WebAssembly in the JavaScript Ecosystem

Unlocking new possibilities with hybrid tooling

WebAssembly Chicago

May 24, 2022

Nate Moore

Astro co-creator and core team member.

Creator of Microsite, TokenCSS, and many other open source projects.

Senior Software Engineer at The Astro Technology Company.

natemoo-re

n_moore

nmoo.dev

The Third Age of JavaScript [1]

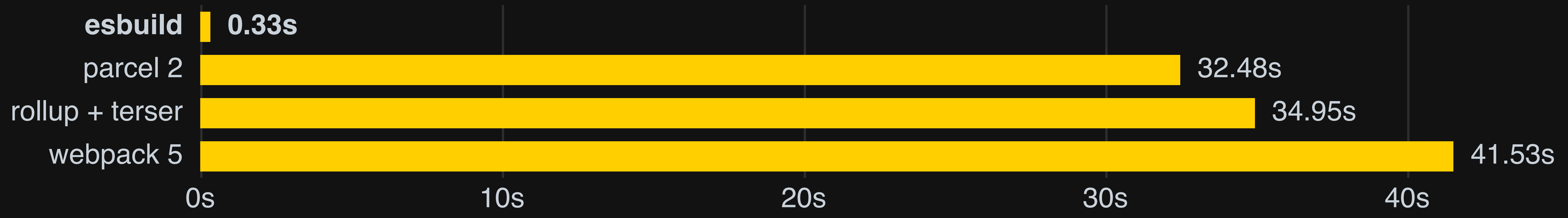
JavaScript grows from a site scripting toy language to full application platform

- Faster tooling
- ESM first
- Single tools that do many things well
- Typesafe
- More secure
- Polyglot (Native, but increasingly **WebAssembly**)

1. As described by @swyx ↩️

The Third Age of JavaScript [1]

JavaScript grows from a site scripting toy language to full application platform



1. As described by @swyx ↩️

WebAssembly: The JavaScript Killer

WebAssembly: The JavaScript Killer?

“The reports of my death are greatly exaggerated.”

WebAssembly + JavaScript

Name a more iconic duo!

> WebAssembly is a different language from JavaScript, but it is not intended as a replacement.
>
> Instead, it is designed to complement and work alongside JavaScript, allowing web developers to take advantage of both languages' strong points.
>
> – WebAssembly Concepts, MDN

WebAssembly + JavaScript

Name a more iconic duo!

JS

- High-level
- No compilation
- Flexible, expressive
- Ecosystem

Wasm

- Low-level
- Compile target
- Near-native performance
- Portable

“Hybrid” Tooling

Takes full advantage of both JavaScript and WebAssembly’s strengths

Traits

- JavaScript/TypeScript module on surface
- Seamless integration with Node/Deno ecosystem
- Offers high-level, user-friendly APIs
- Low barrier to entry

- WebAssembly powers low-level, internal APIs
- Focus on compute-heavy tasks (parsing, compiling)
- Delivers near-native performance

“Hybrid” Tooling

Takes full advantage of both JavaScript and WebAssembly’s strengths

Benefits

- Built on web standards
- Compile a single `.wasm` file
- Shared data primitives
- Fully portable (Node, Deno, Browsers)
- Bridge ecosystem gaps

Tradeoffs

- New workflow, out of comfort zone
- Immature tooling
- Performance, `near-native ≠ native`
- Context-switching for 3 languages
- Maintenance cost, harder to contribute

