WebAssembly: The Future of JS and a Multi-Language Web

About Me

- Developer 🥑 (Advocate) at Cloudflare
- IoT Author/Addict
- Autistic/Agender/Intersex (they/their/them pronouns)
- Interested in the future of the web

About Mx. Kassian Wren

- Developer 🥑 (Advocate) at Cloudflare
- IoT Author/Addict
- Autistic/Agender/Intersex (they/their/them pronouns)
- Interested in the future of the web

What even is WebAssembly?

I find it’s better to start with what WebAssembly isn’t before diving into what it is

What WebAssembly Isn’t

- *just* a programming language or instruction set (you CAN write WebAssembly, but the point is to use a language that compiles to it.)
- The death of JS (probably)
- Something you can just ignore until it goes away
- An out-of-reach experimental feature that can’t be used in the present
- Something you have to install or have your users install

What WebAssembly is

- A compilation target for other languages, allowing them to run in browsers and on the web
- An augmentation of the abilities of web developers by allowing code that is not only from other languages, but compile-optimized and performant (within reason)
- A huge evolution of the capabilities of JS developers to bring in modules from other languages to the web

MAGIC(K)

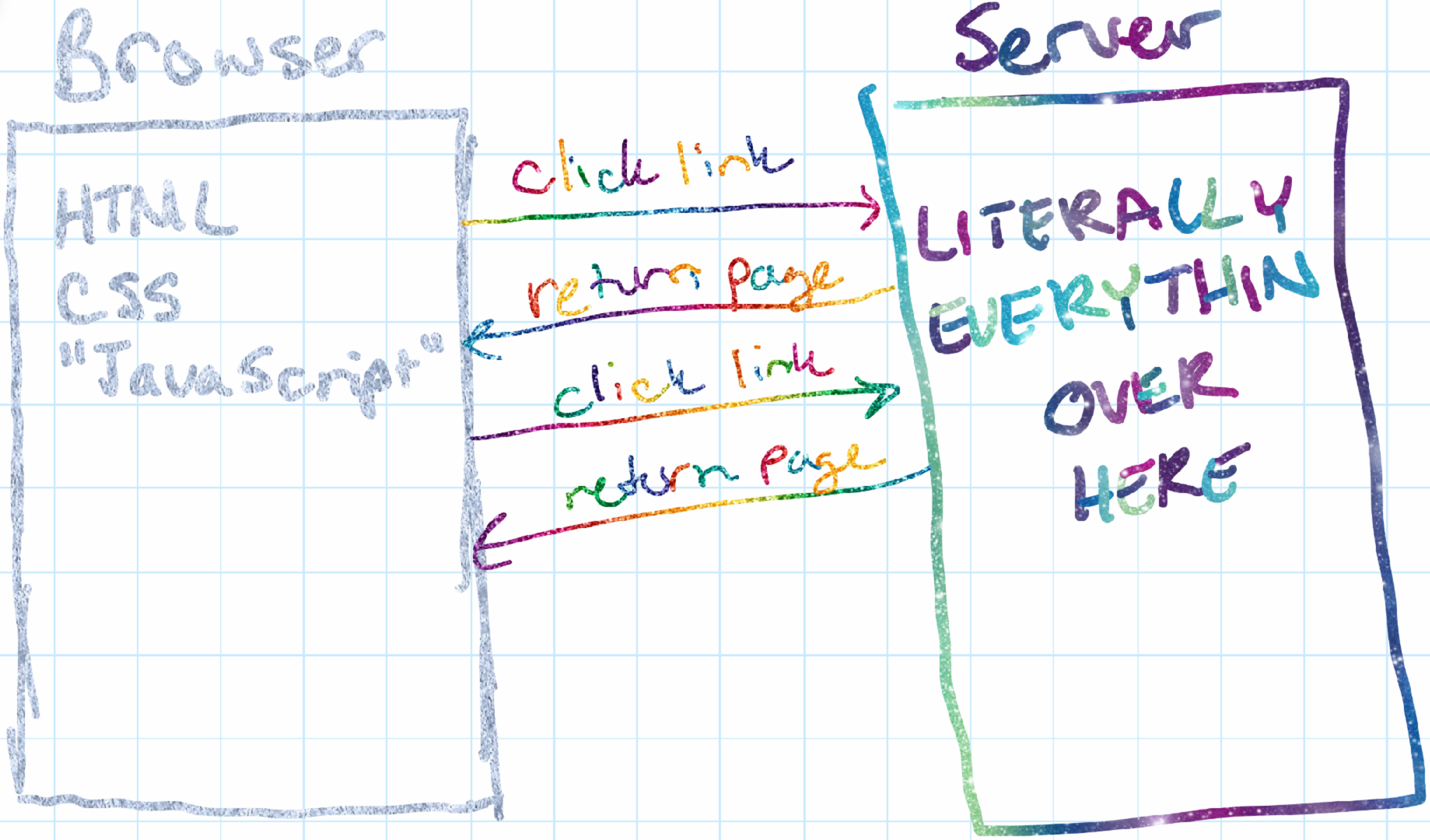
* — Not technically but hang with me.

