



WEBASSEMBLY

@vinaymavi

Agenda

- What is Web Assembly
- Some industry case studies
- How to load/compile web assembly on a web page
- How portability and security works in WASM
- Some basic examples
- Some basic industry use cases
- How to use WASM outside browser

What is WASM?

WebAssembly 1.0 has shipped in 4 major browser engines.



[Learn more](#)

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

Some industry case studies



AUTODESK®

About 90,400,000 results (0.47 seconds)

As of the latest market data, Autodesk AutoCAD currently holds a significant market share of 37% in the Computer Aided Design (CAD) industry, while Dassault Solidworks follows with a market share of 12%.

Jun 15, 2023



MarketSplash

<https://marketsplash.com> › autocad-statistics

60+ AutoCAD Statistics: Facts, Market Share & More



About featured snippets



Feedback



35 Years of Innovation



1982

First release

1992

AutoCAD for Mac

1993

Internet is Born

1998

AutoCAD Verticals

2000i

AutoCAD + internet



- Trusted CAD software across the globe
- Pioneer in the AEC industry
- Most Architects, Engineers, Contractors use AutoCAD .DWG



Figma

What is uses

- ✓ Searching for: **Figma tool usage percentage in design industry**
- ✓ Generating answers for you...



According to a survey conducted by TechMagic in 2020, Figma experienced an increase in its user base, with **66%** of designers adopting the tool, up from 37% the previous year ¹. Another survey conducted by UX Collective in 2021 reported that **63%** of designers use Figma as their primary UI tool ².

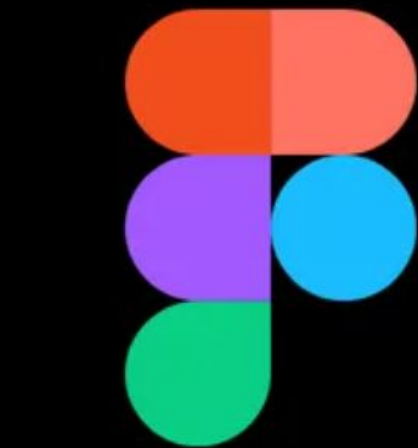
I hope this helps!

Learn more: [1. techmagic.co](#) [2. uxdesign.cc](#) [3. forbes.com](#)

2 of 30 ●

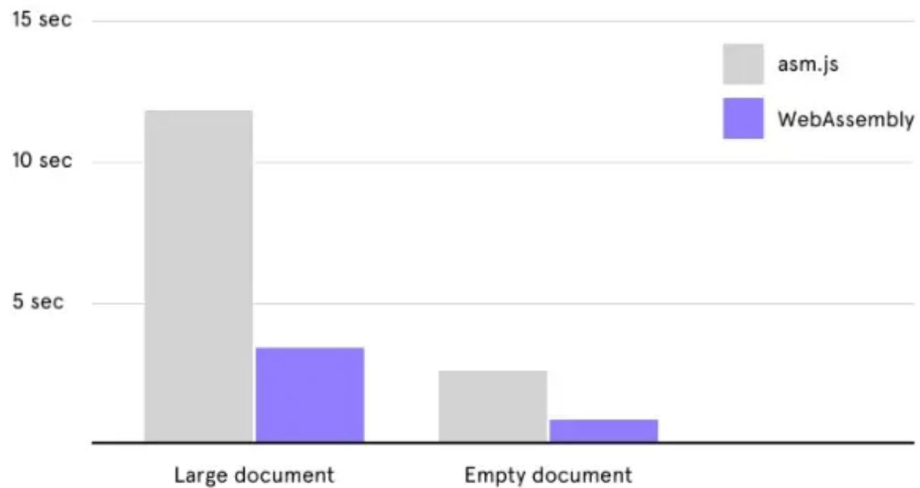
Sources:

<https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/>
<https://www.figma.com/blog/how-we-built-the-figma-plugin-system/>



Figma

Load time (Firefox)



Sources:

<https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/>

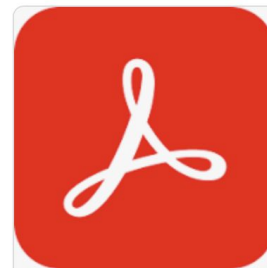
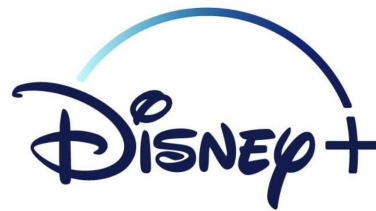
<https://www.figma.com/blog/how-we-built-the-figma-plugin-system/>

moz://a

ASM.JS



emscripten



Brave
Web browser



Google Meet
Video-communication app

Google Earth

3D representation of Earth



CØSMOS

WordPress Playground

playground.wordpress.net

AppscompaniesPrjtechchingInvestElectricity-PaymentVidhutSurakshaEarningsState!

