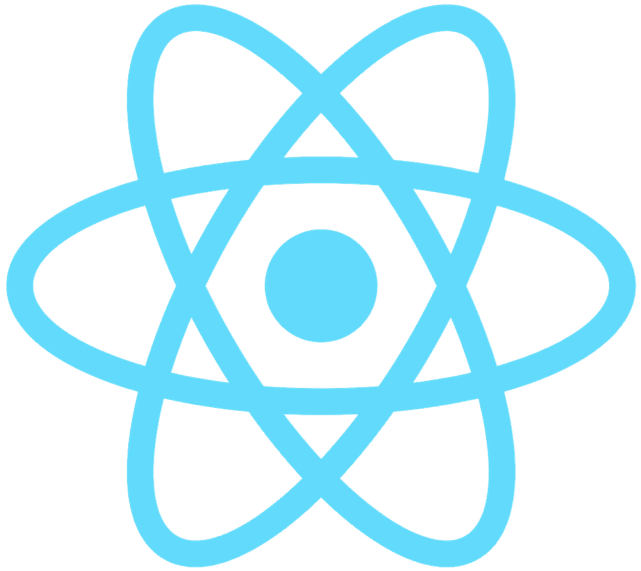


React + Redux

Nik Graf

@nikgraf

nik@nikgraf.com

Creator of Belle (UI Components)

Working with StarterSquad

Travelled around the World

ECMAScript2015

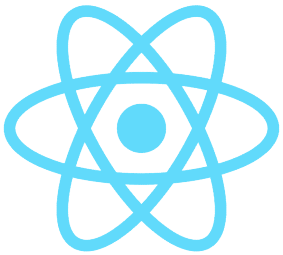
```
const sum = (first, second) => {
  return first + second;
}
```

BABEL

Created by Sebastian McKenzie

- ECMAScript 2015 Support, JSX Support
- Widely adopted


Let’s get started
ource: https://www.flickr.com/photos/mike52ad/



React

React is a JavaScript Library for building user interfaces.

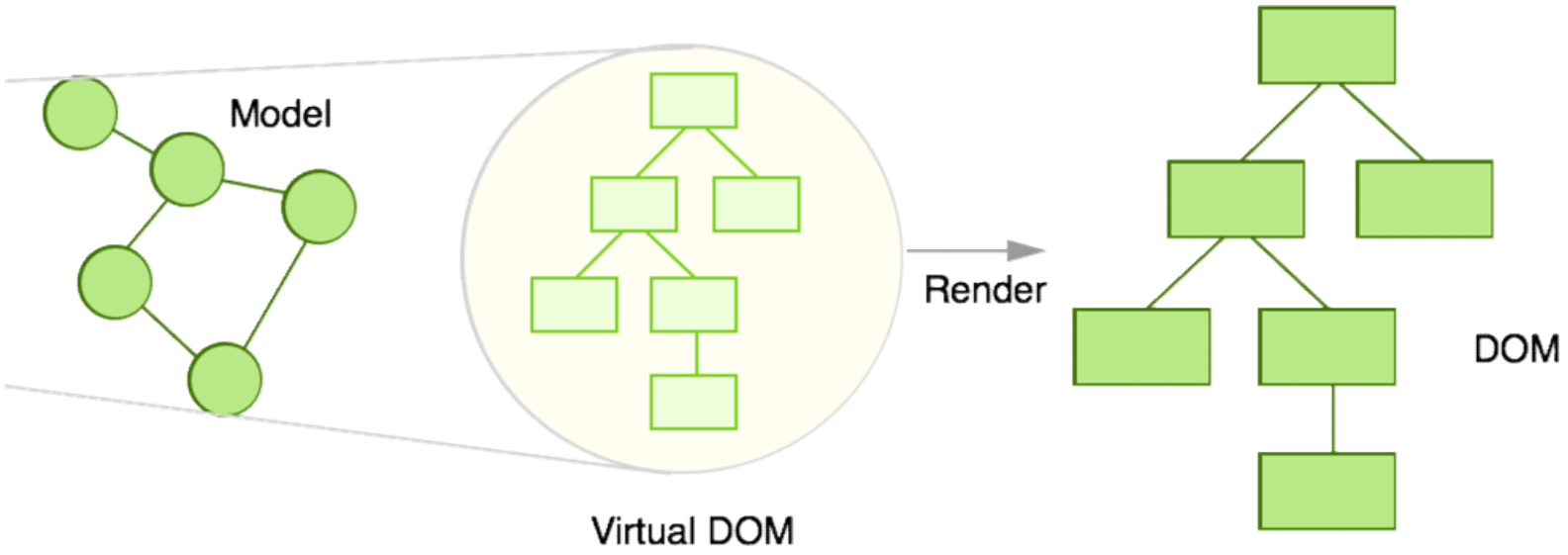
- Focus on the UI, not a Framework
- One-way reactive data flow (no two-way data binding)
- Virtual DOM

Virtual DOM

Keep track of state in DOM is hard.

