React & Reactive Programming

Ziming Miao @ Traintracks.io

Introduction

- 苗梓铭 | Ziming Miao
- 2011-2014 FE Lead of Wandoujia
- 2014-present Senior Engineer of Traintracks.io
- Working on UI development, big data analysis, etc.

「今有雉、兔同笼，上有三十五头，下九十四足。问雉、兔各几何？」

–「孙子算经」

「上置三十五头，下置九十四足。
半其足，得四十七。以少减多」

$$\begin{aligned} x + y &= 35 \\ 2x + 4y &= 94 \end{aligned}$$

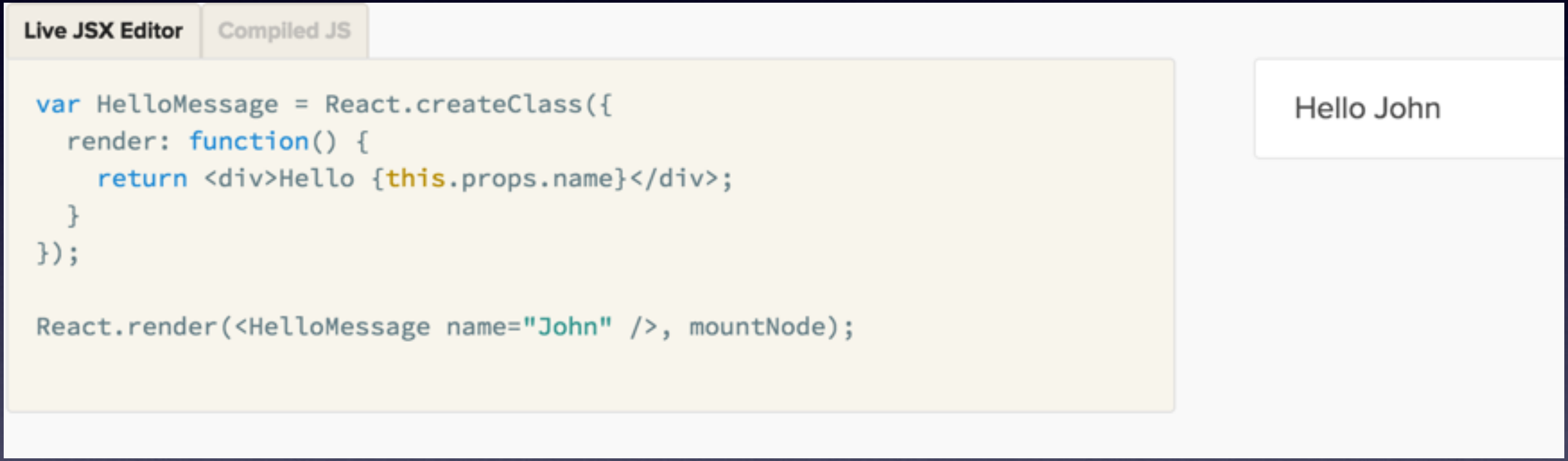
The way we’re doing rendering is <u>inefficient</u>

Problem w/ DOM

- HTML standard is quite loose, makes parser slow in exchange to be error-proof.
- DOM is stateful, but difficult to manage changes or get notified.
- Direct DOM manipulations are boring and repeated.

