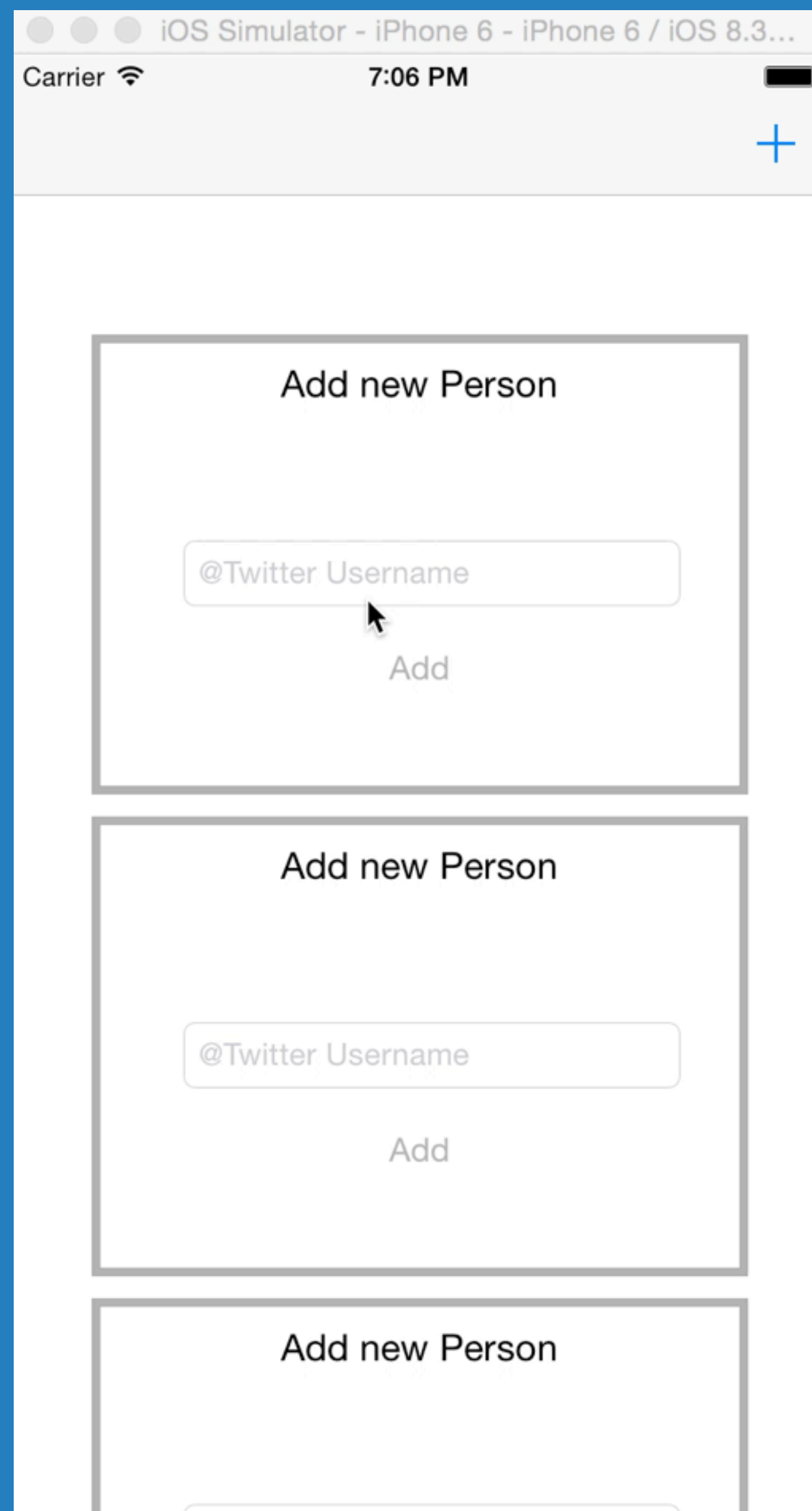


Functional Reactive Programming on iOS



3 High level states

No entry

Add new Person

Add

Default

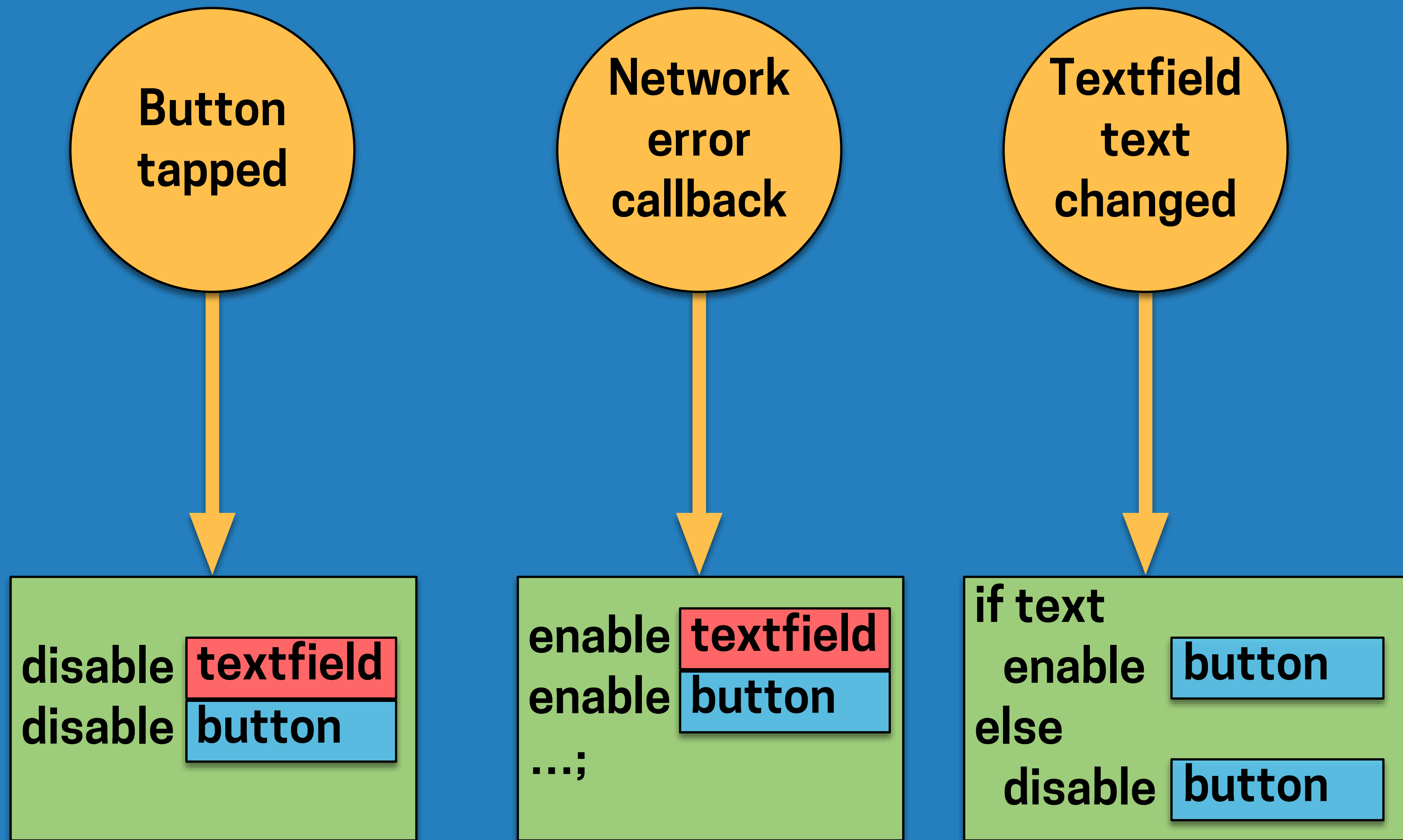
Add new Person

Add

Loading

