FRONT END

Level UP

POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

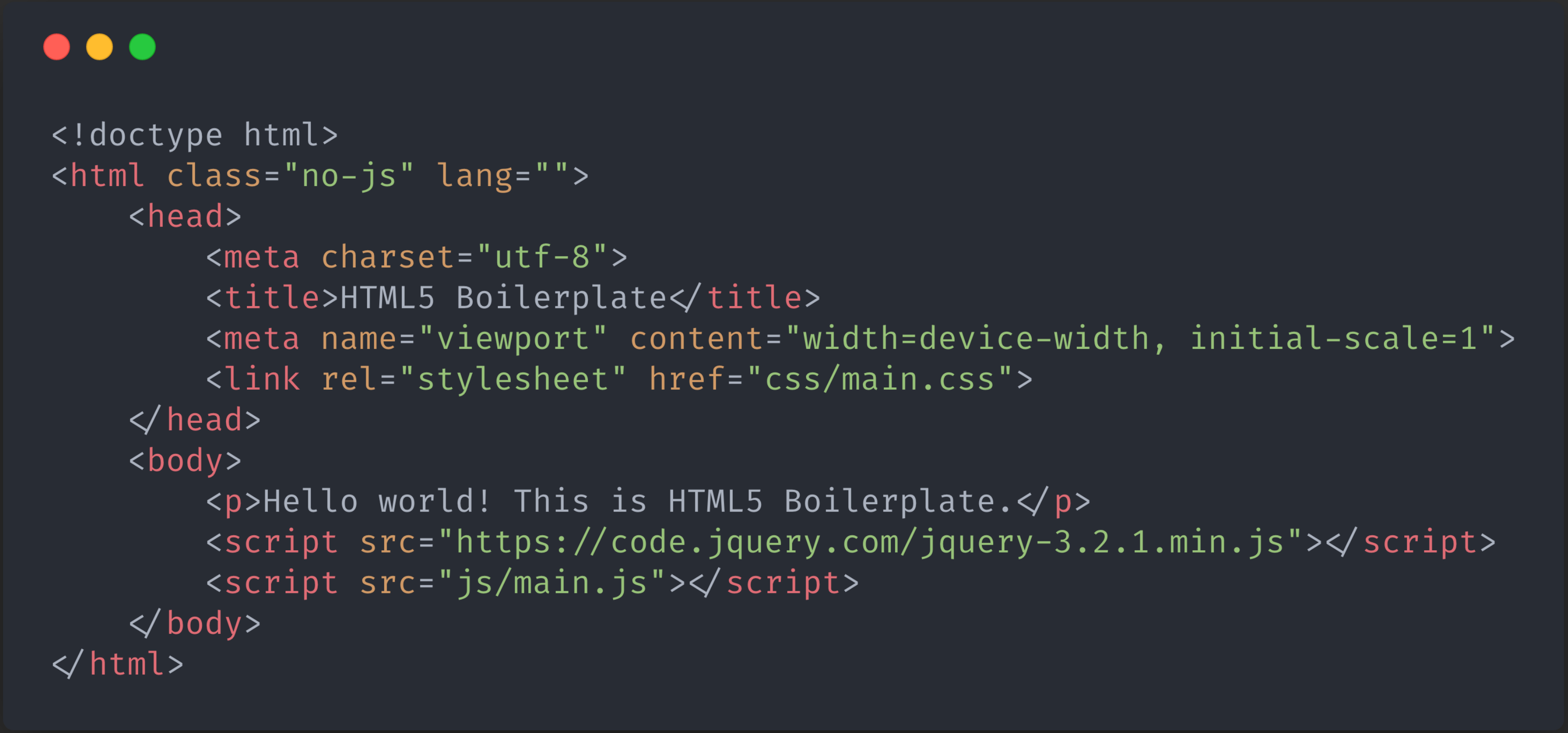
daftcode / spring 2018

JS Frameworks overview

Let's try React!

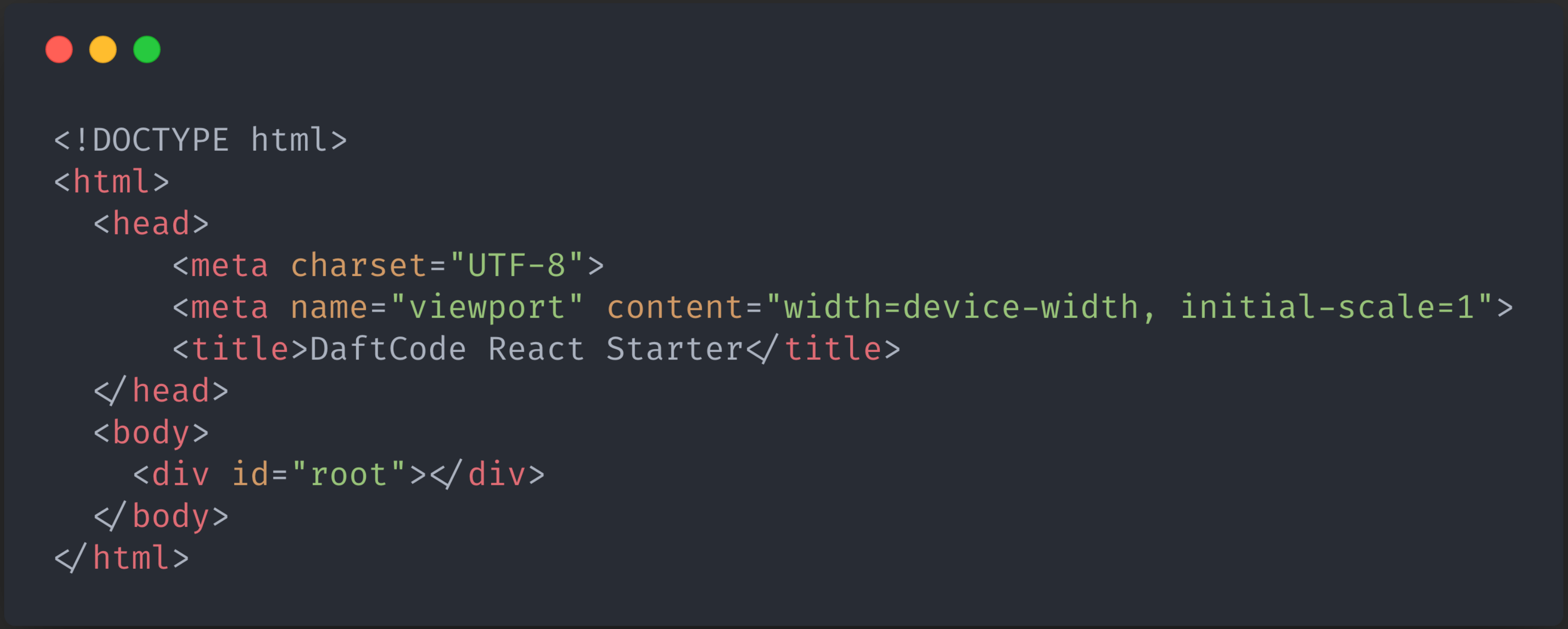
Mariusz Kaczkowski

JQuery way

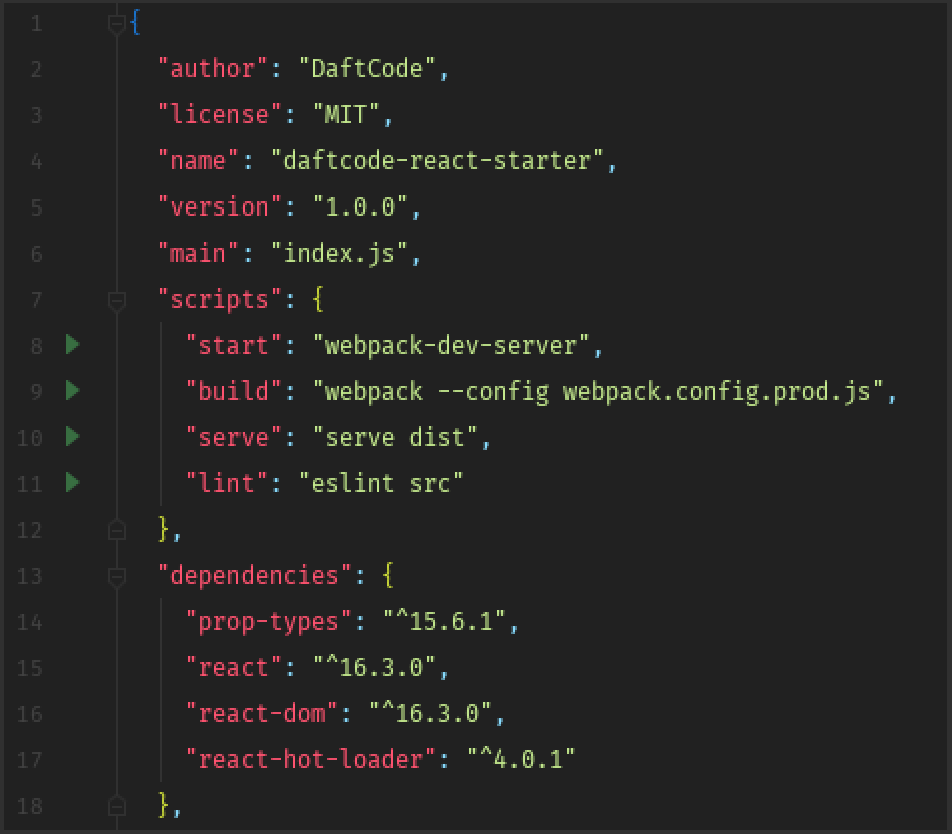


```html
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <title>HTML5 Boilerplate</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="css/main.css">
    </head>
    <body>
        <p>Hello world! This is HTML5 Boilerplate.</p>
        <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
        <script src="js/main.js"></script>
    </body>
</html>
```

