HOW TO AVOID THE SPAGHETTI CODE MONSTER?

Sébastien Morel - Senior developer at **Wherecloud**

@seb_morel

december 11, 2012

FUNDAMENTAL LAWS OF SPAGHETTI CODE

1. You **can't understand** it by looking at it.
2. Repetitive code will always fall **out of synchronization**.
3. The work will always gets **harder and slower** as it progresses.
4. Fixing **bugs** creates much more others.
5. You **can't enhanced** the code if you're spending your time fixing the bugs.
6. You have less time to drink **beers** with your friends and to see your kids!
7. You start loosing your hairs!

FREQUENTLY ENCOUNTERED PROBLEMS

1. **Data** Management.

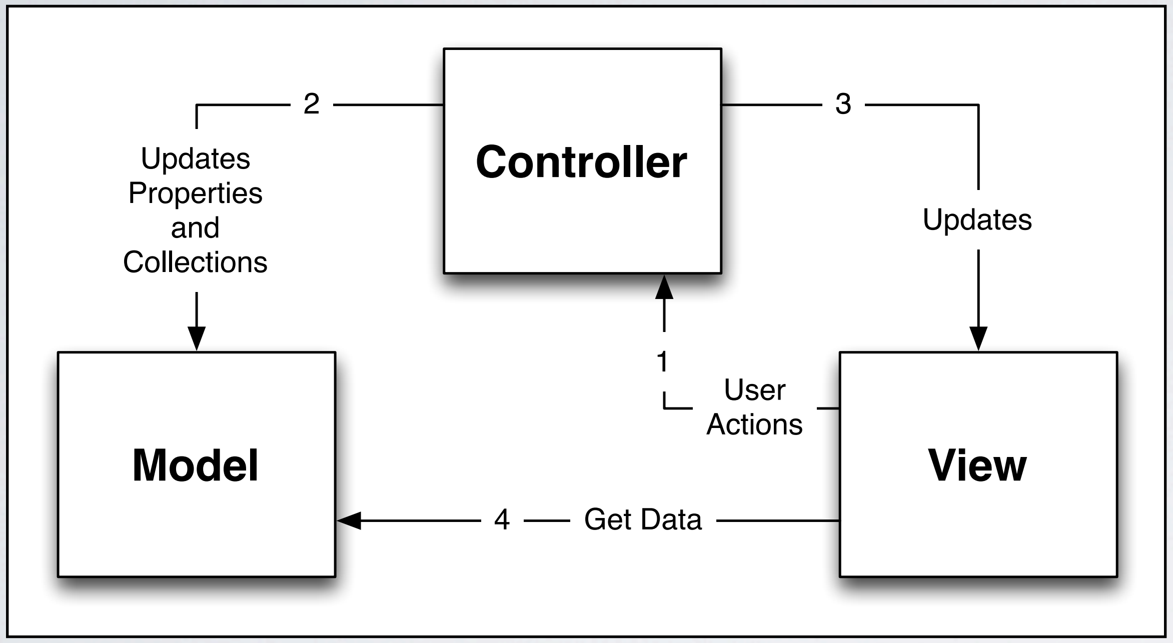
2. Asynchronous Data Updates and Views **Synchronization**.

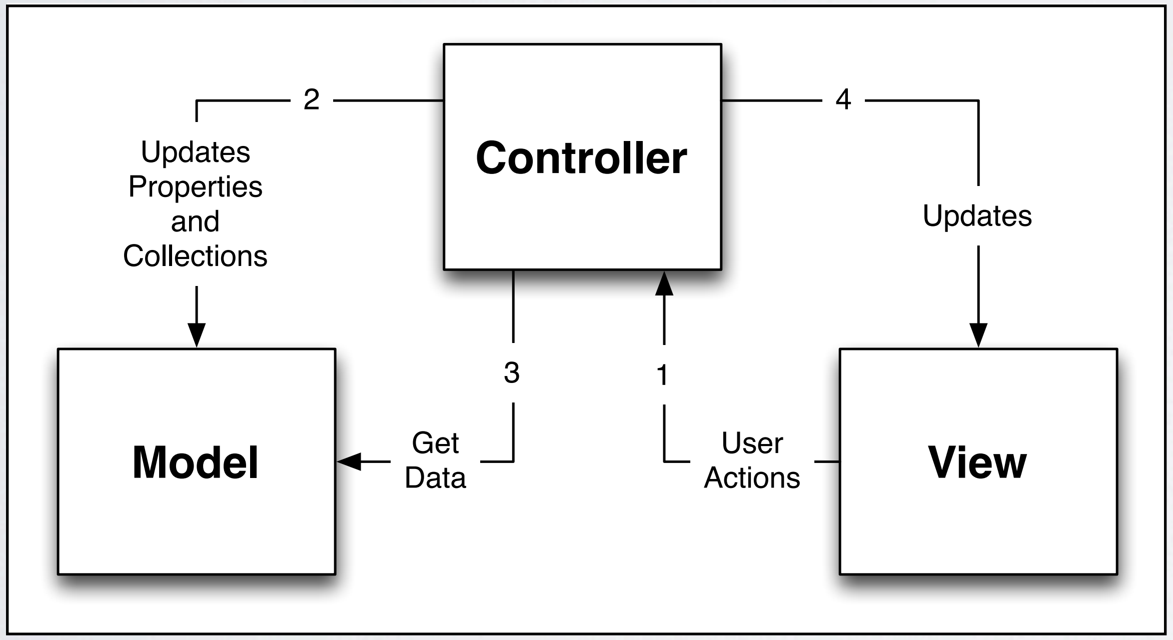
3. **Synchronization** Between Multiple Views representing the same data.

