# Ethereum EVM illustrated

exploring some mental models and implementations

Takenobu T.

WIP

Rev. 0.01.0

NOTE

- Please refer to the official documents in detail.
- This information is current as of Mar, 2018.
- Still work in progress.

# Contents

1. Introduction
  - Blockchain
  - World state
  - Account
  - Transaction
  - Message
  - Decentralised database
  - Atomicity and order

2. Virtual machine
  - Ethereum virtual machine (EVM)
  - Message call
  - Exception
  - Gas and fee
  - Input and output
  - Byte order
  - Instruction set
  - Miscellaneous

Appendix A : Implementation
  - Source code in Geth
  - EVM developer utility
  - Solidity ABI

Appendix B : User interface
  - Web3 API
  - Geth, Mist, Solc, Remix, Truffle, ...

Appendix C :
  - Markle tree and RLP
  - Consensus

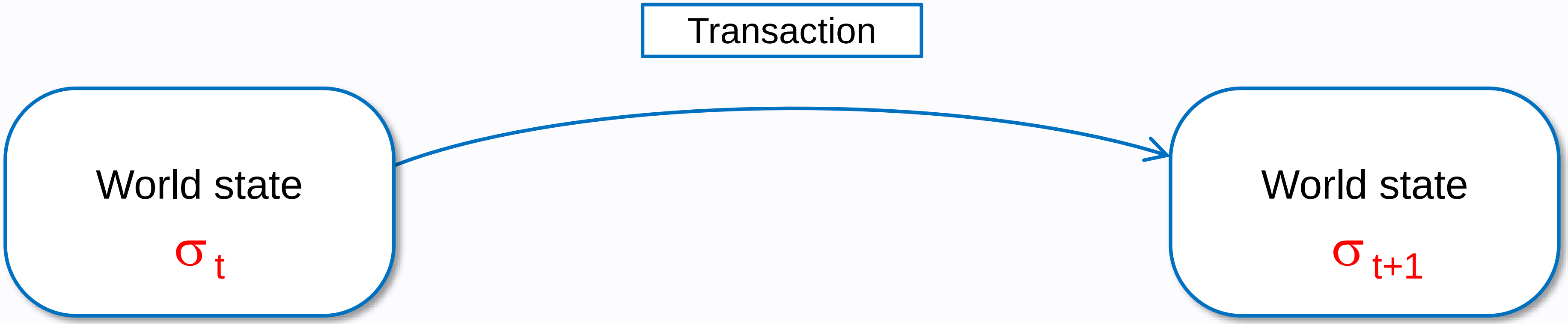
References

# 1. Introduction

# 1. Introduction

## Blockchain

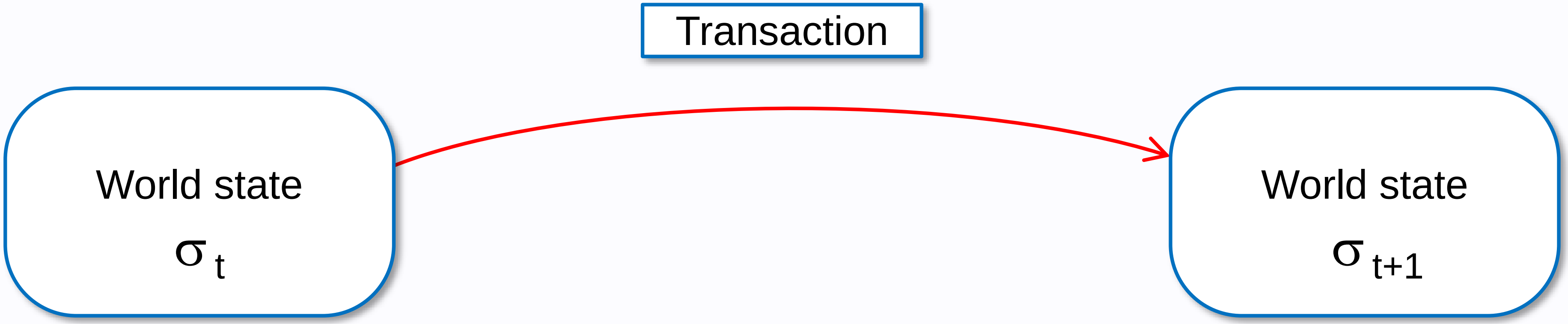
# A transaction-based state machine



Ethereum can be viewed as a transaction-based state machine.

References : [E1] Ch.2, [E3]

# A transaction-based state machine



Transaction

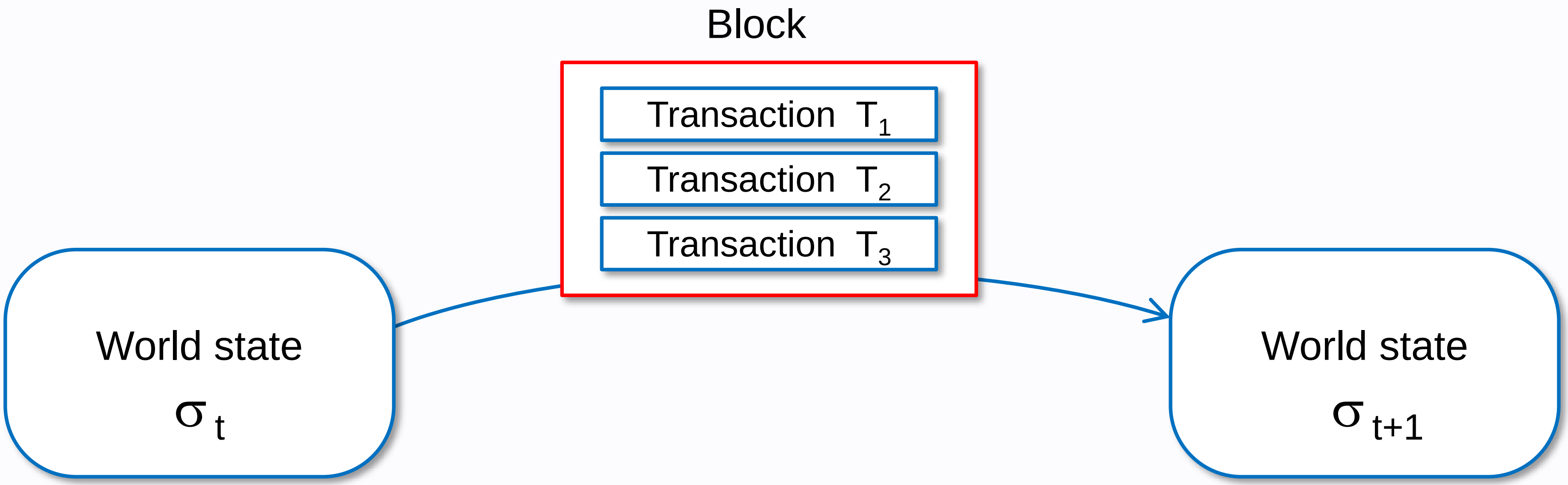
World state $\sigma_t$

World state $\sigma_{t+1}$

A transaction represents a valid arc between two states.

References : [E1] Ch.2, [E3]

# Block and transactions

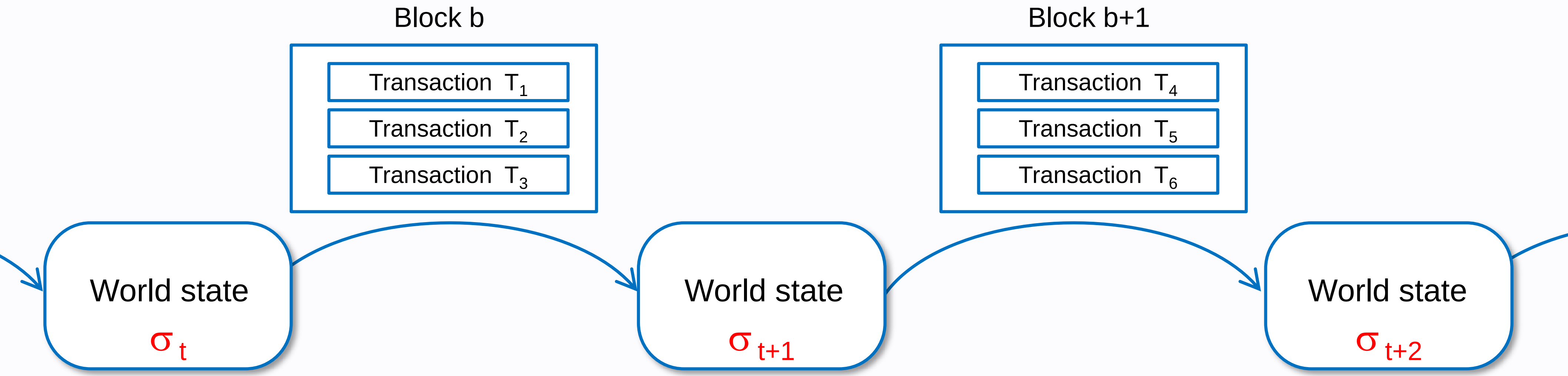


Transactions are collated into blocks.

A block is a package of data.

References : [E1] Ch.2, Ch.4, [E2], [E3]

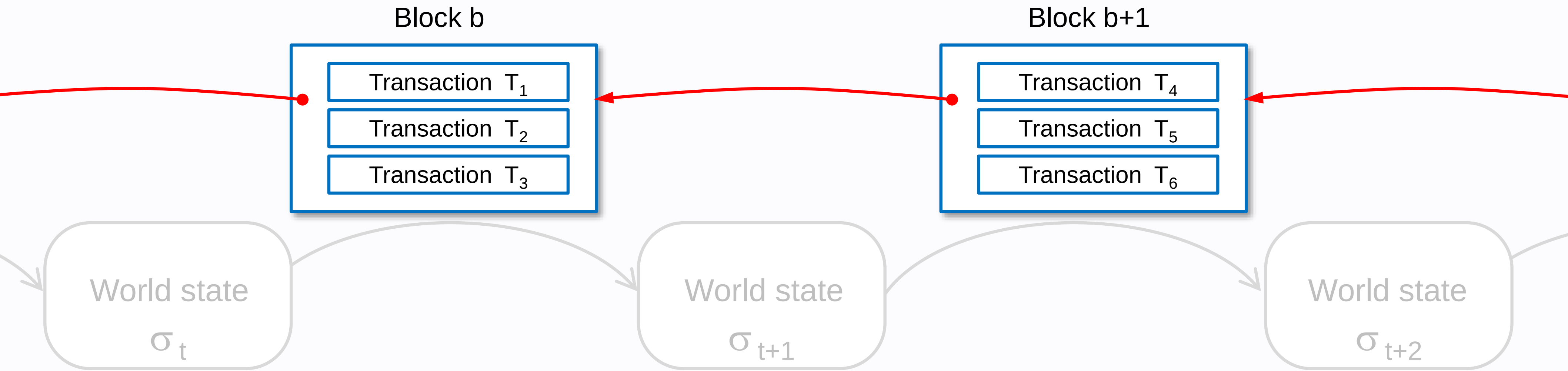
# Chain of states



From the viewpoint of the states,
Ethereum can be seen as a state chain.

References : [E1] Ch.2, Ch.4, [E2], [E3]

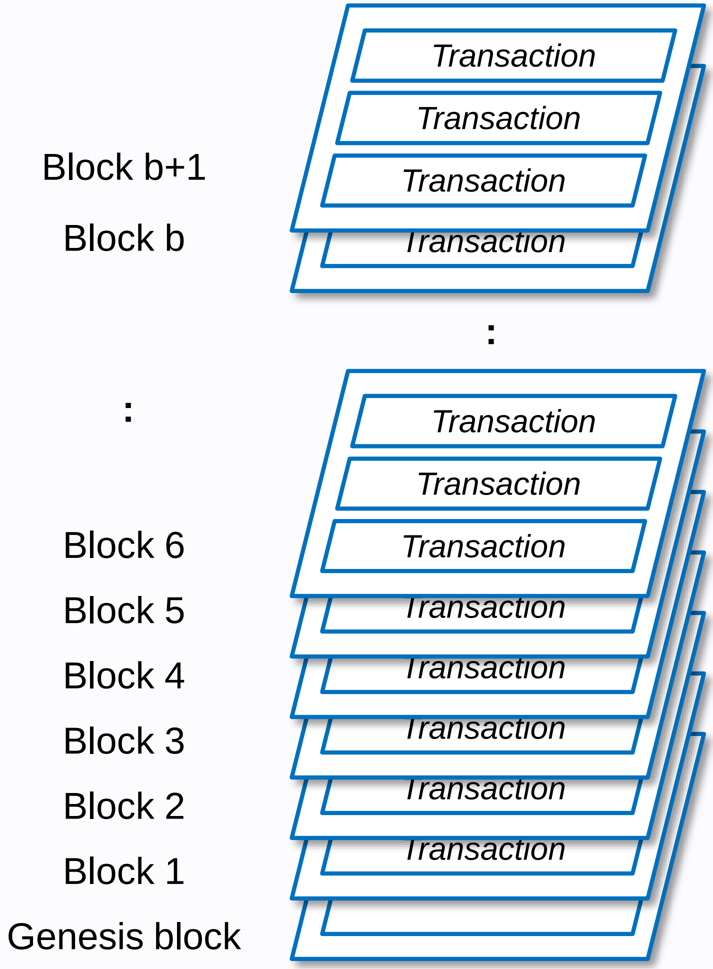
# Chain of blocks: Blockchain



From the viewpoint of the implementation,
Ethereum can also be seen as a chain of blocks, so it is `BLOCKCHAIN`.

References : [E1] Ch.2, Ch.4, [E2], [E3], [W3]

# Stack of transactions : Ledger



From the viewpoint of the ledger,
Ethereum can also be seen as a stack of transactions.

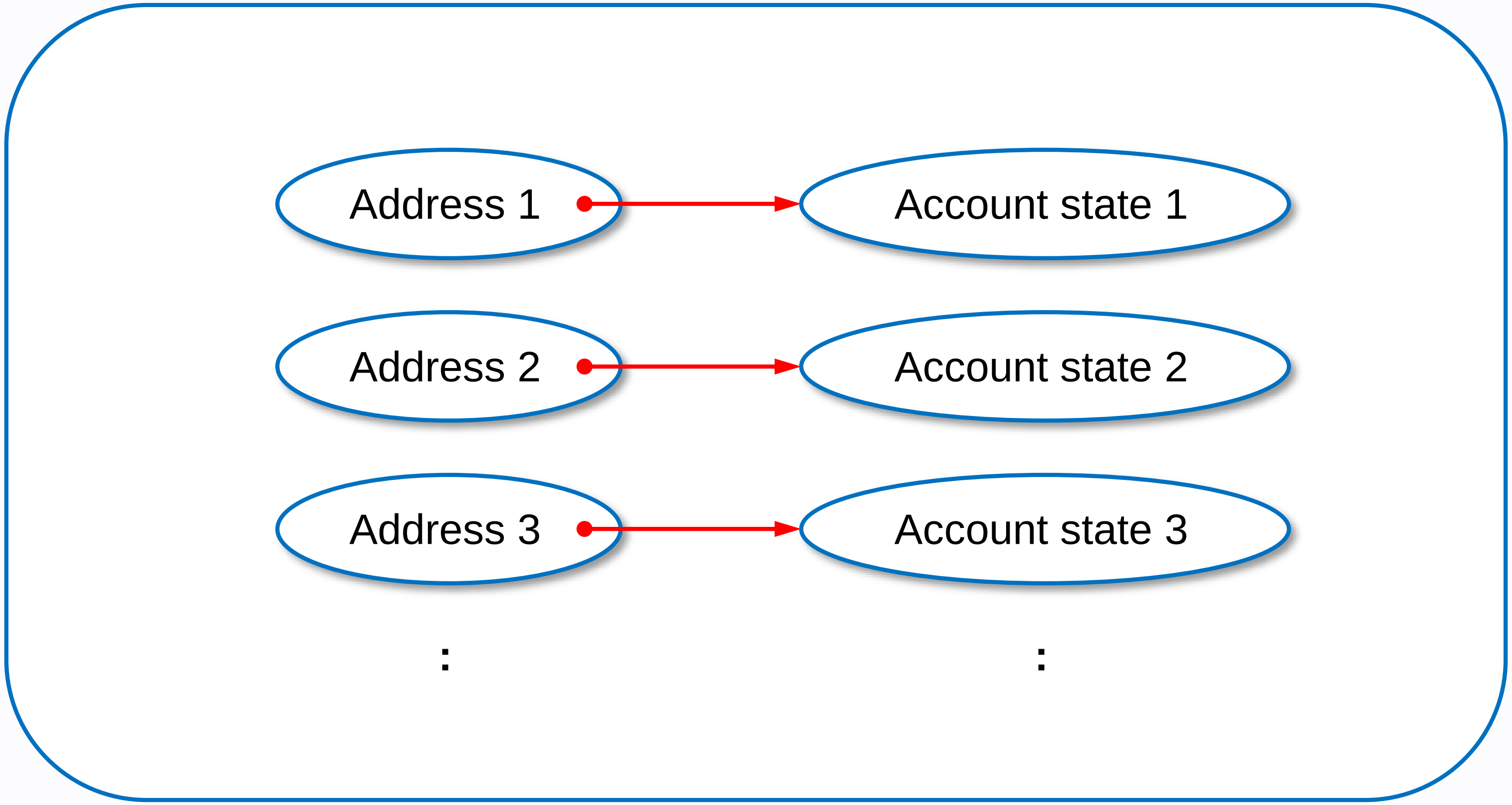
References : [E1] Ch.2, Ch.4, [E2], [E3], [W3]

# 1. Introduction

## World state

# World state

World state $\sigma_t$



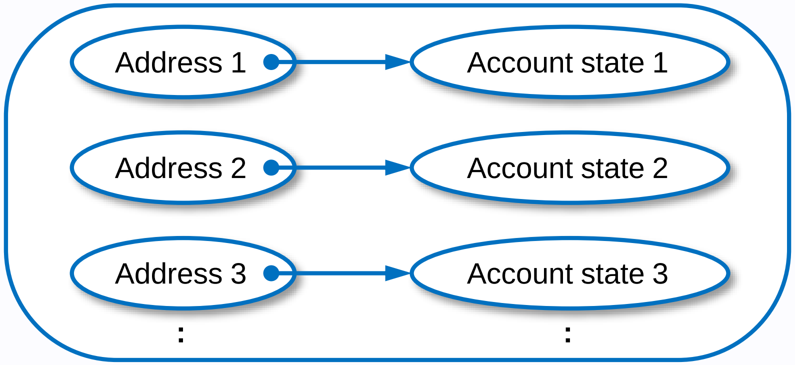
The world state is a mapping between address and account state.

References : [E1] Ch.4, [E2]

# Several views of world state

Mapping view

World state



Address 1 → Account state 1

Address 2 → Account state 2

Address 3 → Account state 3

: :

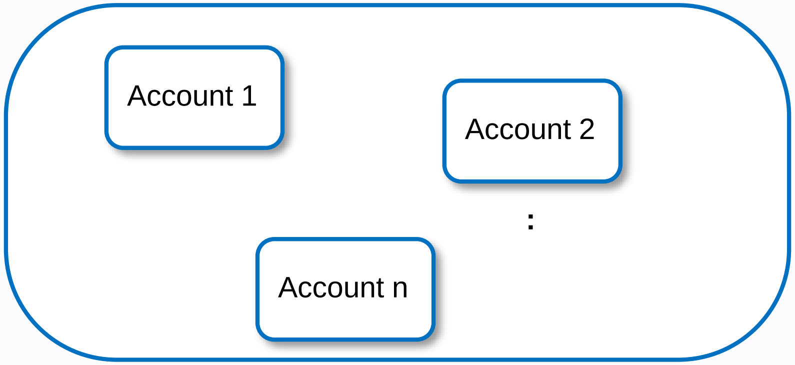
Table view

World state

| | |
|---|---|
| Address 1 | Account state 1 |
| Address 2 | Account state 2 |
| : | : |
| Address n | Account state n |

Object view

World state



Account 1

Account 2

:

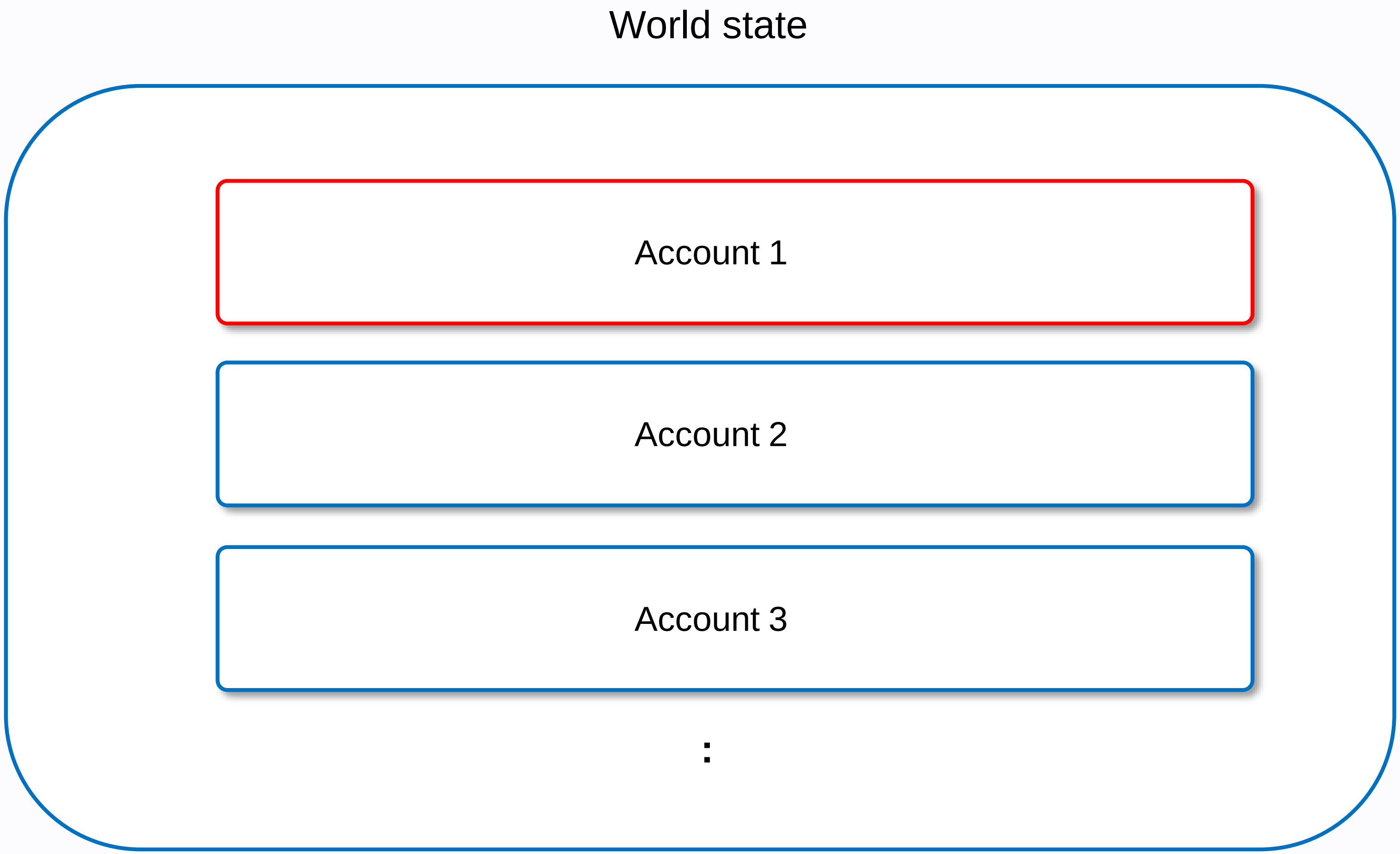
Account n

References : [E1] Ch.4

1. Introduction

## Account

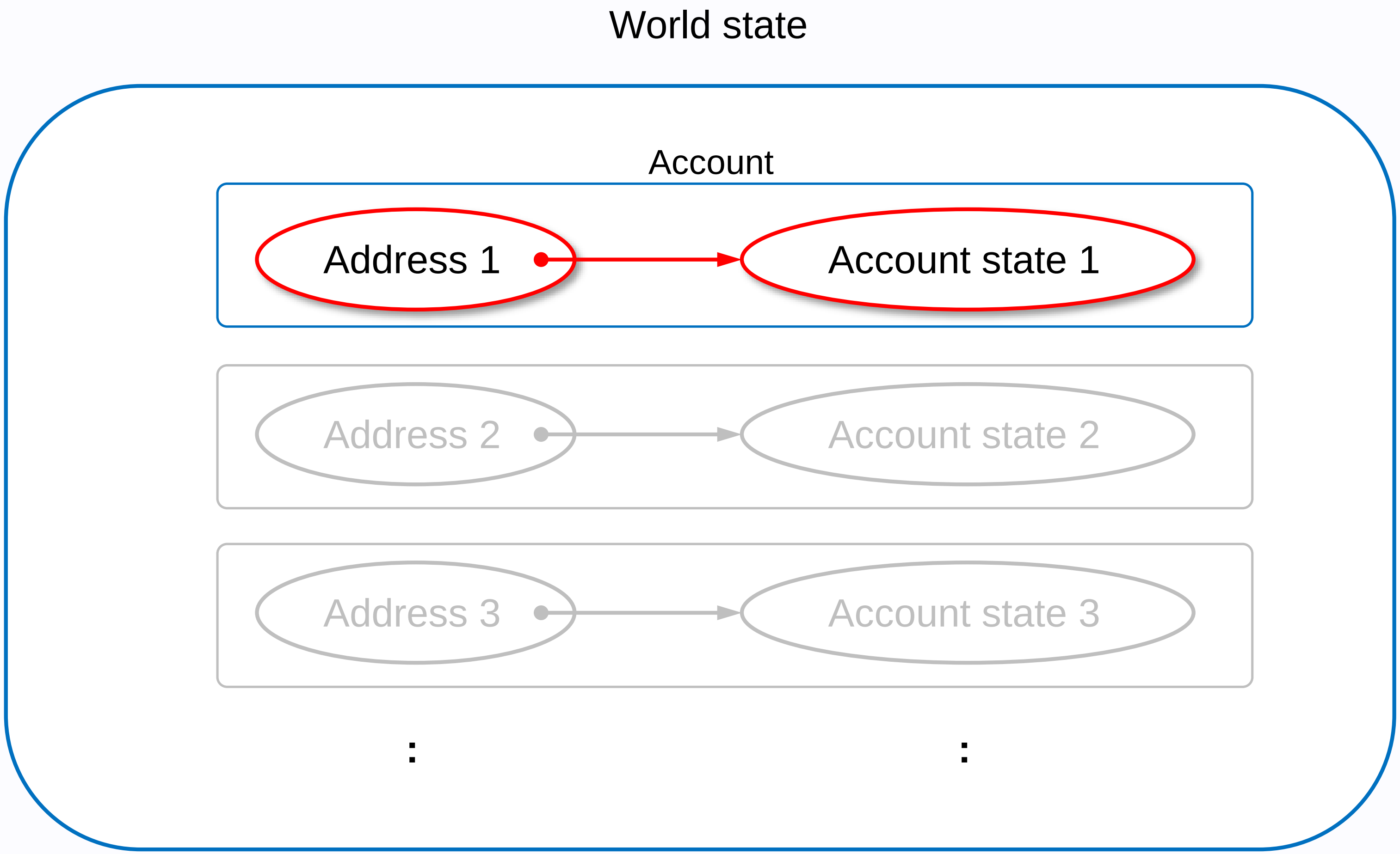
# Account



An account is an object in the world state.

References : [E1] Ch.4, [E2]

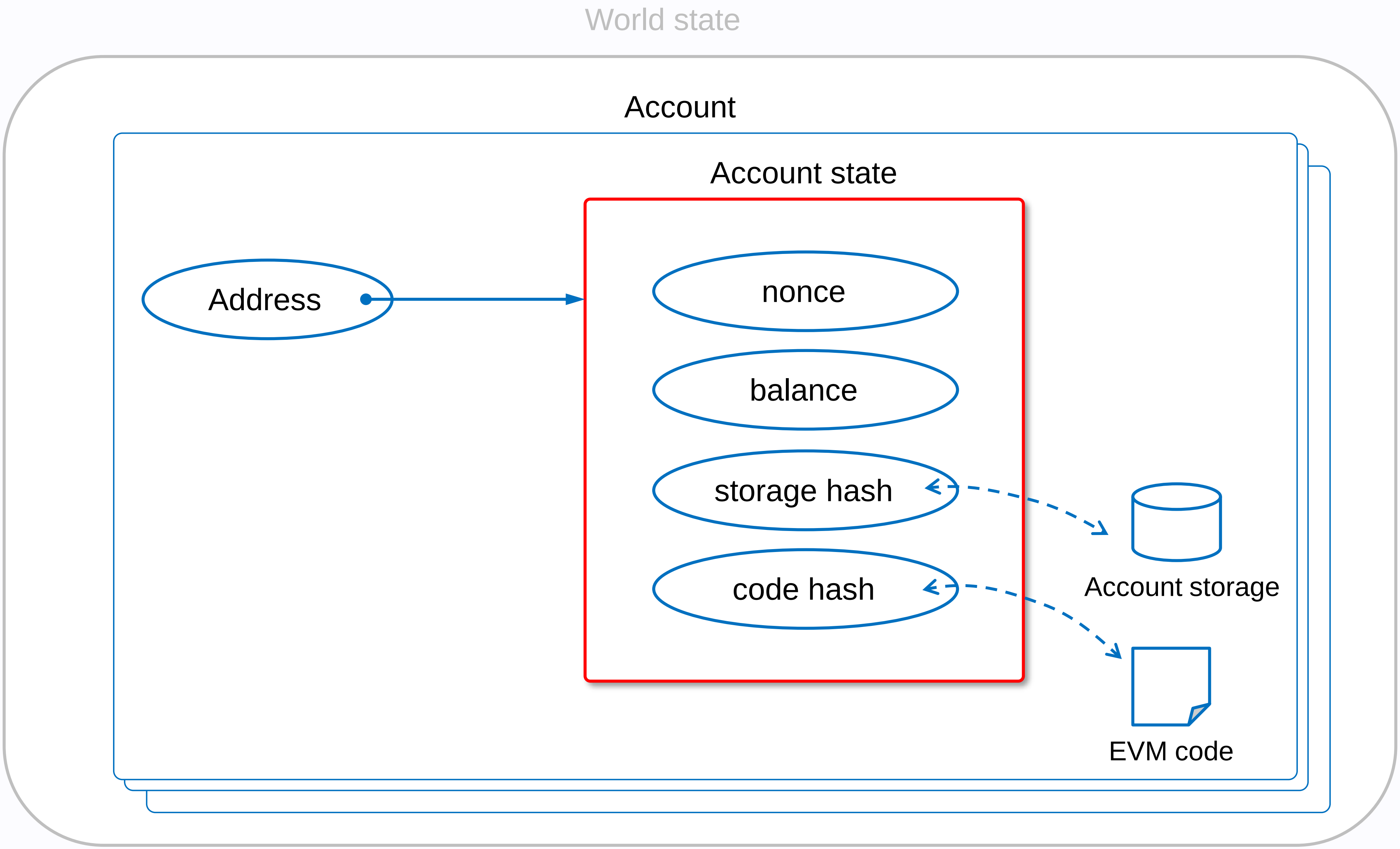
# Account



An account is a mapping between address and account state.