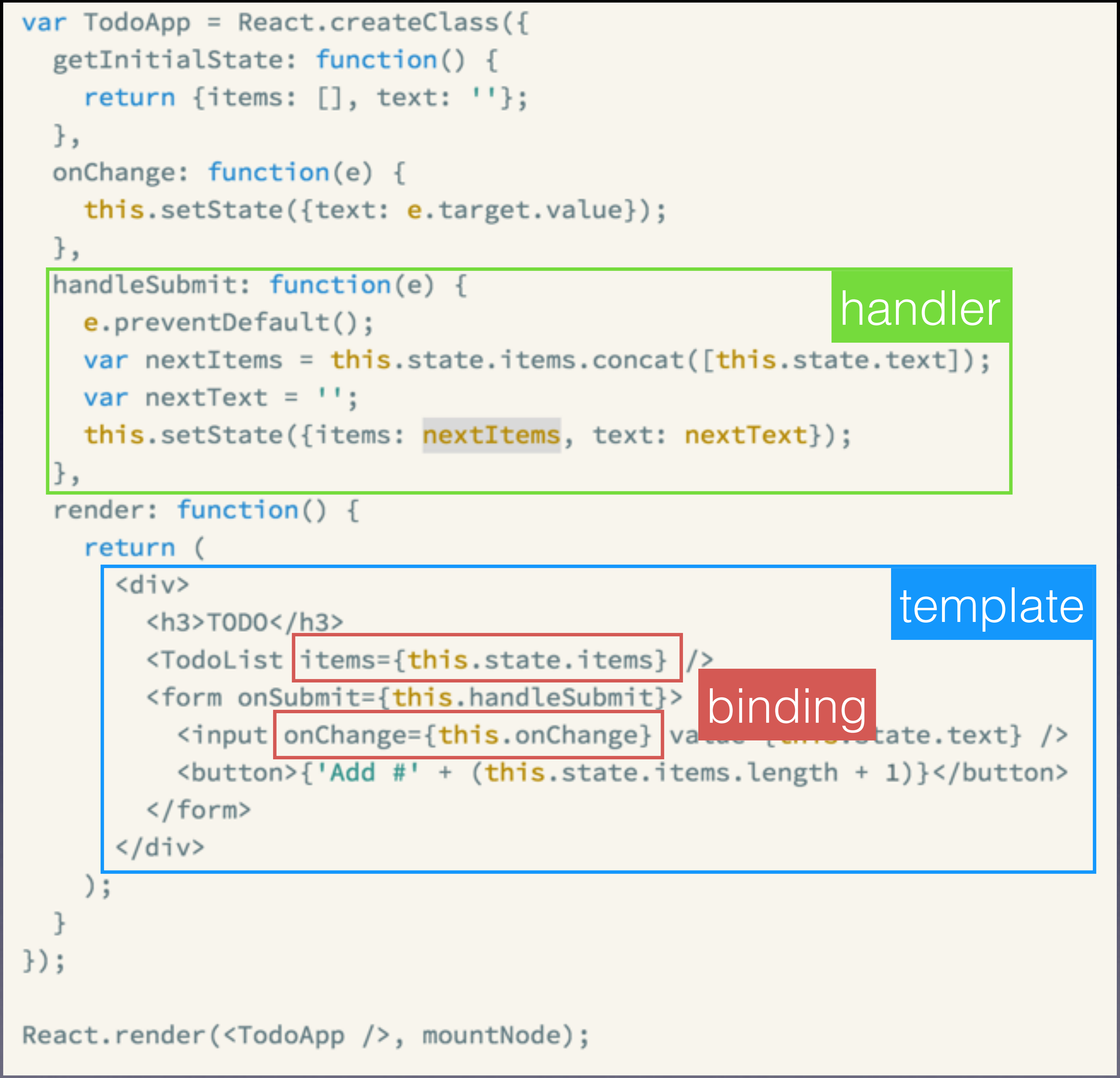
DOM used to be the only way,
but not the best

What is <u>React</u>?

