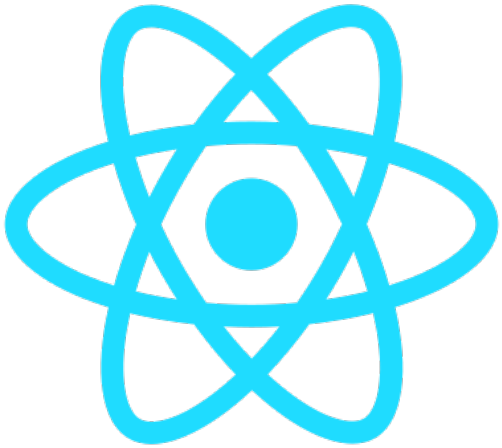
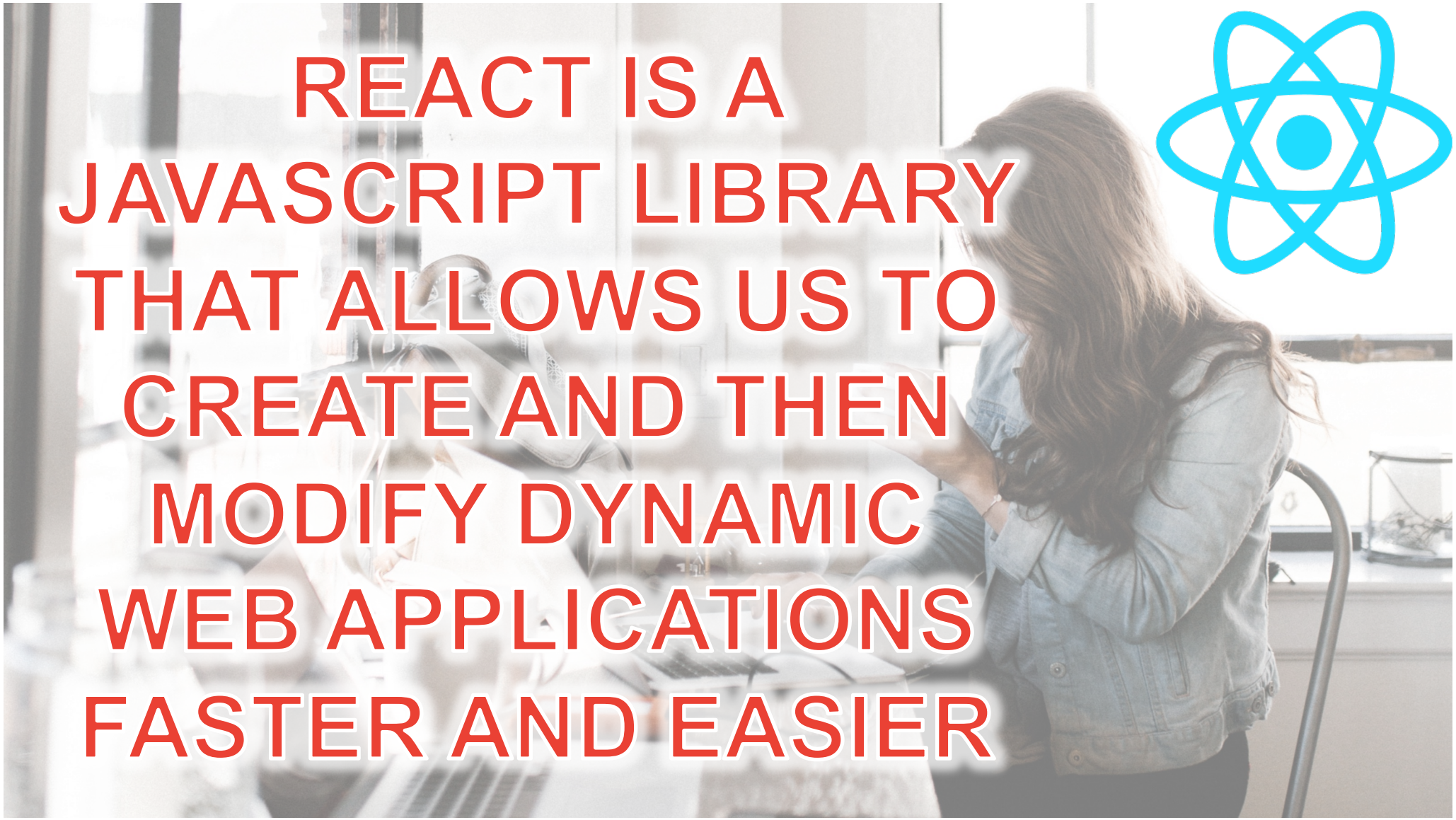


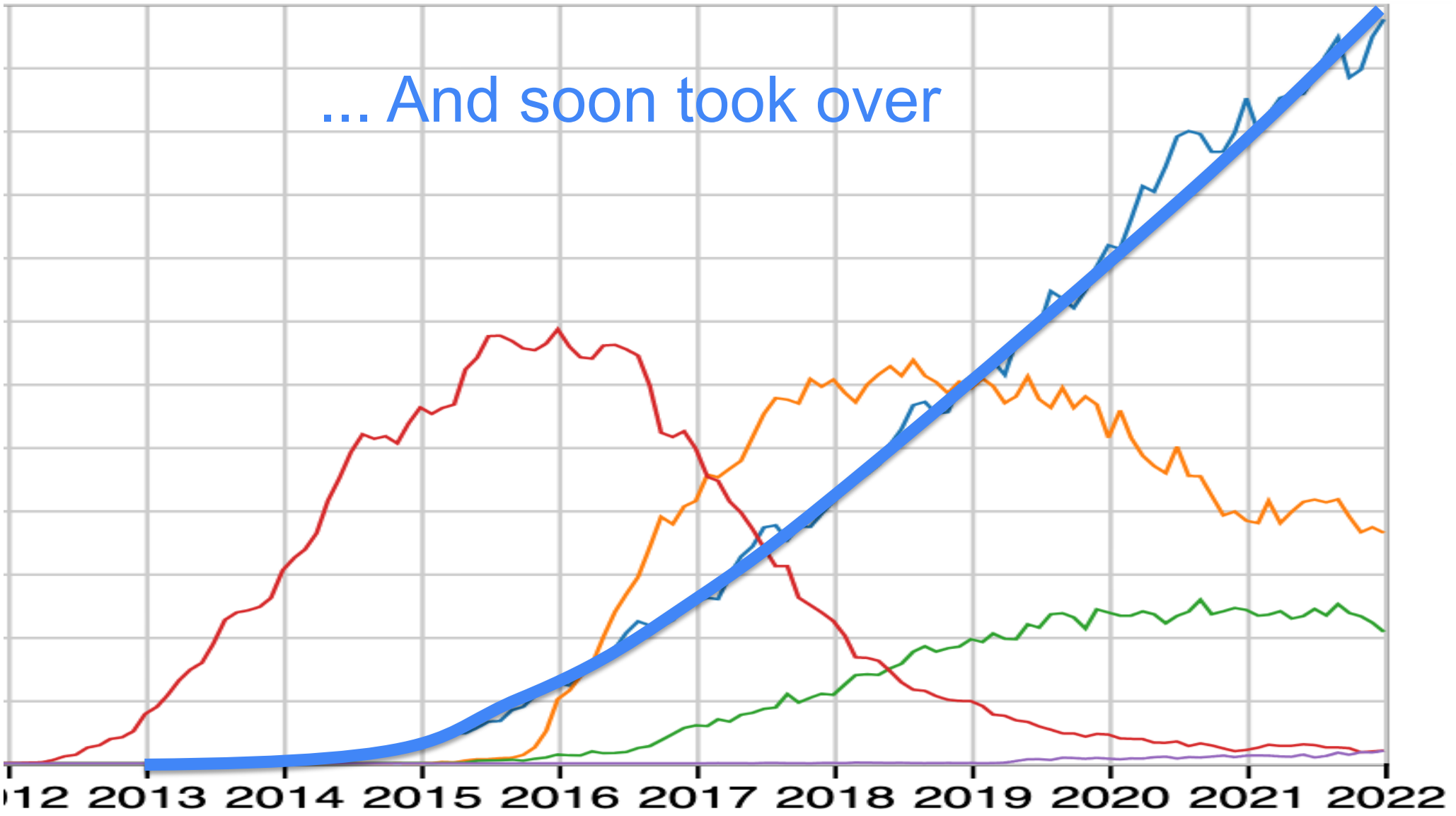
Hello, React

A gentle introduction to React. What is this thing anyway, and why should I learn it?

What is React?