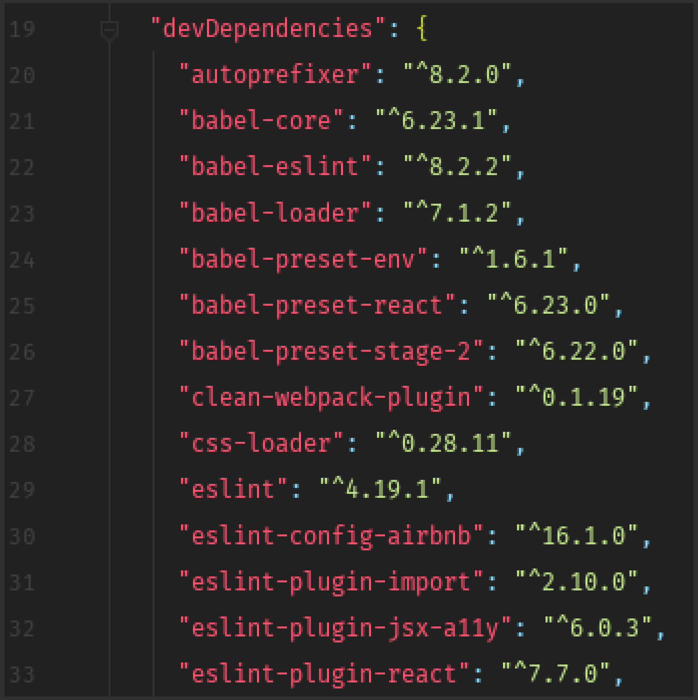
Modern approach



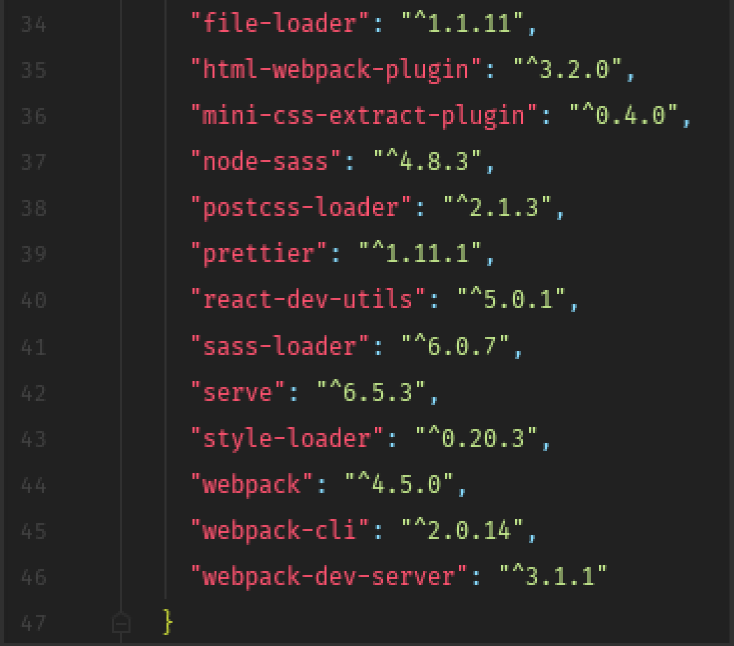
```html
<!DOCTYPE html>
<html>
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <title>DaftCode React Starter</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```



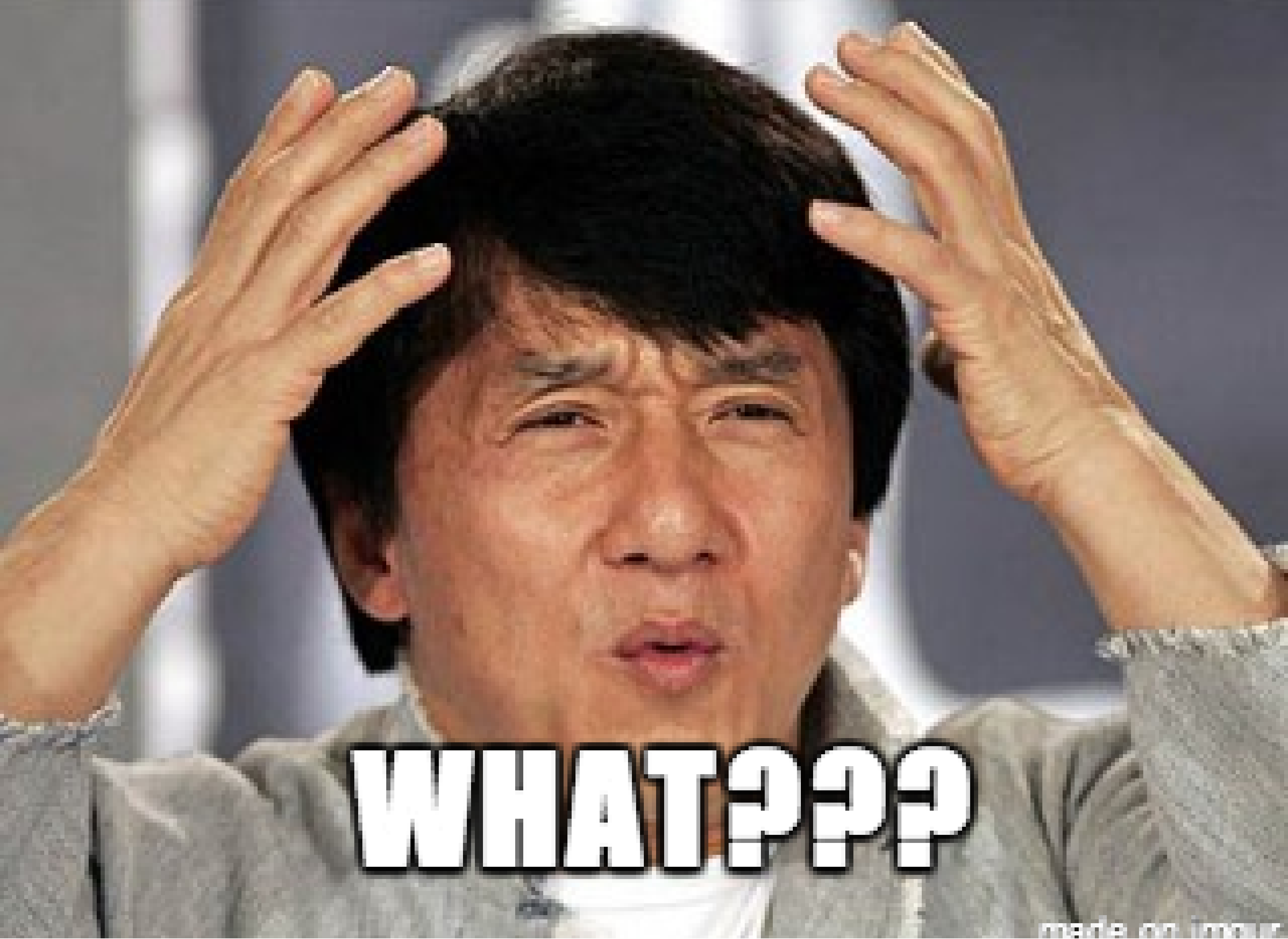
```json
{
  "author": "DaftCode",
  "license": "MIT",
  "name": "daftcode-react-starter",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "webpack-dev-server",
    "build": "webpack --config webpack.config.prod.js",
    "serve": "serve dist",
    "lint": "eslint src"
  },
  "dependencies": {
    "prop-types": "^15.6.1",
    "react": "^16.3.0",
    "react-dom": "^16.3.0",
    "react-hot-loader": "^4.0.1"
  },
```



```json
  "devDependencies": {
    "autoprefixer": "^8.2.0",
    "babel-core": "^6.23.1",
    "babel-eslint": "^8.2.2",
    "babel-loader": "^7.1.2",
    "babel-preset-env": "^1.6.1",
    "babel-preset-react": "^6.23.0",
    "babel-preset-stage-2": "^6.22.0",
    "clean-webpack-plugin": "^0.1.19",
    "css-loader": "^0.28.11",
    "eslint": "^4.19.1",
    "eslint-config-airbnb": "^16.1.0",
    "eslint-plugin-import": "^2.10.0",
    "eslint-plugin-jsx-a11y": "^6.0.3",
    "eslint-plugin-react": "^7.7.0",
```