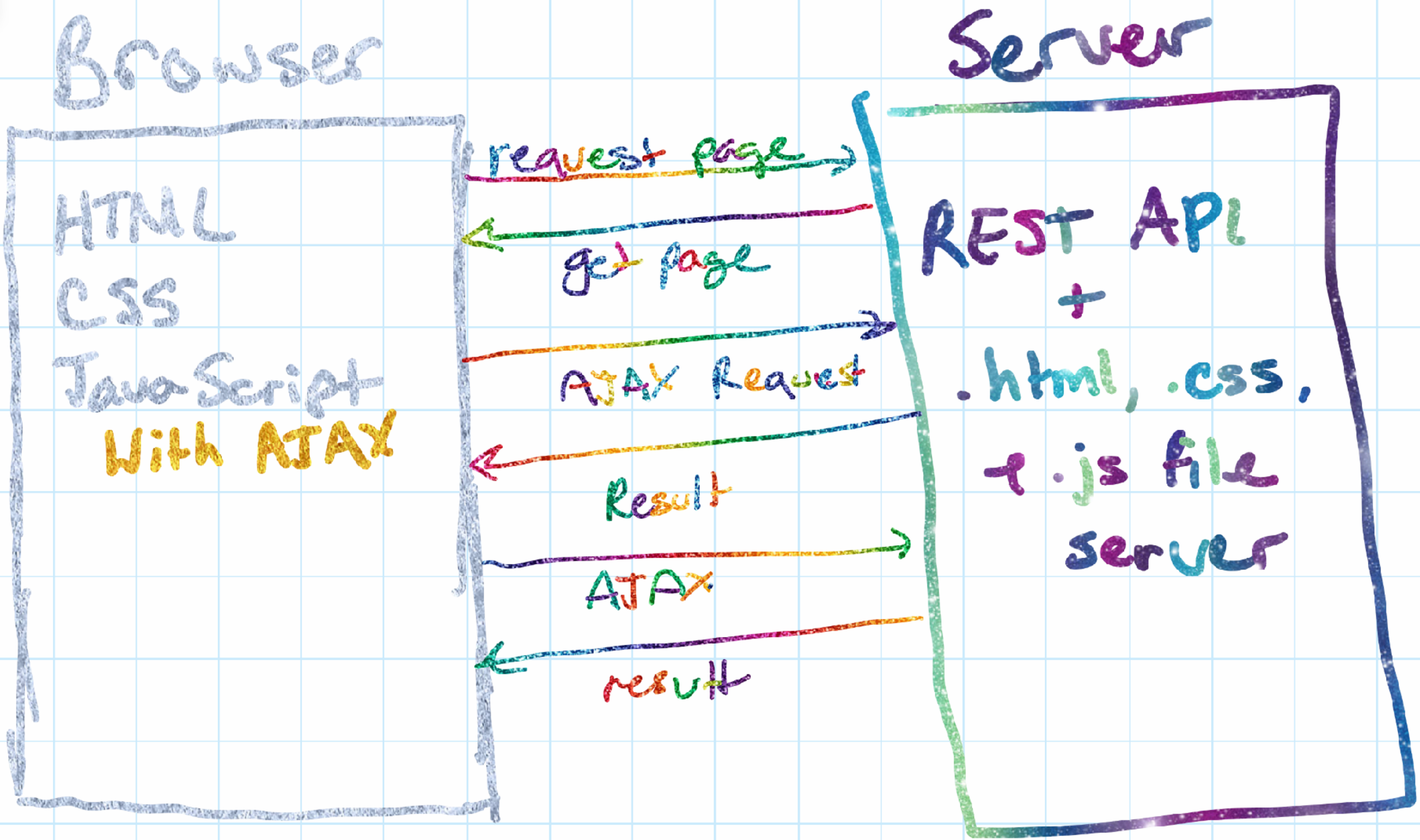
But then the question is…

WHAT
npm

WHY
npm

Let’s take a walk through the
history of the web.