My WordPress Website

Edit site0New

Howdy, admin

My WordPress Website

Sample Page

Mindblown: a blog about philosophy.

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

July 16, 2023

This is a cool fun experimental WordPress running in your browser :) All your changes are private and gone after a page refresh.

PHP 8.0 - WP 6.2 - Temporary

ElementsConsoleSourcesNetworkPerformanceMemory

wasmp

Preserve logDisable cacheNo throttling

494

wasmp

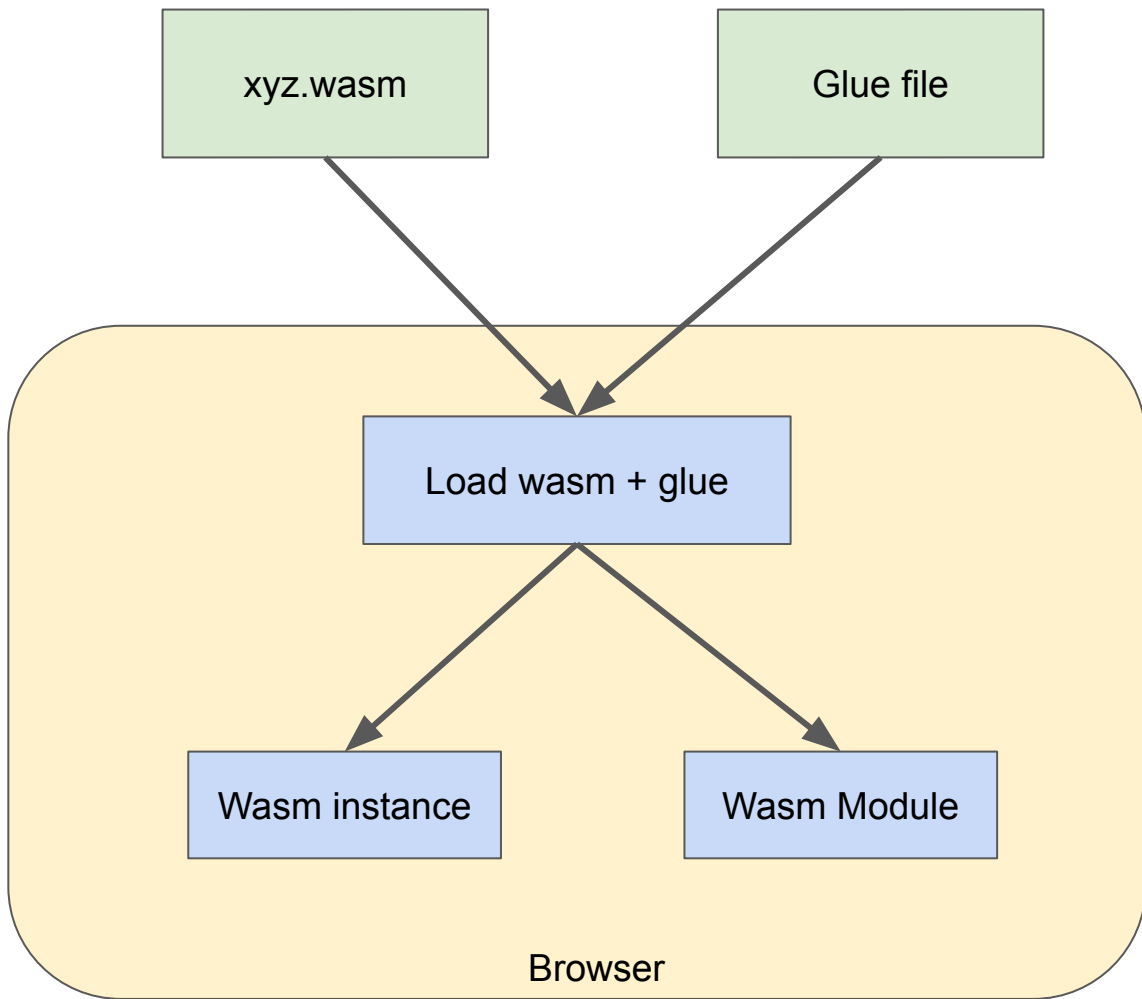
☐ Invert☐ Hide data URLs

☐ 3rd-party requests

Name	Status	Type	Initiator	Size	Time	Waterfall
php_8_0-89f34465.wasm	200	wasm	php_8_0-dcd25f...	1.8 MB	860 ms	

1 / 34 requests1.8 MB / 4.2 MB transferred5.8 MB / 17.8 MB resourcesFinish: 3.67 sDOMContentLoaded: 428 msLoad: 57

How to use web assembly?



```
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

GOOS=js GOARCH=wasm go build -o main.wasm hello.go

main.wasm

/wasm_exec.js

Provided by
platform(GoLang)

Glue file

EADME.md ●

<> helloworld.html M X

JS thumbnail.js

<> thumbnail-withthread.html

<> thumbnail-withworker.html

helloworld > <> helloworld.html > html > head > script

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Hello World WASM</title>
