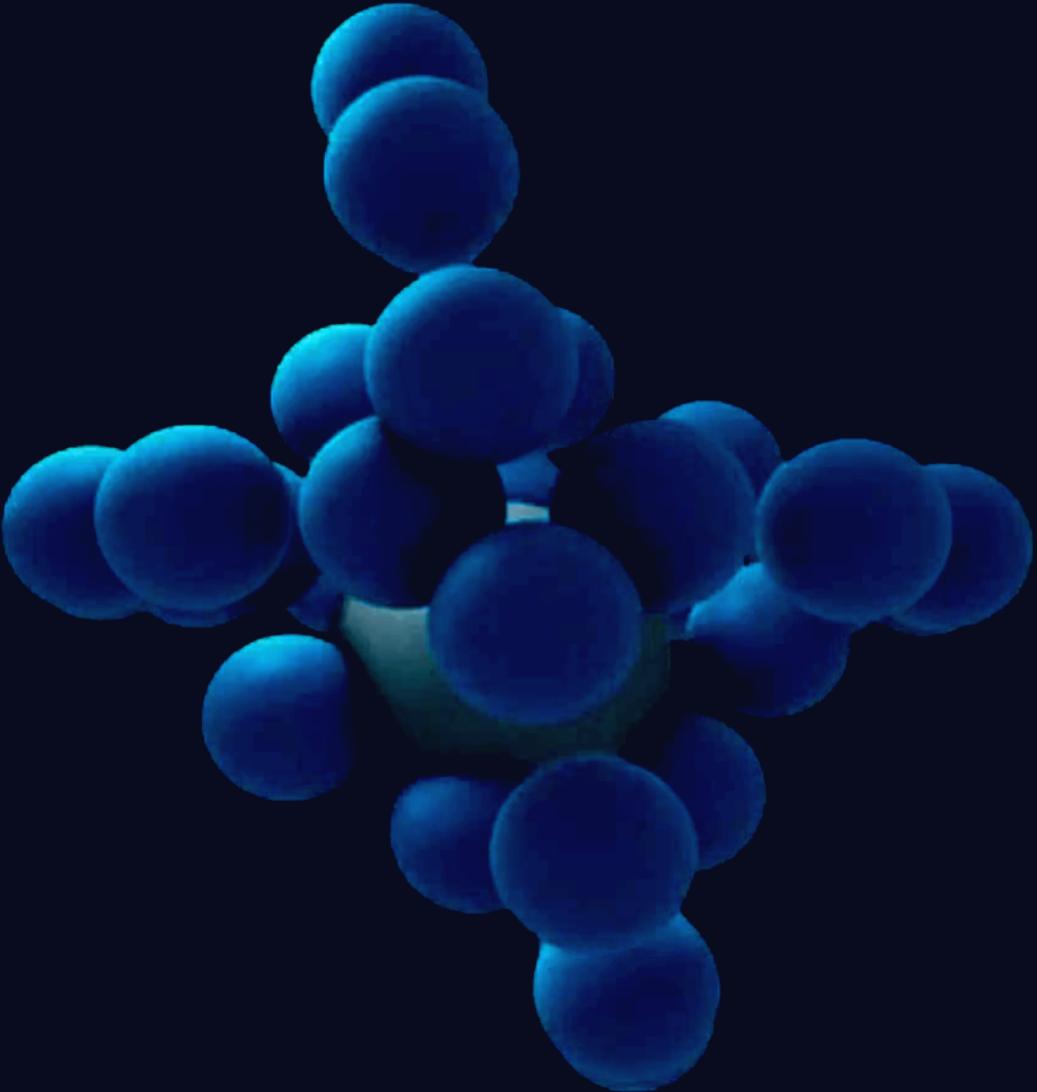


# JavaScript and WebAssembly: story of a symbiosis



> you

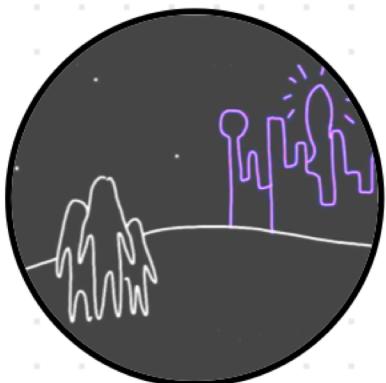
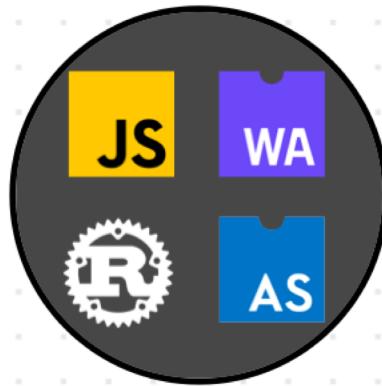
---



Developers (web)

> me

---



Wasm Nano  
Handbook  
+ AS



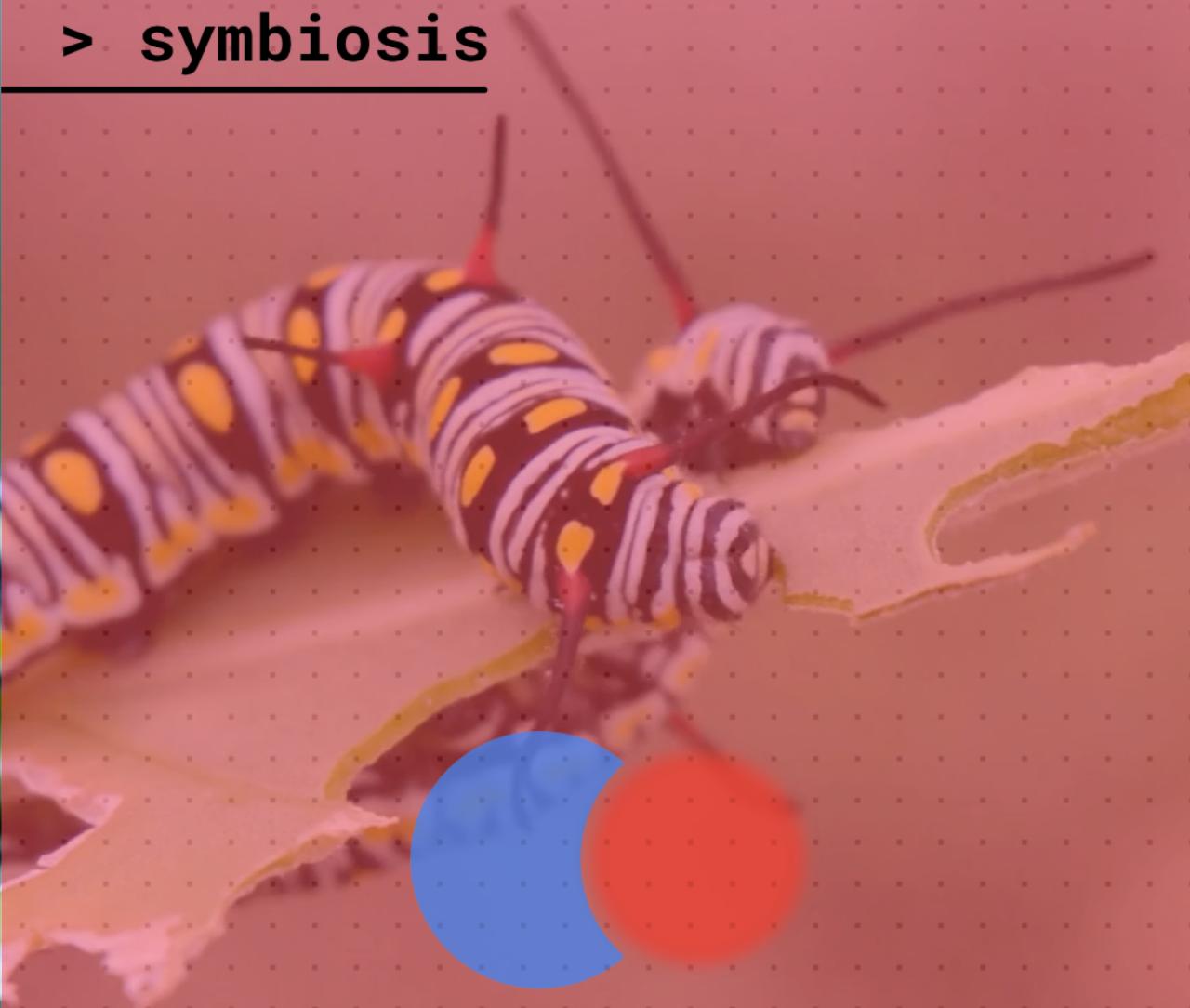
@maudnals



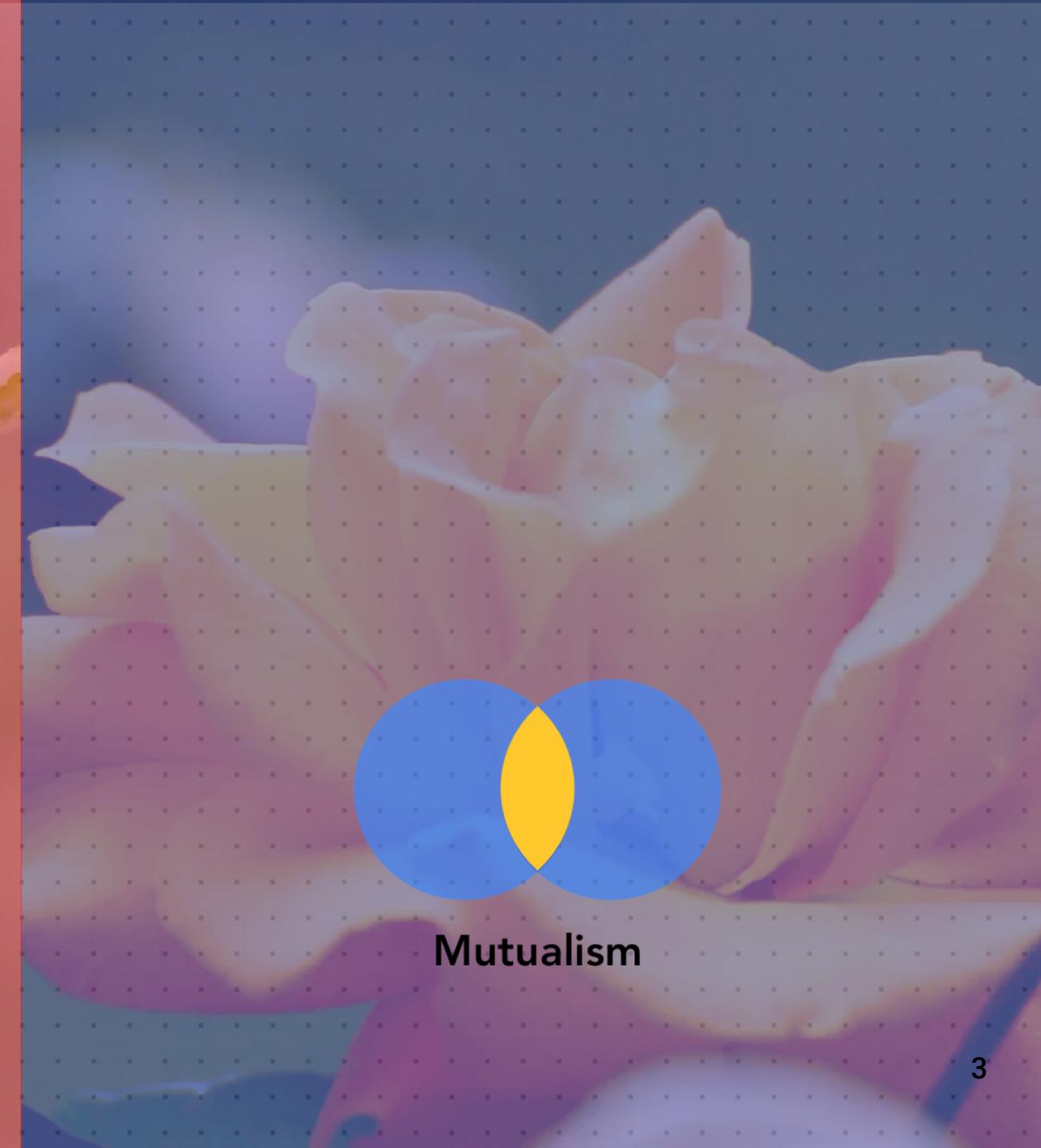
Berlin Wasm  
Meetup

## > symbiosis

---



Parasitism



Mutualism

# > agenda

\*Wasm = WebAssembly



JS, Wasm\* and the browser



Symbiosis: How JS and Wasm benefit from each other  
MICRO/MACRO



Symbiosis: Writing Wasm in TypeScript  
META

Demo



Go



QnA



JS, Wasm  
and the  
browser

# > JS

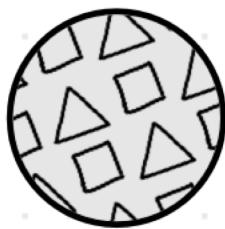
---



Web and more



High-level

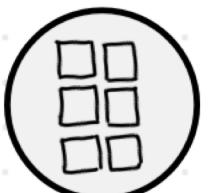


Dynamically typed

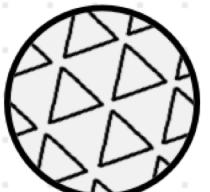
# > Wasm



New language for  
the **web**



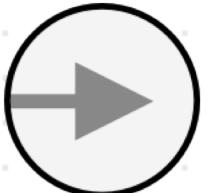
**Low-level** bytecode



**Typed**



Near-native **speed**,  
compact binary  
format



Compilation **target**

The screenshot shows a browser window titled "WASM Example" displaying the "WebAssembly Video Editor" at <https://d2jta7o2zej4pf.cloudfront.net>. The interface includes a sidebar with a dropdown menu for "Switch to Video" containing various filters like Normal, Grayscale, Invert, Bacteria, Sunset, Emboss, Super Edge (which is selected), Super Edge Inv, Gaussian Blur, Moss, Robbery, Brighten, Swamp, Ghost, Good Morning, Acid, Urple, Romance, Hippo, Longhorn, Security, Underground, and Rooster. The main area shows a video frame of a person's face with a superimposed edge detection effect. A message at the bottom says "Switch back to video for player controls". To the right, there is a "Performance Comparison" section stating "WASM is currently 37% faster than JS" with computation times of 13.8 ms for WASM and 19.3 ms for JS. It also shows a "Frame Rate" graph comparing WASM (green line) and JS (blue line) frame rates. At the bottom, there are buttons for "Disable Javascript" and "Hide JS Canvas".

## > Wasm

---

```
(module
  (func $add (param $lhs i32)
  (param $rhs i32) (result i32)
    get_local $lhs
    get_local $rhs
    i32.add)
  (export "add" (func $add))
)
```

# > human to metal

HIGH-LEVEL



LOW-LEVEL

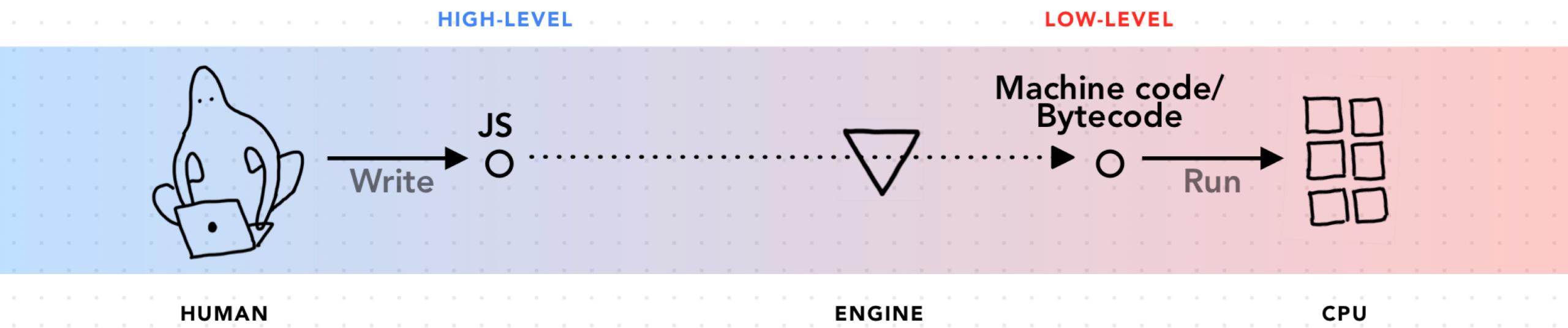
Machine code/  
Bytecode



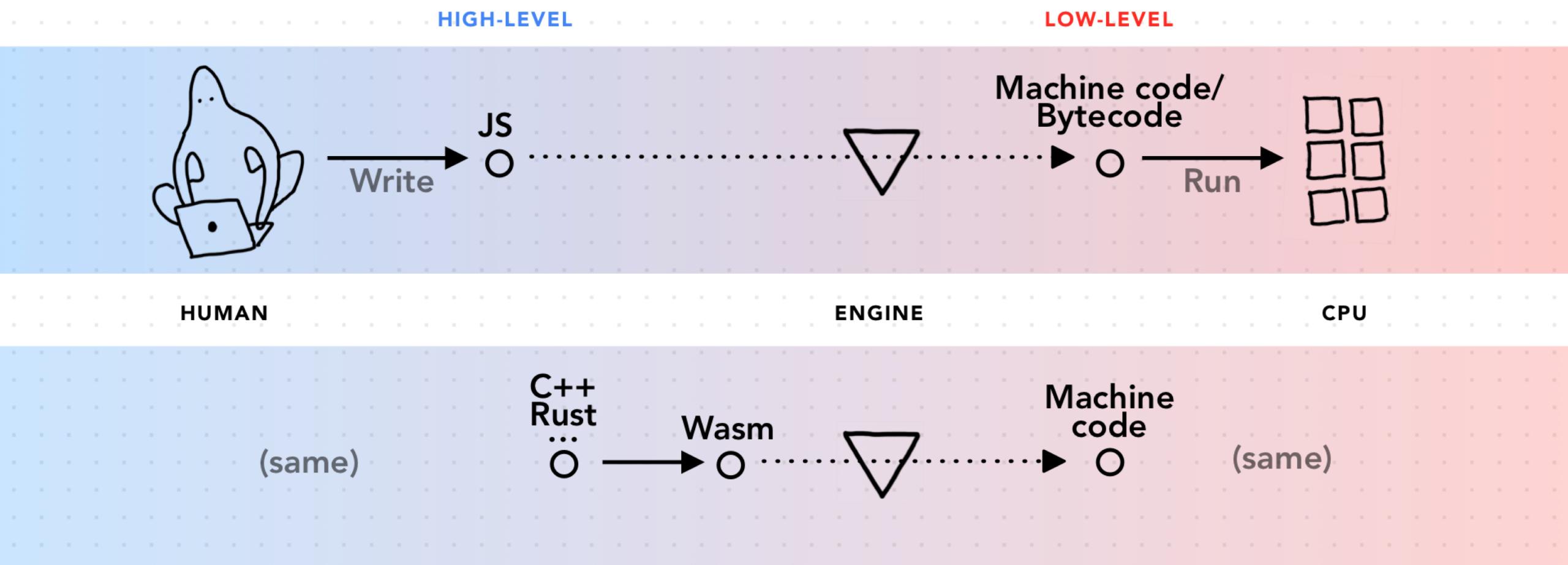
HUMAN

CPU

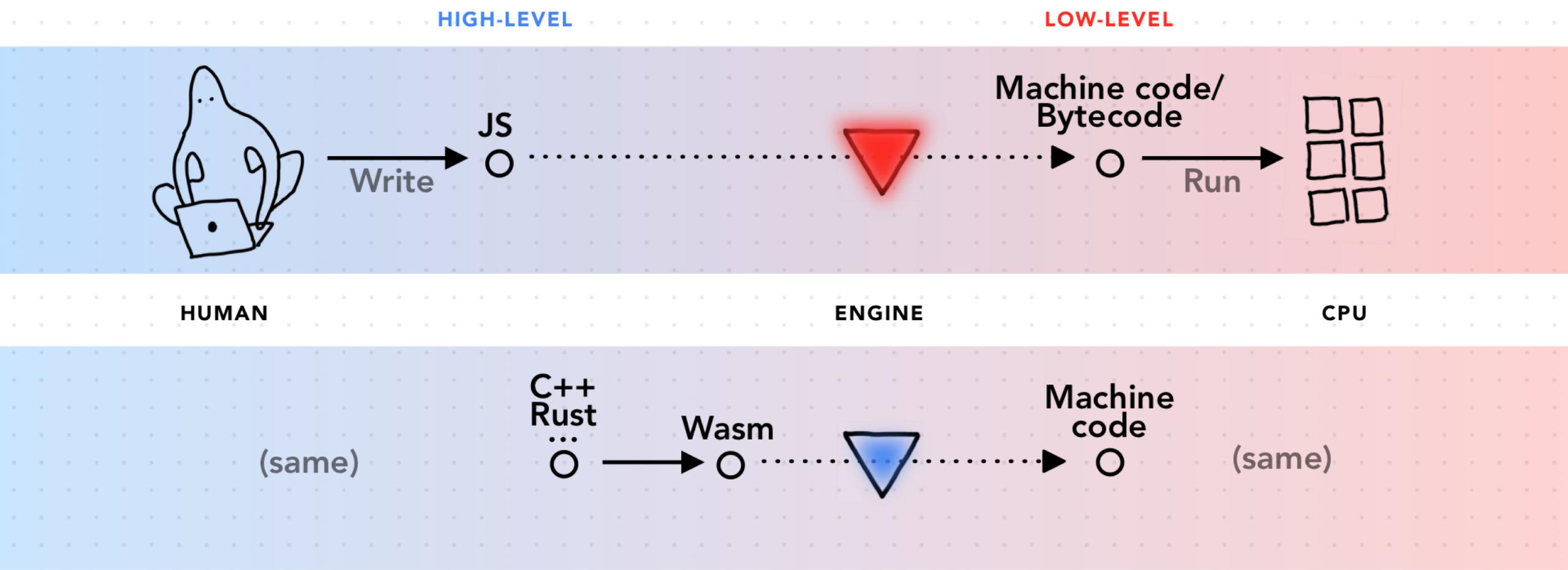
# > human to metal

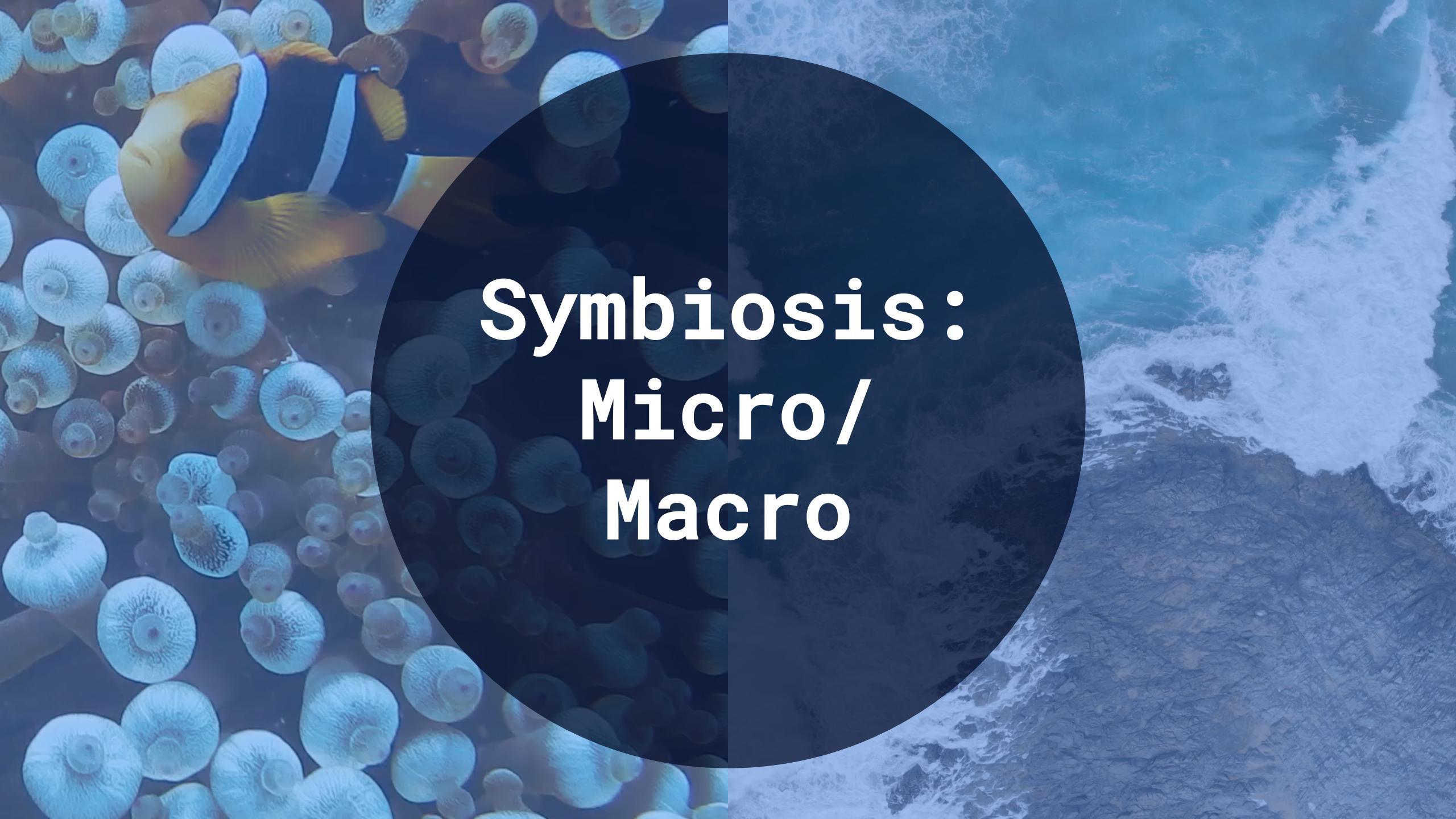


# > human to metal



# > human to metal





# Symbiosis: Micro/ Macro

# > interop

js index.js

```
const importObject = {
  console: {
    log: function(arg) {
      console.log(arg);
    }
  }
};

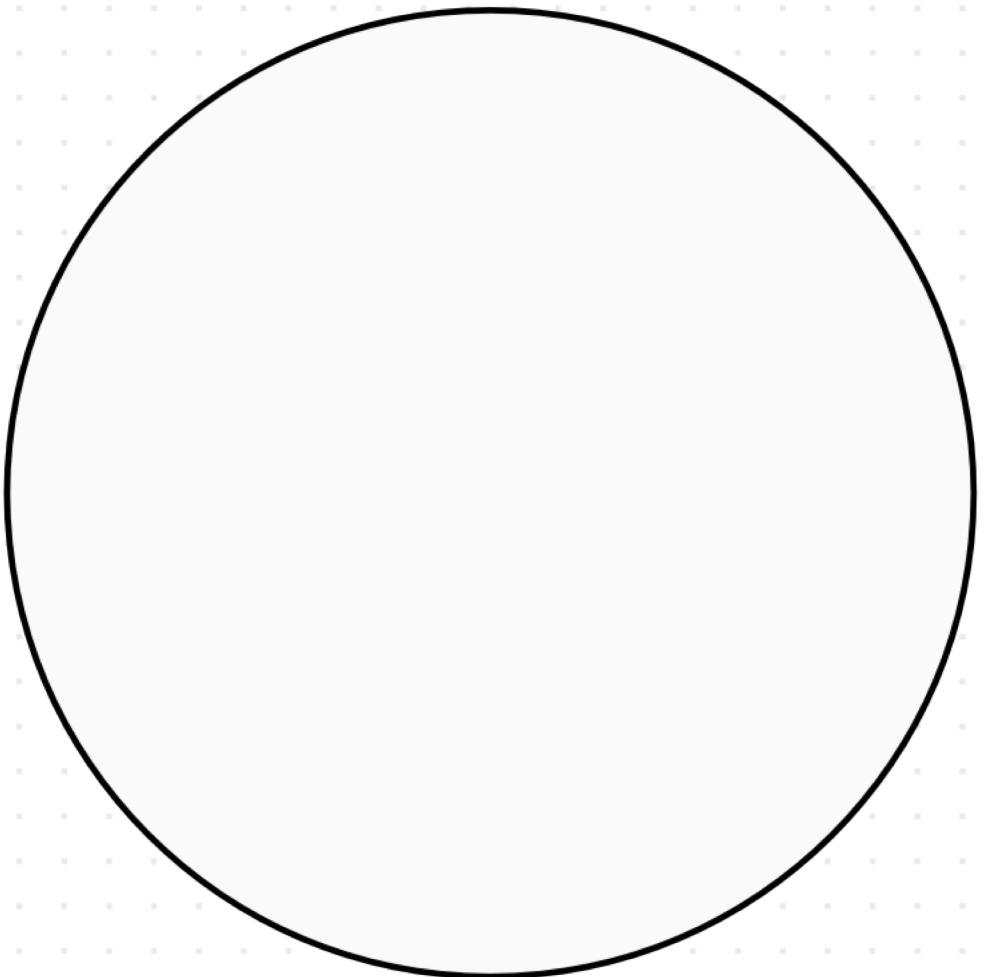
WebAssembly.instantiateStreaming(
  fetch('x.wasm'), importObject)
  .then(obj => {
    obj.instance.exports.log(); // 42
  });

```

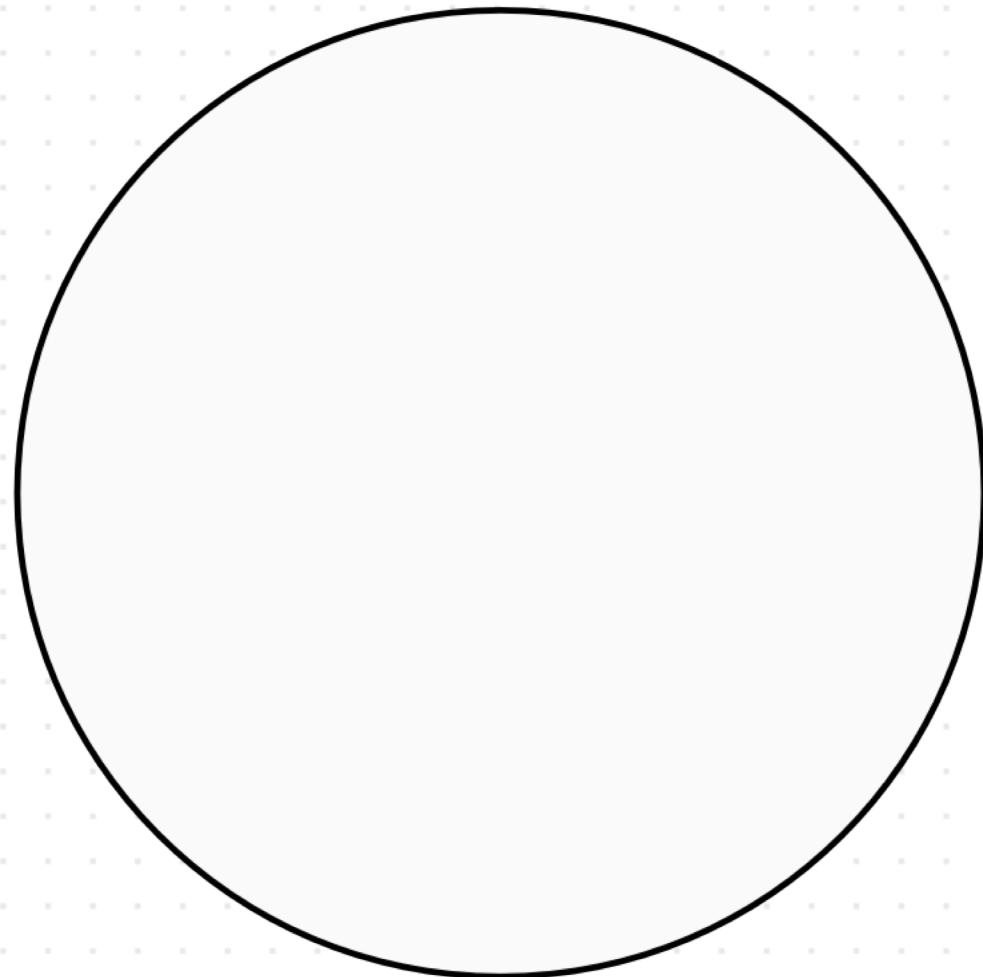
WA x.wasm (wat)

```
(module
  (import "console" "log" (func $log
    (param i32)))
  (func (export "log")
    i32.const 42
    call $log))
```

> **benefits:**  
**micro**

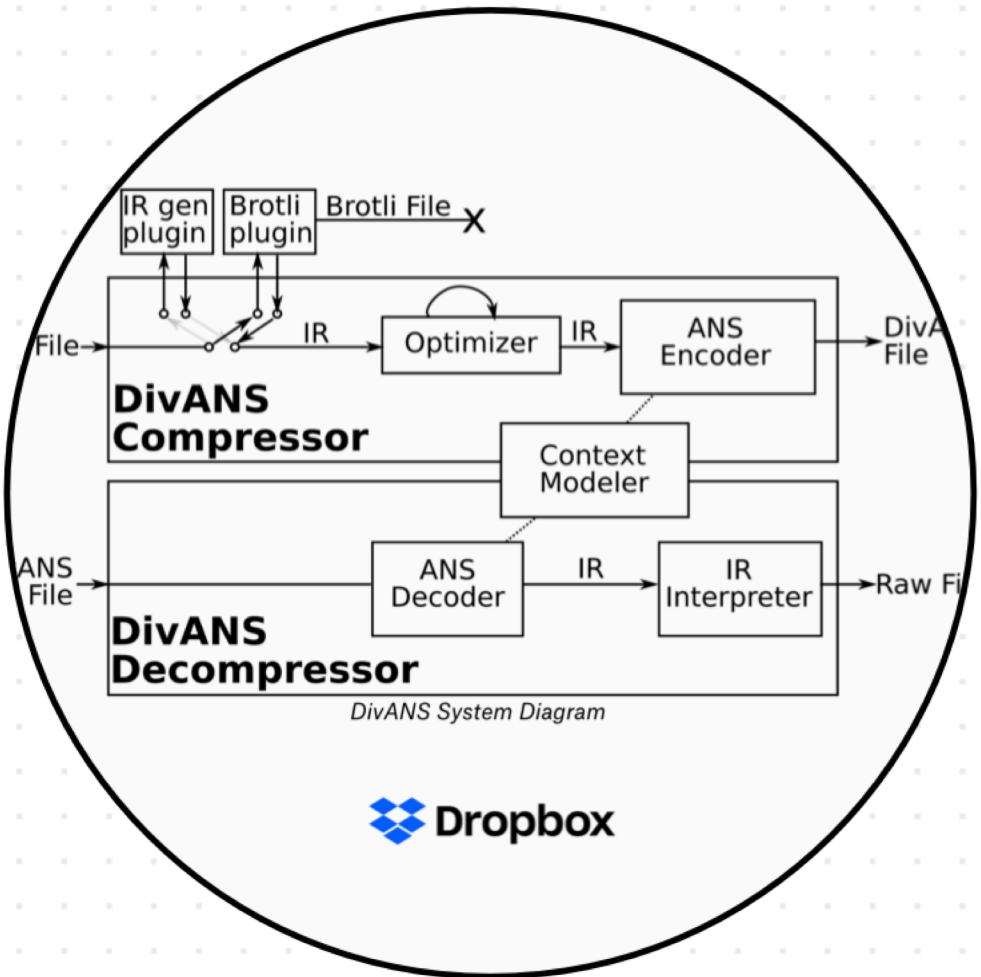


**Wasm for JS**

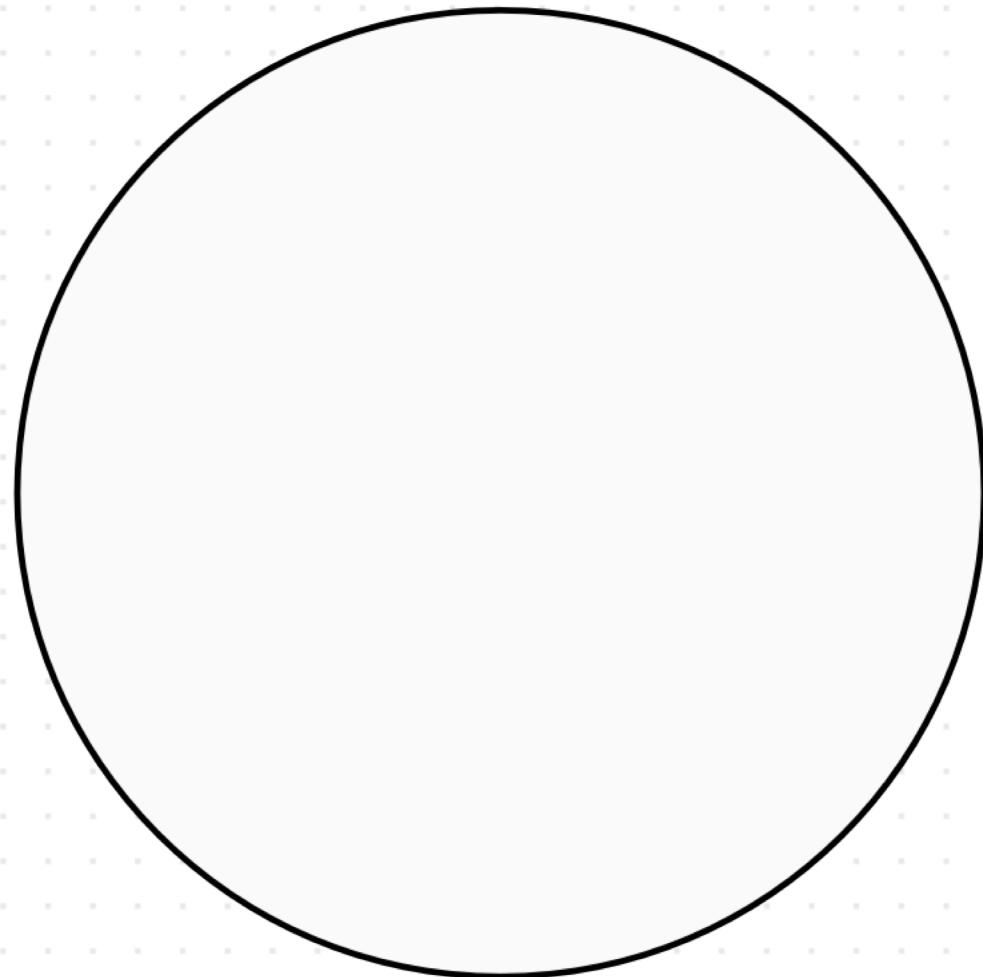


**JS for Wasm**

> benefits:  
micro

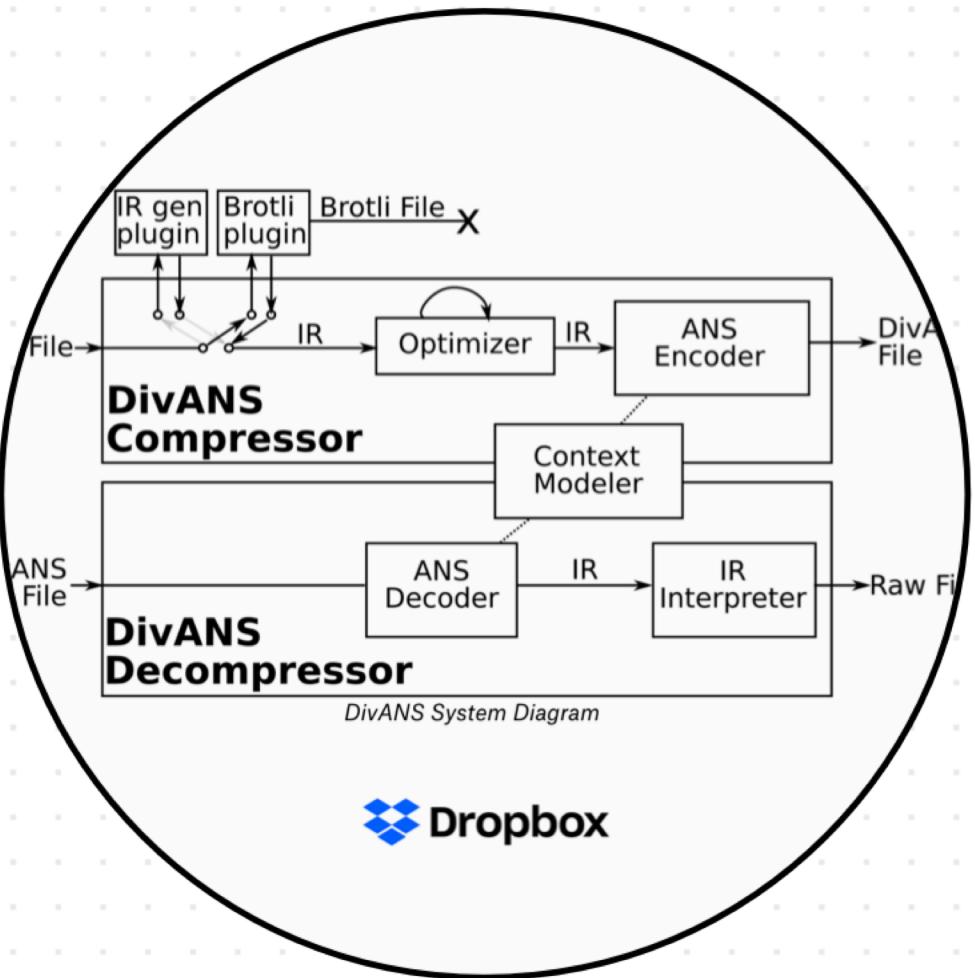


Wasm for JS



JS for Wasm

## > benefits: micro



Wasm for JS

## API reference ↗

### WebAssembly

This object acts as the namespace for all WebAssembly objects.

### WebAssembly.Global()

A `WebAssembly.Global` object represents a global variable that is both JavaScript and importable/exportable across multiple WebAssembly instances. This allows dynamic linking of multiple modules.

### WebAssembly.Module()

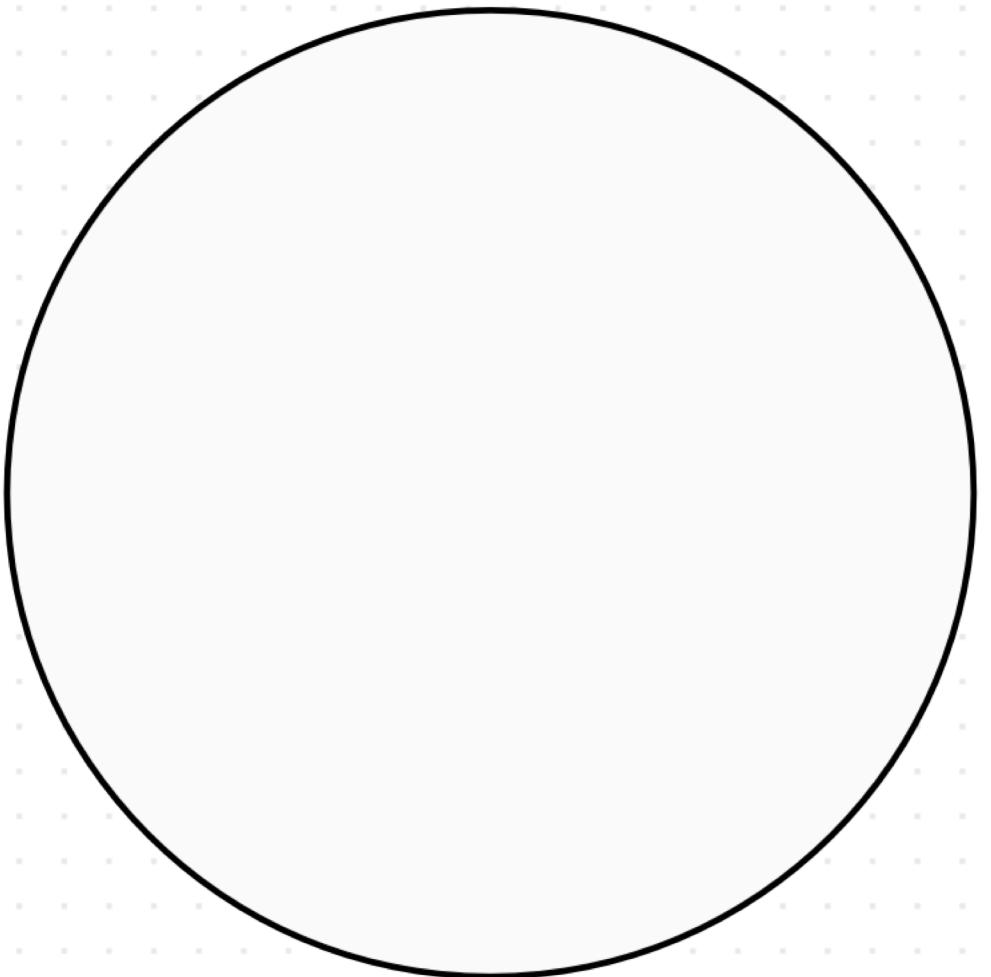
A `WebAssembly.Module` object contains stateless code that has been compiled by the browser and can be efficiently executed multiple times.

### WebAssembly.Memory()

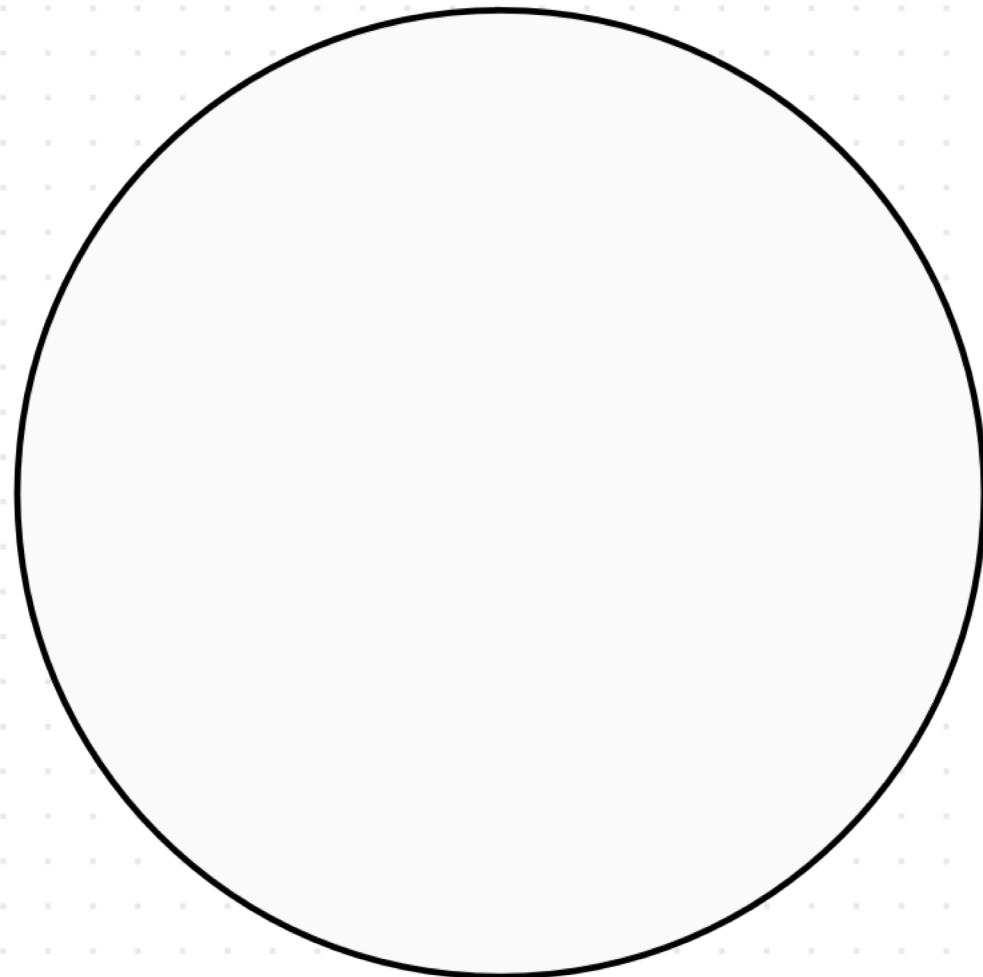
A `WebAssembly.Memory` object represents memory accessed by an `ImportObject`.

JS for Wasm

> **benefits:**  
**macro**

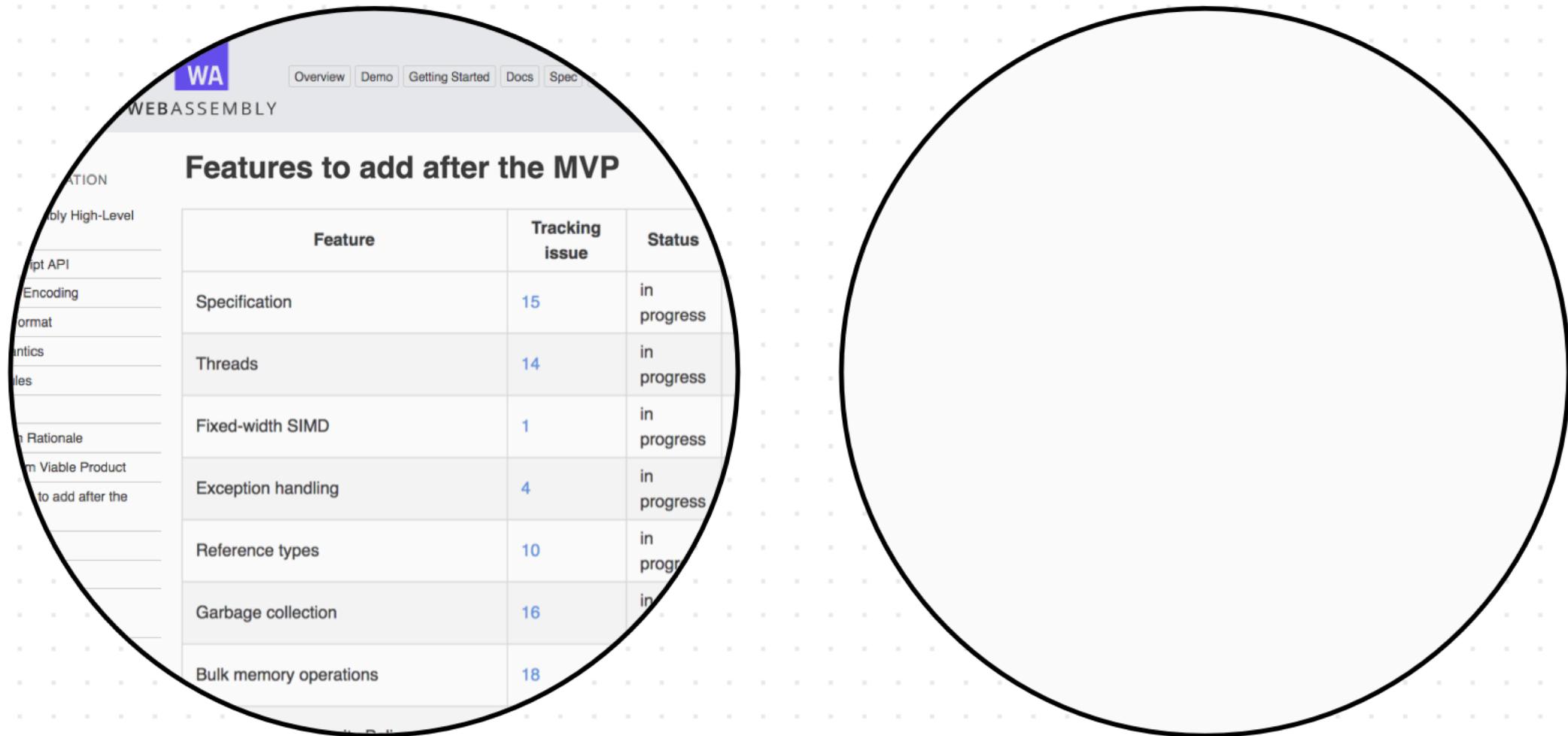


**Wasm for JS**



**JS for Wasm**

## > benefits: macro



# > benefits: macro

The screenshot shows the official WebAssembly website with a navigation bar including 'Overview', 'Demo', 'Getting Started', 'Docs', and 'Spec'. A large black circle highlights the central content area. The title 'Features to add after the MVP' is displayed above a table.

Feature	Tracking issue	Status
Specification	<a href="#">15</a>	in progress
Threads	<a href="#">14</a>	in progress
Fixed-width SIMD	<a href="#">1</a>	in progress
Exception handling	<a href="#">4</a>	in progress
Reference types	<a href="#">10</a>	in progress
Garbage collection	<a href="#">16</a>	in progress
Bulk memory operations	<a href="#">18</a>	in progress

Wasm for JS

@maudnals  
04.19

The screenshot shows the npm search interface with a search bar containing 'wasm'. A large black circle highlights the search results area. The results page displays 675 packages found, with sorting options for 'Optimal', 'Popularity', 'Quality', and 'Maintenance'.

**675 packages found**

Sort Packages

Optimal

Popularity

Quality

Maintenance

**@flash1293/wasm-loader**

Webpack loader for WASM

wasm webassembly webpack loader

**flash1293 published 1.3.0-base64**

walt-compiler

Alternative syntax for WebAssembly text files

wasm wast javascript webassembly

**abuldauskas published 1.0.0**

JS for Wasm



# Symbiosis: Meta

> demo

---

<https://github.com/AssemblyScript/assemblyscript>

<http://localhost:1234>

<https://github.com/maudnals/wasm-as>



Go

> go

---

LEARN  
= fundamentals



EXPERIMENT

USE  
= practice



# > go

LEARN  
= fundamentals



MDN  
o

webassembly  
.studio  
o

## EXPERIMENT

C tutos, AS  
boilerplate,  
rust-wasm  
book  
o

developers  
.google.com  
o

- Replacing a hot path in your JS app with Wasm
- Threads

libs  
o   o   o ...  
react-  
wasm

USE  
= practice





TL;DL



Thank you

QnA time

## > sources

Photos, Videos

<https://www.pexels.com/>

Illustrations, Diagrams

@maudnals

Part 1 –  
JS and Wasm

<v8.dev/blog/liftoff>  
<mathiasbynens.be/notes/shapes-ics>  
<developer.mozilla.org/en-US/docs/WebAssembly>  
<d2jta7o2zej4pf.cloudfront.net/>

Part 2, 3, 4 –  
Symbiosis + Demo

<developers.google.com/web/updates/2019/02/hotpath-with-wasm>  
<webassembly.org/>  
<github.com/AssemblyScript/assemblyscript>

More

<app.sketchup.com/app>  
<blogs.dropbox.com/tech/2018/06/building-better-compression-together-with-divans/>  
<webassembly.org/docs/future-features/>

## > resources

---

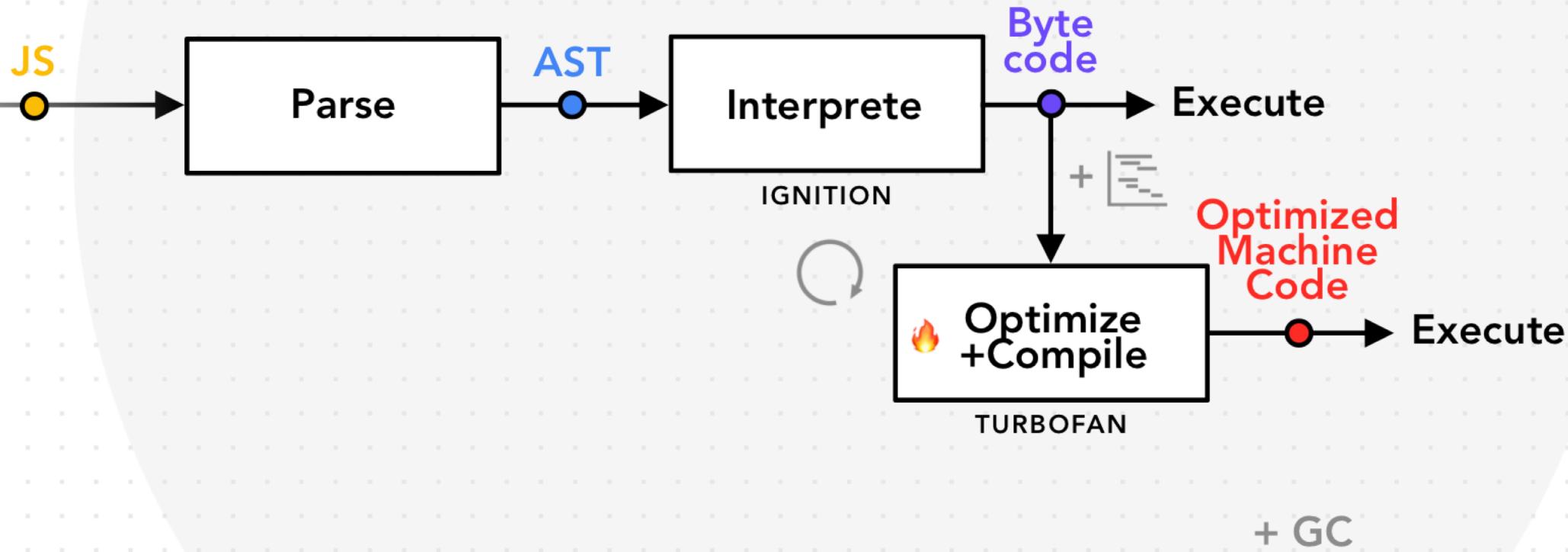
[developers.google.com/web/updates/2019/02/hotpath-with-wasm](https://developers.google.com/web/updates/2019/02/hotpath-with-wasm)  
[developers.google.com/web/updates/2018/03/emscripting-a-c-library](https://developers.google.com/web/updates/2018/03/emscripting-a-c-library)  
[webassembly.org/](https://webassembly.org/)  
[hacks.mozilla.org/](https://hacks.mozilla.org/)  
[rustwasm.github.io/book/](https://rustwasm.github.io/book/)

[github.com/AssemblyScript/assemblyscript](https://github.com/AssemblyScript/assemblyscript)

[github.com/maudnals/wasm-as](https://github.com/maudnals/wasm-as)  
[github.com/maudnals/wasm-nano-handbook](https://github.com/maudnals/wasm-nano-handbook)

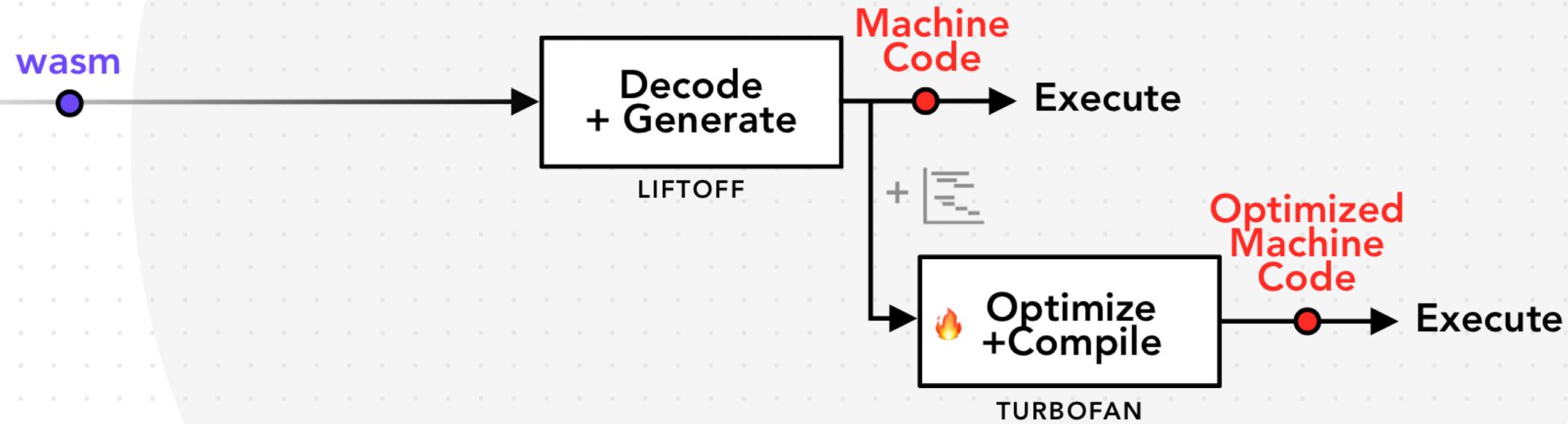
# > how JS runs

JS Engine



# > how Wasm runs

JS+wasm Engine



# > how Wasm runs

JS+wasm Engine