Add new Person

Add



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

Add new Person

Add

Default

Add new Person

Add

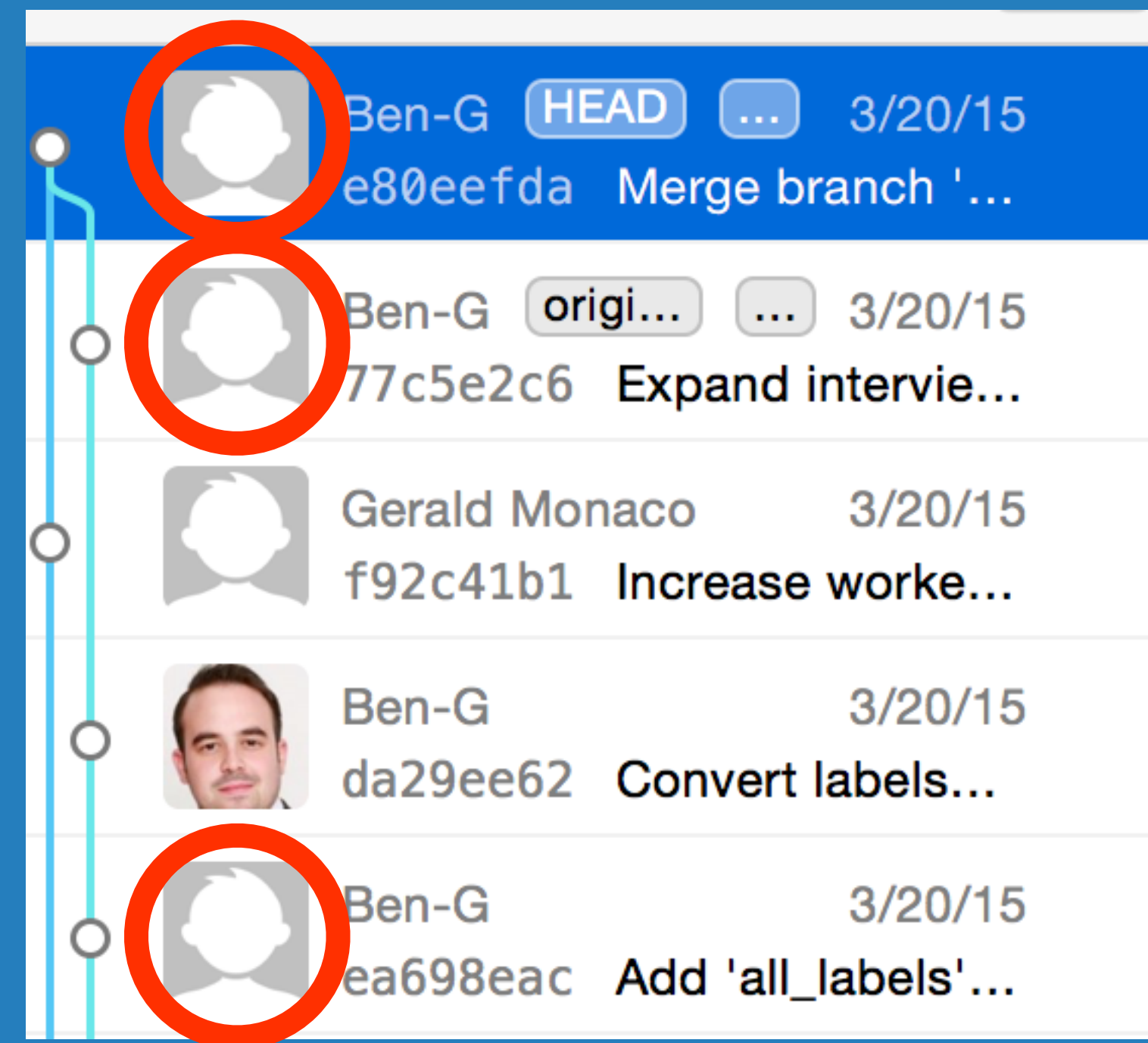
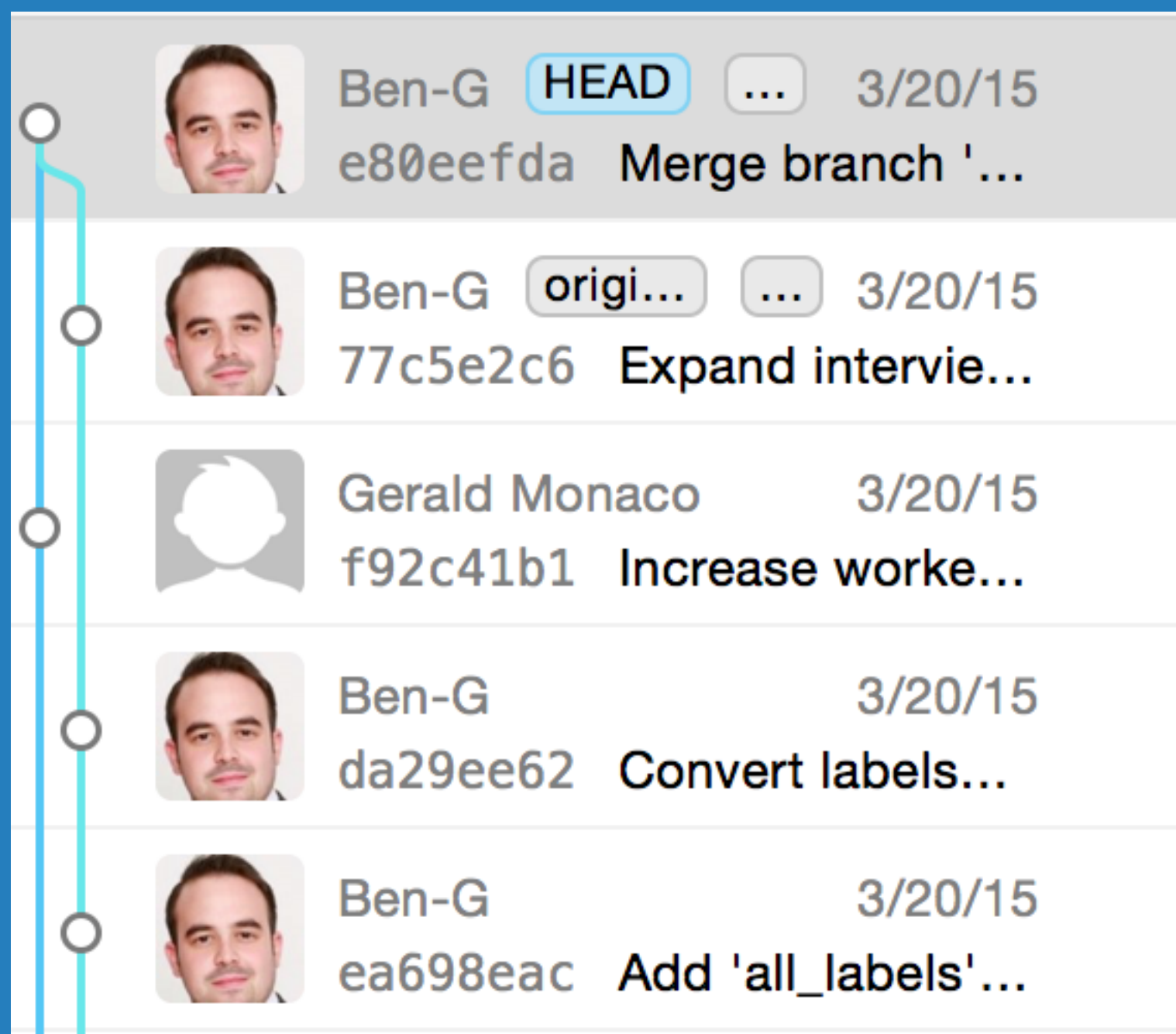
Loading

