Redux I

le wagon

(Just) React

React Gifs

We developed an App with **React** stand alone

To make our App **interactive**, we had to:

- listen to **events** at the **component level,**
- trigger **parent callbacks** passed through **`props`,**
- **change state** at the parent's level with **`setState( )`**,
- triggering the **parent** **re-rendering**.
- The **children** received new **`props`** and **re-rendered** as well

😒

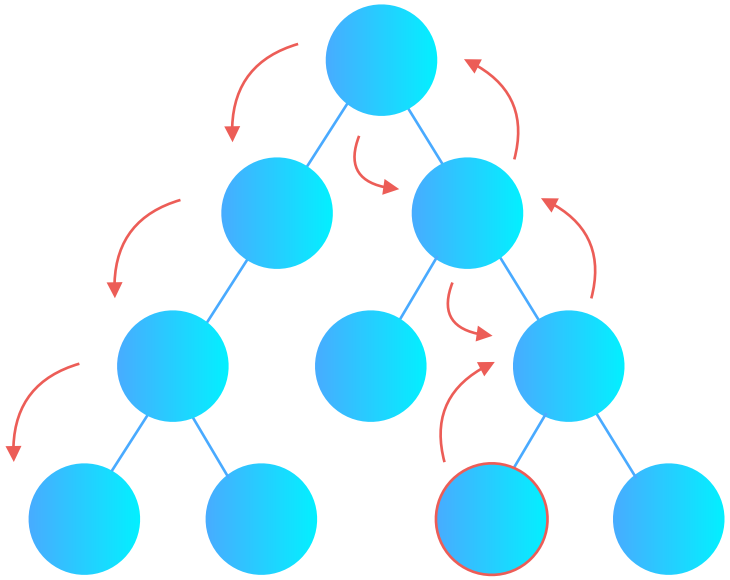
Problems

Manual plumbing. Painful.

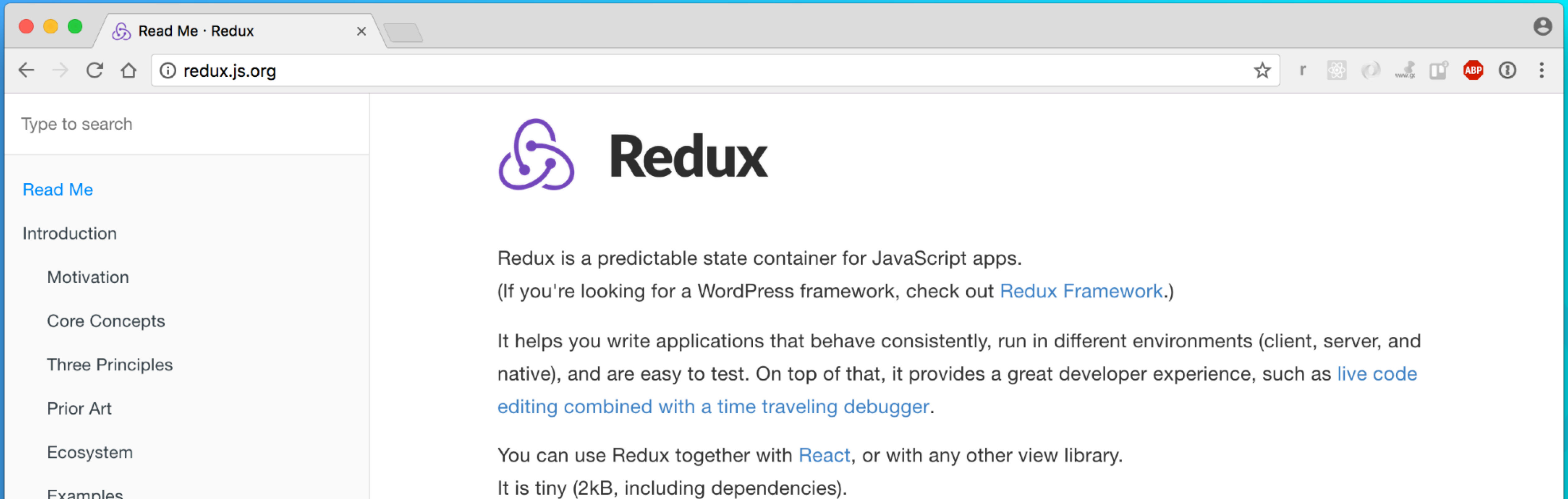
Relies on **parent / children nesting**.

Communication **through the grapevine**

Lack of coordination when apps grow



(React with) Redux



Redux is a predictable state container for JavaScript apps.
(If you're looking for a WordPress framework, check out Redux Framework.)

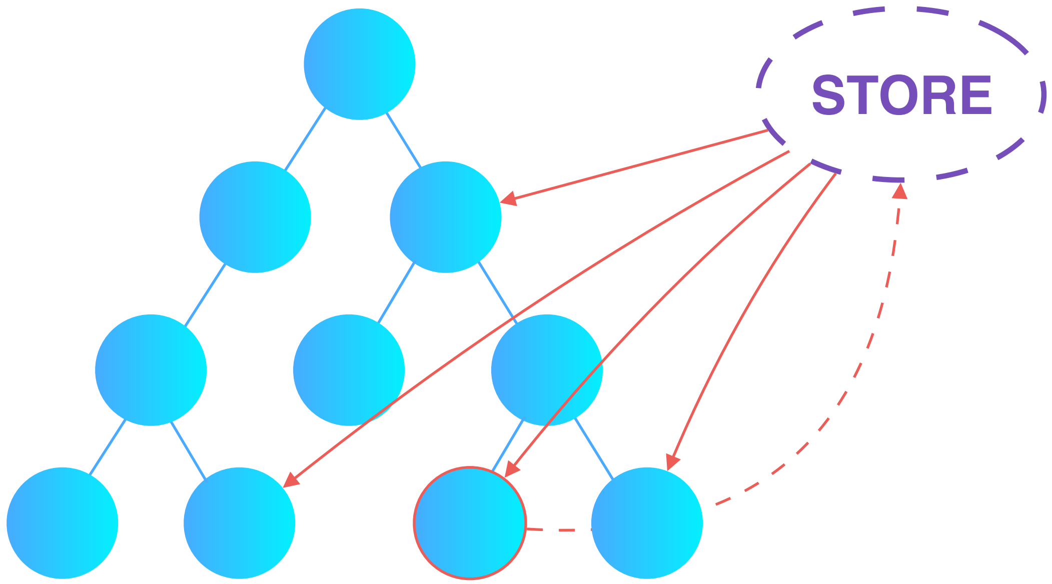
It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger.

You can use Redux together with React, or with any other view library.
It is tiny (2kB, including dependencies).

