A WORLD WITHOUT BUILDS

CascadiaJS 2025


♥ CascadiaJS

nearform.com/open-source

Hi!

👋 I'm Ryan Roemer

 I lead technology & OSS at Nearform

💼 I **like** & **dislike** builds

Let's look at some history...

- **1995**: We met JavaScript.
- **1995-2001**: We wrote JavaScript that actually executed in the browser


And it looked like this…

Legacy - movies.js

```
1 // ⚠️ Have to add properties to global `this`
2 this.movies = [
3   { name: "The Princess Bride", year: 1987 },
4   { name: "The Emperor's New Groove", year: 2000 },
5 ];
```

Legacy - index.html 🔗

```
<script src="movies.js">
  /* ⚠️ attached to global `this` as `movies` */
</script>
<script>
  // 🕰️ ... ye old `var`s and `+` string concatenation!
  var html = "";
  for (var i = 0; i < movies.length; i++) {
    html += "<p>" + movies[i].name + " (" + movies[i].year + ")</p>";
  }

  var root = document.getElementById("root");
  root.innerHTML = html;
</script>
```

And then …

- **2001**: A **minifier**! (*JSMin*)
- **2010-2015**: **Bundlers**!
  - *RequireJS*, *Browserify*, *Webpack*, …
- **2015-on**: **Fancier bundlers**
  - *Rollup*, *Parcel*, *Vite*, …

And now…

A modern simple stack usually entails at least a **bundler**, a **transpiler**, and a **style library**.

We run **minified, heavily transformed** code.

We no longer *run* the code we *write*.


And that's OK!

But...

How far could we go without a build?

Spoilers

- This is just a timeline and greatest hits
- Skipping over a ton of technical detail
- Most large scale web apps probably still **need builds**
- We cheat (✨)

The Interesting Parts

- What works **natively**?
  - No webpack/vite, no TypeScript/JSX/Babel, ...
  - Just **pure JavaScript** running in a browser?
- When can we push the build to **runtime** (e.g. CDN)?
- What can you actually get away with (e.g. Web Components, React)?

Another timeline

The road to buildless...

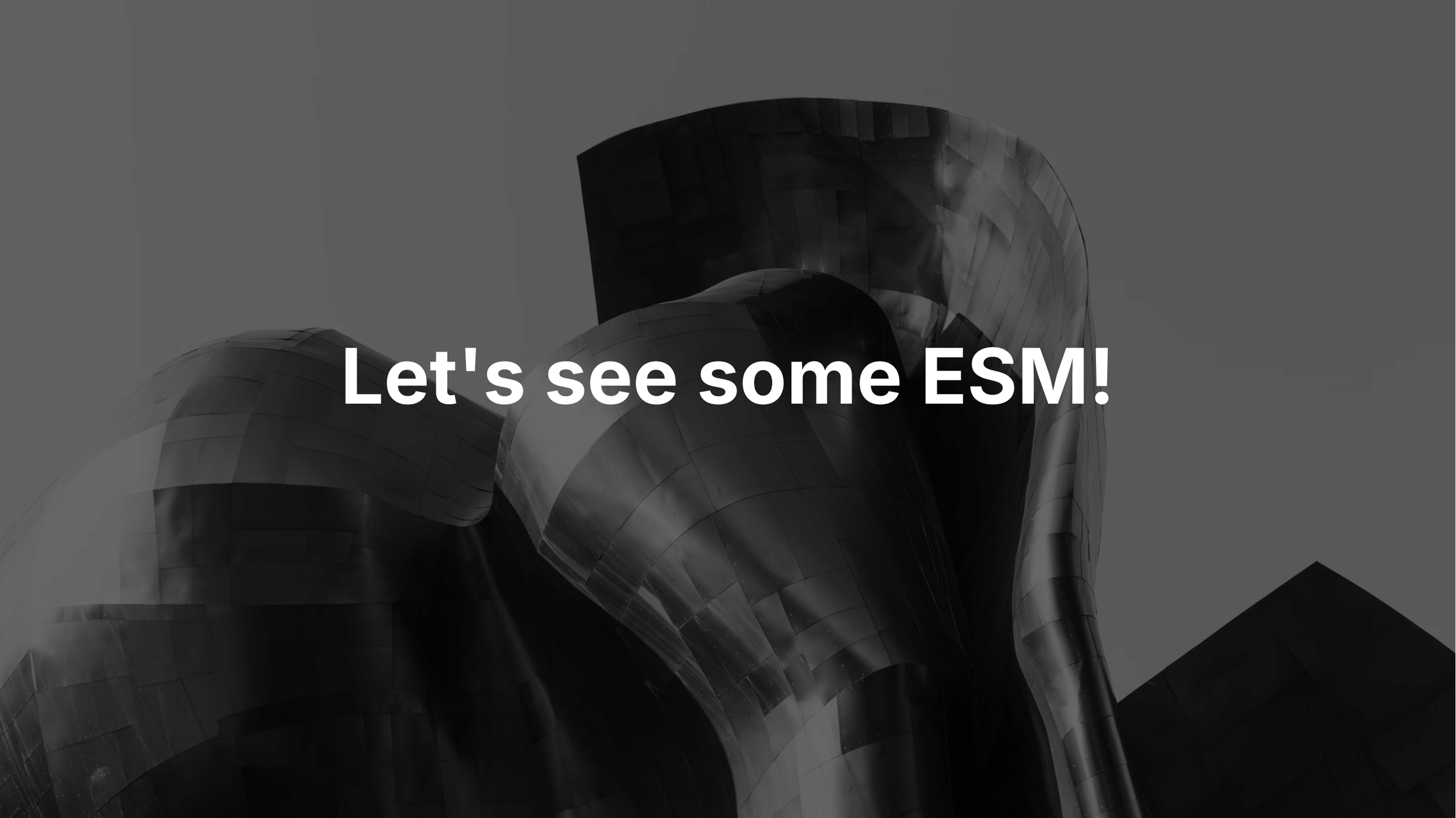
2015: ECMAScript Modules (ESM)

