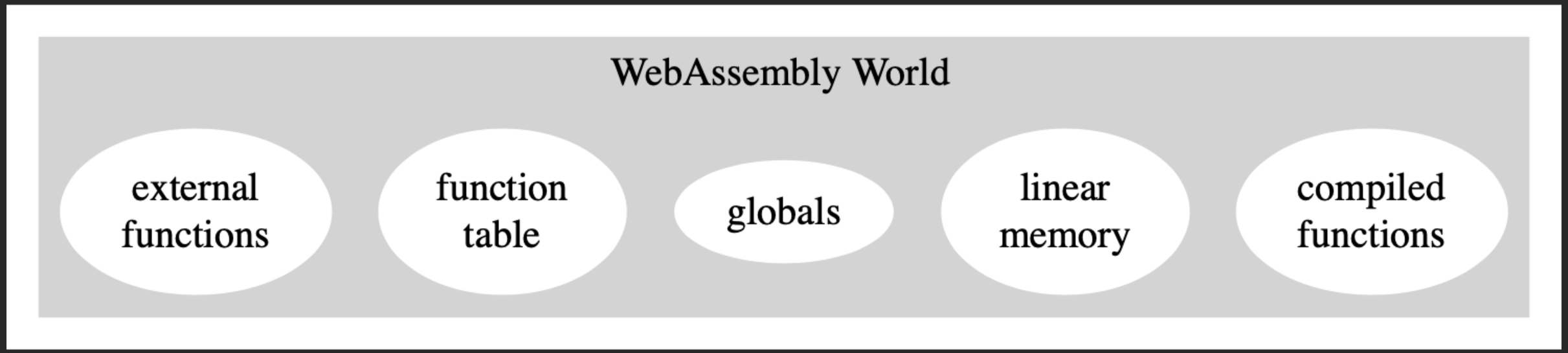
WebAssembly on the Server: How System Calls Work

Christine Dodrill
https://github.com/Xe
https://christine.website
@theprincessxena

What is WebAssembly?

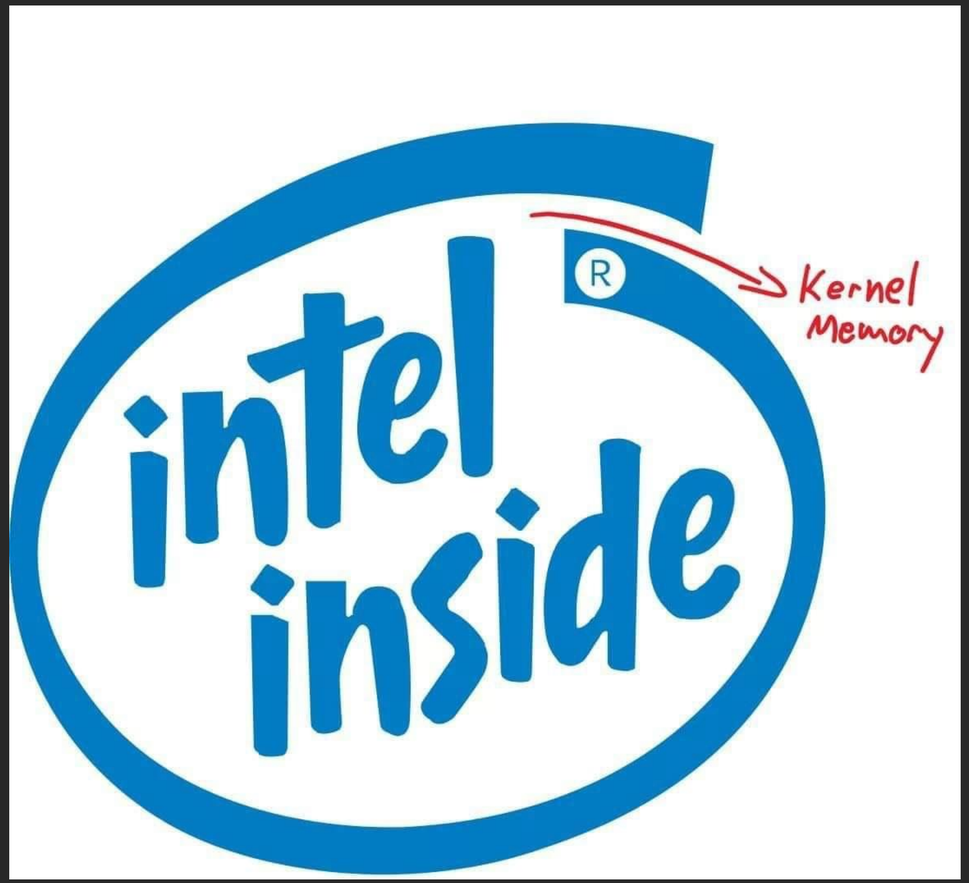
WebAssembly is a Virtual Machine format for the Web