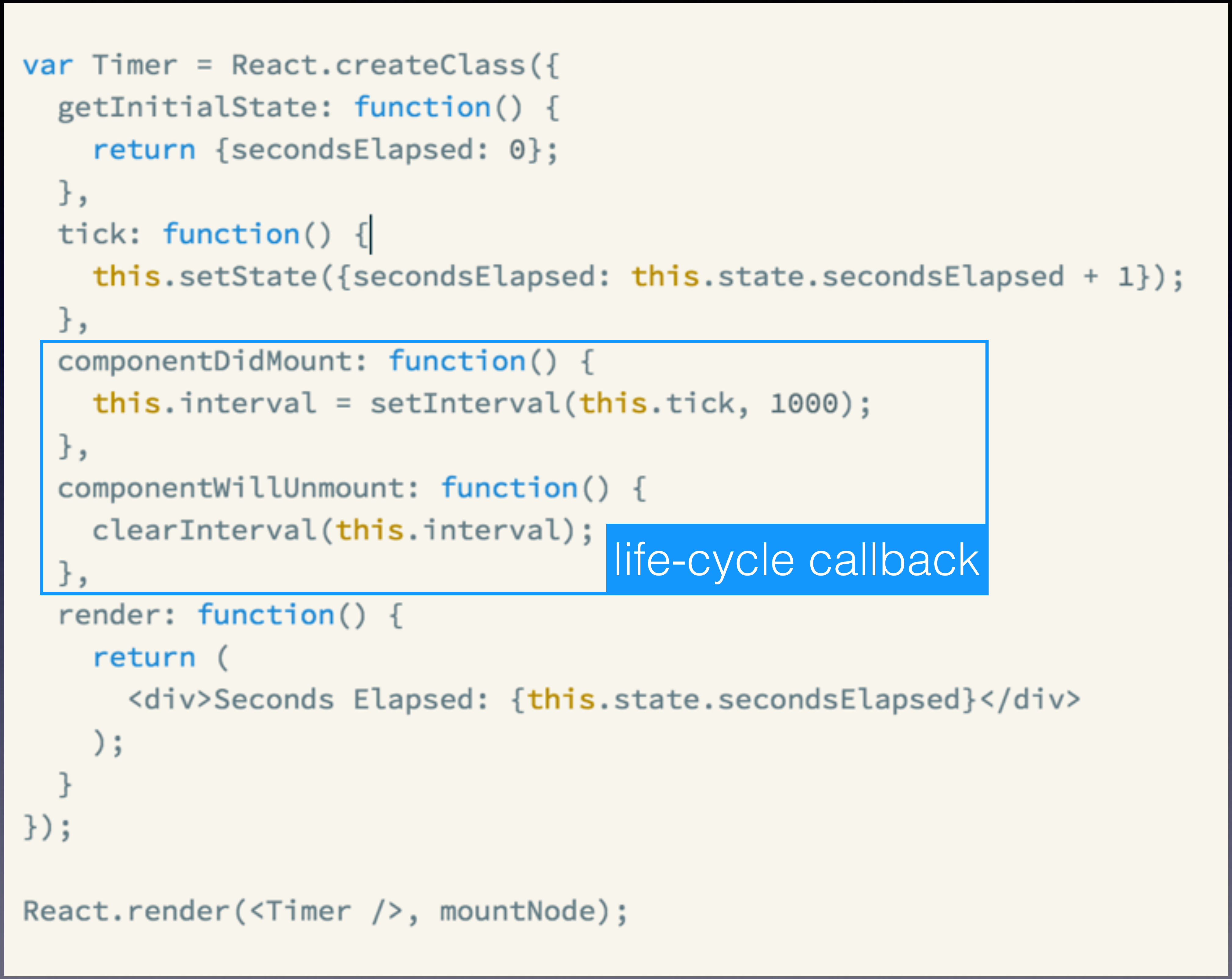


```
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

React.render(<HelloMessage name="John" />, mountNode);
```

```
Hello John
```



