# WebAssembly

By Jonathan Easterman and Vivek Pradhan

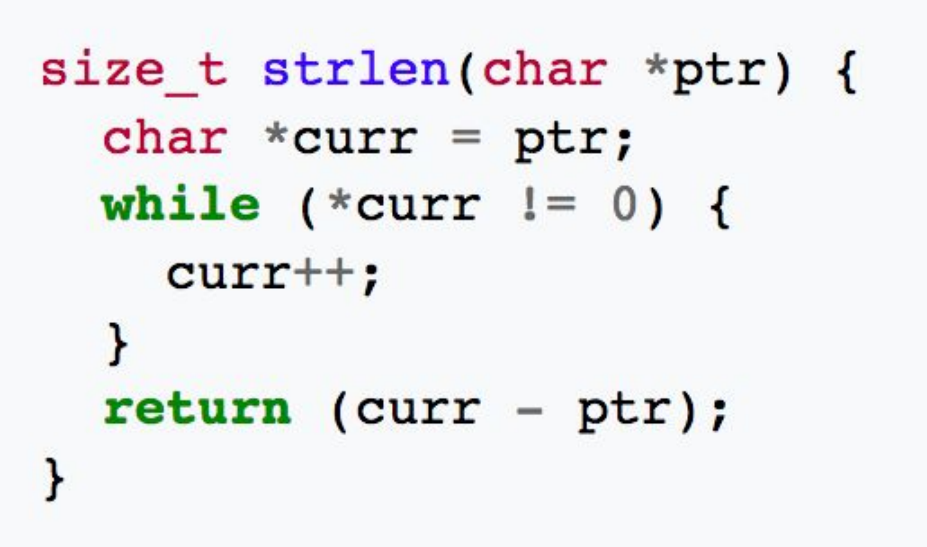
# The Problem

- The browser is the largest platform for applications
    - Statistic or graphic here
    - Electron
    - Google Drive
    - Computer Vision and Machine Learning (Tensorflow.js)
- Javascript is limited
    - Dynamically Typed
    - Automatic Memory Management
    - Single threaded

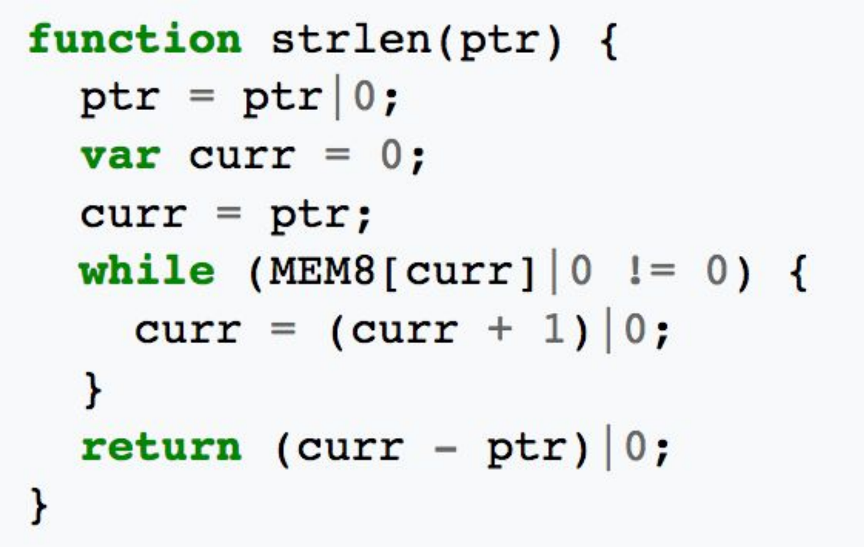
# Solution - Asm.js

- Precursor to WebAssembly
- Transpile C/C++ to optimized dialect of Javascript
  - Type coercion
  - Interact with memory via typed array
  - enables ahead-of-time compilation
- Results in large .js files
  - Need to include entire JS ports of C libraries
- Limited to expressiveness of Javascript
  - E.g. type expressiveness
- Not standardized between browser vendors