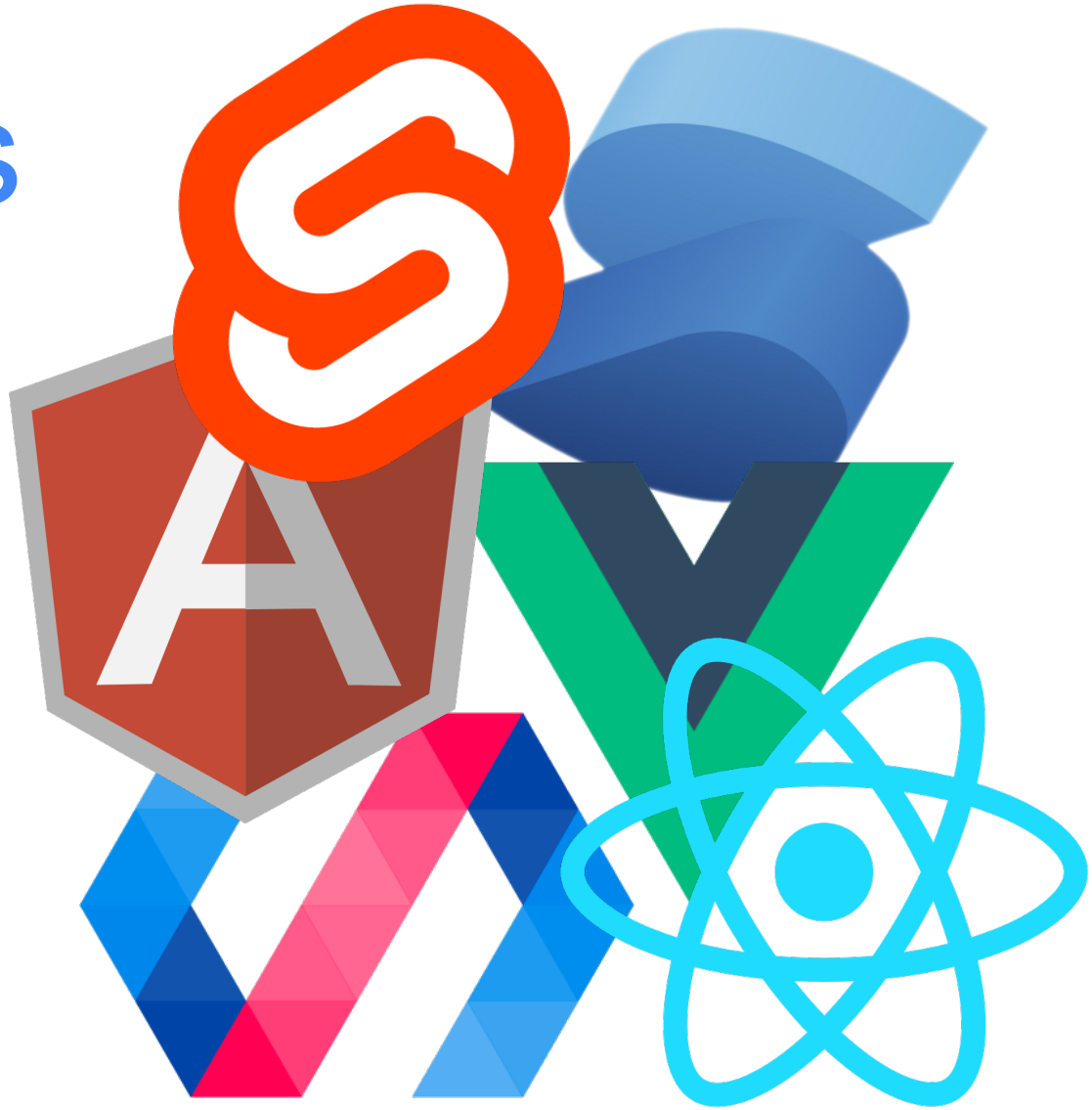

REACT IS A
JAVASCRIPT LIBRARY
THAT ALLOWS US TO
CREATE AND THEN
MODIFY DYNAMIC
WEB APPLICATIONS
FASTER AND EASIER

Written in 2011 by
Jordan Walke of
Facebook's Ads team


... And soon took over
012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022

The web has gone to components

Nearly 100% of modern web app development is done using components

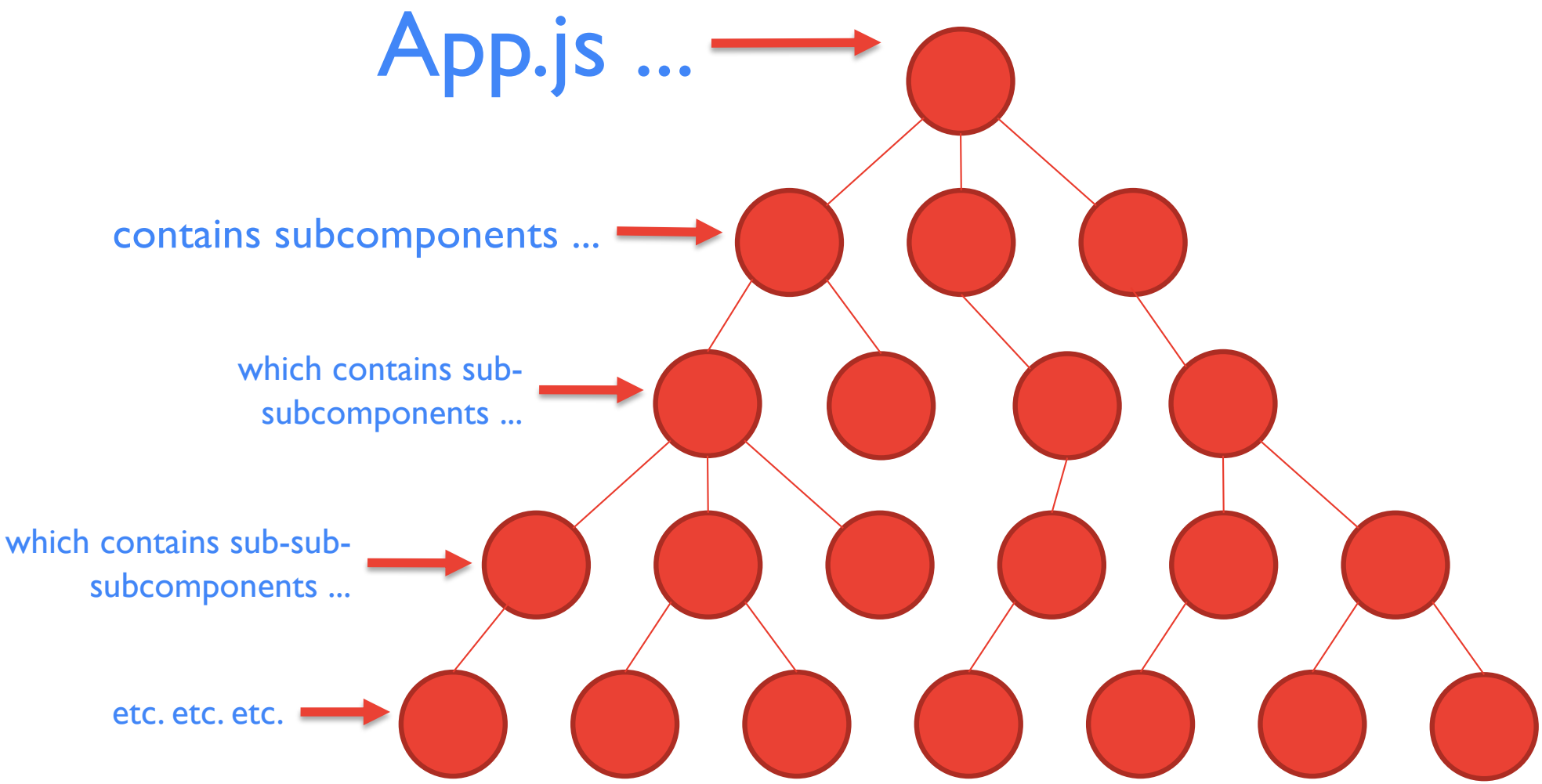


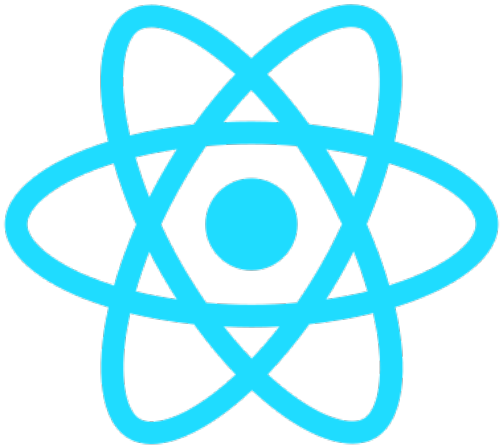
With React, you'll no longer write pages; you'll write components

Each component is self-contained and encapsulated

- Even styles are local; CSS no longer cascades through components


YOU'LL BREAK DOWN THE PRODUCT INTO SECTIONS. THEN YOU BREAK THOSE INTO SMALLER SECTIONS AND SO ON ...
PHOTO: LEGO


App.js ...
contains subcomponents ...
which contains sub-subcomponents ...
which contains sub-sub-subcomponents ...
etc. etc. etc.



