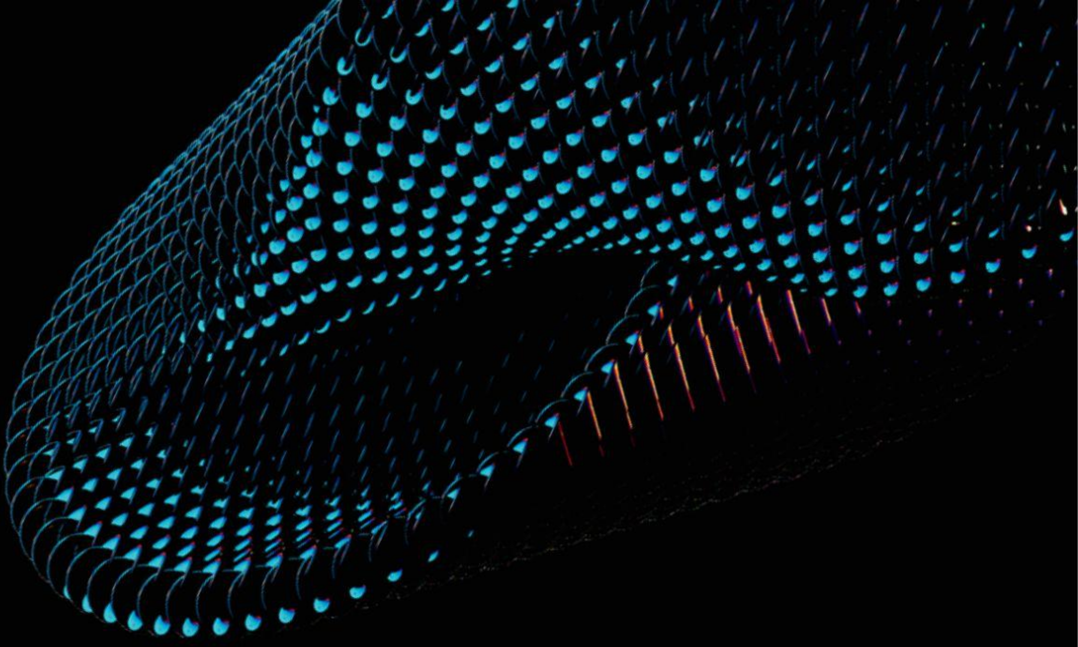
MultiversX



# MultiversX smart contracts from specification to execution

Andrei Marinica

# This Presentation

1. Introduction
2. Objectives
3. The Engine: WebAssembly, Wasmer, EI
4. On-chain composability
5. Off-chain composability
6. Specifying contract systems
7. Formal models
8. Conclusions

# Introduction

# What is MultiversX

- A scalable Layer 1 blockchain protocol (state sharding, PoS)
- An ecosystem of products:
  - xFabric
  - xPortal
  - xWorlds
  - many more …

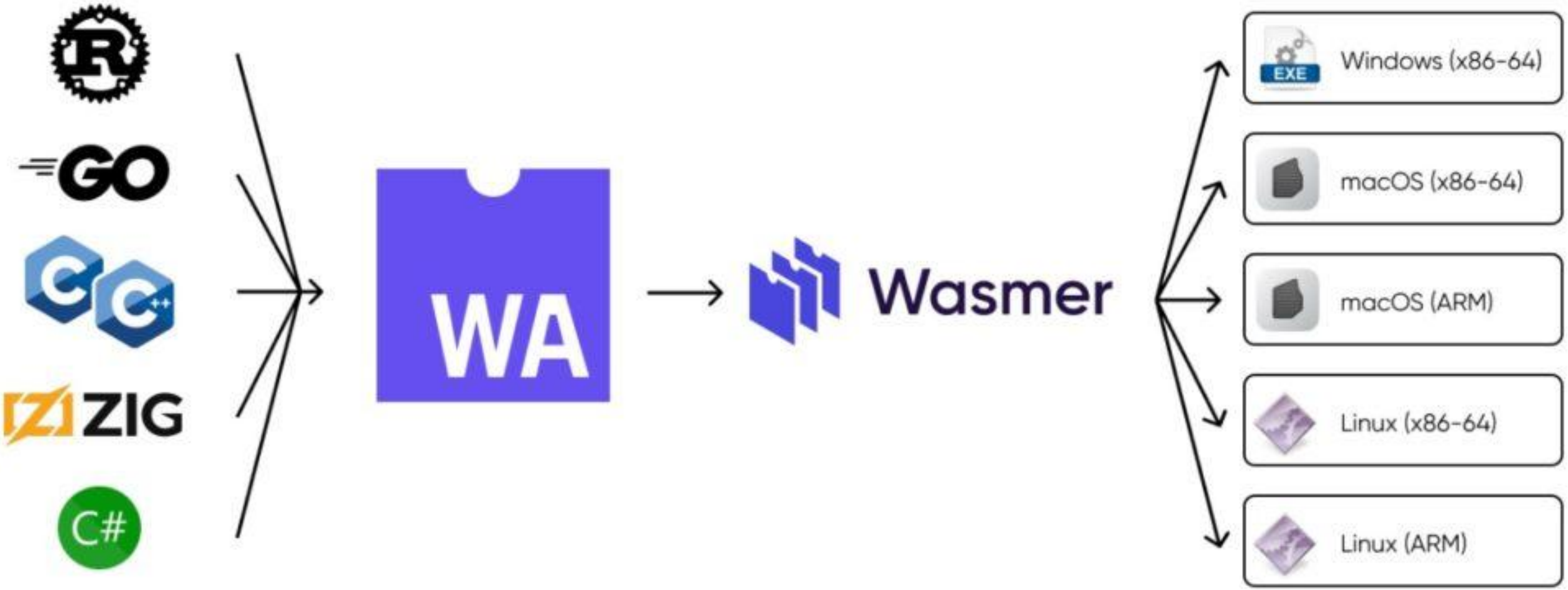
# Objectives

# A successful SC system needs:

- Execution speed
- Determinism
- Safety of:
    - Primitives
    - Smart Contracts
    - Interactions
- Composability

# The Engine: WebAssembly and Wasmer

# From high-level language to execution



Windows (x86-64)

macOS (x86-64)

macOS (ARM)

Linux (x86-64)

Linux (ARM)

