WEBASSEMBLY

@vinaymavi

Agenda

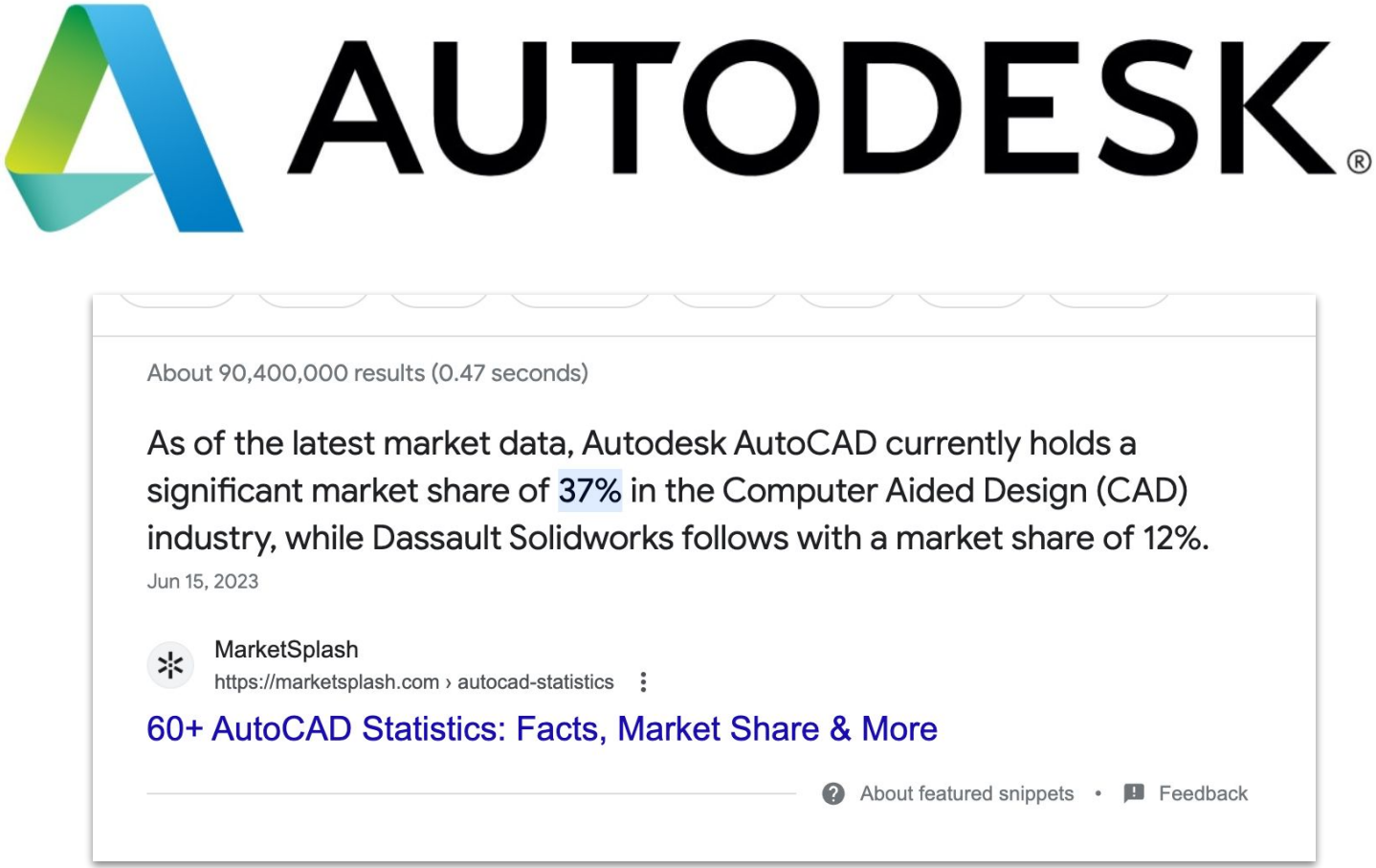
- What is Web Assembly
- Some industry case studies
- How to load/compile web assembly on a web page
- How portability and security works in WASM
- Some basic examples
- Some basic industry use cases
- How to use WASM outside browser

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

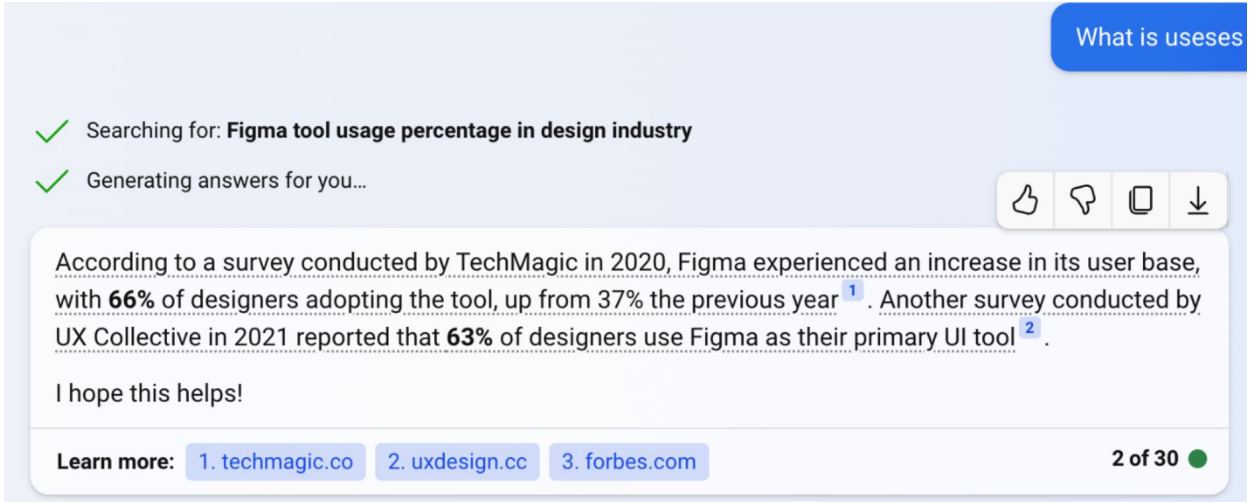
AUTODESK® AUTOCAD® | 35 Years of Innovation

1982	1992	1993	1998	2000i
First release	AutoCAD for Mac	Internet is Born	AutoCAD Verticals	AutoCAD + internet