Browser
HTML
CSS
"JavaScript"
click link
return page
click link
return page
Server
LITERALLY
EVERYTHIN
OVER
HERE


Browser
HTML
CSS
JavaScript
With AJAX
request page
get page
AJAX Request
Result
AJAX
result
Server
REST API
+
.html, .css,
+ .js file
server

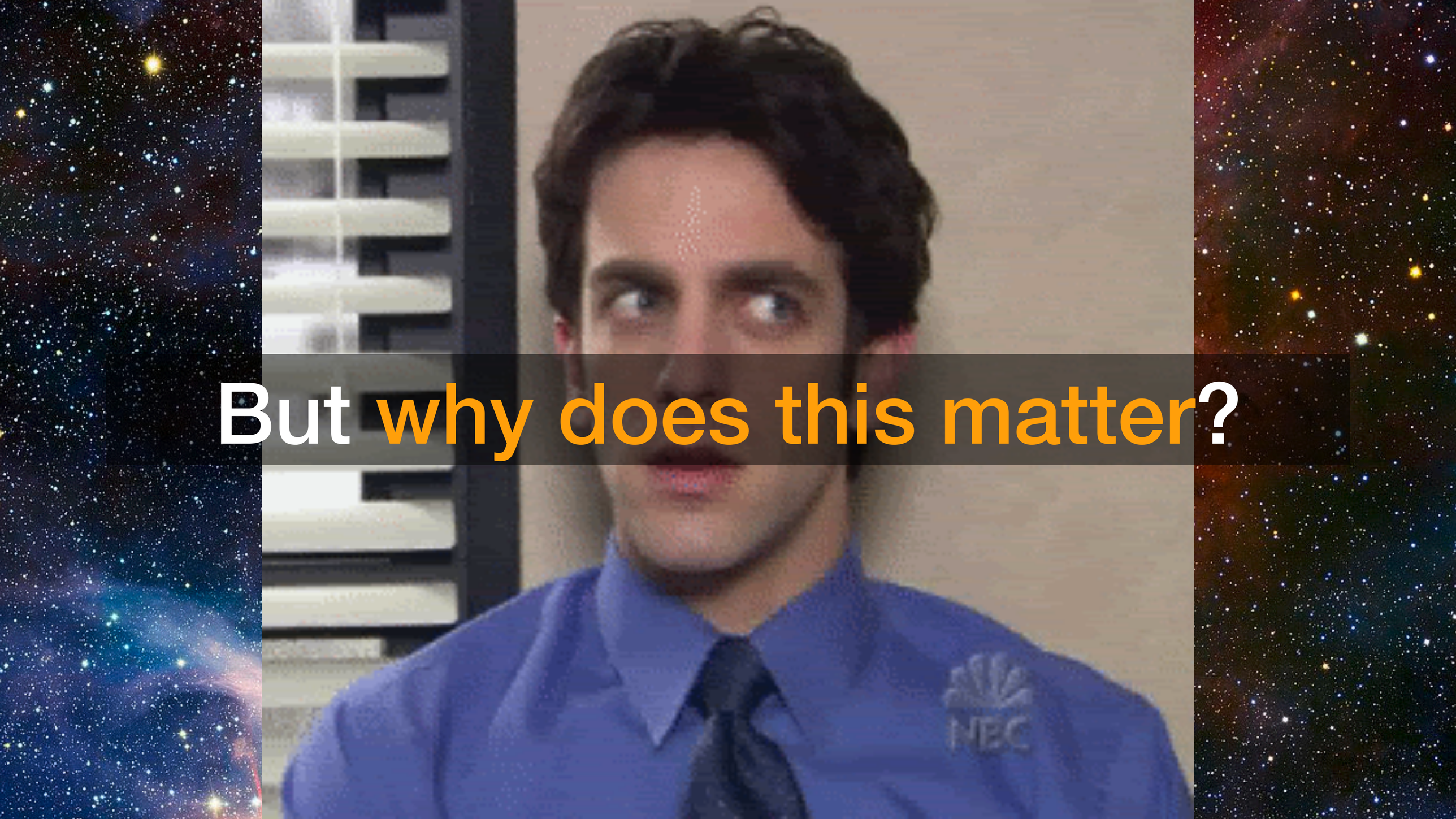
We're about to take a huge step forward, comparable in scope to the jump to AJAX, with WebAssembly

Stay with me. Let’s look deeper
into what WebAssembly is.

