Marine: dynamic Wasm modules linking

Mike Voronov
twitter.com/@vms11

IPFS þing, July 15, 2022

fluence

Agenda

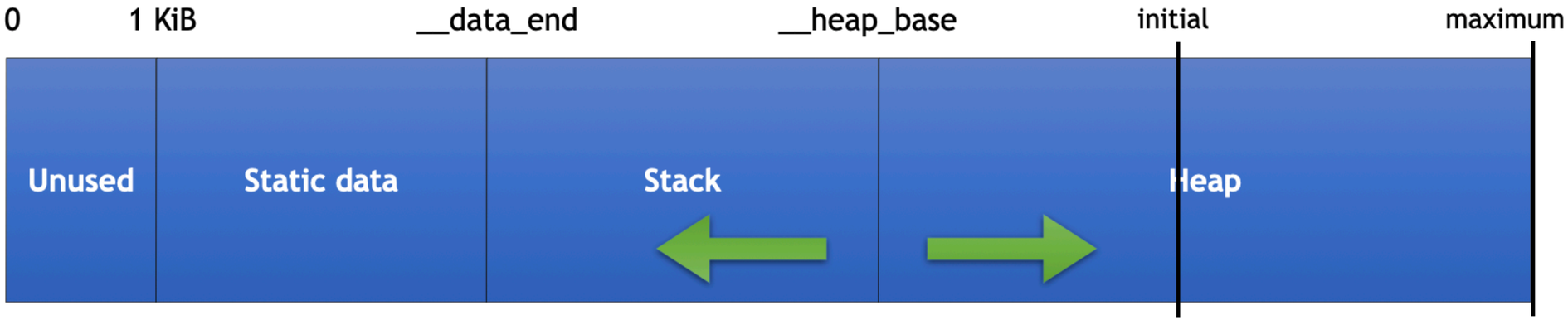
- Wasm component model
- Fluence Wasm use-case
- Marine runtime
- Marine SDK & tools

Wasm component model (basics)

Wasm prerequisites

Values WebAssembly provides only four basic *number types*. These are integers and IEEE 754-2019[6] numbers, each in 32 and 64 bit width. 32 bit integers also serve as Booleans and as memory addresses. The usual operations on these types are available, including the full matrix of conversions between them. There is no distinction between signed and unsigned integer types. Instead, integers are interpreted by respective operations as either unsigned or signed in two's complement representation.

In addition to these basic number types, there is a single 128 bit wide vector type representing different types of packed data. The supported representations are 4 32-bit, or 2 64-bit IEEE 754-2019[7] numbers, or different widths of packed integer values, specifically 2 64-bit integers, 4 32-bit integers, 8 16-bit integers, or 16 8-bit integers.



Component Model High-Level Goals

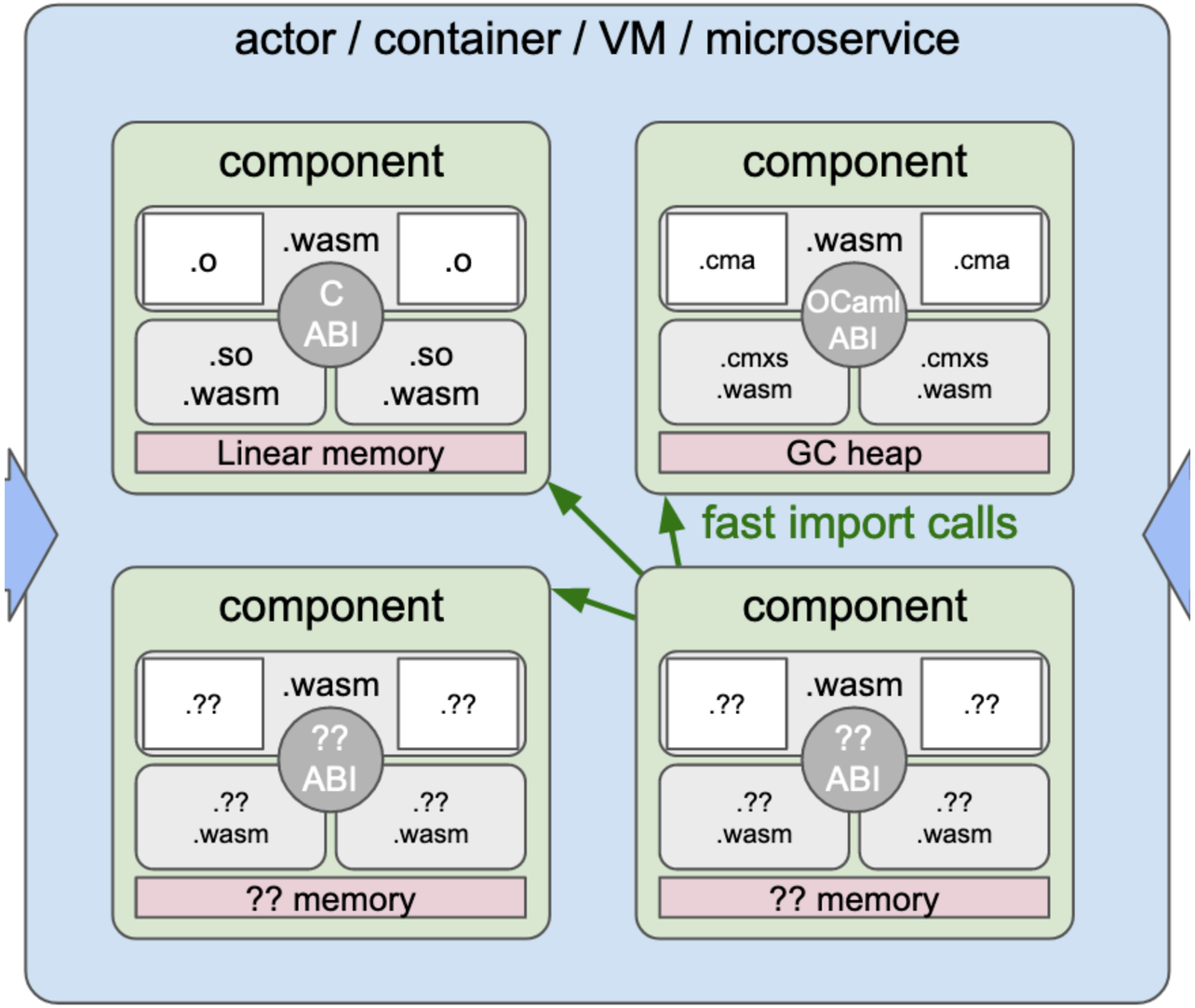
- Define a portable, load- and run-time-efficient **binary format** for separately-compiled components built from WebAssembly core modules that enable portable, cross-language composition.

- Support the definition of portable, virtualizable, statically-analyzable, capability-safe, language-agnostic **interfaces**, especially those being defined by WASI.

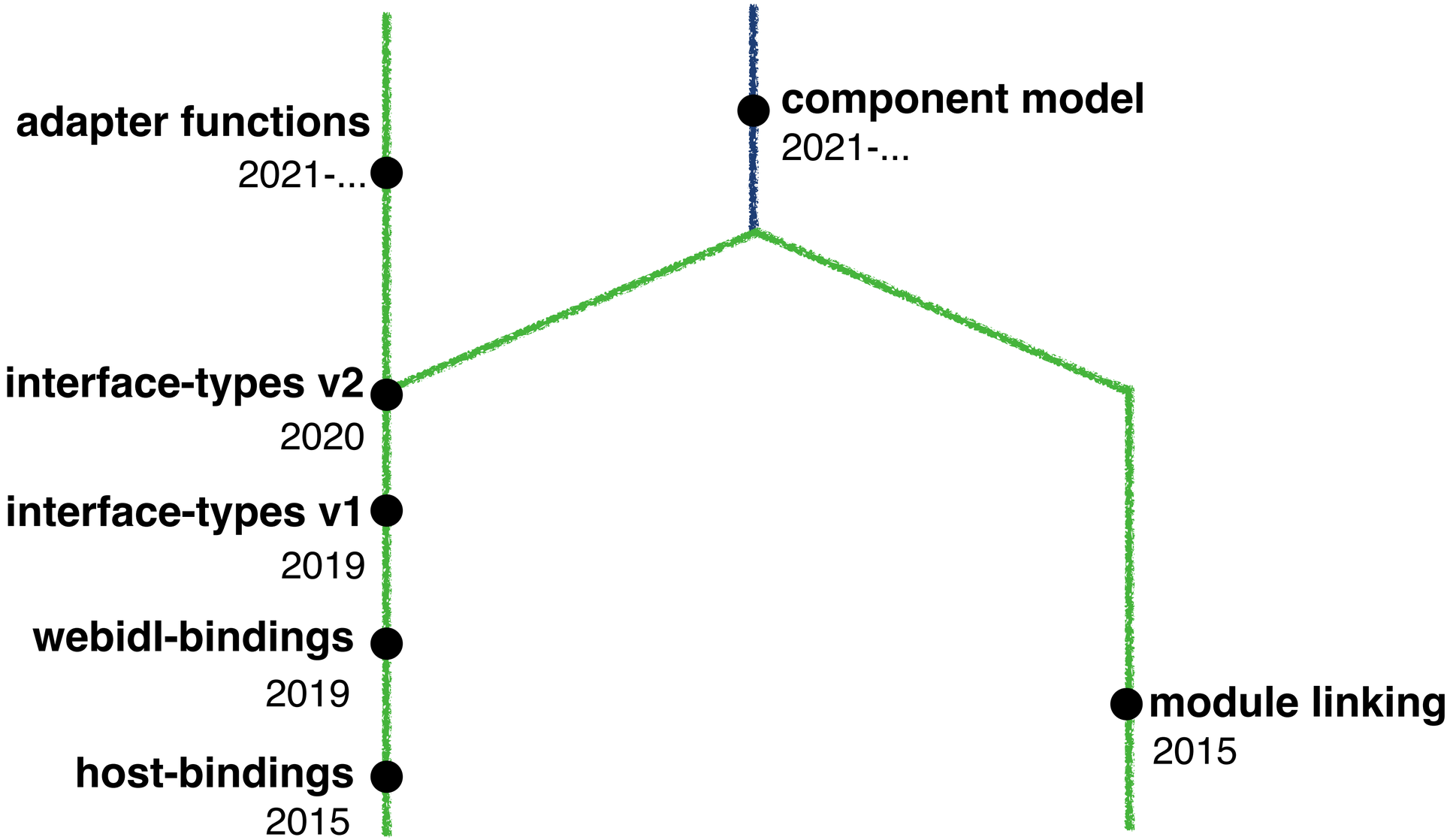
- Maintain and enhance WebAssembly's unique value proposition:
  - **Language neutrality**: avoid biasing the component model toward just one language or family of languages.
  - **Embeddability**: design components to be embedded in a diverse set of host execution environments, including browsers, servers, intermediaries, small devices and data-intensive systems.
  - **Optimizability**: maximize the static information available to Ahead-of-Time compilers to minimize the cost of instantiation and startup.