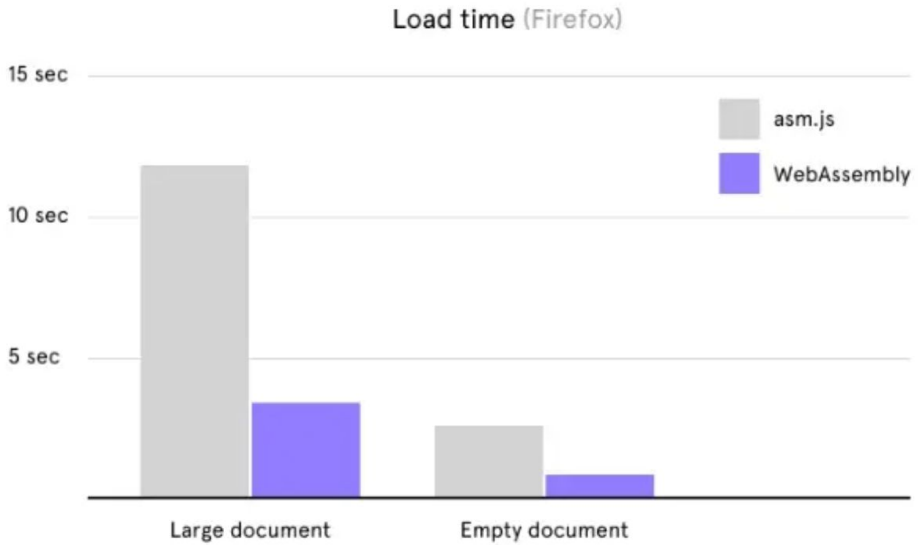
- Trusted CAD software across the globe
- Pioneer in the AEC industry
- Most Architects, Engineers, Contractors use AutoCAD .DWG

Source: Talk from Autodesk

Figma



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/



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/

But how?

- How Autodesk able to launch their 30 year old c++ try and tested code to web with in a single year year?
- How a small startup able to bring their idea to the web with performance?

