# Functional Reactive Programming on iOS

## Problem

## Apps contain multiple layers of state, state propagation is error prone

## Becomes complicated quickly e.g. UITableViewController...

| Model | Controller | View |
|---|---|---|
| State of properties | State of user interaction | State of subviews |

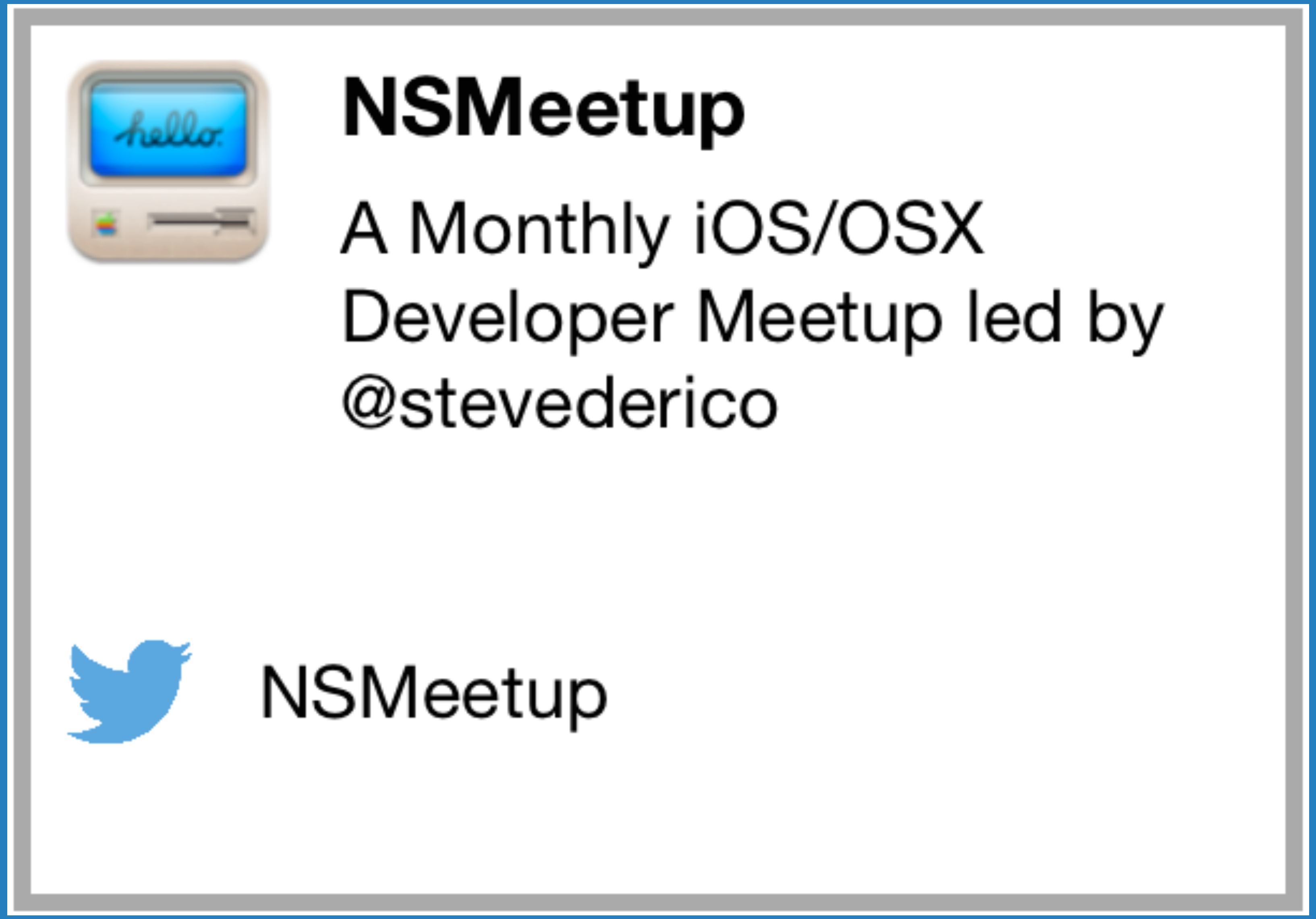
$$\begin{tikzcd} & \text{Controller} \arrow[dr, bend left] & \\ \text{Model} \arrow[ur, bend left] & & \text{View} \end{tikzcd}$$

