

# Front-End Development with React

Mail: [jonas.band@ivorycode.com](mailto:jonas.band@ivorycode.com)

Twitter: [@jbandi](https://twitter.com/jbandi)

# ABOUT ME

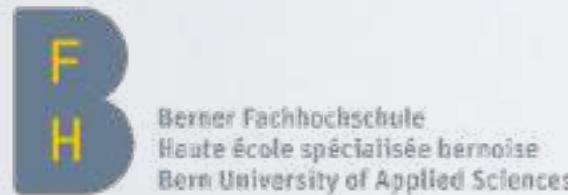
Jonas Bandi

[jonas.bandi@ivorycode.com](mailto:jonas.bandi@ivorycode.com)

Twitter: [@jbandi](https://twitter.com/@jbandi)



- Freelancer, in den letzten 10 Jahren vor allem in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse & Beratungen zu Web-Technologien im Enterprise: UBS, Postfinance, Mobiliar, AXA, BIT, SBB, Elca, Adnovum, BSI ...



JavaScript / Angular / React / Vue / Vaadin  
Schulung / Beratung / Coaching / Reviews

[jonas.bandi@ivorycode.com](mailto:jonas.bandi@ivorycode.com)



About you ...

# AGENDA

DAY 1



**Intro**

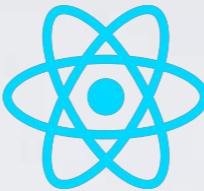
**Getting started quickly  
with create-vite**

**JSX, React Components  
and Hooks**

**Component  
Architecture & Data**

**Routing**

DAY 2



**Side Track: Async JavaScript & Ajax**

**Backend Access**

**In-Depth Options:**

**More Hooks & Custom Hooks**

**State Management Concepts**

**Ecosystem Picks:  
Tanstack-Query  
nuqs**

**Performance Topics**

**Fullstack React**

**"SIDE-TRACK"**



**Modern JavaScript  
Development Tooling**



**Modern JavaScript  
(es2015 - es2024)**



**TypeScript**

# Material

Git Repository:

<https://github.com/ivorycode/react-roche-2025>

Initial Clone:

`git clone https://github.com/ivorycode/react-roche-2025.git`

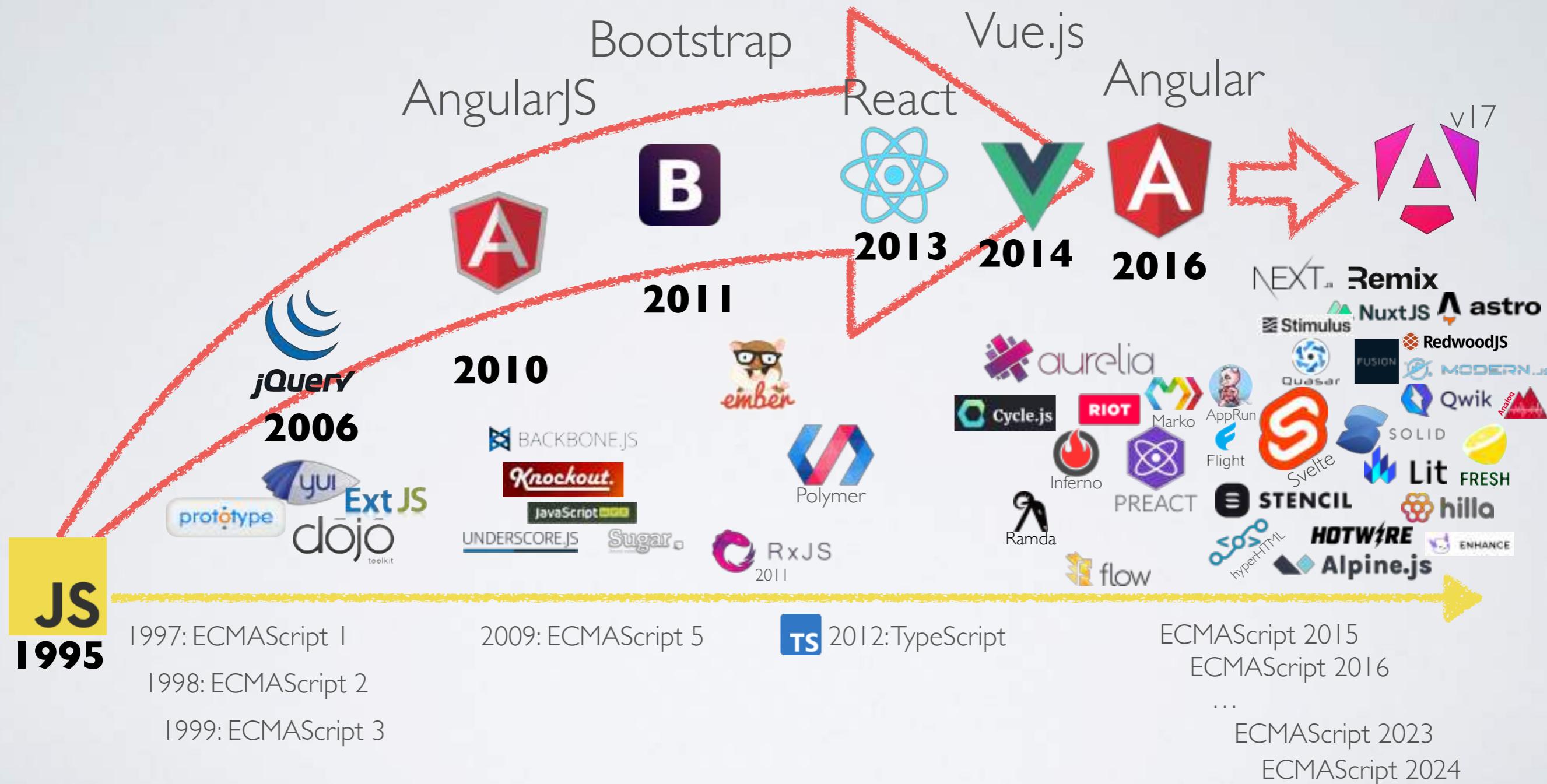
Update: `git pull`

`git reset --hard  
git clean -dfx  
git pull`

(discard all local changes)

Slides & Exercises: <checkout>/00-CourseMaterial

# The Frontend JavaScript Ecosystem



You Retweeted



R. Alex Anderson   
@ralex1993

Want to feel old? ReactJS was released May 29, 2013, or 2468 days ago.

JQuery was released August 26, 2006, or 2468 days before React was released.

That's right. React has been around as long as JQuery was when React came out. 

#reactjs #javascript

6:16 PM · Mar 1, 2020 · Twitter for iPhone

<https://twitter.com/ralex1993/status/1234165541805281280>

# React is very popular



**NETFLIX**

 **Dropbox**



**BBC**

 **Grafana**



**amazon**

 **zalando**

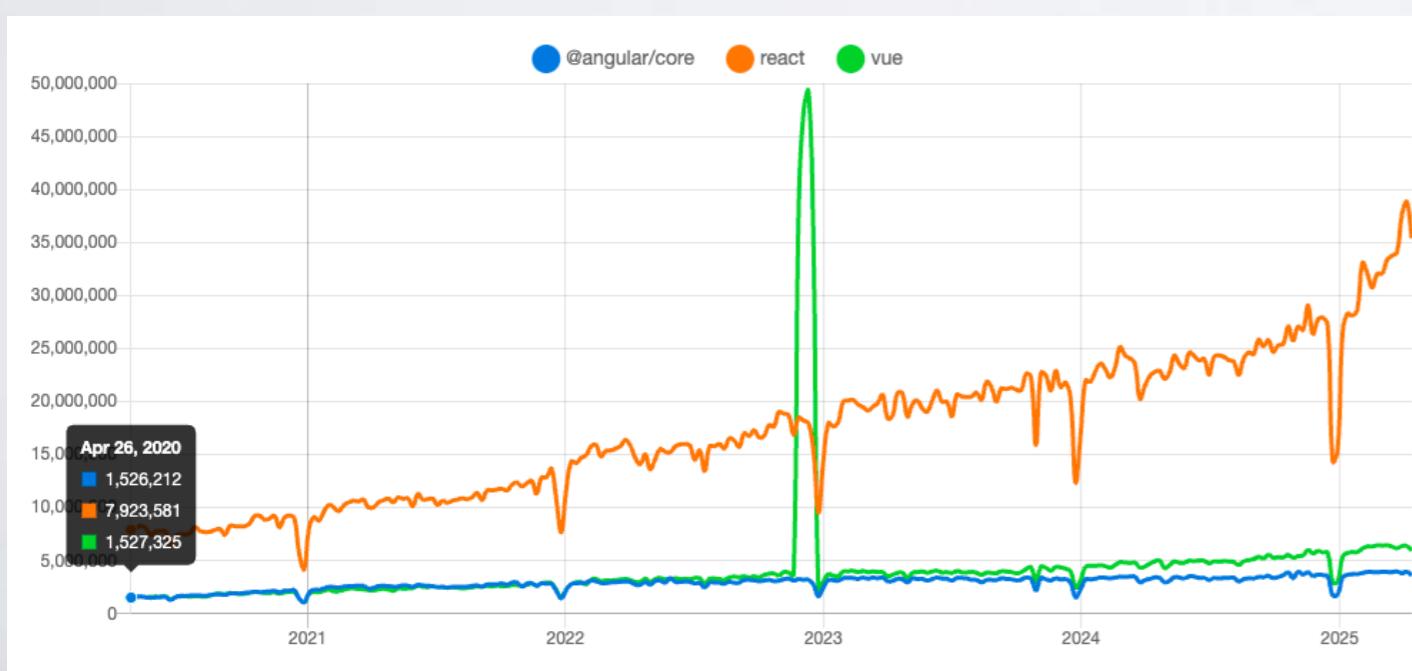
 **JIRA**

 **Outlook.com**

# Popularity: Angular vs. React vs. Vue

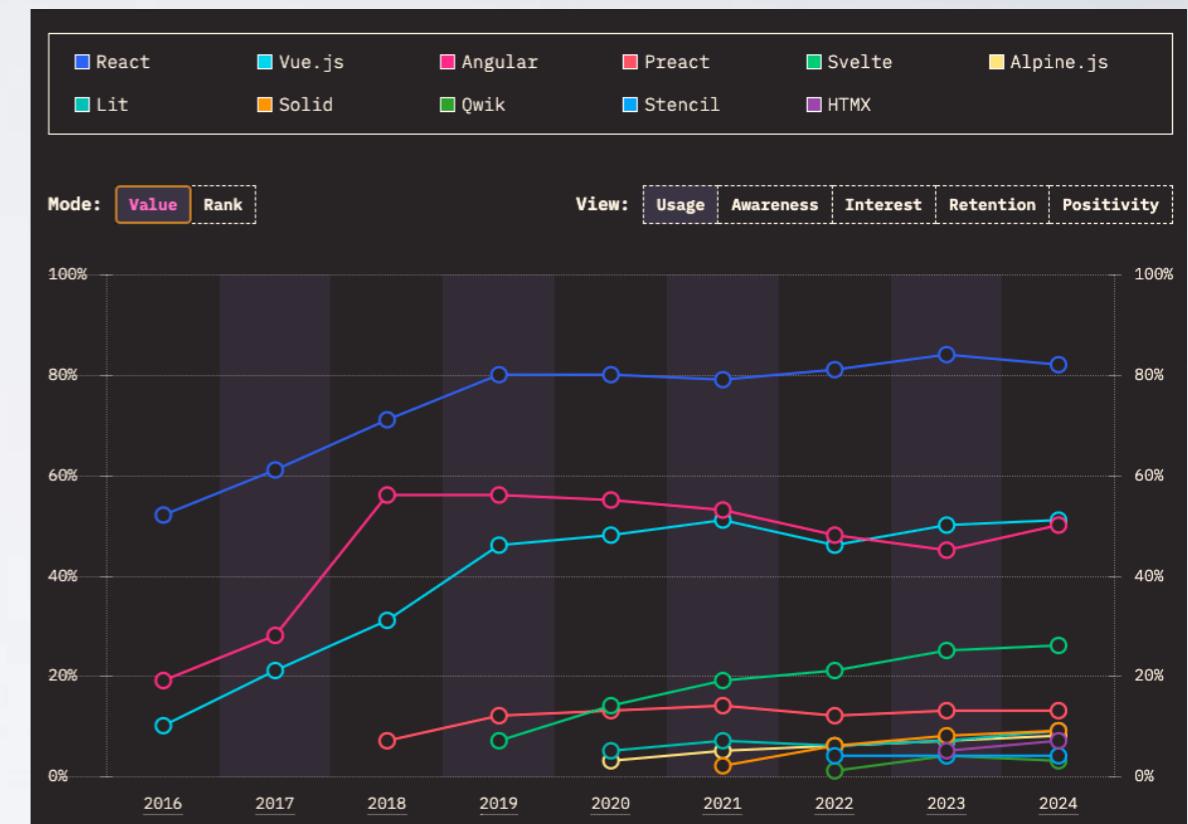


<https://survey.stackoverflow.co/2024/technology#l-web-frameworks-and-technologies>



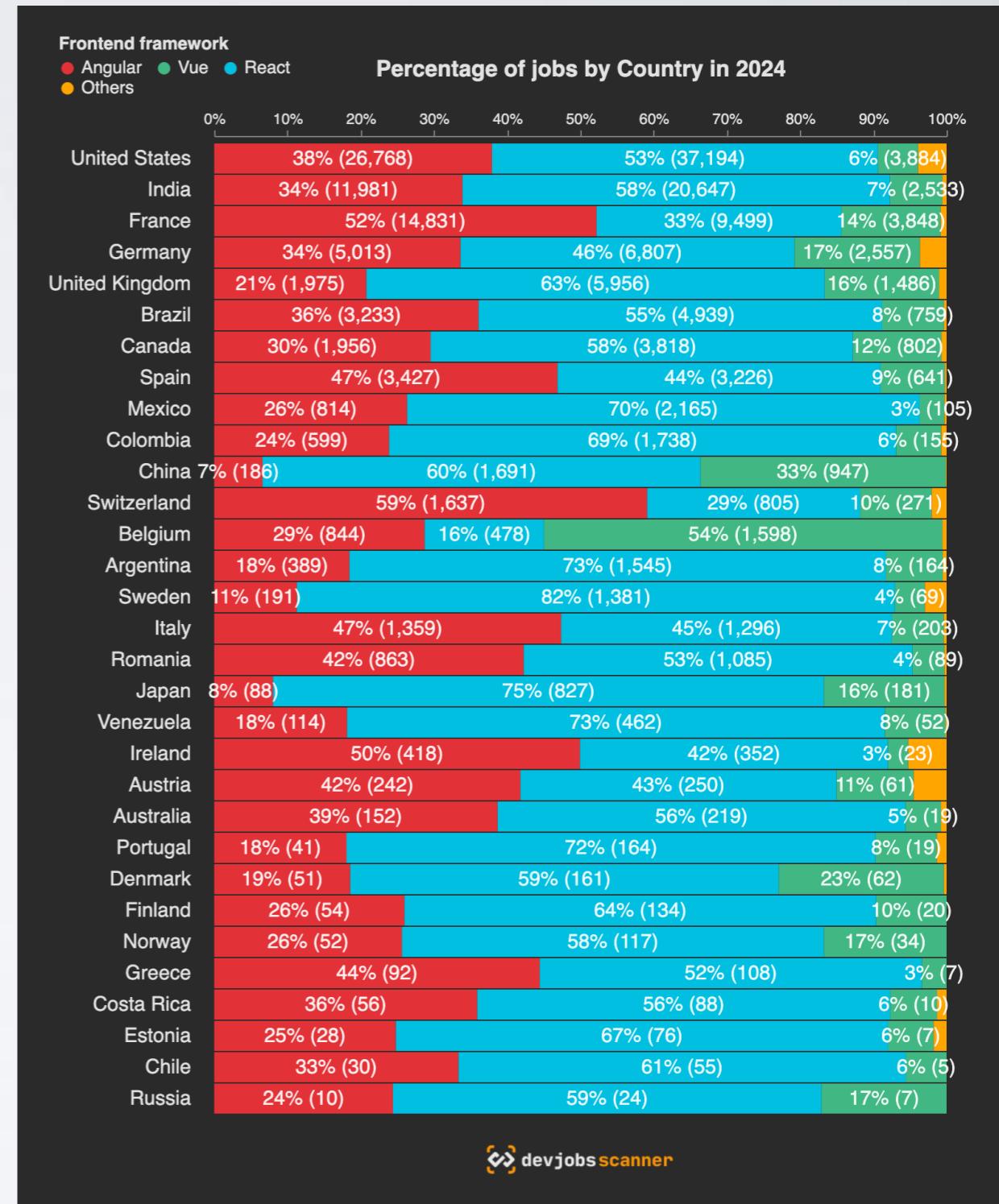
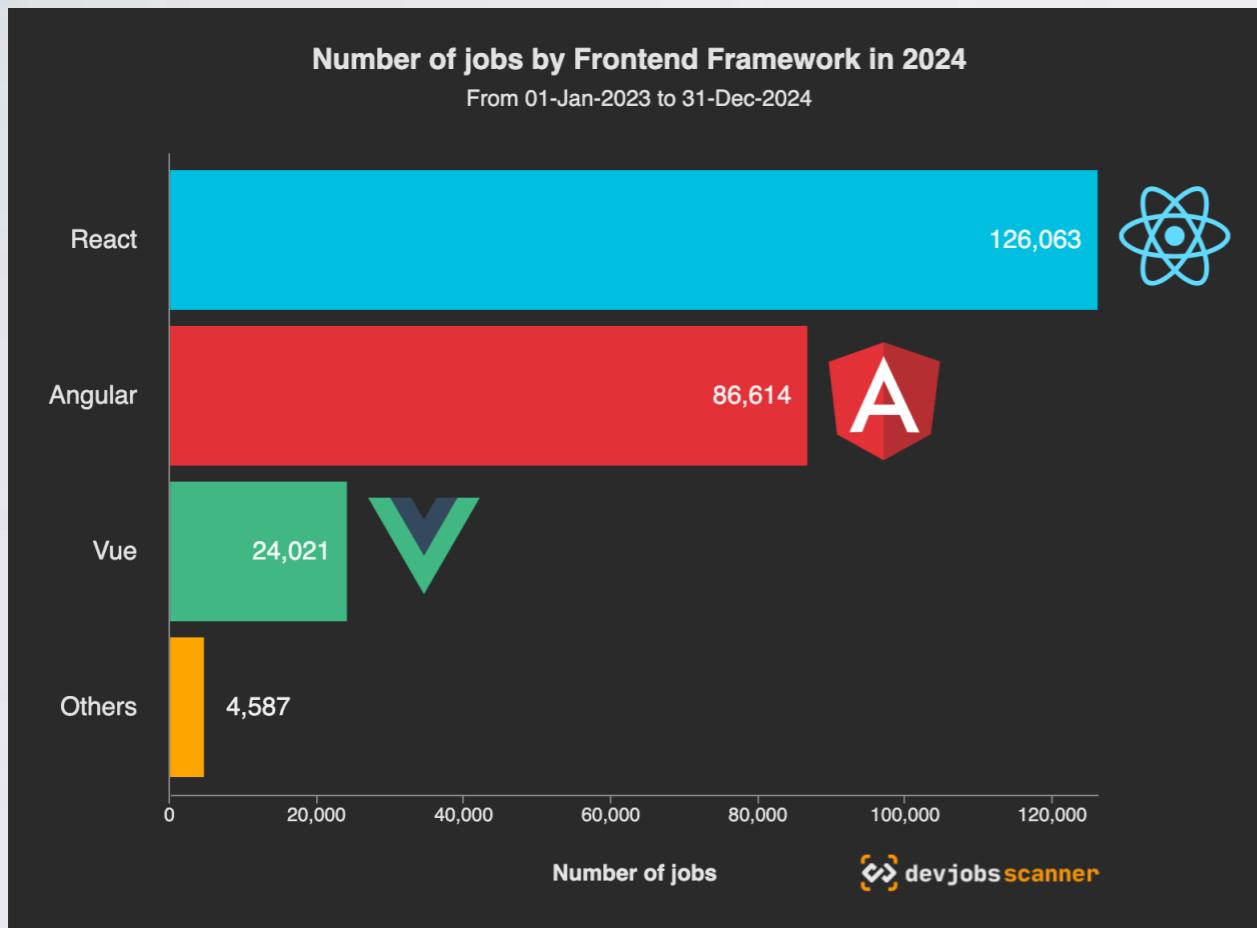
<https://nptrends.com/@angular/core-vs-react-vs-vue>

## State of JavaScript Survey 2022

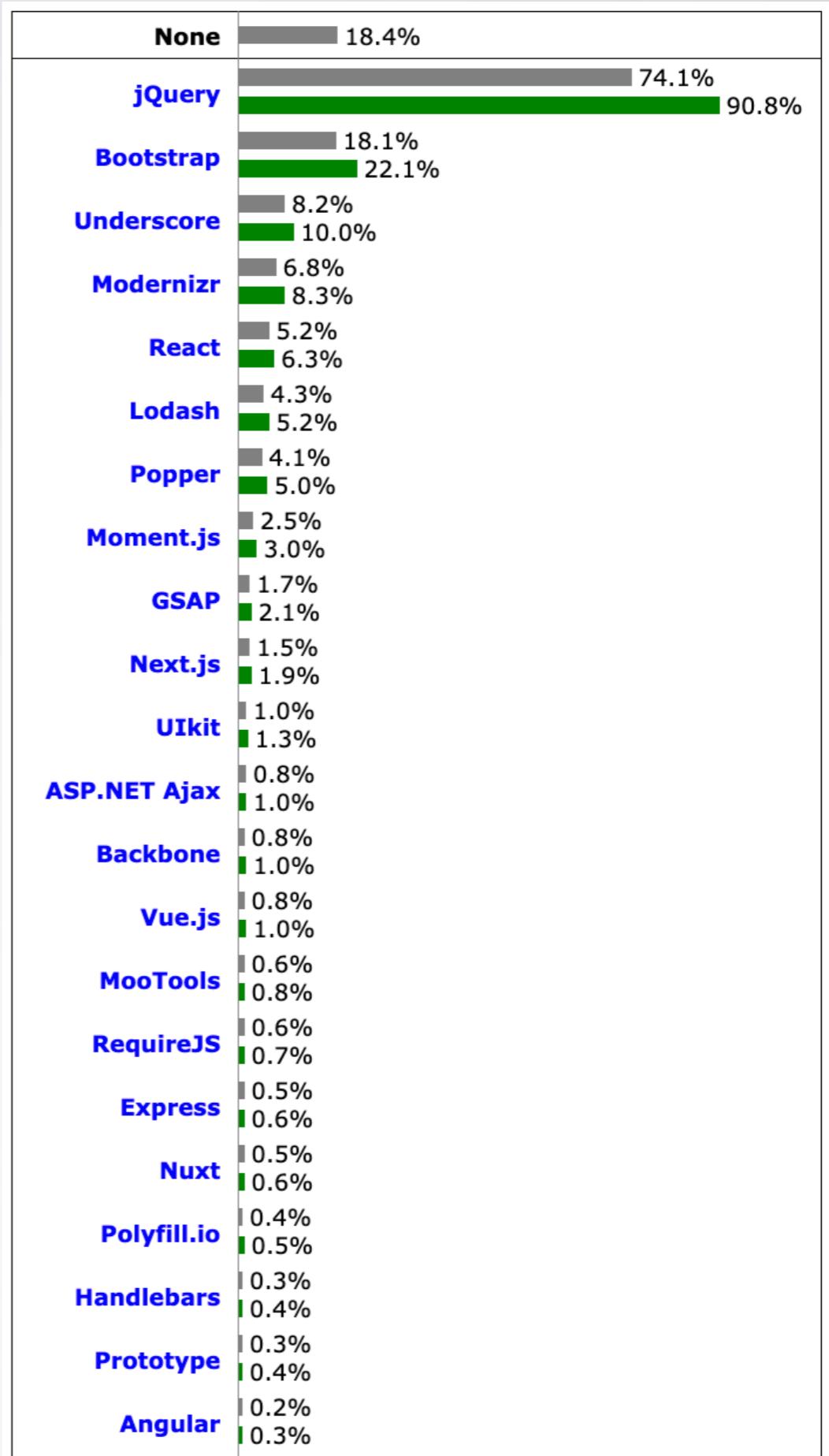


<https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/>

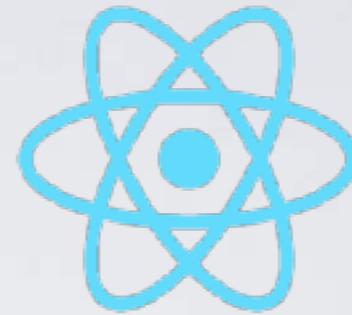
# Popularity: Angular vs. React vs. Vue



By the way ...  
...the web still  
runs on jQuery



# React



React is a JavaScript library for building user interfaces.

