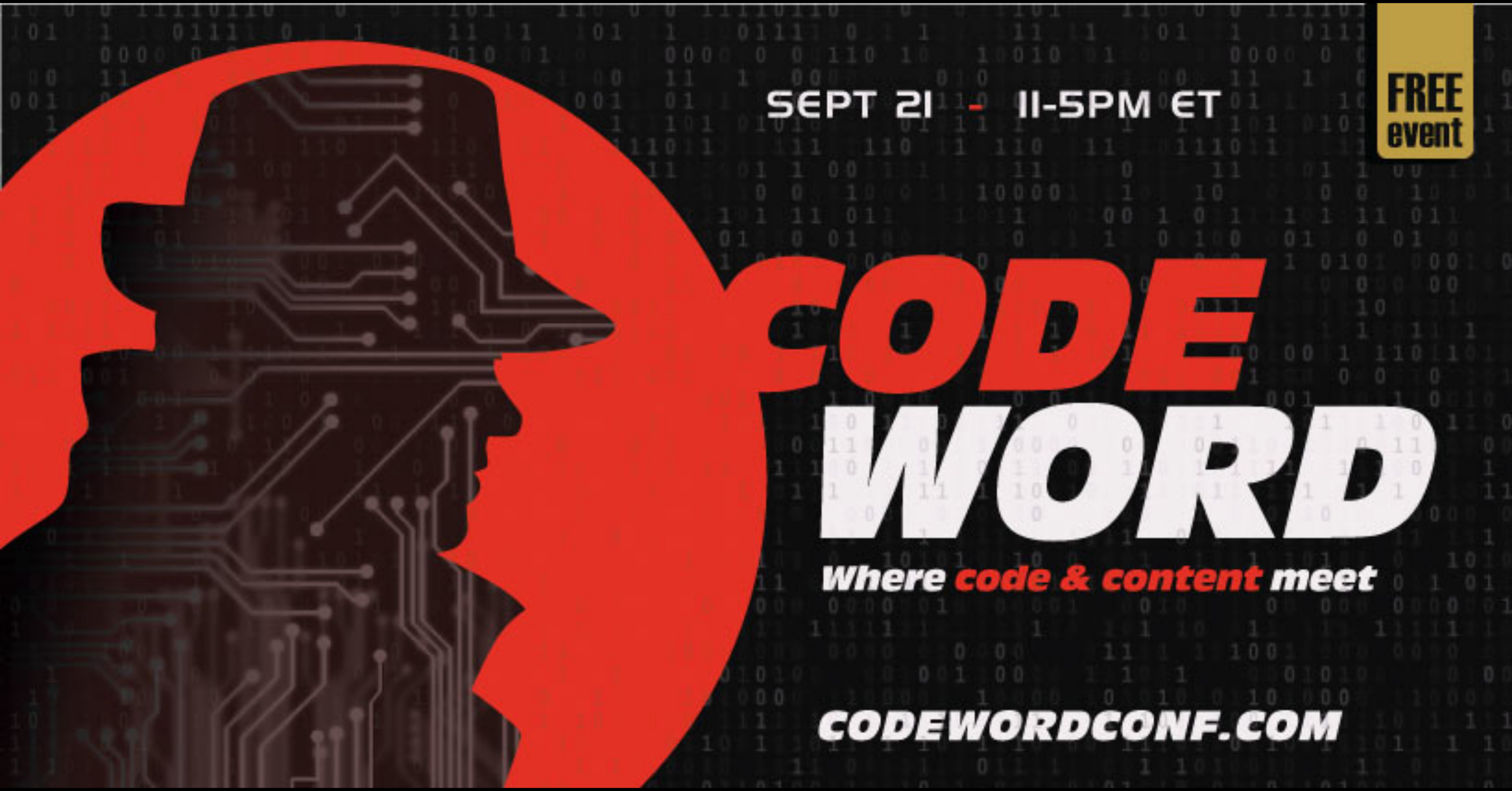
Getting Your App in Shape with SvelteKit

Who am I?

- → Developer Experience Engineer at LaunchDarkly
- → Co-author of The Jamstack Book from Manning
- → Editor of the Jamstacked newsletter from Cooper Press
- → Run the CFE.dev community and Orlando Devs

