F5 NGINX

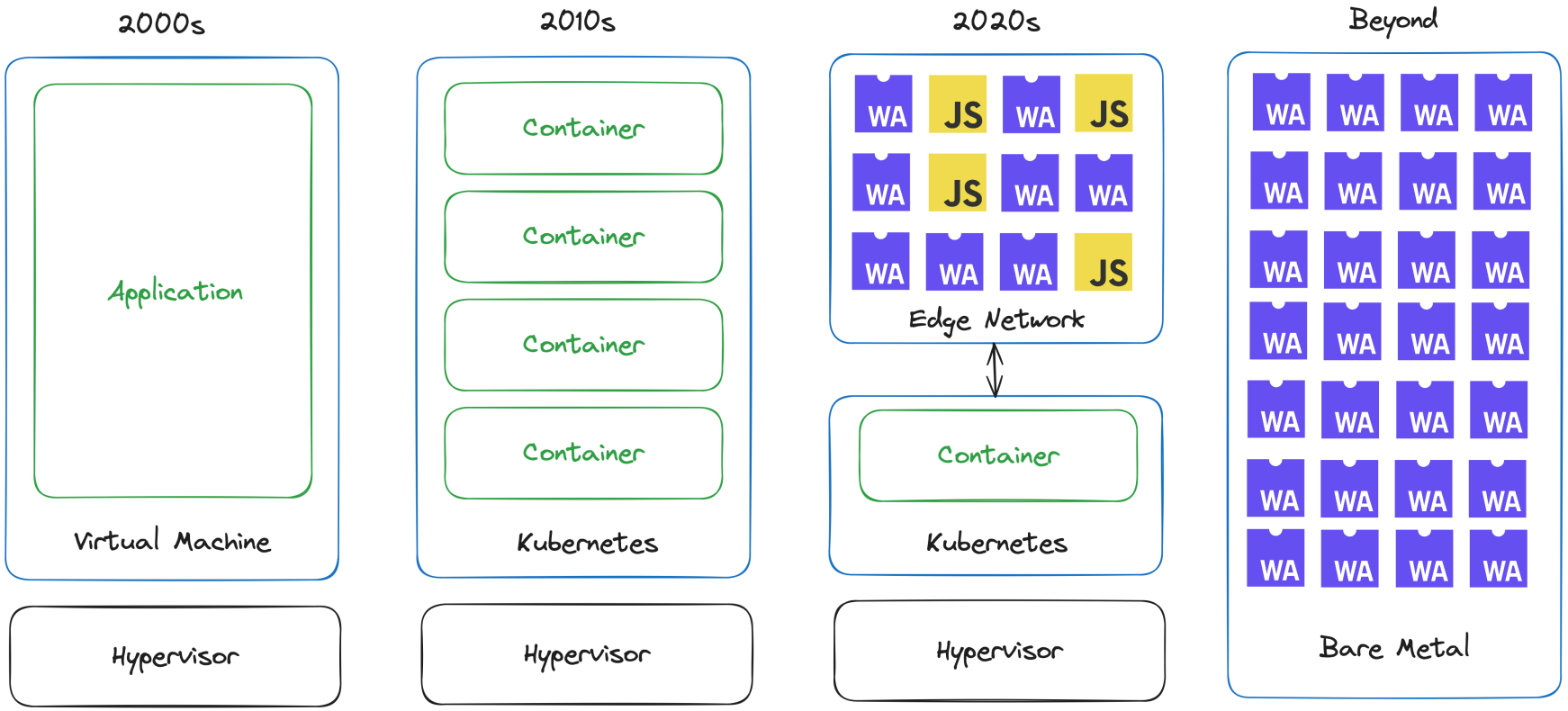
WebAssembly & NGINX

Serving Server Side Wasm with Web Awareness

Dave McAllister
Sr. Open Source Technologist

NGINX
Part of F5

A new wave of cloud computing



Wasm is poised to become the dominant technology powering cloud compute workloads – a $600B+ market – within 10 years.

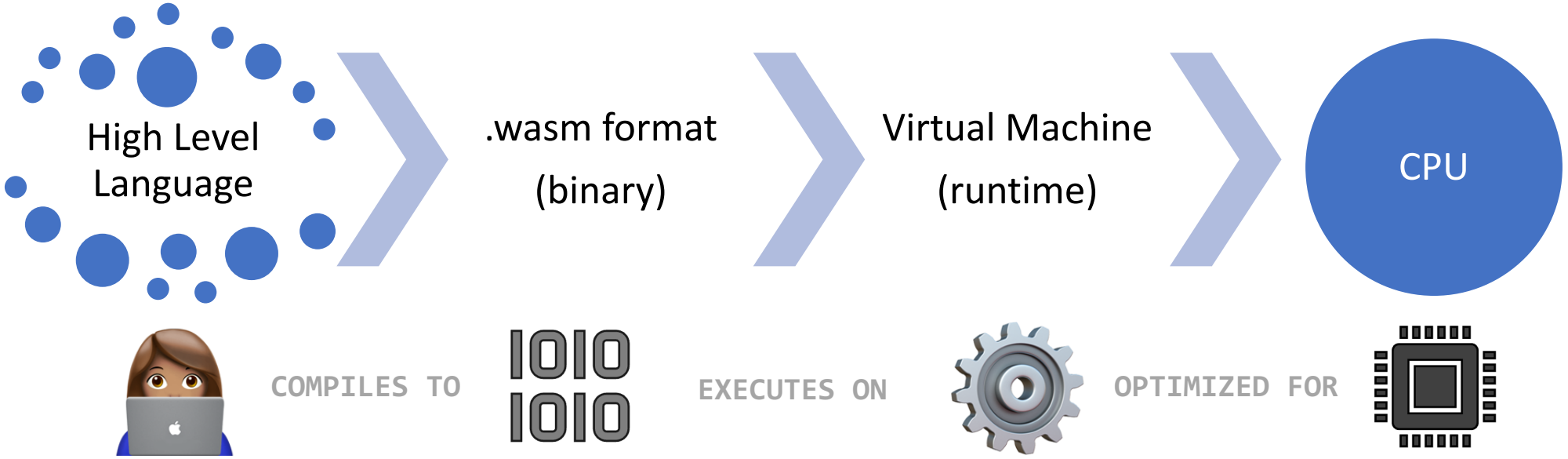
What is WebAssembly (Wasm)?

