# WebAssembly

neither Web nor Assembly, but **Revolutionary**

**Jay Phelps** | @_jayphelps

The **WebAssembly revolution** has begun

# Jay Phelps

Chief Software Architect | THIS DOT

previously NETFLIX

@_jayphelps


**Support, Dev Rel, Staff Augmentation, Mentorship, and more**

www.thisdot.co

# So...**what is WebAssembly?** aka **Wasm**

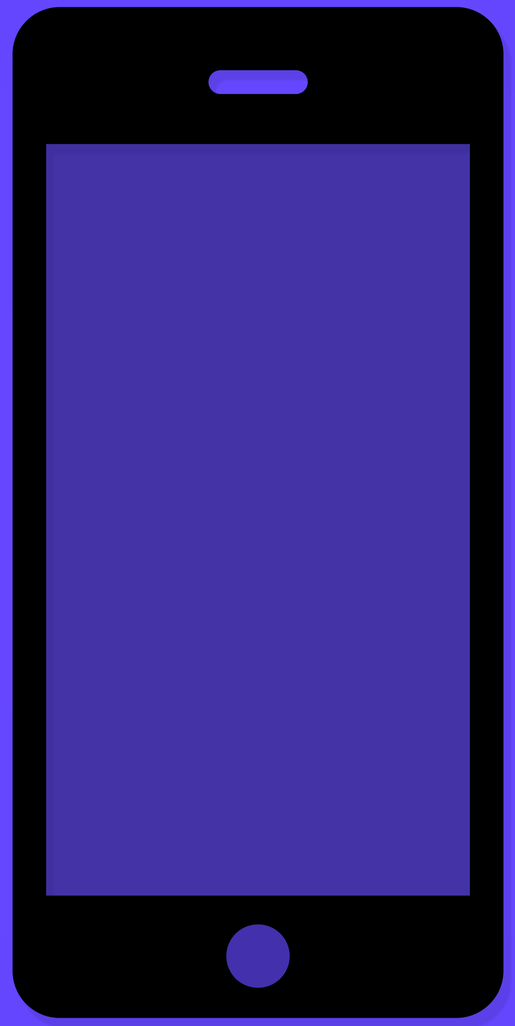
**Efficient, safe, low-level bytecode** for the Web

**Efficient**, safe, low-level bytecode for the Web

Fast to **load** and **execute**

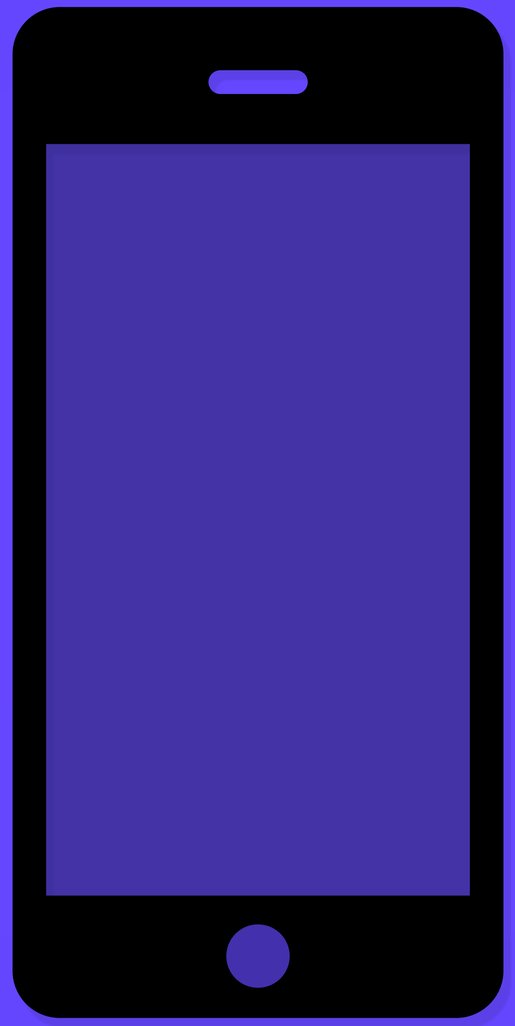
# Streaming compilation

compiled to machine code faster than it downloads



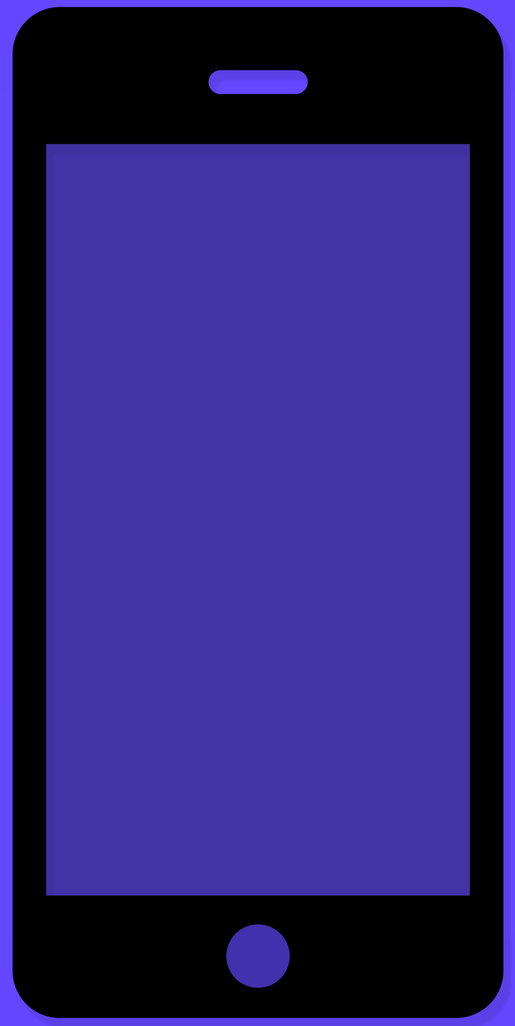
.wasm

```
(fu  $0 (type 0)
  i32.
  i32.load
)

(func $1 (type 0)
  i32.const 0
  i32.load
)

(func $2 (type 0)
  i32.const 0
  i32.load
)

(func $3 (type 0)
  i32.const 0
  i32.load
)
```



## .wasm

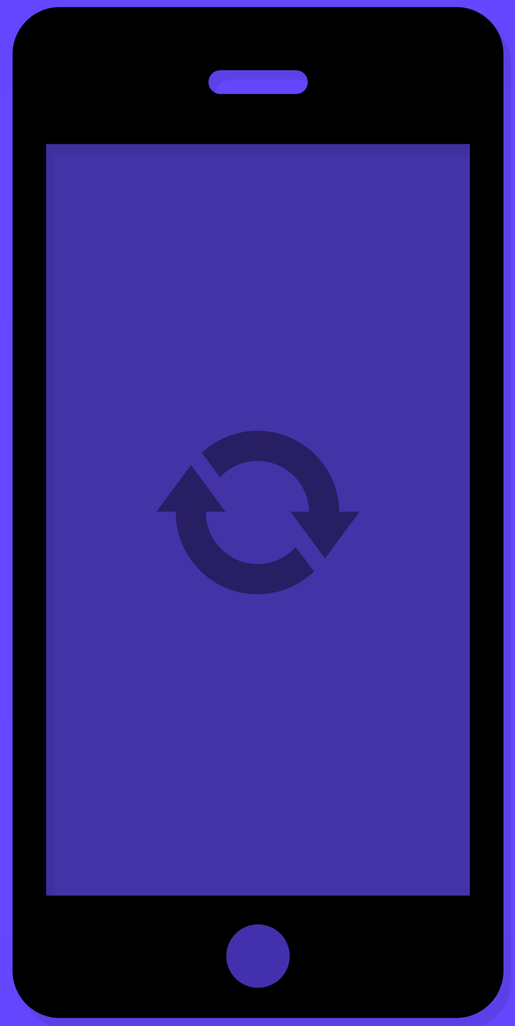
```
(func $0 (type 0)
  i32.const 0
  i32.load
)

(func $1 (type 0)
  i32.const 0
  i32.load
)

(func $2 (type 0)
  i32.const 0
  i32.load
)

(func $3 (type 0)
  i32.const 0
  i32.load
)
```



