Introduction to Ethereum Virtual Machine

Paweł Bylica
Ethereum Foundation / Erigon Technologies

Ethereum Protocol Fellowship Study Group

March 2025

Who am I?

Paweł Bylica

- Ethereum Foundation / Erigon Technologies
- EVM development & research
- Ewasm / EVMC / evmone
- Ipsilon Team ipsilon.xyz

This talk

1. Ethereum State Transition
2. What is Virtual Machine
3. Ethereum Virtual Machine
4. EVM unique features
5. EVM Object Format

Ethereum State Transition

Transactions and State

See also: https://ethereum.org/en/developers/docs/transactions/

State and Accounts

- State = collection of accounts
- address ⇒ account
- Account:
  - balance (ETH amount) — 256-bit number
  - nonce (some number) — 64-bit number
  - **code** — bytes
  - storage (key ⇒ value) — 32-byte ⇒ 32-byte
- *Comittments*

Accounts duality

EOA

(externally owned account)

- balance
- nonce

Contracts

(passive code)

- balance
- code
- storage
- *nonce*

What is Virtual Machine?

system VM vs ***process*** VM

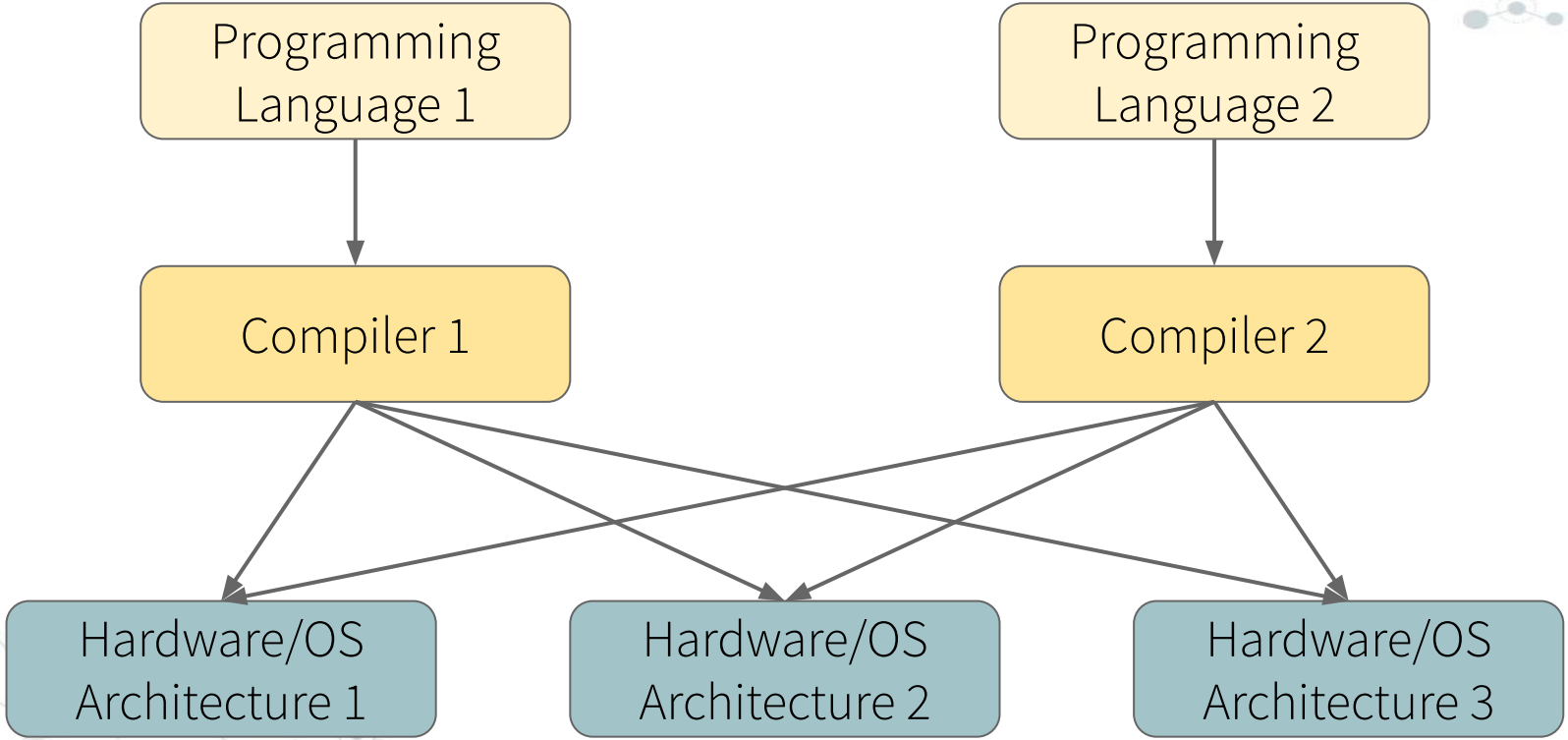
what is VM?

