



mod_wasm

Bringing WebAssembly to Apache

Daniel López Ridruejo / Jesús González Martí
Office of the CTO | Wasm Labs @ VMware

<https://wasmlabs.dev>

About the Speakers

Wasm Labs | Office of the CTO | VMware



Daniel Lopez

- Sr. Director @ Wasm Labs (VMware)
- Bitnami CEO & Co-Founder (acquired by VMware)
- Apache Software Foundation - Emeritus Member

- Engineer @ Wasm Labs (VMware)
- Co-Founder & CTO at Smart Home startup
- AI/ML & NLP at Intel



Jesús González

Agenda

- What's WebAssembly?
- Introducing **mod_wasm**
- **mod_wasm** in action!
- Roadmap
- Q&A

What's WebAssembly? 🤔

What is WebAssembly (aka Wasm)

Neither web, nor assembly!

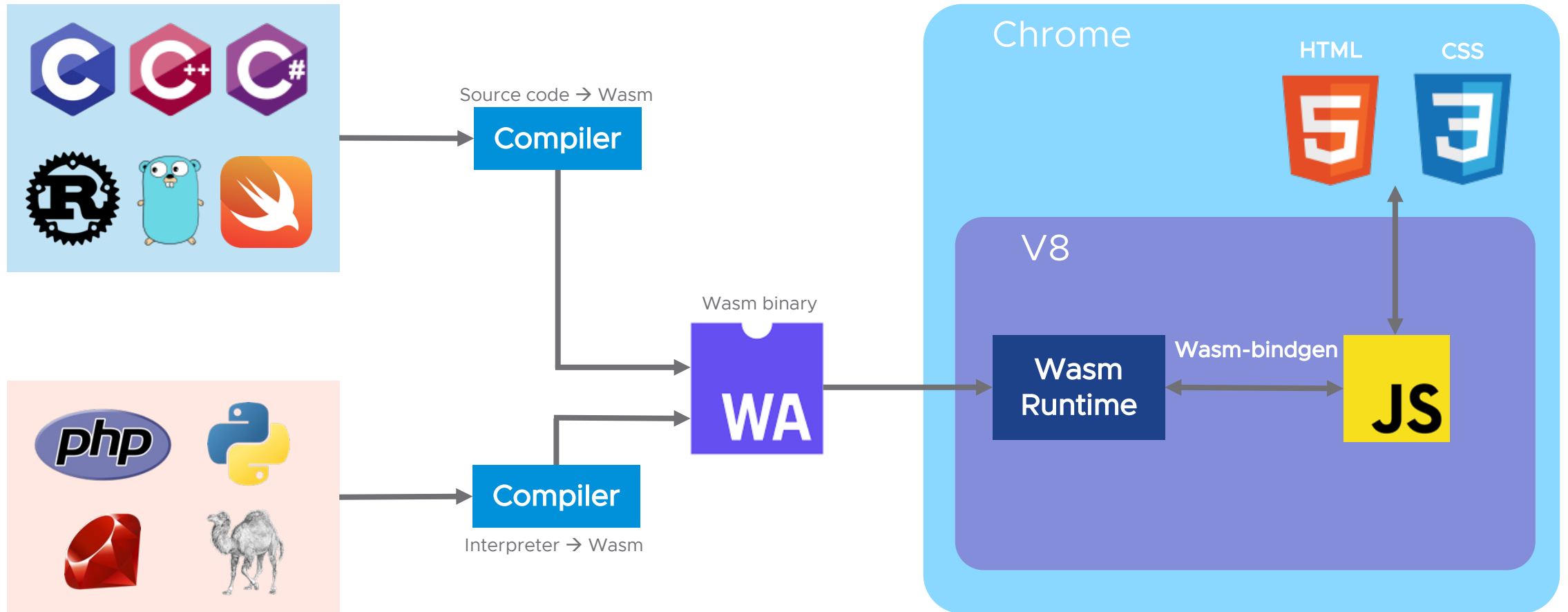


WebAssembly is an open standard that defines a portable binary format for executable programs.

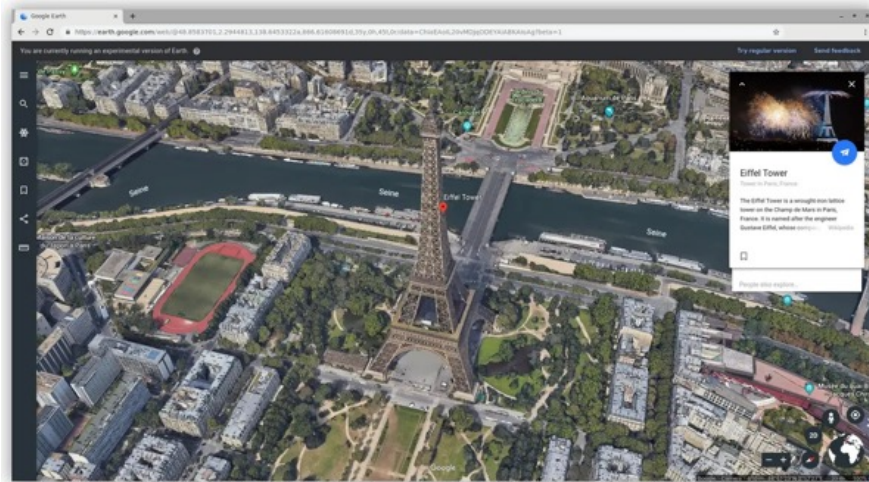


WebAssembly is native-supported in all major browsers

WebAssembly in the Browser



What you can do with WebAssembly in the browser...



