Rust-embedding WebAssembly for scripting

Frank Rehberger
fr@frehberg.com
Software Consultant embedded & security

What is Rust?

System programming language, addressing

- Concurrency
- Memory safety
- Performance

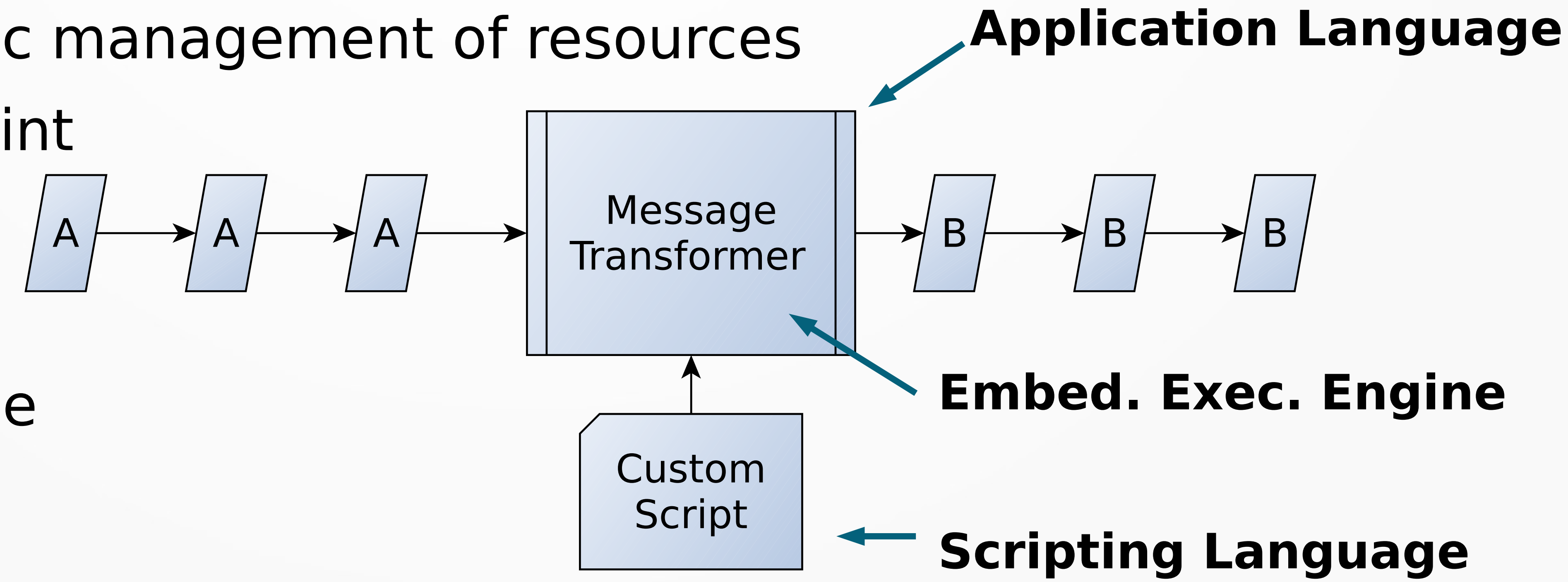
Specific: Memory management

- Deterministic management of resources (no GC)
- Favors stack allocation
- No implicit boxing (heap allocation)

How to Script my Rust-App?