Created by Facebook in 2013.

Current Version: 19 (finally released in December 2024 )

## *Core Principles:*

The DOM is created programmatically.

JSX enables declarative DOM programming.

A virtual DOM abstracts the real DOM.

One-way data flow instead of two-way data binding.

<https://react.dev/>

<https://github.com/facebook/react/>

<https://react.dev/blog/2024/12/05/react-19>

<https://react.dev/versions>

# Single Page Applications

"Rich Clients running in the Browser"

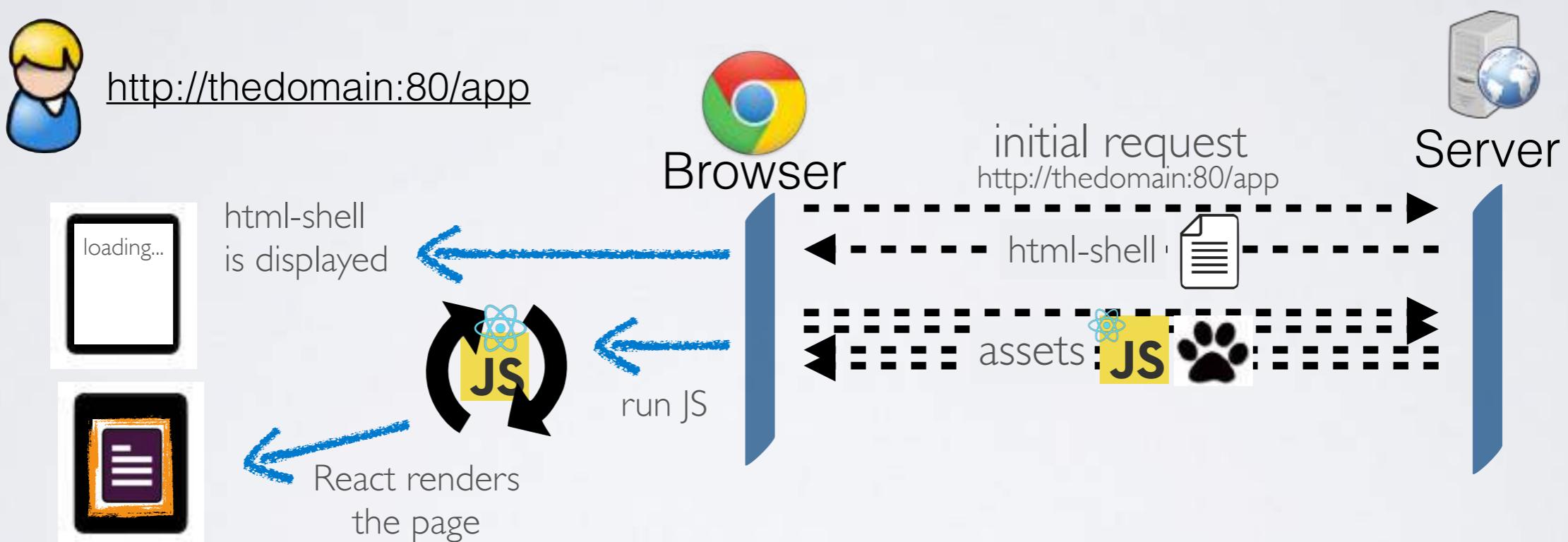


# EXERCISE



Exercise - Three Apps

# Traditional SPA: Client Side Rendering

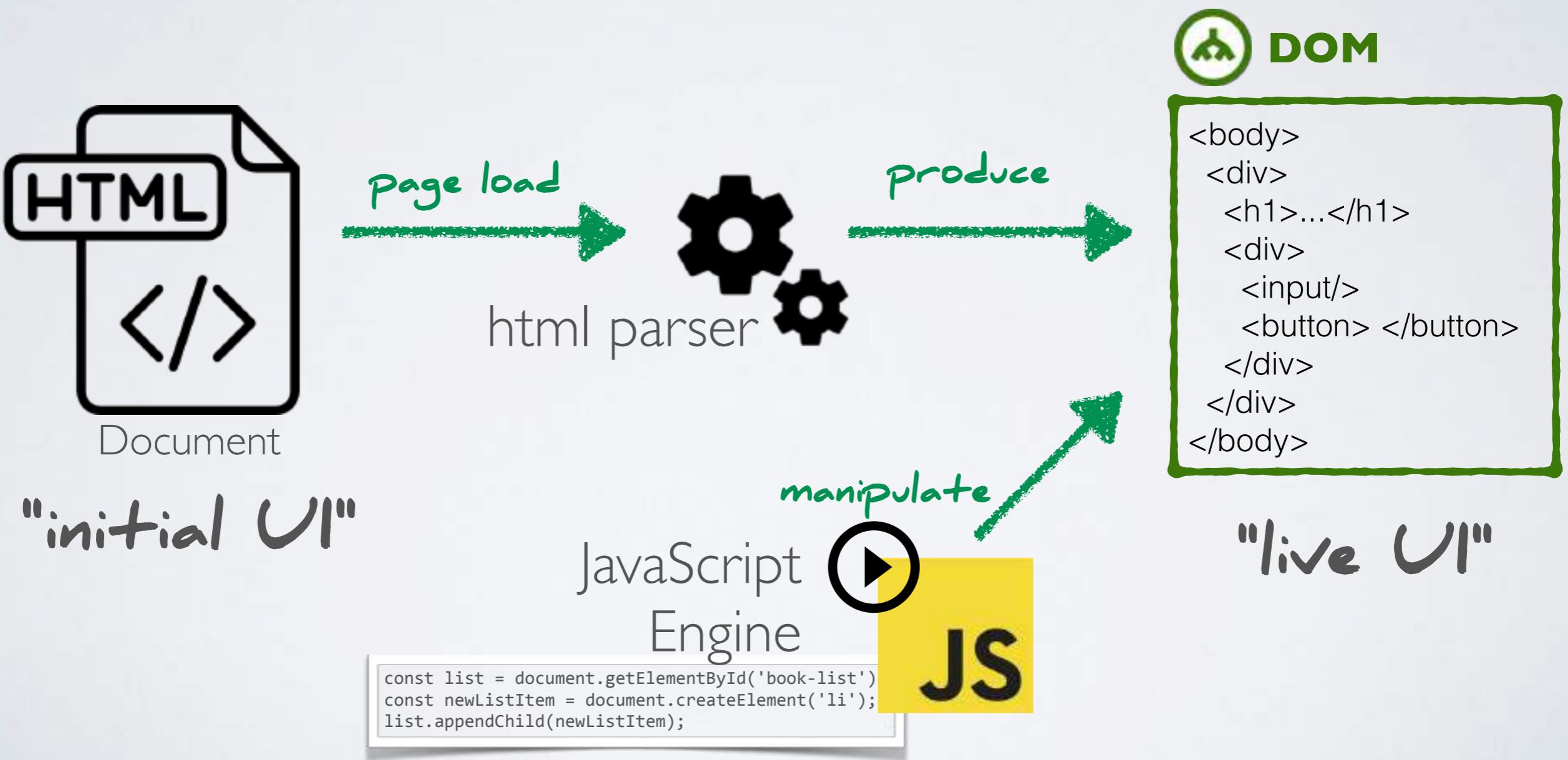


Components are rendered on the client.

# The Browser DOM



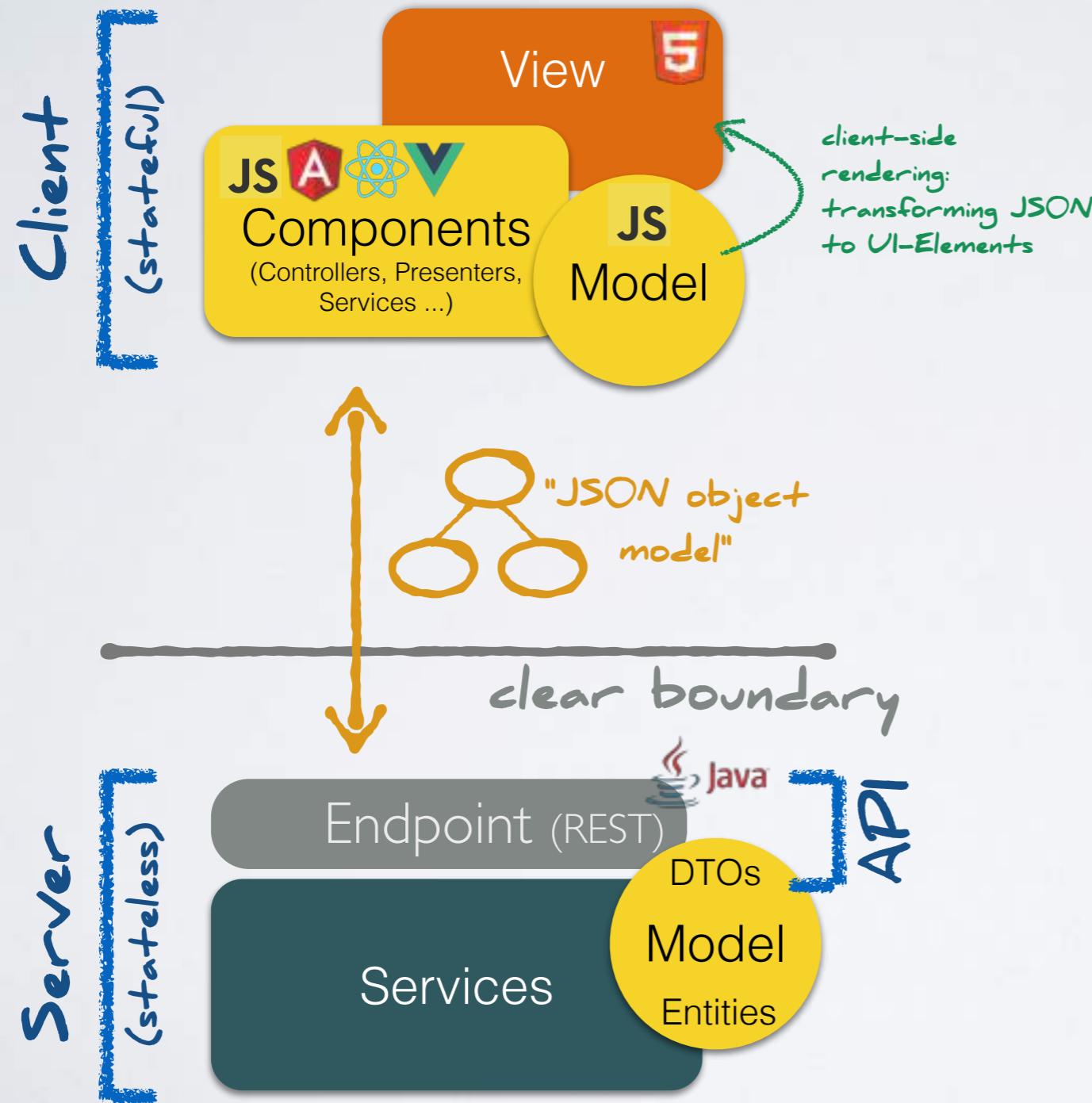
The "UI Engine" of SPAs





# Single Page Applications

# SPA Architecture



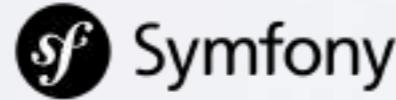
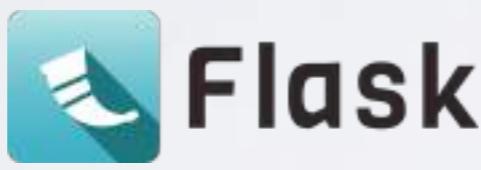
Rich client programming model in the browser.

Clear separation of concern between client and server.

MPA  
Multi-Page-Applications



SPA  
Single-Page-Applications

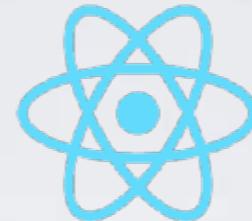


MPAs are rendered on the server and optionally "enhanced" on the client with JavaScript.



- Traditional SPAs are rendered purely on the client-side (CSR).
- Modern SPA frameworks enable initial server-side rendering (SSR).
- Some modern frontend frameworks evolved to full-stack frameworks with server-side logic for ie. data-fetching & data-mutations (Next.js, Remix / ReactRouter v7, SvelteKit, Nuxt, SolidStart ...)
- React Server Components is a innovative approach to blur the boundary between server and client for frontend applications.

# React Documentation



The "new" React documentation (official since March 2023)

<https://react.dev/>

Guide for using TypeScript with React:

<https://react.dev/learn/typescript>

Other Guides for React with TypeScript:

- Community-maintained cheatsheet for using TypeScript with React:  
<https://react-typescript-cheatsheet.netlify.app/>
- TypeScript Handbook for JSX:  
<https://www.typescriptlang.org/docs/handbook/jsx.html>

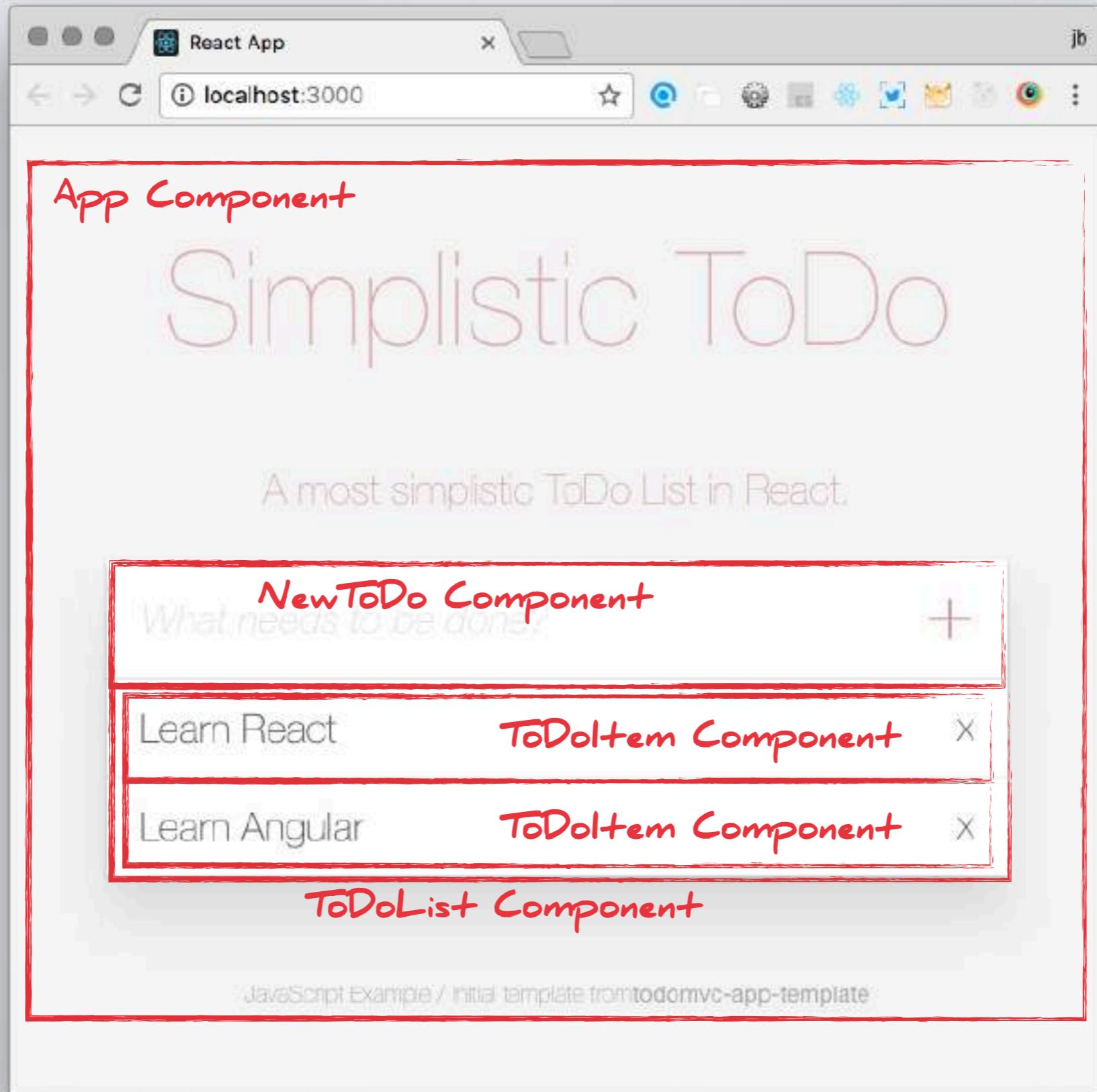
The old and very outdated React documentation:

<https://legacy.reactjs.org/>

# What does React provide?

- A mechanism to create dynamic UIs (DOM manipulations)
  - declarative DOM programming instead of imperative manipulations
- “Componentization” of the UI
  - Provide Structure for the UI
  - Enable reuse of components
  - Manage state in components
- Constructs to create a "data-flow" through the component tree

# Everything is a Component



# React is not a Framework

- React is "only" a view library (however recently the scope is growing)  
Next.js, ReactRouter v7 (formerly Remix), RedwoodJS and TanStack Start are popular Frameworks based on React.
- React is often combined with other libraries/frameworks to build a complete stack for building a frontend-application
  - Routing
  - Data-Fetching (http, websocket ...)
  - i18n
  - Styling ...
  - State-Management

# Evolution of React

React  
(Virtual Dom)

2013

2015

React Native  
components are based on ES2015 classes

2016

~~Next.js~~

Splitting React and ReactDOM  
React 0.14

Create React App 

2019

React 16.8: Hooks  
("Function Components for everything")



2022

React 18  
Next.js App Router with RSCs

2024

React 19  
"official" release of RSCs



**Cory House** @housecor · Oct 30

...

React in 2014:

- Make SPAs.
- Use create-react-app.

React in 2022:

- Make SPAs, native mobile, static, MPAs, and hybrid apps.
- Use Vite, Next, Remix, Astro, etc.
- HTTP via react-query, swr, Apollo, etc.
- Routes via React Router, Remix, Next, Tanstack Router, etc...

38

119

1,131

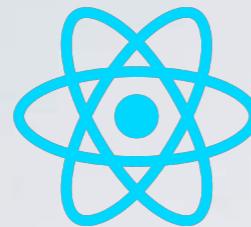


**Cory House** @housecor · Oct 30

...

React 2014: Simple, but limited.

React 2022: Powerful and remarkably flexible, but intimidating and fragmented. A massive number of compelling options to consider. The hardest part is keeping up with the churn and choosing between multiple compelling options.



# React Native

*Learn once, write anywhere.*

Written in JavaScript - rendered with native code.

<https://reactnative.dev/>



For Android + iOS by facebook.

React Native for Windows + macOS by Microsoft

- <https://microsoft.github.io/react-native-windows/>



# History: Fundamental Change in 2019

Ryan Florence (@ryanflorence) posted a tweet on May 20, 2019, at 3:36 AM via Twitter Web App. The tweet reads: "It's a weird time to learn React. It was fundamentally unchanged for 6 years, it's gonna stay a little bumpy this year, and then it'll be smooth sailing for another 6 or more after that". The tweet has 4 retweets and 53 likes. The URL of the tweet is <https://twitter.com/ryanflorence/status/1130286263926284289>.

*React 16.8 (February 2019) introduced "Hooks": a way to use state and other React features without writing a class. This resulted in major changes in the React ecosystem.*

# React Hooks

introduced with React 16.8 in 2019.

<https://react.dev/reference/react/hooks>

Class Component:

```
class Counter1 extends React.Component {  
  
  state = {  
    count: 0  
  };  
  
  increase = () => {  
    this.setState({count: this.state.count + 1})  
};  
  
  render() {  
    return (  
      <div>  
        <h1>Count {this.state.count}</h1>  
        <button onClick={this.increase}>  
          Increase  
        </button>  
      </div>  
    );  
  }  
}
```

using ES2015 classes  
using JSX  
using proposed class fields

method derived from base class

Function Component with Hook:

```
function Counter2() {  
  
  const [count, setCount] = useState(0);  
  
  function increase() {  
    setCount(count + 1);  
  }  
  
  return (  
    <div>  
      <h1>Count {count}</h1>  
      <button onClick={increase}>Increase</button>  
    </div>  
  );  
}
```

using JavaScript functions  
using JSX

Hook

the "new" way

the "old" way

Note : functional components and class components can be mixed in the same application.

# Why Hooks (2019)?

<https://reactjs.org/docs/hooks-intro.html#motivation>

"Hooks solve a wide variety of seemingly unconnected problems of class components."

Hooks reduce complexity in components and enable better patterns for decoupling application logic and ui-logic.

Hooks enable better structure, better reuse and better composition of application logic.

Hooks are often an elegant replacement for previous reusability patterns like "higher order components" and "render props"

Hooks are not undisputed:

- <https://stevenkitterman.com/posts/the-catch-with-react-hooks/>
- <https://blog.logrocket.com/frustrations-with-react-hooks/>
- <https://dillonshook.com/a-critique-of-react-hooks/>
- <https://medium.com/swlh/the-ugly-side-of-hooks-584f0f8136b6>
- <https://medium.com/better-programming/why-react-hooks-are-the-wrong-abstraction-8a44437747c1>
- <https://danielrotterat/2022/01/16/some-reasons-for-disliking-react-hooks.html>

"Why React Hooks":

[https://www.youtube.com/watch?v=eX\\_L39UvZes](https://www.youtube.com/watch?v=eX_L39UvZes)

# Why Hooks (2023)?

Because the React ecosystem has embraced Hooks.

