What is React?
What is webpack?
Why should I care?

Who am I?

- Hi! I’m **Sean Timm**.
- Tweet me: **@seantimm**
    - *#ThisIsMadness*
    - *#BestPresenterEver*
- Blog: **<u>escreturn.com</u>**

Audience Pulse

- Who has had exposure to webpack?
- React?
- JavaScript?

JavaScript is *Everywhere*.

- StackOverflow’s 2016 developer survey places JavaScript as “the most commonly used programming language on earth.”
- “Even Back-End developers are more likely to use it than any other language.”
- http://esc.re/1RV4qQC

JavaScript has *Changed*.

- “Modern” language design started in 1999 with focused development starting in 2009.
- In 1999, ES3 was finalized. IE6 was ES3 compliant.
  - This is the JavaScript many of us learned and use today.
  - For the MS tech stack, this is the equivalent of still using Classic ASP and VB6.

ES6 / ES2015 is a Standard.

- Chrome - 96%
- Firefox - 90%
- MS Edge (v14) - 85%,
- Safari 9 - 54% (but WebKit @ 98%)

What’s new in ES2015?

- Modules
- Classes
- Scoped Variables
- Arrow Functions
- Template Strings
- Spread Operator
- Destructuring
- Default Values
- Rest Parameters
- for-of Iterator
- Symbols
- Promises
- … and more

Modules

```
// my-module.js
export function DoSomething() {
  console.log('Did it')
}

export function DoSomethingElse() {
  console.log('No')
}

// use-module.js
import * as myFuncs from './my-module'
import { doSomething } from './my-module'

myFuncs.doSomething()
myFuncs.doSomethingElse()

doSomething()
```

Classes

```
class MyBase {
  constructor(name) { this.name = name }
}
class MyClass extends MyBase {
  constructor(name) {
    super(name)
  }
  sayMyName() {
    console.log(this.name)
  }
}

var inst = new MyClass('example')
inst.sayMyName()
```

Scoped Variables

```
function scopeIt() {
  let x = 'foo'
  {
    const x = 'bar'
    //x = 'noreassign'
  }
  //let x = 'nodupe'
  console.log(x)
}

scopeIt()
```

Arrow Functions

```
const inventory = {
  prefix: 'Item: ',
  items: ['Thing 1', 'Thing 2'],

  listItems() {
    this.items.forEach(item => {
      console.log(this.prefix + item)
    })
  }
}

inventory.listItems()
```

Template Strings

```
const name = 'Sean'

// Note the backtick
console.log(`${name} loves
  string interpolation
  ${'so much'.toUpperCase()}`)
```

Spread Operator

```
// The spread operator expands
// elements of an array
const values = [1, 2, 3]
const moreValues = [...values, 4]

function addEmUp(a, b, c, d) {
  console.log(a+b+c+d)
}

addEmUp(...moreValues)
```

Destructuring

```
var [thing1, , thing2] =
  ['thing1', null, 'thing2']

console.log(thing2)

function outputName({name: x}) {
  console.log(x)
}

var Sean = { isHere: true, name: 'Sean'}
outputName(Sean)
outputName('test')
```

Default Values

```
function printName(name = 'Missing') {
  console.log(name)
}

printName()
printName(undefined)
printName('Sean')
```

Rest Parameters

```
function showIt(prefix, ...items) {
  items.forEach(item => {
    console.log(`${prefix}: ${item}`)
  })
}

showIt('Item', 'Thing 1', 'Thing 2')
showIt('More', ...['This', 'That'])
```

Semicolons Optional

;

The Rise of ES*

- Switching to a living standard
- Yearly, incremental releases
- Feature stages (0 - 4)

Stage 0	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
Strawman	Proposal	Draft	Candidate	Final

ES* Spread Features

- Rest Properties
- Spread Properties

Rest Properties

```
const Sean = {
  name: 'Sean',
  isSpeaking: true
}
const { name, ...otherInfo} = Sean

console.log(`
  name: ${name},
  otherInfo.name: ${otherInfo.name},
  otherInfo.isSpeaking: ${otherInfo.isSpeaking}
`)
```

Spread Properties

```
const otherInfo = {
  name: 'Sean',
  isSpeaking: true
}

let isHere = true
const Sean = { isHere, ...otherInfo}

console.log(`
  name: ${Sean.name},
  isHere: ${Sean.isHere}`)
```

But I have to support IE 8!

