WebAssembly for the rest of us

WEBASSEMBLY

Adolfo Ochagavía

TOC

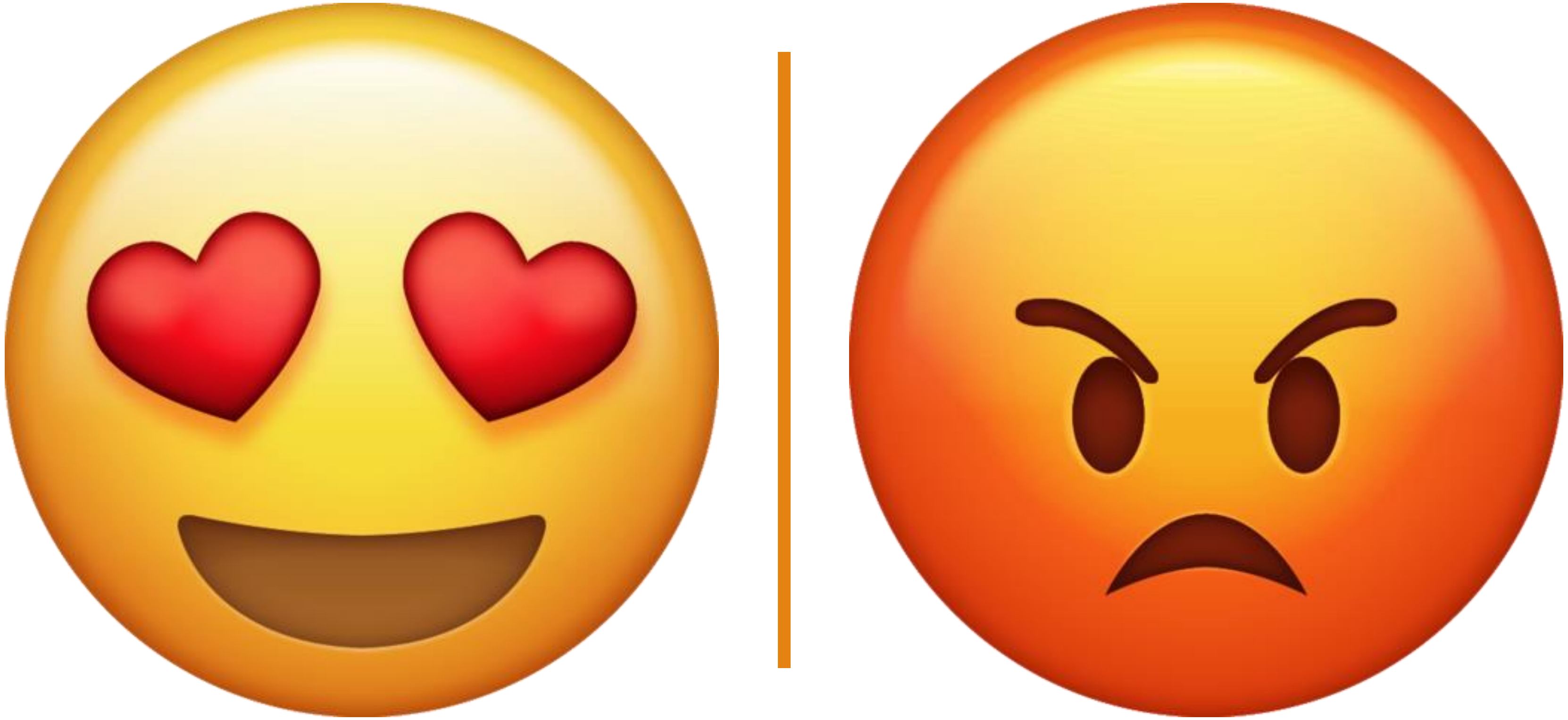
Programming language choice on the Web

What WebAssembly has to offer

Brief tour through a toy project

The Good, the Bad and the Ugly

Conclusions



JavaScript

Current solution: transpiled languages

Transpile: *transform a program in a given language into an equivalent program in JavaScript*

