

VIPER

LOOKING FOR THE PERFECT ARCHITECTURE



@PEPIBUMUR

TWEET ABOUT THE TALK WITH THE #VIPERTALK HASHTAG

INDEX

- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

INDEX

- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

DOCUMENTATION, BOOKS,
EXAMPLES, TUTORIALS

*objc.io, RayWenderlich, iOS Dev Weekly,
NSHipster*

PATTERNS

PROTOCOLS, DELEGATES, DATA SOURCES

KVO

OHMAGIF.COM



Remember
**THEY ARE
EXAMPLES**

VIEWCONTROLLER

**A VIEW CONTROLLER COORDINATES ITS EFFORTS
WITH MODEL OBJECTS AND OTHER CONTROLLER
OBJECTS—INCLUDING OTHER VIEW CONTROLLERS**

— Apple

► ViewControllerDelegator

```
@interface TBThreadDetailViewController : RBViewController <UITableViewDataSource,  
UITableViewDelegate, RBViewControllerURLProtocol, NSFetchedResultsControllerDelegate, TBObjectDetailHeaderCellDelegate,  
TBUploadCellDelegate, TBWatchersViewDelegate>
```



► ViewControllerDelegator

```
@interface TBThreadDetailViewController : RBViewController <UITableViewDataSource,  
UITableViewDelegate, RBViewControllerURLProtocol, NSFetchedResultsControllerDelegate, TBObjectDetailHeaderCellDelegate,  
TBUploadCellDelegate, TBWatchersViewDelegate>
```



► Multi Responsibility

```
[self presentViewController:previewController animated:YES completion:nil]; //Navigation  
[Task downloadNewObjectWithID:self.threadIdentifier.remoteIdentifier withSuccess:^{}]; // Networking  
NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName:[Comment entityName]]; // Data persistence  
[self.view setBackgroundColor:ss_Color_White]; // Formatting
```



► Monster files
size > 1500 lines 😢

- ▶ **Monster files**
size > 1500 lines 
- ▶ **Impossible to test** 

- ▶ **Monster files**
size > 1500 lines 
- ▶ **Impossible to test** 
- ▶ **Components too coupled** 

- ▶ Monster files
size > 1500 lines 😢
- ▶ Impossible to test 😐
- ▶ Components too coupled ☹
- ▶ Hard to understand (and review) classes. 😠







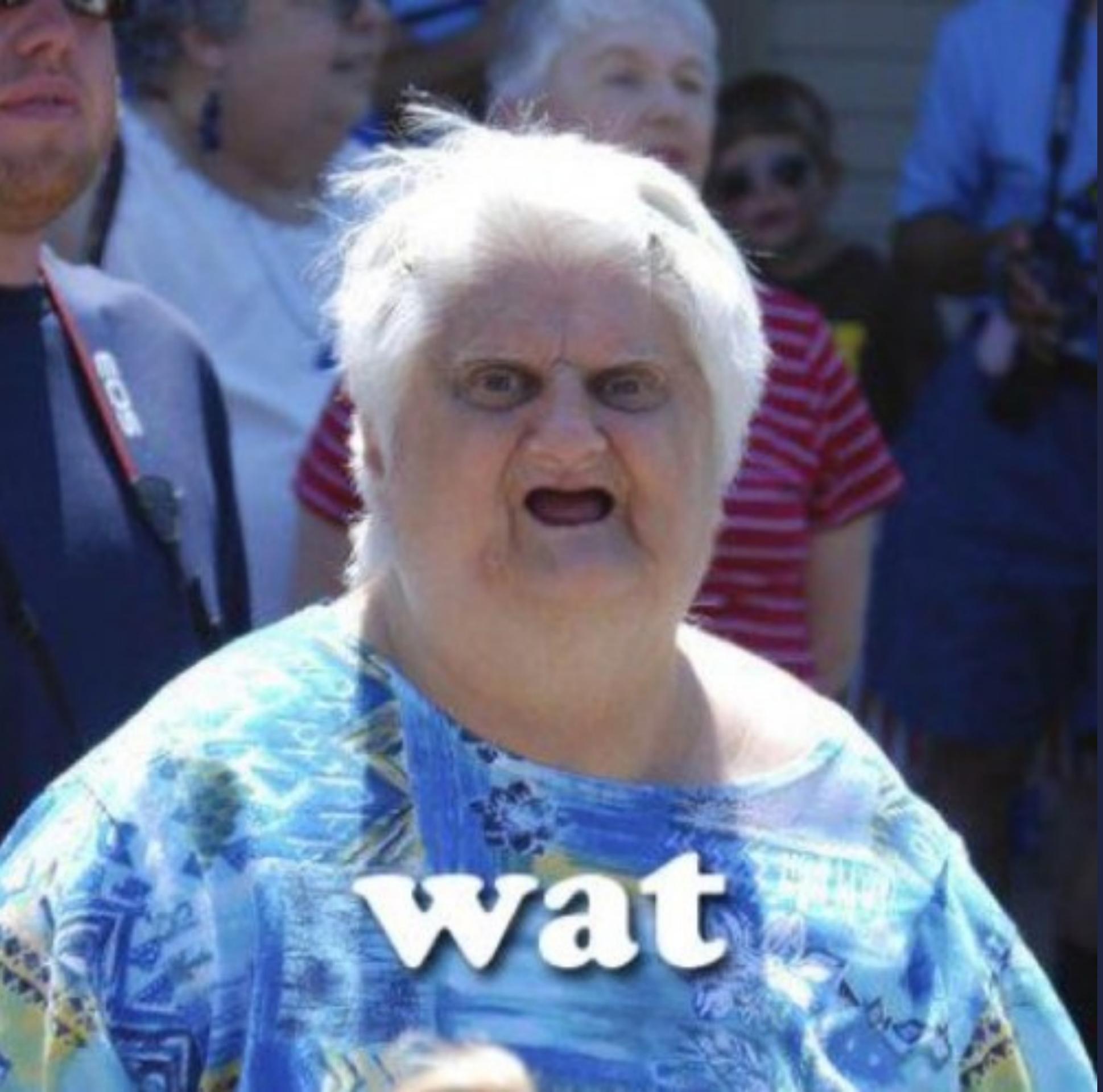
CLEAN ARCHITECTURE
INDEPENDENT FROM FRAMEWORKS, UI,
DATABASE AND EXTERNAL ENTITIES. EASY
TO TEST

► Who will persist the data?

- ▶ Who will persist the data?
- ▶ And the API interaction?

- ▶ Who will persist the data?
- ▶ And the API interaction?
- ▶ Who controls the navigation?

Viper



INDEX

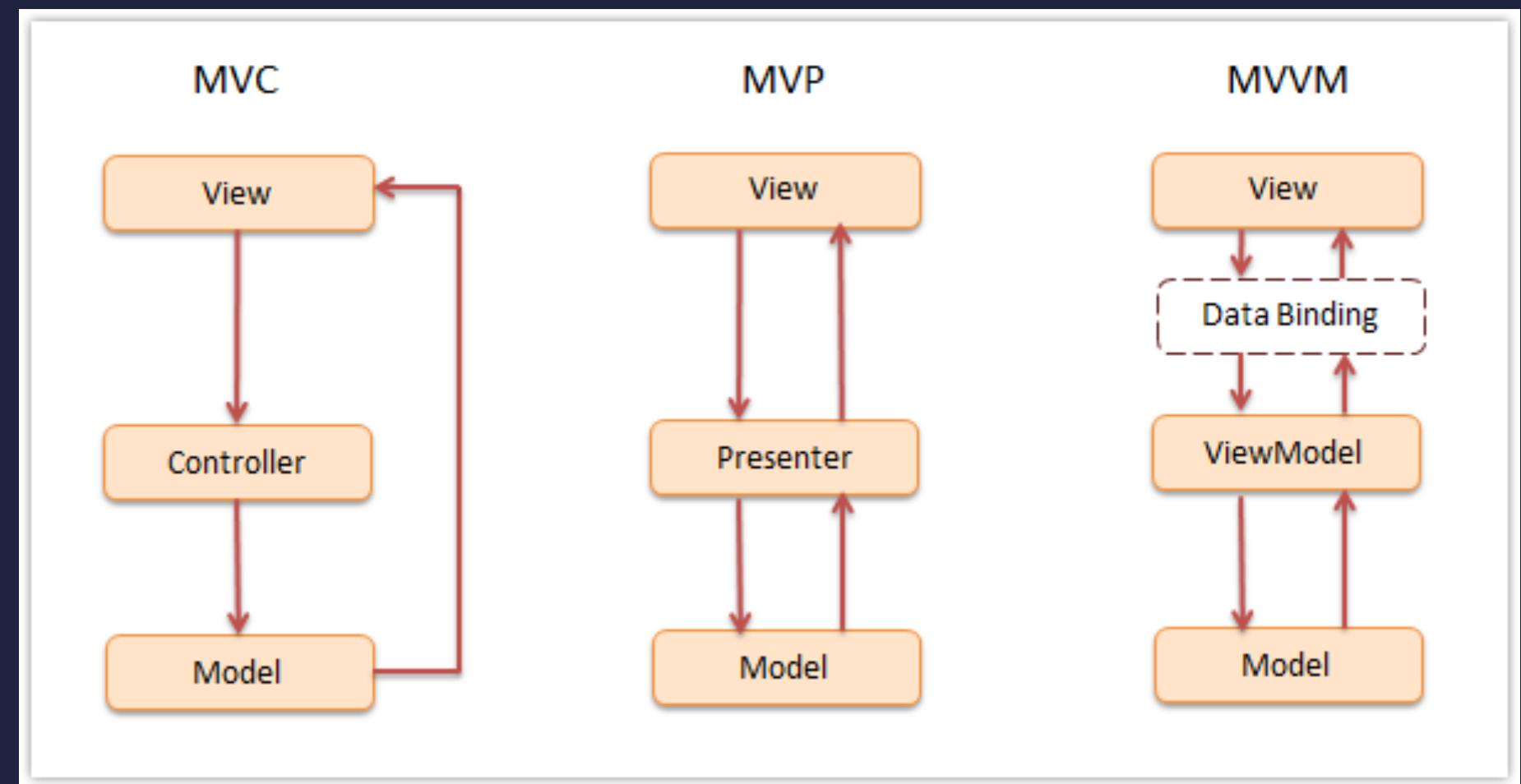
- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

