Italian C++ Community

www.italiancpp.org

wasm

paolo severini

Italian C++ Conference 2018
June 23, Milan

Thanks to the sponsors!

Bloomberg®

JFrog | CONAN C/C++ package manager

JET BRAINS

aresys

AIV ACCADEMIA ITALIANA VIDEOGIOCHI

KDAB

RECOGNITION ROBOTICS
The Visual Guidance Company

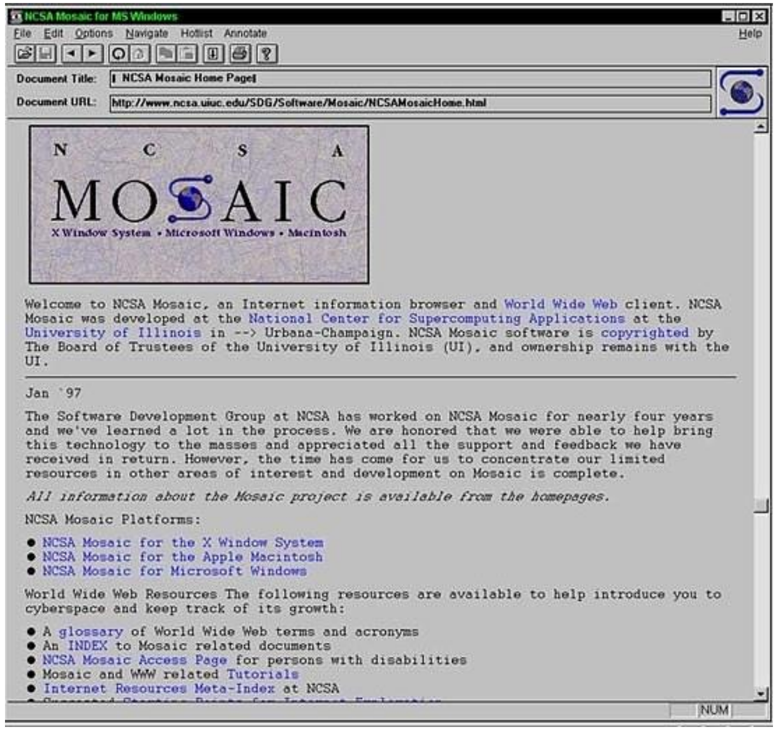
sigeo
Servizi Informativi Geografici

Italian C++ Conference 2018 – June 23, Milan #itcppcon18

A little history

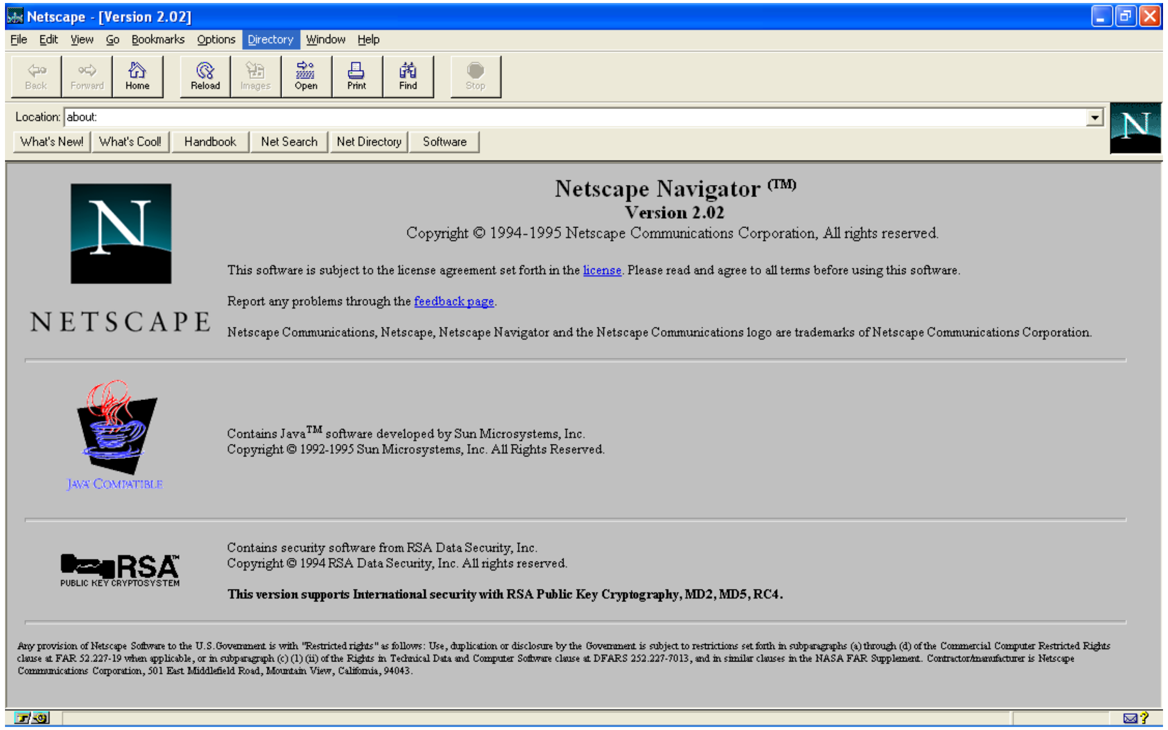
1993: NCSA Mosaic

First popular web browser



1995: JavaScript

First version of JavaScript for Netscape Navigator 2.0



Native code on the web?

