React

Introduction & Overview

New Horizons®

Outline

Why React?

Tradeoffs

Why not React?

Timeline

- 2011 - Created by Facebook
- 2012 - Used on Instagram
- 2013 - open sourced (controversial because of markup and logic being placed in the same file)
- 2014 - Embraced by many large companies
- 2015 - React Native open sourced as well
- 2016 - React 15 Released (previous version was 0.14)
- 2022 - React 18 Released with automatic patching for updates and FB rebrand to Meta
- Today - FB has full-time dev staff that regularly releases fixes and updates

Why React?

- Flexibility
- Developer Experience
- Corporate Investment
- Community Support
- Performance
- Testability

Flexibility

- You can build variety of interfaces for different platforms and different use cases
- It is in fact a library and not a framework (as Angular)
- Initially intended for components for web apps, today you can build static sites, mobile apps and desktop applications (using Electron), server rendering (via Next.js) and even virtual reality software (React VR)

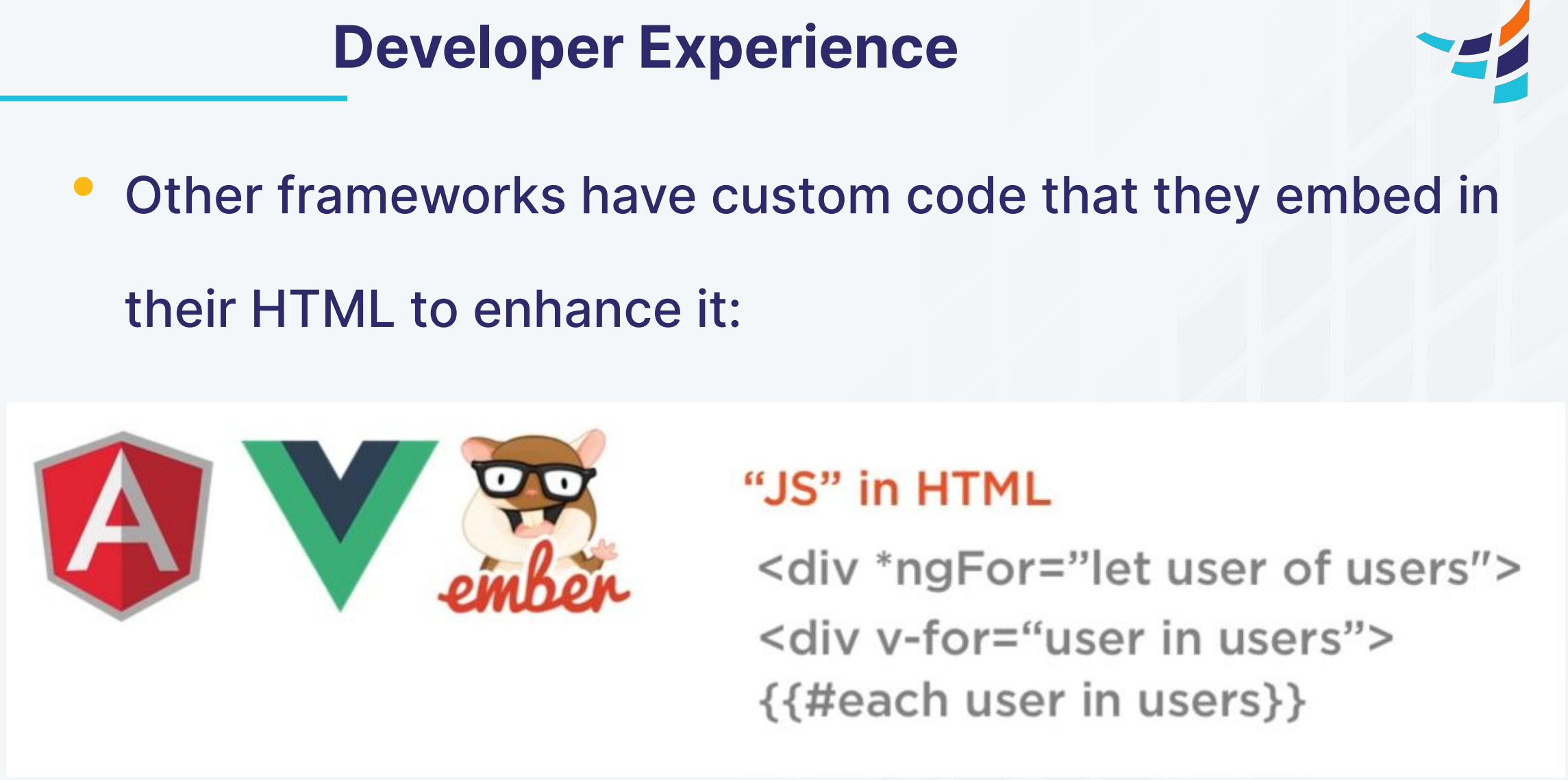
Flexibility

- Existing app? No problem, sprinkle React in
- In fact Facebook slowly used React to replace it’s server-side rendered PHP applications
- You can start by rewriting small portions of your page / application and move to bigger and bigger parts until you rewrite the entire page completely
- Broad Browser Support - it will continue being backed by FB

Developer Experience

- React uses small and simple API which is easy to learn so you will rarely have to use the docs
- You will use already known JavaScript features such as basic imports, creating functions (that turn into components) and so on
- JSX looks like HTML and compiles to JS - easy to learn as well, more for that later

Developer Experience



- Other frameworks have custom code that they embed in their HTML to enhance it:

"JS" in HTML

```
<div *ngFor="let user of users">
<div v-for="user in users">
{{#each user in users}}
```

Developer Experience

- React uses native JavaScript instead. In other word

  React puts “fake” HTML in JavaScript while other frameworks put “fake” JavaScript in HTML.

