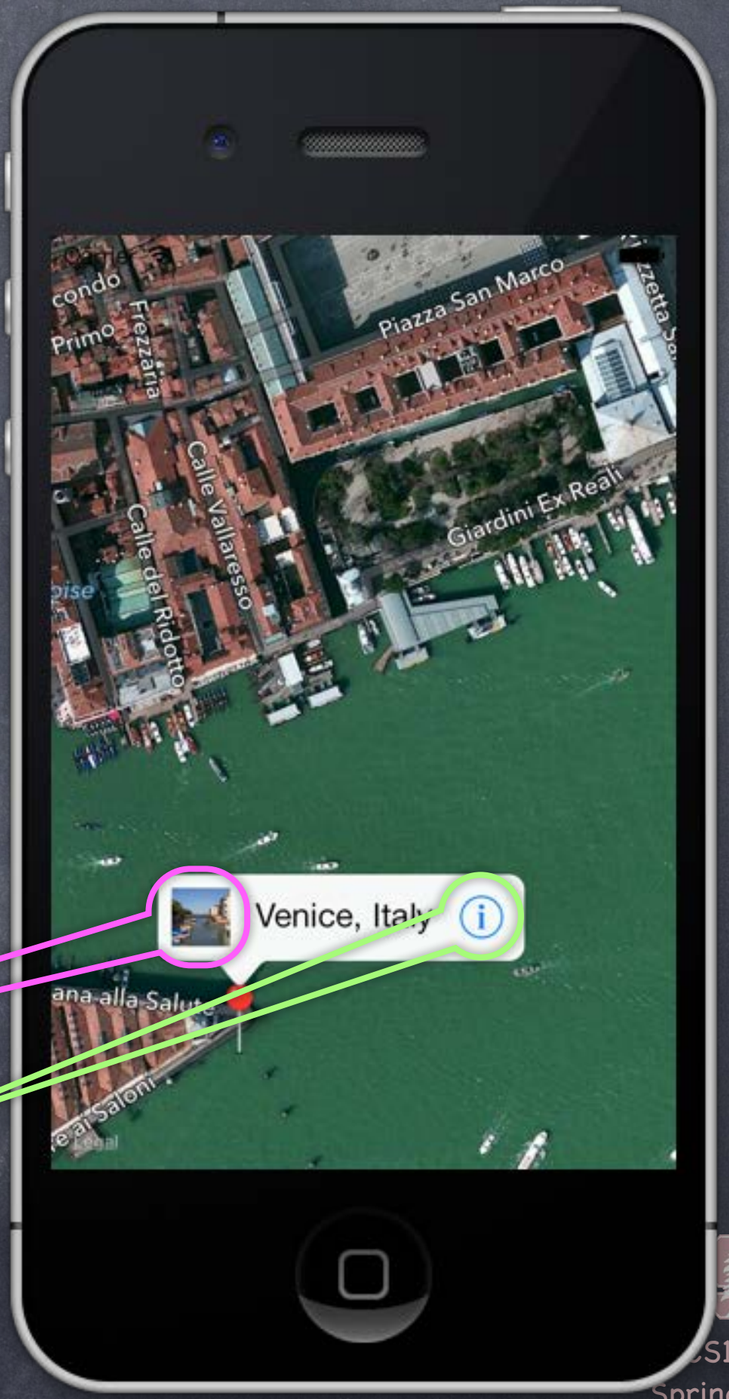
Map Kit

- `MKMapView` displays a map
- The map can have annotations on it
  Each annotation is a `coordinate`, a `title` and a `subtitle`.
  They are displayed using an `MKAnnotationView`
  (`MKPinAnnotationView` shown here).
- Annotations can have a `callout`
  It appears when the annotation view is clicked.
  By default just shows the `title` and `subtitle`.



Map Kit

- MKMapView displays a map
- The map can have annotations on it
  Each annotation is a coordinate, a title and a subtitle.
  They are displayed using an MKAnnotationView
  (MKPinAnnotationView shown here).
- Annotations can have a callout
  It appears when the annotation view is clicked.
  By default just shows the title and subtitle.
- But callout can also have accessory views
  In this example, the left is a UIImageView,
  the right is a UIButton (UIButtonType.DetailDisclosure)



MKMapView

- Create with initializer or drag from object palette in Xcode

```
import MapKit
let mapView = MKMapView()
```

MKMapView

- Displays an array of objects which implement MKAnnotation

```
var annotations: [MKAnnotation] { get }
```

- MKAnnotation protocol

```
protocol MKAnnotation: NSObject
    var coordinate: CLLocationCoordinate2D { get }
    var title: String? { get }      // optional, but expected to be provided
    var subtitle: String? { get }
}
```

Remember this from CoreLocation …

```
struct CLLocationCoordinate2D {
    var latitude: CLLocationDegrees
    var longitude: CLLocationDegrees
}
```

MKAnnotation

- Note that the annotations property is readonly, so ...