- ~~System Virtual Machine~~
  - Emulates physical machine
  - On top of native OS

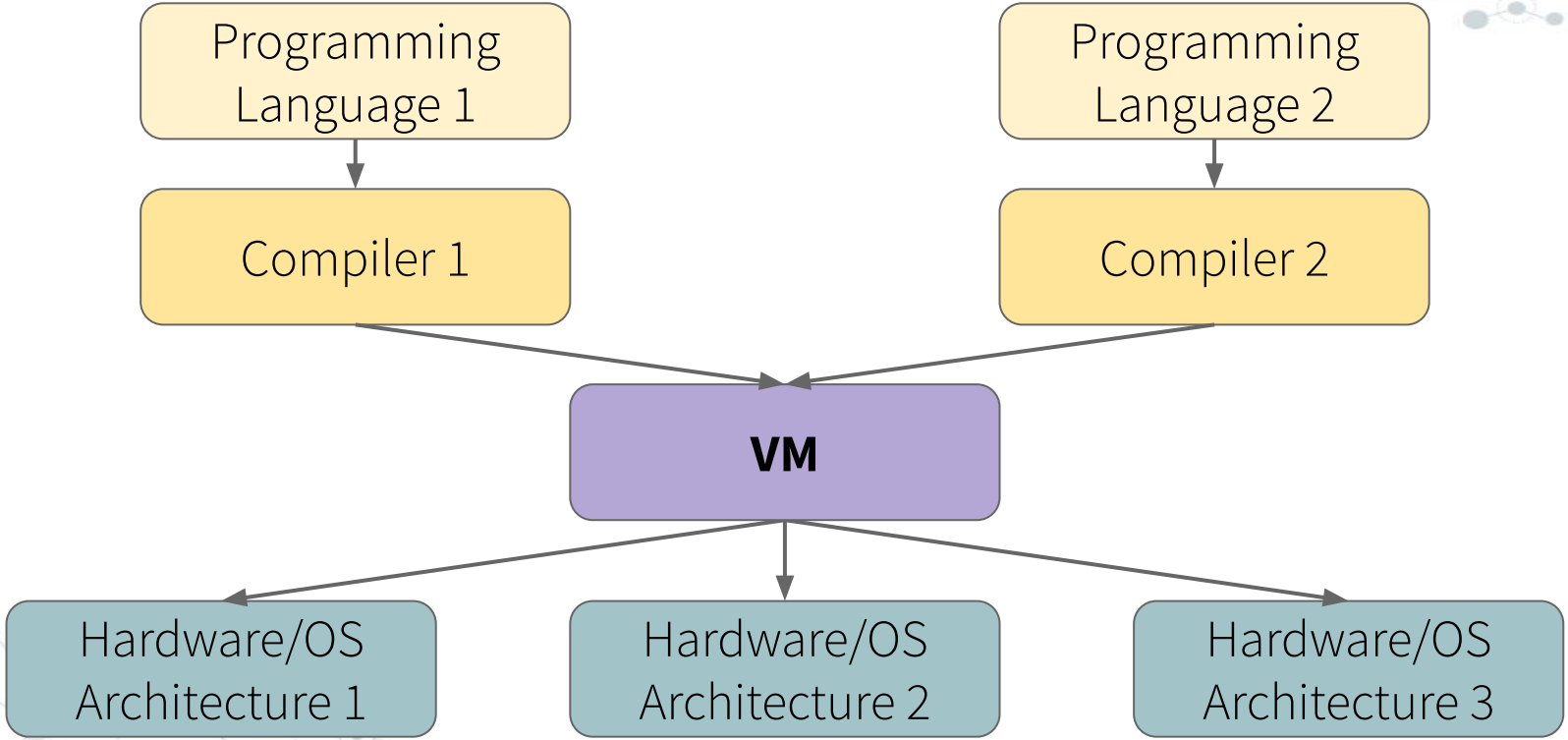
what is VM?

