Don't trust anything

Real-world uses for WebAssembly

Disclaimer!

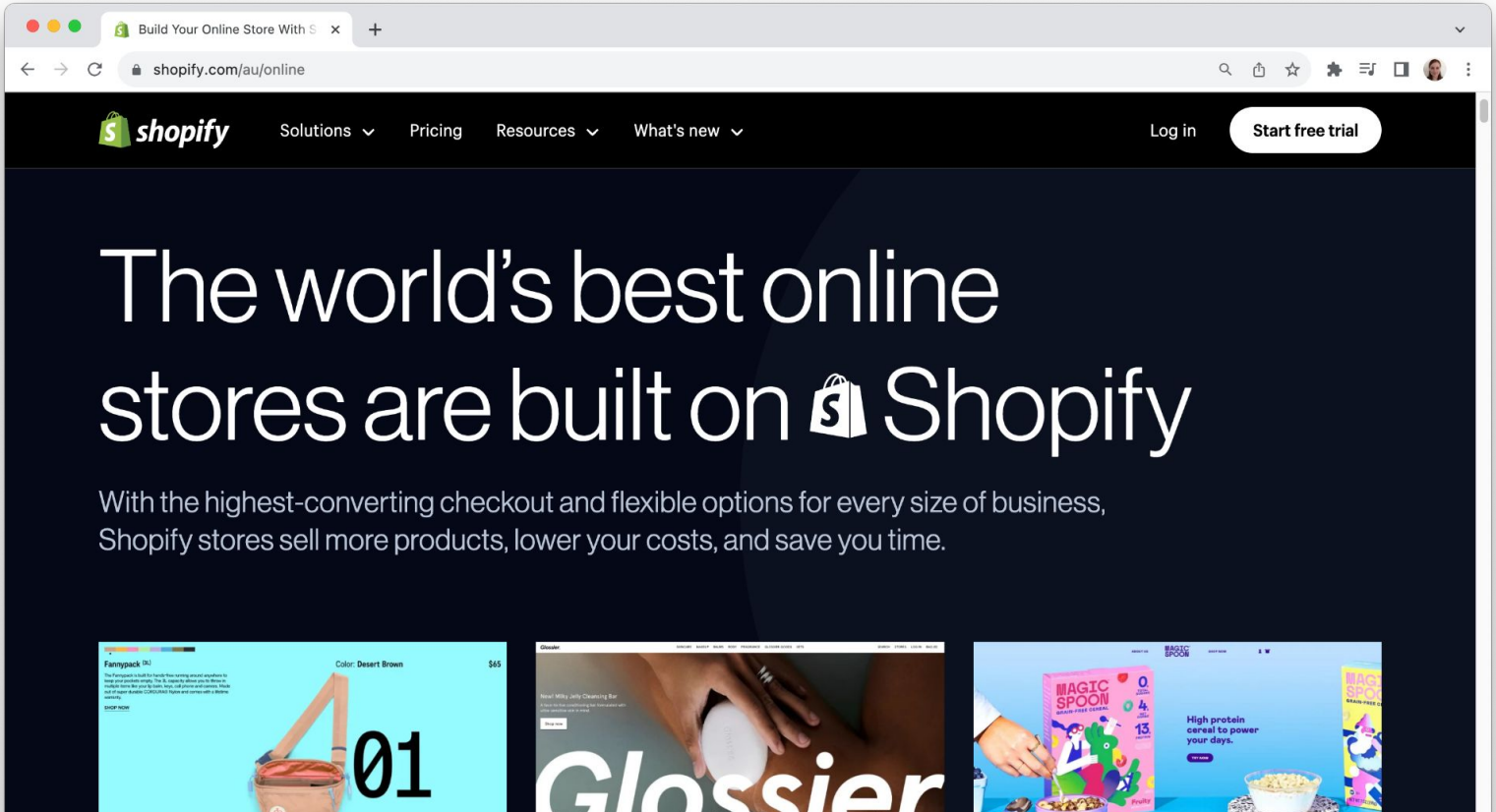
This talk is not security advice :)

Research and consider the tradeoffs for your own situation and risks.

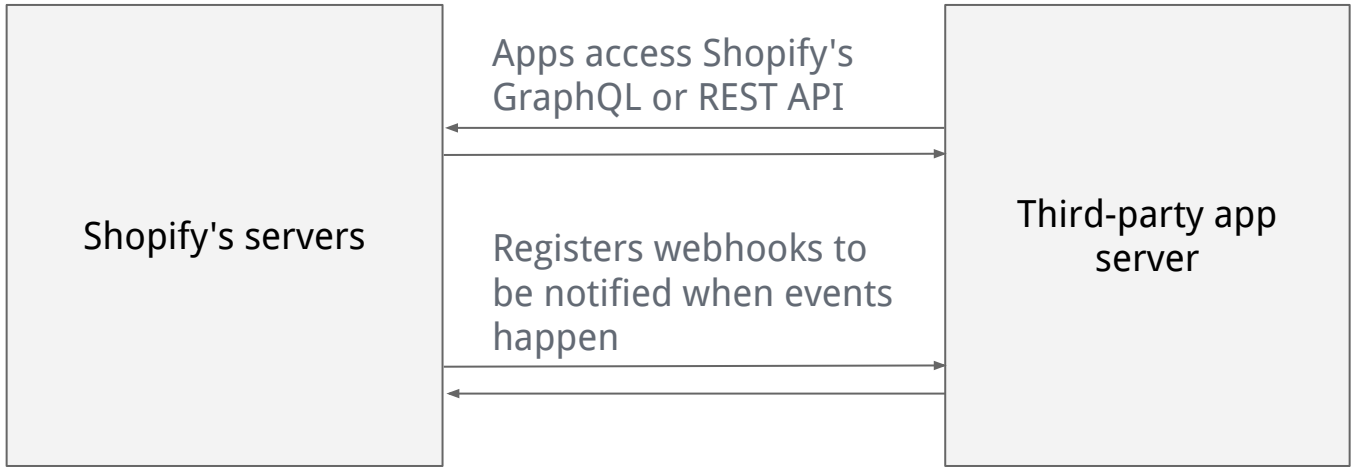
Untrusted code

Code we *really* can't trust.

