

Ethereum EVM illustrated

exploring some mental models and implementations

Takenobu T.

WIP

Rev. 0.01.1

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

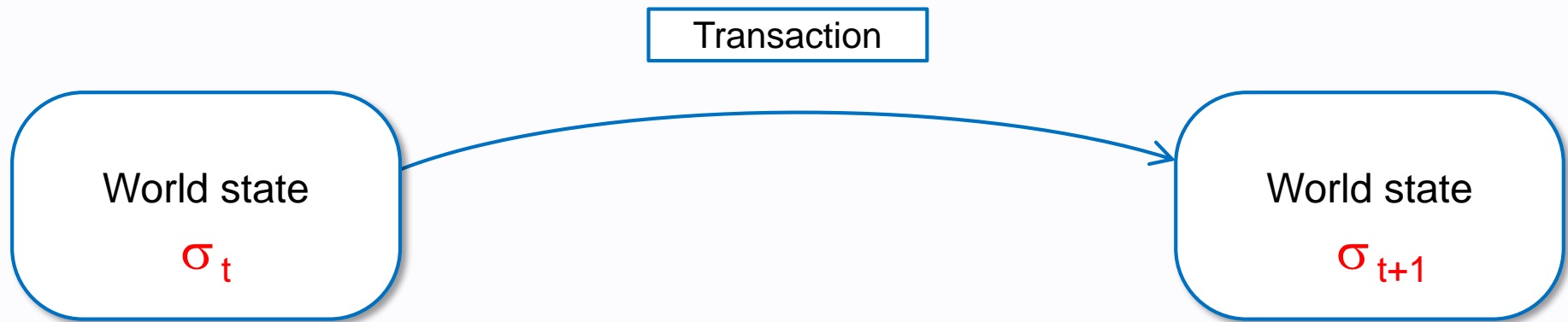
References

1. Introduction

1. Introduction

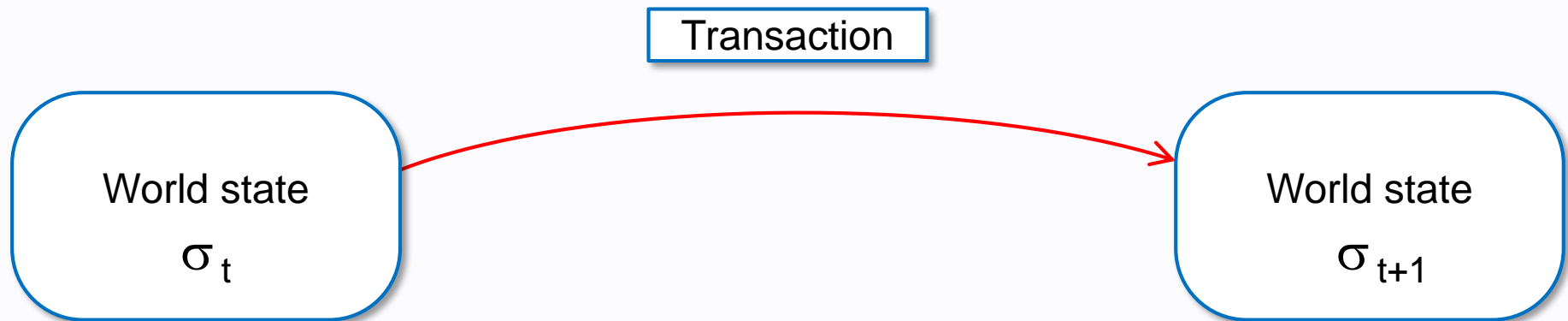
Blockchain

A transaction-based state machine



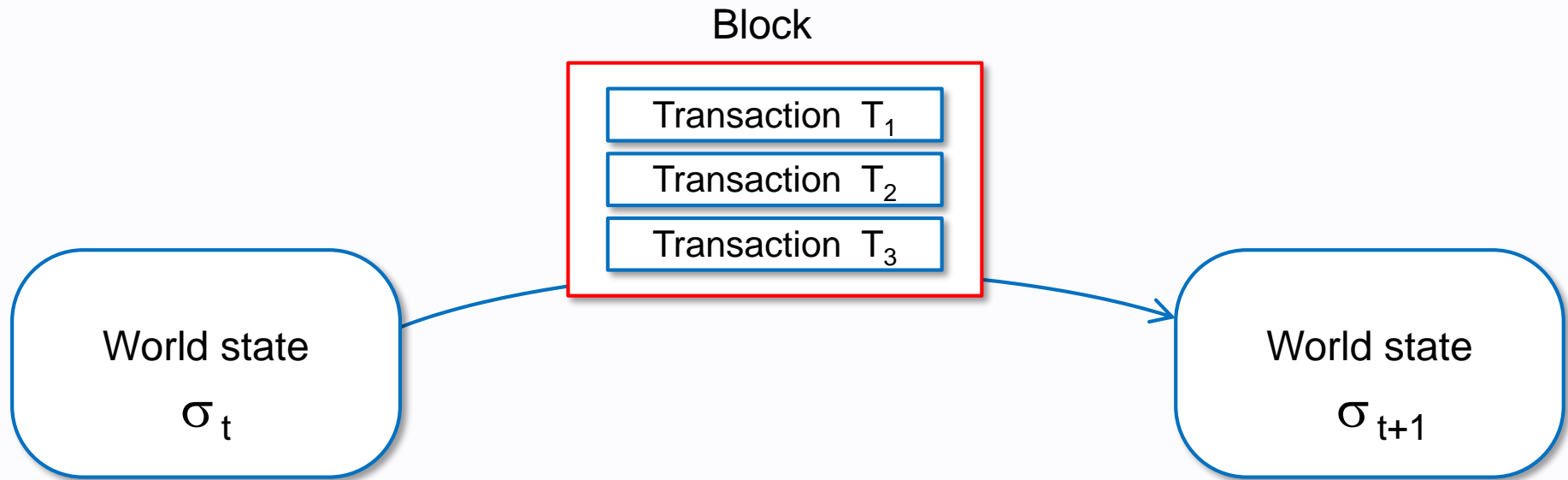
Ethereum can be viewed as a transaction-based state machine.

A transaction-based state machine



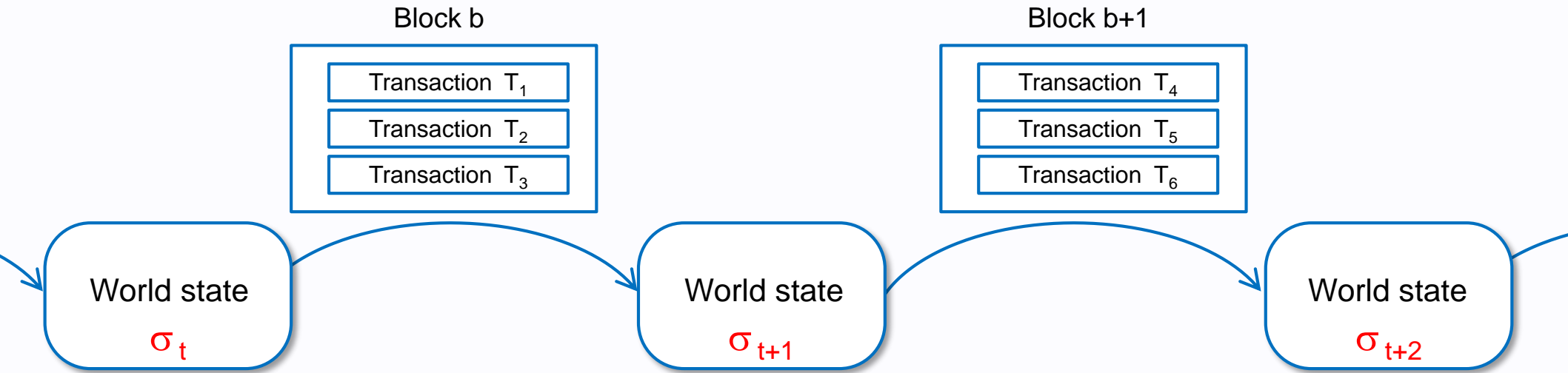
A transaction represents a valid arc between two states.

Block and transactions



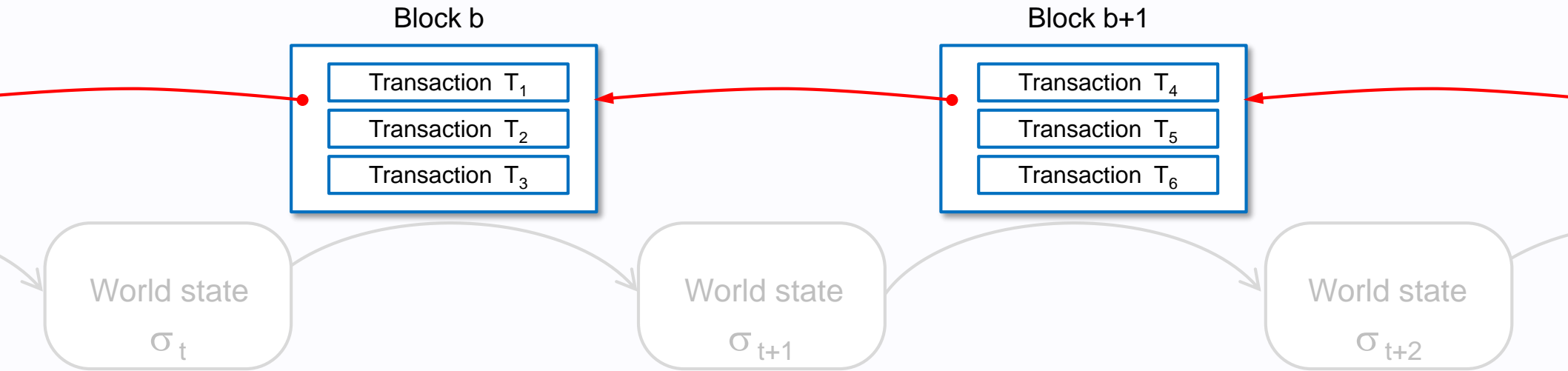
Transactions are collated into blocks.
A block is a package of data.

Chain of states



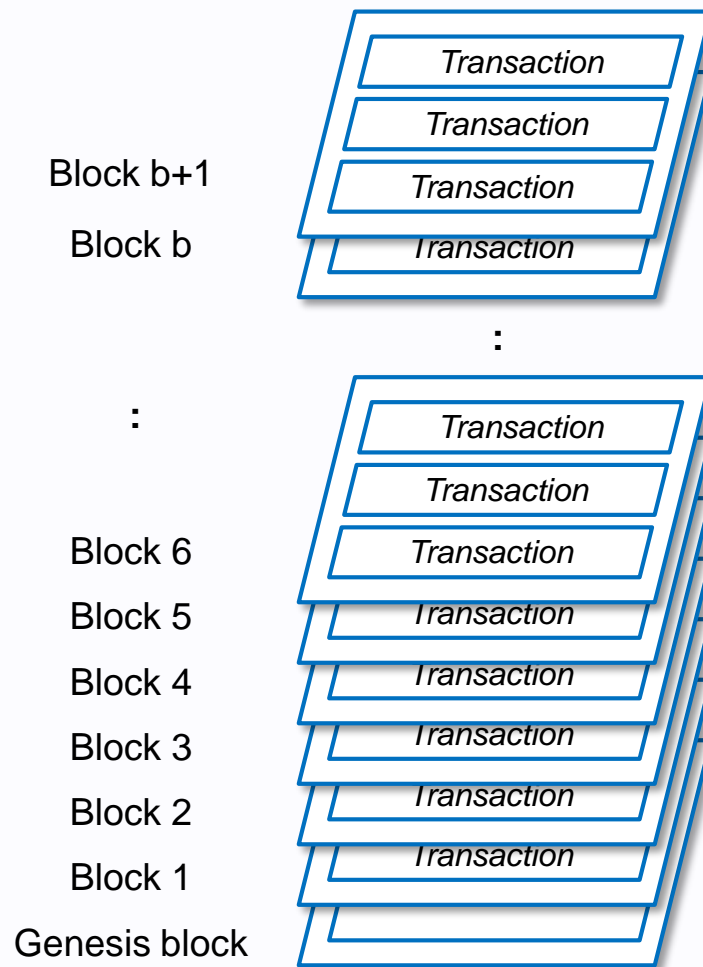
From the viewpoint of the states,
Ethereum can be seen as a state chain.

Chain of blocks: Blockchain



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

Stack of transactions : Ledger



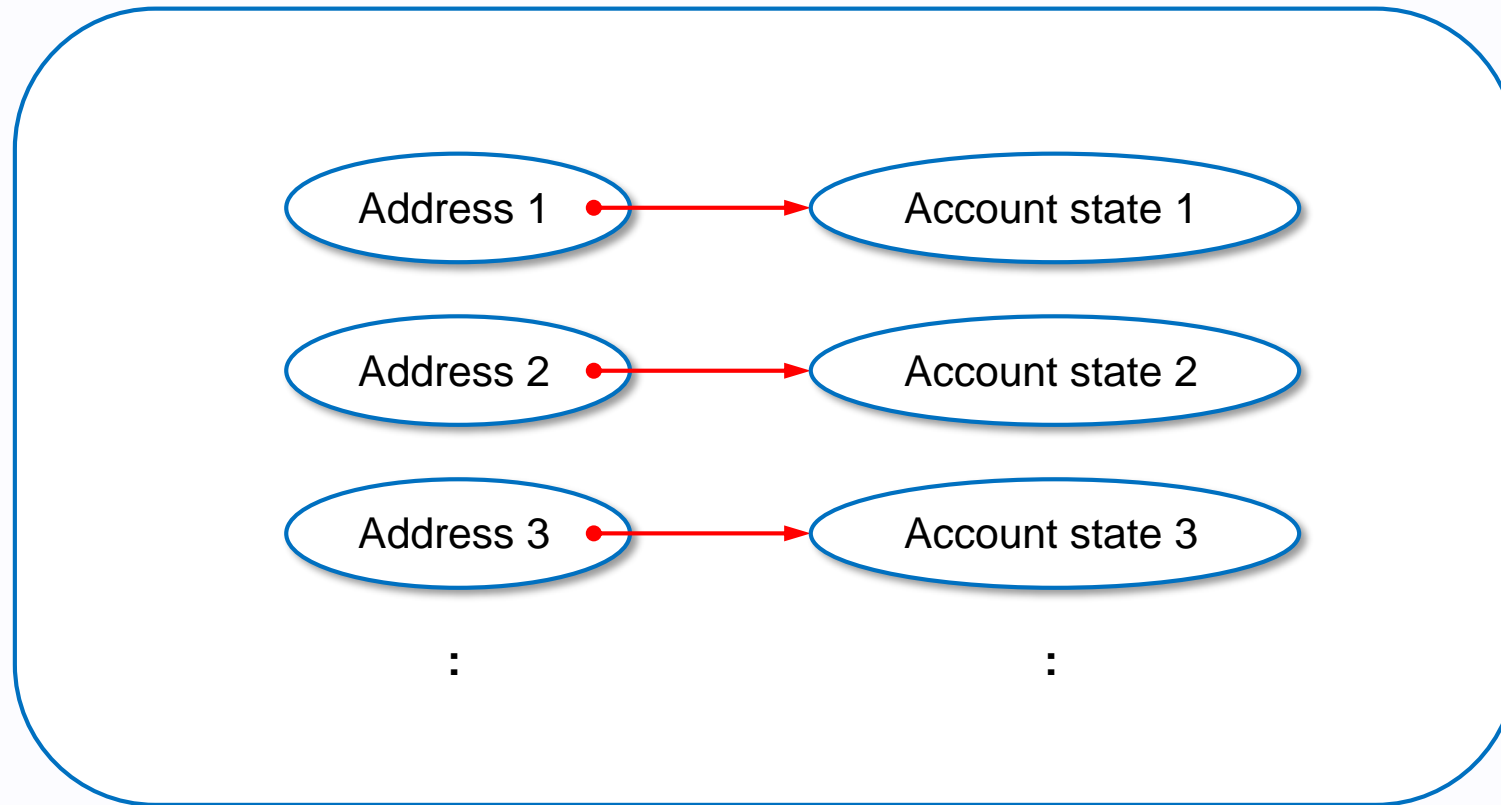
From the viewpoint of the ledger,
Ethereum can also be seen as a stack of transactions.

