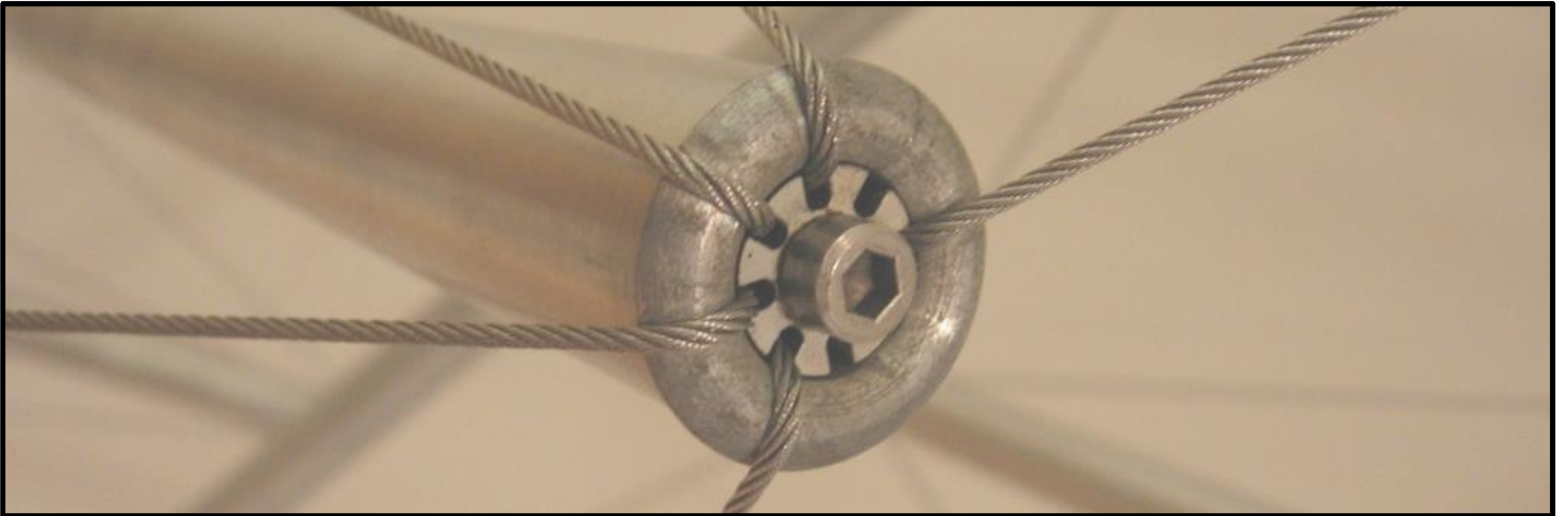


WebAssembly

Julian Arz - 24.07.2021



What is WebAssembly?

WebAssembly is a portable binary format.

It lets you run languages **other than JavaScript** in the browser.