- Like a CPU and supporting hardware
- Hardware independent
- Initially made for browsers
- Generic implementation


WebAssembly World
external functions
function table
globals
linear memory
compiled functions

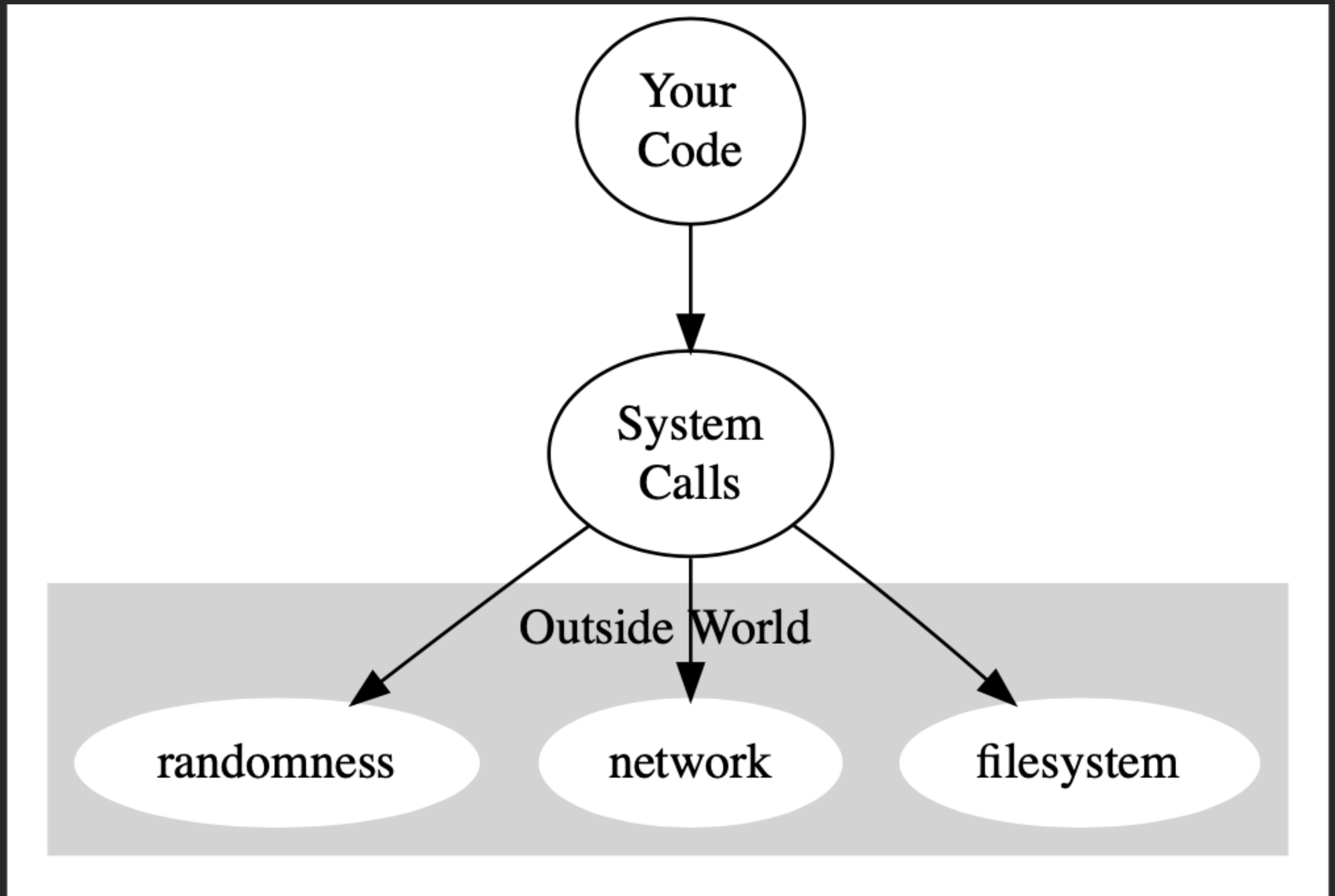
Why WebAssembly on the Server?