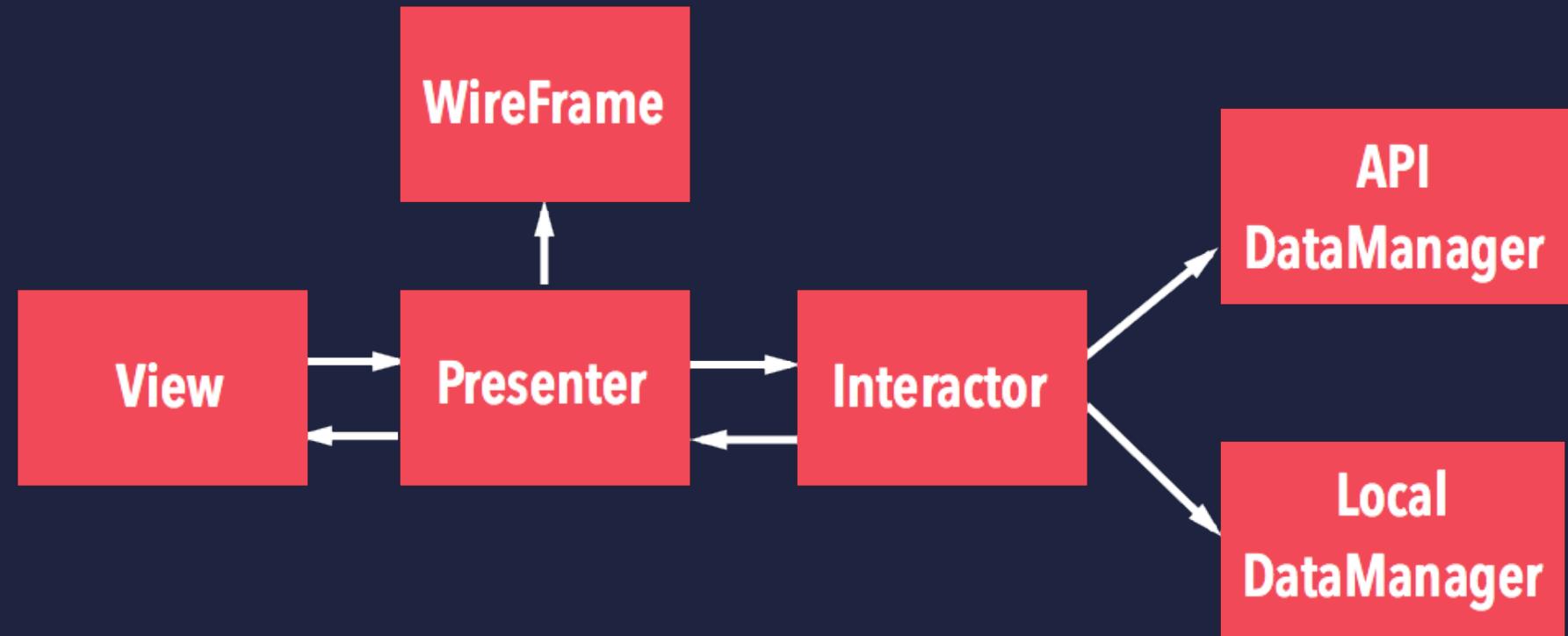
VIPER

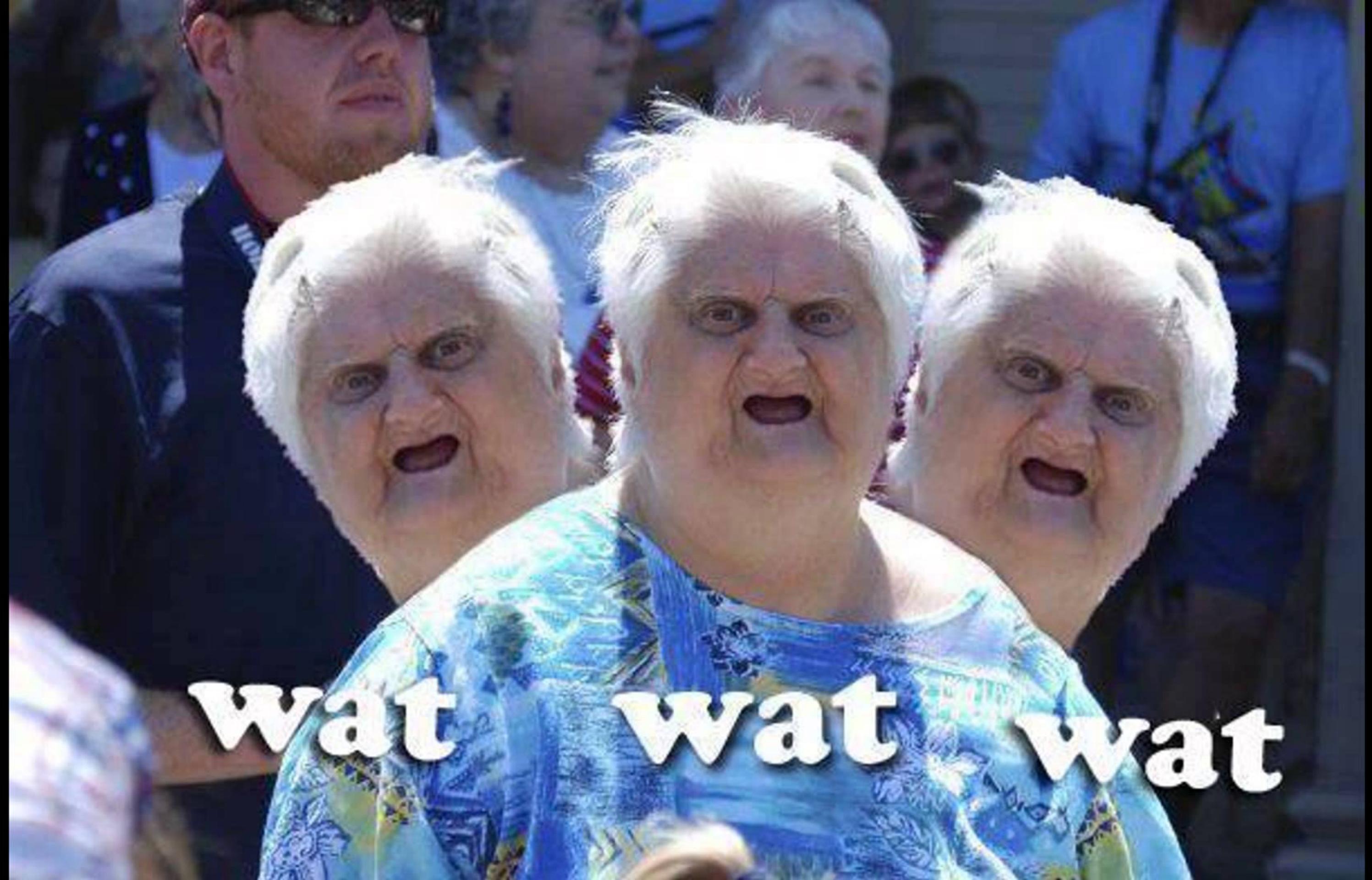
VIEW, INTERACTOR, PRESENTER, ENTITY, AND ROUTING

OBC.IO - VIPER

WHY VIPER?

