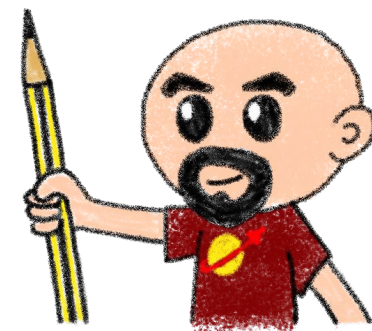


Server-side Web Assembly

Horacio Gonzalez

2021-06-24





Who are we?

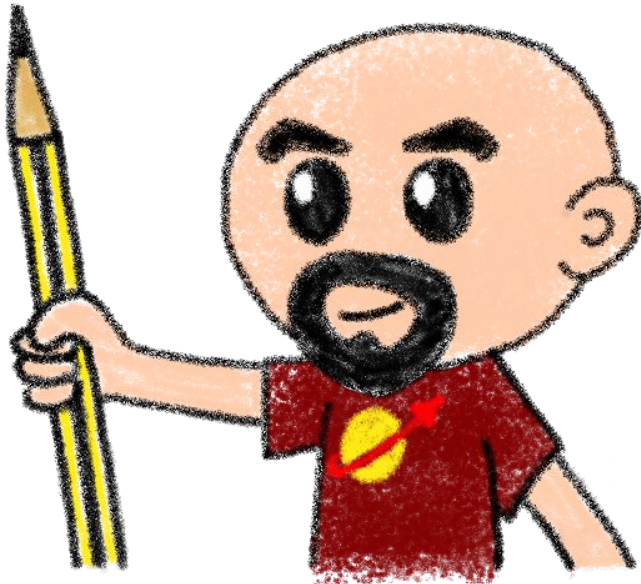
Introducing myself and
introducing ~~OVH~~ OVHcloud



Horacio Gonzalez

@LostInBrittany

Spaniard lost in Brittany,
developer, dreamer and
all-around geek



OVHcloud: A global leader



Web Cloud & Telecom



Private Cloud



Public Cloud



Storage



Network & Security



30 Data Centers
in 12 locations



34 Points of Presence
on a 20 TBPS Bandwidth Network



2200 Employees
worldwide



115K Private Cloud
VMS running



300K Public Cloud
instances running



380K Physical Servers
running in our data centers



1 Million+ Servers
produced since 1999



1.5 Million Customers
across 132 countries



3.8 Million Websites
hosting



1.5 Billion Euros Invested
since 2016



P.U.E. 1.09
Energy efficiency indicator



20+ Years in Business
Disrupting since 1999





Did I say WebAssembly?

Wasm for friends...



WebAssembly, what's that?



What's WASM?



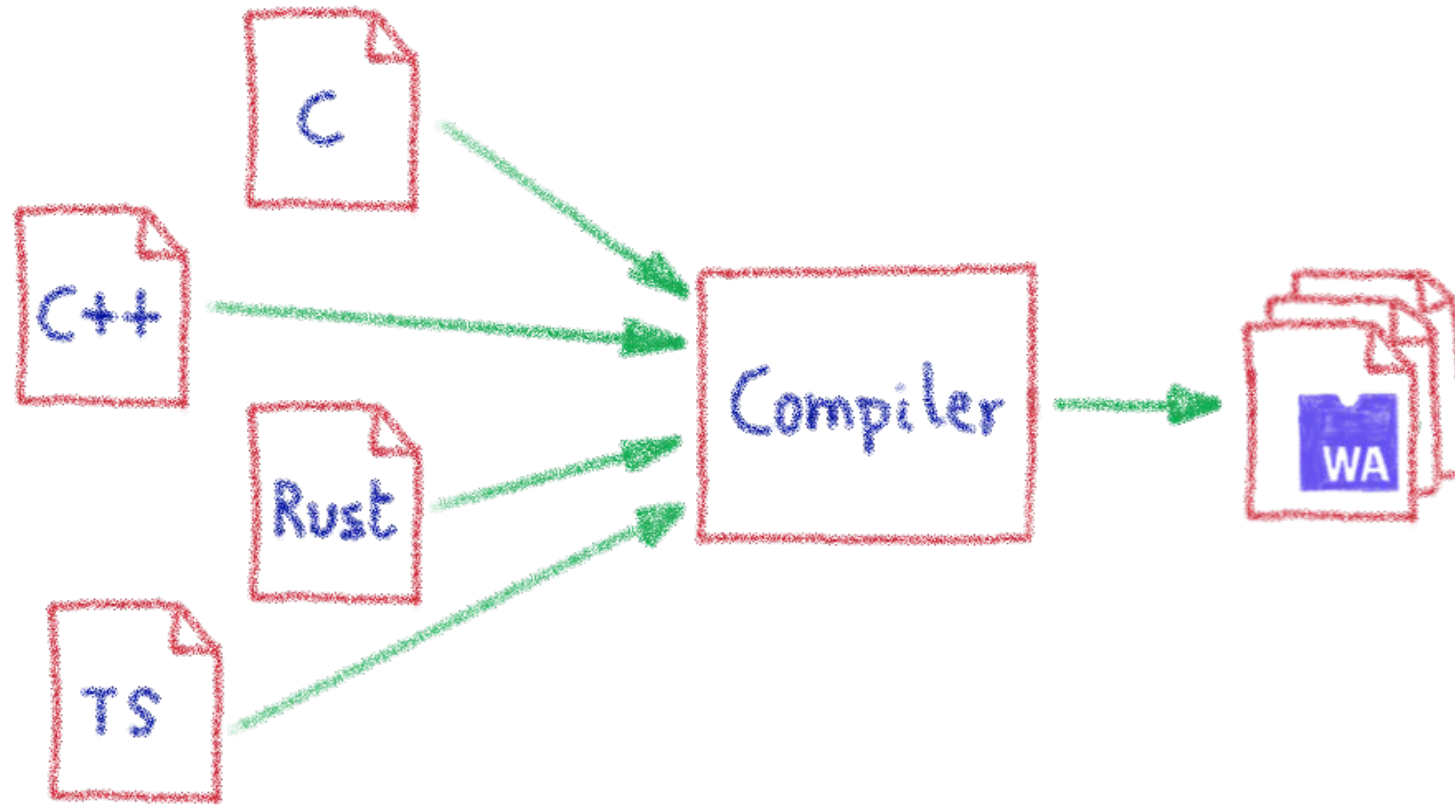
Does it replace JS?

Can I code webapps in Rust?

Is HTML/CSS/JS stack obsolete?



