# Ethereum EVM illustrated

exploring some mental models and implementations

Takenobu T.

WIP

Rev. 0.00.0

NOTE

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

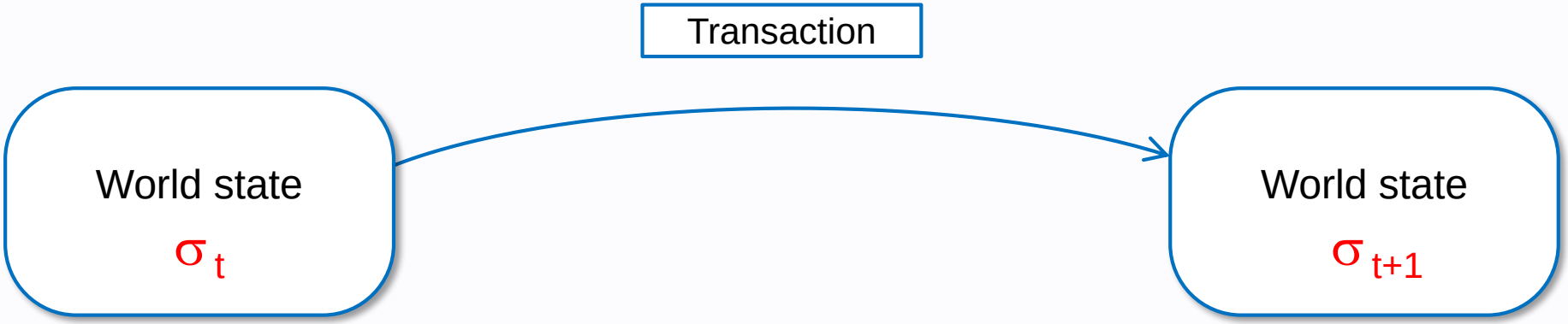
# Contents

# 1. Introduction

# 1. Introduction

## Blockchain

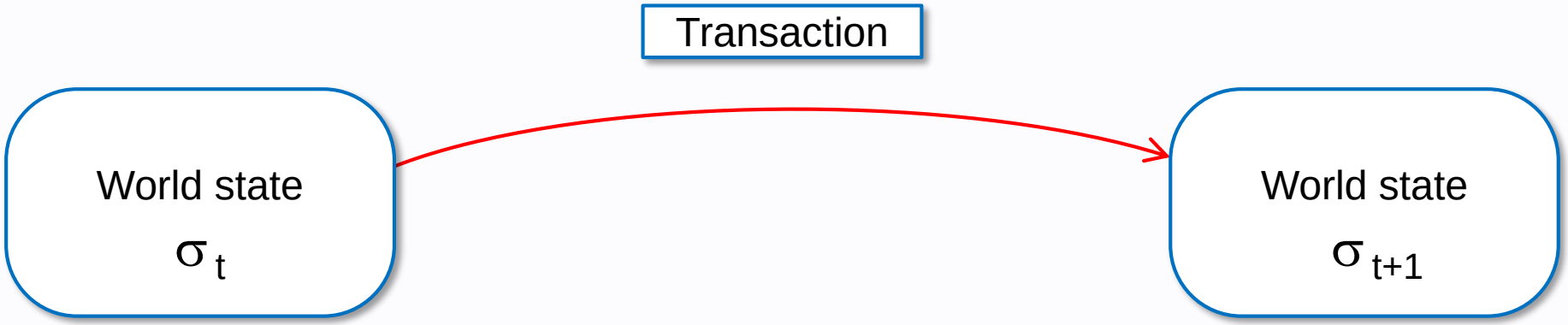
# A transaction-based state machine




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

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

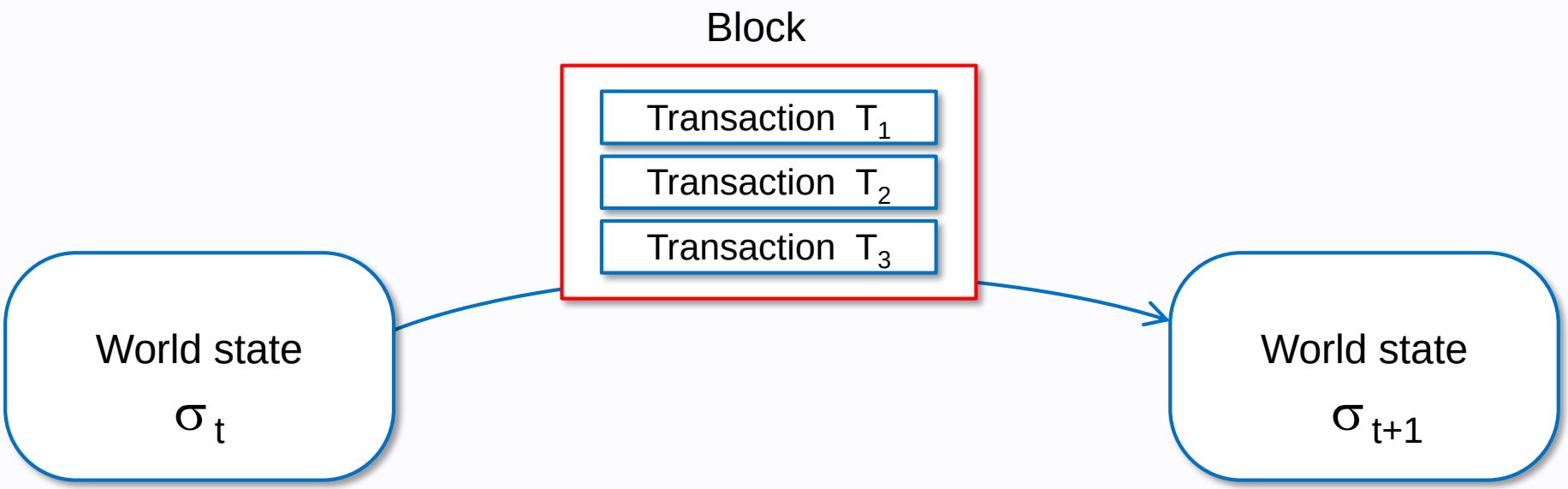
# A transaction-based state machine




A transaction represents a valid arc between two states.

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

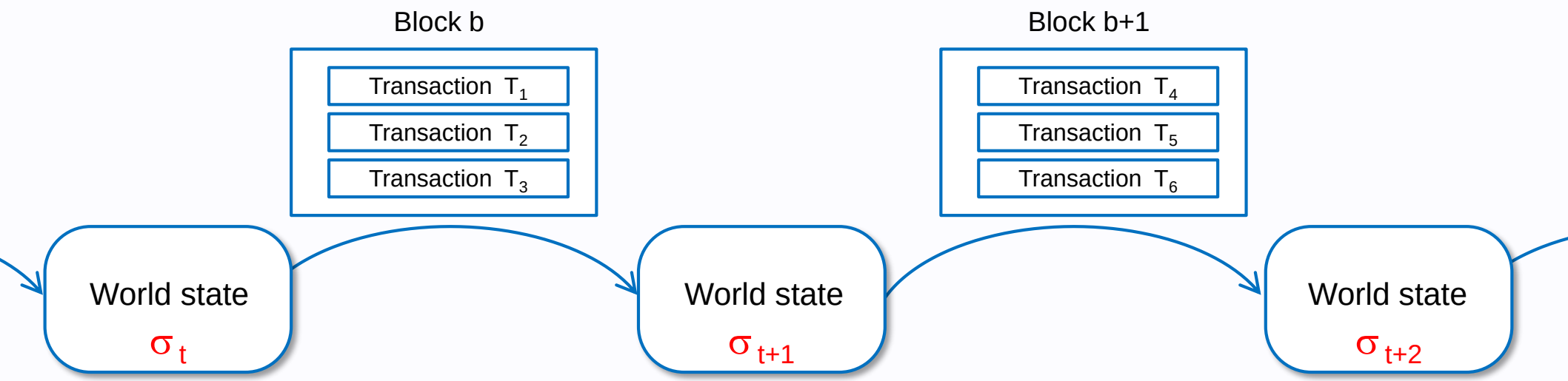
# Block and transactions




Transactions are collated into blocks.
A block is a package of data.

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

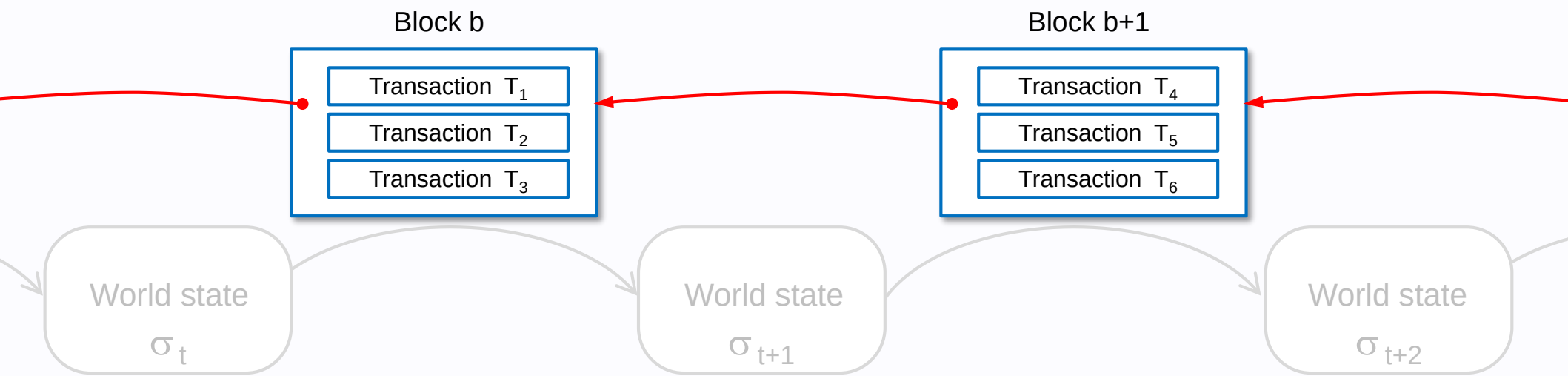
# Chain of states




As states view,
Ethereum can be viewed as a chain of states.

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

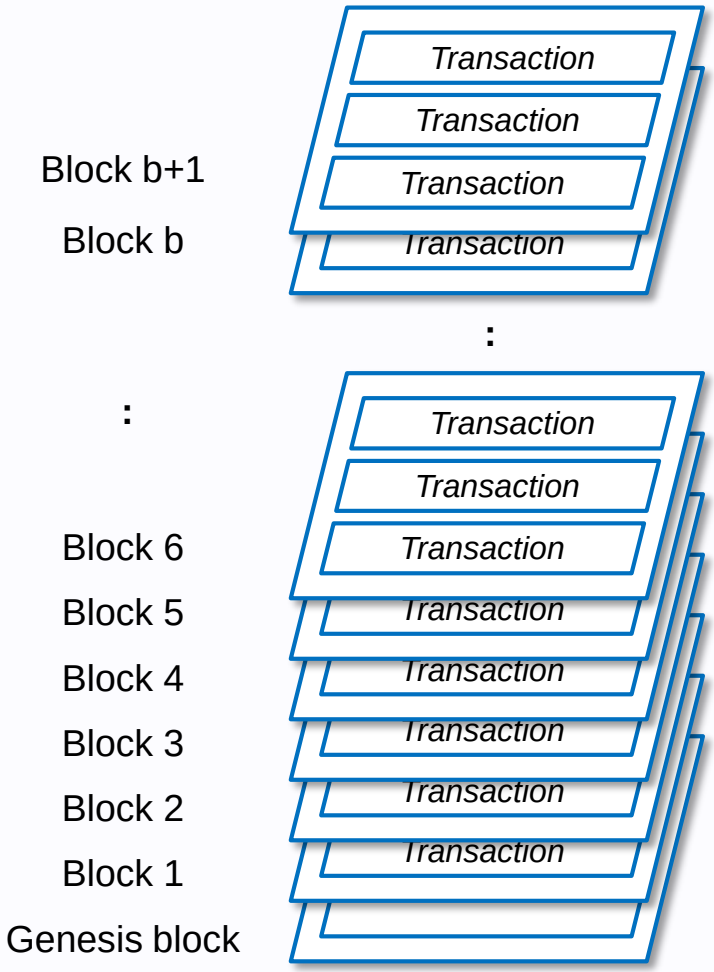
# Chain of blocks: Blockchain




As implementation view,
Ethereum can be also viewed as a chain of blocks, thus `BLOCKCHAIN`.

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

# Stack of transactions : Ledger




As ledger view,

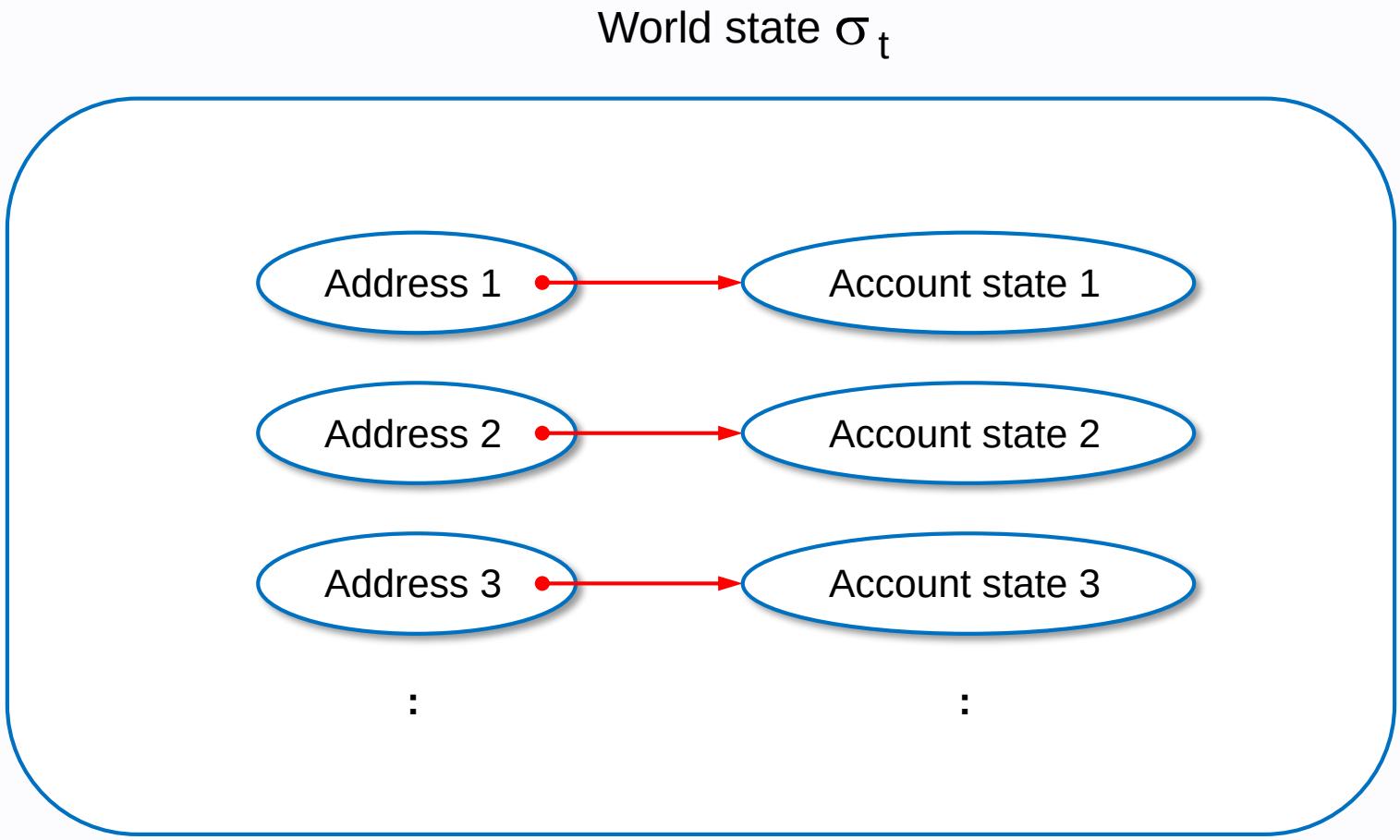
Ethereum can be also viewed as a transaction stack, thus `LEDGER`.

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

# 1. Introduction

World state

# World state




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

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

# Several views of world state

Mapping view

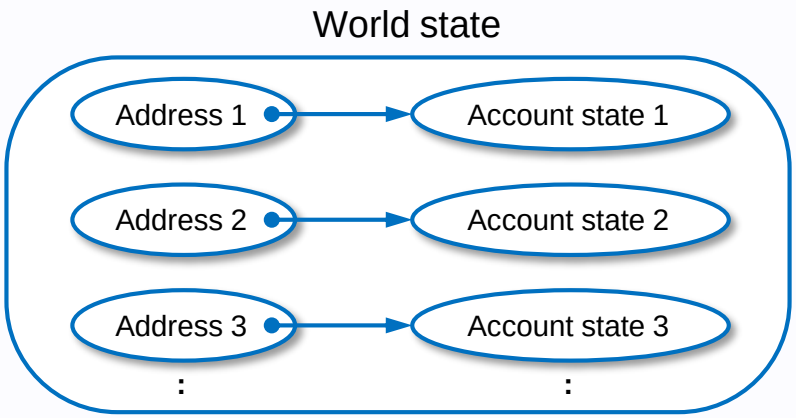
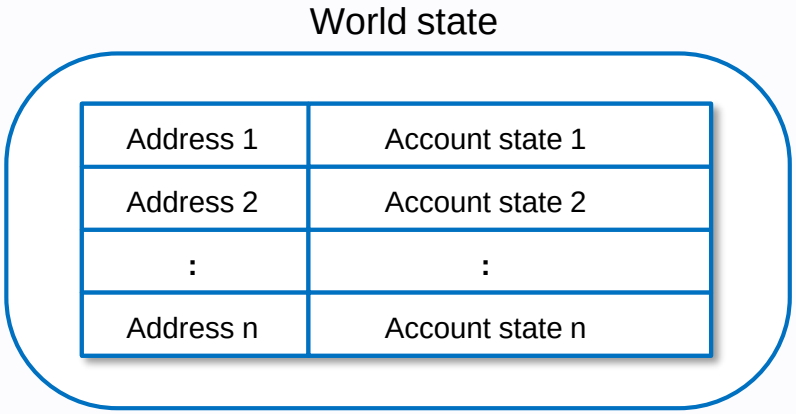
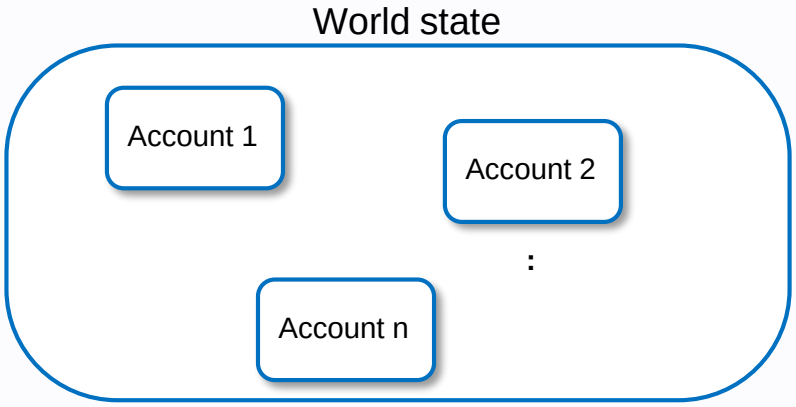



