What Actually Is WebAssembly

Taking a look under the hood

Caleb Schoepp • Cloud Native Rejekts 2023

FERMYON

Caleb Schoepp
Software Engineer
Fermyon

github.com/calebschoepp

calebschoepp.com

FERMYON

Serverless Apps, powered by WebAssembly

FERMYON

Serverless Apps, powered by WebAssembly.

SPIN

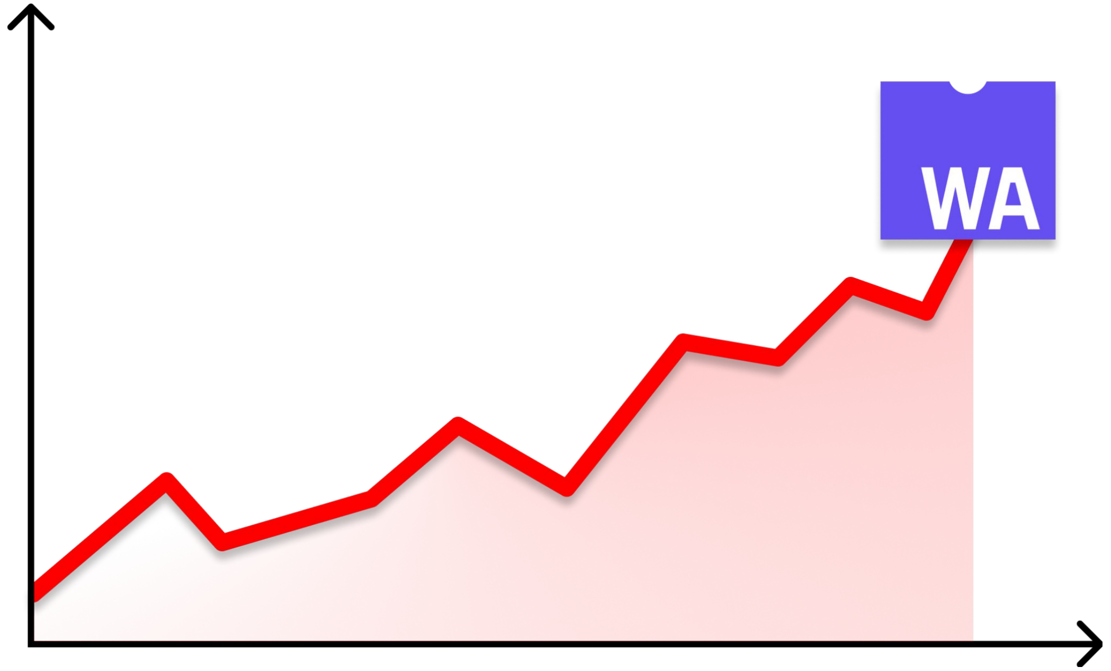
Compose serverless Wasm apps quickly.

FERMYON CLOUD

Deploy and manage serverless Wasm apps.

Some things you’ve probably heard about WebAssembly

WebAssembly is becoming very popular



The textbook definition

“WebAssembly 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.”

WebAssembly has an abbreviation

WebAssembly == Wasm

Wasm is being used everywhere

Browsers, server-side, plugins and more

Wasm has major adoption

Microsoft

fastly®

AUTODESK AutoCAD

Google

Figma

FERMYON

amazon

docker

NETFLIX

ROBLOX

CLOUDFLARE®

People are excited about these four properties

1. Security – Sandboxed execution environment
2. Performance – Near native execution speed
3. Polyglot – Supports a wide array of languages
4. Portability - Cross-platform and cross-architecture

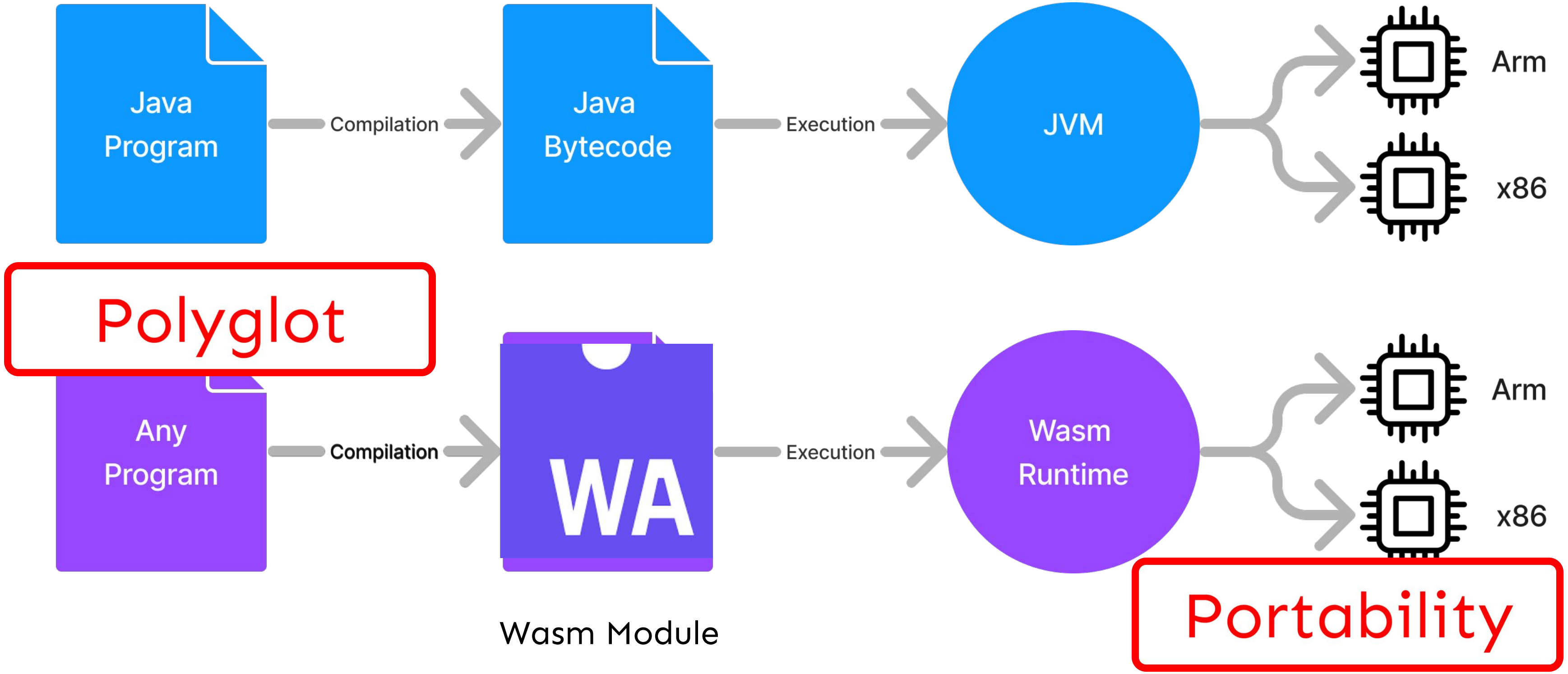
Okay, but what actually is Wasm?