1. Introduction

World state

World state

World state σ_t



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

Several views of world state

Mapping view

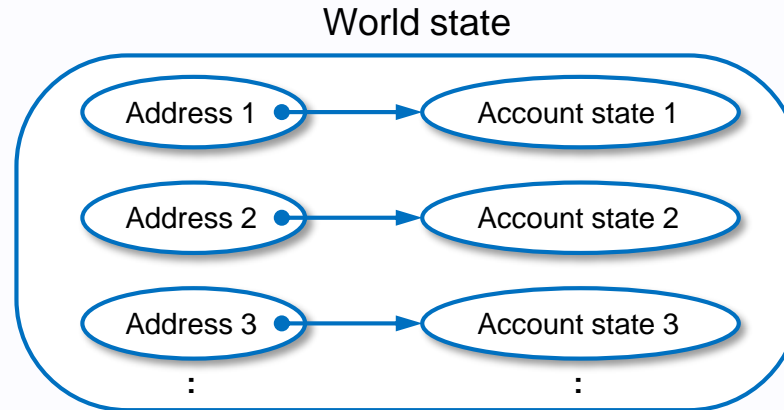
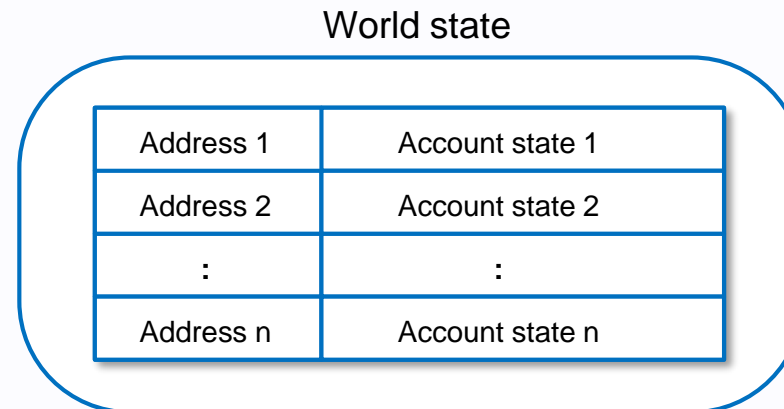
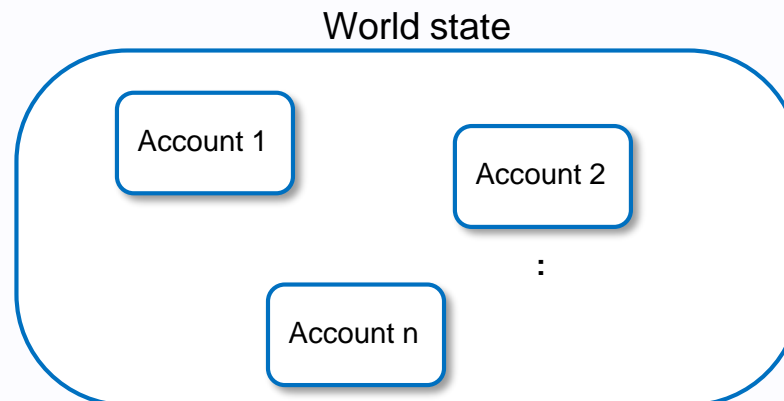


Table view



Object view

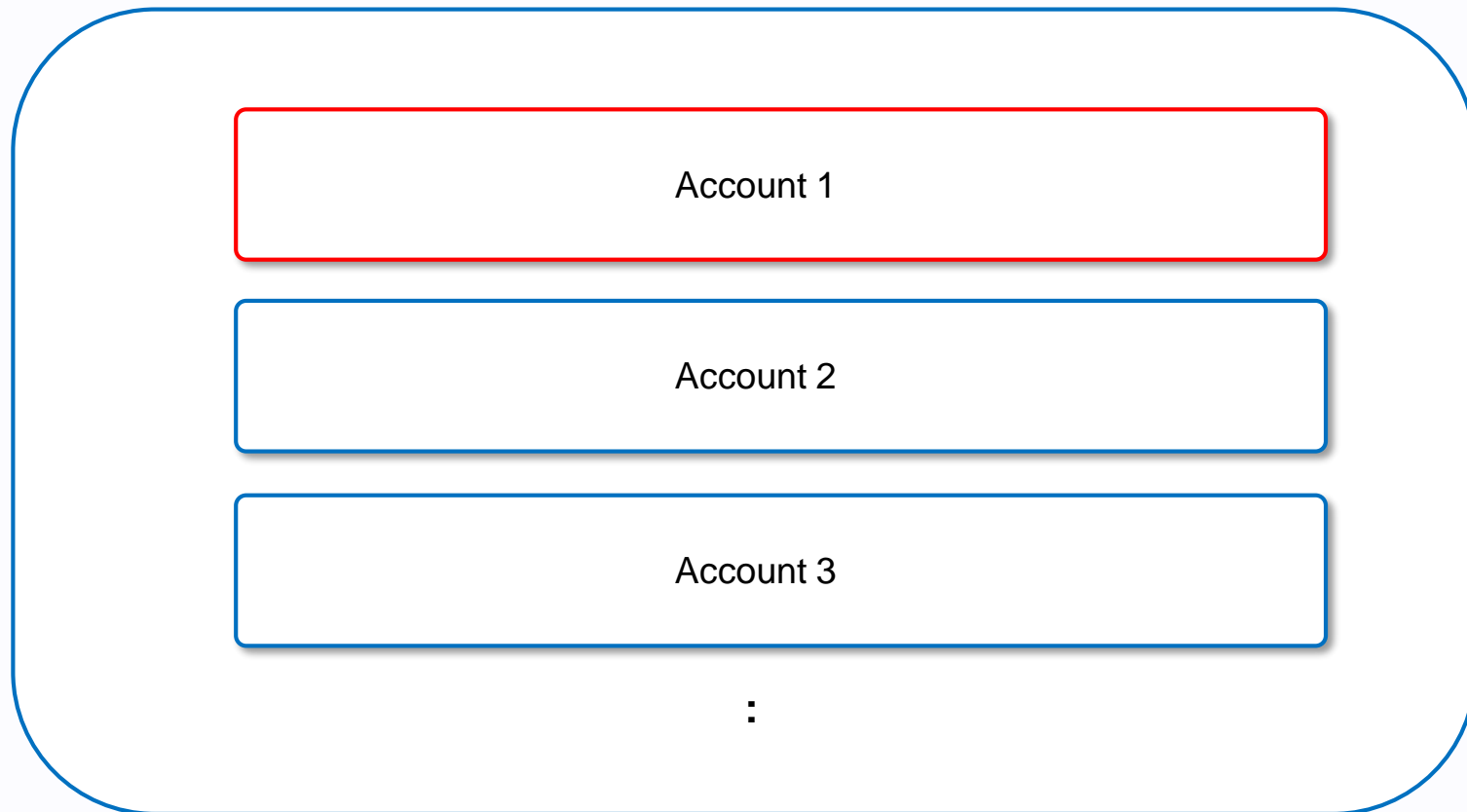


1. Introduction

Account

Account

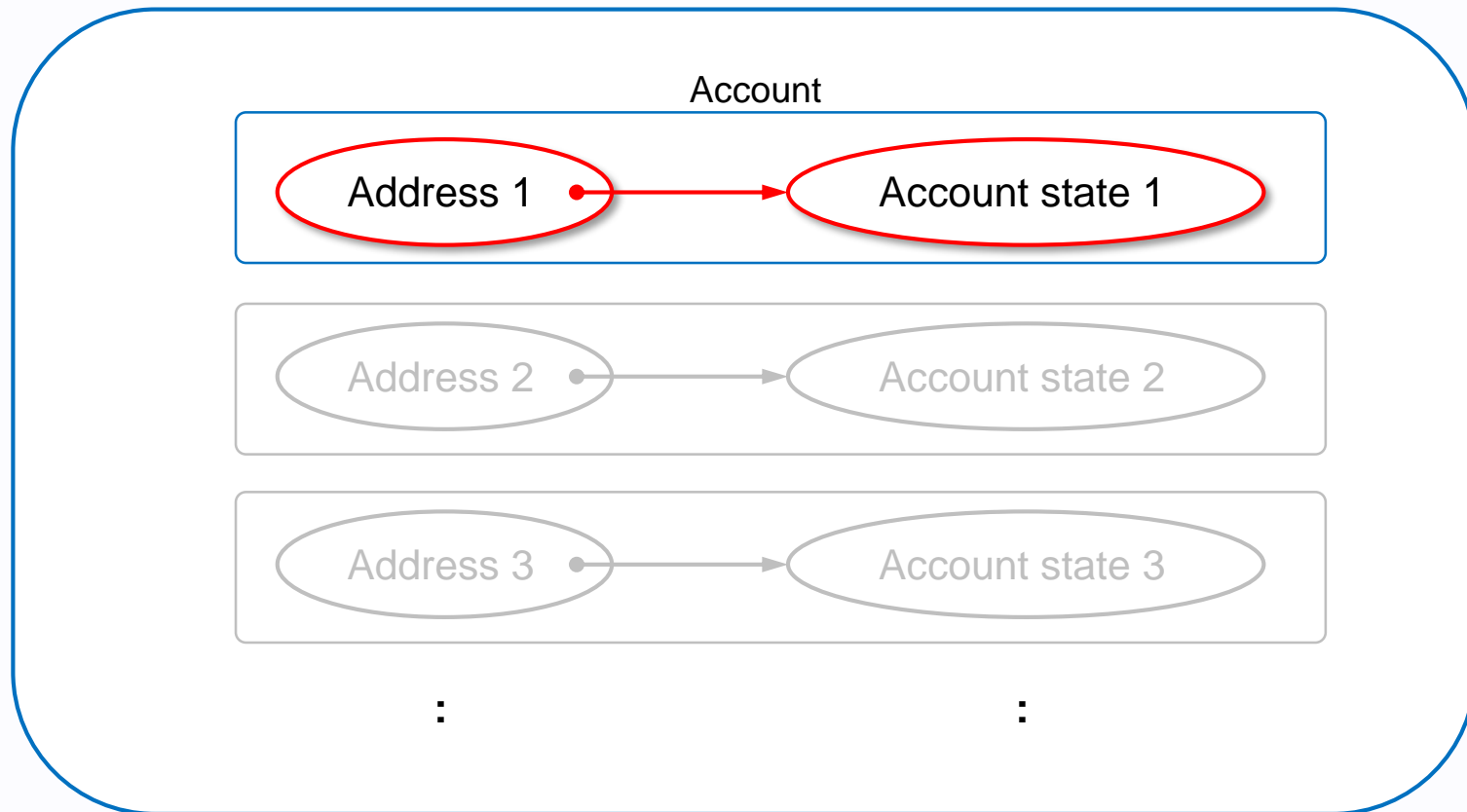
World state



An account is an object in the world state.

Account

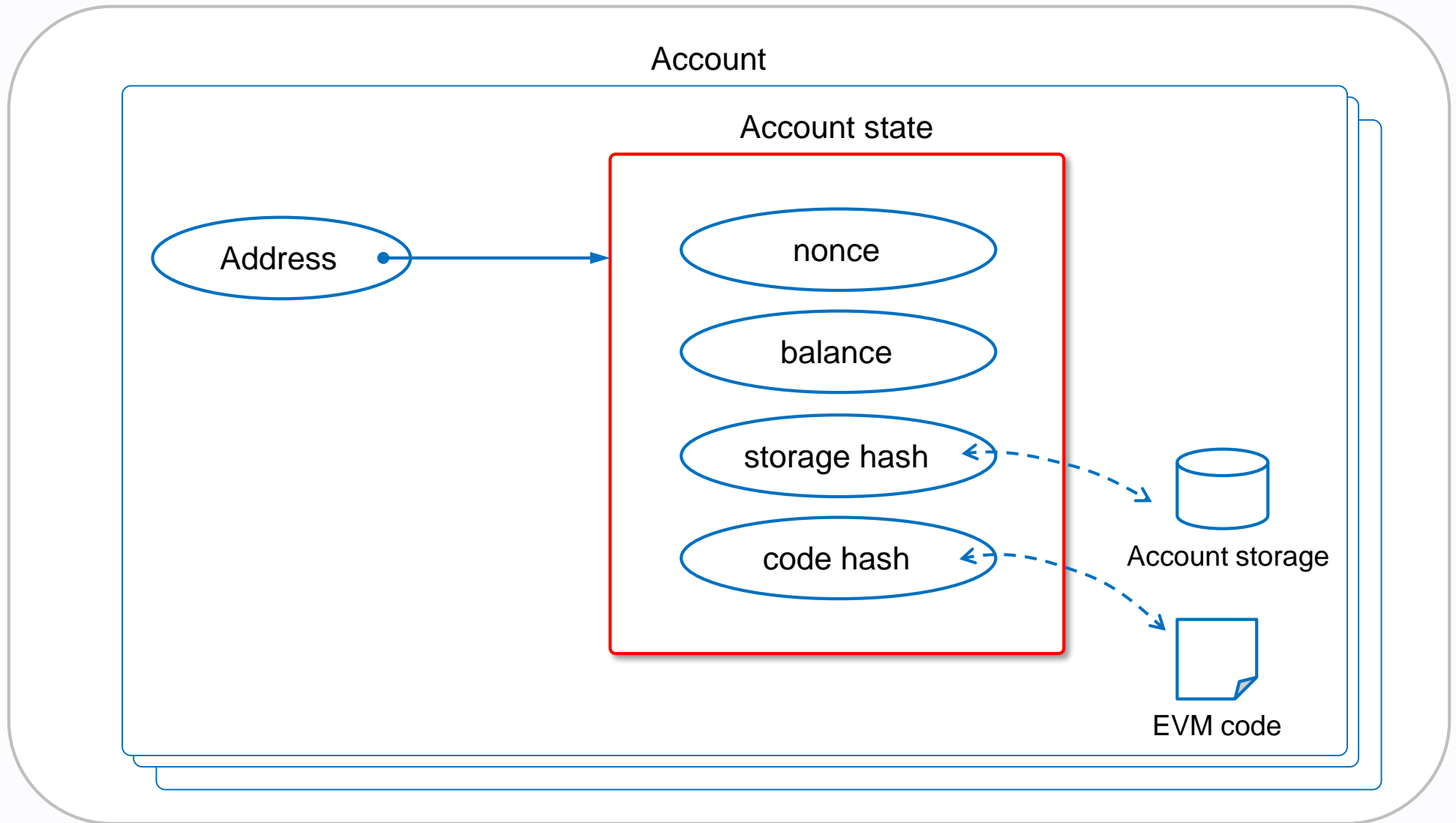
World state



An account is a mapping between address and account state.

Account state

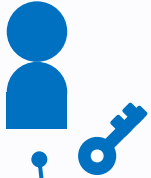
World state



An account state could contain EVM code and storage.

Two practical types of account

External actor

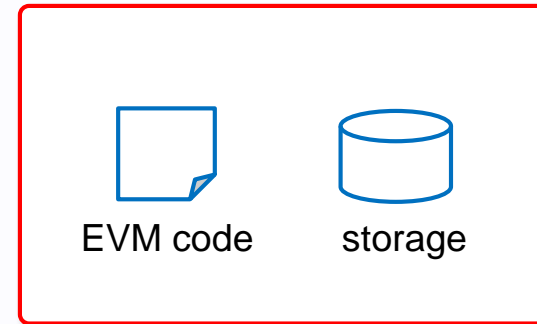


World state

Externally owned account (EOA)



Contract account



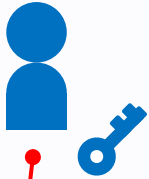
Autonomous object

EOA is controlled by a private key.

Contract account contains EVM code.

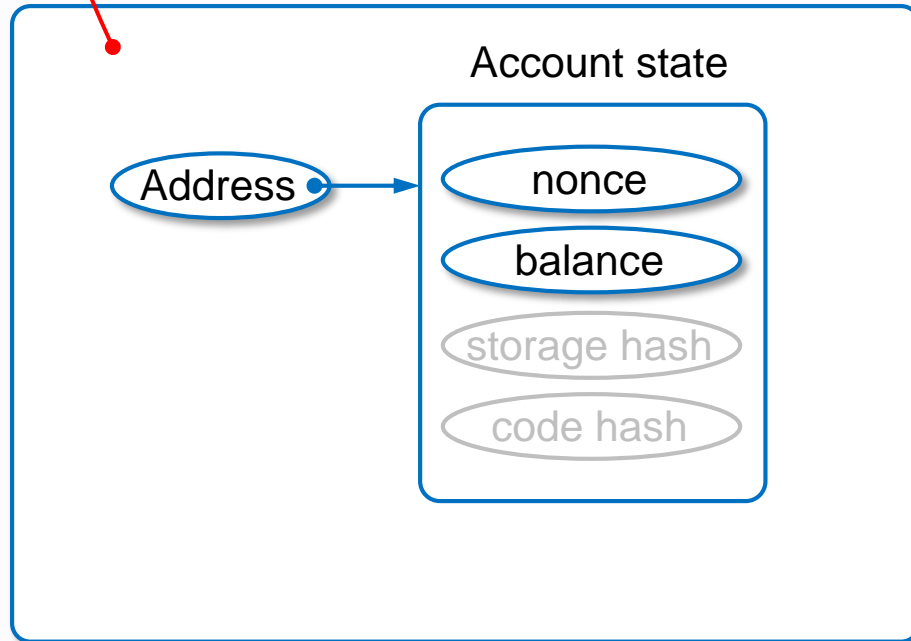
Two practical types of account

External actor

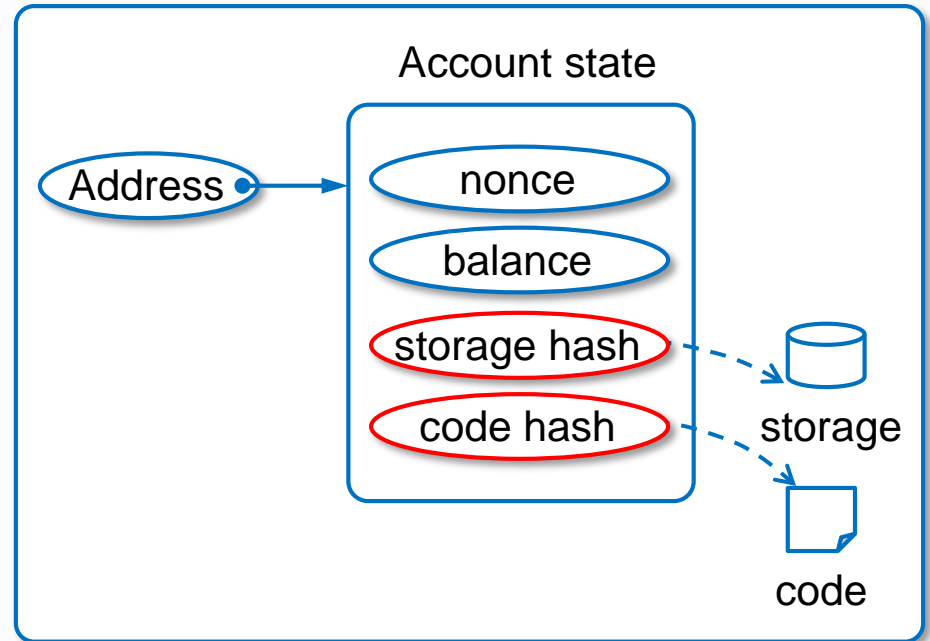


World state

Externally owned account (EOA)



Contract account

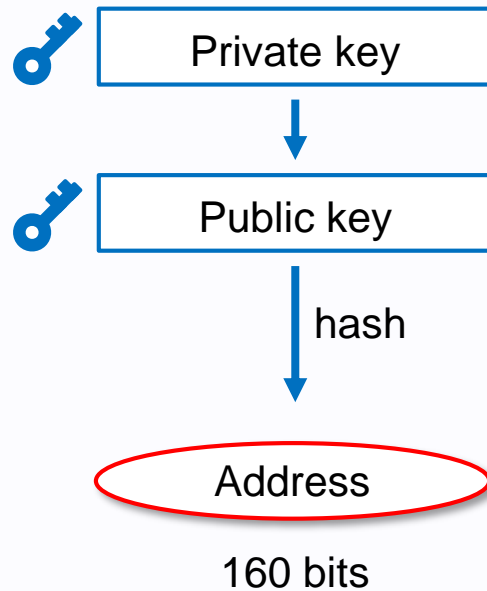


EOA is controlled by a private key.
EOA cannot contain EVM code.

