

# WebAssembly

## and the death of JavaScript?

...

@ColinEberhardt, Scott Logic

# A brief history of the web

# JavaScript

(created in 10 days in May 1995, by Brendan Eich)

# Java Applets

# ActiveX

Flash

# Silverlight

# Dart



# 2018 - JavaScript (still!)

# 2018 - JavaScript (still!)

... but the way we are using it has changed

# We are writing a *lot* of JavaScript

```
$ npx create-react-app my-app  
$ cd my-app  
$ find . -name '*.js' | xargs wc -l | tail -1  
79905 total
```

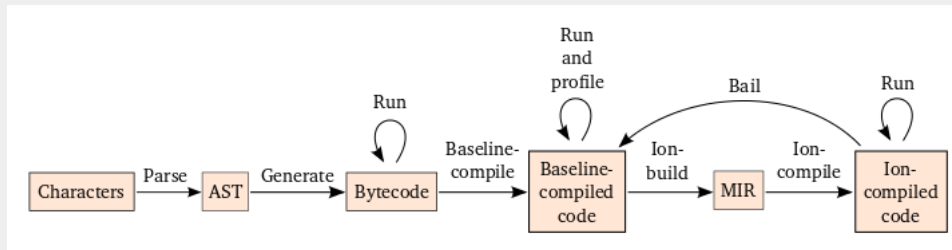
```

prototype&&(b[c]=d.value)},r="undefined"!=typeof window&&window===this?this:"undefined"!=typeof
n(b){if(b){for(var c=r,d=["Promise"],e=0;e<d.length-1;e++){var f=d[e];f in c||{c[f]={}};c=c[f]}c
a(c,d,{configurable:!0,writable:!0,value:b}})},ca=function(){ca=function(){};
(r.Symbol=da)),da=function(){var b=0;return function(c){return"jscomp_symbol_"+(c||"")+b++}}(),f
.iterator;b|)(b=r.Symbol.iterator=r.Symbol("iterator"));typeof Array.prototype[b]!==m&&aa(Array.p
0,value:function(){return ea(this)}});fa=function(){};ea=function(b){var c=0;return ha(function
ne:!0}})},ha=function(b){fa();b={next:b};b[r.Symbol.iterator]=function(){return this};
ia=function(b){fa();var c=b[Symbol.iterator];return c?c.call(b):ea(b)};ba(function(b){function c
of f?b:new f(function(c){c(b)})}if(b)return b;c.prototype.c=function(b){null==this.b&&(this.b=[]
e.g=function(){var b=this;this.f(function(){b.i()});var e=r.setTimeout;c.prototype.f=function(b
.b&&this.b.length){var b=this.b;this.b=[];for(var c=0;c<b.length;++c){var d=
e b[c];try{d()}catch(l){this.h(l)}}this.b=null};c.prototype.h=function(b){this.f(function(){thr
d 0;this.b=[];var c=this.f();try{b(c.resolve,c.reject)}catch(n){c.reject(n)}};f.prototype.f=func
!0,b.call(c,e))}var c=this,d=!1;return{resolve:b(this.B),reject:b(this.g)}};f.prototype.B=funct
"A Promise cannot resolve to itself"));else if(b instanceof f)this.C(b);
tch(typeof b){case "object":var c=null!=b;break a;case m:c=!0;break a;default:c=!1}c?this.A(b):t
d 0;try{c=b.then}catch(n){this.g(n);return}typeof c==m?this.D(c,b):this.i(b)};f.prototype.g=func
e.i=function(b){this.j(1,b)};f.prototype.j=function(b,c){if(0!=this.c)throw Error("Cannot settle
state"+this.c);this.c=b;this.h=c;this.l()};f.prototype.l=function(){if(null!=this.b){for(var b=
;c<b.length;++c)b[c].call(),b[c]=null;this.b=null};var g=new c;f.prototype.C=function(b){var c=
e.D=function(b,c){var d=this.f();try{b.call(c,d.resolve,d.reject)}catch(l){d.reject(l)}};f.proto
typeof b==m?function(c){try{e(b(c))}catch(Za){g(Za)}}:c;var e,g,h=new f(function(b,c){e=b;g=c});
e["catch"]=function(b){return this.then(void 0,b)};f.prototype.o=function(b,
n d(){switch(e.c){case 1:b(e.h);break;case 2:c(e.h);break;default:throw Error("a`"+e.c)};var e=
}{g.c(d)}};f.resolve=d;f.reject=function(b){return new f(function(c,d){d(b)}};f.race=function(

