VINTAGE BUNDLES

Sia Karamalegos

hi, i'm sia

⚡ 👩‍💻 🇬🇷 🐶 🐺 🎨 📓 ✈️

Resources for this talk

bit.ly/vintagebundles

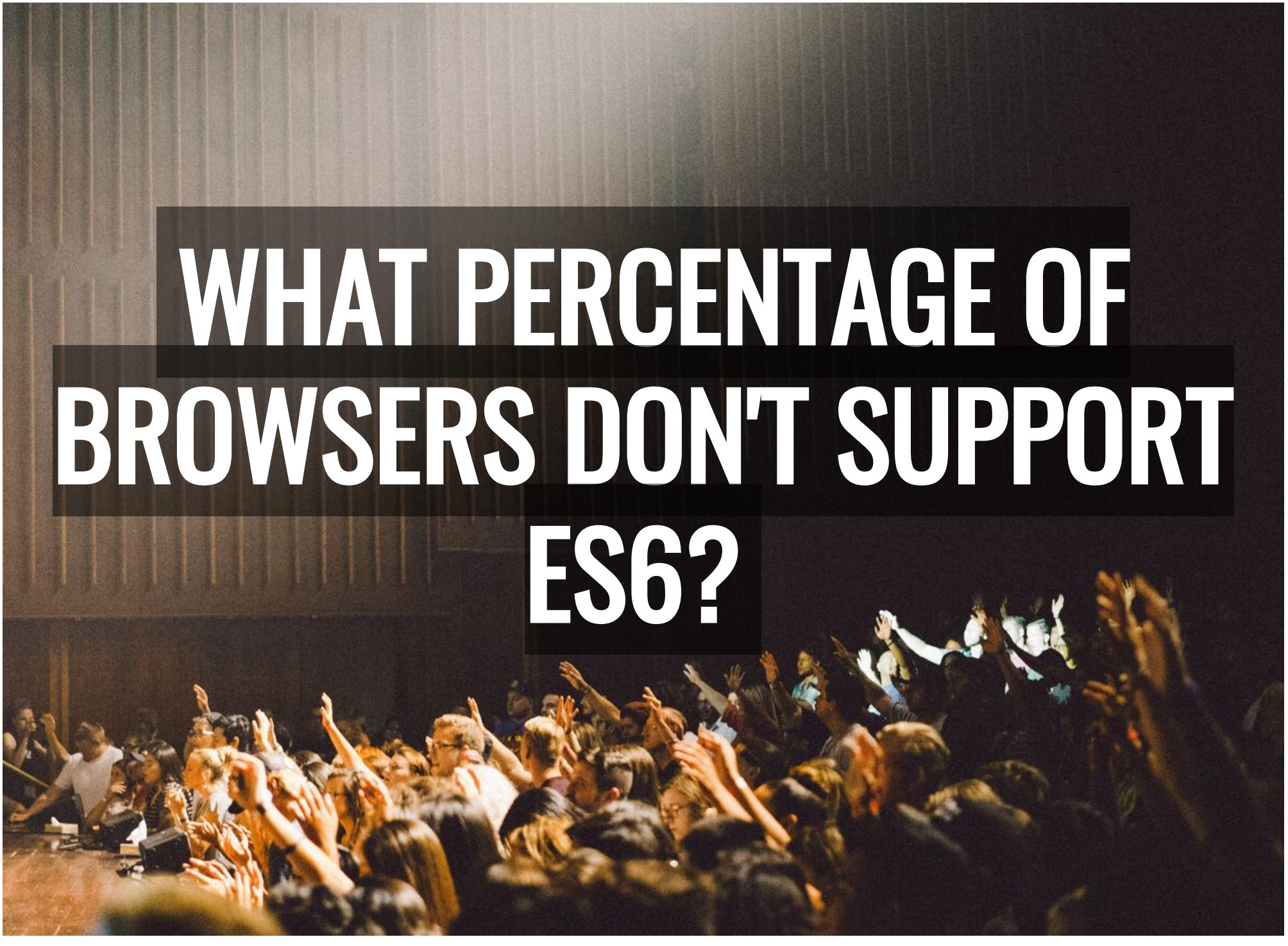

WHO USES ES6+ FEATURES?


WHO TRANSPILES ES6+ TO ES5?

Let's take a peek into transpiling...

babeljs.io/repl

Character counter


WHAT PERCENTAGE OF
BROWSERS DON'T SUPPORT
ES6?

The HTML `<script>` tag accepts helpful attributes:

- `type=module` - this is ES6+ code (and is automatically deferred)
- `nomodule` - don't execute in browsers that support ES6 modules

`<script>`: The Script element on MDN

`<script type=module>`

also includes support for...

- modules
- async/await
- Classes
- arrow functions
- fetch
- Promises
- Map, Set, and more!

Deploying ES2015+ Code in Production Today

~2017 major browsers started supporting `<script type=module>`

Safari 10.1*
Mar 2017

Chrome 61
Sep 2017

Firefox 60
May 2018

Edge 16
Oct 2017

* Needs a `nomodule` workaround

caniuse for `type=module`

<script type=module>

HTML element: script: type: module

Usage % of all users ?

All Web Site Data 97.32%

Global 88.11%

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baid Brows
		2-53													
	12-15	1 54-59	4-60	3.1-10	10-47	3.2-10.2							4-7.4		
6-10	16-17	60-69	61-77	10.1-12.1	48-63	10.3-13.1		2.1-4.4.4	12-12.1				8.2-9.2		
11	18	70	78	13	64	13.2	all	76	46	78	68	12.12	10.1	1.2	7.1
	76	71-72	79-81	TP		13.3									