## machine code

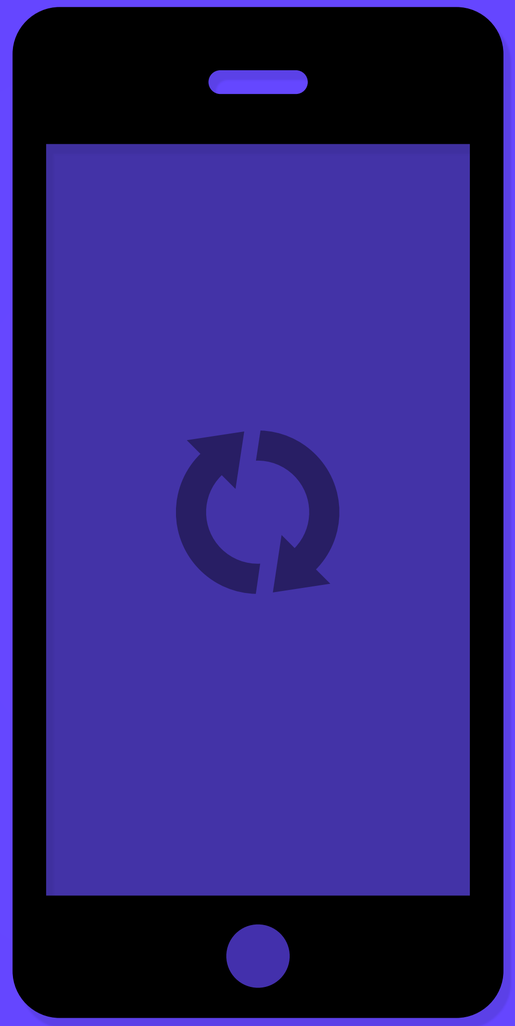
## .wasm

```
(func $0 (type 0)
  i32.const 0
  i32.load
)

(func $1 (type 0)
  i32.const 0
  i32.load
)

(func $2 (type 0)
  i32.const 0
  i32.load
)

(func $3 (type 0)
  i32.const 0
  i32.load
)
```



## machine code

.wasm



machine code

```
wasm-function[0]:
  sub rsp, 8
  mov eax, dword ptr [r15]
  nop
  add rsp, 8

wasm-function[1]:
  sub rsp, 8
  mov eax, dword ptr [r15]
  nop
  add rsp, 8

wasm-function[2]:
  sub rsp, 8
  mov eax, dword ptr [r15]
  nop
  add rsp, 8

wasm-function[3]:
  sub rsp, 8
  mov eax, dword ptr [r15]
  nop
  add rsp, 8
```

Efficient, **safe**, low-level bytecode for the Web

# **Sandboxed** and designed with **security in mind**

Control-flow integrity checks, stack protection,
dynamic dispatch table separate from linear memory

However, **does not prevent all** classes of **exploits**

Code reuse, side channel, race conditions, etc

Efficient, safe, **low-level bytecode** for the Web

WebAssembly is a **portable, binary instruction set** for a **virtual machine**

# 0x6a

01101010

(the i32.add instruction)

Intended (mostly) as a **compilation target**

```
int factorial(int n) {
   if (n == 0) {
      return 1;
   } else {
      return n * factorial(n - 1);
   }
}
```

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

```
00 61 73 6D 01 00 00 00 01
86 80 80 80 00 01 60 01 7F
01 7F 03 82 80 80 80 00 01
00 06 81 80 80 80 00 00 0A
9D 80 80 80 00 01 97 80 80
80 00 00 20 00 41 00 46 04
40 41 01 0F 0B 20 00 41 01
6B 10 00 20 00 6C 0B
```

**Efficient, safe, low-level bytecode** for the Web

# How did we get here?

Primary goals:

**languages other than JavaScript** and

**great—ideally improved—performance**

# Java Applets

Never truly integrated into browsers

# **Why not integrate the JVM** or **CLR?**

misaligned goals, mostly related to validation/compiling

# Portable Native Client (PNaCl)

lead by Google

# asm.js

lead by Mozilla

# C

```c
size_t strlen(char *ptr) {
  char *curr = ptr;
  while (*curr != 0) {
    curr++;
  }
  return (curr - ptr);
}
```

# asm.js

```asm
"use asm"
function strlen(ptr) {
  ptr = ptr|0;
  var curr = 0;
  curr = ptr;
  while (MEM8[curr]|0 != 0) {
    curr = (curr + 1)|0;
  }
  return (curr - ptr)|0;
}
```