Solution

Separate the **views** (React) from the **data** (Redux)

After an event, we'll update the **data**
It will flow down through all of the components
of the App **who need to update**



React core idea

$$\text{UI} = f(\text{props}, \text{state})$$

React state

Redux core ideas

1. Single **State** Tree (hold in a **Store**)
2. **Actions** describe updates
3. **Reducers** apply updates

Redux State tree

The collection of **data** needed to describe the App **entirely** to the **current user**.

```
{
  gifs: [...],
  selectedGifId: 'xT9IgDEI1iZyb2wqo8'
}
```

⚠️ *Completely different from React component's local notion of* ***state***

Action

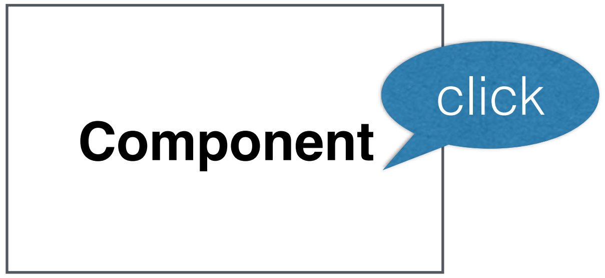
```
(arg) => {
  // Handling action, computing payload.
  return {
    type: 'ACTION_TYPE',
    payload: payload
  }
}
```

An action creator (function) returns an ***action (object)***

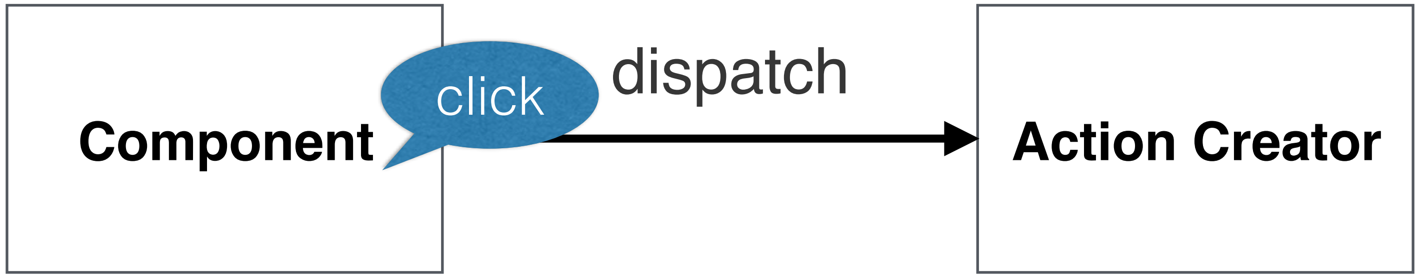
Reducer

```
(previousState, action) => {
  // Computing new state from given action
  return newState;
}
```