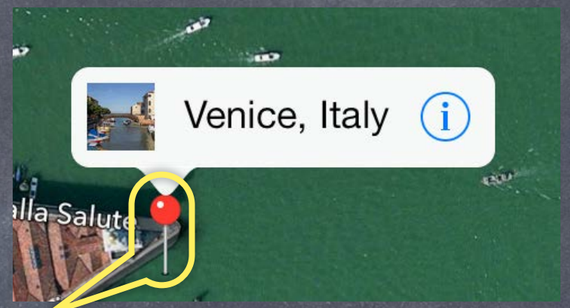
```
var annotations: [MKAnnotation] { get }
```

Must add/remove annotations with these methods ...

```
func addAnnotation(MKAnnotation)
func addAnnotations([MKAnnotation)
func removeAnnotation(MKAnnotation)
func removeAnnotations([MKAnnotation])
```

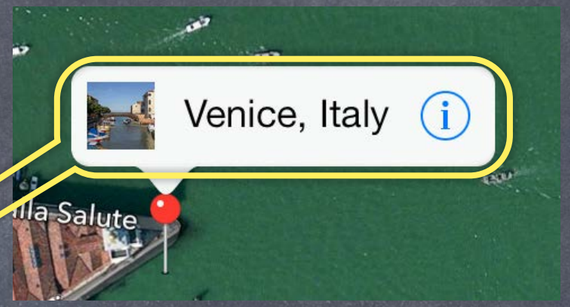
- Generally a good idea to add all your annotations up-front
  - Allows the `MKMapView` to be efficient about how it displays them.
  - Annotations are light-weight, but annotation views are not.
  - Luckily `MKMapView` reuses annotation views similar to how `UITableView` reuses cells.

MKAnnotation



- What do annotations look like on the map?
  - Annotations are drawn using an `MKAnnotationView` subclass.
  - The default one is `MKPinAnnotationView` (which is why they look like pins by default).
  - You can subclass or set properties on existing `MKAnnotationViews` to modify the look.

MKAnnotation



- What do annotations look like on the map?
  - Annotations are drawn using an `MKAnnotationView` subclass.
  - The default one is `MKPinAnnotationView` (which is why they look like pins by default).
  - You can subclass or set properties on existing `MKAnnotationViews` to modify the look.
- What happens when you touch on an annotation (e.g. the pin)?
  - Depends on the `MKAnnotationView` that is associated with the annotation (more on this later).
  - If `canShowCallout` is `true` in the `MKAnnotationView`,
    - then a little box will appear showing the annotation's `title` and `subtitle`.
  - And this little box (the callout) can be enhanced with `left`/`rightCalloutAccessoryViews`.

  The following `MKMapViewDelegate` method is also called…

  ```
  func mapView(MKMapView, didSelectAnnotationView: MKAnnotationView)
  ```

  This is a great place to set up the `MKAnnotationView`'s callout accessory views lazily.
  For example, you might want to wait until this method is called to download an image to show.

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

  There is no "prototype" in your storyboard, you create them in code as above.

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

  There is no "prototype" in your storyboard, you create them in code as above.

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

  There is no "prototype" in your storyboard, you create them in code as above.

MKAnnotationView

- How are MKAnnotationViews created & associated w/annotations?

  Very similar to UITableViewCells in a UITableView.

