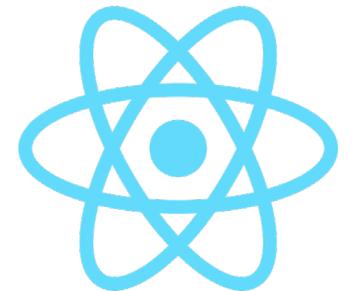# Introduction to React

A workshop for COMP 523

Aaron Smith
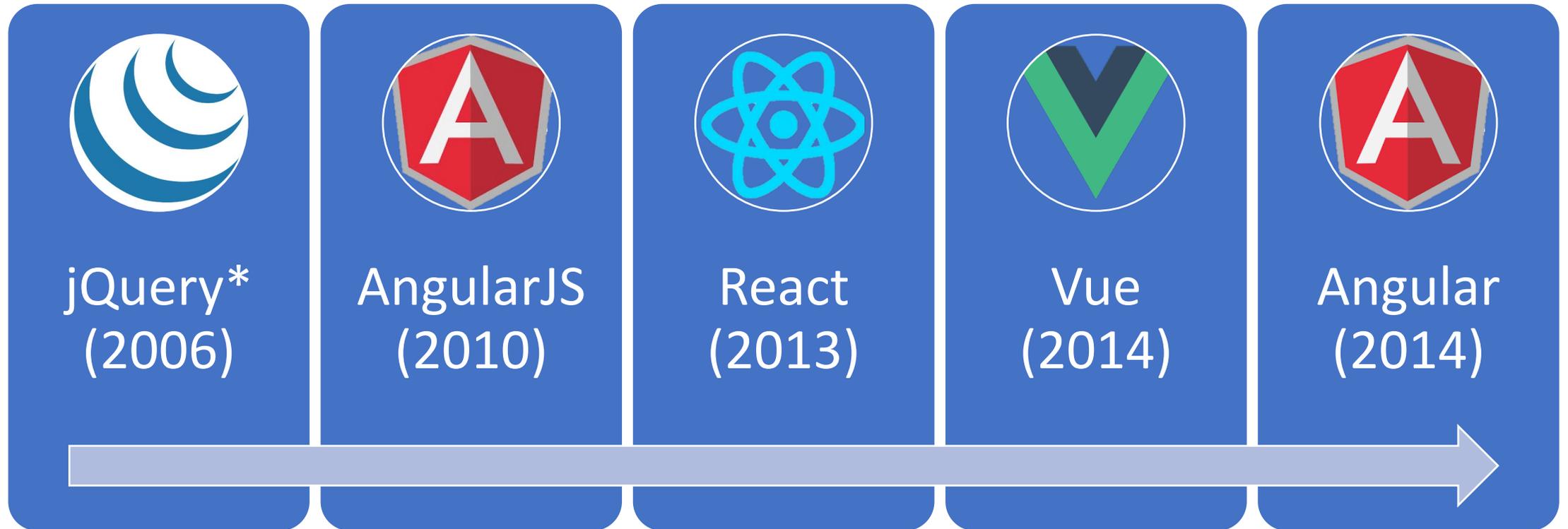
Monday, Feb. 10, 2020

What is React?

- React is a **JavaScript** **framework**

- Used for **front end** **web development**

- Think of jQuery, but more structured

- Created and used by **Facebook**

- Famous for implementing a **virtual dom**

Timeline of front-end JavaScript frameworks

jQuery* (2006) — AngularJS (2010) — React (2013) — Vue (2014) — Angular (2014)

* jQuery is more often considered a **library** than a **framework**

Common tasks in front-end development

App state	Data definition, organization, and storage
User actions	Event handlers respond to user actions
Templates	Design and render HTML templates
Routing	Resolve URLs
Data fetching	Interact with server(s) through APIs and AJAX

