Is Something Wrong With MVC?

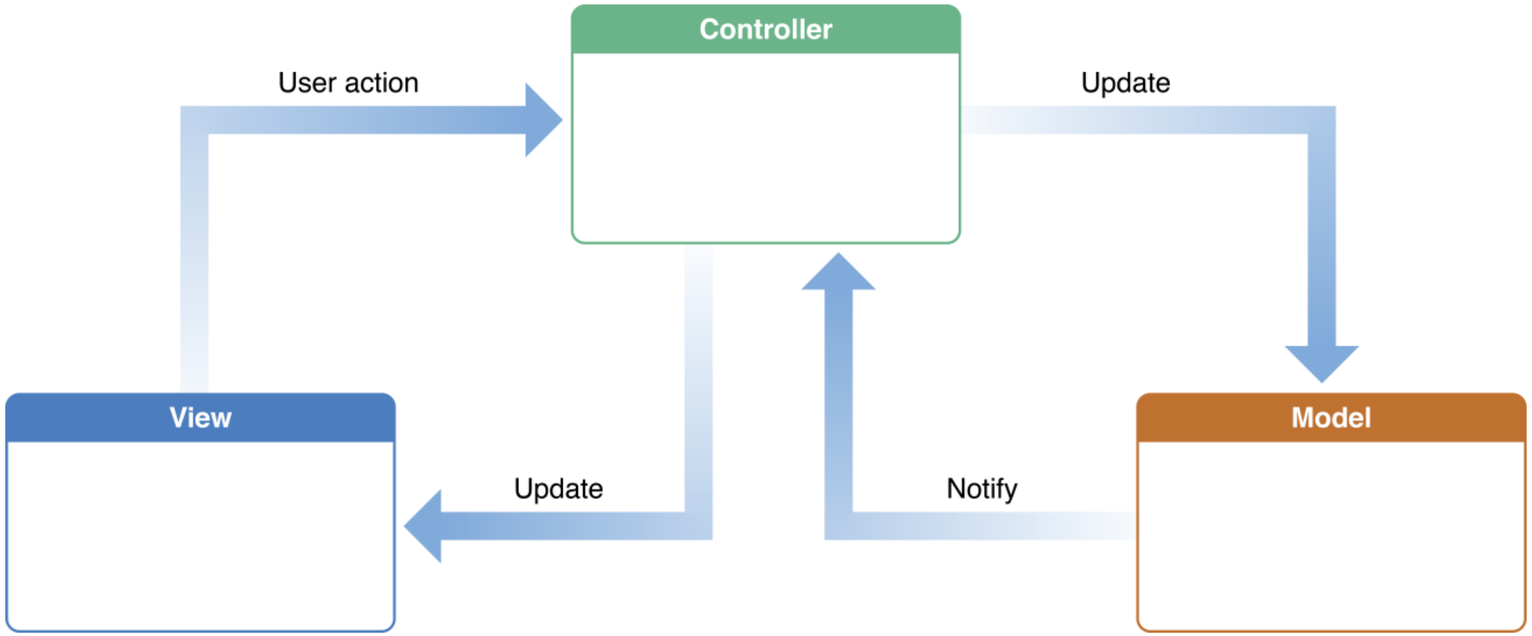
Swift Sofia Meetup

Ivelin Davidov

Overview

- Brief history of MVC
- Issues
- Coordinators
- Other architectures

MVC (Apple's version)

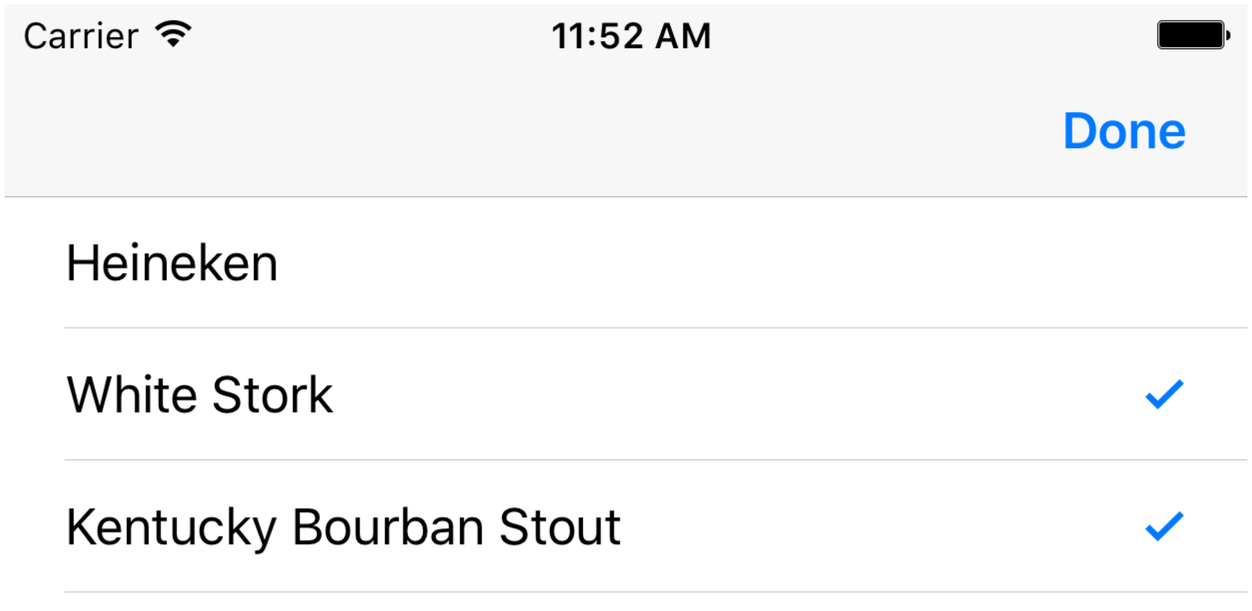


MVC (community's version)

UIViewControllers
do
many
things

UIViewControllers
do
too
many
things

BeerShoppingApp



Load data from a service

```
func loadBeers() {
      BeerService.shared.loadBeers { (beers, error) in
          self.beers = beers
          self.tableView.reloadData()
      }
}
```

Act as a DataSource

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
  -> Int {

    return self.beers?.count ?? 0
}

override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
  -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "cellId")!
    if let beer = self.beers?[indexPath.row],
       let cell = cell as? BeerTableViewCell {
       self.configure(cell, for: beer)
    }

    return cell
}
```

Configure some other view

```
func configure(_ cell: BeerTableViewCell, for beer: Beer) {
     cell.textLabel?.text = beer.name
     cell.selectionStyle = .none
}
```

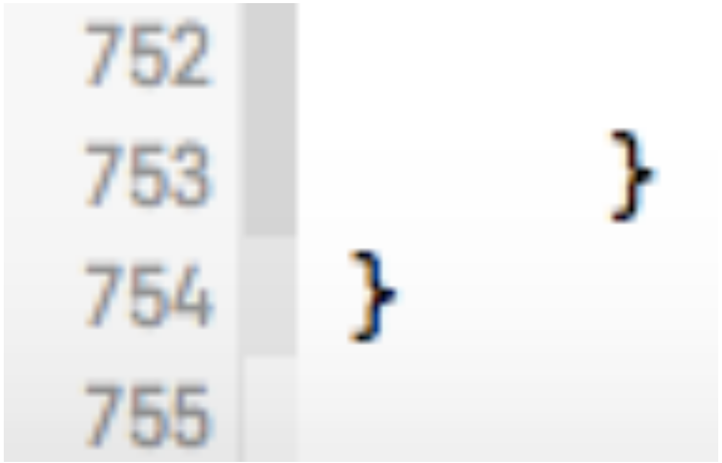
Configure some other UIViewController

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
  super.prepare(for: segue, sender: sender)

  if let fillAddressViewController = segue.destination as FillAddressViewController {

     let selectedBeers = self.selectedBeers()
     fillAddressViewController.configure(with: selectedBeers)
  }
}
```

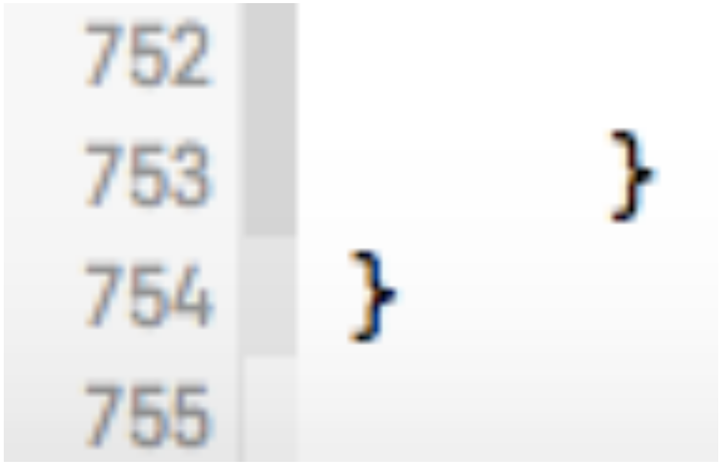
UIViewControllers do:

- Act as some other's view DataSource
- Do some fancy animations
- Save data to CoreData
- Handle user interactions

And you end up with something like



And you end up with something like



At best!