Your favorite language

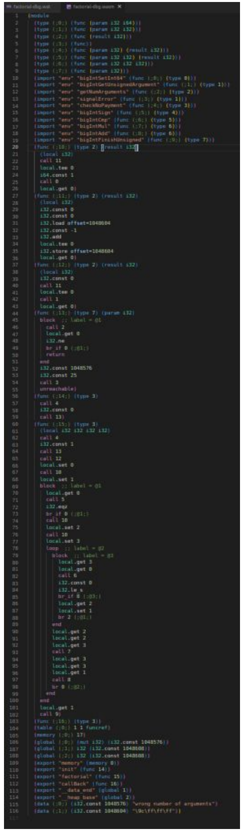
WebAssembly

create-exe

Standalone executables

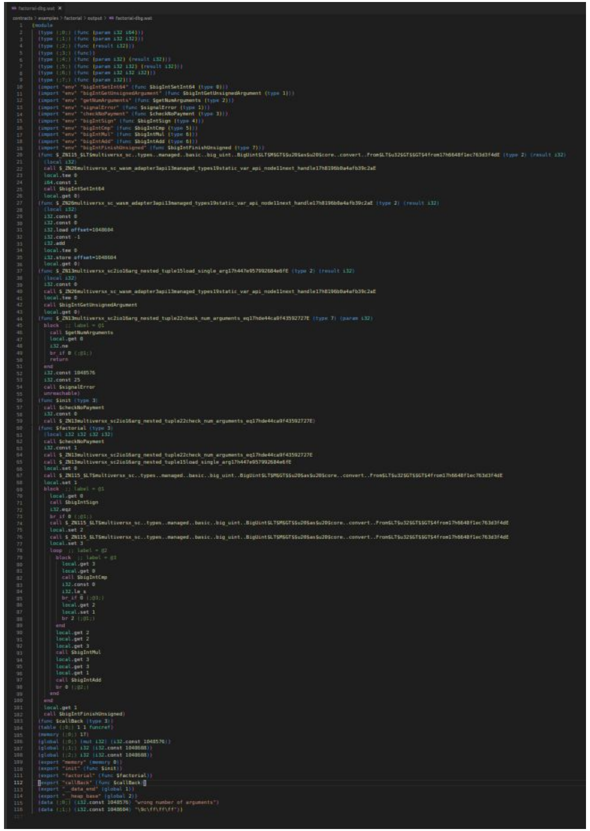
# Understanding WebAssembly

Example SC as .wat:



```
(module
  (type (;0;) (func (param i32 i64)))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func (result i32)))
  (type (;3;) (func))
  (type (;4;) (func (param i32) (result i32)))
  (type (;5;) (func (param i32 i32) (result i32)))
  (type (;6;) (func (param i32 i32 i32)))
  (type (;7;) (func (param i32)))
  (import "env" "bigIntSetInt64" (func (;0;) (type 0)))
  (import "env" "bigIntGetUnsignedArgument" (func (;1;) (type 1)))
  (import "env" "getNumArguments" (func (;2;) (type 2)))
  (import "env" "signalError" (func (;3;) (type 1)))
  (import "env" "checkNoPayment" (func (;4;) (type 3)))
  (import "env" "bigIntSign" (func (;5;) (type 4)))
  (import "env" "bigIntCmp" (func (;6;) (type 5)))
  (import "env" "bigIntMul" (func (;7;) (type 6)))
  (import "env" "bigIntAdd" (func (;8;) (type 6)))
  (import "env" "bigIntFinishUnsigned" (func (;9;) (type 7)))
  (func (;10;) (type 2) (result i32)
    (local i32)
    call 11
    local.tee 0
    i64.const 1
    call 0
    local.get 0)
  (func (;11;) (type 2) (result i32)
    (local i32)
    i32.const 0
    i32.const 0
    i32.load offset=1048604
    i32.const -1
    i32.add
    local.tee 0
    i32.store offset=1048604
    local.get 0)
  (func (;12;) (type 2) (result i32)
    (local i32)
    i32.const 0
    call 11
    local.tee 0
    call 1
    local.get 0)
  (func (;13;) (type 7) (param i32)
    block  ;; label = @1
      call 2
      local.get 0
      i32.ne
      br_if 0 (;@1;)
      return
    end
    i32.const 1048576
    i32.const 25
    call 3
    unreachable)
  (func (;14;) (type 3)
    call 4
    i32.const 0
    call 13)
  (func (;15;) (type 3)
    (local i32 i32 i32 i32)
    call 4
    i32.const 1
    call 13
    call 12
    local.set 0
    call 10
    local.set 1
    block  ;; label = @1
      local.get 0
      call 5
      i32.eqz
      br_if 0 (;@1;)
      call 10
      local.set 2
      call 10
      local.set 3
      loop  ;; label = @2
        block  ;; label = @3
          local.get 3
          local.get 0
          call 6
          i32.const 0
          i32.le_s
          br_if 0 (;@3;)
          local.get 2
          local.set 1
          br 2 (;@1;)
        end
        local.get 2
        local.get 2
        local.get 3
        call 7
        local.get 3
        local.get 3
        local.get 1
        call 8
        br 0 (;@2;)
      end
    end
    local.get 1
    call 9)
  (func (;16;) (type 3))
  (table (;0;) 1 1 funcref)
  (memory (;0;) 17)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048608))
  (global (;2;) i32 (i32.const 1048608))
  (export "memory" (memory 0))
  (export "init" (func 14))
  (export "factorial" (func 15))
  (export "callBack" (func 16))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2))
  (data (;0;) (i32.const 1048576) "wrong number of arguments")
  (data (;1;) (i32.const 1048604) "\9c\ff\ff\ff"))
```

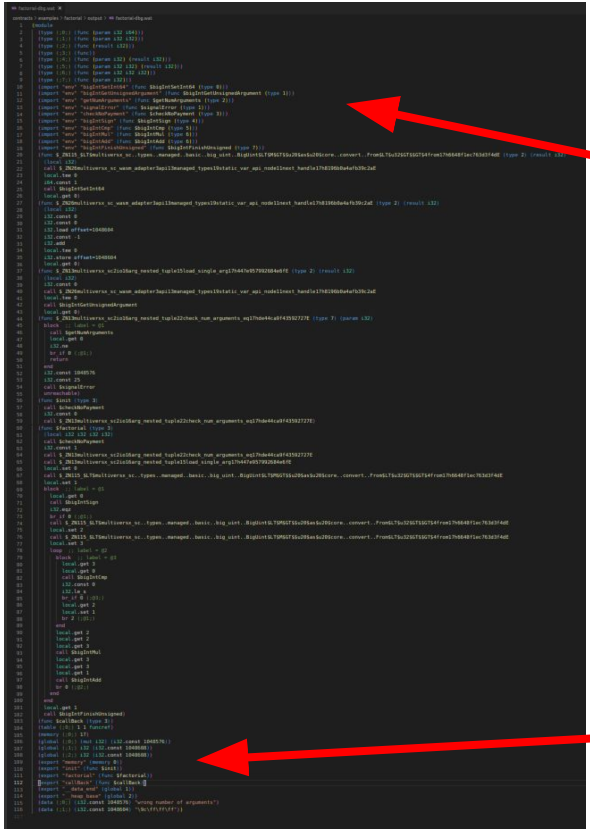
in production –


factorial-dbg.wat
contracts > examples > factorial > output > factorial-dbg.wat

```
(module
  (type (;0;) (func (param i32 i64)))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func (result i32)))
  (type (;3;) (func))
  (type (;4;) (func (param i32) (result i32)))
  (type (;5;) (func (param i32 i32) (result i32)))
  (type (;6;) (func (param i32 i32 i32)))
  (type (;7;) (func (param i32)))
  (import "env" "bigIntSetInt64" (func $bigIntSetInt64 (type 0)))
  (import "env" "bigIntGetUnsignedArgument" (func $bigIntGetUnsignedArgument (type 1)))
  (import "env" "getNumArguments" (func $getNumArguments (type 2)))
  (import "env" "signalError" (func $signalError (type 1)))
  (import "env" "checkNoPayment" (func $checkNoPayment (type 3)))
  (import "env" "bigIntSign" (func $bigIntSign (type 4)))
  (import "env" "bigIntCmp" (func $bigIntCmp (type 5)))
  (import "env" "bigIntMul" (func $bigIntMul (type 6)))
  (import "env" "bigIntAdd" (func $bigIntAdd (type 6)))
  (import "env" "bigIntFinishUnsigned" (func $bigIntFinishUnsigned (type 7)))
  (func $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE (type 2) (result i32)
    (local i32)
    call $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE
    local.tee 0
    i64.const 1
    call $bigIntSetInt64
    local.get 0)
  (func $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE (type 2) (result i32)
    (local i32)
    i32.const 0
    i32.const 0
    i32.load offset=1048604
    i32.const -1
    i32.add
    local.tee 0
    i32.store offset=1048604
    local.get 0)
  (func $_ZN13multiversx_sc2io16arg_nested_tuple15load_single_arg17h447e957992684e6fE (type 2) (result i32)
    (local i32)
    i32.const 0
    call $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE
    local.tee 0
    call $bigIntGetUnsignedArgument
    local.get 0)
  (func $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E (type 7) (param i32)
    block  ;; label = @1
      call $getNumArguments
      local.get 0
      i32.ne
      br_if 0 (;@1;)
      return
    end
    i32.const 1048576
    i32.const 25
    call $signalError
    unreachable)
  (func $init (type 3)
    call $checkNoPayment
    i32.const 0
    call $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E)
  (func $factorial (type 3)
    (local i32 i32 i32 i32)
    call $checkNoPayment
    i32.const 1
    call $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E
    call $_ZN13multiversx_sc2io16arg_nested_tuple15load_single_arg17h447e957992684e6fE
    local.set 0
    call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
    local.set 1
    block  ;; label = @1
      local.get 0
      call $bigIntSign
      i32.eqz
      br_if 0 (;@1;)
      call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
      local.set 2
      call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
      local.set 3
      loop  ;; label = @2
        block  ;; label = @3
          local.get 3
          local.get 0
          call $bigIntCmp
          i32.const 0
          i32.le_s
          br_if 0 (;@3;)
          local.get 2
          local.set 1
          br 2 (;@1;)
        end
        local.get 2
        local.get 2
        local.get 3
        call $bigIntMul
        local.get 3
        local.get 3
        local.get 1
        call $bigIntAdd
        br 0 (;@2;)
      end
    end
    local.get 1
    call $bigIntFinishUnsigned)
  (func $callBack (type 3))
  (table (;0;) 1 1 funcref)
  (memory (;0;) 17)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048608))
  (global (;2;) i32 (i32.const 1048608))
  (export "memory" (memory 0))
  (export "init" (func $init))
  (export "factorial" (func $factorial))
  (export "callBack" (func $callBack))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2))
  (data (;0;) (i32.const 1048576) "wrong number of arguments")
  (data (;1;) (i32.const 1048604) "\9c\ff\ff\ff"))
```