Controller: Fetches a model

View: Renders the current state of model

## With an immutable model this approach is simple

Model →



**NSMeetup**

A Monthly iOS/OSX Developer Meetup led by @stevederico

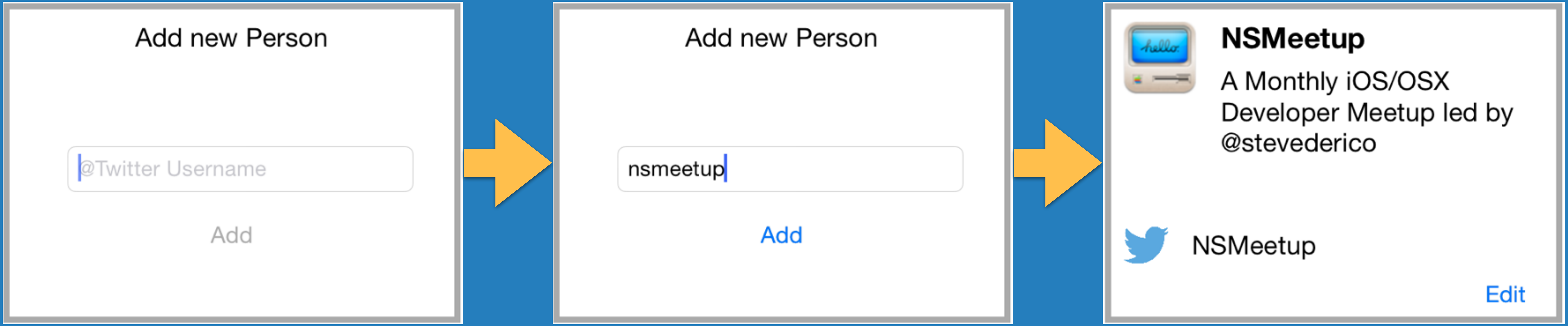
NSMeetup

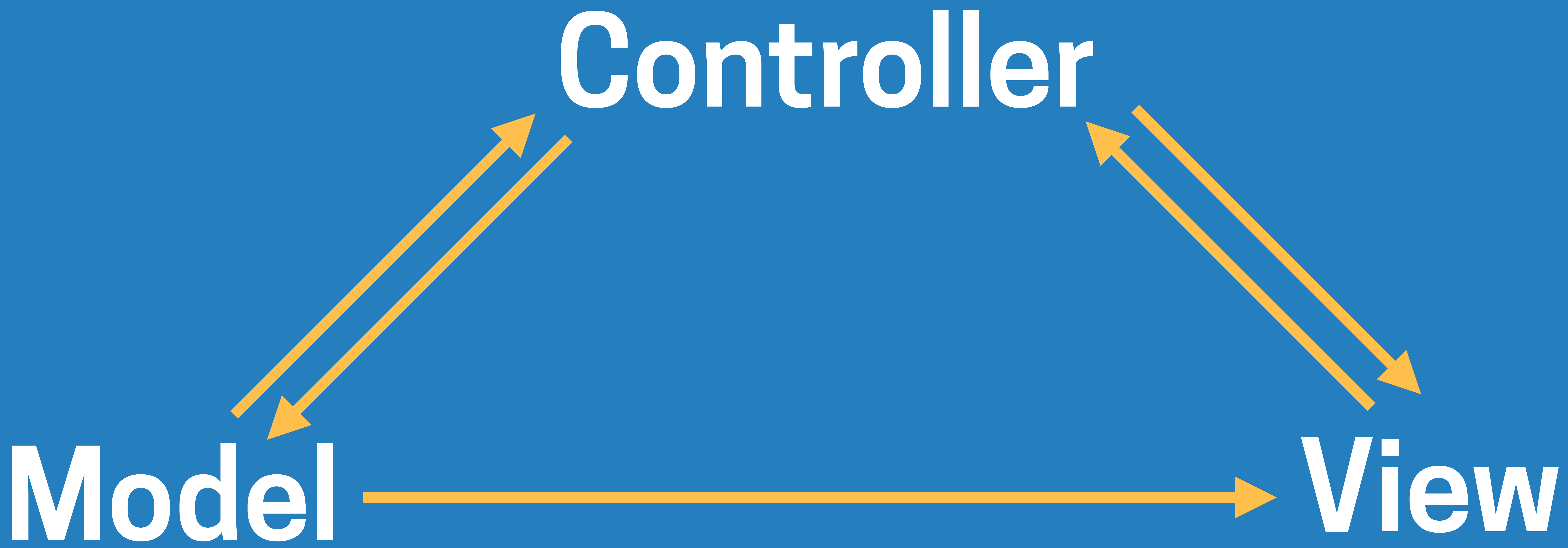
## While in reality…

## User interaction and network requests drive state changes

## User Input changes UI State

## Network request changes model state and UI state





State changes propagate in many directions

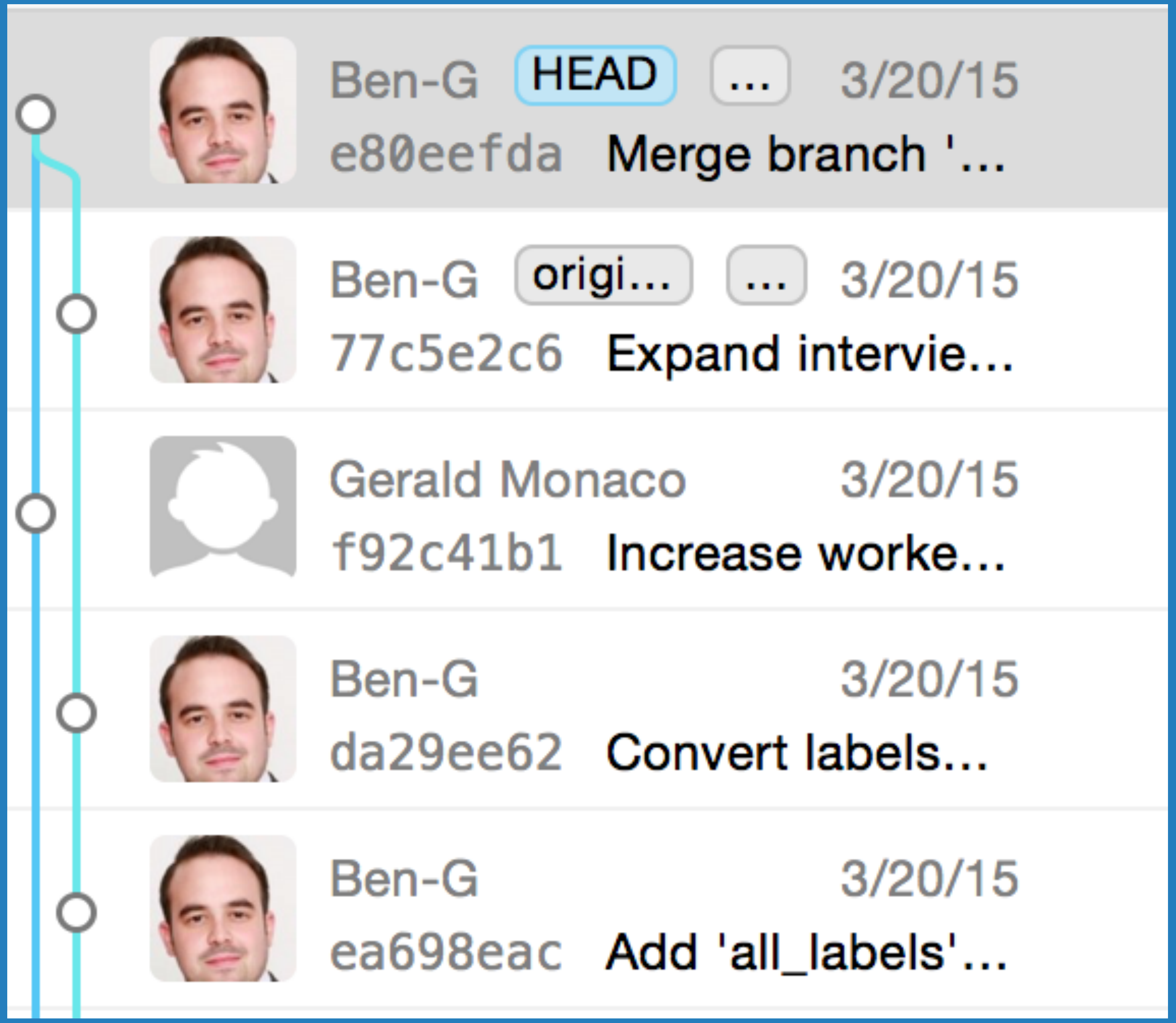
We don’t have the **tools** to **declare** the relationship between model and view