(and React without it's ecosystem is like a car without seats, wheels, windows, pedals ...)

All modern React libraries expose their APIs via Hooks:

- ReactRouter / TanStack Router
- Jotai, Zustand, MobX, Redux, Recoil ...
- TanStackQuery, SWR
- react-i18next
- ...

# Starting a Client-Side React Project with Vite



**Vite** Next Generation Frontend Tooling

<https://vitejs.dev/>

```
npm create vite@latest my-react-project -- --template react-swc-ts
```

```
cd my-react-project  
npm install  
npm run dev
```

generating project files

installing dependencies

running the app in development mode

optional using  
TypeScript & SWC

Vite is one of the recommended ways to "build a React app from scratch":  
<https://react.dev/learn/build-a-react-app-from-scratch>

# Full-Stack React

(aka. Meta Frameworks)

The official React documentation is recommending to use a "production-grade framework".

<https://react.dev/learn/creating-a-react-app#full-stack-frameworks>

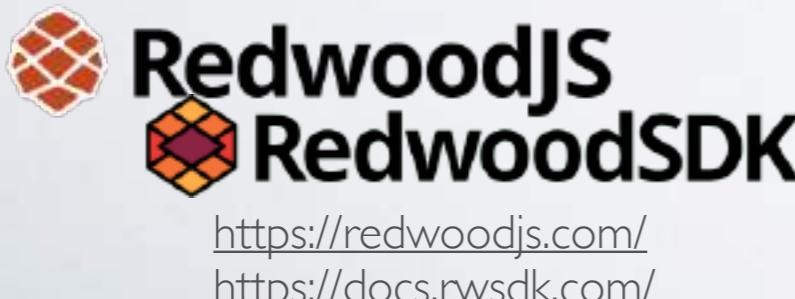


The common goal of those meta-frameworks is to simplify the project setup of frontend applications.

They include concepts for server-side-rendering, routing, data-fetching, mutations ...

But they come with their own conceptual overhead and learning curve!

These frameworks typically require running JavaScript on the server (like Node.js).





**Marc Grabanski**  
@1Marc

Whoa, the React team doesn't recommend using React client-side only  
(as a SPA) anymore:

10:37 PM · Mar 16, 2023 · 385.5K Views

260 Retweets 42 Quotes 1,721 Likes

<https://twitter.com/1Marc/status/1636481900381388802>



**Ryan Carniato**  
@RyanCarniato

And so begins the age of "fully with React":

11:54 PM · Mar 16, 2023 · 107.6K Views

27 Retweets 7 Quotes 325 Likes

<https://twitter.com/RyanCarniato/status/1636501181039276032>

...



**Evan You**  
@youyuxi

Replies to [@dan\\_abramov](#)

I disagree to some extent (particularly the heavy-handed push towards fullstack use cases), but yeah I understand the reasoning and motives.

4:53 AM · Mar 17, 2023 · 10K Views

<https://twitter.com/youyuxi/status/1636576506574176258>

...

...



**\_cloudmu**  
@\_cloudmu

Replies to [@dan\\_abramov](#)

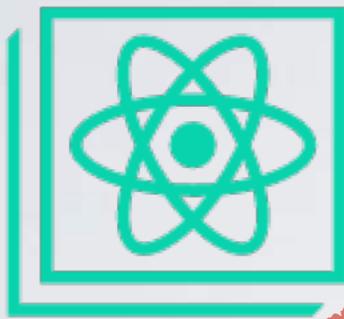
Btw, I am not advocating for CRA per se. Just want to share that there is a large React user base out there (not on Twitter) who has been deploying production systems with a client side front-end. Many of their use cases won't benefit much from server side rendering ...

2:19 AM · Mar 18, 2023 · 485 Views

[https://twitter.com/\\_cloudmu/status/163690018643775488](https://twitter.com/_cloudmu/status/163690018643775488)

...

# Alternatives for Client-Side React Project Setup



## Create-React-App (CRA)

<https://create-react-app.dev/>

CRA was the de-facto starter for a React SPA for a long time. But it is not recommended any more.

<https://react.dev/blog/2025/02/14/sunsetting-create-react-app>

"deprecated"



## create-tsrouter-app

Aiming to be the CRA replacement. Still very early. But might be a good option if you want to use TanStack Router.

<https://github.com/TanStack/create-tsrouter-app/tree/main/cli/create-tsrouter-app>



## PARCEL

The zero configuration build tool.

<https://parceljs.org/>



## Rsbuild

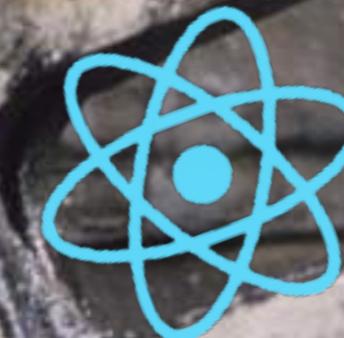
The Rspack Powered Build Tool.

<https://rsbuild.dev/>



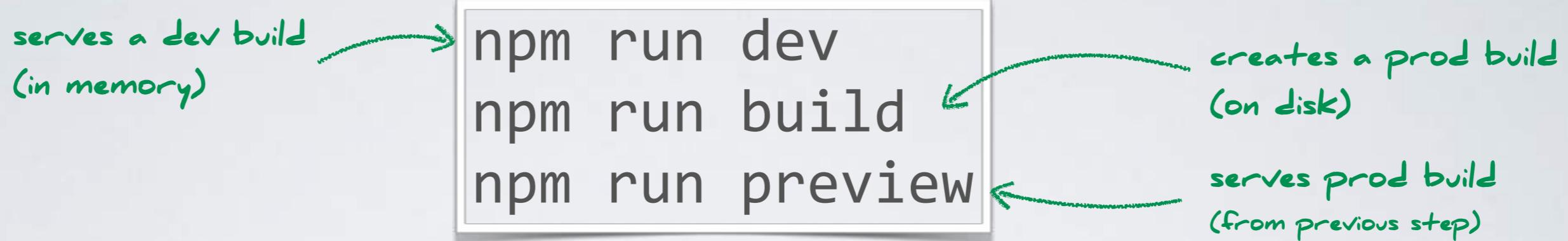
Nx is a scaffolding and build tool for modern frontend projects with the focus on monorepos.  
It can be used as a CRA replacement (using Vite, ESLint, Cypress ...):  
<https://nx.dev/getting-started/react-standalone-tutorial>

# EXERCISES



Exercise 1 - Create a React Project from Scratch

# Understanding a React Project



## Project Configuration:

- package.json
- node\_modules
- package-lock.json
- tsconfig.json

prod build with source maps:

```
export default defineConfig({
  plugins: [react()],
  build: {sourcemap: true }
});
```

<https://vitejs.dev/config/build-options.html>

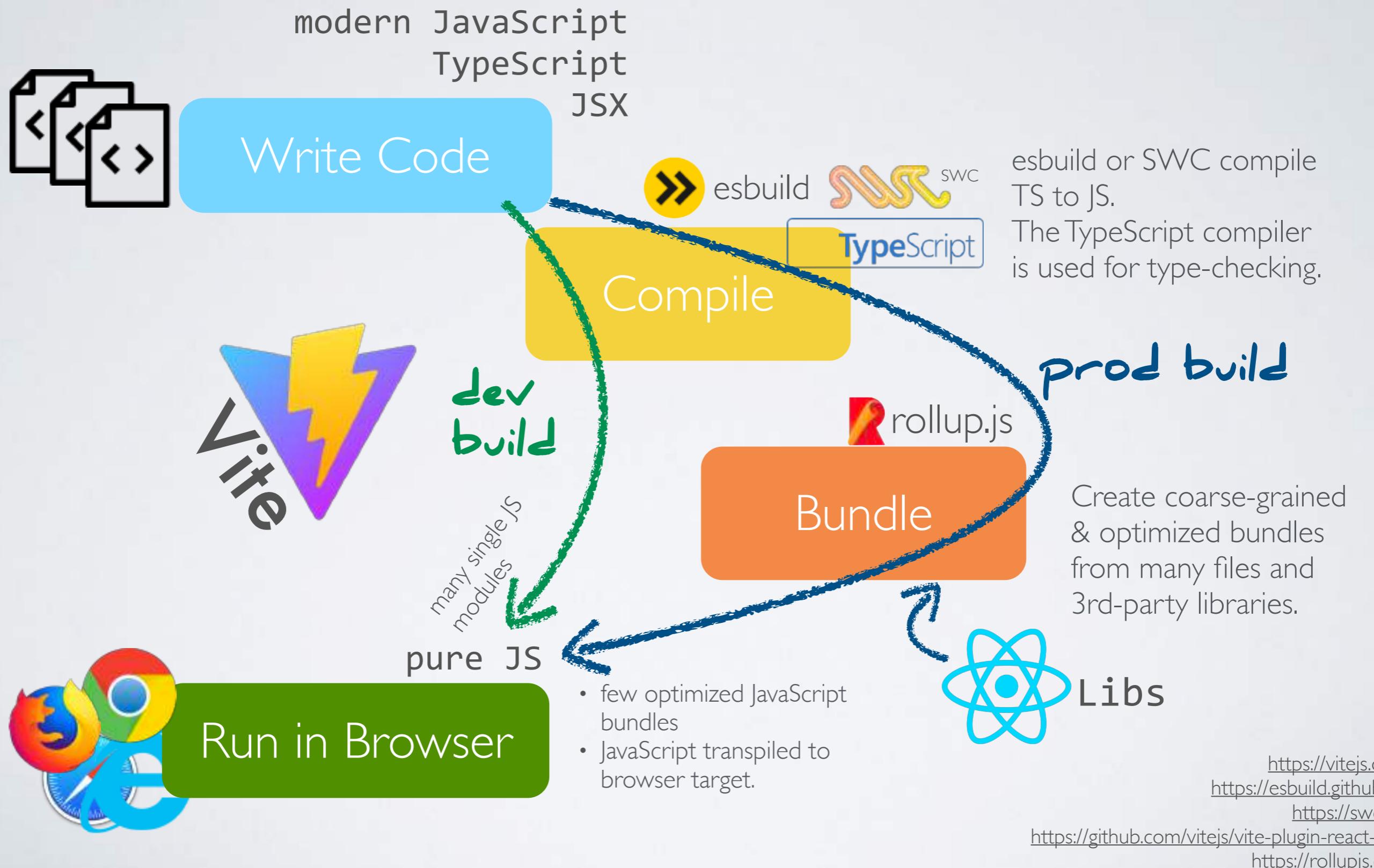
## Bootstrapping:

- index.html
- src/main.tsx

## App Component:

- /src/App.tsx
- /src/App.css

# React Project Build



# Debugging

- Debugging in Browser Developer Tools (Sources -> Ctrl-P)
- IDE Integration is easily possible with Jetbrains IDEs or VS Code

React Developer Tools:  
a browser extension to  
inspect React components

Available for Chrome and Firefox





# JSX

XML-like extension for JavaScript

# JSX

<https://facebook.github.io/jsx/>

```
const heading = <h1>Hello World</h1>;
const button = <button disabled>Click!</button>;
const div = <div> {heading} {button} </div>;
ReactDOM.createRoot(document.getElementById('root')!).render(div);
```

JSX is a XML-like syntax extension for JavaScript which defines a concise and familiar syntax for defining tree structures with attributes.

With JSX we can declare HTML structures in JavaScript.

JSX is *not* an embedded HTML template. It gets compiled to imperative JavaScript statements which use the React API to manipulate the DOM.

Other frameworks supporting JSX: Preact, Solid.js, (Astro), Qwik, Stencil, Vue.js, Inferno, Gatsby.js, Blitz.js ...  
JSX in Java: <https://complate.org/>

# JSX

JSX syntax is just syntactic sugar for the `JSX factory function.`

`(_jsx, _jsxDEV, React.createElement ...)`

```
function render() {  
  return <div color="red">Hello World</div>;  
}
```

  
function render() {  
 return \_jsx("div", {  
 color: "red",  
 children: "Hello World"  
});  
}

React 17+

```
function render() {  
  return React.createElement(  
    "div",  
    { color: "red" },  
    "Hello World"  
);  
}
```

React 16

Many tools can compile JSX to JavaScript: TypeScript, Babel, esbuild, SWC ...

- try it in the Babel REPL: <https://babeljs.io/repl/>
- try it in the TypeScript Playground: <https://www.typescriptlang.org/play>

<https://facebook.github.io/react/docs/jsx-in-depth.html>

<https://reactjs.org/blog/2020/09/22/introducing-the-new-jsx-transform.html>

# HTML in JavaScript?

- React enhances JavaScript in order to declare the UI (“If you want to read React: you have to learn JavaScript”)
  - Leverage the power of JavaScript
  - It makes sense to keep the component and the template in the same file, since they are tightly coupled
  - build-time syntax checking and error messages
  - “Separation of concerns” is still possible
- Comparison: Angular & Vue enhance HTML with their own template language  
 (“If you want to read Angular: you have to learn a long list of Angular specific syntax”)

<https://medium.com/javascript-scene/jsx-looks-like-an-abomination-1c1ec351a918#.xwa2wfc2y>

<https://medium.com/@housecor/react-s-jsx-the-other-side-of-the-coin-2ace7ab62b98>

<https://medium.freecodecamp.com/angular-2-versus-react-there-will-be-blood-66595faafdf51>



Dan Abramov

@dan\_abramov

Following

Separating concerns by files is as effective as separating school friendships by desks. Concerns are “separated” when there is no coupling: changing A wouldn’t break B. Increasing the distance without addressing the coupling only makes it easier to add bugs.

9:37 PM - 16 Jun 2018

540 Retweets 1,817 Likes



40

540

1.8K



[https://twitter.com/dan\\_abramov/status/1008131488481730561](https://twitter.com/dan_abramov/status/1008131488481730561)

# JSX: Collections

In JavaScript you can transform arrays with `map`:

```
const a = [1,2,3];
const b = a.map(e => e * e); // b = [1,4,9]
```

In JSX expressions you can produce dynamic "embedded" JSX:

```
<ul>
  {
    myArray.map(
      (e, index) => <li>Item {index} {e}</li>
    );
  }
</ul>
```

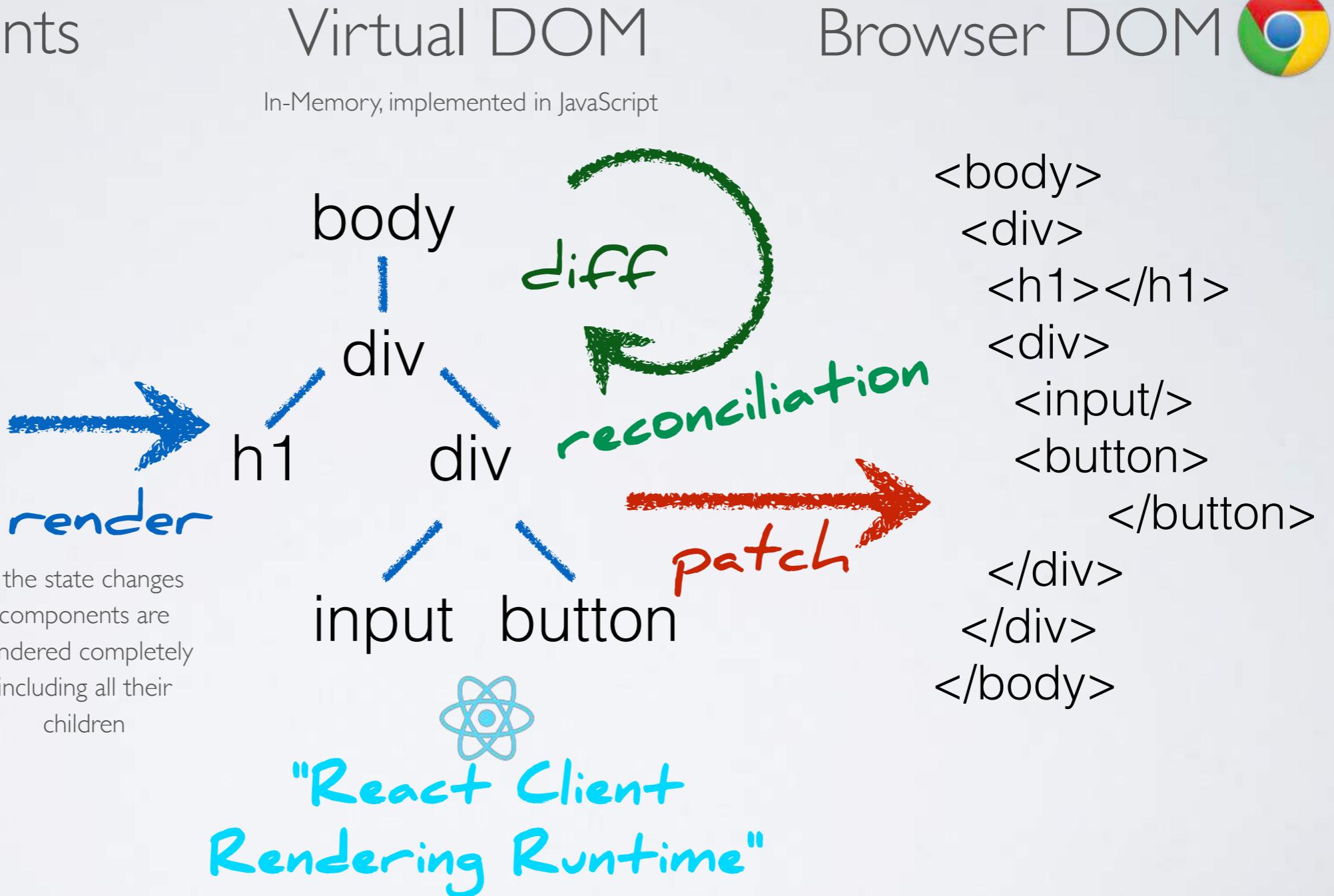
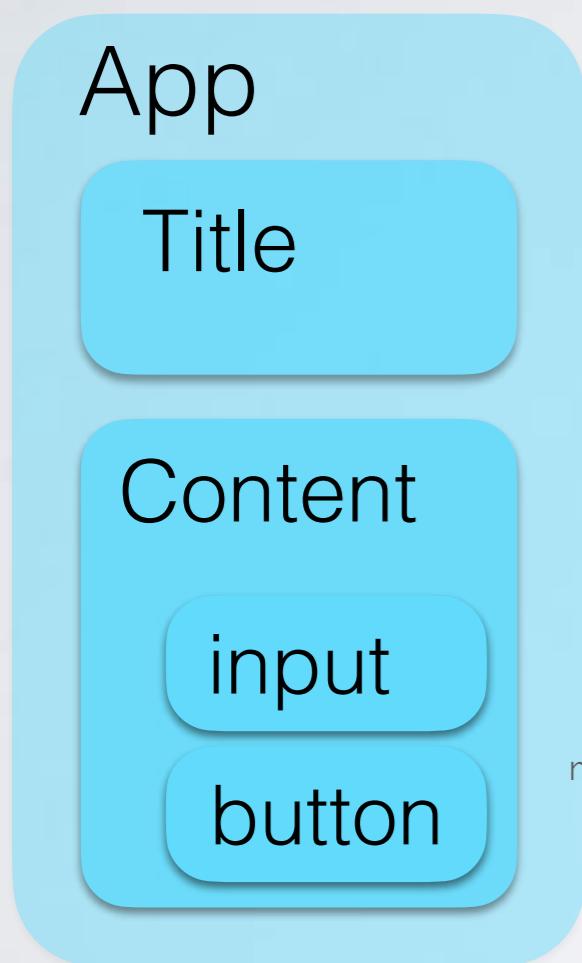
# Rendering a React Application

React itself does not render components, rendering is separated into the **react-dom** package:

```
const root = ReactDOM.createRoot(  
  document.getElementById('root') as HTMLElement  
);  
root.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>  
);
```

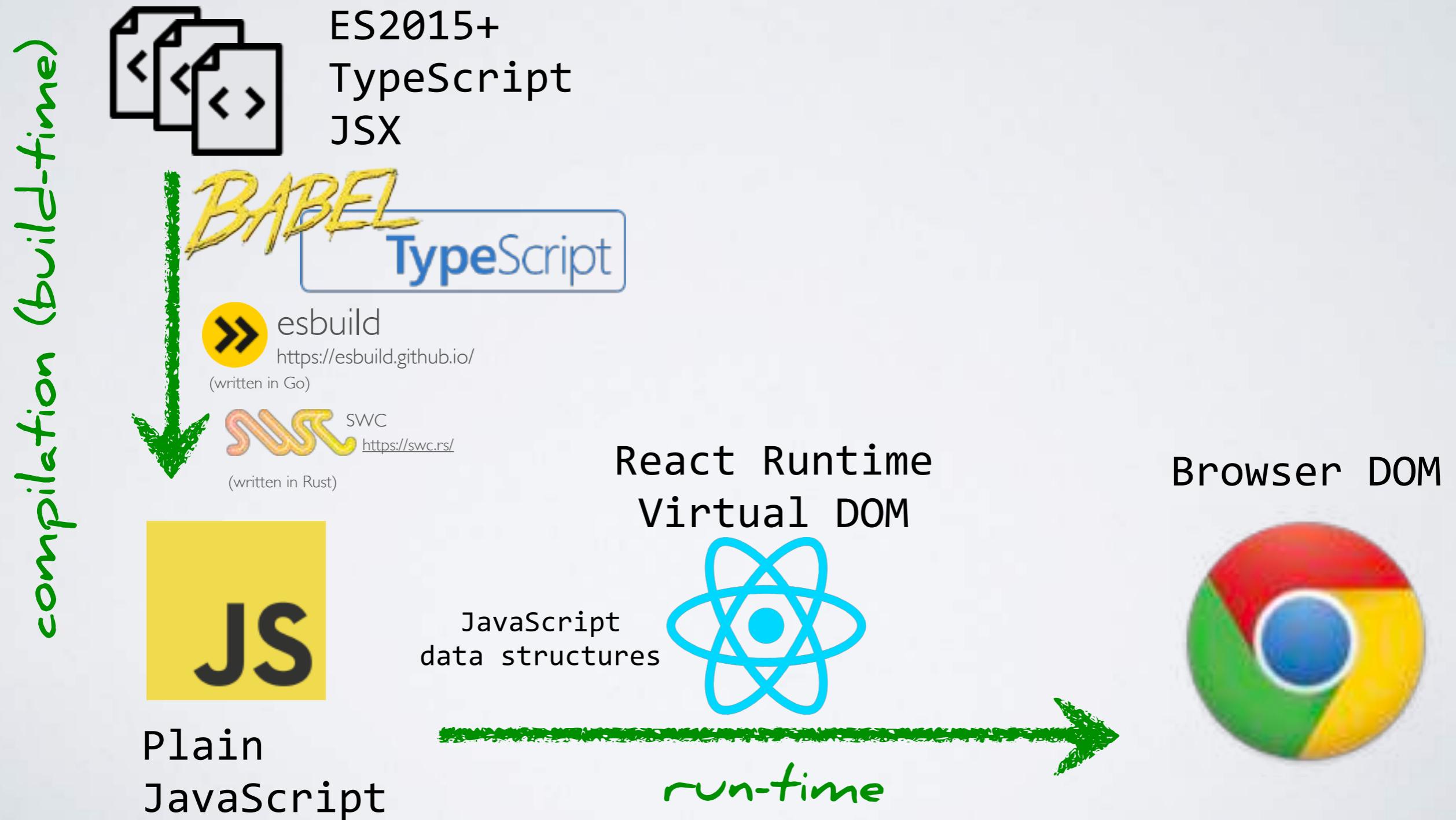
React Native provides other renderers for iOS & Android.  
Server-Side Rendering of React renders components into HTML-Text.  
React for Windows and macOS provide renderers for their platform.