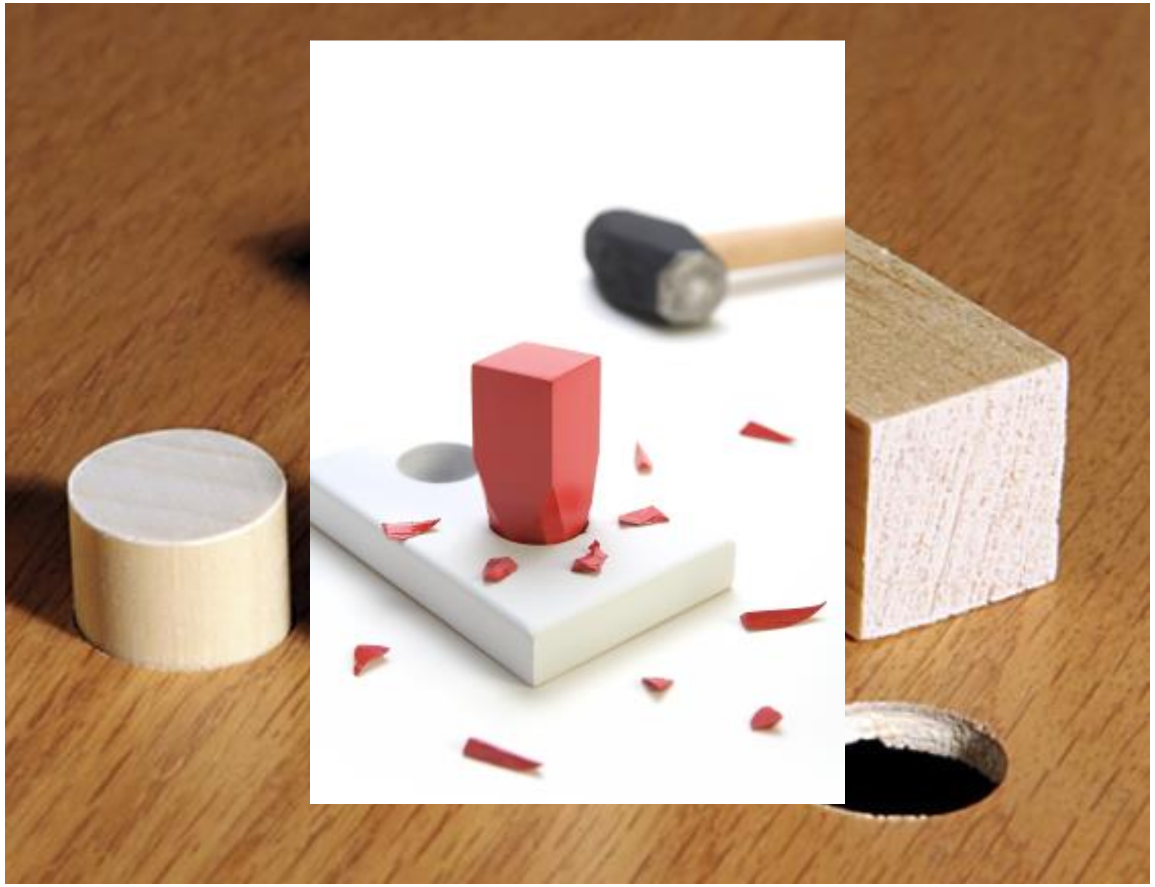
Limits of transpilation – ClojureScript

Some differences between Clojure and ClojureScript:

- Equality on numbers works like JavaScript, not Clojure
- ClojureScript does not have character literals. Characters are single-character strings, as in JavaScript
- ClojureScript regular expression support is that of JavaScript (example edge case)

JavaScript-imposed constraints:

- Number operators
- Regex engine



What this means for existing languages

Mismatch:

- Language semantics
- Language APIs

You may get to choose the color of the figure, but it must be a cylinder!

Transpilation only takes you so far…

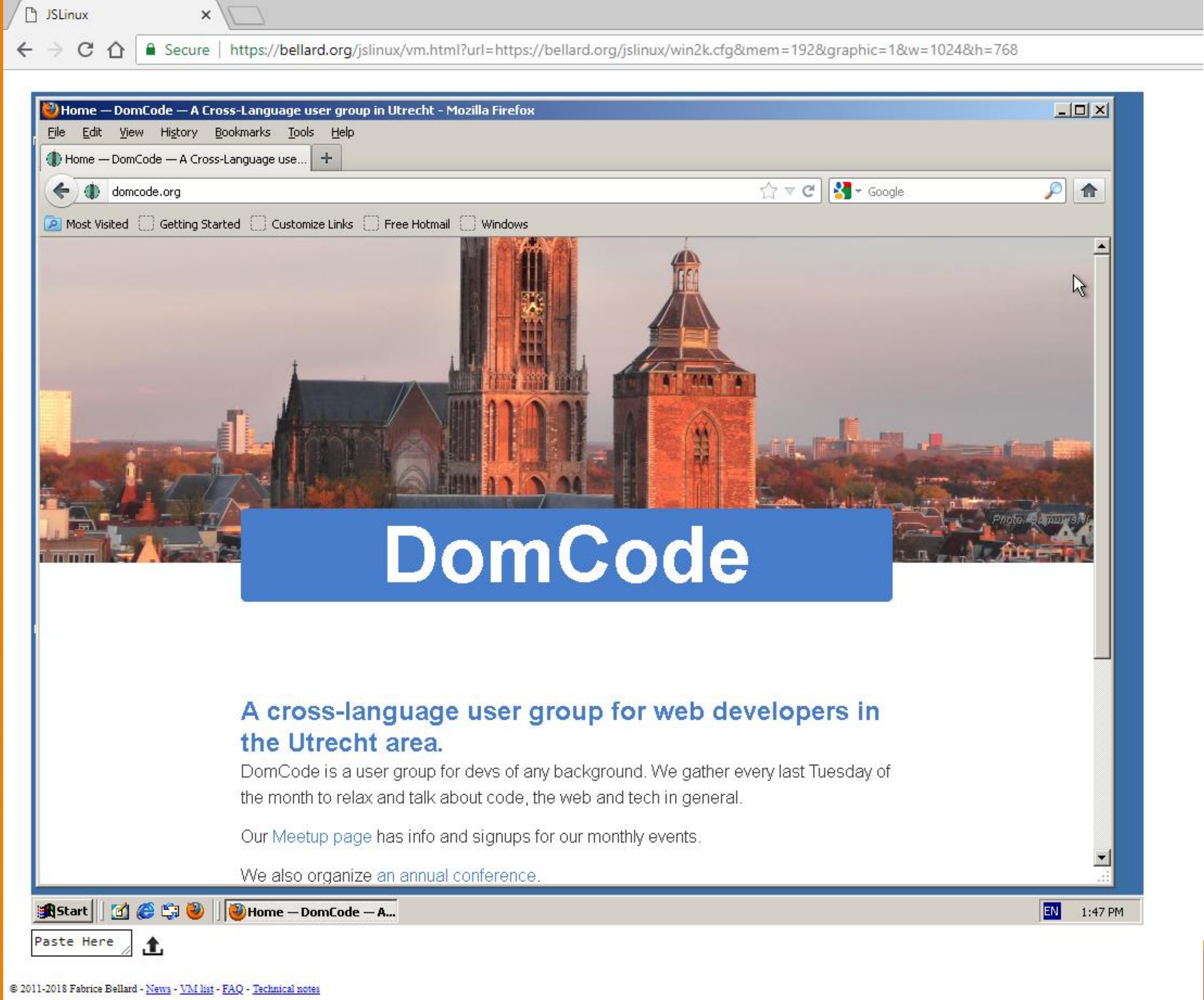
Enter WebAssembly

Assembly language for the web featuring:

- Excellent performance
- Low-level, language agnostic primitives
- JavaScript-like safety

It can have quite unexpected applications…

The future



https://bellard.org/jslinux/vm.html?url=https://bellard.org/jslinux/win2k.cfg&mem=192&graphic=1&w=1024&h=768

Home — DomCode — A Cross-Language user group in Utrecht - Mozilla Firefox

domcode.org

DomCode

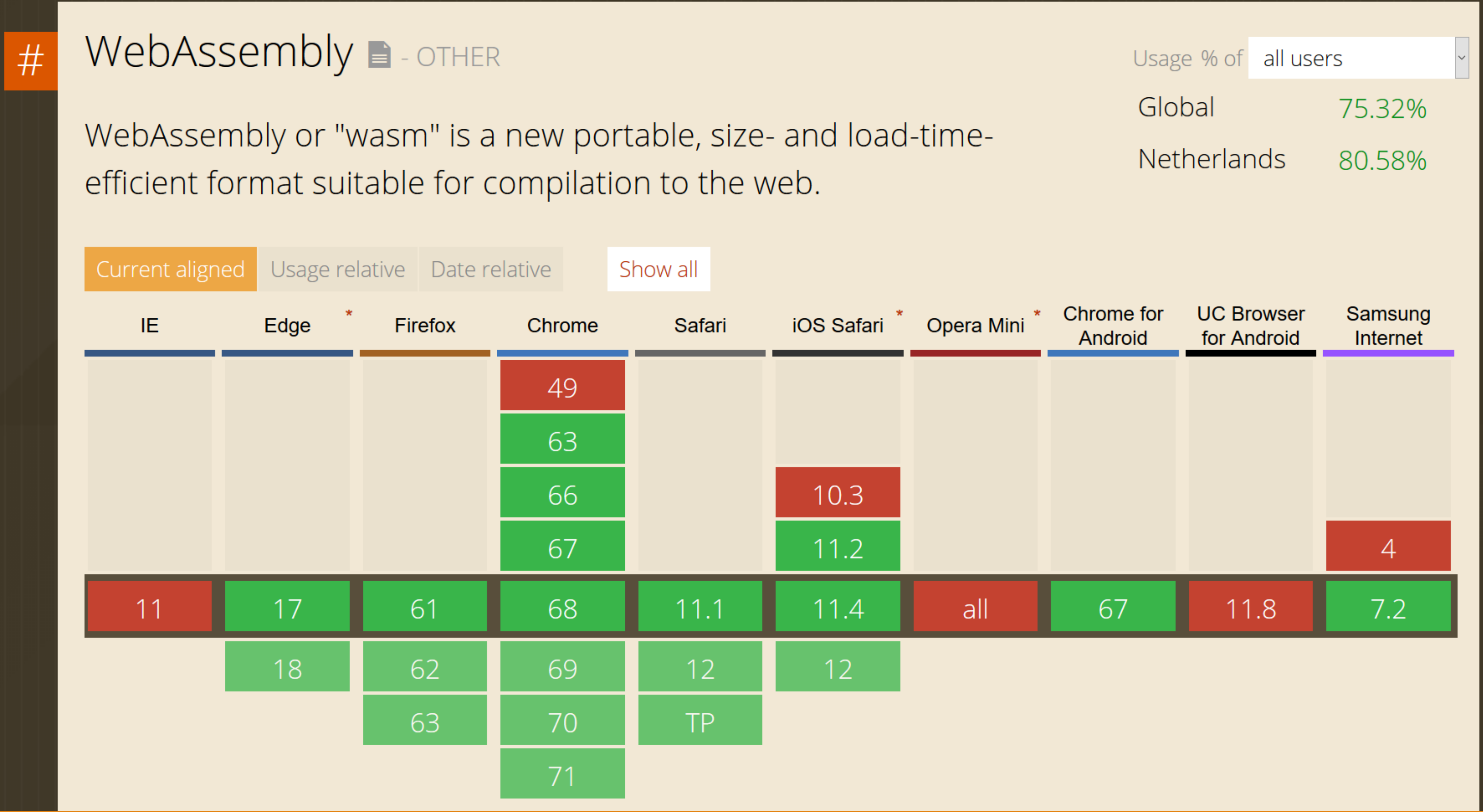
A cross-language user group for web developers in the Utrecht area.

DomCode is a user group for devs of any background. We gather every last Tuesday of the month to relax and talk about code, the web and tech in general.

Our Meetup page has info and signups for our monthly events.

We also organize an annual conference.

© 2011-2018 Fabrice Bellard - News - VM list - FAQ - Technical notes



WebAssembly - OTHER

Usage % of all users

Global	75.32%
Netherlands	80.58%

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

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			🟥 49						
			🟩 63						
			🟩 66		🟥 10.3				
			🟩 67		🟩 11.2				🟥 4
🟥 11	🟩 17	🟩 61	🟩 68	🟩 11.1	🟩 11.4	🟥 all	🟩 67	🟥 11.8	🟩 7.2
	🟩 18	🟩 62	🟩 69	🟩 12	🟩 12				
		🟩 63	🟩 70	🟩 TP					
			🟩 71						

WebAssembly means flexibility

Low level = no assumptions about your programming language!

