

REACTATHON

ES6

ES6

ES6 (ECMASCRIPT 2015)



- 6th edition to the ECMAScript standard released in 2015
- First update since ES5 released in 2009
- Introduced some major syntax and features such as classes, modules, arrow functions, promises...
- ES7 — Finalized in 2016 (Array.prototype.includes(), ** exponent operator)
- ES8 — Finalized June 2017 (async/await)
- ES.Next

CONST / LET / VAR

ES6

- **var** — function scoped variable that can be reassigned (pre ES6)
- **let** — like var but block scoped (scoped to the nearest {}) rather than function scoped

```
for(var i = 0; i<10; i++) {  
    console.log(i)  
}  
console.log(i)
```

10

```
for(let i = 0; i<10; i++) {  
    console.log(i)  
}  
console.log(i)
```

I is not defined!

CONST / LET / VAR

ES6

- **const** — blocked scoped (like **let**) variable that cannot be reassigned
- However you can reassign properties

```
// Foo cannot be reassigned
const foo = 213

const bar = {
  name: 'Bar',
  color: 'Green'
}

// Bar.name can be reassigned
bar.name = 'Blue'
```

- ▶ Use **const** when you can, but if you must reassign use **let**

DECONSTRUCTION

ES6

- Unpack values from arrays or properties from objects into variables
- Object Deconstruction

```
const person = {  
  name: 'Bob',  
  age: 32,  
  occupation: 'Dancer'  
}
```

```
const {name, age, occupation} = person
```

```
const person = {  
  name: 'Bob',  
  age: 32,  
  occupation: 'Dancer'  
}
```

```
const {name: firstName, age, occupation: jobTitle} = person
```

- Array Deconstruction

```
const iterable = ['a', 'b'];  
const [x, y] = iterable;
```

x = 'a' and y = 'b'

Renaming the deconstructed args

ARROW FUNCTIONS

ES6

- Less verbose syntax for function definitions

```
const multiply = function (x,y)  
{  
    return x * y  
}
```

As Arrow Function

```
const multiply = (x, y) => { return x * y }
```

Parameter Syntax

```
() => { /*...*/ } // no parameter  
x => { /*...*/ } // one parameter, an identifier  
(x, y) => { /*...*/ } // several parameters
```

Body Syntax

```
x => { return x * x } // block  
x => x * x           // expression
```

No brackets will return result automatically

- No rebinding of this to reduce usage of *bind*

CLASSES

ES6

- More syntactic sugar over Prototypical Inheritance
- Allows inheritance with the `extends` keyword
- Some Features:
 - * Class Methods
 - * Static Methods
 - * Constructor
 - * Gettings/Setters

```
class Rectangle {  
    constructor(height, width) {  
        this.height = height;  
        this.width = width;  
    }  
  
    // Method  
    calcArea() {  
        return this.height * this.width;  
    }  
  
}  
  
const square = new Rectangle(10, 10);  
console.log(square.calcArea()) // 100
```

PROMISES

ES6

- A Promise is an object representing *the eventual completion or failure of an asynchronous operation*
- Promise Construction — constructor takes 2 call backs
 - **resolve** — get called upon successful execution completion
 - **reject** — gets called upon error during execution

resolve called with result on success

```
function asyncFunc() {
  return new Promise(
    function (resolve, reject) {
      try {
        const result = logicToGetResult()
        resolve(result);
      } catch (error) {
        reject(error);
      }
    });
}
```

reject called on error

PROMISES

ES6

- Using Promise

- ***then*** — get executed if the Promises invoked the resolve callback with any returned value
- ***catch*** — gets executed if the Promises invoked the reject

then called with result on success

catch called on error

```
asyncFunc()
  .then(result => {
    console.log(result)
  })
  .catch(error => {
    // Handle errors
  });
```

PROMISES

ES6

► Chaining Multiple Promises:

```
asyncFunctionOne(a, b)
  .then(result1 => {
    console.log(result1);
    return asyncFunctionTwo(result1);
  })
  .then(result2 => {
    console.log(result2);
  })
  .catch(error => {
    // Handle errors of asyncFuncOne() and asyncFuncTwo()
  })
```

Second `.then` called when `asyncFunctionTwo` resolves

Catches errors from either first or second promise

PROMISES - ASYNC/AWAIT

ES6

- Build upon Promises to help further synchronize asynchronous code.

Promise

```
const makeRequest = () =>
 getJSON()
  .then(data => {
    console.log(data)
    return "done"
  })
```

Async/Await

With async/await

```
const makeRequest = async () => {
  console.log(await getJSON())
  return "done"
}
```

await must be within an **async** function

await pauses function until promise is resolved

async functions actually return another promise

PROMISES - ASYNC/AWAIT

ES6

Promises

```
const makeRequestComplex = () => {
  returngetJSON()
    .then(data => {
      if (data.needsAnotherRequest) {
        return makeAnotherRequest(data)
          .then(moreData => {
            console.log(moreData)
            return moreData
          })
      } else {
        console.log(data)
        return data
      }
    })
}
```

Async/Await

```
const makeRequestComplex = async () => {
  const data = awaitgetJSON()
  if (data.needsAnotherRequest) {
    const moreData = await makeAnotherRequest(data);
    console.log(moreData)
    return moreData
  } else {
    console.log(data)
    return data
  }
}
```

MODULES



- Separating code into their own files (known as **modules**) which do not expose their variables and functions outside the file unless explicitly exported
- **CommonJS** — Standard that inspired Node exports
 - Server Side Focus, Compact Syntax, Synchronous Loading
- **AMD (Asynchronous Module Definition)** - Require.js
 - Browser Focus, Asynchronous Loading, More Complex Syntax
- **ES6 Modules**
 - Tries to be best of both: Compact Syntax with Asynchronous Loading

MODULES

ES6

▶ Named Export

- ▶ Modules can have as many named exports as you want
- ▶ Can export functions, classes, objects.....

```
export const square = x => x*x
export const squareRoot = x => Math.sqrt(x)

export function diag(x, y) {
    return sqrt(square(x) + square(y));
}
```

Import uses a relative path

```
import {square, squareRoot, diag} from './module_one'
```

Named imports go in curly braces{}

MODULES

ES6

▶ Default Export

- ▶ Modules can only have ONE default export

```
export default function(){}
```

```
import foo from './module_one'
```

Import of default export can be renamed ANYTHING

```
import foo, {square, squareRoot, diag} from './module_one'
```

Mixed import of default and named exports

▶ Node Modules

- ▶ Node modules are referenced by the module name (like 'react')

```
import React, { Component } from 'react'  
import {connect} from 'redux'  
import axios from 'axios'
```

BABEL

BABEL

- JavaScript Compiler — *Use next generation JavaScript, today*
- Allows you **transpile** different source codes into JavaScript
 - ES6/ES.Next —> ES5
 - JSX —> ES5
 - Flow —> ES5
 - <https://babeljs.io/repl/>

NPM/YARN



- **JavaScript Package Managers** — register of node_modules
 - **NPM (Node Package Manager)**
 - **Yarn** — build by Facebook, Google, etc... to solve commonly faced problem with NPM
 - Introduced the '*lock*' file
 - Same packages as NPM
 - **package.json** — contains all of a project dependencies

PACKAGE.JSON

Build/Test/Lint Scripts

Production Dependencies

Project metadata

Development Dependencies
(testing and building)