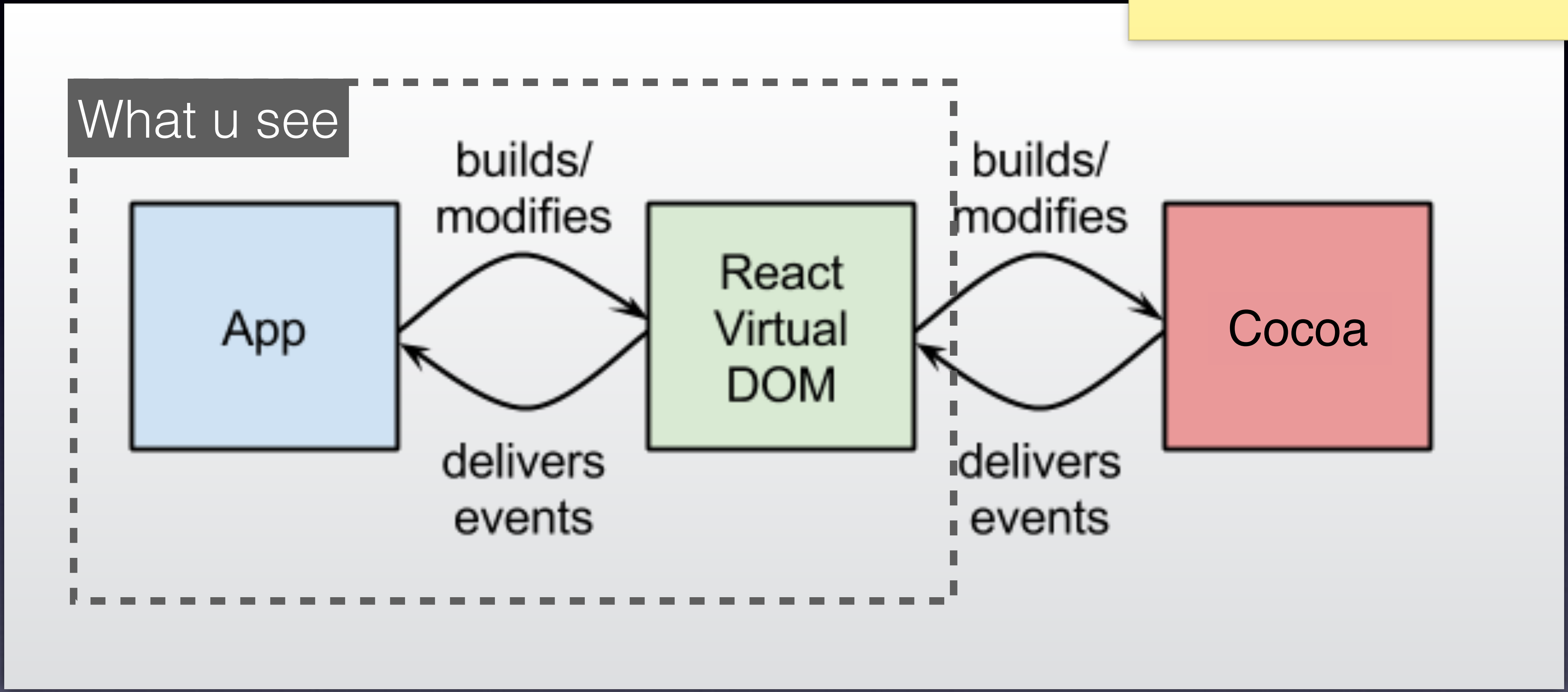
```
var TodoApp = React.createClass({
  getInitialState: function() {
    return {items: [], text: ''};
  },
  onChange: function(e) {
    this.setState({text: e.target.value});
  },
  handleSubmit: function(e) {
    e.preventDefault();
    var nextItems = this.state.items.concat([this.state.text]);
    var nextText = '';
    this.setState({items: nextItems, text: nextText});
  },
  render: function() {
    return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input onChange={this.onChange} value={this.state.text} />
          <button>{'Add #' + (this.state.items.length + 1)}</button>
        </form>
      </div>
    );
  }
});

React.render(<TodoApp />, mountNode);
```



```
var Timer = React.createClass({
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  },
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  render: function() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
});

React.render(<Timer />, mountNode);
```

View layer framework,
or
“Abstraction of drawing layer”