4. **Flow** Management

SEPARATION OF CONCERNS

TWO SAMPLES OF BAD MVC





Lack of Reusability

View is dependent of the model !

Lack of Synchronization

Do not refresh the views directly following user events!

Something is missing ...

THE APPLE GUIDELINES

1. Views and Models are **independent**

2. View Controller is the **Mediator**

3. **Synchronization** between multiple MVC

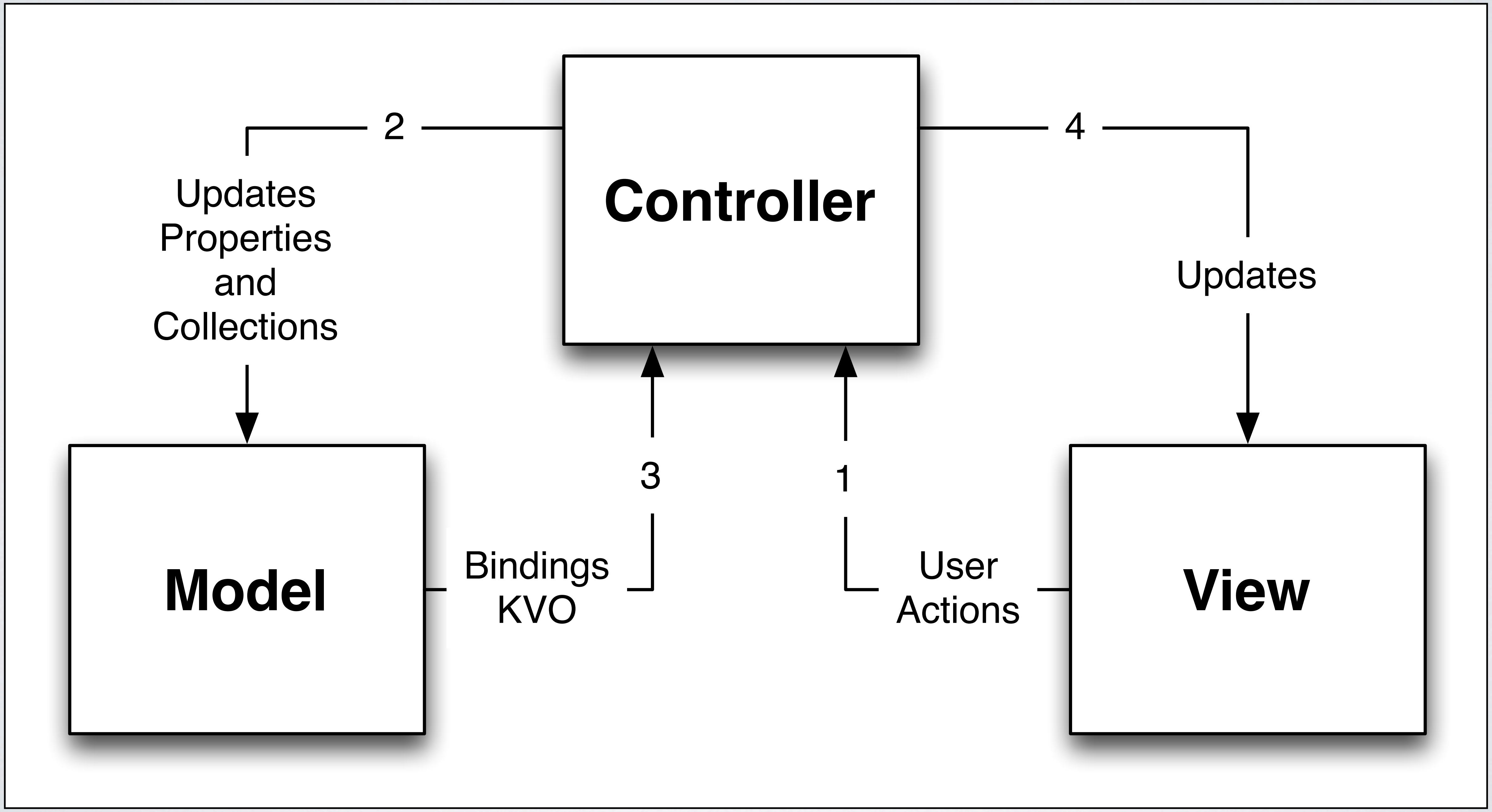
- External Controllers Dependencies : **NEVER**!

