COCOAHEADS MONTRÉAL 2017/09/21

THIBAULT WITTEMBERG - MOBILE ARCHITECT AT SAVOIR-FAIRE LINUX

LET'S WEAVE YOUR APPLICATION

THE FACTS

REGARDING NAVIGATION WITHIN AN IOS APPLICATION,
TWO CHOICES ARE AVAILABLE:

- USE THE BUILT-IN MECHANISMS PROVIDED BY APPLE AND XCODE: STORYBOARDS AND SEGUES
- IMPLEMENT A CUSTOM MECHANISM DIRECTLY IN THE CODE

The Facts

THE DRAWBACKS

Built-in mechanisms

- Navigation is relatively static
- Storyboards are massive / hard to collaborate
- The navigation code pollutes the VCs
- Difficult to do Dependency Injection

Custom mechanisms

- Which pattern ? (Flow Coordinator / Router / Redux)
- Can be hard to understand for new teammates
- Can be complex to set up

WHAT WOULD WE LIKE TO ACHIEVE ?

Promote the cutting of storyboards into atomic units

Reuse VCs within different navigation contexts

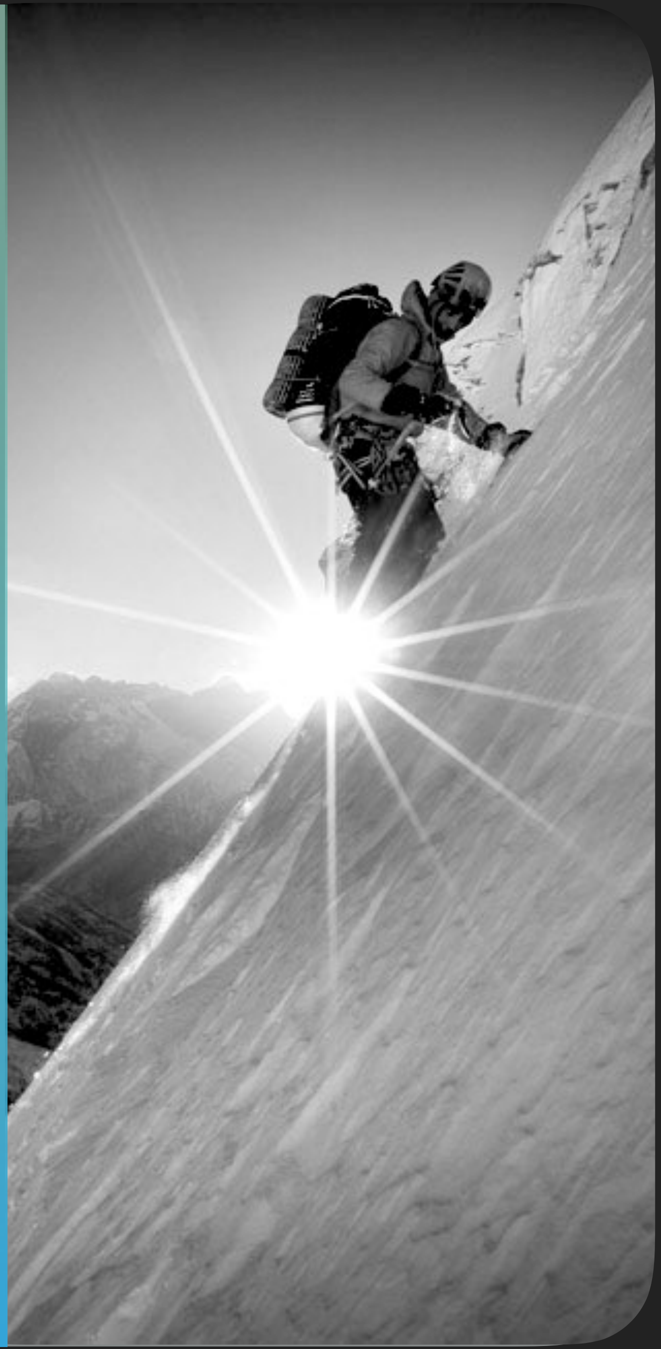
Ease the implementation of Dependency Injection

Remove navigation code from VCs

Promote Reactive Programming

Describe the navigation in a more declarative way

Cut our application into logical units of navigation

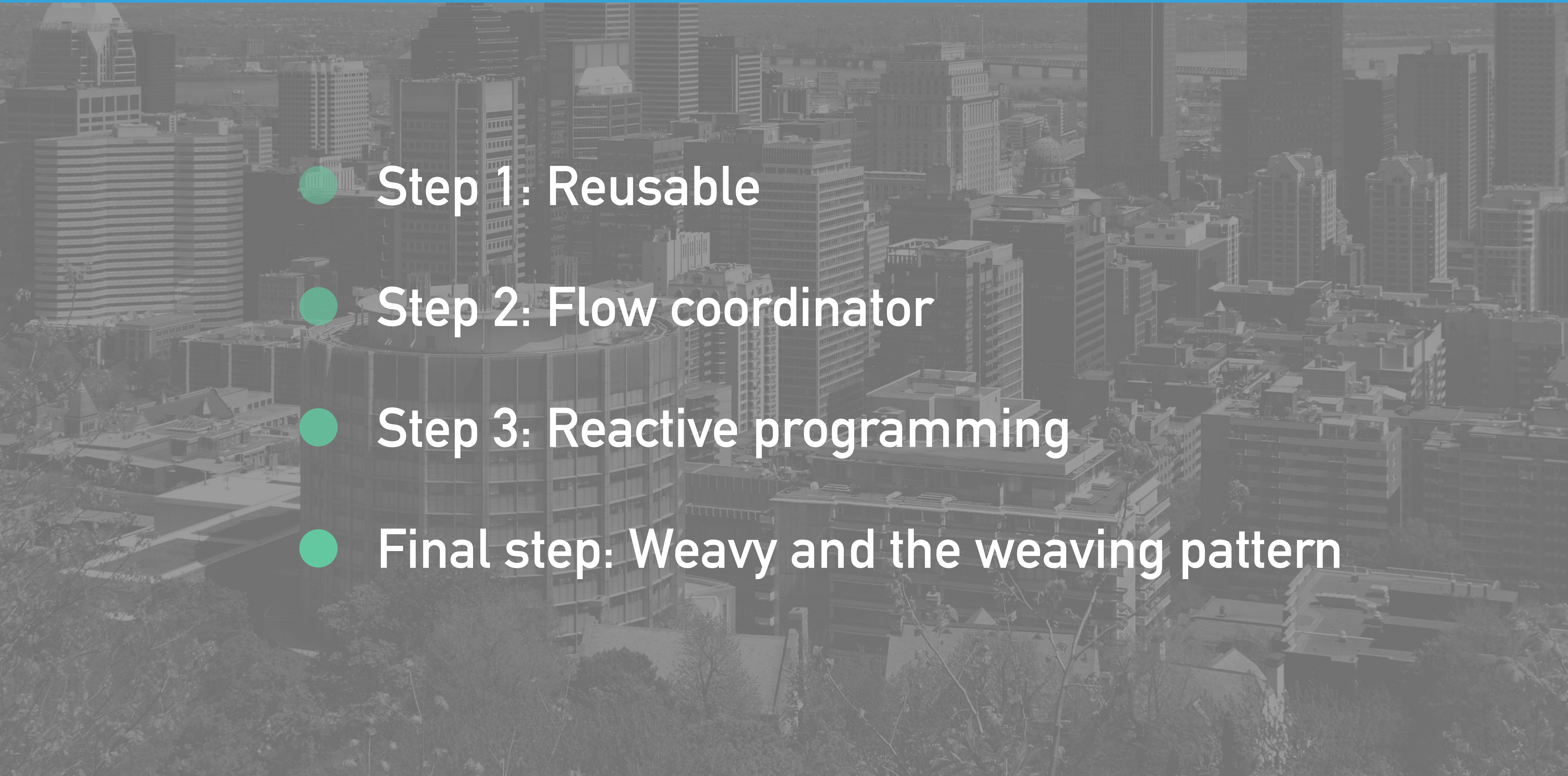


HE’S DEAD, JIM !

Doctor McCoy – StarTrek

LET'S WEAVE YOUR APPLICATION

« THESE ACHIEVEMENTS ARE COMPLETED STEP BY STEP THROUGH A JOURNEY THAT LEADS US TO THE IDEA OF A WEAVING PATTERN »



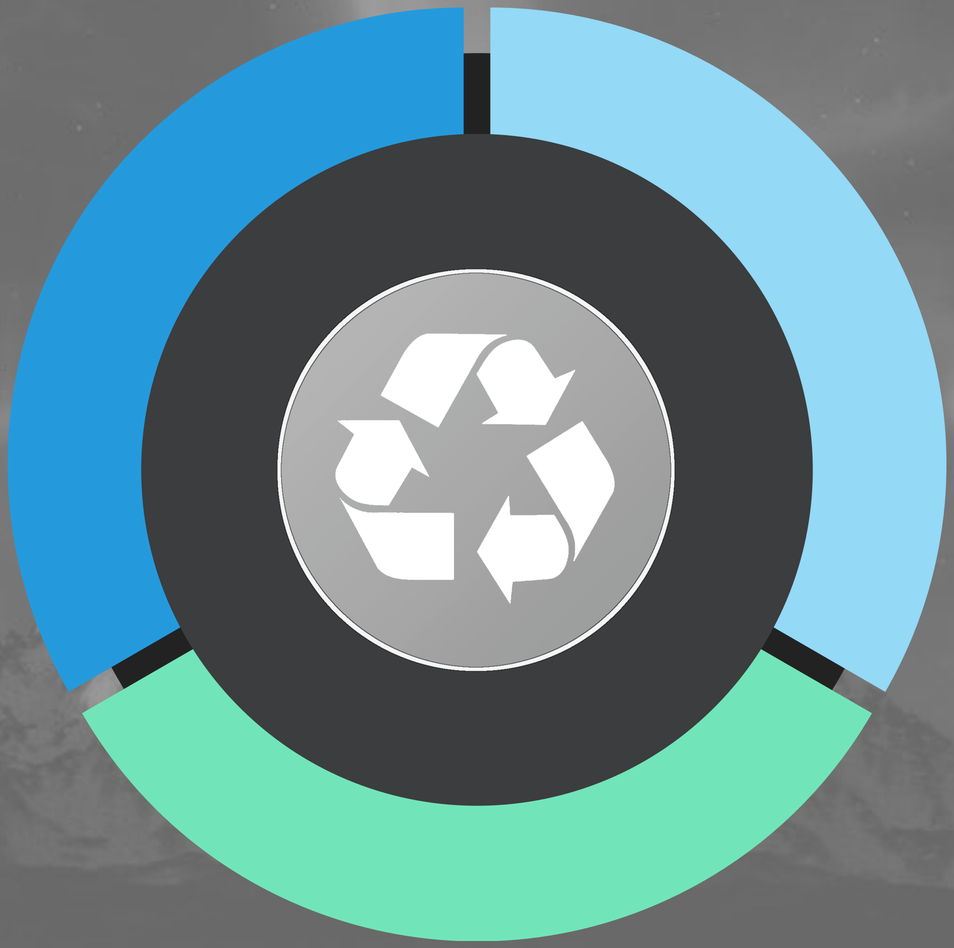
- Step 1: Reusable
- Step 2: Flow coordinator
- Step 3: Reactive programming
- Final step: Weavy and the weaving pattern