## State propagation is handled manually

## State propagation tools

- Callbacks
- Delegate methods
- KVO / Property overriding

# Manual state management is error prone



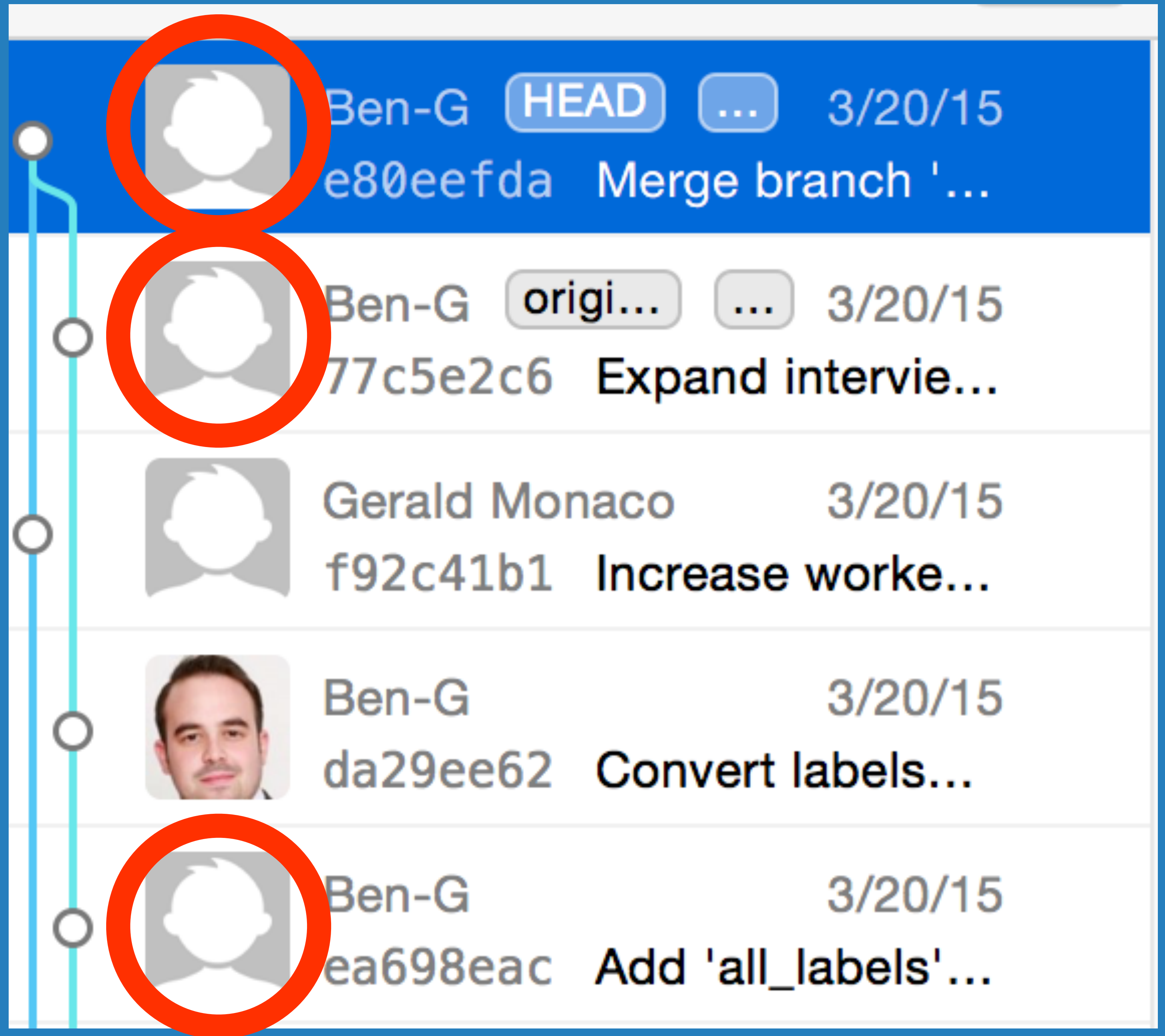
Ben-G HEAD … 3/20/15
e80eefda Merge branch '…

Ben-G origi… … 3/20/15
77c5e2c6 Expand intervie…

Gerald Monaco 3/20/15
f92c41b1 Increase worke…

Ben-G 3/20/15
da29ee62 Convert labels…

Ben-G 3/20/15
ea698eac Add 'all_labels'…



Ben-G HEAD … 3/20/15
e80eefda Merge branch '…

Ben-G origi… … 3/20/15
77c5e2c6 Expand intervie…

Gerald Monaco 3/20/15
f92c41b1 Increase worke…

Ben-G 3/20/15
da29ee62 Convert labels…

Ben-G 3/20/15
ea698eac Add 'all_labels'…

## FRP allows us to declare relationships instead of implementing them manually

## What is functional reactive programming?

## Imperative vs. Declarative

# Imperative

| A | B | C |
|---|---|---|
| 20 | 10 | ? |

0. Perform the following steps whenever A or B changes
1. Add 50 to value of A
2. Subtract 10 from value of B
3. Add the results from 1.) and 2.)
4. Write result from previous step into C