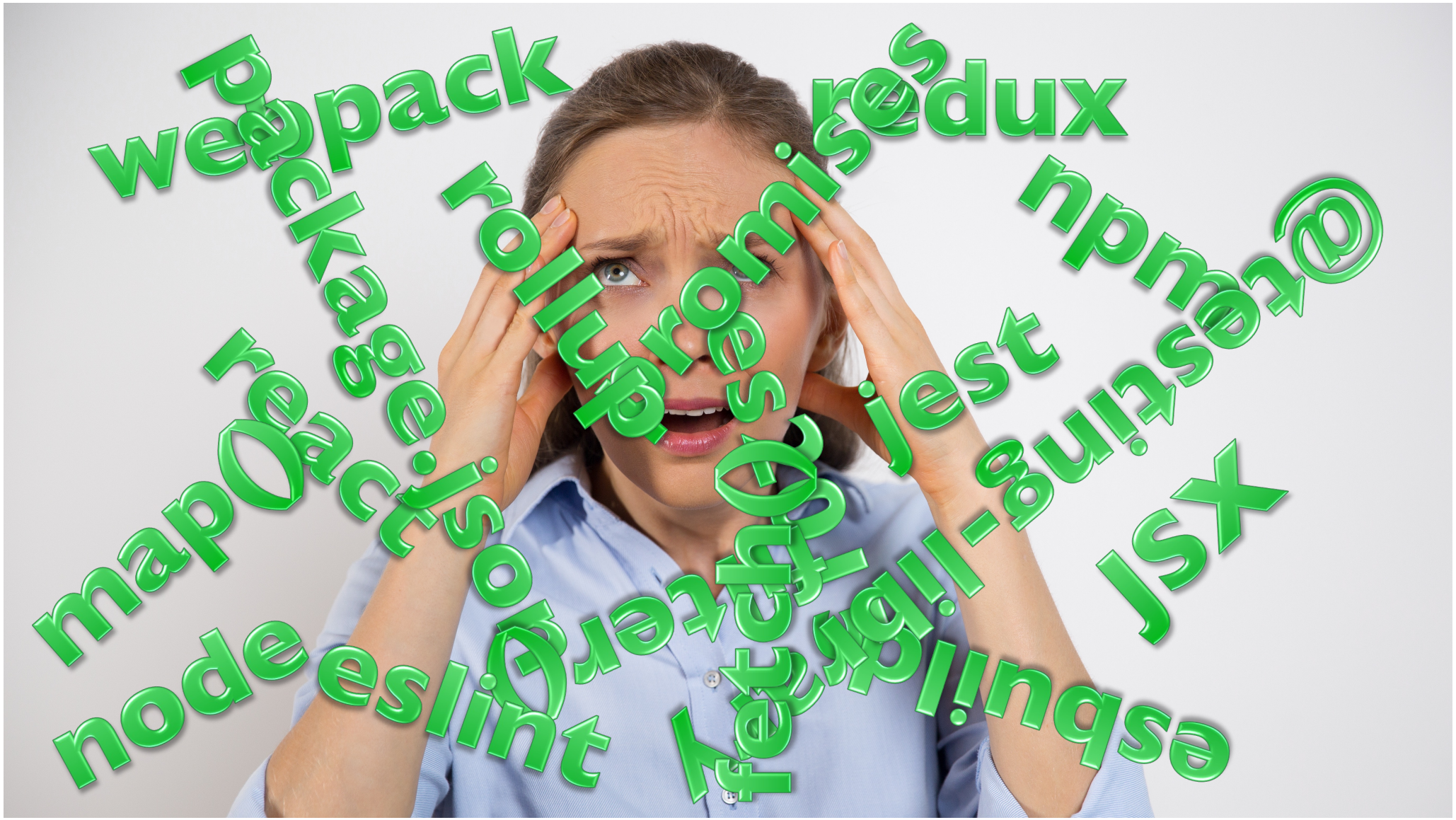
How to create a React app

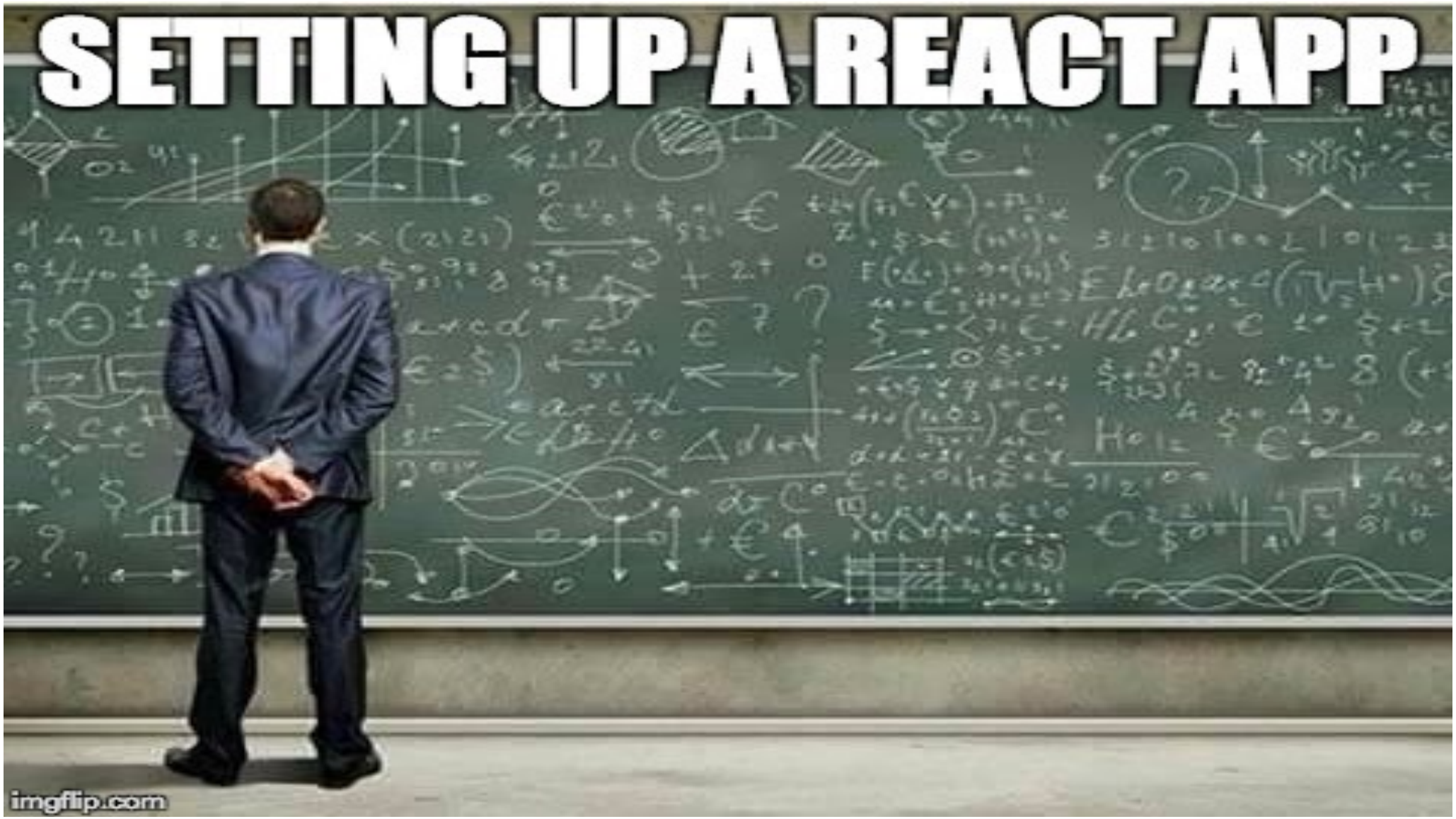
tl;dr

- React is very difficult to configure and brittle.
- So use create-react-app or vite to scaffold the project
- They create a package.json file that allows you to
  - Serve in development watch mode (npm start or npm run dev)
  - Compile for production deployment (npm run build)

React has a lot of moving parts

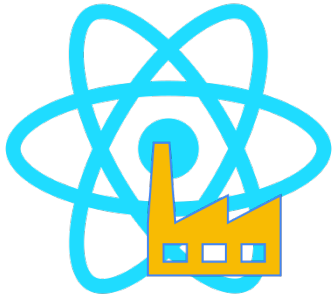
React uses JSX, which isn't JavaScript ...
So it has to be transpiled by Babel or SWC ...
which should be automated by webpack or rollup ...
which also uglifies, minifies, bundles and tree-shakes ...
the CSS, HTML, and hundreds of libraries ...
which must be managed by npm ...
thus carefully configured in package.json ...
linted with eslint and tested with jest and RTL ...
using ES2015 modules and AMD requires ...
run through node processes ...


webpack
redux
rollup
jest
XSL
node
eslint
babel


SETTING UP A REACT APP
imgflip.com

We knew it was frustrating to spend days setting up a project when all you wanted was to learn React. We wanted to fix this.

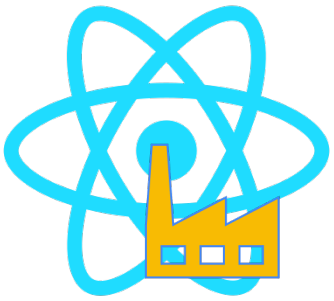
We loved Ember CLI and Elm Reactor which help users to get started.



So Christopher Chedeau, Kevin Lacker, and I wrote create-react-app

create-react-app makes your project

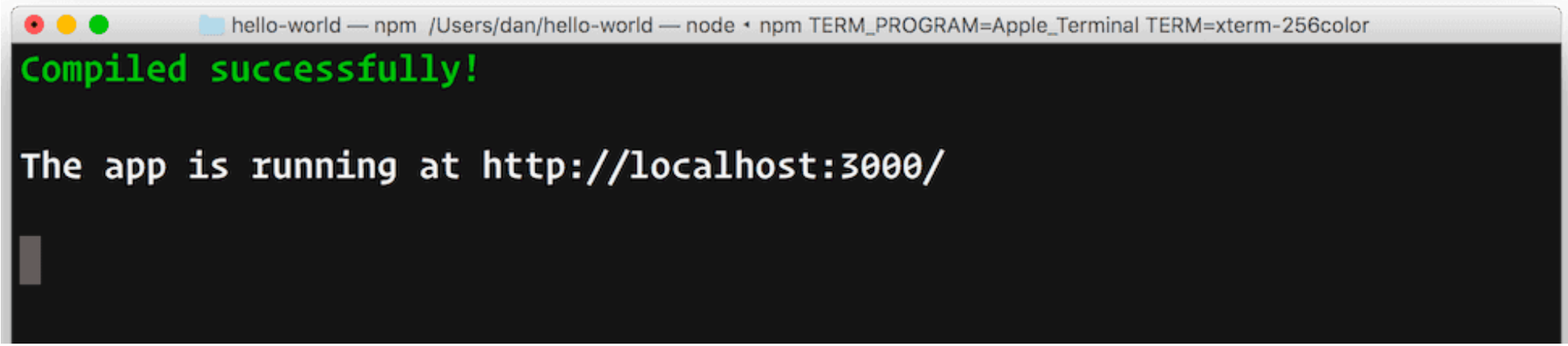
```
npx create-react-app hello-world
```

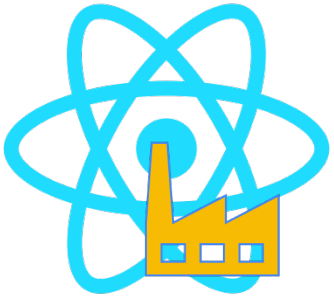


```
create-react-app — fish  /Users/dan/p/create-react-app — -fish
└── react-dom@15.2.1

Success! Created hello-world at /Users/dan/hello-world.

Inside that directory, you can run several commands:
  * npm start: Starts the development server.
  * npm run build: Bundles the app into static files for production.
  * npm run eject: Removes this tool. If you do this, you can't go back!

We suggest that you begin by typing:
  cd /Users/dan/hello-world
  npm start

Happy hacking!
create-react-app)
```

Starting the Server

- Run **`npm start`** to launch the development server. The browser will open automatically with the created app's URL.