Named exports

```
export const movies = [ ... ];
```

Static and dynamic imports

```
// Static import
import { movies } from "./movies.js";

// Dynamic import
const { movies } = await import("./movies.js");
```


Let's see some ESM!

Modern - movies.js

```
// 💡 first class ESM export
export const movies = [
  { name: "Moana", year: 2016 },
  { name: "The Mitchells vs. The Machines", year: 2021 },
  { name: "KPop Demon Hunters", year: 2025 },
];
```

Modern - index.html 🔗

```html
<script type="module">
  // 💡 first class ESM import (static or dynamic).
  // ⚠️ easy to import from relative paths, but what about bare names?
  import { movies } from "./movies.js";

  const root = document.getElementById("root");
  root.innerHTML = movies
    .map((m) => `<p>${m.name} (${m.year})</p>`)
    .join("");
</script>
```


Yay!


... but then we hit all the hard parts

2018: Import Maps

The problem: Can't import `"lodash"` (or any dep!)

```
import lodash from "lodash"; // ⚠️ Doesn't work
```

The solution: map a package name to a URL

```
<script type="importmap">{
  "imports": {
      "lodash": "https://esm.sh/lodash@^4.17.21"
  }
}</script>
```

2018-2021: Import Maps

- **2018**: Another problem - not actually **shipped**
- **2018**: `es-module-shims` (✨) to the rescue!
  - **Rewriting** and **polyfilling** not-yet-supported features
- **2021**: Finally, broadly **shipped**

2017+: CDNs

The problem: `react` only ships CommonJS.

The solution: CDNs get more savvy with lazy ESM conversion (✨).

Enter `esm.sh`, `jspm.io`, `unpkg.com`, ...


Let's ship a React app!

React - movies.js 🔗

```
// 💡 same as before
export const movies = [
  { name: "Moana", year: 2016 },
  { name: "The Mitchells vs. The Machines", year: 2021 },
  { name: "KPop Demon Hunters", year: 2025 },
];
```

React (1) - index.html 🔗

```html
<!-- 💡 The magic -- map bare names to URLs -->
<!-- ✨ React now has ESM exports! -->
<script type="importmap">
  {
    "imports": {
      "htm": "https://esm.sh/htm@^3?external=react,react-dom",
      "react": "https://esm.sh/react@18.2.0?",
      "react/jsx-runtime": "https://esm.sh/react@18.2.0/jsx-runtime?",
      "react-dom": "https://esm.sh/react-dom@18.2.0?",
      "react-dom/client": "https://esm.sh/react-dom@18.2.0/client?"
    }
  }
</script>
```

