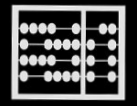
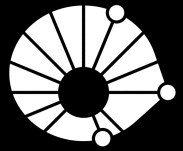




MC921 | Projeto e Construção de Compiladores | 2022.2

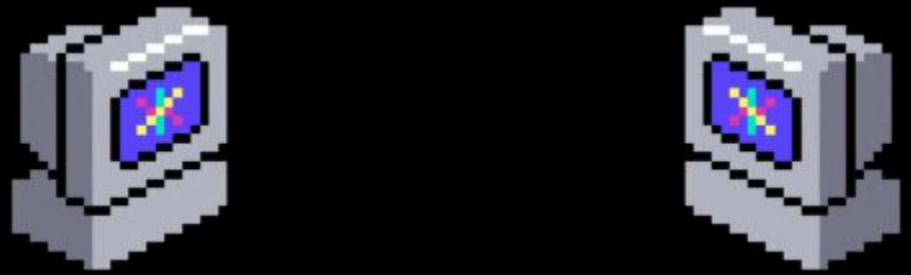
WebAssembly

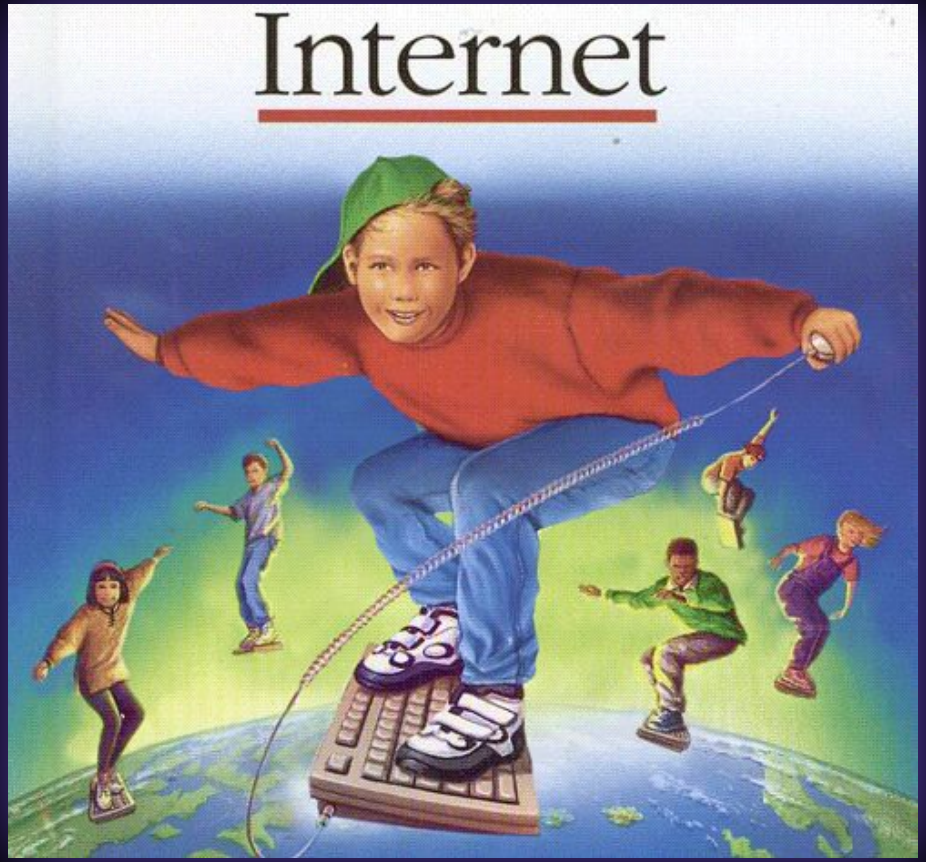
Gabriel Kenji, RA 216295
@paulopacitti, RA 185447

Summary.

- How the Web works -> *"Magic"*
- Problems with the Web -> *I got 99 problems and 99 are caused by Javascript*
- WASM:
  - History
  - Goals
  - Stack Machine
  - Instructions
- Examples
- Experiments
- Demo
- Future

How the Web works?