# The Virtual DOM



The Virtual DOM also enables server-side rendering and rendering to iOS/Android UIs.

# Compilation in React





# Components

# React Components

React components can be implemented as functions or as classes.

```
import React from 'react';

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

```
import React from 'react';

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

"legacy"

Note: The name of a React component (class or function) has to start with an capital letter (this is the distinction to a html element in JSX).

Example usage:

```
import {Greeter} from './Greeter';
...
<div>
  <Greeter>
</div>
```

Since the introduction of *Hooks* in React 16.8) function components offer the same functionality as class components (i.e. state and life-cycle).  
Functional components are the de-facto standard in modern React development.

# Function Components

Components are written as plain JavaScript functions.



```
function AppComponent(props) {  
  return (  
    <div>  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
    </div>  
  );  
}
```



The function is called each time the UI is rendered (i.e. with every "data-change")

React expects that the body of your component behaves like a *pure function*:  
If the "inputs" (props, state, and context) are the same, it should return exactly the same JSX.  
<https://react.dev/learn/keeping-components-pure>

A Visual Guide To React Mental Models:

<https://obedparla.com/code/a-visual-guide-to-react-mental-models/>

# The Essence of Components

🎨 Rendering & Event Handling ⚡

🧩 State

♻️ Lifecycle



# Rendering &



# Event Handling

# Component Rendering

Rendering is the process of turning JavaScript data structures into DOM structures.



```
function Greeter() {  
  const name = 'Tyler';  
  return (  
    <div>  
      Hello {name}!  
    </div>  
  );  
}
```

component



Browser DOM



```
<div>  
  Hello Tyler!  
</div>
```

A component must return *only a single* element as render output.  
A component can return **null** or **undefined** (since React 18) as render output.  
From the perspective of TypeScript, the return value must be **React.ReactNode**.

Rendering should be "pure" and not have any side effects!

# Component Composition

Components can render child components.



App component

```
function App(){  
  ...  
  return (  
    <div>  
      <Greeter/>  
    </div>  
  );  
}
```



Greeter component

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



Browser DOM



```
<div>  
  <h1> Hello World! </h1>  
</div>
```

# Conditional Rendering

using variables:

```
function App(){
  const showGreeter = true;
  ...
  let greeter;
  if (showGreeter) {
    greeter = <Greeter/>
  }
  return (
    <div>
      <h1>Welcome</h1>
      { greeter }
    </div>
  );
}
```

inline expressions:

```
return (
  <div>
    <h1>Welcome</h1>
    { showGreeter && <Greeter/> }
  </div>
);
```

```
return (
  <div>
    { showGreeter ?
      <Greeter/> : <h1>Welcome</h1> }
  </div>
);
```

# Rendering Lists

Using JavaScript expressions that produce arrays of JSX elements.

```
const numbers = [1,2,3,4,5,6,7];
...
return (
  <ul>
    { numbers.map((n) => <li key={n}>{n}</li> )
  </ul>
);
```

The mapping function must return a single top-level JSX element.

Each element in the array must be assigned a unique **key** property.

```
const todoObjects = ...
...
return (
  <div>
    { todoObjects.map((n) => {
      const fullName = p.firstName + p.lastName;
      return (
        <div key={p.id}>
          <p>{fullName}</p>
          <p>{p.dateOfBirth}</p>
        </div>
      )
    })
  </div>
);
```

# React Fragment

A React Fragment is an element that can hold other elements but is not rendered into the browser DOM.

With a Fragment you can return multiple elements from React constructs that allow to return only a single element (i.e. render output of a component, collection mapping function ...)



explicit Fragment rendering

```
...  
return (  
  <React.Fragment>  
    <ChildA/>  
    <div>Hello</div>  
    <ChildB/>  
  </React.Fragment>  
);  
...
```



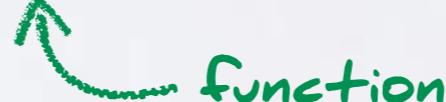
shortcut syntax supported by Babel & TypeScript

```
...  
return (  
  <>  
    <ChildA/>  
    <div>Hello</div>  
    <ChildB/>  
  </>  
);  
...
```

# Event Handling

DOM events are exposed on React elements as properties.  
You pass a callback function to the property.

```
function ClickComponent(){  
  
  function handleClick(event: React.MouseEvent<HTMLButtonElement>) {  
    alert('Event: ' + event.type);  
  }  
  
  return (  
    <button onClick={handleClick}>  
      Click me!  
    </button>  
  );  
}  
export default ClickComponent;
```



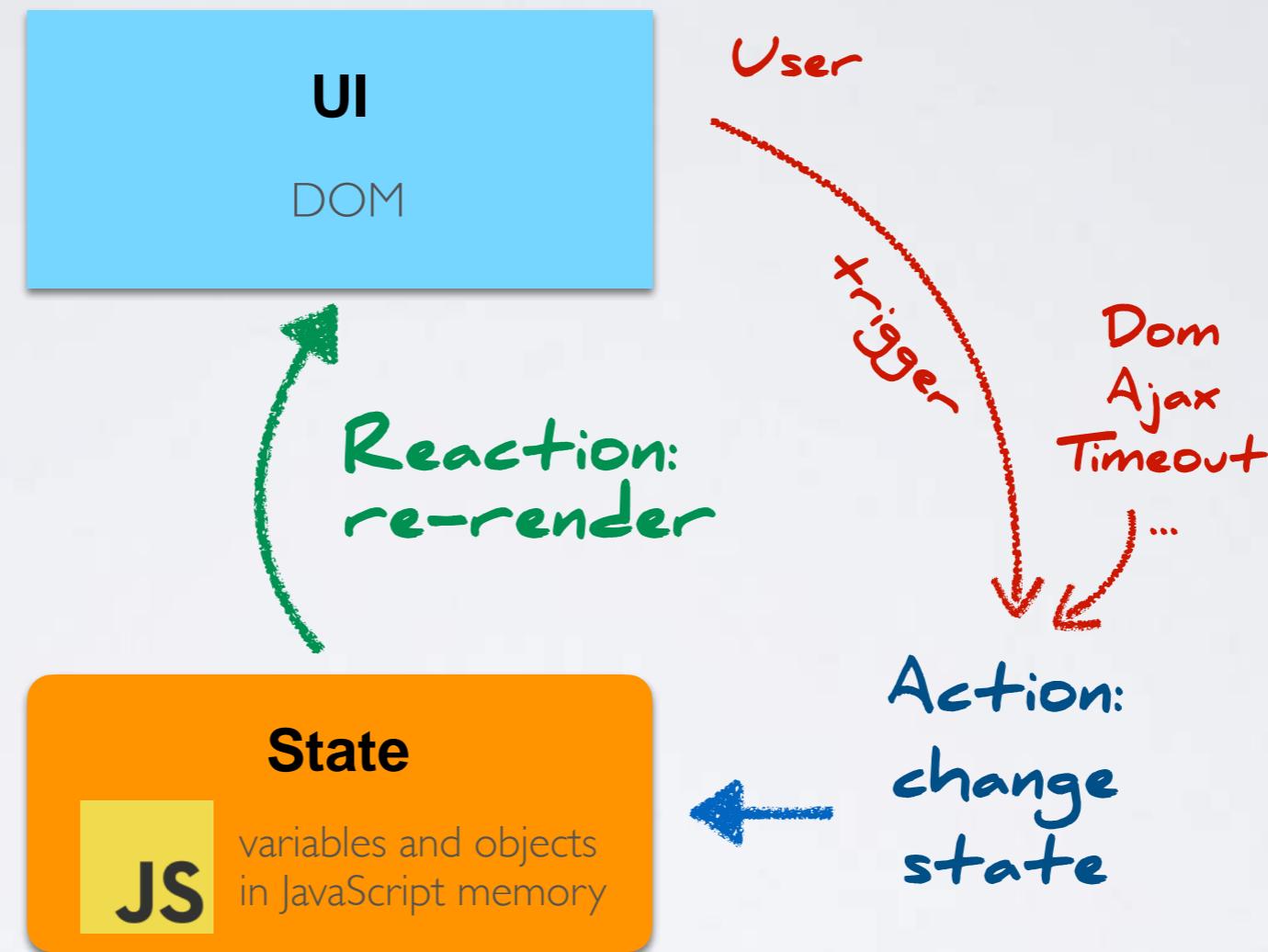
Note: React events are named in camelCase

React provides a *synthetic event system*: event handlers are registered with React and not on DOM elements. React listens to all DOM events on the document and calls the appropriate handlers.



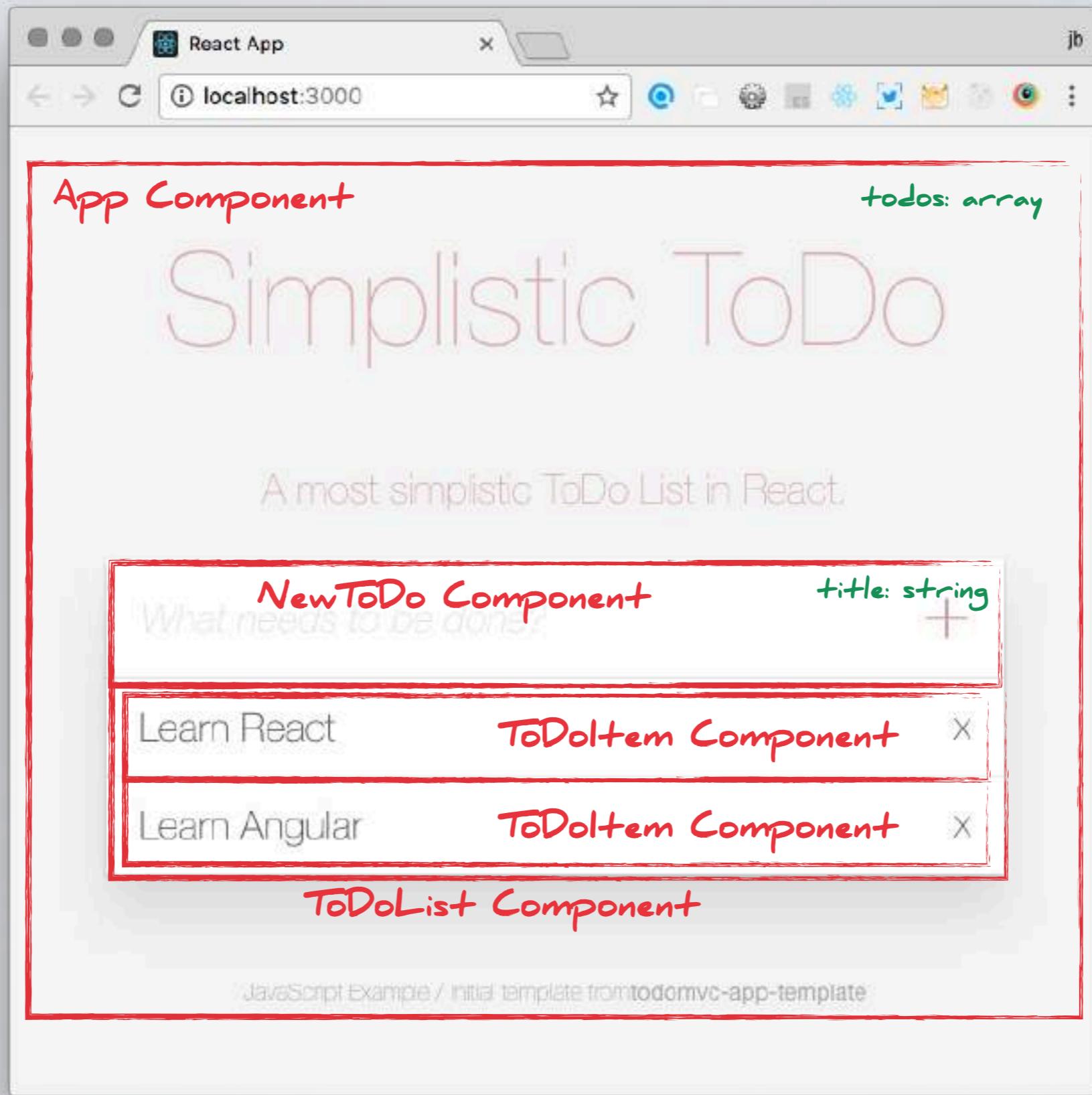
State

# State is Managed in JavaScript



Reactivity: React reacts on state changes and updates the UI.

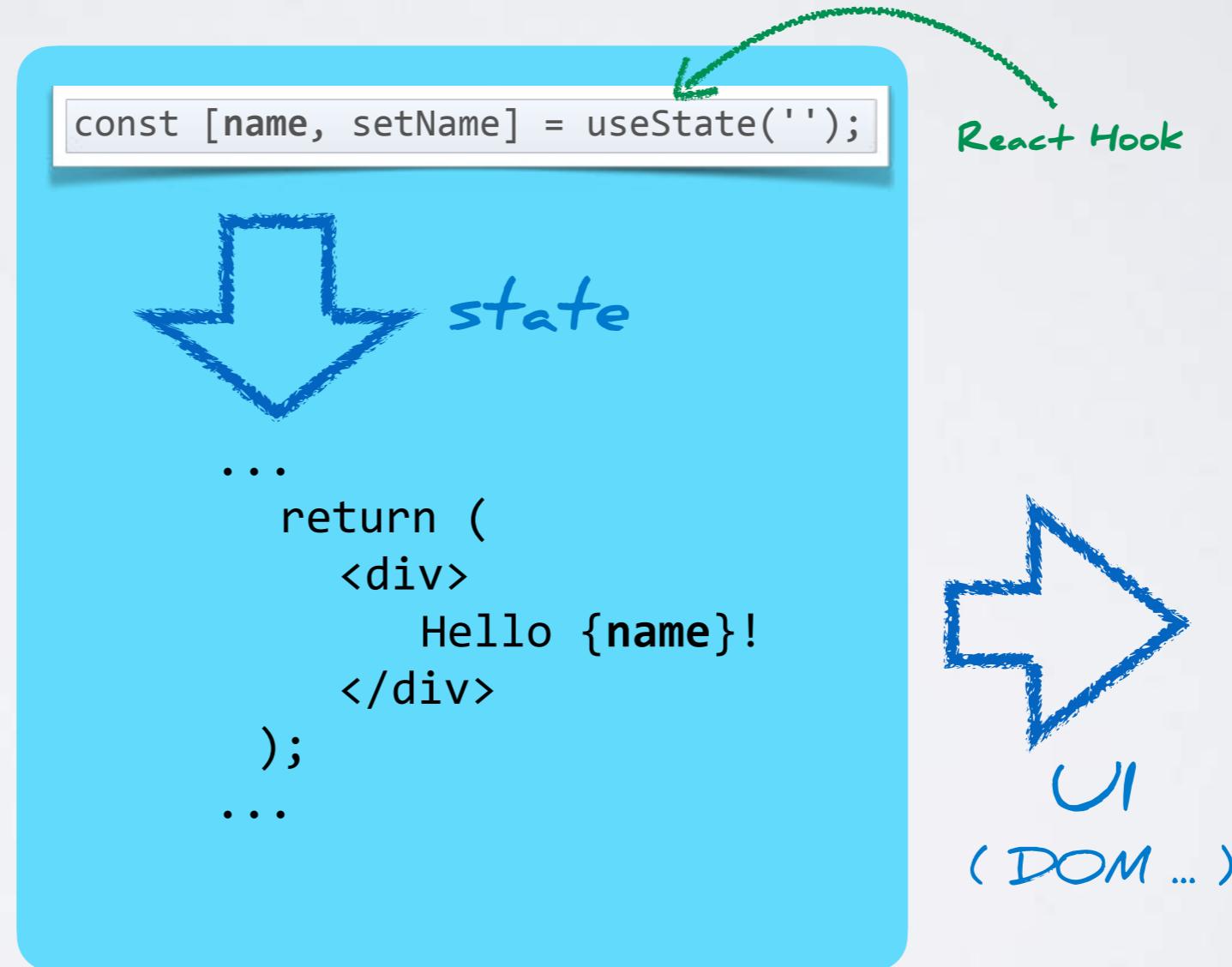
# State in a Component Tree



# Stateful Components

A component can hold state.  
The state is changed by the component.

When the state changes, the component and its children are re-rendered.



# State in Function Components

```
function Greeter() {  
  const [name, setName] = useState('');  
  
  return (  
    <div>  
      <input value={name} onChange={(event) => setName(event.target.value)} />  
      <div>{name}</div>  
    </div>  
  );  
}
```

Note: The example above uses *array destructuring* of ES2015

`useState(initialValue)` is a Hook.

When it is called the first time it instructs React to manage a "slice" of state.  
Each time it is called, it returns the current state and a function to update the state.

Calling the update function:

- sets the complete state (no shallow merge)
- triggers a re-rendering of the component

Note: there are other Hooks to manage state:

`useRef, useReducer, useContext ...`

<https://reactjs.org/docs/hooks-reference.html#usestate>

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

# Functional State Updates: useState

State updates may be asynchronous!

The update function of the useState Hook also accepts a function:

```
const [state, setState] = useState(initialState);
setState((currentState) => newState);
```

```
function MyComponent(){
  const [count, setCount] = useState(0);

  function increment(){
    setCount(count + 1);
    console.log('count after set', count);
    setCount(count + 1);
  }

  return (
    <div>
      <div>Count: {count}</div>
      <button onClick={increment}>Increase</button>
    </div>
  );
}
```

wrong!

count is not incremented yet!

```
function increment(){
  setCount((count) => count + 1);
  setCount((count) => count + 1);
}
```

correct!

the updated count  
will be passed!

# Passing Data to Components

A component can receive **props**.

```
type GreeterProps = { message: string; }

function Greeter(props: GreeterProps) {
  ...
  return (
    <div>
      Hello {props.message}!
    </div>
  );
}
```

It is a good practice to *destruct*ure props in the component signature:

```
function Greeter({message}: GreeterProps) {
```

```
  <div> Hello {message} </div>
```

UI  
(DOM ...)

**props** is data owned by some other component and should not be changed (it should be treated as immutable data)

In TypeScript it could be declared as `ReadyOnly<GreeterProps>`

# Passing Props

```
function App() {  
  
  const today = new Date();  
  
  return (  
    <div>  
      <Info message={'Current date ' + today}>  
    </div>  
  );  
}
```

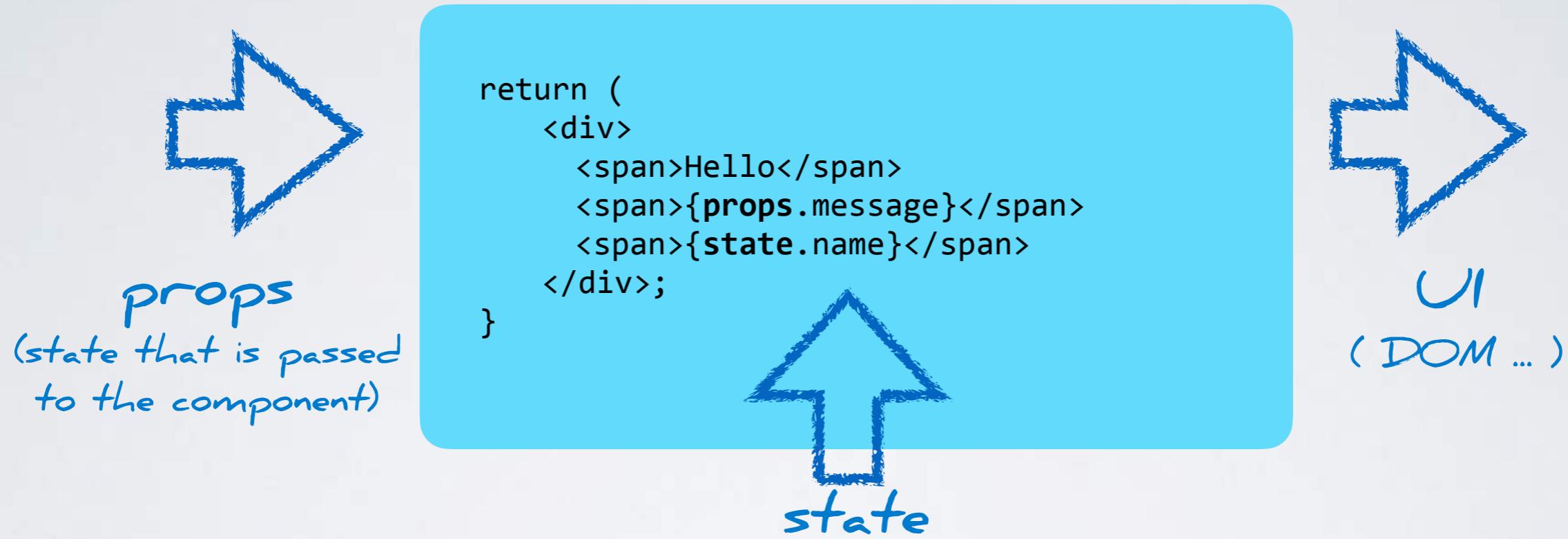
props: {message: 'Current date Tue Nov 15 2022'}

```
type InfoProps = {message: string};  
function Info({message}: InfoProps) {  
  
  return (  
    <div>Message: {message}</div>  
  );  
}
```

Every function component receives a **props** argument. It is an object that holds all the props that are passed to the component instance.

# Rendering Updates

A component transforms JavaScript state into the DOM structure.



- When state changes, the whole component is re-rendered into the virtual dom.
- When a component is rendered all it's children are also rendered.
- Since **props** are state of a parent components, a component typically re-renders when **props** change
- **React.memo** is a mechanism to prevent re-rendering a component if its **props** did not change.

<https://www.joshwcomeau.com/react/why-react-re-renders/>

<https://blog.isquaredsoftware.com/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior/>

# Custom Events

React components can expose events via **props**.

