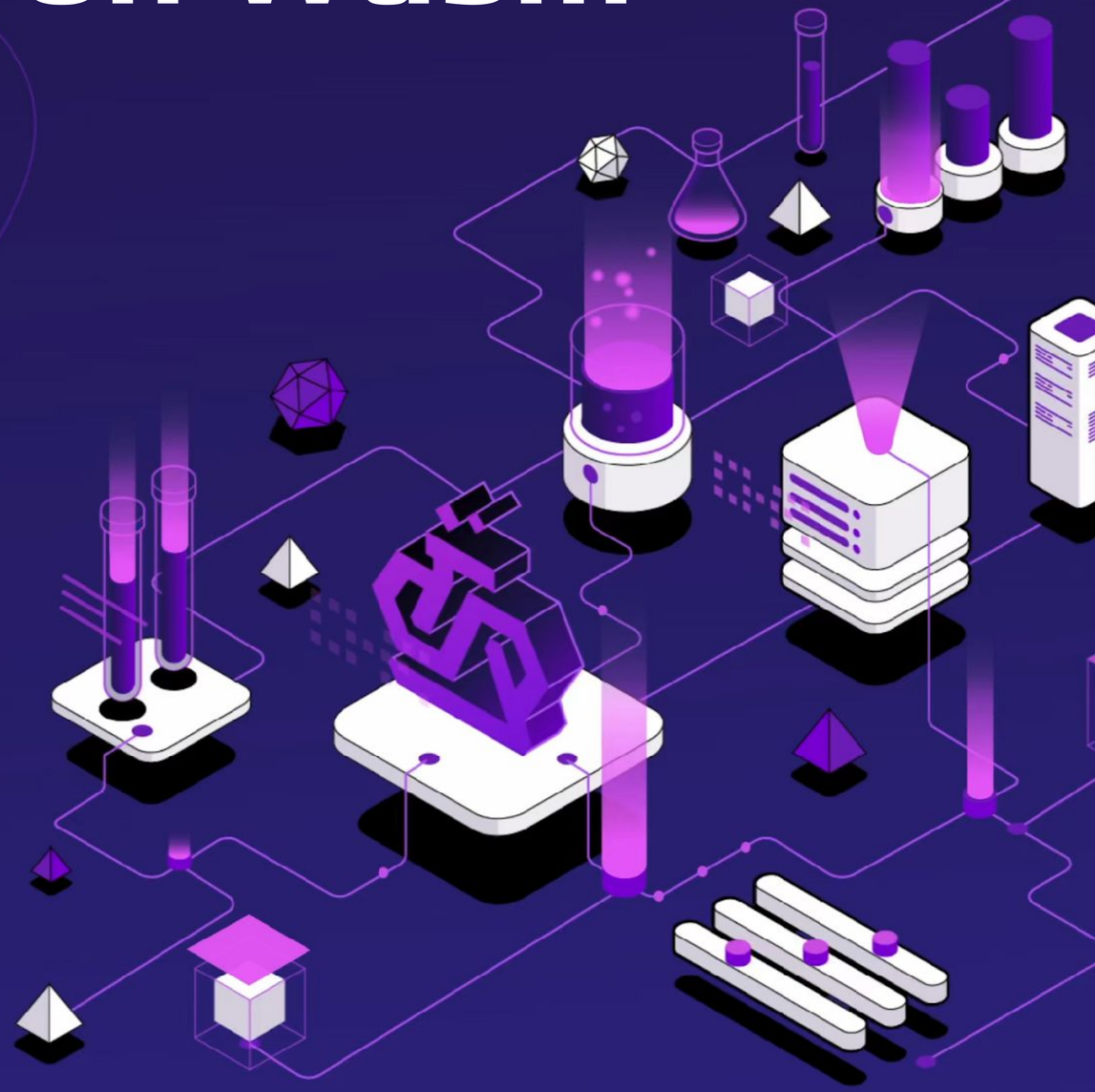
Deploying Python on Wasm

Smaller, Safer, Faster, Universal

Dan Phillips
@d_philla



Loophole Labs

About me:

- Engineer @ Loophole Labs
- Scale Function Runtime (scale.sh)
- @d_philla
- Wasm Chicago Group (wasmchicago.org)

What is WebAssembly?