```

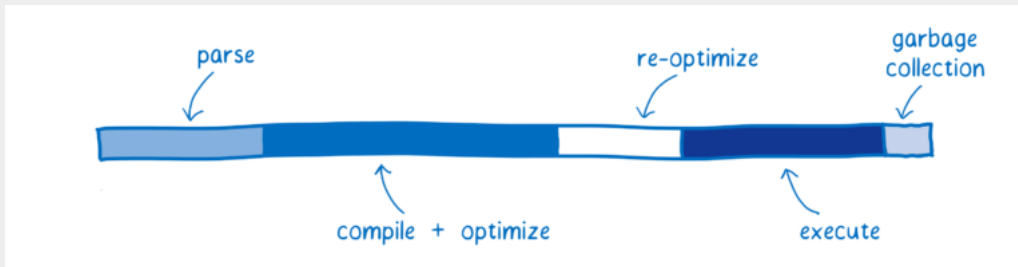
JavaScript is the Assembly Language of the Web

JavaScript isn't a very good Assembly Language!



<https://blog.mozilla.org/luke/2014/01/14/asm-js-aot-compilation-and-startup-performance/>

## What does JavaScript execution look like today?



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

“ *the Web has become the most ubiquitous application platform ever, and yet by historical accident the only natively supported programming language for that platform is JavaScript!*



# WebAssembly

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

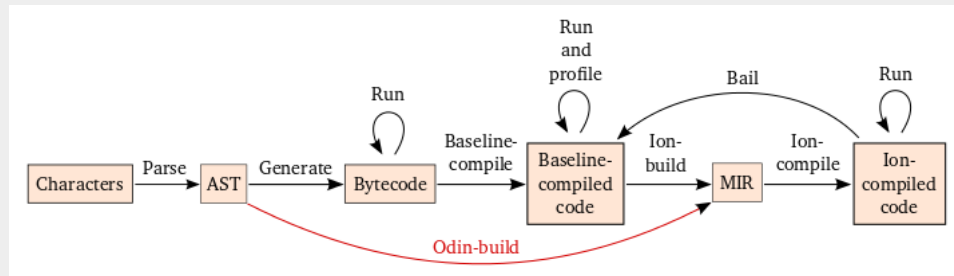
asm.js

```
float power(float number, int pow) {
    float res = number;
    for (int i = 0; i < pow - 1; i++) {
        res = res * number;
    }
    return res;
}
```

```
emcc power.c -O3 -s ONLY_MY_CODE=1 -s EXPORTED_FUNCTIONS=["'_power'"]
```

```
"use asm";
function X(a, b) {
    a = +a;
    b = b | 0;
    var c = 0.0, d = 0;
    d = b + -1 | 0;
    if ((b | 0) > 1) {
        b = 0;
        c = a;
    } else return +a;
    do {
        c = c * a;
        b = b + 1 | 0;
    } while ((b | 0) != (d | 0));
    return +c;
}
```

## asm.js optimised execution





# WebAssembly Roadmap

- 2015 - WebAssembly Community Group formed
- 2017 - WebAssembly MVP released
- 2018 - W3C public draft published



## WebAssembly - OTHER

Usage % of all users   
Global 72.75%

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 [Show all](#)

| IE | Edge <sup>*</sup> | Firefox | Chrome | Safari | iOS Safari <sup>*</sup> | Opera Mini <sup>*</sup> | Chrome for Android | UC Browser for Android | Samsung Internet |
|----|-------------------|---------|--------|--------|-------------------------|-------------------------|--------------------|------------------------|------------------|
|    |                   |         | 49     |        | 10.2                    |                         |                    |                        |                  |
|    |                   |         | 63     |        | 10.3                    |                         |                    |                        | 4                |
| 11 | 16                | 58      | 64     | 11     | 11.2                    | all                     | 64                 | 11.8                   | 6.2              |
|    | 17                | 59      | 65     | 11.1   | 11.3                    |                         |                    |                        |                  |
|    |                   | 60      | 66     | TP     |                         |                         |                    |                        |                  |
|    |                   | 61      | 67     |        |                         |                         |                    |                        |                  |

