



Breaking Up with SVG-in-JS

written version & sources – kurttextrem.de/posts/svg-in-js

Jacob Groß | 9th April 2025

Who am I?



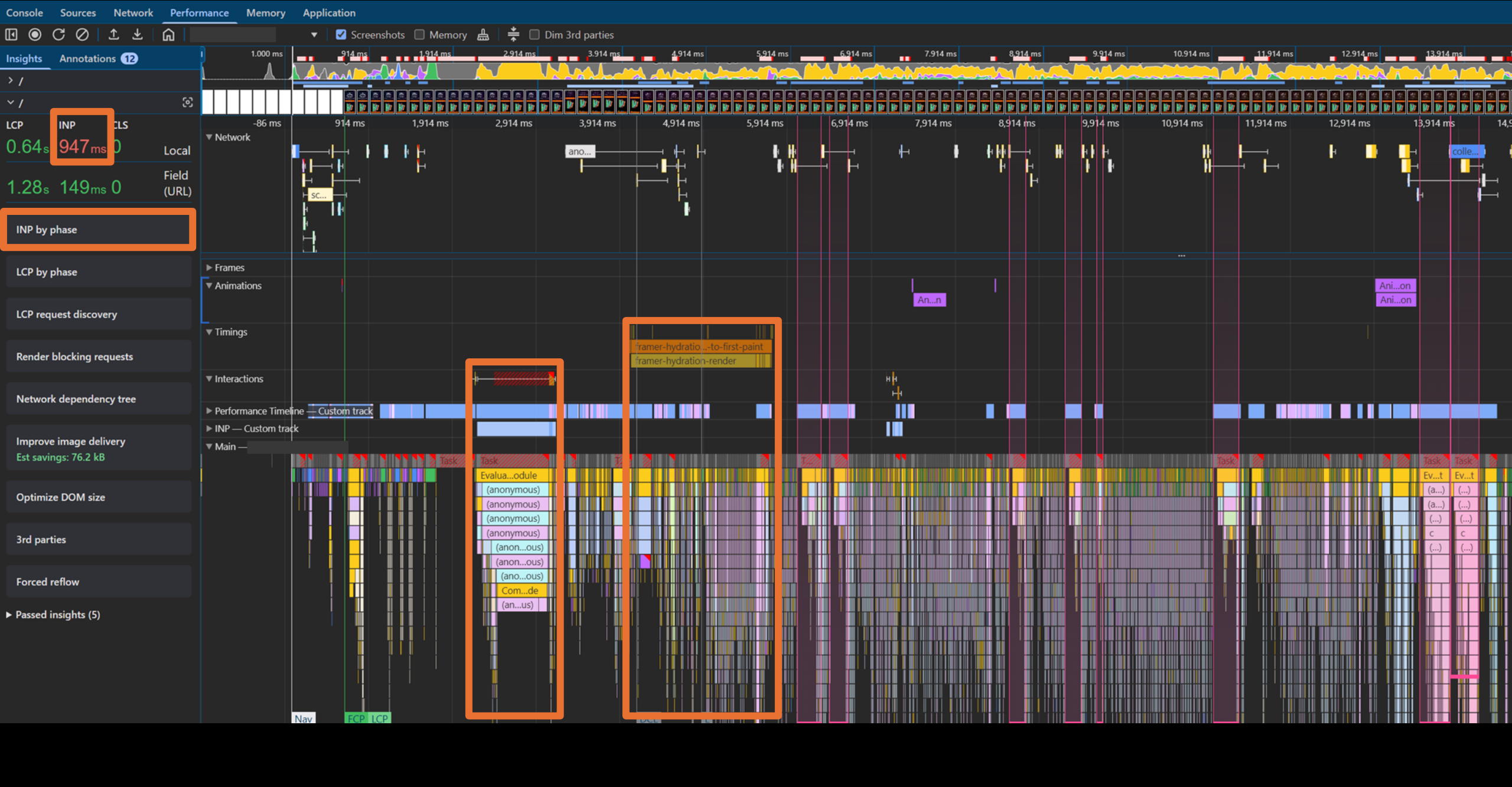
- Senior Performance Engineer @ Framer *(you might know us from "Framer Motion")*
- Participant in W3C WebPerfWG calls, if you have webperf topics, come chat 😊
- Studied Human-Computer-Interaction (UX) @ LMU Munich

