COORDINATORS

RELOADED

ABOUT ME

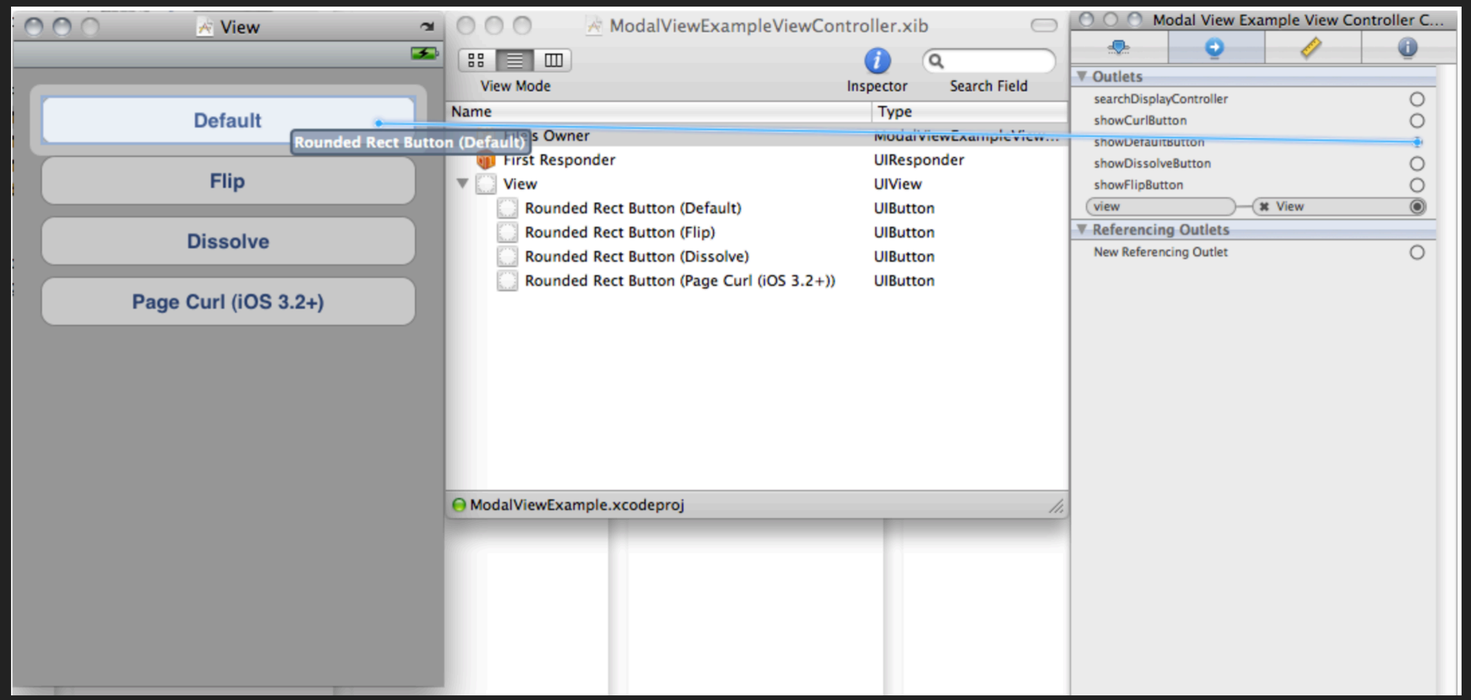
- Chris Combs
- Head of iOS Development @ Nodes