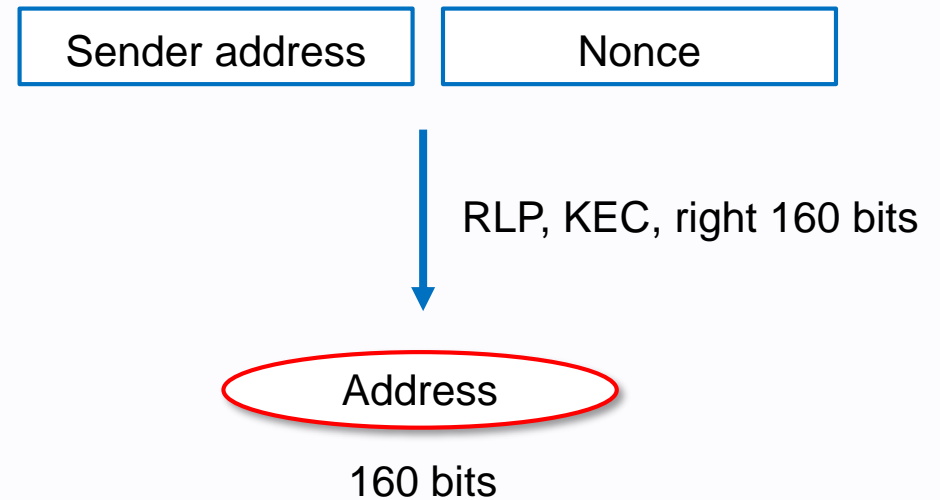
Contract contains EVM code.
Contract is controlled by EVM code.

Address of account

Externally owned account (EOA)



Contract account

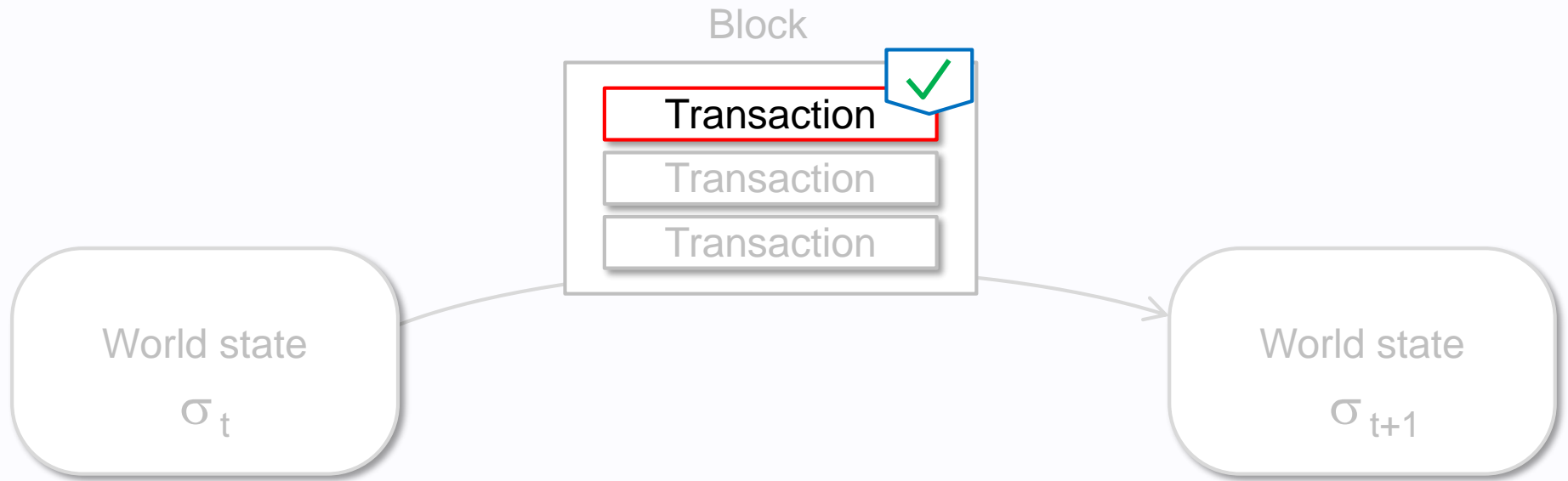


A 160-bit code used for identifying accounts.

1. Introduction

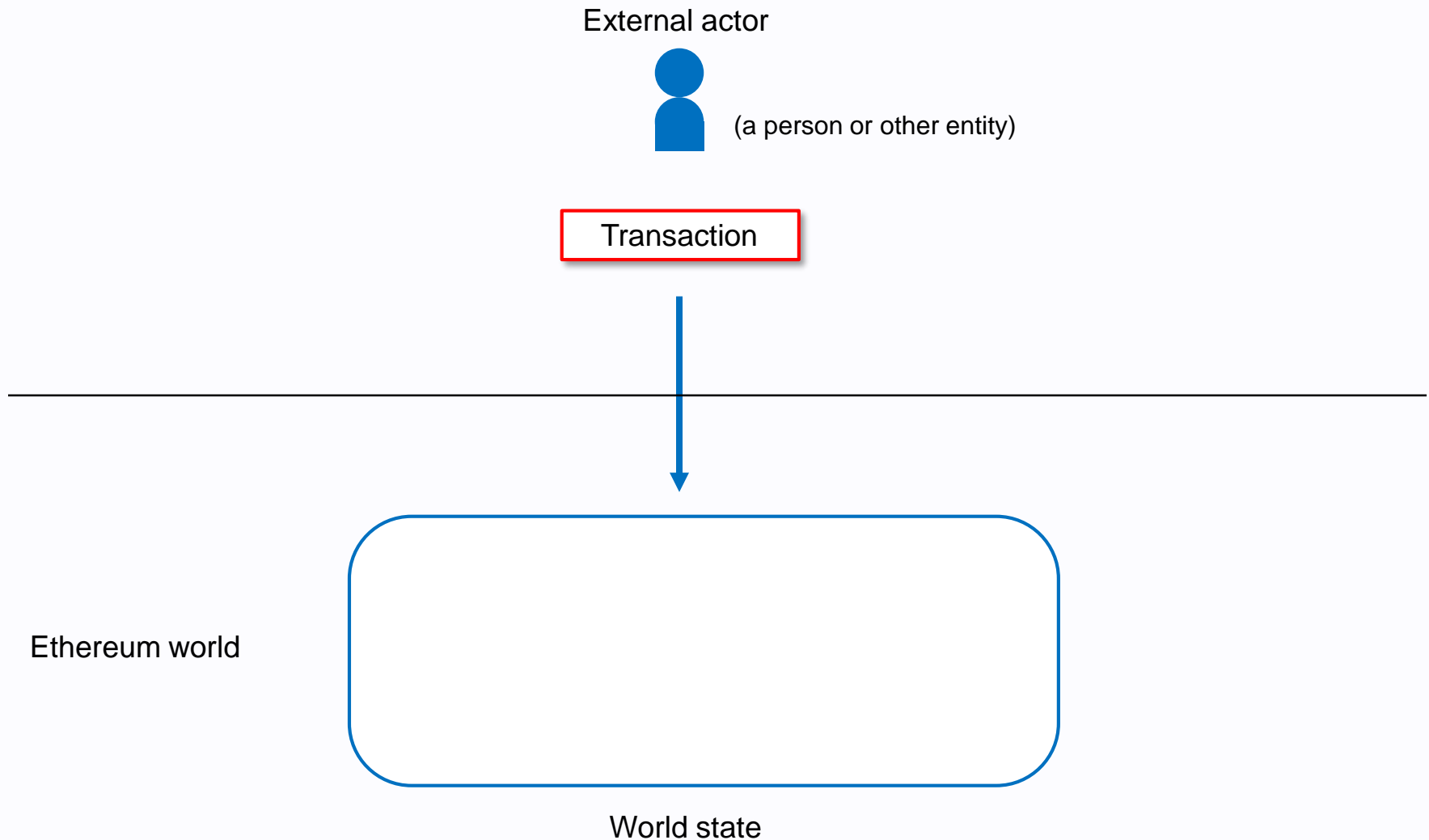
Transaction

A transaction



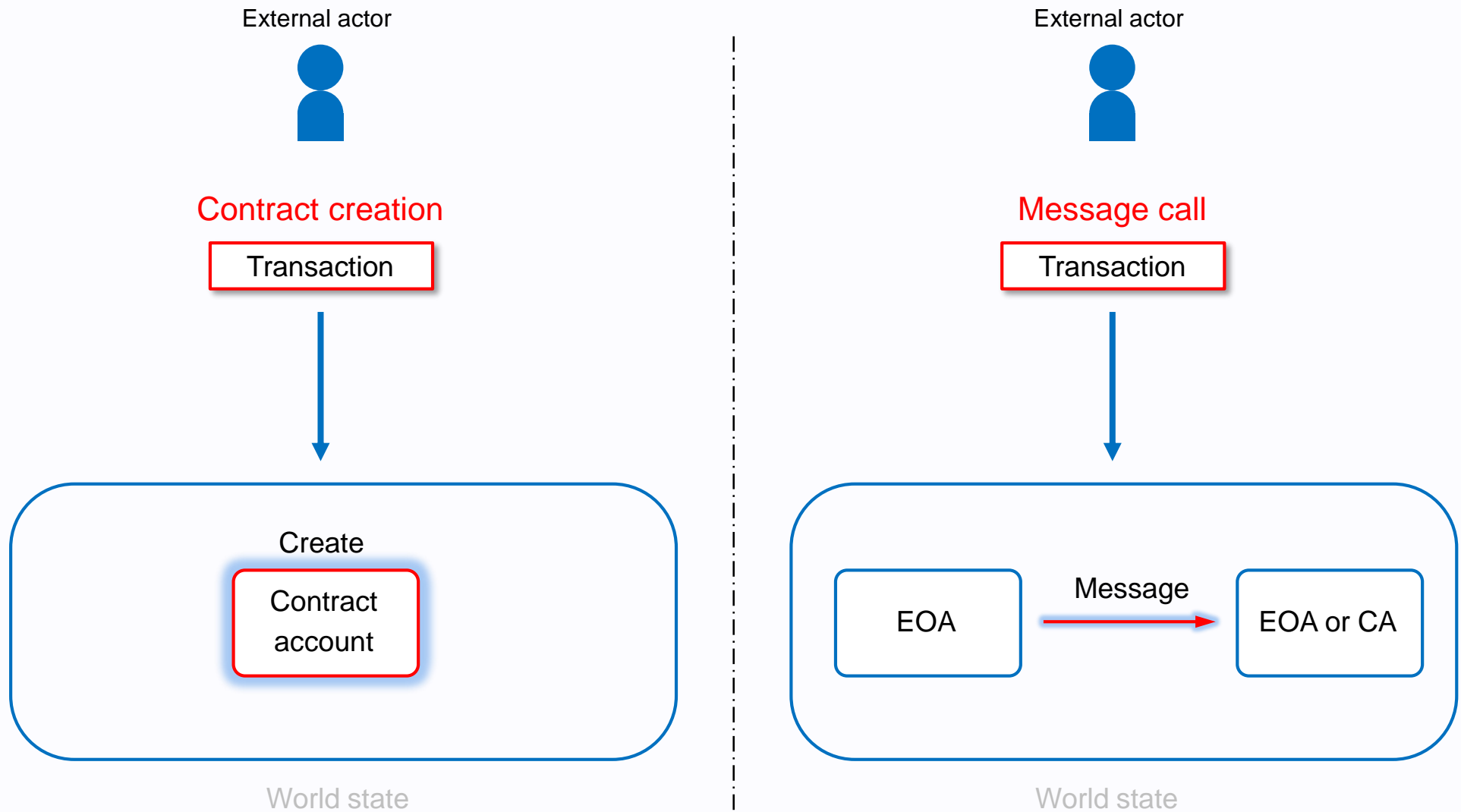
A transaction is a single cryptographically-signed instruction.

A transaction to world state



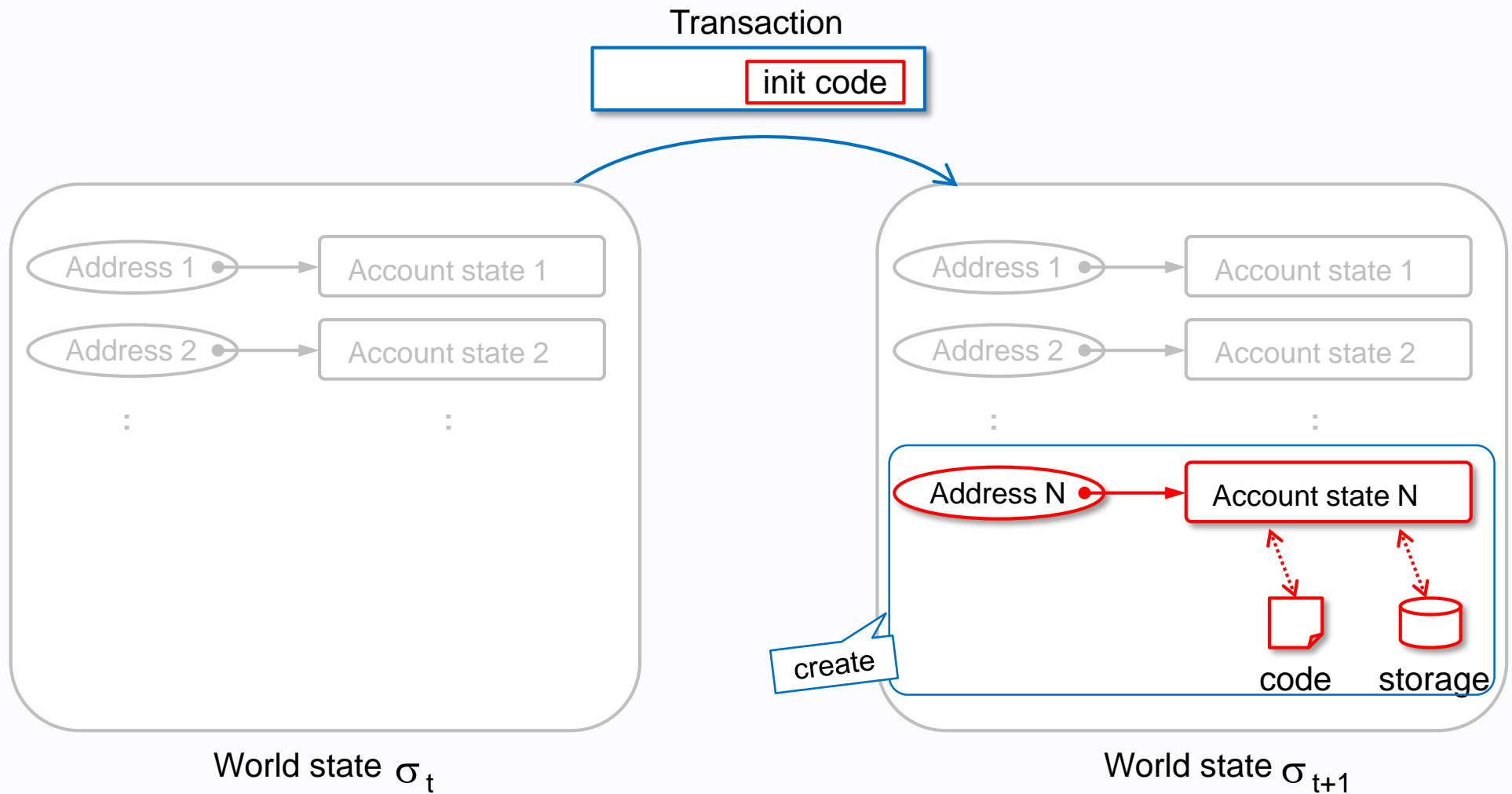
A transaction is submitted by external actor.

Two practical types of transaction

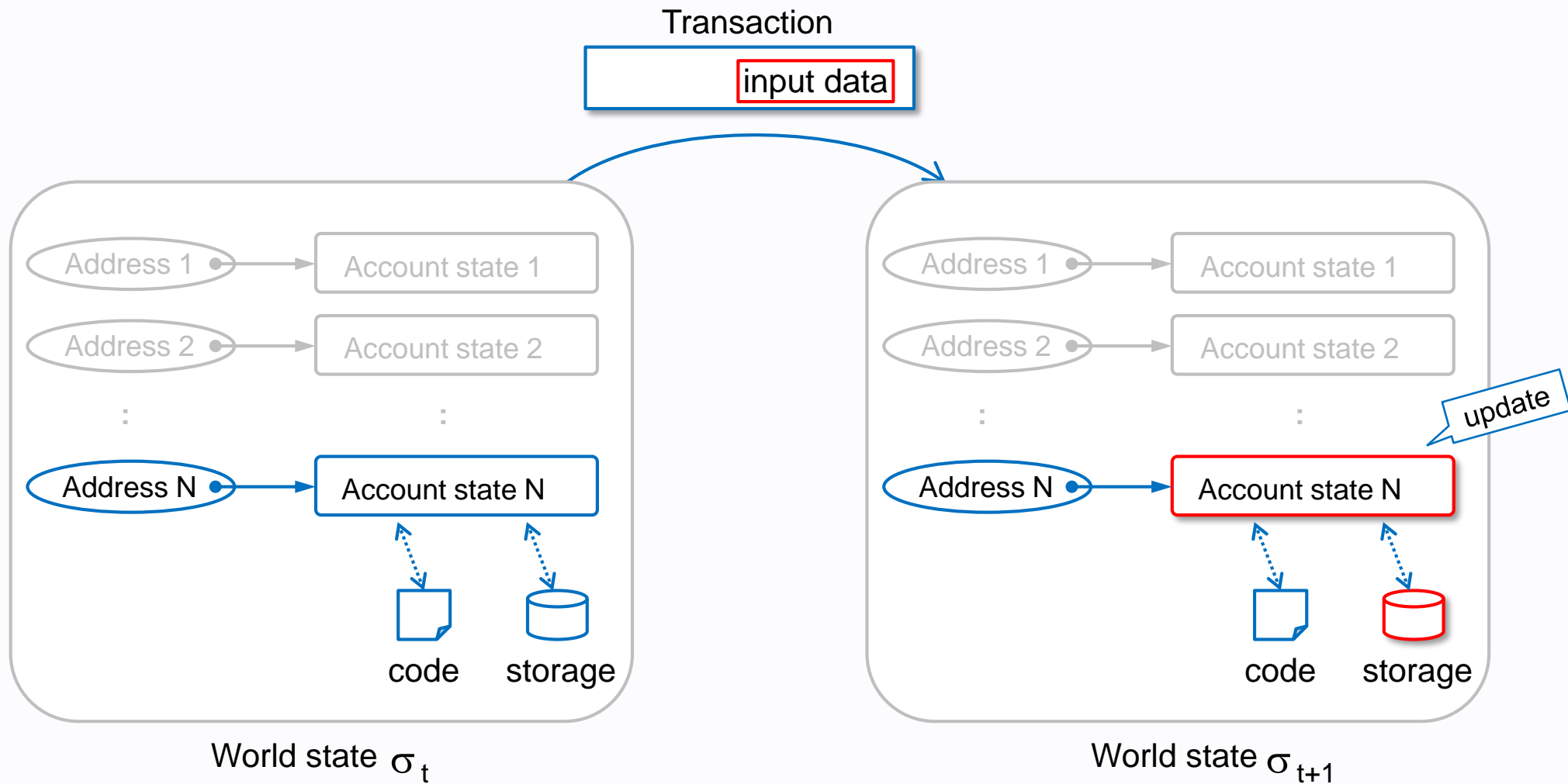


There are two practical types of transaction, contract creation and message call.