Assume we want a customizable message broker

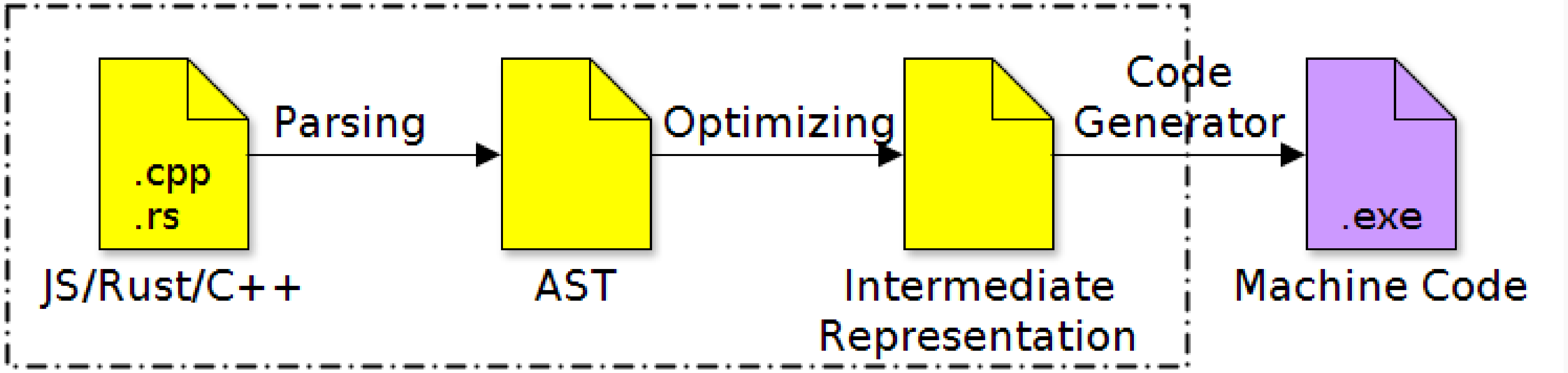
- Byte-level Manipulation
- Deterministic management of resources
- Small footprint
- Secure
- Safe
- Customizable



Analysis Embedded Execution Engines

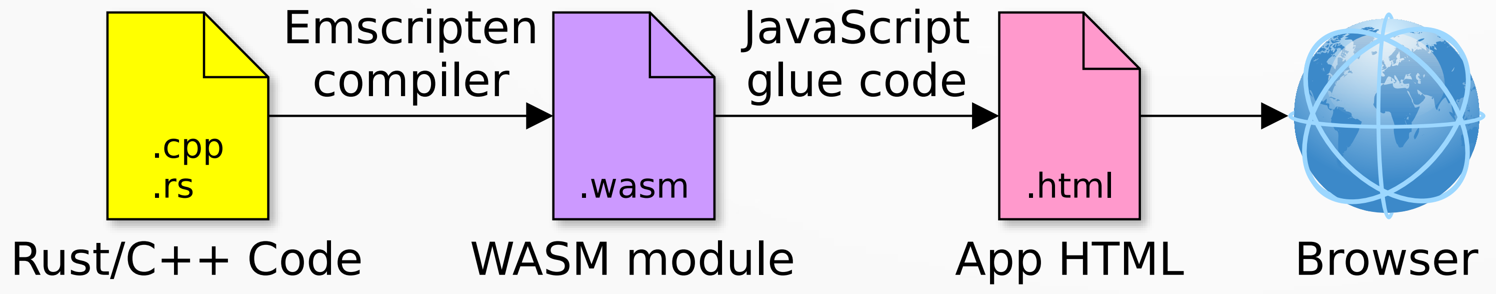
Interpreter/JIT causing Overhead/Complexity

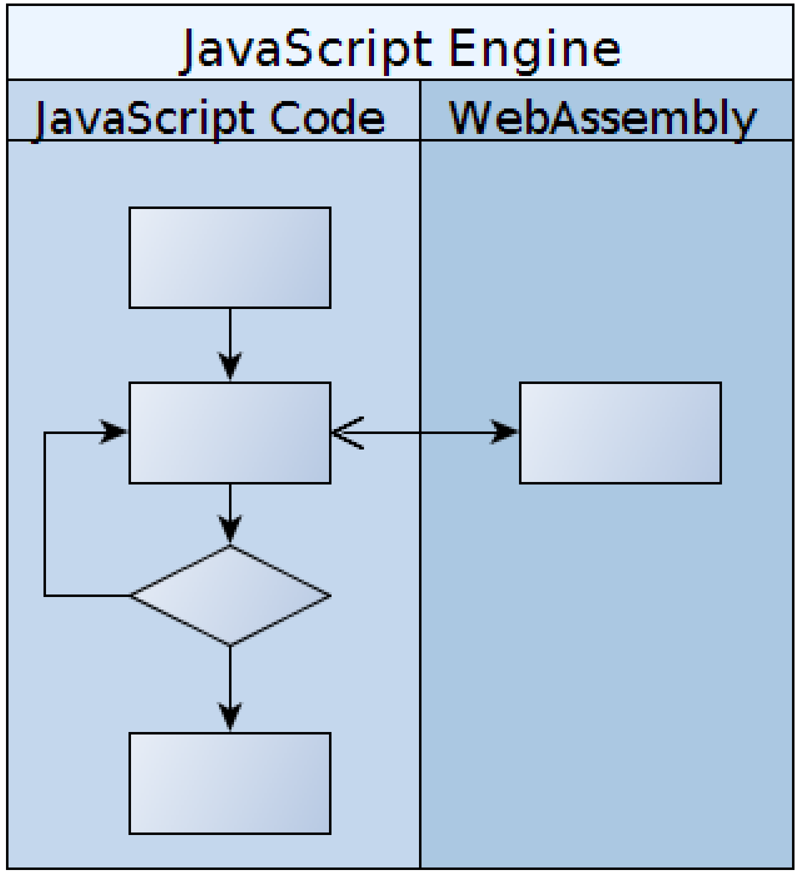
Type	Complexity/ Footprint	Dynamic Heap-Alloc	Exec Overhead	Determinist. Resource Management ?
Intepreted [Lua, [Python, JavasScript]	●	⬤	⬤	NO
JIT-Compiled [JavaScript V8]	⬤	⬤	●	NO
Stack Machine (Lw) [Forth, WebAssembly]	•	•	•	YES