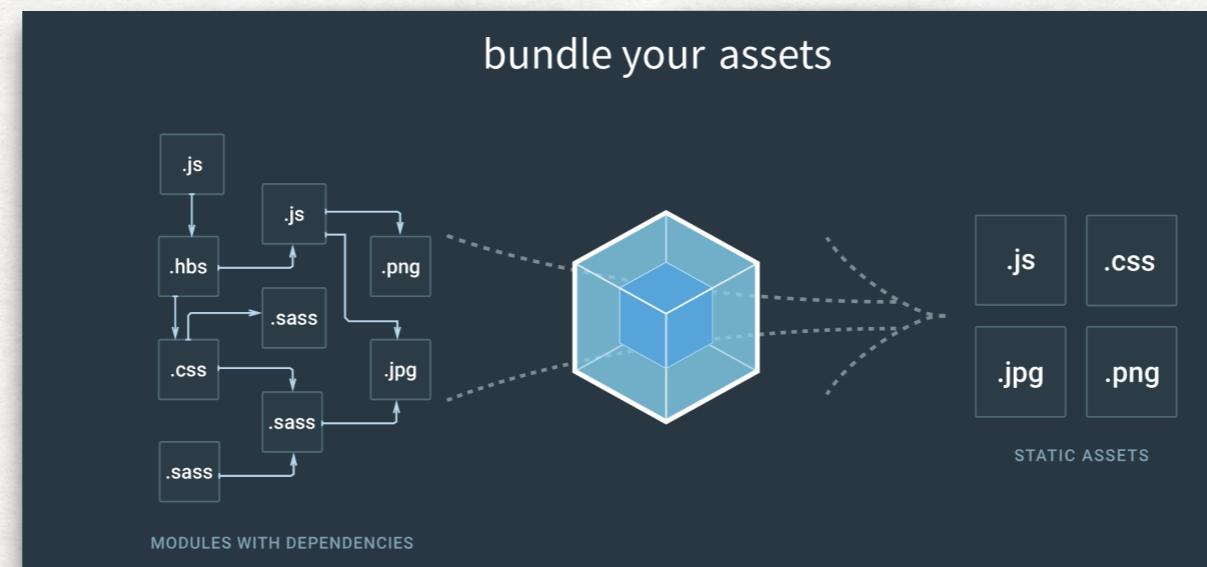
```
{  
  "name": "react-example",  
  "version": "1.0.0",  
  "description": "Simple React example app",  
  "main": "index.js",  
  "scripts": {  
    "dev": "webpack-dev-server --inline --content-base ./ --env.dev",  
    "build": "webpack --env.prod",  
    "lint": "eslint app test",  
    "test": "cross-env NODE_ENV=test nyc mocha",  
    "watch:test": "mocha --watch --compilers js:babel-register",  
    "start": "http-server",  
    "setup": "npm install && npm run validate",  
    "validate": "npm-run-all lint test build"  
  },  
  "repository": {  
    "type": "git",  
    "url": "https://github.com/mzabriskie/react-example"  
  },  
  "author": "Matt Zabriskie",  
  "license": "MIT",  
  "bugs": {  
    "url": "https://github.com/mzabriskie/react-example/issues"  
  },  
  "homepage": "https://github.com/mzabriskie/react-example",  
  "dependencies": {  
    "axios": "0.13.1",  
    "bootstrap": "^3.3.5",  
    "history": "^3.0.0",  
    "moment": "^2.10.6",  
    "react": "15.3.0",  
    "react-dom": "15.3.0",  
    "react-router": "2.6.1",  
  },  
  "devDependencies": {  
    "babel-core": "6.13.2",  
    "babel-loader": "6.2.4",  
    "babel-plugin-istanbul": "1.0.3",  
    "babel-preset-es2015": "6.13.2",  
    "babel-preset-react": "6.11.1",  
    "babel-preset-stage-2": "6.13.0",  
    "babel-register": "6.11.6",  
    "chai": "3.5.0",  
    "chai-enzyme": "0.5.0",  
    "css-loader": "0.23.1",  
    "enzyme": "2.4.1",  
    "eslint": "3.2.2",  
    "eslint-config-kentcdodds": "^9.0.2",  
    "file-loader": "0.9.0",  
    "http-server": "0.9.0",  
    "jsdom": "9.4.1",  
    "lodash": "4.14.2",  
    "mocha": "3.0.2",  
    "moxios": "0.3.0",  
    "npm-run-all": "2.3.0",  
    "nyc": "7.1.0",  
    "react-addons-test-utils": "15.3.0",  
    "sinon": "1.17.5",  
    "webpack": "2.1.0-beta.20",  
    "webpack-config-utils": "2.0.0",  
    "webpack-dev-server": "2.1.0-beta.0",  
    "webpack-validator": "2.2.7"  
  }  
}
```



WEBPACK



- **Modular Bundler for JavaScript**
 - Bundles all your files (javascript, images, styles, ...) and their dependencies into a few static files using a series of **Plugins** and **Loaders**



- **Loaders:** Preprocess files before bundling (example babel-loader for JavaScript transpiling, SASS/Less loader for CSS compiling,
- **Optimizations:**
 - Minimize/Uglify
 - Tree Shaking (dead-code elimination)
 - Code Splitting

WEBPACK CONFIG



Starting Entry Point

babel loader on js files

style/sass loaders

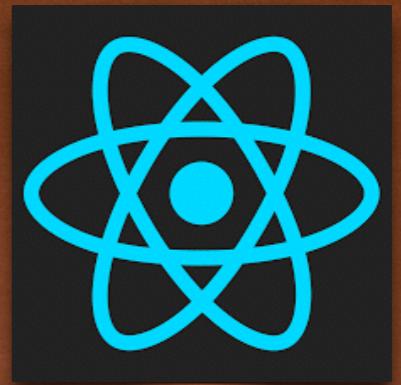
```
import path from 'path';
import HtmlWebpackPlugin from 'html-webpack-plugin';

export default () => ({
  entry: [
    path.join(__dirname, 'src/index.js'),
  ],
  output: {
    path: path.join(__dirname, 'dist'),
    filename: 'bundle.js',
  },
  plugins: [
    new HtmlWebpackPlugin({
      filename: 'index.html',
      template: './src/index.html'
    }),
  ]
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        include: path.join(__dirname, 'src'),
        use: [
          {
            loader: 'babel',
            options: {
              babelrc: false,
              presets: [
                ['es2015', {modules: false}],
                'react',
              ],
            }
          }
        ]
      },
      {
        test: /\.(\css|scss|sass)\$/,
        loader: 'style!css!sass',
      },
    ],
  },
})};
```