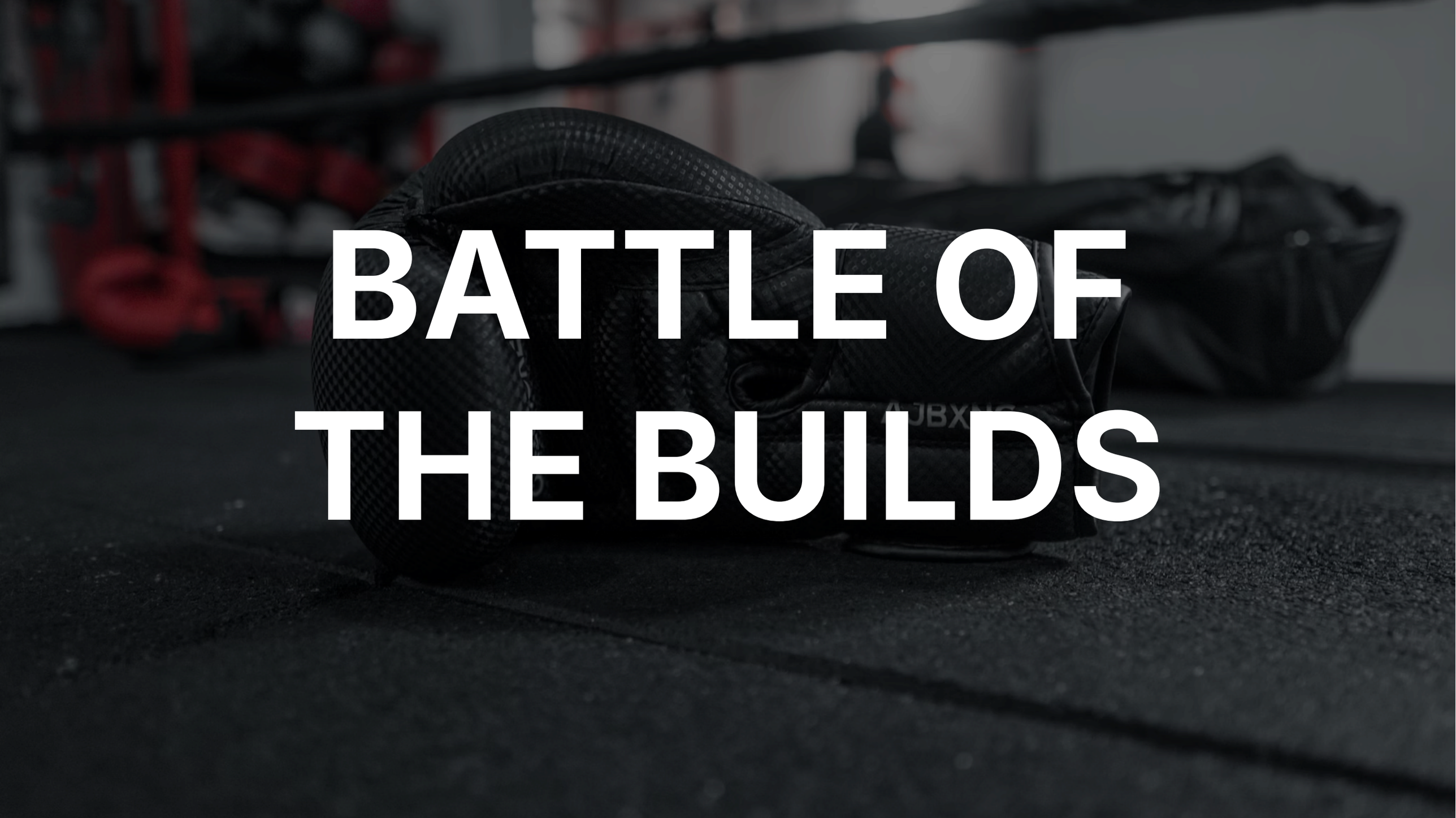
React (2) - index.html 🔗

```
<script type="module">
  import { Fragment, createElement } from "react"; // 💡 Bare names work now!
  import { createRoot } from "react-dom/client";
  import htm from "htm";
  import { movies } from "./movies.js";

  const html = htm.bind(createElement); // ✨ JSX-like template literals!
  const Movie = ({ name, year }) => html`<p>${name} (${year})</p>`;
  const App = () => html`
    <${Fragment}>
      ${movies.map((movie, i) => html`<${Movie} key=${i} ...${movie} />`)}
    </${Fragment}>`;

  const root = createRoot(document.getElementById("root"));
  root.render(html`<${App} />`);
</script>
```


How about the real world?

Performance

- Spoiler: it's faster to bundle
  - ... but how much?
  - ... and what does your application actually need?
- Let's compare build vs. no build in an unscientific...


BATTLE OF
THE BUILDS

∅ No Build

Bigger:

- 112 requests
- 826 kB

Slower:

- 250 ms

🧰 A Build

Smaller:

- 5 requests
- 640 kB

Faster:

- 150 ms

Where might we head next?

- The future that *will be*
  - JSON / CSS modules
  - WASM modules
- The future that *could be* (✨ `es-module-shims`)
  - Hot reload
  - TypeScript!?

TypeScript - movies.ts

```ts
// 💡 TypeScript types!
interface Movie {
  name: string;
  year: number;
}

export const movies: Movie[] = [
  { name: "Moana", year: 2016 },
  { name: "The Mitchells vs. The Machines", year: 2021 },
  { name: "KPop Demon Hunters", year: 2025 },
];
```

TypeScript - index.html 🔗

```html
<!-- ✨ Add the shims -->
<script src="https://ga.jspm.io/npm:es-module-shims@2.6.2/dist/es-module-shims.js"></script>

<!-- ✨ `lang="ts"` is cheating  -->
<!-- 💡 `type="module-shim"` skips native check -->
<script type="module" lang="ts">
  // ✨ Can import `.ts`
  import { movies } from "./movies.ts";

  const root = document.getElementById("root") as HTMLElement;
  root.innerHTML = movies
    .map((m) => `<p>${m.name} (${m.year})</p>`)
    .join("");
</script>
```

Go play around!

See the deck at:
ryan-roemer.github.io/a-world-without-builds

...and check out the code at:
ryan-roemer/a-world-without-builds

Thanks!

- ryan-roemer
- in/ryanroemer
- @ryan-roemer.bsky.social
- ryan.roemer@nearform.com