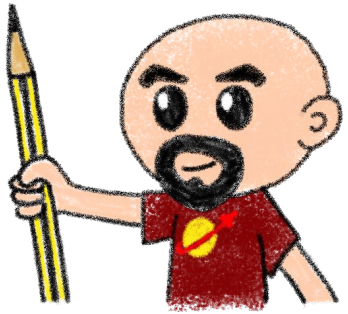
GDG LILLE

WebAssembly for Developers (web... or not)

Horacio Gonzalez

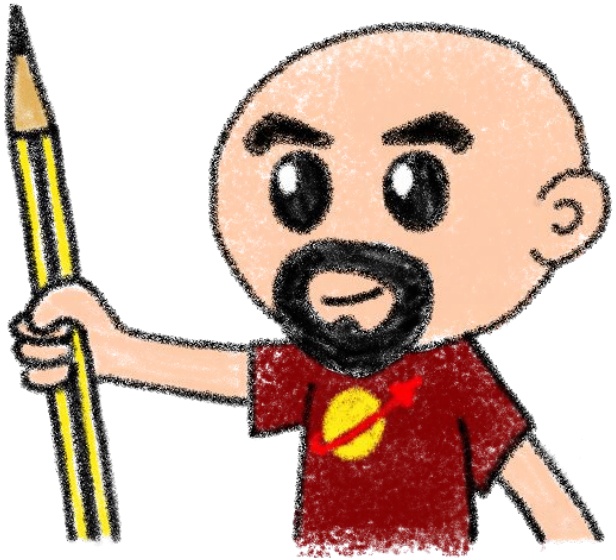
@LostInBrittany



Horacio Gonzalez

@LostInBrittany

Spaniard lost in Brittany, developer, dreamer and all-around geek



OVHcloud

Team DevRel

DevFest du Bout du Monde

Finist Devs

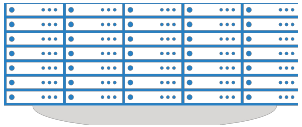
Google Developers Experts 2019

Web Technologies GDE

OVHcloud: A Global Leader

250k Private cloud VMs running

1 **Dedicated IaaS Europe**



Hosting capacity : **1.3M** Physical Servers

360k Servers already deployed

Own **20Tbps** Netwok with **35 PoPs**

30 Datacenters

> **1.3M** Customers in **138** Countries

OVHcloud: Our solutions

Cloud

- VPS
- Public Cloud
- Private Cloud
- Serveur dédié
- Cloud Desktop
- Hybrid Cloud

Mobile Hosting

- Containers
- Compute
- Database
- Object Storage
- Securities
- Messaging

Web Hosting

- Domain names
- Email
- CDN
- Web hosting
- MS Office
- MS solutions

Telecom

- VoIP
- SMS/Fax
- Virtual desktop
- Cloud HubiC
- Over theBox

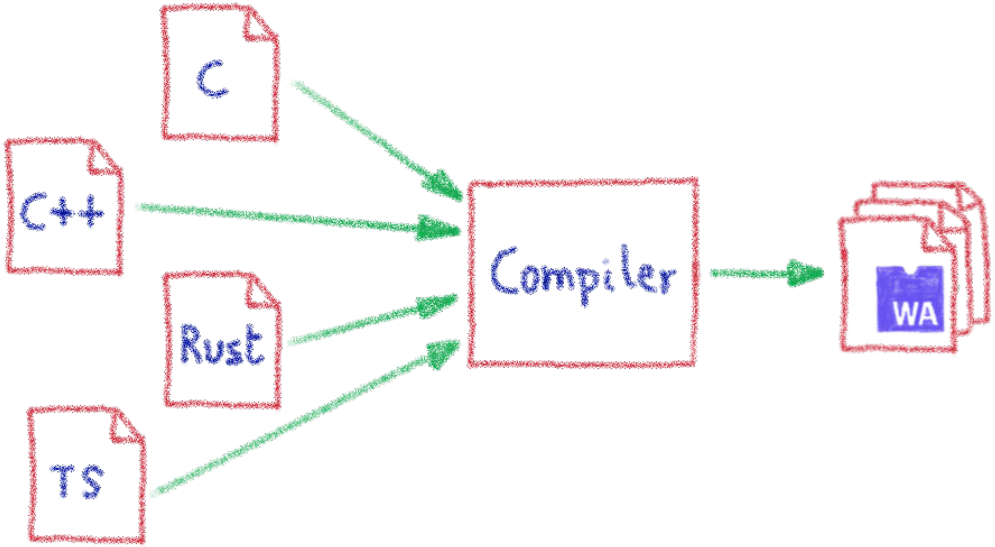
Did I say WebAssembly?

WASM for the friends...

WebAssembly, what's that?

Let's try to answer those (and other) questions...