# Asm.js Example



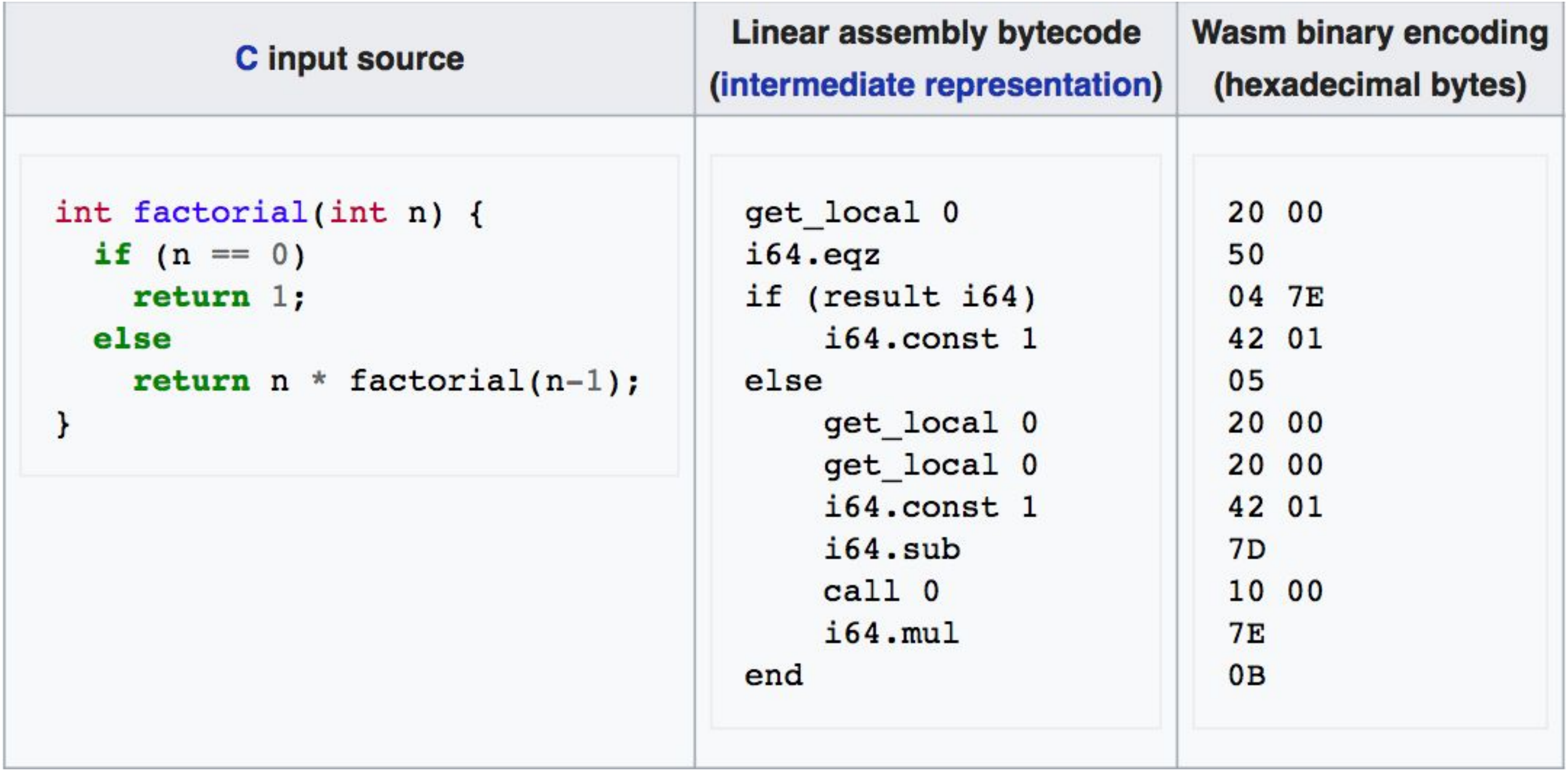
```c
size_t strlen(char *ptr) {
  char *curr = ptr;
  while (*curr != 0) {
    curr++;
  }
  return (curr - ptr);
}
```



