WebAssembly

aka WASM

“Du bon code avec le bon sens”

Julien Rollin

_CTO TalanLabs

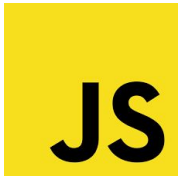
Dev / CoachCraft / Cloud ready

Les promesses de WebAssembly

- rapide et performant
- sécurisé
- debuggable
- open web plateforme (w3c)

TLDNR

WASM *n'est pas le tueur de Javascript*



Au commencement, il y avait Javascript

Les étapes clés

- 1995: création Brendan Eich, employé chez Netscape
- 1997 : spécification ECMAScript
- 2006, création de NodeJS avec Ryan Dahl
- 2008 : progressbar chez Flickr

Nb: pas de vidéos, pas de gros cpu et pas de haut débit...

Les étapes clés (suite)

- 2009: le moteur JavaScript V8 de Google devient OpenSource
- 2009: NPM est créé
- 2014 : finalisation HTML5 (débuté en 2011)
- React, Vue, Electron, etc

Nb: pas de vidéos, pas de gros cpu et pas de haut débit...

Les limitations de Javascript

- pas conçu pour être performant
- pas de typage fort (volonté de prise en main rapide)
- pas adapté au réseau bas niveau
- monothread

De nombreuses initiatives pour améliorer les performances

“

Quelques exemples

- NaCl : Google Native Client
  - sandbox et performance
- PNaCl : Portable Google Native Client
  - pas de portabilité entre navigateurs (plugins firefox)
- Asm.js
  - sandbox + sécurité mais manque de performance

Une envie commune

- code performant
- code portable
- sandbox pour plus de sécurité

Nécessité de faire converger vers un standard

“

Mais cela prend du temps...

HTML5 par exemple

“

2015, *Brendan Eich, créateur de Javascript, annonce que le travail sur WebAssembly a commencé !*

“

“Develop the Web’s polyglot-programming-language object-file format”

Brendan Eich

W3C Working Group (WG) *a été missionné pour définir les standards*

“

Le projet sera donc intégré à tous nos navigateurs !