Final output directory and filename

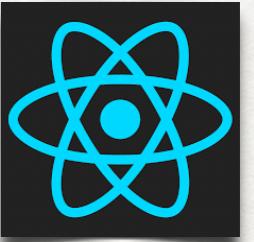
PLUGINS

LOADERS



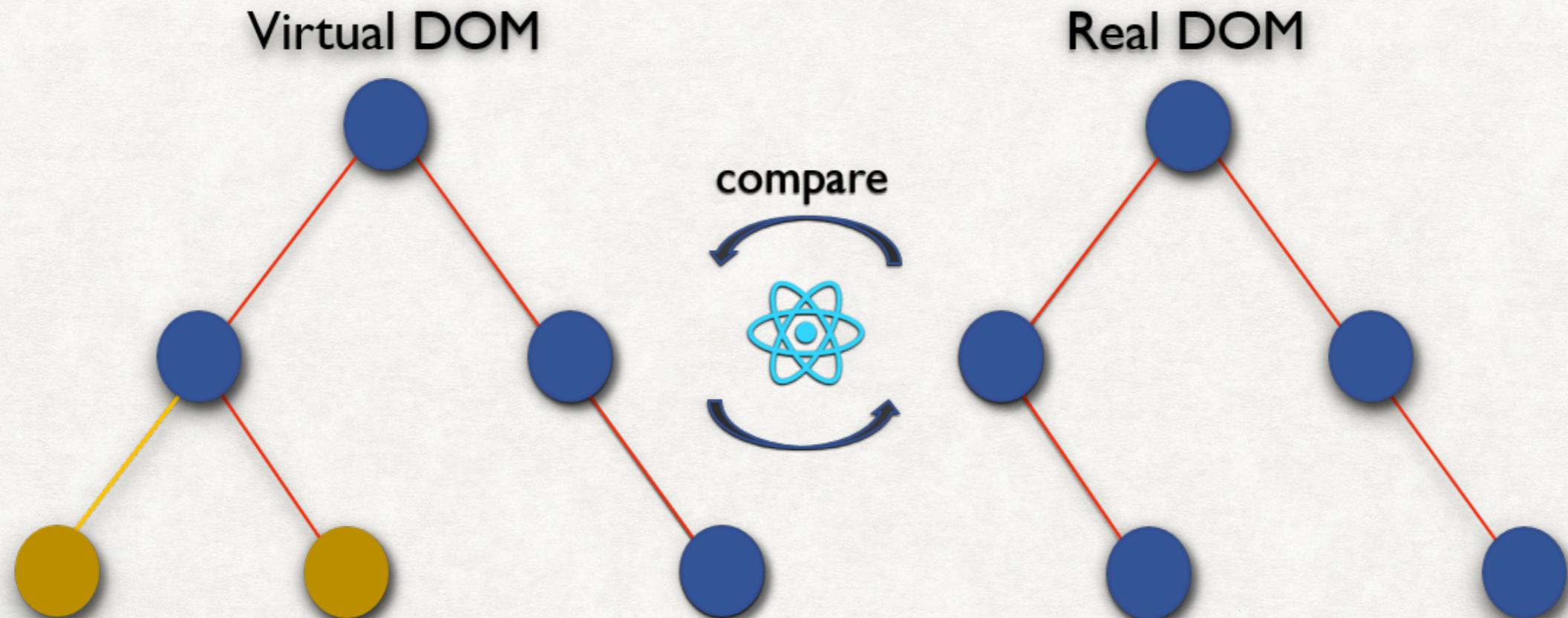
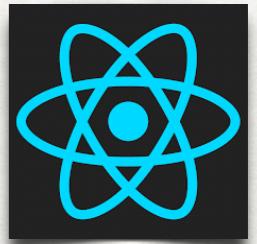
REACT

REACT

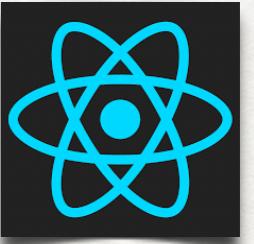


- Declarative component based UI library developed by Facebook
- Been developed at Facebook since ~2011 and was open sourced in 2013
- Latest stable release React 16 (code name *React Fiber*) was released in September 2017 and included a new reimplementation of React's core algorithm (without changing any of the API)

VIRTUAL DOM



INTRODUCTION



- Two main modules **React** and **React-DOM**
- **React DOM** is responsible for DOM interactions - most importantly rendering the initial React application in the DOM:

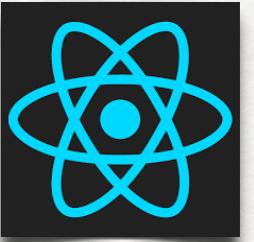
```
import ReactDOM from 'react-dom';
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
)
```

Inserting an `<h1>` element into
element with id root

```
import ReactDOM from 'react-dom';
ReactDOM.render(<App />, document.getElementById('root'));
```

Insert the `<App>` component

JSX



- **JavaScript XML** — syntactic extension of JavaScript that allows you describe what your UI should look like
- Syntax may look like HTML or a templating language but it is fully functional JavaScript

```
const element = <h1>Hello, world!</h1>
```

```
<h1>  
  {2+2}  
</h1>
```

Expressions can be evaluated
within curly brackets ({})

style takes an object

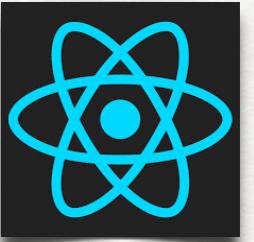
```
const MyComponent = <span>Hello world</span>
```

```
const element = <div tabIndex=0 style={{color: 'green', background:'blue'}} className="my-class">< MyComponent/></div>
```

DOM Attributes become camel case

class → className because class is a keyword

COMPONENTS



- **Class Component**

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

class Welcome extends Component {
  render() {
    return <div>Welcome!</div>
  }
}

export default Welcome
```

Class that extends `React.Component`

- Component State
- Component Lifecycle methods
- More verbose
- *Render method returns JSX*

- **Functional Component**

```
import React from 'react'

const Welcome = () => {
  return <div>Welcome!</div>
}

export default Welcome
```

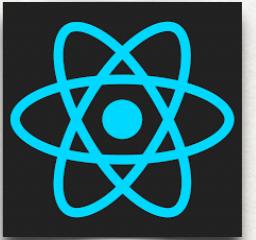
- No State or Lifecycle methods
- More concise
- *Returns JSX to be rendered*

```
import Welcome from './Welcome'

const App = () => {
  return <div><Welcome/></div>
}
```

<Welcome> can be imported and used in another component's JSX

PROPS



- **Props** — a means of passing information from a parent Component to a child component
- *Read Only* within a component

props passed to children as tag attributes

```
<App>
  <Welcome firstName='Jane' lastName='Smith' />
</App>
```

Class Component

```
class Welcome extends Component {
  render() {
    return <div>Welcome {this.props.firstName} {this.props.lastName}!</div>
  }
}
```

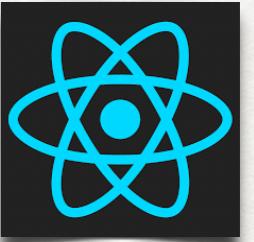
props accessed via `this.props`

Functional Component

props passed as the parameter to the function

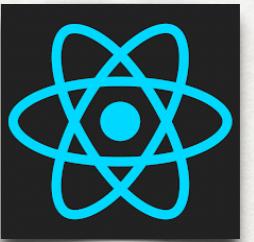
```
const Welcome = (props) => {
  return <div>Welcome {props.firstName} {props.lastName}!</div>
}
```

STATE



- While **props** are a way to pass data from parent to child **State** is a components internal data store
- Class component only
- **Plain JavaScript Object**
 - Accessed with `this.state`
 - Manipulated `this.setState`
- Isolated to the component — cannot be accessed by component's children or parents

STATE



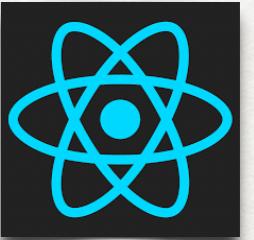
- Initialized in the component's **constructor**
- Access with **this.state**

constructor initialization

```
class Counter extends Component {  
  constructor(props) {  
    super(props)  
    this.state = {  
      count: props.initialCount  
    }  
  }  
  
  render() {  
    return <div>Count: {this.state.count}</div>  
  }  
}
```

Accessing state via **this.state**

STATE



- Setting state with `this.setState`:

```
this.setState({count: 2})
```

Merges new count will all other state properties

```
Object.assign(  
  previousState,  
  {count: state.count + 1},  
  {count: state.count + 1},  
  ...  
)
```

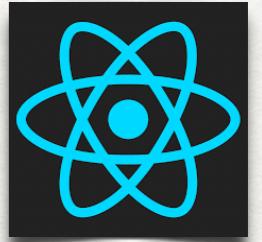


`this.setState` is asynchronous

