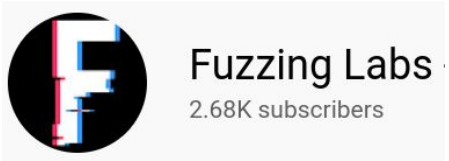
A Journey Into Fuzzing WebAssembly Virtual Machines

Patrick Ventuzelo

Patrick Ventuzelo (@Pat_Ventuzelo)

- Founder & CEO of **FuzzingLabs** | Senior **Security Researcher**
  - Fuzzing and vulnerability research
  - Development of security tools



- Training/Online courses
  - **Rust** Security Audit & Fuzzing
  - **Go** Security Audit & Fuzzing
  - **WebAssembly** Reversing & Analysis
  - Practical Web **Browser** Fuzzing

FUZZING LABS

- Main focus
  - **Fuzzing**, Vulnerability research
  - Rust, Golang, **WebAssembly**, **Browsers**
  - Blockchain Security, Smart contracts

- Previously speaker at:
  - OffensiveCon, REcon, RingZer0, ToorCon, hack.lu, NorthSec, FIRST, etc.

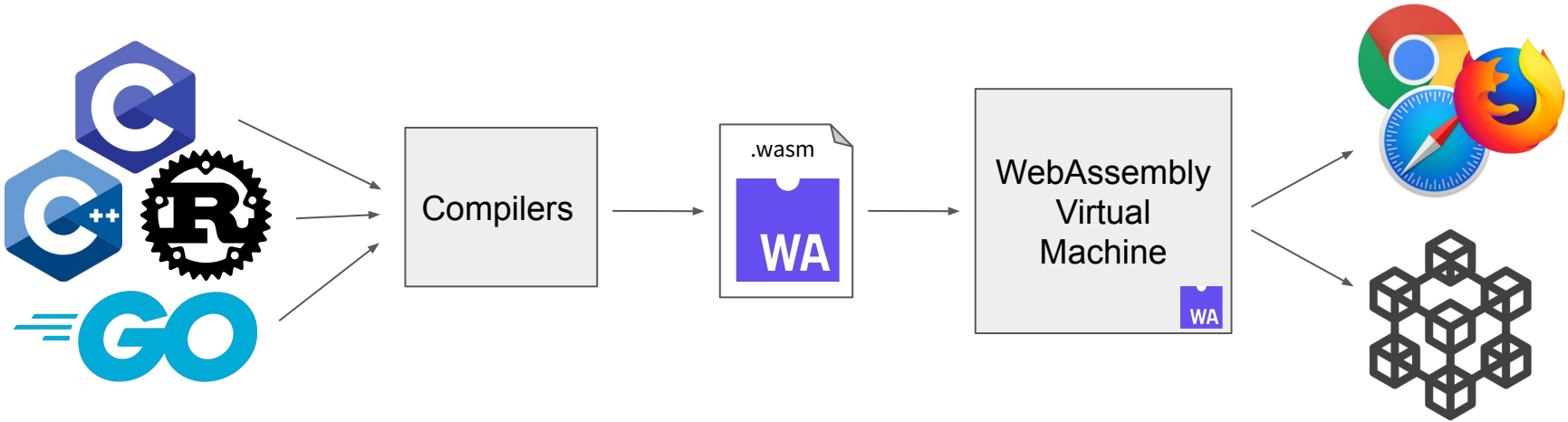
Introduction to WebAssembly

What is WebAssembly?

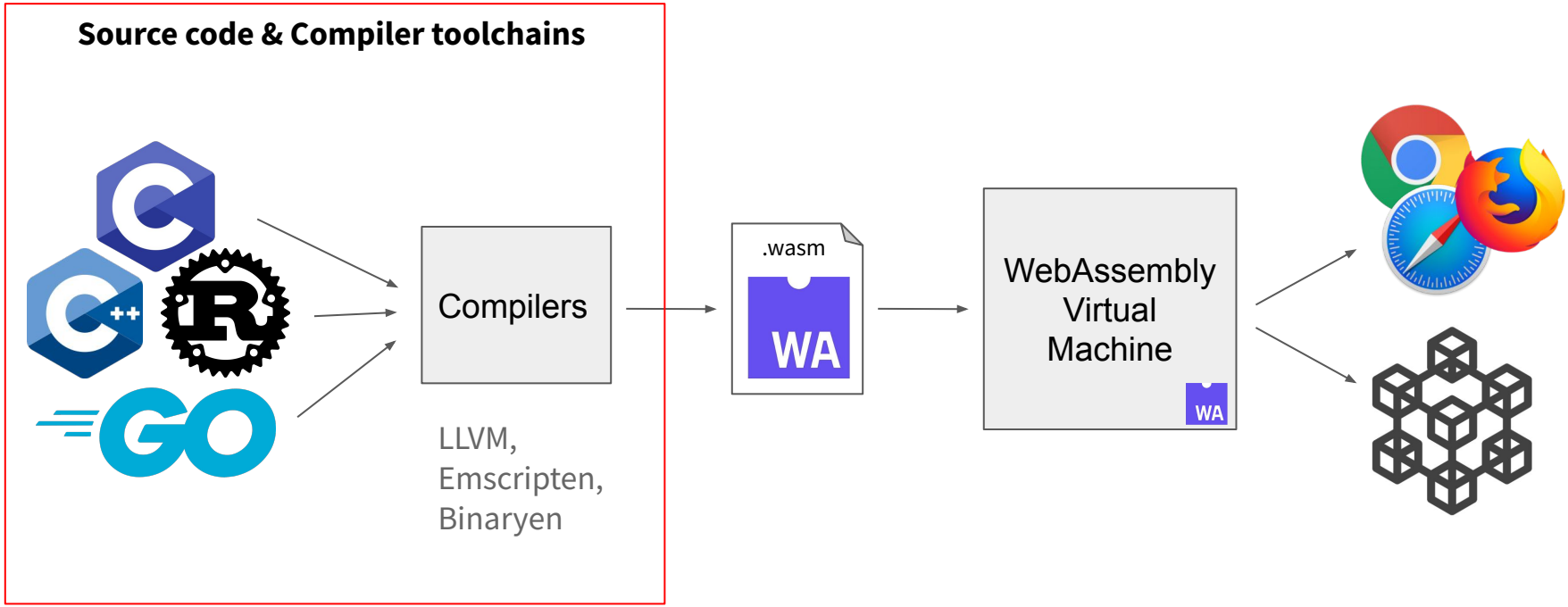
- **Binary** instruction format for a **stack-based virtual machine**
  - Low-level bytecode
  - Compilation target for C/C++/Rust/Go/etc.
- Generic evolution of NaCl & Asm.js
- W3C standard
- MVP 1.0 (March 2017), MVP 2.0 (2022/2023)
- **Natively supported in all major browsers**

- WebAssembly goals:
  - Be fast, **efficient**, and portable
  - Easily readable and **debuggable**
  - **Safe** (using sandboxed execution environment)
  - Open and **modulable**

How WebAssembly works?



Step 1: Compilation into WebAssembly module



WebAssembly Binary Format

C/C++

```c
int fib(int n)
{
  if (n == 0 || n == 1)
    return n;
  else
    return (fib(n-1) + fib(n-2));
}
```