Some of this is obviously bad code…

Some of this is obviously bad code…

but Apple show it in their documentation

View code goes to in the View layer

```
extension BeerTableViewCell {
    func configure(for beer: Beer) {
        self.textLabel?.text = beer.name
        self.selectionStyle = .none
    }
}
```

Data related logic goes in the Model layer

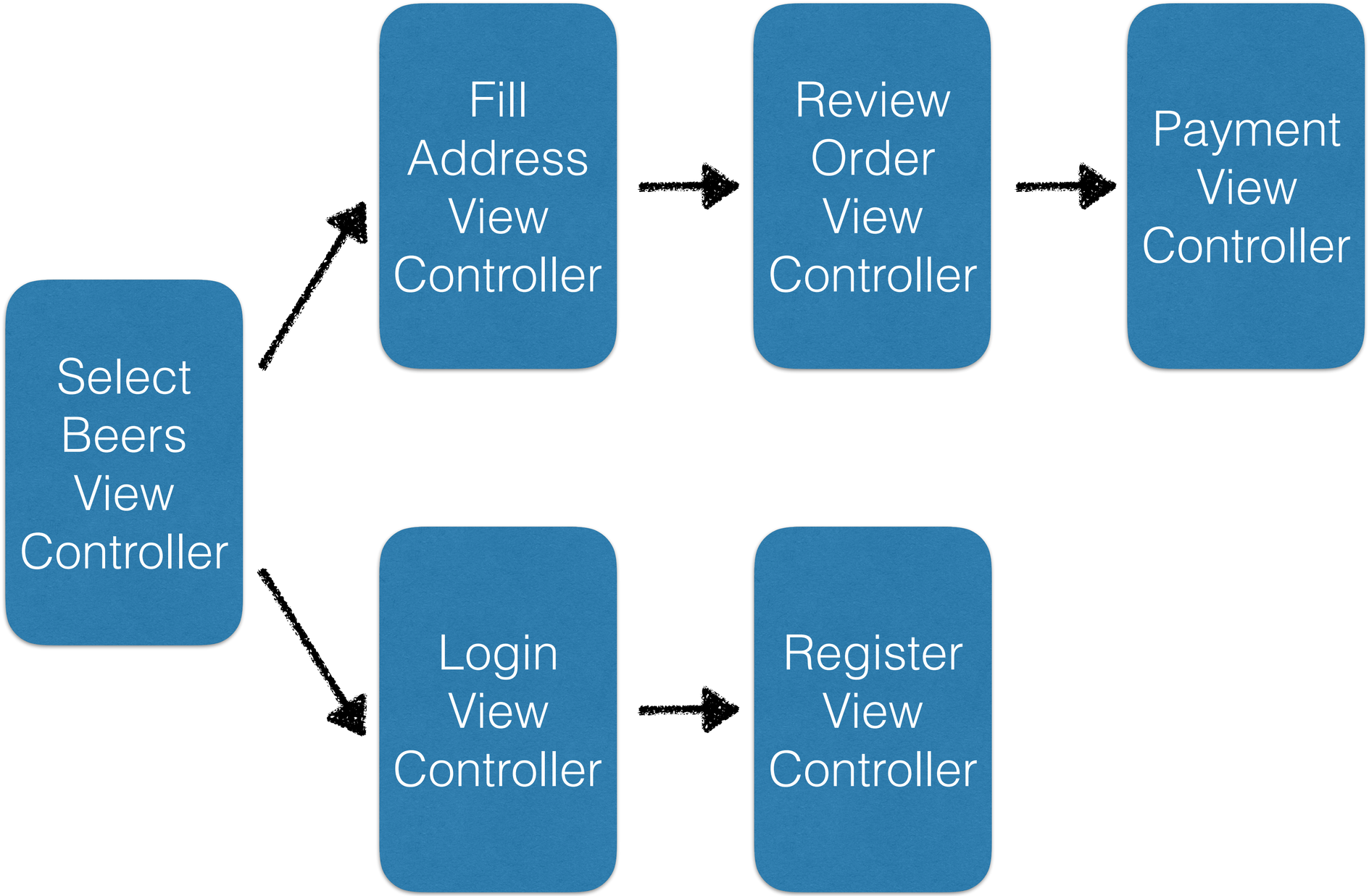
```
class BeersDataProvider {

    private var beers: [Beer] = []

    func loadBeers(completion: (Error?) -> Void) {
        BeerService.shared.loadBeers { (beers, error) in
            self.beers = beers ?? []
            completion(error)
        }
    }

    func numberOfBeers() -> Int {
        return self.beers.count
    }

    func beer(at indexPath: IndexPath) -> Beer {
        return self.beers[indexPath.row]
    }
}
```