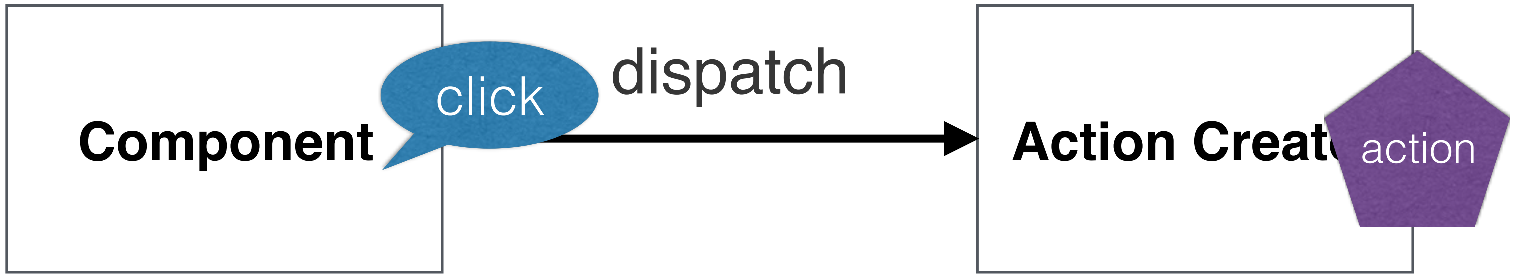
Flow



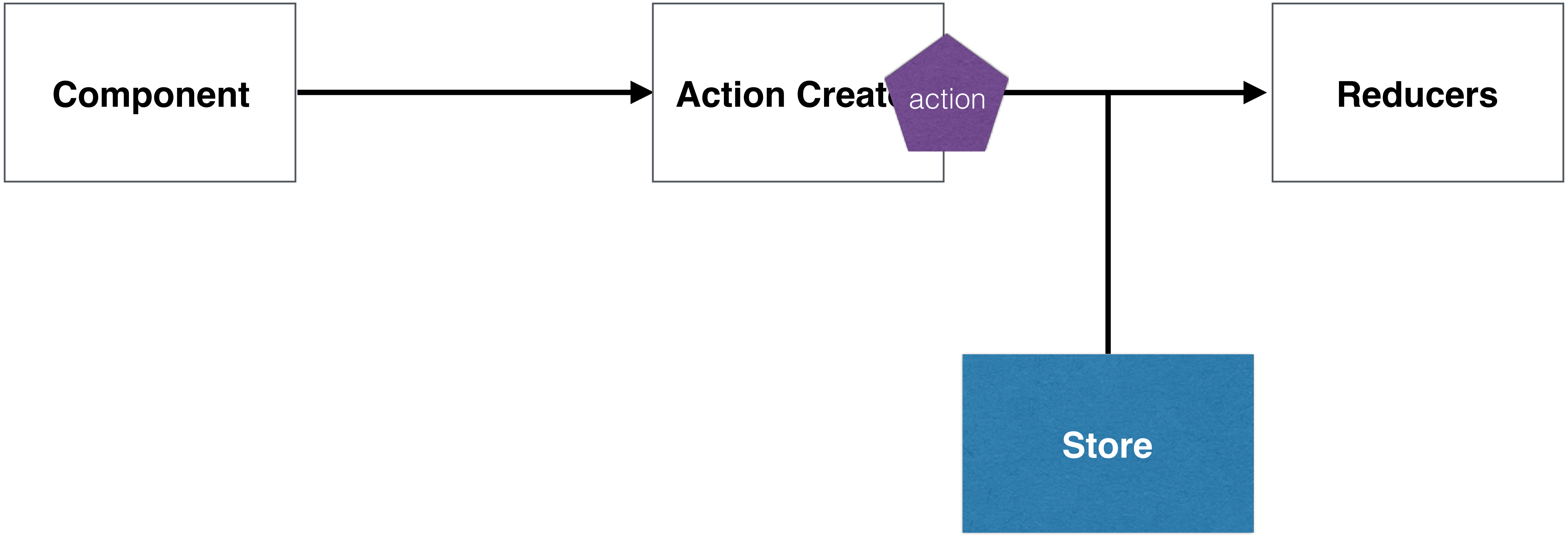
Flow



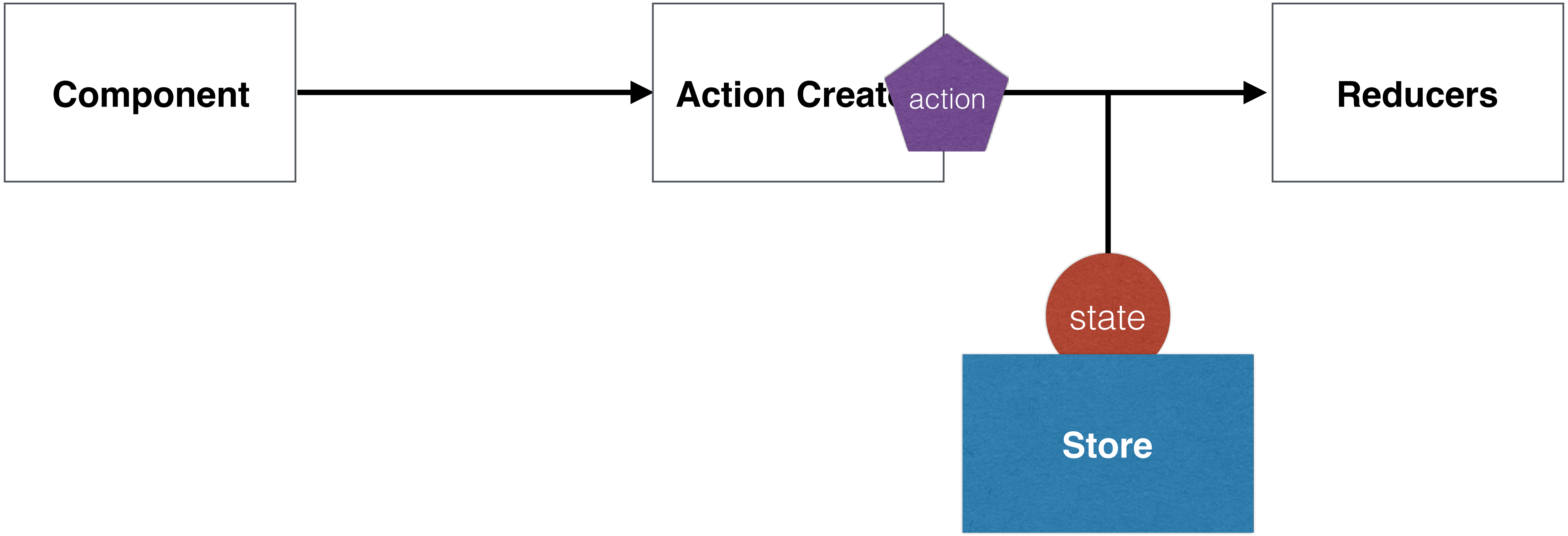
Flow



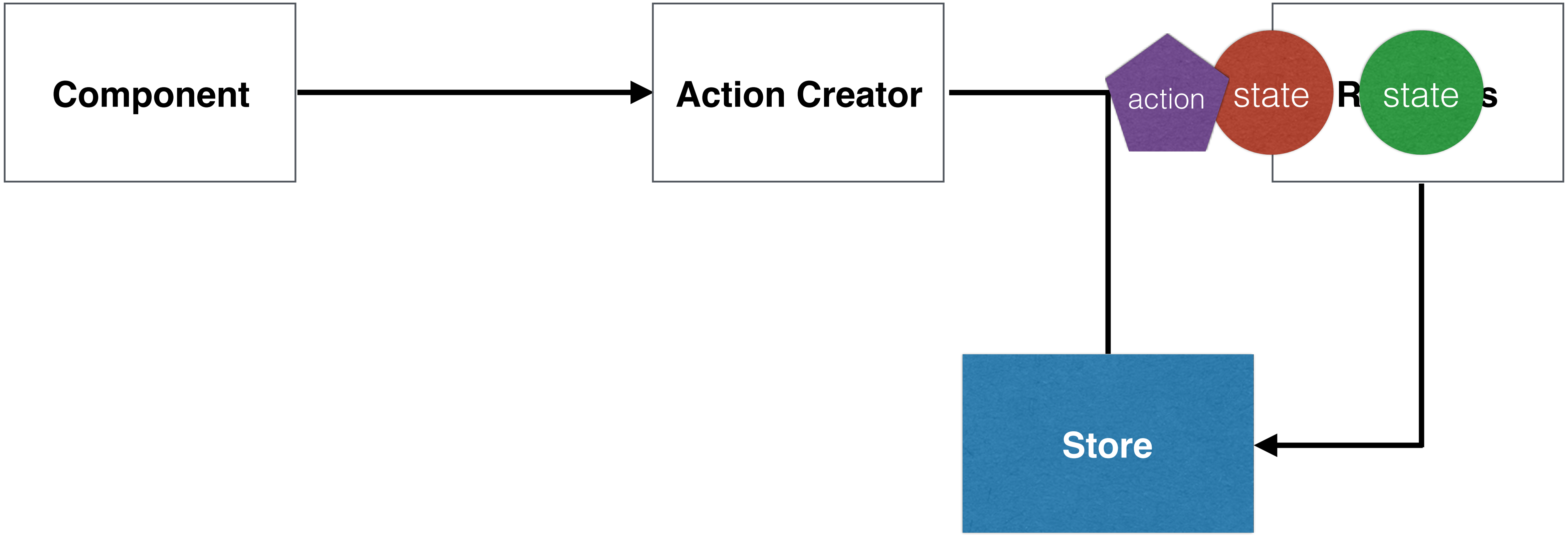
Flow



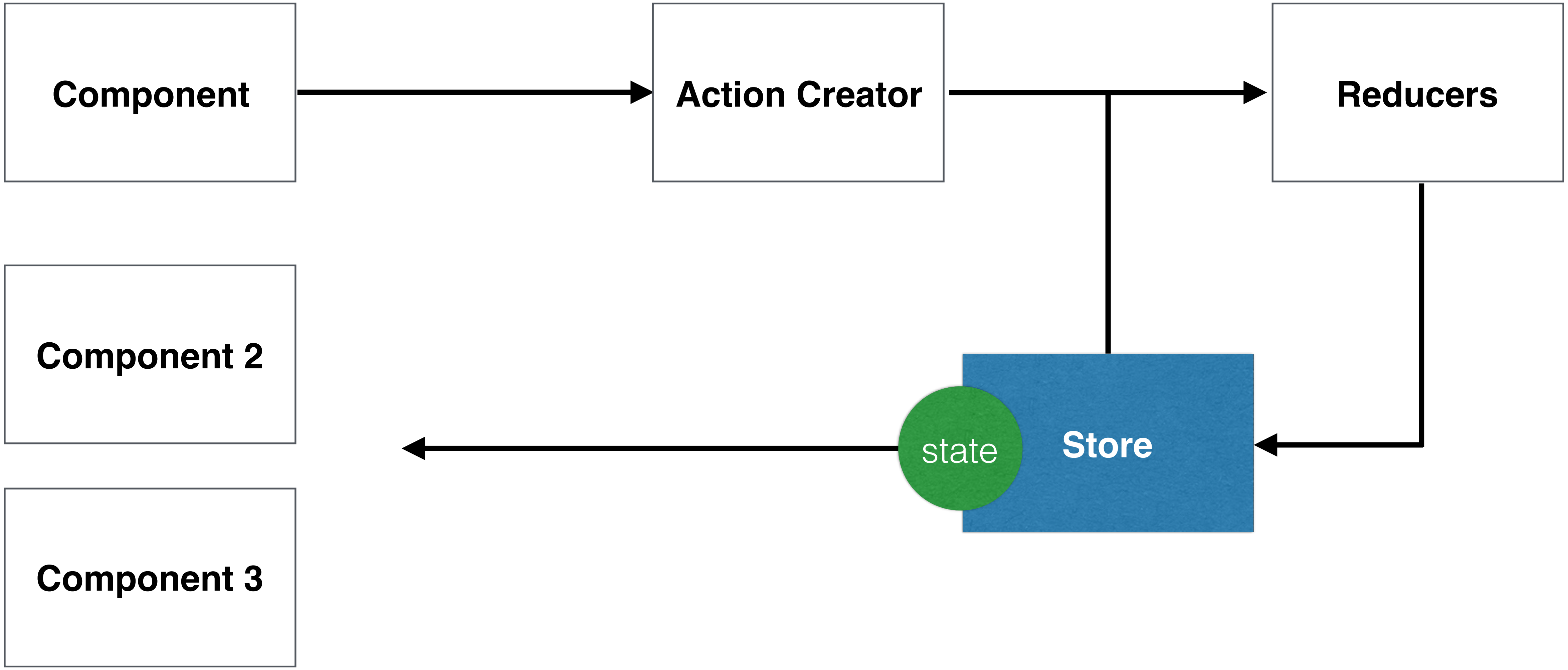
Flow



Flow



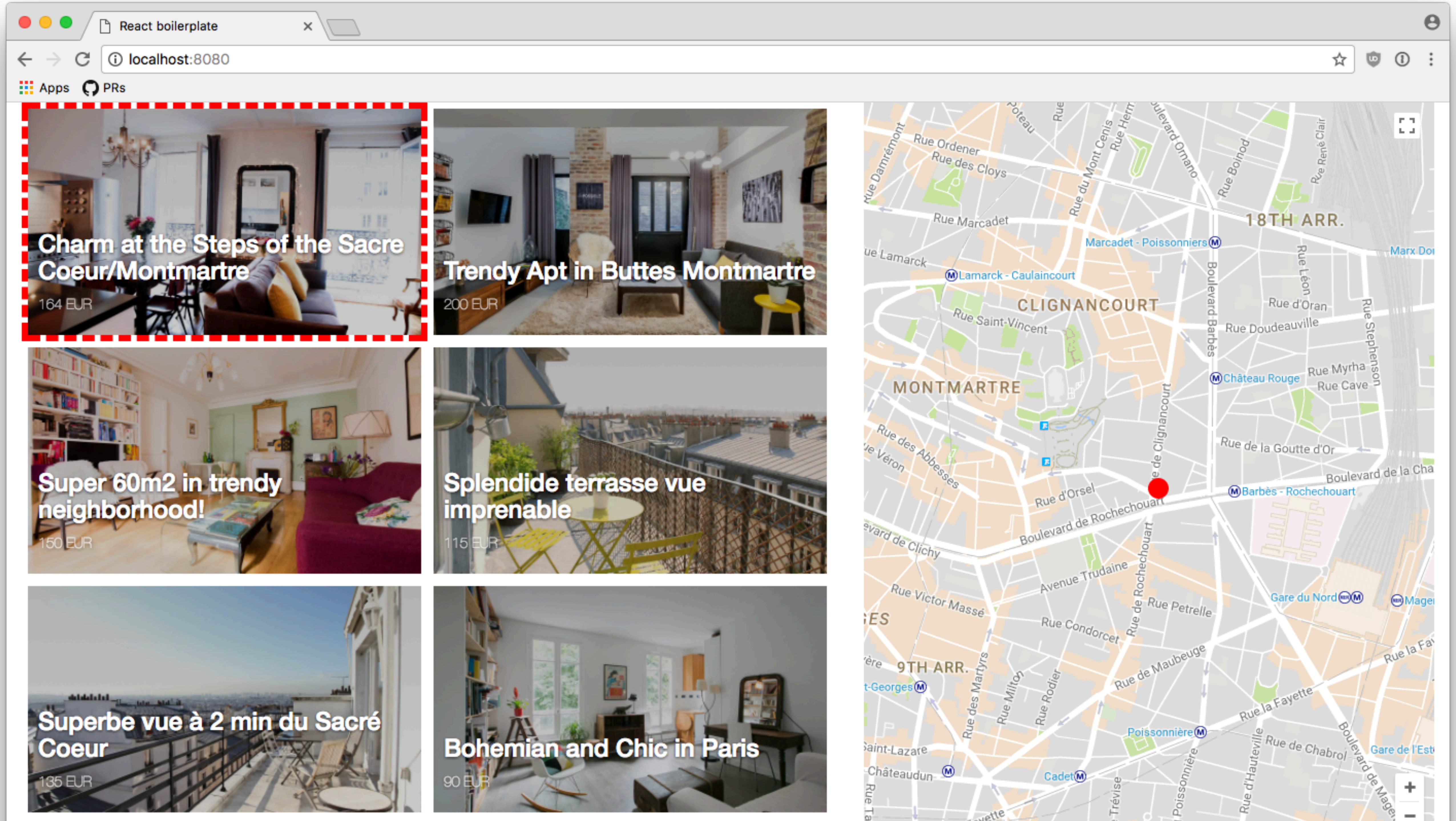
Flow


Component
Action Creator
Reducers
Component 2
state
Store
Component 3

Livecode

Airbnb Search

Components?



React boilerplate

localhost:8080

Apps PRs

Charm at the Steps of the Sacre Coeur/Montmartre
164 EUR

Trendy Apt in Buttes Montmartre
200 EUR

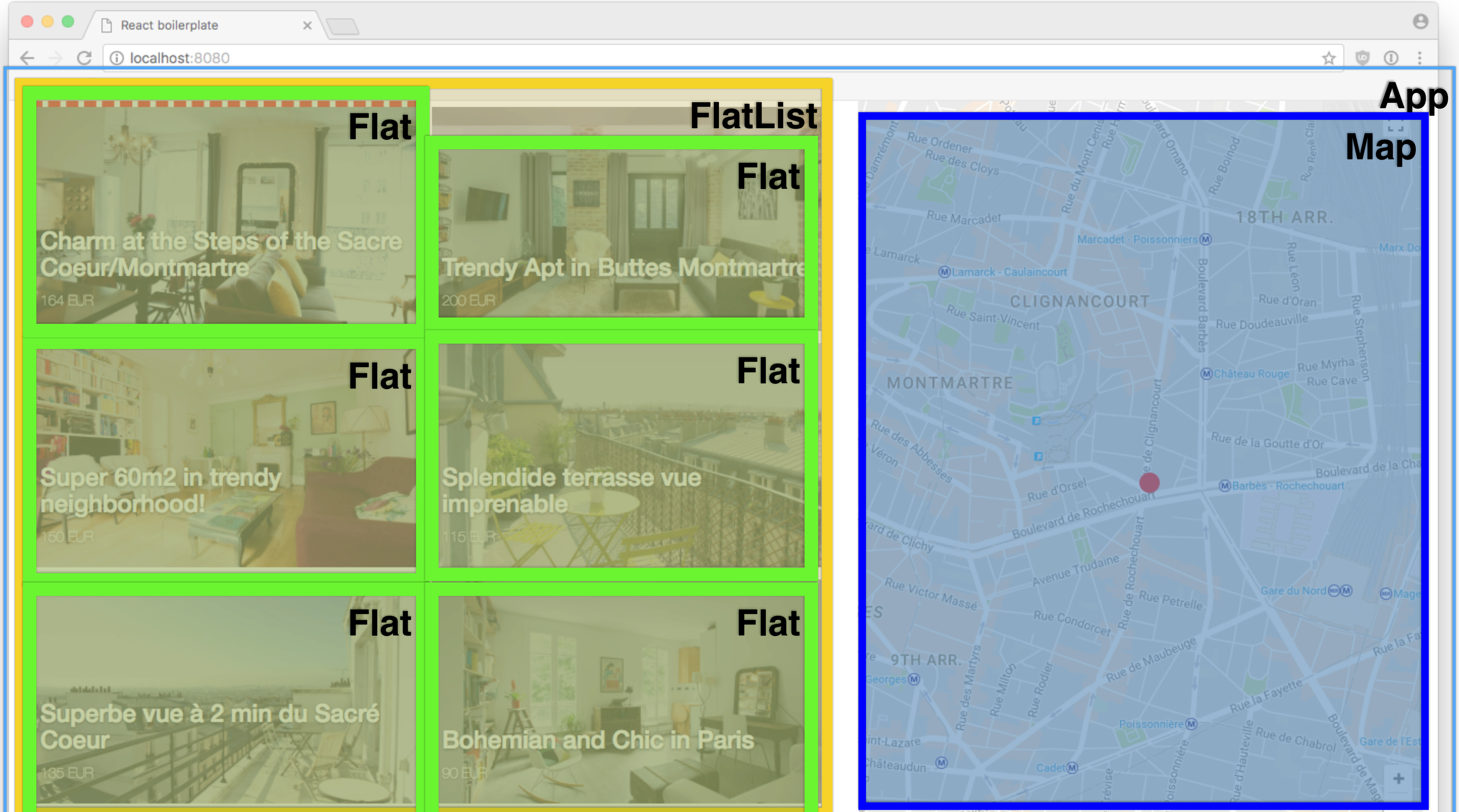
Super 60m2 in trendy neighborhood!
150 EUR

Splendide terrasse vue imprenable
115 EUR

Superbe vue à 2 min du Sacré Coeur
135 EUR

Bohemian and Chic in Paris
90 EUR

Components



Charm at the Steps of the Sacre Coeur/Montmartre
164 EUR

Trendy Apt in Buttes Montmartre
200 EUR

Super 60m2 in trendy neighborhood!
160 EUR

Splendide terrasse vue imprenable
115 EUR

Superbe vue à 2 min du Sacré Coeur
135 EUR

Bohemian and Chic in Paris
90 EUR

Redux state tree?

```
{
  flats: [ ... ],
  selectedFlat: { ... }
}
```

Setup

Starting from https://github.com/lewagon/react-boilerplate

```
git clone git@github.com:lewagon/react-boilerplate.git static-airbnb-redux
cd static-airbnb-redux

rm -rf .git
git init
git add . && git commit -m  "initial commit"

yarn install
yarn add redux react-redux
```