Table view




Object view

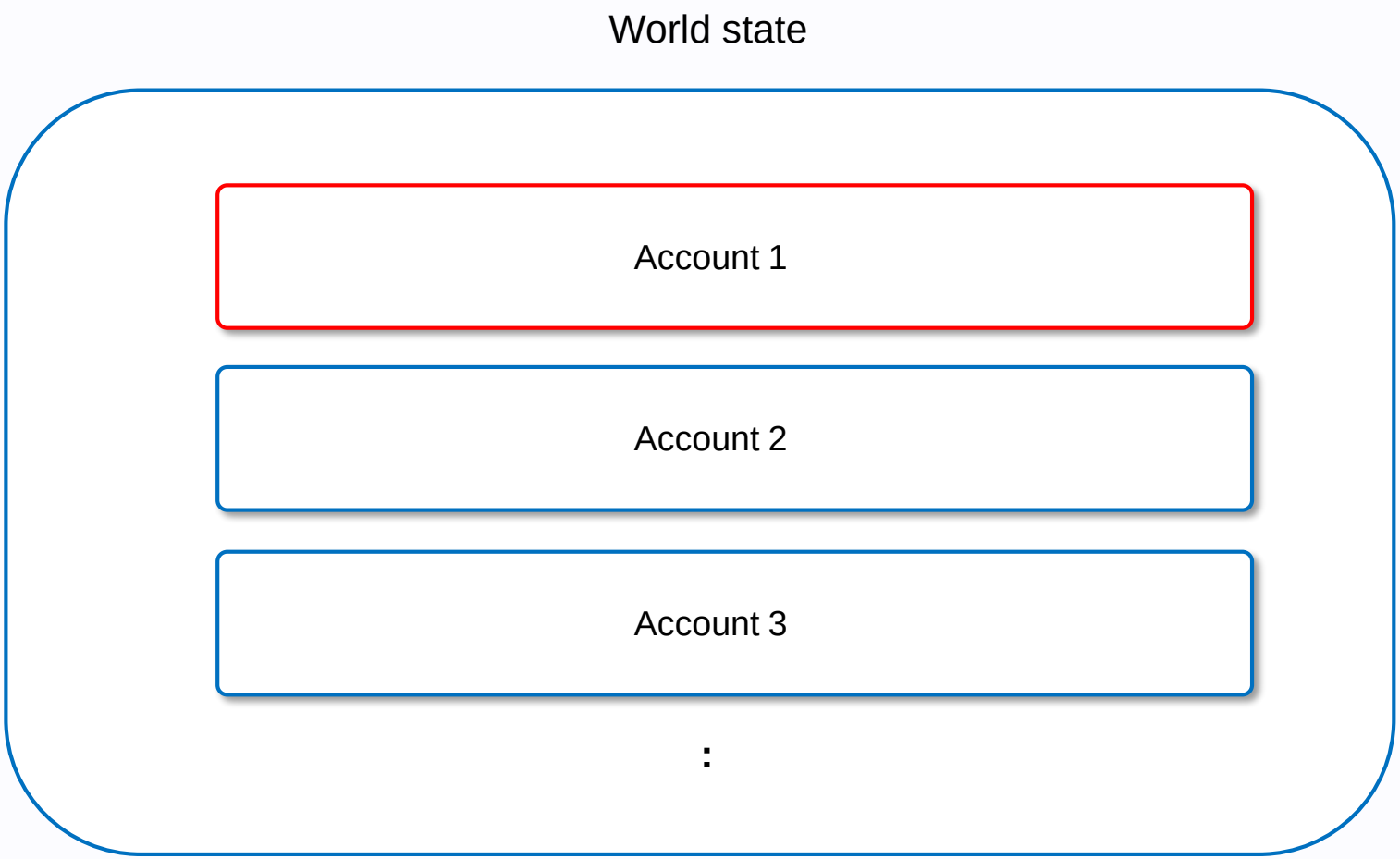



References : [E1] Ch.4

# 1. Introduction

## Account

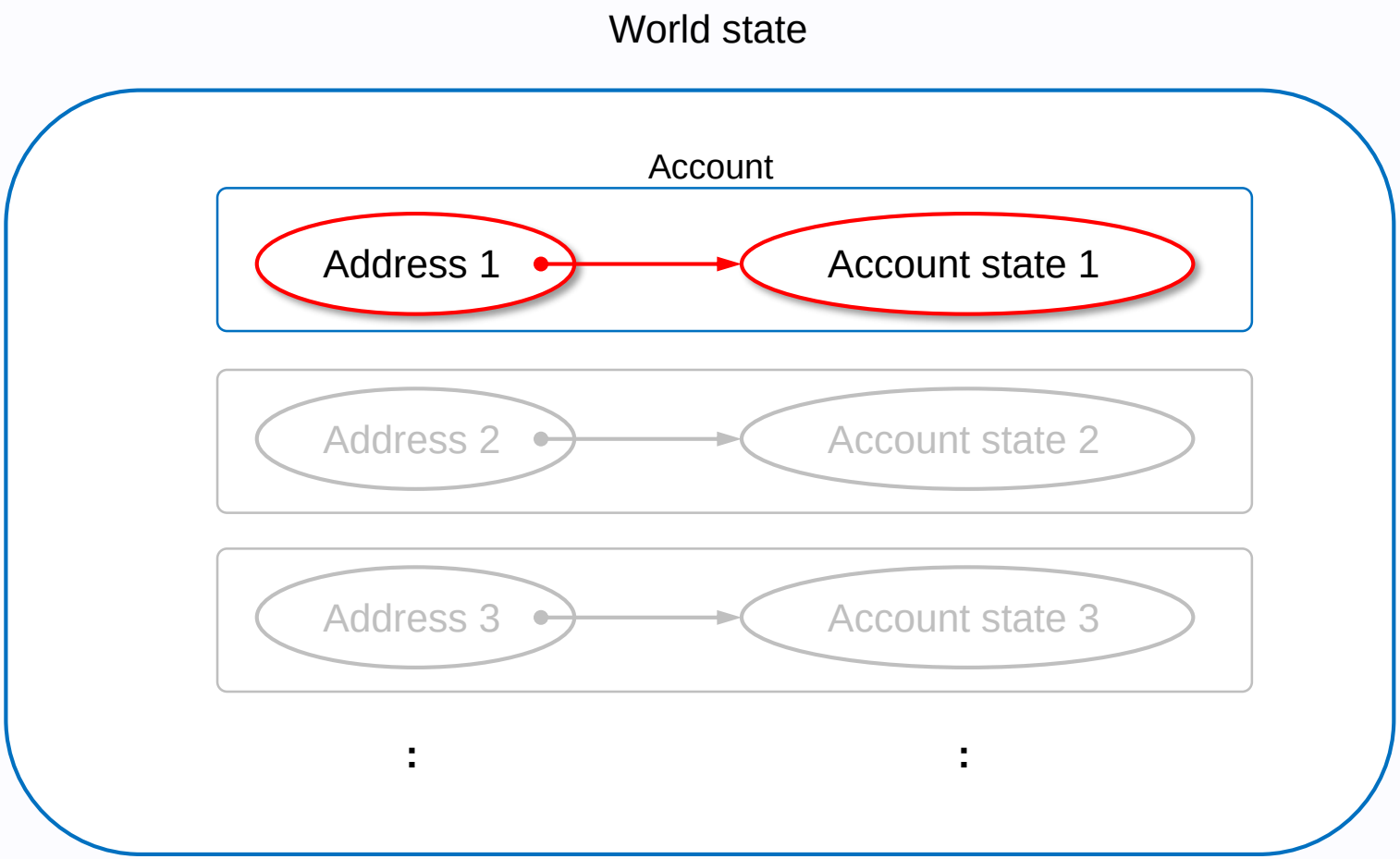
# Account




An account is an object in the world state.

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

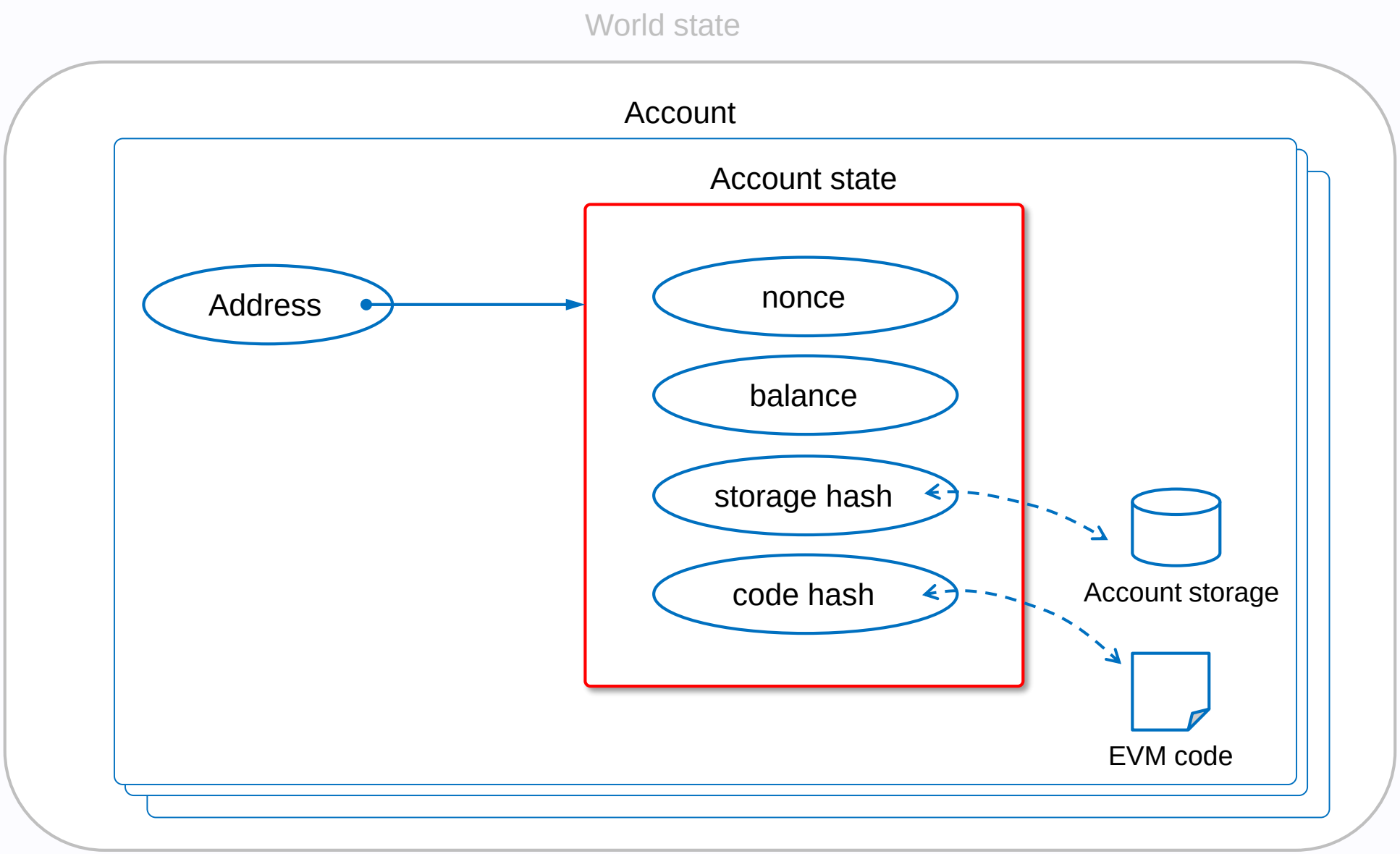
# Account




An account is a mapping between address and account state.

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

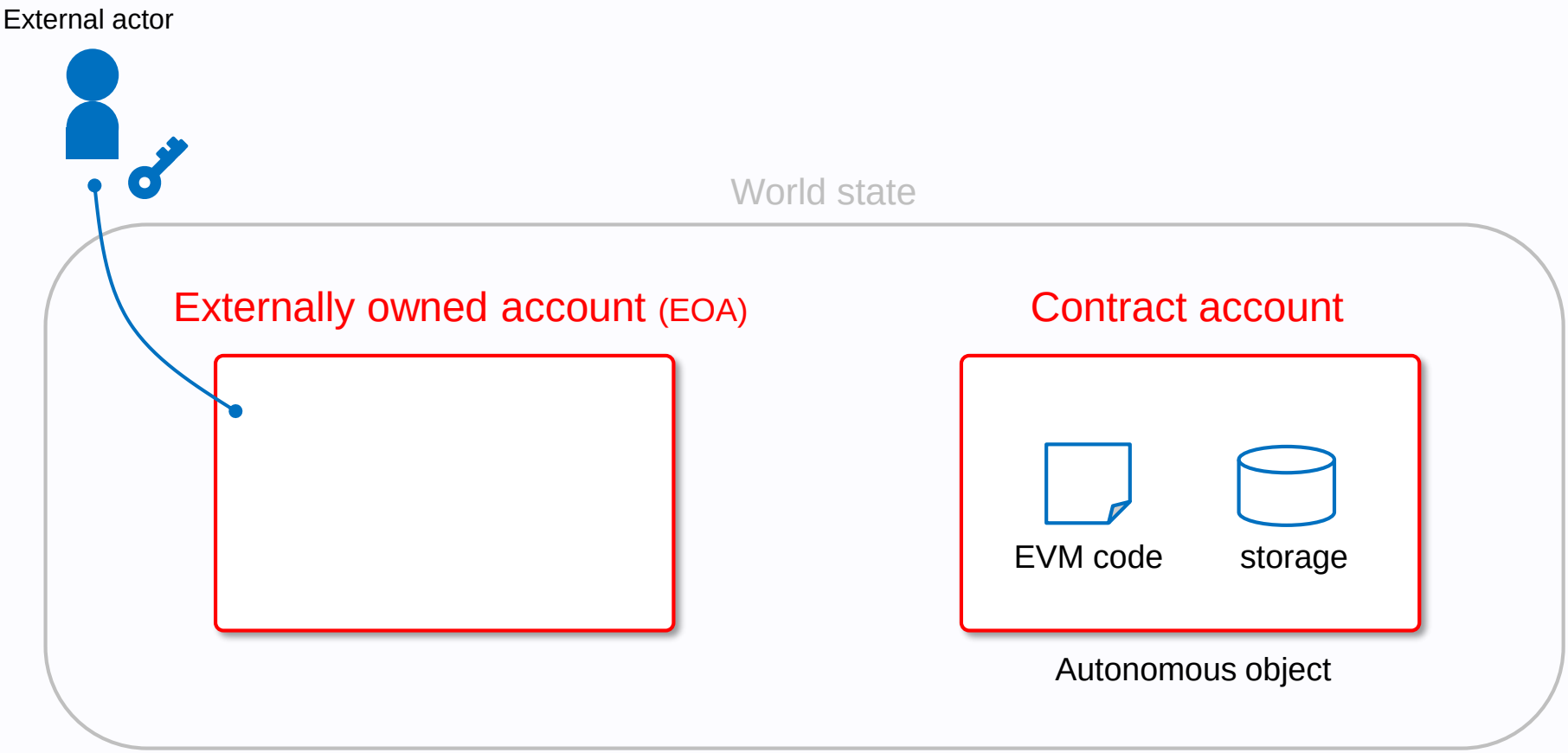
# Account state




An account state could contain EVM code and storage.

References : [E1] Ch.4

# Two practical types of account

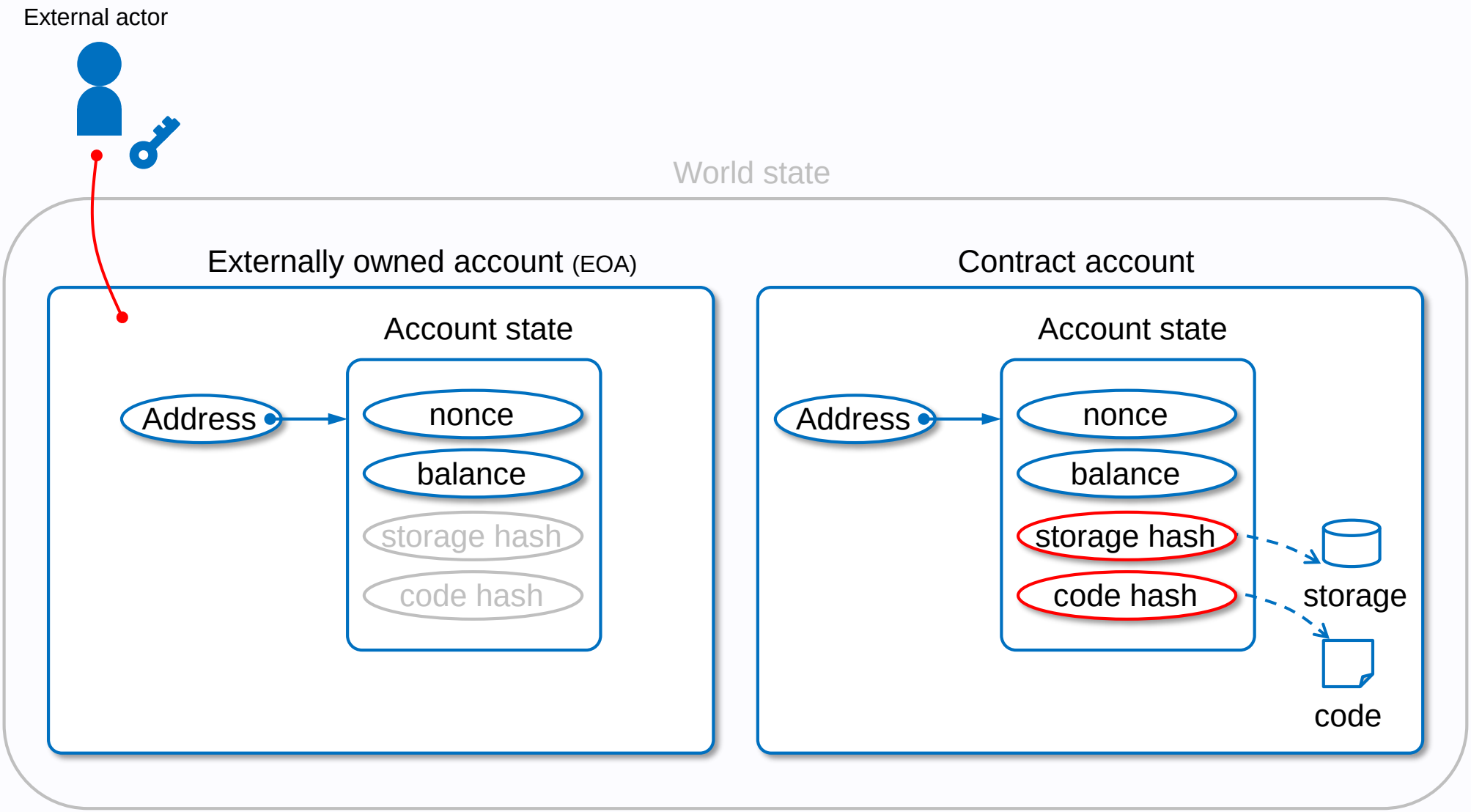



EOA is controlled by a private key.

Contract account contains EVM code.

References : [E1] Ch.4

# Two practical types of account



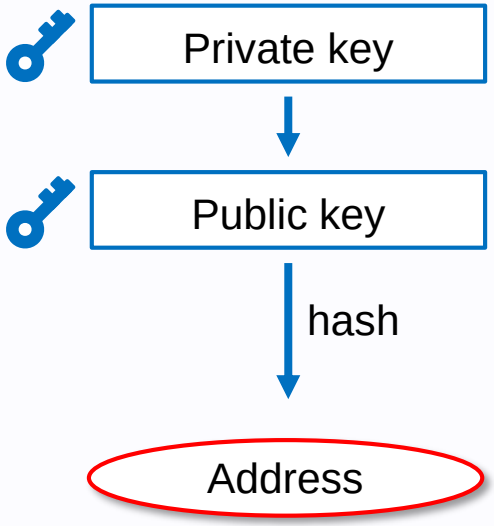

EOA is controlled by a private key.
EOA cannot contain EVM code.

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

References : [E1] Ch.4

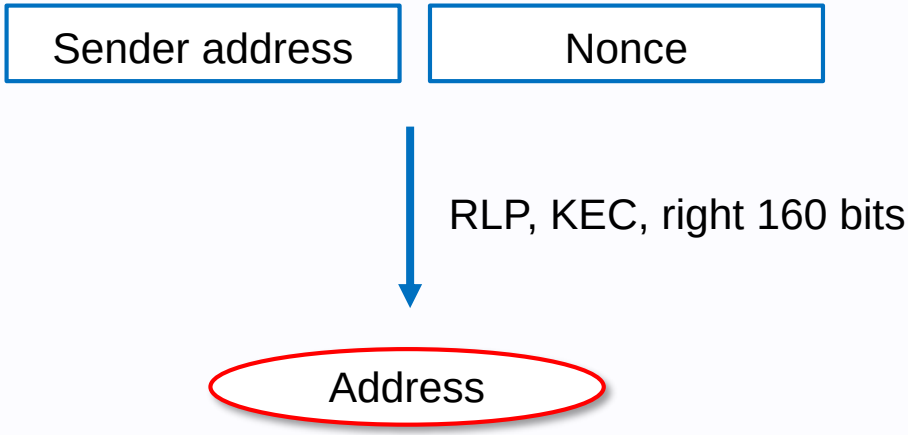
# Address of account

Externally owned account (EOA)




160 bits

Contract account




160 bits

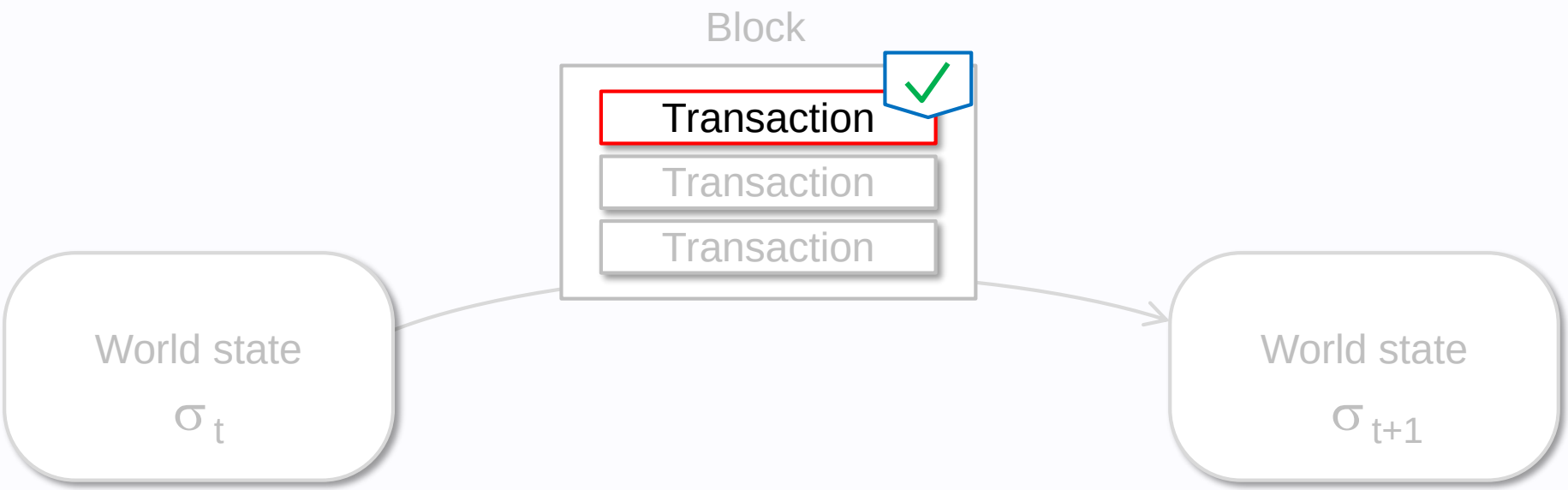
A 160-bit code used for identifying accounts.

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

# 1. Introduction

## Transaction

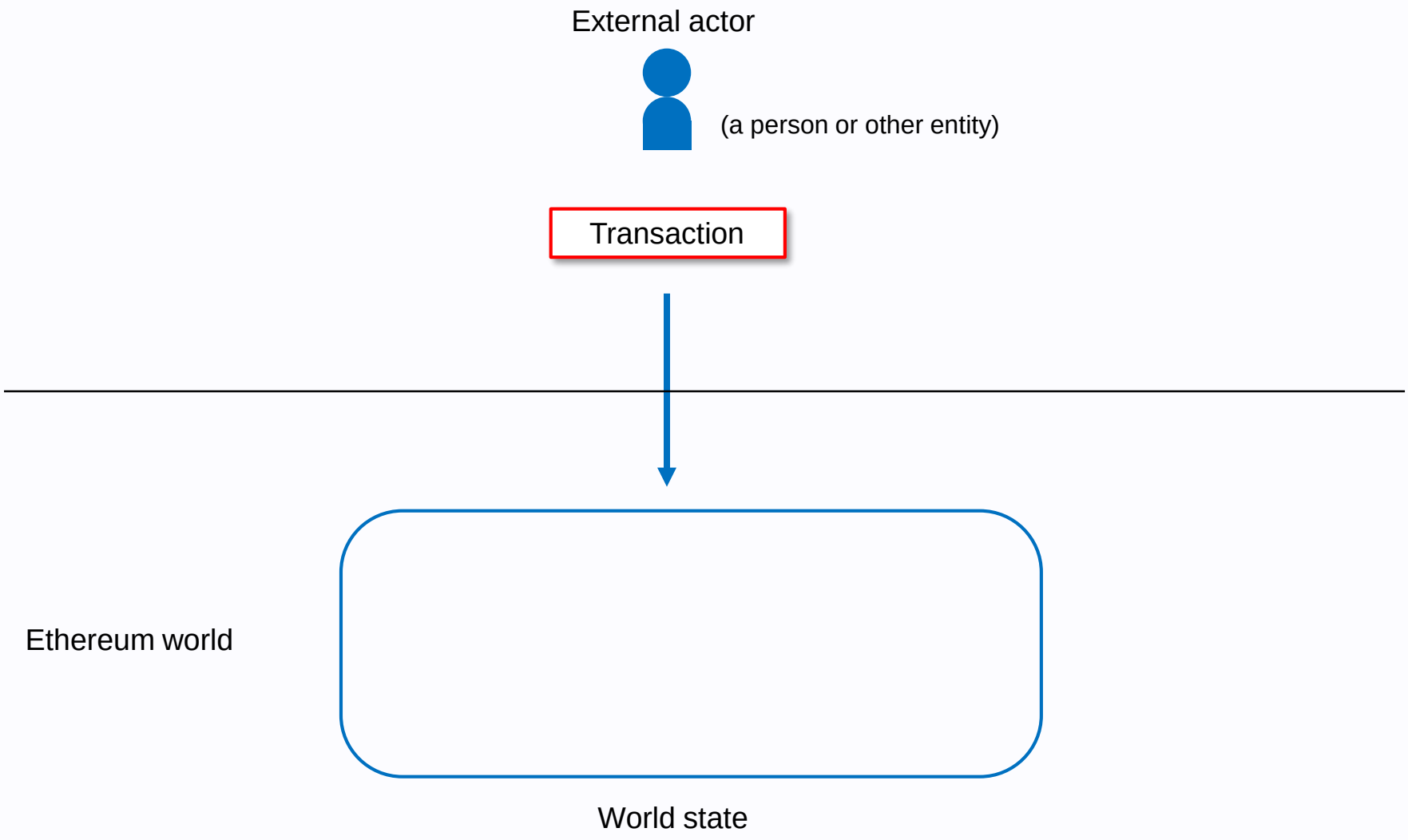
# A transaction




A transaction is a single cryptographically-signed instruction.