LaunchDarkly

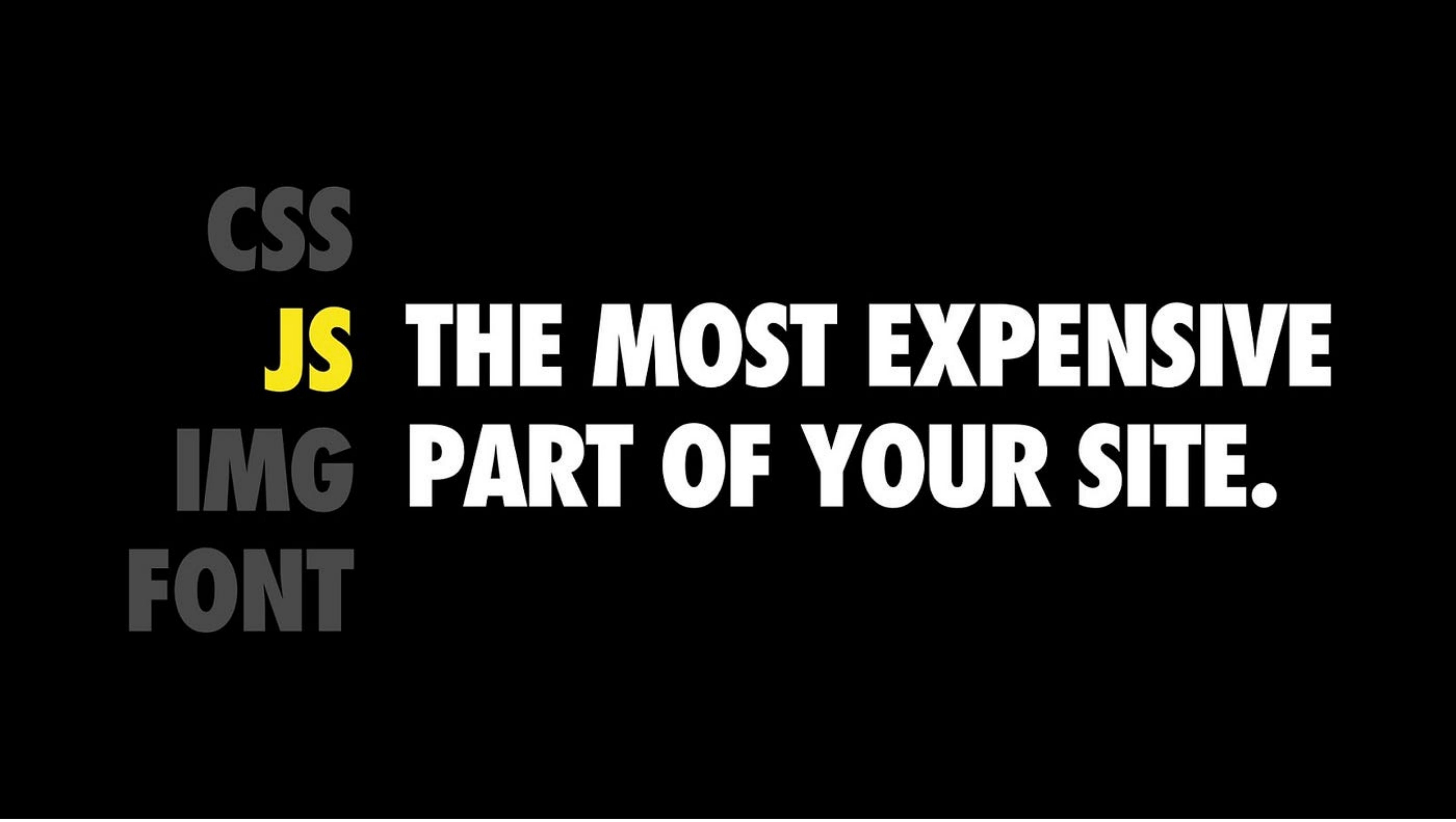

SEPT 21 - 11-5PM ET
FREE event
CODE WORD
Where code & content meet
CODEWORDCONF.COM

CodeWord Conf

Date: September 21, 11-5pm CT (UTC -4)

Cost: Free

URL: https:/ /codewordconf.com

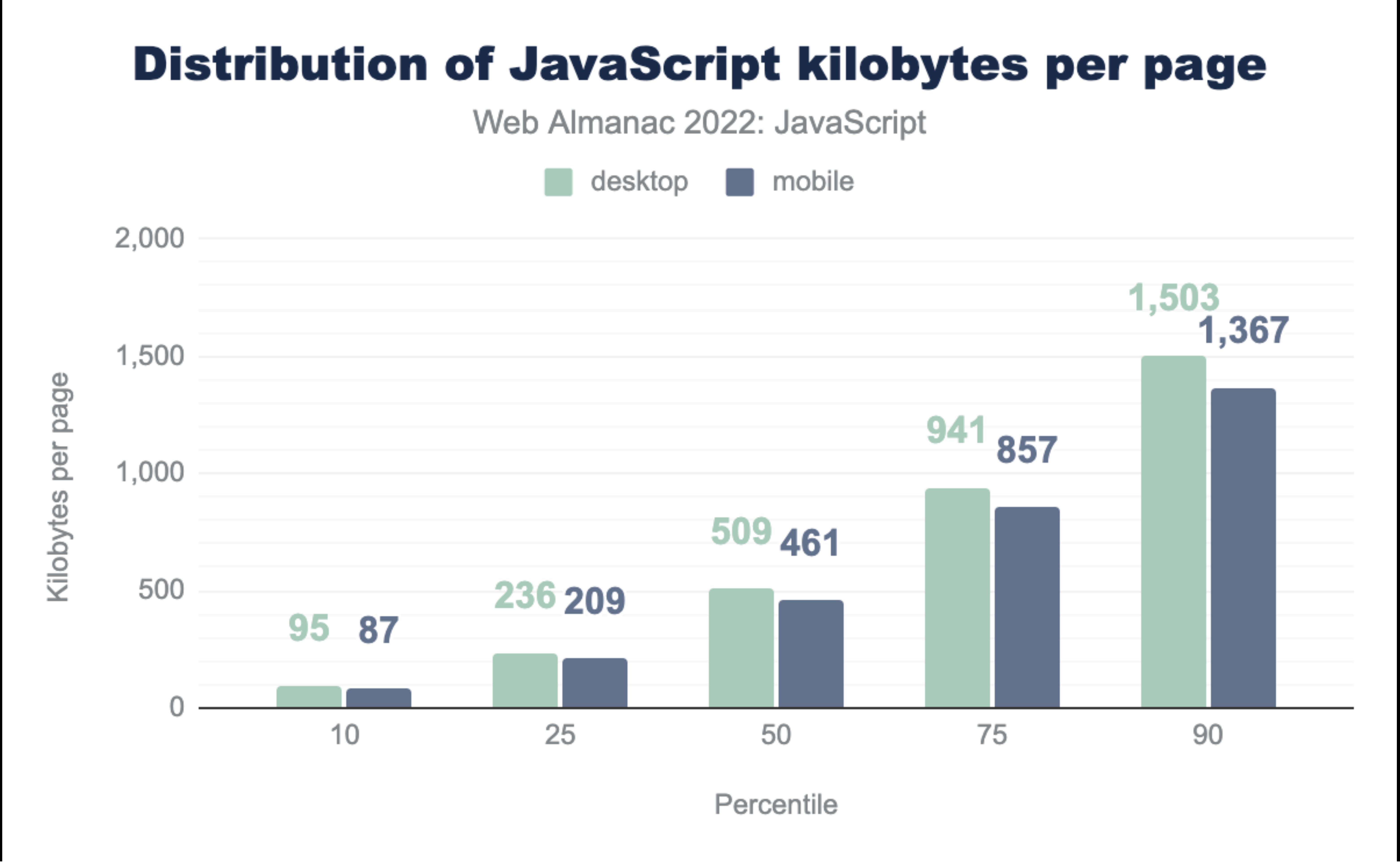


CSS
JS THE MOST EXPENSIVE
IMG PART OF YOUR SITE.
FONT

Let's first talk about JavaScript. Addy osmani has written about or presented about the cost of JavaScript every year. JavaScript remains the most expensive resource on your site and has been the same since he first talked about it in 2018. It's not just about the weight of the resources, which is growing, but also the time it takes to process it

From 2021 to 2022, an increase of 8% [in the amount of JavaScript shipped to browsers] for mobile devices was observed, whereas desktop devices saw an increase of 10%. ...The fact remains that more JavaScript equates to more strain on a device's resources.
Web Almanac

And that weight keeps growing every year. According to the web almanac's latest report, it grew another 8% for mobile in the past year. While that is a smaller increase than in some prior years, this ever expanding JavaScript can have a really detrimental impact on lower end devices that much of the world uses.

