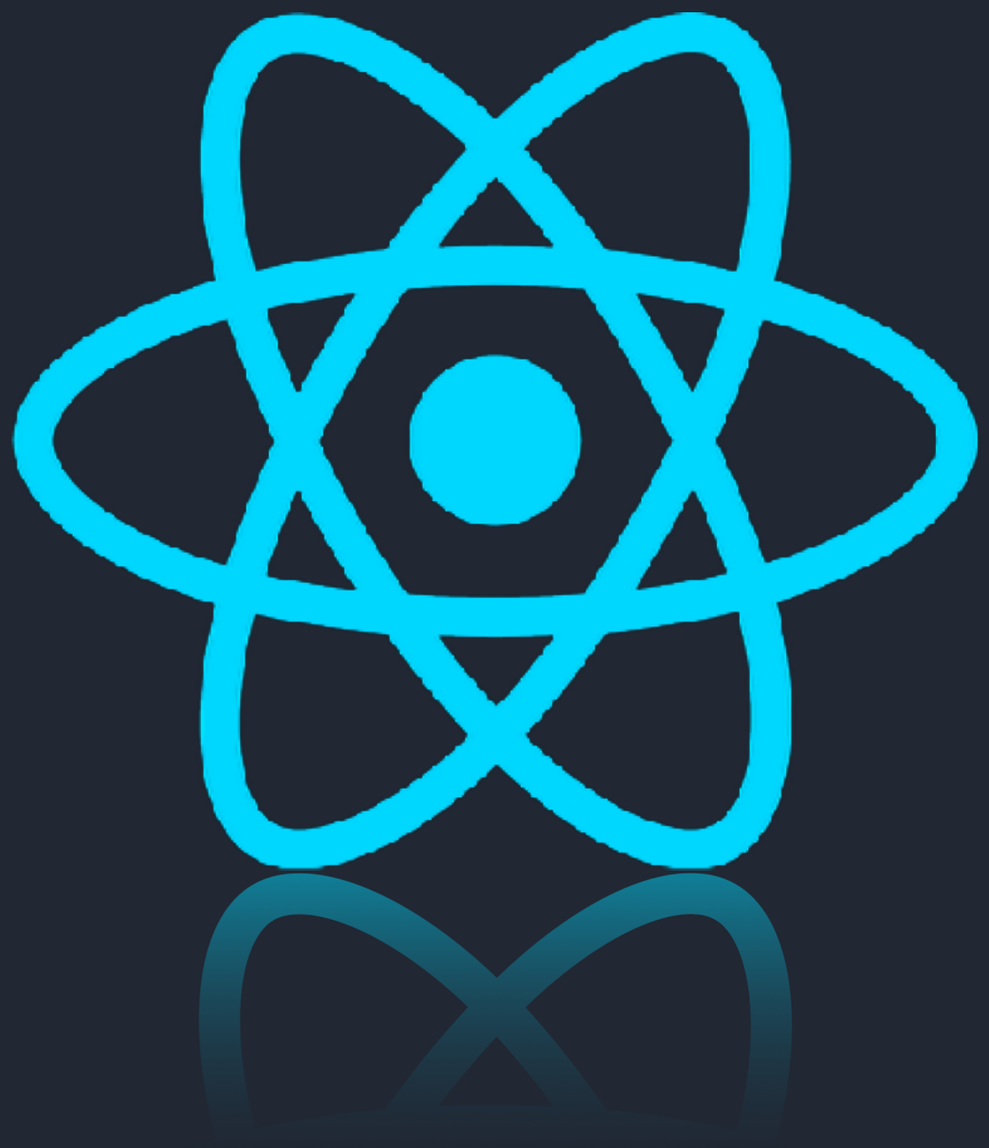


Front-End Development with React

Mail: jonas.bandi@ivorycode.com

Twitter: @jbandi

ABOUT ME

Jonas Bandi
jonas.bandi@ivorycode.com
Twitter: @jbandi

- Freelancer, in den letzten 4 Jahren vor allem in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse: Web-Technologien im Enterprise
  UBS, Postfinance, Mobiliar, BIT, SBB ...

JavaScript / Angular / React
Schulungen / Unterstützung / Reviews:
http://ivorycode.com/#schulung

Material

Git Repository:
`https://bitbucket.org/jonasbandi/react-axa-2018-2/`

Initial Checkout:
```
git clone https://bitbucket.org/jonasbandi/react-axa-2018-2.git
```

Update:
```
git pull
```

```
git reset --hard
git pull
```
(setzt alle lokalen Veränderungen zurück)

Slides & Exercises: `<checkout>/00-CourseMaterial`

AGENDA

DAY 1

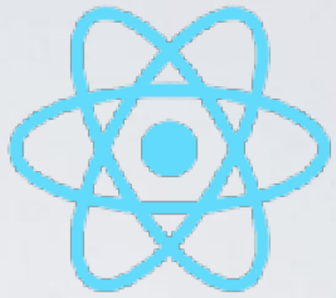
- **Intro**
- **Getting started quickly with create-react-app**
- **JSX and React Components**
- **Exercise: Your React component**
- **Components in Detail**
- **Exercise: ToDo App**

DAY 2

- **Routing**
- **Backend Access**
- **Advanced Statemanagement**
- **Using React with Flow or TypeScript**

SIDE-TRACK

- **Modern JavaScript Development**
- **JavaScript: ES5 & ES2015+**
- **TypeScript & Flow**

React 

React is a JavaScript library for building user interfaces.

Created by Facebook in 2013.

Current Version: 16 (v16 "react fibre" was a complete rewrite)

The DOM is created programmatically.