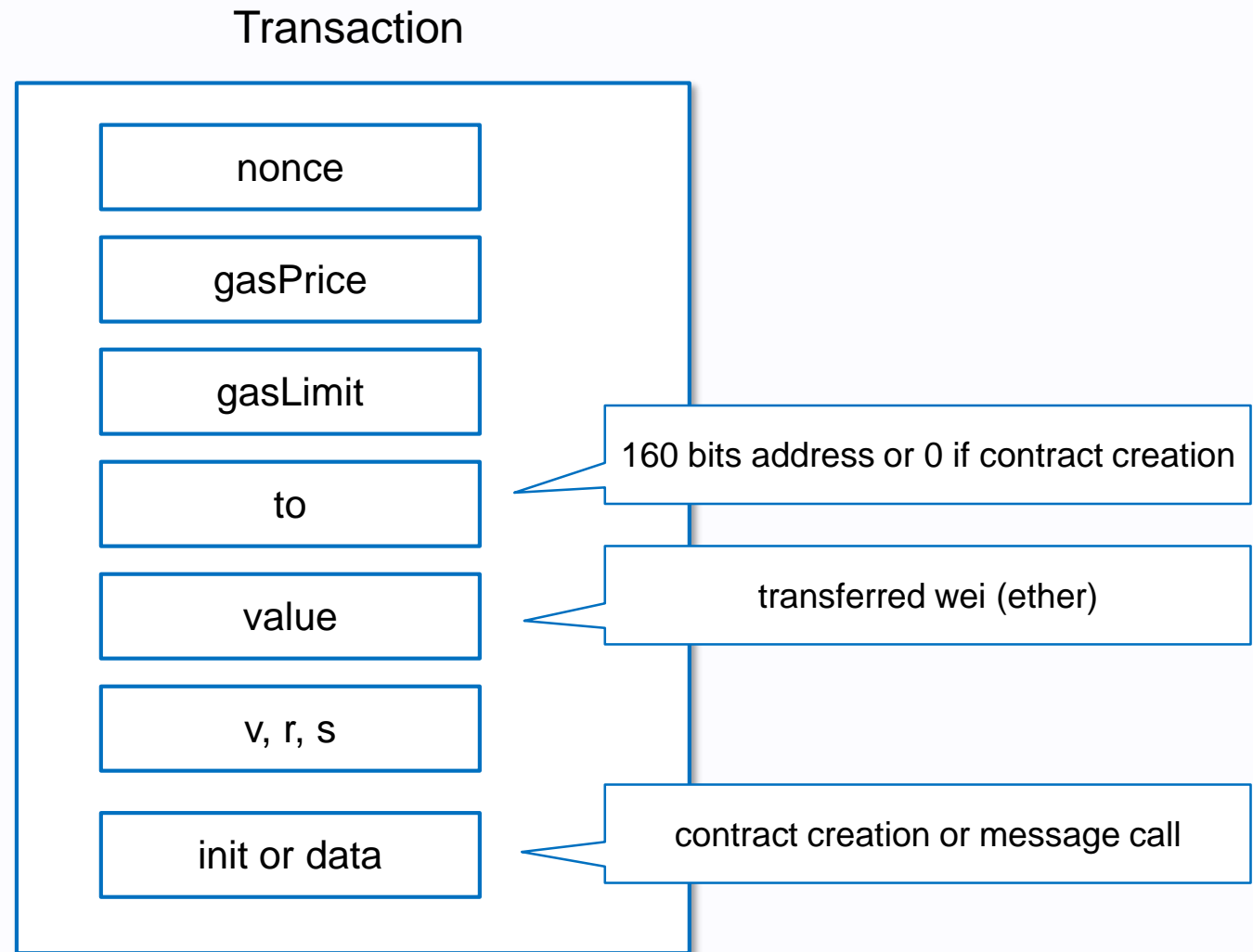
Contract creation



Message call



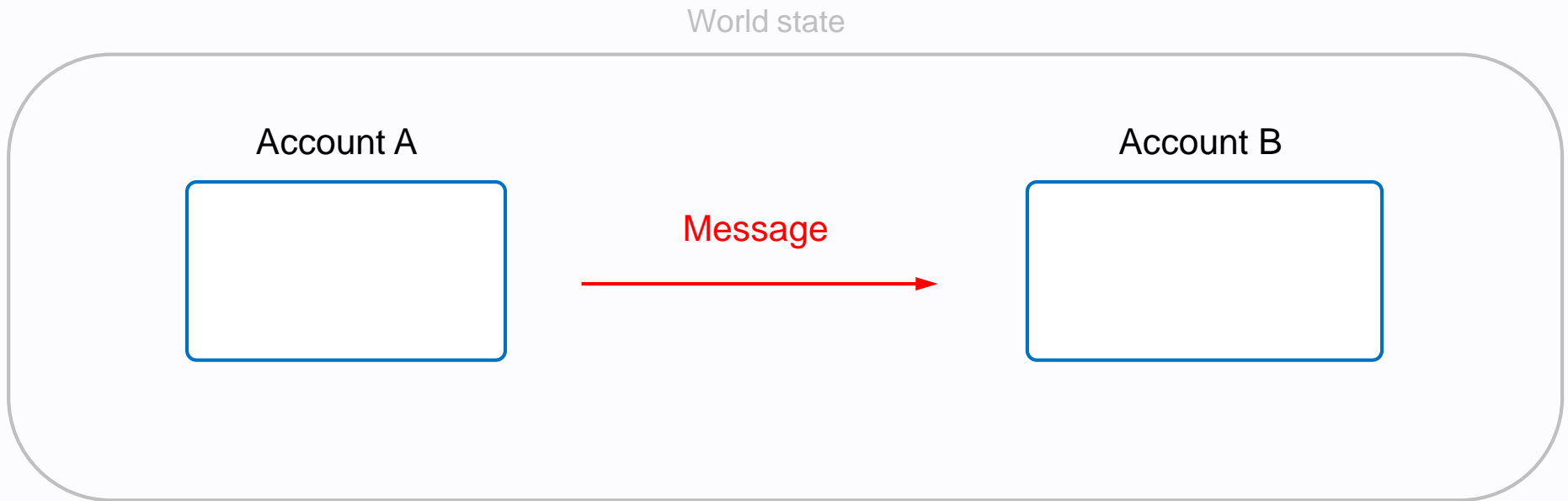
Field of a transaction



1. Introduction

Message

Message

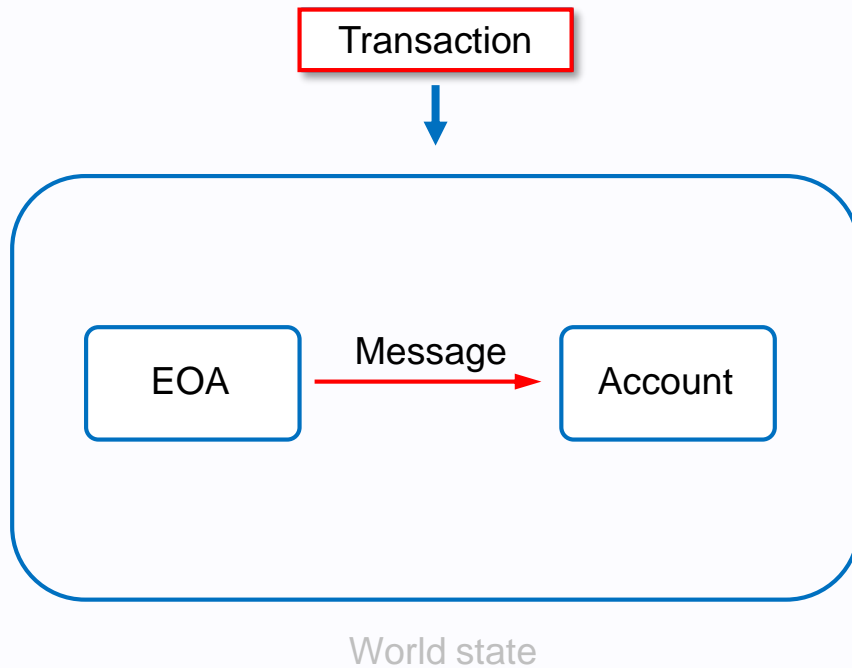


Message is passed between two Accounts.

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

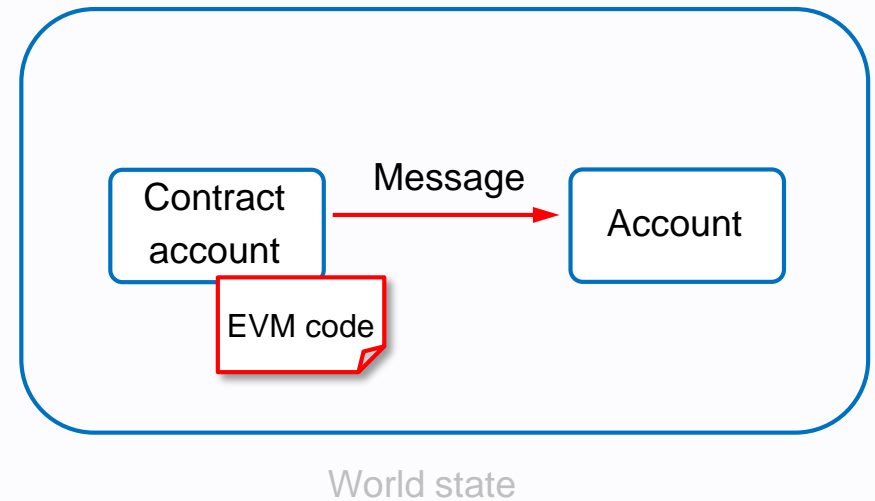
Message

Triggered by transaction



Transaction triggers an associated message.

Triggered by EVM code

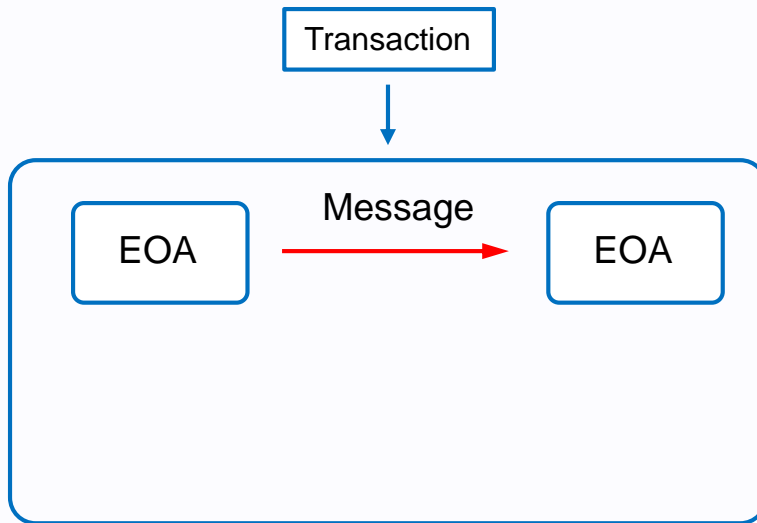


EVM can also send a message.

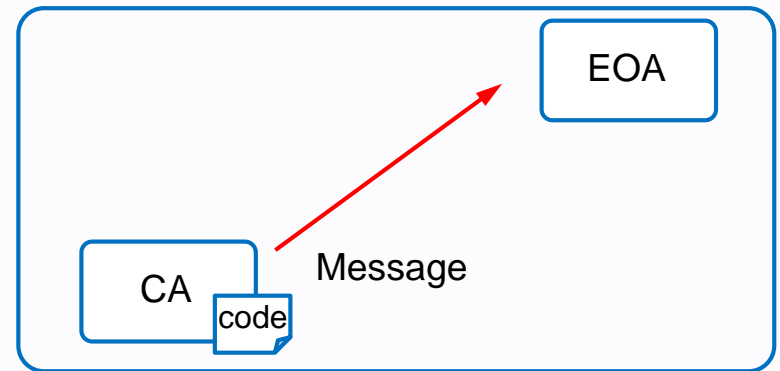
Four cases of message

By Transaction From EOA

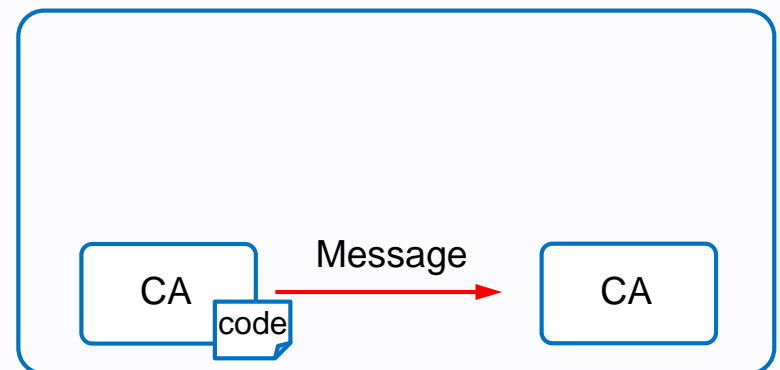
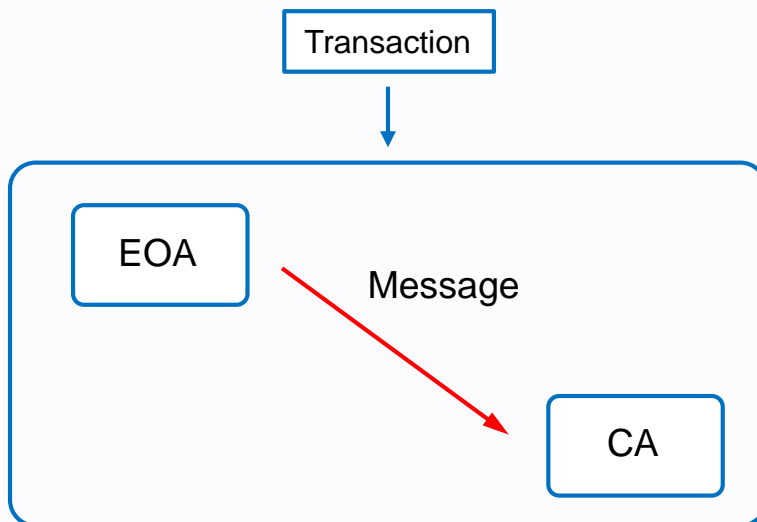
To EOA