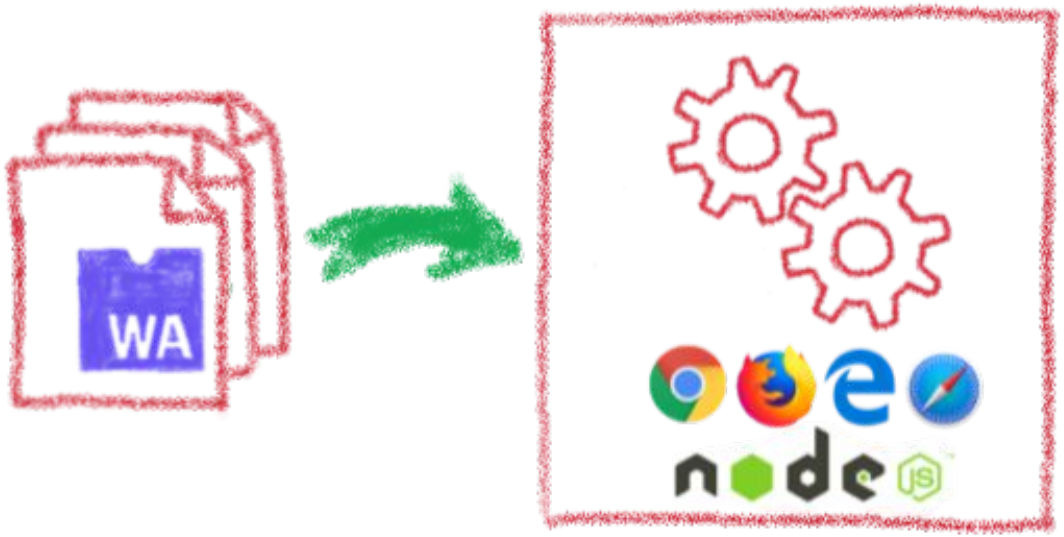
A low-level binary format for the web



Not a programming language

A compilation target

That runs on a stack-based virtual machine



A portable binary format that runs on all modern browsers...
but also on NodeJS!

With several key advantages

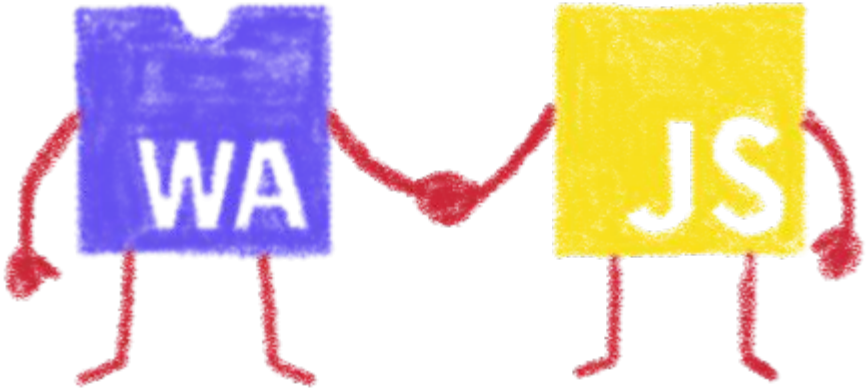
Fast & Efficient

Memory-safe & Sandboxed

Open & Deboggable

WWW Part of the Web Platform

But above all...



WebAssembly is not meant to replace JavaScript

Who is using WebAssembly today?

And many more others...

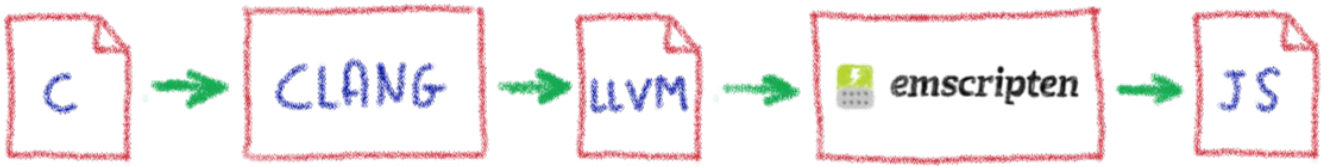
A bit of history

Remembering the past
to better understand the present

Executing other languages in the browser

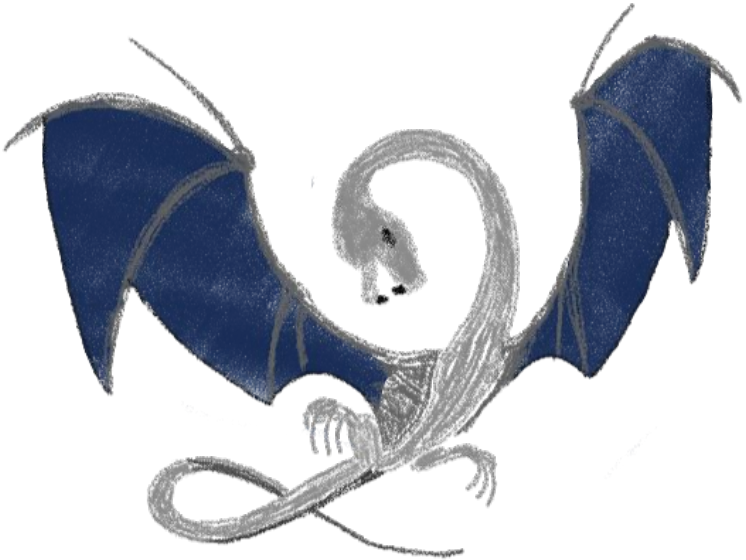
A long story, with many failures...

2012 - From C to JS: enter emscripten



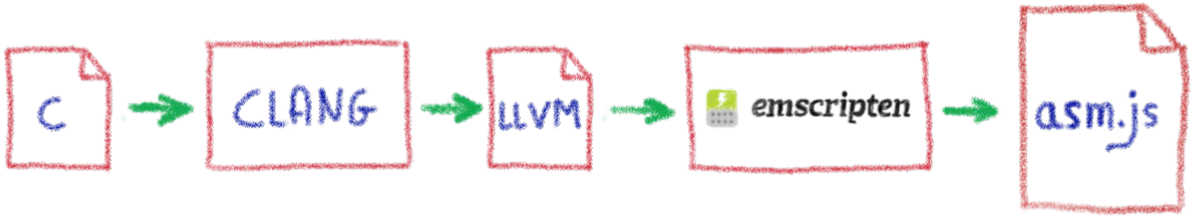
Passing by LLVM pivot

Wait, dude! What's LLVM?



A set of compiler and toolchain technologies

2013 - Generated JS is slow...