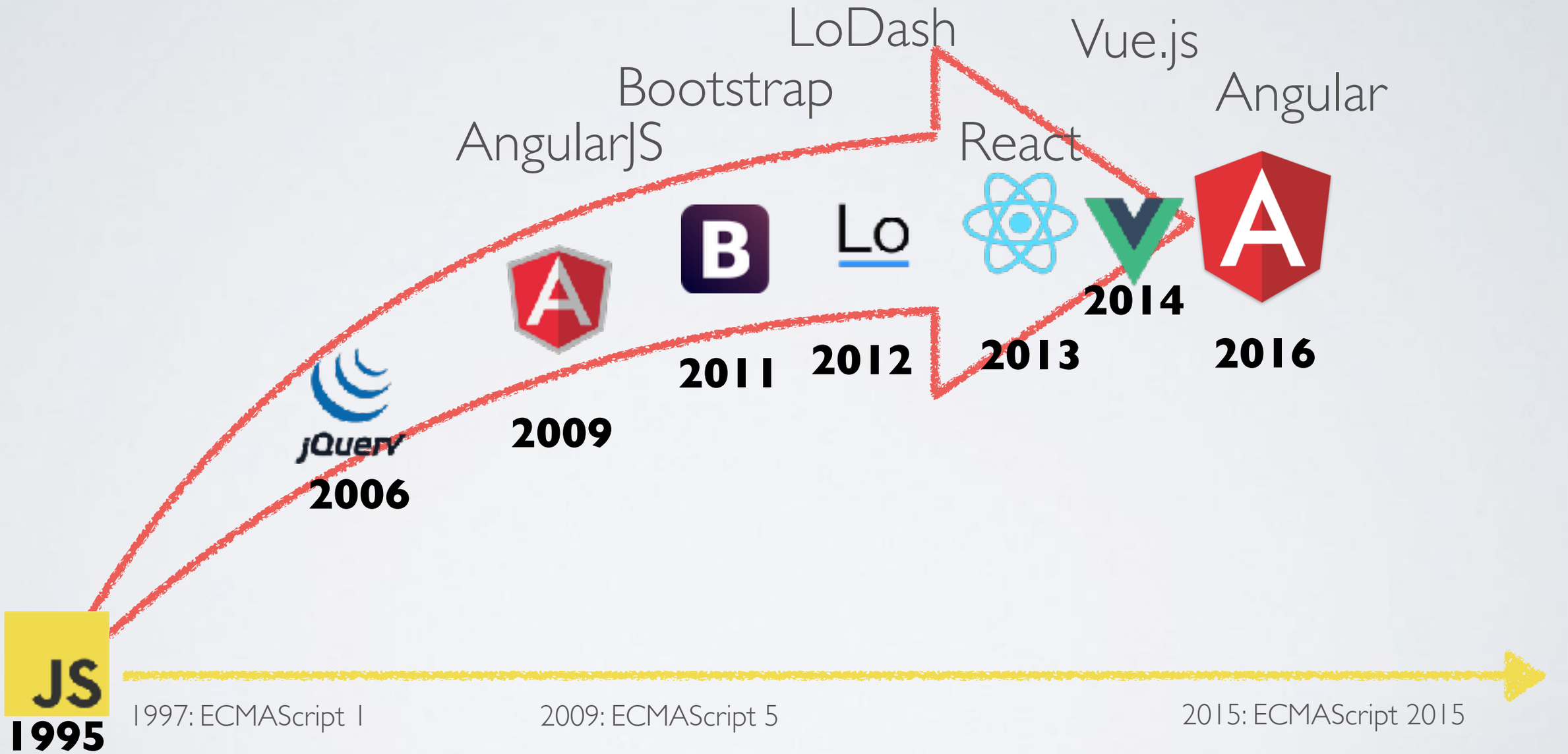
JSX enables declarative DOM programming.

A virtual DOM abstracts the real DOM.

One-way data flow instead of two-way data binding.

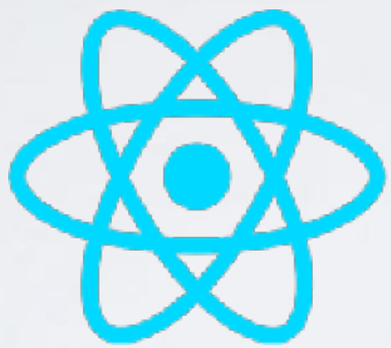
https://facebook.github.io/react/

History of JavaScript & Frameworks



The "Current Generation" of Frontend JavaScript Frameworks

2016



React

2013

Vue.js

2014 / V2: 2016

aurelia

2015

V2: 2015

- component architecture
- functional/reactive approaches
- unidirectional data-flow
- leveraging bleeding-edge technologies (ES2015, TypeScript, WebPack, npm)

React is very popular

How does React help?

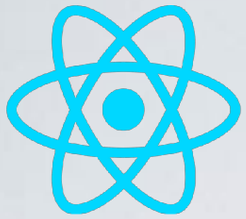
- Working with the DOM
  - declarative DOM programming instead of imperative manipulations
- "Componentization" of the UI
  - Provide Structure for the UI
  - Enable reuse of components
  - Manage state in components

Everything is a Component

React is not a Framework

- React is "only" a view library.
- React needs to be combined with other Libraries/ Frameworks to build a full Stack for building an Frontend-Application
  - Network-Access
  - Routing
  - State-Management
  - i18n
  - Components, Styling …

React vs Angular



React and Angular have different scopes:

UI-Library

- UI-Components
- Unidirectional DataFlow Architecture

Complete SPA Framework

- UI-Components
- Unidirectional DataFlow Architecture
- Routing
- Backend-Access
- Architecture:
  - Services
  - Dependency-Injection
  - Modularization

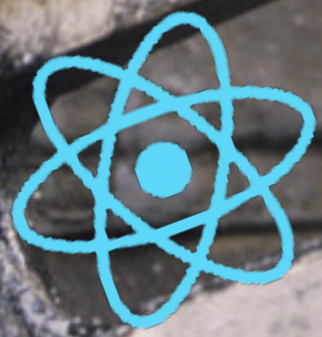
Getting started …

https://github.com/facebookincubator/create-react-app

```
npm i -g create-react-app
create-react-app awesome-app
cd awesome-app
npm start
```