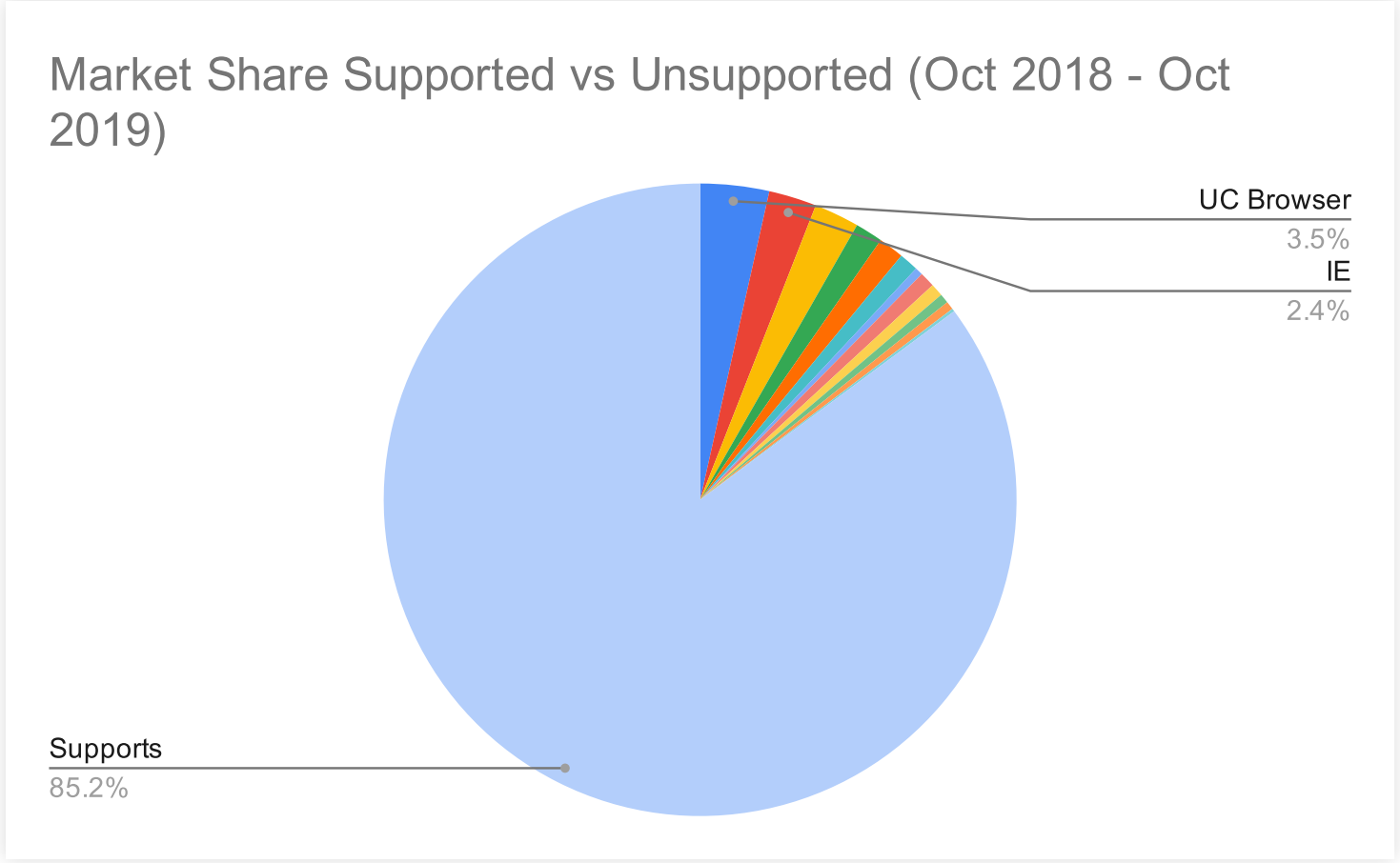
Notes | Feedback

1 Can be enabled by setting `dom.moduleScripts.enabled` to true

Support data for this feature provided by:

MDN browser-compat-data

caniuse for `type=module`



statcounter GlobalStats Browser Version Market Share Worldwide

IMPACT

Smaller size due to:

- reduced number of polyfills
- more efficient JS (less code)

Size Impact

Site/App	ES2015+ / ES5 size (minified)	ES2015+ / ES5 size (minified + gzipped)	Size savings
philipwalton.com	80K / 175K	21K / 43K	51-54%
jeremy.codes	34K / 112K	13K / 39K	66-70%
TodoMVC	8.4K / 11K		24%

Deploying ES2015+ Code in Production Today, Doing Differential Serving in 2019, Smart Bundling: How To Serve Legacy Code Only To Legacy Browsers

Time Impact

Smaller size = Reduced load, parse, & eval time

Deploying ES2015+ Code in Production Today, Doing Differential Serving in 2019, Smart Bundling: How To Serve Legacy Code Only To Legacy Browsers

Time Impact

Smaller size = Reduced load, parse, & eval time

Site/App	ES2015+ / ES5 size (minified)	ES2015+ / ES5 size (minified + gzipped)	Size savings	ES2015+ / ES5 parse/eval time (avg)	Parse/eval time savings
philipwalton.com	80K / 175K	21K / 43K	51-54%	172ms / 367ms	53%
jeremy.codes	34K / 112K	13K / 39K	66-70%	165ms / 466ms	65%
TodoMVC	8.4K / 11K		24%		

Deploying ES2015+ Code in Production Today, Doing Differential Serving in 2019, Smart Bundling: How To Serve Legacy Code Only To Legacy Browsers

WHY DO WE CARE ABOUT SIZE AND TIME?

> *53% of mobile sites are abandoned if pages take longer than 3 seconds to load.*

2016 report by Doubleclick by Google

> *Pinterest reduced load times by 40% and saw a 15% increase in sign ups.*

https://wpostats.com/

> *Starbucks implemented a 2x faster time to interactive resulting in a 65% increase in rewards registrations.*

Get Down to Business: Why the Web Matters (Chrome Dev Summit 2018)

> *Speed is now used as a ranking factor for mobile searches.*

https://developers.google.com/web/updates/2018/07/search-ads-speed

The internet consumes 416.2 TWh of electricity per year.
A 10% savings would be equivalent to:

How is your website impacting the planet?, Greenhouse Gas Equivalencies Calculator

The internet consumes 416.2 TWh of electricity per year. A 10% savings would be equivalent to:

- 6.2 million fewer cars on the road

How is your website impacting the planet?, Greenhouse Gas Equivalencies Calculator

The internet consumes 416.2 TWh of electricity per year.
A 10% savings would be equivalent to:

- 6.2 million fewer cars on the road
- 32 billion less pounds of coal being burned

How is your website impacting the planet?, Greenhouse Gas Equivalencies Calculator

The internet consumes 416.2 TWh of electricity per year.
A 10% savings would be equivalent to:

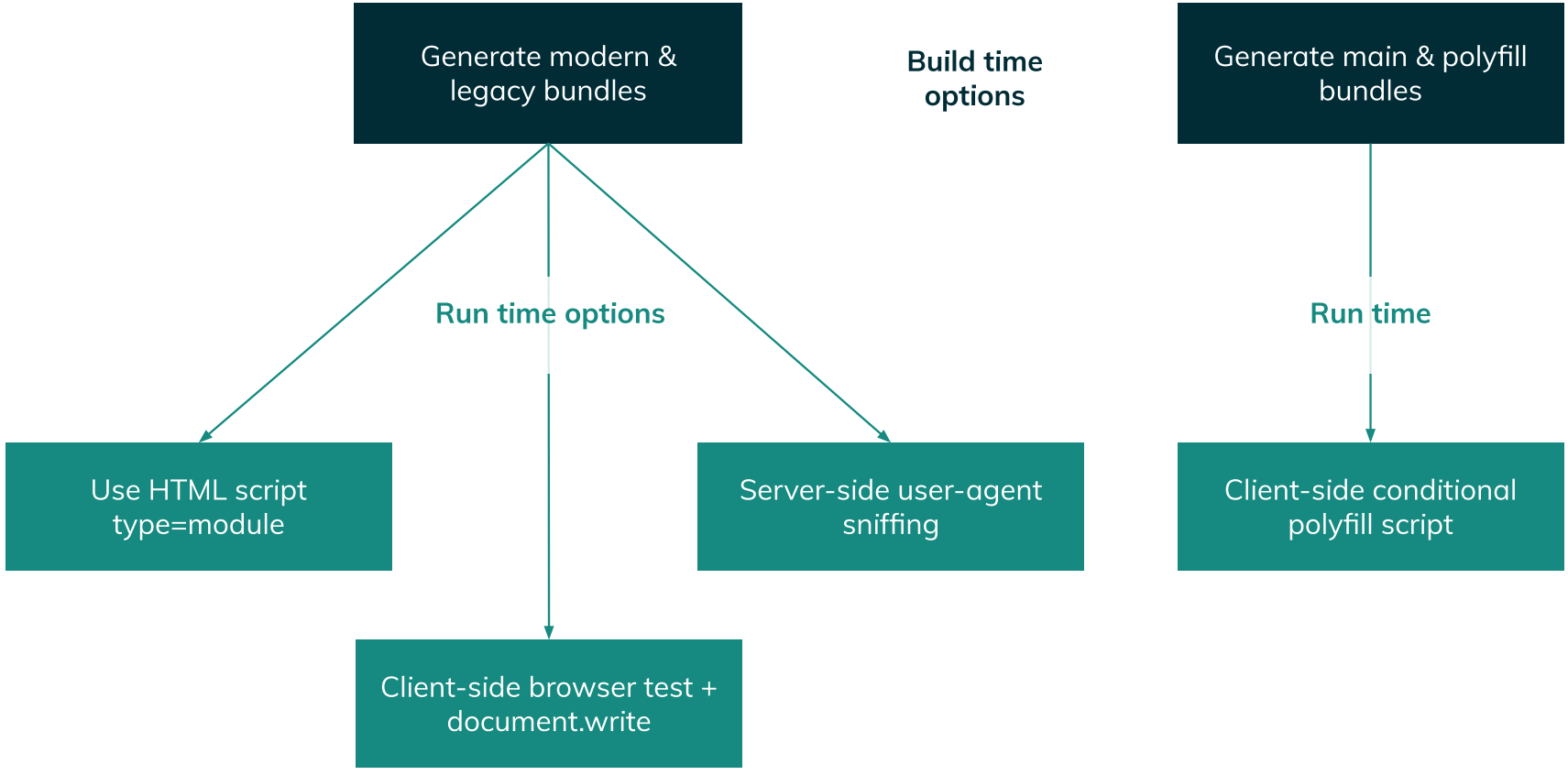
- 6.2 million fewer cars on the road
- 32 billion less pounds of coal being burned
- 486 million tree seedlings grown for 10 years