- WebAssembly (abbreviated *Wasm*) is a safe, portable, low-level code format designed for efficient execution and compact representation
- Safe, sandboxed execution env, “deny-by-default”, makes no assumption about languages, or host.
- Analogy: “Virtual CPU”

Cont’d:

- A Compilation Target
- A Virtual Instruction Set Architecture (ISA)
  -Bytecode binary format
- Stack Machine

Wait, what?

- In a broad sense, Wasm is just another architecture
  - Key diffs:
    - Virtualized
    - Needs a runtime to translate to Machine code
    - Universal

A client-side technology?

- *Safe, sandboxed execution environment... makes no assumption about languages, or host.*
- “Cold start” times in nano- to micro-seconds
- Universal compilation target

- Wasm != Web && Wasm != Assembly

Server-side WebAssembly

-Cloud Infrastructure's "Penicillin Moment"
-VMs → Containers → WebAssembly
  -smaller, safer, *faster, (much more) universal.
-Founder of Docker on Wasm

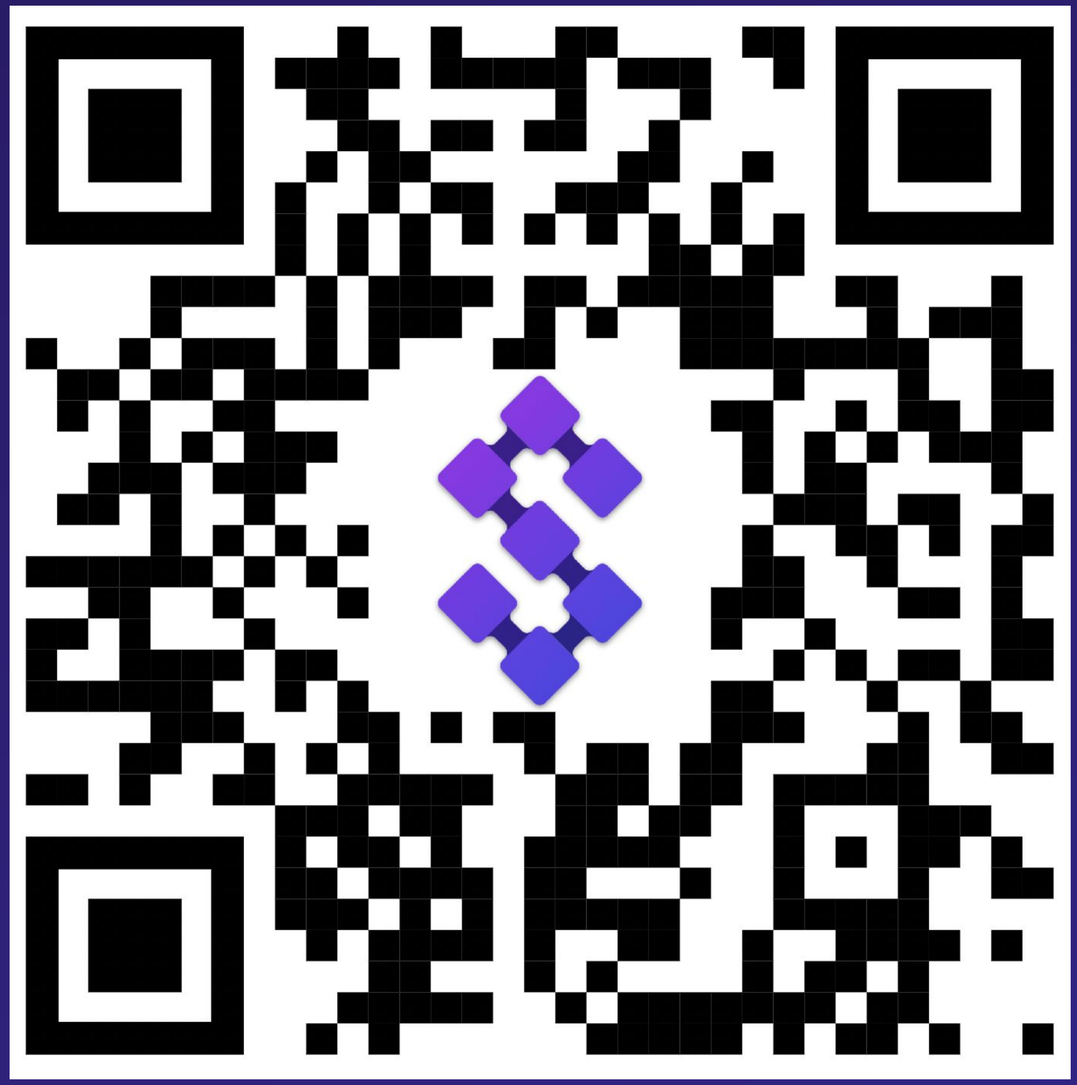
@d_philla

WASI - System Interface

-Started in 2019, initially POSIX Interface for Wasm
-Capability-based security
-evolving standard: Preview 1, 2, 3 (Future)
-is it required ?