# WebAssembly In Practice



# WebAssembly In Practice

```
float power(float number, int pow) {  
    float res = number;  
    for (int i = 0; i < pow - 1; i++) {  
        res = res * number;  
    }  
    return res;  
}
```

```
$ xxd add.wasm
00000000: 0061 736d 0100 0000 0107 0160 027d 7f01  .asm.....`.}..
00000010: 7d03 0201 0004 0401 7000 0005 0301 0001  }.....p.....
00000020: 0712 0206 6d65 6d6f 7279 0200 0570 6f77  ....memory...pow
00000030: 6572 0000 0a33 0131 0101 7d02 4020 0141  er...3.1..}.@ .A
00000040: 0248 0d00 2001 417f 6a21 0120 0021 0203  .H.. .A.j!. .!..
00000050: 4020 0220 0094 2102 2001 417f 6a22 010d  @ . ..!. .A.j"..
00000060: 000b 2002 0f0b 2000 0b                .. ... ..
```

```

(module
  (table 0 anyfunc)
  (memory $0 1)
  (export "memory" (memory $0))
  (export "power" (func $power))
  (func $power (param $0 f32) (param $1 i32) (result f32)
    (local $2 f32)
    (block $label$0
      (br_if $label$0
        (i32.lt_s
          (get_local $1)
          (i32.const 2)
        )
      )
      (set_local $1
        (i32.add
          (get_local $1)
          (i32.const -1)
        )
      )
      (set_local $2
        (get_local $0)
      )
      (loop $label$1
        (set_local $2
          (f32.mul
            (get_local $2)
            (get_local $0)
          )
        )
      )
      (br_if $label$1
        (tee_local $1
          (i32.add
            (get_local $1)
            (i32.const -1)
          )
        )
      )
    )
  )
  (return
    (get_local $2)
  )
  (get_local $0)
)
)

```

```
// read the binary into a buffer
const fs = require("fs");
const buf = fs.readFileSync("./add.wasm");

// create a wasm module
const wasmModule = new WebAssembly.Module(new Uint8Array(buf));

// construct an instance of the module
const wasmInstance = new WebAssembly.Instance(wasmModule);

// invoke the exported function
const result = wasmInstance.exports.power(2, 3)
console.log(result);
```

```
char *message = "hello wasm!";
```

```
char *getMessageRef()  
{  
    return message;  
}
```

```
const { TextDecoder } = require("util");

// read the binary into a buffer
const fs = require("fs");
const buf = fs.readFileSync("./string.wasm");

// create a module and an instance
const wasmModule = new WebAssembly.Module(new Uint8Array(buf));
const wasmInstance = new WebAssembly.Instance(wasmModule);

// obtain a reference to linear and read the string
const linearMemory = wasmInstance.exports.memory;
const offset = wasmInstance.exports.getMessageRef();
const buffer = new Uint8Array(linearMemory.buffer, offset, 12);

// decode and log
const str = new TextDecoder("utf-8").decode(buffer);
console.log(str);
```

# WebAssembly Architecture

- A stack machine, 4 types, 67 instructions
- Designed to support streaming compilation
- Simple validation rules
- Exports / imports functions
- Linear memory is shared with JavaScript



# WebAssembly Future

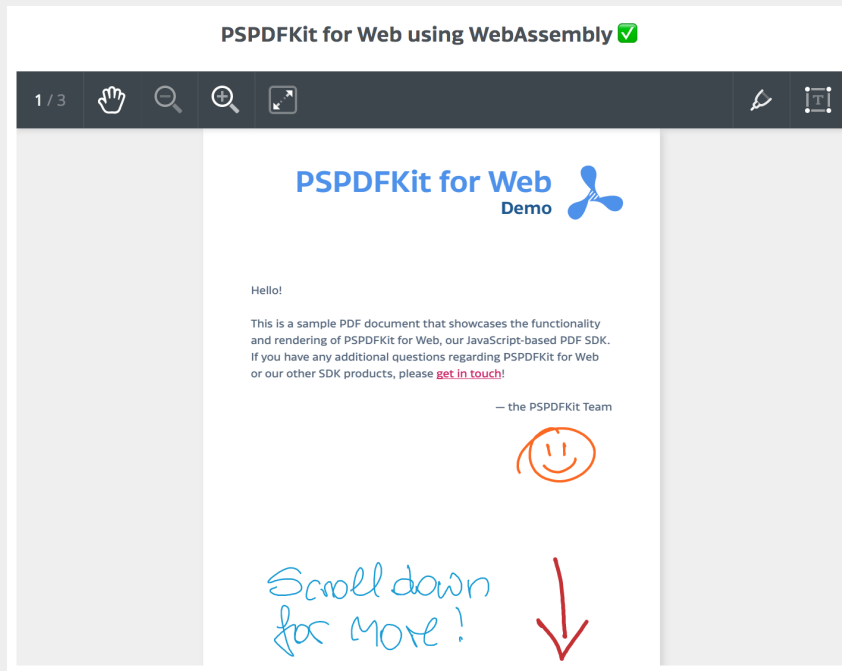
- Garbage collector
- Threads
- Host bindings
- SIMD
- Exception handling