By EVM code From CA



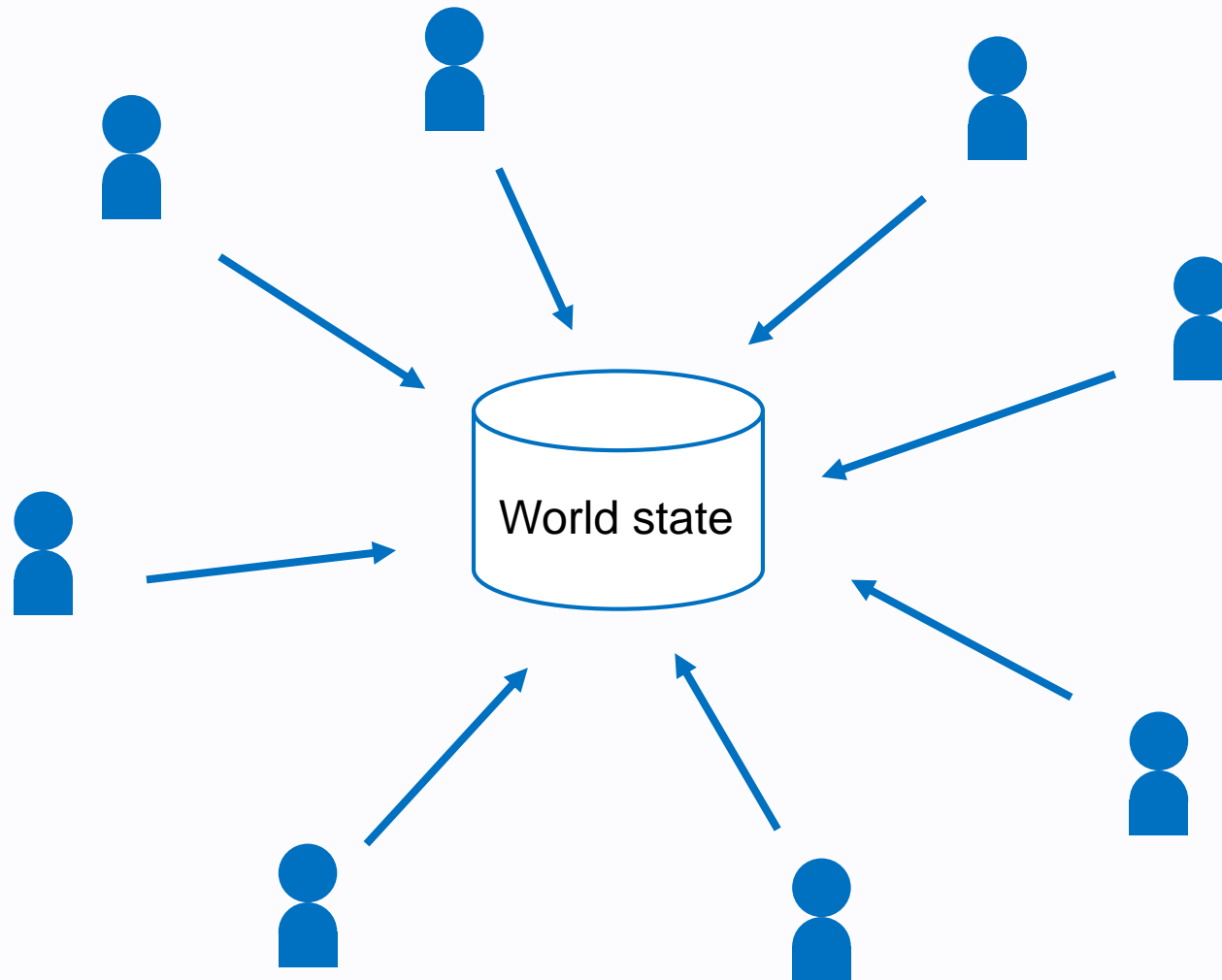
To CA



1. Introduction

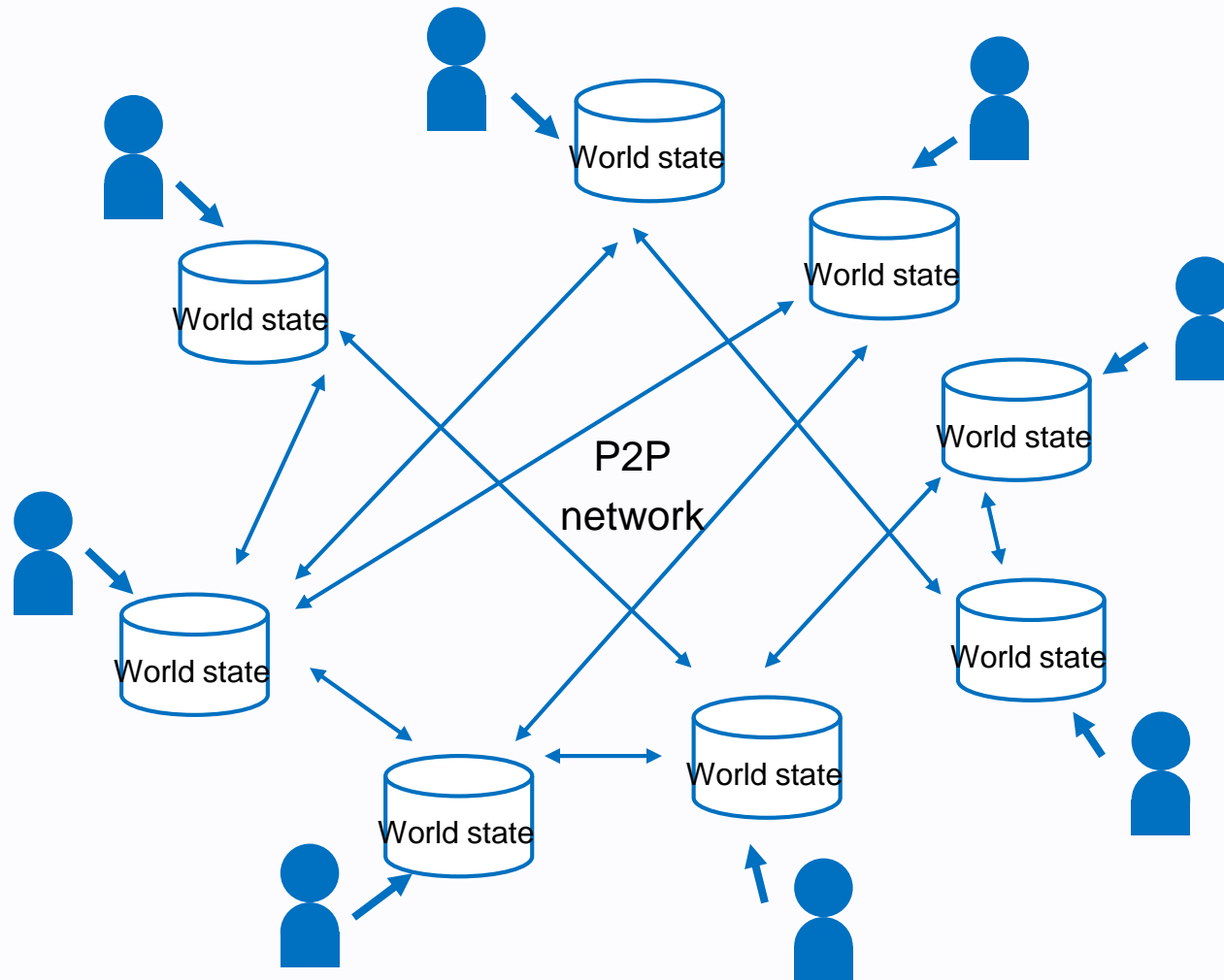
Decentralised database

Globally shared, transactional database



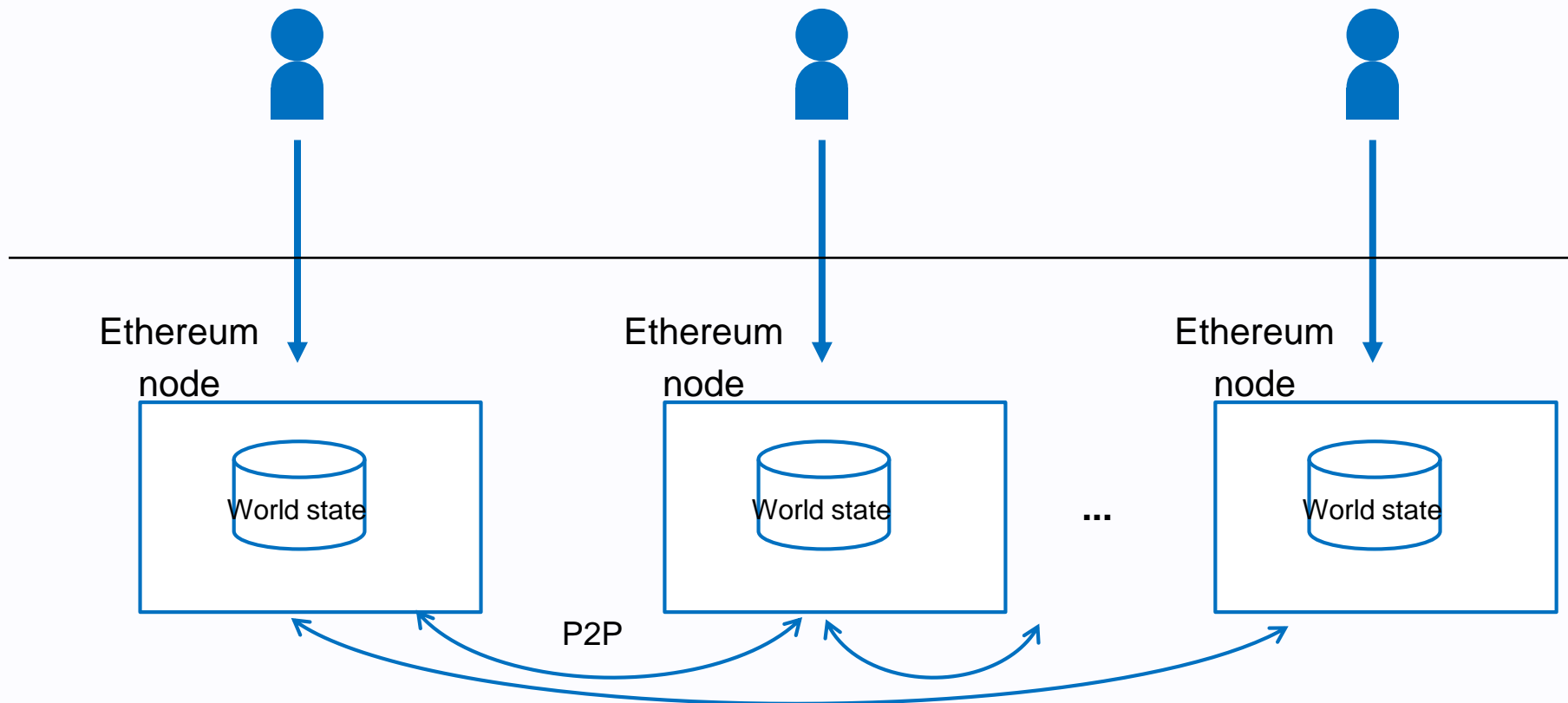
A blockchain is a globally shared, transactional database.

Decentralised database



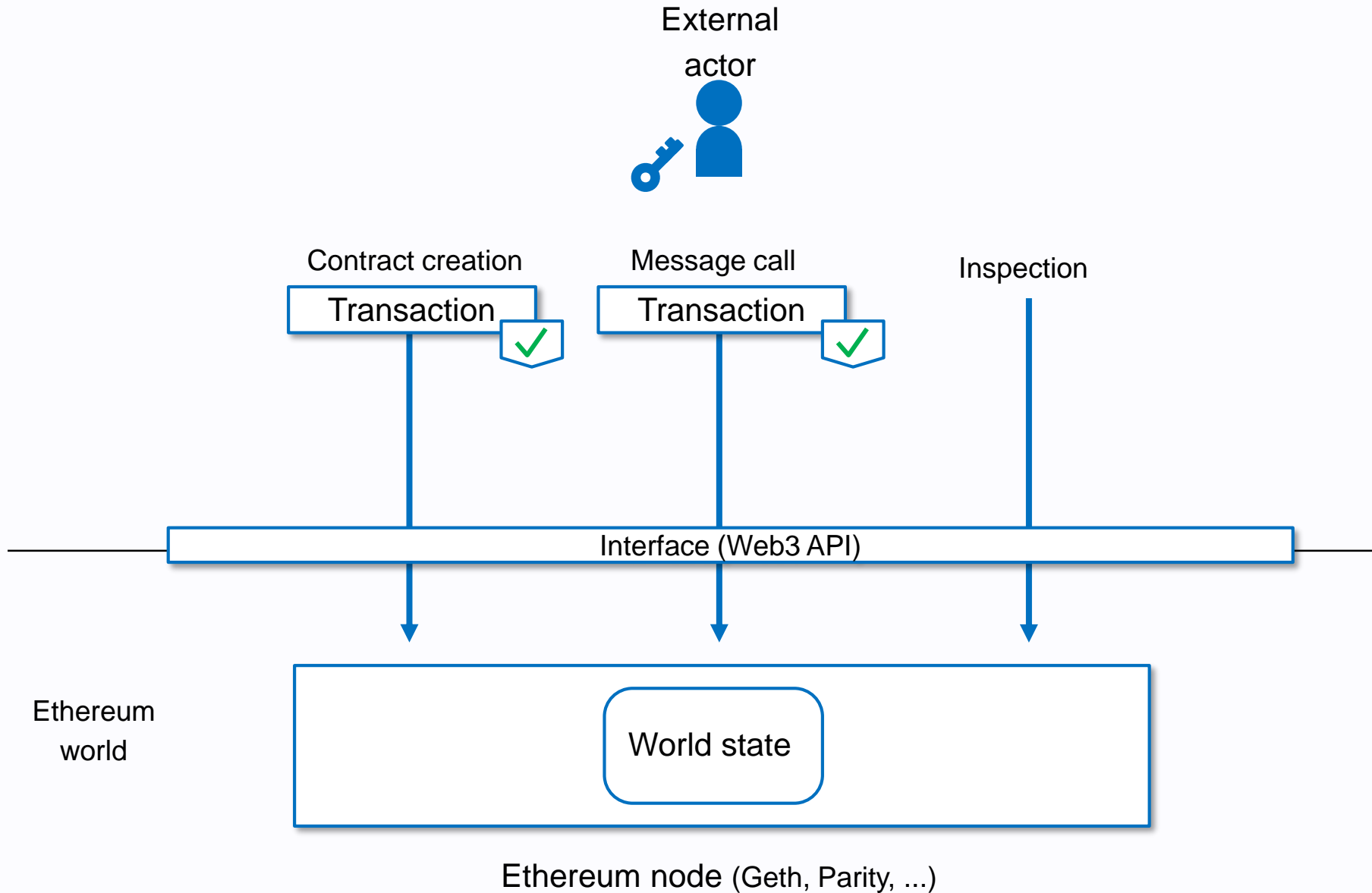
A blockchain is a globally shared, **decentralised**, transactional database.

P2P network inter nodes



Decentralised nodes constitute Ethereum P2P network.

Interface to a node

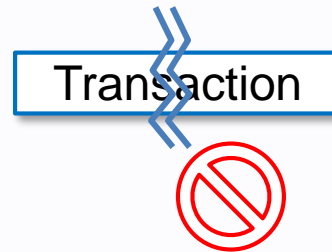


External actors access the Ethereum world through Ethereum nodes.

1. Introduction

Atomicity and order

Atomicity of transaction

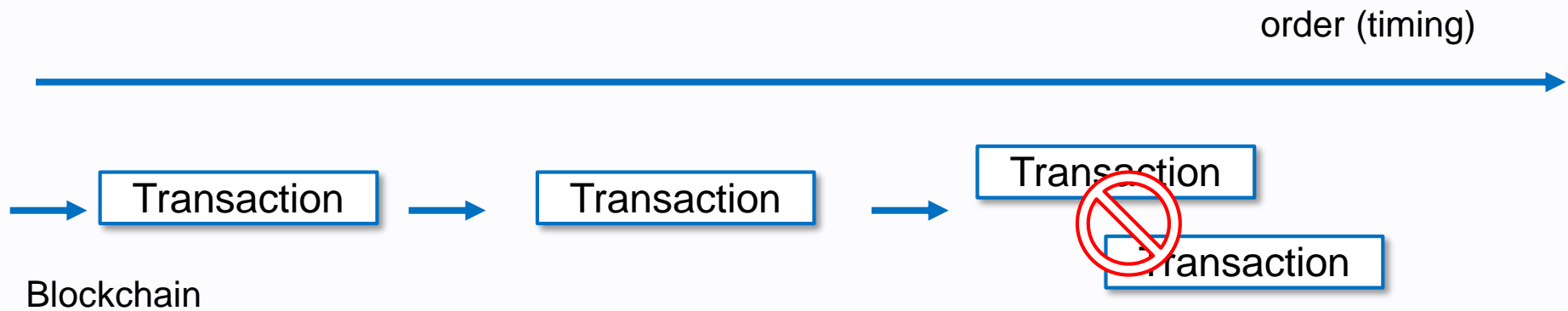


A transaction is **an atomic operation**. Can't divide or interrupt.

Transaction or **Transaction**

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

Order of transactions



Transactions **cannot be overlapped**.

Transactions must be executed sequentially.

Order of transactions

External actor A



3rd submitted

Transaction

External actor B



1st submitted

Transaction

2nd submitted

Transaction

order (timing)

???

???

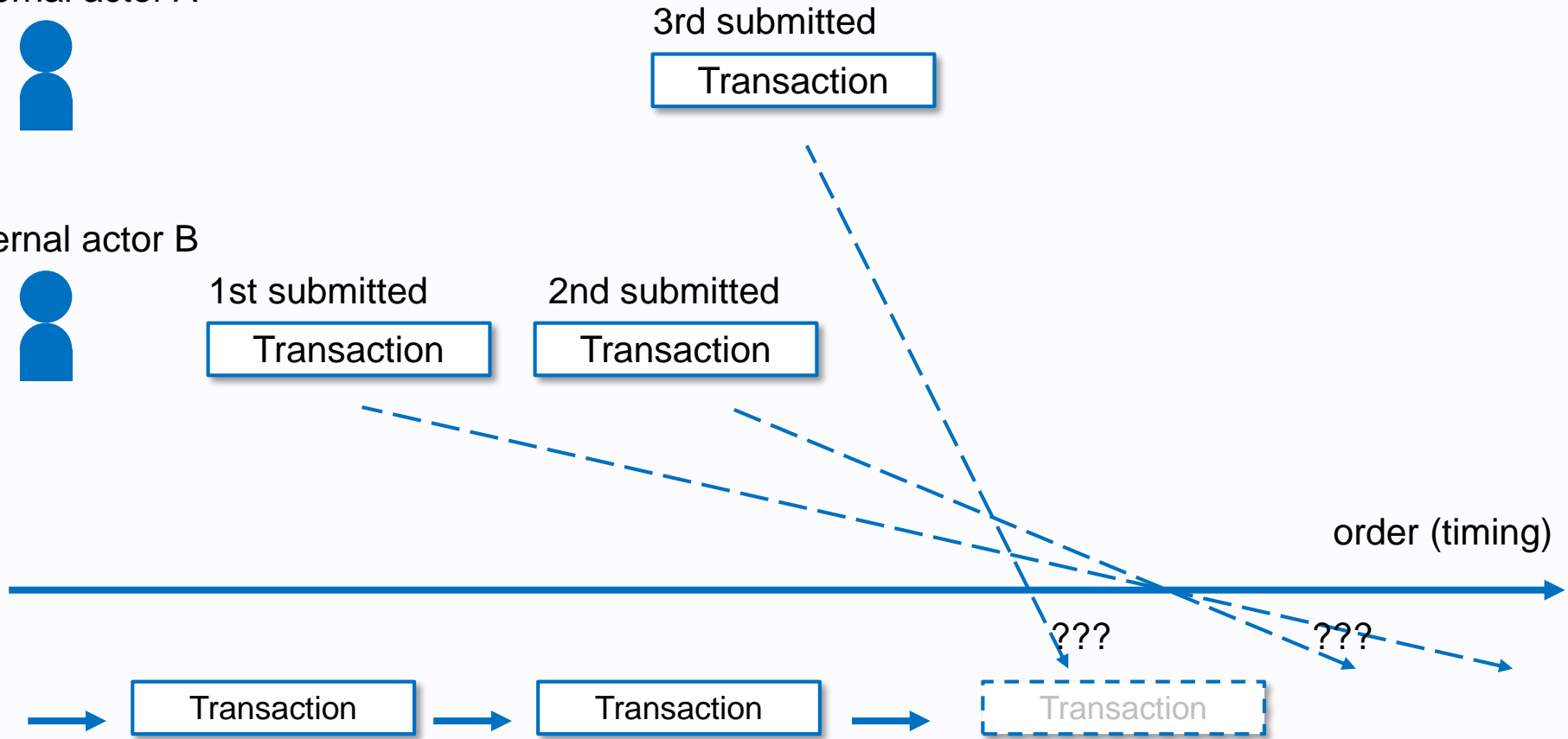
Transaction

Transaction

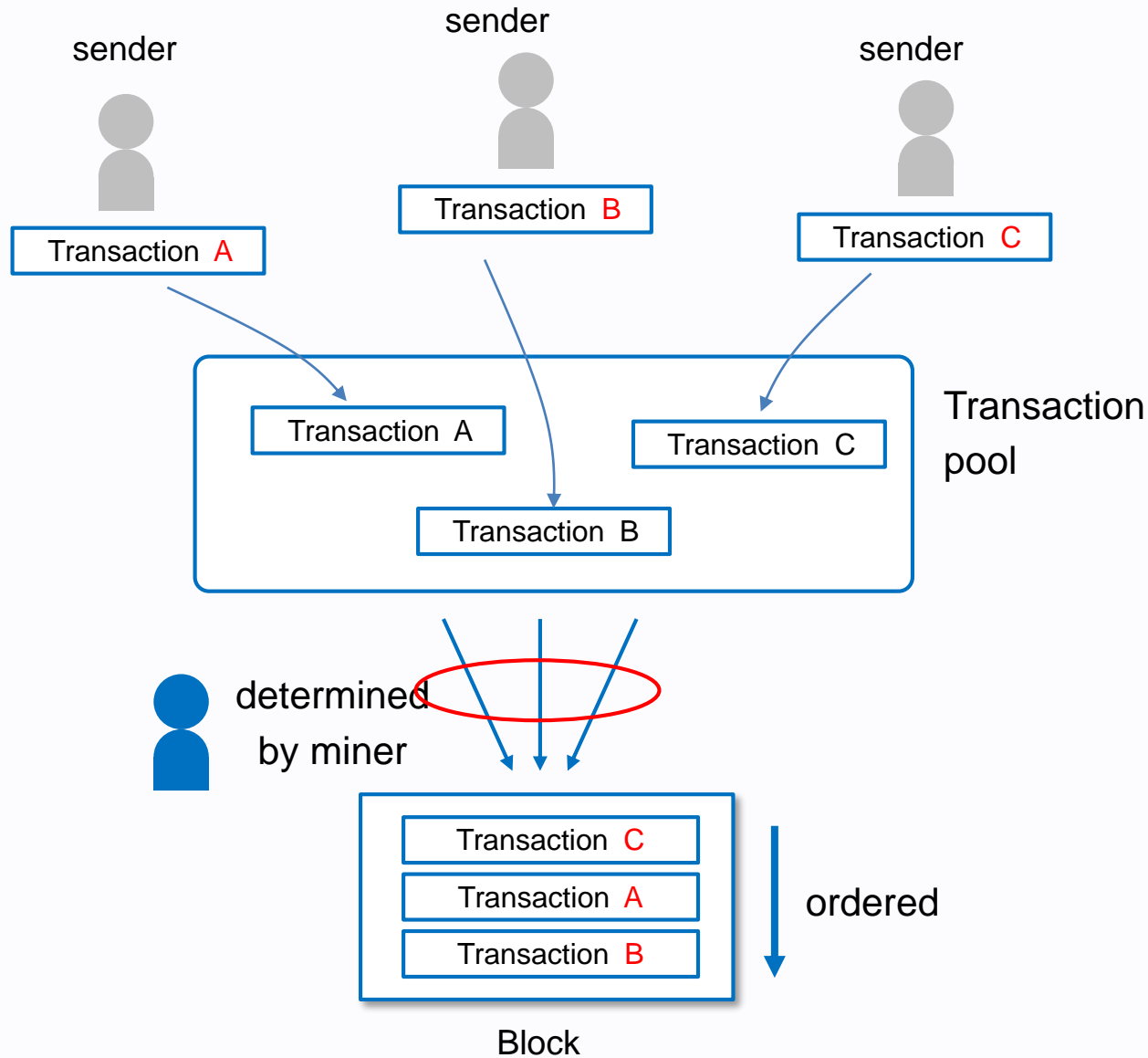
Transaction

Blockchain

Transaction order is **not guaranteed**.