Let's use only a strict subset of JS: asm.js

Only features adapted to AOT optimization

WebAssembly project

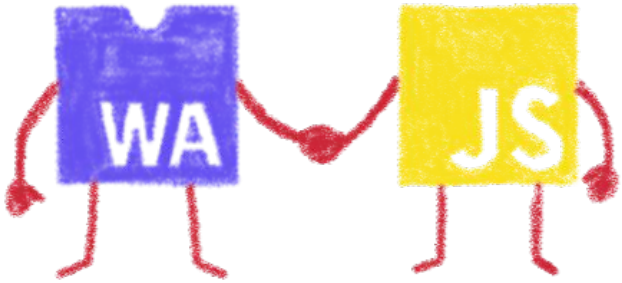
moz://a

Google

Microsoft

W3C

Joint effort



Hello W(ASM)orld

My first WebAssembly program

Do you remember your 101 C course?

```c
#include <stdio.h>

int main(int argc, char ** argv) {
  printf("Hello, world!\n");
}

```

A simple *HelloWorld* in C

We compile it with emscripten

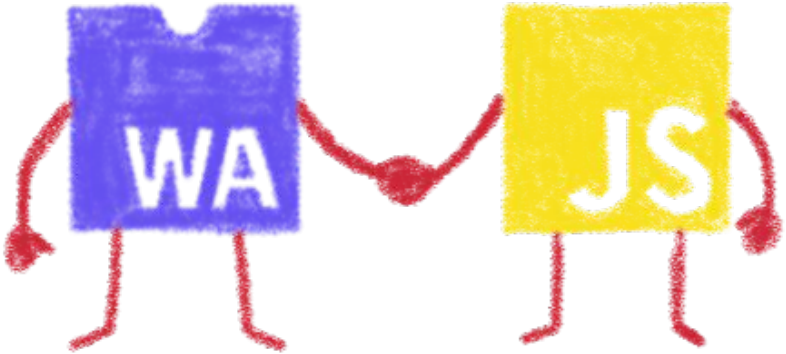
horacio@DESKTOP-6KHP1S2: ~/git/wasm/hello_world | horacio@DESKTOP-6KHP1S2: ~/git/emsdk

```
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$ emcc hello_world.c -o hello_world.html
cache:INFO: generating system asset: is_vanilla.txt... (this will be cached in "/home/horacio/.emscripten_cache/is_vanil
la.txt" for subsequent builds)
cache:INFO:  - ok
shared:INFO: (Emscripten: Running sanity checks)
cache:INFO: generating system library: libcompiler_rt.bc... (this will be cached in "/home/horacio/.emscripten_cache/asm
js/libcompiler_rt.bc" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libc-wasm.bc... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/li
bc-wasm.bc" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libdlmalloc.a... (this will be cached in "/home/horacio/.emscripten_cache/asmjs/l
ibdlmalloc.a" for subsequent builds)
cache:INFO:  - ok
cache:INFO: generating system library: libpthreads_stub.bc... (this will be cached in "/home/horacio/.emscripten_cache/a
smjs/libpthreads_stub.bc" for subsequent builds)
cache:INFO:  - ok
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$ ls
hello_world.c  hello_world.html  hello_world.js  hello_world.wasm
horacio@DESKTOP-6KHP1S2:~/git/wasm/hello_world$
```

We get a .wasm file...

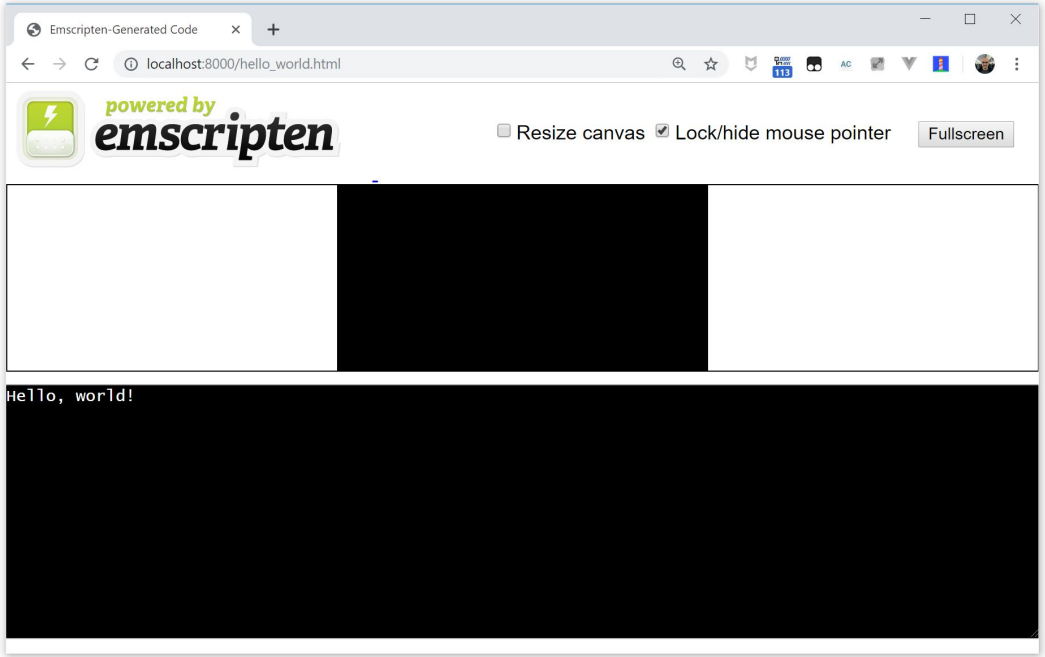
Binary file, in the binary WASM format

We also get a `.js` file...