WordPress entirely in the Browser

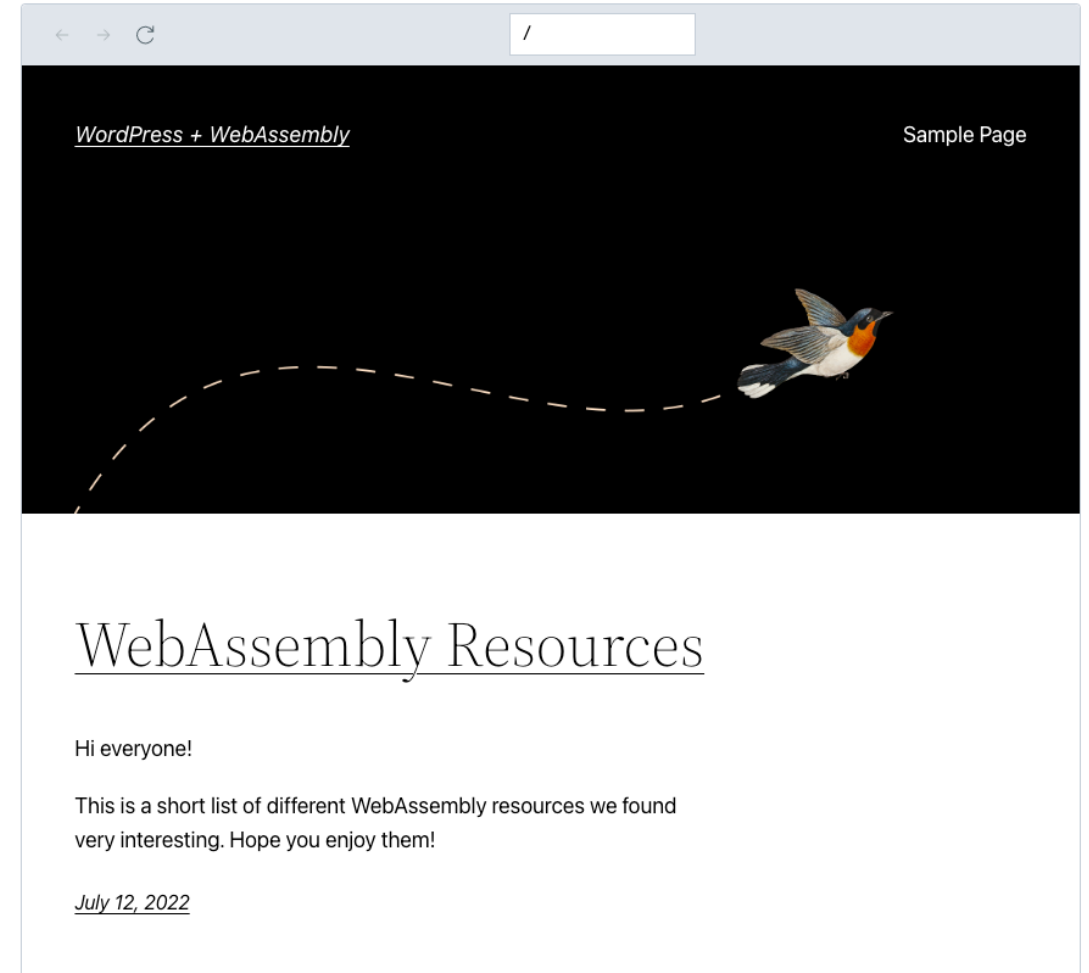
First work at WasmLabs

Running WordPress in the Browser

By  Jesús González At 2022 / 07 15 mins reading

<https://wordpress.wasmlabs.dev/>

- PHP interpreted ported to Wasm (including SQLite)
- Very useful for tutorials and instructions manuals



WebAssembly in the Server

An historic tweet from March 2019

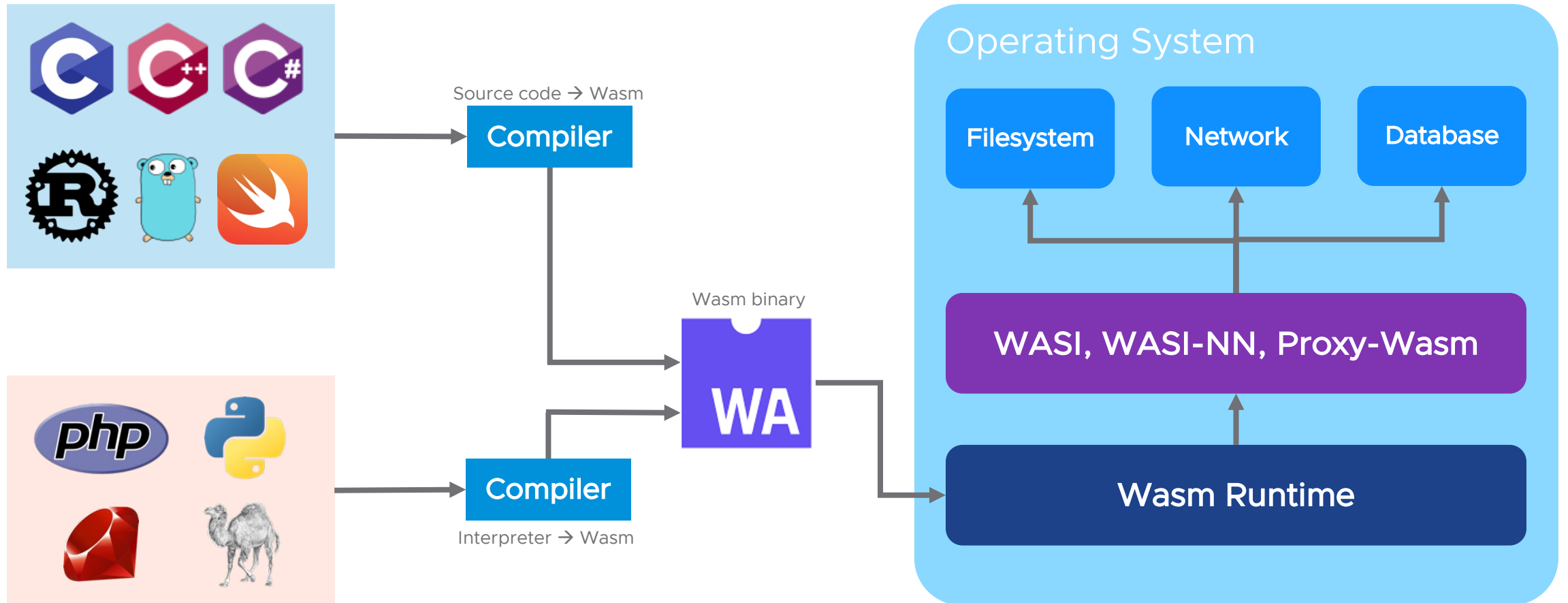


“If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing.”

