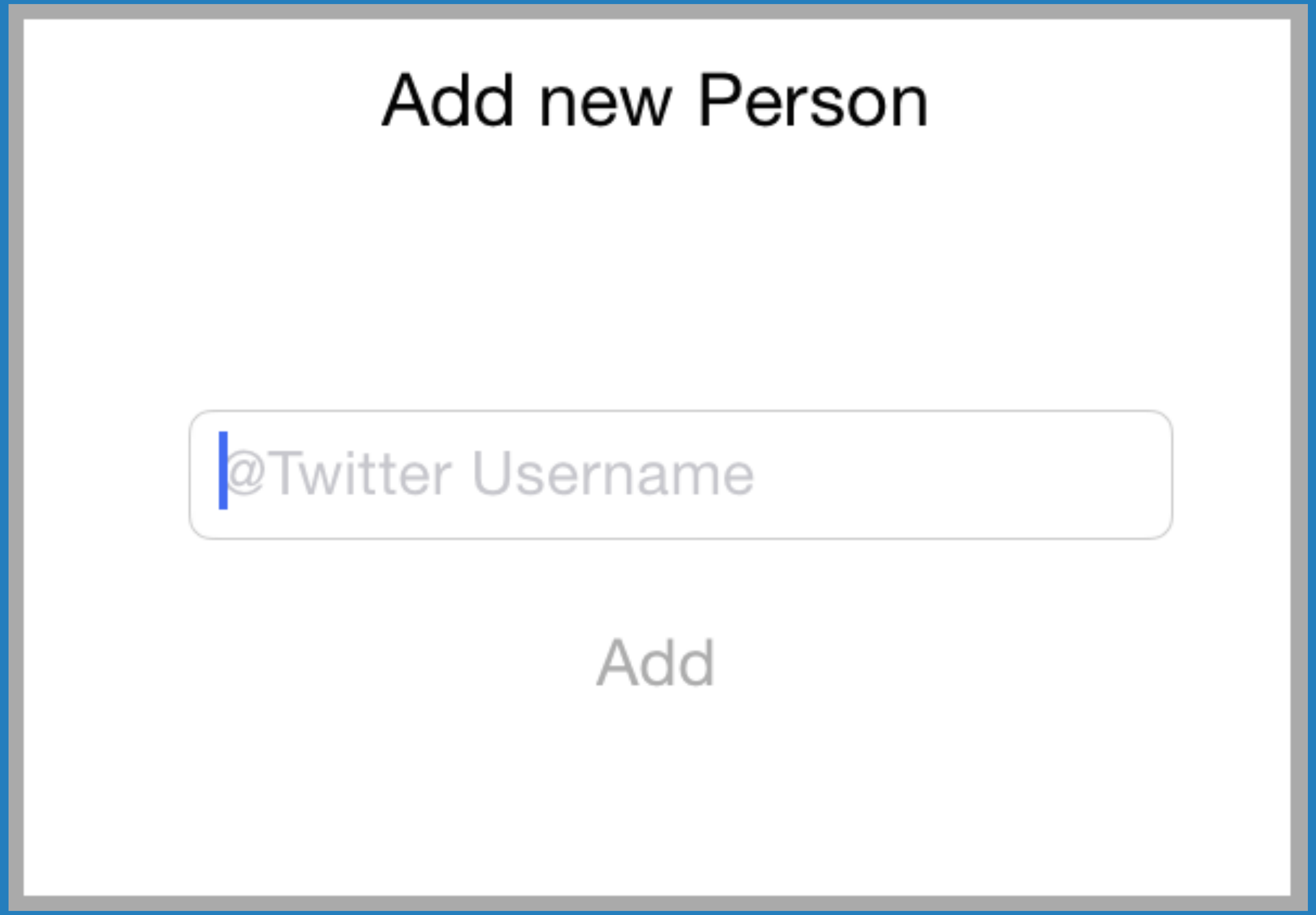
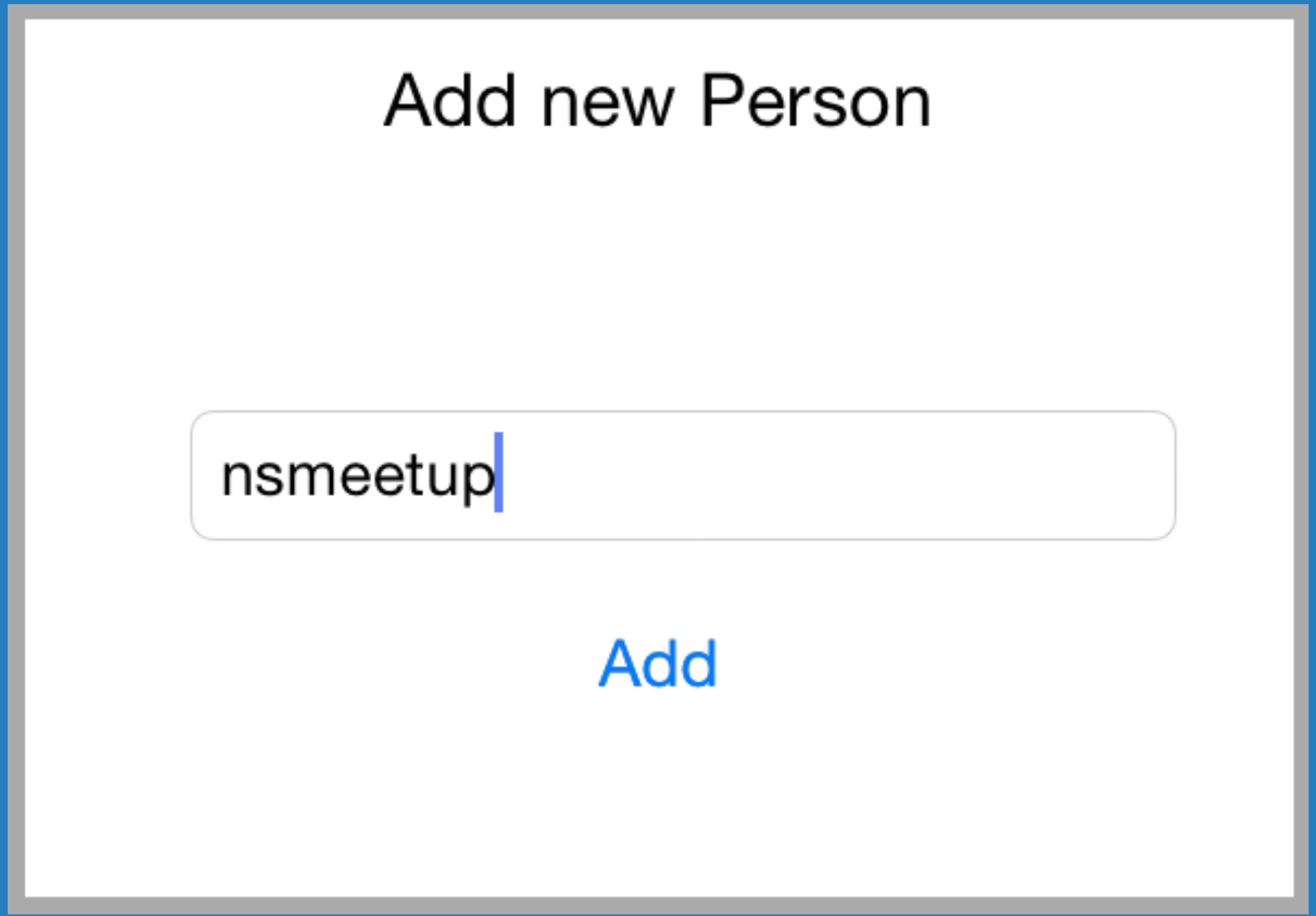
Functional Reactive Programming on iOS