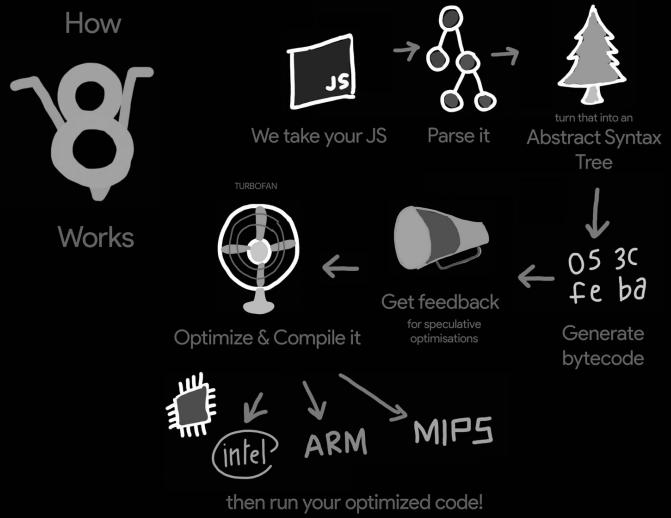

Internet


HTML CSS JS
EVERYWHERE

How the Web works?

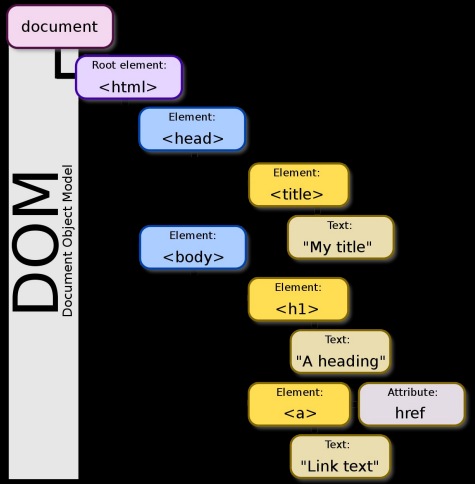
- **Javascript Engine**: compiles ECMAScript directly to native machine code using **just-in-time compilation** before executing it.

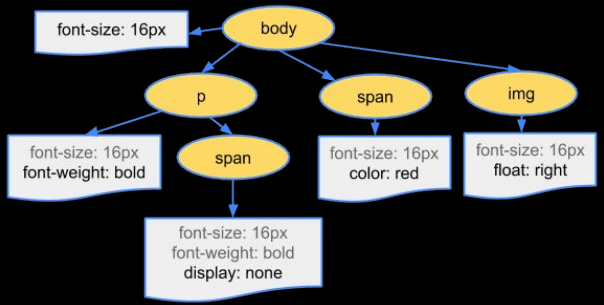
```
function add(a, b){
  return a + b;
}

console.log(add(1,1));
```



How the Web works?

- **Web APIs**: control web browser/device functionality and make things happen (DOM, CSSOM, WebGL, WebSockets, Web Audio API, etc.).





