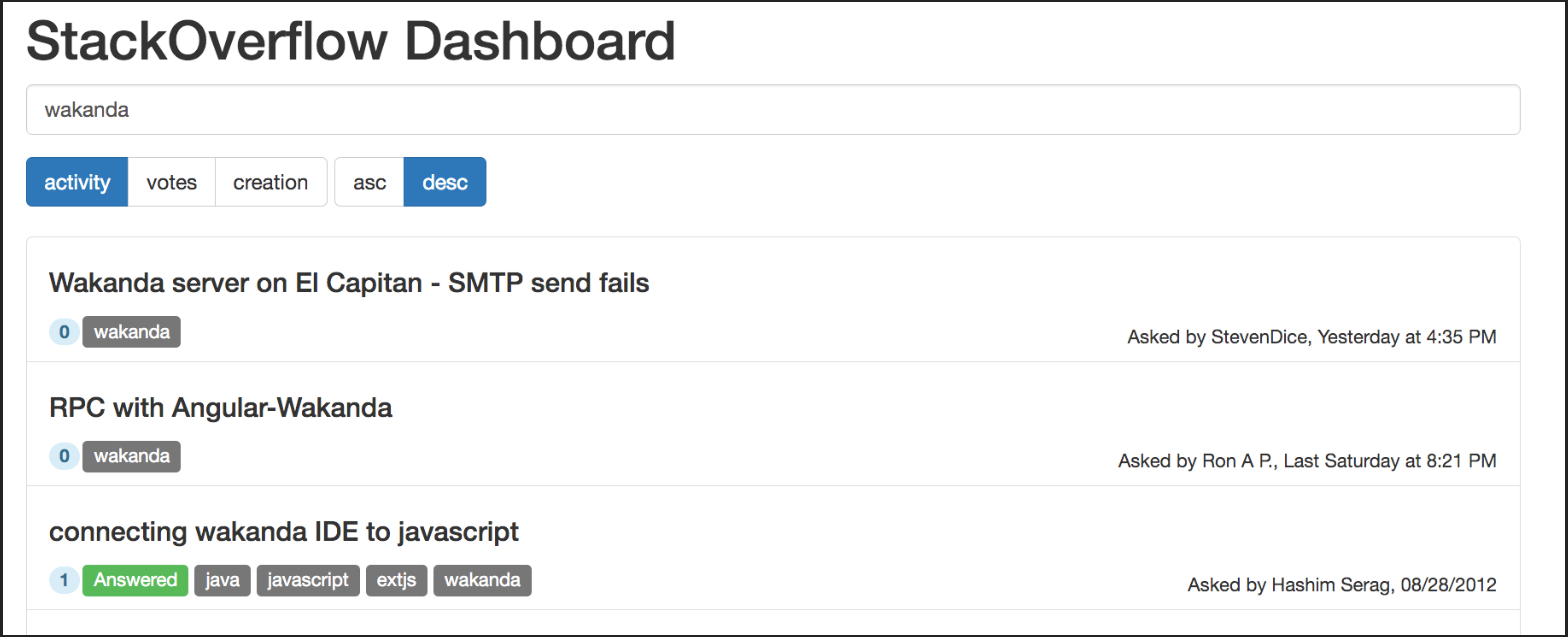
LET'S DISCOVER...

REACT AND REDUX

Mathieu Savy | @mrblackus

WHAT IS REACT?

- JS library to build User Interface (UI)
- Everything is Component
- Unidirectional data-flow (from parent to child)
- Virtual DOM
- **NO MORE!** (no controller, no router, no filter, etc.)



StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda — Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

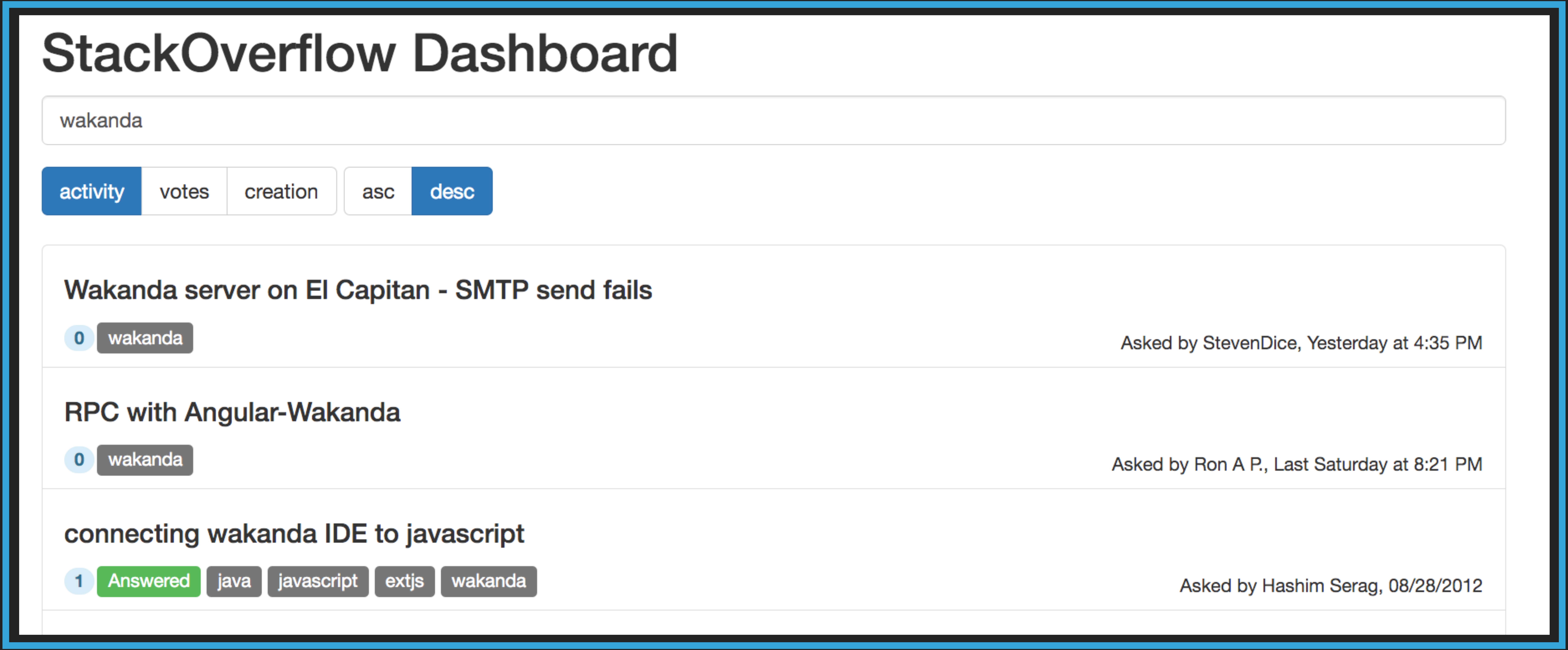
0 wakanda — Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda — Asked by Hashim Serag, 08/28/2012

LET'S DISCOVER REACT AND REDUX

AppComponent



StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda

Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

0 wakanda

Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda

Asked by Hashim Serag, 08/28/2012

AppComponent
SearchBarComponent

StackOverflow Dashboard

wakanda

activity
votes
creation
asc
desc

Wakanda server on El Capitan - SMTP send fails

0 wakanda — Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

0 wakanda — Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda — Asked by Hashim Serag, 08/28/2012

LET'S DISCOVER REACT AND REDUX

AppComponent
SearchBarComponent
SortBarComponent
activity
votes
creation
asc
desc

StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda

Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

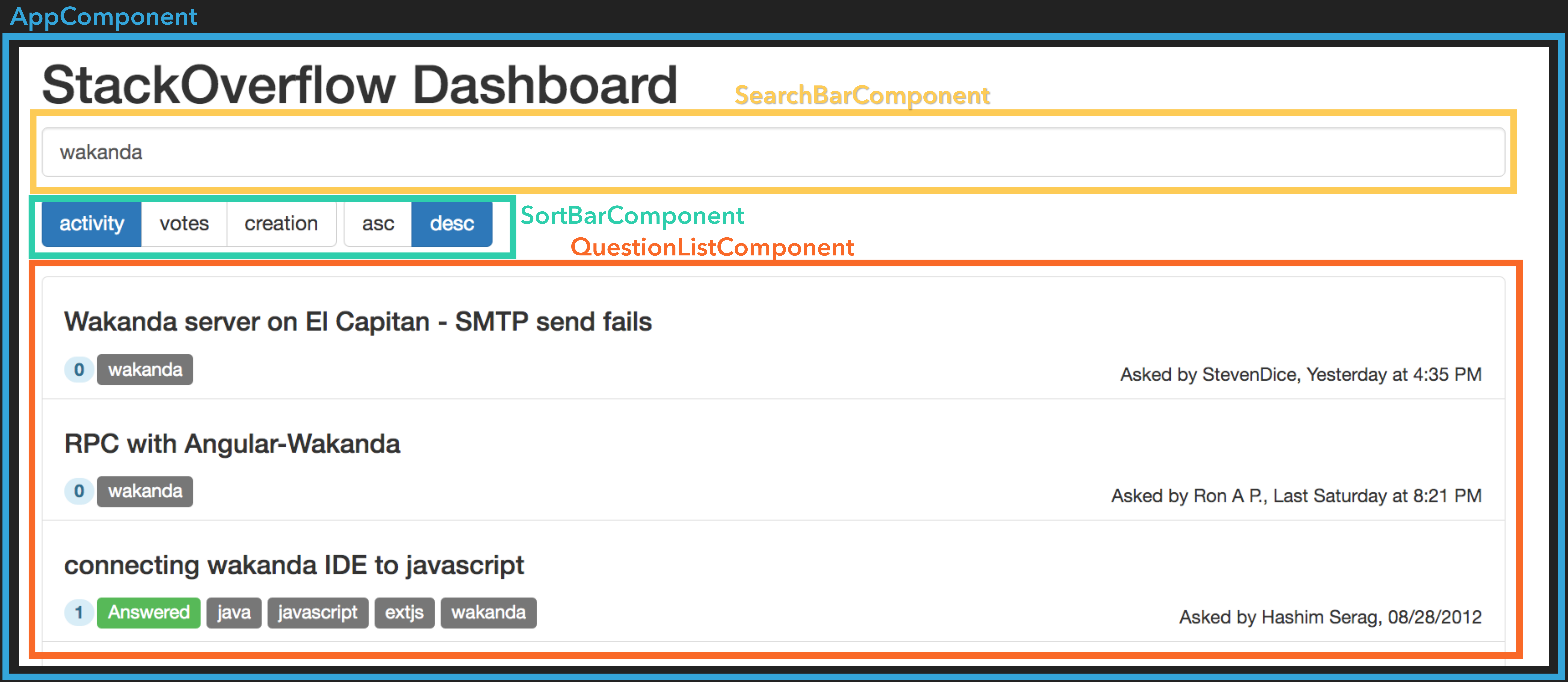
0 wakanda

Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda

Asked by Hashim Serag, 08/28/2012



StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda

Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

0 wakanda

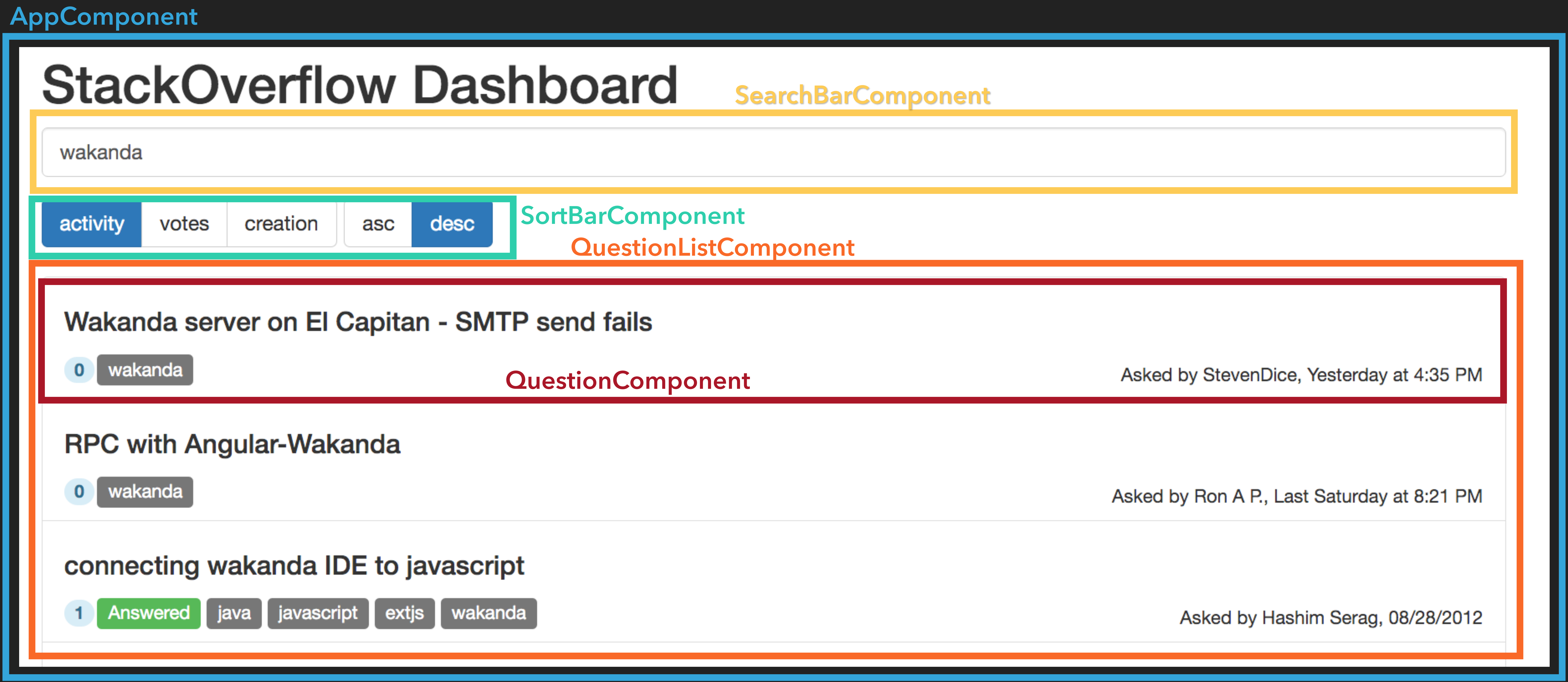
Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda

Asked by Hashim Serag, 08/28/2012

LET'S DISCOVER REACT AND REDUX



StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda

Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

0 wakanda

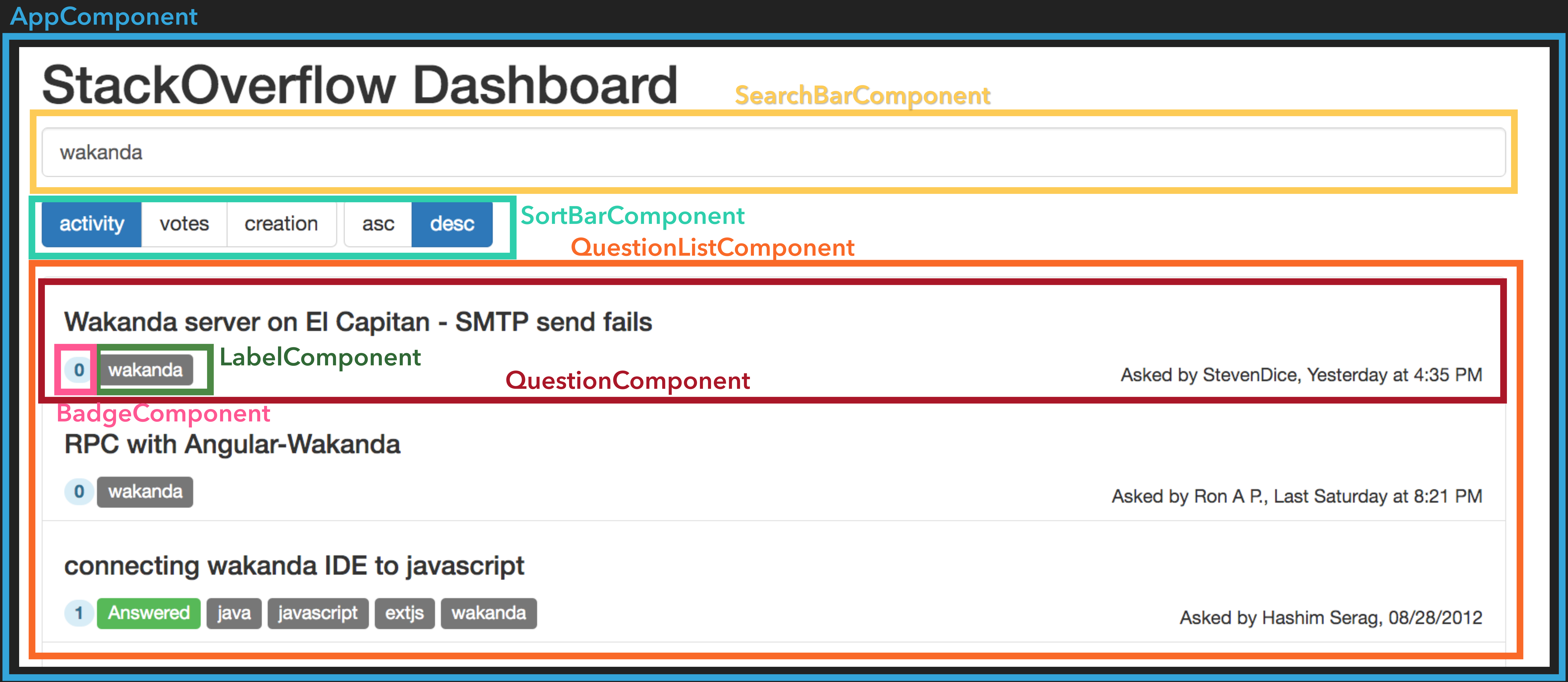
Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda

Asked by Hashim Serag, 08/28/2012

LET'S DISCOVER REACT AND REDUX



StackOverflow Dashboard

wakanda

Wakanda server on El Capitan - SMTP send fails

0 wakanda Asked by StevenDice, Yesterday at 4:35 PM

RPC with Angular-Wakanda

0 wakanda Asked by Ron A P., Last Saturday at 8:21 PM

connecting wakanda IDE to javascript

1 Answered java javascript extjs wakanda Asked by Hashim Serag, 08/28/2012

COMPONENT

```
import * as React from 'react';
import {render} from 'react-dom';

class SayHello extends React.Component<any, any> {
  render() {
    return <div>Hey! Hello there!</div>
  }
}

render(<SayHello />, document.getElementById('react-mount'));
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Let's discover React</title>
  </head>
  <body>
    <div id="react-mount"></div>
    <script src="build/foo.js"></script>
  </body>
</html>
```

COMPONENT

```
import * as React from 'react';
import {render} from 'react-dom';

class SayHello extends React.Component<any, any> {
  render() {
    return <div>Hey! Hello there!</div>
  }
}

render(<SayHello />, document.getElementById('react-mount'));
```

I'm pretty sure you can't do that.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Let's discover React</title>
  </head>
  <body>
    <div id="react-mount"></div>
    <script src="build/foo.js"></script>
  </body>
</html>
```

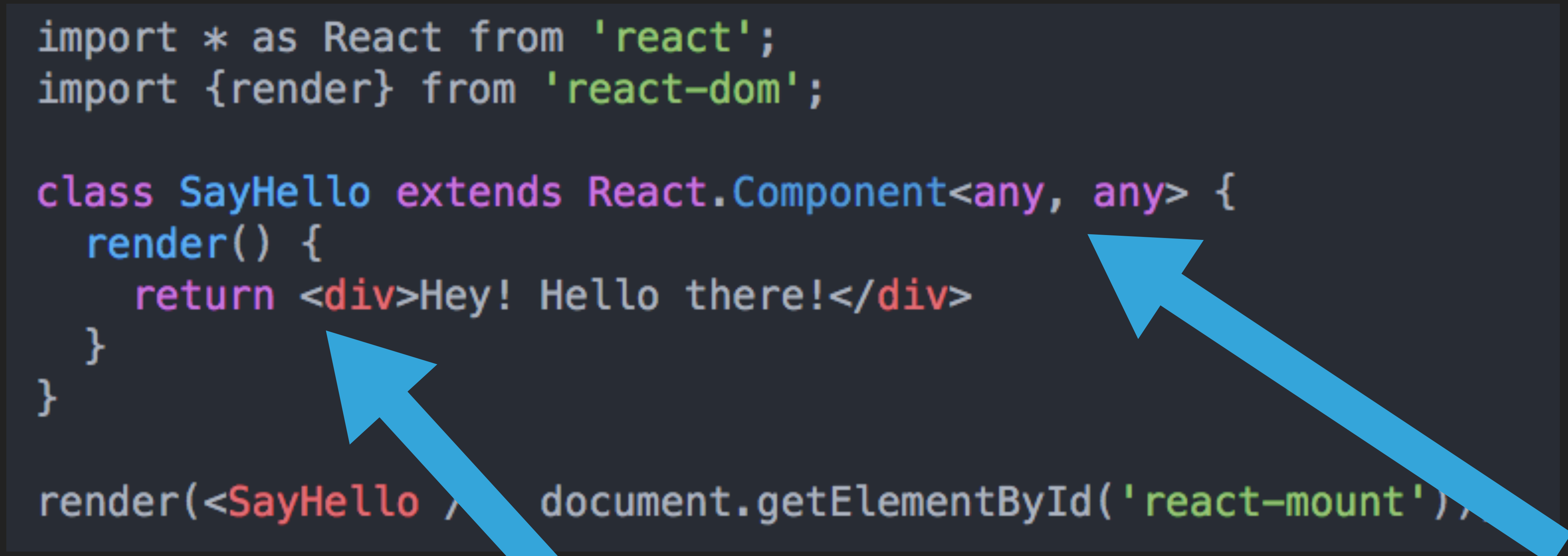
COMPONENT

```
import * as React from 'react';
import {render} from 'react-dom';

class SayHello extends React.Component<any, any> {
  render() {
    return <div>Hey! Hello there!</div>
  }
}

render(<SayHello />, document.getElementById('react-mount'));
```

Anyway, you shouldn't.
Separation of concerns,
MVC, bla bla bla…

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Let's discover React</title>
  </head>
  <body>
    <div id="react-mount"></div>
    <script src="build/foo.js"></script>
  </body>
</html>
```

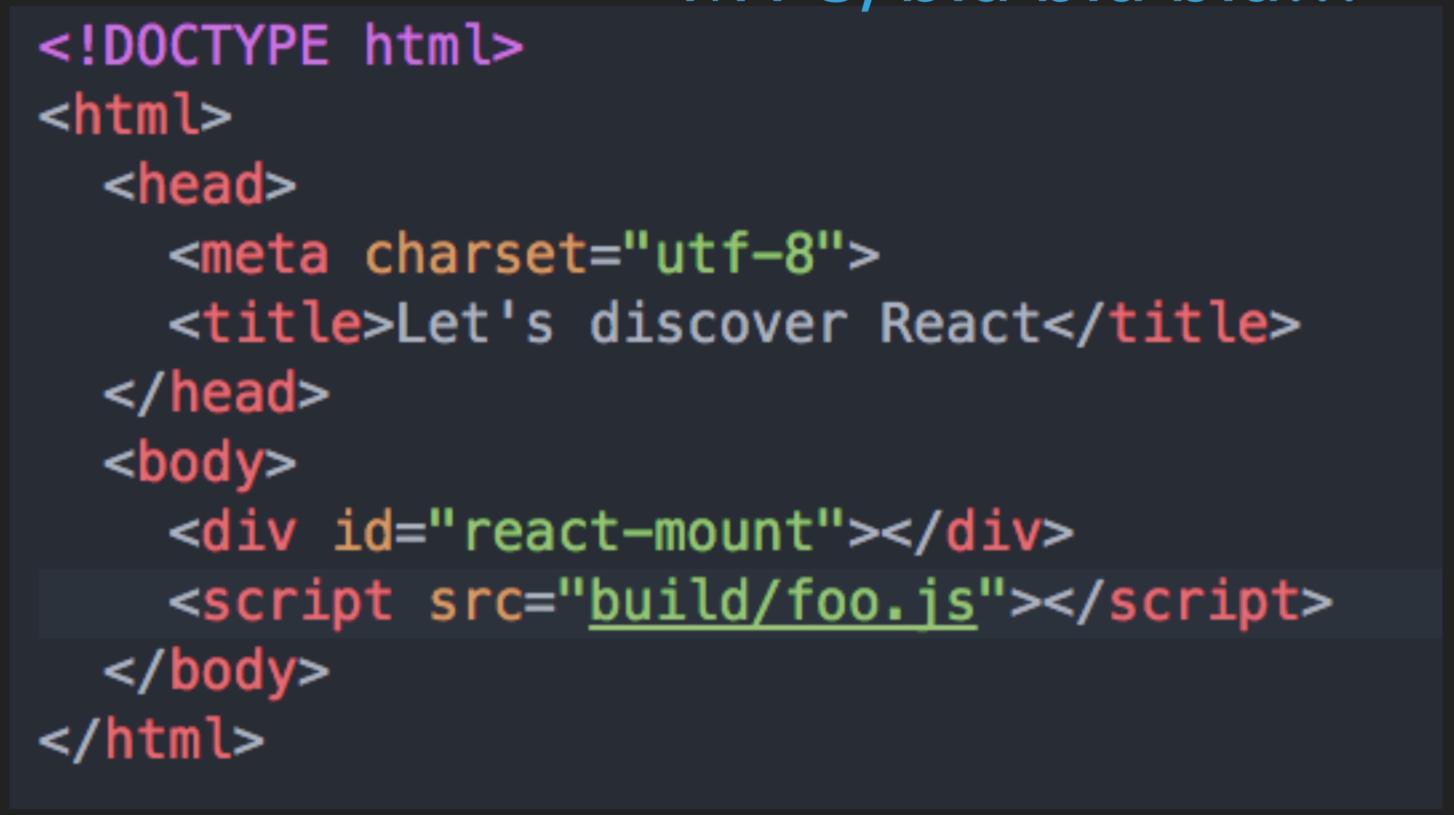
COMPONENT



```
import * as React from 'react';
import {render} from 'react-dom';

class SayHello extends React.Component<any, any> {
  render() {
    return <div>Hey! Hello there!</div>
  }
}

render(<SayHello />, document.getElementById('react-mount'));
```

And that?!

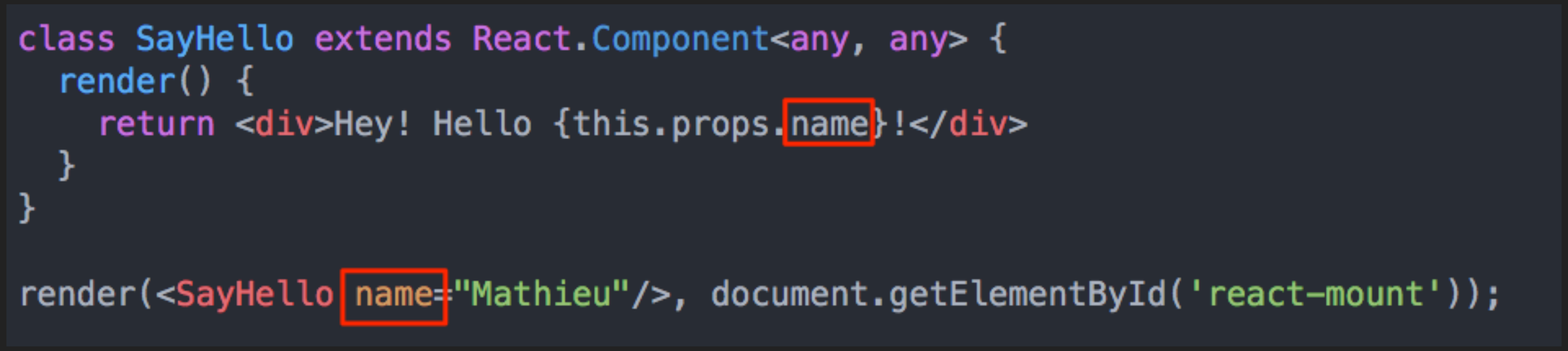
Anyway, you shouldn't.
Separation of concerns,
MVC, bla bla bla…