Note: using yarn instead of npm is popular in the React ecosystem: https://yarnpkg.com/

EXERCISES



Exercise 1 - Create React App

Dissecting a React App

https://github.com/facebookincubator/create-react-app

```
yarn start
yarn build
yarn test
```

Project Configuration:

- package.json
- node_modules
  package-lock.json / yarn.lock

App Component:

- /src/App.js

Bootstrapping:

- public/index.html
- src/index.js
- service-worker.js

Debugging &Testing:

- source-maps
- src/App.test.js

React Project Setup

ES2015+ & JSX

Write Code

BABEL TypeScript

Compile

Babel and TypeScript can compile JSX to JavaScript

webpack
MODULE BUNDLER

Bundle

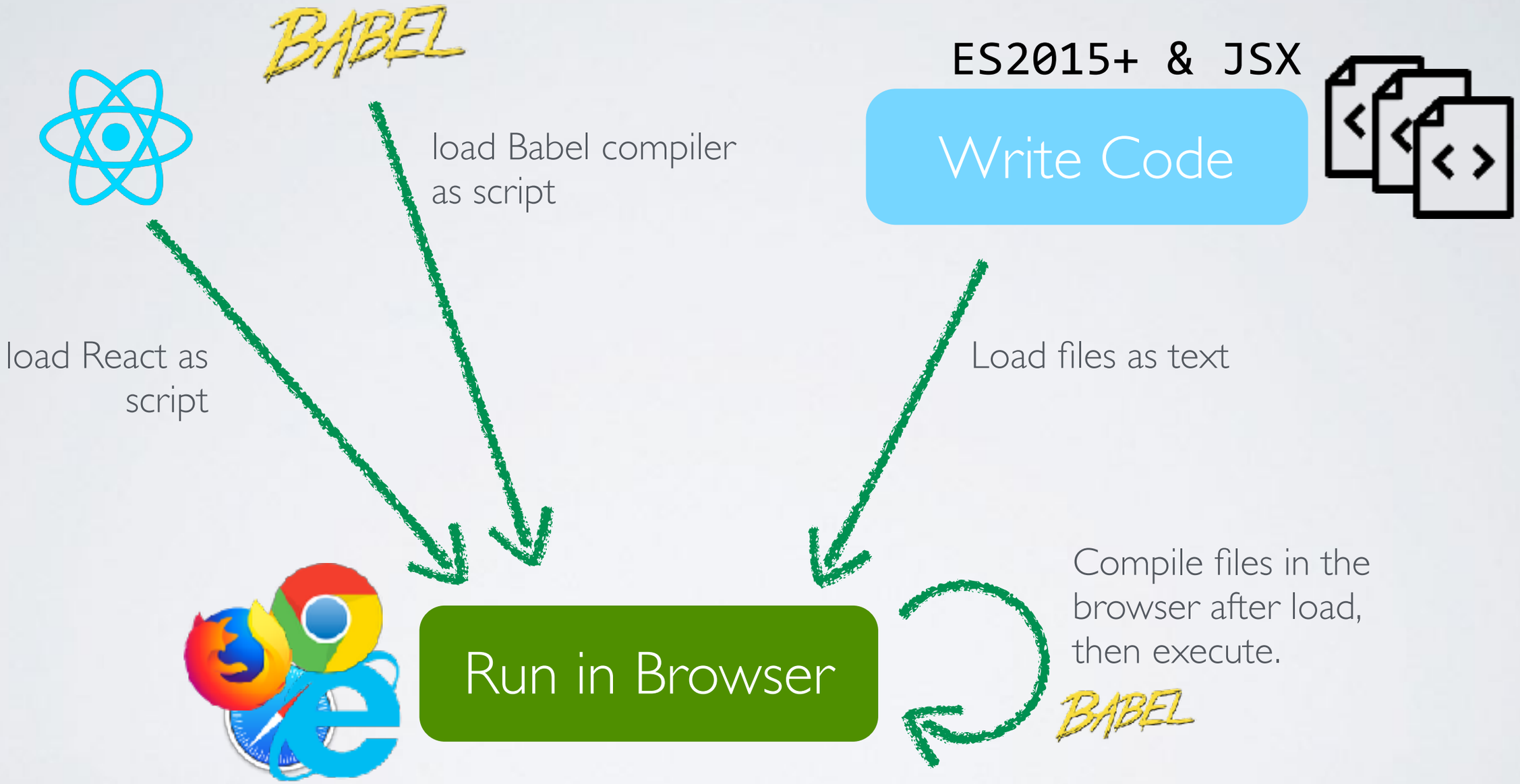
Create coarse-grained & optimised bundles from many files and 3rd-party libraries-