7      <script src="../wasm_exec.js"></script>
8      <script>
9          const go = new Go();
10             WebAssembly.instantiateStreaming(fetch("../main.wasm"), go.importObject).then((result) => {
11                 go.run(result.instance);
12             });
13
14      </script>
15  </head>
16  <body style="text-align: center;">
17
18  </body>
19  </html>
```




Elements

Console

Sources

Network

Performance



top ▼



Filter

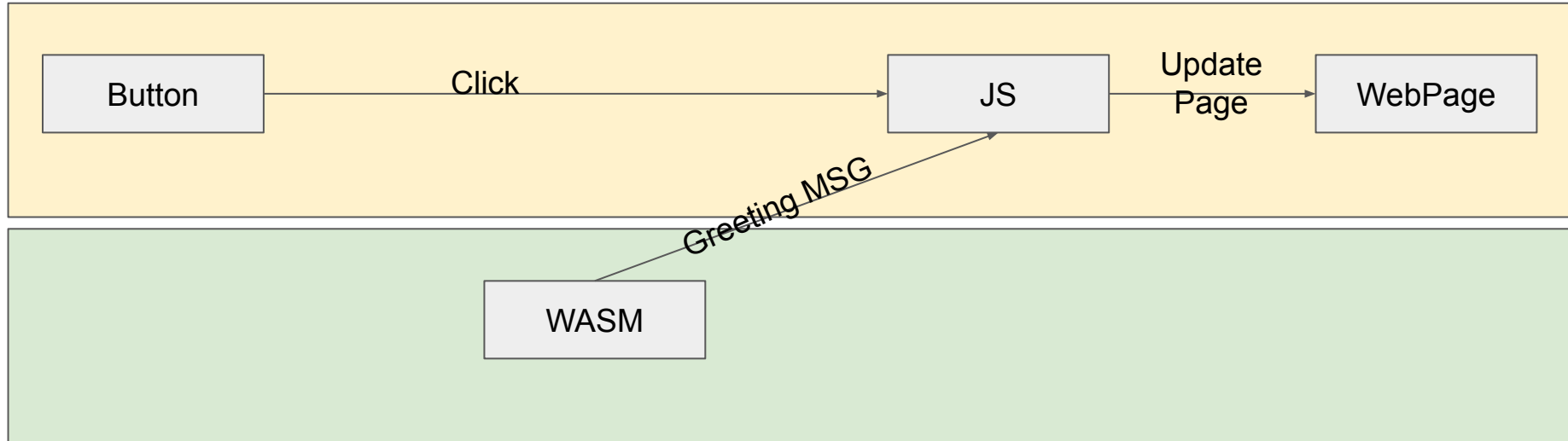
Custom level

Hello, 世界



One more a bit interactive example

- Call a WASM function from JavaScript
- Call JavaScript function from WASM



cpp_dynamic_greetings >  greeting.cpp >  main()

```
1  #include <iostream>
2  #include <emscripten.h>
3
4  extern "C" {
5      char* get_string(int index){
6          char* s = (char*)malloc(100);
7          char str1[] = "Hello from C++!";
8          char str2[] = "Hello World";
9          char str3[] = "Hello there :)";
10         char* strArr[] = {str1, str2, str3};
11
12
13         sprintf(s,"%s", strArr[index]);
14         return s;
15     }
16 }
17
18
19 int main(){
20     std::cout << "Hello, webassembly!" << std::endl;
21     char* s = get_string(1);
22     std::cout << s << std::endl;
23     return 0;
24 }
```