WebAssembly is an open standard that defines a **virtual CPU** and the instruction set that is executed there.

It enables execution of code within a **memory-isolated sandbox** that protects the host system.

You use WebAssembly every day.

WebAssembly beyond the browser



WEBASSEMBLY

Security
- Memory isolation
- Explicit host access

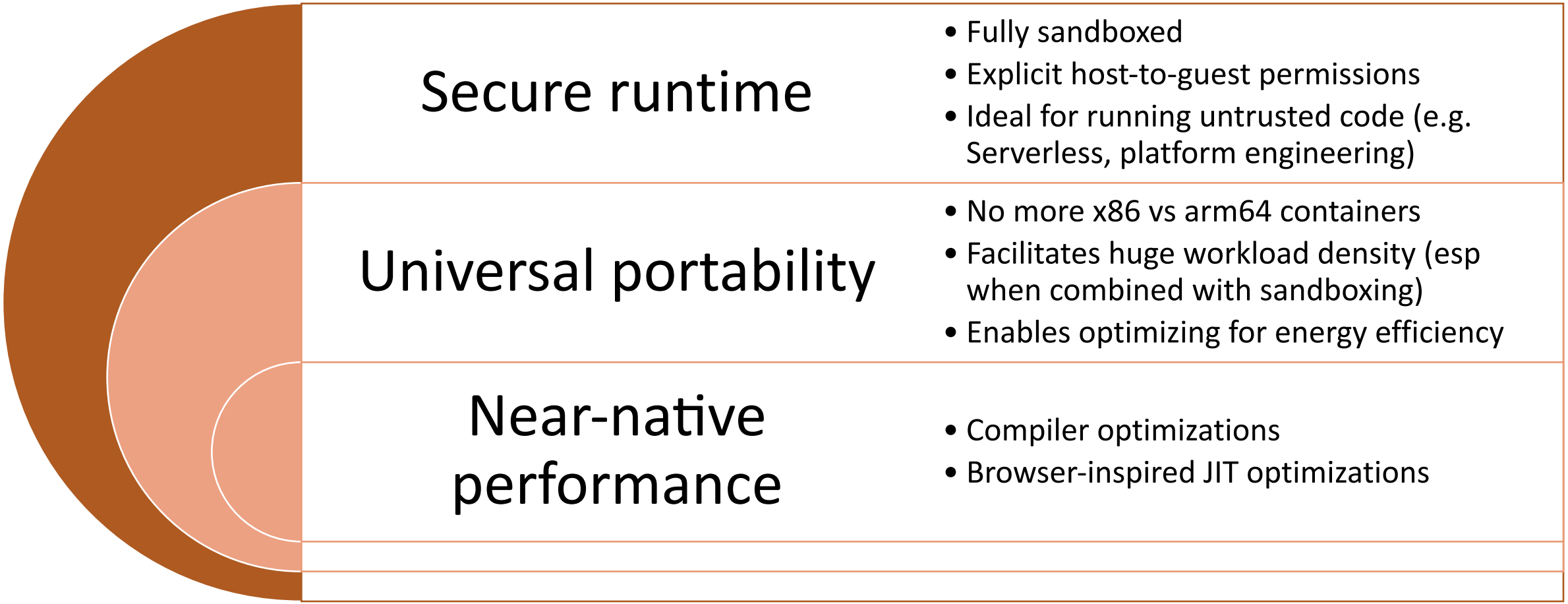
Performance
- Fast startup time
- Native execution speed

Portability
- Compile from any language
- Run on any platform

WASI

Interface to networks, filesystems, cryptography, etc.

WebAssembly key characteristics



Why use WebAssembly?

- WebAssembly code can be decoded and start executing within microseconds, even cold
- Superior levels of isolation can be achieved via Wasm's deny-by-default sandbox,
- Wasm is highly portable
- Nearly any compiled language and popular interpreted languages can compile to Wasm
- Depending on the source language, Wasm artifacts can be kilobytes or single digit megabytes
  - and will get much smaller thanks to module linking and the Component Model

WebAssembly Runtimes

- WebAssembly code is executed by a WebAssembly runtime
- Wasm code can be interpreted, just-in-time compiled (most common), or ahead-of-time compiled
- Popular general-purpose runtimes
  - V8, Wasmtime, Wasmer, WasmEdge, wazero, Wasm3, the WebAssembly Micro Runtime (WAMR), and others
- Certain runtimes are better-suited for certain use cases
- Special-purpose runtimes also exist