3 High level states

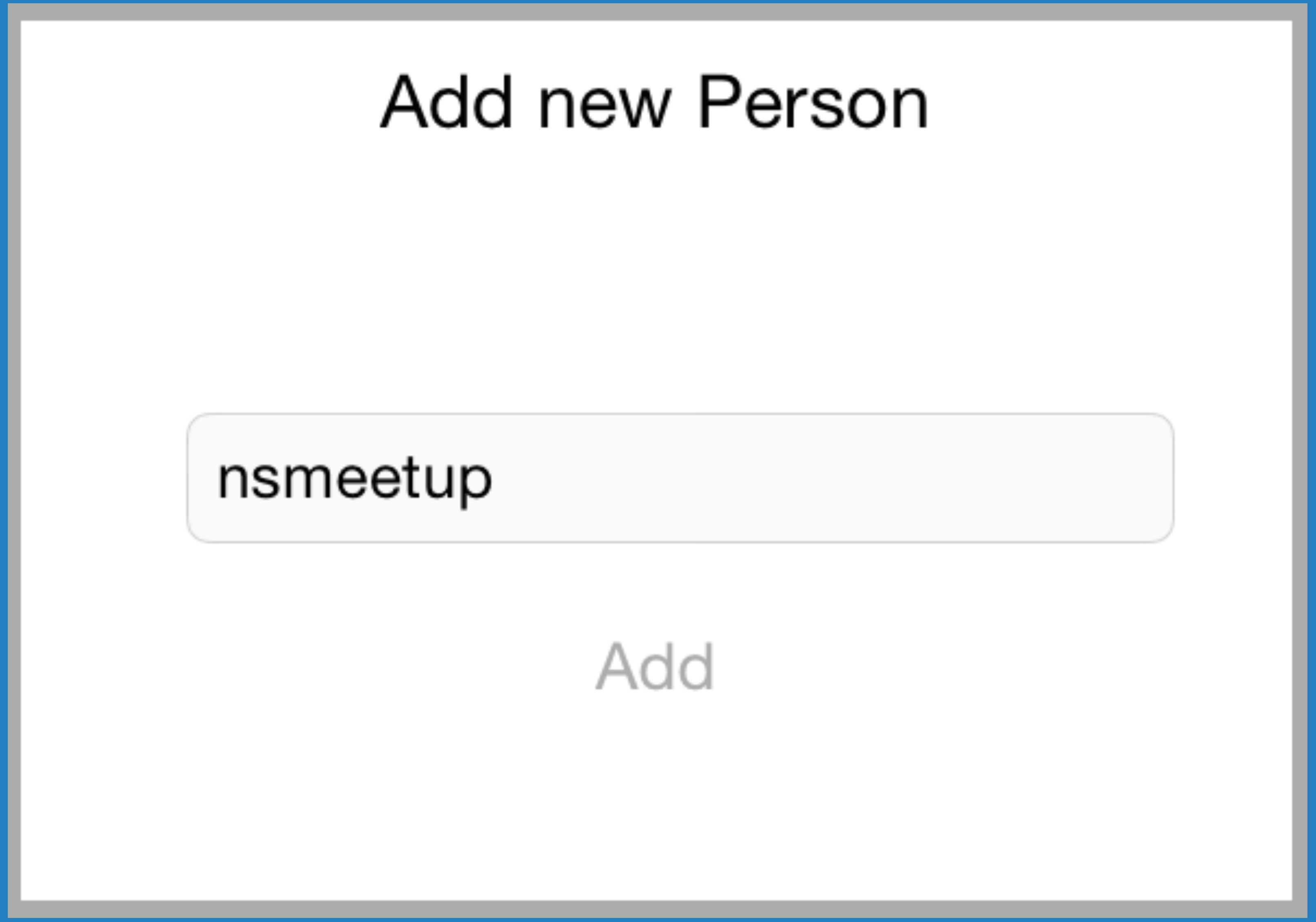
No entry

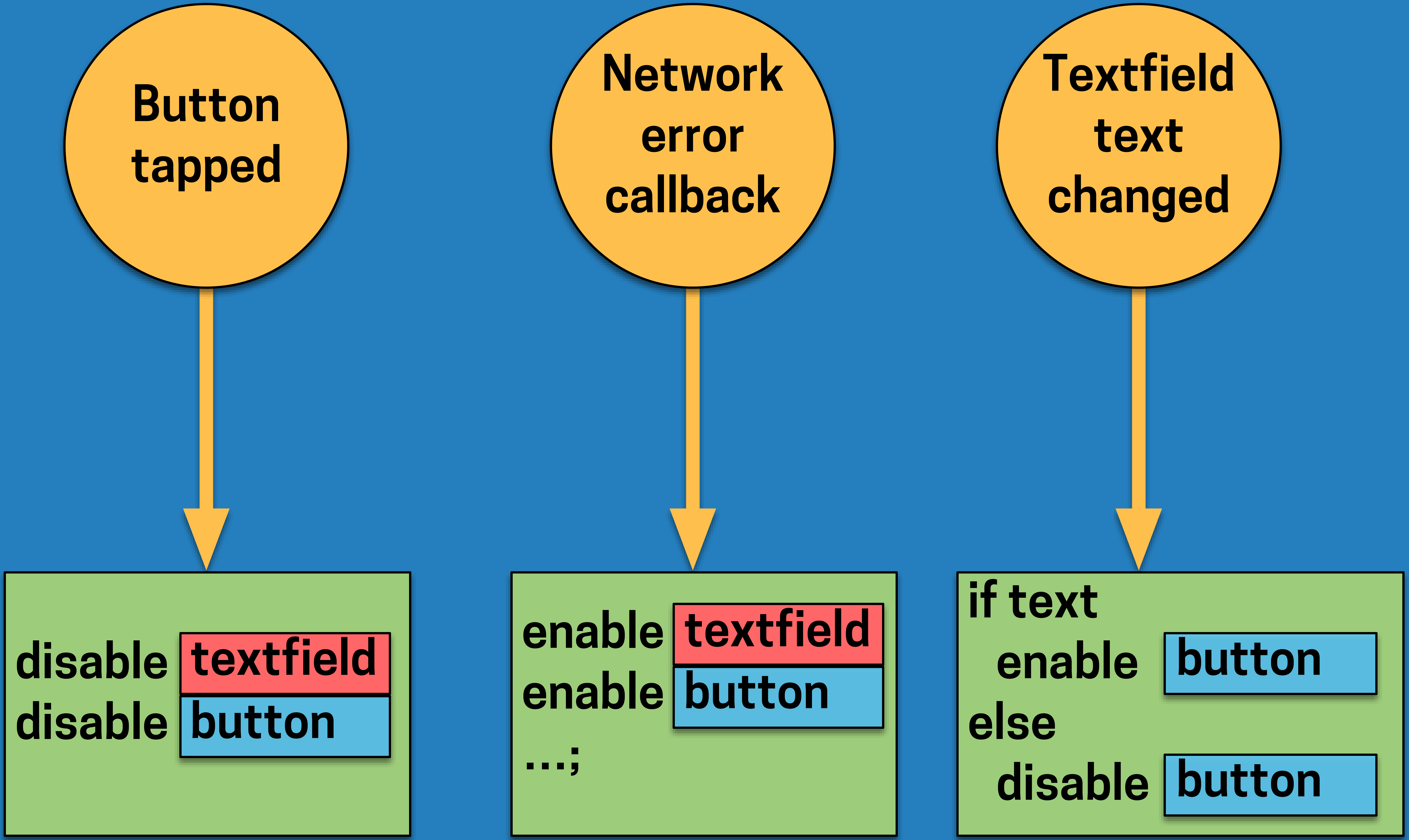


Default



Loading




Button tapped
Network error callback
Textfield text changed
disable textfield
disable button
enable textfield
enable button
...;
if text
enable button
else
disable button

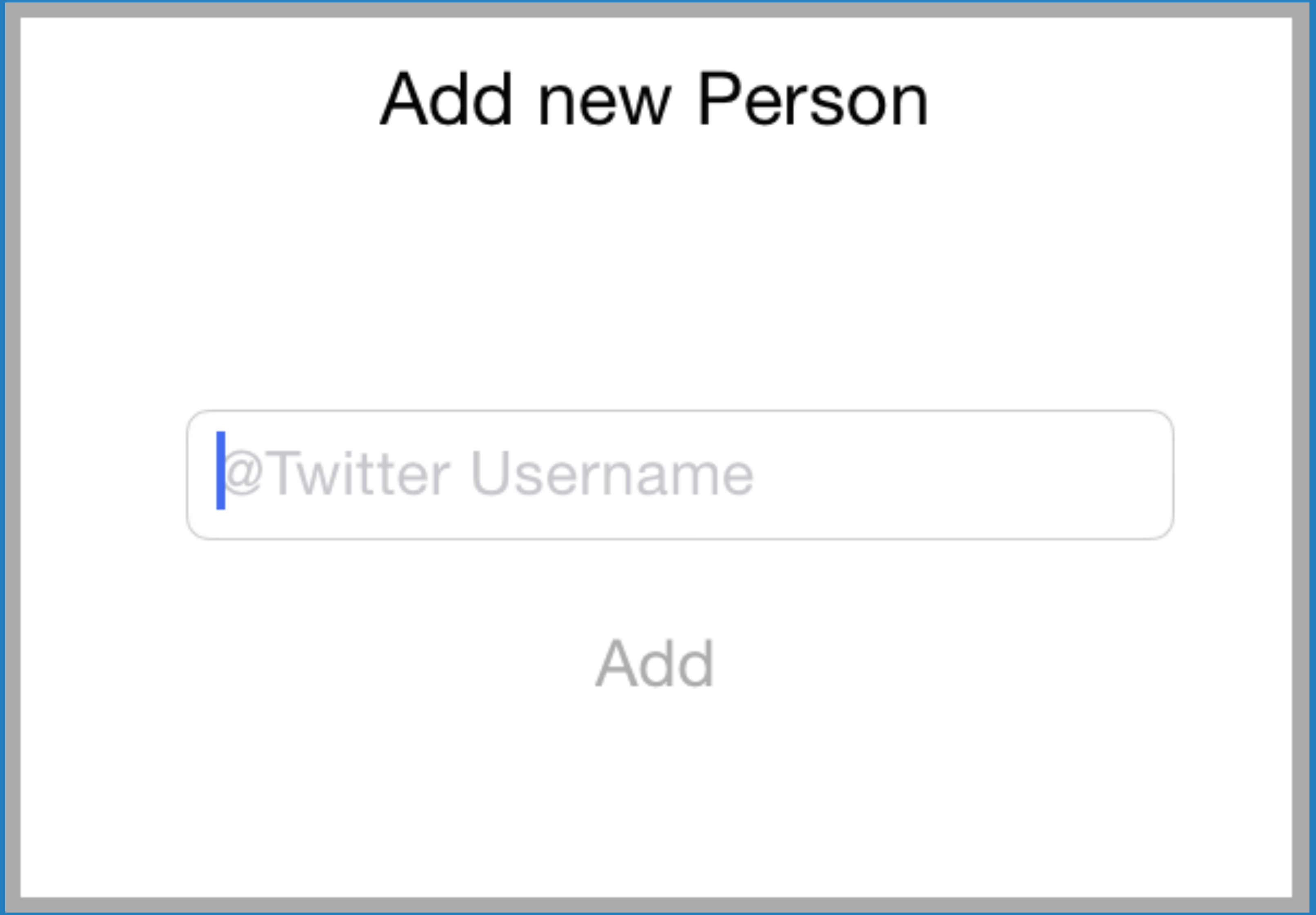
State propagation is handled **manually** by **mutating** variables

Problem

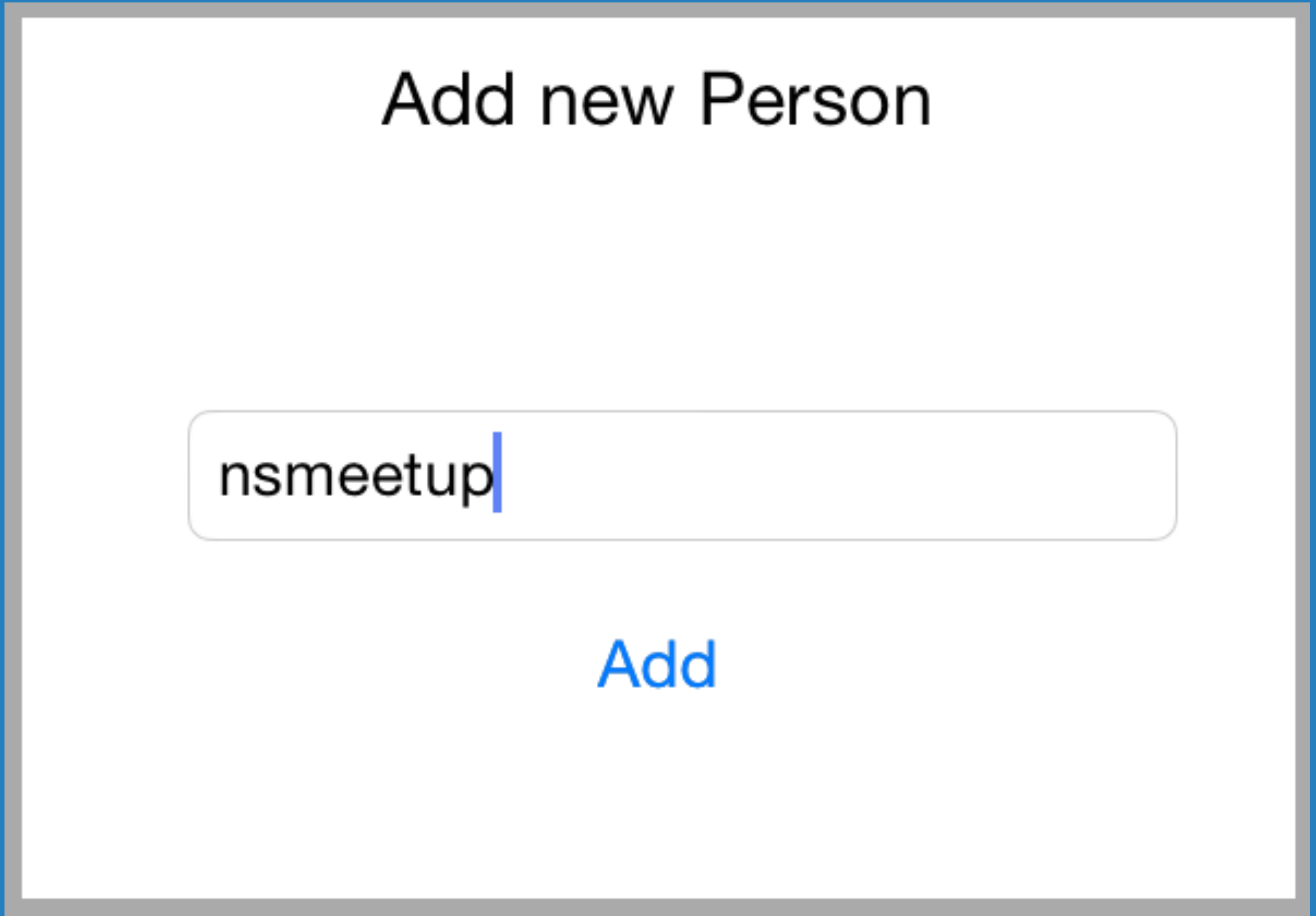
- State handling code is dispersed
  - Code is hard to read
  - Code is hard to maintain