# WebAssembly Language Support

(and what people are doing with it)

## C/C++

- Emscripten
- Based on LLVM
- Originally used to create asm.js



<https://pspdfkit.com/blog/2017/webassembly-a-new-hope/>

```
JSC.js Shell
Downloading contents, please wait...
Preparing...
All downloads complete.
Running...
JSC >>>
JSC >>>
JSC >>> Date();
Fri Sep 15 2017 20:23:37 GMT-0700
JSC >>> Date.now();
1505532221655
JSC >>>
JSC >>> 0.1 + 0.2
0.30000000000000004
JSC >>>
```

<https://mbbill.github.io/JSC.js/>

## Option Chain

An option chain demonstrating connectivity, real-time subscriptions to data and traversal of relationships. [Open the control panel](#) for stream settings and statistics.

| Call        |        |         |         |         |         | Put    |         |         |        |         |        |        |
|-------------|--------|---------|---------|---------|---------|--------|---------|---------|--------|---------|--------|--------|
| OI          | Volume | Change  | Last    | Bid     | Ask     | Strike | Bid     | Ask     | Last   | Change  | Volume | OI     |
| Dec 1, 2017 |        |         |         |         |         |        |         |         |        |         |        |        |
| 59          |        |         |         | 23.80 ↓ | 24.05 ↑ | 150.00 |         | 0.02 ↓  | 0.02   | 0.01    | 200    | 798    |
| 17          |        |         |         | 21.20 ↓ | 21.60 ↓ | 152.50 |         | 0.02 ↓  |        |         |        | 964    |
| 87          | 4      | -0.10 ↓ | 18.90 ↓ | 18.80 ↑ | 19.10 ↑ | 155.00 |         | 0.03 ↓  |        |         |        | 2,098  |
| 332         |        |         |         | 16.35 ↑ | 16.55 ↑ | 157.50 |         | 0.01 ↓  |        | 0.00    | 2      | 1,605  |
| 857         | 1      | -0.22   | 14.15   | 13.85 ↑ | 14.10 ↑ | 160.00 | 0.01    | 0.02 ↓  | 0.02 ↑ | 0.00 ↑  | 84     | 3,933  |
| 665         | 2      | 0.25    | 11.60   | 11.40 ↑ | 11.55 ↑ | 162.50 | 0.03 ↑  | 0.04 ↑  | 0.03 ↑ | -0.01 ↑ | 201    | 4,356  |
| 2,847       | 3,524  | -0.30 ↑ | 8.95 ↑  | 8.90 ↑  | 9.10 ↑  | 165.00 | 0.05 ↑  | 0.06 ↑  | 0.06 ↑ | 0.00 ↑  | 296    | 9,143  |
| 898         | 21     | -0.33 ↓ | 6.40 ↓  | 6.45 ↑  | 6.60 ↓  | 167.50 | 0.09 ↓  | 0.11 ↑  | 0.10 ↑ | 0.00 ↑  | 648    | 5,920  |
| 5,897       | 196    | -0.25 ↑ | 4.15 ↑  | 4.10 ↑  | 4.20 ↑  | 170.00 | 0.20 ↓  | 0.21 ↓  | 0.21 ↓ | -0.01 ↓ | 1,952  | 7,856  |
| 12,536      | 3,258  | -0.17 ↑ | 2.05 ↑  | 2.01 ↑  | 2.04 ↑  | 172.50 | 0.58 ↑  | 0.59 ↑  | 0.58 ↓ | 0.02 ↓  | 3,531  | 9,265  |
| 25,636      | 8,835  | -0.10 ↑ | 0.65 ↑  | 0.63 ↓  | 0.65 ↑  | 175.00 | 1.70 ↑  | 1.71 ↓  | 1.70 ↑ | 0.08 ↑  | 4,116  | 10,876 |
| 19,956      | 1,859  | -0.04 ↓ | 0.15 ↓  | 0.15 ↑  | 0.16 ↑  | 177.50 | 3.65 ↓  | 3.80 ↓  | 3.80 ↓ | 0.25 ↓  | 151    | 8,119  |
| 11,849      | 863    | -0.02 ↓ | 0.04 ↓  | 0.04 ↓  | 0.05 ↑  | 180.00 | 6.05 ↓  | 6.20 ↓  | 5.95 ↓ | -0.05 ↓ | 252    | 338    |
| 4,083       | 600    | -0.01 ↓ | 0.02 ↓  | 0.02 ↑  | 0.03 ↑  | 182.50 | 8.55 ↑  | 8.65 ↑  | 8.35   | -0.45   | 5      | 63     |
| 14,206      | 10     | 0.00    | 0.02    | 0.01    | 0.02 ↓  | 185.00 | 11.00 ↑ | 11.25 ↑ | 11.30  | -3.45   | 8      | 47     |
| 3,390       | 61     | -0.01   | 0.01    |         | 0.01 ↓  | 187.50 | 13.35 ↓ | 13.75 ↓ |        |         |        | 0      |
| 2,928       |        |         |         |         | 0.02 ↓  | 190.00 | 15.85 ↑ | 16.35 ↓ |        |         |        | 2      |
| 1,066       |        |         |         |         | 0.02 ↓  | 192.50 | 18.35 ↓ | 18.80 ↑ |        |         |        | 14     |
| 240         |        |         |         |         | 0.01 ↑  | 195.00 | 20.85 ↑ | 21.25 ↑ |        |         |        | 26     |