Solomon Hykes, creator of Docker

WebAssembly in the Server

Wasm+WASI



WebAssembly Top Features

Open

Adopted by the entire industry

Fast

Native-like speed, JIT/AOT, no cold-starts

Secure

Memory safe, sandboxed, capabilities-based model, better supply chain security

Portable

Most CPUs (x86, ARM, RISC-V) , most OS including Android, ESXi, non-Posix

Efficient

Minimal memory footprint, CPU requirements

Polyglot

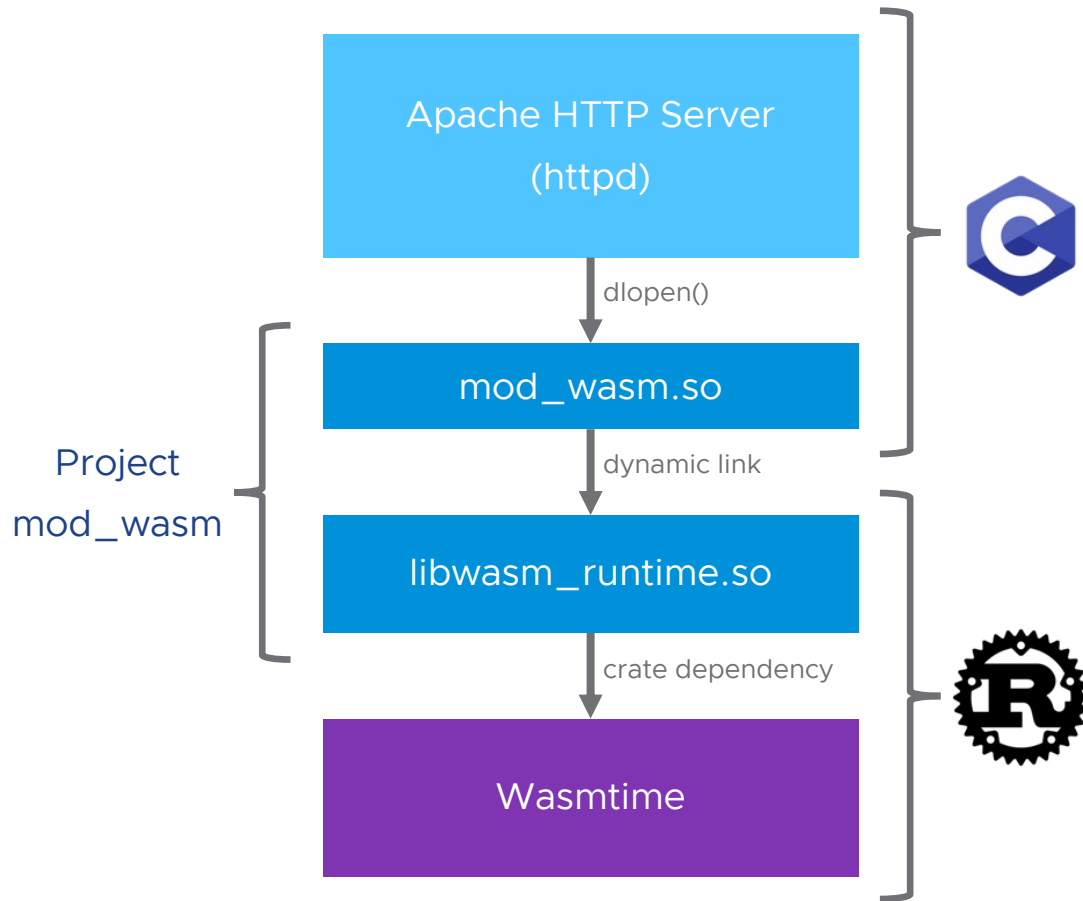
Support for 40+ languages, modern toolchains

In many respects, WebAssembly aims to fulfill Java's original promise, but with the hindsight of 20+ years and unanimous industry backing.

Introducing mod_wasm 🙌

Architecture Overview

mod_wasm.so + libwasm_runtime.so + Wasmtime



mod_wasm.so

- Apache extension module.
- New directives for **httpd.conf** to configure the Wasm context.
- Implements **post_config()** and **content_handler()** hooks.

libwasm_runtime.so

- Very **high-level** library for managing Wasm modules.
- It offers a **C-API** to mod_wasm.so.

