

# **React 1**

**CS571: Building User Interfaces**

**Cole Nelson & Yuhang Zhao**

# Before Lecture

- Download and install [NPM 9 and Node 18](#)
- Clone [today's code](#) to your machine.
  - Run the command `npm install` inside of the `starter` folder.

# Disclaimer

As with JS, this is not a comprehensive introduction to React, so be sure to check out the [React Docs](#).

Avoid [legacy.reactjs.org](https://legacy.reactjs.org)

# What will we learn today?

- History and overview of React
- Setting up a React project
- Building a basic React project
- `useState` and `useEffect` hooks
- Using other components
- Common issues

# What is React?

**Definition:** Also called ReactJS, React is a JS library for building user interfaces.

- Developed by Facebook, dating back to 2010.
- Started as an internal development tool, then open-sourced in 2013.

[More on the history of React](#)

# Why should we use React?

Among many reasons...

- `innerHTML` is bad!
- Efficient DOM updating
- Declarative programming

**innerHTML is bad!**

# Review: innerHTML

```
<html>
  <head></head>
  <body>
    <h1>Badger Book</h1>
    <div id="student"></div>
  </body>
</html>
```

```
const name = "Cole"; // get data from api
document.getElementById("student").innerHTML = '<h2>' + name + '</h2>'
```

# XSS

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

**Remember:** You supplied the data in HW0!

OWASP Definition

# Badger Book XSS

```
<html>
  <head></head>
  <body>
    <h1>Badger Book</h1>
    <div id="student"></div>
  </body>
</html>
```

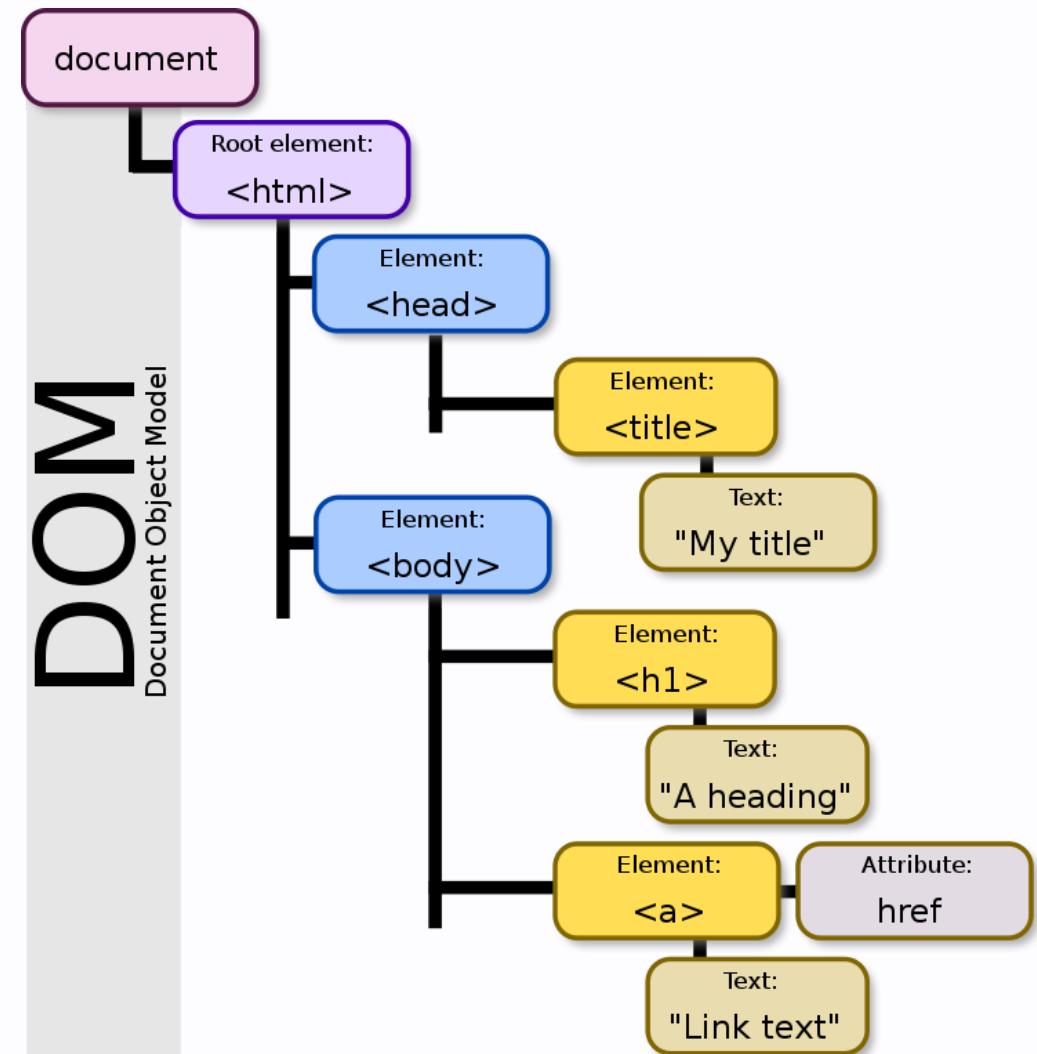
```
const name = "<image src='abc.png' onerror='alert(document.cookie)'>";
document.getElementById("student").innerHTML = '<h2>' + name + '</h2>'
```

Try it! <https://cs571.org/api/f23/hw2/students-xss>

# **Efficient DOM Updating**

# Refresher

**Definition:** Document Object Model (DOM) translates an HTML document into a tree structure where each node represents an object on the page.