Add new Person

Add

9 possible invalid states!

Manual state management is error prone



What is functional reactive programming?

Imperative vs. Declarative

Imperative

A	B	C
20	10	?

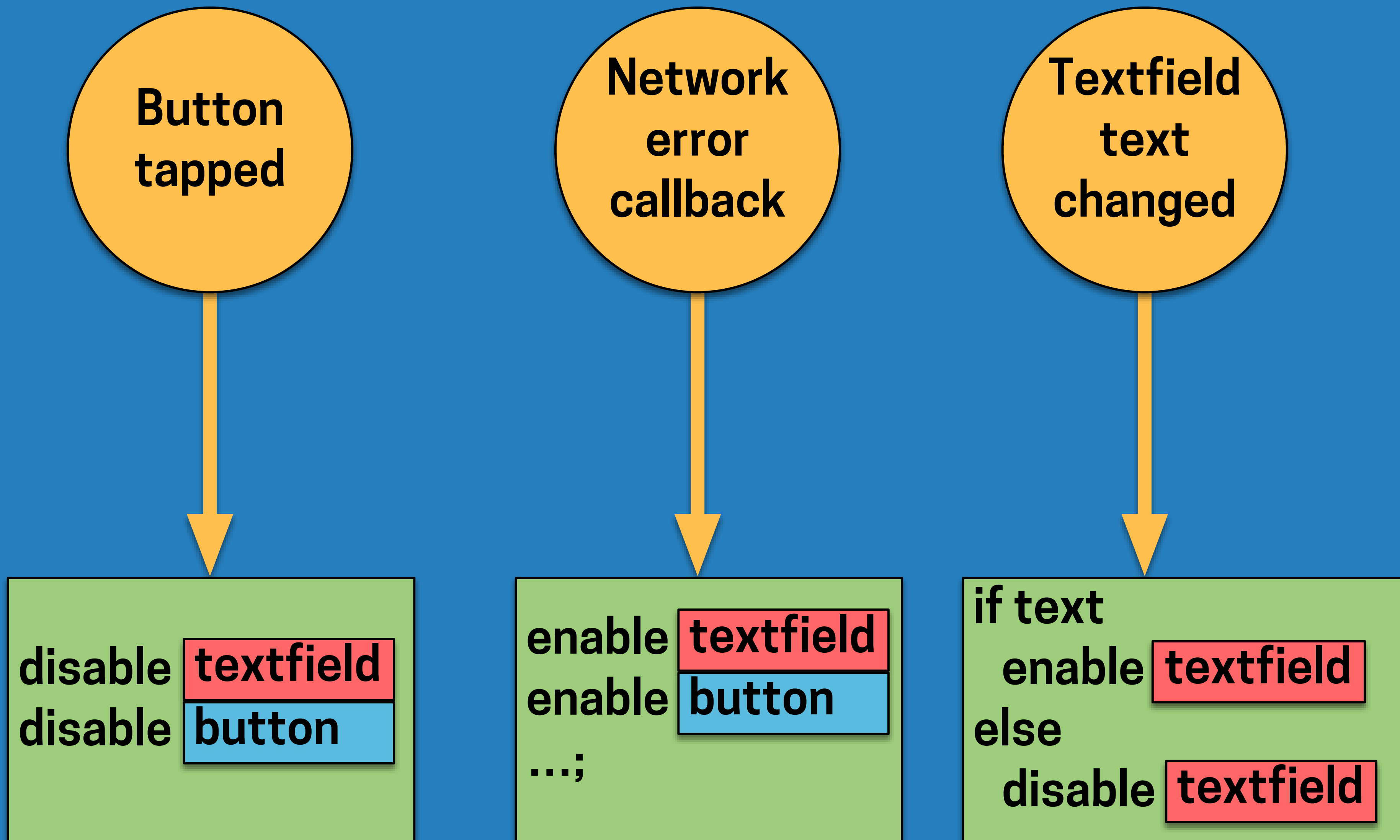
1. Perform the following steps whenever A or B changes
2. Add 50 to value of A
3. Subtract 10 from value of B
4. Add the results from 2.) and 3.)
5. Write result from 4.) into C