- Process/Application Virtual Machine
  - “Managed Runtime Environment”
  - Examples:
    - JVM
    - .NET
    - WebAssembly

“Classic” Programming Languages



“Managed” Programming Languages



VM architecture

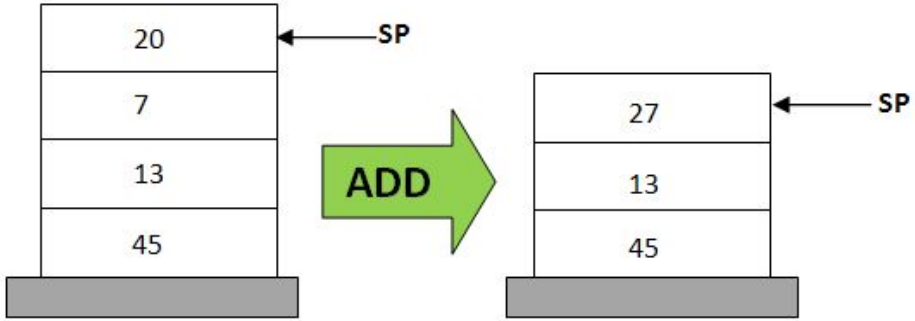
stack based

- “infinite” stack
- short instructions: **bytecode**
- JVM, .NET, wasm, EVM

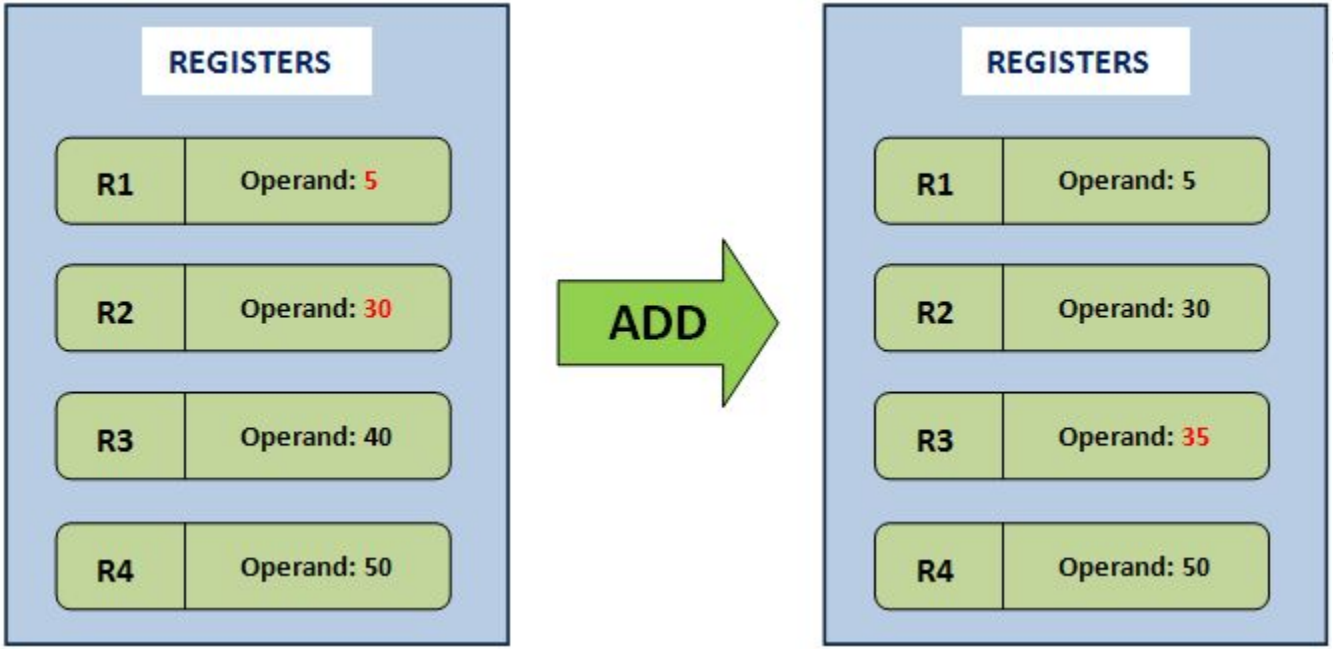
register based

- “infinite” registers
- longer instructions
- Dalvik VM, Lua VM

Stack-based VM



Register-based VM



Ethereum Virtual Machine

EVM

- bytecode
- stack based
- *big* stack items
- no validation
- many *memories*
- *exotic* instructions

EVM Design Goals

- Simplicity
- Total determinism
- Space savings
- Specialization to expected applications
- Simple security
- Optimization-friendliness

The Ethereum virtual machine is the engine in w
between Ethereum and other systems. Note tha
message model - for example, the SIGNEXTEND
contracts and specify gas limits to sub-calls is p

- **Simplicity**: as few and as low-level opcode
  level constructs as possible
- **Total determinism**: there should be absolu
  results should be completely deterministic.
  can be measured so as to compute gas cor
- **Space savings**: EVM assembly should be a
  NOT acceptable)
- **Specialization to expected applications**: t
  byte values, modular arithmetic used in cus
- **Simple security**: it should be easy to come
- **Optimization-friendliness**: it should be eas
  versions of the VM can be built.

256-bit values

- emulate 256-bit arithmetic in software
- fits 256-bit hash

EVM Interpreter Steps

- fetch next instruction (if exists)
- stack underflow
- stack overflow
- gas cost calculation
  + out-of-gas check
- ***do actual work***

EVM Interpreter Steps

- fetch next instruction (if exists) } **5%**
- stack underflow
- stack overflow } **7%**
- gas cost calculation
  + out-of-gas check } **7%**
- ***do actual work***