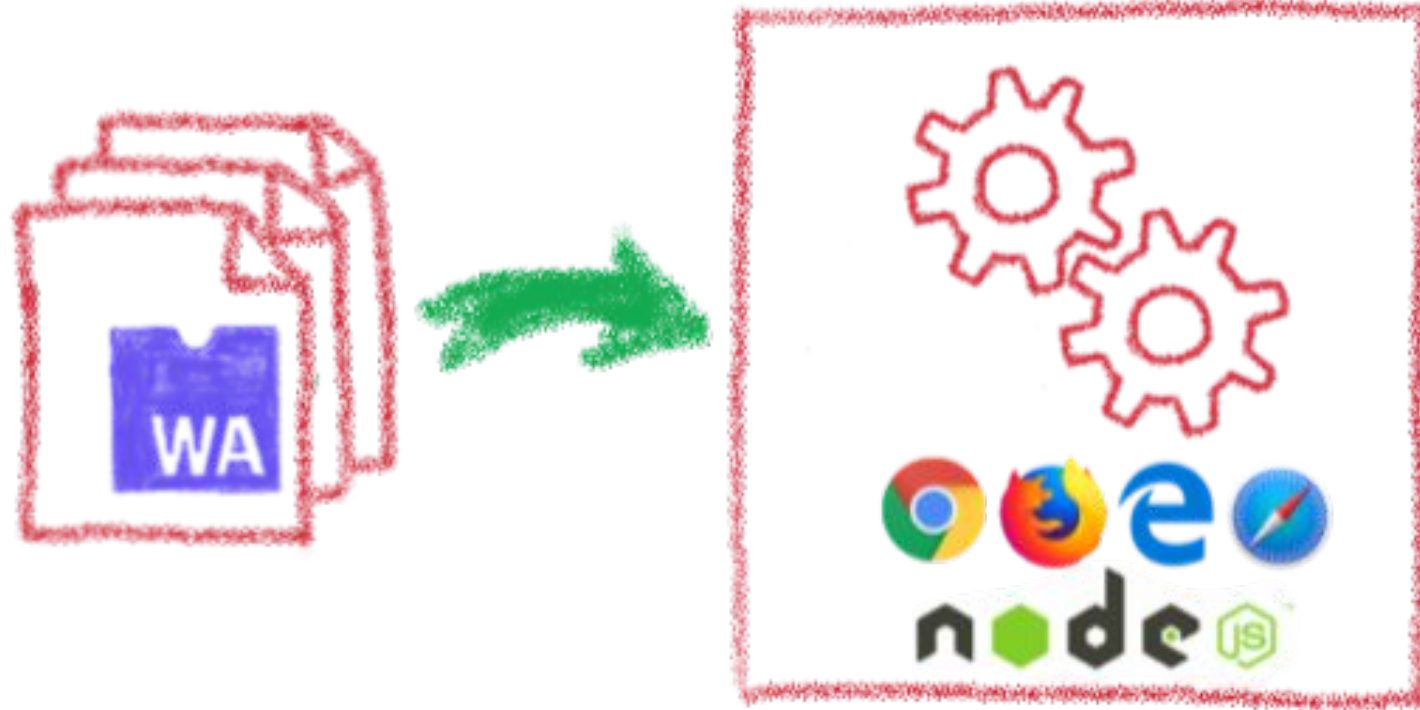
A low-level binary format



Not a programming language, a compilation target



That runs on a stack-based virtual machine



A portable binary format that runs on all modern browsers...
but also on NodeJS and elsewhere!!



With several key advantages



Fast & Efficient ⚡

🔒 Memory-safe & Sandboxed

Open & Debuggable 📄

WWW Part of the Web Platform



But above all...



Wasm is not meant to replace JavaScript



Who is using WebAssembly today?



And many more others...





A bit of history

Remembering the past
to better understand the present



Executing other languages in the browser

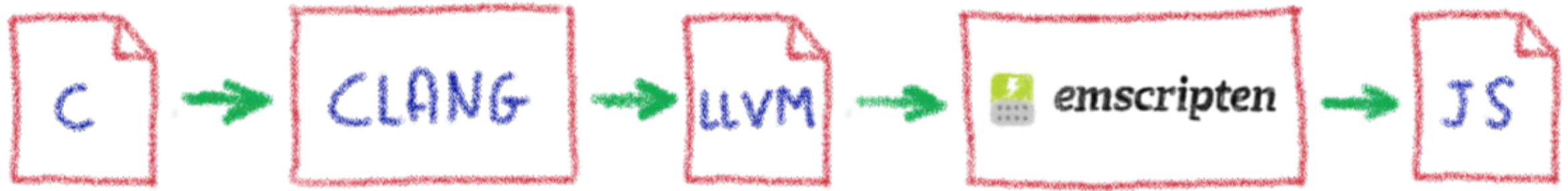


MACROMEDIA
FLASH

A long story, with many failures...



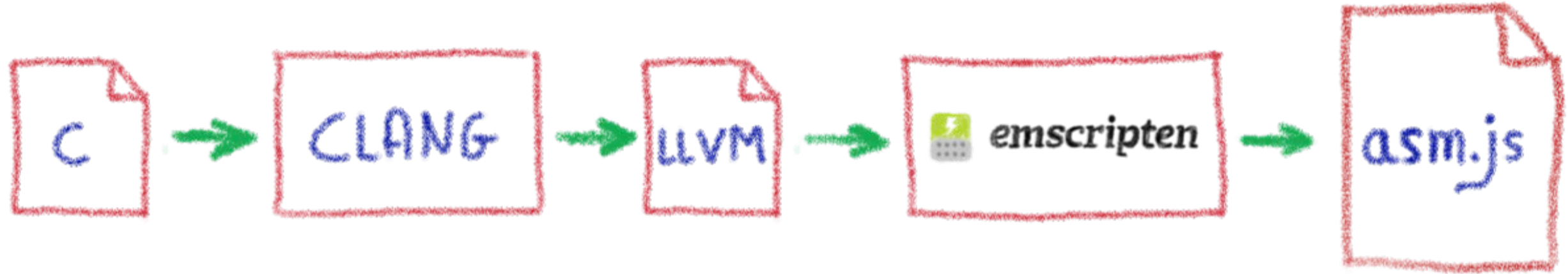
2012 - From C to JS: enter emscripten



Passing by LLVM pivot



2013 - Generated JS is slow...



Let's use only a strict subset of JS: asm.js
Only features adapted to AOT optimization



WebAssembly project



moz://a

Google



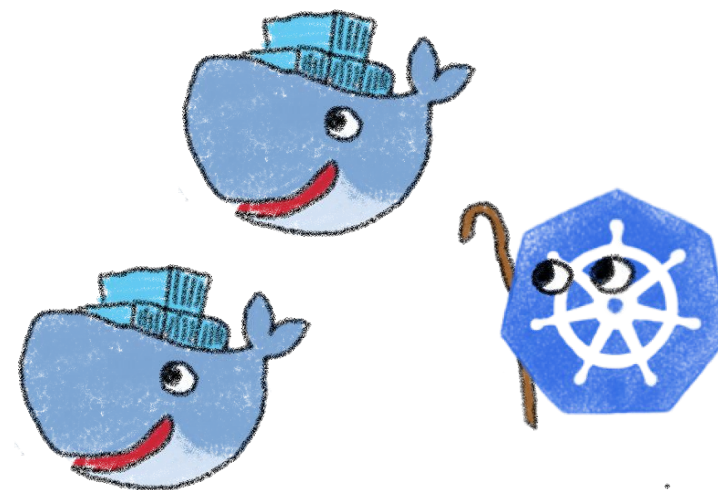
W3C

Joint effort



Server-side WebAssembly

Too good to not to use it



Solomon on Web Assembly



Solomon Hykes
@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

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://hacks.mozilla.org/2019/03/standards-for-webassembly/)

9:39 PM · Mar 27, 2019



1.9K



792 people are Tweeting about this



Solomon on Web Assembly



Solomon Hykes
@solomonstre



"So will wasm replace Docker?" No, but imagine a future where Docker runs linux containers, windows containers and wasm containers side by side. Over time wasm might become the most popular container type. Docker will love them all equally, and run it all :)



Solomon Hykes @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! [twitter.com/linclark/statu...](https://twitter.com/linclark/status/1118111111)

4:50 AM · Mar 28, 2019



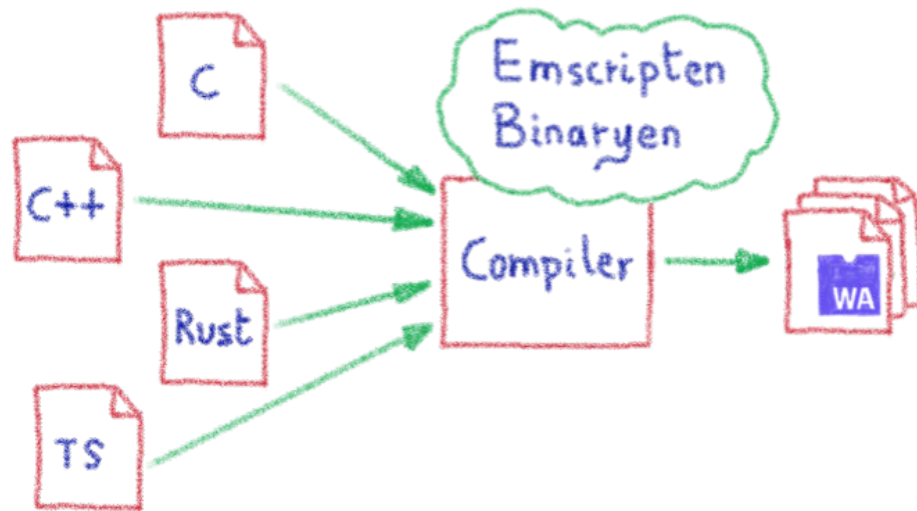
142



52 people are Tweeting about this



A very interesting feature set



Fast & Efficient ⚡

🔒 Memory-safe & Sandboxed

Open & Debuggable 📄

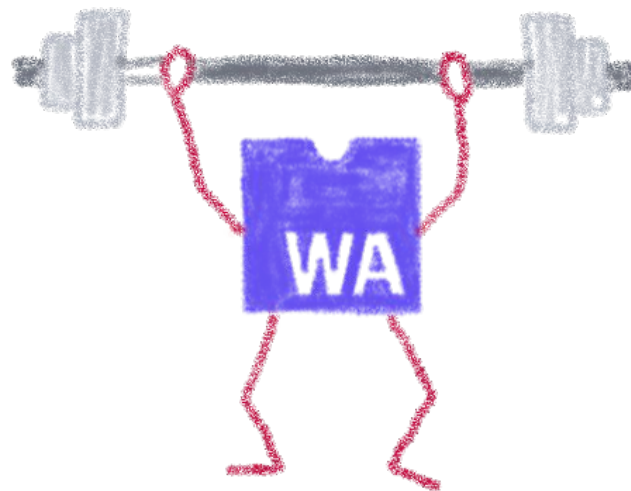
WWW Part of the Web Platform

WHY ONLY FOR THE WEB ??