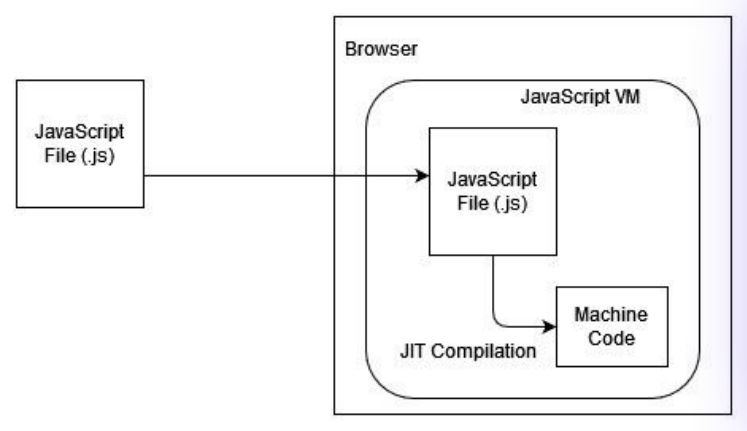
Problems with the Web

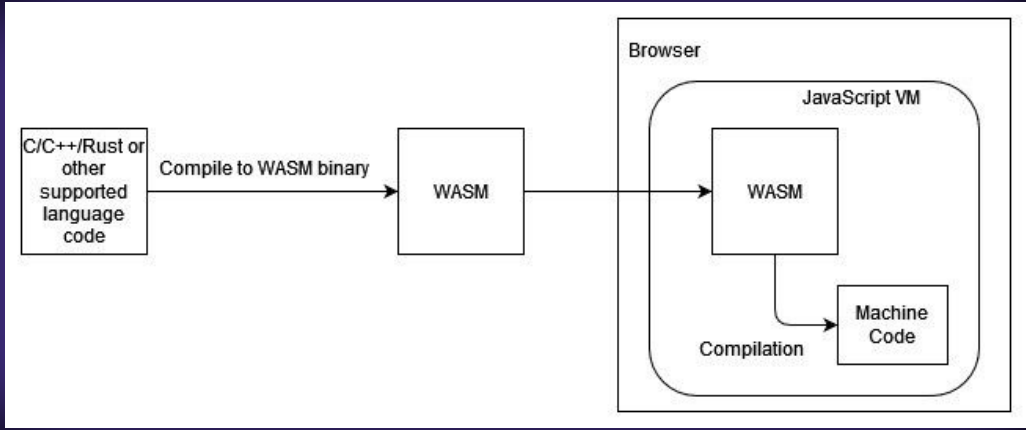
- Can't reuse code written in other languages in the Web.
- JavaScript still is not native -> JIT.
- Performance boundaries.
- Not fully portable -> Web APIs.

WASM: history

- PNaCl (Portable Native Client) by **Google** -> safely running native code from a web browser.
- asm.js by **Mozilla** -> subset of JavaScript with almost native performance;
- Announced in **2015** -> **Unity** demo;
- First version in 2017;
- W3C recommendation in 2019;
- Implemented in Firefox, Chrome, Edge, Safari, mobile browsers…