- Notification : **NOT PERFECT**!

  We need a **context** associated to the changes to take the right decisions!

- Observers / Bindings : **YES**!

DOCUMENT ORIENTED APPLICATIONS



OBSERVERS IN COCOA

Key Value Observing (KVO)

NSKeyValueObserving Protocol

```
- (void)addObserver:(NSObject *)anObserver forKeyPath:(NSString *)keyPath
            options:(NSKeyValueObservingOptions)options context:(void *)context

- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
               context:(void *)context

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context
```

```
NSString *const NSKeyValueChangeKindKey;
NSString *const NSKeyValueChangeNewKey;
NSString *const NSKeyValueChangeOldKey;
NSString *const NSKeyValueChangeIndexesKey;
```

To-One relationships Observing property changes

To-Many relationships Observing Collection changes

BINDINGS

Not Available on iOS! **Available in AppCoreKit!**

To-One Relationships

```
@interface NSObject (CKBindings)
- (void)bind:(NSString *)keyPath toObject:(id)object withKeyPath:(NSString *)keyPath;
- (void)bind:(NSString *)keyPath withBlock:(void (^)(id value))block;
@end

@interface UIControl (CKBindings)
- (void)bindEvent:(UIControlEvents)controlEvents withBlock:(void (^)())block;
@end
```

To-Many Relationships

```
@interface CKCollection (CKBindings)

- (void)bindEvent:(CKCollectionBindingEvents)events
        withBlock:(void(^)(CKCollectionBindingEvents event, NSArray* objects,
NSIndexSet* indexes))block;

@end
```

