FERMYON

AN INTRODUCTION TO THE WEBASSEMBLY COMPONENT MODEL

Mikkel Mørk Hegnhøj, Fermyon

November 2024

WebAssembly

FERMYON

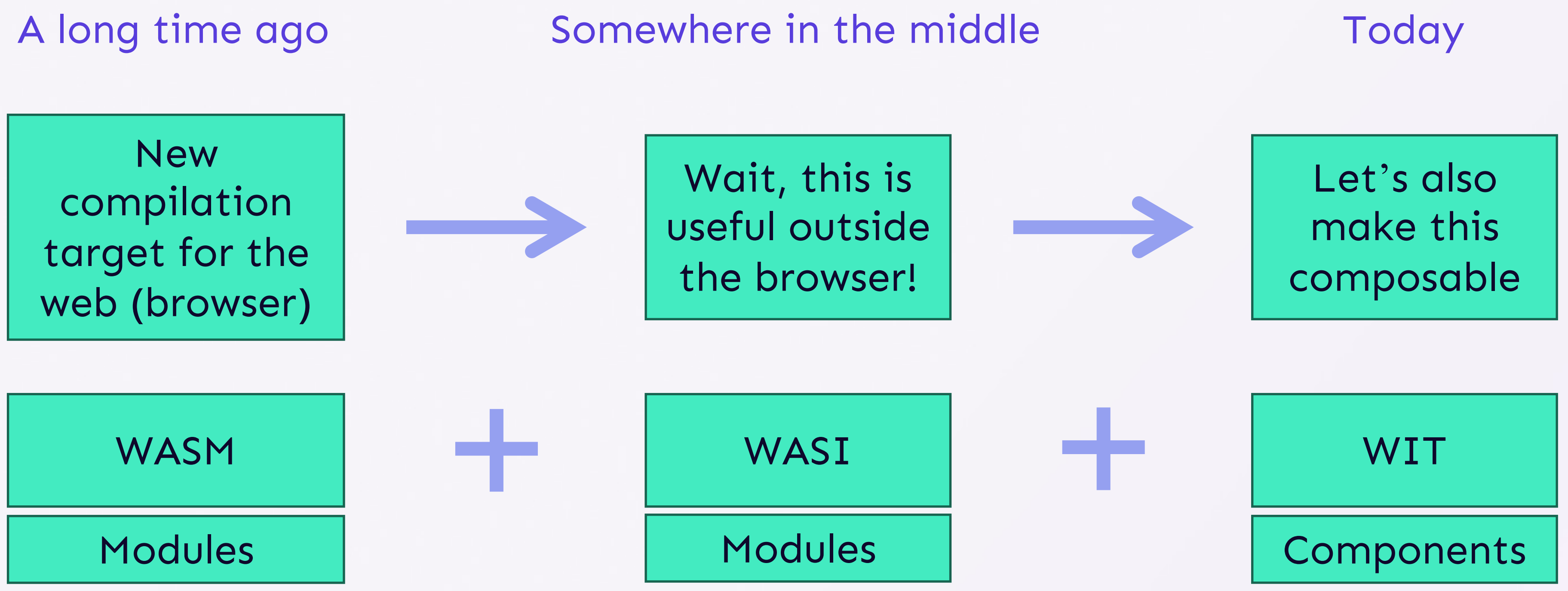
What’s The Problem And The Goal?

Expand programming language support for the web (browser), as a response to what we learned from Flash, Silverlight, and Java Applets.

Build a safe, portable, low-level code format designed for efficient execution and compact representation. The main goal of WebAssembly is to enable high performance applications on the Web, but it does not make any Web-specific assumptions or provide Web-specific features, so it can be employed in other environments as well.

How it Unfolded

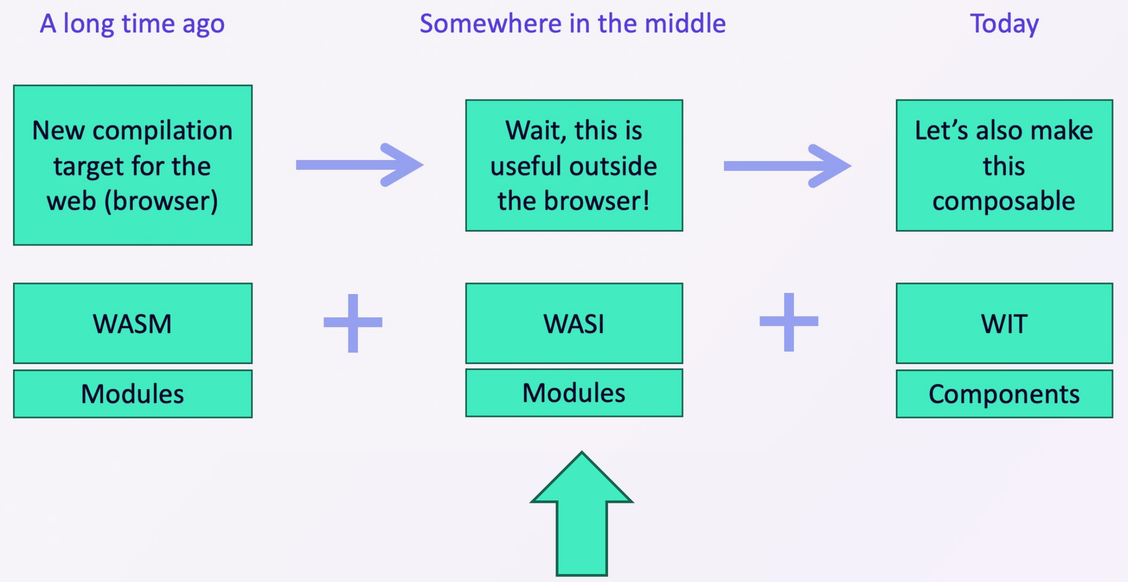
The unofficial very rough timeline of events