Wrapping the WASM

And a .html file



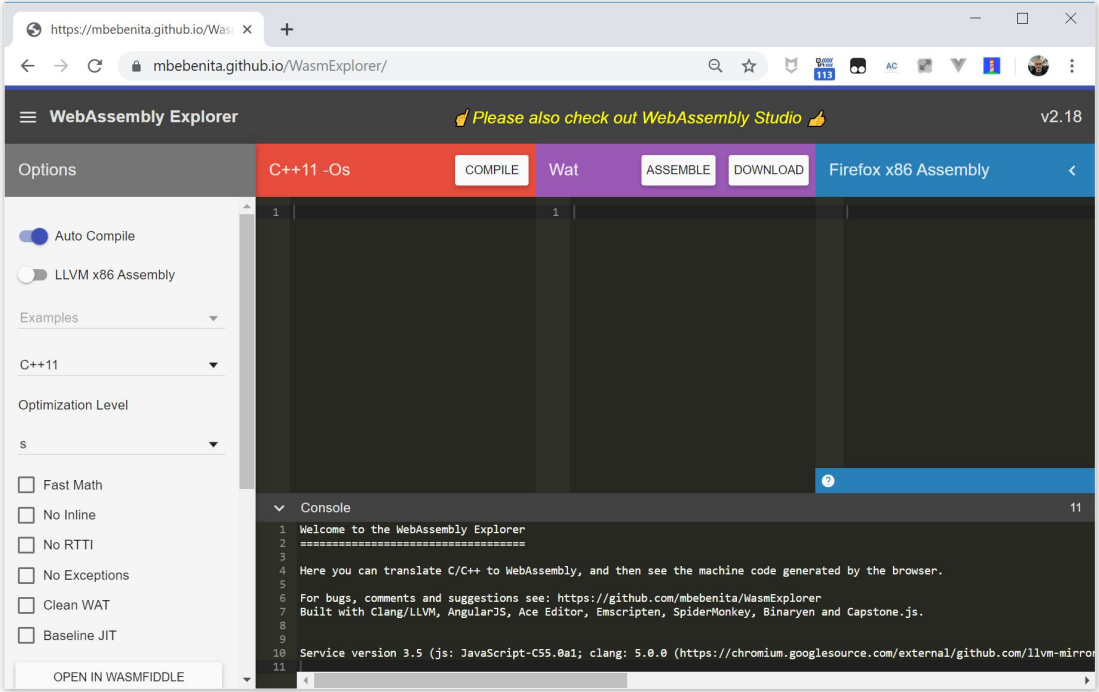
To quickly execute in the browser our WASM

And in a more Real World™ case?

A simple process:

- Write or use existing code
  - In C, C++, Rust, Go, AssemblyScript...
- Compile
  - Get a binary `.wasm` file
- Include
  - The `.wasm` file into a project
- Instantiate
  - Async JavaScript retrieving and instantiating the `.wasm` binary

I think I need a real example now



Let's use WASM Explorer

https://mbebenita.github.io/WasmExplorer/

Let's begin with the a simple function

C99 -Os | COMPILE

```c
int squarer(int num) {
  return num*num;
}
```

Wat | ASSEMBLE | DOWNLOAD

```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "squarer" (func $squarer))
 (func $squarer (; 0 ;) (param $0 i32) (result
   i32)
  (i32.mul
   (get_local $0)
   (get_local $0)
  )
 )
)
```

Firefox x86 Assembly

```
wasm-function[0]:
  sub rsp, 8                    ; 0x000000
  mov edx, edi                  ; 0x000004
  mov ecx, edx                  ; 0x000006
  mov eax, edx                  ; 0x000008
  imul ecx, eax                 ; 0x00000a
  mov eax, ecx                  ; 0x00000c
  nop                           ; 0x00000f
  add rsp, 8                    ; 0x000011
  ret                           ; 0x000015
```

WAT: WebAssembly Text Format

Human readable version of the `.wasm` binary

Download the binary `.wasm` file

Now we need to call it from JS...

Instantiating the WASM

1. Get the `.wasm` binary file into an array buffer
2. Compile the bytes into a WebAssembly module
3. Instantiate the WebAssembly module

Instantiating the WASM

wasm > squarer > JS squarer.js > ...

```
3   var importObject = {
4       imports: {
5         imported_func: function(arg) {
6           console.log(arg);
7         }
8       }
9     };
10
11  async function loadWebAssembly() {
12      let response = await fetch('squarer.wasm');
13      let arrayBuffer = await response.arrayBuffer();
14      let wasmModule = await WebAssembly.instantiate(arrayBuffer, importObject);
15      squarer = await wasmModule.instance.exports._Z7squareri;
16      console.log('Finished compiling! Ready when you are...');
17  }
18
19  loadWebAssembly();
```