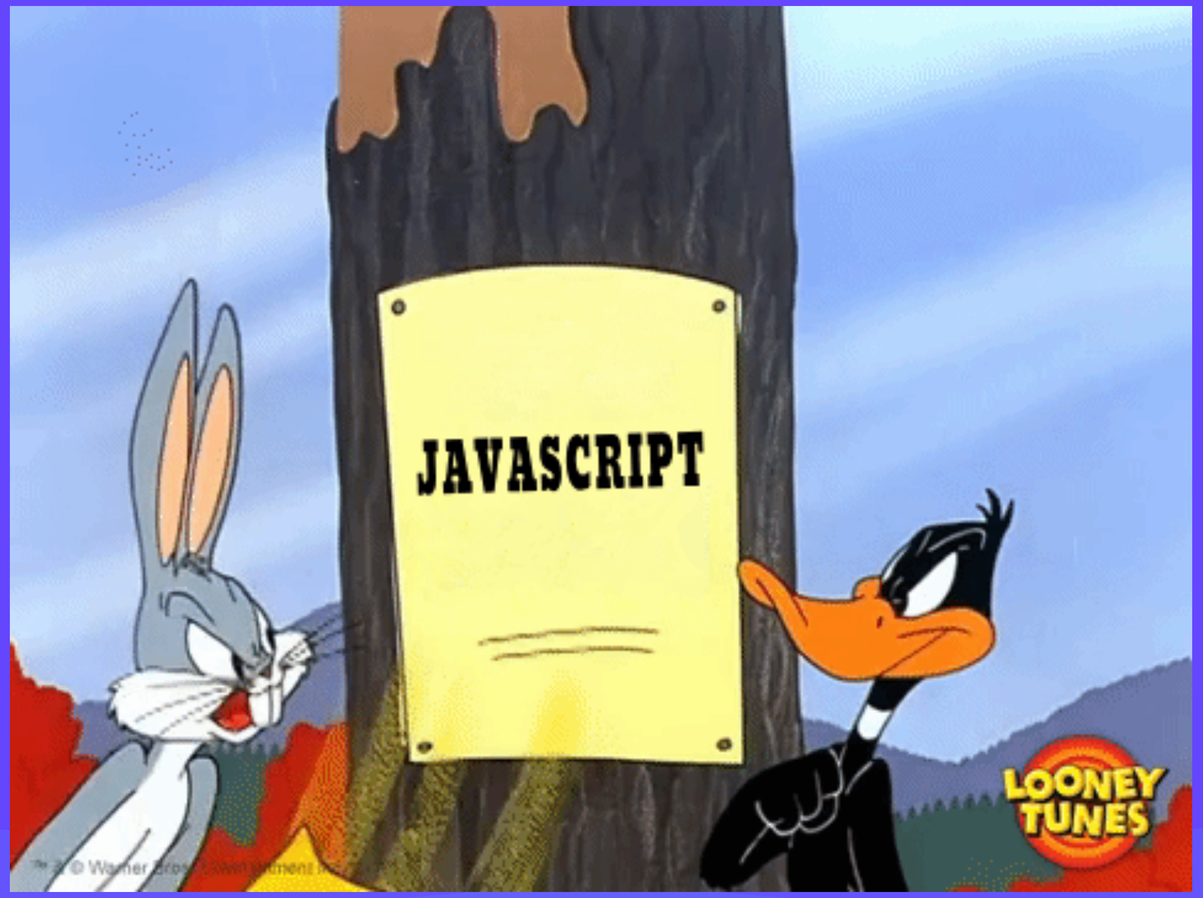
# WebAssembly

Jay Phelps | @_jayphelps

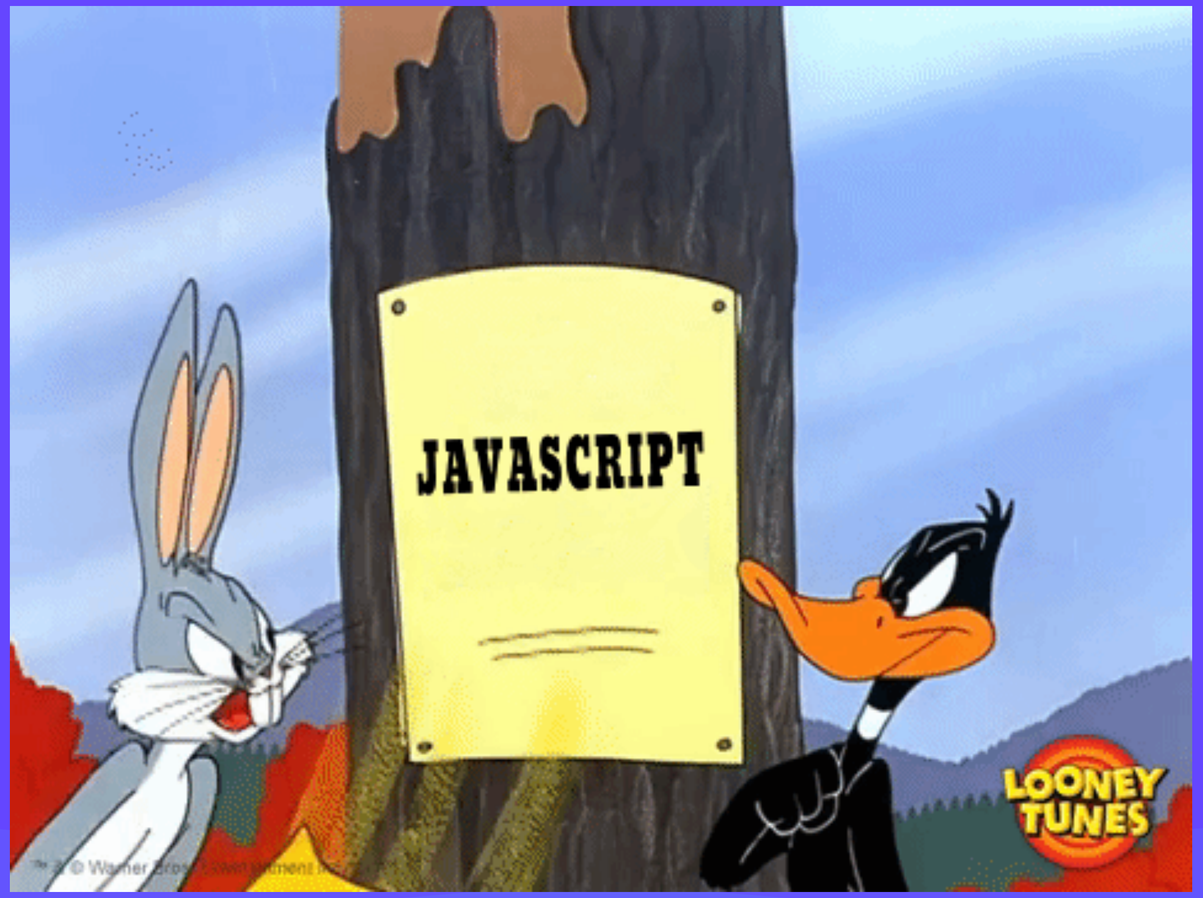
**WebAssembly** is an **unprecedented** collaboration

The **first open and standardized bytecode**

# Is it going to **kill JavaScript**?



# Is it going to **kill JavaScript**?



# Nope!

Will we compile **JavaScript** to **WebAssembly**?

**JavaScript is an *extremely* dynamic** language

**Brandon Dail**
@aweary

Following

💫 you can push into Array.prototype and totally mess up empty arrays

```
> Array.prototype.push("lol")
<· 1
> var empty = [];
<· undefined
> empty[0]
<· "lol"
```

Fully spec compliant **JavaScript compiled to WebAssembly** would be **slower**

…but a strict **subset** of **JavaScript** could be fast!

**Sebastian Markbåge**
@sebmarkbage

Following

Replying to @ken_wheeler

# What if you could AOT compile JS to native machine code and WebAssembly without a runtime or GC? 🤗

6:31 PM - 24 Jul 2018

**21** Retweets **144** Likes

# Prepack

## A tool for making JavaScript code run faster.

*Prepack is still in an early development stage and not ready for production use just yet. Please try it out, give feedback, and help fix bugs.

Getting Started Try It Out

## What does it do?

Prepack is a tool that optimizes JavaScript source code: Computations that can be done at compile-time instead of run-time get eliminated. Prepack replaces the global code of a JavaScript bundle with equivalent code that is a simple sequence of assignments. This gets rid of most intermediate computations and object allocations.

WebAssembly **v1 MVP is best suited for** languages like **C/C++** and **Rust**

Ideal for **relatively low-level, system languages**

Very little dynamic features at run-time, no GC

*Some* modern features of C++
don't perform ideal

**Exceptions** are the most common example

# But other languages are already **supported**, and more **planned**

Things like **Go**, **.NET**, **Java**, **OCaml**, and **even new ones**

WebAssembly will **impact language design and implementation**

# The Web requires unique considerations

**Rust team** has specifically called out
**WebAssembly as a priority**

# File sizes

as well as lazy-loading/code splitting, caching, etc

# Shared libraries

Traditional platforms like iOS/Android/macOS/Windows have more robust stdlibs and UI toolkits

# Offline

Caching story much more complex than desktop

# Interop with JavaScript

Languages which better interop with JS have major advantage

Jay Phelps | @_jayphelps

# Promising: **Dart, Elm, Reason**

Languages designed for the Web

**Jay Phelps** | **@_jayphelps**

a **TypeScript-like language**?