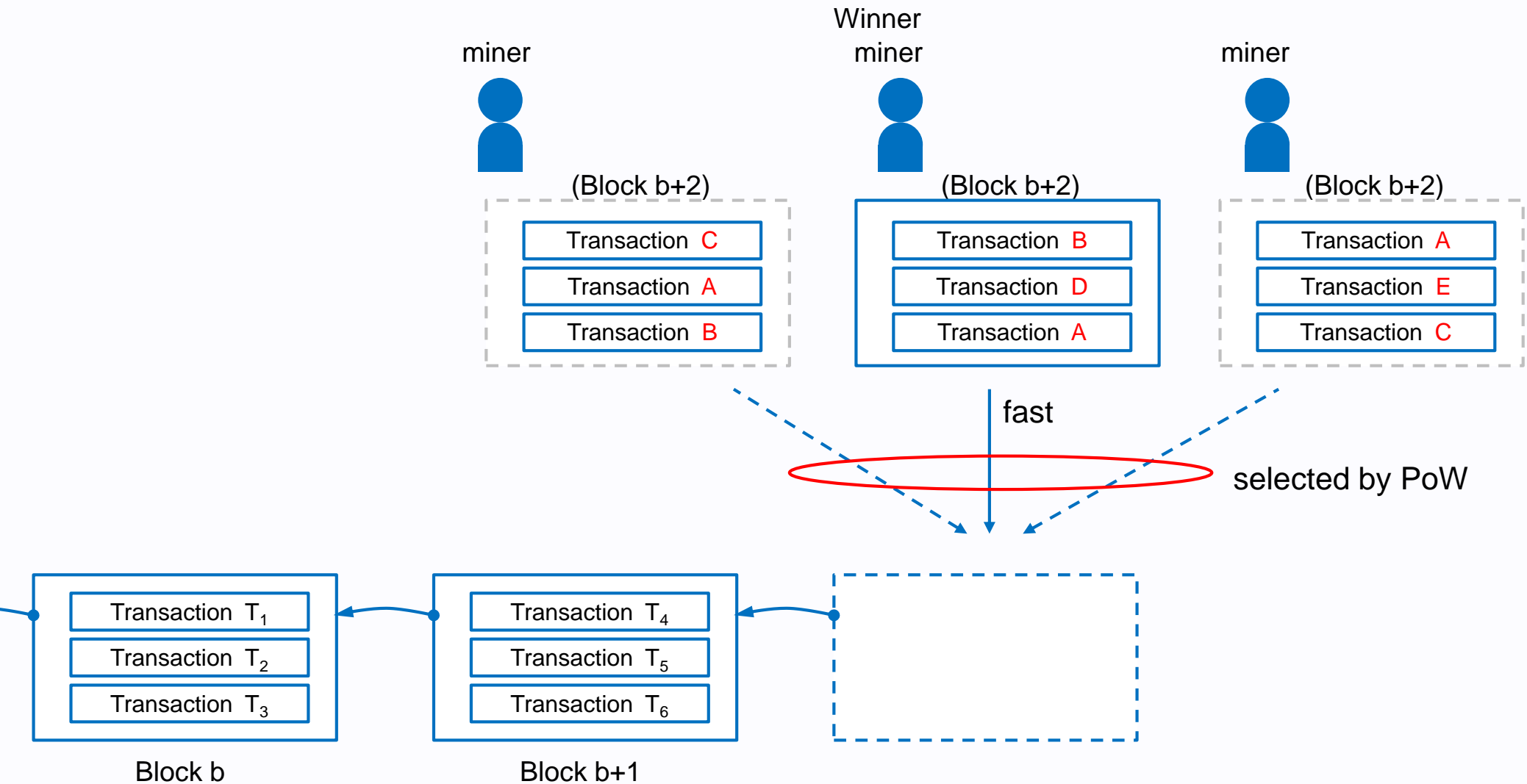


Ordering inner block



Miner can determine the order of transactions in a block.

Ordering inter blocks



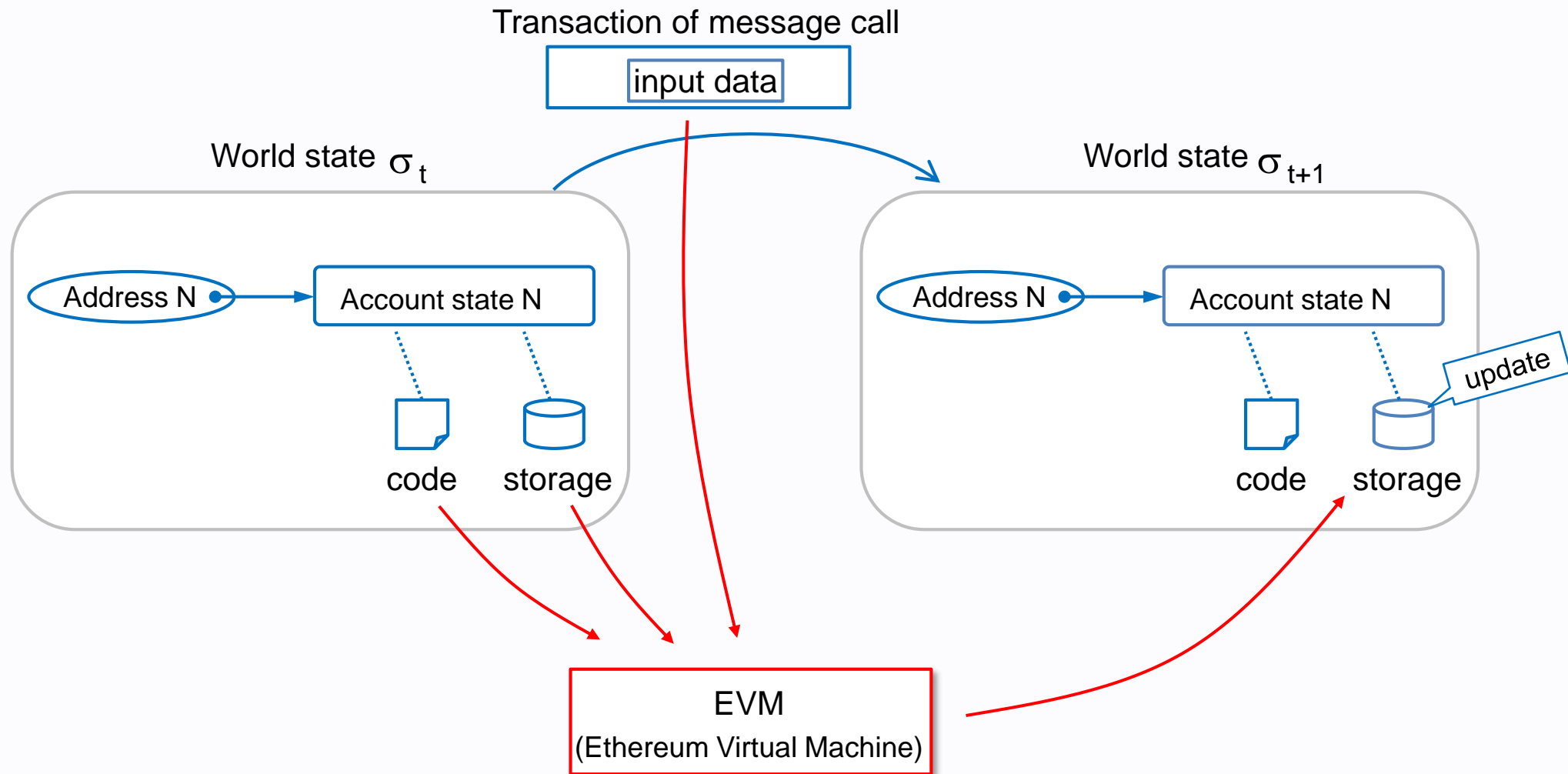
The order between blocks is determined by a consensus algorithm such as PoW.

2. Virtual machine

2. Virtual machine

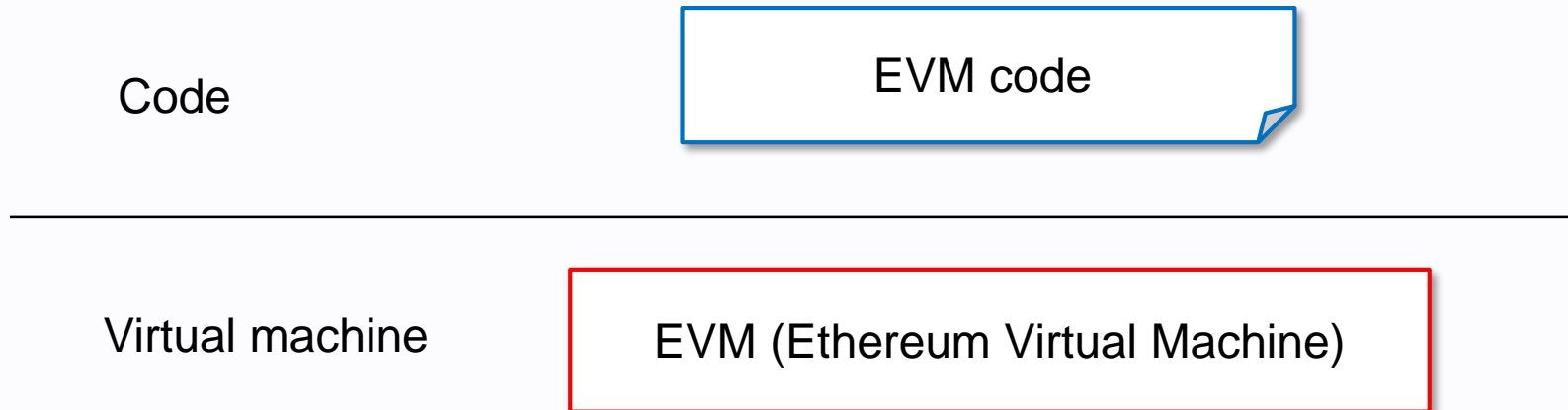
Ethereum virtual machine (EVM)

Ethereum virtual machine



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

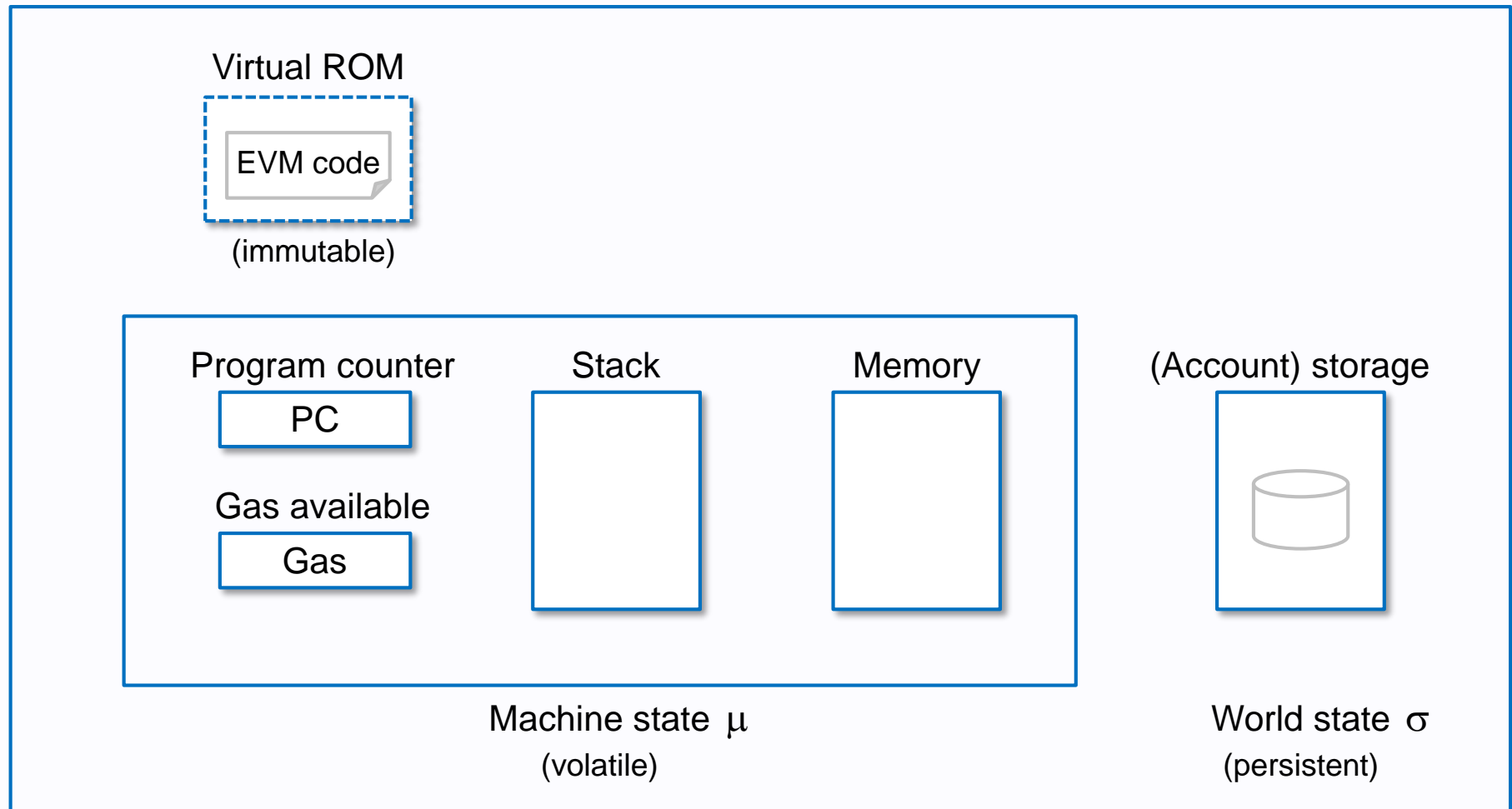
Ethereum virtual machine



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

EVM architecture

Ethereum Virtual Machine (EVM)



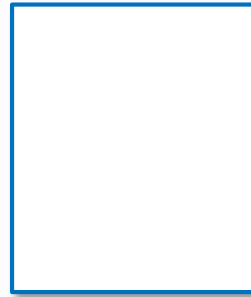
The EVM is a simple stack-based architecture.

Machine space of EVM

Registers



Stack



stack memory

256 bits x 1024 elements

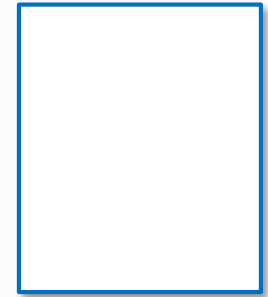
Memory



volatile memory

byte addressing
linear memory

(Account) storage

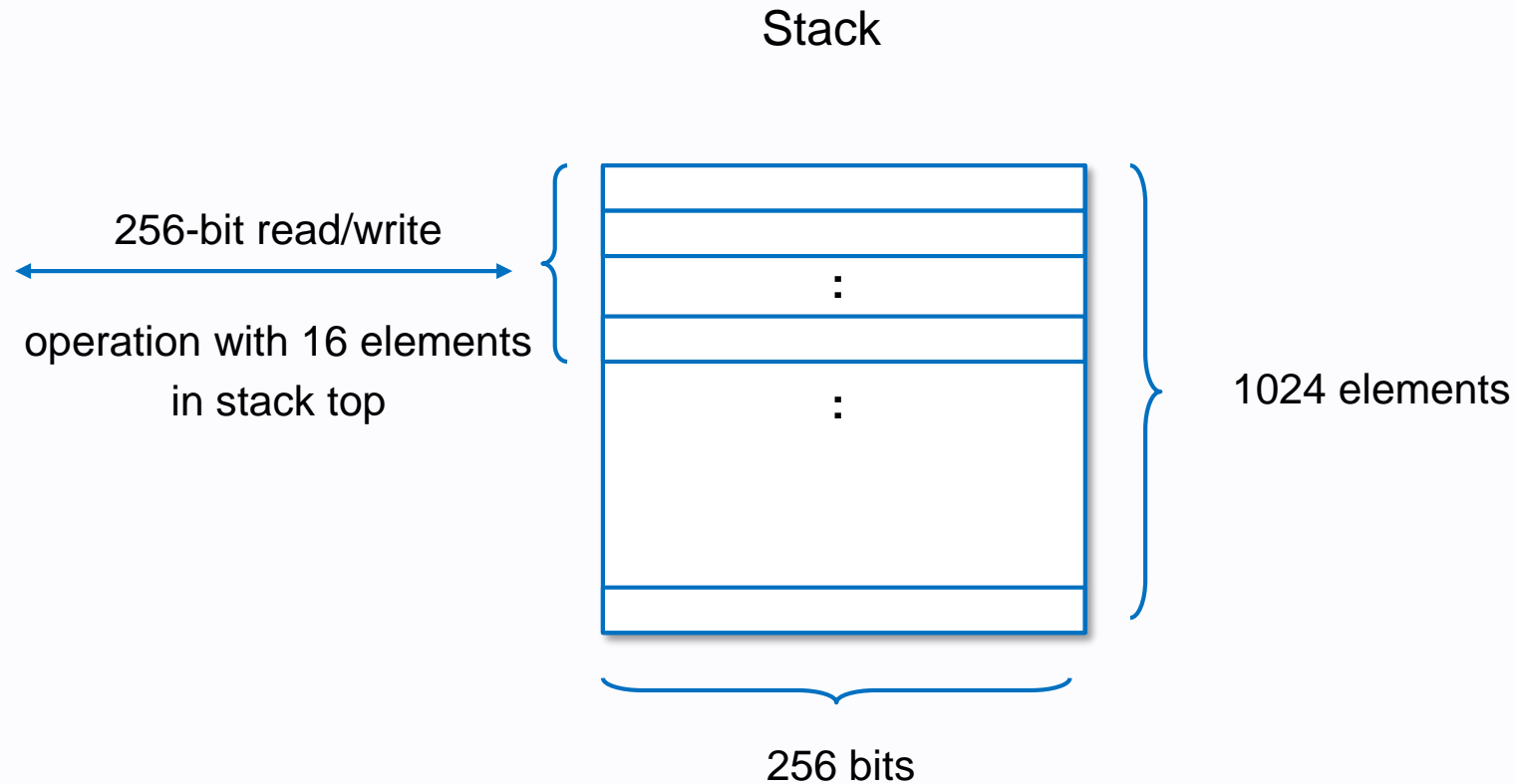


persistent memory

256 bits to 256 bits
key-value store

There are several resources as space.