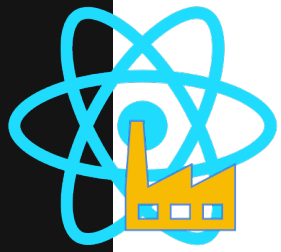


Then you're in watch mode

- When your code changes, webpack lints with eslint and transpiles with babel.

hello-world — npm /Users/dan/hello-world — node • npm TERM_PROGRAM=Apple_Terminal TERM=xterm-256color

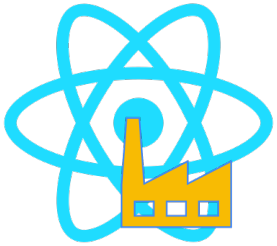
```
Failed to compile.

Error in ./src/App.js
Syntax error: /Users/dan/hello-world/src/App.js: Unexpected token (8:7)
   6 |   render() {
   7 |     return (
>  8 |       <<div className="App">
     |        ^
   9 |         <div className="App-header">
  10 |           <img src={logo} className="App-logo" alt="logo" />
  11 |           <h2>Welcome to React</h2>
 @ ./src/index.js 11:11-27
```



ESLint output shows in the console

- The lint rules are tuned for a React app.

```
hello-world — npm /Users/dan/hello-world — node • npm TERM_PROGRAM=Apple_Terminal TERM=xterm-256color
Compiled with warnings.

Warning in ./src/App.js

/Users/dan/hello-world/src/App.js
  7:9  warning  'hello' is defined but never used  no-unused-vars

✖ 1 problem (0 errors, 1 warning)


You may use special comments to disable some warnings.
Use // eslint-disable-next-line to ignore the next line.
Use /* eslint-disable */ to ignore all warnings in a file.
```

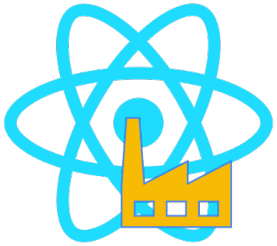


"`npm run build`" to go to production

It is minified, bundled, content hashed (for caching).

fish /Users/dan/hello-world — -fish

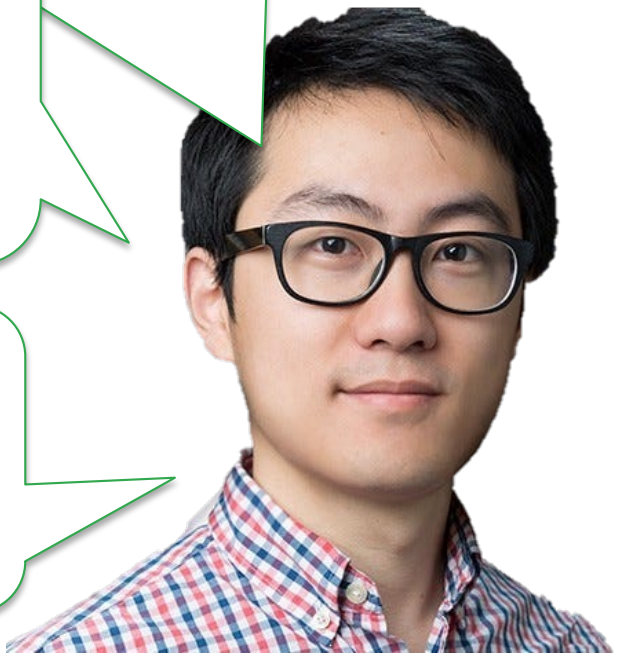
```
hello-world) npm run build

> hello-world@0.0.1 build /Users/dan/hello-world
> react-scripts build

Successfully generated a bundle in the build folder!

You can now serve it with any static server, for example:
  cd build
  npm install -g http-server
  hs
  open http://localhost:8080

The bundle is optimized and ready to be deployed to production.
hello-world)
```



vite

I needed a server for my baby Vue.js. Webpack can work, but it is really big and slow.

So I created vite. ('Vite' is French for 'quick').



Then we expanded it to work with React, Lit, Svelte, Solid, and more.

```
$ npm create vite
Need to install the following packages:
  create-vite@X.Y.Z
Ok to proceed? (y)
✔ Project name: … delete-me
✔ Select a framework: › React
? Select a variant: › - Use arrow-keys. Return to submit.
    JavaScript
    TypeScript
    JavaScript + SWC
›   TypeScript + SWC
```

The SWC is "Speedy Web Compiler", an alternative to Babel. You should definitely use it

```
Done. Now run:


  cd delete-me
  npm install
  npm run dev
```

Three options for compiling:

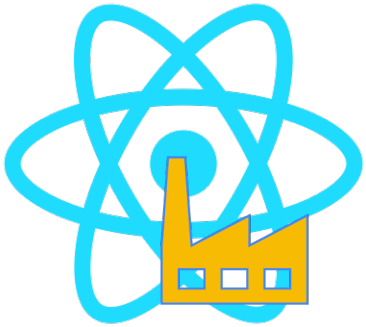
```
npm run dev      # Run app in development mode
npm run build    # Compile app for deployment
npm run preview  # Both of the above - compile and
                 #   run from your compiled files
```

To run your app ... go `npm run dev`

```
$ npm run dev

> my-project@0.0.0 dev
> vite
```

```
  VITE vX.Y.Z  ready in 316 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

So which should I use?



Slower

Bigger bundles

Uses webpack

CommonJS modules

Faster

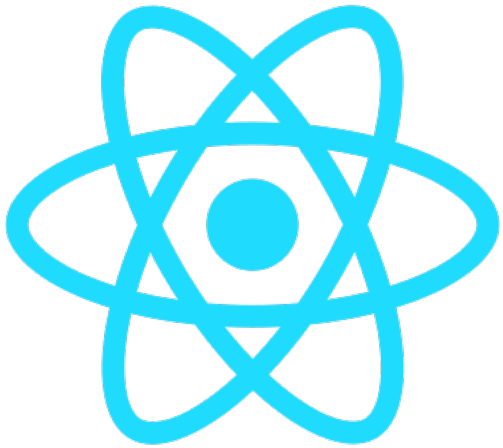
Smaller bundles

Uses SWC and Rollup

ES modules

tl;dr

- React is very difficult to configure and brittle.
- So use create-react-app or vite to scaffold the project
- They create a package.json file that allows you to
  - Serve in development watch mode (npm start or npm run dev)
  - Compile for production deployment (npm run build)



How to create components

tl;dr

- The three steps to make a React component
- The two ways to get that component on a page

How to make a React component

1. Create a component file
2. Create a function
3. Return JSX from it

1. Create a component file

- ... in the src folder

Foo.js

```
export function Foo() {
 return <SomeComponent />;
}
```

2. Create a function ...

- Must start with an uppercase letter!
- Prefer Pascal-cased

Foo.js

```
export function Foo() {
 return <SomeComponent />;
}
```

3. Return JSX from it

Foo.js

```
export function Foo() {
 return <SomeComponent />;
}
```

3. Return JSX from it

When you want a return and then have JSX starting on the next line, wrap them in parentheses or else Babel gets confused.

Foo.js

```
export function Foo() {
 return (
  <SomeComponent />
 );
}
```

Actually, it returns JSX or ...

- You must return
  1. JSX with a single root element
  2. An array of JSX elements
  3. A string
  4. null
- Anything else is an error

Wondering about JSX? Don't worry, we'll cover it in depth next chapter.

Hosting a component

You've got a component but to be seen it must be *hosted*.

You can host it inside another component or it can be the root component.

To host inside another component, put the tag for this one in the parent's JSX

```
function Parent() {
  return <div>
    <Person />
    <Person></Person>
  </div>
}
```

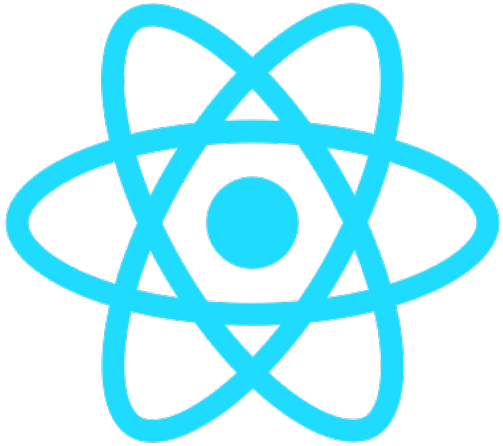
This would show two Person components.

To make this the top-level component, use root.render()

index.js