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

# Enter WebAssembly

- Similar to JVM bytecode
    - Typed ➔ Faster to execute
    - Stack-machine bytecode ➔ Compact ➔ Faster to parse + transmit
- Not a replacement for Javascript
- Currently intended for manually managed languages
    - C/C++ - EmScripten + LLVM
    - Rust - rustc + LLVM



| C input source | Linear assembly bytecode (intermediate representation) | Wasm binary encoding (hexadecimal bytes) |
|---|---|---|
| `int factorial(int n) {` | `get_local 0` | `20 00` |
| `  if (n == 0)` | `i64.eqz` | `50` |
| `    return 1;` | `if (result i64)` | `04 7E` |
| `  else` | `    i64.const 1` | `42 01` |
| `    return n * factorial(n-1);` | `else` | `05` |
| `}` | `    get_local 0` | `20 00` |
| | `    get_local 0` | `20 00` |
| | `    i64.const 1` | `42 01` |
| | `    i64.sub` | `7D` |
| | `    call 0` | `10 00` |
| | `    i64.mul` | `7E` |
| | `end` | `0B` |

# Details

- Four types
    - 32 + 64-bit variants of int + float
- WebAssembly.Module()
    - Stateless Native code
- WebAssembly.Instance()
    - Stateful Instantiated module
- WebAssembly.Memory()
    - Resizable ArrayBuffer of raw bytes
    - Like a Heap for an Instance
- WebAssembly.Table()
    - Resizable typed array of opaque values
    - Currently used for function pointers

# Hypotheses

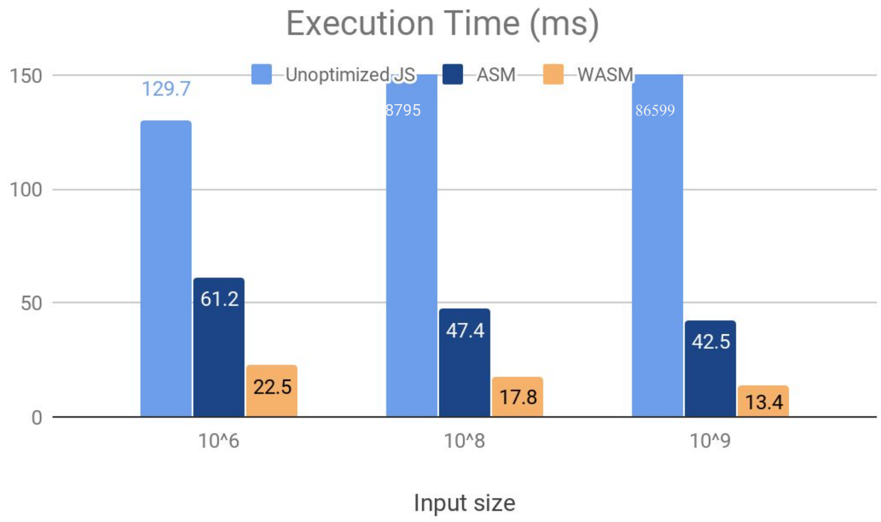
Goal: Compare JS, Asm.js, WebAssembly

- WebAssembly will perform better for computationally intensive tasks
- WebAssembly will suffer for tasks that involve moving/copying from Linear Memory to JS memory