A long time ago
Somewhere in the middle
Today
New compilation target for the web (browser)
Wait, this is useful outside the browser!
Let's also make this composable
WASM
Modules
WASI
Modules
WIT
Components

FERMYON

Let’s start somewhere in the middle

WASI and WebAssembly modules


A long time ago
Somewhere in the middle
Today
New compilation target for the web (browser)
Wait, this is useful outside the browser!
Let’s also make this composable
WASM
Modules
+
WASI
Modules
+
WIT
Components

FERMYON

What is WASI?

The goal is to define a set of portable, modular, runtime-independent, and WebAssembly-native APIs which can be used by WebAssembly code to interact with the outside world.

WASI 0.2.0 defines a set of APIs for common scenarios:

CLI, HTTP with a set of supporting interfaces: Sockets, I/O, Random, Clock, Filesystem access.

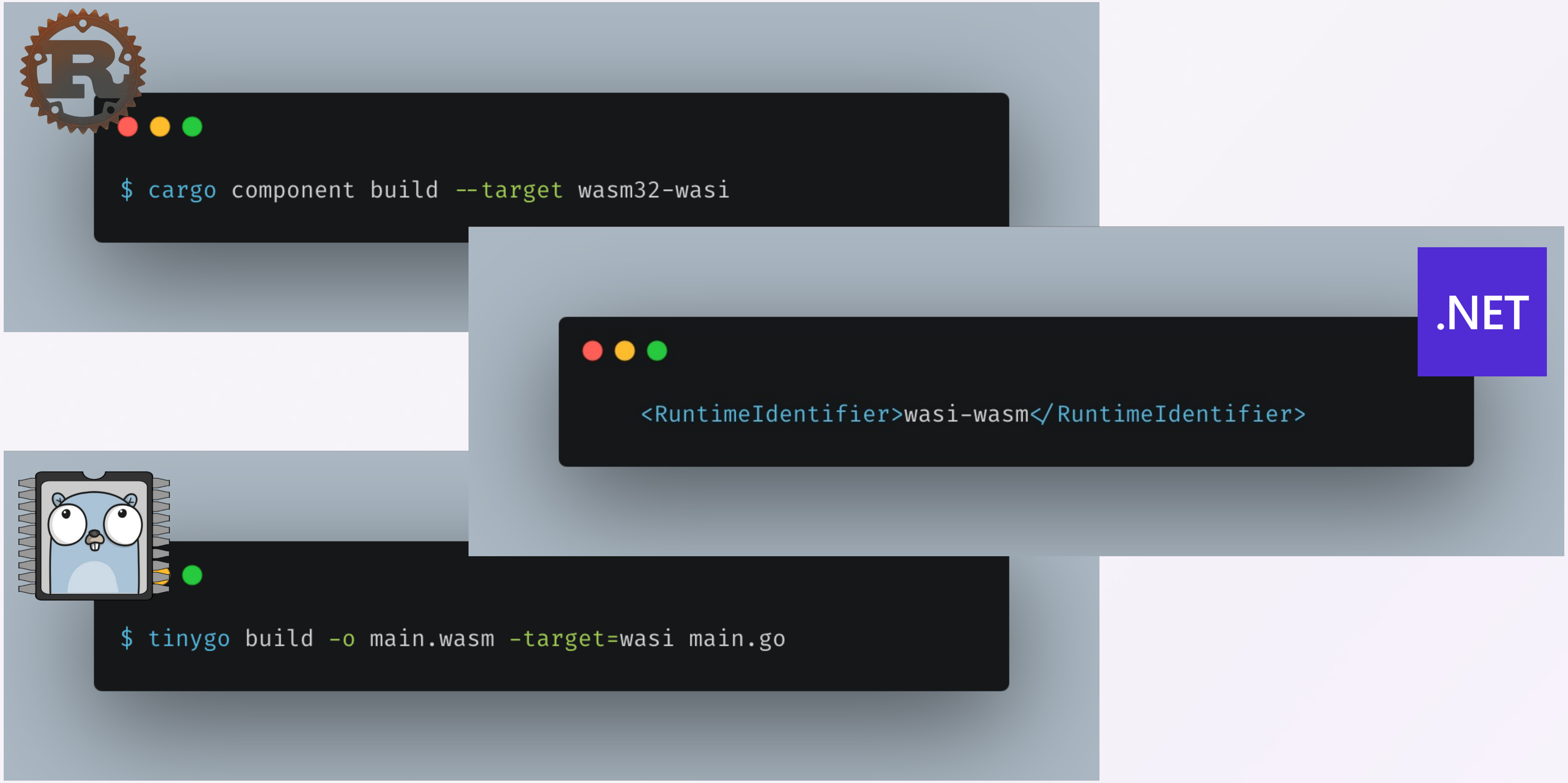
What Do We Need?