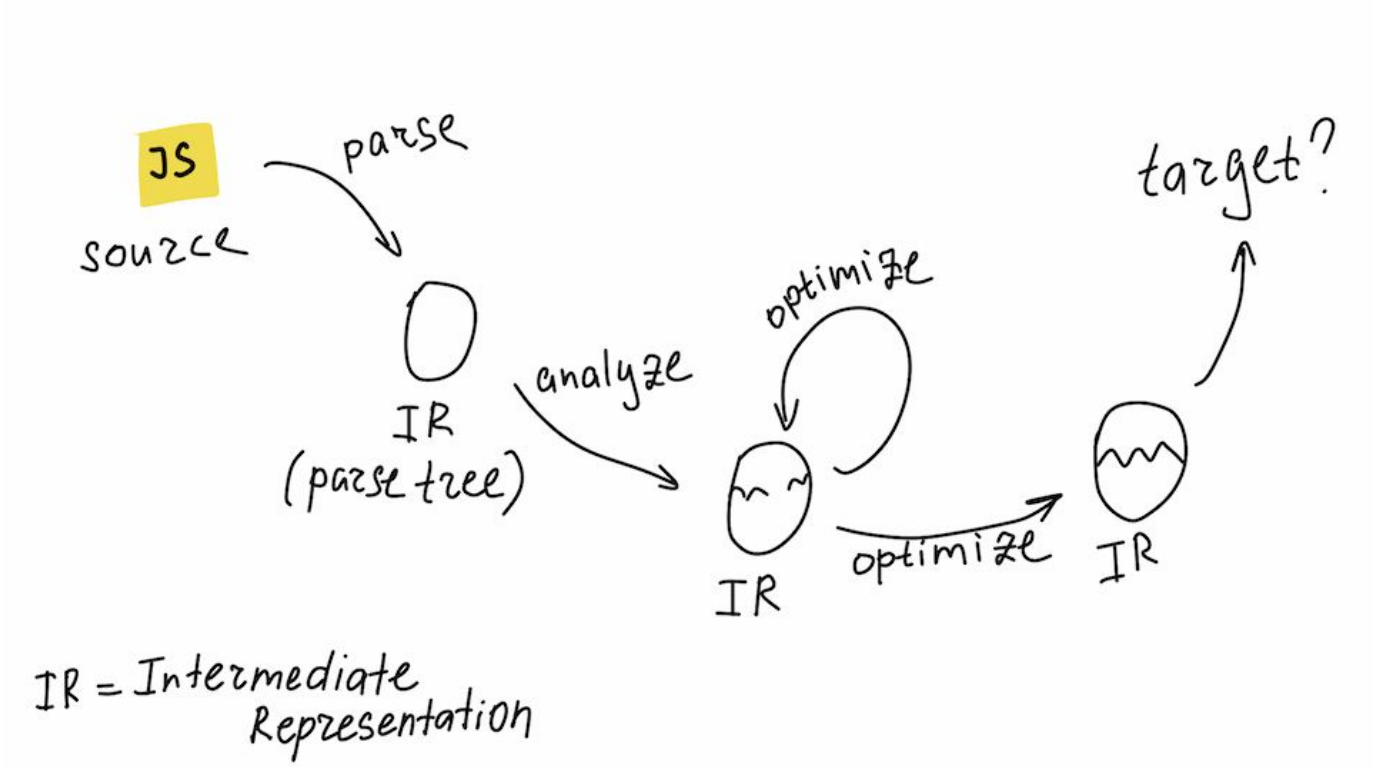
“

et c’est déjà le cas ;)

faut l’oublier maintenant hein...

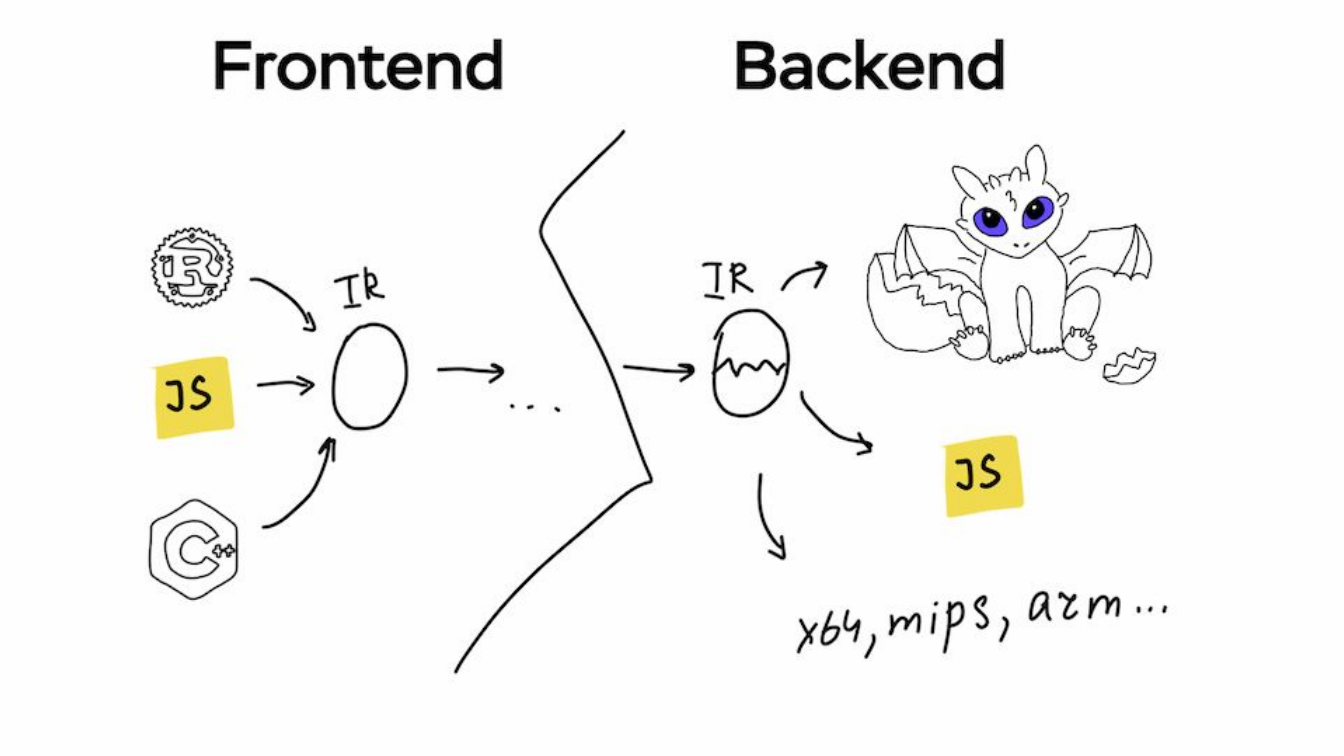
Vous avez dit performance?