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

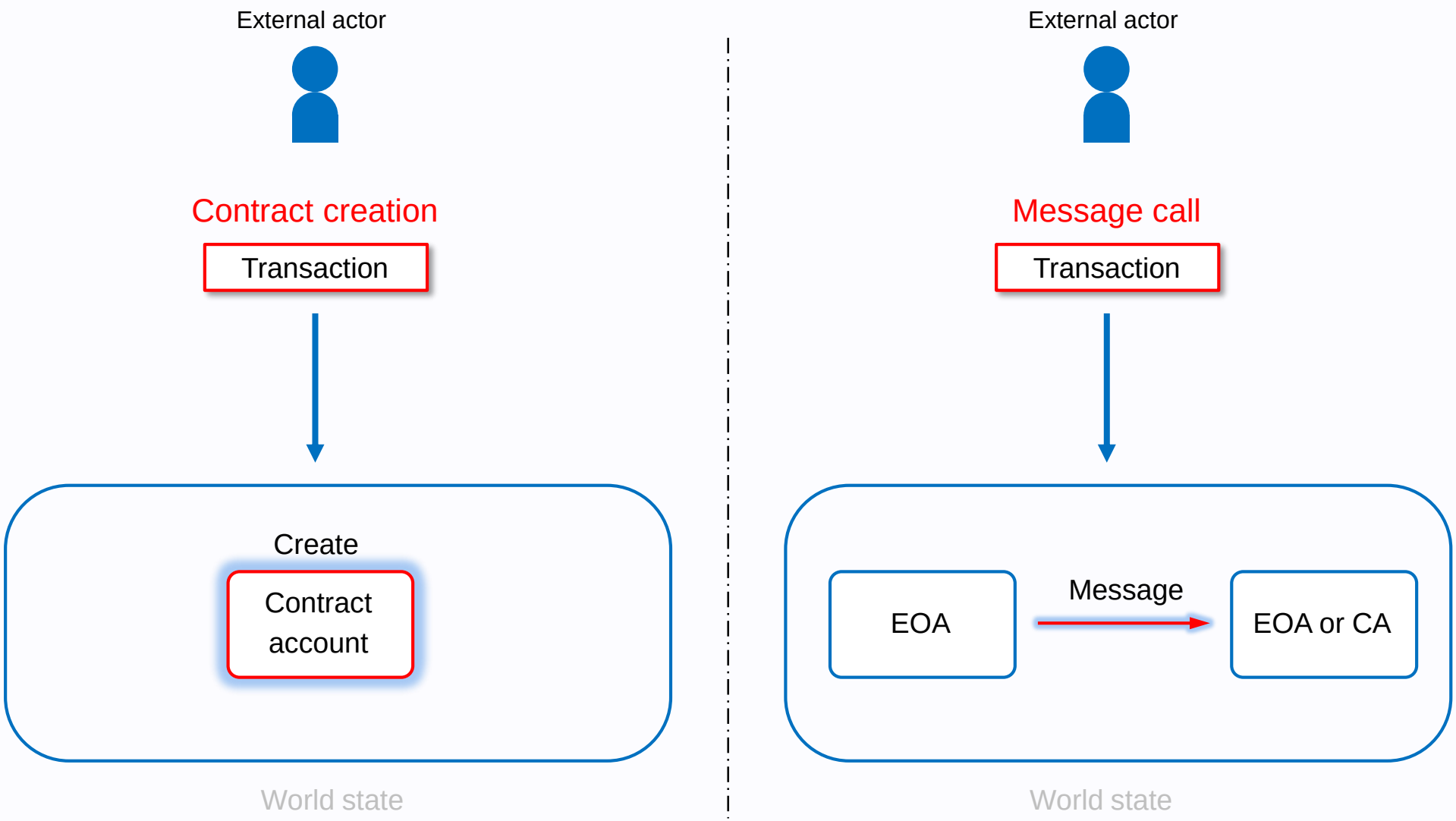
# A transaction to world state




A transaction is submitted by external actor.

References : [E1] Appendix A

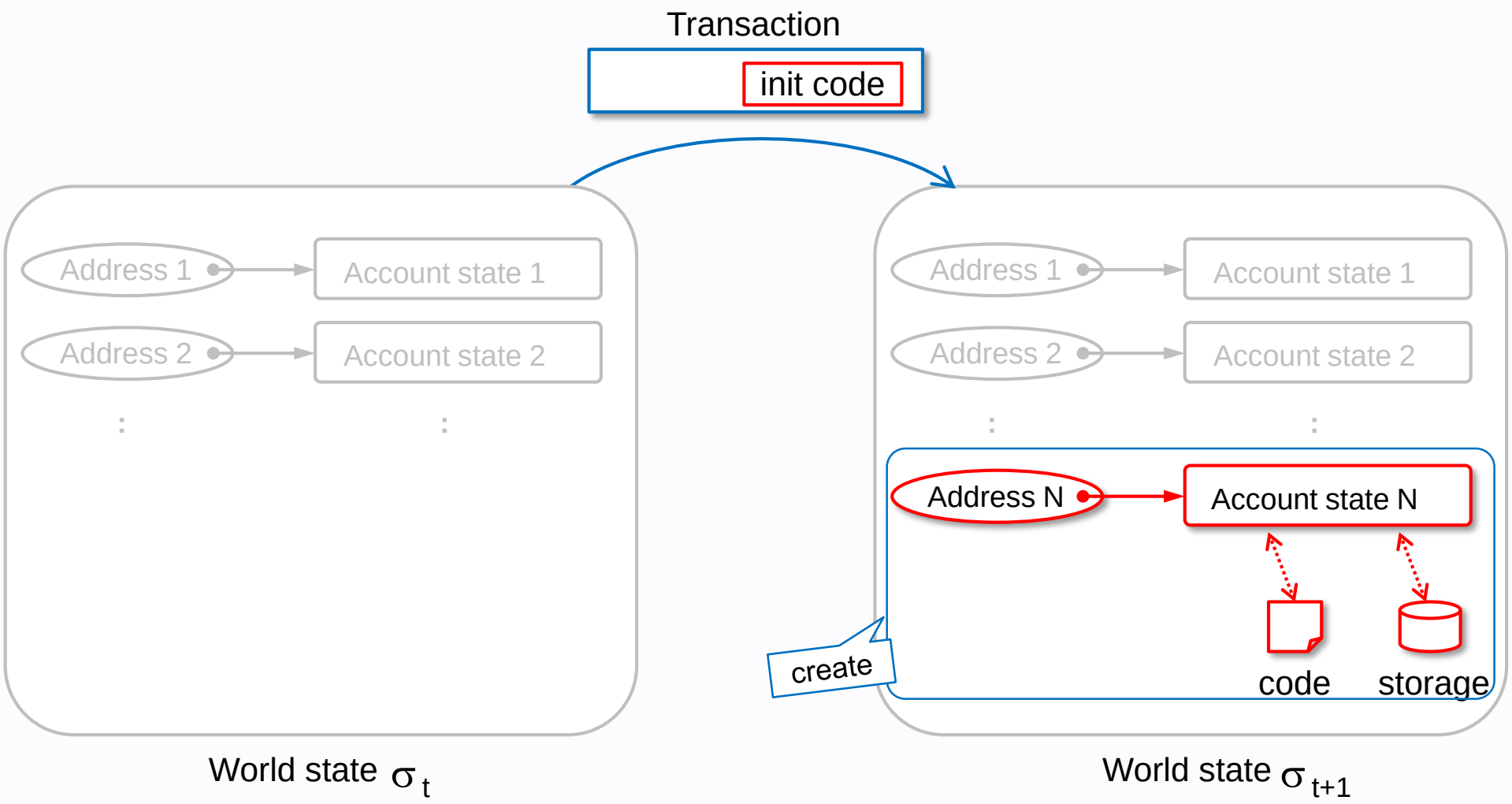
# Two practical types of transaction




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

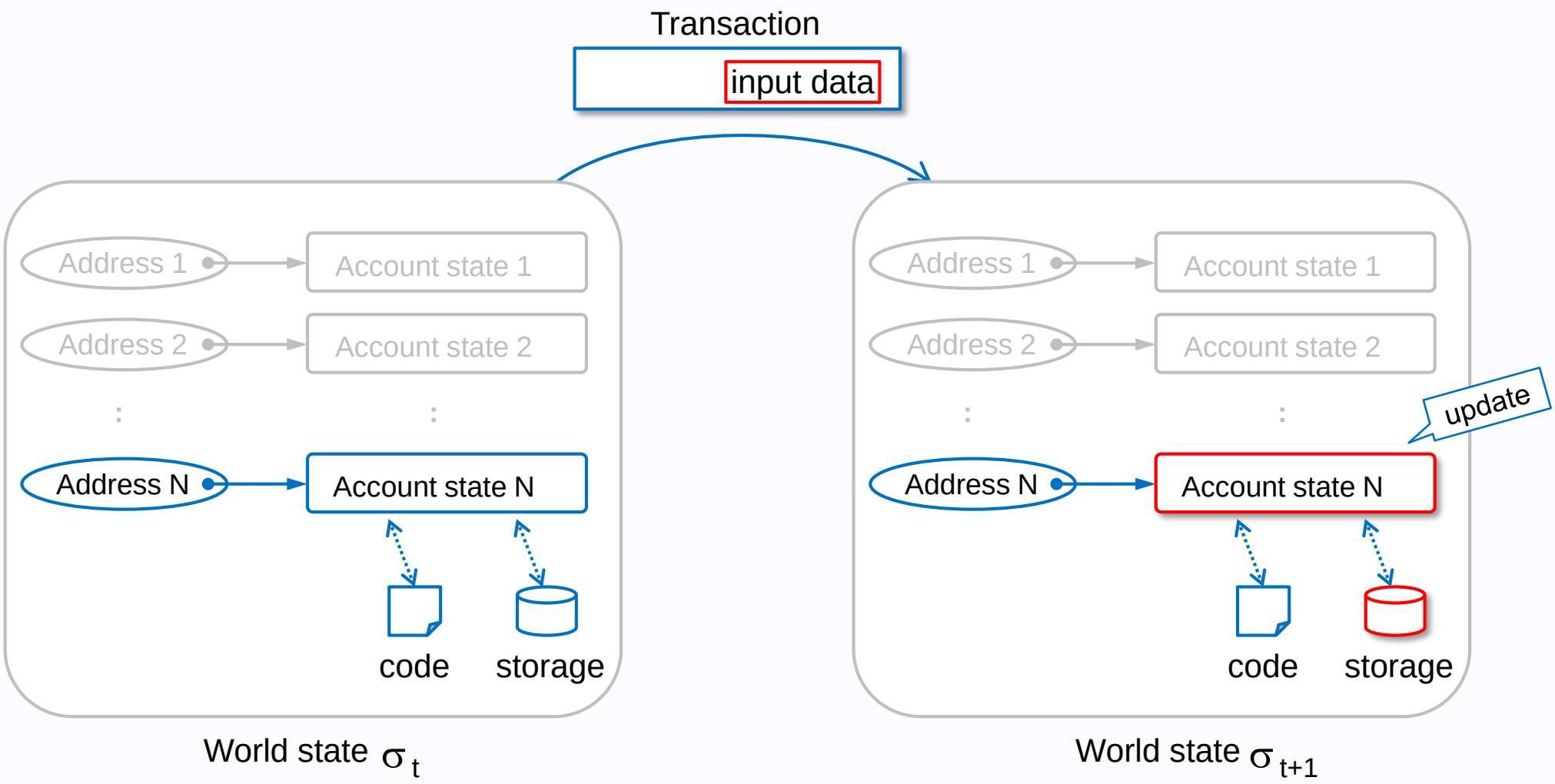
References : [E1] Ch.4

# Contract creation



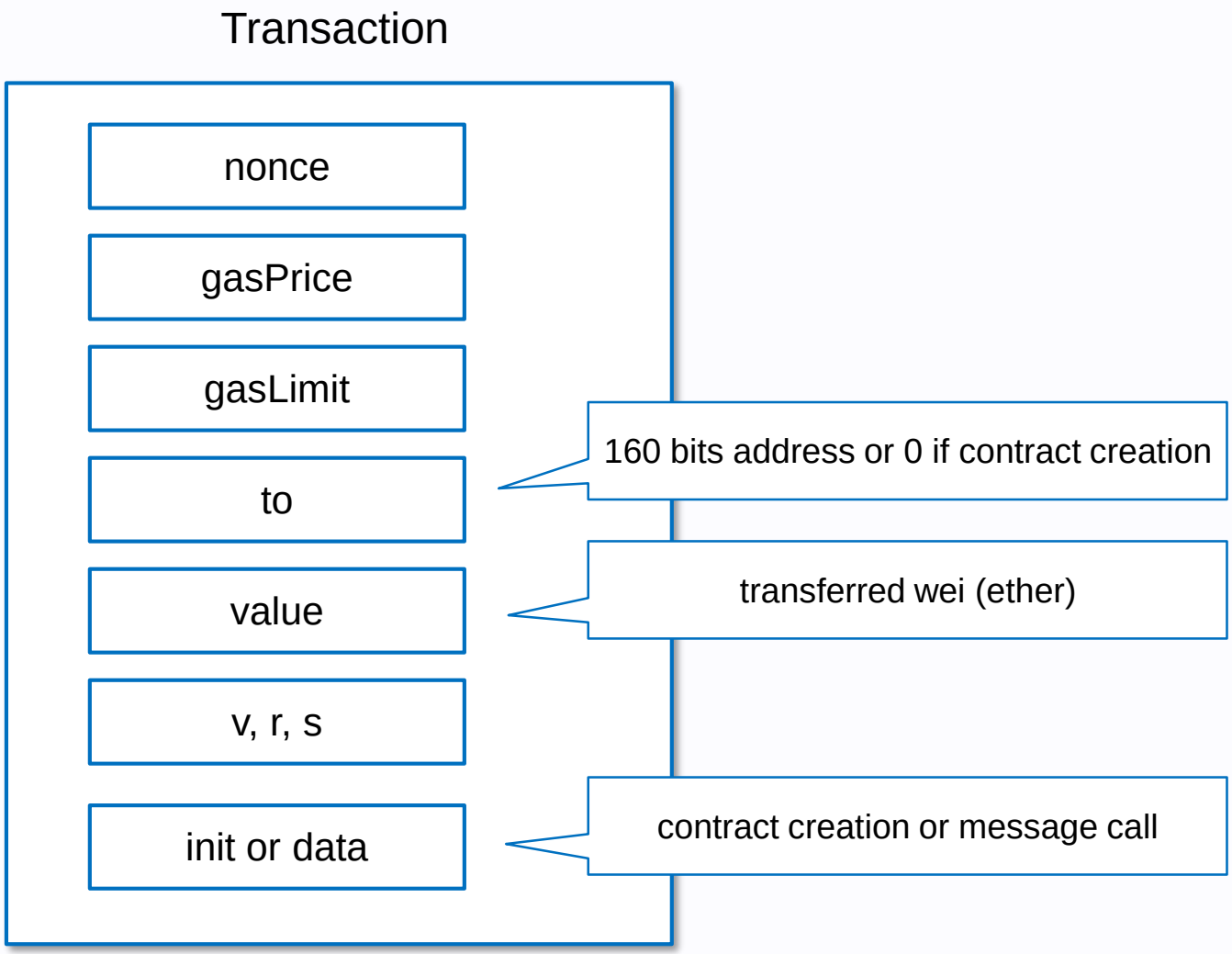

References : [E1] Ch.7

# Message call




References : [E1] Ch.7

# Field of a transaction

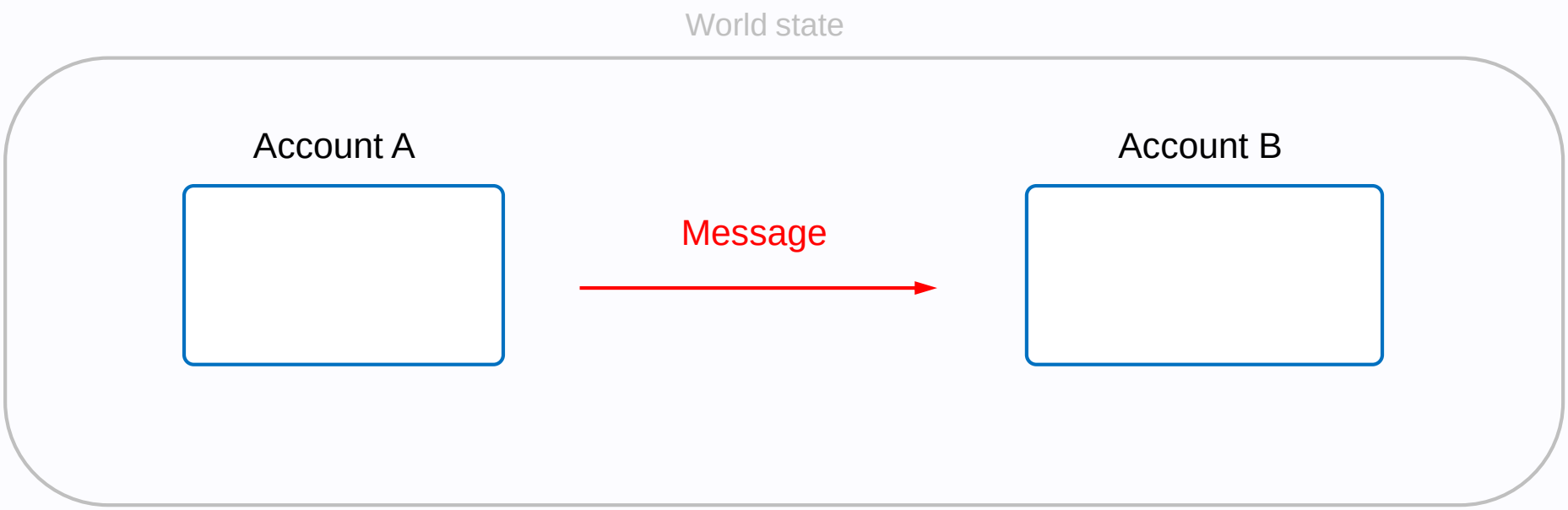



References : [E1] Ch.4

# 1. Introduction

Message

# Message

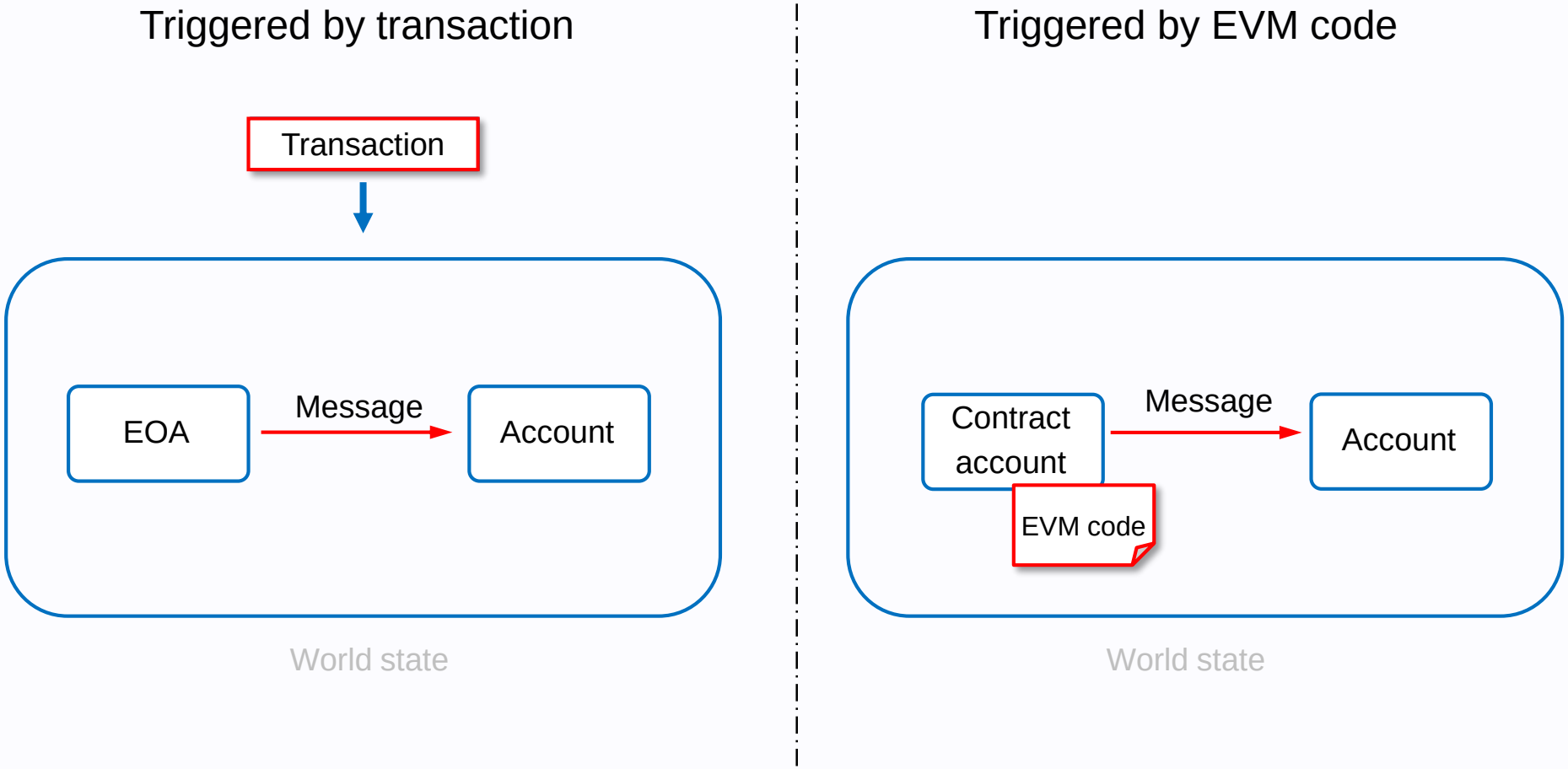



Message is passed between two Accounts.