The parent component passes an event handler (callback function) to the child component. The child component then can trigger the handler of the parent component.

```
function Parent() {  
  function handler(){ ... }  
  return (  
    <Child onInnerClick={handler} />  
  );  
};
```

*prop*  
*function*

Because **handler** is declared inside **Parent**, it has access to the props and state of **Parent**! (via the JavaScript Closure)

```
type ChildProps = {  
  onInnerClick: () => void;  
};  
prop destructuring  
function Child({onInnerClick}) {  
  ...  
  return (  
    <button onClick={onInnerClick}>  
      Click Me 1  
    </button>  
  );  
};
```

*function*



Lifecycle

# Component Lifecycle

React renders your components.

Your code does not call function components or instantiate class components. React does this.

The component "lifecycle" is managed by React:

- A component is "mounted" when it is rendered to the DOM for the first time
- A component can be *updated* while it is present in the DOM
- A component is "unmounted" when it is removed from the DOM

Typically application logic is coupled to the lifecycle of components (i.e. loading data, starting timers, manual DOM changes ...).

This logic is often called "side effects" since it has an effect outside of the component tree.

# Lifecycle: `useEffect`

A component has a "lifecycle": created, updated, deleted.  
But Functions do not have a "lifecycle"! `useEffect` provides an API to "tap" into  
the component lifecycle.

The `useEffect` Hook is used to trigger side effects coupled to the component lifecycle.  
Components re-rendered on any change and therefore also for each "lifecycle-stage".

The `useEffect` Hook can be used for several scenarios:

- to trigger logic *after each rendering*.
- to trigger logic *after a rendering* when the certain props or parts of the state did change.
- to trigger logic *(only) after the first rendering*.

The `useEffect` Hook can optionally define *cleanup logic*, which is executed by React.

The mental model of `useEffect` is closer to implementing "synchronization" than to responding to lifecycle events.

React 18 Strict Mode executes the effects twice in development mode!  
<https://reactjs.org/blog/2022/03/08/react-18-upgrade-guide.html#updates-to-strict-mode>

In future React versions components may "mount" and "unmount" more than once!

<https://react.dev/reference/react/useEffect>

<https://react.dev/learn/you-might-not-need-an-effect>

<https://overreacted.io/a-complete-guide-to-useeffect/>

Note: There is also `useLayoutEffect`:

<https://react.dev/reference/react/useLayoutEffect>

<https://kentcdodds.com/blog/useeffect-vs-uselayouteffect>

# Lifecycle: useEffect

```
useEffect(() => {
  performSideEffect();
  return () => { cleanup(); }
}, [dep1, dep2]);
```

the actual effect

optional cleanup

optional dependencies: effect is only re-executed if dependencies changed.

Should typically contain all the variables used in the effect function.

Executing a function after each render:

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

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(document.getElementById('message').innerText);
  });

  return (
    <div>
      <p id="message">You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Example;
```



access delayed until after render

(note: accessing the document directly is an antipattern in React)

Executing a function after first render,  
cleaning up when component is removed:

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

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Mounted!');
  }, []);

  return (
    <div>
      <p id="message">You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Example;
```

executed when removed

only executed on first render

# You Might Not Need an Effect

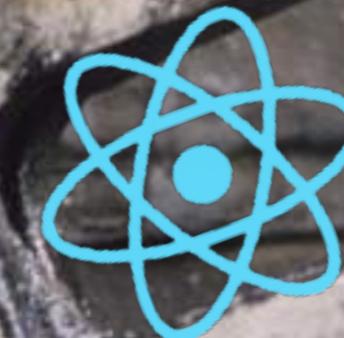
<https://react.dev/learn/you-might-not-need-an-effect>

Effects are an escape hatch from the React paradigm. [...] If there is no external system involved (for example, if you want to update a component's state when some props or state change), you shouldn't need an Effect.

Removing unnecessary Effects will make your code easier to follow, faster to run, and less error-prone.

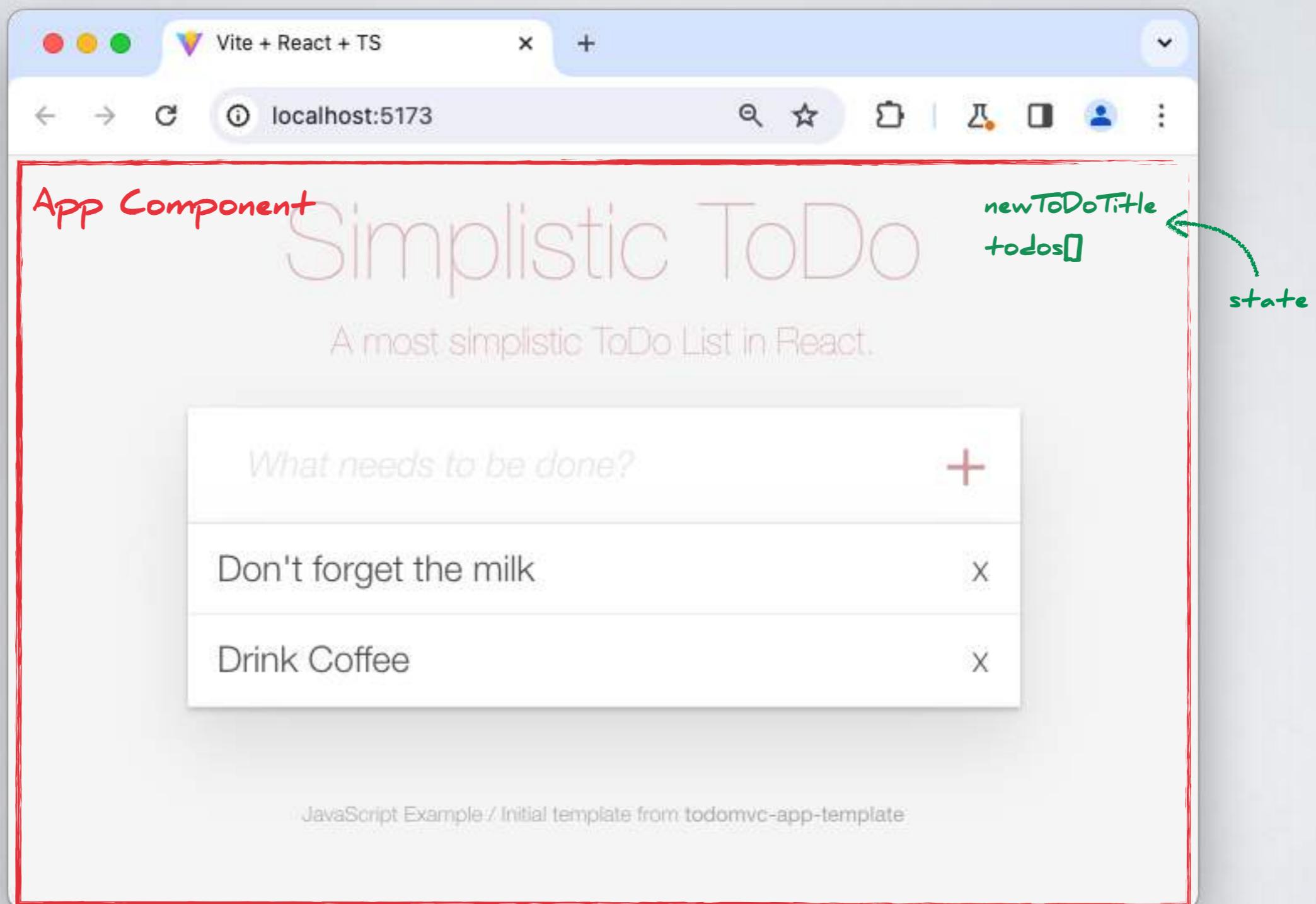
If you need an effect, think about "hiding" it in a custom hook. The fewer raw useEffect calls you have in your components, the easier you will find to maintain your application.

# EXERCISES



Exercise 3 - Simple ToDo App

# Simple ToDo App



# Form: Controlled Components

```
function Form() {
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');

  function submitForm() {
    console.log('Form Values', firstName, lastName);
  }

  return (
    <div className="card">
      <form action={submitForm}>
        <label htmlFor="firstName">Firstname:</label>
        <input value={firstName} onChange={(e) => setFirstName(e.target.value)}
               type="text" id="firstName" name="firstName" />
        <br />
        <label htmlFor="lastName">Lastname:</label>
        <input value={lastName} onChange={(e) => setLastName(e.target.value)}
               type="text" id="lastName" name="lastName" />
        <br/>
        <button type="submit">Submit</button>
      </form>
      <p>
        Hello {firstName} {lastName}
      </p>
    </div>
  );
}
```

This is the basic mechanism provided by React.

There are many alternative techniques:

- uncontrolled components using ref or action.
- custom hook with an object in state.
- useActionState for async state changes
- 3rd party form libraries

# Form Submission in React 19

React 19 introduces Actions:

<https://vercel.com/blog/whats-new-in-react-19#actions>

An action can be used to handle form submission:

```
function FormComponent() {  
  
  function submitForm(formData) {  
    const firstName = formData.get("firstName");  
    console.log('Form Values', firstName);  
  }  
  
  return (  
    <div className="card">  
      <form action={submitForm}>  
        <label htmlFor="firstName">Firstname:</label>  
        <input type="text" id="firstName" name="firstName" />  
        <button type="submit">Submit</button>  
      </form>  
      <p>  
        Hello {firstName}!  
      </p>  
    </div>  
  );  
}
```

Caution:  
Pitfall!

# Form Submission in React before v19

```
function FormComponent() {
  const [firstName, setFirstName] = useState('');

  function submitForm(e: FormEvent) {
    e.preventDefault(); ← Preventing the default
    console.log('Form Values', firstName);
  }

  return (
    <div className="card">
      <form onSubmit={submitForm}>
        <label htmlFor="firstName">Firstname:</label>
        <input value={firstName} onChange={(e) => setFirstName(e.target.value)} type="text" id="firstName" name="firstName" />
        <button type="submit">Submit</button>
      </form>
      <p>
        Hello {firstName}!
      </p>
    </div>
  );
}
```

Preventing the default behavior of the browser with this form submission!

If a form is submitted, the browser will trigger a request to the backend (GET or POST):

[https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending\\_and\\_retrieving\\_form\\_data](https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data)

React does not "magically" prevent this default behavior. We have to prevent the default behavior in the event handler.



Immutability

# Immutability

React propagates a programming model based on immutable data structures: props & state should be treated as immutable.

- props are owned by a parent and should only be changed by this parent ("unidirectional data-flow")
- state changes are more explicit and easier to track if state is immutable
- React can better optimize re-rendering based on shallow comparison of objects.

```
const [todos, setTodos] = useState([]);  
...  
let newTodos = [...this.state.todos, newToDo];  
setTodos({todos: newTodos});
```

the **useState** Hook only re-renders if we pass a new value to the setter function

## Why immutability is important?

<https://reactjs.org/tutorial/tutorial.html#why-immutability-is-important>

<https://reactkungfu.com/2015/08/pros-and-cons-of-using-immutability-with-react-js/>

Why is react doing this? <https://gist.github.com/sebmarkbage/a5ef436427437a98408672108df01919>

Why does React care about purity? <https://beta.reactjs.org/learn/keeping-components-pure>

# Immutable Update Patterns

Whenever your object would be mutated, don't do it.

Instead, create a changed copy of it.

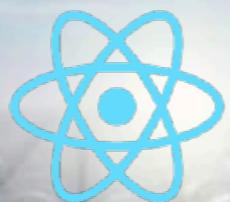
## array changes

```
const arr1 = ['first', 'second', 'third'];
const arr2 = [...arr1, 'fourth']; // adding
const arr3 = arr2.filter(e => e !== 'second'); //removing
const arr4 = [...array.slice(0, 2), 'between', ...array.slice(2)]; //inserting
```

## object changes

```
const person = {firstName: 'Tyler', lastName: 'Durden', age: 42,
               address: {street: 'papermill', number: 2}};
const changedPerson = {...person, age: 43};
const personWithChangedAddress = {...person, address: {...person.address, number: 3}};
```

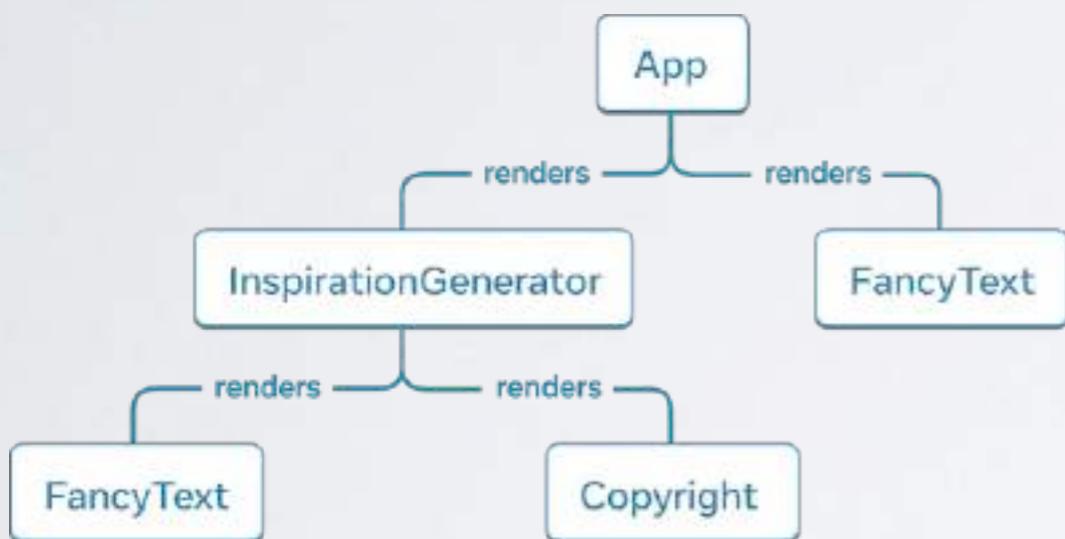
```
const arr = [{id: 1, title: 'eat', completed: true},
             {id: 2, title: 'learn', completed: false},
             {id: 3, title: 'sleep', completed: false}];
const changed = arr.map(e => e.id !== 2 ? e : {...e, completed: true})
```



# Data Flow

# Data Flow Mechanisms

The UI is a tree of components:



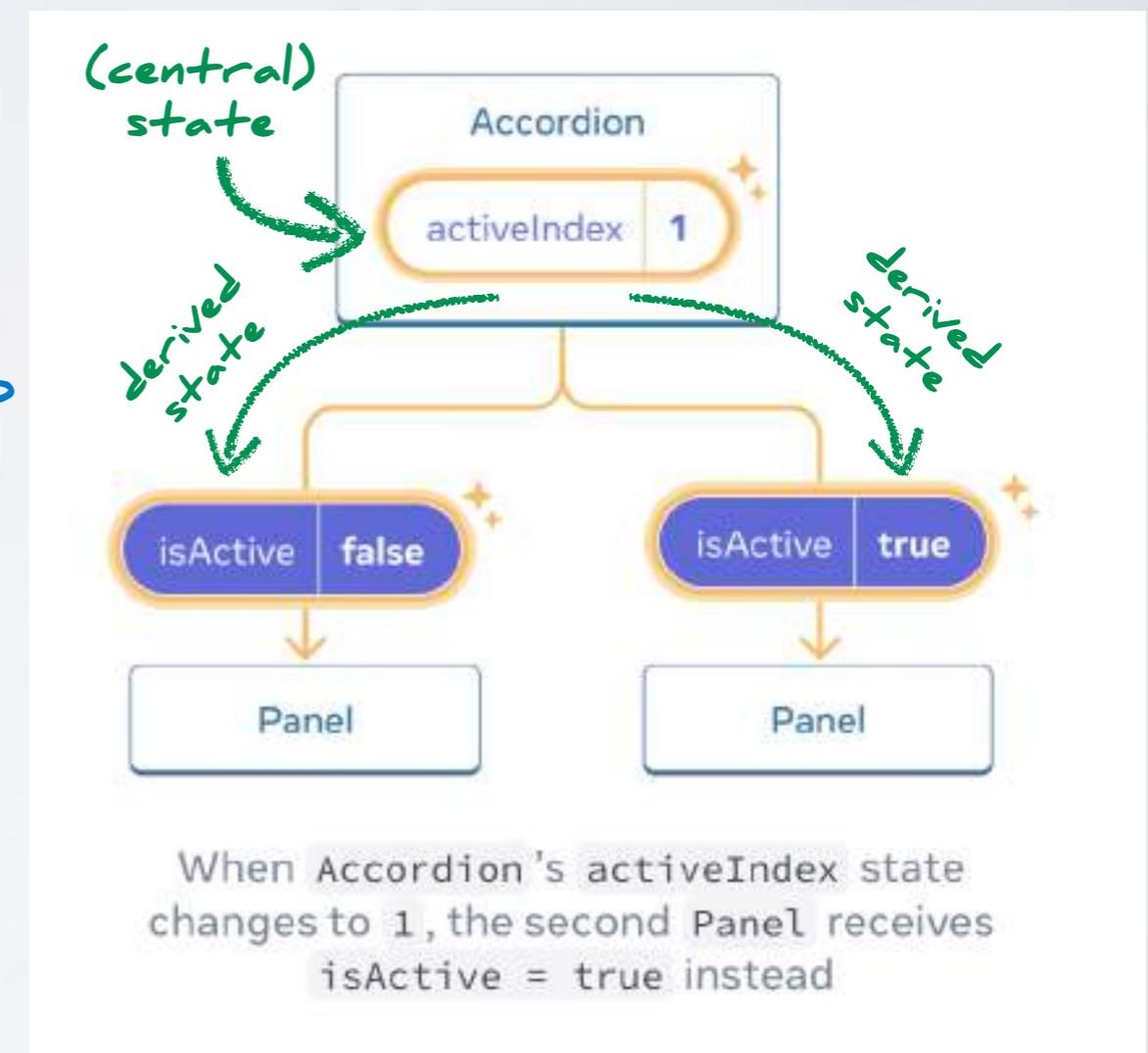
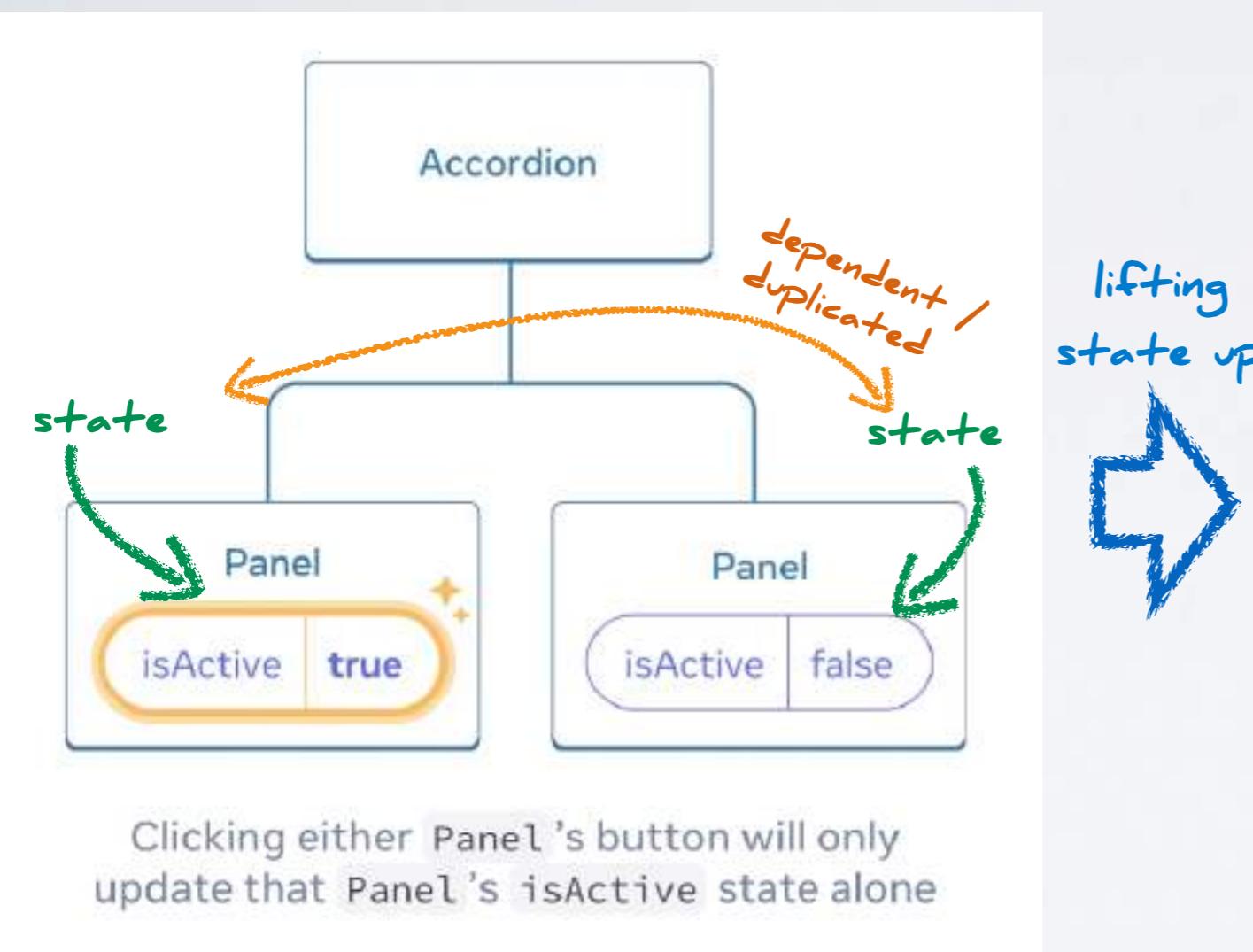
<https://react.dev/learn/understanding-your-ui-as-a-tree>

Data and event handlers are passed to child components via props.

```
type ChildProps = {  
  data: string;  
  onEvent: () => void;  
};  
  
function Child({data, onEvent}) {  
  ...  
  return (  
    <div>  
      <h1>{data}</h1>  
      <button onClick={onEvent}>  
        Click Me 1  
      </button>;  
    </div>  
  );  
}
```

# Thinking in React: Lifting State Up

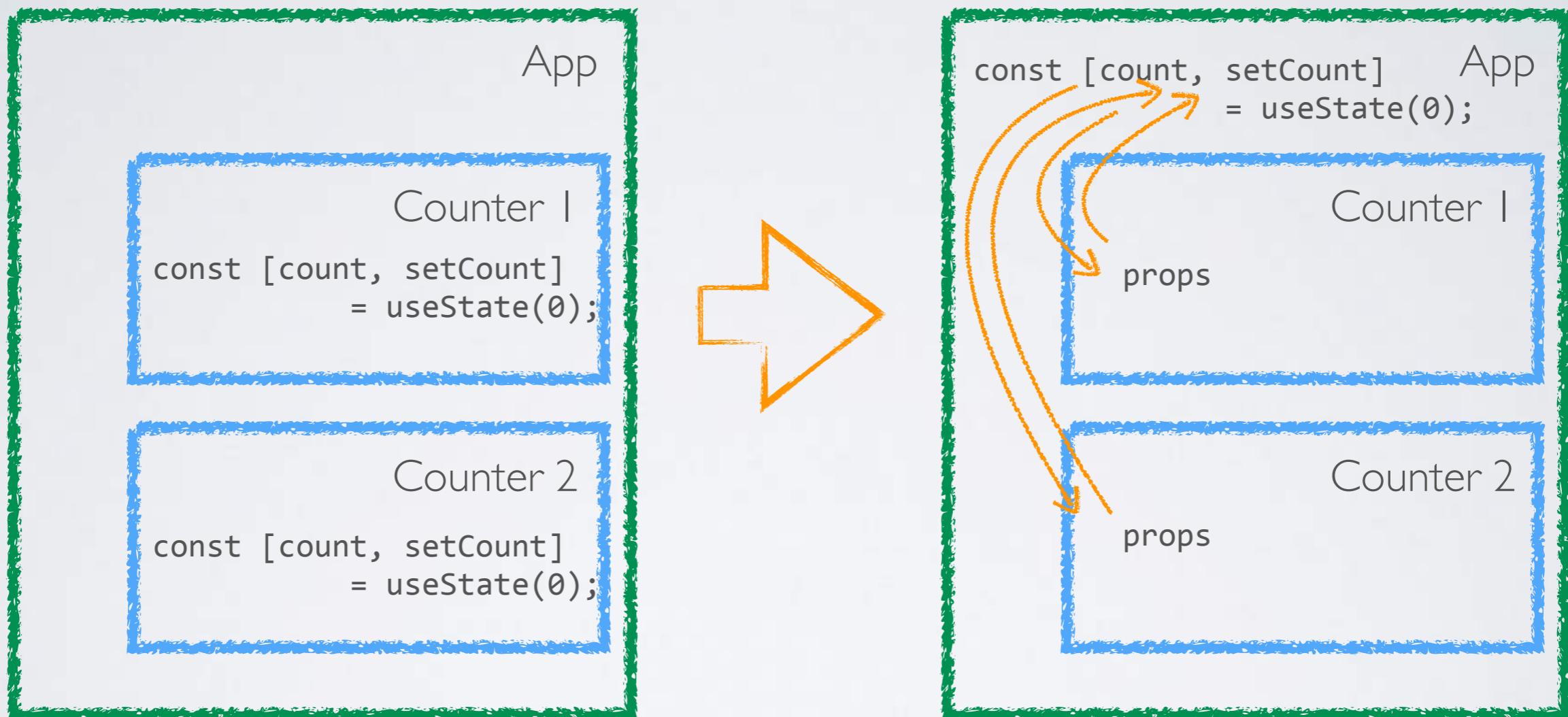
State should be modeled as a single source of truth.  
Duplicated state should be avoided! Sometimes duplication can be avoided by deriving dependent state from existing state.