How is your website impacting the planet?, Greenhouse Gas Equivalencies Calculator

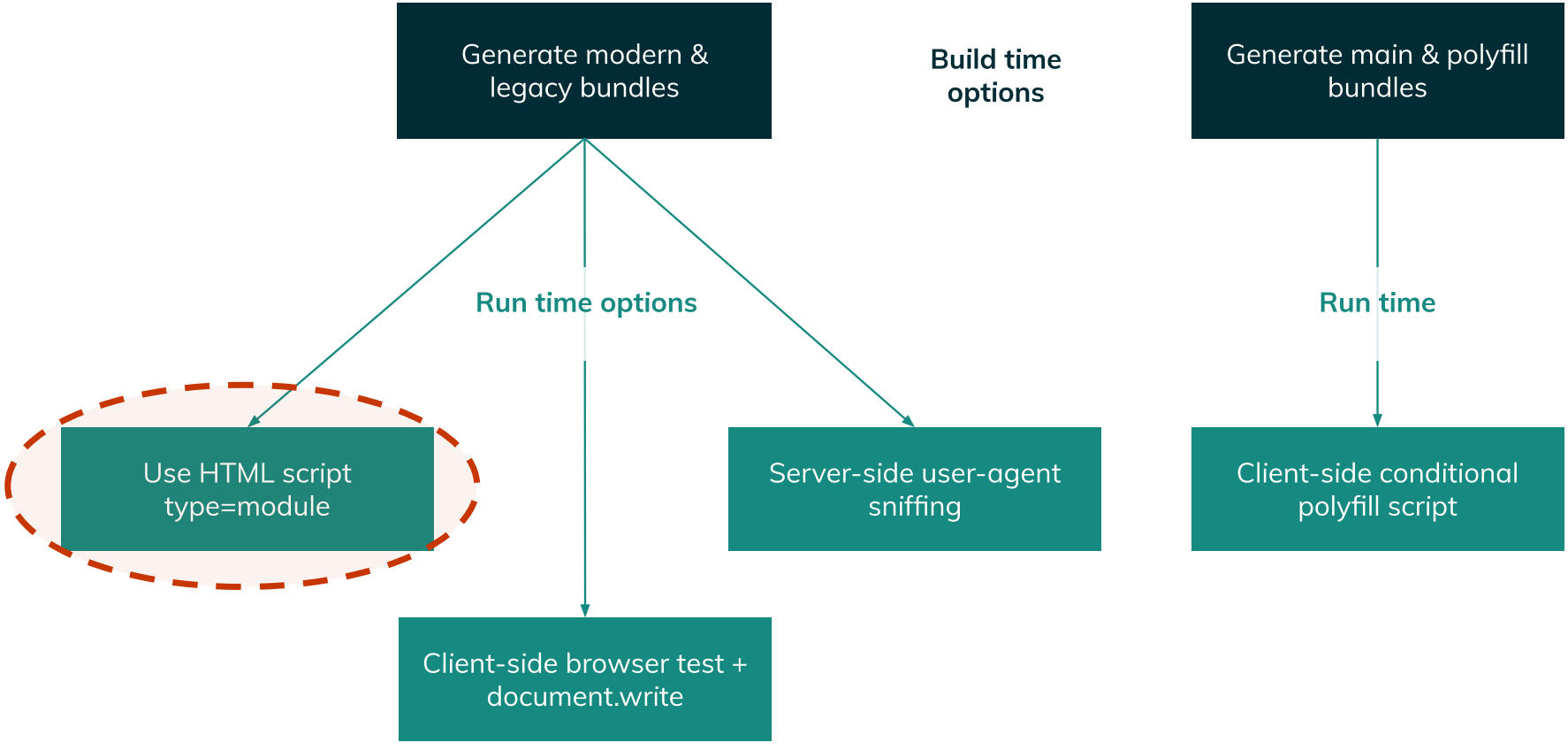
HOW DO WE BALANCE SPEED AND SUPPORT?

Differential serving is the concept of serving modern, ES6+ code to modern browsers and legacy, transpiled code to legacy browsers.

DIFFERENTIAL SERVING STRATEGIES



DIFFERENTIAL SERVING STRATEGIES



```
<!-- Browsers with ES module support load this file. -->
<script type="module" src="/modern.js"></script>

<!-- Older browsers load this file (and module-supporting -->
<!-- ones do not***). -->
<script nomodule src="/legacy.js" defer></script>
```

*** Oops

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)
- ✅ modern Chrome and Firefox only fetch once

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)
- ✅ modern Chrome and Firefox only fetch once
- ⚠️ Safari <11 may or may not double fetch (unclear trigger)

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)
- ✅ modern Chrome and Firefox only fetch once
- ⚠️ Safari <11 may or may not double fetch (unclear trigger)
- ⚠️ Safari 11+ may still double fetch in some cases (see bug)

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)
- ✅ modern Chrome and Firefox only fetch once
- ⚠️ Safari <11 may or may not double fetch (unclear trigger)
- ⚠️ Safari 11+ may still double fetch in some cases (see bug)
- ❌ pre-2018 browsers do double fetches

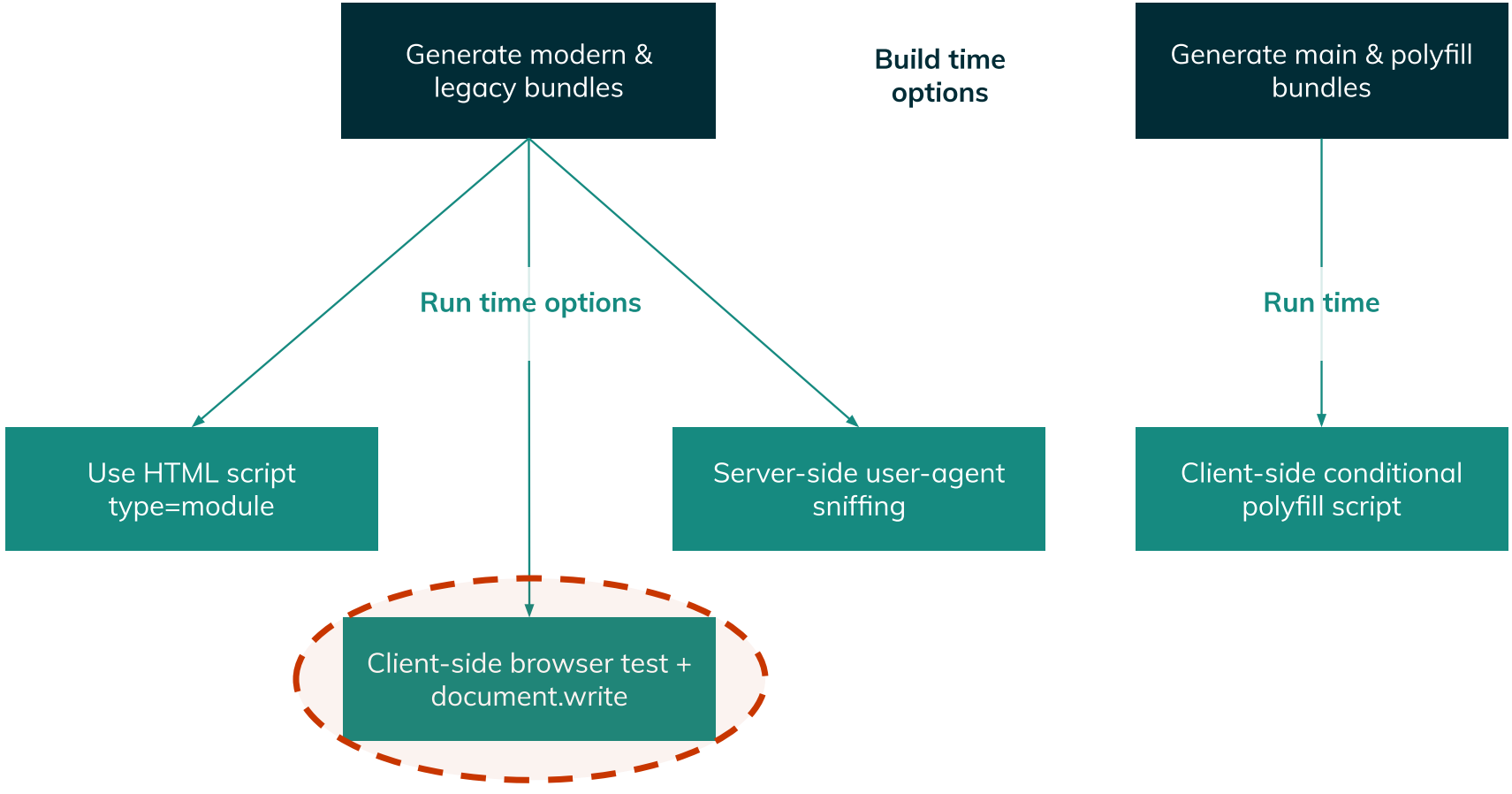
From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