– with debug symbols

# Understanding WebAssembly



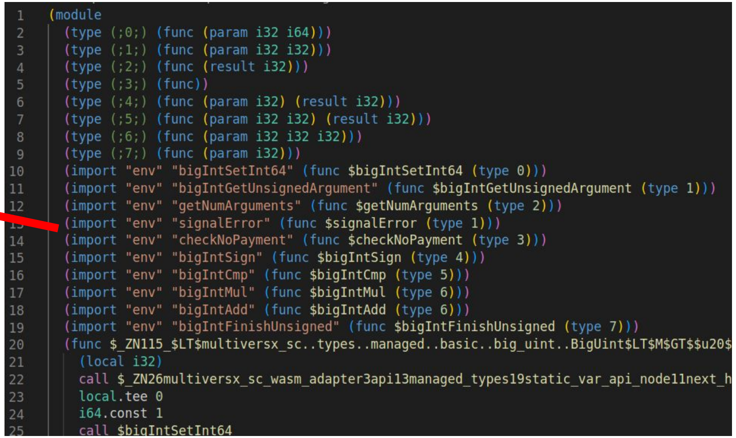
factorial-dbg.wat ×

contracts > examples > factorial > output > factorial-dbg.wat

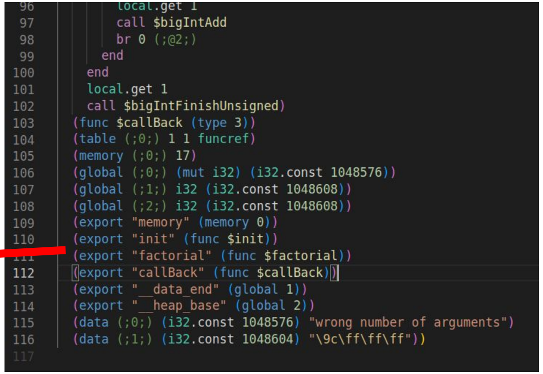
```
(module
  (type (;0;) (func (param i32 i64)))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func (result i32)))
  (type (;3;) (func))
  (type (;4;) (func (param i32) (result i32)))
  (type (;5;) (func (param i32 i32) (result i32)))
  (type (;6;) (func (param i32 i32 i32)))
  (type (;7;) (func (param i32)))
  (import "env" "bigIntSetInt64" (func $bigIntSetInt64 (type 0)))
  (import "env" "bigIntGetUnsignedArgument" (func $bigIntGetUnsignedArgument (type 1)))
  (import "env" "getNumArguments" (func $getNumArguments (type 2)))
  (import "env" "signalError" (func $signalError (type 1)))
  (import "env" "checkNoPayment" (func $checkNoPayment (type 3)))
  (import "env" "bigIntSign" (func $bigIntSign (type 4)))
  (import "env" "bigIntCmp" (func $bigIntCmp (type 5)))
  (import "env" "bigIntMul" (func $bigIntMul (type 6)))
  (import "env" "bigIntAdd" (func $bigIntAdd (type 6)))
  (import "env" "bigIntFinishUnsigned" (func $bigIntFinishUnsigned (type 7)))
  (func $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE (type 2) (result i32)
    (local i32)
    call $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE
    local.tee 0
    i64.const 1
    call $bigIntSetInt64
    local.get 0)
  (func $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE (type 2) (result i32)
    (local i32)
    i32.const 0
    i32.const 0
    i32.load offset=1048604
    i32.const -1
    i32.add
    local.tee 0
    i32.store offset=1048604
    local.get 0)
  (func $_ZN13multiversx_sc2io16arg_nested_tuple15load_single_arg17h447e957992684e6fE (type 2) (result i32)
    (local i32)
    i32.const 0
    call $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_handle17h8196b0a4afb39c2aE
    local.tee 0
    call $bigIntGetUnsignedArgument
    local.get 0)
  (func $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E (type 7) (param i32)
    block  ;; label = @1
      call $getNumArguments
      local.get 0
      i32.ne
      br_if 0 (;@1;)
      return
    end
    i32.const 1048576
    i32.const 25
    call $signalError
    unreachable)
  (func $init (type 3)
    call $checkNoPayment
    i32.const 0
    call $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E)
  (func $factorial (type 3)
    (local i32 i32 i32 i32)
    call $checkNoPayment
    i32.const 1
    call $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E
    call $_ZN13multiversx_sc2io16arg_nested_tuple15load_single_arg17h447e957992684e6fE
    local.set 0
    call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
    local.set 1
    block  ;; label = @1
      local.get 0
      call $bigIntSign
      i32.eqz
      br_if 0 (;@1;)
      call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
      local.set 2
      call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$LT$u32$GT$$GT$4from17h6648f1ec763d3f4dE
      local.set 3
      loop  ;; label = @2
        block  ;; label = @3
          local.get 3
          local.get 0
          call $bigIntCmp
          i32.const 0
          i32.le_s
          br_if 0 (;@3;)
          local.get 2
          local.set 1
          br 2 (;@1;)
        end
        local.get 2
        local.get 2
        local.get 3
        call $bigIntMul
        local.get 3
        local.get 3
        local.get 1
        call $bigIntAdd
        br 0 (;@2;)
      end
    end
    local.get 1
    call $bigIntFinishUnsigned)
  (func $callBack (type 3))
  (table (;0;) 1 1 funcref)
  (memory (;0;) 17)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048608))
  (global (;2;) i32 (i32.const 1048608))
  (export "memory" (memory 0))
  (export "init" (func $init))
  (export "factorial" (func $factorial))
  (export "callBack" (func $callBack))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2))
  (data (;0;) (i32.const 1048576) "wrong number of arguments")
  (data (;1;) (i32.const 1048604) "\9c\ff\ff\ff"))
```

EI
*(imports)*

Endpoints
*(exports)*



```
(module
  (type (;0;) (func (param i32 i64)))
  (type (;1;) (func (param i32 i32)))
  (type (;2;) (func (result i32)))
  (type (;3;) (func))
  (type (;4;) (func (param i32) (result i32)))
  (type (;5;) (func (param i32 i32) (result i32)))
  (type (;6;) (func (param i32 i32 i32)))
  (type (;7;) (func (param i32)))
  (import "env" "bigIntSetInt64" (func $bigIntSetInt64 (type 0)))
  (import "env" "bigIntGetUnsignedArgument" (func $bigIntGetUnsignedArgument (type 1)))
  (import "env" "getNumArguments" (func $getNumArguments (type 2)))
  (import "env" "signalError" (func $signalError (type 1)))
  (import "env" "checkNoPayment" (func $checkNoPayment (type 3)))
  (import "env" "bigIntSign" (func $bigIntSign (type 4)))
  (import "env" "bigIntCmp" (func $bigIntCmp (type 5)))
  (import "env" "bigIntMul" (func $bigIntMul (type 6)))
  (import "env" "bigIntAdd" (func $bigIntAdd (type 6)))
  (import "env" "bigIntFinishUnsigned" (func $bigIntFinishUnsigned (type 7)))
  (func $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$
    (local i32)
    call $_ZN26multiversx_sc_wasm_adapter3api13managed_types19static_var_api_node11next_h
    local.tee 0
    i64.const 1
    call $bigIntSetInt64
```



```
        local.get 1
        call $bigIntAdd
        br 0 (;@2;)
      end
    end
    local.get 1
    call $bigIntFinishUnsigned)
  (func $callBack (type 3))
  (table (;0;) 1 1 funcref)
  (memory (;0;) 17)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048608))
  (global (;2;) i32 (i32.const 1048608))
  (export "memory" (memory 0))
  (export "init" (func $init))
  (export "factorial" (func $factorial))
  (export "callBack" (func $callBack))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2))
  (data (;0;) (i32.const 1048576) "wrong number of arguments")
  (data (;1;) (i32.const 1048604) "\9c\ff\ff\ff"))
```

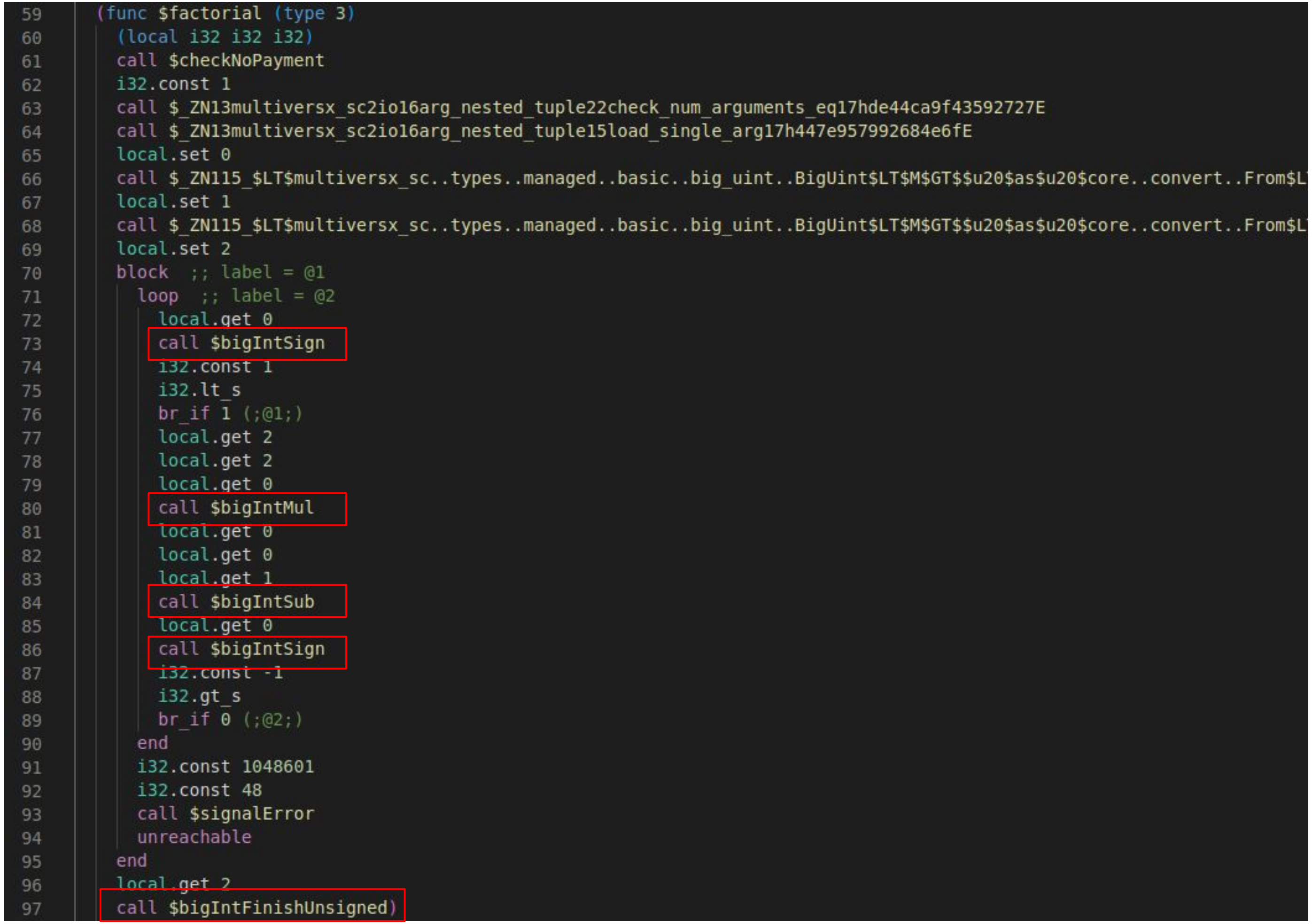
# The sandboxed environment

- Environment interface (EI):
  - Retrieving arguments and payments
  - Pushing results
  - Blockchain info
  - Interactions with other contracts
  - **Managed types**
- Endpoints:
  - init, update, callback
  - A list of all exposed contract functions