The web can do more than HTML+CSS+JS

There is a lot of existing C/C++ code which we might reuse

Performance and battery life matter

Many use cases:

- Games
- VR/AR
- Graphics (editors, codecs, image recognition, …)
- Audio/Video (players, streaming, editors, …)
- Remote desktops
- VMs
- Developer tools
- Encryption
- …

Native code on the web?

Prior attempts at low-level code on the web

- Native plugins (ActiveX)

  Not safe!

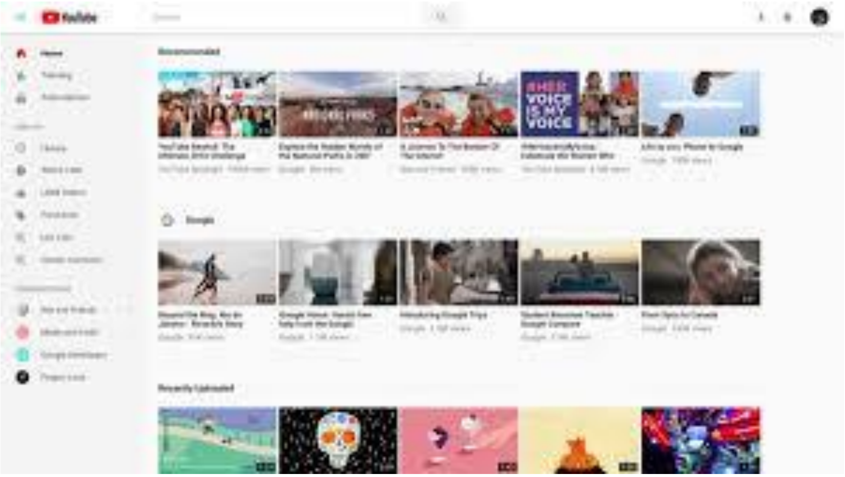
- Managed plugins (Java applets, Flash, Silverlight)

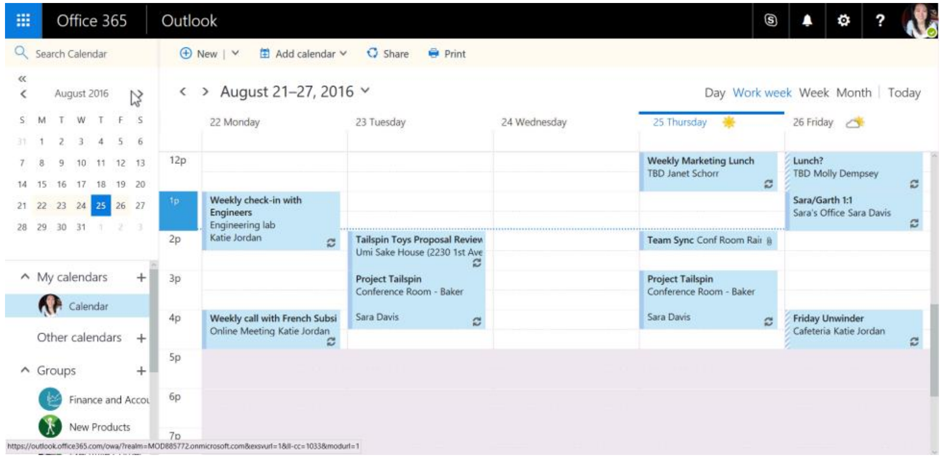
  Security/performance/portability issues

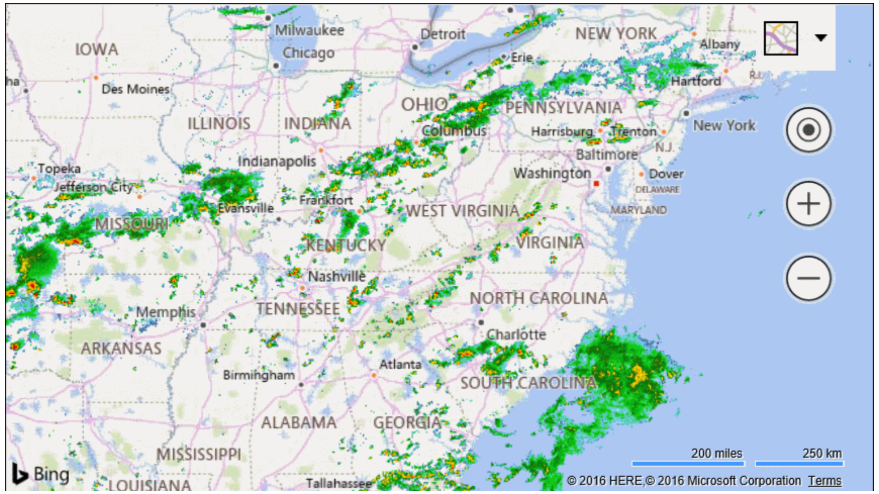
- NaCl/PNaCl
  - Chrome-only.