```
import {createRoot} from 'react-dom/client'
const container =
 document.querySelector('#root')
const root = createRoot(container);
root.render(<Person></Person>)
```

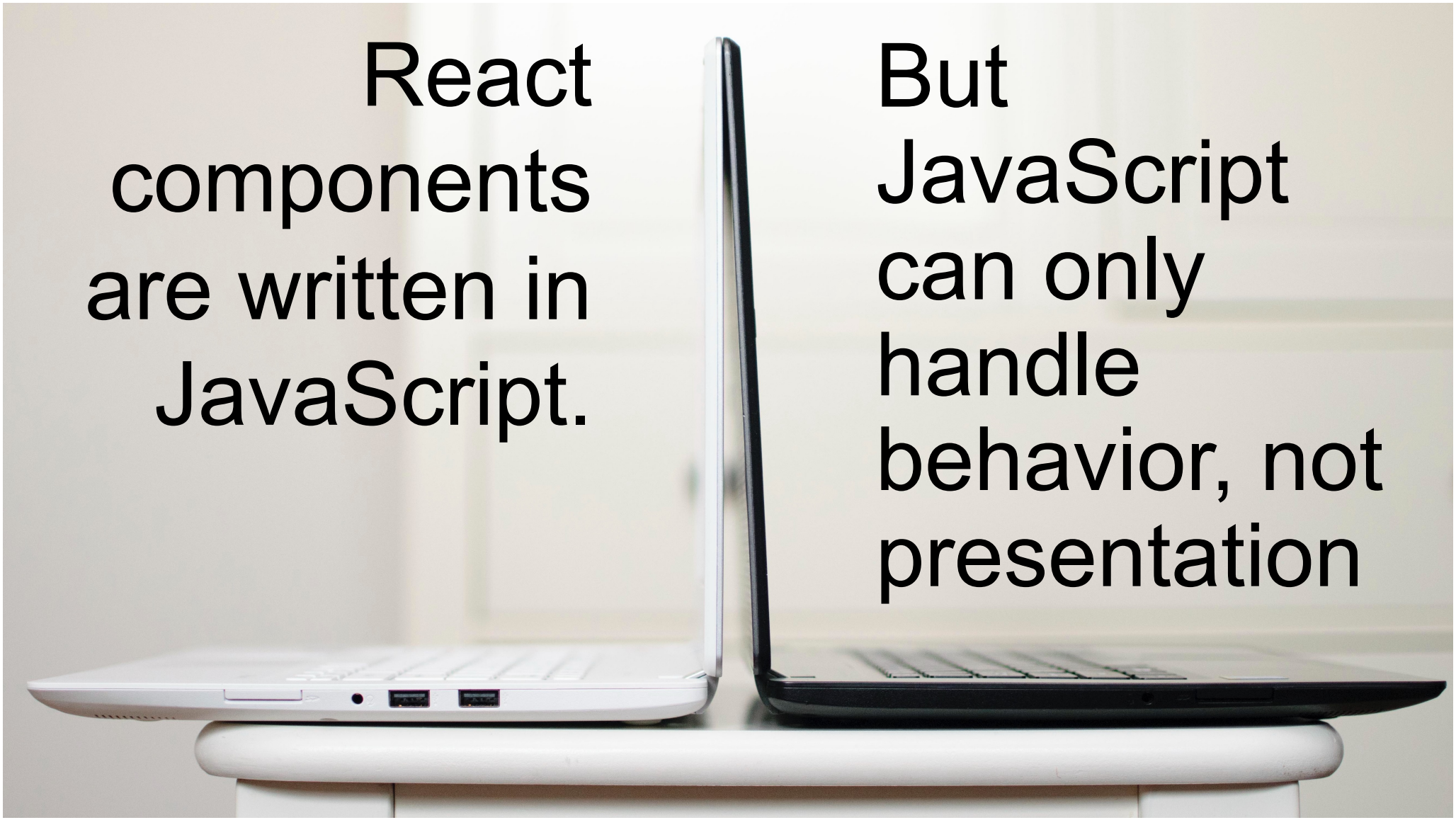
This means that index.html must have an element with id="root".



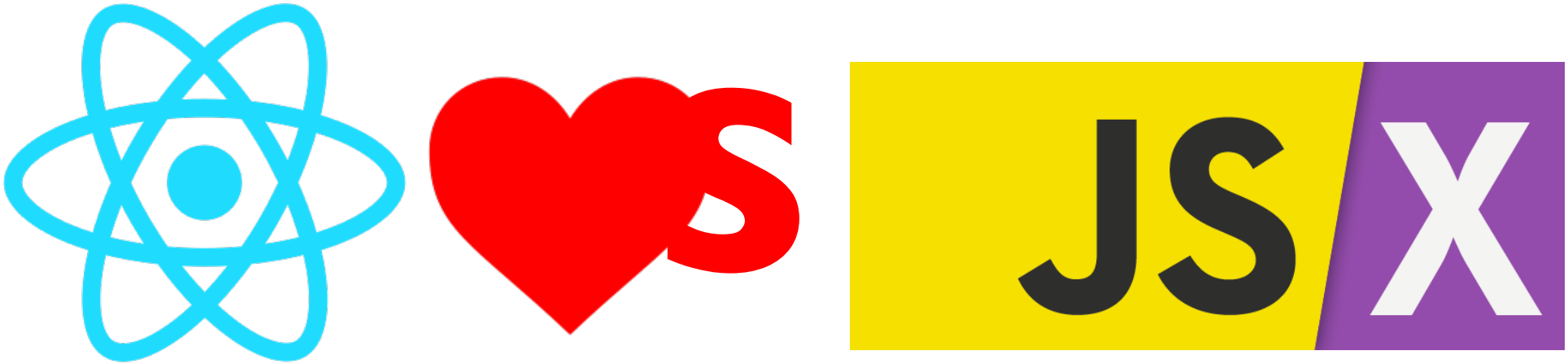
How to display HTML

tl;dr

- What JSX *actually* means (It isn't what most devs think!)
- The 7 rules of JSX
- How JSX is interpreted and run in the browser


React components are written in JavaScript.
But JavaScript can only handle behavior, not presentation

To help with the presentation, React introduces a new DSL* called JSX



* Domain-Specific Language


JavaScript + XML = JSX
It isn't JSH. It is JSX.
HTML is forgiving ... XML is not!

XML must be ...

- Well-formed
  - Conforms to rules
- Valid
  - Conforms to its DTD

Well-formed XML (and therefore, JSX) Rules

- All XML elements must have a closing tag.
- XML tags are case-sensitive.
- All XML elements must be properly nested.
- All XML documents must have a root element.
- Attribute values must always be quoted.

Alright, how about that *valid* thing?

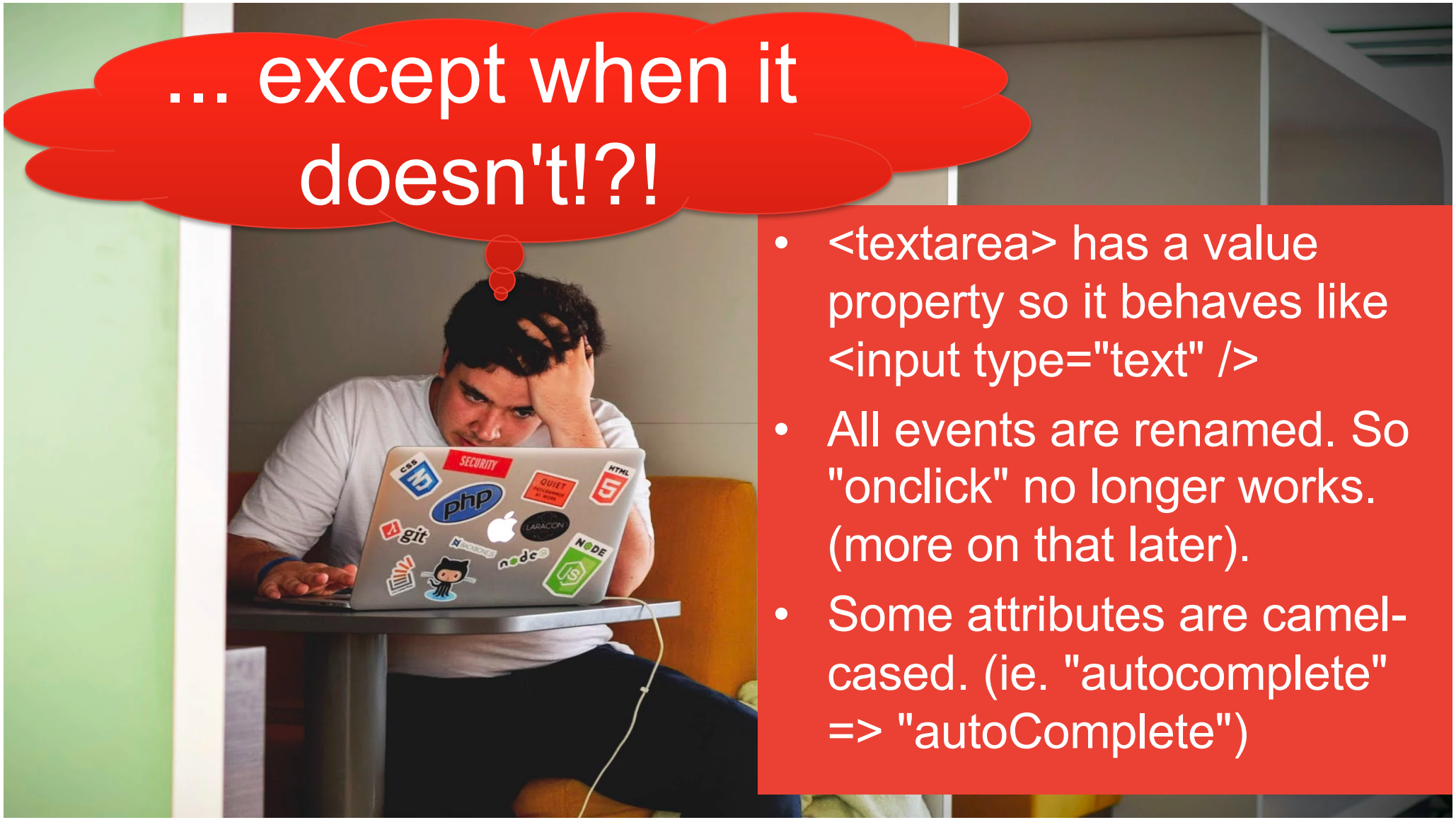
Well, there is no DTD, but it does follow rules ...

- All tags must be pre-defined. They're either ...
  1. Valid W3C-approved elements with proper attributes
  2. Valid pre-defined React components that you wrote
- It uses casing to determine which is which!

W3C tags are lower-cased.
React components begin with an upper-case character

Got compile errors in your JSX? Ask yourself "Is this strictly good W3C HTML?" And fix it where it is not standard.

JSX must conform to W3C standards ...



- <textarea> has a value property so it behaves like <input type="text" />
- All events are renamed. So "onclick" no longer works. (more on that later).
- Some attributes are camel-cased. (ie. "autocomplete" => "autoComplete")

Quiz: What's wrong with this?