 cpp_greeting.html ×

cpp_dynamic_greetings >  cpp_greeting.html >  html >  body

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>C++</title>
7      <script src="./greeting.js"></script>
8  </head>
9  <body>
10     <h1></h1>
11     <button id="greeting">Generate Greeting Message</button>
12     <button id="main">Run main Function</button>
13
14     <script>
15         const greetingButton = document.querySelector('#greeting');
16         let count = 0;
17         greetingButton.addEventListener('click', () => {
18             const h1 = document.querySelector('h1');
19             const js_wrapper = Module.cwrap('get_string', 'string', ['int']);
20             const greeting = () => js_wrapper(count % 3);
21             count++;
22             h1.textContent = greeting();
23         });
24
25         const mainButton = document.querySelector('#main');
26         mainButton.addEventListener('click', () => {
27             const js_wrapper = Module.cwrap('main', 'number', []);
28             js_wrapper();
29         });
30     </script>
31 </body>
32 </html>
```

Hello from C++!

Generate Greeting Message

Run main Function

<https://bit.ly/wasm-e1>



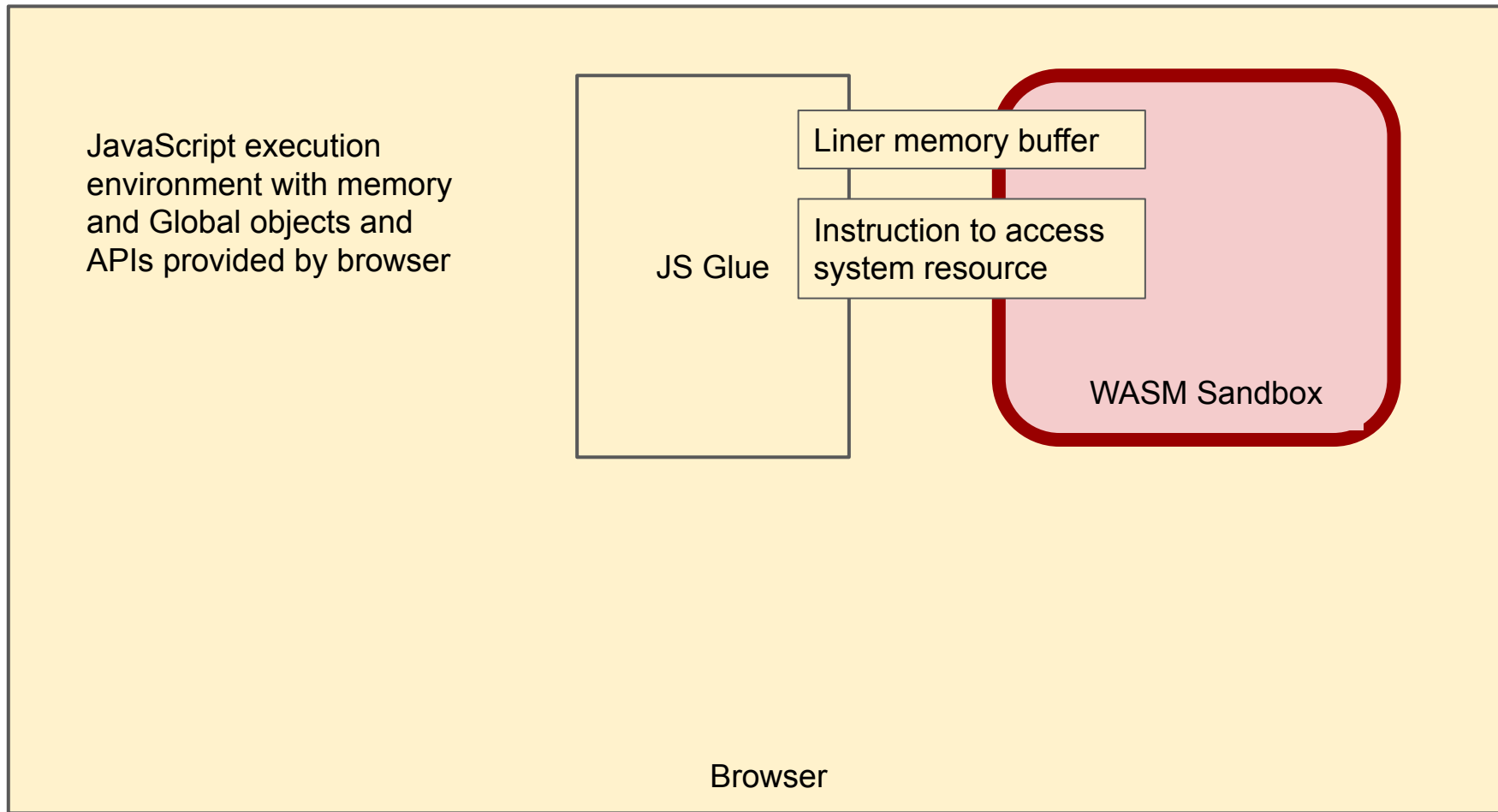
How type conversion is working?