What do you mean by compile target?

- Other languages (or language transpilers) need a way to be compiled into the WebAssembly instruction set to run in the browser
- Examples of languages/transpilers that already have WebAssembly support: Rust, C/C++, Go, TypeScript, PHP, Lua…
- You write your code in the language of your choice, then compile it to WebAssembly for use in the browser!

We can run more than javascript in the browser.


But why does this matter?

Using the right tool for the job

- Augmenting JS at its not-so-strong points
- Re-using existing code in the browser
- Creating better, faster[, stronger] user experiences

Augmenting JS at it’s not-so-strong points

- Anything to do with types (yes yes I know TypeScript)
- Math-based applications (speed and precision)
- Code that could be more performant with a multi-threaded approach
- Anything dealing with graphics, image processing, etc

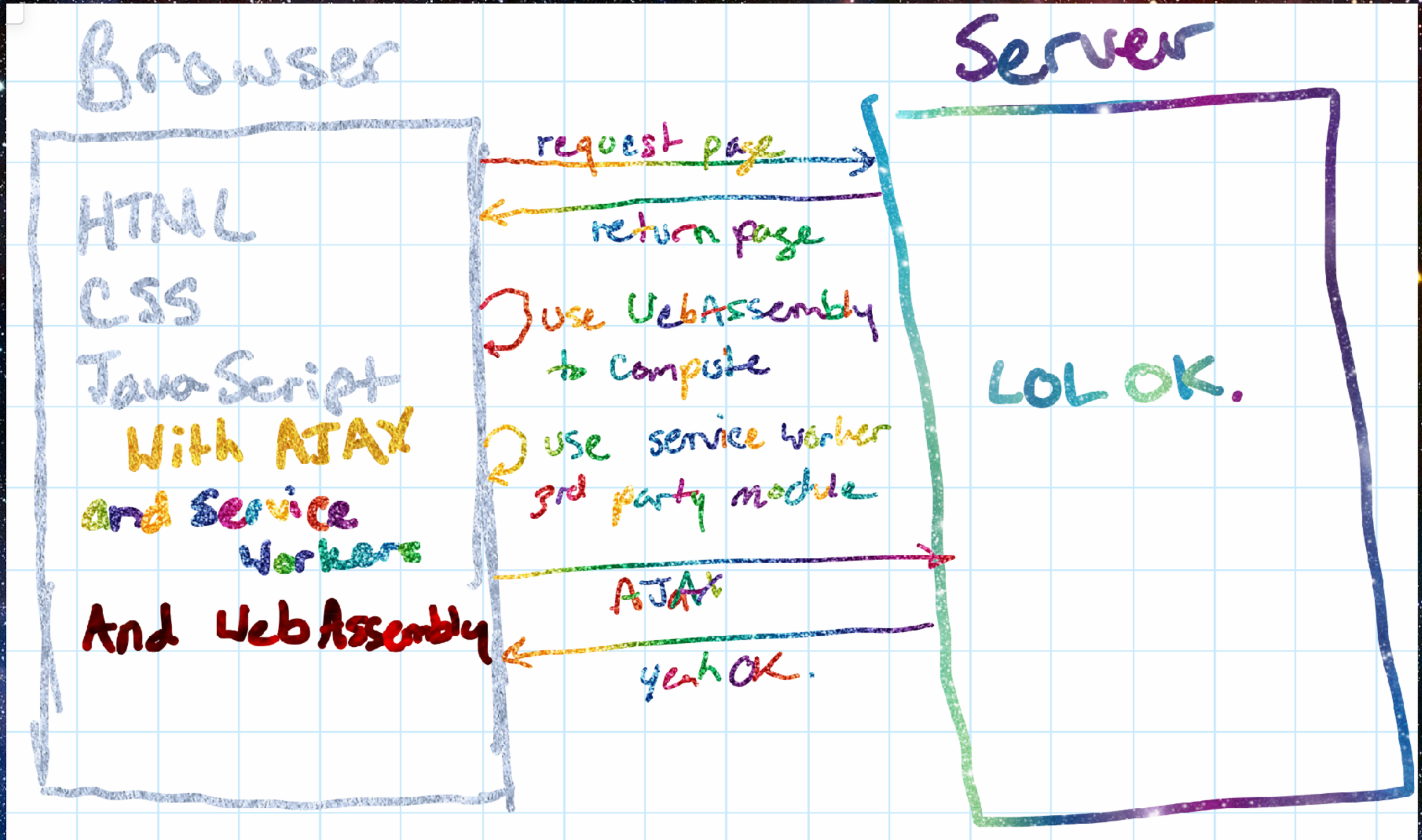
Re-using code bases from other languages

