1. QuickBird HackNight

XCoordinator

Encapsulating transition logic with XCoordinator

What you will build

We will refactor a MVC app to encapsulate transition logic and manage inter-scene dependencies with XCoordinator.

What you will learn

What the Coordinator pattern is
How to make inter-scene dependencies clear
How XCoordinator makes code transitions easier
How to use animations in XCoordinator

Prerequisites

- Xcode 10
- Swift
- Patterns Basics

Clone Git repository at:

http://bit.do/qb-x-coordinator

Key Vocabulary

- Coordinator pattern
- XCoordinator
- Transitions
- Animations
- Dependencies
- Swift

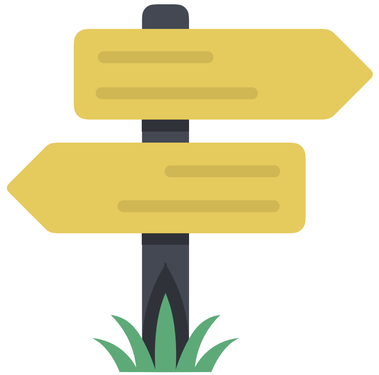
Copyright, Content & Referrals and Links

- **Copyright 2019 QuickBird Studios**
  - Unless explicitly stated otherwise, all rights including those in copyright in the content of this document are owned by or controlled for these purposes by QuickBird Studios.
  - Except as otherwise expressly permitted under copyright law or QuickBird Studios's Terms of Use, the content of this document may not be copied, reproduced, republished, posted, broadcast or transmitted in any way without first obtaining QuickBird Studios's written permission.
- **Content**
  - QuickBird Studios reserves the right not to be responsible for the topicality, correctness, completeness or quality of the information provided.
  - Liability claims regarding damage caused by the use of any information provided, including any kind of information which is incomplete or incorrect, will therefore be rejected.
  - All offers are not-binding and without obligation. Parts of the pages or the complete publication including all offers and information might be extended, changed or partly or completely deleted by the author without separate announcement.
- **Referrals and Links**
  - The author is not responsible for any contents linked or referred to from this document.
  - If any damage occurs by the use of information presented there, only the author of the respective pages might be liable, not the one who has linked to these pages.
  - Furthermore the author is not liable for any postings or messages published by users of discussion boards, guestbooks or mailing lists provided on his page.
- **By obtaining and reading this document, you agree to this copyright statement.**