References : [E1] Ch.4, [E2]

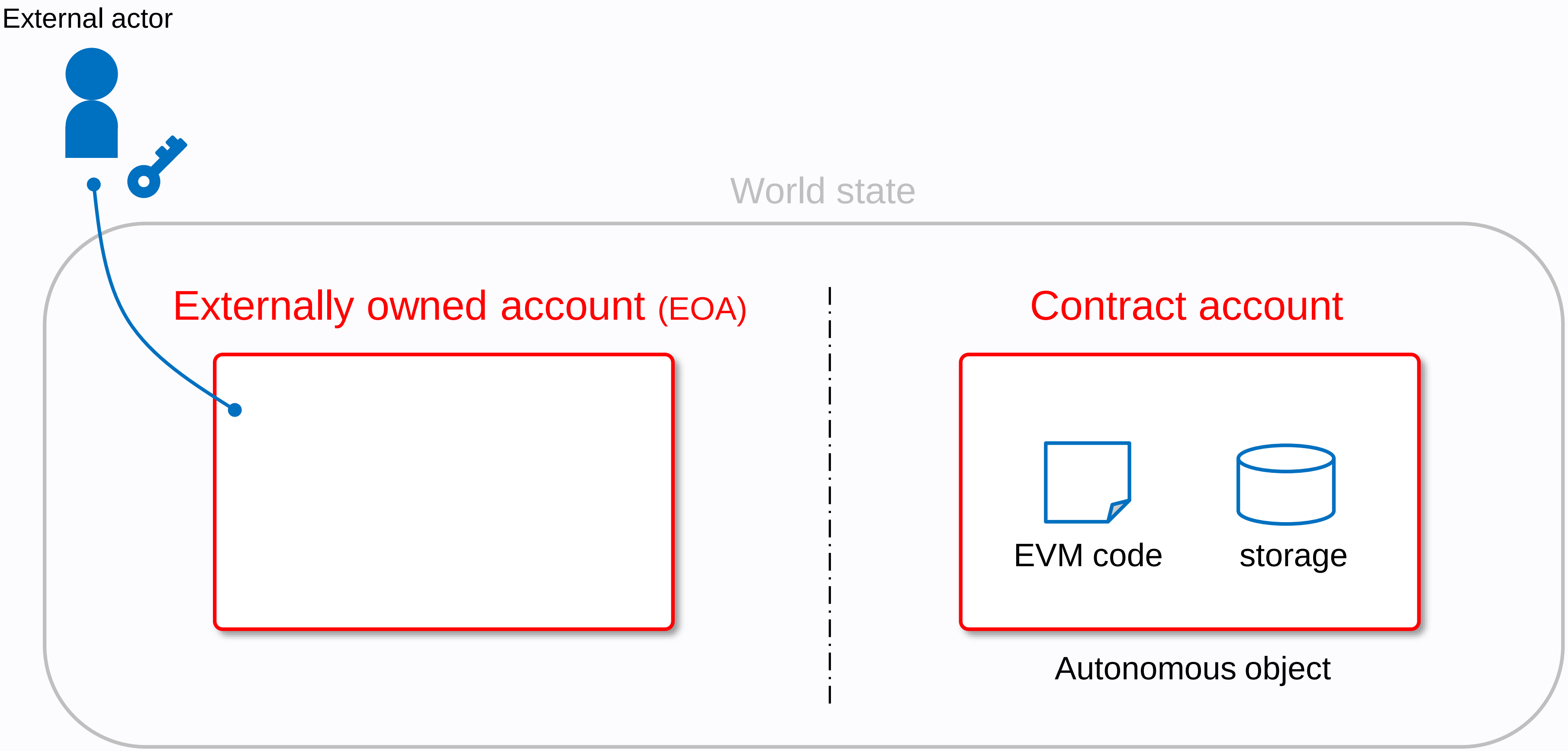
# Account state



An account state could contain EVM code and storage.

References : [E1] Ch.4

# Two practical types of account

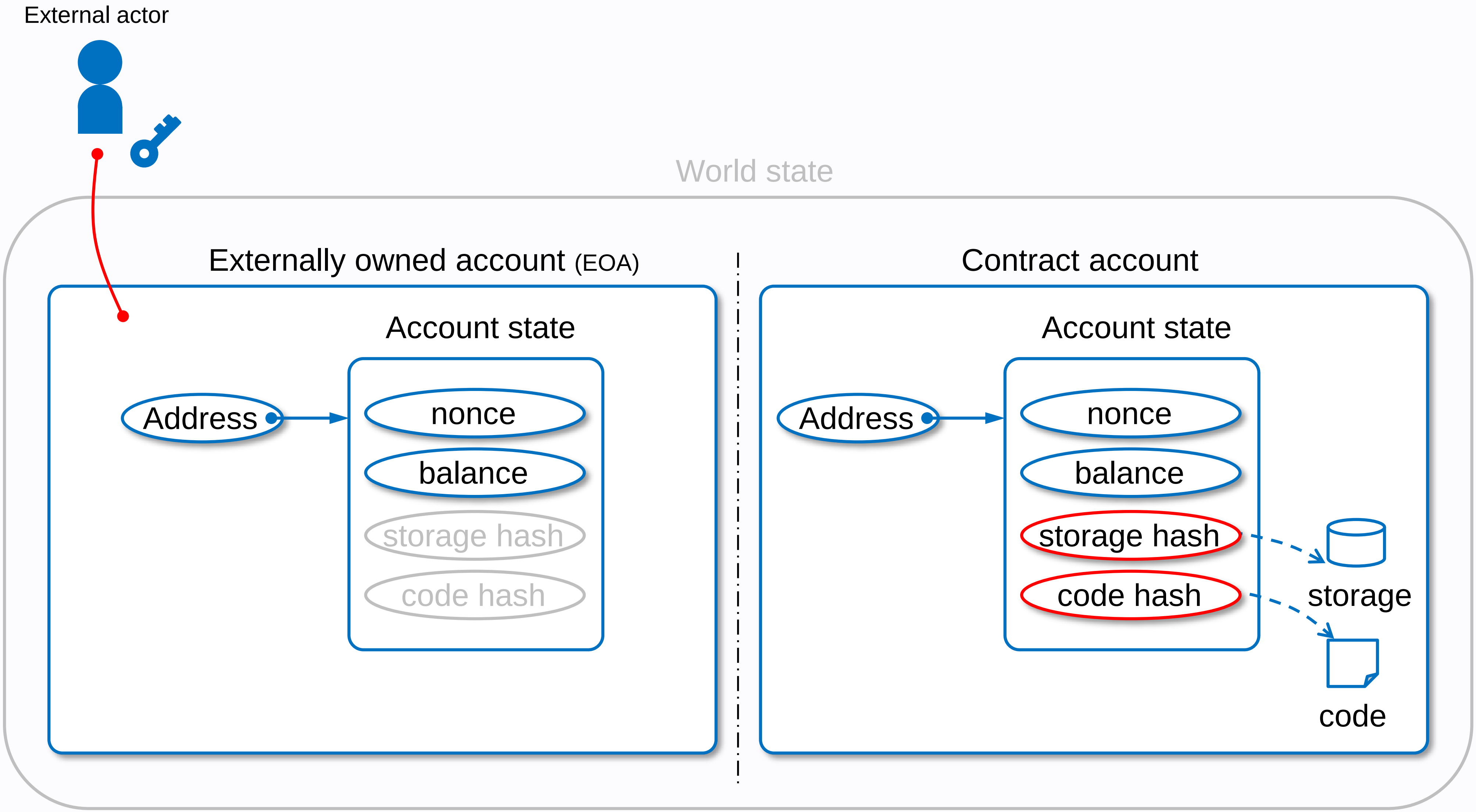


EOA is controlled by a private key.

Contract account contains EVM code.

References : [E1] Ch.4

# Two practical types of account

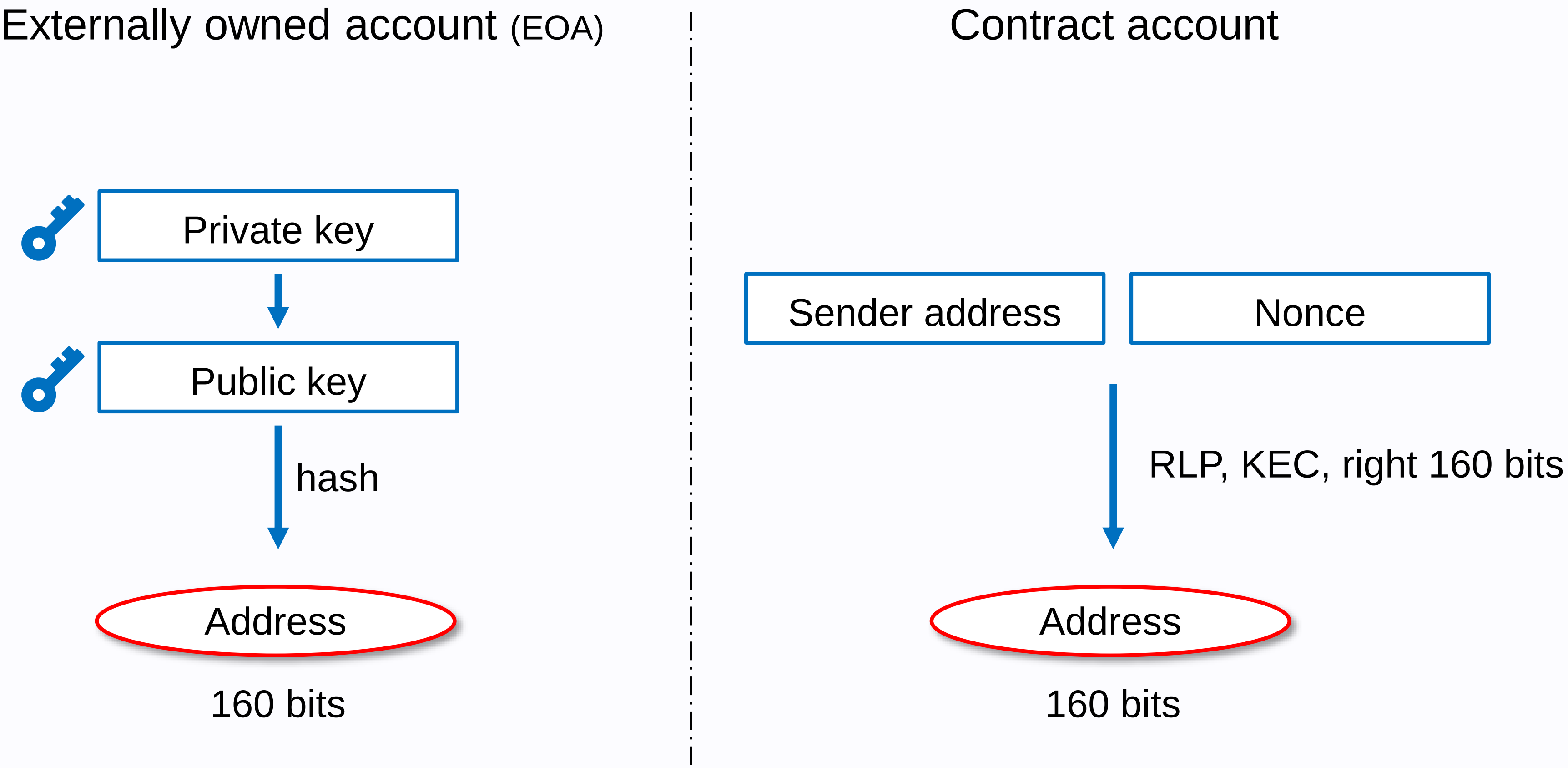


EOA is controlled by a private key.
EOA cannot contain EVM code.

Contract contains EVM code.
Contract is controlled by EVM code.

References : [E1] Ch.4

# Address of account



Externally owned account (EOA)

Private key

Public key

hash

Address

160 bits

Contract account

Sender address

Nonce

RLP, KEC, right 160 bits

Address

160 bits

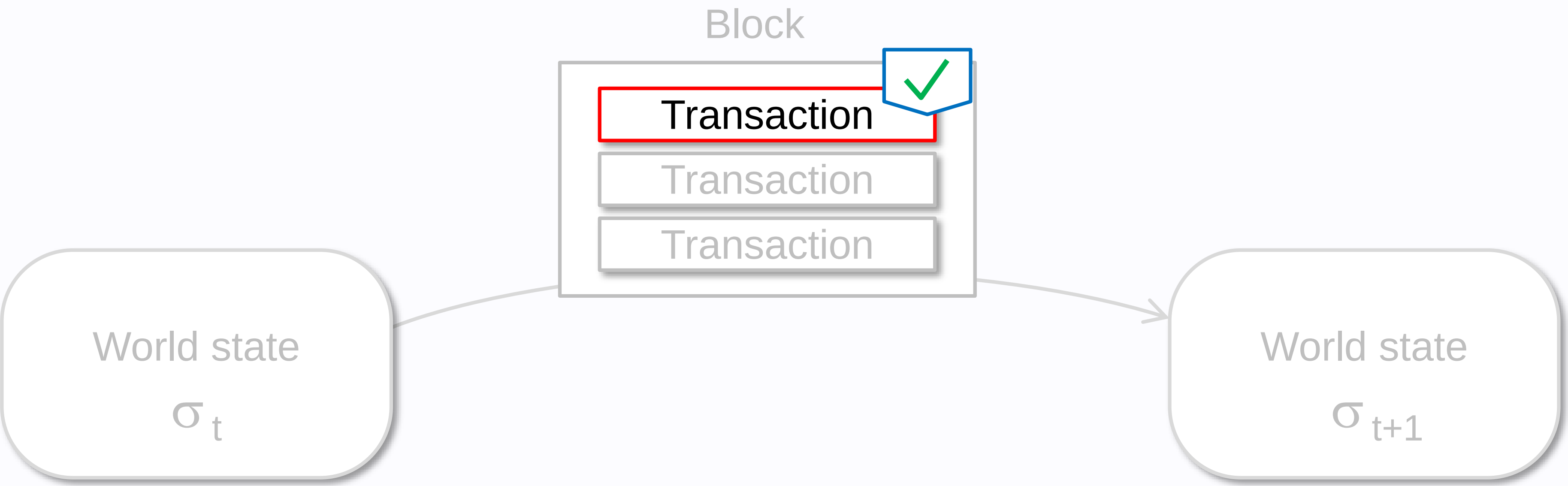
A 160-bit code used for identifying accounts.

References : [E1] Ch.4, Ch.7, [E2]

# 1. Introduction

Transaction

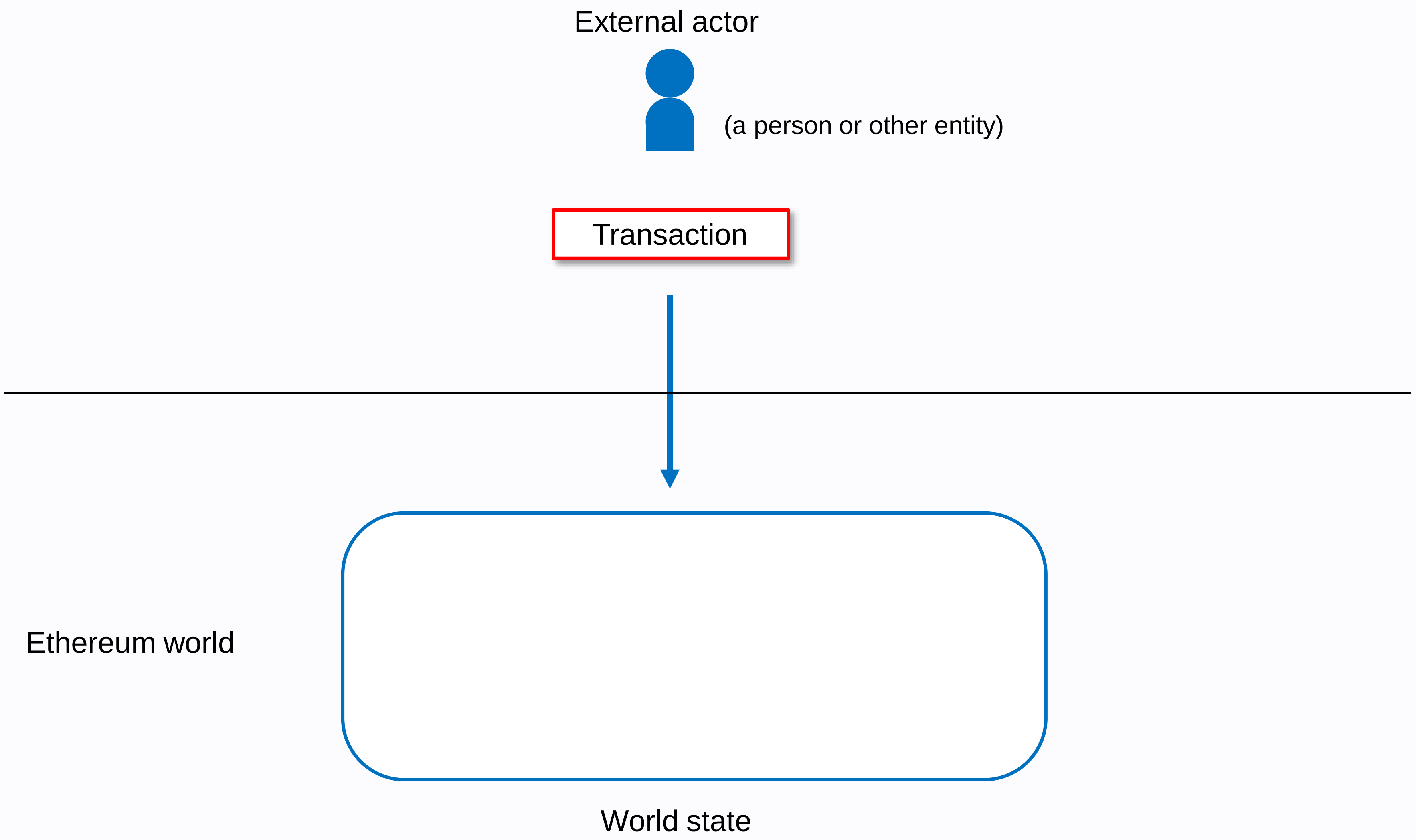
# A transaction



A transaction is a single cryptographically-signed instruction.

References : [E1] Ch.2, Ch.4, [E2]

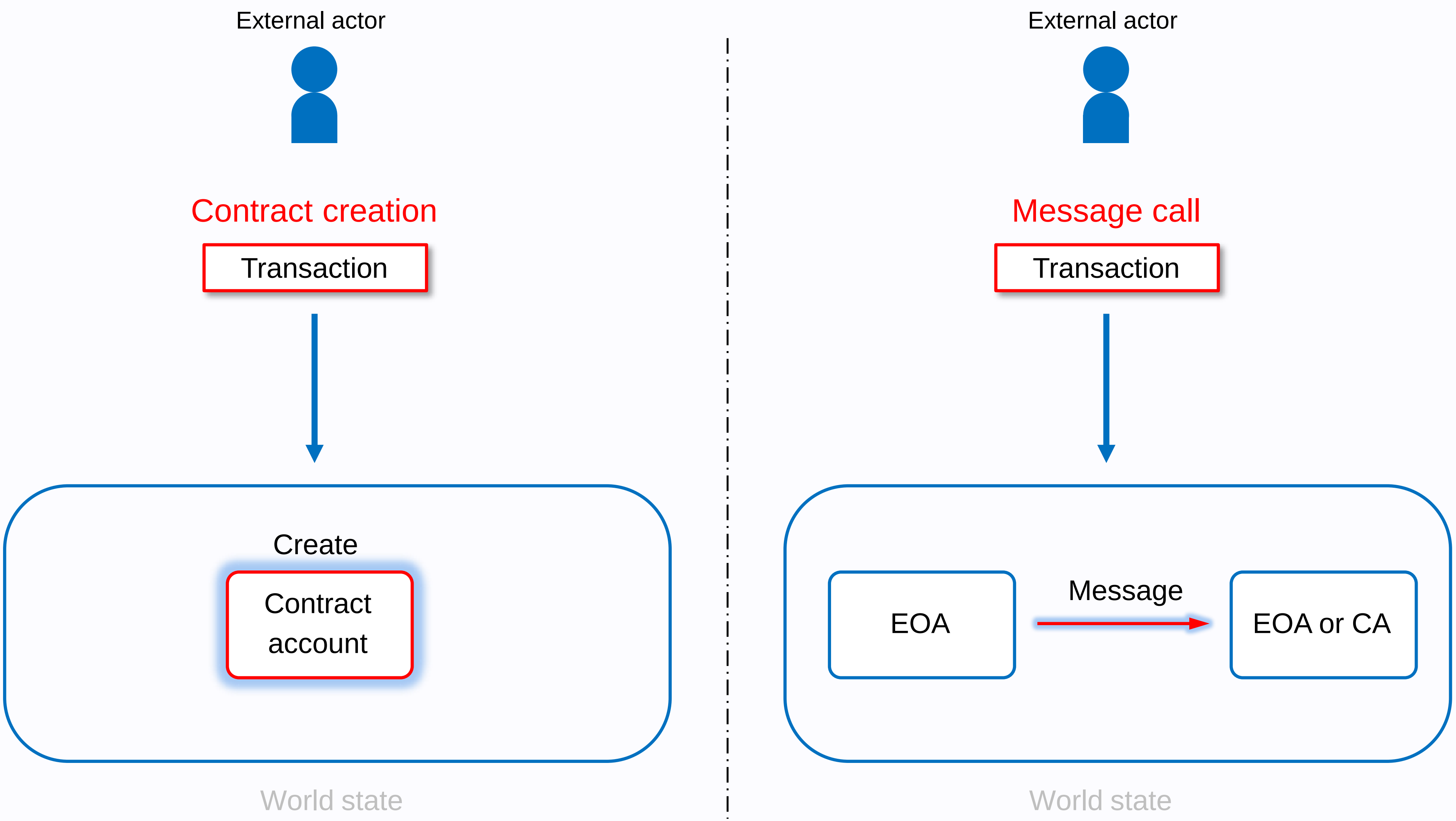
# A transaction to world state



A transaction is submitted by external actor.

References : [E1] Appendix A

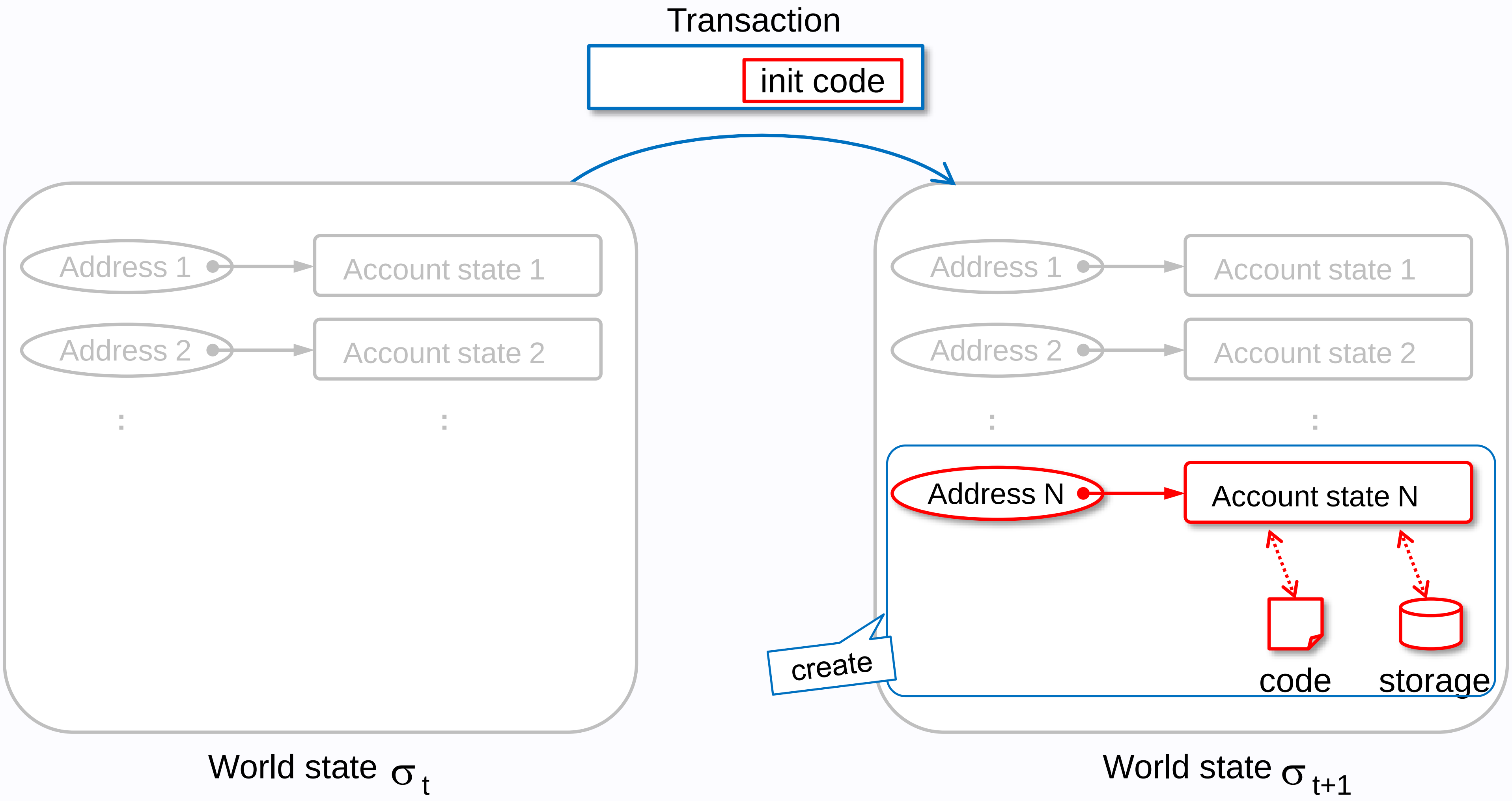
# Two practical types of transaction



There are two practical types of transaction, contract creation and message call.