- It makes hardware less relevant
- No single vendor reliance
- Removes the OS



What are System Calls and why do they matter?

System Calls Enforce Abstractions To The Outside World



How are they implemented?

- The platform knows all
- Programs pass pointers to them

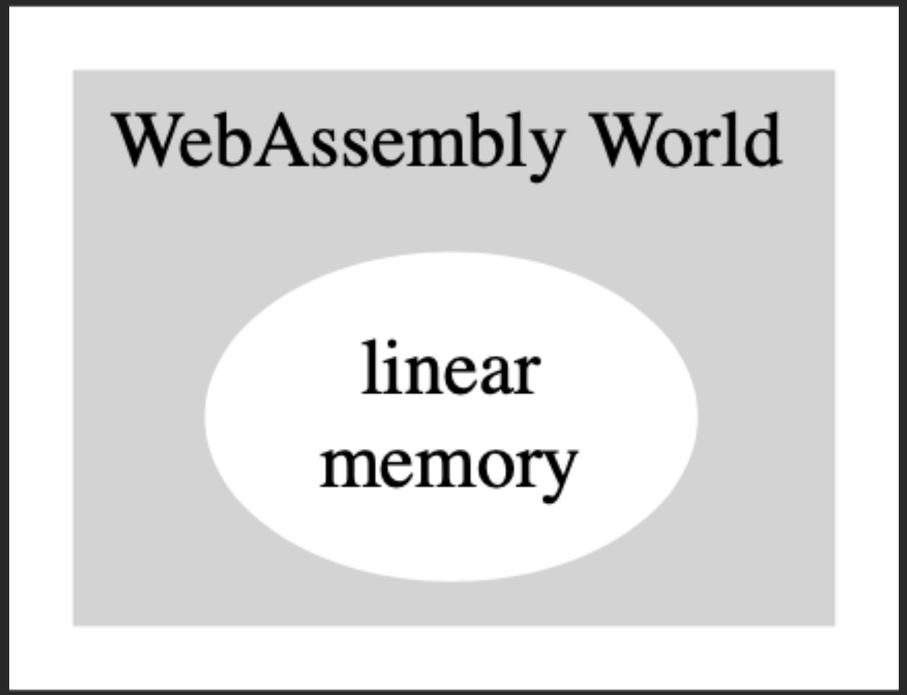
Why is This Relevant to WebAssembly?