[Wikipedia: DOM](#)

For JS to interact with user-facing elements, we use to access them via the `document`, e.g.

```
document.getElementById .
```

But in React...

```
for (let i = 0; i < 10; i++) {  
  console.log('I will no longer use document to access DOM elements.');//  
  console.log('I will no longer use document to manipulate DOM elements.');//  
  console.log('I will let React handle the DOM for me.');//  
}
```

# What's so bad about the DOM?

Interactively directly with it can be slow!

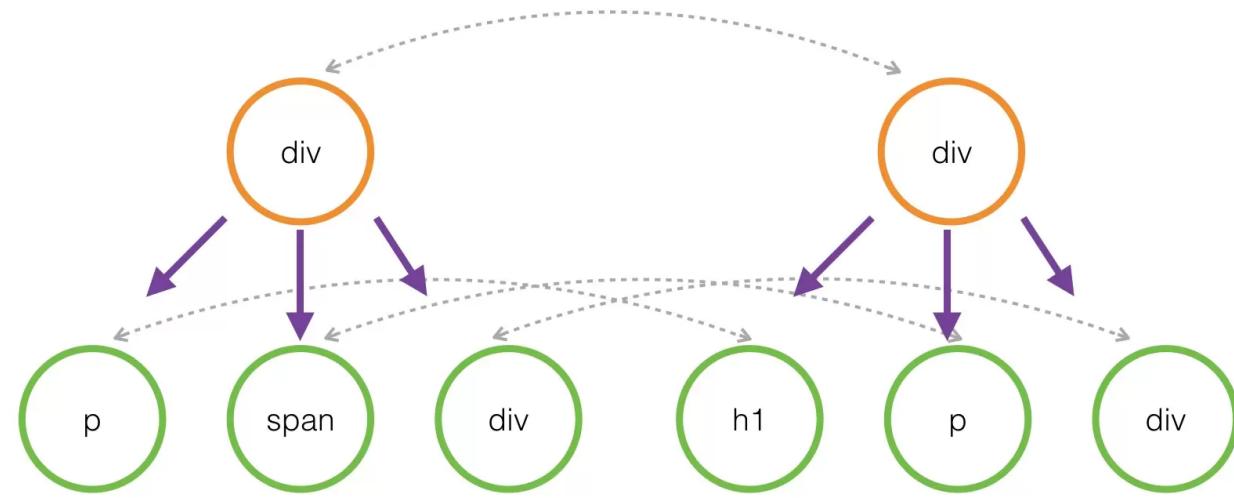
- *Single-page applications (SPAs)* can be huge.
- Interactive applications require a large number of and frequent updates on DOM elements.



Image Source

# Solution: The *Virtual DOM*

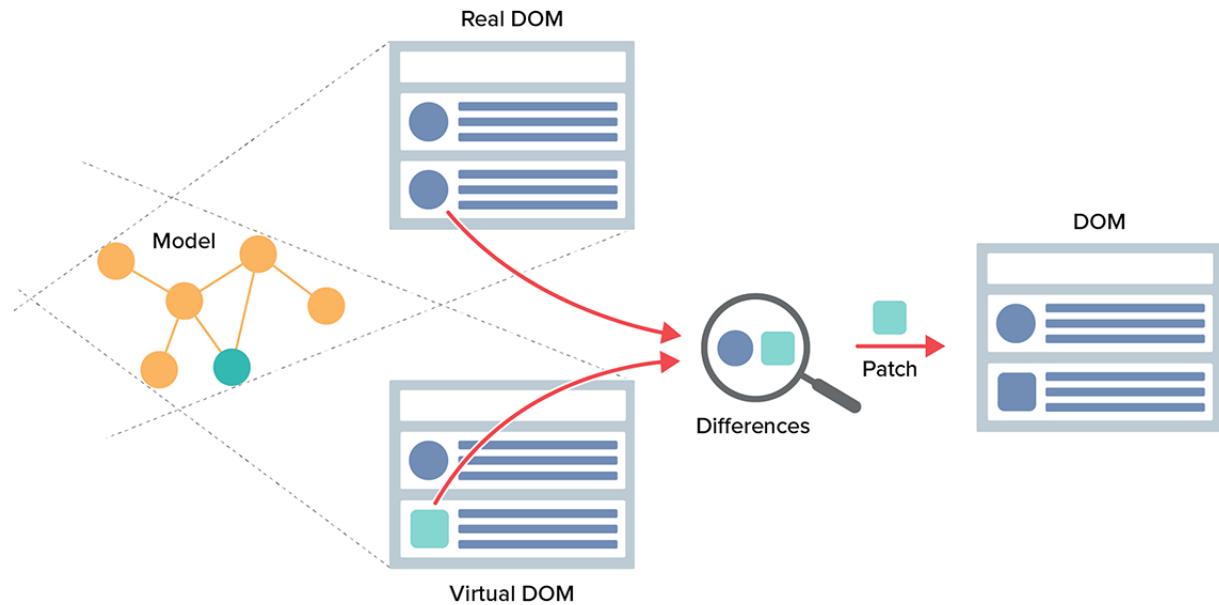
**Definition:** The virtual DOM is a *virtual* representation of the user-facing elements that are kept in memory and synced with the real DOM when DOM elements are updated.