People are excited about these four properties

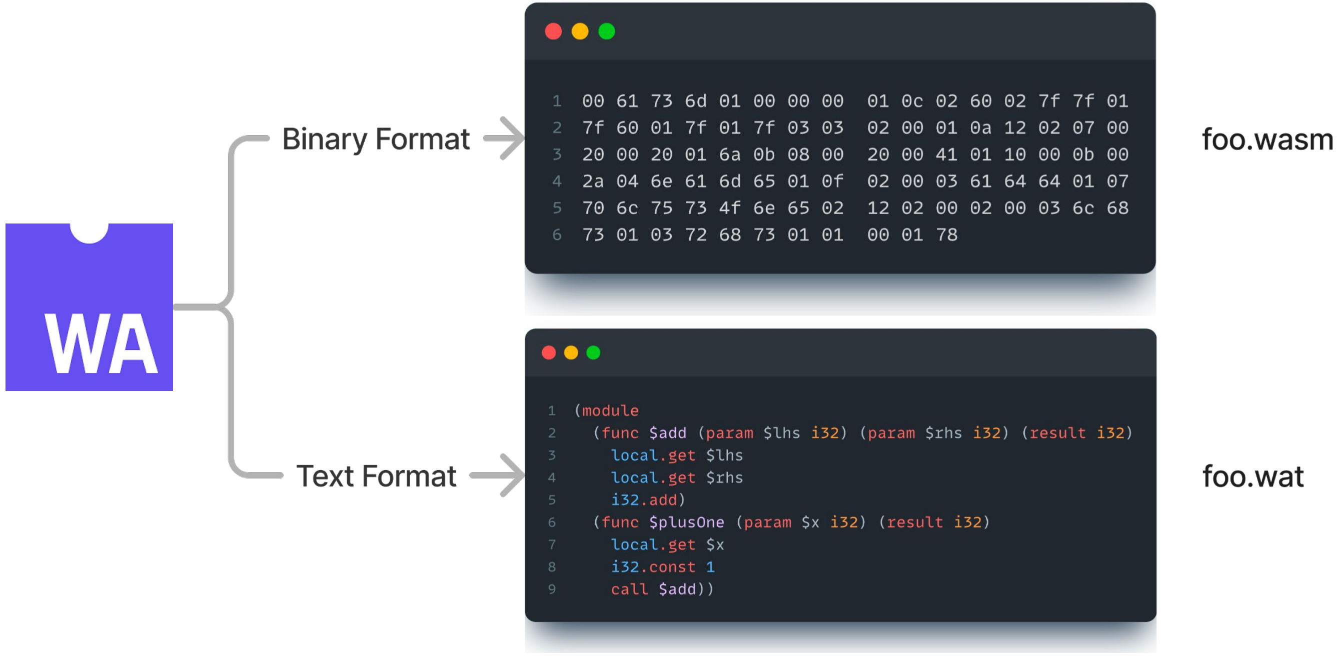
1. Security – Sandboxed execution environment
2. Performance – Near native execution speed
3. Polyglot – Supports a wide array of languages
4. Portability - Cross-platform and cross-architecture

Security	Performance	Polyglot	Portability

Wasm is another bytecode format



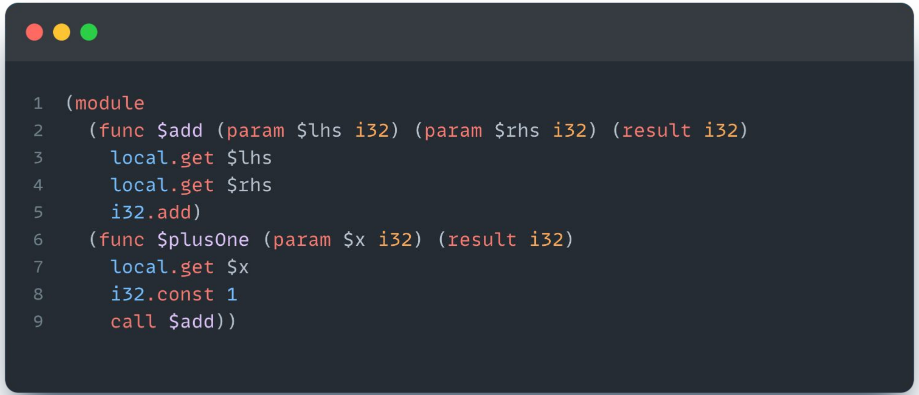
A Wasm module has two representations



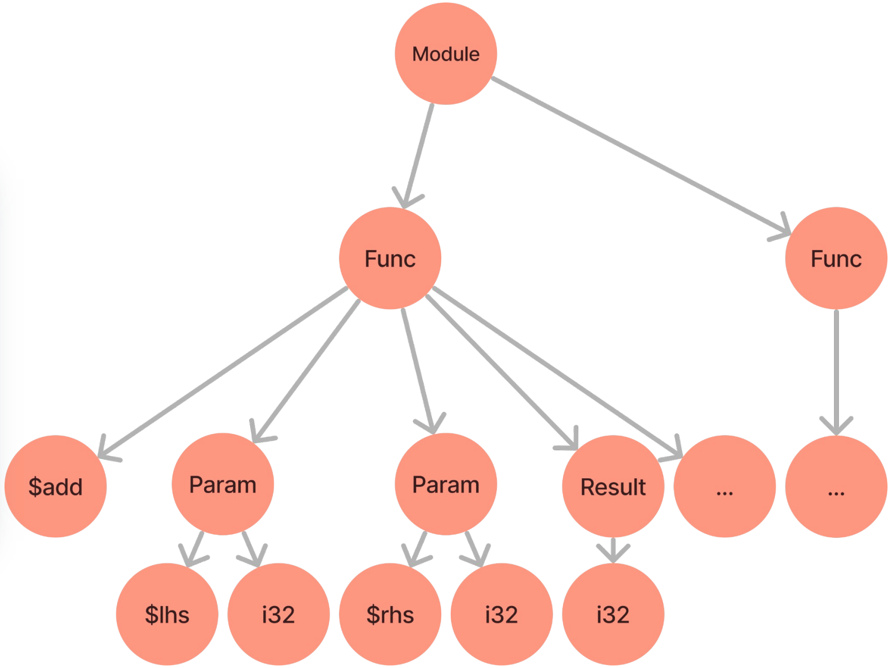
```
00 61 73 6d 01 00 00 00  01 0c 02 60 02 7f 7f 01
7f 60 01 7f 01 7f 03 03  02 00 01 0a 12 02 07 00
20 00 20 01 6a 0b 08 00  20 00 41 01 10 00 0b 00
2a 04 6e 61 6d 65 01 0f  02 00 03 61 64 64 01 07
70 6c 75 73 4f 6e 65 02  12 02 00 02 00 03 6c 68
73 01 03 72 68 73 01 01  00 01 78
```

```
(module
  (func $add (param $lhs i32) (param $rhs i32) (result i32)
    local.get $lhs
    local.get $rhs
    i32.add)
  (func $plusOne (param $x i32) (result i32)
    local.get $x
    i32.const 1
    call $add))
```