Rust

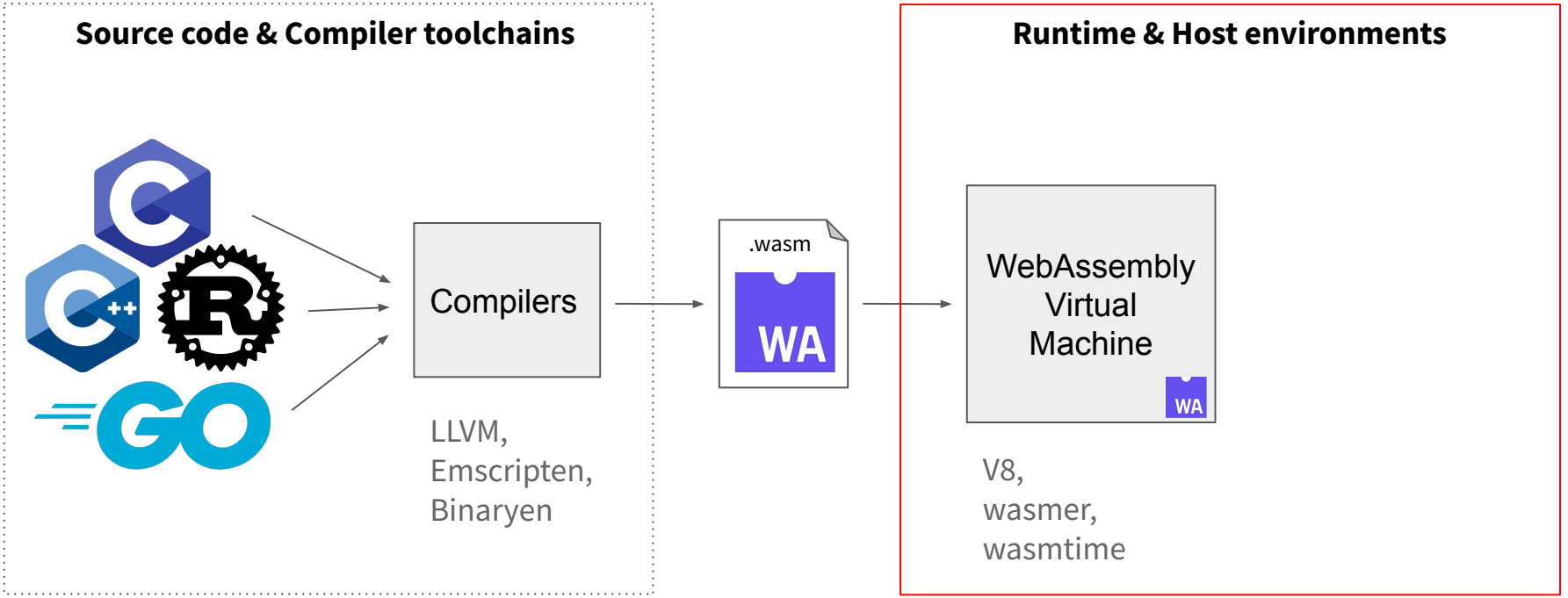
```rust
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

Compilation

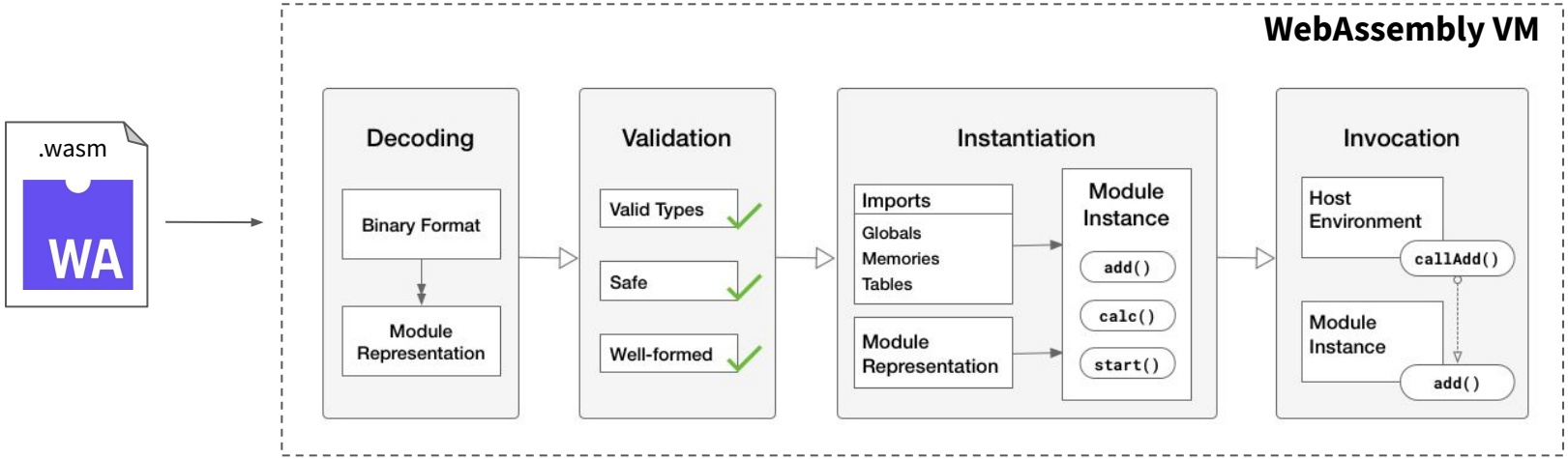
binary file (.wasm)

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8080 8000 0206 6d65
6d6f 7279 0200 0366
6962 0000 0aa7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 000f
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```

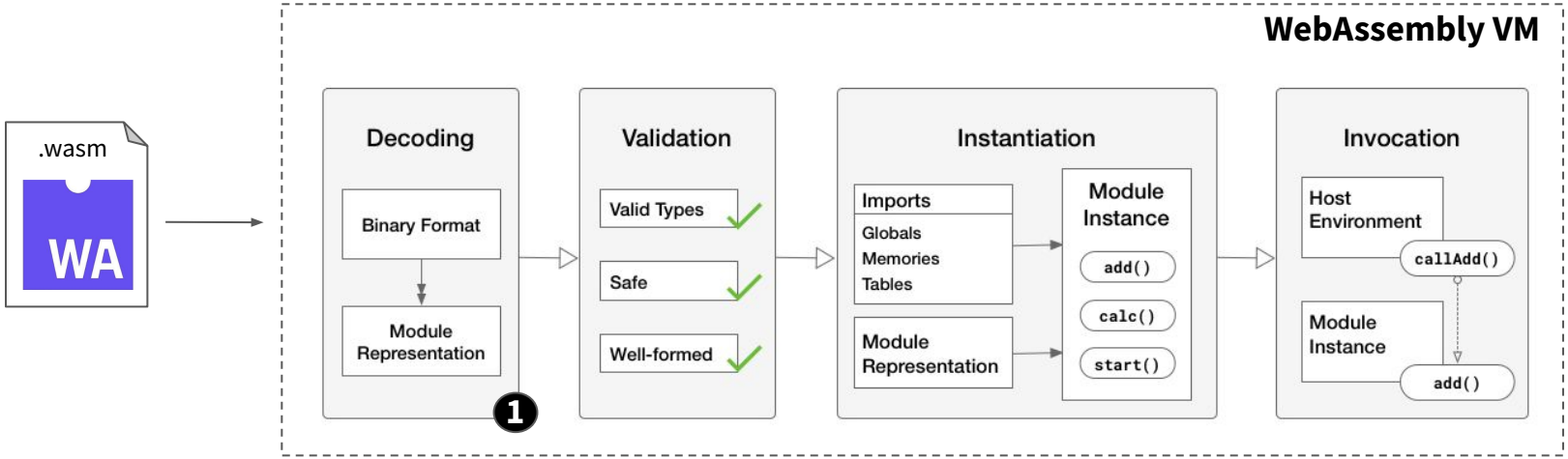
Step 2: Execution by the WebAssembly VM



WebAssembly VM - Execution stages

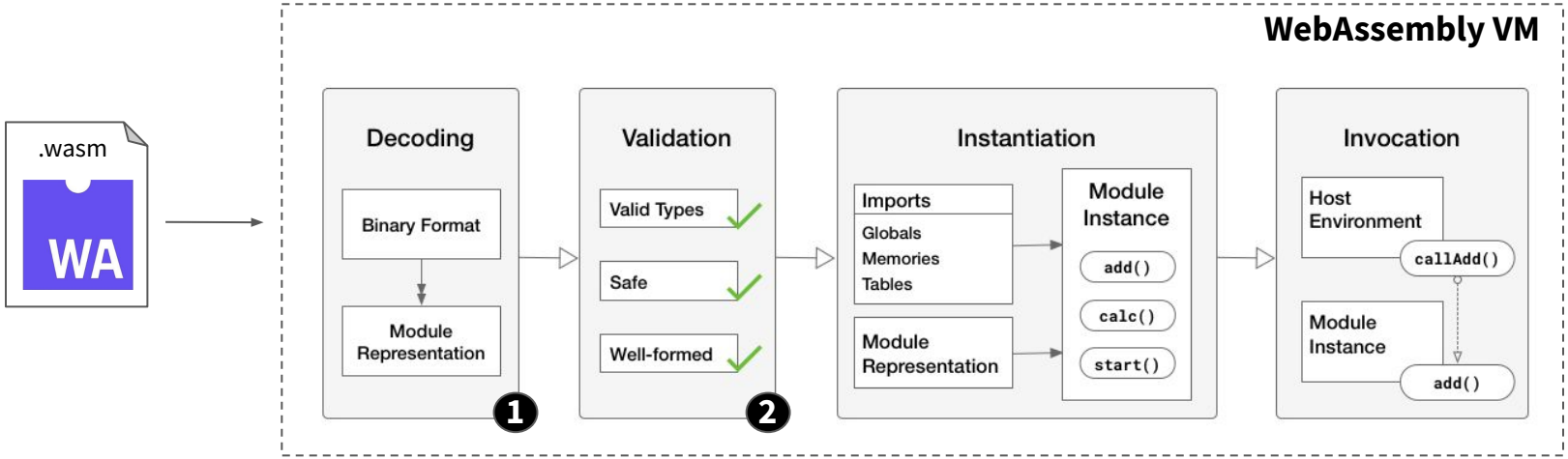


WebAssembly VM - Decoding/Parsing



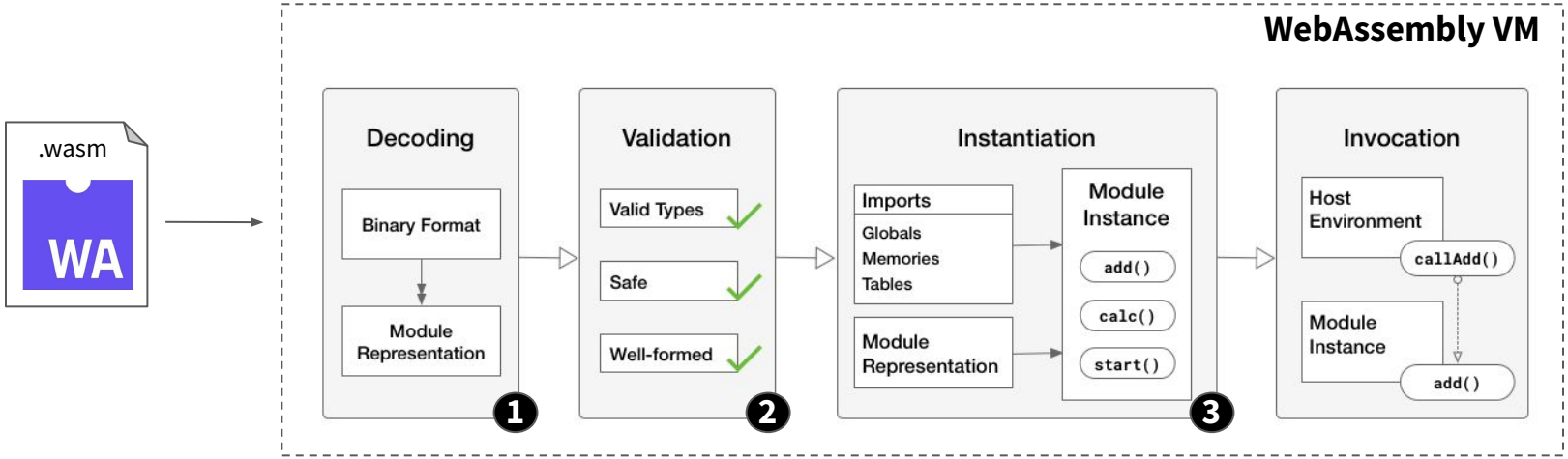
1. **Decoding/Parsing**: The binary format is parsed and converted into a module

WebAssembly VM - Validation



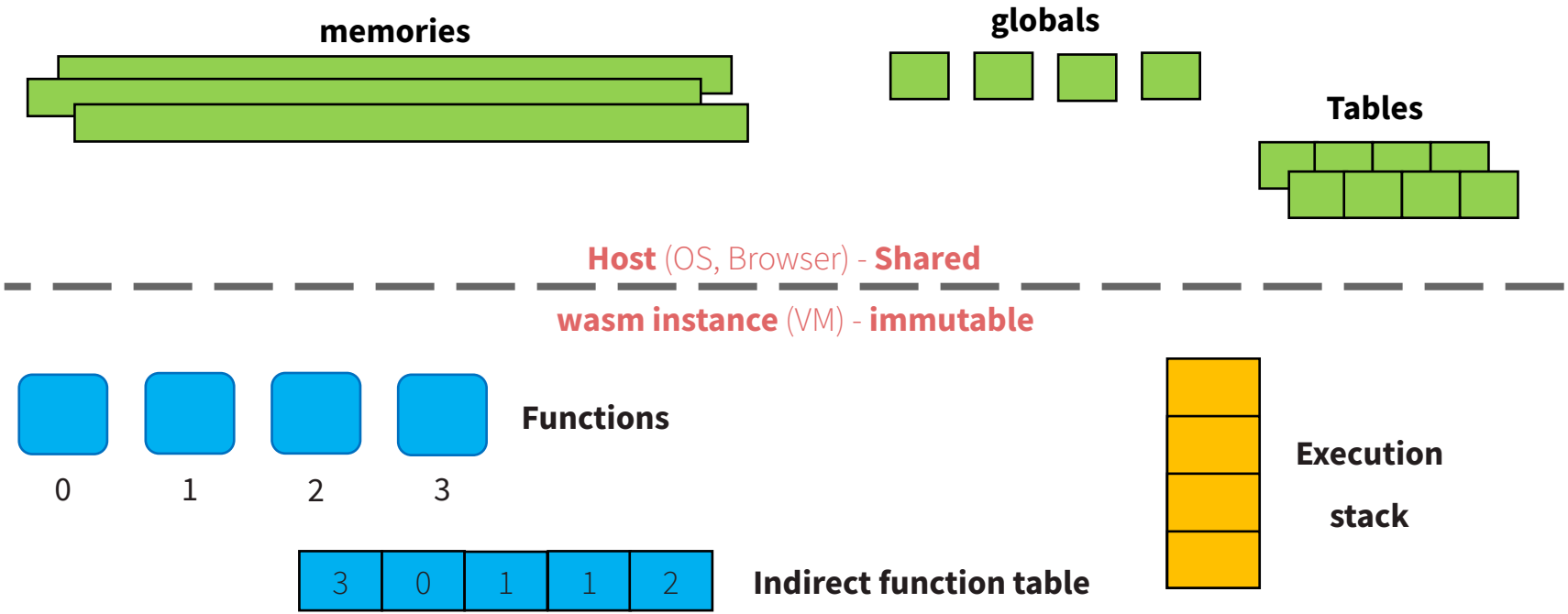
1. **Decoding/Parsing**: The binary format is parsed and converted into a module
2. **Validation**: The decoded module undergoes validation checks (such as type checking)

WebAssembly VM - Instantiation

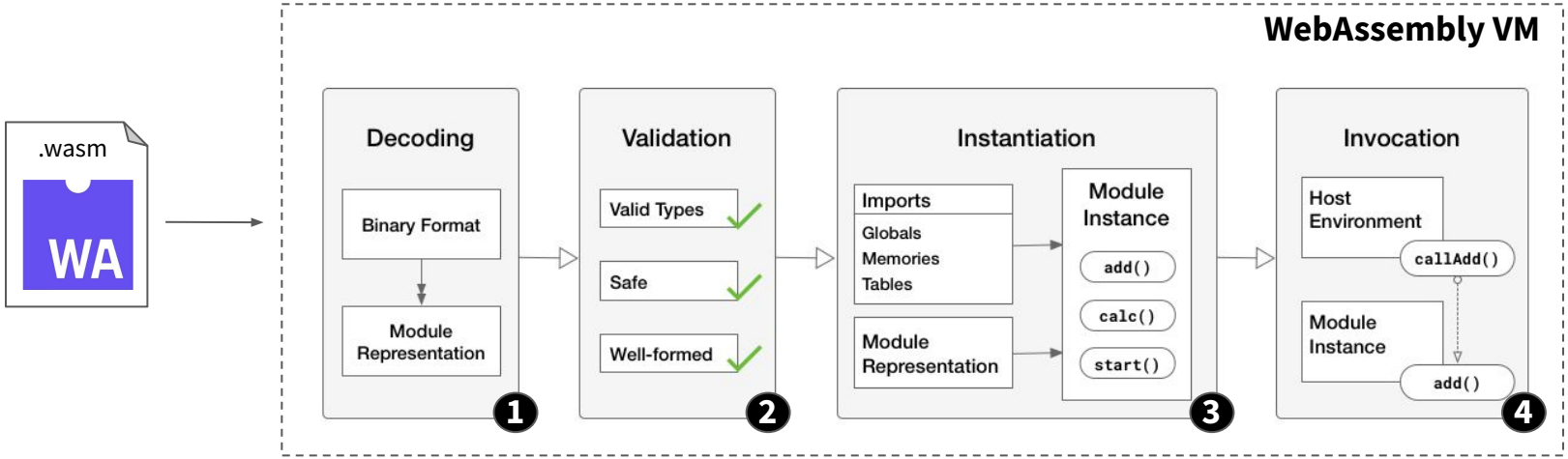


1. **Decoding/Parsing**: The binary format is parsed and converted into a module
2. **Validation**: The decoded module undergoes validation checks (such as type checking)
3. **Instantiation**: Creation of a module instance with all the context instantiated

WebAssembly VM - Instantiation

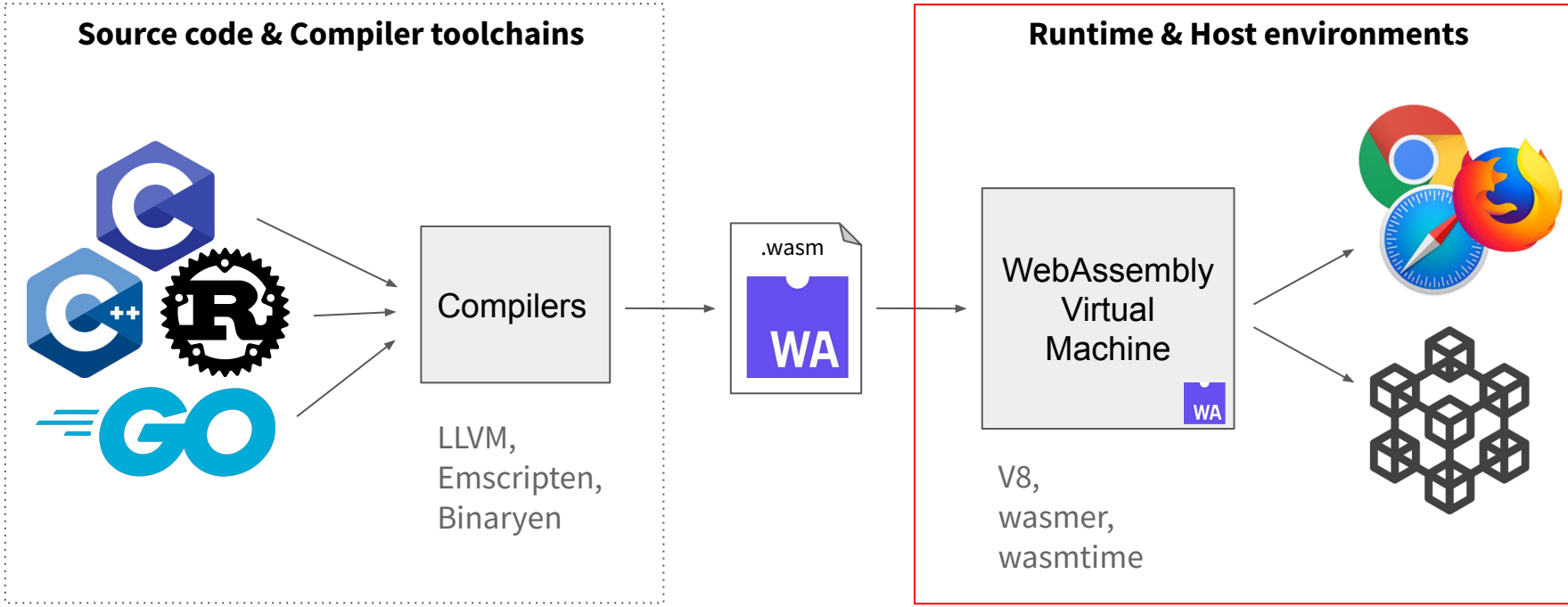


WebAssembly VM - Execution/Invocation



1. **Decoding/Parsing**: The binary format is parsed and converted into a module
2. **Validation**: The decoded module undergoes validation checks (such as type checking)
3. **Instantiation**: Creation of a module instance with all the context instantiated
4. **Execution/Invocation**: Exported functions are called by the host over the module instance

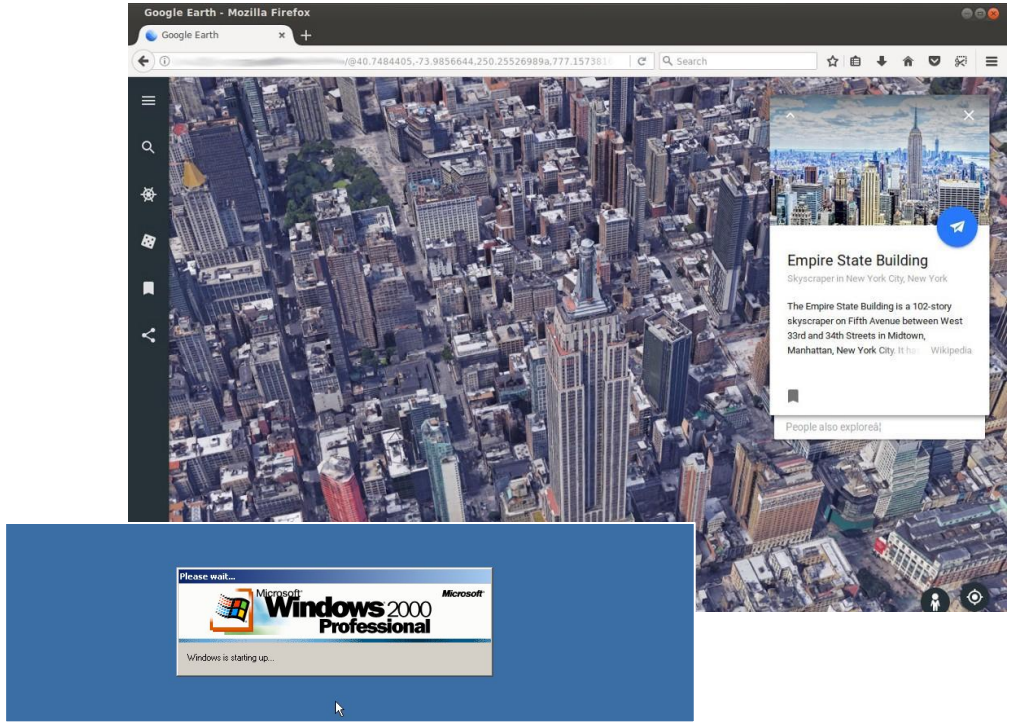
Step 2: Execution by the WebAssembly VM



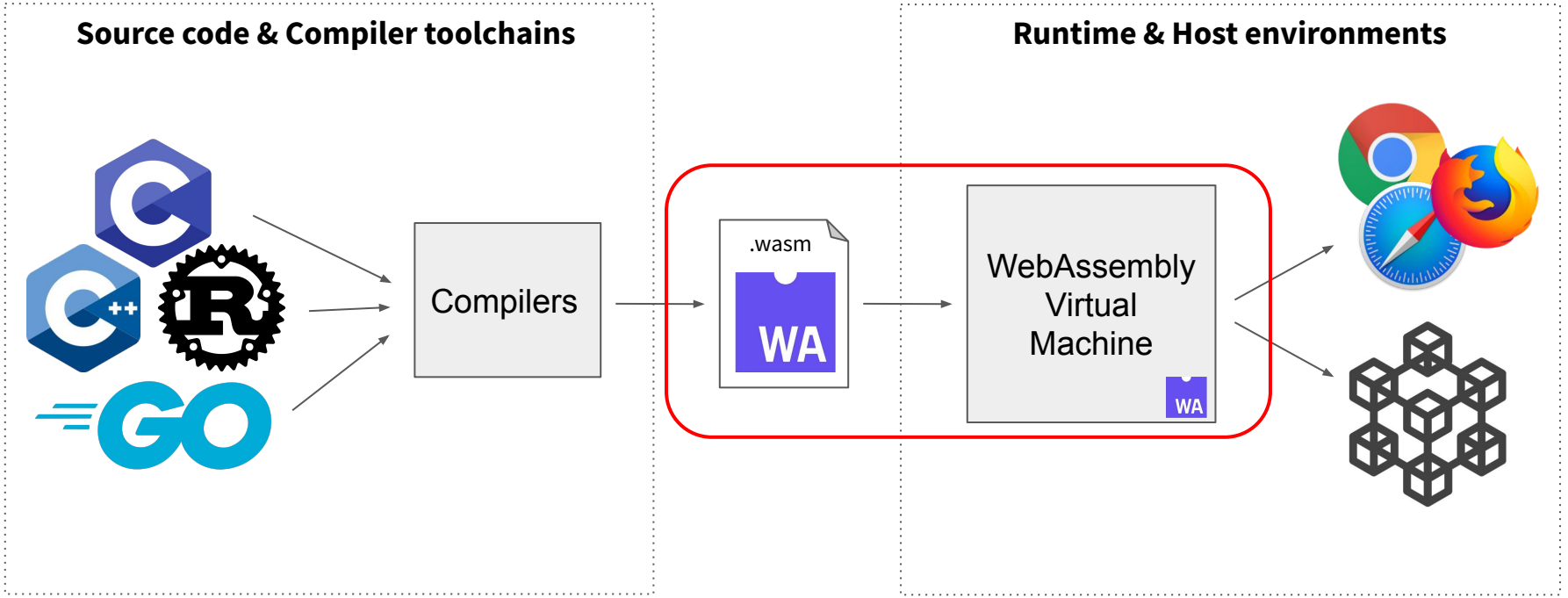
WebAssembly VM - Use-cases

- **Standalone VM** (server)
  - Edge computing
  - Back-end apps
    - Nodejs
  - Mobile & Desktop apps
  - IoT & Embedded OS
  - Blockchain
    - Polkadot, Substrate, Cosmos, NEAR
    - Spacemesh, Golem, EOS, DFINITY

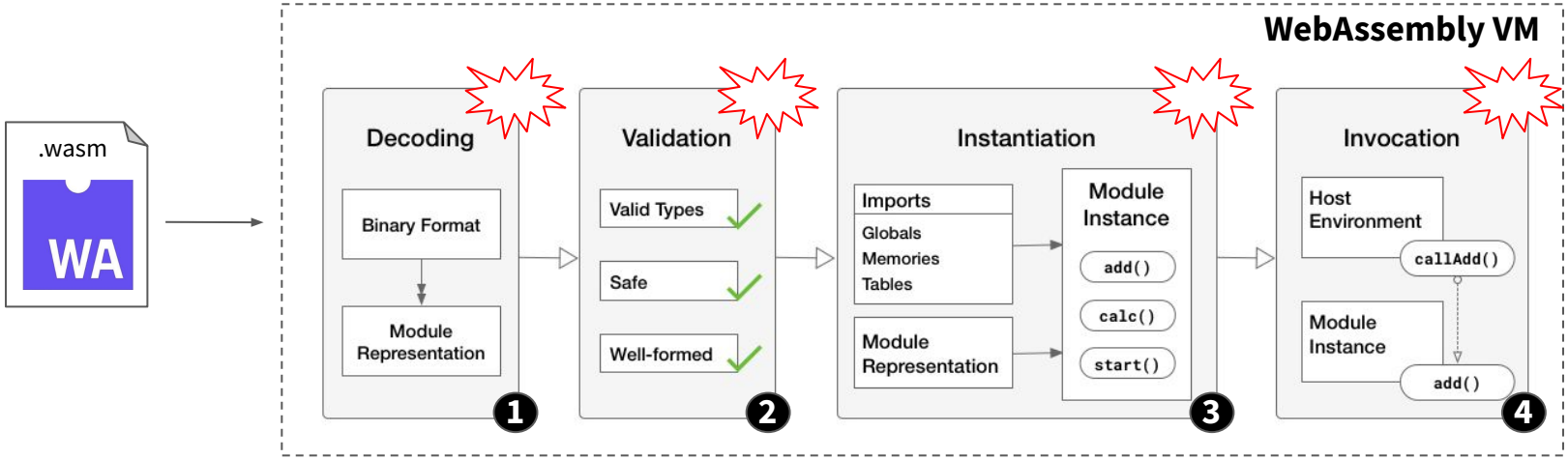
- **Browser** (client)
  - Video, Audio & Image processing
  - Videos Games
  - Complexe web apps
    - Autocad, Google Earth
    - Photoshop, Shopify, Figma
  - OS Emulation



Focus of this talk: WebAssembly VM

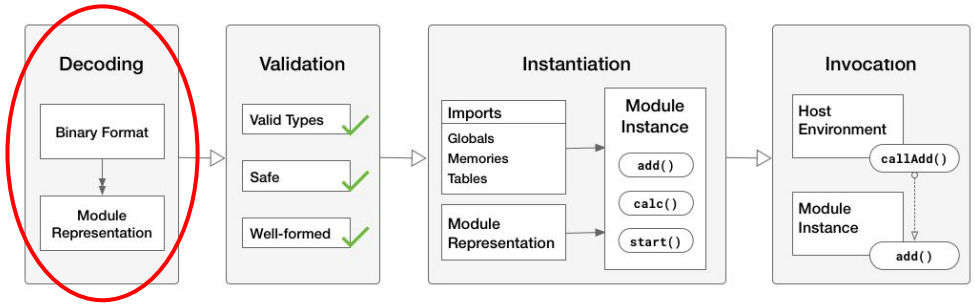


Goal: Find bugs on every stage on different VMs!



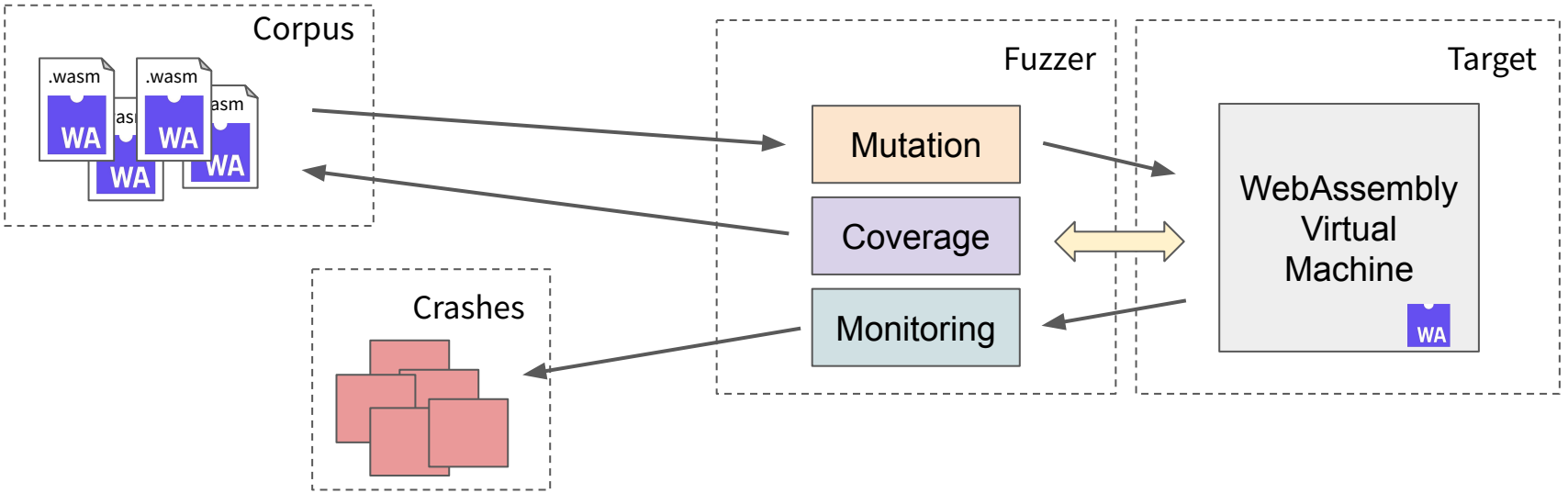
1. **Decoding/Parsing**: The binary format is parsed and converted into a module
2. **Validation**: The decoded module undergoes validation checks (such as type checking)