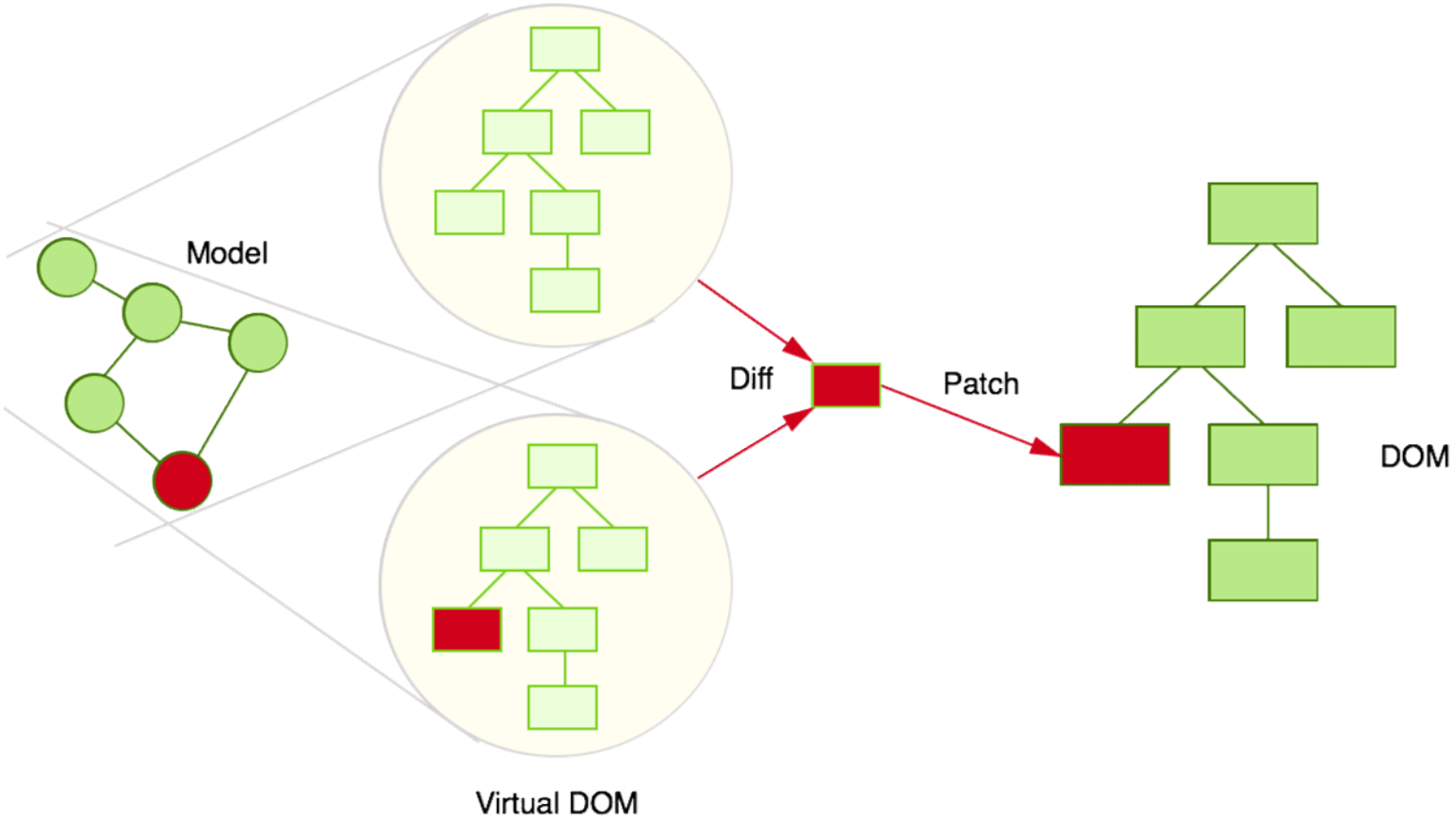
The DOM API is slow.
(Try to re-render the whole DOM on every change)

Virtual DOM



Source: http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

Virtual DOM



Source: http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

Virtual DOM Benefits

Batched DOM read/write operations.