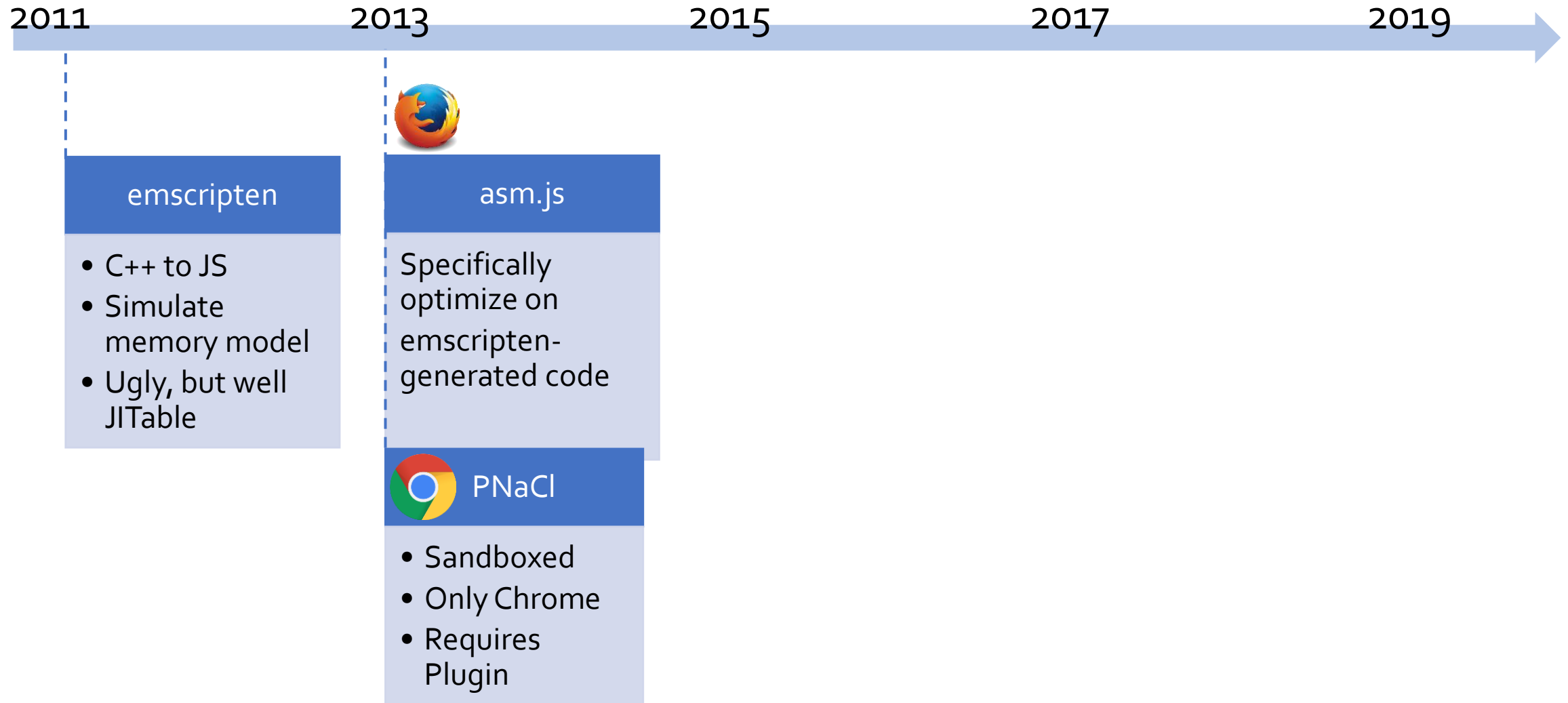
But also:

WebAssembly is not assembly, and it is also not (only) for the web.