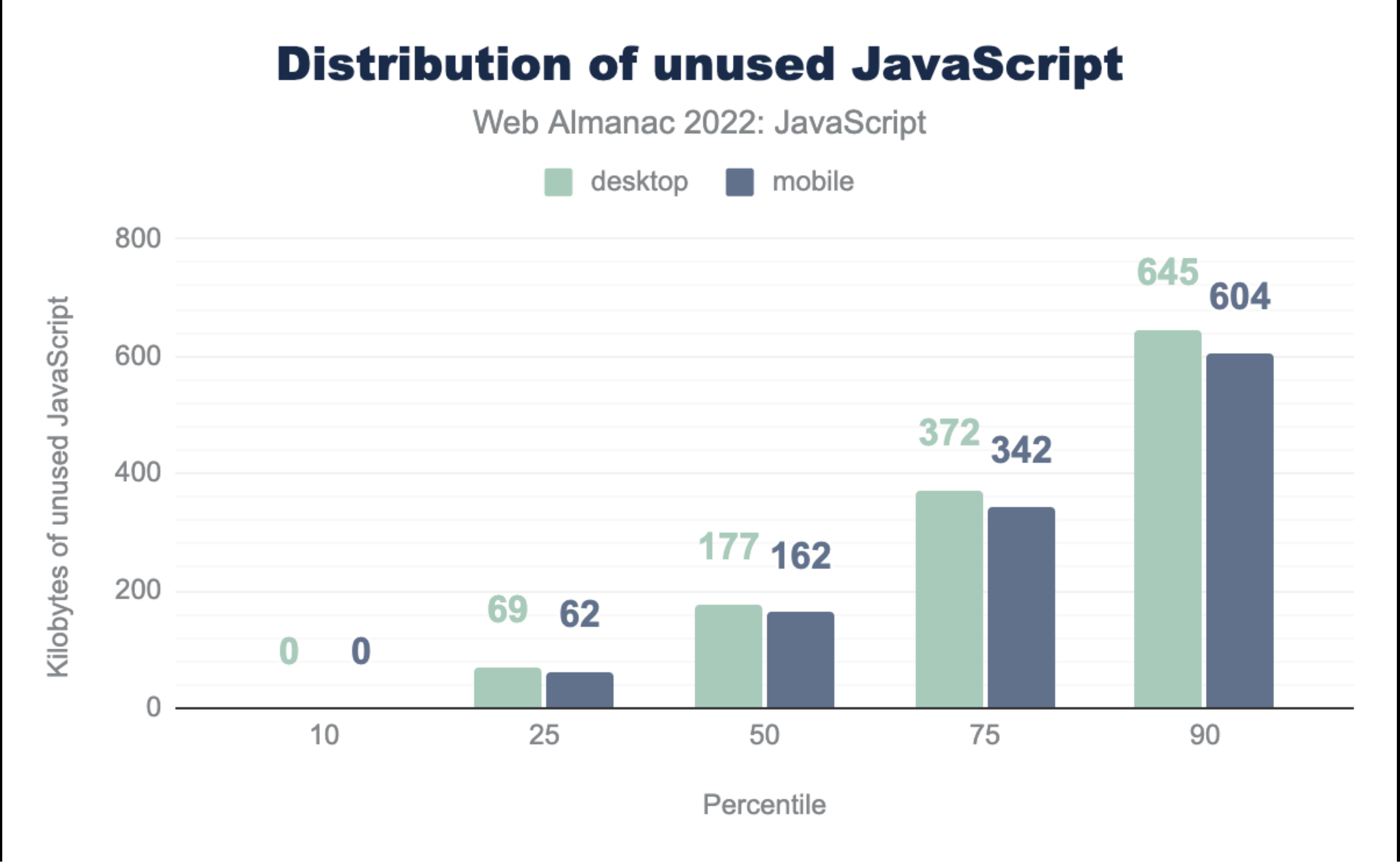


This is the distribution of JavaScript size in the sites they surveyed for their report and it can be rather shocking to imagine some sites are loading about a megabyte and a half of just JavaScript. One suspects that these same sites will have lots of this JavaScript blocking the main thread, meaning the site is unusable while the user waits.

When contrasted with the total number of bytes loaded for mobile pages at the median, unused JavaScript accounts for 35% of all loaded scripts.

Web Almanac

And here's the sadder part. Much of that JavaScript is entirely unused so this extra weight serves absolutely no purpose.



As you can see, some sites are sending as much as 600k in unused JavaScript. Why is there so much unused JavaScript? Well, in many cases, this is probably related to the ever growing list of third party scripts...

Unnecessary JavaScript

...but beyond unused JavaScript there's a category I am making up that I'm going to call unnecessary JavaScript.

Unnecessary JavaScript is what I refer to as JavaScript being used where JavaScript is not needed. It's not unused, but it is unneeded and it's related to the fact that at some point we also started using tools like React, that were meant for application development, to build everything including sites that were heavy on content with limited interactivity or web app type elements

Not only are new services being built to a self-defeatingly low UX and performance standard, existing experiences are pervasively re-developed on unspeakably slow, JS-taxed stacks.
Alex Russell, The Market for Lemons

You may have read Alex Russell's post The Market for Lemons, which dives deep into the history of how developers were convinced to become more reliant on stacks that delivered increasingly large JavaScript bundles. While you may disagree with some of his rhetoric, much of what he cites as evidence can be backed up with data.

New front-end frameworks like Solid and Qwik are suggesting that React might not have all the answers after all, and on the server Astro, Remix and Next.js (among others) are making us reconsider how much code we really need to ship to the client.
State of JavaScript 2022

Thankfully this has caused a lot of rethinking and we're seeing a ton of innovation around this issue, not just from Svelte but from frameworks like Astro, Qwik and SolidJS. Even Next.js has been pushing React server pages as a way to remedy this problem.

compiled

Svelte shifts as much work as possible out of the browser and into your build step. No more manual optimisations — just faster, more efficient apps.