More on Sandboxing

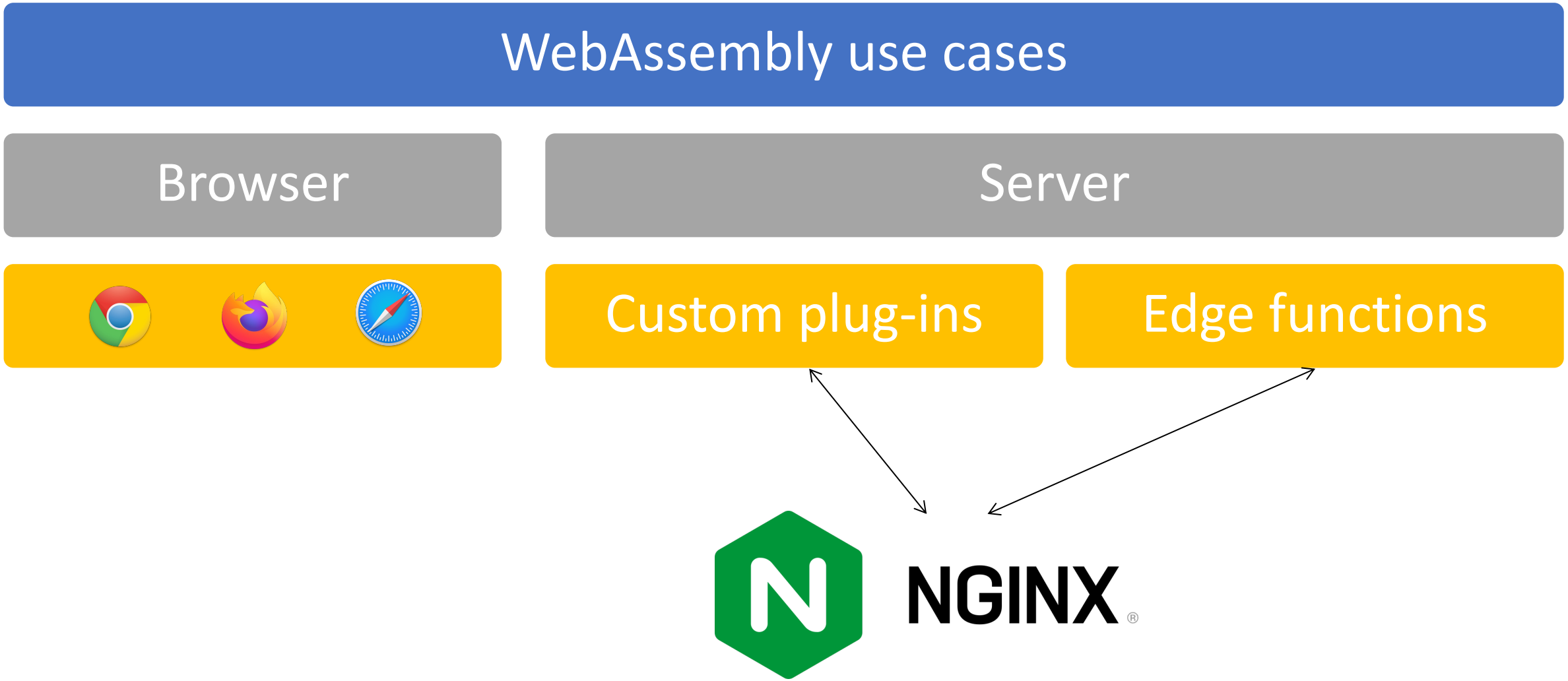
- Wasm modules can really only manipulate numbers and nothing else
- By default, a Wasm module can’t even know as much as the system time
- The **only** way a Wasm module can communicate with the outside world is via explicit host functions
- A host function is a utility provided by the host runtime, typically to access outside resources, such as networking or file IO
- Adding host functions that are unique to your runtime reduces the portability of modules that are compiled to target your runtime

WebAssembly Text (.wat)

Adder

```
(module
  (func $add (param $x i32) (param $y i32) (result i32)
    local.get $x
    local.get $y
    i32.add
  )
  (export “add” (func $add))
)
```

Project/Product opportunities



WebAssembly Plugins

- Because of Wasm's sandboxing properties, along with its near-native speed and quick loading/restart times, it's particularly well-suited for plugins
- Plugins can be written in any language that can target WebAssembly
- Though the code will typically be untrusted, we can securely run it and know its capabilities via static code introspection
- It's very easy to embed a WebAssembly runtime into any application

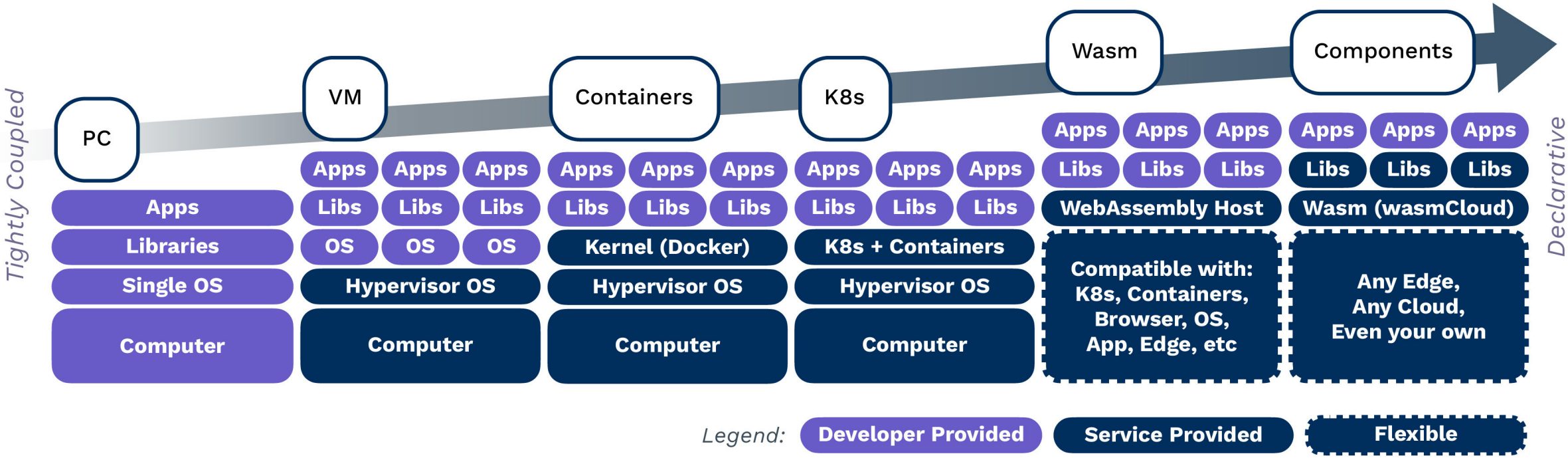
WASI

- WASI, the WebAssembly System Interface, is a standard set of host functions that allow Wasm to access the outside world that one can reasonably expect most runtimes to implement*
- WASI is versioned via preview versions. As of last week, the current version is Preview 2, with an expected Preview 3 before WASI 1.0
- Preview 1 included support for command line arguments, environment variables, system clocks, file IO, sockets, events, signals, and randomness
- Preview 2 refined these capabilities and introduced wasi-http, all built on top of the Component Model
- Preview 3 adds async and streams