A programming language, which can compile to Wasm (WASI)

- https://developer.fermyon.com/wasm-languages/webassembly-language-support

A Wasm and WASI 0.2.0 compliant runtime

- https://wasmtime.dev/



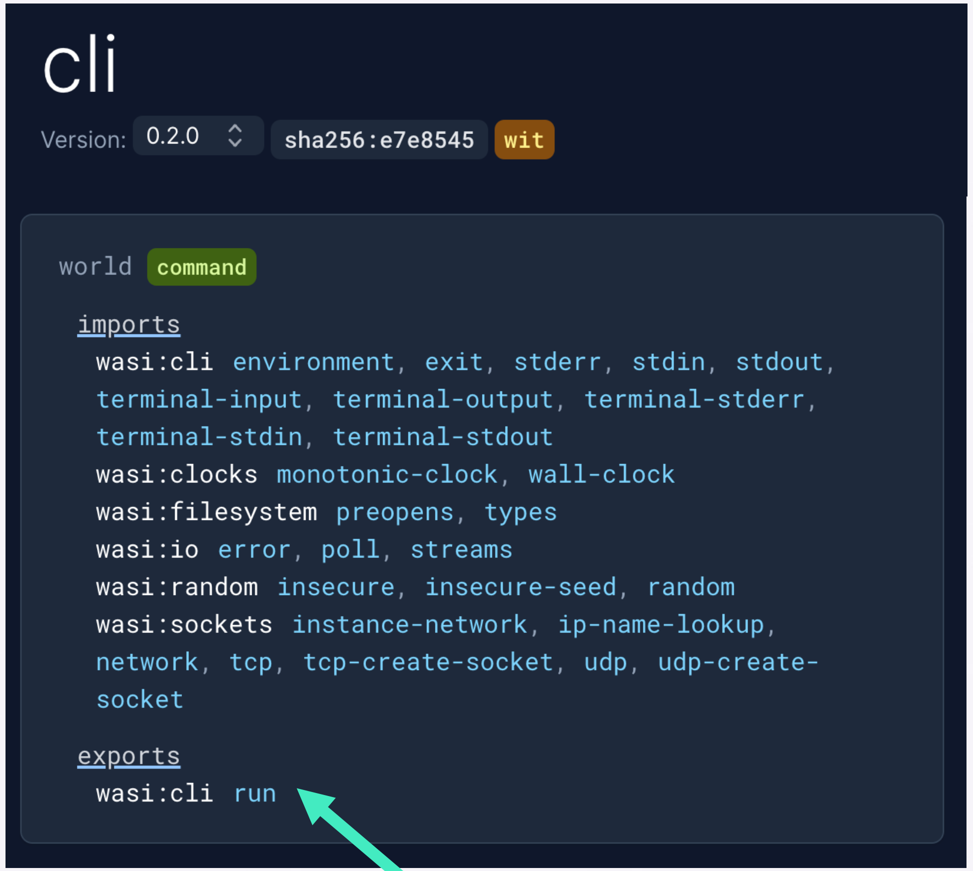
```
$ cargo component build --target wasm32-wasi
```

```
    <RuntimeIdentifier>wasi-wasm</RuntimeIdentifier>
```

```
$ tinygo build -o main.wasm -target=wasi main.go
```