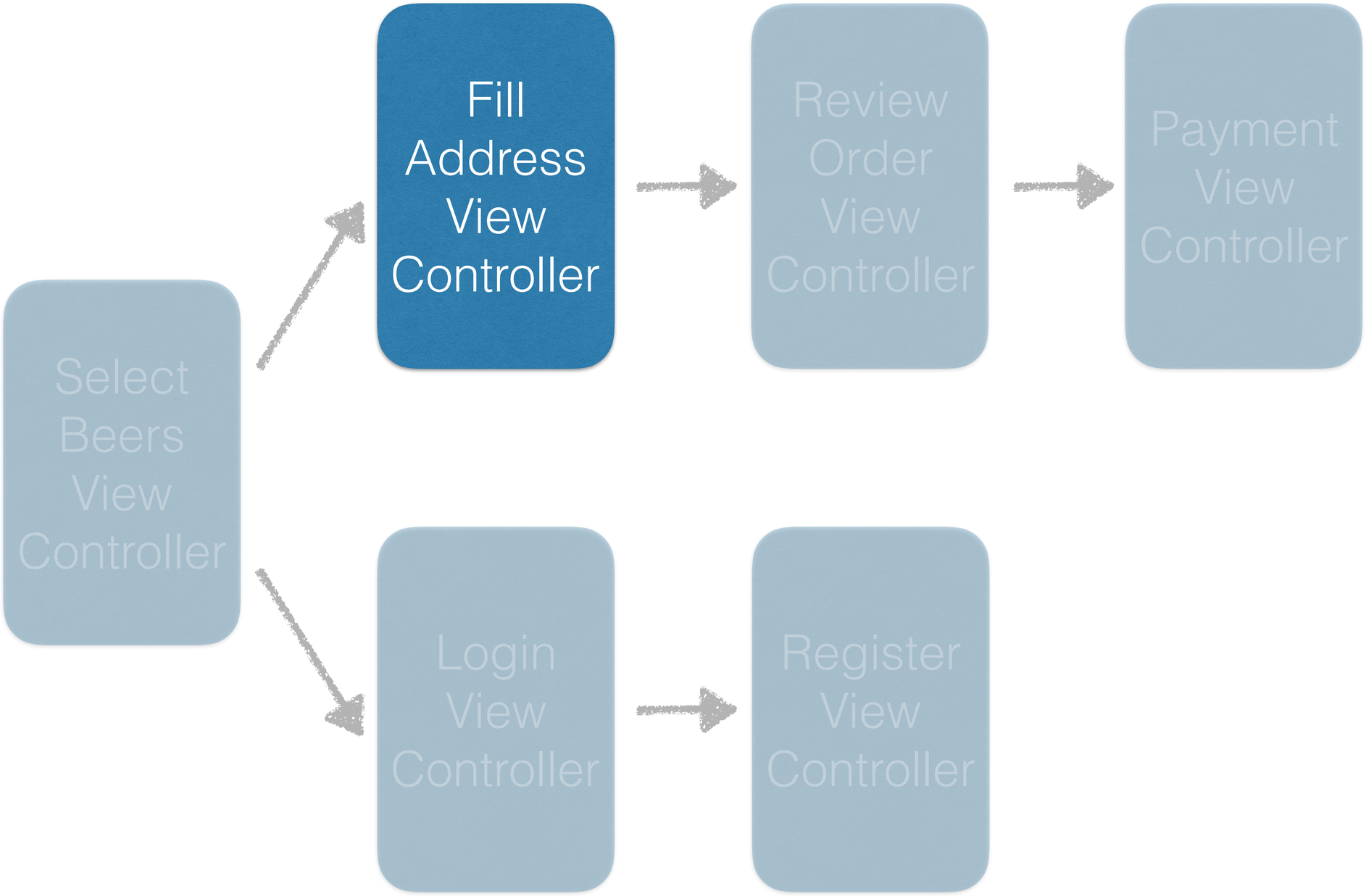
Use a separate DataSource

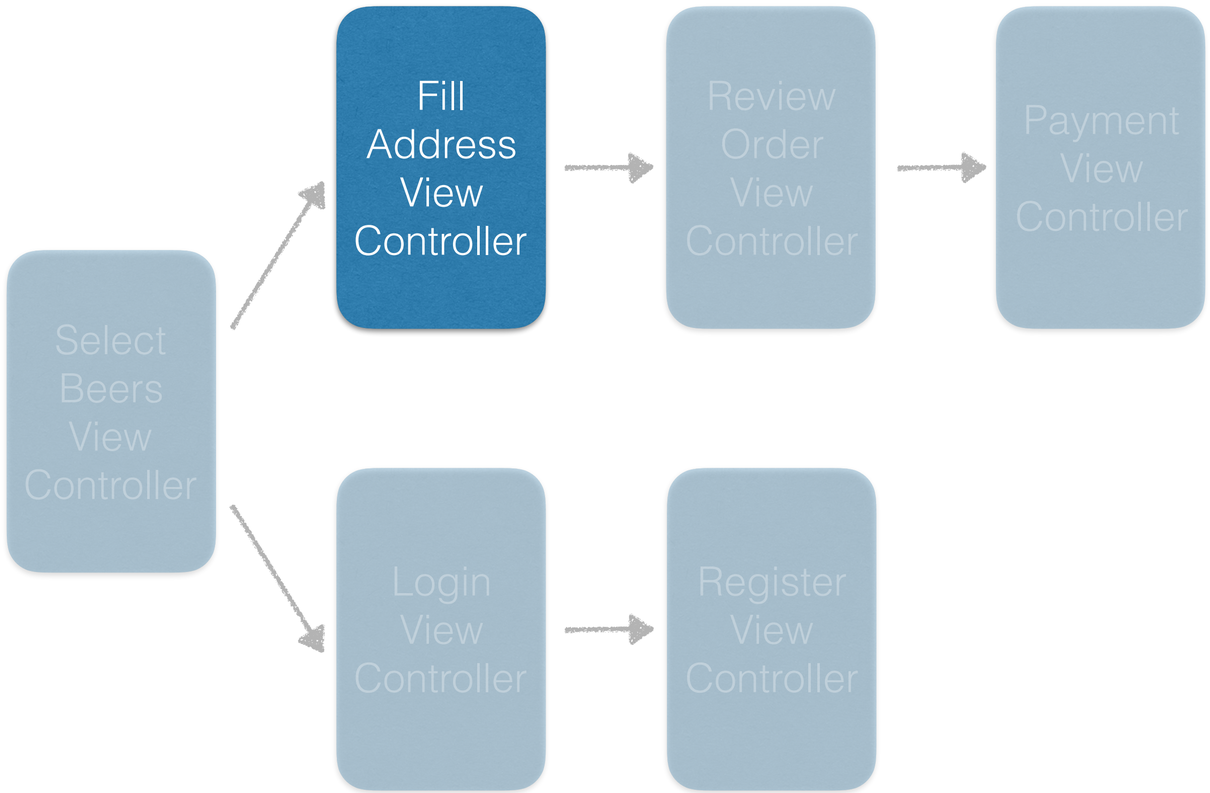
```
class BeersDataSource: NSObject, UITableViewDataSource {

  ...

  func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
     return self.dataProvider.numberOfBeers()
  }

  func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {

      let cell = tableView.dequeueReusableCell(withIdentifier: "cellId")!

      if let cell = cell as? BeerTableViewCell {
          let beer = self.dataProvider.beer(at: indexPath)
          cell.configure(for: beer)
      }
      return cell
  }
}
```

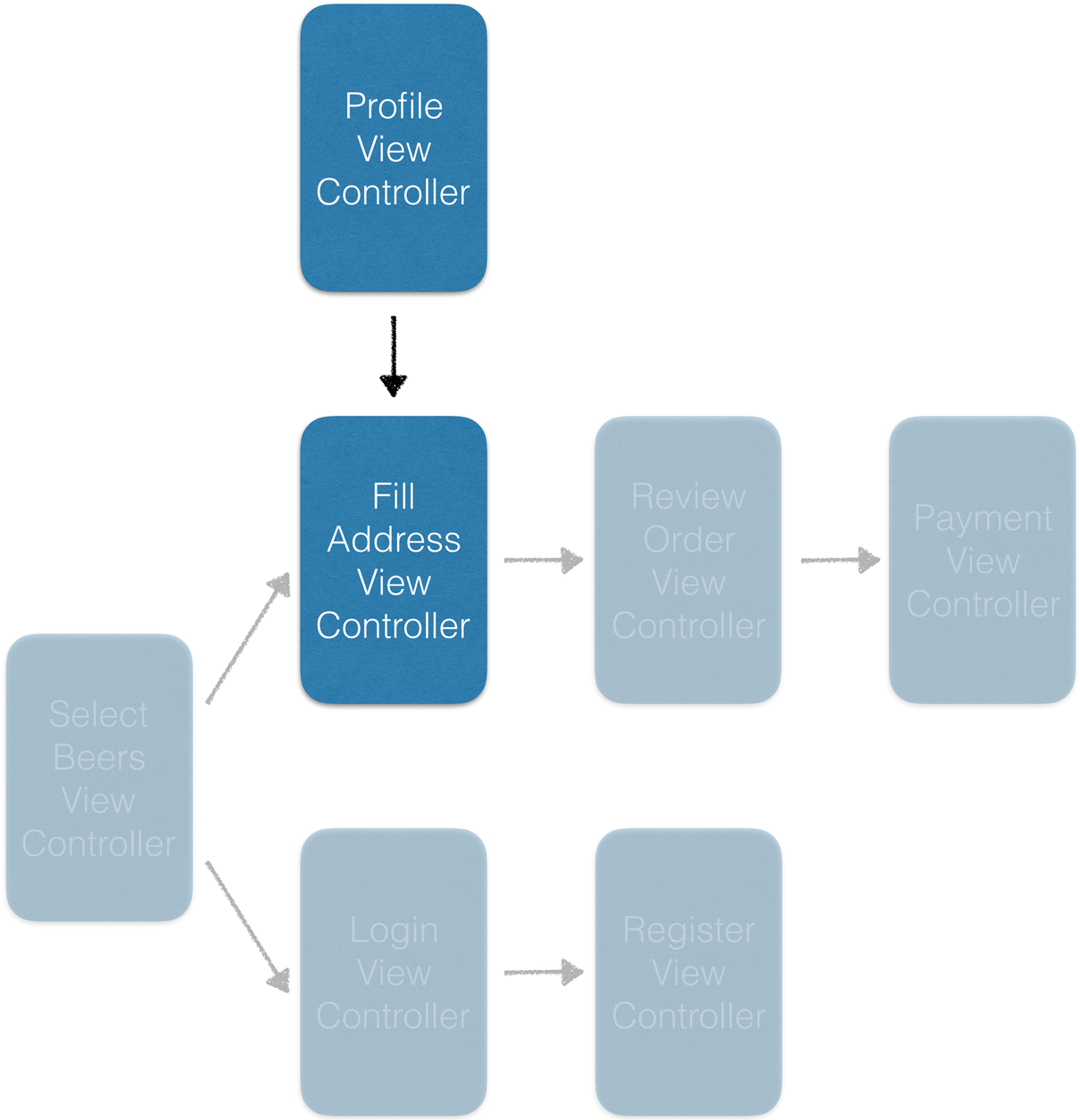
We’ve cleaned a bit,
but we still have this

