React + Flux

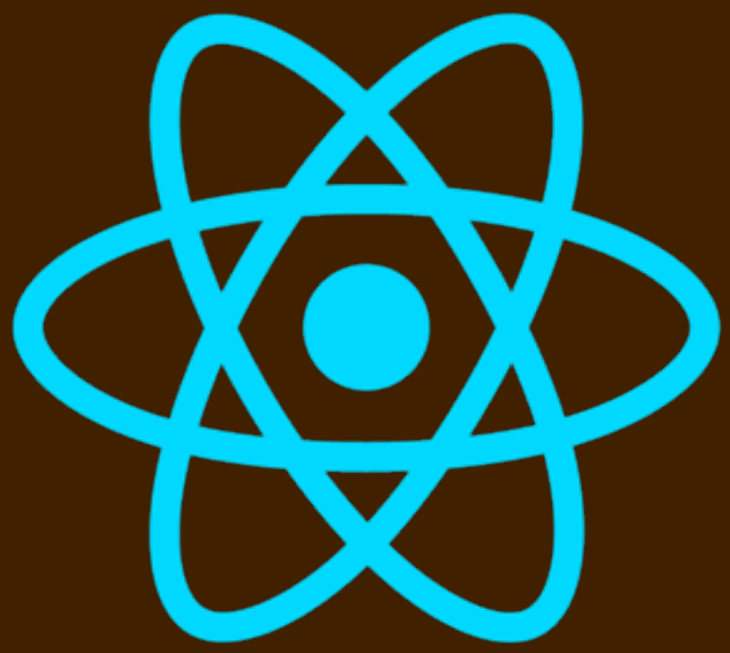
Two Great Tastes that Taste Great Together