# Virtual DOM: *Reconciliation*

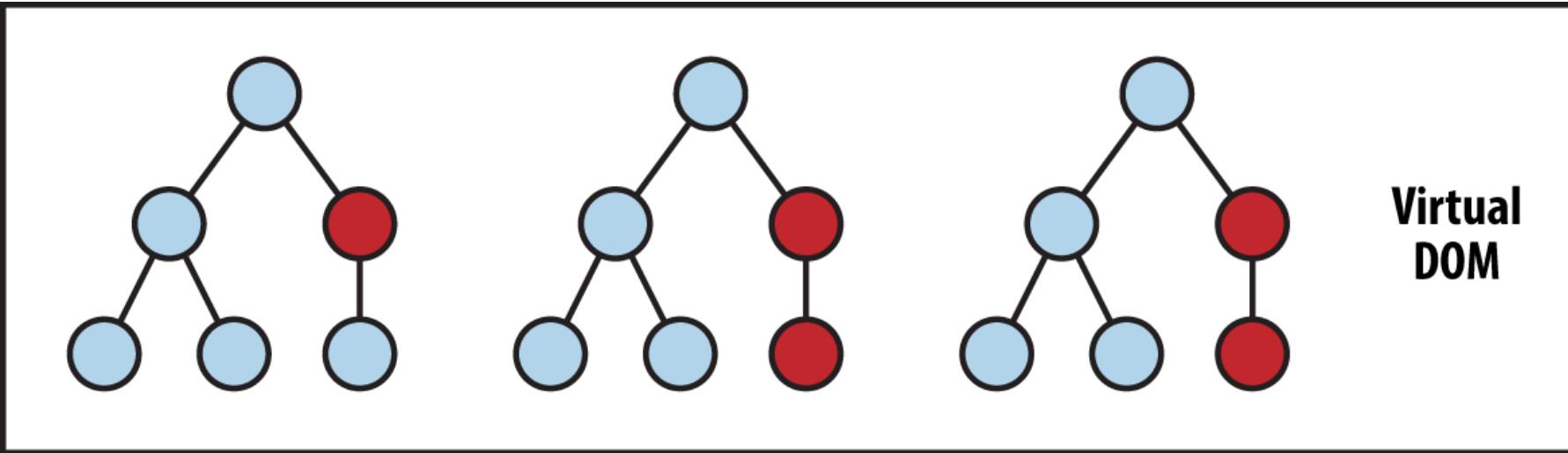
## Definition:

*Reconciliation* is the process of *diffing* and syncing the virtual and real DOM to render changes for the user.



[Image Source](#)