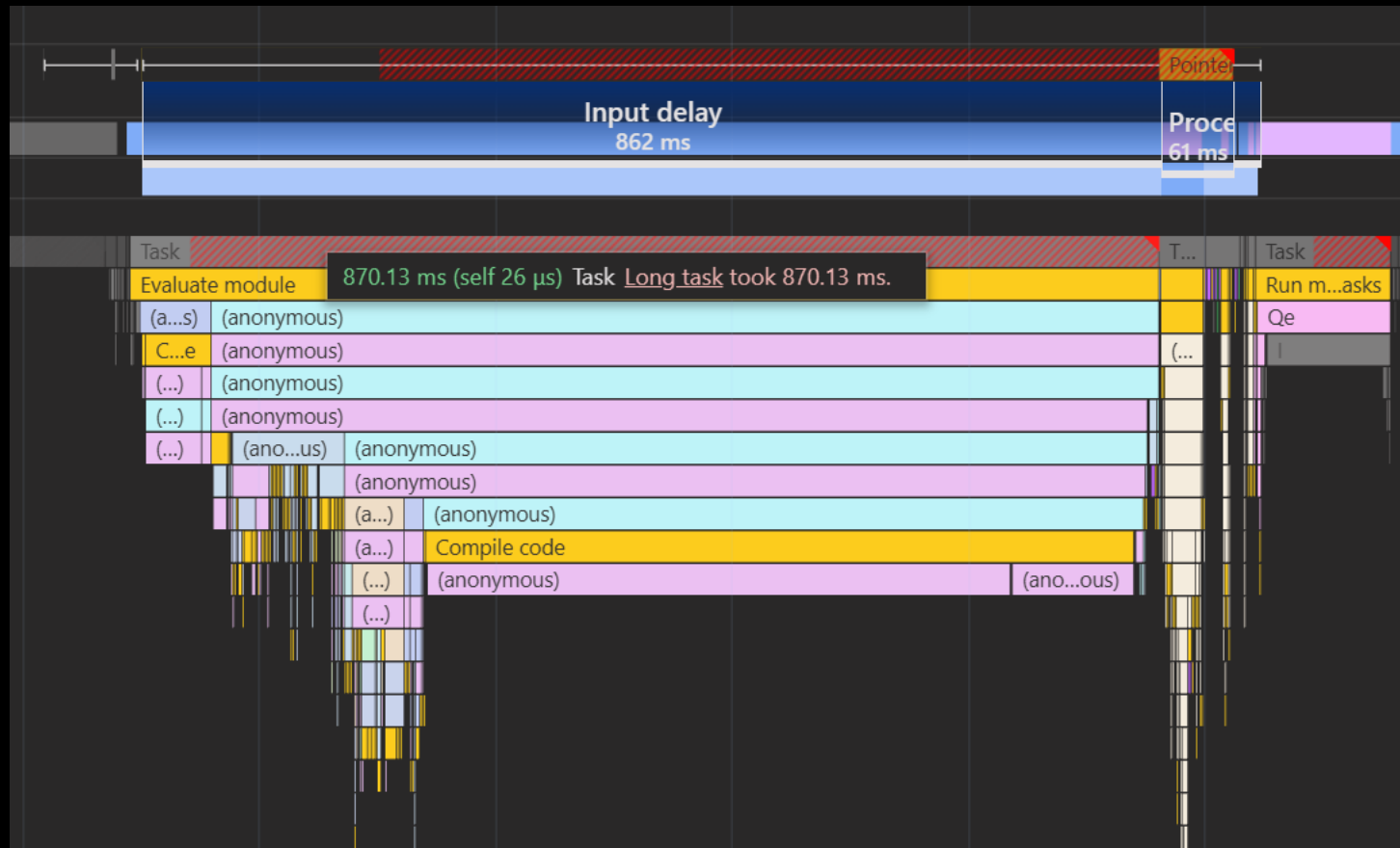
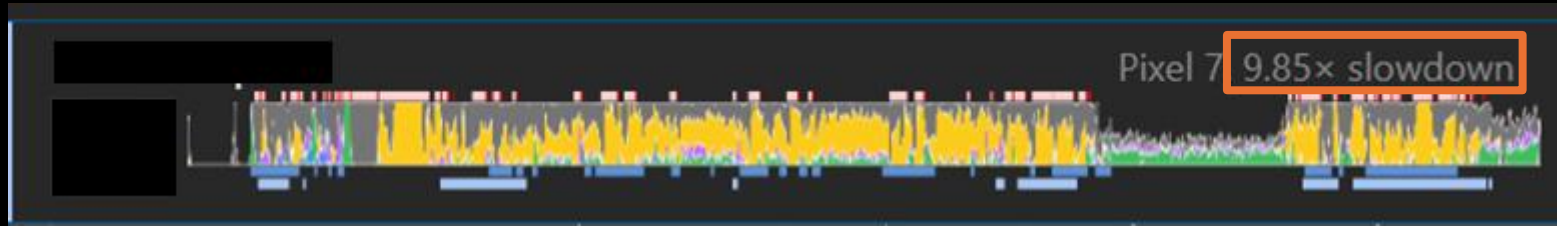
I like making things **fast & accessible**

What Jason meant we shouldn't do:

```
3   import React, { useState, useEffect, FormEvent } from "react"
4   import type { ComponentType } from "react"
⚡  import { Eye, EyeSlash } from "phosphor-react"
```

What happens if you do:





Excursion: It's always those damn barrel files.

Speeding up the JavaScript ecosystem - The barrel file debacle



written by [@marvinh.dev](#) 08 October 2023

P.S.:

In case you're curious - [Next.js/Turbopack](#) includes optimizations for some packages for that reason.

```
1  /* GENERATED FILE */
2  export type { Icon, IconProps, IconWeight } from "./lib";
3  export { IconContext, IconBase } from "./lib";
4  export * as SSR from "./ssr";
5
6  export * from "./csr/Acorn";
7  export * from "./csr/AddressBook";
8  export * from "./csr/AddressBookTabs";
9  export * from "./csr/AirTrafficControl";
10 export * from "./csr/Airplane";
11 export * from "./csr/AirplaneInFlight";
12 export * from "./csr/AirplaneLanding";
13 export * from "./csr/AirplaneTakeoff";
14 export * from "./csr/AirplaneTaxiing";
15 export * from "./csr/AirplaneTilt";
16 export * from "./csr/Airplay";
17 export * from "./csr/Alarm";
18 export * from "./csr/Alien";
19 export * from "./csr/AlignBottom";
20 export * from "./csr/AlignBottomSimple";
21 export * from "./csr/AlignCenterHorizontal";
22 export * from "./csr/AlignCenterHorizontalSimple";
23 export * from "./csr/AlignCenterVertical";
24 export * from "./csr/AlignCenterVerticalSimple";
25 export * from "./csr/AlignLeft";
26 export * from "./csr/AlignLeftSimple";
27 export * from "./csr/AlignRight";
28 export * from "./csr/AlignRightSimple";
29 export * from "./csr/AlignTop";
30 export * from "./csr/AlignTopSimple";
31 export * from "./csr/AmazonLogo";
32 export * from "./csr/Ambulance";
33 export * from "./csr/Anchor";
34 export * from "./csr/AnchorSimple";
35 export * from "./csr/AndroidLogo";
36 export * from "./csr/Angle";
37 export * from "./csr/AngularLogo";
38 export * from "./csr/Aperture";
39 export * from "./csr/AppStoreLogo";
40 export * from "./csr/AppWindow";
41 export * from "./csr/AppleLogo";
42 export * from "./csr/ApplePodcastsLogo";
43 export * from "./csr/ApproximateEquals";
44 export * from "./csr/Archive";
45 export * from "./csr/Armchair";
46 export * from "./csr/ArrowArcLeft";
47 export * from "./csr/ArrowArcRight";
48 export * from "./csr/ArrowBendDoubleUpLeft";
49 export * from "./csr/ArrowBendDoubleUpRight";
50 export * from "./csr/ArrowBendDownLeft";
51 export * from "./csr/ArrowBendDownRight";
52 export * from "./csr/ArrowBendLeftDown";
53 export * from "./csr/ArrowBendLeftUp";
54 export * from "./csr/ArrowBendRightDown";
```