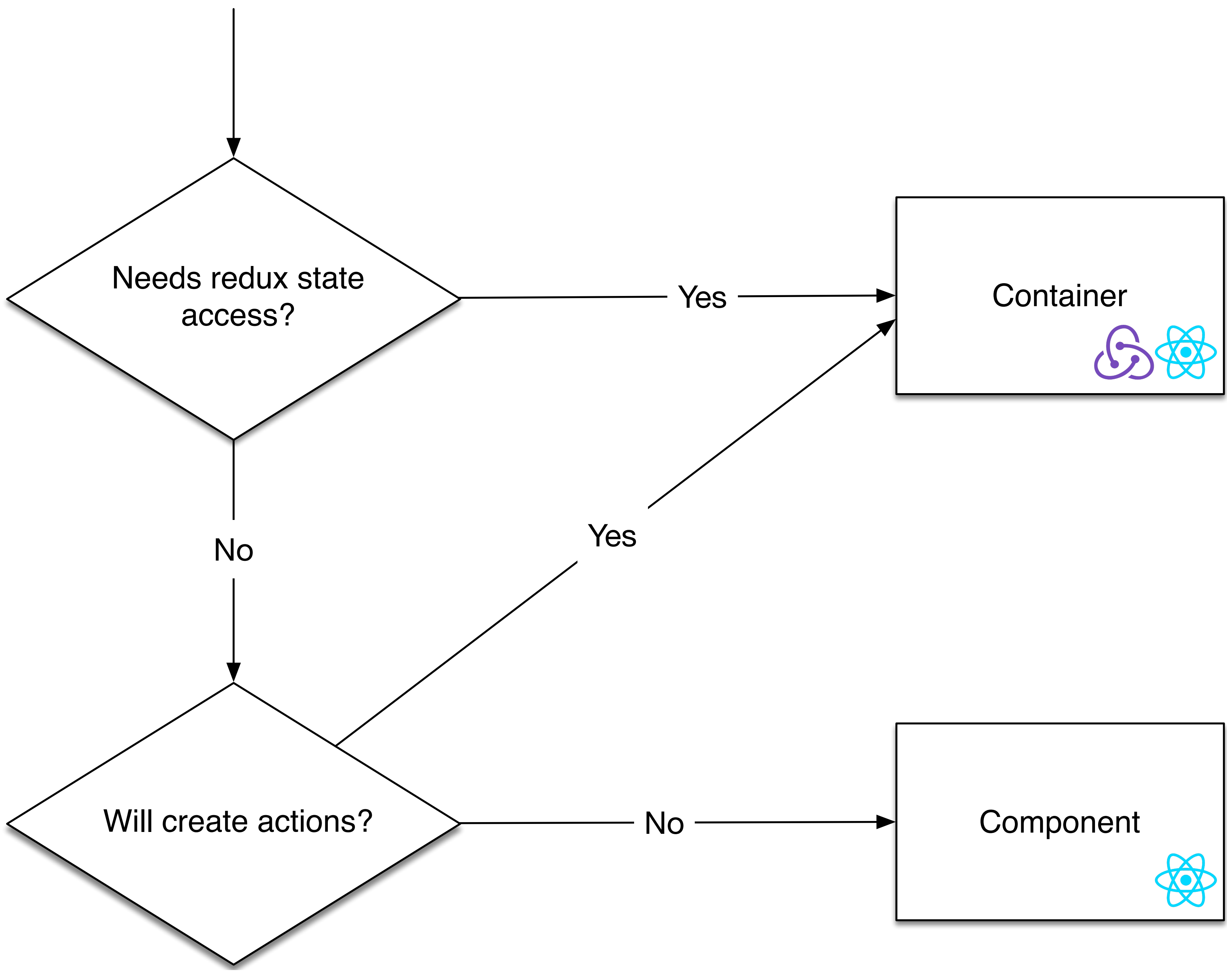
Redux setup

```
mkdir src/actions
mkdir src/reducers
mkdir src/containers
```

Containers vs Components

Container

Connecting **data** (Redux) and **views** (React).
A **container** is a **smart** component
Has **direct access** to one or several pieces of the **Redux state**



Needs redux state access?
Yes
Container
No
Yes
Will create actions?
No
Component

App component

```
import React from 'react';
import FlatList from '../containers/flat_list';
import Map from '../containers/map';

const App = () => {
  return (
    <div className="row">
      <FlatList />
      <Map />
    </div>
  );
};

export default App;
```

Map container

```
import React, { Component } from 'react';

class Map extends Component {
  render() {
    return (
      <div className="col-sm-5" style={{height: '100vh'}}>
        TODO Map
      </div>
    );
  }
}

export default Map;
```

FlatList container

```
import React, { Component } from 'react';

class FlatList extends Component {
  render( ) {
    return (
      <div className="flat-list col-sm-7">
        TODO Flat List
      </div>
    );
  }
}

export default FlatList;
```

```
.flat-list {
  display: flex;
  flex-wrap: wrap;
}
```

Default props (temp)

```
class FlatList extends Component {
  // TEMPORARY CODE TO INTEGRATE HTML
  static defaultProps = {
    flats: [{
      "name": "Charm at the Steps of Montmartre",
      "imageUrl": "https://raw.githubusercontent.com/
lewagon/flats-boilerplate/master/images/flat1.jpg",
      "price": 164,
      "priceCurrency": "EUR"
    }]
  }
}
```

FlatList / Flat

```
import Flat from '../components/flat';

class FlatList extends Component {
  render() {
    return (
     <div className="flat-list col-sm-7">
       {this.props.flats.map((flat) => {
         return <Flat key={flat.name} flat={flat} />;
       })}
     </div>
    );
  }
}
```

Flat component

```
import React from 'react';

const Flat = (props) => {
  const style = {
    backgroundImage: `url(${props.flat.imageUrl})`
  };
  return (
    <div className="flat card-container">
      <div className="card" style={style}>
        [...]
      </div>
    </div>
  );
};

export default Flat;
```

```
.flat {
  flex: 50% 0 0;
  cursor: pointer;
  border: 6px solid transparent;
}
```

Fetch HTML+CSS from lewagon.github.io/ui-components/#cards

FlatList container

```
class FlatList extends Component {

  componentWillMount() {
    // TODO: dispatch an action to load flats!
  }

  // [...]
}
```

State & Reducers

Bootstraping Redux app

```
// src/index.js
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flats_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

Provider

```
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flat_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

Store

```
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flat_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

Reducers combination

```
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flat_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

Redux state tree

```
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flat_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

Flats reducer

```
// [...]

import { Provider } from 'react-redux';
import { createStore, combineReducers } from 'redux';

import flatsReducer from './reducers/flat_reducer';

const reducers = combineReducers({
  flats: flatsReducer
});

ReactDOM.render(
  <Provider store={createStore(reducers)}>
    <App />
  </Provider>,
  document.getElementById('root'));
```

💡 *next step: create flats_reducer.js*

Flat reducer

```
const flatsReducer = (state, action) => {
  if (state === undefined) {
    // Reducer initialisation
    return [];
  }

  // TODO: handle some actions
};

export default flatsReducer;
```

Action creators

Action creator

A **function** that returns an object with a payload

```
// src/actions/index.js

import flats from '../flats';

export function setFlats() {
  // TODO: Api call! For now, simulate a DB

  return {
    type: 'SET_FLATS',
    payload: flats
  }
}
```

💡 *flats is an array copied from github.com/lewagon/flats-boilerplate*

Reducer