CS571 Building User Interfaces | Cole Nelson & Yuhang Zhao | Lecture 06: React 1



State Change

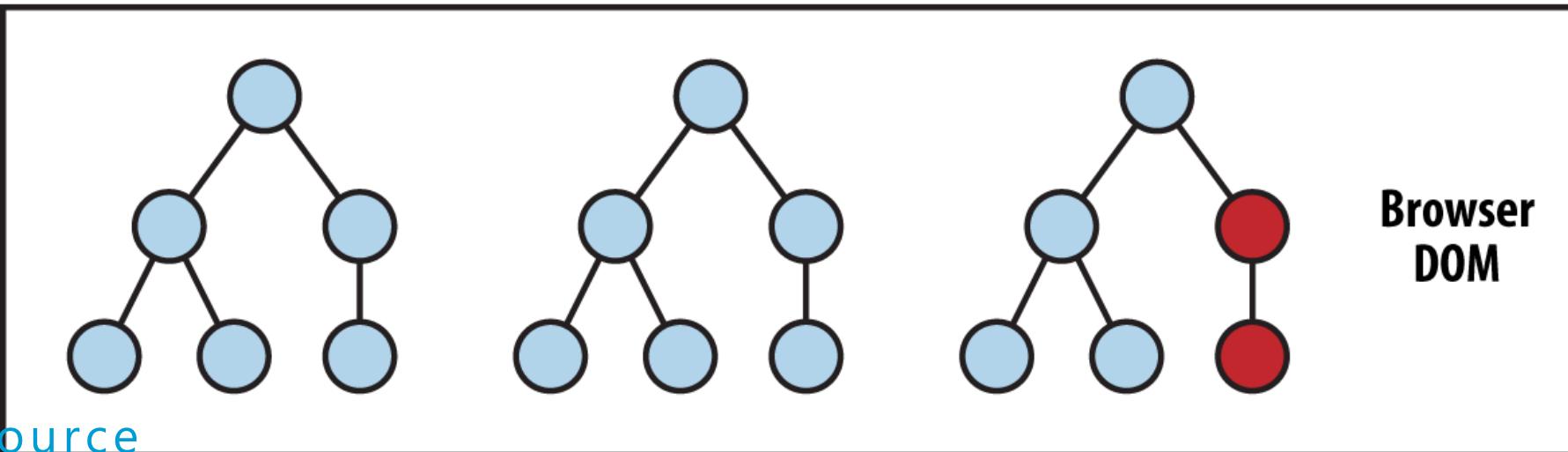


Compute Diff



Re-render

**Virtual  
DOM**



**Browser  
DOM**

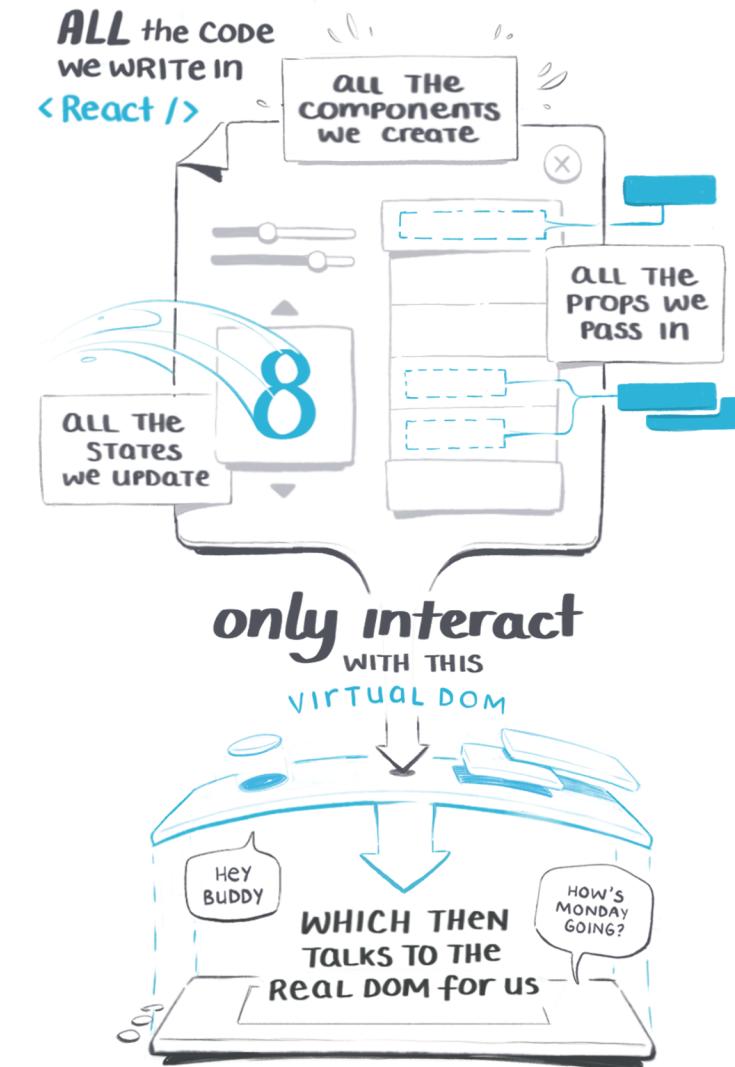
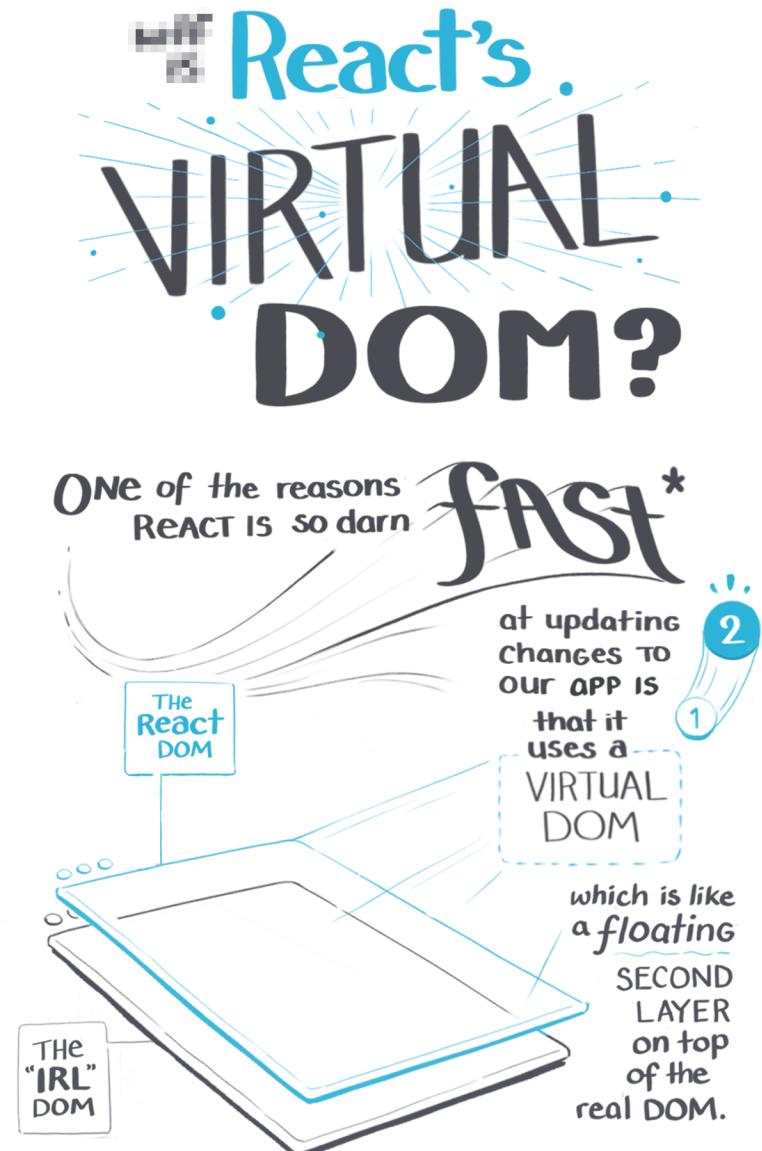
[Image Source](#)