STEP 1

REUSABLE

LET'S WEAVE YOUR APPLICATION: REUSABLE

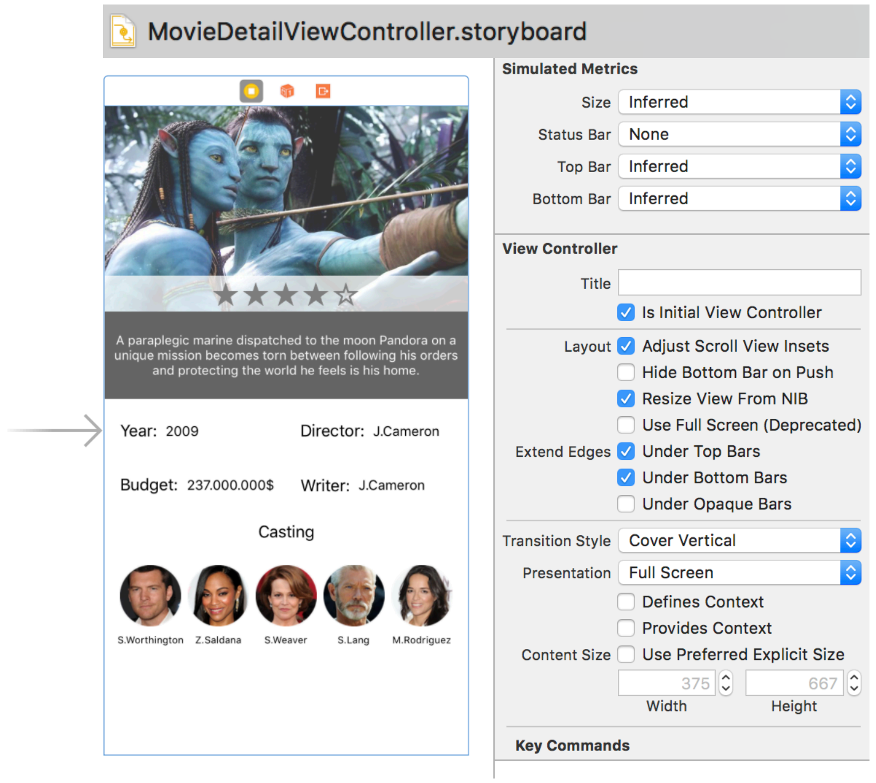
Lightweight OS API
by Olivier Halligon



Instantiates VC
in a type safe way

Protocol Oriented Programming

LET'S WEAVE YOUR APPLICATION: REUSABLE


MovieDetailViewController.storyboard

A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.

Year: 2009 Director: J.Cameron

Budget: 237.000.000$ Writer: J.Cameron

Casting

S.Worthington Z.Saldana S.Weaver S.Lang M.Rodriguez

Simulated Metrics

Size: Inferred
Status Bar: None
Top Bar: Inferred
Bottom Bar: Inferred

View Controller

Title:
[✓] Is Initial View Controller
Layout: [✓] Adjust Scroll View Insets
[ ] Hide Bottom Bar on Push
[✓] Resize View From NIB
[ ] Use Full Screen (Deprecated)
Extend Edges: [✓] Under Top Bars
[✓] Under Bottom Bars
[ ] Under Opaque Bars
Transition Style: Cover Vertical
Presentation: Full Screen
[ ] Defines Context
[ ] Provides Context
Content Size: [ ] Use Preferred Explicit Size
375 Width 667 Height

Key Commands

```
import Reusable

class MovieDetailViewController: UIViewController, StoryboardBased {
    ...
}
```

```
// One line – type safe – instantiation (no more identifier)

let viewController = MovieDetailViewController.instantiate()
window.rootViewController = viewController
```

1 VC per Storyboard

Super easy to instantiate in code

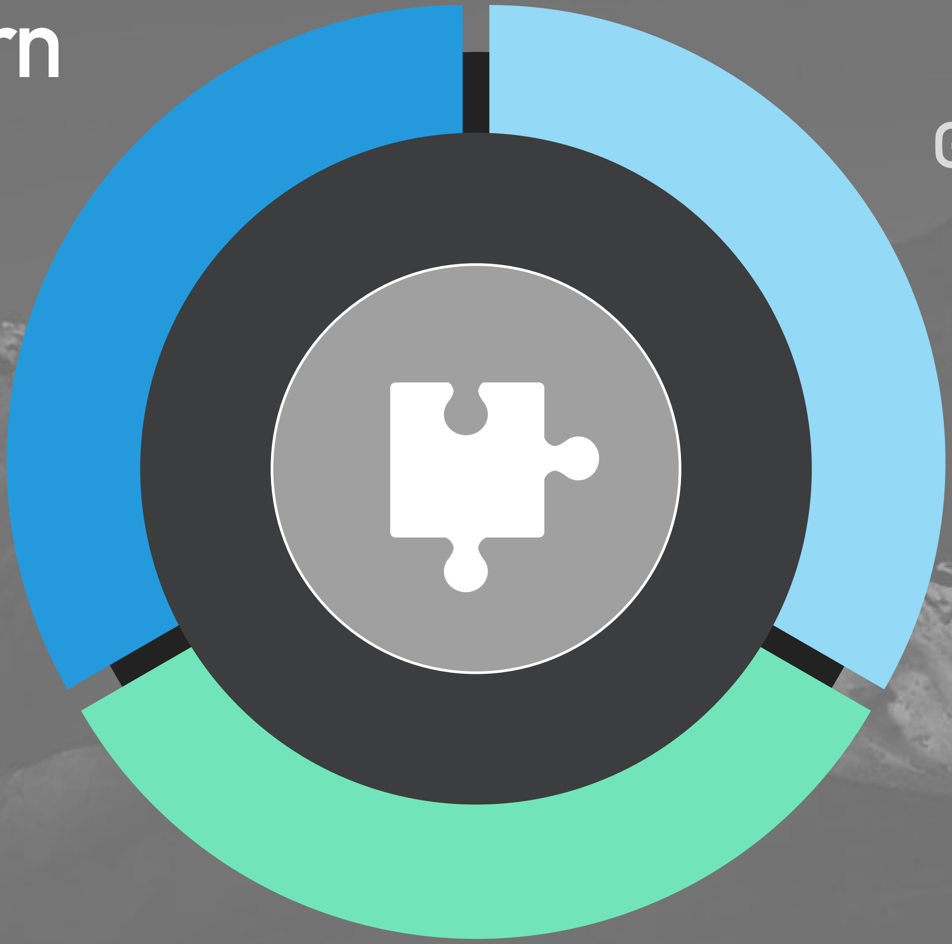
WE ALREADY HAVE 2 ACHIEVEMENTS

- Promote the cutting of storyboards into atomic units
- Reuse VCs within different navigation contexts