@chriscombs76

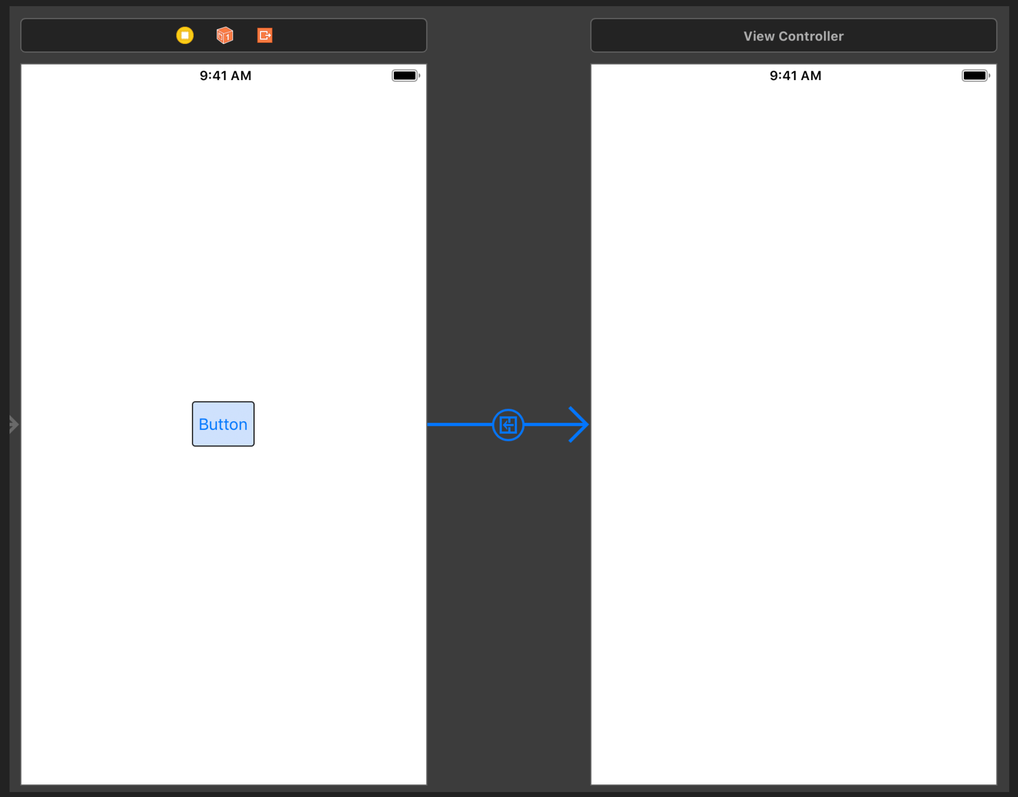
HISTORY

XIB - OLDSCHOOL

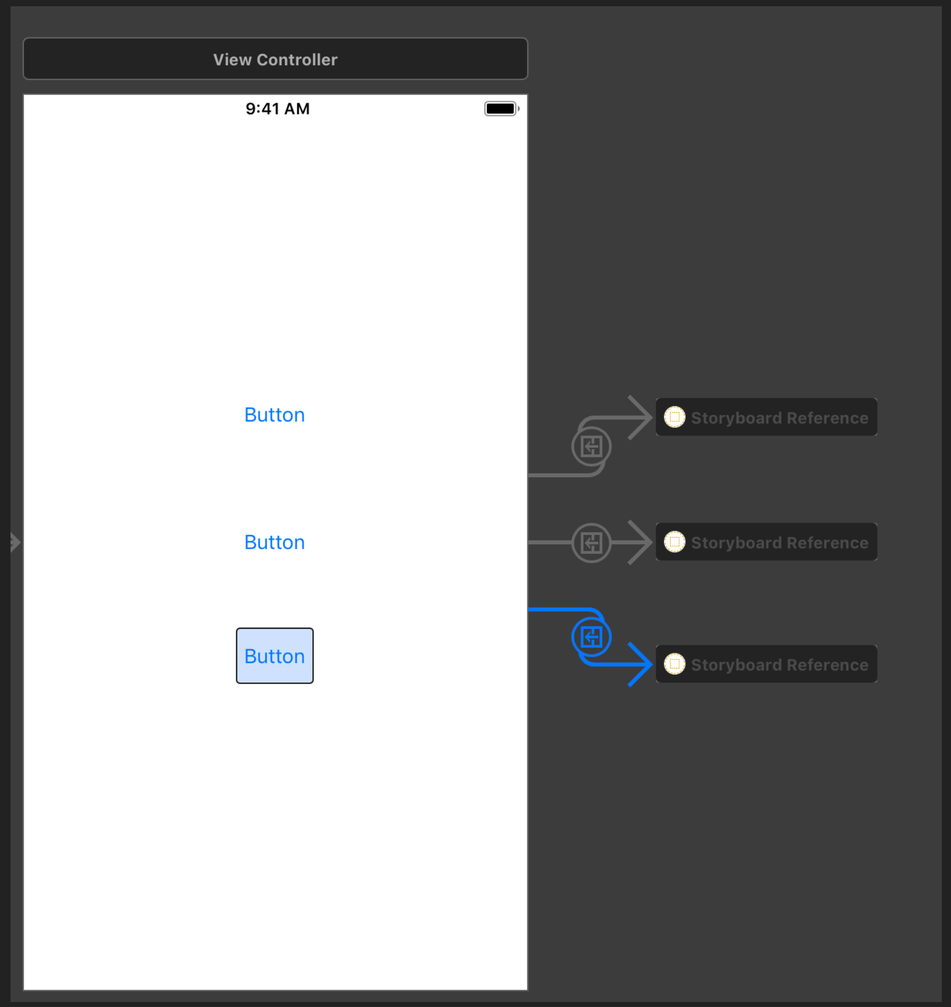
```
MyViewController *myView = [[[MyViewController alloc] init] autorelease];
[self presentModalViewController:myView animated:YES];
```