- I can relate.
- Let me introduce you to transpiling!

Meet Babel - JavaScript Compiler

- Built out of plugins
- Compose a transformation pipeline
  - Use presets, plugins, or a combination of both

ES2015 Preset

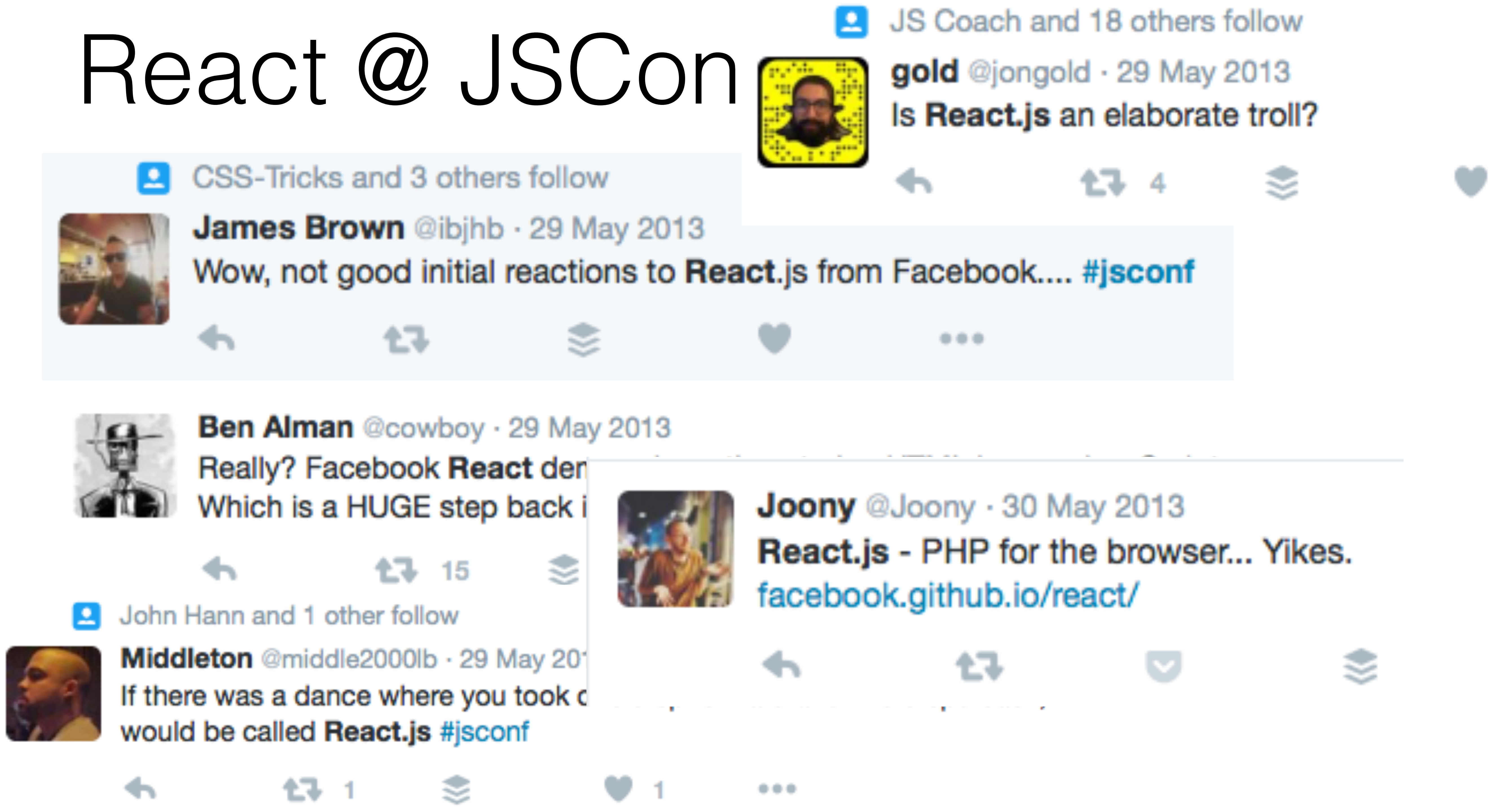
- Composed of individual plugins
- Use the preset… or make your own preset with the plugins you want to use!