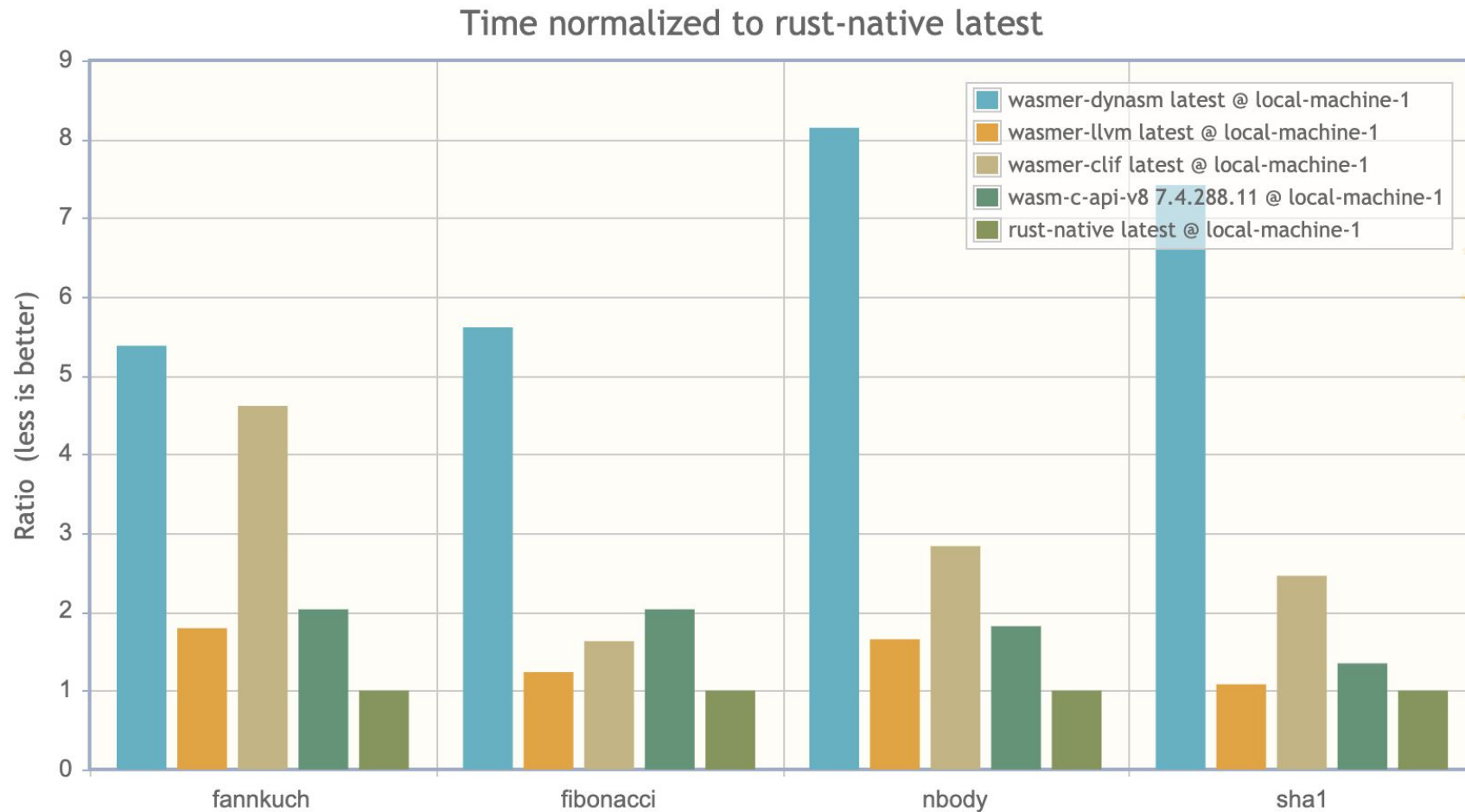


Features of Wasm

Why is everybody looking at it?



Near native speed



<https://medium.com/wasmer/benchmarking-webassembly-runtimes-18497ce0d76e>



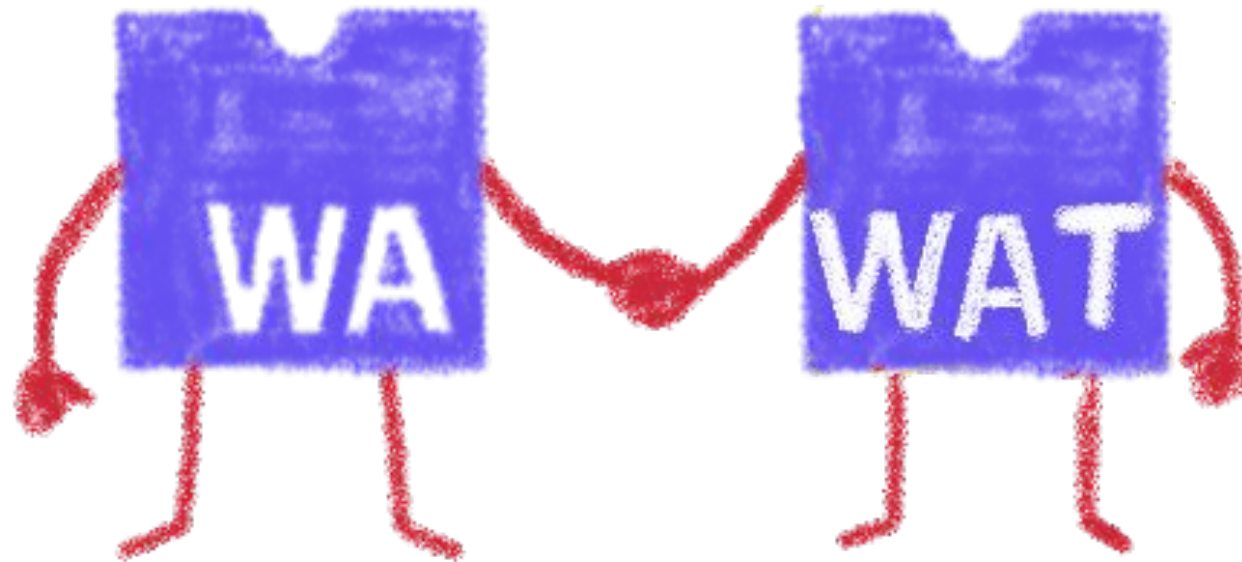
Highly portable



It can be run almost everywhere...