We need **safe**, **fast** and **portable** code for the web

Today…







Large, complex web applications

- HTML5 (WebGL, WebAudio, Video, WebRTC, WebSockets, …)
- Javascript ES6
- TypeScript
- Libraries and frameworks (React.js, Vue.js, Angular, D3, …)
- WebPack
- Very fast script engines (JIT compiler…)

The web browser is becoming the universal OS!

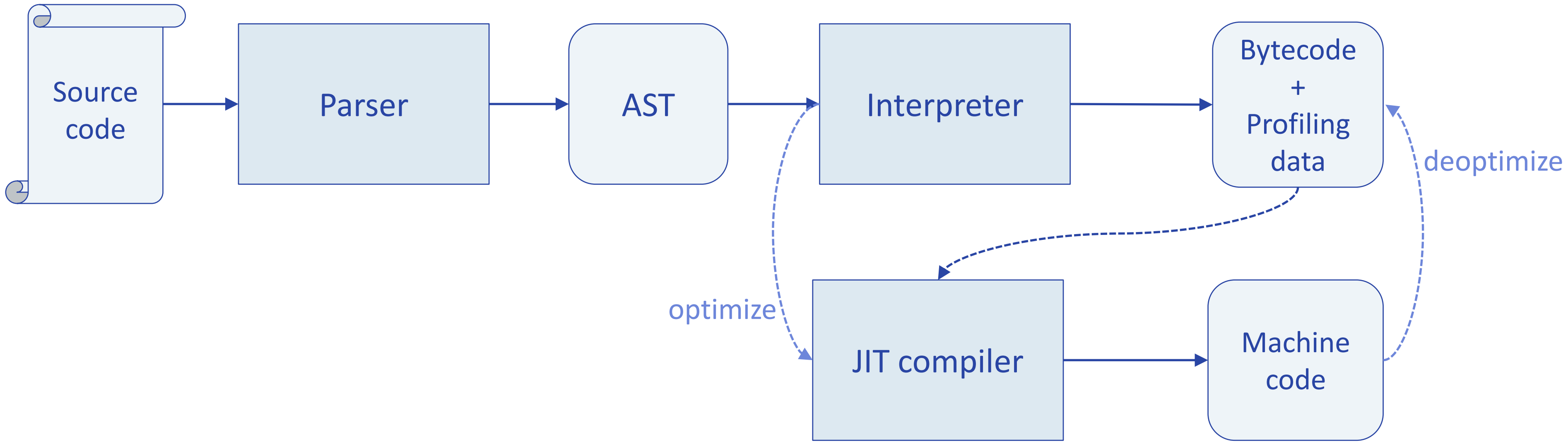
JS as a target language

JavaScript is the only language that runs in all web browsers

- We can run only JavaScript in browsers, but we can **write** in another language… if we compile (or transpile) to JavaScript!
- Many examples:
  - Google Web Toolkit
  - Google Dart
  - Script#
  - Elm
  - TypeScript
  - …

"JavaScript is the x86 of the Web" [Brendan Eich, 2009]

Inside a JavaScript engine…



asm.js

http://asmjs.org

- Statically-typed low-level subset of JavaScript
- Adds type coercion and code validation
- Disables potentially “slow” features, like garbage collection
- Highly optimizable, good as target for ahead-of-time compilation
- The JS engine may generate simpler and faster code

- Supported by all major browsers

asm.js: code example

```
void Frame::SetPixel(int x, int y, int color)
{
    _pixels[(_width * y) + x] = color;
}
```

Compiled to asm.js (with no optimizations) becomes:

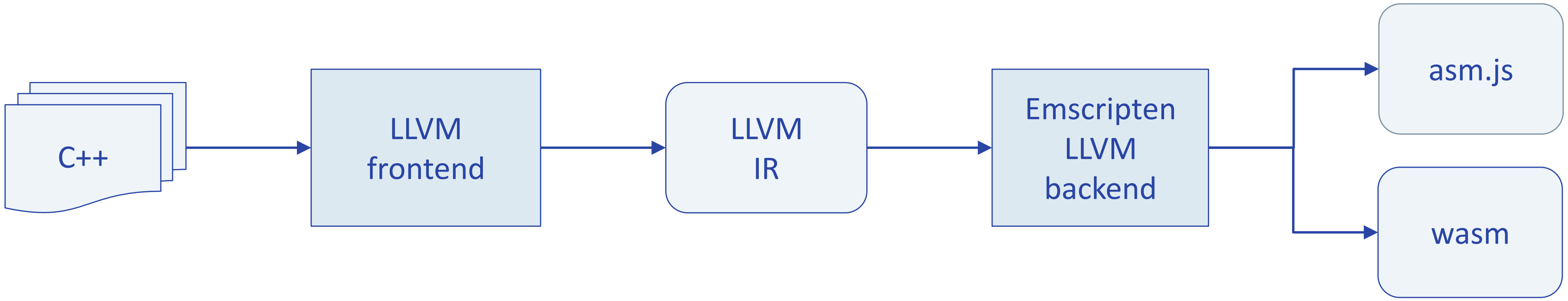
```
function __ZN6Galaga5Frame8SetPixelEiii($this, $x, $y, $color) {
    $this = $this | 0;
    $x = $x | 0;
    $y = $y | 0;
    $color = $color | 0;
    var $0 = 0, $1 = 0, $2 = 0, $3 = 0, $4 = 0, $5 = 0, label = 0, sp = 0;
    sp = STACKTOP;
    $0 = HEAP32[$this >> 2] | 0;     // $0: _width
    $1 = Math_imul($0, $y) | 0;      // $1: _width * y
    $2 = (($1)+($x)) | 0;            // $2: _width * y + x
    $3 = ((($this)) + 8 | 0);
    $4 = HEAP32[$3 >> 2] | 0;        // $4: &(_pixels)
    $5 = (($4)+($2 << 2) | 0);       // $5: &(_pixels[_width * y + x])
    HEAP32[$5 >> 2] = $color;        // $6: _pixels[_width * y + x] = color
    return;
}
```

emscripten

http://kripken.github.io/emscripten-site

LLVM-based compiler to asm.js

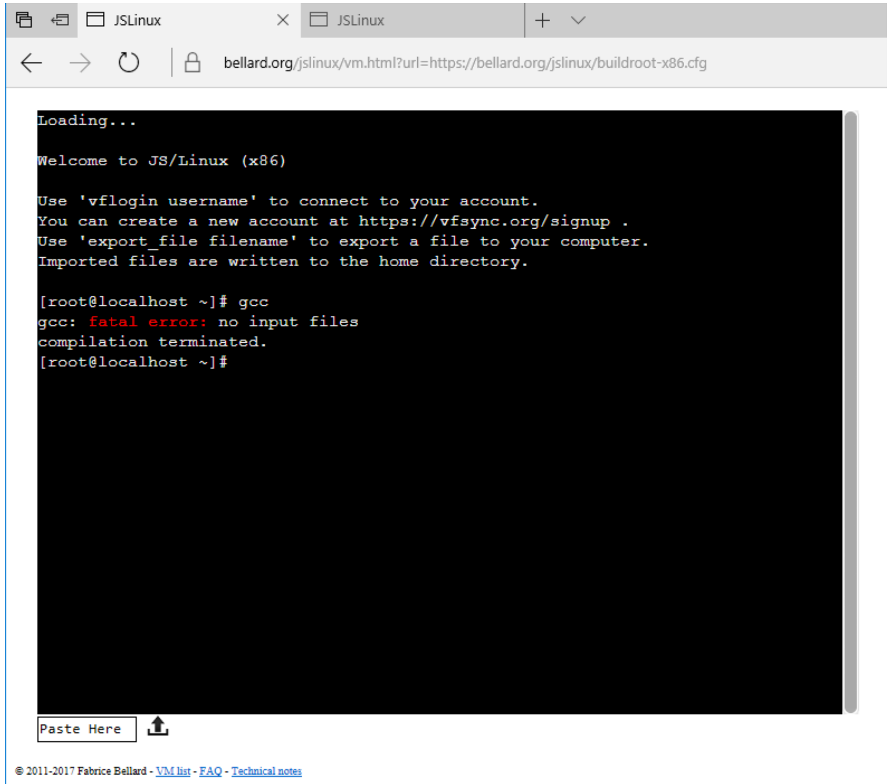
- Compiles from all LLVM-compatible languages (C/C++, Rust, …)
- Produces asm.js or WebAssembly code
- Supports many common libraries using HTML5 as backend:
  - Standard libraries (libc, libc++, SDL)
  - SDL (audio, input, …)
  - Virtual File System (libc, libcxx) -> Browser sandbox
  - Multimedia and Graphics (EGL, OpenGL) -> WebGL
  - Audio (OpenAL) -> WebAudio API
  - Pthreads, shared memory -> web workers, SharedArrayBuffer
  - …