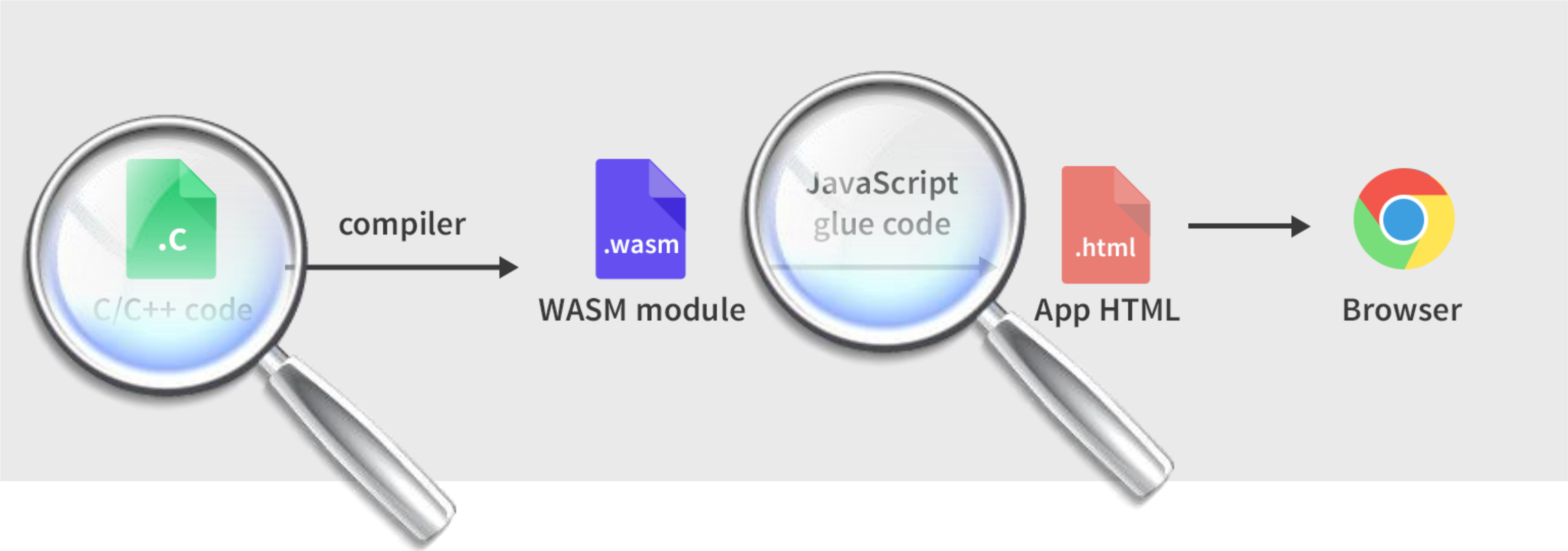
Regarding ClojureScript…

- Native integer support
- Fast enough to use your own regex engine

Decomposing WebAssembly

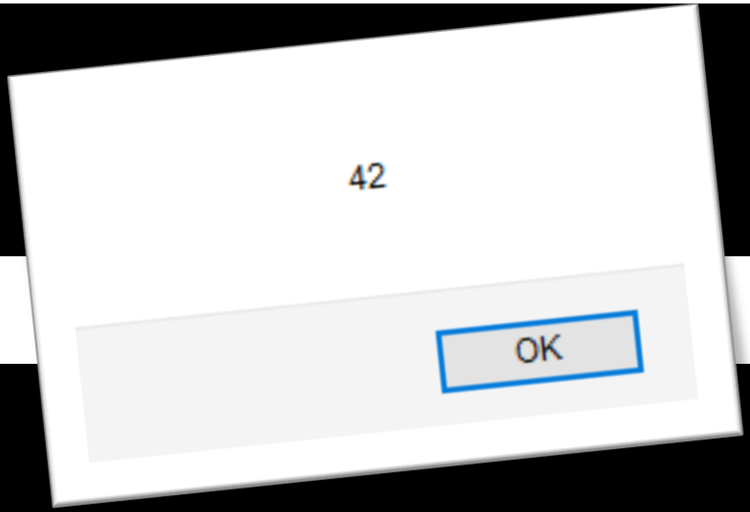
WebAssembly is:

- A binary instruction format
- That can be executed by a virtual machine
- Meant as a compilation target for high-level languages


.c
C/C++ code
compiler
.wasm
WASM module
JavaScript
glue code
.html
App HTML
Browser

