WEB ASSEMBLY

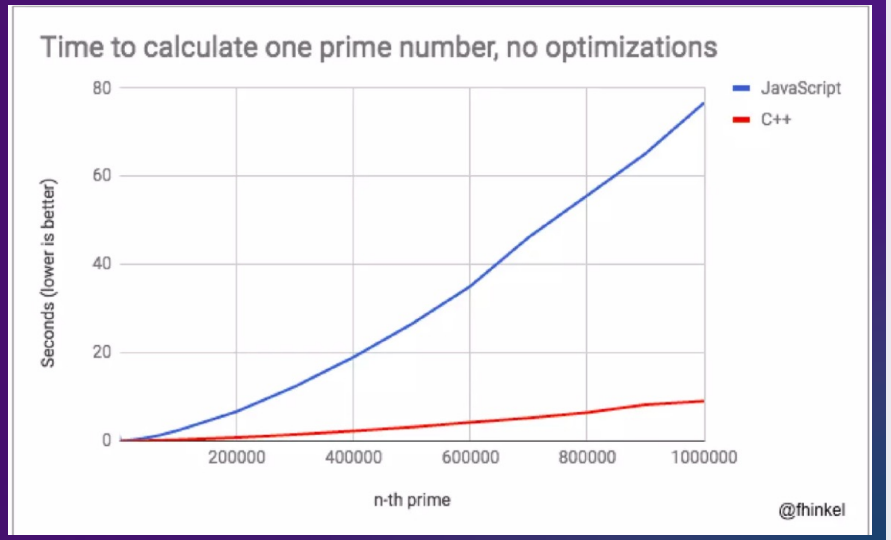
RUNNING NATIVE CODE IN THE BROWSER (AND BEYOND)

SHIVAM GUPTA | APR 9, 2024

AGENDA

- Javascript & Browsers
- Web Assembly – Intro, History, Browsers supported
- Compilers
- Languages supported
- Demo
- Applications
- Resources

JAVASCRIPT



Why so slow?

1. JavaScript was the only language that web browsers can run.

2. JavaScript was originally created to manage user interactions with DOM.

3. Now, we can write complete web applications on it.

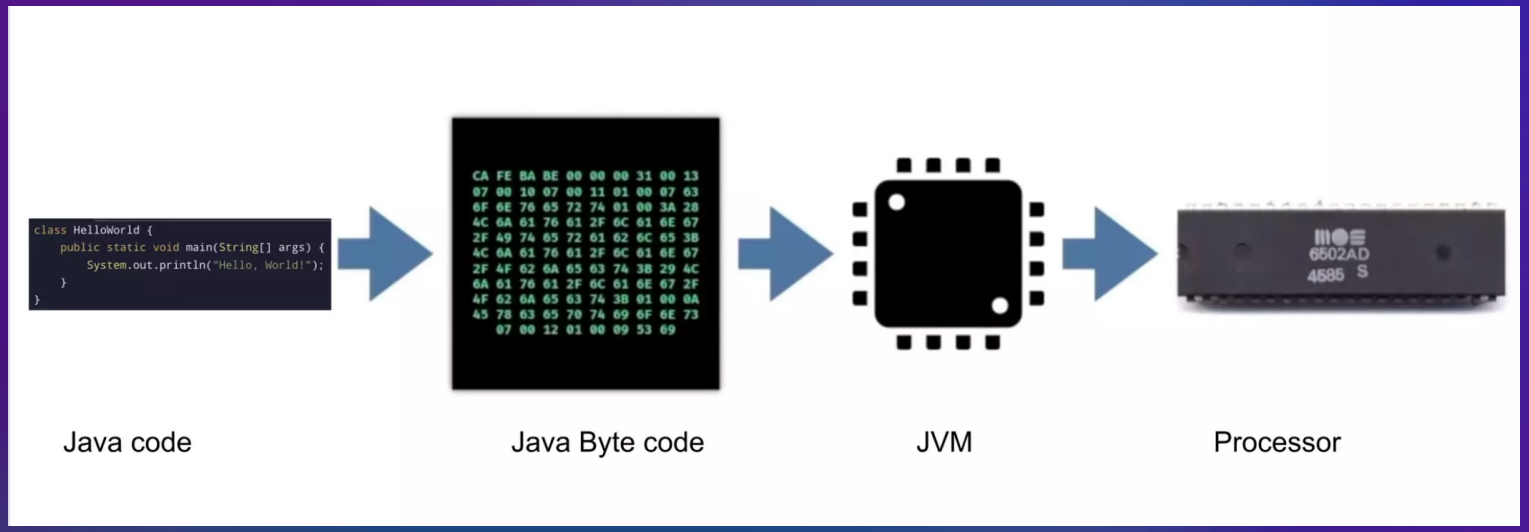
WHAT IS WEB ASSEMBLY?