Declarative

A	B	C
20	10	?

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

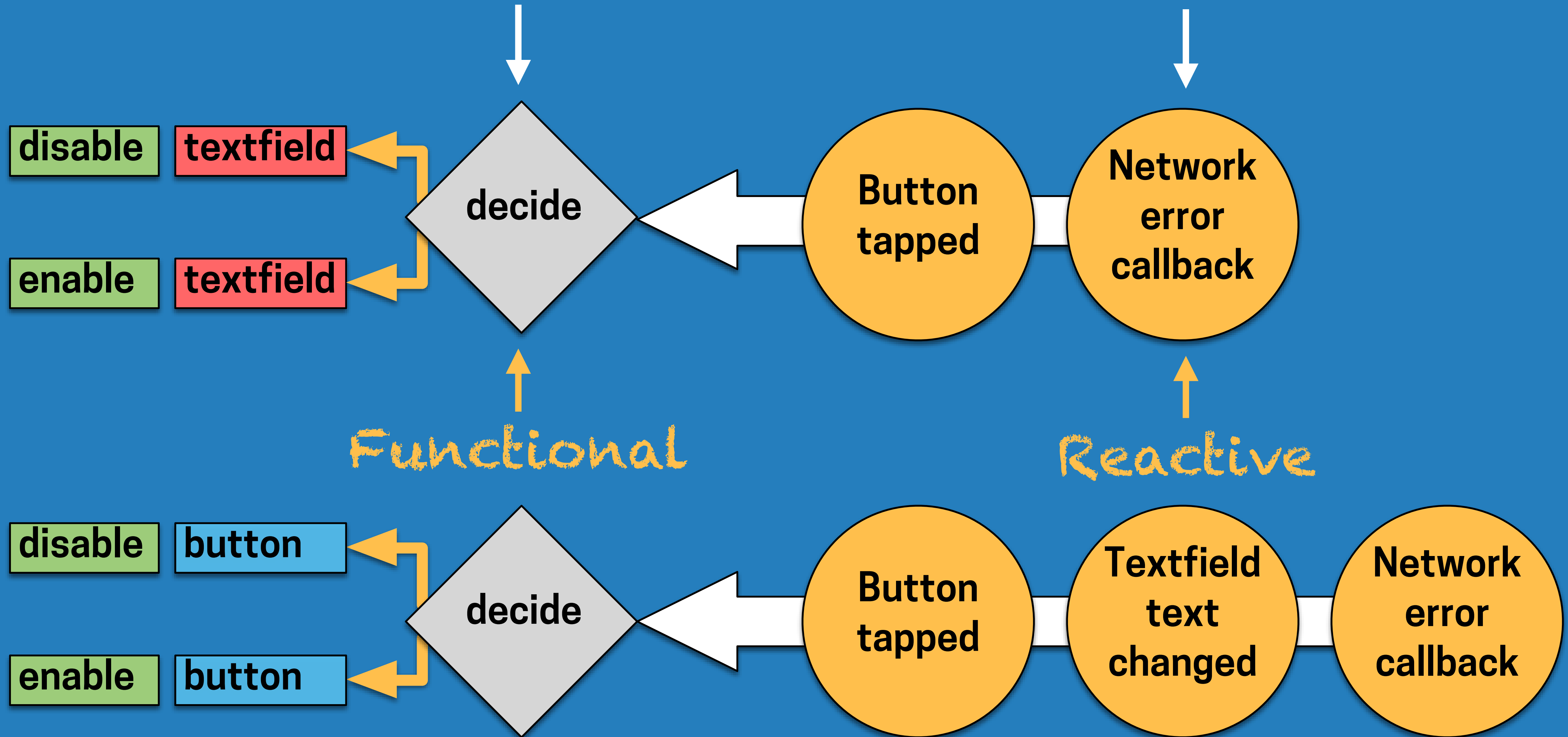
Imperative



Declarative

Stateless Function

Event emitters



State is **derived** from a
defined **set of inputs**

Intro to Reactive Cocoa 2.x