- No need to completely re-write existing, tested libraries that just happen to be in other languages
- One tricky bit: beware compiled file sizes!
  - Depends on the language/compiler
- You can also tell most wasm compilers what to keep and what to discard to help with this

Building better user experiences

- Allowing users to do more in the browser
- Can be used to craft better offline experiences
- Making faster, more capable web apps

So what does the web look like
with WebAssembly?


Browser
HTML
CSS
JavaScript
With AJAX
and Service
Workers
And WebAssembly
request page
return page
use WebAssembly
to compute
use service worker
3rd party module
AJAX
yeah OK.
Server
LOL OK.

What does a WebAssembly application look like?

The basic parts of a WebAssembly application

- Looks mostly like any other web app
- WebAssembly code stored in .wasm files
- Currently, you bring in WebAssembly modules using ES6 import syntax
  - <script> tag support in the works

Let’s dive into a bit bigger
WebAssembly Demo.

Editing images in the browser with ImageMagick

- Uses ImageMagick that's been compiled down to WebAssembly
- Allows you to take a capture with your webcam and modify it
- Not only allows fast image manipulation, but re-use of a very useful library with minimal modification and no re-writes

But can I use WebAssembly today?

Yep!

WebAssembly - OTHER

Usage
Global 88.41%

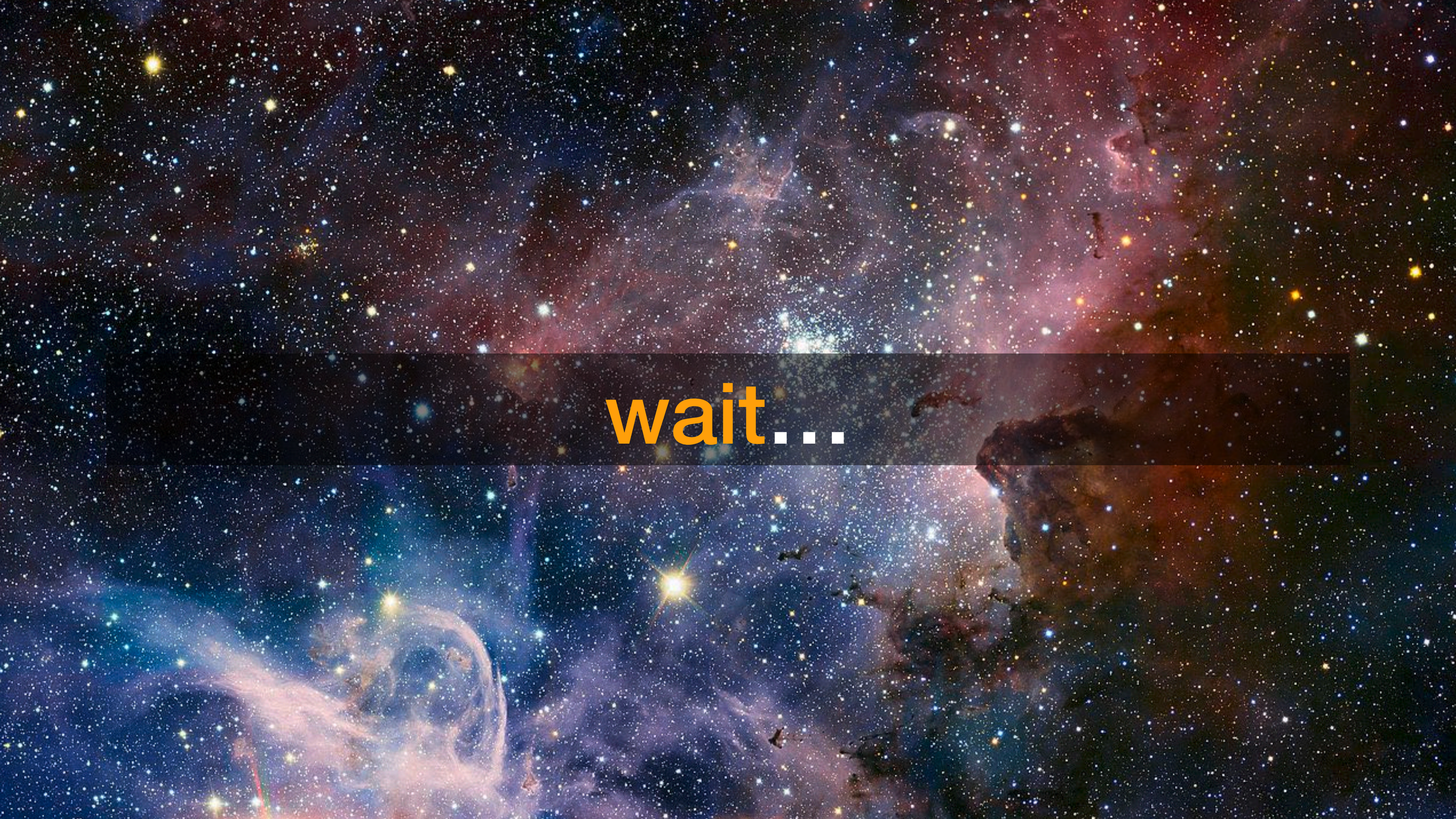
WebAssembly or "wasm" is a new portable, size- and load-time-efficient format suitable for compilation to the web.