ES5

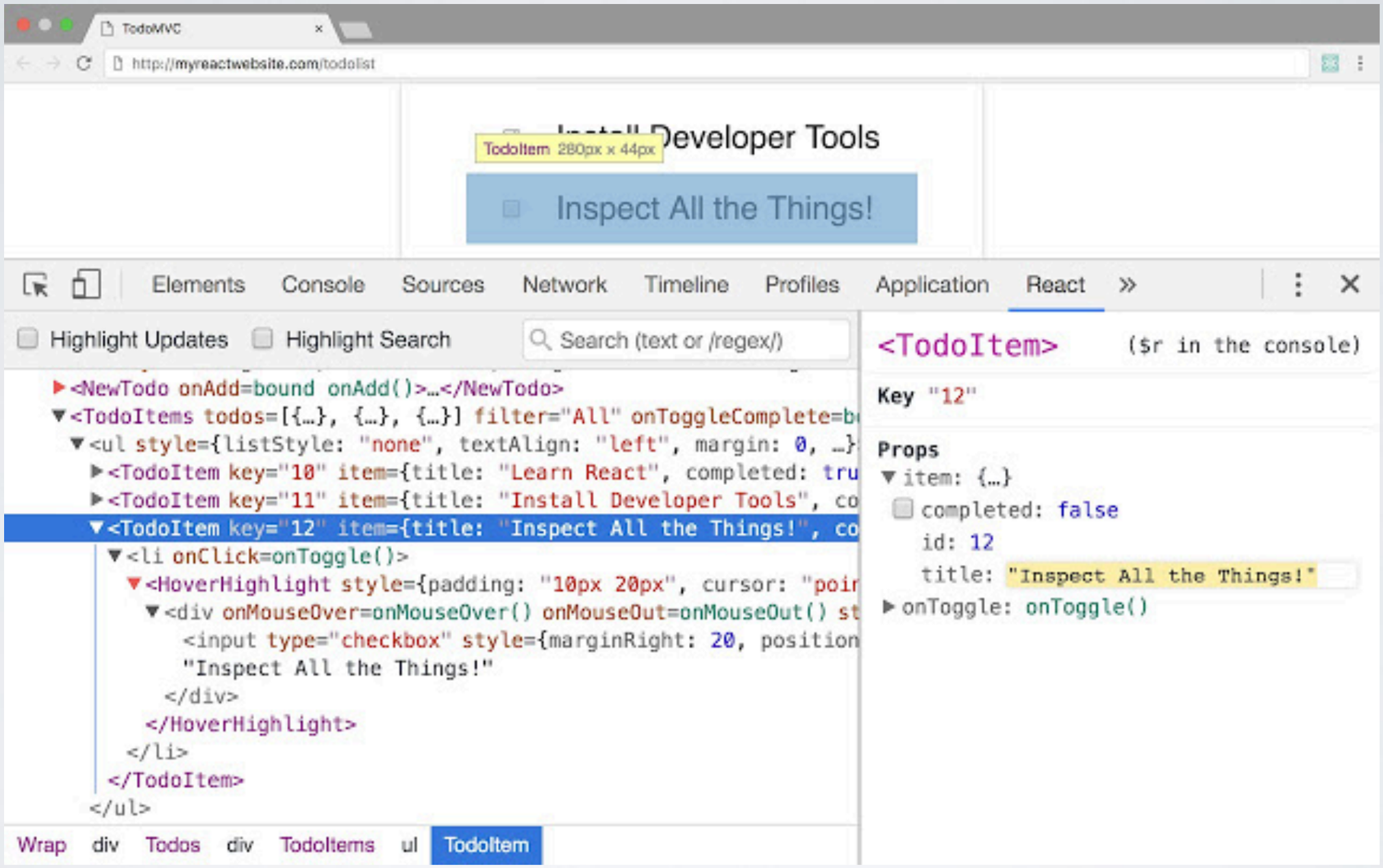
Run in Browser

Load pure JavaScript bundles.

React Demo Setup



Debugging React



React Developer Tools for Chrome and Firefox

JSX

```
const button = <button color="blue">Click Me</button>;
const div = <div> {button} </div>;
render(div);
```

JSX is a XML-like syntax extension to ECMAScript. which defines a concise and familiar syntax for defining tree structures with attributes.

With JSX we can declare HTML structures in JavaScript.

JSX is not an embedded HTML template. It gets compiled to imperative JavaScript statements which use the React API to manipulate the DOM.

Other frameworks supporting JSX:
Vue.js, Cycle.js, Preact, Inferno, Stencil …

https://facebook.github.io/jsx/

JSX

Fundamentally, JSX just provides syntactic sugar for the `React.createElement(component, props, ...children)` function.

```
function render() {
   return <div color="red">Hello World</div>;
 }
```

compilation

```
function render() {
  return React.createElement(
    "div",
    { color: "red" },
    "Hello World"
  );
}
```

Babel and TypeScript can compile JSX to JavaScript.
Try it in the Babel REPL: https://babeljs.io/repl/

https://facebook.github.io/react/docs/jsx-in-depth.html

React Components

React components are implemented as ES2015 classes.

```
class HelloMessage extends React.Component{
  render() {
    return <div>Hello {this.props.name}</div>;
  }
};
```

Depending on the setup this gets transpiled to ES5.

React 15 provided a `createClass` API for ES5. This is not supported in React 16 any more.

```
var HelloMessage = React.createClass({
  render: function() {
    return React.createElement(
      "div", null, "Hello ", this.props.name
    );
  }
});
```

HTML in JavaScript?

- Angular enhances HTML with a new language
  ("To read Angular: Learn a long list of Angular specific syntax")
- React enhances JavaScript in order to declare the UI ("To read React: Learn JavaScript")
  - Leverage the power of JavaScript
  - It makes sense to keep the component and the template in the same file, since they are highly coupled
  - build-time syntax checking and error messages
  - "Separation of concerns" is still possible

https://medium.com/javascript-scene/jsx-looks-like-an-abomination-1c1ec351a918#.xwa2wfc2y
https://medium.com/@housecor/react-s-jsx-the-other-side-of-the-coin-2ace7ab62b98
https://medium.freecodecamp.com/angular-2-versus-react-there-will-be-blood-66595faafd51

Rendering a React Application

React does not render components, rendering is separated into the **`react-dom`** package:

```
ReactDOM.render(
    <App/>,
    document.getElementById('root')
);
```

React Native provides other renderers for iOS & Android.
Server-Side Rendering of React renders into HTML-Text.