*** Oops

- ✅ no browser executes twice (with the Safari hack)
- ✅ modern Chrome and Firefox only fetch once
- ⚠️ Safari <11 may or may not double fetch (unclear trigger)
- ⚠️ Safari 11+ may still double fetch in some cases (see bug)
- ❌ pre-2018 browsers do double fetches
- ❌❌ latest Edge does triple fetch (2x module + 1x nomodule)

From Will it double-fetch? Browser behavior with `module` / `nomodule` scripts by jakub-g
Safari hack, Safari bug

DIFFERENTIAL SERVING STRATEGIES



```
<script>
  var MODERN_BUNDLE = "assets/dist/js/scripts.modern.min.js";
  var LEGACY_BUNDLE = "assets/dist/js/scripts.min.js";

  function isModern() {
    try {
      new Function('import("")');
      return true;
    } catch (err) {
      return false;
    }
  }

  var scriptTag = document.createElement("script");
  scriptTag.setAttribute("src", isModern() ? MODERN_BUNDLE : LEGACY_BUNDL
  document.body.appendChild(scriptTag);
</script>
```

Example code from Should We All Start Implementing Differential Serving? by Alex MacArthur

What's wrong with this approach?

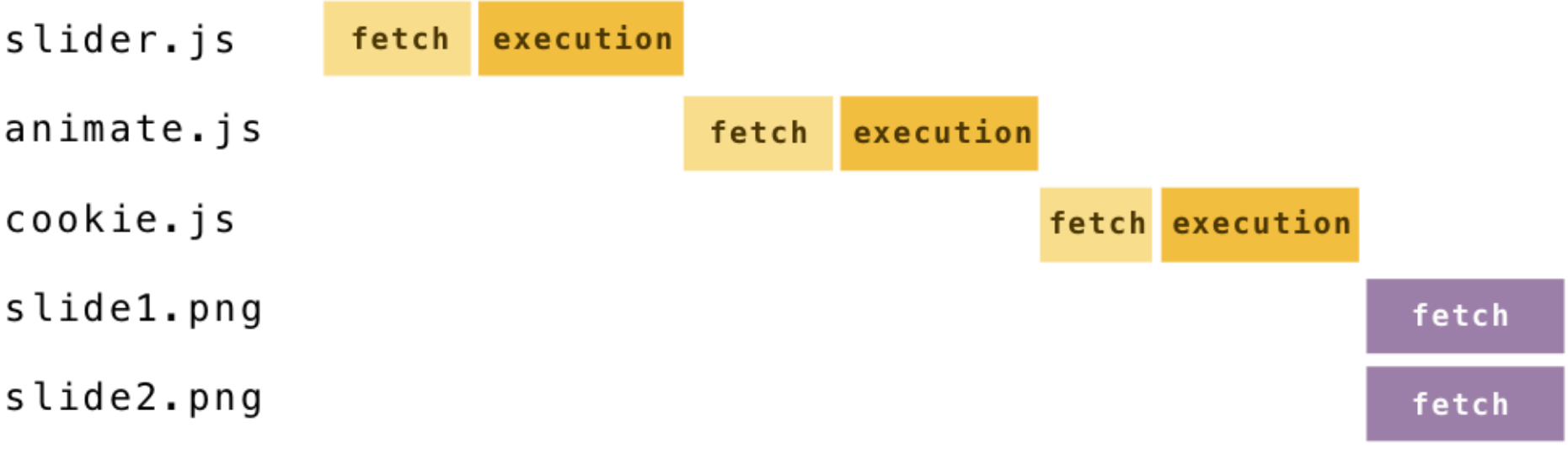
❌ Big latency performance penalty from breaking **speculative parsing / preload scanning**

Say we have some HTML like this:

```
<script src="slider.js"></script>
<script src="animate.js"></script>
<script src="cookie.js"></script>
<img src="slide1.png">
<img src="slide2.png">
```

Building the DOM faster: speculative parsing, async, defer and preload by Milica Mihajlija

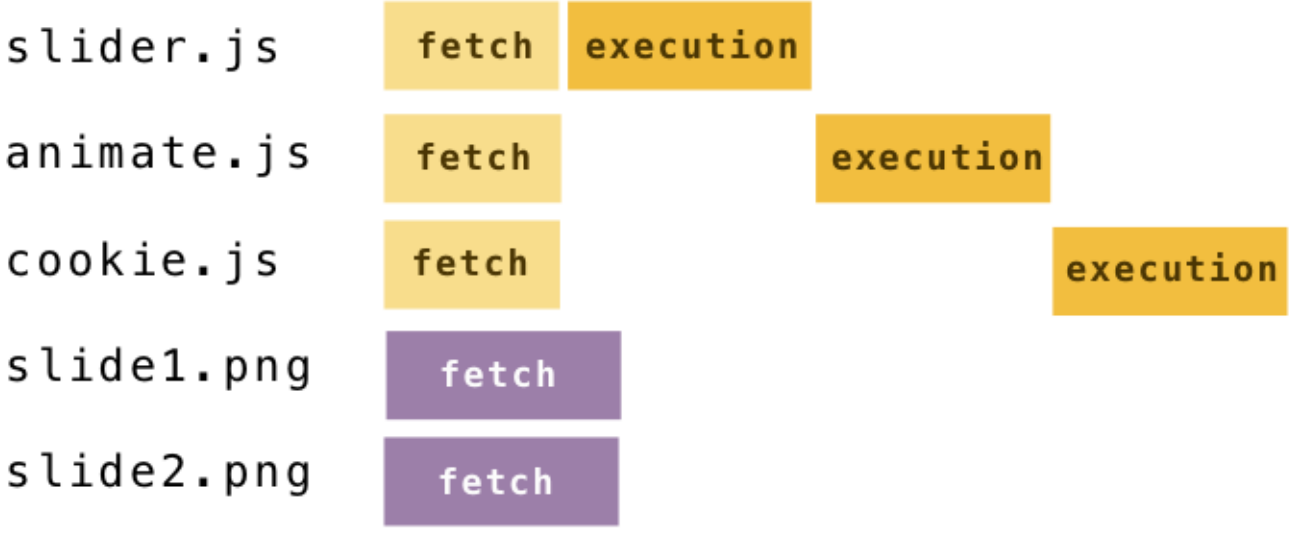
Pre-2008 HTML Parsing



Sequential script loading

Building the DOM faster: speculative parsing, async, defer and preload by Milica Mihajlija