The Component Model

- The Component Model defines high-level types for WebAssembly modules to enable module composability
- Each type has a specific memory-level representation, known as the Canonical ABI
- Modules will be composable, regardless of the source language, regardless of if the source language uses linear memory or Wasm GC
- This enables **cross-ecosystem library sharing**, which enables us to build full applications from components available from Rust, Python, Ruby, C, or any other language which can run in WebAssembly

The Component Model, cont.



https://cosmonic.com/product

Using WebAssembly

F5 NGINX

NGINX®

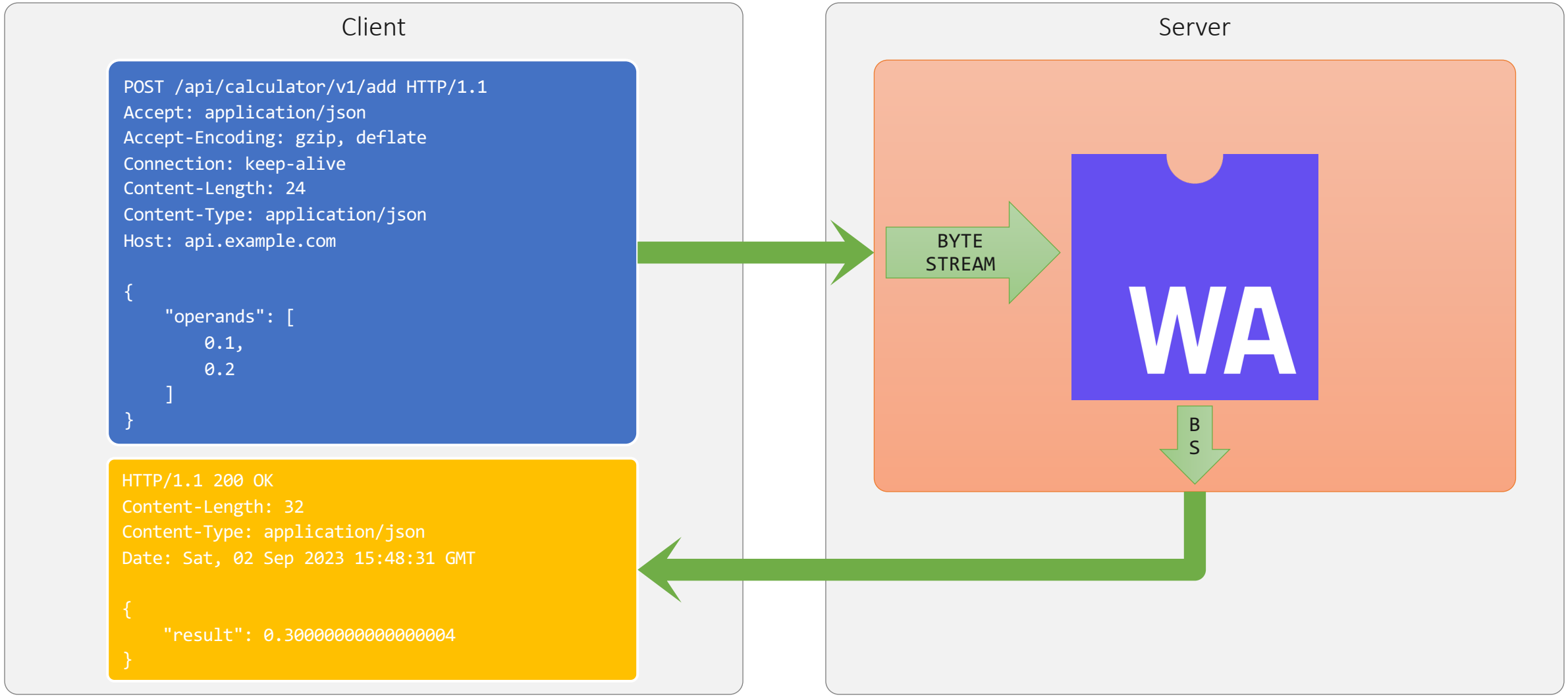
Part of F5

Unit®

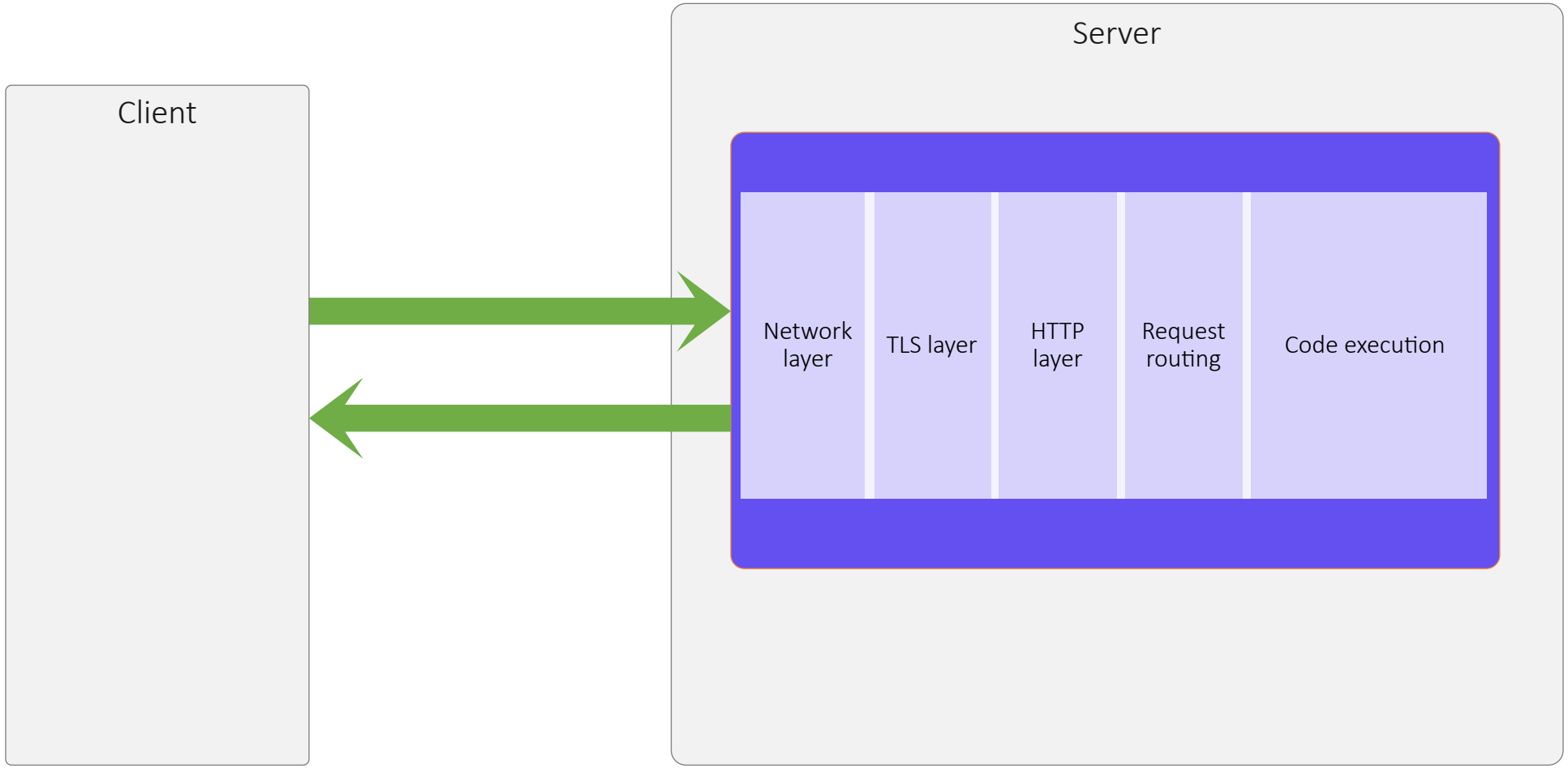
An NGINX Project

- We at NGINX are exploring all of the capabilities and flexibility that WebAssembly has to offer
- NGINX Unit is a universal application server—able to run both native application runtimes as well as WebAssembly-based applications

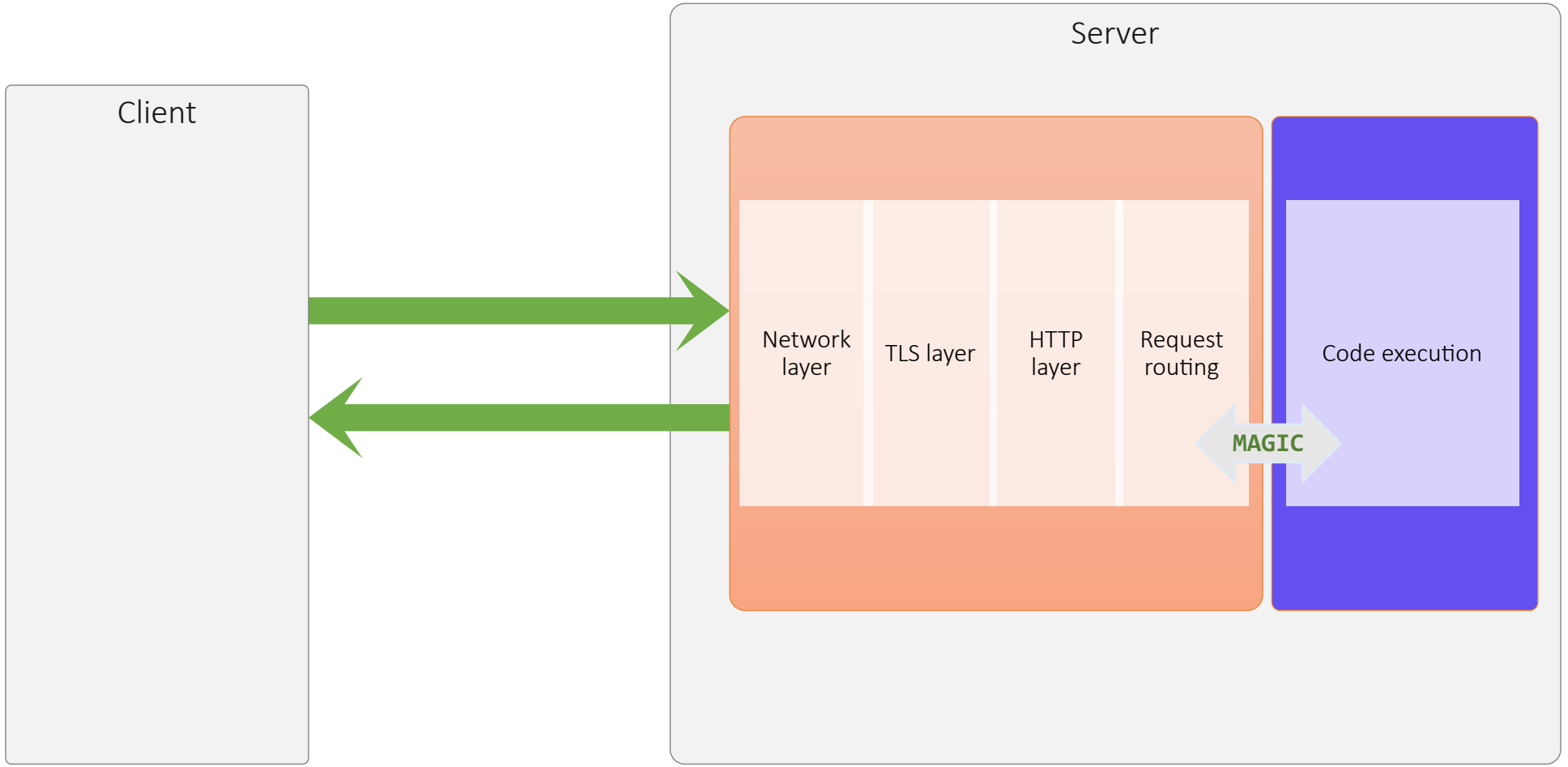
WebAssembly is a slam dunk for executing web applications