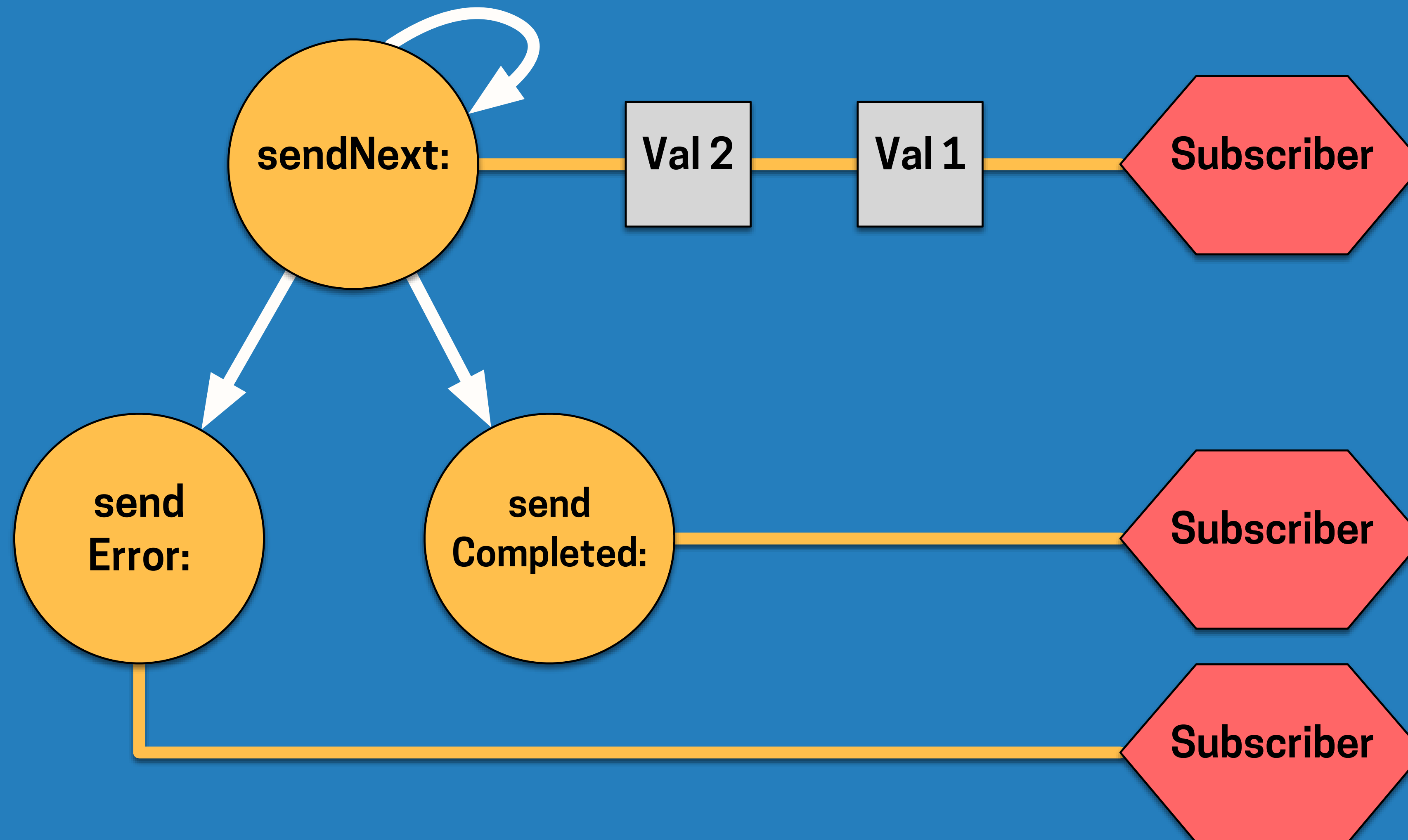
Signals send values over
time

- Callbacks
- Delegate methods
- KVO / Property overriding

} Signals

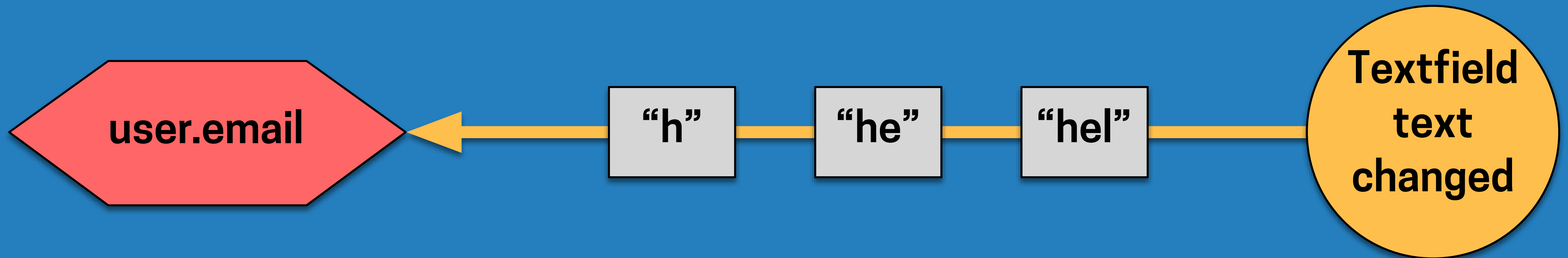
↑
values over time

RACSignal



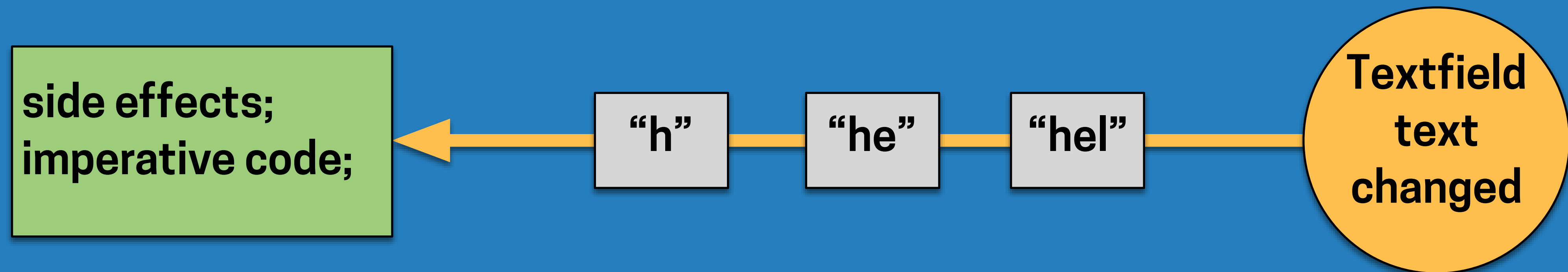
We can **bind** Signals
OR
subscribe to Signals