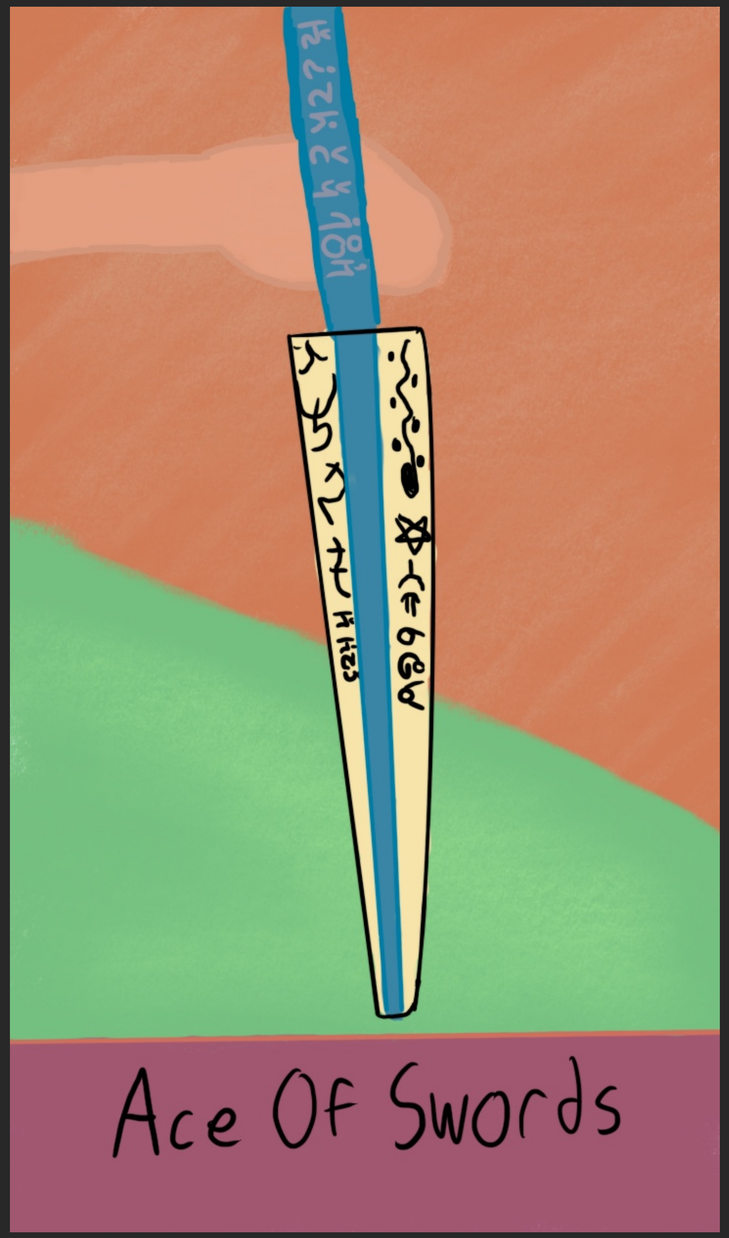
WebAssembly System Calls Out of The Box

What's a pointer in WebAssembly?

```
type wasm.VirtualMachine struct {
  // ...
  Memory []byte
  // ...
}
```

What's a pointer in WebAssembly?





Dagger: A Stepping Stone

Dagger

- A proof of concept system call API
- Simple implementation
- Intended for learning/teaching

Dagger Concepts

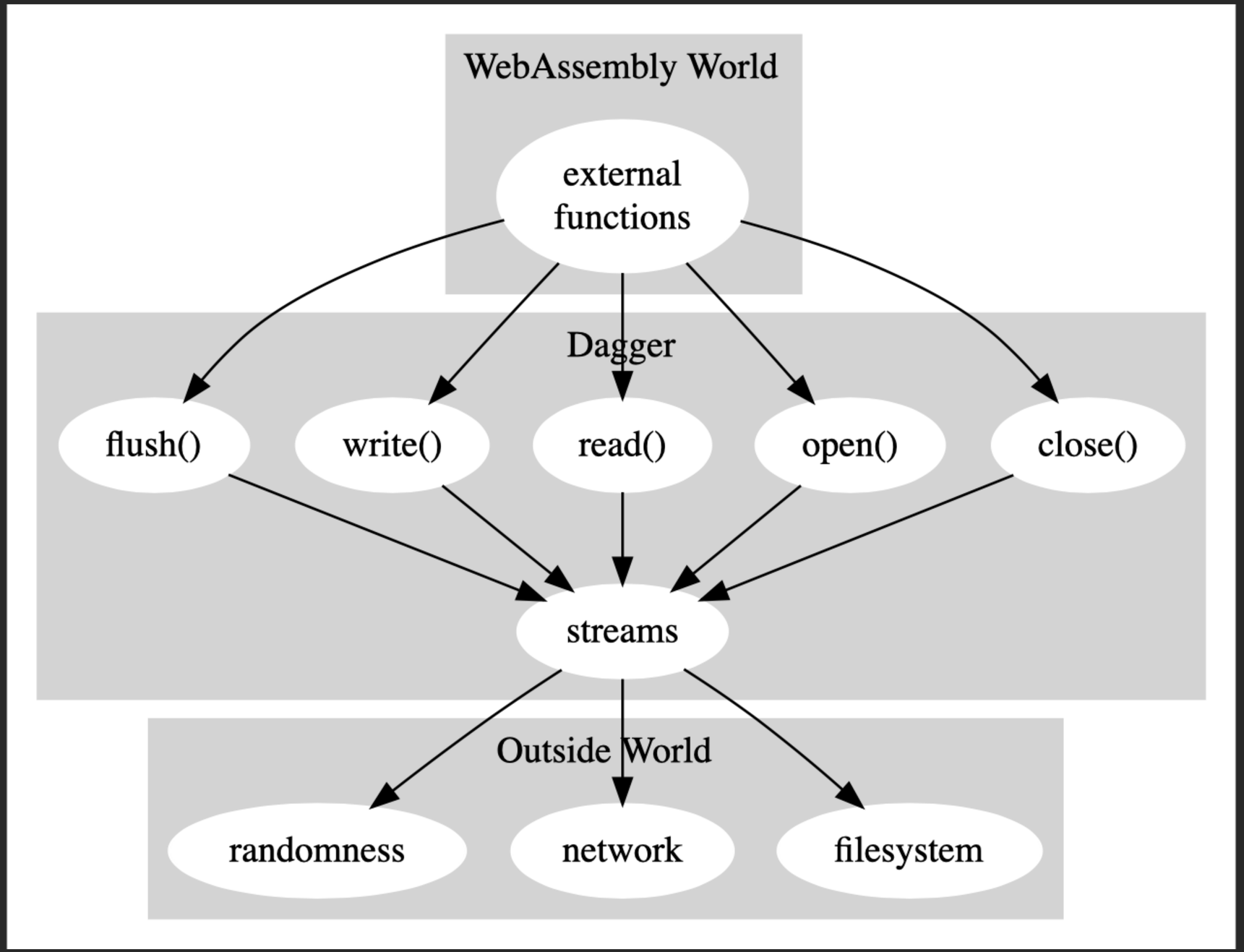
- Streams of data are the only interface
- NO MAGIC
- Simple doesn't have to mean useless