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Let's discover React</title>
  </head>
  <body>
    <div id="react-mount"></div>
    <script src="build/foo.js"></script>
  </body>
</html>
```

PROPERTIES

- Used to pass values from a parent component to a child component
- Like HTML attributes
- **Immutable** values (can't be used to pass data from child to parent)



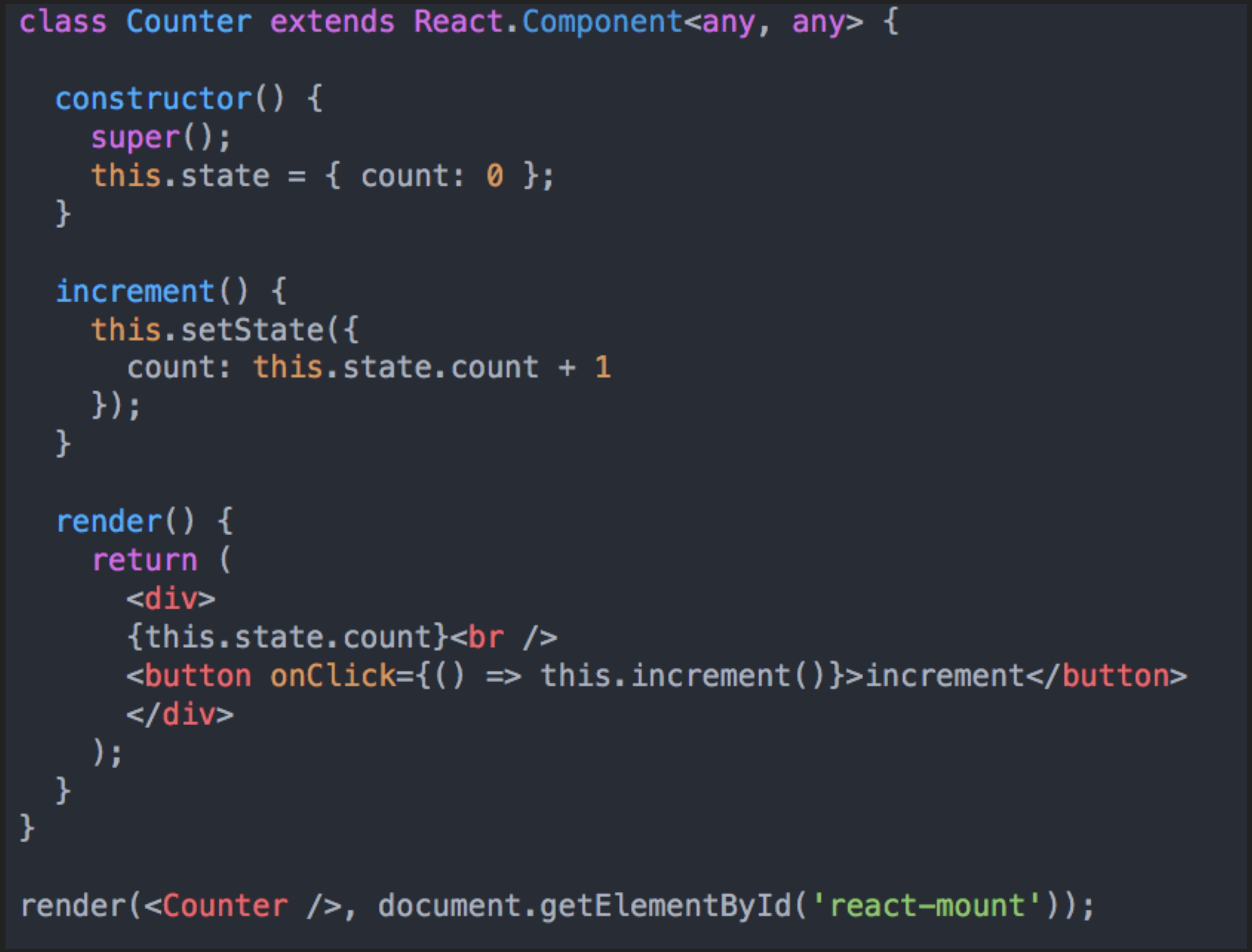
```
class SayHello extends React.Component<any, any> {
  render() {
    return <div>Hey! Hello {this.props.name}!</div>
  }
}

render(<SayHello name="Mathieu"/>, document.getElementById('react-mount'));
```

Will display : "Hey! Hello Mathieu!"

STATE

- Represent the **internal state** of a component
- State can be modified
- Must have an **initial value**
- Component is re-rendered if the state is modified

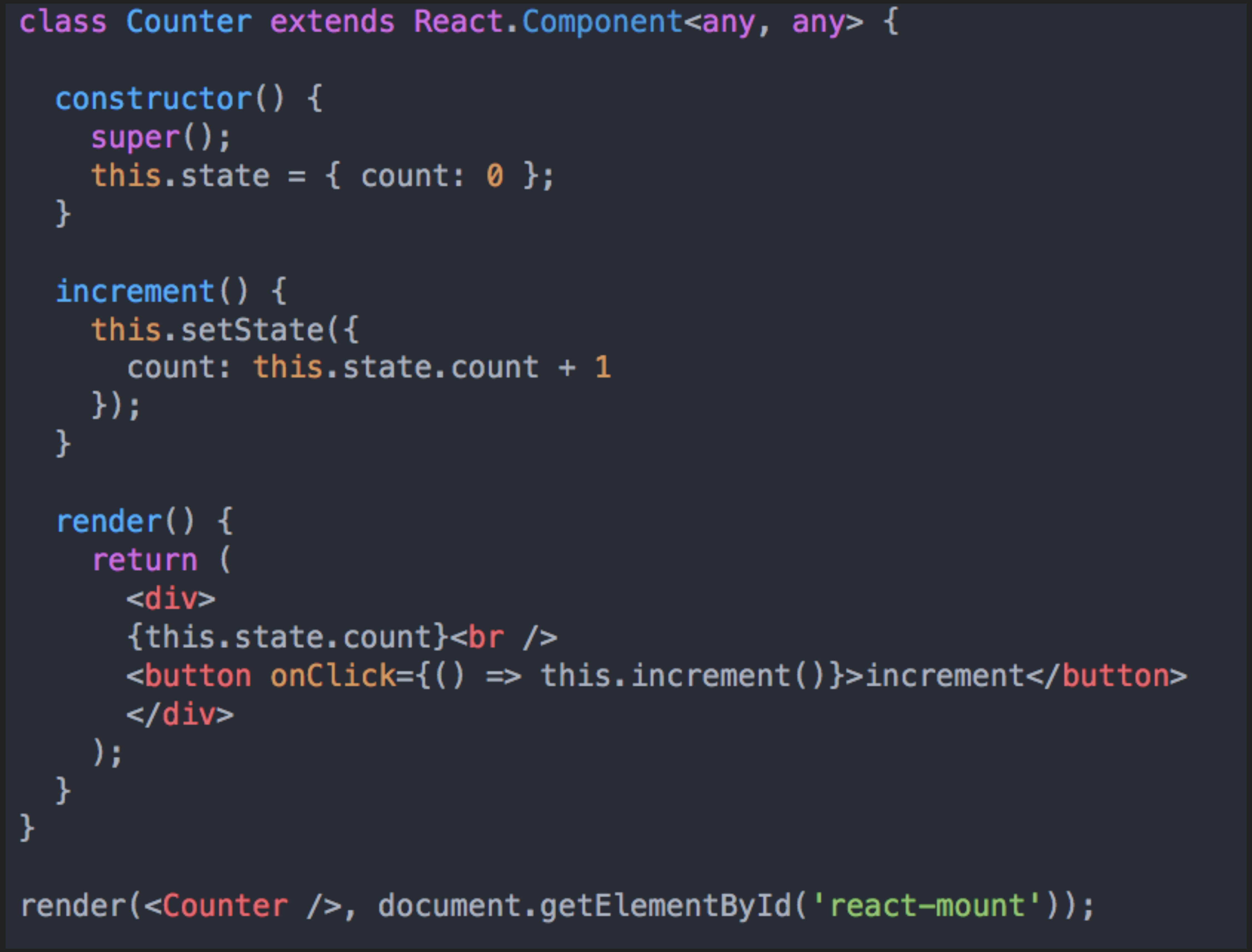
STATE



```
class Counter extends React.Component<any, any> {

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

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

  render() {
    return (
      <div>
      {this.state.count}<br />
      <button onClick={() => this.increment()}>increment</button>
      </div>
    );
  }
}

render(<Counter />, document.getElementById('react-mount'));
```

STATE

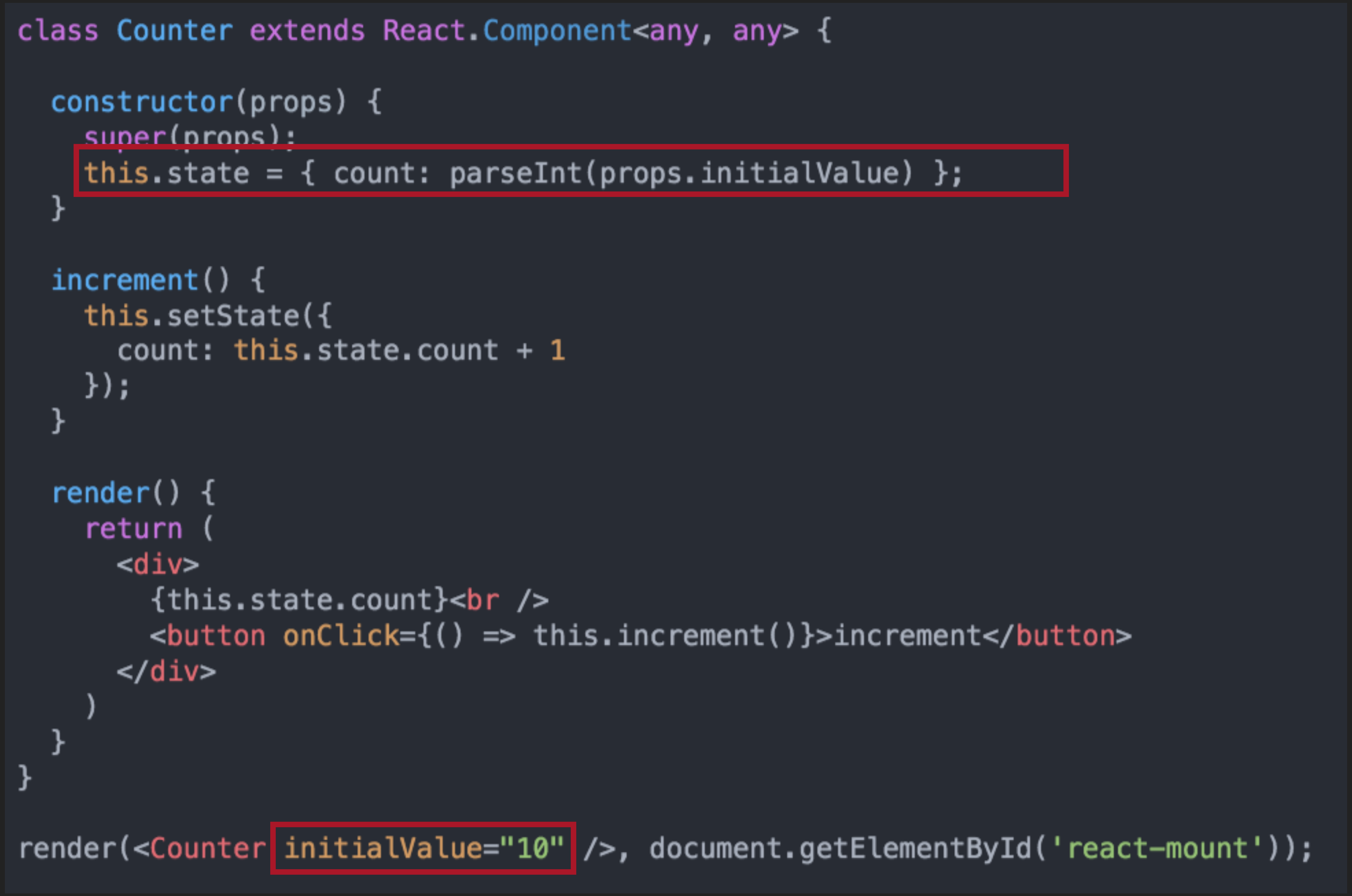
And if I want to choose my initial state at runtime?



```
class Counter extends React.Component<any, any> {

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

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

  render() {
    return (
      <div>
      {this.state.count}<br />
      <button onClick={() => this.increment()}>increment</button>
      </div>
    );
  }
}

render(<Counter />, document.getElementById('react-mount'));
```

STATE

Use props!



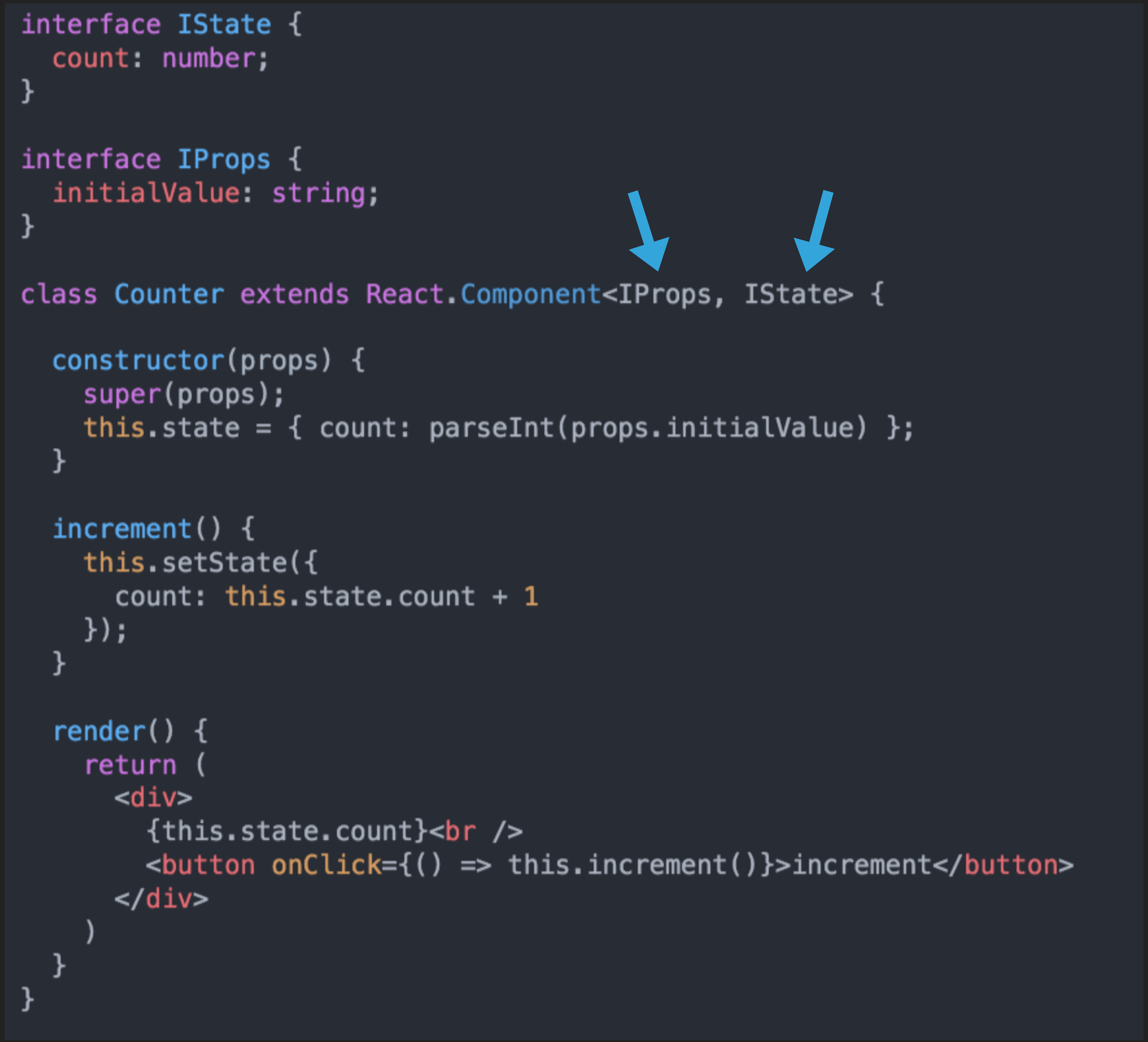
```
class Counter extends React.Component<any, any> {

  constructor(props) {
    super(props);
    this.state = { count: parseInt(props.initialValue) };
  }