# What are the benefits of a Virtual DOM?

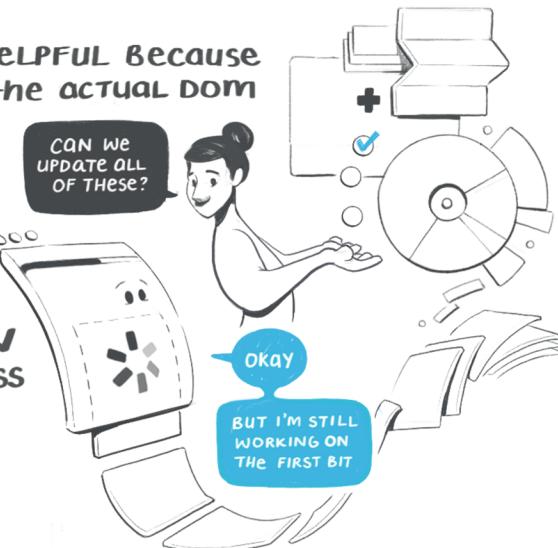
- Incredibly fast, as only what is updated in the Virtual DOM is updated in the real DOM.
- Abstracts away interactions with DOM; makes programming more *declarative*.
- Used in React and vue.js; Angular does its own thing.

# The ReactDOM

Credit: [Maggie Appleton](#)



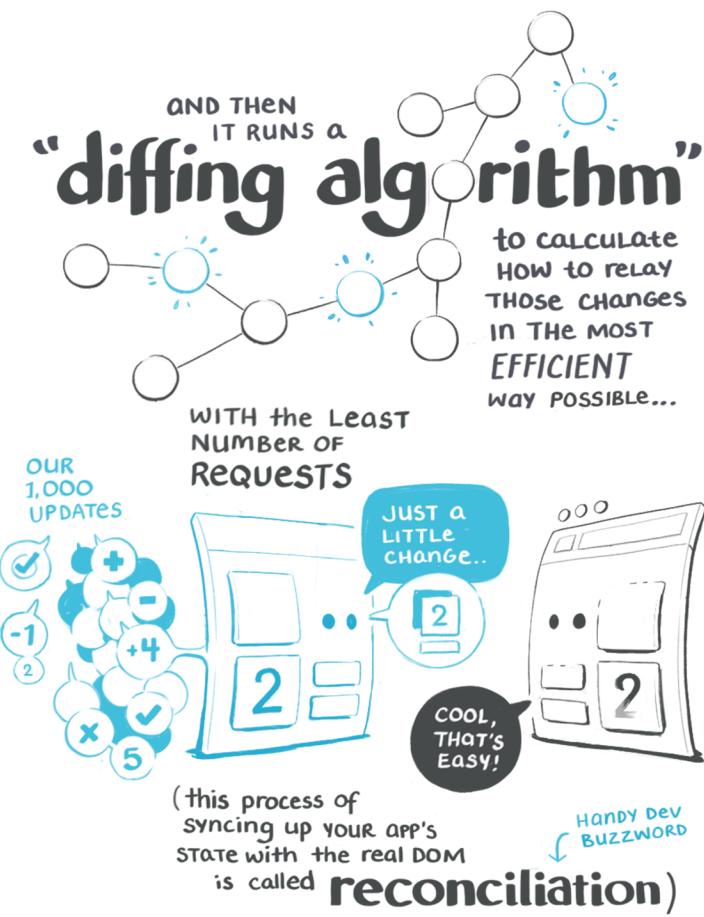
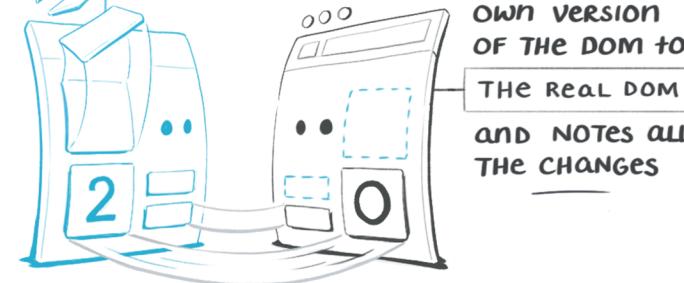
This is helpful because passing the actual DOM lots of changes to our app's state is a slow process



whereas, when we pass

the VIRTUAL DOM

a pile of updates...