```
// reducers/flat_reducer.js

export default function(state, action) {
  if (state === undefined) {
    return [];
  }

  switch (action.type) {
    case 'SET_FLATS':
      return action.payload;
    default:
      return state;
  }
}
```

Redux magic

mapDispatchToProps()

```
// src/containers/flat_list.jsx
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import { setFlats } from '../actions';

// [...]
function mapDispatchToProps(dispatch) {
  return bindActionCreators(
    { setFlats: setFlats },
    dispatch
  );
}

export default connect(null, mapDispatchToProps)(FlatList);
```

💡 *this.props.setFlats is now available in the container*

FlatList container

```
// [...]

class FlatList extends Component {

  componentWillMount() {
    this.props.setFlats();
  }

  // [...]
}

// [...]
```

mapStateToProps()

```
// src/containers/flat_list.jsx

// [...]

function mapStateToProps(state) {
  return {
    flats: state.flats
  };
}

export default connect(mapStateToProps, mapDispatchToProps)
(FlatList);
```

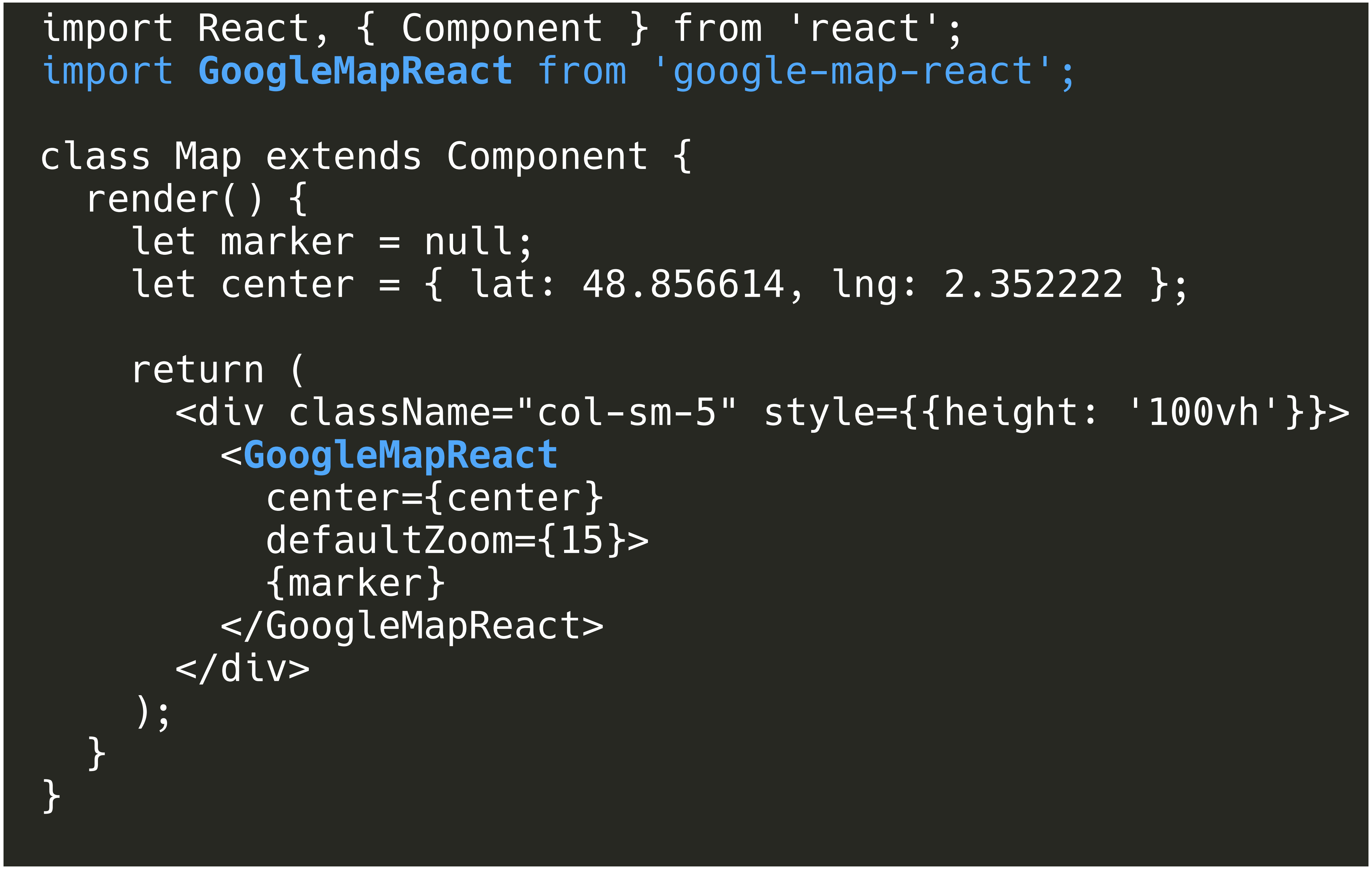
💡 *this.props.flats is now mapped to the redux state subtree "flats"*

Map

Map container

```
yarn add google-map-react
```

Map container



```
import React, { Component } from 'react';
import GoogleMapReact from 'google-map-react';

class Map extends Component {
  render( ) {
    let marker = null;
    let center = { lat: 48.856614, lng: 2.352222 };

    return (
      <div className="col-sm-5" style={{height: '100vh'}}>
        <GoogleMapReact
          center={center}
          defaultZoom={15}>
          {marker}
        </GoogleMapReact>
      </div>
    );
  }
}
```

Select a flat

Component => Container

We need to promote Flat to a **Container**

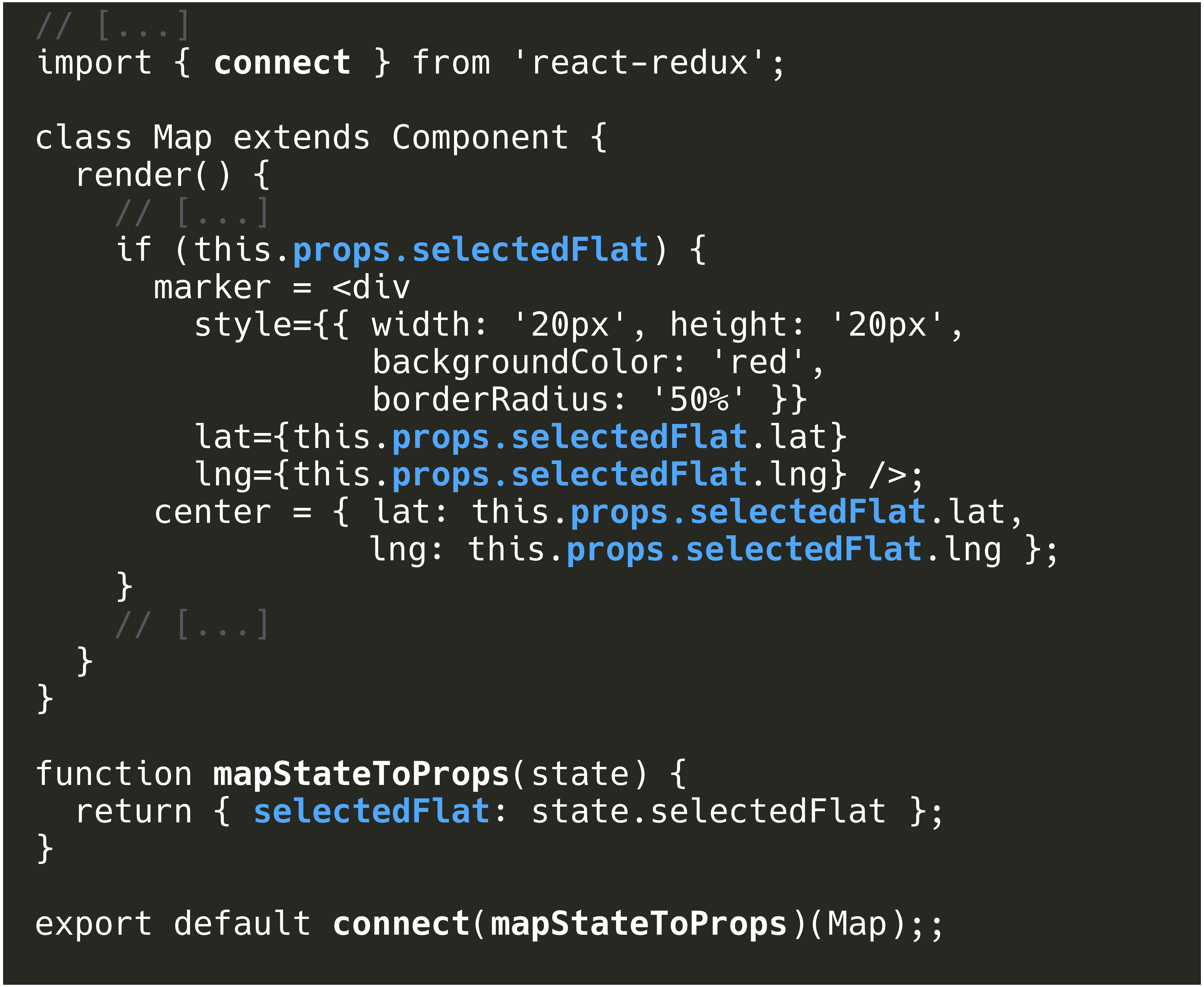
Action, State & Reducer

1. We need to define a new action **`selectFlat`**
2. Redux state tree needs a new key: **`selectedFlat`**
3. This new key needs a reducer: **`selectedFlatReducer`**

UI tweak: add a **`.selected`** class to Flat