The Virtual DOM

Components

App

Title

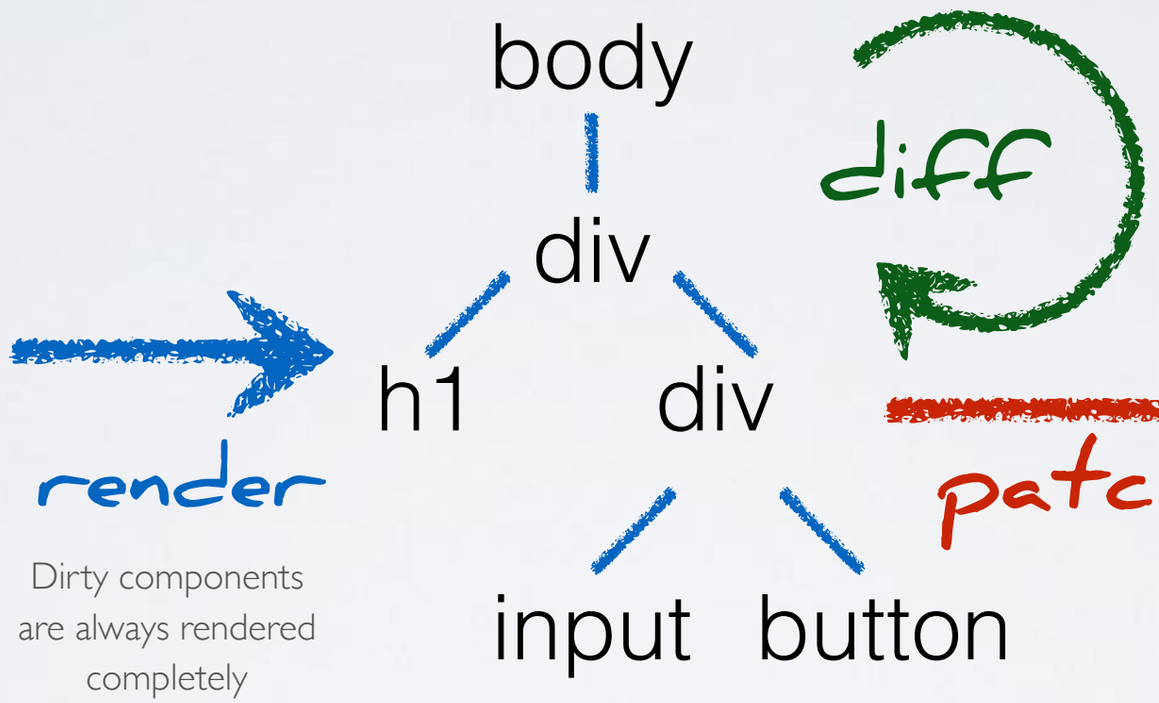
Content

input

button

Virtual DOM

In-Memory, implemented in JavaScript



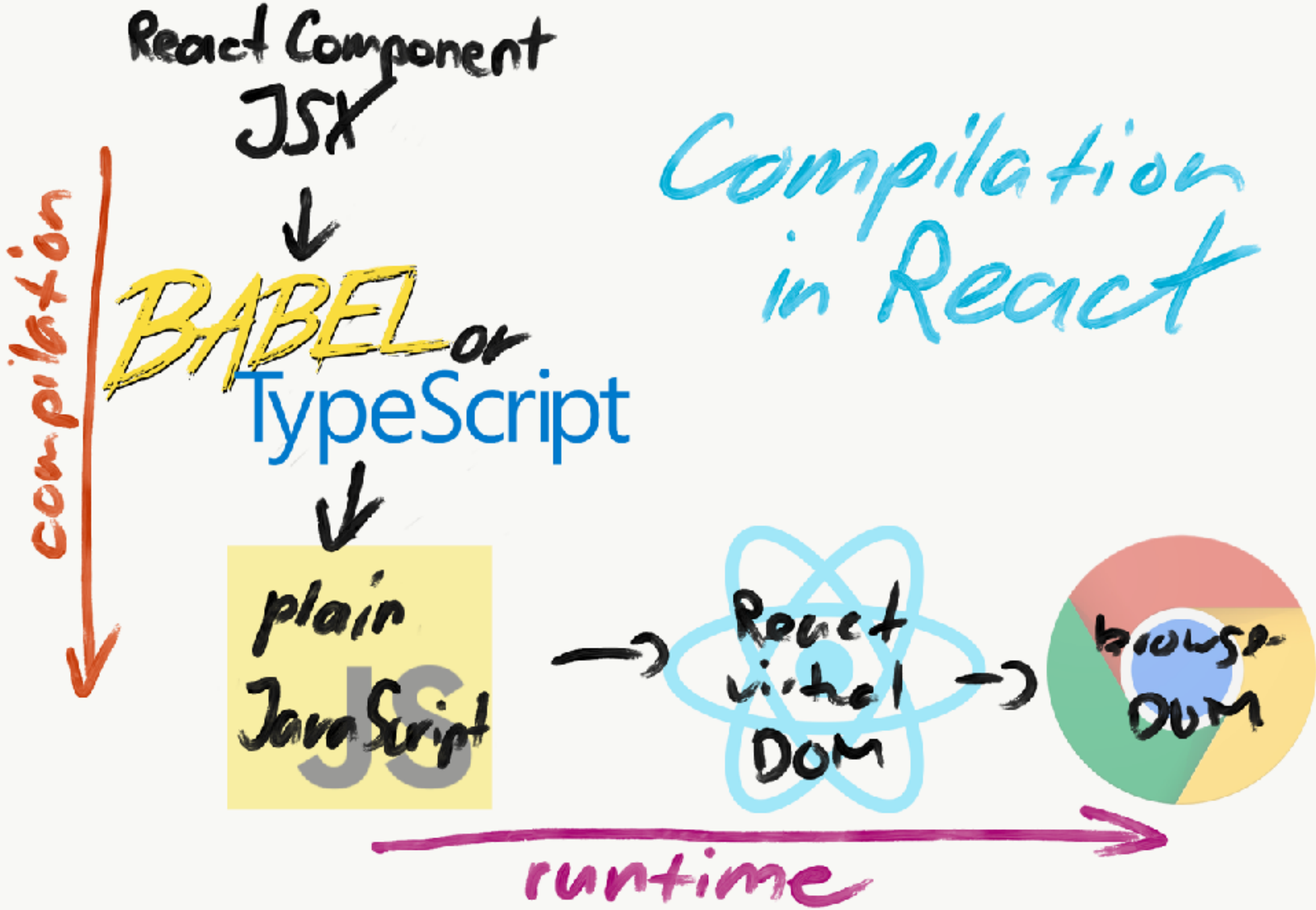
Dirty components are always rendered completely

Browser DOM

```
<body>
  <div>
    <h1></h1>
    <div>
      <input/>
      <button>
        </button>
    </div>
  </div>
</body>
```

Virtual DOM also enables server-side rendering ("isomorphic javascript","universal javascript") or rendering to iOS/Android UIs.