Instructions overview

evm.codes

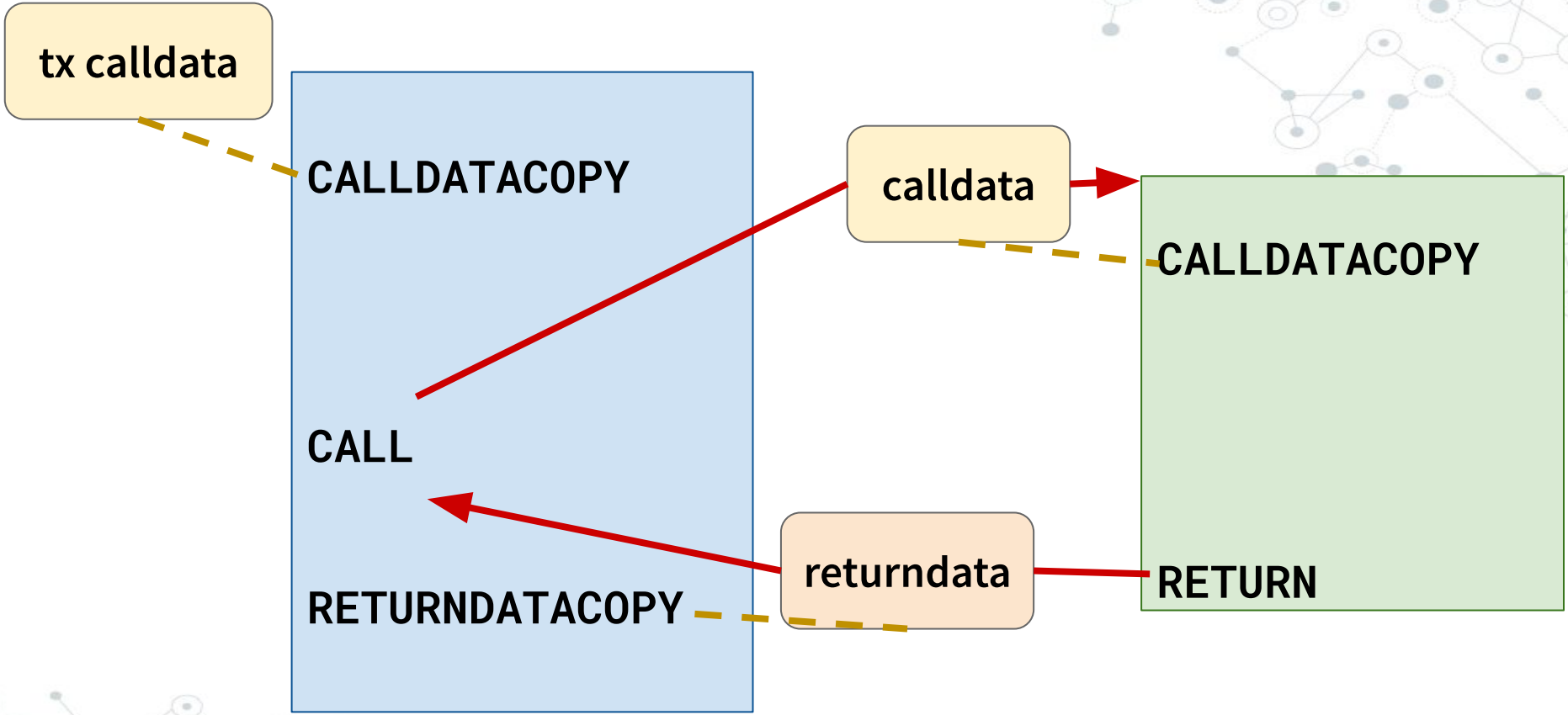
- arithmetic
- modular arithmetic
- bit manipulation
- comparison
- control-flow
- memory access
- transaction info
- block info
- storage
- call
- create
- keccak256

What are *internal* calls?

- **Powerful instructions — invoke other contracts**
  - CALL
  - DELEGATECALL
  - STATICCALL
- **Arguments:**
  - address
  - gas — limit gas provided to callee
  - value — optionally send some ETH
  - input — pass data
- **Results:**
  - return data
  - remaining gas

EVM memories

- `stack` — instruction operands
- `memory` — main volatile memory
- `calldata` — input data
- `returndata` — output from sub-calls
- `storage` — persistent storage


tx calldata
CALLDATACOPY
calldata
CALLDATACOPY
CALL
returndata
RETURN
RETURNDATACOPY

Gas metering ⛽🔥

- execution is limited by gas units
- on all levels:
  - “internal call”
  - transaction
  - block
- instruction cost *some* gas
  - constant
  - complex formula

ADD (0x01)

- pop two items from stack
- add them (256-bit)
- push result to stack
- gas cost: 3

CALLDATACOPY (0x37)

- pop 3 items from stack:
  - calldata offset
  - memory offset
  - size (number of bytes)
- copies part of calldata to memory
- gas cost:
  - 3
  - + 3*size (rounded to 32-byte chunks)
  - + memory expansion cost