Efficient update of sub-tree only.

Our first Experiment Part I

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <script src="bundle.js"></script>
  </head>
  <body>
    <div id="example"></div>
  </body>
</html>
```

index.html

Our first Experiment Part II

```
import React from 'react';
import ReactDOM from 'react-dom';

const exampleElement = document.getElementById('example');
ReactDOM.render(<h1>Hello, world!</h1>, exampleElement);
```

main.js -> bundle.js

JSX

JSX is a JavaScript syntax extension
that looks similar to XML.

```
// Input (JSX):
var app = <Nav color="blue" />;
// Output (JS):
var app = React.createElement(Nav, {color:"blue"});
```

Rendering a Component

```
import React from 'react';
import ReactDOM from 'react-dom';

const App = () => {
  return (<p>Hello World!</p>);
}

const exampleNode = document.getElementById('example');
ReactDOM.render(<App />, exampleNode);
```

main.js -> bundle.js

Rendering a Component

```
<body>
  <div id="example">
    <p> <!-- App start -->
      Hello World!
    </p> <!-- App end -->
  </div>
</body>
```

index.html

Nested Components Part I

```
import React from 'react';

const Profile = ({avatar, name}) => {
  return (
    <div>
      <img src={avatar} />
      <span>{name}</span>
    </div>
  );
}
```

profile.js

Nested Components Part II

```
import React from 'react';
import ReactDOM from 'react-dom';
import Profile from `./profile';

const App = () => {
  return (
    <div>
      <h1>Hello World!</h1>
      <Profile avatar="http://test.png" name="Nik" />
    </div>
  );
}

const exampleNode = document.getElementById('example');
ReactDOM.render(<App />, exampleNode);
```

main.js -> bundle.js

Nested Components Part III

```
<body>
  <div id="example">
    <div> <!-- App start -->
      <h1>Hello World!</h1>
      <div> <!-- Profile start -->
        <img src="http://test.png" />
        <span>Nik</span>
      </div> <!-- Profile end -->
    </div> <!-- App end -->
  </div>
</body>
```

index.html

Stateless Function Components

Functional Programming:

- avoid changing-state
- avoid mutable data
- calling a function twice with the same values as arguments will produce the same result

Stateless Function Components:

- avoid changing-state
- avoid mutable data
- calling a function twice with the same values as arguments will produce the same result

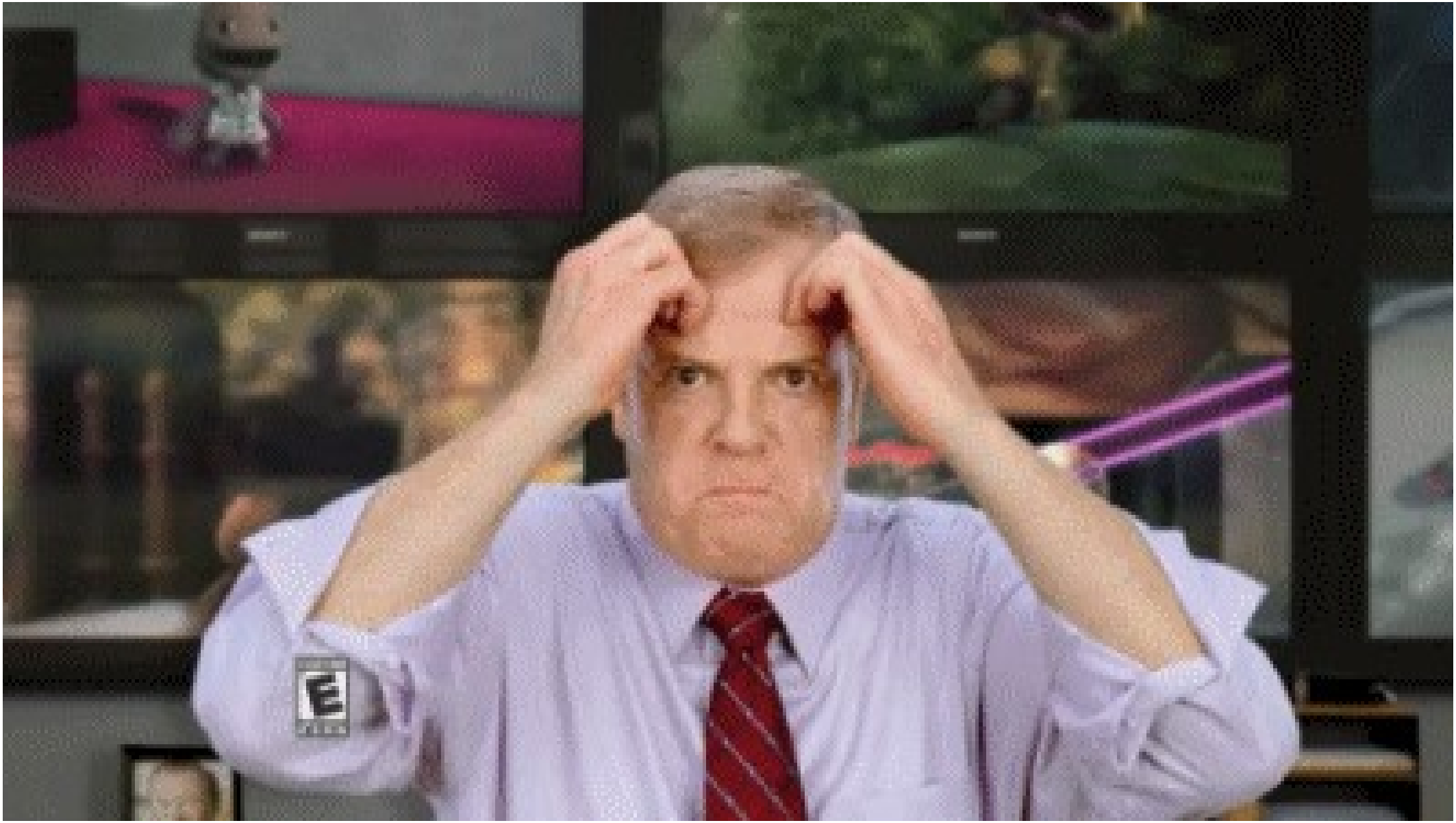
Wait, but why?

Predictable

easy to understand

&

easy to test



If/Else