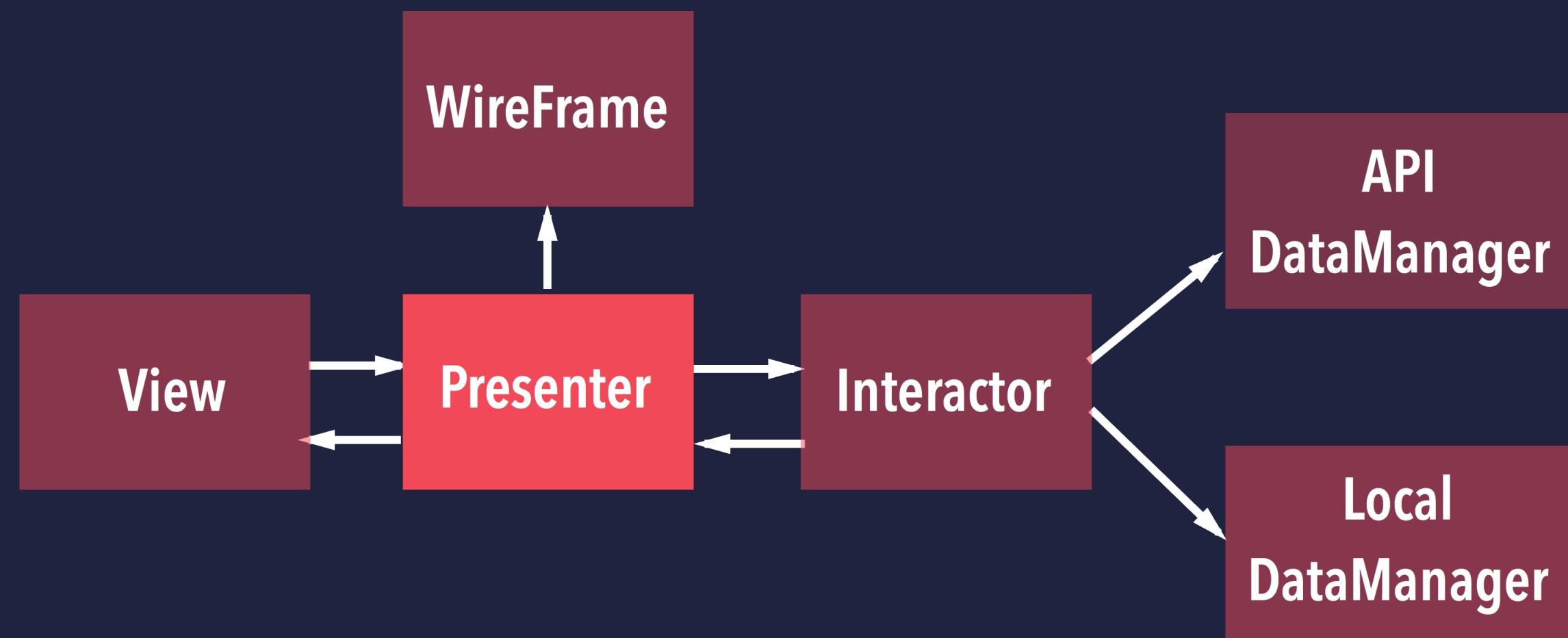






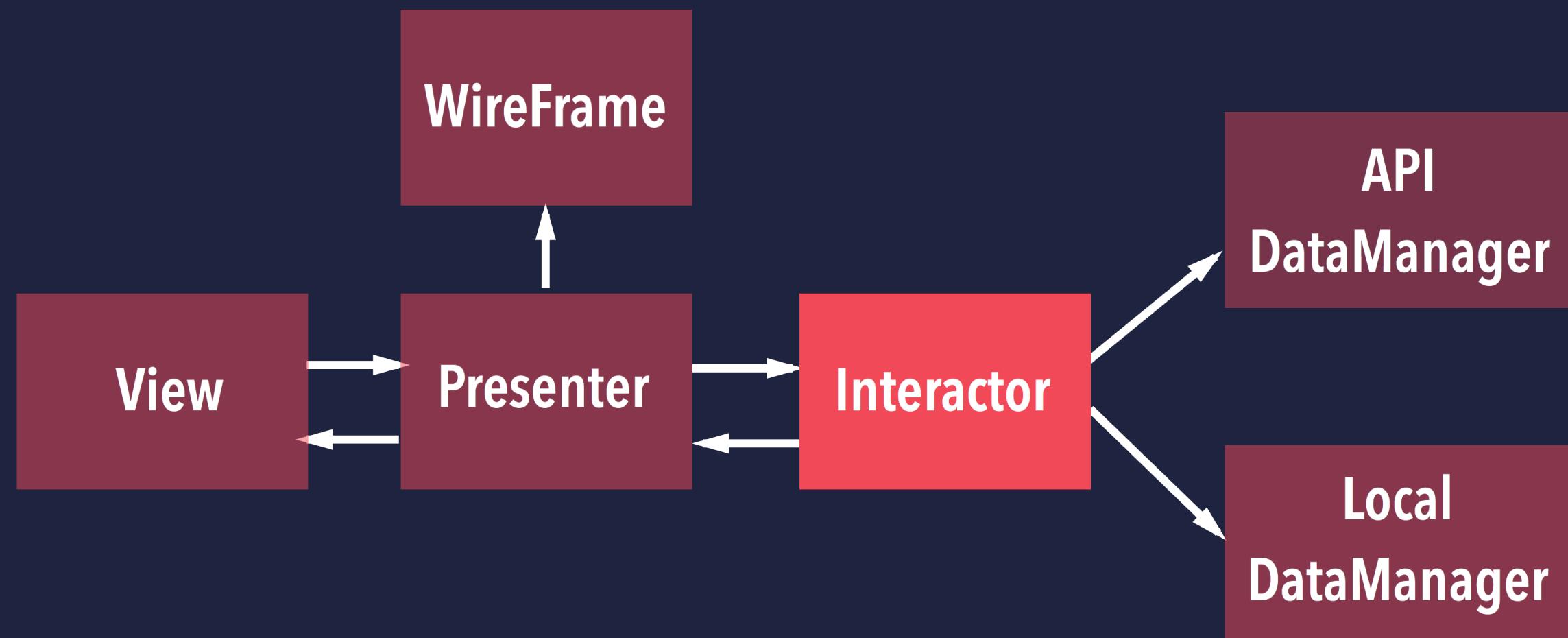
VIEW

- ▶ Shows the content received from the presenter
- ▶ Notifies user's actions to the presenter
- ▶ The presenter doesn't know anything about UI



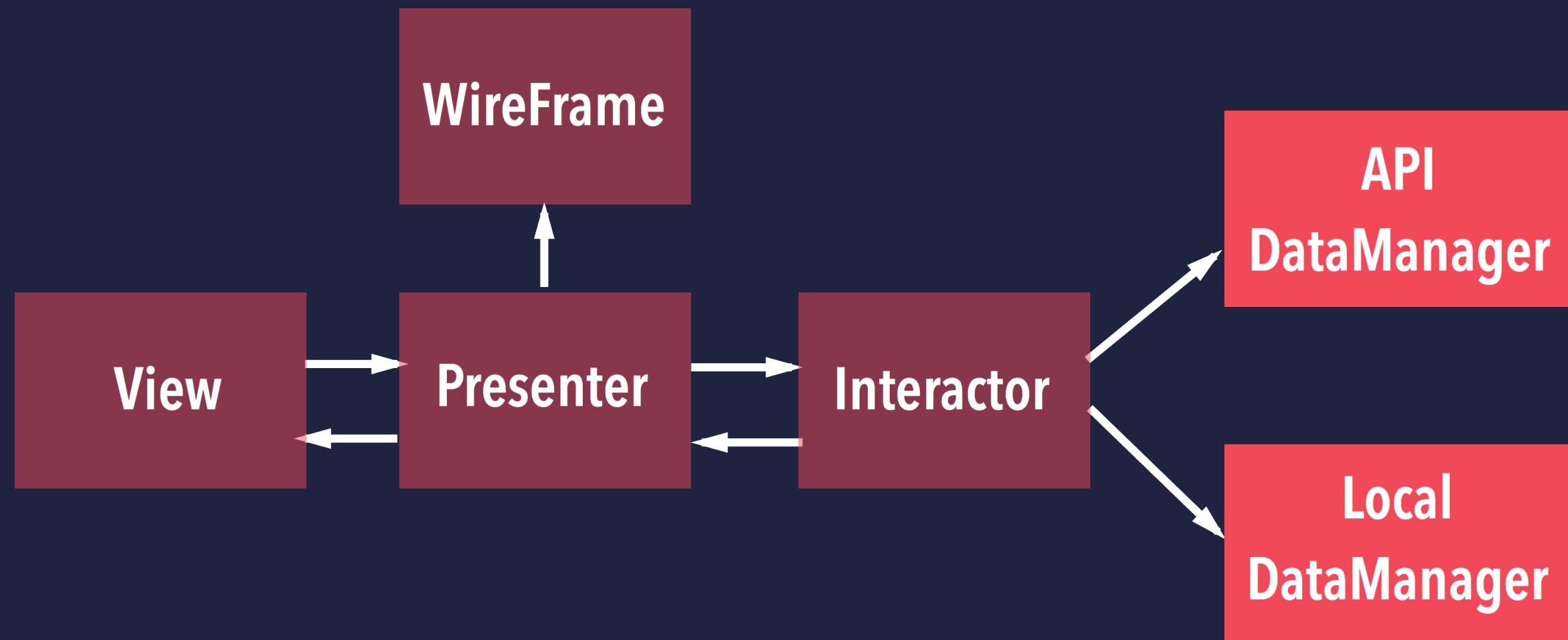
PRESENTER

- ▶ Includes the logic to format the view
- ▶ Gets the data from the interactor
- ▶ Receives actions from the view and traduces them into: *Navigation actions (wireframe) and Interactor requests*



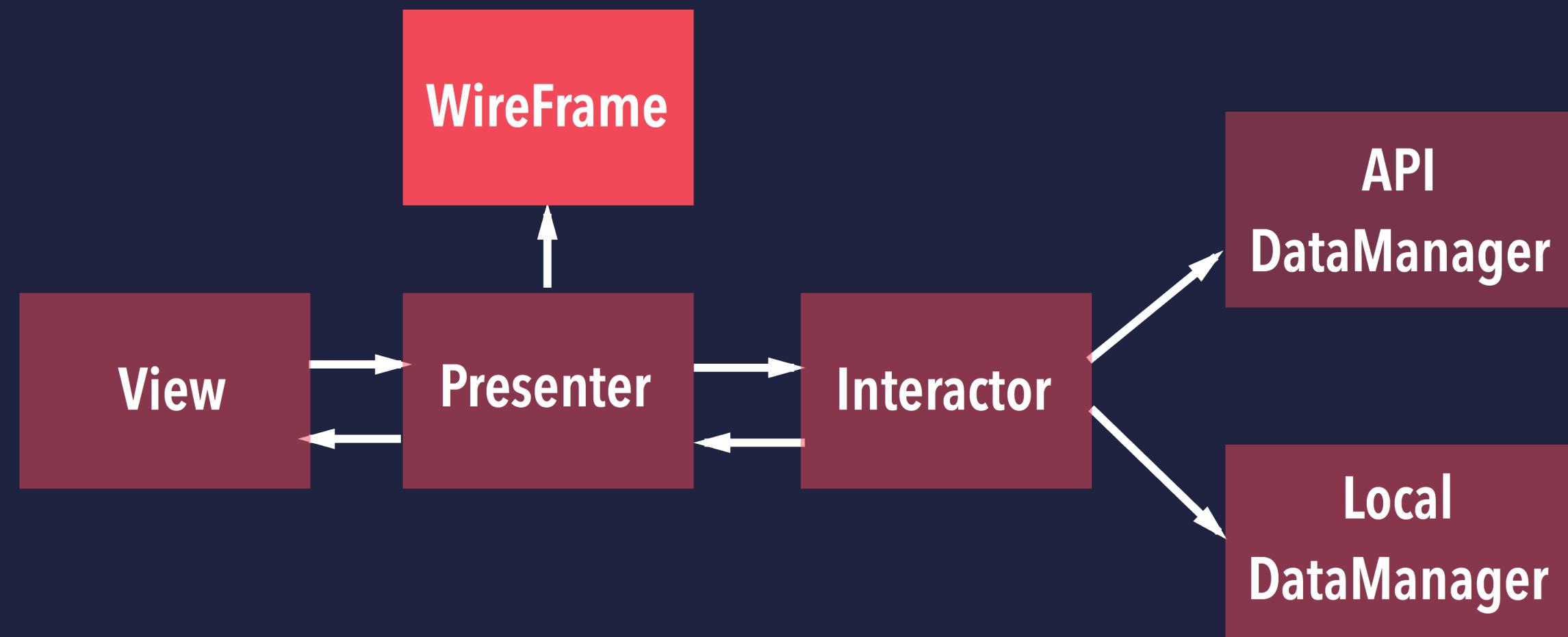
INTERACTOR

- ▶ Associated to an unique use case of the application
 - ▶ Works with PONSO entities
 - ▶ Coordinates both data managers



DATAMANAGER

- ▶ Provider of entities for the Interactor
- ▶ Responsible of the persistence (Web and Local)
- ▶ The entities don't know about how to persist themselves



WIREFRAME

- ▶ Initializes the VIPER module
- ▶ It knows how to navigate
- ▶ Delegate of transitions animations

KEEP IN MIND

PROTOCOLS
DEPENDENCY INVERSION
INTERFACE SEGREGATION
(SOLID)

STRONG/WEAK

BE CAREFUL WITH RETAIN CYCLES 

```
@interface TweetDetailViewController: UIViewController
@property (nonatomic, strong) id <TweetDetailPresenterInput> presenter;
@end

@interface TweetDetailPresenter: NSObject<TweetDetailPresenterInput>
@property (nonatomic, weak) id <TweetDetailViewInput> view;
@end

@implementation TweetDetailPresenter
- (void)sendTweet:(NSString*)tweet
{
    __weak typeof(self) welf = self;
    [self.view showLoader];
    [self.interactor sendTweetWithCompletion:^(NSError *error) {
        [welf.view hideLoader];
        if (!error) [welf.wireframe moveBack];
    }];
}
@end
```

ENTITIES

DON'T PASS NSMANAGEDOBJECTS!