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

  render() {
    return (
      <div>
        {this.state.count}<br />
        <button onClick={() => this.increment()}>increment</button>
      </div>
    )
  }
}

render(<Counter initialValue="10" />, document.getElementById('react-mount'));
```

LET'S TYPE CHECK OUR COUNTER

- We use TypeScript for **type checking**, so let's be sure our number is a number (who knows…)
- Remember the <any, any> part of React.Component?

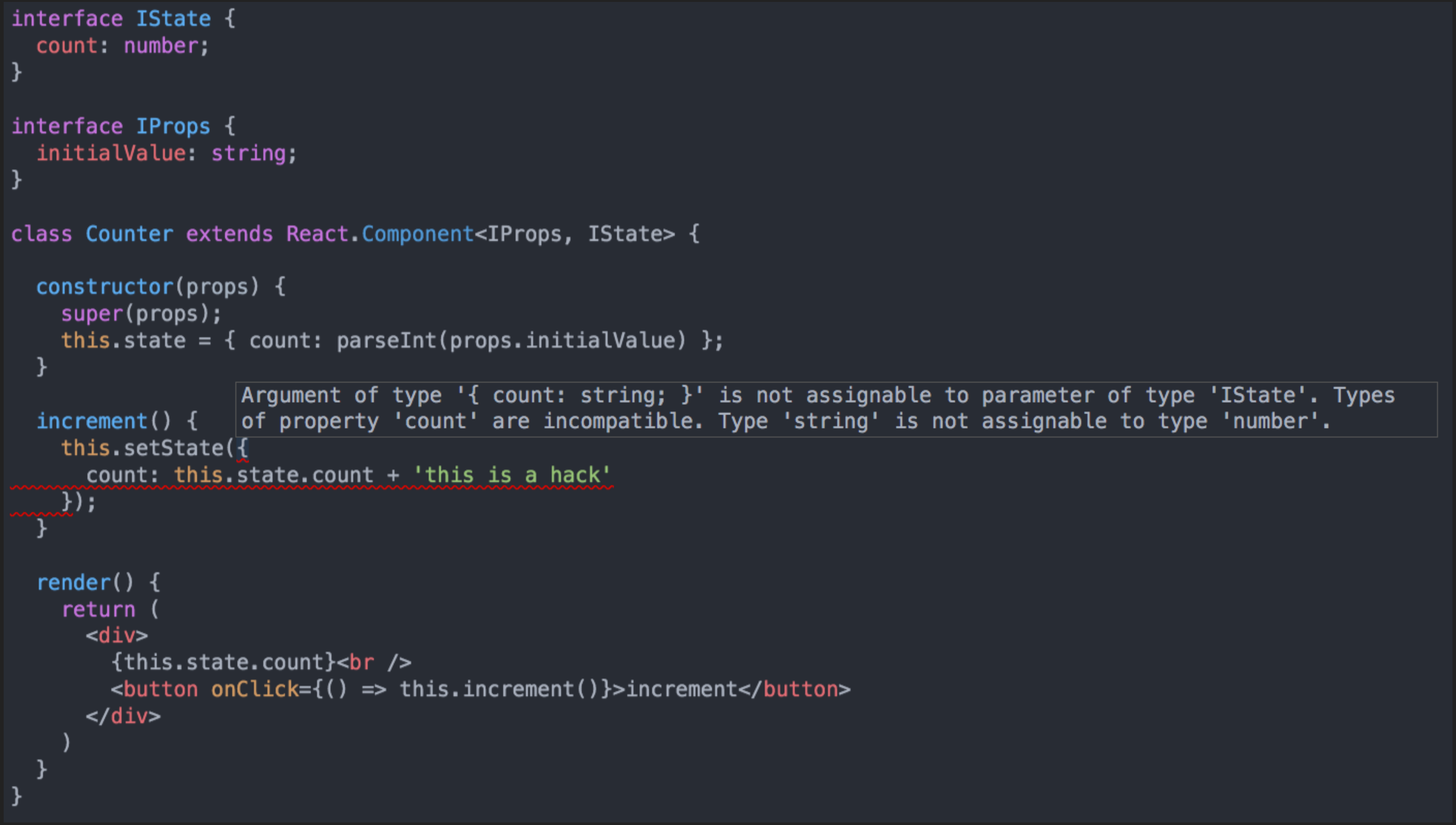
LET'S TYPE CHECK OUR COUNTER

Now, you can try to put something else than a number on our counter, it won't work



```
interface IState {
  count: number;
}

interface IProps {
  initialValue: string;
}

class Counter extends React.Component<IProps, IState> {

  constructor(props) {
    super(props);
    this.state = { count: parseInt(props.initialValue) };
  }

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

  render() {
    return (
      <div>
        {this.state.count}<br />
        <button onClick={() => this.increment()}>increment</button>
      </div>
    )
  }
}
```

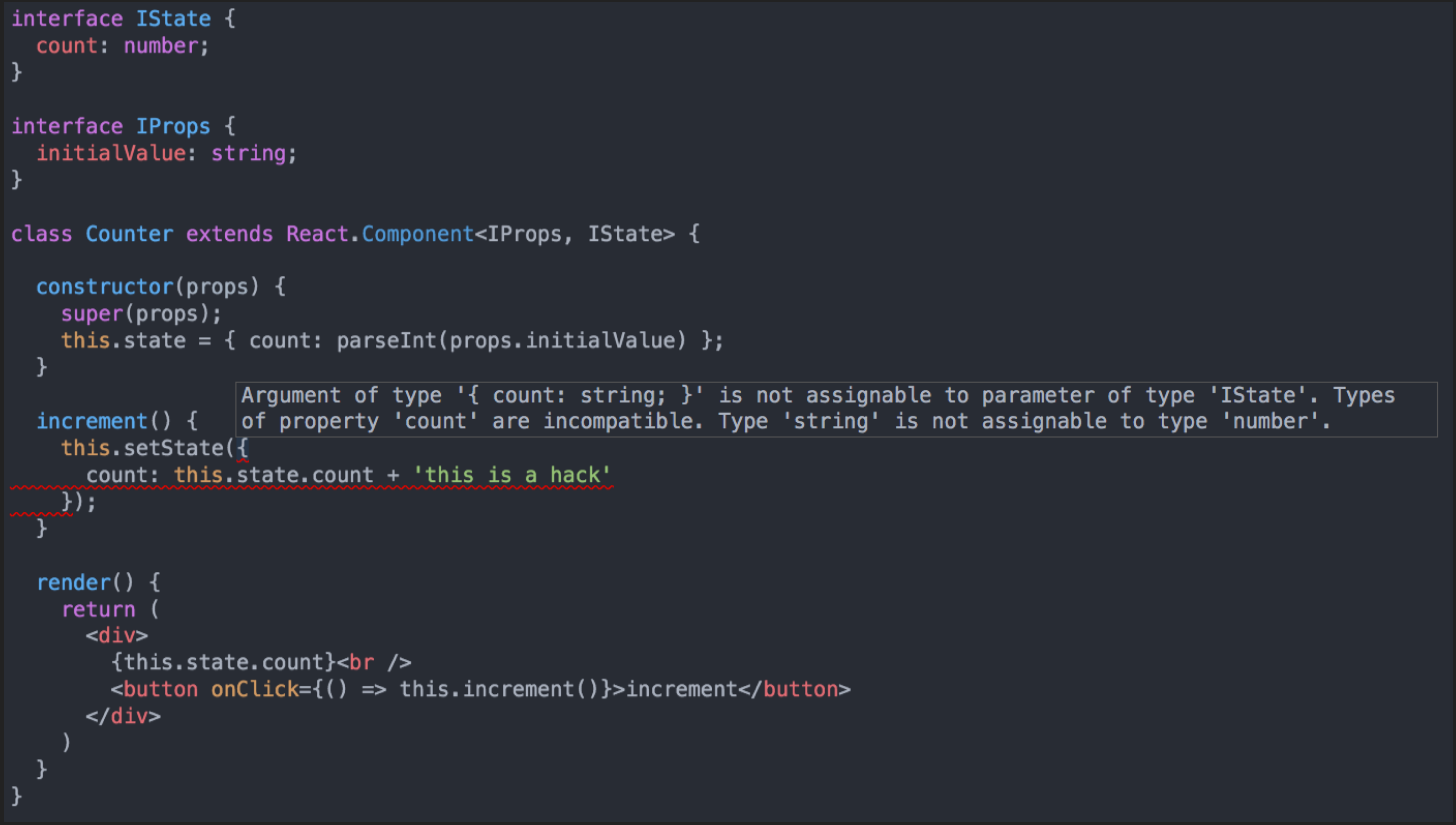
LET'S TYPE CHECK OUR COUNTER



```
interface IState {
  count: number;
}

interface IProps {
  initialValue: string;
}

class Counter extends React.Component<IProps, IState> {

  constructor(props) {
    super(props);
    this.state = { count: parseInt(props.initialValue) };
  }

  increment() {
    this.setState({
      count: this.state.count + 'this is a hack'
    });
  }

  render() {
    return (
      <div>
        {this.state.count}<br />
        <button onClick={() => this.increment()}>increment</button>
      </div>
    )
  }
}
```

```
Argument of type '{ count: string; }' is not assignable to parameter of type 'IState'. Types
of property 'count' are incompatible. Type 'string' is not assignable to type 'number'.
```

LET'S TYPE CHECK OUR COUNTER



```
interface IState {
  count: number;
}

interface IProps {
  initialValue: string;
}

class Counter extends React.Component<IProps, IState> {

  constructor(props) {
    super(props);
    this.state = { count: parseInt(props.initialValue) };
  }

  increment() {
    this.setState({
      count: this.state.count + 'this is a hack'
    });
  }

  render() {
    return (
      <div>
        {this.state.count}<br />
        <button onClick={() => this.increment()}>increment</button>
      </div>
    )
  }
}
```

```
Argument of type '{ count: string; }' is not assignable to parameter of type 'IState'. Types
of property 'count' are incompatible. Type 'string' is not assignable to type 'number'.
```

Type checking = peace of mind

DATA FLOW

- Data flows from parent to children components
- To pass values to a child, we use props
- There only is a one way data-binding

```
const initialItems = [
  {idx: 0, label: 'learn react',    done: false},
  {idx: 1, label: 'try out redux',  done: false},
  {idx: 2, label: 'be positive',    done: true}
];

interface IItem {
  idx: number;
  label: string;
  done: boolean;
}

interface ITodoListProps {
  initialItems: IItem[];
}

interface ITodoListState {
  items: IItem[];
}

class TodoList extends React.Component<ITodoListProps, ITodoListState> {

  constructor(props) {
    super(props);
    this.state = { items: this.props.initialItems };
  }

  renderItem(item: IItem) {
    return <TodoItem key={item.idx} item={item} />
  }

  render() {
    return (
      <ul>
        {this.state.items.map(this.renderItem.bind(this))}
      </ul>
    );
  }
}

render(<TodoList initialItems={initialItems} />, document.getElementById('react-mount'));
```

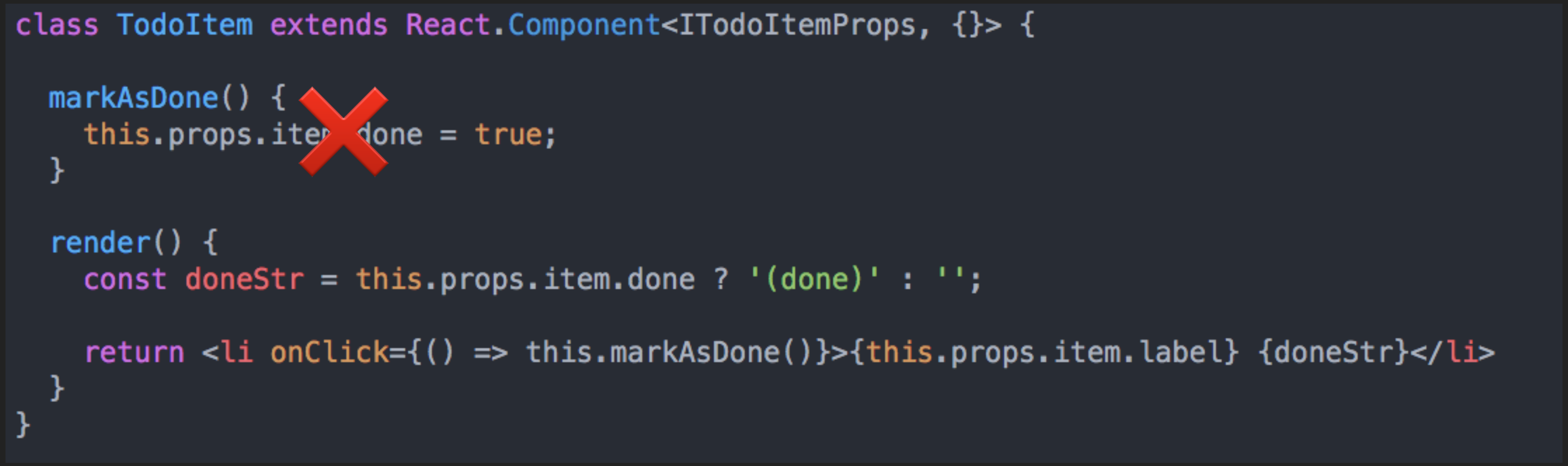
Parent -> child
communication with property

```
interface ITodoItemProps {
  item: IItem;
}

class TodoItem extends React.Component<ITodoItemProps, {}> {
  render() {
    const doneStr = this.props.item.done ? '(done)' : '';

    return <li>{this.props.item.label} {doneStr}</li>
  }
}
```

DATA FLOW

But now, I want to mark an item as "done" when I click on it.
Props are immutable, so I can't do this.props.item.done = true;

How can I warn the parent to mark my item as "done" ?