## Java / C#

- More challenging, these languages require a GC

## Java / C#

- More challenging, these languages require a GC
- Blazor, experimental project, using Mono
  - Testing interpreted mode, vs. AOT (with runtime for GC etc ...)
  - Blazor is an SPA framework



# JavaScript

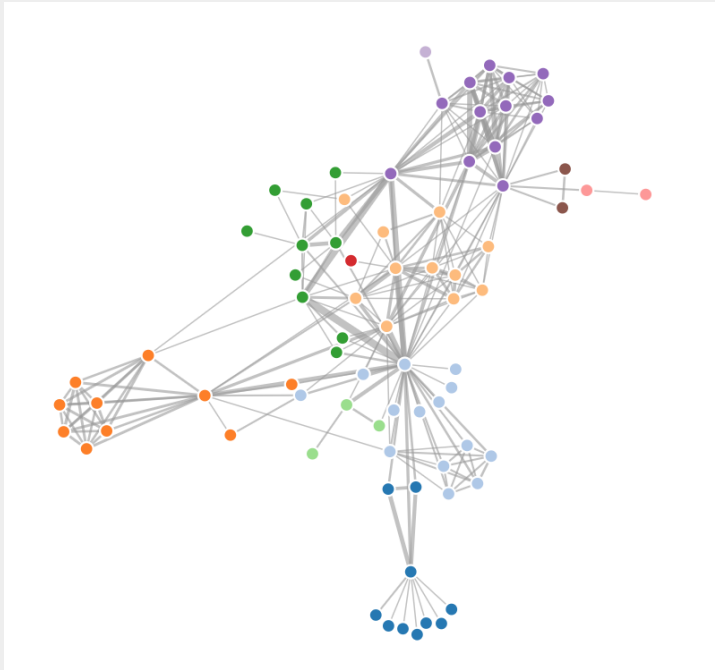
- Needs a GC and isn't statically typed

# JavaScript

- Needs a GC and isn't statically typed
- “Walt is a JavaScript-like syntax for WebAssembly text format”

# JavaScript

- Needs a GC and isn't statically typed
- “Walt is a JavaScript-like syntax for WebAssembly text format”
- AssemblyScript - a TypeScript to WebAssembly compiler
  - Awaiting GC before it can really become powerful



<https://bl.ocks.org/ColinEberhardt/6ceb7ca74aabc9c8534d7120d31b382>

```
simulation = d3.forceSimulation()  
  .force("link", d3.forceLink().id(function(d) { return d.id; })))  
  .force("charge", d3.forceManyBody())  
  .force("center", d3.forceCenter(width / 2, height / 2));  
  
function ticked() {  
  simulation.tick();  
  link  
    .attr('x1', d => d.source.x)  
    .attr('y1', d => d.source.y)  
    .attr('x2', d => d.target.x)  
    .attr('y2', d => d.target.y);  
  
  node  
    .attr('cx', d => d.x)  
    .attr('cy', d => d.y);  
}  
  
setInterval(ticked, 25);
```