# Managed types

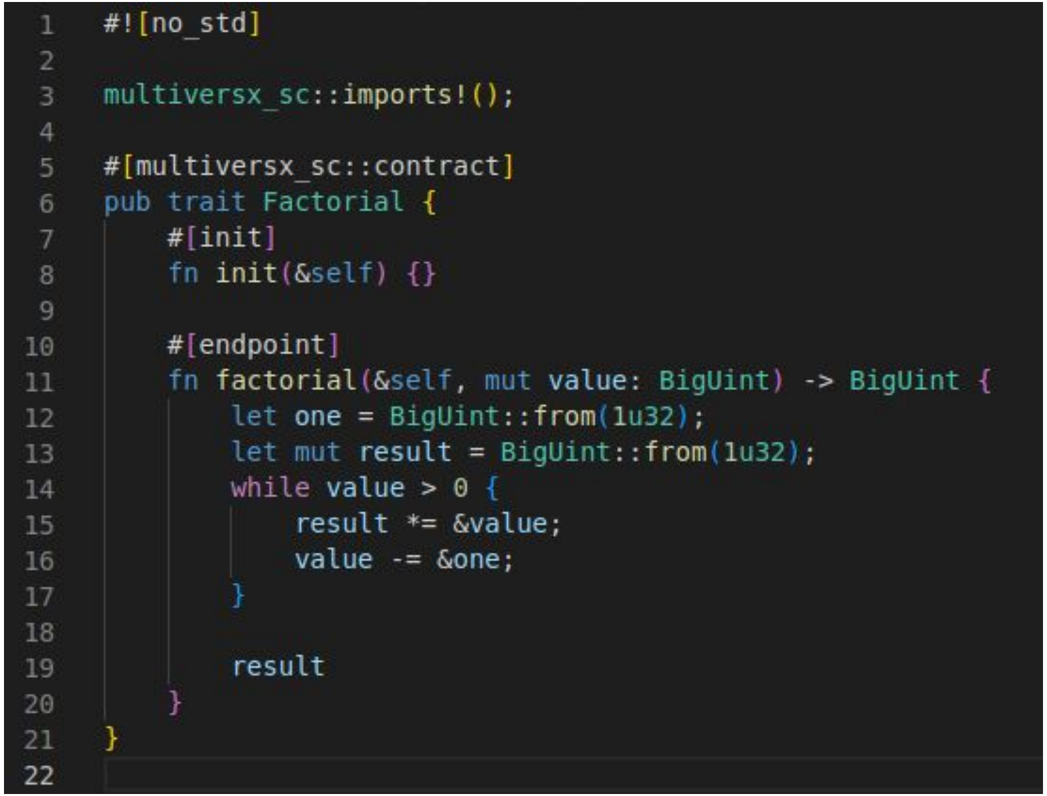
- Maps from handles (think of them as pointers) to data
- The types:
  - Big Int
  - Managed Buffer
  - Managed HashMap
  - Elliptic Curves, Big Float, etc.
- Act like a “virtual heap”
- Offer higher-level atomic operations
- Replace the need for an allocator
- Help write very small contracts

# Managed types in action



```
(func $factorial (type 3)
  (local i32 i32 i32)
  call $checkNoPayment
  i32.const 1
  call $_ZN13multiversx_sc2io16arg_nested_tuple22check_num_arguments_eq17hde44ca9f43592727E
  call $_ZN13multiversx_sc2io16arg_nested_tuple15load_single_arg17h447e957992684e6fE
  local.set 0
  call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$L
  local.set 1
  call $_ZN115_$LT$multiversx_sc..types..managed..basic..big_uint..BigUint$LT$M$GT$$u20$as$u20$core..convert..From$L
  local.set 2
  block  ;; label = @1
    loop  ;; label = @2
      local.get 0
      call $bigIntSign
      i32.const 1
      i32.lt_s
      br_if 1 (;@1;)
      local.get 2
      local.get 2
      local.get 0
      call $bigIntMul
      local.get 0
      local.get 0
      local.get 1
      call $bigIntSub
      local.get 0
      call $bigIntSign
      i32.const -1
      i32.gt_s
      br_if 0 (;@2;)
    end
    i32.const 1048601
    i32.const 48
    call $signalError
    unreachable
  end
  local.get 2
  call $bigIntFinishUnsigned)
```

# Managed types in action



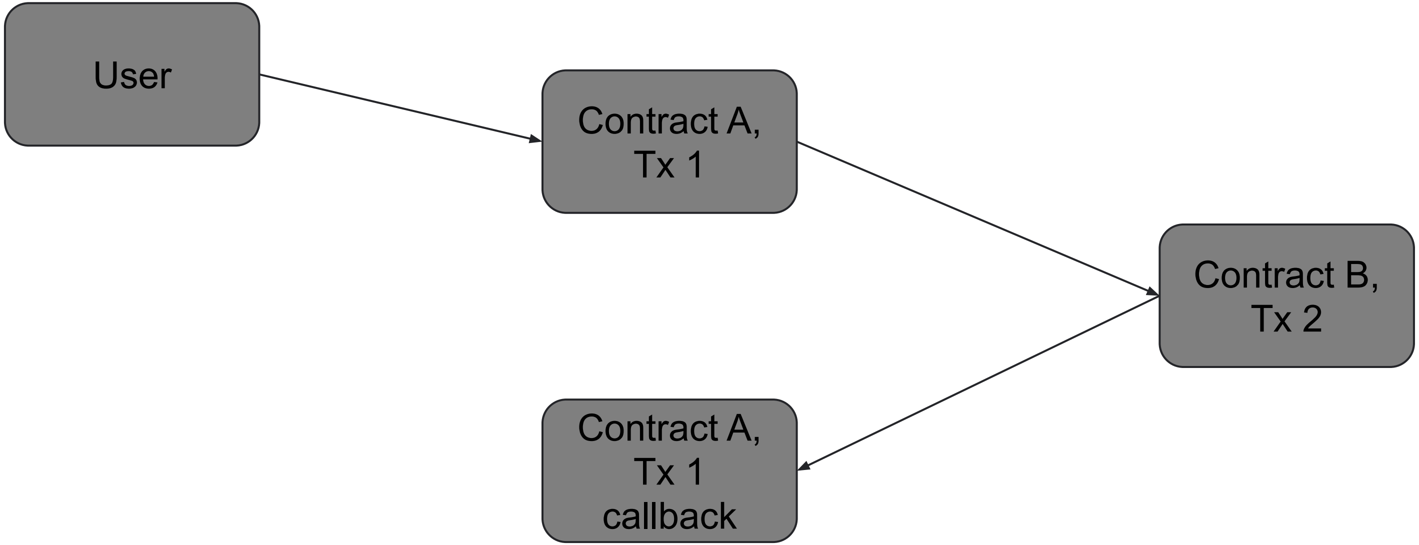
```rust
#![no_std]

multiversx_sc::imports!();

#[multiversx_sc::contract]
pub trait Factorial {
    #[init]
    fn init(&self) {}

    #[endpoint]
    fn factorial(&self, mut value: BigUint) -> BigUint {
        let one = BigUint::from(1u32);
        let mut result = BigUint::from(1u32);
        while value > 0 {
            result *= &value;
            value -= &one;
        }

        result
    }
}

```

# On-Chain Composability

# SC composability in a sharded architecture

- Shards don’t have direct access to each other’s state
- Contract-to-contract calls:
    - Synchronous calls, Ethereum style
        - Only if the contracts are known to be in the same shard
        - Atomic
        - The result of the nested call is available in the calling transaction
    - Asynchronous calls
        - Shard-agnostic (they work identically in the same shard as cross-shard)
        - Not atomic, the calling contract must handle rollback explicitly in case of failure
        - The answer comes back later as a callback transaction

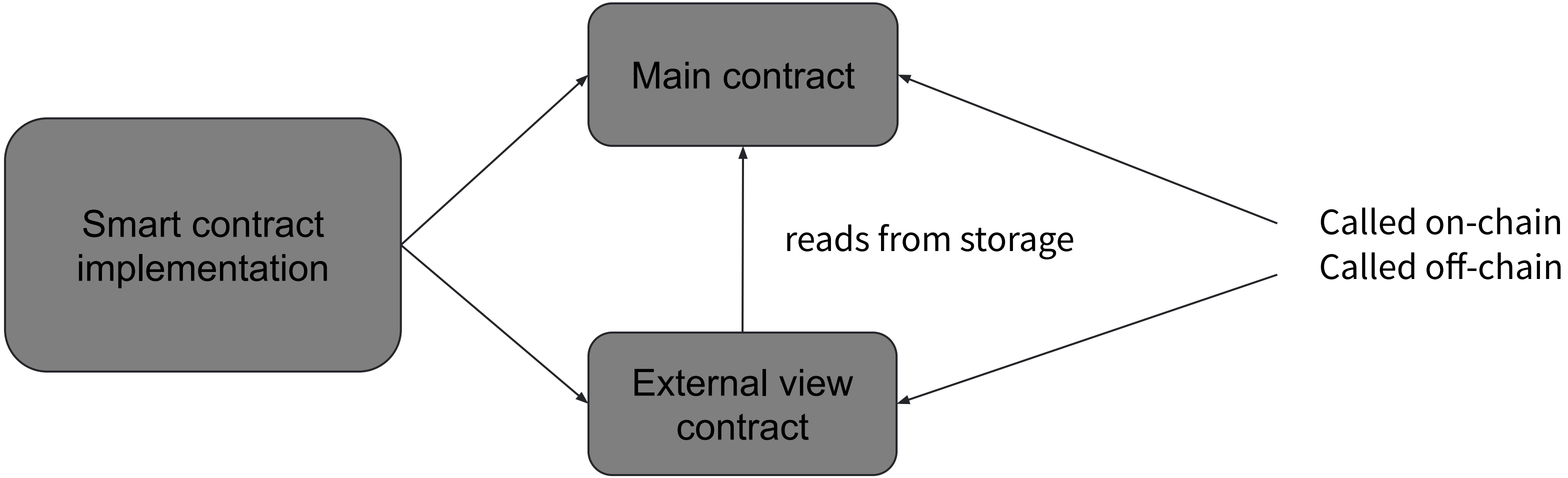
# SC Asynchronous calls explained



# ESDT tokens

- Native
- ESDT ownership stored in account trie (both for SC and EOA)
- No need for *ERC-20*-style *allowance*
- Fungible/Semi-fungible/Non-fungible
- Multiple tokens can be transferred in the same transaction
- Smart contracts can receive and send ESDT tokens
- Alternative to persistence in storage