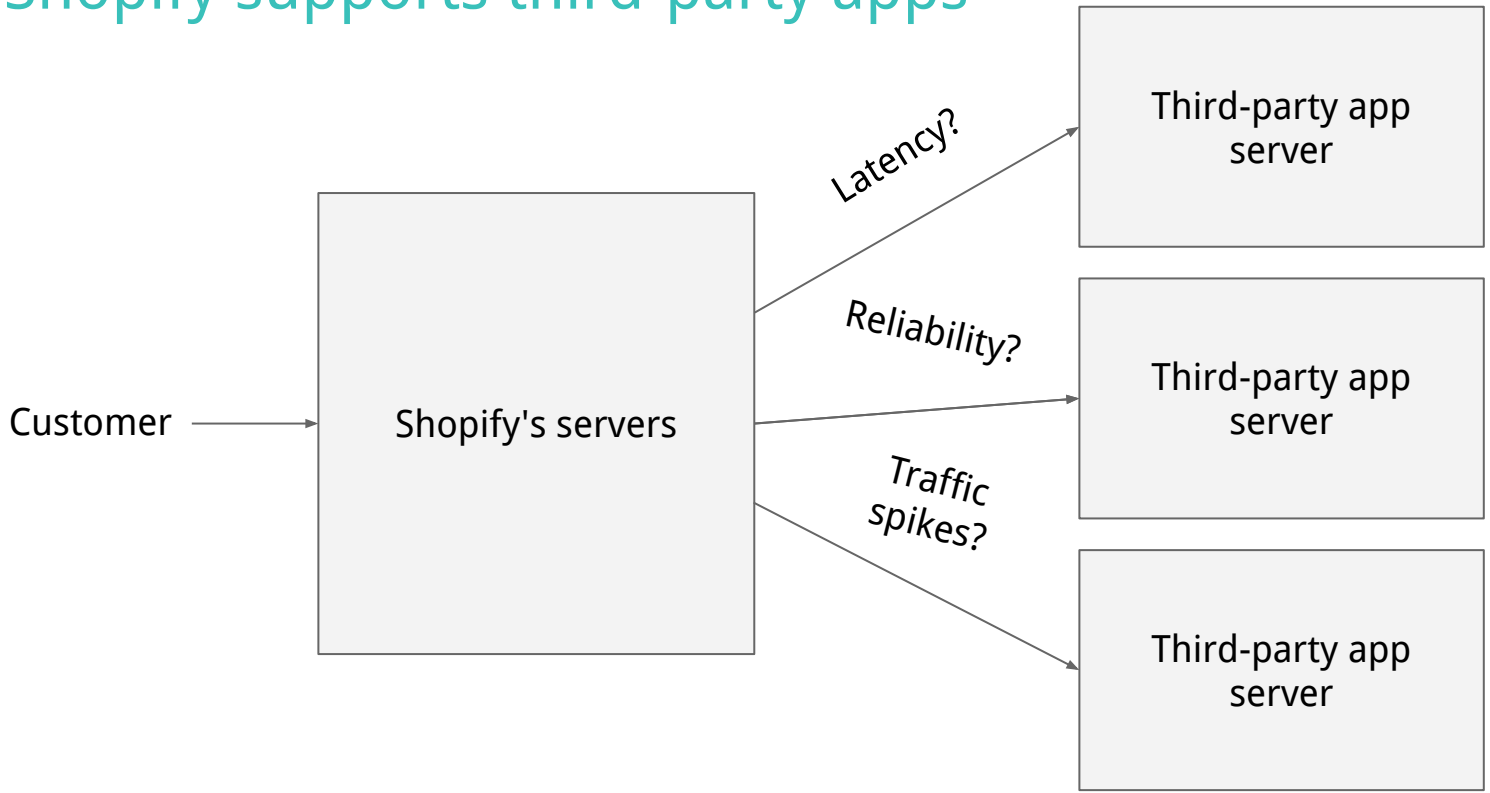
Imagine you're working at Shopify



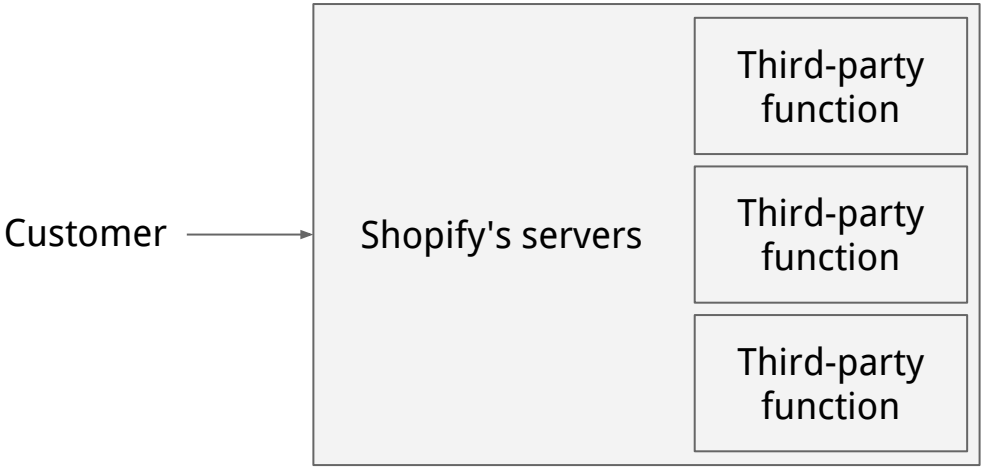
Shopify supports third-party apps



Shopify supports third-party apps



Shopify decided to run third-party code themselves



Spoiler! They used WebAssembly

Duncan Uszkay
How Shopify Uses WebAssembly Outside of the Browser
https://shopify.engineering/shopify-webassembly (2020)

Mitch Dickinson
Making Commerce Extensible with WebAssembly
youtube.com/watch?v=h4bWS1Mmnaw (2020)

```
eval(untrusted_code);
```

```
eval(untrusted_code);
```







Firecracker (or other microVM managers)

- Developed for AWS Lambda & Fargate
- KVM-based MicroVMs (kernel isolation)
- See also: QEMU, Kata Containers etc.




Docker (or containers)

- Lots of settings to lock down process capabilities
- Shared kernel - can be vulnerable to kernel bugs
- Be extra careful with the Docker daemon itself as an attack vector




gVisor

- Isolation layer for containers
- Intercepts system calls and talks to the kernel itself - more isolation from kernel vulnerabilities
- Used by Google Cloud Run, Cloud Functions (1st generation only)

Start time (cold start)

Measured from launch to network stack initialised

- Kata Container (QEMU) ~1.4s
- Kata Container (Firecracker) ~ 1.3s
- Docker ~900ms
- gVisor (kvm) ~800ms
- gVisor (ptrace) ~800ms

Guoqing Li et al. Comparative Performance Study of Lightweight Hypervisors Used in Container Environment (2021)
https://www.scitepress.org/Papers/2021/104405/104405.pdf