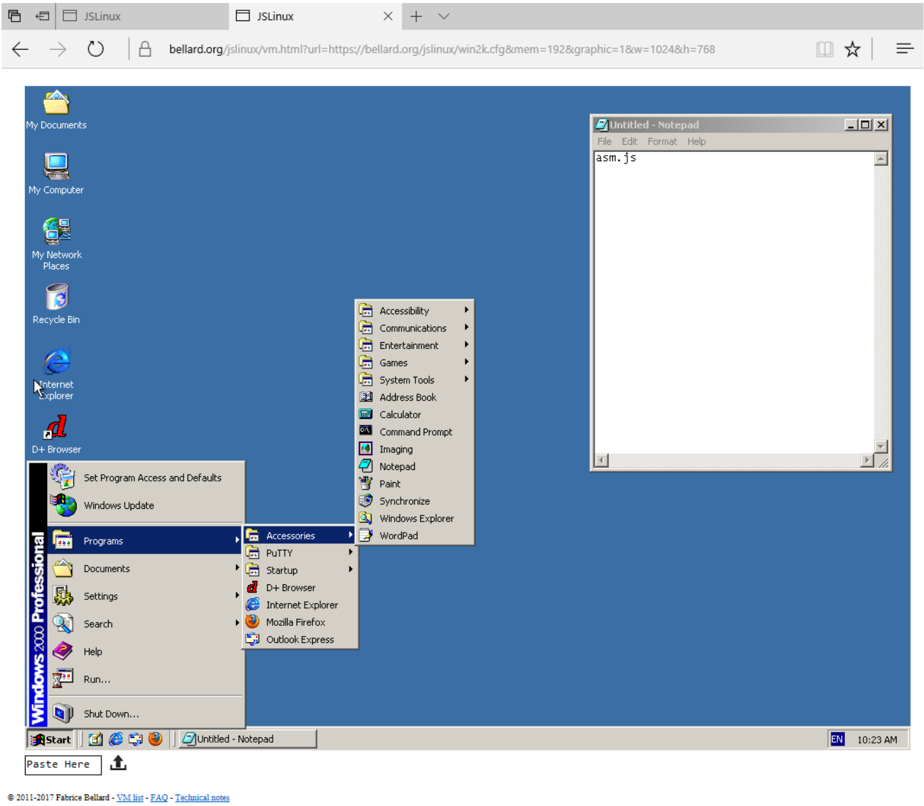
asm.js example: jsLinux

X86 emulator compiled to asm.js (Fabrice Bellard ~2011)

Linux VM



Windows 2000 VM



WebAssembly

http://webassembly.org/

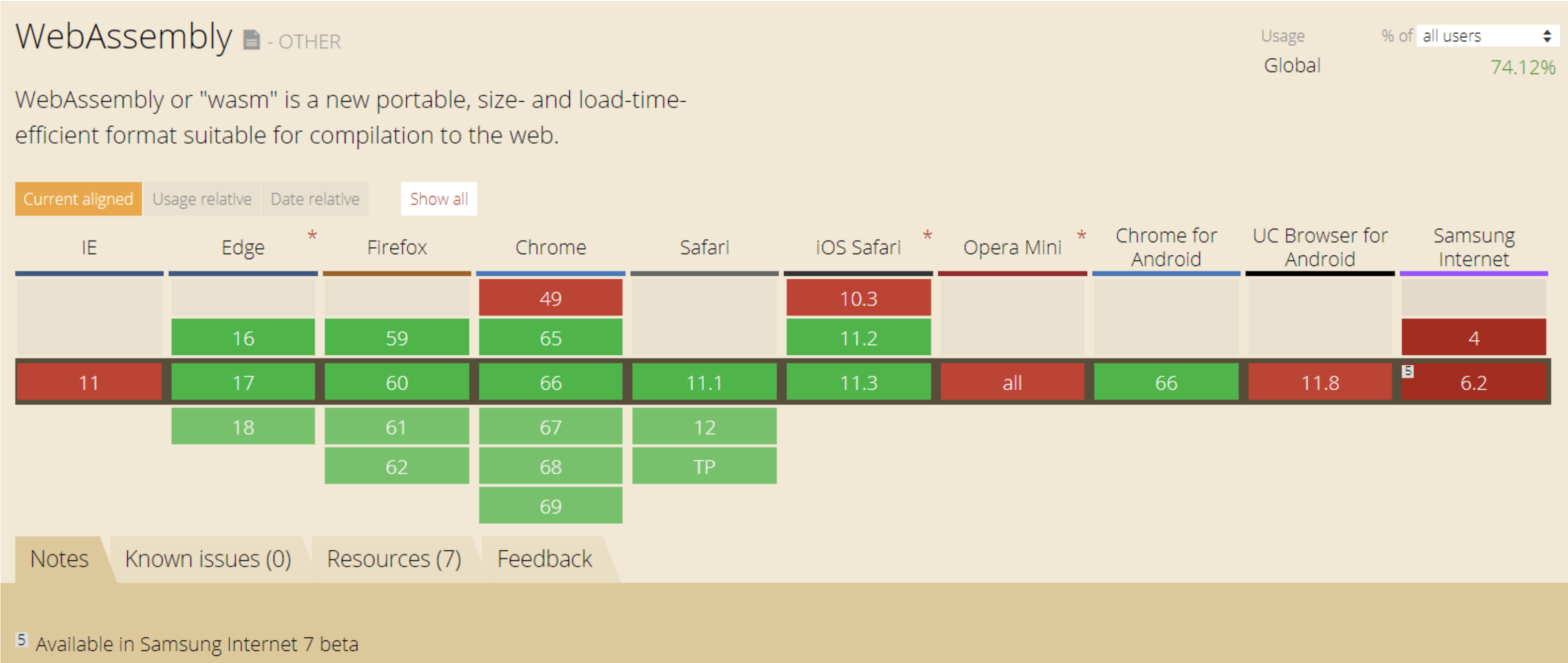
WebAssembly (*wasm*): a new binary code format suitable for compilation to the web