Bill Fisher @fisherwebdev #ReactJS

https://speakerdeck.com/fisherwebdev/react-flux-fluent-2015

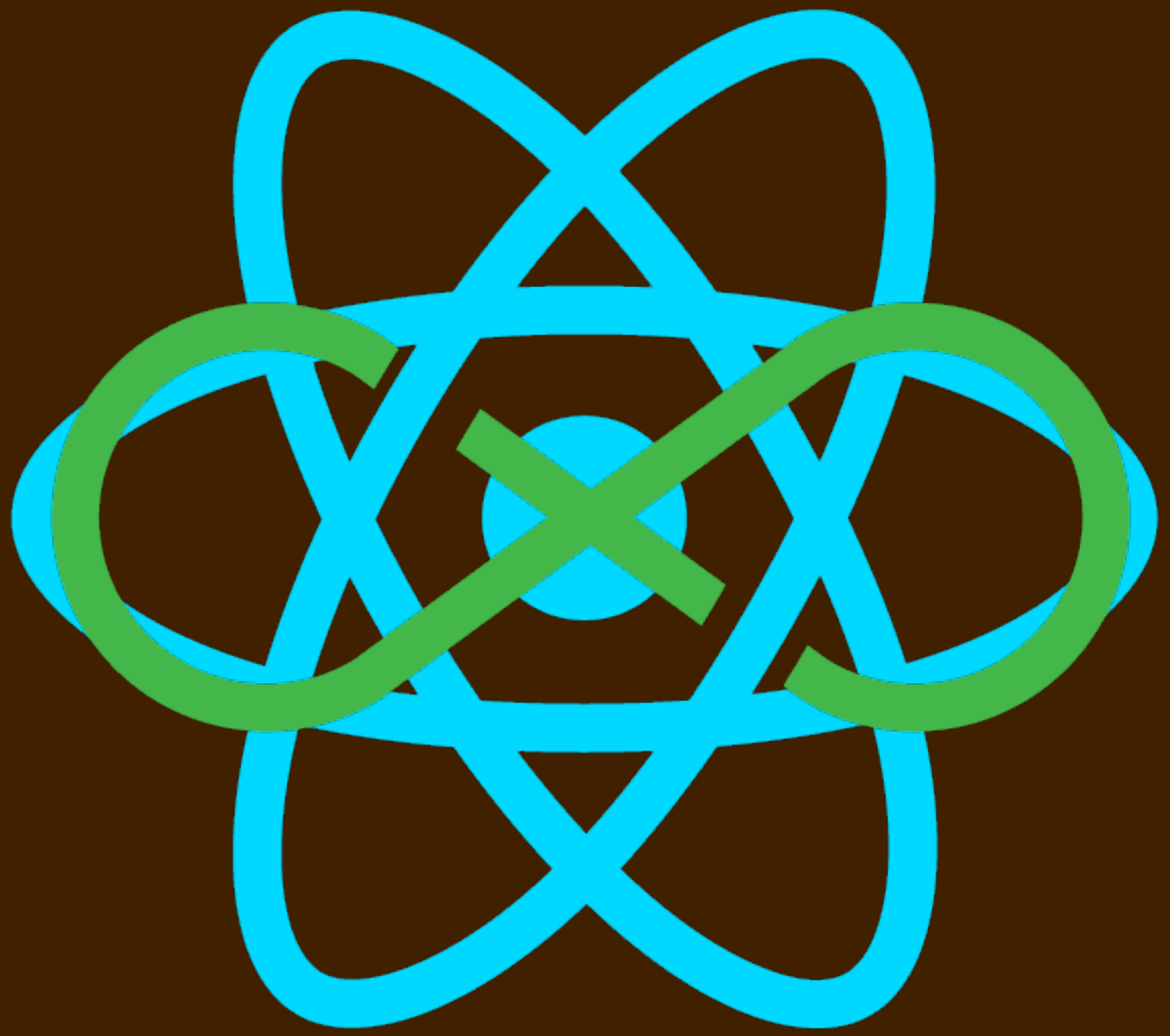
bit.ly/1Hrqfpc

https://speakerdeck.com/fisherwebdev/react-flux-fluent-2015

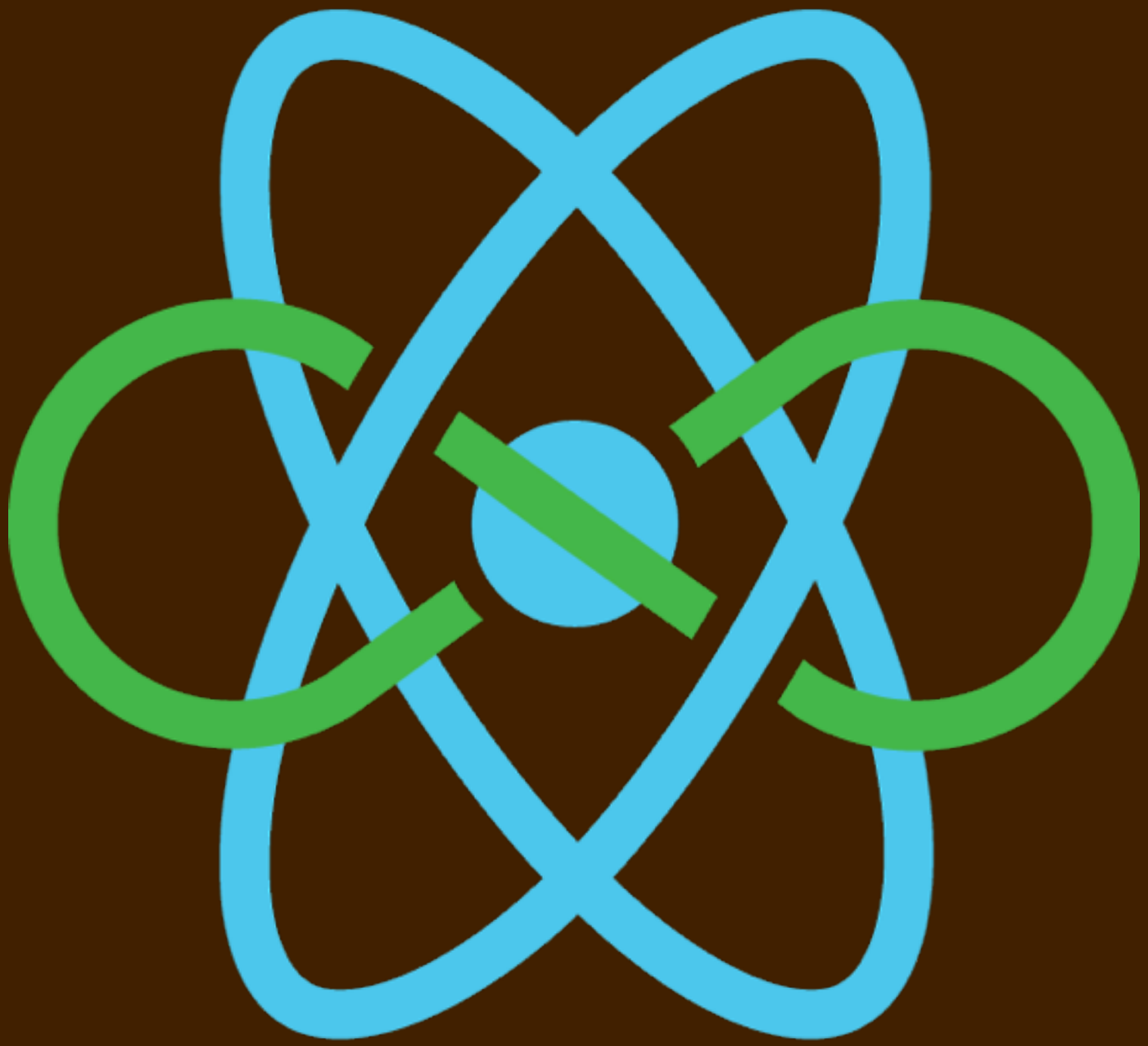


React + Flux

https://speakerdeck.com/fisherwebdev/react-flux-fluent-2015



https://speakerdeck.com/fisherwebdev/react-flux-fluent-2015

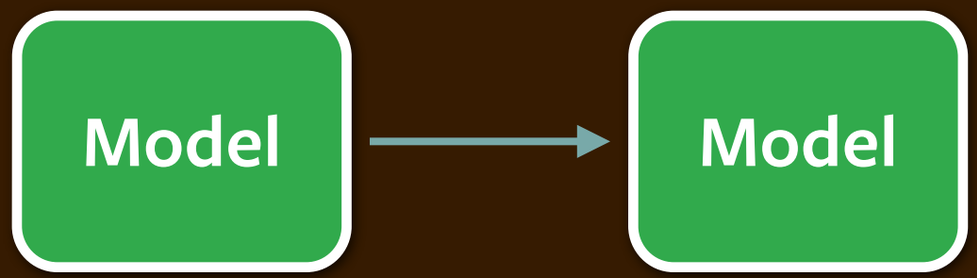


Model

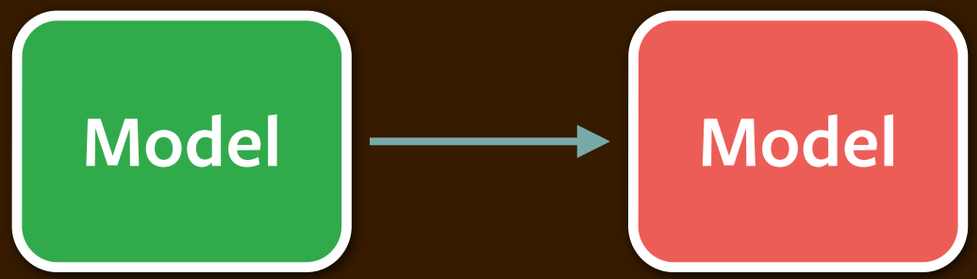
Model

Model

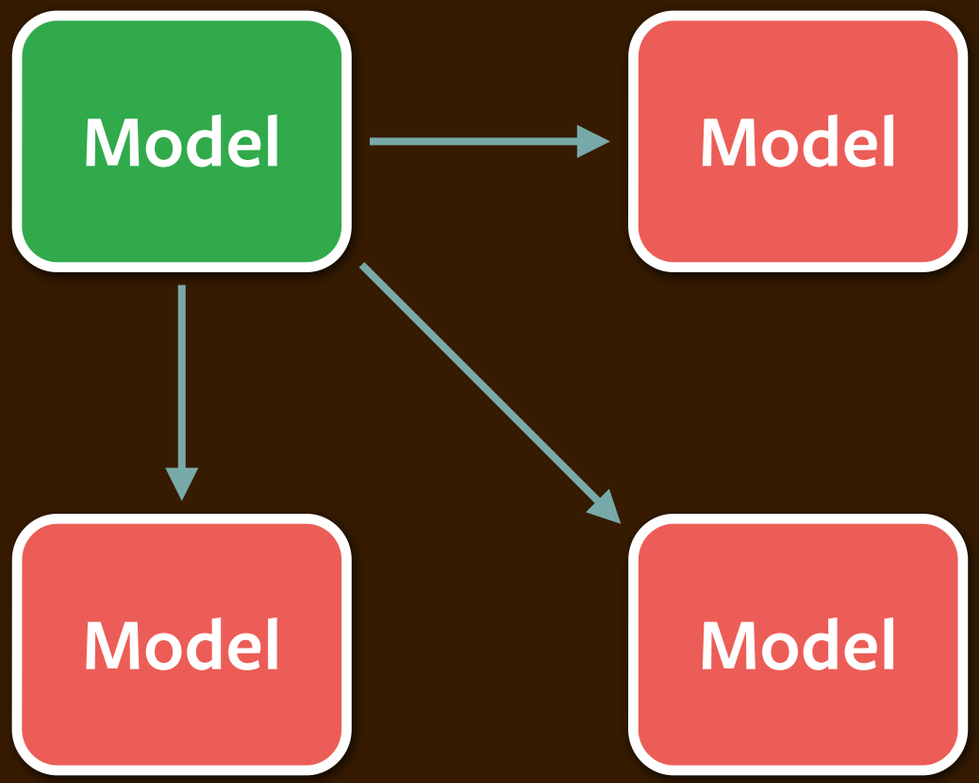