USE PONSOS INSTEAD

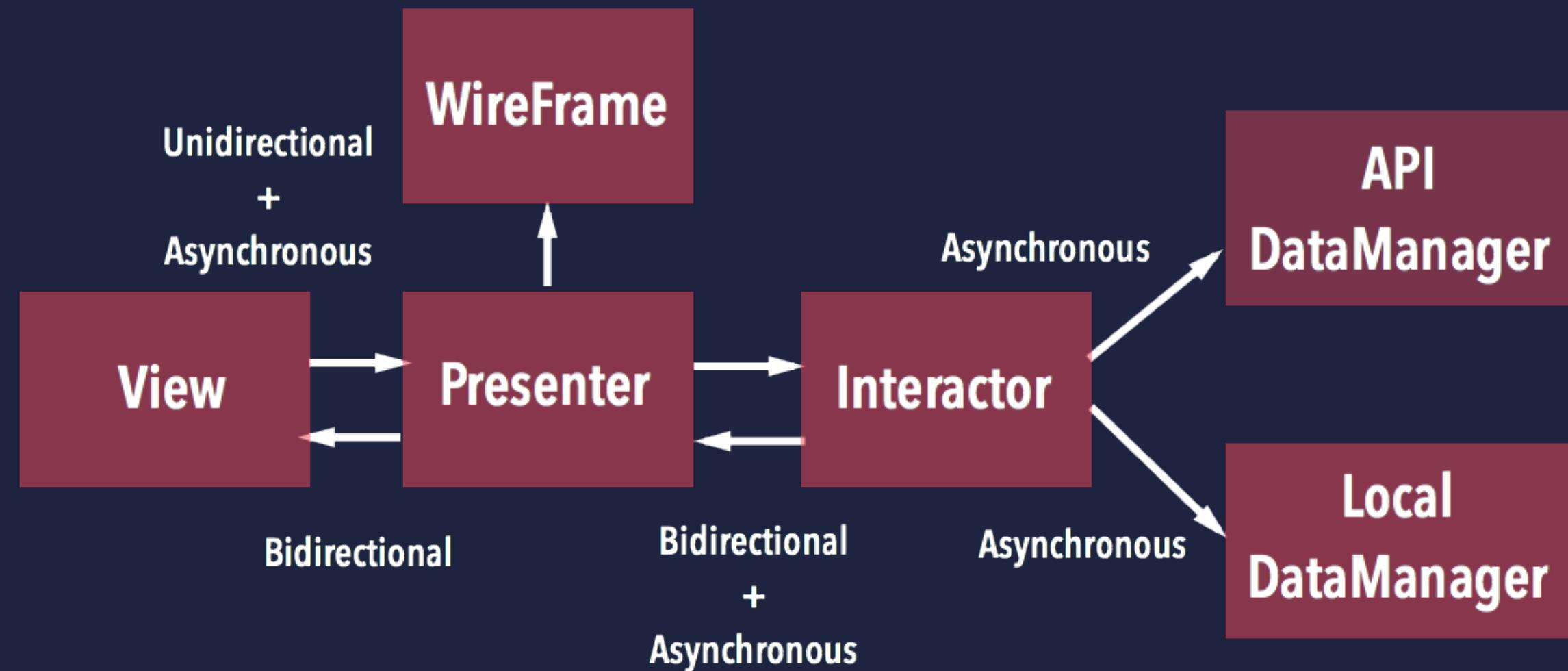
```
@interface TweetEntity: NSObject  
@property (nonatomic, strong) NSString *body;  
@property (nonatomic, strong) NSString *authorName;  
@property (nonatomic, strong) NSDate *creationDate;  
+ (TweetEntity*)tweetEntityFromTweet:(Tweet*)tweet;  
@end
```