V8 Isolates

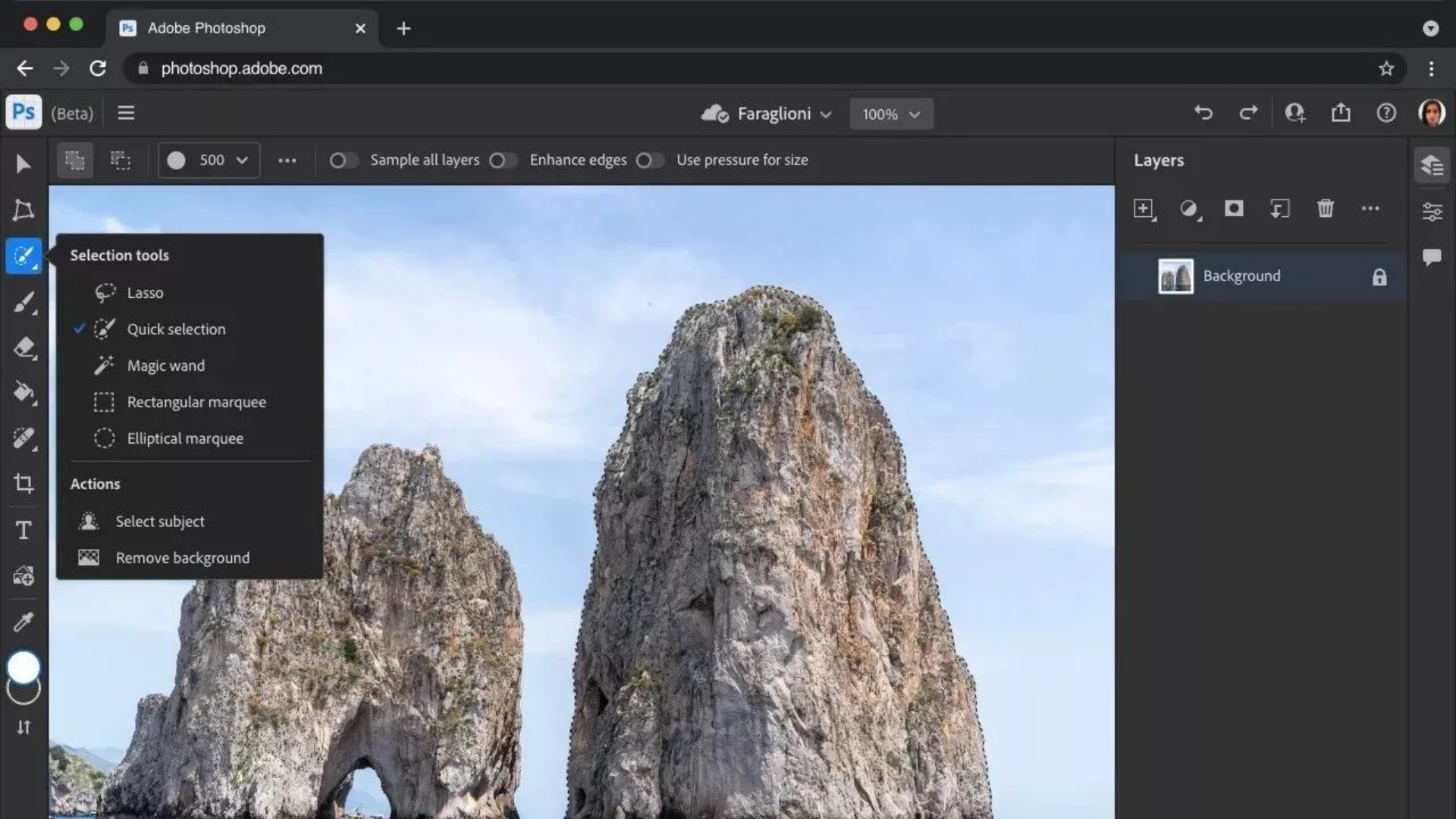
- Built into V8 engine, used for isolating JS execution within a Chrome tab
- Used by CloudFlare to run many workers within a process - 5ms start time

https://blog.cloudflare.com/cloud-computing-without-containers/
https://blog.cloudflare.com/mitigating-spectre-and-other-security-threats-the-cloudflare-workers-security-model/

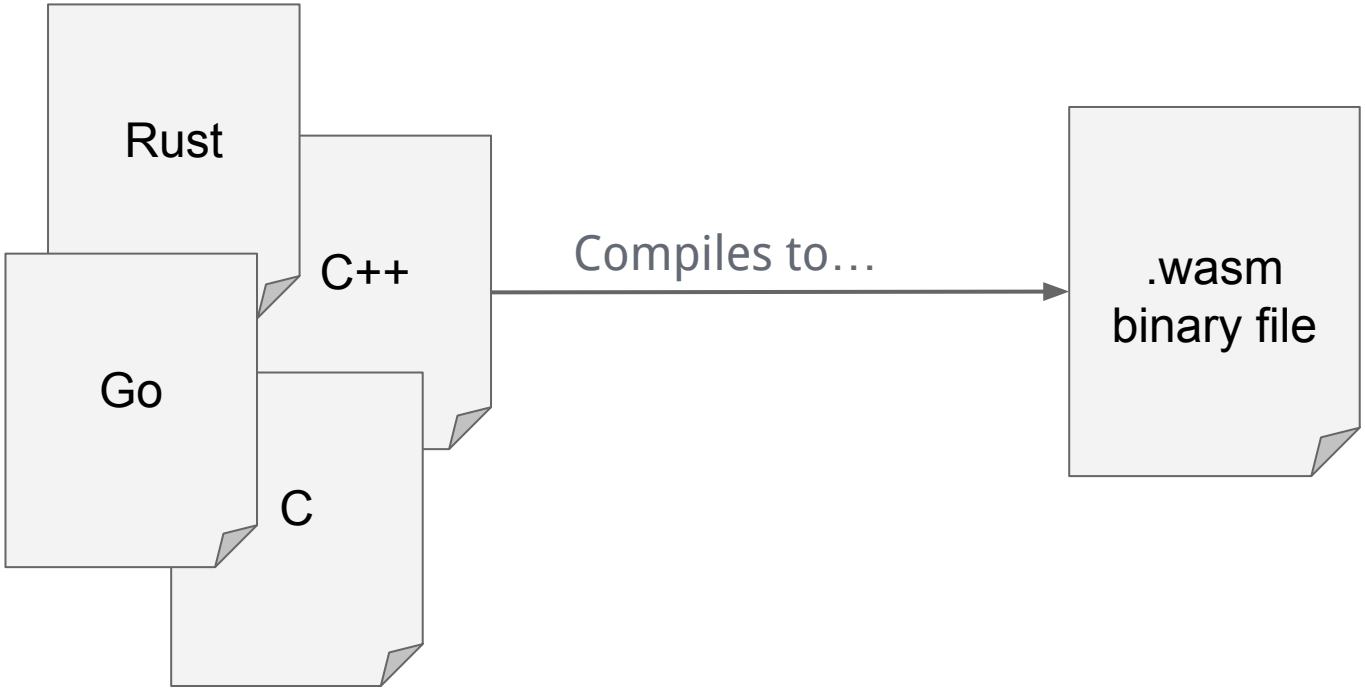
That's not a fair comparison!