```
this.setState ((prevState, props) => {  
  return {count: prevState.count + 1}  
})
```

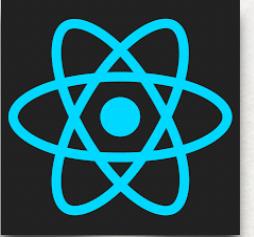
Instead of an object pass an updater function

LIFE CYCLE METHODS



- Class components have hooks into several of React's Lifecycle methods
- 4 Categories:
 - Mounting
 - Updating
 - Unmounting
 - Error Handling (new to React 16)

LIFE CYCLE METHODS



Mounting — Called once when the component is first being inserted into the DOM:

1. `constructor(props)`

```
constructor(props) {
  super(props)
  this.state = {
    count: props.initialCount
}
```

2. `componentWillMount(props)`

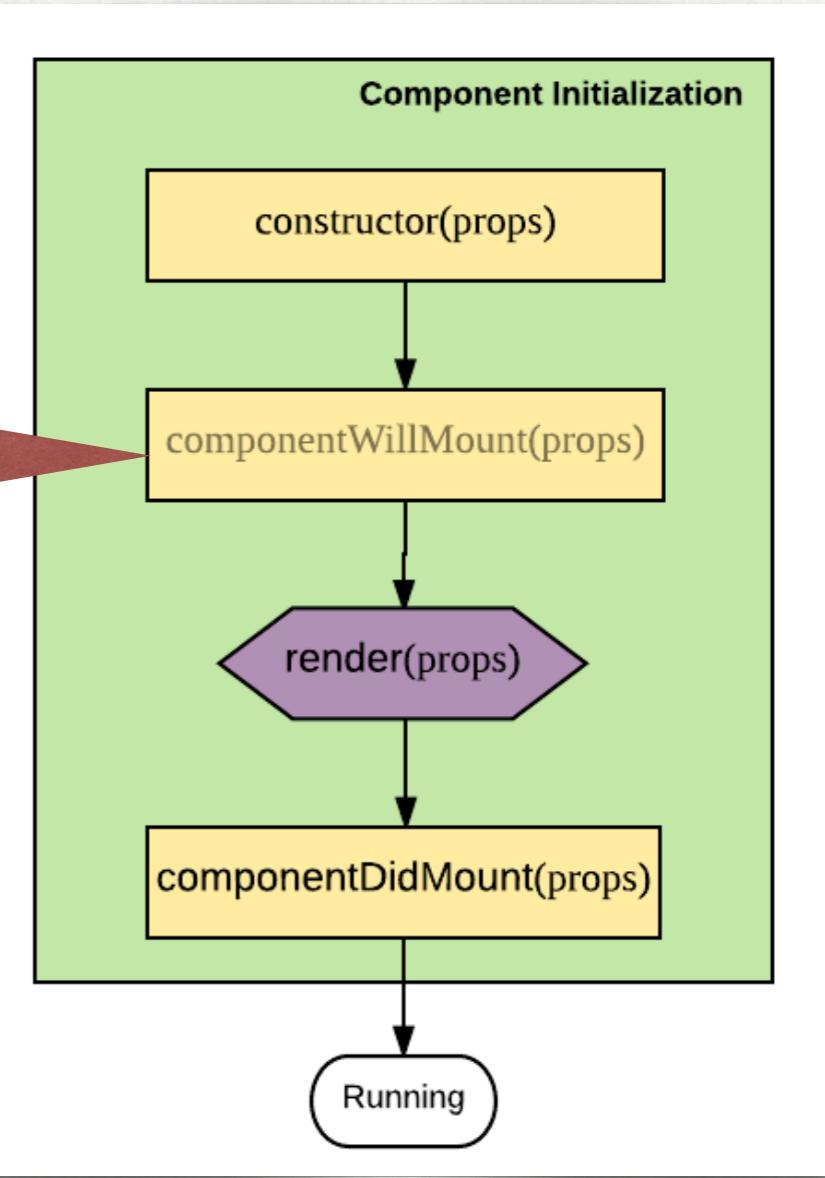
Avoid componentWillMount

3. `render()`

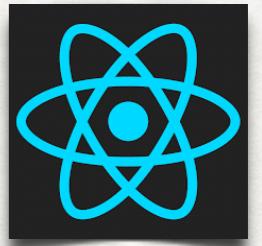
4. `componentDidMount(props)`

```
componentDidMount() {
  fetch('https://gitconnected.com')
    .then((res) => {
      this.setState({
        user: res.user
      });
    });
}
```

Dom manipulation and data loading in
`componentDidMount`



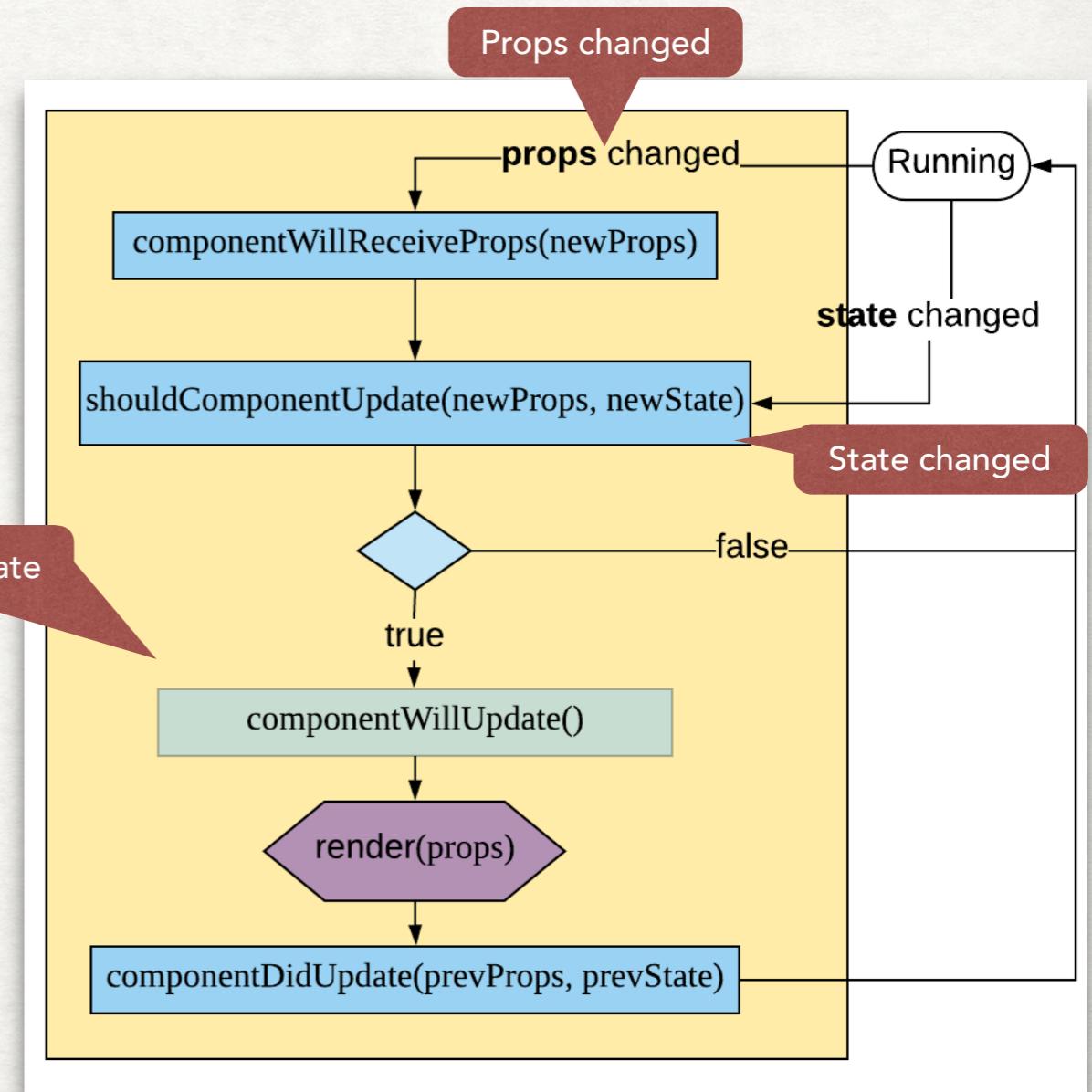
LIFE CYCLE METHODS



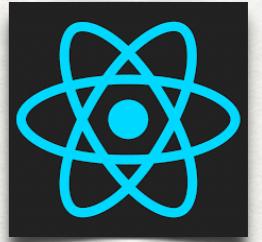
Updating — Called many times in a component's life cycle in response to changes in `this.props` or `this.state`

1. `componentWillReceiveProps(newProps)`
2. `shouldComponentUpdate(newProps, newState)`
3. ~~`componentWillUpdate()`~~
4. `render()`
5. `componentDidUpdate(prevProps, prevState)`

Avoid `componentWillUpdate`



LIFE CYCLE METHODS



1. componentWillReceiveProps(newProps)

- Called when new props are passed in from the parent