Model

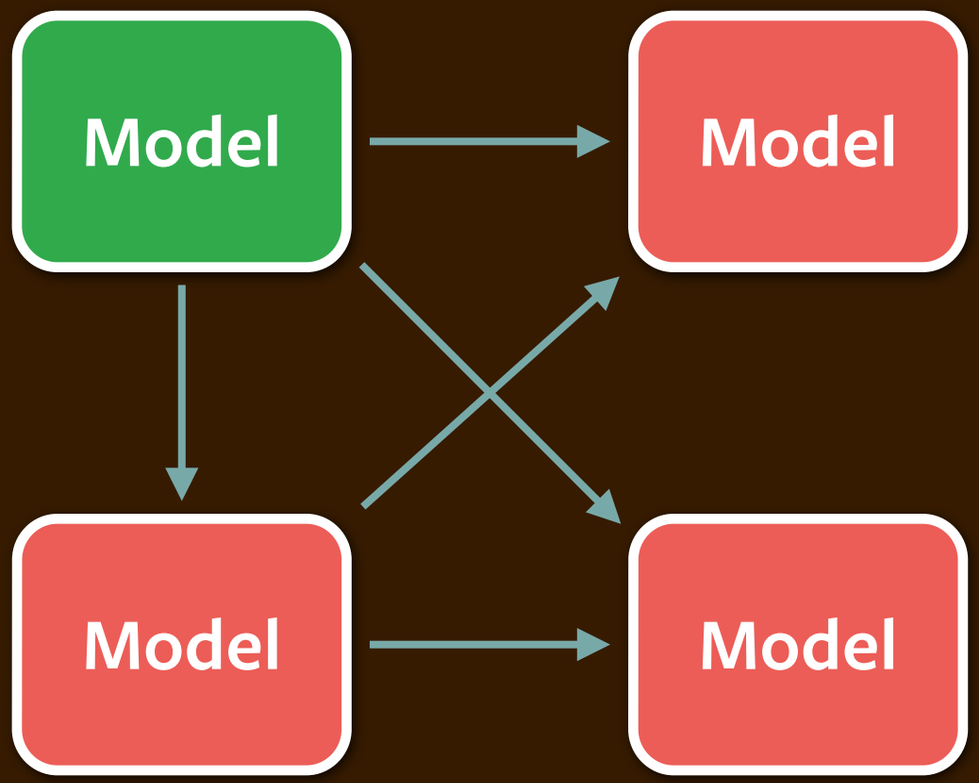

Model
Model

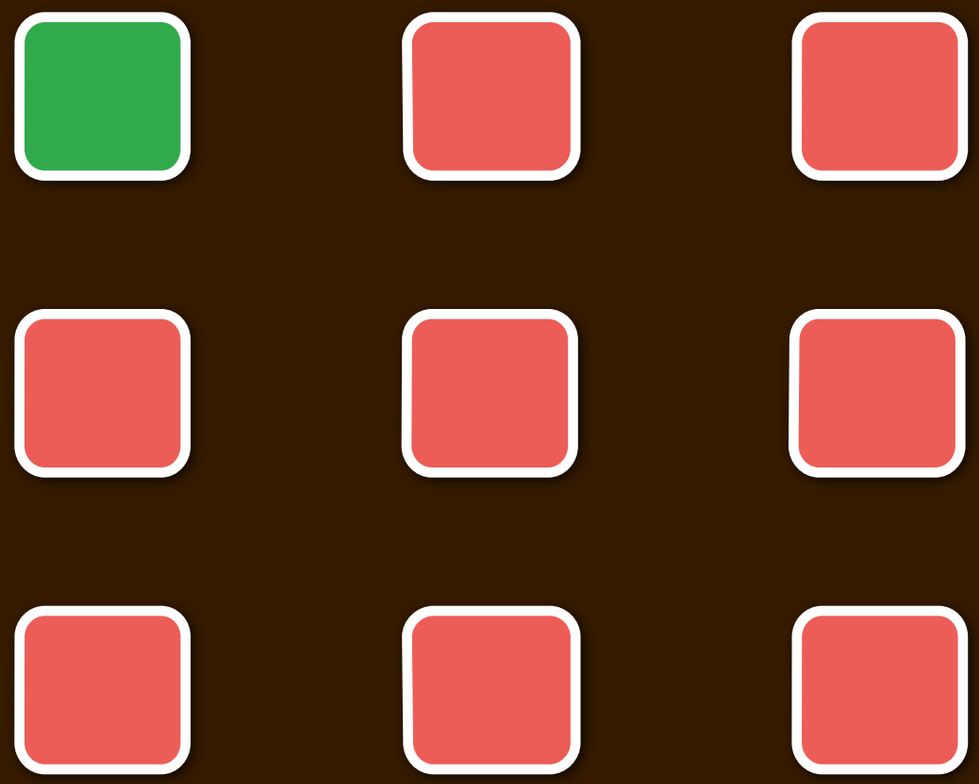

Model
Model

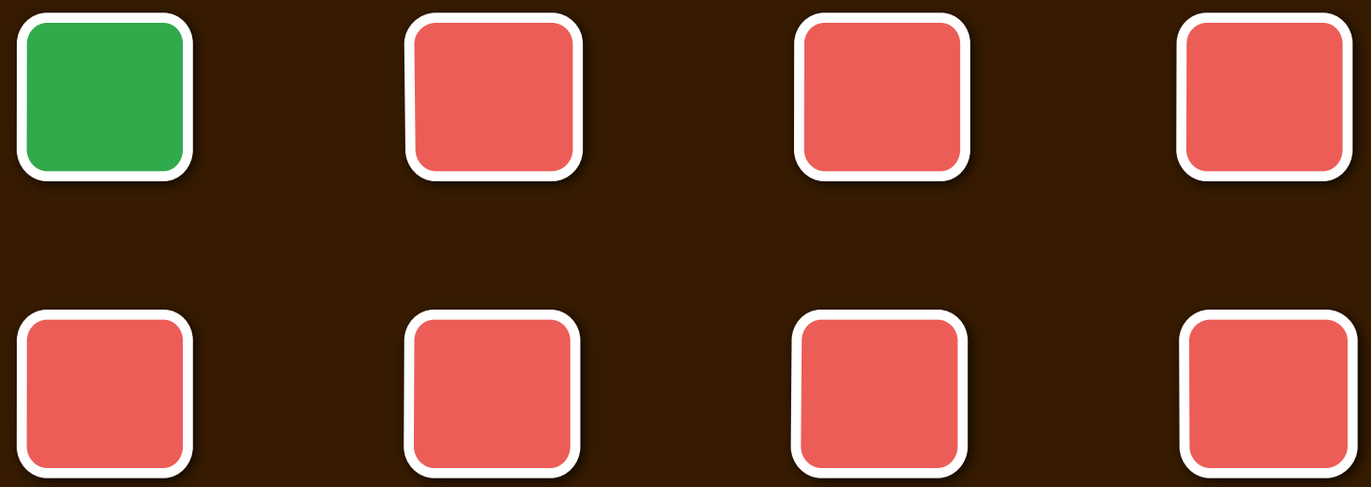

Model
Model

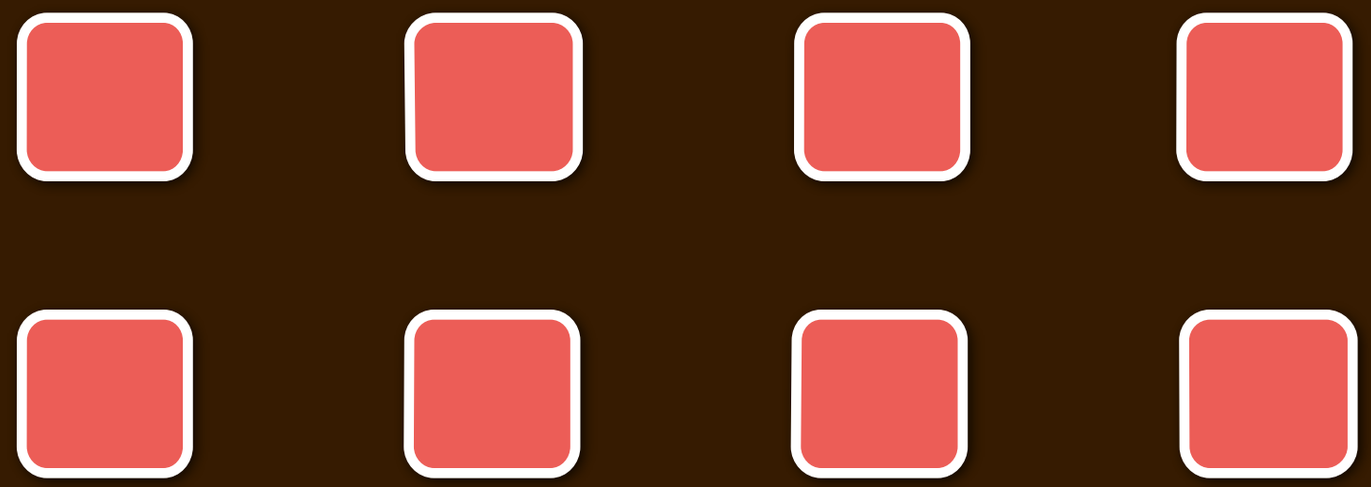

Model
Model


Model
Model
Model
Model
Model

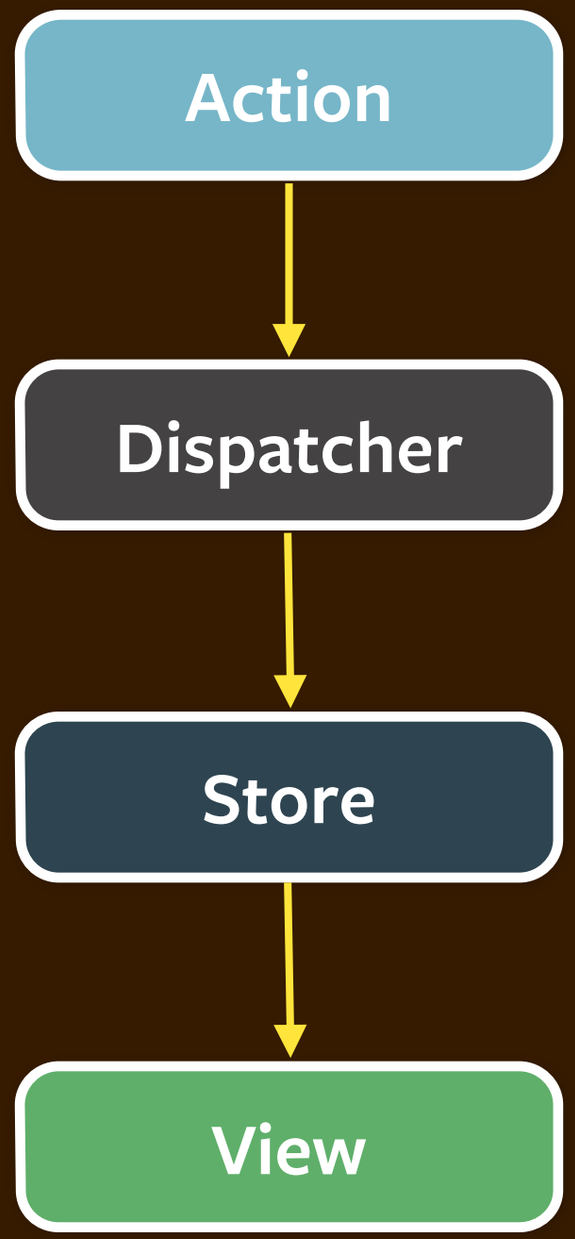

Model
Model
Model
Model
Model

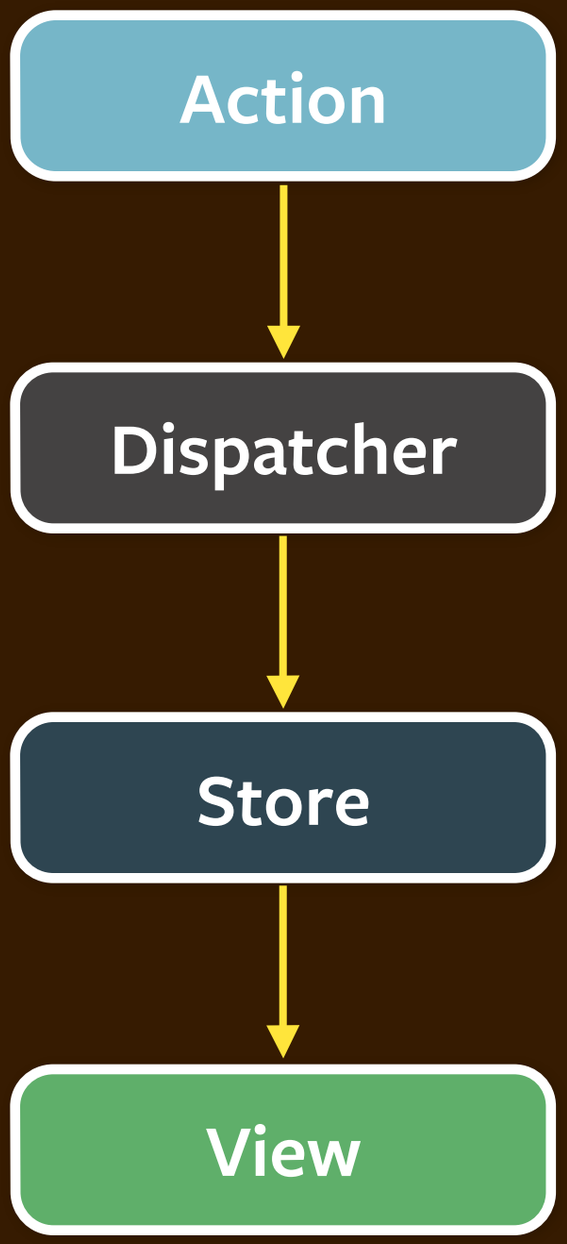




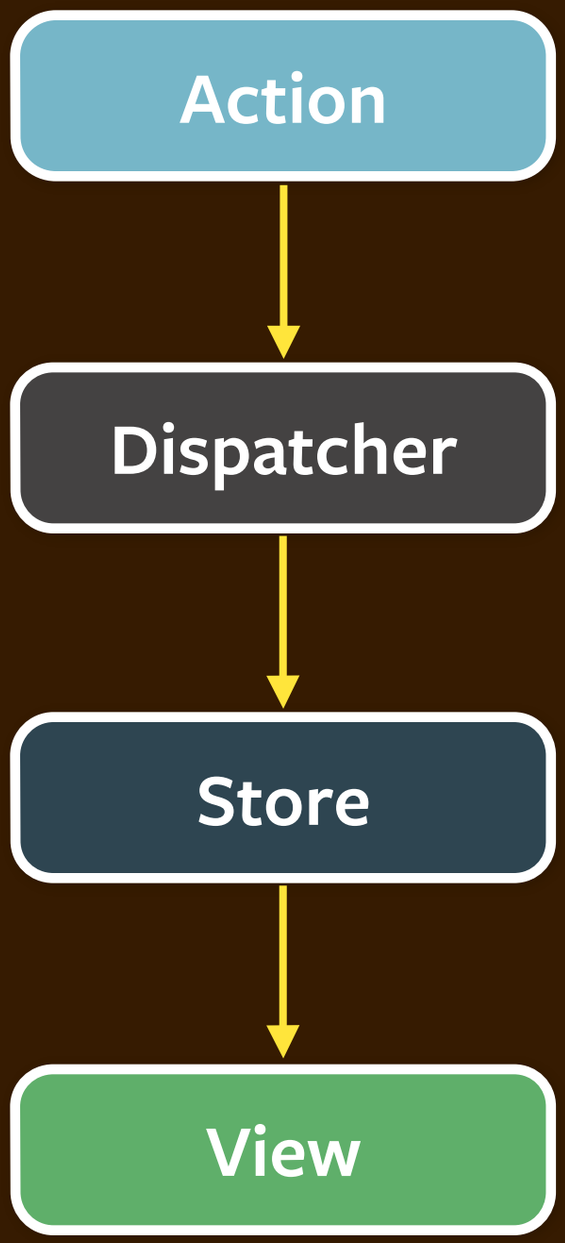


"Tangled wires, Freegeek, Portland, Oregon, USA" by Cory Doctorow, used under CC BY 2.0\


Action
Dispatcher
Store
View




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```

Actions & Action Creators


```
actions
└── MessageActionCreators.js
```

Actions & Action Creators

Actions:

- an object with a type property and new data, like events

Action creators:

- semantic methods that create actions
- collected together in a module to become an API

EXTRA EXTRA EXTRA EXTRA

The Weather
Bay Area: Fair today except night and morning fog. High, to 70s; low, 50s. Wind, 10-12 m.p.h.
See Page 51