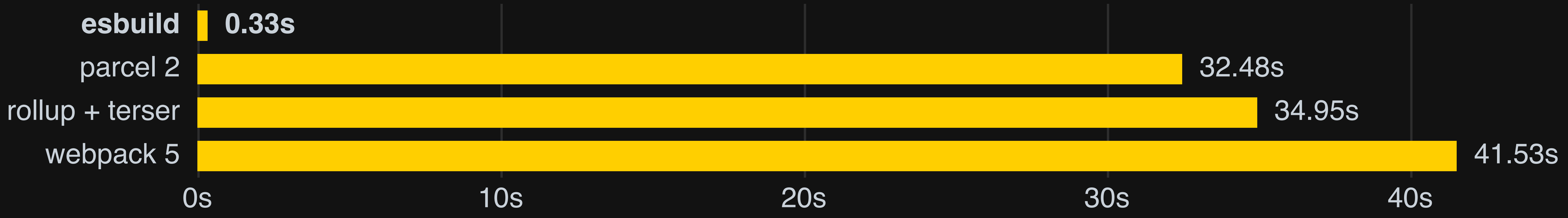
Examples

`esbuild`

Example

An extremely fast JavaScript bundler

esbuild.github.io



- Written in Go
- Includes JS and CSS parsers and compilers, fully featured JS bundler
- Powers next-gen build tools like Vite
- Native bindings for Node and Deno, `.wasm` bindings for the web (automatic in StackBlitz)

> The WebAssembly version is much, much slower than the native version. In many cases it is an order of magnitude (i.e. 10x) slower. This is for various reasons including a) node re-compiles the WebAssembly code from scratch on every run, b) Go's WebAssembly compilation approach is single-threaded, and c) node has WebAssembly bugs that can delay the exiting of the process by many seconds.

`esbuild-android-64` `esbuild-android-arm64` `esbuild-darwin-64`
`esbuild-darwin-arm64` `esbuild-freebsd-64` `esbuild-freebsd-arm64`
`esbuild-linux-32` `esbuild-linux-64` `esbuild-linux-arm`
`esbuild-linux-arm64` `esbuild-linux-mips64le` `esbuild-linux-ppc64le`
`esbuild-linux-riscv64` `esbuild-linux-s390x` `esbuild-netbsd-64`
`esbuild-openbsd-64` `esbuild-sunos-64` `esbuild-windows-32`
`esbuild-windows-64` `esbuild-windows-arm64`

`esbuild.wasm`

`es-module-lexer`

Example

Low-overhead lexer dedicated to ES module parsing for fast analysis

guybedford/es-module-lexer

```
import { init, parse } from 'es-module-lexer';

await init;

const [imports, exports] = parse(`
  import { name } from "mod";
  export const data = { a: 0 };
`);
```

- Written in C
- Spec-compliant JavaScript lexer
- Extracts static `import` statements, `export` statements, and dynamic `import()` usage
- Claims speed of about **5ms per MB**

> Angular 1 (720KiB) is fully parsed in 5ms, in comparison to the fastest JS parser, Acorn which takes over 100ms.

`shiki`

Example

A beautiful Syntax Highlighter

shiki.matsu.io

```js
import shiki from 'shiki';

const highlighter = await shiki.getHighlighter({ theme: 'github-dark' });
const code = highlighter.codeToHtml(/* source */, { lang: 'js' });
```

- High-level JS library, but dependency is written in C
- First JavaScript syntax highlighter with editor-level fidelity (matches Visual Studio Code)
- Works with TextMate grammars and Code themes
- Powered by `vscode-oniguruma`, WebAssembly bindings for `oniguruma` RegExp library

`goldmark`

Example

A very fast Markdown compiler for Deno

deno.land/x/goldmark

```
import { init, transform } from "https://deno.land/x/goldmark/mod.ts";

await init();
const markdown = await Deno.readTextFile(new URL('./content.md', import.meta.url));
const { frontmatter, content } = await transform(markdown, /* opts */);
```

- Written in Go
- WebAssembly bindings for Goldmark, a CommonMark-compliant Markdown parser
- Original package powers Hugo
- Deno's adherence to web platform spec makes working with WebAssembly easy

> Sampling **100,000 runs** completed in **58s** with an average run of **0.57ms** per file.