Fundamentals of React

1. JavaScript and HTML in the same file (JSX)

2. Embrace functional programming

3. Components everywhere

JavaScript and HTML *in the same file*

HTML   **CSS**   **JS**

JSX   **CSS or JSS**

Traditional
approach

React
approach

JSX: the React programming language

```
const first = "Aaron";
const last  = "Smith";

const name = <span>{first} {last}</span>;

const list = (                          const listWithTitle = (
  <ul>                                    <>
    <li>Dr. David Stotts</li>               <h1>COMP 523</h1>
    <li>{name}</li>                         <ul>
  </ul>                                       <li>Dr. David Stotts</li>
);                                            <li>{name}</li>
                                            </ul>
                                          </>
                                        );
```

"React is **just JavaScript**"

Functional programming

1. Functions are "first class citizens"

2. Variables are immutable

3. Functions have no side effects

Functional programming

Functions are "first class citizens"

This means functions can be...

1. Saved as **variables**
2. Passed as **arguments**
3. **Returned** from functions

```javascript
let add = function() {
  console.log('Now adding numbers');
  const five = 3 + 2;
};
```

```javascript
function performTask(task) {
  task();
  console.log('Task performed!');
}

performTask(add);
```

```javascript
function foo() {
  return function() {
    console.log('What gets printed?');
  };
}

foo
foo();
foo()();
```

Functional programming

Variables are immutable

```
let a = 4;

a = 2; // Mutates `a`
```

```
let b = [1, 2, 3];

b.push(4); // Mutates `b`

let c = [...b, 4]; // Does not mutate `b`
```

Functional programming

Functions have no side effects

```
const b = [];

function hasSideEffects() {
  b = [0];
}
```

Components

Components are functions for user interfaces

Math function:

Input **x** →

```
let y = f(x);
```

→ Output **number**

Component function:

Input **x** →

```
let y = <FancyDiv value={x} />;
```

→ Output **HTML**

Anatomy of a React component

The component is just a function

Inputs are passed through a single argument called "props"

The function outputs HTML

```
export default function MyComponent(props) {
  return <div>Hello, world! My name is {props.name}</div>;
}

const html = <MyComponent name="aaron" />;
```

The function is **executed** as if it was an HTML tag

Parameters are passed in as HTML attributes

Component rendering

- When a component function **executes**, we say it "**renders**"

- Assume components may re-render at any time

Our job is to ensure that
every time the component re-renders,
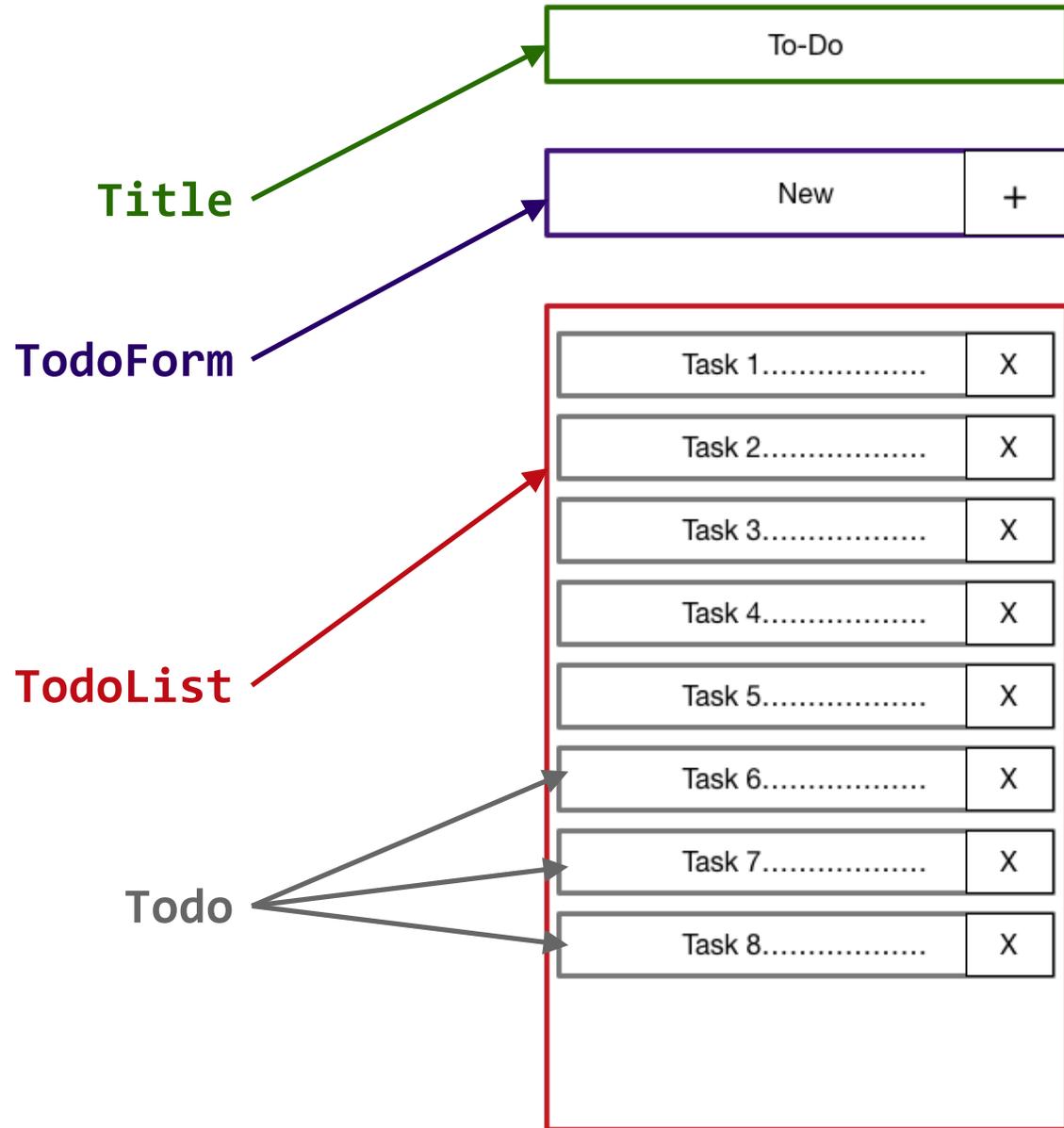the correct output is produced