Corporate Investment

- Facebook is fully committed to maintaining React
- They maintain a blog which outlines each release:

  https://react.dev/blog
- When breaking changes occur, Facebook provides a Code Mod that automates the change. That's a command line tool that automates the changes and brings up your components up to date with the latest specification
- Facebook uses the Code Mods themselves on their own code as they have over 50 000 components in Production

Community

- React is in the Top 10 most starred repos in GitHub and that’s out of about 6 million repos total
- It is used by 10.5 million repositories and it has over 1500 contributors
- It has over 16 million downloads weekly
- Many online dev communities such as Reactiflux (170k developers in it only)
- Almost 400 000 topics in StackOverflow tagged with reactjs and more related

Community

- All of that means that the answer to your case likely exists online or you can find enough resources to solve it

Companies Using React

Community

- You don’t even have to write your own components as there are a lot of open source component libraries
- Microsoft maintains a huge library called Fluent UI
- Material UI has components that implement Google’s material design guidelines
- React Bootstrap has popular components that make it easy to work with React and Bootstrap
- Numerous standalone useful components on GitHub

Community

- Huge Ecosystem of tools built around React such as:

React Router

Redux

Mobx

Jest

GraphQL

Next.js

Performance

- React is fast by default as most JS libraries
- There are a lot of additional options that allow you to further optimize the performace

Performance

- You can Memoize expensive computations and components so they are only rerendered when necessary
- Specify low-priority updates so they are rendered only when React has time to do so
- Lazy Load components so user renders one page at a time
- Can server-render to improve load times
- Can generate as static HTML
- Powerful dev tools

Testability

- Testing the front end is hard thus often omitted but

  React provides good support to implement it

Traditional UI tests	React
Hassle to set up	Little to no config required
Requires browser	Run in-memory via Node
Slow	Fast
Brittle integration tests	Reliable, deterministic unit tests
Time-consuming to write, maintain	Write quickly, update easily

Testability

- The fact that React components are functions which will produce the same output given the same input allows you to write Reliable, Deterministic tests with No Side-Effects
- In general these tests do not rely on global state so you can easily maintain them in your code base

Testability

- React supports variety of testing JS frameworks

Mocha

Jasmine

Tape

QUnit

AVA

Jest

Tradeoffs

- After the sales pitch, what are the tradeoffs?

Framework	Library
Concise	Explicit
Template-centric	JavaScript-centric
Separate template	Single-file component
Standard	Non-standard
Community backed	Corporate backed

Framework vs Library

- Angular is a framework, React is considered library with neither being better, it’s more of a tradeoff
- Framework has clear opinions on how things should be done which leads to easier decision making and less setup overhead
- The library is light-weight and can be sprinkled into existing apps only picking what you need

Framework vs Library

- The framework comes bundled with a lot while in the library you get to choose what technologies to use

	React	Angular
Components	✓	✓
Testing	Jest, Mocha	✓
HTTP library	Fetch, Axios	✓
Routing	React Router	✓
I18n	react-intl	✓
Animation	react-motion	✓
Form validation	react-forms	✓
CLI	create-react-app	angular-cli

Concise vs Explicit

Two-way binding

Less coding

Automatic

Doable with React, but not popular or recommended.

```
let user = 'Cory';

<input
  type="text"
  value={user}
/>
```

One-way binding

More control

More explicit

Easy to debug

```
const [user, setUser] = useState("Cory");

function handleChange(event) {
  setUser(event.target.value);
}

<input
  type="text"
  value={user}
  onChange={handleChange}
/>;
}
```

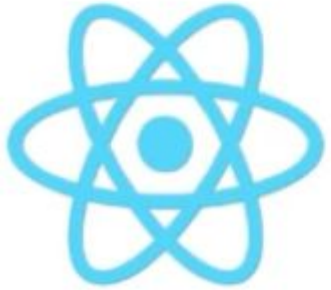
Concise vs Explicit

- React requires (most of the time) more typing to implement the traditional two way approaches but with the benefit of easier maintenance, greater clarity, liability and performance making it easier to debug

Template-centric vs JavaScript-centric

"JS" in HTML

Learn their unique syntax



"HTML" in JS

Learn JS

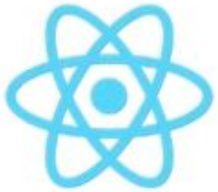
Template-centric vs JavaScript-centric

Conditional

```
<h1 *ngIf="isAdmin">Hi Admin</h1>
```

```
<h1 v-if="isAdmin">Hi Admin</h1>
```


```
<h1>{{if isAdmin 'Hi Admin'}}</h1>
```



```
{isAdmin && <h1>Hi Admin</h1>}
```

Since plain JS, you get:
1. Autocomplete support
2. Error messages

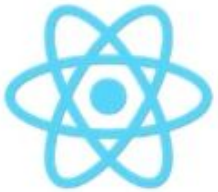
Template-centric vs JavaScript-centric

Loop

```
<div *ngFor="let user of users">{{user.name}}</div>
```

```
<div v-for="user in users">{{user.name}}</div>
```

```
{{#each users as |user|}}
<div>{{user.name}}</div>
{{/each}}
```



```
users.map(user => <div>{user.name}</div>)
```

map is a function for iterating over arrays that's built into JS

Template-centric vs JavaScript-centric

Template-centric

Little JS knowledge required

Avoid confusion with JS binding

Rule of least power

JavaScript-centric

Little framework-specific syntax

Fewer concepts to learn. It's JS.