AssemblyScript is an early example

# AssemblyScript

```
export function factorial(n: i32): i32 {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

When should I **target WebAssembly right now**?

Heavily **CPU-bound number computations**

# Games

both **Unity** and **Unreal Engine** offer support

# Using **existing portable code**

e.g. video/audio decoders and other processing

hunspell

RLWE

McEliece

mcl

Zopfli

ttf2woff2

bcrypt

web-dsp

OpenCV

SPHINCS

SIDH

bls

XSalsa20

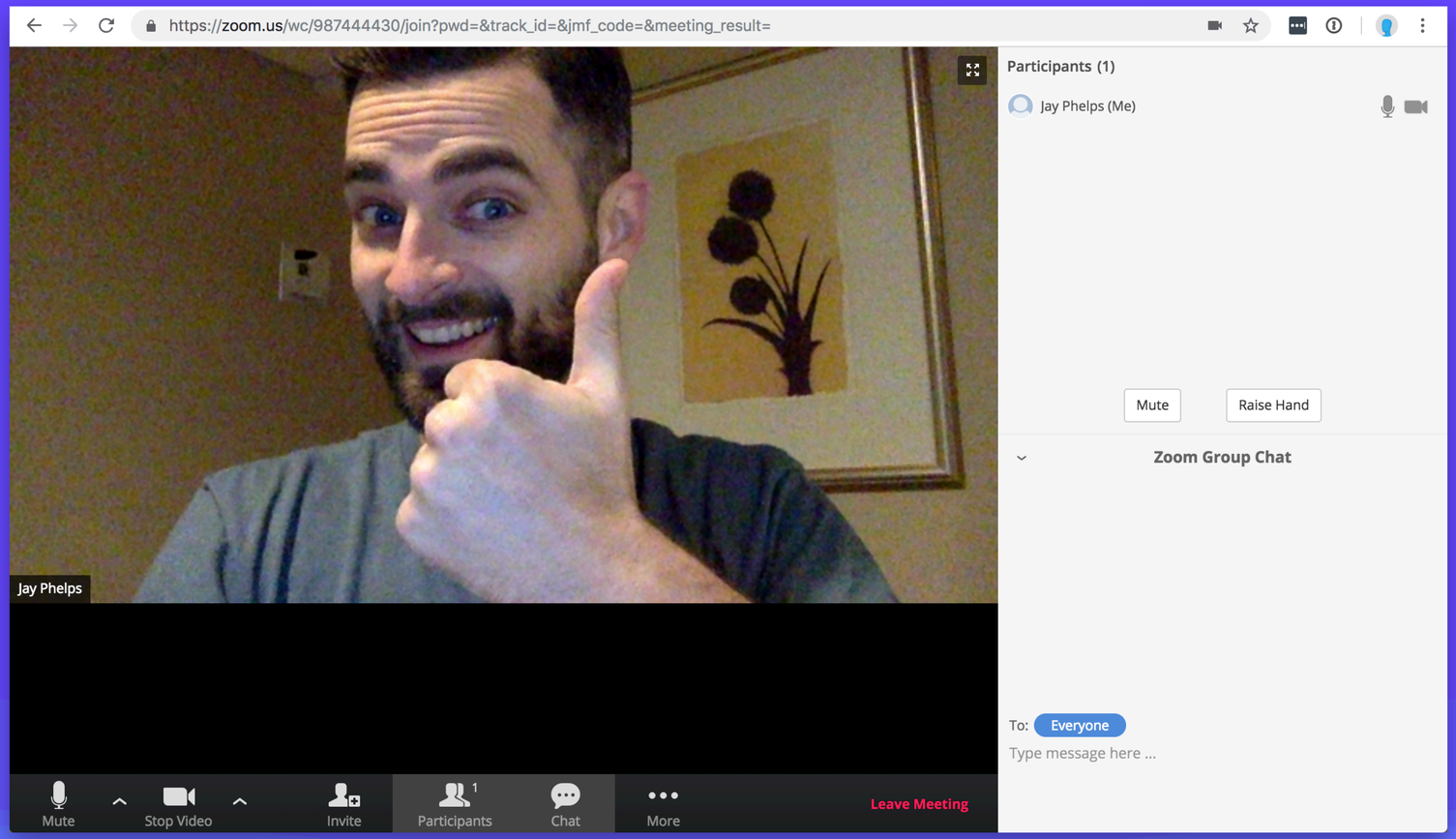
xxHash

GDAL

NTRU

# Zoom for Web client

Video conferencing powered by WebAssembly,
video/audio decoding off the main thread



# react-native-dom

(not react-native-web)

## React Native DOM · circleci passing npm package 0.4.1 maintained with lerna all contributors 9

An experimental, comprehensive port of React Native to the web.

- **Multithreaded by default:** Following the exact same architecture as React Native on mobile, all of your react components/app logic are run in web worker, leaving the main thread to entirely focus on rendering.
- **Same layout behavior as React Native on mobile:** Powered by custom bindings to Yoga and compiled to Web Assembly, avoid layout inconsistencies between your native and web projects.
- **Built with the same bundler used for existing React Native platforms:** Build both the "native" main and JS threads with the Metro Bundler along with all the developer experience features it provides.
- **Ecosystem compatible escape hatch to the DOM:** Using the same native module bridge, expose DOM-specific APIs in a more generic way that can easily be made into a cross-platform module.

To see it in action, check out these live demos:

- Movies Demo

Web UI developers are **probably already** using WebAssembly **without knowing it!**

# **source-map** npm package

**used by Firefox, Babel, create-react-app, LESS, etc**


# 10.9x faster!

Jay Phelps | @_jayphelps

# Other use cases are just around the corner

# What was that binary stuff?

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

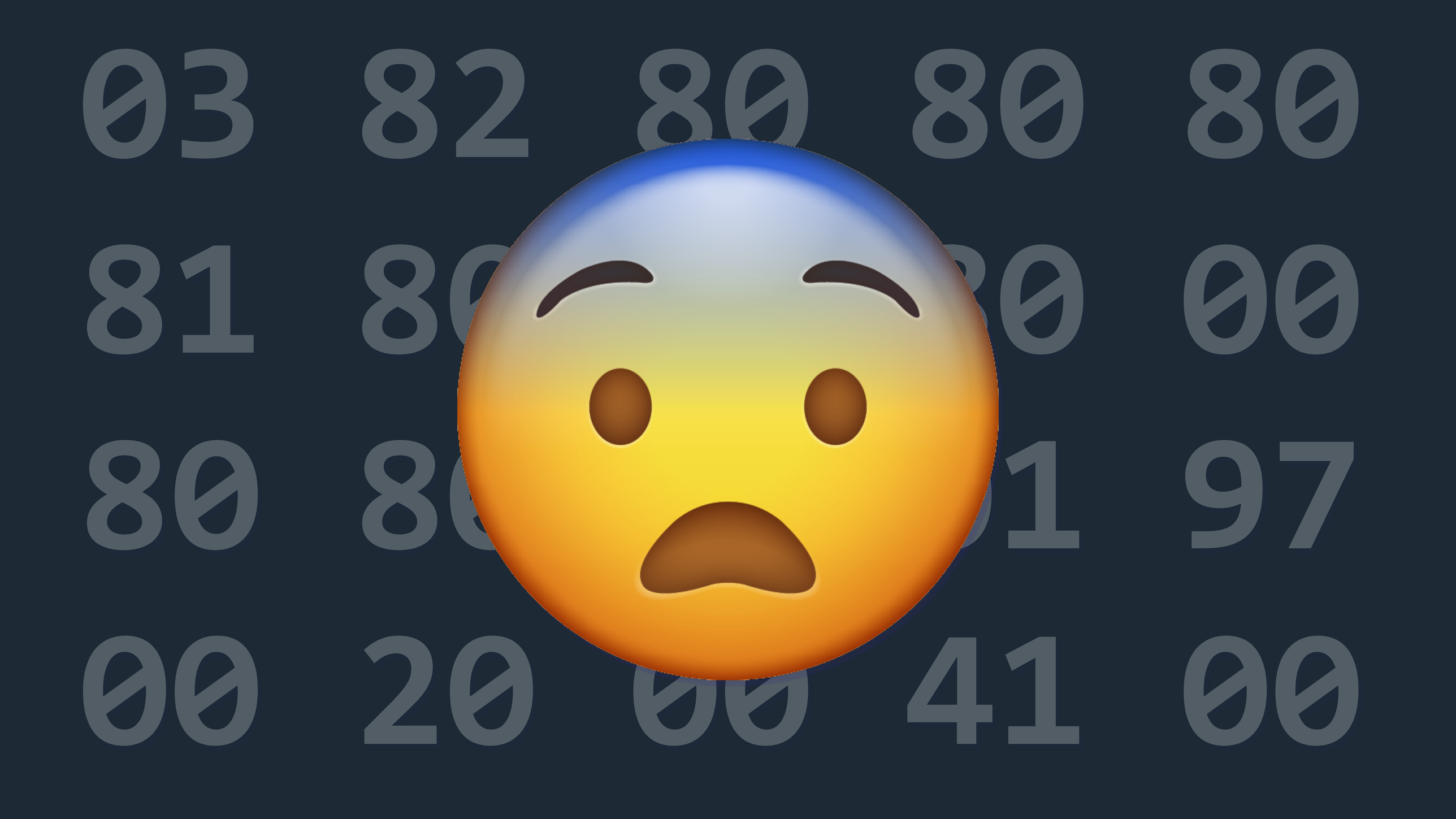
```
00 61 73 6D 01 00 00 00 01
86 80 80 80 00 01 60 01 7F
01 7F 03 82 80 80 80 00 01
00 06 81 80 80 80 00 00 0A
9D 80 80 80 00 01 97 80 80
80 00 00 20 00 41 00 46 04
40 41 01 0F 0B 20 00 41 01
6B 10 00 20 00 6C 0B
```

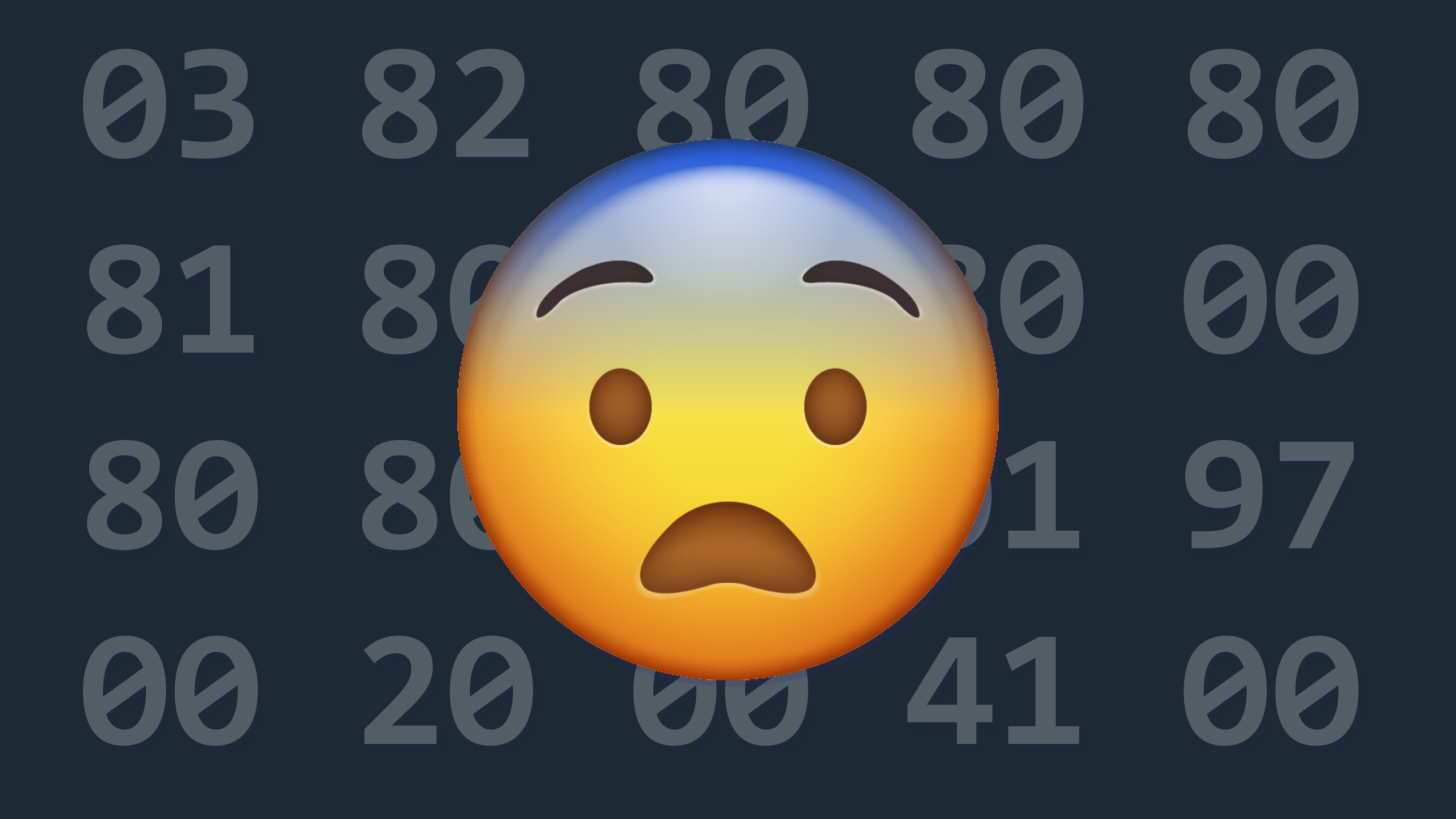
```
00 61 73 6D 01 00 00 00 01
86 80 80 80 00 01 60 01 7F
01 7F 03 82 80 80 80 00 01
00 06 81 80 80 80 00 00 0A
9D 80 80 80 00 01 97 80 80
80 00 00 20 00 41 00 46 04
40 41 01 0F 0B 20 00 41 01
6B 10 00 20 00 6C 0B
```