San Francisco Chronicle

★★★ FINAL

105th Year No. 201 ★★★ MONDAY, JULY 21, 1969 10 CENTS GArfield 1-1111

MEN ON MOON

Man Makes a Lunar Landing --Astronauts Walk on Surface

A Sort of Holiday Today

Today is Moon Day—or, more officially, a National

How They Made the Descent

By David Perlman

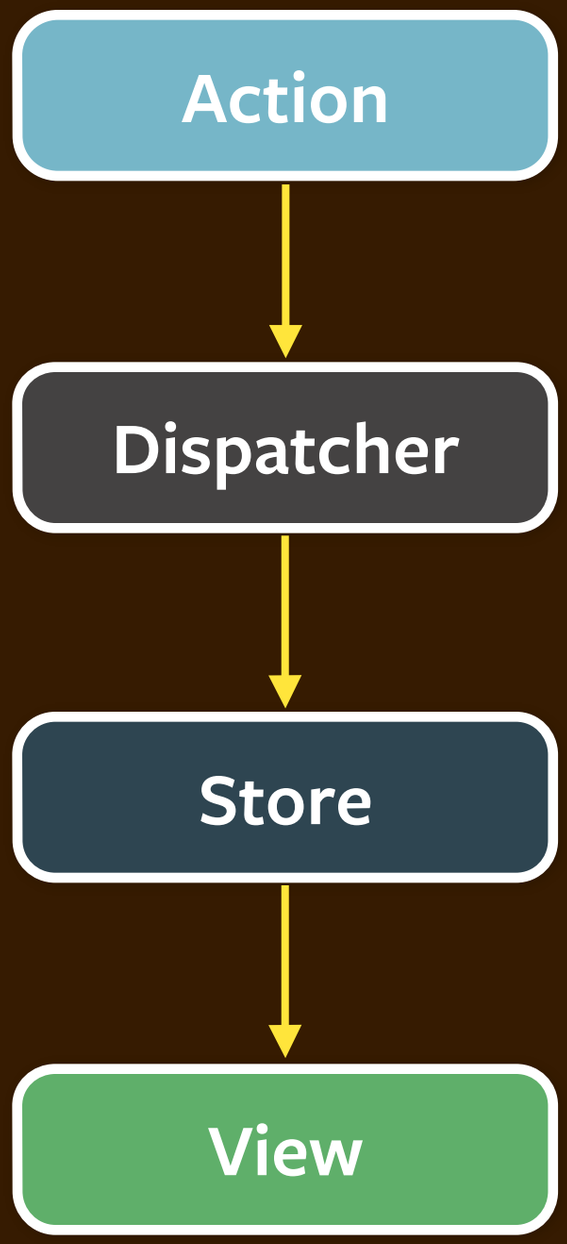
Historic Moment

Spacemen Scout The Landscape

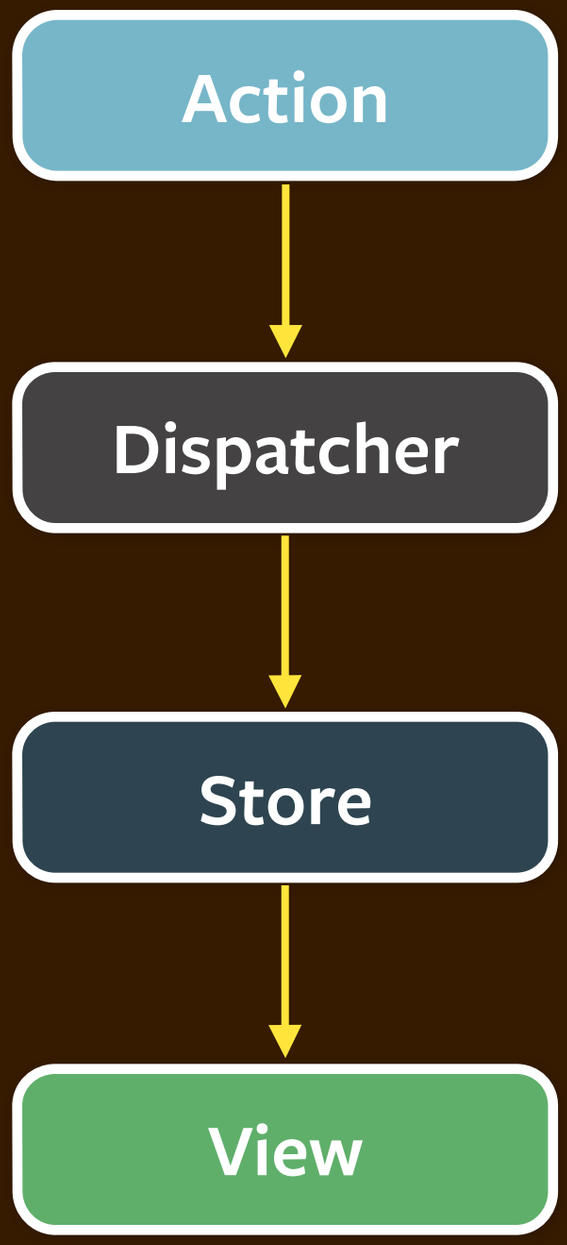
```
// MessageActionCreators.js

var AppDispatcher = require('../AppDispatcher');
var AppConstants = require('../AppConstants');

var ActionTypes = Constants.ActionTypes;

module.exports = {

  messageCreated: text => {
    AppDispatcher.dispatch({
      type: ActionTypes.MESSAGE_CREATED,
      text
    });
  },

};
```

```
  messageCreated: text => {
    AppDispatcher.dispatch({
      type: ActionTypes.MESSAGE_CREATED,
      text
    });
  },
```

```
  messageCreated: text => {
    AppDispatcher.dispatch({
      type: ActionTypes.MESSAGE_CREATED,
      text
    });
  }
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```



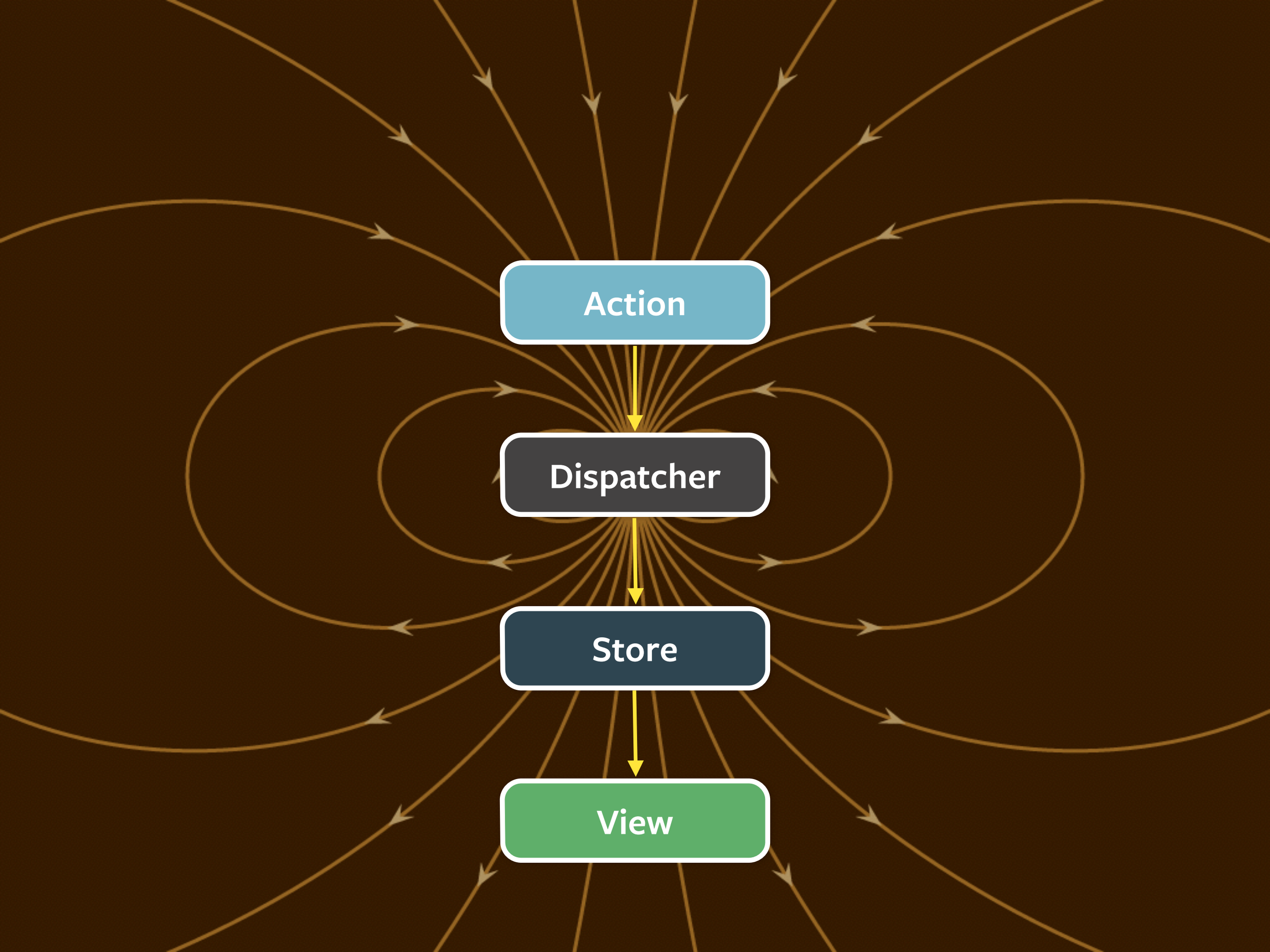

