FERMYON

WEBASSEMBLY COMPONENT MODEL

WHAT? HOW? AND, WHY YOU SHOULD NOT IGNORE IT!

MARCH 2024

Thorsten Hans

Cloud Advocate @ Fermyon Technologies

mail: thorsten.hans@fermyon.com

x: @ThorstenHans

web: fermyon.com

thorsten-hans.com

FERMYON

WebAssembly Component Model

What Is It

FERMYON

What is the WebAssembly Component Model

The WebAssembly Component Model is an addition to core WebAssembly that addresses shortcomings and streamlines interacting with WebAssembly modules

FERMYON

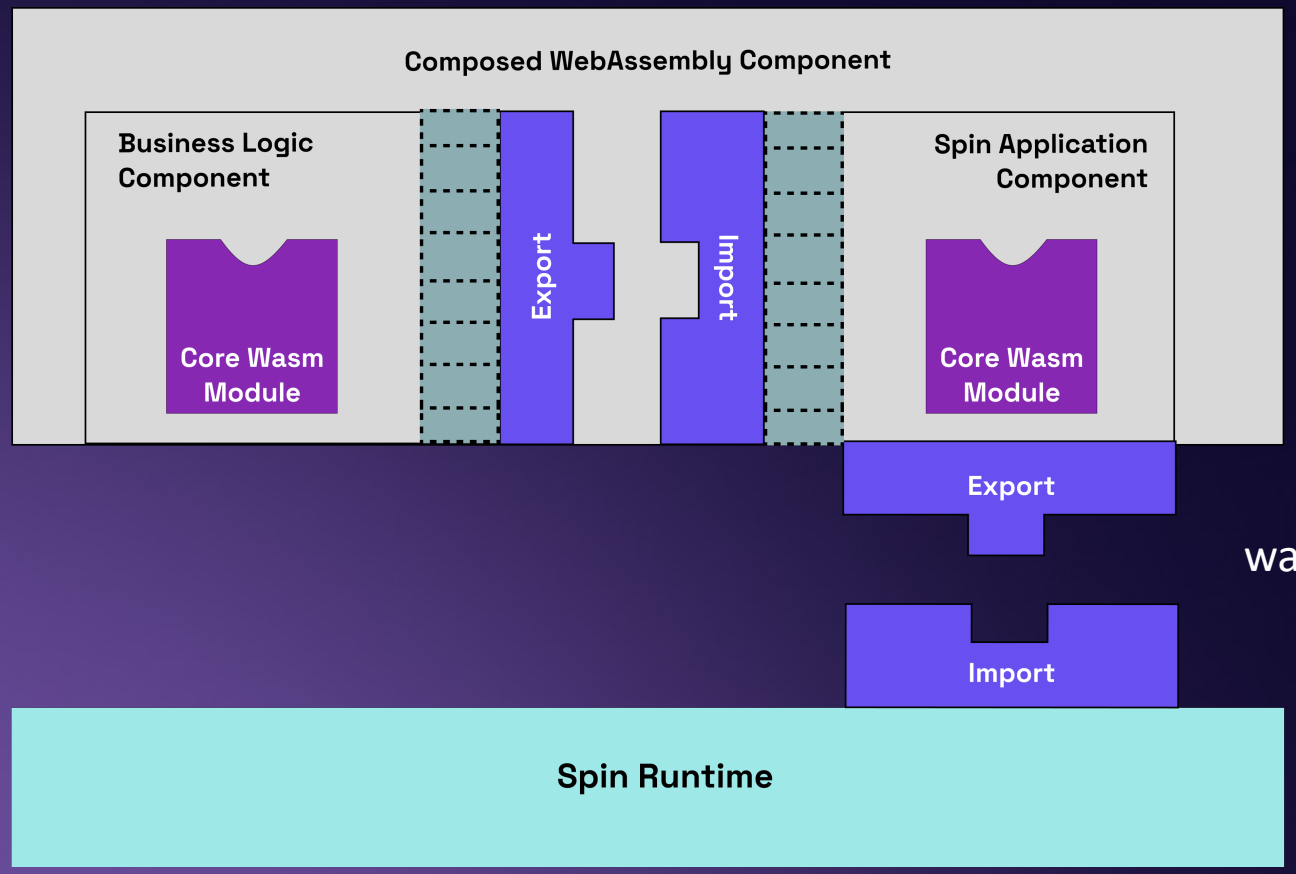
What is the WebAssembly Component Model

- Think of the Wasm Component Model as a wrapper around your "Core" WebAssembly modules
- It defines a canonical ABI which all components adhere to
  - Specifying and standardizing data layout in memory
  - Giving us richer types such as strings, structs, lists