Compilation in React



React Components

A Simple Component

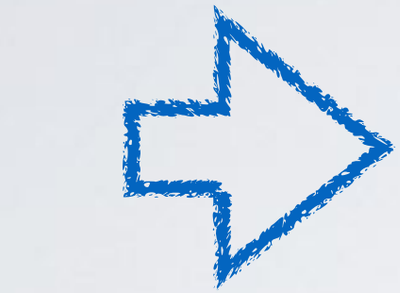
```
import React from 'react';

class Greeter extends React.Component {
    render() {
        return <h1>Hello World!</h1>;
    }
}

export default Greeter;
```

The class name of a React components has to start with a uppercase letter.

A React Component



properties

properties represent data
owned by someone else
and should not be changed

```
class HelloMessage extends React.Component{
  ...
  render() {
    return <div>
              Hello {this.props.props}
                 + {this.state.name}
           </div>;
  }
};
```

state

html

components can be stateful

state is managed and
changed by the component
itself

If properties or state change, the component is (re-)rendered

https://facebook.github.io/react/docs/reusable-components.html

Event Handling

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

class ClickMe extends Component {
    render() {
        return (
            <button onClick={(e) => this.handleClick(e)}>
                Click me!
            </button>
        );
    }

    handleClick(event) {
        alert('Event: ' + event.type);
    }
}
export default ClickMe;
```

Component State

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

class Counter extends Component {

    constructor() {
        super();
        this.state = {count: 1};
    }

    render() {
        return (
            <div>
                <h1>Count {this.state.count}</h1>
                <button onClick={() => this.handleClick()}>
                    Click me!
                </button>
            </div>
        );
    }

    handleClick() {
        this.setState({count: this.state.count + 1})
    }
}

export default Counter;
```

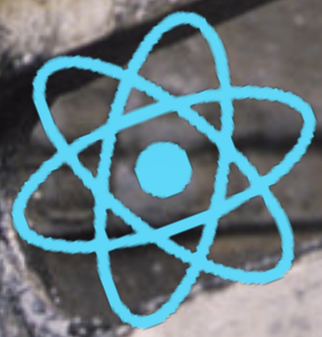
State is set up in the constructor by setting the **`state`** property.

State can be rendered in JSX

State-changes must be set via **`this.setState`**

- the passed object is merged with the current state
- it triggers a re-rendering

EXERCISES



Exercise 2 - Write your first component

Immutability

props & state should be treated as immutable

- props are owned by a parent and should only be changed by this parent ("unidirectional data-flow")
- state changes are easier to track if state is immutable

```
let newTodos = [...this.state.todos, newToDo];
this.setState({todos: newTodos});
```

Libraries: immutability-helper https://github.com/kolodny/immutability-helper
Immutable.js https://facebook.github.io/immutable-js/

https://medium.com/@housecor/handling-state-in-react-four-immutable-approaches-to-consider-d1f5c00249d5

Functional State Changes

Calls to **`setState`** can be batched, state changes may be asynchronous.

setState also accepts a function:

```
setState( (oldState, props) => "changes" )
```

This code does not work:

```
increaseScore () {
 this.setState({score : this.state.score + 1});
 this.setState({score : this.state.score + 1});
}
```

This code does work:

```
increaseScore () {
    this.setState( (state) => ({score : state.score + 1}) ),
    this.setState( (state) => ({score : state.score + 1}) ),
 }
```

https://facebook.github.io/react/docs/state-and-lifecycle.html
https://medium.freecodecamp.org/functional-setstate-is-the-future-of-react-374f30401b6b
https://www.robinwieruch.de/learn-react-before-using-redux/

React Events

React provides a synthetic event system:

Event handlers are registered with React components and not on DOM elements. React listens to all DOM elements and calls the appropriate handlers.

React events work like DOM events: Inner components can trigger events and outer components can handle them.

```
class Parent extends React.Component{
  ...
  render() {
    return <Child
               onInnerClick={handler}>
           </Child>;
  }
};
```

```
class Child extends React.Component{
  ...
  render() {
    return <button
               onClick={this.props.onInnerClick}>
               Click Me 1
           </button>;
  }
};
```

https://facebook.github.io/react/docs/handling-events.html

Stateless Functional Components

Stateless components can be written as plain JavaScript functions.


```
const AppComponent = props => (
    <div>
        <h1>{props.title}</h1>
        <p>{props.message}</p>
    </div>
);
```

https://facebook.github.io/react/docs/reusable-components.html

Collections

In JavaScript you can transform arrays with **`map`**:

```
const a = [1,2,3];
const b = a.map(e => e * e); // b = [1,4,9]
```

In JSX expressions can produce dynamic "embedded" JSX:

```
<ul>
   {
      this.state.myArray.map(
         (e, index) => <li>Item {index} {e} </li>
      );
   }
</ul>
```

React Lifecycle

A component can provide hooks that are called by React during the lifecycle of the component:

```
componentWillMount
componentDidMount
componentWillReceiveProps
componentWillUpdate
componentDidUpdate
componentWillUnmount
```

https://facebook.github.io/react/docs/component-specs.html
https://engineering.musefind.com/react-lifecycle-methods-how-and-when-to-use-them-2111a1b692b1