# External view contracts



Smart contract implementation

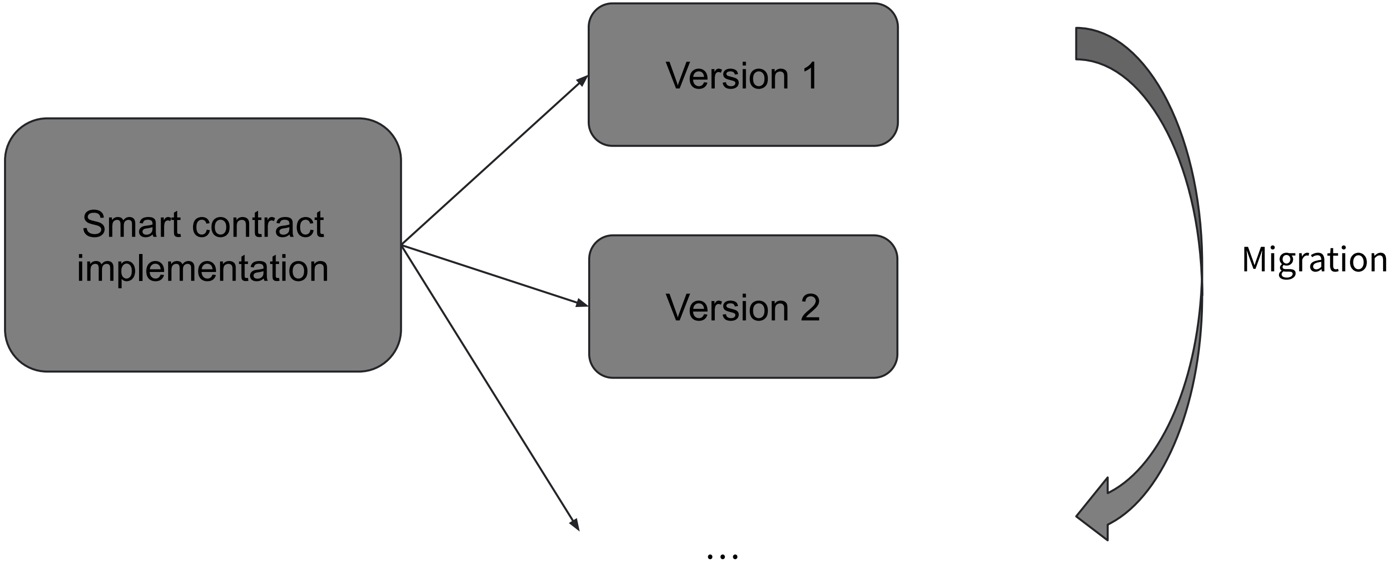
Main contract

External view contract

reads from storage

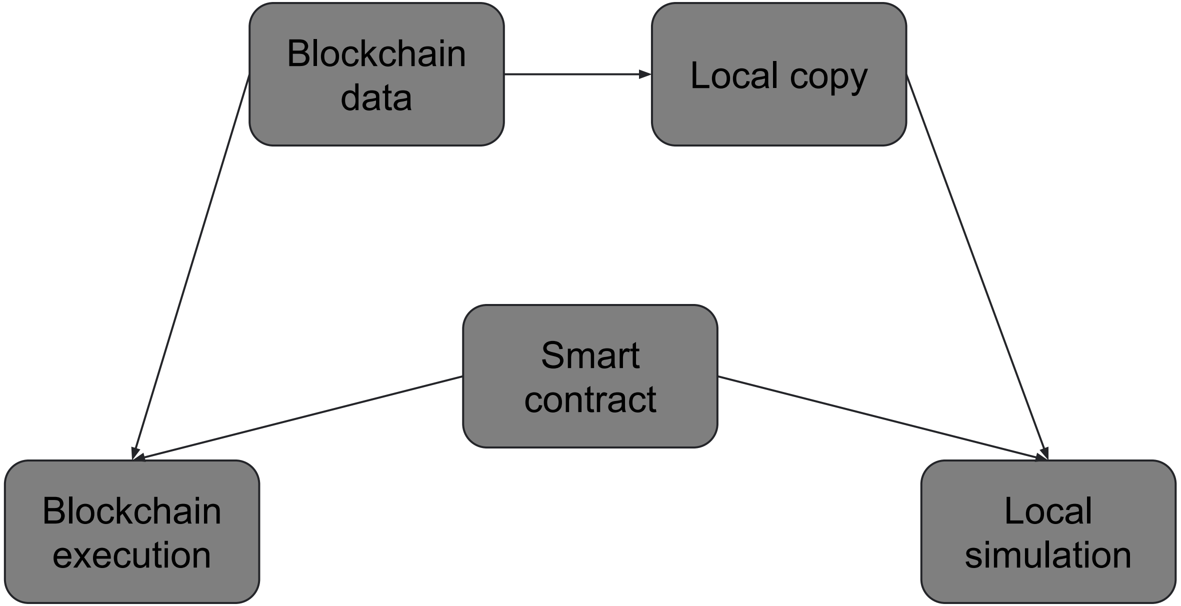
Called on-chain
Called off-chain

# Multi-contracts used for versioning?



# Off-Chain Composability

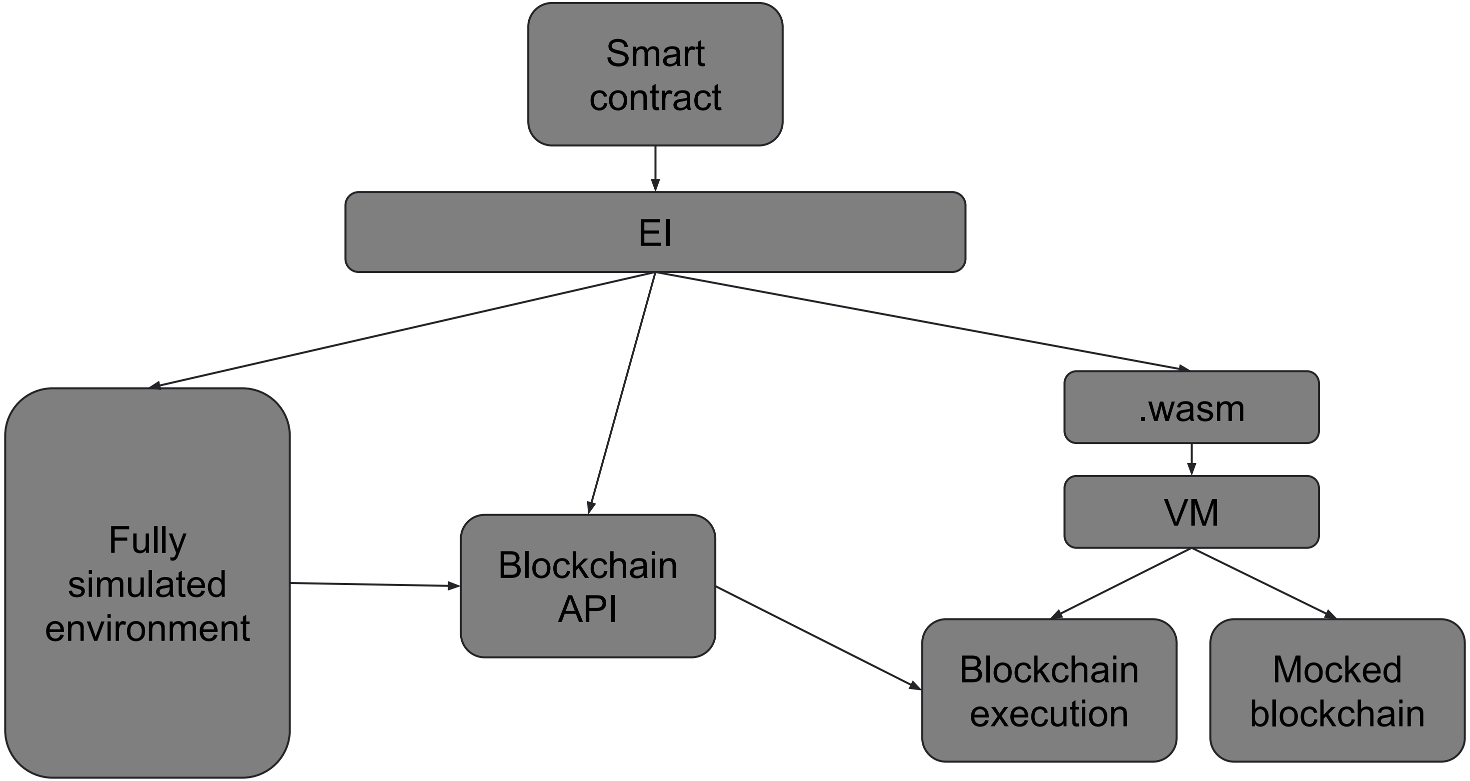
# Different execution environments



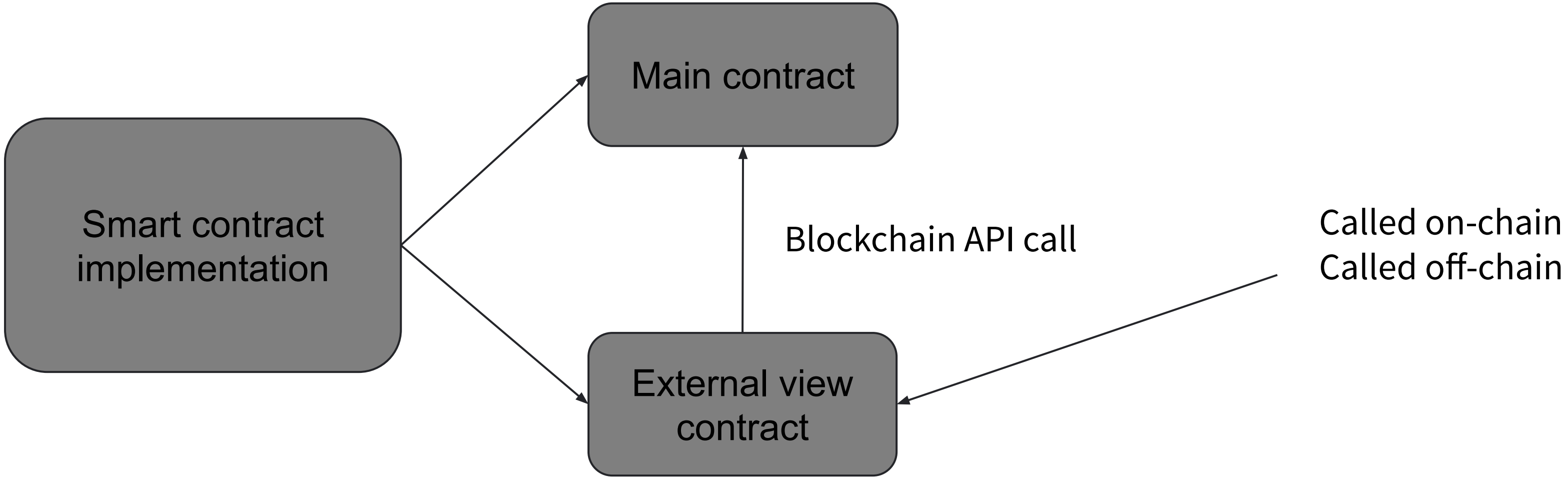
# Many ways to run a smart contract

- On-chain
- Locally, with a real VM, but mocked blockchain
- Locally, in a completely simulated environment
- Locally, but plugging the EI to a blockchain API (“off-chain” SC query)

# Different execution environments



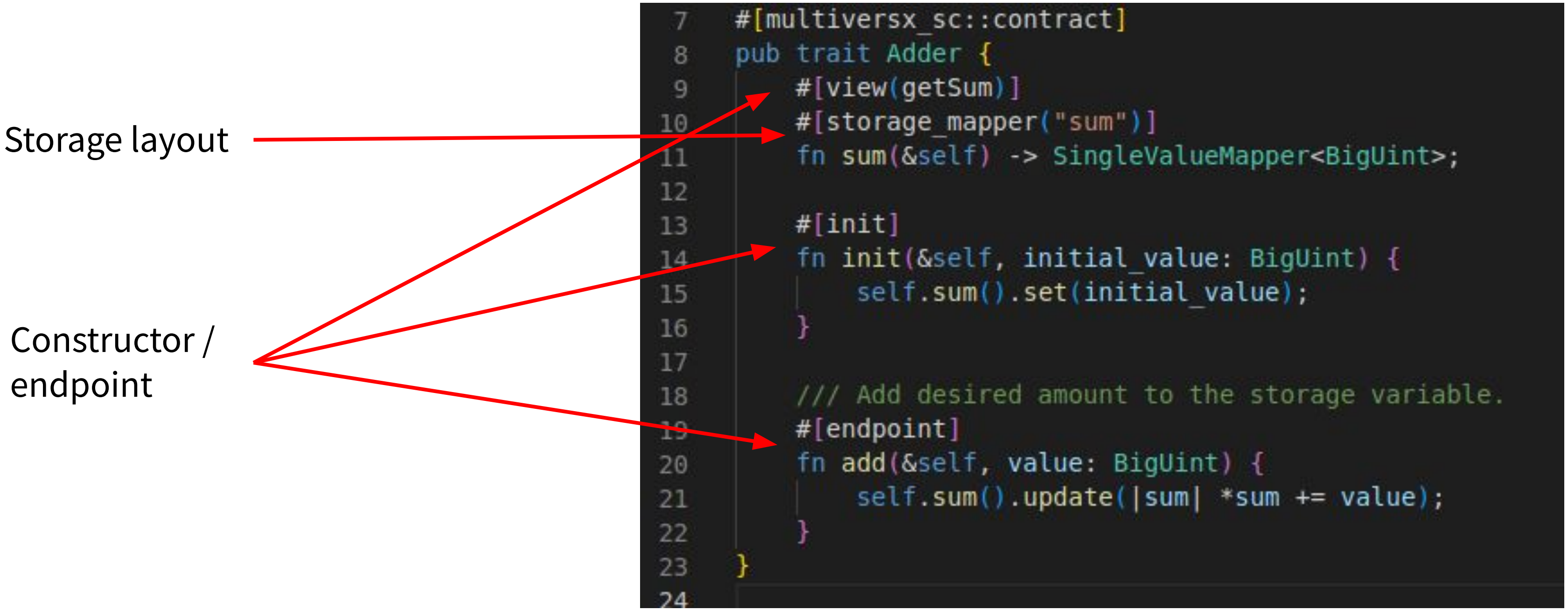