```
componentWillReceiveProps(nextProps) {  
  if (this.props.id !== nextProps.id) {  
    this.setState({  
      feedContent: []  
    });  
  }  
}
```

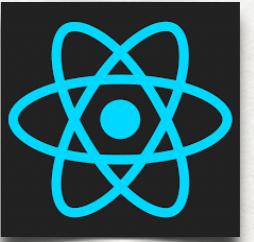
2. shouldComponentUpdate(nextProps, newState)

- Determines if a component should re-render based on the new props and state

```
shouldComponentUpdate(nextProps, nextState) {  
  return this.props.color !== nextProps.color  
}
```

Override for required optimizations only

LIFE CYCLE METHODS



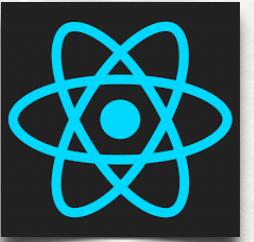
5. componentDidUpdate(prevProps, prevState)

- Called after a component has re-rendered after a change

```
componentDidUpdate(prevProps, prevState) {  
  // only update chart if the data has changed  
  if (prevProps.data !== this.props.data) {  
    this.chart = c3.load({  
      data: this.props.data  
    });  
  }  
}
```

Place for code requiring DOM interactions

LIFE CYCLE METHODS



Unmounting — begins *just before* a component is about to be removed from the DOM (unmounted)

- Simple lifecycle method `componentWillUnmount`

Place for any clean up like removing event listeners or timers

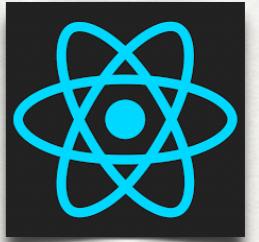
```
componentWillUnmount() {  
  window.removeEventListener('resize', this.resizeEventHandler);  
}
```

Error Handling

- Introduced in React 16 - `componentDidCatch` allows you to create Error Boundaries

```
class ErrorBoundary extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { hasError: false };  
  }  
  
  componentDidCatch(error, info) {  
    // Display fallback UI  
    this.setState({ hasError: true });  
    logErrorToMyService(error, info);  
  }  
  
  render() {  
    if (this.state.hasError) {  
      // You can render any custom fallback UI  
      return <h1>Something went wrong.</h1>;  
    }  
    return this.props.children;  
  }  
}
```

HIGHER ORDER COMPONENTS



- Pattern developed to facilitate avoiding code duplication and replace depreciated mixins
- Function that takes in a component to be wrapped and returns a new component with additional functionality

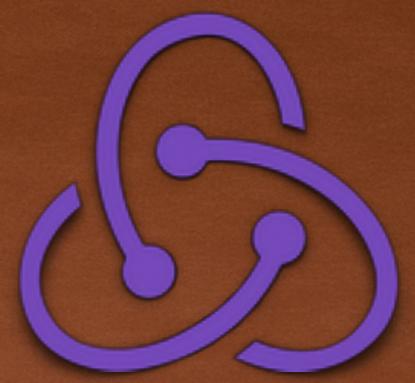
Creates a new component with a custom componentWillMountReceiveProps

```
function logProps(WrappedComponent) {
  return class extends Component {
    componentWillMount(nextProps) {
      console.log('Current props: ', this.props);
      console.log('Next props: ', nextProps);
    }
    render() {
      return <WrappedComponent {...this.props} />
    }
  }
}
```

Returns the original component passing in all the props with the spread operator

```
const MyComponent extends Component {
  render() {
    return <div>Component to be Wrapped</div>
  }
}