<https://react.dev/learn/sharing-state-between-components>

<https://react.dev/learn/thinking-in-react>

# Lifting State up



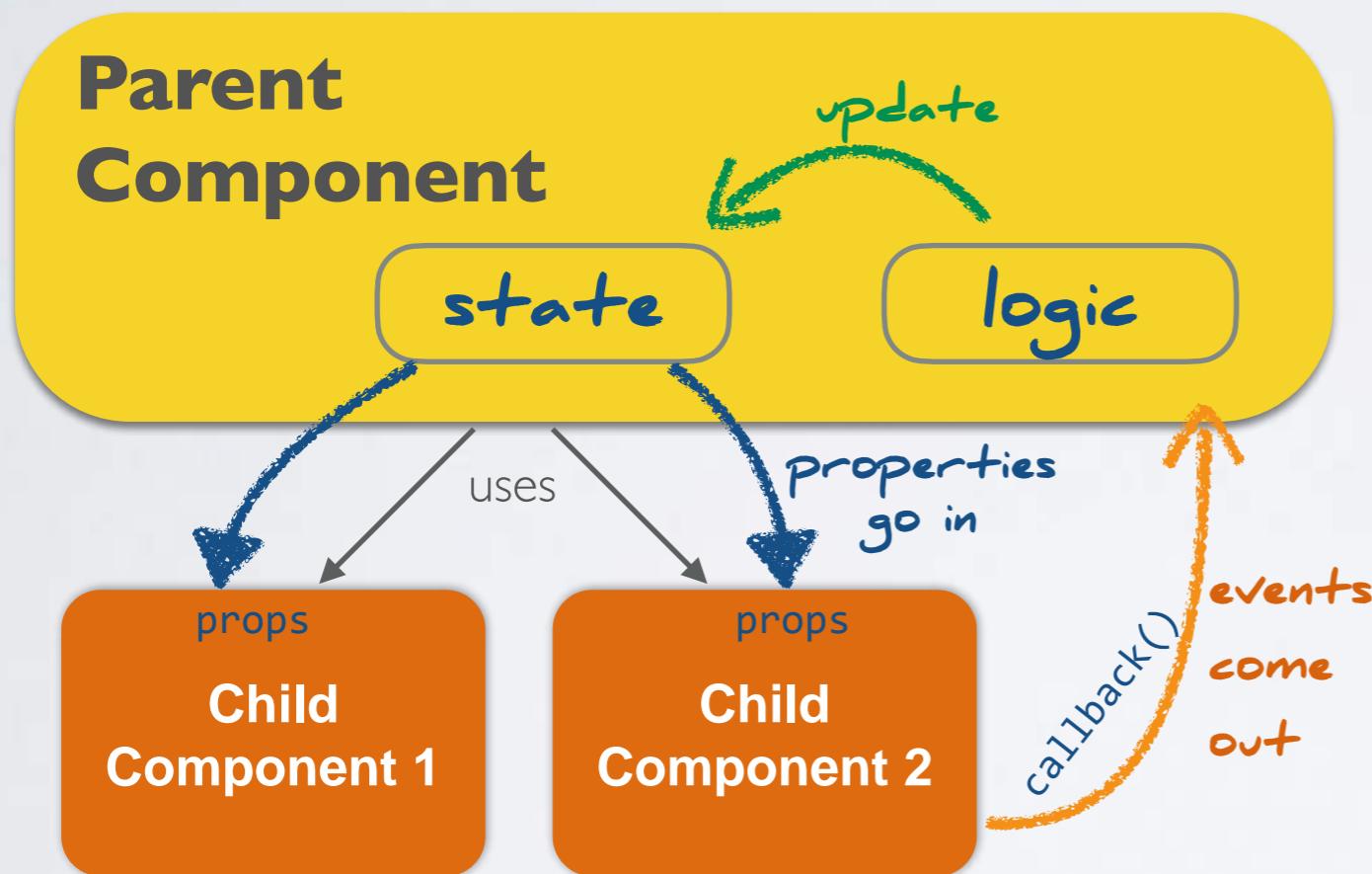
<https://react.dev/learn/sharing-state-between-components>

# Unidirectional Data-Flow

State should be explicitly owned by one component.

State should *never* be duplicated in multiple components.

(sometimes it is tricky to detect the *minimal state* from which other state properties can be derived)



- A parent component passes state to children as properties. Children are re-rendered when these properties change.
- Children should not edit state of their parent
- Children “notify” parents (events, callbacks ...)

React formalises **unidirectional data-flow** via **props**:  
passing data and callbacks to child components

# Data flows Down / Events flow Up

Passing props from parent to child:

```
function Parent() {  
  
  let message = 'Test';  
  function handler(){  
    console.log('Child clicked!');  
  }  
  
  return (  
    <Child  
      message={message}  
      onInnerClick={handler}>  
    </Child>  
  );  
};
```

Using props and emitting events:

```
interface ChildProps {  
  message: string,  
  onInnerClick: () => void  
}  
function Child({message, onInnerClick}: ChildProps){  
  return (  
    <div>  
      <div>{message}</div>  
      <button  
        onClick={onInnerClick}>  
        Click Me  
      </button>;  
    </div>  
  );  
};
```

This programming model is very similar to the native DOM elements:

```
<input value={firstName} onChange={handleChange} />
```

# Container vs. Presentation Components

"Separation of Concerns"

Application should be decomposed in container- and presentation components:

<b>Container</b>	<b>Presentation</b>
Little to no markup	Mostly markup
Pass data and actions down	Receive data & actions via props
typically stateful / manage state	mostly stateless better reusability

aka: Smart- vs. Dumb Components

# Separation of Concerns

Separation of concerns is not equal to  
separation of file types!

Keep things together that change together.

You can split a component into a controller and a view:

```
import {View} from './View';

export function Controller {
  ... // state & behavior
  return (
    <View data={...}
          onEvent={...} />
  );
}
```

Controller.js

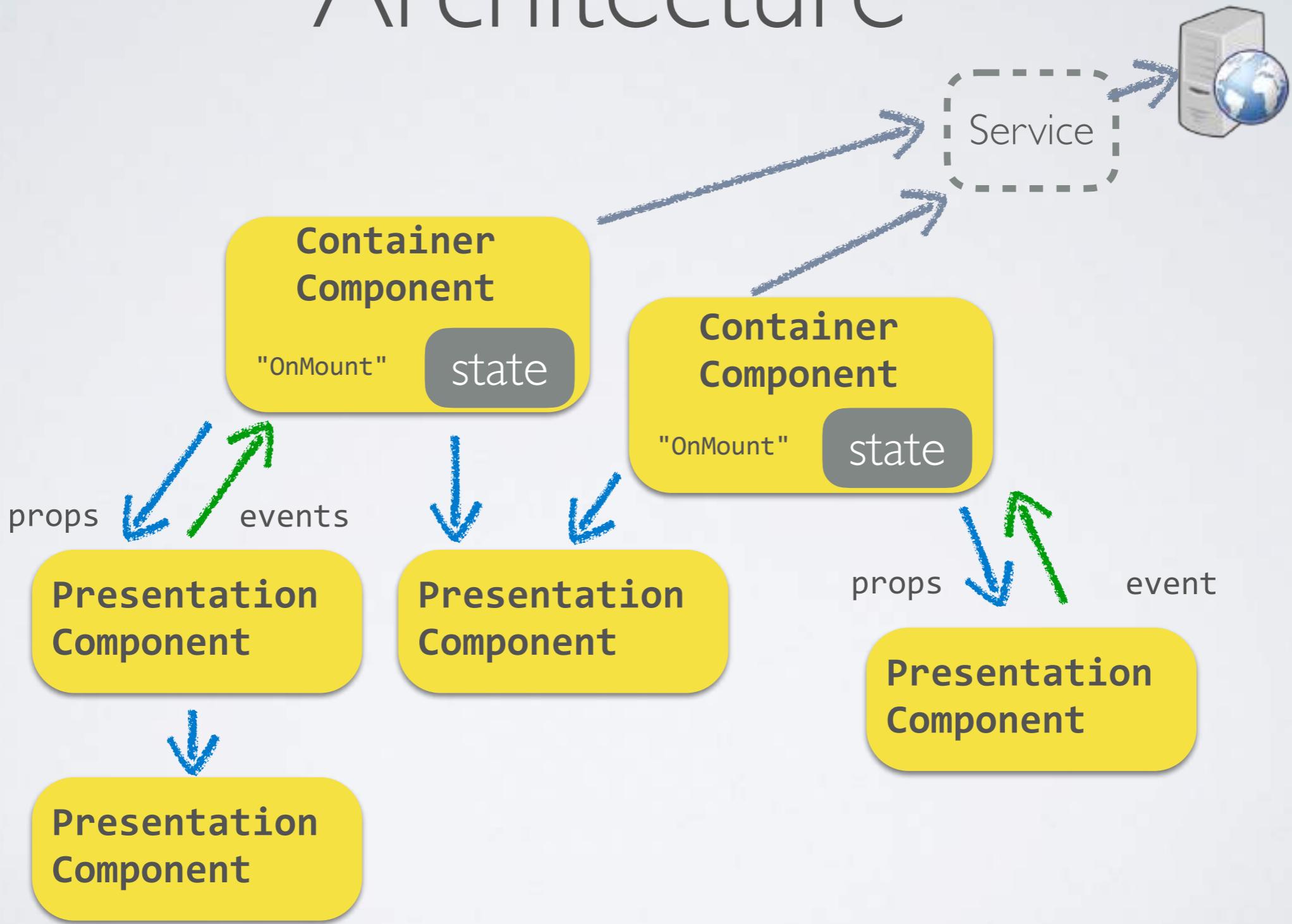
```
export function View({data, onEvent}){
  return (
    <div>
      {data.message}
      <button onClick={()=>onEvent()}>
        Go!
      </button>
    </div>
  );
}
```

View.js

<https://codesandbox.io/s/NxqMqyxID>

<https://medium.com/styled-components/component-folder-pattern-ee42df37ec68>

# Data Flow & Component Architecture

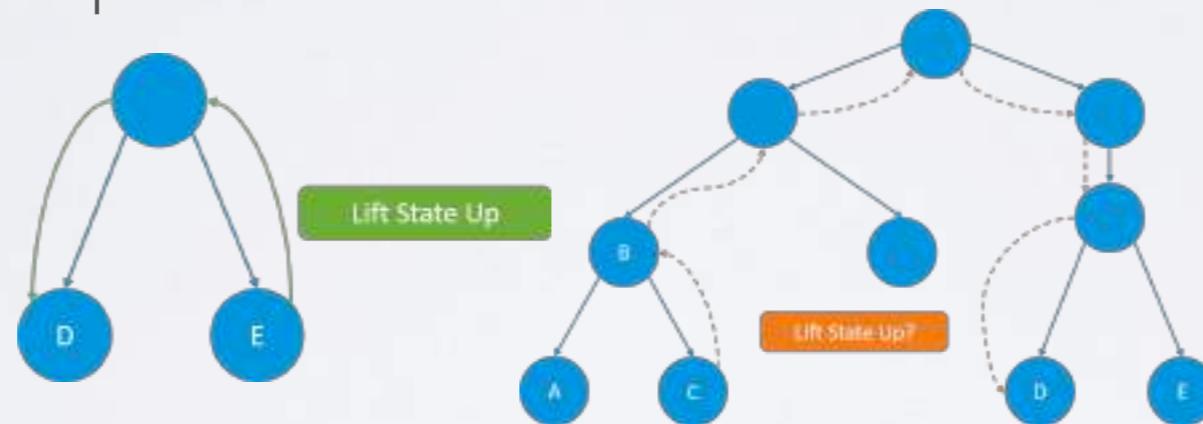


# React Data Flow Critique

In React the data flow is heavily coupled to the component structure and the re-rendering of the component tree.

The idea behind "lifting" state up is, that the component tree matches the data-flow tree. This is an abstraction that might become a problem for more complicated applications.

At a certain point, "lifting state up" will hurt the "Composability and Reusability" principles.



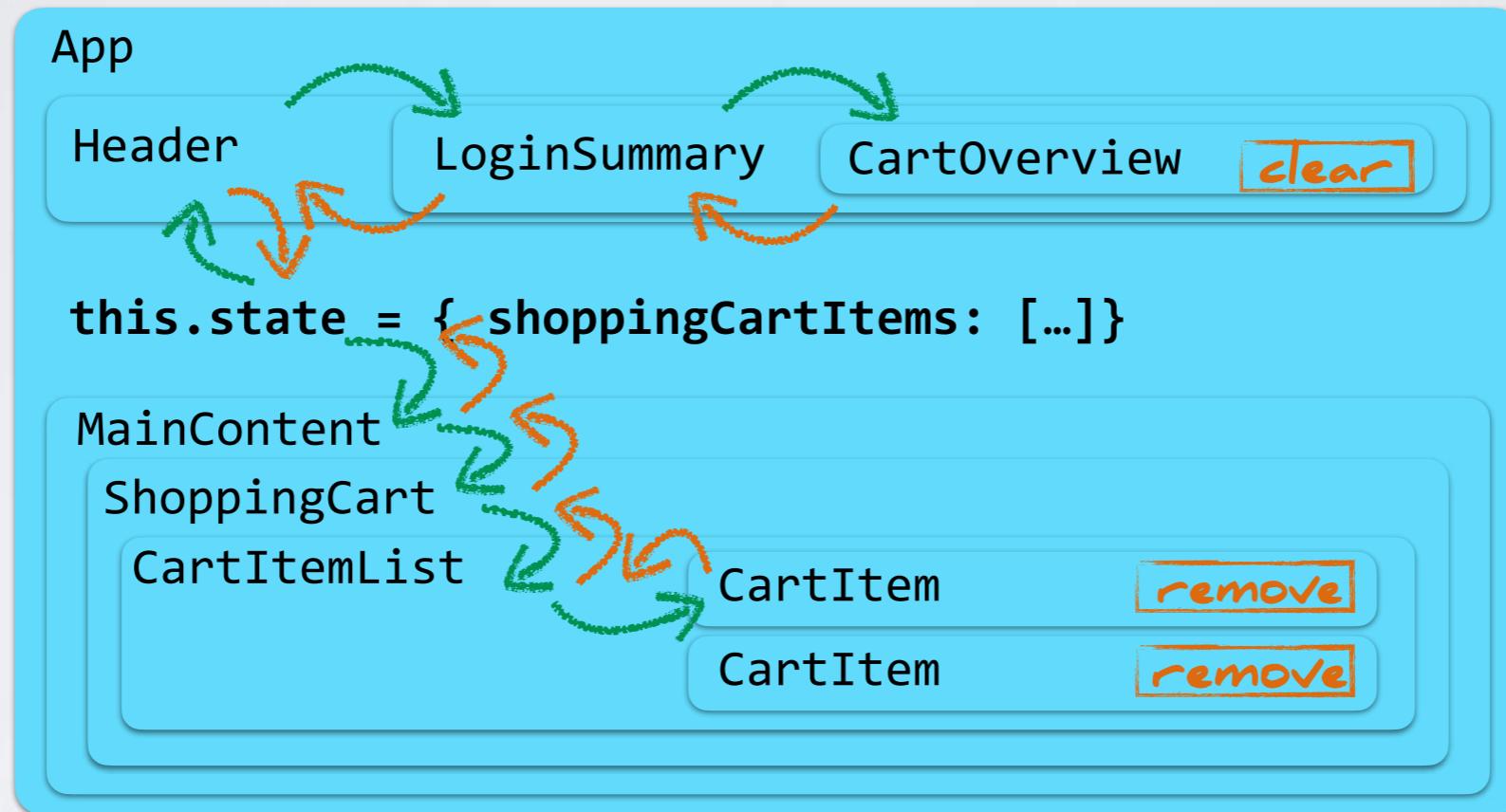
"Fine-grained reactivity" is not possible in React. Changes only propagate if a component-tree is re-rendered.

Some 3rd party solutions propose to "lift state out" instead of "lifting state up".

# Prop Drilling

From the component architecture follows the pattern of "lifting state up": if several components need to reflect the same changing data, then the shared state should be lifted up to their closest common ancestor.

<https://react.dev/learn/sharing-state-between-components>



"Prop Drilling" is the process you have to go through to pass data and events through the component tree.

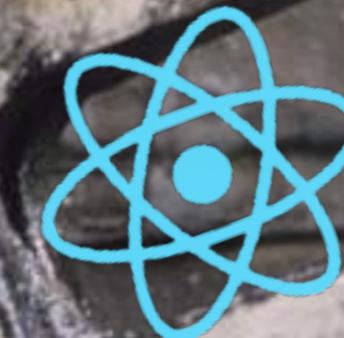
Prop drilling can be a *good thing*: it makes the data-flow very explicit!

Prop drilling can be a *bad thing*: passing data from its holder to a consumer via several intermediates is tedious and makes changing the component tree more difficult.

<https://blog.kentcdodds.com/prop-drilling-bb62e02cb691>

<https://react.dev/learn/passing-data-deeply-with-context>

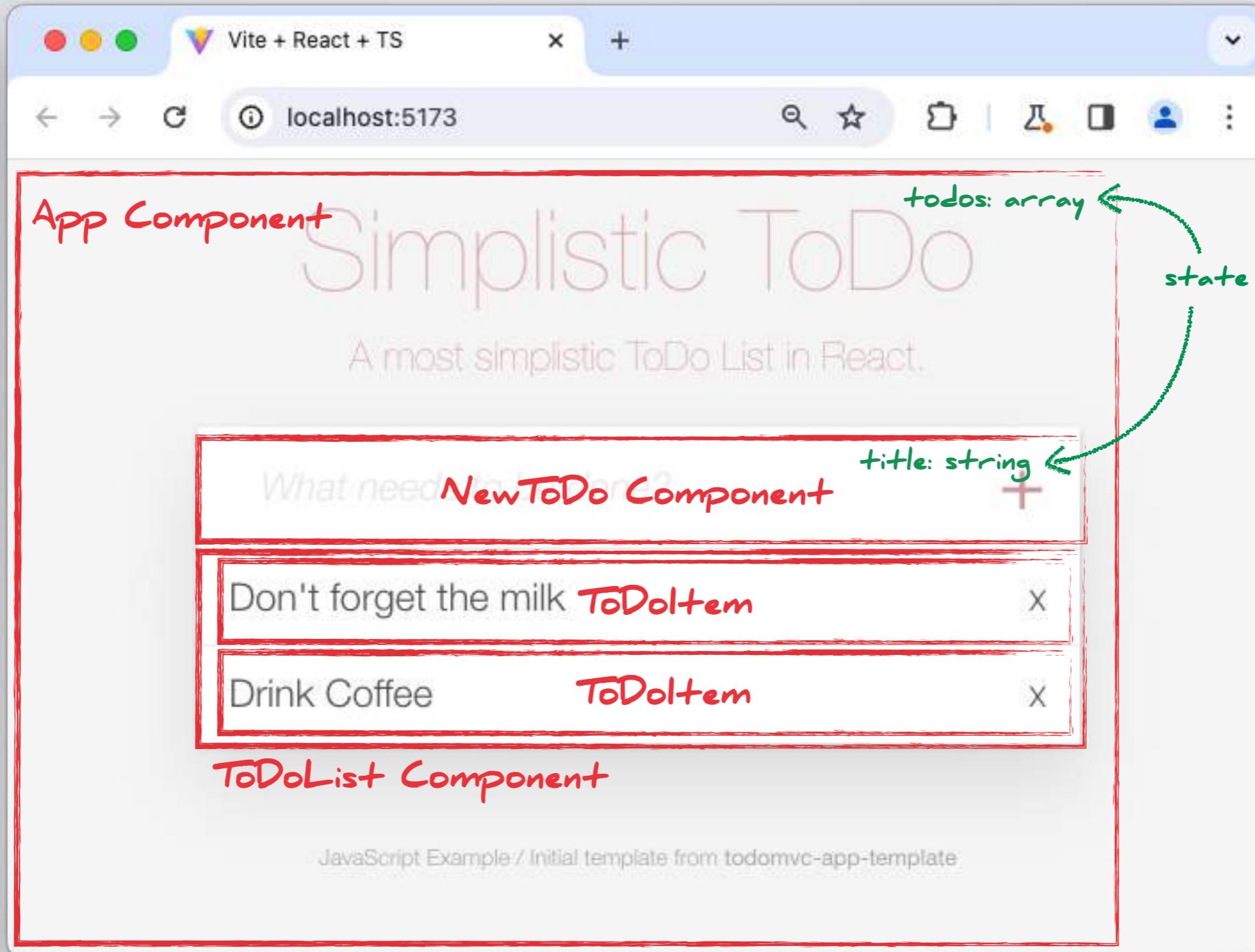
# EXERCISES



Exercise 4 - ToDo App Component  
Architecture

# The Component Tree

Designing the component tree and the corresponding structure of state.