# External view contracts



# So what does it mean to write a smart contract?

# Specifying contract systems

# How we specify smart contracts

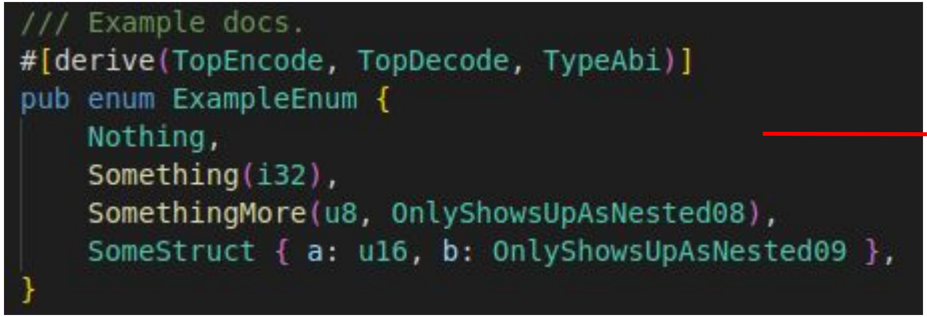


Storage layout

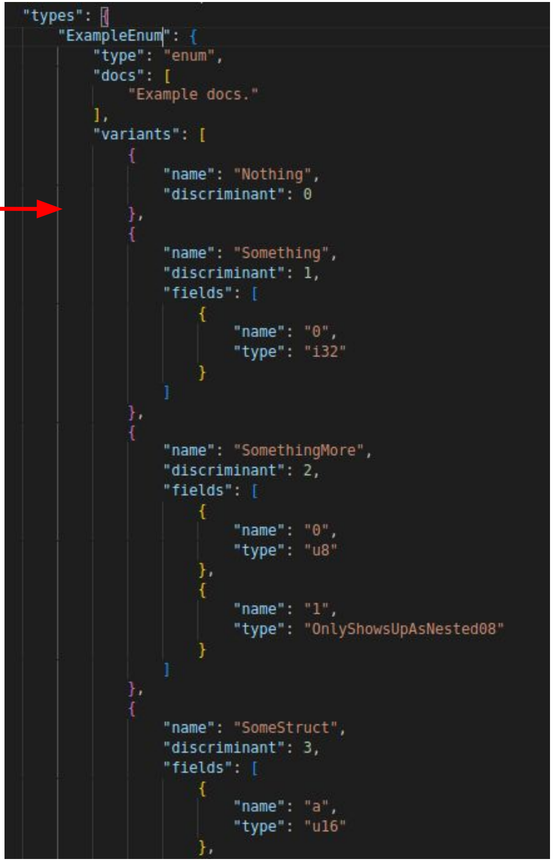
Constructor / endpoint

```rust
#[multiversx_sc::contract]
pub trait Adder {
    #[view(getSum)]
    #[storage_mapper("sum")]
    fn sum(&self) -> SingleValueMapper<BigUint>;

    #[init]
    fn init(&self, initial_value: BigUint) {
        self.sum().set(initial_value);
    }

    /// Add desired amount to the storage variable.
    #[endpoint]
    fn add(&self, value: BigUint) {
        self.sum().update(|sum| *sum += value);
    }
}