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

References : [E1] Ch.8, Appendix A

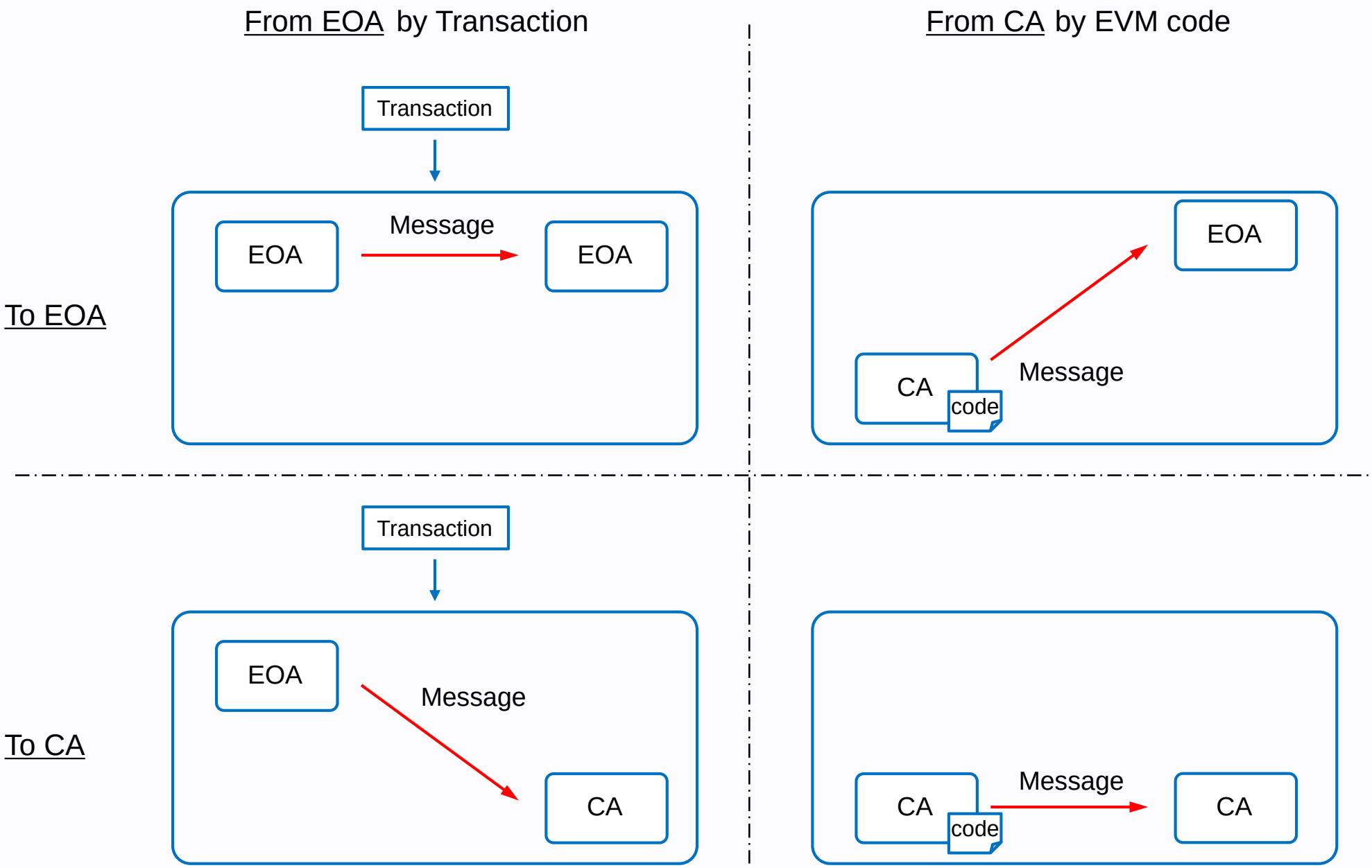
# Message




Every transaction triggers an associated message.
EVM can also send a message.

References : [E1] Ch.8

# Four cases of message

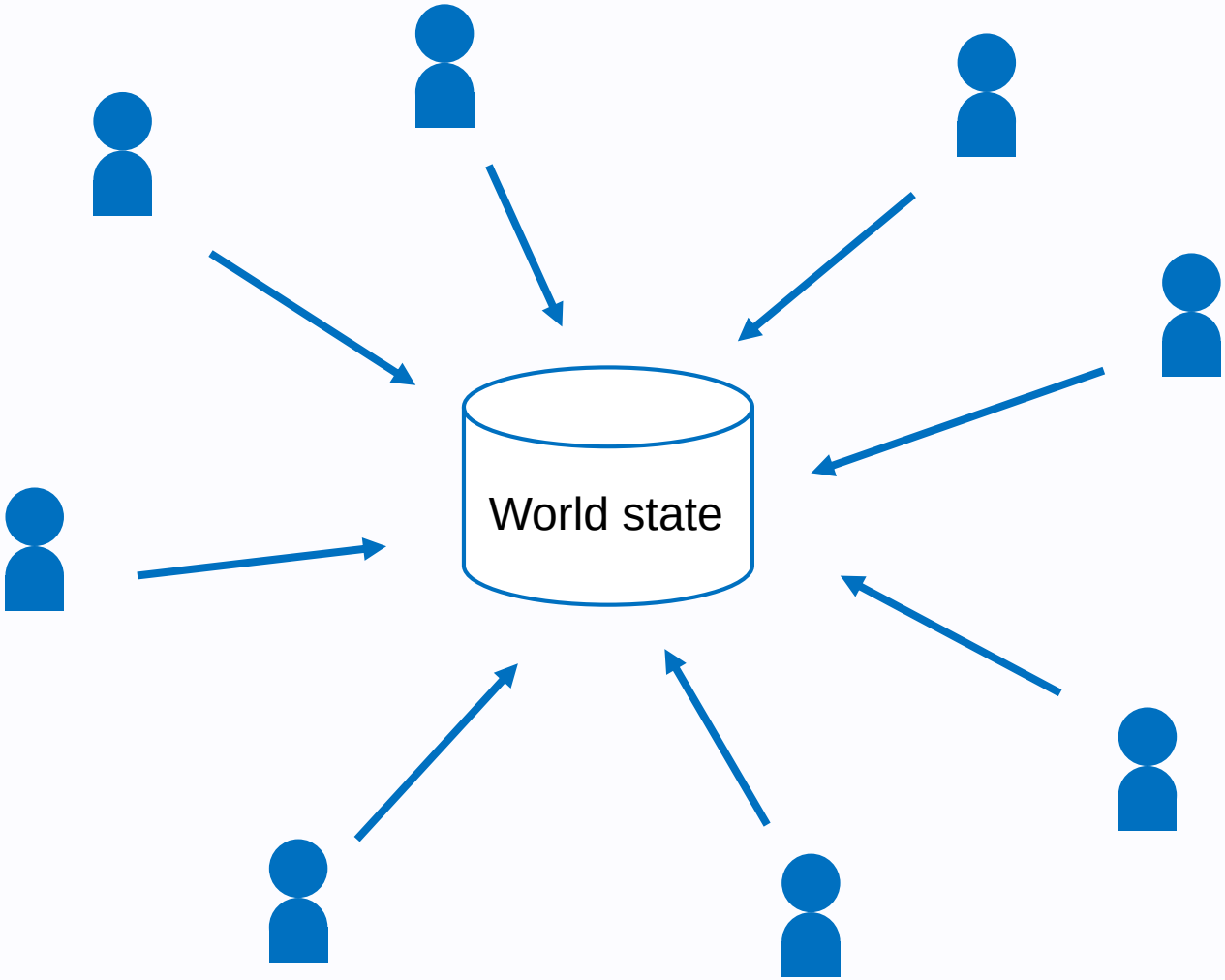



References : [E1] Ch.8

# 1. Introduction

## Decentralised database

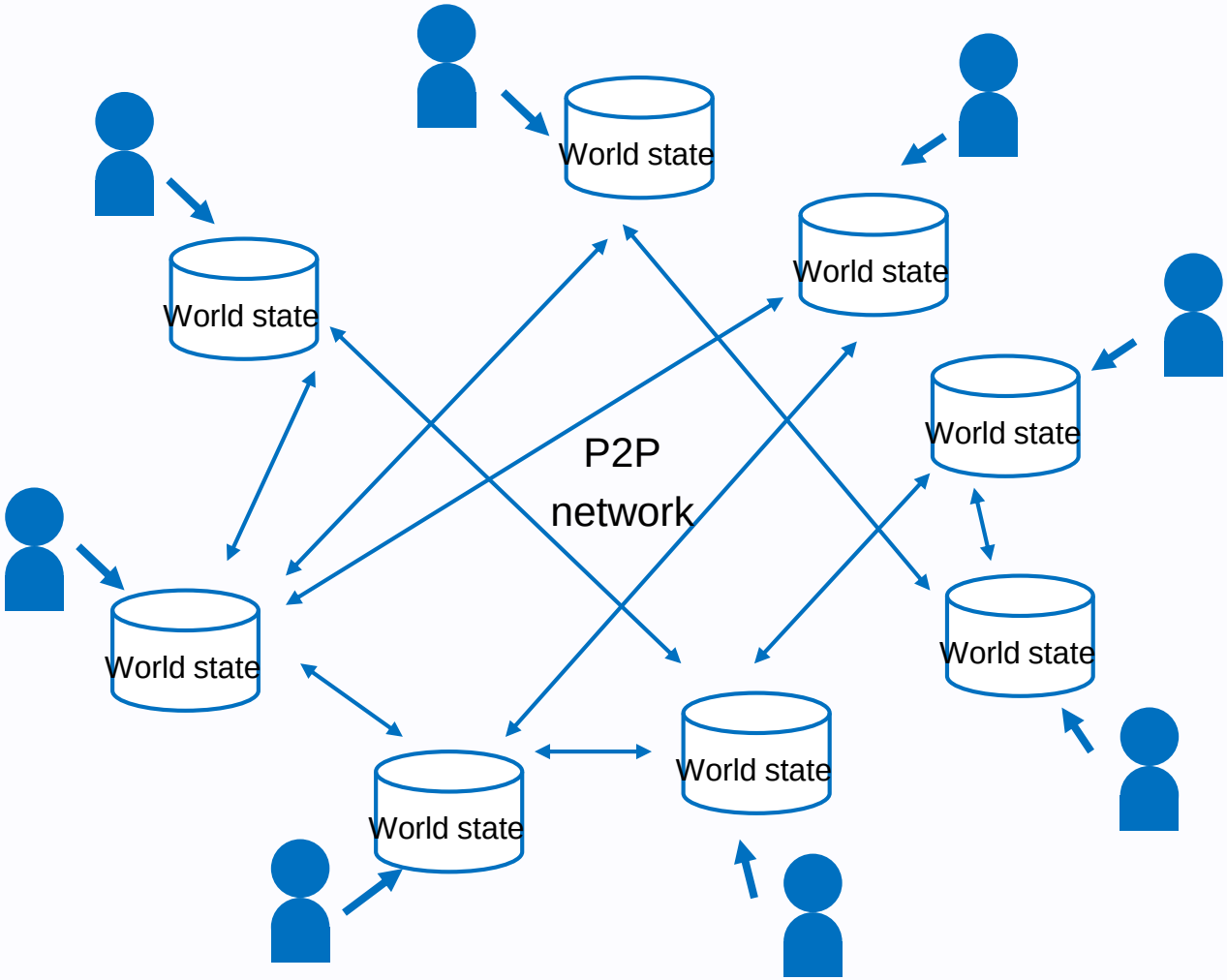
# Globally shared, transactional database




A blockchain is a globally shared, transactional database.

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

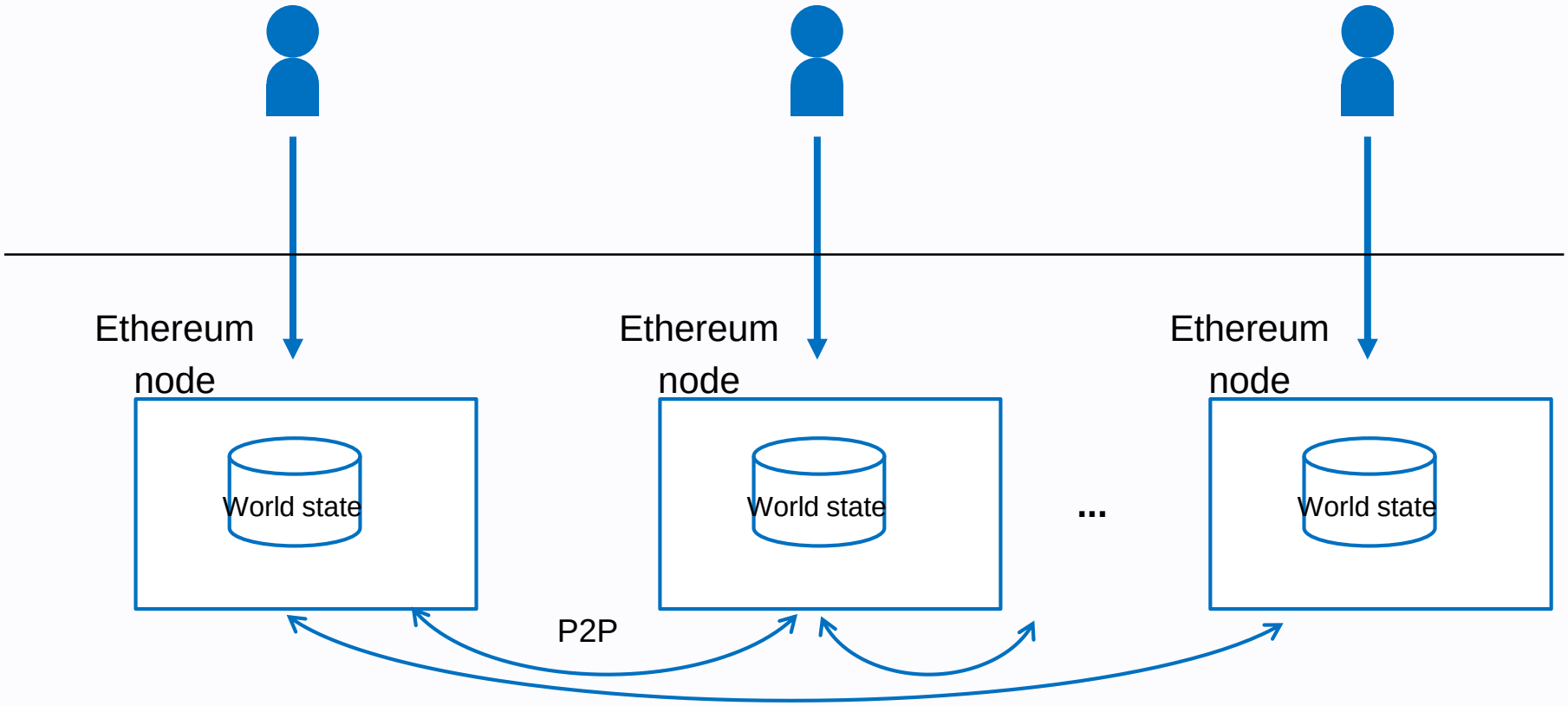
# Decentralised database




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

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

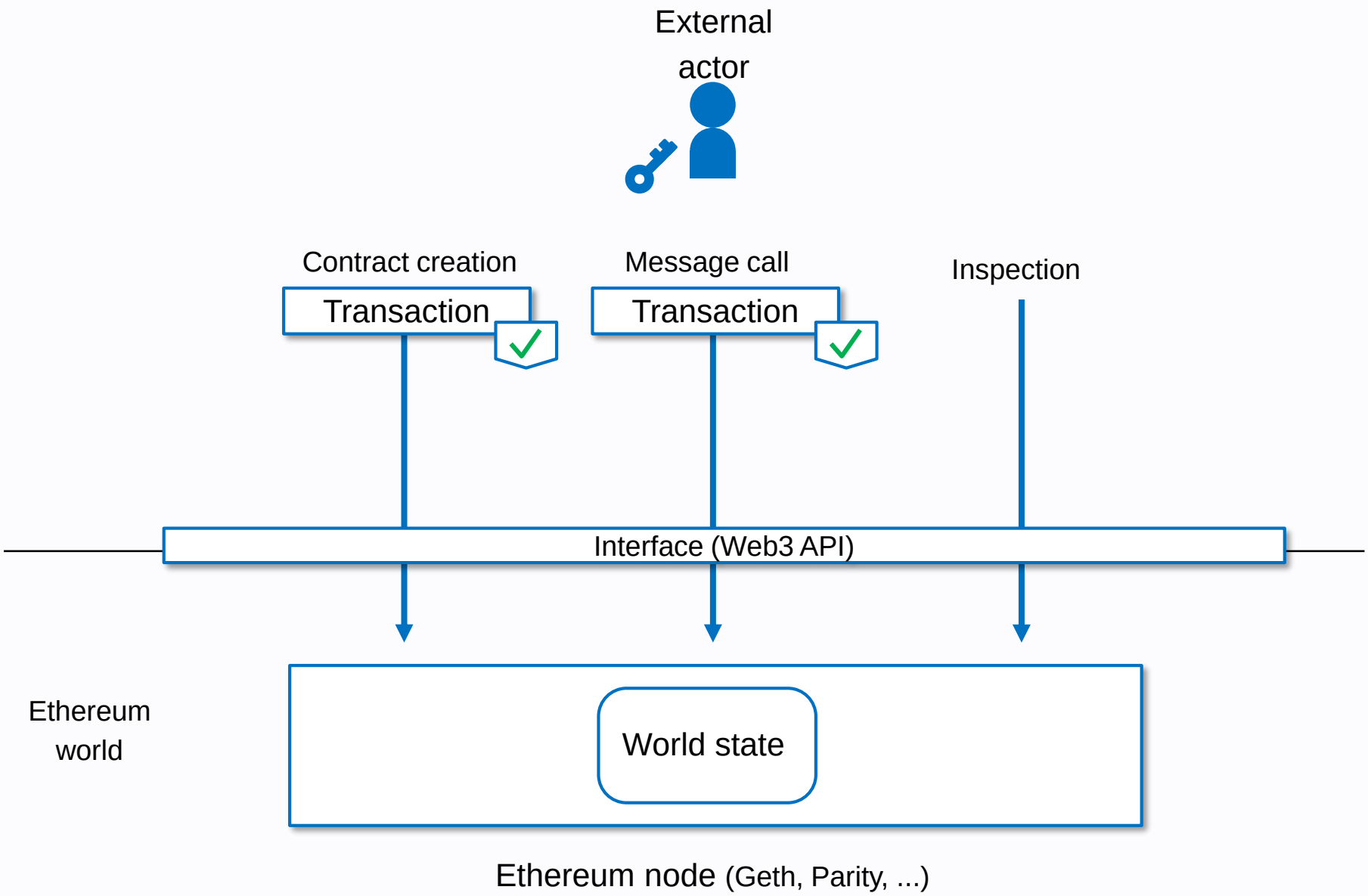
# P2P network inter nodes




Decentralised nodes construct Ethereum P2P network.

References : [E3],

# Interface to a node




External actor accesses Ethereum world via Ethereum node.

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

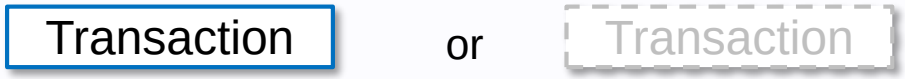
# 1. Introduction

## Atomicity and ordering

# Atomicity of transaction

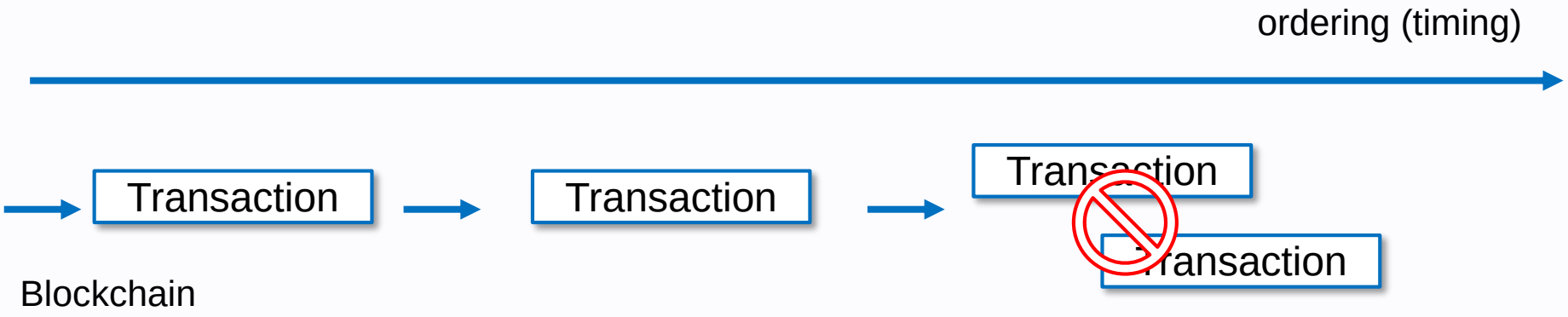

A transaction is atomic operation. Can't divide.