3 High level states

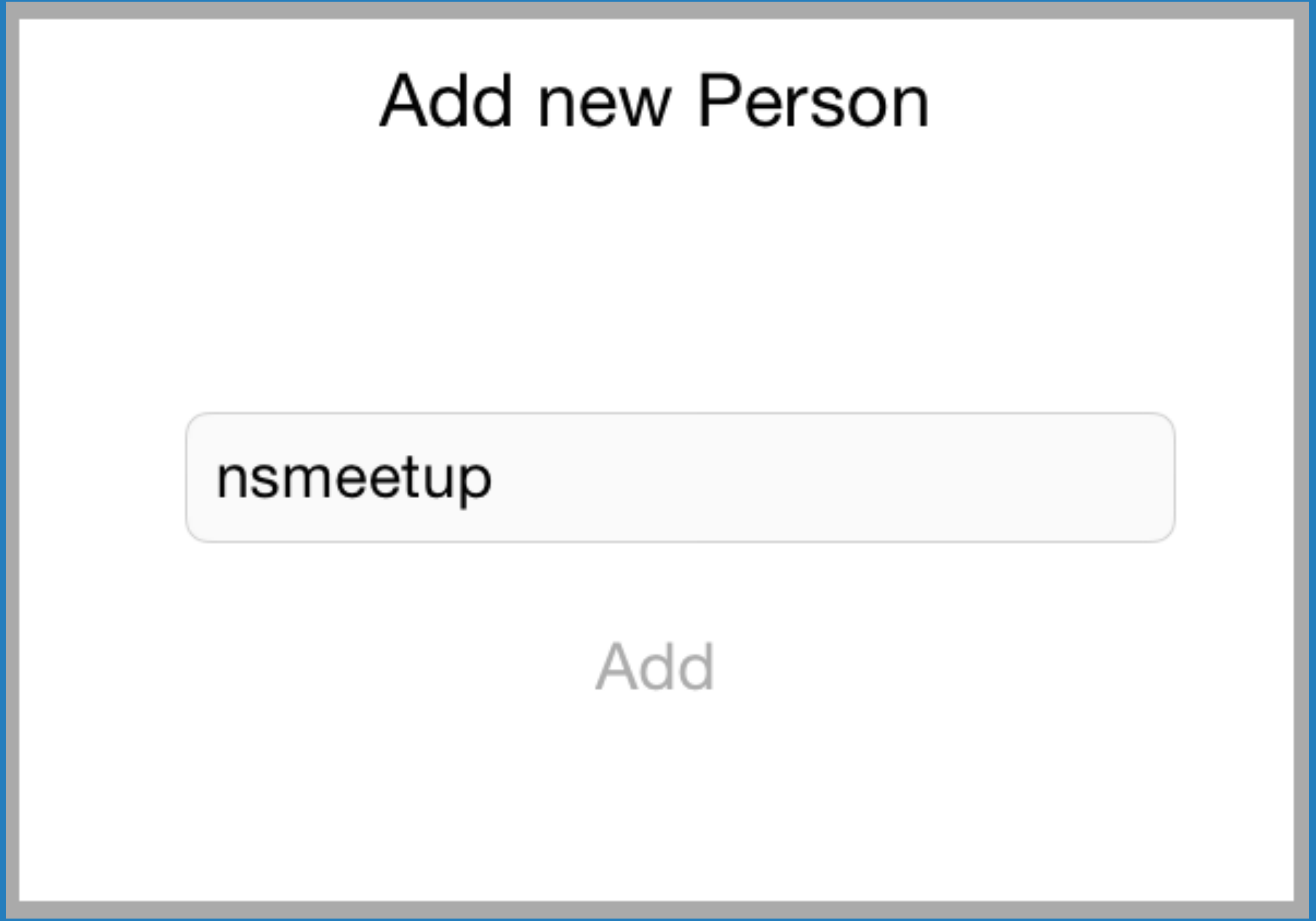
No entry



Default

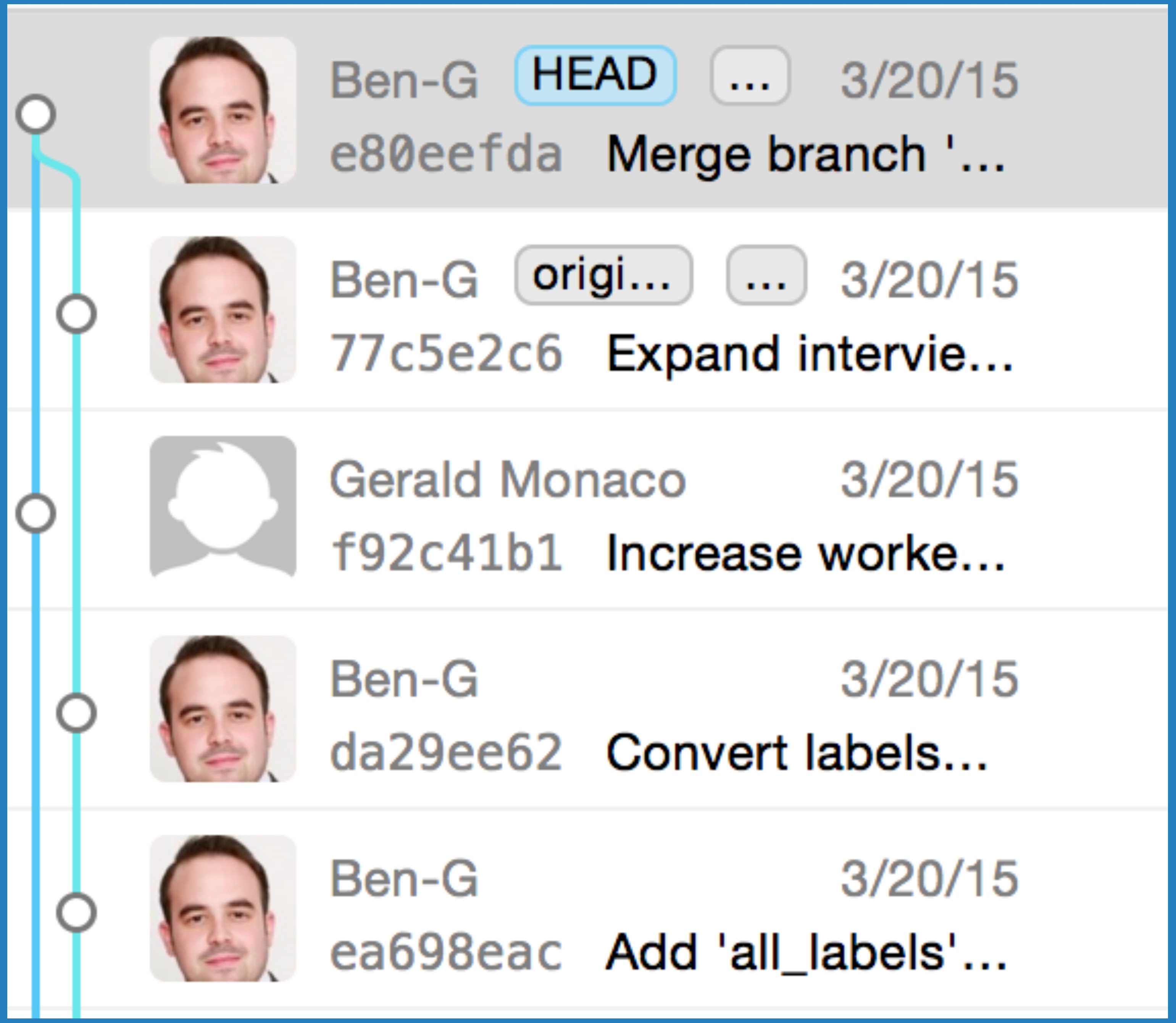


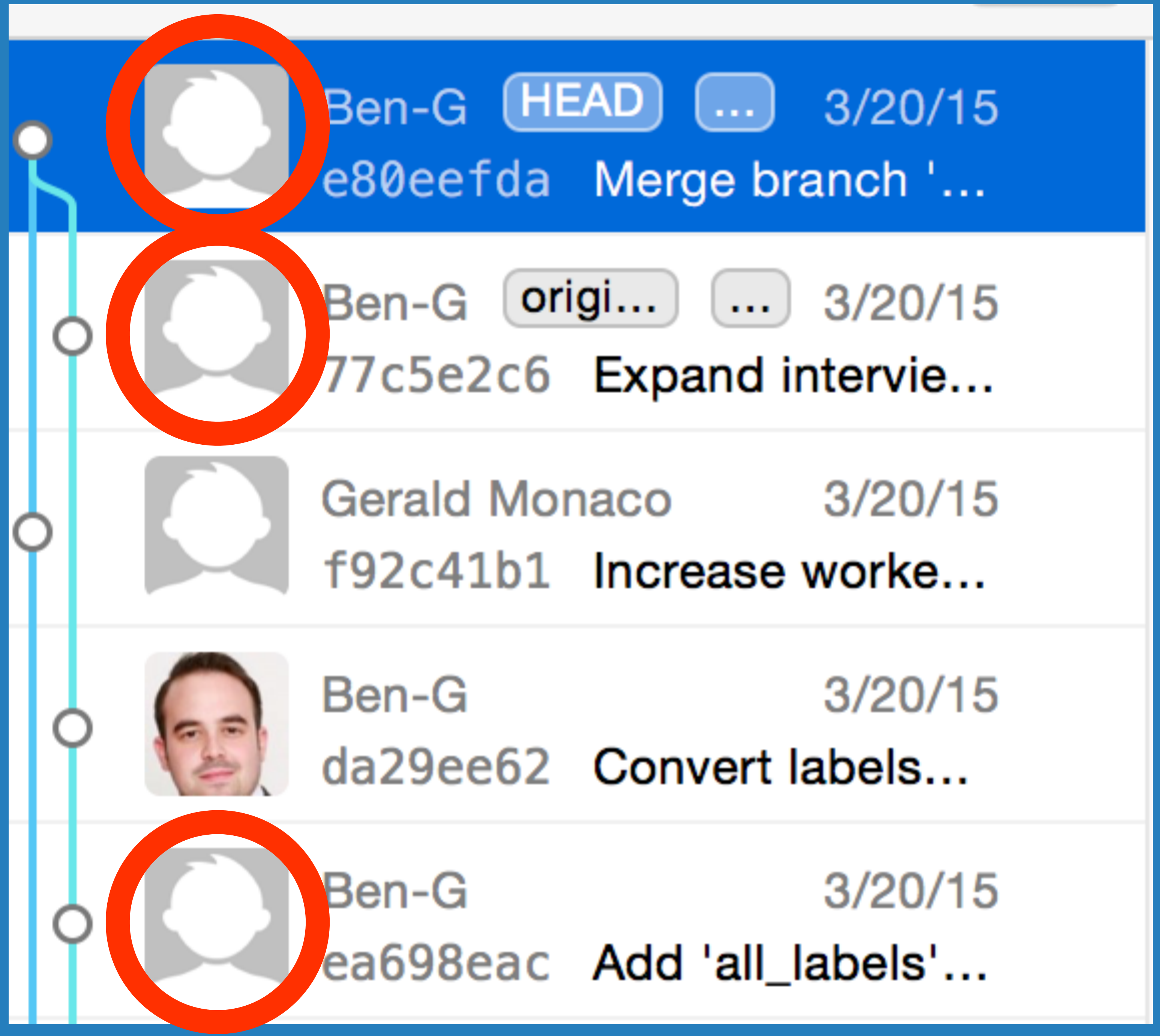
Loading



9 possible invalid states!

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'…

What is functional reactive programming?

Imperative vs. Declarative

Imperative

A	B	C
20	10	?

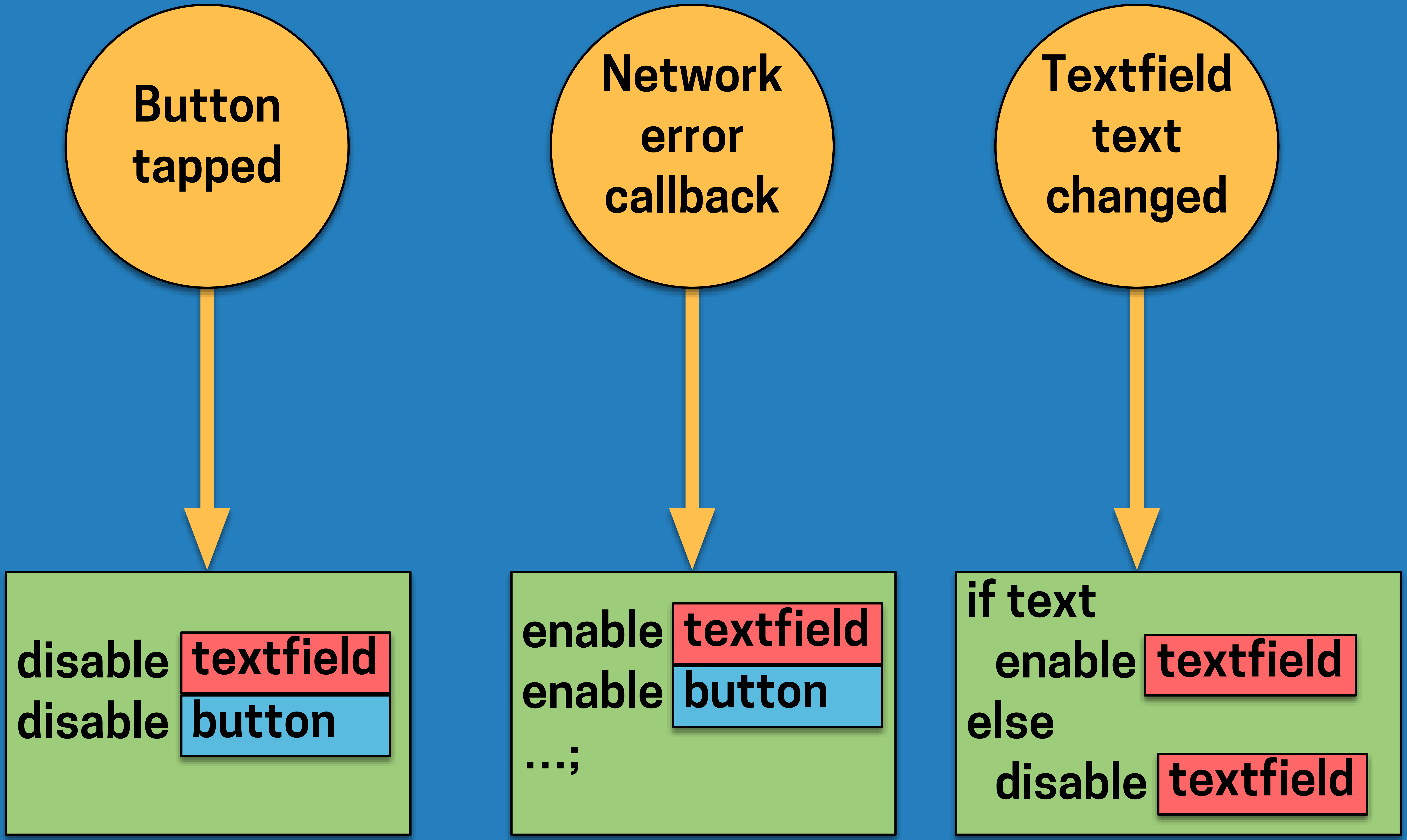
1. Perform the following steps whenever A or B changes
2. Add 50 to value of A
3. Subtract 10 from value of B
4. Add the results from 1.) and 2.)
5. Write result from 3.) into C

Declarative

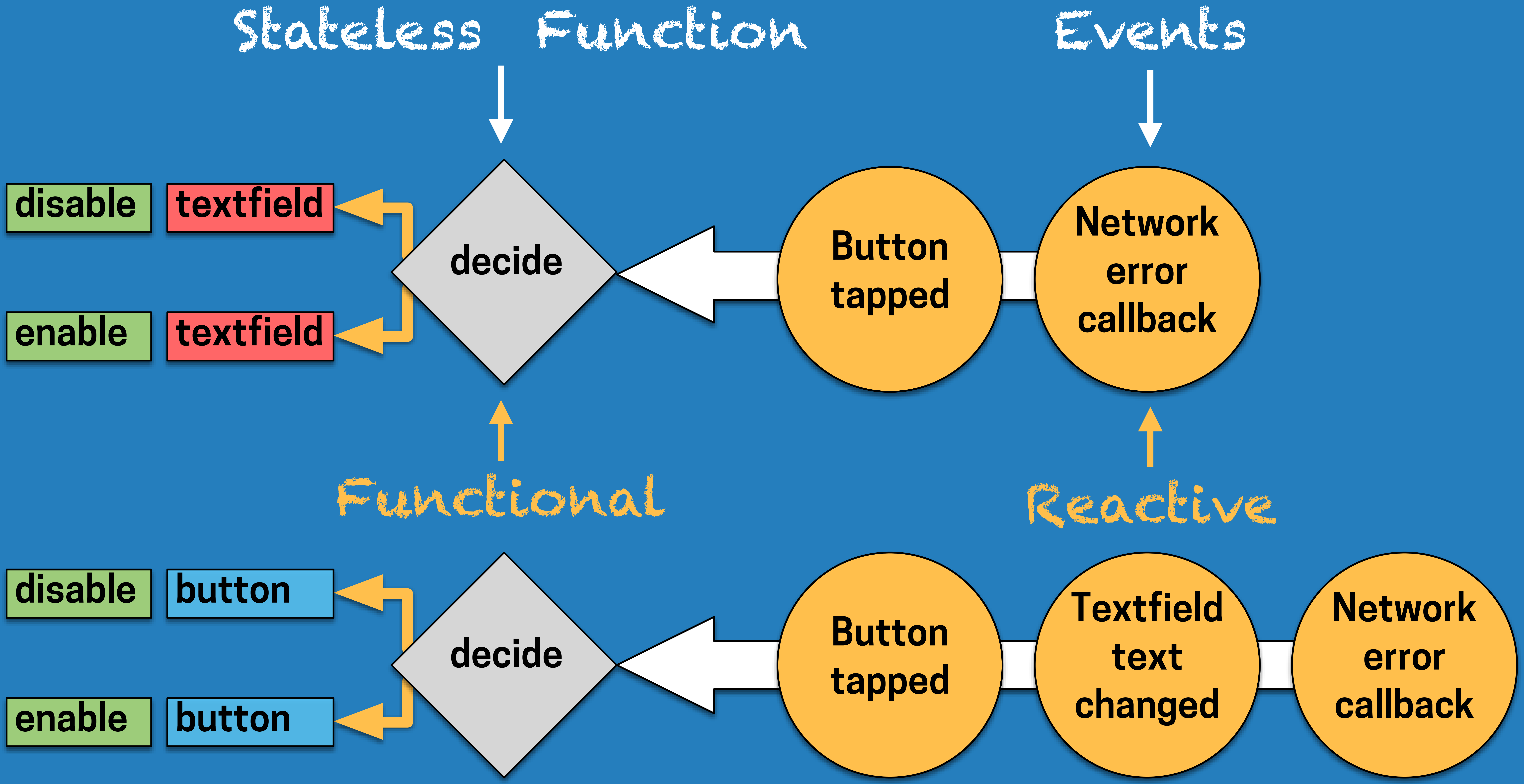
A	B	C
20	10	?