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

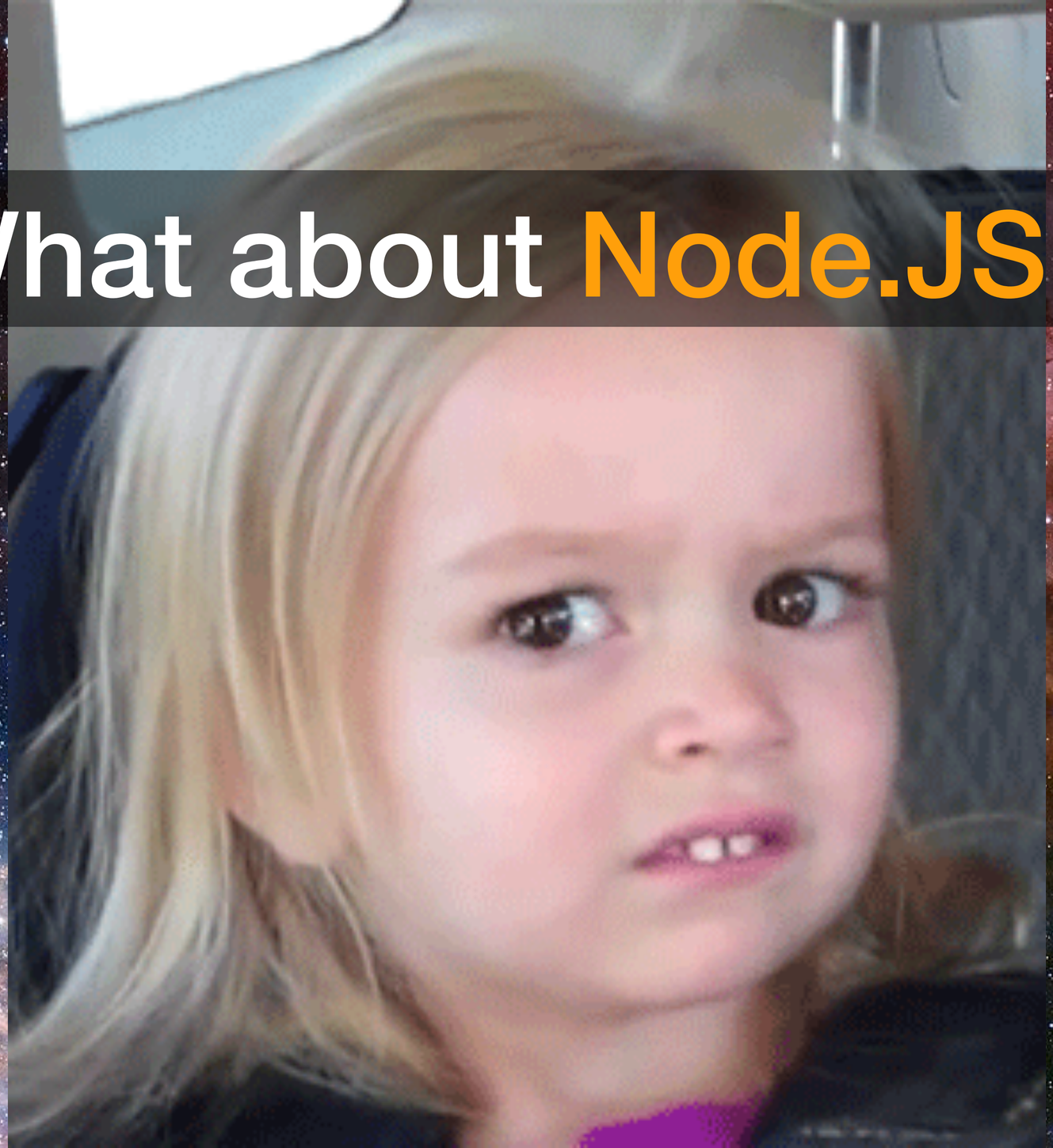
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baid Brow
		2-46													
	12-14	[1] 47-51	4-50		10-37										
	[3] 15	[4] 52	[2] 51-56	3.1-10.1	[2] 38-43	3.2-10.3							4-6.4		
6-10	16-17	53-69	57-77	11-12.1	44-63	11-12.4		2.1-4.4.4	12-12.1				7.2-9.2		
11	18	70	78	13	64	13.2	all	76	46	78	68	12.12	10.1	[2] 1.2	7.1
	76	71-72	79-81	TP											