3. **Instantiation**: Creation of a module instance with all the context instantiated
4. **Execution/Invocation**: Exported functions are called by the host over the module instance

1. Coverage-guided fuzzing



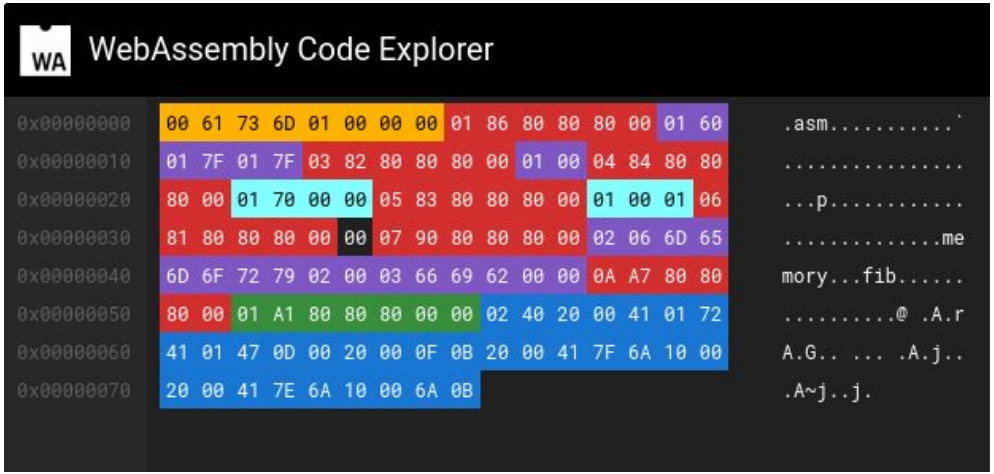
Fuzzing strategy: Coverage-guided fuzzing

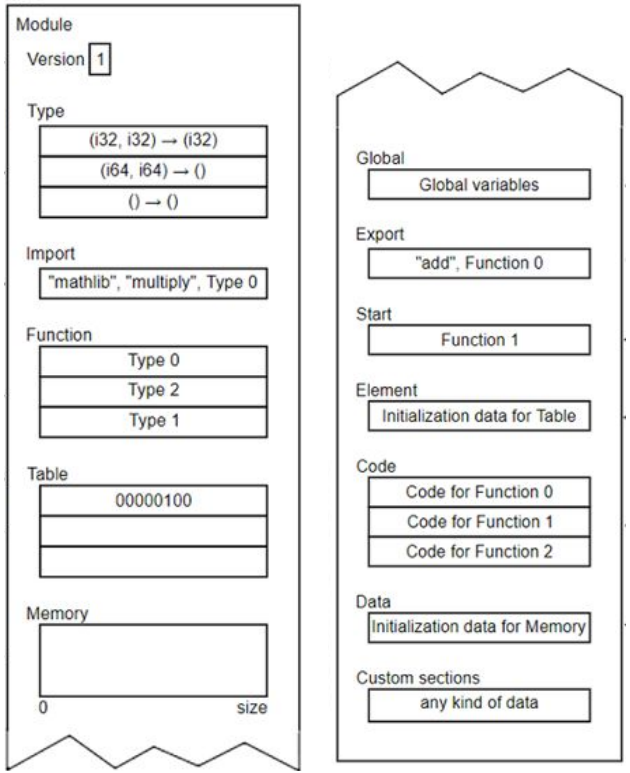
- **Coverage-guided** fuzzing
  - Observe how inputs are processed to **learn** which mutations are interesting.
  - Save inputs to be **re-used** and mutated in future iterations.



Input: WebAssembly Binary Format

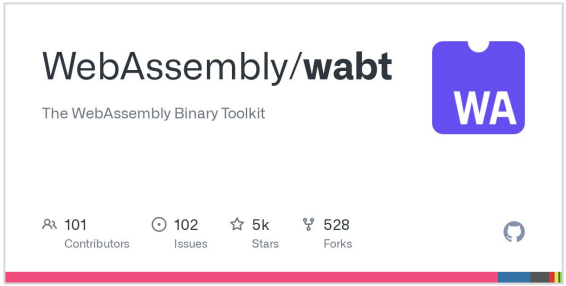
- Module structure
  - **Header**: magic number + version
  - 11 **Sections**: may appear at most once
  - 1 **custom** section: unlimited

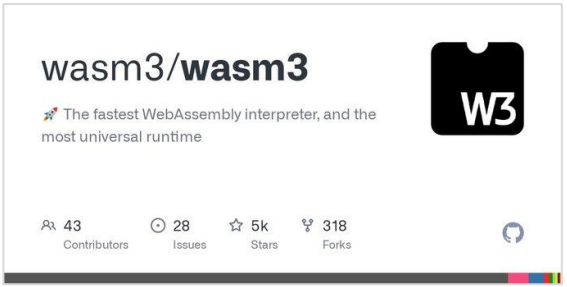


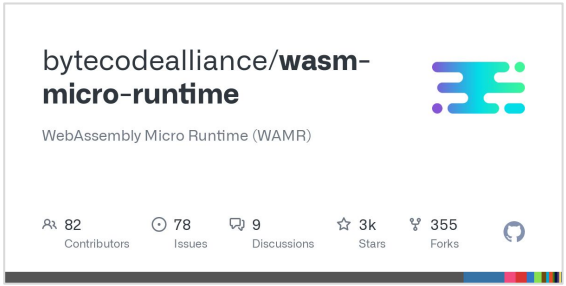


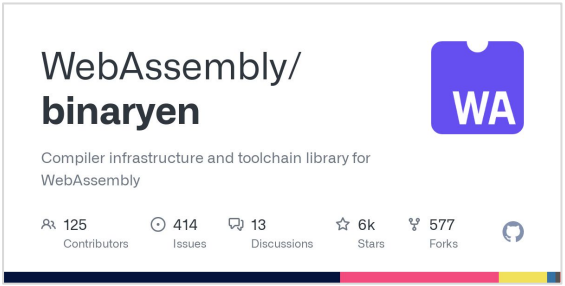
Targets: Standalone VMs & parsing libraries

- Targets (C/C++)
  - Binaryen: Compiler and **toolchain** libraries
  - WABT: The WebAssembly Binary **Toolkit**
  - Wasm3: WebAssembly **interpreter**
  - WAMR: WebAssembly Micro **Runtime**
  - WAC: WebAssembly **interpreter** in C
  - Radare2: Reverse engineering framework
  - Etc.



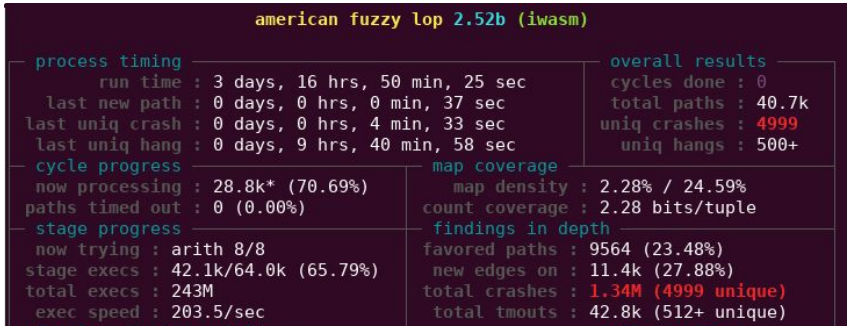


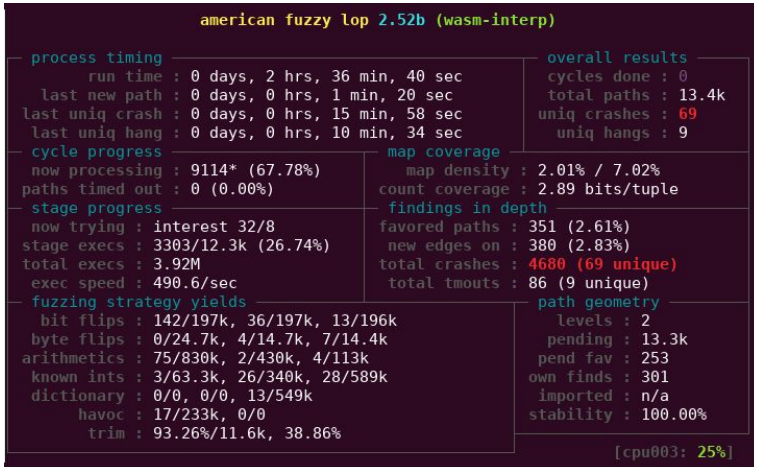




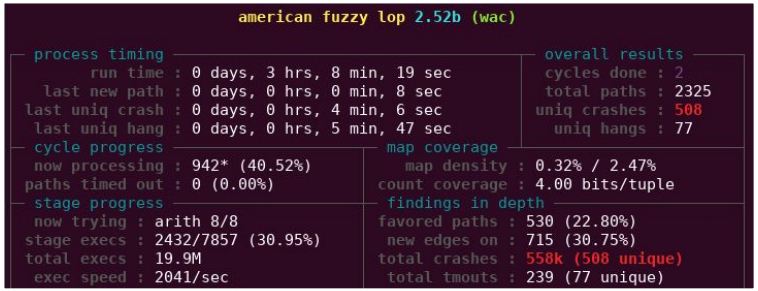
C/C++ Coverage-guided Fuzzing

- C/C++ Fuzzers
  - **AFL**: american fuzzy lop
  - **Honggfuzz**: Feedback-driven/evolutionary fuzzer
  - **AFL++**: AFL with community patches





- Complexity: **None**
  - Instrumentation using custom gcc/clang
  - Overwrite CC or CXX flags
  - **Prefered AFL++ instead of vanilla AFL**



Results: ~**46** bugs/vulnerabilities

- Binaryen
  - **Out-of-bound** read - issue
- WABT
  - **Assertion errors** - issue#1, issue#2, issue#3, issue#4
  - **Uncontrolled memory allocation** - issue
- WAMR
  - **Null pointer dereference** - issues (5)
  - **Heap out of bounds read** - issues (29)
  - **Assertion errors** - issue#1, issue#2
  - **Heap out of bounds write** - issue
  - **Segmentation fault** - issue
- Radare2
  - **Heap out of bounds read** - issue
  - **Heap out of bounds read** - issue

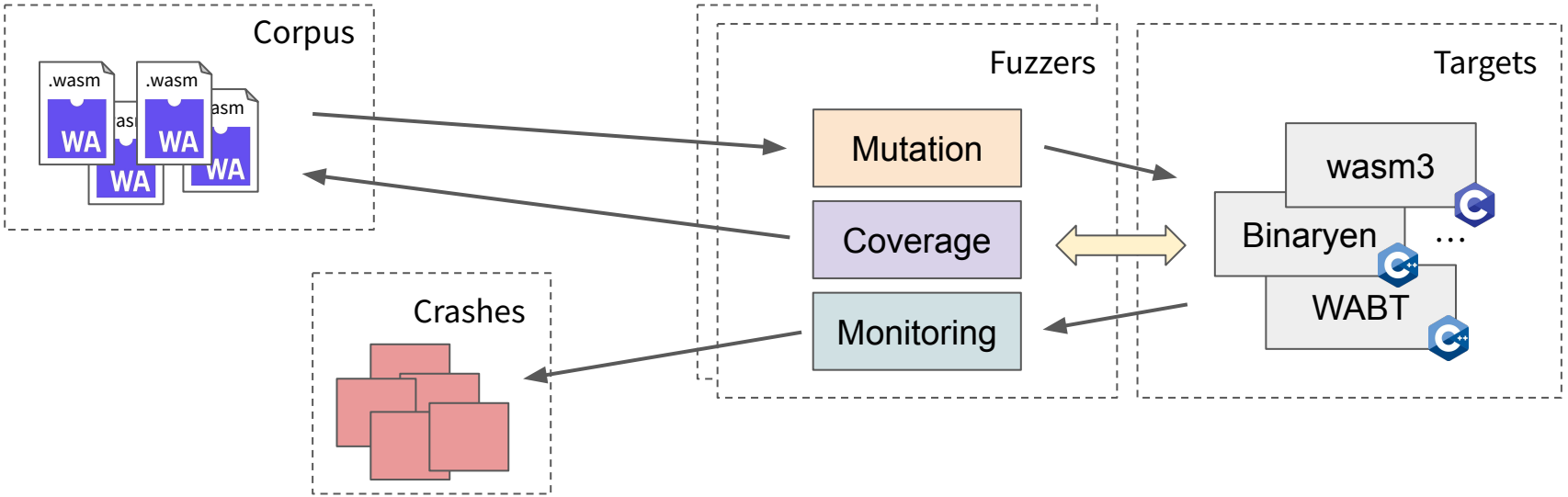
```
Invalid read of size 8
   at 0x11710E: wasm_interp_call_func_bytecode (wasm_interp.c:1180)
   by 0x11C6FC: wasm_interp_call_wasm (wasm_interp.c:2158)
   by 0x10D446: wasm_runtime_call_wasm (wasm_runtime.c:102)
   by 0x10C841: wasm_application_execute_main (wasm_application.c:109)
   by 0x10BAD7: app_instance_main (main.c:54)
   by 0x10C0EA: main (main.c:217)
 Address 0x18 is not stack'd, malloc'd or (recently) free'd
```