What is the WebAssembly Component Model

- We use WebAssembly Interface Type (WIT) as Interface Description Language (IDL) to define contracts
- Ultimately, the Wasm Component Model dramatically simplifies using components written in different languages

What is the WebAssembly Component Model



wasi:http/incoming-handler

FERMYON

WebAssembly Component Model

Why You Should Not Ignore It

FERMYON

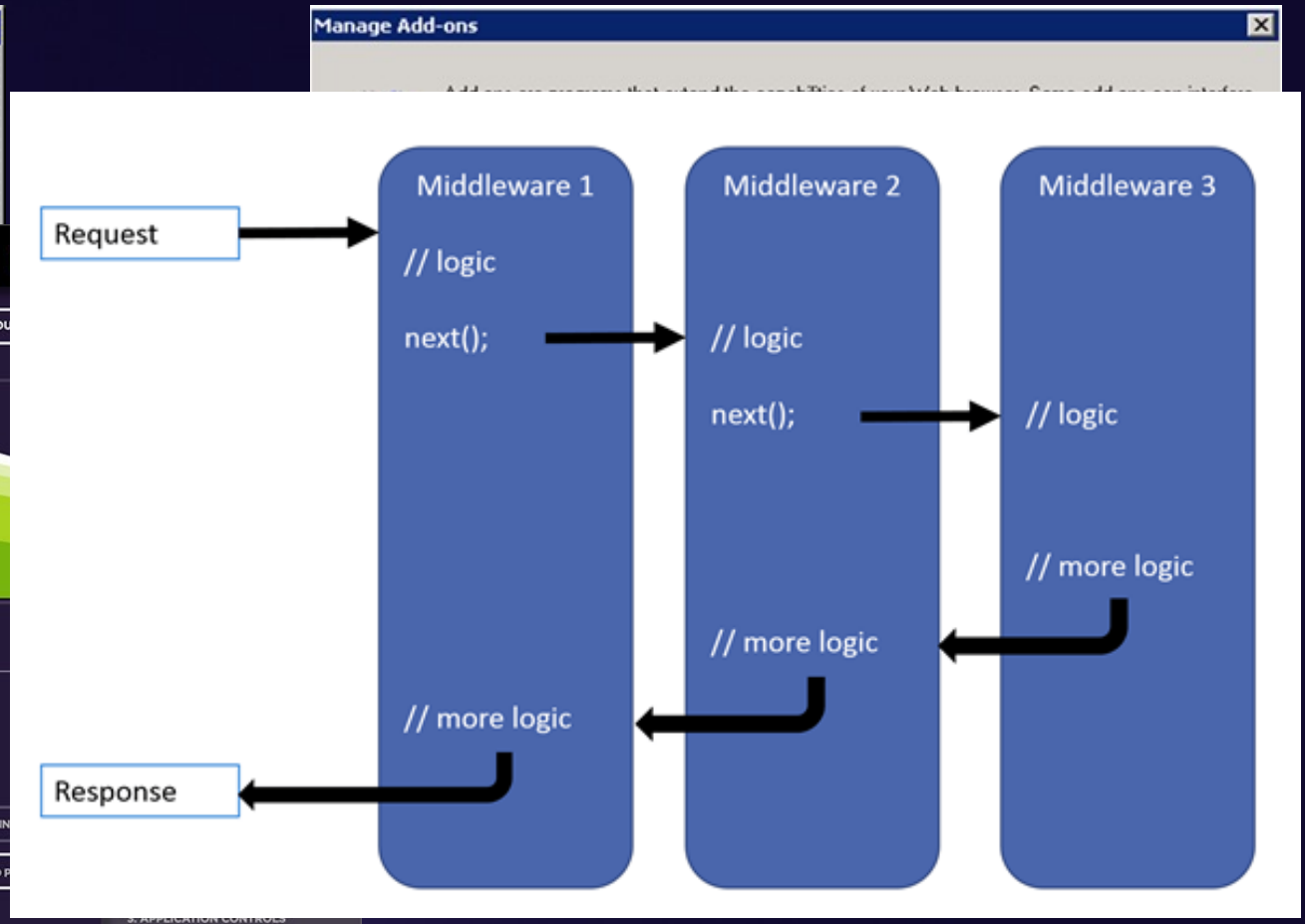
Why should you not ignore it!