```
00 61 73 6D 01 00 00 00 01
86 80 80 80 00 01 60 01 7F
01 7F 03 82 80 80 80 00 01
00 06 81 80 80 80 00 00 0A
9D 80 80 80 00 01 97 80 80
80 00 00 20 00 41 00 46 04
40 41 01 0F 0B 20 00 41 01
6B 10 00 20 00 6C 0B
```

```
00 61 73 6D 01 00 00 00 01
86 80 80 80 00 01 60 01 7F
01 7F 03 82 80 80 80 00 01
00 06 81 80 80 80 00 00 0A
9D 80 80 80 00 01 97 80 80
80 00 00 20 00 41 00 46 04
40 41 01 0F 0B 20 00 41 01
6B 10 00 20 00 6C 0B
```

```
03 82 80 80 80
81 80 80 80 00
80 80 00 01 97
00 20 00 41 00
```


03 82 80 80 80
81 80 80 00
80 80 01 97
00 20 00 41 00


03 82 80 80 80
81 80 80 00
80 80 01 97
00 20 00 41 00

Binary can be *a little* intimidating

# Protip: don't worry about it

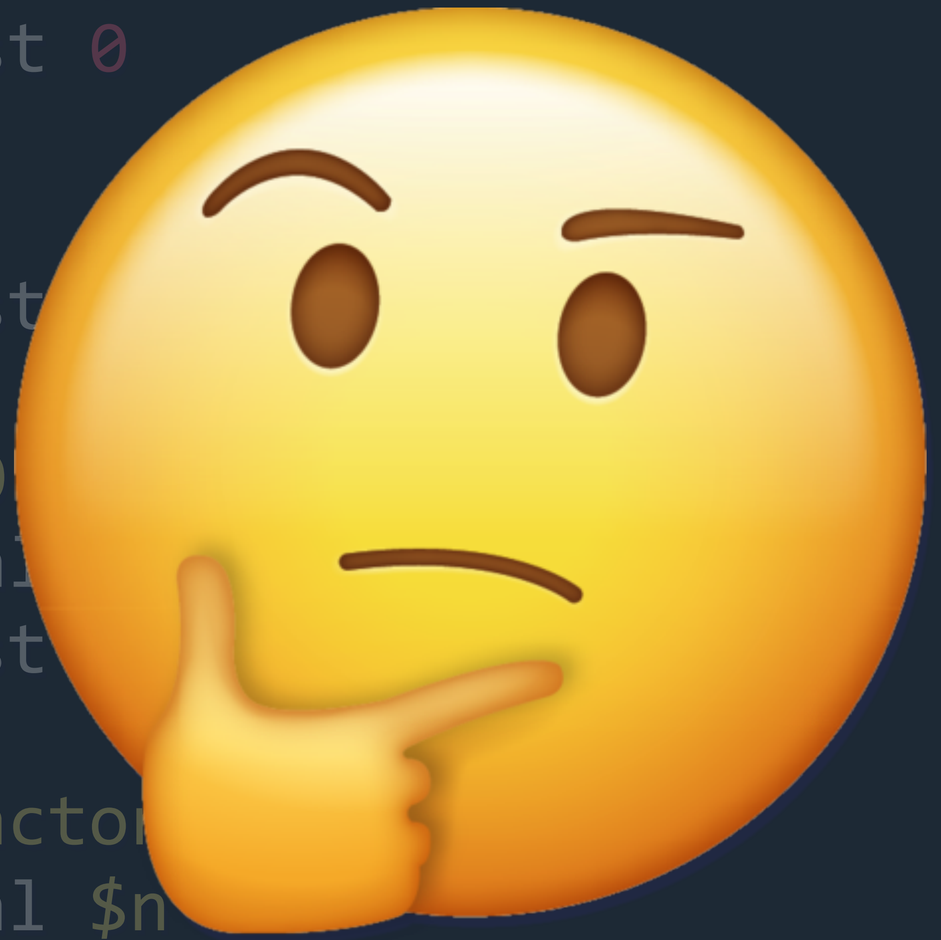
(unless of course, you want to)

**Tooling** will eventually **make it a non-issue**

**Textual representation** to the rescue!