```
greeting.cpp 2 x
cpp_dynamic_greetings > C++ greeting.cpp > main()
1  #include <iostream>
2  #include <emscripten.h>
3
4  extern "C" {
5      char* get_string(int index){
6          char* s = (char*)malloc(100);
7          char str1[] = "Hello from C++!";
8          char str2[] = "Hello World";
9          char str3[] = "Hello there :)";
10         char* strArr[] = {str1, str2, str3};
11
12
13         sprintf(s,"%s", strArr[index]);
14         return s;
15     }
16 }
17
18
19 int main(){
20     std::cout << "Hello, webassembly!" << std::endl;
21     char* s = get_string(1);
22     std::cout << s << std::endl;
23     return 0;
24 }
```

Pointer of char array

```
cpp_greeting.html x
cpp_dynamic_greetings > < cpp_greeting.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>C++</title>
7      <script src="./greeting.js"></script>
8  </head>
9  <body>
10     <h1></h1>
11     <button id="greeting">Generate Greeting Message</button>
12     <button id="main">Run main Function</button>
13
14     <script>
15         const greetingButton = document.querySelector('#greeting');
16         let count = 0;
17         greetingButton.addEventListener('click', () => {
18             const h1 = document.querySelector('h1');
19             const js_wrapper = Module.cwrap('get_string', ['int'], []);
20             const greeting = () => js_wrapper(count++);
21             h1.textContent = greeting();
22         });
23
24         const mainButton = document.querySelector('#main');
25         mainButton.addEventListener('click', () => {
26             const js_wrapper = Module.cwrap('main', 'number', []);
27             js_wrapper();
28         });
29     </script>
30
31 </body>
32 </html>
```

How JS handling pointer?



```
17 greetingButton.addEventListener('click', () => {
18     const h1 = document.querySelector('h1');
19     const js_wrapper = Module.cwrap('get_string', 'string', ['int']);
20     const greeting = () => js_wrapper(count % 3);
21     count++;
22     h1.textContent = greeting();
23 });
```

JS: count

JS: h1

JS Fn:
js_wrapper

JS Fn: greeting

JS: Hello from c++(msg)

JS Glue

W:count

w:Hello from c++

WASM Memory

Browser