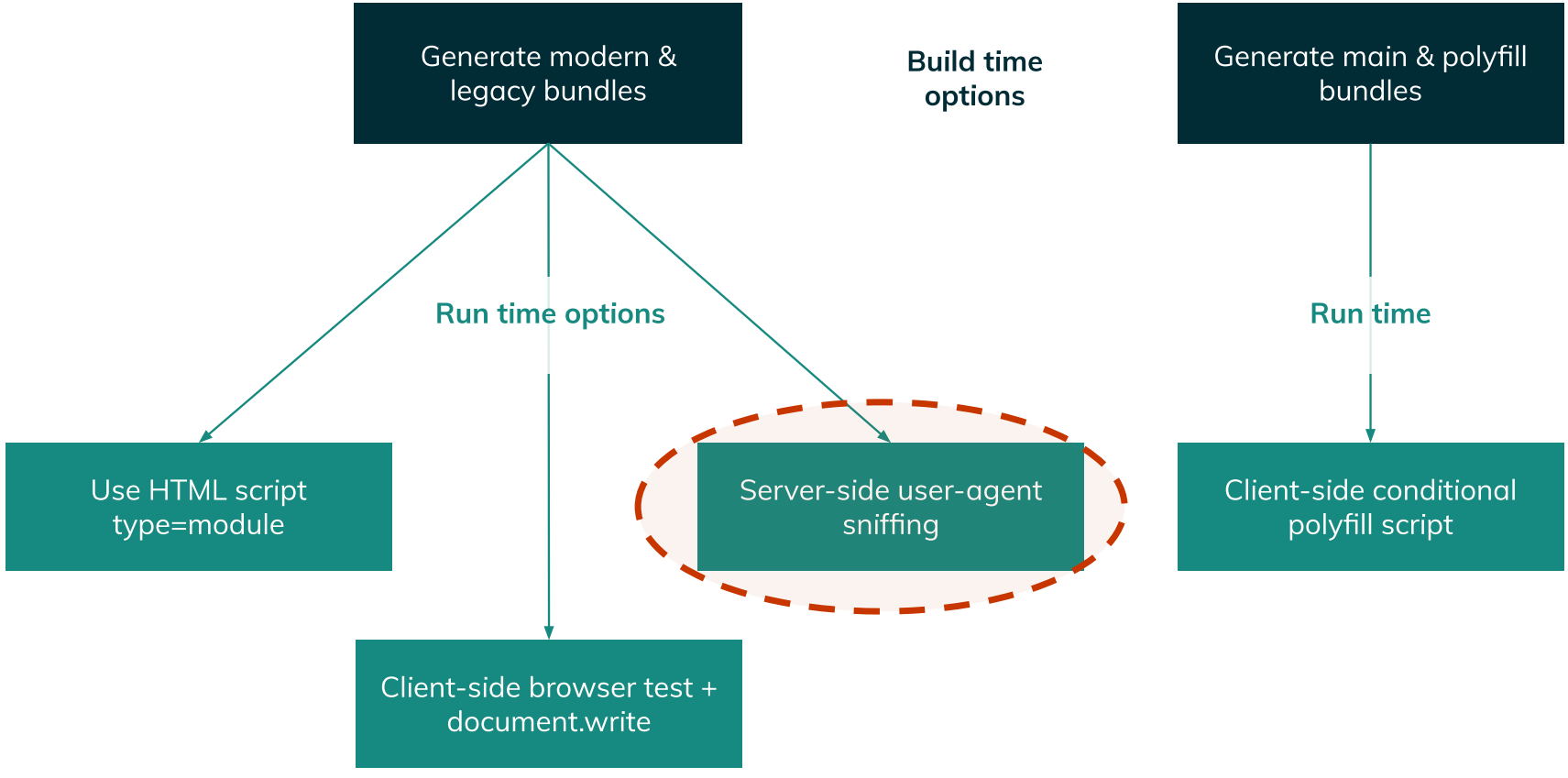
Modern HTML Parsing



Preloading resources with speculative parsing

Building the DOM faster: speculative parsing, async, defer and preload by Milica Mihajlija

DIFFERENTIAL SERVING STRATEGIES

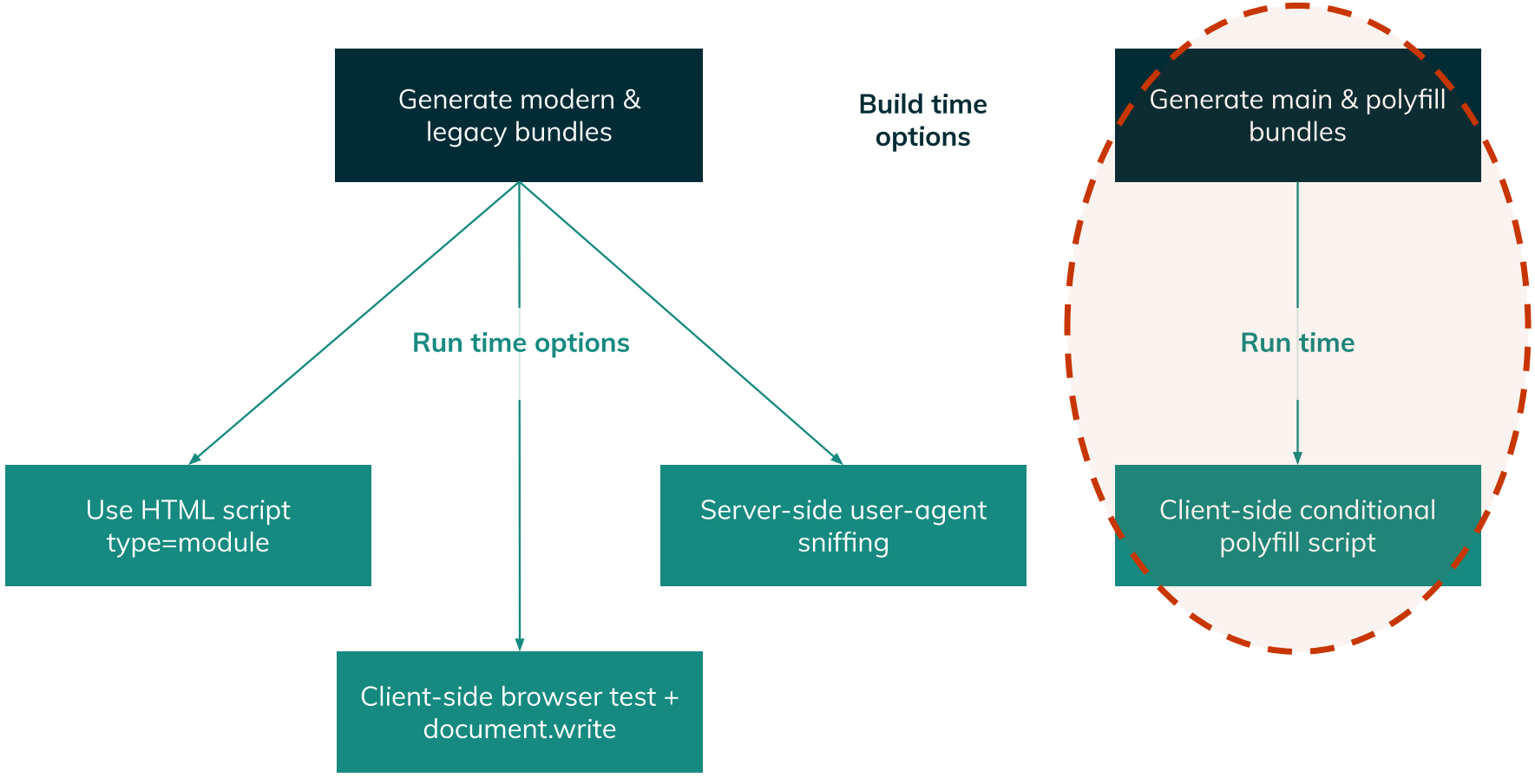


What's wrong with user-agent sniffing?