Scale

- Plugin framework
- Serverless function runtime
- Polyglot programming in the **same runtime environment**
- written in golang
- Current plugin support for rust, go, typescript



https://scale.sh

Building Python: Assumptions

- Assumptions:
  - Unix
  - Filesystem
  - Dynamic Linking
  - Syscalls/Libc

Pain-points

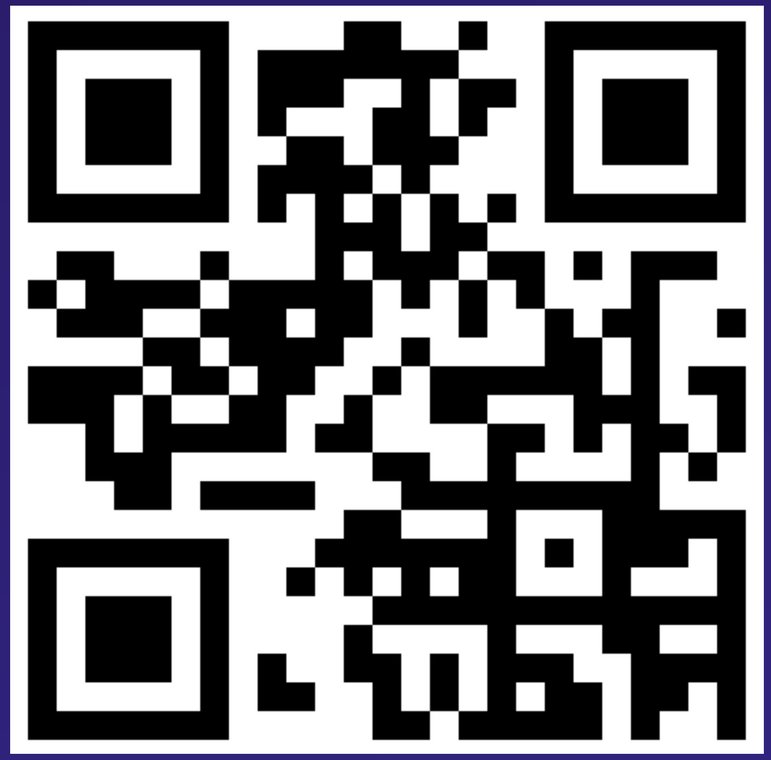
- Limited number of supported Syscalls
- No pthread APIs
- No socket APIs
- Non-comprehensive signal support

- More Detail: Christian Heimes' Wasm Day Talk

📦📦 boxer 📦📦

Container Declaration→ Wasm Binary (+ runtime)

boxer.dev



@d_philla

What is in a (Wasm) Box?

- Base layer
- VFS + Virtualized Sys Code Stubs
- Compiled runtime
- User source code
- *Exports/Imports*

How?

- libc/syscall Interfaces (wasm-libc, wasi-libc)
- a sandboxed FS (wasm-vfs)
- wizer

```
→ FROM ubuntu:latest
→ RUN mkdir -p /app
→ COPY a.out /app
→ WORKDIR /app
→ CMD ["/app/a.out"]
```

@d_philla

wasm-vfs

https://github.com/dphilla/wasm-vfs

- Implemented Syscalls
- POSIX semantics
- Key differences with sync/flush/lock and more
- Architecture Patterns
- (code)