Tree-shaking: Impossible

Code

Blame

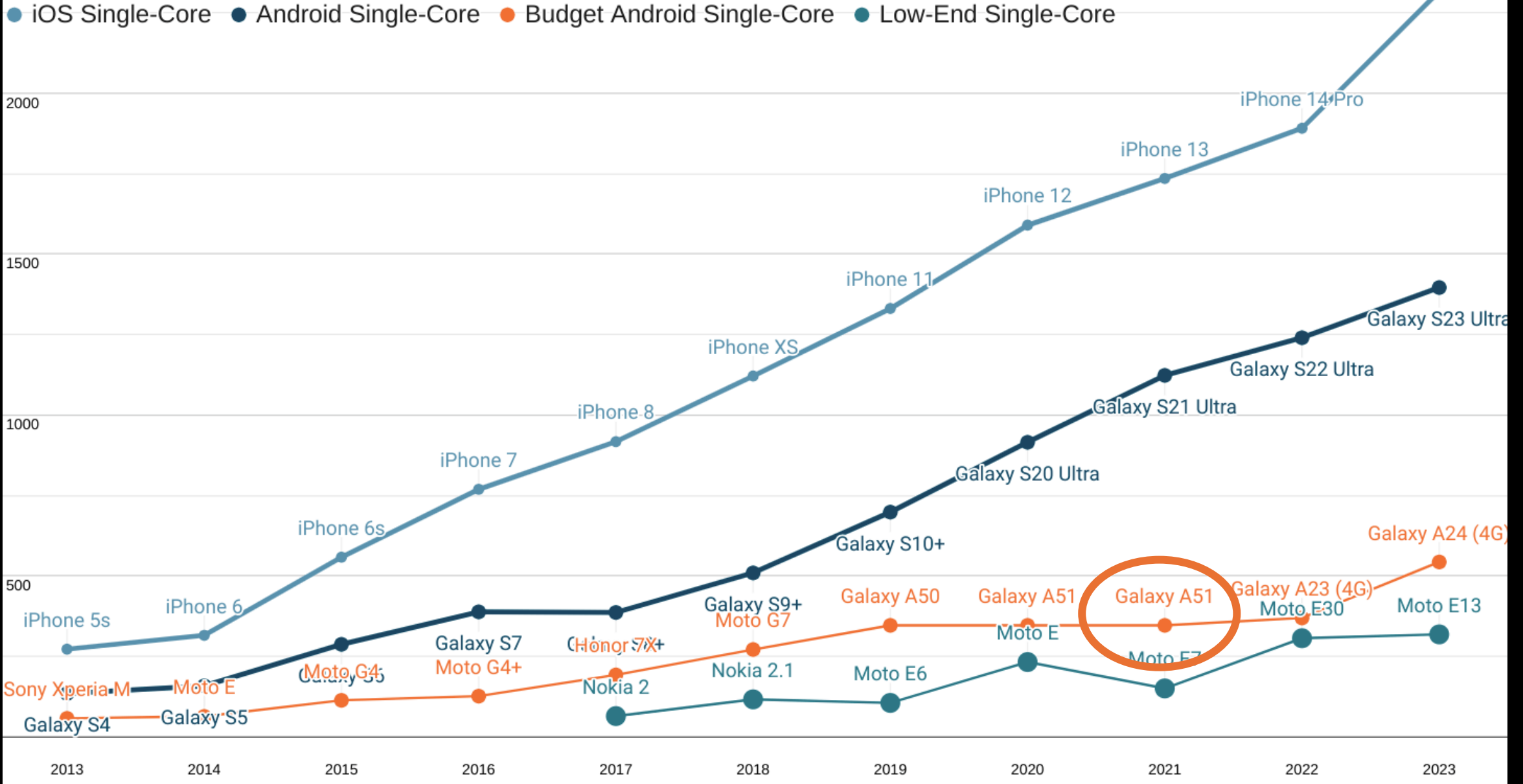
20 lines (18 loc) · 6.13 KB

```
1  /* GENERATED FILE */
2  import React, { forwardRef } from "react";
3  import type { Icon } from "../lib/types";
4  import IconBase from "../lib/IconBase";
5  import weights from "../defs/Acorn";
6
7  /**
8   * @regular ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5v
9   * @thin ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5v
10  * @light ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5v
11  * @bold ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5v
12  * @fill ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5v
13  * @duotone ![img](data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My
14  */
15  const I: Icon = forwardRef<Icon, React.PropsWithRef<Icon>>((props, ref) => (
16    <IconBase ref={ref} {...props} weights={weights} />
17  ));
18
19  I.displayName = "Acorn";
20  export { I as Acorn };
```