WebAssembly [As Exec. Engine]

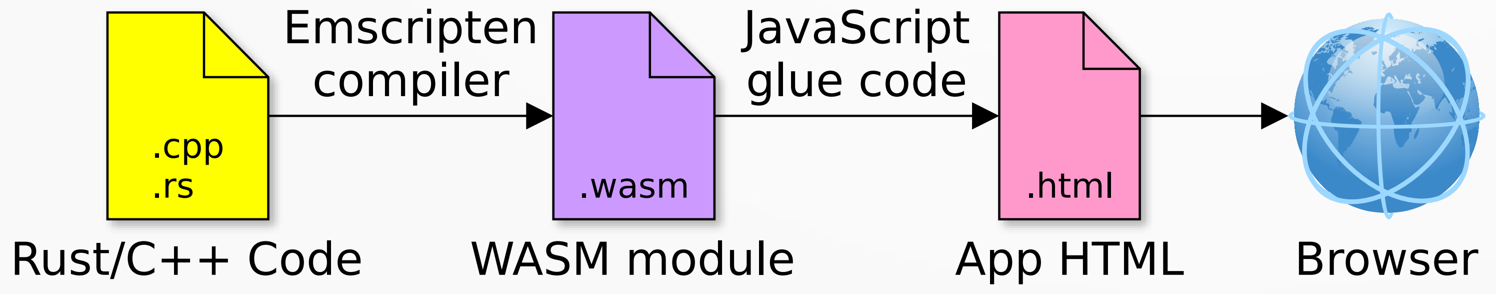
- Web-Standard - Announced May 2015
- Complements JavaScript, speeding up performance critical parts
- Portable Stack-Machine
- Binary/asm-like executable code
- Code Rust/C++, run in browser





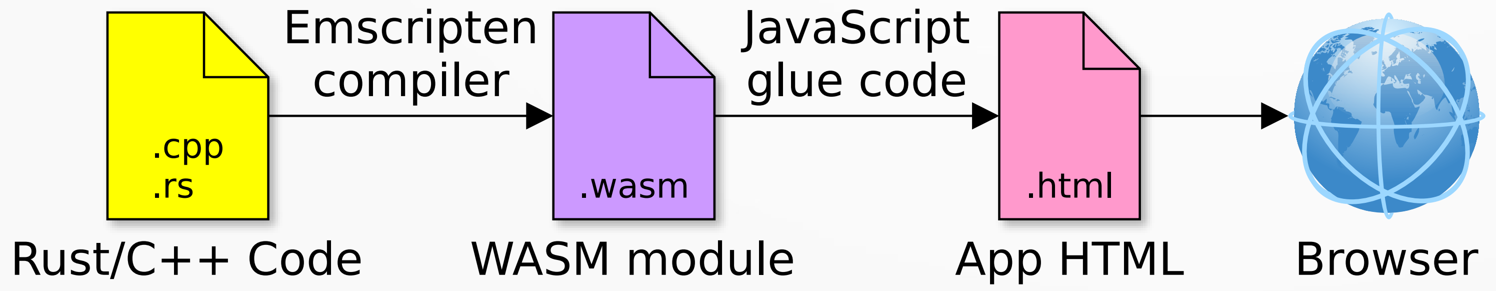
WebAssembly [As Exec. Engine]