- Components have a long history in software development
  - Rooting back to Douglas McIlroy in 1968
- Developers gained productivity by composing applications using different components

Why should you not ignore it!

- Numerous technologies were successful due to having a component model & allowing component composition
- Some famous representatives are (my take)
  - COM & ActiveX
  - Open Web Interface for .NET (OWIN)
  - Web Components

FERMYON



DataQ Instruments LabWindows Controls. Introduction to ActiveX
https://tinyurl.com/wasmio01
https://tinyurl.com/wasmio02
https://tinyurl.com/wasmio03

Infragistics ASP.NET Core Middlewares
https://tinyurl.com/wasmio04
https://tinyurl.com/wasmio05

FERMYON

Why should you not ignore it!

- WebAssembly makes apps portable across platforms & architectures
- The Wasm Component Model takes portability one step further
  - Components are portable across programming languages

FERMYON

WebAssembly Component Model

How You Can Use It Today

FERMYON

From Source to a Wasm Component

- Language specific tooling is available to build WebAssembly Components
  - For Python we use componentize-py
  - For JavaScript we use jco and componentize-js
  - For Rust we use cargo-component

wasm-tools

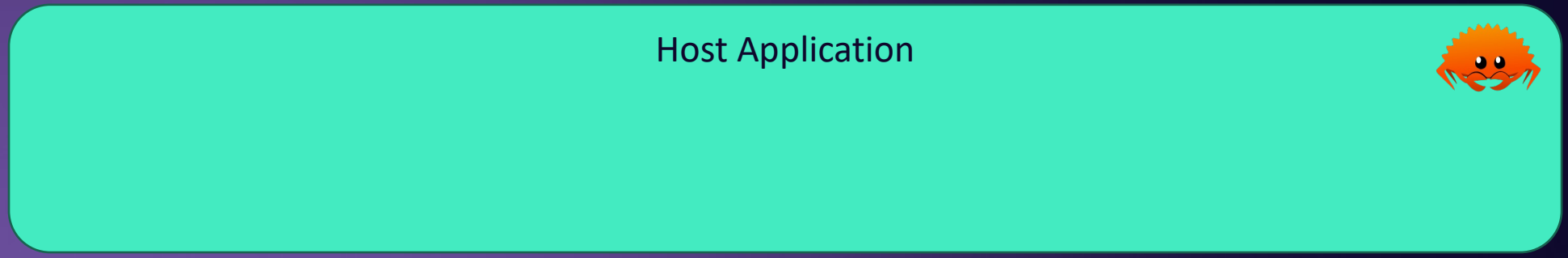
- CLI & library
- github.com/bytecodealliance/wasm-tools
- Tool for manipulate Wasm modules & components
  - Composing Wasm Components with wasm-tools compose
  - Inspect a WIT contract with wasm-tools component wit

WebAssembly Composition (WAC)

- CLI for composing WebAssembly Components
- https://github.com/peterhuene/wac
- Provides the WAC DSL to describe how components should be composed together
- Find further information on integration with wasm-tools at https://github.com/peterhuene/wac/issues/35