F*ck

**But maybe no
one has such a
slow device?**

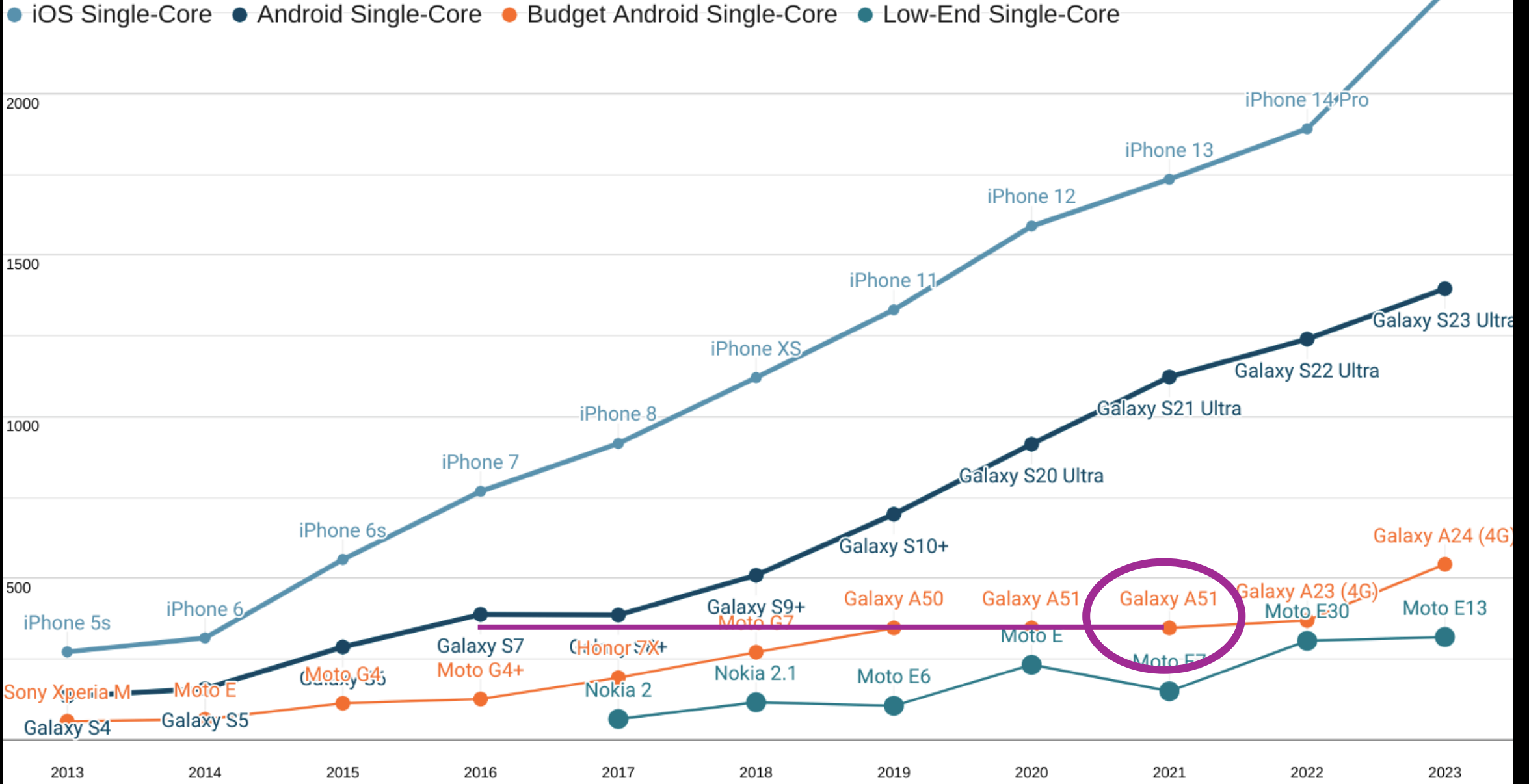
Geekbench 5 Single-Core Scores



The web should be equal for everyone

- Mobile devices can be sloooooow 🐌
- iPhone 16 Pro != avg. users' device (unless you're in a special field)
- Avg. 🇩🇪 Android device != avg. developing country device
- Galaxy A51 for 240€ from 2021 is the avg. device
Sounds recent?