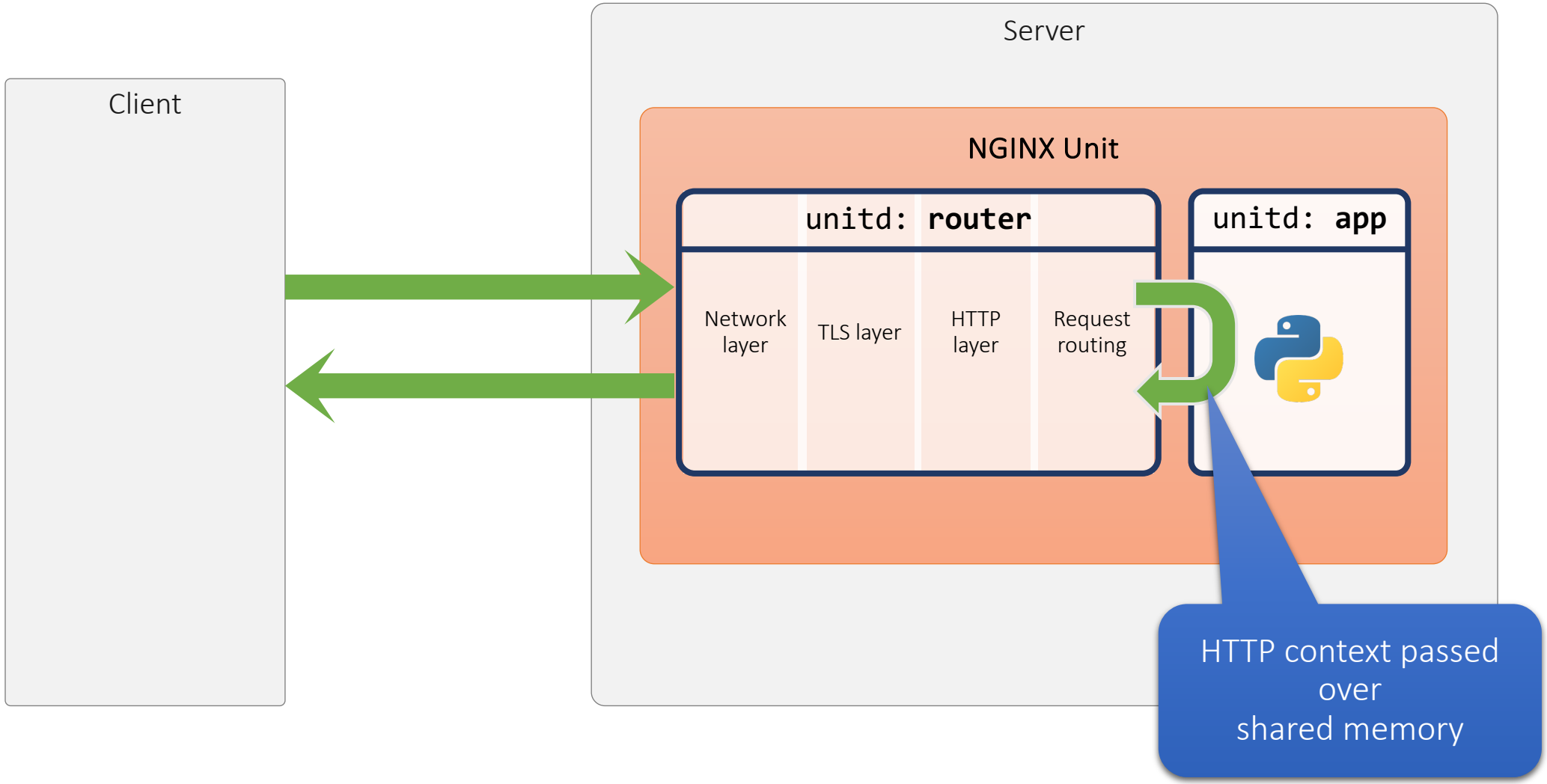
How to build a WebAssembly web app server?