```
(func $factorial (param $n i32) (result i32)
  get_local $n
  i32.const 0
  i32.eq
  if $if0
  i32.const 1
  return
  end $if0
  get_local $n
  i32.const 1
  i32.sub
  call $factorial
  get_local $n
  i32.mul
)
```

```
(func $factorial (param $n i32) (result i32)
  get_local $n
  i32.const 0
  i32.eq
  if $if0
  i32.const
  return
  end $if0
  get_local
  i32.const
  i32.sub
  call $factor
  get_local $n
  i32.mul
)
```



Let's **learn the fundamentals**

WebAssembly is a **stack machine**

...what's a **stack machine?**

# Stack

a **data structure** with two operations:

**push** and **pop**

stack machine: ***instructions*** **on a stack**

# **Why** a stack machine?

instead of AST, SSA, or register machine

**Smaller binary** encoding, **easier and faster single pass verification and VM** implementation

1 + 2

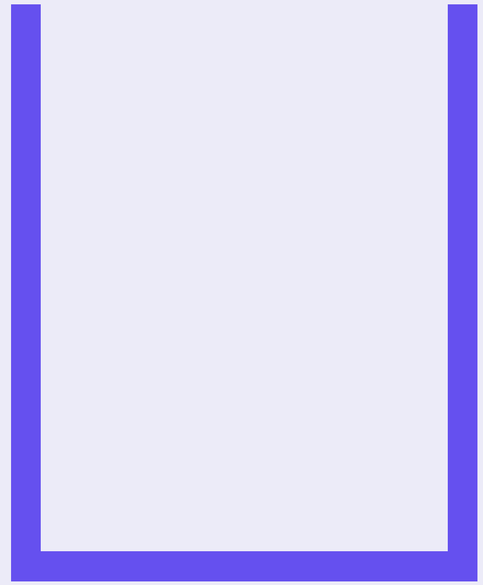
# opcode mnemonics

`i32.add`  `0x6a`

01101010

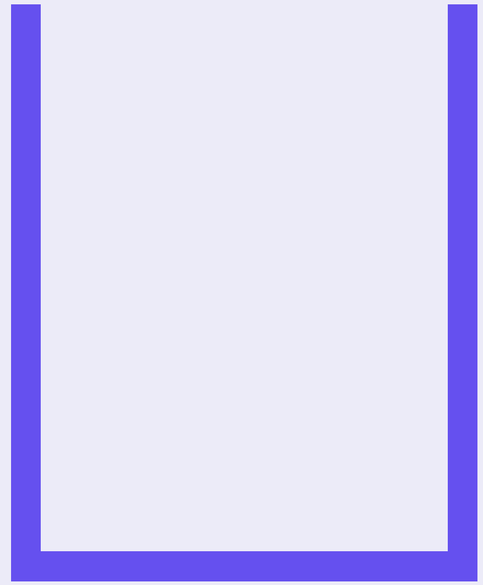
```
i32.const 1
i32.const 2
i32.add
```

```
i32.const 1
i32.const 2
i32.add
```



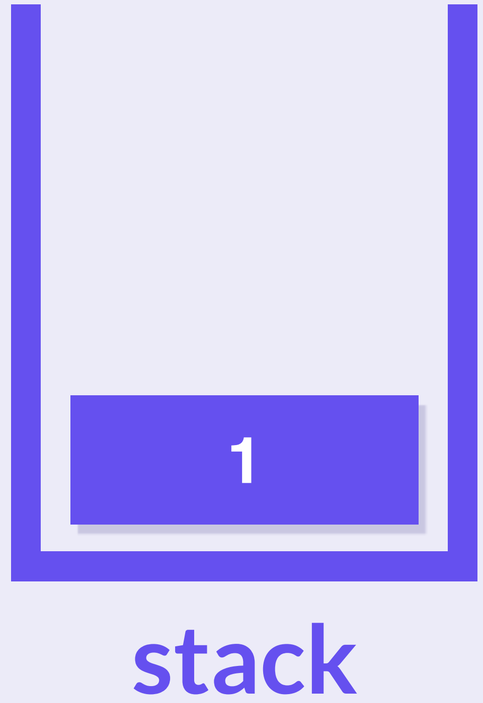
stack

```
i32.const 1
i32.const 2
i32.add
```

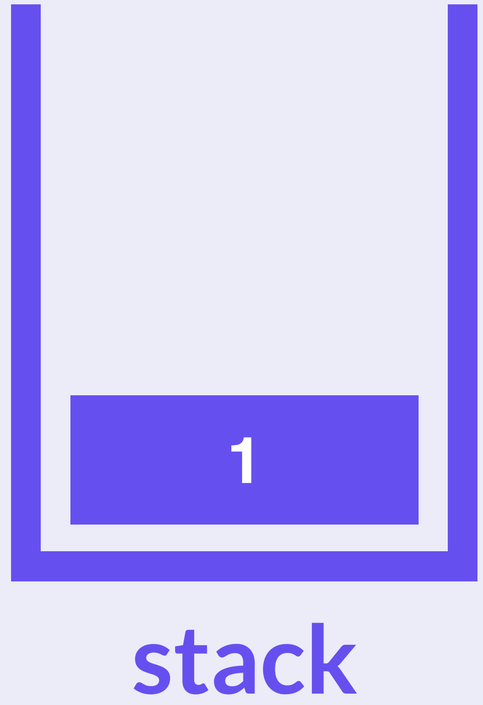


stack

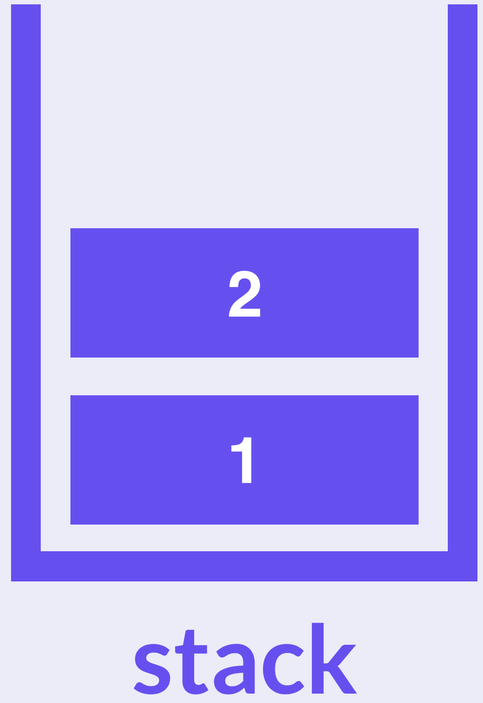
```
i32.const 1
i32.const 2
i32.add
```



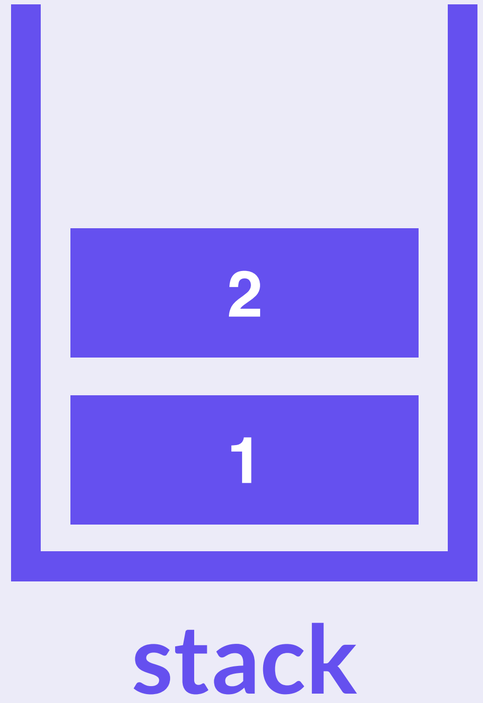
```
i32.const 1
i32.const 2
i32.add
```