WELCOME TO STORYBOARDS



REFERENCES


View Controller
9:41 AM
Button
Button
Button
Storyboard Reference
Storyboard Reference
Storyboard Reference

IS THIS STILL A GOOD IDEA?

WHY NOT REVERT TO NAVIGATION IN CODE?

```
[self performSegueWithIdentifier:@"mySegueIdentifier" sender:self];
```

- or...

```
let viewController = UIStoryboard(name: "Main", bundle: nil).instantiateInitialViewController()
present(viewController, animated: true, completion: nil)
```

ENTER THE COORDINATOR PATTERN

NSSPAIN 2015

- Soroush Khanlou @khanlou presents the Coordinator Pattern
  - Blog post: http://khanlou.com/2015/01/the-coordinator/

WHAT DO WE WANT TO SOLVE?

- Storyboard nightmares
- Separation of concerns (i.e. MassiveViewController)
  - *“At their core, coordinators allow us to take navigation logic out of our view controllers, so they have no idea what comes next in our app flow or even that an app flow exists. This means your view controller no longer needs to know about, create, configure, then display an entirely different view controller.”* - Paul Hudson

THE BASIC IDEA

- Coordinators direct the app.
- Top level AppCoordinator has child coordinators, which have child coordinators, and so on, each one representing new sections of the app.
- Coordinators listen to ViewController navigation requests, figure out the next step in the flow, and then create and push/present the next view.

BASIC EXAMPLE

```
class MainCoordinator: Coordinator {
    var children: [Coordinator] = []
    let navigationController = UINavigationController()

    func start() {
        let vc = UIStoryboard(name: "Main", bundle: nil).instantiateInitialViewController()!
        vc.coordinator = self
        navigationController.setViewControllers([vc], animated: true)
    }

    func navigateToDetails() {
        let vc = UIStoryboard(name: "Details", bundle: nil).instantiateInitialViewController()!
        vc.coordinator = self
        navigationController.pushViewController(vc, animated: true)
    }

    func presentModal() {
        let coordinator = ModalCoordinator(presentingController: navigationController)
        children.append(coordinator)
        coordinator.start()
    }
}
```

3 YEARS LATER

PROS

- Easy to reuse views
- There are no surprises in app routing
- Cleaner AppDelegate
- Cleaner deep linking

CONS

- Complex and steep learning curve
- A lot of boilerplate and reused code
- Not at all how Apple intended