```
const Profile = ({name, isOnline}) => {
  let onlineIndicator;
  if (isOnline) {
    onlineIndicator = (<span>green</span>);
  } else {
    onlineIndicator = (<span>red</span>);
  }

  return (
    <div>
      {name} {onlineIndicator}
    </div>
  );
}
```

profile.js

If/Else

```
<Profile name="Nik" isOnline={false}/>
```

```
<div>
  Nik <span>red</span>
</div>
```

Loop

```
const FriendList = ({friends}) => {
  return (
    <ul>
        {friends.map((friend) => {
          return <li>{friend.name}</li>;
        })}
    </ul>
  );
}
```

friendlist.js

Loop

```
const friends = [
  { name: 'Max'},
  { name: 'Tom'},
];

<FriendList friends={friends} />
```

```
<ul>
  <li>Max</li>
  <li>Tom</li>
</ul>
```

React Summary

- We can create out own components

- We can nest components as we like

- Stateless Function Components are pure

- We can control flow via JS (if, else, for, map …)

Interaction

```
const Profile = ({name}) => {
  return (
    <div>
      {name}
      <button onClick={ console.log('Clicked!'); }>
        Click me!
      </button>
    </div>
  );
}
```

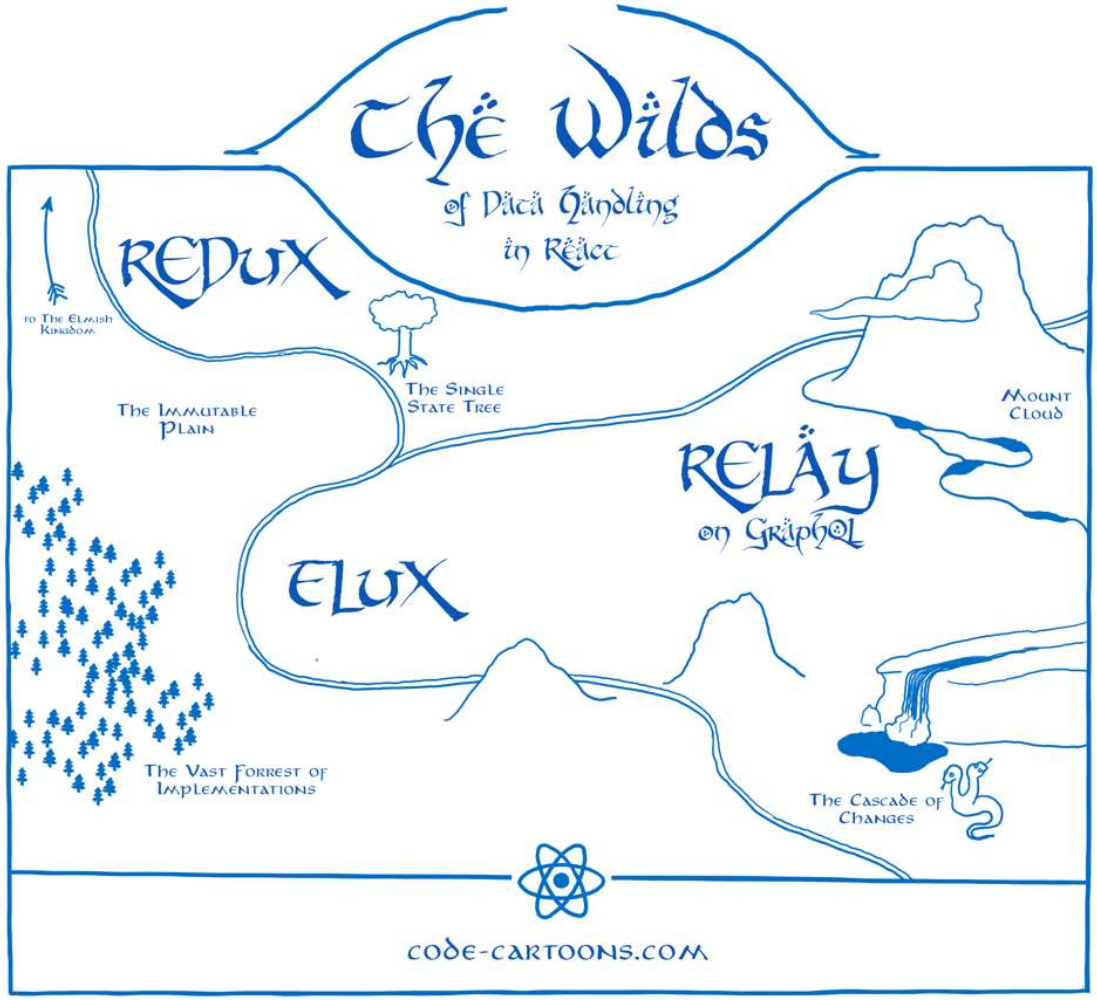
profile.js

What to do with interactions like **onMouseOver**, **onSubmit** & **onClick**?

Redux to rescue!

Redux allows you to manage the *state* with a minimal API but completely *predictable* behaviour.

What about Flux?

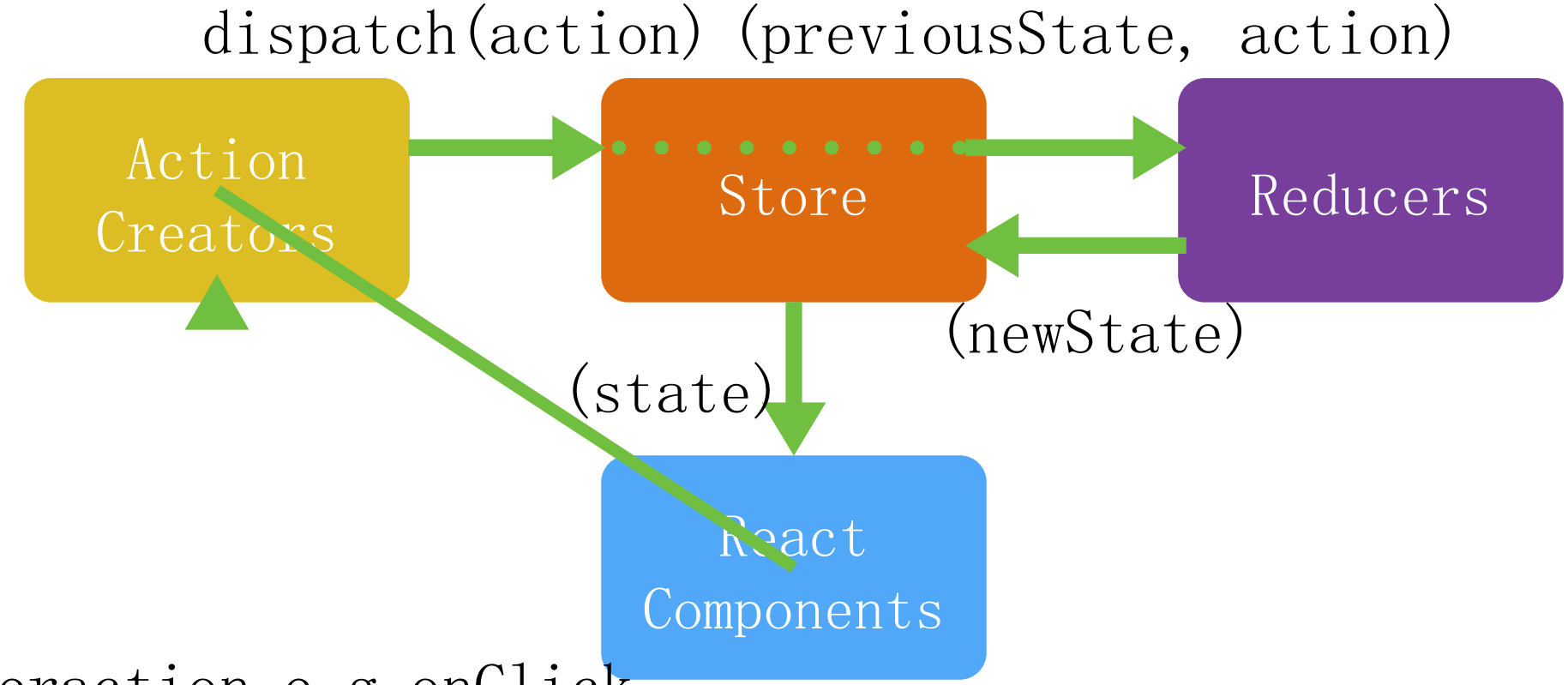


Source: https://twitter.com/codecartoons/status/667348216669741056

Basic Principle

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

Redux Flow



Interaction e.g onClick

Feels like Fear just turned into a Superpower

Action