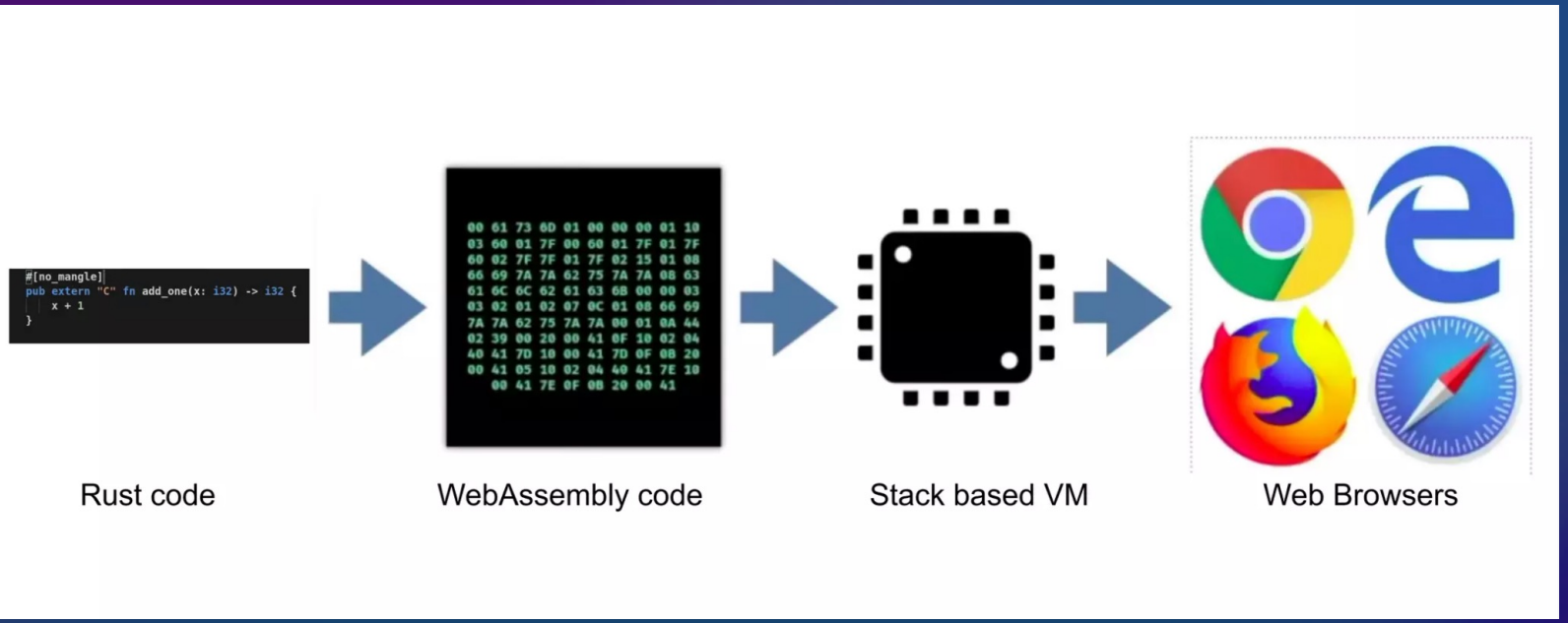
"WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for high-level languages like C/C++/Rust/Blazor, etc. , enabling deployment on the web for client and server applications."

June 2015 – Web Assembly was first announced	March 2016 – Google, Microsoft, Mozilla preview WA in their browsers	March 2017 – Begins to be shipped on-by-default in browsers

https://caniuse.com/?search=wasm

COMPILERS


class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}
CA FE BA BE 00 00 00 31 00 13
07 00 10 07 00 11 01 00 07 63
6F 6E 76 65 72 74 01 00 3A 28
4C 6A 61 76 61 2F 6C 61 6E 67
2F 49 74 65 72 61 62 6C 65 3B
4C 6A 61 76 61 2F 6C 61 6E 67
2F 4F 62 6A 65 63 74 3B 29 4C
6A 61 76 61 2F 6C 61 6E 67 2F
4F 62 6A 65 63 74 3B 01 00 0A
45 78 63 65 70 74 69 6F 6E 73
07 00 12 01 00 09 53 69
6502AD
4585 S
Java code
Java Byte code
JVM
Processor