MAKING IT ALL WORTH IT

ONE UINAVIGATIONCONTROLLER = ONE COORDINATOR

- One coordinator for each view controller is redundant
- Navigation controllers are the root of an app section
- New navigation controllers presented on top will have their own coordinators

DELEGATES VS CLOSURES

- Closures can be cleaner, and yet also messier
- Navigation events handled in-scope

```
vc.podcastNavigationHandler = { podcast in
    self.navigateToPodcast(podcast)
}
```

- Nested closures become overwhelming

DELEGATES VS CLOSURES

- Delegates are simpler and more straightforward
- Can end up with way too many delegates

```
    func navigateToProfile() { }
    func navigateToDetails() { }
    func navigateToSearch() { }
    func navigateToSettings() { }
    func navigateToLogin() { }
```

- etc..

NAVIGATE FUNCTION

- A simple compromise

```
func navigate(to route: Search.Route) {
        switch route {
        case .episode(let episode, let podcast):
            navigateToEpisode(episode, podcast: podcast)
        case .podcast(let podcast):
            navigateToPodcast(podcast)
        }
    }
```

CAN YOU FORESEE A PROBLEM?

ONE PROTOCOL PER VIEW

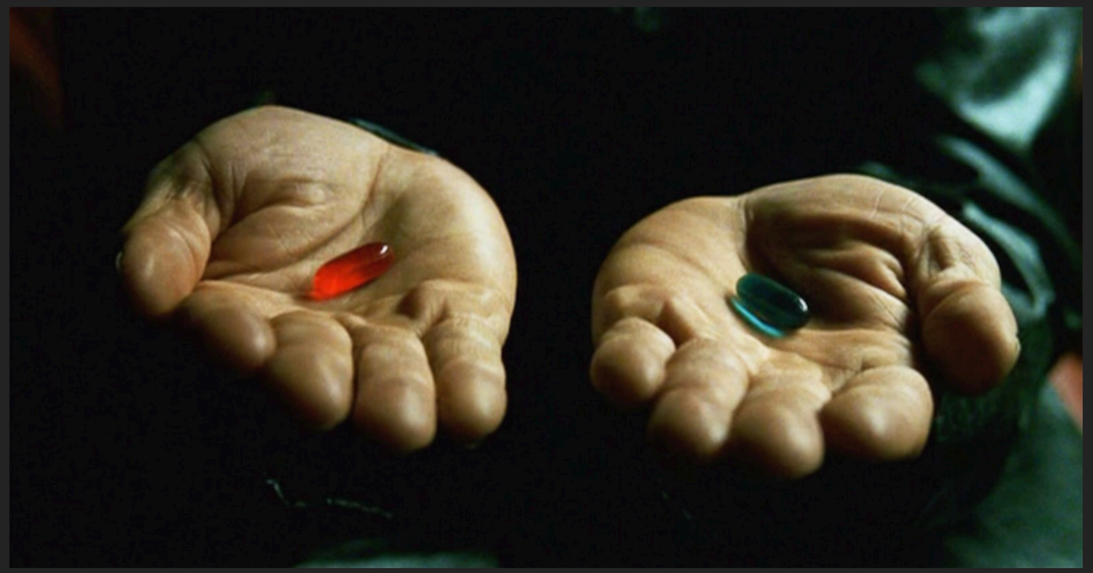
- Dependency Injection using Protocols!

```
extension SearchCoordinator: PodcastDetailsCoordinatorInput {
    func navigate(to route: PodcastDetails.Route) {
        switch route {
        case .episode(let episode, let podcast):
            navigateToEpisode(episode, podcast: podcast)
        }
    }
}

extension SearchCoordinator: EpisodeDetailsCoordinatorInput {
    func navigate(to route: EpisodeDetails.Route) {
        switch route {
        case .player(let streamUrl):
            navigateToPlayer(streamUrl)
        }
    }
}
```

CLOSING REMARKS

- Will this pattern continue to grow?
- Will the community continue to search for alternatives?
- Will Apple adopt it?
- Will Nodes keep using it?
- Will you use it?



@CHRISCOMBS76

THANK YOU