## Declarative

| A | B | C |
|---|---|---|
| 20 | 10 | ? |

$$C = (A+50) + (B-10)$$

## How can we use FRP to propagate state changes **declaratively**?

- Callbacks
- Delegate methods
- KVO / Property overriding

$\Big\}$ Signals

Reactive

$\swarrow$

## Signals model values over time

[1]

## Signals can be transformed using higher order functions

↖

Functional

## Example?
## Printing mouse position in Elm

A functional reactive language
for interactive applications

[2]

Immutable variable

Transform

Signal

```
main = map asText Mouse.position
```

Higher-Order Function

**FRP in a nutshell!**

[3]

We can assign the **current value** and all **future values** to a variable

➔ **Binding**

## Bindings are one way of subscribing to a Signal

## Intro to Reactive Cocoa 2.x

```
- (void)awakeFromNib {
  RAC(self, avatarImageView.image) =
    RACObserve(self, model.avatar);

  RAC(self, nameLabel.text) =
    RACObserve(self, model.name);

  // more binding code
}
```

**View updates whenever model or model properties change**

**NSMeetup**

A Monthly iOS/OSX Developer Meetup led by @stevederico

NSMeetup

```
RAC(target, keypath) = …
```

## Assigns a signal to an object property

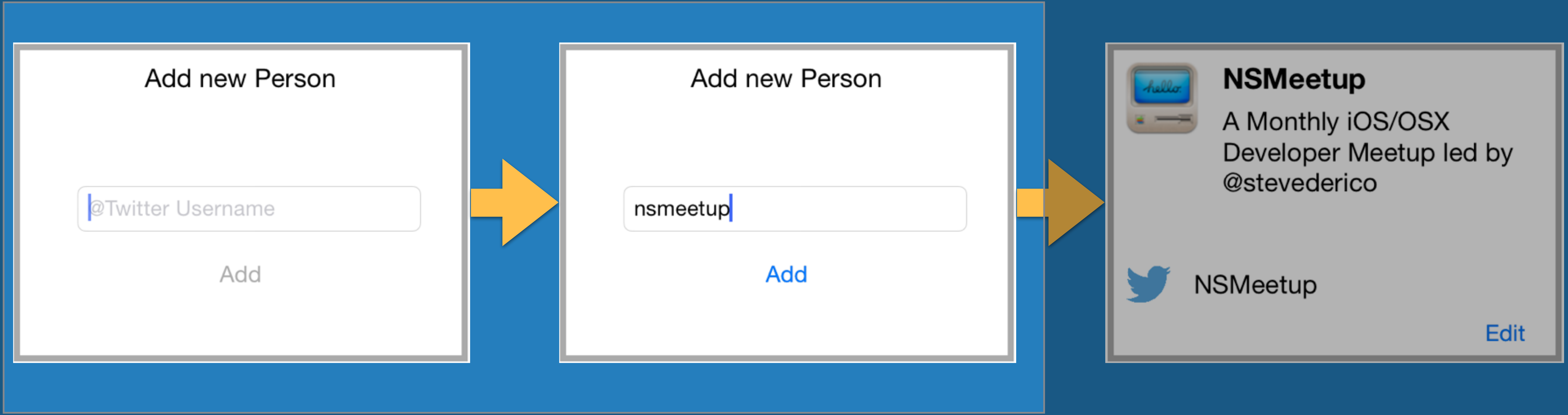
```
RACObserve(self, model.name);
```

**Creates a signal from KVO changes**

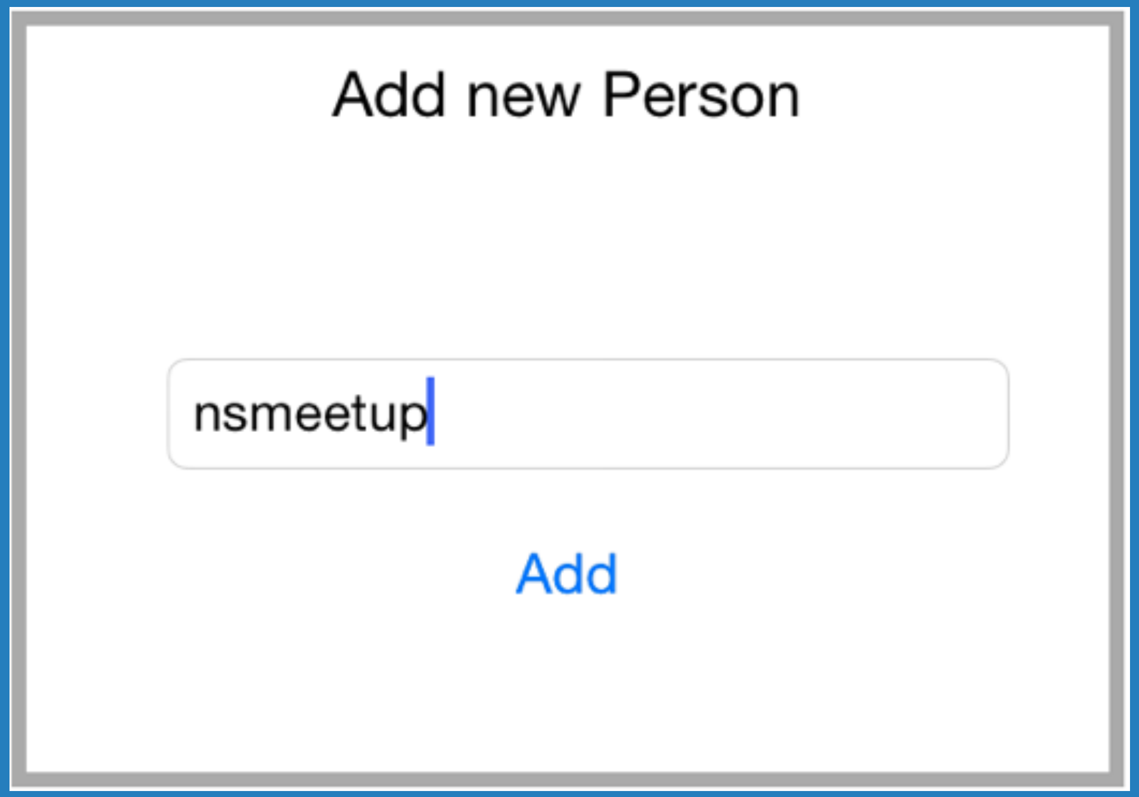
$$\text{Model} \longrightarrow \text{View} \quad ✓$$

$$\text{Model} \rightleftarrows \text{View} \quad ?$$

| Model | ↔ | ViewModel | ↔ | View |
|---|---|---|---|---|
| Stores model state, provides business logic | | Stores View state, communicates with model | | Bindings |