```
Program received signal SIGSEGV, Segmentation fault.
0x0000555555a9a02f in wasm::WasmBinaryBuilder::readImports() ()
(gdb) bt
#0  0x0000555555a9a02f in wasm::WasmBinaryBuilder::readImports() ()
#1  0x0000555555a9f918 in wasm::WasmBinaryBuilder::read() ()
#2  0x0000555555acd498 in wasm::ModuleReader::readBinaryData(std::vect
#3  0x0000555555acd921 in wasm::ModuleReader::readBinary(std::__cxx11:
#4  0x0000555555ace788 in wasm::ModuleReader::read(std::__cxx11::basic
#5  0x000055555573ead2 in main ()
```

```
==3759==ERROR: AddressSanitizer: heap-buffer-overflow
READ of size 1 at 0x611000016300 thread T0
```

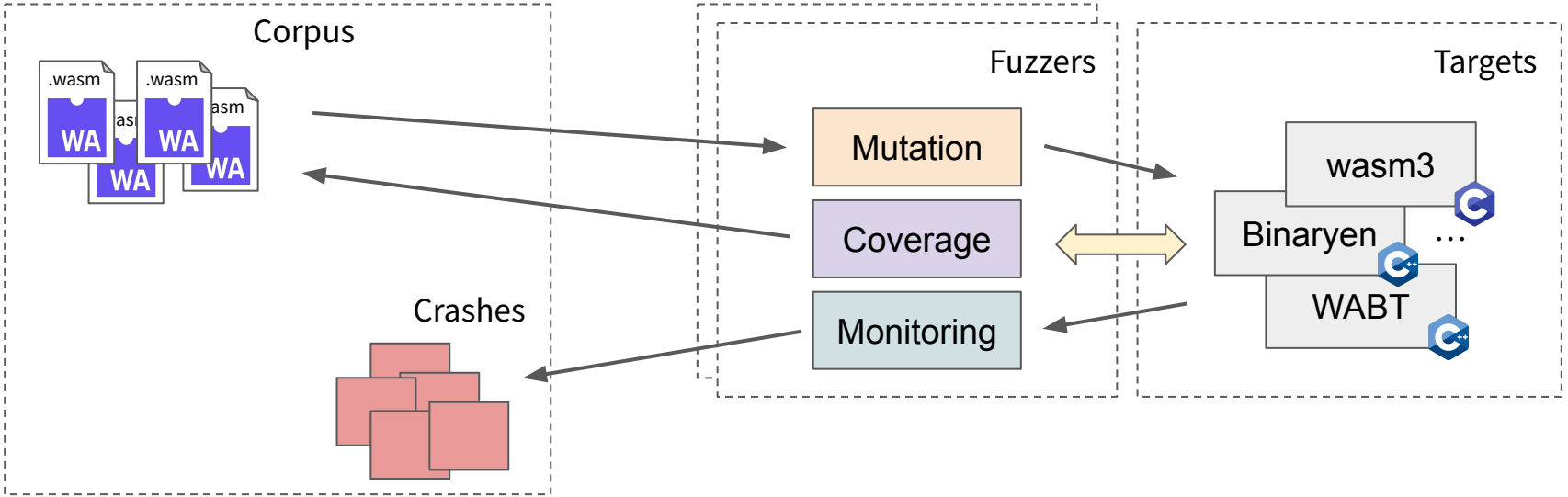
Fuzzing strategy: Improvements #1

- **Reusing corpora** between all targets

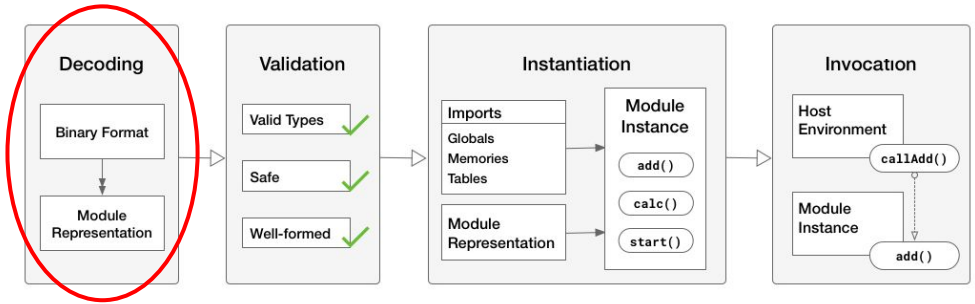


Fuzzing strategy: Improvements #1

- **Reusing corpora** between all targets
- Add crashing files inside the existing corpus
  - It might make crash some other targets

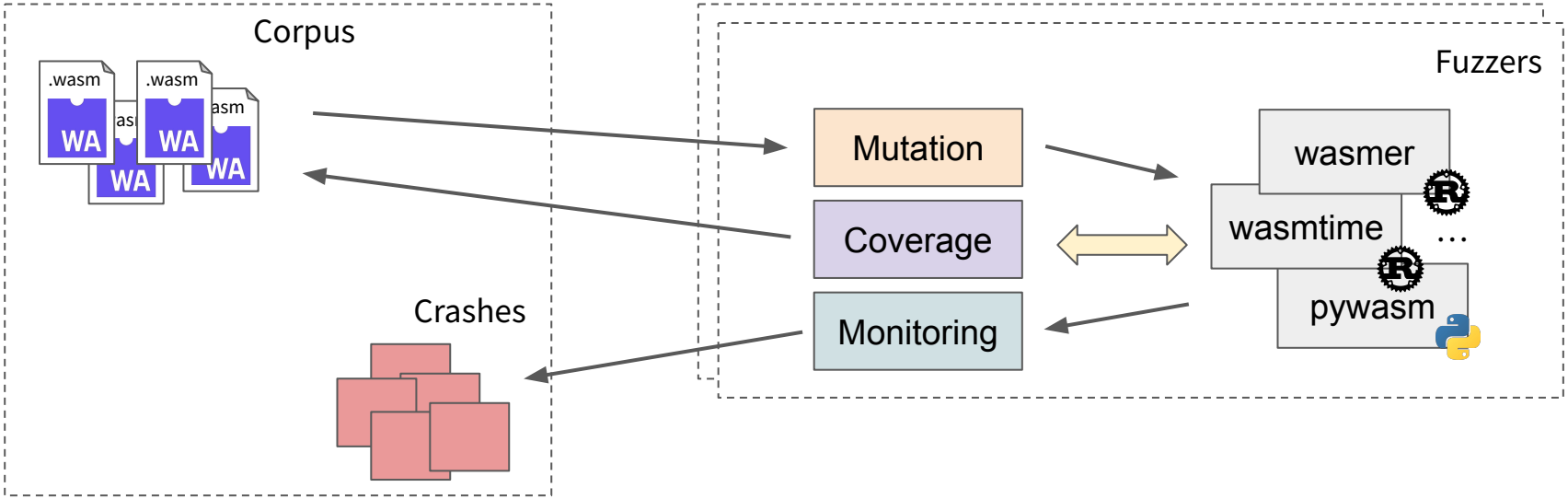


2. In-process fuzzing



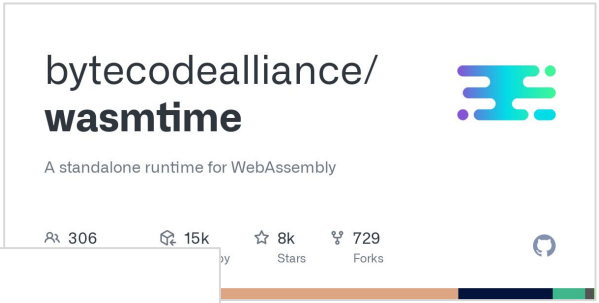
Fuzzing strategy: In-process fuzzing

- **In-Process** fuzzing
  - Fuzz a specific entry point of the program in **only one dedicated process**
  - For every test case, the **process isn't restarted** but the values are changed in memory.

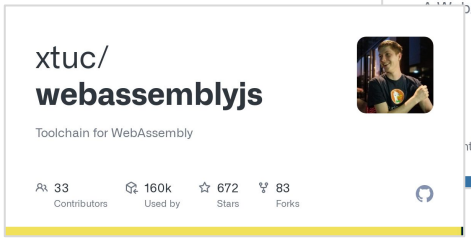


Targets: Standalone VMs & parsing libraries

- Targets (Rust)
  - Wasmer: WebAssembly **Runtime** supporting WASI and Emscripten
  - Wasmtime: A standalone **runtime** for WebAssembly
  - wain: WebAssembly **interpreter** written in Rust from scratch
  - Wasmparser: **Decoding/parsing library** of wasm binary files
  - wasmi: WebAssembly (Wasm) **interpreter**.
  - Cranelift: JIT compiler for wasm
  - Lucet: Sandboxing WebAssembly Compiler
  - Etc.



- Targets
  - pywasm: A WebAssembly interpreter written in pure **Python**
  - webassemblyjs: JavaScript Toolchain for WebAssembly



Rust In-process fuzzing

- Rust Fuzzers
  - **cargofuzz**: A cargo subcommand for fuzzing with **libFuzzer**
  - **honggfuzz-rs**: Fuzz your Rust code with **Honggfuzz**!
  - **afl.rs**: Fuzzing Rust code with **AFLplusplus**