  Implement the following MKMapViewDelegate method (if not implemented, returns a pin view).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView {
    var view: MKAnnotationView! =
        sender.dequeueReusableAnnotationViewWithIdentifier(IDENT)
    if view == nil {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: IDENT)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // yes, this happens twice if no dequeue
    // prepare and (if not too expensive) load up accessory views here
    // or reset them and wait until mapView(didSelectAnnotationView:) to load actual data
    return view
}
```

  There is no "prototype" in your storyboard, you create them in code as above.

MKAnnotationView

- Interesting properties …

```
var annotation: MKAnnotation  // the annotation; treat as if readonly
var image: UIImage  // instead of the pin, for example (not an image on the callout)
var leftCalloutAccessoryView: UIView  // maybe a UIImageView
var rightCalloutAccessoryView: UIView  // maybe a "disclosure" UIButton
var enabled: Bool  // false means it ignores touch events, no delegate method, no callout
var centerOffset: CGPoint  // where is the "head of the pin" is relative to the image
var draggable: Bool  // only works if the annotation's coordinate property is { get set }
```

- If you set one of the callout accessory views to be a UIControl

e.g. `view.rightCalloutAccessoryView = UIButton(type: .DetailDisclosure)`

The following MKMapViewDelegate method will get called when the callout view is touched …

```
func mapView(MKMapView, annotationView: MKAnnotationView,
             calloutAccessoryControlTapped: UIControl)
```

Maybe you might manually segue from here, for example.

MKAnnotationView

- Using `didSelectAnnotationView:` to load up callout accessories

  Example … downloading a an image into a `leftCalloutAccessoryView` that is a `UIImageView`.

  In `mapView(viewForAnnotation:)`, `let view.leftCalloutAccessoryView = UIImageView()`

  Reset the `UIImageView`'s `image` to `nil` there as well (because of reuse).

  Then load the `image` on demand in `mapView(didSelectAnnotationView:)` …

```
func mapView(MKMapView, didSelectAnnotationView aView: MKAnnotationView)
{
    if let imageView = aView.leftCalloutAccessoryView as? UIImageView {
       imageView.image = ... // if you do this on another thread, be careful, views are reused!
    }
}
```

MKMapView

- Configuring the map view's display type

```
var mapType: MKMapType // .Standard, .Satellite, .Hybrid
```

- Showing the user's current location

```
var showsUserLocation: Bool
var isUserLocationVisible: Bool
var userLocation: MKUserLocation
```

MKUserLocation is an object which conforms to MKAnnotation which holds the user's location.

- Restricting the user's interaction with the map

```
var zoomEnabled: Bool
var scrollEnabled: Bool
var pitchEnabled: Bool   // 3D
var rotateEnabled: Bool
```

MKMapCamera

- Setting where the user is seeing the map from (in 3D)

```
var camera: MKMapCamera // property in MKMapView
```

- MKMapCamera

  Specify centerCoordinate, heading, pitch and altitude of the camera.

  Or use convenient initializer in MKMapCamera …

```
let camera = MKMapCamera(lookingAtCenterCoordinate: CLLocationCoordinate2D,
                                     fromEyeCoordinate: CLLocationCoordinate2D,
                                           eyeAltitude: CLLocationDistance)
```

MKMapView

- Controlling the region (part of the world) the map is displaying

```
var region: MKCoordinateRegion
struct MKCoordinateRegion {
    var center: CLLocationCoordinate2D // remember from CoreLocation, this is lat/long
    var span: MKCoordinateSpan
}
struct MKCoordinateSpan {
    var latitudeDelta: CLLocationDegrees
    var longitudeDelta: CLLocationDegrees
}
func setRegion(MKCoordinateRegion, animated: Bool) // animate setting the region
```

- Can also set the center point only or set to show annotations

```
var centerCoordinate: CLLocationCoordinate2D
func setCenterCoordinate(CLLocationCoordinate2D, animated: Bool)
func showAnnotations([MKAnnotation], animated: Bool)
```

MKMapView

- Lots of C-like functions to convert points, regions, rects, etc.

  See documentation, e.g. `MKMapRectContainsPoint`, `MKMapPointForCoordinate`, etc.

- Converting to/from map points/rects from/to view coordinates

```
func mapPointForPoint(CGPoint) -> MKMapPoint
func mapRectForRect(CGRect) -> MKMapRect
func pointForMapPoint(MKMapPoint) -> CGPoint
func rectForMapRect(MKMapRect) -> CGRect
```

  etc.

MKMapView

- Another `MKMapViewDelegate` method …

  `func mapView(MKMapView, didChangeRegionAnimated: Bool)`

  This is a good place to "chain" animations to the map.

  When you display somewhere new in the map that is far away, zoom out, then back in.

  This method will let you know when it's finished zooming out, so you can then zoom in.

MKLocalSearch

- Searching for places in the world

  Can search by "natural language" strings asynchronously (uses the network) …