FERMYON

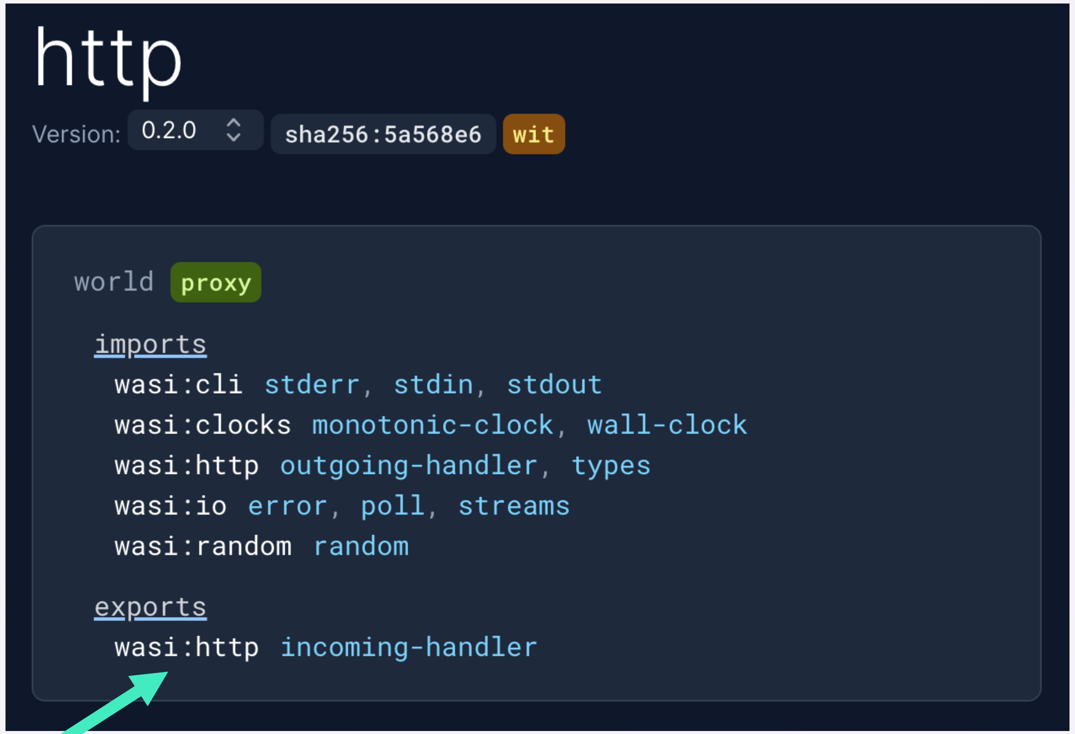
WASI 0.2.0 Use Cases – CLI and HTTP



cli

Version: 0.2.0 sha256:e7e8545 wit

```
world command

  imports
    wasi:cli environment, exit, stderr, stdin, stdout,
    terminal-input, terminal-output, terminal-stderr,
    terminal-stdin, terminal-stdout
    wasi:clocks monotonic-clock, wall-clock
    wasi:filesystem preopens, types
    wasi:io error, poll, streams
    wasi:random insecure, insecure-seed, random
    wasi:sockets instance-network, ip-name-lookup,
    network, tcp, tcp-create-socket, udp, udp-create-
    socket

  exports
    wasi:cli run
```



http

Version: 0.2.0 sha256:5a568e6 wit

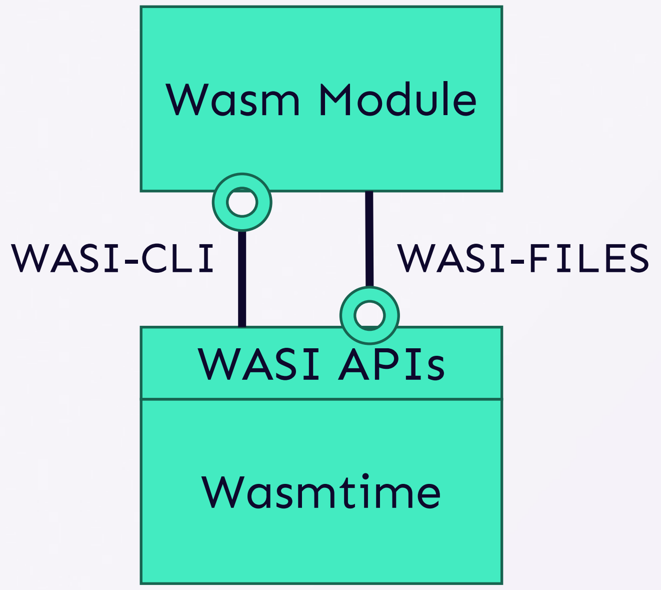
```
world proxy

  imports
    wasi:cli stderr, stdin, stdout
    wasi:clocks monotonic-clock, wall-clock
    wasi:http outgoing-handler, types
    wasi:io error, poll, streams
    wasi:random random

  exports
    wasi:http incoming-handler
```