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

Imperative


Button tapped
Network error callback
Textfield text changed
disable textfield
disable button
enable textfield
enable button
...;
if text
enable textfield
else
disable textfield

Declarative


Stateless Function
Events
disable textfield
enable textfield
decide
Button tapped
Network error callback
Functional
Reactive
disable button
enable button
decide
Button tapped
Textfield text changed
Network error callback

State is derived from a defined set of inputs

Intro to Reactive Cocoa 2.x

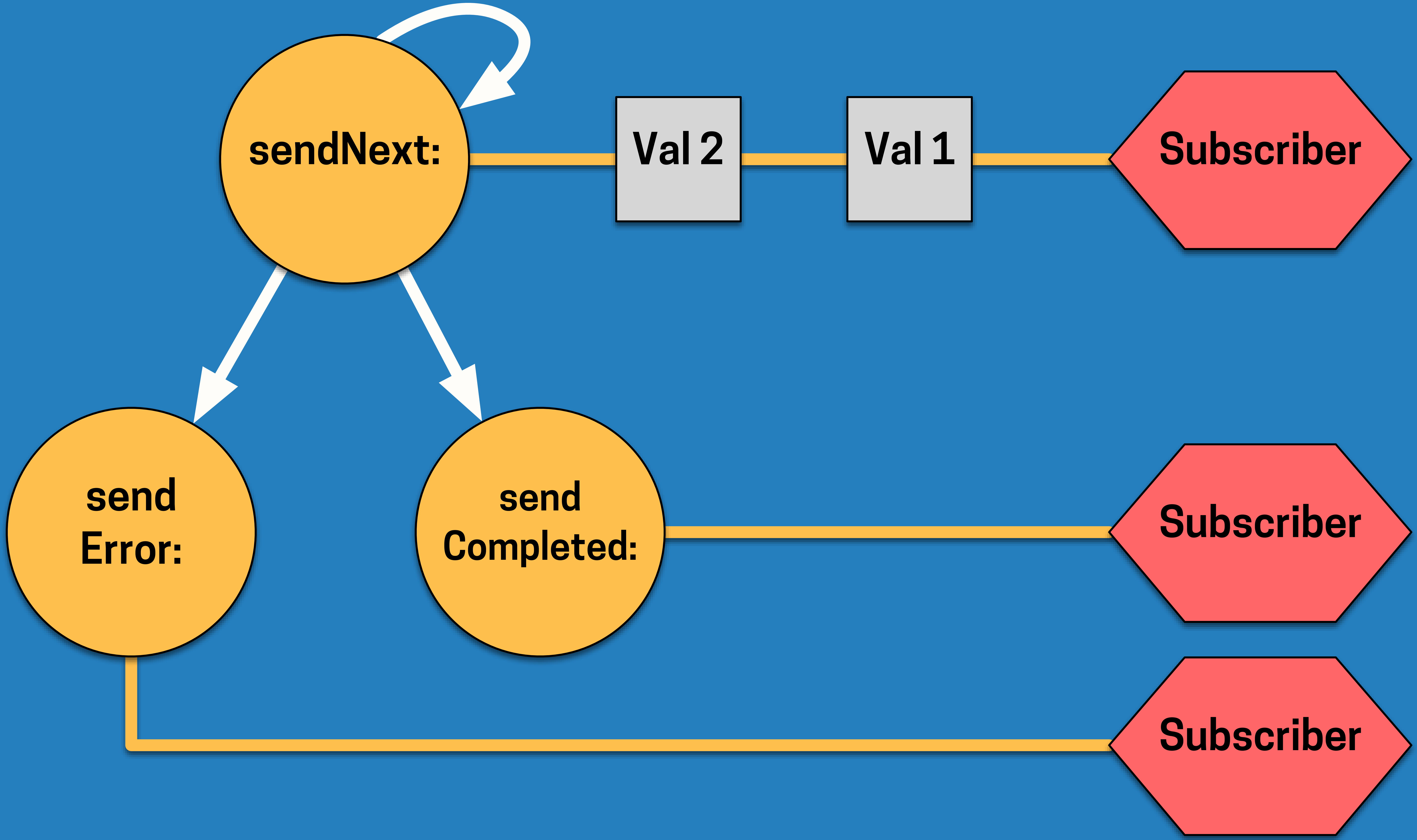
- Callbacks
- Delegate methods
- KVO / Property overriding

$\Big\}$ Signals

$\uparrow$

Values over time

RACSignal

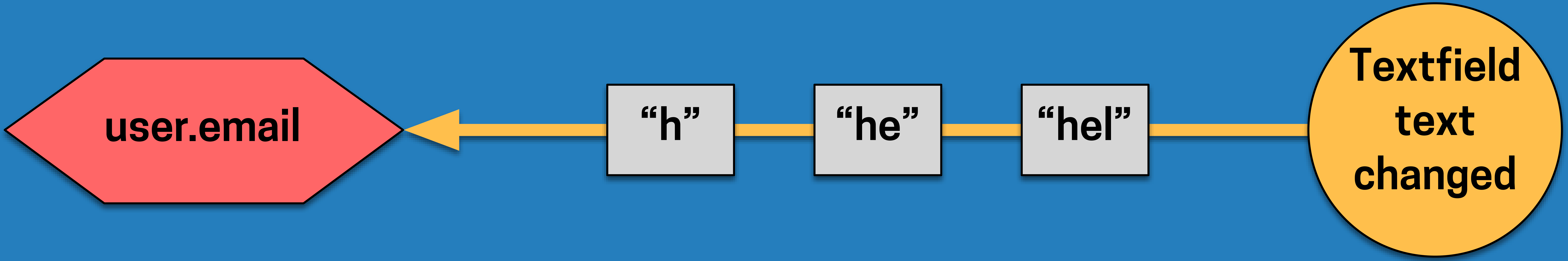

sendNext:
Val 2
Val 1
Subscriber
send Error:
send Completed:
Subscriber
Subscriber

We can **bind** Signals

OR

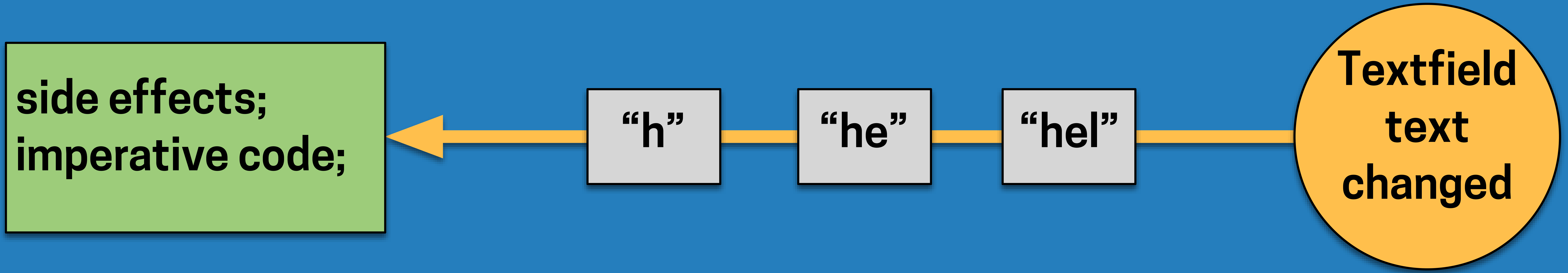
subscribe to Signals

Bind



```
RAC(self.user, username) =
    self.usernameTextfield.rac_textSignal
```

Subscribe



```
[self.usernameTextfield.rac_textSignal
  subscribeNext:^(NSString *t) {
    NSLog(@“New value: %@", x);
}];
```