```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```

The Dispatcher

AppDispatcher.js

The Dispatcher

"Sunset at Sutro" by ohad, used under CC BY 2.0*


Action
Dispatcher
Store
View

The Dispatcher

Available through npm or Bower as **flux**

A singleton registry of callbacks, similar to EventEmitter

`dispatch(action)`: invokes all callbacks, provides action

Primary API: `dispatch()`, `register()`, **`waitFor()`**

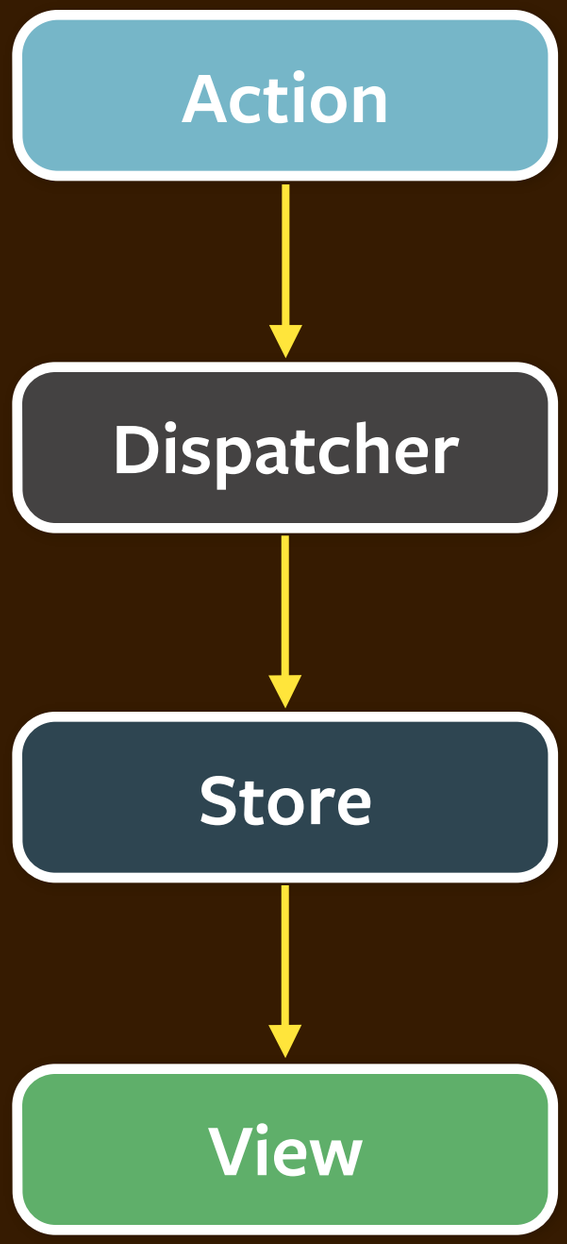
The Dispatcher

`waitFor()` is a distinctive feature, vital for derived state

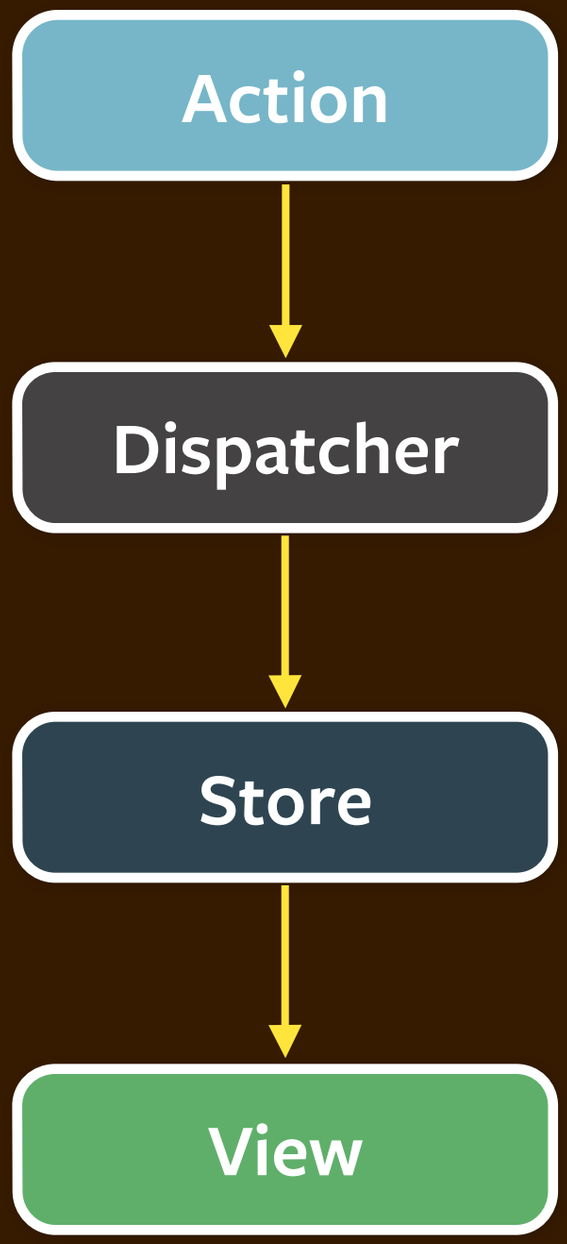
Cannot dispatch within a dispatch

Each dispatch is a discrete moment of change

```
// AppDispatcher.js

var Dispatcher = require('Flux.Dispatcher');

// export singleton
module.exports = new Dispatcher();
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```

Stores

Store

```
stores
├── MessageStore.js
└── __tests__
    └── MessageStore-test.js
```

Stores

Courtesey of The Bancroft Library, UC Berkeley BANC PIC 1905.17500 v.10:141--ALB

Stores

Each store is a singleton

The locus of control within the application

Manages application state for a logical domain

Private variables hold the application data

Numerous collections or values can be held in a store

Stores

Register a callback with the dispatcher

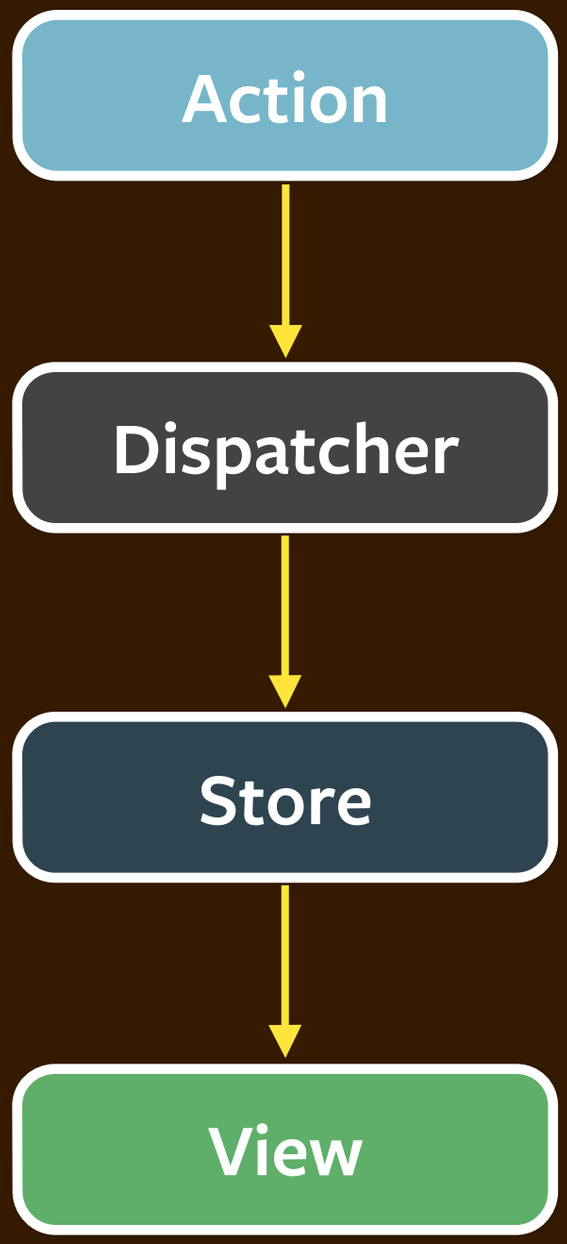
Callback is the only way data gets into the store

No setters, only getters: a universe unto itself

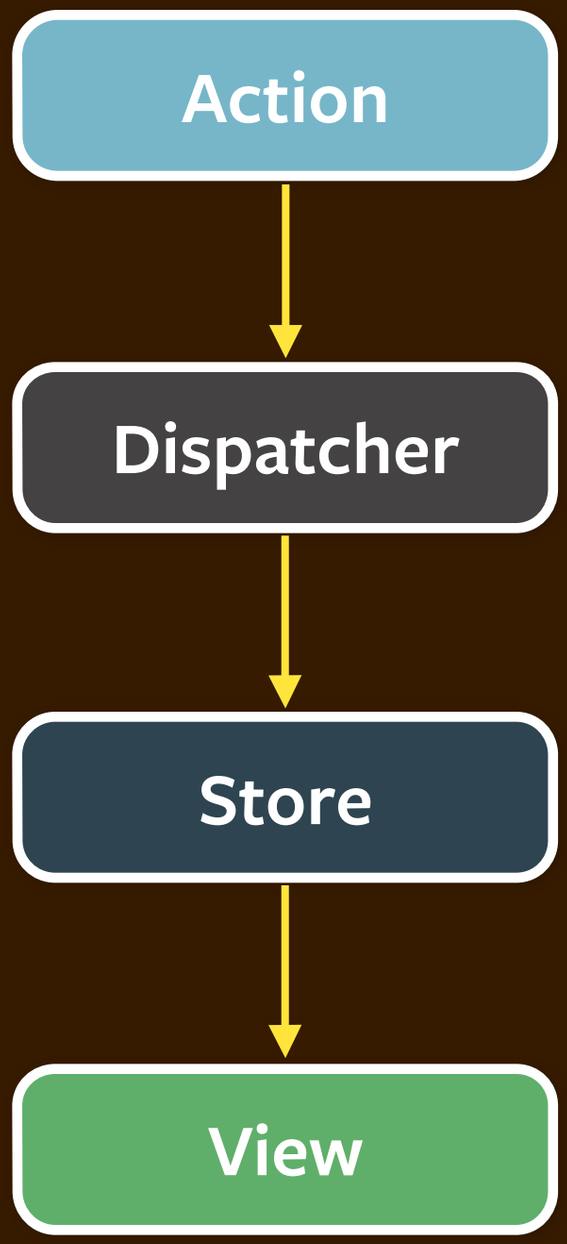
Emits an event when state has changed