Standardized exports

FERMYON

DEMO

Building a CLI Wasm module

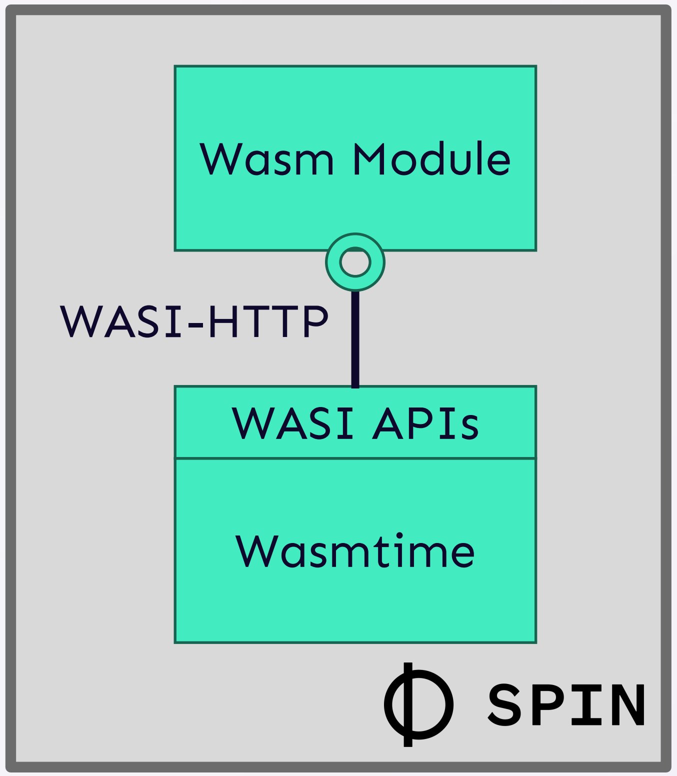


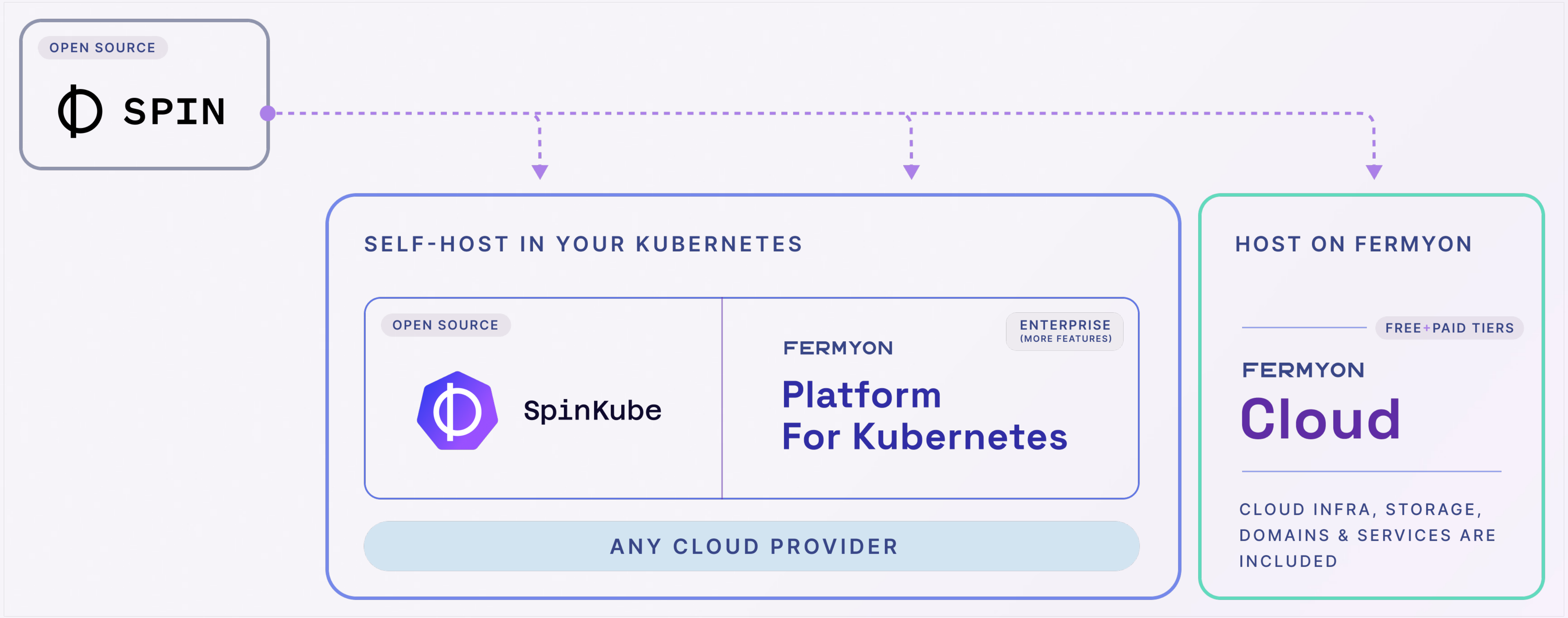
SPIN

The developer tool for building WebAssembly microservices and web applications

DEMO

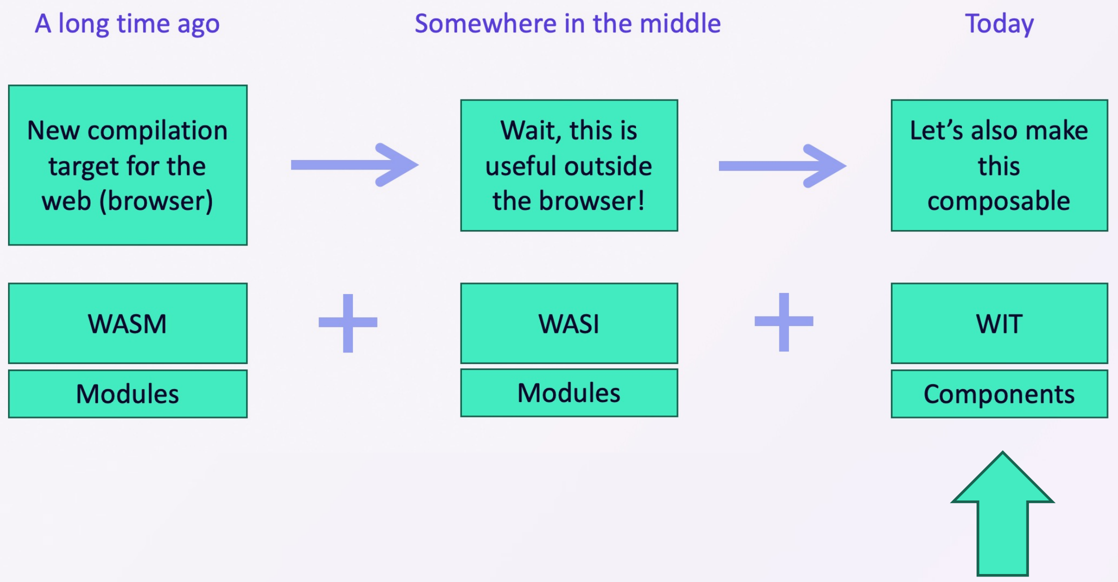
Building an HTTP handler using Spin




OPEN SOURCE
SPIN
SELF-HOST IN YOUR KUBERNETES
OPEN SOURCE
SpinKube
ENTERPRISE
(MORE FEATURES)
FERMYON
Platform For Kubernetes
ANY CLOUD PROVIDER
HOST ON FERMYON
FREE+PAID TIERS
FERMYON
Cloud
CLOUD INFRA, STORAGE, DOMAINS & SERVICES ARE INCLUDED

The Component Model

Arriving at the now…



A long time ago
Somewhere in the middle
Today
New compilation target for the web (browser)
Wait, this is useful outside the browser!
Let's also make this composable
WASM
Modules
+
WASI
Modules
+
WIT
Components

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 cross-language composition.

What is the WebAssembly Component Model?