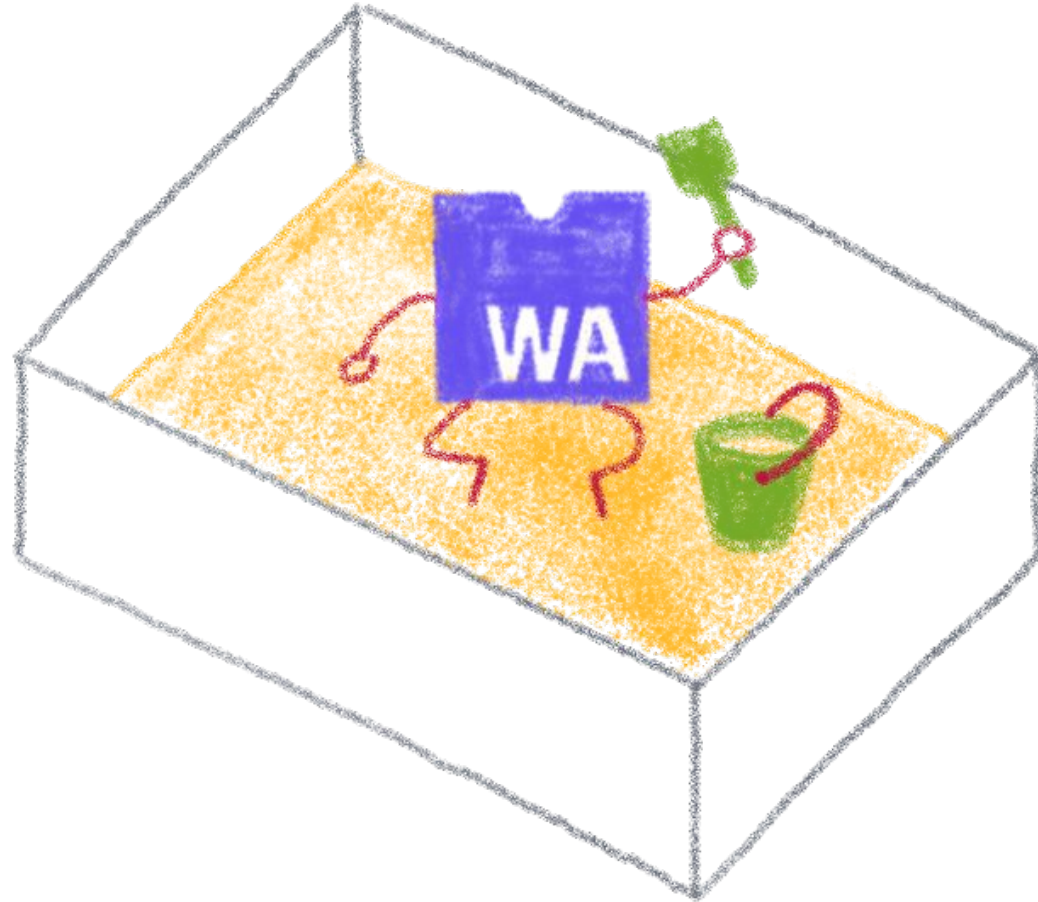
Readable and debuggable



Each `.wasm` file with its `.wat` companion file



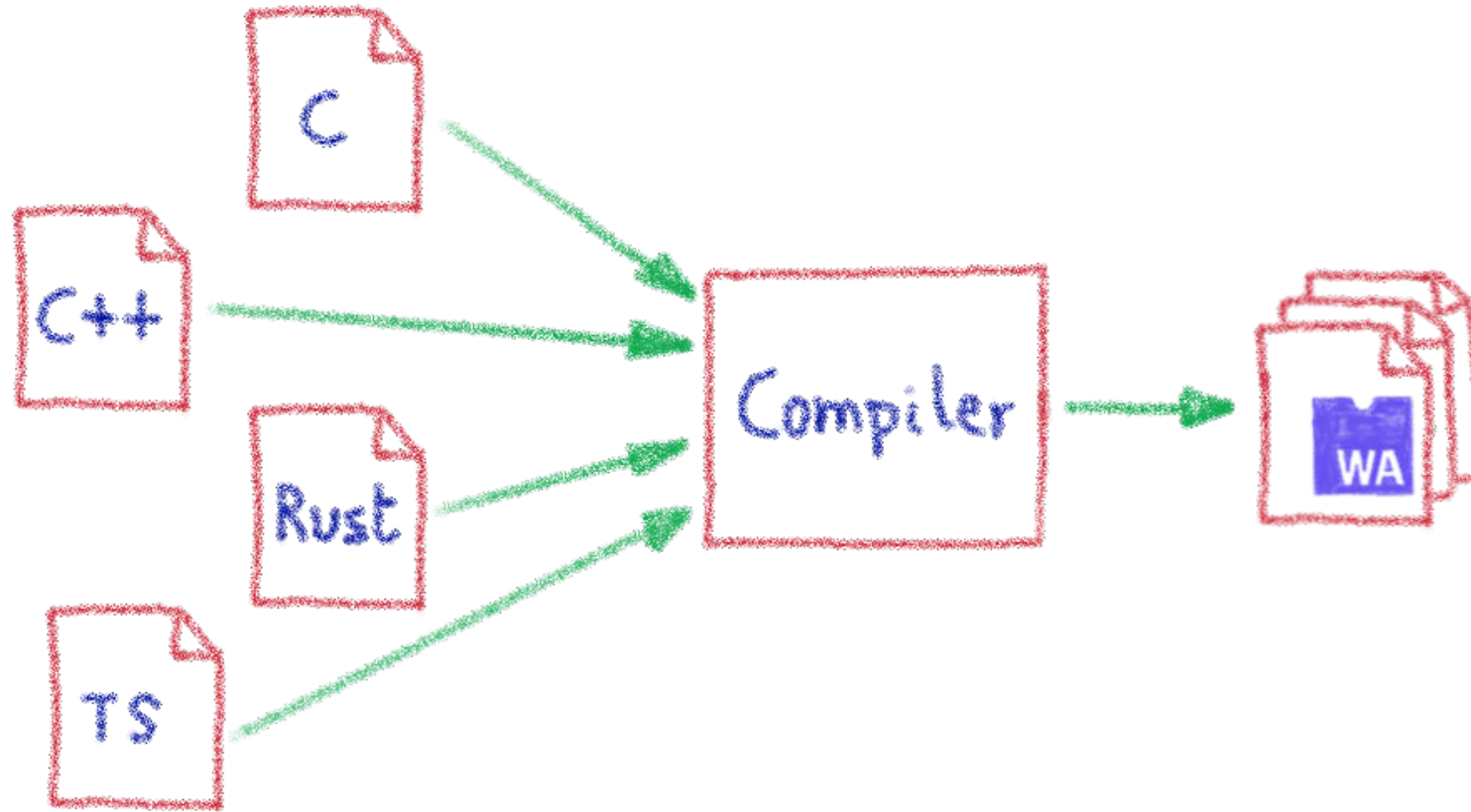
Memory safe & secure



Running in a fully sandboxed environment



Accepting many source languages



And more and more...



Still a young platform...

But growing up fast!



Native WASM types are limited



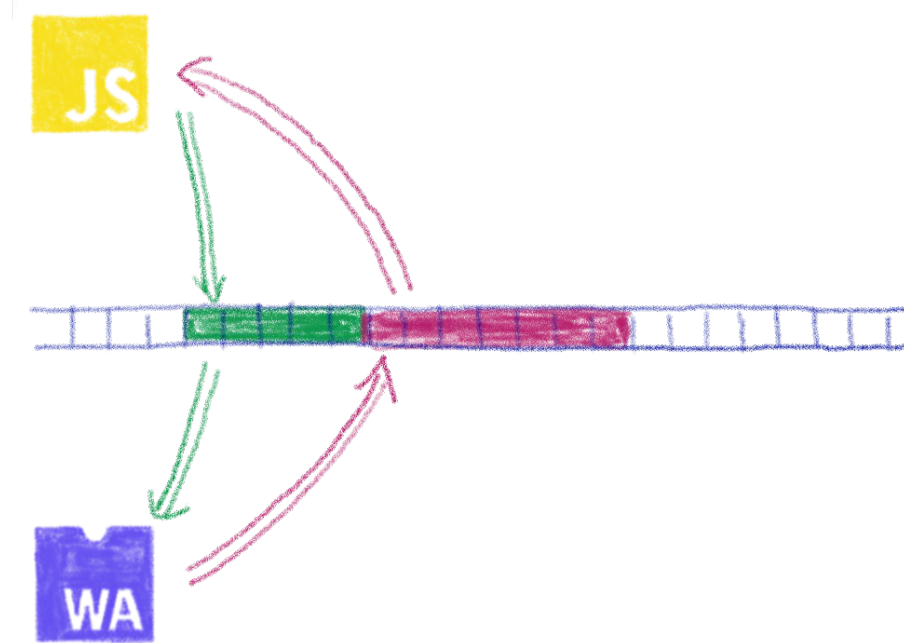
WASM currently has four available types:

- **i32**: 32-bit integer
- **i64**: 64-bit integer
- **f32**: 32-bit float
- **f64**: 64-bit float