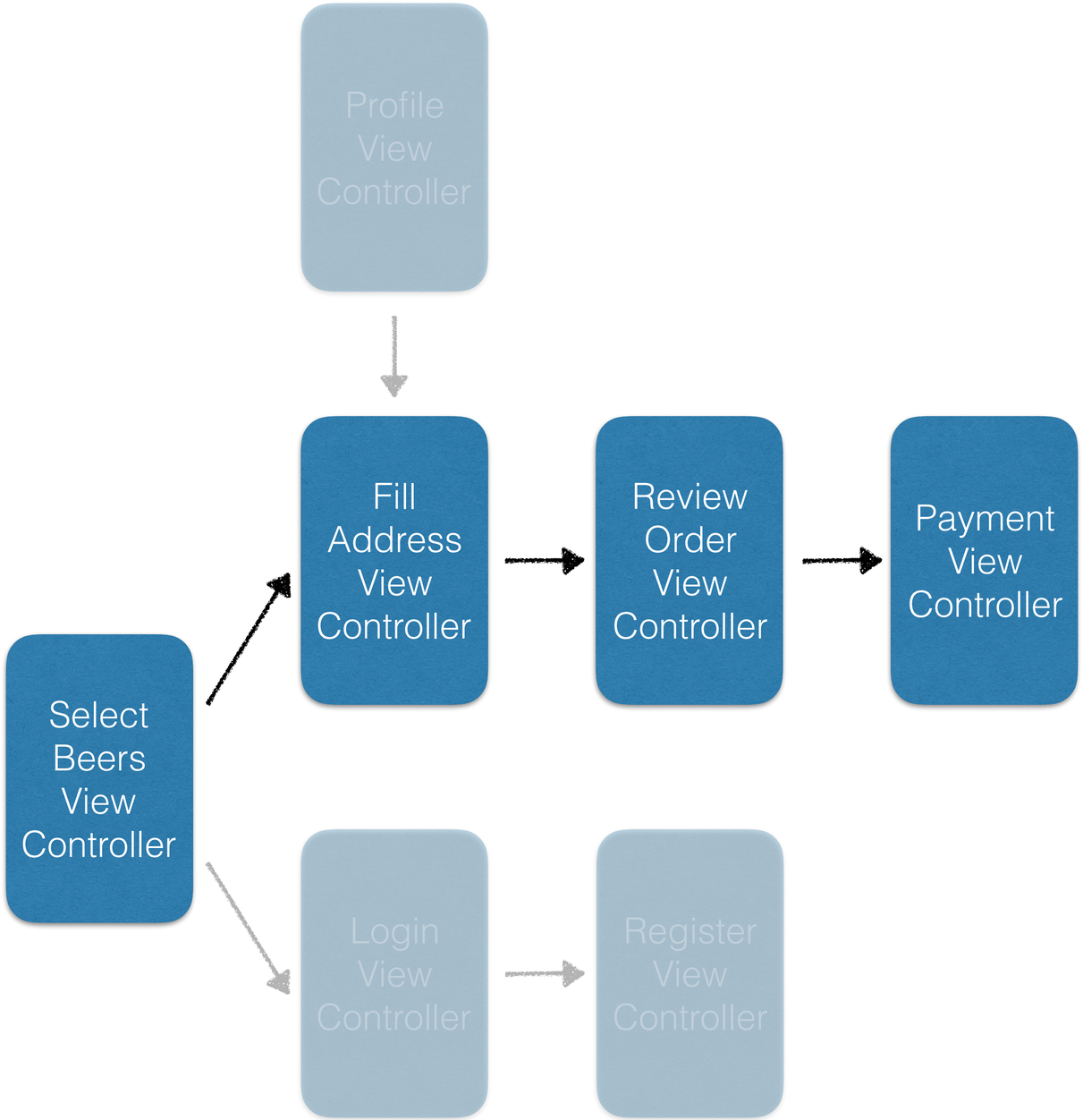
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
  super.prepare(for: segue, sender: sender)

  if let fillAddressViewController = segue.destination as? FillAddressViewController {
     let selectedBeers = self.selectedBeers()
     fillAddressViewController.configure(with: selectedBeers)
  }
}
```


Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Login View Controller
Register View Controller


Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Login View Controller
Register View Controller


Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Login View Controller
Register View Controller


Profile View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Select Beers View Controller
Login View Controller
Register View Controller


Profile View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Select Beers View Controller
Login View Controller
Register View Controller

Issue #1
Navigation logic inside UIViewControllers makes them tightly coupled and hardly reusable

How are dependencies resolved?

How are dependencies resolved?

- Singletons

Singletons

```
class SelectBeersViewController: UITableViewController {
    …
    func didTapDoneButton(sender: UIBarButtonItem) {
        let selectedBeers = self.selectedBeers()
        SelectBeersManager.shared.selectBeers(selectedBeers)
    }
    …
}
```

Singletons

```
class SelectBeersViewController: UITableViewController {
    …
    func didTapDoneButton(sender: UIBarButtonItem) {
        let selectedBeers = self.selectedBeers()
        SelectBeersManager.shared.selectBeers(selectedBeers)
    }
    …
}
```

```
class ReviewOrderViewController: UIViewConntroller {
    …
    func getSelectedBeers() -> [Beer] {
        return SelectBeersManager.shared.selectBeers(selectedBeers)
    }
    …
}
```

How are dependencies resolved?

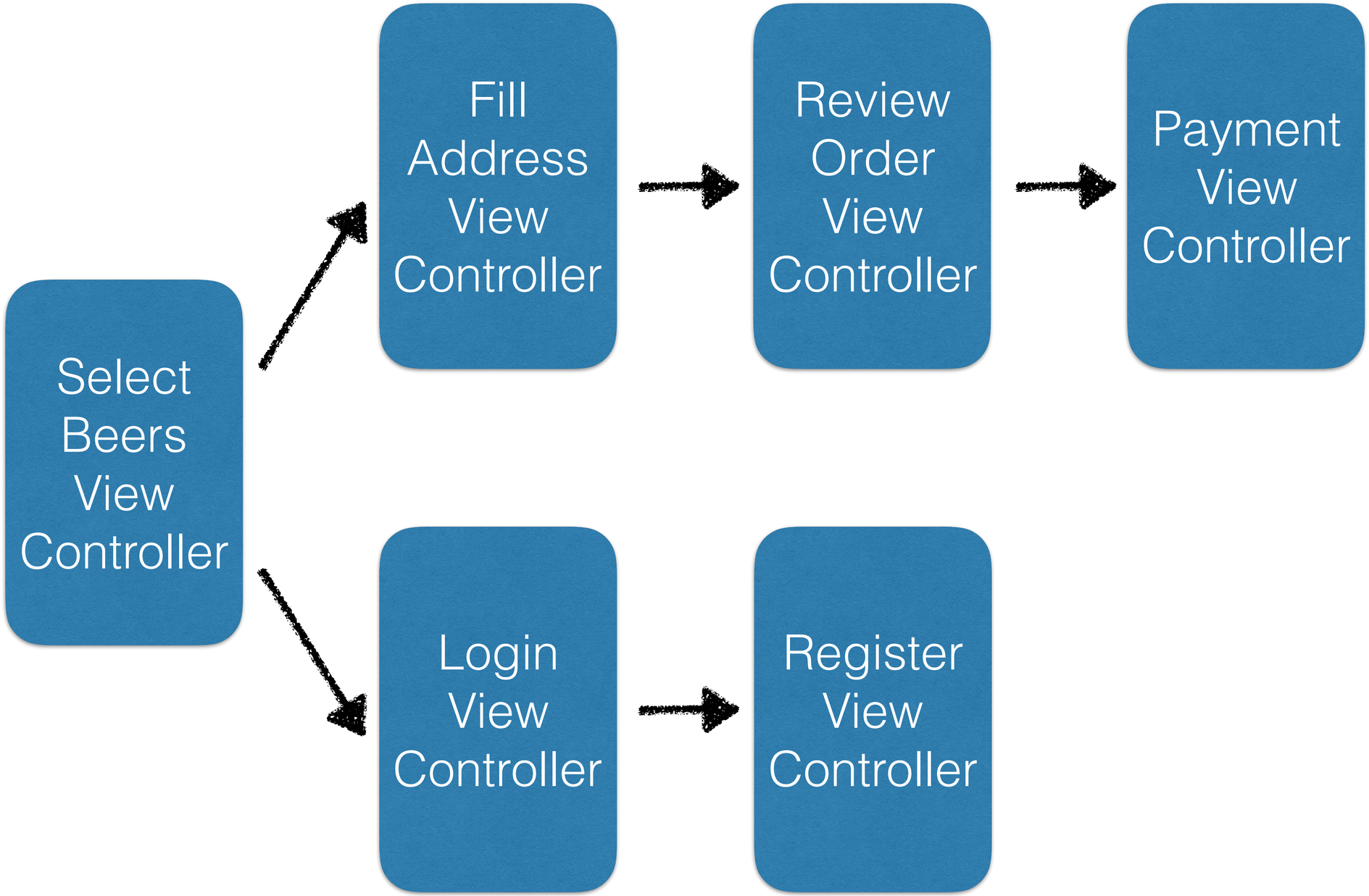
- Singletons
- AppDelegate is used as a container for dependencies

Misusing AppDelegate