Agenda

Common practice

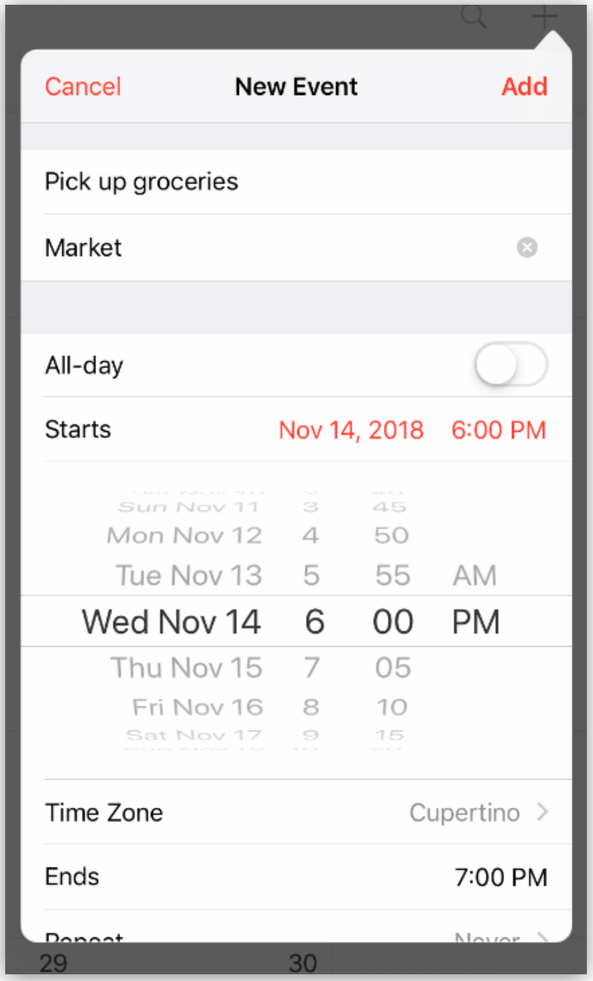
Coordinator pattern

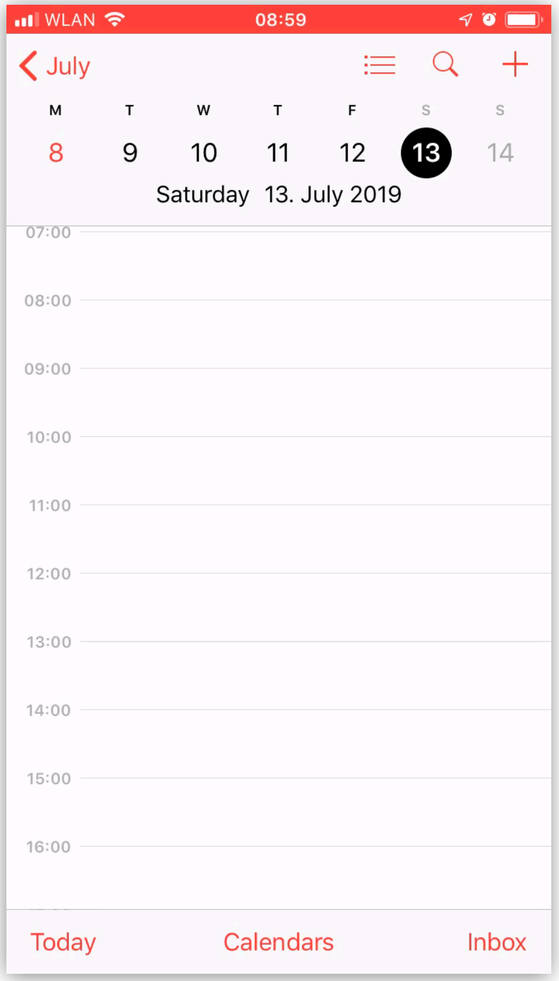


XCoordinator

Live coding

What are transitions?

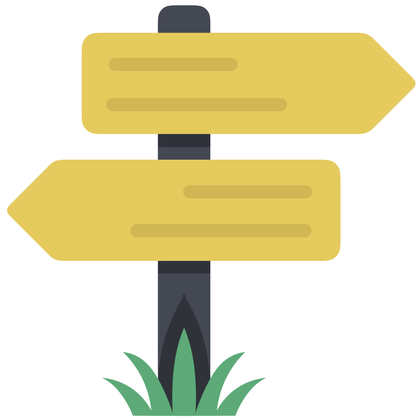




Navigation techniques

Storyboard Segues

UIKit transition code



XCoordinator

Storyboard Segues

```
override func prepare(for segue: UIStoryboardSegue,
                      sender: Any?) {
    switch segue.identifier {
    case "PushNewViewController":
        guard let viewController =
            segue.destination as? NewViewController else {
            return assertionFailure()
        }
        viewController.data = Data()
    default:
        assertionFailure()
    }
}
```

String constants

assumptions about viewController context

viewController-dependent transitions

no type-safety for transition destination

Transitions in code

```
@IBAction func loginButtonTapped(_ sender: UIButton) {
    guard let navigationController = navigationController else {
        return assertionFailure()
    }

    let viewController = NewViewController()
    viewController.data = Data()
    navigationController.pushViewController(viewController, animated: true)
}
```

no String constants

assumptions about viewController context

viewController-dependent transitions

type-safety for transition destination

Transitions using a Coordinator pattern

LoginViewController:

```
@IBAction func loginButtonTapped(_ sender: UIButton) {
    router.trigger(.loginSuccessful)
}
```

LoginCoordinator:

```
override func prepareTransition(for route: LoginRoute) -> NavigationTransition {
    switch route {
    case .loginSuccessful:
        let viewController = NewViewController()
        viewController.data = Data()
        return .push(viewController)
    }
}
```