```
const action = {
  type: 'ADD_TODO',
  text: 'Call Mom',
}
```

Action Creator

```
function addTodo(text) {
  const trimmedText = text.trim();
  return {
    type: 'ADD_TODO',
    text: trimmedText,
  }
}
```

actions.js

```
<button onClick={ dispatch(addTodo('Call Mom  ') }>
  Add Todo
</button>
```

Reducer

```
const todos = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    default:
      return state
  }
}
```

reducers.js

Store

```
import { createStore } from 'redux'
import todoReducer from '../reducers'
let store = createStore(todoReducer);

store.subscribe(() =>
  console.log(store.getState())
)

store.dispatch(addTodo('Learn about reducers'));
store.dispatch(addTodo('Call Mom'));
```

Connect React with Redux

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore } from 'redux';
import { Provider } from 'react-redux';
import todoApp from './reducers';
import App from './containers/App';

let store = createStore(todoApp);

let exampleNode = document.getElementById('example');
ReactDOM.render(
  <Provider store={store}><App /></Provider>,
  exampleNode
);
```

Connect React +Redux

```
import React from 'react';
import { connect } from 'react-redux';
import { addTodo } from '../actions.js';

const App = ({dispatch, state}) => {
  return (
    <button onClick={ dispatch(addTodo('Call Mom') }>
      Add Todo
    </button>
  );
};

export default connect(App);
```

Redux Summary

- Redux allows you to manage the *state* with *predictable* behaviour.

- (previousState, action) => newState


And when Superman flew around the world
counterclockwise, he went back in time.

Time-travel Demo

Why this Stack?

- Reusable Components
- Predictable Code (functional)
- TimeTravel
- Performant & lightweight

Is it production ready?

React

- used by Facebook, Firefox, Airbnb and many more

Redux

- used by Firefox, Docker, Technical University of Vienna, Mattermark and many more

- “Love what you’re doing with Redux”
  Jing Chen, creator of Flux

Vienna React Meetup

Source: http://www.wolfography.at/

The End

Thanks for listening!

https://github.com/nikgraf
https://twitter.com/nikgraf

Vienna React Meetup

http://www.meetup.com/Vienna-ReactJS-Meetup/