```
let class = "Programming 101";
let for = "because I want to graduate";
```

- Therefore, with JSX you can't go ...

```
<label class="big" for="firstname">First</label>
<input id="firstname" />
```

- Instead you have to ...

```
<label className="big" htmlFor="firstname">First</label>
<input id="firstname" />
```

Warning in the console.
Not a compile error

Pop quiz!

On the next few pages,
let's look at some
problems and solutions

Quiz: What's wrong with this JSX?

That's not a valid element

BadJSX.js

```
<goodShows>
 <li>Game of Thrones</li>
 <li>Big Bang Theory</li>
 <li>Mr. Robot</li>
</goodShows>
```

And how can we fix it?

BetterJSX.js

Make the custom element properly cased

```
<GoodShows>
  <li>Game of Thrones</li>
  <li>Big Bang Theory</li>
  <li>Mr. Robot</li>
</GoodShows>
```

Quiz: What's wrong with this JSX?

Can't have >1 root element

BadJSX.js

```
<h1>Good shows</h1>
<ul>
 <li>Game of Thrones</li>
 <li>The Expanse</li>
 <li>Mr. Robot</li>
</ul>
```

And how can we fix it?

BetterJSX.js

```
<>
<h1>Good shows</h1>
<ul>
 <li>Game of Thrones</li>
 <li>The Expanse</li>
 <li>Mr. Robot</li>
</ul>
</>
```

Wrap it in a or a React *fragment*

Quiz: What's wrong with this JSX?

You can't overlap elements

BadJSX.js

```
<ul>
 <li><a href="got.com">Game of
Thrones</li></a>
 <li><a href="bbt.com">Big Bang
Theory</a></li>
 <li><a href="mrr.com">Mr.
Robot</a></li>
</ul>
```

And how can we fix it?

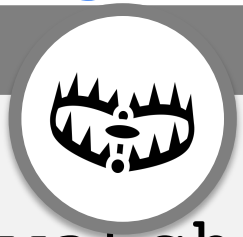
BetterJSX.js

```
<ul>
 <li>
  <a href="got.com">
   Game of Thrones
  </a>
 </li>
 <li><a href="bbt.com">Big Bang
Theory</a></li>
 <li><a href="mrr.com">Mr.
Robot</a></li>
</ul>
```

Nest them instead of overlapping them

Quiz: What's wrong with this JSX?

BadJSX.js



Open tags must be closed

```
<>
<p>All geeks should watch these
shows:
<ul>
 <li>Game of Thrones</li>
 <li>Big Bang Theory</li>
</ul>
</>
```

And how can we fix it?

BetterJSX.js

```
<>
<p>All geeks should watch these
shows:</p>
<ul>
 <li>Game of Thrones</li>
 <li>Big Bang Theory</li>
</ul>
</>
```

Close the tag or make it self-closing

But wait! Which browsers support JSX?

- All of them! (Or none of them?)
- Because JSX never lands in the browser.
- It is transpiled out and replaced with equivalent JavaScript

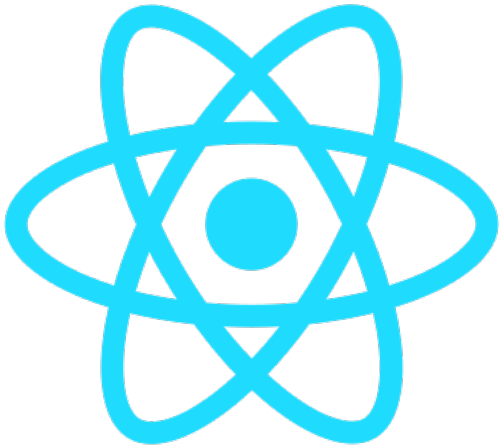
This transpiles to this

With_JSX.js

```
function Hello() {
 return (
  <div>
   <label htmlFor="n">Name</label>
   <input id="n" required
          value="foo.jpg" />
  </div>
 );
}
```

Without_JSX.js

```
function Hello() {
 return
React.createElement(
  "div",
  null,
  React.createElement(
   "label",
   { "htmlFor": "n" },
   "Name"
  ),
  React.createElement(
   "input",
   {id:"n", required:true,
     value: "foo.jpg" })
  );
}
```



How to handle events

tl;dr

- JSX strips out the native HTML events and replaces them with their own.
- They're called synthetic events and they don't behave exactly like their native mirrors

utilities.js

```
export const foo = (x,y) => x + y;
```

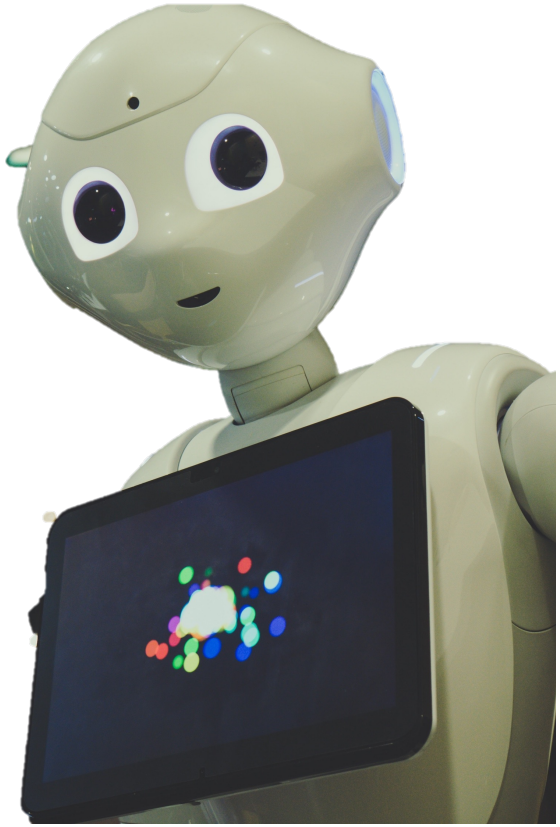
before we go on, pop quiz! ... What is 'foo'?

It's a function! You could call it with

```
const z = foo(35,7);
```

React has created its own synthetic version of most events

- React overrides the native events with their own synthetic events



man-made

unnatural

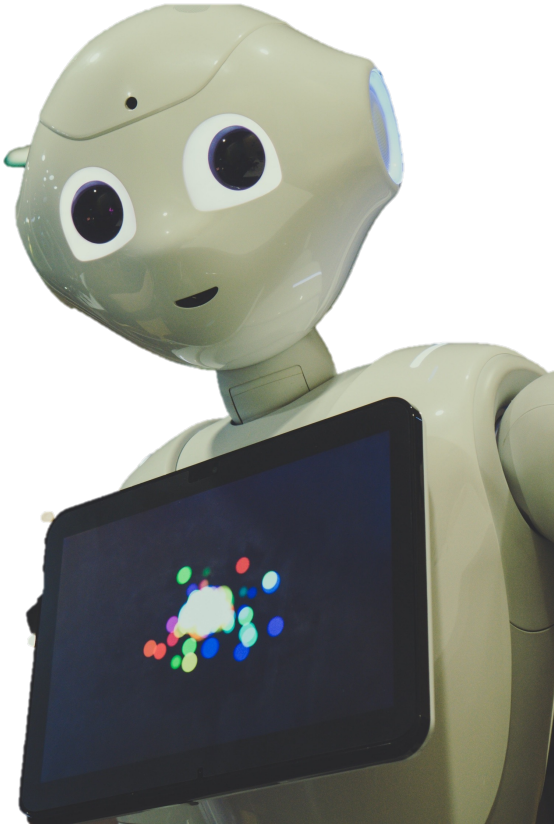
simulated

mock

artificial

manufactured

Synthetic



Events that ARE supported

https://reactjs.org/docs/events.html

onCopy onCut onPaste onKeyDown onKeyPress onKeyUp onFocus onBlur onChange onInput onInvalid onReset onSubmit onError onLoad onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp onPointerDown onPointerMove onPointerUp onPointerCancel onGotPointerCapture onLostPointerCapture onPointerEnter onPointerLeave onPointerOver onPointerOut onSelect onTouchCancel onTouchEnd onTouchMove onTouchStart onScroll onWheel onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied onEncrypted onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause onPlay onPlaying onProgress onRateChange onSeeked onSeeking onStalled onSuspend onTimeUpdate onVolumeChange onWaiting onAnimationStart onAnimationEnd onAnimationIteration onTransitionEnd onToggle

Camel case the event and precede it by *on*