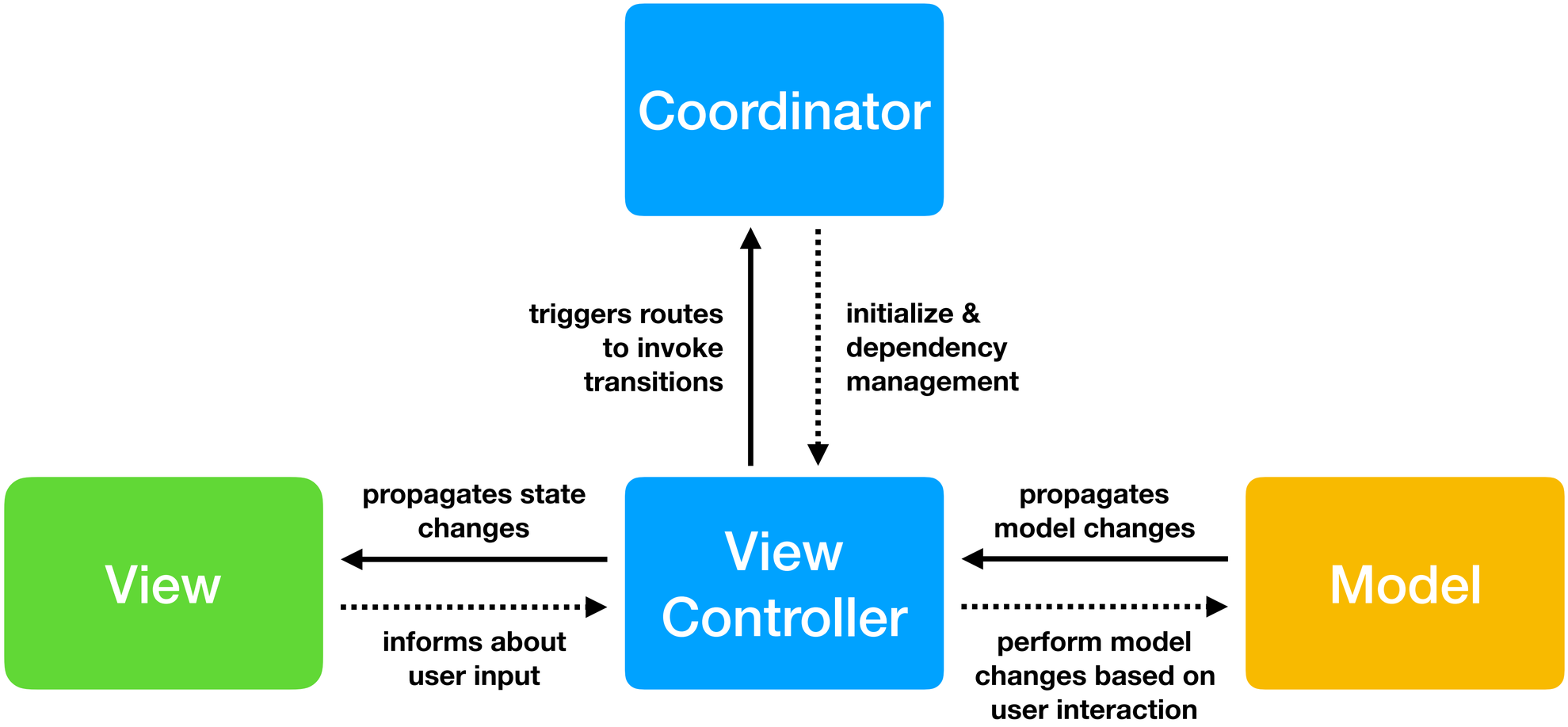
no String constants

no assumptions about viewController context

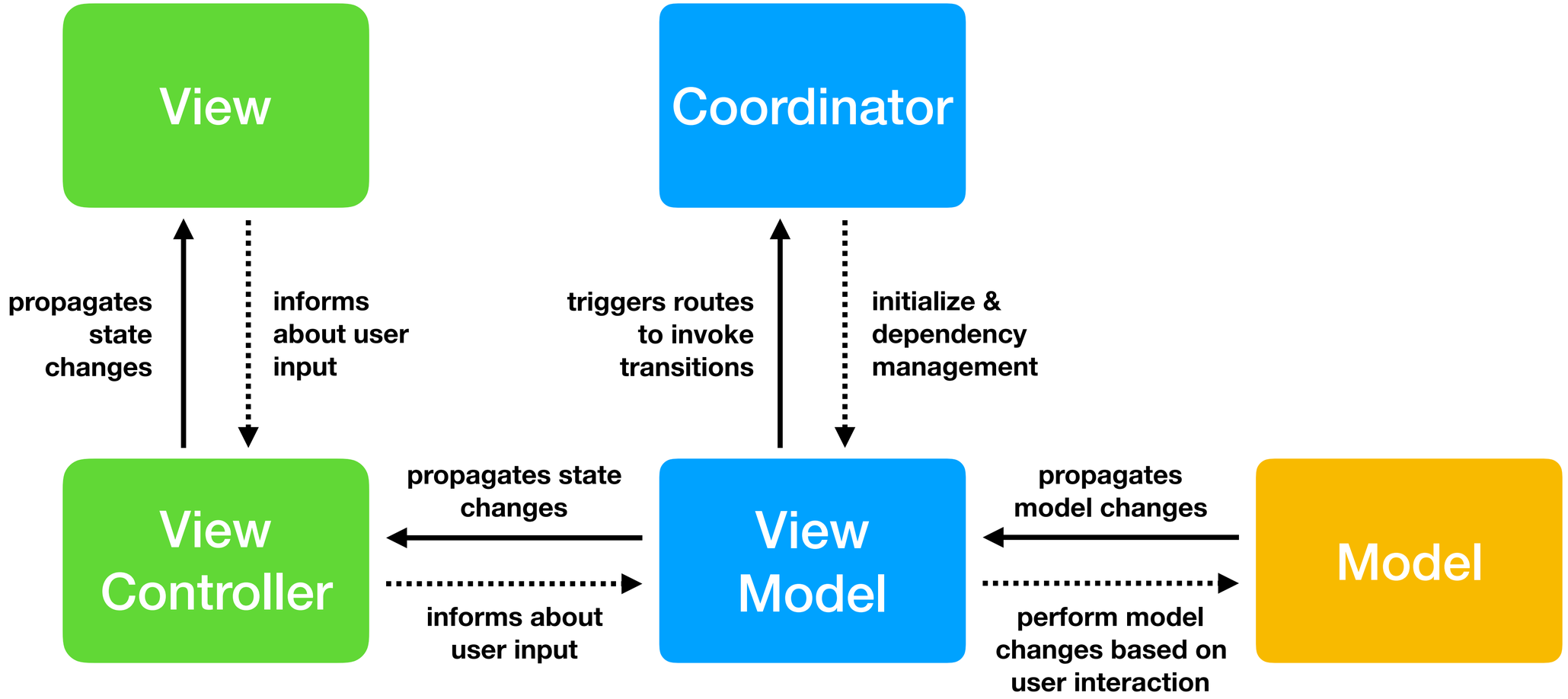
viewController-independent transitions

type-safety for transition destination

Coordinator pattern in MVC



Coordinator pattern in MVVM



Coordinator Pattern?