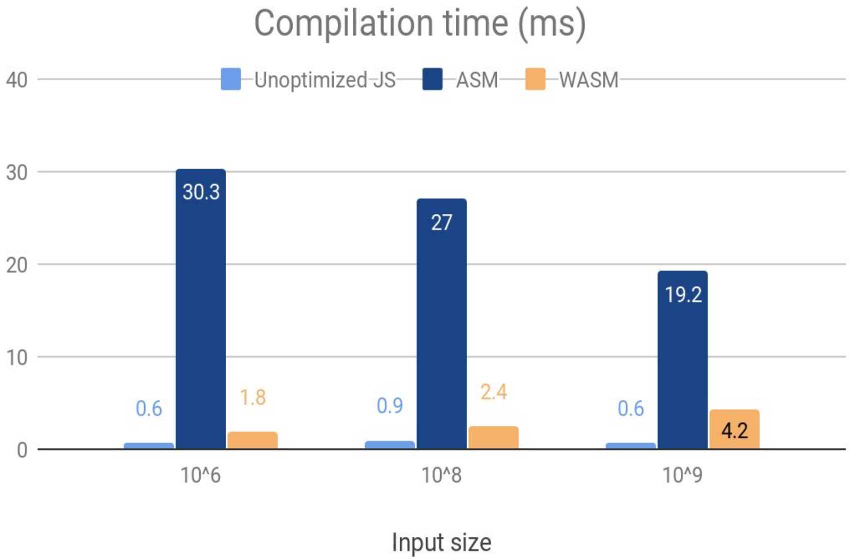
# Experimental Setup

- 3 programs written in JS and C
  - monte-carlo
  - spectral-norm
  - deepcopy
- EmScripten compiler
  - C → LLVM → WebAssembly
  - C → LLVM → Asm.js
  - Includes implementation of C stdlib
- Collect statistics via V8 profiler and Chrome Dev tools
  - Sample size: n = 10

# Monte Carlo

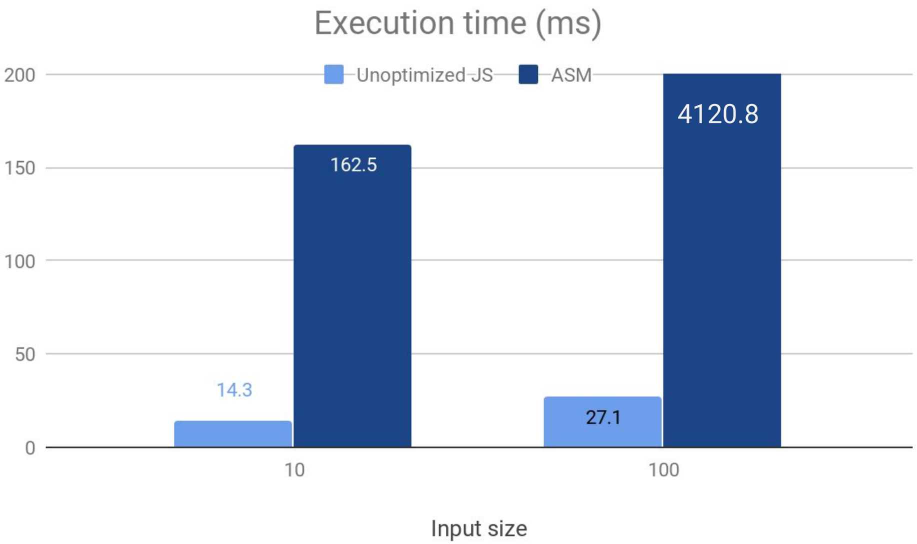


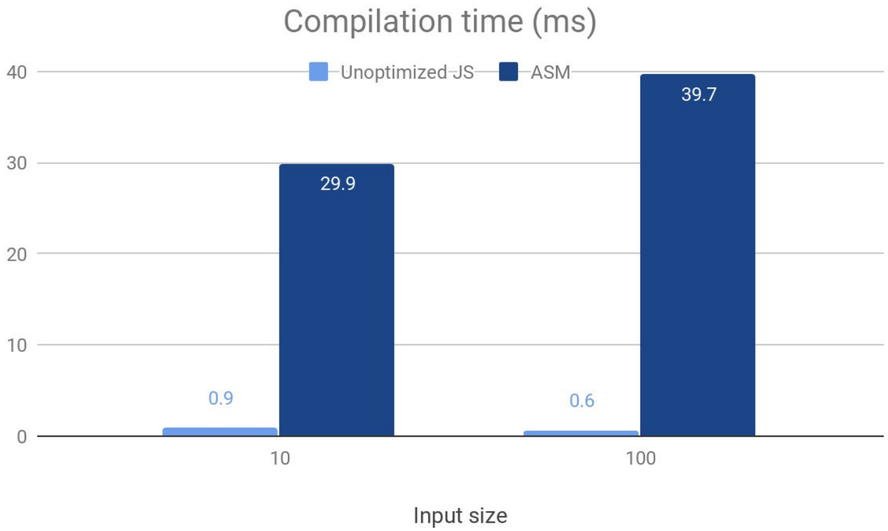
Execution Time (ms)
Unoptimized JS
ASM
WASM
150
100
50
0
129.7
8795
86599
61.2
47.4
42.5
22.5
17.8
13.4
10^6
10^8
10^9
Input size



