CDN & React CDN

JavaScript was made specially for browsers.
So every browser has a JavaScript engine inside it.

Examples:

- Chrome → V8 engine
- Firefox → SpiderMonkey
- Safari → JavaScriptCore

These engines run JavaScript directly.

Browsers understand only plain JavaScript, HTML, and CSS — not JSX.

Browser doesn't understand React because JSX is not built into the browser.

What is CDN?

CDN = Content Delivery Network

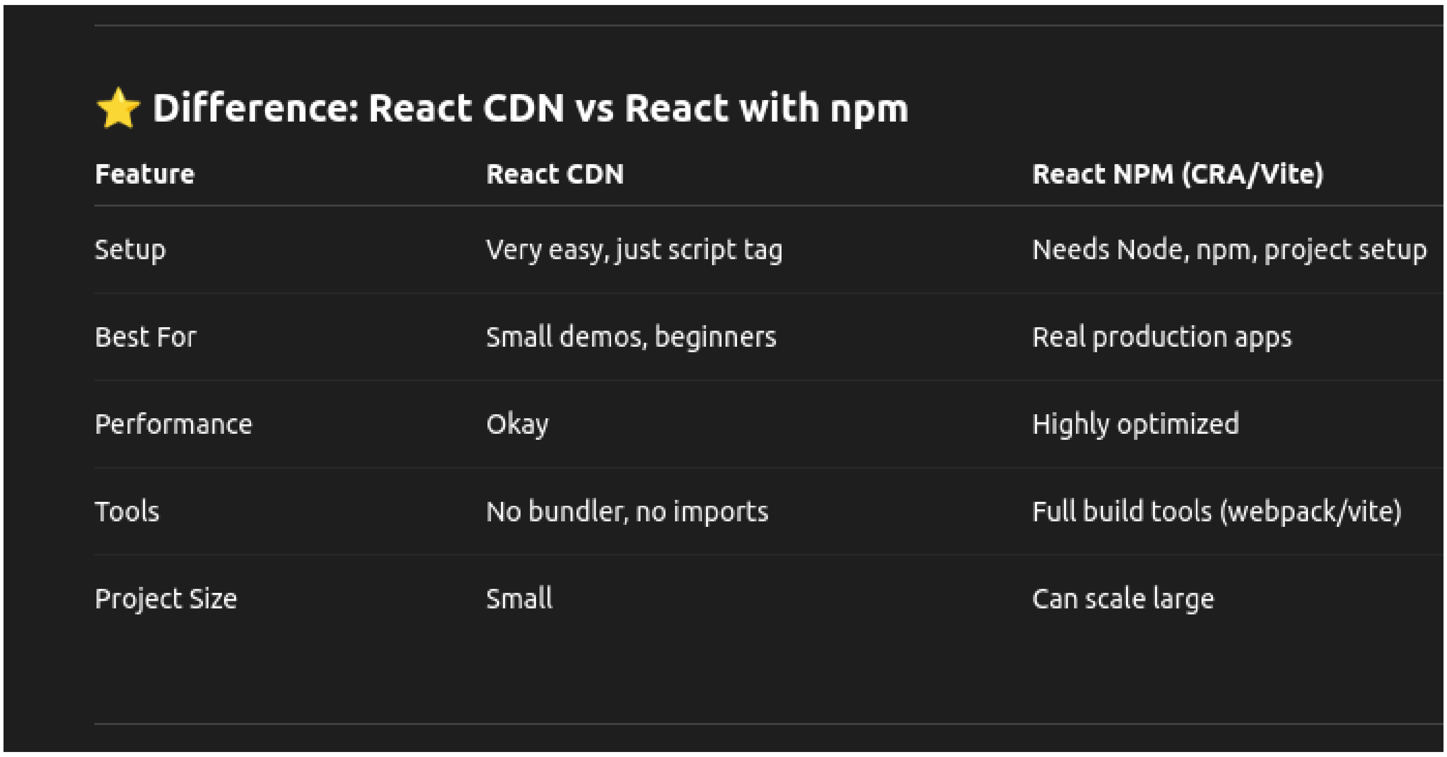
A collection of global servers that deliver files **quickly** to users.