React Preset

- **`plugins`**
  - **`transform-react-jsx`**
  - **`transform-flow-strip-types`**
  - **`syntax-flow`**
  - **`syntax-jsx`**
  - **`transform-react-display-name`**

Umm… JSX?

React @ JSCon



JS Coach and 18 others follow

gold @jongold · 29 May 2013

Is **React**.js an elaborate troll?

4

CSS-Tricks and 3 others follow

James Brown @ibjhb · 29 May 2013

Wow, not good initial reactions to **React**.js from Facebook.... #jsconf

Ben Alman @cowboy · 29 May 2013

Really? Facebook **React** den

Which is a HUGE step back i

15

Joony @Joony · 30 May 2013

React.js - PHP for the browser... Yikes.

facebook.github.io/react/

John Hann and 1 other follow

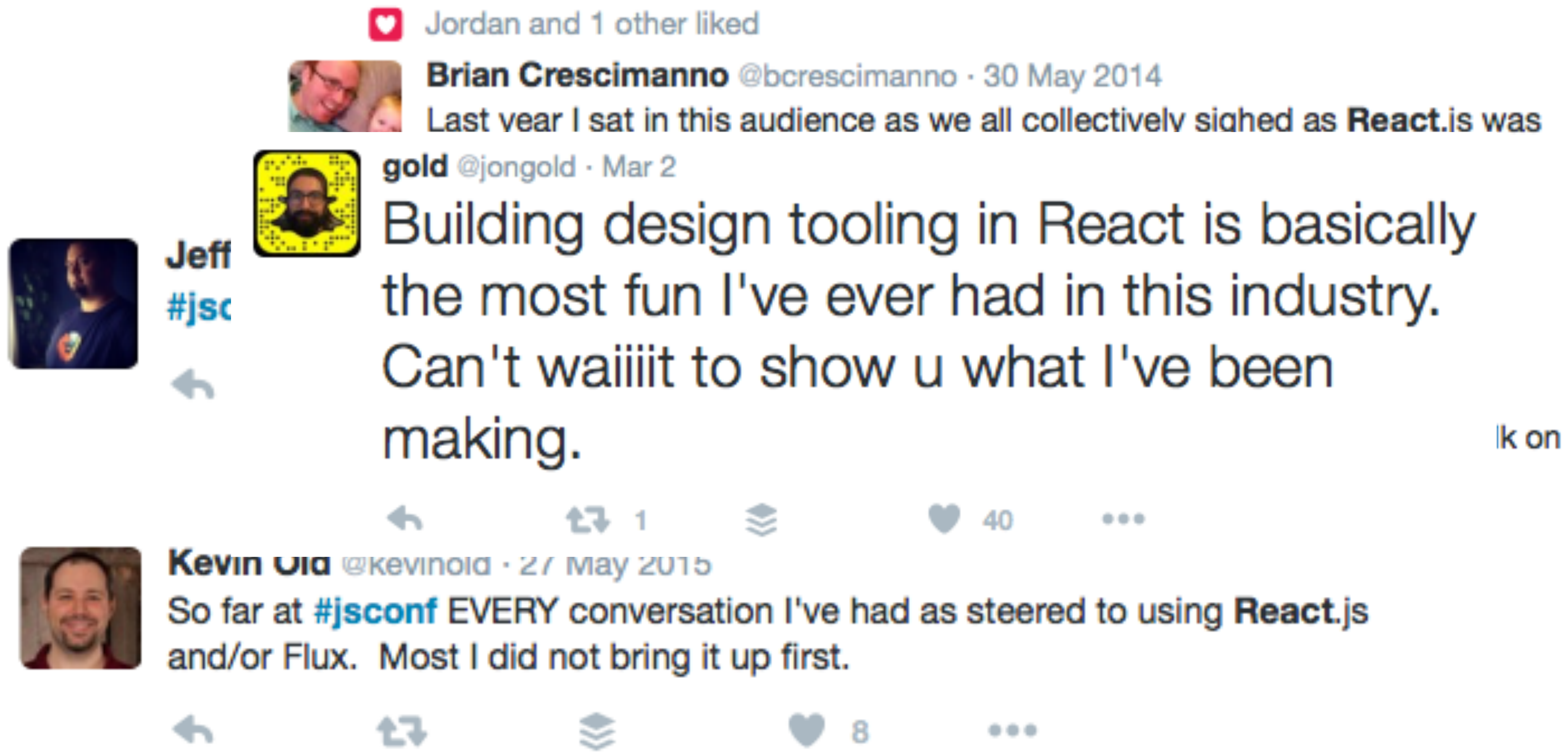
Middleton @middle2000lb · 29 May 20

If there was a dance where you took c

would be called **React.js** #jsconf

1 1

Fast-forward



Jordan and 1 other liked

Brian Crescimanno @bcrescimanno · 30 May 2014
Last year I sat in this audience as we all collectively sighed as **React**.js was

gold @jongold · Mar 2
Building design tooling in React is basically the most fun I've ever had in this industry. Can't waiiiit to show u what I've been making.