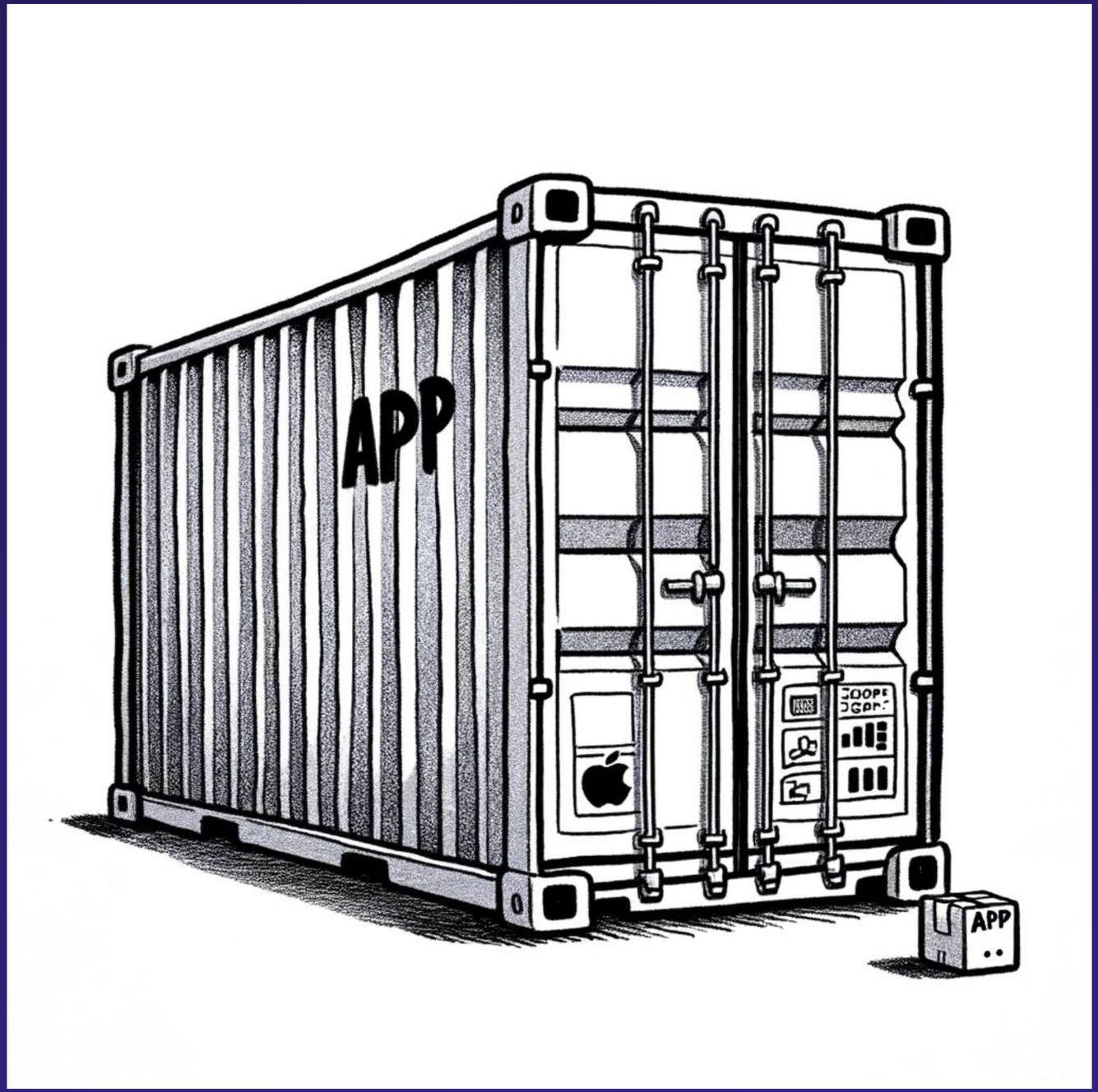
@d_philla

Demo

@d_philla

🚨 (Big) Caveats 🚨

- Threads
- Networking
- Native Dependencies, the problem, and what to do
  - What the benefits Wasm-ifying these does, though
- Future work, current solutions



@d_philla

Key Python Deployment Metrics

Containers
Size: 80 - 900mb
Startup Speed: 800ms - ~2s
Security Model: Shared Kernel

Boxes
Size: 16 mb
Startup Speed ~100µs - ~1 ms:
Security Model: Virtualized
Sandboxed, Machine code execution

@d_philla

The Future

- Full support for Libc + Syscall Interfaces: Import/Exports, runtime host function generation

- Modularize Kernel Stacks 😱
  - Pluggable, Networking stack, etc.
  - Wasm VFS
  - Shims, when needed, modules elsewhere

- A Paradigm-shift: A kernel-free, composable, universal Wasm-based Operating Environment
  - WASI
  - WALI
  - Baremetal Runtimes + Unikernel, etc.