```
.selected {
  border: 6px dashed red;
}
```

```
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';

import { selectFlat } from '../actions';

class Flat extends Component {
  // [...]
}

function mapStateToProps(state) {
  return {
    selectedFlat: state.selectedFlat
  };
}

function mapDispatchToProps(dispatch) {
  return bindActionCreators(
    { selectFlat: selectFlat }, dispatch);
}

export default connect(
 mapStateToProps, mapDispatchToProps)(Flat);
```



```
// [...]
import { connect } from 'react-redux';

class Map extends Component {
  render() {
    // [...]
    if (this.props.selectedFlat) {
      marker = <div
        style={{ width: '20px', height: '20px',
                 backgroundColor: 'red',
                 borderRadius: '50%' }}
        lat={this.props.selectedFlat.lat}
        lng={this.props.selectedFlat.lng} />;
      center = { lat: this.props.selectedFlat.lat,
                 lng: this.props.selectedFlat.lng };
    }
    // [...]
  }
}

function mapStateToProps(state) {
  return { selectedFlat: state.selectedFlat };
}

export default connect(mapStateToProps)(Map);;
```

Conclusion

Good to know

Consider the **Redux state immutable**.

A reducer should always return:
the unchanged Redux state or an entirely new object.

```
// src/reducers/*.js

export default function(state = null, action) {
  ~~state.activeCity = action.payload;~~ 😱⛔☠️
  return state;
}
```

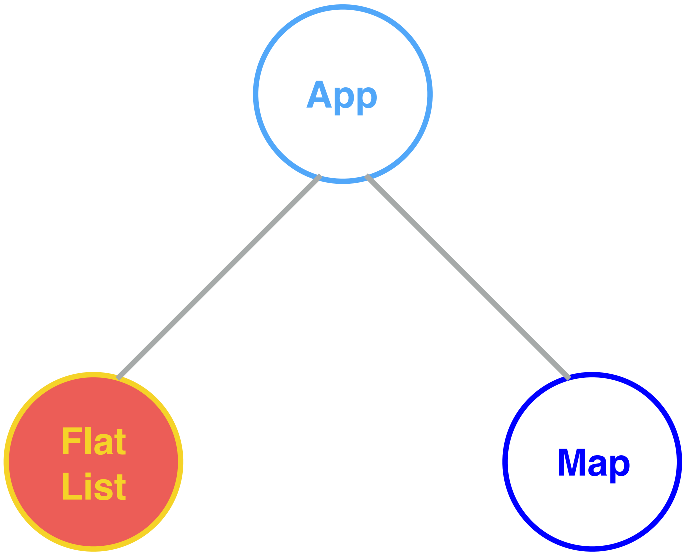
💡 *Create copies of state with Object.assign*

Wrap-up

To make an **interactive** web page with
React + **Redux**...

Wrap-up

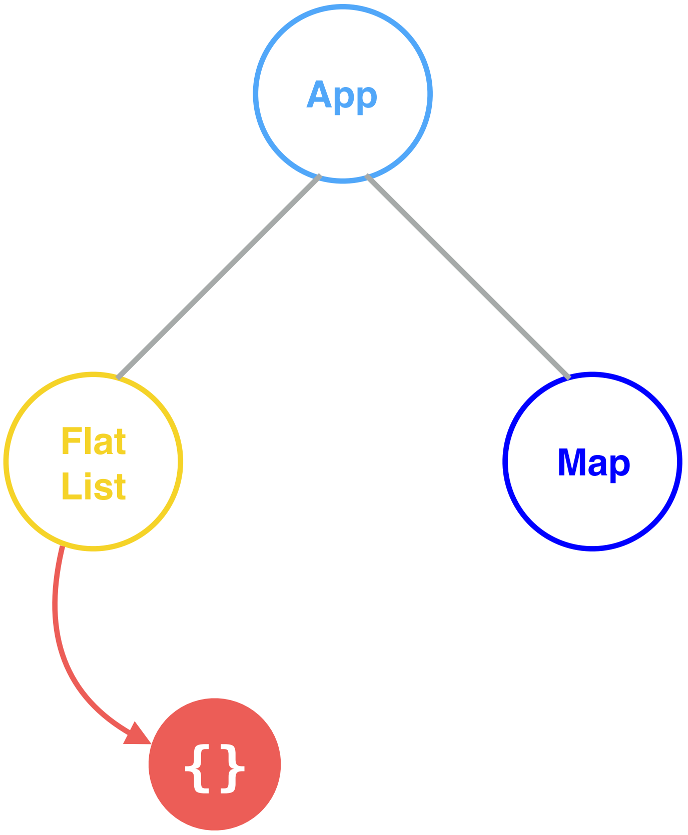
we'll handle **events** with an **action creator** as **callback**



```
onClick={() => this.props.selectFlat(flat)}
```

Wrap-up

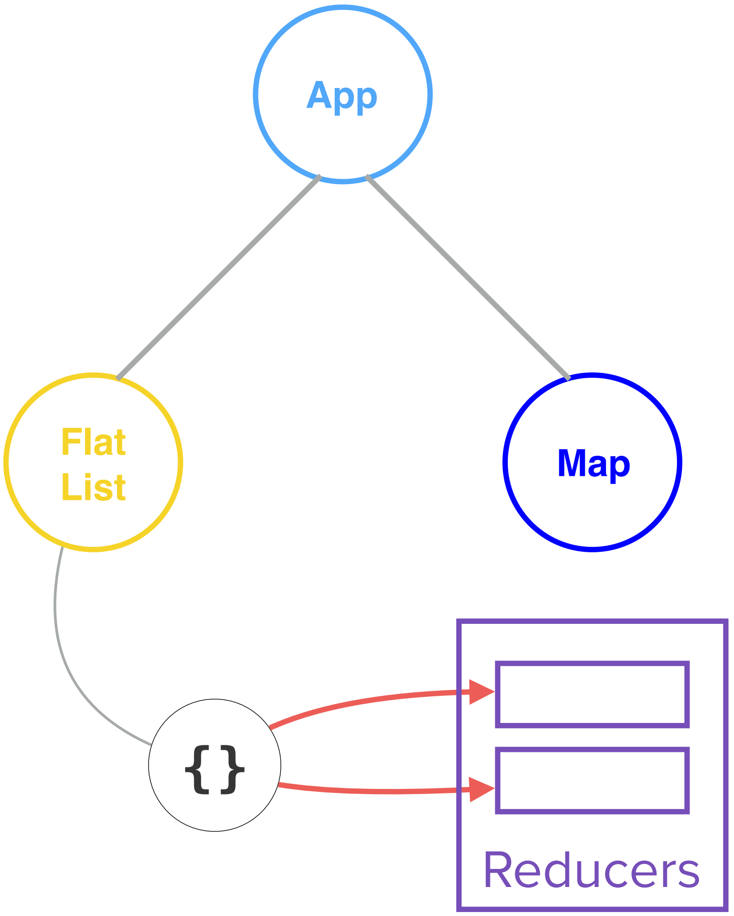
the **action creator** returns
an **action** with a type/ payload