---




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

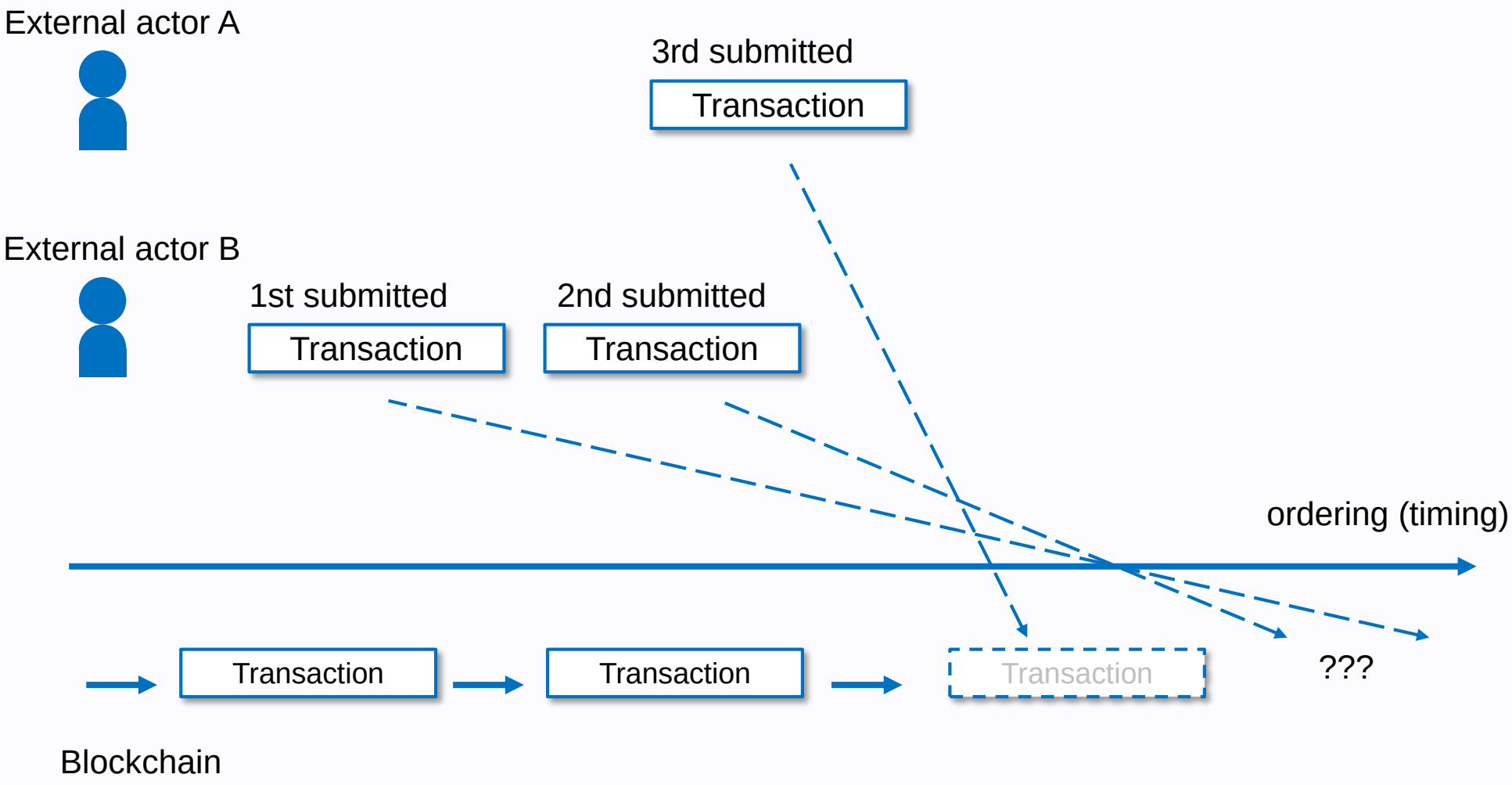
# Ordering of transactions




Transactions can not be overwrap.

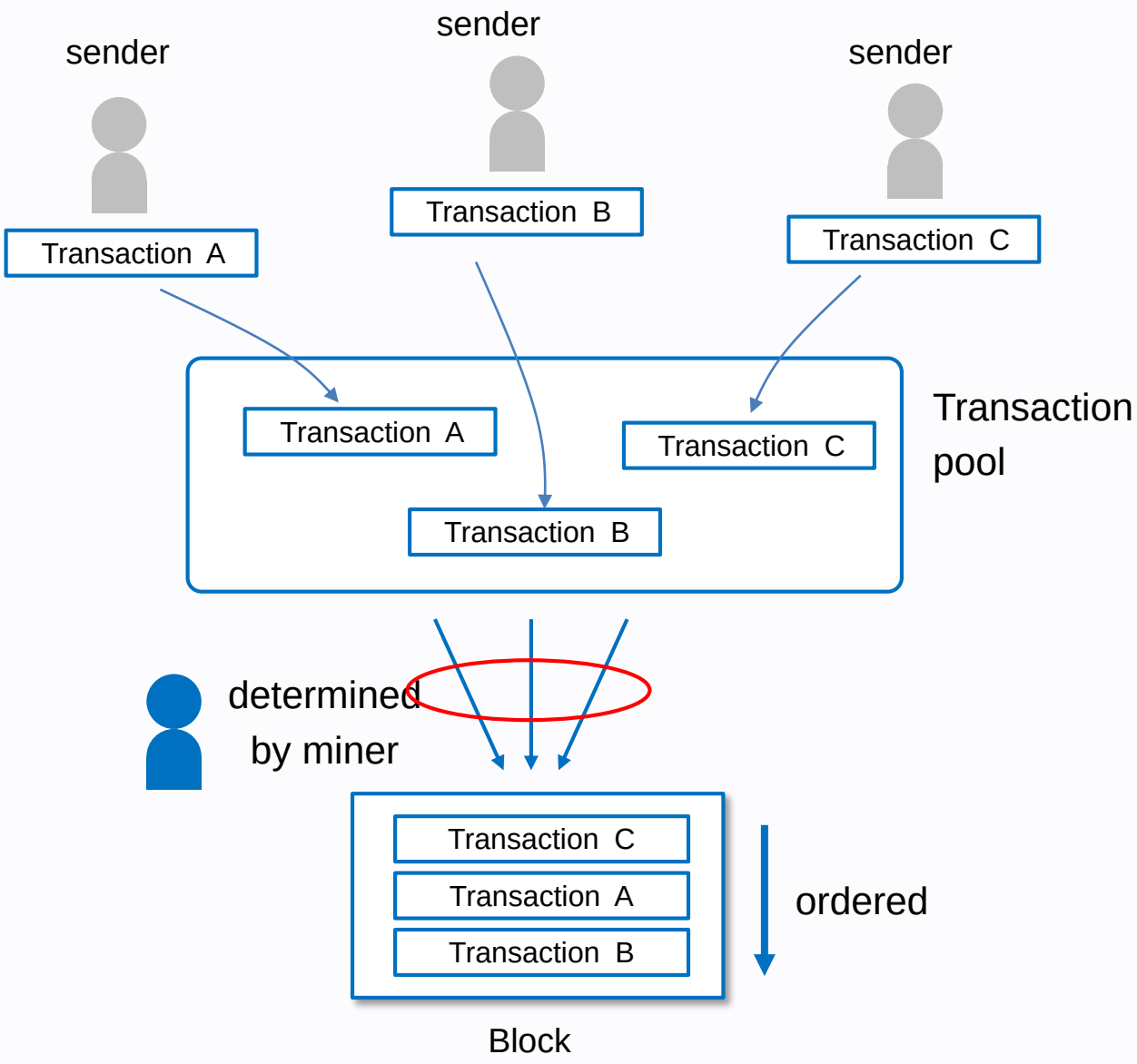
Transactions must perform sequentially.

# Ordering of transactions



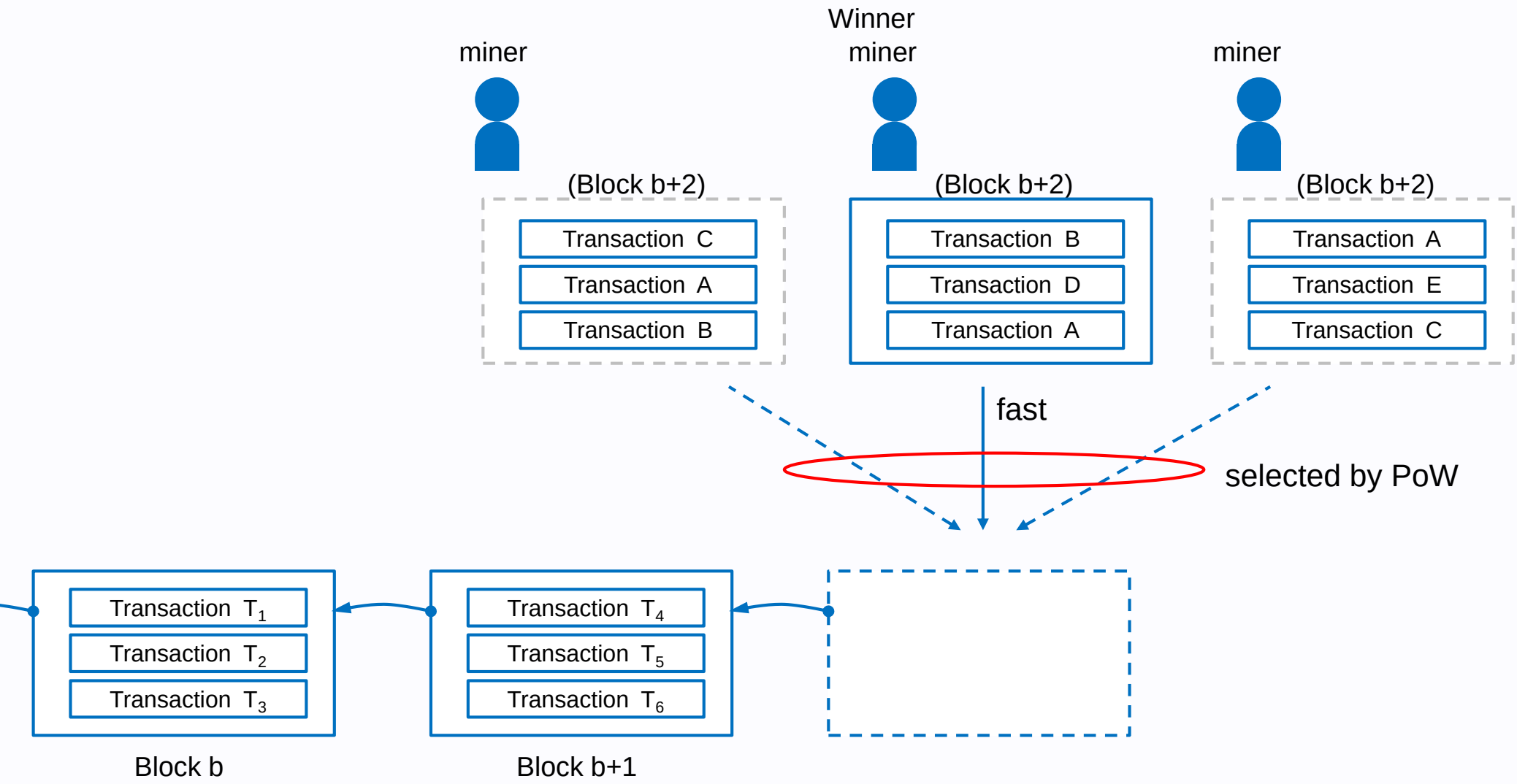

Ordering of transactions does NOT be guaranteed.

# Ordering inner block




Miner can determine order of transactions in a block.

# Ordering inter blocks




Ordering inter blocks is determined by consensus algorism, such as PoW.

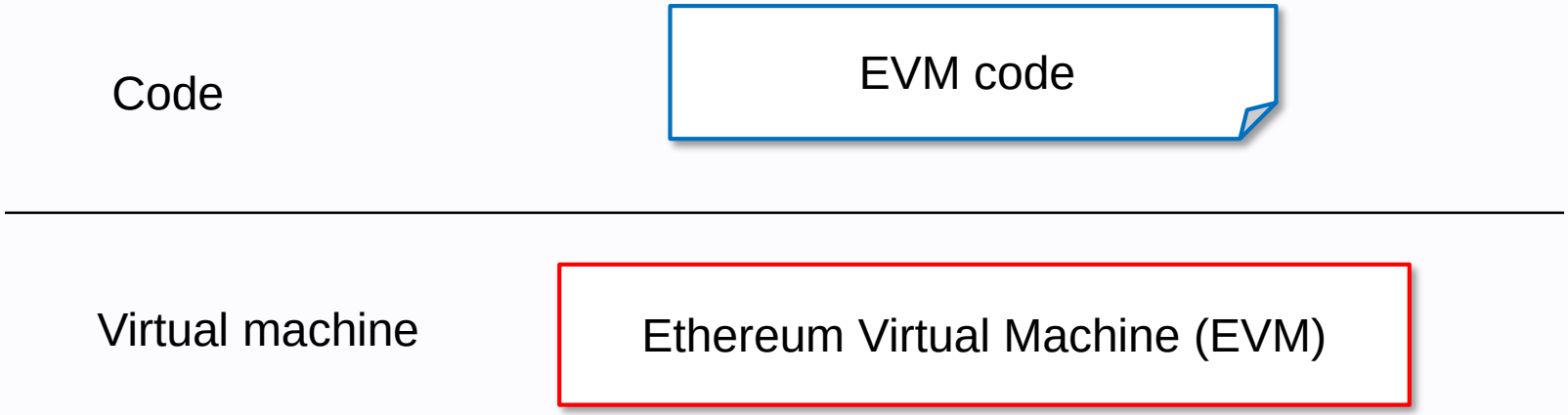
References : [E1] Ch.2, Ch.4

# 2. Virtual machine

# 2. Virtual machine

Ethereum virtual machine (EVM)

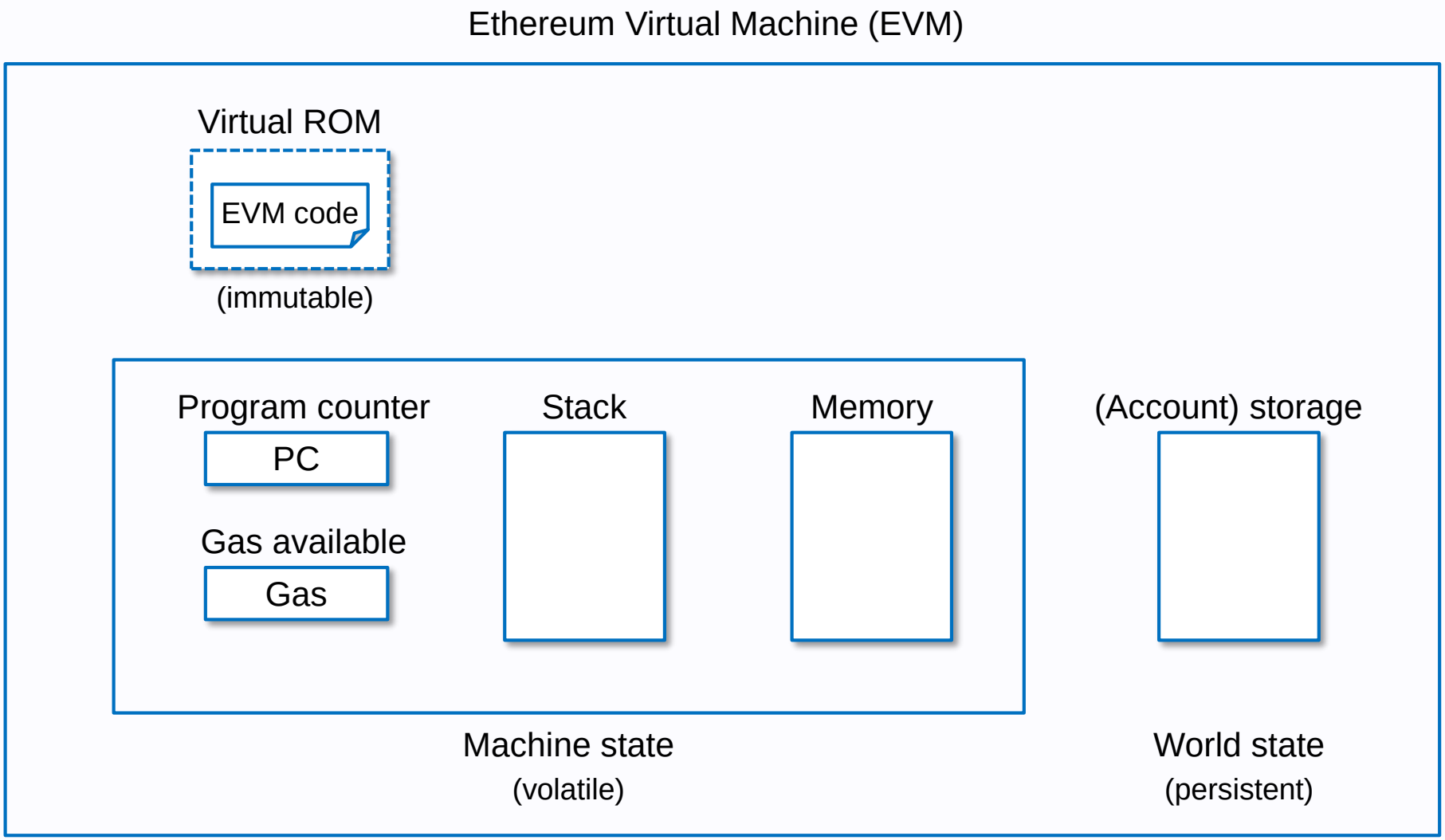
# Ethereum virtual machine




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

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

# EVM architecture



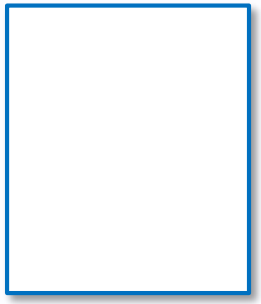

The EVM is a simple stack-based architecture.

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

# Machine space of EVM

Registers

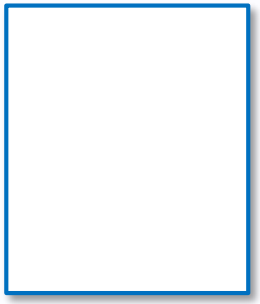
Stack



stack memory

256 bits x 1024 elements

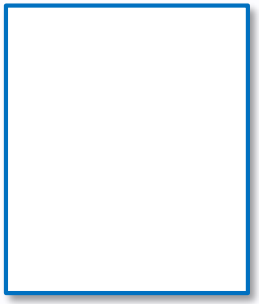
Memory



volatile memory

byte addressing
linear memory

(Account) storage



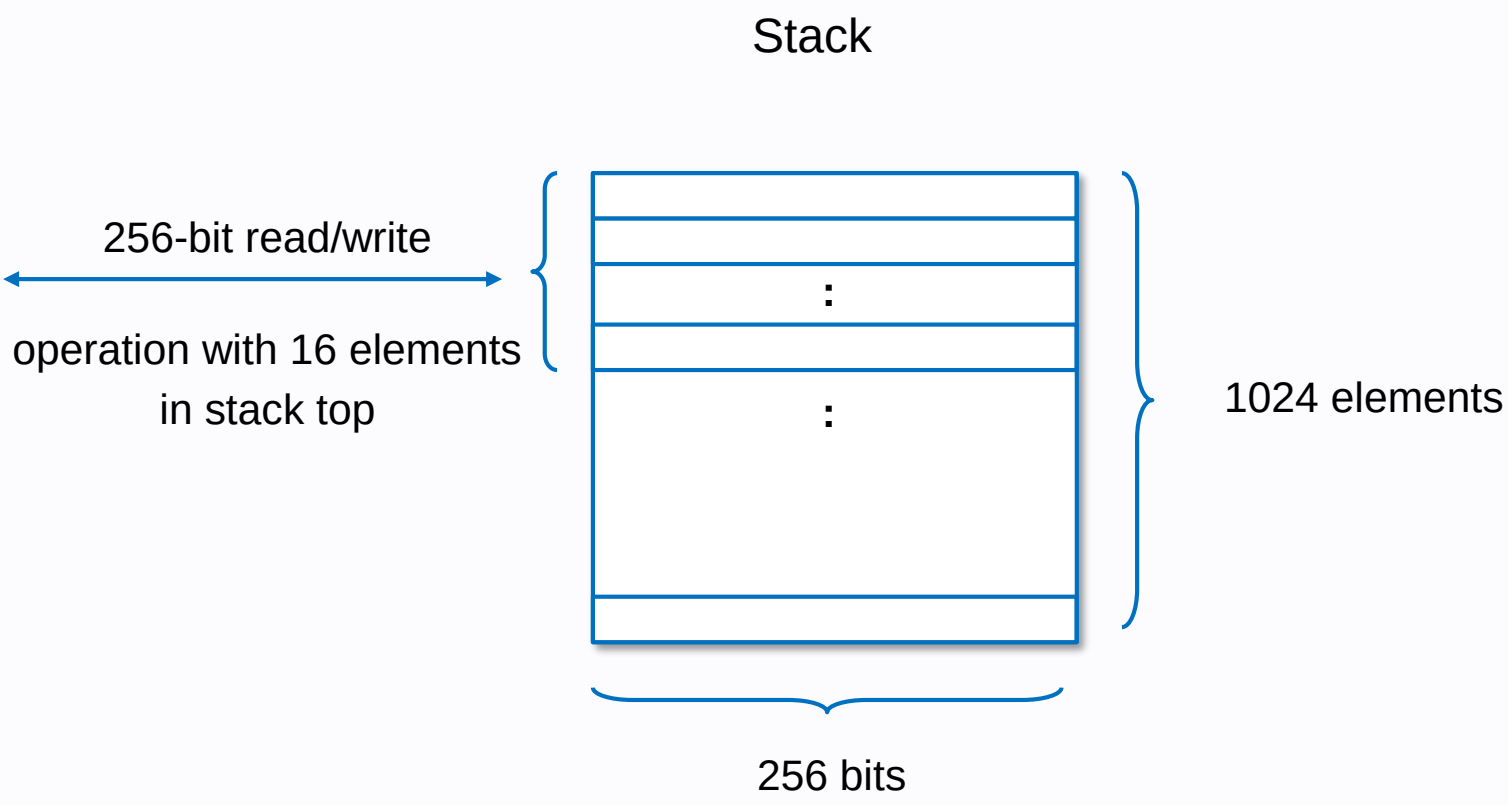
persistent memory

256 bits to 256 bits
key-value store

There are several space resources.