WASM: history





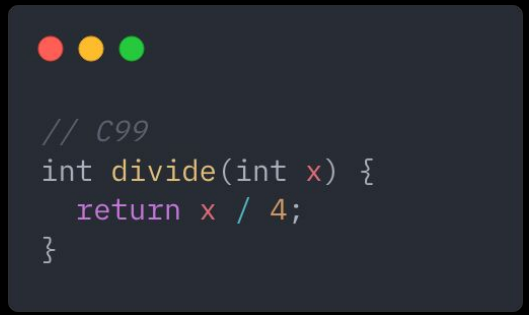
WASM: history

WASM: goals

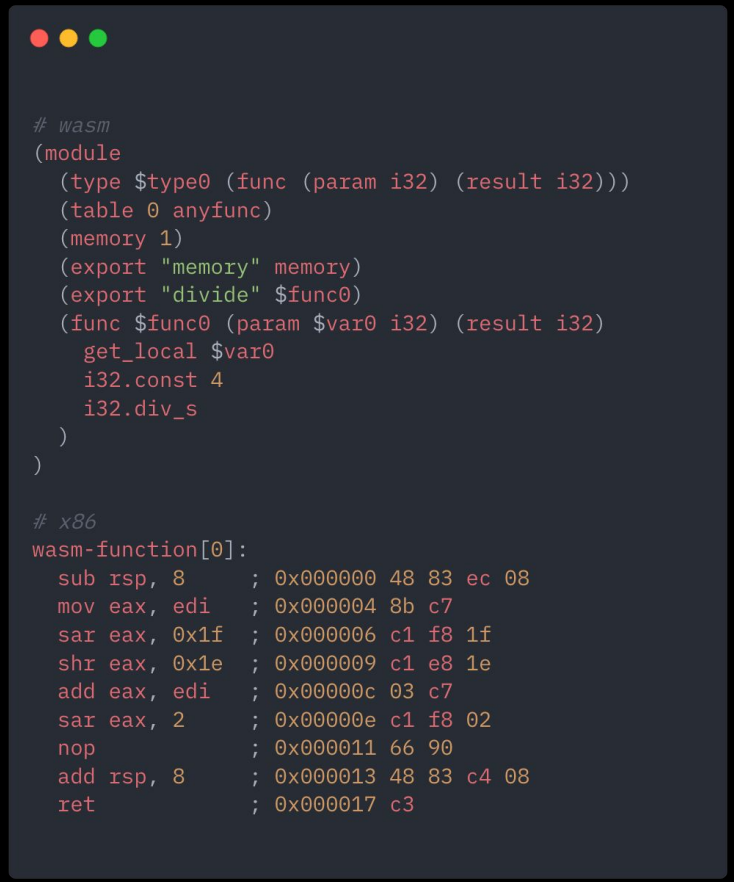
- ***Be fast, efficient, and portable*** –> executed at near-native speed across different platforms.
- ***Be readable and debuggable*** –> a human-readable text format (the specification for which is still being finalized) that allows code to be written, viewed, and debugged by hand.
- ***Keep secure*** –> specified to be run in a safe, sandboxed execution environment. Like other web code, it will enforce the browser's same-origin and permissions policies.
- ***Don't break the web*** –> maintains backwards compatibility.

WASM: goals

- JavaScript bindings, **doesn't replace it, but expands the Web.**
- Portable -> Allow code reuse in Web and **native**. Cross-Browser support.
- **For compilers, it's just another target.**



```c
// C99
int divide(int x) {
  return x / 4;
}
```



```
# wasm
(module
  (type $type0 (func (param i32) (result i32)))
  (table 0 anyfunc)
  (memory 1)
  (export "memory" memory)
  (export "divide" $func0)
  (func $func0 (param $var0 i32) (result i32)
    get_local $var0
    i32.const 4
    i32.div_s
  )
)

# x86
wasm-function[0]:
  sub rsp, 8      ; 0x000000 48 83 ec 08
  mov eax, edi    ; 0x000004 8b c7
  sar eax, 0x1f   ; 0x000006 c1 f8 1f
  shr eax, 0x1e   ; 0x000009 c1 e8 1e
  add eax, edi    ; 0x00000c 03 c7
  sar eax, 2      ; 0x00000e c1 f8 02
  nop             ; 0x000011 66 90
  add rsp, 8      ; 0x000013 48 83 c4 08
  ret             ; 0x000017 c3
```

```c
// C99
unsigned int divide(unsigned int x) {
  return x / 4;
}
```

```
# wasm
(module
  (type $type0 (func (param i32) (result i32)))
  (table 0 anyfunc)
  (memory 1)
  (export "memory" memory)
  (export "divide" $func0)
  (func $func0 (param $var0 i32) (result i32)
    get_local $var0
    i32.const 2
    i32.shr_u
  )
)

# x86
wasm-function[0]:
  sub rsp, 8      ; 0x000000 48 83 ec 08
  mov ecx, edi    ; 0x000004 8b cf
  mov eax, ecx    ; 0x000006 8b c1
  shr eax, 2      ; 0x000008 c1 e8 02
  nop             ; 0x00000b 66 90
  add rsp, 8      ; 0x00000d 48 83 c4 08
  ret             ; 0x000011 c3
```

WASM: stack machine.