Prefer binding over explicit subscription

Model -> View Binding with Reactive Cocoa

```
- (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

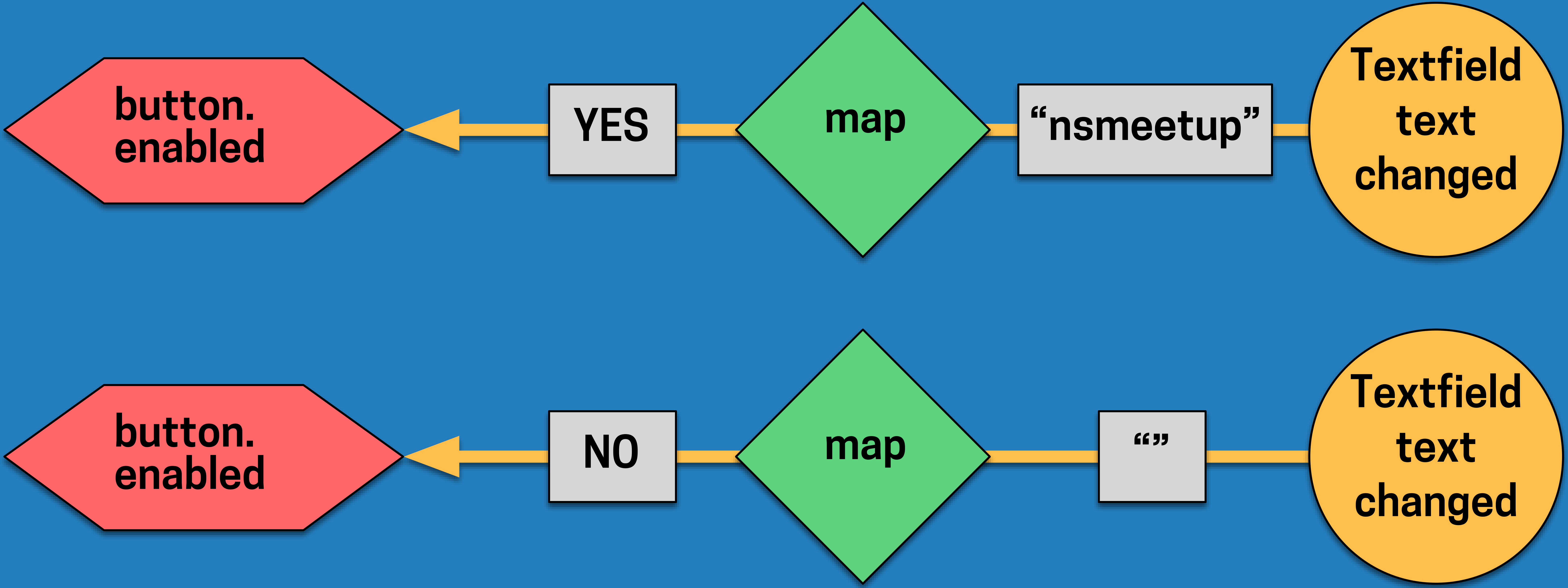
A Monthly iOS/OSX Developer Meetup led by @stevederico

NSMeetup

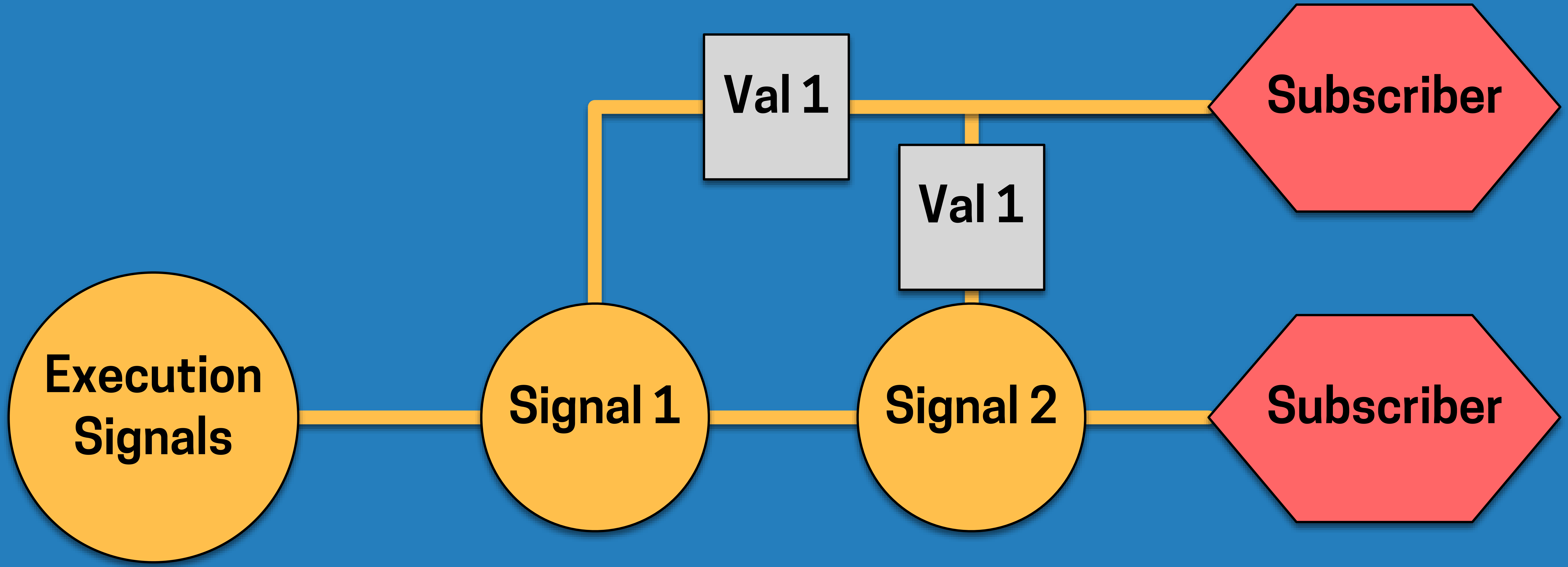
$$\text{Model} \longrightarrow \text{View} \ ✓$$

Model $\rightleftarrows$ View ?

Signal Operators



RACCommand


Val 1
Subscriber
Val 1
Execution Signals
Signal 1
Signal 2
Subscriber

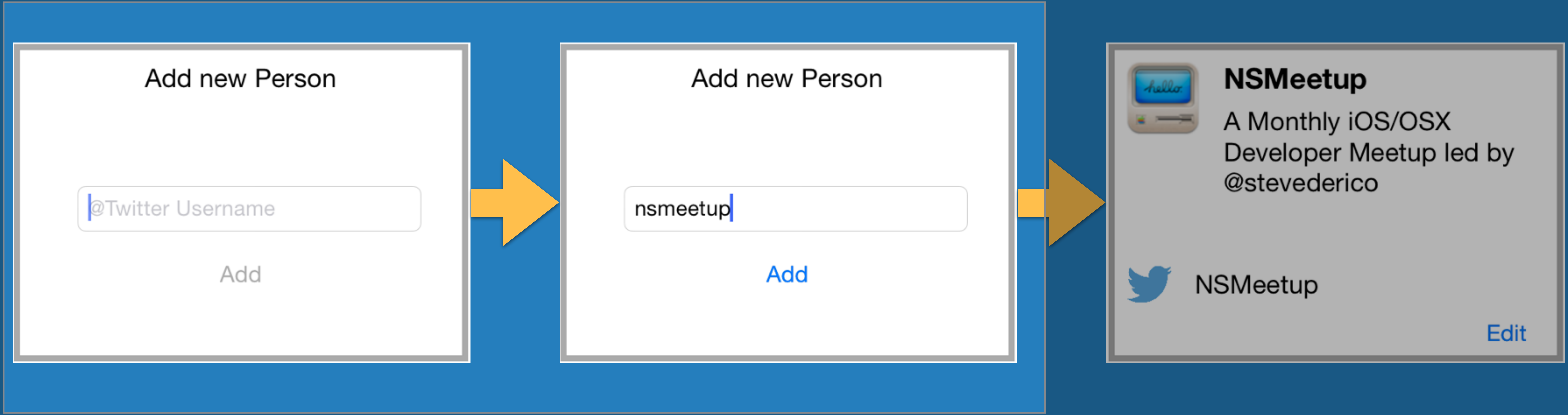
Model $\rightleftarrows$ View ?

Model ↔ ViewModel ↔ View

Model
Stores model state, provides business logic

ViewModel
Stores View state, communicates with model

View
Bindings



Add new Person