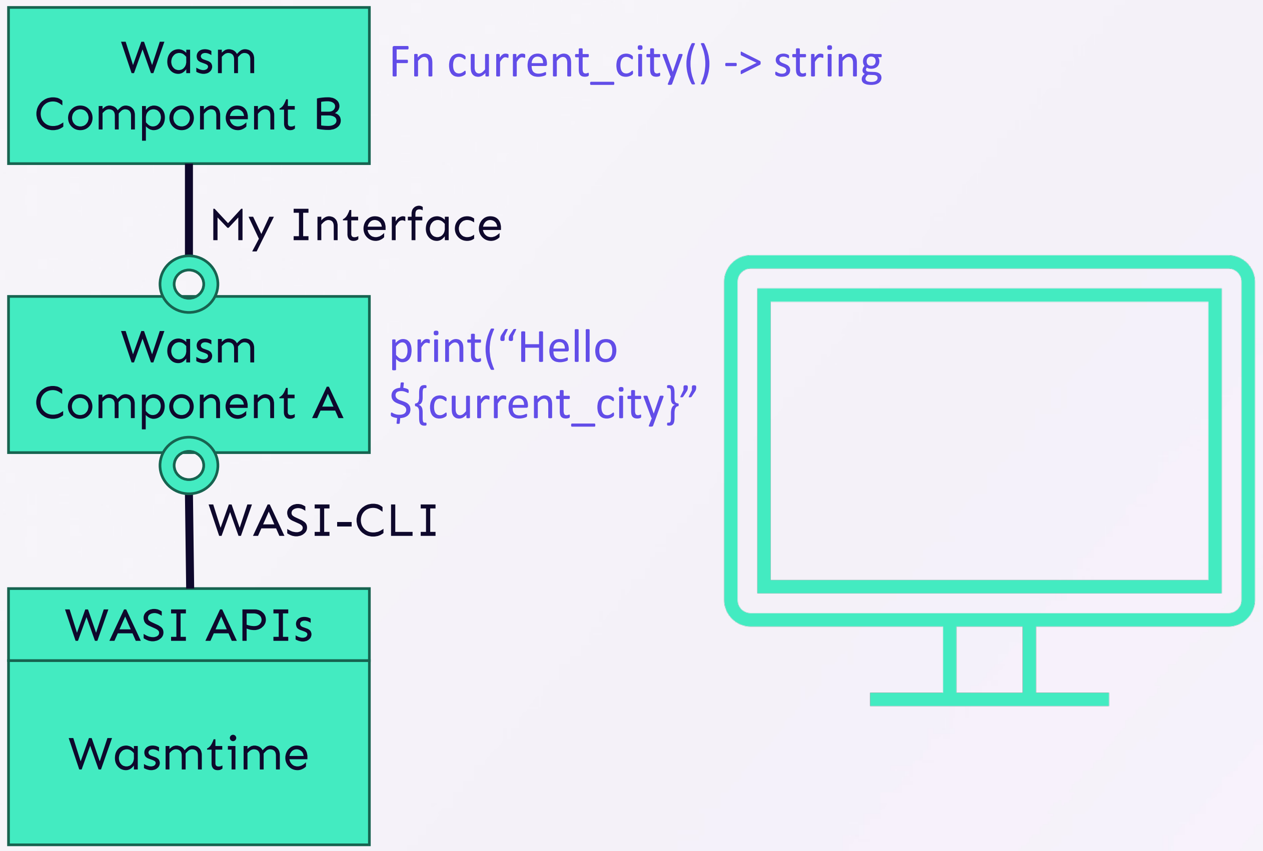
Provides a rich type system (compared to Wasm)

Defines the WebAssembly Interface Type language (WIT) to define contracts (interfaces and worlds)

Enables us to compose applications and functionality across programming languages

DEMO

Composing two components together



SPIN

Spin ject, built with open standards like W oAssembly Component Model.

AT A GLANCE:

Serverless AI NEW

Quickly test and run inferencing workloads with LLMs.

Key/

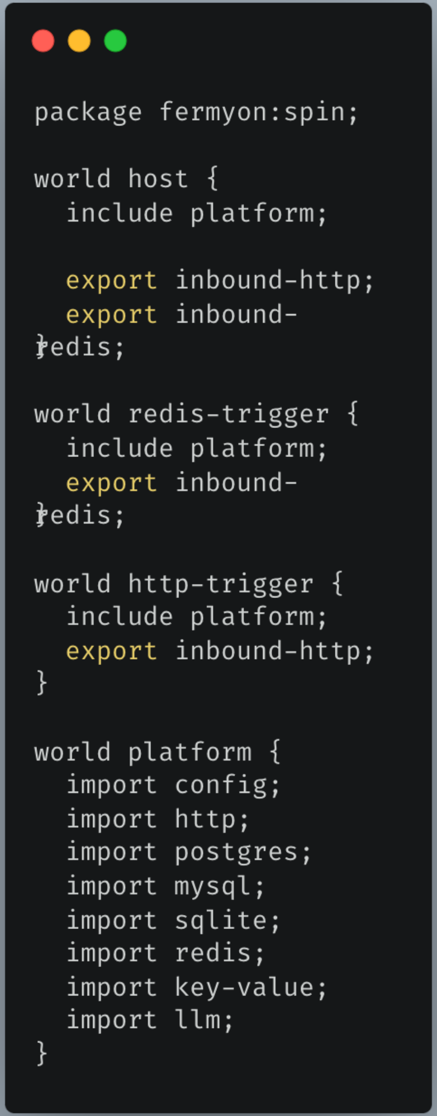
Easily apps v

Powerful CLI

Easy to create, run and deploy projects - in as little as 66 seconds.