https://github.com/WebAssembly/component-model/blob/main/design/high-level/Goals.md

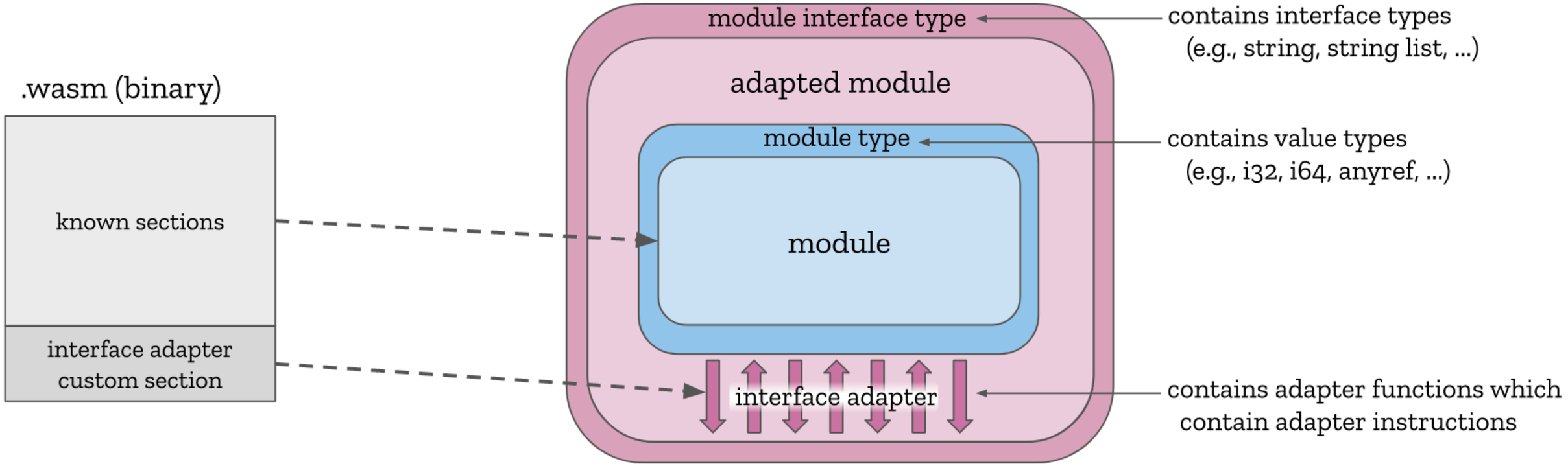
Lightweight Component Model overview



Component model proposal



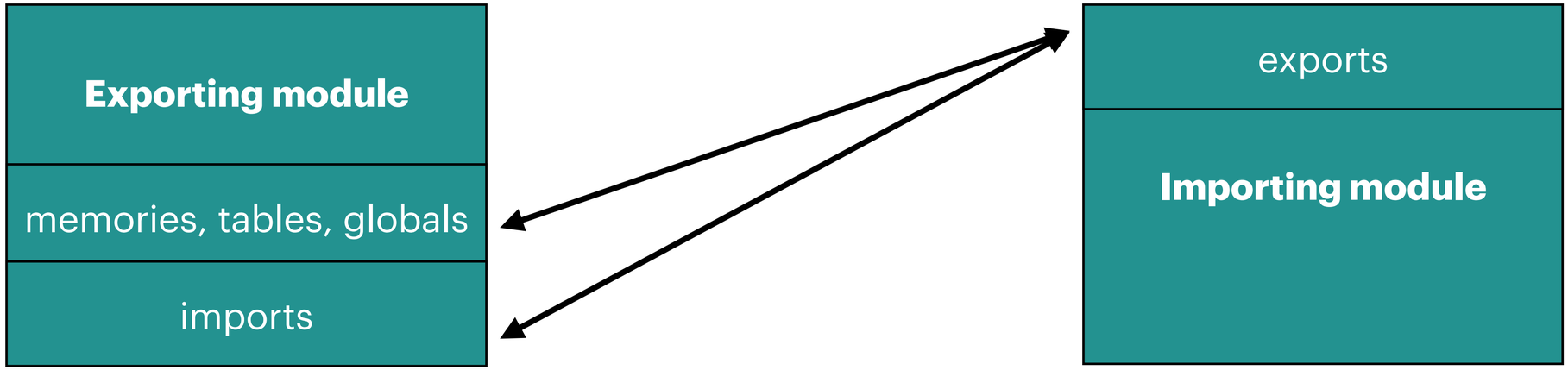
Interface types



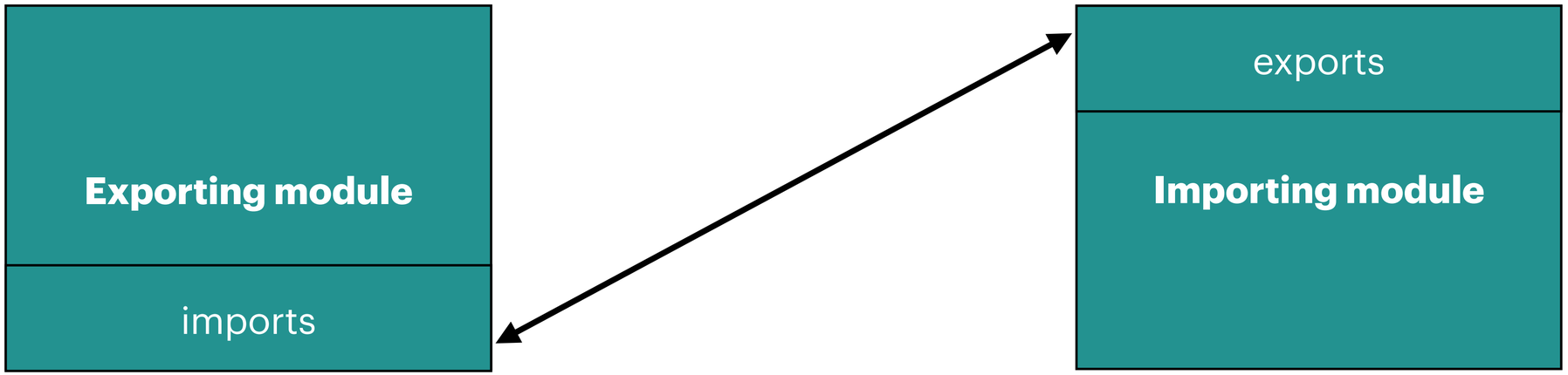
https://github.com/WebAssembly/interface-types