Types from languages compiled to WASM are mapped to these types



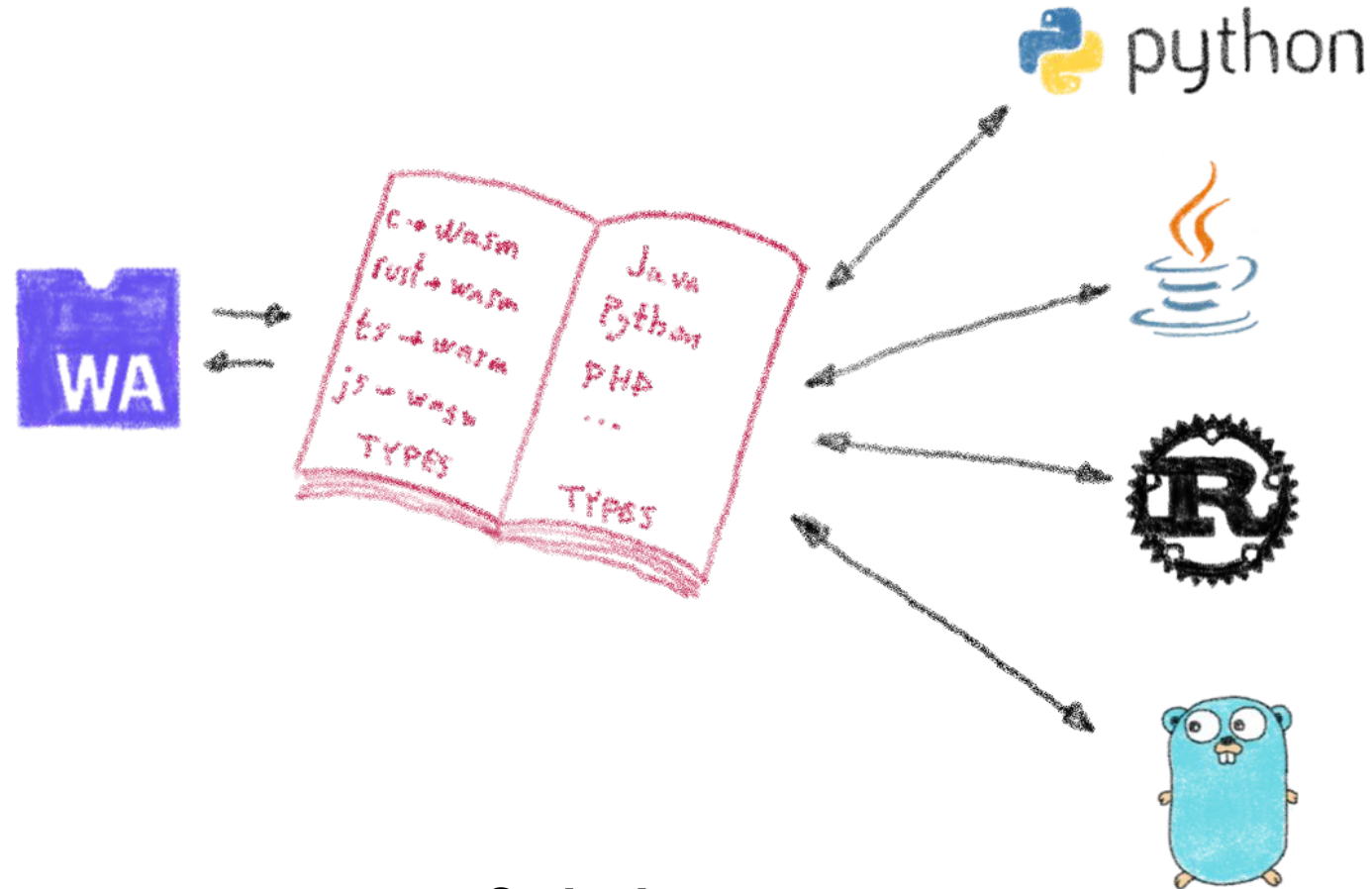
How can we share data?



Using the same data in WASM and JS?
Shared linear memory between them,
and serializing the data to one Wasm types



Solution is coming: Interface types



Beautiful description at:

<https://hacks.mozilla.org/2019/08/webassembly-interface-types>



No outside access



By design, communication is done
using the shared linear memory only



Solution exists: WASI



 WASI 

WASI

The WebAssembly System Interface

WASI is a modular system interface for WebAssembly. As described in [the initial announcement](#), it's focused on security and portability.

WASI is being standardized in [a subgroup of the WebAssembly CG](#). Discussions happen in [GitHub issues](#), [pull requests](#), and [bi-weekly Zoom meetings](#).

For a quick intro to WASI, including getting started using it, see [the intro document](#).

The Wasmtime runtime's [tutorial](#) contains [examples](#) for how to target WASI from [C](#) and [Rust](#). The resulting .wasm modules can be run in any WASI-compliant runtime.

For more documentation, see [the documents guide](#).

Already available

in  Wasmer



Mono-thread and scalar operations only



Multiple scalar
operations

$$\begin{array}{l} \boxed{A1} + \boxed{B1} = \boxed{C1} \\ \boxed{A2} + \boxed{B2} = \boxed{C2} \\ \boxed{A3} + \boxed{B3} = \boxed{C3} \end{array}$$

Not the most efficient way...



Solution exists: SIMD



Multiple scalar
operations

$$\boxed{A1} + \boxed{B1} = \boxed{C1}$$

$$\boxed{A2} + \boxed{B2} = \boxed{C2}$$

$$\boxed{A3} + \boxed{B3} = \boxed{C3}$$

Single vectorial
operation

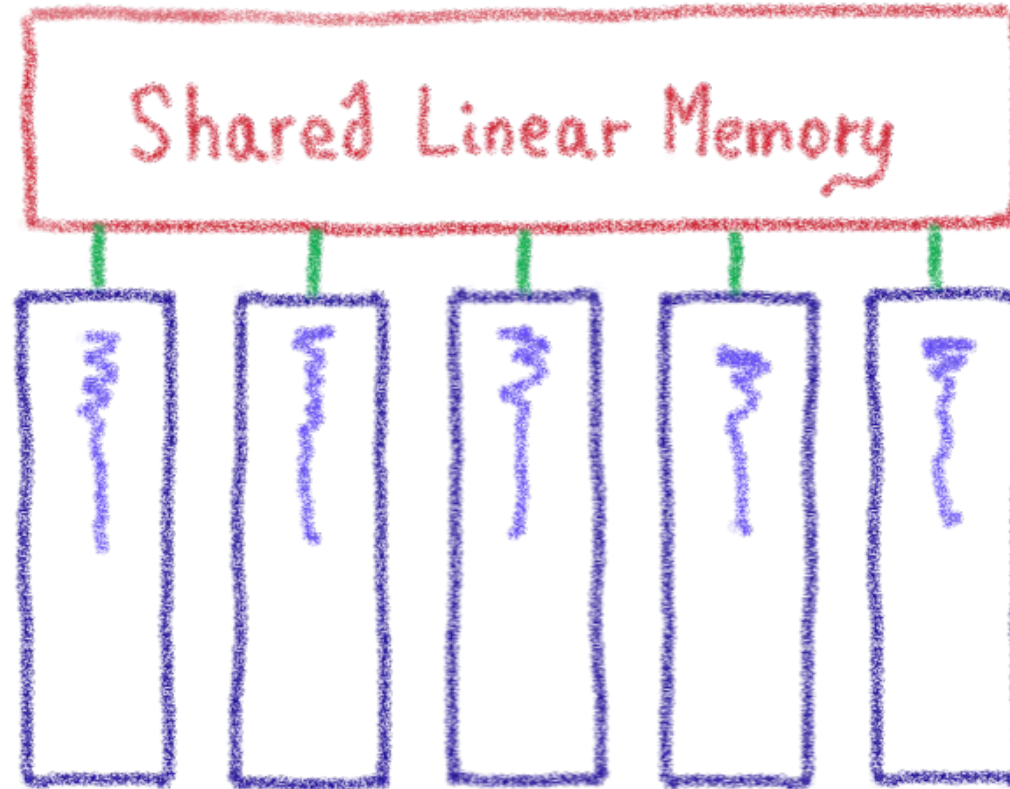
$$\begin{bmatrix} A1 \\ A2 \\ A3 \end{bmatrix} + \begin{bmatrix} B1 \\ B2 \\ B3 \end{bmatrix} = \begin{bmatrix} C1 \\ C2 \\ C3 \end{bmatrix}$$

Single Instruction, Multiple Data

Already available
in  Wasmer



Solutions are coming too: Wasm Threads



Threads on Web Workers
with shared linear memory



Incoming proposals: Garbage collector



And exception handling





The Bytecode Alliance

Taking WASM out of the browser



The Bytecode Alliance



**BYTECODE
ALLIANCE**



moz://a

fastly



Red Hat



Bytecode Alliance projects



Wasmtime

Cratelif

WebAssembly
Micro Runtime
(WAMR)

Lucet

Enarx



Wasmtime



wasmtime

A standalone runtime for [WebAssembly](#)

A [Bytecode Alliance](#) project

 CI  passing  zulip  join chat  rustc  1.37+  docs  0.19.0

[Guide](#) | [Contributing](#) | [Website](#) | [Chat](#)

Installation

The Wasmtime CLI can be installed on Linux and macOS with a small install script:

```
$ curl https://wasmtime.dev/install.sh -sSf | bash
```

Windows or otherwise interested users can download installers and binaries directly from the [GitHub Releases](#) page.