```
    "file-loader": "^1.1.11",
    "html-webpack-plugin": "^3.2.0",
    "mini-css-extract-plugin": "^0.4.0",
    "node-sass": "^4.8.3",
    "postcss-loader": "^2.1.3",
    "prettier": "^1.11.1",
    "react-dev-utils": "^5.0.1",
    "sass-loader": "^6.0.7",
    "serve": "^6.5.3",
    "style-loader": "^0.20.3",
    "webpack": "^4.5.0",
    "webpack-cli": "^2.0.14",
    "webpack-dev-server": "^3.1.1"
  }
```


WHAT???
made on imgur

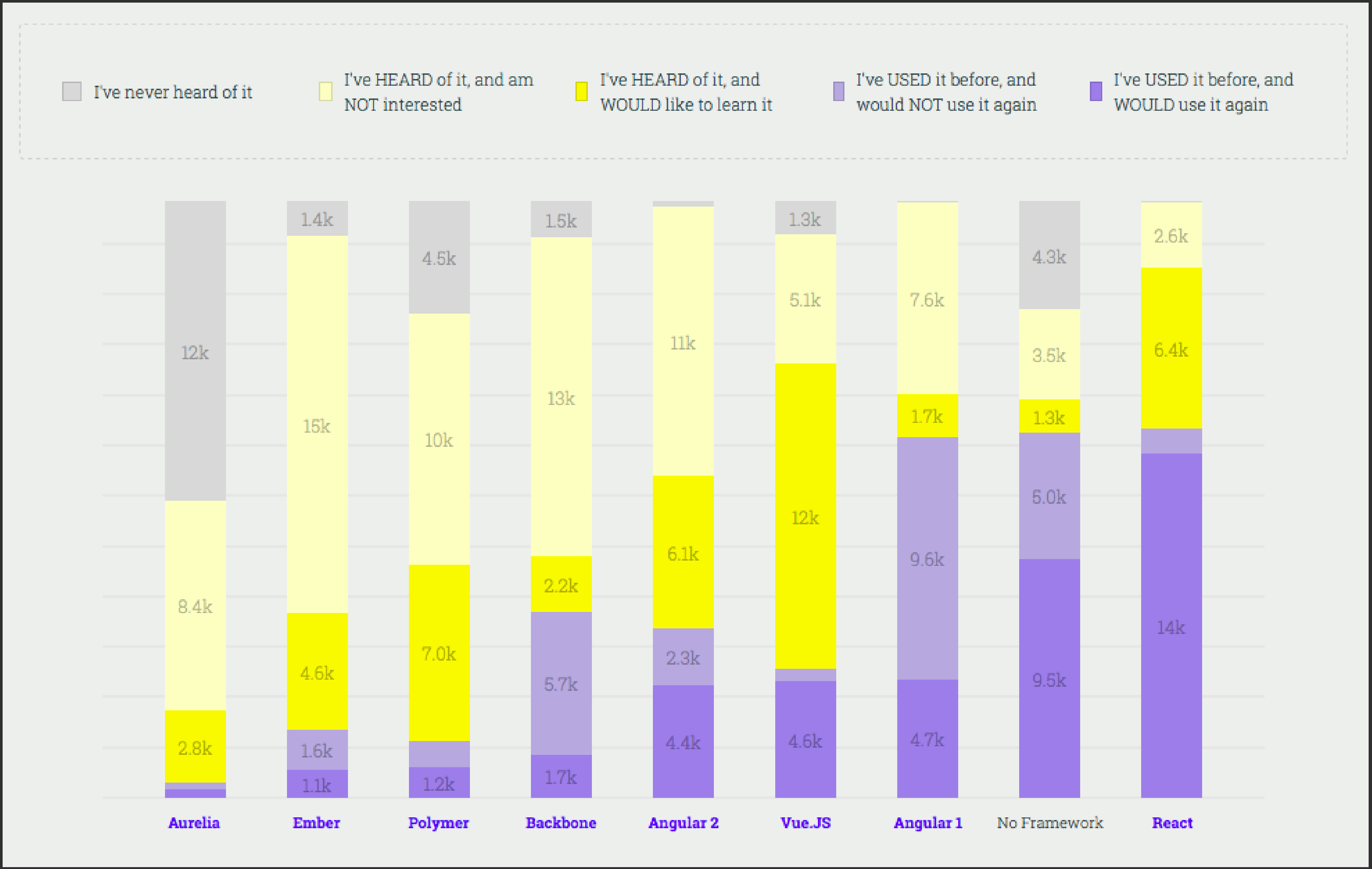
When we really need it?

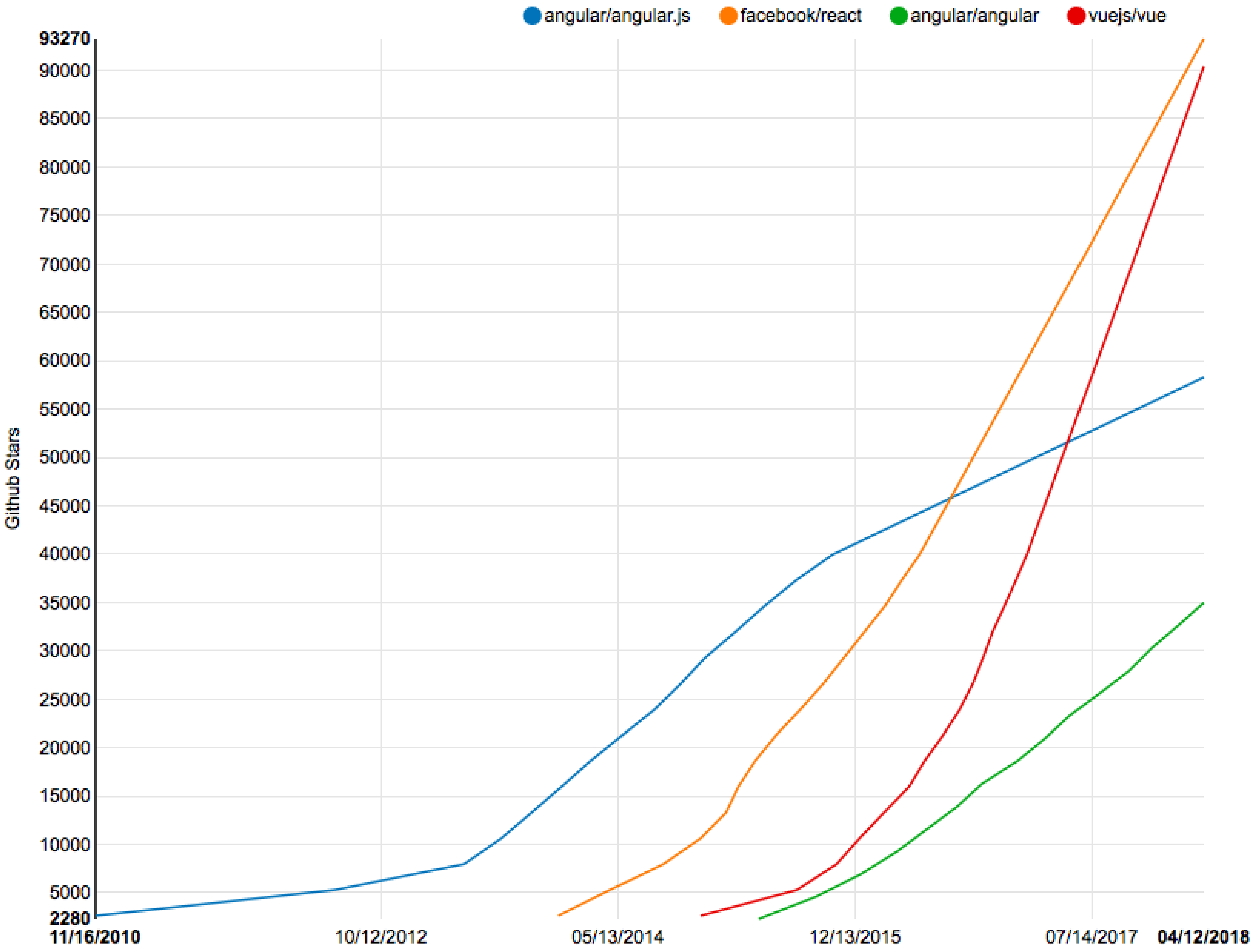
To build large apps with data changes over time

What we get with framework?

- Development Speed
- Best practices
- Support
- Cost

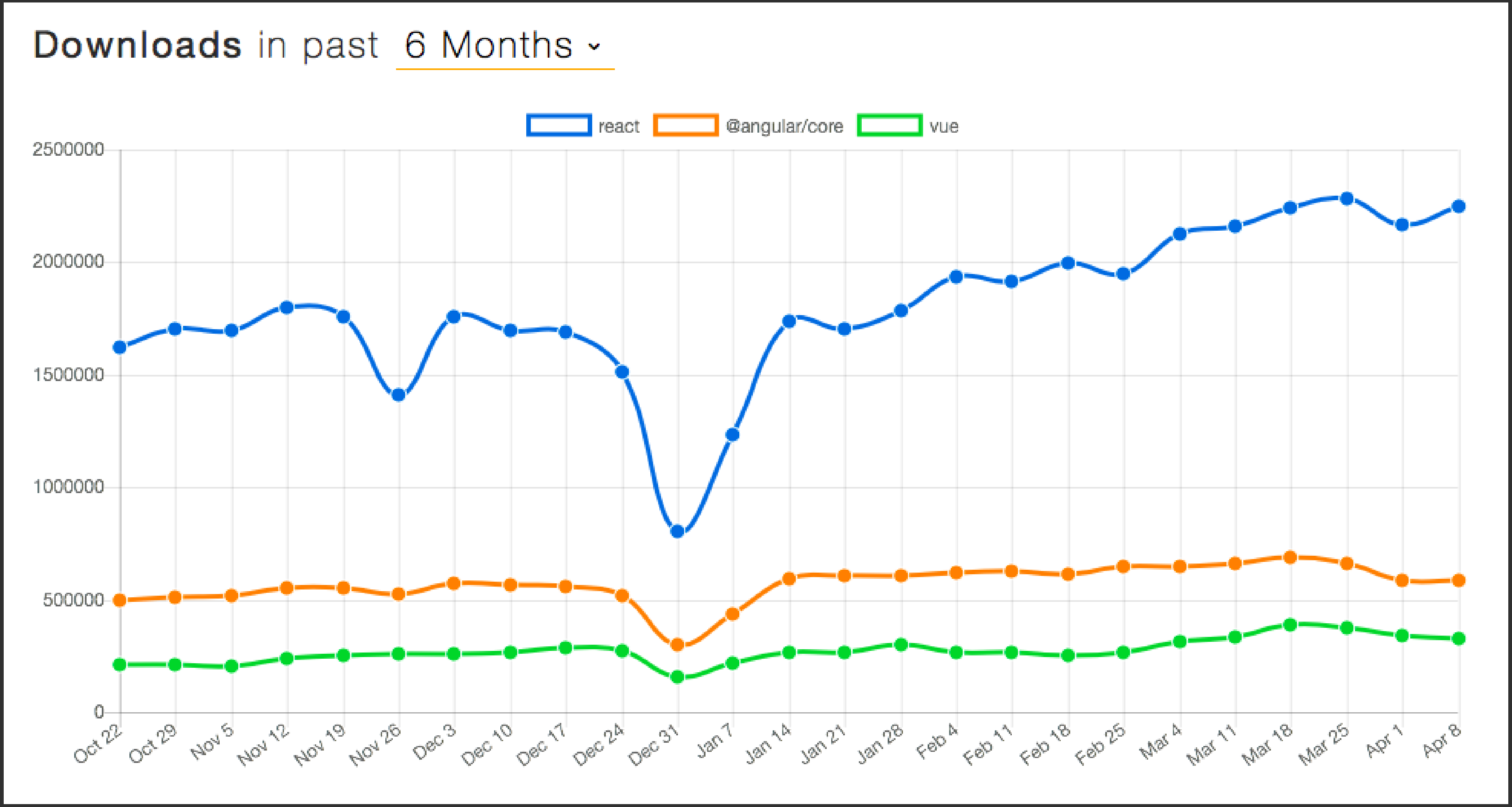
Popularity (2017)

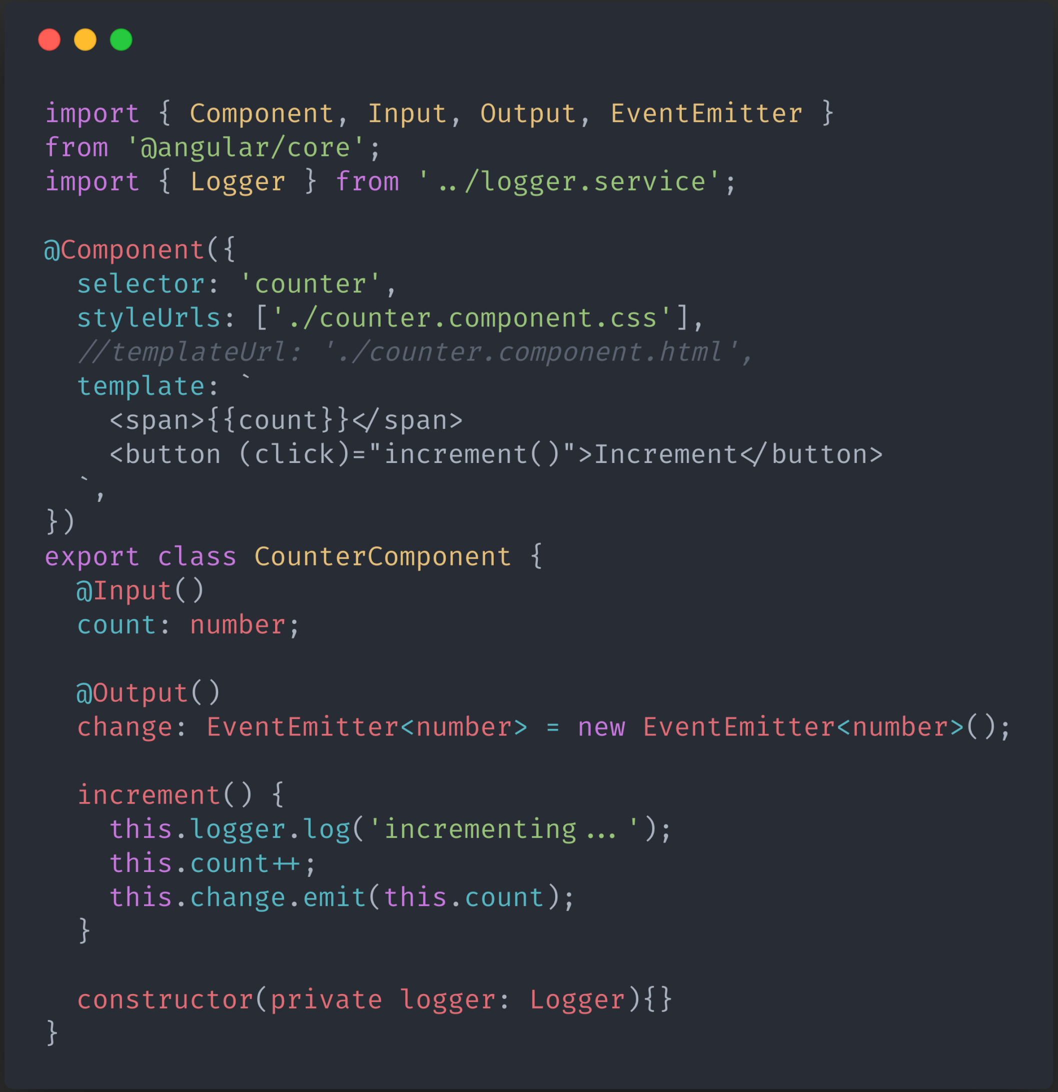




● angular/angular.js
● facebook/react
● angular/angular
● vuejs/vue
Github Stars
93270
90000
85000
80000
75000
70000
65000
60000
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
2280
11/16/2010
10/12/2012
05/13/2014
12/13/2015
07/14/2017
04/12/2018

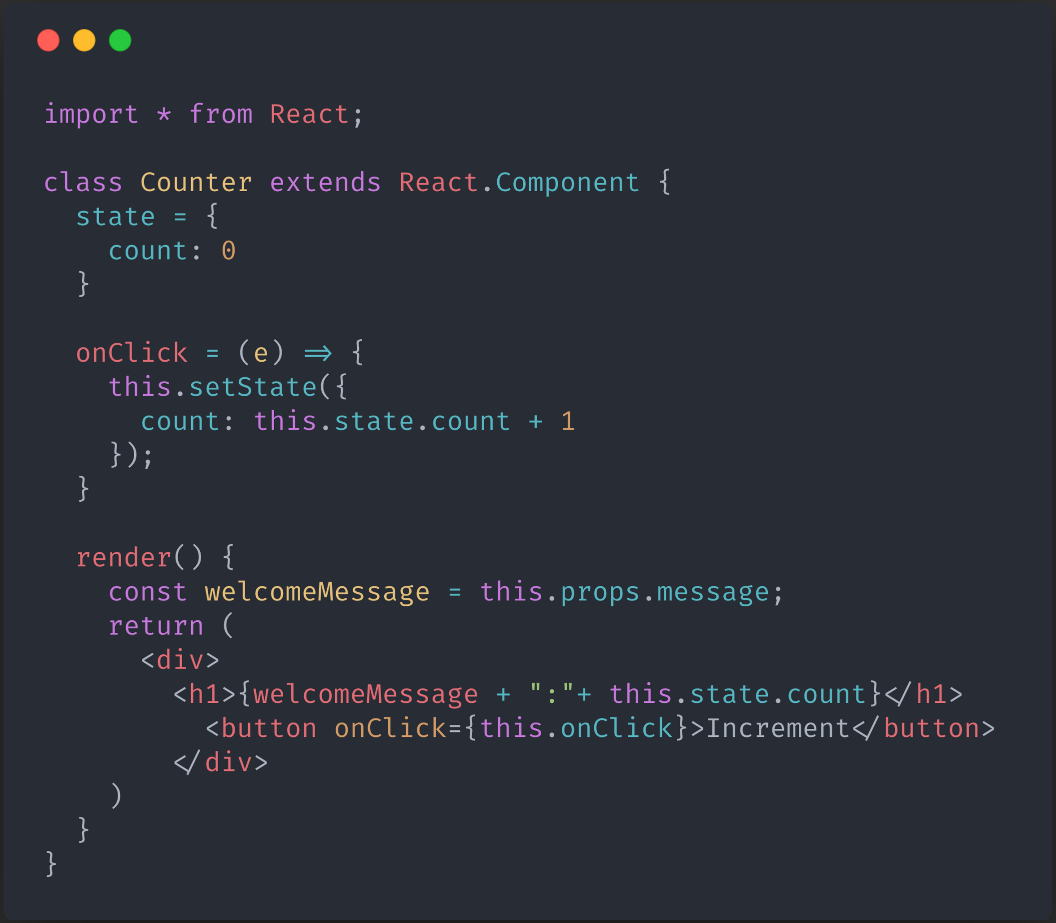
NPM downloads





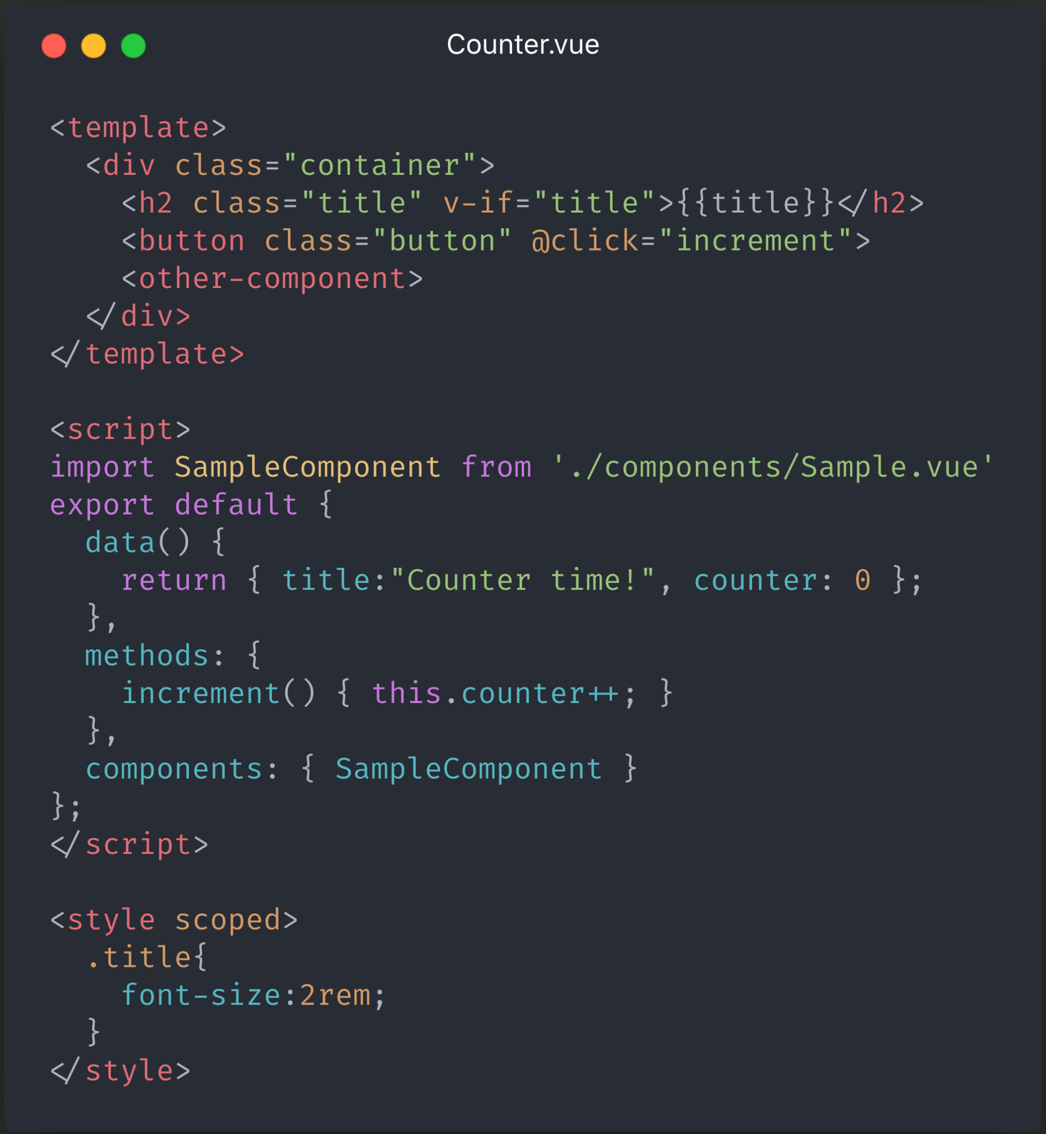
```
import { Component, Input, Output, EventEmitter }
from '@angular/core';
import { Logger } from '../logger.service';