👍🏻 **Separation of concerns**

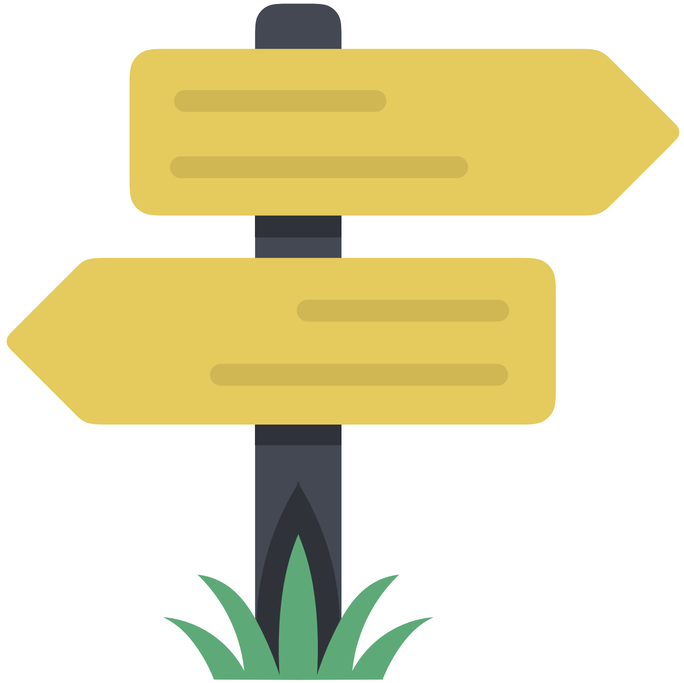
👍🏻 **Flow-independent scenes / view controllers**

👍🏻 **Easily changeable navigation logic**

👎🏻 **Programming overhead**

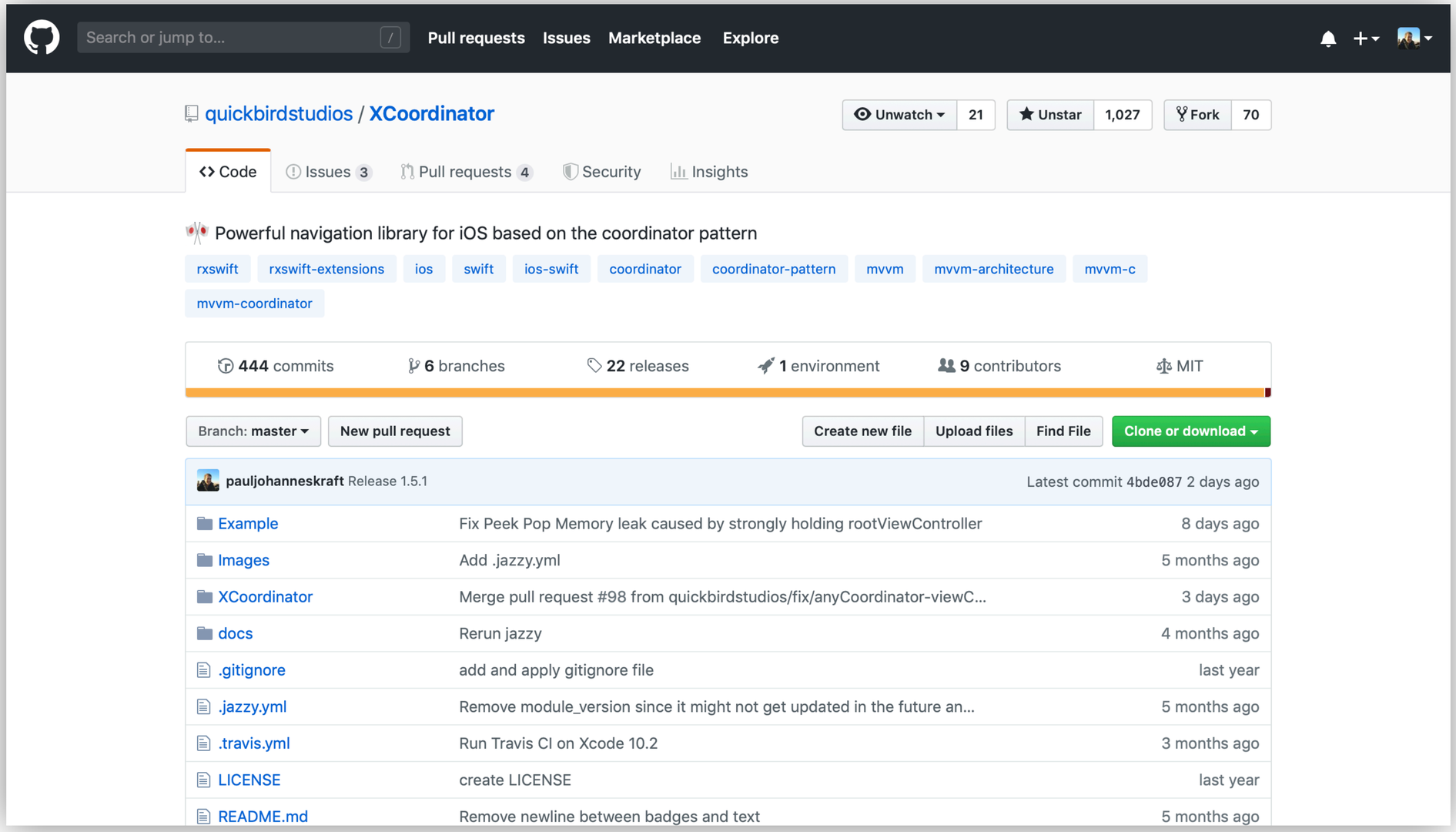
👎🏻 **Not the “standard way”**

And that’s how we got to build…



XCoordinator

XCoordinator is on GitHub!