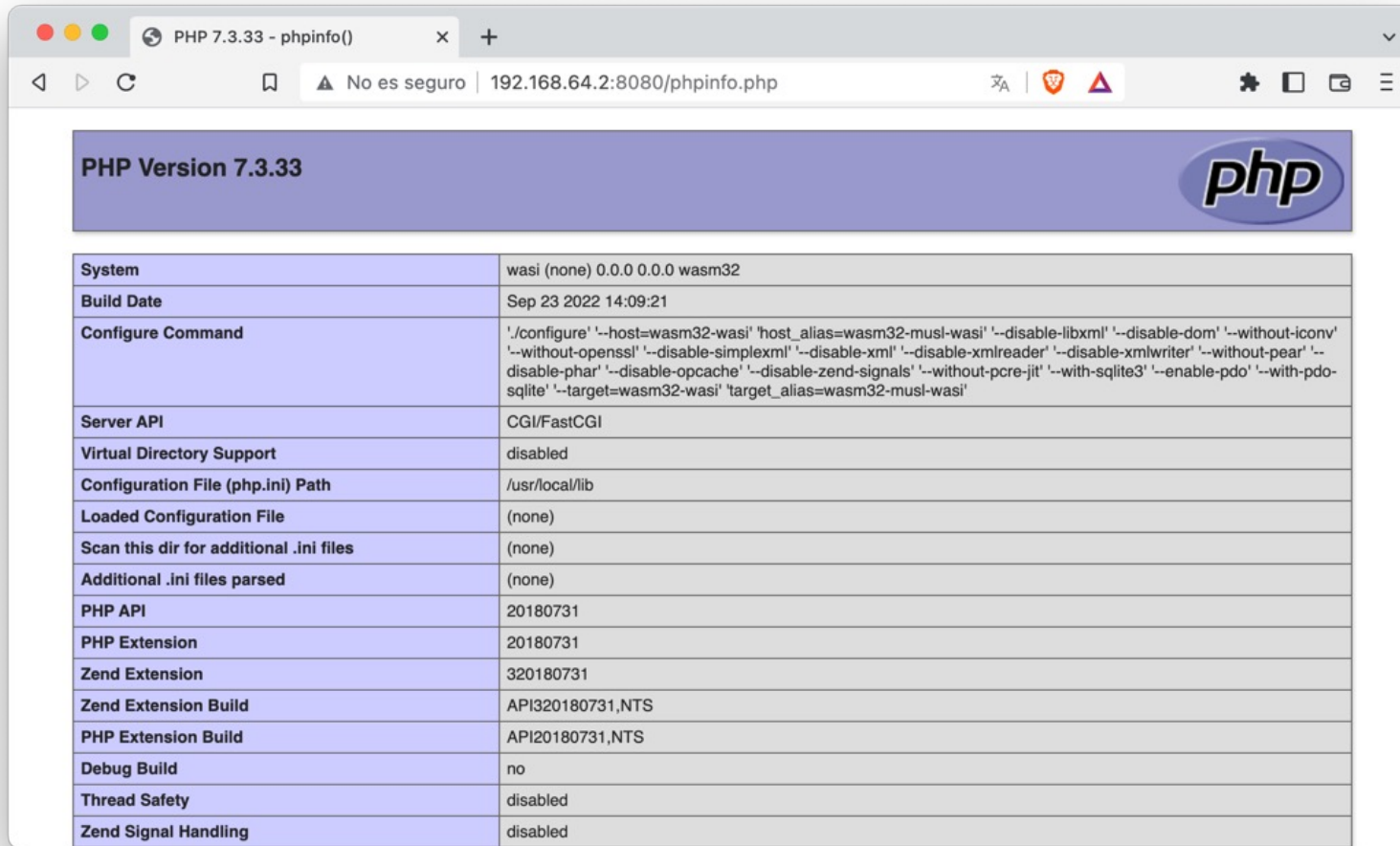
Wasmtime

- WebAssembly runtime from the **Bytecode Alliance**.

mod_wasm in action! 🎬

Demo #1: PHP running as Wasm in the Server

<http://192.168.64.2:8080/phpinfo.php>



PHP Version 7.3.33	
System	wasi (none) 0.0.0 0.0.0 wasm32
Build Date	Sep 23 2022 14:09:21
Configure Command	'./configure' '--host=wasm32-wasi' 'host_alias=wasm32-musl-wasi' '--disable-libxml' '--disable-dom' '--without-iconv' '--without-openssl' '--disable-simplexml' '--disable-xml' '--disable-xmlreader' '--disable-xmlwriter' '--without-pear' '--disable-phar' '--disable-opcache' '--disable-zend-signals' '--without-pcre-jit' '--with-sqlite3' '--enable-pdo' '--with-pdo-sqlite' '--target=wasm32-wasi' 'target_alias=wasm32-musl-wasi'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,NTS
PHP Extension Build	API20180731,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled

- Note **System** is **wasi** on **wasm32**.
- Environment variables

Demo #2: PrettyFy WebApp

CGI Vs. WebAssembly



“PrettyFy” is a one-script, Python-based WebApp:

- Reads contents from a previously uploaded file
- Outputs a full color prettified code in HTML

Two running environments:

1) CGI

- The **prettyfy.py** script is served as CGI.
- Python interpreter installed in the OS.

2) WebAssembly

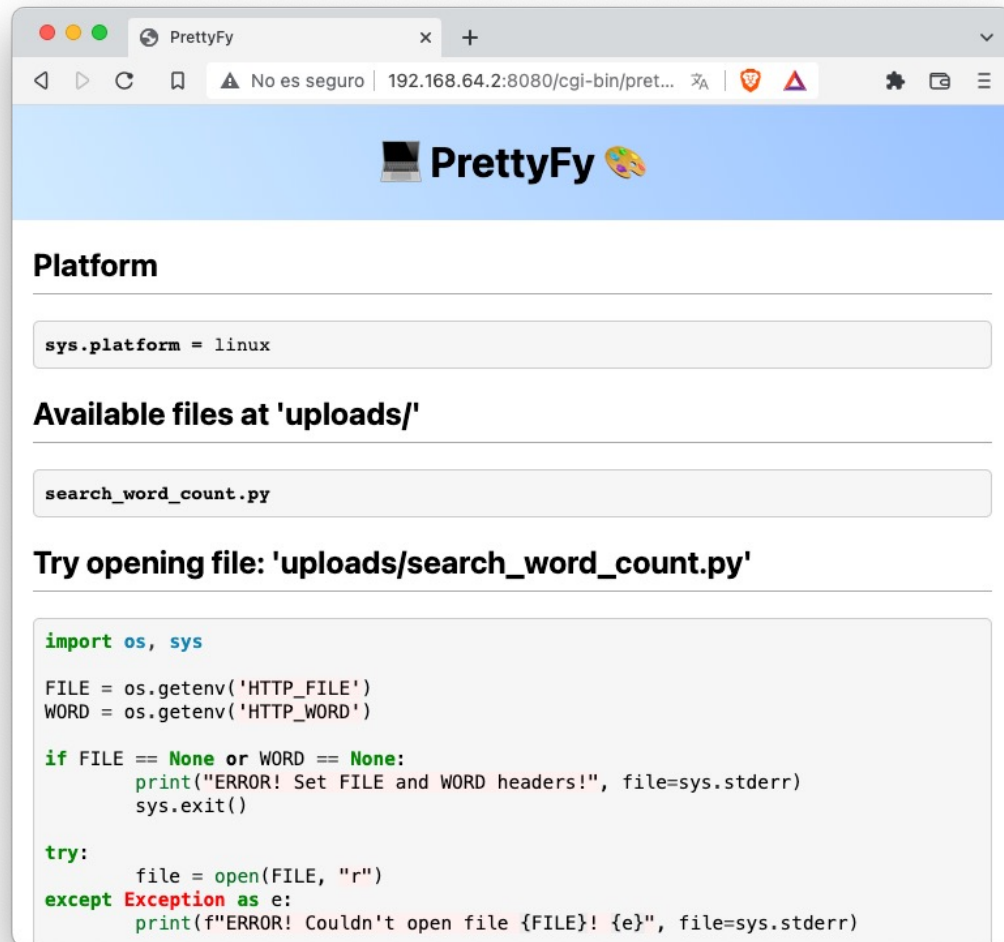
- It will use the same unmodified Python script.
- The Python interpreter will be compiled to Wasm and executed within a Wasm engine.