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

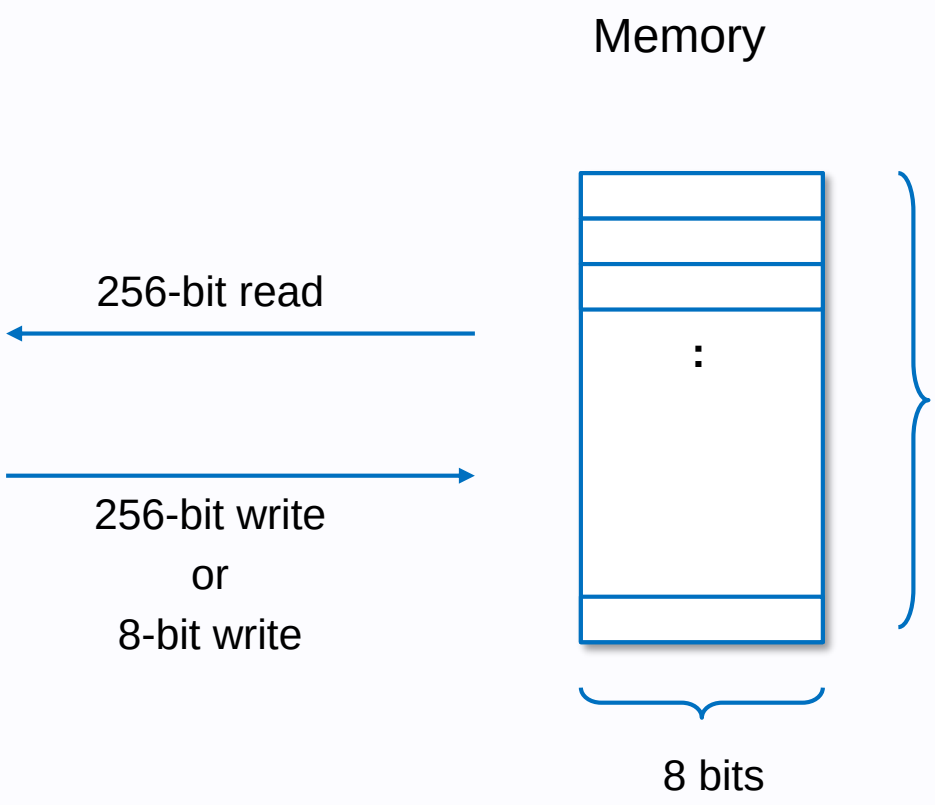
# Stack




All operation are performed on the stack.
Access with many instructions such as PUS/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.

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

# Account storage

(Account) storage

256-bit read/write

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

256 bits 256 bits

Storage is a key-value store that maps 256-bit words to 256-bit words.
Access with SSTORE/SLOAD instructions.

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

# EVM code

Assembly view

```
ADD
EXP
PUSH1
MLOAD
  :
  :
```

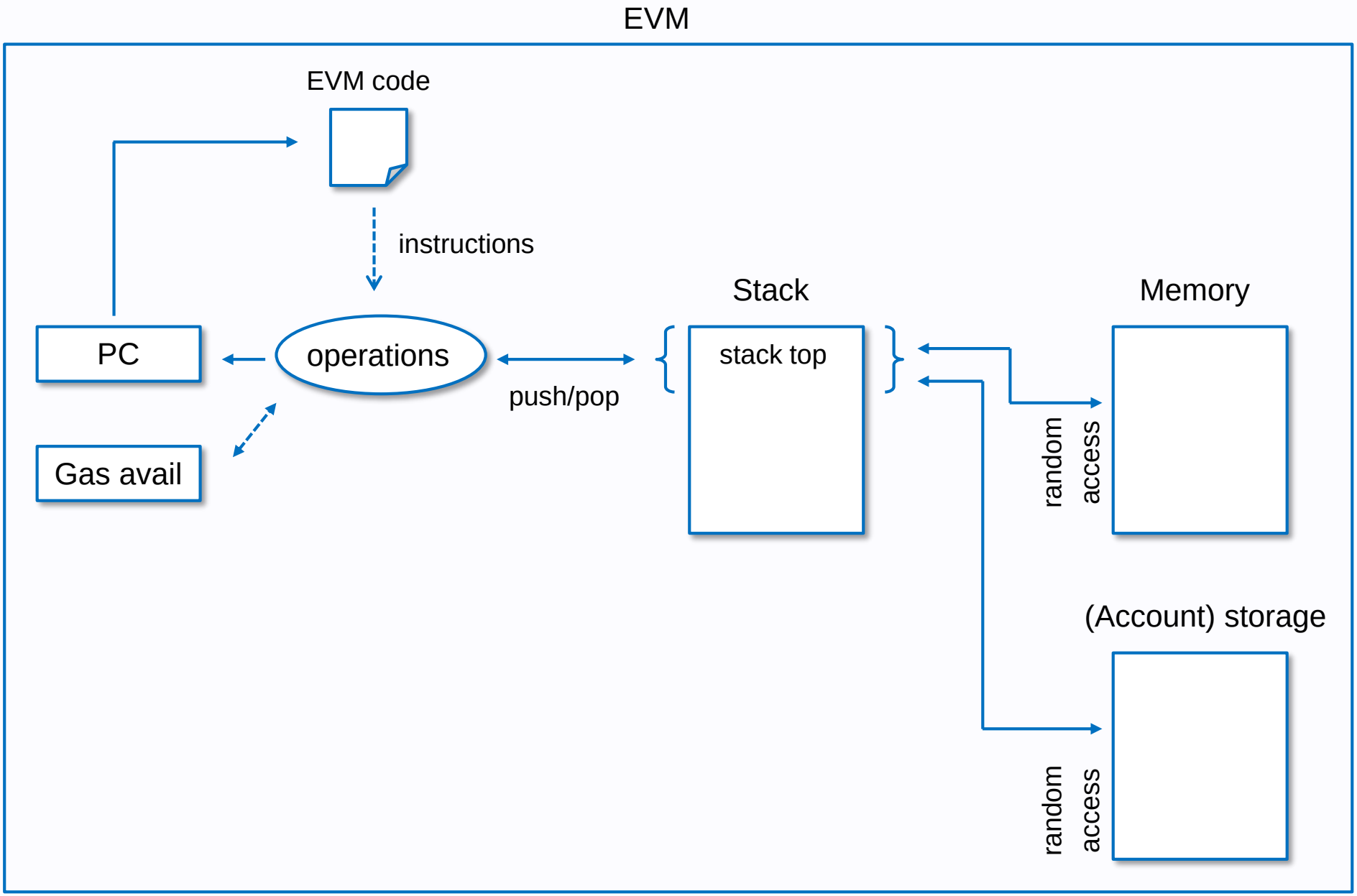
Bytecode view

```
0x010A6051...
```

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

References : [E1] Ch.9, Appendix H

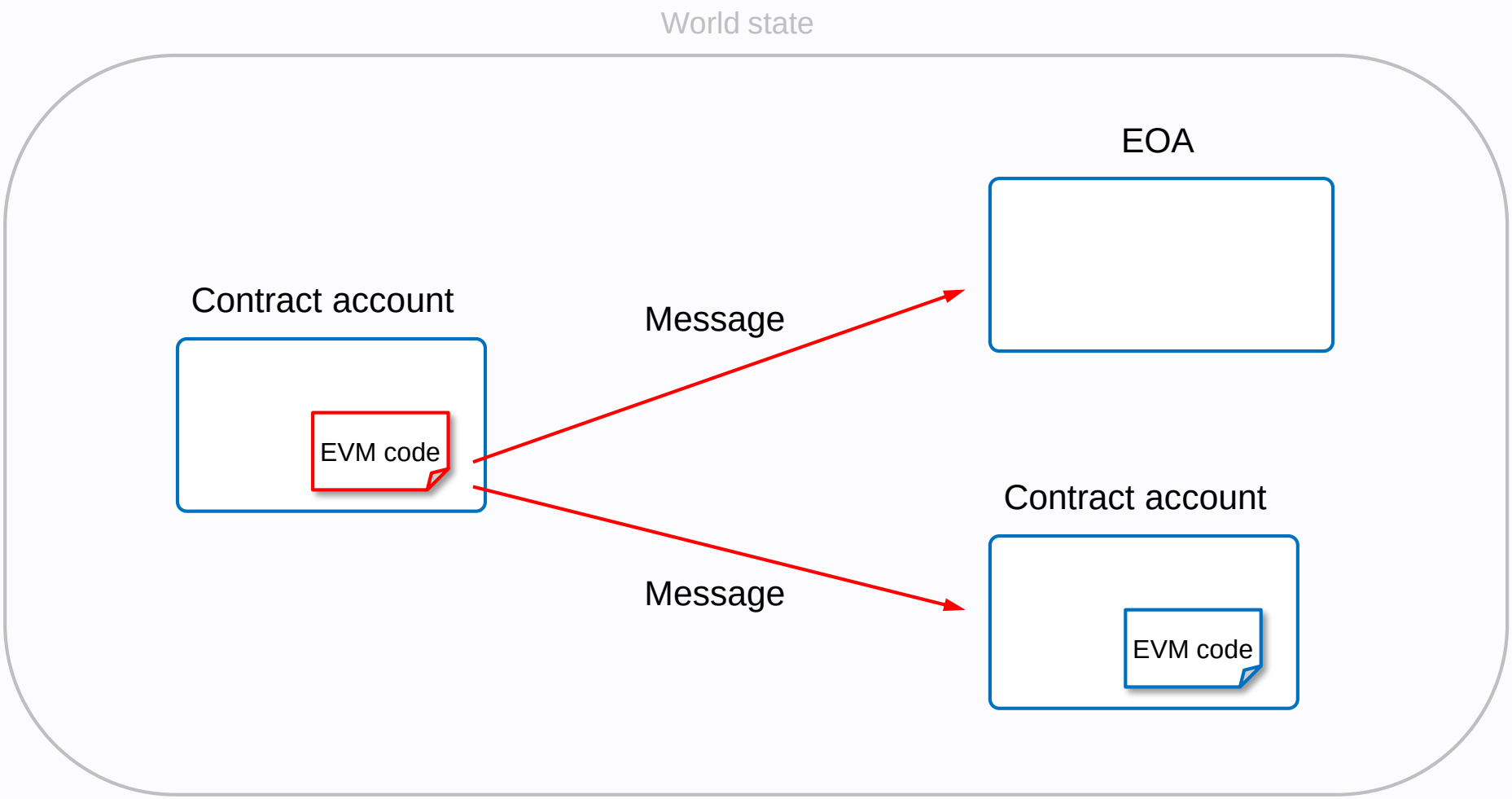
# Execution model




References : [E1] Ch.9, Appendix H

# Message call

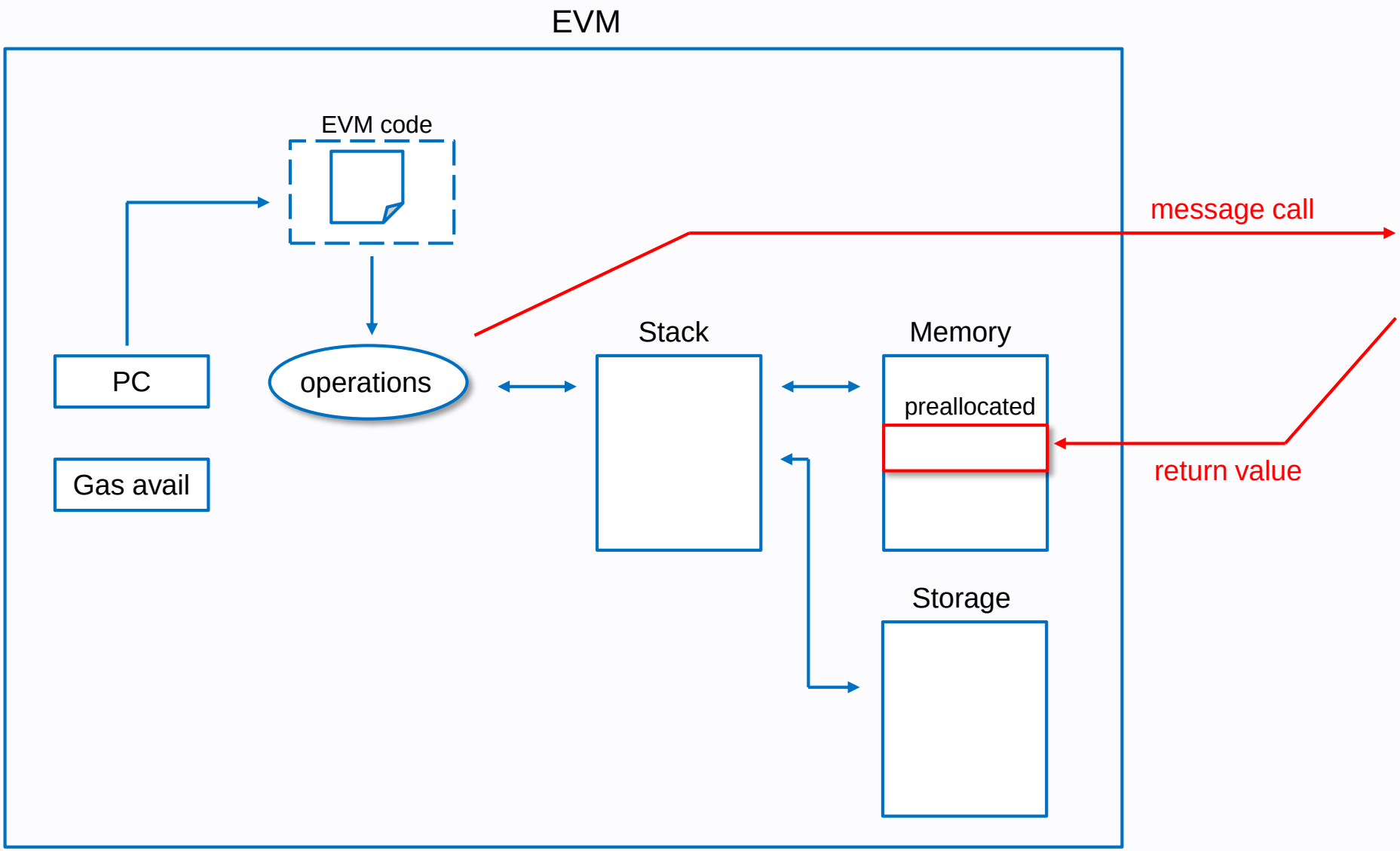
# Message call




EVM code can send a message to other account.

References : [E1] Ch.8, Appendix A

# Message call

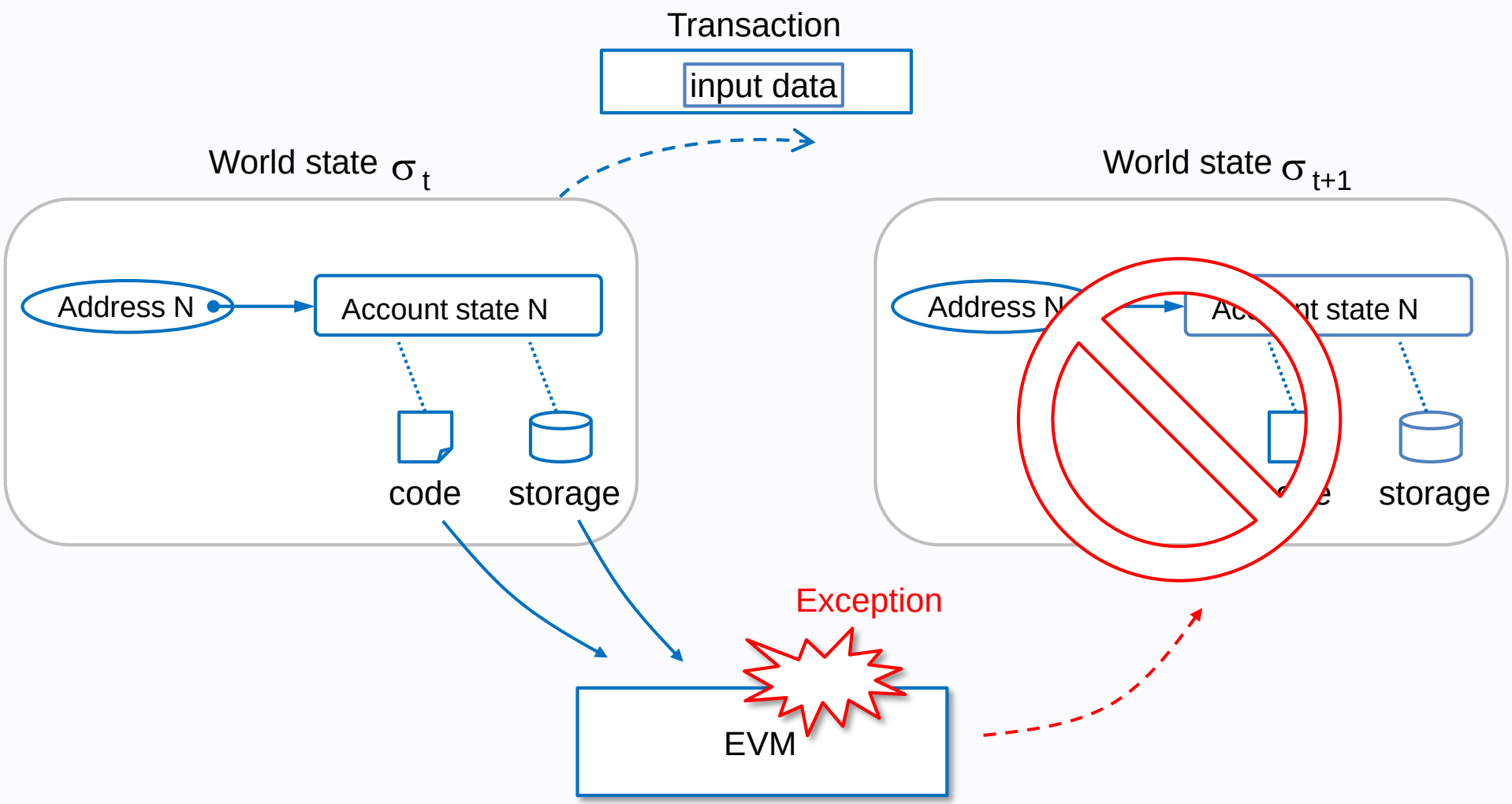



References : [E1] Ch.7, Ch.9

# 2. Virtual machine

## Exception

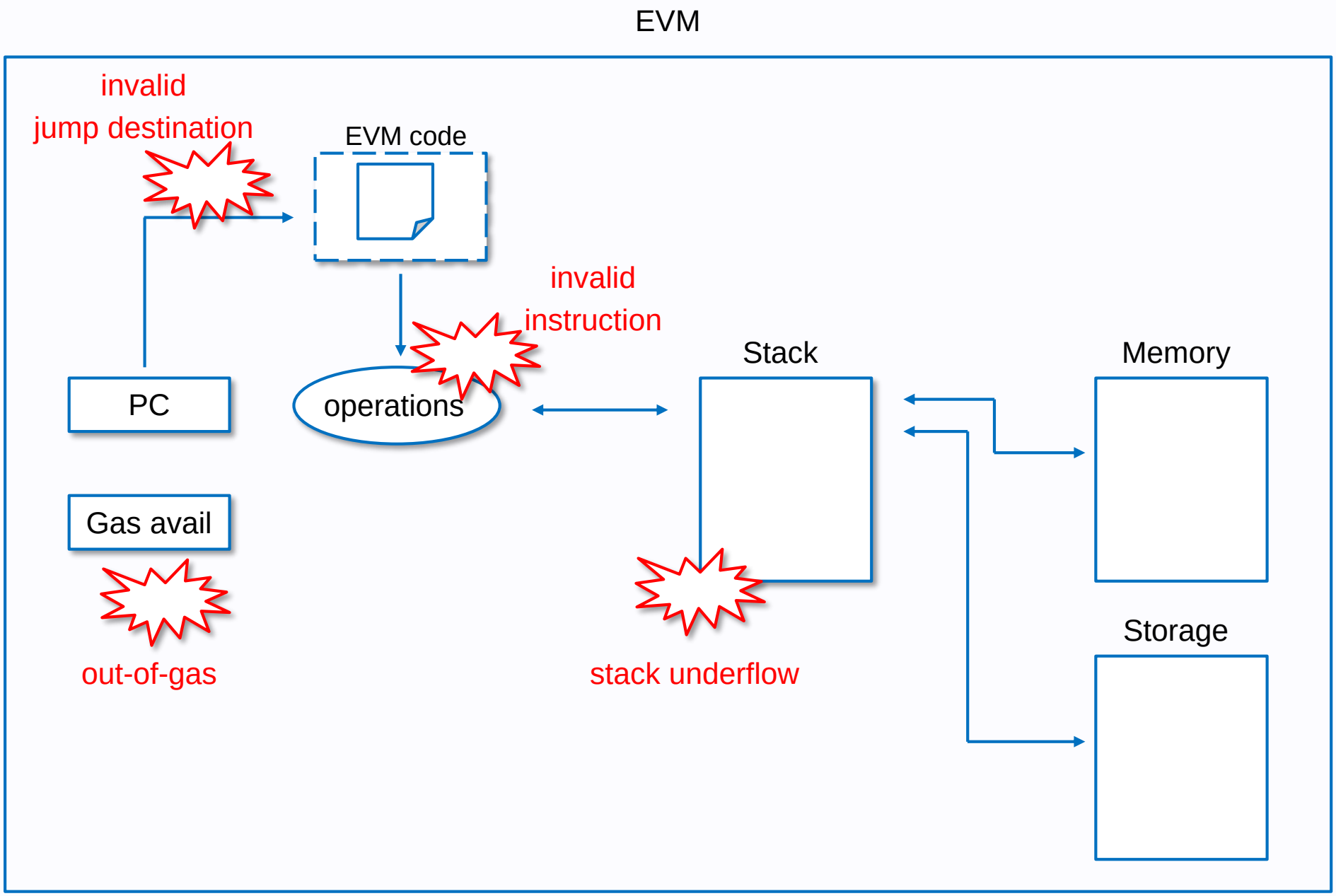
# Exception




When exception is occurred on EVM, state will NOT update.