1 40

Jeff
#jsc

lk on

Kevin Old @kevinold · 27 May 2015
So far at #jsconf EVERY conversation I've had as steered to using **React**.js and/or Flux. Most I did not bring it up first.

8

JSX Example

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

class MyComponent extends Component {
  render() {
    return (
      <div>
        <span>What is this madness?</span>
      </div>
    )
  }
}
```

Components… All the Way Down

- Often termed as the ‘V’ in ‘MVC’
- All about building reusable, composable, testable components
- Declarative, one-way data flow
  - Render the state that is passed in
  - Re-render only what has changed
- Client and / or server-side rendering!

Component Props

```
render() {
  return (
    <ul>
      <li>Name: {this.props.name}</li>
      <li>Role: {this.props.role}</li>
    </ul>
  )
}

ReactDOM.render(
  <User name="Sean" role="Speaker" />,
  document.body)
```

Component Lifecycle

- Mounting
  - componentWillMount()
  - componentDidMount()
- Updating
  - componentWillReceiveProps()
  - shouldComponentUpdate()
  - componentWillUpdate()
  - componentDidUpdate()
- Unmounting
  - componentWillUnmount()

Wrapping Another Library

```
import $ from 'jquery'
require('jquery-tooltipster/js/jquery.tooltipster.js')

class HoverMe extends Component {
  componentDidMount() {
    $(ReactDOM.findDOMNode(this.refs.tooltip)).tooltipster({
      content: $('<h1>BOOM</h1>')
    });
  }

  render() {
    return (<div ref="tooltip">Hover Me!</div>)
  }
}

ReactDOM.render(<HoverMe />, document.body)
```

Re-using Our Component

```
class Wrapper extends Component {
  render() {
    return (
      <div><h1>Children</h1>{this.props.children}</div>
    )
  }
}

class App extends Component {
  render() {
    return (
      <Wrapper>
        <User name="Sean" role="Speaker" /><ThirdParty />
      </Wrapper>
    )
  }
}
```

Styling Our Component

- Inline styling
- CSS modules

Problems with CSS at Scale

- Christopher Chedeau’s “CSS in JS” talk

- Global namespace
- Dependencies
- Dead code elimination
- Minification
- Sharing constants
- Non-deterministic resolution
- Isolation

CSS Modules

- Local by default
- Explicit dependencies
- Composable
  - Even from other files
- Particularly nice w/ React

Importing CSS into JS? How does that even work?

It’s time to talk about webpack.

Your front-end has many dependencies

- HTML
- JavaScript
- CSS / SASS / Less
- Fonts
- Images
- JSON
- Etc.

webpack understands and bundles dependencies

- Understands dependency relationships
- Provides mechanisms to split the dependency tree into on-demand chunks
- Highly extensible via loaders and plugins

webpack has a learning curve

- Documentation is confusing
- Not clear at first why you want it
- Feels like progress initially occurs by beating your head against it until something moves

… but it turns out it’s not complicated (just flexible)

- Ignore starter kits
- Start small
- Add one thing at a time
- Think about entry points

Loaders

- webpack processes JavaScript natively.
- Loaders transform resources into JavaScript representations.
- Loaders can be chained (final one *must* return JavaScript).
- Loaders are generally bound to specific file extensions / RegExs.
- Processed right-to-left

Some Loaders

- babel
- style
- css
- sass
- less
- postcss
- mustache
- eslint
- stylelint
- ngtemplate
- Many, many more

Plugins

- Augment loaders and bundling with additional functionality
- Generally less file specific and more bundle specific

Some Plugins

- Banner
- UglifyJS
- Commons Chunk
- HtmlWebpack
- S3

Thanks for coming!

Twitter: **@seantimm**

Blog: **escreturn.com**