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  
}
```

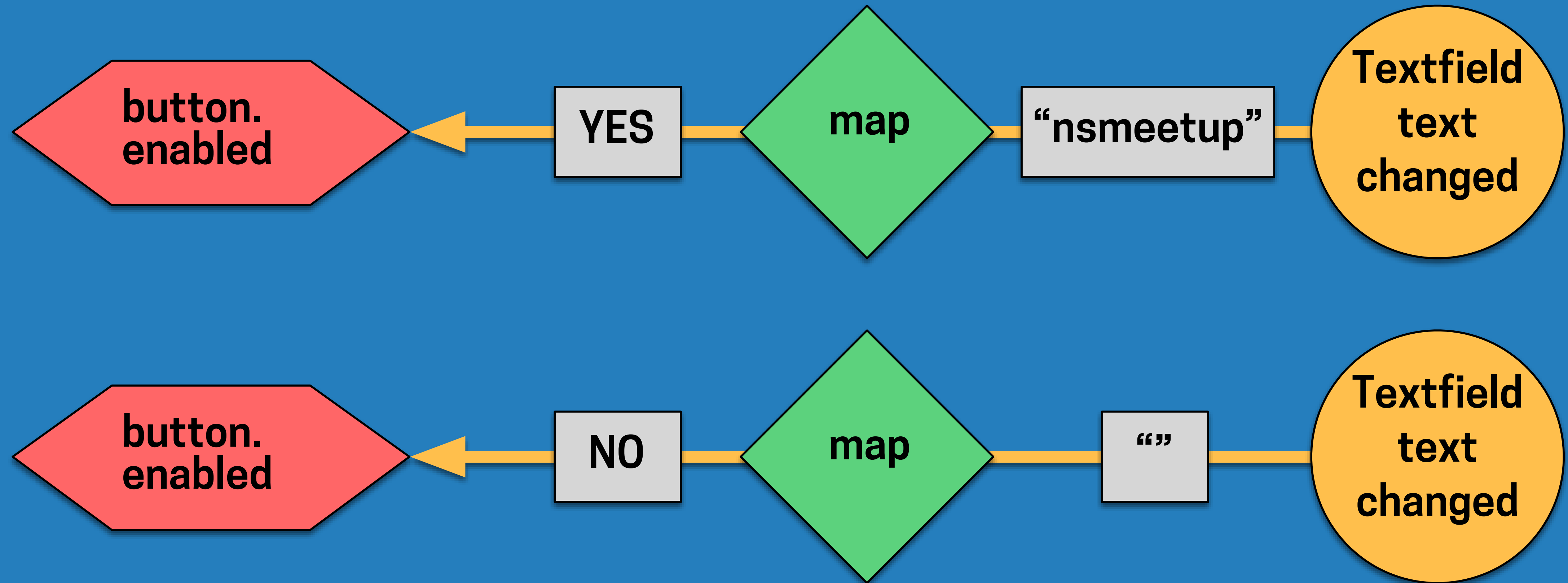
**View updates whenever
model or model properties change**



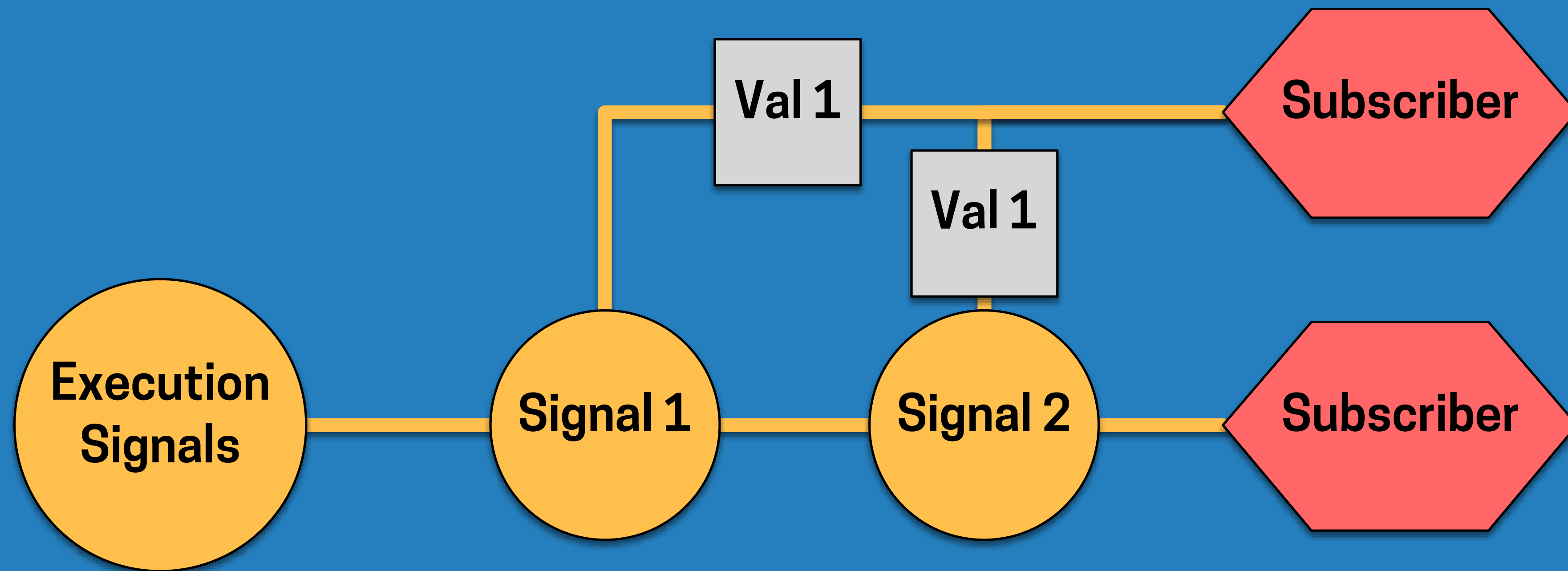
Model  **View** 


Model  View ?

Signal Operators



RACCommand



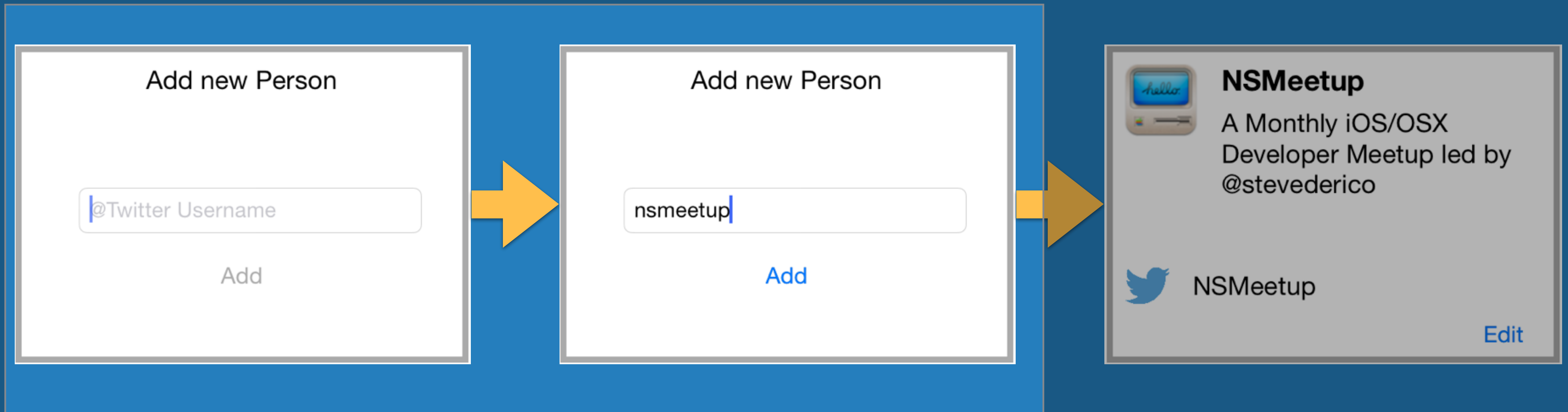
Model  View ?

Model ↔ **ViewModel** ↔ **View**

Stores model state,
provides business
logic

Stores View state,
communicates
with model

Bindings



PersonAddingViewModel

usernameSearchText

addButtonCommand

addButtonEnabledSignal

PersonAddingView*

usernameTextfield.text

addButton.rac_command



Add new Person

nsmeetup

Add

*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