Add new Person
@Twitter Username
Add
Add new Person
nsmeetup
Add
NSMeetup
A Monthly iOS/OSX Developer Meetup led by @stevederico
NSMeetup
Edit

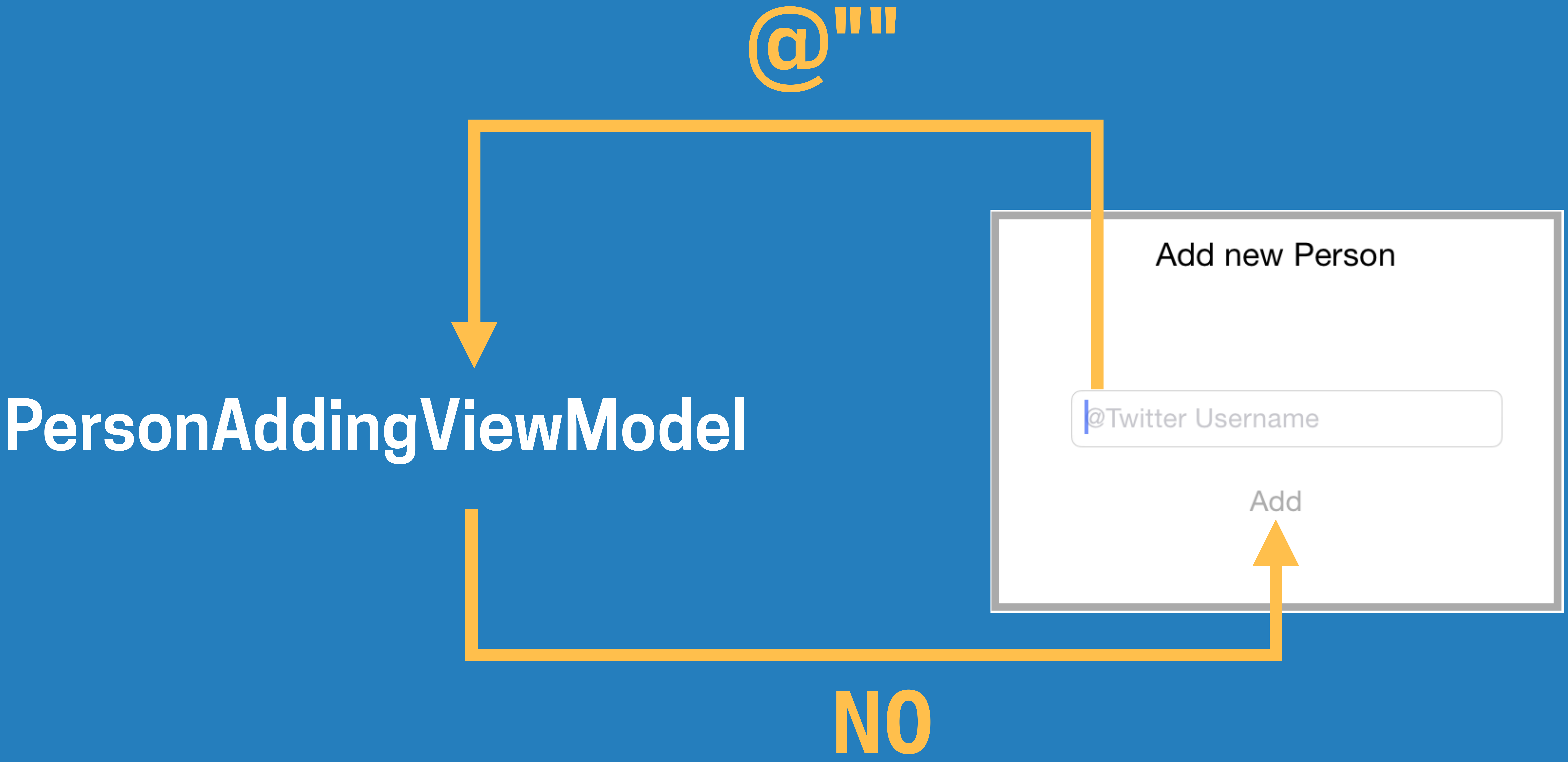
| PersonAddingViewModel | | PersonAddingView* |
|---|---|---|
| usernameSearchText | ↔ | usernameTextfield.text |
| addButtonEnabledSignal | → | addButton.enabled |
| addButtonCommand | → | addButton.rac_command |
| (performs network request) | | |