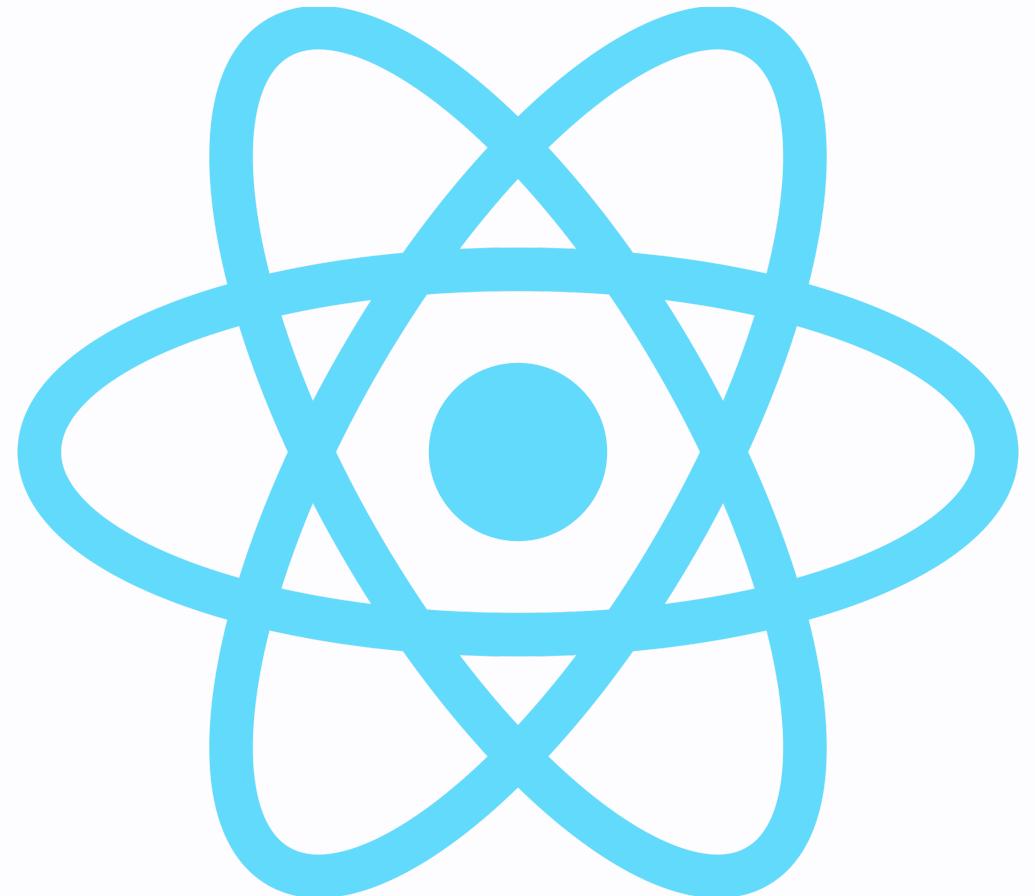
\* I know React's speed is a hotly debated point. To clarify: The virtual DOM doesn't speed up our initial render. It's only quicker at rendering changes to state, compared to the bog-standard, unoptimised DOM. It's also not the absolute fastest solution to the issue of slow DOMs.

# Declarative Programming

Every component is a function!

# React

by Meta





## React in 100 Seconds

# Getting Started

What you will need: *terminal, IDE, NPM, and Node.js*

```
npm create vite@latest my-app-name -- --template react  
cd my-app-name  
npm install  
npm run dev
```

For the HWs, these steps will already be done for you.

# Getting Started

For an existing React project, you simply need to...

```
npm install  
npm run dev
```

Clone, install, and start the starter code. You can install and start the completed code later.

This has `react-bootstrap` installed.

# What did I just run?

`npm install` downloads the dependencies from your `package.json` to `node_modules`

`npm run dev` starts a local webserver. We don't open `index.html` anymore -- we go to `localhost:5173`

# React Essentials

Every "thing" is a component.

Every component is a function, inheriting `props` and maintaining an internal `state`.