FERMYON

DEMO – Extensibility with Wasm Components



FERMYON

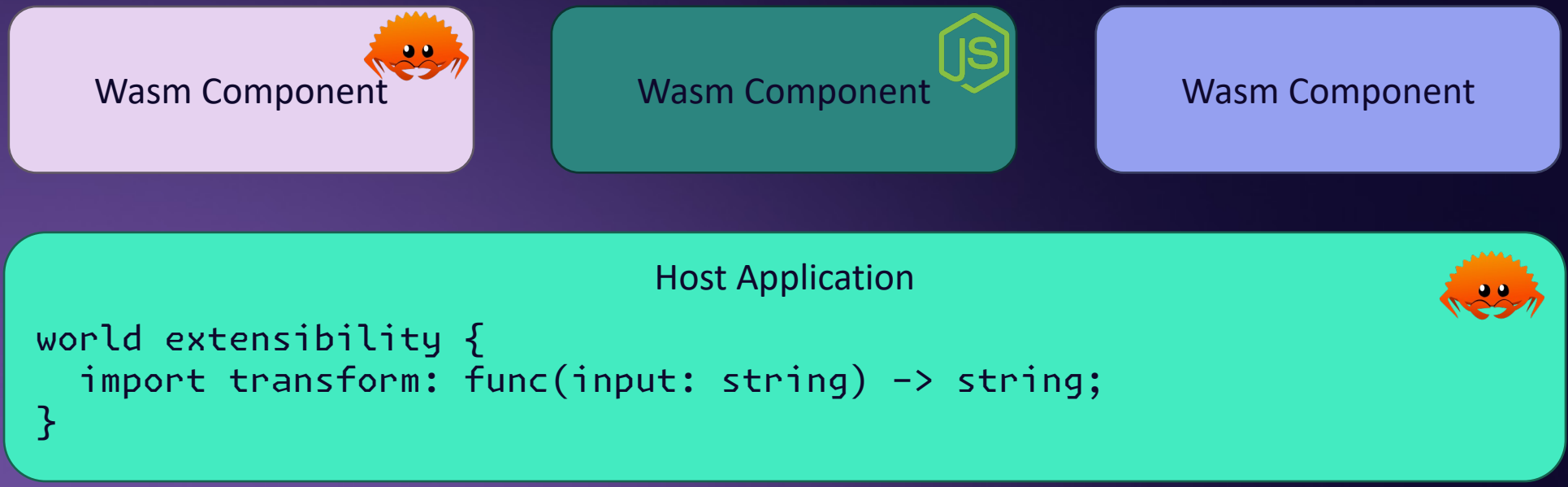
DEMO – Extensibility with Wasm Components

Host Application

```
world extensibility {
  import transform: func(input: string) -> string;
}
```