```
return { type: 'SELECT_FLAT', payload: … }
```

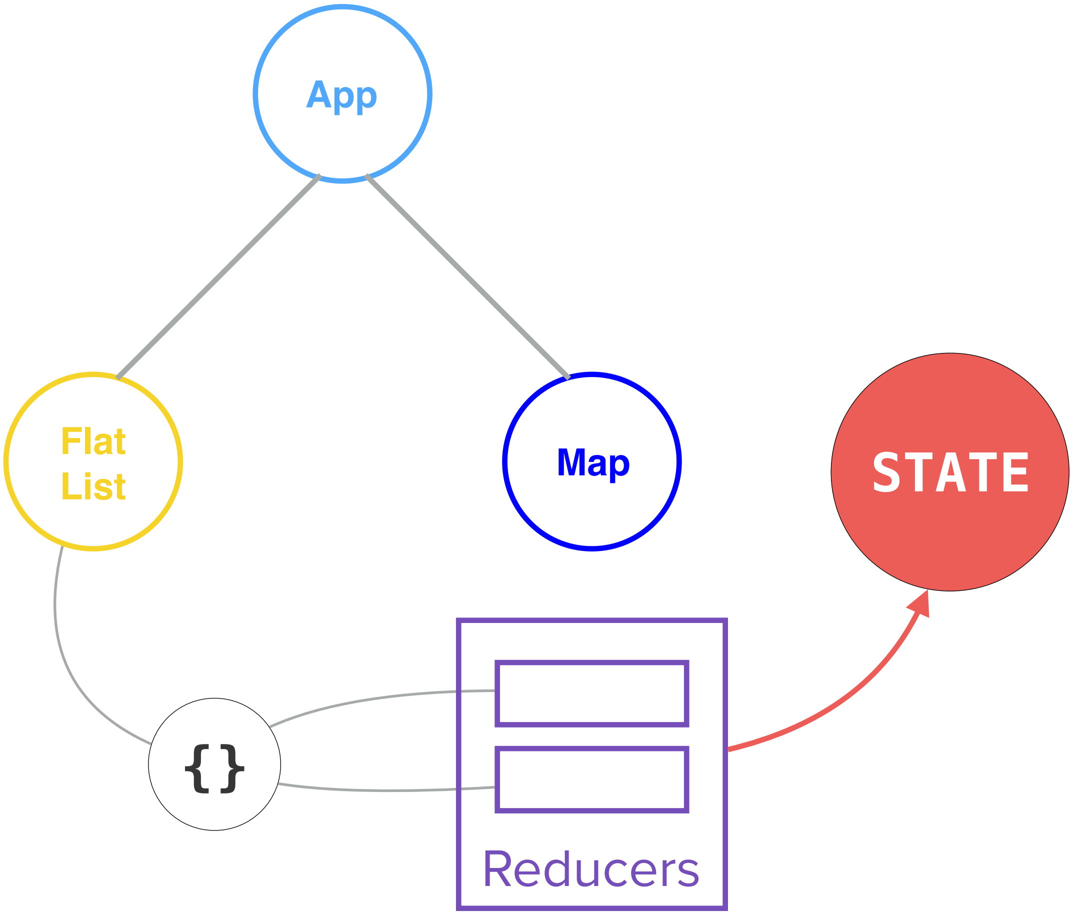
Wrap-up

the action **flows** through
all of the **reducers**



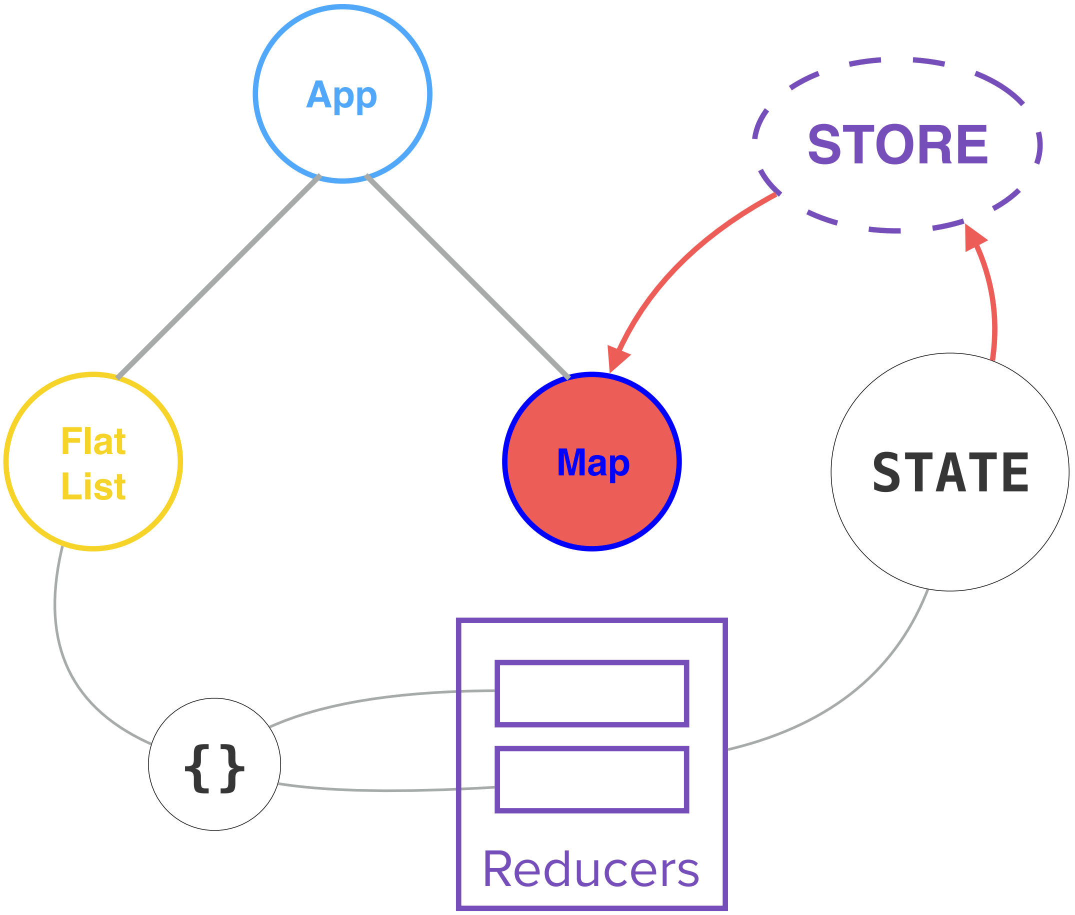
Wrap-up

creating the up-to-date
Redux state



Wrap-up

Re-rendering only the containers who **need to**



Take away

Redux State is the single source of truth for **data**

Get the code

Livecode available at /lewagon/static-airbnb-redux

Your turn!

Boilerplate

Start from https://github.com/lewagon/redux-boilerplate