```rust
pub fn fuzz_wain_parser(data: &[u8]) -> bool {
    // Parse binary into syntax tree
    match parse(&data) {
        Ok(_) => true,
        Err(_) => false,
    }
}
```

- Complexity: **Low**
  - You need to write some fuzzing harnesses
  - honggfuzz-rs is my favorite (faster and better interface)
  - New fuzzer **cargo-libafl** is promising

```rust
#[macro_use] extern crate honggfuzz;
extern crate wasmi;

fn main() {
    loop {
        fuzz!(|data: &[u8]| {
            // Just check if loading some arbitrary
            buffer doesn't panic.
            let _ = wasmi::Module::from_buffer(data);
        });
    }
}
```

```
------------------------[  0 days 10 hrs 13 mins 56 secs ]----------------------
  Iterations : 177,964,635 [177.96M]
  Mode [3/3] : Feedback Driven Mode
      Target : hfuzz_target/x86_64-unknown-linux-gnu/release/load
     Threads : 2, CPUs: 4, CPU%: 200% [50%/CPU]
       Speed : 5,592/sec [avg: 4,831]
     Crashes : 0 [unique: 0, blacklist: 0, verified: 0]
    Timeouts : 0 [10 sec]
 Corpus Size : 740, max: 22,893 bytes, init: 24,575 files
  Cov Update : 0 days 07 hrs 19 mins 32 secs ago
    Coverage : edge: 6,879 pc: 3 cmp: 68,248
------------------------------ [ LOGS ] -----------------/ honggfuzz 1.9 /-
Size:35 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6866/3/68020
Size:99 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6866/3/68021
Size:58 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/0, Tot:0/0/0/6867/3/68021
Size:58 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/0, Tot:0/0/0/6868/3/68021
Size:247 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6868/3/68022
Size:283 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6868/3/68023
Size:291 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6868/3/68024
Size:18 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6868/3/68025
Size:78 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/6868/3/68026
```

Python/JS In-process fuzzing

- Fuzzers
  - **Atheris**: Coverage-guided **Python** fuzzing engine based on Libfuzzer
  - **jsfuzz**: Coverage-guided fuzzer for **javascript**/nodejs packages

```
const parser = require("@webassemblyjs/wasm-parser");

function fuzz(buf) {
    try {
        parser.decode(buf, {});
    } catch (e) {
    // Those are "valid" exceptions. we can't catch them
    in one line as
    if (e.message.indexOf('Unexpected section') !== -1  ||
        e.message.indexOf('Atomic instructions') !== -1 ||
        e.message.indexOf('unknown table') !== -1 ||
        e.message.indexOf('Internal failure') !== -1 ||
        e.message.indexOf('Unexpected ') !== -1 ||
```

- Complexity: **Low**
  - You need to write some fuzzing harnesses
  - Learn how to use different fuzzing frameworks

```
import atheris
import sys
import io


with atheris.instrument_imports():
    import pywasm

def TestOneInput(input_bytes):
    """The code under test"""
    try:
        data = io.BytesIO(input_bytes)
        mod.from_reader(data)
    except Exception as e:
        msg = "{}".format(e)
        if "pywasm" in msg:
            pass
        else:
            raise

mod = pywasm.binary.Module

atheris.Setup(sys.argv, TestOneInput)
atheris.Fuzz()
```

Results: ~**62** bugs/vulnerabilities

- Results
  - Wasmer - issues (22)
  - Cranelift - issues (2)
  - Wasmparser - issues (3)
  - Wasmtime - issues (17)
  - wain - issues (4)
  - lucet - issues (2)
  - Pywasm - not reported (10)
  - webassemblyjs - issue

```
$ RUST_BACKTRACE=1 ./target/release/parse panic_wasmparser_index_oob_check_select.wasm
thread 'main' panicked at 'index out of bounds: the len is 0 but the index is 18446744073709551613',
```

wain/wain-validate/src/insn.rs
Lines 390 to 392 in 27f9ef4

```
390     // func.idx was already validated
391     let fty = &ctx.outer.module.types[func.idx as usize];
392     // Pop extracts parameters in reverse order
```

```
$ RUST_BACKTRACE=1 wasmer run panic_assert_wasmparser_operators_validators.wasm
thread 'main' panicked at 'assertion failed: stack_starts_at + index < self.stack_types.len()',
```

- Type of bugs found
  - Panicking **macros**
  - **Index out of bound** panic
  - **Assertion** failure
  - **Unwrapping** panics
  - Arithmetic **overflows**
  - Out of Memory (**OOM**) error
  - Unhandled exception (Python)

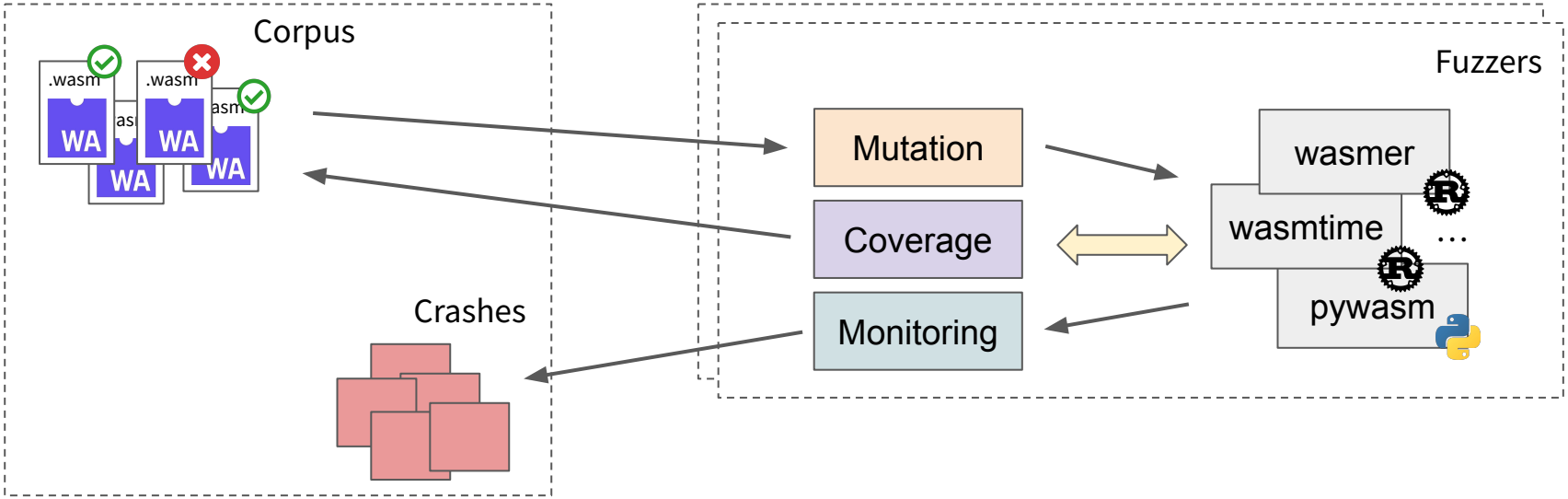
```
=== Uncaught Python exception: ===
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xae in position 0: invalid start byte
```

```
=== Uncaught Python exception: ===
TypeError: ord() expected a character, but string of length 0 found
```

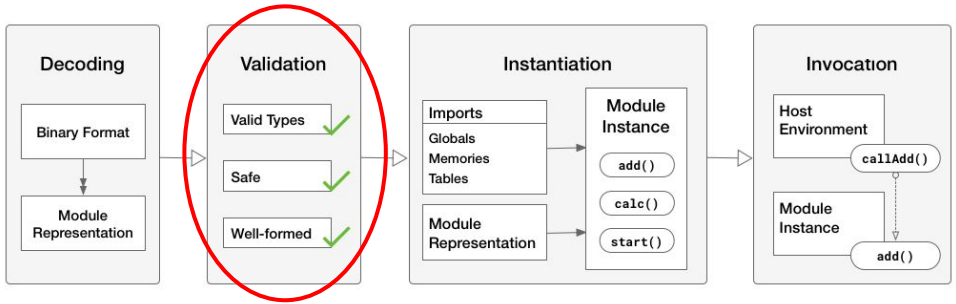
```
FATAL ERROR: invalid array length Allocation failed - JavaScript heap out of memory
#50 PULSE     cov: 2715 corp: 3520 exec/s: 0 rss: 688.21 MB
MEMORY OOM: exceeded 2048 MB. Killing worker
Worker killed
crash was written to crash-257ec9ec6e9f0fc2b1fdc6885fb96fedd89b5af1542beacbb2694347
Worker exited
```

Fuzzing strategy: Improvements #2

- **Improving the corpora** by gathering valid inputs/seeds from internet
  - WebAssembly/spec: WebAssembly core testsuite
  - Existing WebAssembly fuzzing corpora - here, here or there



3. Grammar-based fuzzing



Fuzzing strategy: Improvements #3

- **Add new fuzzing harnesses** to target validation **entry points**.
  - Module decoding will also be called by the validation function

```rust
pub fn wasmi_validate(data: &[u8]) -> bool {
    use parity_wasm::{deserialize_buffer, elements};
    use wasmi_validation::{validate_module, PlainValidator};

    let module: elements::Module = match deserialize_buffer(&data) {
        Ok(module) => module,
        _ => return false,
    };
    validate_module::<PlainValidator>(&module).is_ok()
}
```

```rust
/// Fuzzing `wasmtime::validate` with default Store/Config/Engine
pub fn fuzz_wasmtime_validate(data: &[u8]) -> bool {
    let store = Store::default();
    Module::validate(&store.engine(), &data).is_ok()
}

/// Fuzzing `wasmtime::validate` with all the features enabled
pub fn fuzz_wasmtime_validate_all_feat(data: &[u8]) -> bool {
    let store = match get_store_all_feat(Strategy::Cranelift) {
        None => return false,
        Some(a) => a,
    };
    Module::validate(&store.engine(), &data).is_ok()
}
```

```rust
pub fn fuzz_wasmparser_validate(data: &[u8]) -> bool {
    use wasmparser::validate;

    validate(&data).is_ok()
}
```

```rust
pub fn fuzz_wain_validate(data: &[u8]) -> bool {
    // Parse binary into syntax tree
    match parse(&data) {
        // Validate module
        Ok(tree) => validate(&tree).is_ok(),
        Err(_) => false,
    }
}
```

Main issue: Strict module validation mechanism

- The decoded module undergoes **validation checks** (such as type checking)
  - Validation mechanism is documented in the specs (here)
    - Conventions
    - Types
    - Instructions
    - Modules

> The WebAssembly stack machine is restricted to structured control flow and structured use of the stack. This greatly simplifies one-pass verification, avoiding a fixpoint computation like that of other stack machines such as the Java Virtual Machine (prior to stack maps). This also simplifies compilation and manipulation of WebAssembly code by other tools. Further generalization of the WebAssembly stack machine is planned post-MVP, such as the addition of multiple return values from control flow constructs and function calls.

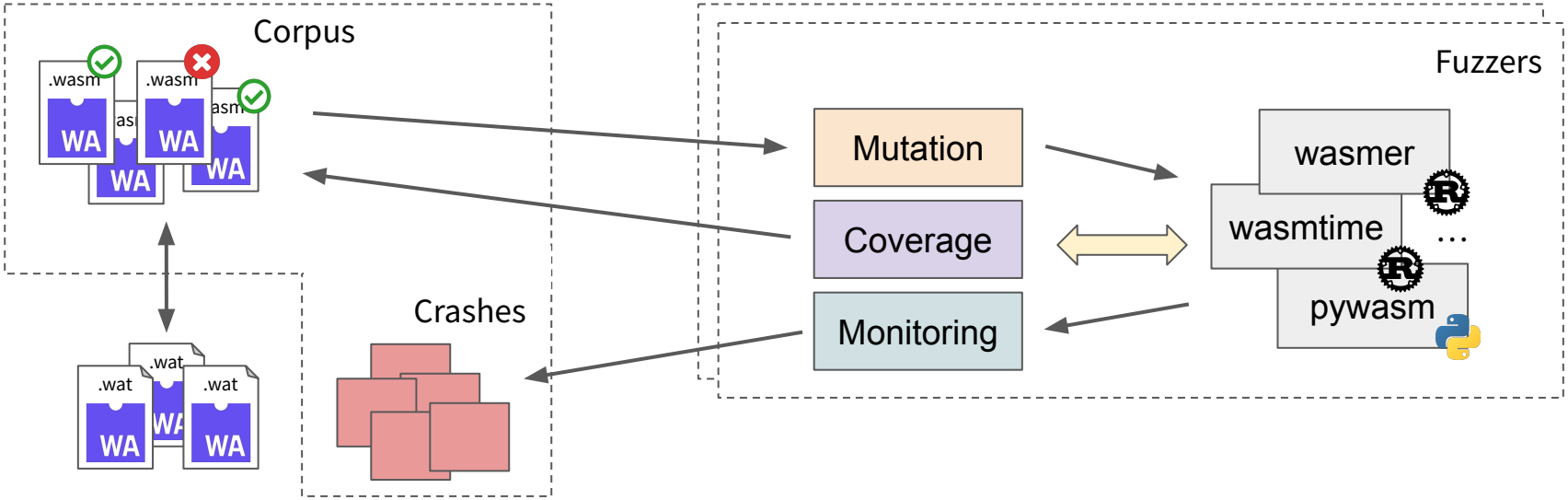
- Different implementations
  - wasm-validator tool (binaryen - C/C++)
  - wasm-validate tool (wabt - C/C++)
  - WebAssembly.validate (JS API - JavaScript)