# Client Side Routing



# Routing Libraries in the React Ecosystem

With client-side routing the browser URL and the rendered UI can change without making a request to the server.



React Router has been almost a "de-facto" standard for a long time.

<https://reactrouter.com/>

Routing libraries used to be simple and with a single concern. In recent years routing libraries evolved into powerful, but complicated "frameworks".



React Router 7 can still be used as a library, but it is also a "production grade framework".

<https://reactrouter.com/start/modes>

<https://www.youtube.com/watch?v=BKi4YwLaMBI>

# Router Alternatives



<https://reactrouter.com/>



Next.js has its own file-based router  
<https://nextjs.org/docs/app/building-your-application/routing>



## TanStack Router

<https://tanstack.com/router/>



Wouter: tiny router that relies on hooks

<https://github.com/molefrog/wouter>



Chicane: A safe router for React and TypeScript

<https://swan-io.github.io/chicane/>

Alternative Projects (but no recent activity 😞):

- Remix: has been merged into ReactRouter v7: <https://remix.run/blog/merging-remix-and-react-router>
- hookrouter: <https://github.com/Paratron/hookrouter>
- Type Route: <https://www.type-route.org/>
- navi: <https://frontarm.com/navi/en/>
- router 5: <https://router5.js.org/>
- ReachRouter was merged with React Router 6 <https://reach.tech/router/>



# TanStack Router

<https://tanstack.com/router/>

Reasons to choose TanStack Router:

- it is focusing on client-side scenarios.
- it is built from ground-up with type-safety
- it is well documented
- it has powerful features

<https://tanstack.com/router/latest/docs/framework/react/comparison>



# TanStack Router

<https://tanstack.com/router/>

TanStack Router code-based routing and file-based routing.

File-based routing is the recommended setup:

<https://tanstack.com/router/latest/docs/framework/react/decisions-on-dx>

File based routing uses a build plugin that generates code at development time.

Project Scaffolding:

```
npx create-tsrouter-app@latest my-app --template file-router
```

CLI options: <https://github.com/TanStack/create-tsrouter-app/tree/main/cli/create-tsrouter-app>

Manual Installation:

```
npm install @tanstack/react-router @tanstack/react-router-devtools
npm install -D @tanstack/router-plugin
```

Documentation:

<https://tanstack.com/router/latest/docs/framework/react/overview>



# TanStack Router

<https://tanstack.com/router/>

API:

```
import {  
  createRouter,  
  createRoute,  
  createRootRoute,  
  Outlet,  
  RouterProvider,  
  Link,  
  useLocation,  
  useParams  
} from '@tanstack/react-router'
```

*factory functions*

*components*

*hooks*

... and more.

Documentation:

<https://tanstack.com/router/latest/docs/framework/react/overview>



# TanStack Router

<https://tanstack.com/router/>

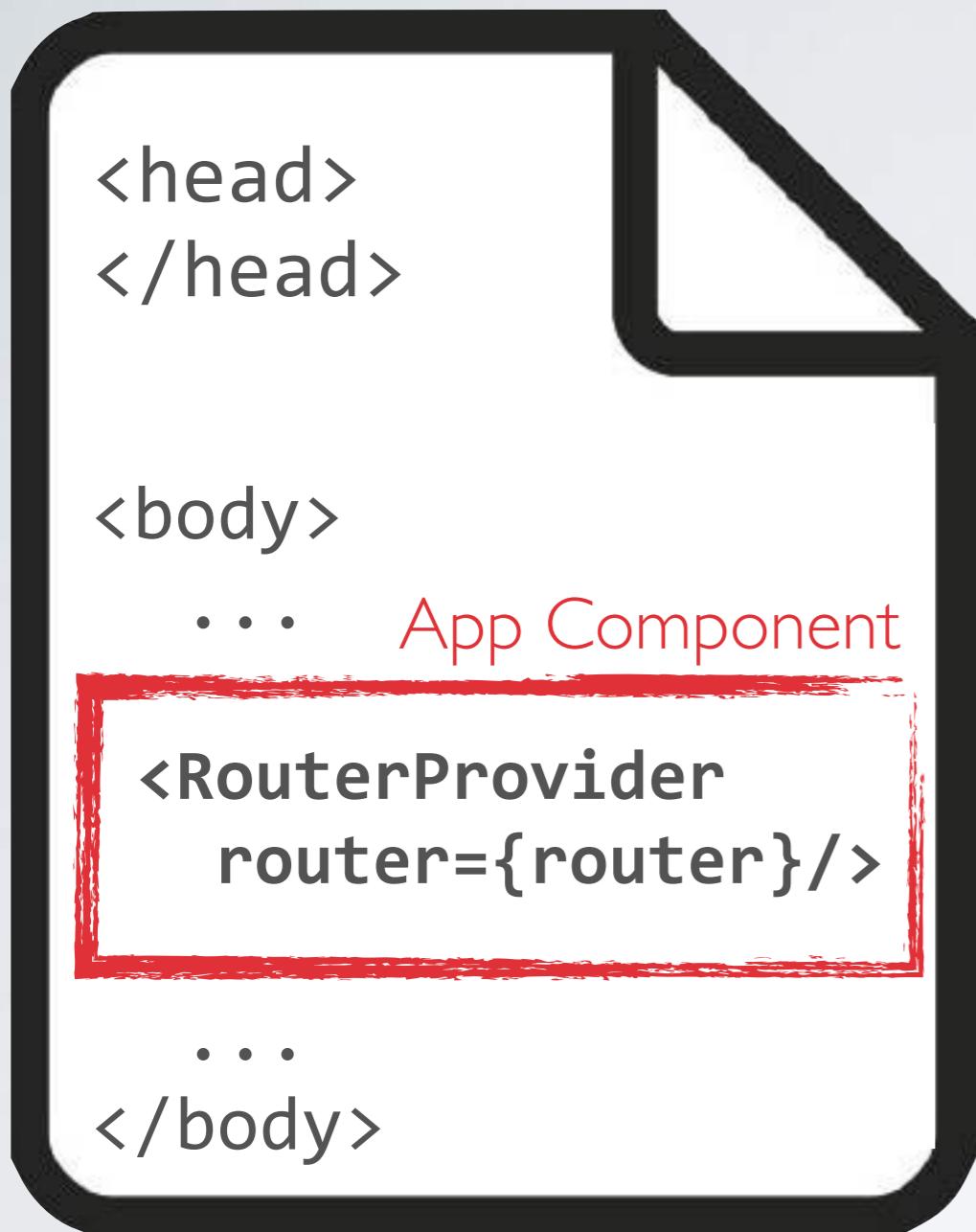
Many features beyond routing:

- data fetching
- url based state management
- query param validation
- dependency injection with router context

Documentation:

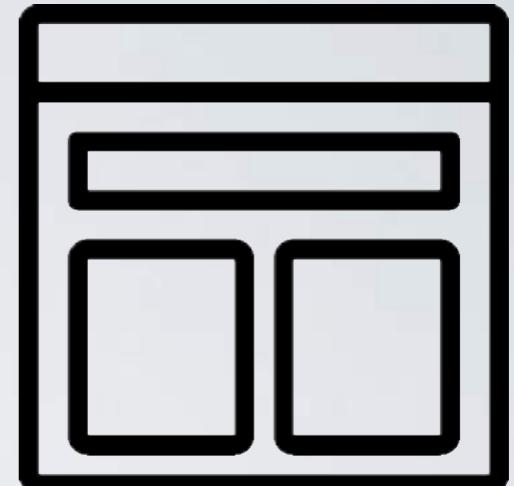
<https://tanstack.com/router/latest/docs/framework/react/overview>

http://localhost:8080/



SPA Shell

/path1



<ToDoScreen/>

/path2



<DoneScreen/>

```
const router = createBrowserRouter([
  {path: '/',
   element: <ToDoScreen/>},
  {path: '/done',
   element: <DoneScreen/>},
]);
```



# React Router

<https://reactrouter.com/>

React Router enables "client-side routing".

With client-side routing the browser URL and the rendered UI can change without making a request to the server.

Installation:

```
npm install react-router-dom
```

Documentation:

<https://reactrouter.com/en/main>

API:

```
import {  
  createBrowserRouter,  
  NavLink,  
  Outlet,  
  RouterProvider,  
  useNavigate,  
  useParams  
} from "react-router-dom";
```

factory

components

hooks

... and more.

# Router Configuration

```
const rootRoute = createRootRoute({
  component: <Layout/> ←
});
```

```
const todoRoute = createRoute({
  getParentRoute: () => rootRoute,
  path: '/',
  component: ToDoScreen
});
```

```
const routerTree = rootRoute.addChildren([todoRoute]);
const router = createRouter({ routeTree: rootRoute });
```

nested routing

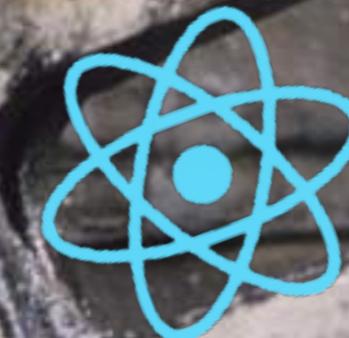
```
function App() {
  return (
    <RouterProvider router={router} />
  );
}
```

```
function Layout() {
  return (
    <div>
      <Link to="/">ToDo</Link>
      <Link to="/done">Done</Link>
      <Outlet /> ←
    </div>
  );
}
```

Dynamic styling of NavLink:

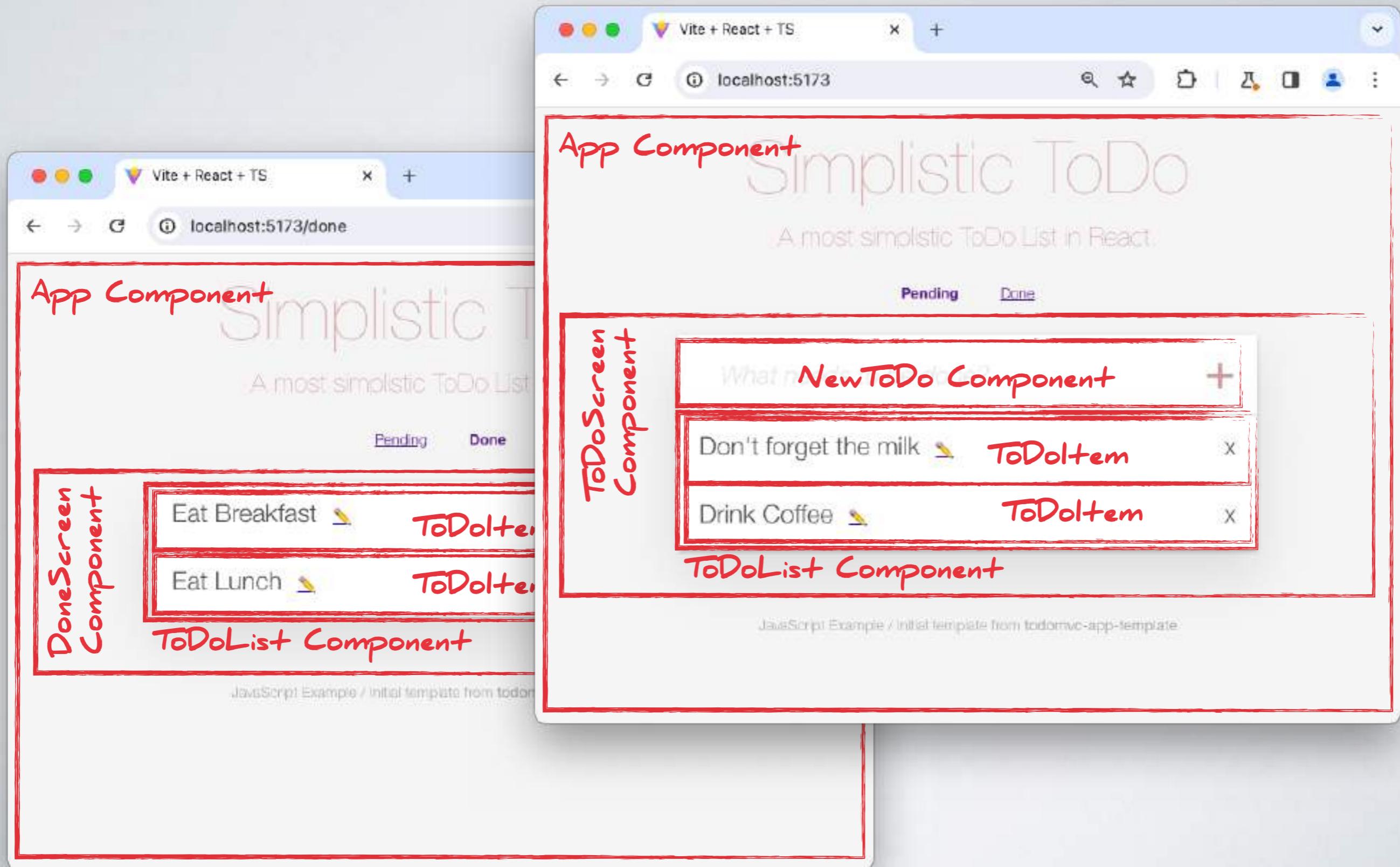
```
<Link to="/" activeProps={{ className: `active` }}>Pending</Link>
```

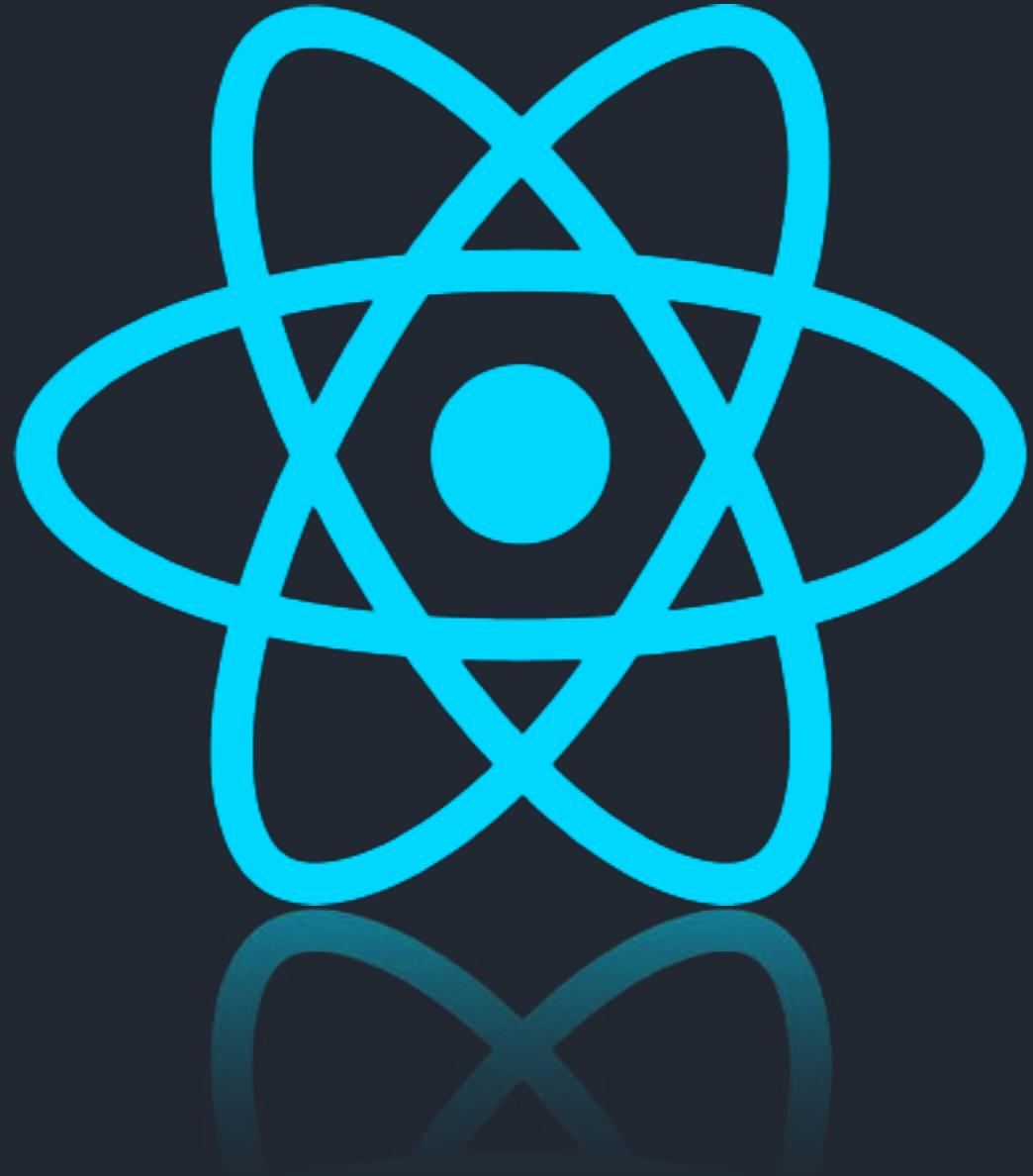
# EXERCISES



Exercise 5 - ToDo App Routing

# Routing





# Front-End Development with React

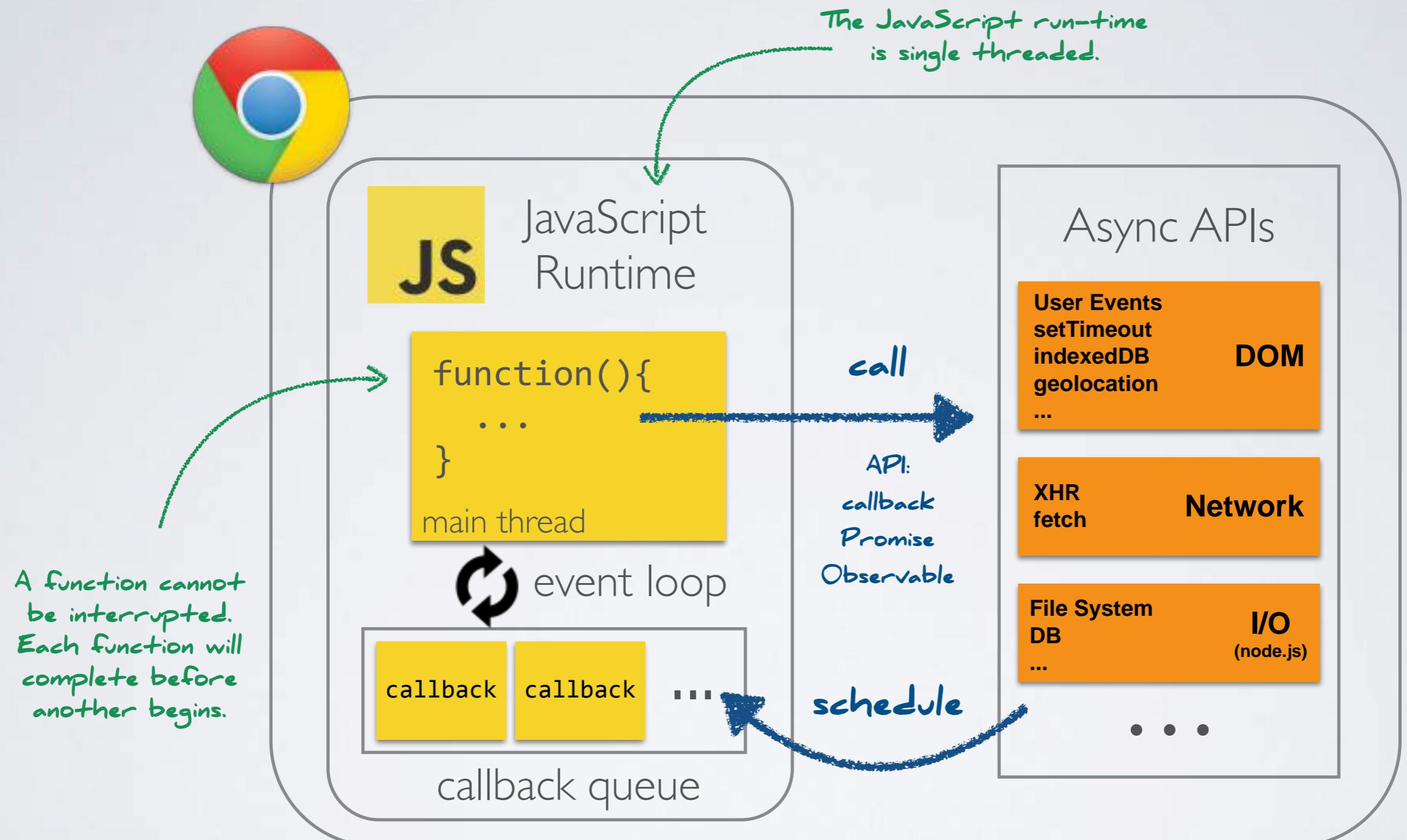
Mail: [jonas.band@ivorycode.com](mailto:jonas.band@ivorycode.com)

Twitter: [@jbandi](https://twitter.com/jbandi)



Side-Track: Async JavaScript

# Concurrency in JavaScript: Event Loop



<https://www.tedinski.com/2018/10/16/concurrency-vs-parallelism.html>

What the heck is the event loop anyway? - <https://www.youtube.com/watch?v=8aGhZQkoFbQ>

Interactive Visualization: <http://latentflip.com/loupe/>

Jake Archibald: In The Loop: <https://www.youtube.com/watch?v=cCOL7MC4PIO>

<https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/>

# Side-Track: Async JavaScript

There are different models/APIs for concurrency in JavaScript.

Callbacks → "Pyramid of Doom"



Observables → RxJS 

# XMLHttpRequest

The underlying technology for AJAX.

```
const req = new XMLHttpRequest();
req.addEventListener("load", done);
req.addEventListener("error", failed);
req.addEventListener("readystatechange", stateChanged);
req.open("GET", "https://jsonplaceholder.typicode.com/albums");
req.setRequestHeader("Content-Type", "application/json");
req.send(JSON.stringify({title: 'Learn the Fetch API', completed: false}));

function stateChanged() {
    if (this.readyState === 4) {
        if (this.status === 201) {
            console.log('success state', this.responseText)
        }
    } else {
        console.log('error state', this.statusText);
    }
}

function done() {
    console.log('DONE', this.responseText);
}

function failed() {
    console.log('ERROR', this.statusText);
}
```

The API is low-level, complicated and verbose.