Example

If you've got the [Rust compiler installed](#) then you can take some Rust source code:

```
fn main() {  
    println!("Hello, world!");  
}
```

and compile/run it with:

```
$ rustup target add wasm32-wasi  
$ rustc hello.rs --target wasm32-wasi  
$ wasmtime hello.wasm  
Hello, world!
```





Cranelift Code Generator

A [Bytecode Alliance](#) project

Cranelift is a low-level retargetable code generator. It translates a [target-independent intermediate representation](#) into executable machine code.



For more information, see [the documentation](#).

For an example of how to use the JIT, see the [SimpleJIT Demo](#), which implements a toy language.

For an example of how to use Cranelift to run WebAssembly code, see [Wasmtime](#), which implements a standalone, embeddable, VM using Cranelift.

Status

Cranelift currently supports enough functionality to run a wide variety of programs, including all the functionality needed to execute WebAssembly MVP functions, although it needs to be used within an external WebAssembly embedding to be part of a complete WebAssembly implementation.

The x86-64 backend is currently the most complete and stable; other architectures are in various stages of development. Cranelift currently supports both the System V AMD64 ABI calling convention used on many platforms and the Windows x64 calling convention. The performance of code produced by Cranelift is not yet impressive, though we have plans to fix that.

The core codegen crates have minimal dependencies, support `no_std` mode (see below), and do not require any host floating-point support, and do not use callstack recursion.



WebAssembly Micro Runtime (wamr)



WebAssembly Micro Runtime

[Build WAMR VM core](#) | [Embed WAMR](#) | [Export native function](#) | [Build WASM applications](#) | [Samples](#)

A [Bytecode Alliance](#) project

WebAssembly Micro Runtime (WAMR) is a standalone WebAssembly (WASM) runtime with a small footprint. It includes a few parts as below:

- The "iwasm" VM core, supporting WebAssembly interpreter, ahead of time compilation (AoT) and Just-in-Time compilation (JIT)
- The application framework and the supporting API's for the WASM applications
- The dynamic management of the WASM applications

iwasm VM core

key features

- 100% compliant to the W3C WASM MVP
- Small runtime binary size (85K for interpreter and 50K for AoT) and low memory usage
- Near to native speed by AoT
- Self-implemented module loader enables AoT working cross Linux, SGX and MCU systems
- Choices of WASM application libc support: the built-in libc subset for the embedded environment or [WASI](#) for standard libc
- [Embeddable with the supporting C API's](#)
- [The mechanism for exporting native API's to WASM applications](#)
- [Multiple modules as dependencies](#)
- [Thread management and pthread library](#)



Lucet



Lucet 

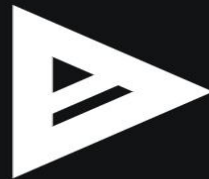
A [Bytecode Alliance](#) project

Lucet is a native WebAssembly compiler and runtime. It is designed to safely execute untrusted WebAssembly programs inside your application.

Check out our [announcement post on the Fastly blog](#).

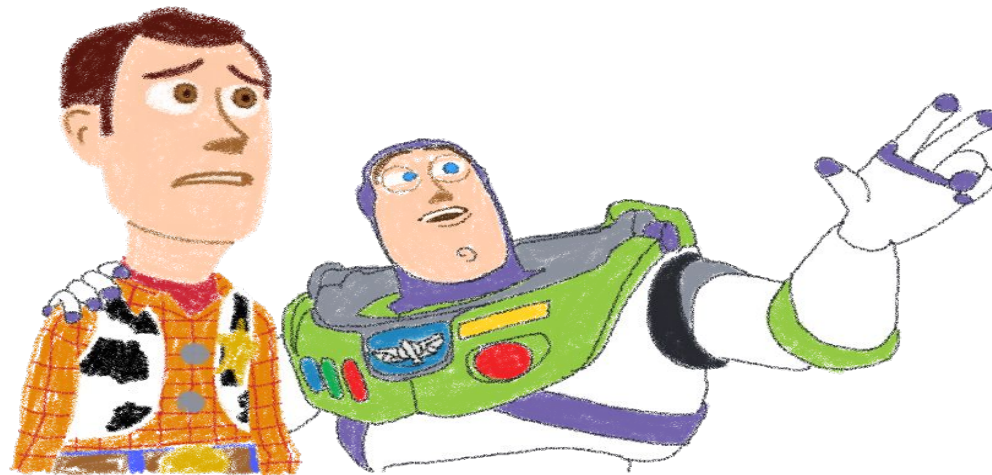
Lucet uses, and is developed in collaboration with, the Bytecode Alliance's [Cranelift](#) code generator. It powers Fastly's [Terrarium](#) platform.

```
(func $_start (type 7)
  (local i32 i32 i32 i32)
  global.get 0
  i32.const 16
  i32.sub
  local.tee 0
  global.set 0
  call $__wasilibc_init_preopen
  i32.const 3
  local.set 1
  block ;; label = @1
    block ;; label = @2
      block ;; label = @3
        block ;; label = @4
          loop ;; label = @5
            local.get 1
            local.get 0
```

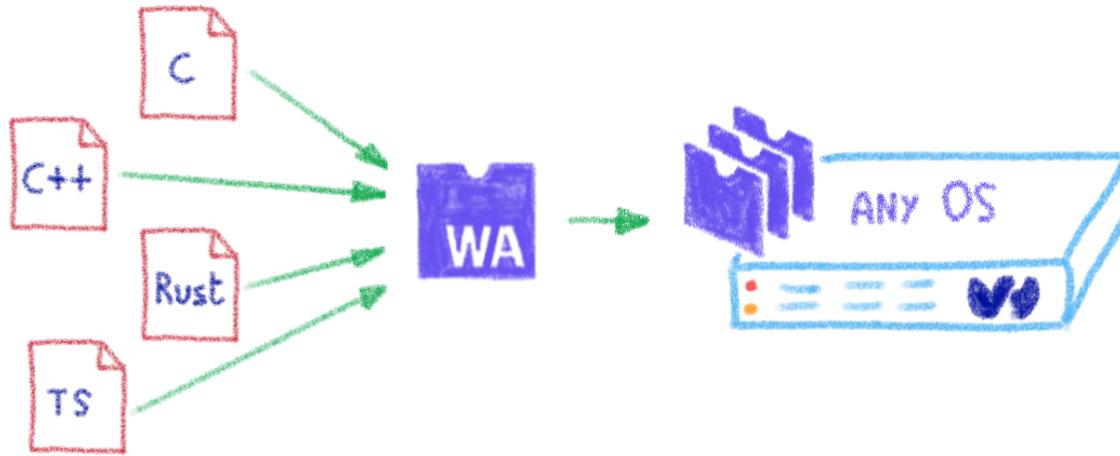


Other runtimes

Runtimes, runtimes everywhere



Wasmer



Run any code* on any client... almost

* Languages compiling to WASM



Wasmer



Run or embed anywhere super lightweight containers
based on Wasm

 FAST

 WASI

 Sandboxed

 SIMD

Multi platform



 Pluggable

LLVM Cranelift
Singlepass

 @wasmerio

 wasmerio/wasmer



Wasmer ecosystem



Web Assembly Package Manager



 WebAssembly.sh

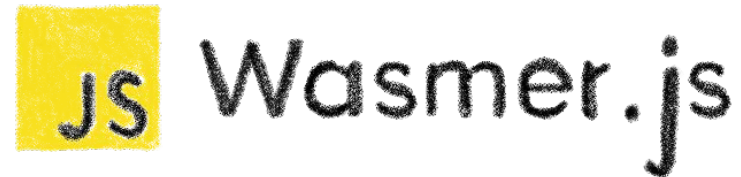
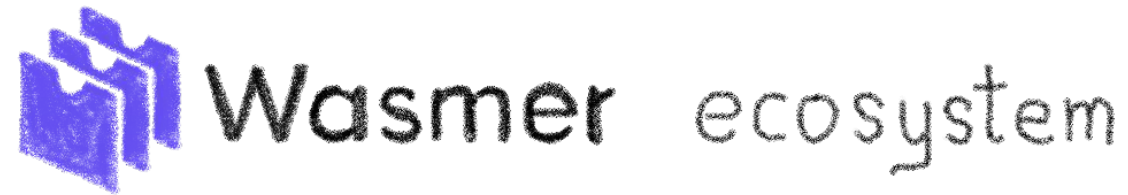
```
Install  wpm install cowsay
or
Execute  wpm cowsay Hello
```



Online WebAssembly shell
Execution done in browser



Wasmer ecosystem



Bringing languages to WASI



Wasm3

Wasm3

wasm v0.4.7 issues 30 tests failing license MIT

A high performance WebAssembly interpreter written in C.