`@parcel/css`

Example

A CSS parser, transformer, and minifier written in Rust

parcel-bundler/parcel-css

Input

```css
@custom-media --modern (color), (hover);

.foo {
  background: yellow;

  -webkit-border-radius: 2px;
  border-radius: 2px;
  -webkit-transition: background 200ms;
  transition: background 200ms;

  &.bar {
    color: green;
  }
}

@media (--modern) and (width > 1024px) {
  .a {
    color: green;
  }
}
```

Output

```css
.foo {
  background: #ff0;
  border-radius: 2px;
  transition: background 0.2s;
}
.foo.bar {
  color: green;
}
@media ((color) or (hover)) and (min-width: 1024px) {
  .a {
    color: green;
  }
}
```

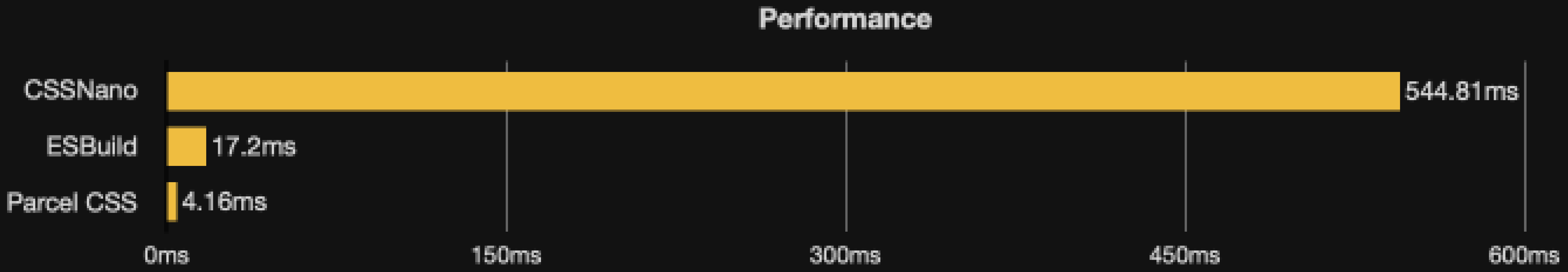
`@parcel/css`

Example

A CSS parser, transformer, and minifier written in Rust

parcel-bundler/parcel-css

- Written in Rust
- Lowers modern CSS syntax to be compatible with configurable browser targets
- **Browser-grade** CSS parser, powered by Mozilla’s own `cssparser` and `selectors` crates



`astro`

A website build tool for the modern web

Example

withastro/astro

`astro`

A website build tool for the modern web

withastro/astro

Astro DSL

```
---
import Layout from '../components/Layout.astro';
import Counter from '../components/Layout.tsx'; // or vue, svelte, etc
const { items } = await fetch('https://service.dev/api/v1/items').then(res => res.json());
---

<Layout title="Items">
  <ul>
    {items.map(item => <li>{item}</li>)}
  </ul>

  <Counter slot="footer" client:idle />
</Layout>

<style>
  ul {
    color: red;
  }
</style>
```

`astro`

Example

A website build tool for the modern web

withastro/astro

- High-level JS library, but compiler is written in Go
- Features custom `.astro` DSL for building server-side websites
- Truly Hybrid: leverages WebAssembly for compute-heavy tasks, everything else is JavaScript
- Why Go?
  - Iteration speed
  - Proximity to `esbuild`
  - Superset of HTML, leverage Go's `std` library
- Ability to communicate between JS <=> Wasm

`astro`

A website build tool for the modern web

Example

withastro/astro

JavaScript

```
import { transform } from '@astrojs/compiler';

const result = await transform(`...`, {
  async preprocessStyle(value, attributes) {
    const newValue = await compileSass(value);
    return newValue;
  }
});
```

Go

```
func preprocessStyle(i int, style *astro.Node, transformOptions transform.TransformOptions, cb func()) {
    defer cb()
    attrs := wasm_utils.GetAttrs(style)
    data, _ := wasm_utils.Await(transformOptions.PreprocessStyle.(js.Value).Invoke(style.FirstChild.Data, attrs))
    return data[0].Get("code").String()
}
```

WebAssembly doesn’t replace JavaScript

Use the best tool (from any ecosystem) for the job

Thank you!

Slides can be found at nmoo.dev/slides