WASM: stack machine.

```c
// C99
int mutiply(int a, int b) {
  return a*b;
}
```

```
(module
  (type $type0 (func (param i32 i32) (result i32)))
  (table 0 anyfunc)
  (memory 1)
  (export "memory" memory)
  (export "mutiply" $func0)
  (func $func0 (param $var0 i32) (param $var1 i32) (result i32)
    get_local $var1
    get_local $var0
    i32.mul
  )
)
```

Instruction	Stack
local.get 0	10
local.get 1	5, 10
int32.mul	50

WASM: instructions

- ***Memory***: load, store, size, grow…
- ***Control flow***: block, loop, br, if/else, call, drop…
- ***Numeric***:
  - *comparison* -> eq, ne, lt…
  - *arithmetic* -> add, sub, mul…
  - *bitwise* -> and, or, xor, shl, shr…
  - *conversion* -> extend, wrap, promote, demote, truncate…
  - *float specific* -> max, min, ceil, floor, sqrt…
- ***Variables***: get, set, local, global…
- ***Vector/SIMD***;

WASM: instructions

```
(module
  (memory $memory 1)
  (export "memory" (memory $memory))

  (func (export "store_in_mem") (param $num i32)
    i32.const 0
    local.get $num

    ;; store $num at position 0
    i32.store
  )
)
```

```
let url = "{%wasm-url%}";
await WebAssembly.instantiateStreaming(fetch(url)).then(
  (result) => {
    const store_in_mem = result.instance.exports.store_in_mem;
    const memory = result.instance.exports.memory;

    store_in_mem(100);

    var dataView = new DataView(memory.buffer);
    var first_number_in_mem = dataView.getUint32(0, true);

    console.log(first_number_in_mem);
  }
);
```

output:
> 100

WASM: instructions

```
(module
  (import "env" "log_bool" (func $log_bool (param i32)))
  (func $main
    ;; load `10` and `2` onto the stack
    i32.const 10
    i32.const 2

    i32.le_u ;; check if `10` is  less than or equal to '2'
    call $log_bool ;; log the result
  )
  (start $main)
)
```

```
let url = "{%wasm-url%}";

function log_bool(value) {
  console.log(Boolean(value));
}

await WebAssembly.instantiateStreaming(fetch(url), {
  env: {log_bool}
});
```

output:
> false

WASM: instructions

```
(module
  ;; import the browser console object, you'll need to
  ;; pass this in from JavaScript
  (import "console" "log" (func $log (param i32)))
  ;; create a function that takes in a number as a param,
  ;; and logs that number if it's not equal to 100.
  (func (export "log_if_not_100") (param $num i32)
    (block $my_block
      ;; $num is equal to 100
      local.get $num
      i32.const 100
      i32.eq
      (if
        (then
          ;; branch to the end of the block
          br $my_block
        )
      )
      ;; not reachable when $num is 100
      local.get $num
      call $log

    )
  )
)
```

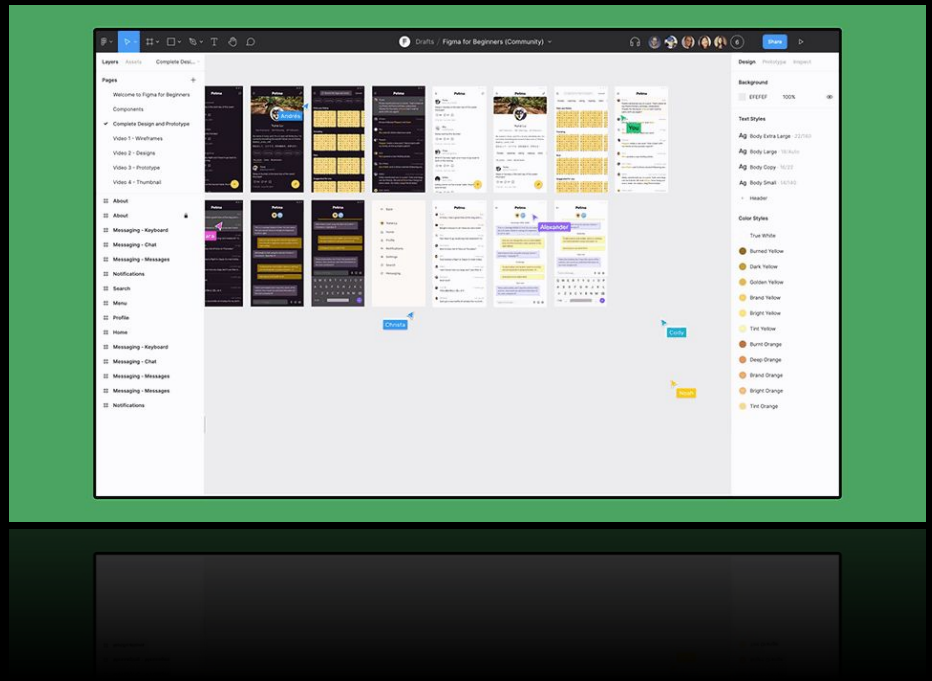
```
let url = "{%wasm-url%}";
await WebAssembly.instantiateStreaming(fetch(url), { console }).then(
  (result) => {
    const log_if_not_100 = result.instance.exports.log_if_not_100;

    log_if_not_100(99); // should log 99
    log_if_not_100(100); // should not log anything
    log_if_not_100(101); // should log 101
  }
);
```