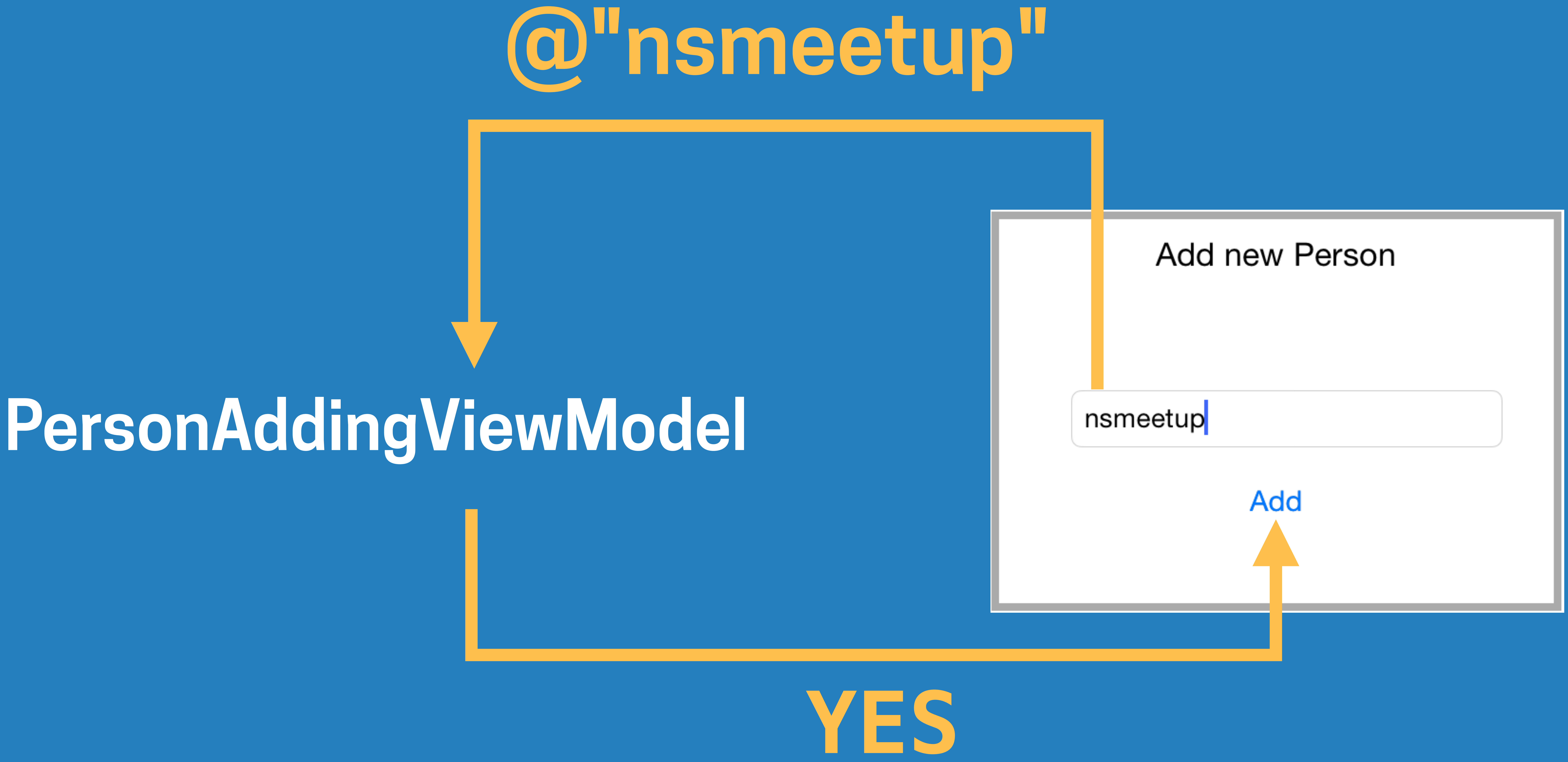


*some variables have been renamed for brevity

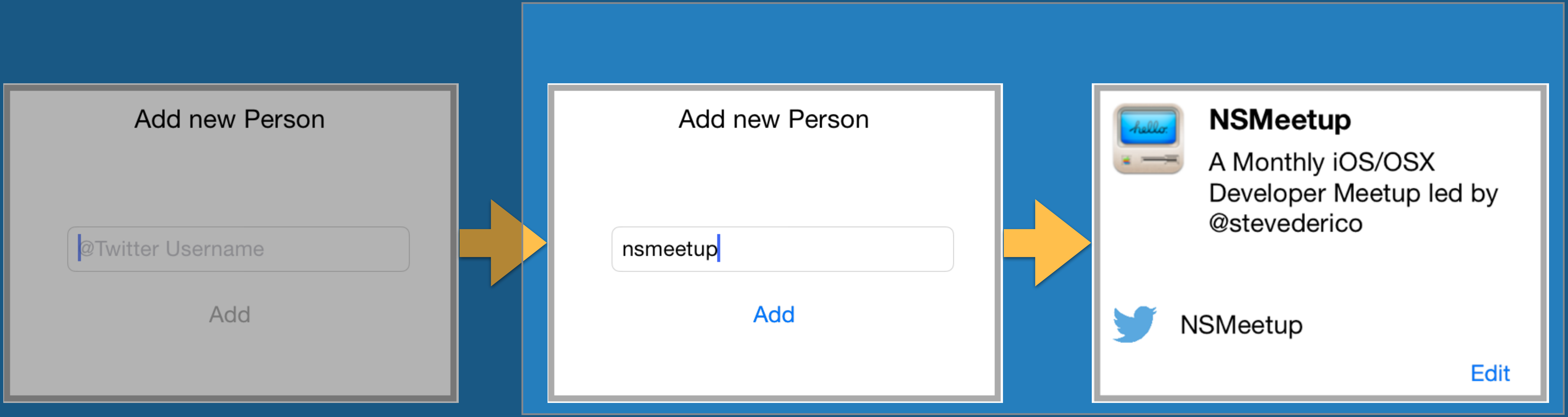
```
@interface PersonAddingViewModel : NSObject

// generates a signal when the ‘add’ button is pressed
@property (strong) RACCommand *addTwitterButtonCommand;
// is bound to the text field in the UI
@property (strong) NSString *usernameSearchText;
// a signal that determines wether ‘add’ button is enabled
@property (strong) RACSignal *addButtonEnabledSignal;

// less relevant interface declarations…

@end
```

```
self.addButtonEnabledSignal = [RACObserve(self, usernameSearchText)
                               map:^id(NSString *searchText) {
  if (!searchText || [searchText  isEqualToString:@""]) {
    return @(NO);
  } else {
    return @(YES);
  }
}];
```