Files stored on CDN:

- JavaScript files
- CSS files
- Images
- Libraries (React, jQuery, Bootstrap, etc.)

What is React CDN?

  - React CDN means you load React library using **script tags** directly from a CDN provider
  - instead of installing it via npm.



⭐ Difference: React CDN vs React with npm

Feature	React CDN	React NPM (CRA/Vite)
Setup	Very easy, just script tag	Needs Node, npm, project setup
Best For	Small demos, beginners	Real production apps
Performance	Okay	Highly optimized
Tools	No bundler, no imports	Full build tools (webpack/vite)
Project Size	Small	Can scale large

WHY TWO CDN links?

```
<script src="https://unpkg.com/react@18/umd/react.development.js"></script>
```

```
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
```

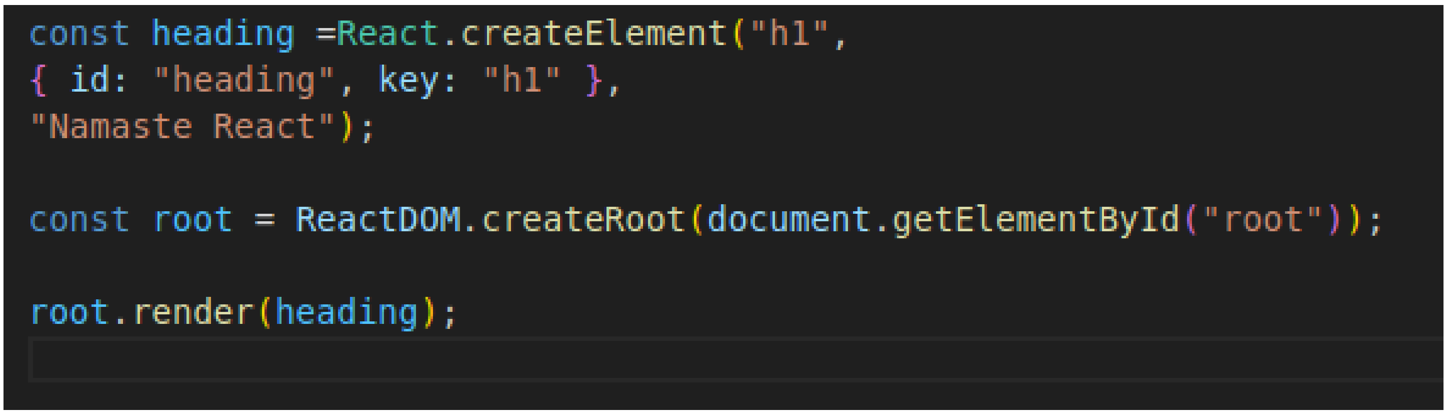
- **React** = core library (components, state, logic).
- **ReactDOM** = renders React into the browser.

They are separate because:

1. React works on **many platforms** (web, mobile, TV, VR).
2. ReactDOM works **only for browsers**.
3. Keeping them separate makes the bundle **smaller, faster, and easier to maintain**.
4. Each can be **updated independently**.

Creating element(heading)in React

REACT Element(Object) ===== Browser understand(Html)