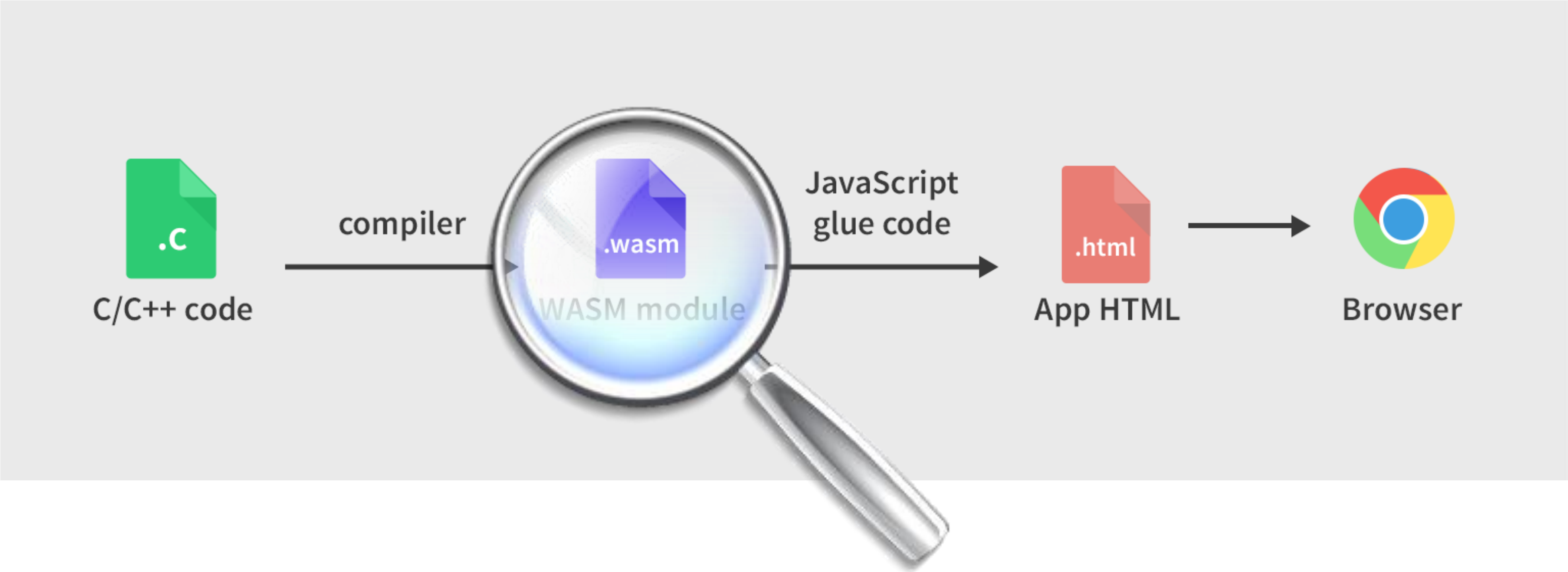
A minimal example

```c
int32_t the_answer() {
  return 42;
}
```



42

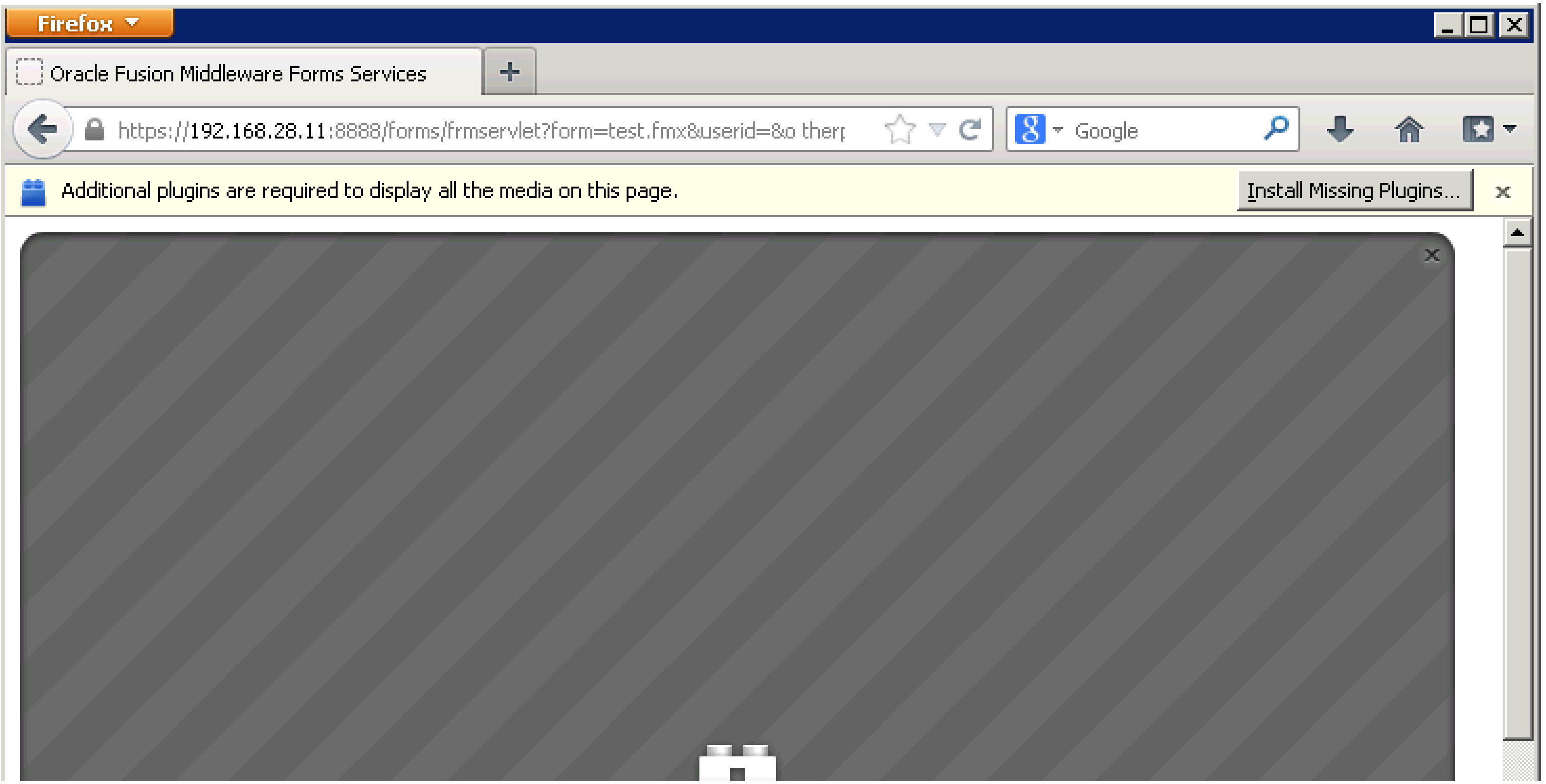
```html
<script>
fetch('main.wasm’)
  .then(response => response.arrayBuffer())
  .then(bytes => WebAssembly.instantiate(bytes))
  .then(mod => alert(mod.instance.exports.the_answer()));
</script>
```