@""
PersonAddingViewModel
Add new Person
@Twitter Username
Add
NO


@"nsmeetup"
PersonAddingViewModel
YES
Add new Person
nsmeetup
Add

## Networking with Reactive Cocoa 2.x

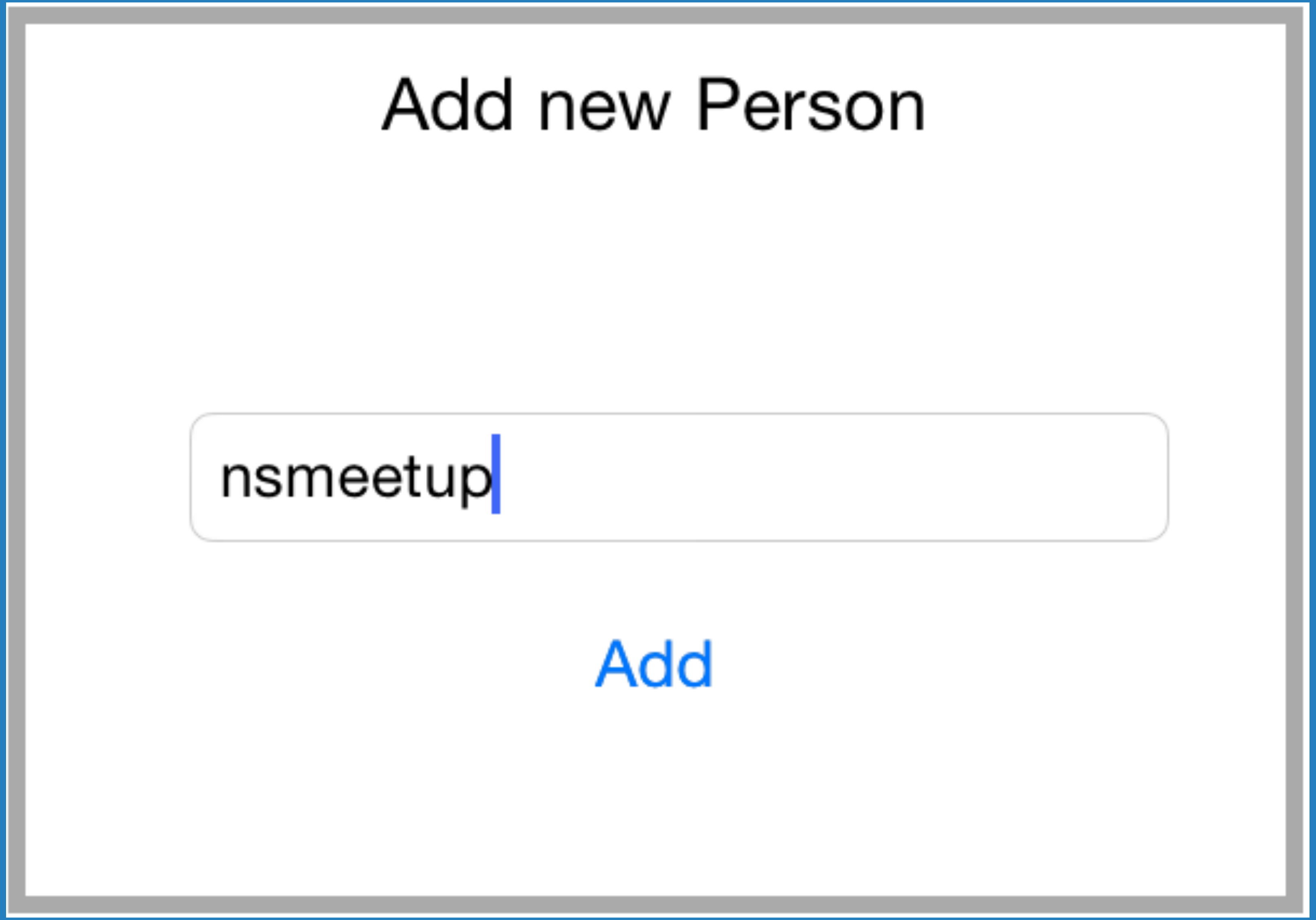


A Monthly iOS/OSX Developer Meetup led by @stevederico

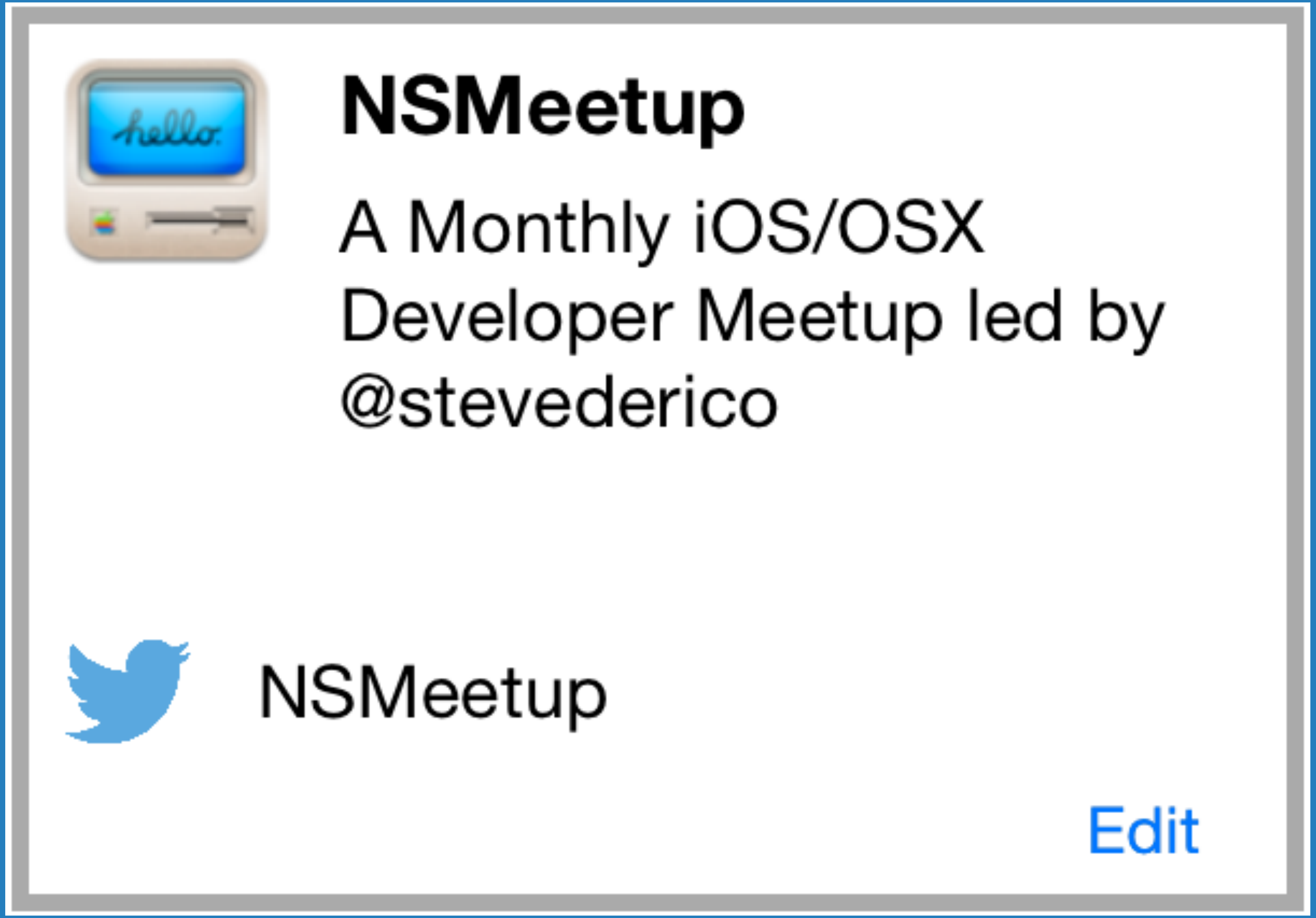
## PersonContainerView

PersonAddingView

PersonDetailView


Add new Person

nsmeetup


**NSMeetup**

A Monthly iOS/OSX Developer Meetup led by @stevederico

NSMeetup

## PersonAddingViewModel

### Kicking off the network request

```
self.addTwitterButtonCommand = [[RACCommand alloc]
  initWithEnabled:self.addButtonEnabledSignal
    signalBlock:^RACSignal *(id input) {
      RACSignal *signal = [self.twitterClient
        infoForUsername:self.usernameSearchText];

      return signal;
    }
];
```

## PersonContainerViewModel

### Changing the UIState upon completed request

```
// subscribe to twitter network request
RACSignal *twitterFetchSignal = [RACObserve(self, personAddingViewModel)
   flattenMap:^RACStream *(id value) {
      return [self.personAddingViewModel.
        addTwitterButtonCommand.executionSignals concat];
}];

RACSignal *UIStateSignal = [[twitterFetchSignal map:^id(id value) {
  return @(PersonCollectionReusableViewStateDetails);
}] startWith:@(PersonCollectionReusableViewStateAddingTwitter)];

RAC(self, UIState) = UIStateSignal;
```

## PersonContainerViewModel

### Updating the model upon completed request

```
RAC(self.person, avatar) = [twitterFetchSignal reduceEach:
  ^id(UIImage *avatar, NSDictionary *userInfo){
    return avatar;
}];

RAC(self.person, twitterUsername) = [twitterFetchSignal reduceEach:
  ^id(UIImage *avatar, NSDictionary *userInfo) {
    return userInfo[@"twitterHandle"];
}];

RAC(self.person, name) = [twitterFetchSignal reduceEach:
  ^id(UIImage *avatar, NSDictionary *userInfo){
    return userInfo[@"name"];
}];
```

## Twitter network request

```
- (RACSignal *)infoForUsername:(NSString *)username {
    RACScheduler *bgScheduler = [RACScheduler
      schedulerWithPriority:RACSchedulerPriorityBackground];

    return [[[[self _login]
      deliverOn:bgScheduler]
      flattenMap:^RACStream *(STTwitterAPI *client) {
        return [self client:client fetchUserInfo:username];
    }] flattenMap:^RACStream *(NSDictionary *userInfo) {
    …
```

## Twitter network request

```
    flattenMap:^RACStream *(STTwitterAPI *client) {