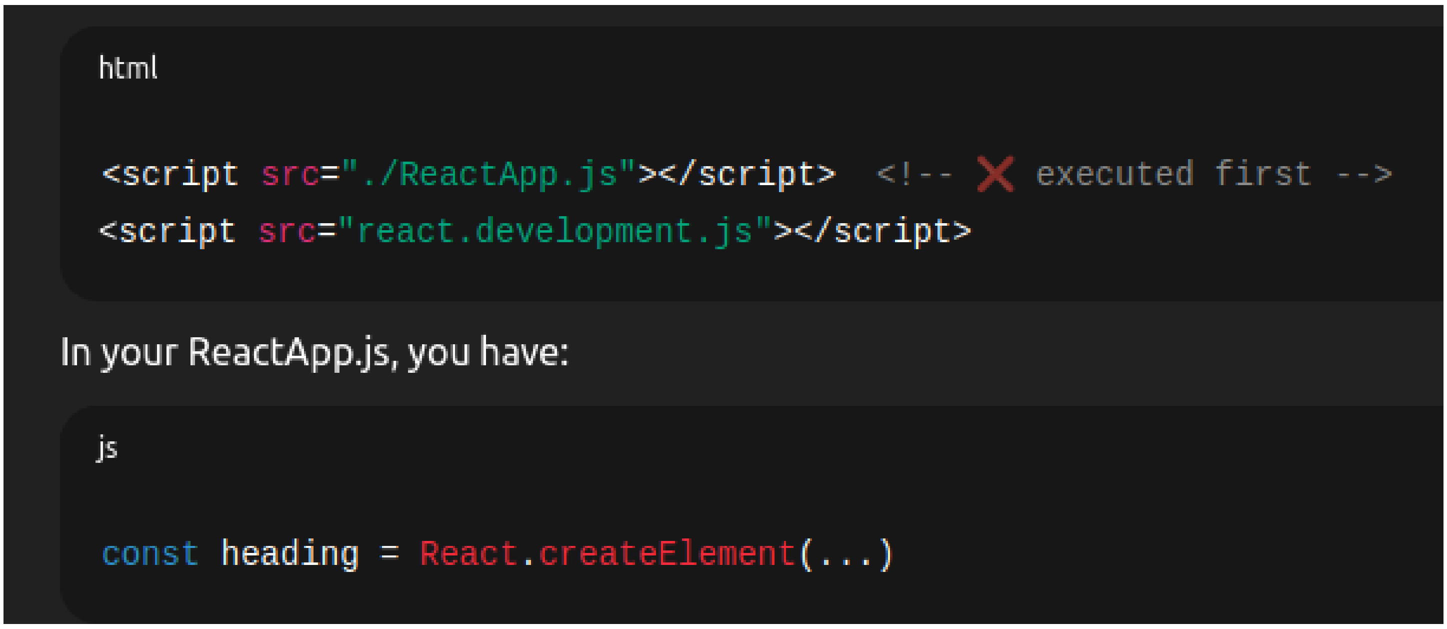
```
const heading =React.createElement("h1",
{ id: "heading", key: "h1" },
"Namaste React");

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(heading);
```

heading ------------->object

root.render convert

object ---------→ tag (heading)

What happens if we write YOUR script before React CDN



html

```html
<script src="./ReactApp.js"></script>  <!-- ❌ executed first -->
<script src="react.development.js"></script>
```

In your ReactApp.js, you have:

js

```js
const heading = React.createElement(...)
```

But React is not loaded yet, so:

🔥 **Error:**
 Uncaught ReferenceError: React is not defined

Browser can’t find React, so your app fails.

Why crossorigin is used in React CDN scripts

We add crossorigin because React is coming from another website (a different origin). crossorigin tells the browser to accept it and show proper error messages.

Your HTML file is in one place

Your computer or localhost
http://localhost:3000

React CDN file is in another place

https://unpkg.com/react

These TWO places are **different**.

So when your browser tries to get React from another house (another website), it becomes a **cross-origin request**.

🔥 Think of it like this:

You borrowed **salt** from your neighbor.

Your mom will ask:

"Who gave this? Should I trust them?"

That **trust check** = crossorigin

Why <script> tags are placed at the end of <body>

If script loads **before** HTML, then:

document.getElementById("root")

returns null, because DOM is not created yet.

✓ Therefore:

We place scripts at the end of <body>, so:

- HTML loads first
- <div id="root"> exists
- React can attach successfully

why react called library not framework

A **framework controls you**,
A **library is controlled by you**.