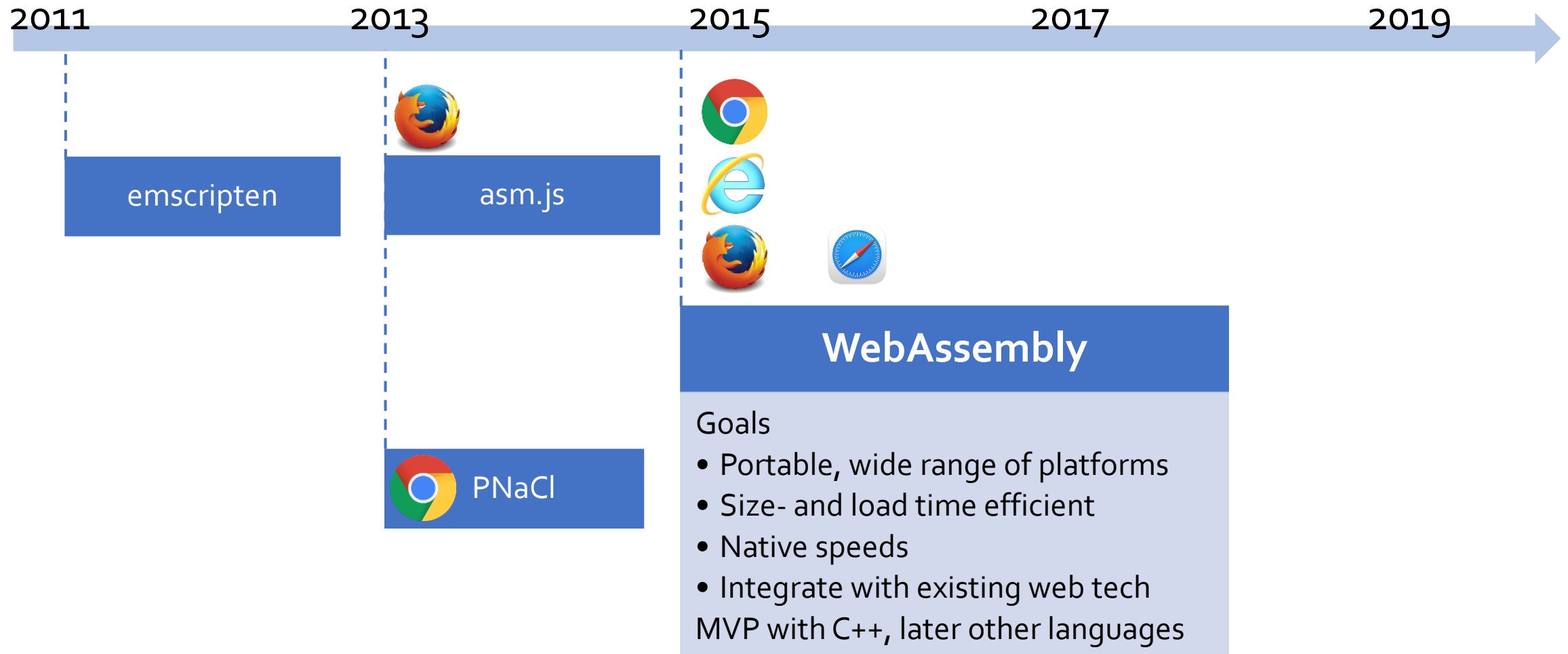
Topics

- History & Goals
- Demo
- Use cases
- Languages

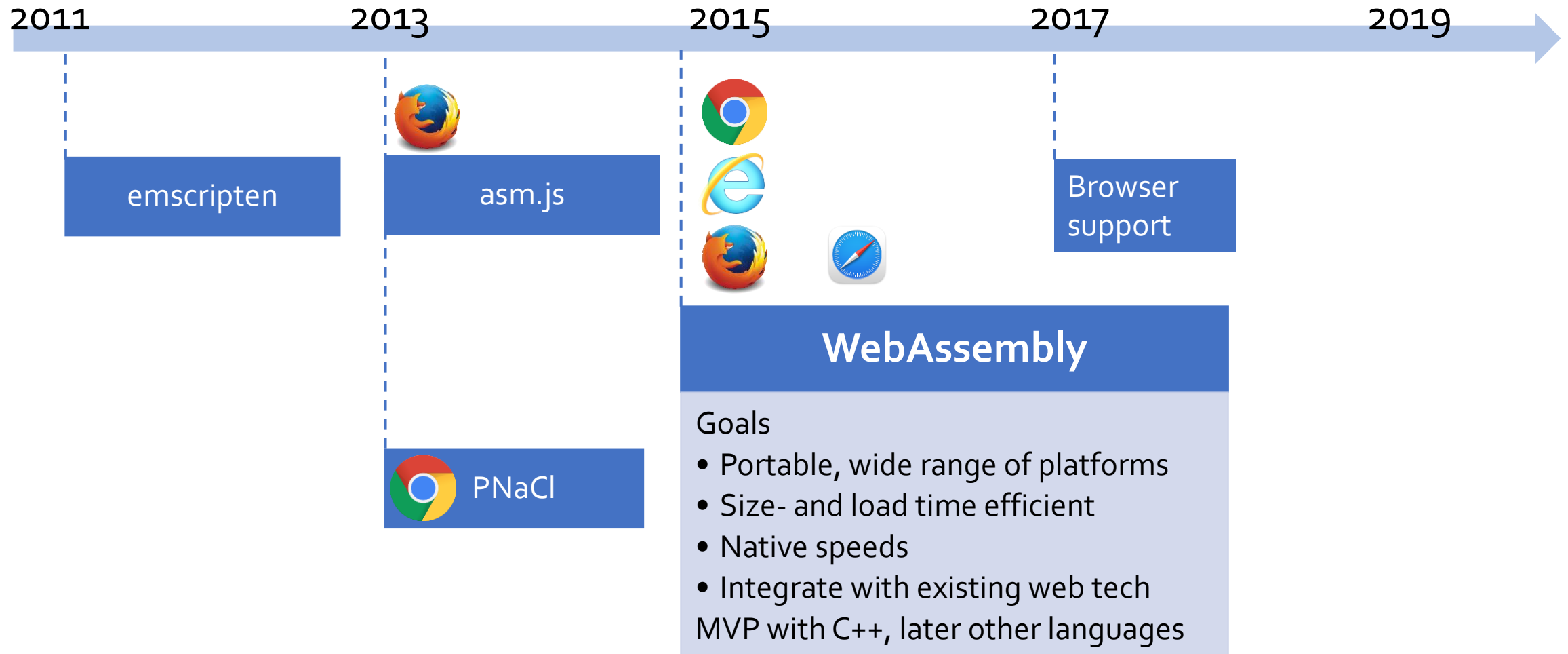
History: C++ in Browser



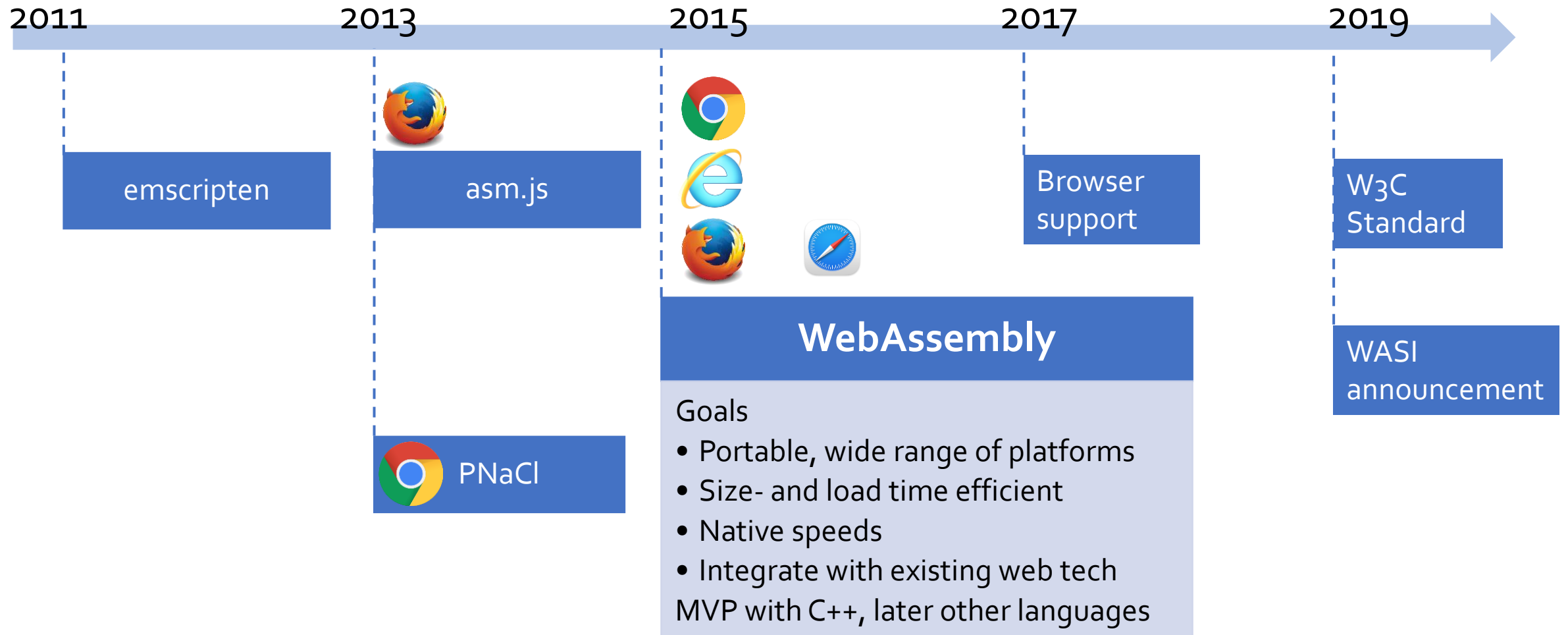
History: C++ in Browser



History: C++ in Browser



History: C++ in Browser



Demo

Hello World

Heron

Labyrinth

Demo: Hello World

```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello WebAssembly!");
5  }
```

```
$ emcc hello.cpp -o hello.html
```



```
✓ build
  <> hello.html
  JS hello.js
  📄 hello.wasm
```

Text format:

```
63  (import "wasi_snapshot_preview1" "fd_close" (func (;0;) (type 0)))
64  (import "wasi_snapshot_preview1" "fd_read" (func (;1;) (type 10)))
65  (import "wasi_snapshot_preview1" "fd_write" (func (;2;) (type 10)))