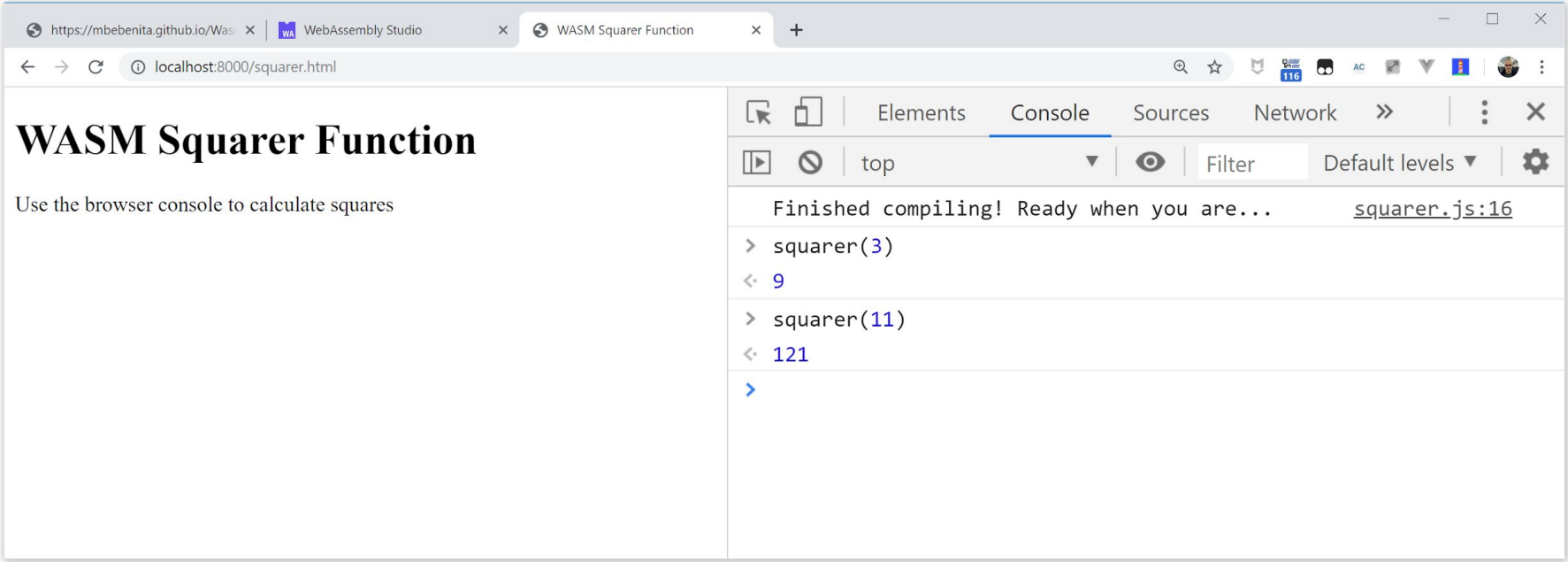
Loading the squarer function

wasm > squarer > squarer.html > ...

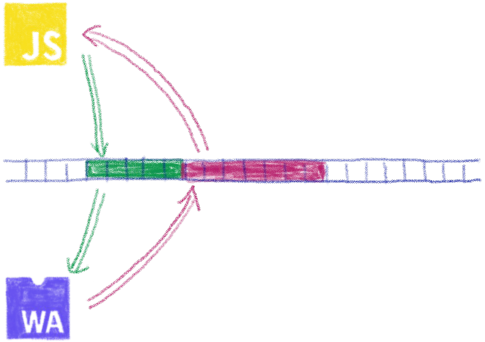
```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>WASM Squarer Function</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>

    <h1>WASM Squarer Function</h1>

    <script src="squarer.js"></script>

    <p>Use the browser console to calculate squares</p>
</body>
</html>

```

We instantiate the WASM by loading the wrapping JS

Using it!



Directly from the browser console (it's a simple demo...)



Communicating between JS and WASM

Shared memory, functions...

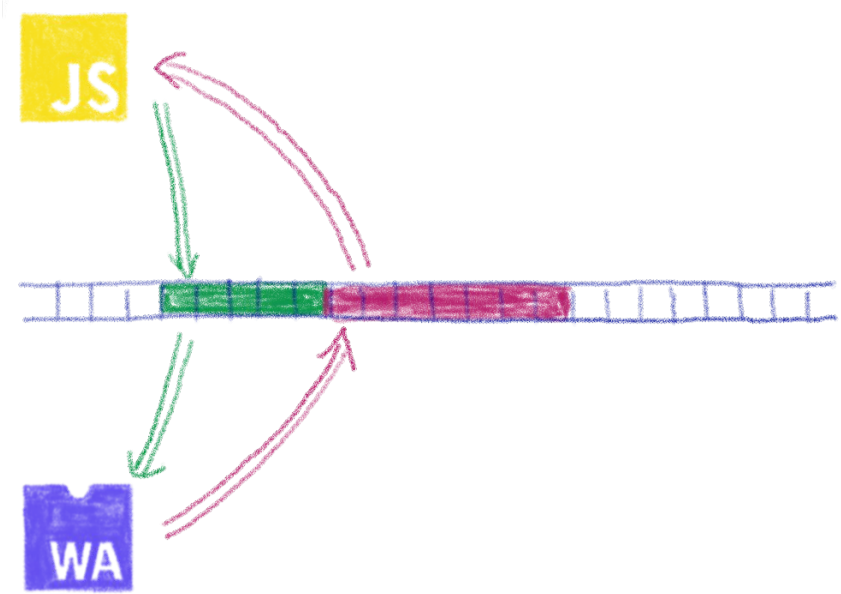
Native WASM types are limited

WASM currently has four available types:

- `i32`: 32-bit integer
- `i64`: 64-bit integer
- `f32`: 32-bit float
- `f64`: 64-bit float

Types from languages compiled to WASM are mapped to these types

How can we share data?



Using the same data in WASM and JS?

Shared linear memory between them!

Some use cases

What can I do with it?