```
type val_type = I32 | I64 | F32 | F64 | V128 | Funcref | Externref

func is_num(t : val_type | Unknown) : bool =
  return t = I32 || t = I64 || t = F32 || t = F64 || t = Unknown

func is_vec(t : val_type | Unknown) : bool =
  return t = V128 || t = Unknown

func is_ref(t : val_type | Unknown) : bool =
  return t = Funcref || t = Externref || t = Unknown
```

- Further reading:
  - WebAssembly Core Specification: Validation Algorithm - link
  - Mechanising and Verifying the WebAssembly Specification - link
  - “One pass verification process” explains - link

Standalone VMs: Grammar-based fuzzing

- **Grammar-based** fuzzing
  - Grammar allows for systematic and efficient test generation, particularly for complex formats.
  - Convert **WebAssembly text** files into wasm binaries and **add them to the corpora**
    - Found interesting wat files online, create and generate custom wat files



Input: WebAssembly Binary Format & Text Format

C/C++

```c
int fib(int n)
{
  if (n == 0 || n == 1)
    return n;
  else
    return (fib(n-1) + fib(n-2));
}
```

Rust

```rust
fn fib(n: u32) -> u32 {
    match n {
        0 => 1,
        1 => 1,
        _ => fib(n - 1) + fib(n - 2),
    }
}
```

Compilation

binary file (.wasm)

```
0061 736d 0100 0000
0186 8080 8000 0160
017f 017f 0382 8080
8000 0100 0484 8080
8000 0170 0000 0583
8080 8000 0100 0106
8180 8080 0000 0790
8080 8000 0206 6d65
6d6f 7279 0200 0366
6962 0000 0aa7 8080
8000 01a1 8080 8000
0002 4020 0041 0172
4101 470d 0020 000f
0b20 0041 7f6a 1000
2000 417e 6a10 006a
0b
```

wasm text format (.wat)

```
(module
  (table (;0;) 0 anyfunc)
  (memory (;0;) 1)
  (export "memory" (memory 0))
  (export "fib" (func 0))
  (type (;0;) (func (param i32) (result i32)))
  (func (;0;) (type 0) (param i32) (result i32)
    block  ;; label = @1
      get_local 0
      i32.const 1
      i32.or
      i32.const 1
      i32.ne
      br_if 0 (;@1;)
      get_local 0
      return
    end
    get_local 0
    i32.const -1
    i32.add
    call 0
    get_local 0
    i32.const -2
    i32.add
    call 0
    i32.add
  )
)
```

Input: WebAssembly Text Format

- Standardized text format
  - File extensions: `.wat`
  - **S-expressions** (like LISP): Module and section definitions
  - **Linear representation**: Functions body and Low-level instructions

- **MVP Instruction set**
  - Small Turing-complete ISA: ~172 instructions
  - Data types: **i32**, **i64**, **f32**, **f64**
  - **Control-Flow** operators
    - Label — `block loop if else end`
    - Branch — `br br if br table`
    - Function call — `call call indirect`
  - **Memory** operators — `load, store`
  - **Variables** operators — `local, global`
  - **Arithmetic** operators — `+ - * / % && >> sqrt`
  - **Constant** operators — `i32.const`
  - **Conversion** operators — `wrap trunc convert`

```
(module
  (table (;0;) 0 anyfunc)
  (memory (;0;) 1)
  (export "memory" (memory 0))
  (export "fib" (func 0))
  (type (;0;) (func (param i32) (result i32)))
  (func (;0;) (type 0) (param i32) (result i32)
    block  ;; label = @1
      get_local 0
      i32.const 1
      i32.or
      i32.const 1
      i32.ne
      br_if 0 (;@1;)
      get_local 0
      return
    end
    get_local 0
    i32.const -1
    i32.add
    call 0
    get_local 0
    i32.const -2
    i32.add
    call 0
    i32.add
  )
)
```

MVP 1.0 Instruction Set Architecture (ISA)

i32	i64	f32	f64			
i32.add	i64.add	f32.add	f64.add	i32.wrap/i64	i32.load8_s	i32.store8
i32.sub	i64.sub	f32.sub	f64.sub	i32.trunc_s/f32	i32.load8_u	i32.store16
i32.mul	i64.mul	f32.mul	f64.mul	i32.trunc_s/f64	i32.load16_s	i32.store
i32.div_s	i64.div_s	f32.div	f64.div	i32.trunc_u/f32	i32.load16_u	i64.store8
i32.div_u	i64.div_u	f32.abs	f64.abs	i32.trunc_u/f64	i32.load	i64.store16
i32.rem_s	i64.rem_s	f32.neg	f64.neg	i32.reinterpret/f32	i64.load8_s	i64.store32
i32.rem_u	i64.rem_u	f32.copysign	f64.copysign	i64.extend_s/i32	i64.load8_u	i64.store
i32.and	i64.and	f32.ceil	f64.ceil	i64.extend_u/i32	i64.load16_s	f32.store
i32.or	i64.or	f32.floor	f64.floor	i64.trunc_s/f32	i64.load16_u	f64.store
i32.xor	i64.xor	f32.trunc	f64.trunc	i64.trunc_s/f64	i64.load32_s	
i32.shl	i64.shl	f32.nearest	f64.nearest	i64.trunc_u/f32	i64.load32_u	
i32.shr_u	i64.shr_u			i64.trunc_u/f64	i64.load	call
i32.shr_s	i64.shr_s	f32.sqrt	f64.sqrt	i64.reinterpret/f64	f32.load	call_indirect
i32.rotl	i64.rotl	f32.min	f64.min		f64.load	
i32.rotr	i64.rotr	f32.max	f64.max			
i32.clz	i64.clz				nop	grow_memory
i32.ctz	i64.ctz				block	current_memory
i32.popcnt	i64.popcnt			f32.demote/f64	loop	
i32.eqz	i64.eqz			f32.convert_s/i32	if	get_local
i32.eq	i64.eq			f32.convert_s/i64	else	set_local
i32.ne	i64.ne			f32.convert_u/i32	br	tee_local
i32.lt_s	i64.lt_s			f32.convert_u/i64	br_if	
i32.le_s	i64.le_s			f32.reinterpret/i32	br_table	get_global
i32.lt_u	i64.lt_u	f32.eq	f64.eq	f64.promote/f32	return	set_global
i32.le_u	i64.le_u	f32.ne	f64.ne	f64.convert_s/i32	end	
i32.gt_s	i64.gt_s	f32.lt	f64.lt	f64.convert_s/i64		i32.const
i32.ge_s	i64.ge_s	f32.le	f64.le	f64.convert_u/i32	drop	i64.const
i32.gt_u	i64.gt_u	f32.gt	f64.gt	f64.convert_u/i64	select	f32.const
i32.ge_u	i64.ge_u	f32.ge	f64.ge	f64.reinterpret/i64	unreachable	f64.const

Results: ~**6** bugs/vulnerabilities

- Found some new bugs **by accident** during conversion from text format (wat) to binary format (wasm)
- Wasmprinter (Rust)
  - **Out of Memory (OOM) error** - issue

```
memory allocation of 4294967296 bytes failed[1]    12638 abort (core dumped)
```

- WABT (C/C++) - wasm2wat, wast2json
  - **Assertion failure** - issues (5)

wabt/src/binary-reader-ir.cc
Lines 736 to 738 in e88bc66

```
736  Result BinaryReaderIR::OnReturnCallIndirectExpr(Index sig_index, Index table_index) {
737    assert(sig_index < module_->types.size());
738    auto expr = MakeUnique<ReturnCallIndirectExpr>();
```

wabt/src/binary-reader-ir.cc
Lines 724 to 726 in e88bc66

```
724  Result BinaryReaderIR::OnCallIndirectExpr(Index sig_index, Index table_index) {
725    assert(sig_index < module_->types.size());
726    auto expr = MakeUnique<CallIndirectExpr>();
```

```
SIGABRT.PC.7ffff7a8818b.STACK.1924350c46.CODE.-6.ADDR.0.INSTR.mov____0x108(%rsp),%rax.fuzz
wast2json: /home/wasm_training/Documents/wasm_tools/wabt/src/binary-writer.cc:1545: wabt::Result wabt::(a
nonymous namespace)::BinaryWriter::WriteModule(): Assertion `module_->GetMemoryIndex(segment->memory_var)
 == 0' failed.
Aborted (core dumped)

SIGABRT.PC.7ffff7a8818b.STACK.1b61dc8673.CODE.-6.ADDR.0.INSTR.mov____0x108(%rsp),%rax.fuzz
wast2json: /home/wasm_training/Documents/wasm_tools/wabt/src/token.h:113: const wabt::Literal &wabt::Toke
n::literal() const: Assertion `HasLiteral()' failed.
Aborted (core dumped)

SIGABRT.PC.7ffff7a8818b.STACK.eea4d22cf.CODE.-6.ADDR.0.INSTR.mov____0x108(%rsp),%rax.fuzz
wast2json: /home/wasm_training/Documents/wasm_tools/wabt/src/wast-parser.cc:567: wabt::Token wabt::WastPa
rser::Consume(): Assertion `!tokens_.empty()' failed.
Aborted (core dumped)
```

Fuzzing strategy: Improvements #4

- Create **edge case** modules
  - Duplicate sections (unique & customs)
  - Redefinition of exported/imported functions & memory
  - Change sections ordering
  - Create a lot of sections, elements, etc.
  - Inject unusual values for int/float

```
10_memory.wast:6:3: error: only one memory block allowed
      (memory 1 )
  ^^^^^^
```

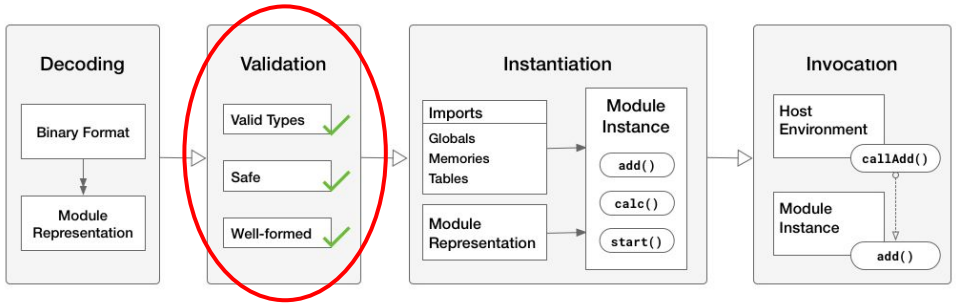
```
10_table.wast:6:3: error: only one table allowed
      (table 1 anyfunc)
  ^^^^^
```

```
2_func0.wast:5:4: error: redefinition of function "$func1"
  (func $func1 (type 0) (result i32)
   ^^^^
```

- Create a **polyglot** WebAssembly module
  - Valid HTML/JS/wasm file
    - Data section injection
    - Custom section injection
  - Detailed blogpost here

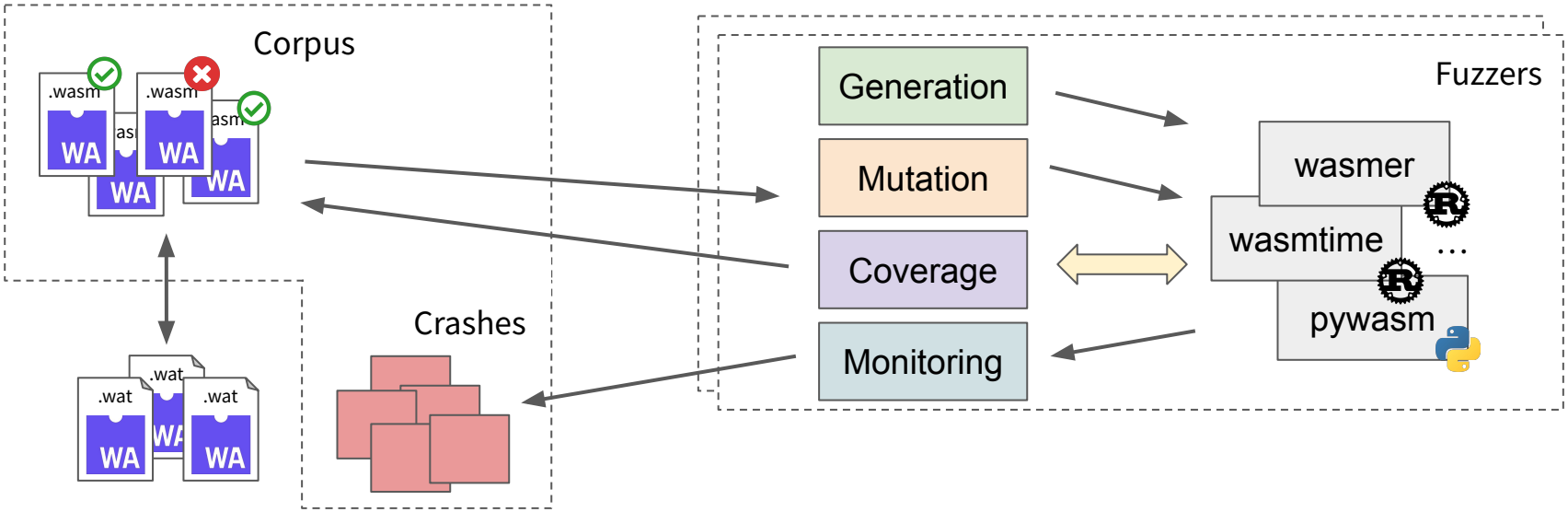
```
unop:  ctz | clz | popcnt | ...
binop: add | sub | mul | ...
relop: eq | ne | lt | ...
sign:  s|u
offset: offset=<nat>
align: align=(1|2|4|8|...)
cvtop: trunc | extend | wrap | ...

val_type: i32 | i64 | f32 | f64
elem_type: funcref
block_type : ( result <val_type>* )*
func_type:   ( type <var> )? <param>* <result>*
global_type: <val_type> | ( mut <val_type> )
table_type:  <nat> <nat>? <elem_type>
memory_type: <nat> <nat>?