*In both scenarios, HTTP request headers and URL parameters will be passed as environment variables.

Demo #2: PrettyFy WebApp

CGI

http://192.168.64.2:8080/cgi-bin/prettyfy.py?file=search_word_count.py



```
import os, sys

FILE = os.getenv('HTTP_FILE')
WORD = os.getenv('HTTP_WORD')

if FILE == None or WORD == None:
    print("ERROR! Set FILE and WORD headers!", file=sys.stderr)
    sys.exit()

try:
    file = open(FILE, "r")
except Exception as e:
    print(f"ERROR! Couldn't open file {FILE}! {e}", file=sys.stderr)
```

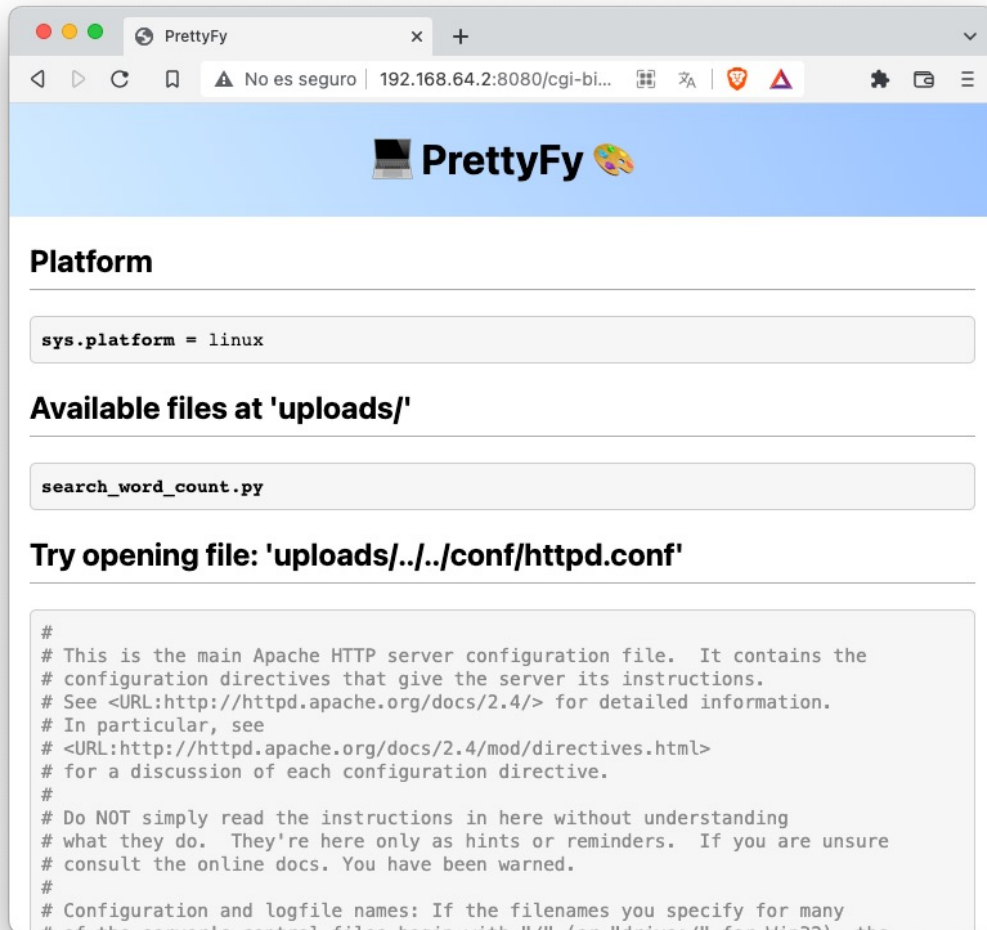
- Apache, CGI and Python are working as expected.
- Note Python's `sys.platform` value is `linux`.

- What if we try a Path Traversal attack? 

Demo #2: PrettyFy WebApp

CGI + Path Traversal Attack

<http://192.168.64.2:8080/cgi-bin/prettyfy.py?file=../../conf/httpd.conf>



- A simple Path Traversal attack was successful 😱
- PrettyFy didn't meet OWASP guidelines 😓

CONCLUSION:

Do not trust 3rd party software!!

