WEBASSEMBLY

What is WebAssembly

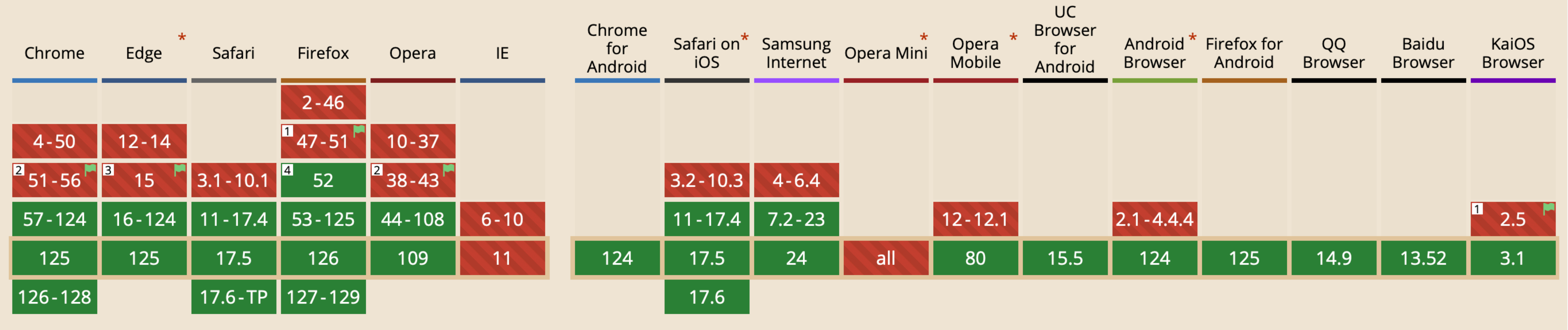
> WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

What is WebAssembly? (cont.)

- Stack-based virtual machine (similar to the JVM or the CIL of .NET)
  - Non object-oriented (only supports simple datatypes)
  - Linear memory - memory is just an `ArrayBuffer` (no GC)
- Compilation target of compiled programming languages
- Sandboxed execution environment
  - Host interaction via imports/exports
  - security checks applied
- W3C standard (1.0)

How to run?

- Built-in browser support: https://caniuse.com/?search=wasm



Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
			🟥 2-46													
🟥 4-50	🟥 12-14		🟥 [1] 47-51 ⚑	🟥 10-37												
🟥 [2] 51-56 ⚑	🟥 [3] 15 ⚑	🟥 3.1-10.1	🟩 [4] 52	🟥 [2] 38-43 ⚑			🟥 3.2-10.3	🟥 4-6.4								
🟩 57-124	🟩 16-124	🟩 11-17.4	🟩 53-125	🟩 44-108	🟥 6-10		🟩 11-17.4	🟩 7.2-23		🟥 12-12.1		🟥 2.1-4.4.4				🟥 [1] 2.5 ⚑
🟩 125	🟩 125	🟩 17.5	🟩 126	🟩 109	🟥 11	🟩 124	🟩 17.5	🟩 24	🟥 all	🟩 80	🟩 15.5	🟩 124	🟩 125	🟩 14.9	🟩 13.52	🟩 3.1
🟩 126-128		🟩 17.6-TP	🟩 127-129				🟩 17.6									

- Built-in *node.js* support

- Dedicated / stand-alone interpreters (see later)

Which languages can compile to WASM?

- C/C++ `emscripten`
- Rust *built-in*
- Go *built-in*, `TinyGo`
- C# `Blazor`, `Uno` / F# *community*
- Kotlin *built-in*
- AssemblyScript *native*
- Dart/Flutter *built-in*
- Swift *community*
- Zig *built-in*
- Nim (via generated C-Code)
- ...

Why WASM?

- Performance
- Re-use existing (C/C++) code bases
- Hide implementation details (better then JS obfuscation)

Use Case #1: Web Apps

- Re-use existing code bases in browser apps
- Use WASM to provide calculation intense parts of an app
  - signal processing (images, video, audio)
  - Complex calculations (i.e. graphics, 3d models)
  - Weather simulation
  - Games (of course 😃)
- In use today:
  - Jupyter Notebooks (running full Python interpreter on WASM incl. numpy, …)
  - Photoshop/Lightroom browser apps (Adobe)
  - Figma
  - TinkerCAD
  - …

Downsides (Web Apps)

- No built-in API for DOM manipulations
    - Needs JS interaction hooks (slow)
    - Will be refined in upcomining standards draft
- "Fat" binaries
    - No GC built-in -- languages need to ship a custom GC as part of the module
    - No standard lib - languages need to package everything they need

WebAssembly 2.0 (draf) will address all of these with

- reference types
- optional GC

Alternative strategy: 3-tier app: JS (UI) -> WASM middleware -> backend service

Use Case #2: Backend Apps/Functions

- Write portable backend apps/functions in WASM
- Additional specs: *WebAssembly System Interface* (WASI)
    - Currently two milestone versions of the spec: 0.1 and 0.2
- Different VM implementations: wasmtime, WAMR, WasmEdge, wazero, Wasmer, wasmi, wasm3
- Kubernetes devs work on providing k8s for WASM (instead of Containers): FAAS

Downsides (Backend Apps)

- WASI specs early drafts
- Lots of features missing (i.e. direct network access)
- Area of active development

How fast is WASM?

Very simple CPU-intensive benchmark:

```
function fib (x) {
    if (x < 2) {
        return 1
    }

    return fib(x-1) + fib(x-2)
}
```

```
int fib(const int x) {
    if (x < 2) {
        return 1;
    }

    return fib(x-1) + fib(x - 2);
}
```

How fast is WASM? (cont.)

Implementation	Time for `fib(45)` [s]	Size of the binary [kB]	Remarks
JavaScript	10.5	0.2	node v22
C (native)	3.4	33	macOS
C (WASM)	4.5	(6.3 + 12) = 18.3	node v22

How fast is WASM? (cont.)

Implementation	Time for `fib(45)` [s]	Size of the binary [kB]	Remarks
Go (native)	3.9	2,000	macOS
Go (WASM)	20.5	2,100	node v22 w/ custom JS
Go (WASM)	75.8	2,100	wasmtime
TinyGo (WASM)	3.9	603	wasmtime
TinyGo (WASM)	4.7	603	node v22 w/ custom JS

Stuff to check out

- https://webassembly.org
- https://developer.mozilla.org/en-US/docs/WebAssembly
- https://wasi.dev
- https://www.cncf.io/blog/2024/03/12/webassembly-on-kubernetes-from-containers-to-wasm-part-01/