```
<any onFoo={e=>bar()}></any>
```

Examples:

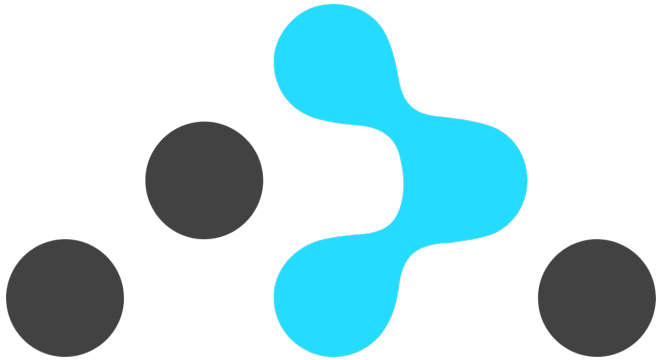
```
<button onClick={e=>doIt()}>
  Press me</button>
<img onMouseOver={e=>go()} />
<input onBlur={e=>blurred()}
       onKeyUp={e=>run(e)} />
```

And to pass the event object ...

MyComponent.js

```
...
return (
 <button onDoubleClick={e => doStuff(e, obj1, obj2)}>
  Click me!
 </button>
);
```

- This works because when an event is triggered, the (synthetic) event object is passed into the registered function



How to do simple routing

tl;dr

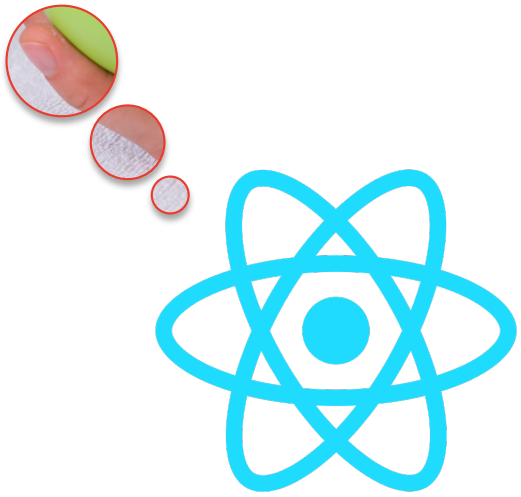
- Routing tricks the user into thinking they're navigating to pages when we're only swapping out components
- To route, you ...

1. Define the routing domain with a <Router>
2. Match URLs to components with <Route>s

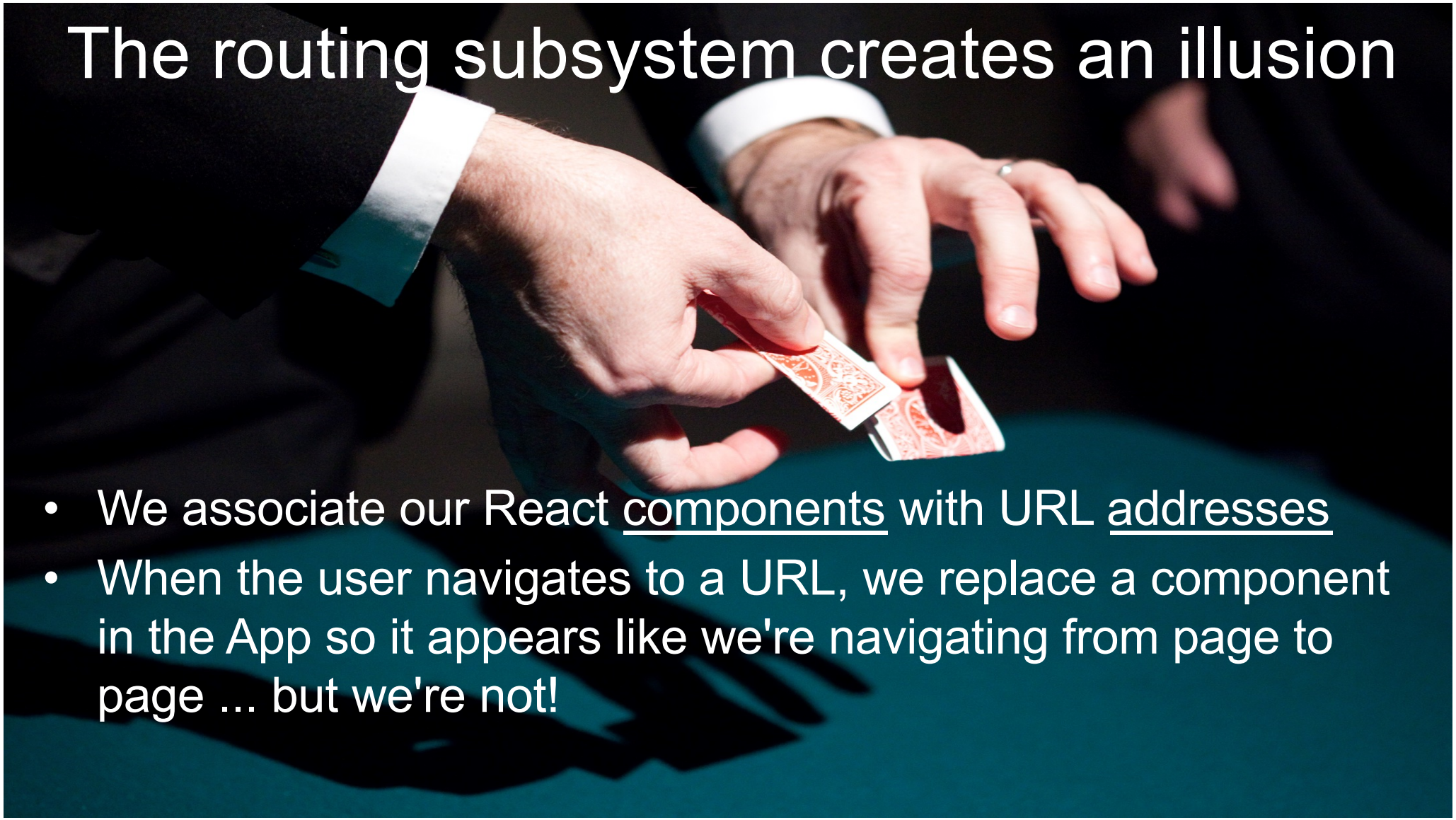
- You visit those routes, you hit a url

React thinks in SPAs first

- But if there's only one page, how do we navigate from page to page?
- We don't!



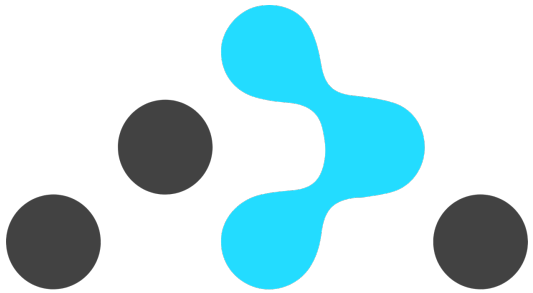
The routing subsystem creates an illusion



- We associate our React components with URL addresses
- When the user navigates to a URL, we replace a component in the App so it appears like we're navigating from page to page ... but we're not!

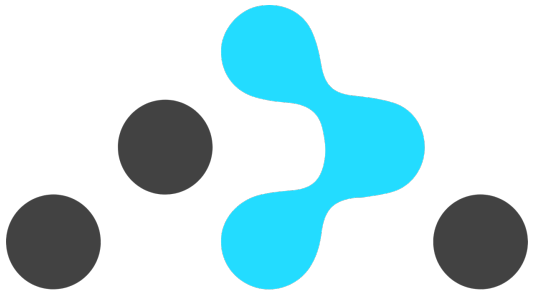
Routing is that process of keeping the browser URL in sync with what's being rendered on the page

React Router lets you handle routing declaratively.



React Router is NOT from Facebook!

- It's from the community
- But it is the de facto router



We route through JSX

```
<Route path="/about" element={<About/>} />
```

Hey, wait! "<Route>" isn't a thing! What is it?

It's a React component.

So where did it come from?

```
npm install react-router-dom    //For web
// And then ...
import { Route } from 'react-router-dom';
```

Only <Route>s that are nested somewhere -- at any level -- inside a <Router> will be honored

So <Router>s are usually put around or directly inside the <App />

For example ...

A Router around the App

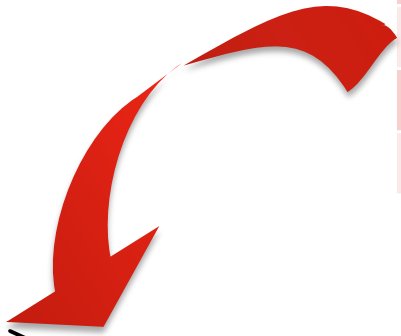
index.js

```
const node =
 document.getElementById('mainDiv');
const root = ReactDOM.createRoot(node);
root.render(
 <BrowserRouter>
  <App />
 </BrowserRouter>
);
```

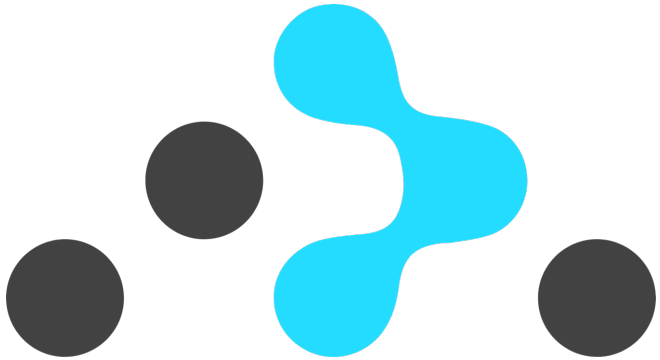
How to create the routes

Design the routes ...

URL	Component
/people	People
/people/123	Person
/teams	Teams
/cart	ManageCart
/	Welcome
Anything else	FourOhFour