- ❌ Not trivial and prone to false classification
- ❌ New User Agent strings are added daily
- ❌ Can't be used for static deployments
- ❌ Bad for caching
- ❌ Mozilla will smite you

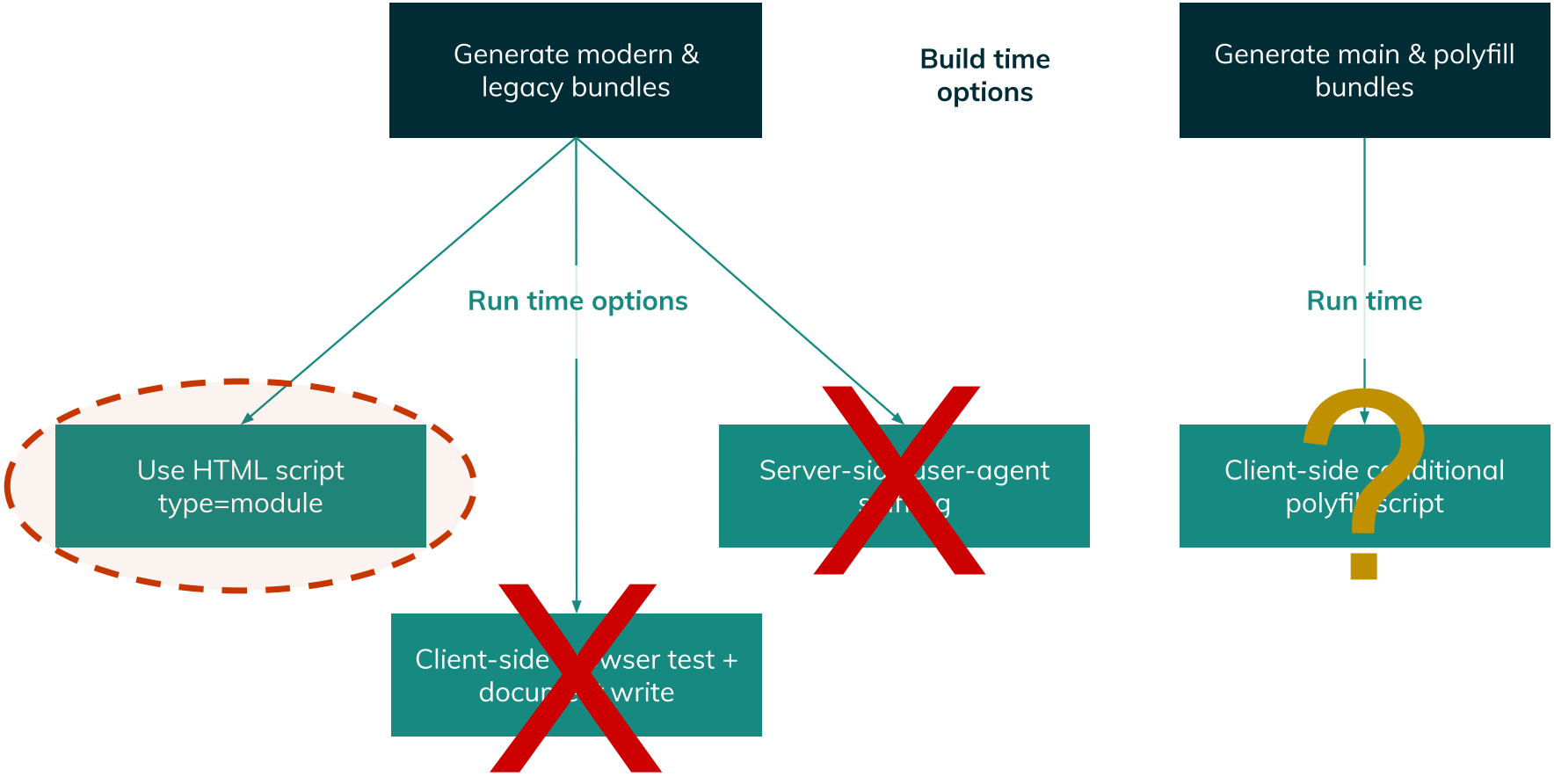
Modern Script Loading by Jason Miller, Smart Bundling: How To Serve Legacy Code Only To Legacy Browsers by Shubham Kanodia, Cloudfront may even remove the header

DIFFERENTIAL SERVING STRATEGIES



```
<!-- newer browsers won't load this bundle: -->
<script nomodule src="polyfills.js"></script>

<!-- all browsers load this one: -->
<script src="/bundle.js"></script>
```

DIFFERENTIAL SERVING STRATEGIES



Problematic for Safari 10.1, modern Edge, and older versions of all:

```
<!-- Browsers with ES module support load this file. -->
<script type="module" src="/modern.js"></script>

<!-- Older browsers load this file (and module-supporting -->
<!-- ones do not***). -->
<script nomodule src="/legacy.js" defer></script>
```

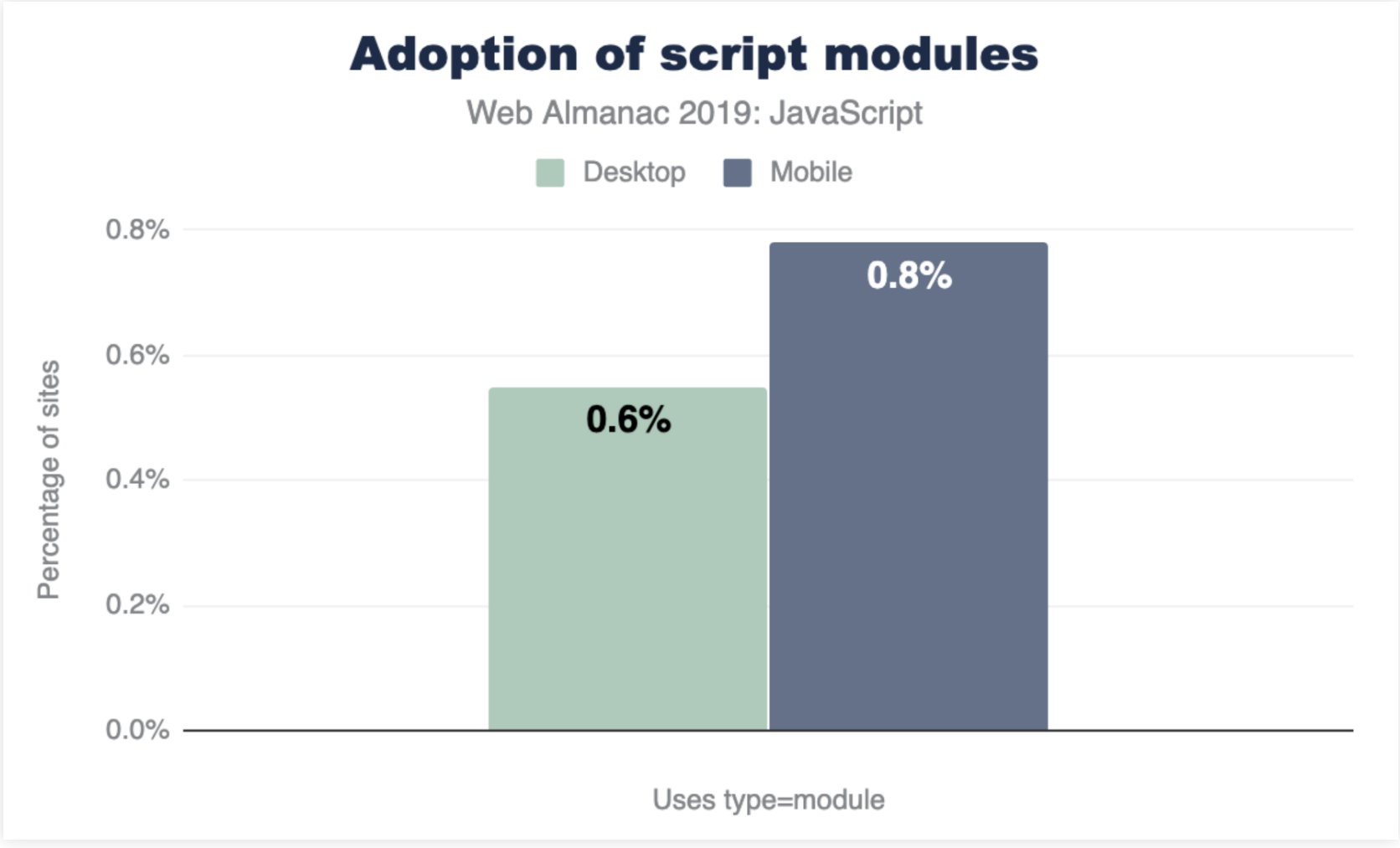
Problematic for ~~Safari 10.1~~, modern Edge, and older versions of all:

```
<!-- polyfill `nomodule` in Safari 10.1: -->
<script type=module>
!function(e,t,n){!("noModule"in(t=e.createElement("script")))&&
  "onbeforeload"in t&&(n=!1,e.addEventListener("beforeload",function(e){
  if(e.target===t)n=!0;else if(!e.target.hasAttribute("nomodule")||!n)
  return;e.preventDefault()},!0),t.type="module",t.src=".",
  e.head.appendChild(t),t.remove())}(document)
</script>

<!-- Browsers with ES module support load this file. -->
<script type="module" src="/modern.js"></script>

<!-- Older browsers load this file (and module-supporting -->
<!-- ones do not***). -->
<script nomodule src="/legacy.js" defer></script>
```