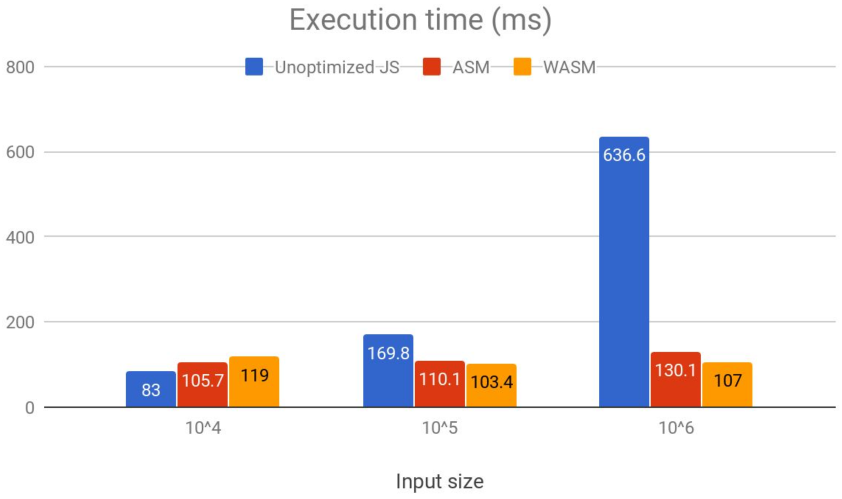
Compilation time (ms)
Unoptimized JS
ASM
WASM
40
30
20
10
0
0.6
30.3
1.8
0.9
27
2.4
0.6
19.2
4.2
10^6
10^8
10^9
Input size

# Spectral Norm

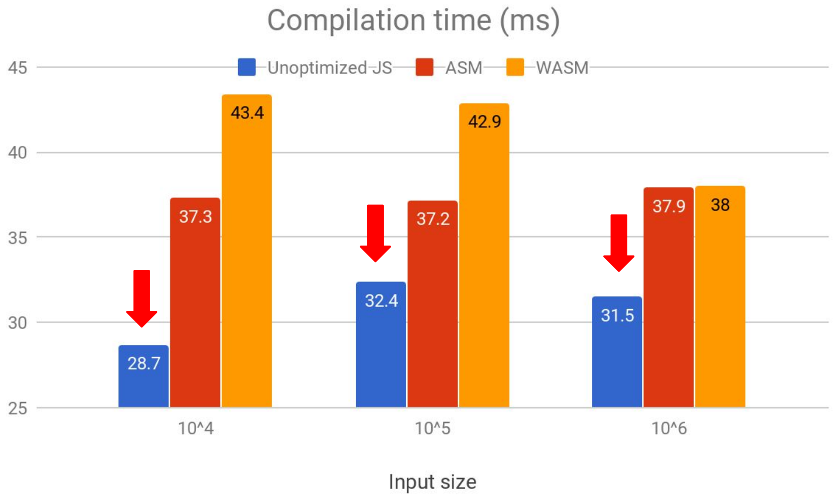

Execution time (ms)


Compilation time (ms)

# Deep copy



Execution time (ms)
Unoptimized JS
ASM
WASM
800
600
400
200
0
83
105.7
119
169.8
110.1
103.4
636.6
130.1
107
10^4
10^5
10^6
Input size



Compilation time (ms)
Unoptimized JS
ASM
WASM
45
40
35
30
25
28.7
37.3
43.4
32.4
37.2
42.9
31.5
37.9
38
10^4
10^5
10^6
Input size

# WebAssembly Limitations

- Does not support SIMD
- No GC
- Relies on JS APIs for many things
  - E.g. DOM manipulation
- Single threaded
- Early support in browsers
- Tooling is not the most accessible

# Conclusions

- WebAssembly is a promising tool for stealing performance back

- Opens web development to performance-oriented systems engineers

- In practice, hypothesize and test!!