```
if let selectBeersViewController = segue.destination as? SelectBeersViewController {
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let dataSource = appDelegate.getSelectBeersDataSource()

    selectBeersViewController.dataSource = dataSource
}
```

How are dependencies resolved?

- Singletons
- AppDelegate is used as a container for dependencies
- Every UIViewController know about other's dependencies


Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller
Login View Controller
Register View Controller

Issue #2
Navigation logic inside UIViewControllers makes application flow hard to change
and
dependency injection a burden

Issue #1 and Issue #2

- Navigation logic inside UIViewControllers makes them tightly coupled and hardly reusable
- Navigation logic inside UIViewControllers makes application flow hard to change and dependency injection a burden

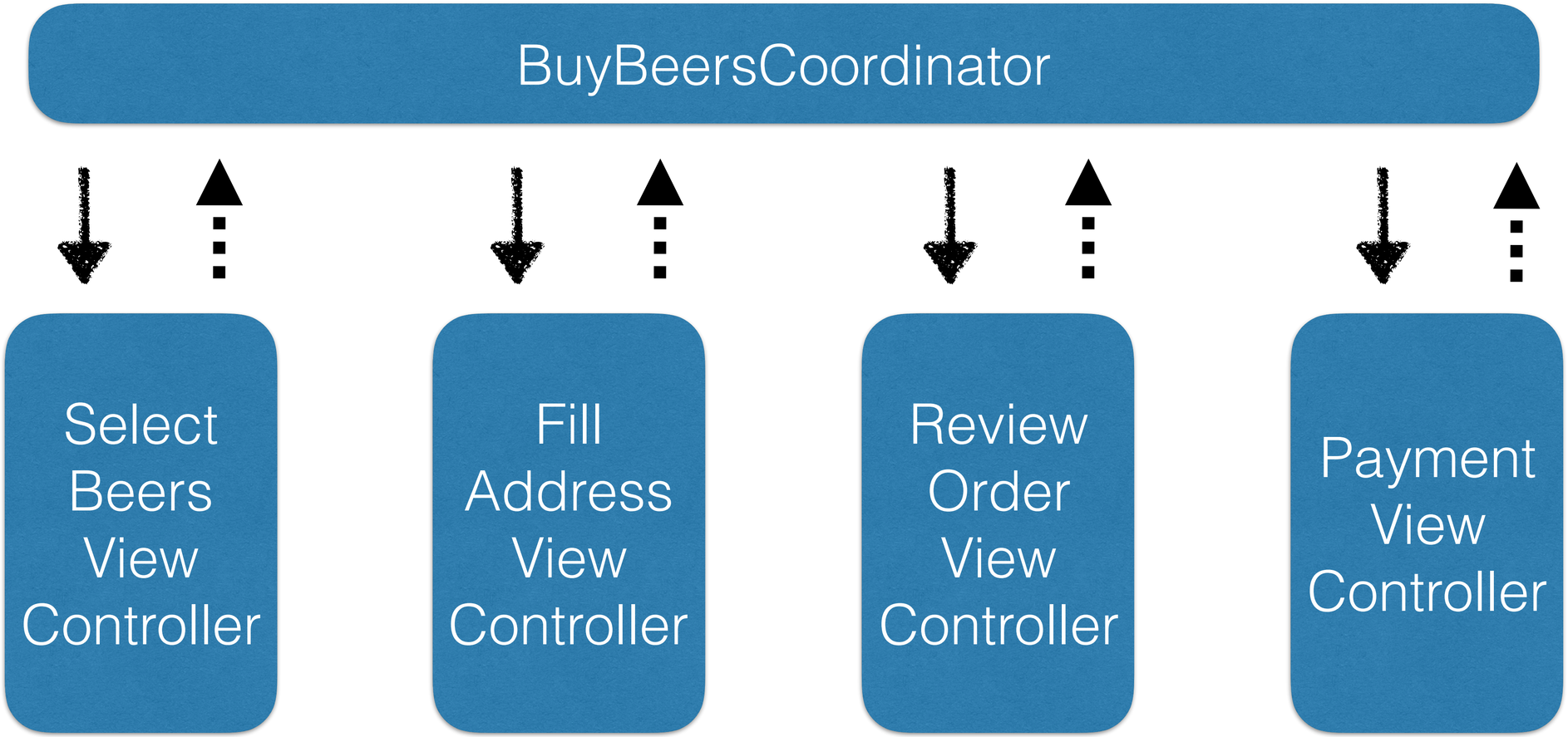
Coordinators

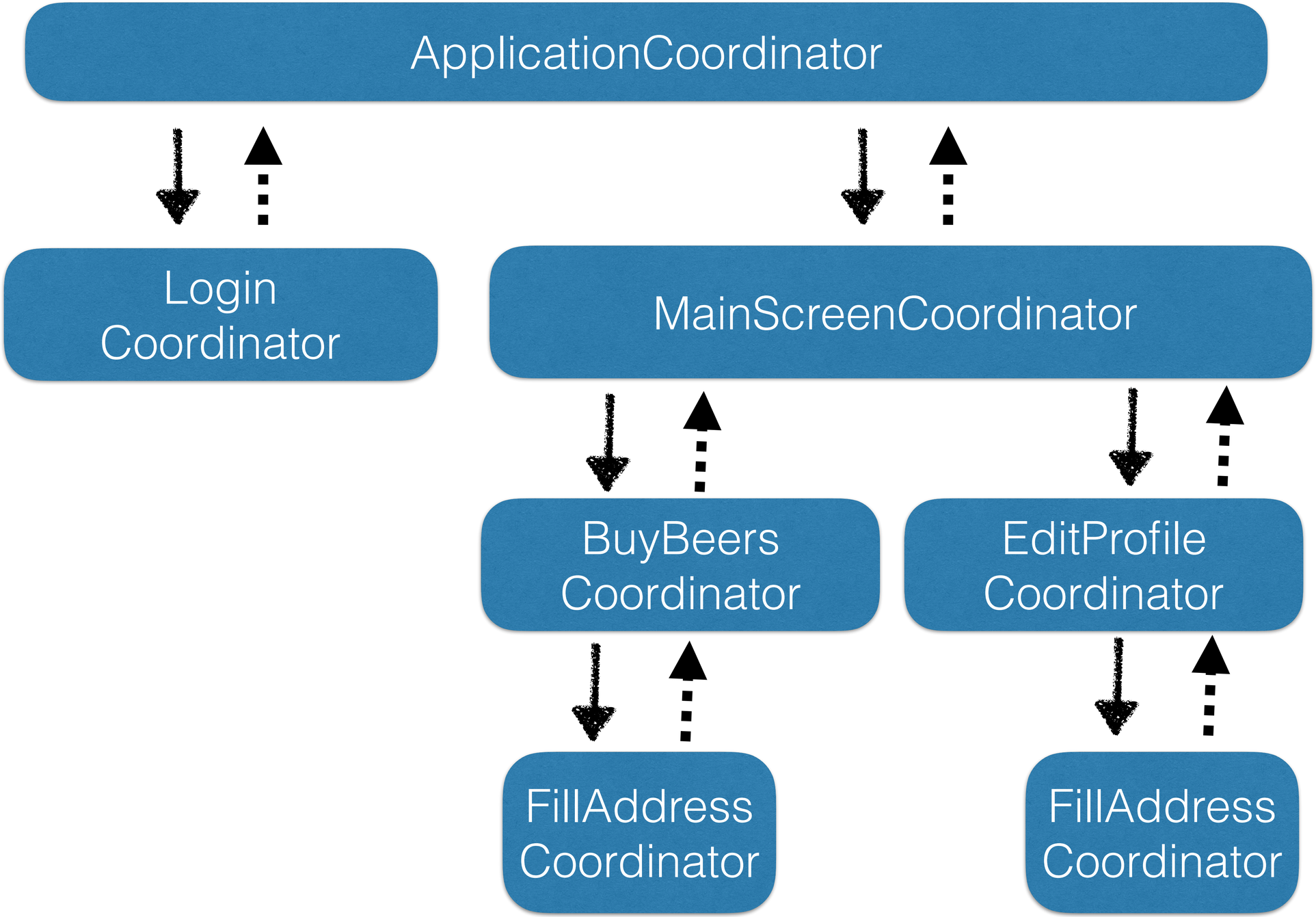
Coordinators

- Manage the application flow

Coordinators

- Manage the application flow
- Act as dependency injection container


BuyBeersCoordinator
Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller


ApplicationCoordinator
Login Coordinator
MainScreenCoordinator
BuyBeers Coordinator
EditProfile Coordinator
FillAddress Coordinator
FillAddress Coordinator

Coordinators source code

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

  let navigationController = UINavigationController()

  self.applicationCoordinator = ApplicationCoordinator(navigationController:
    navigationController)
  self.applicationCoordinator.start()

  self.window = UIWindow(frame: UIScreen.main.bounds)
  self.window!.rootViewController = navigationController
  self.window!.makeKeyAndVisible()

  return true
}
```

Coordinators source code (2)

```
class ApplicationCoordinator {

    private weak var navigationController: UINavigationController!

    private var childViewCoordinators: [String : Coordinator] = [:]

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        if self.isUserLoggedIn() {
            self.showMainScreen()
        }
        else {
            self.showLogin()
        }
    }
    …
}
```

Coordinators source code (3)

```
class ApplicationCoordinator {
  …
    private func showMainScreen() {
        let coordinator = MainScreenCoordinator(navigationController:
            self.navigationController)

        coordinator.start()

        let key = String(describing: coordinator.self)
        self.childViewCoordinators[key] = coordinator
    }
  …
}
```

Coordinators source code (2)

```
class BuyBeersCoordinator {
    ...
    private func showBeerList() {
        let dataProvider = BeersDataProvider()
        let dataSource = BeersDataSource(dataProvider: dataProvider)
        let controller = SelectBeersViewController(dataProvider: dataProvider,
           dataSource: dataSource)

        controller.delegate = self

        let navigationController = self.initializeNavigationController()
        navigationController.addChildViewController(controller)

        self.presentNavigationController()
    }
    ...
}
```