@Twitter Username

Add

Add new Person

nsmeetup

Add

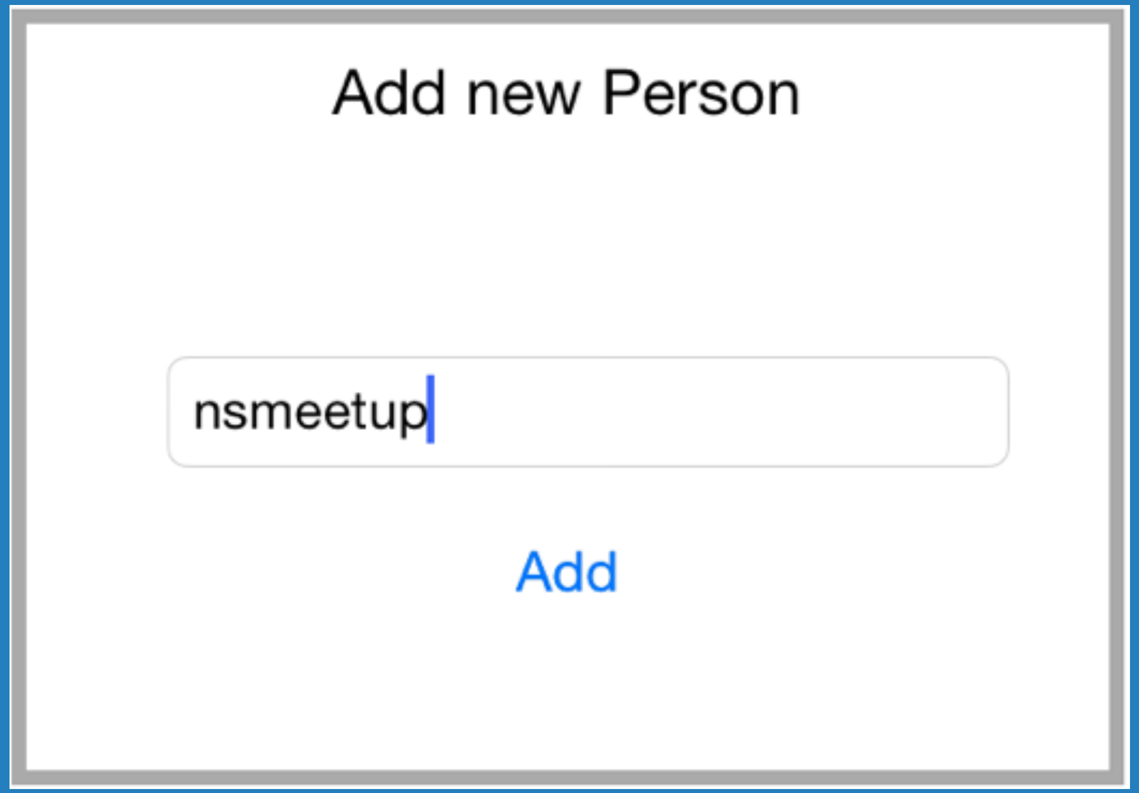
NSMeetup

A Monthly iOS/OSX Developer Meetup led by @stevederico

NSMeetup

Edit

$$\begin{tikzcd}
\textbf{PersonAddingViewModel} & \textbf{PersonAddingView*} \\
\text{usernameSearchText} \arrow[r, leftrightarrow] & \text{usernameTextfield.text} \\
\text{addButtonCommand} \arrow[r] & \text{addButton.rac\_command} \\
\text{addButtonEnabledSignal} &
\end{tikzcd}$$



*some variables have been renamed for brevity

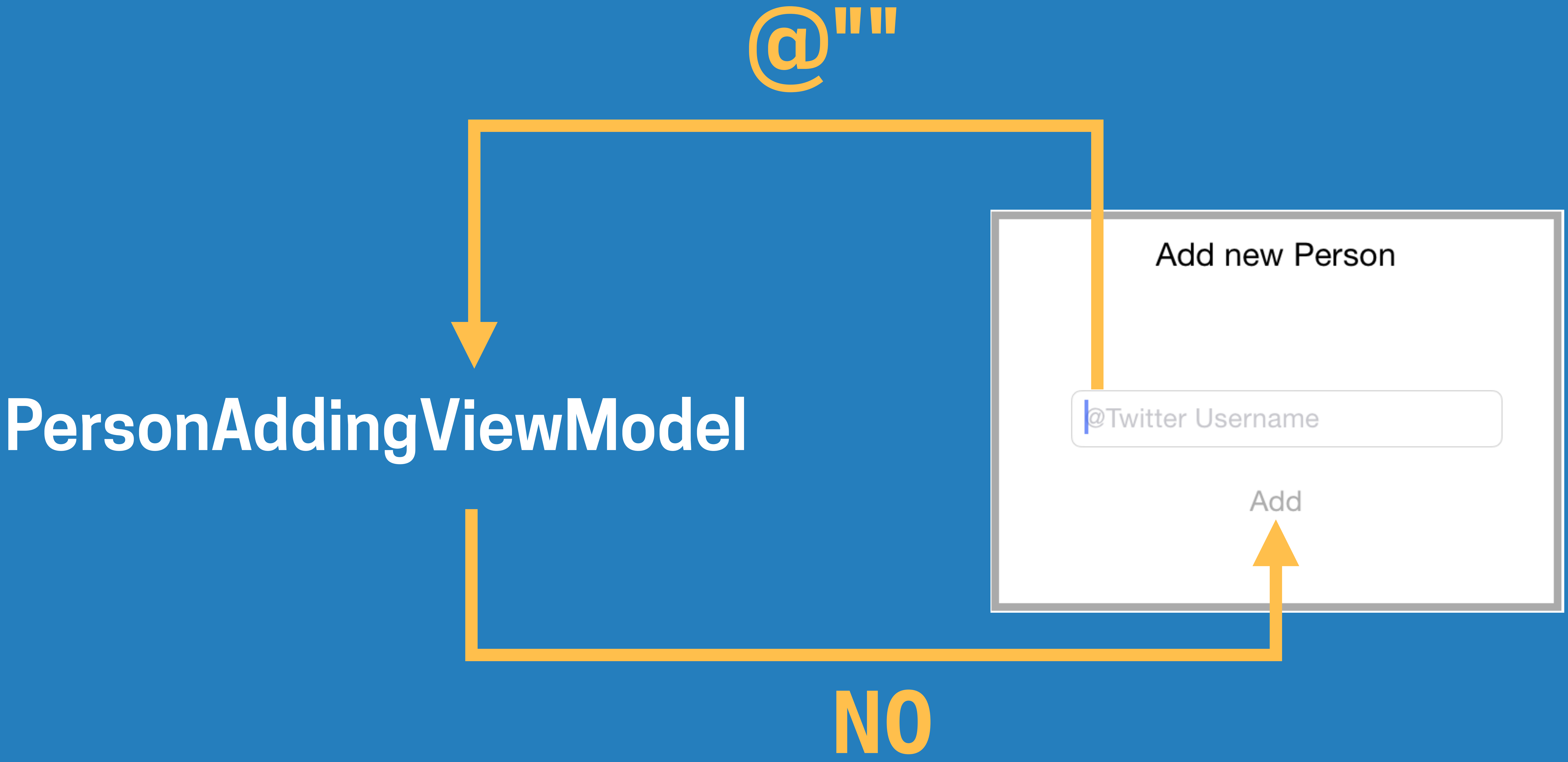
PersonAddingView
Initialization

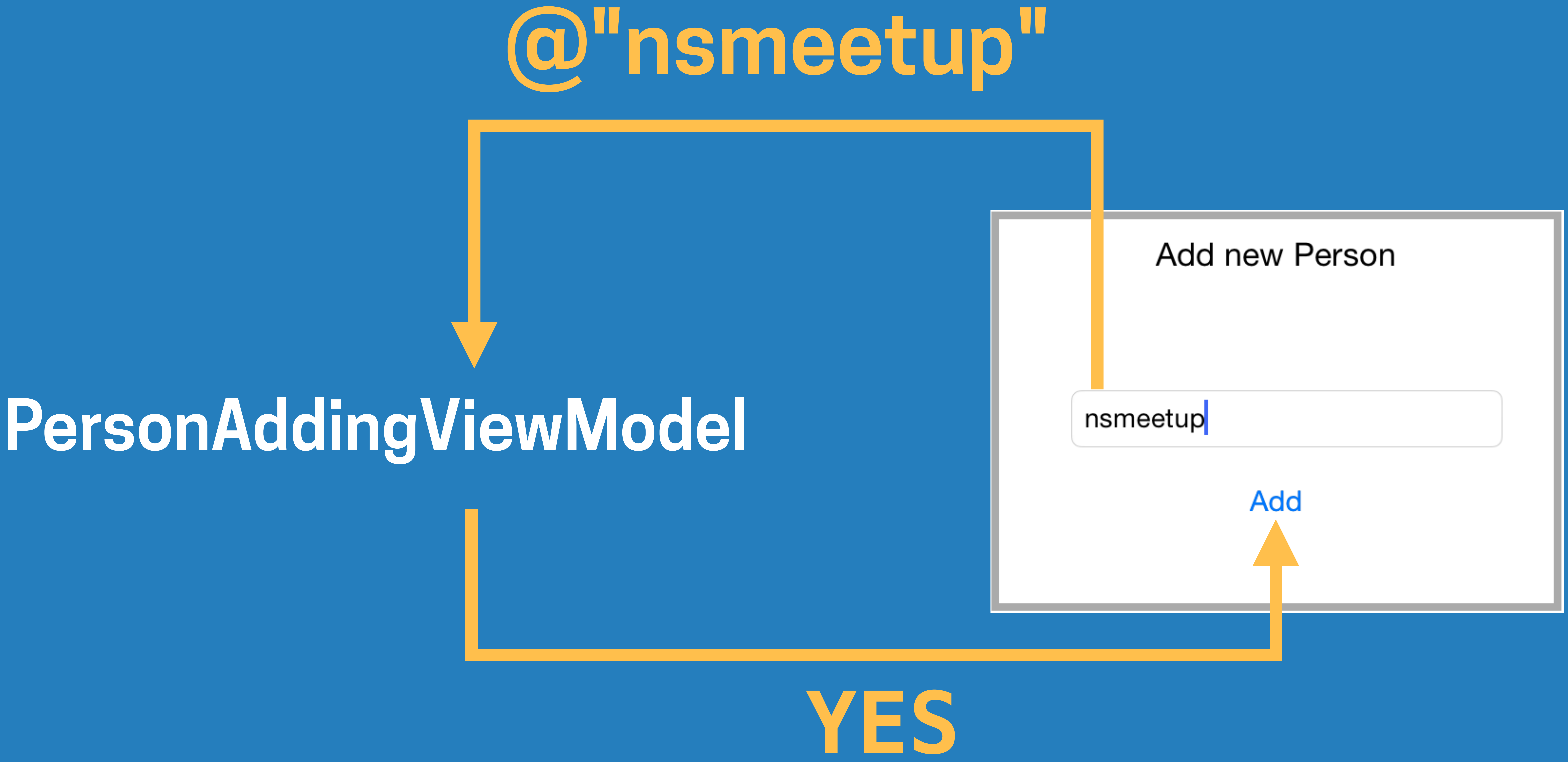
```
self.addTwitterButton.rac_command =
    self.viewModel.addTwitterButtonCommand;