@Component({
  selector: 'counter',
  styleUrls: ['./counter.component.css'],
  //templateUrl: './counter.component.html',
  template: `
    <span>{{count}}</span>
    <button (click)="increment()">Increment</button>
  `,
})
export class CounterComponent {
  @Input()
  count: number;

  @Output()
  change: EventEmitter<number> = new EventEmitter<number>();

  increment() {
    this.logger.log('incrementing...');
    this.count++;
    this.change.emit(this.count);
  }

  constructor(private logger: Logger){}
}
```

- 2009/2016 Google
- TypeScript (OOP)
- MVC
- Framework
- angular-cli
- NativeScript
- 143kb



```
import * from React;

class Counter extends React.Component {
  state = {
    count: 0
  }

  onClick = (e) => {
    this.setState({
      count: this.state.count + 1
    });
  }

  render() {
    const welcomeMessage = this.props.message;
    return (
      <div>
        <h1>{welcomeMessage + ":"+ this.state.count}</h1>
          <button onClick={this.onClick}>Increment</button>
        </div>
    )
  }
}
```

- 2013 Facebook
- ES6 + Babel
- View library / JSX
- Virtual DOM
- one-directional
- create-react-app
- React Native
- 43kb



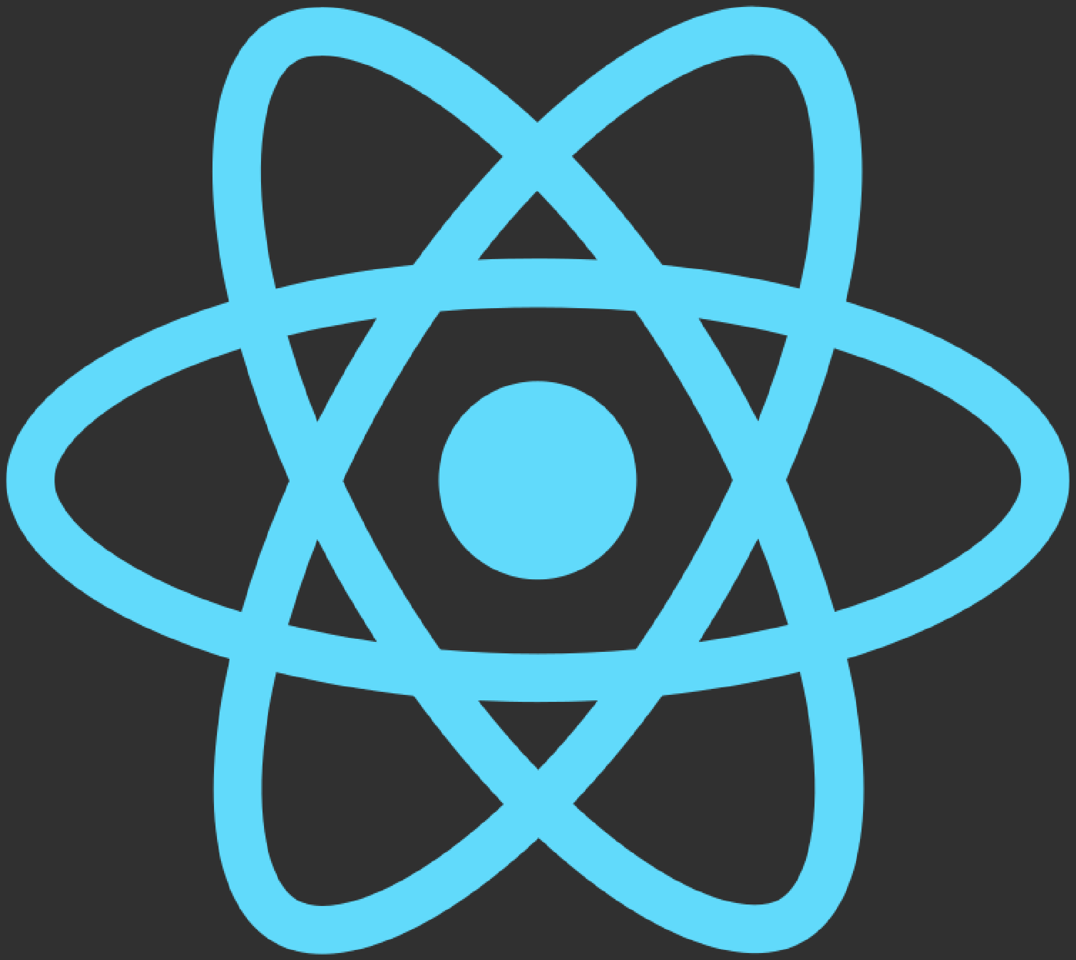
Counter.vue

```
<template>
  <div class="container">
    <h2 class="title" v-if="title">{{title}}</h2>
    <button class="button" @click="increment">
    <other-component>
  </div>
</template>

<script>
import SampleComponent from './components/Sample.vue'
export default {
  data() {
    return { title:"Counter time!", counter: 0 };
  },
  methods: {
    increment() { this.counter++; }
  },
  components: { SampleComponent }
};
</script>

<style scoped>
  .title{
    font-size:2rem;
  }
</style>
```

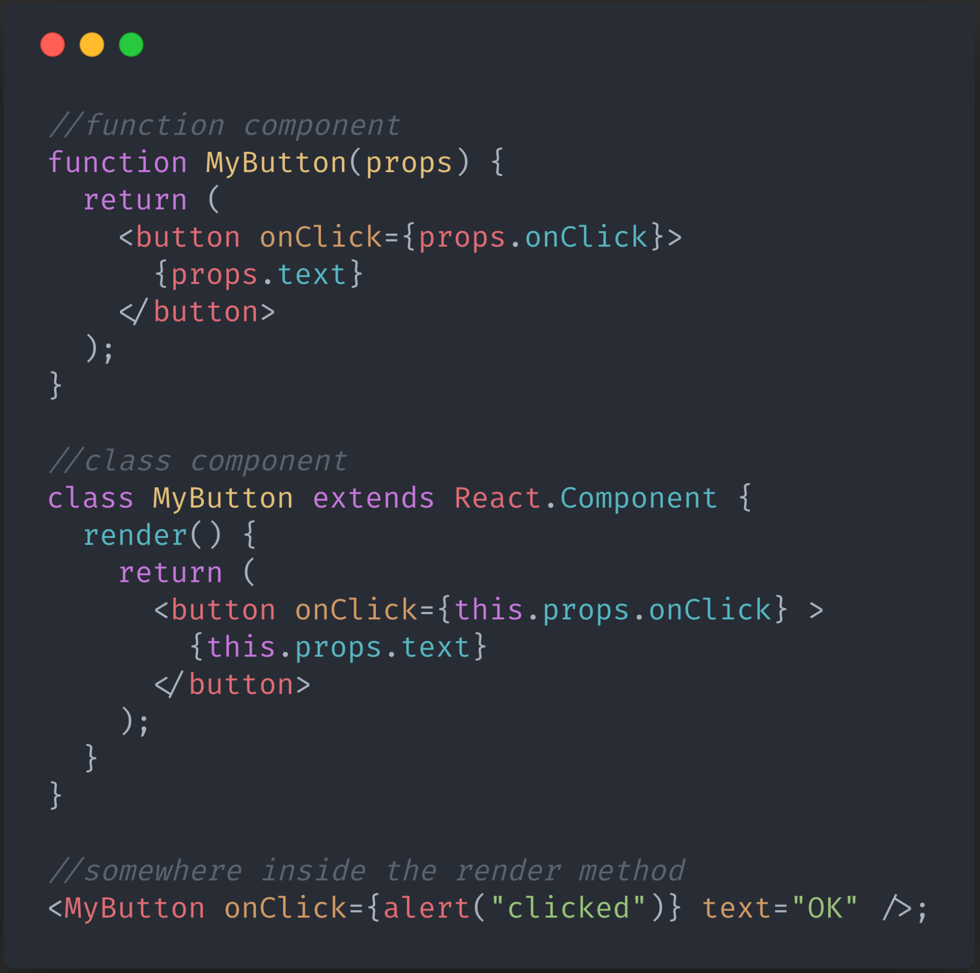
- 2014 Evan You
- Babel / Buble
- Virtual DOM
- View library
- relatively simple
- vue-cli
- Weex by Alibaba
- 23kb

And the winner is ...

- 🏎 Speed ➡ *All*
- 👴🏻 Mature ➡ *Angular / React*
- 📉 Learning curve ➡ *Vue*
- 💪🏻 Big company behind ➡ *Angular / React*
- ⭐ Popularity ➡ *React*
- 🏠 Startup ➡ *Vue*
- 🏢 Enterprise ➡ *Angular / React*
- 📱 Mobile ➡ *React*



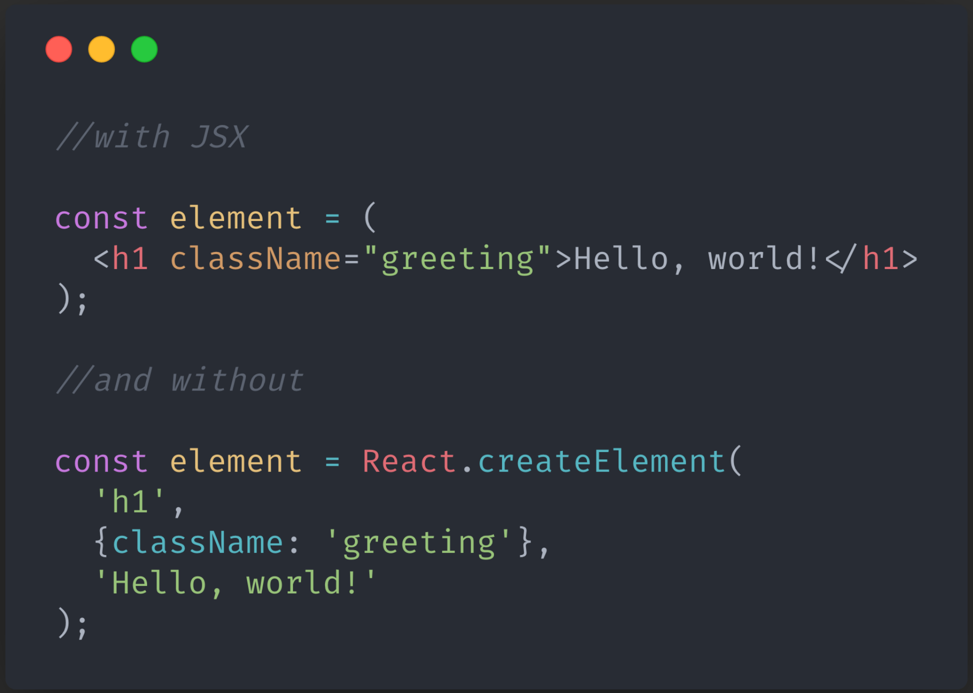
React Components



```
//function component
function MyButton(props) {
  return (
    <button onClick={props.onClick}>
      {props.text}
    </button>
  );
}

//class component
class MyButton extends React.Component {
  render() {
    return (
      <button onClick={this.props.onClick} >
        {this.props.text}
      </button>
    );
  }
}