```
char* get_string(int index){
char* s = (char*)malloc(100);
char str1[] = "Hello from C++!";
char str2[] = "Hello World";
char str3[] = "Hello there :)";
char* strArr[] = {str1, str2, str3};
sprintf(s,"%s", strArr[index]);
return s;
}
```

WASM Sandbox

Example Thumbnail Creation

<https://bit.ly/vp-wasm>



Example Thumbnail Creation With Web Worker

<https://bit.ly/vp-wasm>



Example Thumbnail Creation with Threads

<https://bit.ly/vp-wasm>



how threads are working here as browser do not have something called threads in built?



How WebAssembly threads work

WebAssembly threads is not a separate feature, but a combination of several components that allows WebAssembly apps to use traditional multithreading paradigms on the web.

Web Workers

First component is the regular [Workers](#) you know and love from JavaScript. WebAssembly threads use the `new Worker` constructor to create new underlying threads. Each thread loads a JavaScript glue, and then the main thread uses `Worker#postMessage` method to share the compiled `WebAssembly.Module` as well as a shared `WebAssembly.Memory` (see below) with those other threads. This establishes communication and allows all those threads to run the same WebAssembly code on the same shared memory without going through JavaScript again.

Web Workers have been around for over a decade now, are widely supported, and don't require any special flags.

example.c:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *thread_callback(void *arg)
{
    sleep(1);
    printf("Inside the thread: %d\n", *(int *)arg);
    return NULL;
}

int main()
{
    puts("Before the thread");

    pthread_t thread_id;
    int arg = 42;
    pthread_create(&thread_id, NULL, thread_callback, &arg);

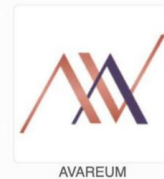
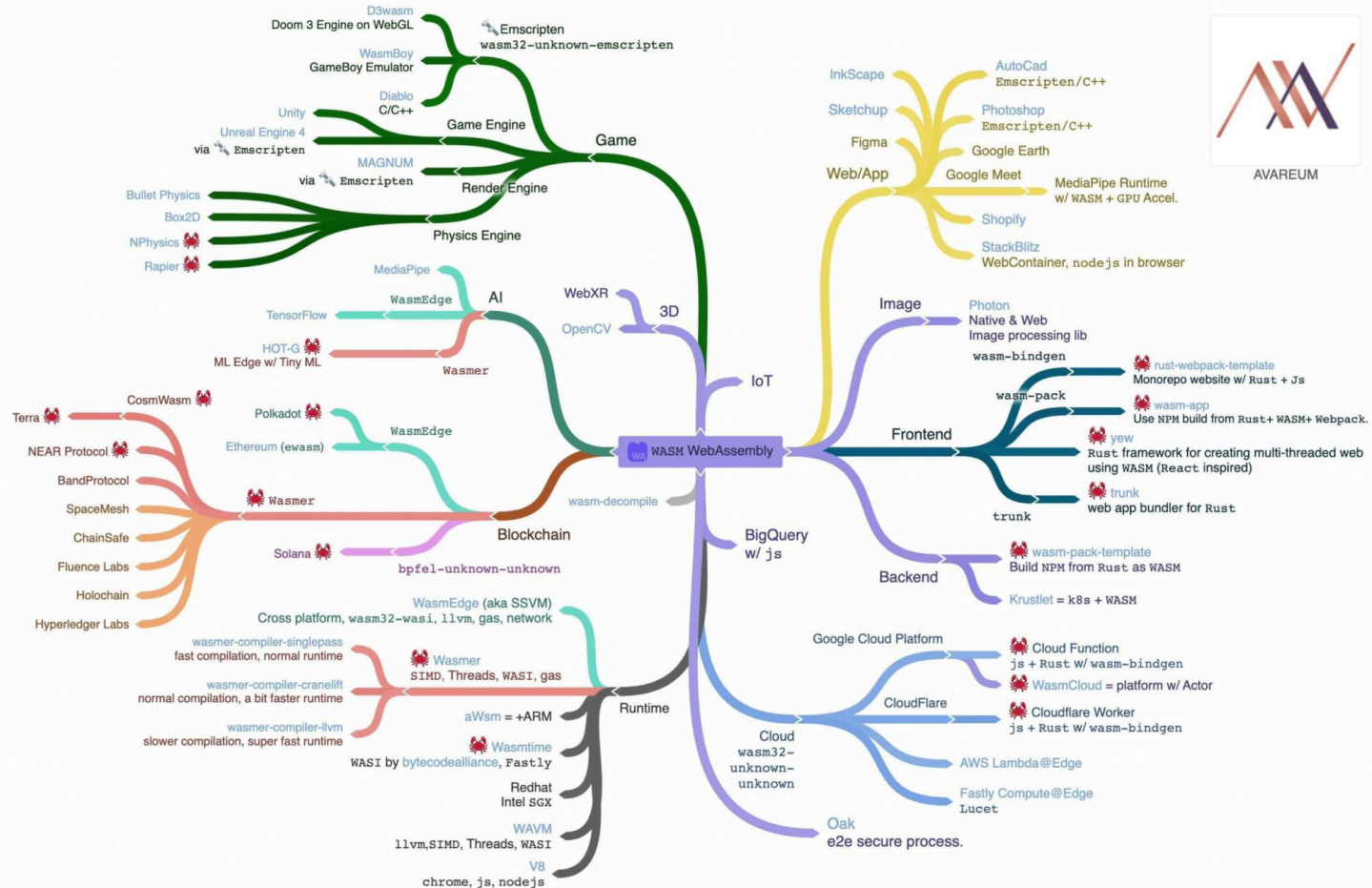
    pthread_join(thread_id, NULL);

    puts("After the thread");

    return 0;
}
```

```
emcc -pthread example.c -o example.js
```