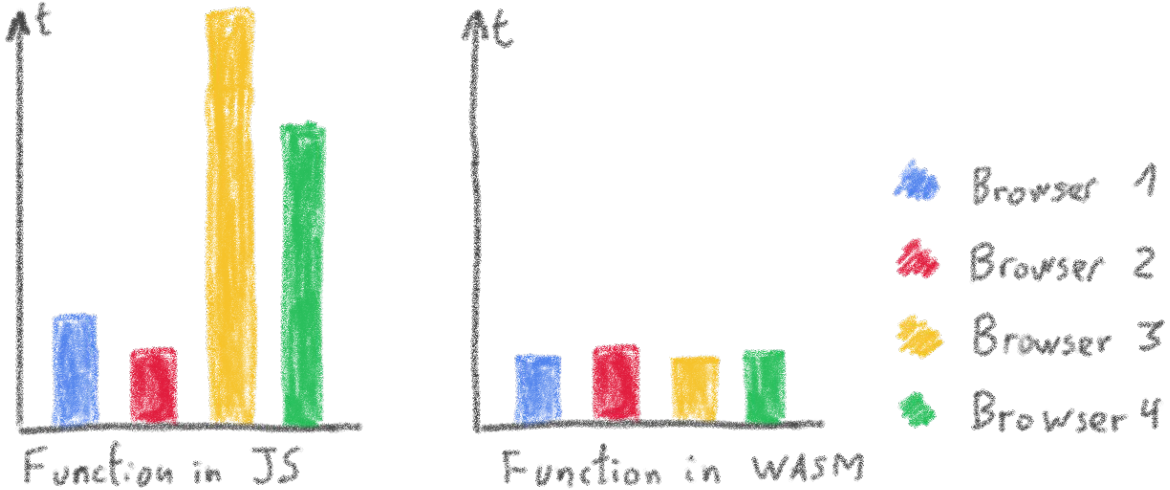
Tapping into other languages ecosystems

SQUOOSH.APP

OptiPNG (C)
Resize (Rust)
MozJPEG (C++)
webp (C)

Don't rewrite libs anymore

Replacing problematic JS bits



Predictable performance

Same peak performance, but less variation

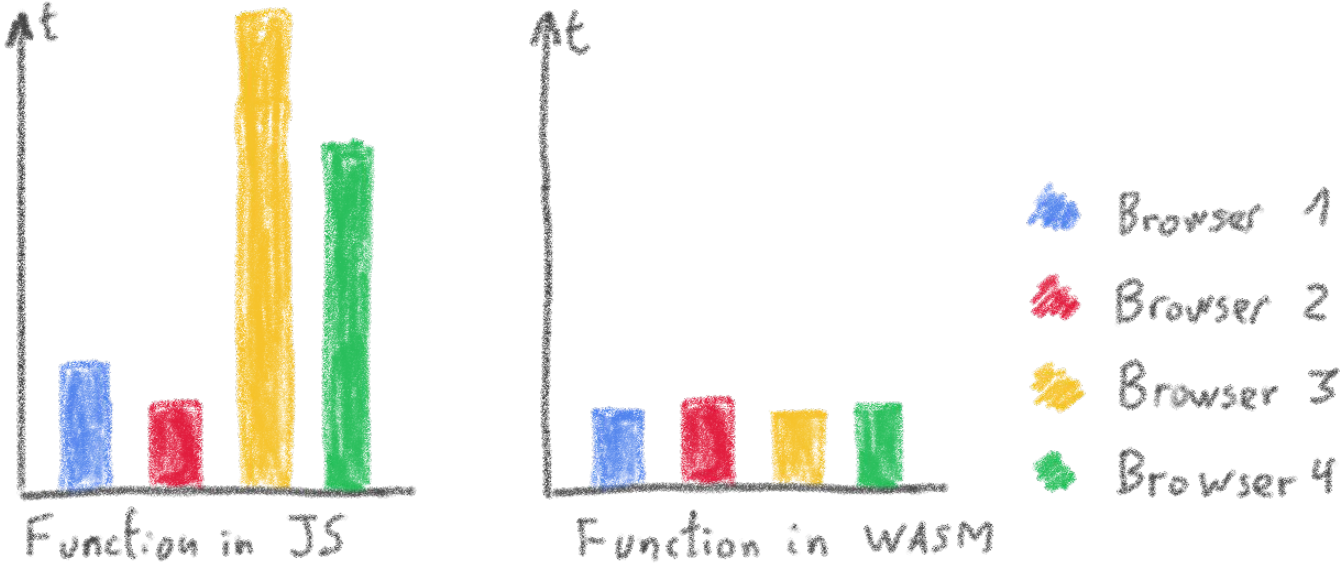
AssemblyScript

Writing WASM without learning a new language

TypeScript subset compiled to WASM

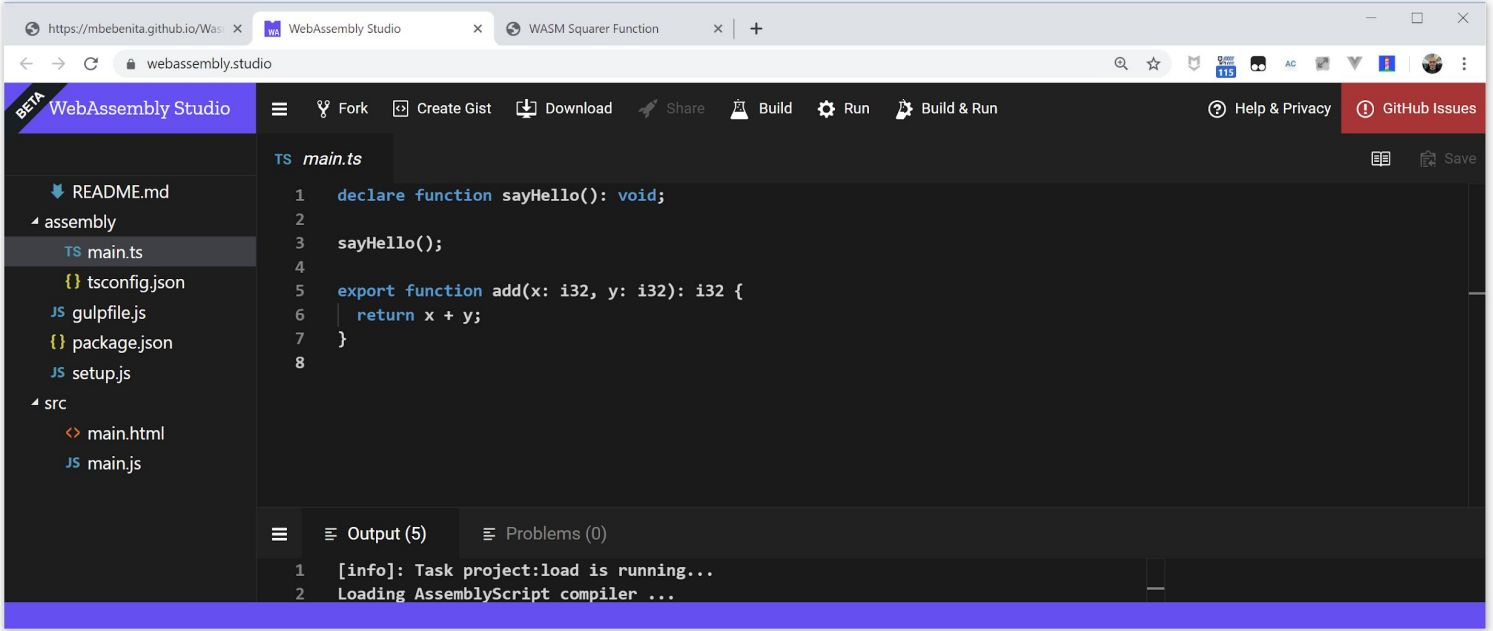
Why would I want to compile TypeScript to WASM?