STEP 2

FLOW COORDINATOR

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR



Composition Pattern

Great for navigation structuration

Instantiates VCs

Great for separation of concerns

Great with Reusable API

Great for DI

Acts like a black box

VCs are not aware of their navigation context

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR

Main flow - Navigation stack

Settings - root

Dashboard - push

Wishlist flow - Navigation stack

Movies list - root

Movie detail - push

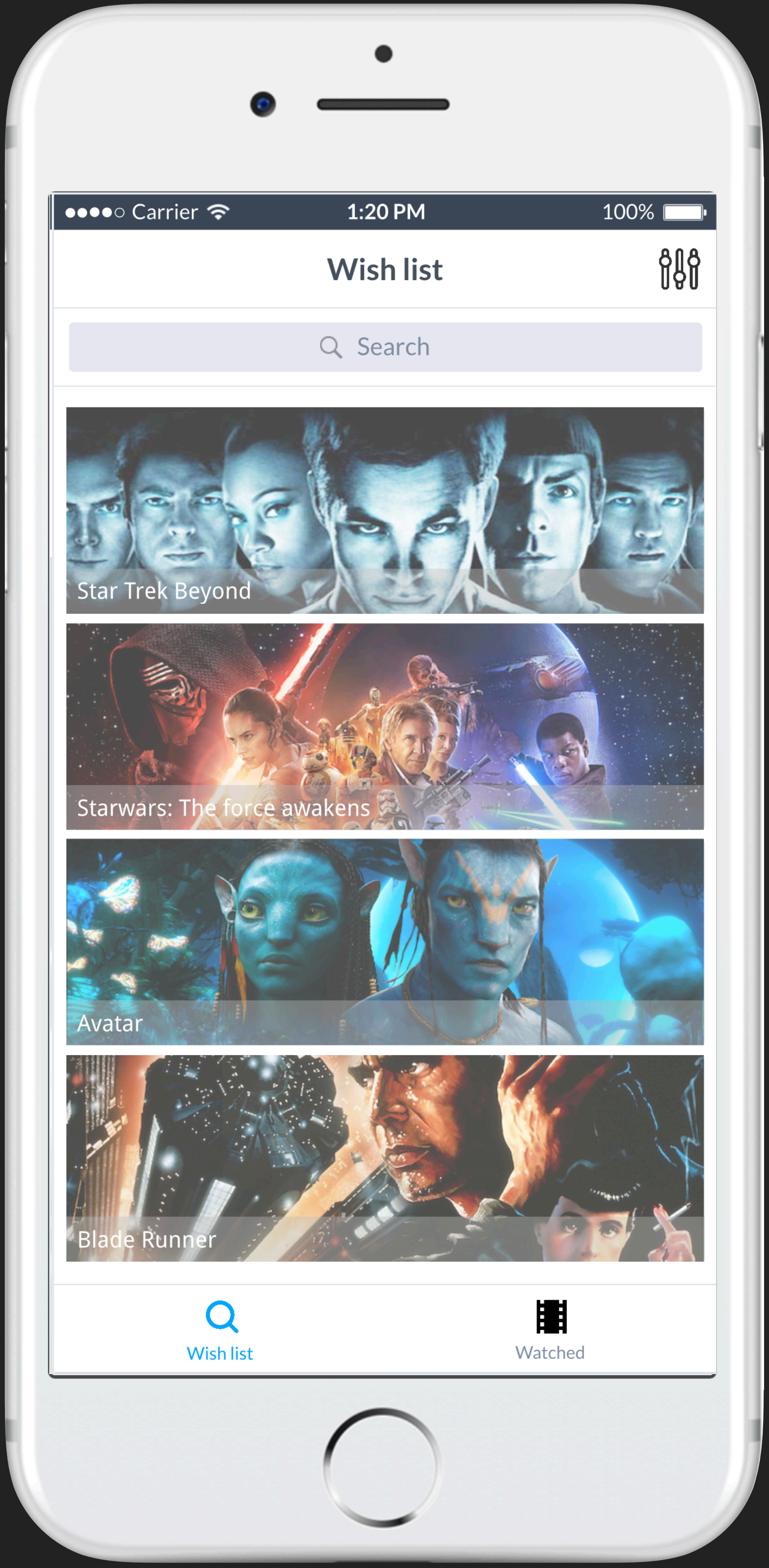
Cast detail - push

Settings - popup

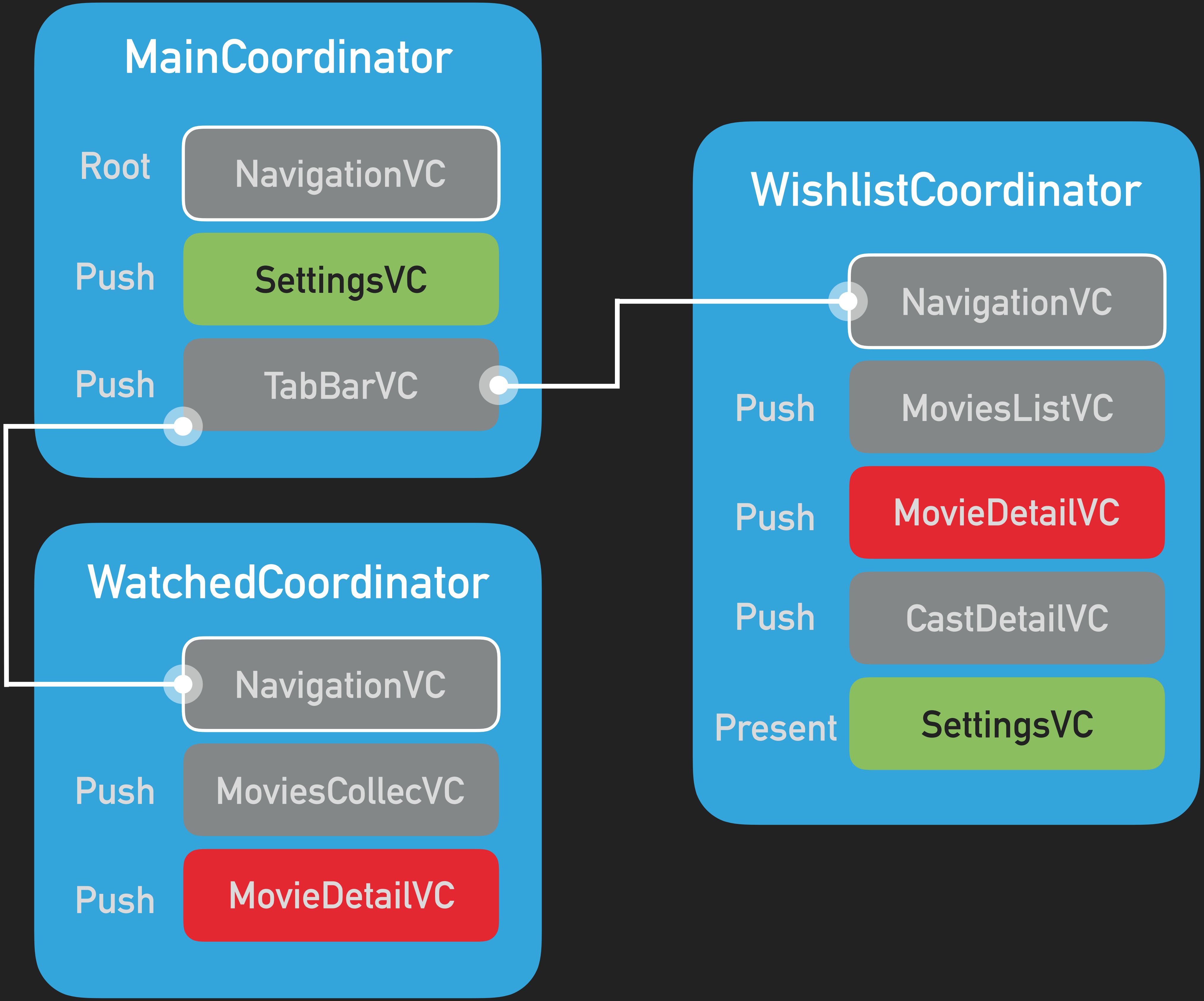
Watched flow - Navigation stack

Movies collection - root

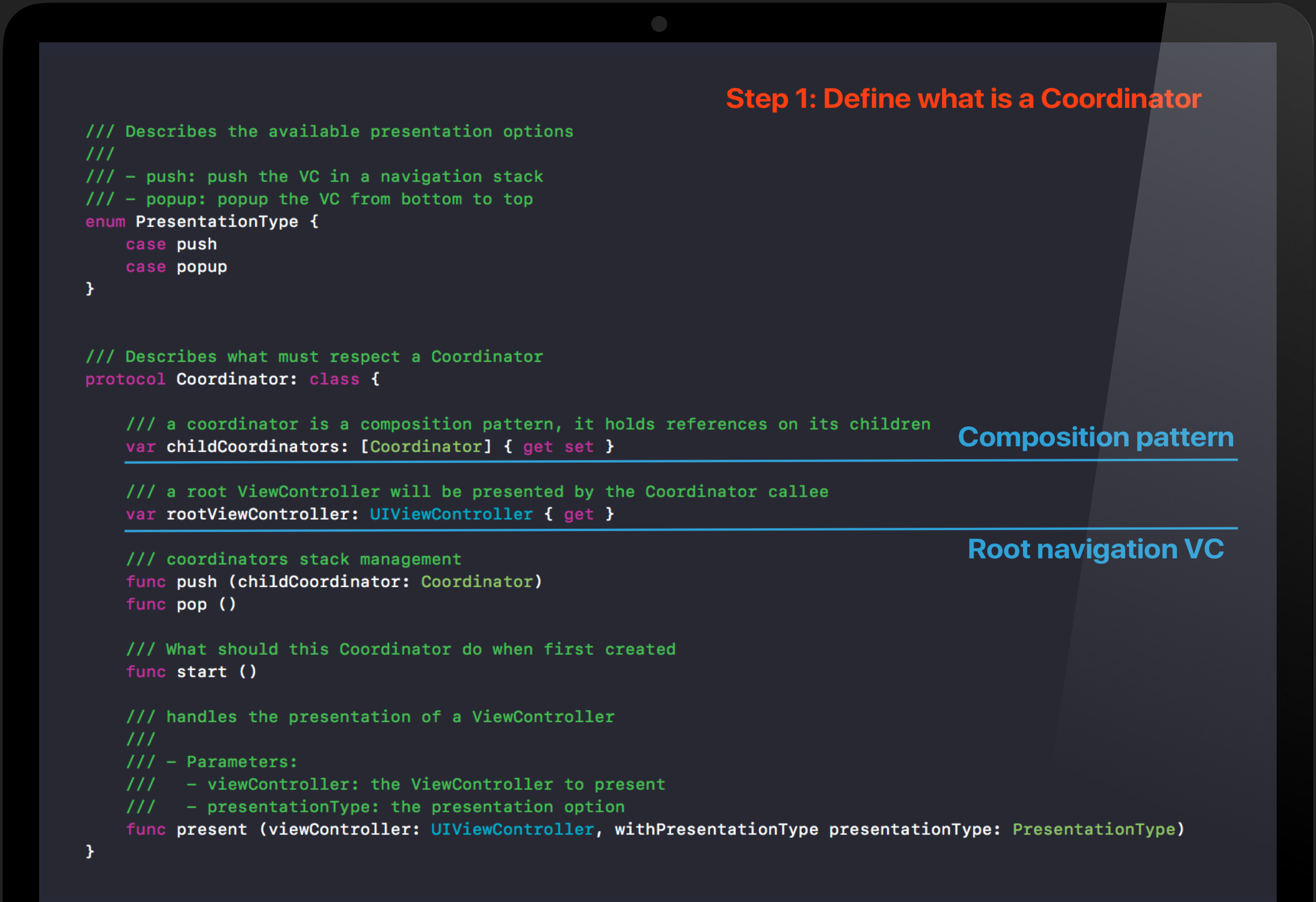
Movie detail - push



LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR



LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR



Step 1: Define what is a Coordinator

```swift
/// Describes the available presentation options
///
/// - push: push the VC in a navigation stack
/// - popup: popup the VC from bottom to top
enum PresentationType {
    case push
    case popup
}


/// Describes what must respect a Coordinator
protocol Coordinator: class {

    /// a coordinator is a composition pattern, it holds references on its children
    var childCoordinators: [Coordinator] { get set }

    /// a root ViewController will be presented by the Coordinator callee
    var rootViewController: UIViewController { get }

    /// coordinators stack management
    func push (childCoordinator: Coordinator)
    func pop ()

    /// What should this Coordinator do when first created
    func start ()

    /// handles the presentation of a ViewController
    ///
    /// - Parameters:
    ///   - viewController: the ViewController to present
    ///   - presentationType: the presentation option
    func present (viewController: UIViewController, withPresentationType presentationType: PresentationType)
}
```