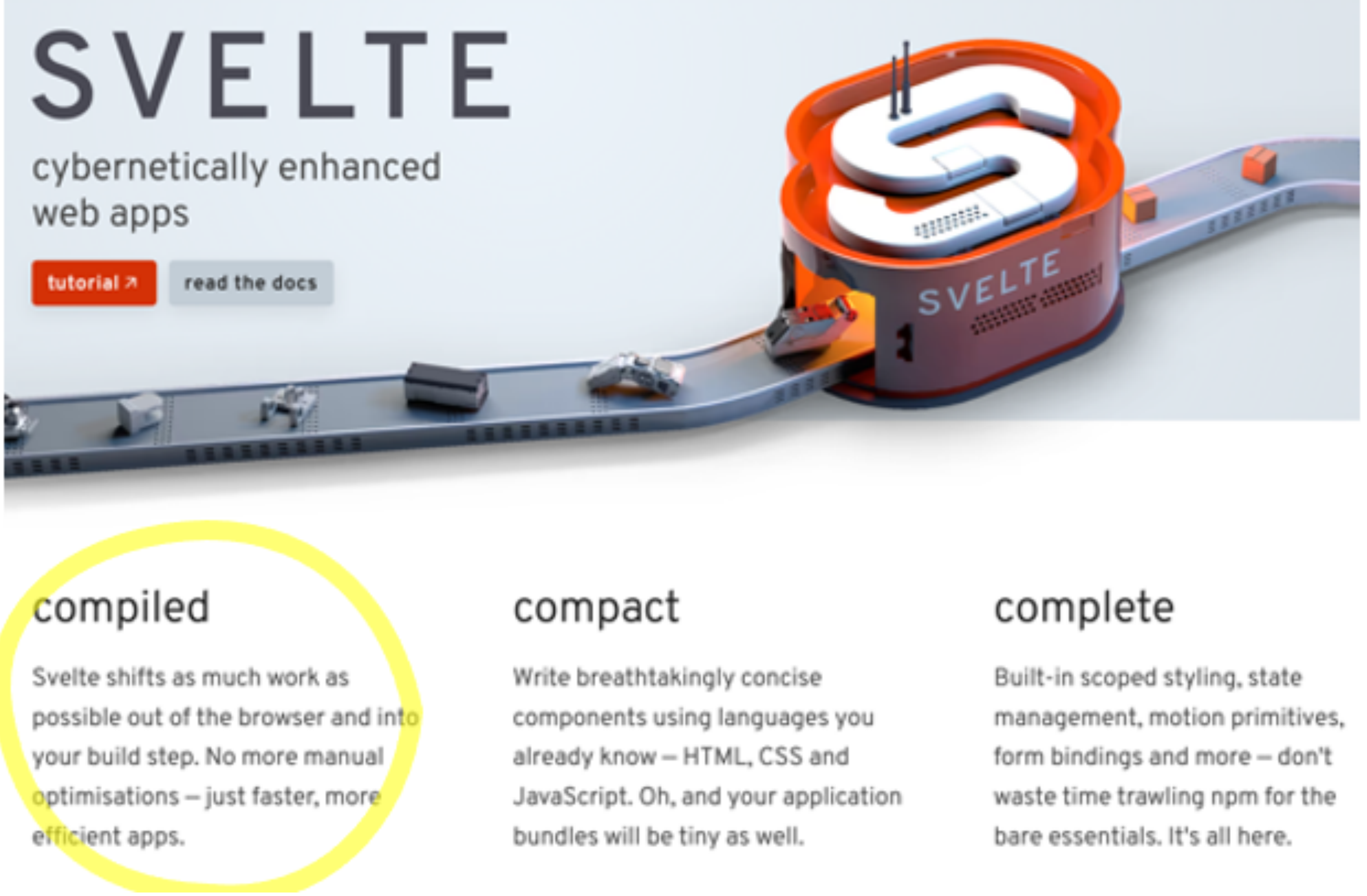
compact

Write breathtakingly concise components using languages you already know — HTML, CSS and JavaScript. Oh, and your application bundles will be tiny as well.

complete

Built-in scoped styling, state management, motion primitives, form bindings and more — don't waste time trawling npm for the bare essentials. It's all here.

Svelte was created by Rich Harris and initially released in 2016. It was one of the first frameworks to completely rethink how much JavaScript we send and how we approach building single page applications.



cybernetically enhanced web apps

compiled

Svelte shifts as much work as possible out of the browser and into your build step. No more manual optimisations – just faster, more efficient apps.