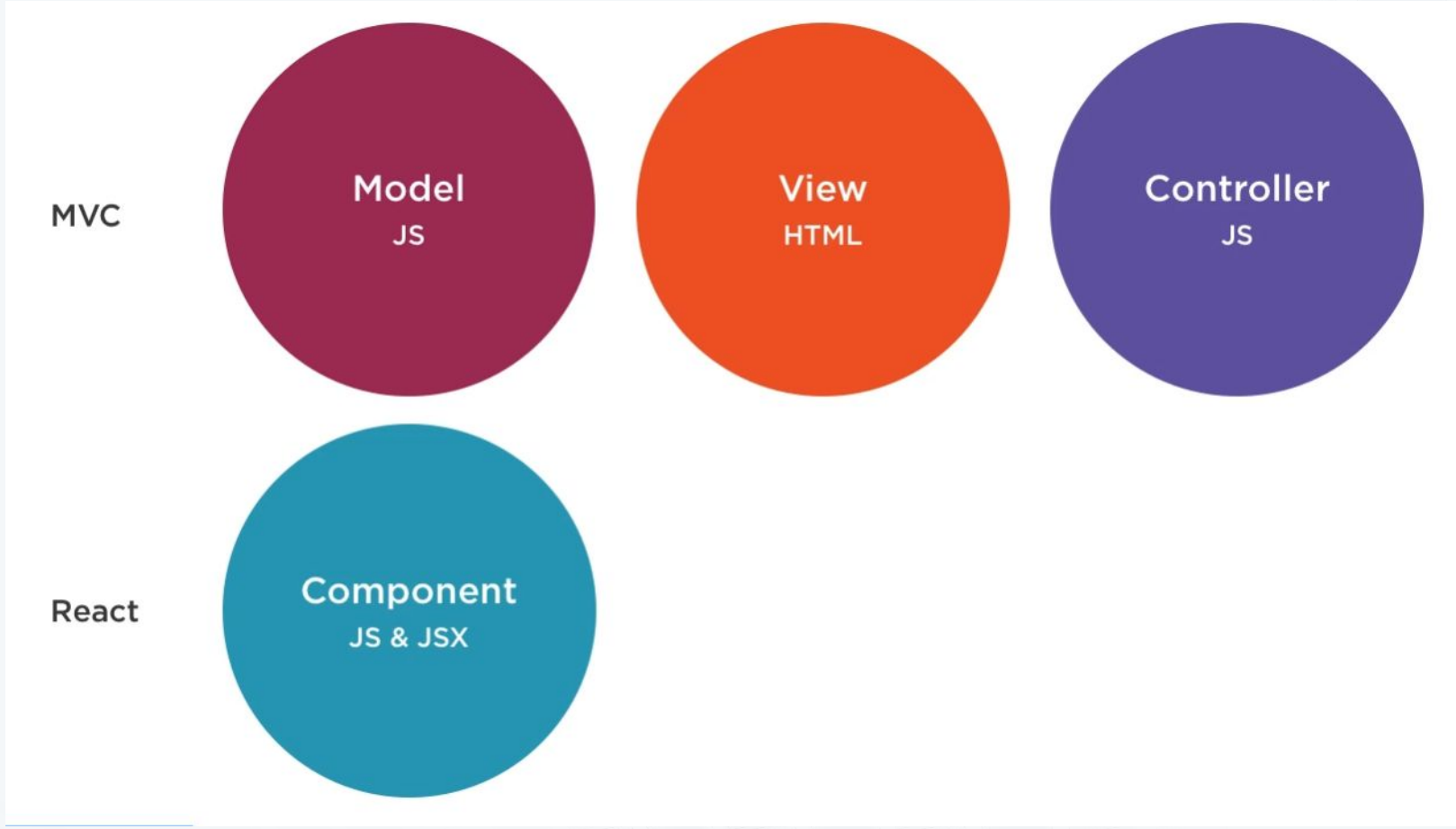
Less code

Easy to read

Encourages improving JS skills

Separate Template vs Single File

- In traditional MVC setup we have 3 separate files



Separate Template vs Single File

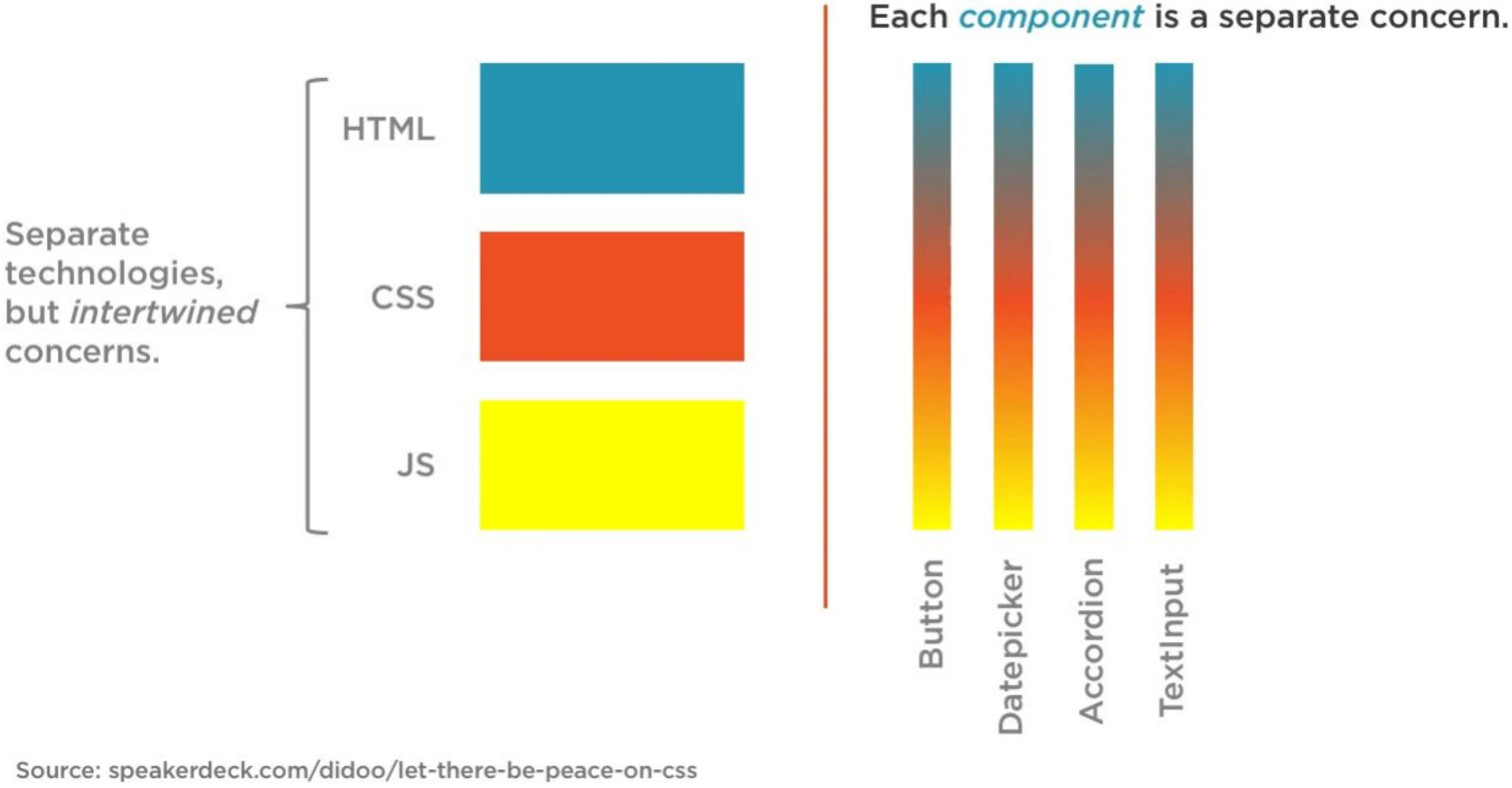
- In React we have logic and markup in the same file