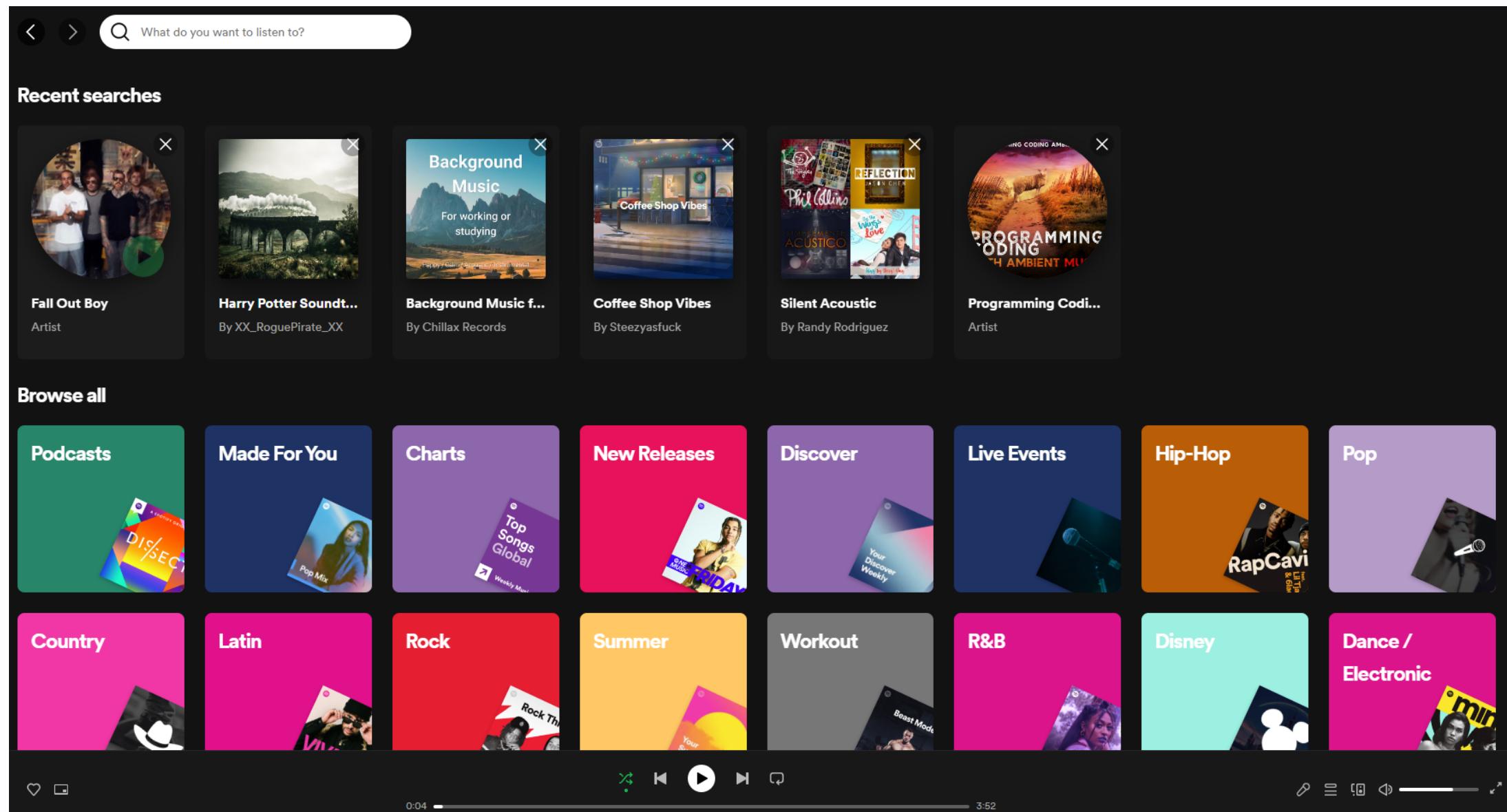
A component will re-render when...

1. its `props` changes
2. its `state` changes
3. its parent re-renders

# What defines a component?

- Similar question: *what defines a class in Java?*
- Some re-usable piece of the interface.
- May have many children, but only one parent.

**Identify components in the next slide..**



# Example of a React Component

This React component displays Hello World on the webpage using JSX.

The `return` is what is displayed for a component.

```
function Welcome() {  
  return <h1>Hello World!</h1>;  
}
```

ESBuild transpiles JSX into HTML, CSS, and JS.

StackBlitz

# Note: Class vs Functional Components

A class component looks like this...

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello World!</h1>;  
  }  
}
```

Functional components were introduced in React 16.8. They are the most commonly used in new React code. We will not cover class components in this course.

# React Components

React components can have props given by its parent...

```
function App() {
  return (
    <div>
      <Welcome person="Charlie"></Welcome>
      <Welcome person="Jessica"></Welcome>
      <Welcome person="Tonya"></Welcome>
    </div>
  );
}
function Welcome(props) {
  return <h1>Welcome, {props.person}</h1>;
}
```

# React Components

...or can maintain an internal state.

```
function Welcome() {  
  const [name, setName] = useState("Alba");  
  return <h1>Welcome, {name}</h1>;  
}
```

StackBlitz

# React Components

... or have both!

```
function App() {
  return <Welcome message="Good evening, "></Welcome>
}

function Welcome(props) {
  const [name, setName] = useState("Rodriguez");
  return <h1>{props.message} {name}</h1>;
}
```

StackBlitz

# JSX Basics

Curly braces `{}` interpolate a JS expression.