Apps we built with XCoordinator

soundperience LINOVA SOFTWARE GMBH

Soundfit Pro

ZEPPELIN CATERPILLAR

Operator Challenge

Lufthansa

Design Discovery

XSTUDY

Schultopf

Why use XCoordinator?

🌓 **Many different predefined transitions**

🏃 **Simpler transition animation interface**

🔩 **Generic BaseCoordinator with many provided subclasses**

👨‍✈️ **Fast switching of coordinators without changing viewController code**

🛤 **Deep Linking of routes of different types**

🚀 **RxSwift extensions**

Components in XCoordinator

🎢 Route

- generally an enum
- cases for each possible transition
- New enumeration for each separate flow of your app

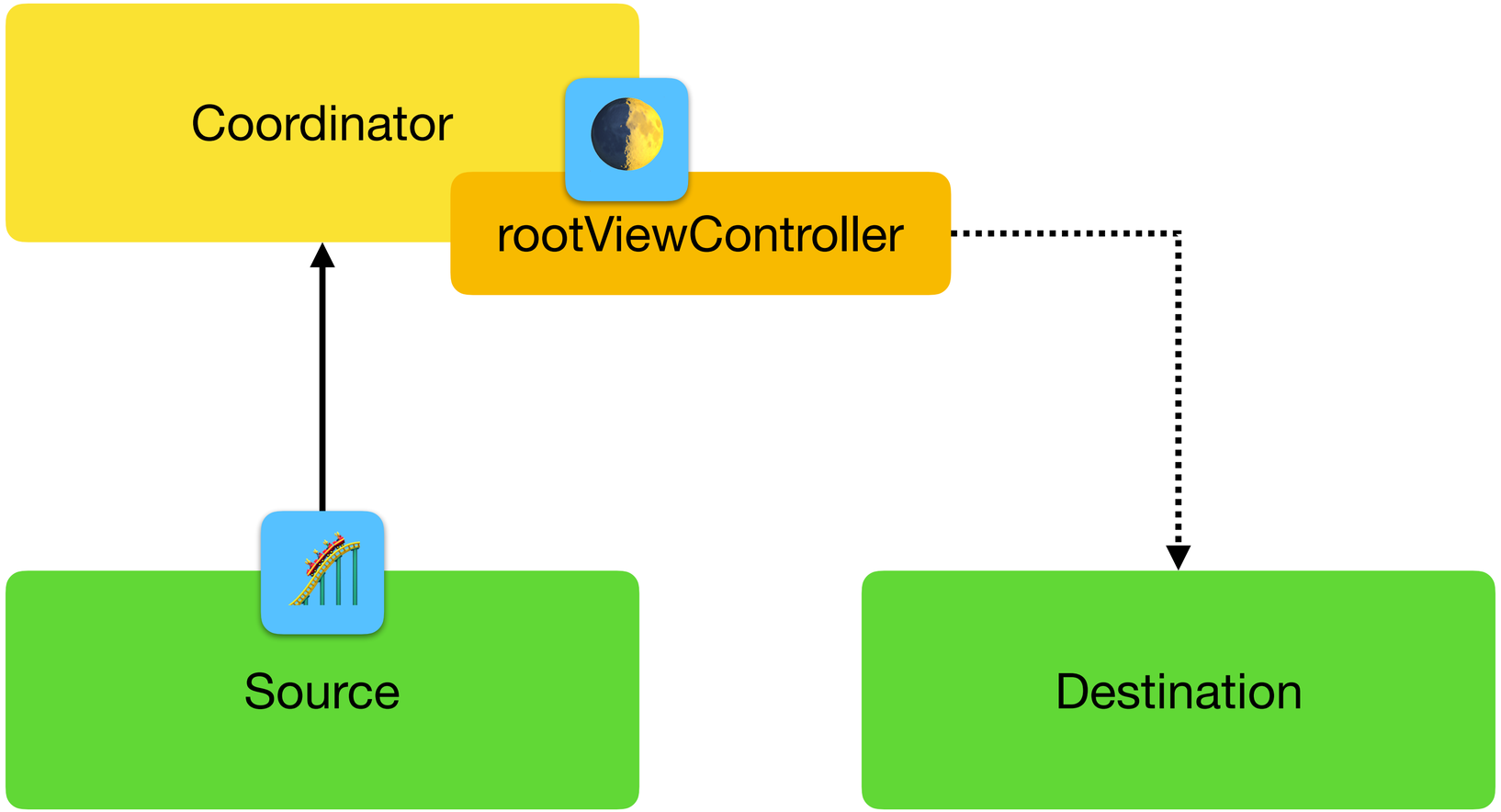
👨‍✈️ Coordinator

- Has a rootViewController
- Prepares transitions for triggered routes by initializing scenes
- performs transitions between scenes