SQLi

Add S with a SQLlit



```
package fermyon:spin;

world host {
  include platform;

  export inbound-http;
  export inbound-
}redis;

world redis-trigger {
  include platform;
  export inbound-
}redis;

world http-trigger {
  include platform;
  export inbound-http;
}

world platform {
  import config;
  import http;
  import postgres;
  import mysql;
  import sqlite;
  import redis;
  import key-value;
  import llm;
}
```

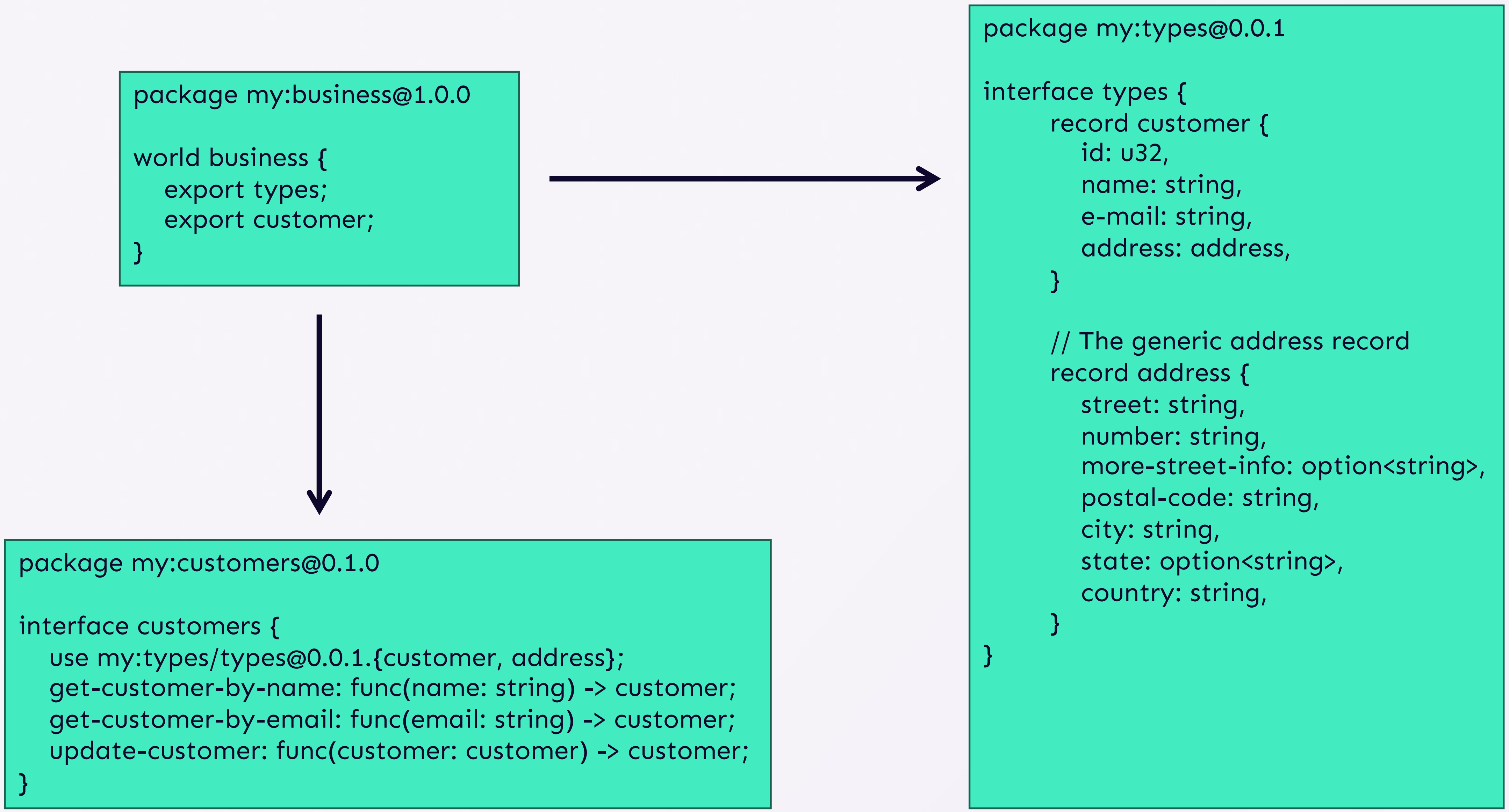
COMPOSING APPS:

- HTTP & Redis Triggers
- Relational Database Support
- Variables & Secrets Rotation

DEV EXPERIENCE:

- Supports almost any programming language
- Easy to debug with included helper commands