66  (import "env" "__cxa_atexit" (func (;3;) (type 3)))
67  (import "env" "abort" (func (;4;) (type 6)))
68  (import "wasi_snapshot_preview1" "environ_sizes_get" (func (;5;) (type 2)))
69  (import "wasi_snapshot_preview1" "environ_get" (func (;6;) (type 2)))
70  (import "env" "strftime_l" (func (;7;) (type 9)))
71  (import "env" "emscripten_resize_heap" (func (;8;) (type 0)))
72  (import "env" "emscripten_memcpy_big" (func (;9;) (type 3)))
73  (import "env" "setTempRet0" (func (;10;) (type 4)))
74  (import "wasi_snapshot_preview1" "fd_seek" (func (;11;) (type 9)))
```


WebAssembly has no library, imports everything from environment -> **portability**

Demo: Heron

```
1 extern "C" {  
2     double heron(const double number) {  
3         double x = 42.0;  
4         for (int i = 0; i < 1000; ++i) {  
5             x = (x + number / x) / 2.0;  
6         }  
7         return x;  
8     }  
9 }
```

optimize for size, dce

\$ emcc heron.cpp --no-entry -Os -s EXPORTED_FUNCTIONS=[_heron] -o heron.wasm

 heron.wasm

no main method

make it callable from JS

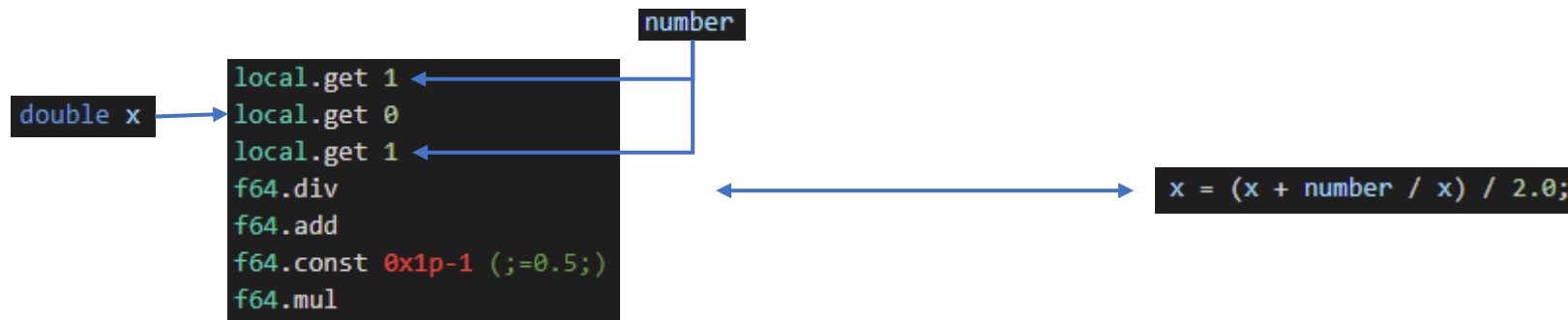
Call from JS:

Browser API

```
WebAssembly.instantiateStreaming(fetch("heron.wasm")).then((obj) => {  
    heron = obj.instance.exports.heron;  
});  
  
computeButton.onclick = function (event) {  
    const value = parseFloat(input.value);  
    const result = heron(value);  
    outputDiv.innerHTML = result;  
};
```

Demo: Heron

Code: Reverse Polish Notation (stack machine semantics)



It's not JavaScript, but is also not assembly!