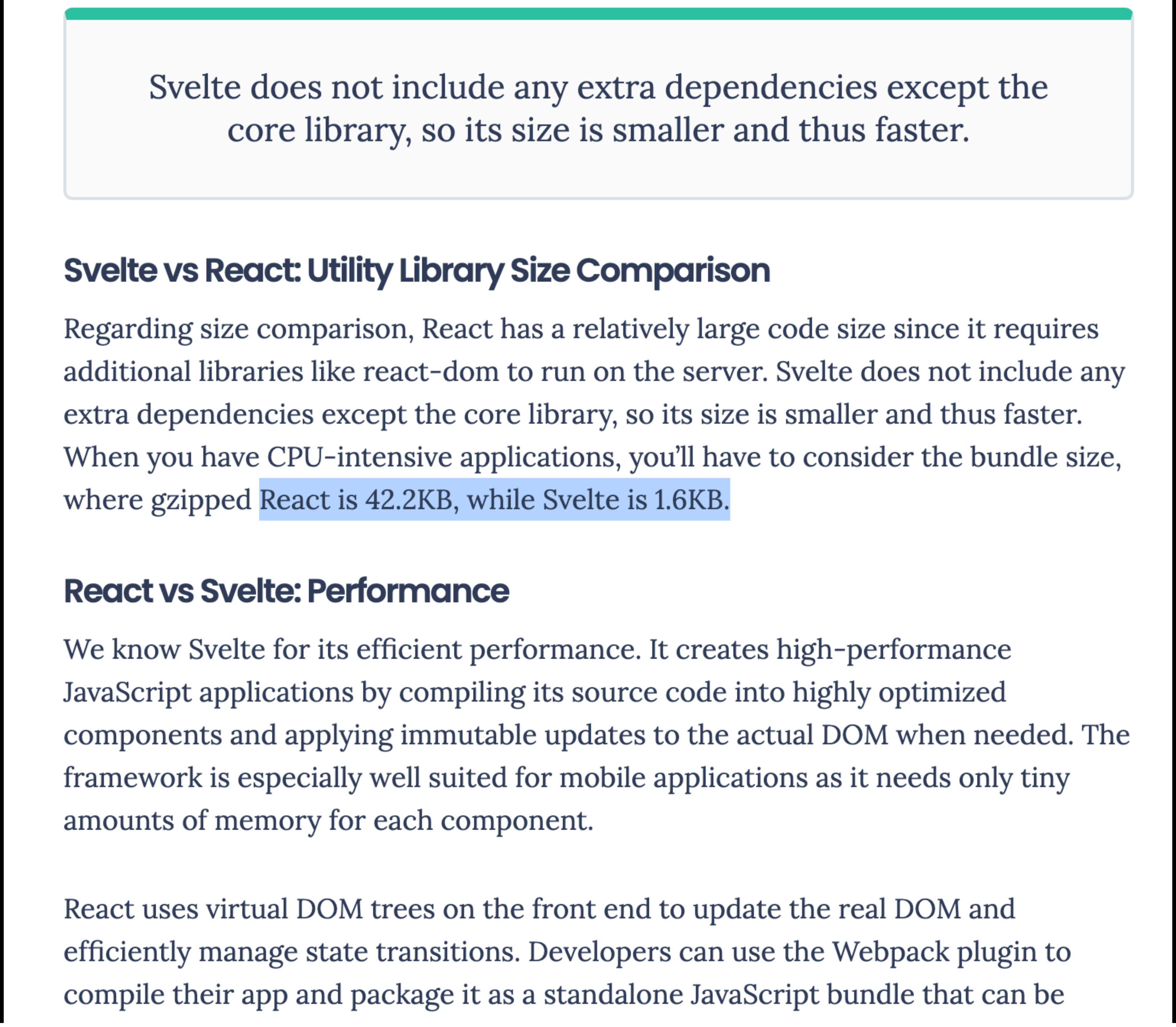
compact

Write breathtakingly concise components using languages you already know – HTML, CSS and JavaScript. Oh, and your application bundles will be tiny as well.

complete

Built-in scoped styling, state management, motion primitives, form bindings and more – don't waste time trawling npm for the bare essentials. It's all here.

This was the key difference in Svelte. It included a compile process to eliminate as much of that unnecessary JavaScript as possible and keep the bundle size small.



Svelte does not include any extra dependencies except the core library, so its size is smaller and thus faster.

Svelte vs React: Utility Library Size Comparison

Regarding size comparison, React has a relatively large code size since it requires additional libraries like react-dom to run on the server. Svelte does not include any extra dependencies except the core library, so its size is smaller and thus faster. When you have CPU-intensive applications, you'll have to consider the bundle size, where gzipped React is 42.2KB, while Svelte is 1.6KB.

React vs Svelte: Performance

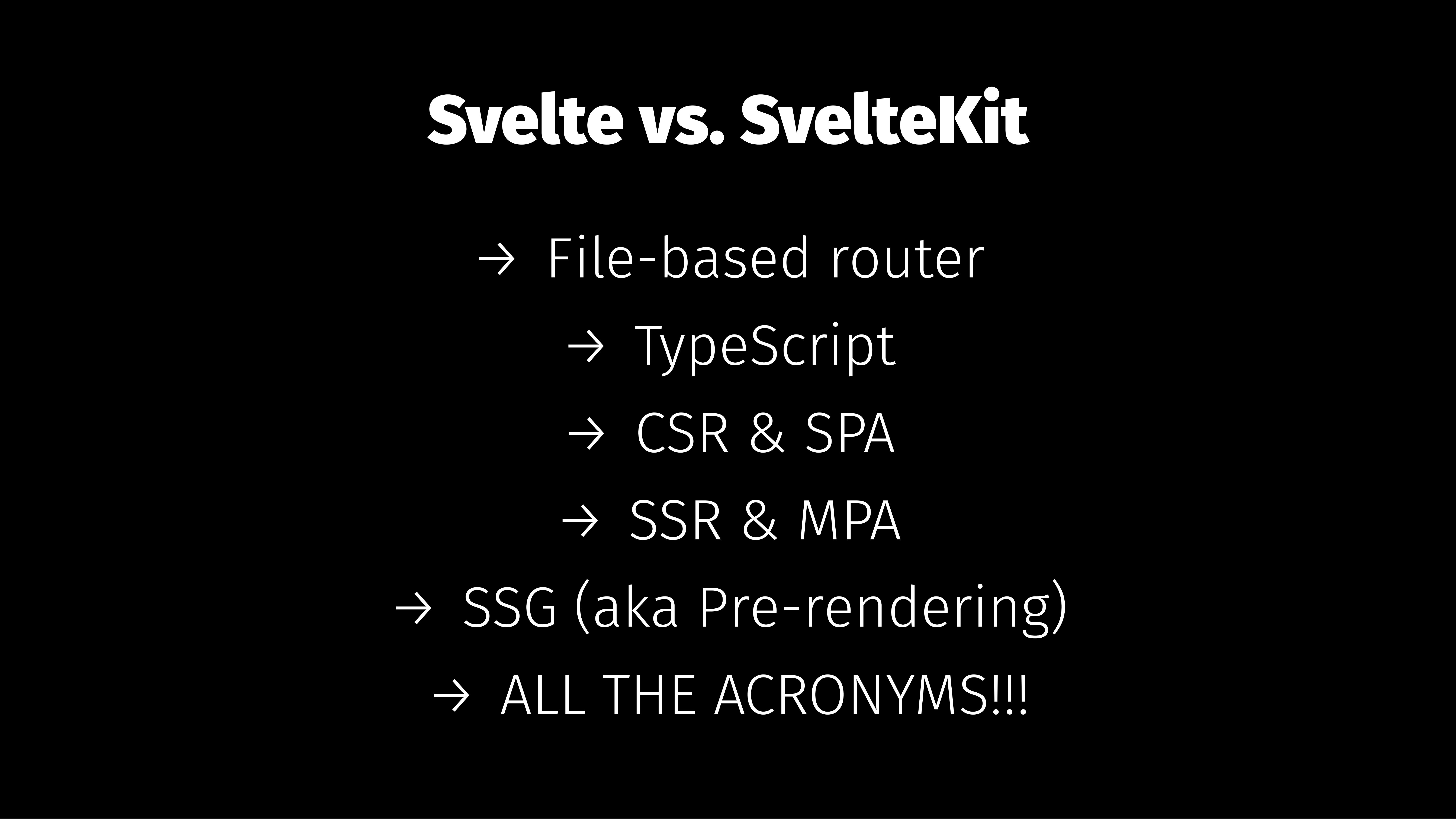
We know Svelte for its efficient performance. It creates high-performance JavaScript applications by compiling its source code into highly optimized components and applying immutable updates to the actual DOM when needed. The framework is especially well suited for mobile applications as it needs only tiny amounts of memory for each component.

React uses virtual DOM trees on the front end to update the real DOM and efficiently manage state transitions. Developers can use the Webpack plugin to compile their app and package it as a standalone JavaScript bundle that can be

According to this comparison from February, Svelte's bundle size is less than 5% of the standard React bundle size. This was done before Svelte 4 was released in June and reduced the package size and dependencies for Svelte down significantly and made large reductions to the compiled output size of SvelteKit apps.