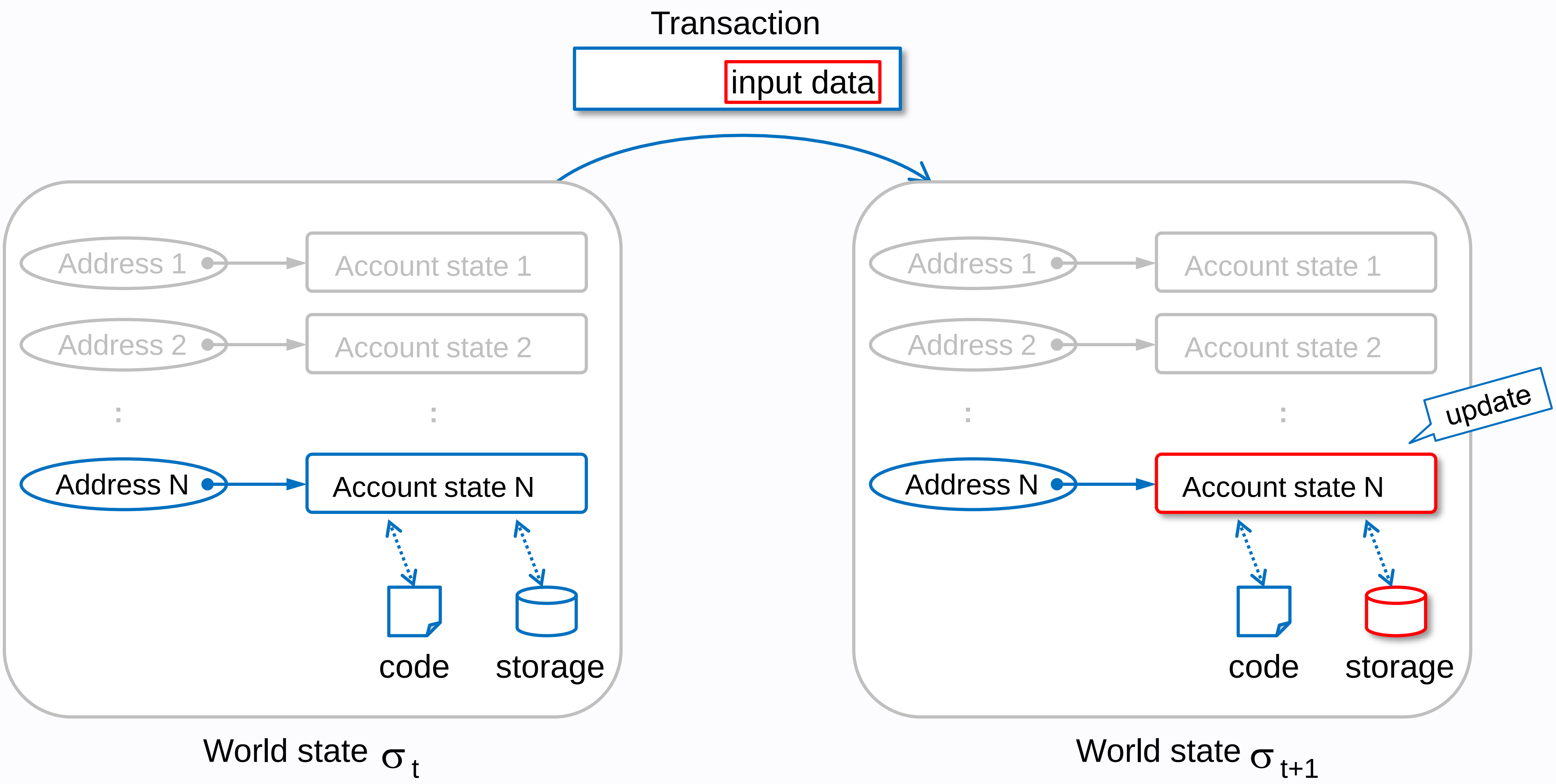
References : [E1] Ch.4

# Contract creation



References : [E1] Ch.7

# Message call



Transaction

input data

Address 1

Account state 1

Address 2

Account state 2

:

:

Address N

Account state N

code

storage

World state $\sigma_t$

Address 1

Account state 1

Address 2

Account state 2

:

:

update

Address N

Account state N

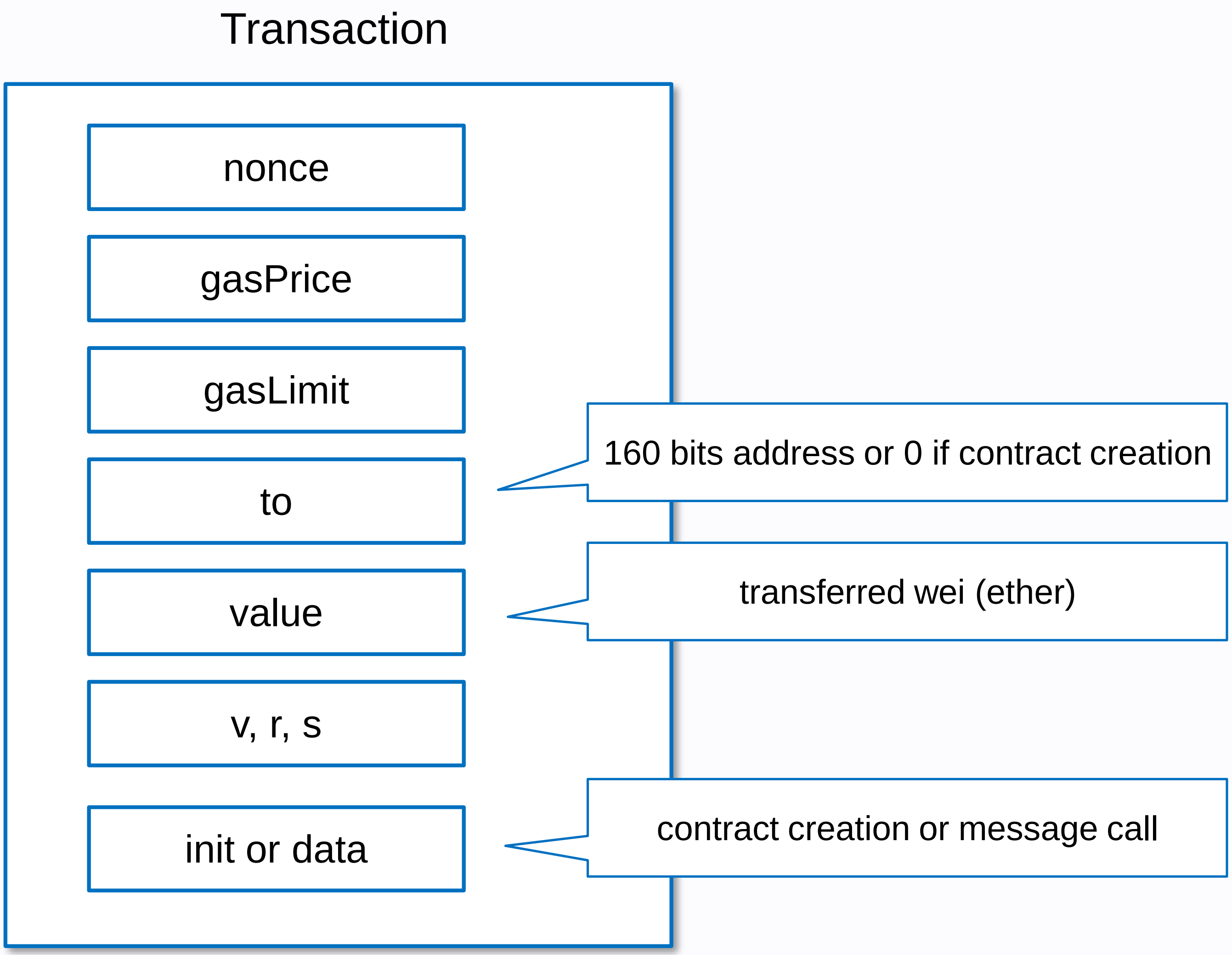
code

storage

World state $\sigma_{t+1}$

References : [E1] Ch.7

# Field of a transaction

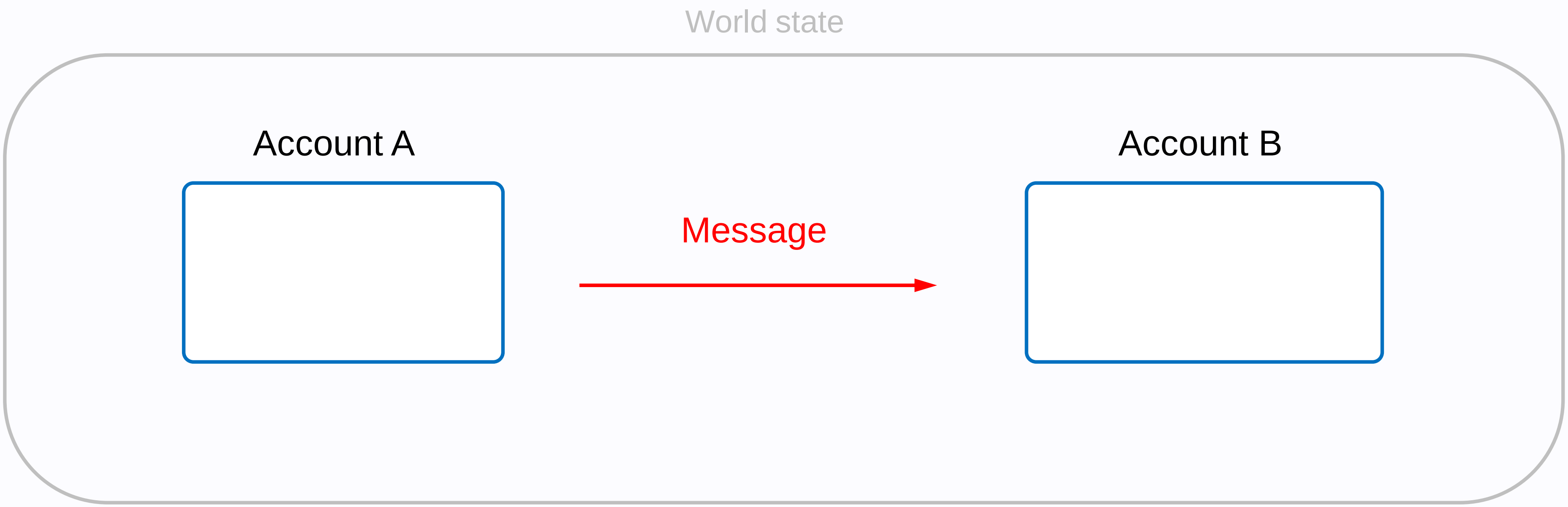


References : [E1] Ch.4

# 1. Introduction

Message

# Message



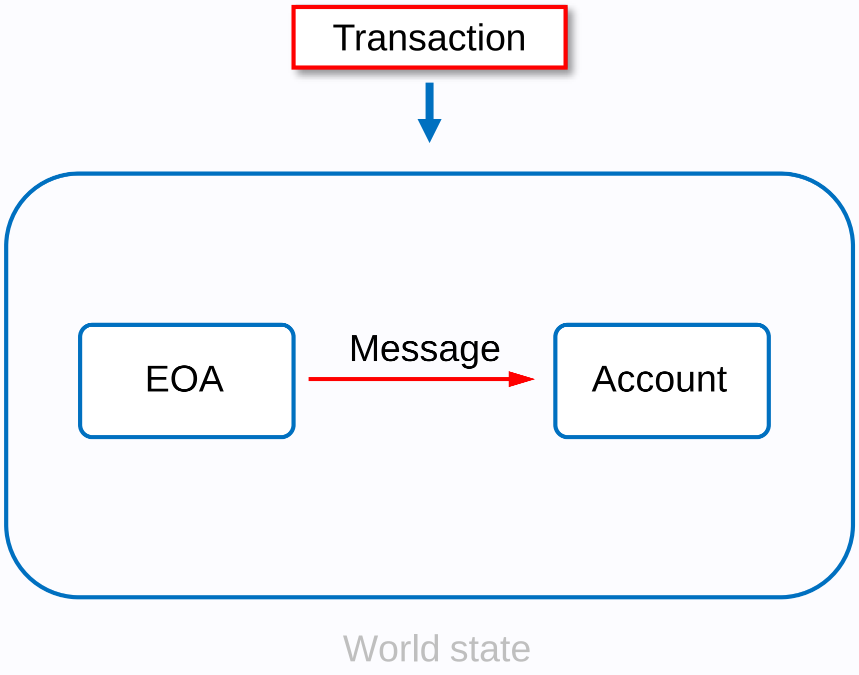
Message is passed between two Accounts.

Message is Data (as a set of bytes) and Value (specified as Ether) .

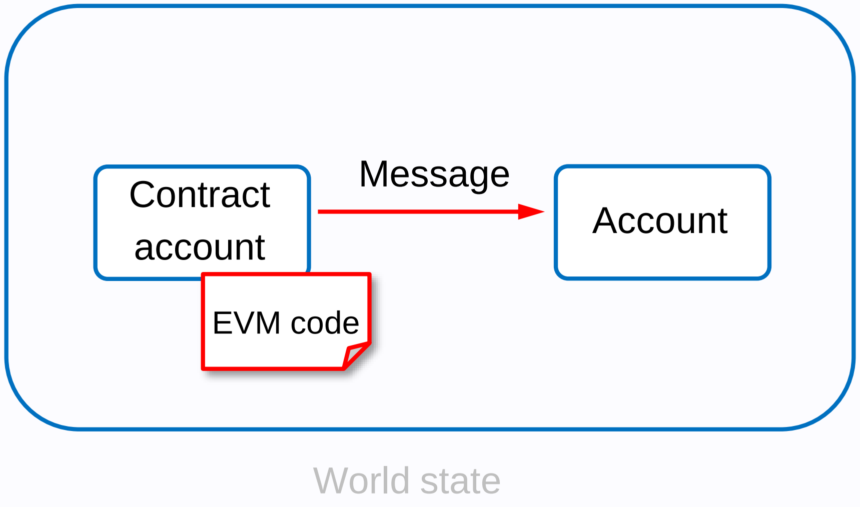
References : [E1] Ch.8, Appendix A

# Message

Triggered by transaction



Triggered by EVM code

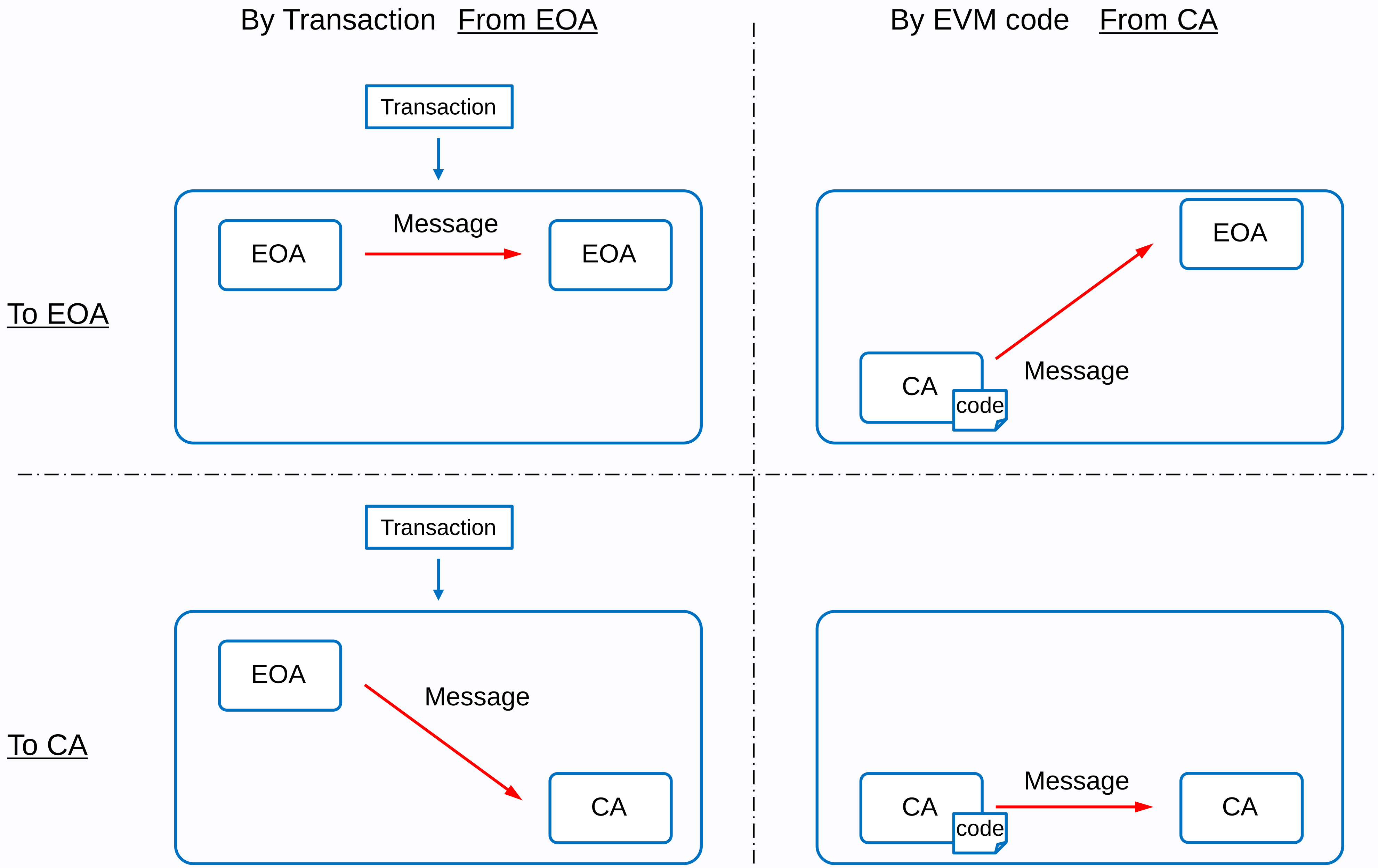


Transaction triggers an associated message.

EVM can also send a message.

References : [E1] Ch.8

# Four cases of message



By Transaction From EOA

By EVM code From CA

To EOA

To CA

Transaction

EOA

Message

EOA

EOA

CA

code

Message

Transaction

EOA

Message

CA

Message

CA

code

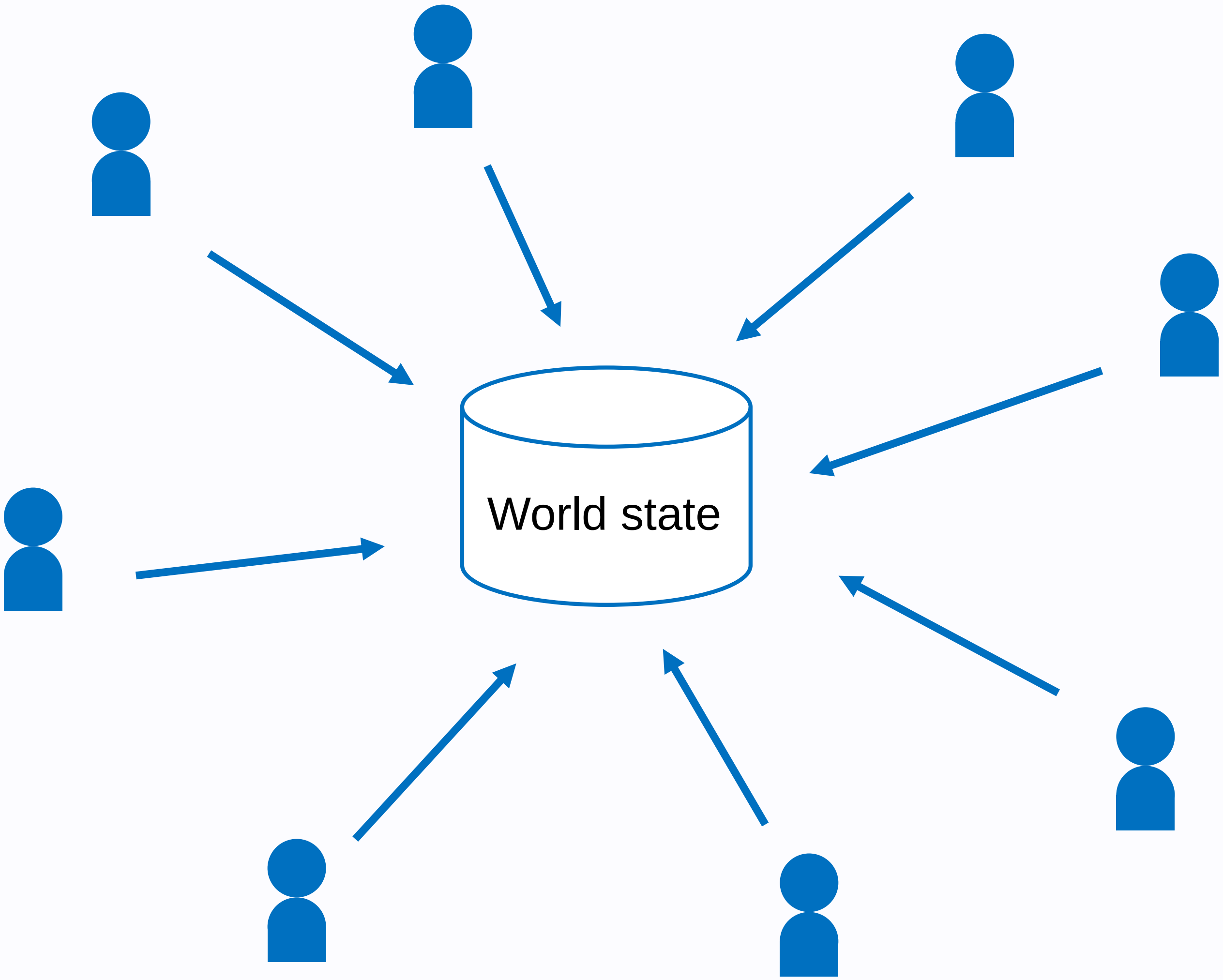
CA

References : [E1] Ch.8

## 1. Introduction

# Decentralised database

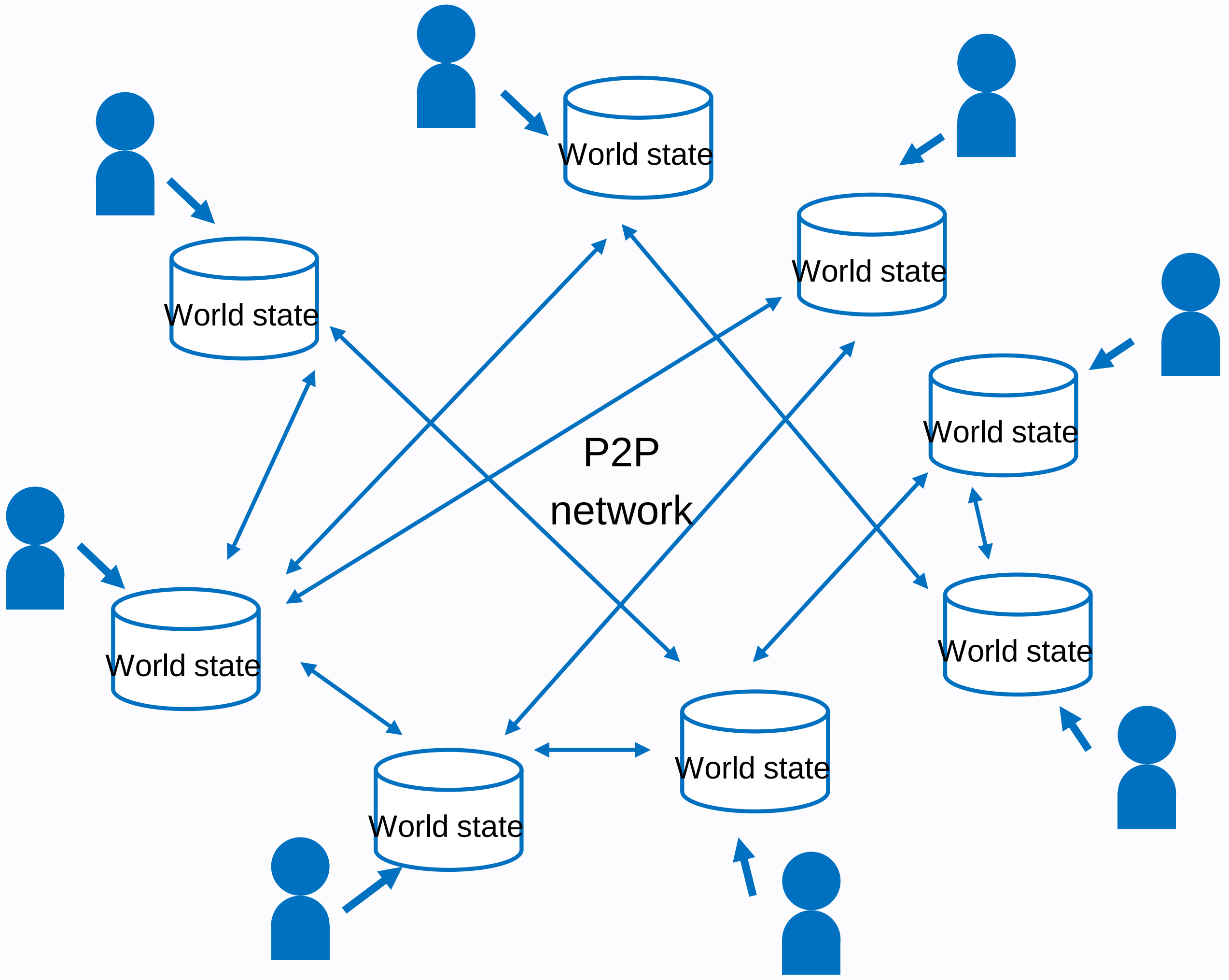
# Globally shared, transactional database



A blockchain is a globally shared, transactional database.

References : [E3], [E7] Ch.7

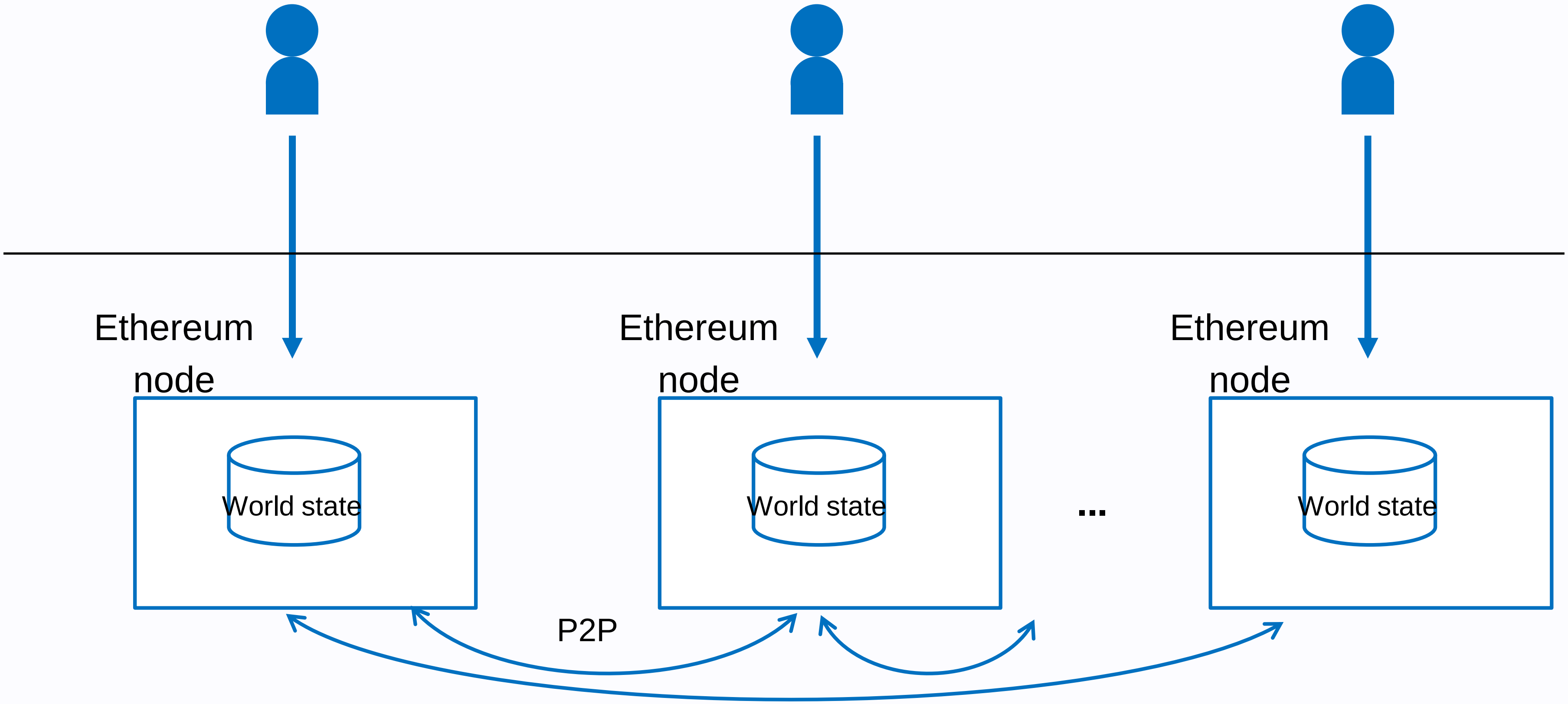
# Decentralised database



A blockchain is a globally shared, decentralised, transactional database.

References : [E3], [E7] Ch.7

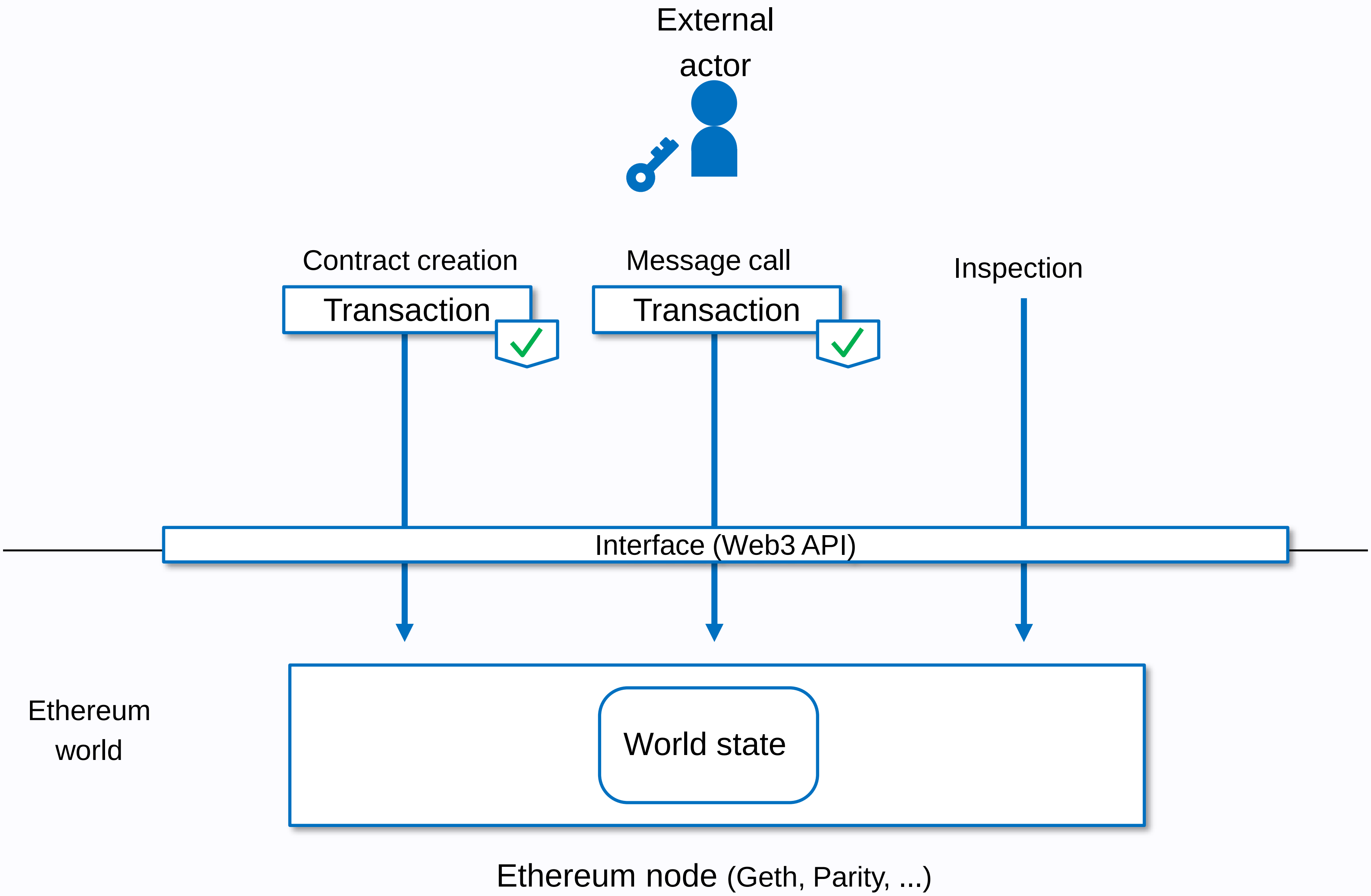
# P2P network inter nodes



Decentralised nodes constitute Ethereum P2P network.

References : [E3],

# Interface to a node

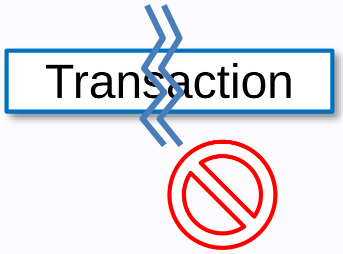


External actors access the Ethereum world through Ethereum nodes.