References : [E1] Ch.9, Appendix H

# Exception

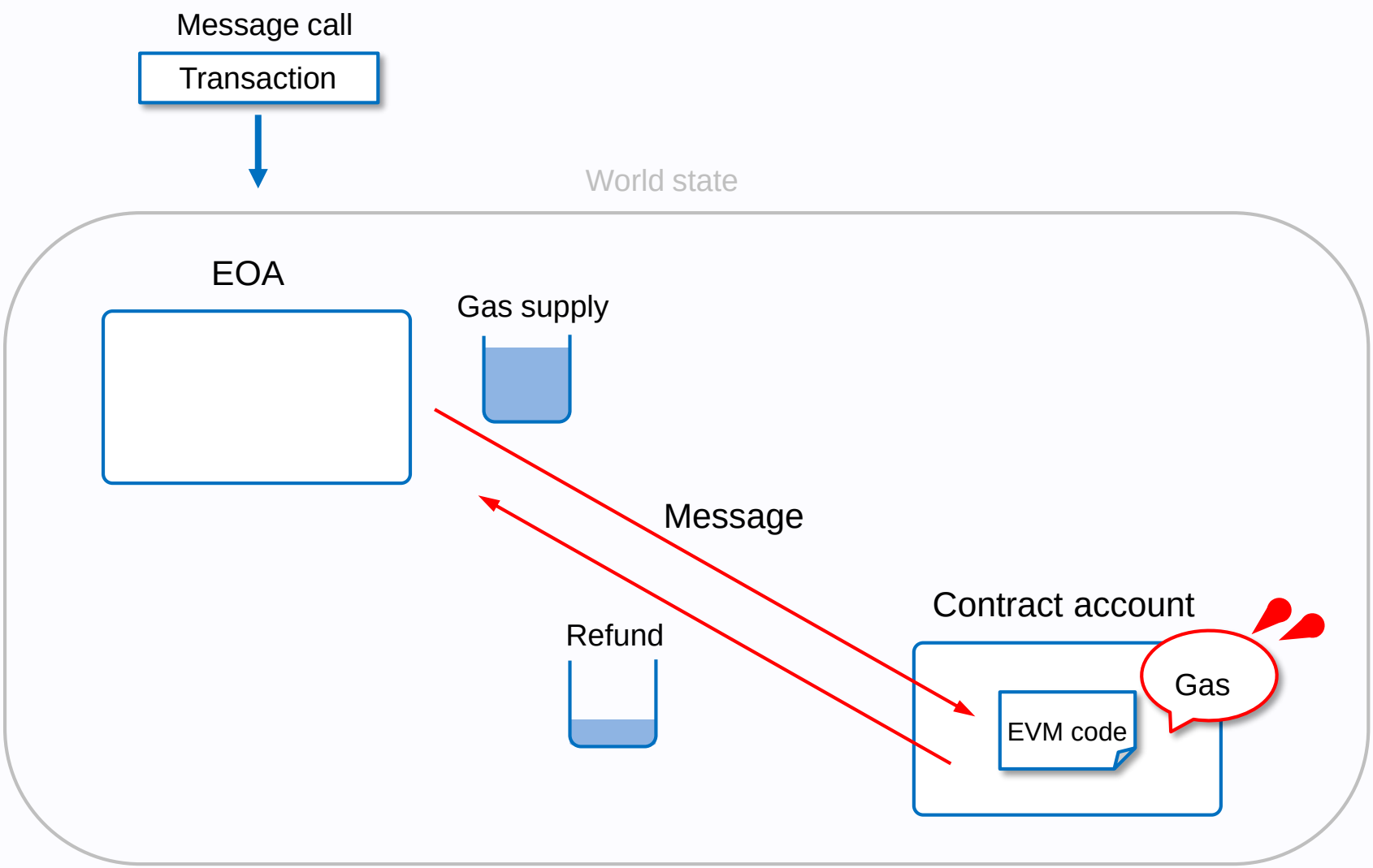



References : [E1] Ch.7

# 2. Virtual machine

## Gas and fee

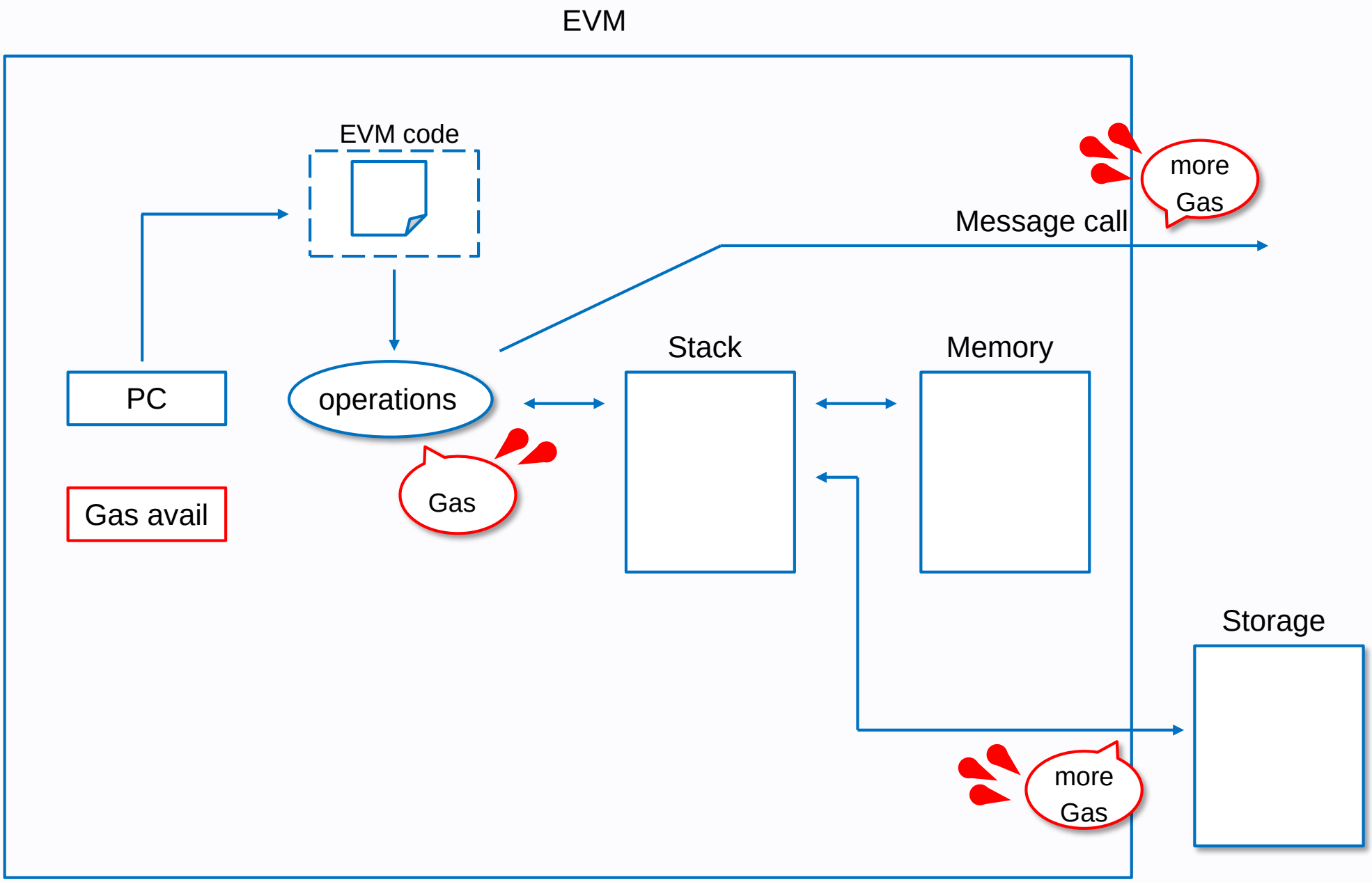
# Gas and fee




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

References : [E1] Ch.8, Appendix A

# Gas




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

# 2. Virtual machine

## Instruction set

# Instruction set

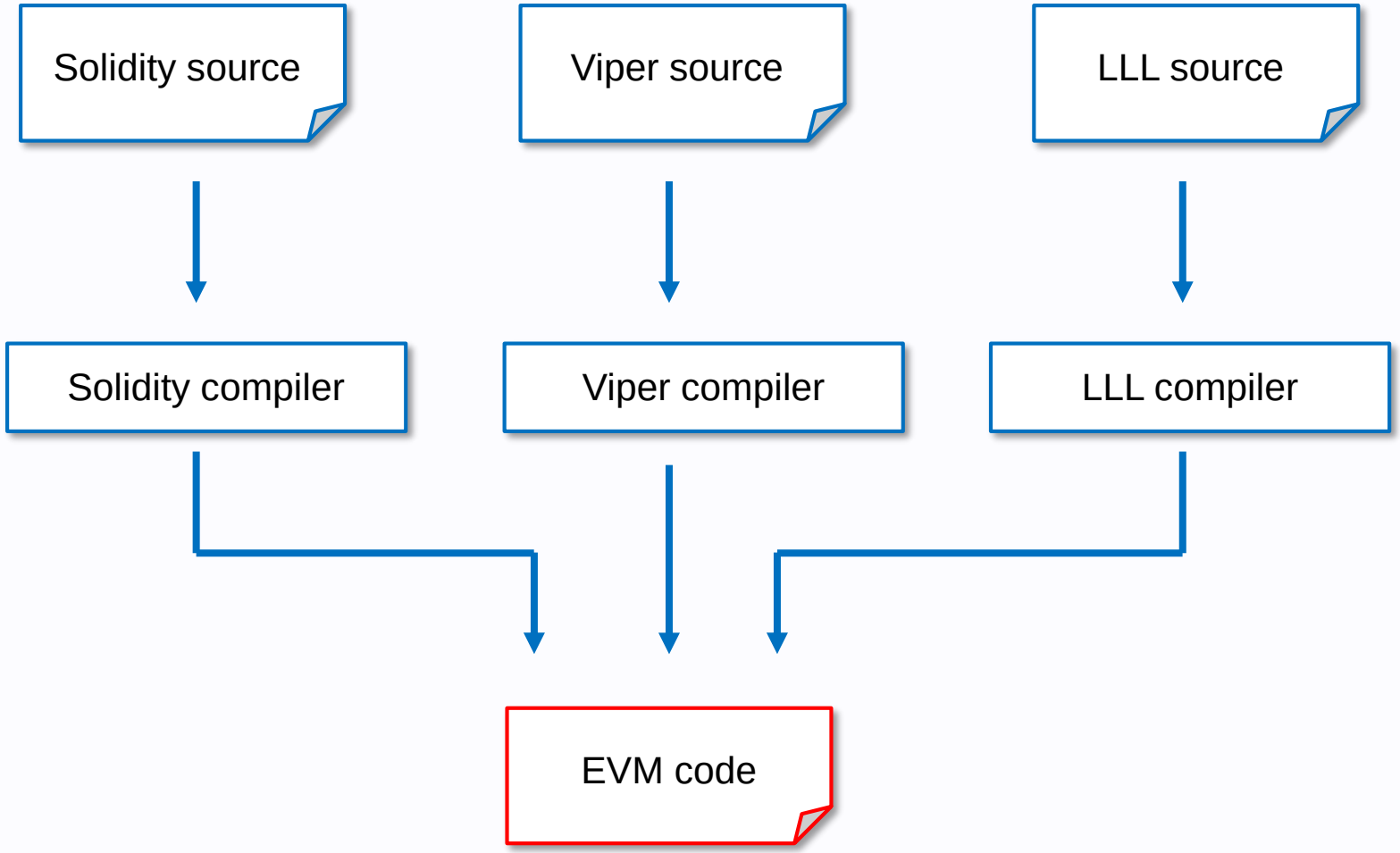
WIP

References : [E1] Ch.9, Appendix H

# 2. Virtual machine

## Miscellaneous

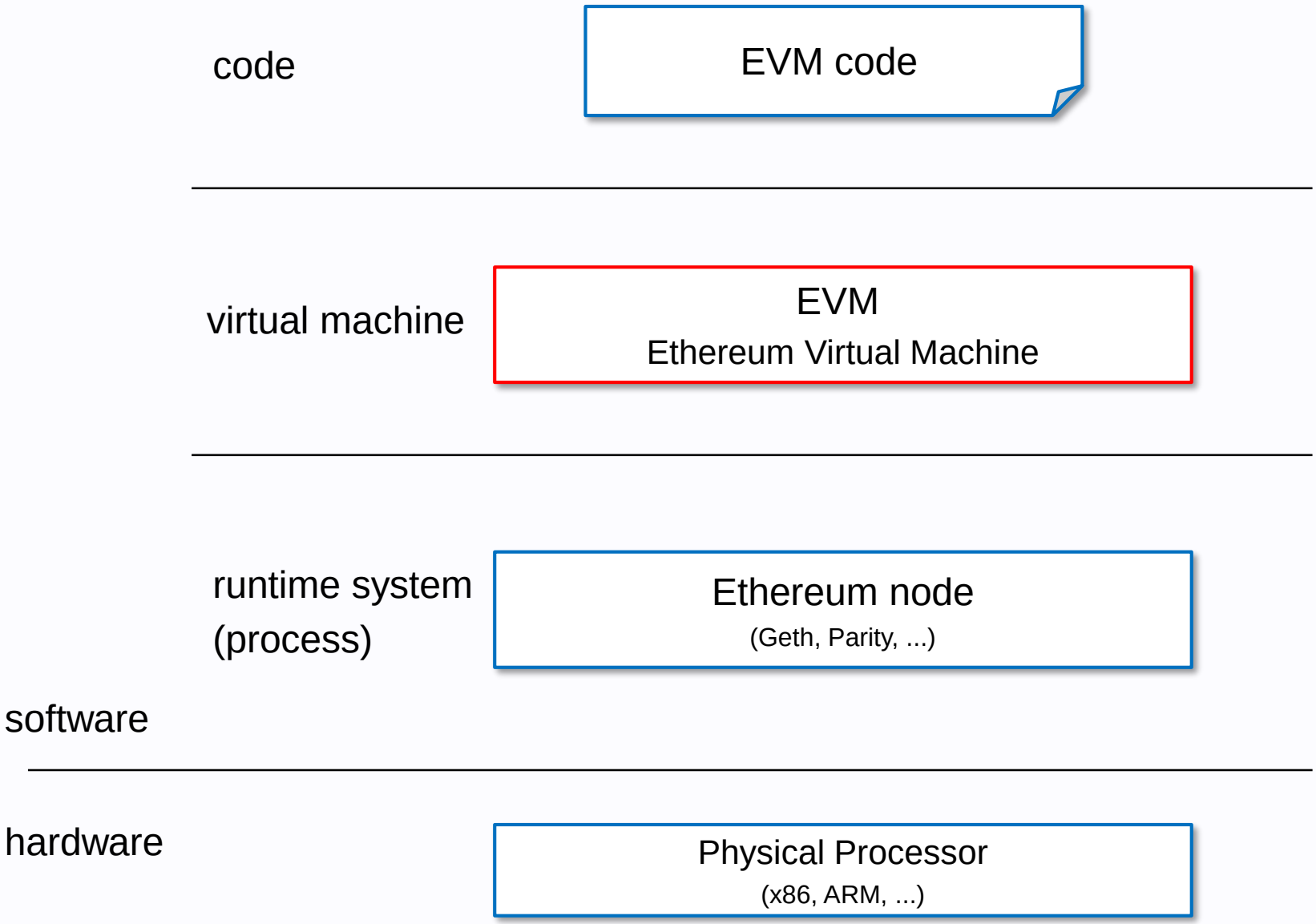
# EVM code generation




Ethereum virtual machine code

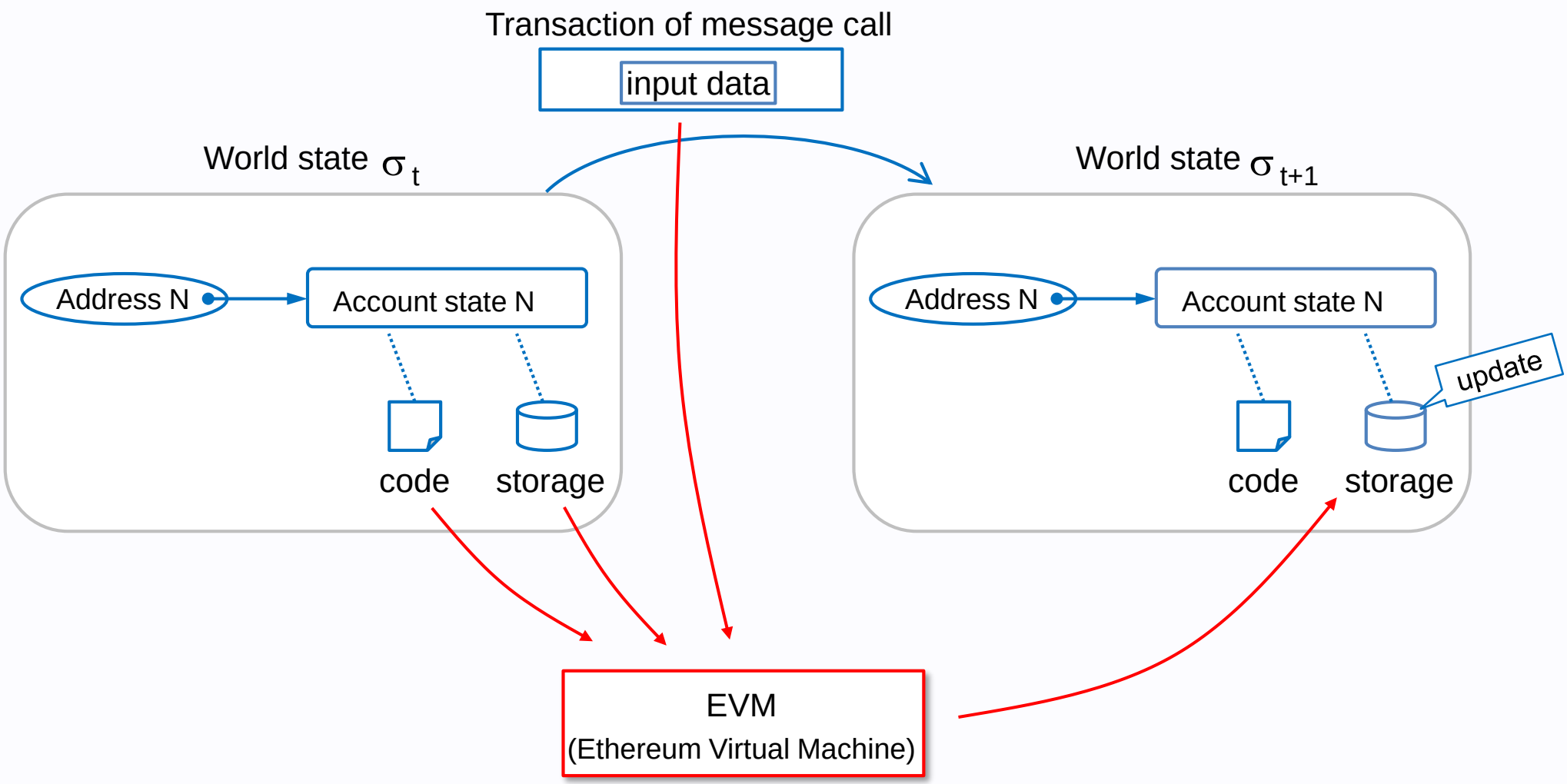
References : [E7]

# Ethereum virtual machine layer




References : [E1] Ch.9

# Starting EVM




EVM is started with a message call.

References : [E1] Ch.9, Appendix H

# WASM

WASM is next generation VM.

WIP

# Appendix A

# Appendix A

## Implementation code in Geth

# Block header

[core/types/block.go]

Block header

```
type Header struct {
        ParentHash   common.Hash     `json:"parentHash"        gencodec:"required"`
        UncleHash    common.Hash     `json:"sha3Uncles"        gencodec:"required"`
        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"      gencodec:"required"`
        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"`
}
```

Root of State

Root of Transaction

References : [C1]

# Transaction

[core/types/transaction.go]

Transaction

```
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:"-"`
}
```

to address

value (Ether)

input data

References : [C1]

# World state

[core/state/statedb.go]

World state

```
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

           :
```

Mapping for
Address to Account state

References : [C1]

# Account object (state object)

[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

[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

[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)

[core/vm/instruction.go]

Add 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)