This changes the entire
landscape of the web.

Let’s talk about Node.JS.


wait...

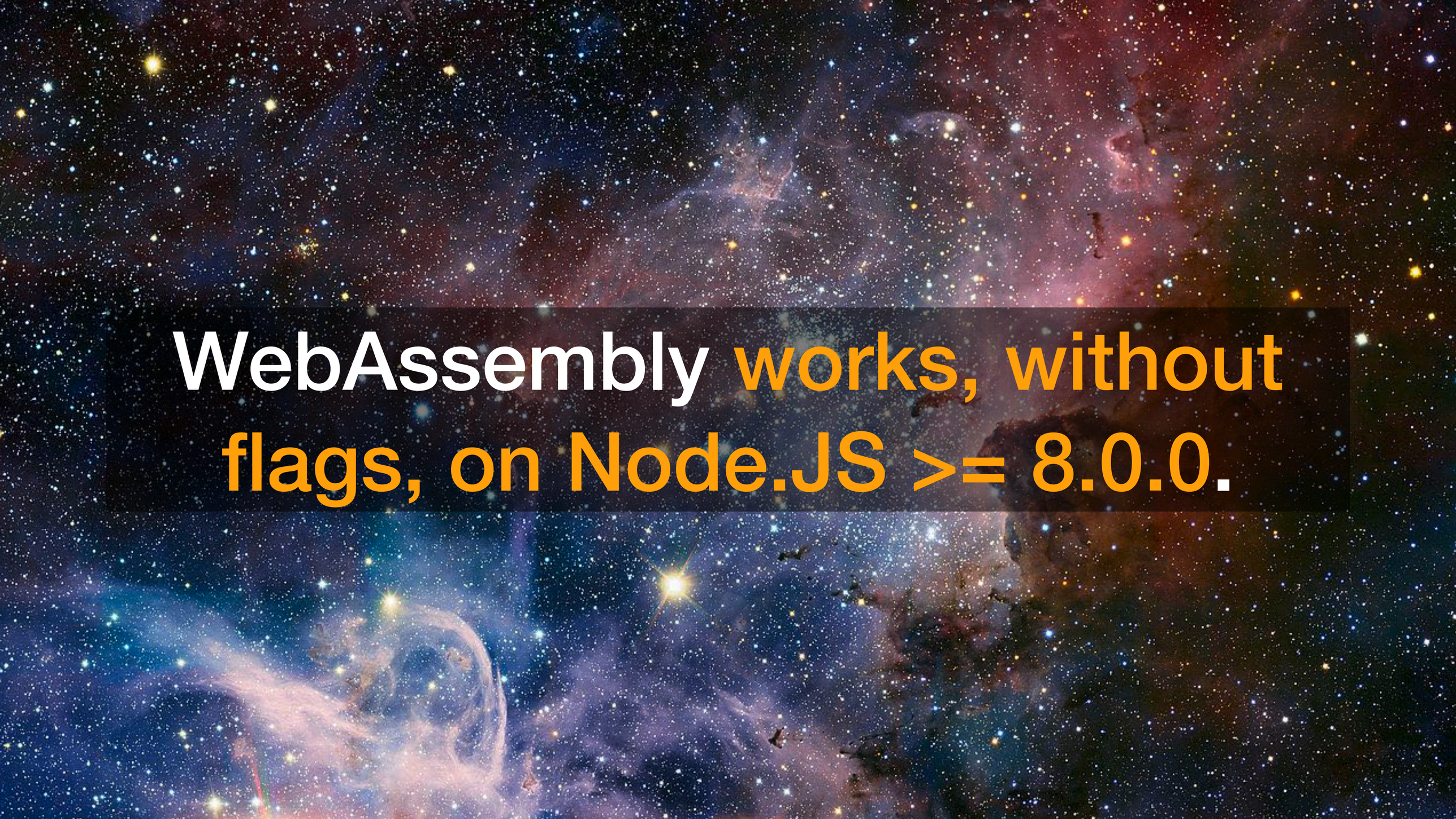
What about Node.JS!?