References : [E1] Appendix A, Ch.4, Ch.7, Ch.8

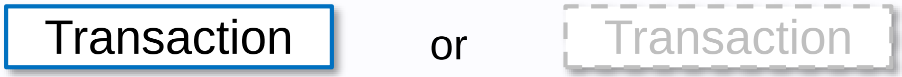
# Atomicity and order

# Atomicity of transaction



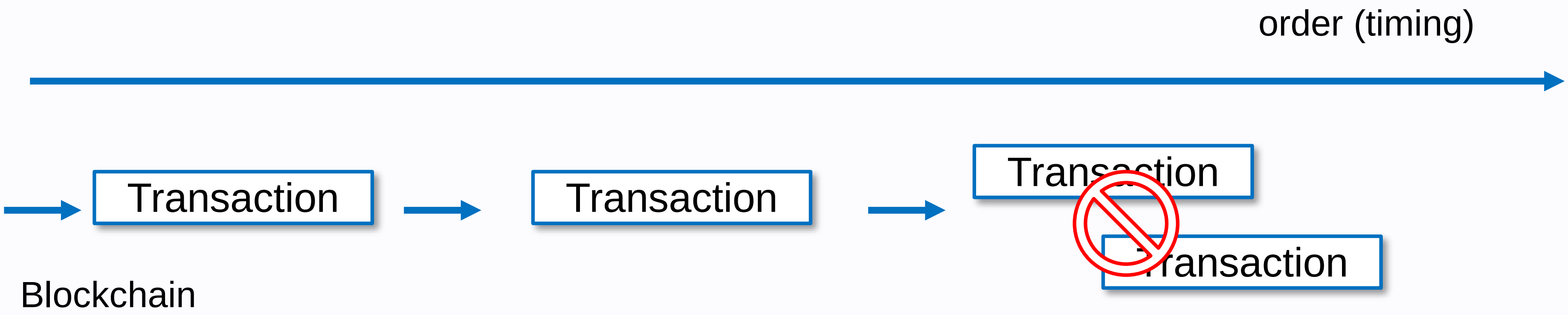
A transaction is an atomic operation. Can't divide or interrupt.

---



That is, All (complete done) or Nothing (zero effect).

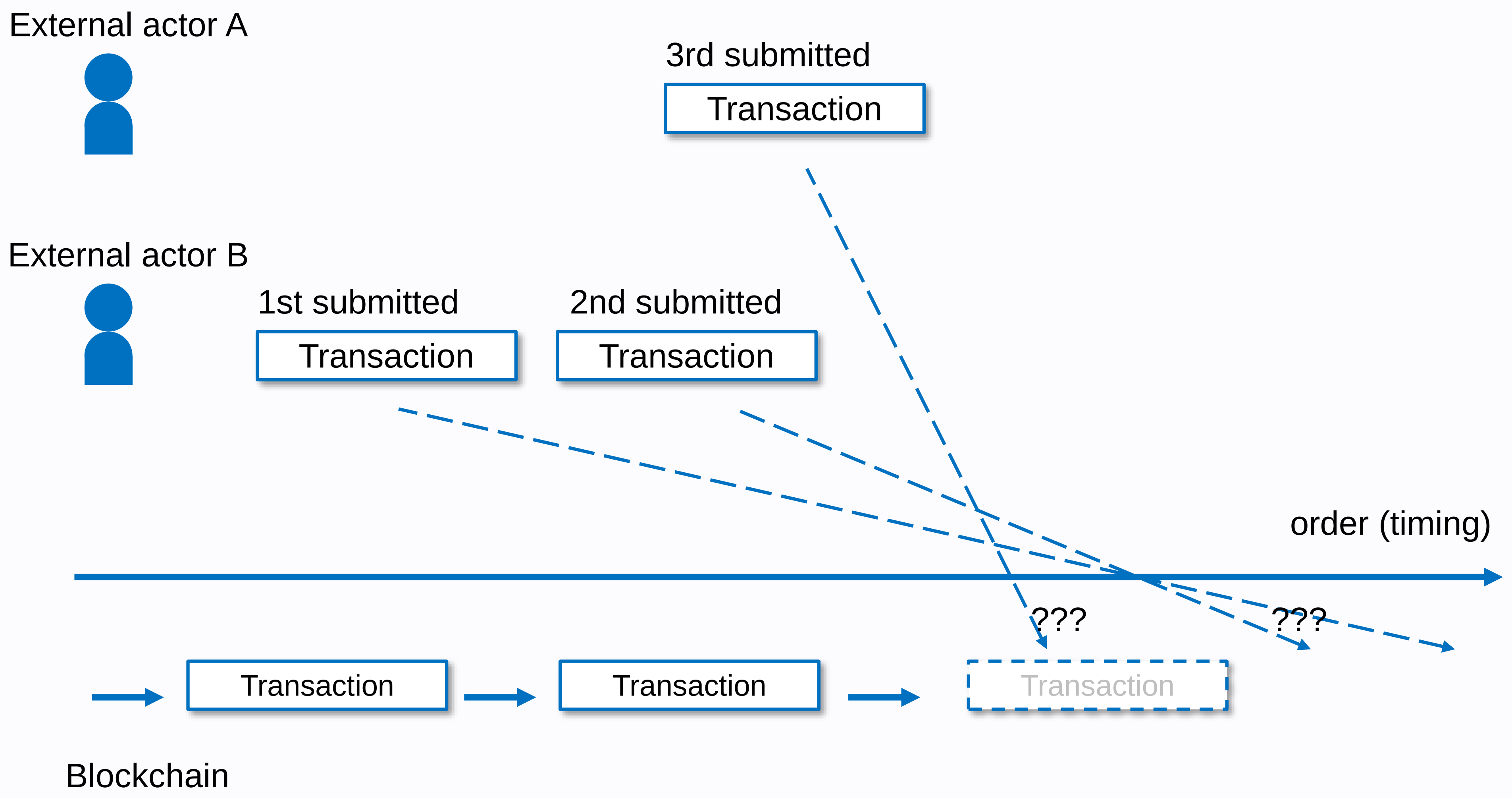
# Order of transactions



Transactions cannot be overlapped.

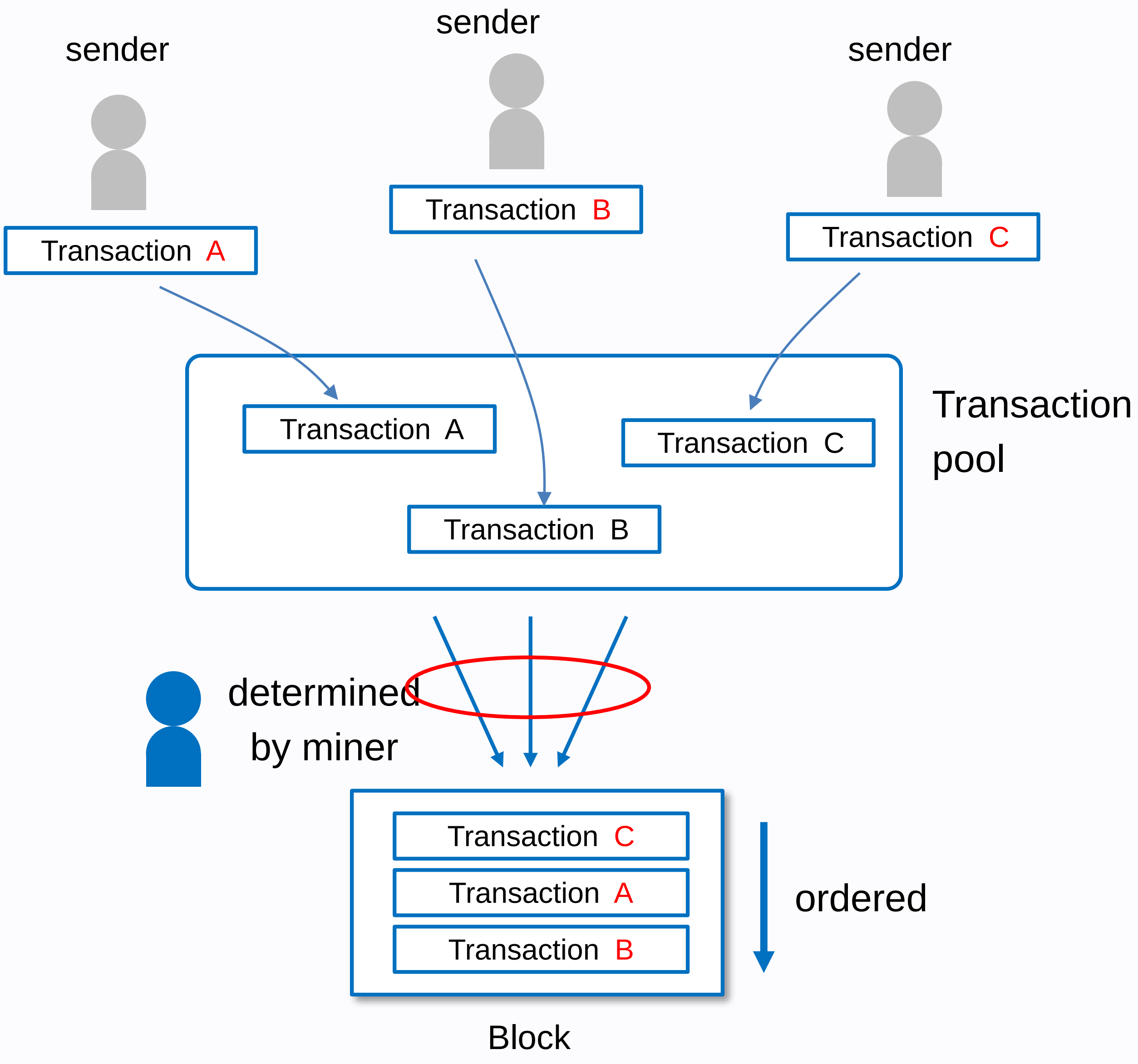
Transactions must be executed sequentially.

# Order of transactions



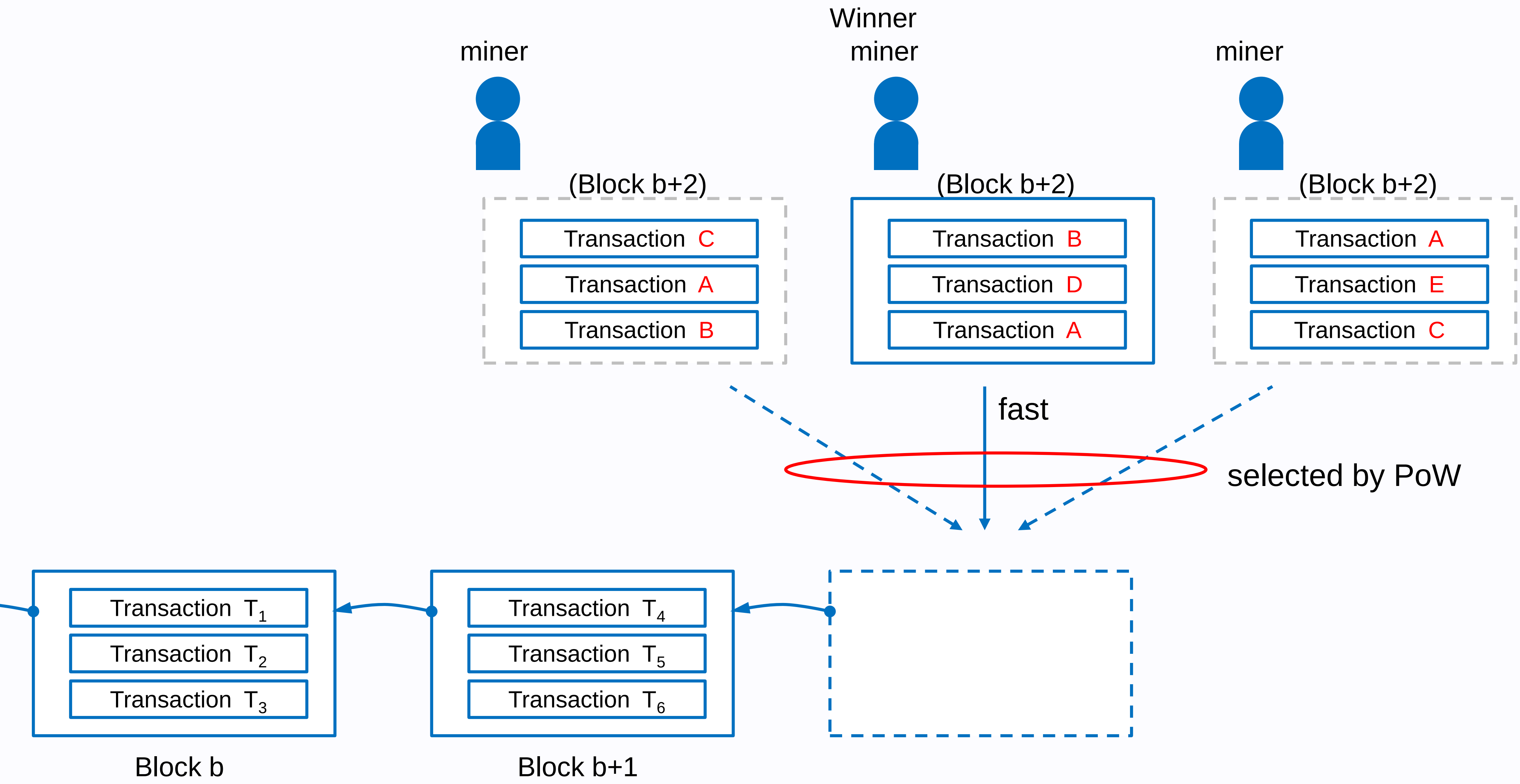
Transaction order is not guaranteed.

# Ordering inner block



Miner can determine the order of transactions in a block.

# Ordering inter blocks



The order between blocks is determined by a consensus algorithm such as PoW.

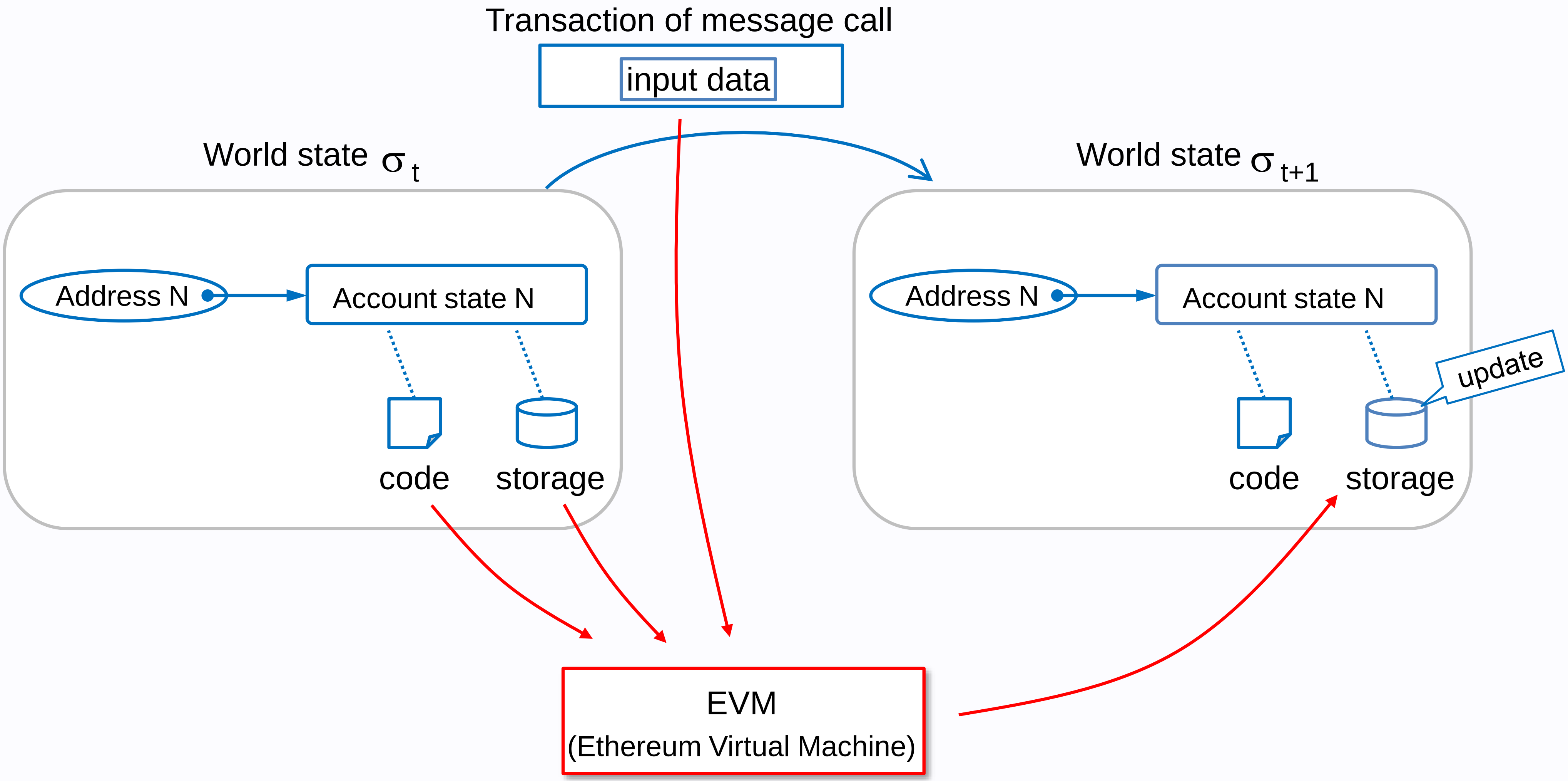
References : [E1] Ch.2, Ch.4

# 2. Virtual machine

# 2. Virtual machine

Ethereum virtual machine (EVM)

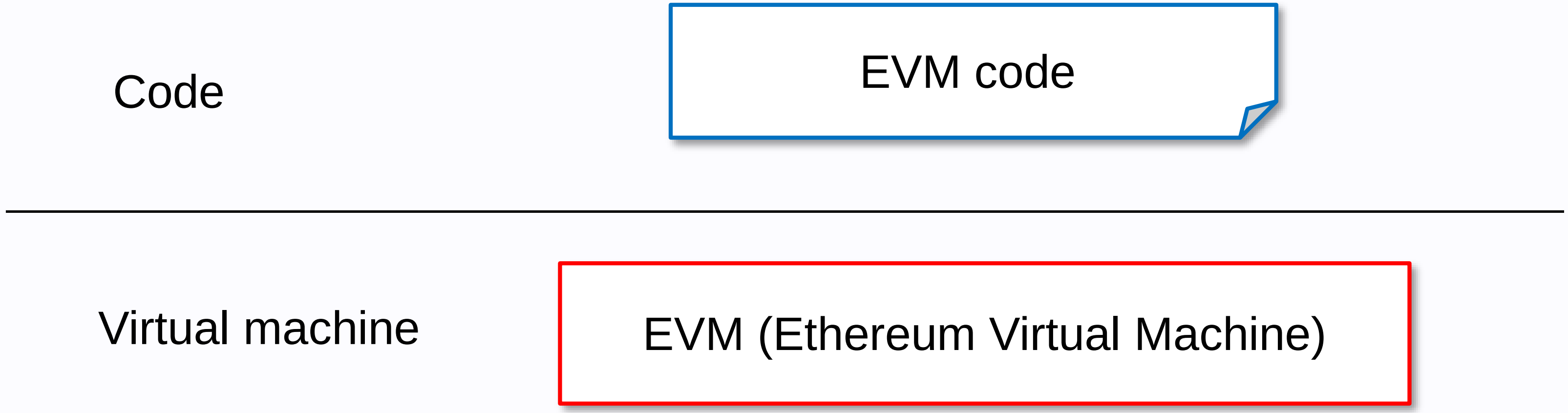
# Ethereum virtual machine



EVM code is executed on Ethereum Virtual Machine (EVM).

References : [E1] Ch.9, Appendix H

# Ethereum virtual machine


Code

EVM code

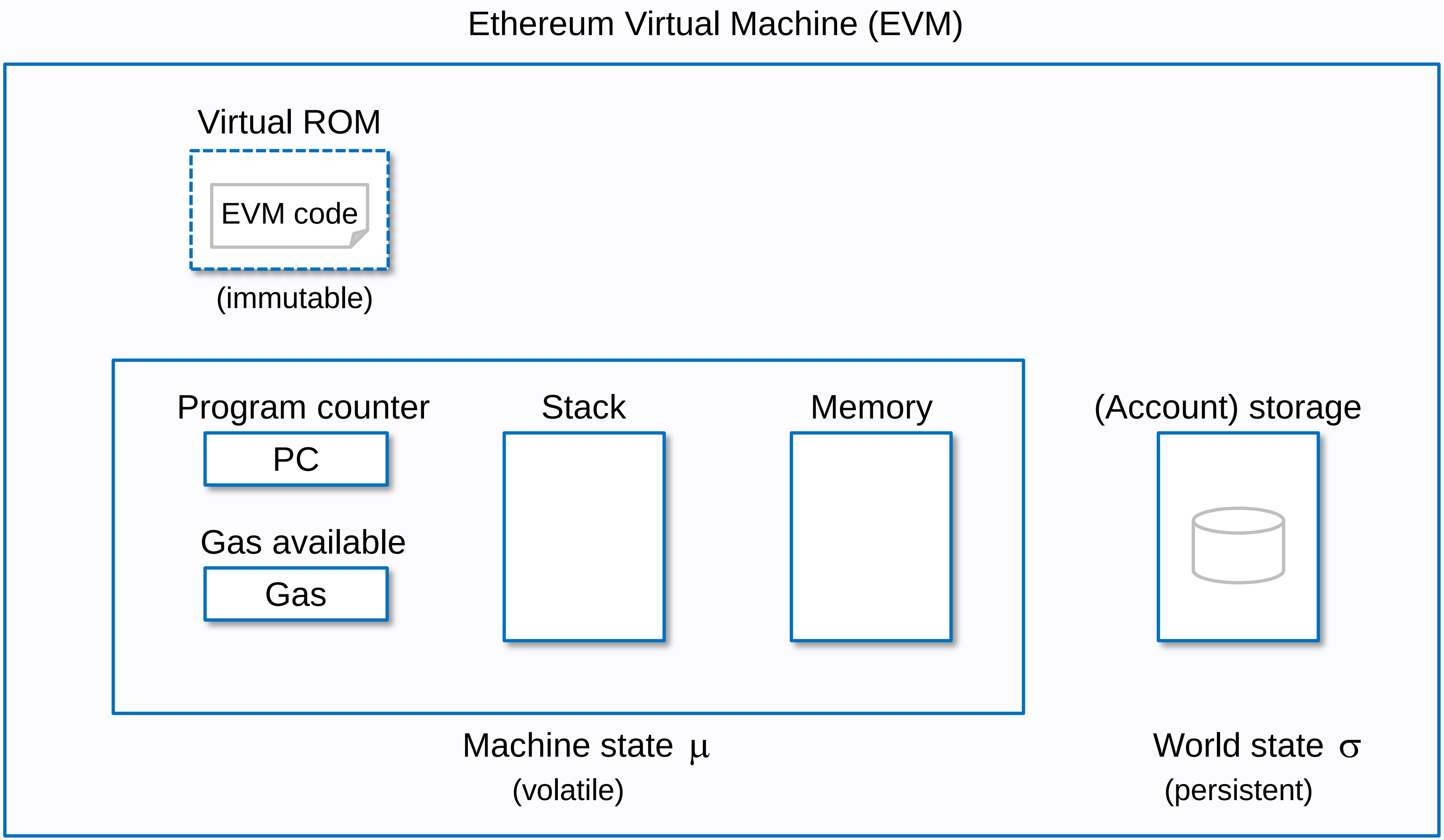
Virtual machine

EVM (Ethereum Virtual Machine)

The Ethereum Virtual Machine is the runtime environment for smart contracts in Ethreum.

References : [E1] Ch.9, Appendix H, [E4], [W1]

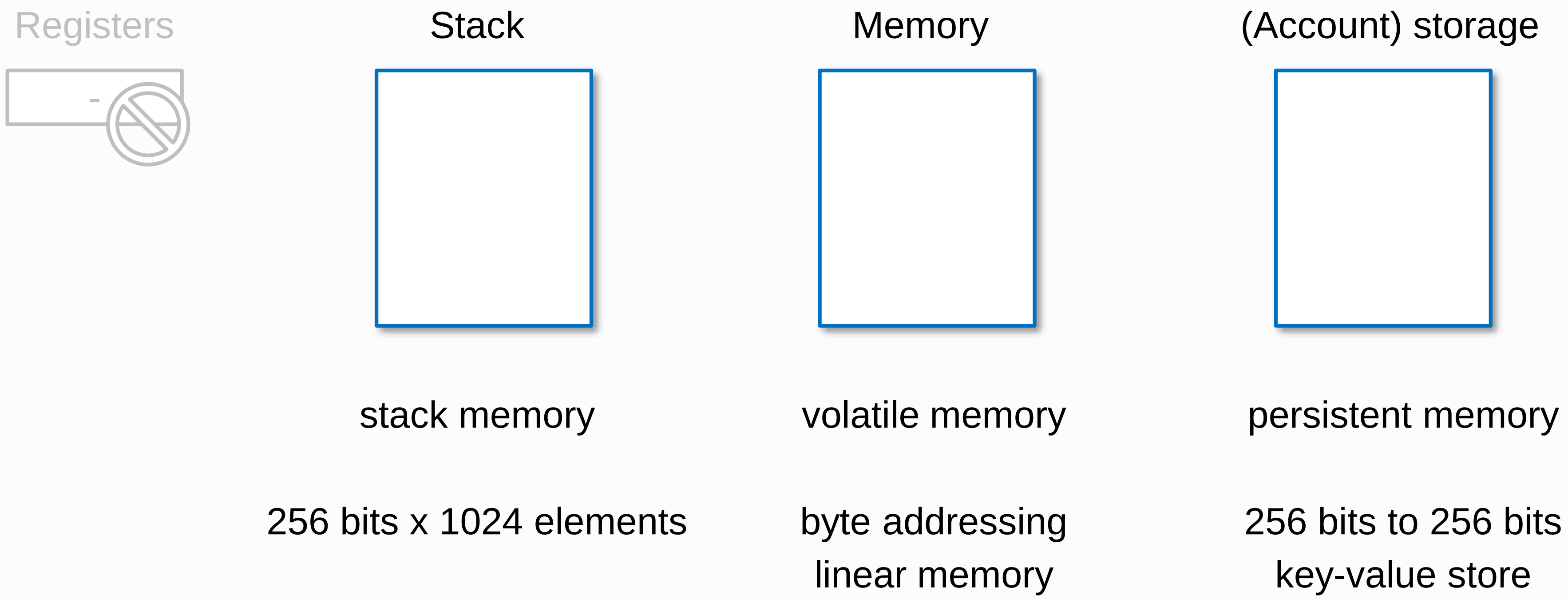
# EVM architecture



The EVM is a simple stack-based architecture.

References : [E1] Ch.9, Appendix H, [E4]

# Machine space of EVM



Registers

-

Stack

stack memory

256 bits x 1024 elements

Memory

volatile memory

byte addressing
linear memory

(Account) storage

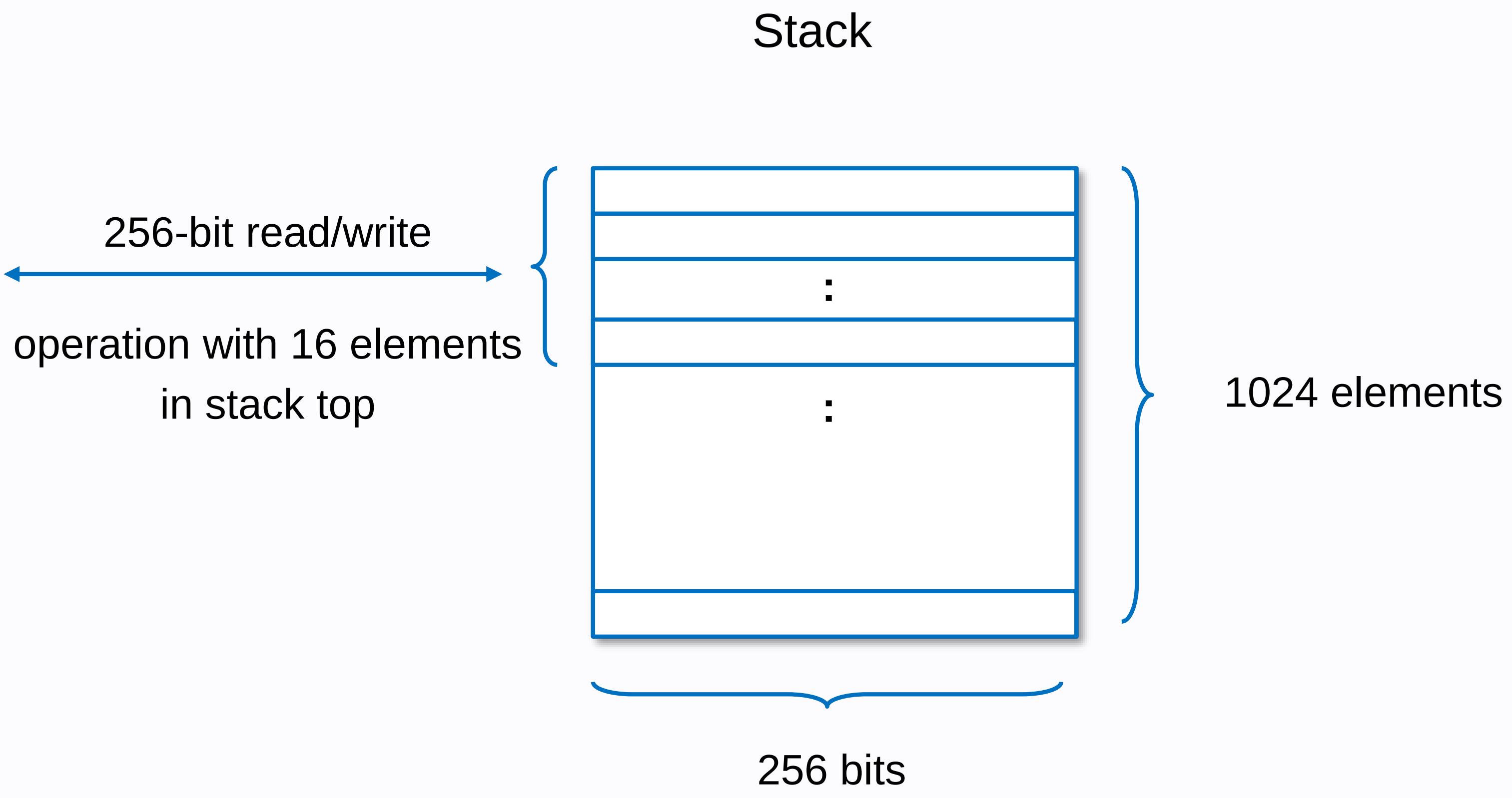
persistent memory

256 bits to 256 bits
key-value store

There are several resources as space.