- ~ 8x faster than other known wasm interpreters
- ~ 4-5x slower than state of the art wasm JIT engines
- ~ 12x slower than native execution

* Based on [CoreMark 1.0 benchmark](#). Your mileage may vary.

twitter 513 discord 6 online telegram chat

Status

wasm3 passes the [WebAssembly spec testsuite](#) and is able to run many WASI apps.

Minimum useful system requirements: ~64Kb for code and ~10Kb RAM

wasm3 runs on a wide range of architectures (x86 , x86_64 , ARM , RISC-V , PowerPC , MIPS , Xtensa , ARC32 , ...) and [platforms](#):

- Linux, Windows, OS X, FreeBSD
- Android, iOS
- OpenWRT-enabled routers
- Raspberry Pi, Orange Pi and other SBCs
- MCUs: Arduino, ESP8266, ESP32, Particle, ... [see full list](#)
- Browsers... yes, using WebAssembly itself!
- wasm3 can execute wasm3 (self-hosting)

Wasm3

```
Wasm3 v0.4.6 on iOS (arm64-v8a)
Build Feb  7 2020 21:48:53
Device: iPhone10,4

Loading WebAssembly...
Running fib(40) on WebAssembly...
Result: 102334155
Elapsed: 4844 ms

Running fib(40) on Native C...
Result: 102334155
Elapsed: 611 ms
```



Wasm3



The fastest WebAssembly interpreter



WASI



Multi platform



Wasm inception

Web Assembly
& WASI
self-hosting

 @wasm3_engine

 wasm3/wasm3



And even Wasm over GraalVM!

GraalVM.

ALL ARTICLES

LEARN MORE



Announcing GraalWasm — a WebAssembly engine in GraalVM



Aleksandar Prokopec

Follow

Dec 2, 2019 · 10 min read



We're happy to announce the initial public work on GraalWasm — the WebAssembly engine implemented in GraalVM. GraalWasm currently implements the WebAssembly MVP (Minimum Viable Product) specification, and can run WebAssembly programs in the binary format, generated with compiler backends such as Emscripten.

Supporting WebAssembly expands the set of languages GraalVM can execute with a whole other set of languages to the ones supported by GraalVM and is further step towards making it a universal platform for programming language execution. This feature was also highly requested by the GraalVM community and we are happy to share our first results.



GraalVM™



Or in Kubernetes...

Krustlet: Kubernetes Kubelet in Rust for running WASM

⚠️ ⚠️ This project is highly experimental. ⚠️ ⚠️ It should not be used in production workloads.

Krustlet acts as a Kubelet by listening on the event stream for new pods that the scheduler assigns to it based on specific Kubernetes [tolerations](#).

The default implementation of Krustlet listens for the architecture `wasm32-wasi` and schedules those workloads to run in a `wasmtime`-based runtime instead of a container runtime.

Documentation

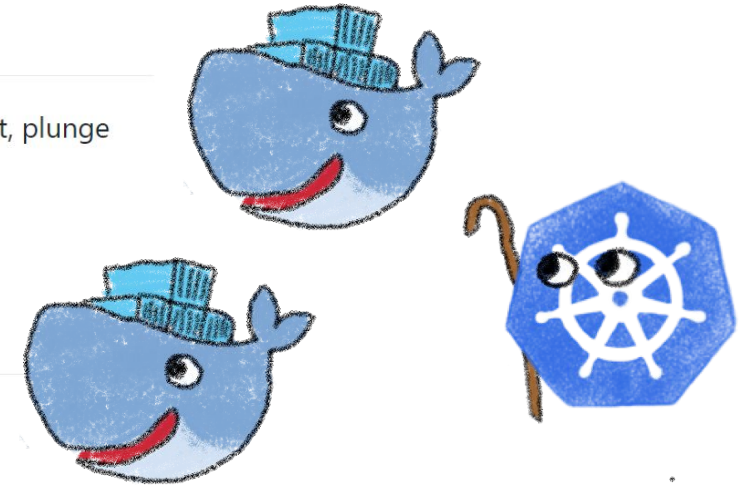
If you're new to the project, get started with [the introduction](#). For more in-depth information about Krustlet, plunge right into the [topic guides](#).

Looking for the developer guide? [Start here](#).

Community, discussion, contribution, and support

You can reach the Krustlet community and developers via the following channels:

- [Kubernetes Slack](#):
 - [#krustlet](#)
- Public Community Call on Mondays at 1:00 PM PT:
 - [Zoom](#)
 - Download the meeting calendar invite [here](#)



Some examples IRL?

Like companies using these things



fastly



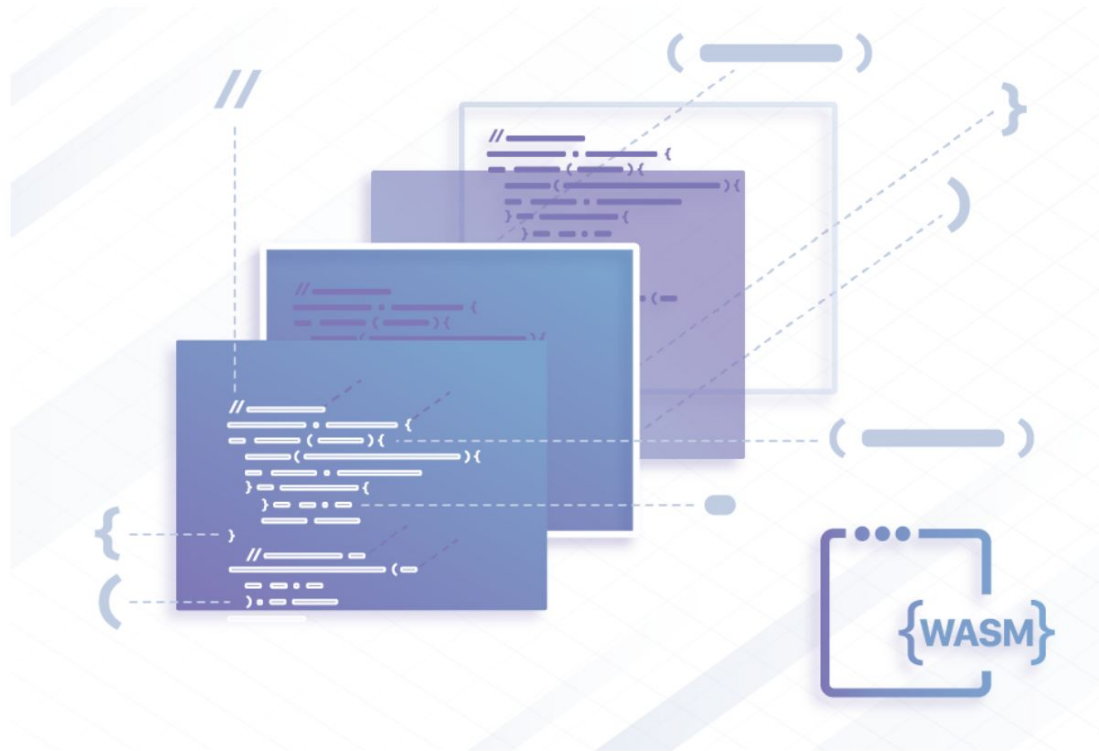
Cloudflare Workers (FaaS)

WebAssembly on Cloudflare Workers

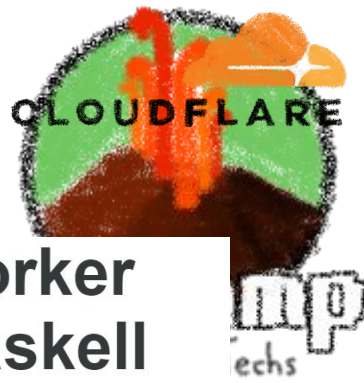
10/01/2018



Kenton Varda



Cloudflare Workers (FaaS)



Exploring WebAssembly AI Services on Cloudflare Workers

10/09/2020



Guest Author

This is a guest post by Videet Parekh, Abelardo Lopez-Lagunas, Sek Chai at [Latent AI](#).