🌓 Transition

- describe navigation from one scene to another
- rootViewController-specific

How routes are triggered



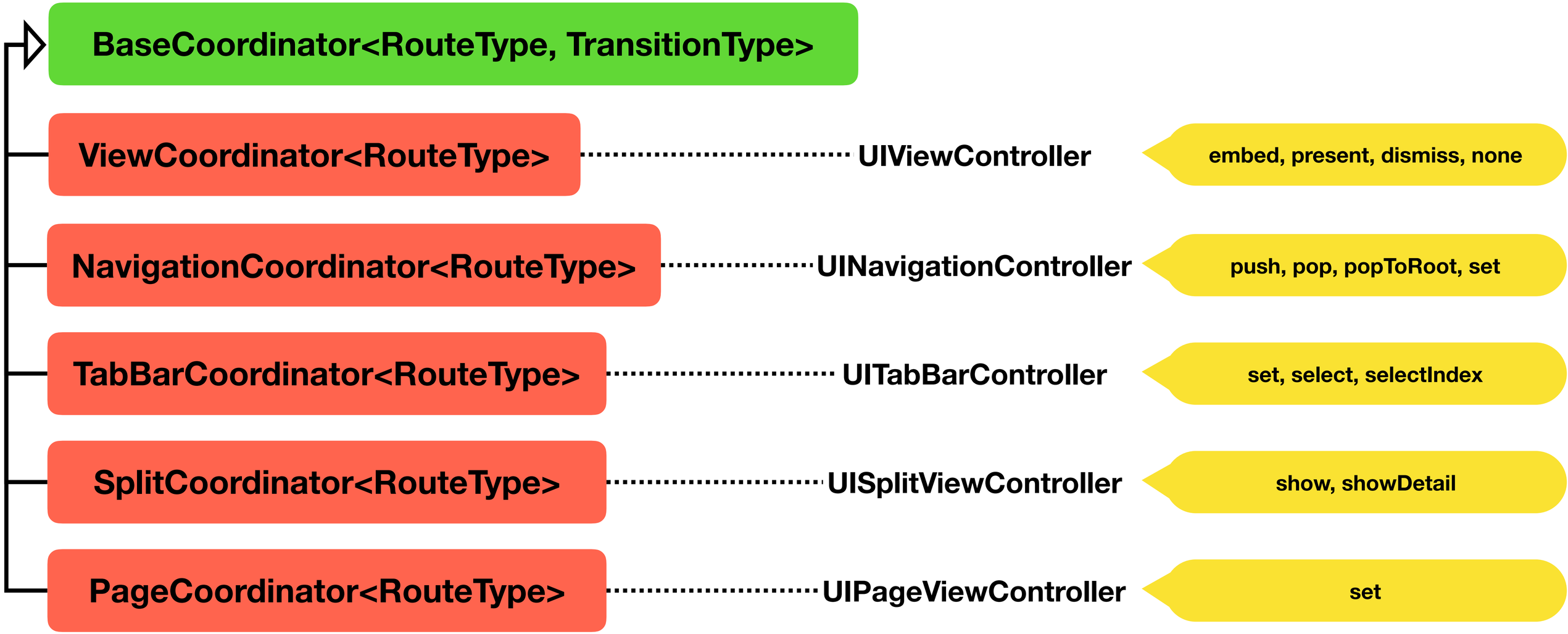
What is a Coordinator?

- **init(initialRoute:)**
- **trigger(_:with:completion:)**
- **prepareTransition(for:)**
- **generateRootViewController()**

What is a Coordinator?

- **init(initialRoute:)**
  - You can trigger a route right at initialisation time
  - Depending on the Coordinator type, there might be other options
- **trigger(_:with:completion:)**
  - Triggers a route, options can define to (not) animate the transition
- **prepareTransition(for:)**
  - Prepares transitions for a given route - might not be the same for different coordinator implementations of the same route
- **generateRootViewController()**
  - Creates the rootViewController for your coordinator - if you want to specify a custom rootViewController make sure to do this here, since it is read-only afterwards

Coordinator & Transition types



Let’s integrate XCoordinator in our app!

QB HackNight

Introduce XCoordinator to your app

- **Create Route enum cases** for all possible segues or transition code segments
- **Add AnyRouter** of the created Route-enum to the viewControllers handling transitions and replace the transition code / segues with triggering of routes
- **Implement a Coordinator** by overriding the prepareTransition(for:) method
- Make sure to **use the Coordinator** 😉

Exercise: Integrating XCoordinator

Task 1: Create LoginCoordinator & use as initial coordinator

Task 2: Create HomeCoordinator as TabBarCoordinator

QB HackNight

Solution 1: LoginCoordinator

Create **LoginRoute.swift** in **Routing**

- Create a LoginRoute enum with one case for each possible transition of that flow
- Introduce an `AnyRouter<LoginRoute>` into `LoginViewController` and replace transition logic with triggering routes
- Create a LoginCoordinator as a ViewCoordinator
  - Override `generateRootViewController` to create a `LoginViewController` as the coordinator's rootViewController
  - Override `prepareTransition(for:)` to prepare transitions for the given routes
  - Create an empty initializer to make sure, the coordinator is always correctly initialized
- Use `coordinator.setRoot` to set the rootViewController of the app's window
- Try it out! Check, if everything still works.