export default logProps(MyComponent)
```

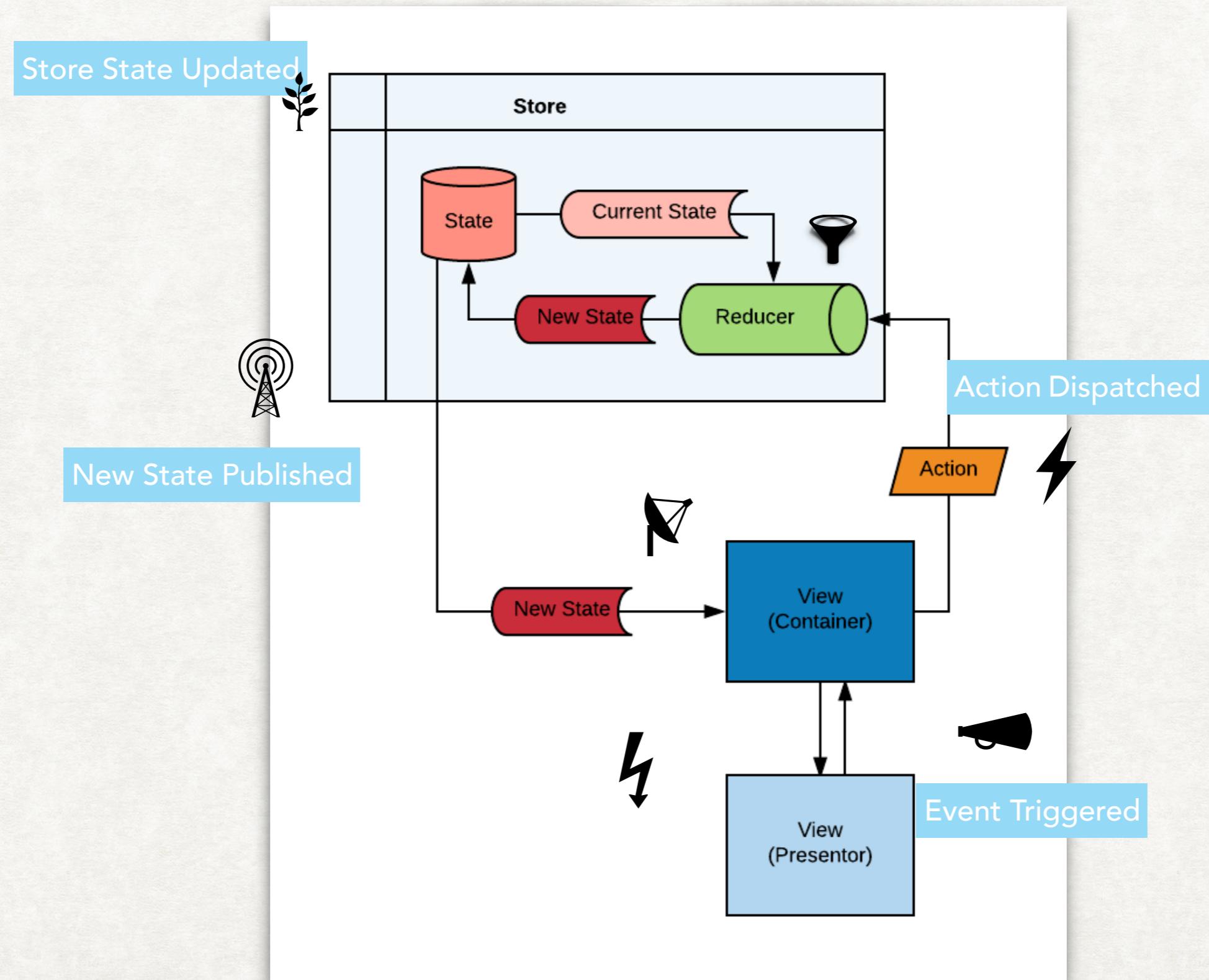


REDUX

REDUX

- Redux is library build to simplify and centralize application state which predictable changes.
- Enforces unidirectional data flow based on Flux Architecture
- 3 Core Concepts
 1. Single Source of Truth
 2. Store is Read Only
 3. Changes are made with Pure Functions

REDUX



REDUX - ACTIONS



► Actions

- Plain JavaScript Object that describes a change potential changes to the store
- Must contain a **type**

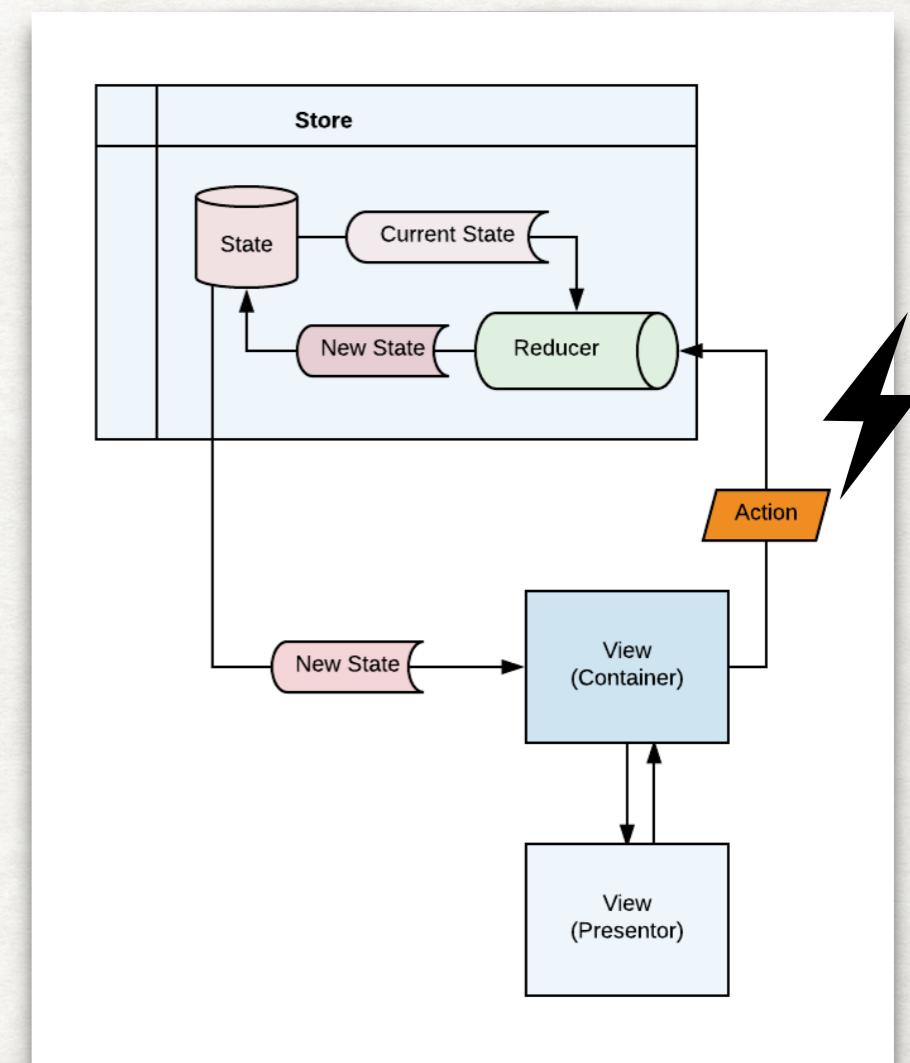
```
const ADD_TODO = 'ADD_TODO'

const addAction = {
  type: ADD_TODO,
  text: 'Buy milk from the store'
}
```

► Action Creators

- Common pattern — functions that are responsible for creating actions

```
const addTodo = text =>{
  return {
    type: ADD_TODO,
    text: 'Buy milk from the store'
  }
}
```



REDUX - REDUCERS



- ▶ Functions responsible for generating new state based on the Action and Previous State

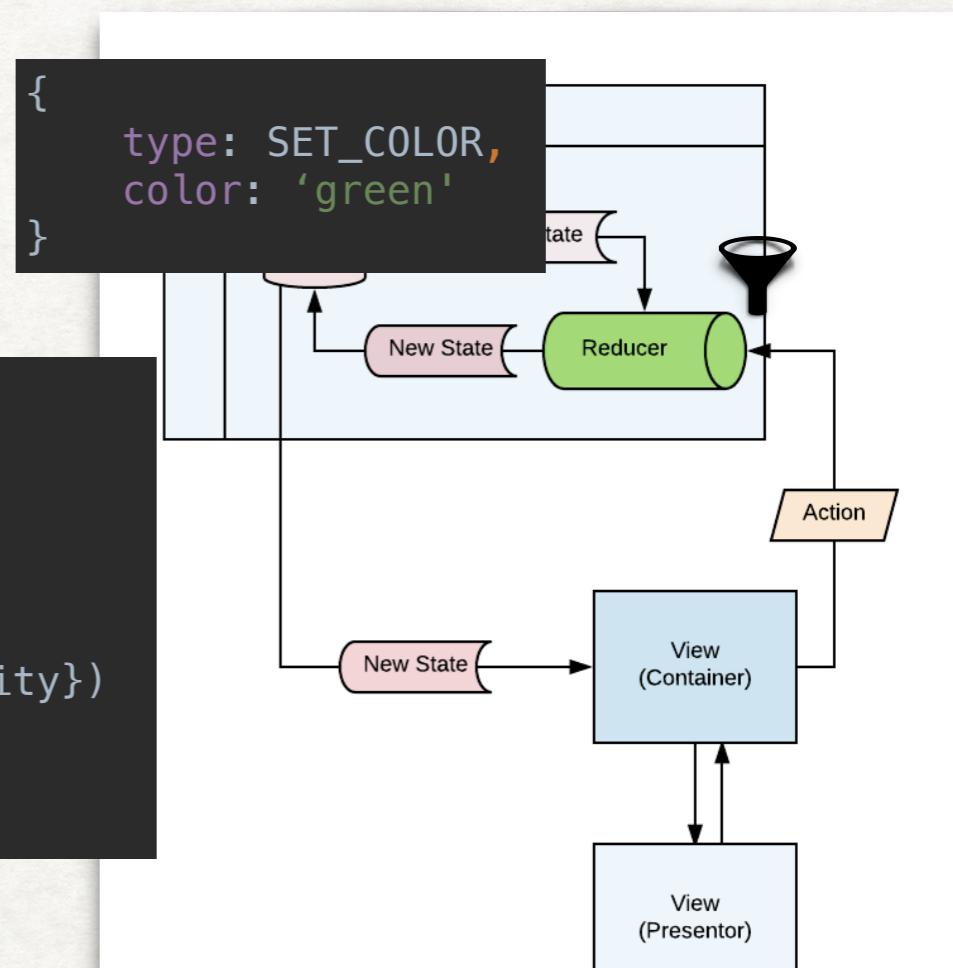
```
(previousState, action) => newState
```

- ▶ Must be **PURE** — cannot mutate the previous state

Assigns default initial state