Ahead of Time compiled TypeScript



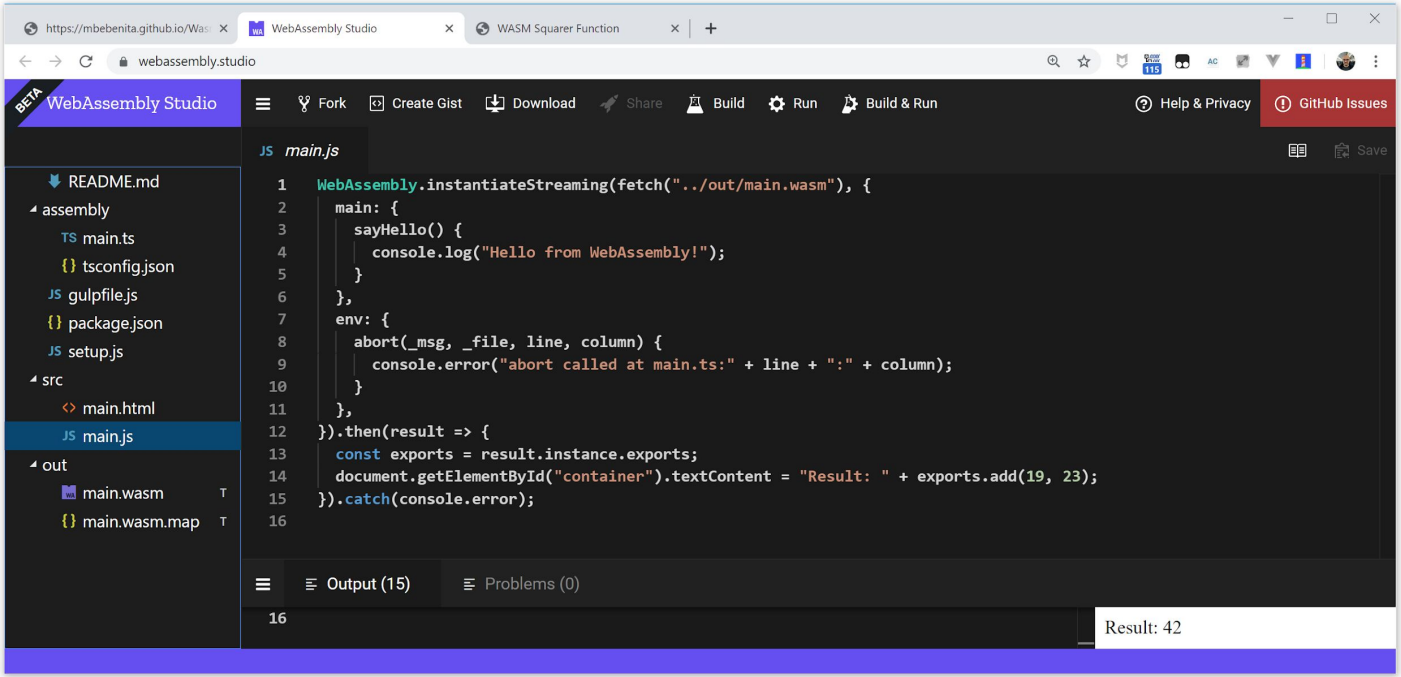
More predictable performance

Avoiding the dynamicness of JavaScrip



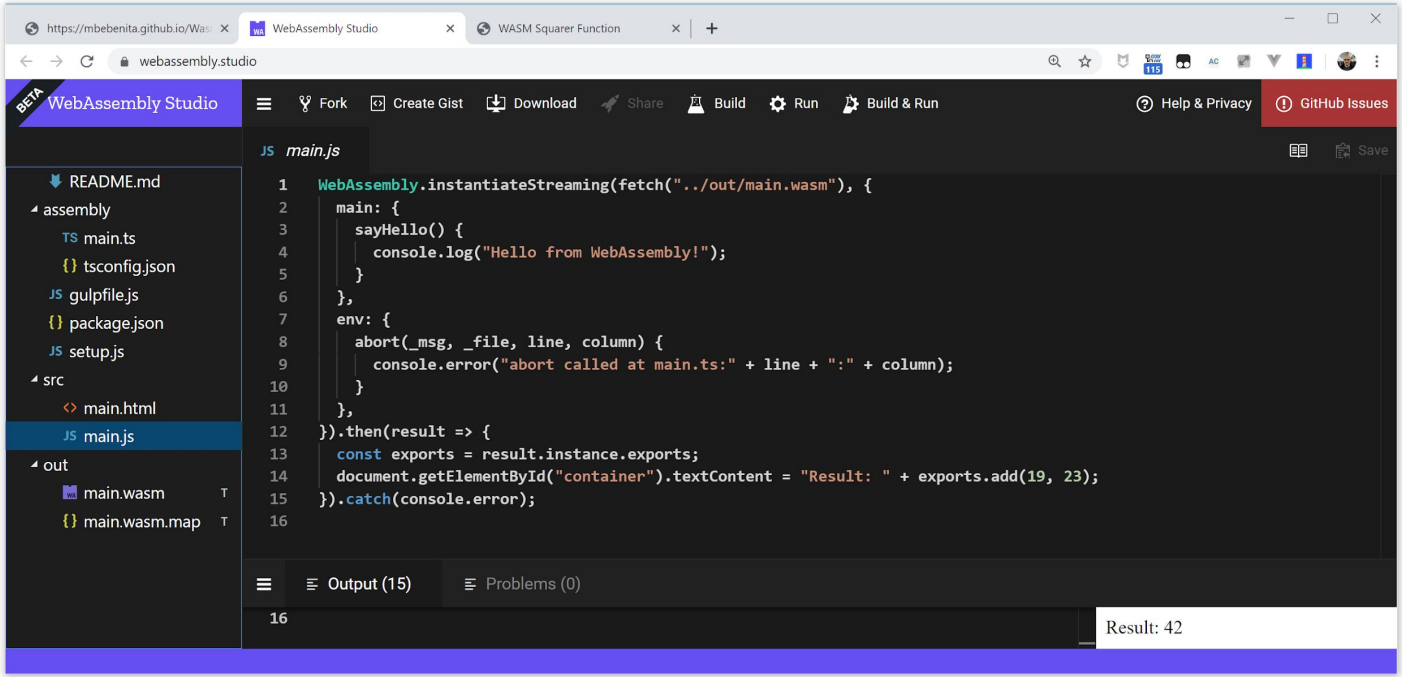
More specific integer and floating point types

Objects cannot flow in and out of WASM yet



Using a loader to write/read them to/from memory

No direct access to DOM



Glue code using exports/imports to/from JavaScript

WASM outside the browser

Not only for web developers

Run any code on any client... almost

Languages compiling to WASM

Includes WAPM

wapm install optipng

Oh, like npm for WASM!

The WebAssembly Package Manager

But I need some POSIX

WebAssembly System Interface