- Build-Target for C/C++/Rust
- Binary format (WASM) & Text. S-expr. (WAST)
- Native scalar-types i32, i64, f32, f64
- Less than 256 instructions, easy to map onto HW
- Sandboxed, Linear memory, Tables, boundchecks
- Pointers are indeces into the linear memory
- Speed: 1.2x of native code (JIT-compiled)

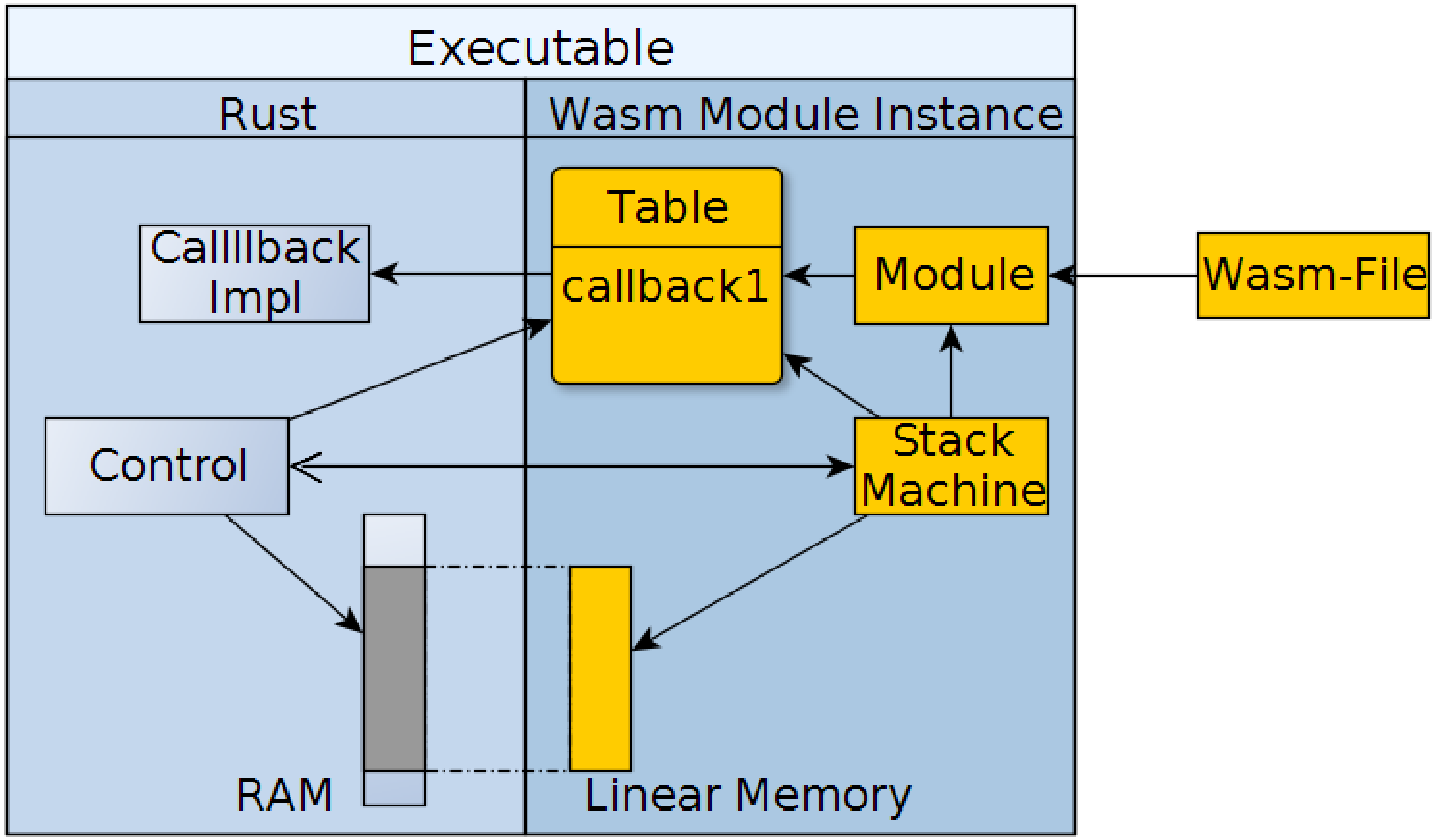


WebAssembly [As Exec. Engine]