```
<div>
  <h2>{name}</h2>
  <a href={url} target="_blank">My Website</a>
  {
    classes.map(clazz =>
      <div key={clazz.name}>
        <p>{clazz.name} is worth {clazz.creds} credits</p>
      </div>
    )
  }
</div>
```

# React Hooks

Hooks are small React features. Today, we will cover...

- useState
- useEffect

# useState Hook

Used to maintain state! Takes an initial value as an argument. Returns a pair of the *read-only* state value and a *mutator* function.

*Always* use the mutator function to modify state.

*Never* modify the state directly.

```
const [name, setName] = useState("James");
```

# useState Hook

```
const [name, setName] = useState("James");
```

We can use `name` to *read* the name and `setName` to *change* the name...

```
console.log(name);
setName("Jim");
console.log(name); // still James???
```

`setName` happens *asynchronously*. See `useEffect`.

# useEffect Hook

Used to perform an action on component load or state change. Takes a callback function and an array of state dependencies as arguments.

```
useEffect(() => {
  alert("The page has been reloaded!");
}, [])
```

```
useEffect(() => {
  alert("You changed your name to " + name);
}, [name])
```

# useState Hook

```
const [name, setName] = useState("James");
```

The mutator can be called like...

```
setName("Jim");
```

... or with a callback function of the previous value.

```
setName(oldName => oldName.substring(0, oldName.length - 1));
```

# useState Hook

Why is this useful? Arrays!

```
const [names, setNames] = useState(["James", "Jess"]);
```

Remember we cannot *mutate* the state variable.

```
setNames((oldNames) => [...oldNames, "Jim"]);
```

We cannot (rather, should not) do push .

# Why?

When does React re-render a component? When...

1. its `props` changes
2. its `state` changes
3. its parent re-renders

# So...

## Good Example

```
const [name, setName] = useState("James");
// ...
setName("Jim"); // Good! React will pick up on this change and re-render soon.
```

## Bad Example

```
let name = "James";
// ...
name = "Jim"; // Bad! React won't pick up on this.
```

# Imports and Exports

Functions must be exported to be used in other files,  
e.g. `export default FindMyBadgers` .

This can then be imported, e.g. `import FindMyBadgers  
from "./components/FindMyBadgers"`

# Imports and Exports

Functions can export one object as default, other exports can be non-default, e.g. `export HelperFunc2` .

These can then be imported, e.g.

```
import { HelperFunc2 } from "./utils/HelperFuncs"
```

# Imports and Exports

Imports from 3rd party libraries do *not* use relative pathing, e.g.

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

```
import { Container } from 'react-bootstrap'
```

Getting a 'not defined' error? Check your imports!

# Let's make a React App!

Find my Badgers using [randomuser.me](https://randomuser.me)

[StackBlitz Solution](#)

Also see the solution from today's example!

# Common HW Issues

# Blank Screen

Use your Developer Tools! (F12)

Check both the "Console" and "Network" tabs.



CS571 @ UW-Madison

# HTTP 401

Unauthorized...

Are you logged in?

## BadgerAuth Center

You are currently logged in using Badger ID starting with  
41f69d8.

Would you like to logout?

**You will not be able to use CS571 APIs until you are logged back in.**

Logout

# HTTP 429

Too many requests...

```
useEffect(() => {  
  // your code here!  
}, []) // <--- don't forget me!
```

... did you forget the  
empty dep. array?

	Status	Type	Initiator	Size	Time
ured-baked-good	200	fetch	BadgerBakery...	307 B	107 ms
ured-baked-good	200	fetch	BadgerBakery...	587 B	173 ms
ured-baked-good	200	fetch	BadgerBakery...	587 B	168 ms
ured-baked-good	200	fetch	BadgerBakery...	587 B	171 ms
ured-baked-good	429	fetch	BadgerBakery...	0 B	160 ms
ured-baked-good	429	fetch	BadgerBakery...	0 B	356 ms

quests | 110 kB transferred | 4.1 MB resources | Finish: 1.1 min

# X is not defined

Did you import it? e.g.

## Common React Imports

```
import { useState, useEffect } from "react";
```

- ✖ ➔ Uncaught ReferenceError: Button is not defined  
at BakedGood ([BakedGood.jsx:20:14](#))
- ✖ ➔ Uncaught ReferenceError: Card is not defined  
at BakedGood ([BakedGood.jsx:15:13](#))

## Common Bootstrap Imports

```
import { Button, Card } from "react-bootstrap";
```

# Empty State

Setting state is asynchronous!

```
const [name, setName] = useState("James");
console.log(name);
setName("Jim");
console.log(name); // still James!
```

Use `useEffect` to listen for changes.

```
useEffect(() => {
  console.log("Your new name is... " + name)
}, [name])
```

# What did we learn today?

- History and overview of React
- Setting up a React project
- Building a basic React project
- `useState` and `useEffect` hooks
- Using other components
- Common issues

# Questions?