References : [E1] Ch.9, Appendix H, [E3], [E7], [W2]

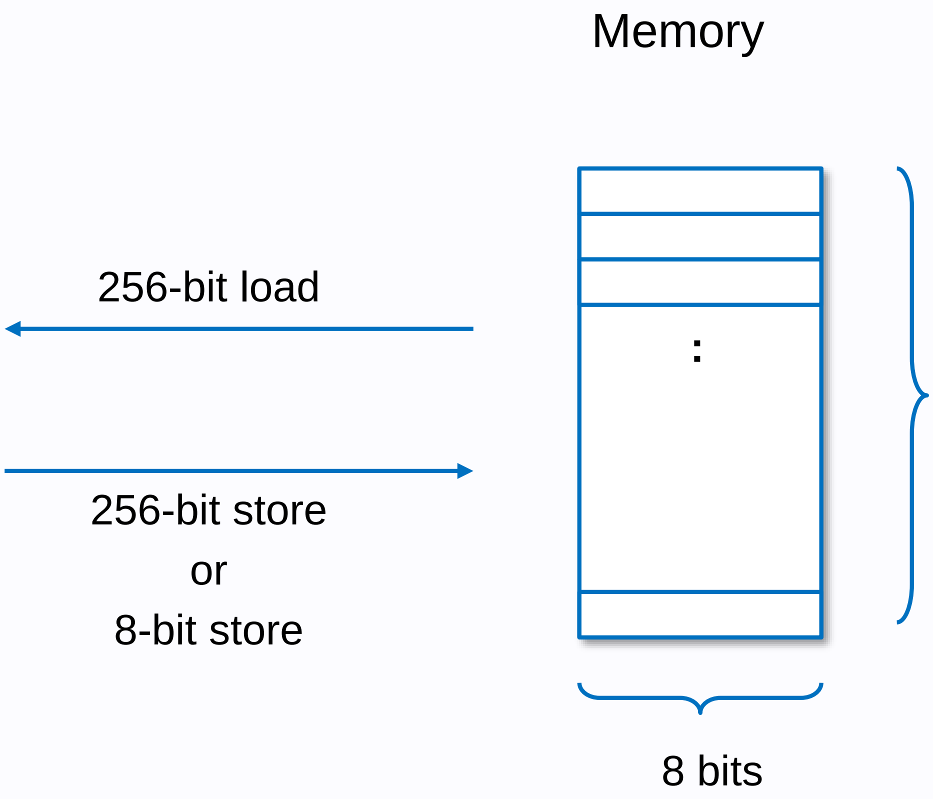
# Stack



All operation are performed on the stack.
Access with many instructions such as PUSH/POP/COPY/SWAP, ...

References : [E1] Ch.9, Appendix H, [E7], [W2]

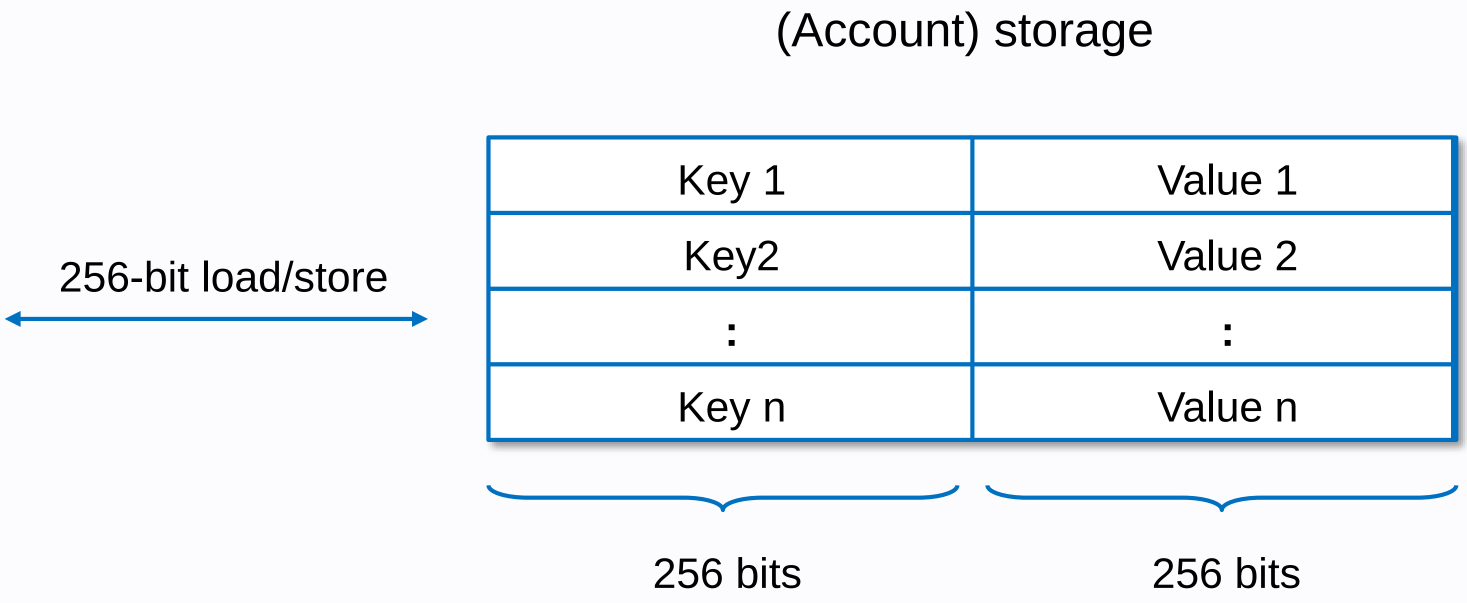
# Memory



Memory is linear and can be addressed at byte level.
Access with MSTORE/MSTORE8/MLOAD instructions.
All locations in memory are well-defined initially as zero.

References : [E1] Ch.9, Appendix H, [E7], [W2]

# Account storage



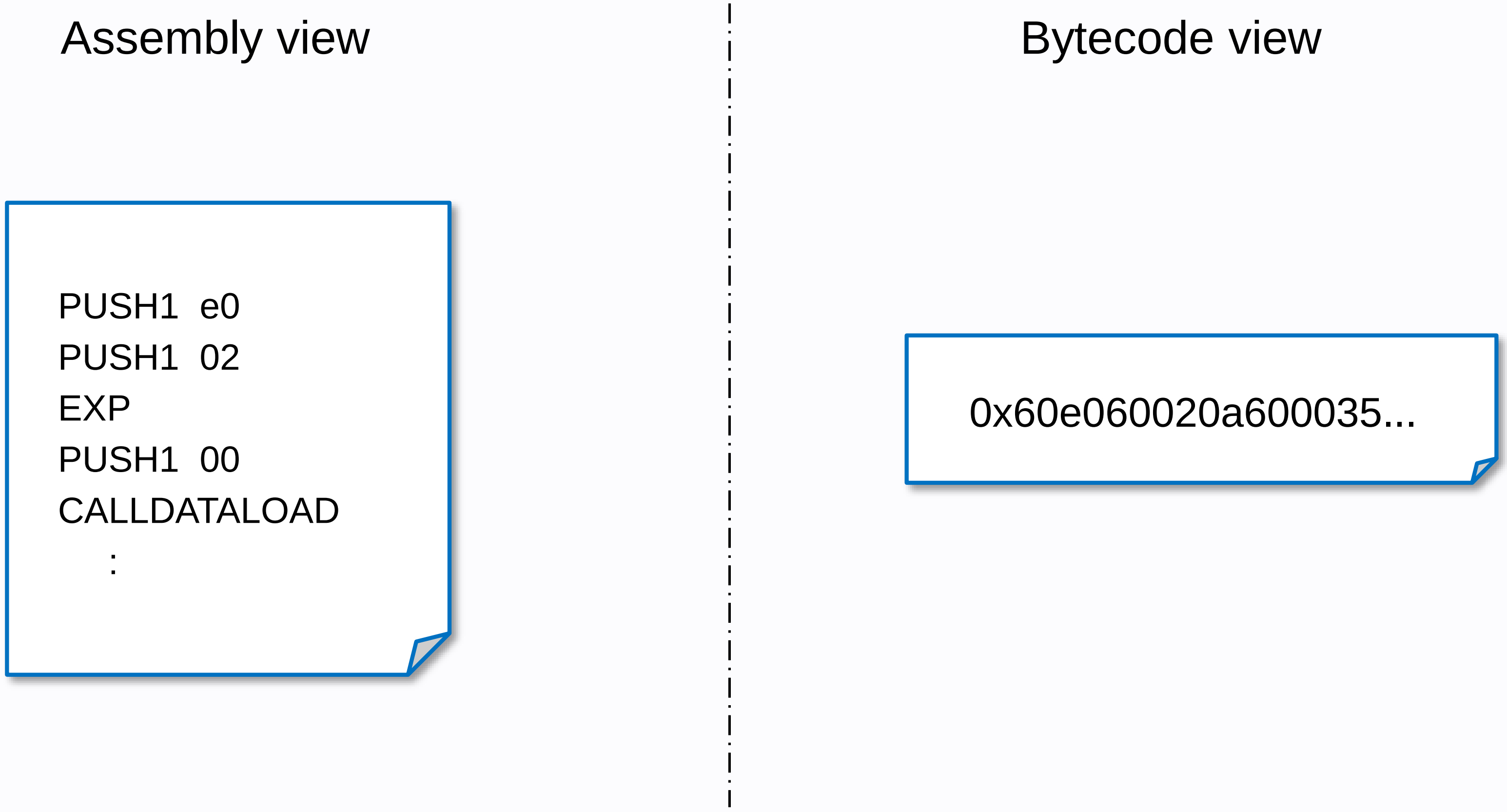
(Account) storage

| | |
|---|---|
| Key 1 | Value 1 |
| Key2 | Value 2 |
| : | : |
| Key n | Value n |

Storage is a key-value store that maps 256-bit words to 256-bit words.
Access with SSTORE/SLOAD instructions.
All locations in storage are well-defined initially as zero.

References : [E1] Ch.9, Appendix H, [E7], [W2]

# EVM code



Assembly view

```
PUSH1  e0
PUSH1  02
EXP
PUSH1  00
CALLDATALOAD
    :
```

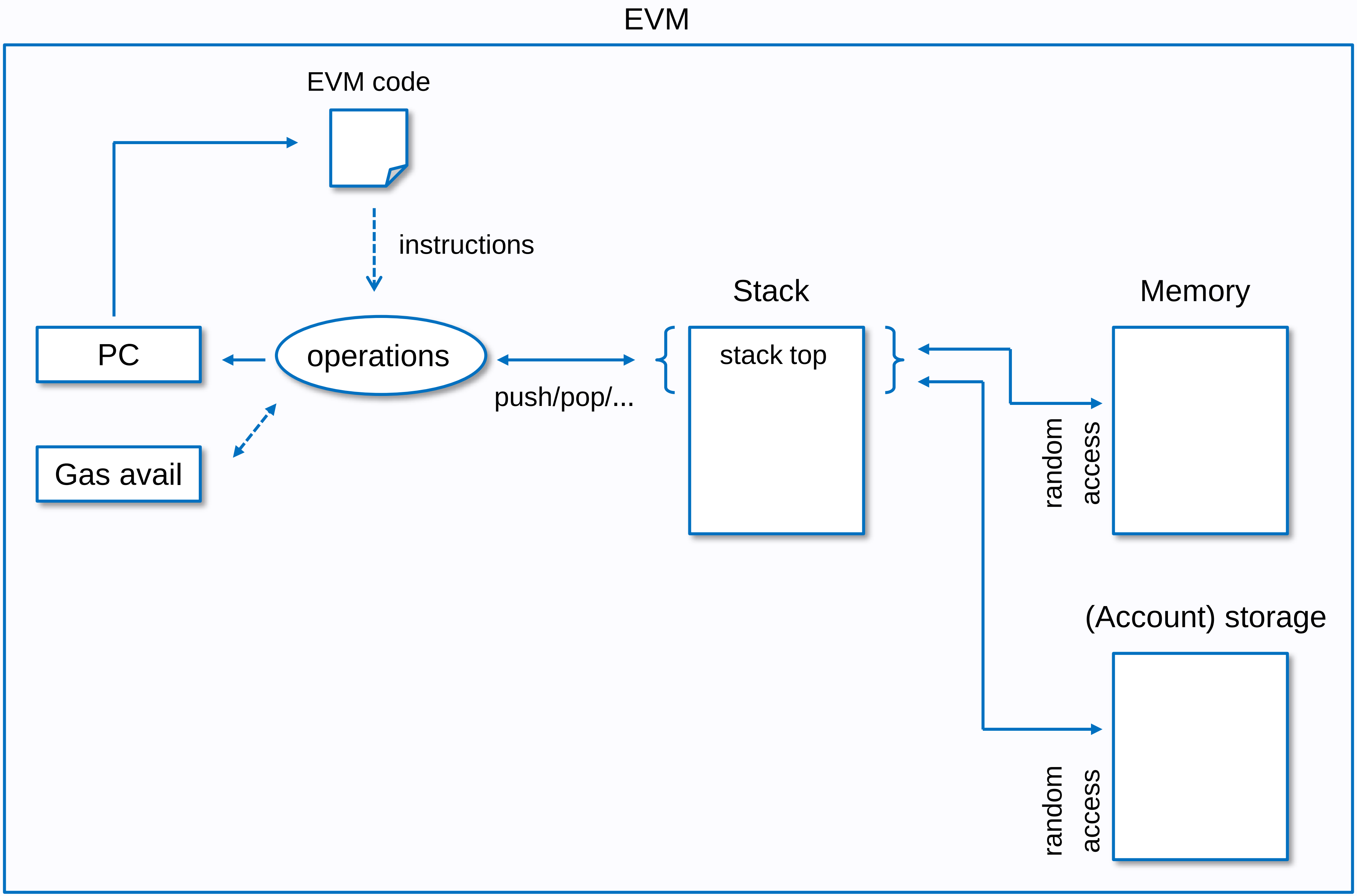
Bytecode view

```
0x60e060020a600035...
```

EVM Code is the bytecode that the EVM can natively execute.

References : [E1] Ch.9, Appendix H

# Execution model

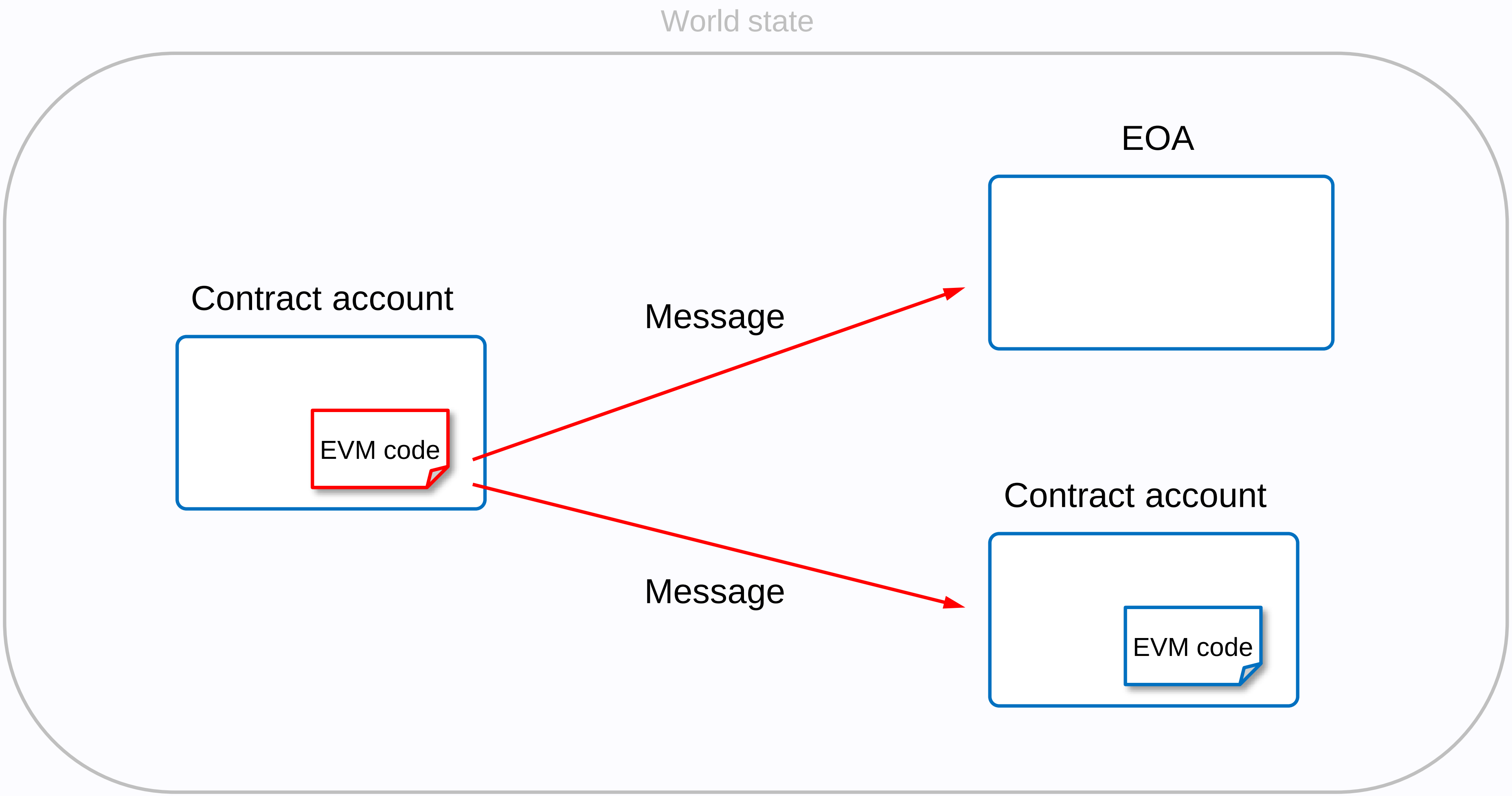


References : [E1] Ch.9, Appendix H

## 2. Virtual machine

# Message call

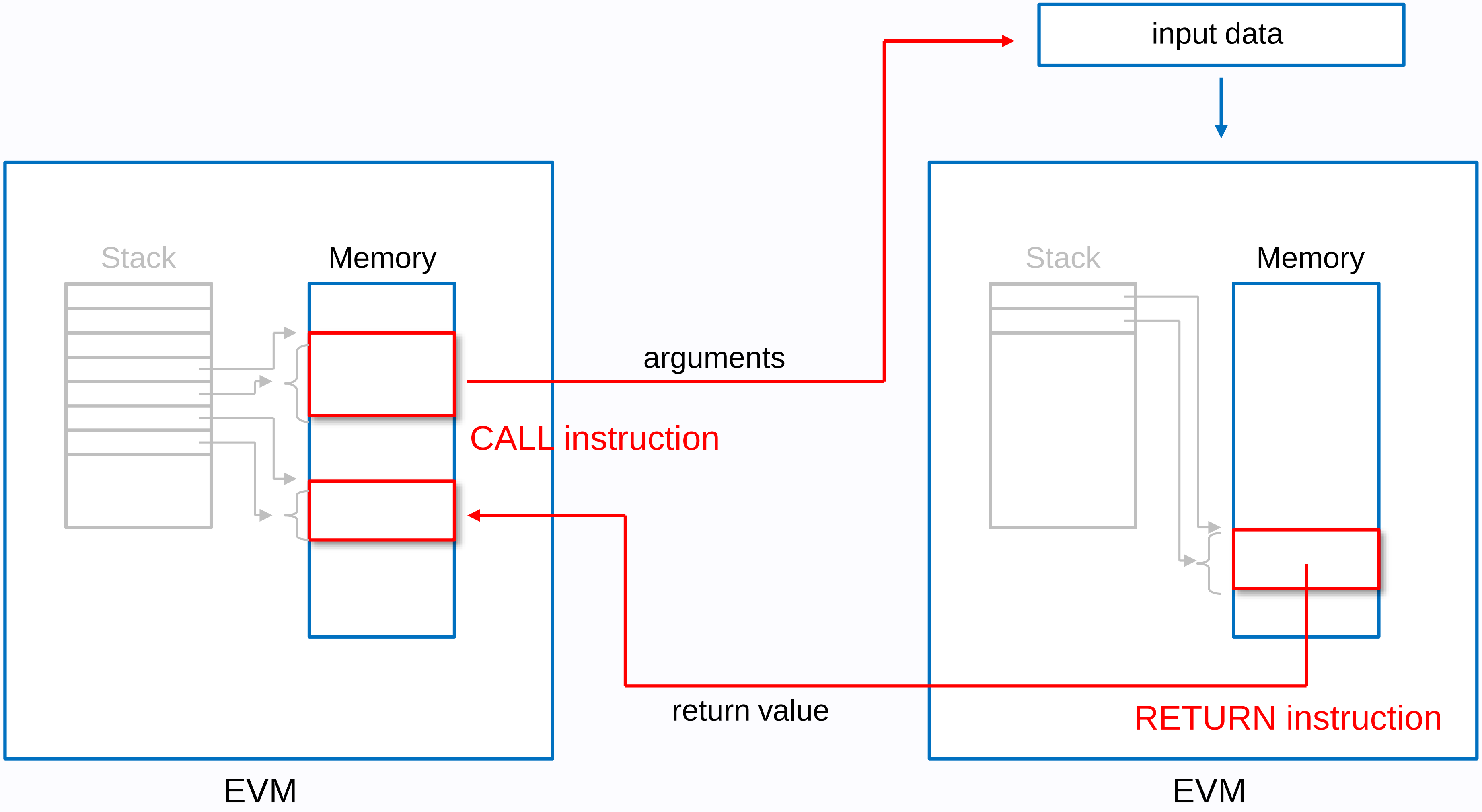
# Message call



EVM can send a message to other account.
The depth of message call is limited to less than 1024 levels.

References : [E1] Ch.8, Appendix A

# Instructions for Message call



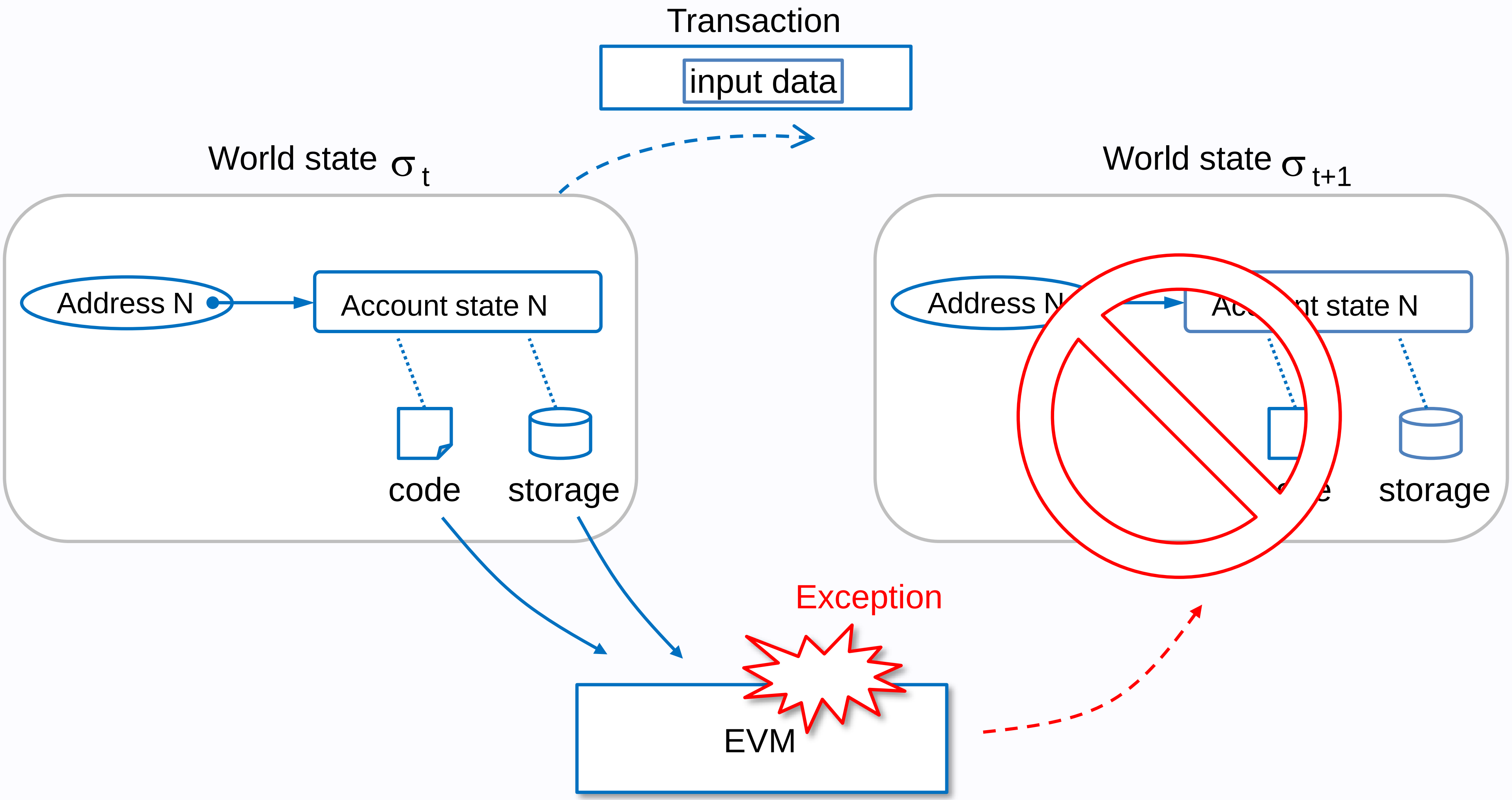
Message call is triggered by CALL instruction.
Arguments and return values are passed using memory.

References : [E1] Ch.8, Ch.9

2. Virtual machine

# Exception

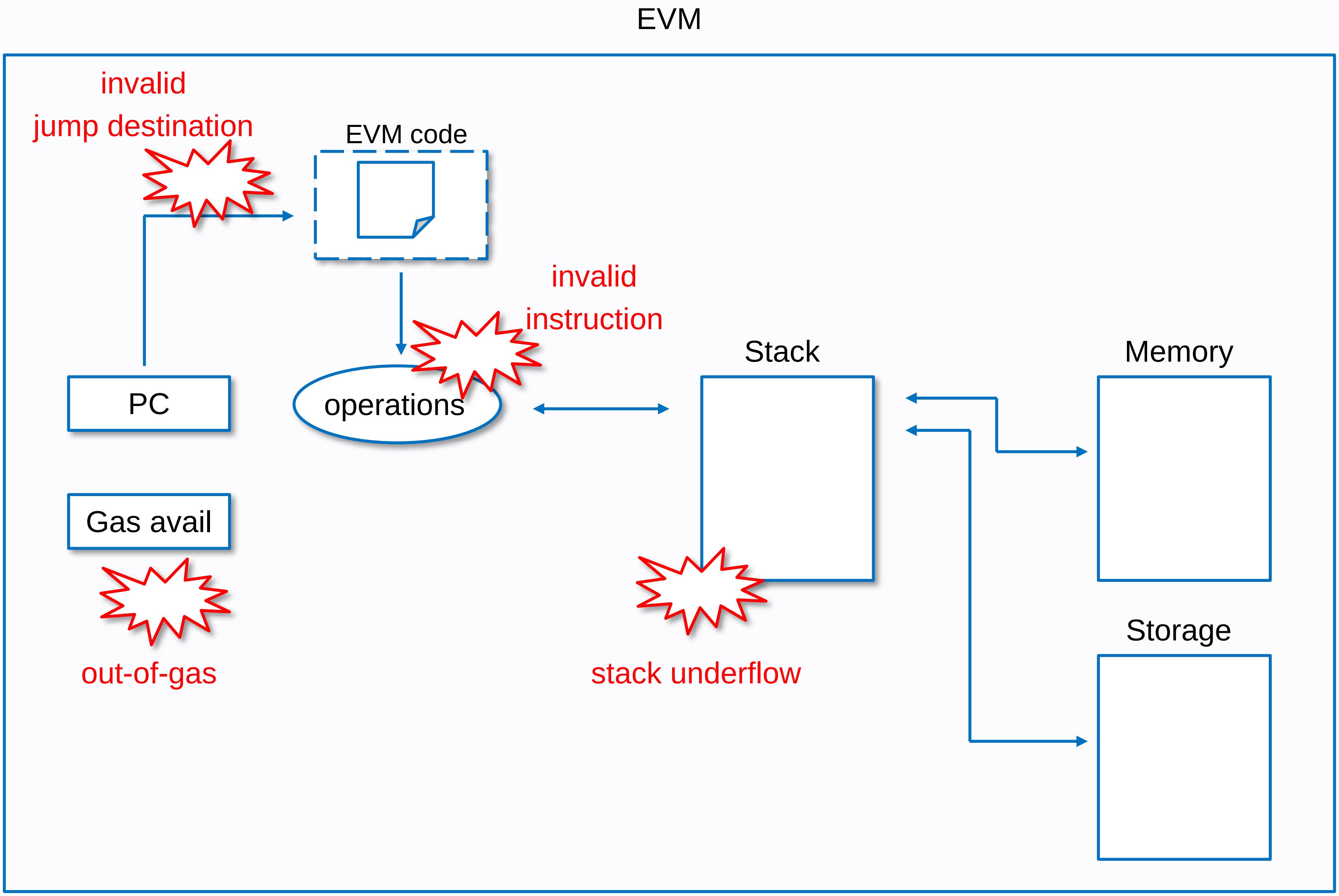
# Exception



If an exception occurs in the EVM, the state is not updated.