- Portable
- Safe
- Fast
- Size and load-time efficient

wasm: portable

Developed as a collaboration between Apple, Google, Microsoft and Mozilla

Supported by all major browsers, and by Node.js



WebAssembly - OTHER

Usage % of all users
Global 74.12%

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 *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49		10.3				
	16	59	65		11.2				4
11	17	60	66	11.1	11.3	all	66	11.8	[5] 6.2
	18	61	67	12					
		62	68	TP					
			69						

Notes | Known issues (0) | Resources (7) | Feedback

[5] Available in Samsung Internet 7 beta

Demo

Compile C/C++ code into a wasm module:

```
emcc factorial.c -Os -s SIDE_MODULE=1 -o factorial.wasm
```

<table>
<tr><th>C++</th><th>Binary</th><th>Text</th></tr>
<tr><td><pre>int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}</pre></td><td><pre>20 00
42 00
51
04 7e
42 01
05
20 00
20 00
42 01
7d
10 00
7e
0b</pre></td><td><pre>get_local 0
i64.const 0
i64.eq
if i64
    i64.const 1
else
    get_local 0
    get_local 0
    i64.const 1
    i64.sub
    call 0
    i64.mul
end</pre></td></tr>
</table>

wasm: Web embedding

For the moment WASM provides a JS API to load and execute wasm code in the browser:

1. Download wasm file with XHR; get bytes into a typed array or ArrayBuffer
2. Compile bytes and creates a WebAssembly.Module
3. Instantiate the module

Emscripten generates a js file that takes care of module instantiation.

In the future wasm modules should be supported "natively":

```
<script type="module" … />
```

wasm: Internals

Module: like a DLL

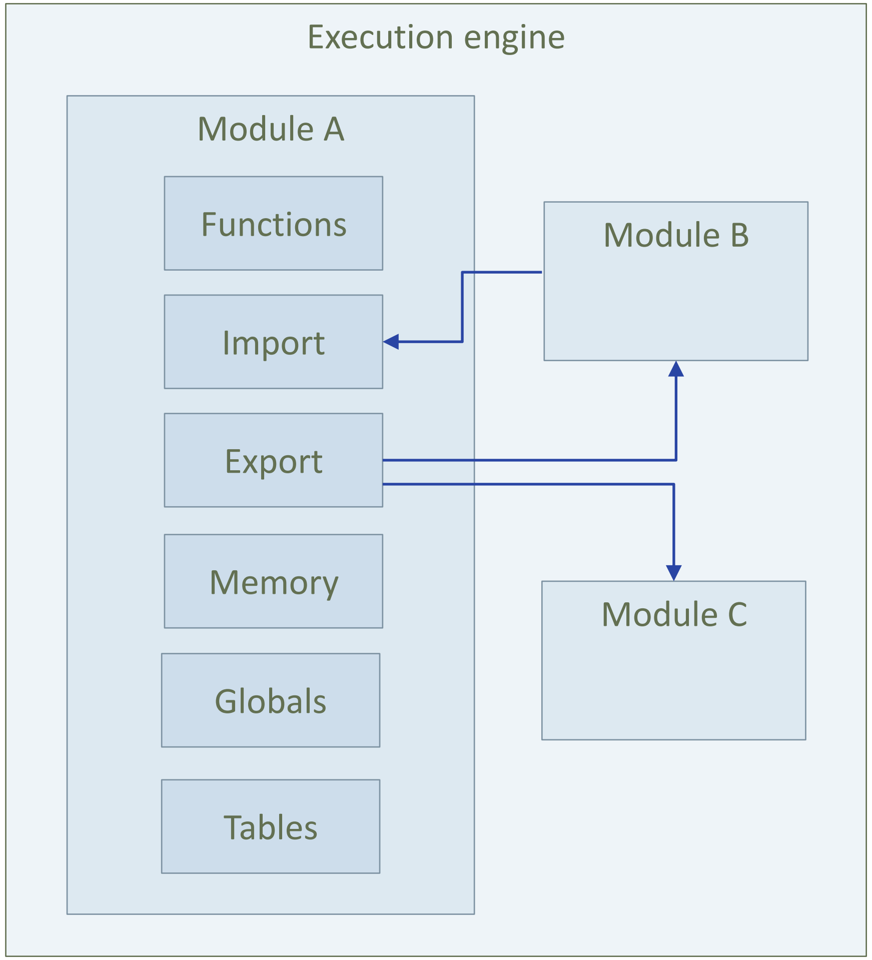
- Contains code, data, import/export tables