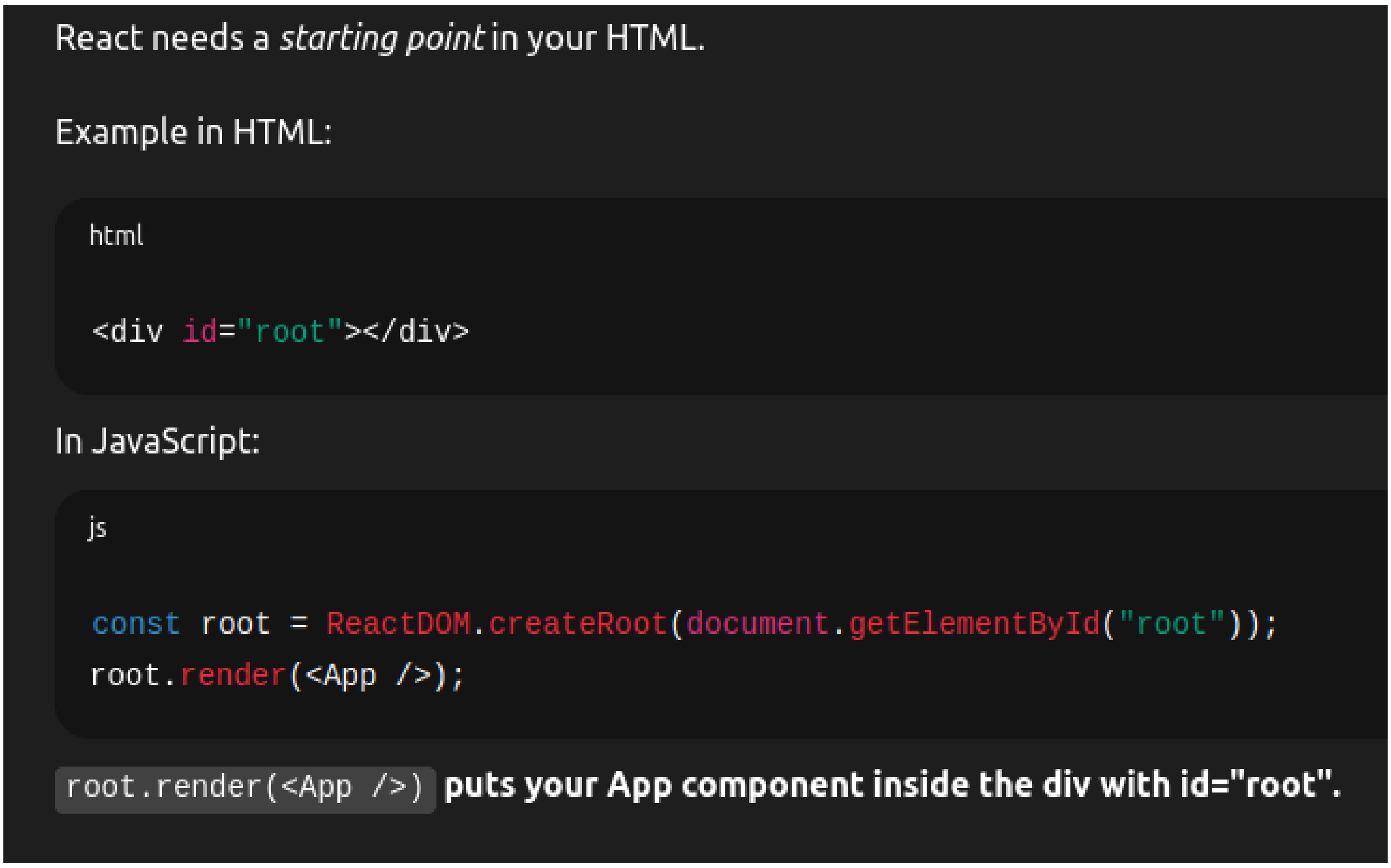
In React, we choose tools for routing, state management, and architecture ourselves. Frameworks provide all of these built-in and control the entire flow. React is flexible and developer-controlled, which is why it's a library

What does root.render() do

root.render() tells React to **display your React component inside the browser's HTML page**.

It connects:

✓ The **React world** (JS, components, virtual DOM)
to
✓ The **HTML world** (real DOM in the browser)


React needs a *starting point* in your HTML.

Example in HTML:

```html
<div id="root"></div>
```

In JavaScript:

```js
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

`root.render(<App />)` **puts your App component inside the div with id="root".**

Why do we need this?

React cannot directly modify HTML.
React updates a **virtual DOM**, and then syncs it with the browser DOM.

root.render() is the bridge between:

- React UI → Virtual DOM
- Browser → Real DOM

Why only one root div?

React apps are usually **Single Page Applications (SPA)**.
All UI updates happen inside one place:

`<div id="root"></div>`

What happens internally?

`root.render(<App />);`

React:

1. Creates a virtual DOM of <App />
2. Compares with the existing UI
3. Writes the new UI inside <div id="root">
4. Controls everything inside that root element