//somewhere inside the render method
<MyButton onClick={alert("clicked")} text="OK" />;
```

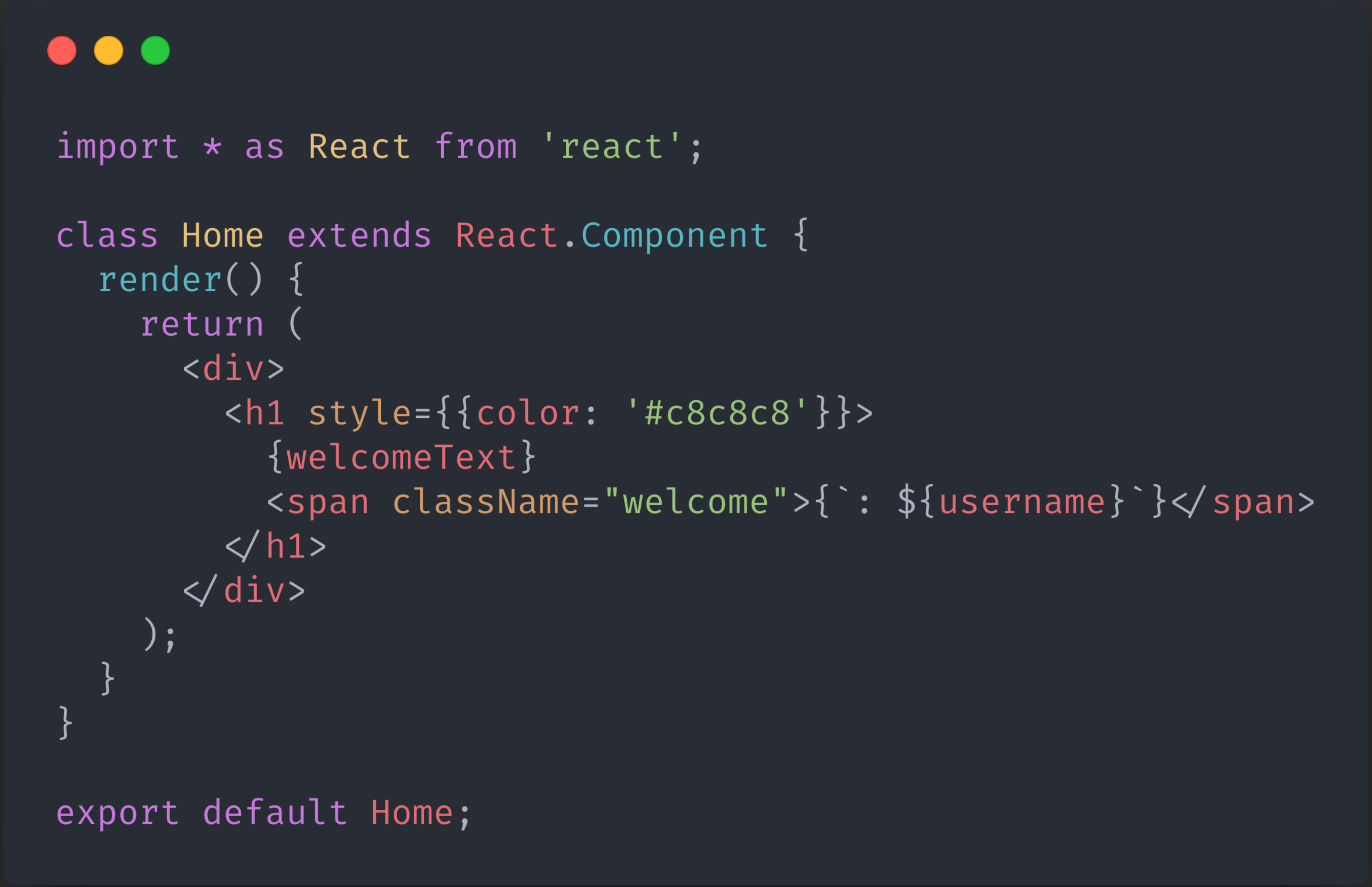
- reusable
- composable
- self-contained
- unit testable

JavaScript XML (JSX)

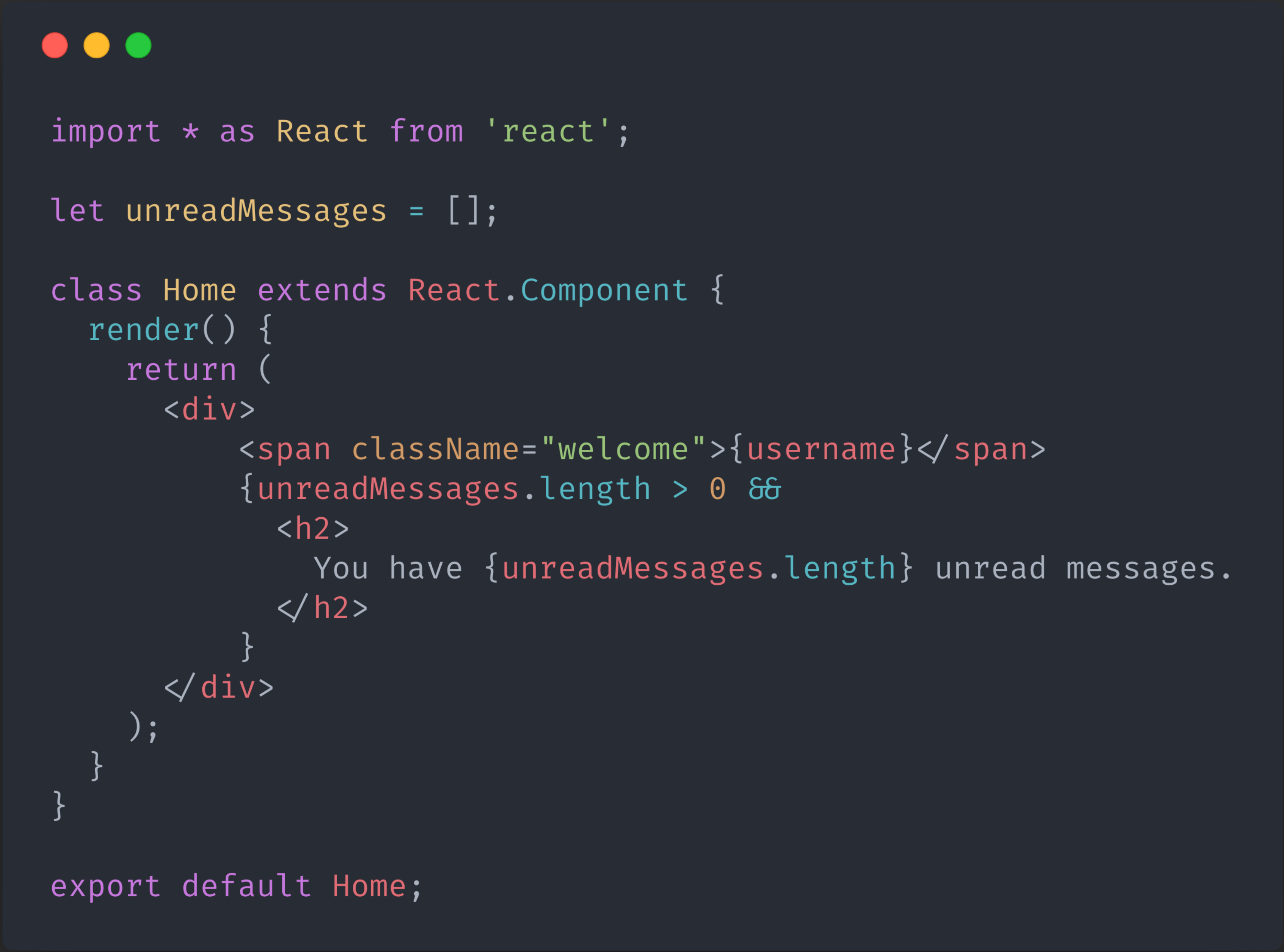


```
//with JSX

const element = (
  <h1 className="greeting">Hello, world!</h1>
);

//and without

const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

- create JS objects using HTML syntax
- improves development time
- forces safer coding

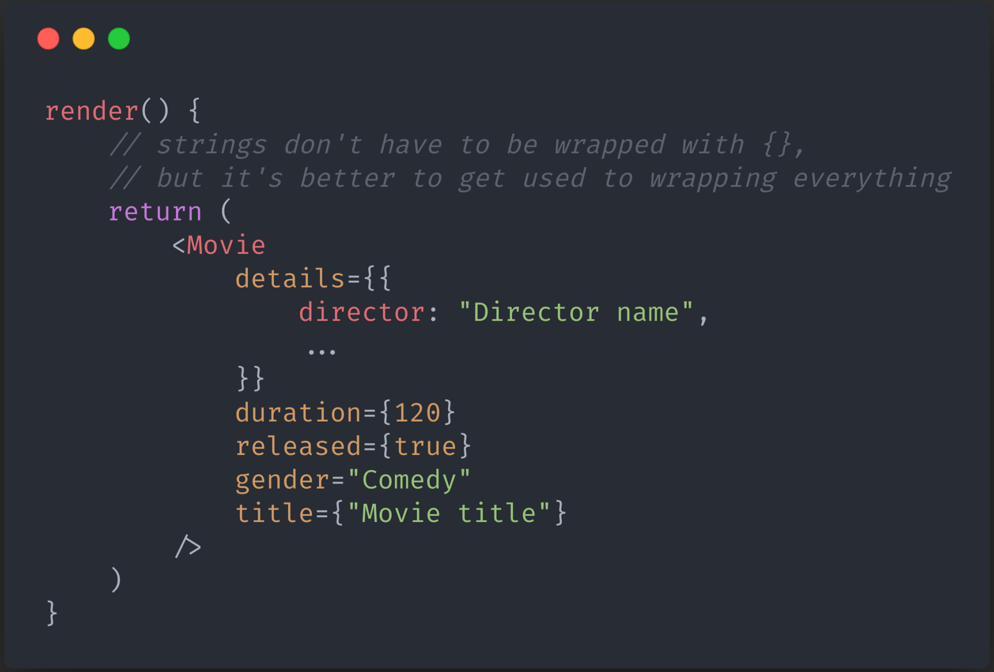


```
import * as React from 'react';

class Home extends React.Component {
  render() {
    return (
      <div>
        <h1 style={{color: '#c8c8c8'}}>
          {welcomeText}
          <span className="welcome">{`: ${username}`}</span>
        </h1>
      </div>
    );
  }
}

export default Home;
```



```
import * as React from 'react';

let unreadMessages = [];

class Home extends React.Component {
  render() {
    return (
      <div>
          <span className="welcome">{username}</span>
          {unreadMessages.length > 0 &&
            <h2>
              You have {unreadMessages.length} unread messages.
            </h2>
          }
      </div>
    );
  }
}

export default Home;
```

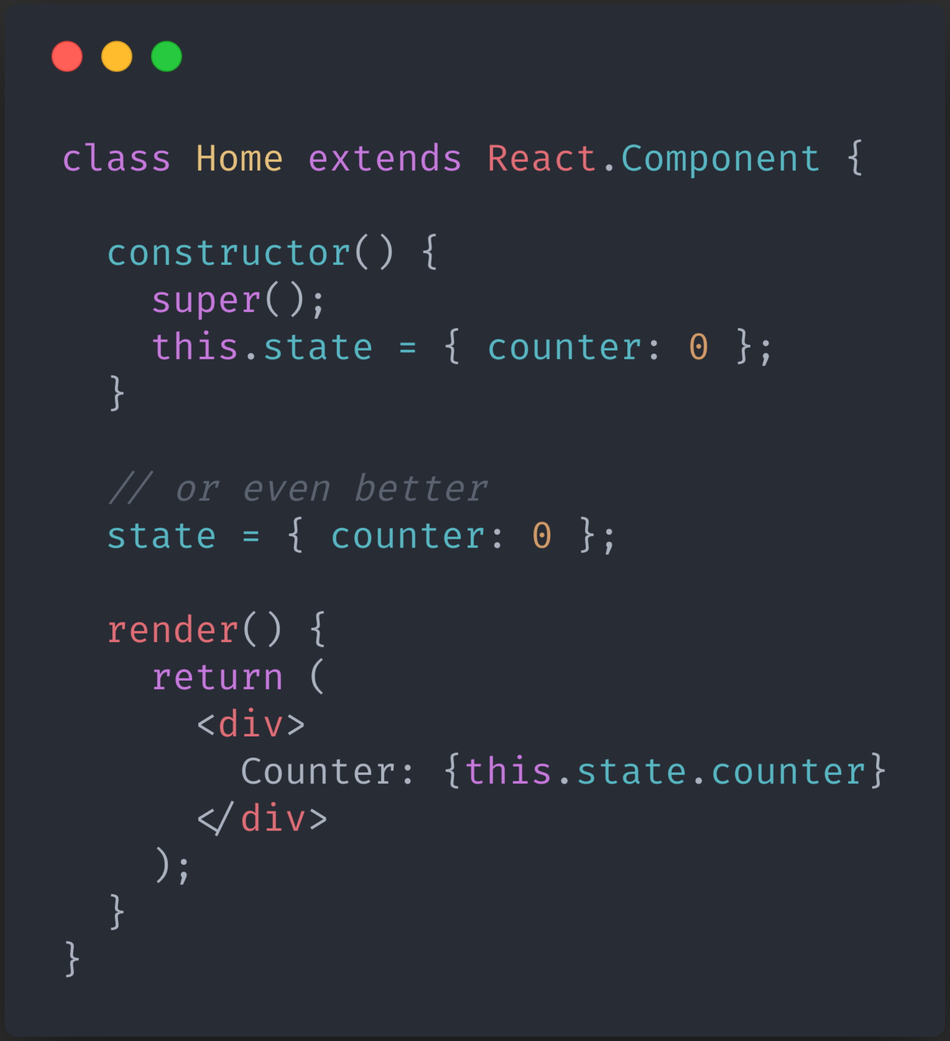
Component's Properties



```
render() {
    // strings don't have to be wrapped with {},
    // but it's better to get used to wrapping everything
    return (
        <Movie
            details={{
                director: "Director name",
                ...
            }}
            duration={120}
            released={true}
            gender="Comedy"
            title={"Movie title"}
        />
    )
}
```