expr:
  ( <op> )
  ( <op> <expr>+ )
  ( block <name>? <block_type> <instr>* )
  ( loop <name>? <block_type> <instr>* )
  ( if <name>? <block_type> ( then <instr>* ) (
  ( if <name>? <block_type> <expr>+ ( then <inst
```

4. Structure-aware fuzzing



Fuzzing strategy: Structure-aware fuzzing

- **Structure-aware** fuzzing
  - Generate semi-well-formed inputs based on knowledge of structure, file format, or protocol.
  - Modules are generated, **without losing time in parsing**, with fuzzy values placed at strategic locations.



Standalone VMs (Rust): Structure-based fuzzing

- Fuzzers
  - **Arbitrary trait**: The trait for generating structured data from arbitrary, unstructured input.
  - **wasm-smith**: A WebAssembly test case generator.

- Targets (all)
  - Rust code directly **via in-process fuzzing** (cargofuzz, honggfuzz-rs, etc.)
  - Other targets via **shared corpora**

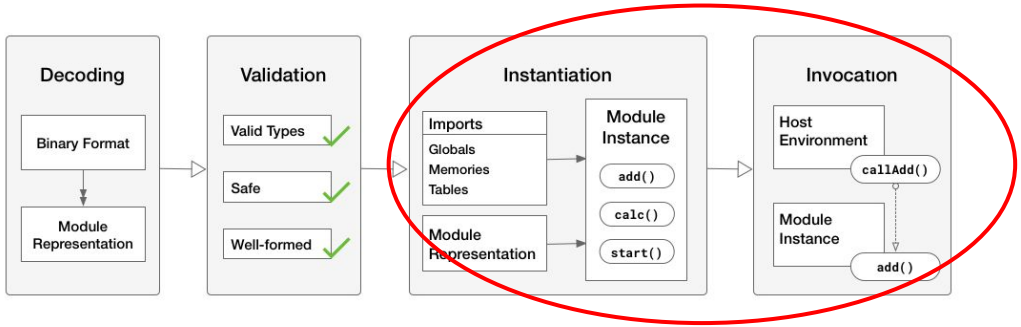
- Complexity: **Low/Medium**
  - Integrating the arbitrary trait can be challenging
  - Wasm-smith is really good, fast and easy to use

- Results: **0 new direct bugs**
  - Generate interesting inputs that will be mutated later
  - Helps to increase coverage

```
[dependencies]
wasm-smith = "0.4.0"
```

```
#![no_main]

use libfuzzer_sys::fuzz_target;
use wasm_smith::Module;

fuzz_target!(|module: Module| {
    let wasm_bytes = module.to_bytes();

    // Your code here...
});
```

5. Differential fuzzing



Fuzzing strategy: Improvements #5

- **Add new fuzzing harnesses** to target instantiation phases.
  - Create simple imports and provide them to Instance constructors.

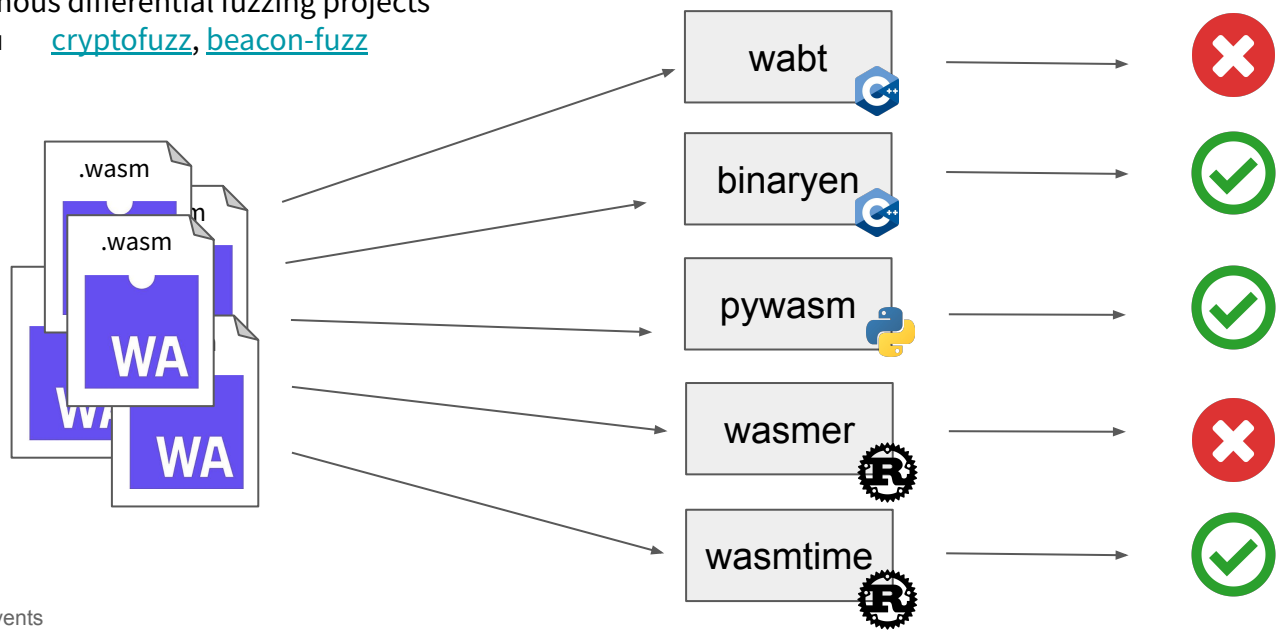
```rust
pub fn fuzz_wasmer_instantiate(data: &[u8]) -> bool {
    use wasmer_runtime::{imports, instantiate};
    let import_object = imports! {};

    instantiate(&data, &import_object).is_ok()
}
```

```rust
pub fn wasmi_instantiate(data: &[u8]) -> bool {
    use wasmi::{ImportsBuilder, Module, ModuleInstance};

    match Module::from_buffer(&data) {
        Ok(module) => ModuleInstance::new(&module, &ImportsBuilder::default()).is_ok(),
        _ => false,
    }
}
```

```rust
pub fn fuzz_wasmtime_instantiate_all_cranelift(data: &[u8]) -> bool {
    let store = match get_store_all_feat(Strategy::Cranelift) {
        None => return false,
        Some(a) => a,
    };
    // Create a Module
    let module = match Module::from_binary(&store.engine(), &data) {
        Ok(a) => a,
        _ => return false,
    };
    Instance::new(&store, &module, &[]).is_ok()
```

Fuzzing strategy: Differential fuzzing

- **Differential** fuzzing
  - Observe if two program implementations/variants **produce different outputs** for the **same input**.
  - Really **efficient way to find logic bugs**, unimplemented cases, etc.
  - Famous differential fuzzing projects
    - cryptofuzz, beacon-fuzz



Differential fuzzing

- Type of bugs:
  - Logic bugs or unimplemented features
  - **Consensus bugs** (critical for blockchains)
- **Fuzzers**: Just a Python or Bash script is working
- **Targets**: All of them

```rust
pub fn fuzz_diff_instantiate(data: &[u8]) {
    let a = wasmi::wasmi_instantiate(&data);
    let b = wasmer::fuzz_wasmer_instantiate(&data);
    let c = wasmtime::fuzz_wasmtime_instantiate_all_cranelift(&data);
    let _ = match (a, b, c) {
        (true, true, true) => true,
        (false, false, false) => false,
        _ => panic!("fuzz_diff_instantiate panic: {}-{}-{}", a, b, c),
    };
}
```

- Complexity: **Low**
  - No need for any bindings if you're using threads/subprocesses
  - A lot of false positives due to WebAssembly feature supports

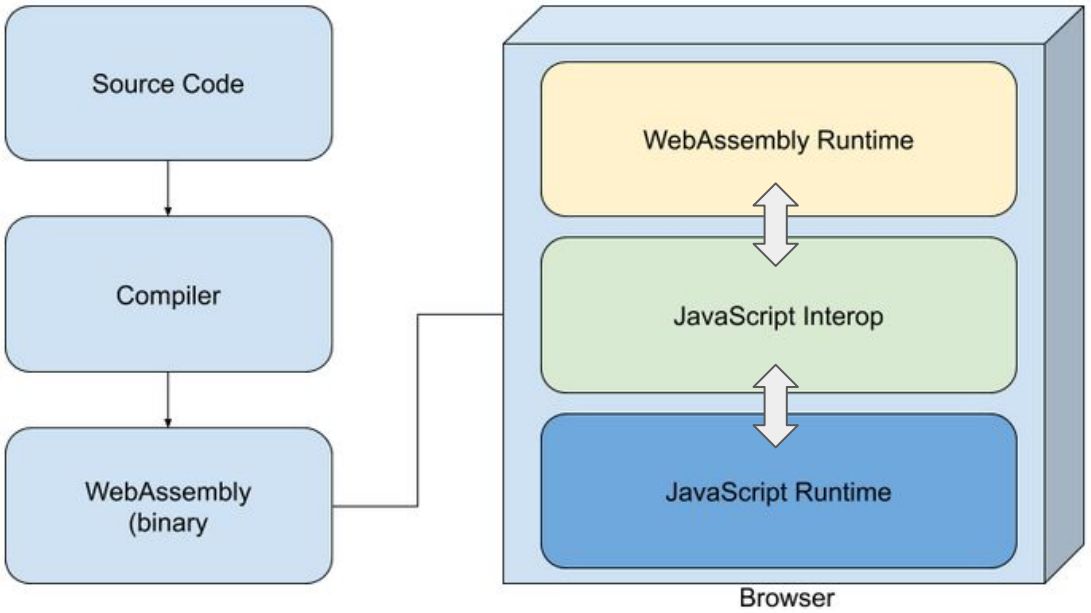
- Results: **2 bugs/vulnerabilities**
  - [wabt] Incorrect validation/rejection - [issues]

```python
import glob
import subprocess
import sys

def execute(input_file, target):
    # Run wasm2wat until completion
    sp = subprocess.Popen([target, input_file]
        ,stdout=subprocess.DEVNULL
        ,stderr=subprocess.DEVNULL)
    ret = sp.wait()

    return ret

corpus_filenames = glob.glob("corpus/*")  #
glob is better b/c full paths
print(corpus_filenames)

corpus = set()
for file in corpus_filenames:
    ret_wasm2wat = execute(file, "wasm2wat")
    ret_wasmdis = execute(file, "wasm-dis")

    if ret_wasm2wat != ret_wasmdis:
        print(file)
        print(f"replay: wasm2wat {file} > /dev
            /null && echo $?")
        print(f"replay: wasm-dis {file} > /dev
            /null && echo $?")
        print(f"diff bug: wasm2wat {
            ret_wasm2wat} | wasmdis {
            ret_wasmdis}")
        sys.exit()
```

What about browsers?

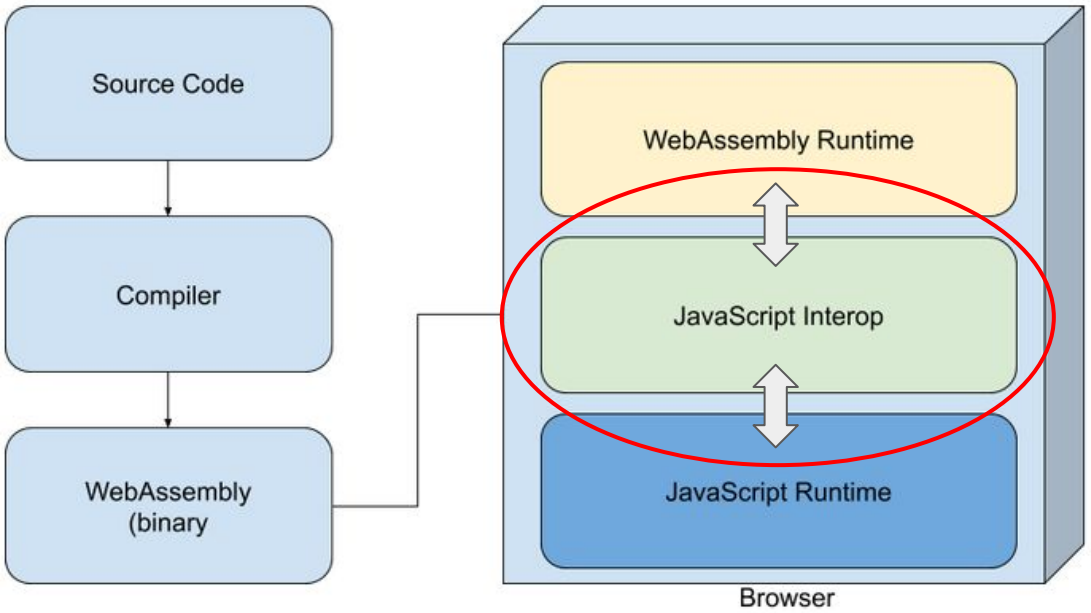
Targets: Browser's WebAssembly VMs

- In browsers, the WebAssembly runtime is **part of the JavaScript engine.**
- Targets
  - SpiderMonkey (Firefox)
  - JavaScriptCore (Safari)
  - V8 (Google chrome)



Targets: Browser's WebAssembly VMs

- In browsers, the WebAssembly runtime is **part of the JavaScript engine.**
- Targets
  - SpiderMonkey (Firefox)
  - JavaScriptCore (Safari)
  - V8 (Google chrome)



WebAssembly JavaScript APIs

- Complete documentation on Mozilla [MDN for WebAssembly]
  - Methods/Constructors
  - [Browser compatibility table]

`WebAssembly.instantiate()`
The primary API for compiling and instantiating WebAssembly code, returning both a `Module` and its first `Instance`.

`WebAssembly.instantiateStreaming()`
Compiles and instantiates a WebAssembly mod
source, returning both a `Module` and its first `I`

`WebAssembly.compile()`
Compiles a `WebAssembly.Module` from WebA
instantiation as a separate step.

`WebAssembly.compileStreaming()`
compiles a `WebAssembly.Module` directly fror
instantiation as a separate step.

`WebAssembly.validate()`
Validates a given typed array of WebAssembly b
are valid WebAssembly code (`true`) or not (`fa`

`WebAssembly.Global()`
Creates a new WebAssembly `Global` object.

`WebAssembly.Module()`
Creates a new WebAssembly `Module` object.

`WebAssembly.Instance()`
Creates a new WebAssembly `Instance` object.

`WebAssembly.Memory()`
Creates a new WebAssembly `Memory` object.

`WebAssembly.Table()`
Creates a new WebAssembly `Table` object.

`WebAssembly.CompileError()`
Creates a new WebAssembly `CompileError` object.

`WebAssembly.LinkError()`
Creates a new WebAssembly `LinkError` object.

`WebAssembly.RuntimeError()`
Creates a new WebAssembly `RuntimeError` object.

Basic support	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`CompileError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Global`	No	No	62	No	No	No	No	No	No	62	No	No	No	No
`Instance`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`LinkError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Memory`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Module`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`RuntimeError`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`Table`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`compile`	57	16	52 *	No	44	11	57	57	Yes	52 *	44	11	7.0	8.0.0
`compileStreaming`	61	16	58	No	47	No	61	61	No	58	?	No	No	No
`instantiate`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0
`instantiateStreaming`	61	16	58	No	47	No	61	61	No	58	?	No	No	No
`validate`	57	16	52 *	No	44	11	57	57	Yes	52 *	?	11	7.0	8.0.0

WebAssembly JavaScript APIs

- WebAssembly.**Instance**
  - **Stateful, executable instance** of a WebAssembly.**Module**.

```
var m = new WebAssembly.Instance(new WebAssembly.Module(buffer));
```

- WebAssembly.**instantiate**
  - **Compile** and **instantiate** WebAssembly code.