References : [E1] Ch.9, Appendix H

# Exception

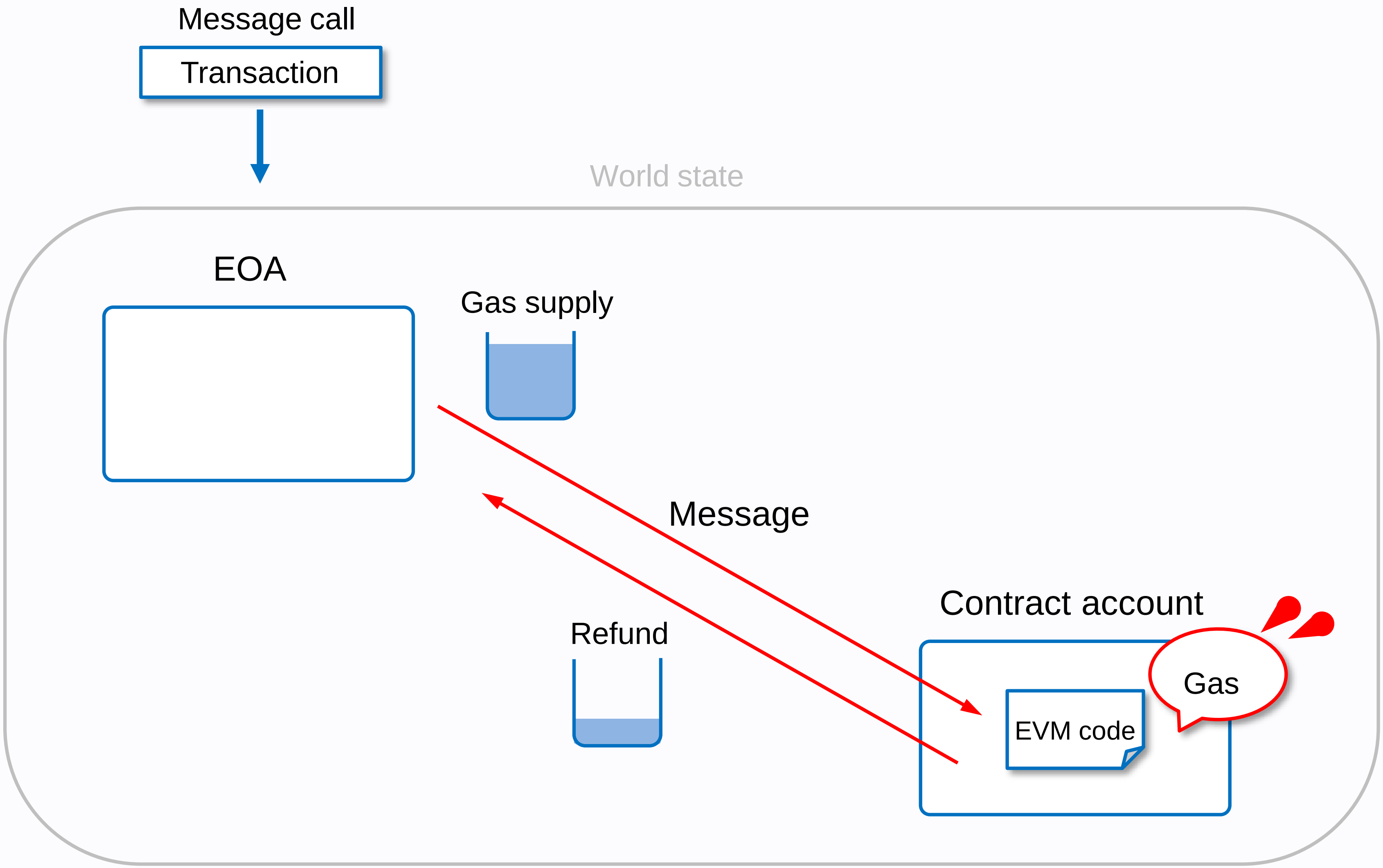


References : [E1] Ch.9, Appendix H

## 2. Virtual machine

# Gas and fee

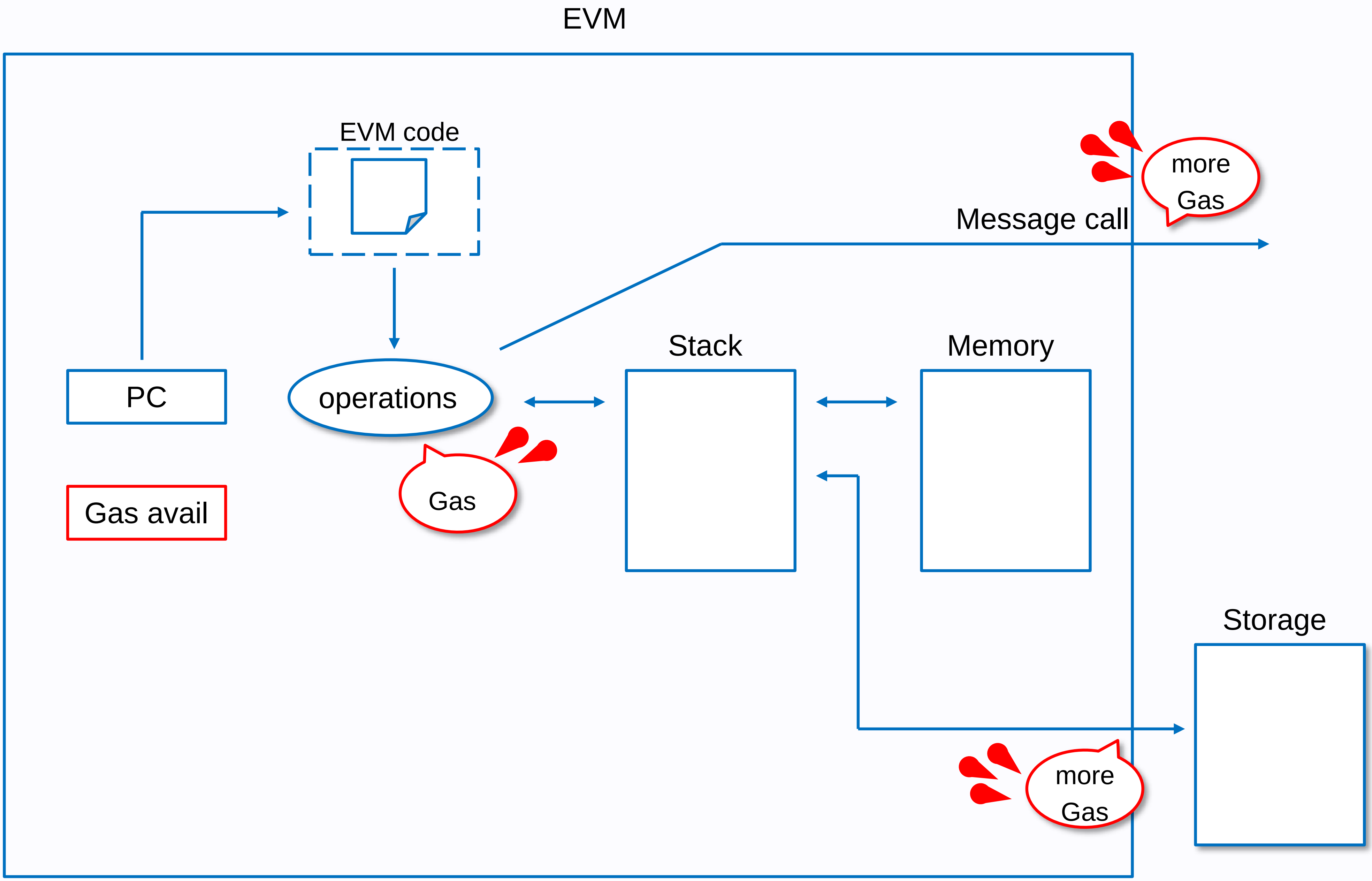
# Gas and fee



All programmable computation in Ethereum is subject to fees (denominated in gas).

References : [E1] Ch.5, Ch.9, Appendix G

# Gas

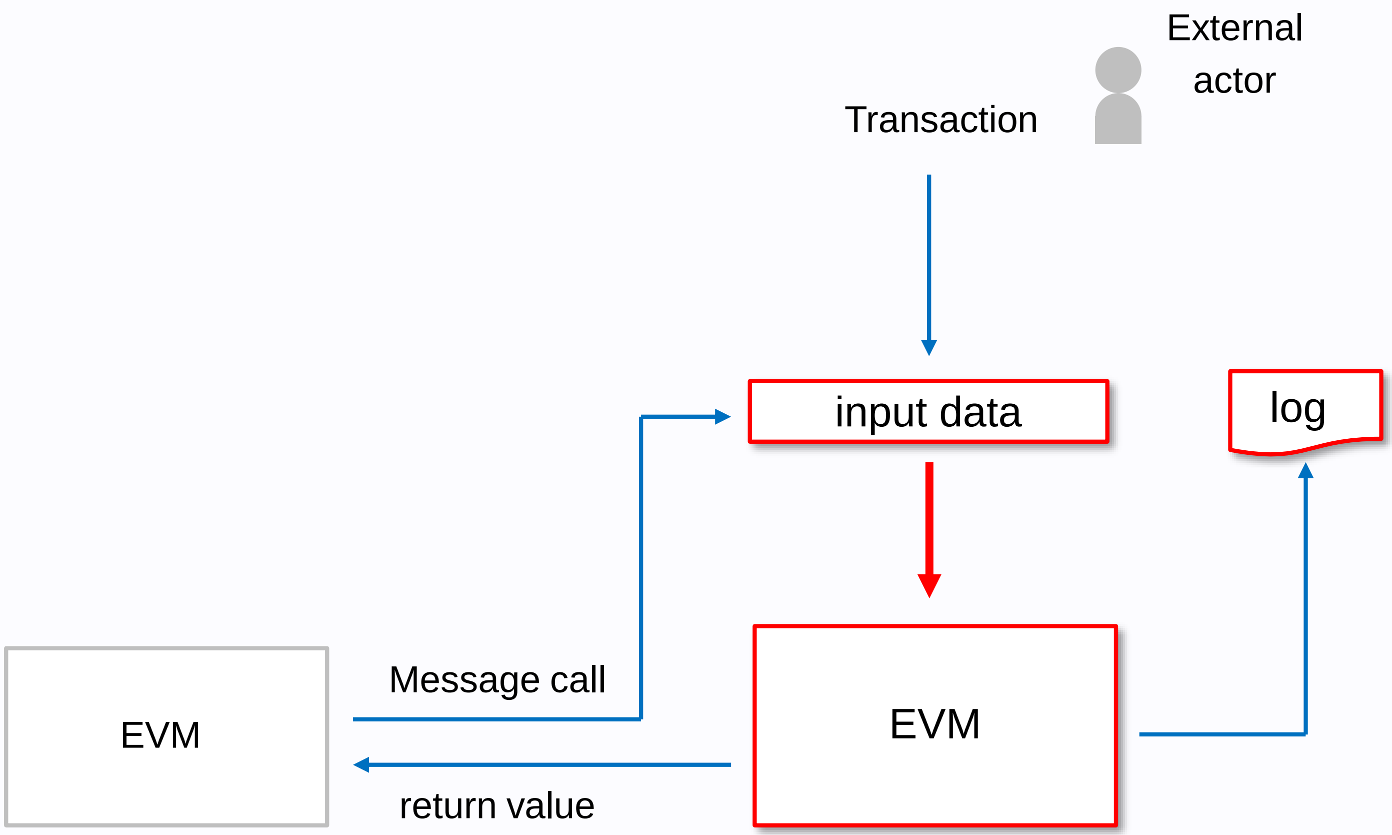


References : [E1] Ch.5, Ch.9, Appendix G

# 2. Virtual machine

## Input and output

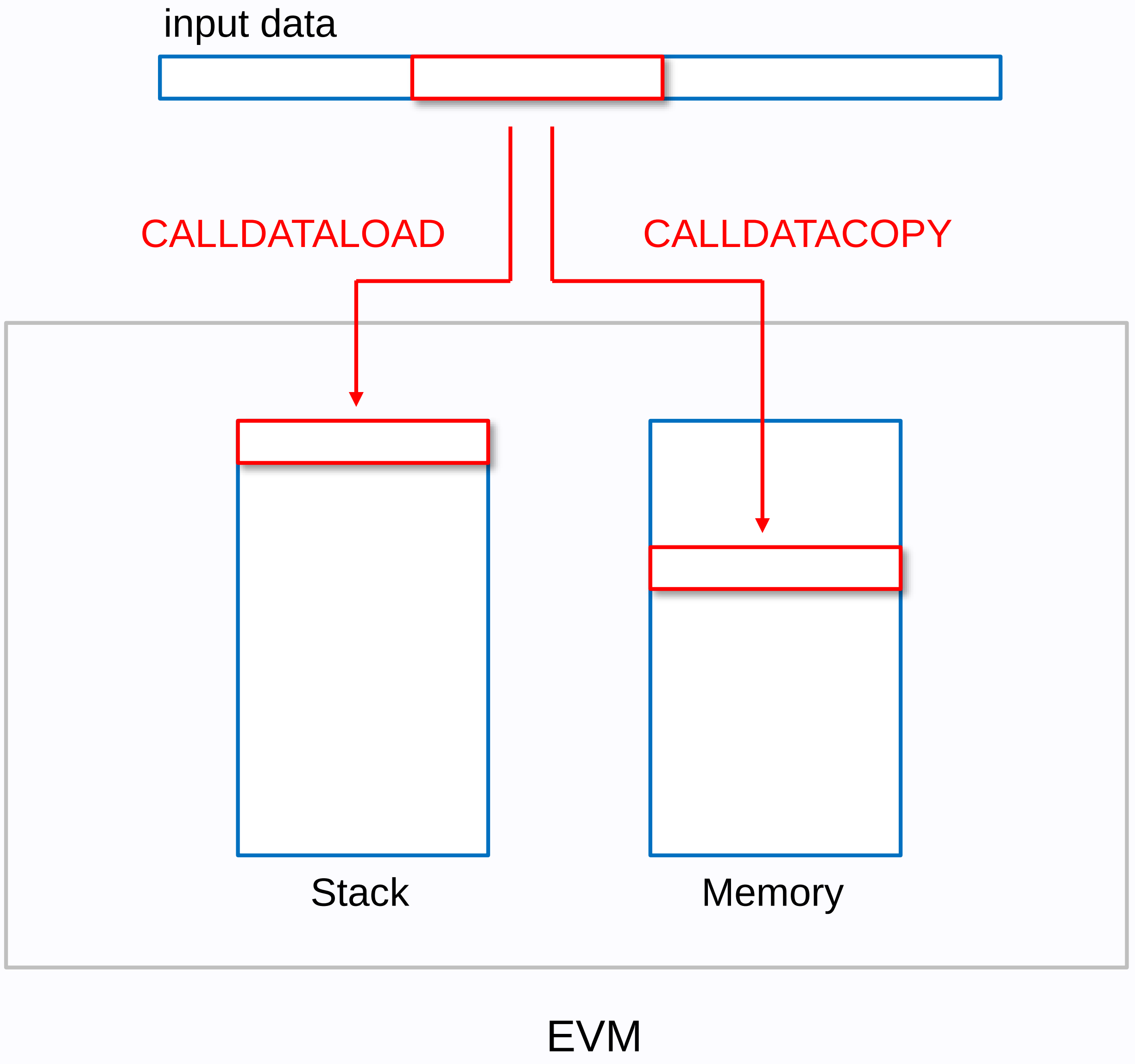
# Input and Output of EVM



EVM can input external data from a message call.
EVM can output log. EVM can also return values to Caller EVM.

References : [E1] Ch.9, Appendix H

# Instructions for input data

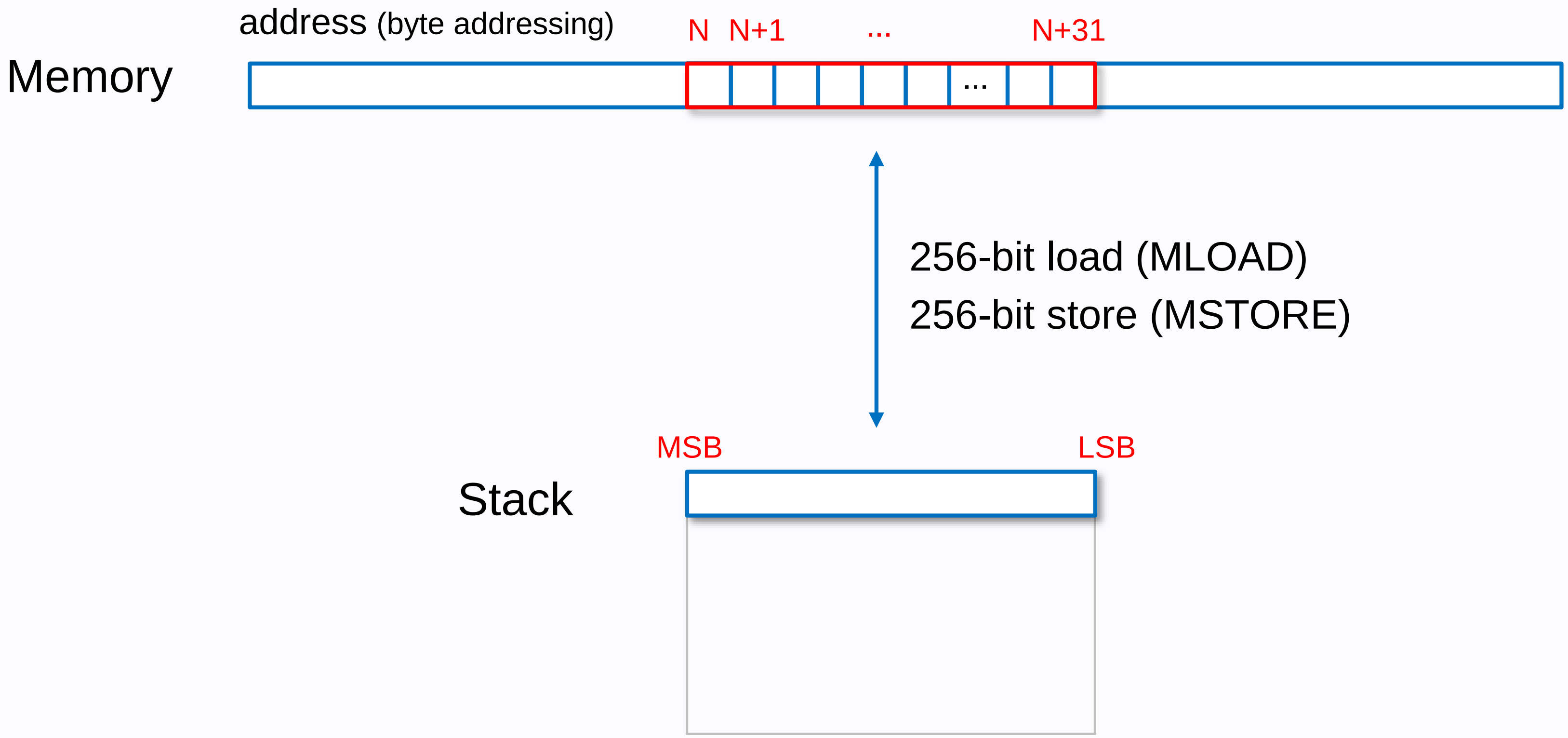


References : [E1] Ch.7, Ch.9

## 2. Virtual machine

# Byte order

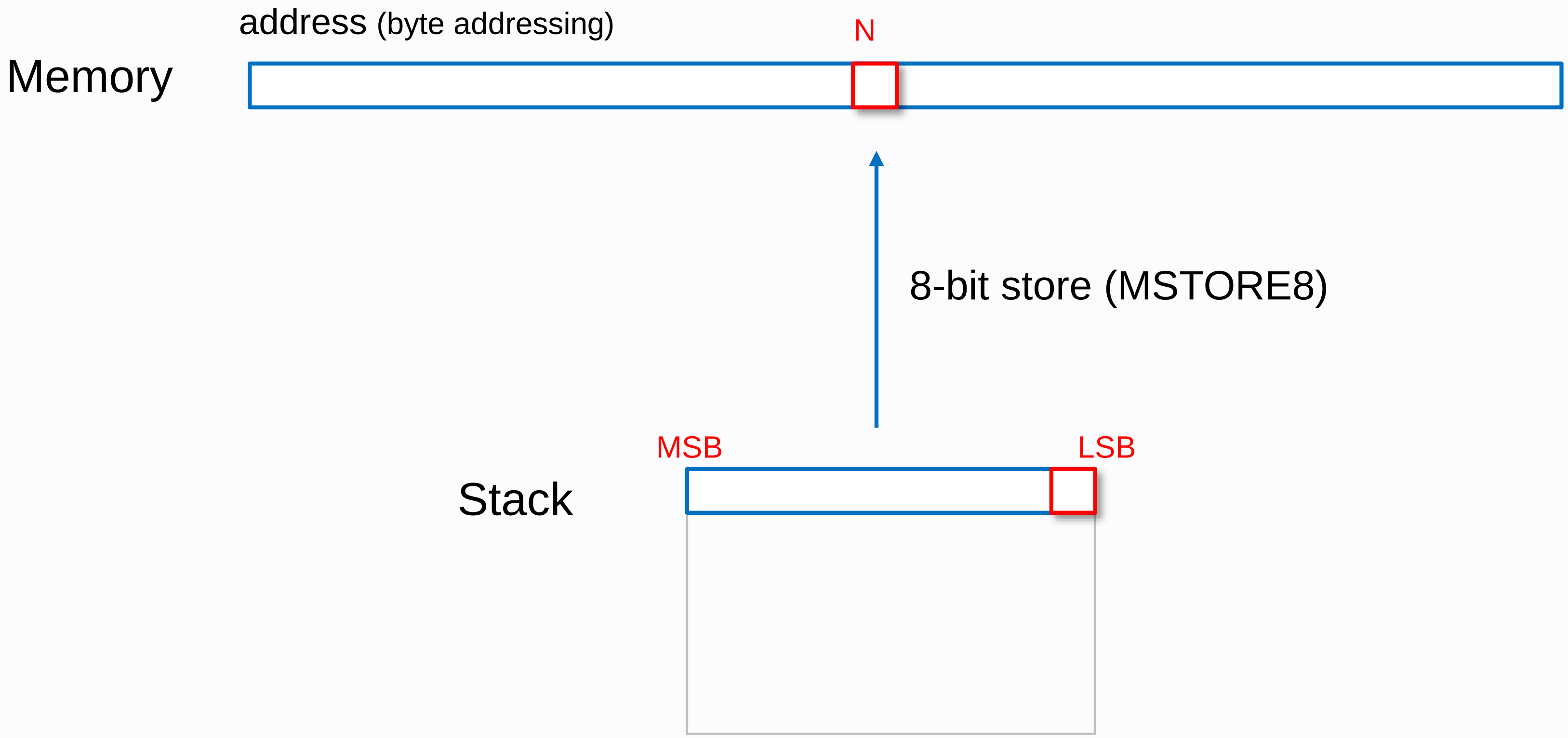
# Endian for Memory



EVM is big endian order (network byte order).

References : [E1] Ch.9, Appendix H, [E7], [W2]

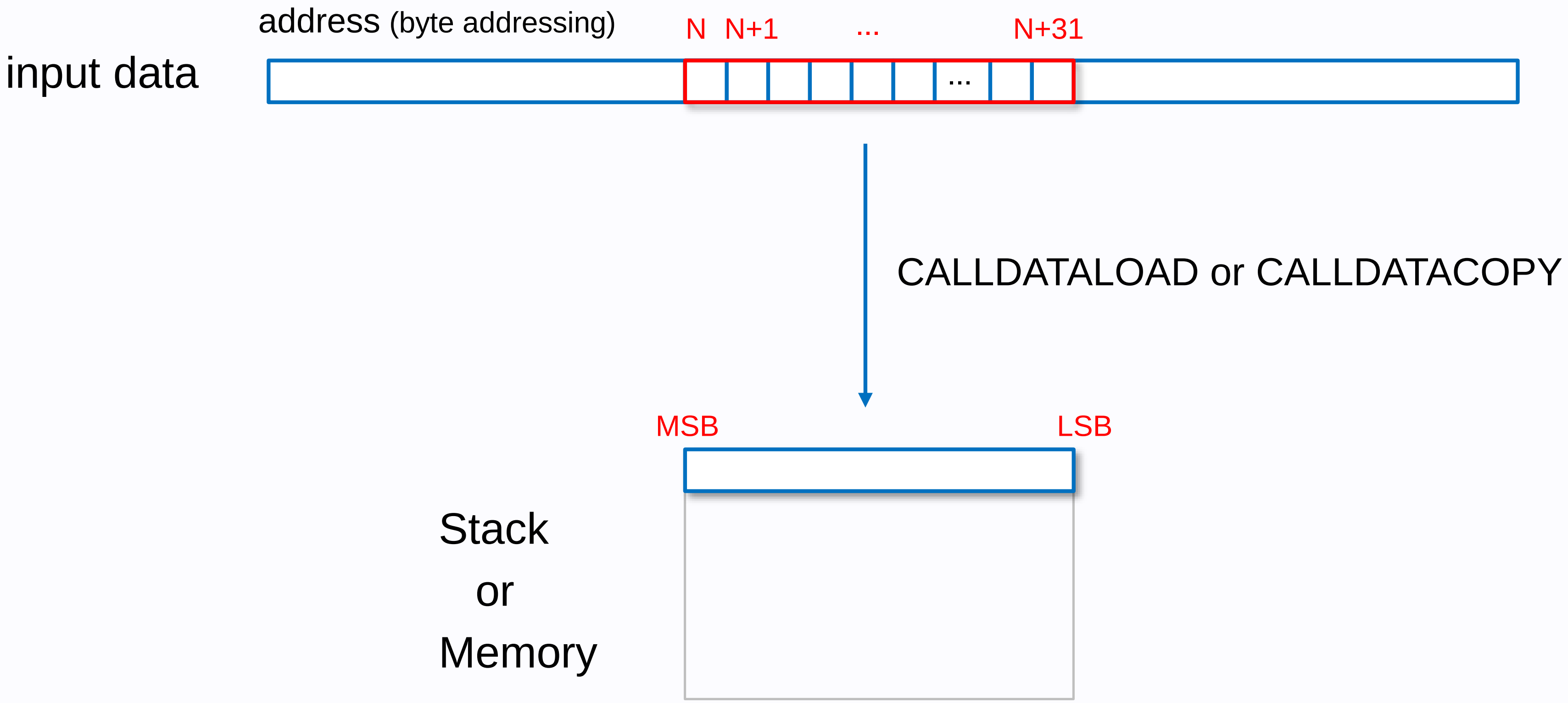
# Endian for Memory



EVM is big endian order (network byte order).

References : [E1] Ch.9, Appendix H, [E7], [W2]

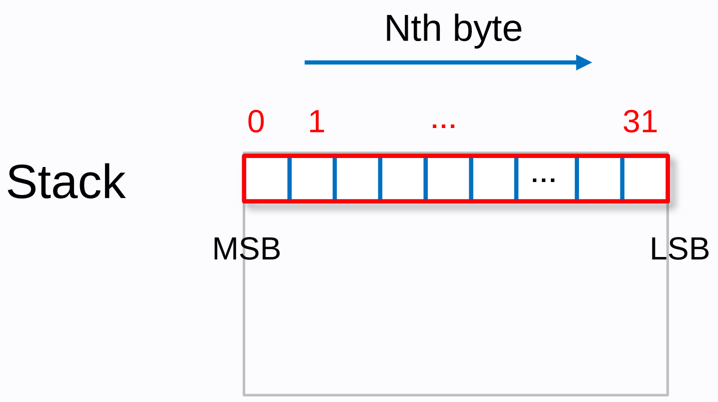
# Endian for input data



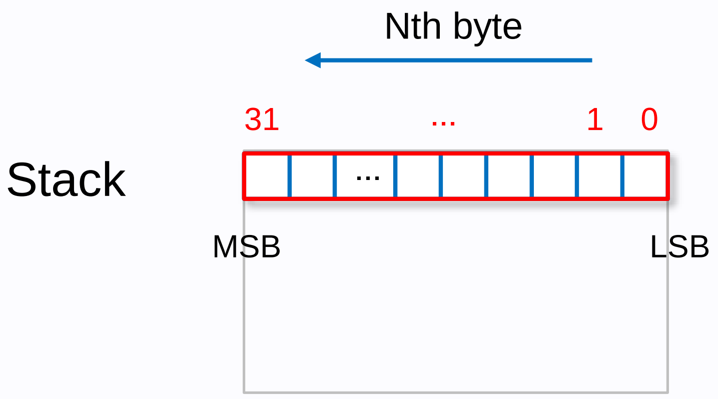
EVM is big endian order (network byte order).

References : [E1] Ch.9, Appendix H, [E7], [W2]

# Byte order of BYTE and SIGNEXTEND instruction



BYTE instruction counts from MSB.



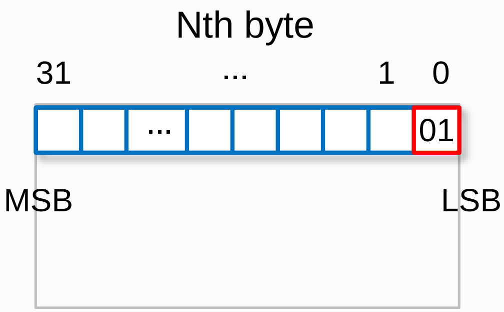
SIGNEXTEND instruction counts from LSB.

References : [E1] Ch.9, Appendix H, [E7], [W2]

# Byte order of PUSH instructions

PUSH1  0x01

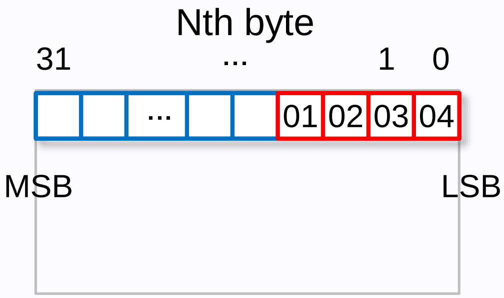
Stack



right-aligned, big endian

PUSH4  0x01020304

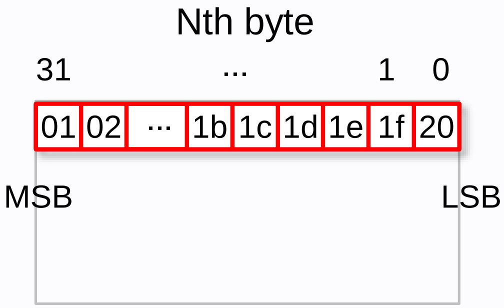
Stack



right-aligned, big endian

PUSH32  0x0102...1f20

Stack



right-aligned, big endian

References : [E1] Ch.9, Appendix H, [E7], [W2]

2. Virtual machine

# Instruction set

# Instruction set

* Basically, 256-bit operation.