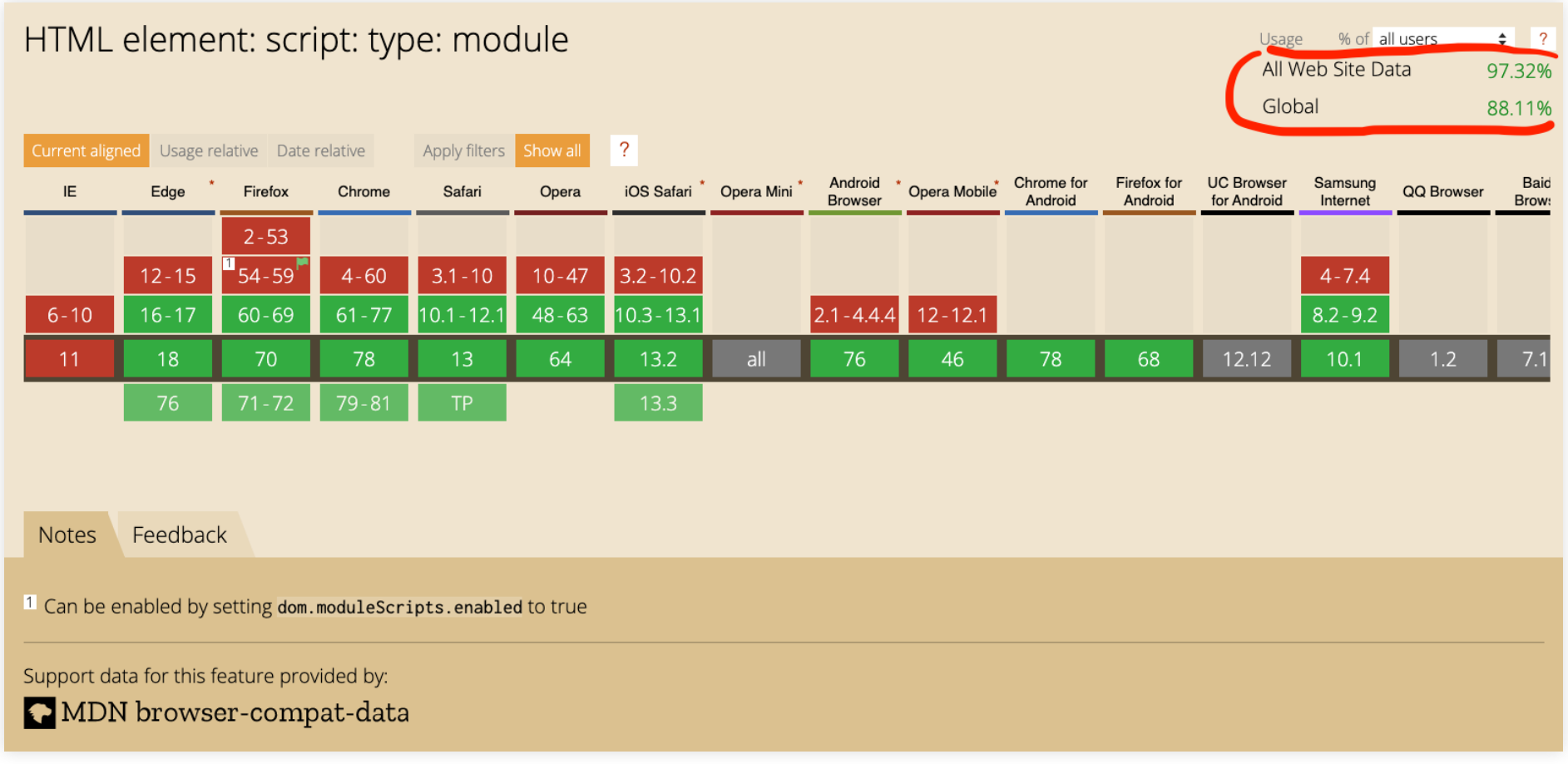
"Problematic" for ~~Safari 10.1~~, modern Edge, and older versions of all:

```
<!-- polyfill `nomodule` in Safari 10.1: -->
<script type=module>
!function(e,t,n){!("noModule"in(t=e.createElement("script")))&&
  "onbeforeload"in t&&(n=!1,e.addEventListener("beforeload",function(e){
  if(e.target===t)n=!0;else if(!e.target.hasAttribute("nomodule")||!n)
  return;e.preventDefault()},!0),t.type="module",t.src=".",
  e.head.appendChild(t),t.remove())}(document)
</script>

<!-- Browsers with ES module support load this file. -->
<script type="module" src="/modern.js"></script>

<!-- Older browsers load this file (and module-supporting -->
<!-- ones do not***). -->
<script nomodule src="/legacy.js" defer></script>
```

So how many people are using modules?