.c
C/C++ code
compiler
.wasm
WASM module
JavaScript
glue code
.html
App HTML
Browser

Instruction format

```
(module
  (func $the_answer (result i32)
    i32.const 42)
  (export "the_answer" (func $the_answer))
)
```

```
00 61 73 6D 01 00 00 00 01 05 01 60 00 01 7F 03
02 01 00 07 0E 01 0A 74 68 65 5F 61 6E 73 77 65
72 00 00 0A 06 01 04 00 41 2A 0B 00 19 04 6E 61
6D 65 01 0D 01 00 0A 74 68 65 5F 61 6E 73 77 65
72 02 03 01 00 00
```



Firefox

Oracle Fusion Middleware Forms Services

https://192.168.28.11:8888/forms/frmservlet?form=test.fmx&userid=&o therp

Additional plugins are required to display all the media on this page.

Sounds familiar?

Why not existing VMs?

Unique requirements:

- Excellent performance
- Low-level, language agnostic primitives
- JavaScript-like safety

Furthermore:

- Simplicity
- Decision making

Using wasm in the browser

Right now, wasm modules must be initialized in JavaScript code:

```
<script>
fetch('main.wasm’)
  .then(response => response.arrayBuffer())
  .then(bytes => WebAssembly.instantiate(bytes))
  .then(mod => alert(mod.instance.exports.the_answer()));
</script>
```

Using wasm from JavaScript

Step 1: write a library in your favorite language

Step 2: use the library from JavaScript

Supported ways to interact with your library:

- Function calls
- Callbacks
- Global variables
- Pointers to wasm data

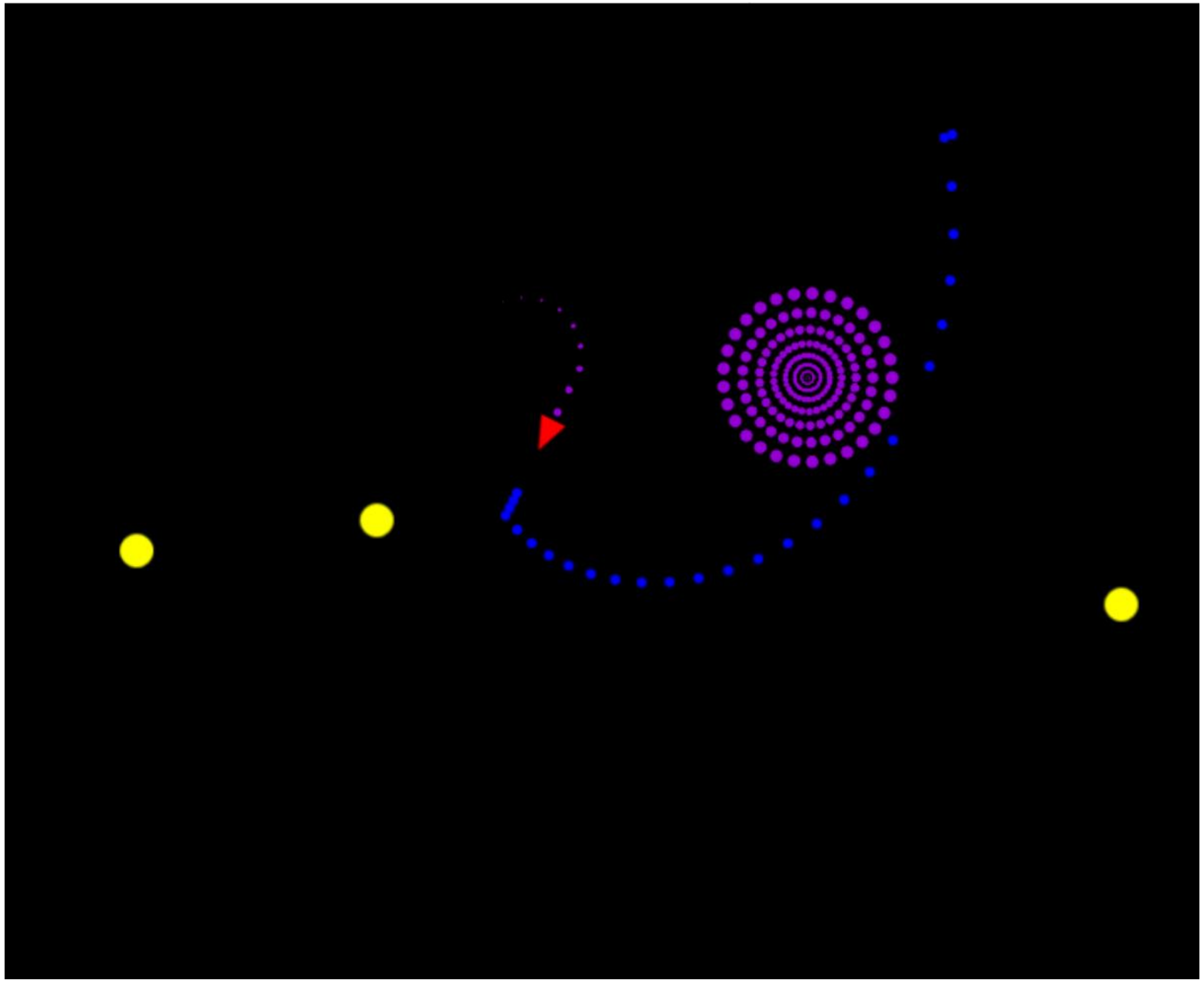
Limitations in the wasm/JS boundary

WebAssembly:

- Supports only numeric types (integer and float)
- Globals, parameters and return types can only be of integers or floats
- Doesn’t know what a JavaScript object is, not even strings

Escape hatch: pointers to WebAssembly memory

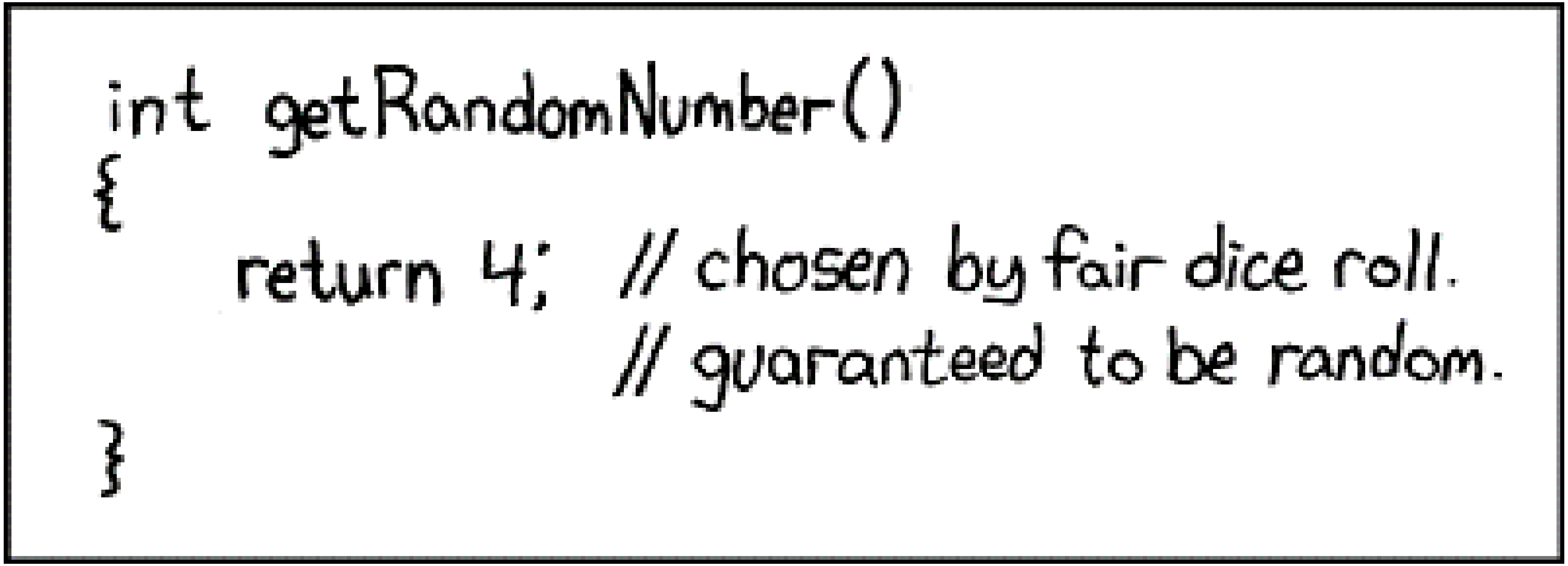
A low level tour of WebAssembly



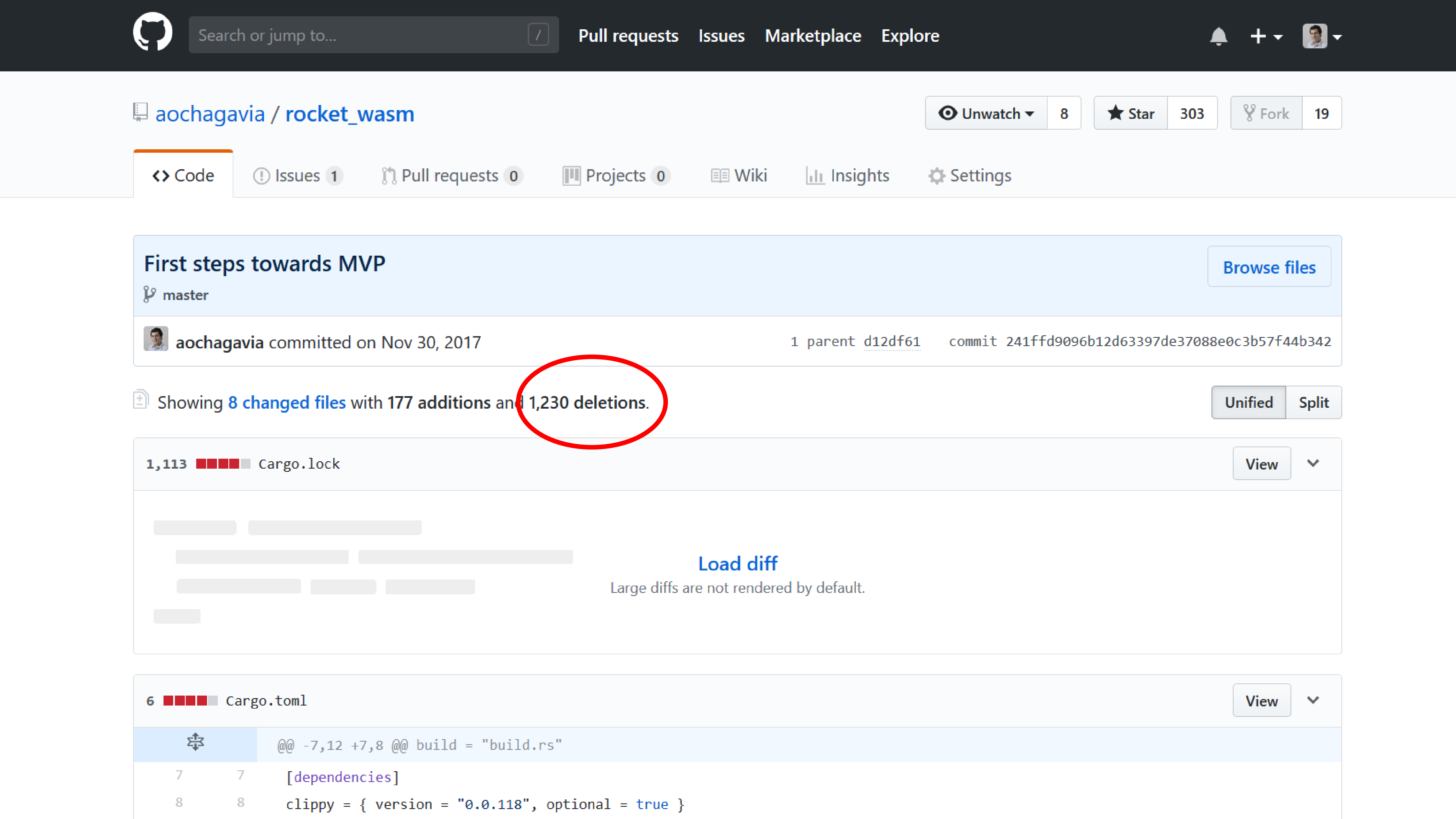
Rocket

Challenges

Solving the problem of random numbers



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```



aochagavia / rocket_wasm

First steps towards MVP

master

aochagavia committed on Nov 30, 2017

1 parent d12df61 commit 241ffd9096b12d63397de37088e0c3b57f44b342

Showing 8 changed files with 177 additions and 1,230 deletions.

1,113 ■■■■□ Cargo.lock

Load diff

Large diffs are not rendered by default.

6 ■■■■□ Cargo.toml

```
@@ -7,12 +7,8 @@ build = "build.rs"
7	7	[dependencies]
8	8	clippy = { version = "0.0.118", optional = true }
```

And then?

We still need to somehow:

- Draw to the screen
- Fire events upon user input

Solution:

- Draw to the screen using JavaScript callbacks
- Trigger events from the JavaScript side

Triggering input events

```
function processKey(key, b) {
  switch (key) {
    case "ArrowLeft" : module.toggle_turn_left(b);  break;
    case "ArrowRight": module.toggle_turn_right(b); break;
    case "ArrowUp"   : module.toggle_boost(b);      break;
    case " "         : module.toggle_shoot(b);      break;
  }
}
document.addEventListener('keydown’,
  e => processKey(e.key, true));
document.addEventListener('keyup’,
  e => processKey(e.key, false));
```

Drawing to the screen

```
// JavaScript
let ctx = canvas.getContext("2d");
function draw_enemy(x, y) {
  ctx.drawImage(res.enemy, x - 10, y - 10);
}
```

```
// Rust
for enemy in &world.enemies {
  draw_enemy(enemy.x(), enemy.y());
}
```

Demo time!

https://thread-safe.nl/rocket

Summary (in LOC)

Original lines of Rust code:

- 23 files