How to use WASM outside browsers?





Tweet



Solomon Hykes / @shykes@hachyderm.io ✓

@solomonstre

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!



Lin Clark @linclark · Mar 27, 2019

WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

hacks.mozilla.org/2019/03/standa...



Solomon Hykes / @shykes@hachyderm.io ✓

@solomonstre

If WASM
Docker.
future of
link. Let'



Follow

Solomon Hykes / @shykes@... ✓

@solomonstre

Making @dagger_io, the programmable CI/CD engine. Before that: founded Docker. "No is temporary, yes is forever".



Lin
WebAs:
one gia

📢 Ann
the web

hacks.n

15.3K Following **48.8K** Followers

Followed by The GraphQL Guide, Netlify, and 25 others you follow

ave neede
ly on the s
erface wa

future. And

g WebAsse

WASI

WASI stands for "WebAssembly System Interface." It is a standard interface designed to allow WebAssembly code to run in a wide range of environments beyond just web browsers.

Originally, WebAssembly was primarily focused on running code within web browsers, but with the introduction of WASI, it became possible to execute WebAssembly modules in standalone environments outside of the browser, such as on servers, edge devices, and IoT devices.

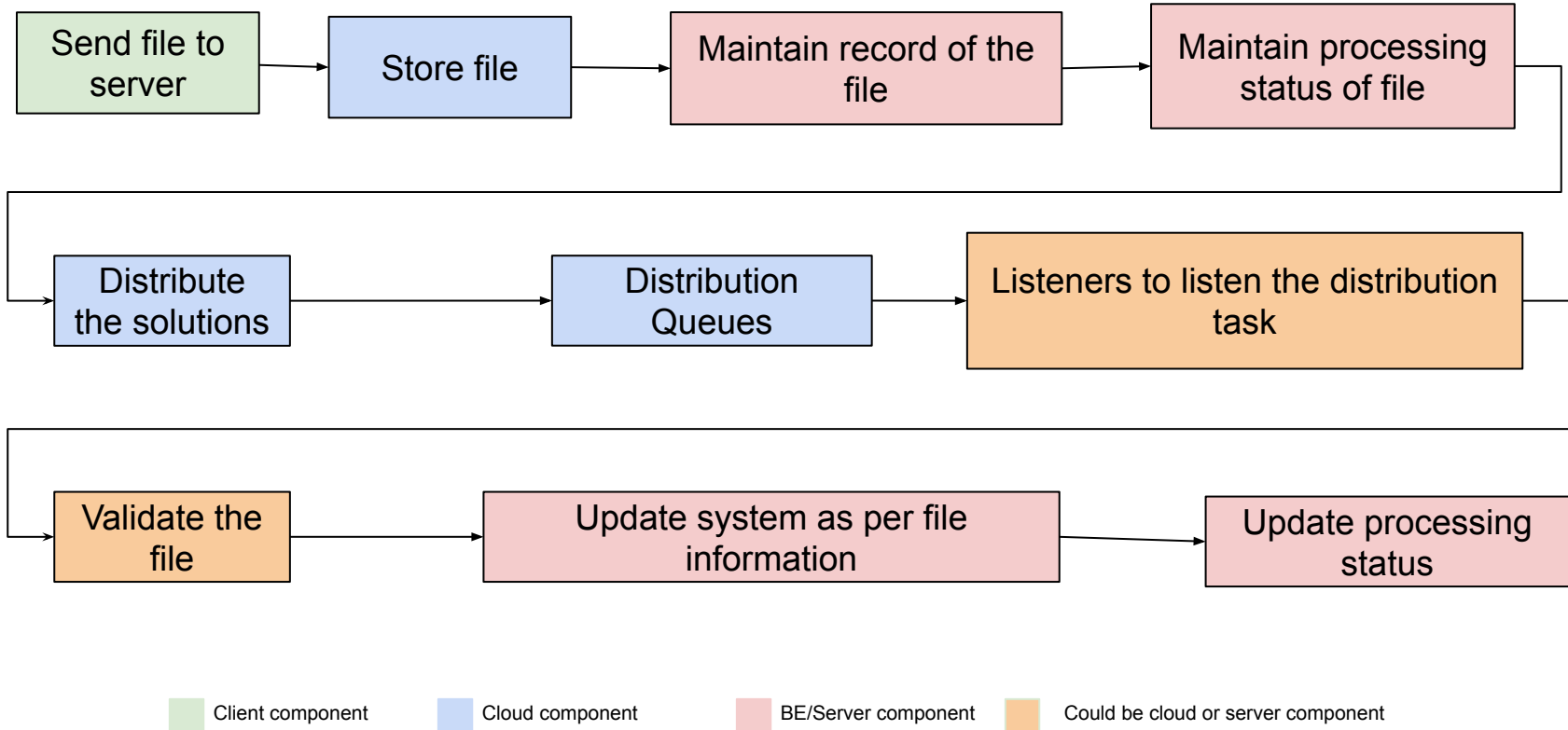
TODO: Prepare WASI examples

Use cases for using WASM?

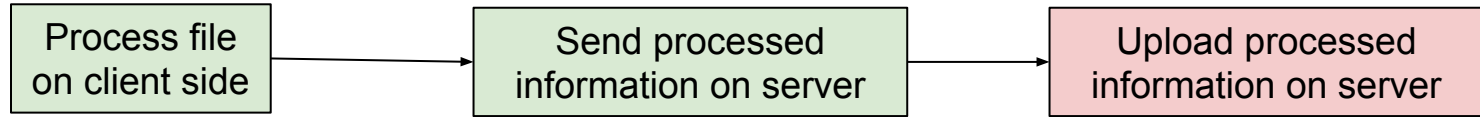
Edge Computing

Edge computing is a decentralized computing paradigm that brings data processing closer to the source of data generation, which is typically at or near the edge of the network. In traditional cloud computing, data is sent to centralized data centers for processing, but with edge computing, data processing occurs locally, on devices or edge servers, reducing latency, bandwidth usage, and reliance on a distant data center.

Data processing BE flow