```
i32.const 1
i32.const 2
i32.add
```



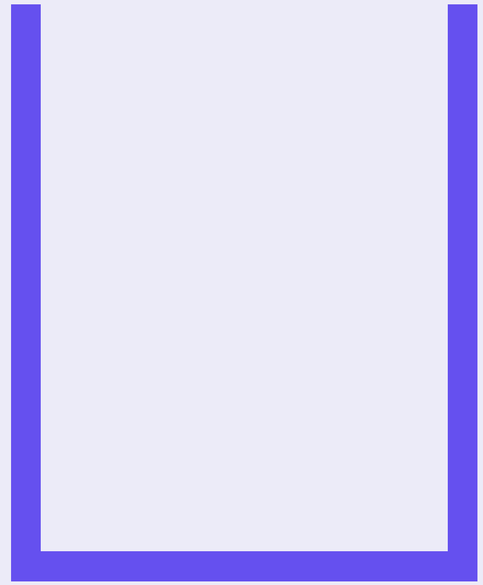
```
i32.const 1
i32.const 2
i32.add
```



```
i32.const 1
i32.const 2
i32.add
```

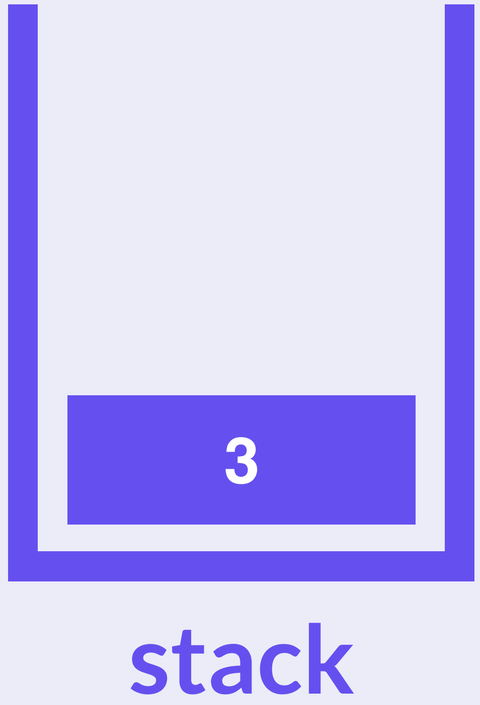
```
i32.const 1
i32.const 2
i32.add
```

3



stack