WebAssembly

WebAssembly was built for browsers

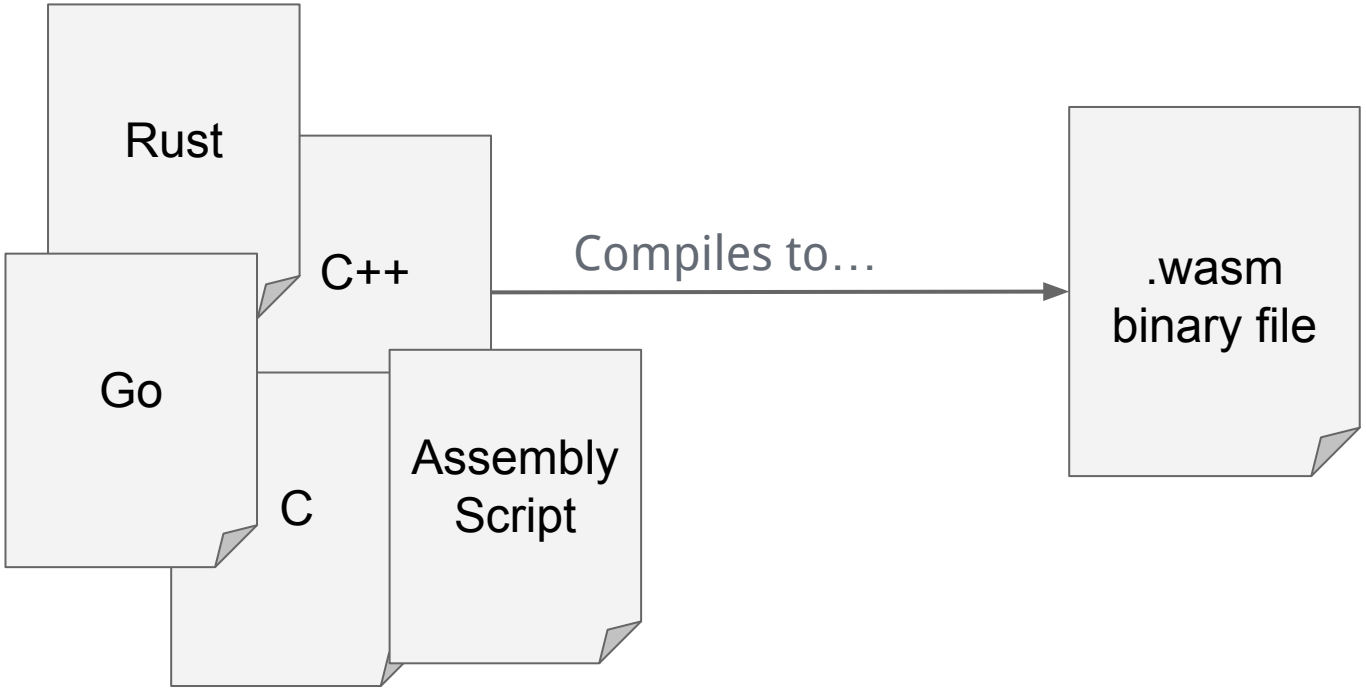

Adobe Photoshop
photoshop.adobe.com
(Beta)
Faraglioni
100%
500
Sample all layers
Enhance edges
Use pressure for size
Selection tools
Lasso
Quick selection
Magic wand
Rectangular marquee
Elliptical marquee
Actions
Select subject
Remove background
Layers
Background

WebAssembly is a compile target



```
// Add two numbers in AssemblyScript
export function add(a: i32, b: i32): i32 {
  return a + b;
}
```

WebAssembly is a compile target



```
// Add two numbers in AssemblyScript
export function add(a: i32, b: i32): i32 {
  return a + b;
}
```

```
// Add two numbers in AssemblyScript
export function add(a: i32, b: i32): i32 {
  return a + b;
}
```

Compiled to wasm:

```
(module
 (type $0 (func (param i32 i32) (result i32)))
 (memory $0 0)
 (export "add" (func $assembly/index/add))
 (export "memory" (memory $0))
 (func $assembly/index/add (param $0 i32) (param $1 i32) (result i32)
  local.get $0
  local.get $1
  i32.add
 )
)
```

```
// JavaScript this time!

// Using a .wasm (binary) module.
const module = await WebAssembly.compile(fs.readFileSync('add.wasm'));
const { exports } = await WebAssembly.instantiate(module);

console.log(exports.add(234, 23)) // 257
```

WebAssembly

Like a tiny simulated computer

- with its own call stack and instructions
- with its own byte-array of memory
- with no external access unless explicitly provided

What can't WebAssembly code do?

1. Access variables or memory outside its own
2. Anything really…
    - Read/write files
    - Use the network
    - Spawn processes
    - Interact with other processes
    - Generate random numbers
    - Get the current time
    - Interact with hardware devices
    - Create or listen to sockets
    - Send signals

What *can* WebAssembly code do?