1. replace underlying drawing layer
2. async rendering



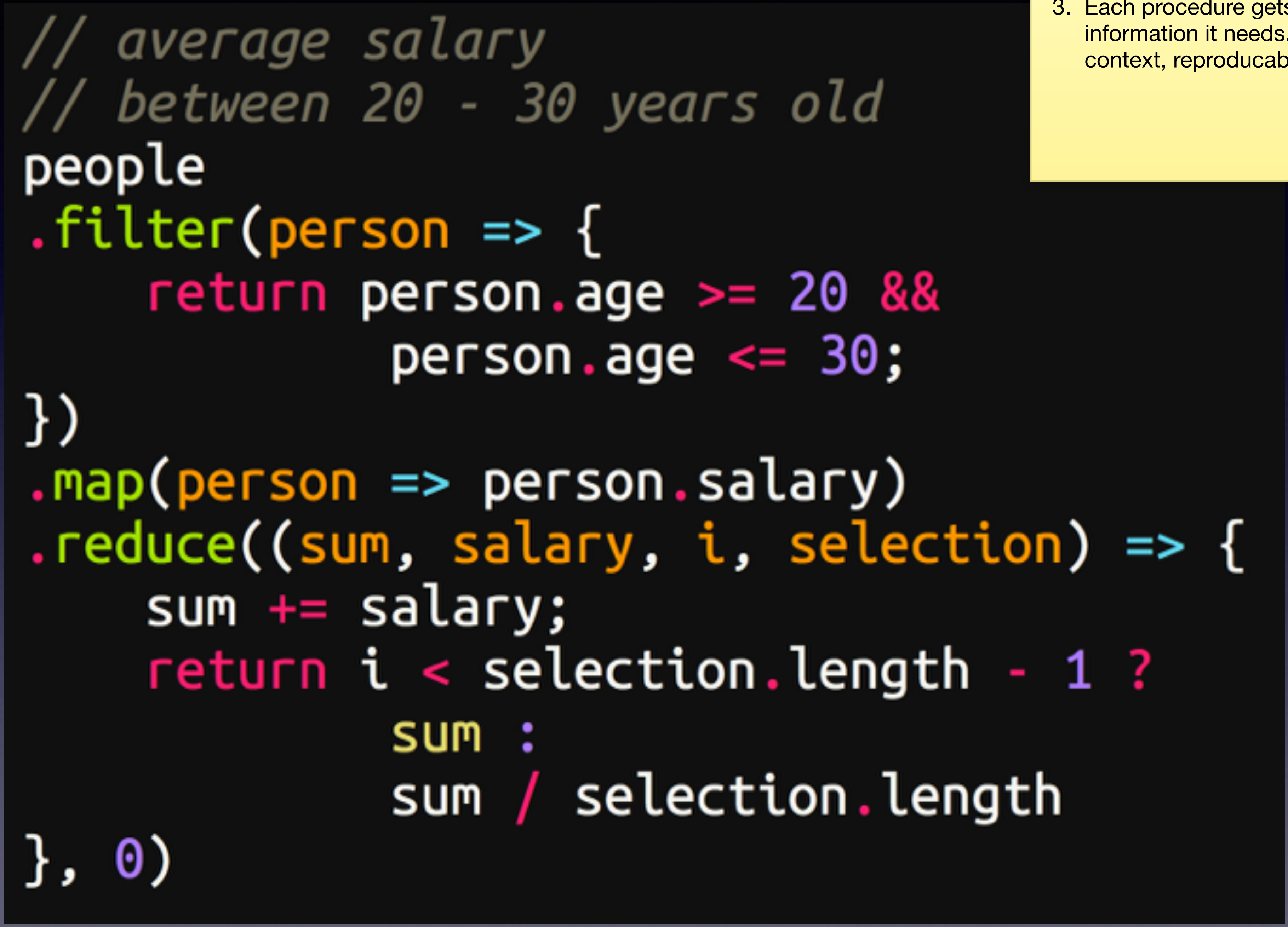
http://jiyinyiyong.u.qiniudn.com/react-demystified/r2.png

Reactive

"… This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow."

–Wikipedia: Reactive programming

http://en.wikipedia.org/wiki/Reactive_programming



```
// average salary
// between 20 - 30 years old
people
.filter(person => {
    return person.age >= 20 &&
            person.age <= 30;
})
.map(person => person.salary)
.reduce((sum, salary, i, selection) => {
    sum += salary;
    return i < selection.length - 1 ?
            sum :
            sum / selection.length
}, 0)
```

1. this is for a specific time point
2. What if people changes by time?
3. Each procedure gets all the information it needs. No global context, reproducable