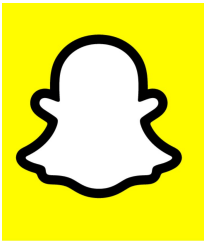
ASM.JS


emscripten


Ps
Photoshop


prime video


WordPress




unity


HASH

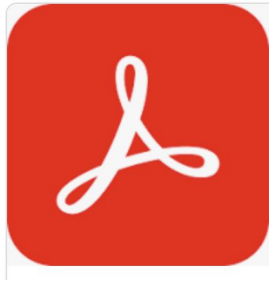

Disney+


Brave
Web browser


Bloomberg


Spotify®

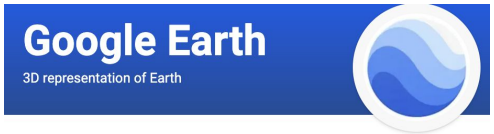

Adobe



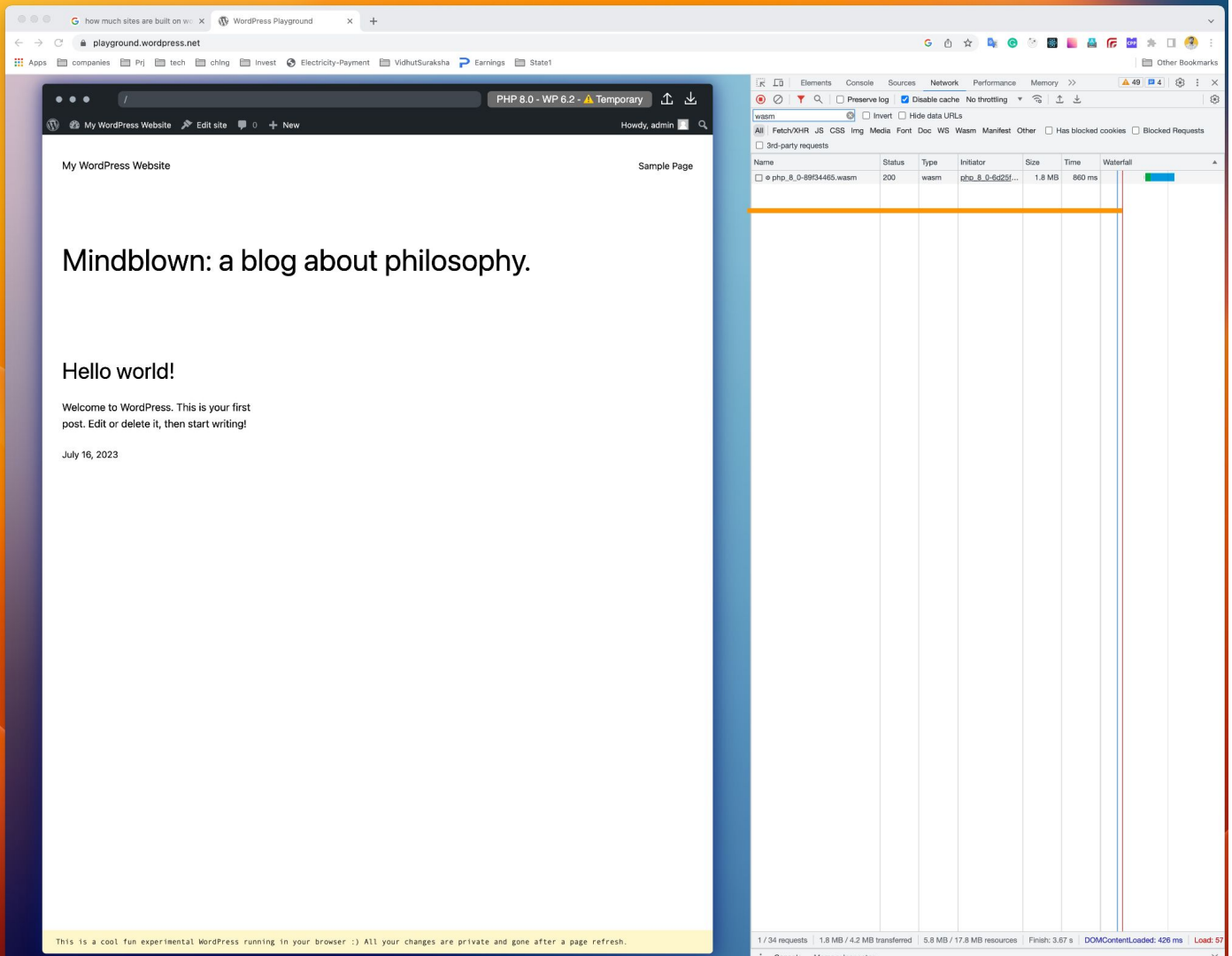

Deno


BBC

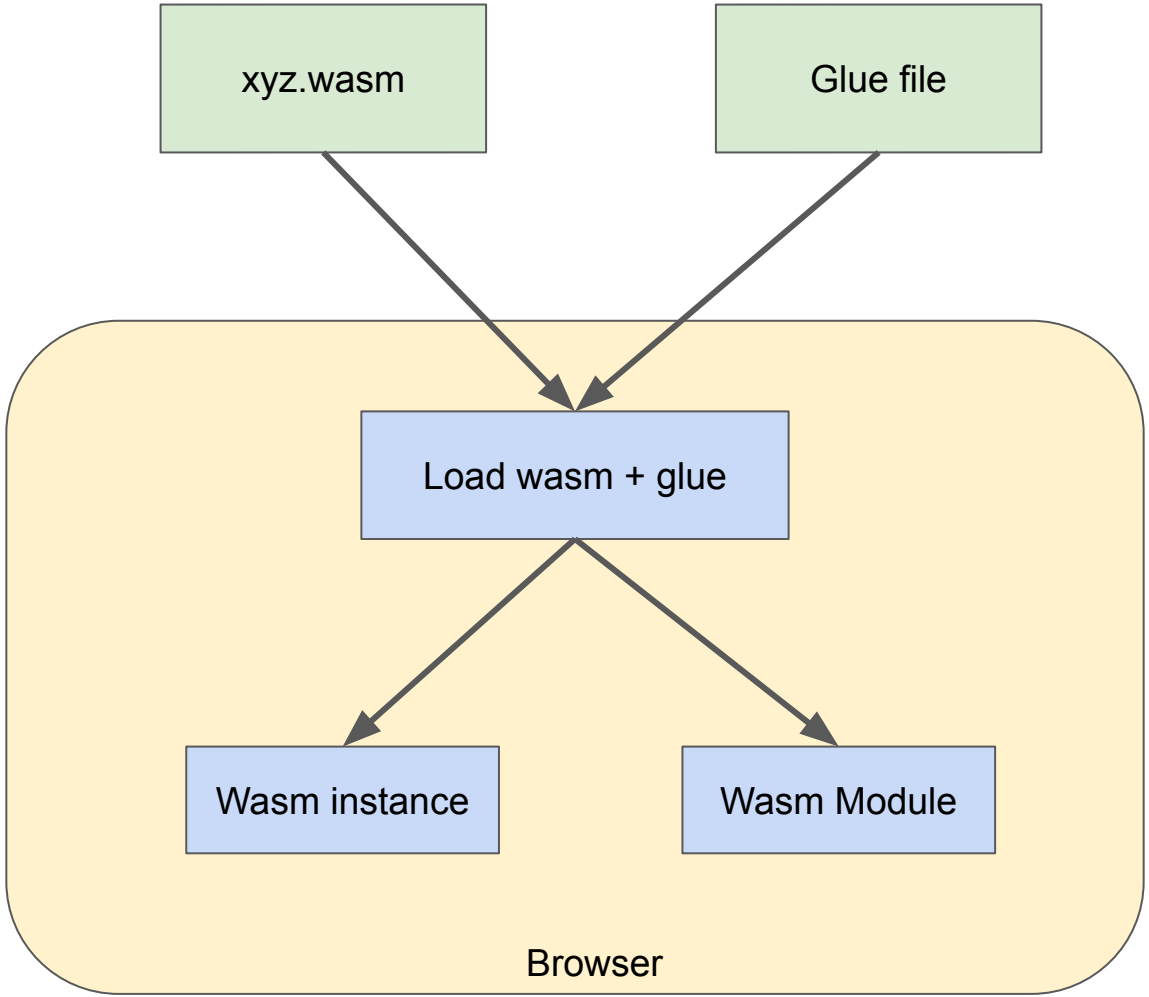

Google Meet
Video-communication app

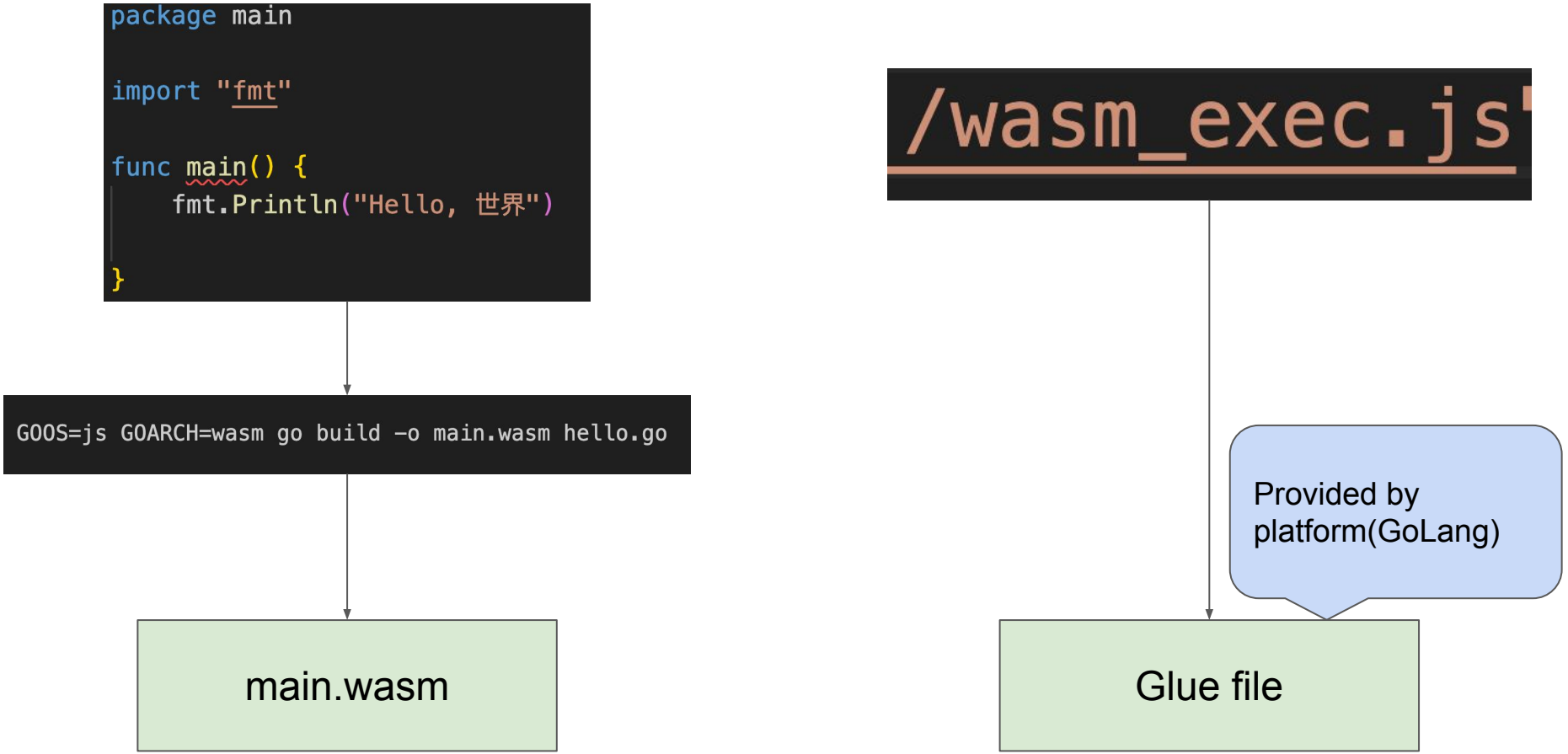

Google Earth
3D representation of Earth


COSMOS


how much sites are built on wo
WordPress Playground
playground.wordpress.net
Apps
companies
Prj
tech
chlng
Invest
Electricity-Payment
VidhutSuraksha
Earnings
State1
Other Bookmarks
/
PHP 8.0 - WP 6.2 - Temporary
My WordPress Website
Edit site
0
New
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.
Elements
Console
Sources
Network
Performance
Memory
Preserve log
Disable cache
No throttling
wasm
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
Name
Status
Type
Initiator