Dagger Process

- Has a slice of streams
- Stream descriptors are offsets into that slice

Dagger API

- Open
- Close
- Read
- Write
- Flush



```
int32 open(const char *surl, int32 surl_len);
```

- Opens a stream descriptor or errors
- URL scheme determines the stream target

int32 open(const char *surl, int32 surl_len);

- Stream kinds:
  - log://
  - file://
  - http:// or https://
  - random://
  - stdin:// or stdout:// or stderr://

int32 open(const char *surl, int32 surl_len);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var (
    furlPtr = uint32(args[0])
    fLen    = uint32(args[1])
    furl    = string(vm.Memory[furlPtr : furlPtr+fLen])
  )

  return p.OpenFile(furl)
}
```

int32 open(const char *surl, int32 surl_len);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var (
    furlPtr = uint32(args[0])
    fLen    = uint32(args[1])
    furl    = string(vm.Memory[furlPtr : furlPtr+fLen])
  )

  return p.OpenFile(furl)
}
```

int32 open(const char *surl, int32 surl_len);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var (
    furlPtr = uint32(args[0])
    fLen    = uint32(args[1])
    furl    = string(vm.Memory[furlPtr : furlPtr+fLen])
  )

  return p.OpenFile(furl)
}
```

int32 open(const char *surl, int32 surl_len);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var (
    furlPtr = uint32(args[0])
    fLen    = uint32(args[1])
    furl    = string(vm.Memory[furlPtr : furlPtr+fLen])
  )

  return p.OpenFile(furl)
}
```

int32 close(int32 sd);

- Closes a stream
- Errors should not normally happen

int32 close(int32 sd);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var desc = uint32(args[0])

  return p.CloseFile(desc)
}
```

int32 close(int32 sd);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var desc = uint32(args[0])

  return p.CloseFile(desc)
}
```

int32 close(int32 sd);

```
func (p *Process) open(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var desc = uint32(args[0])

  return p.CloseFile(desc)
}
```

int32 read(int32 sd, void *data, int32 data_len);

- Reads up to data_len bytes
- Returns the number of bytes read

int32 read(int32 sd, void *data, int32 data_len);

```
func (p *P) read(vm *exec.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var (
    desc = uint32(args[0])
    ptr  = uint32(args[1])
    len  = uint32(args[2])
    buf  = make([]byte, len)
    ret  = p.ReadFile(desc, buf)
  )
  p.CopyRam(ptr, buf)
  return ret
}
```

int32 read(int32 sd, void *data, int32 data_len);

```
// in read function ...
  var (
    desc = uint32(args[0])
    ptr  = uint32(args[1])
    len  = uint32(args[2])
    buf  = make([]byte, len)
  )
// ...
```

int32 read(int32 sd, void *data, int32 data_len);

```
// in read function ...
  var (
    desc = uint32(args[0])
    ptr  = uint32(args[1])
    len  = uint32(args[2])
    buf  = make([]byte, len)
  )
// ...
```

int32 read(int32 sd, void *data, int32 data_len);

```
// in read function ...
  var (
    ret  = p.ReadFile(desc, buf)
  )
  p.CopyRam(ptr, buf)
  return ret
}
```

int32 read(int32 sd, void *data, int32 data_len);

```
// in read function ...
  var (
    ret  = p.ReadFile(desc, buf)
  )
  p.CopyRam(ptr, buf)
  return ret
}
```

int32 write(int32 sd, void *data, int32 data_len);