Close to assembly, allows compiling while downloading

Demo: Labyrinth

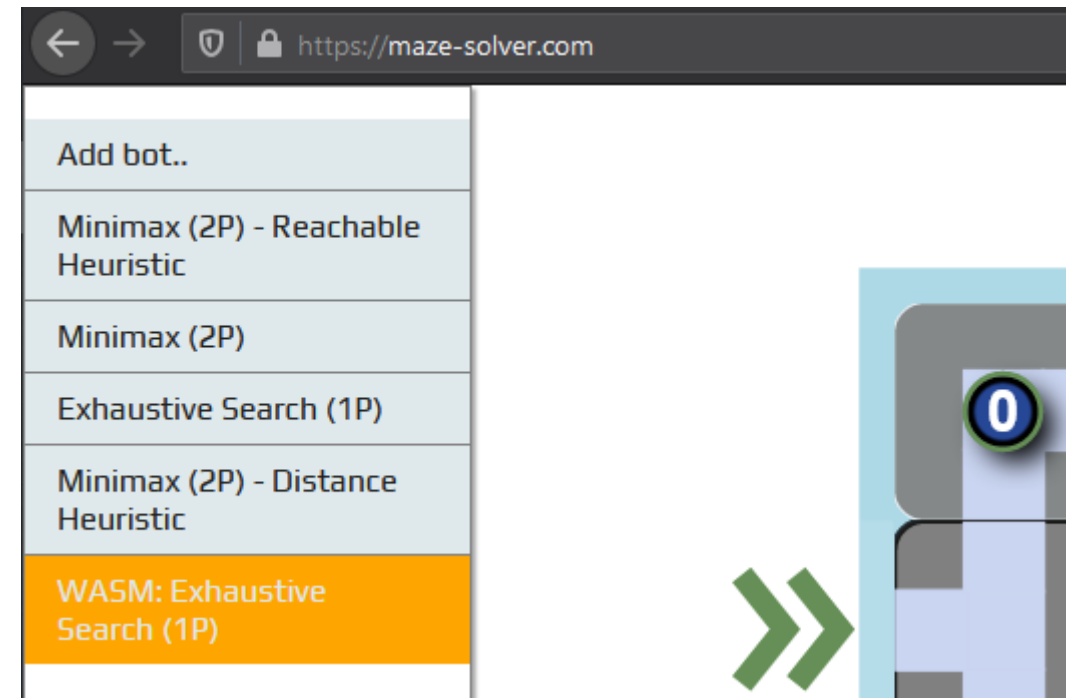
- Hello World & simple functions: easy
- Pass / return complex data types: tricky

My first experiences (around 2019):

- No clear guidelines, bad documentation, no debugging capabilities
- C++ - issue? Maybe Rust better?