[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
}
```

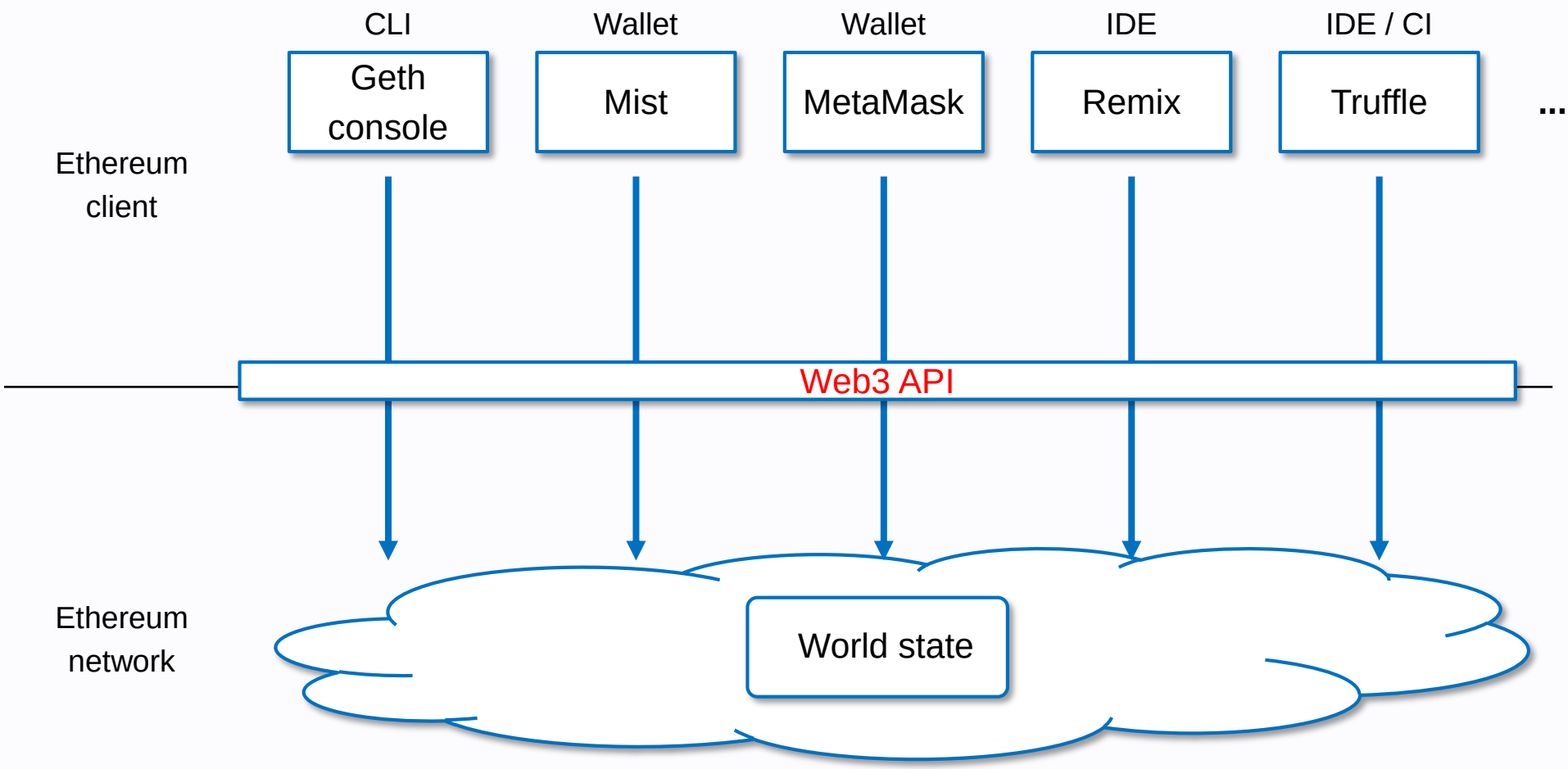
References : [C1]

# Appendix B

# Appendix B

## Web3 API

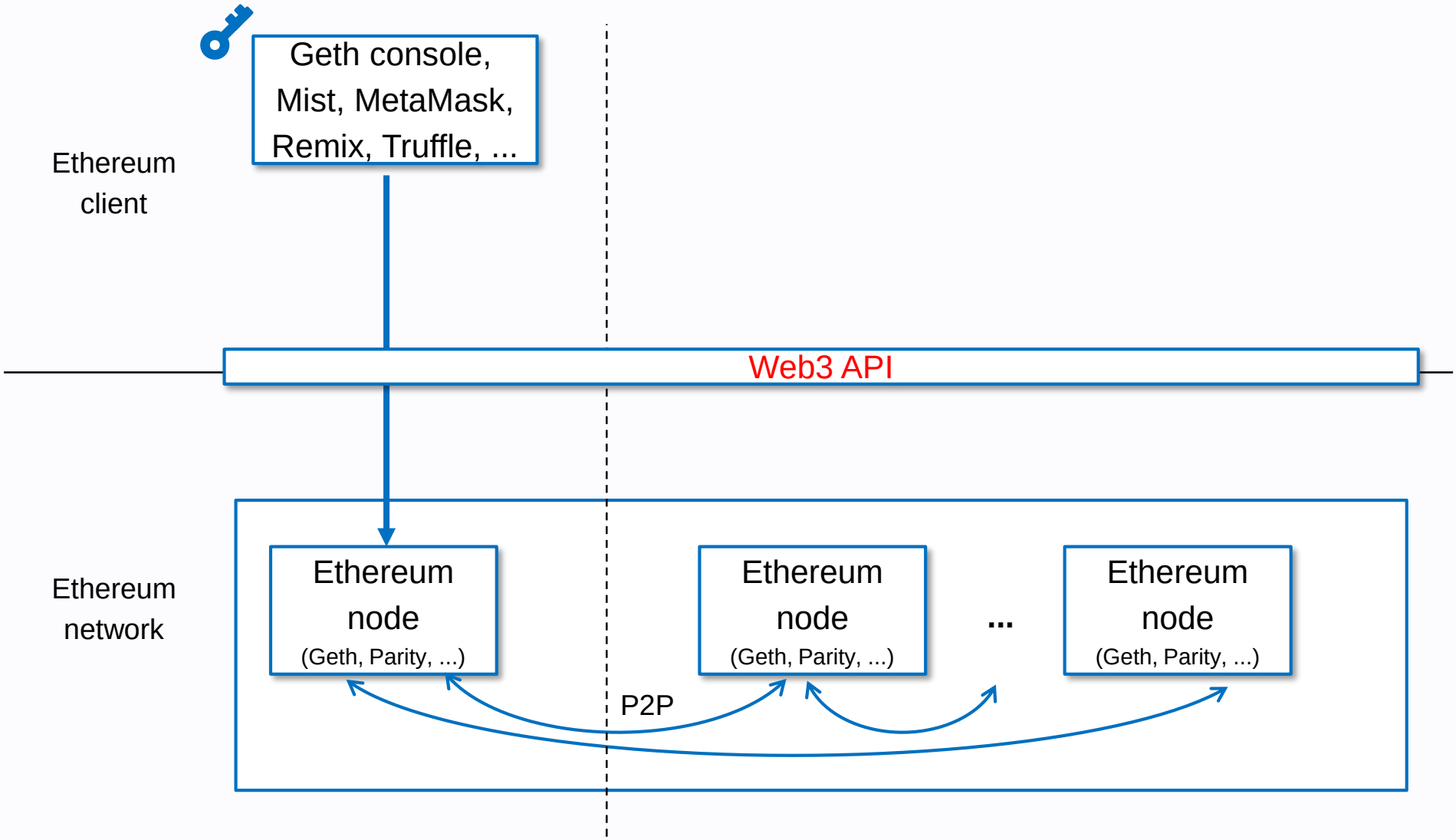
# Web3 API and client




Ethereum clients access to Ethereum network via Web3 API.

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

# Web3 API and client



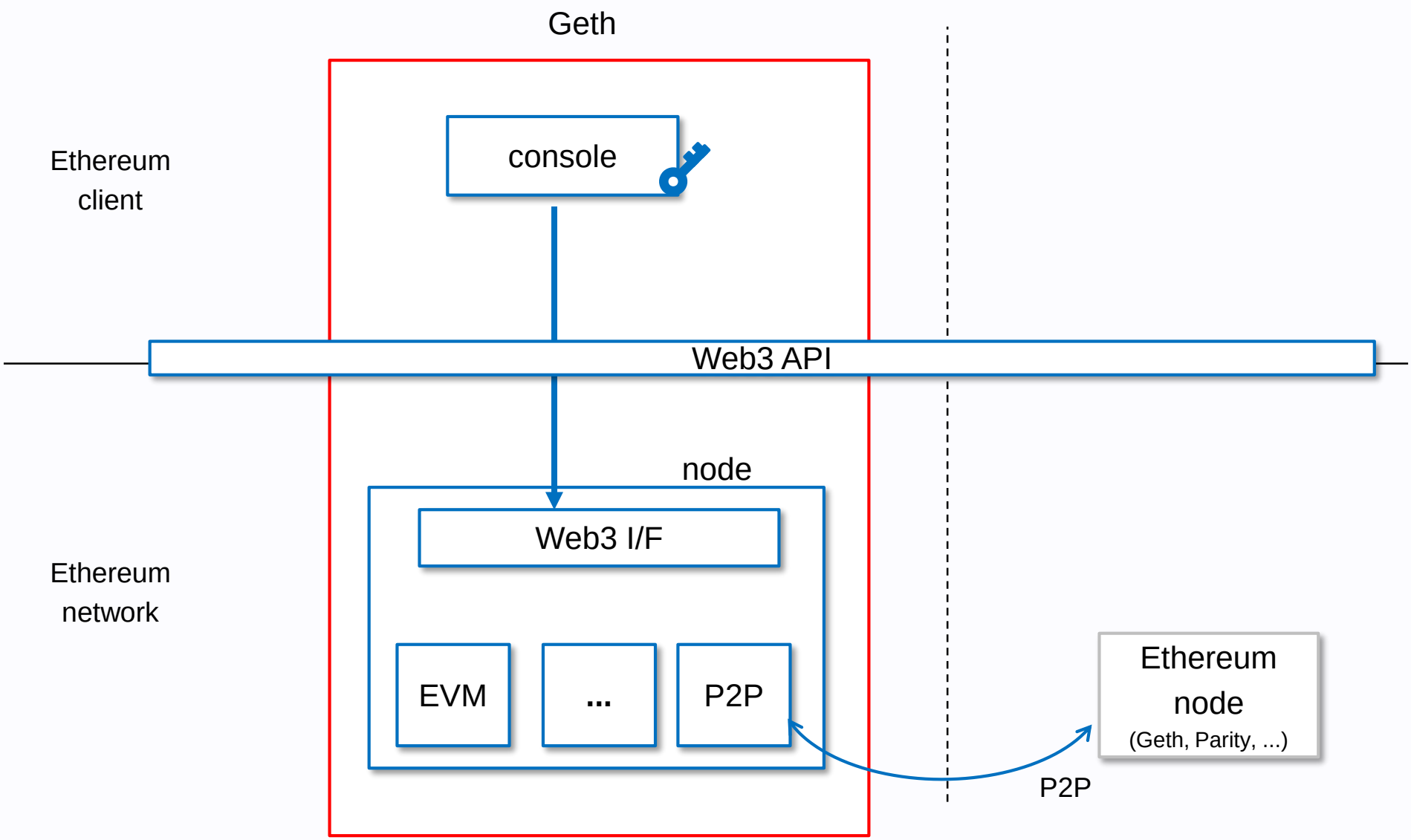

Ethereum clients access to Ethereum network via Web3 API.

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

# Appendix B

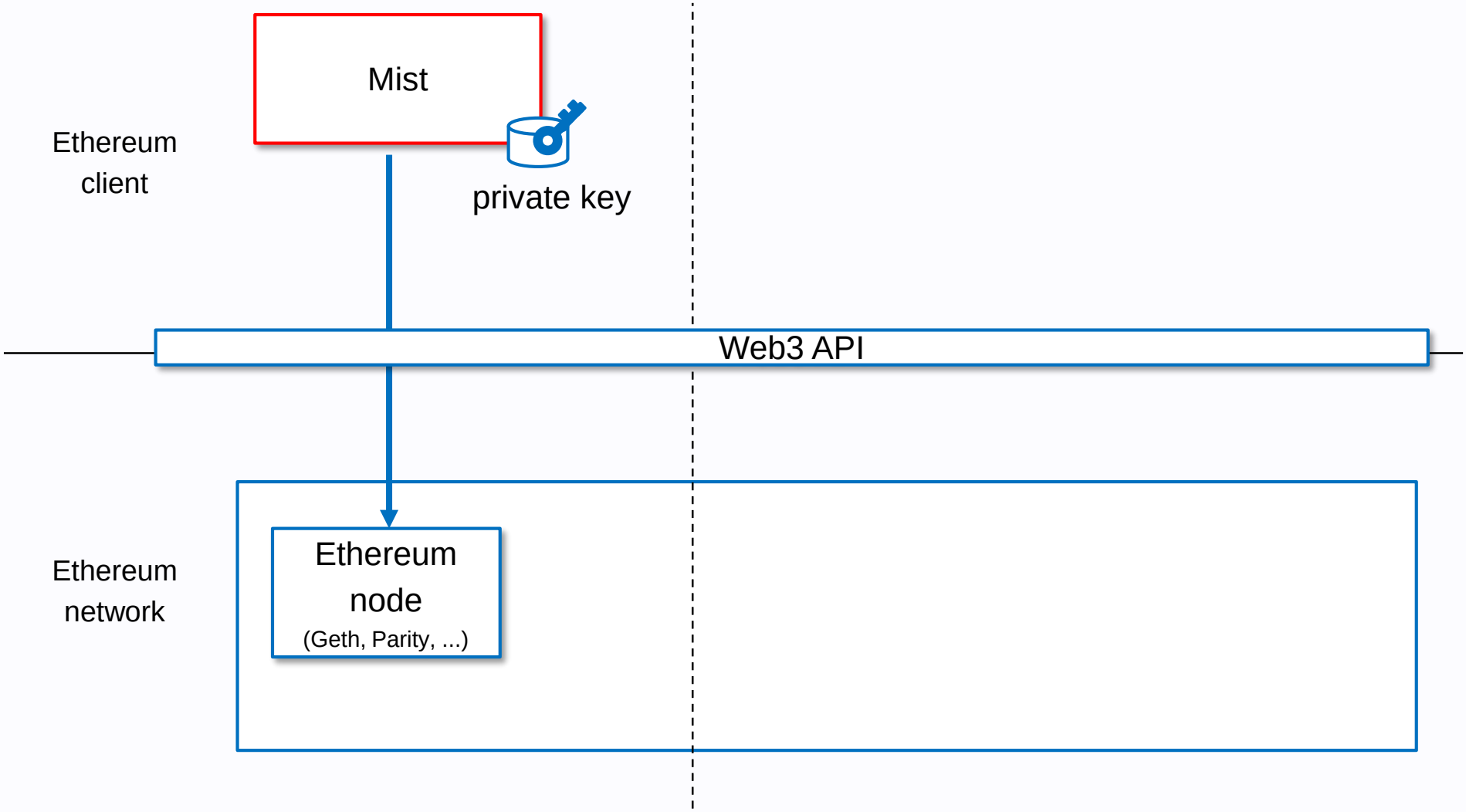
Geth, Mist, Solc, Remix, Truffle, ...

# Geth



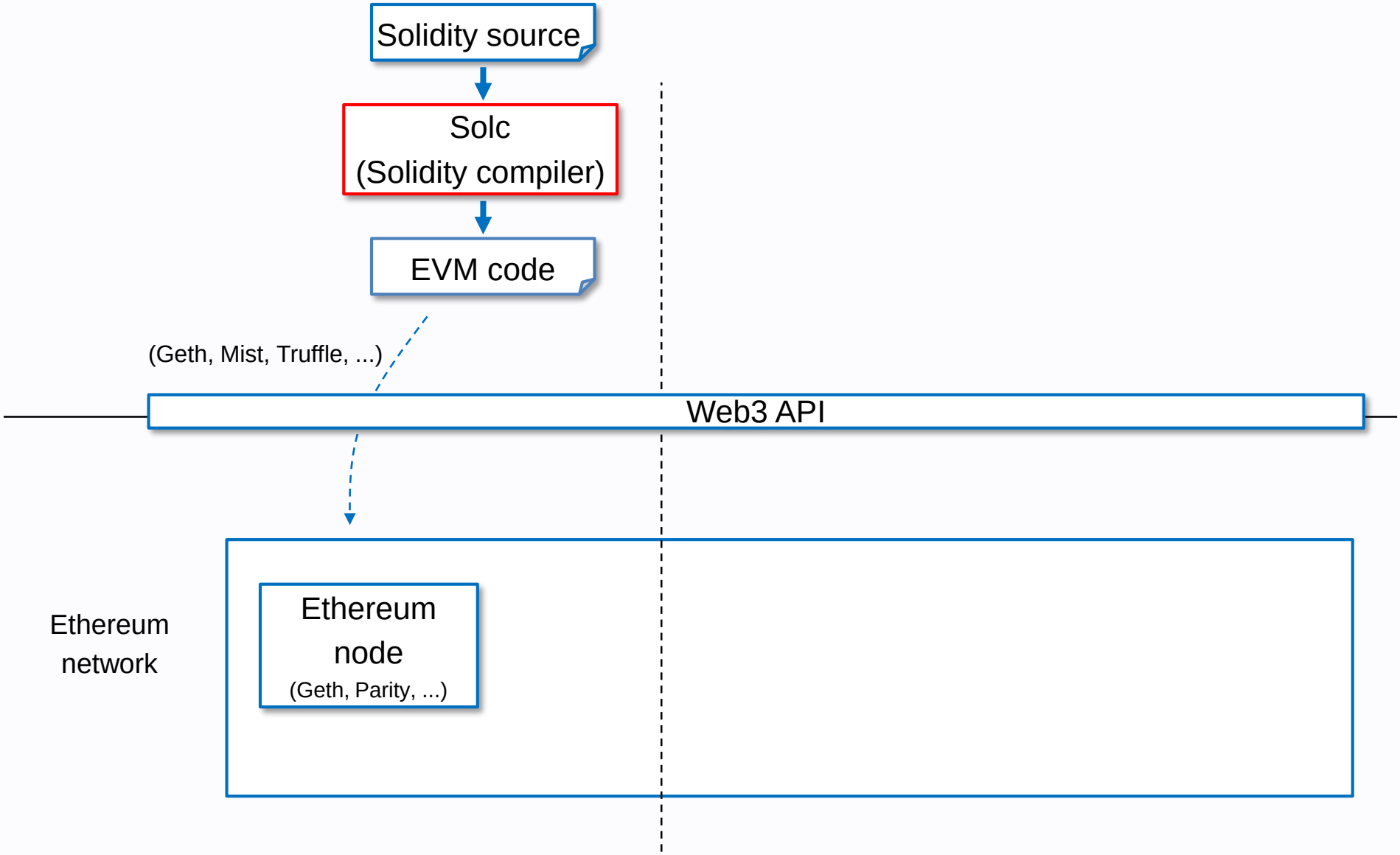

References : [C1]

# Mist



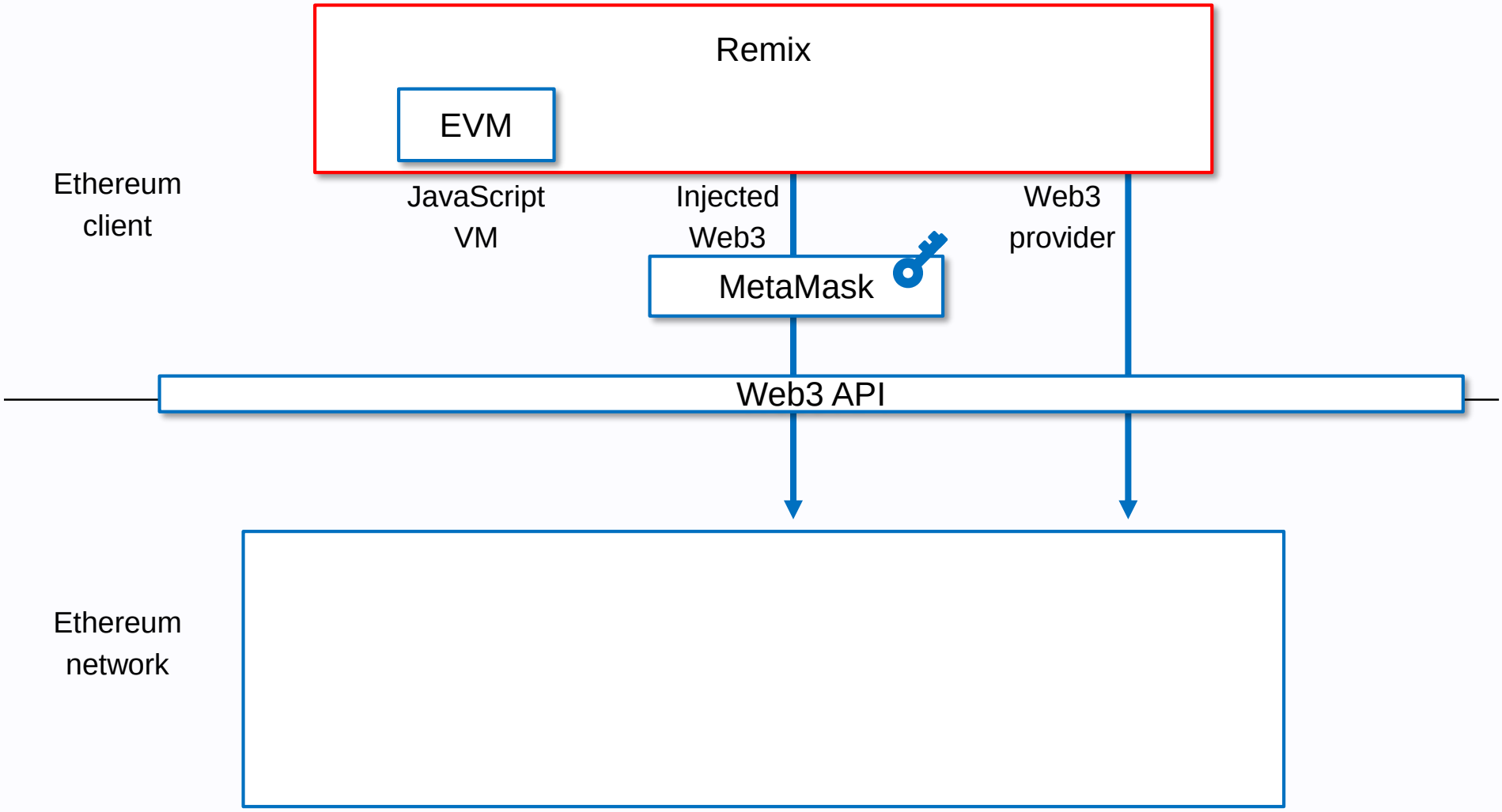

References : [C3]

# Solc



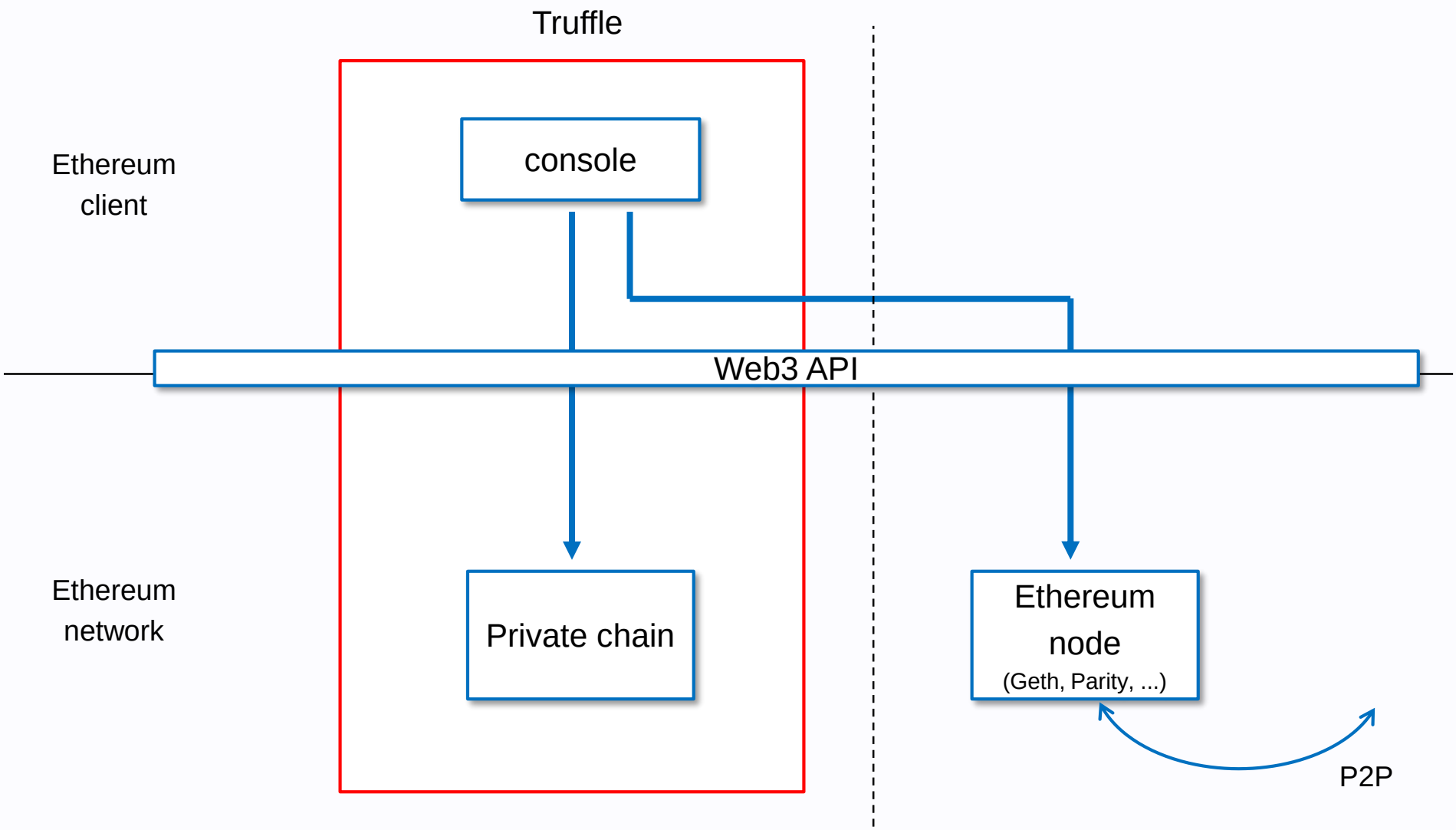

References : [C2]

# 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#6413

# 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