```
const reduce = (state = {}, action) => {  
  switch (action.type) {  
    case SET_COLOR:  
      return Object.assign({}, state, {color: action.color})  
    case SET_QUANTITY:  
      return Object.assign({}, state, {quantity: action.quantity})  
    default:  
      return state  
  }  
}
```

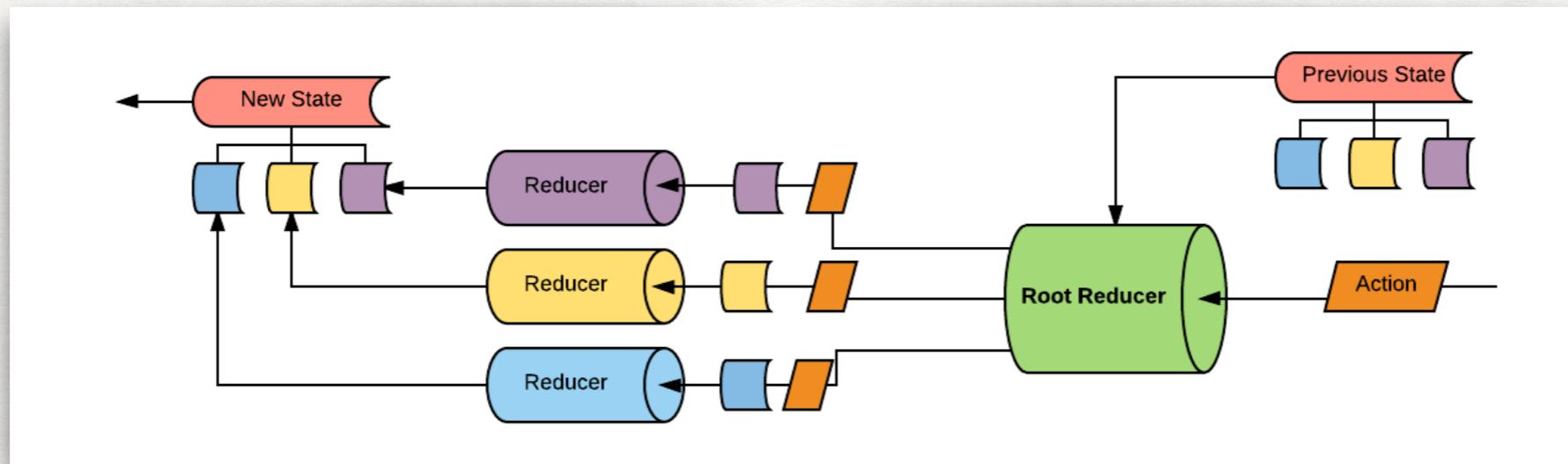
Returns previousState if no changes performed



REDUX - REDUCERS



- ▶ Multiple reducers can be combined to form a tree of reducers — the top level state will also be



```
import combineReducers from 'redux'
import potatoReducer from './potato'
import tomatoReducer from './tomato'

rootReducer = combineReducers({
  potato: potatoReducer,
  tomato: tomatoReducer
})
```



Resulting nested state

```
{
  potato: {
    // ... potatoes, and other state managed by the potatoReducer
  }
  tomato: {
    // ... tomatoes, and other state managed by the tomatoReducer
  }
}
```

REDUX - STATE/STORE



▶ State

- ▶ Single plain JavaScript object containing nested structure for the entire application
- ▶ Lives within the Store

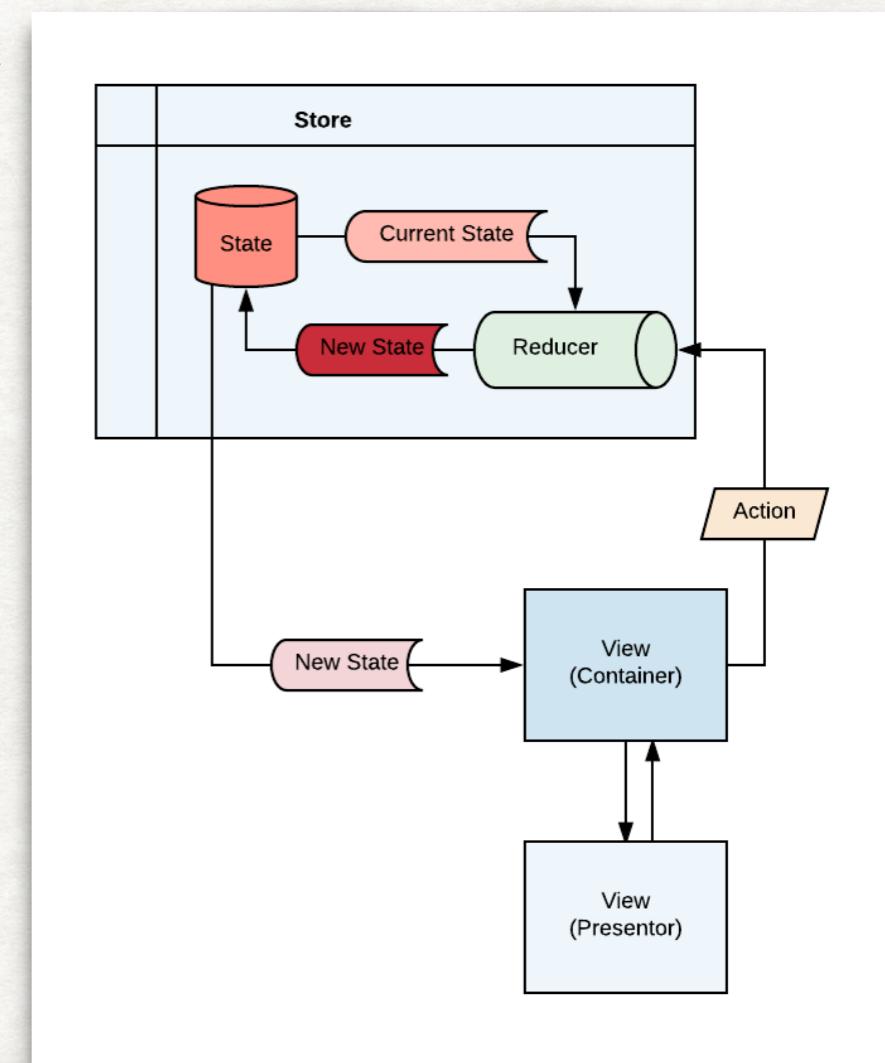


▶ Store

- ▶ Holds the state
- ▶ Allows actions to be dispatched to update state
- ▶ Notifies subscribed listeners on state changes

```
import { createStore } from 'redux'  
import app from './reducers'  
const store = createStore(app)
```

Initialized with root reducer



REACT REDUX BINDINGS



- ▶ Library that contains a series of bindings that greatly help integrate Redux into a react app
- ▶ Provider wraps entire app

```
import {Provider} from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <MyRootComponent />
  </Provider>,
  rootEl
)
```

Provider takes in the created store as a prop

- ▶ Provider gives all of the children component's access to the store via the second binding **connect**

REACT REDUX BINDINGS



- ▶ Connect is a Higher Order component that takes 2 main functions as parameters:
 - ▶ `mapStateToProps(state, [ownProps]): stateProps`

```
const mapStateToProps = state => {
  return {
    todos: state.todos
  }
}
```

Extract redux state and injects them into the component as *props*

- ▶ `mapDispatchToProps(dispatch, [ownProps]): dispatchProps`