THE PROBLEM IT SOLVES

Synchronizing models and views

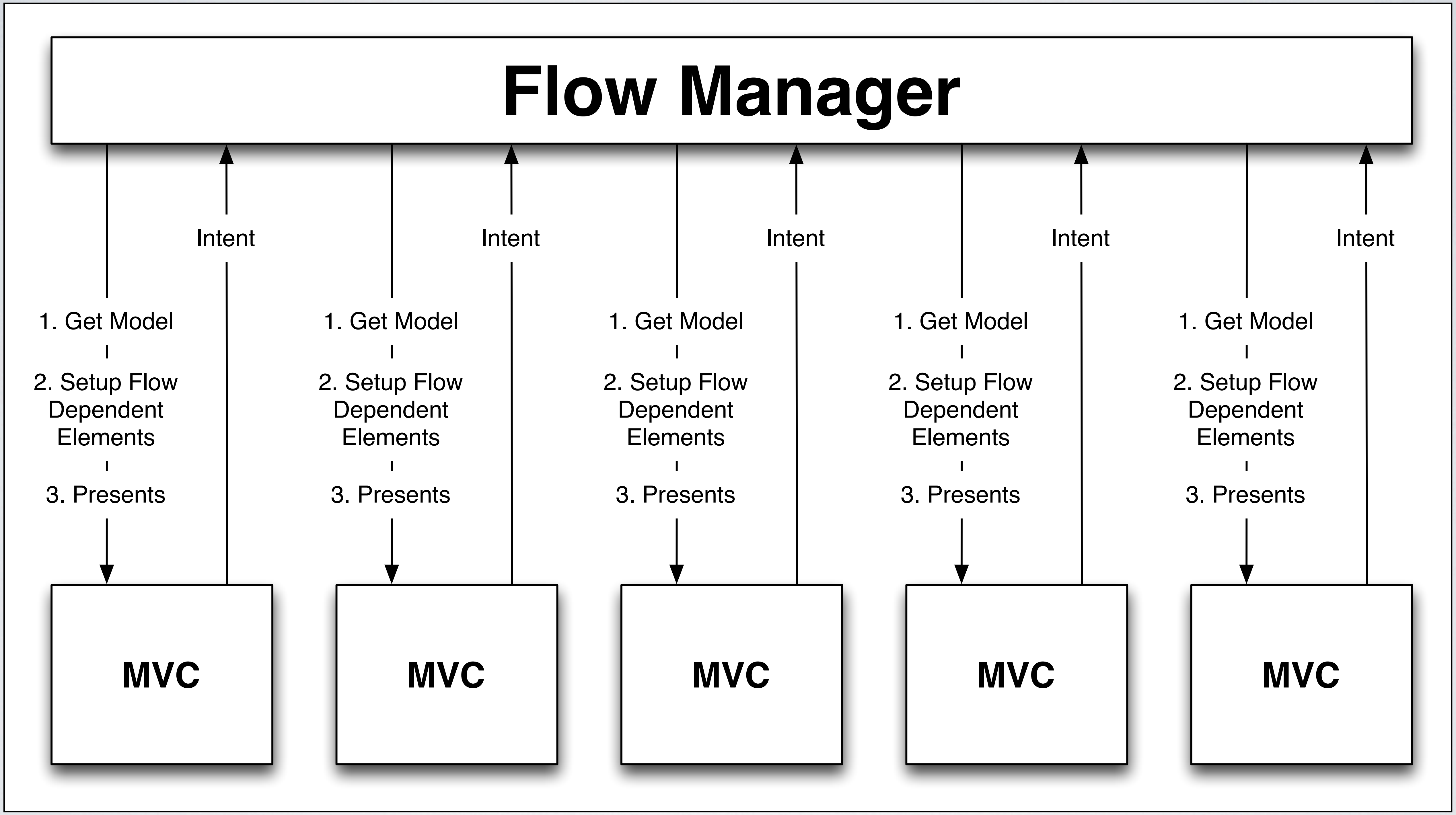
THE PROBLEMS IT DOESN'T SOLVES

Maximizing **Reusability** : Phone vs. Pad

Decoupling **Flow Management**, **Data sources** and **Data Representation**

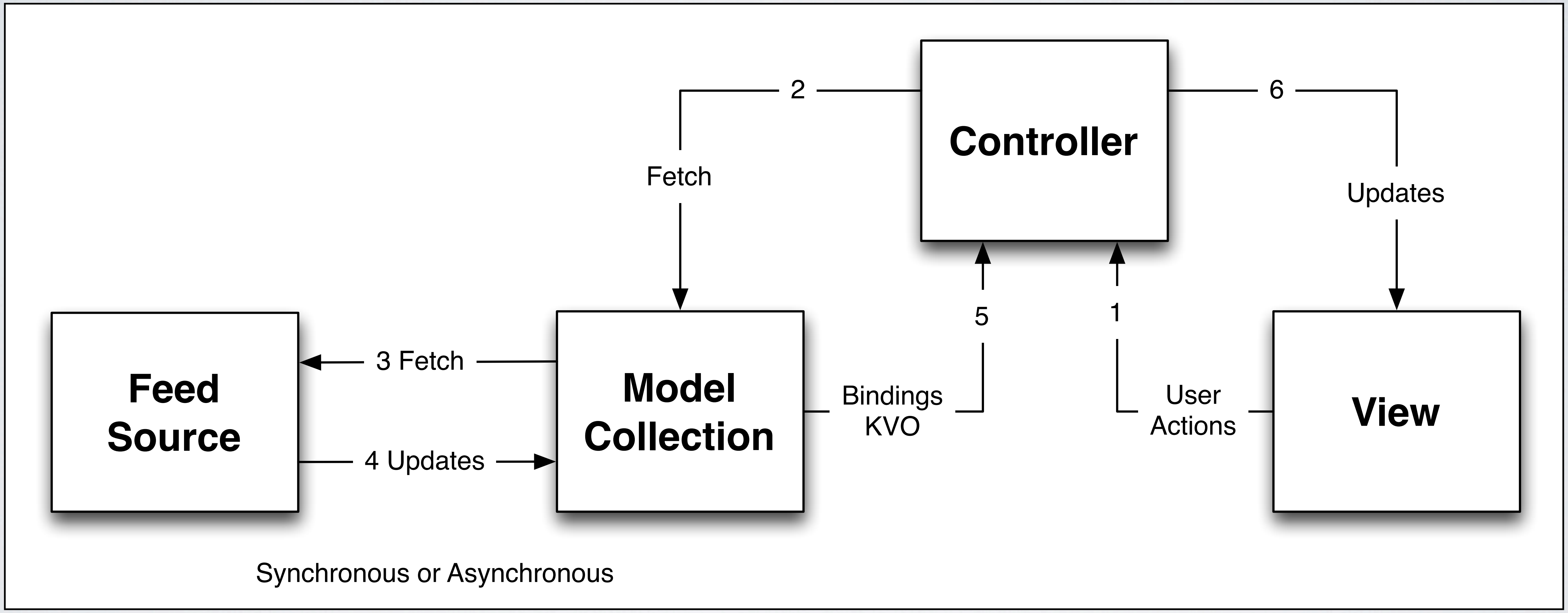
=> Controllers MUST NOT embed flow dependent logic <=
(Navigation Items, View Controllers Presentation, etc)