Default Props & Prop Types

```
npm install prop-types
```

```
import {PropTypes} from 'prop-types';

class AppComponent extends React.Component {
  ...
}
AppComponent.propTypes = {
    title: PropTypes.string.isRequired,
    message: PropTypes.string.isRequired,
    onInnerClick: PropTypes.func.isRequired
};
AppComponent.defaultProps = {
    message: 'Hello',
    onInnerClick: () => alert('Hoppala!!!')
};
```

EXERCISES

Exercise 3 & 4- Components

DOM Access with Refs

```
class GreeterComponent extends React.Component {
  render() {
    return (
      <div>
        <button ref={domElement => this.button1 = domElement}>
          Button 1
        </button>
      </div>
    );
  }
  componentDidMount() {
    this.button1.focus();
  }
}
```

The **`ref`** attribute takes a callback function which is executed immediately after the component is mounted. The argument for the callback is the underlying DOM element.

https://reactjs.org/docs/refs-and-the-dom.html

Forms

There is no two-way databinding in React.

controlled components:

- state is managed by react
- value and onChange must be set

```
<input
    name="numberOfGuests"
    type="number"
    value={this.state.numberOfGuests}
    onChange={(e) => this.handleInputChange(e)} />
```

```
handleInputChange(event) {
    const target = event.target;
    const name = target.name;
    this.setState({
        [name]: target.value
    });
}
```

uncontrolled components:

- value must not be set
- use ref to access value from DOM element

```
<input
    ref={elem => this.numberOfGuests = elem}
    type="number"
    defaultValue={this.props.initialNumber}
/>
```

```
submitForm(e) {
    e.preventDefault();
    console.log(this.numberOfGuests.value);
};
```

Binding Class Methods

- No "implicit" `this` binding. (reasoning: idiomatic javascript)
- Different solutions:
  - arrow functions or explicit bind in render method
  - bind in constructor
  - stage 3 class fields & arrow functions
  - https://github.com/andreypopp/autobind-decorator

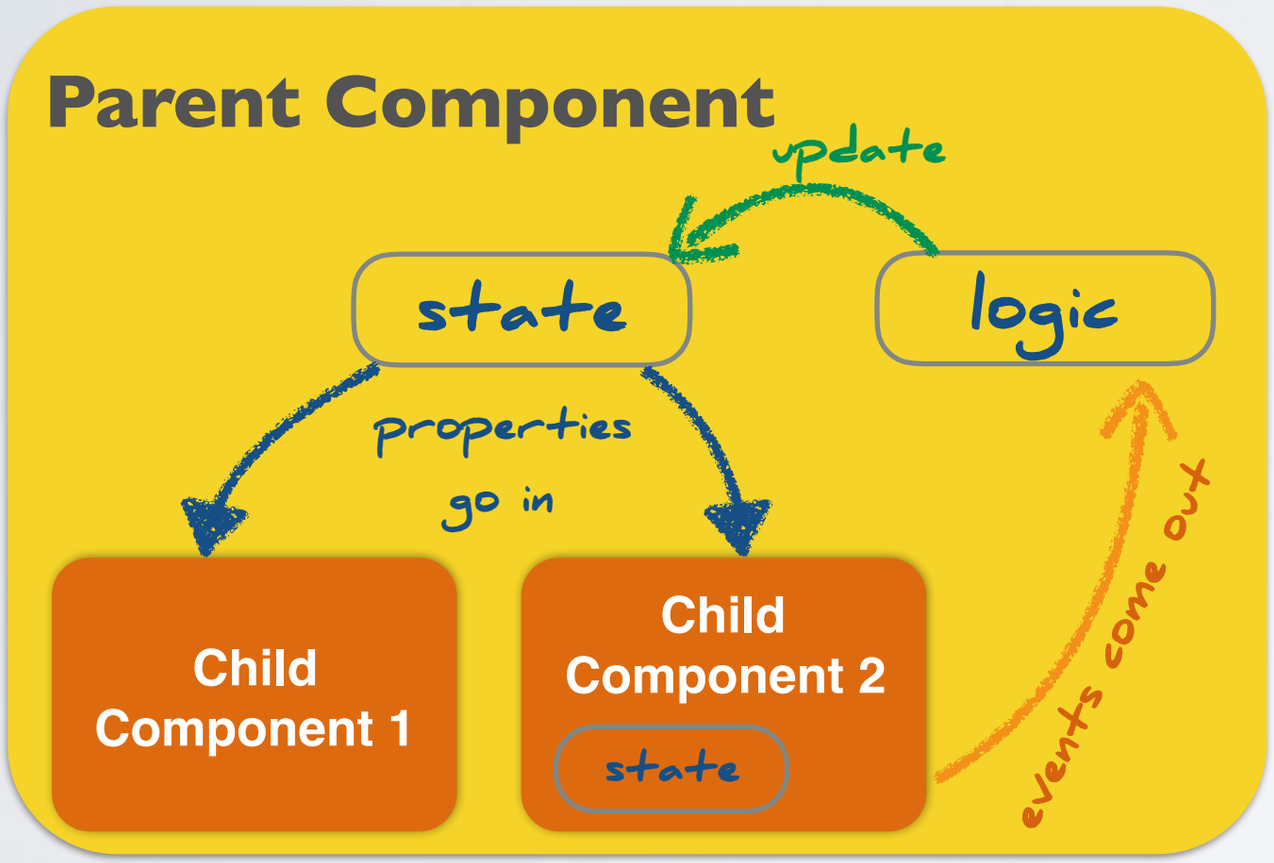
For "normal" applications there should not be a performance implication of any of these approaches:

https://twitter.com/bmeurer/status/938060007546945536
https://cdb.reacttraining.com/react-inline-functions-and-performance-bdff784f5578

https://medium.freecodecamp.com/react-binding-patterns-5-approaches-for-handling-this-92c651b5af56

Nested Components: Data-Flow

State should be explicitly owned by a component.