```

# How we specify smart contracts

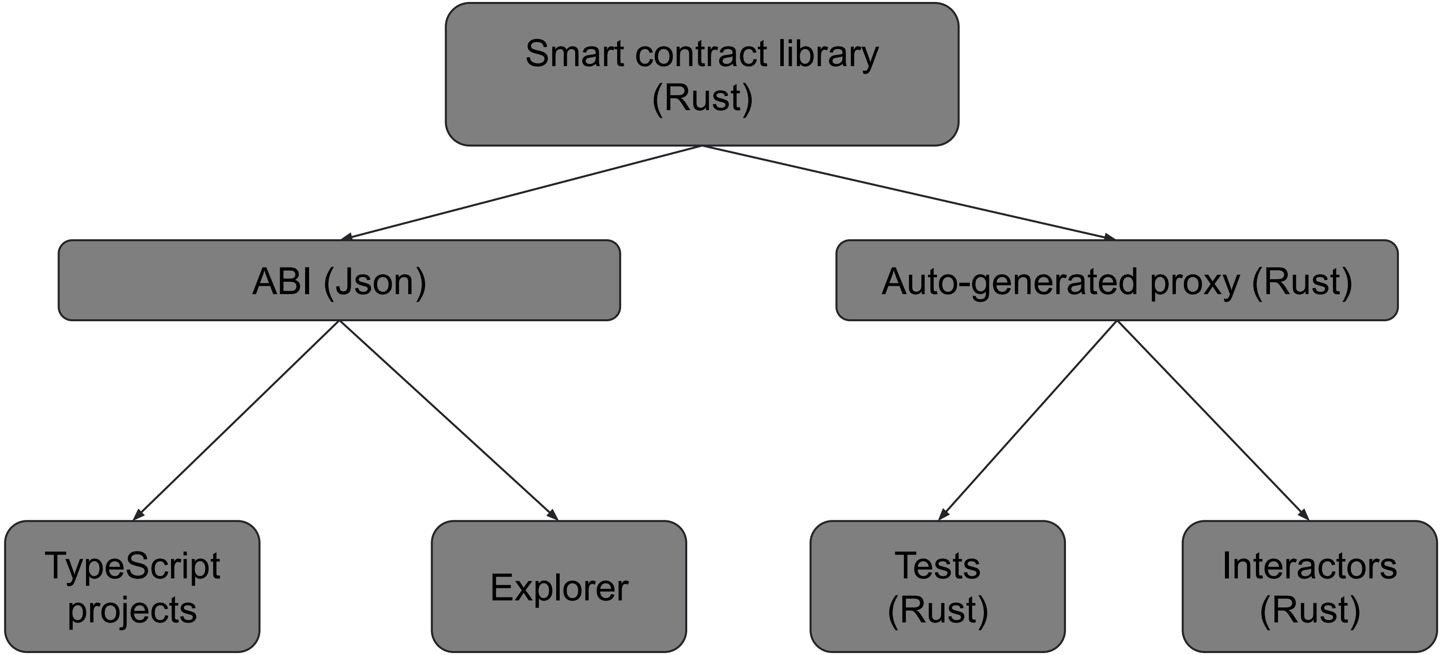


```rust
/// Example docs.
#[derive(TopEncode, TopDecode, TypeAbi)]
pub enum ExampleEnum {
    Nothing,
    Something(i32),
    SomethingMore(u8, OnlyShowsUpAsNested08),
    SomeStruct { a: u16, b: OnlyShowsUpAsNested09 },
}
```



```json
"types": {
    "ExampleEnum": {
        "type": "enum",
        "docs": [
            "Example docs."
        ],
        "variants": [
            {
                "name": "Nothing",
                "discriminant": 0
            },
            {
                "name": "Something",
                "discriminant": 1,
                "fields": [
                    {
                        "name": "0",
                        "type": "i32"
                    }
                ]
            },
            {
                "name": "SomethingMore",
                "discriminant": 2,
                "fields": [
                    {
                        "name": "0",
                        "type": "u8"
                    },
                    {
                        "name": "1",
                        "type": "OnlyShowsUpAsNested08"
                    }
                ]
            },
            {
                "name": "SomeStruct",
                "discriminant": 3,
                "fields": [
                    {
                        "name": "a",
                        "type": "u16"
                    },
```

# Auto-generated ABI and its uses



Smart contract library (Rust)

ABI (Json)

Auto-generated proxy (Rust)

TypeScript projects

Explorer

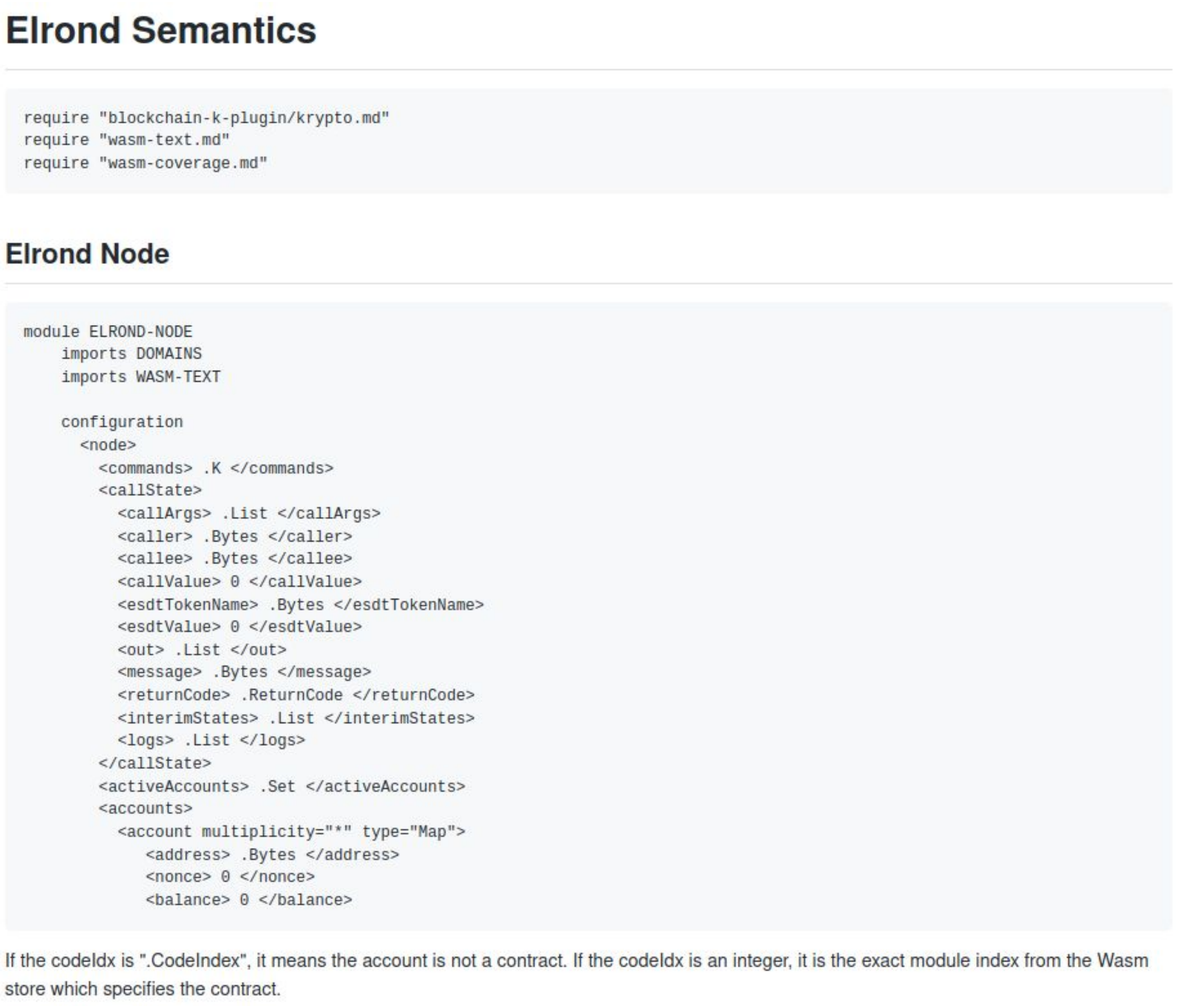
Tests (Rust)

Interactors (Rust)

# … but we can do better!

- Invariants
- Storage consistency checks
- Migrations

# Formal models



# Elrond Semantics

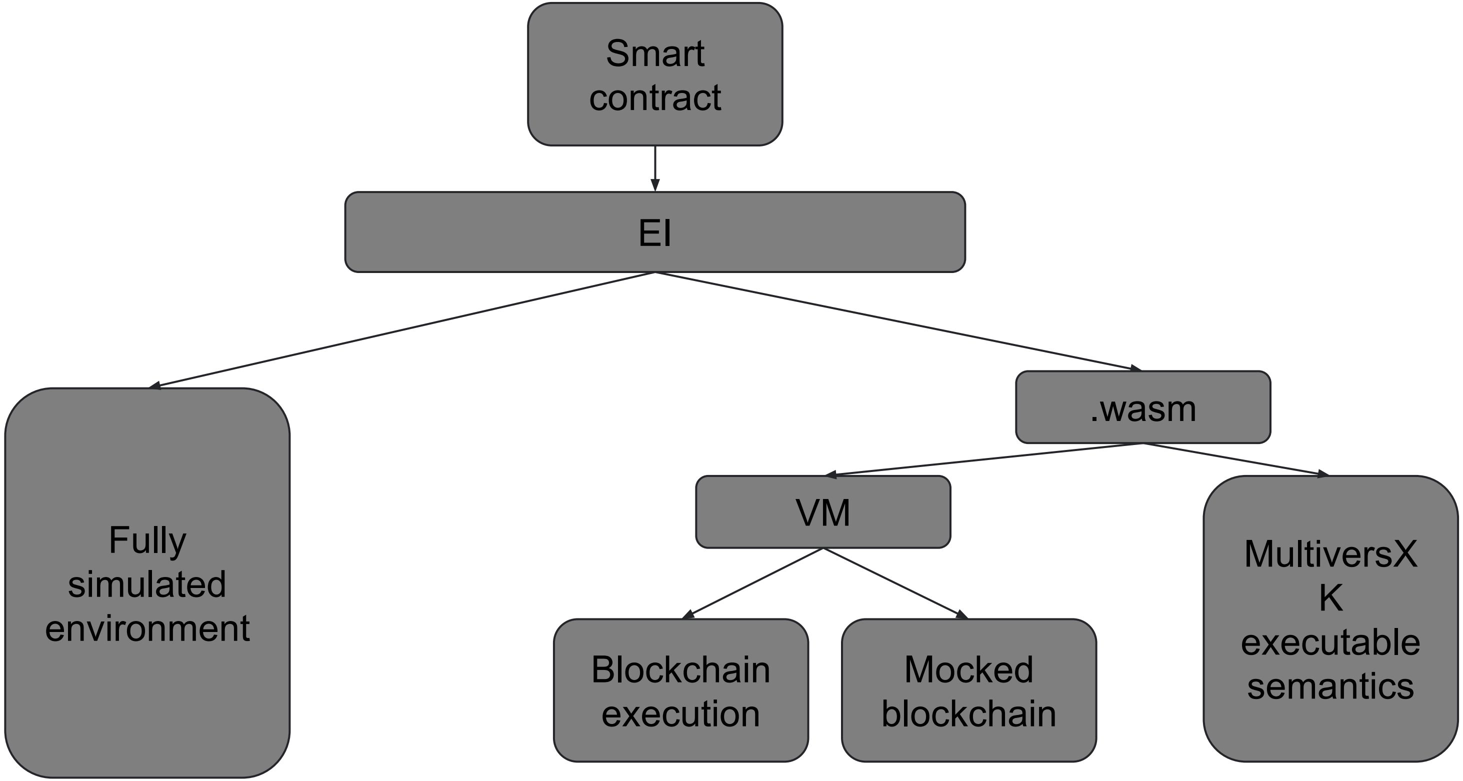
```
require "blockchain-k-plugin/krypto.md"
require "wasm-text.md"
require "wasm-coverage.md"
```