compilateurs et performances



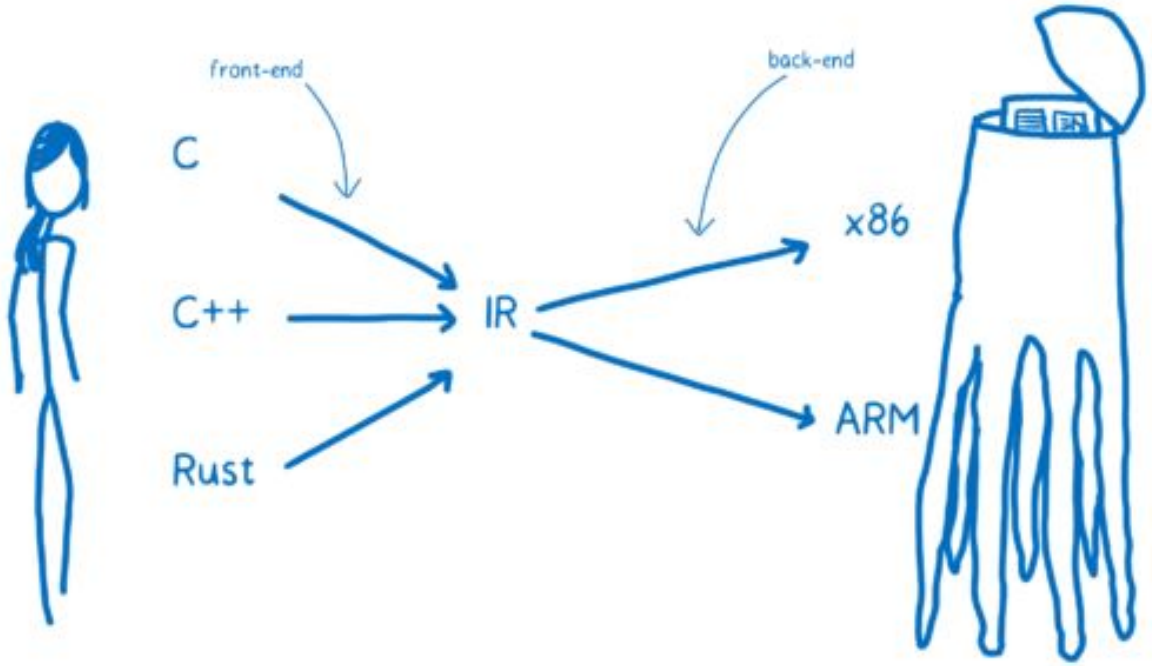
Optimisation du code à la compilation

crédit images: Evilmartians



Séparer la compilation en deux parties

crédit images: Evilmartians



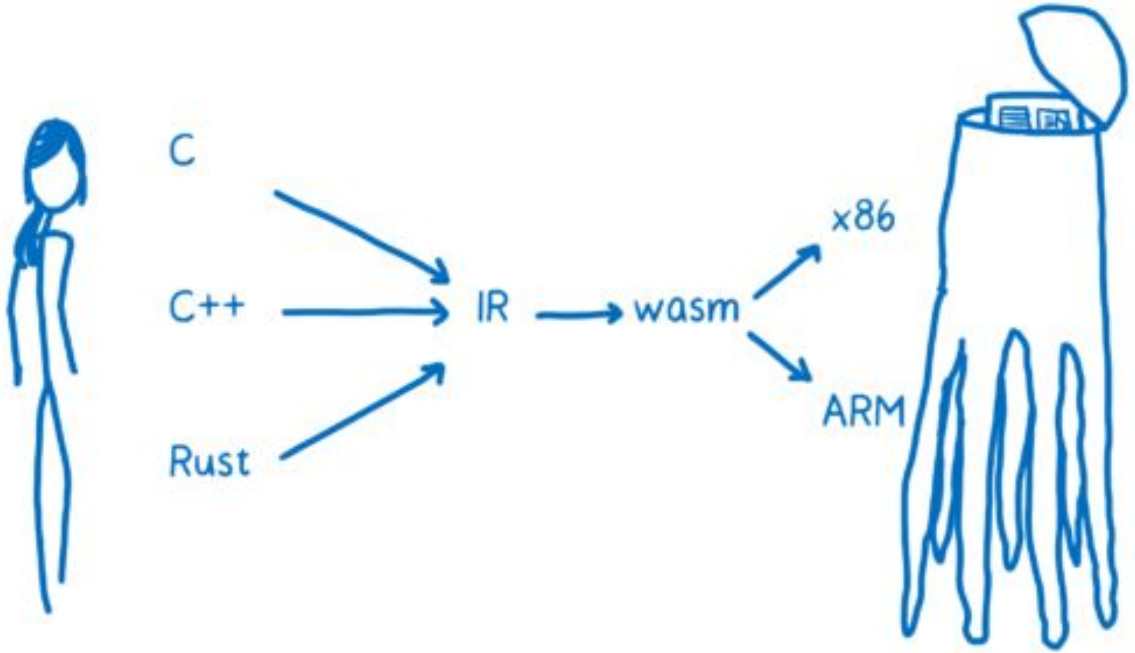
Beaucoup de combinaisons possibles

crédit images: Code cartoon

WASM *est l'IR du web*

Un code sans runtime spécifique

- Virtual Instruction Set Architecture (VISA)
  - le bytecode est comme du code machine mais conçu pour être exécuté dans machine virtuelle
- suffisamment bas niveau pour avoir de bonnes performances
- portable
  - pas besoin de s'occuper du type de cpu
- les navigateurs ont le support de la machine virtuelle WebAssembly déjà intégré !



crédit images: [Code cartoon]

Montrez moi le code !

human vs machine

WebAssembly Text Format (Wat) pour les humains

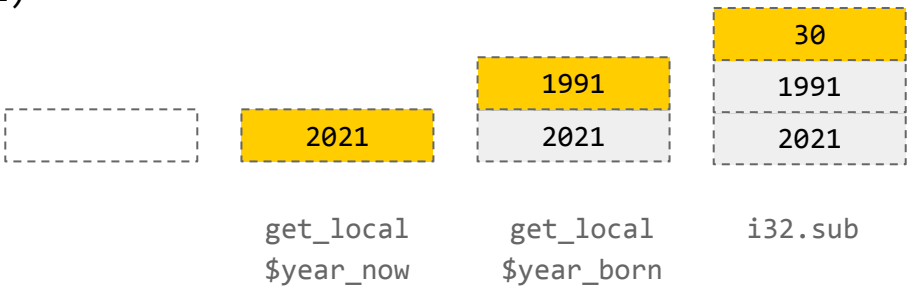
Contenu d'un fichier WAT

```
(module
     (func $how_old (param $year_now i32)
(param $year_born i32) (result i32)
     get_local $year_now
     get_local $year_born
     i32.sub)

     (export "how_old" (func $how_old))
)
```

- Lisp Like
  - définition des fonctions en expression comme des listes (S-expression)
  - stack machine instructions
- définition d'un module
- fonctions privées vs publiques

Focus sur les instructions “Stack machine”

```
(module
     (func $how_old (param $year_now i32)
(param $year_born i32) (result i32)
     get_local $year_now
     get_local $year_born
     i32.sub)

     (export "how_old" (func $how_old))
)
```



WASM *Binary file*
pour les machines

Conversion en “binaire”

```
(module
      (func $how_old (param $year_now i32) (param
$year_born i32) (result i32)
      get_local $year_now
      get_local $year_born
      i32.sub)

      (export "how_old" (func $how_old))
)
```

```
wasm-function[0]:
  sub rsp, 8                    ; 0x000000 48 83 ec 08
  mov ecx, edi                  ; 0x000004 8b cf
  mov eax, ecx                  ; 0x000006 8b c1
  sub eax, esi                  ; 0x000008 2b c6
  nop                           ; 0x00000a 66 90
  add rsp, 8                    ; 0x00000c 48 83 c4 08
  ret                           ; 0x000010 c3
```

> wat2wasm hello.wat

Utilisation du fichier WASM

```
const wasmInstance = new WebAssembly.Instance(wasmModule, {});

const { how_old } = wasmInstance.exports;

console.log(how_old(2021, 2000));
```

Gestion de la mémoire

sûr ?

Gestion de la mémoire sûre

- la pile d'exécution est séparée de la mémoire linéaire et n’est pas accessible par le code
- les modules n'ont pas accès directement à la mémoire du système
- mémoire linéaire
- pas d’accès direct aux éléments stockés en mémoire (!= C)
- pas de “garbage collector”

JS et WASM ne peuvent dialoguer que par des entiers

Seulement 4 types

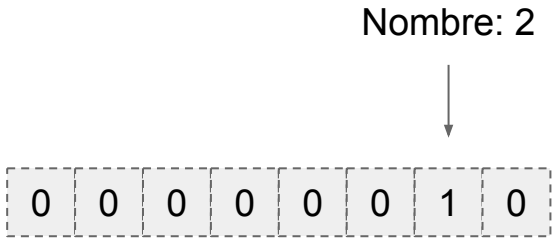
- float 32
- float 64
- integer 32
- integer 64 bits

Rq: pas de type chaîne de caractère ou même tableau

Pas de tableaux… ?

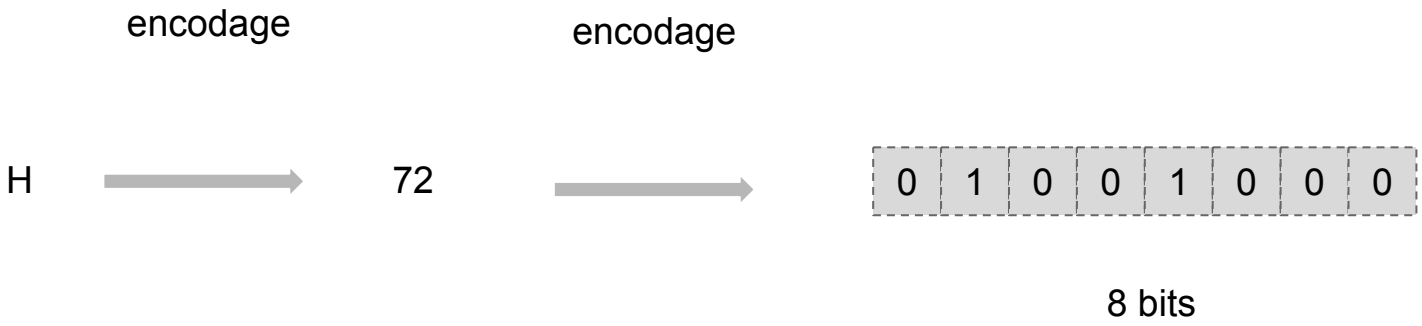
Stocker un nombre en 8 bits

8 bits



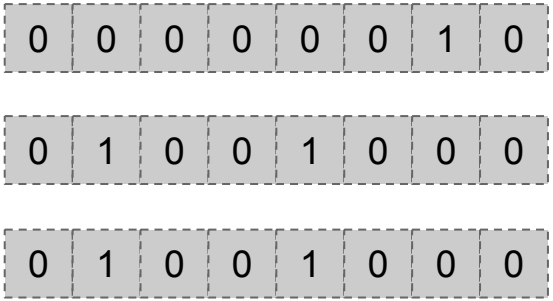
8 bits

Stocker une lettre en 8 bits



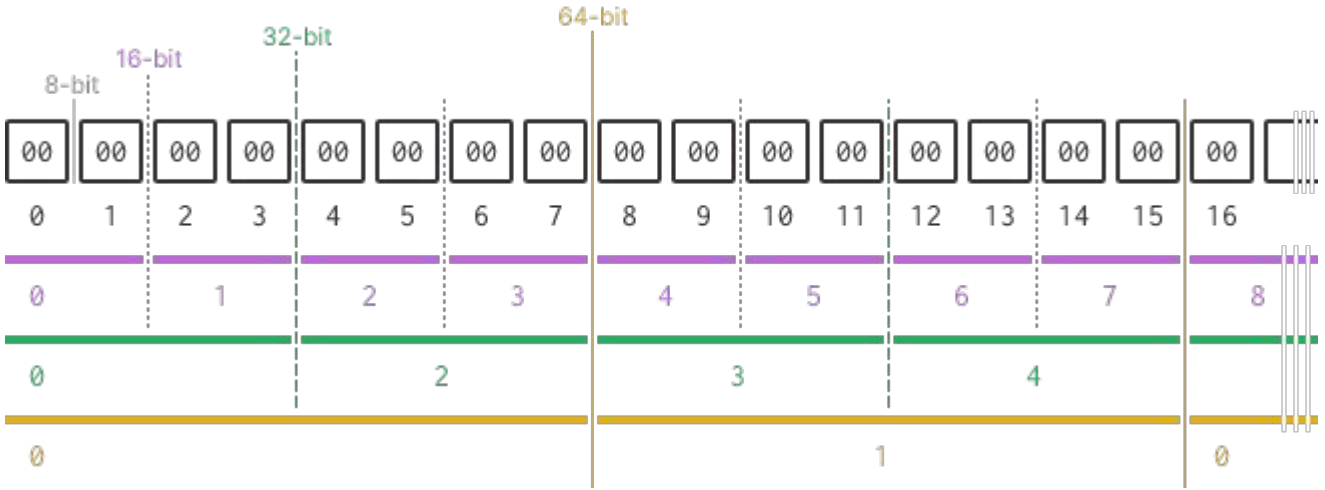
Des tableaux quand même

int8Array = [



]

Une taille + un indice pour récupérer un élement

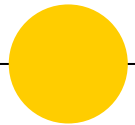


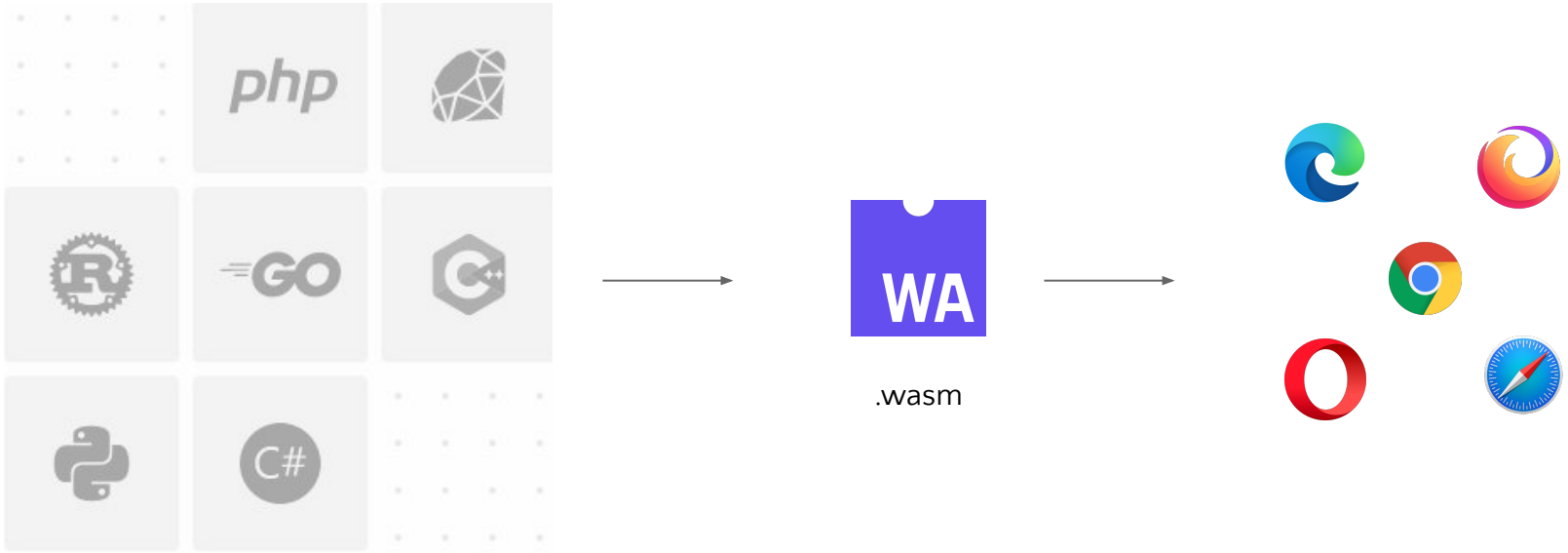
effective-address = address-operand + offset-immediate

WASM module dans un Array Buffer

```
function fetchAndInstantiate(url, importObject) {
  return fetch(url).then(response =>
    response.arrayBuffer()
  ).then(bytes =>
    WebAssembly.instantiate(bytes, importObject)
  ).then(results =>
    results.instance
  );
}
```

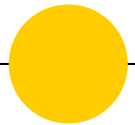
La représentation intermédiaire permet de choisir son langage





Toolchain très fournie

Opérabilité
avec le DOM possible



JS fonction dans un module WAT

```
(module
      (import "js" "import1" (func $i1))
      (import "js" "import2" (func $i2))
      (func $main (call $i1))
      (start $main)
      (func (export "f") (call $i2))
)
```

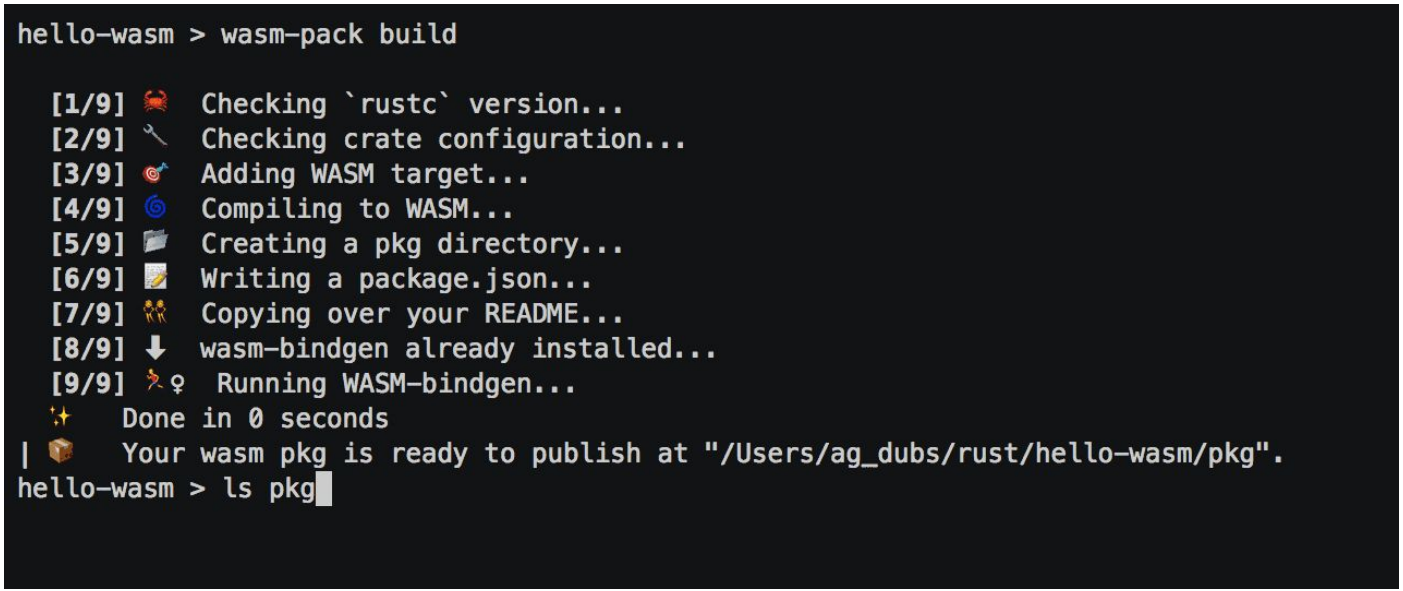
```
var importObj = {js: {
      import1: () => console.log("hello,"),
      import2: () => console.log("world!")
}};
fetch('demo.wasm').then(response =>
      response.arrayBuffer()
).then(buffer =>
      WebAssembly.instantiate(buffer, importObj)
).then(({module, instance}) =>
      instance.exports.f()
);
```

Rust facilite le binding

Crate “Wasm-bindgen”

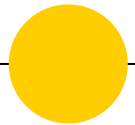
```
#[wasm_bindgen]
extern {
      type console;

      #[wasm_bindgen(static = console)]
      fn log(s: &str);
}
```

```
#[wasm_bindgen]
pub fn foo() {
      console::log("hello!");
}
```



Rust Wasm pack pour configurer npm / rust / wasm

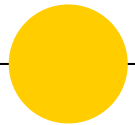
Wasm côté Serveur avec Node

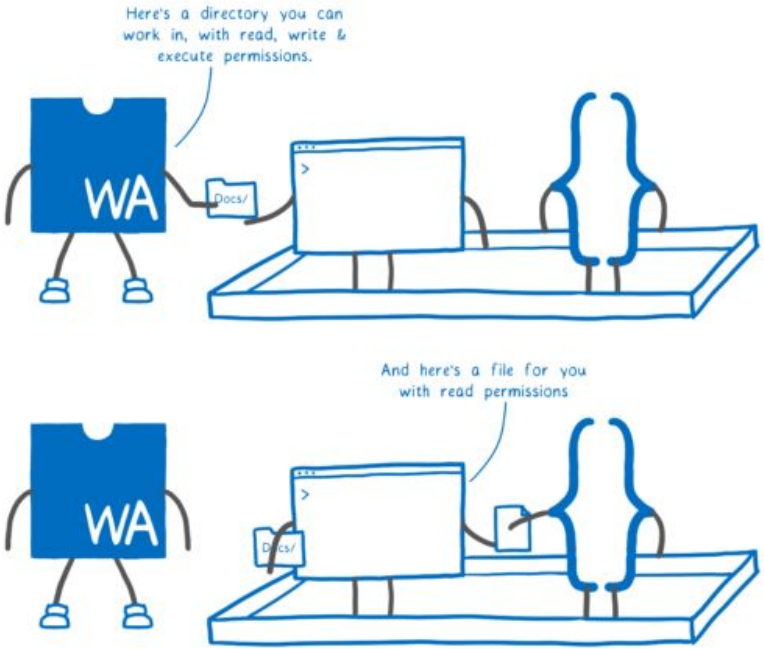


Disponible dans Node

```
// Assume add.wasm file exists that contains a single function adding 2 provided arguments
const fs = require('fs');
const wasmBuffer = fs.readFileSync('/path/to/add.wasm');
WebAssembly.instantiate(wasmBuffer).then(wasmModule => {
  // Exported function live under instance.exports
  const add = wasmModule.instance.exports.add;
  const sum = add(5, 6);
  console.log(sum); // Outputs: 11
});
```

Code côté serveur mais sans Node !