- A parent component passes state to children
- Children should not edit state of their parent
- Children "notify" parents (events, actions …)

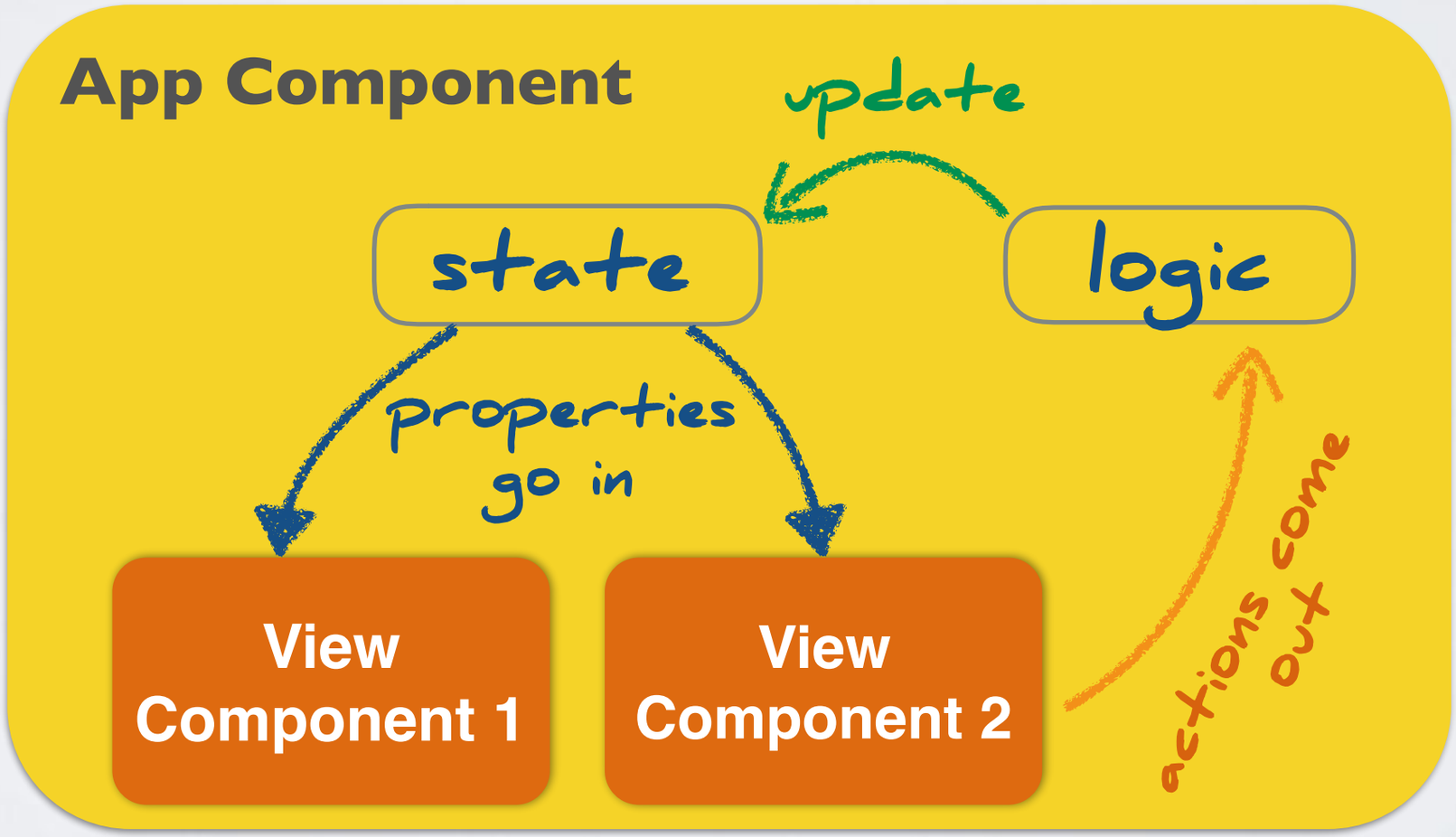
React formalises unidirectional data-flow via properties an events.

Unidirectional Data-Flow

State flows down the component tree / Actions flow up the component tree.

State can be passed to child components. Child components are not allowed to change the state. When the parent state is changed, the children are re-rendered.

No two-way databinding!



http://techblog.constantcontact.com/software-development/reactive-component-based-uis/

Data flows down / Events Flow Up

Passing props from parent to child:

```
class Parent extends React.Component{
  ...
  render() {
    return <Child
                message={message}
                onInnerClick={handler}>
           </Child>;
  }
};
```

Using props and emitting events:

```
class Child extends React.Component{
  ...
  render() {
    return (
      <div>
        <div>{this.props.message}</div>
        <button
          onClick={this.props.onInnerClick}>
          Click Me
        </button>;
      </div>
  }
};
```

This programming model is very similar to the native DOM APIs.

Container vs. Presentation Components

Application should be decomposed in container- and presentation components:

Container	**Presentation**
Little to no markup	Mostly markup
Pass data and actions down	Receive data & actions via props
typically stateful / manage state	mostly stateless
	better reusability

aka: Smart- vs. Dumb Components

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Separation of Concerns

Separation of concerns is not equal to separation of file types!

Keep things together that change together.

You can split a component into a controller and a view:

```
import View from './View';

export class GiphySearch extends Component {
   … // controller logic & state
   render(){
        return <View data={...}
   }
}
```

GiphySearch.js

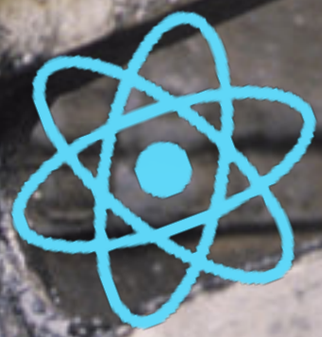
```
const View = ({data}) => (
   <div>
        {data.message}
   </div>
);
export default View;
```

View.js

https://codesandbox.io/s/NxqMqyxlD

https://medium.com/styled-components/component-folder-pattern-ee42df37ec68

EXERCISES



Exercise 5 & 6 - ToDo App