```
class TodoItem extends React.Component<ITodoItemProps, {}> {

  markAsDone() {
    this.props.item.done = true;
  }

  render() {
    const doneStr = this.props.item.done ? '(done)' : '';

    return <li onClick={() => this.markAsDone()}>{this.props.item.label} {doneStr}</li>
  }
}
```

DATA FLOW

But now, I want to mark an item as "done" when I click on it.
Props are immutable, so I can't do this.props.item.done = true;

How can I warn the parent to mark my item as "done" ?

EVENTS

Data flows down, events flow up

DATA FLOW

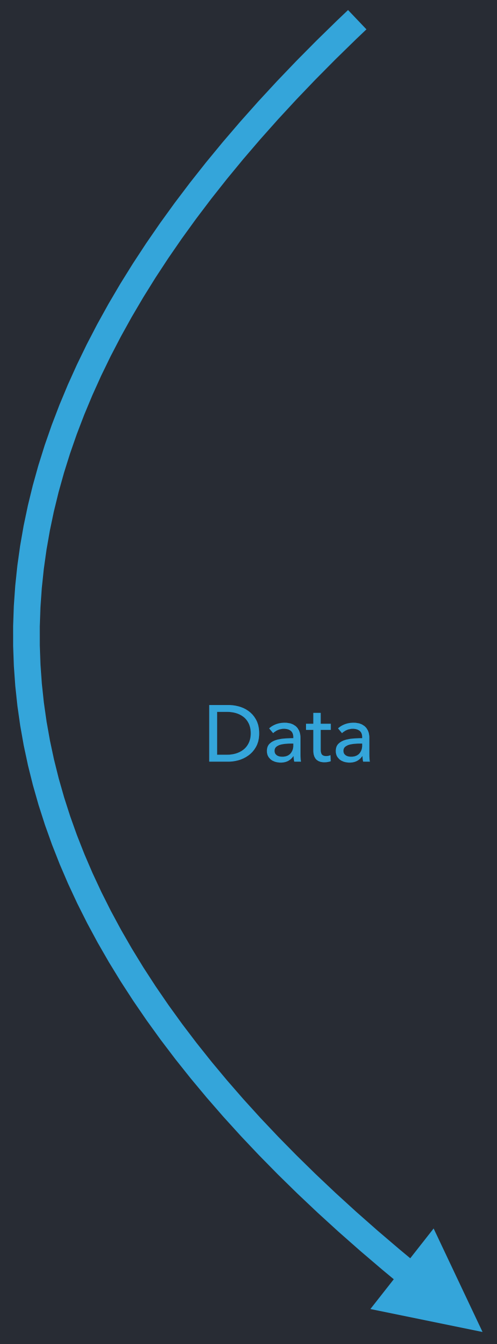
1. On TodoList component, define an action we want to execute when clicking on a TodoItem
2. Pass this action as a prop to every TodoItem
3. When click is triggered on TodoItem, call the action that was passed as prop

DATA FLOW

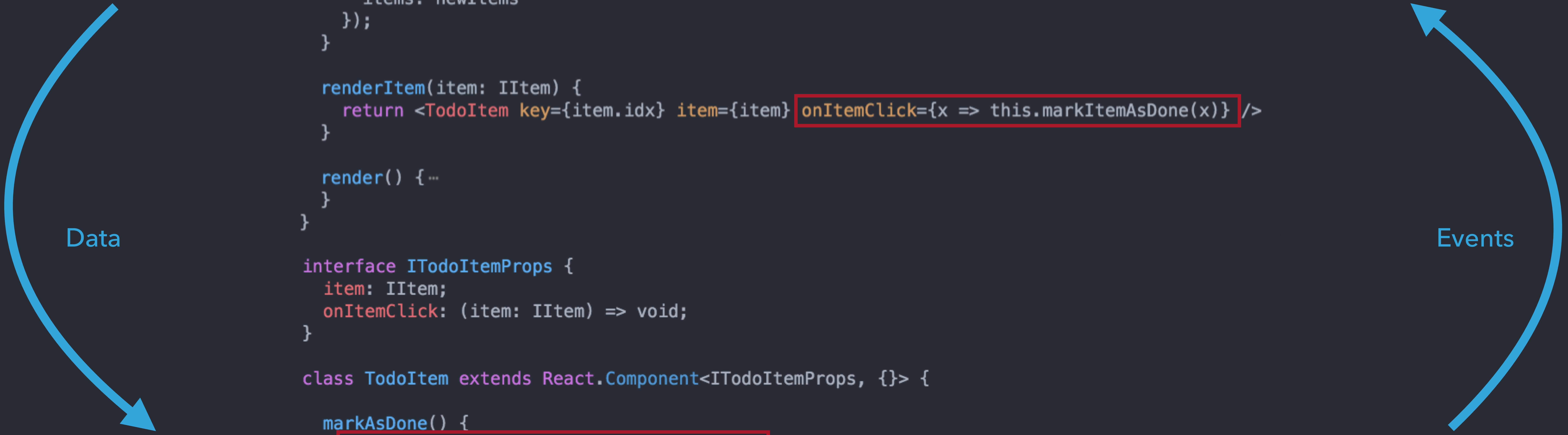
1. On TodoList component, define an action we want to execute when clicking on a TodoItem
2. Pass this action as a prop to every TodoItem
3. When click is triggered on TodoItem, call the action that was passed as prop

DELEGATION

```
class TodoList extends React.Component<ITodoListProps, ITodoListState> {

  constructor(props) {⋯
  }

  markItemAsDone(item: IItem) {
    let newItems = this.state.items;
    newItems[item.idx].done = true;
    this.setState({
      items: newItems
    });
  }

  renderItem(item: IItem) {
    return <TodoItem key={item.idx} item={item} onItemClick={x => this.markItemAsDone(x)} />
  }

  render() {⋯
  }
}

interface ITodoItemProps {
  item: IItem;
  onItemClick: (item: IItem) => void;
}

class TodoItem extends React.Component<ITodoItemProps, {}> {

  markAsDone() {
    this.props.onItemClick(this.props.item);
  }

  render() {
    const doneStr = this.props.item.done ? '(done)' : '';

    return <li onClick={() => this.markAsDone()}>{this.props.item.label} {doneStr}</li>
  }
}
```



```
class TodoList extends React.Component<ITodoListProps, ITodoListState> {

  constructor(props) {…
  }

  markItemAsDone(item: IItem) {
    let newItems = this.state.items;
    newItems[item.idx].done = true;
    this.setState({
      items: newItems
    });
  }

  renderItem(item: IItem) {
    return <TodoItem key={item.idx} item={item} onItemClick={x => this.markItemAsDone(x)} />
  }

  render() {…
  }
}

interface ITodoItemProps {
  item: IItem;
  onItemClick: (item: IItem) => void;
}

class TodoItem extends React.Component<ITodoItemProps, {}> {

  markAsDone() {
    this.props.onItemClick(this.props.item);
  }

  render() {
    const doneStr = this.props.item.done ? '(done)' : '';

    return <li onClick={() => this.markAsDone()}>{this.props.item.label} {doneStr}</li>
  }
}
```

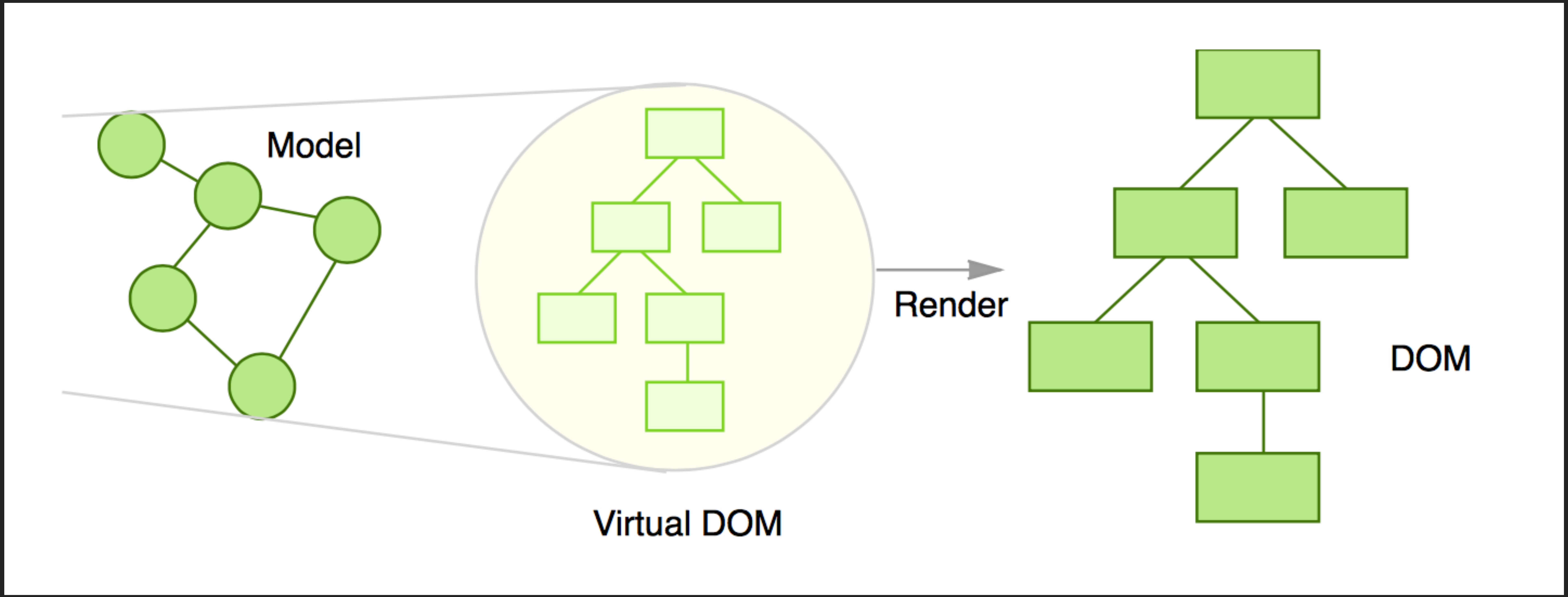


```
class TodoList extends React.Component<ITodoListProps, ITodoListState> {

  constructor(props) {⋯
  }

  markItemAsDone(item: IItem) {
    let newItems = this.state.items;
    newItems[item.idx].done = true;
    this.setState({
      items: newItems
    });
  }

  renderItem(item: IItem) {
    return <TodoItem key={item.idx} item={item} onItemClick={x => this.markItemAsDone(x)} />
  }

  render() {⋯
  }
}

interface ITodoItemProps {
  item: IItem;
  onItemClick: (item: IItem) => void;
}

class TodoItem extends React.Component<ITodoItemProps, {}> {

  markAsDone() {
    this.props.onItemClick(this.props.item);
  }

  render() {
    const doneStr = this.props.item.done ? '(done)' : '';