Composition pattern

Root navigation VC

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR

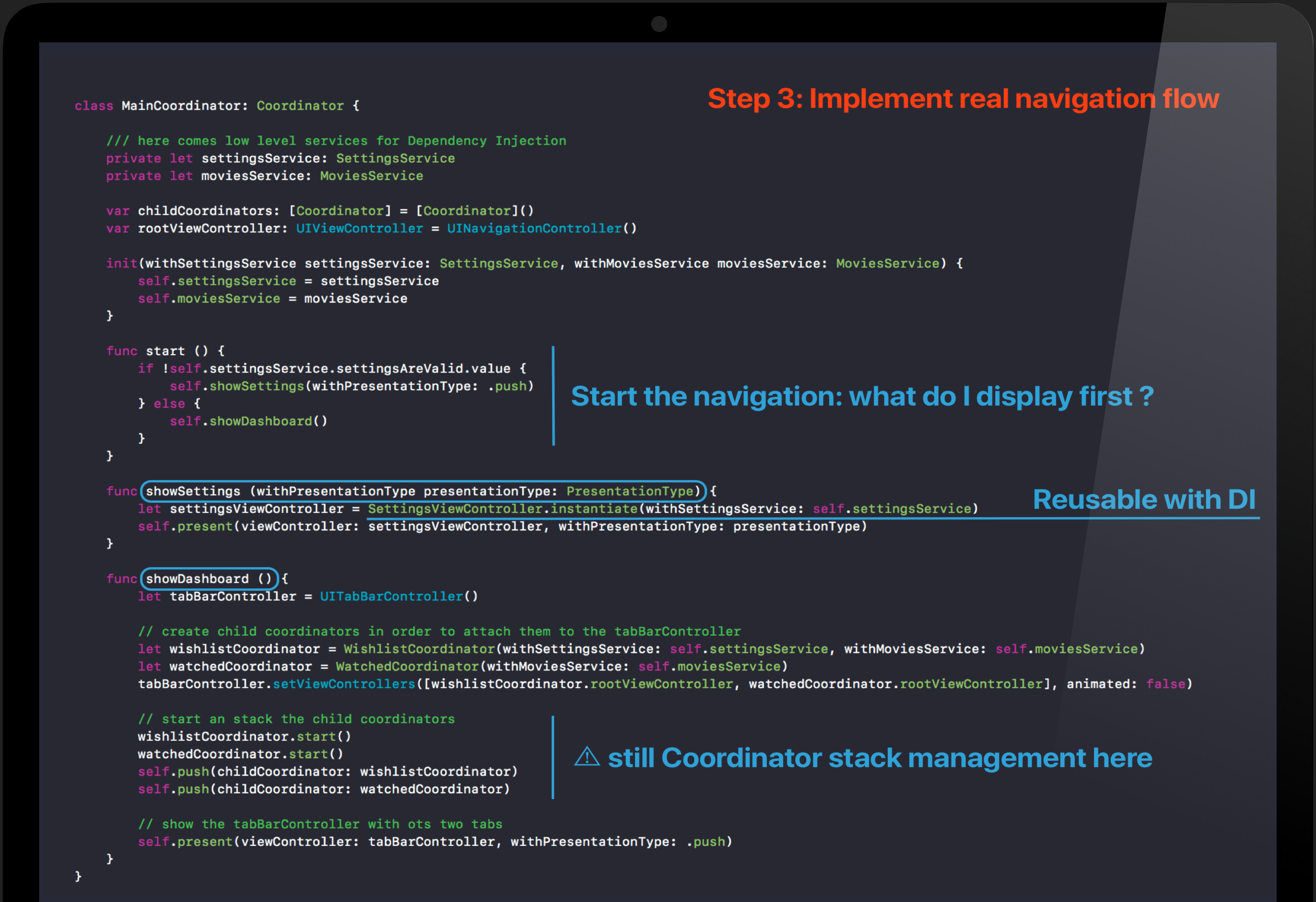
Step 2: Implement a default navigation stack management and a VCs presentation function

```
extension Coordinator {
    func push (childCoordinator: Coordinator) {
        self.childCoordinators.append(childCoordinator)
    }

    func pop () {
        self.childCoordinators.removeLast()
    }

    func present (viewController: UIViewController, withPresentationType presentationType: PresentationType) {
        switch presentationType {
        case .popup:
            viewController.modalPresentationStyle = .overFullScreen
            viewController.modalTransitionStyle = .coverVertical
            self.rootViewController.present(viewController, animated: true)
        case .push:
            self.rootViewController.show(viewController, sender: nil)
        }
    }
}
```

Coordinators stack

VCs Presentation

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR



Step 3: Implement real navigation flow

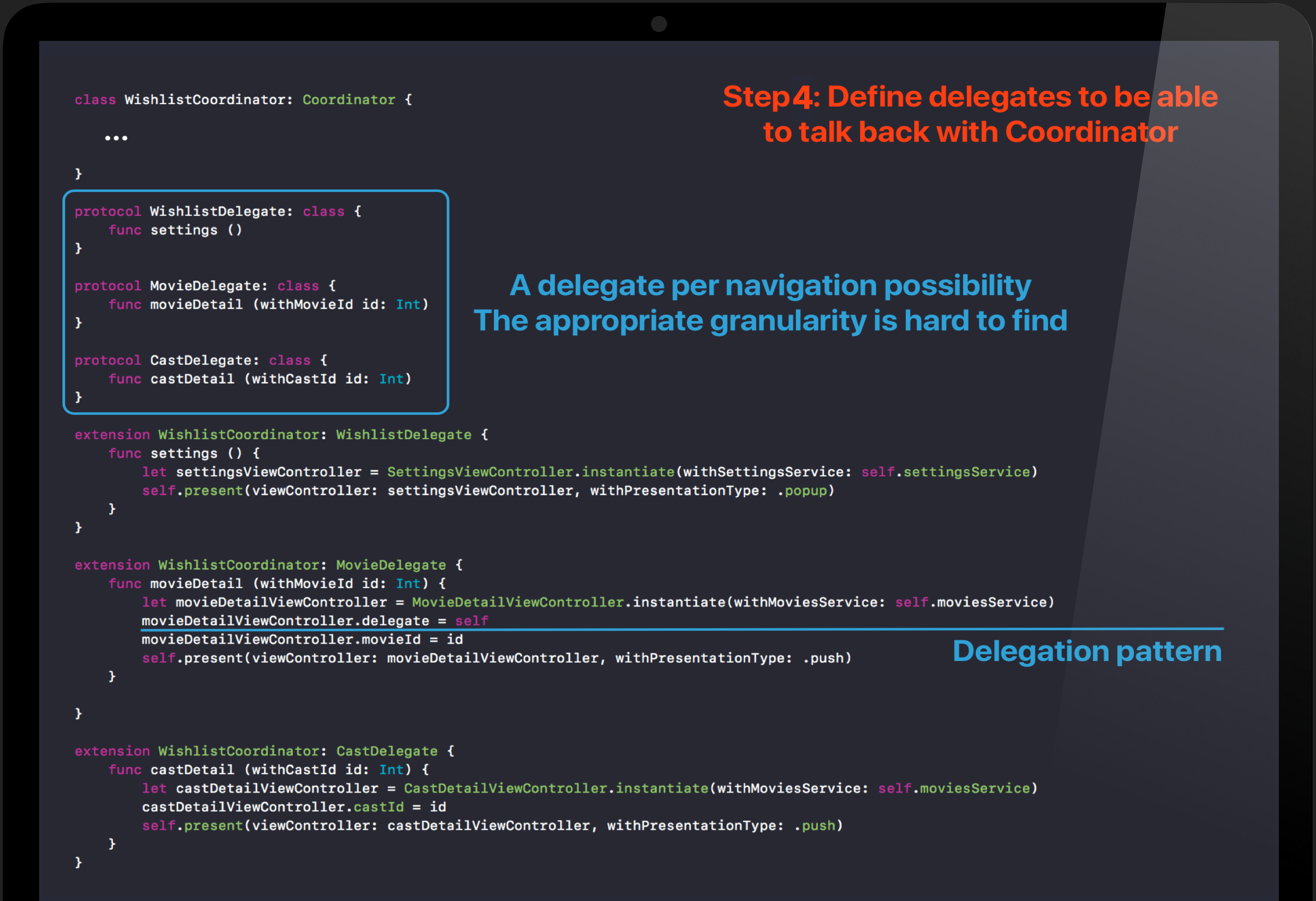
```swift
class MainCoordinator: Coordinator {

    /// here comes low level services for Dependency Injection
    private let settingsService: SettingsService
    private let moviesService: MoviesService

    var childCoordinators: [Coordinator] = [Coordinator]()
    var rootViewController: UIViewController = UINavigationController()

    init(withSettingsService settingsService: SettingsService, withMoviesService moviesService: MoviesService) {
        self.settingsService = settingsService
        self.moviesService = moviesService
    }

    func start () {
        if !self.settingsService.settingsAreValid.value {
            self.showSettings(withPresentationType: .push)
        } else {
            self.showDashboard()
        }
    }

    func showSettings (withPresentationType presentationType: PresentationType) {
        let settingsViewController = SettingsViewController.instantiate(withSettingsService: self.settingsService)
        self.present(viewController: settingsViewController, withPresentationType: presentationType)
    }

    func showDashboard () {
        let tabBarController = UITabBarController()

        // create child coordinators in order to attach them to the tabBarController
        let wishlistCoordinator = WishlistCoordinator(withSettingsService: self.settingsService, withMoviesService: self.moviesService)
        let watchedCoordinator = WatchedCoordinator(withMoviesService: self.moviesService)
        tabBarController.setViewControllers([wishlistCoordinator.rootViewController, watchedCoordinator.rootViewController], animated: false)

        // start an stack the child coordinators
        wishlistCoordinator.start()
        watchedCoordinator.start()
        self.push(childCoordinator: wishlistCoordinator)
        self.push(childCoordinator: watchedCoordinator)

        // show the tabBarController with ots two tabs
        self.present(viewController: tabBarController, withPresentationType: .push)
    }
}
```

Start the navigation: what do I display first ?