Coordinators source code (2)

```
extension BuyBeersCoordinator: SelectBeersViewControllerDelegate {
   ...
  func didSelectBeers(controller: SelectBeersViewController, selectedBeers: [Beer]) {
     self.selectedBeers = selectedBeers
     self.showFillAddress()
  }
   ...
}
```

Coordinators source code (2)

```
class BuyBeersCoordinator {
    ...
    private func showFillAddress() {
        let dataProvider = FillAddressDataProvider()
        let dataSource = FillAddressDataSource(dataProvider: dataProvider)
        let validator = FillAddressValidator()

        let controller = FillAddressViewController(dataSource: dataSource,
           dataProvider: dataProvider, validator: validator)

        controller.delegate = self

        self.navigationController.pushViewController(controller, animated: true)
    }
    ...
}
```

Coordinators source code (2)

```
extension BuyBeersCoordinator: FillAddressViewControllerDelegate {
    ...
    func didFinishedFillingAddress(address: FilledAddress) {
        self.address = address
        self.finishBuyBeersFlow()
    }
    ...
}
```

Coordinators source code (2)

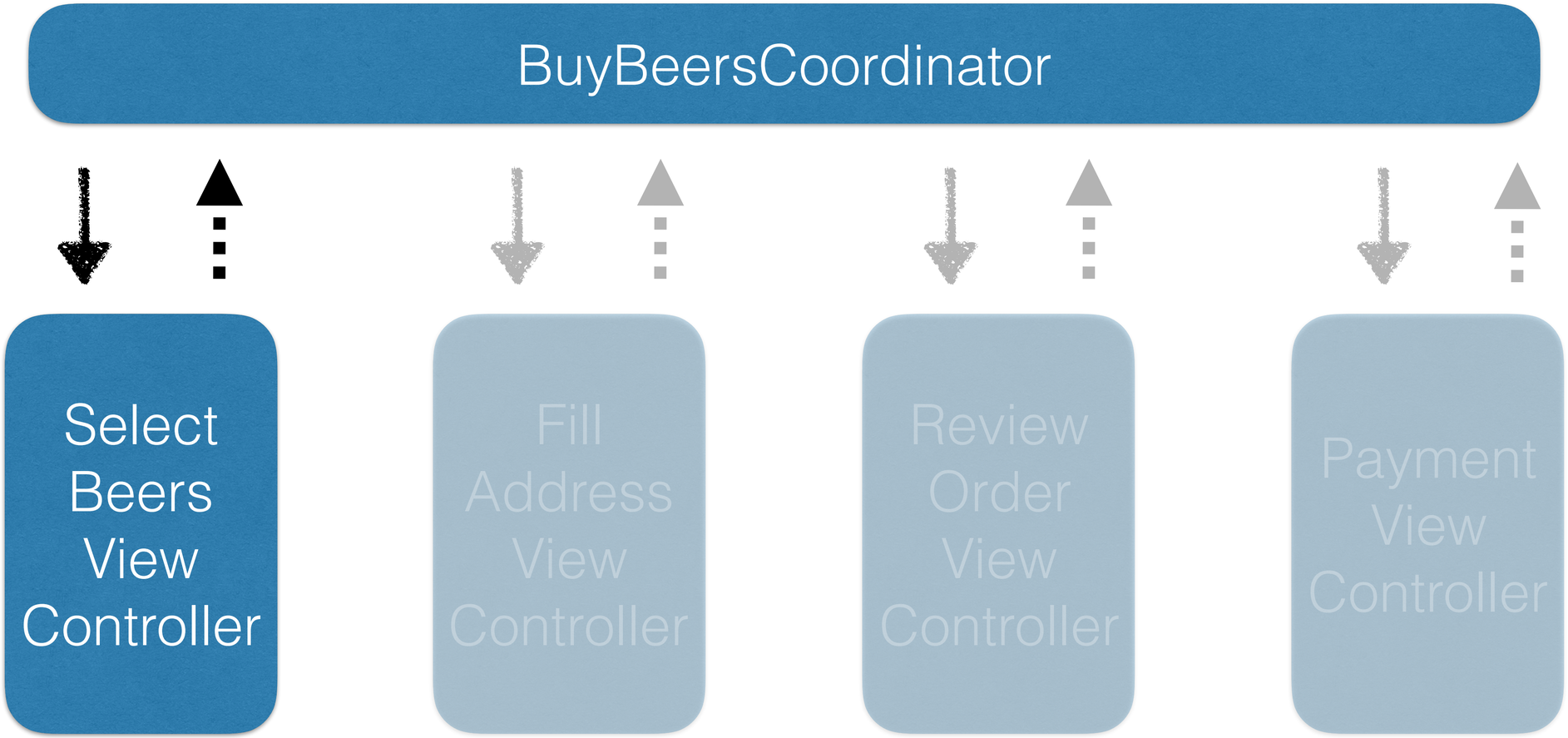
```
class BuyBeersCoordinator {
    ...
    private func finishBuyBeersFlow() {
        //TODO: Write actual implementation for next Swift Sofia Meetup :)
        print("Did bought \(self.selectedBeers) for address \(self.address)”)
        self.parentNavigationController.dismiss(animated: true, completion: nil)
        self.delegate?.didFinishBuyingBeers(coordinator: self)
    }
    ...
}
```

Coordinators source code (2)

```
extension MainScreenCoordinator : BuyBeersCoordinatorDelegate {
    func didFinishBuyingBeers(coordinator: BuyBeersCoordinator) {
        let key = String(describing: coordinator.self)
        self.childViewCoordinators.removeValue(forKey: key)
    }
}
```