Module linking

shared-everything linking scheme



shared-nothing linking scheme



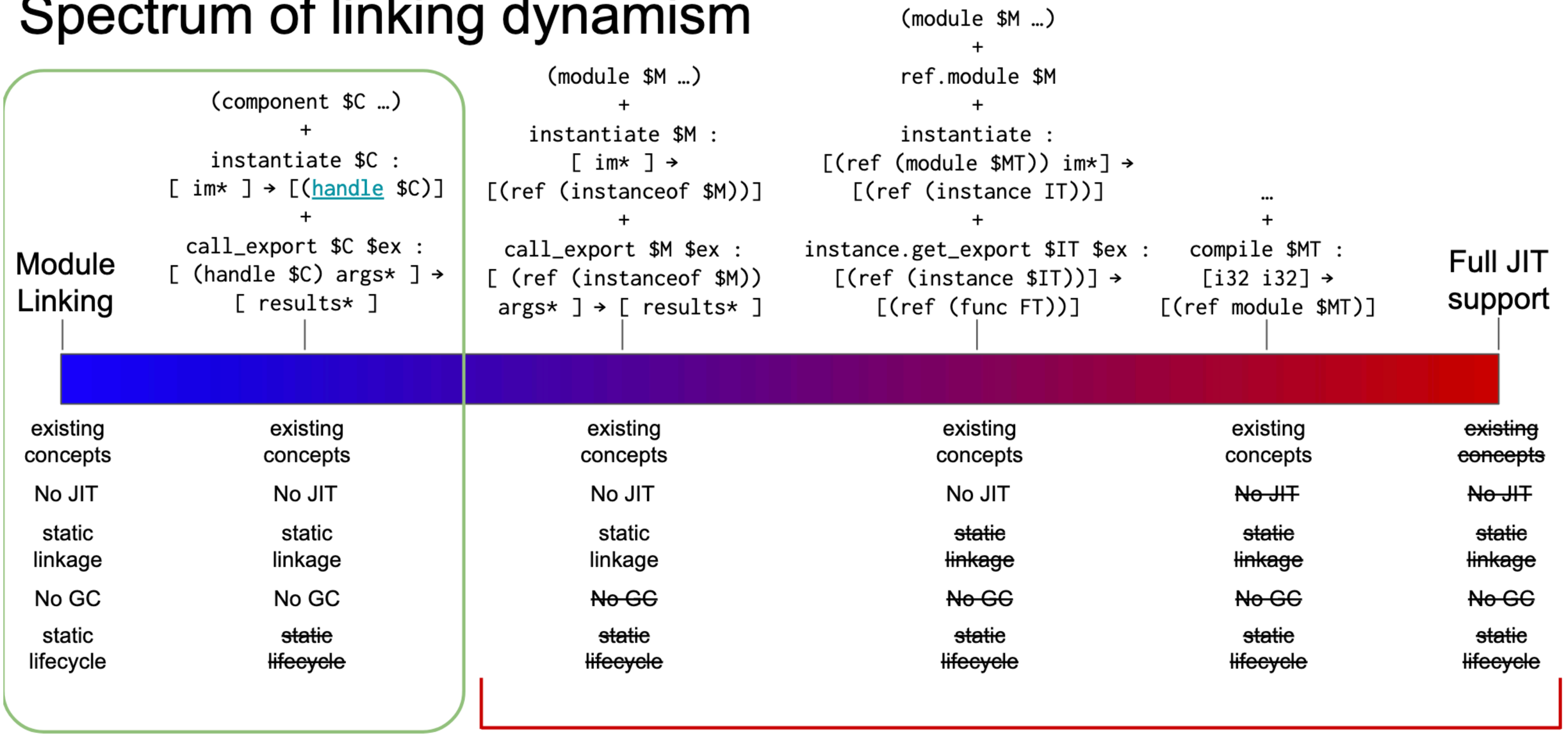
Component model example

Example of: Core + (Module Linking + Interface Types):

```
(adapter module
  (import “libc” (module $LIBC …))
  (instance $libc (instantiate $LIBC))
  (module
    (import “libc” “malloc” (func (param i32) (result i32)))
    …
    (func (export “run”) (param i32 i32) (result i32 i32) …)
  )
  (instance $core (instantiate $CORE (import “libc” “malloc” (func $libc “malloc"))))
  (adapter_func (export “run”) (param string) (result string)
    … lower param
    call (func $core “run”)
    … lift result
  )
)
```

https://docs.google.com/presentation/d/1PSC3Q5oFsJEaYyV5lNJvVgh-SNxhySWUqZ6puyojMi8/edit#slide=id.gced688a2b6_0_0

Spectrum of linking dynamism



In scope for component model

May be added (in some form) to core wasm in the future; may be used by individual component *bodies*; but not baked into the *component model*

WIT Interface

```
type           ::= (type <id>? <deftype>)
deftype        ::= <defvaltype>
                 | <functype>
                 | <componenttype>
                 | <instancetype>
defvaltype     ::= unit
                 | bool
                 | s8 | u8 | s16 | u16 | s32 | u32 | s64 | u64
                 | float32 | float64
                 | char | string
                 | (record (field <name> <valtype>)*)
                 | (variant (case <id>? <name> <valtype> (refines <id>)?)+)
                 | (list <valtype>)
                 | (tuple <valtype>*)
                 | (flags <name>*)
                 | (enum <name>+)
                 | (union <valtype>+)
                 | (option <valtype>)
                 | (expected <valtype> <valtype>)
```