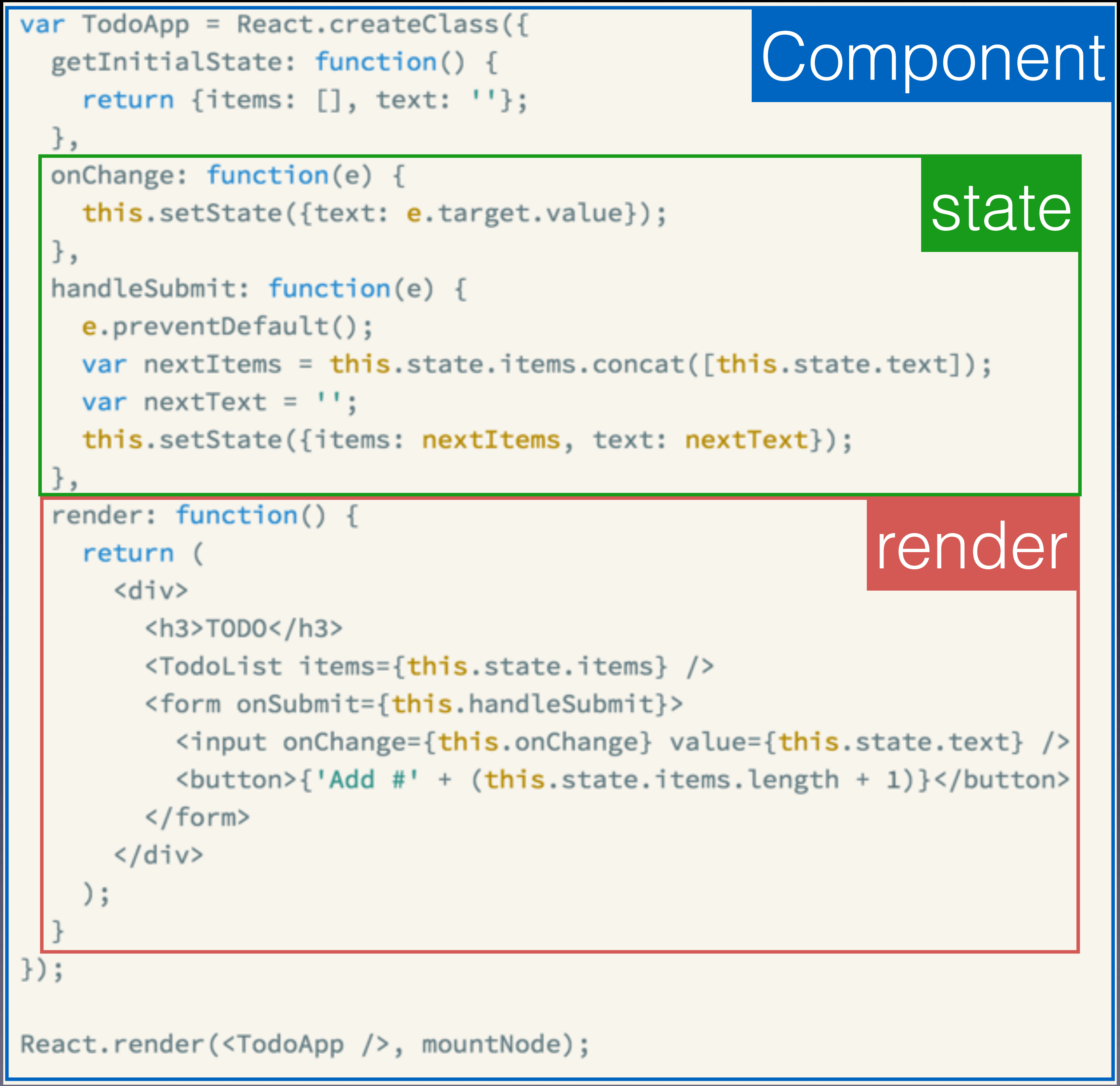
JavaScript enables you processing data functionally

$$f(x) = x$$

```
f(x) = x => A
```

```
A = (f, x)
```

```
Component = (render, state)
```



```
var TodoApp = React.createClass({
  getInitialState: function() {
    return {items: [], text: ''};
  },
  onChange: function(e) {
    this.setState({text: e.target.value});
  },
  handleSubmit: function(e) {
    e.preventDefault();
    var nextItems = this.state.items.concat([this.state.text]);
    var nextText = '';
    this.setState({items: nextItems, text: nextText});
  },
  render: function() {
    return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input onChange={this.onChange} value={this.state.text} />
          <button>{'Add #' + (this.state.items.length + 1)}</button>
        </form>
      </div>
    );
  }
});

React.render(<TodoApp />, mountNode);
```