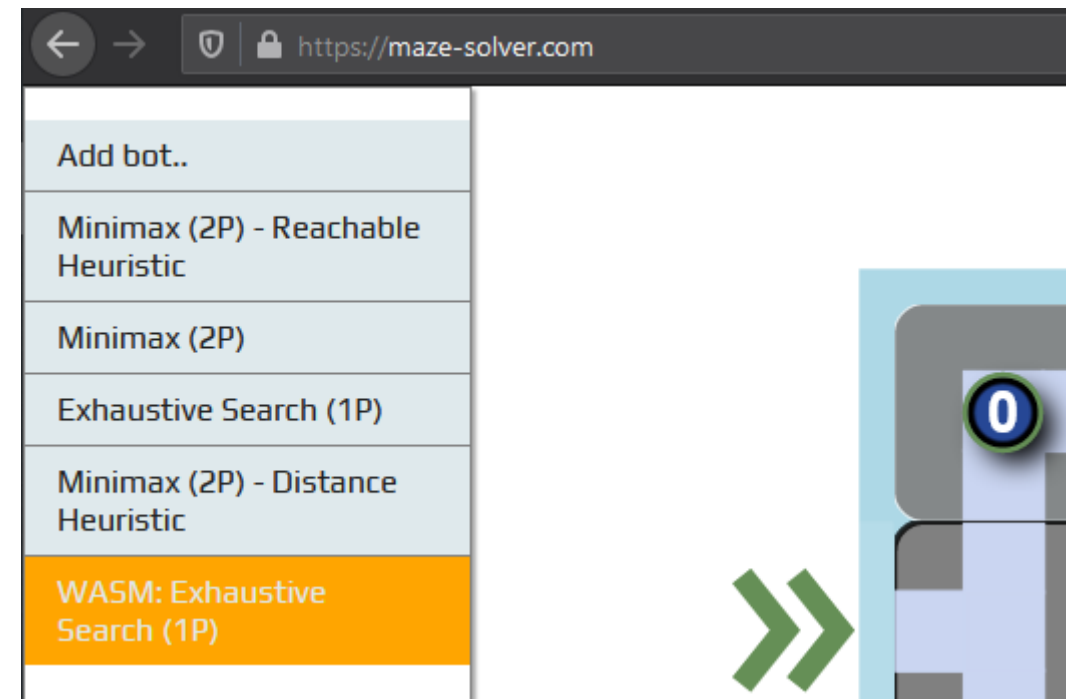
Today (2021):

- Better documentation, both Chrome and Firefox have debuggers



Demo: Labyrinth

- Opens new possibilities
- Choose the right tool for the job



How does WebAssembly relate to...

... JavaScript?

- Goal: **complement** of JS, not replacement
- Releases feature pressure from JS (e.g. SIMD, multi-threading, ...)
- Use cases: high performance, low start-up

gaming virtual / augmented reality

CAD AI / machine learning language interpreters

scientific computing VMs

video editing cryptography

How does WebAssembly relate to...

... the JVM?

- Java Applets: heavy-weight, slow startup
- WebAssembly: use existing functionalities of browsers
- WebAssembly is not limited to the Web
 - Runtimes [outside of browser](#): wasmtime, Wasmer, lucet
 - Standardization process -- WebAssembly System Interface (WASI)
- Java had a similar goal: „Write once, run anywhere“

How does WebAssembly relate to...

... Docker?

- Containers: isolated OS-Level virtualized sandbox
- WebAssembly (+ WASI): sandboxed, isolated execution of binary files
- Can WebAssembly replace Docker?

How does WebAssembly relate to...

... Docker?

- Container
- WebAssembly
- Can WebAssembly



Solomon Hykes

@solomonstre



If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

9:39 PM · Mar 27, 2019 · Twitter Web Client

770 Retweets **127** Quote Tweets **1,963** Likes

try files

How does WebAssembly relate to...

... Docker?

- Containers: isolated OS-Level virtualized sandbox
- WebAssembly (+ WASI): sandboxed, isolated execution of binary files
- Can WebAssembly replace Docker containers?

How does WebAssembly relate to...

... Docker?

- Containers: isolated OS-Level virtualized sandbox
- WebAssembly (+ WASI): sandboxed, isolated execution of binary files
- Can WebAssembly replace Docker containers?
 - Maybe some of them
 - e.g. serverless: faster startup
 - Krustlet

Languages

Stable

- C/C++, Rust (both via LLVM)
- Go
- C# (Blazor)
- A few more ...

Unstable, but usable

- Python (e.g. pyodide)
- Java?

Languages

Difficult to adopt: dynamic typing, garbage collection

Work in progress: add GC to WebAssembly

- Idea: let host system do GC

But then, how do Python and C# do it?

In general, two options:

- Compile language to WebAssembly, or
- Compile runtime/interpreter to WebAssembly

Languages

- Compile **language** to WebAssembly, or
- Compile **runtime/interpreter** to WebAssembly

Stable

- C/C++, Rust (both via LLVM)
- Go
- **C# (Blazor), .NET**
- A few more ...