Fluence Wasm use-case

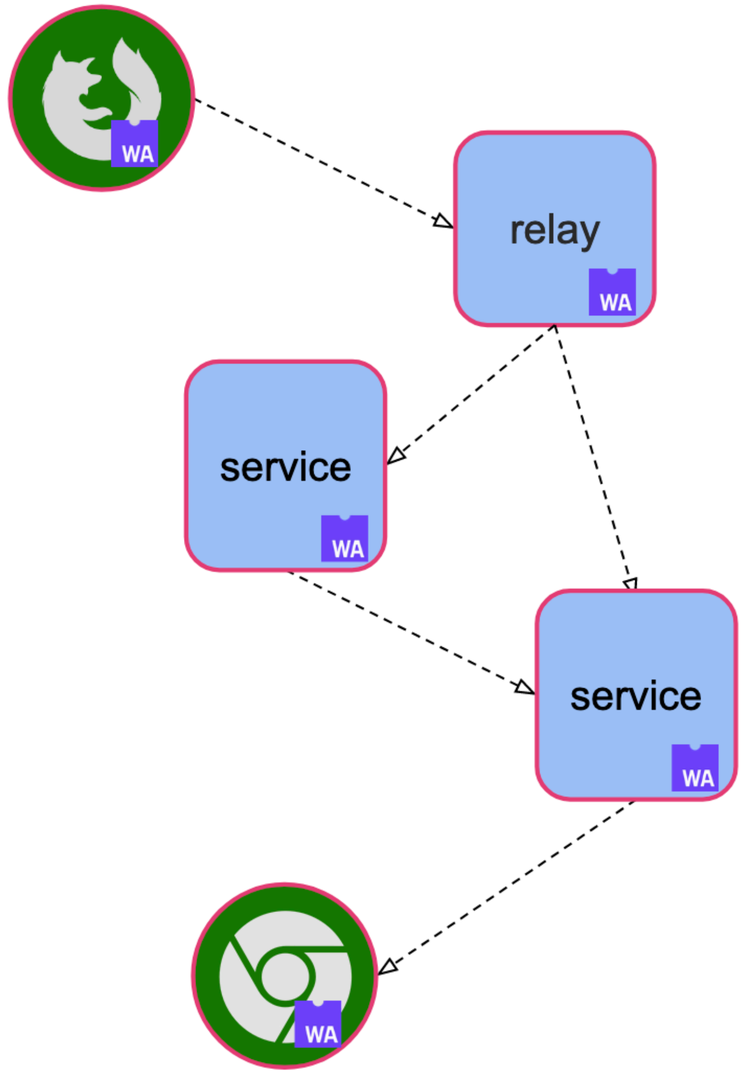
Fluence: composing peers

Aqua - programming language for **peer-to-peer**

- inspired by process calculus

- programmable network requests and algorithms

- coordination and orchestration with no central gateway



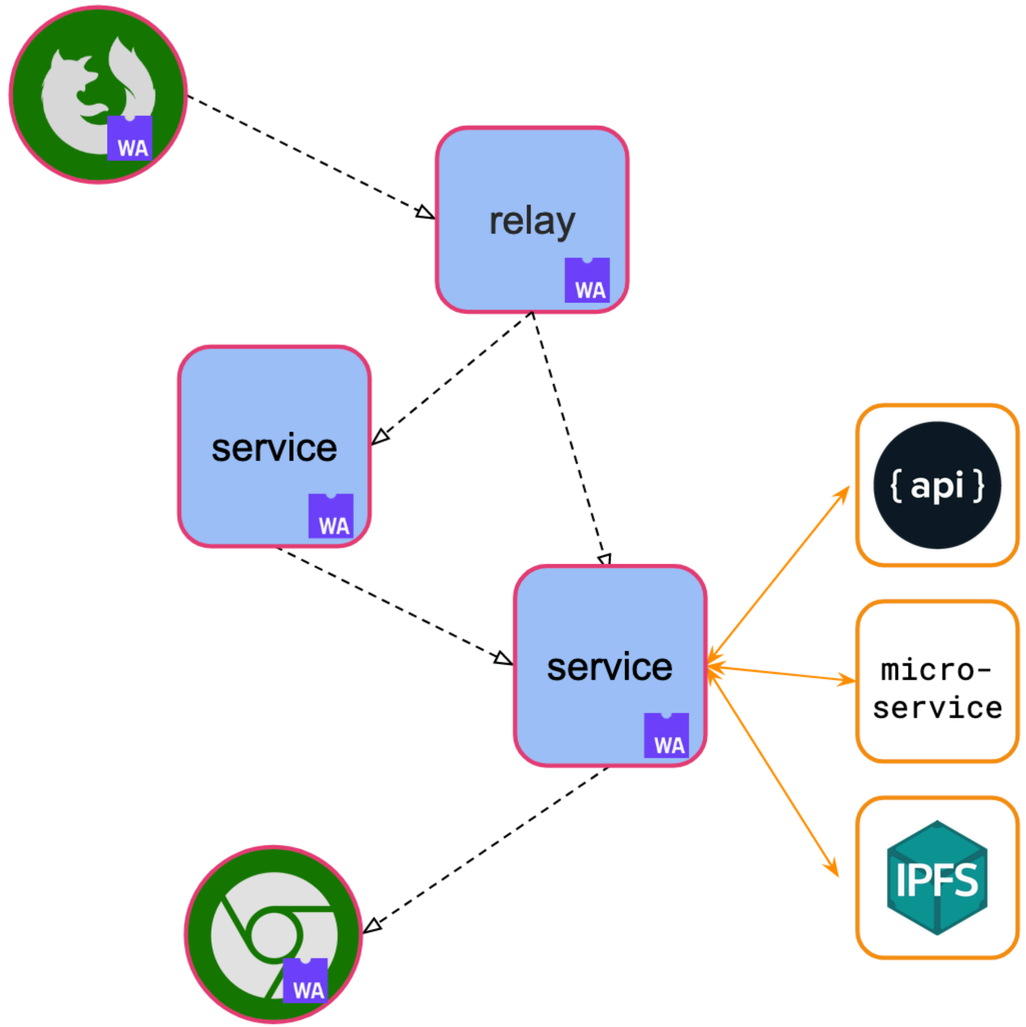
Fluence: ~~micro~~Services

- WebAssembly-based and sandboxed

- composable with Aqua and Wasm Component Model

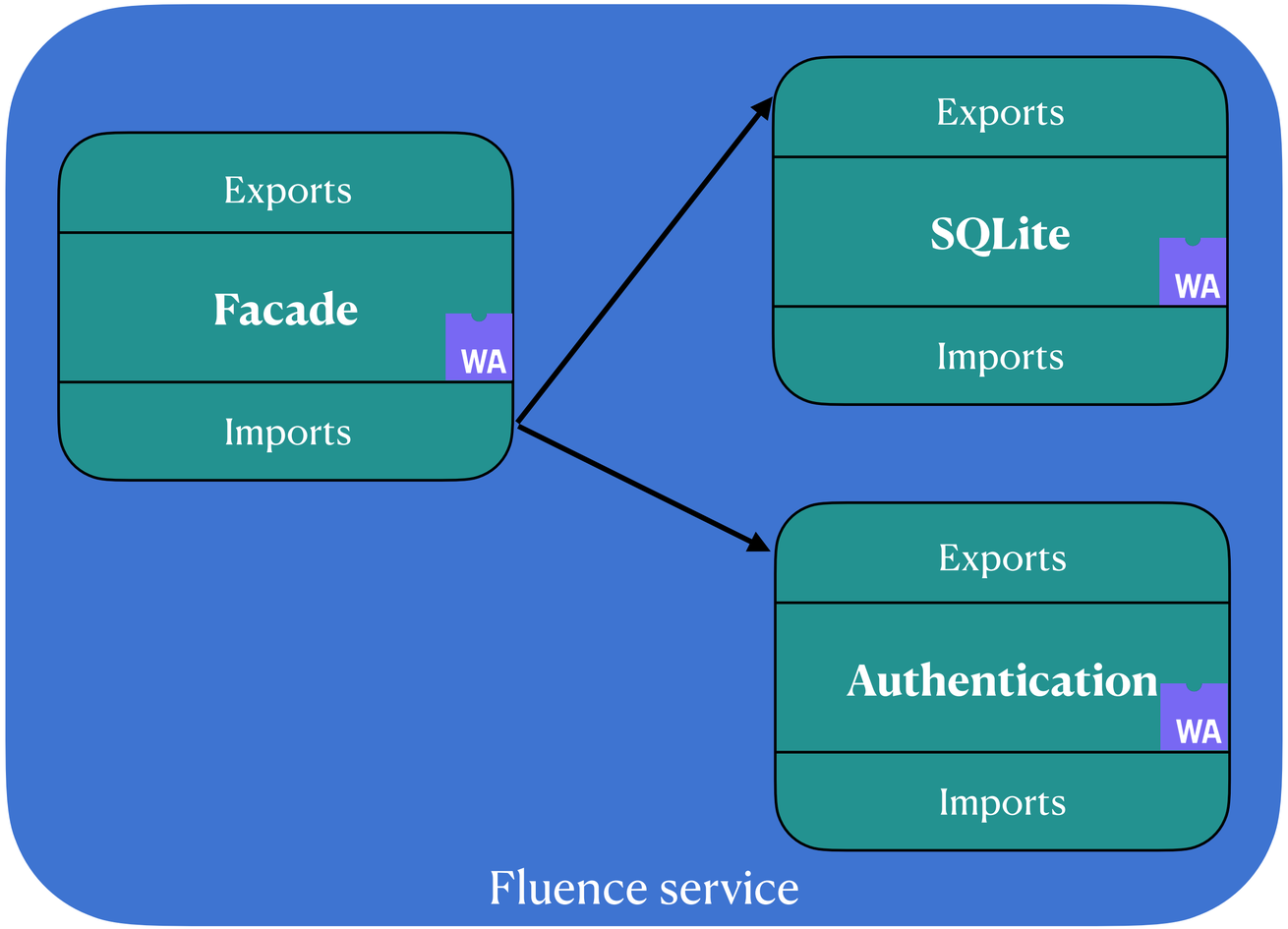
- stateful or stateless

- extendable via external executables, APIs, data sources



Fluence service

Fluence service is a group of modules linked together by shared-nothing linking scheme with help of IT

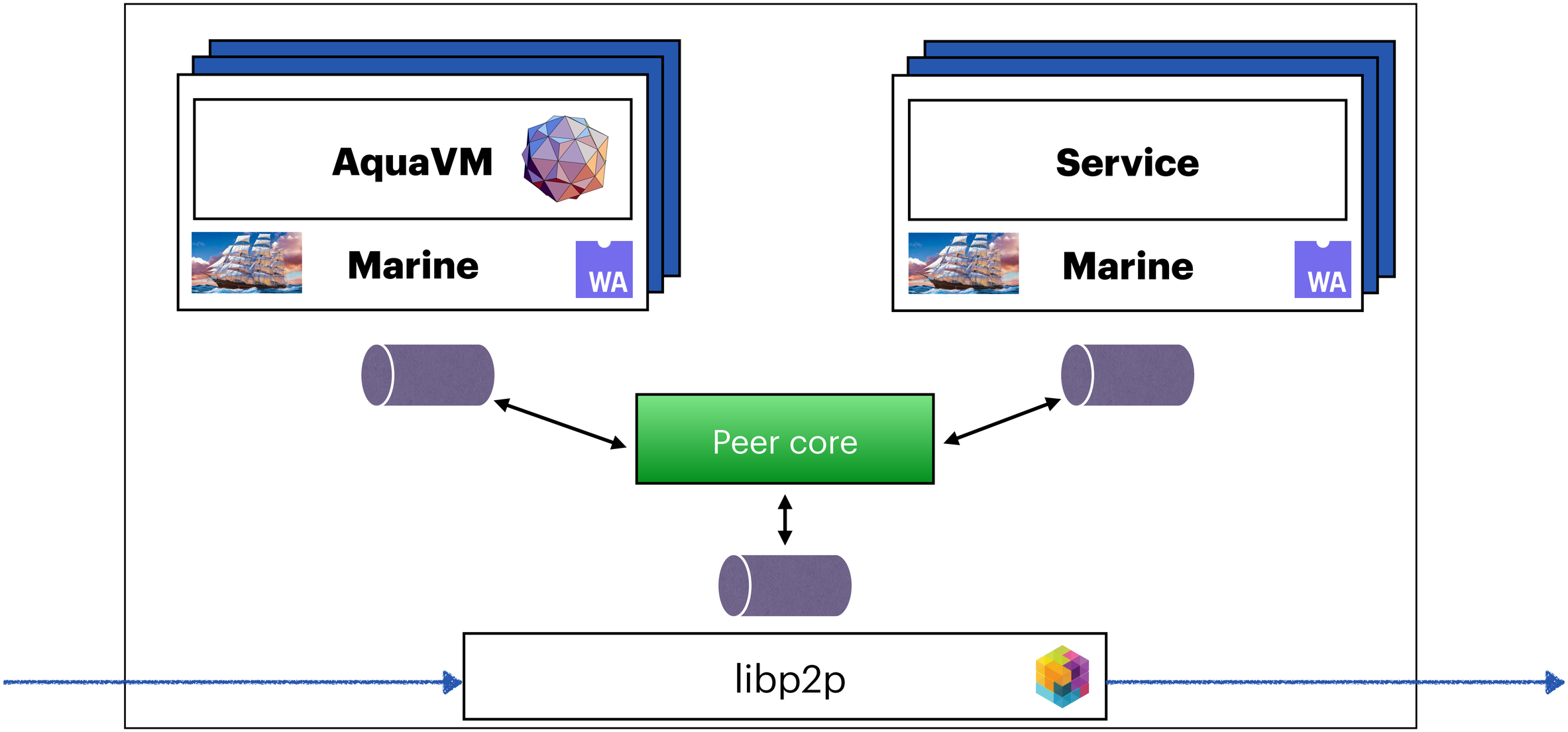


Fluence service

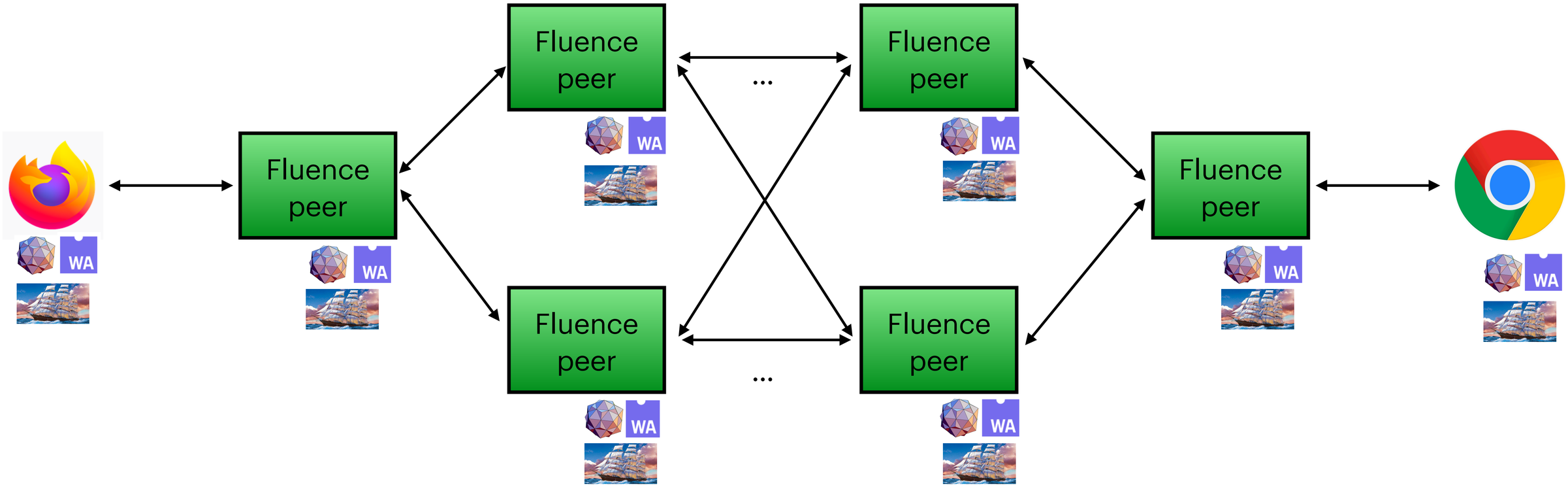
We've chosen Wasm in the Fluence Labs as the main building mechanism for services because it allows us to