* Contract creation and destruct
    * CREATE, DELEGATECALL

* Hash
    * SHA3

* Shift operation
    * using MUL or DIV, SDIV

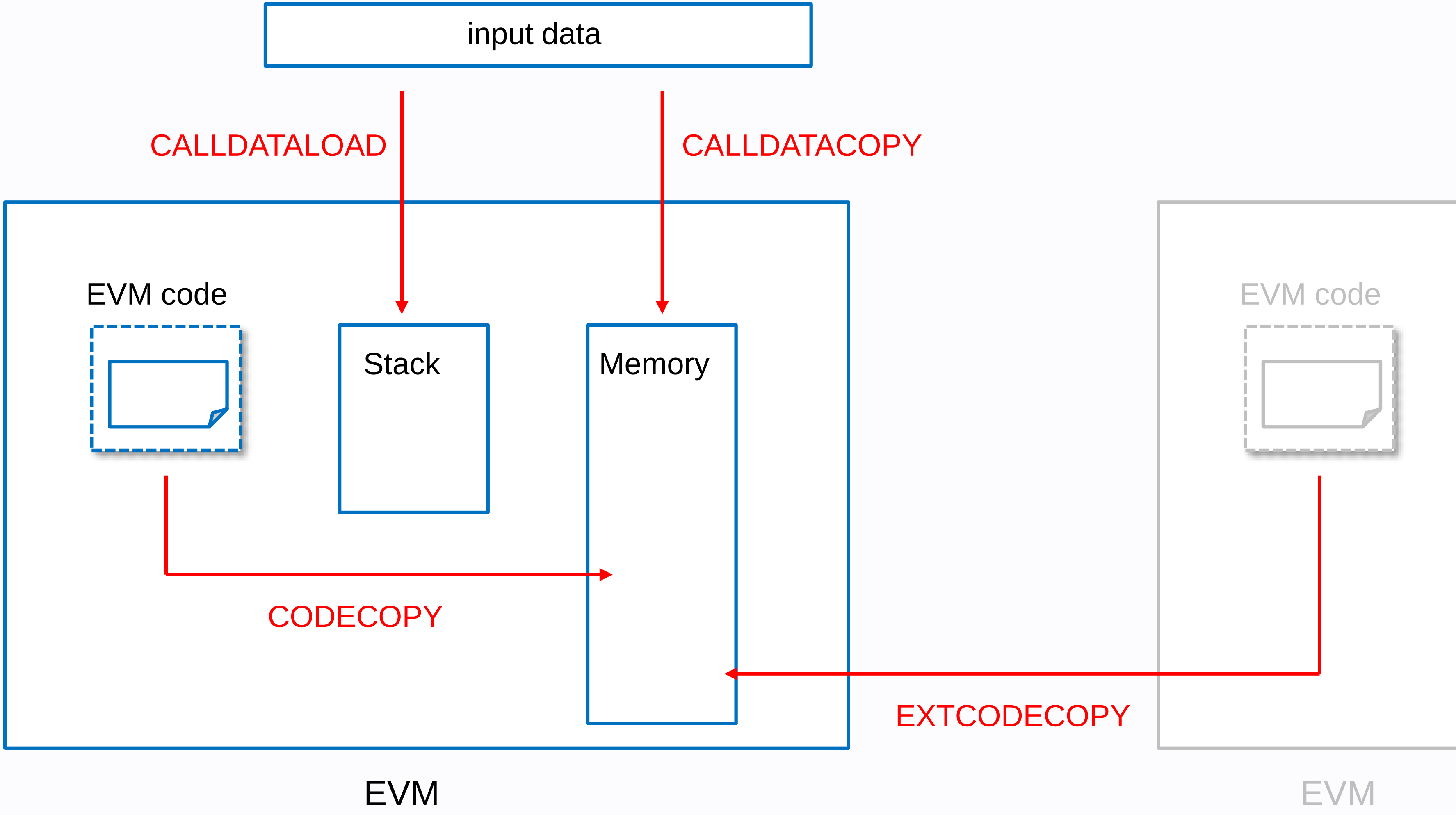
* Div operation
    * without zero divisional exception

* ...

WIP

References : [E1] Ch.9, Appendix H

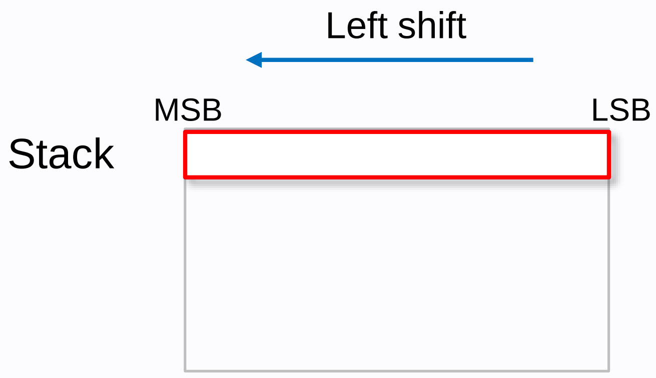
# Copy of code and input data



There are several copy instructions for inter spaces.

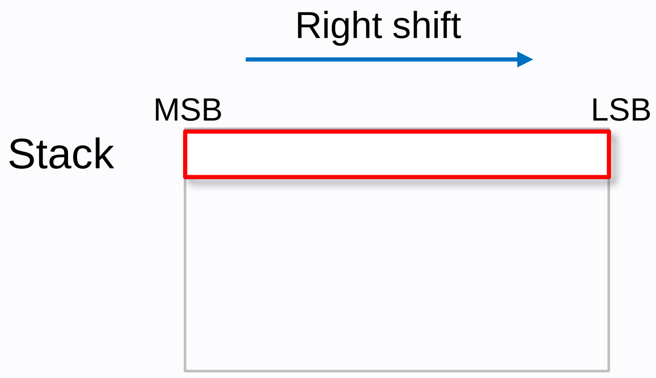
References : [E1] Ch.8, Ch.9

# Shift by MUL, DIV and SDIV



MUL m (2^n) == m << n

Left shift is represented by MUL instruction.



DIV m (2^n) == m >> n

DIV for logical right shift
SDIV for arithmetic right shift

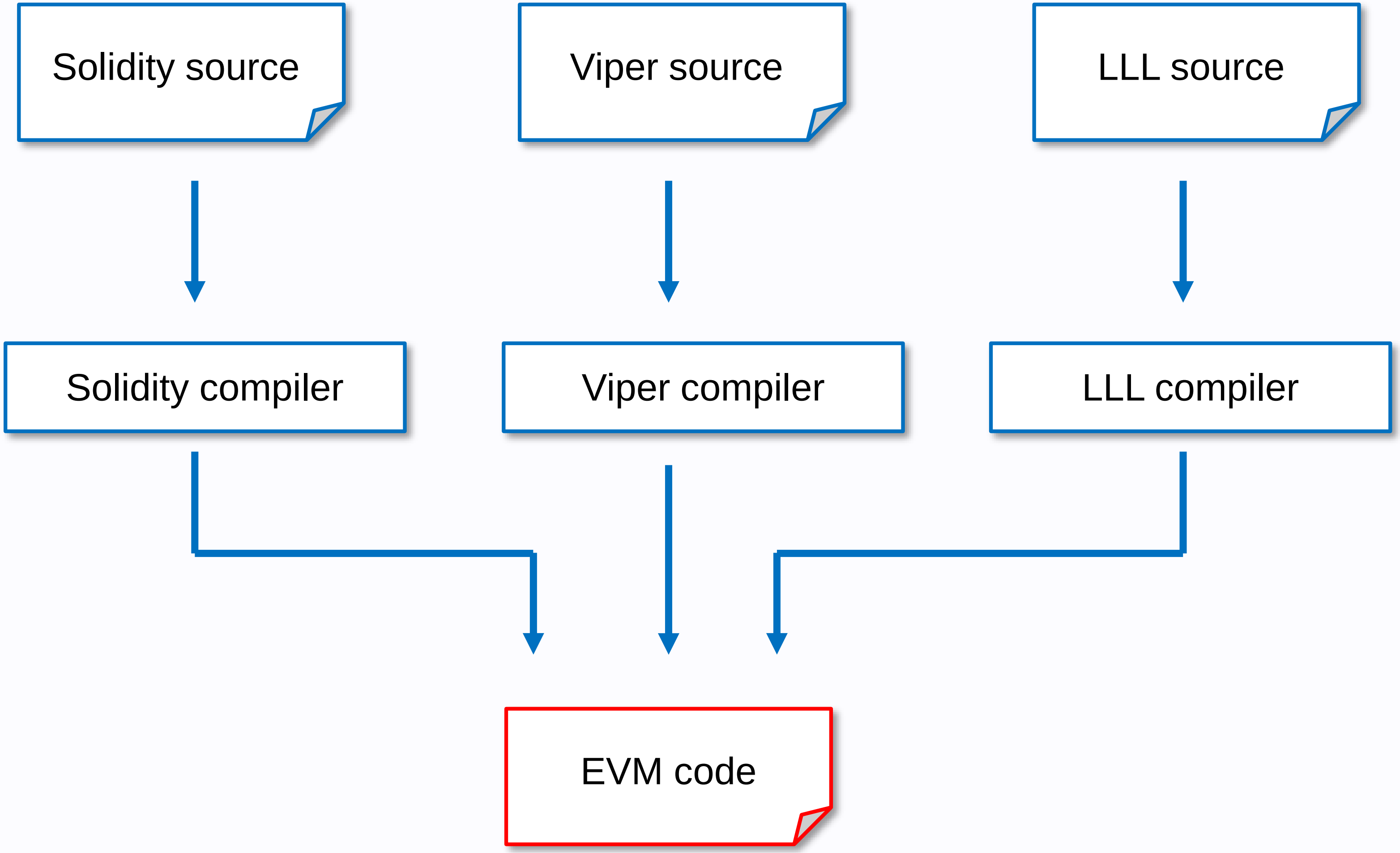
Right shift is represented by DIV and SDIV instruction.

References : [E1] Ch.9, Appendix H, [E7], [W2]

2. Virtual machine

# Miscellaneous

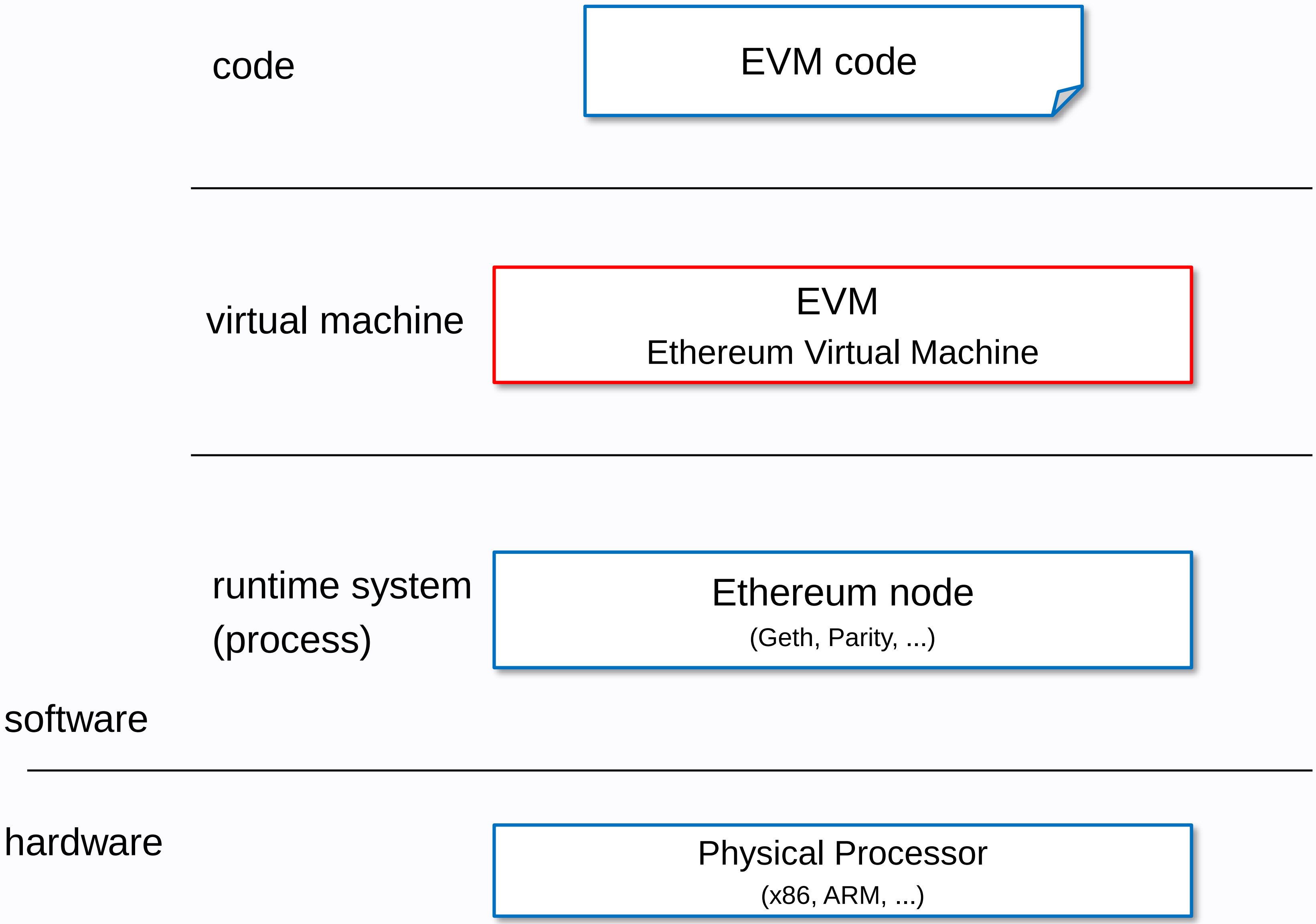
# EVM code generation



Ethereum virtual machine code

References : [E7]

# Ethereum virtual machine layer



code

EVM code

virtual machine

EVM

Ethereum Virtual Machine

runtime system

(process)

Ethereum node

(Geth, Parity, ...)

software

hardware

Physical Processor

(x86, ARM, ...)

References : [E1] Ch.9

# eWASM

The eWASM is next generation VM.

WIP

# Appendix A

# Appendix A

# Source code in Geth

# Block header

(go-ethereum version 1.8)

[core/types/block.go]

```
type Header struct {
        ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
        UncleHash   common.Hash    `json:"sha3Uncles"                     ed"`
        Coinbase    common.Address `json:"miner"            gencodec:"required"`
        Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
        TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
        ReceiptHash common.Hash    `json:"receiptsRoot"                   ed"`
        Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
        Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
        Number      *big.Int       `json:"number"           gencodec:"required"`
        GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
        GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
        Time        *big.Int       `json:"timestamp"        gencodec:"required"`
        Extra       []byte         `json:"extraData"        gencodec:"required"`
        MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
        Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```

Block header

Root of State

Root of Transaction

References : [C1]

# Transaction

(go-ethereum version 1.8)

[core/types/transaction.go]

```
type txdata struct {
        AccountNonce uint64          `json:"nonce"    gencodec:"required"`
        Price        *big.Int        `json:"gasPrice" gen
        GasLimit     uint64          `json:"gas"      ge
        Recipient    *common.Address `json:"to"       rlp:"nil"`
                                               // nil means contract creation
        Amount       *big.Int        `json:"value"    gen
        Payload      []byte          `json:"input"    ge

        // Signature values
        V *big.Int `json:"v" gencodec:"required"`
        R *big.Int `json:"r" gencodec:"required"`
        S *big.Int `json:"s" gencodec:"required"`

        // This is only used when marshaling to JSON.
        Hash *common.Hash `json:"hash" rlp:"-"`
}
```

Transaction

to address

value (Wei)

input data

References : [C1]

# World state

(go-ethereum version 1.8)

[core/state/statedb.go]

```
type StateDB struct {
        db   Database
        trie Trie

        stateObjects      map[common.Address]*stateObject
        stateObjectsDirty map[common.Address]struct{}

        dbErr error

        refund uint64

        thash, bhash common.Hash
        txIndex      int
        logs         map[common.Hash][]*types.Log
        logSize      uint

        preimages map[common.Hash][]byte

           :
```

World state

Mapping for
Address to Account state

References : [C1]

# Account object (state object)

(go-ethereum version 1.8)

[core/state/state_object.go]

```
type stateObject struct {
        address  common.Address
        addrHash common.Hash
        data     Account
        db       *StateDB

        dbErr error

        trie Trie // storage trie, which becomes non-nil on first access
        code Code // contract bytecode, which gets set when code is loaded

        cachedStorage Storage // Storage entry cache to avoid duplicate reads
        dirtyStorage  Storage // Storage entries that need to be flushed to disk

        dirtyCode bool // true if the code was updated
        suicided  bool
        touched   bool
        deleted   bool
        onDirty   func(addr common.Address)
}
```

Address

Account state

References : [C1]

# Account state, Code and Storage

(go-ethereum version 1.8)

[core/state/state_object.go]

```
type Account struct {
        Nonce    uint64
        Balance  *big.Int
        Root     common.Hash // merkle root of the storage trie
        CodeHash []byte
}
```

Account state

```
type Code []byte
```

EVM code

```
type Storage map[common.Hash]common.Hash
```

Account storage

References : [C1]

# Stack and Memory

(go-ethereum version 1.8)

[core/vm/stack.go]

```
type Stack struct {
        data []*big.Int
}

func newstack() *Stack {
        return &Stack{data: make([]*big.Int, 0, 1024)}
}
```

Stack

[core/vm/memory.go]

```
type Memory struct {
        store       []byte
        lastGasCost uint64
}

func NewMemory() *Memory {
        return &Memory{}
}
```

Memory

References : [C1]

# Instruction operation (arithmetic and stack)

(go-ethereum version 1.8)

[core/vm/instruction.go]

Arithmetic operation

```
func opAdd(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack)
([]byte, error) {
        x, y := stack.pop(), stack.pop()
        stack.push(math.U256(x.Add(x, y)))

        evm.interpreter.intPool.put(y)

        return nil, nil
}
```

Stack operation

```
func opPop(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack)
([]byte, error) {
        evm.interpreter.intPool.put(stack.pop())
        return nil, nil
}
```

References : [C1]

# Instruction operation (memory and storage)

(go-ethereum version 1.8)

[core/vm/instruction.go]

Memory operation

```
func opMload(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack
*Stack) ([]byte, error) {
        offset := stack.pop()
        val := new(big.Int).SetBytes(memory.Get(offset.Int64(), 32))
        stack.push(val)

        evm.interpreter.intPool.put(offset)
        return nil, nil
}
```

Storage operation

```
func opSload(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack
*Stack) ([]byte, error) {
        loc := common.BigToHash(stack.pop())
        val := evm.StateDB.GetState(contract.Address(), loc).Big()
        stack.push(val)
        return nil, nil
}
```

References : [C1]

# Instruction operation (call)

(go-ethereum version 1.8)

[core/vm/instruction.go]

Flow operation

```
func opCall(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack)
([]byte, error) {
        // Pop gas. The actual gas in in evm.callGasTemp.
        evm.interpreter.intPool.put(stack.pop())
        gas := evm.callGasTemp
        // Pop other call parameters.
        addr, value, inOffset, inSize, retOffset, retSize := stack.pop(),
            stack.pop(), stack.pop(), stack.pop(), stack.pop(), stack.pop()
        toAddr := common.BigToAddress(addr)
        value = math.U256(value)
        // Get the arguments from the memory.
        args := memory.Get(inOffset.Int64(), inSize.Int64())

        if value.Sign() != 0 {
                gas += params.CallStipend
        }
        ret, returnGas, err := evm.Call(contract, toAddr, args, gas, value)
        if err != nil {
            :
```

References : [C1]

# Gas

(go-ethereum version 1.8)

[core/vm/gas.go]

```
const (
        GasQuickStep   uint64 = 2
        GasFastestStep uint64 = 3
        GasFastStep    uint64 = 5
        GasMidStep     uint64 = 8
        GasSlowStep    uint64 = 10
        GasExtStep     uint64 = 20

        GasReturn       uint64 = 0
        GasStop         uint64 = 0
        GasContractByte uint64 = 200
)
```

$G_{base}$

$G_{verylow}$

[core/vm/gas_table.go]

```
func gasSStore(gt params.GasTable, evm *EVM, contract *Contract, stack *Stack, mem
*Memory, memorySize uint64) (uint64, error) {
        var (
                y, x = stack.Back(1), stack.Back(0)
                val  = evm.StateDB.GetState(contract.Address(),
         :
```

References : [C1]

# Interpreter

(go-ethereum version 1.8)

[core/vm/interpreter.go]

```
func (in *Interpreter) Run(contract *Contract, input []byte) (ret []byte, err
error) {
        // Increment the call depth which is restricted to 1024
        in.evm.depth++
        defer func() { in.evm.depth-- }()

        in.returnData = nil

        if len(contract.Code) == 0 {
                return nil, nil
        }

        codehash := contract.CodeHash // codehash is used when doing jump dest caching
        if codehash == (common.Hash{}) {
                codehash = crypto.Keccak256Hash(contract.Code)
        }

        var (
                op    OpCode        // current opcode
                mem   = NewMemory() // bound memory
                stack = newstack()  // local stack
         :
```

increment call depth

create Memory

create Stack

References : [C1]

# ApplyTransaction

(go-ethereum version 1.8)

[core/state_processor.go]

```
func ApplyTransaction(config *params.ChainConfig, bc *BlockChain, author
*common.Address, gp *GasPool, statedb *state.StateDB, header *types.Header, tx
*types.Transaction, usedGas *uint64, cfg vm.Config) (*types.Receipt, uint64, error)
{
        msg, err := tx.AsMessage(types.MakeSigner(config, header.Number))
        if err != nil {
                return nil, 0, err
        }
        // Create a new context to be used in the EVM environment
        context := NewEVMContext(msg, header, bc, author)
        // Create a new environment which holds all relevant information
        // about the transaction and calling mechanisms.
        vmenv := vm.NewEVM(context, statedb, config, cfg)
        // Apply the transaction to the current state (included in the env)
        _, gas, failed, err := ApplyMessage(vmenv, msg, gp)
        if err != nil {
                return nil, 0, err
        }
        // Update the state with pending changes
        var root []byte
        if config.IsByzantium(header.Number) {
            :
```

create EVM

References : [C1]

# Version of EVM instruction set

(go-ethereum version 1.8)

[core/vm/interpreter.go]

```
func NewInterpreter(evm *EVM, cfg Config) *Interpreter {
if !cfg.JumpTable[STOP].valid {
                switch {
                case evm.ChainConfig().IsByzantium(evm.BlockNumber):
                        cfg.JumpTable = byzantiumInstructionSet
                case evm.ChainConfig().IsHomestead(evm.BlockNumber):
                        cfg.JumpTable = homesteadInstructionSet
                default:
                        cfg.JumpTable = frontierInstructionSet
                  :
```

added instructions:
STATICCALL, RETURNDATASIZE,
RETURNDATACOPY and REVERT

added instruction:
DELEGATECALL

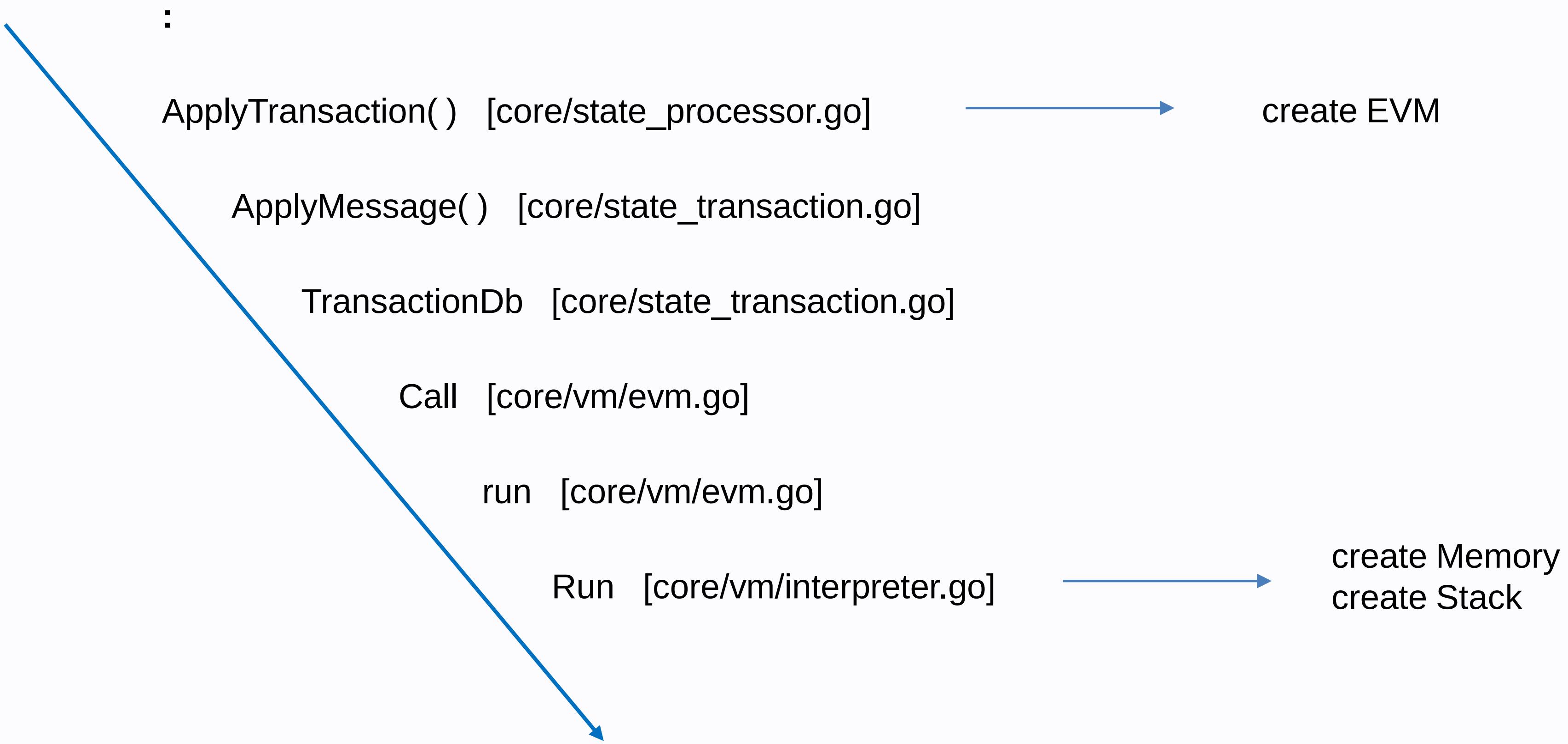
[core/config.go]

```
var (
 MainnetChainConfig = &ChainConfig{
                ChainId:        big.NewInt(1),
                HomesteadBlock: big.NewInt(1150000),
                DAOForkBlock:   big.NewInt(1920000),
                DAOForkSupport: true,
                EIP150Block:    big.NewInt(2463000),
                EIP150Hash: common.HexToHash("0x2086799aeebeae135c246c65021c82b4e15a2c451340993a
                EIP155Block:    big.NewInt(2675000),
                EIP158Block:    big.NewInt(2675000),
                ByzantiumBlock: big.NewInt(4370000),
                   :
```

References : [C1]

# Bootstrap of EVM in Geth

(go-ethereum version 1.8)



:

ApplyTransaction( ) [core/state_processor.go] → create EVM

ApplyMessage( ) [core/state_transaction.go]

TransactionDb [core/state_transaction.go]

Call [core/vm/evm.go]

run [core/vm/evm.go]

Run [core/vm/interpreter.go] → create Memory
create Stack

References : [C1]

# Appendix A

## EVM developer utility

# Example of evm command

(go-ethereum version 1.8)

The go-ethereum project provides evm utility command.

Compile EVM assembly code

```
$ cat sample.asm
push 0x1
push 0x2
add

$ evm compile sample.asm
6001600201
```

Disassemble EVM bytecode

```
$ cat sample.bin
6001600201

$ evm disasm sample.bin
000000: PUSH1 0x01
000002: PUSH1 0x02
000004: ADD
```

References : [C1]

# Example of evm command

(go-ethereum version 1.8)

Run EVM assembly code

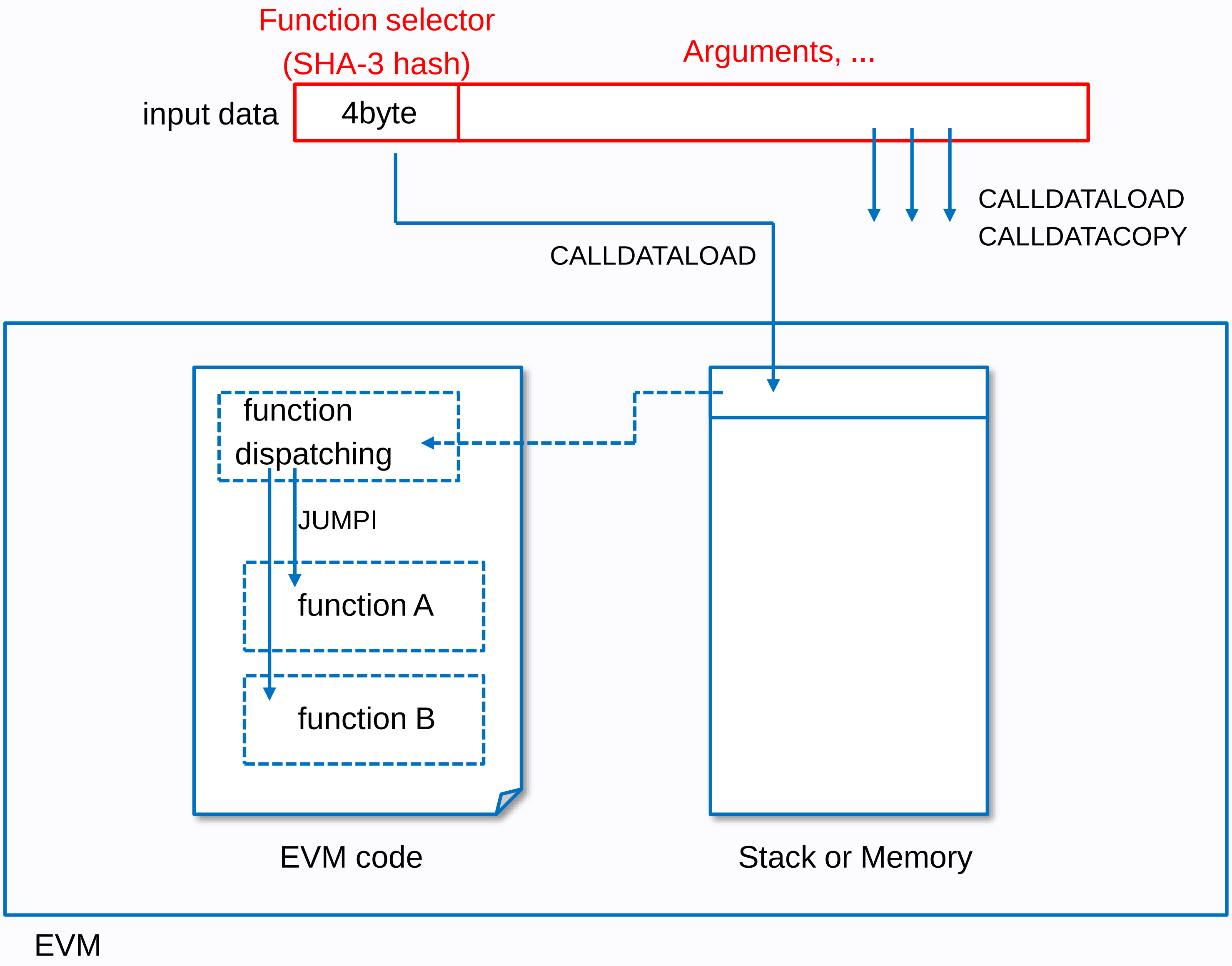
```
$ evm --debug run sample.asm