```
i32.const 1
i32.const 2
i32.add
```



```
i32.const 1
i32.const 2
i32.add
```

```
i32.const 1
i32.const 2
i32.add
call $log
```

# What's missing?

# **Direct access to Host APIs** (e.g. the **DOM**)

no direct access to sys calls, you have to call into JavaScript

# Garbage collection

necessary for better interop with JavaScript and WebIDL (e.g. the DOM)

# Multi-threading

SharedArrayBuffer re-enabled in Chrome 68

Jay Phelps | @_jayphelps

# Single Instruction Multiple Data (SIMD)

Hardware parallelization of vector computations

Jay Phelps | @_jayphelps

# Zero-cost exceptions

someday maybe even Algebraic Effects (!!!)

Jay Phelps | @_jayphelps

There's more, but **advancing quickly!**

How do I **get started**?

# webassembly.org

# https://github.com/mbasso/**awesome-wasm**

## Awesome Wasm 

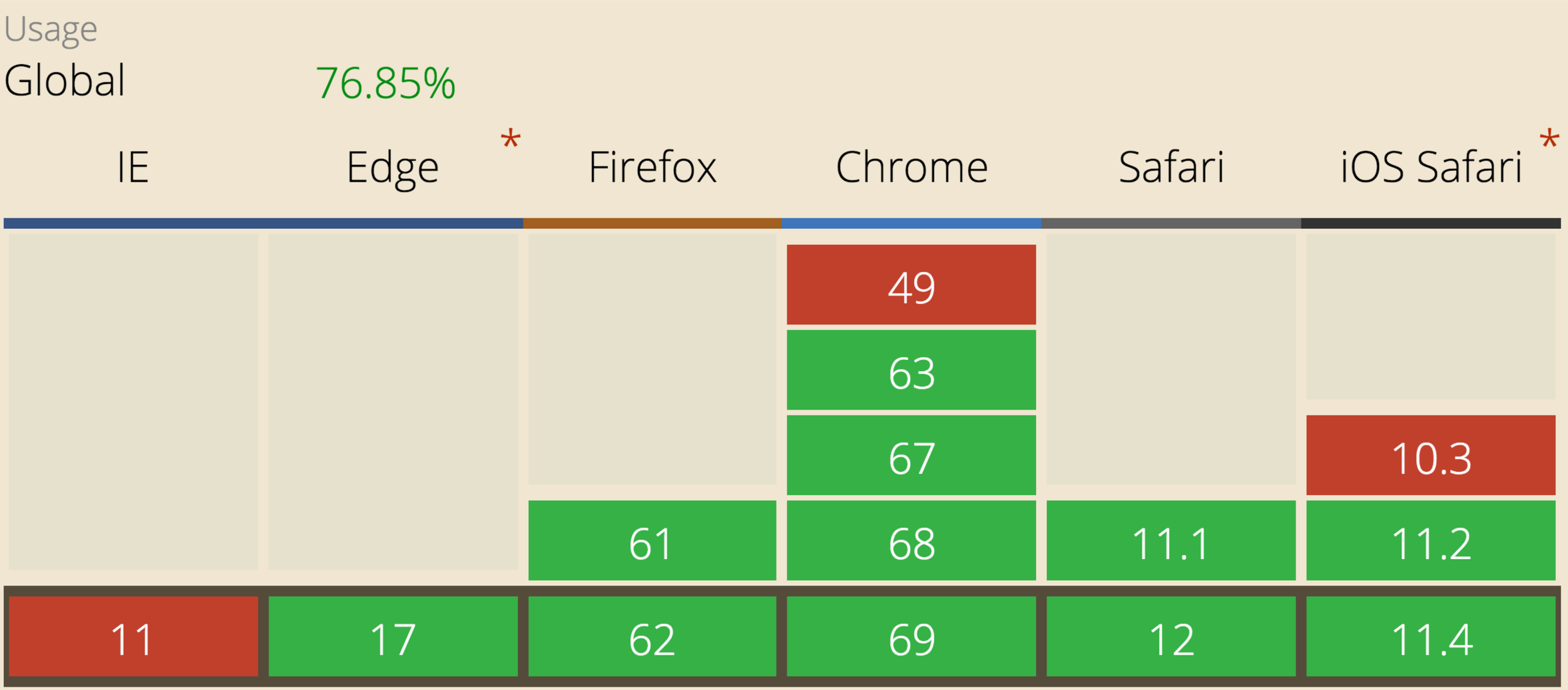

Collection of awesome things regarding WebAssembly (wasm) ecosystem.

Please read the contribution guidelines if you want to contribute.

### Contents


- General Resources
- Online Playground
- Tutorials
- Compilers
- Non-Web Embeddings
- Projects
  - Web frameworks-libraries
  - Data processing
  - WebGL
  - webpack
  - Browserify
  - Languages
  - node.js
  - Others
- Tools

# Supported in all modern browsers



The revolution is **just beginning**

**Efficient, safe, low-level bytecode** for the Web

Efficient, safe, low-level bytecode **for the Web**?

The **first open and standardized bytecode**

W3C Editor's Draft

TABLE OF CONTENTS


**1 Introduction**
1.1 Introduction
1.1.1 Design Goals
1.1.2 Scope
1.1.3 Dependencies
1.2 Overview
1.2.1 Concepts
1.2.2 Semantic Phases

**2 Structure**
2.1 Conventions
2.1.1 Grammar Notation
2.1.2 Auxiliary Notation
2.1.3 Vectors
2.2 Values
2.2.1 Bytes
2.2.2 Integers
2.2.3 Floating-Point
2.2.4 Names
2.3 Types
2.3.1 Value Types
2.3.2 Result Types
2.3.3 Function Types
2.3.4 Limits
2.3.5 Memory Types
2.3.6 Table Types


# WebAssembly Core Specification

## Editor's Draft, 8 August 2018


**This version:**
https://webassembly.github.io/spec/core/bikeshed/

**Latest published version:**
https://www.w3.org/TR/wasm-core-1/



**Editor:**
Andreas Rossberg (Dfinity Stiftung)



**Issue Tracking:**
GitHub Issues




## Abstract


This document describes version 1.0 of the core WebAssembly standard, a safe, portable, low-level code mat designed for efficient execution and compact representation.

Part of a collection of related documents: the Core WebAssembly Specification, the WebAssembly JS Inte face, and the WebAssembly Web API.


## Status of this document

tures, desktop or mobile devices and embedded systems alike.
    - **Language-independent**: does not privilege any particular language, programming model, or object model.
    - **Platform-independent**: can be embedded in browsers, run as a stand-alone VM, or integrated in other environments.
    - **Open**: programs can interoperate with their environment in a simple and universal manner.
- Efficient and portable *representation*:
    - **Compact**: has a binary format that is fast to transmit by being smaller than typical text or native code formats.
    - **Modular**: programs can be split up in smaller parts that can be transmitted, cached, and consumed separately.

WebAssembly is **not just for the Web**!

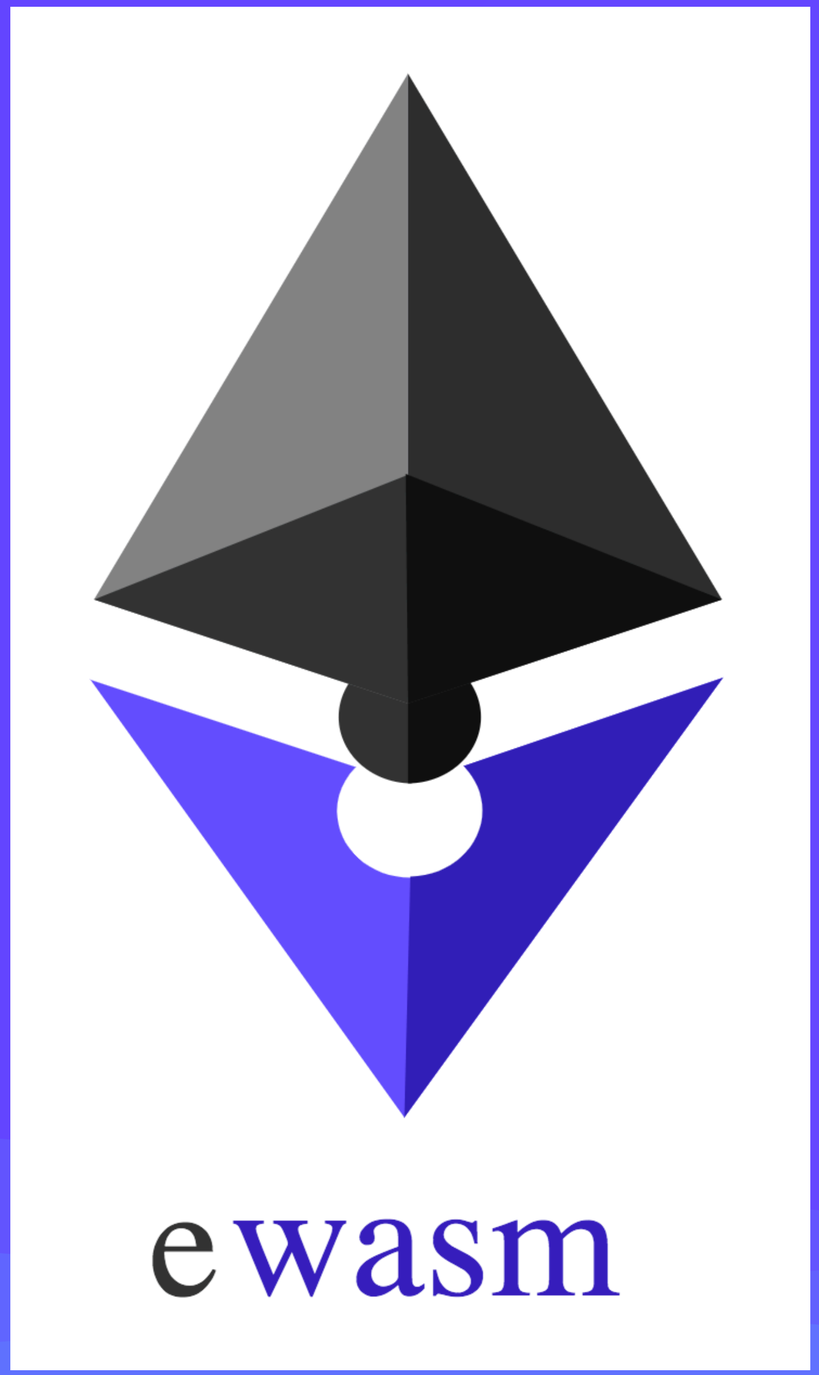
**JF Bastien**

@jfbastien

# WebAssembly: neither Web nor Assembly.

9:53 AM - 30 Jun 2017

**9** Retweets **75** Likes




# ewasm

**Etherium**-flavored WebAssembly VM for running distributed **smart contracts**

```
$ cat selfpipe.c
#include <stdio.h>
#include <unistd.h>

char buf[4096];

int main(int argc, char *argv[])
{
        int pipes[2], ret;
        size_t i;

        ret = pipe(pipes);
        if (ret) {
                perror("pipe");
                return -1;
        }

        for (i = 0; i < 16ULL * 1024ULL * 1024ULL * 1024 / 4096; ++i) {
                ret = write(pipes[1], buf, sizeof(buf));
                if (ret < 0) {
                        perror("write");
                        return -1;
                }

                ret = read(pipes[0], buf, sizeof(buf));
                if (ret < 0) {
                        perror("read");
                }
        }

        return 0;
}
$ cc -O3 -o selfpipe selfpipe.c
$ time ./selfpipe

real    0m4.496s
user    0m0.948s
sys     0m3.546s
$ emcc -O3 -o selfpipe.js selfpipe.c
$ time ./wasmjit selfpipe.wasm

real    0m2.025s
user    0m0.004s
sys     0m2.019s
$
```

# rianhunter/wasmjit

VM and Linux kernel module for

**running WebAssembly in “ring 0”**


# nebulet

**microkernel** that runs WebAssembly exclusively

**Efficient, safe, low-level bytecode** for the Web

# Efficient, safe, low-level bytecode ~~for the Web~~

# Thanks!

 @_jayphelps