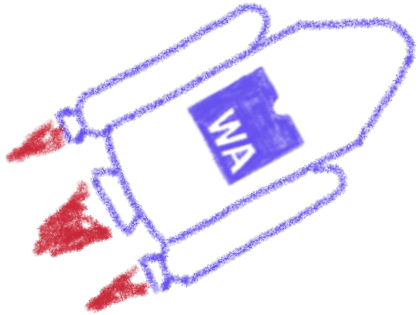
Huge potential

← **Tweet**

Solomon Hykes @solomonstre · Mar 28

If WASM+WASI existed in 2008, we wouldn't have needed to created 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! twitter.com/linclark/statu...

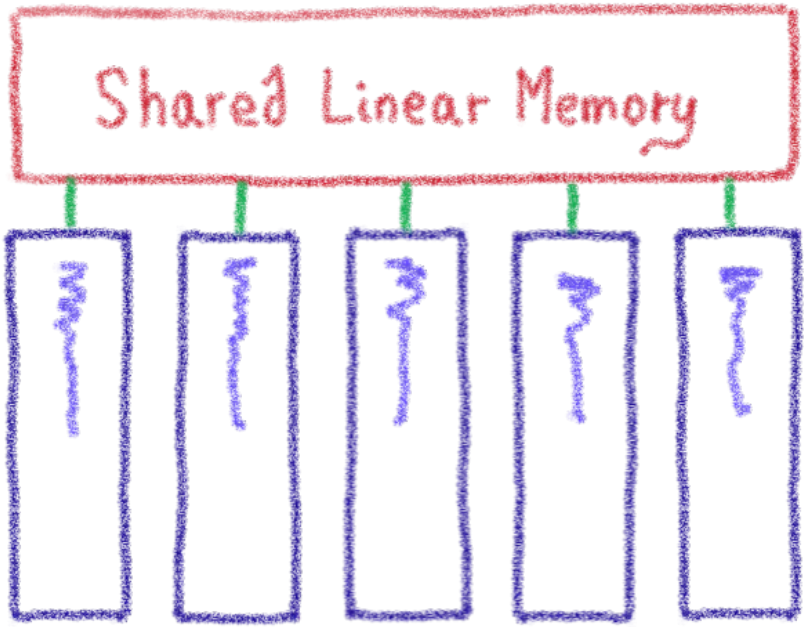
Show this thread



Future

To the infinity and beyond!

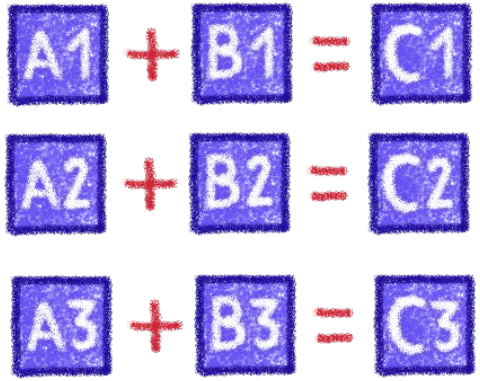
WebAssembly Threads



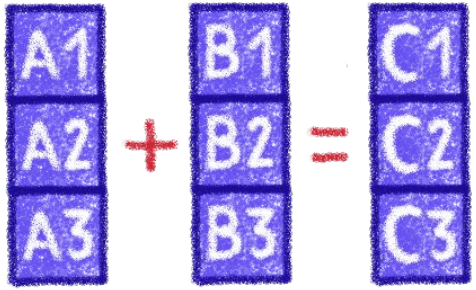
Threads on Web Workers with shared linear memory

SIMD

Multiple scalar operations



Single vectorial operation



Already available in Wasmer

Single Instruction, Multiple Data

Garbage collector

And exception handling