"In React, everything is a component"

Todo application

Big idea:
- A digital to-do list

First step:
- mockup / wireframe

To-Do

Title

New | +

TodoForm

TodoList

Task 1.................. | X
Task 2.................. | X
Task 3.................. | X
Task 4.................. | X
Task 5.................. | X
Task 6.................. | X
Task 7.................. | X

Todo

Task 8.................. | X

Creating a new React app

Creating a new React app is simple!

1. Install Node.js

2. Run: **npx create-react-app** app-name

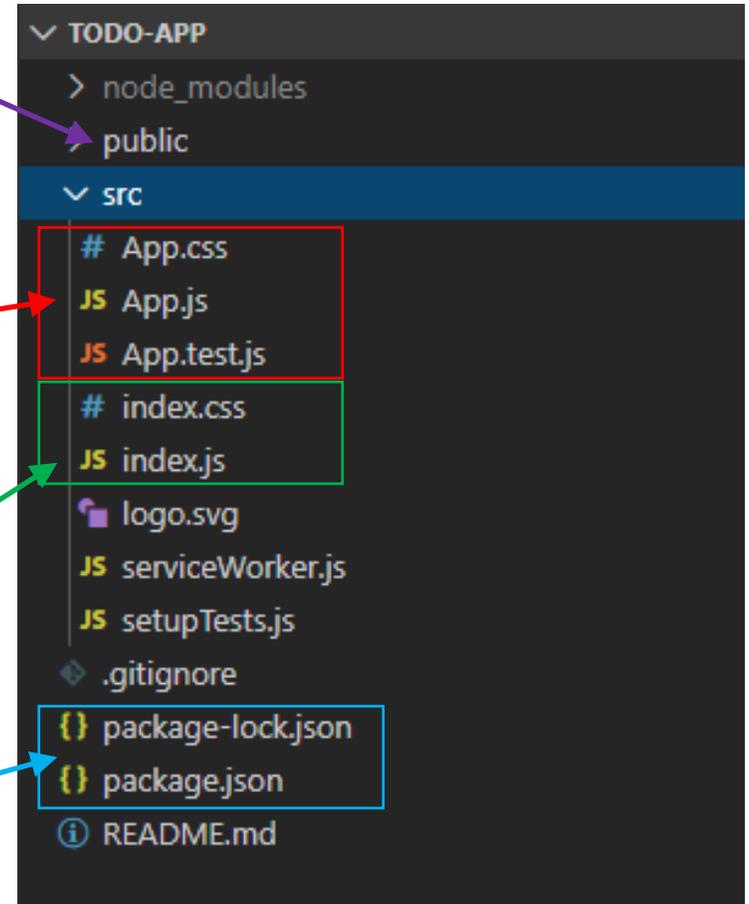3. New app created in folder: **./**app-name