  which is basically against best adopted practices

- What about separation of concerns?

  Well, React just thinks about the concerns in a little bit different way

Separate Template vs Single File



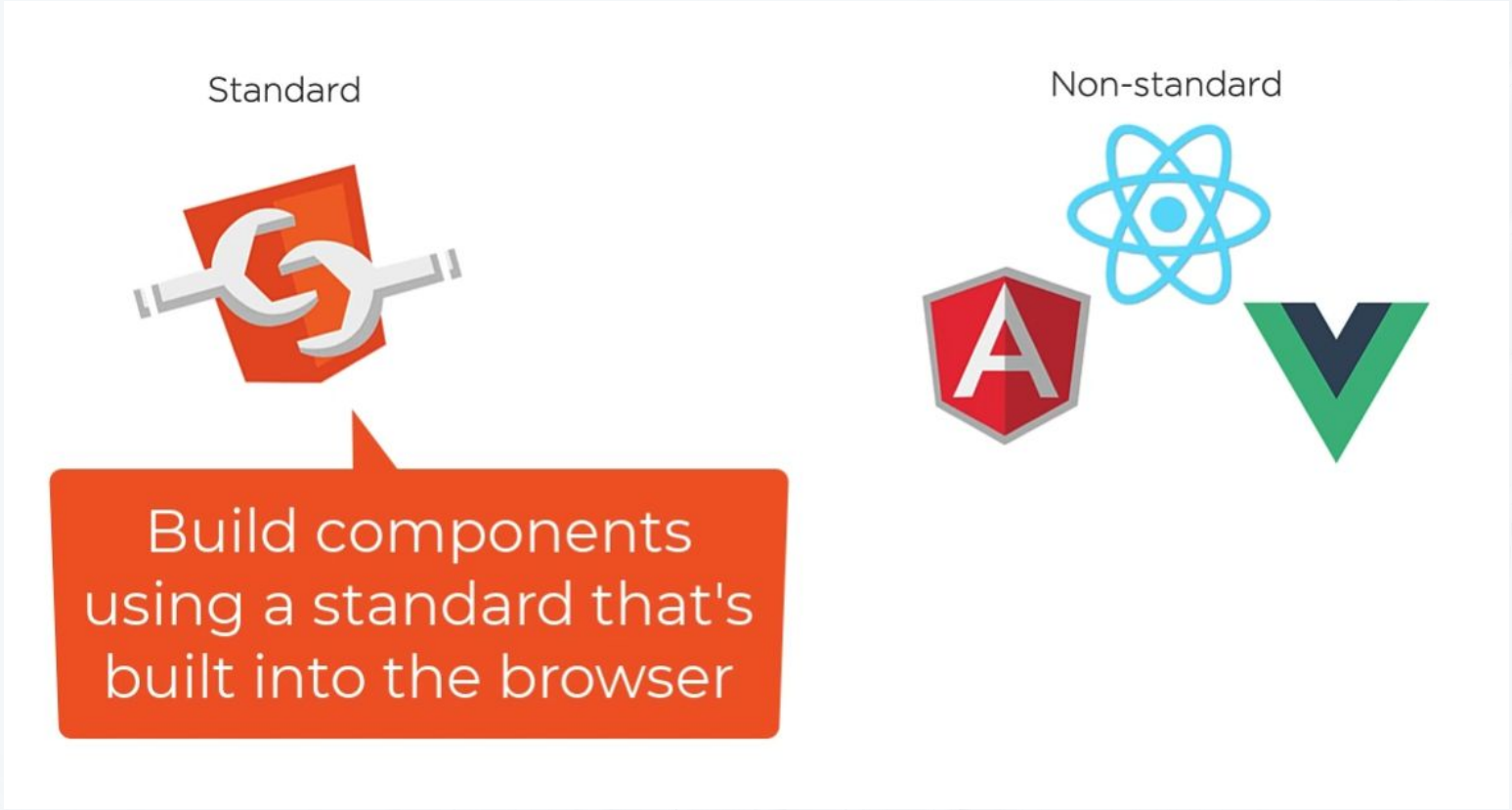
Source: speakerdeck.com/didoo/let-there-be-peace-on-css

Separate Template vs Single File

- In React you can then combine the small components into bigger ones
- You have the option to have separate CSS for each component if you want to instead of everything in one big file which is the same across the application
- Key is to create big components which are created from a variety of small nested components which can then be tested separately in isolation

Standard vs Non-standard

- There is a defined standard of what web component should look like that JS frameworks do not follow



Standard vs Non-standard

- Standardized web components are defined to include:

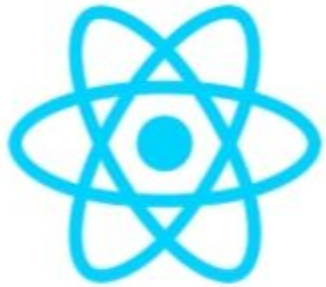
Inert, reusable markup

Define our own elements

Encapsulated styling

Standard vs Non-standard

- However, developers continue to prefer to use JS frameworks which have their own take on everything



Web Components	React
Templates	JSX, JS
Custom Elements	Declare React components
Shadow DOM	CSS modules, CSS in JS, “inline”

Standard vs Non-standard

- Standardized web components do not enable anything new
- JS libraries and frameworks have nicer APIs and they keep on innovating
- Standard web components run only in the browser while React components can work in mobile too and can be rendered on the server as well
- The standard might take off at some point but currently everyone seems to prefer JS frameworks

Community vs Corporate Backing

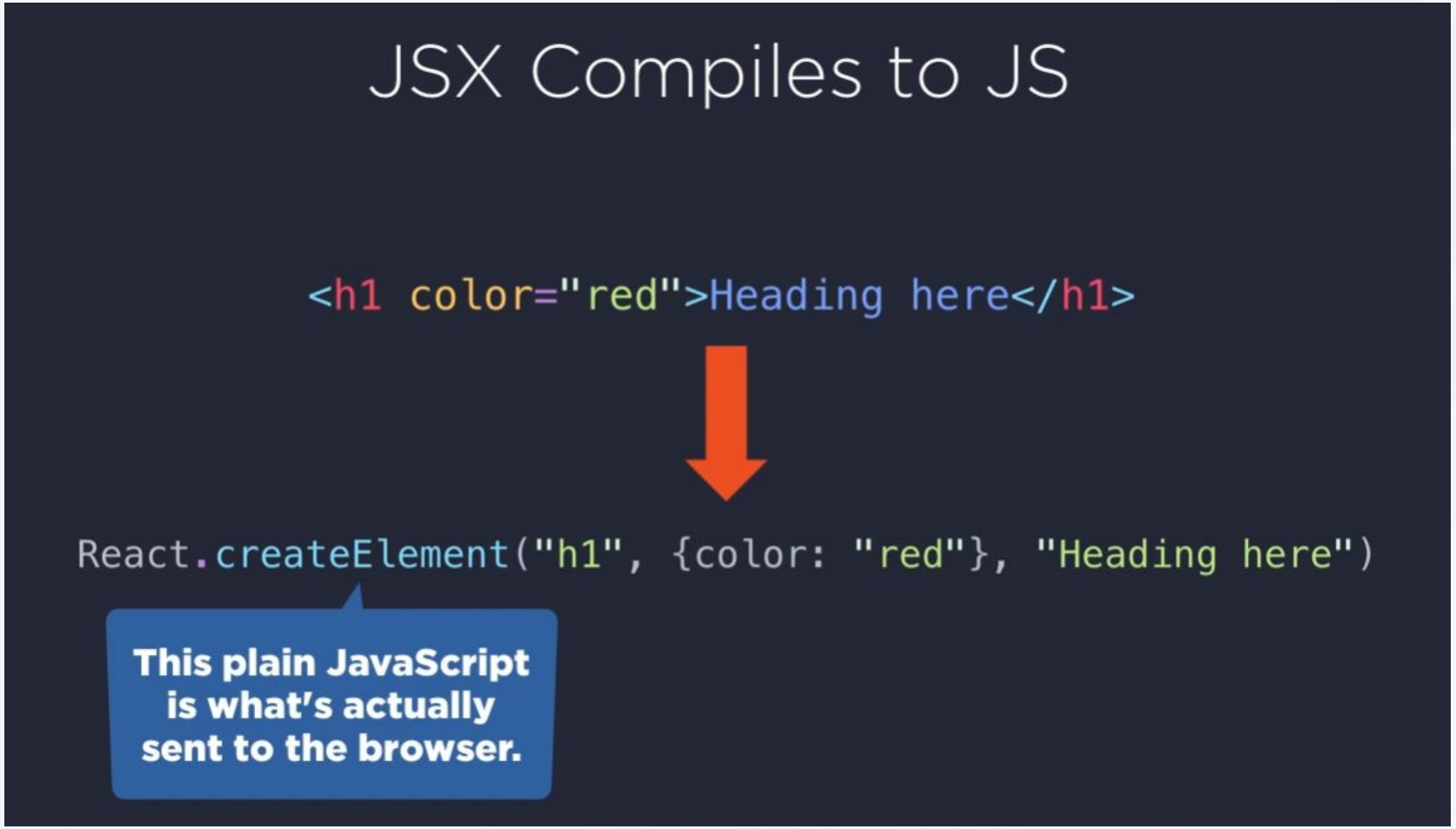
- Being corporate backed is a concern but in this case even if FB goes away the community would definitely continue the project on its own (and FB is deeply involved so that's unlikely)

Concerns: Why not React?

- JSX differs from HTML
- Requires additional build step
- Potential version conflicts
- Old features in searches
- Decision fatigue

Concerns: Why not React?

- We write JSX which is then sent to the browser as JS



Concerns: Why not React?

- You can use the React JS syntax but JSX feels much more natural and easier to use instead
- There are just a few differences between HTML and JSX

HTML	JSX
for	htmlFor
class	className
<style color="blue">	<style={{color: 'blue'}}>
<!-- Comment -->	{*/ Comment /*}

Concerns: Why not React?

- What if you have a lot of HTML that you need to convert?