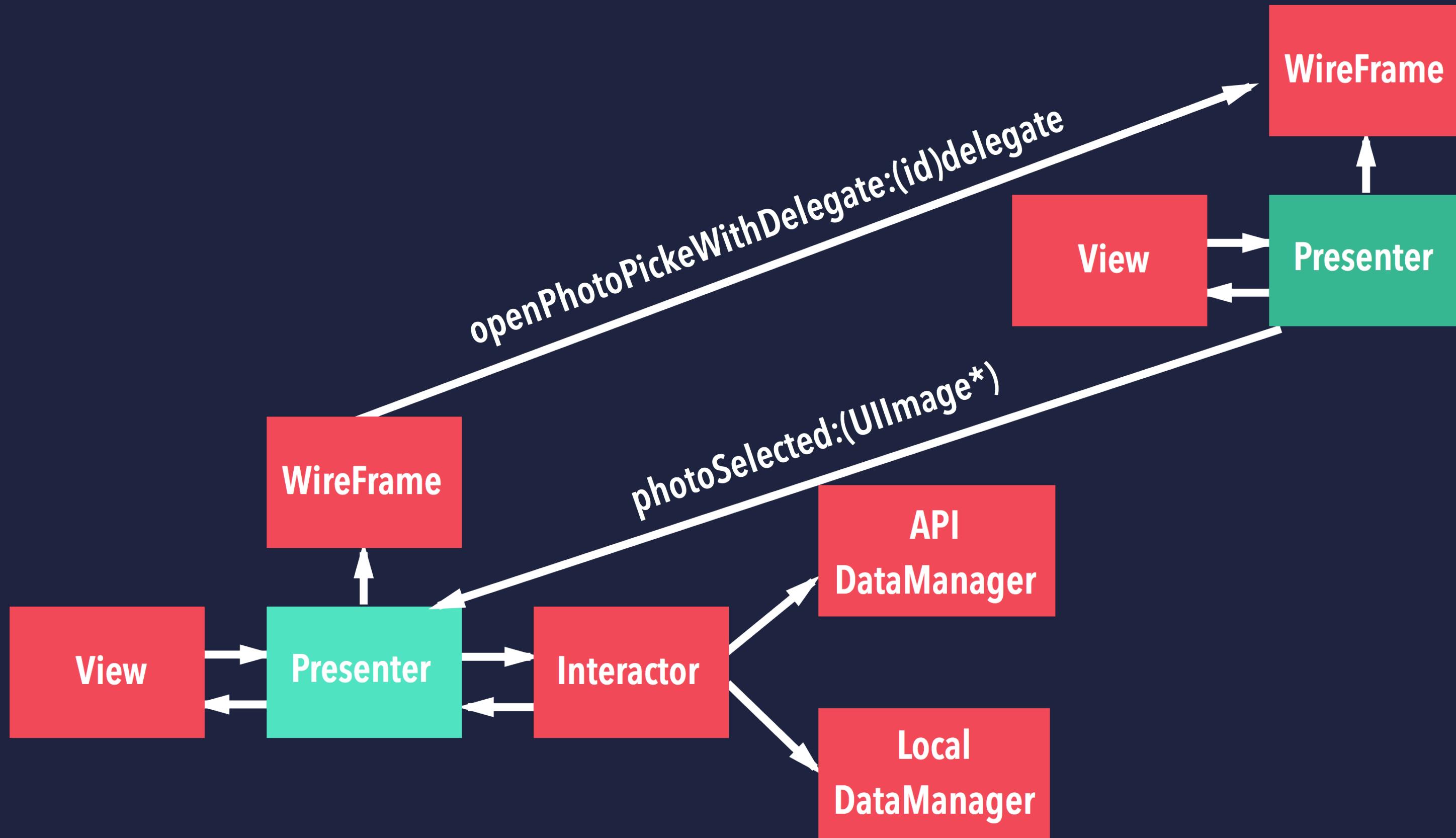
INDEX

- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

**INNER
COMMUNICATION**



**OUTER
COMMUNICATION**

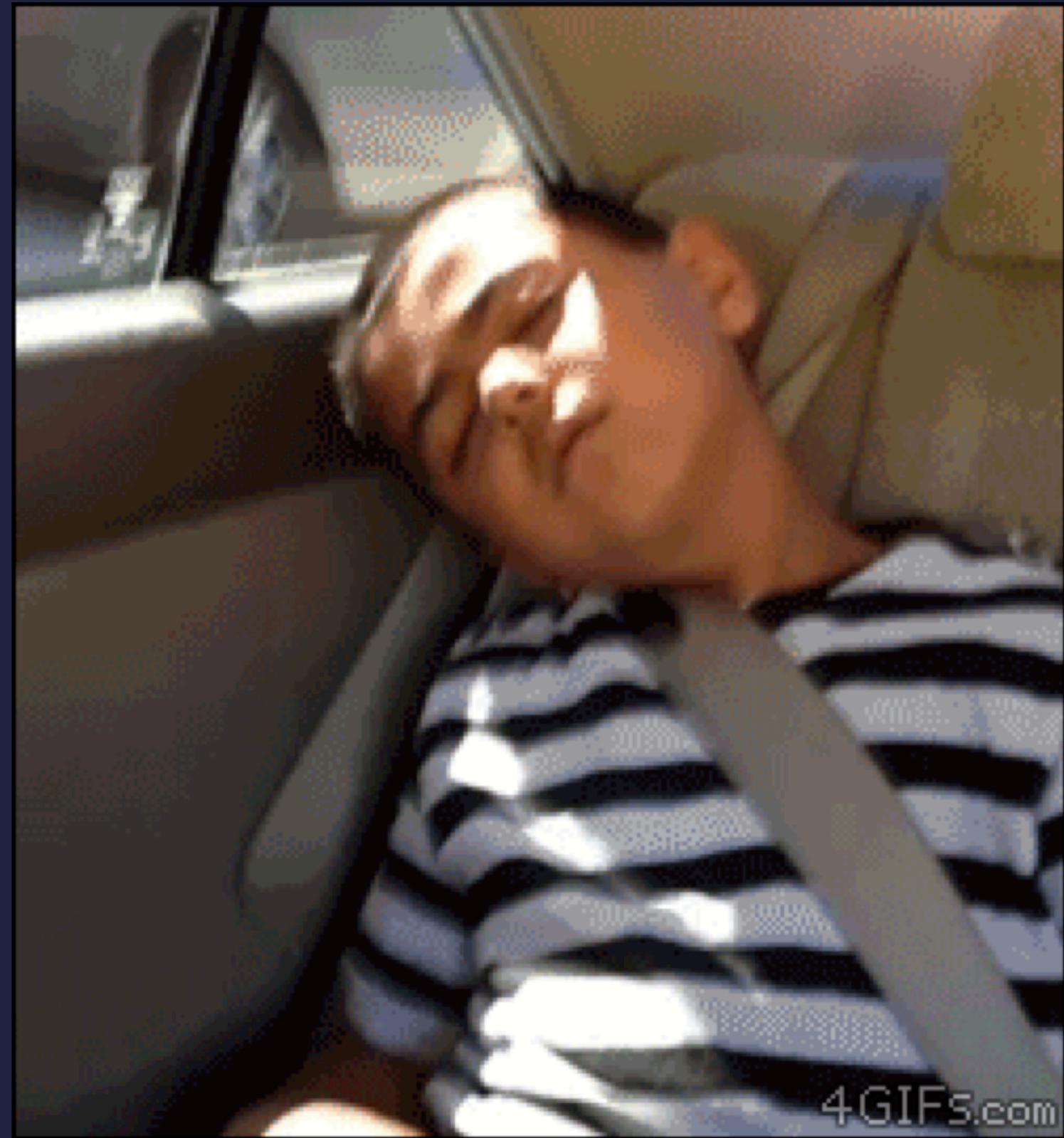






WAIT...