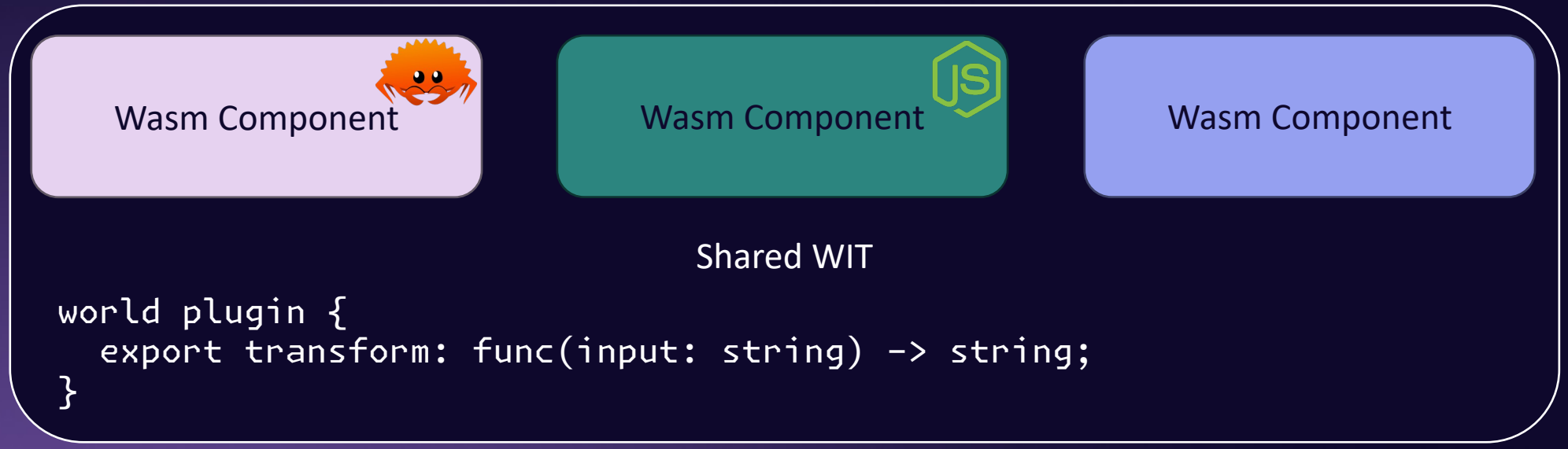
FERMYON

DEMO – Extensibility with Wasm Components



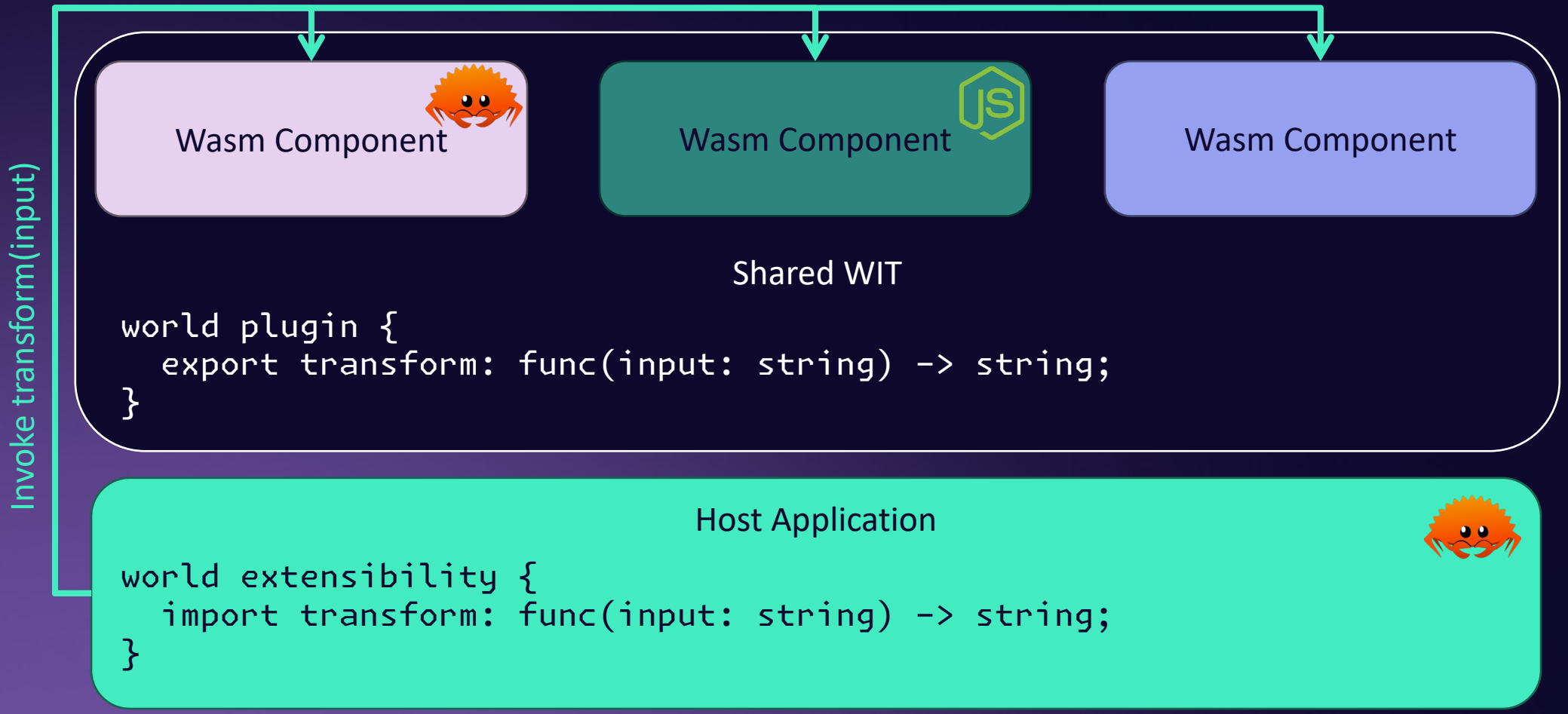
FERMYON

DEMO – Extensibility with Wasm Components



FERMYON

DEMO – Extensibility with Wasm Components



FERMYON

DEMO

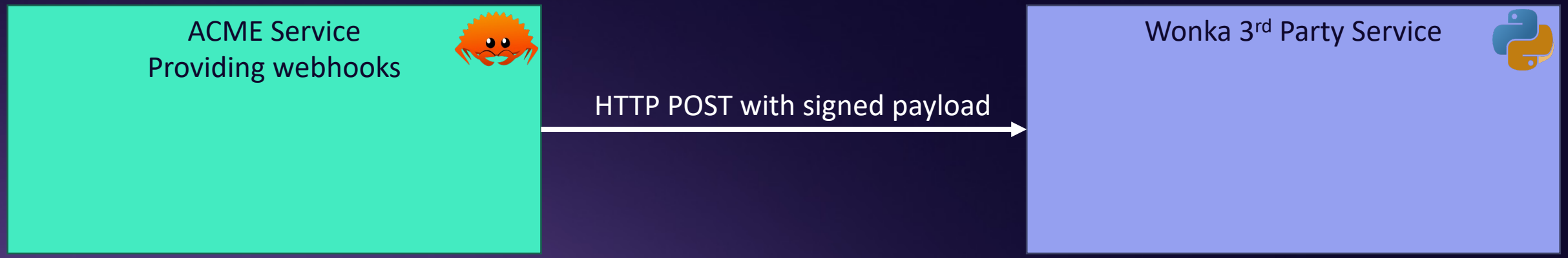
Extensibility using the Wasm Component Model