Functions: sequence of instructions

- Computation is based on a stack machine
- Control flow is expressed by structured constructs (blocks, ifs, loops)

Memory: like a heap

- Separated from code space and execution stack
- Can be shared across modules



wasm: Safety

WebAssembly is designed to be very safe

- In a browser, wasm code runs in the same sandbox as JavaScript
- Before being executed a wasm module can be **validated** to ensure that it is type-safe and memory-safe
- Code in a wasm module cannot have any undefined behavior, or access memory areas it does not own

wasm: Speed

WebAssembly code is designed to run at almost-native speed

- Binaries are AOT-compiled, skipping the parsing phase
- Simple stack machine (and no dynamic typing), makes it possible to generate "better" native code
- The validation algorithm for a wasm module is very simple.

Benchmarks:

- Wasm is faster than asm.js (no parsing)
- Wasm can run almost as fast as native code (usually slow-down factor: 1.1 to 2.0)

Wasm is also space-efficient

- Wasm modules are smaller than the equivalent native modules (~85%), and much smaller than minified js (~63%)

wasm: Debugging

Debugging wasm is complicated

- Still very limited debugging support (no wasm source maps, browsers show wast code: it’s like debugging assembly without PDBs ☹)
- Breakpoints can be set in wast code
- Asm.js supports source maps, some browsers show the original C++ code if available with support for breakpoints
- Better to compile and test natively with LLVM before compiling with Emscripten to run on the web

wasm: What is it for?

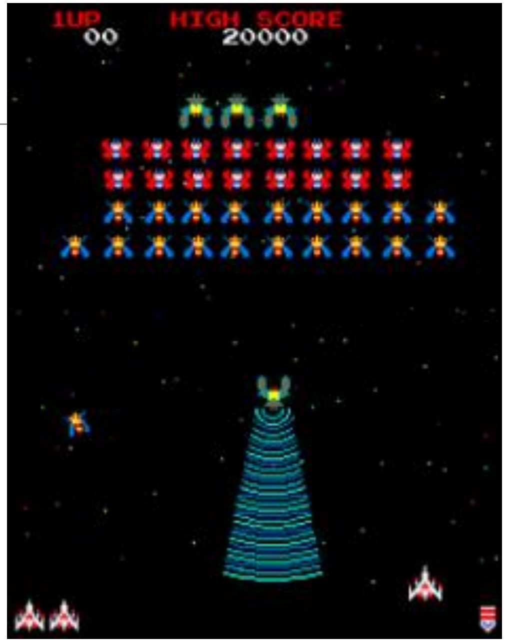
The goal of WebAssembly **is not** to replace JavaScript as the main programming language for the web, but to complement it.

Different programming models:

- Main codebase in WebAssembly, in a single-page web application (Example: games)
- Reusable WASM components embedded in a large JS/HTML5/CSS app (Example: helper libraries used to offload compute-oriented tasks or to reuse legacy code)

WebAssembly is not ideal to directly access/modify the DOM but in the future there could be new wasm-based UI frameworks.

Example: Galaga



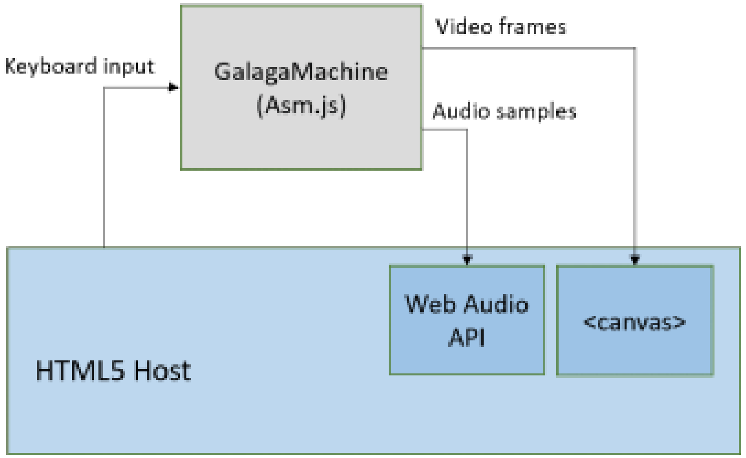
C++ emulator for an old arcade machine

- Executes the code from original ROMs
- Emulate z80 microprocessors

Compile the game engine to wasm

Host the game engine with HTML/JS

- Use the engine as a black box: feeds input controls, receives screen frames and audio samples
- Use <canvas> API to display the graphics
- Use WebAudio API to render synthesized sounds