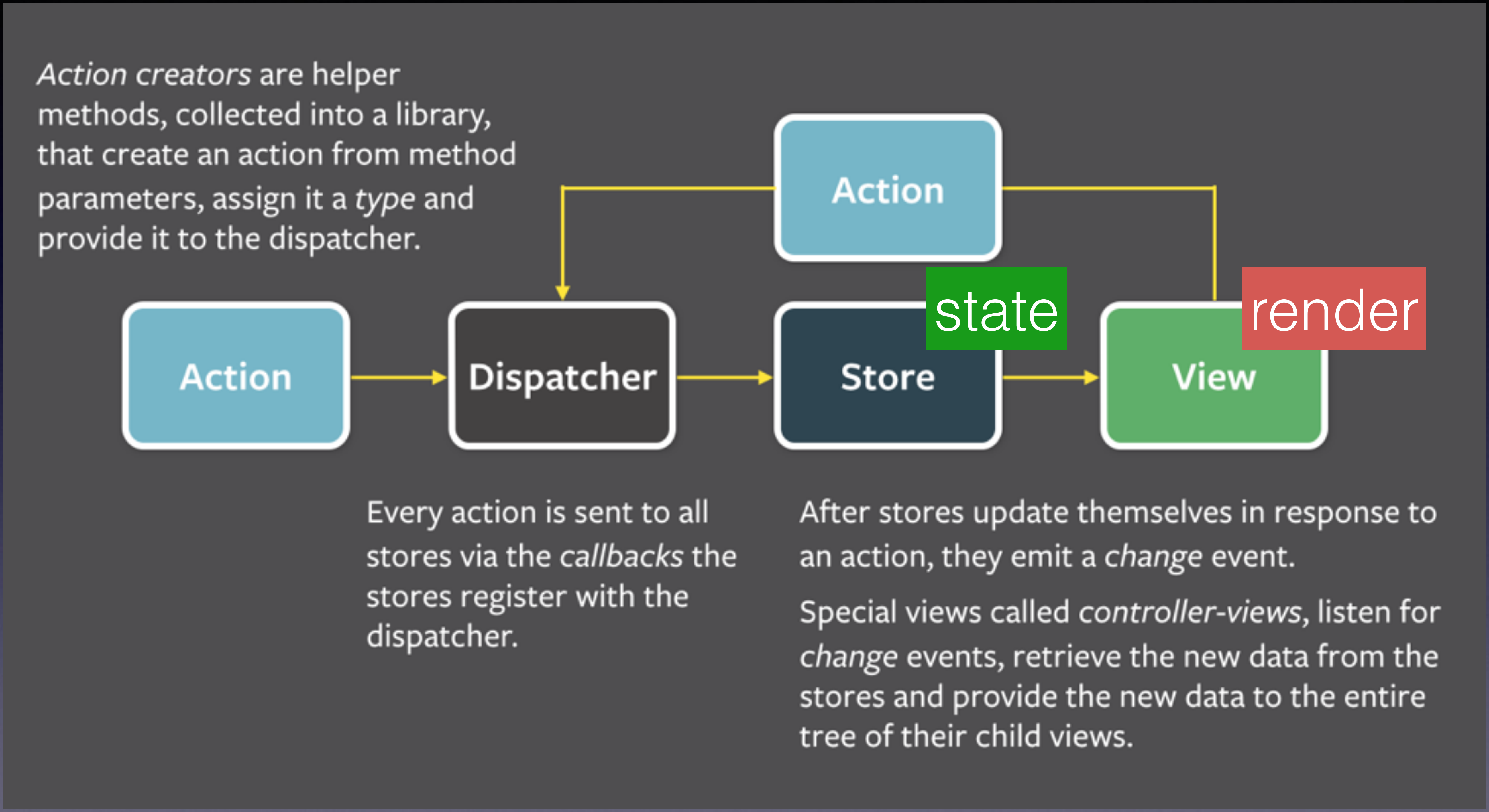
Features

- Predictable, advanced in testing & tooling
- Unidirectional data flow, leans on central data storage
- All functional programming benefits

How could you imagine to implement App-level undo & redo functions by DOM?

GoyaPixel

Unidirectional data flow



Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

https://facebook.github.io/flux/docs/overview.html#content

Core concepts

- Abstraction of drawing layer, portable
- Reactive, unidirectional data-flow
- Functional friendly, immutable, multithreading
- Balance between productivity and performance