```
// MessageStore.js

var _dispatchToken;
var _messages = {};

class MessageStore extends EventEmitter {

  constructor() {
    super();
    _dispatchToken = AppDispatcher.register(action => {

      switch(action.type) {

        case ActionTypes.MESSAGE_CREATED:
          var message = {
            id: Date.now(),
            text: action.text
          }
          _messages[message.id] = message;
          this.emit('change');
          break;

        case ActionTypes.MESSAGE_DELETED:
          delete _messages[action.messageID];
          this.emit('change');
          break;

        default:
          // no op
      }

    });
  }

  getDispatchToken() {
    return _dispatchToken;
  }

  getMessages() {
    return _messages;
  }

}

module.exports = new MessageStore();
```

```
_dispatchToken = AppDispatcher.register(action => {

  switch(action.type) {

    case ActionTypes.MESSAGE_CREATED:
      var message = {
        id: Date.now(),
        text: action.text
      }
      _messages[message.id] = message;
      this.emit('change');
      break;

    case ActionTypes.MESSAGE_DELETED:
      delete _messages[action.messageID];
      this.emit('change');
      break;

    default:
      // no op
  }

});
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```




```
js
├── actions
│   └── MessageActionCreators.js
├── AppBootstrap.js
├── AppConstants.js
├── AppDispatcher.js
├── stores
│   ├── MessageStore.js
│   └── __tests__
│       └── MessageStore-test.js
├── utils
│   ├── AppWebAPIUtils.js
│   ├── FooUtils.js
│   └── __tests__
│       ├── AppWebAPIUtils-test.js
│       └── FooUtils-test.js
└── views
    ├── MessageControllerView.react.js
    └── MessageListItem.react.js
```

Views & “Controller” Views

View

```
views
├── MessageControllerView.react.js
└── MessageListItem.react.js
```