Reusable with DI

⚠ still Coordinator stack management here

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR



Step 4: Define delegates to be able to talk back with Coordinator

```swift
class WishlistCoordinator: Coordinator {

    ...

}

protocol WishlistDelegate: class {
    func settings ()
}

protocol MovieDelegate: class {
    func movieDetail (withMovieId id: Int)
}

protocol CastDelegate: class {
    func castDetail (withCastId id: Int)
}

extension WishlistCoordinator: WishlistDelegate {
    func settings () {
        let settingsViewController = SettingsViewController.instantiate(withSettingsService: self.settingsService)
        self.present(viewController: settingsViewController, withPresentationType: .popup)
    }
}

extension WishlistCoordinator: MovieDelegate {
    func movieDetail (withMovieId id: Int) {
        let movieDetailViewController = MovieDetailViewController.instantiate(withMoviesService: self.moviesService)
        movieDetailViewController.delegate = self
        movieDetailViewController.movieId = id
        self.present(viewController: movieDetailViewController, withPresentationType: .push)
    }

}

extension WishlistCoordinator: CastDelegate {
    func castDetail (withCastId id: Int) {
        let castDetailViewController = CastDetailViewController.instantiate(withMoviesService: self.moviesService)
        castDetailViewController.castId = id
        self.present(viewController: castDetailViewController, withPresentationType: .push)
    }
}
```

A delegate per navigation possibility
The appropriate granularity is hard to find

Delegation pattern

LET'S WEAVE YOUR APPLICATION: FLOW COORDINATOR

Step 5: Talk back with my delegate to tell him my new state

```
class MovieListViewController: UIViewController, StoryboardBased {

    public weak var delegate: MovieDelegate!
    public var movieId: Int!

     ...

    self.delegate.movieDetail (withMovieId: 2)

    ...

}
```

In a @IBAction or a didSelectRowAt

3 MORE ACHIEVEMENTS

- Ease the implementation of Dependency Injection
- Remove navigation code from VCs
- Cut our application into logical units of navigation

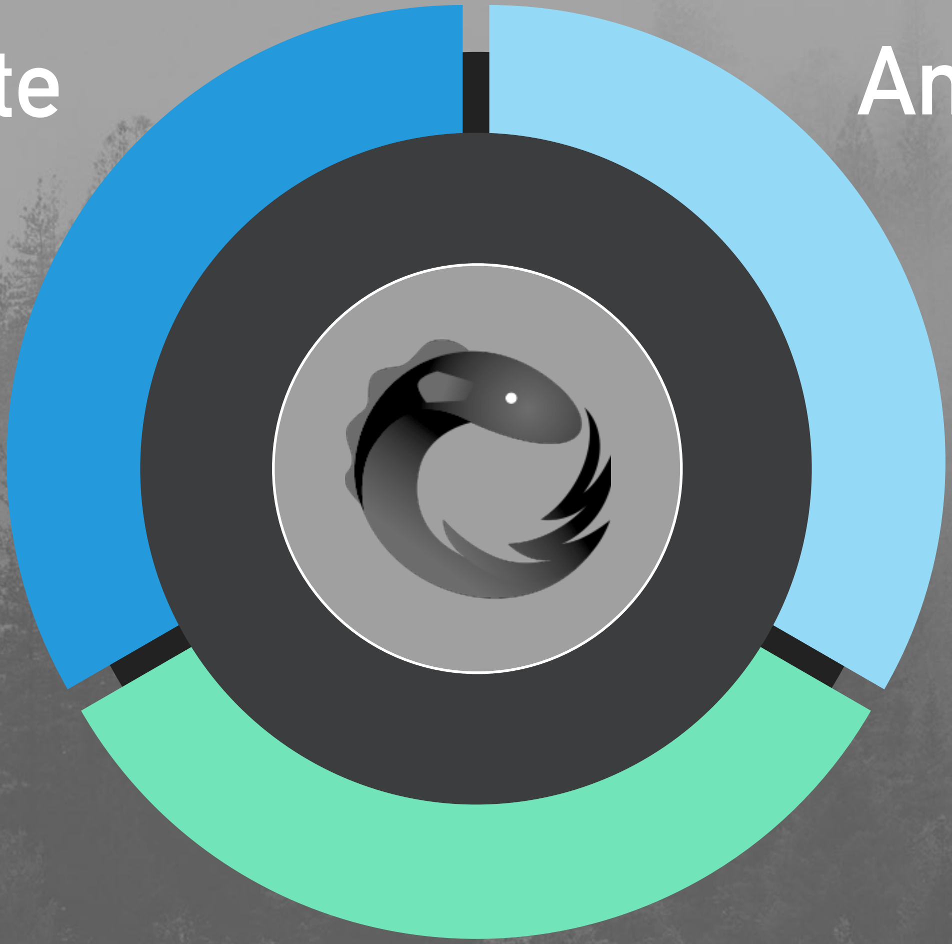
Boring repetitive code

Still some boilerplate code (delegation)

STEP 3

REACTIVE PROGRAMMING

LET'S WEAVE YOUR APPLICATION: REACTIVE PROGRAMMING



Easy to expose states and propagate state changes

Avoid delegation pattern And Notification Center

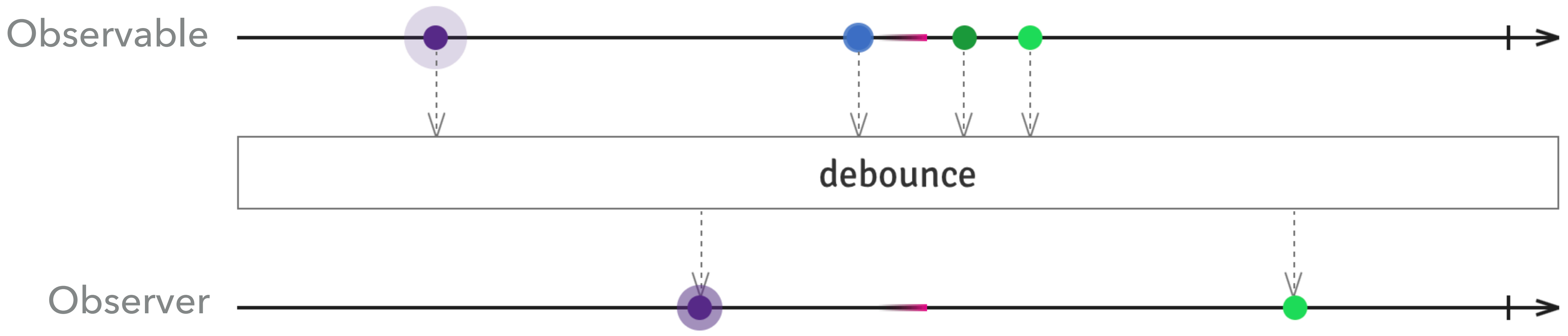
Adopted by many developers

Fits well with MVVM for instance, which I like

LET'S WEAVE YOUR APPLICATION: REACTIVE PROGRAMMING

The Observer pattern done right

ReactiveX is a combination of the best ideas from
the Observer pattern, the Iterator pattern, and functional programming



CREATE

Easily create event streams or data streams.

COMBINE

Compose and transform streams with query-like operators.

LISTEN

Subscribe to any observable stream to perform side effects.

1 ESSENTIAL ACHIEVEMENT

- Promote Reactive Programming

SOMETHING IS BEGINNING TO EMERGE !