#### TRACE ####
PUSH1           pc=00000000 gas=10000000000 cost=3

PUSH1           pc=00000002 gas=9999999997 cost=3
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000001

ADD             pc=00000004 gas=9999999994 cost=3
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000002
00000001  0000000000000000000000000000000000000000000000000000000000000001

STOP            pc=00000005 gas=9999999991 cost=0
Stack:
00000000  0000000000000000000000000000000000000000000000000000000000000003

#### LOGS ####
```

References : [C1]

# Appendix A

## Solidity ABI

# Solidity Application Binary Interface



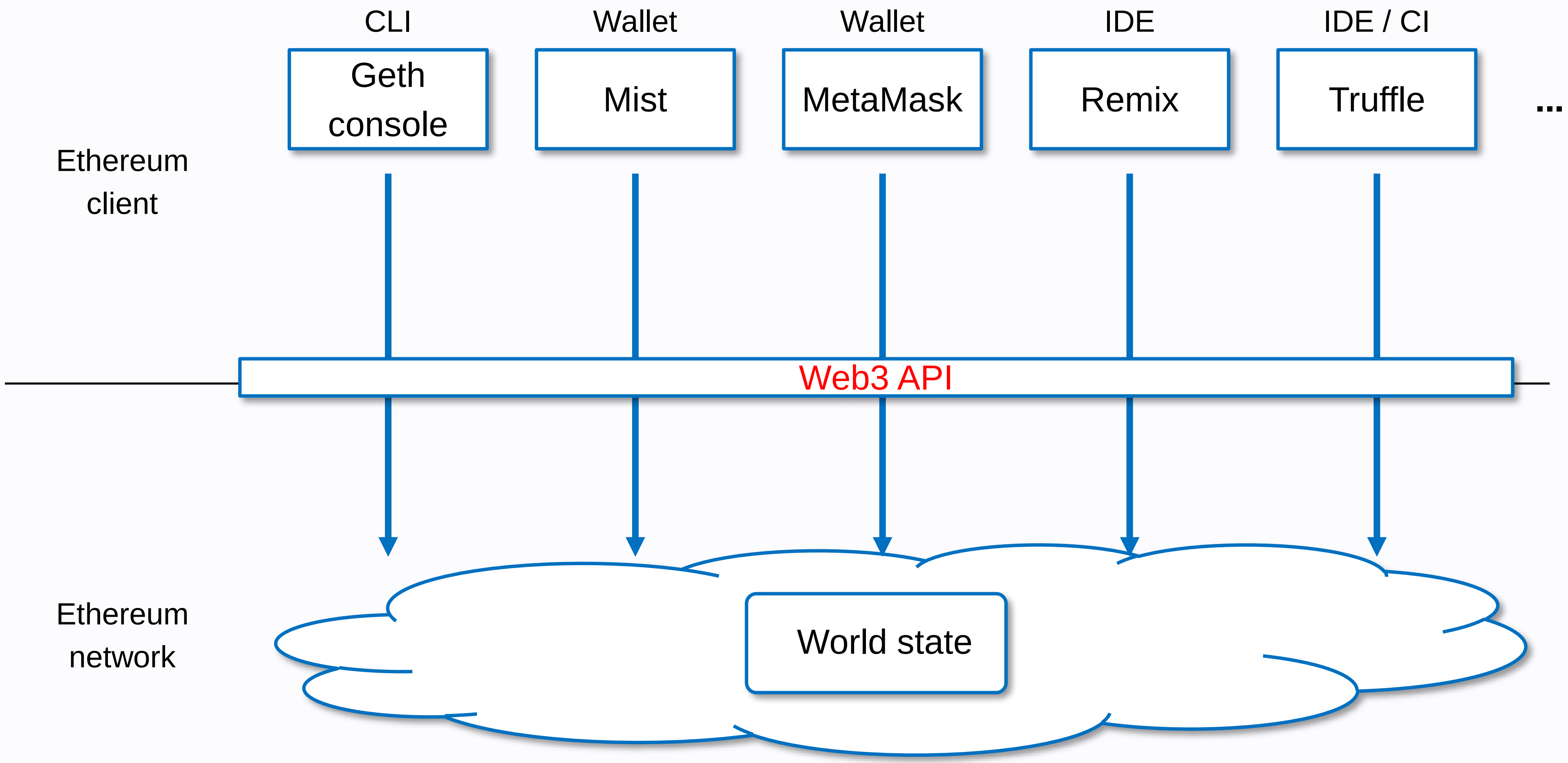
References : [E7]. Ch.7, [E1] Ch.9, Appendix H, [W4], [W2]

# Appendix B

# Appendix B

## Web3 API

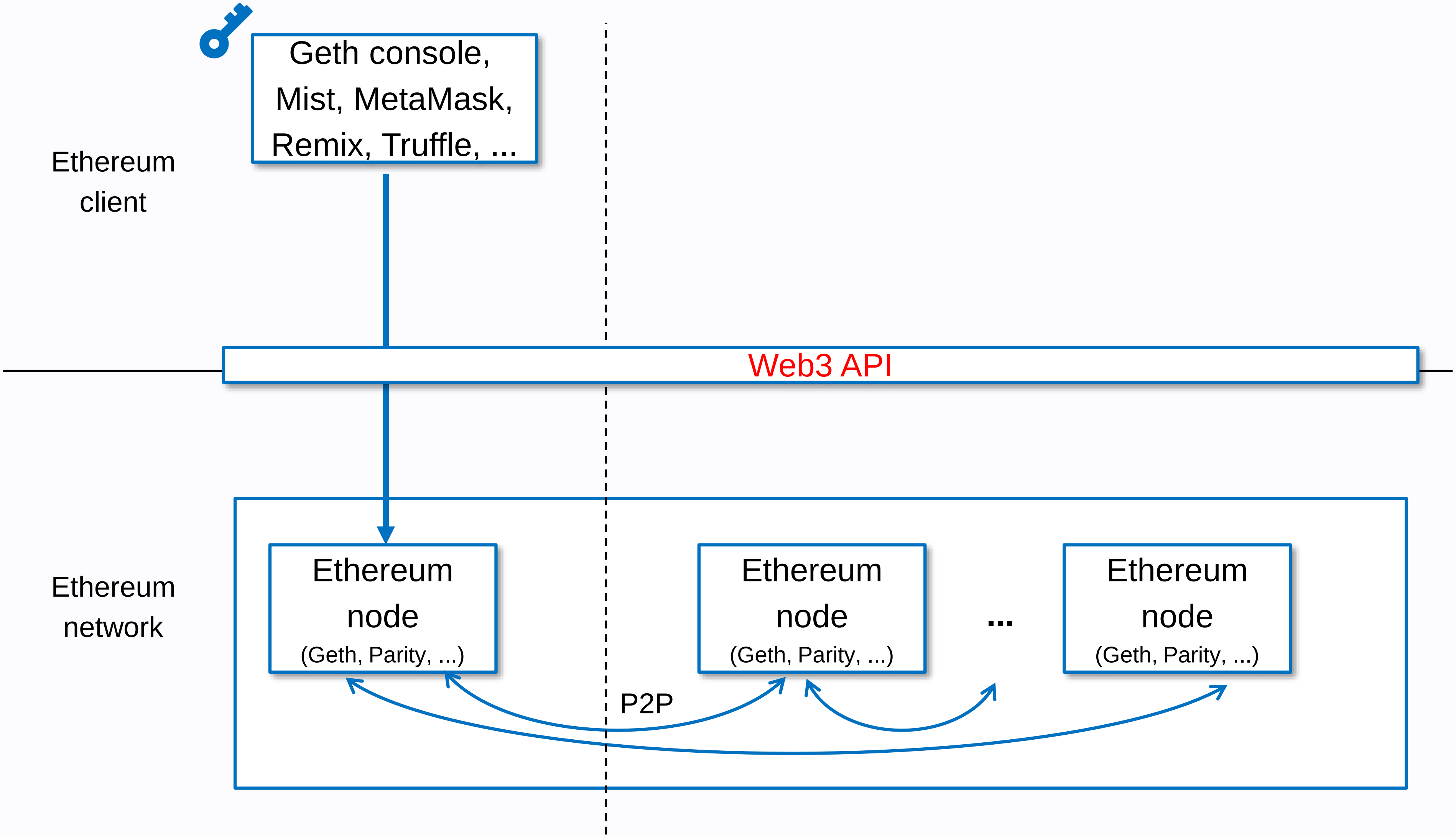
# Web3 API and client



Ethereum clients access to Ethereum network via Web3 API.

References : [E8], [C1], [C3], [C4], [C5], [C6]

# Web3 API and client



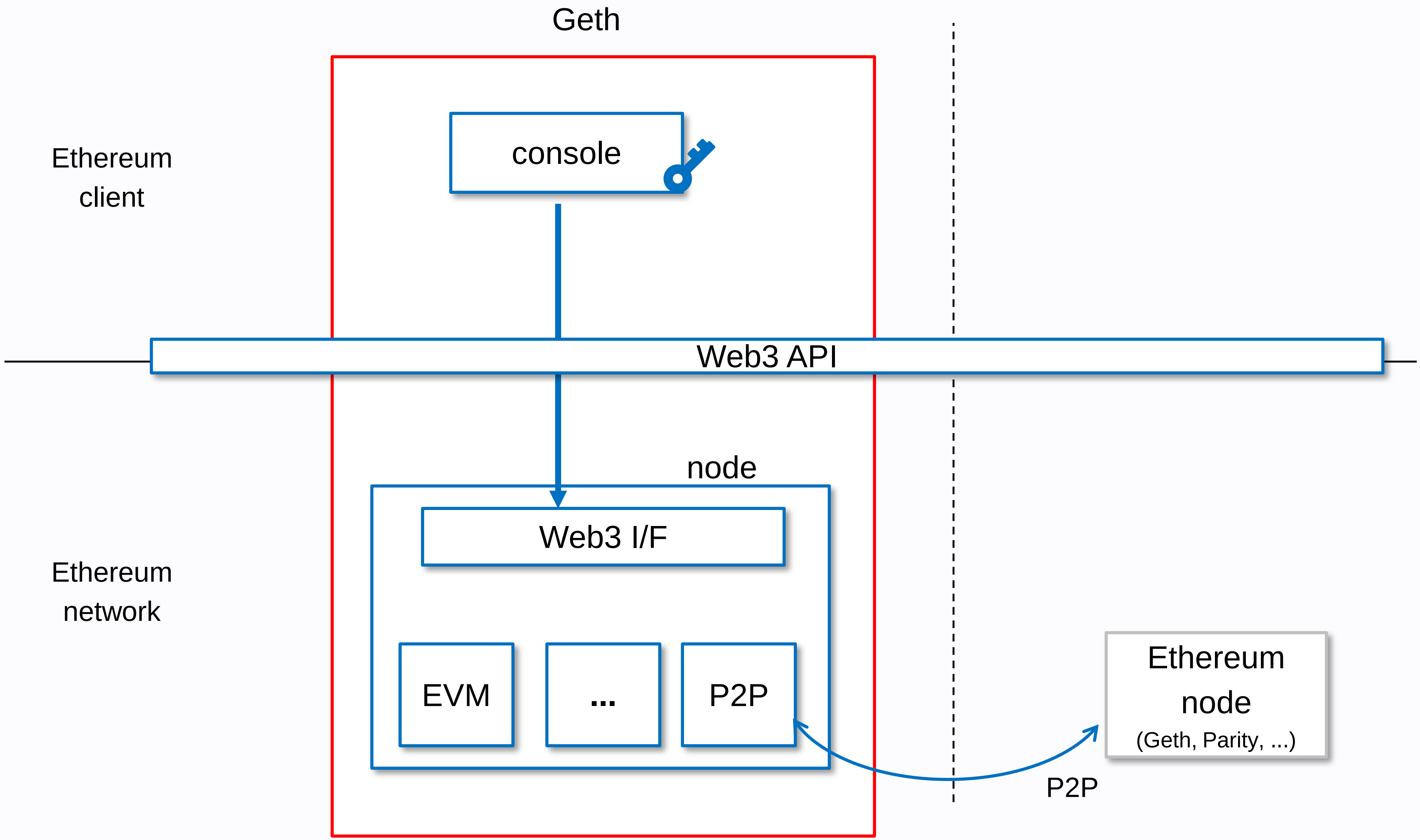
Ethereum clients access to Ethereum network via Web3 API.

References : [E8], [C1], [C3], [C4], [C5], [C6]

# Appendix B

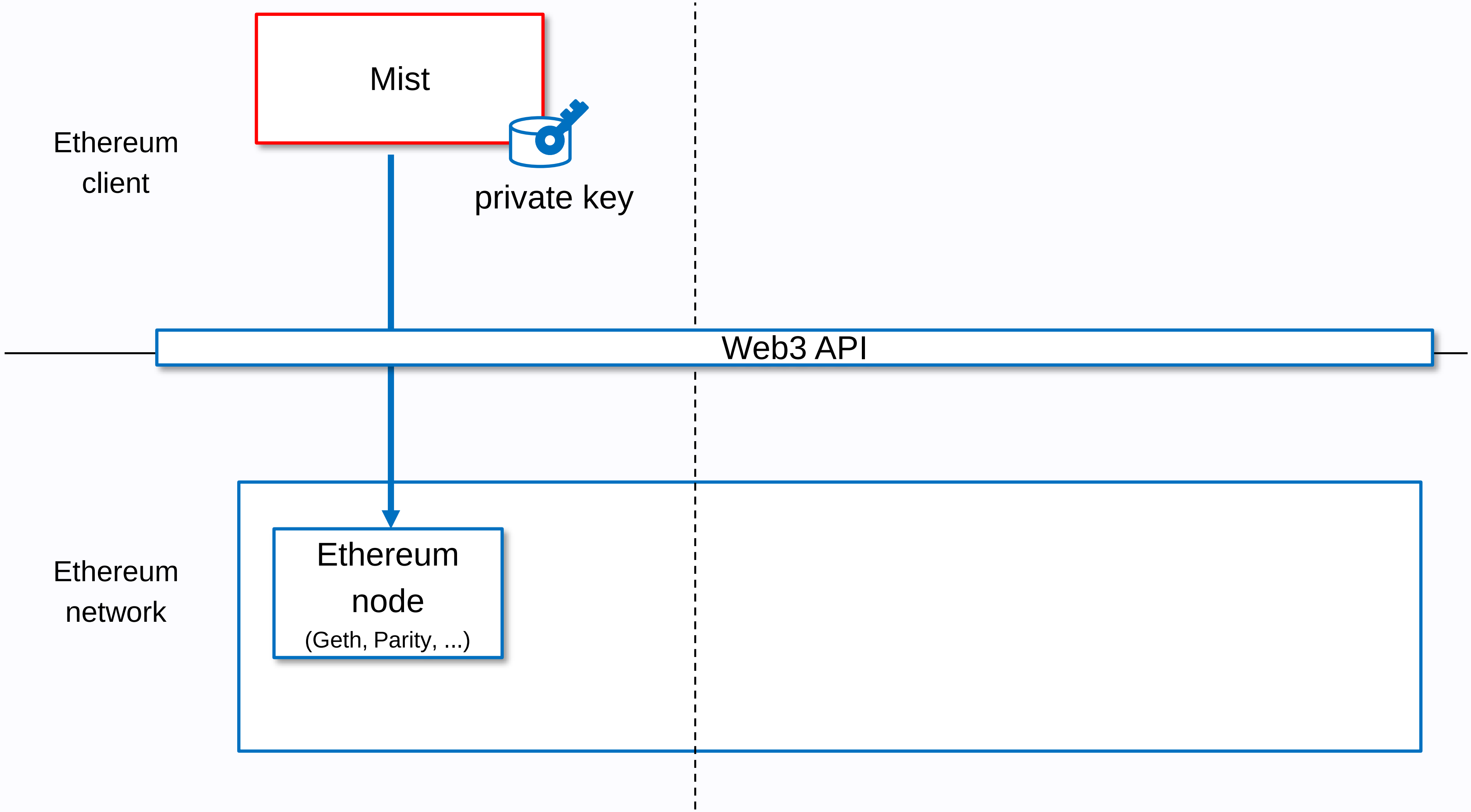
Geth, Mist, Solc, Remix, Truffle, ...

# Geth



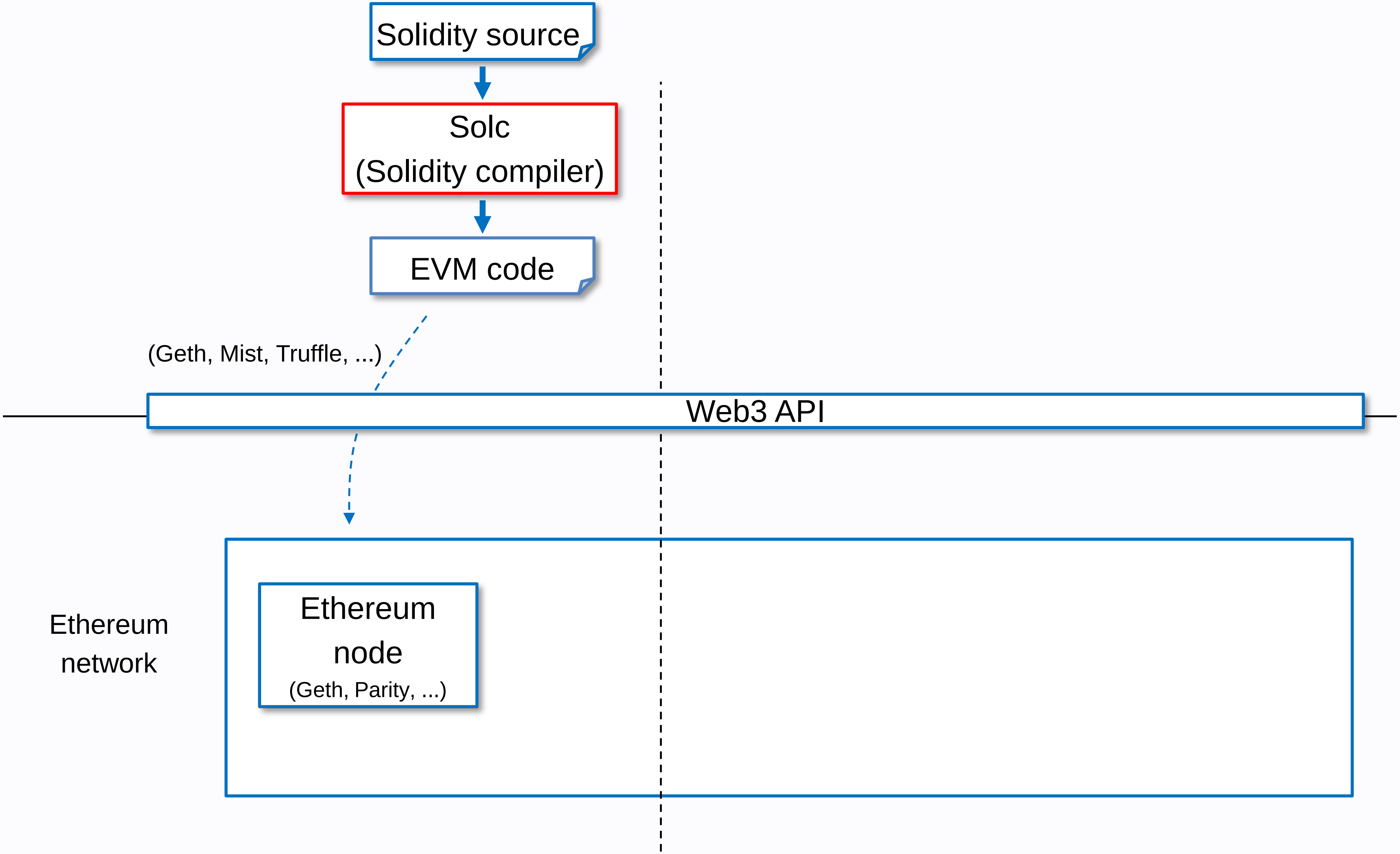
References : [C1]

# Mist



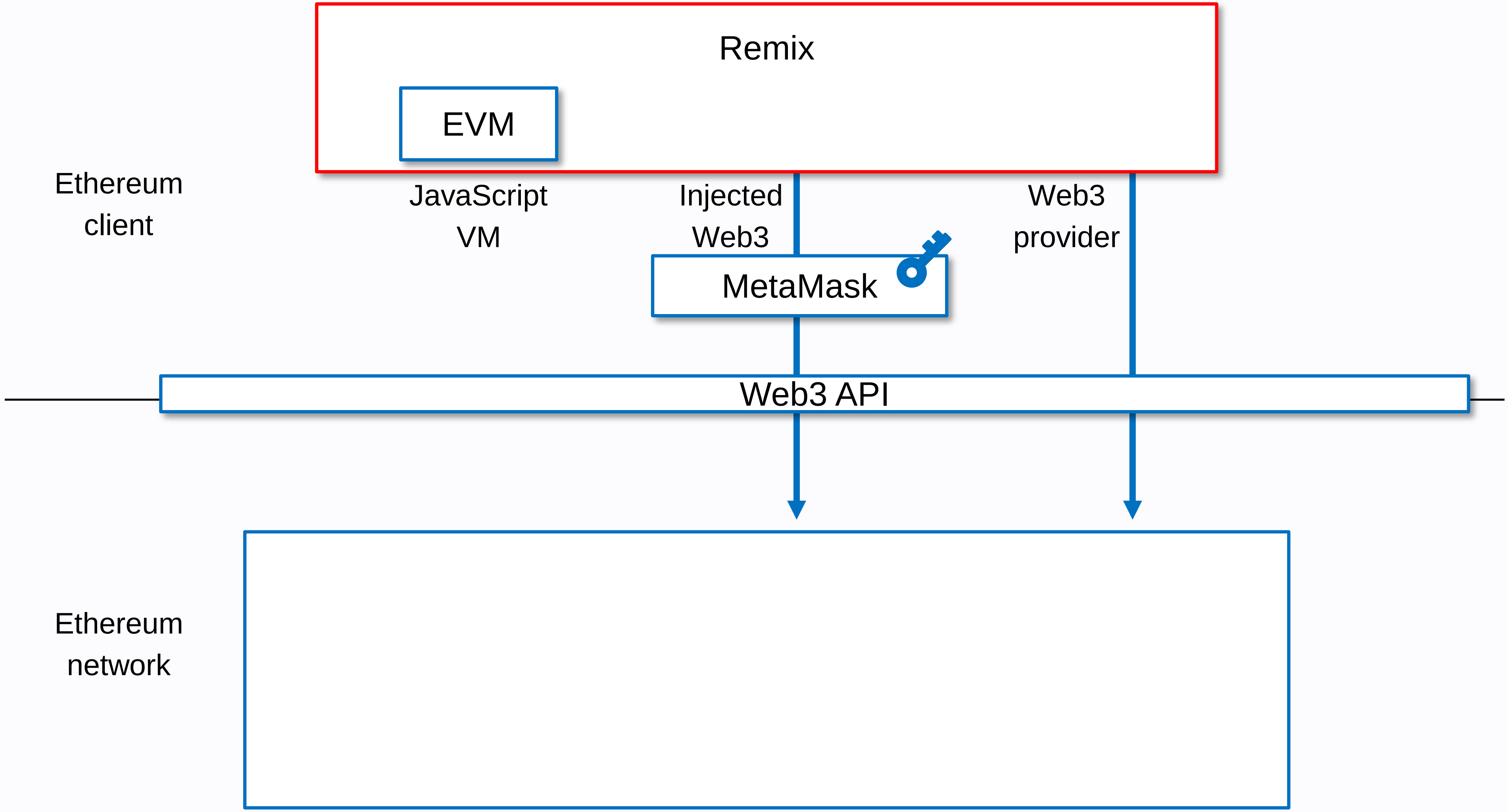
References : [C3]

# Solc



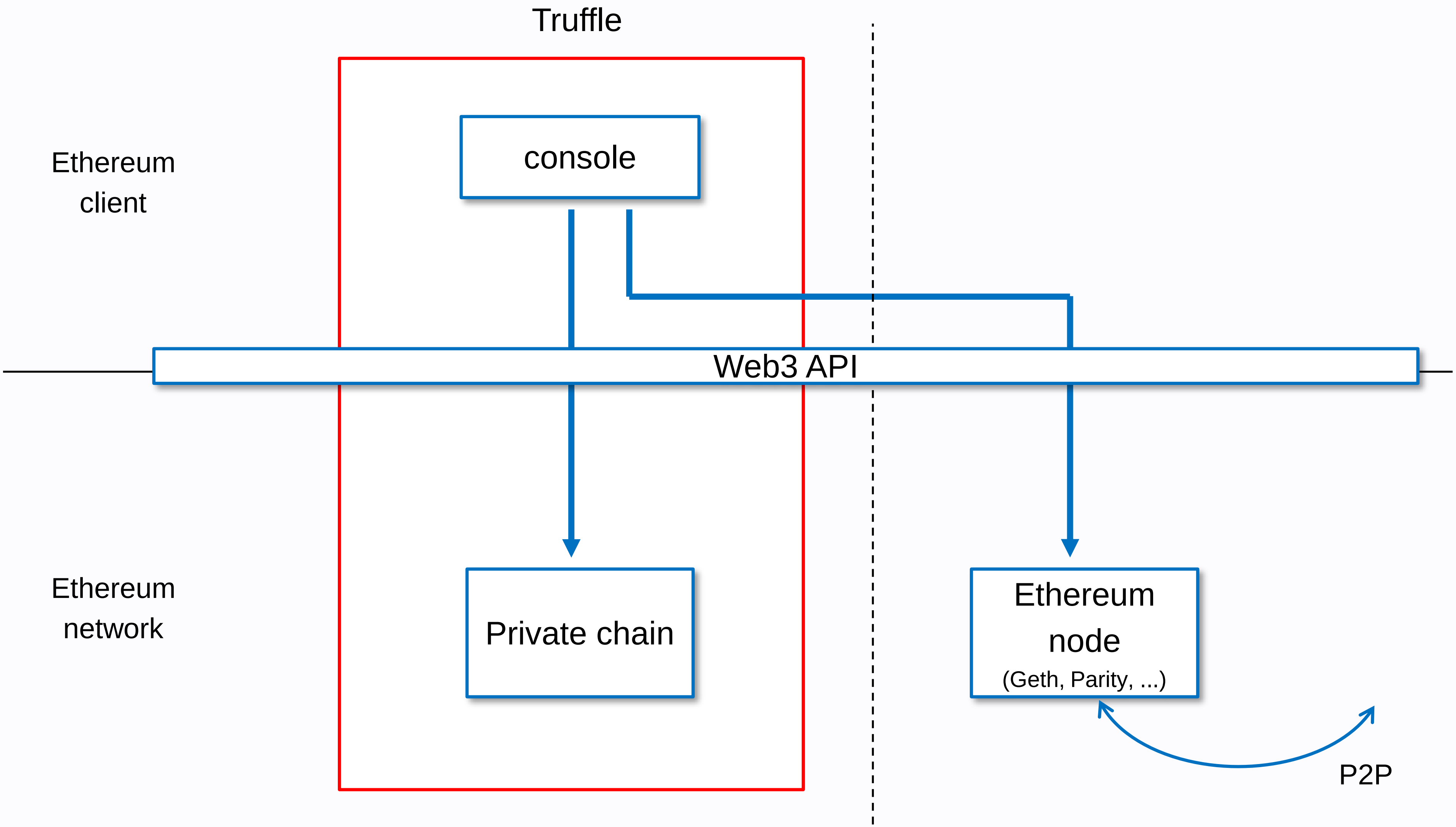
References : [C2]

# Remix



References : [C5]

# Truffle



References : [C6]

# References

# References

[E1] Ethereum Yellow Paper
ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
https://ethereum.github.io/yellowpaper/paper.pdf

[E2] Glossary
https://github.com/ethereum/wiki/wiki/Glossary

[E3] White Paper
A Next-Generation Smart Contract and Decentralized Application Platform
https://github.com/ethereum/wiki/wiki/White-Paper

[E4] Design Rationale
https://github.com/ethereum/wiki/wiki/Design-Rationale

[E5] Ethereum Development Tutorial
https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial

[E6] Ethereum Introduction
https://github.com/ethereum/wiki/wiki/Ethereum-introduction

[E7] Solidity Documentation
https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf
https://solidity.readthedocs.io/en/develop/

[E8] Web3 JavaScript app API for 0.2x.x
https://github.com/ethereum/wiki/wiki/JavaScript-API

# References

[W1] Awesome Ethereum Virtual Machine
https://github.com/pirapira/awesome-ethereum-virtual-machine

[W2] Diving Into The Ethereum VM
https://blog.qtum.org/diving-into-the-ethereum-vm-6e8d5d2f3c30

[W3] Stack Exchange: Ethereum block architecture
https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture/6413

[W4] Porosity
https://www.comae.io/reports/dc25-msuiche-Porosity-Decompiling-Ethereum-Smart-Contracts.pdf

# References

[C1] Go Ethereum
https://github.com/ethereum/go-ethereum

[C2] Solc (Solidity compiler)
https://github.com/ethereum/solidity

[C3] Mist (Ethereum Wallet)
https://github.com/ethereum/mist

[C4] MetaMask
https://github.com/MetaMask/metamask-extension

[C5] Remix
https://github.com/ethereum/browser-solidity

[C6] Truffle
https://github.com/trufflesuite/truffle