Geekbench 5 Single-Core Scores





GALAXY A51 LOADING MY JS

MY IPHONE

imgflip.com

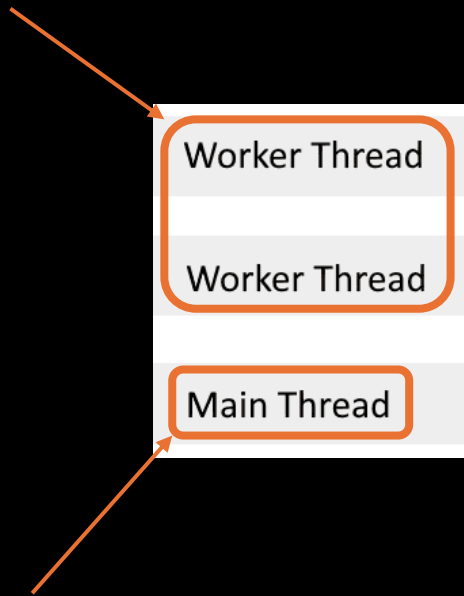
How engines (V8) treat JS

How engines (V8) treat JS



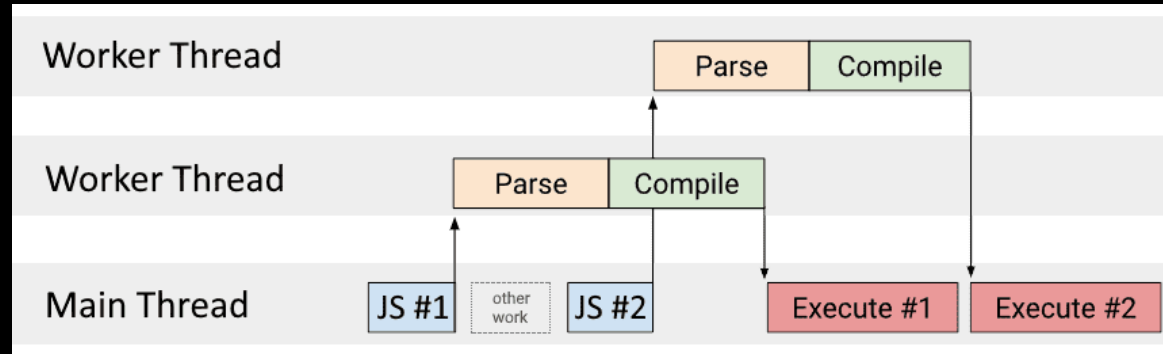
How engines (V8) treat JS

Done in “background”



Anything here impacts UX directly
Nothing else can run meanwhile.

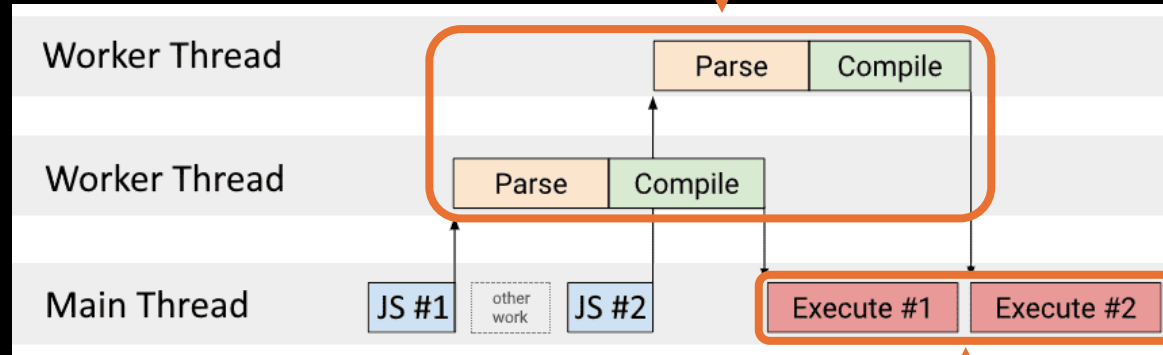
How engines (V8) treat JS



HTML parser finds JS

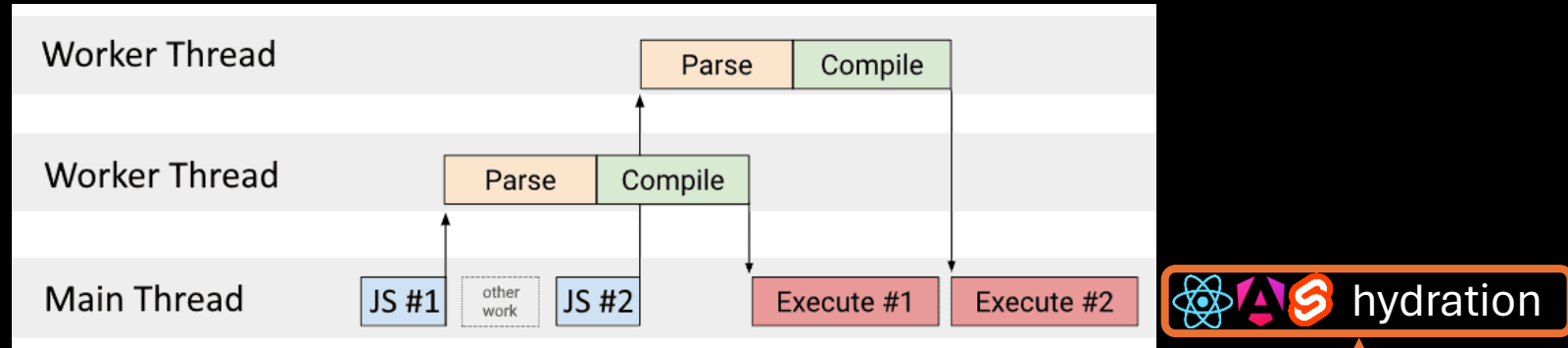
How engines (V8) treat JS

Done in “background”