Anatomy of a new React app

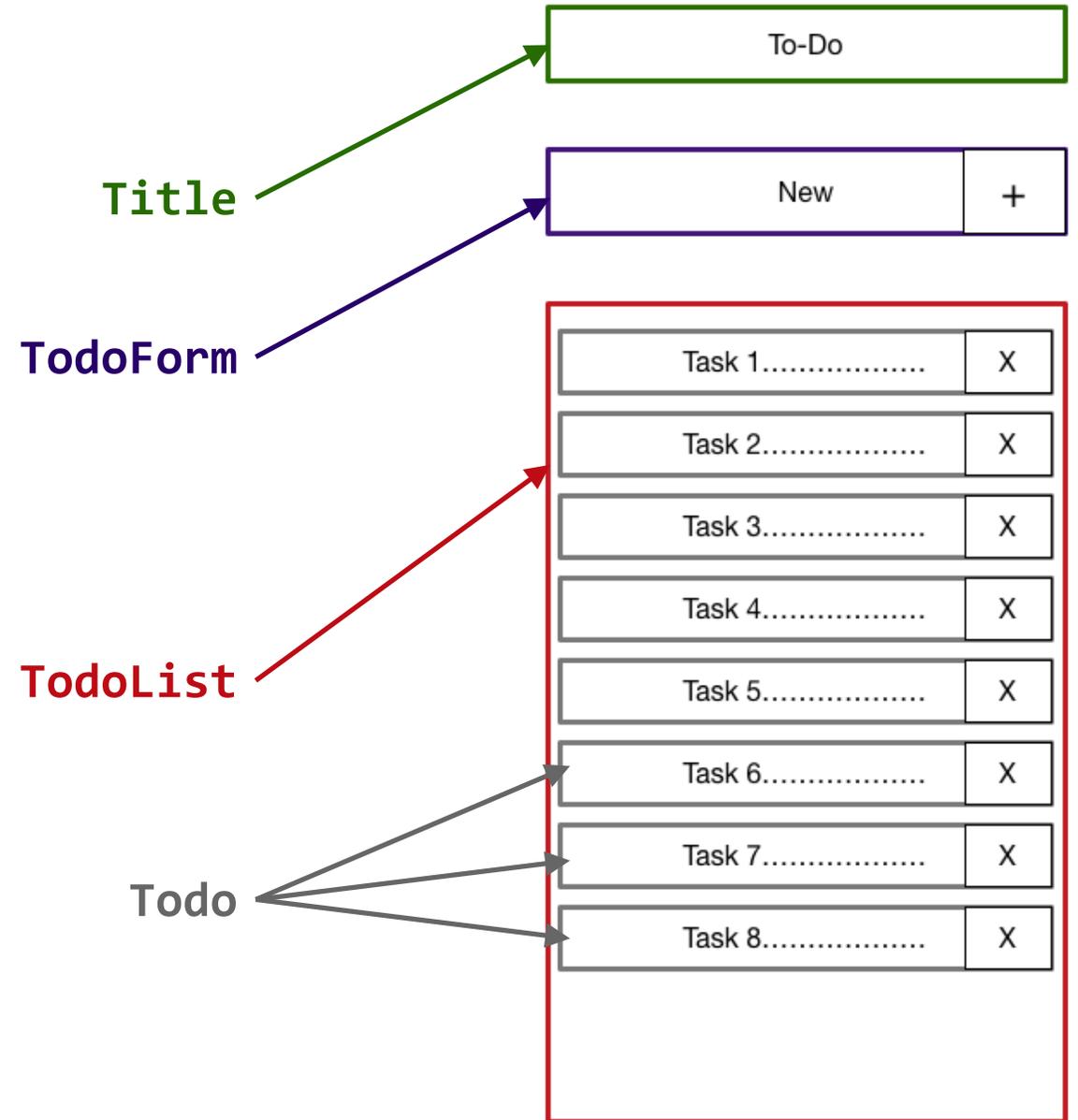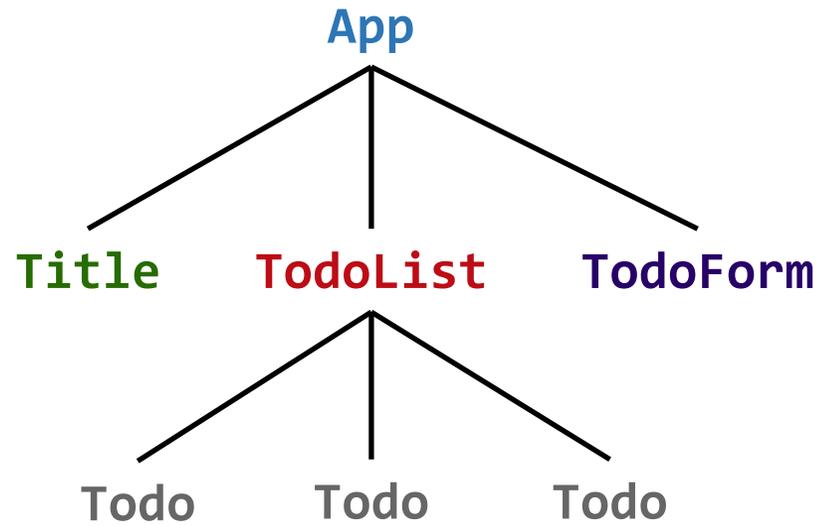**public** holds the initial html document and other static assets