output:

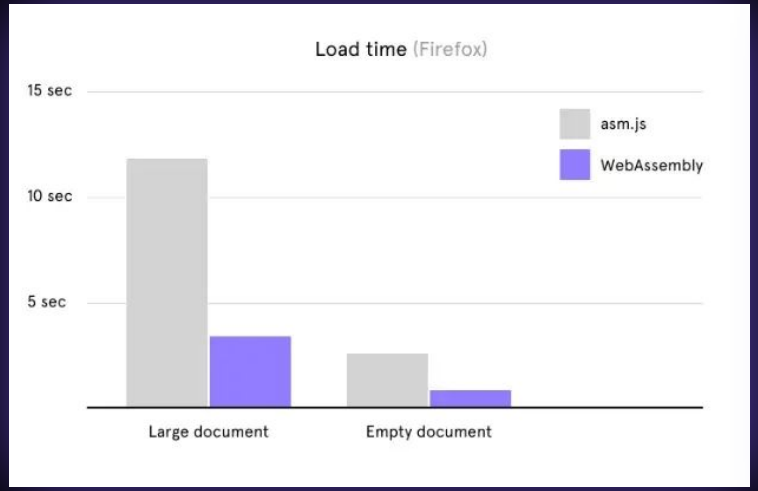
> 99
> 101

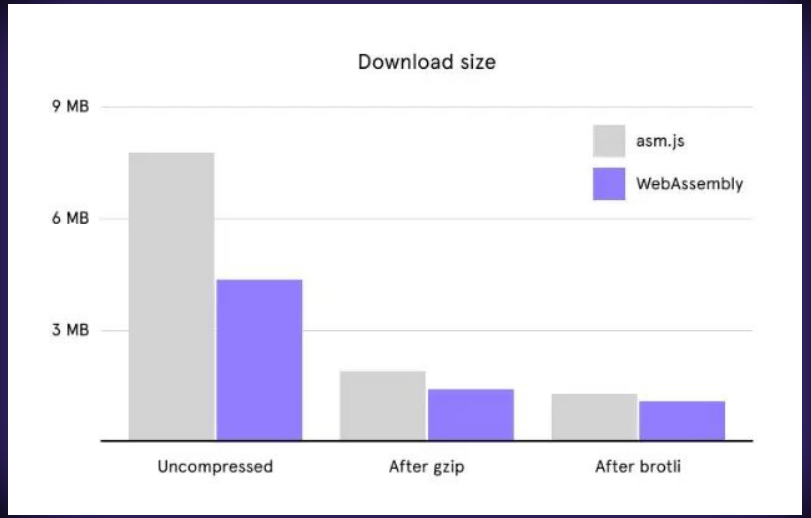
Examples: Figma

- It's a browser-based interface design tool with a powerful 2D WebGL rendering engine that supports very large documents.
- Built with C++ and previously compiled to asm.js.
- Cut load time by 3x.
- Load time no longer depends on application size -> cache.