- flexible resources control
- sandboxing modules by design
- use heterogeneous runtime with support of any language compiled to Wasm
- simple, but featurefull composition between modules
- make lifecycles of modules are independent on each other

Fluence Peer architecture



Fluence network



Marine runtime

Marine high-level goals

- **DX**

  best DX == as close to vanilla Rust as possible

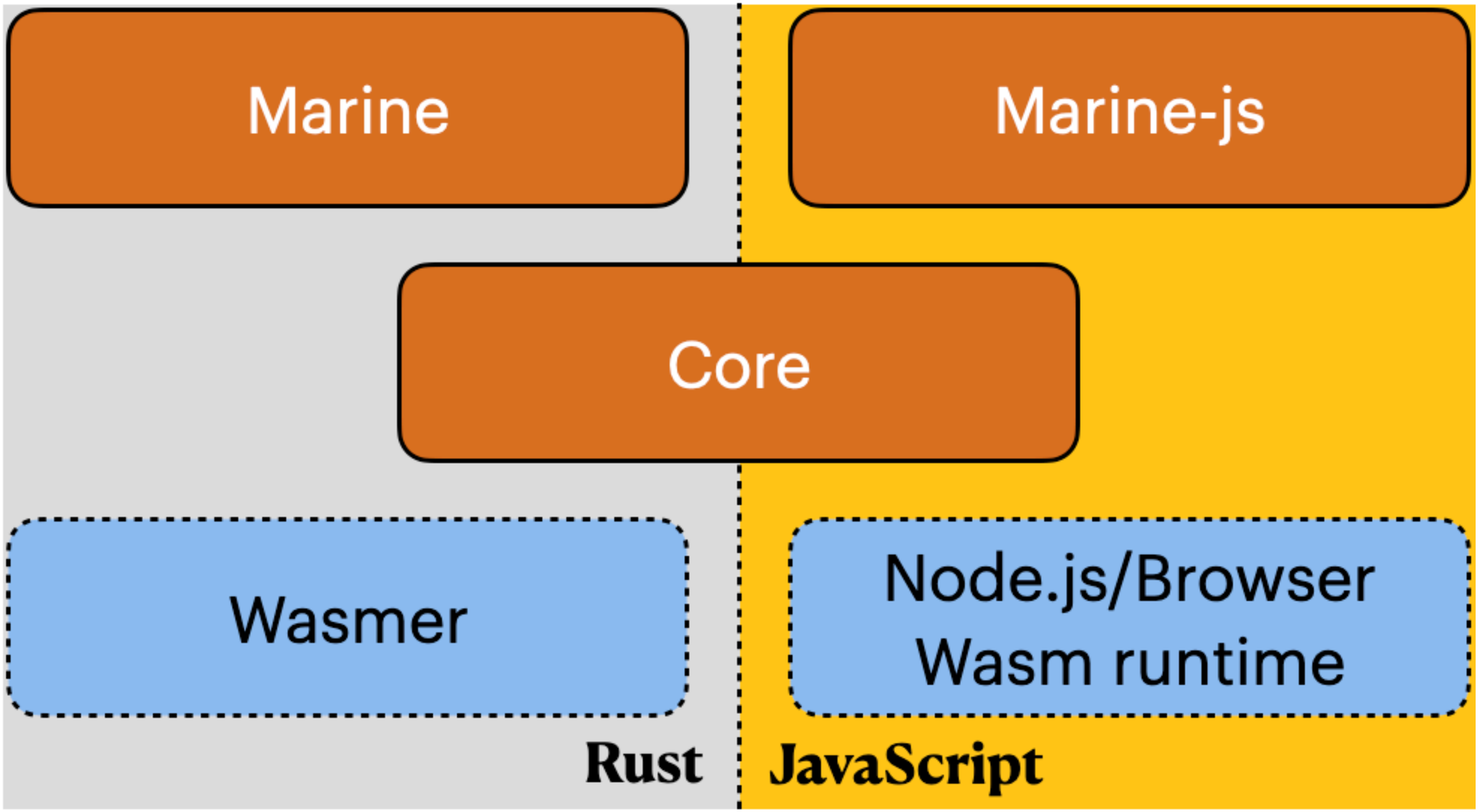
- Ability to **compose** modules

- Ability to run **server-side** and **client-side**

- Extensibility

- Fine-grained resource control

Marine architecture



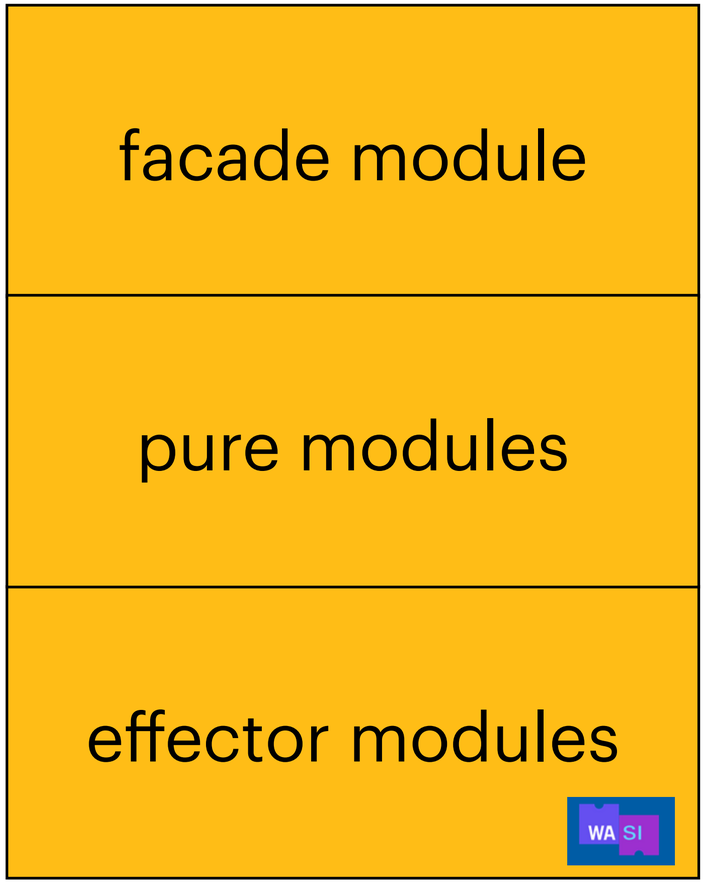
doc.fluence.dev/marine-book/marine-runtime/architecture

github.com/fluencelabs/marine

Fluence modules type

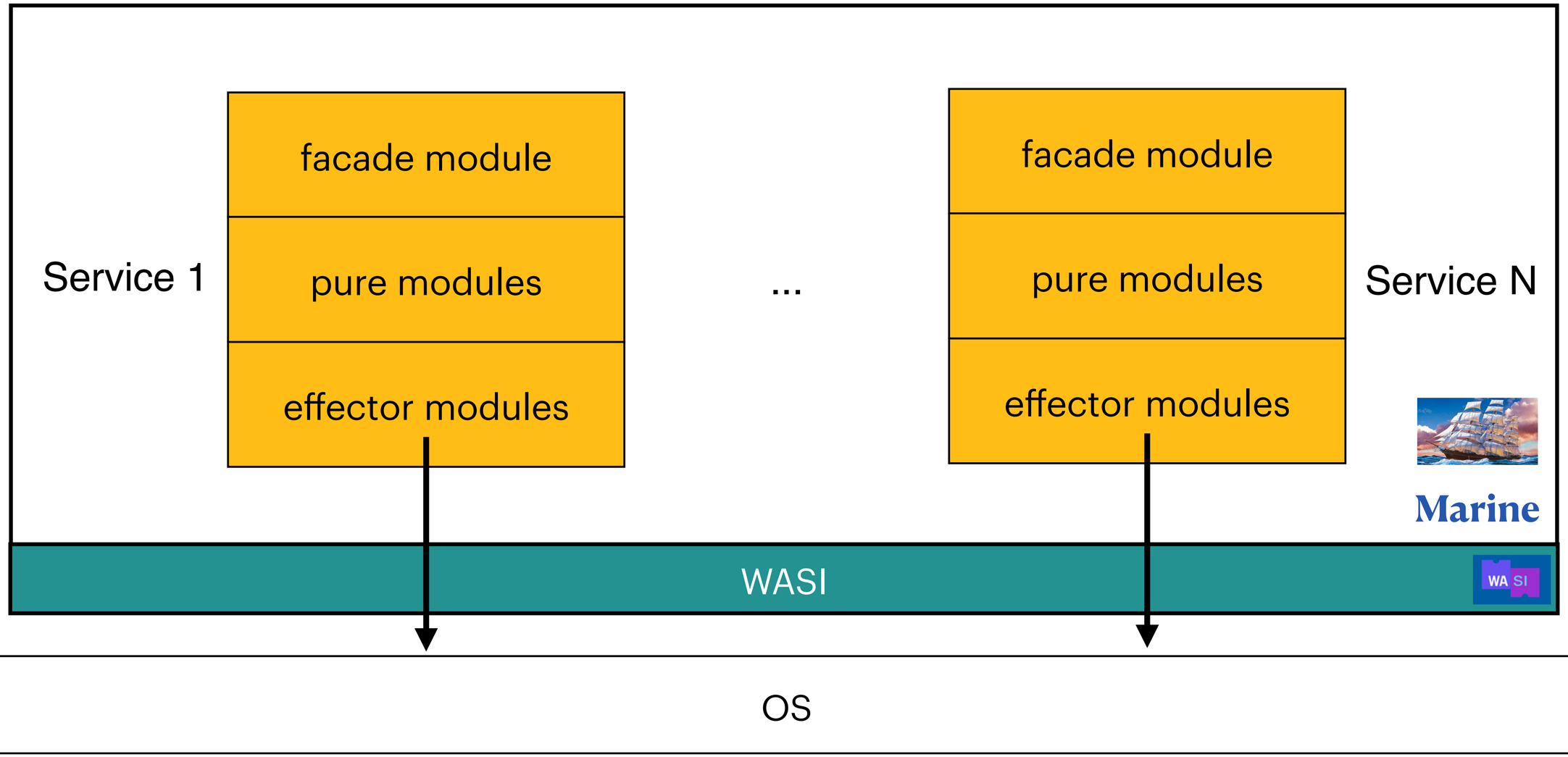
There are three types of modules:

- **facade** modules expose the API of an entire service that is callable from Aqua scripts
- **pure** modules contain pure logic, and they can't access the filesystem or call external binaries
- **effector** modules can access filesystem through WASI and call external binaries



Fluence service

Marine: multi-module Wasm runtime



Service configuration

```
[[module]]
    name = "local_storage"
    logger_enabled = true
    mem_pages_count = 1

    [module.wasi]
    preopened_files = ["./sites"]
    mapped_dirs = { "sites" = "./sites" }

[[module]]
    name = "curl_adapter"

    [module.mounted_binaries]
    curl = "/usr/bin/curl"

[[module]]
    name = "facade"
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

Service configuration

```
[[module]]
    name = "local_storage"
    logger_enabled = true
    mem_pages_count = 1

    [module.wasi]
    preopened_files = ["./sites"]
    mapped_dirs = { "sites" = "./sites" }

[[module]]
    name = "curl_adapter"

    [module.mounted_binaries]
    curl = "/usr/bin/curl"

[[module]]
    name = "facade"
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

Service configuration

```
[[module]]
    name = "local_storage"
    logger_enabled = true
    mem_pages_count = 1

    [module.wasi]
    preopened_files = ["./sites"]
    mapped_dirs = { "sites" = "./sites" }

[[module]]
    name = "curl_adapter"

    [module.mounted_binaries]
    curl = "/usr/bin/curl"

[[module]]
    name = "facade"
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

IValue

```
pub enum IValue {
    Boolean(bool),
    S8(i8),
    S16(i16),
    S32(i32),
    S64(i64),
    U8(u8),
    U16(u16),
    U32(u32),
    U64(u64),
    F32(f32),
    F64(f64),
    String(String),
    ByteArray(Vec<u8>),
    Array(Vec<IValue>),
    Record(NEVec<IValue>),
}
```

Host closures

```rust
pub type HostExportedFunc = Box<dyn Fn(&mut Ctx, Vec<IValue>) -> Option<IValue> + 'static>;

pub struct HostImportDescriptor {
    /// This closure will be invoked for corresponding import.
    pub host_exported_func: HostExportedFunc,

    /// Type of the closure arguments.
    pub argument_types: Vec<IType>,

    /// Types of output of the closure.
    pub output_type: Option<IType>,

    /// If Some, this closure is called with error when errors is encountered while lifting.
    /// If None, panic will occur.
    pub error_handler: Option<Box<dyn Fn(&HostImportError) -> Option<IValue> + 'static>>,
}
```

github.com/fluencelabs/marine

Host closures

```
pub(crate) fn create_mounted_binary_import(mounted_binary_path: String) -> HostImportDescriptor {
    let host_cmd_closure = move |_ctx: &mut Ctx, raw_args: Vec<IValue>| {
        let result =
            mounted_binary_import_impl(&mounted_binary_path, raw_args).unwrap_or_else(Into::into);

        let raw_result = crate::to_interface_value(&result).unwrap();

        Some(raw_result)
    };

    HostImportDescriptor {
        host_exported_func: Box::new(host_cmd_closure),
        argument_types: vec![IType::Array(Box::new(IType::String))],
        output_type: Some(IType::Record),
        error_handler: None,
    }
}
```

Host closures

```
pub(crate) fn create_mounted_binary_import(mounted_binary_path: String) -> HostImportDescriptor {
    let host_cmd_closure = move |_ctx: &mut Ctx, raw_args: Vec<IValue>| {
        let result =
            mounted_binary_import_impl(&mounted_binary_path, raw_args).unwrap_or_else(Into::into);

        let raw_result = crate::to_interface_value(&result).unwrap();

        Some(raw_result)
    };

    HostImportDescriptor {
        host_exported_func: Box::new(host_cmd_closure),
        argument_types: vec![IType::Array(Box::new(IType::String))],
        output_type: Some(IType::Record),
        error_handler: None,
    }
}
```

Host closures

```rust
pub(crate) fn create_mounted_binary_import(mounted_binary_path: String) -> HostImportDescriptor {
    let host_cmd_closure = move |_ctx: &mut Ctx, raw_args: Vec<IValue>| {
        let result =
            mounted_binary_import_impl(&mounted_binary_path, raw_args).unwrap_or_else(Into::into);

        let raw_result = crate::to_interface_value(&result).unwrap();

        Some(raw_result)
    };

    HostImportDescriptor {
        host_exported_func: Box::new(host_cmd_closure),
        argument_types: vec![IType::Array(Box::new(IType::String))],
        output_type: Some(IType::Record),
        error_handler: None,
    }
}
```

Marine SDK & tools

Fluence Hello world

```
use marine_rs_sdk::marine;

pub fn main() {}

#[marine]
pub fn greeting(name: String) -> String {
    format!("Hi, {}", name)
}
```

github.com/fluencelabs/marine/tree/master/examples/greeting

url-downloader service

```
[[module]]
    name = "local_storage"
    mem_pages_count=1

    [module.wasi]
    preopened_files = ["./sites"]
    mapped_dirs = { "sites" = "./sites" }

[[module]]
    name = "curl_adapter"

    [module.mounted_binaries]
    curl = "/usr/bin/curl"

[[module]]
    name = "facade"
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

Modules linking

```rust
#[marine]
pub fn get_n_save(url: String, file_name: String) -> String {
    let result = download(url);
    file_put(file_name, result.into_bytes());
    String::from("Ok")
}

#[marine]
#[link(wasm_import_module = "curl_adapter")]
extern "C" {
    pub fn download(url: String) -> String;
}

#[marine]
#[link(wasm_import_module = "local_storage")]
extern "C" {
    #[link_name = "put"]
    pub fn file_put(name: String, file_content: Vec<u8>) -> String;
}
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

Modules linking

```rust
#[marine]
pub fn get_n_save(url: String, file_name: String) -> String {
    let result = download(url);
    file_put(file_name, result.into_bytes());
    String::from("Ok")
}

#[marine]
#[link(wasm_import_module = "curl_adapter")]
extern "C" {
    pub fn download(url: String) -> String;
}

#[marine]
#[link(wasm_import_module = "local_storage")]
extern "C" {
    #[link_name = "put"]
    pub fn file_put(name: String, file_content: Vec<u8>) -> String;
}
```

github.com/fluencelabs/marine/tree/master/examples/url-downloader

Modules linking

```rust
#[marine]
pub fn get_n_save(url: String, file_name: String) -> String {
    let result = download(url);
    file_put(file_name, result.into_bytes());
    String::from("Ok")
}

#[marine]
#[link(wasm_import_module = "curl_adapter")]
extern "C" {
    pub fn download(url: String) -> String;
}

#[marine]
#[link(wasm_import_module = "local_storage")]
extern "C" {
    #[link_name = "put"]
    pub fn file_put(name: String, file_content: Vec<u8>) -> String;
}
```

facade.wasm

```rust
#[marine]
pub fn download(url: String) -> String {
    let result = curl(vec![url]);
    String::from_utf8(result.stdout).unwrap()
}

#[marine]
#[link(wasm_import_module = "host")]
extern "C" {
    fn curl(cmd: Vec<String>) -> MountedBinaryResult;
}
```

curl.wasm

```rust
#[marine]
pub fn put(name: String, file_content: Vec<u8>) -> String {
    let rpc_tmp_filepath = format!("{}{}", SITES_DIR, name);
    let result = fs::write(PathBuf::from(rpc_tmp_filepath.clone()), file_content);
    if let Err(e) = result {
        return format!("file can't be written: {}", e);
    }
    String::from("Ok")
}
```

local_storage.wasm

github.com/fluencelabs/marine/tree/master/examples/url-downloader

More comprehensive example

```rust
#[marine]
pub struct TestRecord0 {
    pub field_0: i32,
}

#[marine]
pub struct TestRecord1 {
    pub field_0: i32,
    pub field_1: String,
    pub field_2: Vec<u8>,
    pub test_record_0: TestRecord0,
}
```

```rust
#[marine]
#[link(wasm_import_module = "records_passing_effector")]
extern "C" {
    pub fn test_record_ref(test_record: &TestRecord1) -> TestRecord0;

    pub fn test_record_array(test_record: &Vec<TestRecord1>) -> Vec<TestRecord1>;
}
```

github.com/fluencelabs/marine/blob/master/fluence-faas/tests/wasm_tests/records_passing

Marine JS side

Marine rust service

```rust
#[marine]
pub struct MyStruct {
    data1: String,
    data2: Vector<String>,
}

#[marine]
pub struct MyResult {
    ret_code: i32,
}

#[marine]
pub fn service_func(id: &str, object: MuStruct) ->
MyResult {
    // do something
}
```

How to call it using Marine-JS

```rust
let result = call_module("module_name", "service_function",
    '["id",' +
    ' { ' +
    '   "data1": "test",' +
    '   "data2": ["a", "b"]' +
    '}' +
    ']');

if (result.error.length() > 0) {
    throw result.error;
}

check_ret_code(result.result.retCode);
```

Marine tests: modules testing

```rust
#[marine]
pub fn greeting(name: String) -> String {
    format!("Hi, {}", name)
}