#[no_mangle]
pub extern "C" fn add_one(x: i32) -> i32 {
x + 1
}
00 61 73 6D 01 00 00 00 01 10
03 60 01 7F 00 60 01 7F 01 7F
60 02 7F 7F 01 7F 02 15 01 08
66 69 7A 7A 62 75 7A 7A 0B 63
61 6C 6C 62 61 63 6B 00 00 03
03 02 01 02 07 0C 01 08 66 69
7A 7A 62 75 7A 7A 00 01 0A 44
02 39 00 20 00 41 8F 10 02 04
40 41 7D 10 00 41 7D 0F 0B 20
00 41 05 10 02 04 40 41 7E 10
00 41 7E 0F 0B 20 00 41
Rust code
WebAssembly code
Stack based VM
Web Browsers

LANGUAGES

Some of the languages that can be used with (or compiled to) WebAssembly

- C/C++ with Emscripten
- Kotlin with Kotlin/Native
- Swift with SwiftWasm
- C# with Mono or Blazer
- Java TeaVM
- Python Pyodide
- Rust with official compiler (rustc)

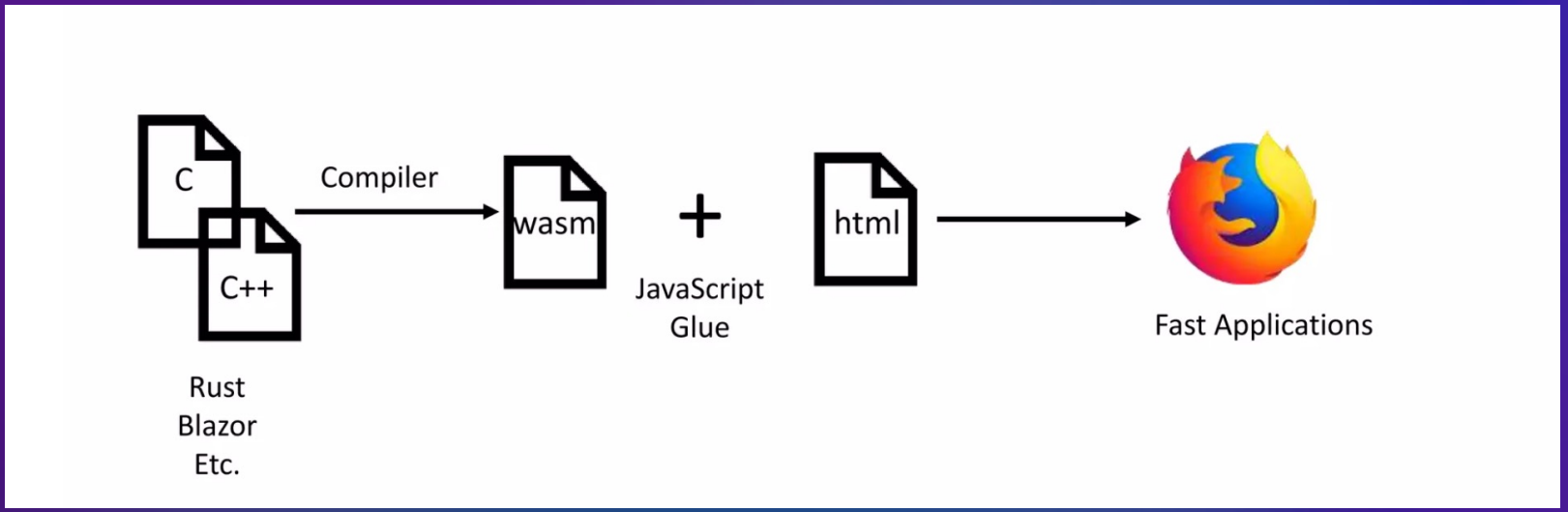
WHAT DOES IT LOOK LIKE?