Convert HTML to JSX

Ctrl+F

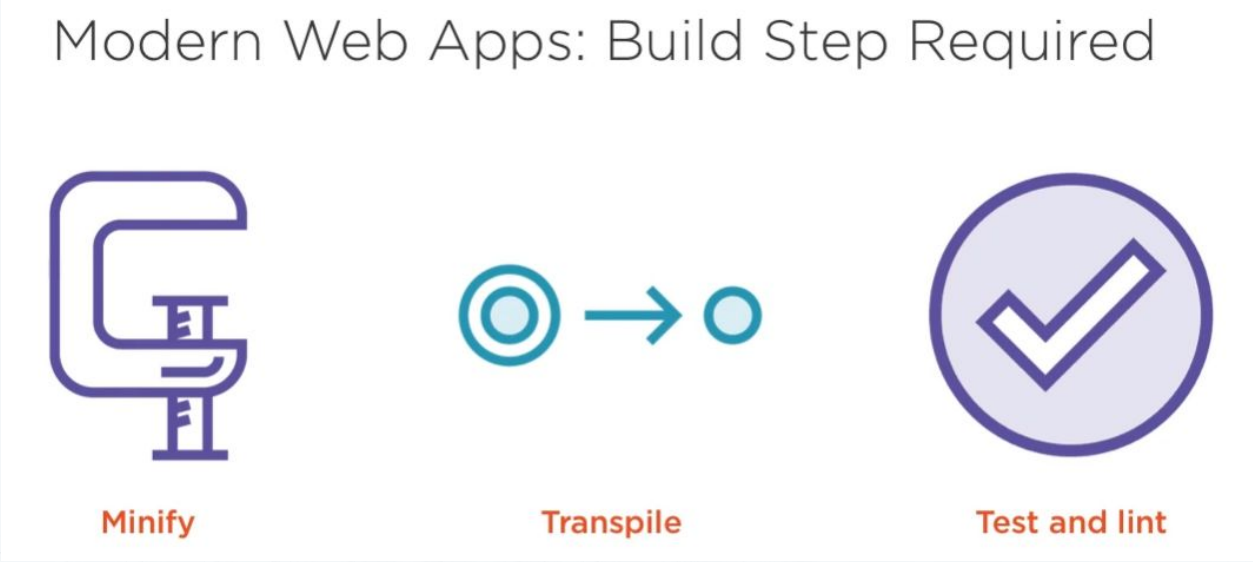
find/replace

Online compiler

htmltojsx on npm

Concerns: Why not React?

- React requires an extra build step to compile the JSX code
- However, most modern web apps and JS frameworks have multiple steps to get the code up and running so the JSX compilation is just another step in that process



Concerns: Why not React?

- Besides, the two most popular transpilers work seemingly and effortlessly with JSX:
  Babel and TypeScirpt
- One command is going to build everything for you and compile the JSX in the process

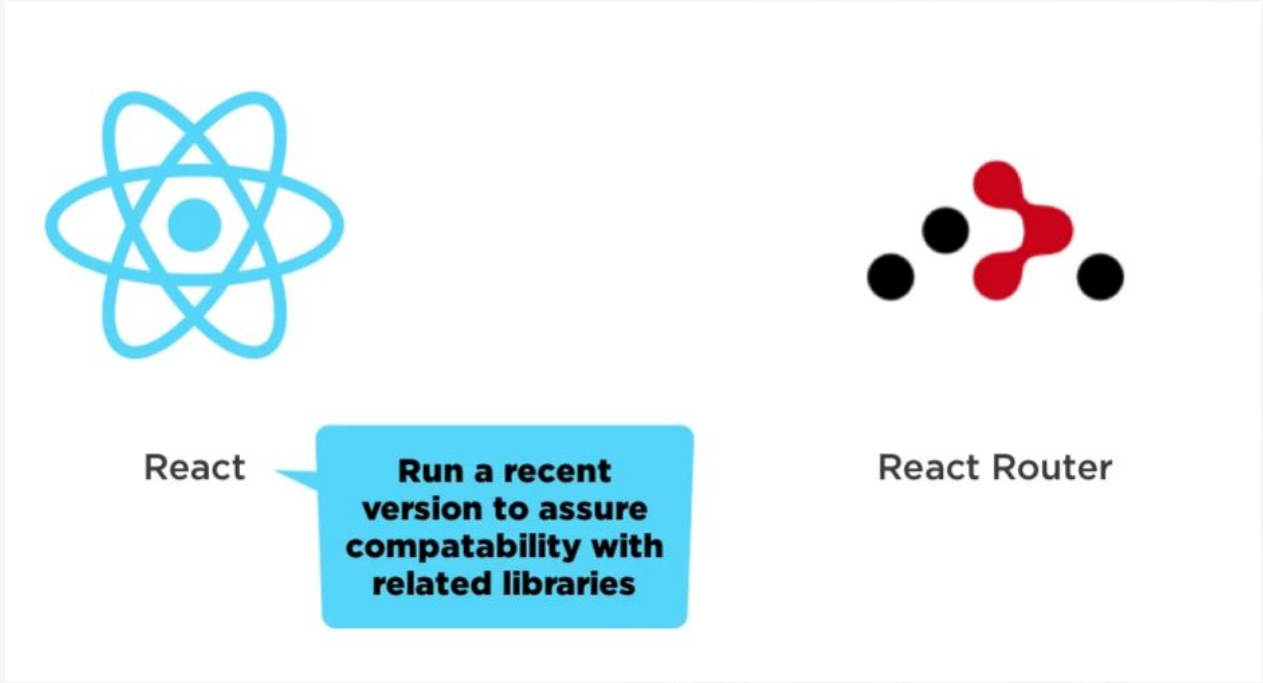
Concerns: Why not React?

- What about versioning conflicts?

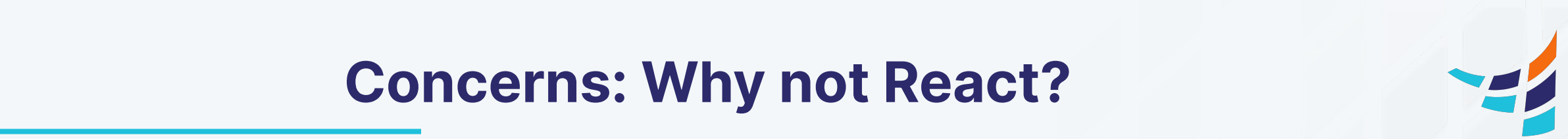
- React has a runtime which means that you can’t run different versions of React at the same time

Concerns: Why not React?