      return [self client:client fetchUserInfo:username];
    }] flattenMap:^RACStream *(NSDictionary *userInfo) {
      NSDictionary *userDetails =
         @{@"name": userInfo[@"name"],
         @"description": userInfo[@"description"],
         @"twitterHandle": userInfo[@"screen_name"]};

      NSString *downloadURL = [userInfo[@"profile_image_url_https"];

      return [[self imageFromURLString:downloadURL]
              combineLatestWith:[RACSignal return:userDetails]];
    }];
  }
```

## RACSignal can be used like a promise

## Wrapping network requests into a RACSignal

```
- (RACSignal *)client:(STTwitterAPI *)client fetchUserInfo:(NSString
*)username {
  return [RACSignal createSignal:
    ^RACDisposable *(id<RACSubscriber> subscriber) {

     [client getUserInformationFor:username successBlock:^(NSDictionary
 *user) {
      [subscriber sendNext:user];
      [subscriber sendCompleted];
    } errorBlock:^(NSError *error) {
      [subscriber sendError:error];
    }];

    return nil;
  }];
};
```

$$\begin{tikzcd}\text{Model} \arrow[r, leftrightarrow] & \text{ViewModel} \arrow[r, leftrightarrow] & \text{View}\end{tikzcd}$$ ✓

## Testing with Reactive Cocoa 2.x

## Testing UI without UIKit

```
it(@"calls the Twitter API when add button is tapped", ^{
  id twitterClient = [TwitterClient new];
  id twitterMock = OCMPartialMock(twitterClient);
  OCMStub([twitterMock infoForUsername:@“username"])
     .andReturn([RACSignal return:@(YES)]);

  viewModel = [[PersonAddingViewModel alloc]
    initWithTwitterClient:twitterMock];
  viewModel.usernameSearchText = @"username";
  [viewModel.addTwitterButtonCommand execute:nil];

  OCMVerify([twitterMock infoForUsername:@"username"]);
});
```

## Drawbacks

- New Programming Model
- Debugging
- (Performance)

## Summary

- RAC provides tools for writing simpler **declarative code**
- **Signals** are unified way of handling different types of future values
- State propagation can be handled through **bindings**

- **MVVM** plays nicely with bindings, reduces controller complexity
- **MVVM** make it easier to write tests
- RAC introduces vastly different programming model that can be harder to debug

# Resources

- [1] http://stackoverflow.com/questions/1028250/what-is-functional-reactive-programming
- [2] http://elm-lang.org/
- [3] http://elm-lang.org/edit/examples/Reactive/Position.elm