Svelte vs. SvelteKit

While Svelte's been around since 2016, SvelteKit didn't hit 1.0 until December of last year. It existed previously but went through a lot of changes heading into 1.0, but has since stabilized as you'd expect. The easiest way to think about SvelteKit is basically as the full stack option for building Svelte apps kind of like Next.js is to React.



Svelte vs. SvelteKit

→ File-based router

→ TypeScript

→ CSR & SPA

→ SSR & MPA

→ SSG (aka Pre-rendering)

→ ALL THE ACRONYMS!!!

SvelteKit provides a router and supports static site generation (or prerendering), client-side rendering and server side rendering. It also bakes in a lot of build optimizations and best practices for web app development out of the box.

What's So Great About SvelteKit?

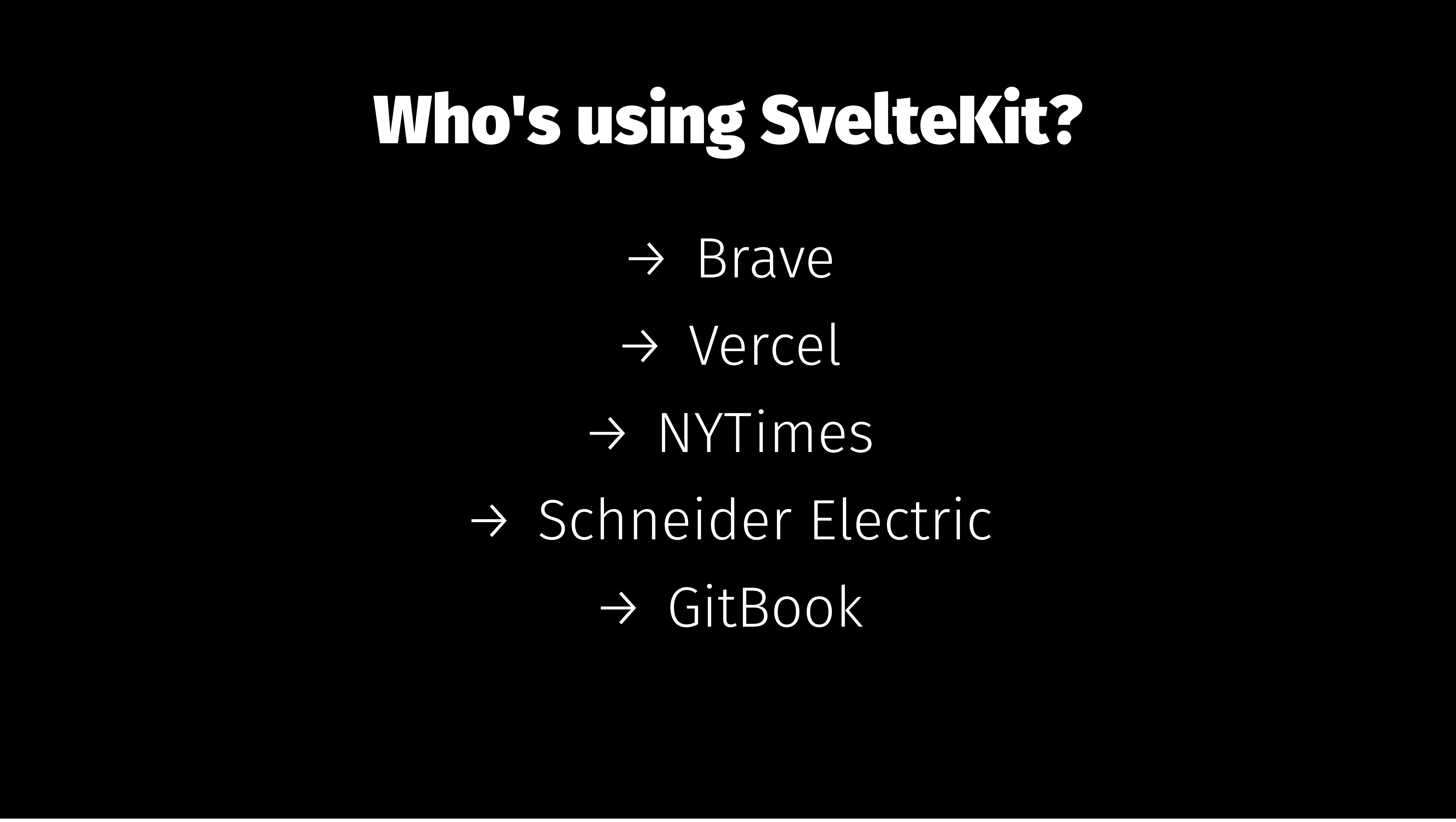
→ It's just JavaScript...

→ ...but less of it (at least on the client)

→ Separation of backend and frontend code

→ It's (relatively) easy

One of the things I like about tools like SvelteKit and also Astro is that I don't need to learn their way of doing things. I always felt that, I knew how to do things in JavaScript but not necessarily in React. I don't feel that way with SvelteKit. Plus, as we discussed you get less JavaScript on the client. It's also slightly easier to wrap your head around than something like Astro (which I love by the way) because there's none of the complexity surrounding islands. And I really like the way it separates frontend from backend code, eliminating a common confusion, especially in tools like Next. That being said, the structure can be difficult to navigate in your IDE.



Who's using SvelteKit?