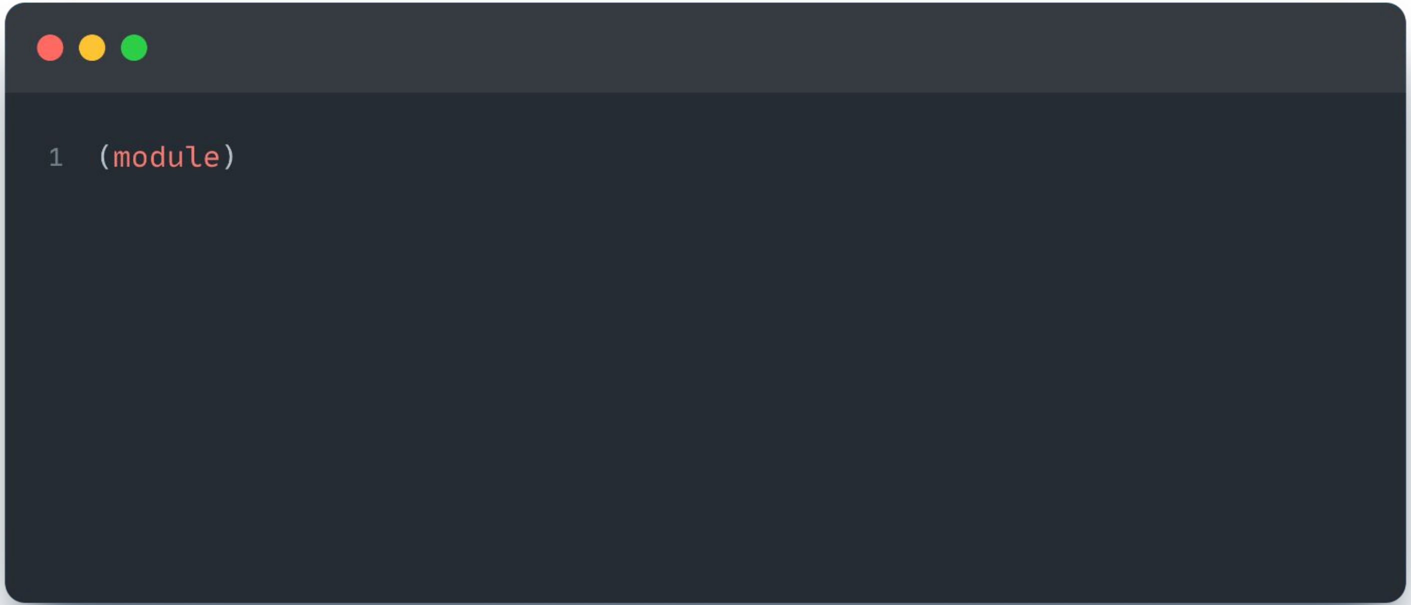
The text format uses s-expressions



```
(module
  (func $add (param $lhs i32) (param $rhs i32) (result i32)
    local.get $lhs
    local.get $rhs
    i32.add)
  (func $plusOne (param $x i32) (result i32)
    local.get $x
    i32.const 1
    call $add))
```

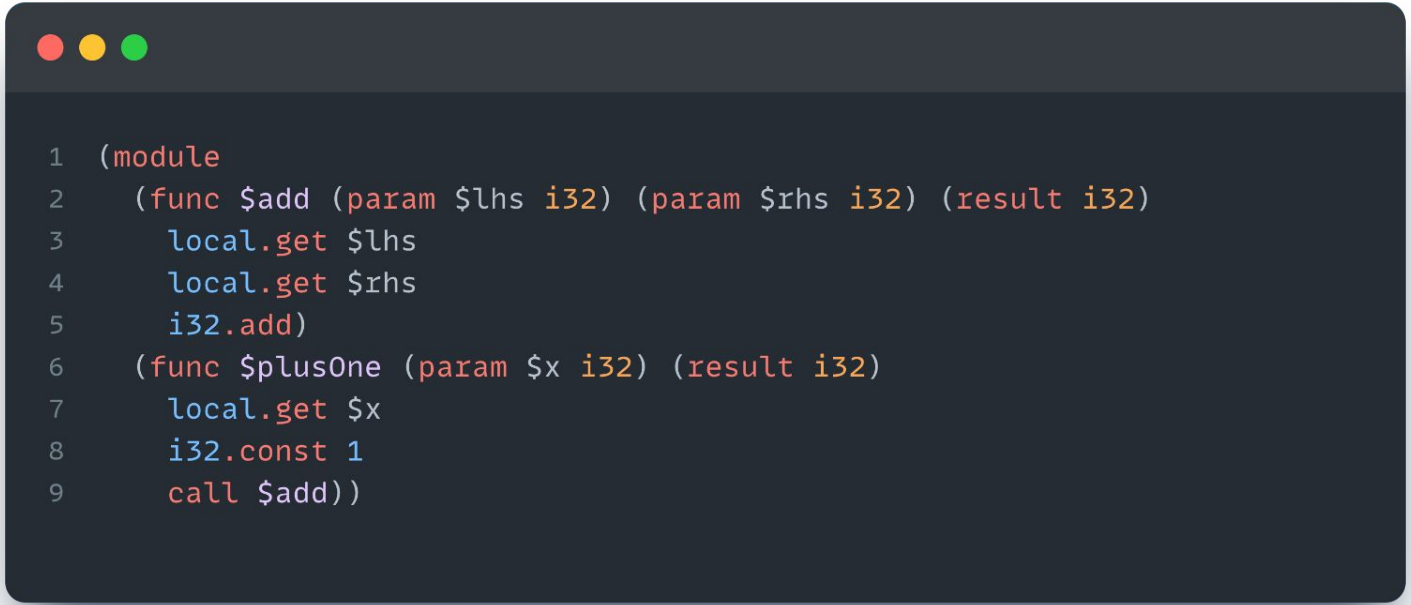


The most basic Wasm module

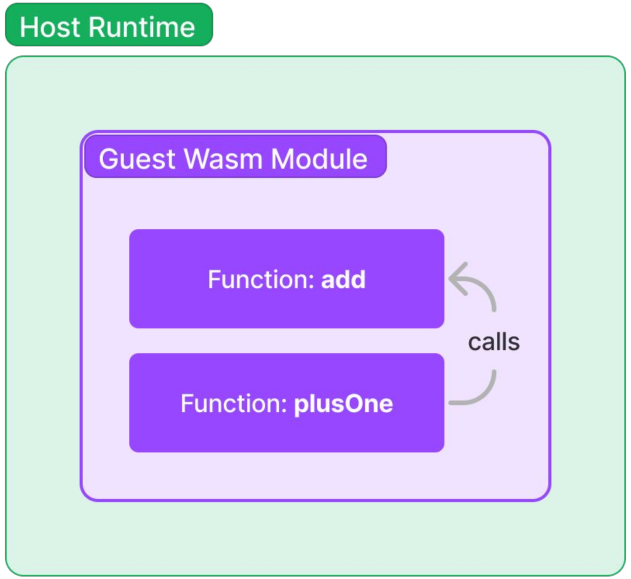


```
(module)
```

A Wasm module has functions

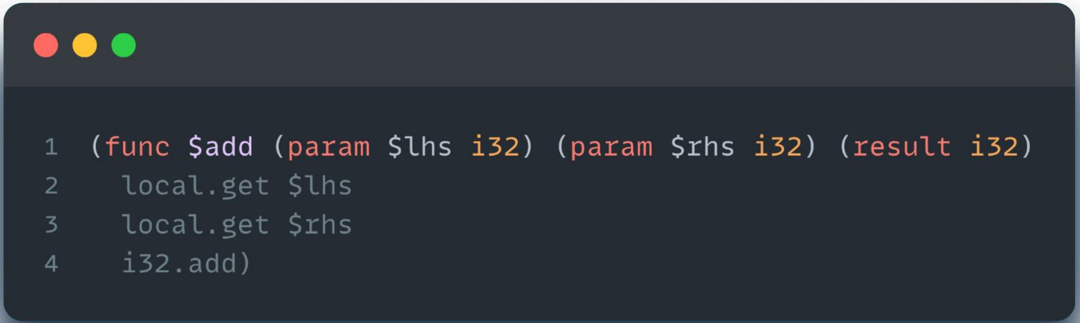


```
(module
  (func $add (param $lhs i32) (param $rhs i32) (result i32)
    local.get $lhs
    local.get $rhs
    i32.add)
  (func $plusOne (param $x i32) (result i32)
    local.get $x
    i32.const 1
    call $add))
```


Host Runtime
Guest Wasm Module
Function: add
calls
Function: plusOne

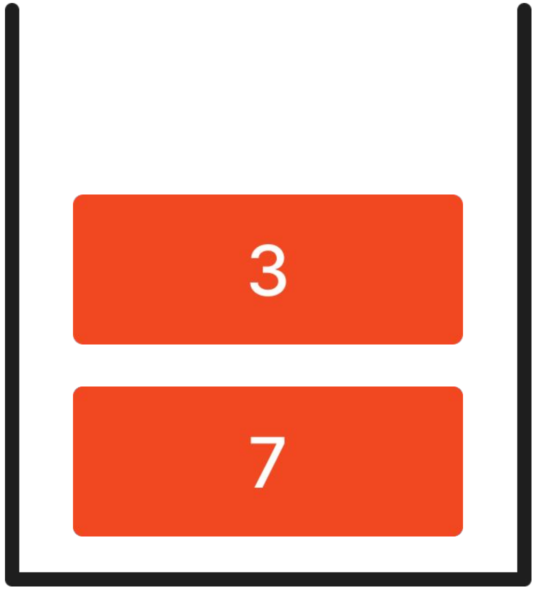
Wasm runs on a stack machine

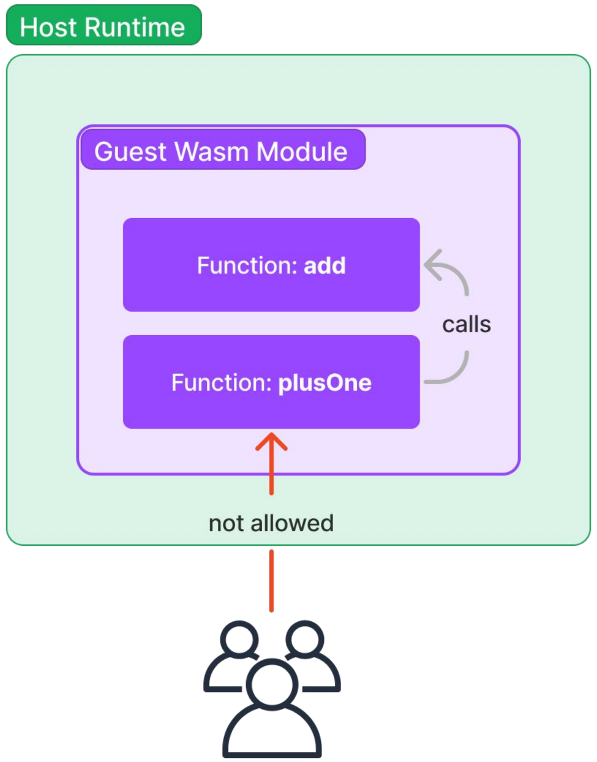
Stack machine example of add(4,3)



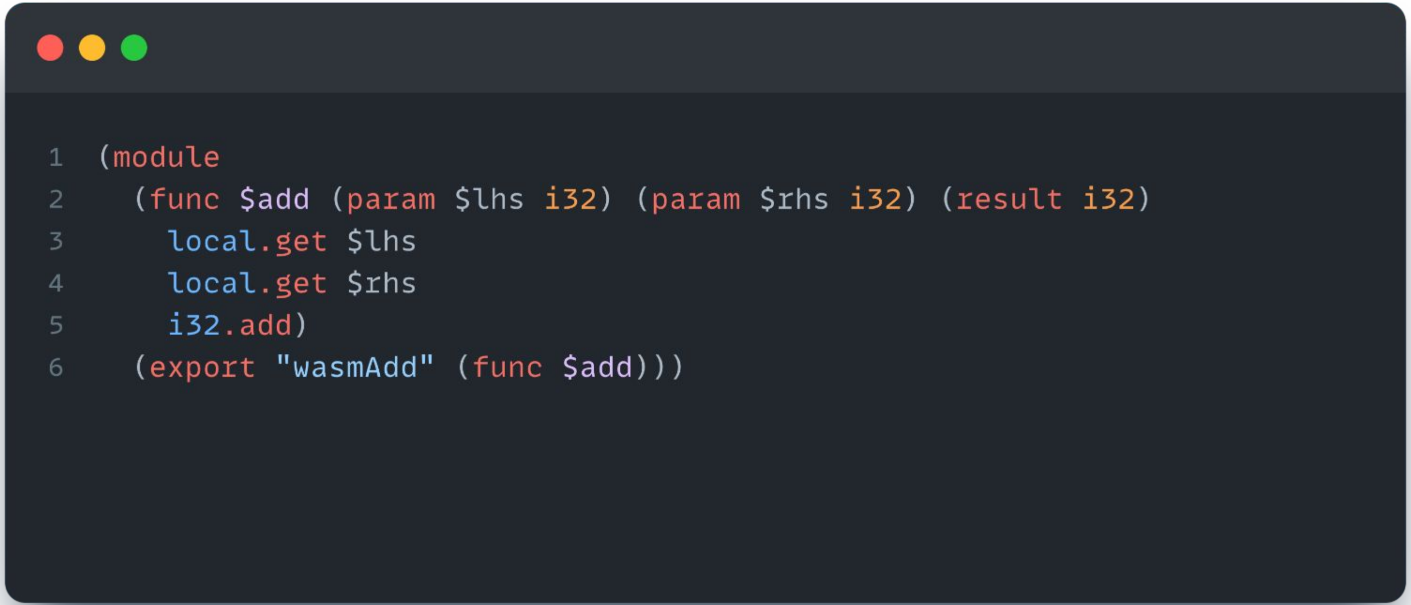
```
(func $add (param $lhs i32) (param $rhs i32) (result i32)
  local.get $lhs
  local.get $rhs
  i32.add)
```

Performance

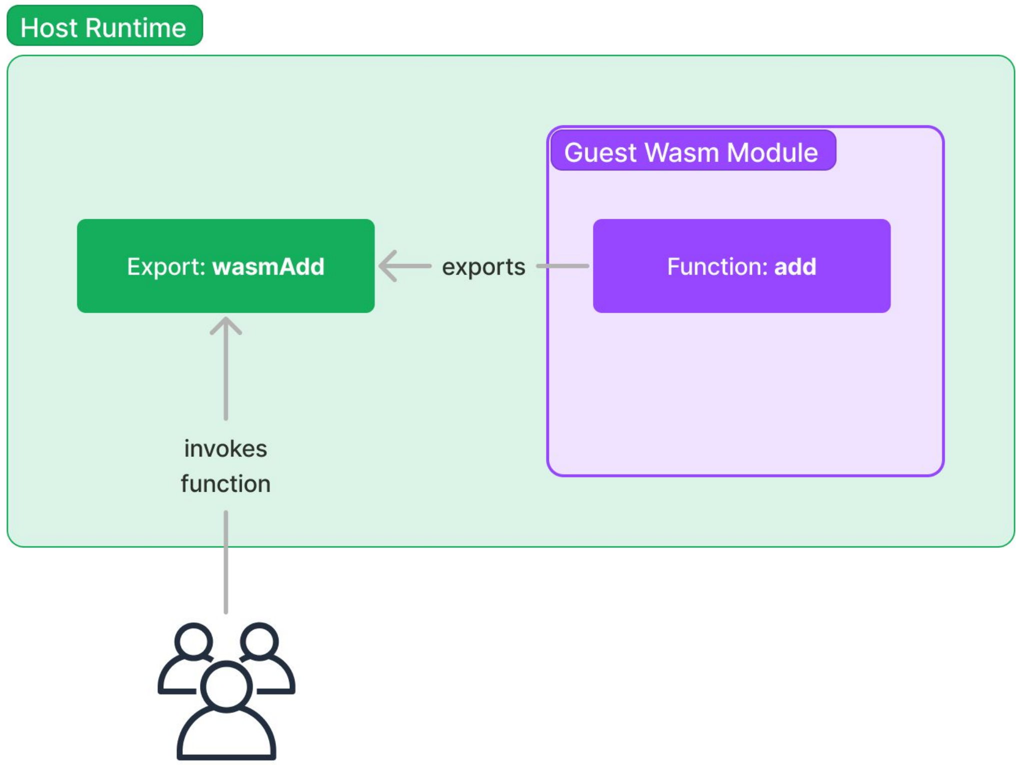



Host Runtime
Guest Wasm Module
Function: **add**
calls
Function: **plusOne**
not allowed

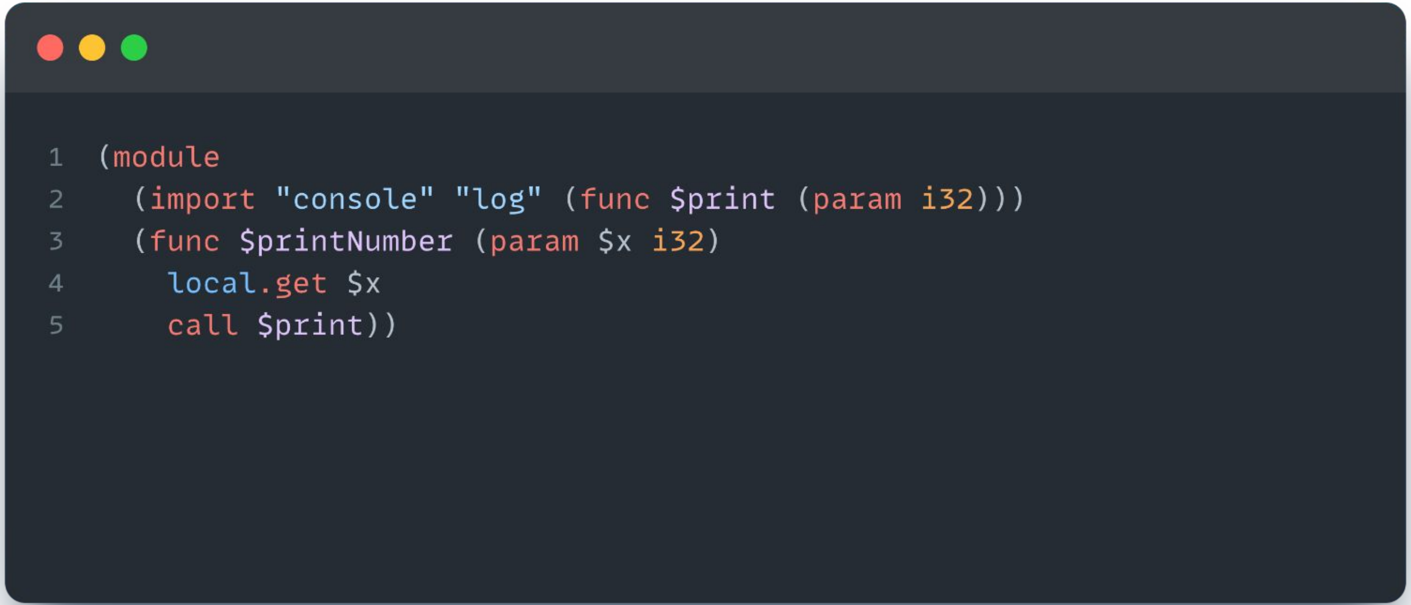
Wasm let’s you export functionality



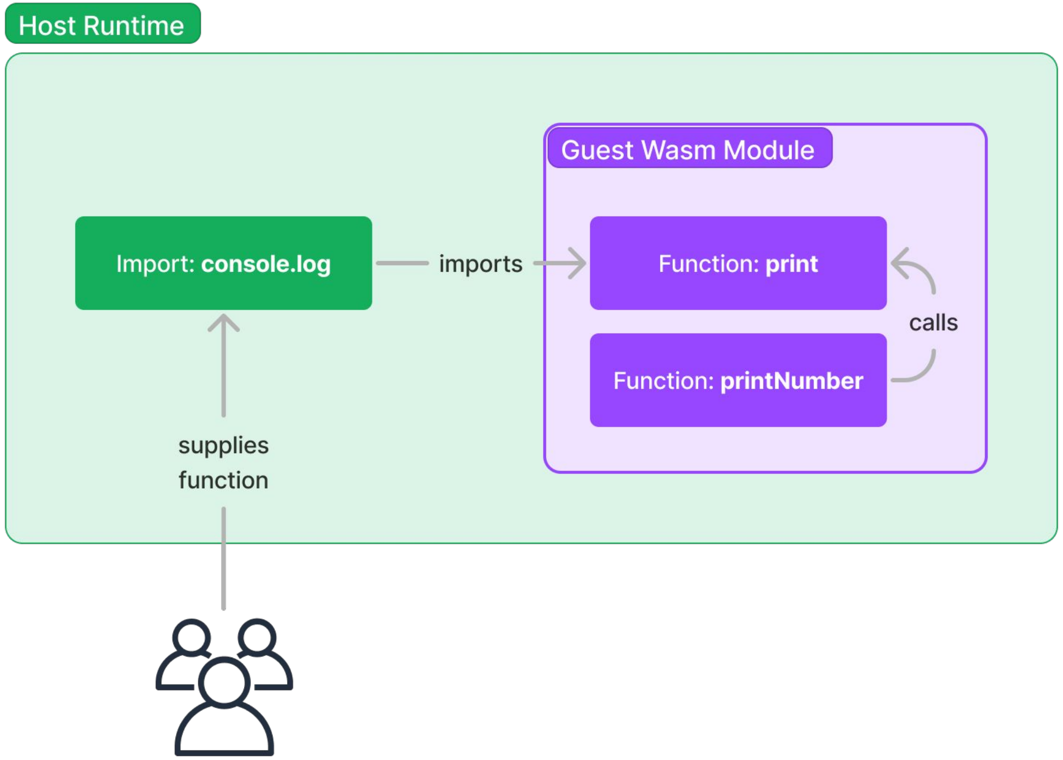
```
(module
  (func $add (param $lhs i32) (param $rhs i32) (result i32)
    local.get $lhs
    local.get $rhs
    i32.add)
  (export "wasmAdd" (func $add)))
```


Host Runtime
Guest Wasm Module
Export: wasmAdd
exports
Function: add
invokes function

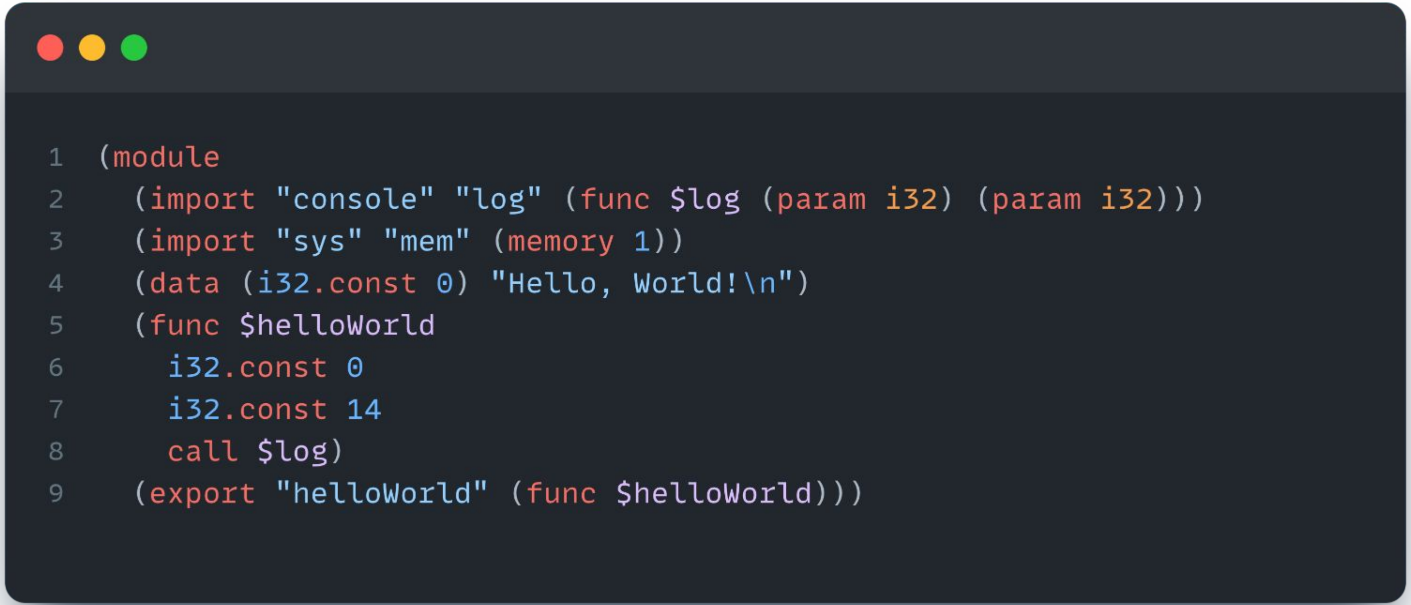
And it let’s you import functionality



```
(module
  (import "console" "log" (func $print (param i32)))
  (func $printNumber (param $x i32)
    local.get $x
    call $print))
```

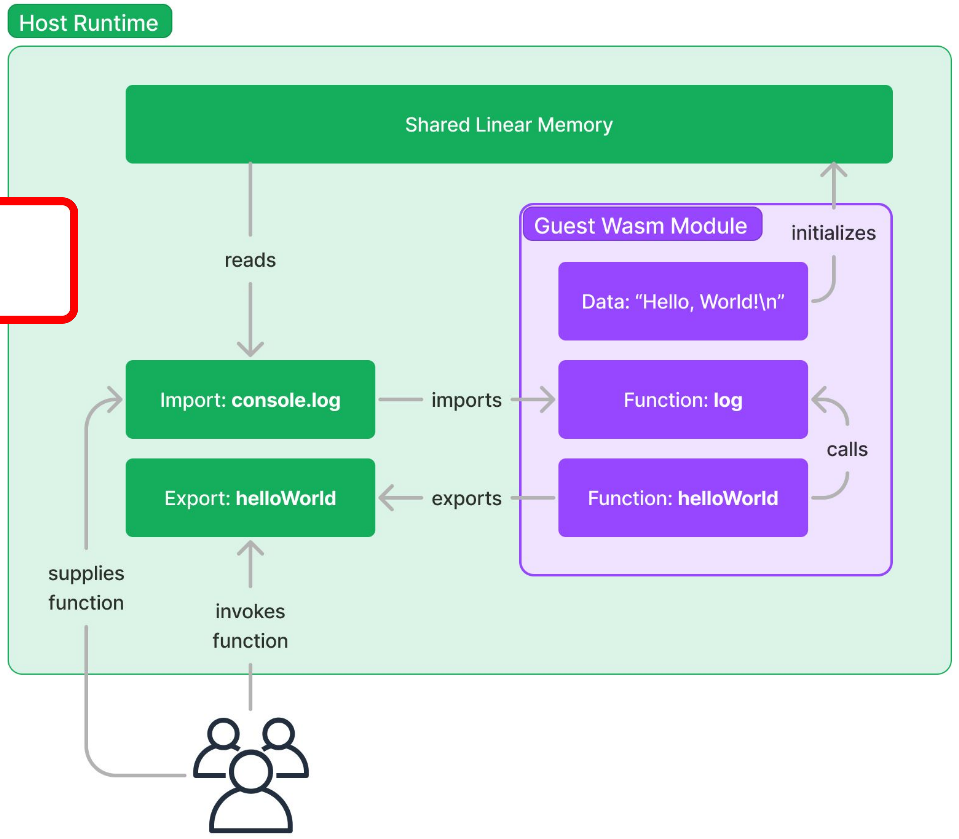

Host Runtime
Guest Wasm Module
Import: **console.log**
imports
Function: **print**
calls
Function: **printNumber**
supplies function

Wasm has shared linear memory



```
(module
  (import "console" "log" (func $log (param i32) (param i32)))
  (import "sys" "mem" (memory 1))
  (data (i32.const 0) "Hello, World!\n")
  (func $helloWorld
    i32.const 0
    i32.const 14
    call $log)
  (export "helloWorld" (func $helloWorld)))
```

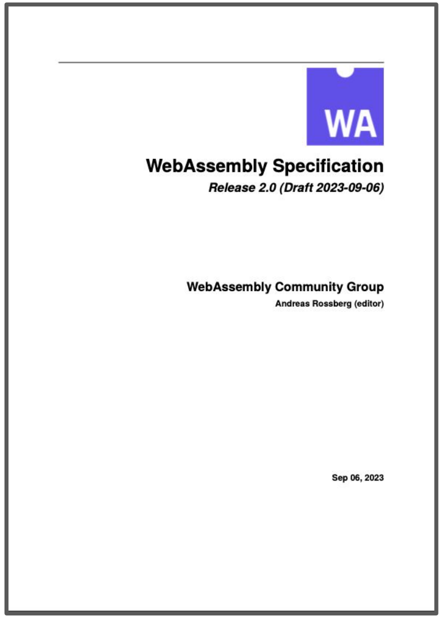
Security


Host Runtime
Shared Linear Memory
reads
Guest Wasm Module
initializes
Data: "Hello, World!\n"
Import: console.log
imports
Function: log
calls
Export: helloWorld
exports
Function: helloWorld
supplies function
invokes function

These are just the basics

MDN web docs

moz://a



WebAssembly Specification

Release 2.0 (Draft 2023-09-06)

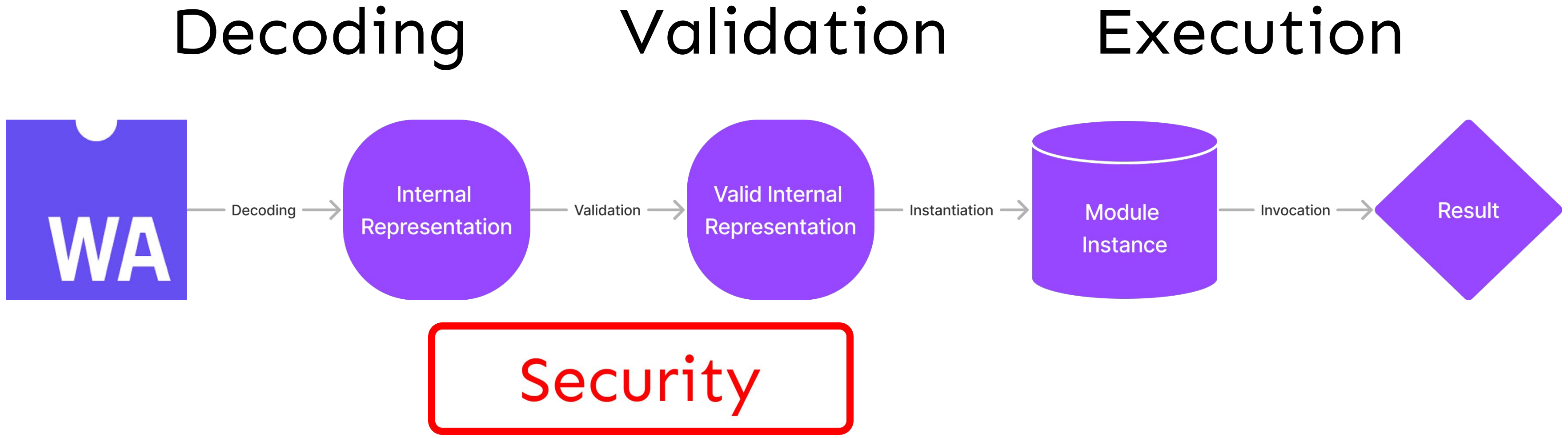
WebAssembly Community Group

Andreas Rossberg (editor)

Sep 06, 2023

How does a host runtime execute my Wasm?

The three semantic phases



Some popular Wasm runtimes

Browser

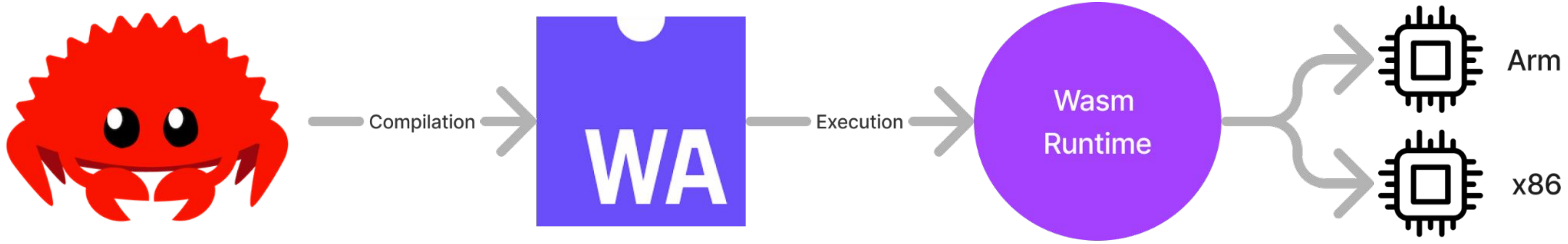
Wasmtime

Wasm3

Portability

How do I compile my code to Wasm?

Rust has great Wasm support



C/C++ has good support too

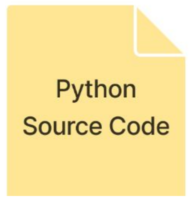
emscripten

ZIG

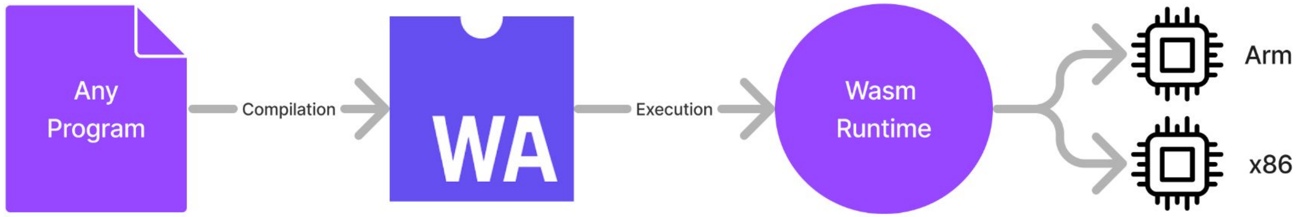
LLVM
COMPILER INFRASTRUCTURE

Interpreted languages are a little more tricky

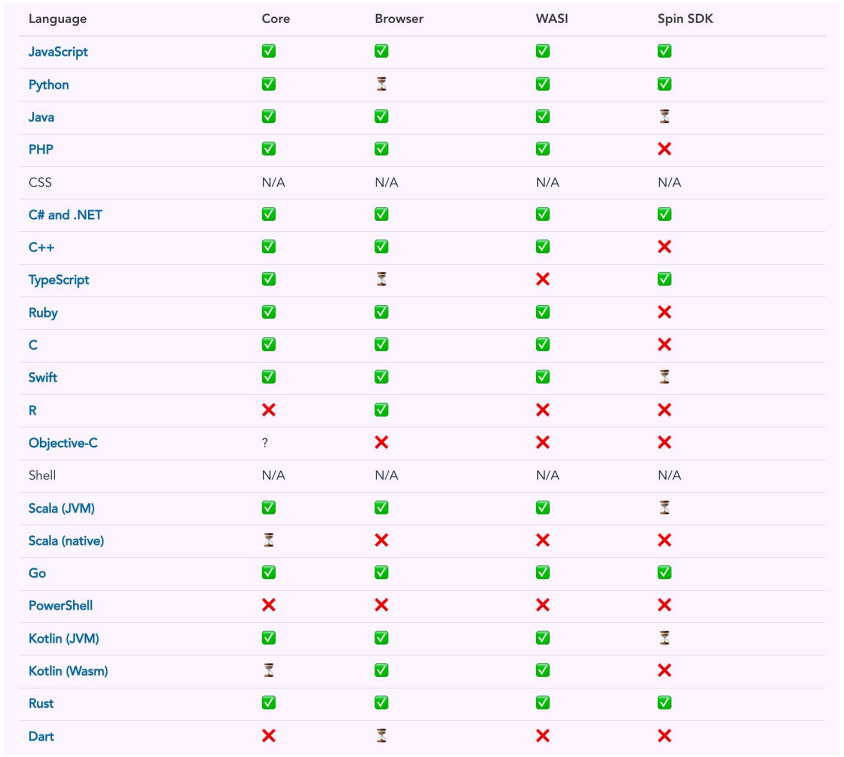
You need to compile the interpreter to Wasm