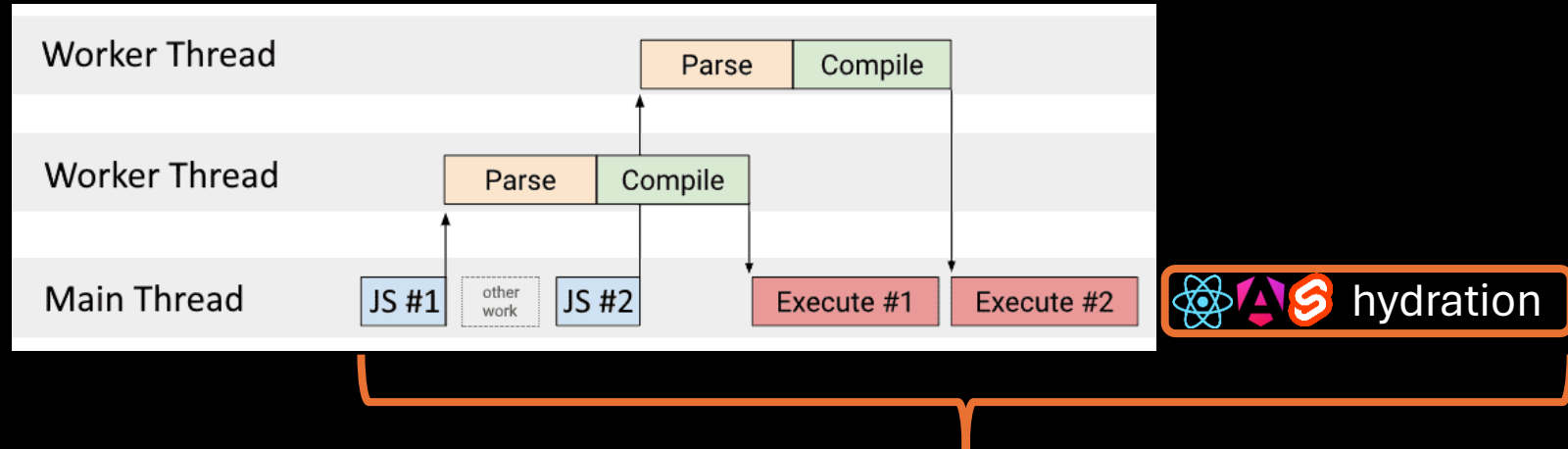
Anything here impacts UX directly
Nothing else can run meanwhile.

How engines (V8) treat JS



For most frameworks, hydration follows execution, too.

How engines (V8) treat JS



Time until something meaningful happens on user interaction (think hamburger menu)

How engines (V8) treat JS

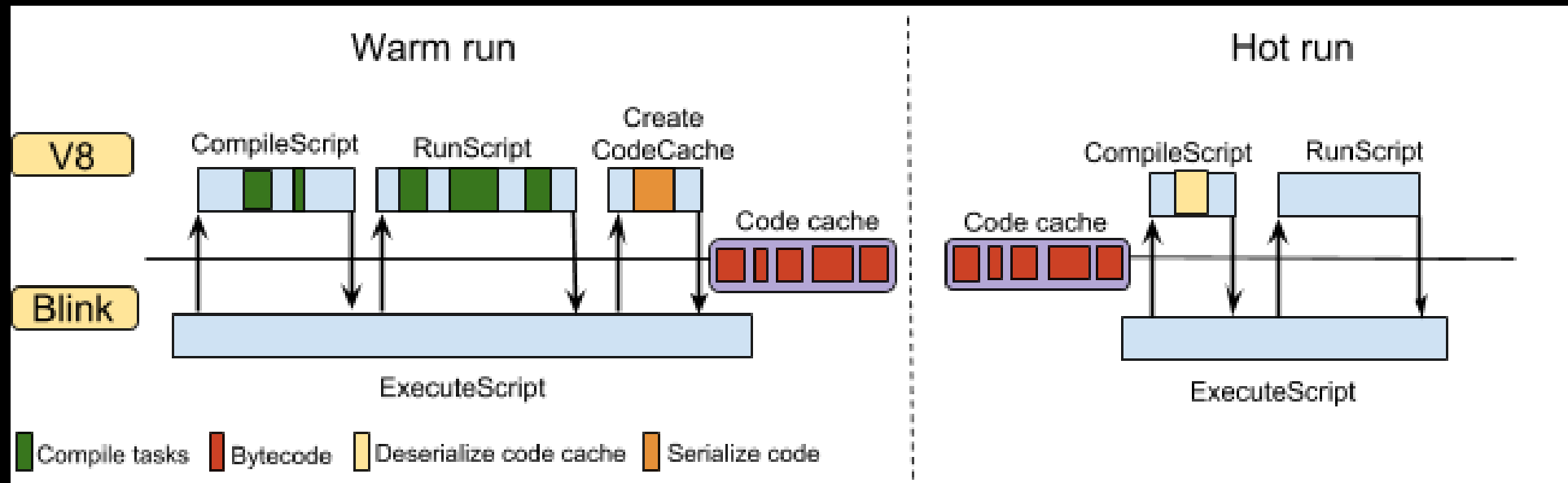
JavaScript is always blocking *when executed*

- attributes like `async/defer` don't change that
- `link rel="modulepreload"@ServiceWorker` caching only moves when parse + compile happens (to right after download)
- Inlined JS always blocks the parser until parsing, compilation and execution is done
- Inlined JS cannot be byte cached
- Be careful of what you inline, especially for JS

How engines (V8) treat JS – Nits

- How truly parallel parsing & compilation is, depends on the number of cores the devices CPU has and how many scripts are loading
- ESM: top-level imports execute synchronously – now we know why importing from a barrel file is so slow
- More files = more overhead, especially for small ones
 - HTTP headers can get bigger than the file content
 - V8: disables byte cache if below the 1 kb threshold

How engines (V8) treat JS – Byte Cache



“On Android, this optimization also translates to a 1–2% reduction in the top-level page-load metrics like the time a webpage takes to become interactive.”

How engines (V8) treat JS – Summary

- A busy background (CPU) can also impact the main thread implicitly – your website is likely not the only thing running
- Not only slow, but can cause bad INP if a user clicks somewhere meanwhile

Your users are waiting for all those reasons.
Sometimes they might experience bad INP.
Sometimes it's just nothing happens for a while.