Native. Heccin. Modules.

The problem with native modules

- Disclaimer: I respect the hecc out of the node-gyp team.
- Native modules mean recompilation of native code on every machine
- This means if your architecture doesn't support the native code, or the build decides to fail, you're left with no working library


WebAssembly works, without
flags, on Node.JS >= 8.0.0.

V = 1
lim (1 + 1/n)^n = e
cos α =

WebAssembly vs. Native Modules

- Pre-compiled to the WASM instruction set: no re-compilation on each machine
- WASM instruction set works wherever Node.JS works
- We can use native code wherever Node.JS runs
- Drawback: Node.JS can't do everything through WebAssembly yet (like fs)

- Must be re-compiled on each system or rely on cached compiled code
- Underlying native code must work on any system you want to run it on (ARMv6, for instance?)
- Native builds can take a bit and don't always go as planned
- Upside: native modules give you way more control than WASM at the moment.

Example: magica

- ImageMagick cli & Node.JS tool compiled down to WASM
- Installs in seconds with no compilation step
- Gives you the power of ImageMagick without the native install step (no more `$ brew install imagemagick`

```
14:05:37 in code/wasm-demo/node-app is 📦 v1.0.0 via ⬢ v10.16.3
➜ _
```

```
14:43:20 in code/wasm-demo/node-app is 📦 v1.0.0 via ⬢ v10.16.3
➜ magica --command "identify javascript.jpg" --input javascript.jpg
javascript.jpg JPEG 3007x1691 3007x1691+0+0 8-bit sRGB 649688B 0.000u 0:00.000

14:43:57 in code/wasm-demo/node-app is 📦 v1.0.0 via ⬢ v10.16.3 took 4s
➜ magica --command "convert javascript.jpg -scale 50% js-smaller.png" --input j
avascript.jpg

14:45:22 in code/wasm-demo/node-app is 📦 v1.0.0 via ⬢ v10.16.3 took 5s
➜ magica --command "identify js-smaller.png" --input js-smaller.png
js-smaller.png PNG 1504x846 1504x846+0+0 8-bit sRGB 2012230B 0.000u 0:00.000
```

WASM, with some more work, can start replacing native modules.

WebAssembly and Serverless

- Cloudflare Workers uses V8 and supports WebAssembly (mainly Rust/C/C++ due to package size)

Search the docs...

- Quick Start
- Template Gallery
- Tutorials
- Workers Sites NEW
- Tooling
- About
- Reference

Boilerplates

Originless

Hello World

Simple Hello World in JS

Paste this into your terminal:

```
wrangler generate my-app
https://github.com/cloudflare/worker-
template
```

Click to Copy

Demo

Originless Wasm

Hello World Rust

Simple Hello World in Rust

Paste this into your terminal:

```
wrangler generate my-app
https://github.com/cloudflare/rustwasm
worker-template
```

Click to Copy

Tutorial Demo

Router

Selects the logic based on the `request` method and URL. Use with REST APIs or apps that require routing logic.

Originless

Hello World TypeScript

Simple Hello World in TypeScript

The future of the web with WebAssembly

- The short version: augmenting JS with code from other languages.
- The big picture: a multi-language web.
- The future: a bit of a question mark.

Thanks For Listening!

- kas@cloudflare.com
- @nodebotanist