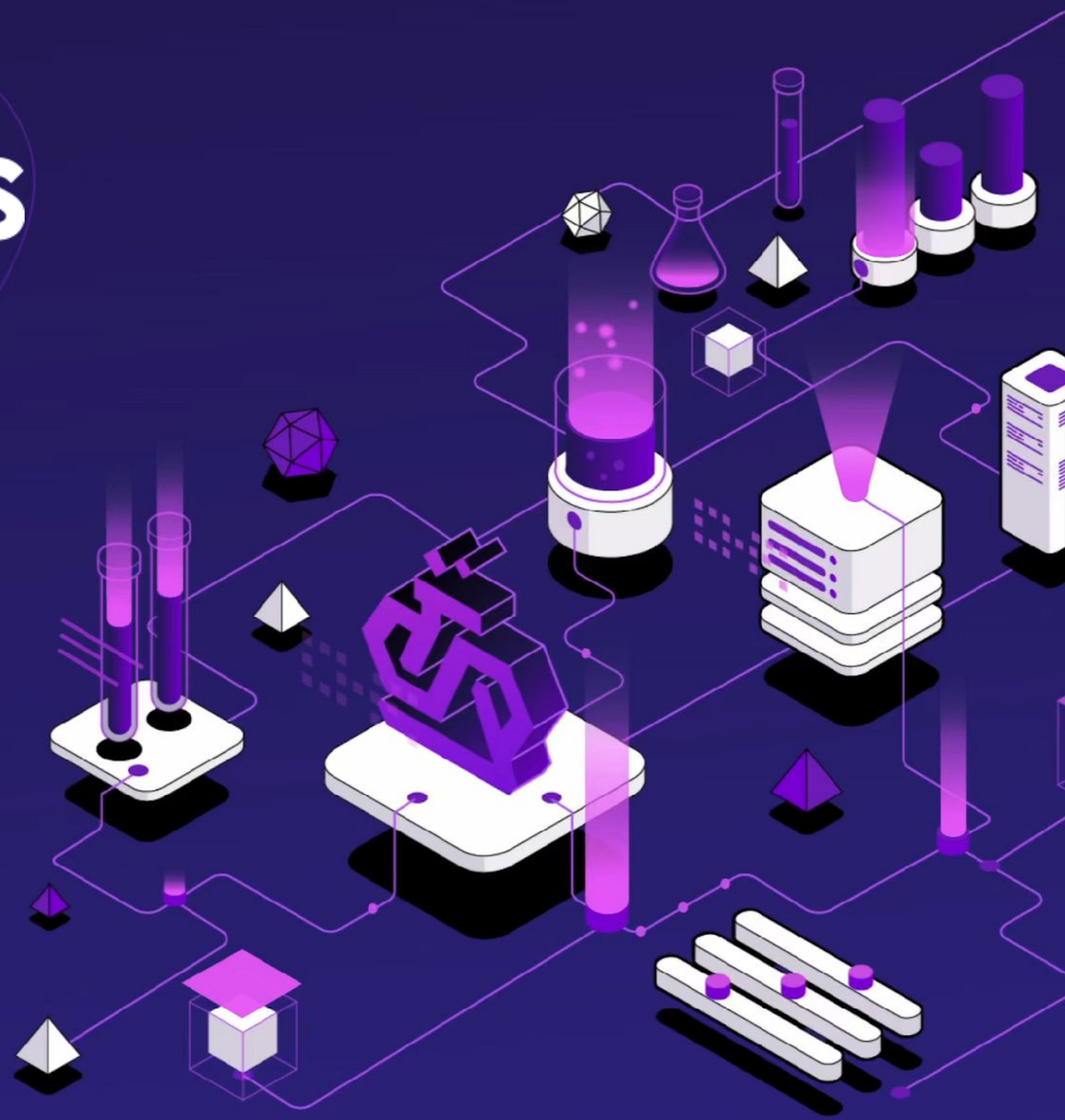
- Unprecedented, True Isomorphism

@d_philla

Thanks!

@d_philla

Loophole Labs



twitter: @LoopholeLabs

web: loopholelabs.io

discord: loopholelabs.io/discord

Dan Phillips

Engineer / Wasm Lead

twitter: d_philla

web: loopholelabs.io

linkedin: linkedin.com/in/d-philla/

Loophole Labs