Solution 2: HomeCoordinator

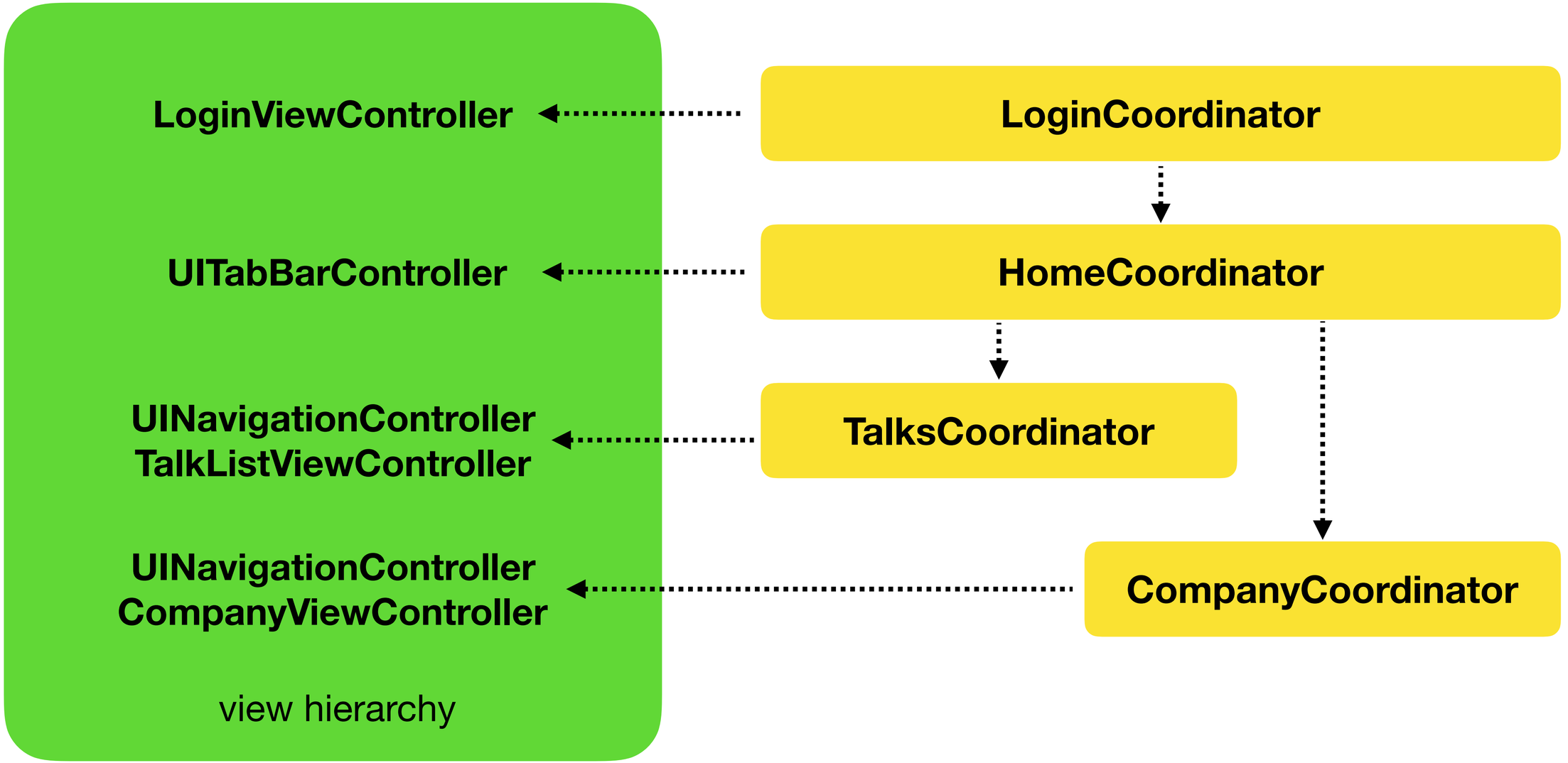
Create **HomeRoute.swift** in **Routing**

- Identify `HomeCoordinator` as a useful abstraction
- Create a `HomeRoute` with only an `initial` route, since there are no interactions possible
- Implement `HomeCoordinator` as a `TabBarCoordinator`
  - Create empty initializer to make sure it is always correctly initialized
  - Override `prepareTransition(for:)` to provide transitions for triggered routes
- Change `prepareTransition(for:)` in `LoginCoordinator` to present the coordinator instead of creating viewControllers

What did we achieve? 🎉

- Abstraction from transition logic & individual scenes/viewControllers
  - You can easily swap out, which transition is performed, when a route is triggered without the need of changing a viewController —> higher reusability
- Type-safe transitions (You cannot trigger a push transitions on a UITabBarController)
- Simpler transition animation interface
- Creation of scenes and connected model data at one place
  - You can also pass data using associated values in enum cases, if you want/need to pass data

Coordinator hierarchies



When do I need a new Route/Coordinator?