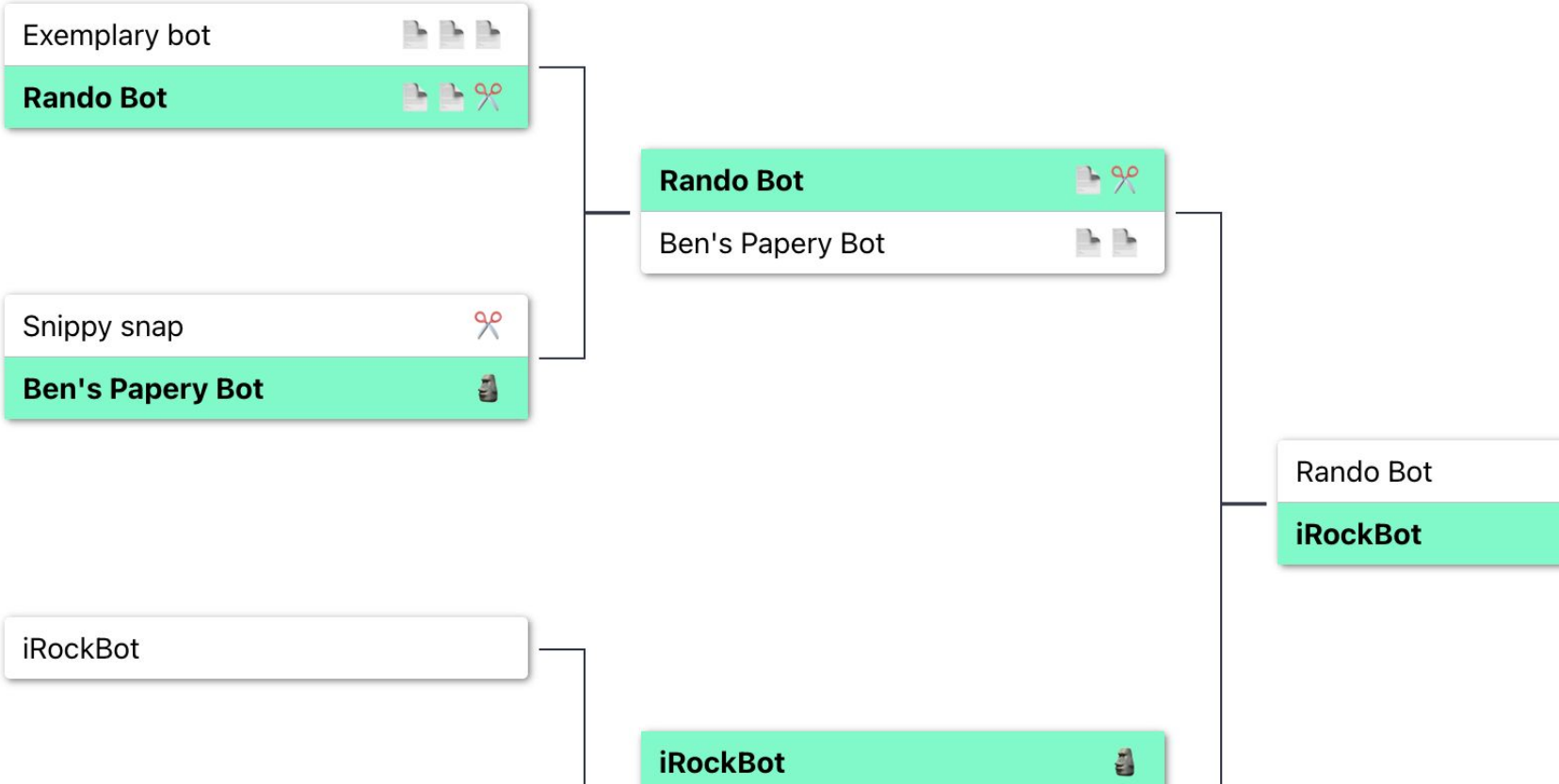
1. Calculate things.

2. Call import functions (if provided).

Sandboxing without even using a separate process!

Sounds easy!
Let's try using it.

Snippy: A Rock-Paper-Scissors bot tournament



```
// Rock-paper-scissors bot function (AssemblyScript).
export function rockPaperScissors(): string {
  return "rock";
}
```

```
// Rock-paper-scissors bot function (AssemblyScript).
export function rockPaperScissors(): string {
  return "rock";
}
```

Compiled to wasm:

```
(module
 (type $0 (func (result i32)))
 (memory $0 1)
 (data $0 (i32.const 1036) "\1c")
 (data $0.1 (i32.const 1048) "\02\00\00\00\08\00\00\00r\00o\00c\00k")
 (export "rockPaperScissors" (func $assembly/index/rockPaperScissors))
 (export "memory" (memory $0))
 (func $assembly/index/rockPaperScissors (result i32)
  i32.const 1056
 )
)
```

```
// Rock-paper-scissors bot function (AssemblyScript).
export function rockPaperScissors(): string {
  return "rock";
}
```

Compiled to wasm:

```
(module
 (type $0 (func (result i32)))
 (memory $0 1)
 (data $0 (i32.const 1036) "\1c")
 (data $0.1 (i32.const 1048) "\02\00\00\00\08\00\00\00r\00o\00c\00k")
 (export "rockPaperScissors" (func $assembly/index/rockPaperScissors))
 (export "memory" (memory $0))
 (func $assembly/index/rockPaperScissors (result i32)
  i32.const 1056
 )
)
```

```
// Rock-paper-scissors bot function (AssemblyScript).
export function rockPaperScissors(): string {
  return "rock";
}
```

Compiled to wasm:

```
(module
 (type $0 (func (result i32)))
 (memory $0 1)
 (data $0 (i32.const 1036) "\1c")
 (data $0.1 (i32.const 1048) "\02\00\00\00\08\00\00\00r\00o\00c\00k")
 (export "rockPaperScissors" (func $assembly/index/rockPaperScissors))
 (export "memory" (memory $0))
 (func $assembly/index/rockPaperScissors (result i32)
  i32.const 1056
 )
)
```

```
// Rock-paper-scissors bot function (AssemblyScript).
export function rockPaperScissors(): string {
  return "rock";
}
```

Compiled to wasm:

```
(module
 (type $0 (func (result i32)))
 (memory $0 1)
 (data $0 (i32.const 1036) "\1c")
 (data $0.1 (i32.const 1048) "\02\00\00\00\08\00\00\00r\00o\00c\00k")
 (export "rockPaperScissors" (func $assembly/index/rockPaperScissors))
 (export "memory" (memory $0))
 (func $assembly/index/rockPaperScissors (result i32)
  i32.const 1056
 )
)
```

Challenges:

1. Getting data in/out is hard without generated wrapper code

Challenges:

1. Getting data in/out is hard without generated wrapper code

2. Bots will need random numbers

WASI
WebAssembly System Interface

```
console.log("Hello world!")
console.log(Math.random().toString())
console.log("rock")
```

```
console.log("Hello world!")
console.log(Math.random().toString())
console.log("rock")
```

```
$ wasmtime build/release.wasm
Hello world!
0.45294487304497630
rock
```

```
console.log("Hello world!")
console.log(Math.random().toString())
console.log("rock")
```

Compiled to wasm with WASI (~4000 lines)

```
(module
 (type $0 (func (param i32)))
 (type $1 (func))
...snip...
 (type $11 (func (result i32)))
 (type $12 (func (param i32 i32 i32 i32)))
 (import "wasi_snapshot_preview1" "fd_write" (func $~lib/bindings/wasi_snapshot_preview1/fd_wri
 (import "wasi_snapshot_preview1" "proc_exit" (func $~lib/bindings/wasi_snapshot_preview1/proc_
 (import "wasi_snapshot_preview1" "random_get" (func $~lib/bindings/wasi_snapshot_preview1/rand
 (global $~argumentsLength (mut i32) (i32.const 0))
 (global $~lib/rt/tlsf/ROOT (mut i32) (i32.const 0))
 (global $~lib/math/random_seeded (mut i32) (i32.const 0))
 (global $~lib/math/random_state0_64 (mut i64) (i64.const 0))
 (global $~lib/math/random_state1_64 (mut i64) (i64.const 0))
```

```
console.log("Hello world!")
console.log(Math.random().toString())
console.log("rock")
```