```
<script>
  fetch('fib.wasm').then(response =>
    response.arrayBuffer()
  ).then(bytes =>
    WebAssembly.instantiate(bytes, {})
  ).then(results => {
```

- WebAssembly.**instantiateStreaming**
  - **Compiles** and **instantiates** a WebAssembly module directly from a **streamed** underlying source.

```
var memory = new WebAssembly.Memory({initial:10, maximum:100});
```

- WebAssembly.**Memory**
  - **Accessible** and **mutable** from both JavaScript and WebAssembly.

```
const global = new WebAssembly.Global({value:'i32', mutable:true}, 0);
```

- WebAssembly.**Global**
  - **Global variable instance**, accessible from both JavaScript and importable/exportable across one or more WebAssembly.Module instances.

```
var tbl = new WebAssembly.Table({initial:2, element:"anyfunc"});
```

- WebAssembly.**Table**
  - **Array-like structure** accessible & mutable from both JavaScript and WebAssembly.

Fuzzing strategy: Grammar-based fuzzing

- **Grammar-based** fuzzing
  - Javascript files are generated by the fuzzer based on a given grammar
  - We are generating **sequence of WebAssembly JavaScript APIs** calls
  - Fuzzers
    - Dharma: Generation-based, context-free grammar fuzzer - wasm.dg
    - Domato: DOM fuzzer
    - Fuzzilli4wasm: Fuzzer for wasm fuzzing based on fuzzilli
- Targets
  - SpiderMonkey (Firefox)
  - JavaScriptCore (Safari)
  - V8 (Google chrome)
- Complexity: **Medium**
  - You need to manually write grammars
  - It's time-consuming
- Results: **Some bugs & duplicates**
  - Not public

```
function main() {
    const v6 = {mutable:true,value:"i32"};
    let v7 = v6;
    const v10 = {mutable:true,value:"i64"};
    let v11 = v10;
    const v14 = new WebAssembly.Global(v7,-4024951421);
    const v15 = v14.toString();
    const v16 = new WebAssembly.Global(v7,0);
    const v32 = v14.toString();
    const v33 = v16.valueOf();
    const v34 = {mutable:true,value:"f32"};
    let v35 = v34;
    const v83 = v16.toString();
    const v84 = v16.valueOf();
    const v85 = v16.valueOf();
    const v97 = v16.valueOf();
    const v98 = v16.valueOf();
}
```

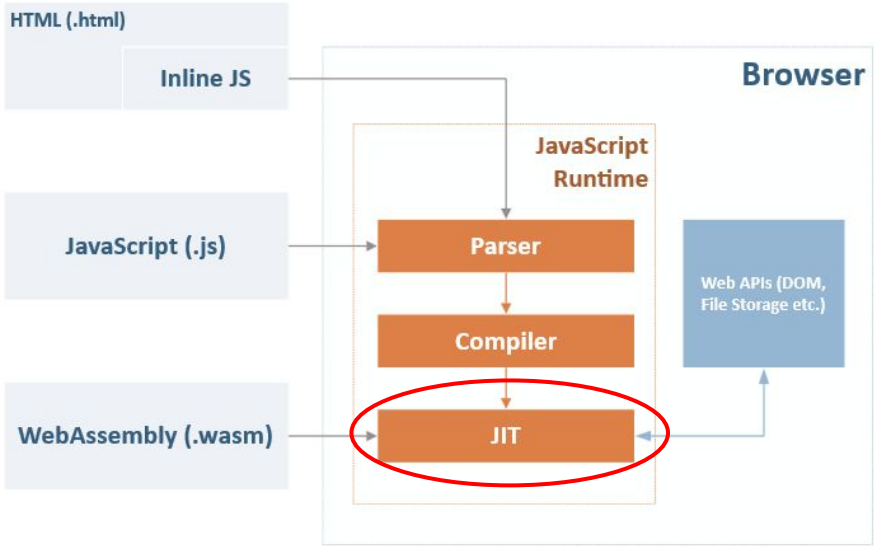
```
try { modulewasm7 = new WebAssembly.Module(new Uint8Array([0,97,115,109,1,0,0,0,1,133,128,
    128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,128,0,0,7,145,128,128,
    0,0,10,138,128,128,128,0,1,132,128,128,128,0,0,65,42,11])); } catch(e) {}
try { memorywasm7 = instancewasm7.exports.memory; } catch(e) {}
try { for (var i = 0; i < memorywasm13.buffer.length; i++) {memorywasm6.buffer[i] = 7;} }
try { string15 = WebAssembly.Module.exports(modulewasm13).toString(); } catch(e) {}
try { memorywasm6 = instancewasm1.exports.memory; } catch(e) {}
try { globalwasm6.value = number2; } catch(e) {}
try { number4 = tablewasm12.grow(number2); } catch(e) {}
try { number4 = globalwasm13.valueOf(); } catch(e) {}
try { number2 = memorywasm2.buffer.length - 1; } catch(e) {}
try { number13 = tablewasm10.grow(number12); } catch(e) {}
try { memorywasm1 = instancewasm6.exports.memory; } catch(e) {}
try { array6 = WebAssembly.Module.imports(modulewasm1); } catch(e) {}
try { array2 = WebAssembly.Module.customSections(modulewasm4, "debug"); } catch(e) {}
```

Targets: WebAssembly JIT engines

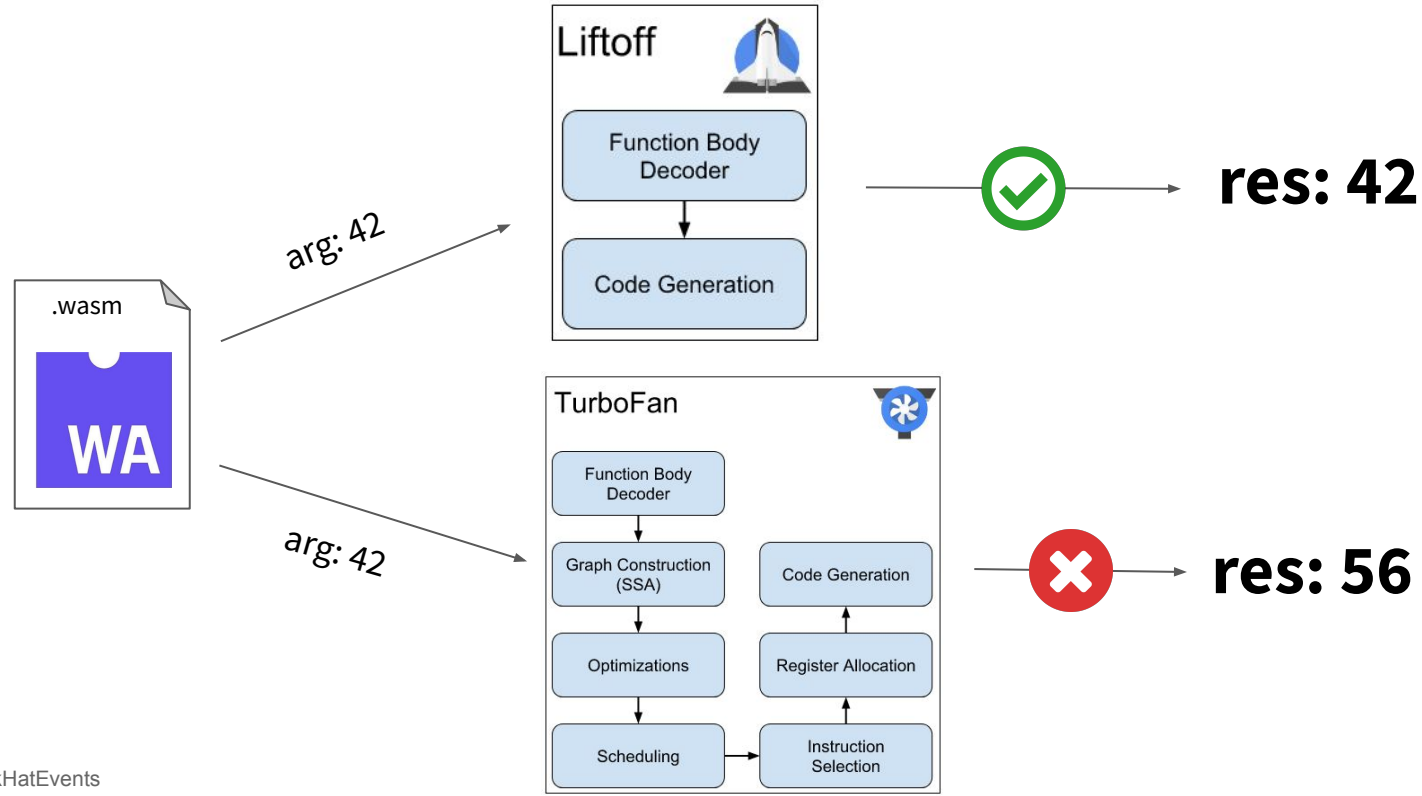
- **Spidermonkey** (Firefox)
  - WASM-**Baseline**: fast translation to machine code
  - WASM-**Ion**: wasm to MIR translator
  - **Cranelift**: low-level retargetable code generator

- **JavaScriptCore** (Safari)
  - **LLInt**: Low Level Interpreter
  - **BBQ**: Build Bytecode Quickly
  - **OMG**: Optimized Machine-code Generator

- **V8** (Google chrome)
  - **Liftoff**: baseline compiler for WebAssembly
  - **TurboFan**: optimizing compiler



Fuzzing strategy: Differential fuzzing



Fuzzing strategy: Differential fuzzing

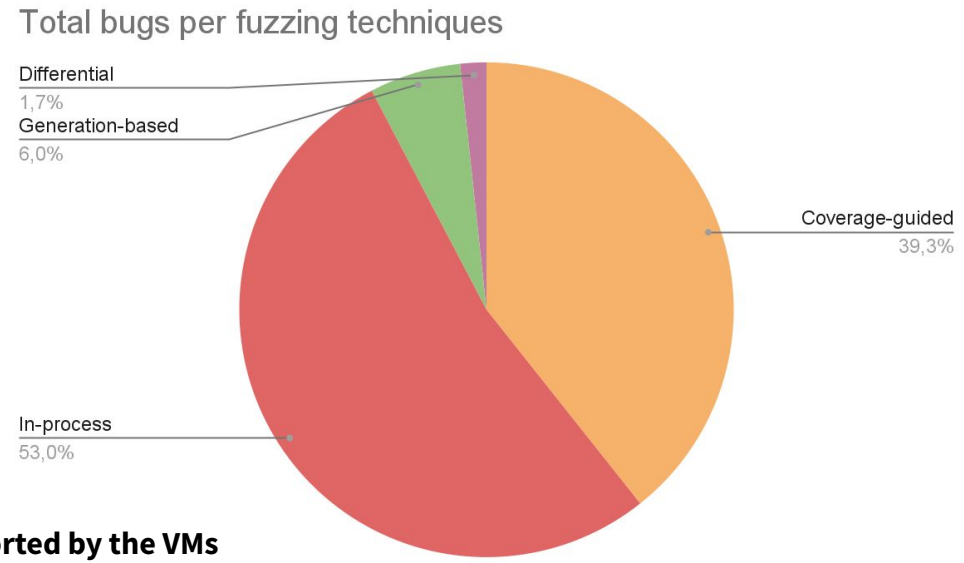
- Type of JIT bugs
  - Memory corruption bugs in the compiler
  - **Incorrect optimization**
  - Bugs in code generators
- Targets
  - WASM-**Baseline** vs WASM-**Ion** vs **Cranelift**
  - **LLInt** vs **BBQ** vs **OMG**
  - **Liftoff** vs **TurboFan**
- Complexity: **Hard**
  - You need to **generate valid wasm modules**
  - You can force optimization using JS loops
- Results: **0 bugs/vulnerabilities (WIP)**
  - JIT compilers for WebAssembly are really simple for the moment
  - Not a lot of public research, it's still an early stage idea but some non-public bugs have been reported by researchers.

```
--- WebAssembly code ---
name: wasm-function[0]
index: 0
kind: wasm function
compiler: Liftoff
Body (size = 128 = 116 + 12 padding)
Instructions (size = 104)
0x7f4b338e74c0     0  55                  push rbp
0x7f4b338e74c1     1  4889e5              REX.W movq rbp,rsp
0x7f4b338e74c4     4  6a08                push 0x8
0x7f4b338e74c6     6  56                  push rsi
0x7f4b338e74c7     7  4881ec18000000      REX.W subq rsp,0x18
0x7f4b338e74ce     e  48c745e800000000    REX.W movq [rbp-0x18],0x0
0x7f4b338e74d6    16  48c745e000000000    REX.W movq [rbp-0x20],0x0
0x7f4b338e74de    1e  488b4627            REX.W movq rax,[rsi+0x27]
0x7f4b338e74e2    22  483b20              REX.W cmpq rsp,[rax]
0x7f4b338e74e5    25  0f8631000000        jna 0x7f4b338e751c  <+0x5c>
0x7f4b338e74eb    2b  b810000000          movl rax,0x10
0x7f4b338e74f0    30  48837df808          REX.W cmpq [rbp-0x8],0x8
0x7f4b338e74f5    35  7420                jz 0x7f4b338e7517  <+0x57>
                                          movl rdi,0x27
                                          REX.W movq r10,rsp
                                          REX.W subq rsp,0x8
                                          REX.W andq rsp,0xf0
                                          REX.W movq [rsp],r10
                                          REX.W movq rax,0x7f4b4b820230
                                          call rax
                                          REX.W movq rsp,rbp
                                          pop rbp
                                          retl
                                          call 0x7f4b338e72b0  (jump table)
                                          REX.W movq rsi,[rbp-0x10]
                                          jmp 0x7f4b338e74eb  <+0x2b>
                                          nop
compiler: TurboFan
Body (size = 64 = 28 + 36 padding)
Instructions (size = 20)
0x7f4b338e7540     0  55                  push rbp
0x7f4b338e7541     1  4889e5              REX.W movq rbp,rsp
0x7f4b338e7544     4  6a08                push 0x8
0x7f4b338e7546     6  56                  push rsi
0x7f4b338e7547     7  b810000000          movl rax,0x10
0x7f4b338e754c     c  488be5              REX.W movq rsp,rbp
0x7f4b338e754f     f  5d                  pop rbp
0x7f4b338e7550    10  c3                  retl
0x7f4b338e7551    11  90                  nop
0x7f4b338e7552    12  6690                nop
```

Results & Closing Remarks

Conclusion & Final results

- Some numbers
  - **~117 bugs found**
    - Rust: 53, C/C++: 53
    - Python: 10, JavaScript: 1
    - Some non-public bugs
  - Final corpora size: **~2M** wasm modules
  - Total research time: **2 years**
  - Active research time: **6 months full-time**
  - **~84** fuzzing **harnesses** created
  - WARF: WebAssembly Runtimes Fuzzing



- Challenges
  - Complex to keep everything up-to-date
  - Not the same WebAssembly features are **supported by the VMs**
  - Need to adapt to multiple fuzzing frameworks and languages

- Future / Next steps
  - Add new targets and fuzzing harnesses (Go, Java, etc.)
  - **Update fuzzing harnesses** for WebAssembly MVP 2.0

Thanks for your time! Any questions?

- Twitter: **@Pat_Ventuzelo**
- Mail: patrick@fuzzinglabs.com

black hat USA 2022

FUZZING LABS

SLIDES