Unstable, but usable

- **Python (e.g. pyodide)**
- Java
 - TeaVM, JWebAssembly
 - **CheerpJ**
- Some more
 - (Ruby, Swift, PHP, ...)

Conclusion

- WebAssembly is a W3C standard for a binary format, designed to be memory-safe, portable, and fast.
- Not only in browser
- Main use cases: **portability** and **performance**
- C#: have a look at Blazor
- Java: ?

Sources

Articles & Presentations

<https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/>

http://kripken.github.io/mloc_emscripten_talk/

<https://adlrocha.substack.com/p/adlrocha-can-wasm-become-the-new>

<https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>

Projects

<https://wasmtime.dev/>

<https://wasmer.io/>

<https://github.com/bytecodealliance/lucet>

<https://emscripten.org/>

<https://blazor.net/>

Talks

RustConf 2019: Clark, L. (2019). Closing Keynote <https://www.youtube.com/watch?v=IBZFJzGnBoU>

Podcasts

In **Software Engineering Daily**: Meyerson, J. (2021, March 23). Suborbital: WebAssembly Infrastructure with Connor Hicks (No. 1226)

In **Software Engineering Daily**: Meyerson, J. (2019, June 20). WebAssembly Compilation with Till Schneidereit (No. 855)

In **Software Architecture Radio**: Stine, M. (2019, March 27). WebAssembly with Brian Sletten (No. 7)

In **CppCast**: Turner, J & Irving, R. (2015, July 9). WebAssembly with JF Bastien (No. 15)

In **CppCast**: Turner, J & Irving, R. (2020, June 11). WebAssembly with Ben Smith (No. 251)

Sources

Misc

<https://webassembly.org/docs>

<https://twitter.com/solomonstre/status/1111004913222324225>

<https://github.com/appcypher/awesome-wasm-langs>

<https://madewithwebassembly.com/>

Security

<https://webassembly.org/docs/security/>

Lehmann, Daniel, Johannes Kinder, and Michael Pradel. "Everything old is new again: Binary security of webassembly." 29th {USENIX} Security Symposium ({USENIX} Security 20). 2020.

<https://www.virusbulletin.com/virusbulletin/2018/10/dark-side-webassembly/>

<https://spectrum.ieee.org/tech-talk/telecom/security/more-worries-over-the-security-of-web-assembly>

Benchmarks

<https://medium.com/@torch2424/webassembly-is-fast-a-real-world-benchmark-of-webassembly-vs-es6-d85a23f8e193>

Real world benchmark of GameBoy emulator

- Two ROMs, multiple devices, three browsers, comparing JS vs. WebAssembly
- Desktop: WebAssembly is 67% faster than JS on Chrome, on FF even 11 times faster. Results for mobile show greater speed-up.

<https://developers.google.com/web/updates/2019/02/hotpath-with-wasm>

Rotating a 16 MP image, JS vs WebAssembly.

- Browsers have different execution speeds in JS (different JS-engines). WebAssembly has same speed across all browsers -> predictable performance
- Measuring one-shot times, not allowing JS to be optimized from the start
- Even in fastest browser WebAssembly is 20% faster than JS

<https://surma.dev/things/js-to-asc/>

blurring images

- Rust and C - compiled WebAssembly slightly slower than fastest JS-Compiler (TurboFan)
- TurboFan 50times faster than Ignition.

<https://medium.com/vacatronics/webassembly-in-go-vs-javascript-a-benchmark-6deb28f24egd>

WebAssembly compiled from Go and C

- is_primrecursive fibonacci: WebAssembly 3times slower than JS
- iterative e: WebAssembly 4times faster than JS

<https://dev.to/linkuriousdev/to-wasm-or-not-to-wasm-3803>

- n-body-problem: WebAssembly compiled from C, Rust, and AssemblyScript.
- WebAssembly 20% to 50% slower than native, and 20% slower than JS.

https://www.youtube.com/watch?v=aC_QLLilwso

- Calculating prime numbers: C++ native 2 times faster than JS, WebAssembly slightly slower than native. JS warmup is discussed.