    return <li onClick={() => this.markAsDone()}>{this.props.item.label} {doneStr}</li>
  }
}
```

VIRTUAL DOM

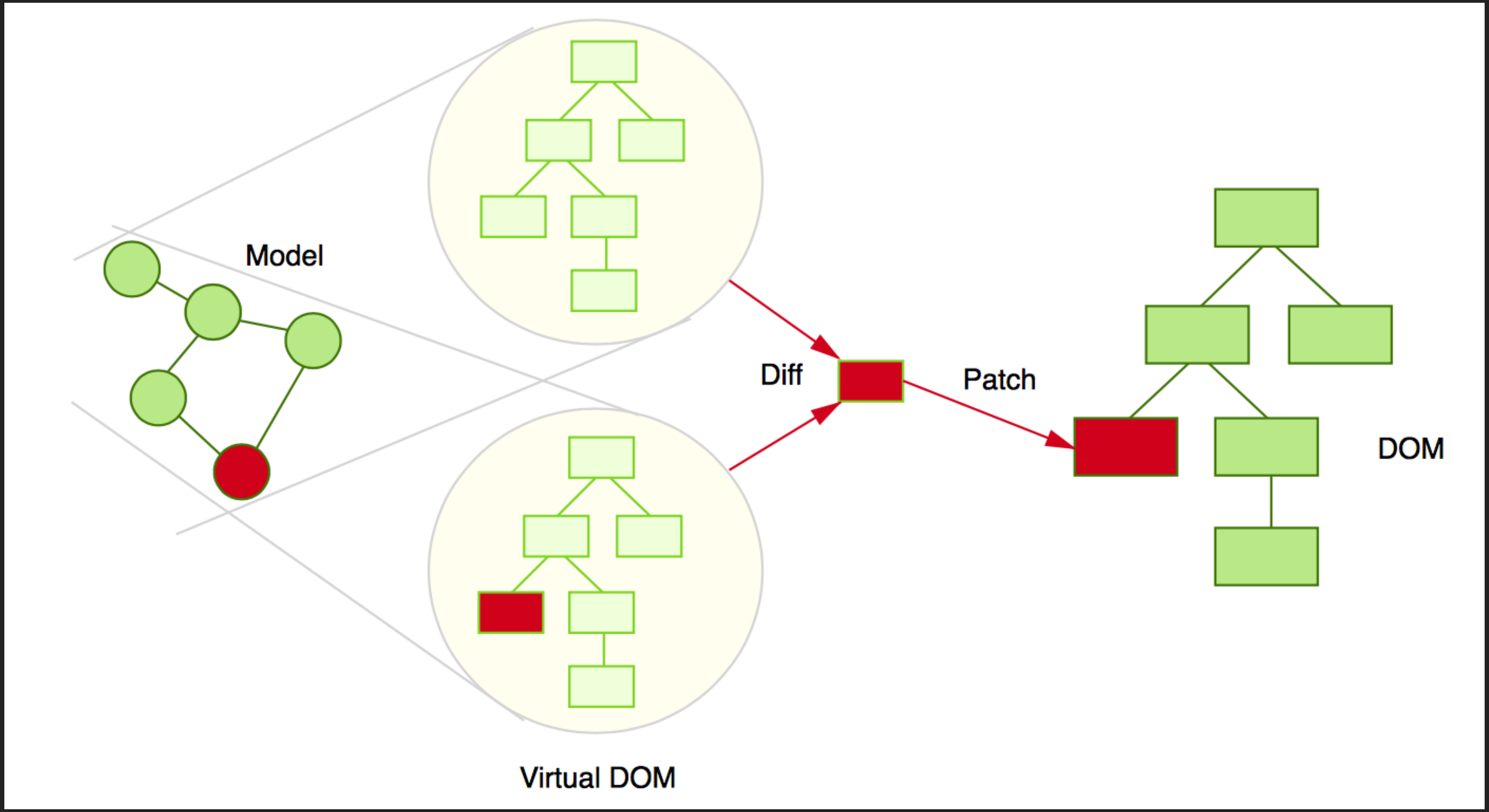
- DOM manipulations are (very) slow
- Use of Virtual DOM allows us to re-render the DOM only when necessary
- Re-render only needed subtrees (diff between old state and new state)
- Allow server-side rendering
- => Better performance

VIRTUAL DOM



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

VIRTUAL DOM



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

OKAY, THAT SEEMS COOL. BUT WHY?

- UI became **predictable and deterministic**
- Predictable => easier to understand and test
- Components are **reusable** and more **maintainable**
- Use of virtual DOM allows very **good performances**

Hey, it's okay for a todo-list. But what about a *real* application?

Server response

Hey, it's okay for a todo-list. But what about a *real* application?

Server response

Hey, it's okay for a todo-list. But what about a *real* application?

Cached data

Server response

Temporary form data

Hey, it's okay for a todo-list. But what about a *real* application?

Cached data

Server response

Temporary form data

Hey, it's okay for a todo-list. But what about a *real* application?

UI state

Cached data

Server response

Temporary form data

Loader during async tasks

Hey, it's okay for a todo-list. But what about a *real* application?

UI state

Cached data

Server response

Temporary form data

Loader during async tasks

Hey, it's okay for a todo-list. But what about a *real* application?

UI state

Cached data

Error message

State management is going to be a nightmare!

State management is going to be a nightmare!

😈

Flux Relay Redux

to the rescue

WHAT IS REDUX?

"Redux is a predictable state container for JavaScript apps."

http://redux.js.org

WHAT IS REDUX?

State container?

There is one unique state container called **store**. It contains your application state.

Same store = same application render

WHAT IS REDUX?

Predictable?

State is immutable

We can get a new state by dispatching actions

WHAT IS REDUX?

Predictable?

State is **immutable**

We can get a **new state** by dispatching **actions**

"Mark item 3 as read" =>

```
{
    type: "MARK_READ",
    itemId: 3
}
```

WHAT IS REDUX?

Predictable?

Actions are consumed by **reducers**

Reducers take an action, the actual state and **return a new state**

```
function reducer(state, action) => newState
```

WHAT IS REDUX?

Predictable?

Reducers are pure functions.

Same parameters = same result. Always.

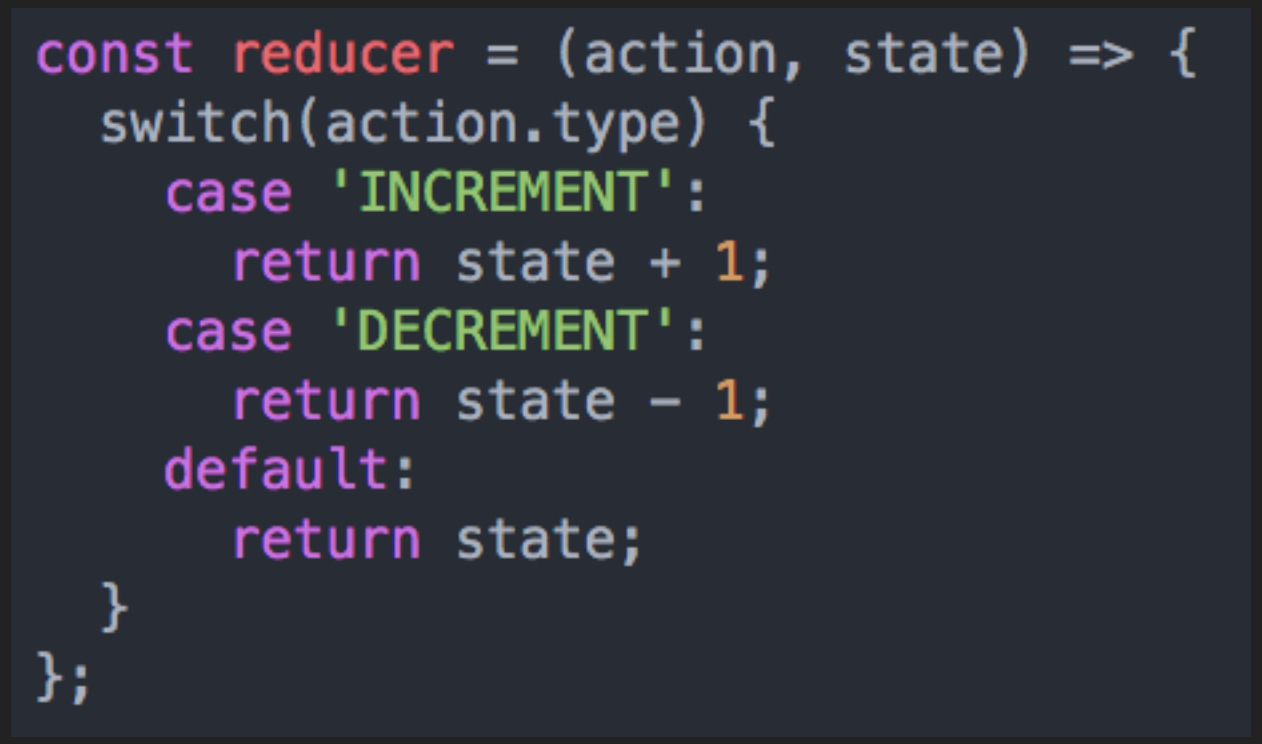
A LITTLE EXAMPLE?

Let's build a counter (yeah, again)!

- We have two actions: increment and decrement
- If another (or an unknown) action is passed, do nothing (i.e. return the current state)

A LITTLE EXAMPLE?

The reducer

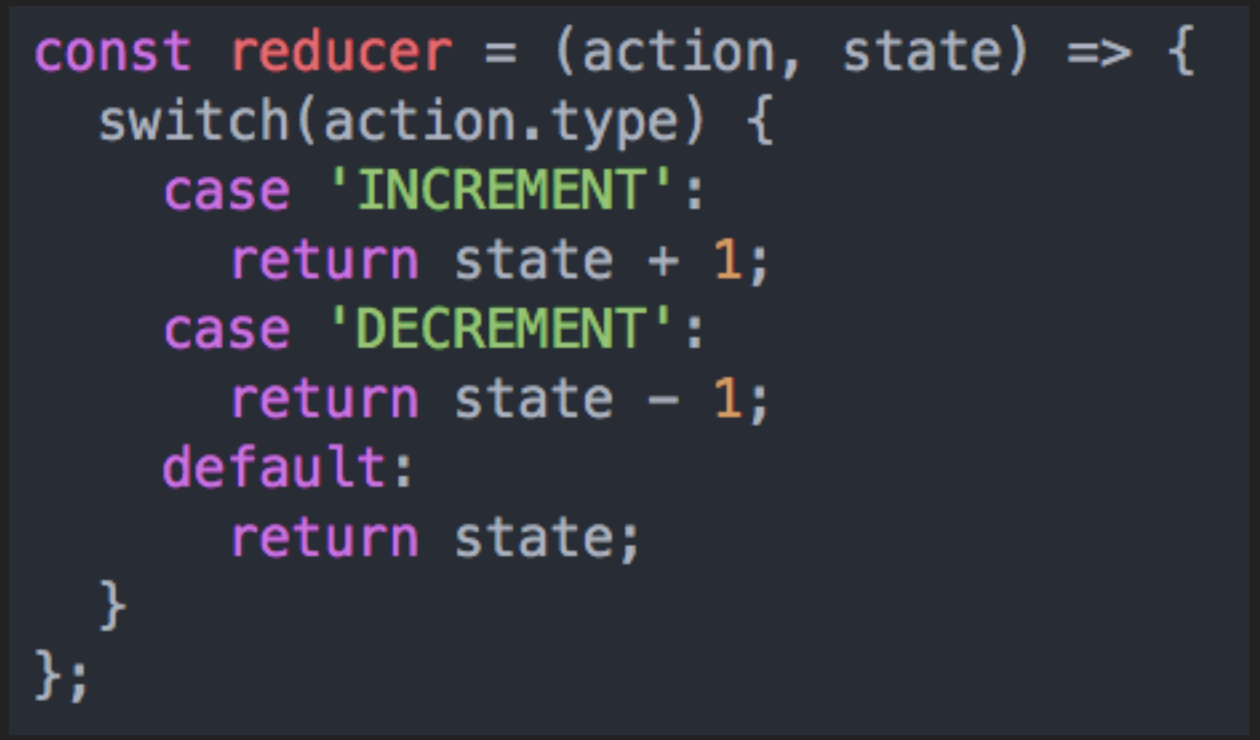


```
const reducer = (action, state) => {
  switch(action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      return state;
  }
};
```

A LITTLE EXAMPLE?

The reducer

Hum, on the first call, state will be undefined, isn't it?

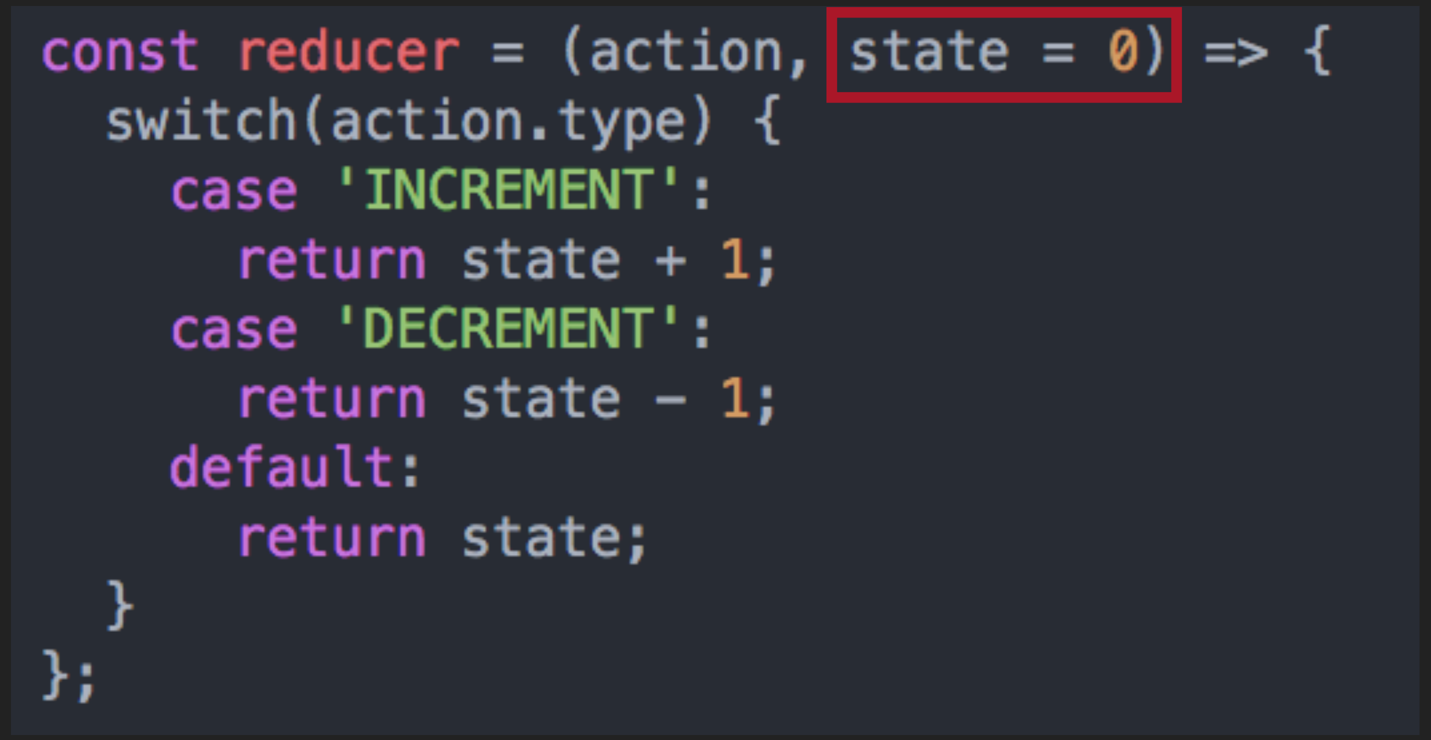


```
const reducer = (action, state) => {
  switch(action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      return state;
  }
};
```

A LITTLE EXAMPLE?

The reducer

ES6 default values to the rescue!
This is the way to define our app initial state.

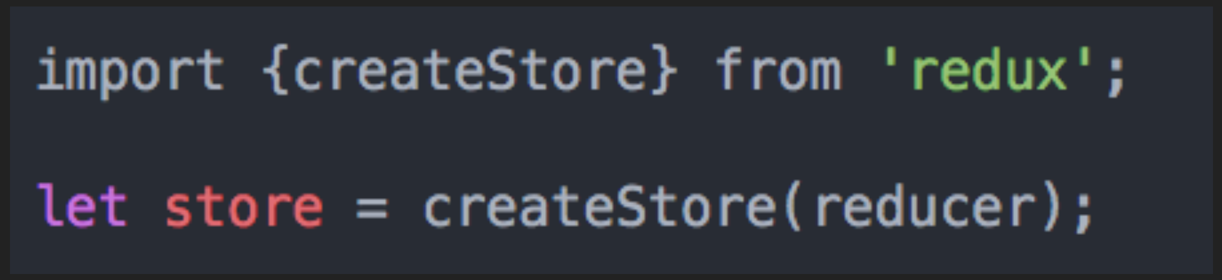