→ Brave

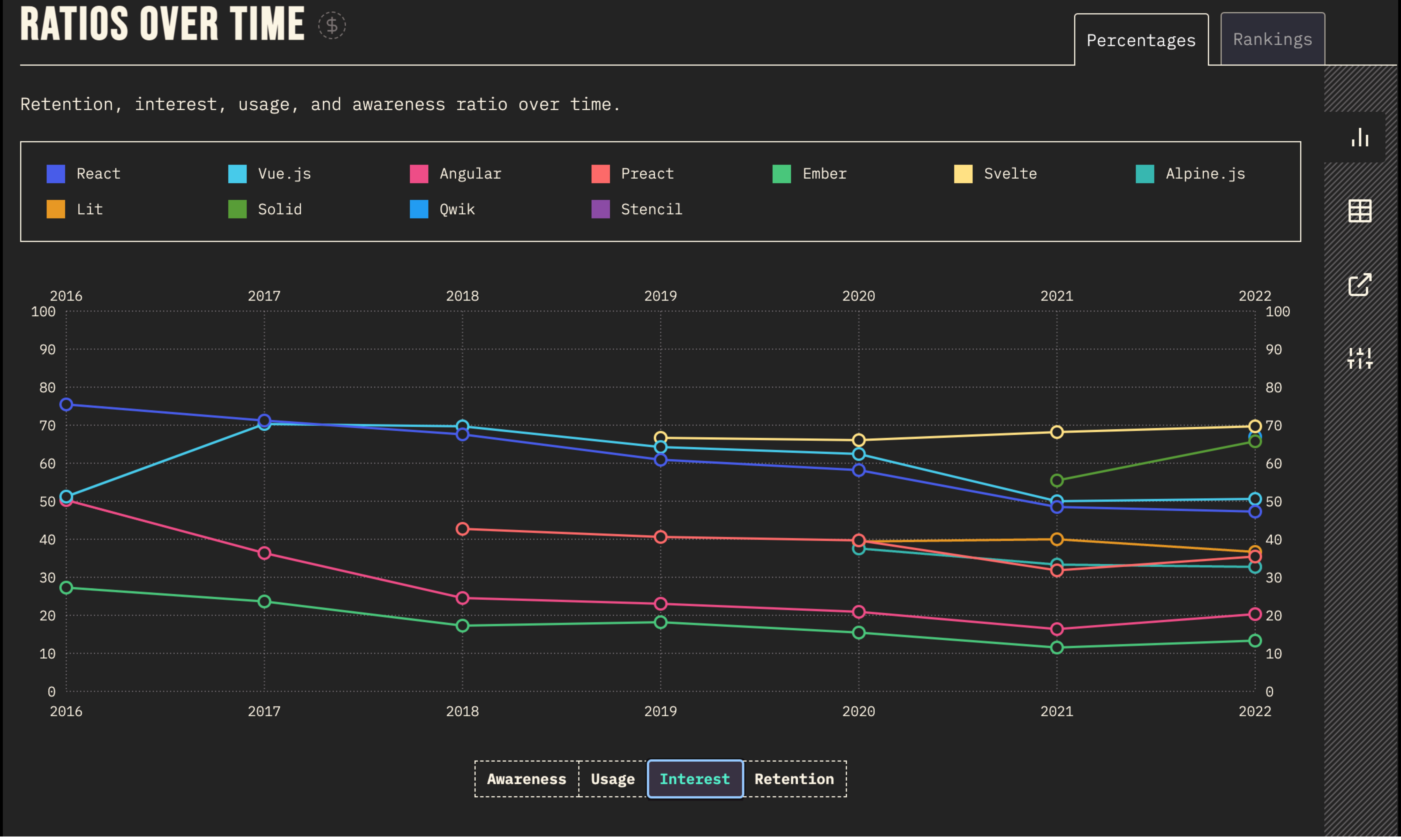
→ Vercel

→ NYTimes

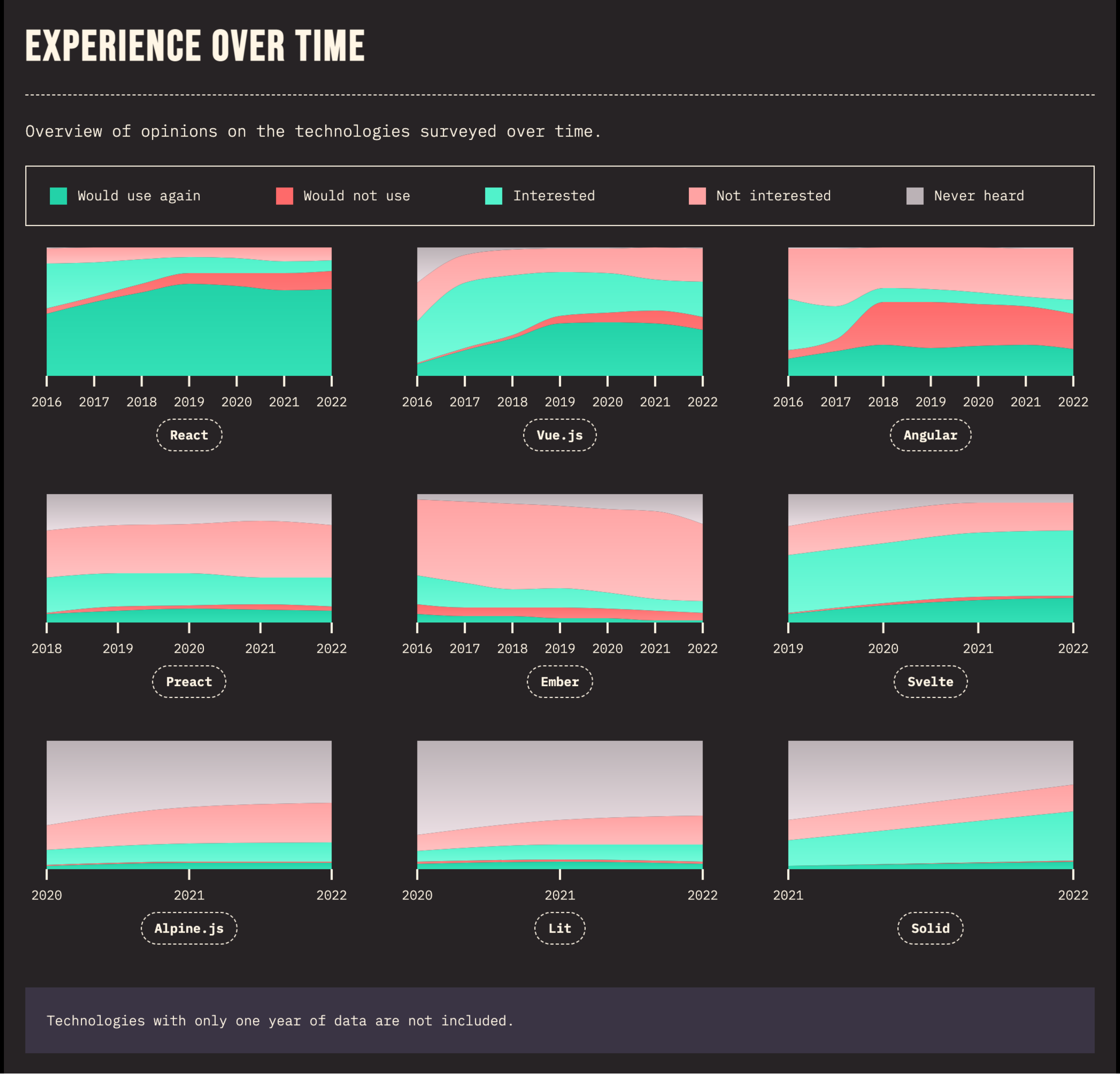
→ Schneider Electric

→ GitBook

SvelteKit doesn't really advertise specific usage but, based on Wappalyzer and other sources, there are some known companies using it. Vercel is where Rich Harris works now and NYTimes is where he was before.