We need another Mediator
Flow Manager


Flow Manager
Intent
1. Get Model
2. Setup Flow Dependent Elements
3. Presents
MVC
Intent
1. Get Model
2. Setup Flow Dependent Elements
3. Presents
MVC
Intent
1. Get Model
2. Setup Flow Dependent Elements
3. Presents
MVC
Intent
1. Get Model
2. Setup Flow Dependent Elements
3. Presents
MVC
Intent
1. Get Model
2. Setup Flow Dependent Elements
3. Presents
MVC

WORKING WITH REMOTE DATA

1. Representing a remote collection of model objects



Do not pollute the view controller with data requests and transforms !

In The Flow Manager

```
//Get/Create or/and setup The Model
CKFeedSource* myPurposeFeedSource = [WebService feedSourceForMyPurpose];
CKArrayCollection* myCollection   = [CKArrayCollection collectionWithFeedSource:myPurposeFeedSource];

//Creates the controller
CKViewController* controller = [ViewControllerFactory viewControllerForMyPurpose:myCollection
                                                                           intent:^(Intent* intent) {
    switch([intent.event intValue]){
        case PurposeSelectionIntent:{
            [self presentsViewControllerForDetails:intent.object fromViewController:intent.source];
        }
        //...
    }
}];

//Setup the Flow dependent features of the controller

//Presents the controller
```

In The WebService

```
@implementation WebService

+ (CKFeedSource*)feedSourceForMyPurpose{
    CKFeedSource* feedSource = [CKFeedSource feedSource];
    feedSource.fetchBlock = ^(CKFeedSource* feedSource, NSRange range){
        //Request the remote data in range Synchronously or Asynchronously
        //On Completion
        //    => Transforms the results to a collection of model objects
        //    => [feedSource insertItems:TheModelObjects];
    };
    return feedSource;
}

@end
```

WORKING WITH REMOTE DATA

2. Requesting more details for an object

```
@implementation WebService

+ (void)performRequestForDetails:(MyObject*)object
                      completion:(void(^)(MyObject* object, NSError* error))completionBlock{
    //Request the remote data in range Synchronously or Asynchronously
    //On Completion
    //    => Maps the result in the object's properties
    //    => calls the completionBlock with the object and potential errors
}

@end
```

Controllers are binded to this object’s properties
They will refresh automatically when mapping the results

SAMPLE TWITTER CLIENT

https://github.com/smorel/CocoaHeads-Avoiding-Spaghetti-Code

Made with APPCOREKIT

http://appcorekit.net/

CONCLUSION

Do Not forget to:

- Focus on decoupling.
- Always build with reuse, synchronization and separation of concerns in mind.
- Organizes your models to help you represent it (Presentation Models vs. Business Models).
- Mock your data with fake data sources waiting for API's to be ready.

Multiple Advantages:

- Reduces side effects of "minor changes".
- Team Work & Productivity.
- Code is Simple, Readable, Comprehensible, Homogenous.

BIBLIOGRAPHY

http://theprogrammersparadox.blogspot.ca/2009/10/ok-technically-im-under-influence.html
http://theprogrammersparadox.blogspot.ca/2009/11/spaghetti-code.html

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html#//apple_ref/doc/uid/TP40002974-CH6-SW6

http://developer.apple.com/library/ios/#documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

http://martinfowler.com/eaaDev/uiArchs.html
http://martinfowler.com/eaaDev/FlowSynchronization.html
http://martinfowler.com/eaaDev/MediatedSynchronization.html

http://amix.dk/blog/post/19615

http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/
http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/

http://aspiringcraftsman.com/2009/10/05/the-arrow-anti-pattern/

https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Protocols/NSKeyValueObserving_Protocol/Reference/Reference.html