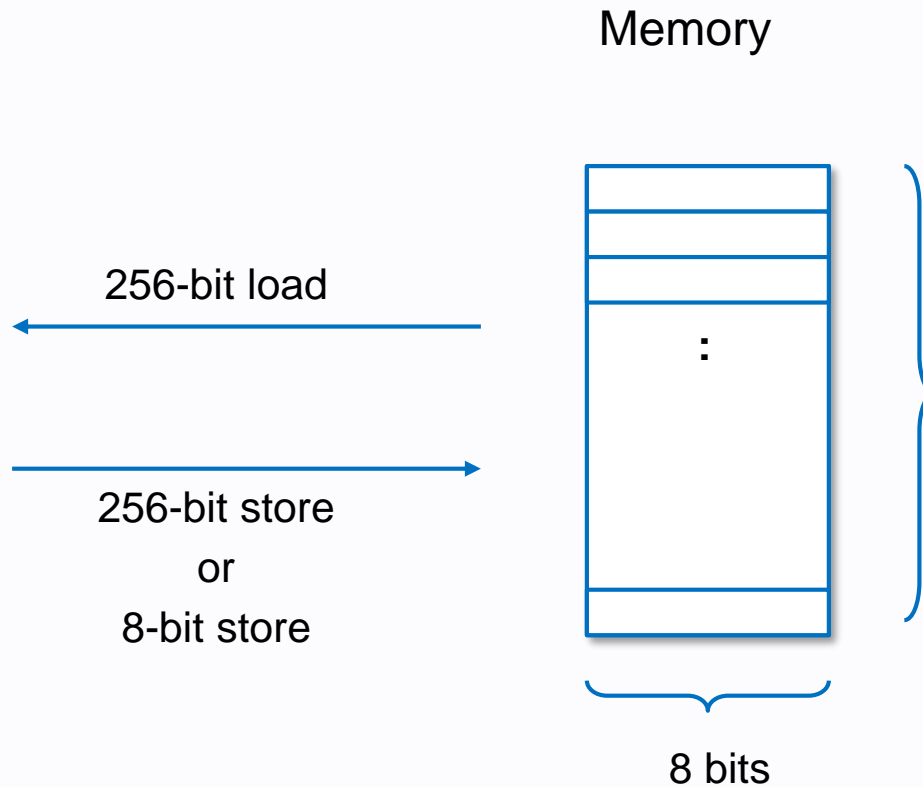
Stack



All operation are performed on the stack.

Access with many instructions such as PUSH/POP/COPY/SWAP, ...

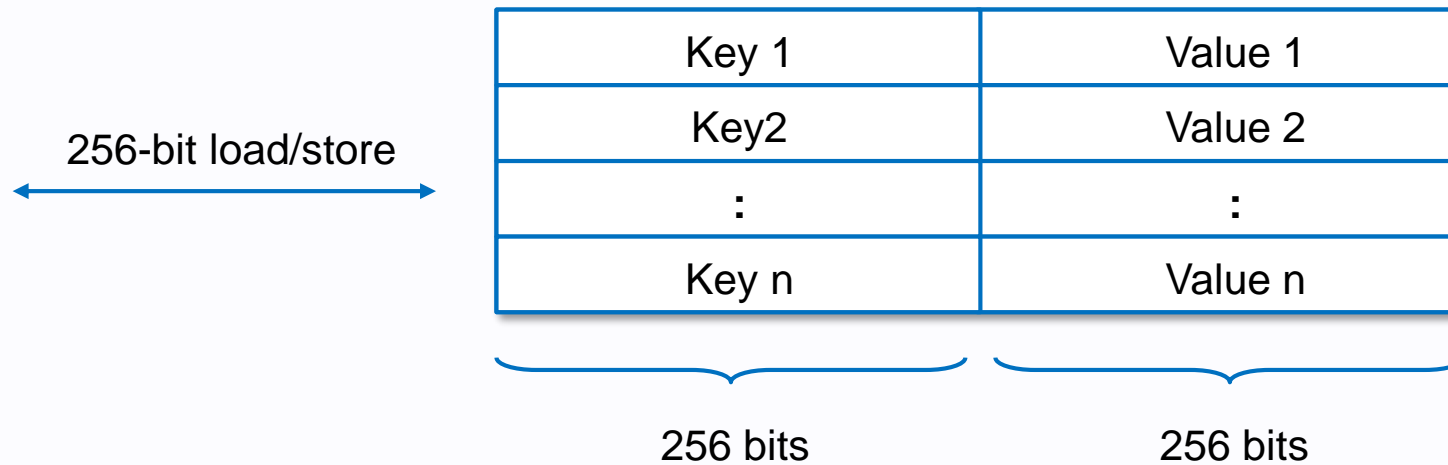
Memory



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.

Account storage

(Account) storage



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.

EVM code

Assembly view

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

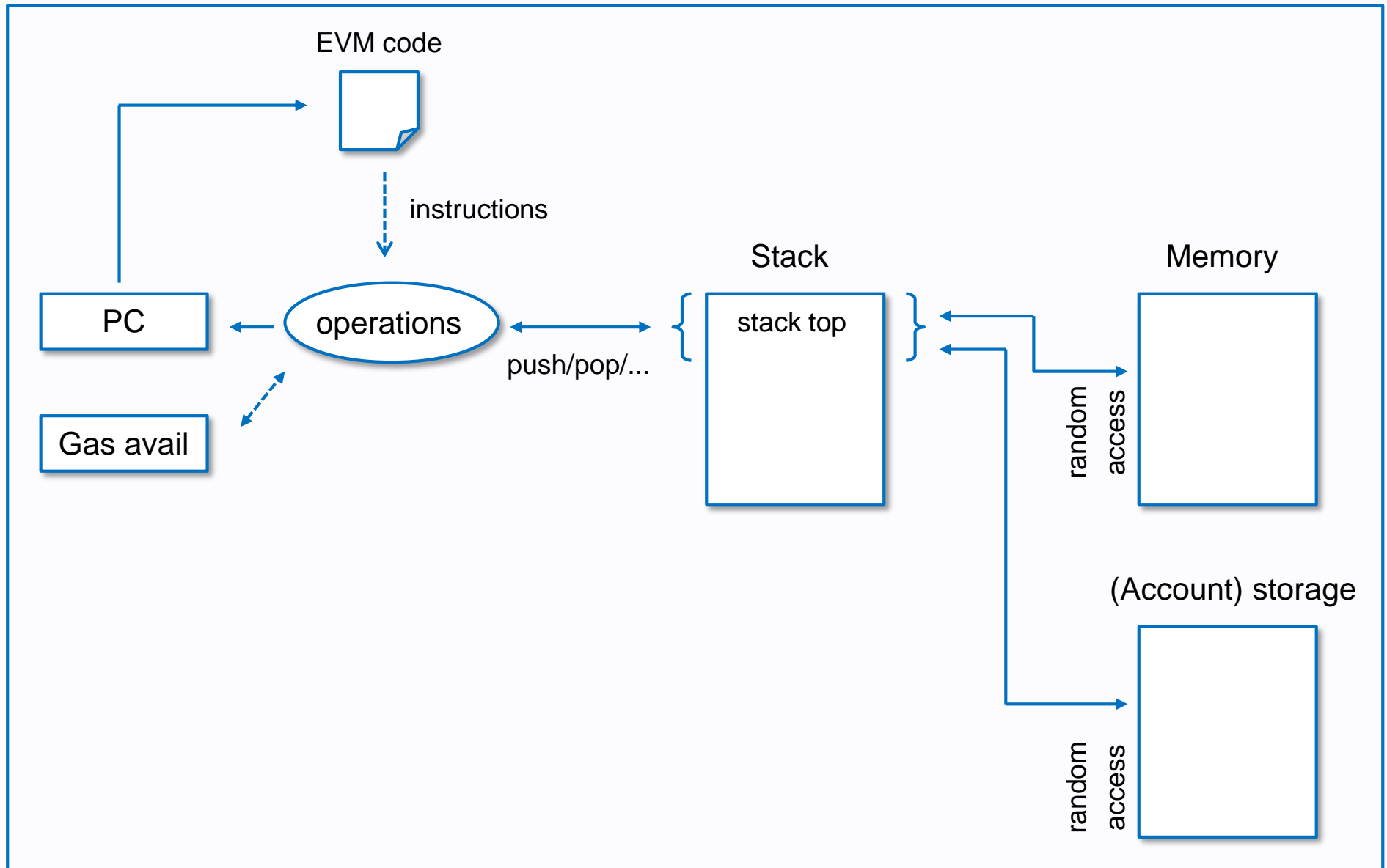
Bytecode view

```
0x60e060020a600035...
```

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

Execution model

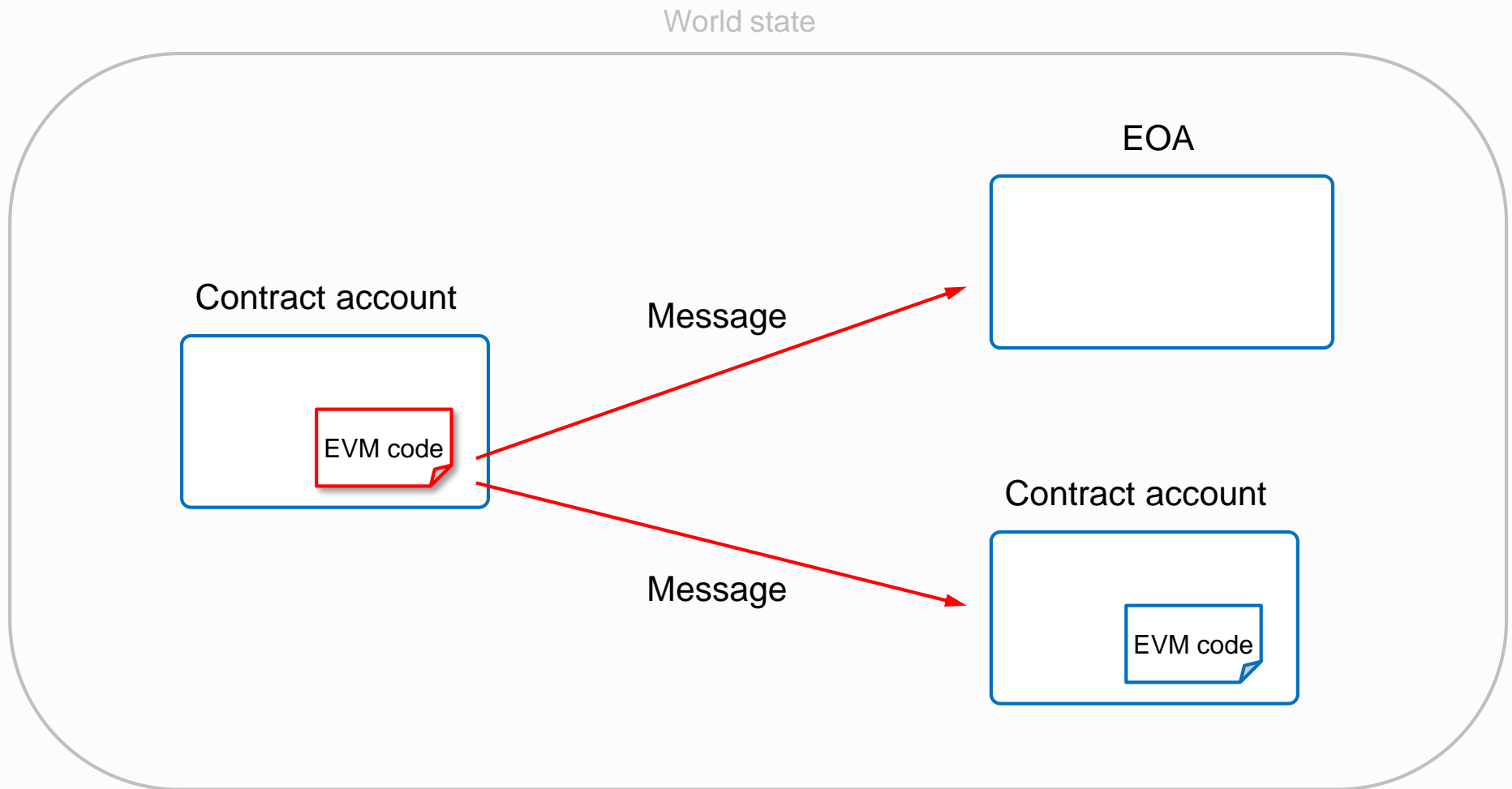
EVM



2. Virtual machine

Message call

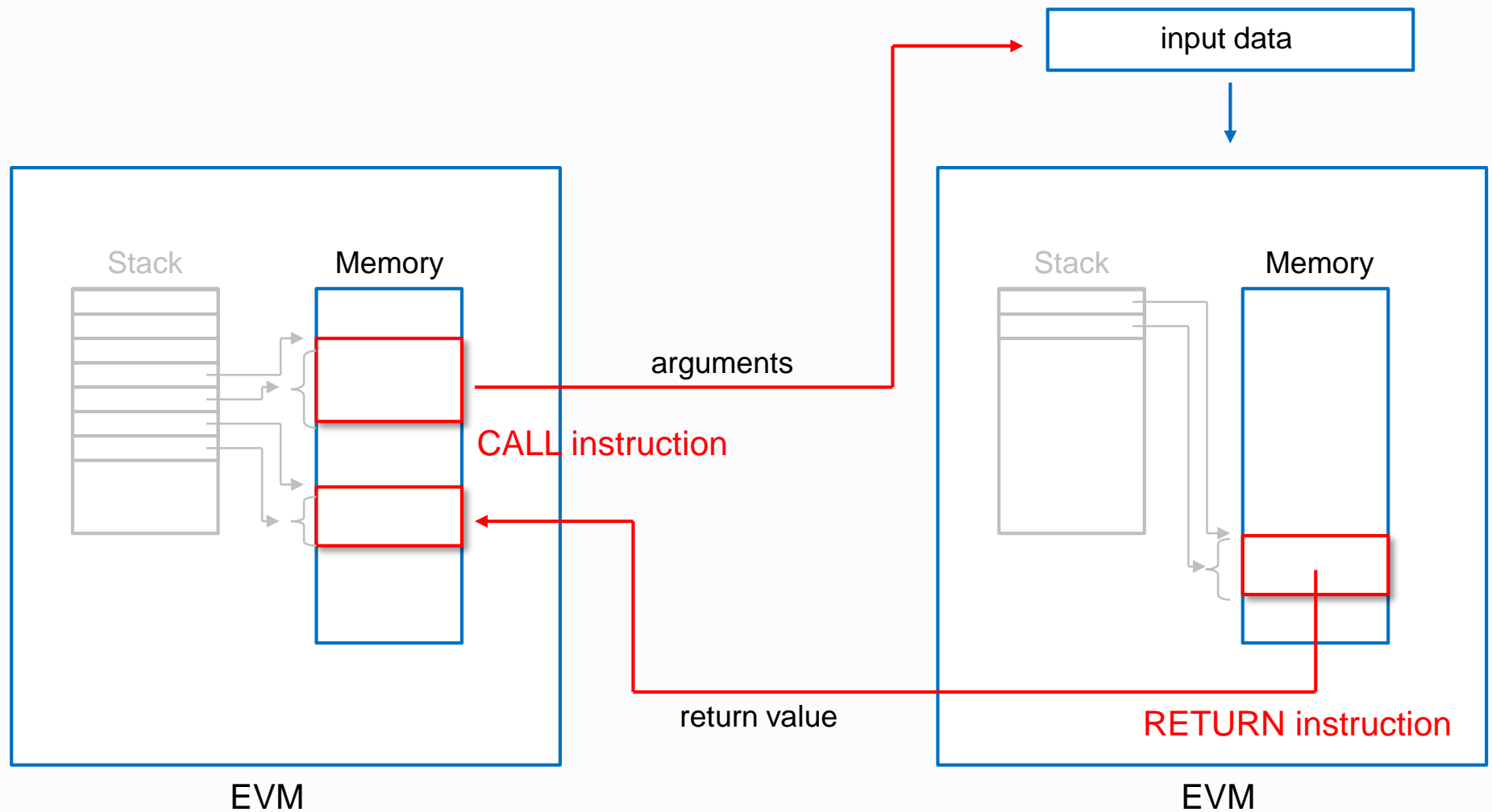
Message call



EVM can send a message to other account.

The depth of message call is limited to less than 1024 levels.