Add

NO

@ "nsm meetup"



PersonAddingViewModel



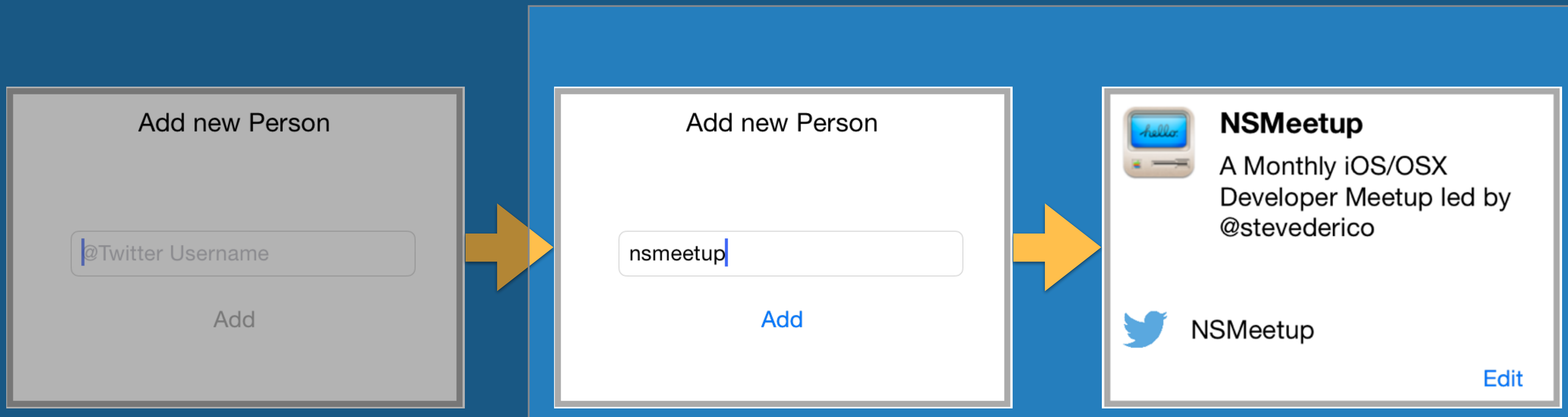
Add new Person

nsmeetup

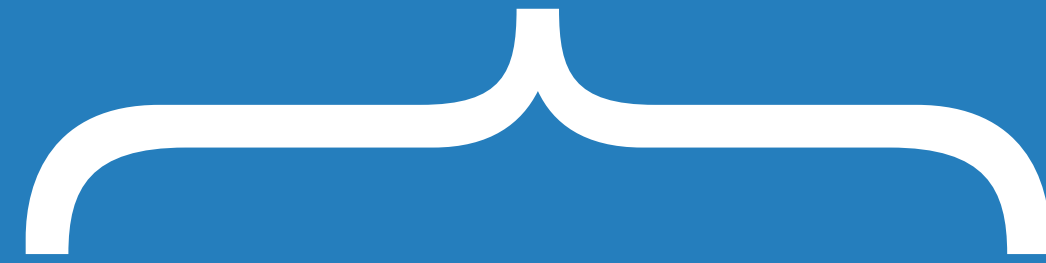
Add

YES

Networking with Reactive Cocoa



PersonContainerView



PersonAddingView

Add new Person

Add

PersonDetailView



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 *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 value) {
    return @(DetailViewState);
}] startWith:@(AddingViewState)];

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

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 value) {
    return @(AddingViewState);
}] startWith:@(DetailViewState)];
```