RAC(self.usernameTextfield, enabled) =
     self.viewModel.textFieldEnabledSignal;
```

PersonAddingViewModel

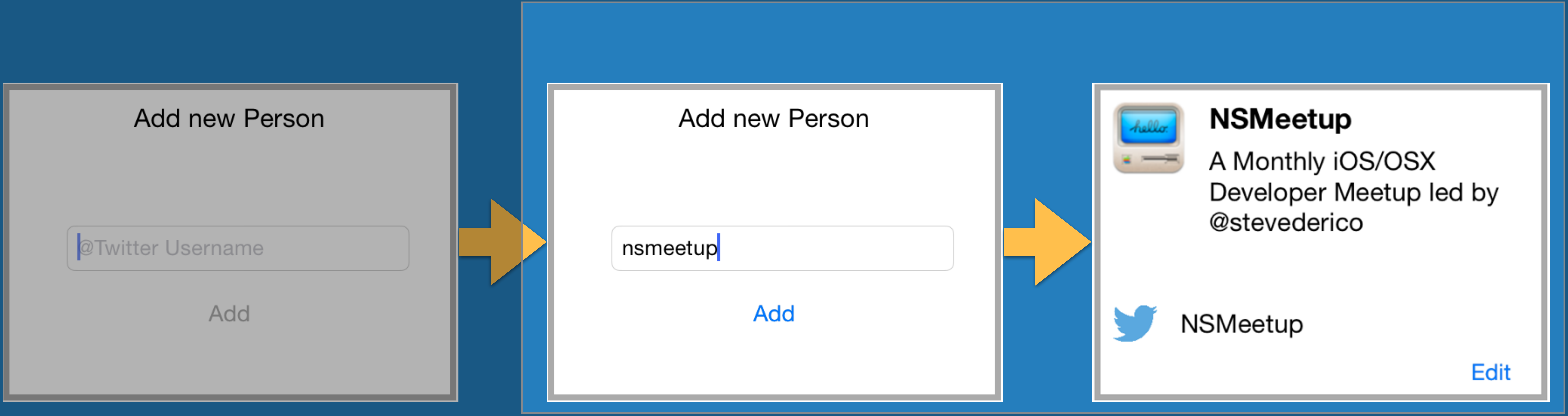
Enabling / Disabling the add button

```
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
Add new Person
nsmeetup
Add
YES