```
<Routes>
 <Route path="/" element={<Welcome />} />
 <Route path="/people" element={<People />} />
 <Route path="/people/:id" element={<Person />} />
 <Route path="/teams" element={<Teams />} />
 <Route path="/cart" element={<ManageCart />} />
 <Route path="*" element={<FourOhFour />} />
</Routes>
```



How to read route parameters

Defining routes

URL	Component
/people	People
/people/123	Person
/teams	Teams
/cart	ManageCart
/	Welcome
Anything else	FourOhFour

```
<Routes>
 <Route path="/" element={<Welcome />} />
 <Route path="/people" element={<People />} />
 <Route path="/people/:id" element={<Person />} />
 <Route path="/teams" element={<Teams />} />
 <Route path="/cart" element={<ManageCart />} />
 <Route path="*" element={<FourOhFour />} />
</Routes>
```


But how do we get data from the url into the component?

use the useParams hook

useParams looks for any matches to parameters and returns an object with all of the matches.

Don't forget to import useParams before using it

PickSeats.js

```
import { useParams } from 'react-router-dom';
export const PickSeats = () => {
 const { showingId } = useParams();
 return <SomeJSX />;
}
```

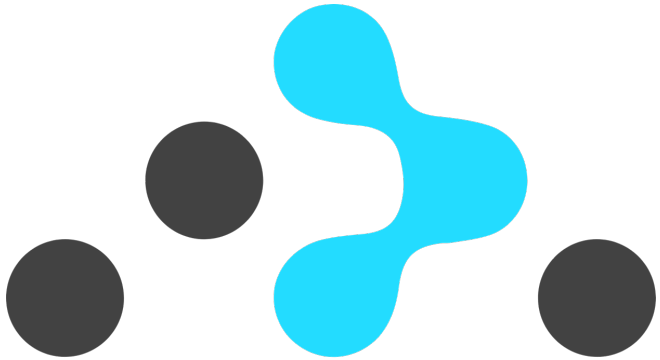
App.js

```
<Routes>
 <Route path="/teams/:id" element={</Teams />} />
</Routes>
```

http://us.com/teams/1234

Teams.js

```
const Teams = (props) => {
 const {id} = useParams();
 console.log(id);  // 1234
 return <h1>Team {id}</h1>
}
```



How to create efficient links

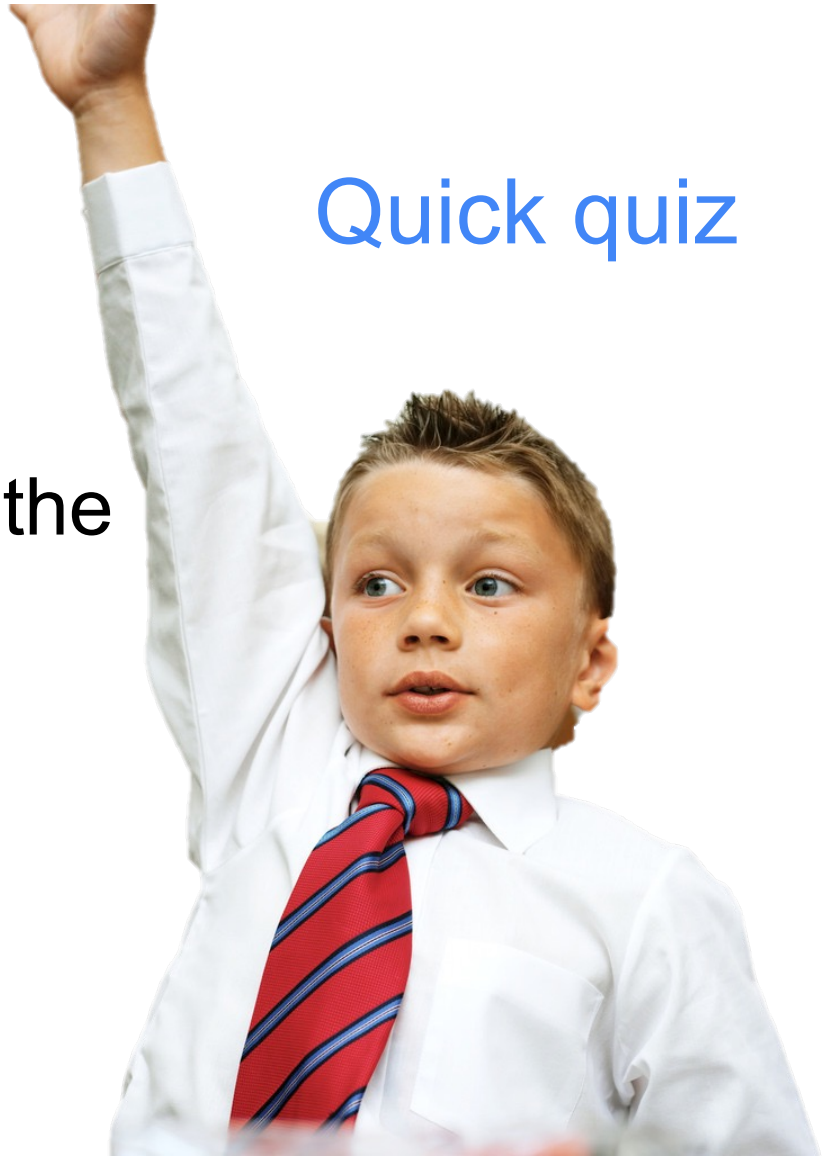
Quick quiz

How do you create a hyperlink?

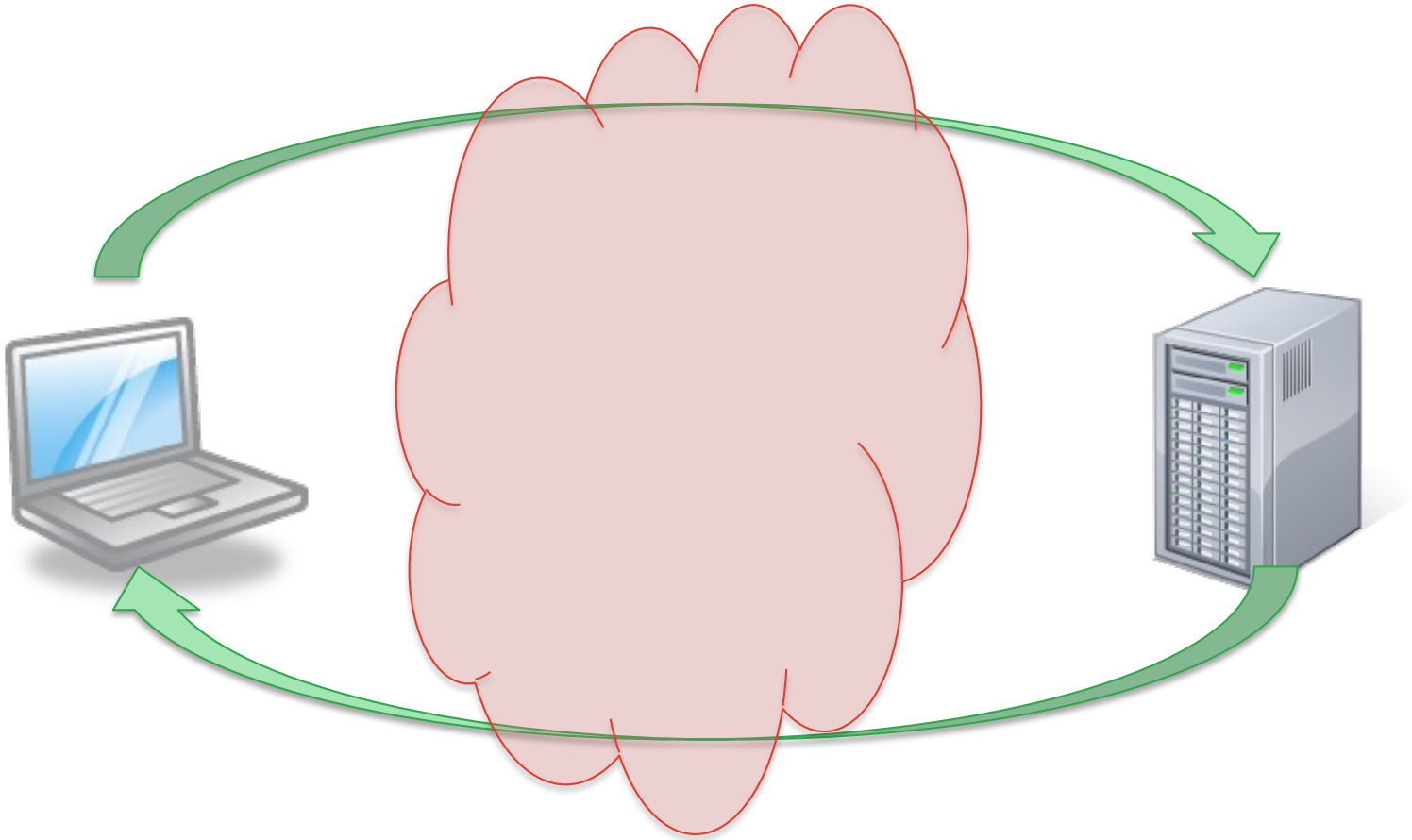
With the <a> tag

And what does a click do?

It sends a whole new request to the server



A new request reloads the entire bundle and clears your state!



So don't use an <a>, use a <Link>

- Write your links like this:

```
<Link to="/people">All people</Link>
<Link to="/people/{p.id}">This Person</Link>
<Link to="/teams">Teams</Link>
<Link to="/cart">My cart</Link>

<Link to="/someLink">Text to display</Link>
```

tl;dr

- Read route parameters via the useParams hook
- Optional route parameters are possible when defining the route by putting a "?" after them
- You can read query strings with the useSearchParams() hook