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

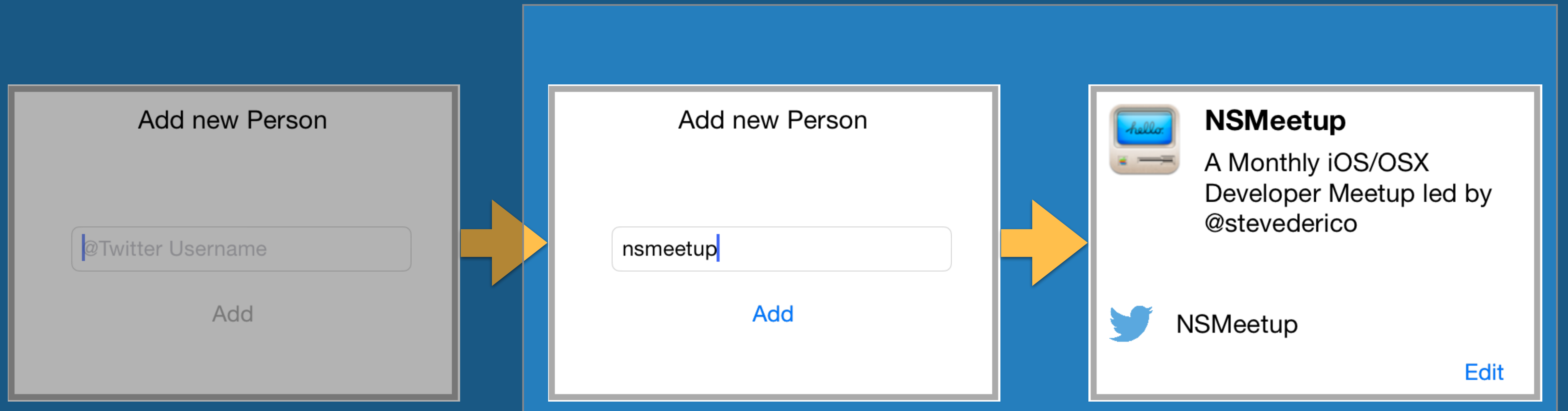

PersonContainerViewModel

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 @(AddingViewState);
}] startWith:@(DetailViewState)];
```

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

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 *personInfoTuple) {
            return [RACSignal return:[self
                _personFromUserInfo:personInfoTuple]];
        }]];
}
```

Model ↔ ViewModel ↔ View



Testing with Reactive Cocoa 2.x

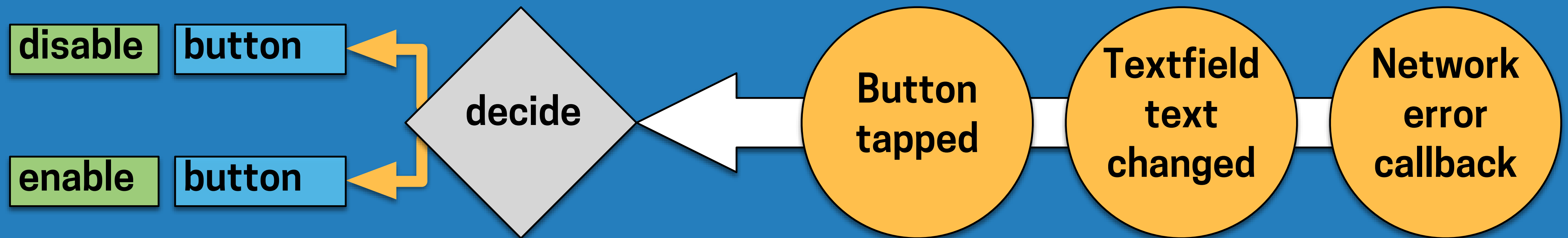
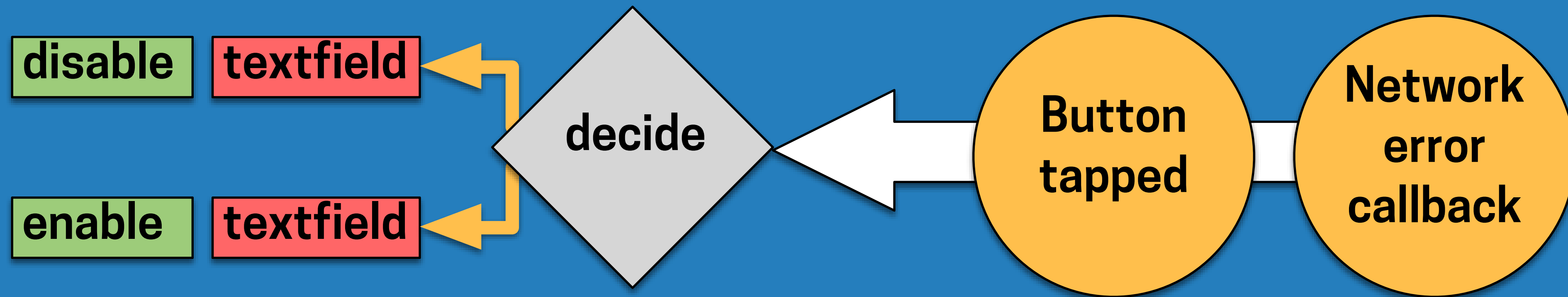
Testing UI **without** UIKit

```
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



- RAC introduces a vastly different programming model that can be harder to debug
- RAC provides tools for writing simpler **declarative code** that embraces **derived state**

- **MVVM** plays nicely with bindings, eliminates controller complexity
- **MVVM** makes it easier to write testable code

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

E.W. Dijkstra

(And a ton of talks & blog posts that quoted him in the context of Reactive Programming)

Thank you!

- **Code:** <https://github.com/Ben-G/PeopleCRM>
- **Further Resources:**
 - <http://www.sprynthesis.com/2014/06/15/why-reactivecocoa/>
 - Functional Reactive Programming on iOS, Ash Furrow (<https://leanpub.com/iosfrp/>)

Thanks to Ash Furrow, Morgan Chen, Gerald Monaco, Florian Krueger and Dave Lee for input and feedback on this talk!