FERMYON



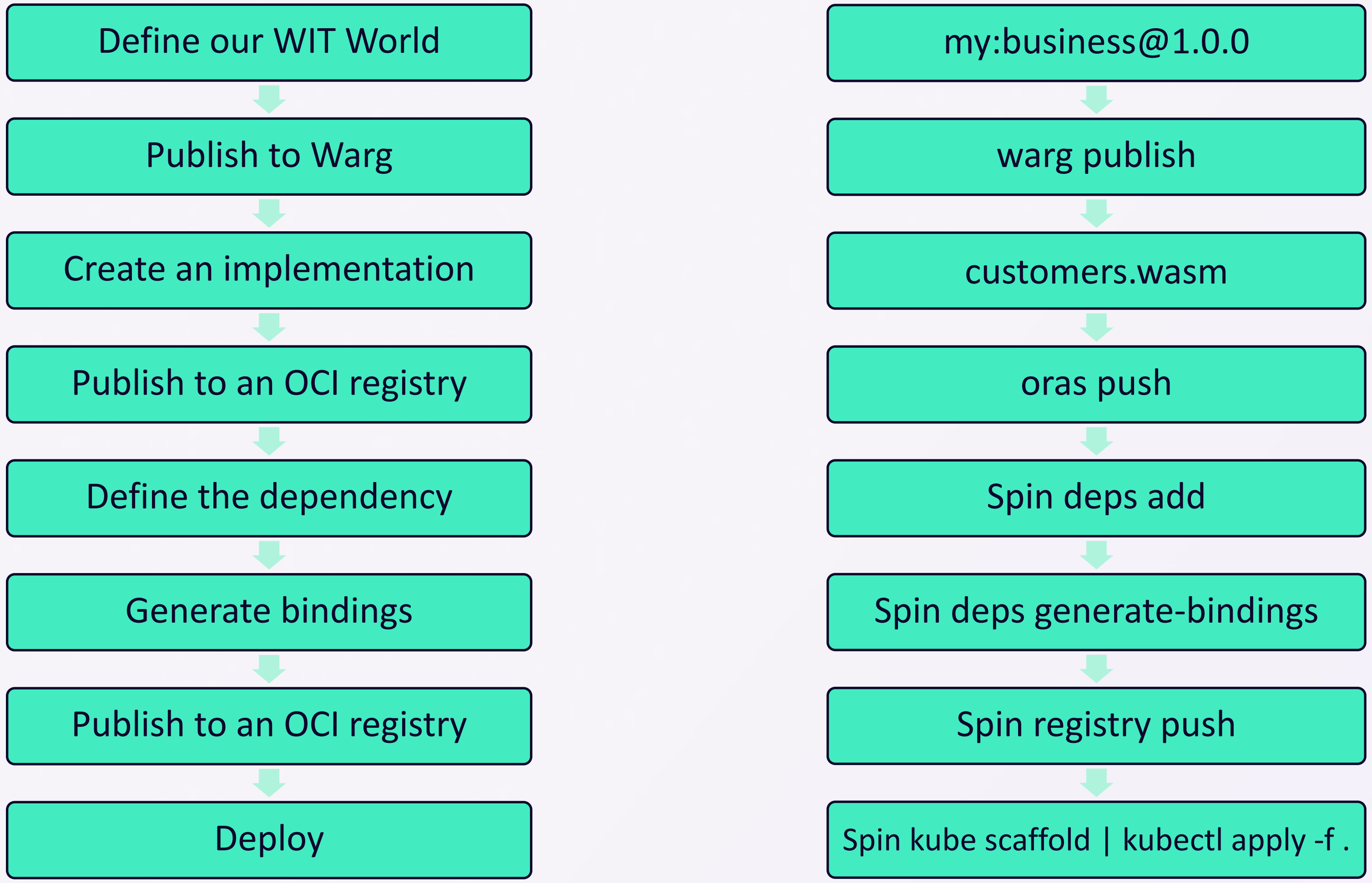
```
package my:business@1.0.0

world business {
    export types;
    export customer;
}
```

```
package my:types@0.0.1

interface types {
        record customer {
            id: u32,
            name: string,
            e-mail: string,
            address: address,
        }

        // The generic address record
        record address {
            street: string,
            number: string,
            more-street-info: option<string>,
            postal-code: string,
            city: string,
            state: option<string>,
            country: string,
        }
}
```

```
package my:customers@0.1.0

interface customers {
    use my:types/types@0.0.1.{customer, address};
    get-customer-by-name: func(name: string) -> customer;
    get-customer-by-email: func(email: string) -> customer;
    update-customer: func(customer: customer) -> customer;
}
```

FERMYON

End-to-end workflow



Define our WIT World	my:business@1.0.0
Publish to Warg	warg publish
Create an implementation	customers.wasm
Publish to an OCI registry	oras push
Define the dependency	Spin deps add
Generate bindings	Spin deps generate-bindings
Publish to an OCI registry	Spin registry push
Deploy	Spin kube scaffold | kubectl apply -f .

The useful reality

Standardized worlds for common scenarios:

- Command-line Interface: WASI-CLI
- Handle HTTP requests: WASI-HTTP
- Use a Key-Value store: WASI-KeyValue

Host implementations:

- Spin and SpinKube – Serverless framework
- Nginx Unit Web Server – Web Server!

Inspiration

The Onion (Separation of concerns)

- Transport, Logging, Metrics, Security, Compliance, Service Connectivity, etc. as outer layers

Comprehensive Compute Platforms (X-as-a-Component)

KeyValue Store-as-a-component

Data obfuscation-as-a-component

Hosting-as-a-component

Auth-as-a-component

Key Takeawaus

Wasm Co ... work that

allow inte ... ges using rich

contracts ... ructures are

represent

Wasm Co

We can co

Polyglot

Dynamic

Platforms

FERMYON

Thank you!

Search For:

WebAssembly
WASI
WebAssembly Component Model
Fermyon Spin
SpinKube
Spin Up Hub