FERMYON

DEMO – Middlewares & Portability

FERMYON

DEMO – Middlewares & Portability



FERMYON

DEMO – Middlewares & Portability

ACME Service
Providing webhooks

```
world signing {
  import sign(...) -> list<u8>;
}
```

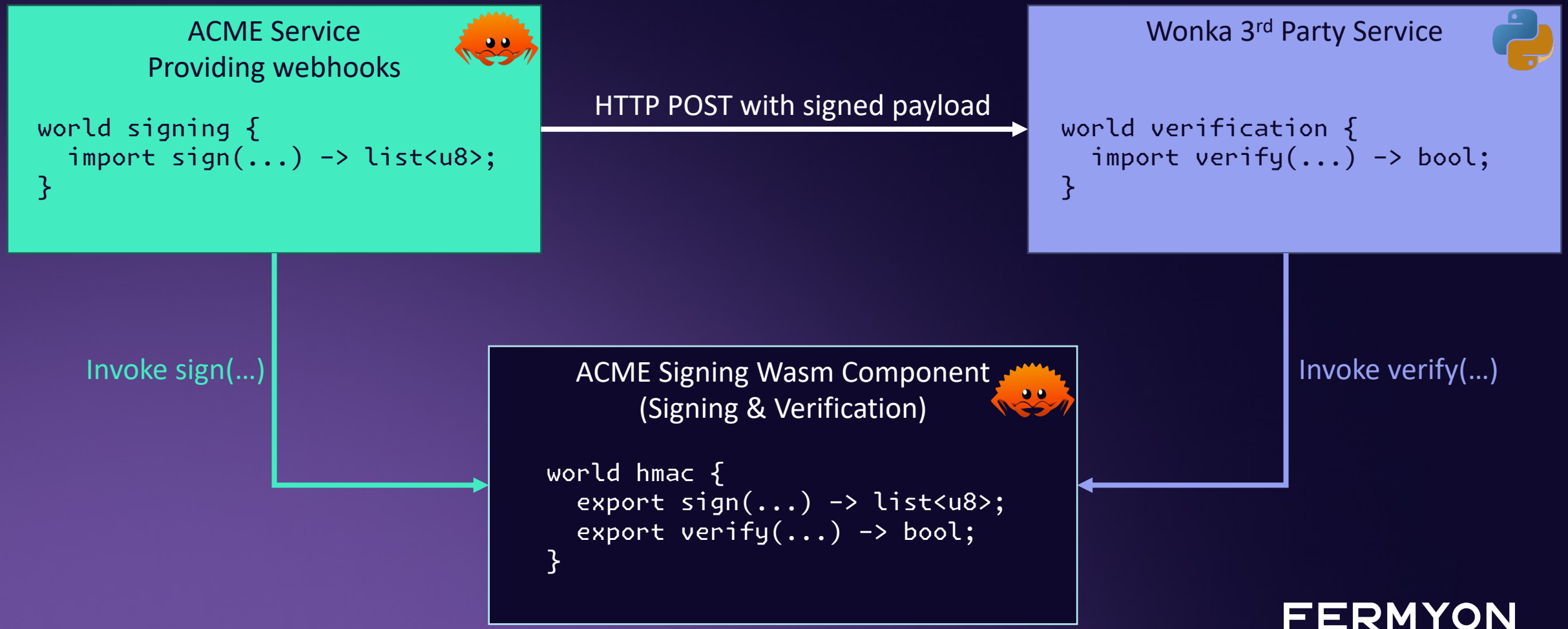
HTTP POST with signed payload

Wonka 3rd Party Service

```
world verification {
  import verify(...) -> bool;
}
```

FERMYON

DEMO – Middlewares & Portability



FERMYON

DEMO

Middlewares & Portability using the Wasm Component Model

FERMYON

Key Takeaways

- Wasm Components are self-contained units of work that allow interaction via rich contracts defined in WIT
- We can load components dynamically
- We compose bigger systems from components written in different languages without worrying about how data structures are represented in memory

Do you want to dive deeper?

Join Ryan's talk tomorrow (March 15th) @ 12:40

Deconstructing WebAssembly Components

FERMYON

Thank you!

github.com/ThorstenHans/wasmio-2024-demos

@fermyontech