Separating concerns FTW

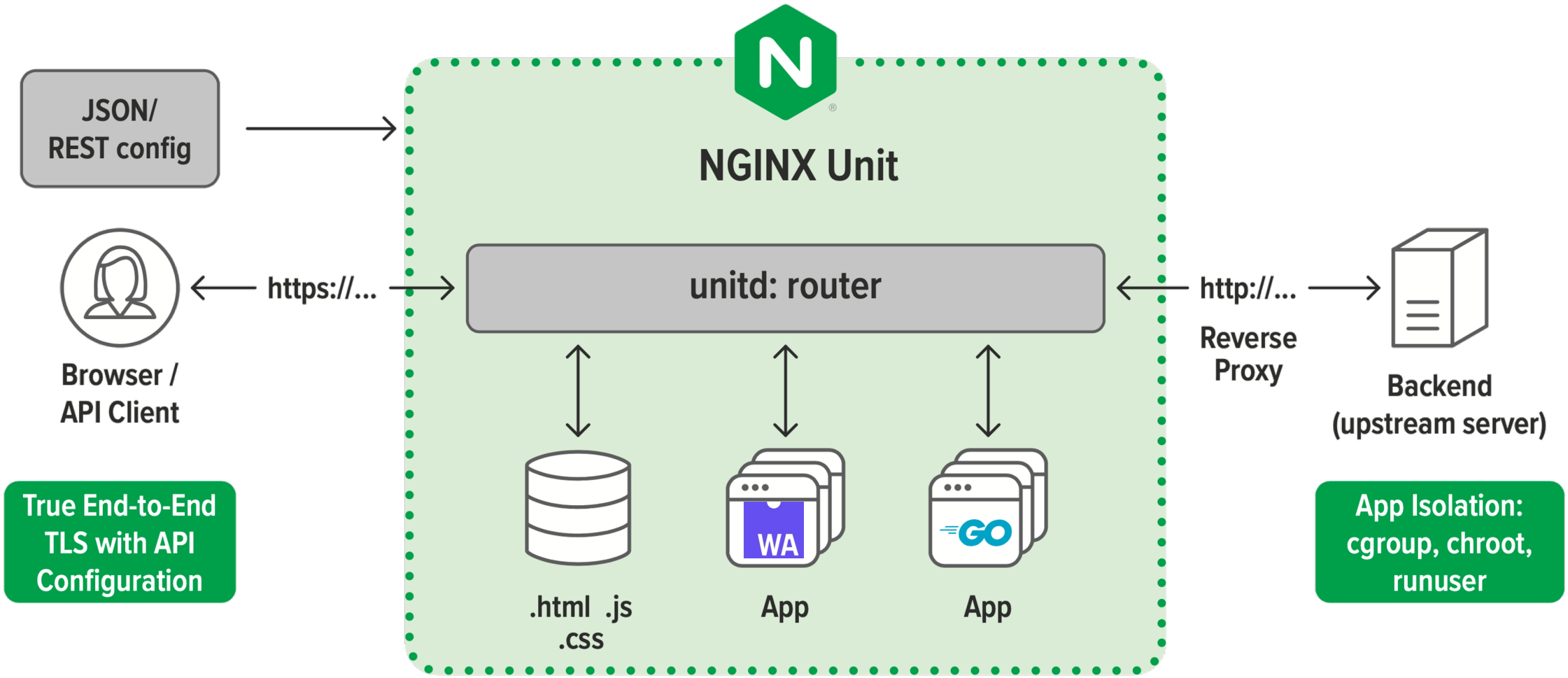


Introducing NGINX Unit

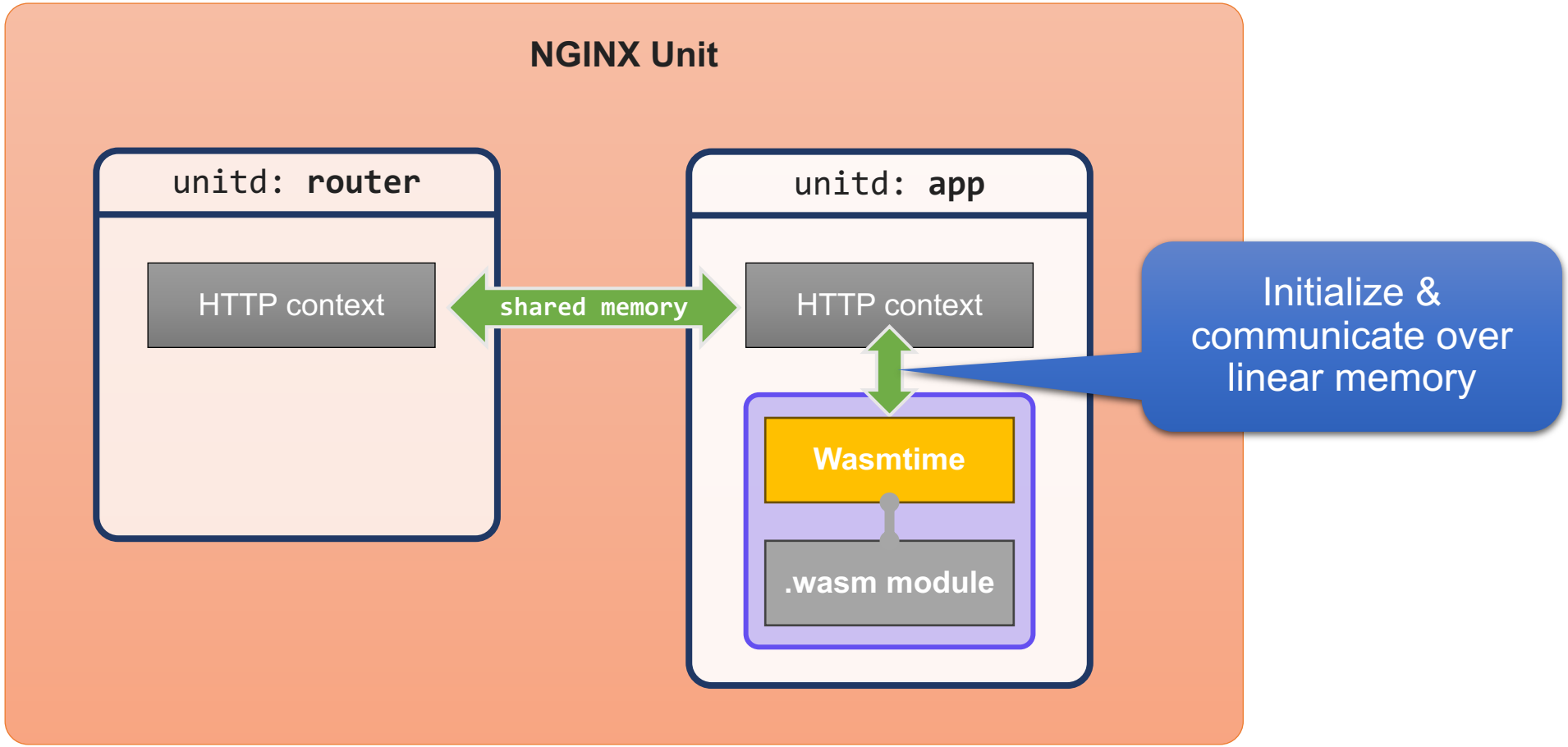


Understanding Web Languages

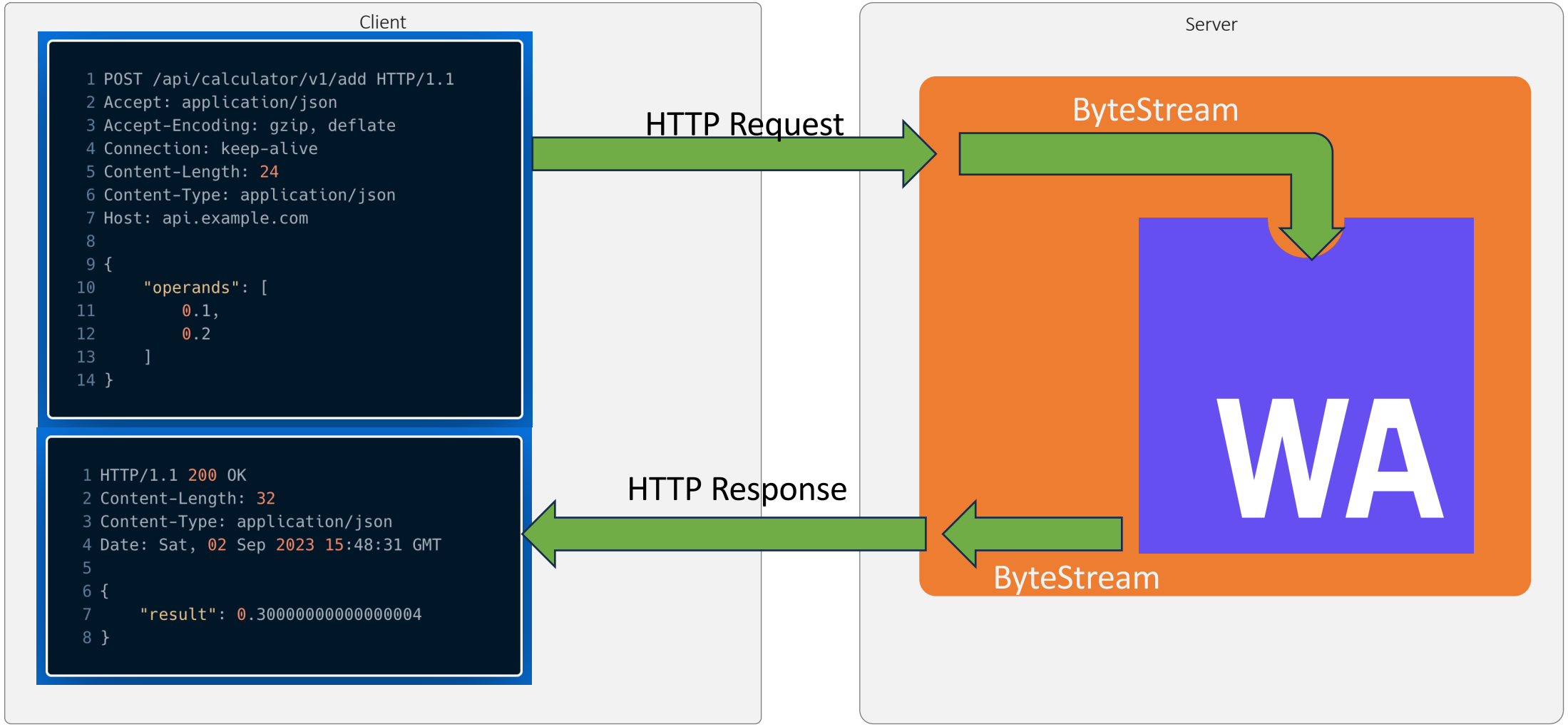
NGINX Unit System Architecture



WebAssembly in NGINX Unit



WebApplications using WebAssembly



Try it for yourself

- Two Repos to grab
  - https://github.com/nginx/unit
  - https://github.com/tippexs/wasm32-wasi-playground
- Quick install directions in each
  - On Mac: brew install nginx/unit/unit

This is for NGINX Unit 1.31.1. These two together will enable the WASI 0.2.0 preview

NGINX Unit 1.32.0 includes WASI Preview 2 – releases 27-February

F5 NGINX: WebAssembly

Involved with Worldwide Web Consortium (W3C) governs WebAssembly alongside HTML and CSS

BYTECODE ALLIANCE

Sponsors work and is active voting member for the Bytecode Alliance (where Wasm meets cloud)

Unit®
An NGINX Project

NGINX provides an open source implementation of server-side Wasm

Thanks for listening

NGINX Unit

- unit.nginx.org
- github.com/nginx/unit
- github.com/nginx/unit-wasm

Dave McAllister

- Linkedin : in/davemc
- Twitter : @dwmcallister

NGINX Unit
On GitHub