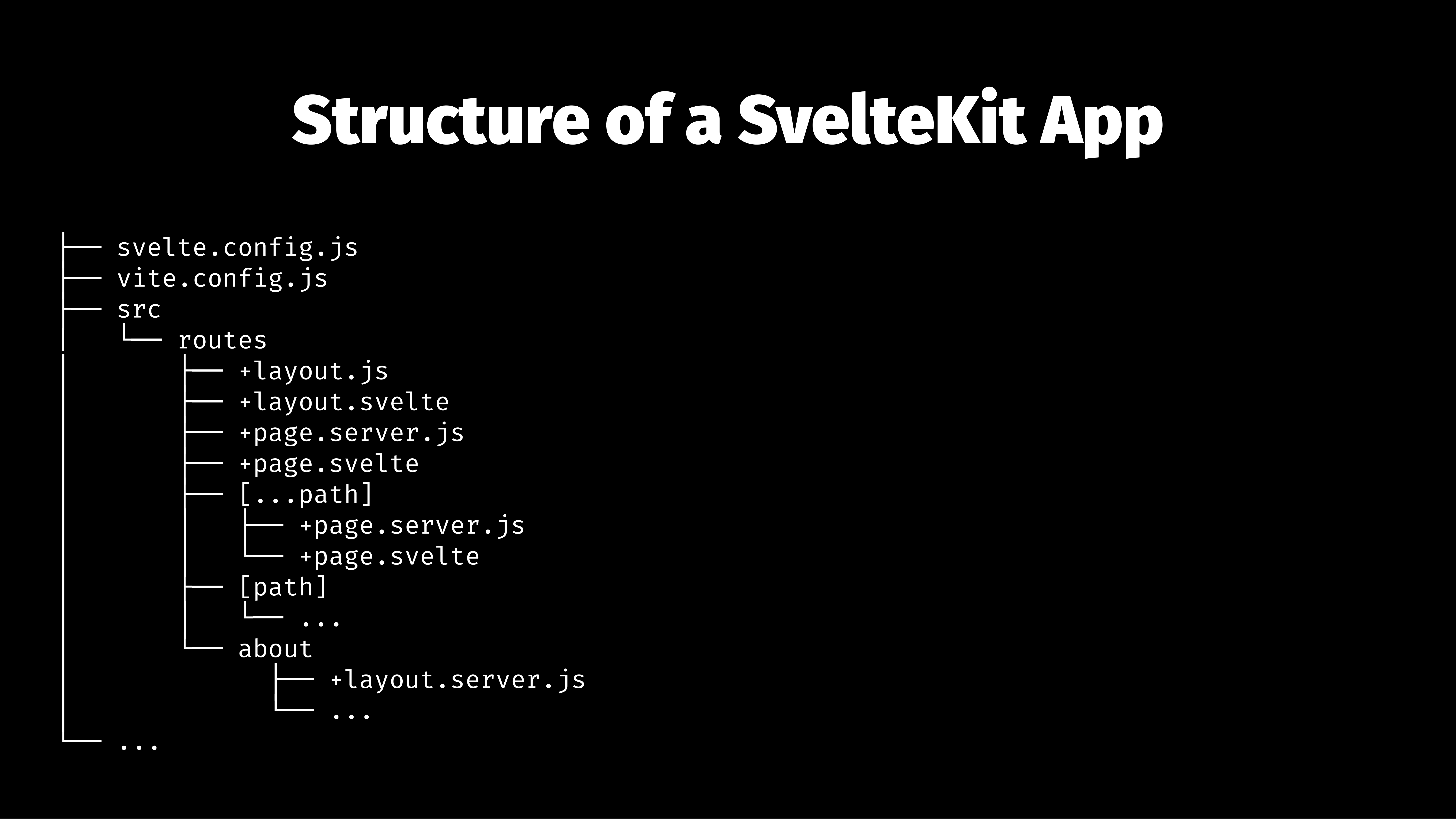
But I expect that to continue to expand, and not just because Vercel is now backing the project. While remaining stable, interest in Svelte continues to outpace other frameworks, especially the big three.



In fact it compares to React and Vue for the green streams, which are would use again and interested, though leaning far more heavily on interested. That being said, it's the only one that seems to be trending up on both.

5

Work has already started on Svelte 5 which will include a rewrite of the Svelte compiler and runtime



Structure of a SvelteKit App

```
├── svelte.config.js
├── vite.config.js
├── src
│   └── routes
│         ├── +layout.js
│         ├── +layout.svelte
│         ├── +page.server.js
│         ├── +page.svelte
│         ├── [...path]
│         │   ├── +page.server.js
│         │   └── +page.svelte
│         ├── [path]
│         │   └── ...
│         └── about
│               ├── +layout.server.js
│               └── ...
└── ...
```

Before we jump into the demo I want to set some context, because SvelteKit's structure is somewhat unique.

Demo

now let's dig into a simple demo I created to give you a better sense of how SvelteKit works.

LaunchDarkly + SvelteKit (server-side)

Wrapping the SDK in a JavaScript library:

```
import LaunchDarkly from 'launchdarkly-node-server-sdk'
let launchDarklyClient
async function initialize() {
  const client = LaunchDarkly.init(import.meta.env.VITE_LAUNCHDARKLY_SDK_KEY)
  await client.waitForInitialization()
  return client
}

...

export async function getFlagValue(key, context) {
  const client = await getClient()
  let flagValue
  if (!context) {
    context = {
      key: 'anonymous',
    }
  }
  flagValue = await client.variation(key, context, false)
  return flagValue
}
```

LaunchDarkly + SvelteKit (server-side)

Get the flag value in your load method

```
// launchdarkly libraries
import { getFlagValue } from '../../lib/launchdarkly/server'

export async function load({ params }) {
  const featuredUsername = await getFlagValue('my-flag')
  // use the flag or return it
}
```

LaunchDarkly + SvelteKit (client-side)

On the client, just add a script tag. Be sure to enable hydration.

```
import { browser } from '$app/environment';
import { getFlagValue } from '../launchdarkly/client'

let myFlag
if (browser) {
  getFlagValue('flag-key-123abc').then(setMyFlag)
}
function setMyFlag(val) {
  myFlag = val
}
```

Get started

https://learn.svelte.dev

If you're ready to get started, there is a great interactive tutorial that covers both the svelte and sveltekit side

Slides and Code

https://github.com/remotesynth/sveltekit-demo

Questions?

Email: brinaldi@launchdarkly.com
Mastodon: https://mastodon.xyz/@remotesynth
BlueSky: https://staging.bsky.app/profile/remotesynthesis.com