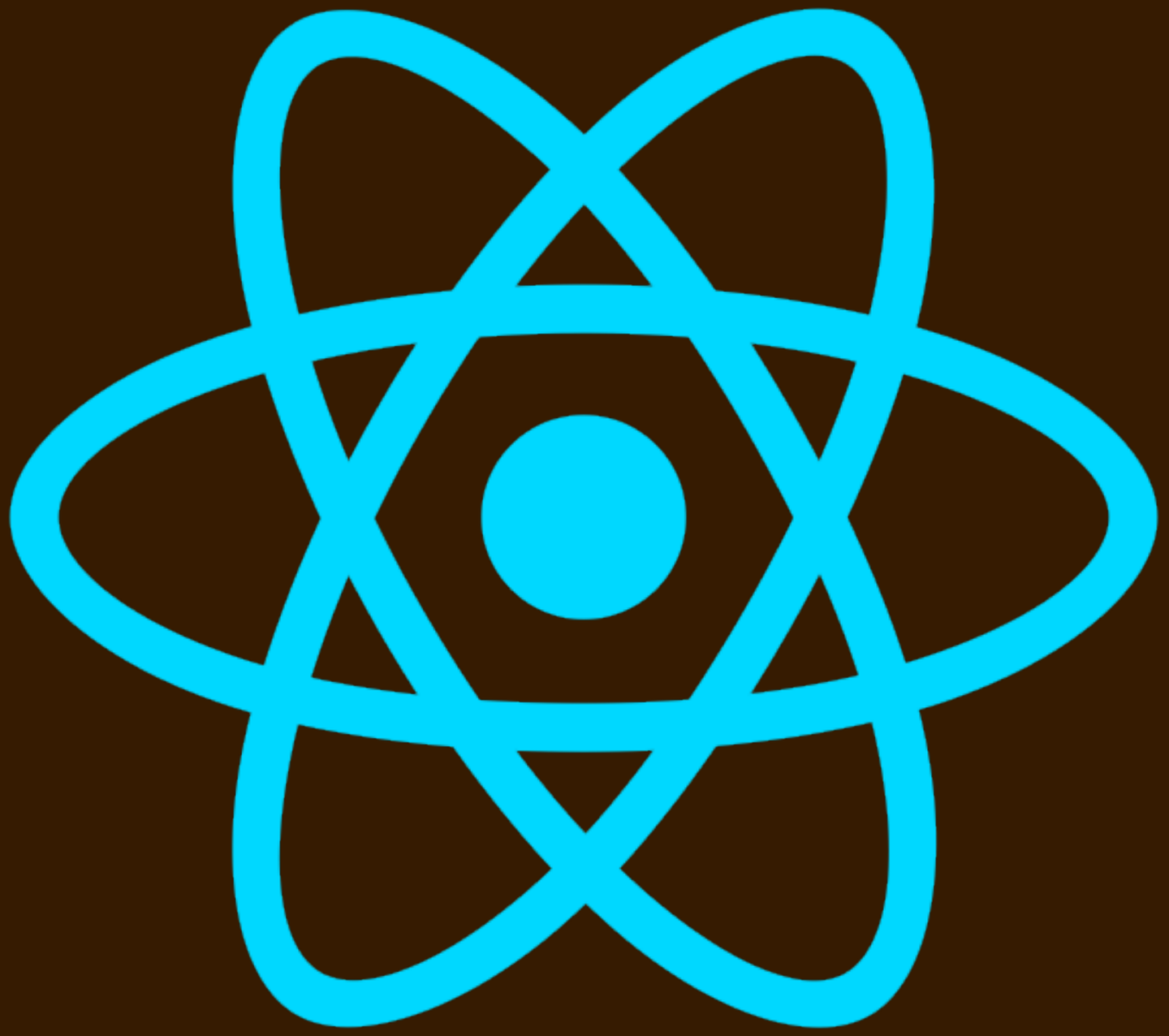
Views & “Controller” Views

Tree of React components

Controller views are near the root, listen for change events

On change, controller views query stores for new data

With new data, they re-render themselves & children



React & the DOM

Component-based framework for managing DOM updates

Uses a “Virtual DOM”: data structure and diff algorithm

Updates the DOM as efficiently as possible

Huge performance boost

Bonus: we can stop thinking about managing the DOM

React's Paradigm

Based on Functional-Reactive principles

Unidirectional data flow

Composability

Predictable, Reliable, Testable

Declarative: what the UI should look like, given props

Using React

Data is provided through props

Rendering is a function of `this.props` and `this.state`

Keep components as stateless as possible

“Re-render” (or not) on every state change or new props

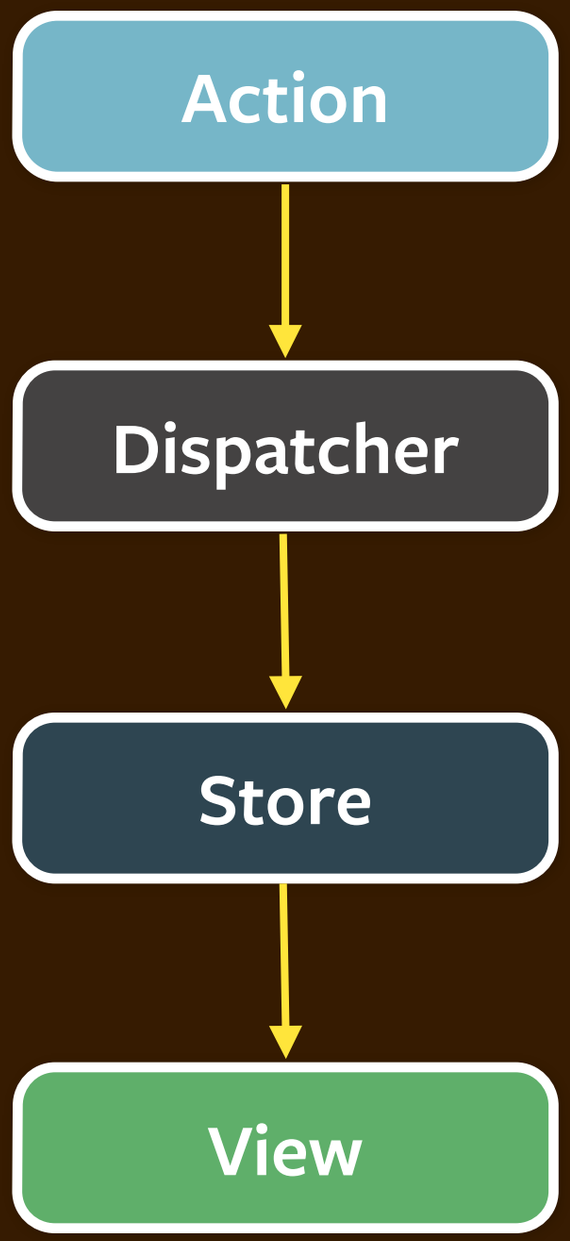
Component lifecycle and update cycle methods

```
// MessagesControllerView.react.js

var MessagesControllerView = React.createClass({

  getInitialState: function() {
    return { messages: MessageStore.getMessages() };
  },

  componentDidMount: function() {
    MessageStore.on('change', this._onChange);
  },

  componentWillUnmount: function() {
    MessageStore.removeListener('change', this._onChange);
  },

  render: function() {
    // TODO
  },

  _onChange: function() {
    this.setState({ messages: MessageStore.getMessages() });
  }

});
```

```
// MessagesControllerView.react.js

var MessagesControllerView = React.createClass({

  getInitialState: function() {
    return { messages: MessageStore.getMessages() };
  },

  componentDidMount: function() {
    MessageStore.on('change', this._onChange);
  },

  componentWillUnmount: function() {
    MessageStore.removeListener('change', this._onChange);
  },

  render: function() {
    // TODO
  },

  _onChange: function() {
    this.setState({ messages: MessageStore.getMessages() });
  }

});
```

```
render: function() {
  var messageListItems = this.state.messages.map(message => {
    return (
      <MessageListItem
        key={message.id}
        messageID={message.id}
        text={message.text}
      />
    );
  });
  return (
    <ul>
      {messageListItems}
    </ul>
  );
},
```

```
// MessageListItem.react.js

var MessageActionCreators = require('MessageActionCreators');
var React = require('react');

var MessageListItem = React.createClass({

  propTypes = {
    messageID: React.PropTypes.number.isRequired,
    text: React.PropTypes.string.isRequired
  },

  render: function() {
    return (
      <li onClick={this._onClick}>
        {this.props.text}
      </li>
    );
  },

  _onClick: function() {
    MessageActionCreators.messageDeleted(this.props.messageID);
  }

});

module.exports = MessageListItem;
```




```
messageDeleted: messageID => {
  AppDispatcher.dispatch({
    type: ActionTypes.MESSAGE_DELETED,
    messageID
  });
}
```

```
case ActionTypes.MESSAGE_DELETED:
  delete _messages[action.messageID];
  this.emit('change');
  break;
```

```
_onChange: function() {
  this.setState({
    messages: MessageStore.getMessages()
  });
}
```

Initialization of the App

Usually done in a bootstrap module

Initializes stores with an action

Renders the topmost React component

```
// AppBootstrap.js

var AppConstants = require('AppConstants');
var AppDispatcher = require('AppDispatcher');
var AppRoot = require('AppRoot.react');
var React = require('React');

require('FriendStore');
require('LoggingStore');
require('MessageStore');

module.exports = (initialData, elem) => {
  AppDispatcher.dispatch({
    type: AppConstants.ActionTypes.INITIALIZE,
    initialData
  });
  React.render(<AppRoot />, elem);
};
```

Calling a Web API

Use a WebAPIUtils module to encapsulate XHR work.

Start requests directly in the Action Creators, or in the stores.

Important: create a new action on success/error.

Data must *enter* the system through an action.

Immutable Data

Boost performance in React's `shouldComponentUpdate()`

React.addons.PureRenderMixin

immutable-js: http://facebook.github.io/immutable-js/

More Flux Patterns

LoggingStore

Error handling with client ID / dirty bit

Error handling with actions queue

Resolving dependencies in the Controller-view

Anti-patterns

Application state/logic in React components

Getters in `render()`

Public setters in stores & the setter mindset trap

Props in `getInitialState()`

Dataflow Programming

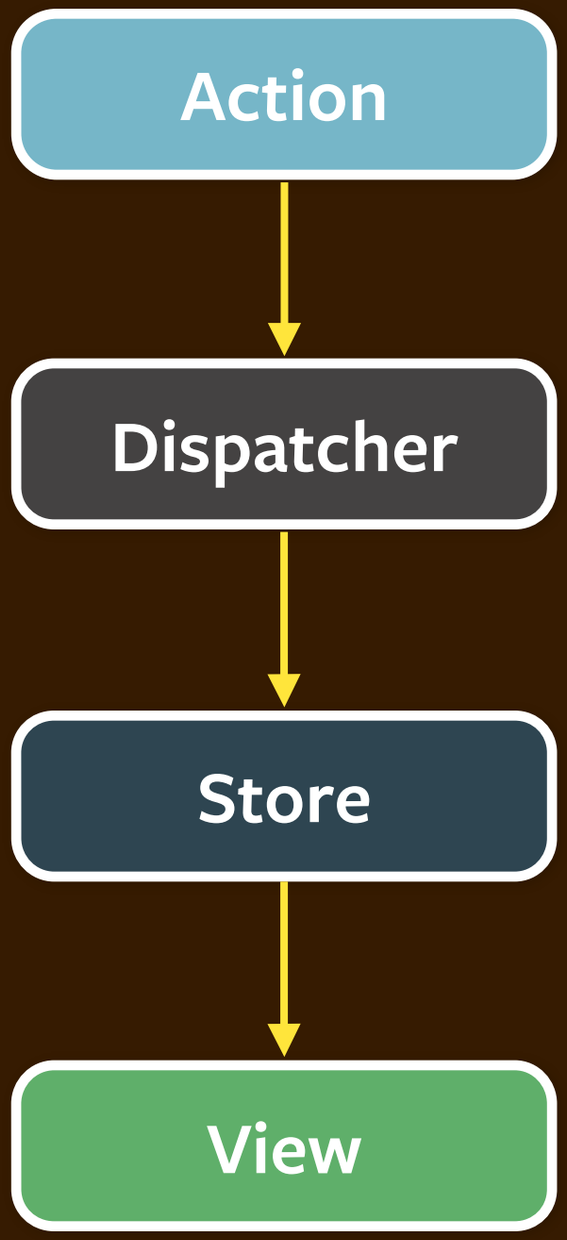
CQRS

Reactive

Flux

MVC

Functional and FRP



FLUX TOTALLY WORKS

https://speakerdeck.com/fisherwebdev/react-flux-fluent

Thank You!

https://speakerdeck.com/fisherwebdev/react-flux-fluent-2015

http://facebook.github.io/react/

http://facebook.github.io/flux/

http://facebook.github.io/jest/

http://facebook.github.io/immutable-js/

Thank You!