We know how to cut StoryBoards and reuse ViewControllers (Reusable)

●

We know how to orchestrate navigation and isolate navigation code from Views (Coordinator)

●

We know how to express and propagate a change of state (Reactive)

ONLY 1 ACHIEVEMENT LEFT

- Describe the navigation in a more declarative way

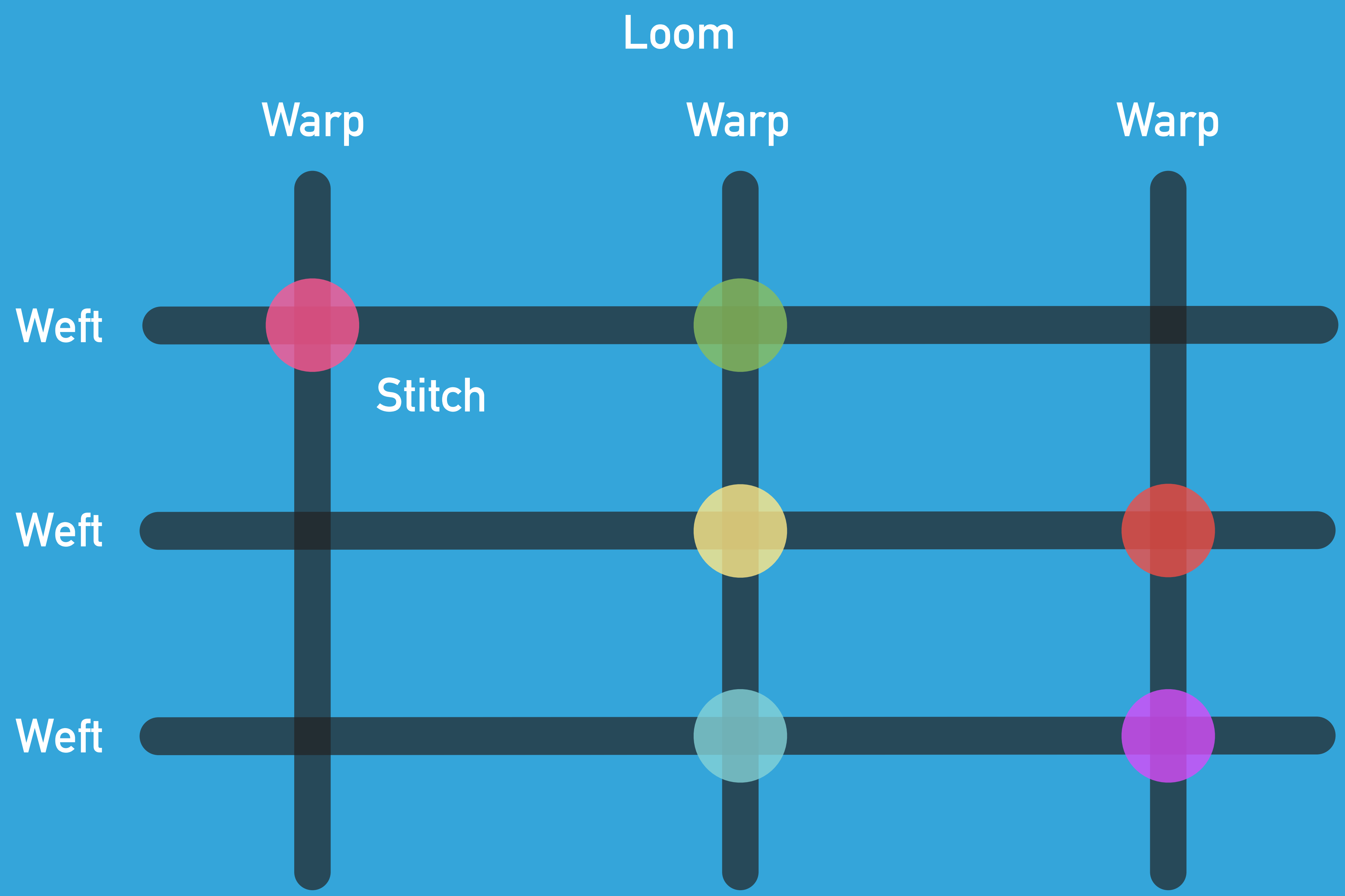
FINAL STEP: WEAVY

THE WEAVING PATTERN

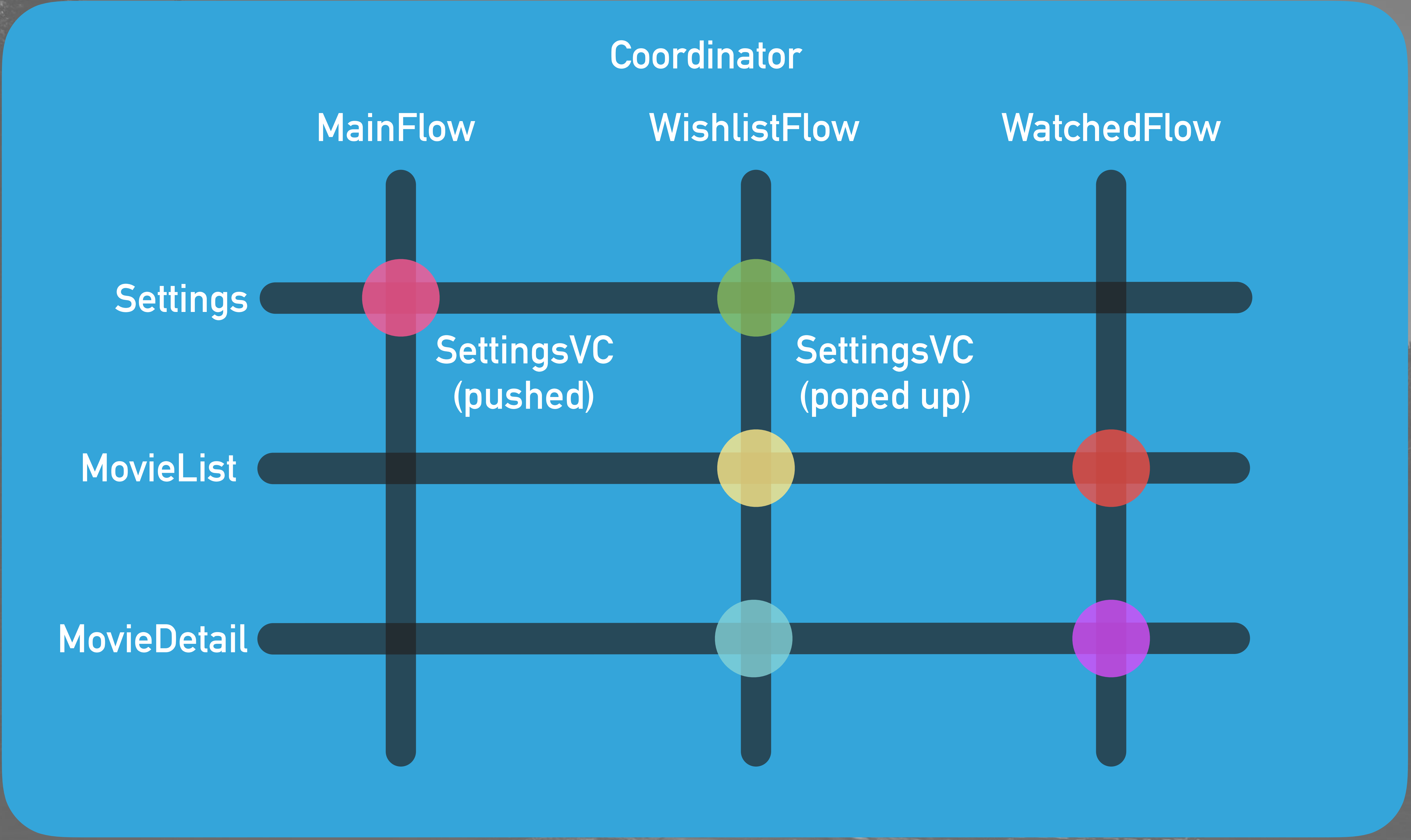
« WEAVING INVOLVES USING A LOOM TO INTERLACE TWO SETS OF THREADS AT RIGHT ANGLES TO EACH OTHER: THE **WARP** WHICH RUNS LONGITUDINALLY AND THE **WEFT** THAT CROSSES IT [...] CLOTH IS USUALLY WOVEN ON A **LOOM**, A DEVICE THAT HOLDS THE WARP THREADS IN PLACE WHILE FILLING WEFTS ARE WOVEN THROUGH THEM »

Weaving from Wikipedia

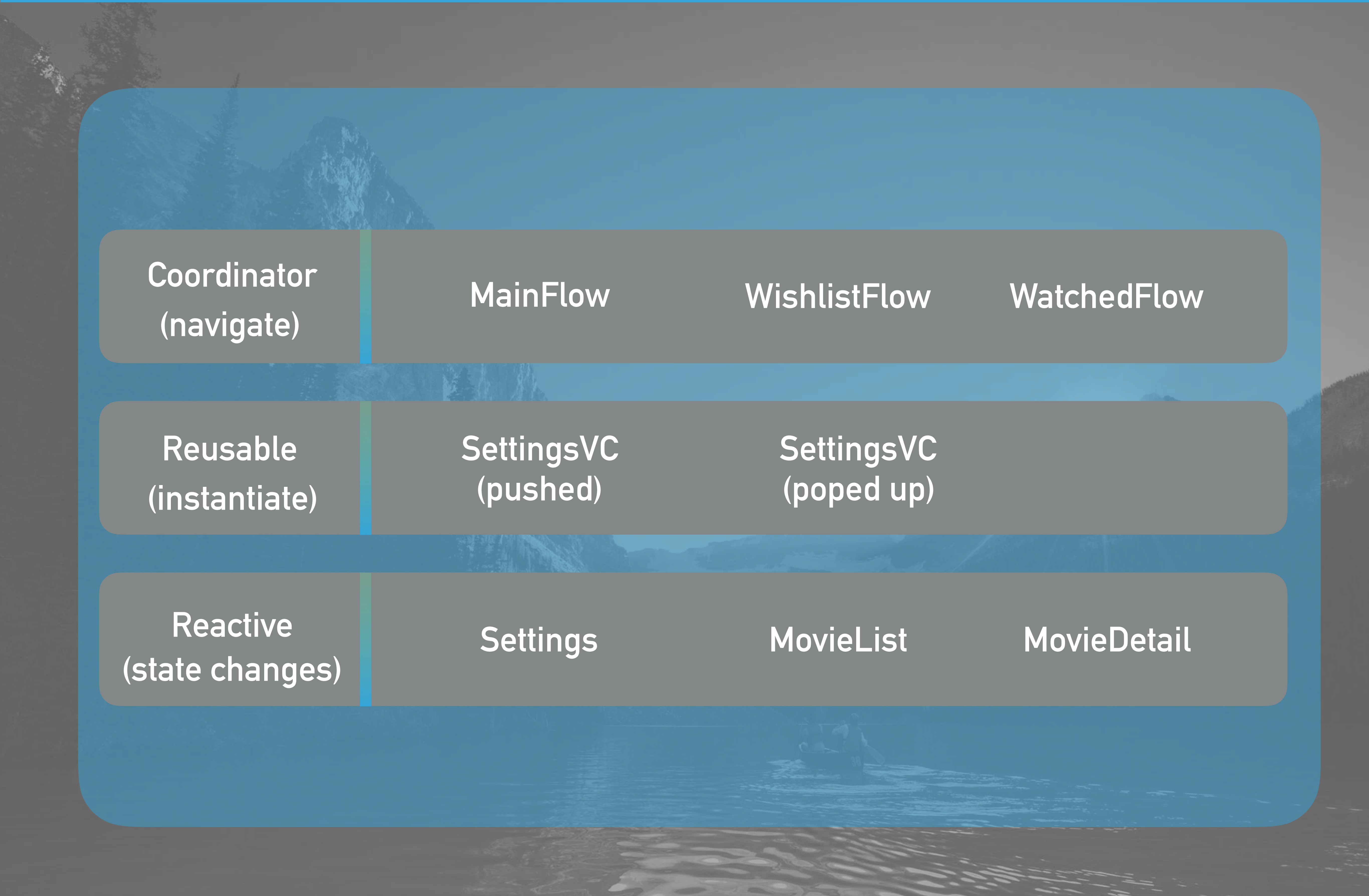
LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN



LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN

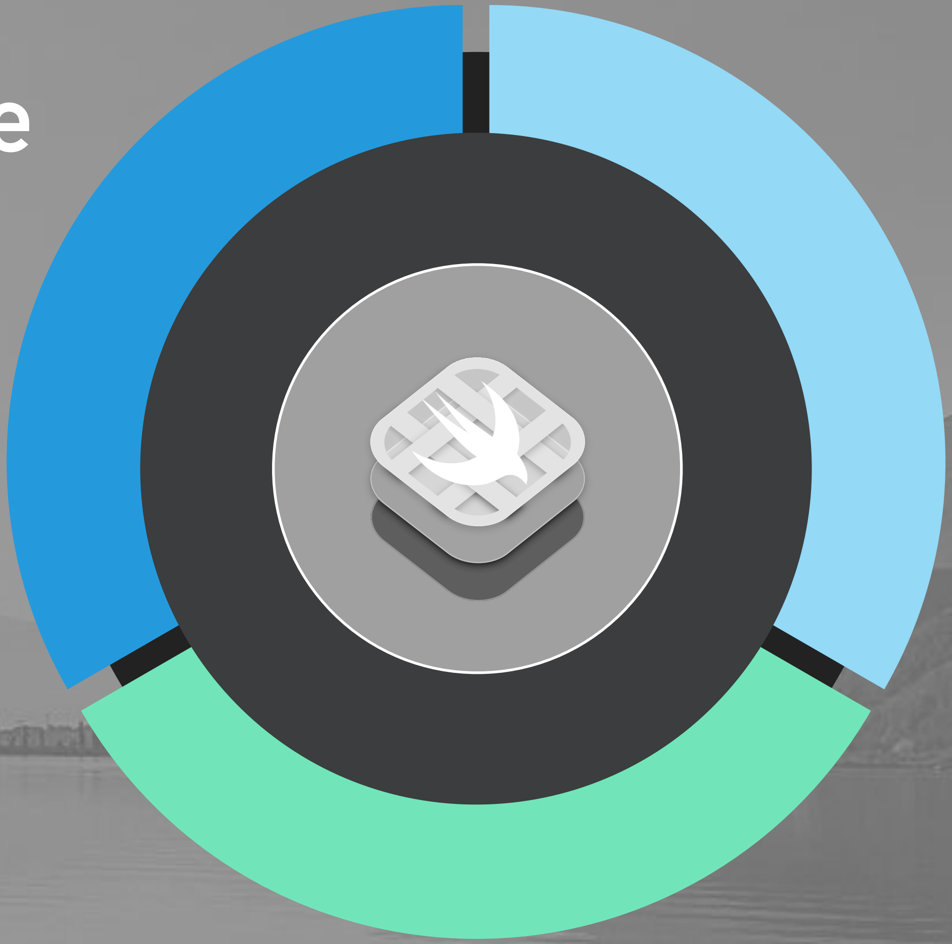


LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN



Coordinator (navigate)	MainFlow	WishlistFlow	WatchedFlow
Reusable (instantiate)	SettingsVC (pushed)	SettingsVC (poped up)	
Reactive (state changes)	Settings	MovieList	MovieDetail

LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN

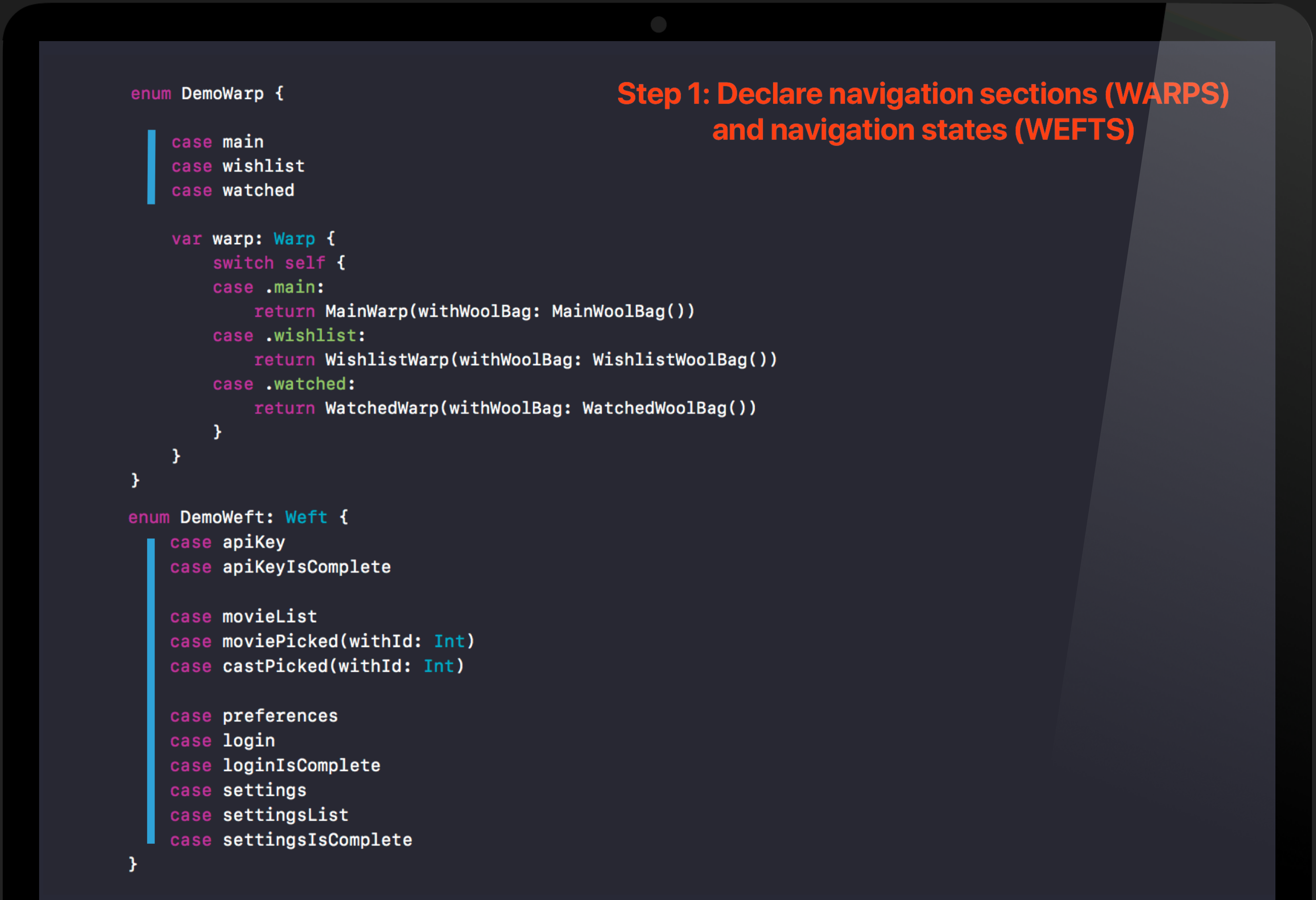


Weavy OpenSource framework (WIP)

Not related to a specific pattern (MVVM, MVP, MVC)

Build on top of the 3 steps seen before but without the boring and boilerplate code