#[cfg(test)]
mod tests {
    #[marine_test(config_path = "../Config.toml", modules_dir = "../artifacts")]
    fn empty_string(greeting: marine_test_env::greeting::ModuleInterface) {
        let actual = greeting.greeting(String::new());
        assert_eq!(actual, "Hi, ");
    }
}
```

github.com/fluencelabs/marine/tree/master/examples/greeting

Marine tests: modules testing

```rust
#[marine]
pub fn greeting(name: String) -> String {
    format!("Hi, {}", name)
}

#[cfg(test)]
mod tests {
    #[marine_test(config_path = "../Config.toml", modules_dir = "../artifacts")]
    fn empty_string(greeting: marine_test_env::greeting::ModuleInterface) {
        let actual = greeting.greeting(String::new());
        assert_eq!(actual, "Hi, ");
    }
}
```

github.com/fluencelabs/marine/tree/master/examples/greeting

Marine tests: services testing

```
#[marine_test(
    service_1(
        config_path = "../service_1/Config.toml",
        modules_dir = "../service_1/artifacts"
    ),
    service_2(
        config_path = "../service_2/Config.toml",
        modules_dir = "../service_2/artifacts"
    ),
)]
fn test() {
    let service_1 = service_1::ServiceInterface::new();
    let service_2 = service_2::ServiceInterface::new();
    ...
    let service_1_res = service_1.process_data(data);
    let service_2_res = service_2.process_data(service_1_res);
    ...
}
```

github.com/fluencelabs/marine/tree/master/examples/greeting

Marine tests: services testing

```
#[marine_test(
    service_1(
        config_path = "../service_1/Config.toml",
        modules_dir = "../service_1/artifacts"
    ),
    service_2(
        config_path = "../service_2/Config.toml",
        modules_dir = "../service_2/artifacts"
    ),
)]
fn test() {
    let service_1 = service_1::ServiceInterface::new();
    let service_2 = service_2::ServiceInterface::new();
    ...
    let service_1_res = service_1.process_data(data);
    let service_2_res = service_2.process_data(service_1_res);
    ...
}
```

github.com/fluencelabs/marine/tree/master/examples/greeting

How do multi-module calls work?

```rust
#[marine]
pub fn get_n_save(url: String, file_name: String) -> String {
    let result = download(url);
    file_put(file_name, result.into_bytes());
    String::from("Ok")
}
```

facade.wasm

```rust
#[marine]
pub fn download(url: String) -> String {
    let result = curl(vec![url]);
    String::from_utf8(result.stdout).unwrap()
}
```

curl.wasm

```rust
#[marine]
pub fn put(name: String, file_content: Vec<u8>) -> String {
    let rpc_tmp_filepath = format!("{}{}", SITES_DIR, name);
    let result = fs::write(PathBuf::from(rpc_tmp_filepath.clone()), file_content);
    if let Err(e) = result {
        return format!("file can't be written: {}", e);
    }
    String::from("Ok")
}
```

local_storage.wasm

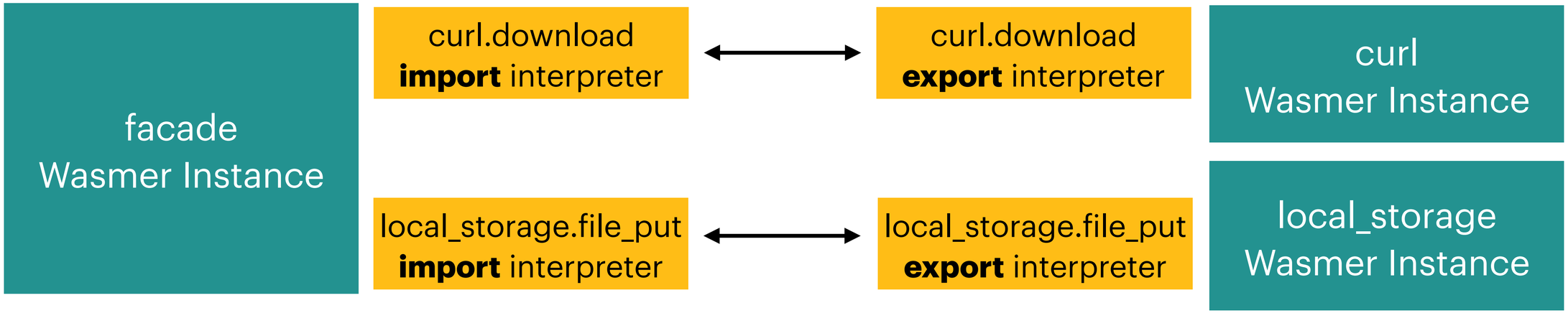
How do multi-module calls work?

```rust
#[marine]
pub fn get_n_save(url: String, file_name: String) -> String {
    let result = download(url);
    file_put(file_name, result.into_bytes());
    String::from("Ok")
}
```

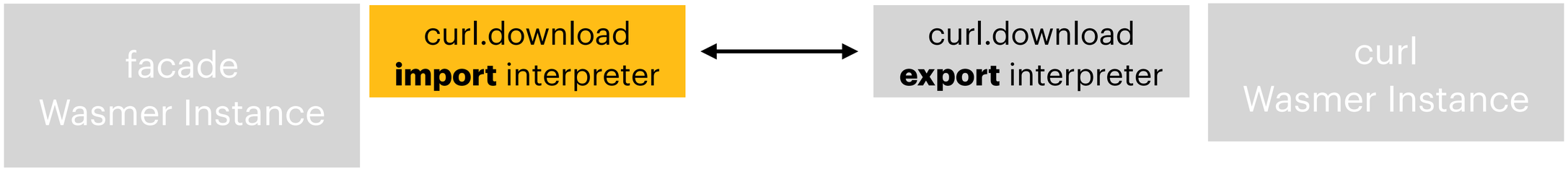
facade.wasm

```rust
#[marine]
pub fn download(url: String) -> String {
    let result = curl(vec![url]);
    String::from_utf8(result.stdout).unwrap()
}
```

curl.wasm



How do multi-module calls work?



1. Adapter of the imported function ***curl*** from curl.wasm is called

```
(@interface func (import adapter curl.curl)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 9          ;; call curl.curl adaptor
    dup
    string.size
    call-core 0
    swap2
    string.lower_memory
)
```

```
(@interface func (export adapter curl.curl)
    arg.get 0
    string.size
    call-core 0          ;; call allocate
    arg.get 0
    string.lower_memory
    call-core 7          ;; call self.curl
    string.lift_memory
)
```

How do multi-module calls work?

facade
Wasmer Instance

curl.download
import interpreter

curl.download
export interpreter

curl
Wasmer Instance

```
(@interface func (import adapter curl.download)
    arg.get 0           ;; push a pointer to a string
    arg.get 1           ;; push a size of a string
    string.lift_memory  ;; lift a string from
                        ;; pointer and size
    call-core 1         ;; call release_objects
    call-core 7         ;; call curl.download
    dup
    string.size
    i32.push 1
    call-core 0
    swap2
    string.lower_memory
    call-core 4
    call-core 5
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0
    string.size
    i32.push 1
    call-core 0
    arg.get 0
    string.lower_memory
    call-core 6
    call-core 3
    call-core 2
    string.lift_memory
    call-core 1
)
```

How do multi-module calls work?

facade
Wasmer Instance

curl.download
import interpreter

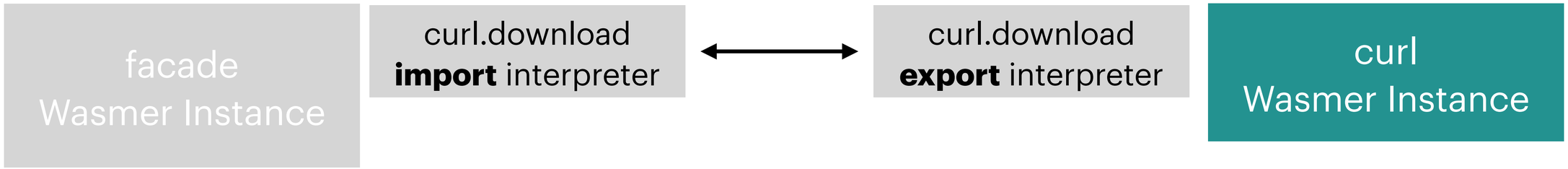
curl.download
export interpreter

curl
Wasmer Instance

```
(@interface func (import adapter curl.download)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 1
    call-core 7
    dup
    string.size
    i32.push 1
    call-core 0
    swap2
    string.lower_memory
    call-core 4
    call-core 5
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0             ;; push a provided string
    string.size           ;; take a string from the
                          ;; stack and push it size
    i32.push 1            ;; push 1i32 (an allocation tag)
    call-core 0           ;; call curl.allocate function
    arg.get 0             ;; push a provided string
    string.lower_memory   ;; lowers a string from the stack
                          ;; to a provided by previous allocate
                          ;; memory region
    call-core 6
    call-core 3
    call-core 2
    string.lift_memory
    call-core 1
)
```

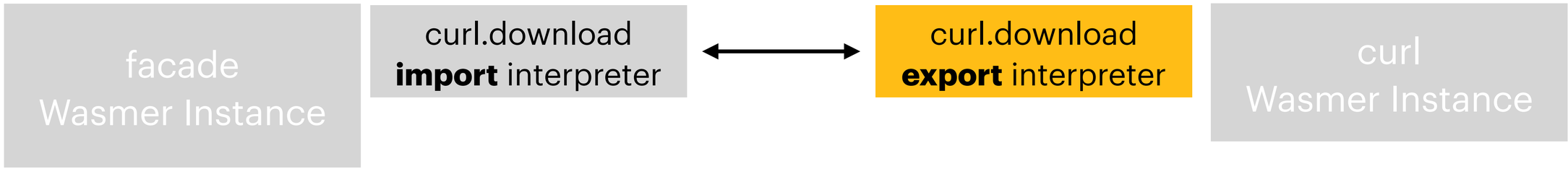
How do multi-module calls work?