- Using Interpreter Wasmi
  https://github.com/pepyakin/wasmi
- Native Rust implementation
- Just 6000 lines of code



WebAssembly [As Exec. Engine]



WebAssembly [Rust Source]

```
#[no_mangle]
pub extern "C" fn transform(arlen: i32, ar: &mut [u8]) -> i32 {
    const IVAL: u8 = 'i' as u8;
    const OVAL: u8 = 'o' as u8;
    for i in 0..arlen as usize { if ar[i] == IVAL { ar[i] = OVAL; } }
    return arlen;
}
```

```
cargo +nightly build --release \
--target wasm32-unknown-unknown
```

Producing file “transform.wasm” of size 57KB

Name mangling in module **transform**

WebAssembly [C/C++ Source]

```
...
extern "C" {
int32_t
EMSCRIPTEN_KEEPALIVE transform(const int32_t arlen, uint8_t *ar)  {
   const char IVAL = 'i';
   const char OVAL = 'o';
   for (int i=0; i<arlen; ++i) { if (ar[i]==IVAL) { ar[i]=OVAL; } }
   return arlen;
}
} // extern "C"
```

```
emcc transform.cpp -v -O3 -s ONLY_MY_CODE=1 -s \
            WASM=1 -s SIDE_MODULE=1 -o transform.wasm
```

Producing file “transform.wasm” of size 296 Byte

Name mangling in module **_transform**

WebAssembly [wasm/wast formats]

```
0061736d01000000000c0664796c696e
6b8080c00200018a808080000260027f
7f017f60000002c1808080000403656e
760a6d656d6f727942617365037f0003
656e76066d656d6f7279020080020365
6e76057461626c650170000003656e76
097461626c6542617365037f00038480
808000030001010068b80808000027f01
41000b7f0141000b07a380808000020a
5f7472616e73666f726d0000125f5f70
6f73745f696e7374616e746961746500
0209818080800000ae98080800003c1
8080800001027f027f200041004a0440
410021020520000f0b0340200120026a
22032c000041e900460440200341ef00
3a00000b200241016a22022000470d00
0b20000b0b838080800000010b958080
80000002402300240223024180 80c002
6a240310010b0b
```

```
(module
 (type $0 (func (param i32 i32) (result i32)))
 (type $1 (func))
 (import "env" "memoryBase" (global $import$0 i32))
 (import "env" "memory" (memory $0 256))
 (import "env" "table" (table 0 anyfunc))
 (import "env" "tableBase" (global $import$3 i32))
 (global $global$0 (mut i32) (i32.const 0))
 (global $global$1 (mut i32) (i32.const 0))
 (export "_transform" (func $0))
 (export "__post_instantiate" (func $2))
 (func $0 (type $0) (param $var$0 i32) (param $var$1 i32) (result i32)
  (local $var$2 i32)
  (local $var$3 i32)
  (block $label$0 (result i32)
...
```

Credits

- Nik Volf: WebAssembly Interpreter https://github.com/paritytech/parity-wasm (Now https://pepyakin.github.io/wasmi/)
- Alex Crichton https://github.com/tokio-rs/tokio-core
- Carl Lerche: https://github.com/carllerche/mio/

Appendix

Rust/Wasm toolchain

Rust/Wasm Toolchain Installation

Wasm-Target-Arch not supported yet (1.23.0 stable), nightly required

- rustup toolchain install **nightly**
- rustup update
- rustup target add wasm32-unknown-unknown --toolchain nightly
- cargo install --git https://github.com/alexcrichton/wasm-gc
- cargo install wasm-nm
- In your project, change the crate type to cdylib in file Cargo.toml

```
[lib]
path = "src/lib.rs"
crate-type = ["cdylib"]
```