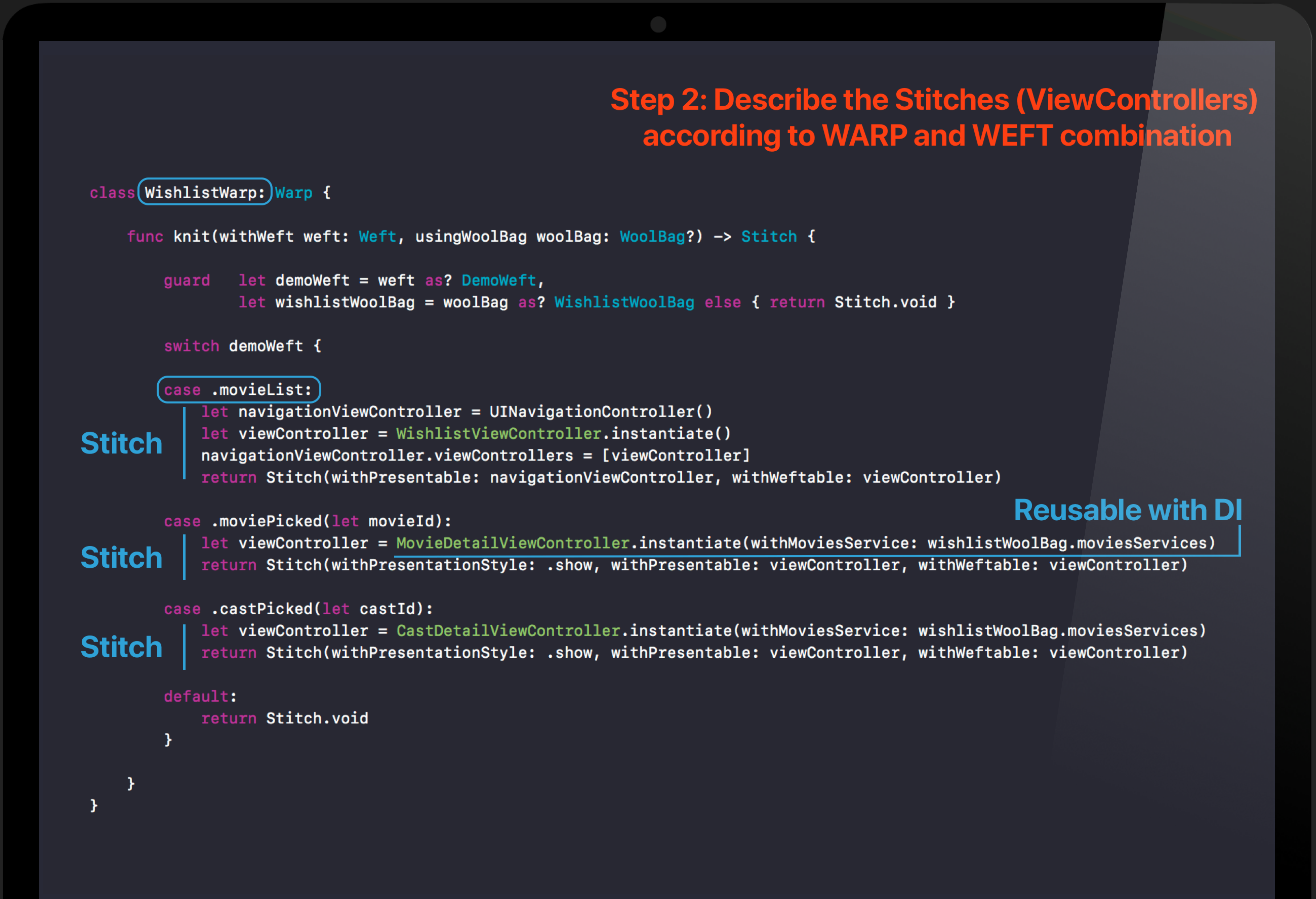
LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN



Step 1: Declare navigation sections (WARPS) and navigation states (WEFTS)

```swift
enum DemoWarp {

    case main
    case wishlist
    case watched

    var warp: Warp {
        switch self {
        case .main:
            return MainWarp(withWoolBag: MainWoolBag())
        case .wishlist:
            return WishlistWarp(withWoolBag: WishlistWoolBag())
        case .watched:
            return WatchedWarp(withWoolBag: WatchedWoolBag())
        }
    }
}

enum DemoWeft: Weft {
    case apiKey
    case apiKeyIsComplete

    case movieList
    case moviePicked(withId: Int)
    case castPicked(withId: Int)

    case preferences
    case login
    case loginIsComplete
    case settings
    case settingsList
    case settingsIsComplete
}
```

LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN



Step 2: Describe the Stitches (ViewControllers) according to WARP and WEFT combination

```swift
class WishlistWarp: Warp {

    func knit(withWeft weft: Weft, usingWoolBag woolBag: WoolBag?) -> Stitch {

        guard   let demoWeft = weft as? DemoWeft,
                let wishlistWoolBag = woolBag as? WishlistWoolBag else { return Stitch.void }

        switch demoWeft {

        case .movieList:
            let navigationViewController = UINavigationController()
            let viewController = WishlistViewController.instantiate()
            navigationViewController.viewControllers = [viewController]
            return Stitch(withPresentable: navigationViewController, withWeftable: viewController)

        case .moviePicked(let movieId):
            let viewController = MovieDetailViewController.instantiate(withMoviesService: wishlistWoolBag.moviesServices)
            return Stitch(withPresentationStyle: .show, withPresentable: viewController, withWeftable: viewController)

        case .castPicked(let castId):
            let viewController = CastDetailViewController.instantiate(withMoviesService: wishlistWoolBag.moviesServices)
            return Stitch(withPresentationStyle: .show, withPresentable: viewController, withWeftable: viewController)

        default:
            return Stitch.void
        }

    }
}
```

LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN

Step 3: Navigation states are propagated as the user plays with the application

```
class WishlistViewController: UIViewController, StoryboardBased, Weftable {

    ...

    self.weftSubject.onNext (DemoWeft.moviePicked(withId: 3))

    ...

}

class MovieDetailViewController: UIViewController, StoryboardBased, Weftable {

    ...

    self.weftSubject.onNext (DemoWeft.castPicked(withId: 2))

    ...

}
```

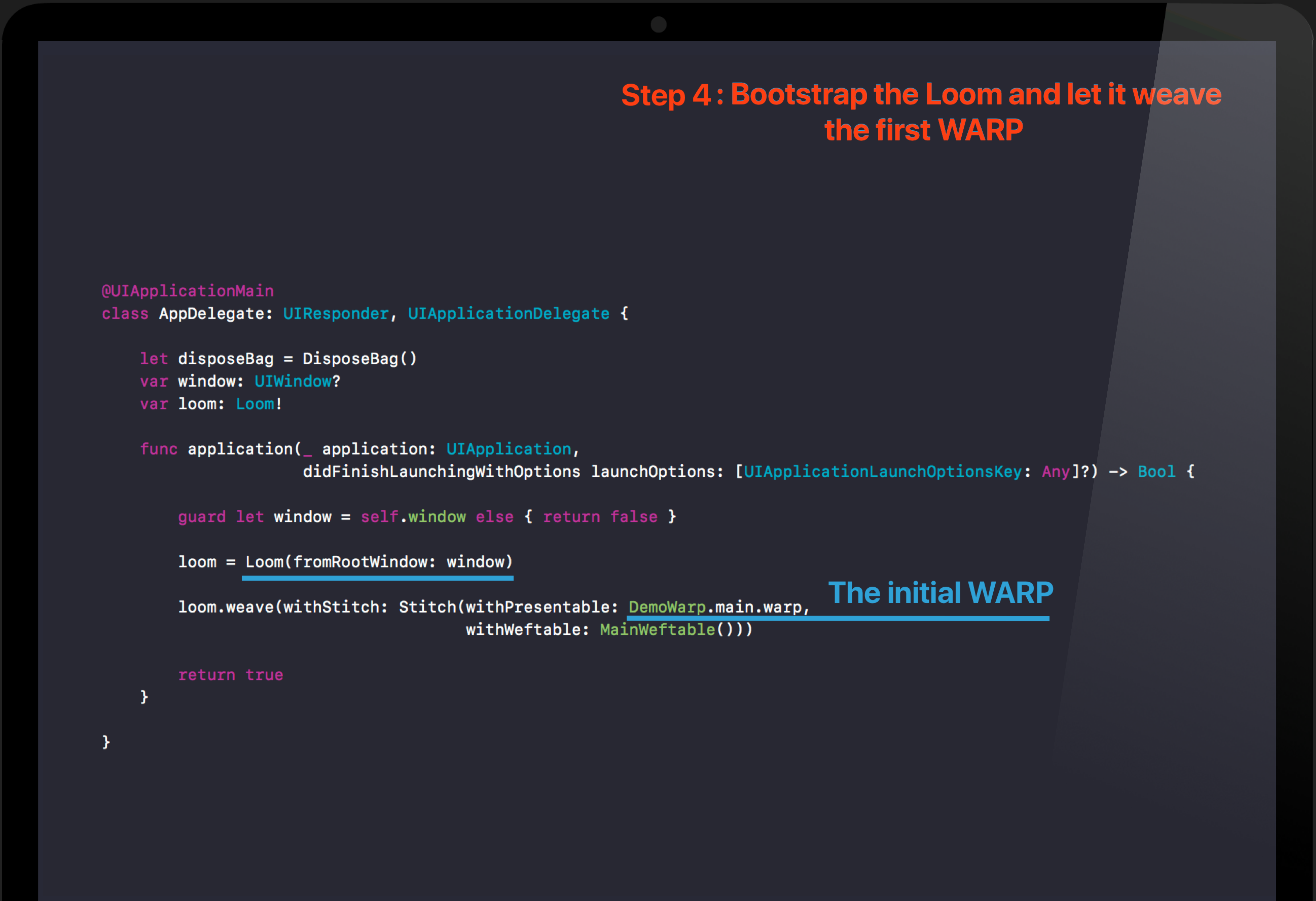
RxSwift

in @IBAction or didSelectRowAt

RxSwift

in @IBAction or didSelectRowAt

LET'S WEAVE YOUR APPLICATION: WEAVY AND THE WEAVING PATTERN



Step 4 : Bootstrap the Loom and let it weave the first WARP

```swift
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    let disposeBag = DisposeBag()
    var window: UIWindow?
    var loom: Loom!

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        guard let window = self.window else { return false }

        loom = Loom(fromRootWindow: window)

        loom.weave(withStitch: Stitch(withPresentable: DemoWarp.main.warp,
                                      withWeftable: MainWeftable()))

        return true
    }

}
```

The initial WARP

THE LAST ACHIEVEMENT

- Describe the navigation in a more declarative way

Weavy uses abstraction and protocols, it does not freeze your inheritance tree

•

Weavy doesn't rely on a centralized navigation state but on a distributed state spread across the application

- **WEAVY FITS WELL IF:**
  - Your application has a complex navigation flow
  - The navigation is dynamic, depending on business rules
  - You are already working with RxSwift
- **WEAVY DOESN'T FITS WELL IF:**
  - You need to do a 2-screen app (would be overkill)
- **GIVE IT A TRY (CONTRIBUTIONS ARE WELCOMED):**
  - Github: https://github.com/twittemb/Weavy
  - Twitter: #thwittem

COCOAHEADS MONTRÉAL 2017/09/21

THANK YOU

QUESTIONS ?