Size
Time
Waterfall
php_8_0-89f34465.wasm
200
wasm
1.8 MB
860 ms
1 / 34 requests
1.8 MB / 4.2 MB transferred
5.8 MB / 17.8 MB resources
Finish: 3.67 s
DOMContentLoaded: 426 ms
Load: 57

How to use web assembly?


xyz.wasm
Glue file
Load wasm + glue
Wasm instance
Wasm Module
Browser


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 × | thumbnail.js | thumbnail-withthread.html | thumbnail-withworker.html

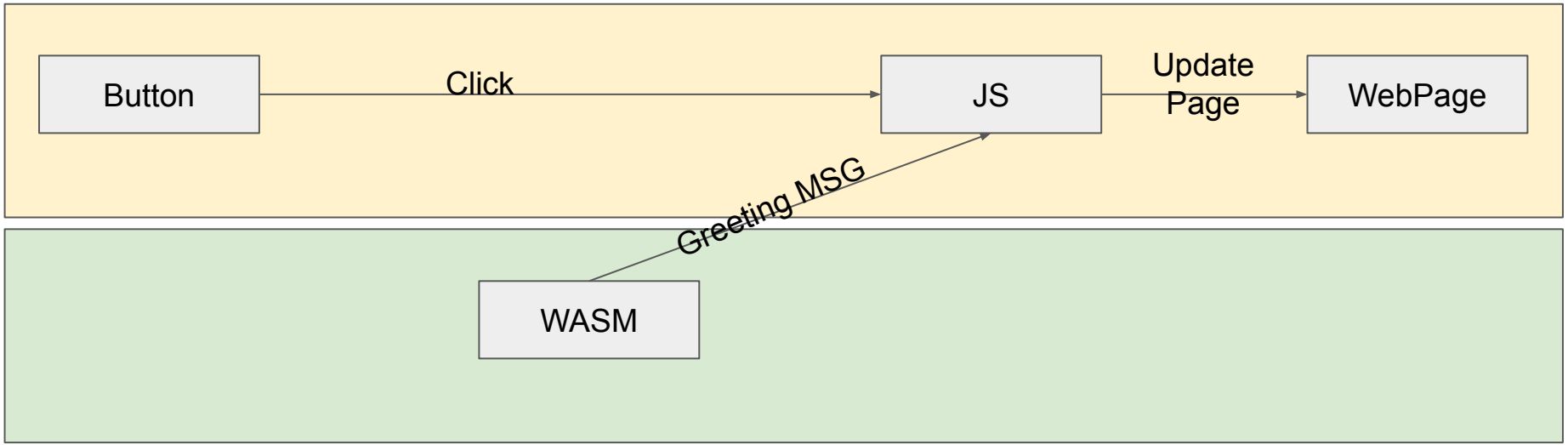
helloworld > helloworld.html > html > head > script

```
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

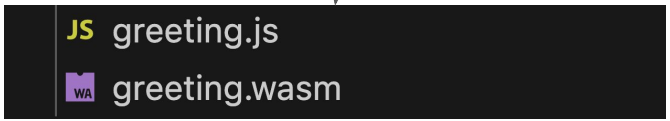
One more a bit interactive example

- Call a WASM function from JavaScript

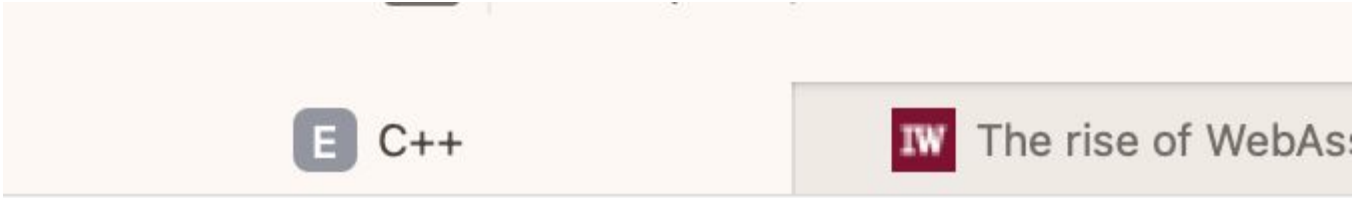


```cpp
#include <iostream>
#include <emscripten.h>

extern "C" {
    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;
}
}


int main(){
    std::cout << "Hello, webassembly!" << std::endl;
    char* s = get_string(1);
    std::cout << s << std::endl;
    return 0;
}
```

```
→  ~ emcc -o greeting.js greeting.cpp
```




```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>C++</title>
    <script src="./greeting.js"></script>
</head>
<body>
    <h1></h1>
    <button id="greeting">Generate Greeting Message</button>
    <button id="main">Run main Function</button>

    <script>
        const greetingButton = document.querySelector('#greeting');
        let count = 0;
        greetingButton.addEventListener('click', () => {
            const h1 = document.querySelector('h1');
            const js_wrapper = Module.cwrap('get_string', 'string', ['int']);
            const greeting = () => js_wrapper(count % 3);
            count++;
            h1.textContent = greeting();
        });

        const mainButton = document.querySelector('#main');
        mainButton.addEventListener('click', () => {
            const js_wrapper = Module.cwrap('main', 'number', []);
            js_wrapper();
        });