TESTING

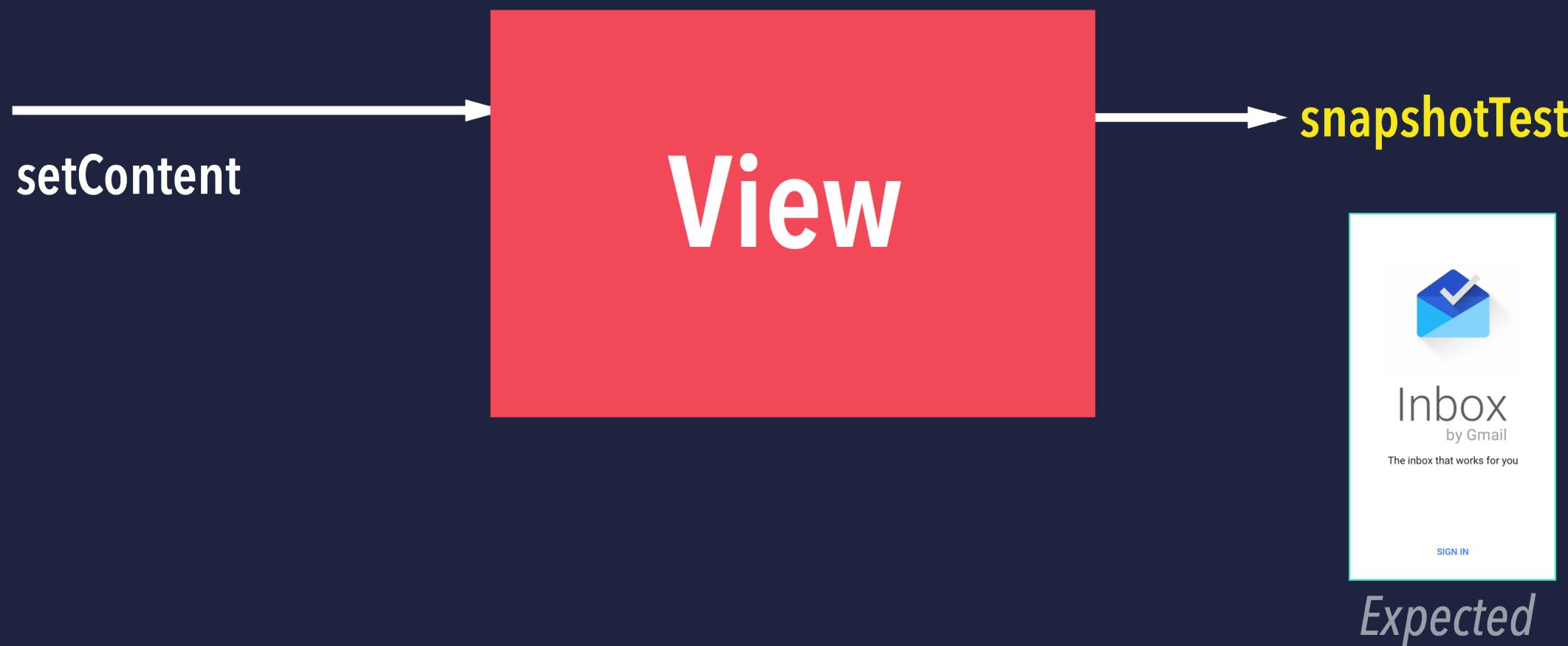


4GIFS.com

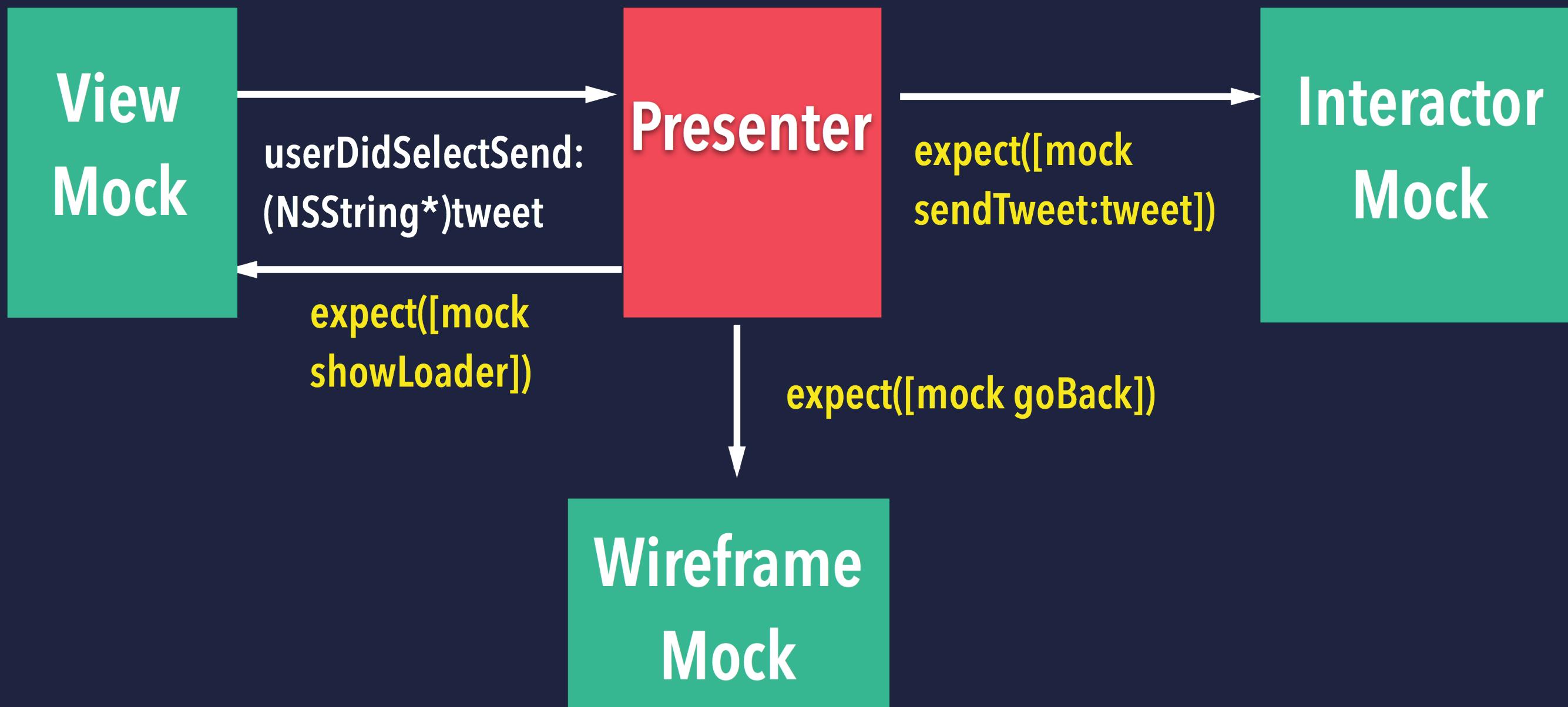
INDEX

- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

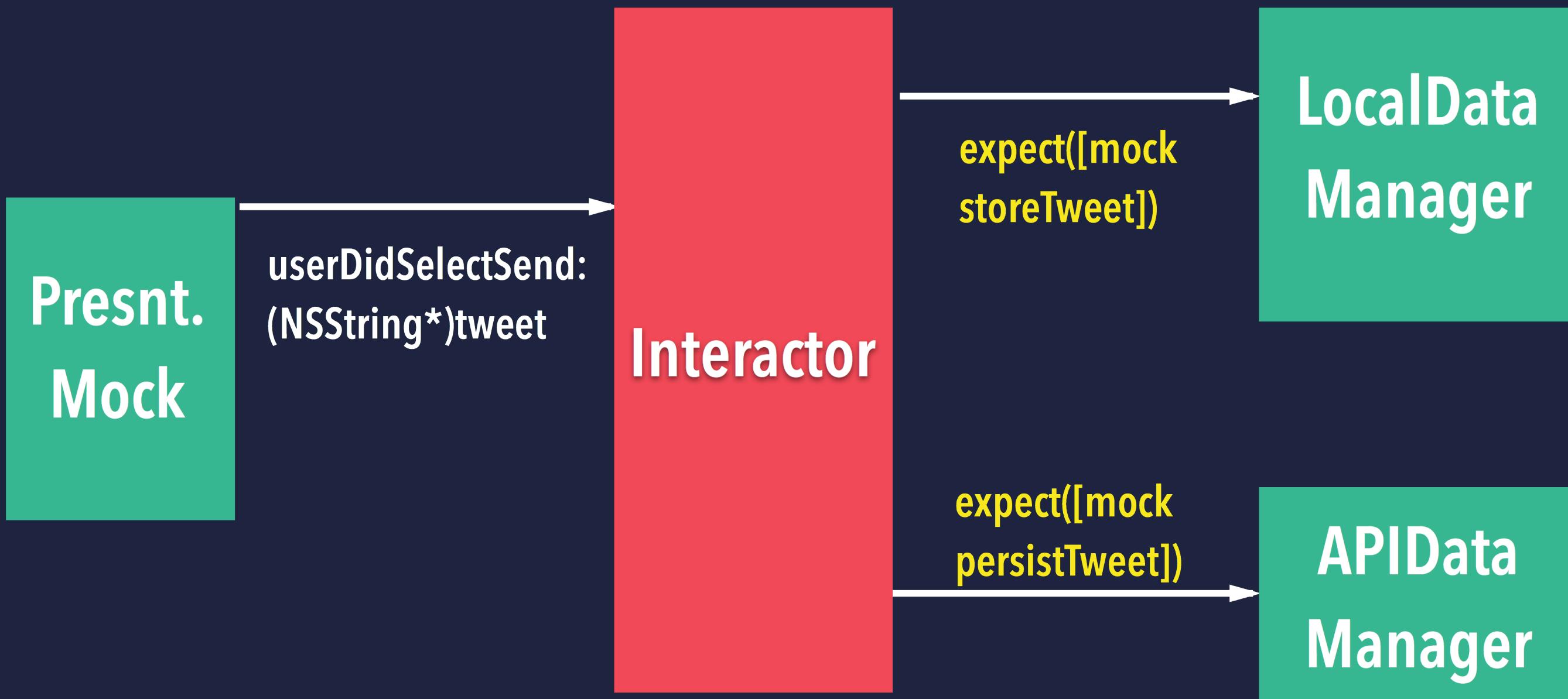
TESTING VIEW



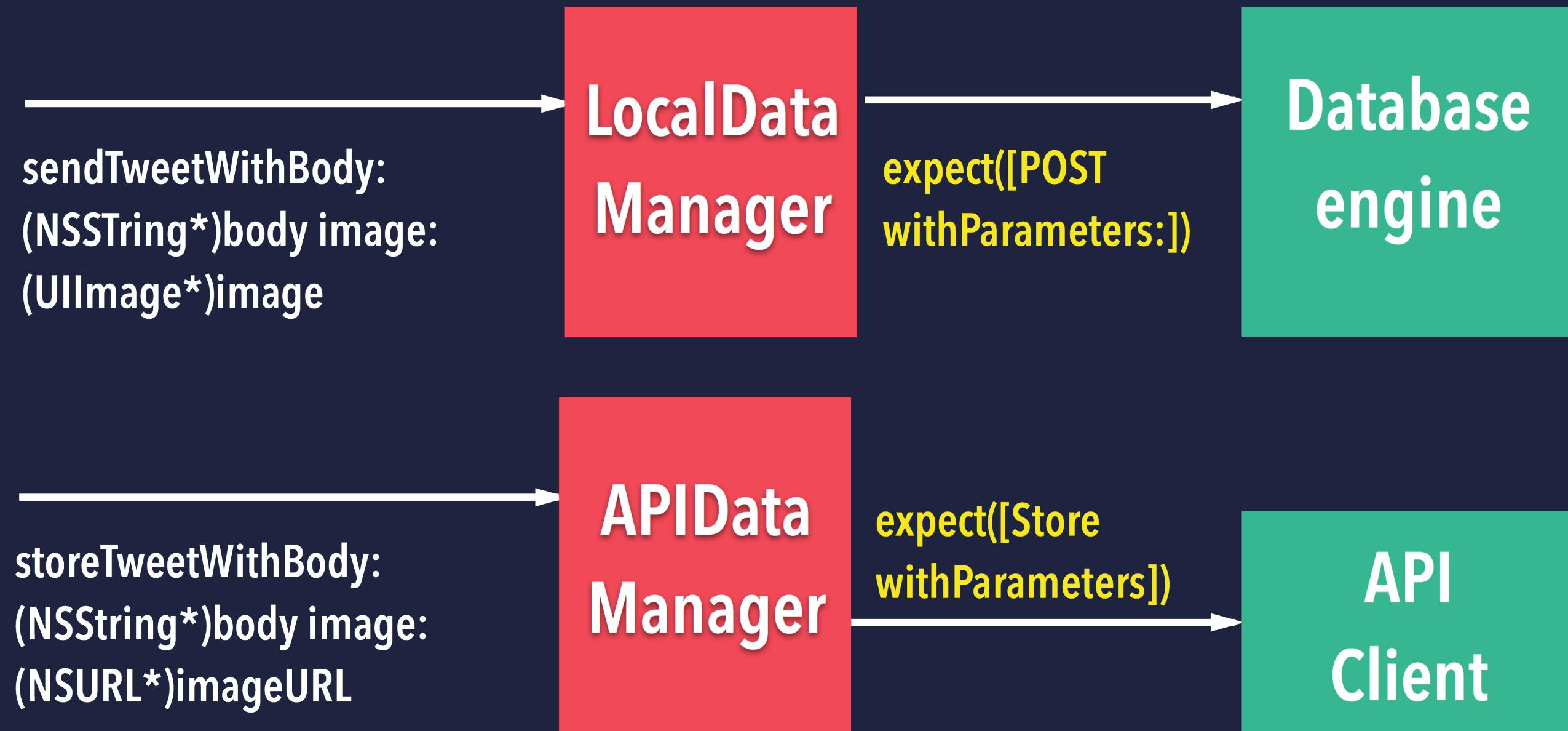
TESTING PRESENTER



TESTING INTERACTOR



TESTING DATAMANAGER



Demo

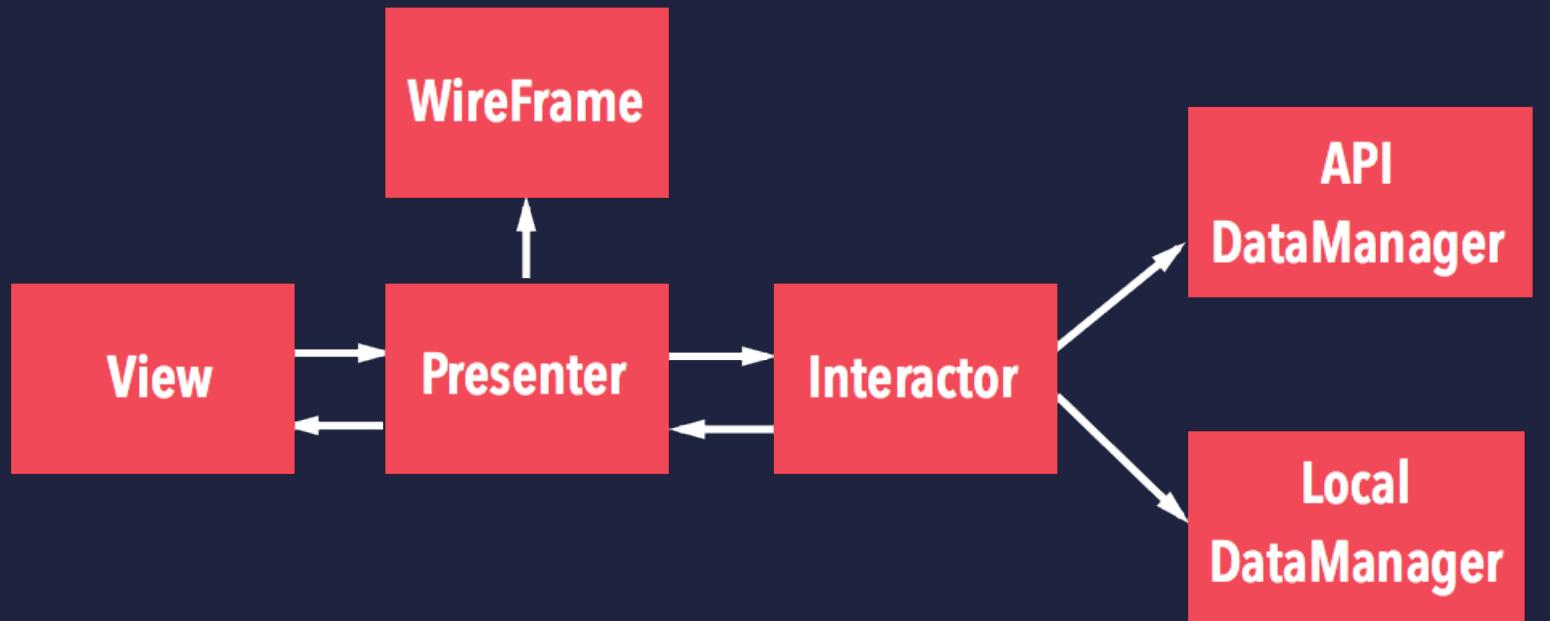
TWITTER APP

Login and Home views

WRITTEN 100% IN SWIFT

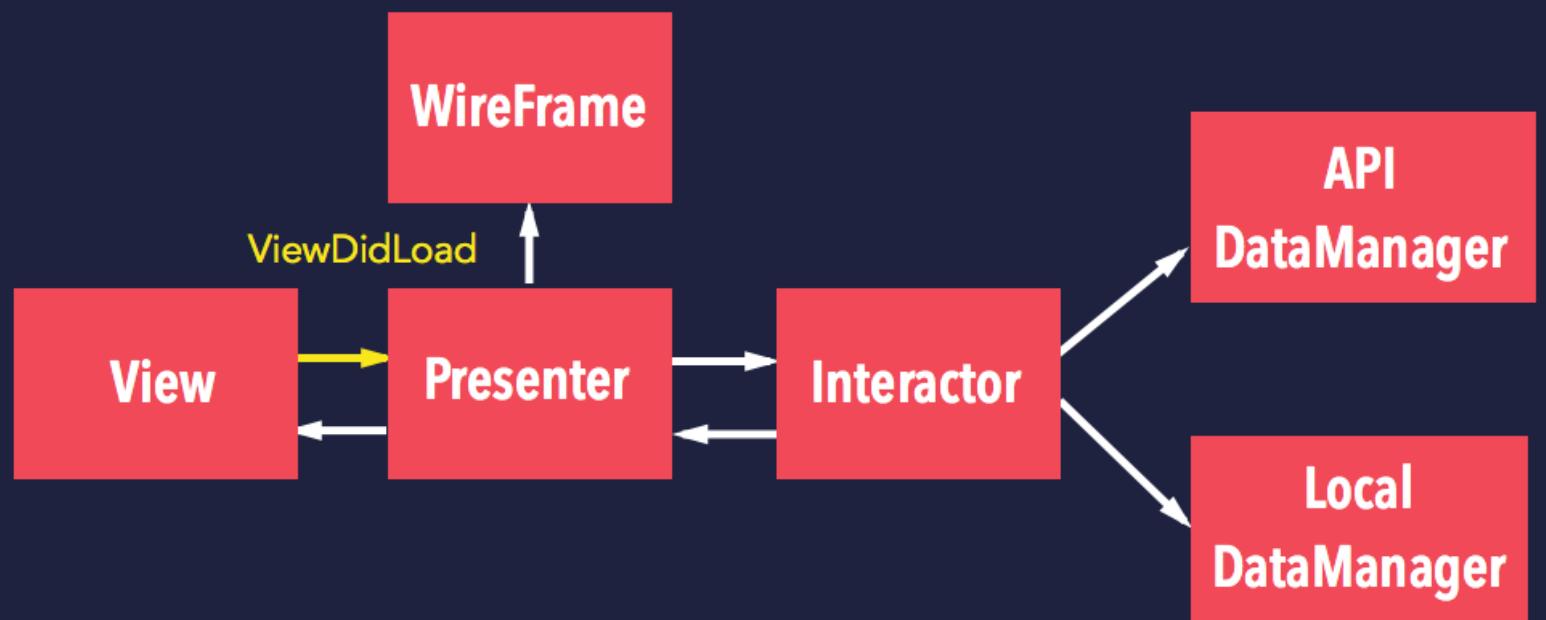
[GITHUB.COM/PEPIBUMUR/VIPER-MODULE-GENERATOR](https://github.com/pepibumur/VIPER-Module-Generator)