- immutable
- Uni directional data flow

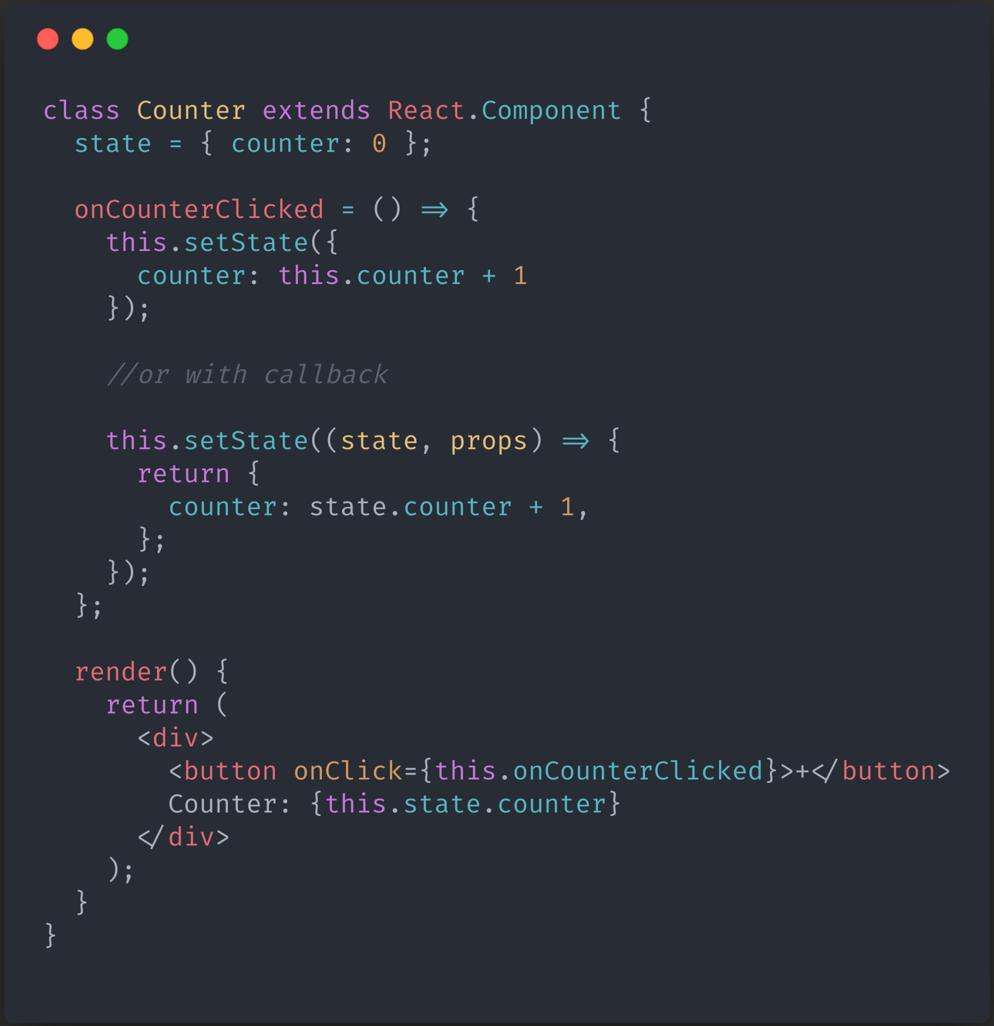
Component's State



```
class Home extends React.Component {

  constructor() {
    super();
    this.state = { counter: 0 };
  }

  // or even better
  state = { counter: 0 };

  render() {
    return (
      <div>
        Counter: {this.state.counter}
      </div>
    );
  }
}
```

- only for classes
- local
- mutable

Component's State p.2

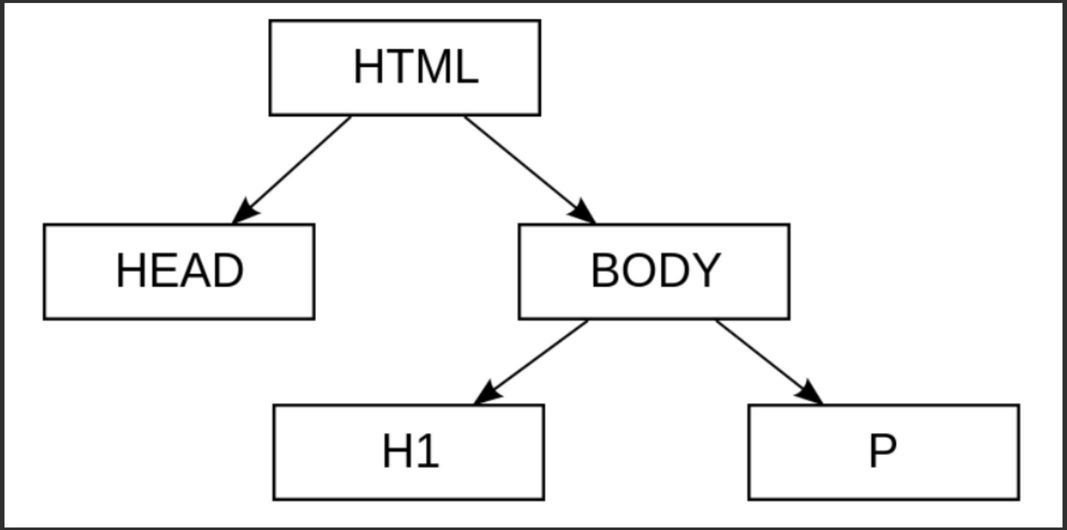


```
class Counter extends React.Component {
  state = { counter: 0 };

  onCounterClicked = () => {
    this.setState({
      counter: this.counter + 1
    });

    //or with callback

    this.setState((state, props) => {
      return {
        counter: state.counter + 1,
      };
    });
  };

  render() {
    return (
      <div>
        <button onClick={this.onCounterClicked}>+</button>
        Counter: {this.state.counter}
      </div>
    );
  }
}
```

- used to handle internal changes
- asynchronous

Document Object Model (DOM)

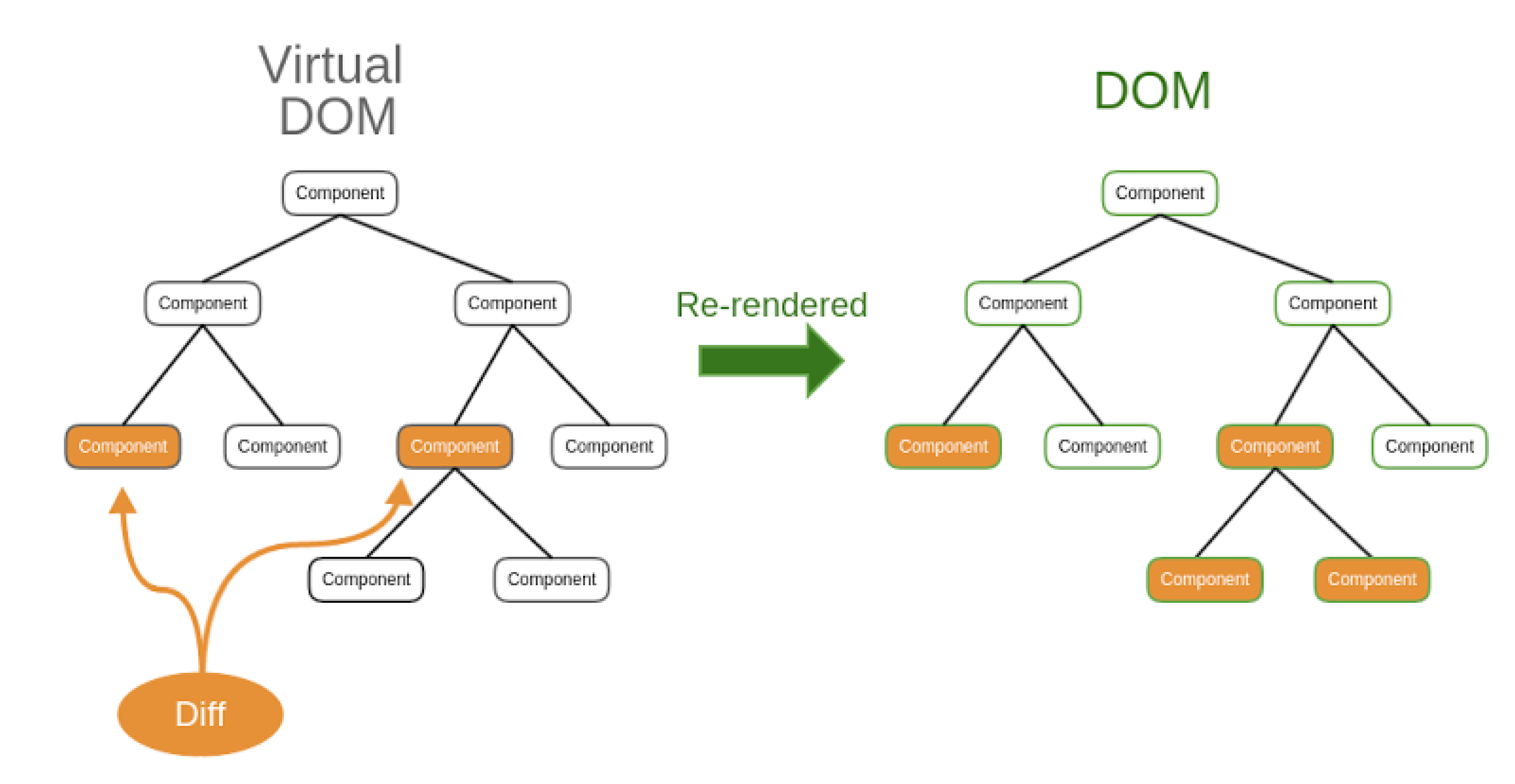
- The HTML DOM is always tree-structured
- `getElementById`, `parentNode` and `removeChild` are methods from HTML DOM API.
- Changes to the DOM cause re-rendering of the page.
- Operations on DOM are slow



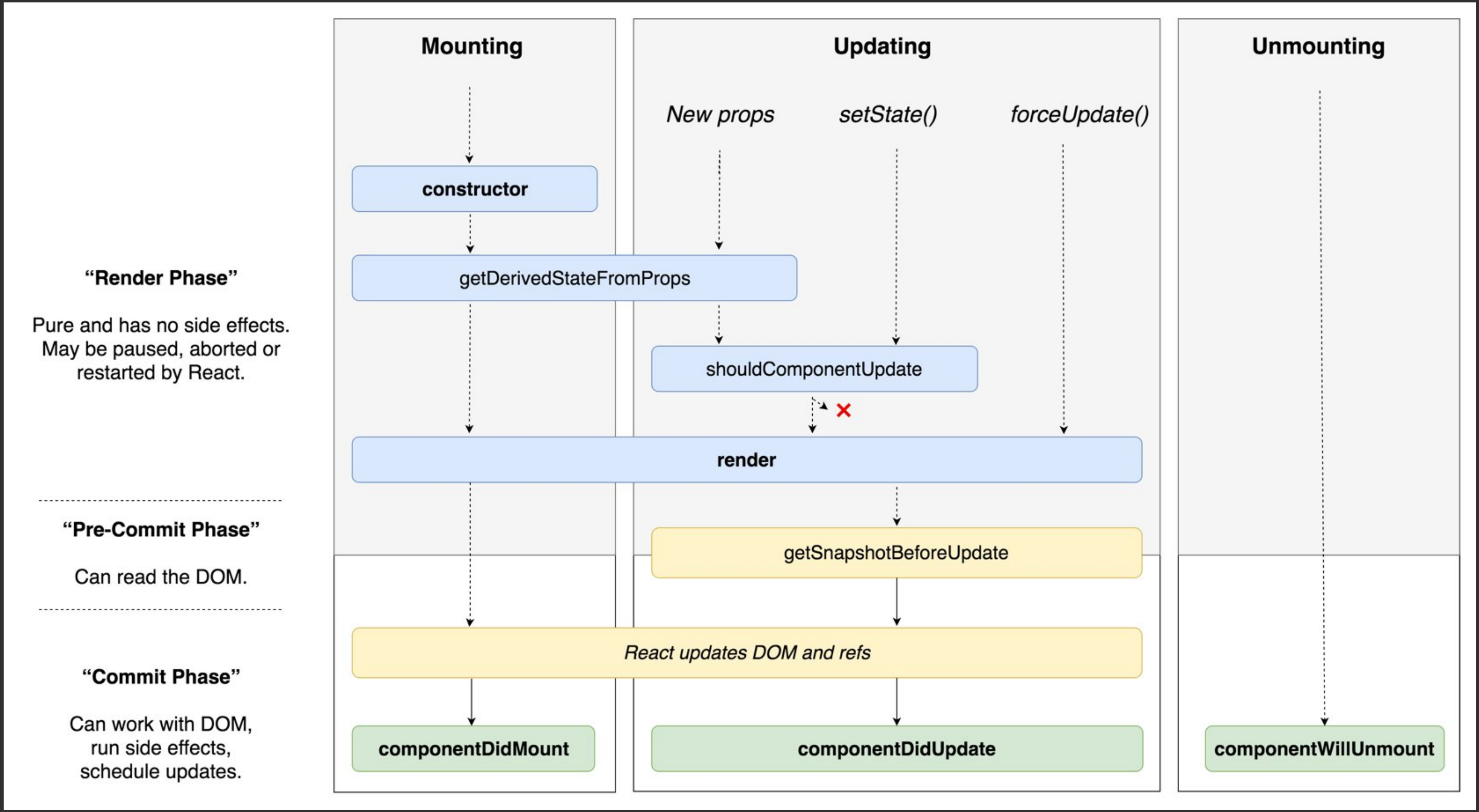
Virtual DOM

- allows to collect several changes to be applied at once.
- re-rendering only happens once after a set of changes was applied to the DOM.
- is located in memory
- avoiding unnecessary changes to the DOM.