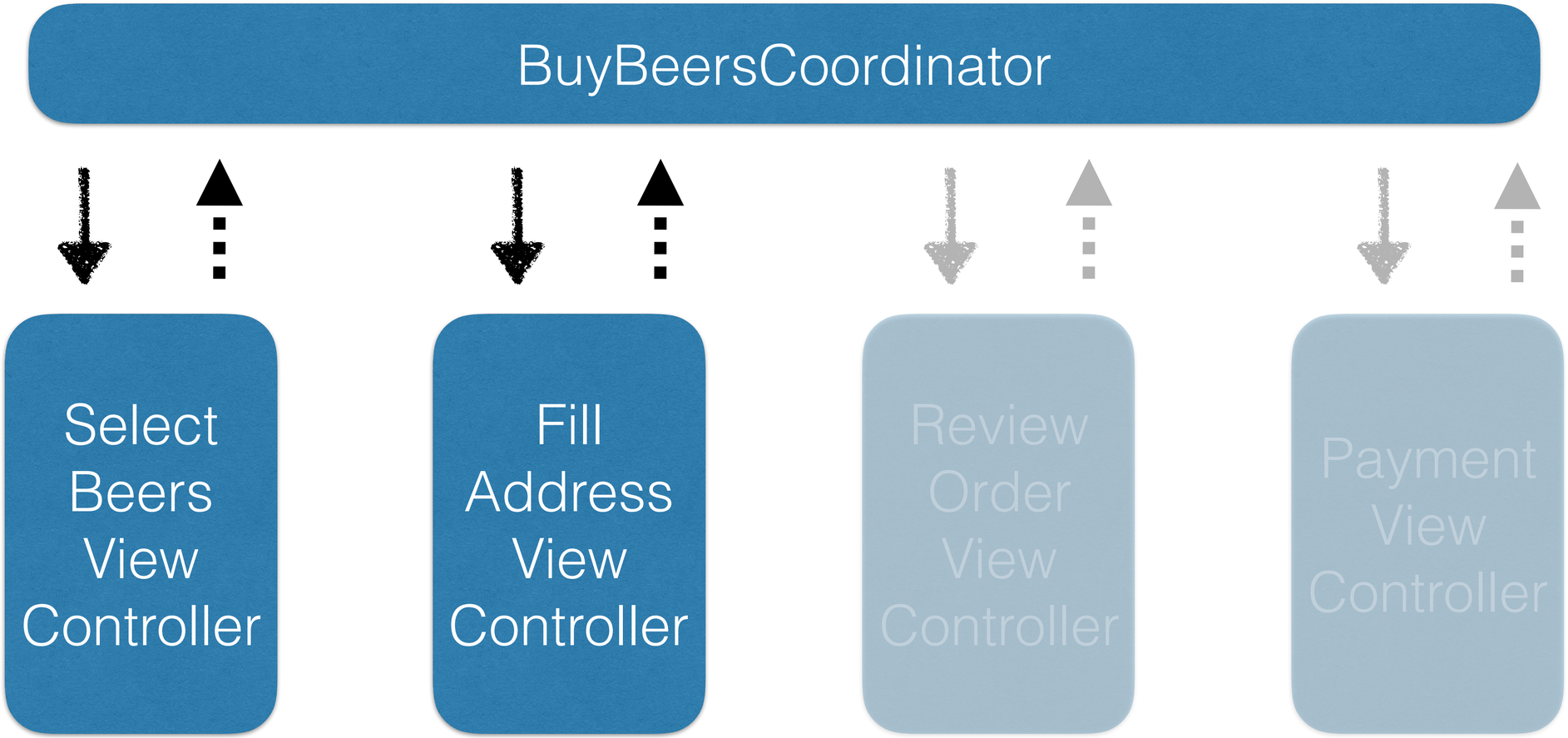
Coordinators

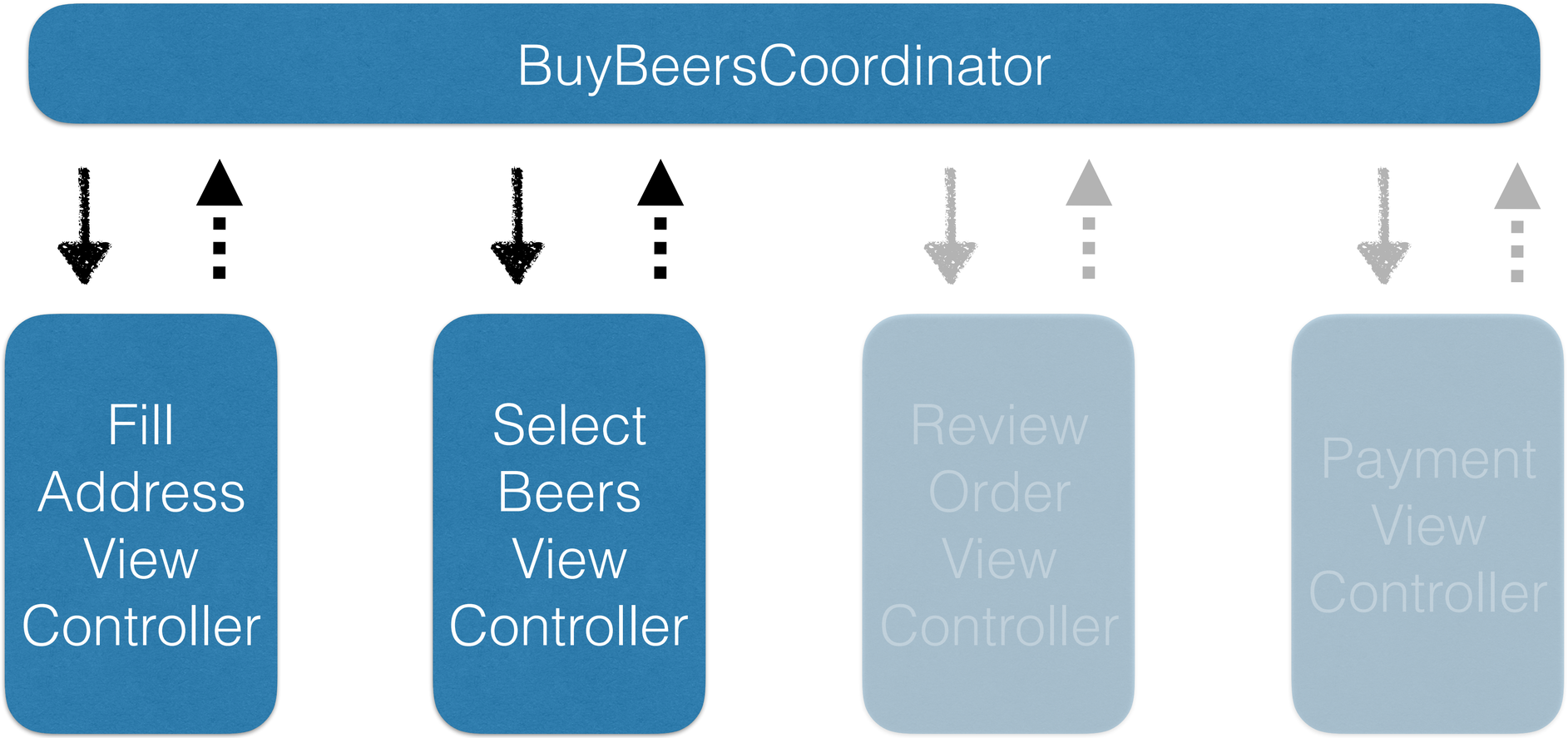
- Make UIViewControllers independent and reusable


BuyBeersCoordinator
Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller

Coordinators

- Make UIViewControllers independent and reusable
- Give us flexibility with application flow


BuyBeersCoordinator
Select Beers View Controller
Fill Address View Controller
Review Order View Controller
Payment View Controller


BuyBeersCoordinator
Fill Address View Controller
Select Beers View Controller
Review Order View Controller
Payment View Controller

Coordinators

- Make UIViewControllers independent and reusable
- Give us flexibility with application flow
- Allow us to use dependency injection with an ease

Dependency Injection with Coordinators

```
private func showSelectBeersList() {
    let dataProvider = BeersDataProvider()
    let dataSource = BeersDataSource(dataProvider: dataProvider)
    let vc = SelectBeersViewController(dataProvider: dataProvider, dataSource:
        dataSource)

    vc.delegate = self

    self.navigationController.pushViewController(vc, animated: true)
}
```

Coordinators

- Make UIViewControllers independent and reusable
- Give us flexibility with application flow
- Allow us to use dependency injection with an ease
- Flow Controllers, Presenting Controllers, etc

Coordinators

- Make UIViewControllers independent and reusable
- Give us flexibility with application flow
- Allow us to use dependency injection with an ease
- Flow Controllers, Presenting Controllers, etc
- Soroush Khanlou - Coordinators Redux

Aren’t Storyboards the same thing?

Aren’t Storyboards the same thing?

- Storyboards define application flow

Aren’t Storyboards the same thing?

- Storyboards define application flow
- Can inject dependencies

Storyboards have significant flaws

Storyboards have significant flaws

- They make view code hard to reuse

Storyboards have significant flaws

- They make view code hard to reuse
- They make view code hard to modify at runtime

Storyboards have significant flaws

- They make view code hard to reuse
- They make view code hard to modify at runtime
- They fail at runtime not at compile time

Storyboards have significant flaws

- They make view code hard to reuse
- They make view code hard to modify at runtime
- They fail at runtime not at compile time
- Code reviews are close to impossible

Storyboards have significant flaws

- They make view code hard to reuse
- They make view code hard to modify at runtime
- They fail at runtime not at compile time
- Code reviews are close to impossible
- Merge conflicts can be really hard

Storyboards have significant flaws (2)

- They can only kind of inject dependencies

Storyboards have significant flaws (2)

- They can only kind of inject dependencies
- They are slow

Storyboards have significant flaws (2)

- They can only kind of inject dependencies
- They are slow
- They are out of your control

Storyboards have significant flaws (2)

- They can only kind of inject dependencies
- They are slow
- They are out of your control
- They change every single time you open them @!#@!#

Storyboards make easy tasks easier and hard tasks harder

If you still want to use Storyboards with Coordinators

http://www.apokrupto.com/blog-1/2016/3/17/coordinators-with

What about other architectural patterns?