    </script>
</body>
</html>
```

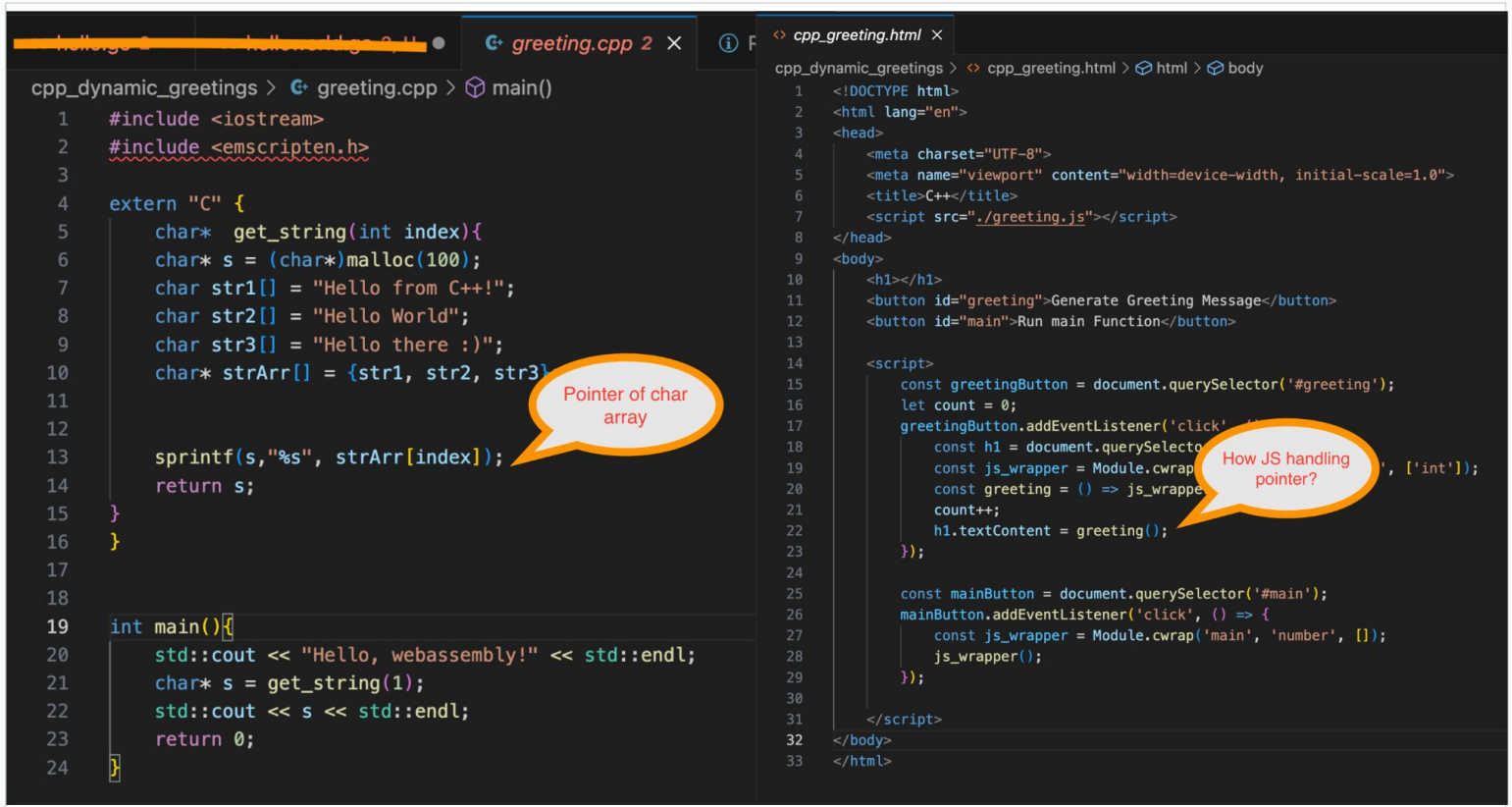


Hello from C++!

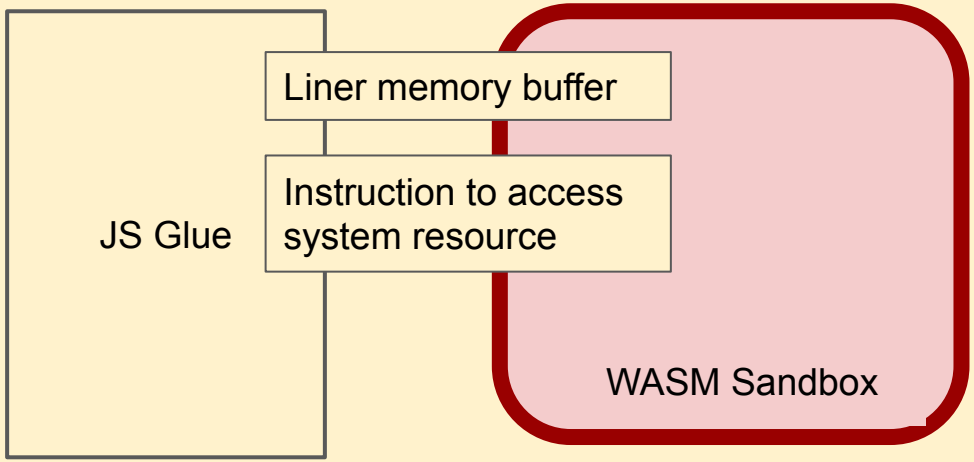
Generate Greeting Message | Run main Function

https://bit.ly/wasm-e1

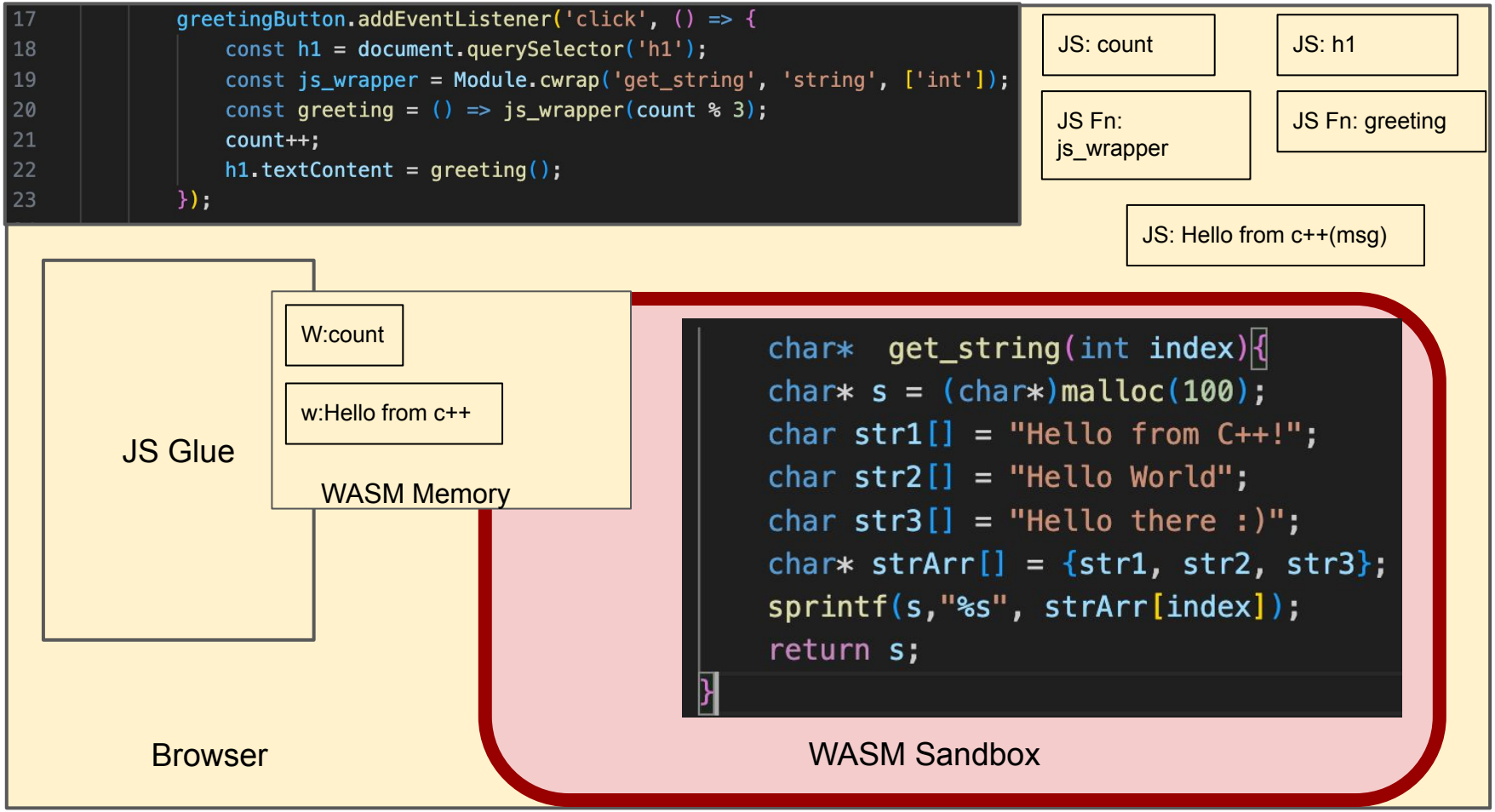
How type conversion is working?



JavaScript execution environment with memory and Global objects and APIs provided by browser



Browser


greetingButton.addEventListener('click', () => {
const h1 = document.querySelector('h1');
const js_wrapper = Module.cwrap('get_string', 'string', ['int']);
const greeting = () => js_wrapper(count % 3);
count++;