```
const mapDispatchToProps = dispatch => {
  return {
    onTodoClick: id => {
      dispatch(toggleTodo(id))
    }
  }
}
```

Create functions that allow the component to dispatch actions to the redux store and injects them as *props*

REACT REDUX BINDINGS



```
import React, { Component } from 'react'
import { connect } from 'react-redux'

import UserProfile from './UserProfile'
import { updateProfile } from './profileActions'

class Profile extends Component {
  render() {
    const { profile, updateProfile } = this.props
    return <UserProfile firstName={profile.firstName} lastName={profile.lastName} email={profile.email}
      updateProfile={updateProfile}/>
  }
}

const mapStateToProps = state => {
  return {
    profile: state.profile
  }
}

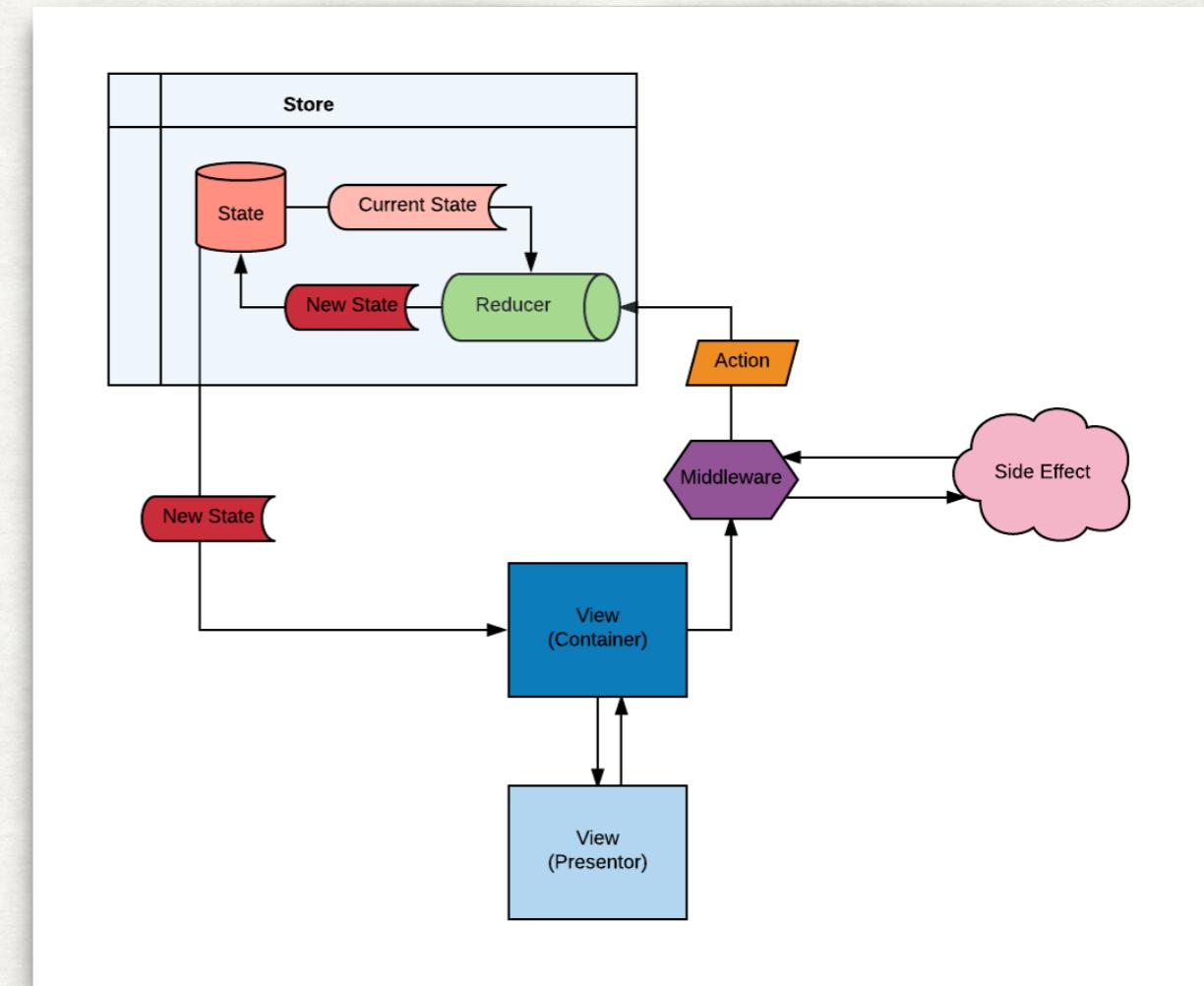
const mapDispatchToProps = dispatch => {
  return {
    updateProfile: (profile) => dispatch(updateProfile(profile))
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(Profile)
```

REACT SIDE EFFECTS - MIDDLEWARE



- ▶ **Redux Middleware** — Provides 3rd party extension points between dispatching an action and it reaching the reducer.
 - ▶ Api Calls, Logging, etc...
- ▶ Many middleware for asynchronous logic:
 1. Redux-Thunks
 2. Redux-Saga
 3. Redux-Promise



REACT SIDE EFFECTS - THUNKS

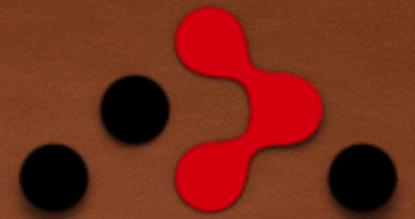


- ▶ Thunks allow you to dispatch a function to your store instead of just an action object
- ▶ This function allows the delay of dispatching the action until after the asynchronous call has completed

```
function setCats(cats) {  
  return {  
    type: 'SET_CATS',  
    cats  
  }  
  
function loadCats(catType) {  
  return async function (dispatch) {  
    const cats = await loadCatsofType(type)  
    return dispatch(setCats(cats))  
  }  
}
```

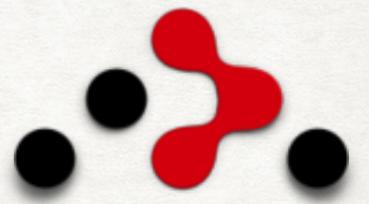
Thunk returns a function with `dispatch` as its parameter

```
store.dispatch(loadCats('Siamese'))
```



REACT ROUTER

REACT ROUTER



- The most popular routing library for React apps
- The entire app should be wrapped in a `<Router>` component
- Individual `<Route>` components map urls to specific components
- `<Link>` components allow for navigation within the application

```
import {BrowserRouter as Router} from 'react-router-dom'

<Router>
  <div>
    <ul>
      <li><Link to="/">Home</Link></li>
      <li><Link to="/about">About</Link></li>
      <li><Link to="/topics">Topics</Link></li>
      <li><Link to="/profiles/123">Profile</Link></li>
    </ul>

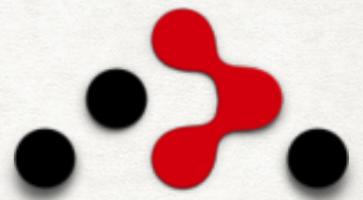
    <hr/>

    <Route exact path="/" component={Home}/>
    <Route path="/about" component={About}/>
    <Route path="/topics/" component={Topics}/>
    <Route path="/profiles/:id" component={Profile}/>
  </div>
</Router>
```

<Routes> map urls to components

<Link> provide navigation links

REACT ROUTER



- Component's passed to Router get passed 3 props from the Router:

1. **match** — information about the matched url to extract params

2. **location** — information where the app is and query params

3. **history** — allows for programmatic navigation

```
const Child = ({ match, location, history }) => (
  <div>
    <h3>ID: {match.params.id}</h3>
  </div>
)
```