Virtual DOM vs HTML DOM

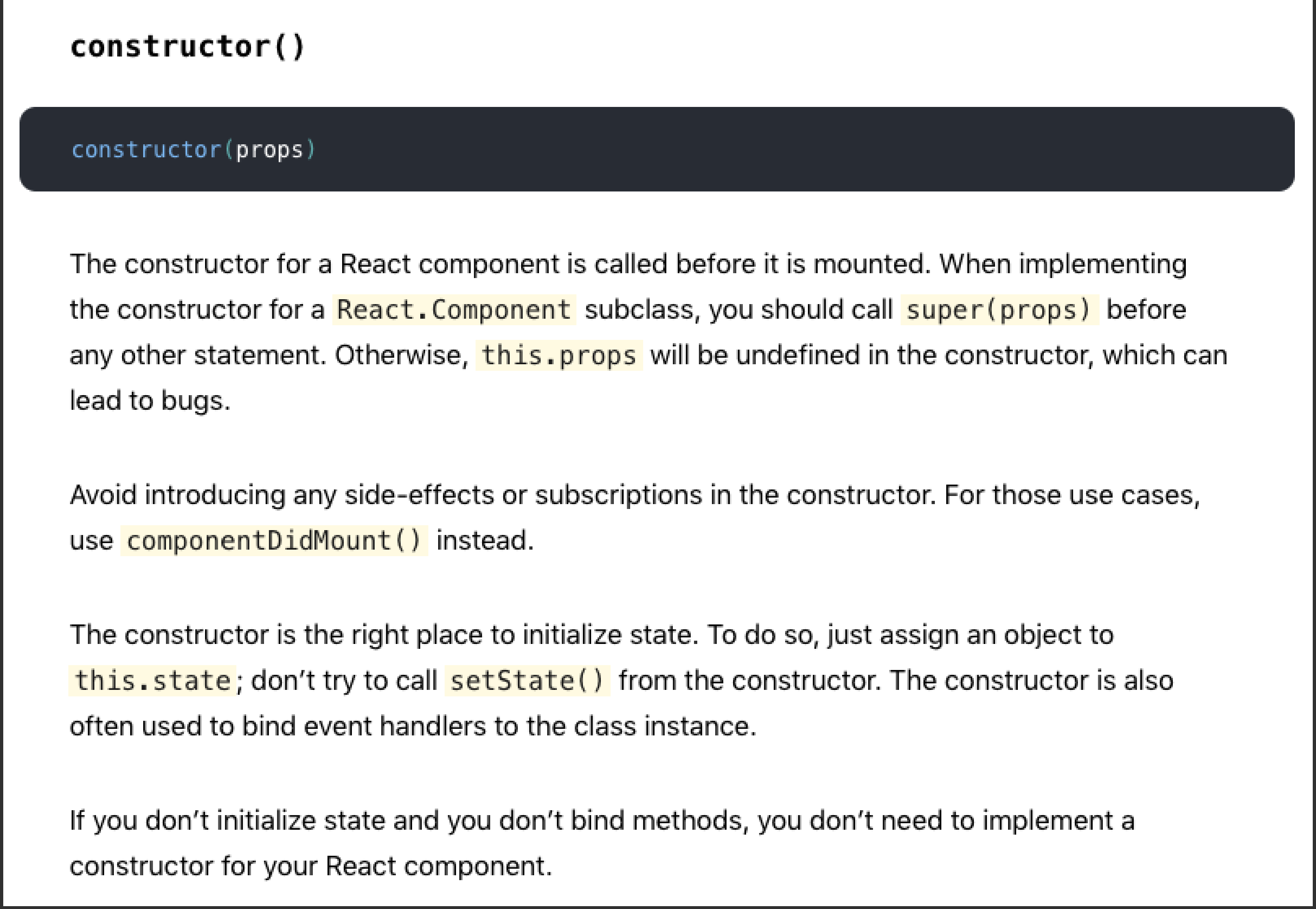


Component Lifecycle



Interactive diagram here

constructor



`constructor()`

```
constructor(props)
```

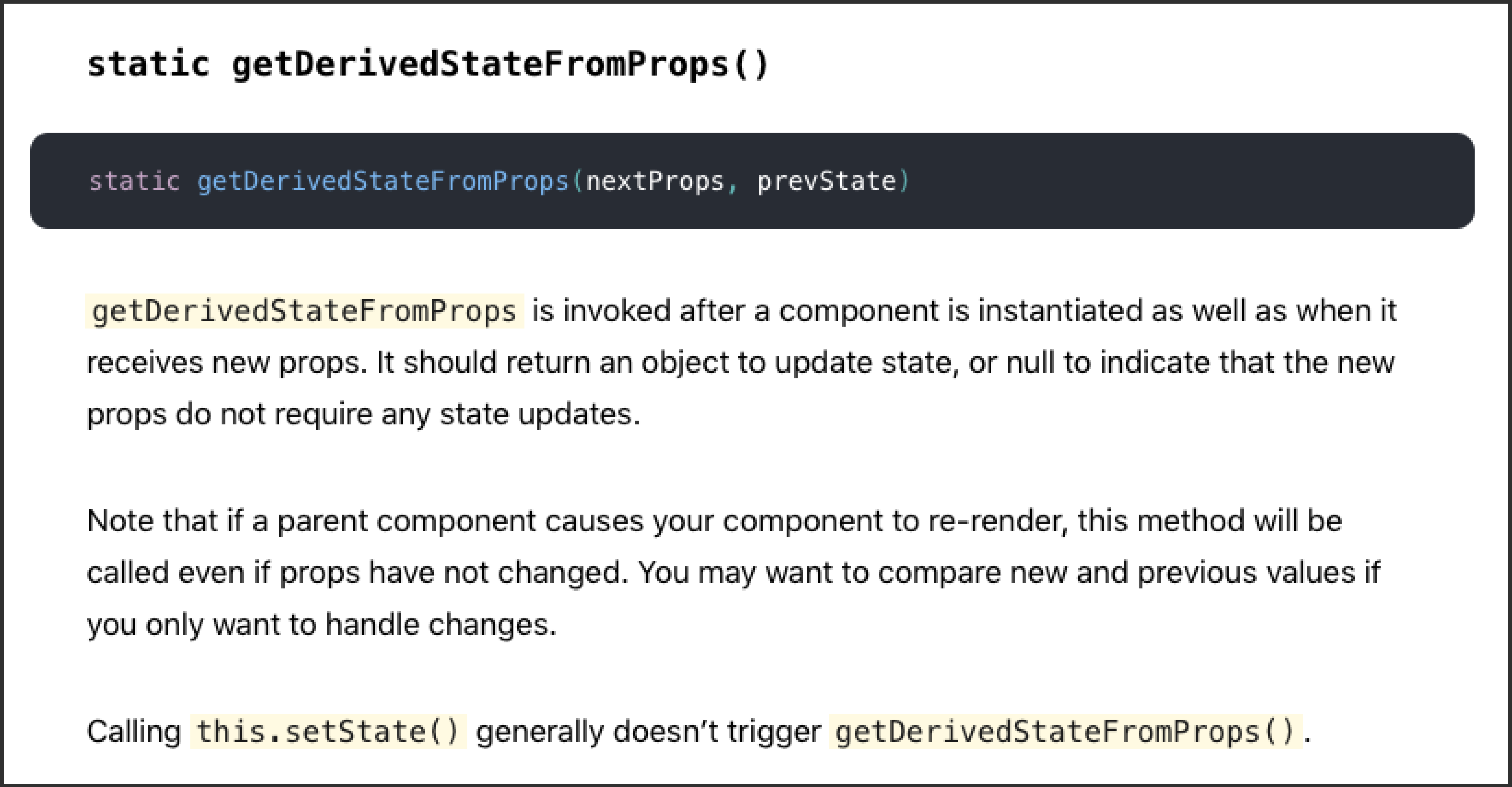
The constructor for a React component is called before it is mounted. When implementing the constructor for a `React.Component` subclass, you should call `super(props)` before any other statement. Otherwise, `this.props` will be undefined in the constructor, which can lead to bugs.

Avoid introducing any side-effects or subscriptions in the constructor. For those use cases, use `componentDidMount()` instead.

The constructor is the right place to initialize state. To do so, just assign an object to `this.state`; don't try to call `setState()` from the constructor. The constructor is also often used to bind event handlers to the class instance.

If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.

getDerivedStateFromProps



`static getDerivedStateFromProps()`

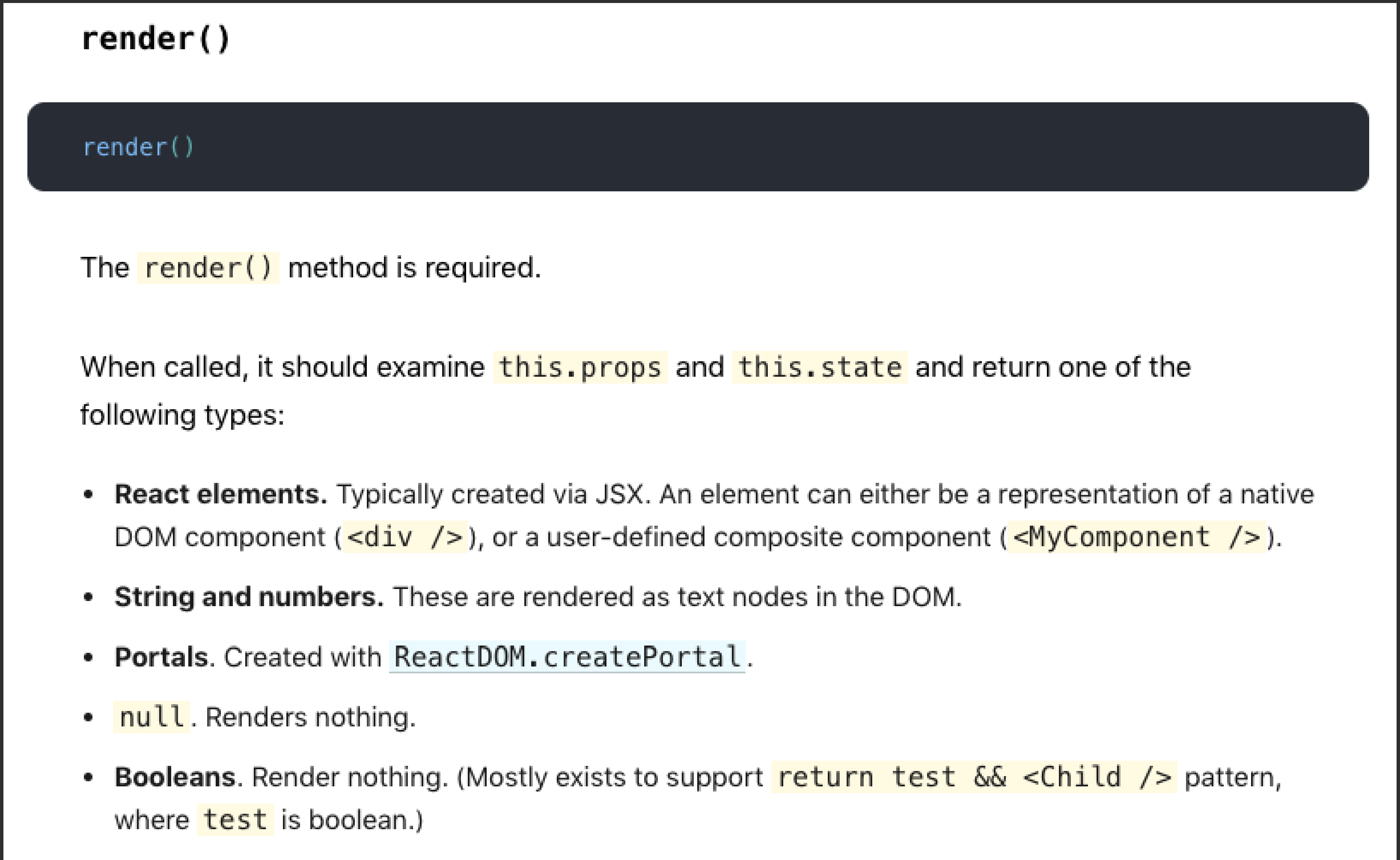
```
static getDerivedStateFromProps(nextProps, prevState)
```

`getDerivedStateFromProps` is invoked after a component is instantiated as well as when it receives new props. It should return an object to update state, or null to indicate that the new props do not require any state updates.

Note that if a parent component causes your component to re-render, this method will be called even if props have not changed. You may want to compare new and previous values if you only want to handle changes.