NO BYTECODE VALIDATION

Intro to EOF

EVM Object Format

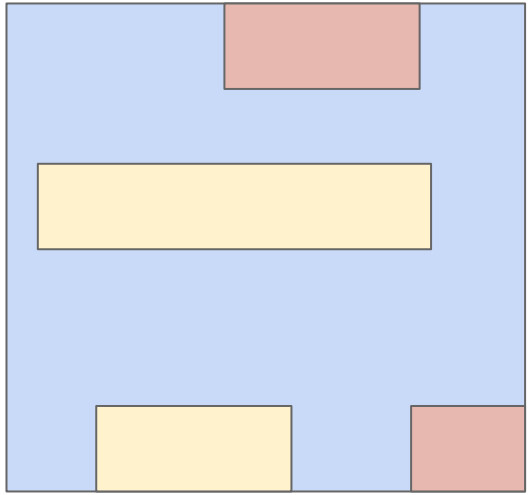
What is EOF?

"an extensible and versioned
container format for the EVM
with a once-off validation at deploy time"

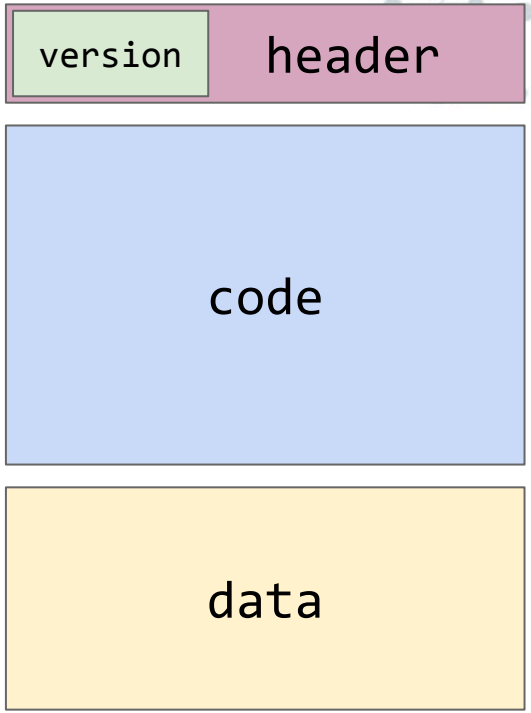
📖 EIP-3540

"Prevents bad behavior with complexity"

legacy



EOF



EVM Interpreter Steps

- ~~fetch next instruction (if exists)~~ } **5%**
- ~~stack underflow~~
- ~~stack overflow~~ } **7%**
- gas cost calculation
  + out-of-gas check } **7%**
- ***do actual work***

Control Flow in EVM

jumps in EVM

- `JUMP`
  - unconditional
  - takes target <u>from EVM stack</u>
- `JUMPI`
  - conditional
  - takes (target, condition) <u>from EVM stack</u>

EVM jumps are “dynamic”

- “static” vs “dynamic”
- direct vs indirect

Control-flow overview

	EVM	wasm	LLVM IR	x86
direct jump		`br`	`br`	`JMP rel32`
indirect jump	`JUMP`	`br_table`	`indirectbr`	`JMP r/m32`
direct call		`call`	`call`	`CALL rel32`
indirect call		`indirect_call`	`call ptr`	`CALL r/m32`

Control-flow overview

	EVM	wasm	LLVM IR	x86
direct jump		`br`	`br`	`JMP rel32`
indirect jump	`JUMP`	`br_table`	`indirectbr`	`JMP r/m32`
direct call		`call`	`call`	`CALL rel32`
indirect call		`indirect_call`	`call ptr`	`CALL r/m32`

Control-flow in EOF

	legacy EVM	EOF
direct jump		`RJUMP` 📖 EIP-4200: Static relative jumps
indirect jump	`JUMP`	`RJUMPV` 📖 EIP-4200: Static relative jumps
direct call		`CALLF` / `RETF` 📖 EIP-4750: EOF Functions
indirect call		

Thank you!