Data processing on Edge



Client component



Cloud component



BE/Server component



Could be cloud or server component

How WASM helps in edge computing?

- Code sharing (Same code can be used on server and Edge)
- Build once run anywhere
- Consistent deployment on all edge types
- Less development and maintenance

LLVM, POSIX, JS Glue

Than you very much.

Merci beaucoup.

Que tú mucho.

非常比你。

どうもありがとうございます。

Feedback Link

<https://forms.gle/Povj2XEn6nj5Fma88>



References

- <https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/>
- <https://www.figma.com/blog/how-we-built-the-figma-plugin-system/>
- <https://web.dev/ps-on-the-web/>
- <https://web.dev/emscripting-a-c-library/#the-holy-grail-compiling-a-c-library>
- https://eng.snap.com/cross_platform_messaging_experience
- <https://web.dev/wordpress-playground/>
- <https://blog.unity.com/technology/webassembly-is-here>
- [How we're bringing Google Earth to the web](#)
- <https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>
- <https://blog.developer.adobe.com/acrobat-on-the-web-powered-by-webassembly-782385e4947e>
- <https://wasmer.io/posts/webassembly-core-2-reporting-from-the-town-square>
- <https://www.amazon.science/blog/how-prime-video-updates-its-app-for-more-than-8-000-device-types>
- <https://www.w3.org/TR/wasm-core-1/#design-goals%E2%91%A0>
- <https://www.infoworld.com/article/3651503/the-rise-of-webassembly.html>
- https://twitter.com/solomonstre/status/1111004913222324225?s=20&t=PCg2X4O9D_96xLSlqUzSRw
- <https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>
- <https://wasi.dev/>
- <https://github.com/bytecodealliance/wasmtime>
- <https://www.fastly.com/blog/announcing-lucet-fastly-native-webassembly-compiler-runtime>
- <https://chat.openai.com/share/ce56b3fb-3577-4738-a4a9-cbf40c955261>