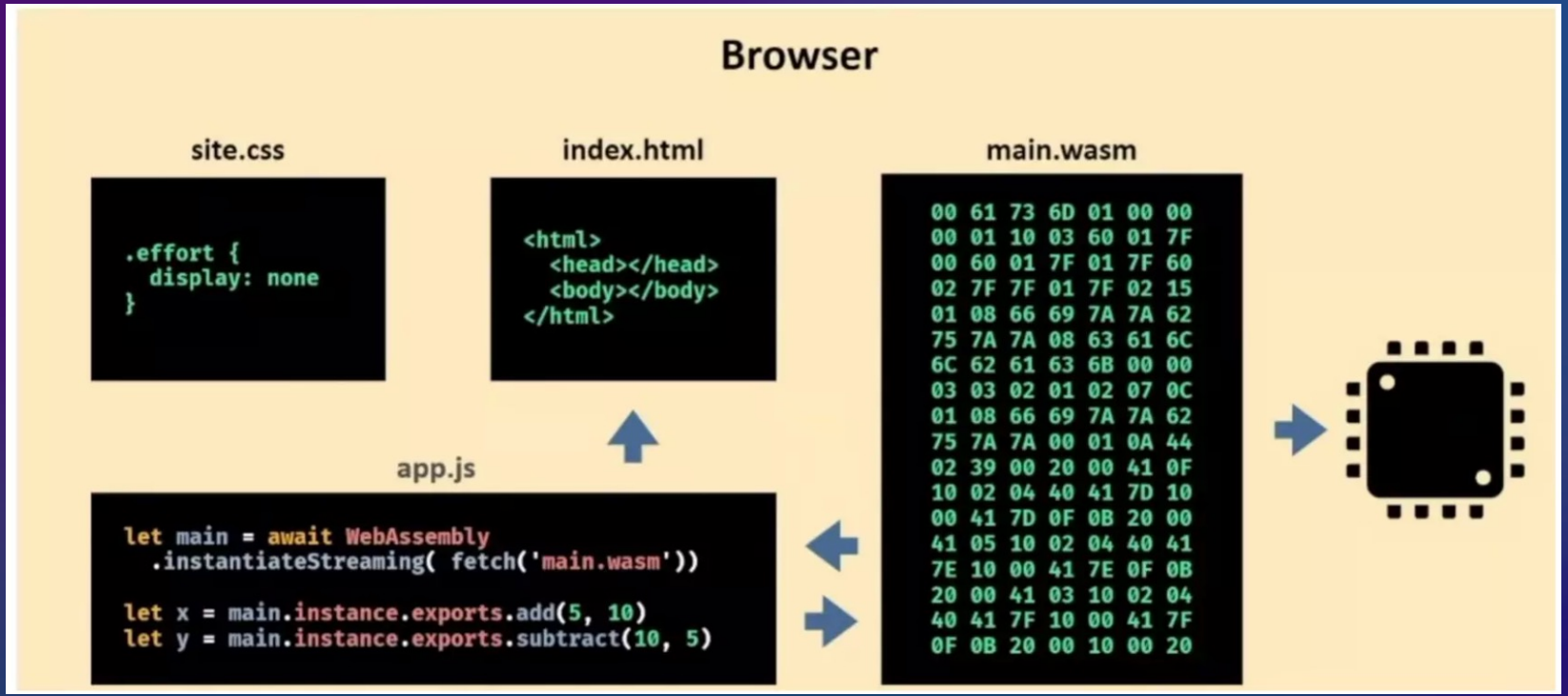
Rust source code	WebAssembly Text Format (.wat)	WebAssembly Binary Format (.wasm)

`#[no_mangle]`
`fn factorial(n: i64) -> i64 {`
`  if n == 0 {`
`  1`
`  } else {`
`  n * factorial(n - 1)`
`  }`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 06 01 60 01 7E 01 7E`
		`03 02 01 00`
		`0A 17 01`
		`15 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B`

ID	Section
0	custom section
1	type section
2	import section
3	function section
4	table section
5	memory section
6	global section
7	export section
8	start section
9	element section
10	code section
11	data section
12	data count section

SIMPLIFIED


C
C++
Compiler
wasm
+
JavaScript
Glue
html
Fast Applications
Rust
Blazor
Etc.