How engines (V8) treat JS – Summary

*“Byte-for-byte, JavaScript is **more expensive** for the browser to process **than the equivalently sized image or Web Font**” – Tom Dale, web.dev*

SVGs are HTML-like XML tags that describe images.

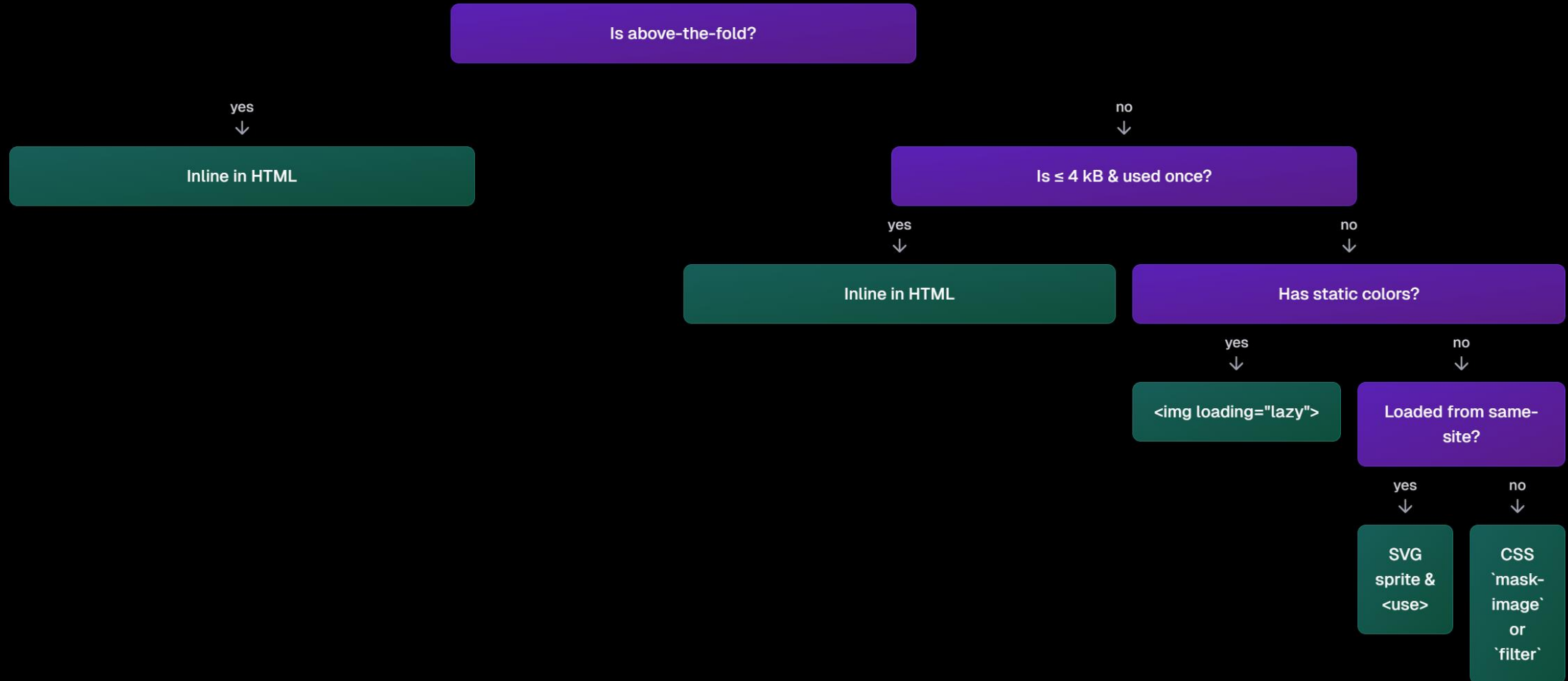
I don't think anyone wants (jpeg/png) images in their JS.

... all of the strings also live in the JS memory heap – which is limited on cheap phones

So, what do we do about SVGs?

Moving SVGs out of JS

Moving SVGs out of JS



1. Good ol' `` tag

Webpack:

```
const config = {  
  // ... other parts of the Webpack config ...  
  module: {  
    rules: [  
      {  
        test: /\.svg/,  
        type: "asset/resource",  
      },  
    ],  
  },  
};
```

Astro:

No config needed! 🎉

Usage:

```
import HeartIcon from "./HeartIcon.svg";  
  
const App = () => <img src={HeartIcon} loading="lazy" />;
```

Don't forget to turn on Gzip / Brotli compression for .svg files on your CDN/server.

1. Good ol' `` tag

Pro's:

- You can use attributes like `loading="lazy"` & `fetchpriority="high"` if it's important
- Keeps DOM complexity simple - just 1 DOM node
- Most performant option of all
- SVG animations on DPR > 1x screens might consume less CPU compared to an inline SVG

Con's:

- Needs workaround for `currentcolor` and custom props:
They don't inherit values from the current page (the SVG file is treated as an external resource and not as part of the DOM)
- Chromium: SVG animations run capped to 60 Hz and use more CPU on DPR = 1x screens
- `<a>` tags *embedded* in the SVG can't be clicked

2. But how can I <use> more colors?

- Make it part of the DOM, by using <use>

```
import HeartIcon from "./HeartIcon.svg";  
  
const App = () => <svg><use href={`#${HeartIcon.#heart}`} /></svg>;
```

- Must reference an ID, so the file might look like this:

```
<svg viewBox="0 0 300 300" id="heart">...</svg>
```

Now you can just use `currentColor` & CSS props again.