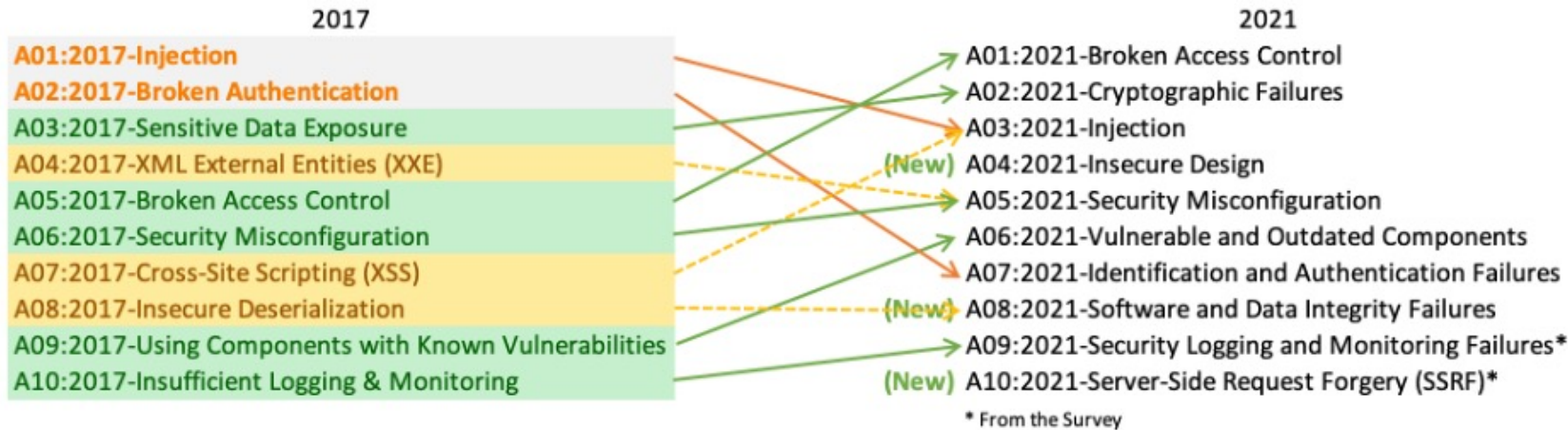
Demo #2: PrettyFy WebApp

OWASP 2021 - Top 10



<https://owasp.org/Top10>

#1 – Broken Access Control

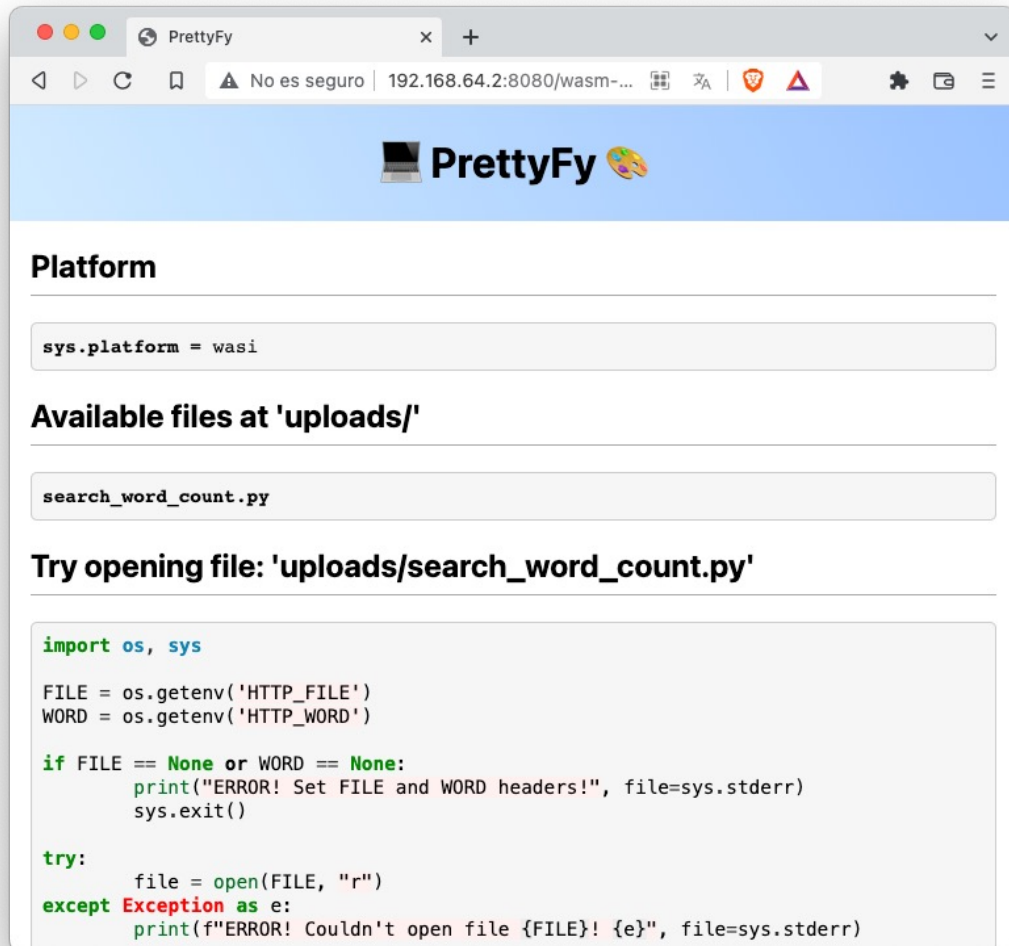


- Least privilege (deny by default)
- Bypassing access control checks by modifying URL or API requests
- Accessing API with missing access control
- Elevation of privilege
- ...