Browser
site.css
.effort {
display: none
}
index.html
<html>
<head></head>
<body></body>
</html>
main.wasm
00 61 73 6D 01 00 00
00 01 10 03 60 01 7F
00 60 01 7F 01 7F 60
02 7F 7F 01 7F 02 15
01 08 66 69 7A 7A 62
75 7A 7A 08 63 61 6C
6C 62 61 63 6B 00 00
03 03 02 01 02 07 0C
01 08 66 69 7A 7A 62
75 7A 7A 00 01 0A 44
02 39 00 20 00 41 0F
10 02 04 40 41 7D 10
00 41 7D 0F 0B 20 00
41 05 10 02 04 40 41
7E 10 00 41 7E 0F 0B
20 00 41 03 10 02 04
40 41 7F 10 00 41 7F
0F 0B 20 00 10 00 20
app.js
let main = await WebAssembly
.instantiateStreaming(fetch('main.wasm'))
let x = main.instance.exports.add(5, 10)
let y = main.instance.exports.subtract(10, 5)

LET'S SEE IT IN ACTION

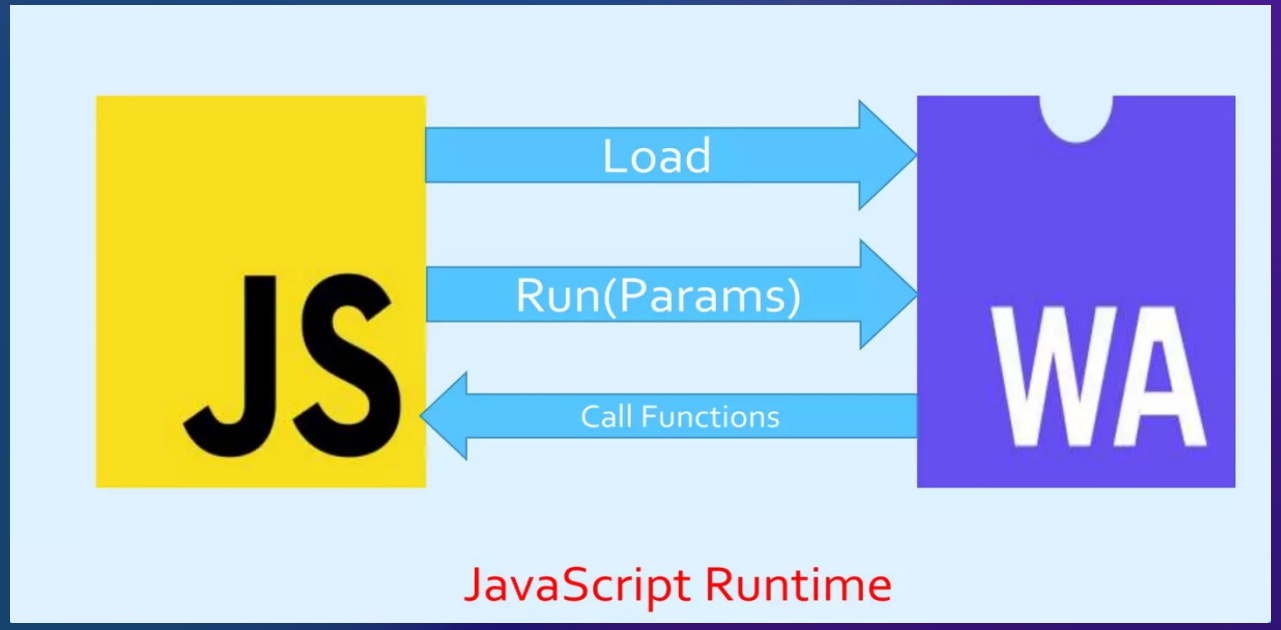
LINK TO DEMO REPO:

DEBUGGING WEB ASSEMBLY

Debug C/C++ WebAssembly | DevTools | Chrome for Developers

IS IT JAVASCRIPT REPLACEMENT?

- WebAssembly complements JavaScript
- Useful for heavier processes
- Most applications don’t need complex processing



WEB ASSEMBLY APPLICATIONS

Autodesk AutoCAD

Adobe Lightroom

Unity Engine

Quake

- eBay barcode scanner – 50FPS vs 1FPS (scans per seconds), 95% vs 20% success rate
- Wordpress Gutenberg post parser – 96% - 317% parsing speed increase

WASM & TEAMS

Generalized WASM Import on T21 Web.docx (sharepoint.com)

- Scalable secure solution for IC3 team to create WebWorker and WASM compilation into Teams 2.x Web to support better audio processing for noise suppression, and any scenario utilizing WebWorker and WASM modules from IC3.

Some efforts were also made in CDL to combine Rust/WASM. Reference

RESOURCES

- https://webassembly.org/
- https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm
- https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts
- https://emscripten.org/docs/getting_started/downloads.html#sdk-download-and-install

Q & A

Thank you