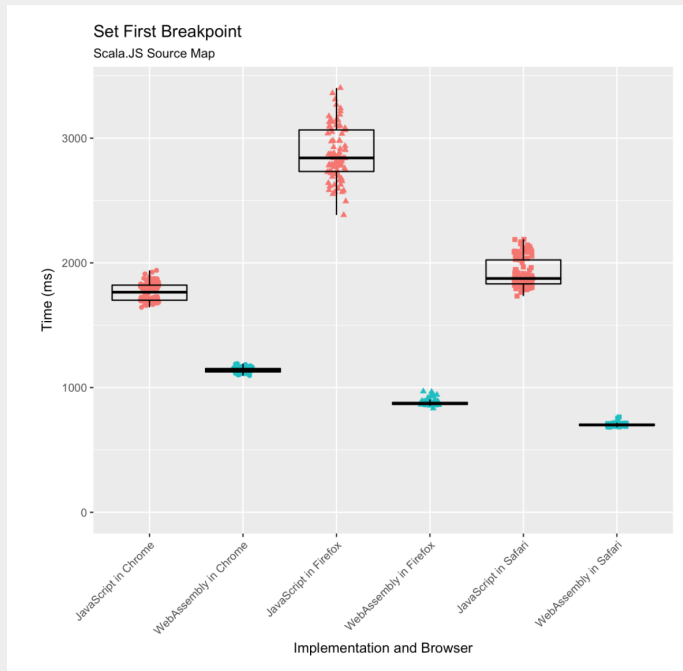
# Rust

- Doesn't require a GC
- Originally used Emscripten, but moved to a simpler toolchain

“ *We're poised to be THE language of choice for wasm.*



<http://blog.scottlogic.com/2017/12/13/chip8-emulator-webassembly-rust.html>



<https://hacks.mozilla.org/2018/01/oxidizing-source-maps-with-rust-and-webassembly/>



# Crystal Ball Gazing



# 2018

- Rust, C, C++ used in production for performance critical, algorithmic tasks

# 2018

- Rust, C, C++ used in production for performance critical, algorithmic tasks
- Webpack

# 2018

- Rust, C, C++ used in production for performance critical, algorithmic tasks
- Webpack
- Java, C#, Typescript lots of creative experiments / POCs

# 2018

- Rust, C, C++ used in production for performance critical, algorithmic tasks
- Webpack
- Java, C#, Typescript lots of creative experiments / POCs
- Native node modules

# 2018

- Rust, C, C++ used in production for performance critical, algorithmic tasks
- Webpack
- Java, C#, Typescript lots of creative experiments / POCs
- Native node modules
- GC support

# 2019

- Host bindings, SIMD, threading, ...

# 2019

- Host bindings, SIMD, threading, ...
- Java, C# become production ready



# 2019

- Host bindings, SIMD, threading, ...
- Java, C# become production ready
- Another wave of mobile, desktop and server-side UI frameworks will re-target the web
  - write once, run everywhere

# 2019

- Host bindings, SIMD, threading, ...
- Java, C# become production ready
- Another wave of mobile, desktop and server-side UI frameworks will re-target the web
  - write once, run everywhere
- Performance gains fail to materialise, with backlash from early adopters

## 2019

- Host bindings, SIMD, threading, ...
- Java, C# become production ready
- Another wave of mobile, desktop and server-side UI frameworks will re-target the web
  - write once, run everywhere
- Performance gains fail to materialise, with backlash from early adopters
- Heavyweight productivity tools start moving to the web (e.g. Photoshop, AutoCAD)

## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use  
wasm"

## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use **wasm**"
- Native Android apps die-out in favour of Progressive Web Apps (PWA) running on WebAssembly

## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use **wasm**"
- Native Android apps die-out in favour of Progressive Web Apps (PWA) running on WebAssembly
- Windows store moves to PWA / WASM

## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use **wasm**"
- Native Android apps die-out in favour of Progressive Web Apps (PWA) running on WebAssembly
- Windows store moves to PWA / WASM
- A new DOM alternative will emerge?

## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use **wasm**"
- Native Android apps die-out in favour of Progressive Web Apps (PWA) running on WebAssembly
- Windows store moves to PWA / WASM
- A new DOM alternative will emerge?
- JavaScript's monopoly will be lost, and it's popularity will fade



## 2020 - and beyond

- JavaScript will compile directly to WebAssembly, "use **wasm**"
- Native Android apps die-out in favour of Progressive Web Apps (PWA) running on WebAssembly
- Windows store moves to PWA / WASM
- A new DOM alternative will emerge?
- JavaScript's monopoly will be lost, and its popularity will fade
- The ubiquity of the web extends further still

# WebAssembly

## and the death of JavaScript?

...

Colin Eberhardt