- You are also going to use related libraries and you need to ensure that your React version is compatible with them



Concerns: Why not React?



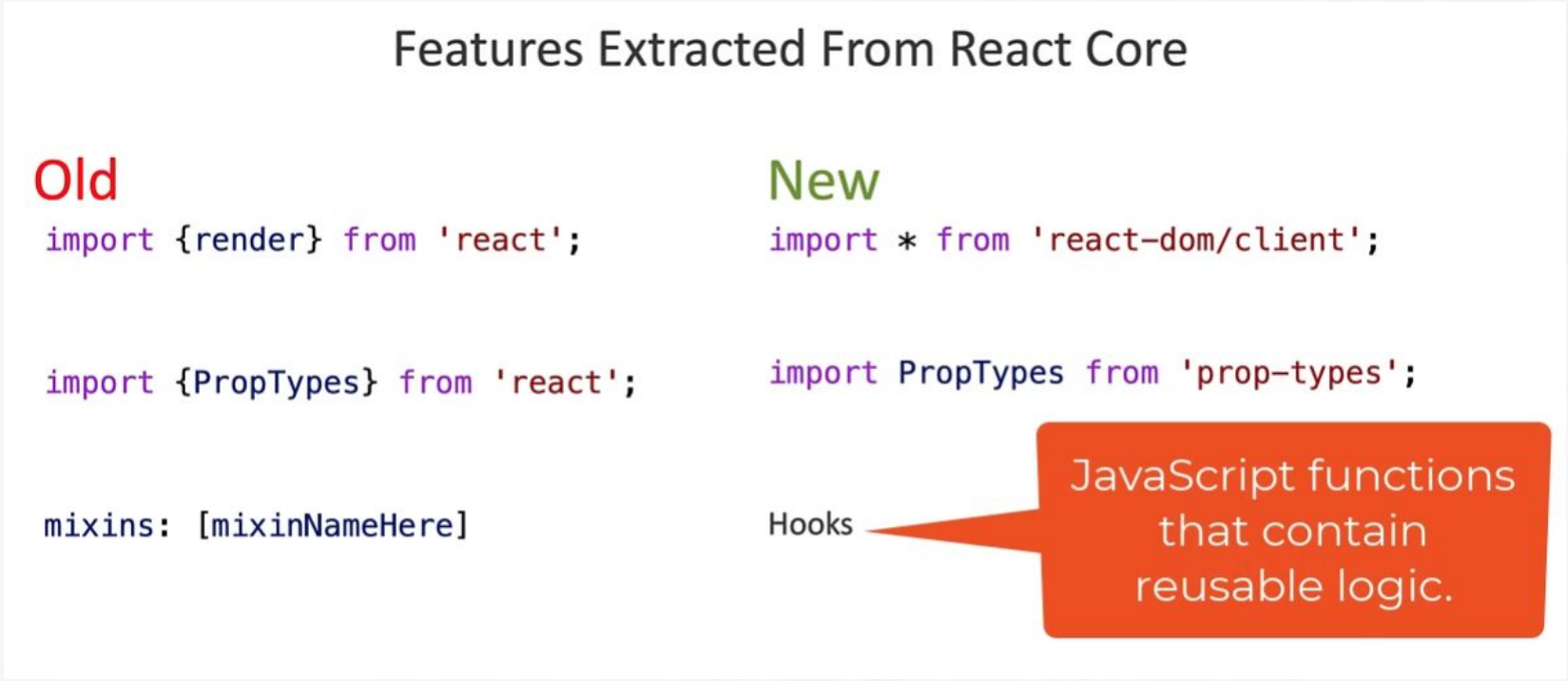
- Using FB codemods to update frequently as well as some internal discipline is required

Concerns: Why not React?

- What about old information online?
- There are more than 324 million results in Google if you just type “react”
- Having a lot of resources is great but there’s also a risk as some of this public content is outdated
- React today is quite different from the one when it released

Concerns: Why not React?

- In order to be efficient, we have to keep up with changes
- React's documentation is pretty good and well maintained



Concerns: Why not React?

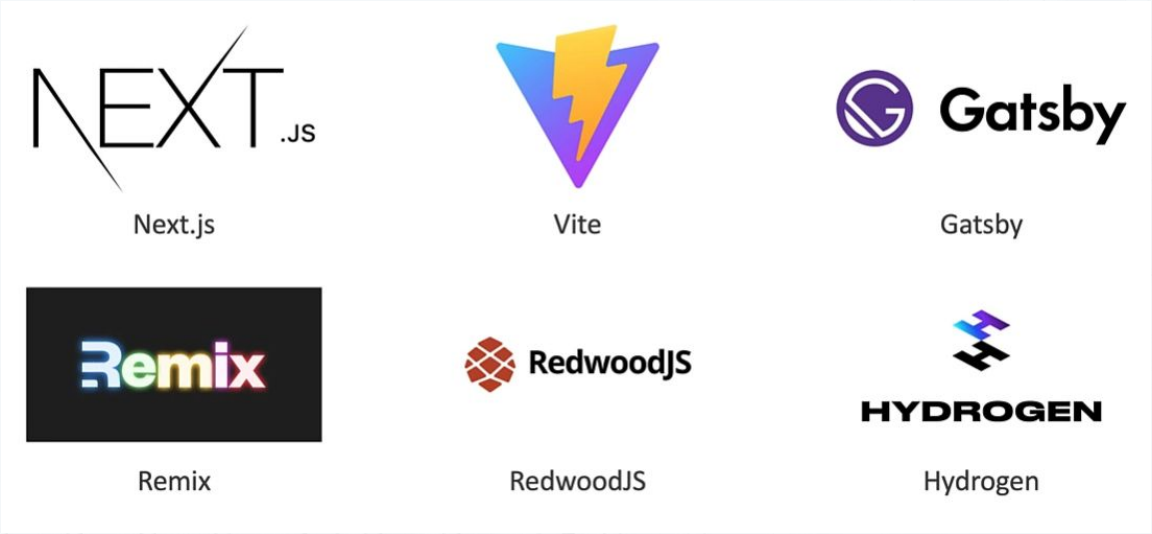
- Being a library gives you a lot of options and different ways to solve problems
- Is that a glass half empty or a glass half full for you?
- Generally having many options is a great problem to have as long as it does not get intimidating

Concerns: Why not React?