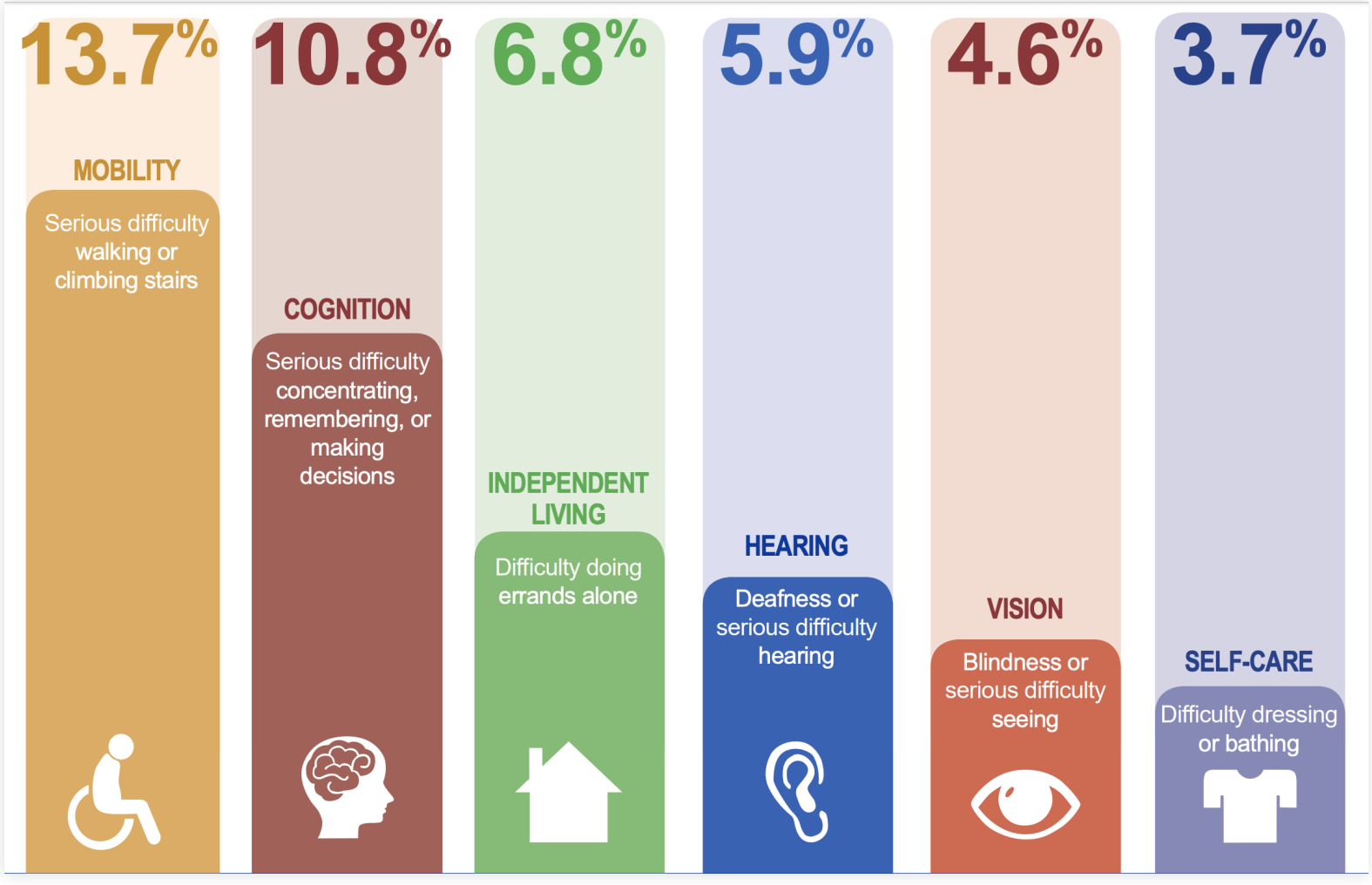


Web Almanac by HTTP Archive - JavaScript chapter



caniuse for `type=module`


61 MILLION, OR 26% OF ADULTS IN THE U.S. LIVE WITH A DISABILITY
Donate to Team Gleason



CDC: Disability Impacts All of Us

> *The WCAG failure rate for home pages was at least 97.8%.*

The WebAIM Million: What we learned analyzing 1,000,000 web site home pages


WHY ARE WE PRIORITIZING
BACKWARD COMPATIBILITY
OVER ACCESSIBILITY?


THANKS!
Resources, and more at bit.ly/vintagebundles

HANDS-ON WALK THROUGH

Start with the `4-code-splitting` branch of
github.com/siakaramalegos/markdown-viewer

Generating modern + legacy bundles

- 2 different Browserslist configs, which feed into
- 2 different Babel configs, which feed into
- 2 different Webpack configs

What's Babel + Browserslist doing?

1. Run Bundle Analyzer on prod to see our starting bundle size. Run the coverage checker in DevTools to see some of the unused code. Much of this is polyfills.
2. Update `.babelrc` to add debug:

```
{
 "presets": [
-    "@babel/preset-env",
+    [
+      "@babel/preset-env",
+      {
+        "debug": true
+      }
+    ]
 ],
 "plugins": [
   "@babel/plugin-syntax-dynamic-import",
```

What's Babel doing?

- Run `npm start` and look for the targets in terminal log:

```
Using targets:
{
  "android": "4.4.3",
  "chrome": "49",
  "edge": "17",
  "firefox": "65",
  "ie": "11",
  "ios": "10.3",
  "opera": "58",
  "safari": "5.1",
  "samsung": "4"
}
```

- Notice this:

```
Using polyfills: No polyfills were added, since the `useBuiltIns`
option was not set.
```

`useBuiltIns` docs

Add polyfills

1. Add core-js for polyfills

```
$ npm install --save core-js@3.4.7
```

2. Add core-js to the entry point

```
// src/index.js
import 'core-js';
```

3. Limit the polyfills to only match the targeted browsers:

```
    {
      "debug": true
+      "useBuiltIns": "entry"
+      "corejs": "3.4.7"
    }
```

"useBuiltIns": "entry"

Our import gets replaced with imports needed for our target browsers. For example, for Chrome 71:

```
// src/index.js
import 'core-js';
```

changes to:

```
// src/index.js
import "core-js/modules/es.array.unscopables.flat";
import "core-js/modules/es.array.unscopables.flat-map";
import "core-js/modules/es.object.from-entries";
import "core-js/modules/web.immediate";
```

core-js

Better polyfills

1. Limit the polyfills to only match those used:

```
-       "useBuiltIns": "entry"
+       "useBuiltIns": "usage"
```

2. Remove `import 'core-js';` from **src/index.js**
3. How did the bundle change?

Differential Serving: Setting up

1. Add new dependencies:

```
npm i --save-dev webpack-merge script-ext-html-webpack-plugin
```

2. Create **webpack.common.js** and move most of our config there except the JS rules and the `output` property.
3. In **webpack.config.js**, import merge and common:

```
const merge = require('webpack-merge')
const common = require('./webpack.common.js')
```

4. Merge it with our existing `output` and JS rules to make sure it works before moving forward.

Create a **legacy** Babel config based off of our **.babelrc**, and then delete **.babelrc**:

```
// babel.legacy.js
module.exports = {
  presets: [
    [
      "@babel/preset-env",
      {
        useBuiltIns: "usage",
        corejs: "3.4.7",
      }
    ]
  ],
  plugins: [
    "@babel/plugin-syntax-dynamic-import",
    "@babel/plugin-transform-runtime"
  ]
}
```

Create a **modern** Babel config targeting ES modules:

```
// babel.modern.js
module.exports = {
  presets: [
    [
      "@babel/preset-env",
      {
        modules: false,
        targets: { esmodules: true }
      }
    ]
  ],
    plugins: [
      "@babel/plugin-syntax-dynamic-import",
      "@babel/plugin-transform-runtime"
    ]
}
```

`@babel/preset-env` docs for `modules` and `targets.esmodules`

Create separate webpack configs for modern and legacy in **webpack.config.js**:

```
const legacyConfig = merge(common, {
  output: {
    filename: '[name].js',
    path: path.resolve('./dist')
  },
  module: {
    rules: [
      {
        test: /\.m?js$/,
        exclude: [/node_modules/],
        use: {
          loader: 'babel-loader',
          options: babelLegacy,
        }
      },
    ],
  },
  optimization: {
```

```
const modernConfig = merge(common, {
  output: {
    filename: '[name].mjs',
    path: path.resolve('./dist')
  },
  module: {
    rules: [
      {
        test: /\.m?js$/,
        exclude: [/node_modules/],
        use: {
          loader: 'babel-loader',
          options: babelModern,
        }
      },
    ],
  },
  optimization: {
```

Export them both as an array for webpack to build both:

```
// webpack.config.js
module.exports = [ legacyConfig, modernConfig ]
```

Tada! Oops...

Add the scripts to the html!

1. Manually add in our modern script at the bottom of **src/index.html**

```
<script type="module" src="main.mjs"></script>
```

2. Add a plugin to generate the correct script tag for our legacy script into **webpack.common.js**

```
const ScriptExtHtmlWebpackPlugin = require("script-ext-html-webpack-plugin
```

```
// add to plugins array:
new ScriptExtHtmlWebpackPlugin({
 module: /\.mjs$/,
 custom: [
   {
     test: /\.js$/,
     attribute: 'nomodule',
     value: ''
   },
 ]
})
```