Edge networks present a significant opportunity for Artificial Intelligence (AI) performance and applicability. AI technologies already make it possible to run compelling applications like object and voice recognition, navigation, and recommendations.

[AI at the edge](#) presents a host of benefits. One is scalability—it is simply impractical to send all data to a centralized cloud. In fact, [one study](#) has predicted a global scope of 90 zettabytes generated by billions of IoT devices by 2025. Another is privacy—many users are reluctant to move their personal data to the cloud, whereas data processed at the edge are more ephemeral.

When AI services are distributed away from centralized data centers and closer to the service edge, it becomes possible to enhance the overall application speed without moving data unnecessarily. However, there are still challenges to make AI from the deep-cloud run efficiently on edge hardware. Here, we use the term deep-cloud to

Let's build a Cloudflare Worker with WebAssembly and Haskell

10/06/2020



Cristhian Motoche

This is a guest post by Cristhian Motoche of [Stack Builders](#).

At [Stack Builders](#), we believe that Haskell's system of expressive static types offers many benefits to the software industry and the world-wide community that depends on our services. In order to fully realize these benefits, it is necessary to have proper training and access to an ecosystem that allows for reliable deployment of services. In exploring the tools that help us run our systems based on Haskell, our developer Cristhian Motoche has created a tutorial that shows how to compile Haskell to WebAssembly using Asterius for deployment on Cloudflare.

What is a Cloudflare Worker?

[Cloudflare Workers](#) is a serverless platform that allows us to run our code on the edge of the Cloudflare infrastructure. It's built on Google V8, so it's possible to write functionalities in JavaScript or any other language that targets WebAssembly.

[WebAssembly](#) is a portable binary instruction format that can be executed in a memory-safe sandboxed environment. For this reason, it's especially useful for tasks



Fastly works on the edge of the cloud

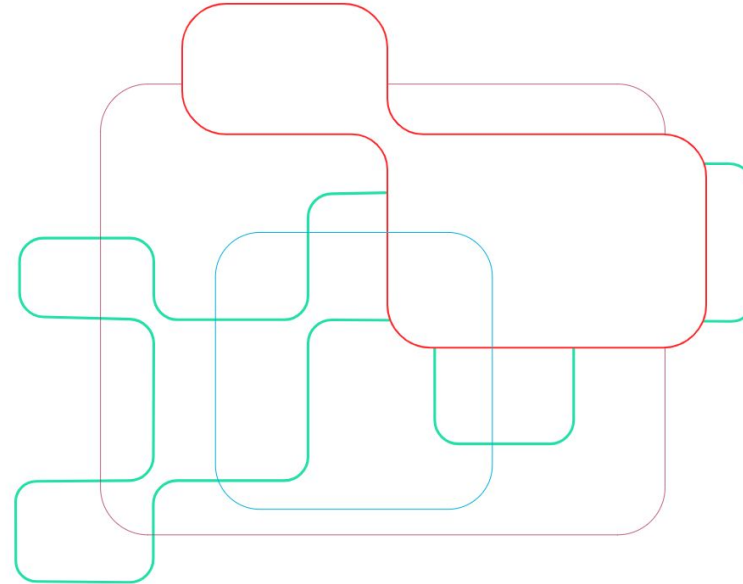


Edge compute technology

Give your developers an edge

Extend the power of your core cloud to the edge, and empower your teams to innovate. By moving data and applications as close to your end users as possible, you can deliver fast, highly personalized experiences to customers around the world. And now you can [join the private beta for Compute@Edge](#), our new serverless compute environment built to take you further at the edge.

Get a demo



fastly



Fastly works on the edge of the cloud



COMPUTE@EDGE

Serverless Compute Environment

Compute@Edge is the next generation of serverless computing. It combines the best in emerging technology to give developers unmatched power, scalability, security, and speed at the edge.

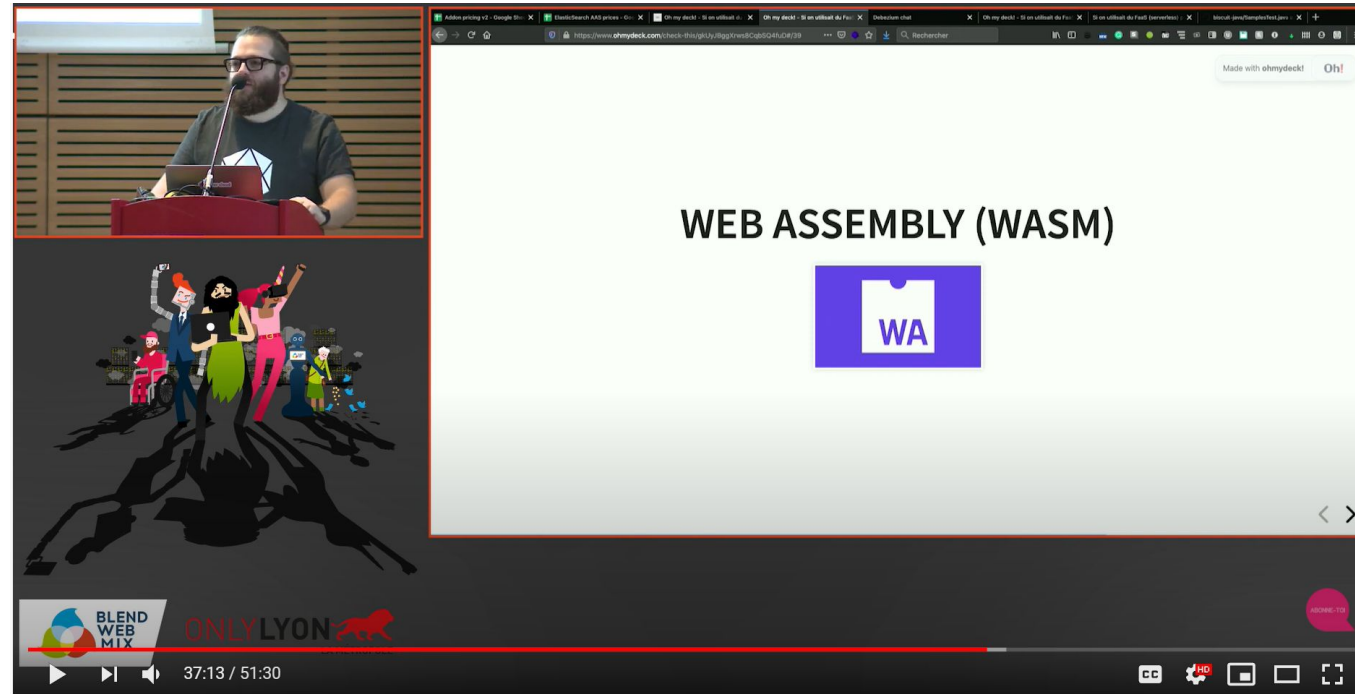
WebAssembly



We built Compute@Edge on WebAssembly, a technology we've been collaborating on with the [Bytecode Alliance](#). WebAssembly is uniquely suited to empower developers to write code in their preferred language, then run that code anywhere at near-native speeds.



Clever Cloud FaaS infrastructure



clever cloud

<https://www.youtube.com/watch?v=wchehMIsu80>



Clever Cloud FaaS Infrastructure

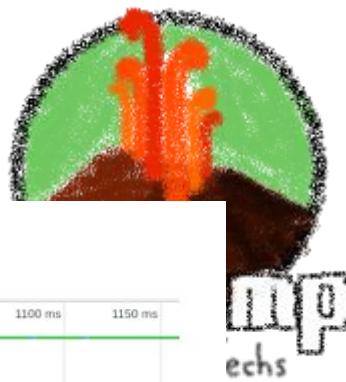


clever cloud

Already used in internal projects
Able to execute complex loads
(neural networks models)



Clever Cloud FaaS Infrastructure



le chargement directement depuis le site original



au travers du faas



on met 130ms à la première requête, parce qu'il faut boot la VM (9ms), la fork (1.5ms), puis établir la connexion TLS au backend



Blazing fast

Clever Cloud FaaS Infrastructure



le site complet:



le site complet au travers du faas:



en gros, 260ms de plus sur le DomContentLoaded, 300ms de plus sur le load sachant que mes VM s'exécutent sur un seul thread, j'ai pas parallélisé



Blazing fast

That's all, folks!

Thank you all!