- Major decision points could include:
  - Dev environment
  - Using Classes or Functions
  - Handling Types
  - Handling State
  - Handling Styling

Concerns: Why not React?

- Dev environment - Create React App is a great place to start as it provides hot reloading, automated testing, transpiling, bundling, linting and automated build
- You can also check other options:



Concerns: Why not React?

- Decide whether you want your components to be

  Classes or Functions - both work and achieve the same goals and it comes to personal or project preference

- Most people tend to prefer functions

Concerns: Why not React?

- Different ways to handle types:

PropTypes

```
import React from "react";
import PropTypes from 'prop-types';

function Greeting(props) {
  return (
    <h1>Hello {props.name}</h1>
  )
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

Types are only check during runtime

TypeScript

```
import * as React from "react";

interface Props {
  name: string
}

function Greeting(props : Props) {
  return (
    <h1>Hello {props.name}</h1>
  )
}
```

Types are checked at compile

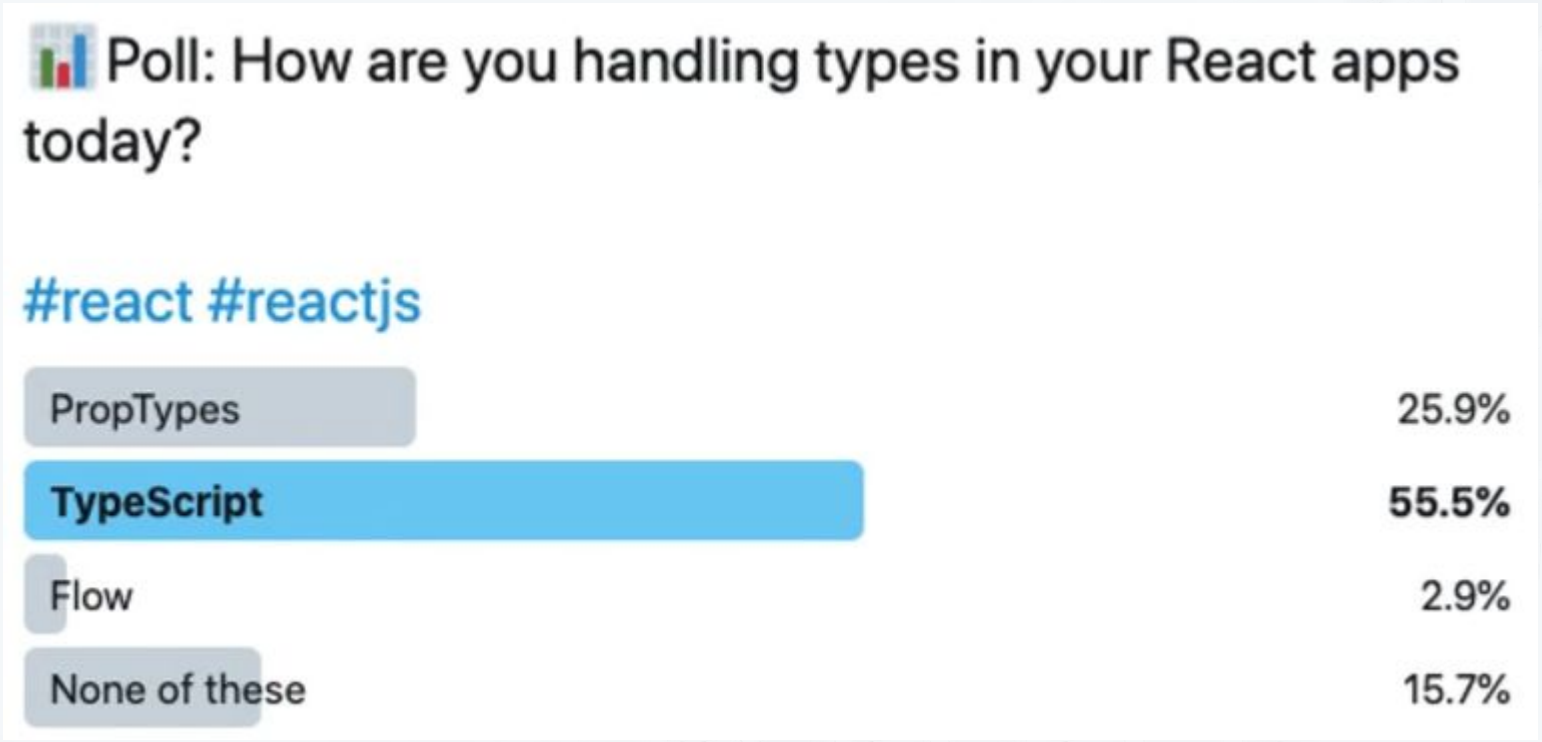
Flow

```
// @flow
import React from "react";

type Props = {
    name: string
};

function Greeting(props: Props) {
  return (
    <h1>Hello {props.name}</h1>
  )
}
```

Created to infer types in existing plain JavaScript Code

Concerns: Why not React?

- PropTypes is great for beginners but React has support for TypeScript as well which seems to be prefered



Concerns: Why not React?

- What about your state management - your app's data?

Flux / Redux

Mobx

React query

React handles state great all by itself.

Concerns: Why not React?

- You can start with plain React and get Redux later

Flux / Redux

Mobx

React query

React handles state great all by itself.

Concerns: Why not React?

- What about styling? - use what you already know
- Plain CSS with or without Sass and Less
- CSS-in-JS
- CSS Modules and libraries

Questions?

Thank You!