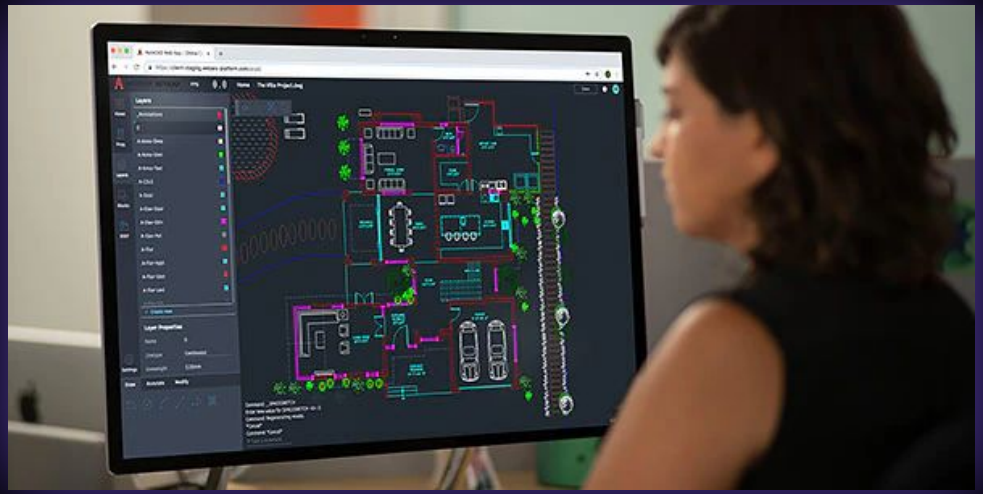
Examples: Figma





Examples: AutoCAD

- Uses **emscripten** to port pieces from the > 35 years old native application for AutoCAD.
- Codebase 10 years older than JavaScript itself!
- Built with native C/C++.