Compiled to wasm with WASI (~4000 lines)

```
(module
 (type $0 (func (param i32)))
 (type $1 (func))
...snip...
 (type $11 (func (result i32)))
 (type $12 (func (param i32 i32 i32 i32)))
 (import "wasi_snapshot_preview1" "fd_write" (func $~lib/bindings/wasi_snapshot_preview1/fd_wri
 (import "wasi_snapshot_preview1" "proc_exit" (func $~lib/bindings/wasi_snapshot_preview1/proc_
 (import "wasi_snapshot_preview1" "random_get" (func $~lib/bindings/wasi_snapshot_preview1/rand
 (global $~argumentsLength (mut i32) (i32.const 0))
 (global $~lib/rt/tlsf/ROOT (mut i32) (i32.const 0))
 (global $~lib/math/random_seeded (mut i32) (i32.const 0))
 (global $~lib/math/random_state0_64 (mut i64) (i64.const 0))
 (global $~lib/math/random_state1_64 (mut i64) (i64.const 0))
```

WASI spec (preview1)

Provides API for:

- Commandline args
- Environment variables
- Clock time
- Get random
- Filesystem access
- Use (but not create) sockets

Wait…

Wasn't the whole point that wasm couldn't do those??

Goal of WASI

"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.

These APIs preserve the essential sandboxed nature of WebAssembly through a Capability-based API design."

https://github.com/WebAssembly/WASI

WASI spec (preview1)

Provides API for:

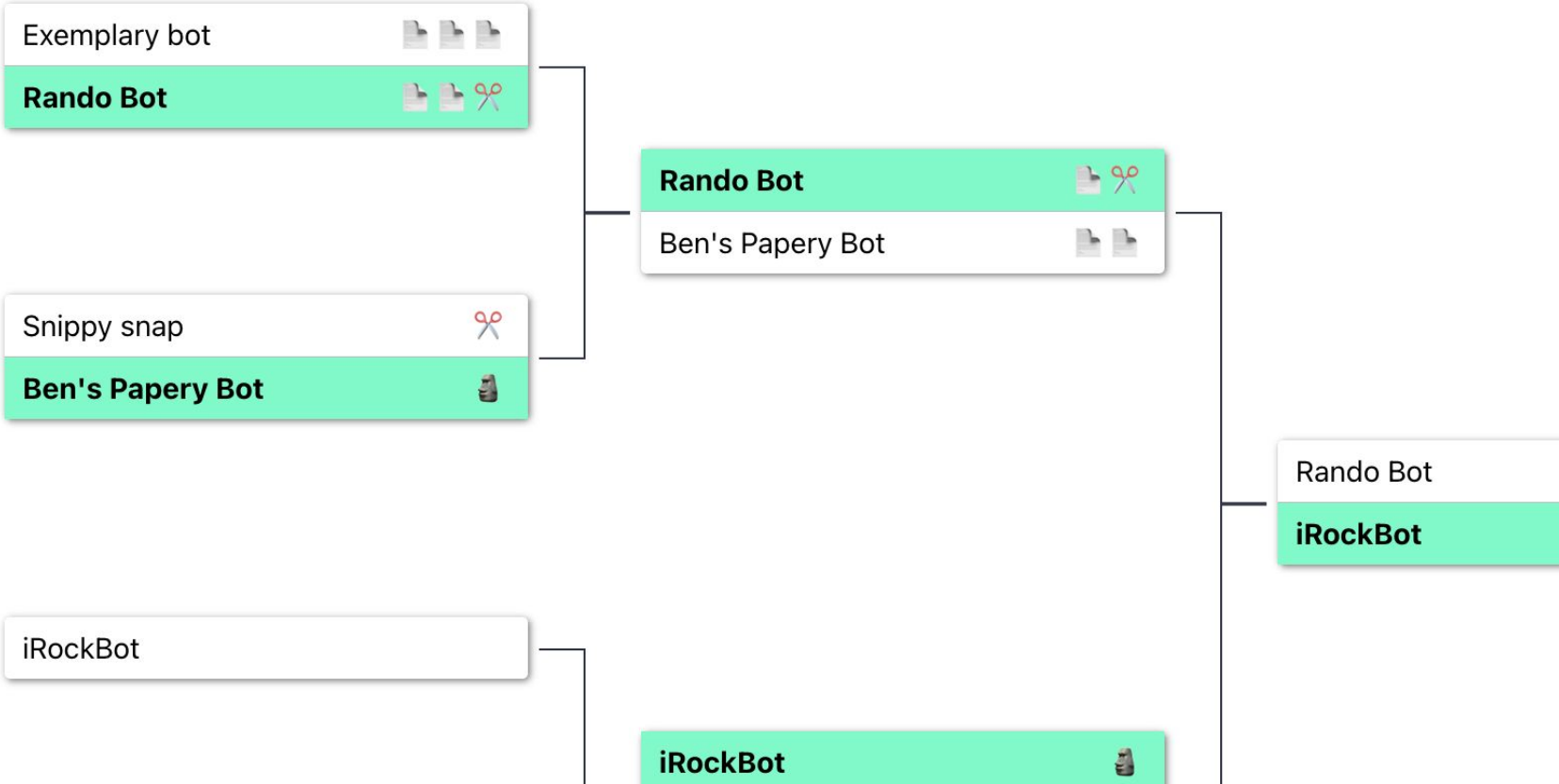
- Commandline args
- Environment variables
- Clock time
- Get random
- Filesystem access
- Use (but not create) sockets

Many languages can run under Wasm+WASI

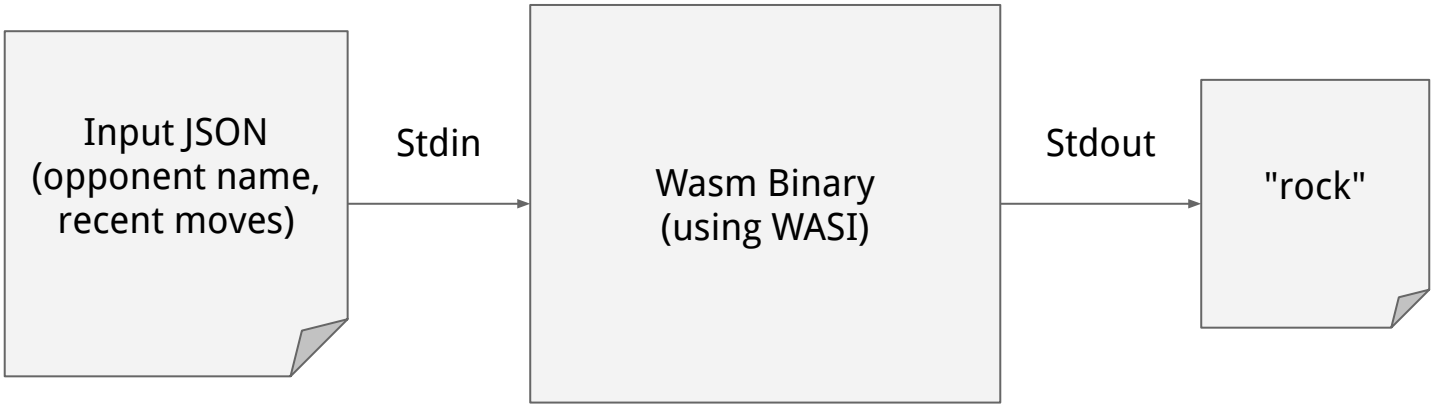
- C/C++
- Rust
- Go
- AssemblyScript
- Python
- Ruby
- PHP
- JavaScript (but not Node)
- C# *
- Java *

* WebAssembly support is good, WASI not as much.