WebAssembly System Inferface

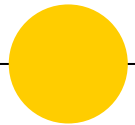
Pas d'accès direct aux fonctionnalités de l'OS

WebAssembly System Inferface WASI

Quelques Runtime

wasmtime

Quelques usages



Machine learning

Lower latency, greater privacy, and lower serving cost.

Latency because no request and response needs to go to the server and come back.

Privacy because all of the data that's fed into the model can live on the device and never go to the servers.

TensorFlow

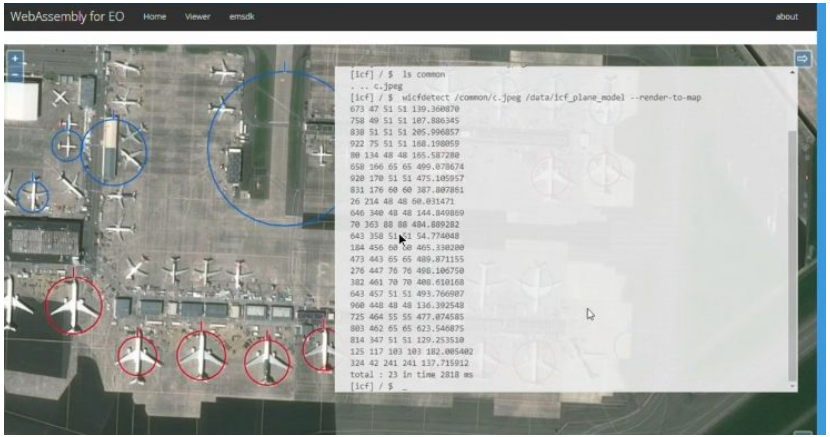
Emulateur Gameboy

[wasmboy]

Analyse d'image

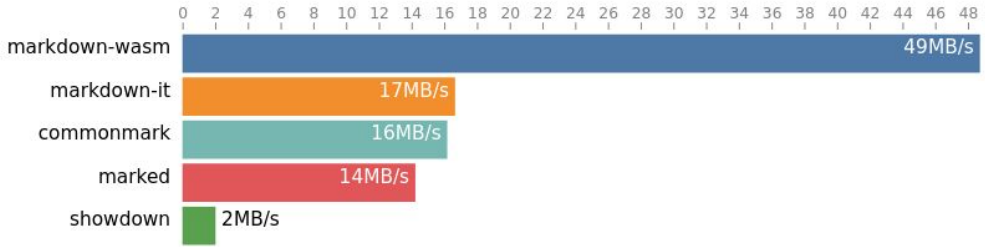
import lib to read importer des images jpg2000 de plusieurs GO

analyser les images pour détecter des avions par ex



WebAssembly dans la vraie vie - MixIT2019
Nicolas Decoster

Portage de lib C en WASM




```
# Example

A third of the distance across the Beach, the meadow ends and sand begins. This slopes gradually up
for another third of the distance, to the foot of the sand hills, which seem tumbled into their places
by some mighty power, sometimes three tiers of them deep, sometimes two, _and sometimes only one._ A
third of the distance across the Beach, the meadow ends and sand begins.

The outline of this inner shore is most irregular, curving and bending in and out and back upon
itself, making coves and points and creeks and channels, and often pushing out in flats with not water
enough on them at low tide to wet your ankles.

## Subtitle

This is another fine paragraph

### Smaller subtitle

This is a paragraph `with` ~style~ *italic* _italic_ **bold** __bold__

![image](https://rsms.me/raster/examples/image1.jpg)

*Hello [link](https://rsms.me/) lol*

Hello [*link*](https://rsms.me/) lol "cat"

Hello from *[link](https://rsms.me/)* to __everyone__ `reading this`
```

Portage de la librairies C MD4C pour parser le markdown

https://github.com/rsms/markdown-wasm

Et pour finir

- Wasm3 pour éxecuter du wasm sur des devices
- WebAssembly Micro Runtime (WAMR)

Mangez du WASM,
c’est déjà le présent !

Merci !