```
var request = MKLocalSearchRequest()
request.naturalLanguageQuery = "Ike's"
request.region = … // e.g., Stanford campus
let search = MKLocalSearch(request: request)
search.startWithCompletionHandler { (MKLocalSearchResponse?, NSError?) -> Void in
    // response contains an array of MKMapItem which contains MKPlacemark
    // MKPlacemark contains location, name of location, postalCode, region, etc.
}
```

- MKMapItem

  You can open a MKMapItem that you get back from a MKLocalSearch in the Maps app …

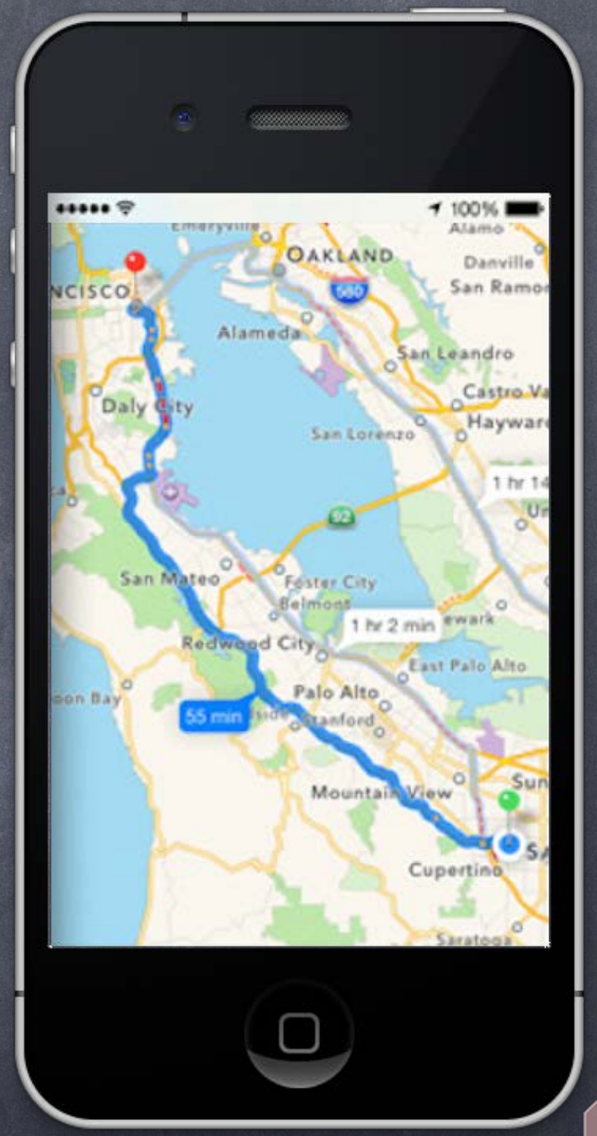
```
func openInMapsWithLaunchOptions([String:AnyObject]?) -> Bool
// the options can specify region, show traffic, etc
```

MKDirections

- Getting directions from one place to another
  - Very similar API to searching.
  - Specify source and destination `MKMapItem`.
  - Asynchronous API to get a bunch of `MKRoutes`.

  `MKRoute` includes a name for the route,
  turn-by-turn directions, expected travel time, etc.

  Also come with `MKPolyline` descriptions of the routes
  which can be overlaid on the map …



Overlays

- Overlays

  Add overlays to the MKMapView and it will later ask you for a renderer to draw the overlay.

  ```
  func addOverlay(MKOverlay, level: MKOverlayLevel)
  ```

  Level is (currently) either AboveRoads or AboveLabels (over everything but annotation views).

  ```
  func removeOverlay(MKOverlay)
  ```

- MKOverlay protocol

  Protocol which includes MKAnnotation plus …

  ```
  var boundingMapRect: MKMapRect
  func intersectsMapRect(MKMapRect) -> Bool // optional, uses boundingMapRect otherwise
  ```

- Overlays are associated with MKOverlayRenderers via delegate

  Just like annotations are associated with MKAnnotationViews …

  ```
  func mapView(MKMapView, rendererForOverlay: MKOverlay) -> MKOverlayRenderer
  ```

MKOverlayView

- Built-in Overlays and Renderers for numerous shapes …

```
MKCircleRenderer
MKPolylineRenderer
MKPolygonRenderer
MKTileOverlayRenderer  // can also be used to replace the map data from Apple
```

There's a whole set of `MKShape` and subclasses thereof for you to explore.