Calling `this.setState()` generally doesn’t trigger `getDerivedStateFromProps()`.

render



`render()`

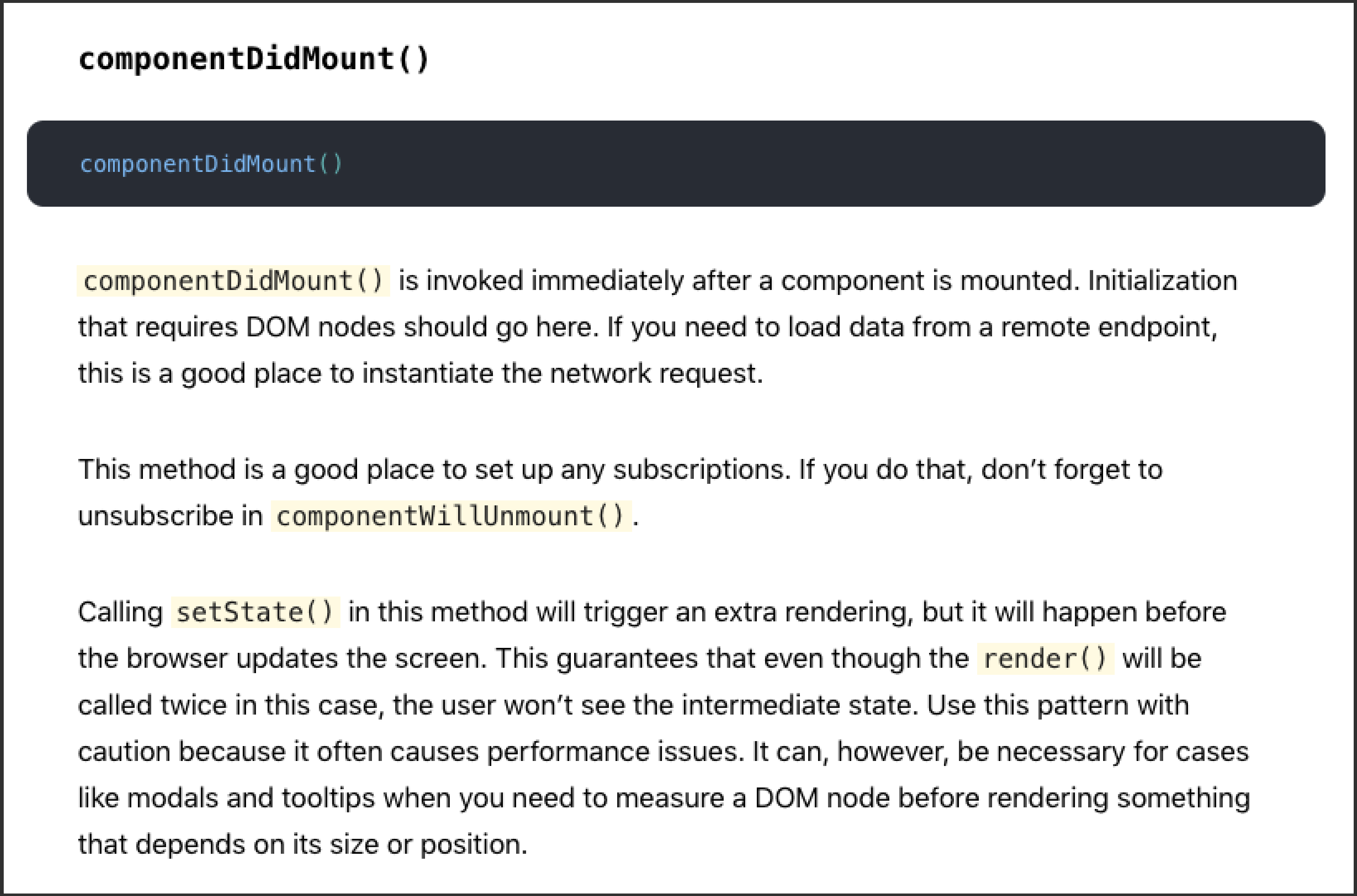
```
render()
```

The `render()` method is required.

When called, it should examine `this.props` and `this.state` and return one of the following types:

- **React elements.** Typically created via JSX. An element can either be a representation of a native DOM component (`<div />`), or a user-defined composite component (`<MyComponent />`).
- **String and numbers.** These are rendered as text nodes in the DOM.
- **Portals**. Created with `ReactDOM.createPortal`.
- `null`. Renders nothing.
- **Booleans**. Render nothing. (Mostly exists to support `return test && <Child />` pattern, where `test` is boolean.)

componentDidMount



`componentDidMount()`

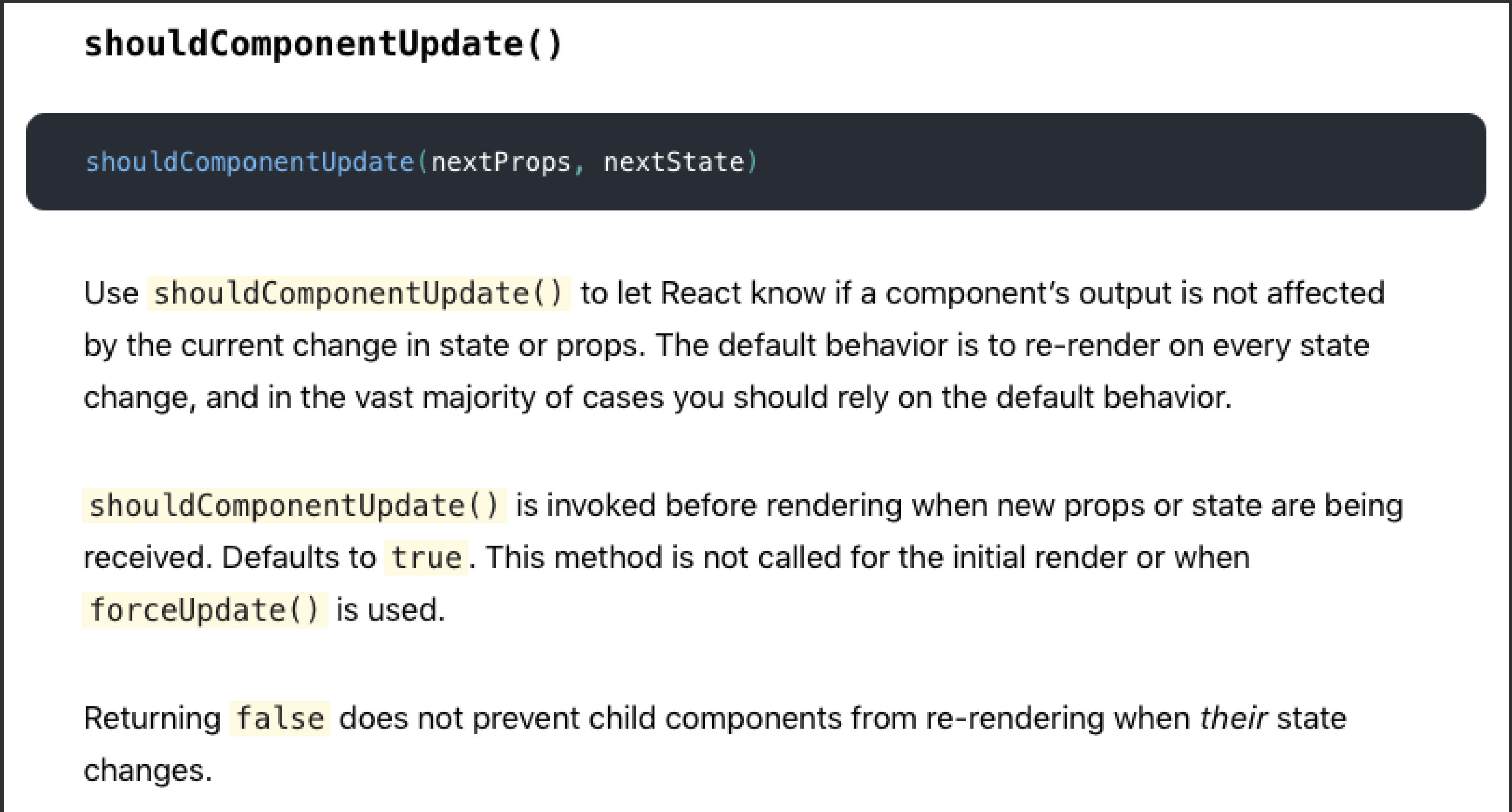
```
componentDidMount()
```

`componentDidMount()` is invoked immediately after a component is mounted. Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.

This method is a good place to set up any subscriptions. If you do that, don't forget to unsubscribe in `componentWillUnmount()`.

Calling `setState()` in this method will trigger an extra rendering, but it will happen before the browser updates the screen. This guarantees that even though the `render()` will be called twice in this case, the user won't see the intermediate state. Use this pattern with caution because it often causes performance issues. It can, however, be necessary for cases like modals and tooltips when you need to measure a DOM node before rendering something that depends on its size or position.

shouldComponentUpdate



`shouldComponentUpdate()`

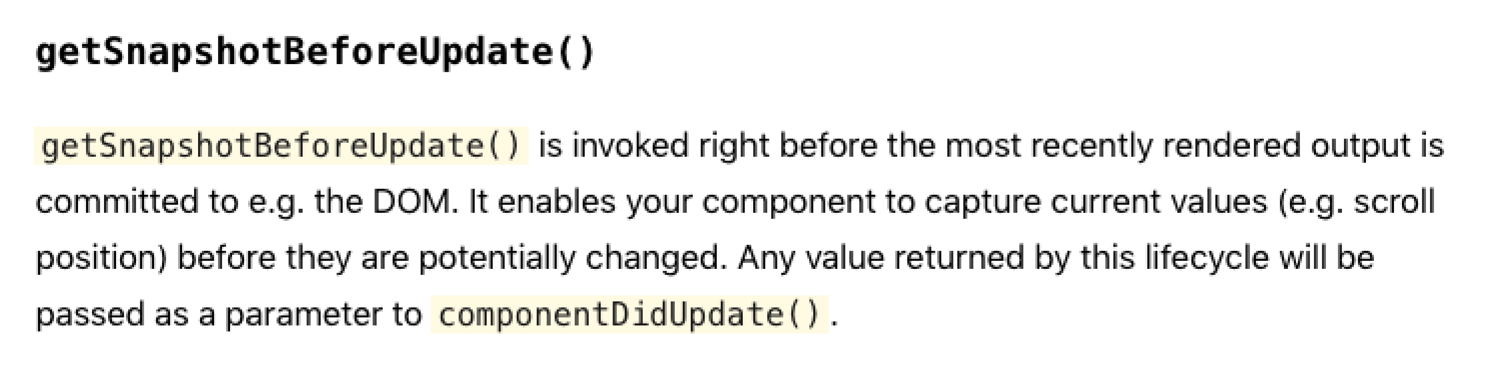
```
shouldComponentUpdate(nextProps, nextState)
```

Use `shouldComponentUpdate()` to let React know if a component’s output is not affected by the current change in state or props. The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.

`shouldComponentUpdate()` is invoked before rendering when new props or state are being received. Defaults to `true`. This method is not called for the initial render or when `forceUpdate()` is used.

Returning `false` does not prevent child components from re-rendering when *their* state changes.

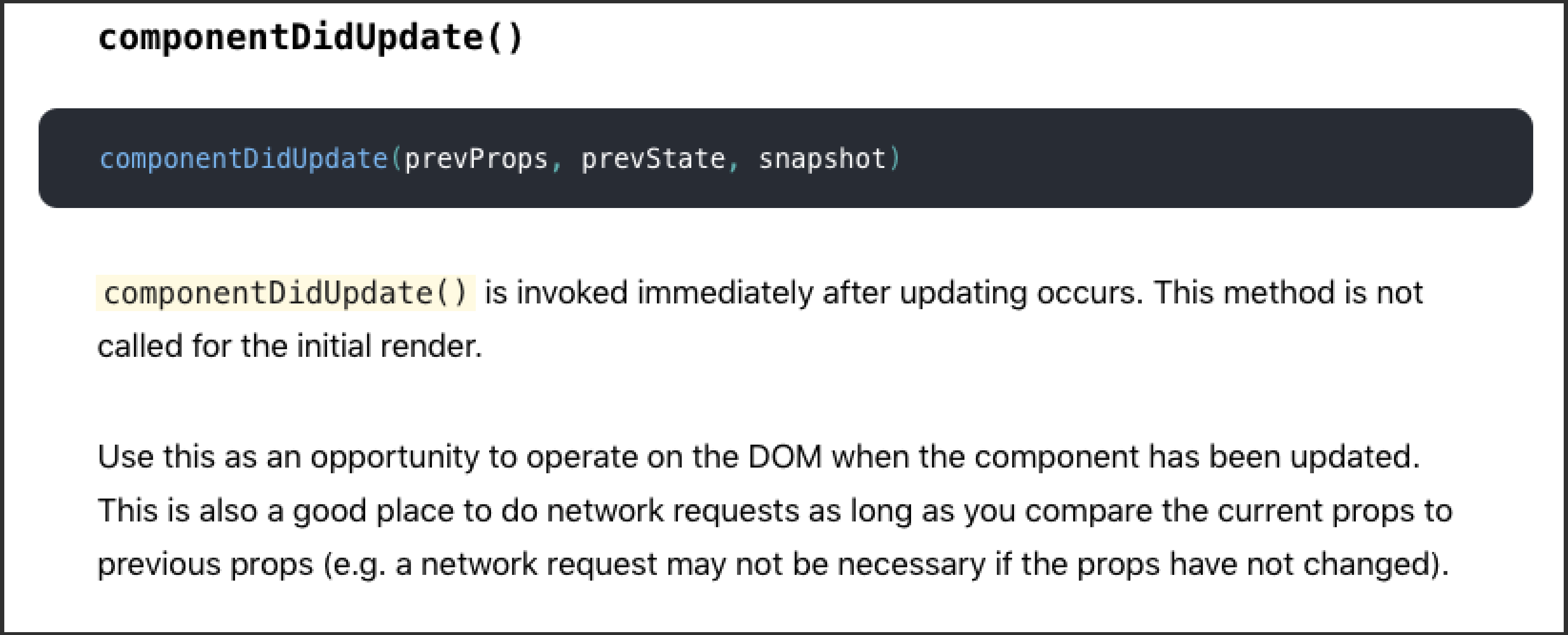
getSnapshotBeforeUpdate



`getSnapshotBeforeUpdate()`

`getSnapshotBeforeUpdate()` is invoked right before the most recently rendered output is committed to e.g. the DOM. It enables your component to capture current values (e.g. scroll position) before they are potentially changed. Any value returned by this lifecycle will be passed as a parameter to `componentDidUpdate()`.

componentDidUpdate



`componentDidUpdate()`

```
componentDidUpdate(prevProps, prevState, snapshot)
```

`componentDidUpdate()` is invoked immediately after updating occurs. This method is not called for the initial render.

Use this as an opportunity to operate on the DOM when the component has been updated. This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed).