Philippe Charrière

Customer Success Engineer @ GitLab + Golang 🐣 dev.

Give super powers to your Golang applications

with WebAssembly and Extism+Wazero

Agenda

- WASM?
- WASI?
  - Demo
- Limitations
- Wazero
  - Demo
- Extism
  - Demo

👋 *Some parts of this talk have already been discussed this morning by Francesco Romani*

WebAssembly?

WebAssembly (or Wasm)?

WASM is the nickname for WebAssembly

https://webassembly.org

WA

- Code > Bytecode (wasm binary file)
- Binary format for executing code on the Web
- The JavaScript VM is responsible for the execution of the WASM code
- WASM is polyglot
- WASM is **safe**

Why WASM?

WASM is the nickname for WebAssembly

- A complement to JavaScript
- Near-native speeds
- Complex applications in the browser

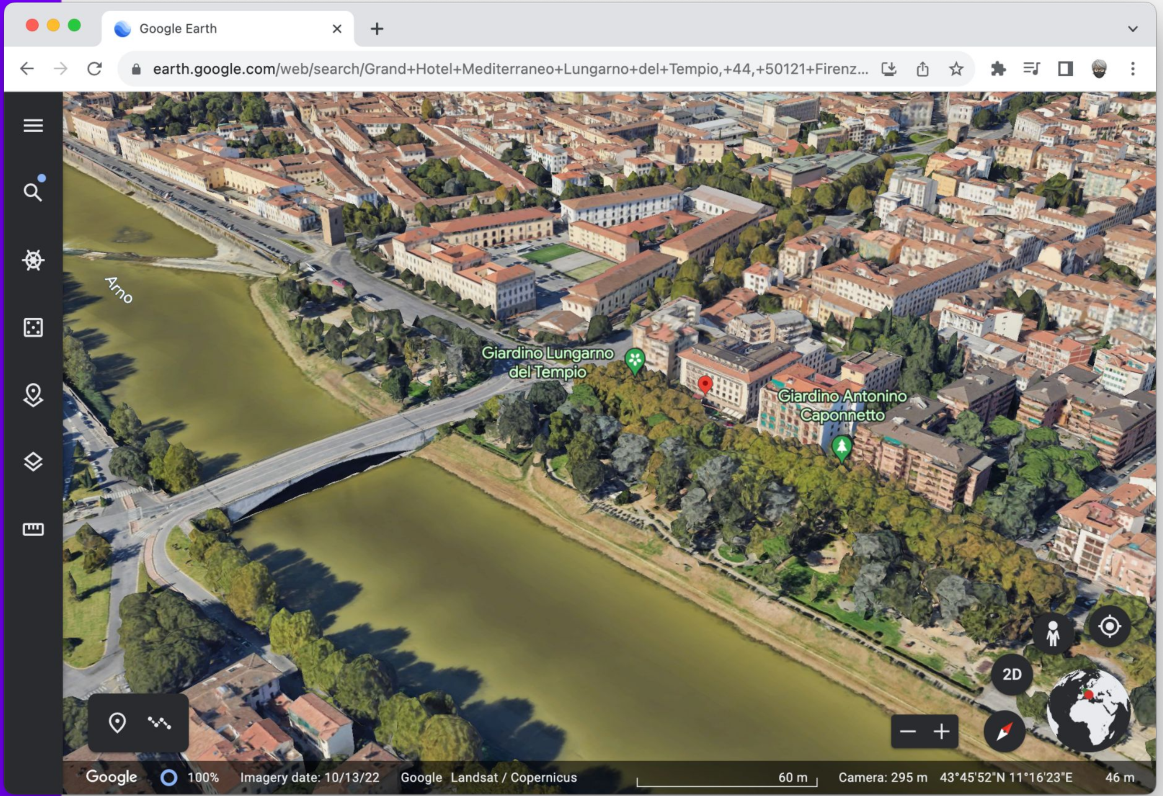
WebAssembly in the browser is amazing

Give super powers to your Golang applications

WebAssembly in the browser is amazing.

Google Earth

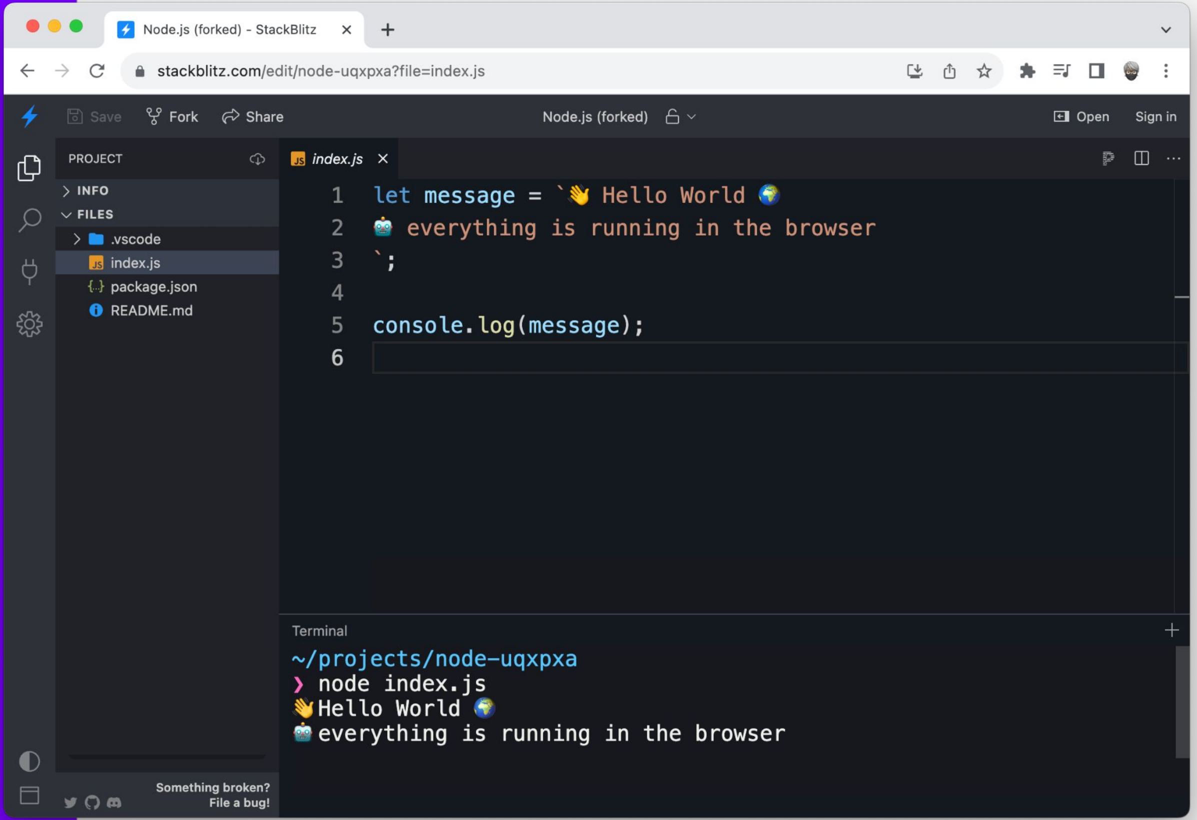
https://earth.google.com


Google Earth
earth.google.com/web/search/Grand+Hotel+Mediterraneo+Lungarno+del+Tempio,+44,+50121+Firenz...
Arno
Giardino Lungarno del Tempio
Giardino Antonino Caponnetto
2D
Google
100%
Imagery date: 10/13/22
Google Landsat / Copernicus
60 m
Camera: 295 m 43°45'52"N 11°16'23"E
46 m

https://stackblitz.com/edit/node-sea49e?file=index.js

WebAssembly in the browser is amazing.

Stackblitz


Node.js (forked) - StackBlitz
stackblitz.com/edit/node-uqxpxa?file=index.js
Save
Fork
Share
Node.js (forked)
Open
Sign in
PROJECT
INFO
FILES
.vscode
index.js
package.json
README.md
Something broken? File a bug!
Terminal

```js
let message = `👋 Hello World 🌍
🤖 everything is running in the browser
`;

console.log(message);
```

```
~/projects/node-uqxpxa
❯ node index.js
👋Hello World 🌍
🤖everything is running in the browser
```

The primary qualities of WASM

- Speed
- Efficiency
- Safe
- Versatile
- Portable

Give super powers to your Golang applications

Free WASM from the browser

Let it Go!

WASI?

WebAssembly System Interface

WASI?

https://wasi.dev

- WebAssembly System Interface
- Interface between
    - WebAssembly (WASM) code
    - And a Runtime environment
- Allowing WASM code to be run in various contexts

Some WASI Use Cases

- **CLI applications**
- **Applications with plug-ins** (Zellij, Lapce)
- **Database UDF** (ScyllaDB, PostgreSQL)
- **WebHooks, Filters, …** (Webhook Relay, Envoy)
- **FaaS** (Fermyon cloud, WasmCloud, Shopify, …)
- **…**

At least, 3 ways to run Wasm programs outside the browser

- WASI Runtimes **CLI**
- WASI Runtimes **SDK**
- Ready to use applications with embedded Wasm runtime
  - Spin from Fermyon
  - Wasm Workers Server from Wasm Lab
  - …

WASI Runtimes

- WasmEdge,
- Wasmtime,
- Wasmer,
- Wazero □,
- NodeJS,
- …

Demo

01-first-wasm-program

✋ I use code snippets because I have no memory
🥰 I use **TinyGo** to build the wasi programs

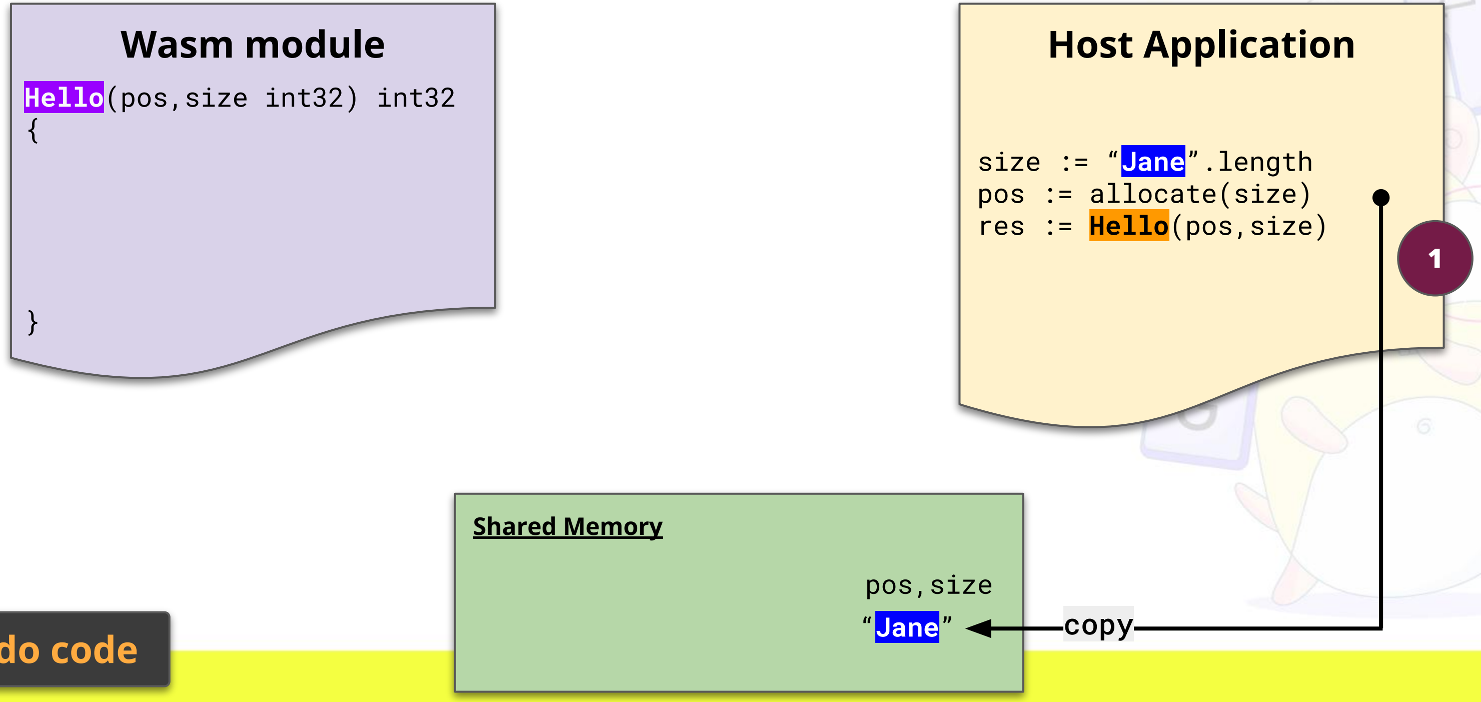
Some limitations

One of the "annoying" limitations

- Only numbers 😮
- How to pass string arguments to a Wasm function?
- How to return a string as the result of a Wasm function call?

Solution:
Exchange data with the
Shared Memory Buffer

Copy the string to the memory (position & size), Then, call Hello(pos, size)



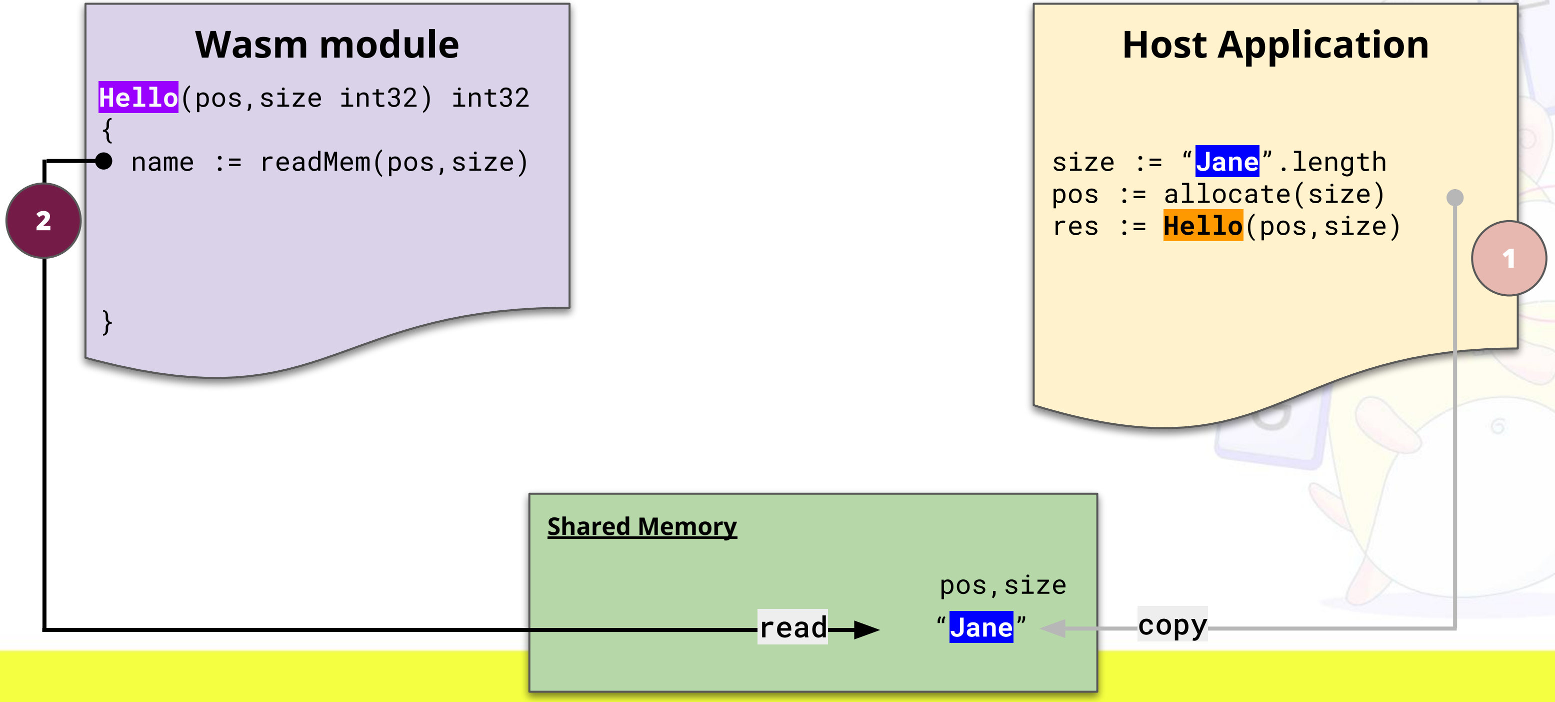
```
Hello(pos,size int32) int32
{

}
```

```
size := “Jane”.length
pos := allocate(size)
res := Hello(pos,size)
```

✋pseudo code

Hello can read the string into the memory with pos & size

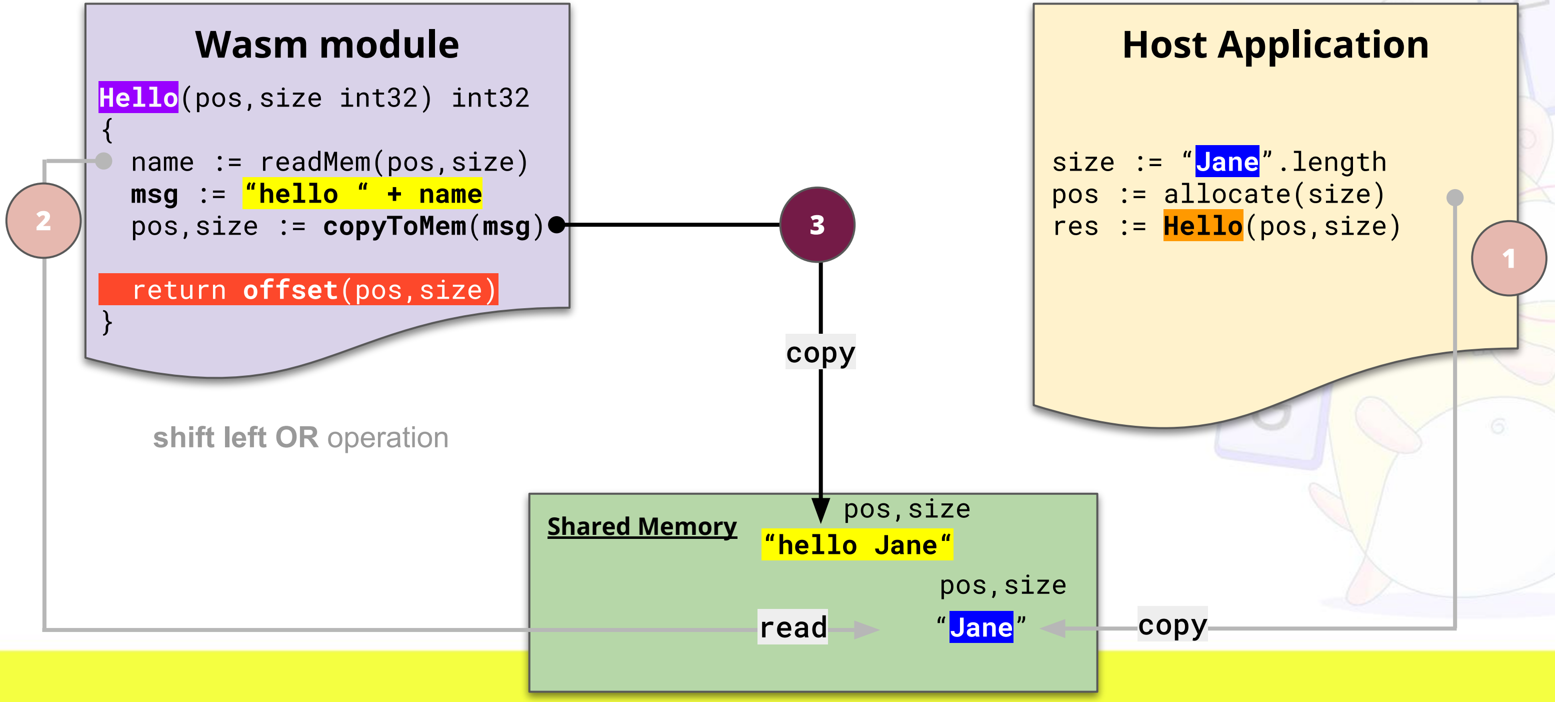


```
Hello(pos,size int32) int32
{
  name := readMem(pos,size)


}
```

```
size := "Jane".length
pos := allocate(size)
res := Hello(pos,size)
```

Hello can copy a string into the memory and return the pos & size

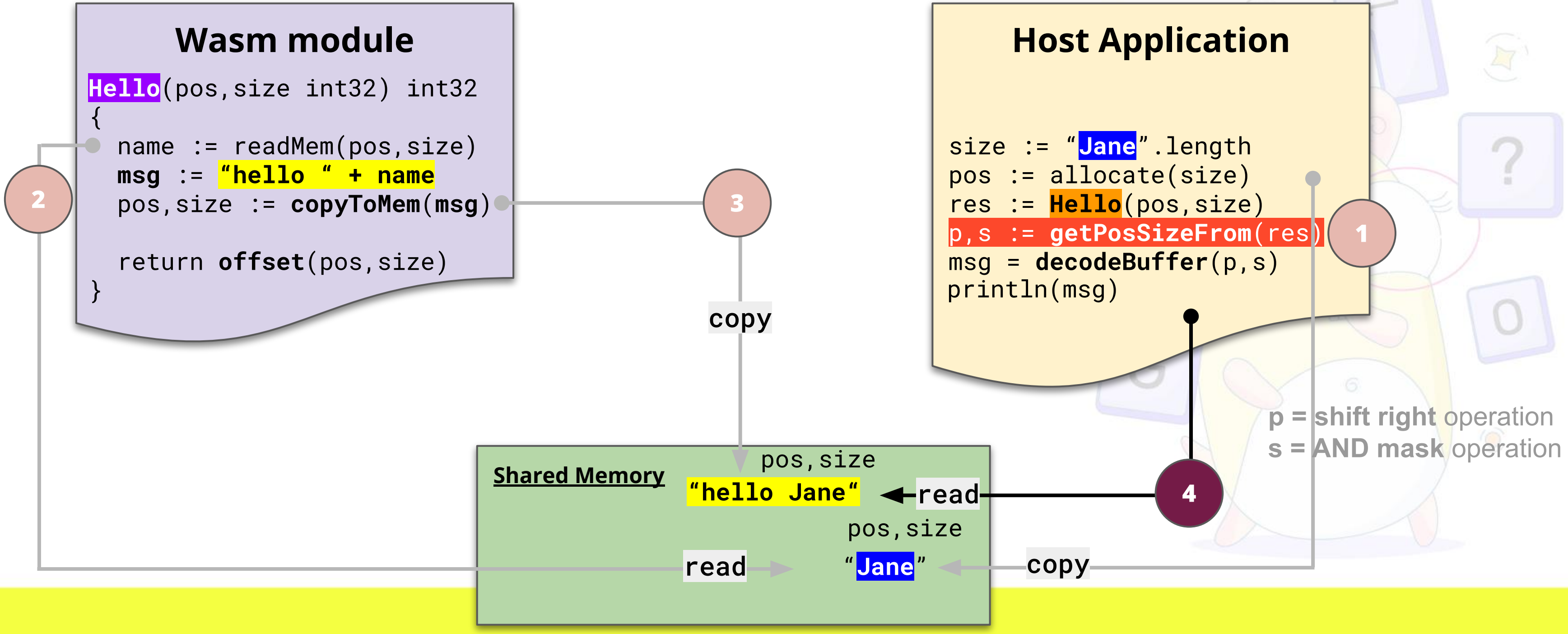


```
Hello(pos,size int32) int32
{
  name := readMem(pos,size)
  msg  := "hello " + name
  pos,size := copyToMem(msg)

  return offset(pos,size)
}
```

```
size := "Jane".length
pos := allocate(size)
res := Hello(pos,size)
```

Then the Host Application can read the pos & size and decode the buffer memory



```
Hello(pos,size int32) int32
{
  name := readMem(pos,size)
  msg := "hello " + name
  pos,size := copyToMem(msg)

  return offset(pos,size)
}
```

```
size := "Jane".length
pos := allocate(size)
res := Hello(pos,size)
p,s := getPosSizeFrom(res)
msg = decodeBuffer(p,s)
println(msg)
```

p = shift right operation
s = AND mask operation

Give super powers to your Golang applications

Wazero

the zero dependency WebAssembly runtime for Go developers

- You can develop your own **CLI**
- But, you need to handle the limitations
- Develop all the “plumbing”

Solution: **Wazero** Runtime □ & SDK
https://wazero.io

Demo

02-wazero

Write your 1st CLI 🚀

But, sometimes, you need more

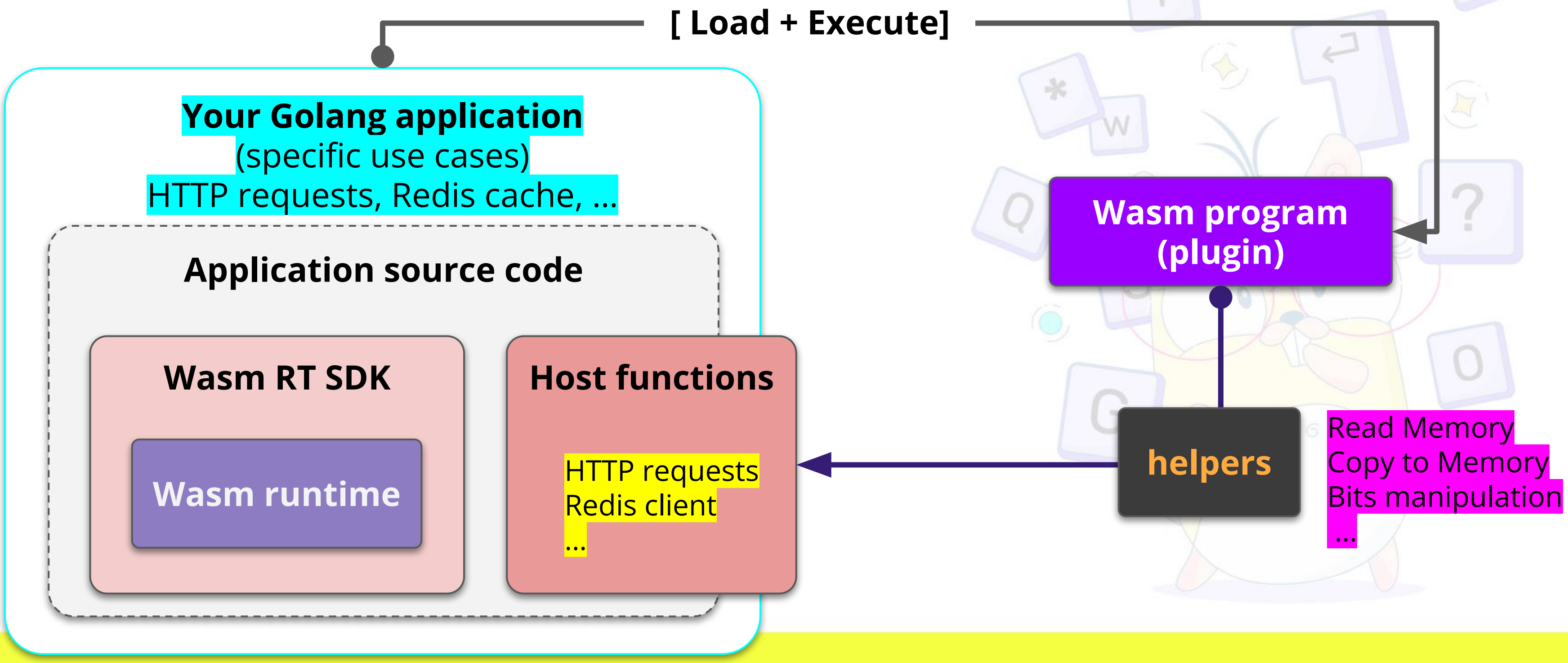
- Make HTTP requests
- Make Redis requests from the Wasm module
- Use MQTT or NATS
- …

Solution: **Host Functions**

Host Function?

- A function defined in the Host application
- For The Wasm program, it’s used as an import function

Your Golang application + Host functions



“Helpers”, but…

- ✋You need to write your own glue
- For every language you want to support on the Wasm side 😵‍💫

🤬 It’s complicated! But…

Give super powers to your Golang applications

There is another way (easier) 👀

The cross-language framework for building with WebAssembly

Extism is a plug-in system for everyone.

dylibso

Browser / JS
C
C++
.NET
Elixir / Erlang
Go
Haskell
Java
Node
OCaml
PHP
Python
Ruby
Rust
Zig

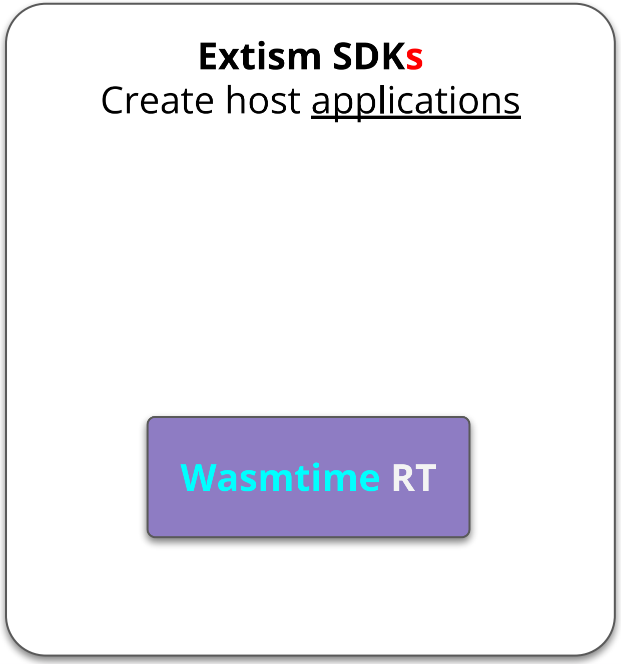
Extism **S**DKs + **P**DKs

Rust
JavaScript
Go
Haskell
AssemblyScript
C
Zig

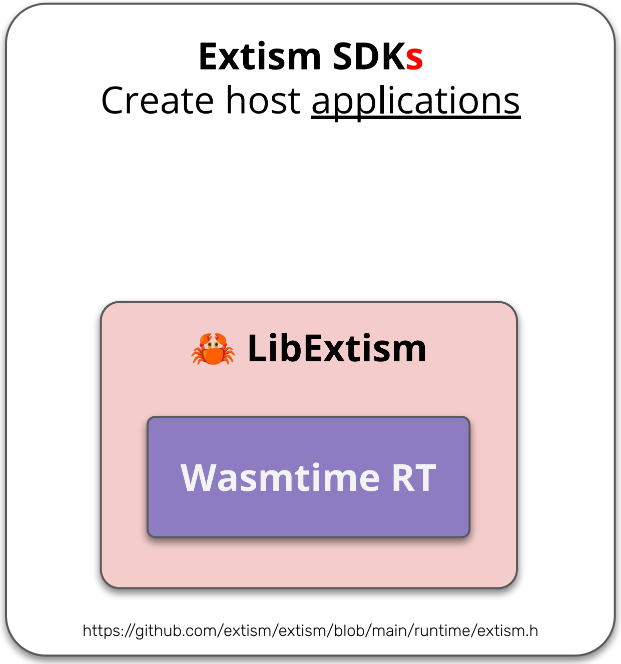
Extism SDKs

https://extism.org/docs/category/integrate-into-your-codebase

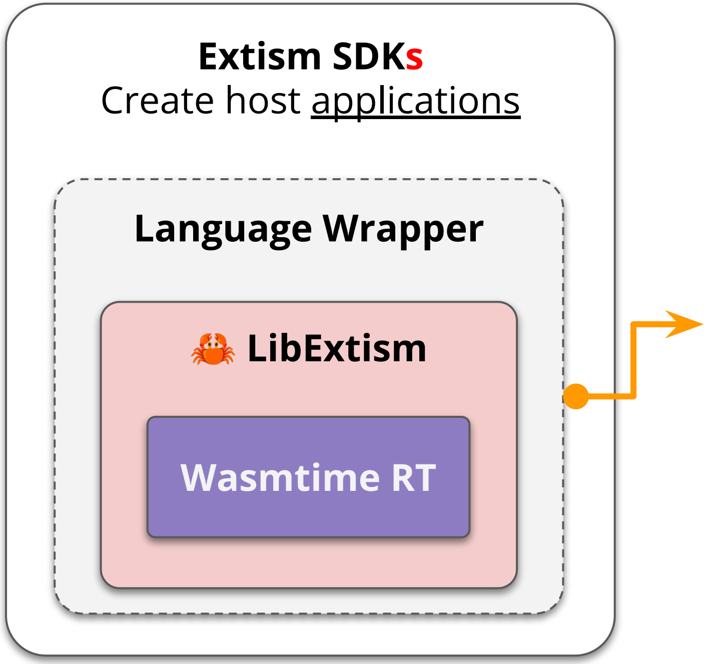
Extism SDKs



Extism SDKs

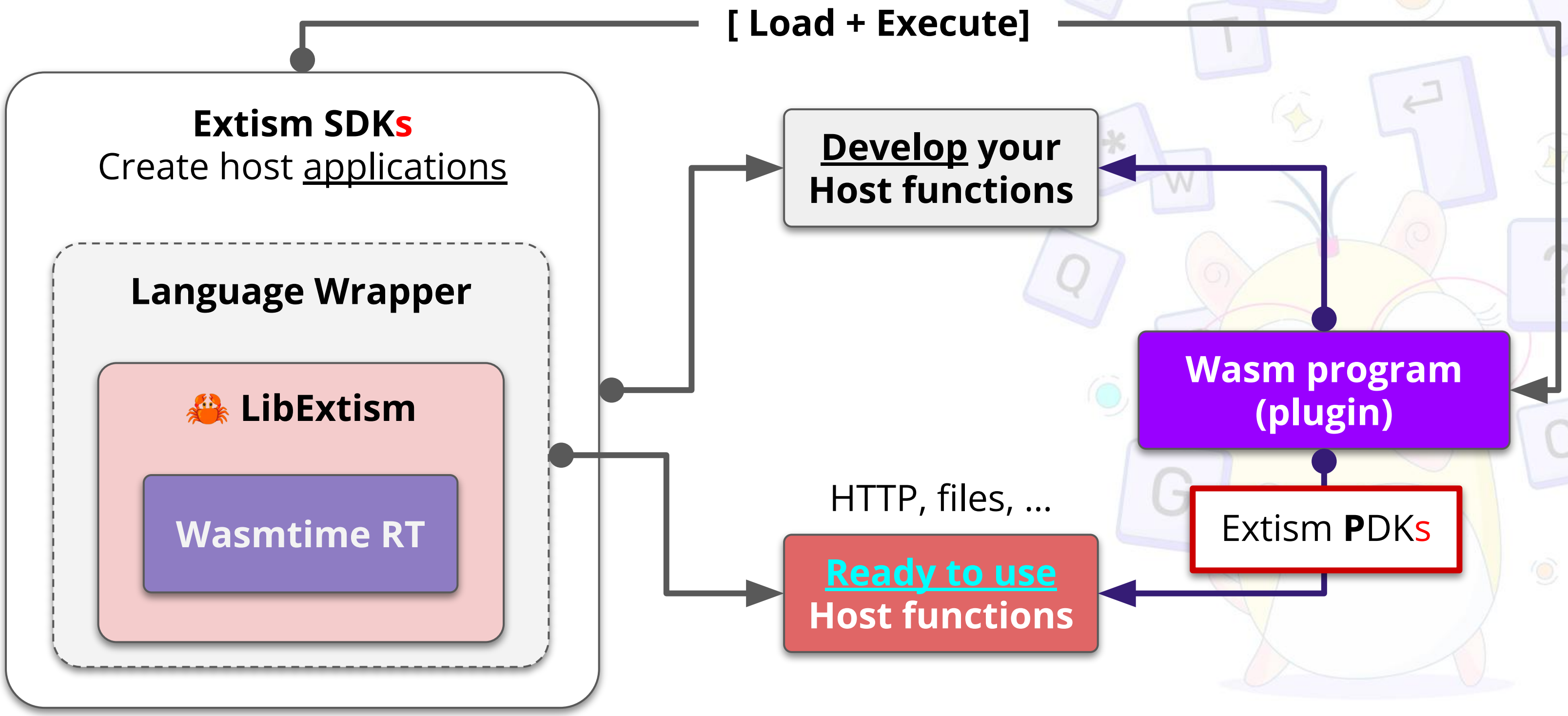

Extism SDKs
Create host applications
🦀 LibExtism
Wasmtime RT
https://github.com/extism/extism/blob/main/runtime/extism.h

Extism SDKs

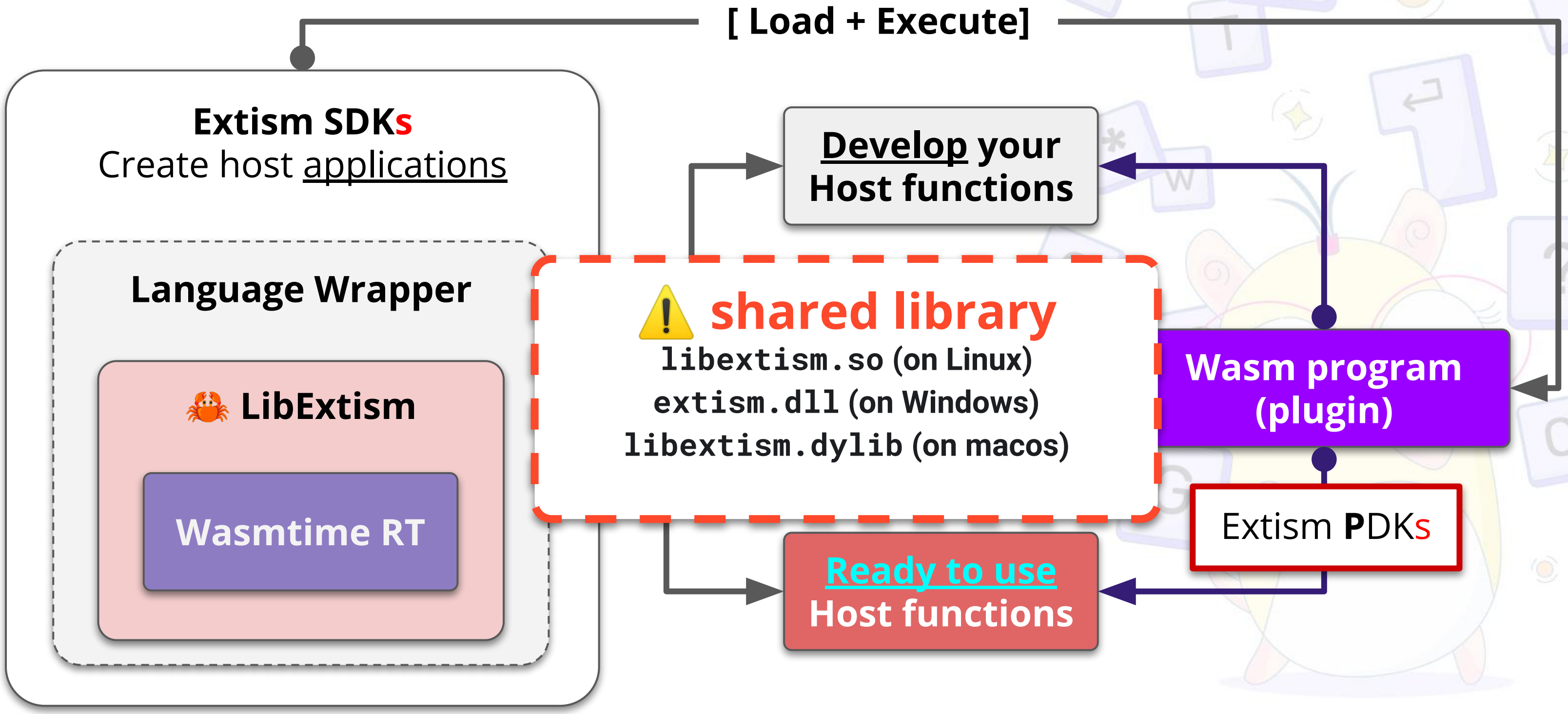


Browser / JS
C
C++
.NET
Elixir / Erlang
Go
Haskell
Java
Node
OCaml
PHP
Python
Ruby
Rust
Zig

Extism SDKs + Ready to use Host Function



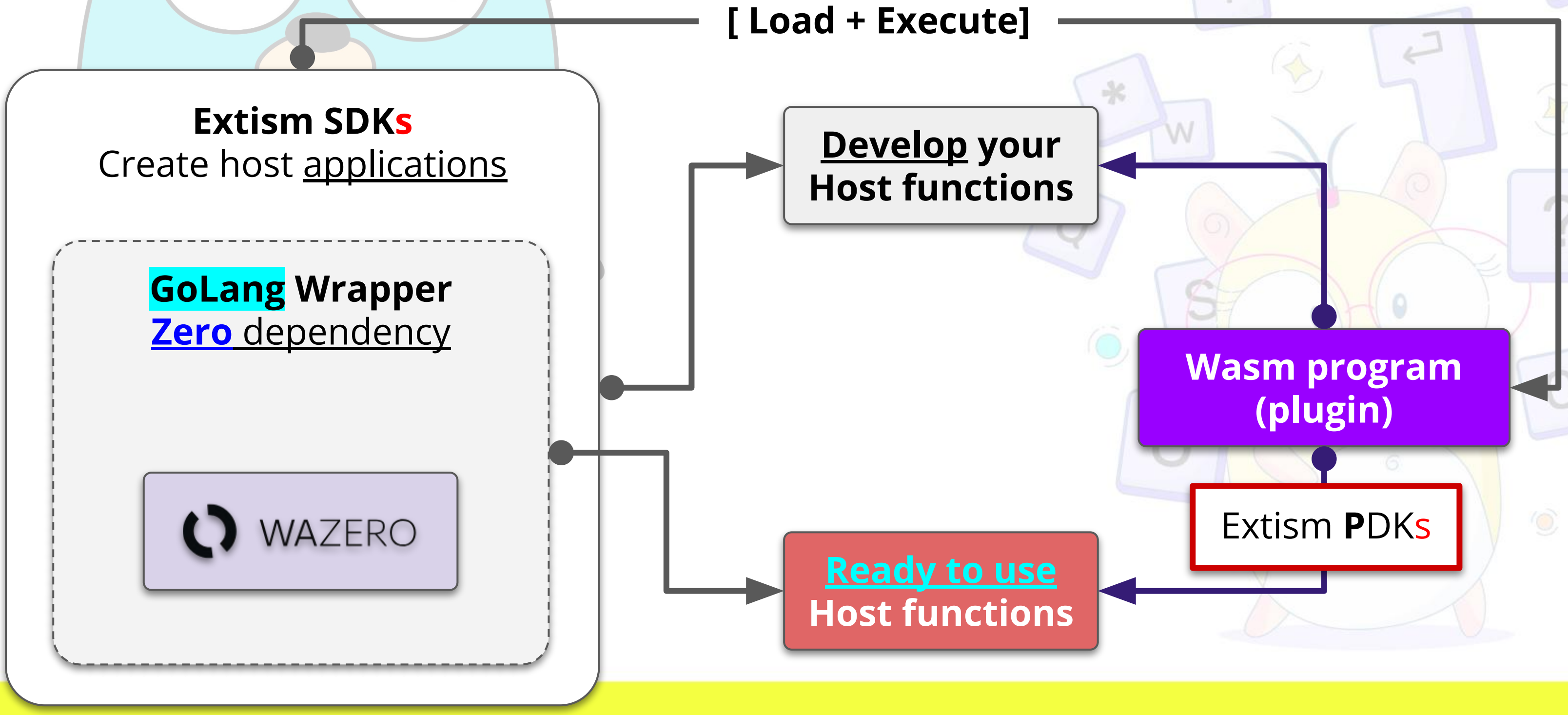
Extism SDKs + Ready to use Host Function

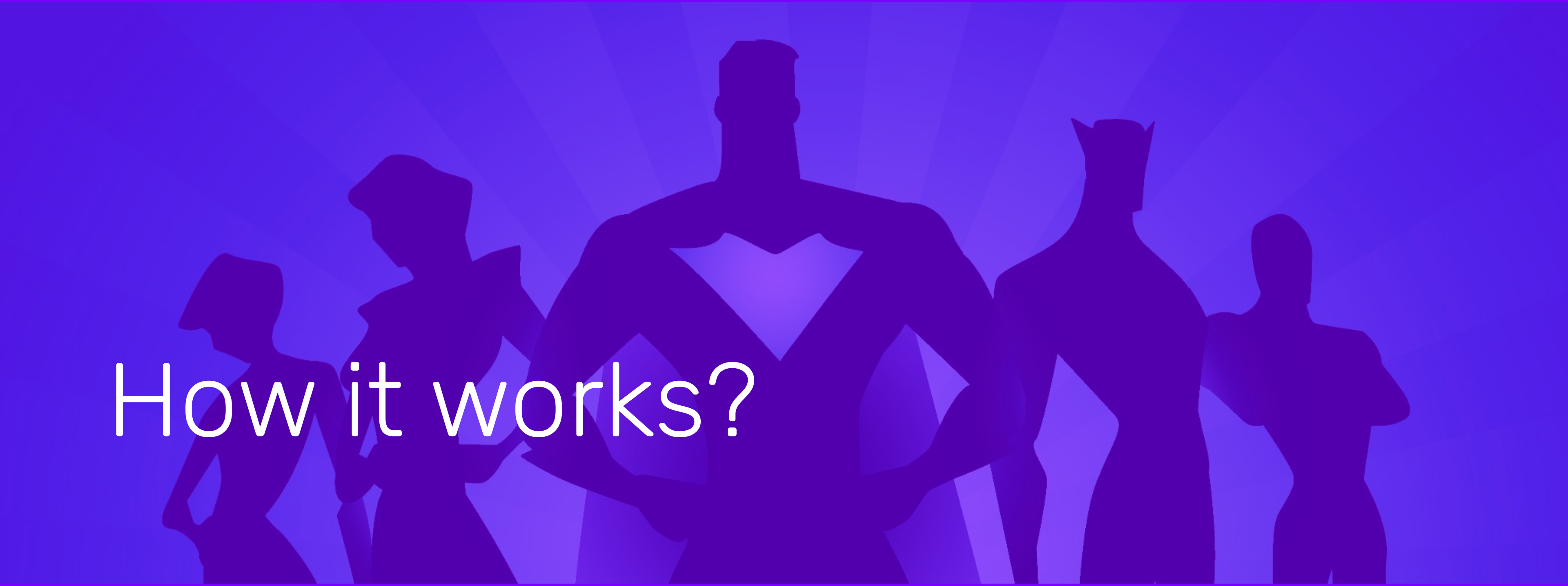


Go-SDK:
Extism 💖 Wazero

https://github.com/extism/go-sdk

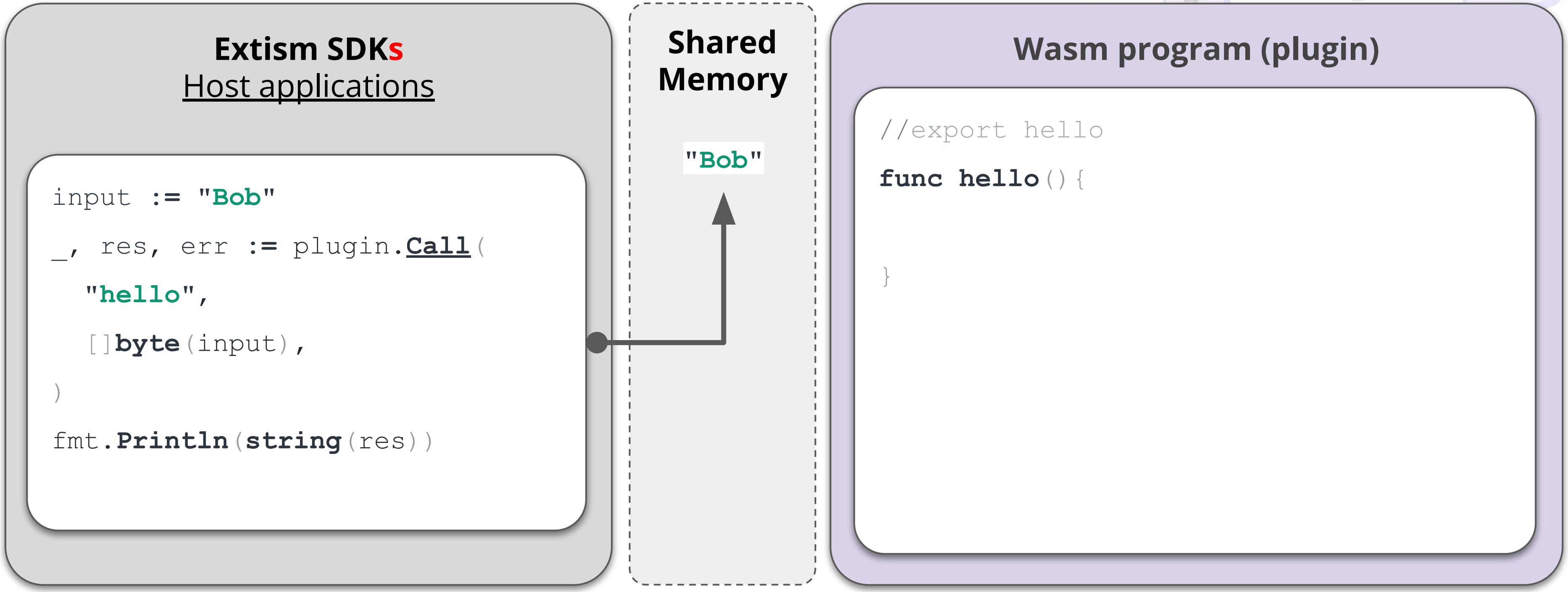
Extism Go-SDK





How it works?

Extism *-SDK + Extism *-PDK

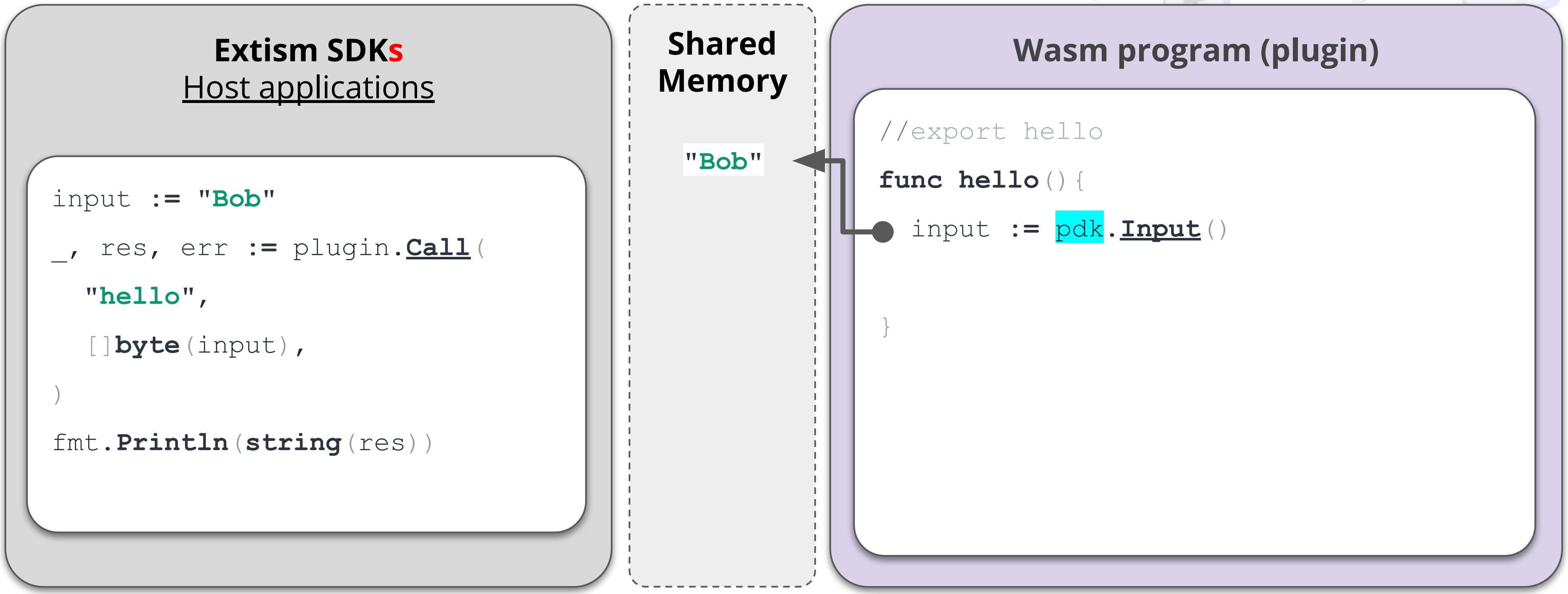


```
input := "Bob"
_, res, err := plugin.Call(
   "hello",
   []byte(input),
)
fmt.Println(string(res))
```

```
//export hello
func hello(){


}
```

Extism *-SDK + Extism *-PDK

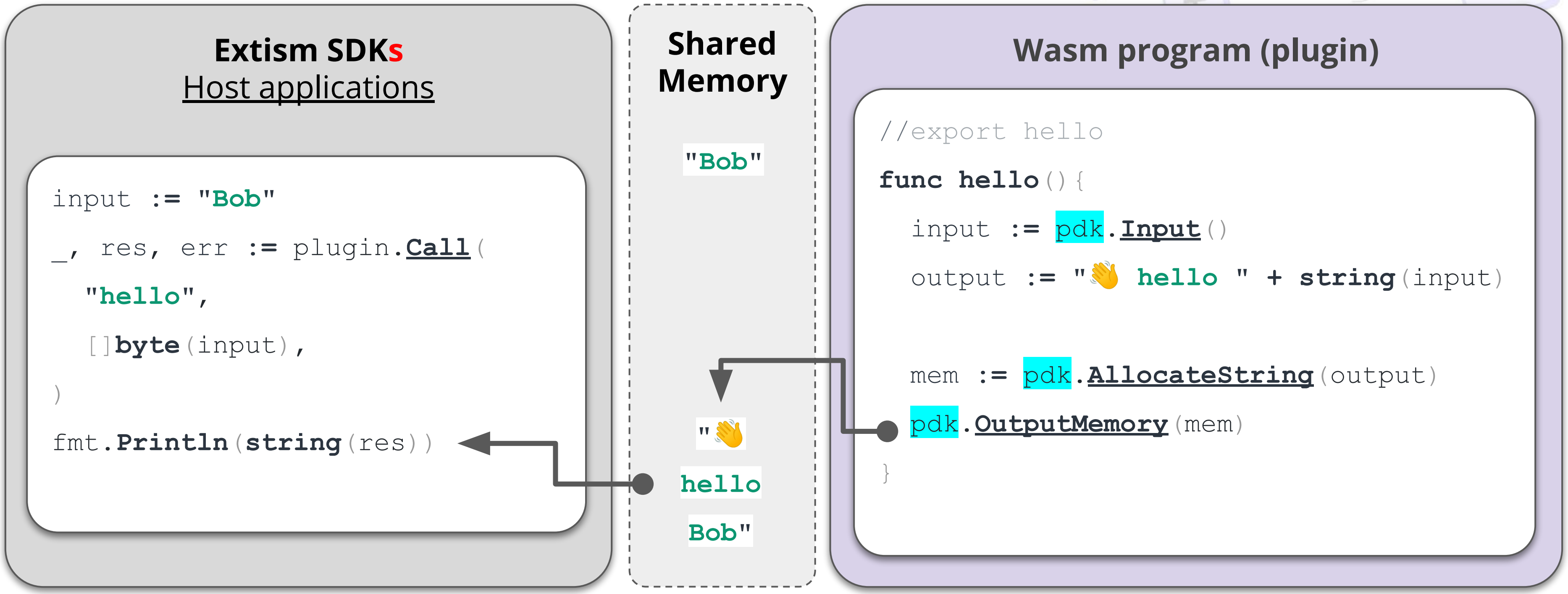


```go
input := "Bob"
_, res, err := plugin.Call(
  "hello",
  []byte(input),
)
fmt.Println(string(res))
```

```go
//export hello
func hello(){
  input := pdk.Input()


}
```

Extism *-SDK + Extism *-PDK



```go
input := "Bob"
_, res, err := plugin.Call(
  "hello",
  []byte(input),
)
fmt.Println(string(res))
```

```go
//export hello
func hello(){
  input := pdk.Input()
  output := "👋 hello " + string(input)

  mem := pdk.AllocateString(output)
  pdk.OutputMemory(mem)
}
```

Demo time!

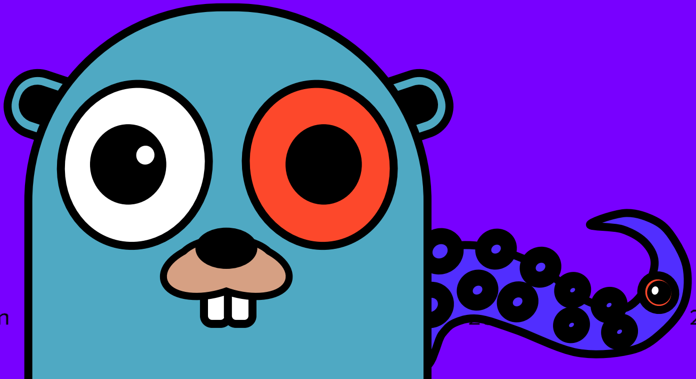
Let’s write some Extism Wasm plugins (with the PDKs)

03-go-plugin + Extism **CLI**

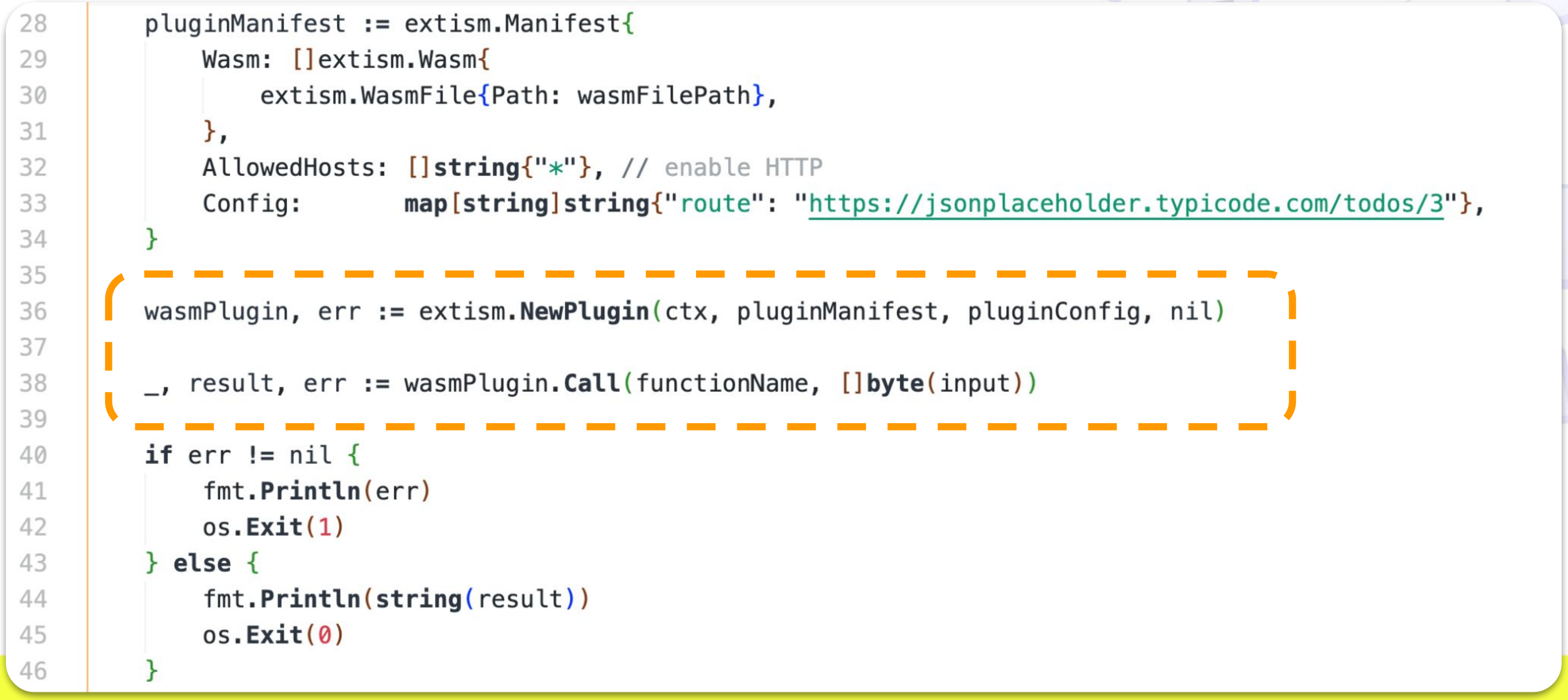
Examples:
04-rust-plugin
05-js-plugin

Create a Host Application

Write a CLI with the Extism Go-SDK



With Extism **Golang-S**DK 🥰



```
    pluginManifest := extism.Manifest{
        Wasm: []extism.Wasm{
            extism.WasmFile{Path: wasmFilePath},
        },
        AllowedHosts: []string{"*"}, // enable HTTP
        Config:       map[string]string{"route": "https://jsonplaceholder.typicode.com/todos/3"},
    }

    wasmPlugin, err := extism.NewPlugin(ctx, pluginManifest, pluginConfig, nil)

    _, result, err := wasmPlugin.Call(functionName, []byte(input))

    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    } else {
        fmt.Println(string(result))
        os.Exit(0)
    }
```

Demo time!

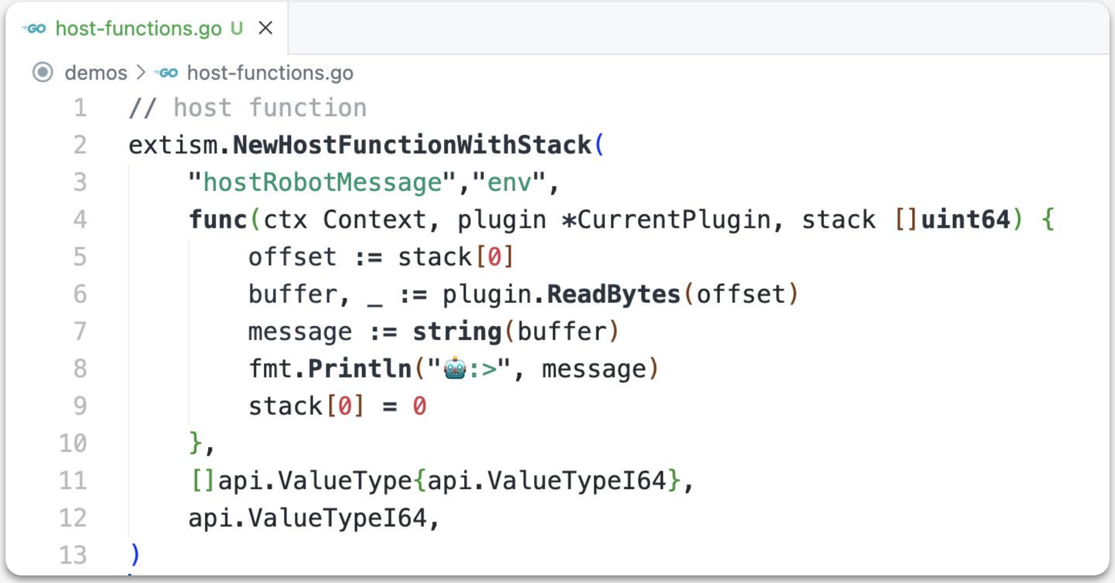
Let’s write a Host Application (with the Go SDK)

06-go-host-application

and a last example: 07-http-server

Give super powers to your Golang applications

Extism & Host Functions


host-functions.go U ×

demos > host-functions.go

```go
// host function
extism.NewHostFunctionWithStack(
    "hostRobotMessage","env",
    func(ctx Context, plugin *CurrentPlugin, stack []uint64) {
        offset := stack[0]
        buffer, _ := plugin.ReadBytes(offset)
        message := string(buffer)
        fmt.Println("🤖:>", message)
        stack[0] = 0
    },
    []api.ValueType{api.ValueTypeI64},
    api.ValueTypeI64,
)
```

https://extism.org/docs/integrate-into-your-codebase/go-host-sdk#host-functions

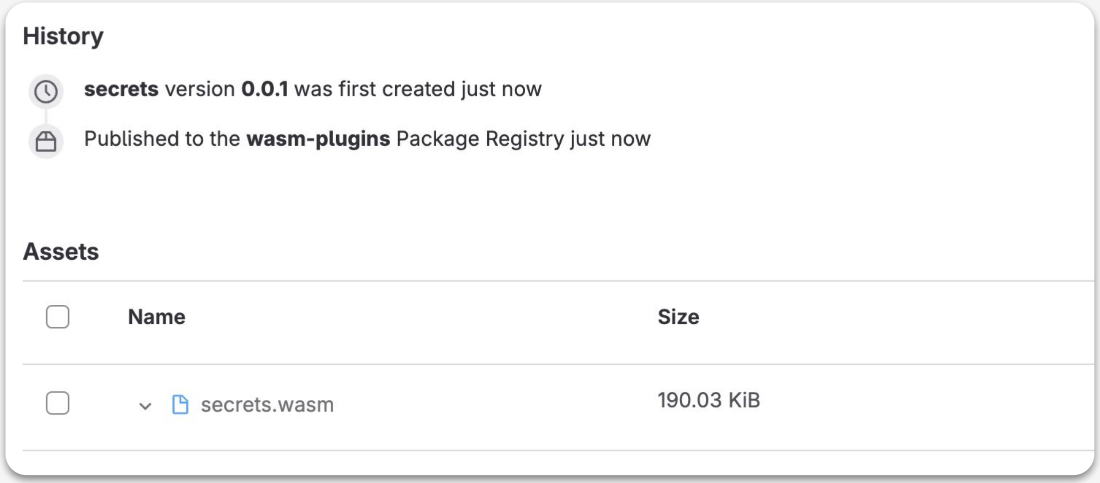
SDKs & PDKs are evolving

(and probably new ones to come)

With Go & Extism + Wazero the possibilities are numerous https://github.com/bots-garden/minism

Accelerate your CI with WASM plugins

Minism CLI **6.5MB** (zero dependency)
🐳 image: botsgarden/minism **7.03MB**
Wasm plugin few KB



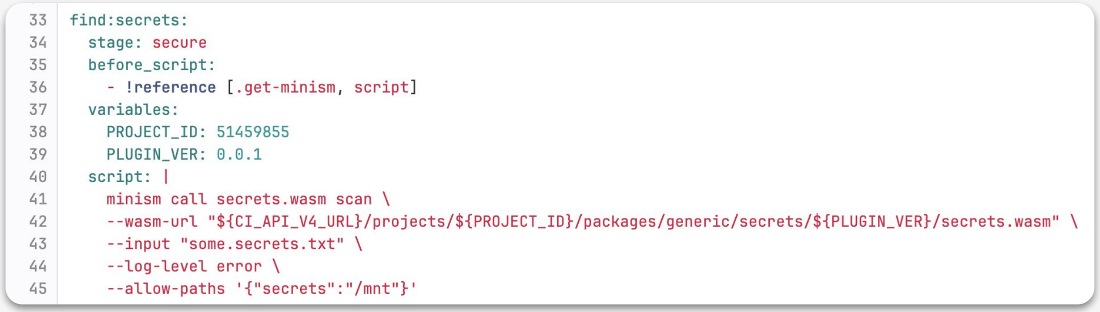
History

secrets version **0.0.1** was first created just now

Published to the **wasm-plugins** Package Registry just now

Assets

Name	Size
secrets.wasm	190.03 KiB



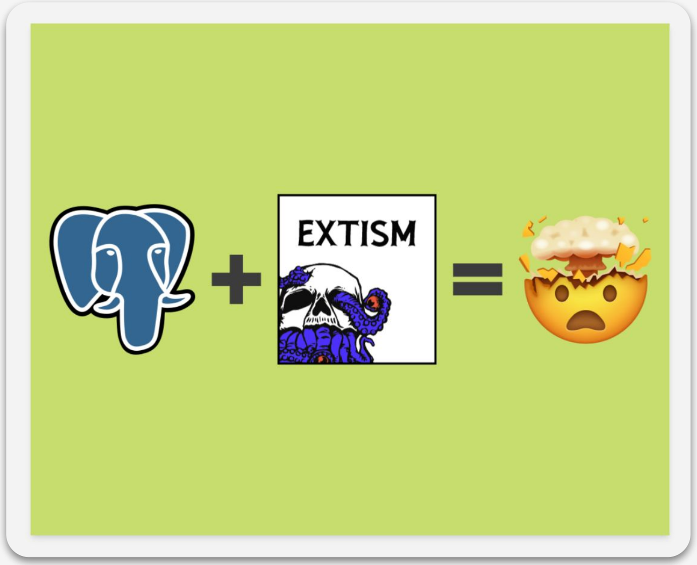
```yaml
find:secrets:
  stage: secure
  before_script:
    - !reference [.get-minism, script]
  variables:
    PROJECT_ID: 51459855
    PLUGIN_VER: 0.0.1
  script: |
    minism call secrets.wasm scan \
    --wasm-url "${CI_API_V4_URL}/projects/${PROJECT_ID}/packages/generic/secrets/${PLUGIN_VER}/secrets.wasm" \
    --input "some.secrets.txt" \
    --log-level error \
    --allow-paths '{"secrets":"/mnt"}'
```

Running Extism Plugins in PostgreSQL

by: **Muhammad Azeez**

Bringing WebAssembly to PostgreSQL using Extism

https://dylibso.com/blog/pg-extism/



Philippe Charrière

✉️ ph.charriere@gmail.com
❎ @k33g_org
□ @k33gorg.bsky.social
📝 https://k33g.hashnode.dev

Source code

https://github.com/bots-garden/golab-2023

🍊
https://gitpod.io/#https://github.com/bots-garden/golab-2023

🐳
https://open.docker.com/dashboard/dev-envs?url=https://github.com/bots-garden/golab-2023/tree/main

Some blog posts to help

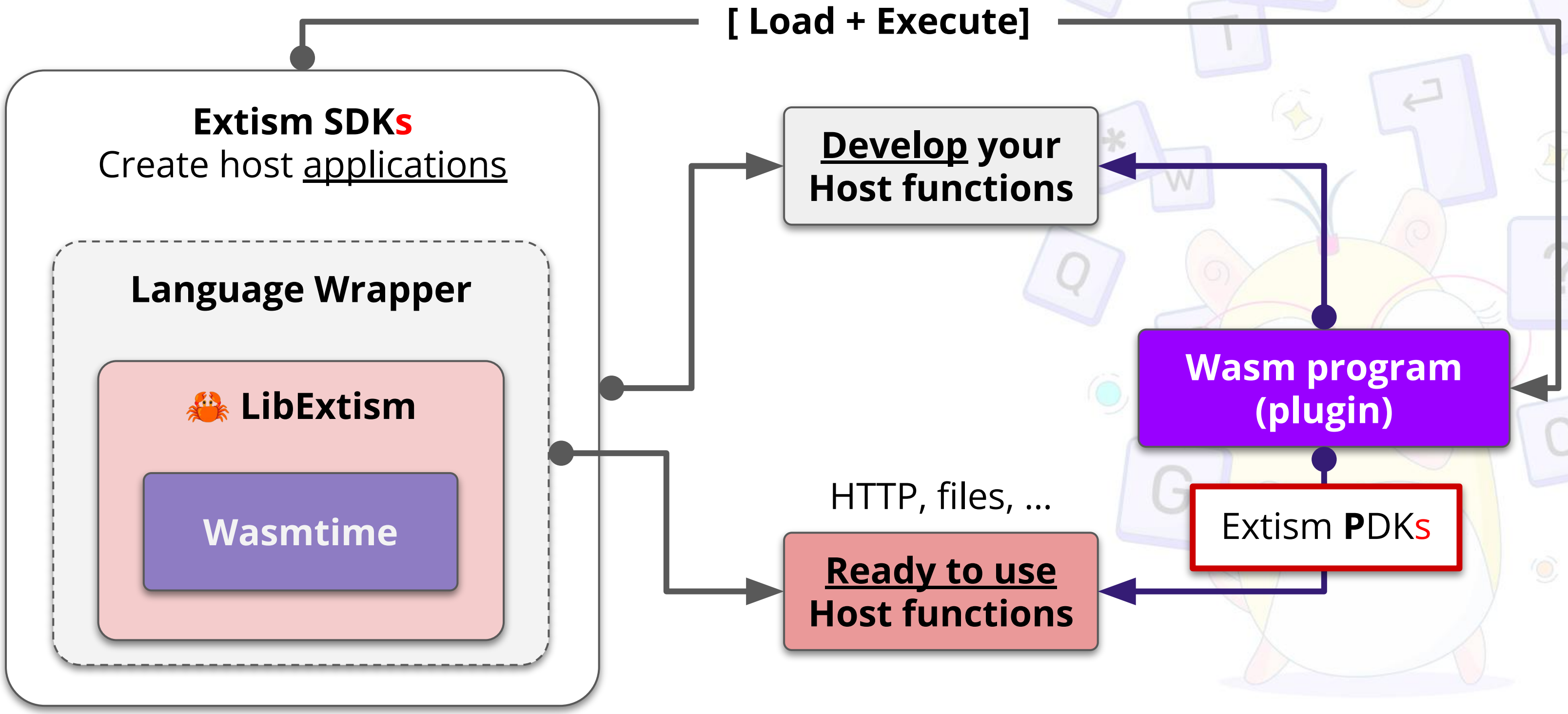
📝 WASI and Node.js:
https://k33g.hashnode.dev/series/wasi-nodejs
📝 Wazero, first steps
https://k33g.hashnode.dev/series/wazero-first-steps
📝 Discovery of Extism
https://k33g.hashnode.dev/series/extism-discovery

Thank you for your attention

Q&A

Use Extism & Wazero, this is the way

Extism SDKs


[Load + Execute]
Extism SDKs
Create host applications
Language Wrapper
🦀 LibExtism
Wasmtime
Develop your Host functions
HTTP, files, ...
Ready to use Host functions
Wasm program (plugin)
Extism PDKs

Extism & Host Functions

https://extism.org/docs/integrate-into-your-codebase/go-host-sdk#host-functions

Extism *-SDK: create a host function

Extism SDKs Host applications

```
robotMessage := func(ctx Context, plugin *CurrentPlugin, stack []uint64)
{
  offset := stack[0]
  buffer, _ := plugin.ReadBytes(offset)
  message := string(buffer)
  fmt.Println("🤖:>", message)
  stack[0] = 0
}
extism.NewHostFunctionWithStack("hostRobotMessage","env", robotMessage,
[]api.ValueType{api.ValueTypeI64},api.ValueTypeI64)
```

Extism *-SDK: create a host function

Extism SDKs Host applications

```go
robotMessage := func(...) {
  offset := stack[0]
  buffer, _ := plugin.ReadBytes(offset)
  message := string(buffer)
  fmt.Println("🤖:>", message)
  stack[0] = 0
}
```

[trigger]

Wasm Plugin

```go
//export hostRobotMessage
func hostRobotMessage(offset uint64) uint64

func RobotMessage(message string) {
  mem := pdk.AllocateString(message)
  hostRobotMessage(mem.Offset())
}

func say_hello() {
  RobotMessage("hello " + string(input))
}
```

Extism *-SDK + Extism *-PDK

Extism SDKs
Host applications

```
input := "Bob"
_, res, err := plugin.Call(
  "hello",
  []byte(input),
)
fmt.Println(string(res))
```

Wasm program (plugin)

```
//export hello
func hello(name string) string {
  return "👋 hello " + name
}
```

Extism *-SDK + Extism *-PDK

Extism SDKs
Host applications

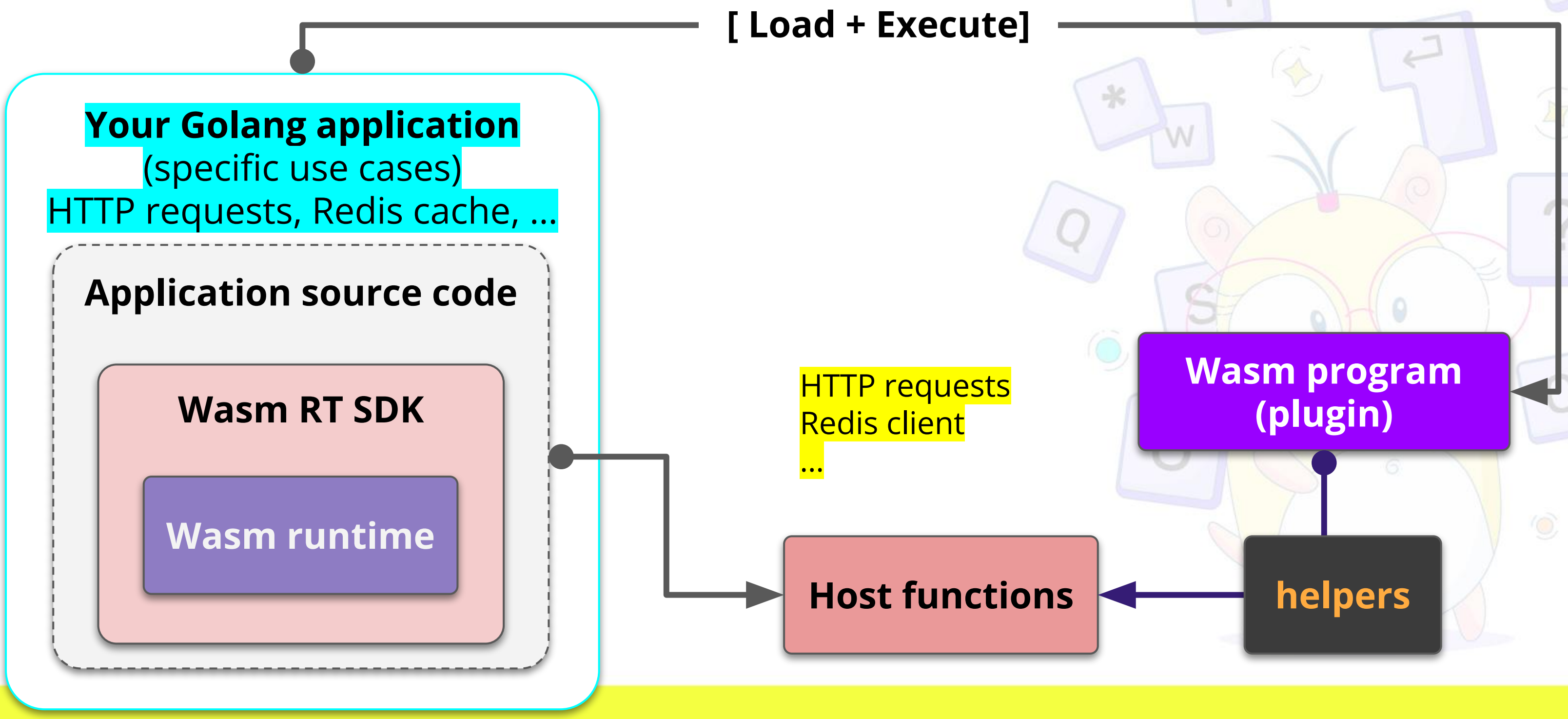
```
input := "Bob"
_, res, err := plugin.Call(
  "hello",
  []byte(input),
)
fmt.Println(string(res))
```

Wasm program (plugin)

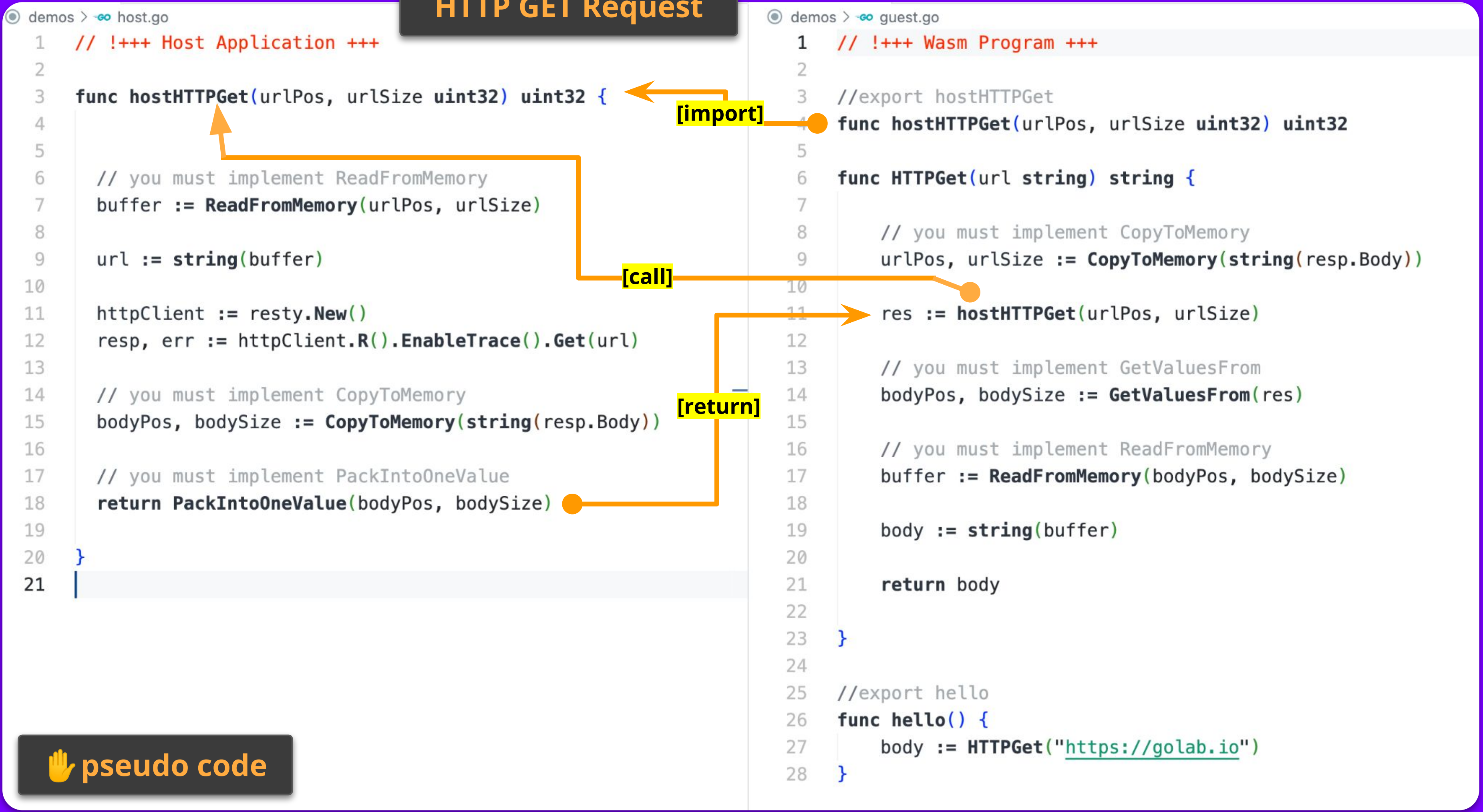
```
//export hello
func hello(~~name string~~) ~~string~~ {

}
```

Your Golang application



HTTP GET Request



```go
// !+++ Host Application +++

func hostHTTPGet(urlPos, urlSize uint32) uint32 {


  // you must implement ReadFromMemory
  buffer := ReadFromMemory(urlPos, urlSize)

  url := string(buffer)

  httpClient := resty.New()
  resp, err := httpClient.R().EnableTrace().Get(url)

  // you must implement CopyToMemory
  bodyPos, bodySize := CopyToMemory(string(resp.Body))

  // you must implement PackIntoOneValue
  return PackIntoOneValue(bodyPos, bodySize)

}
```

```go
// !+++ Wasm Program +++

//export hostHTTPGet
func hostHTTPGet(urlPos, urlSize uint32) uint32

func HTTPGet(url string) string {

    // you must implement CopyToMemory
    urlPos, urlSize := CopyToMemory(string(resp.Body))

    res := hostHTTPGet(urlPos, urlSize)

    // you must implement GetValuesFrom
    bodyPos, bodySize := GetValuesFrom(res)

    // you must implement ReadFromMemory
    buffer := ReadFromMemory(bodyPos, bodySize)

    body := string(buffer)

    return body

}

//export hello
func hello() {
    body := HTTPGet("https://golab.io")
}
```