Demo #2: PrettyFy WebApp

Wasm

http://192.168.64.2:8080/wasm-module?file=search_word_count.py

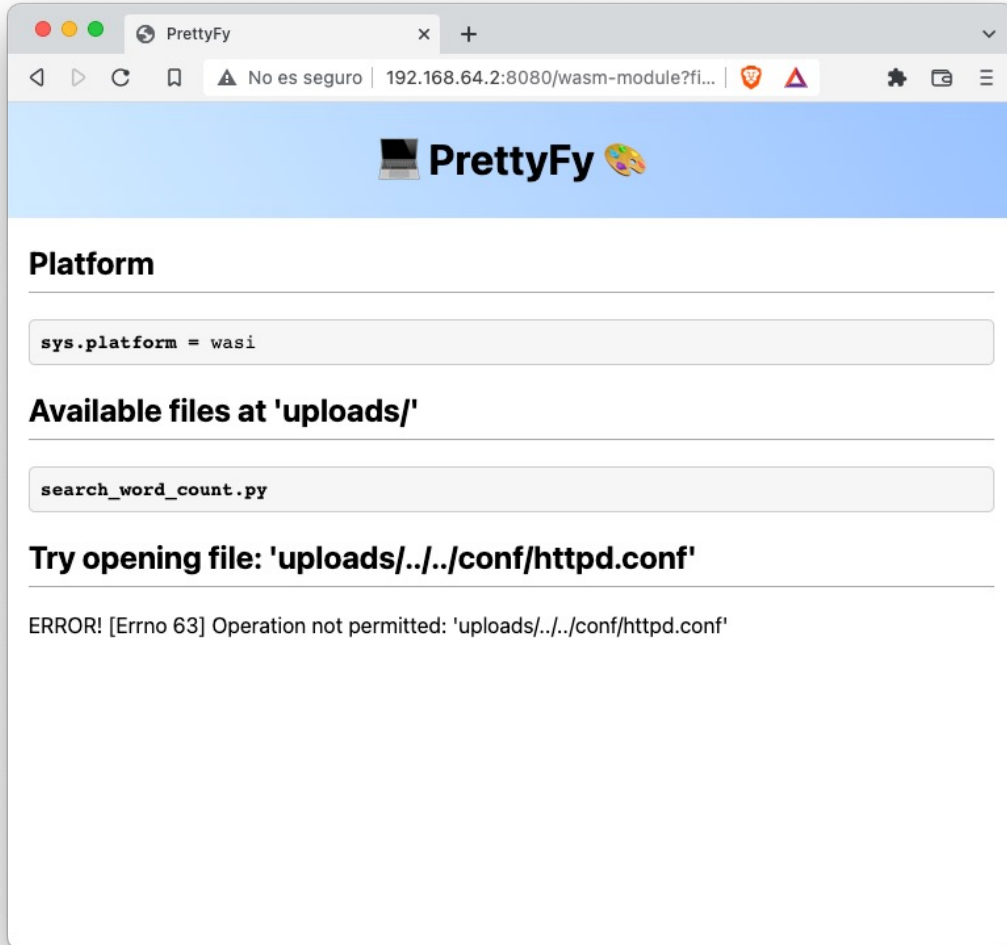


- Executing the script within the Wasm environment.
- Running the same unmodified script:
 - Note **sys.platform** now indicates **wasi**.

Demo #2: PrettyFy WebApp

Wasm + Path Traversal Attack

`http://192.168.64.2:8080/wasm-module?file= ../../../../conf/httpd.conf`



- Path Traversal attack didn't work! 🥳
- The Wasm Capabilities Model prevented the code to get out from its sandboxed context.

CONCLUSION:

mod_wasm allows running untrusted code in a secure environment in Apache (without containers!)

Demo #2: PrettyFy WebApp

What just happened?

- ✓ We already knew that CGI is not secure enough, especially to run untrusted code.
- ✓ Executing the entire Python interpreter in WebAssembly provided a secure environment to run:
 - Untrusted code.
 - Unmodified applications.
 - And with no heavy image containers needed!
- ✓ The Wasm capabilities model can limit the access to the resources:
 - No capabilities enabled by default.
 - Execution is allowed/denied at syscall level.
 - Capabilities can be updated per request.



Behind the Scenes

Directives

Setting up `httpd.conf` for PrettyFy demo

```
LoadModule wasm_module modules/mod_wasm.so

<Location /wasm-module>
    SetHandler wasm-handler
</Location>

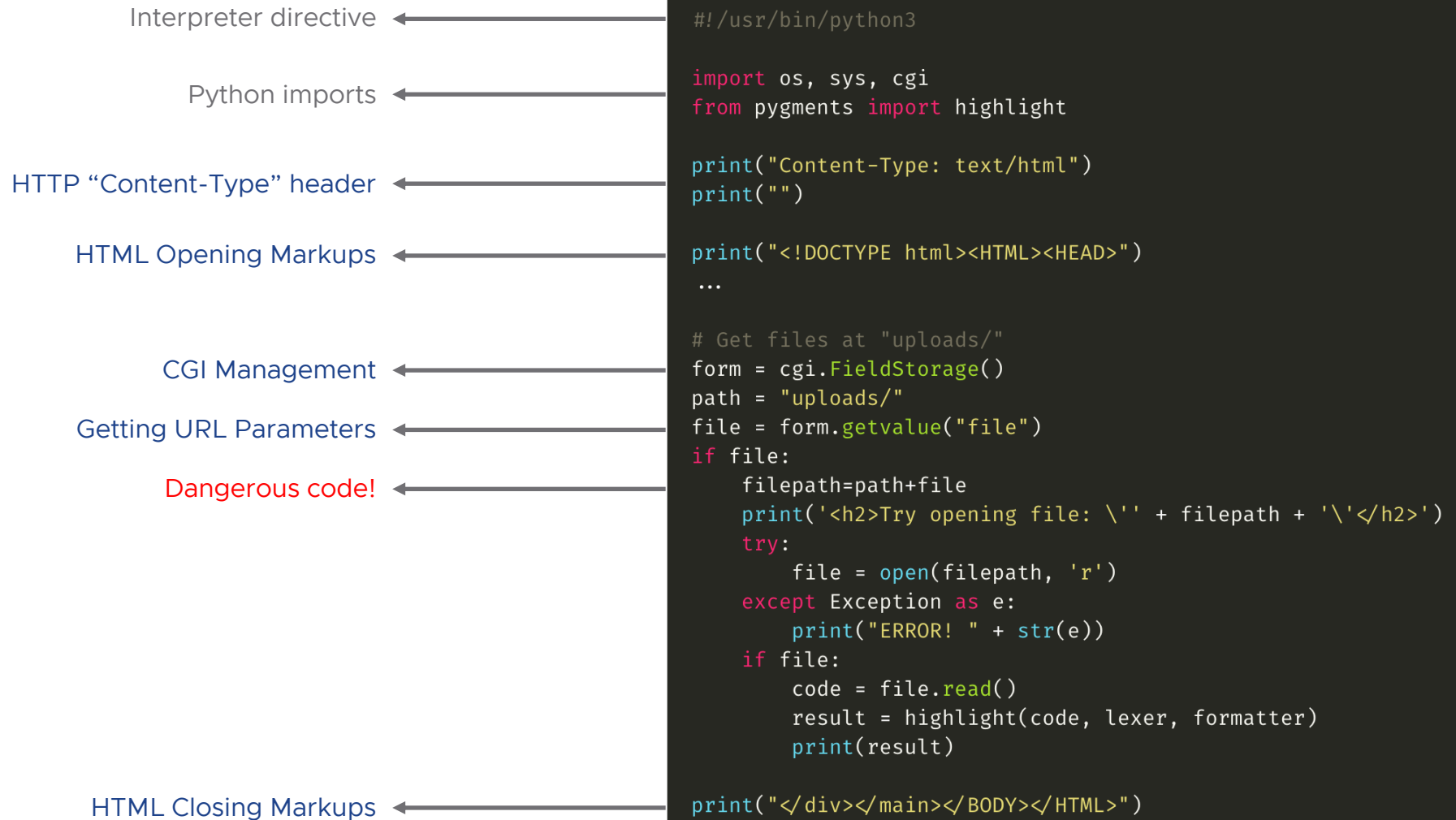
<IfModule wasm_module>
    WasmRoot    /var/www/wasm_modules
    WasmModule  python3.11.wasm
    WasmMapDir  /python /var/www/python
    WasmArg     /python/prettyfy.py
    WasmEnv     PYTHONHOME /python/wasi-python/lib/python3.11
    WasmEnv     PYTHONPATH /python/wasi-python/lib/python3.11
    WasmEnableCGI On
</IfModule>
```