MVVM

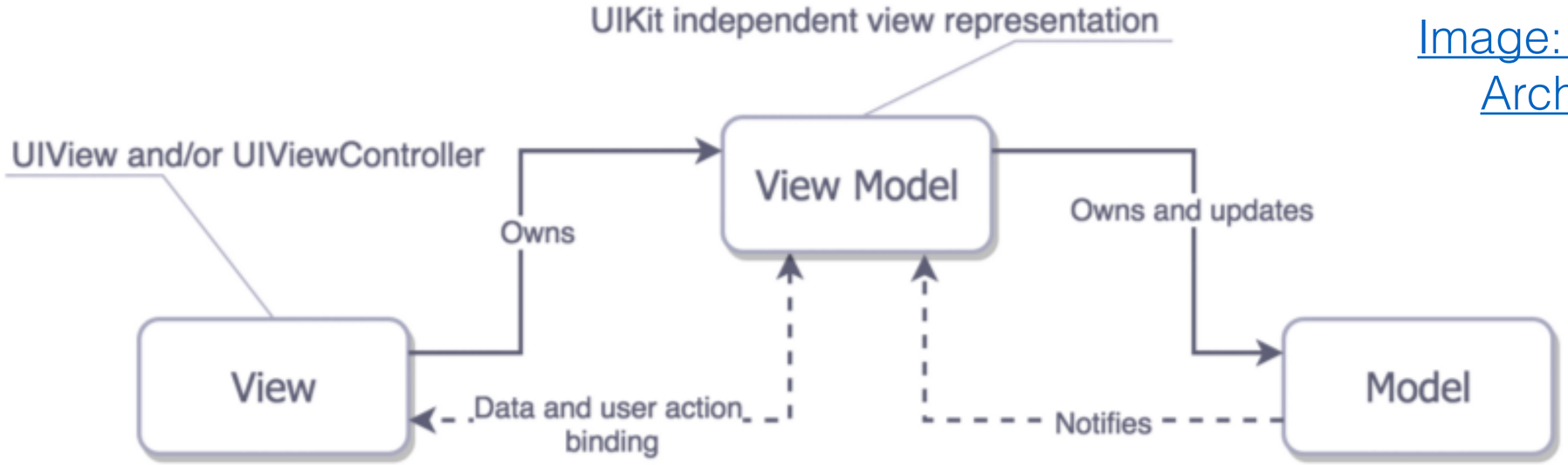
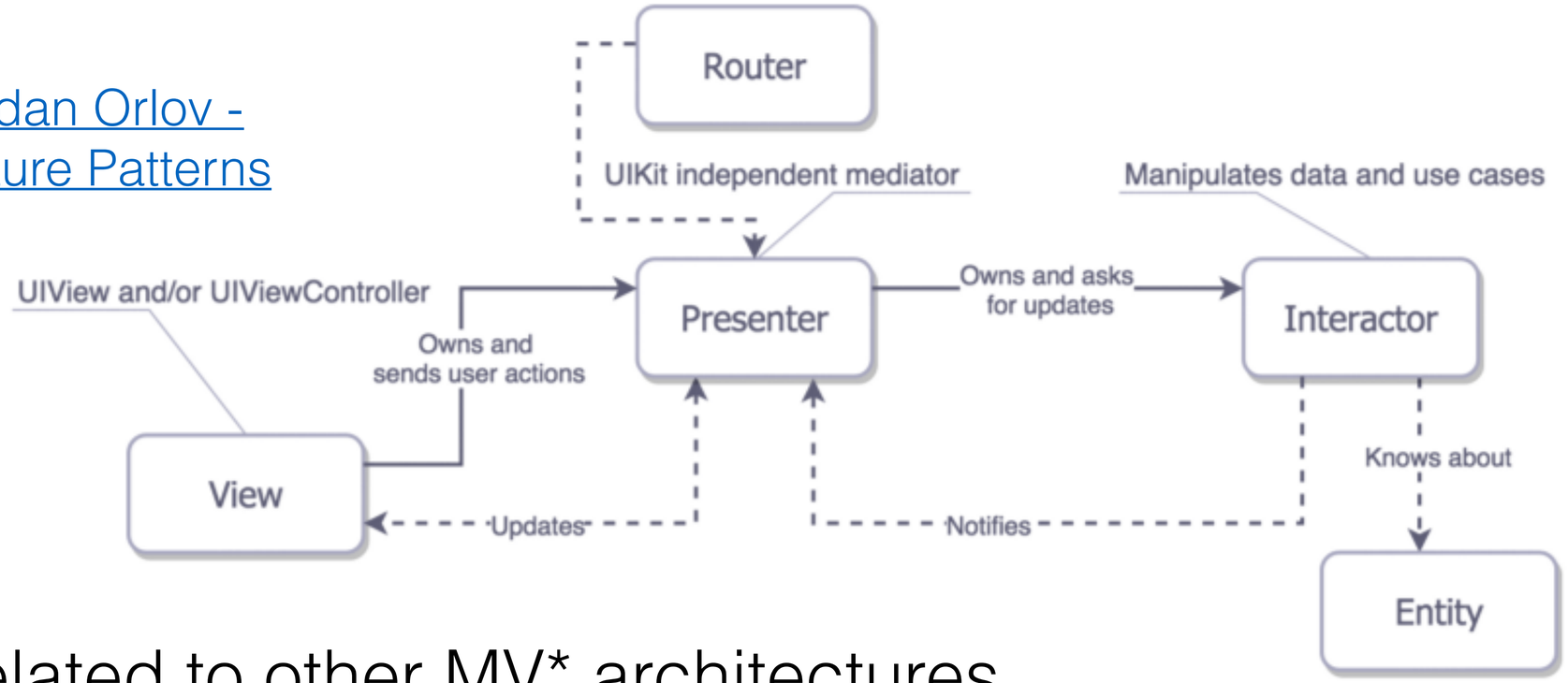


Image: Bohdan Orlov - iOS Architecture Patterns

- Adds a ViewModel layer
- ViewModels are view representations that contain no reference to UIKit
- UIViewControllers are treated as part of the View layer
- Data binding if we use a third party functional reactive framework
- Doesn’t solve the application flow issue

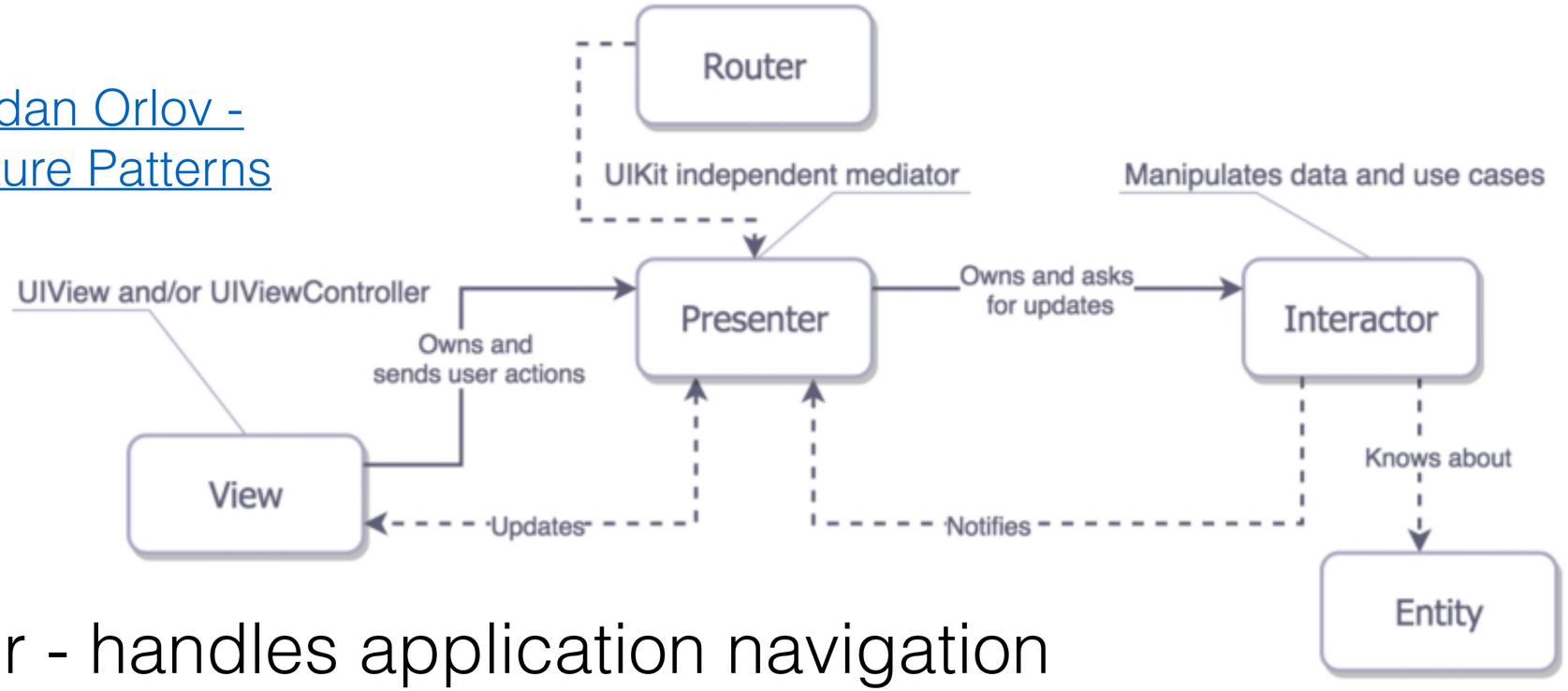
VIPER

[Image: Bohdan Orlov - iOS Architecture Patterns]



- Not related to other MV* architectures
- Implementation of Uncle Bob's Clean Architecture
- Five layers

[Image: Bohdan Orlov - iOS Architecture Patterns]



- Router - handles application navigation
- Entity - plain data objects
- Interactor - business logic and networking
- Presenter - prepares data to be displayed by the View
- View - includes both UIViews and UIViewControllers

Is something wrong with MVC?

- We can do better job by making smaller UIViewControllers
  - Put logic in the layer it belongs
  - Use composition to separate concerns
  - Lighter View Controllers

Is something wrong with MVC?

- We can do better job by making smaller UIViewControllers
  - Put logic in the layer it belongs
  - Use composition to separate concerns
  - Lighter View Controllers
- Coordinators can help by moving the application flow in a separate layer
  - Makes navigation more flexible
  - Help us with dependency injection
  - Coordinators Redux

Is something wrong with MVC?

- Other architectures solve some problems MVC has
  - 3rd party framework is required
  - New paradigms are used
  - Can be too much for smaller projects

Is something wrong with MVC?

- Other architectures solve some problems MVC has
  - 3rd party framework is required
  - New paradigms are used
  - Can be too much for smaller projects
    - Are there small projects really?

Is something wrong with MVC?

- Other architectures solve some problems MVC has
  - 3rd party framework is required
  - New paradigms are used
  - Can be too much for smaller projects
    - Are there small projects really?
- Always use common sense

Questions?

Thank you!

Let’s have some