Snippy!


Exemplary bot
Rando Bot
Snippy snap
Ben's Papery Bot
iRockBot
Rando Bot
Ben's Papery Bot
Rando Bot
iRockBot
iRockBot

Snippy!


Input JSON
(opponent name,
recent moves)
Stdin
Wasm Binary
(using WASI)
Stdout
"rock"

Live Demo!

Audience Participation Live Demo!

katiebell.net/snippy

Snippy uses the Rust Wasmtime API

Uploaded modules are restricted:

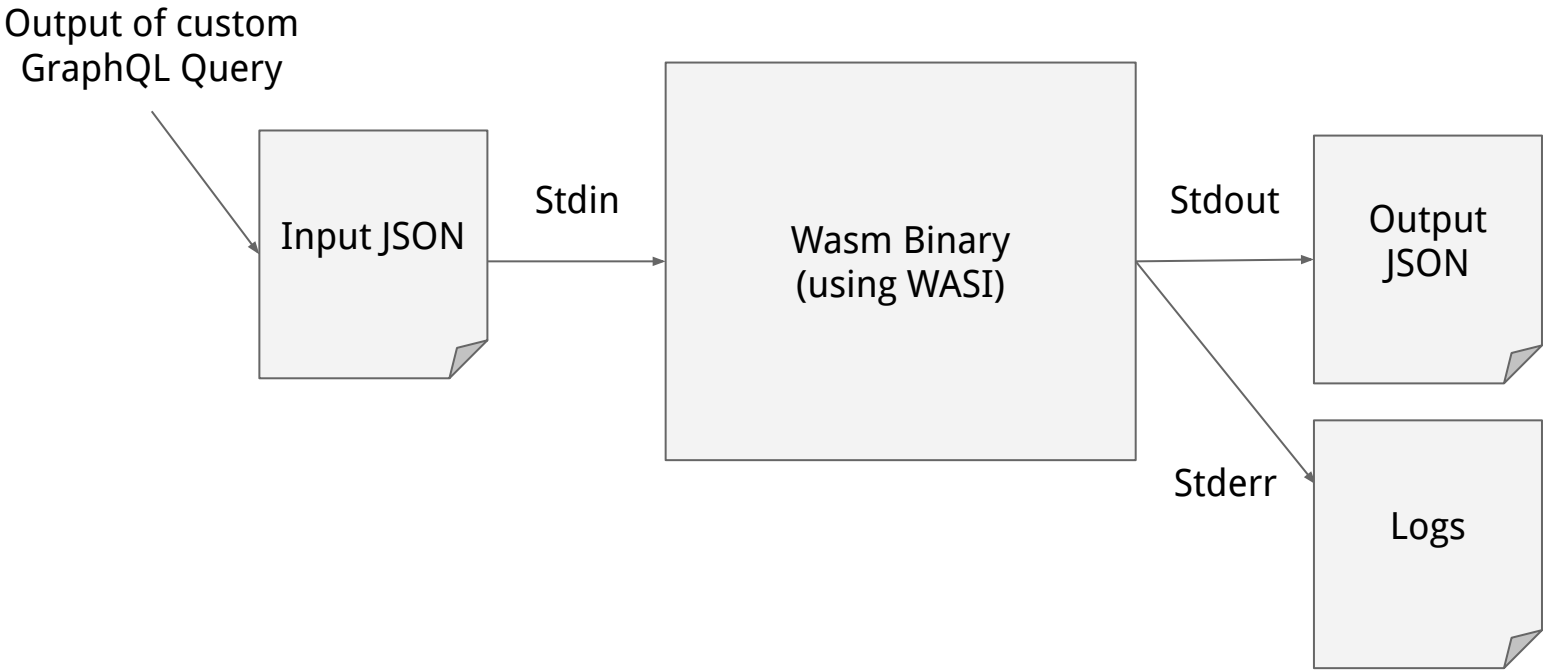
- In-memory streams for stdin/stdout/stderr
- Limited "fuel" (number of instructions)
- Limited time
- Limited memory
- No filesystem access*

*Python bots are given read-only access to a temp directory

How is WebAssembly used in the real world?

Case Study: Shopify Functions

Case study: Shopify Functions



See: https://shopify.dev/docs/apps/functions

Case Study: Mozzilla Firefox

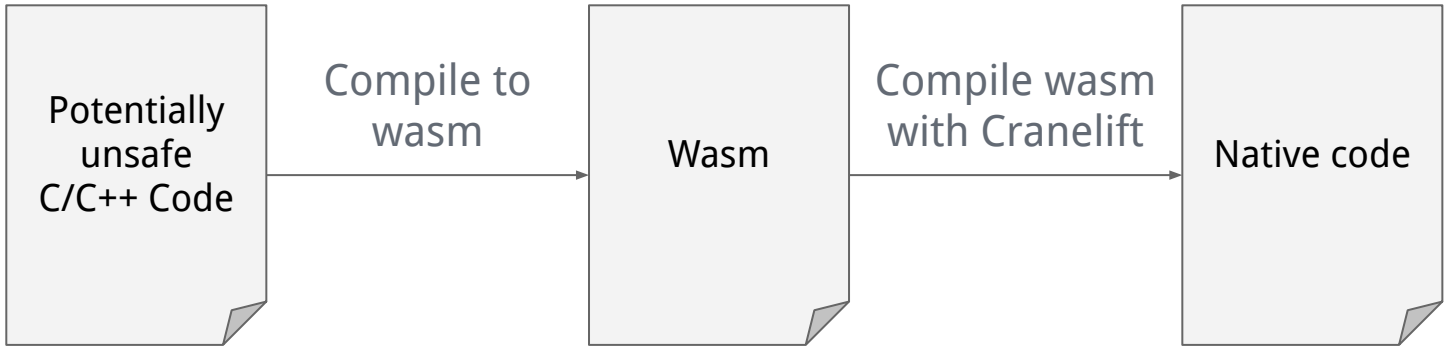
Case study: Mozzilla Firefox

"Google, Microsoft, and Mozilla have each independently found that around 70% of security bugs in their browsers were memory safety bugs."