```
const reducer = (action, state = 0) => {
  switch(action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      return state;
  }
};
```

A LITTLE EXAMPLE?

The store

We create the store by passing our reducer to Redux **createStore()** function

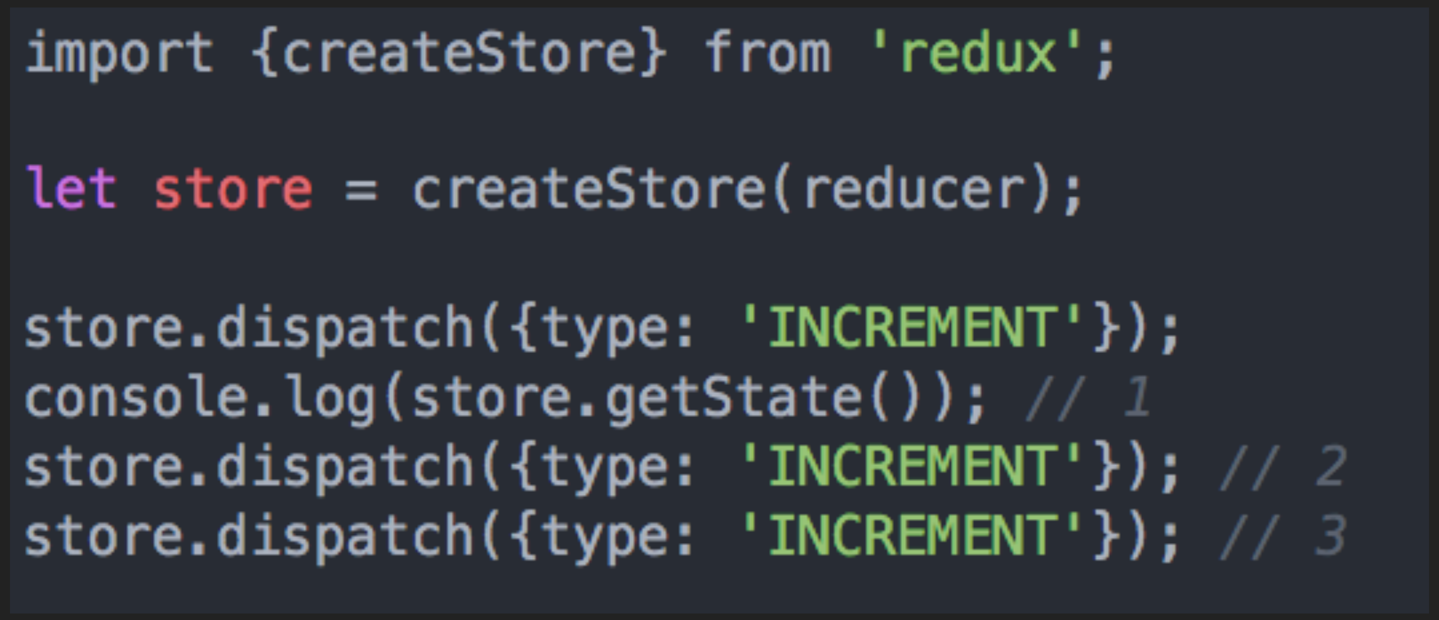


```
import {createStore} from 'redux';

let store = createStore(reducer);
```

A LITTLE EXAMPLE?

And we are done!

Yep, **Redux is that simple**. If you got this, you understood all the basics of Redux.



```
import {createStore} from 'redux';

let store = createStore(reducer);

store.dispatch({type: 'INCREMENT'});
console.log(store.getState()); // 1
store.dispatch({type: 'INCREMENT'}); // 2
store.dispatch({type: 'INCREMENT'}); // 3
```

A LITTLE EXAMPLE?

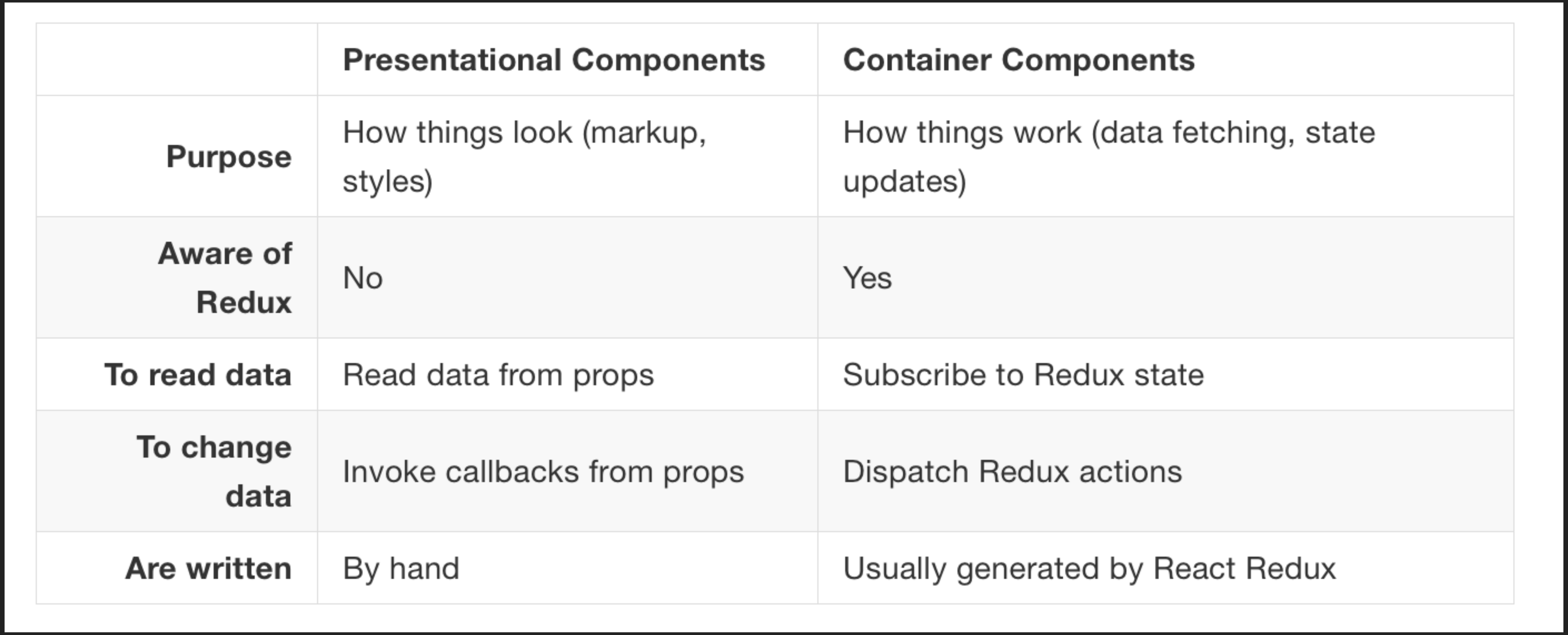
Hey, don't trick me!
There's not a single line of React in your example.

😐

REDUX × REACT

- Use `react-redux` package
- Relies on presentational and container components

REDUX × REACT: PRESENTATIO-WHAT?

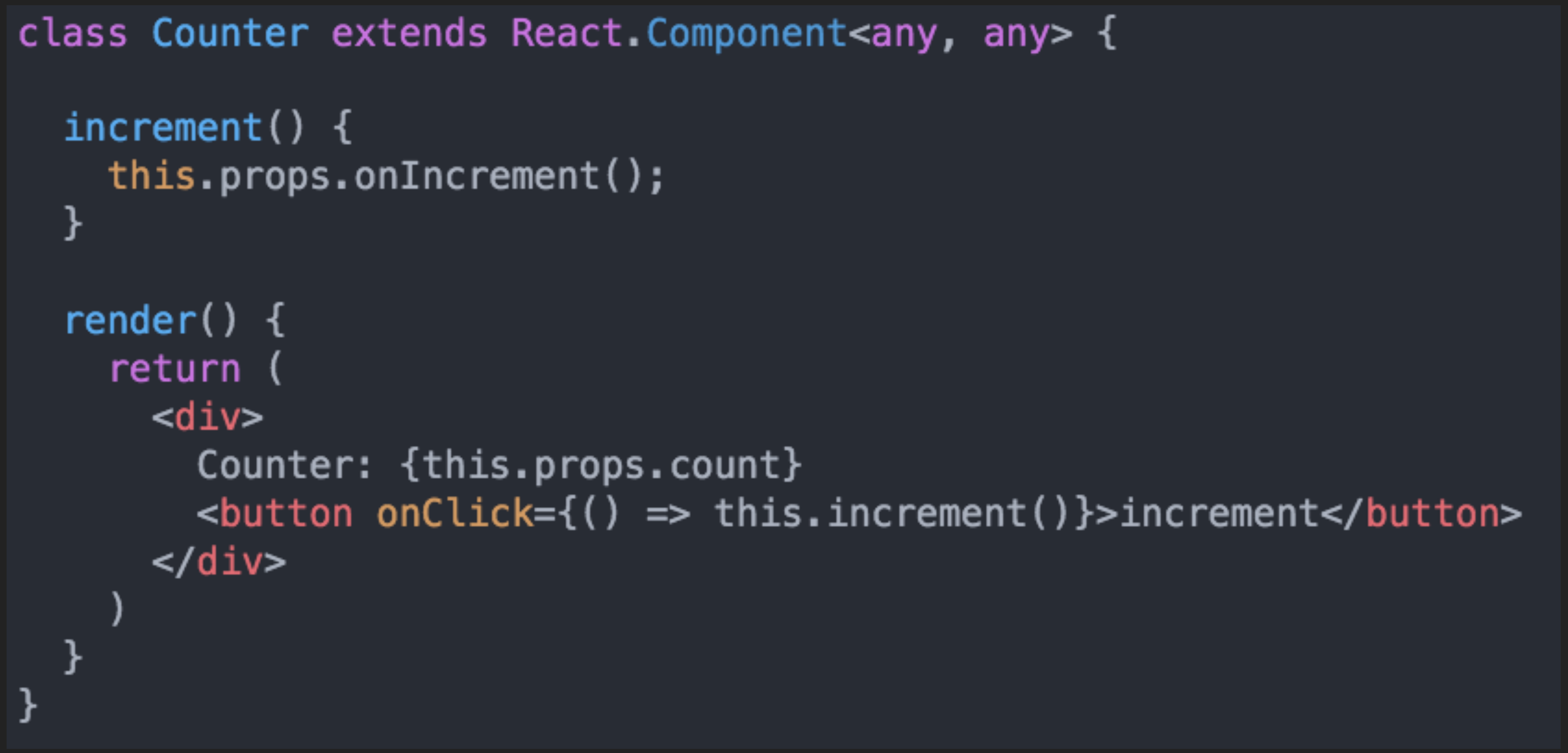


	Presentational Components	Container Components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of Redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions
Are written	By hand	Usually generated by React Redux

Credit: http://redux.js.org/docs/basics/UsageWithReact.html

REDUX × REACT: PRESENTATIONAL COMPONENTS

- Data from **props**
- Invoke callbacks (from props) to change data – **Delegation**
- **Rarely have a state** (in such case, just UI state, no data)