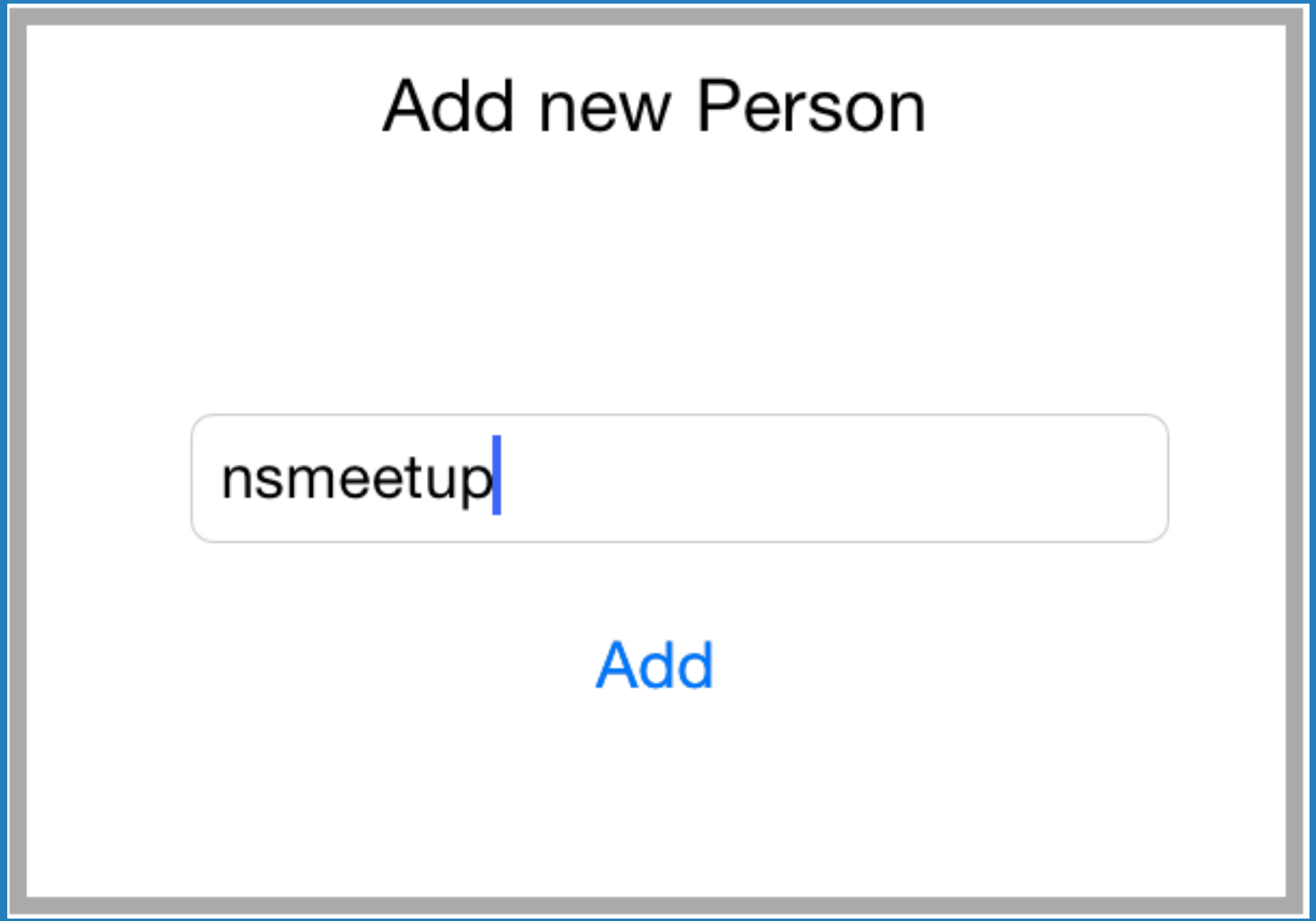
Networking with Reactive Cocoa


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

PersonContainerView

PersonAddingView

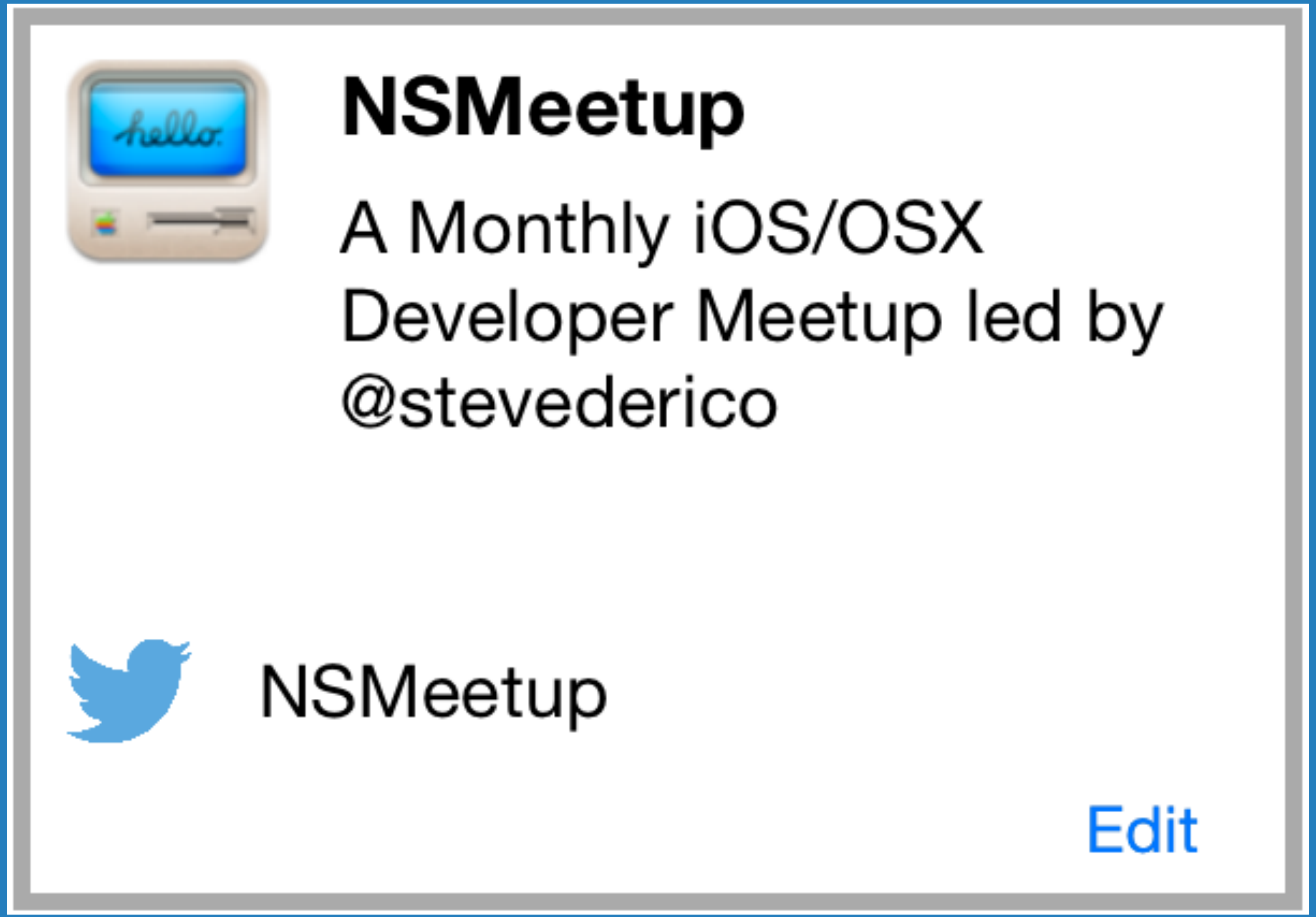
PersonDetailView



Add new Person

nsmeetup

Add



NSMeetup

A Monthly iOS/OSX Developer Meetup led by @stevederico

NSMeetup

Edit

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;
    }
];
```

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;
    }
];
```

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;
    }
];
```

We are doing exactly *one* thing. We don't need to handle callbacks here, just start the request!

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

Changing the UIState upon completed request

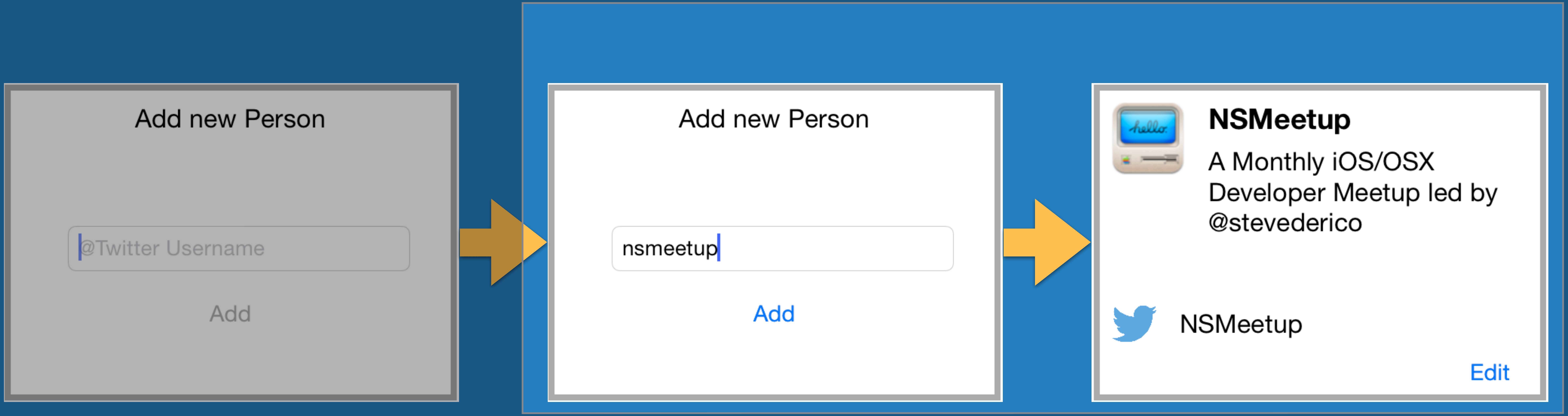
```
// subscribe to twitter network request
RACSignal *twitterFetchSignal = [RACObserve(self, personAddingViewModel)
   flattenMap:^RACStream *(PersonAddingViewModel *addingViewModel) {
      return [addingViewModel.addTwitterButtonCommand.executionSignals
              concat];
}];

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

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

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;
RAC(self, person) = twitterFetchSignal;
```



A Monthly iOS/OSX Developer Meetup led by @stevederico

Twitter API request

Chaining network operations

```
- (RACSignal *)infoForUsername:(NSString *)username {
 ...
 return [[[[[self _login] deliverOn:bgScheduler]
          flattenMap:^RACStream *(STTwitterAPI *client) {
            return [self client:client fetchUserInfo:username];
          }]
          flattenMap:^RACStream *(NSDictionary *userInfo) {
            return [[self imageFromURLString:userInfo[@"userInfo"]]
                    combineLatestWith:[RACSignal return:userInfo]];
          }]
          flattenMap:^RACStream *(RACTuple *personInfoTupel) {
            return [RACSignal return:[self
                    _personFromUserInfo:personInfoTupel]];
          }];
}
```

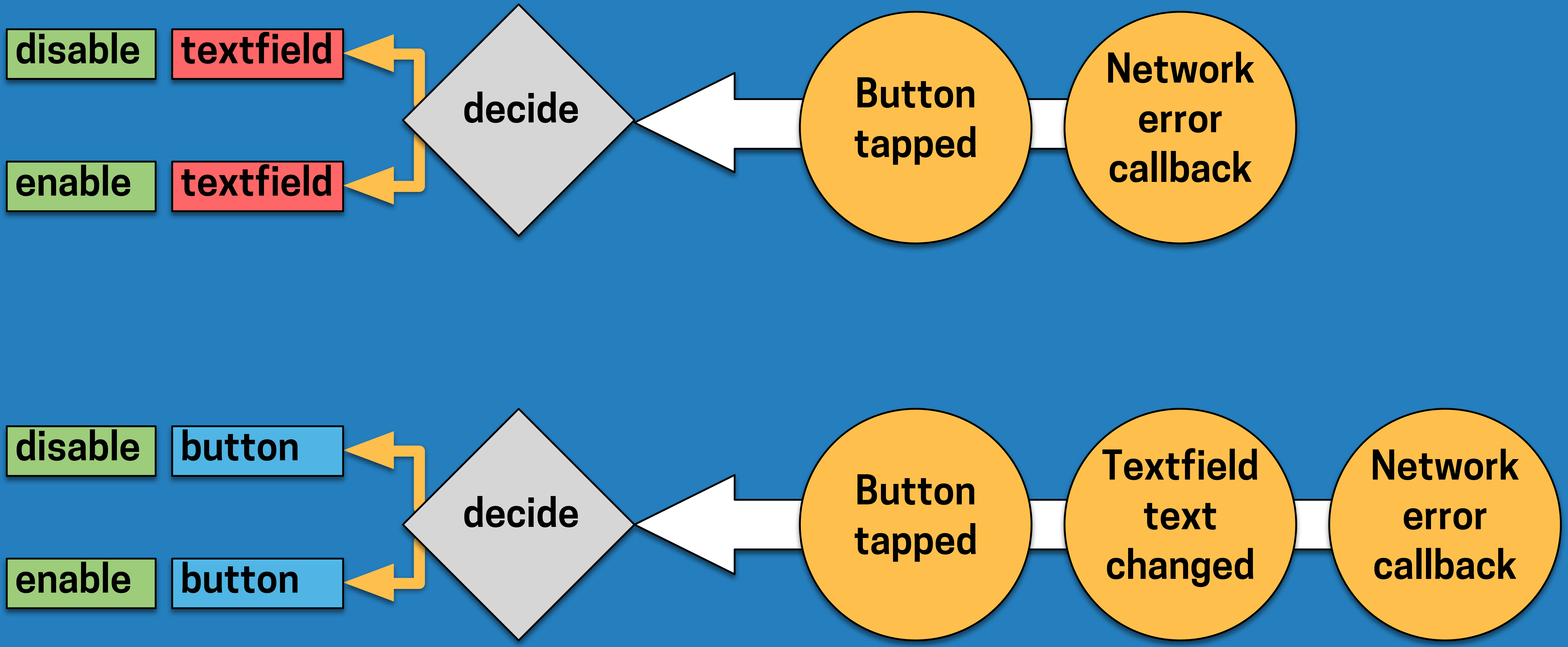
Model ⟷ ViewModel ⟷ View ✓

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"]);
});
```

Summary


disable
textfield
enable
textfield
decide
Button tapped
Network error callback
disable
button
enable
button
decide
Button tapped
Textfield text changed
Network error callback

- RAC provides tools for writing simpler declarative code
- Signals are unified way of handling different types of future values
- Input and outputs of Signals are well defined, state is derived in stateless function

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

“[...] our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed.“

(E.W. Dijkstra)

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