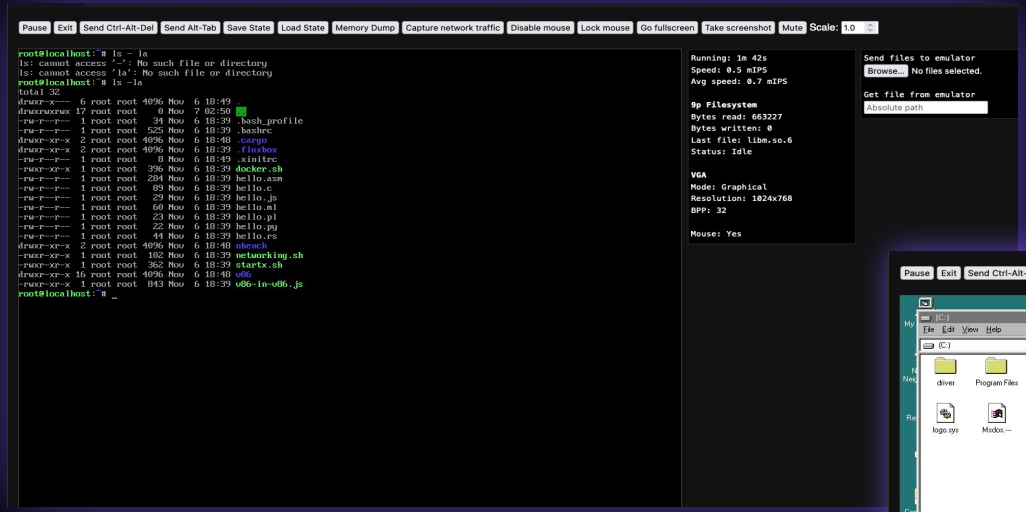


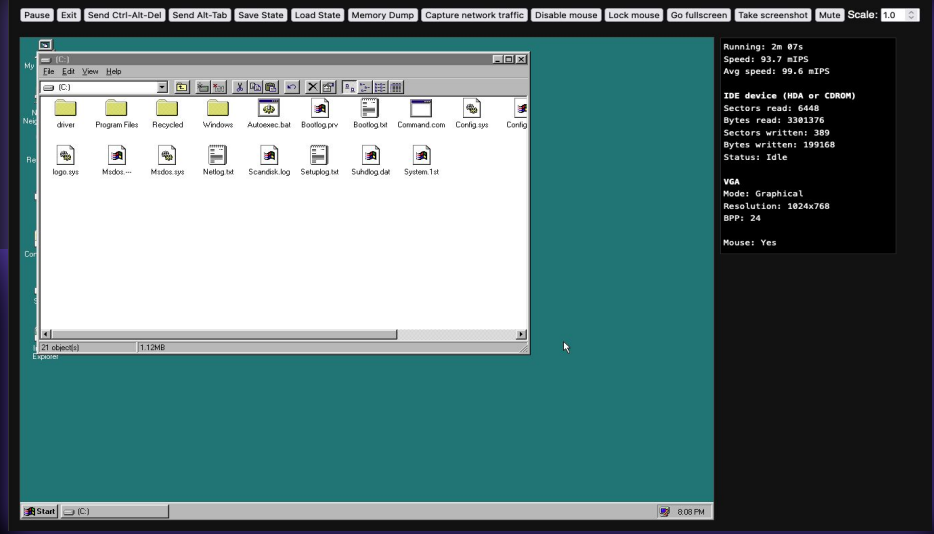
Examples

Experiments: Doom 3

Experiments: Doom 3 in VR

Experiments: v86

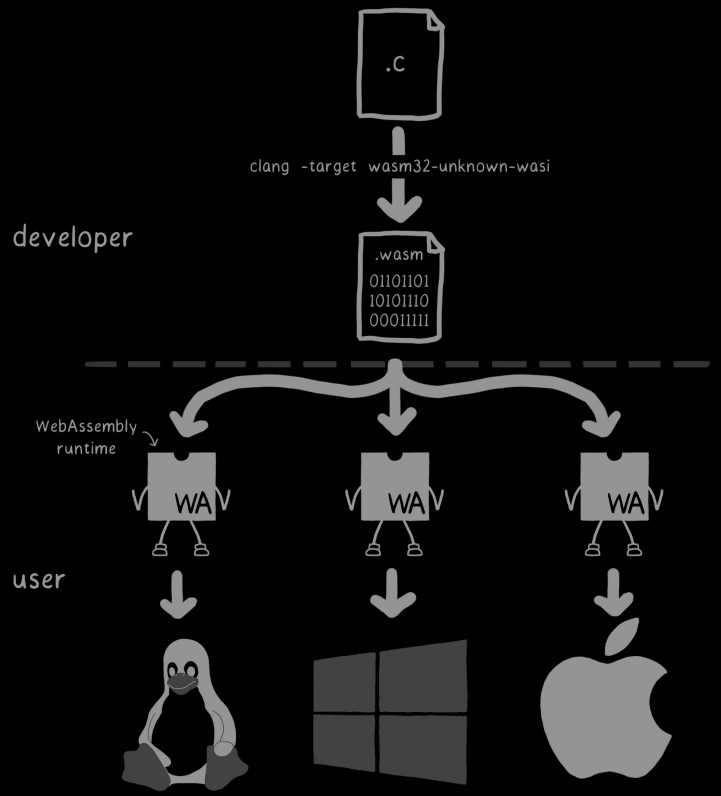




Demo.

Future

- **WASI** -> WebAssembly System Interface.
- **Cloud and Edge** -> Faster and lighter than containers.
- **Democratization** -> frontend developers can use libraries written in different languages with low level interaction/performance.



Questions?

Resources

- https://github.com/paulopacitti/wasm-demos
- https://www.youtube.com/watch?v=3sU557ZKjUs
- https://www.youtube.com/watch?v=sR22HtWztrY
- https://www.youtube.com/watch?v=7mBf3Gig9io
- https://www.youtube.com/watch?v=vqBtoPJoQOE
- https://www.youtube.com/watch?v=OGcm3rHg630
- https://www.youtube.com/watch?v=DFPD9yI-C70
- https://www.youtube.com/watch?v=6Y3W94_8scw

Thank you!

Gabriel Kenji, RA 216295
@paulopacitti, RA 185447