# AJAX with Callbacks

using jQuery

```
$.get('http://localhost:3001/comments', (data) => {
  console.log(data);
});
```

```
$.post('http://localhost:3001/comments',
  {text: 'test - ' + new Date()},
  () => {
    console.log('POST!');
});
```

# fetch: a modern AJAX API

fetch is a new API available in modern browsers that is more elegant than XMLHttpRequest.

```
fetch('https://swapi.co/api/people/1/')
  .then(response => response.json())
  .then(data => console.log('SUCCESS', data))
  .catch(error => console.log('ERROR:', error));
```

fetch is available in all modern browsers.

fetch is built on Promises.

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)

<https://fetch.spec.whatwg.org/#fetch-api>

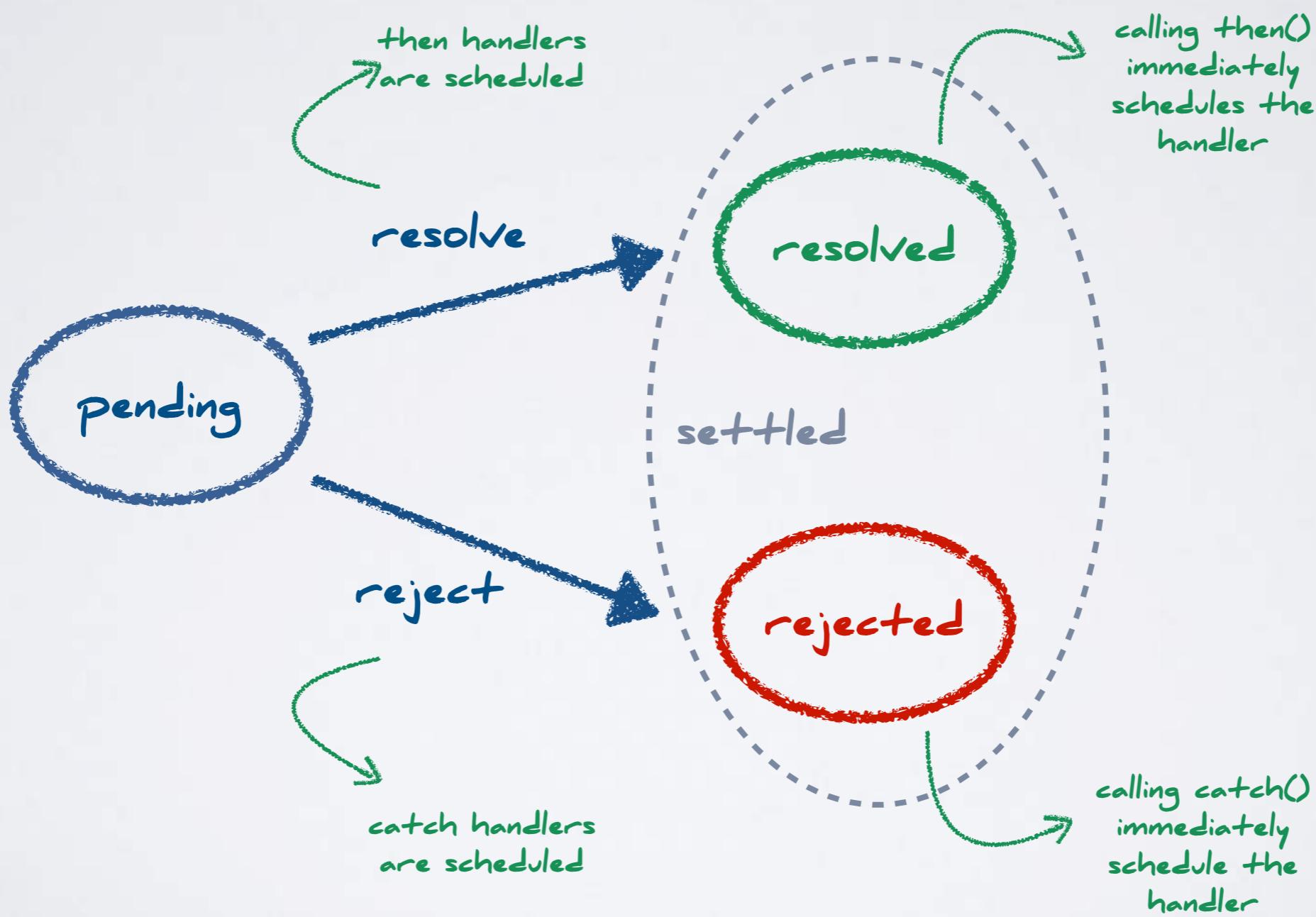
<https://jakearchibald.com/2015/thats-so-fetch/>

Polyfill: <https://github.com/github/fetch>

Browser support:

<http://caniuse.com/#search=fetch>

# Promises represent a Statemachine



# Advantages of Promises: Chaining Async Operations

Callback-Hell:  
(aka Pyramid of Doom)

```
api(function(result){  
    api2(function(result2){  
        api3(function(result3){  
            // do work  
        });  
    });  
});
```

Promise Chaining:  
If a "then-callback" returns  
another promise, the next  
"then" waits until it is settled.

```
api().then(function(result){  
    return api2();  
}).then(function(result2){  
    return api3();  
}).then(function(result3){  
    // do work  
}).catch(function(error) {  
    //handle any error  
});
```

```
api().then(api2)  
    .then(api3)  
    .then(/* do work */);
```

# Promise Chaining

→ t

```
doSomething().then(function () {  
    return doSomethingElse();  
}).then(finalHandler);
```

```
doSomething().then(function () {  
    doSomethingElse();  
}).then(finalHandler);
```

```
doSomething()  
.then(doSomethingElse())  
.then(finalHandler);
```

```
doSomething()  
.then(doSomethingElse)  
.then(finalHandler);
```

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

# Evolution of ECMAScript Promises

ECMAScript 2015:

Construction:

- `new Promise((resolve, reject) => {})`
- `Promise.resolve()`, `Promise.reject()`

Combinators:

- `Promise.all()`
- `Promise.race()`

Handling/Flow:

- `Promise.prototype.then()`
- `Promise.prototype.catch()`

DOM API: Abort Controller (2017) -> can be used to cancel a Promise API

ECMAScript 2018:

`Promise.prototype.finally()`

ECMAScript 2020:

`Promise.allSettled()`

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise/allSettled](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/allSettled)

ECMAScript 2021:

`Promise.any()`

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise/any](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/any)

# async / await

async/await provides convenient language support  
on top of any Promise based API.

```
try {  
    const response = await axios.get(API_URL);  
    console.log(response);  
} catch (error) {  
    console.log(error)  
}
```

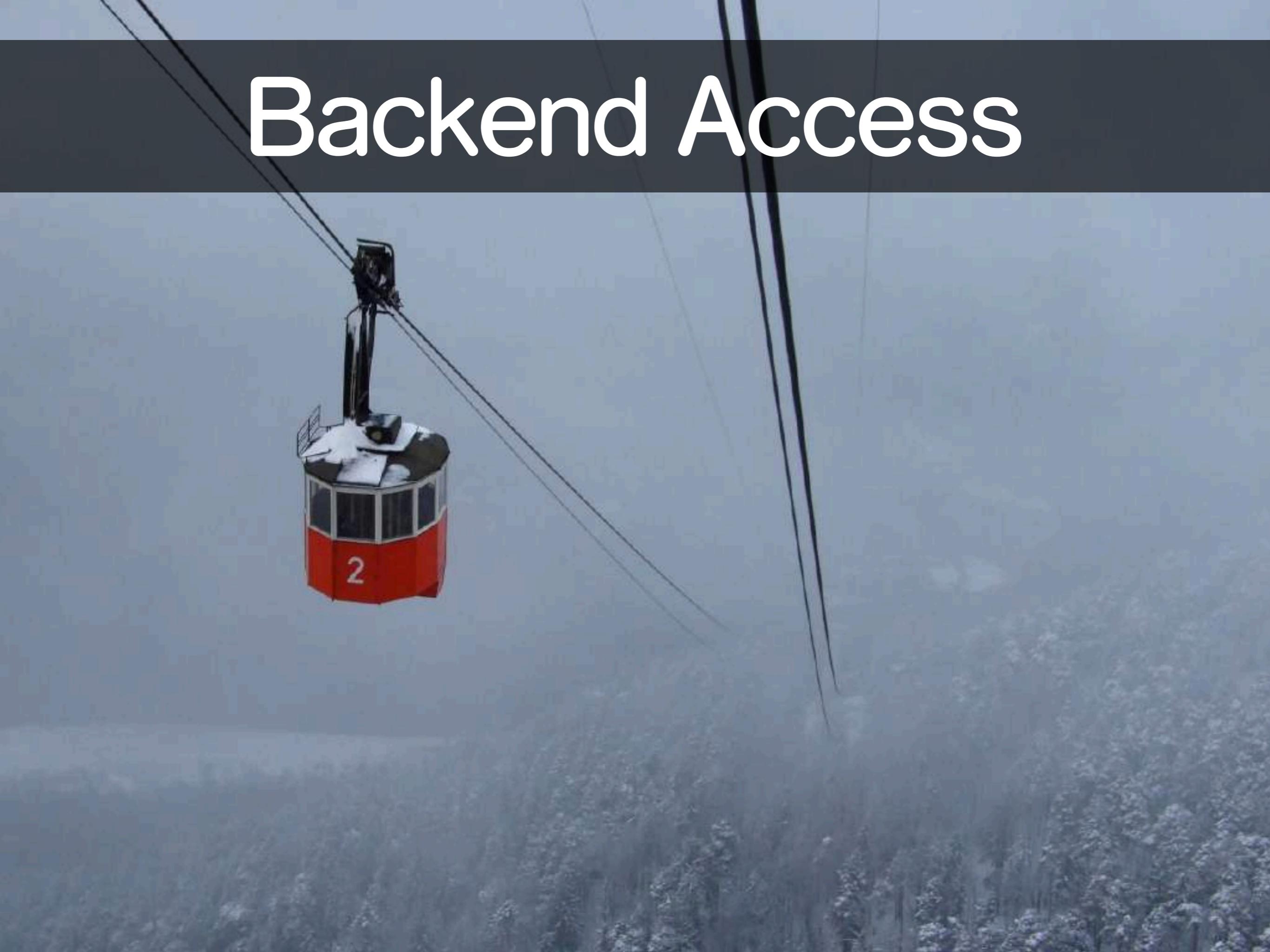
```
const response = await axios.post(API_URL, payload);  
console.log(response);
```

```
const response = await axios.delete(` ${API_URL}/${id}`);  
console.log(response);
```

Note: **await** can only be used inside an **async** function

(or on top level of modules, see [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await#top\\_level\\_await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await#top_level_await) )

# Backend Access



# Promise based Backend Access: **fetch** vs. **axios** vs. **ky**

**fetch** is an browser API that is more elegant than XMLHttpRequest.

```
const res = await fetch('https://swapi.co/api/people/1/');  
if (!res.ok) { ←  
    throw new Error('Network response was not ok');  
} else {  
    const data = await res.json();  
    console.log('SUCCESS', data);  
}
```

non-2XX status is  
NOT an error!

**fetch** is available in all modern browsers (<http://caniuse.com/#search=fetch>).

For many projects, it makes sense to use a http library with more features:

- **ky**: <https://github.com/sindresorhus/ky> (modern, based on fetch)
- **axios**: <https://github.com/axios/axios> (legacy, based on XHR)

```
import ky from 'ky';  
const response = await ky.get('http://www.example-api.com');  
const data = await response.json<ResponseType>();
```

# Backend Access: **ky**

**ky** is a promise based HTTP client library for the browser and Node.js

```
npm i ky
```

```
import ky from 'ky';
```

```
try {
  const response = await ky.get(API_URL);
  const data = await ky.json<GetResponseType>()
  console.log(response);
} catch (error) {
  console.log(error)
}
```

```
const response = await ky.post(API_URL, {json: payload});
...
```

```
const response = await ky.put(` ${API_URL}/${id}` , {json: payload});
...
```

```
const response = await ky.delete(` ${API_URL}/${id}` );
...
```

# Pragmatic Data Fetching with `useEffect`

There is no dedicated mechanism for data fetching in client-side React. It is a common practice to use `useEffect` for data fetching.

```
function MyComponent() {
  const [data, setData] = useState(undefined);

  useEffect(() => {
    async function loadData(){
      const data = await fetch(...);
      setData(data);
    }
    loadData();
  });

  return <div>{data.toString()}</div>
}
```

effect function  
can't be async!

missing error handling!

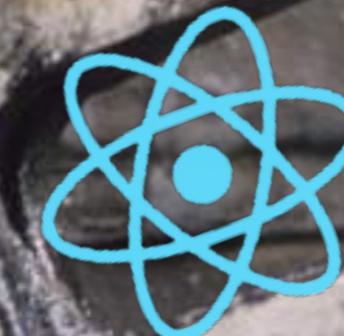
potential race condition if  
the effect is outdated!

Tanstack Query is a widely used declarative data fetching library: <https://tanstack.com/query/>

React Server Components provide a mechanism for data fetching in full-stack React:  
<https://nextjs.org/docs/app/building-your-application/data-fetching/patterns>

Support for promises in React client components  
might provide other patterns in the future:  
<https://react.dev/reference/react/use>

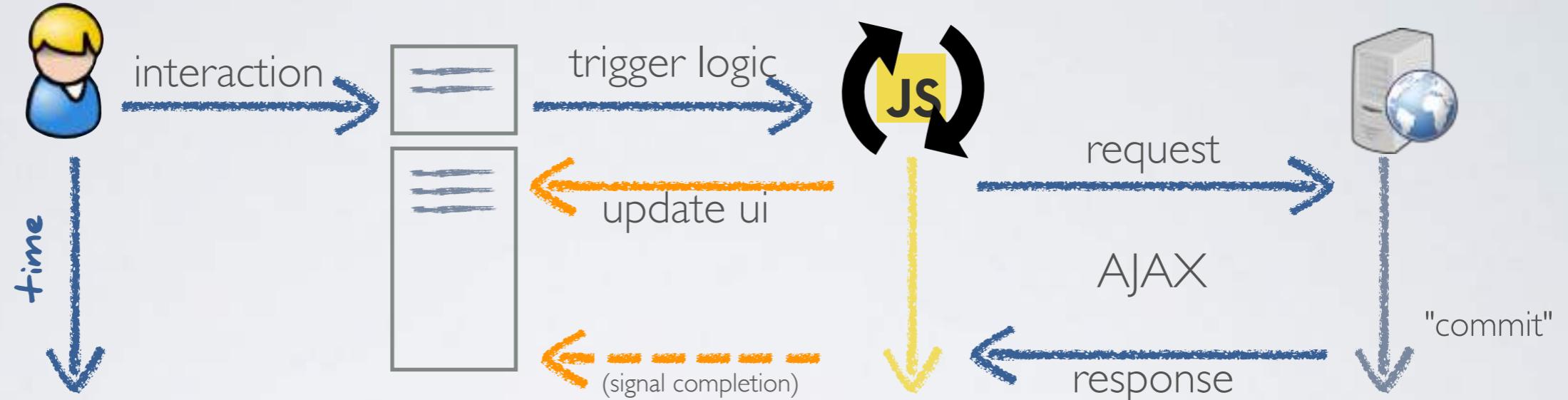
# EXERCISES



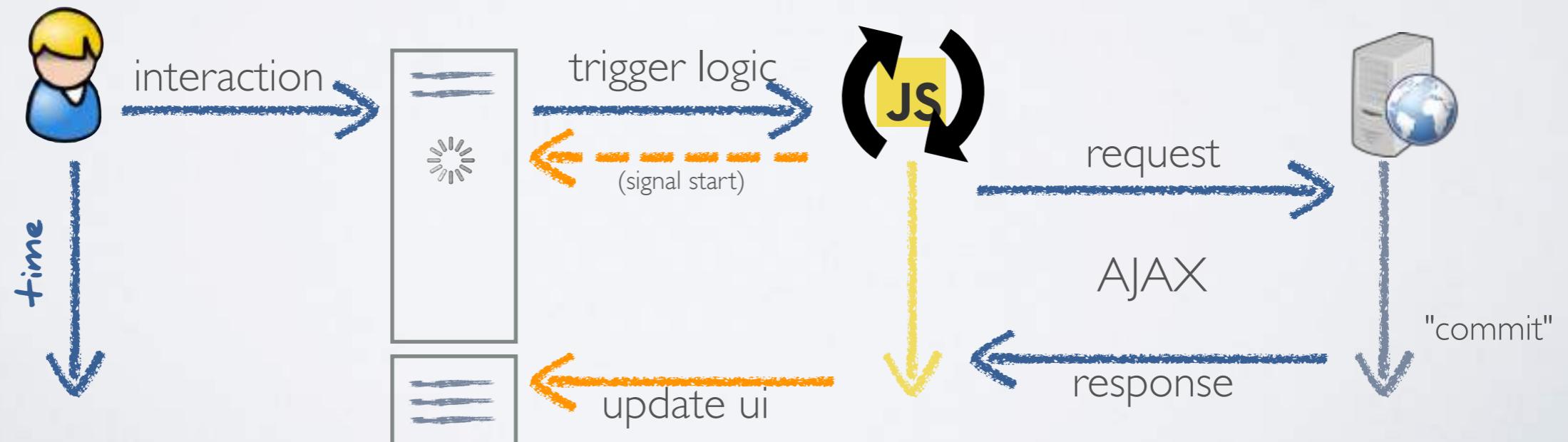
Exercise 8 - BackendAccess

# Mutations: Optimistic vs. Pessimistic UI

Optimistic UI:



Pessimistic UI:



# Mutating Data

Pessimistic UI



Optimistic UI

```
await triggerMutation(change);
const newState = await refetchState();
setState(newState)
```

```
const mutationResponse = await triggerMutation(change);
const newState = merge(state, mutationResponse);
setState(newState)
```

```
const newState = merge(state, change);
setState(newState)
const mutationResponse = await triggerMutation(change);
const updatedState = merge(state, mutationResponse);
setState(updatedState)
```

# Proper Data Fetching with useEffect

<https://react.dev/learn/synchronizing-with-effects#fetching-data>

```
useEffect(() => {
  let ignore = false;
  effect function can't be async!
```

```
async function startFetching() {
  try {
    const json = await fetchTodos(userId);
    if (!ignore) {
      setTodos(json);
    }
  } catch (e: any) {
    setError('An error occurred: ' + e.getMessage());
  }
}
error handling
```

```
startFetching();
```

```
return () => {
  ignore = true;
};
cleanup function:
- ignore the result
- alternative: abort the fetch
, [userId]);
```

avoiding race conditions by ignoring "outdated" fetches

potential for further improvement:

- caching / deduping
- loading state

Proper data fetching verbose and complicated:

- consider using a library
- encapsulate the logic into custom hooks

<https://react.dev/learn/synchronizing-with-effects#what-are-good-alternatives-to-data-fetching-in-effects>

# Custom Hooks for Data Fetching

```
function useUsers() {  
  
  const [data, setData] = useState(initialValue);  
  const [error, setError] = useState(null);  
  const [loading, setLoading] = useState(true);  
  
  useEffect(() => {  
    let cancelled = false;  
    (async () => {  
      try {  
        const res = await fetch('https://jsonplaceholder.typicode.com/users');  
        const resJson = await res.json();  
        if (!cancelled) setData(resJson);  
      } catch (err) {  
        if (!cancelled) setError(err);  
      } finally {  
        if (!cancelled) setLoading(false);  
      }  
    })();  
    return () => { cancelled = true }  
  }  
  
  return {loading, data, error};  
}
```

```
function AppComponent() {  
  const {loading, data, error} = useUsers();  
  
  if (loading) return <Spinner/>;  
  if (error) return <>...</>;  
  ...  
  return <>...</>  
}
```

custom Hook

Consider using **axios-hooks**, **use-http** or **TanStack Query** or **SWR** instead of implementing the low-level fetch.

<https://github.com/simoneb/axios-hooks>

<https://github.com/ava/use-http>

<https://swr.vercel.app/>

<https://tanstack.com/query>

# Data Fetching Libraries

If you are using "plain" axios or fetch, then you have to care about cancellation, loading & error state, caching, deduplication, refetching ...

Consider a data-fetching library:

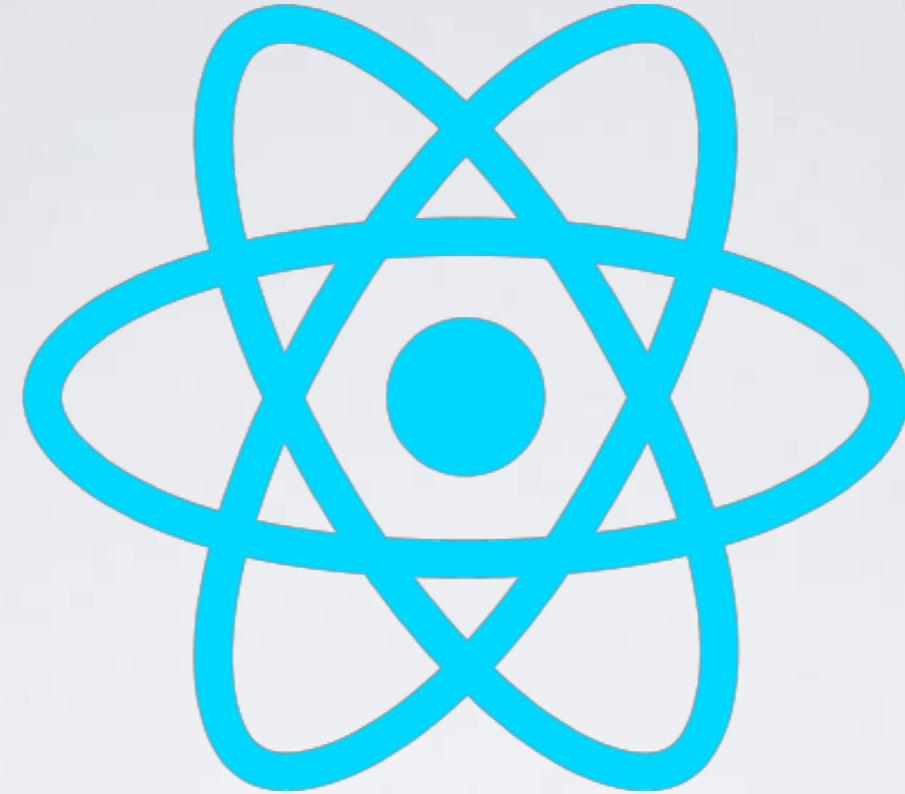
- **TanStack Query** <https://tanstack.com/query/>
- **SWR** <https://github.com/vercel/swr>
- RTK Query (only if already using Redux): <https://redux-toolkit.js.org/rtk-query/overview>

Consider data loading and mutation via the router:

-  ReactRouter: <https://reactrouter.com/en/main/start/overview#data-loading>
-  **TanStack Router** <https://tanstack.com/router/latest/docs/framework/react/guide/data-loading>

... or a more-lightweight hook

- use-http: <https://use-http.com>
- axios-hooks: <https://github.com/simoneb/axios-hooks>



# Have Fun with React!



JavaScript / Angular / React / Vue / Vaadin  
Schulung / Beratung / Coaching / Reviews  
[jonas.band@ivorycode.com](mailto:jonas.band@ivorycode.com)