```
(@interface func (import adapter curl.download)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 1
    call-core 7
    dup
    string.size
    i32.push 1
    call-core 0
    swap2
    string.lower_memory
    call-core 4
    call-core 5
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0
    string.size
    i32.push 1
    call-core 0
    arg.get 0
    string.lower_memory
    call-core 6          ;; call curl.download
                         ;; from a raw Wasm module
    call-core 3
    call-core 2
    string.lift_memory
    call-core 1
)
```

How do multi-module calls work?




```
(@interface func (import adapter curl.download)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 1
    call-core 7
    dup
    string.size
    i32.push 1
    call-core 0
    swap2
    string.lower_memory
    call-core 4
    call-core 5
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0
    string.size
    i32.push 1
    call-core 0
    arg.get 0
    string.lower_memory
    call-core 6
    call-core 3          ;; call get_result_ptr
    call-core 2          ;; call get_result_size
    string.lift_memory   ;; lift a string from
                         ;; a pointer and a size
    call-core 1          ;; call release_objects
)
```

How do multi-module calls work?

facade
Wasmer Instance

curl.download
import interpreter

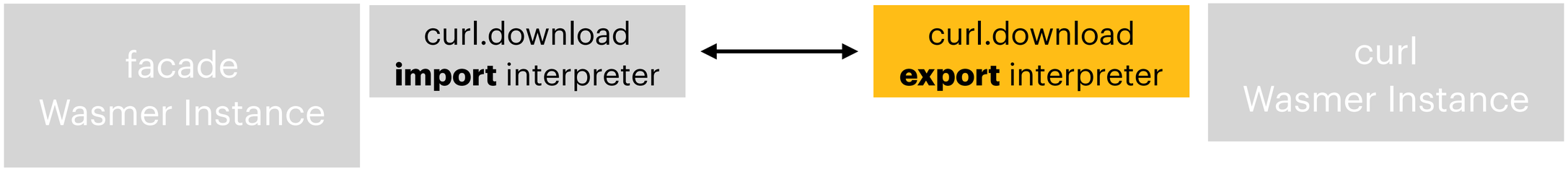
curl.download
export interpreter

curl
Wasmer Instance

```
(@interface func (import adapter curl.download)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 1
    call-core 7
    dup                  ;; duplicate a string on the stack
    string.size          ;; take a string from the
                         ;; stack and push it size
    i32.push 1           ;; push 1i32 (an allocation tag)
    call-core 0          ;; call curl.allocate function
    swap2                ;; swapping operands on a stack
    string.lower_memory  ;; lowers a string from the stack
                         ;; to a provided by previous allocate
                         ;; memory region
    call-core 4          ;; call set_result_size
    call-core 5          ;; call set_result_ptr
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0
    string.size
    i32.push 1
    call-core 0
    arg.get 0
    string.lower_memory
    call-core 6
    call-core 3
    call-core 2
    string.lift_memory
    call-core 1
)
```

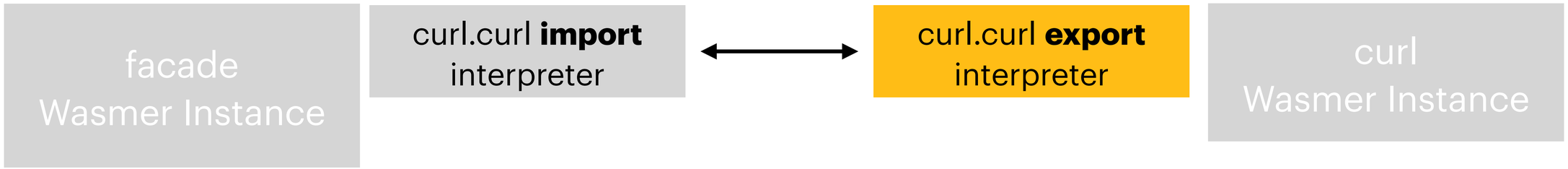
How do multi-module calls work?




```
(@interface func (import adapter curl.download)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 1
    call-core 7
    dup
    string.size
    i32.push 1
    call-core 0
    swap2
    string.lower_memory
    call-core 4
    call-core 5
)
```

```
(@interface func (export adapter curl.download)
    arg.get 0             ;; push a provided string
    string.size           ;; take a string from the
                          ;; operand stack and push it size
    i32.push 1            ;; push 1i32 (an allocation tag)
    call-core 0           ;; call curl.allocate function
    arg.get 0             ;; push a provided string
    string.lower_memory   ;; lowers a string from the stack
                          ;; to a provided by previous allocate
                          ;; memory region
    call-core 6
    call-core 3
    call-core 2
    string.lift_memory
    call-core 1
)
```

How do multi-module calls work?

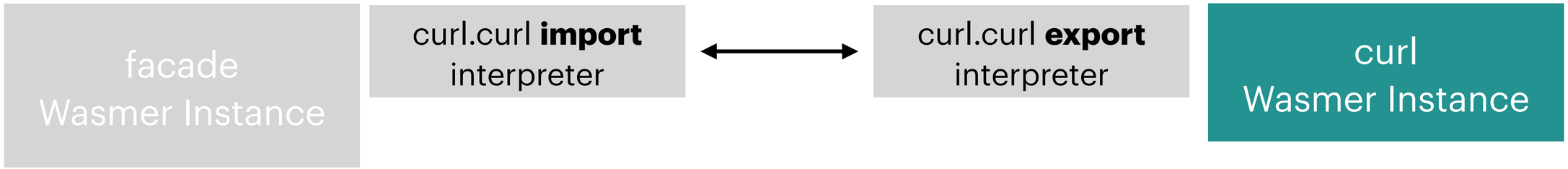


2. The interpreter of imported ***curl*** function calls the adaptor for exported function ***curl*** from curl.wasm module

```
(@interface func (import adapter curl.curl)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 9          ;; call curl.curl adaptor
    dup
    string.size
    call-core 0
    swap2
    string.lower_memory
)
```

```
(@interface func (export adapter curl.curl)
    arg.get 0
    string.size
    call-core 0          ;; call allocate
    arg.get 0
    string.lower_memory
    call-core 7          ;; call self.curl
    string.lift_memory
)
```

How do multi-module calls work?

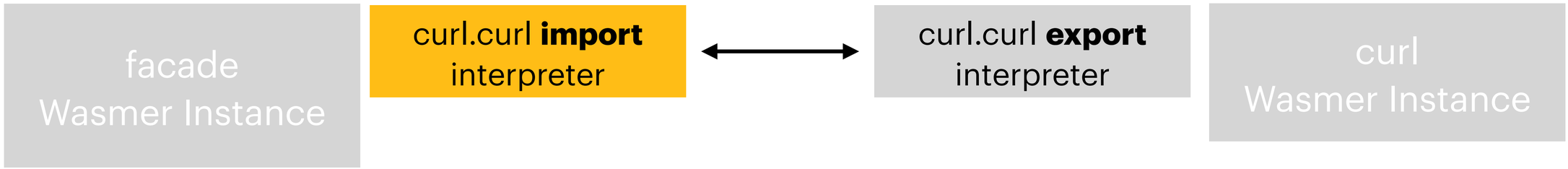


3. The "real" function ***curl*** from curl.wasm is called (call-core 7)

```
(@interface func (import adapter curl.curl)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 9          ;; call curl.curl adaptor
    dup
    string.size
    call-core 0
    swap2
    string.lower_memory
)
```

```
(@interface func (export adapter curl.curl)
    arg.get 0
    string.size
    call-core 0          ;; call allocate
    arg.get 0
    string.lower_memory
    call-core 7          ;; call self.curl
    string.lift_memory
)
```

How do multi-module calls work?



4. Execution flow is passed back to the imported adapter of facade.wasm module

```
(@interface func (import adapter curl.curl)
    arg.get 0
    arg.get 1
    string.lift_memory
    call-core 9          ;; call curl.curl adaptor
    dup
    string.size
    call-core 0
    swap2
    string.lower_memory
)
```

```
(@interface func (export adapter curl.curl)
    arg.get 0
    string.size
    call-core 0          ;; call allocate
    arg.get 0
    string.lower_memory
    call-core 7          ;; call self.curl
    string.lift_memory
)
```

REPL

```
[18:02] :ipfs-node (badge *) | mrepl Config.toml
Welcome to the Marine REPL (version 0.9.1)
Minimal supported versions
  sdk: 0.6.0
  interface-types: 0.20.0

app service was created with service id = 9807c2e7-804e-427a-935f-924e61994ede
elapsed time 82.119658ms

1> i
Loaded modules interface:

ipfs_effector:
  fn put(file_path: string) -> string
  fn get(hash: string) -> string
  fn get_address() -> string
ipfs_pure:
  fn put(file_content: []u8) -> string
  fn invoke() -> string
  fn get(hash: string) -> []u8

2> call ipfs_pure put [[1,2,3]]
result: String("QmanYeBG6APPbZZgAgmzM9nkQpvcBRwAb8AYgdERwH6Amb\n")
 elapsed time: 170.740194ms

3> call ipfs_pure get "QmanYeBG6APPbZZgAgmzM9nkQpvcBRwAb8AYgdERwH6Amb"
result: Array([Number(1), Number(2), Number(3)])
 elapsed time: 151.219771ms
```

The project overview

Marine VM - github.com/fluencelabs/marine

Marine book - doc.fluence.dev/marine-book/

Rust sdk - github.com/fluencelabs/rust-sdk

Aqua - github.com/fluencelabs/aqua

The project updates

- twitter.com/fluence_project
- fluence.network/ (newsletter)
- t.me/fluencedev - the telegram channel

My email: mike@fluence.one, tg: @voronovm

Thanks!

`fluence.network`