Apache's **mod_wasm** new directives:

- **WasmRoot**
- **WasmModule**
- **WasmDir**
- **WasmMapDir**
- **WasmArg**
- **WasmEnv**
- **WasmEnableCGI**

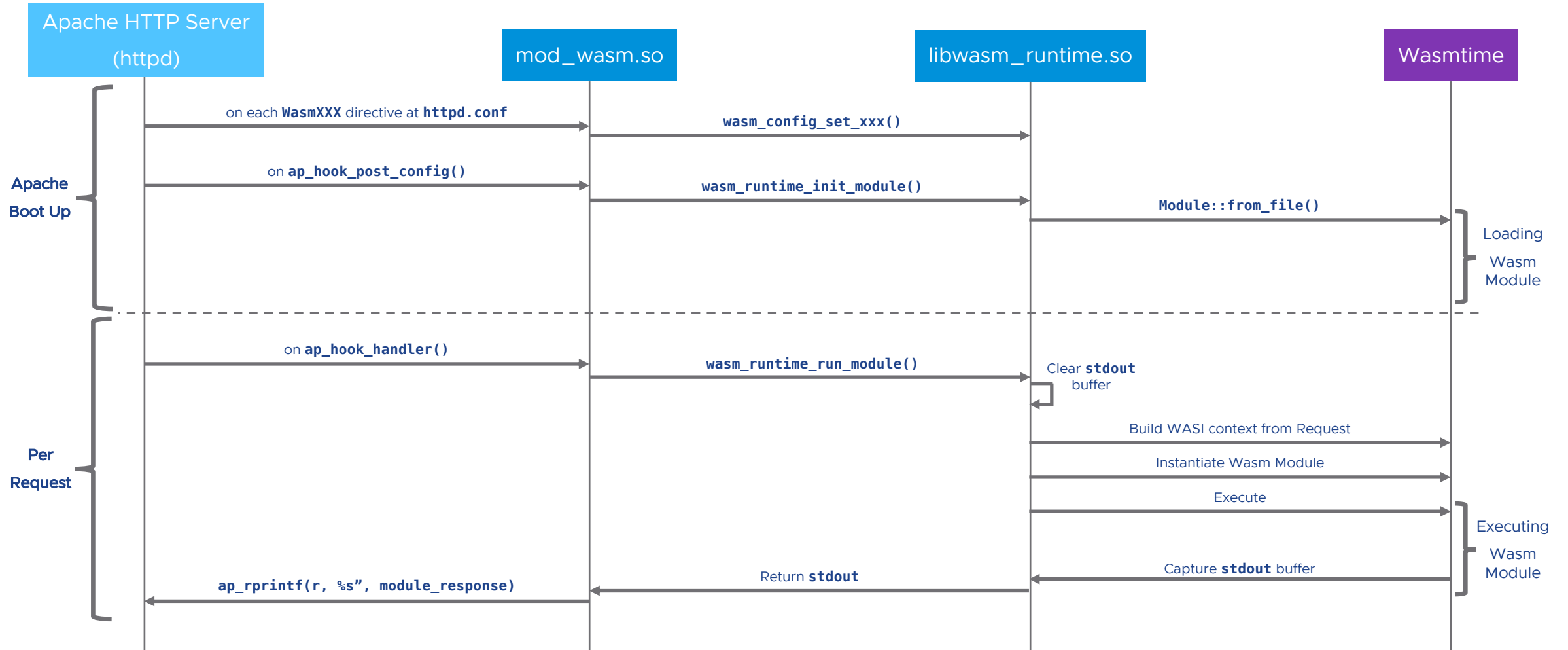
PrettyFy Source Code

prettyfy.py



Workflow

Wasm initialization and request execution



Roadmap

Roadmap

- **Today:**

- **mod_wasm** is already available in GitHub!

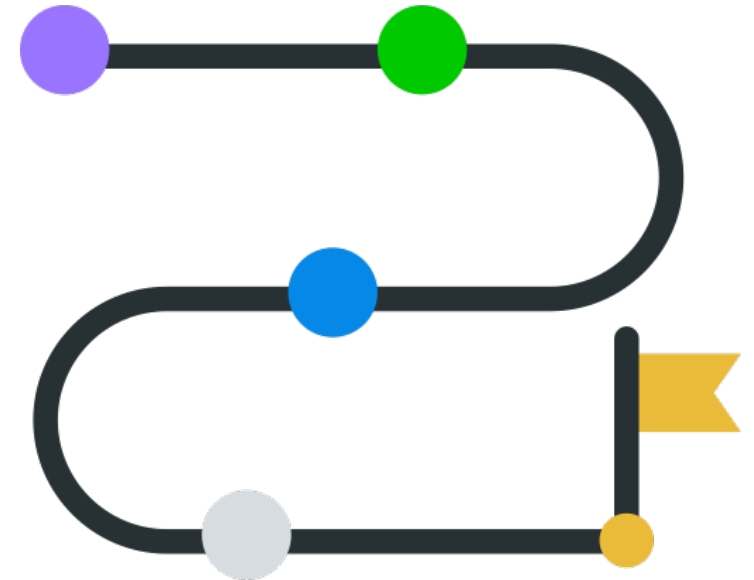
<https://github.com/vmware-labs/mod-wasm>

- **Short-term:**

- Interact with the community for adoption and new features
- Contribute Upstream

- **Mid-term:**

- Implement performance improvements
- Support for more than one Wasm module and entry points
- WebAssembly Multi-engine support



Thanks! 🙌



https://github.com/vmware-labs/mod_wasm



<https://wasmlabs.dev>



@vmwwasm