- 788 lines

Final lines of Rust code:

- 18 files
- 628 lines (160 lines less than before)

Final lines of JavaScript code: 134

The good

We are running Rust code on the browser!

…

The bad

Cumbersome to use

Requires understanding of low level WebAssembly concepts

The ugly

It makes your eyes bleed…

```
#[no_mangle]
pub extern "C" fn toggle_turn_left(b: c_int) {
  let game = &mut GAME.lock().unwrap();
  game.actions.rotate_left = int_to_bool(b);
}
```

6.3.2005

Improvements to WebAssembly

Access to web APIs

Garbage collection support

Performance

More available types

Loading modules without JavaScript intermediation

Better tooling

Glue code should be automatically generated

Libraries should abstract away the low level details

Check out:

- stdweb if you are want to run Rust code on the browser
- Blazor to develop single page applications entirely in C#
- Unity and UE4 to develop games that run on the browser

Is it worth it right now?

Already used in production!

PSPDFKit: web pdf editor (C++)

AutoCAD

Firefox: source map parsing and generation (Rust)

Conclusions

Will it replace JavaScript?

Challenges: improvements to wasm itself and tooling

Main use cases right now:

- Performance
- Code reuse

Further info/reading

Official WASM website

The future of WebAssembly – A look at upcoming features and proposals

WebAssembly studio (online IDE)

WASM weekly: a weekly newsletter

Talks:

- The Birth & Death of JavaScript, by Gary Bernhardt
- The WebAssembly Revolution Has Begun, by Jay Phelps

Interesting projects

- Emulators (NES, Windows 2000, …)
- Go REPL
- Qt framework
- Cryptocurrency miner

Benchmarks

Raytracing

Mandelbrot

Realtime pitch detection

PSPDFKit: real world benchmark by a company using wasm in production

Measured improvements to real codebases:

- Firefox source maps
- Secp256k1 algorithm

github.com/aochagavia/domcode-wasm

WA

WEBASSEMBLY