Example: Galaga

C++: declare the classes and functions exported by the module

JS: instantiate the wasm module; exported classes and functions are visible to JS

C++

```cpp
#include <emscripten/bind.h>
using namespace emscripten;

#include "galagamachine.h"

// Binding code
EMSCRIPTEN_BINDINGS(galaga_machine)
{
    class_<GalagaMachine>("GalagaMachine")
        .constructor<>()
        .function("Run", &GalagaMachine::Run)
        .function("set_InsertCoin", &GalagaMachine::set_InsertCoin)
        .function("set_Start1Player", &GalagaMachine::set_Start1Player)
        .function("set_Start2Player", &GalagaMachine::set_Start2Player)
        .function("set_MoveLeft", &GalagaMachine::set_MoveLeft)
        .function("set_MoveRight", &GalagaMachine::set_MoveRight)
        .function("set_Button1", &GalagaMachine::set_Button1);
}
```

JS

```js
galaga = new Module.GalagaMachine();

...

galaga.Run(interval,
        function (videoFrame, audioBuffer) {
            // render videoFrame
            ...

            // play audioBuffer
            ...
        }
    );
```

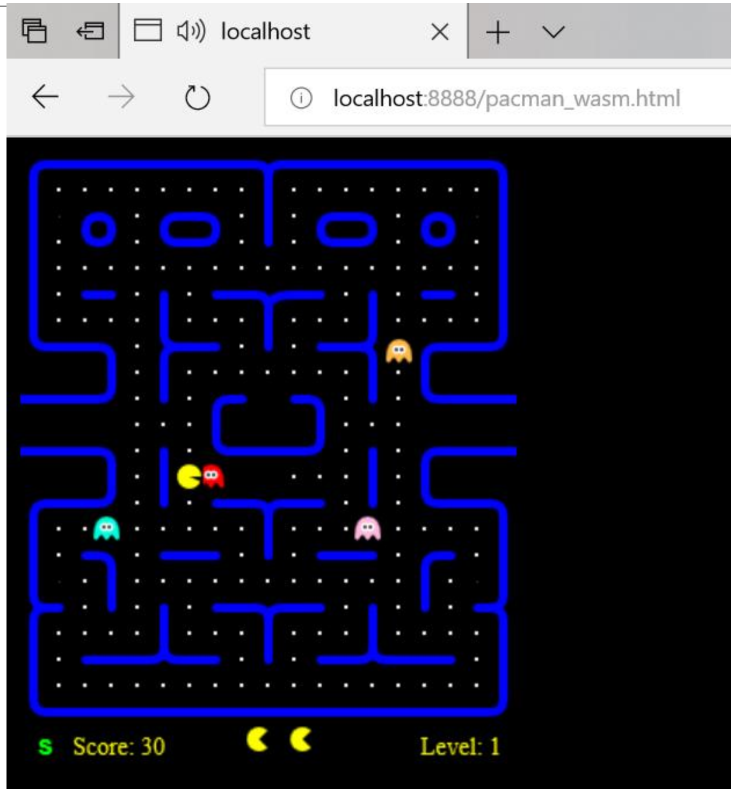
Example: Pac-Man

It is possible to interact with the web page and access the DOM from C++.

Example:

```
class CanvasContext
{
public:
    …
    void lineTo(PointF point) {
        _canvasContext.call<void>("lineTo", point.x, point.y);
    }

private:
    emscripten::val _canvasContext;
};
```



wasm: What’s next?

Still in its infancy. MVP ready, but with limitations:

- Missing features:
  - Multithreading and atomics
  - Garbage collection
  - Exception handling
  - Host bindings (better interoperability
- Limited debugging support

Also, it’s difficult to compile native code that relies on external or system libraries (need to port all libraries)

- However, Emscripten already supports many C++ libraries and the STL.
- Everything that is not platform related compiles and just works.

Blazor

https://github.com/aspnet/blazor

A .NET web-framework based on C#/Razor, which runs in the browser via WebAssembly

Is it possible to compile from Java, C# to wasm? Not easily (No support for GC yet)

But mono has been compiled to wasm

Blazor can run in two modes:

1. AOT: static compilation (creates a big wasm module with the whole mono runtime)
2. Interpreted mode (faster dev cycle)

counter.cshtml

```
@page "/counter"
<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button @onclick(IncrementCount)>Click
me</button>

@functions {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

main.cshtml

```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.

<Counter />
```

Resources

Emscripten

- http://kripken.github.io/emscripten-site
- https://github.com/kripken/emscripten

WebAssembly

- https://webassembly.org
- https://github.com/WebAssembly
- Spec v1.0: https://webassembly.github.io/spec/core/index.html
- Paper: Bringing the Web up to Speed with WebAssembly

Questions?