HANEKE, SUGARRECORD, SWIFTER, PURELAYOUT, PROGRESSHUD



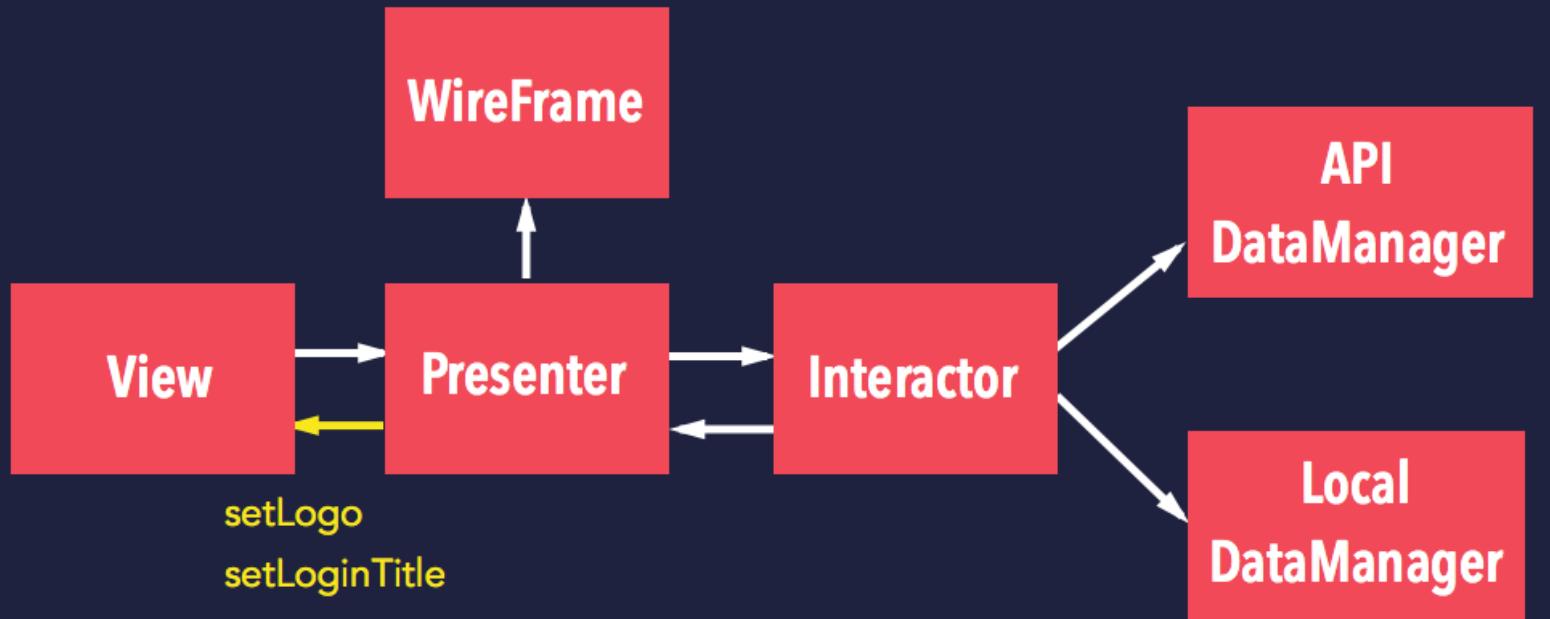
LOGIN FLOW

- ▶ The VIPER module is initialized and presented by the *Wireframe*



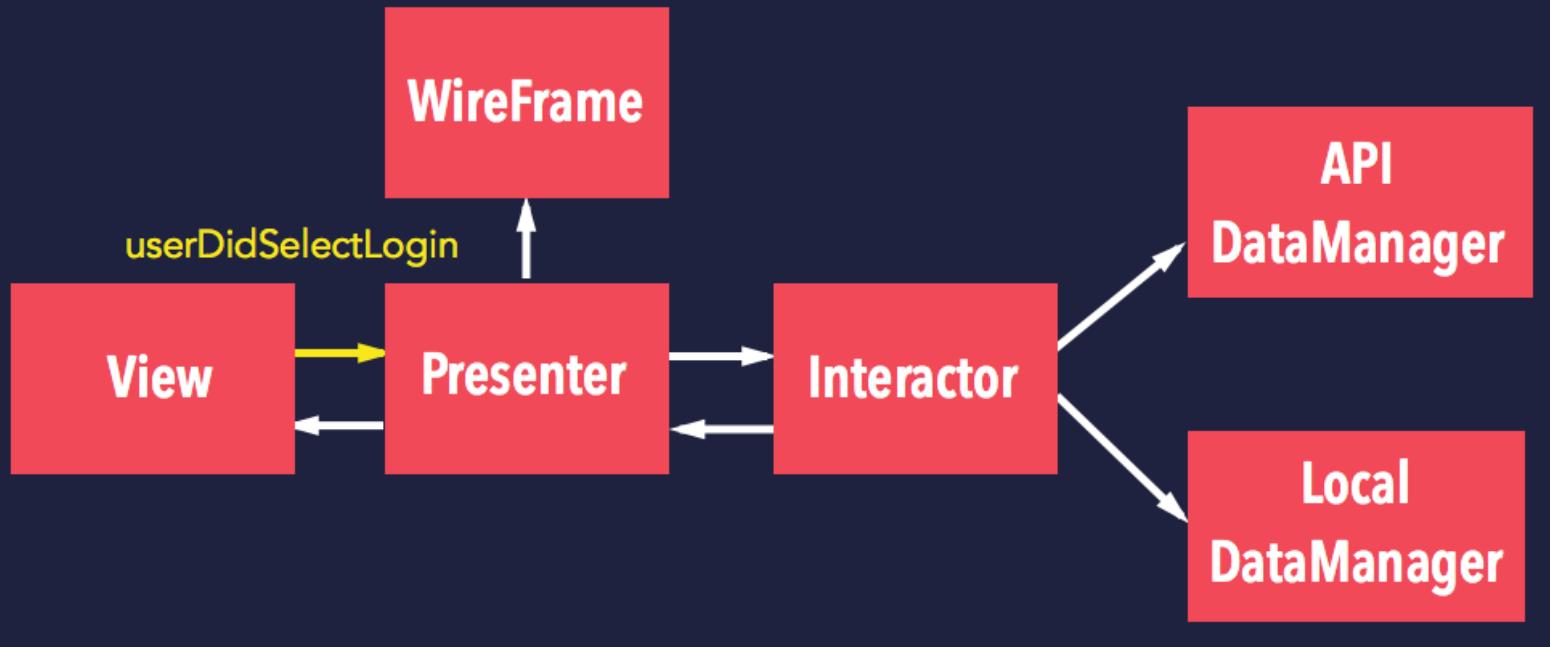
- ▶ The view notifies that DidLoad to the *Presenter*

```
override func viewDidLoad() {  
    self.setupSubviews()  
    self.setupConstraints()  
    self.setNeedsStatusBarAppearanceUpdate()  
    self.presenter?.viewDidLoad()  
}
```



► The *Presenter* formats the View's content

```
func viewDidLoad()
{
    self.view?.setLoginTitle("Login Twitter")
    self.view?.setLogo(UIImage(named: "twitter_logo")!)
}
```