```
class Counter extends React.Component<any, any> {

  increment() {
    this.props.onIncrement();
  }

  render() {
    return (
      <div>
        Counter: {this.props.count}
        <button onClick={() => this.increment()}>increment</button>
      </div>
    )
  }
}
```

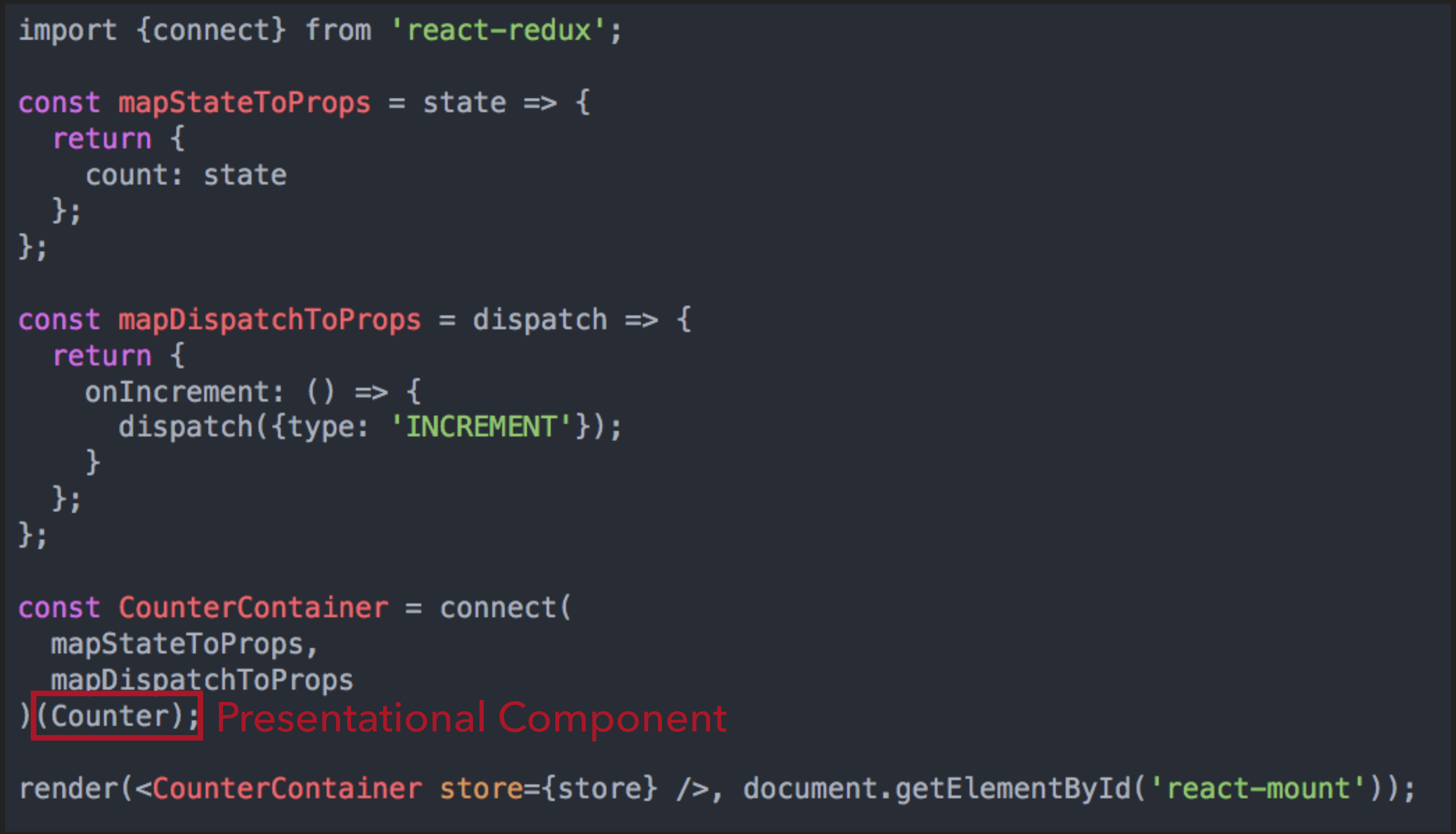
REDUX × REACT: CONTAINER COMPONENTS

- Are **generated** (by react-redux) to work with a presentational component
- Generated from two functions that describes how to:
  - map the **state** to child component props
  - map **dispatch calls** to child components props
- Need **Redux store** passed as property

REDUX × REACT: CONTAINER COMPONENTS

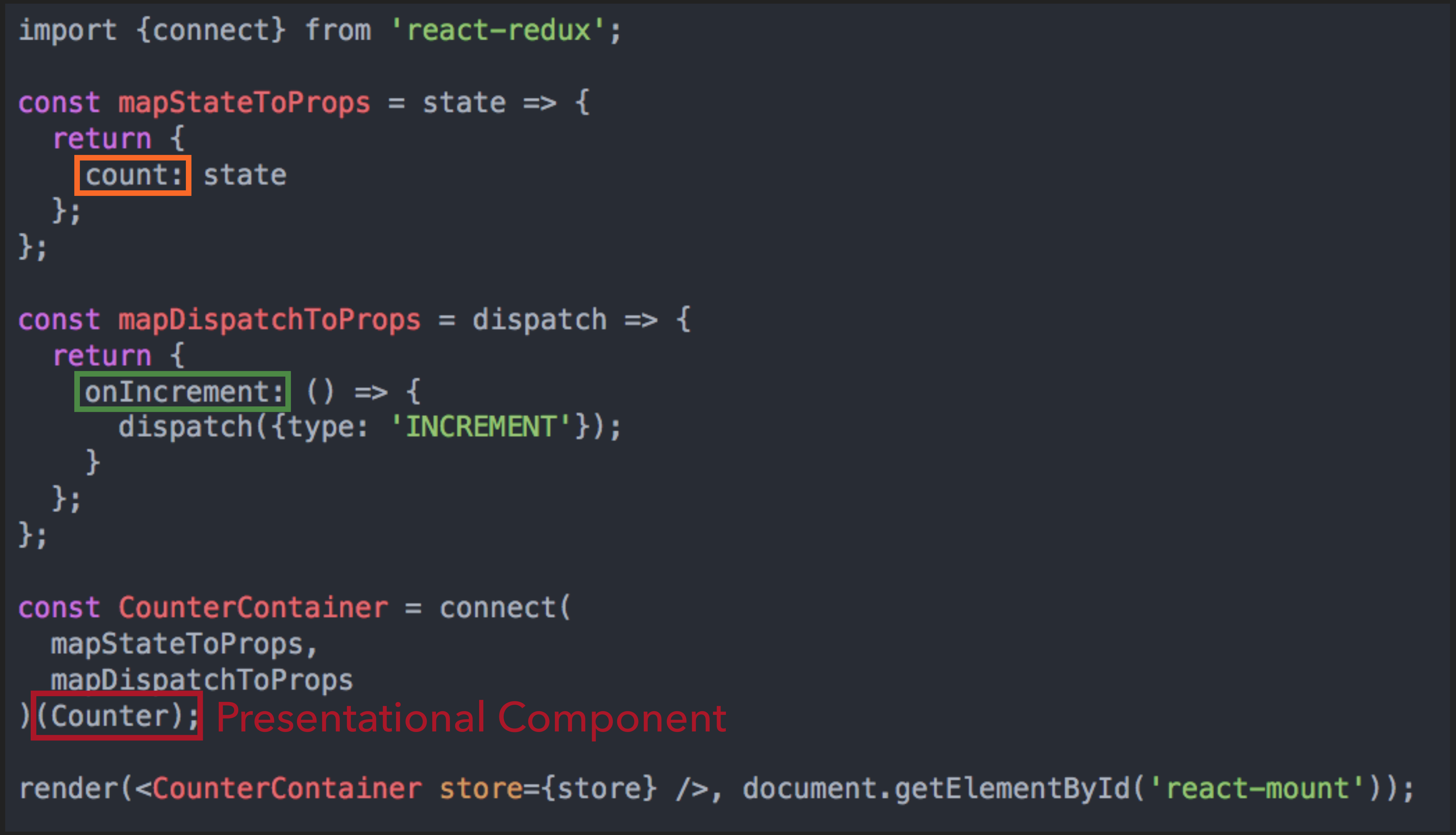
```
import {connect} from 'react-redux';

const mapStateToProps = state => {
  return {
    count: state
  };
};

const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => {
      dispatch({type: 'INCREMENT'});
    }
  };
};

const CounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

render(<CounterContainer store={store} />, document.getElementById('react-mount'));
```

REDUX × REACT: CONTAINER COMPONENTS



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

const mapStateToProps = state => {
  return {
    count: state
  };
};

const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => {
      dispatch({type: 'INCREMENT'});
    }
  };
};

const CounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter); Presentational Component

render(<CounterContainer store={store} />, document.getElementById('react-mount'));
```

REDUX × REACT: CONTAINER COMPONENTS

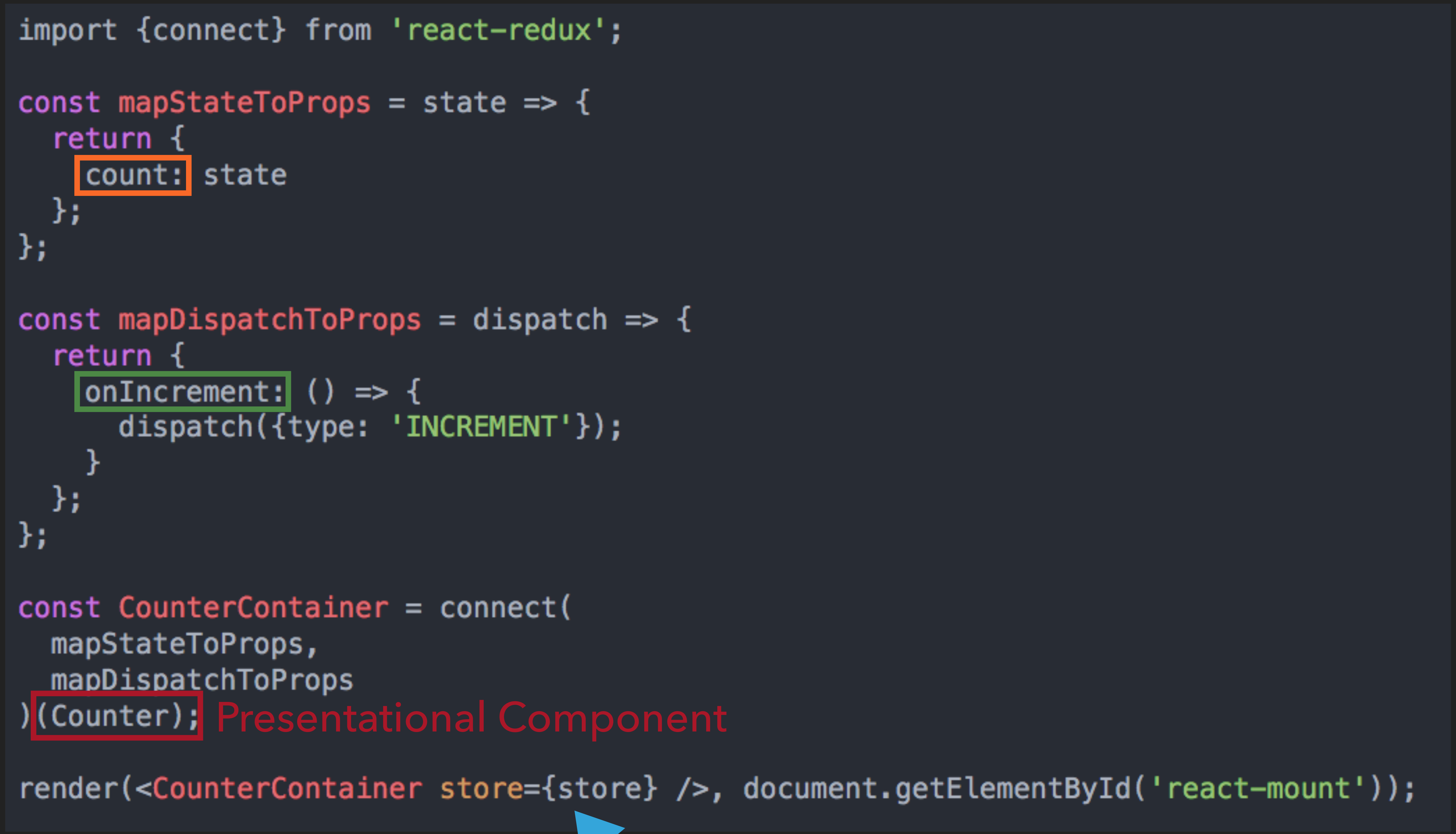


```
import {connect} from 'react-redux';

const mapStateToProps = state => {
  return {
    count: state
  };
};

const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => {
      dispatch({type: 'INCREMENT'});
    }
  };
};

const CounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

render(<CounterContainer store={store} />, document.getElementById('react-mount'));
```

Presentational Component

`Counter` component will have `count` and `onIncrement` properties

REDUX × REACT: CONTAINER COMPONENTS



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

const mapStateToProps = state => {
  return {
    count: state
  };
};

const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => {
      dispatch({type: 'INCREMENT'});
    }
  };
};

const CounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);

render(<CounterContainer store={store} />, document.getElementById('react-mount'));
```

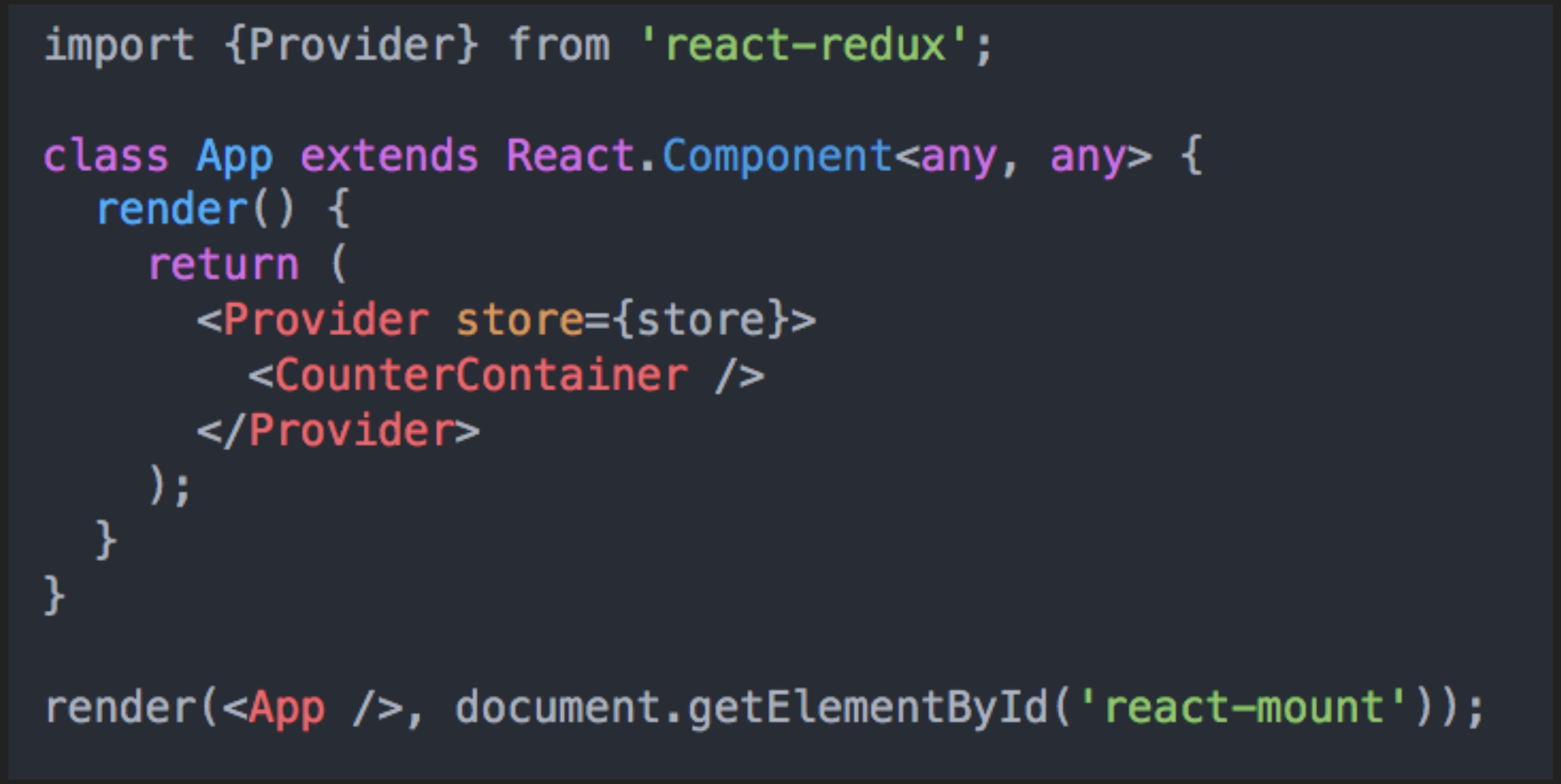
Redux Store

`Counter` component will have `count` and `onIncrement` properties

REDUX × REACT: CONTAINER COMPONENTS

Passing `store` to every container components will rapidly become impossible to handle properly.

That's why react-redux exposes a `Provider` component that handles it for us.



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

class App extends React.Component<any, any> {
  render() {
    return (
      <Provider store={store}>
        <CounterContainer />
      </Provider>
    );
  }
}

render(<App />, document.getElementById('react-mount'));
```

A LITTLE MORE: ASYNC

Once again, that's cool for a todo-list, but what about async calls with Redux?

- See redux-thunk middleware
- Notion of async action, that dispatches actions (request start, request end, etc.)

More: http://redux.js.org/docs/advanced/AsyncActions.html

RESOURCES

- React doc: https://facebook.github.io/react/
- Holy Redux Bible: http://redux.js.org/index.html

Thanks for listening!

Questions?