The return of the sprites

- SVG sprites are an option to combine many icons into one file
- This can make sense if you load 100 1kb files – the overhead of HTTP headers then might be bigger than the svg itself
- ... but using just 1 icon out of a 100 kb file is less reasonable

```
←!— icons.svg →  
<svg>  
  ←!— 1: add a `<defs>` tag →  
  <defs>  
    ←!— 2: wrap in `<symbol>` and give it an ID (and other attributes such as `viewBox`) →  
    <symbol id="icon1">  
      ←!— 3: paste the content of the SVG inside of the `<symbol>` →  
      ...  
    </symbol>  
    <symbol id="icon2">...</symbol>  
  </defs>  
</svg>
```

The return of the sprites


- SVG sprites are an option to combine many icons into one file
- This can make sense if you load 100 1kb files – the overhead of HTTP headers then might be bigger than the svg itself
- ... but using just 1 icon out of a 100 kb file is less reasonable

Usage is simple:

```
<svg><use href="icons.svg#icon1" /></svg>  
<svg><use href="icons.svg#icon2" /></svg>
```

<use> con's:

- <use> can only load from same-site – more on that soon
- <mask> & <clipPath> don't work when the SVG is external
- Needs manual work ... or tools like:

Icon-pipeline - "  SVG icon pipeline - Optimize icons & build SVG sprites"

JetBrains SVG sprite loader - "Webpack loader for creating SVG sprites."

@svg-use - "a set of tools and bundler plugins, to ergonomically load SVG files as components, via SVG's <use href> mechanism"

Replacing JS with CSS

```
const Icon = (className) => (  
  // add a class and let any consumer handle the details via CSS ↓  
  <svg><use href={`#${HeartIcon}#heart`} className={`heart ${className}`} /></svg>  
);  
const YellowHeart = () => <Icon className="yellow" />;  
const BigHeart = () => <Icon className="big" />;  
const App = () => <<YellowHeart /><BigHeart /></>;
```

```
.heart { fill: currentcolor } /* ← apply the current `color` */  
  
/* ↓ those classes might even come from your design system and don't need to be SVG specific */  
.big { width: 300px }  
.yellow { color: #FFFF00 }
```

3. Inline in HTML

Pro's:

- No extra HTTP request -> renders immediately
 - + good for logos
 - + good for search & hamburger icons
- Pretty much makes sense for anything important above-the-fold, where you want the user to be able to interact right away

Con's:

- More DOM nodes / DOM depth
- More to download, less to cache (unless you cache the entire page)

3. Inline in HTML

1. Like with critical CSS, inline carefully.
Critical CSS is recommended to stay below 14 kB.
2. Use data URIs (``) or if you run into the caveats, inline the full `<svg>`

Inling != “put into the JS bundle again”

Inlining the server way

Next.js: Don't add `"use client"` -> makes the SVG automatically a React Server Component (= removed from JS bundle)

Inlining the server way

All other scenarios:

```
import fs from "node:fs";

const svgIcons = await fs.readFile("path/to/icons.svg"); // ← load SVG file content
const svgIcons2 = await fs.readFile("path/to/icons-2.svg"); // ← load SVG file content

// ... router might come from HTTP frameworks like fastify or express ...
app.get("/", function () {
  const reactOutput = renderToString(App);
  return `<!doctype html><head><title>SVG-in-HTML</title></head>
  <body>
    <!-- ↓ output the SVG sprite to be re-used and make it invisible →
    <div style="display:none">${svgIcons}</div>

    ${reactOutput}

    <!-- ↓ less important ones here →
    <div style="display:none">${svgIcons2}</div>
  </body>`;
});
```

In the case of CORS

“SVG <use> elements don’t currently have any way to ask for cross-origin permissions. They just don’t work cross-origin, at all.”

- [O’Reilly Media book by Amelia Bellamy-Royds, Kurt Cagle, and Dudley Storey](#)

SVGs are “old”, back then CORS wasn’t a thing.
But CSS knows how to treat CORS:

```
.heart {  
  mask-image: url("somecdn.com/HeartIcon.svg#heart");  
  /* 📌 'color' the SVG */  
  background-color: currentcolor;  
}
```

But: A mask, like , is also not part of the DOM, so caveats apply here as well.

Give it some love

- Although basically everyone knows and uses SVGs, browsers are slow to implement new SVG features
- This leads to a limited feedback cycle between spec authors & browser devs
- Check out how cool this would be: <https://kizu.dev/svg-linked-parameters-workaround/#the-spec>

```
.icon {  
  link-parameters:  
    param(--background, pink)  
    param(--accent, var(--accent))  
};  
}
```

Any `` could then read *--background*

Measure, then optimize.

(or if you start, use one of the mentioned tools)

**But no matter what you do,
NEVER put a PNG/JPG in the SVG:**



Matt Zeunert · 1.

Founder at DebugBear: Core Web Vitals Monitoring

[Zur Website](#)

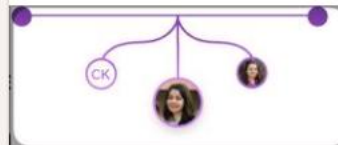
2 Wochen · 🌐

Normally, SVG images are good for web performance 🚀

When you find a huge SVG file it's usually not because of the vector image itself... but rather because there are raster images embedded inside it.

Just found a page loading two **17 megabyte SVGs** today!

Übersetzung anzeigen



SVG files are usually pretty small and load quickly...

...but sometimes they contain several large JPEGs images

Rendered size: 4096 x 2731 px
Rendered aspect ratio: 4096:2731
File size: 11.5 MB
Current source: data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAQABAAQ.../LdV7Mo7H7aVhtRv4O9a2dqTjG9u6f0f3eIcu/HXIP68f/9k=



Thank you. Questions?

Find me on

X: [@kurtextrem](#) | Bluesky: [@kurtextrem.de](#) | LinkedIn: [in/kurtextrem](#)