- Nathan Froyd (Mozzilla Engineer)

https://www.zdnet.com/article/chrome-70-of-all-security-bugs-are-memory-safety-issues/

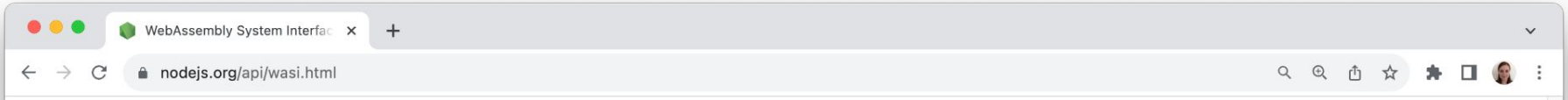
Case study: Mozzilla Firefox



https://hacks.mozilla.org/2020/02/securing-firefox-with-webassembly/

How secure are we talking?

Not all Wasm/WASI implementations are created equal.



WebAssembly System Interface (WASI)

Stability: 1 - Experimental

The `node:wasi` module does not currently provide the comprehensive file system security properties provided by some WASI runtimes. Full support for secure file system sandboxing may or may not be implemented in future. In the mean time, do not rely on it to run untrusted code.

Source Code: lib/wasi.js

The WASI API provides an implementation of the WebAssembly System Interface specification.

Wasmtime

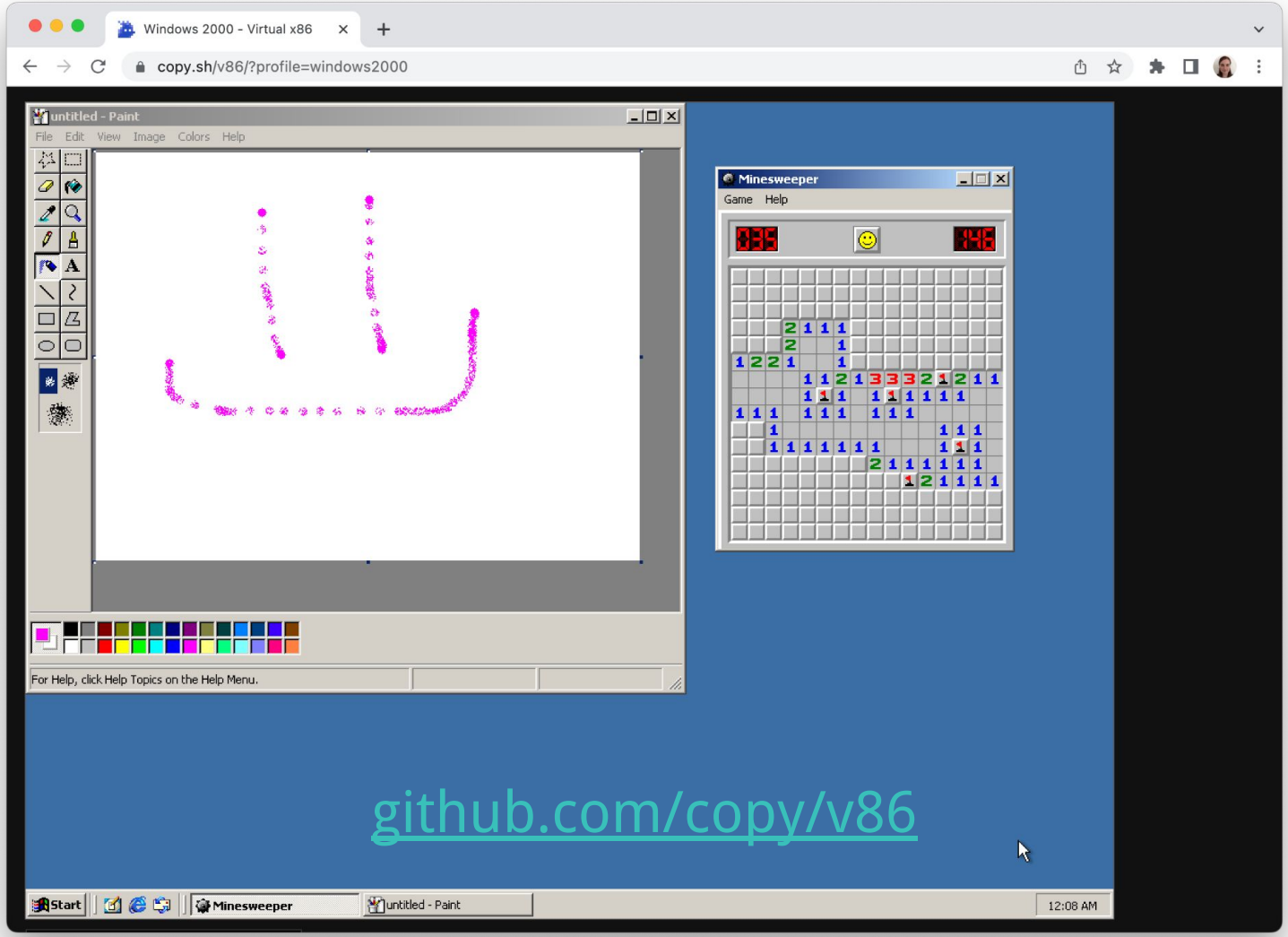
Wasmtime was built for sandboxing

- Built in Rust for memory safety
- Very limited use of unsafe blocks
- Audit and review of all dependencies
- Continuous fuzzing
- Spectre mitigations

https://bytecodealliance.org/articles/security-and-correctness-in-wasmtime

Reminder: Security in depth

Where are we now?


Windows 2000 - Virtual x86
copy.sh/v86/?profile=windows2000
untitled - Paint
File Edit View Image Colors Help
For Help, click Help Topics on the Help Menu.
Minesweeper
Game Help
github.com/copy/v86
Start
Minesweeper
untitled - Paint
12:08 AM

It's still early days!

- WebAssembly is pretty solid.
- WASI specification is still in "preview"
- Wasmtime API developer experience is variable

It's getting better!

- Tooling and language support is getting better
- WebAssembly Component Model will make function-call interfaces consistent
- Wasm & WASI containers (beta) https://docs.docker.com/desktop/wasm/

With Wasmtime, we have achieved very fast instance instantiation times - typically within a few microseconds, not milliseconds.

Fastly is therefore able to provide a significantly faster code execution startup time than any other serverless solution, freeing developers to focus on application development.

- Fastly website

When are you running untrusted code?

How are those bots doing?

Please hack Snippy :)

github.com/katharosada/wasm-snippy

Thanks!

You can find me at:

katiebell.net

aus.social/@notsolonecoder