h1.textContent = greeting();
});
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
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;
}
Browser
WASM Sandbox

Example Thumbnail Creation

https://bit.ly/vp-wasm

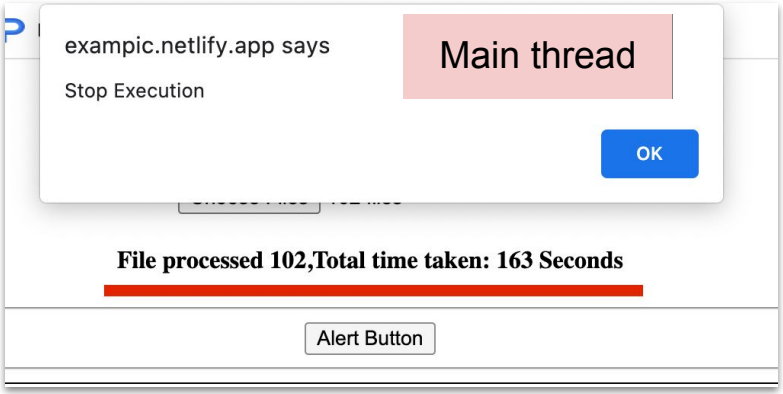
Example Thumbnail Creation With Web Worker

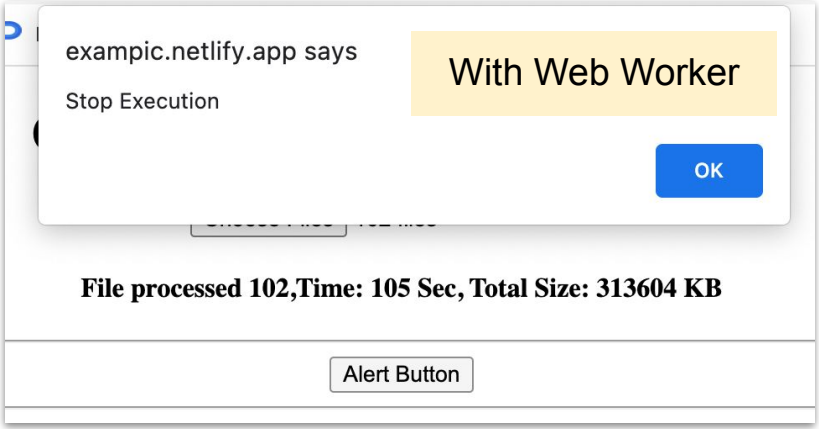
https://bit.ly/vp-wasm

Example Thumbnail Creation with Threads

https://bit.ly/vp-wasm

Example comparison





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.

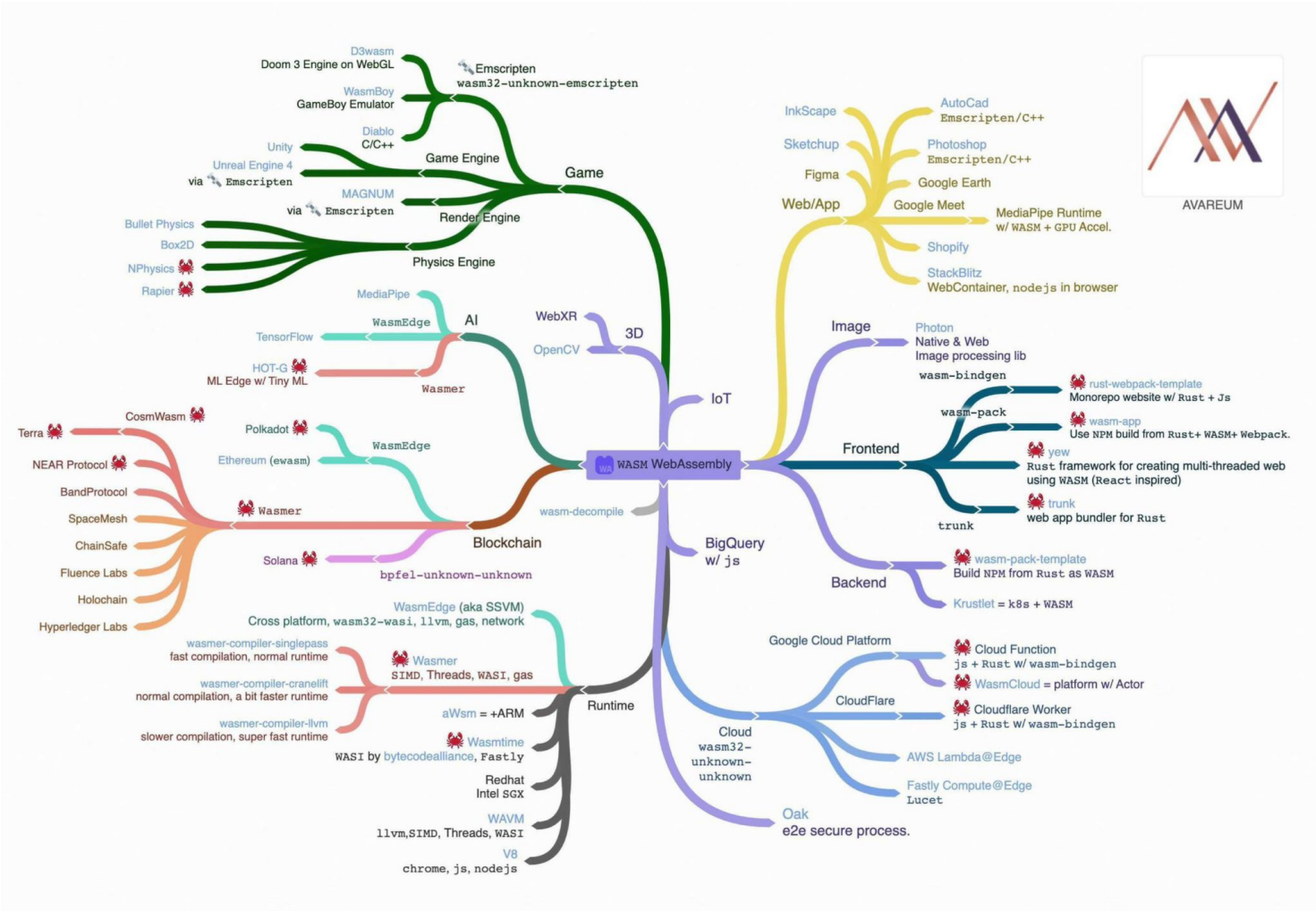
Source:https://web.dev/webassembly-threads/

example.c:

```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
```

WASM outside the browser env.


WASM WebAssembly
Game
Emscripten
wasm32-unknown-emscripten
D3wasm
Doom 3 Engine on WebGL
WasmBoy
GameBoy Emulator
Diablo
C/C++
Game Engine
Unity
Unreal Engine 4
via Emscripten
Render Engine
MAGNUM
via Emscripten
Physics Engine
Bullet Physics
Box2D
NPhysics
Rapier
3D
WebXR
OpenCV
AI
MediaPipe
WasmEdge
TensorFlow
HOT-G
ML Edge w/ Tiny ML
Wasmer
IoT
Blockchain
CosmWasm
Terra
NEAR Protocol
BandProtocol
SpaceMesh
ChainSafe
Fluence Labs
Holochain
Hyperledger Labs
Polkadot
Ethereum (ewasm)
WasmEdge
Wasmer
Solana
bpfel-unknown-unknown
wasm-decompile
Runtime
WasmEdge (aka SSVM)
Cross platform, wasm32-wasi, llvm, gas, network
Wasmer
SIMD, Threads, WASI, gas
wasmer-compiler-singlepass
fast compilation, normal runtime
wasmer-compiler-cranelift
normal compilation, a bit faster runtime
wasmer-compiler-llvm
slower compilation, super fast runtime
aWsm = +ARM
Wasmtime
WASI by bytecodealliance, Fastly
Redhat
Intel SGX
WAVM
llvm,SIMD, Threads, WASI
V8
chrome, js, nodejs
BigQuery
w/ js
Cloud
wasm32-
unknown-
unknown
Google Cloud Platform
Cloud Function
js + Rust w/ wasm-bindgen
WasmCloud = platform w/ Actor
CloudFlare
Cloudflare Worker
js + Rust w/ wasm-bindgen
AWS Lambda@Edge
Fastly Compute@Edge
Lucet
Oak
e2e secure process.
Web/App
InkScape
Sketchup
Figma
AutoCad
Emscripten/C++
Photoshop
Emscripten/C++
Google Earth
Google Meet
MediaPipe Runtime
w/ WASM + GPU Accel.
Shopify
StackBlitz
WebContainer, nodejs in browser
Image
Photon
Native & Web
Image processing lib
Frontend
wasm-bindgen
wasm-pack
trunk
rust-webpack-template
Monorepo website w/ Rust + Js
wasm-app
Use NPM build from Rust+ WASM+ Webpack.
yew
Rust framework for creating multi-threaded web
using WASM (React inspired)
trunk
web app bundler for Rust
Backend
wasm-pack-template
Build NPM from Rust as WASM
Krustlet = k8s + WASM
AVAREUM

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...

https://twitter.com/solomonstre/status/1111004913222324225?s=20&t=PCg2X4O9D_96xLSlqUzSRw

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.

Source: https://chat.openai.com/share/4c741cb4-a23f-46ea-9fca-6c99e789a8c9

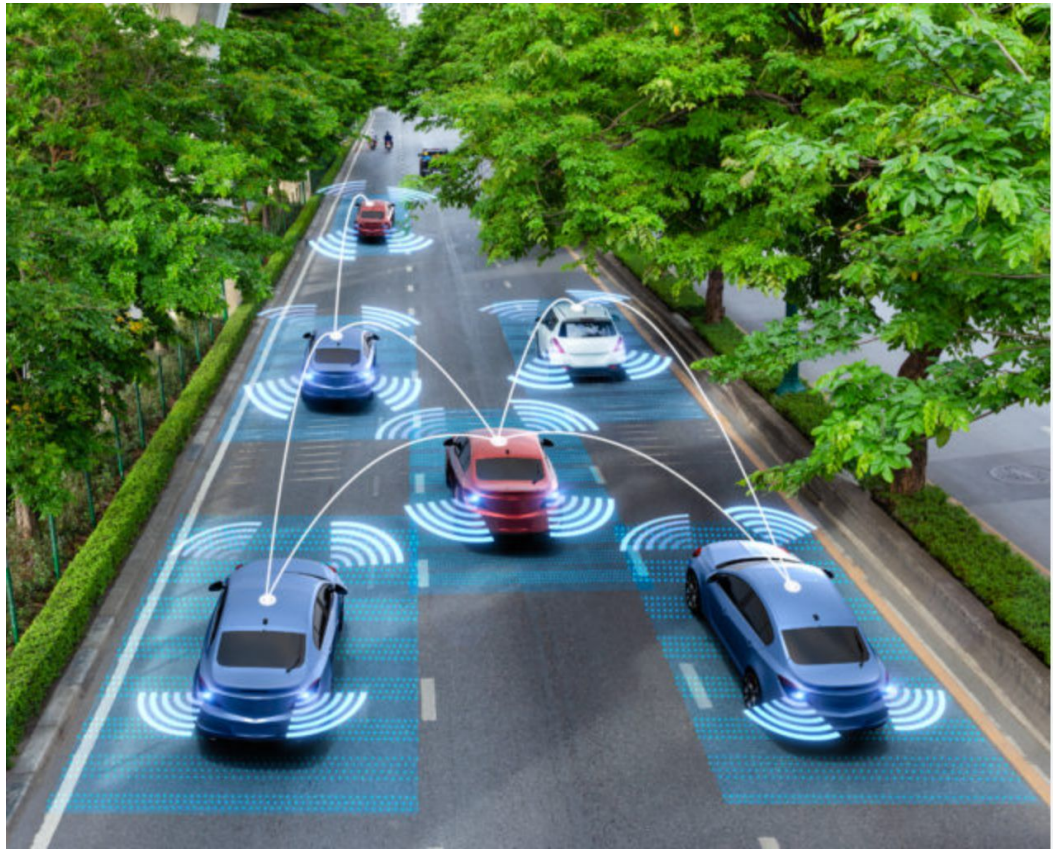
TODO: Prepare WASI examples

Use cases for using WASM?

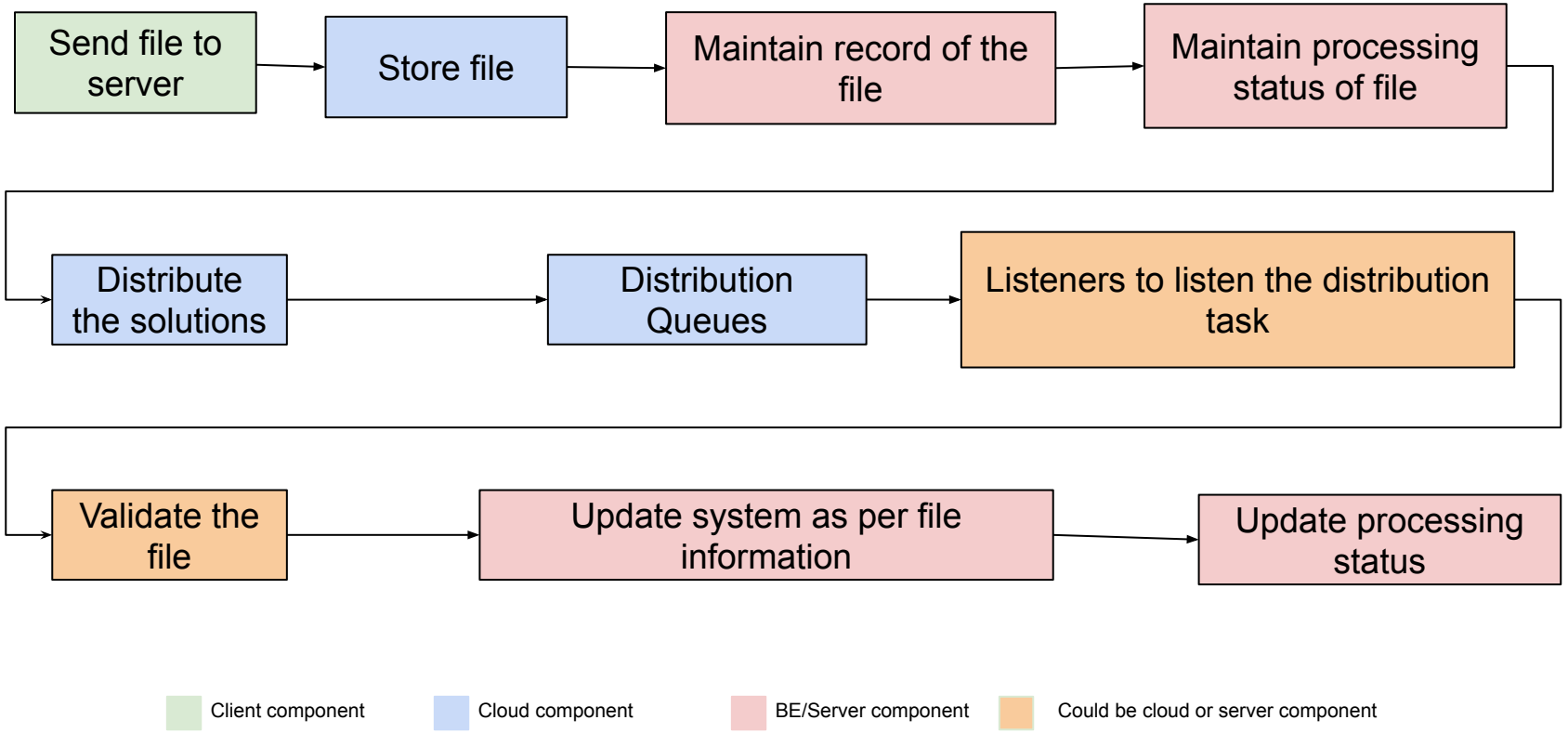
Edge Computing

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.

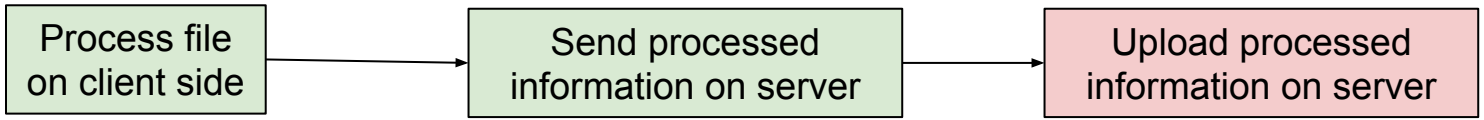

IOT



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

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
- https://chat.openai.com/share/71d4c1ad-3580-4d50-9f64-bb501da6a0d6