Polyglot



WebAssembly Language Support Matrix



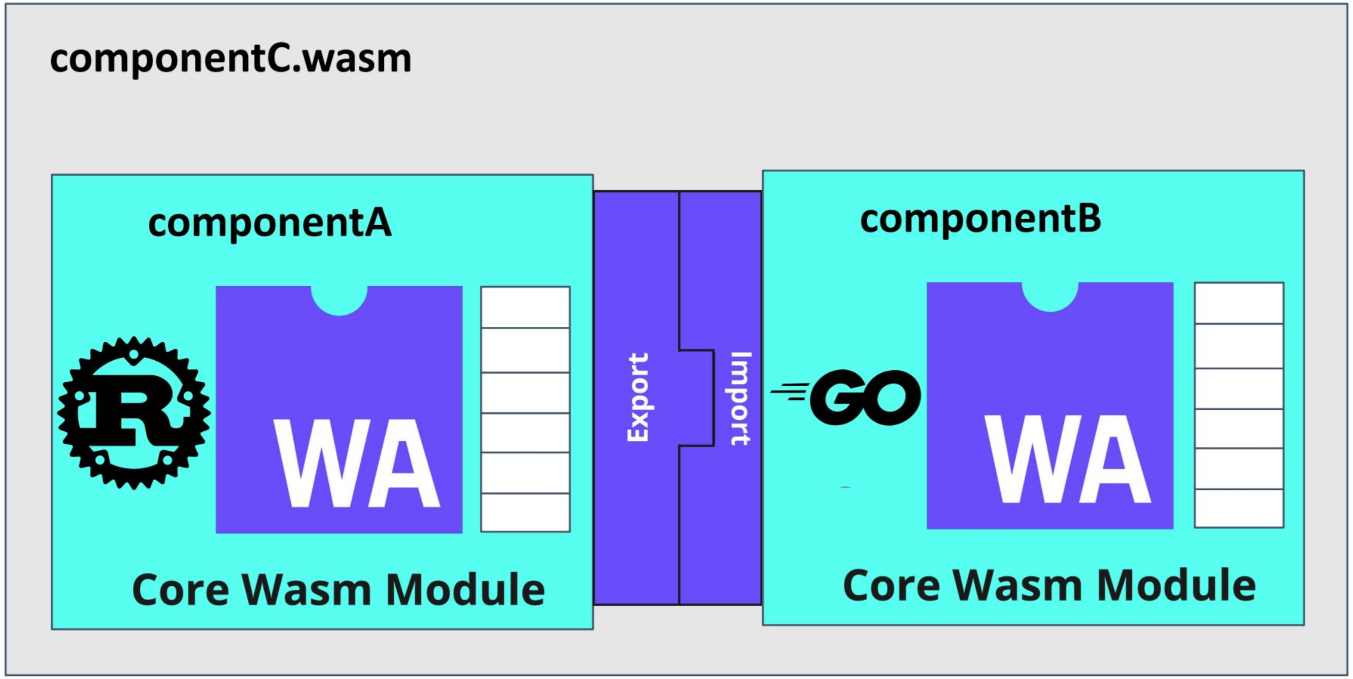
Language	Core	Browser	WASI	Spin SDK
JavaScript	✅	✅	✅	✅
Python	✅	⏳	✅	✅
Java	✅	✅	✅	⏳
PHP	✅	✅	✅	❌
CSS	N/A	N/A	N/A	N/A
C# and .NET	✅	✅	✅	✅
C++	✅	✅	✅	❌
TypeScript	✅	⏳	❌	✅
Ruby	✅	✅	✅	❌
C	✅	✅	✅	❌
Swift	✅	✅	✅	⏳
R	❌	✅	❌	❌
Objective-C	?	❌	❌	❌
Shell	N/A	N/A	N/A	N/A
Scala (JVM)	✅	✅	✅	⏳
Scala (native)	⏳	❌	❌	❌
Go	✅	✅	✅	✅
PowerShell	❌	❌	❌	❌
Kotlin (JVM)	✅	✅	✅	⏳
Kotlin (Wasm)	⏳	✅	✅	❌
Rust	✅	✅	✅	✅
Dart	❌	⏳	❌	❌

What is WASI and the Component Model?

WASI Preview 1

WASI

WebAssembly Component Model



WASI Preview 2

How can I actually use WebAssembly?

SPIN

The framework to compose serverless WebAssembly apps quickly

FERMYON

Cloud

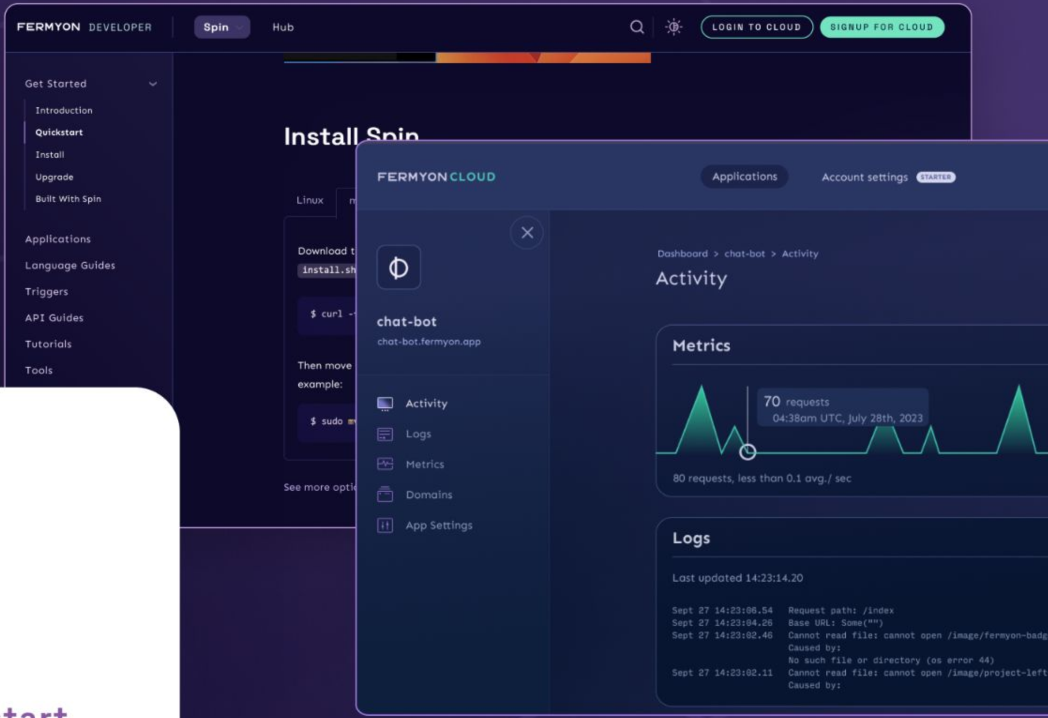
The quickest way to deploy and manage your serverless WebAssembly apps

SpinKube

Hyper-efficient serverless on Kubernetes, powered by WebAssembly

NEXT STEPS

FERMYON



Download t
`install.sh`

```
$ curl -
```

Then move
example:

```
$ sudo m
```

Last updated 14:23:14.20

```
Sept 27 14:23:06.54   Request path: /index
Sept 27 14:23:04.26   Base URL: Some("")
Sept 27 14:23:02.46   Cannot read file: cannot open /image/fermyon-badg
                      Caused by:
                      No such file or directory (os error 44)
Sept 27 14:23:02.11   Cannot read file: cannot open /image/project-left
                      Caused by:
```

Quickstart

Go from blinking cursor to deployed serverless app in 66 seconds.

https://developer.fermyon.com/spin/quickstart

Thank you!

Spin Quickstart