- Copies data–data+data_len to the stream
- Returns number of bytes written

int32 write(int32 sd, void *data, int32 data_len);

```
func (p *P) Write(vm *exec.VirtualMachine) int32 {
  var (
    args = vm.GetCurrentFrame().Locals
    fd   = uint32(args[0])
    ptr = f.Locals[1]
    len = f.Locals[2]
    mem = vm.Memory[int(ptr):int(ptr+len)]
  )

   return p.WriteFD(fd, mem)
}
```

int32 write(int32 sd, void *data, int32 data_len);

```
func (p *P) Write(vm *exec.VirtualMachine) int32 {
  var (
    args = vm.GetCurrentFrame().Locals
    fd   = uint32(args[0])
    ptr = f.Locals[1]
    len = f.Locals[2]
  )
// ...
```

int32 write(int32 sd, void *data, int32 data_len);

```
func (p *P) Write(vm *exec.VirtualMachine) int32 {
  var (
    // ...
    mem = vm.Memory[int(ptr):int(ptr+len)]
  )

  return p.WriteFD(fd, mem)
}
```

int32 write(int32 sd, void *data, int32 data_len);

```
func (p *P) Write(vm *exec.VirtualMachine) int32 {
  var (
    // ...
    mem = vm.Memory[int(ptr):int(ptr+len)]
  )

  return p.WriteFD(fd, mem)
}
```

int32 flush(int32 sd);

- Flushes any intermediately unwritten bytes
- Mostly used for the HTTP client

int32 flush(int32 sd);

```
func (p *P) flush(vm *wasm.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var desc = uint32(args[0])
  return p.Flush(desc)
}
```

int32 flush(int32 sd);

```
func (p *P) flush(vm *wasm.VirtualMachine) int32 {
  args := vm.GetCurrentFrame().Locals
  var desc = uint32(args[0])
  return p.Flush(desc)
}
```

Hello World (Zig)

```
fn main() !void {
  const out = try Stream.open("stdout://");
  const msg = "Hello, world!\n";
  const ign = try out.write(&msg, msg.len);
  try out.close();
}
```

Hello World (Zig)

```
fn main() !void {
    const out = try Stream.open("stdout://");
    const msg = "Hello, world!\n";
    const ign = try out.write(&msg, msg.len);
    try out.close();
}
```

Hello World (Zig)

```
fn main() !void {
    const out = try Stream.open("stdout://");
    const msg = "Hello, world!\n";
    const ign = try out.write(&msg, msg.len);
    try out.close();
}
```

Hello World (Zig)

```
fn main() !void {
    const out = try Stream.open("stdout://");
    const msg = "Hello, world!\n";
    const ign = try out.write(&msg, msg.len);
    try out.close();
}
```

Hello World (Zig)

```
$ dagger dagger_hello.wasm
Hello, world!
$
```

Future Plans for Dagger

- "Control streams" for meta-operations
- More stream kinds

What This Can Build To

- Functions as a service backend
- Handling events
- Distributed computing
- Transactional computing

What You Can Do

- Play with the code
- Implement this from scratch
- CGI-Gopher?

Got Questions?

- Time is limited on stage
- Please email, tweet or ask me in person after the talk
- I'm happy to go into detail

GReeTZ

- Brian Ketelsen
- Andrew Kelly
- Eric McClure
- Faith Alderson
- My coworkers at Lightspeed
- Andrei Tudor Călin
- Elliot Speck
- Justin Clift
- Terry A. Davis
- Ayke van Laethem
- The people of #zig
- Vladimir Pouzanov
- https://pony.dev on Discord
- The WebAssembly Team
- You, for coming to/watching this talk

Follow my progress on GitHub

https://github.com/Xe/olin