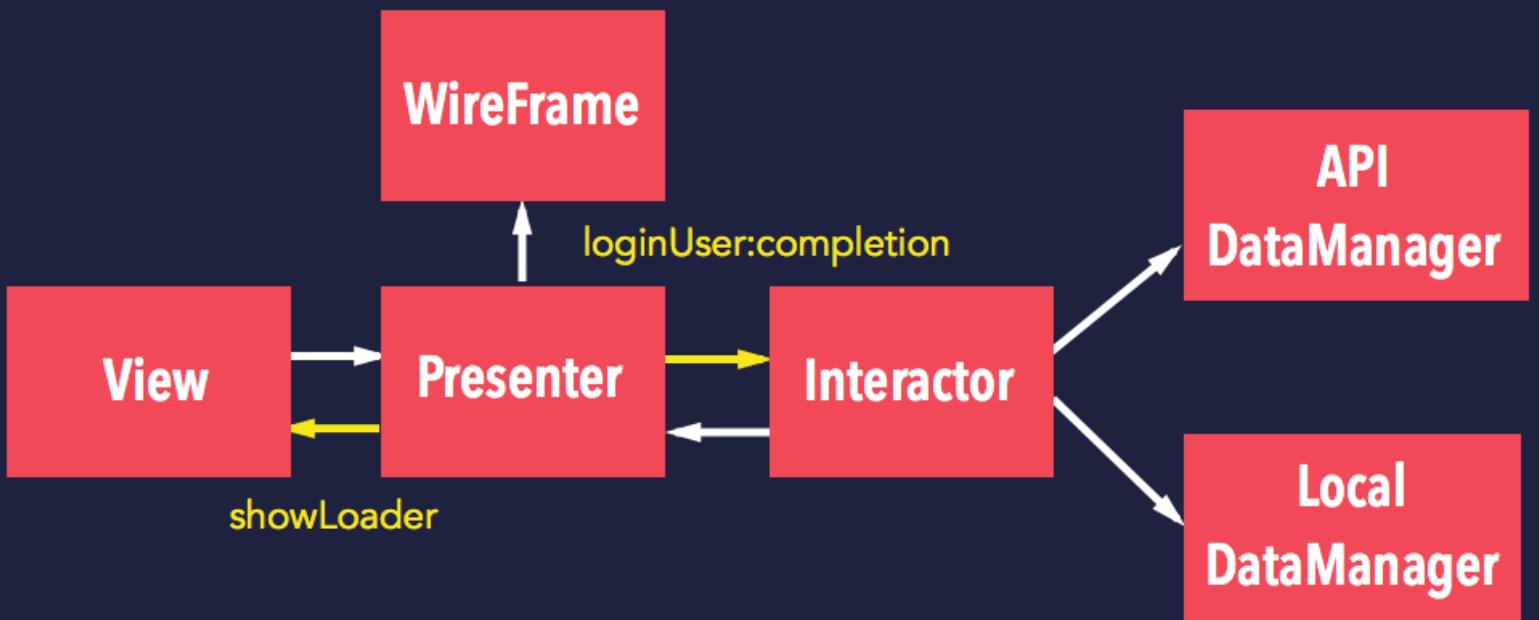


When the user taps on Login

► **The View notifies the Presenter**

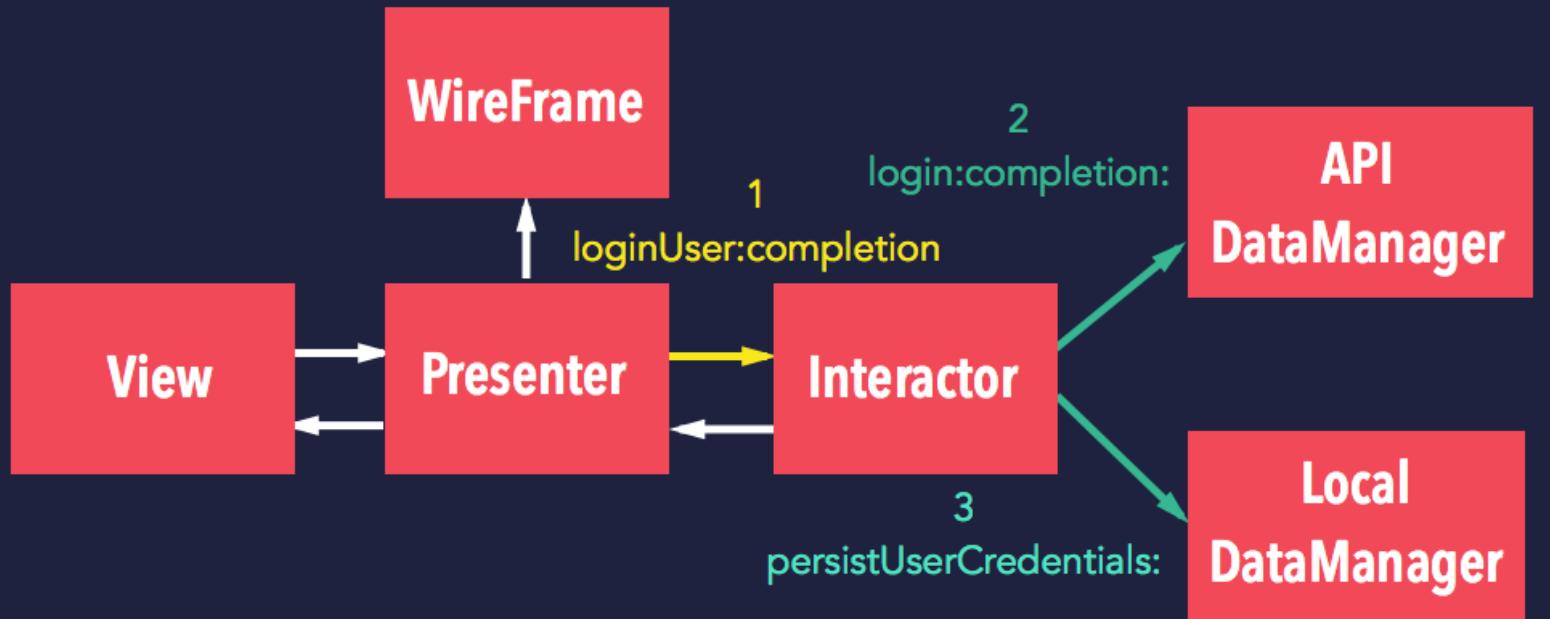
```
func userDidSelectLogin(sender: AnyObject)
{
    self.presenter?.userDidSelectLogin()
}
```

The *Presenter*:



- ▶ Tells the *View* to show a loader
- ▶ Asks the *Interactor* for Login

```
func userDidSelectLogin()  
{  
    self.view?.showLoader()  
    self.interactor?.login() { [weak self] (error: NSError?) -> () in  
        if error != nil {  
            // What should we do here?  
        }  
        else {  
            self?.view?.hideLoader()  
            // And here?  
        }  
    }  
}
```



The *Interactor*:

- ▶ Login **the user through the *APIDataManager***
- ▶ Persists **the user's credentials using the *LocalDataManager***

```
func login(completion: (error: NSError?) -> ())
{
    self.APIDataManager?.login({ [weak self] (error: NSError?, credentials: TwitterLoginItem?) -> () in
        if (credentials != nil) {
            self?.localDatamanager?.persistUserCredentials(credentials: credentials!)
            completion(error: nil)
        }
        else {
            completion(error: error)
        }
    })
}
```

APIDATAMANAGER

```
func login(completion: (error: NSError?, loginItem: TwitterLoginItem?) -> ())
{
    TwitterClient.requestAccesss { (error, credentials) -> () in
        if credentials != nil {
            completion(error: nil, loginItem: TwitterLoginItem(swifterCredentials: credentials!))
        }
        else {
            completion(error: error, loginItem: nil)
        }
    }
}
```

LOCALDATAMANAGER

```
func persistUserCredentials(#credentials: TwitterLoginItem)  
{  
    TwitterAccountManager.persistAccount(fromLoginItem: credentials)  
}
```

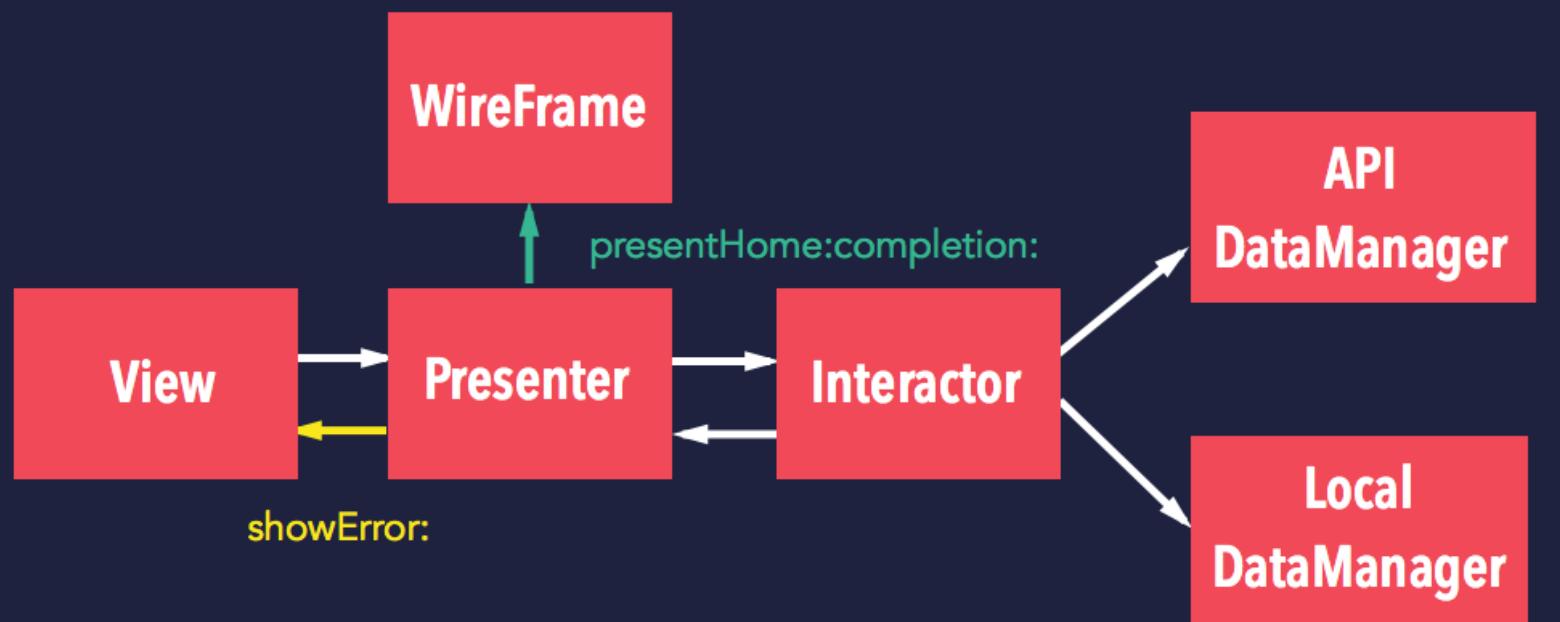
If the login fails

- The *Presenter* asks the *View* to show an error

```
func showError(let errorMessage: String)  
{  
    ProgressHUD.showError(errorMessage)  
}
```

If the login success

- The *Presenter* asks the *Wireframe* to show the home view



Demo

INDEX

- ▶ ViewControllers
- ▶ VIPER
- ▶ Communication
- ▶ Testing
- ▶ Conclusions

SOME CONCLUSIONS

- ▶ Lighter, more specific and readable classes
- ▶ Each team member can be working on a different component once the interfaces are defined
- ▶ There're no excuses for TDD 😊

TIPS

- ▶ Heavy work but you and your team will thank it
- ▶ Keep in mind the SOLID principles
- ▶ Refactor your components through iterations
- ▶ Decouple your code from the database models and data layers

RESOURCES

- ▶ VIPER Module Generator
- ▶ Objc.io post
- ▶ Mutual Mobile Engineering blog post
- ▶ Doubts/Ideas/Suggestions on Github issues
- ▶ Slides <http://bit.ly/14iWsPK>

Thank
you



DOUBTS?