## Elrond Node

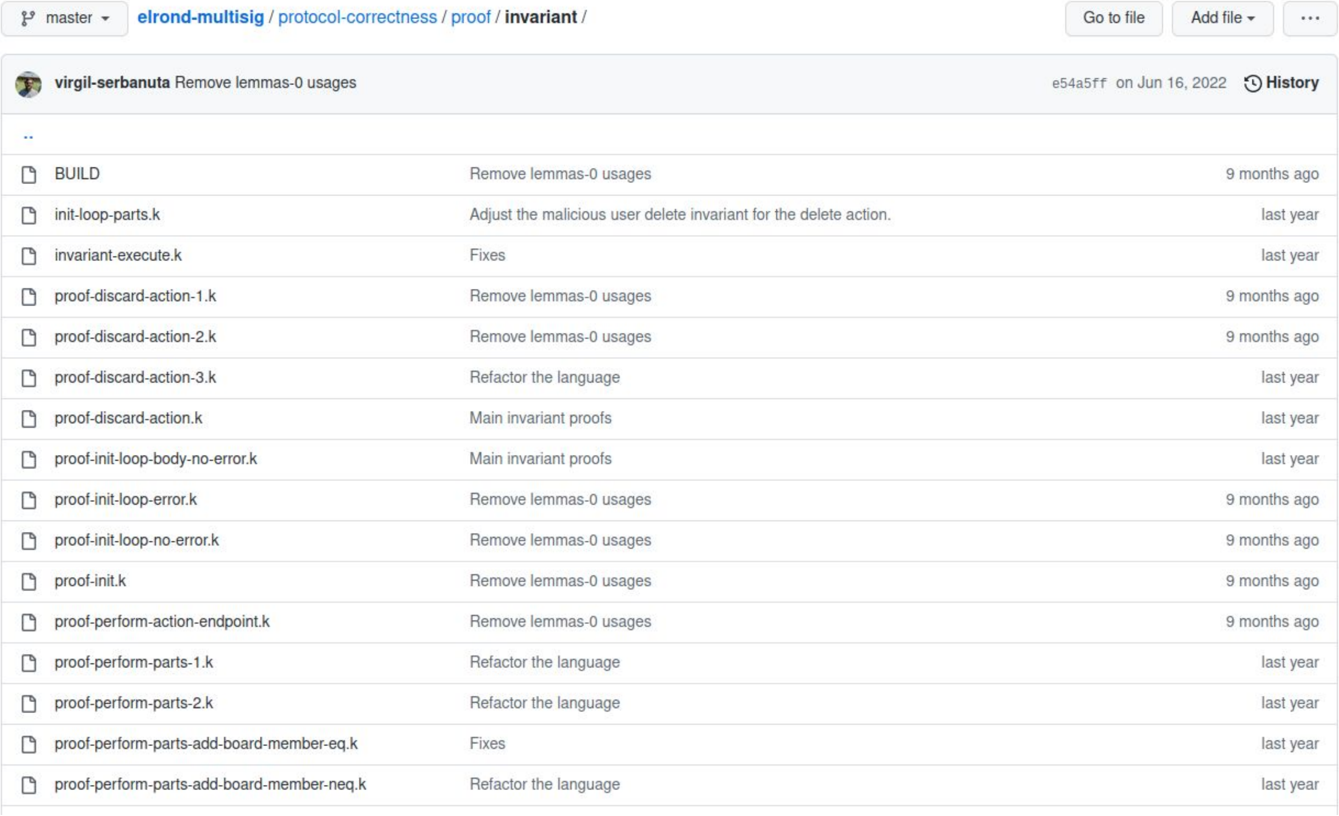
```
module ELROND-NODE
    imports DOMAINS
    imports WASM-TEXT

    configuration
      <node>
        <commands> .K </commands>
        <callState>
          <callArgs> .List </callArgs>
          <caller> .Bytes </caller>
          <callee> .Bytes </callee>
          <callValue> 0 </callValue>
          <esdtTokenName> .Bytes </esdtTokenName>
          <esdtValue> 0 </esdtValue>
          <out> .List </out>
          <message> .Bytes </message>
          <returnCode> .ReturnCode </returnCode>
          <interimStates> .List </interimStates>
          <logs> .List </logs>
        </callState>
        <activeAccounts> .Set </activeAccounts>
        <accounts>
          <account multiplicity="*" type="Map">
             <address> .Bytes </address>
             <nonce> 0 </nonce>
             <balance> 0 </balance>
```

If the codeIdx is ".CodeIndex", it means the account is not a contract. If the codeIdx is an integer, it is the exact module index from the Wasm store which specifies the contract.

# Different execution environments


Smart contract
EI
Fully simulated environment
.wasm
VM
MultiversX K executable semantics
Blockchain execution
Mocked blockchain

# Multisig specification & proofs



elrond-multisig / protocol-correctness / proof / invariant /

virgil-serbanuta Remove lemmas-0 usages — e54a5ff on Jun 16, 2022

..

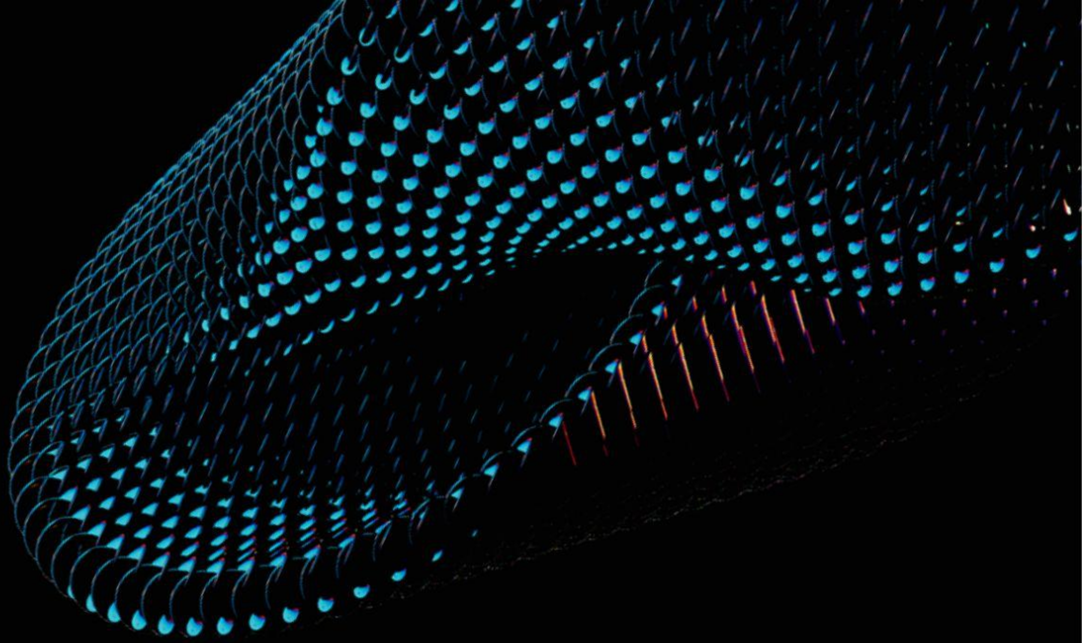
| | | |
|---|---|---|
| BUILD | Remove lemmas-0 usages | 9 months ago |
| init-loop-parts.k | Adjust the malicious user delete invariant for the delete action. | last year |
| invariant-execute.k | Fixes | last year |
| proof-discard-action-1.k | Remove lemmas-0 usages | 9 months ago |
| proof-discard-action-2.k | Remove lemmas-0 usages | 9 months ago |
| proof-discard-action-3.k | Refactor the language | last year |
| proof-discard-action.k | Main invariant proofs | last year |
| proof-init-loop-body-no-error.k | Main invariant proofs | last year |
| proof-init-loop-error.k | Remove lemmas-0 usages | 9 months ago |
| proof-init-loop-no-error.k | Remove lemmas-0 usages | 9 months ago |
| proof-init.k | Remove lemmas-0 usages | 9 months ago |
| proof-perform-action-endpoint.k | Remove lemmas-0 usages | 9 months ago |
| proof-perform-parts-1.k | Refactor the language | last year |
| proof-perform-parts-2.k | Refactor the language | last year |
| proof-perform-parts-add-board-member-eq.k | Fixes | last year |
| proof-perform-parts-add-board-member-neq.k | Refactor the language | last year |

# To conclude …

# To conclude ...

- Specification and execution are independent systems
- A more denotational approach helps with composability & tooling
- Formal models and traditional systems can work together

MultiversX



Thank you for listening!

More information at https://docs.multiversx.com/

Reach out

andrei.marinica@multiversx.com  
https://discord.gg/multiversxbuilders  
https://t.me/MultiversXDevelopers

Follow on:

https://twitter.com/MultiversX