App is a boilerplate starter component

index.js binds React to the DOM

package.json configures npm dependencies

> TODO-APP
> > node_modules
> public
> ∨ src
>   # App.css
>   JS App.js
>   JS App.test.js
>   # index.css
>   JS index.js
>   logo.svg
>   JS serviceWorker.js
>   JS setupTests.js
>   .gitignore
>   {} package-lock.json
>   {} package.json
>   ⓘ README.md

Component Hierarchy

Special list key property

- **Situation:** Display a **dynamic array of elements**

- Must specify a special "**key**" property for each element

- The key of an item **uniquely identifies it**

- Used by React internally for **render optimization**

- Can be any unique value (string or number)

What are hooks?

Hooks: Special functions that allow developers to hook into **state** and **lifecycle** of React components.

State: One or more data values associated with a React component instance.

Lifecycle: The events associated with a React component instance (create, render, destroy, etc).

Built-in hooks:

We will cover these today
{
```
useState

useEffect
```

We will **not** cover these today
{
```
useReducer

useMemo

useRef

useCallback
```

First React hook: useState

Purpose:

1. Remember values internally when the component re-renders

2. Tell React to re-render the component when the value changes

Syntax:

```
const [val, setVal] = useState(100);
```

The current value | A setter function to change the value | The initial value to use

Predicting component re-rendering

A component will only re-render when...

1. A value inside **props** changes

      – or –

2. A **useState** setter is called

This means all data values displayed in the HTML should depend on either **props** or **useState**

Second React hook: useEffect

Purpose:

Act as an **observer**, running code in response to value changes

Syntax:

```
useEffect(() => {
  console.log(`myValue was changed! New value: ${myValue}`);
}, [myValue]);
```

A list of values such that changes should trigger this code to run

The code to run when values change

Building a React project

- When you're ready to launch your app, run this command:

  npm run build

- This bundles your app into CSS/JS/HTML files and puts them in the **/build** folder

- These files can be served from an AWS S3 bucket

3rd party components and libraries

- React-Router

- Redux

- Material-UI

- Bootstrap

- Font-Awesome

- SWR