- **Rule of thumb:** new rootViewController to perform transitions on
- New Context
  - creation/editing of a model element
  - tabs in a UITabBarController that have distinct features
- Presentation of new Scene
- Encapsulation of behavior —> restricting access of certain scenes to specific routes (see **RedirectionRouter/RedirectionCoordinator**)

Setup for Exercise

- Log into WiFi:

SSID: **“Quickbird Guests”**

Password: **“QuickBirdStudios!”**

- Clone the following repository using Git (or download as .zip):

http://bit.do/qb-x-coordinator

- Open 

HackNight-XCoordinator.xcworkspace

Exercise: Finishing up integration

Task 1 (TalksRoute.swift): Introduce a TalksCoordinator

Task 2 (HomeRoute.swift): Make sure to highlight the Company-tab when routing to the Home route

Task 3 (CompanyRoute.swift): Introduce an CompanyCoordinator

Task 4 (LoginRoute.swift): Add login input validation

Solution 1.1 - Slide 1/2

- Create **TalksRoute** in **TalksRoute.swift**

```
enum TalksRoute: Route {
    case initial
}
```

Solution 1.1 - Slide 2/2

- Create **TalksCoordinator** in **TalksRoute.swift**

```
class TalksCoordinator: NavigationCoordinator<TalksRoute> {

    init() {
        super.init(initialRoute: .initial)
    }

    override func prepareTransition(for route: TalksRoute) -> NavigationTransition {
        switch route {
        case .initial:
            let viewController = TalkListViewController()
            viewController.talks = Model.createTalks()
            viewController.router = anyRouter
            return .push(viewController)
        }
    }
}
```

Solution 1.2 - Slide 1/2

- Adapt **TalksRoute** in **TalksRoute.swift**

```
enum TalksRoute: Route {
    case initial
    case detail(Talk)
}
```

- Adapt **prepareTransition(for:)** in **TalksCoordinator**

```
case let .detail(talk):
    let viewController = TalkDetailViewController()
    viewController.talk = talk
    viewController.router = anyRouter
    return .push(viewController, animation: .fade)
```

(optional) Solution 1.3

Solution 1.2 - Slide 2/2

- Add **router** property to **TalksViewController** in **TalksViewController.swift**

```
class TalkListViewController: UIViewController {
    /* .. */
    var router: AnyRouter<TalksRoute>!
}
```

- Trigger detail route in **tableView(_:didSelectRowAt:)**

```
extension TalkListViewController: UITableViewDelegate {
    func tableView(_ tableView: UITableView,
                   didSelectRowAt indexPath: IndexPath) {
        let model = talks[indexPath.row]
        router.trigger(.detail(model))
    }
}
```

Solution 2

- Use **TalksCoordinator** by adapting **HomeCoordinator** in **HomeRoute.swift**

```
override func prepareTransition(for route: HomeRoute) -> TabBarTransition {
    switch route {
    case .initial:
        let companyViewController = CompanyViewController()
        companyViewController.company = Model.createCompany()
        let companyRoot = UINavigationController(rootViewController: companyViewController)
        let talksCoordinator = TalksCoordinator()
        return .multiple(
            .set([talksCoordinator, companyRoot]),
            .select(companyRoot)
        )
    }
}
```

Solution 3.1

- Create **CompanyRoute** in **CompanyRoute.swift**

```
enum CompanyRoute: Route {
    case initial
}
```

Solution 3.1

- Create **CompanyRoute & CompanyCoordinator** in **CompanyRoute.swift**

```
enum CompanyRoute: Route {
    case initial
}


class CompanyCoordinator: NavigationCoordinator<CompanyRoute> {

    init() {
        super.init(initialRoute: .initial)
    }

    override func prepareTransition(for route: CompanyRoute) -> NavigationTransition {
        switch route {
        case .initial:
            let viewController = CompanyViewController()
            viewController.company = Model.createCompany()
            return .push(viewController)
        }
    }
}
```

(Optional) Solution 3.2

- Add **website(URL)** case to **CompanyRoute**

```
enum CompanyRoute: Route {
    case initial
    case website(URL)
}
```

- Adapt **prepareTransition(for:)** in **CompanyCoordinator**

```
case let .website(url):
    UIApplication.shared.open(url)
    return .none()
```

(Optional) Solution 3.2

- Trigger **.website(URL)** in **CompanyViewController**

```
var router: AnyRouter<CompanyRoute>!


@objc private func websiteButtonTapped() {
    guard let url = company?.website,
        UIApplication.shared.canOpenURL(url) else {
        return assertionFailure()
    }

    router.trigger(.website(url))
}
```

Don’t forget to set the **router** of the **CompanyViewController** in the **CompanyCoordinator**!

XCoordinator

QUICKBIRD STUDIOS

- Powerful navigation library for iOS using the **Coordinator pattern**
- Developed by **QuickBird Studios**, located in Munich
- Provides **base classes for different coordinator types**, such as **NavigationCoordinator, TabBarCoordinator,** and many more
- **Encapsulates navigation code** for UIKit and provides **consistent API for different transition types**
- **Predefined transition types** with completion handler to ensure that transitions are executed sequentially
- **RxSwift** extensions!
- Full support for **custom transitions & animations**

References

- **XCoordinator**: Powerful navigation library for iOS based on the coordinator pattern (https://github.com/quickbirdstudios/XCoordinator)

- **Introducing an iOS navigation library based on the coordinator pattern** by Stefan Kofler (https://quickbirdstudios.com/blog/ios-navigation-library-based-on-the-coordinator-pattern)

- **The Coordinator** by Soroush Khanlou (http://khanlou.com/2015/01/the-coordinator)