Instructions for Message call

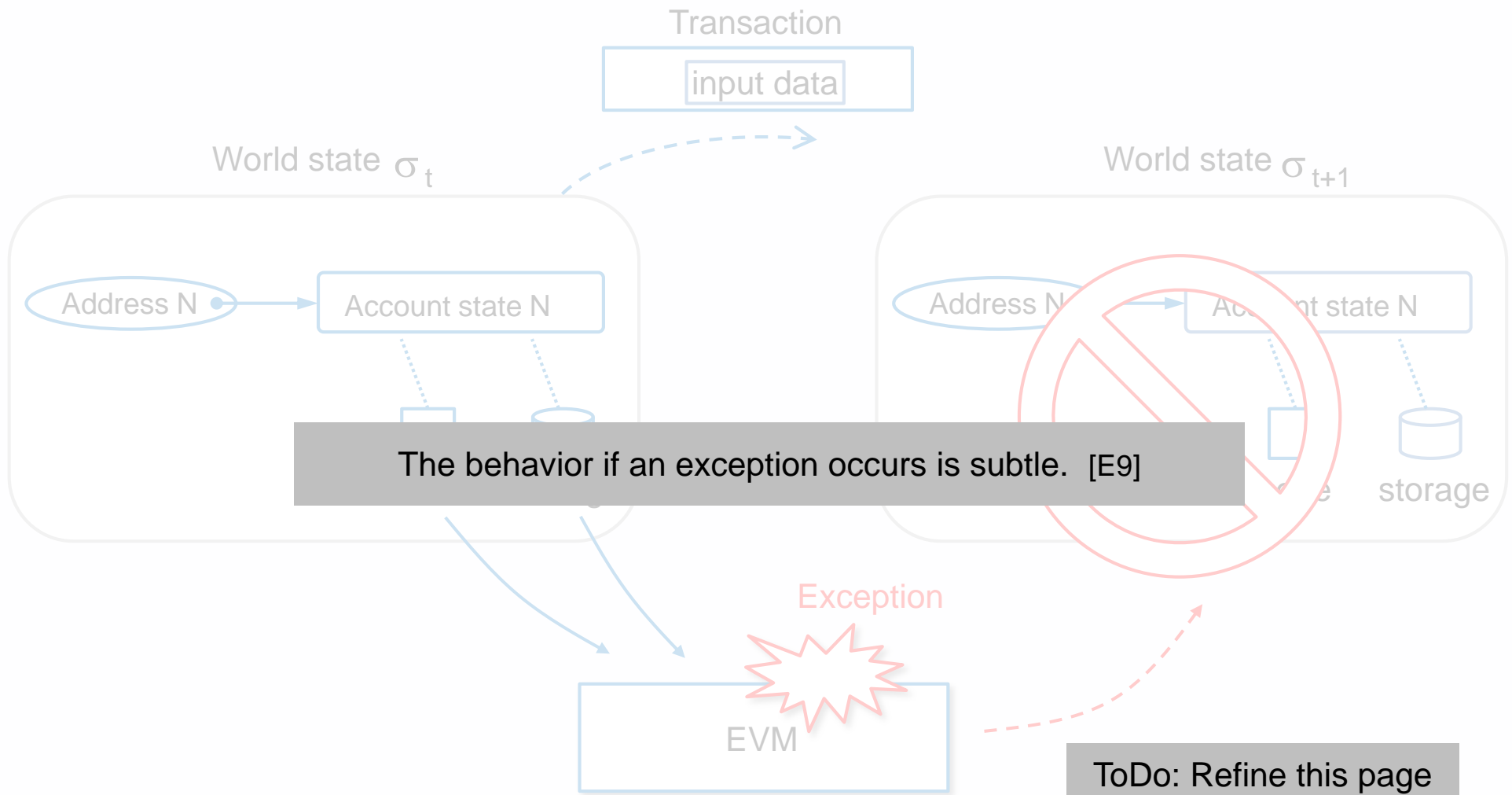


Message call is triggered by CALL instruction.
Arguments and return values are passed using memory.

2. Virtual machine

Exception

Exception

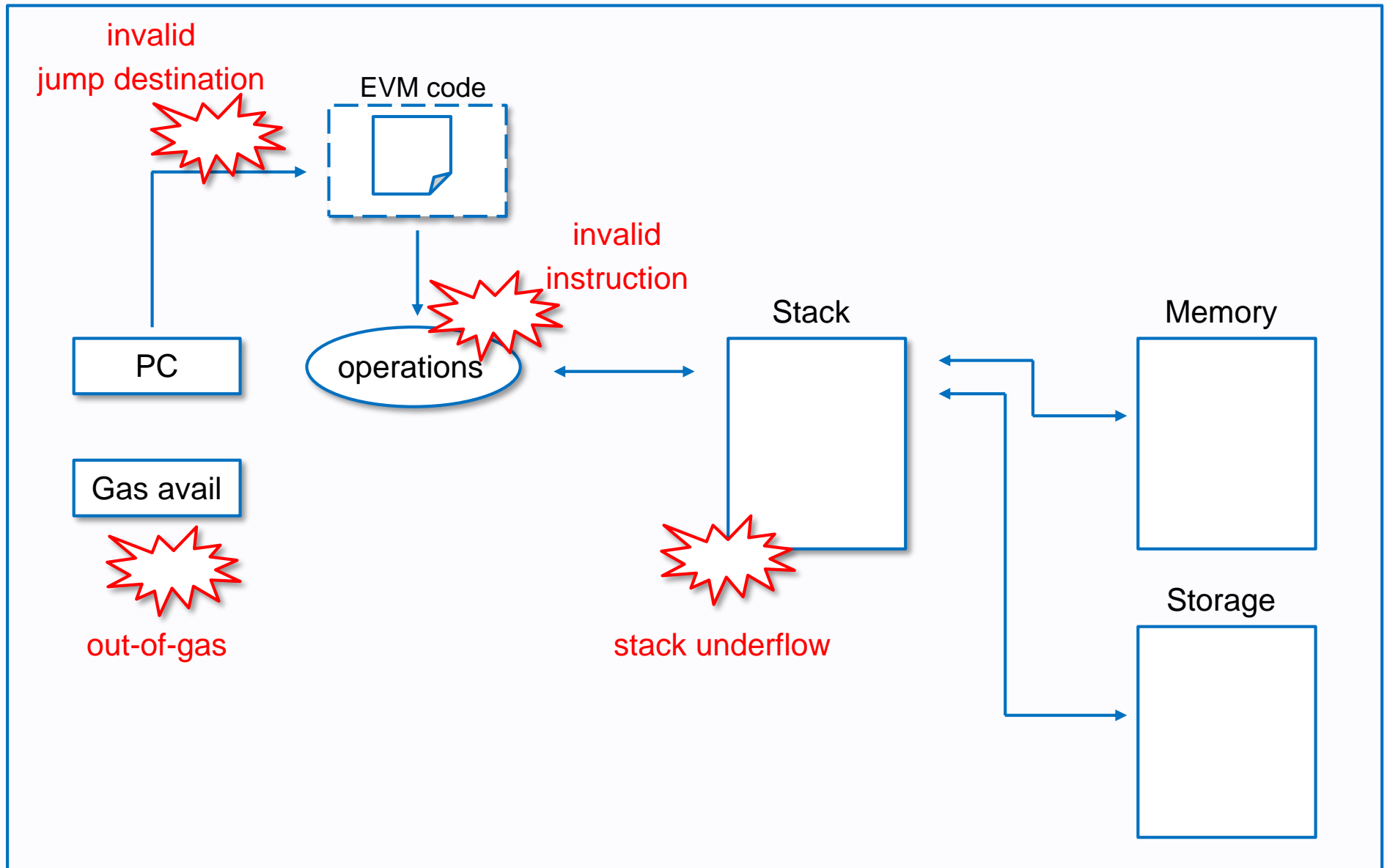


~~If an exception occurs in the EVM, the state is not updated.~~

WIP

Exception

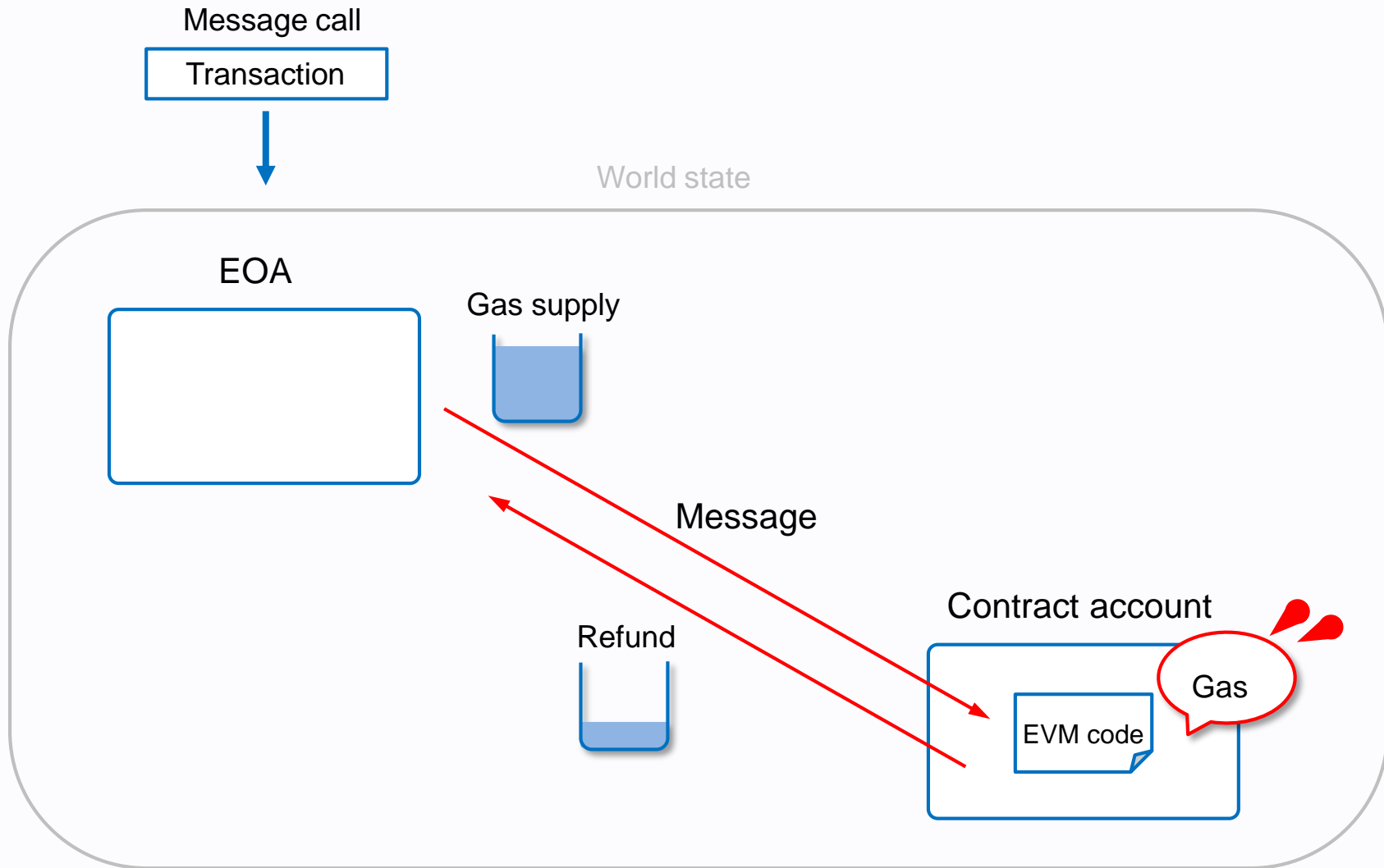
EVM



2. Virtual machine

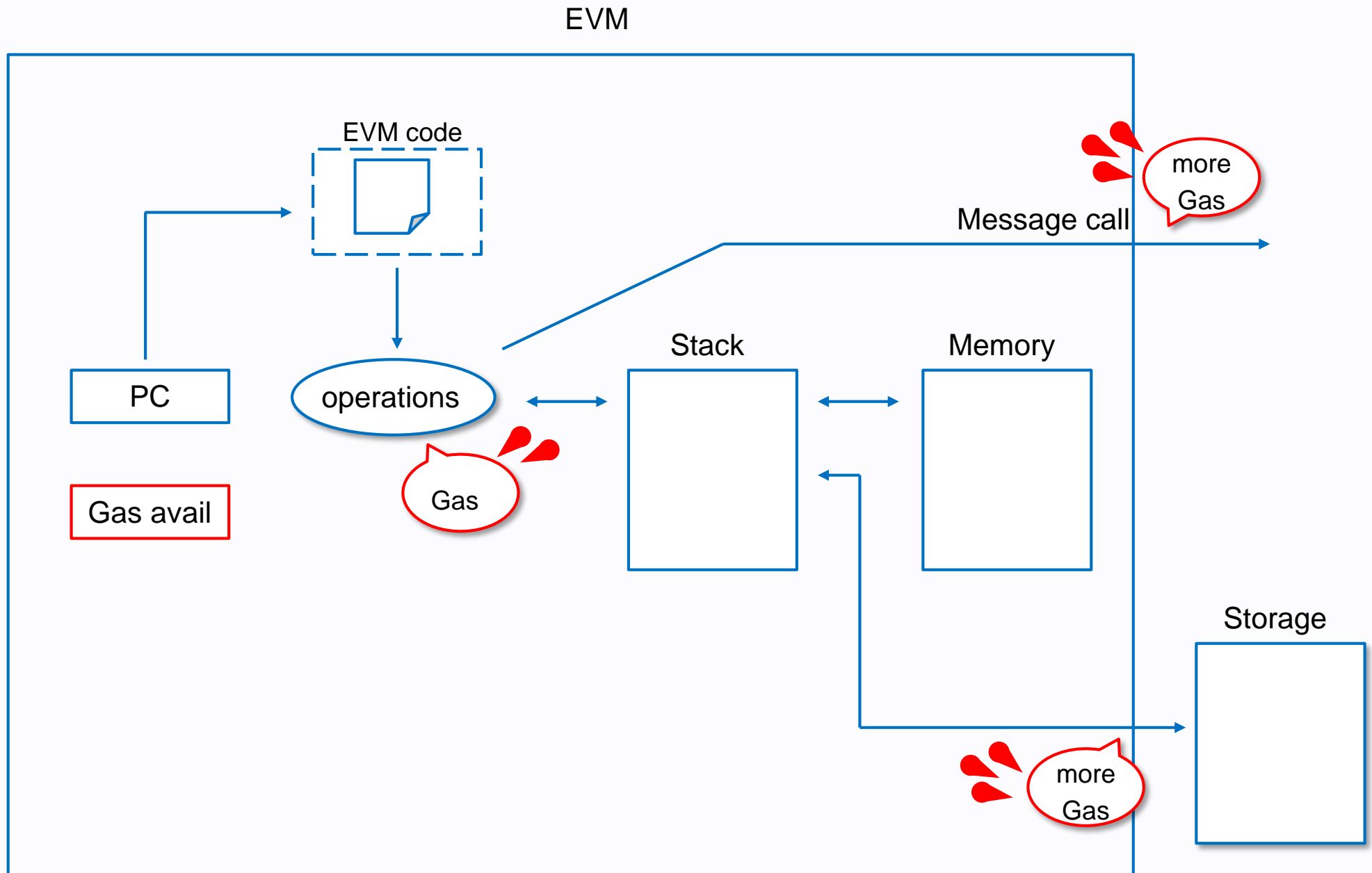
Gas and fee

Gas and fee



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

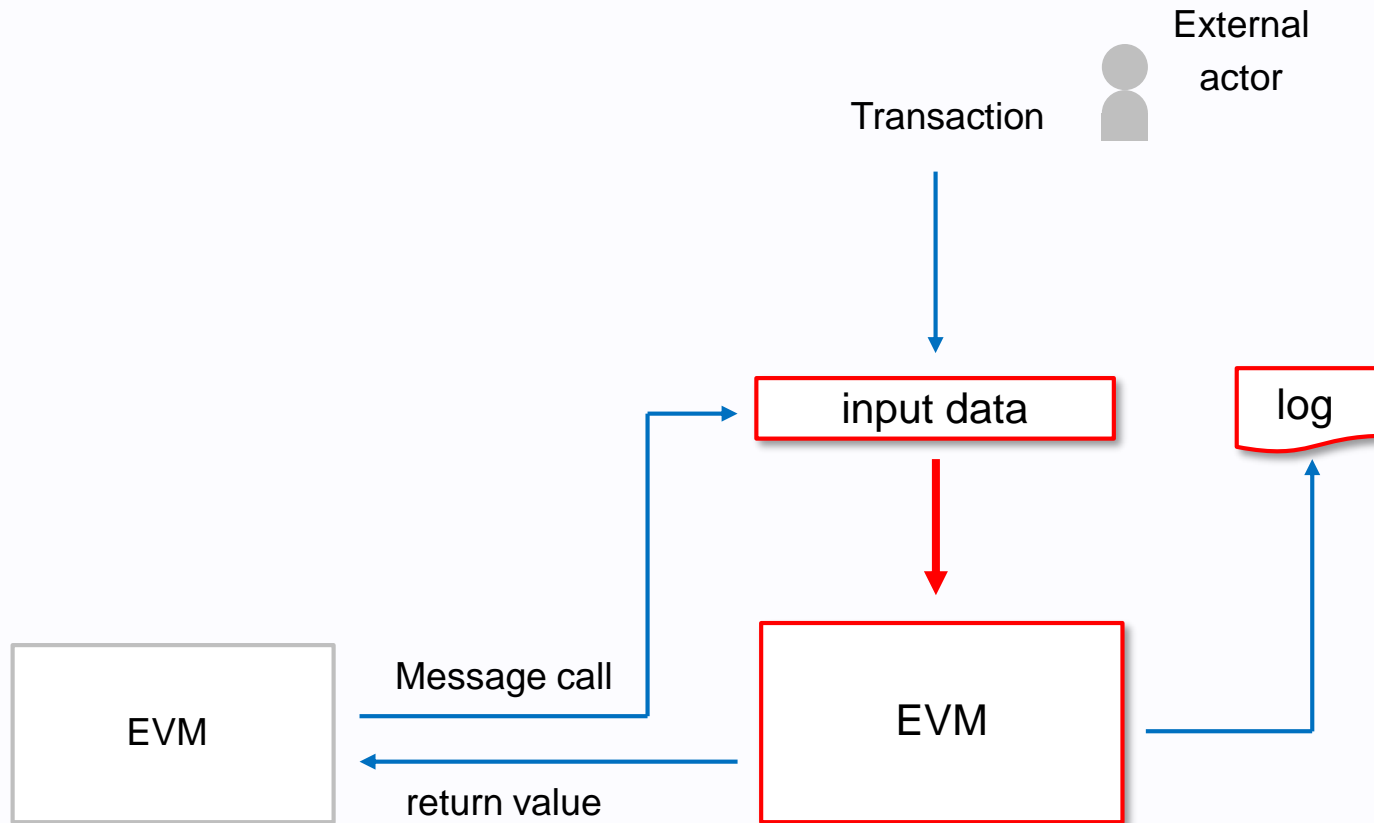
Gas



2. Virtual machine

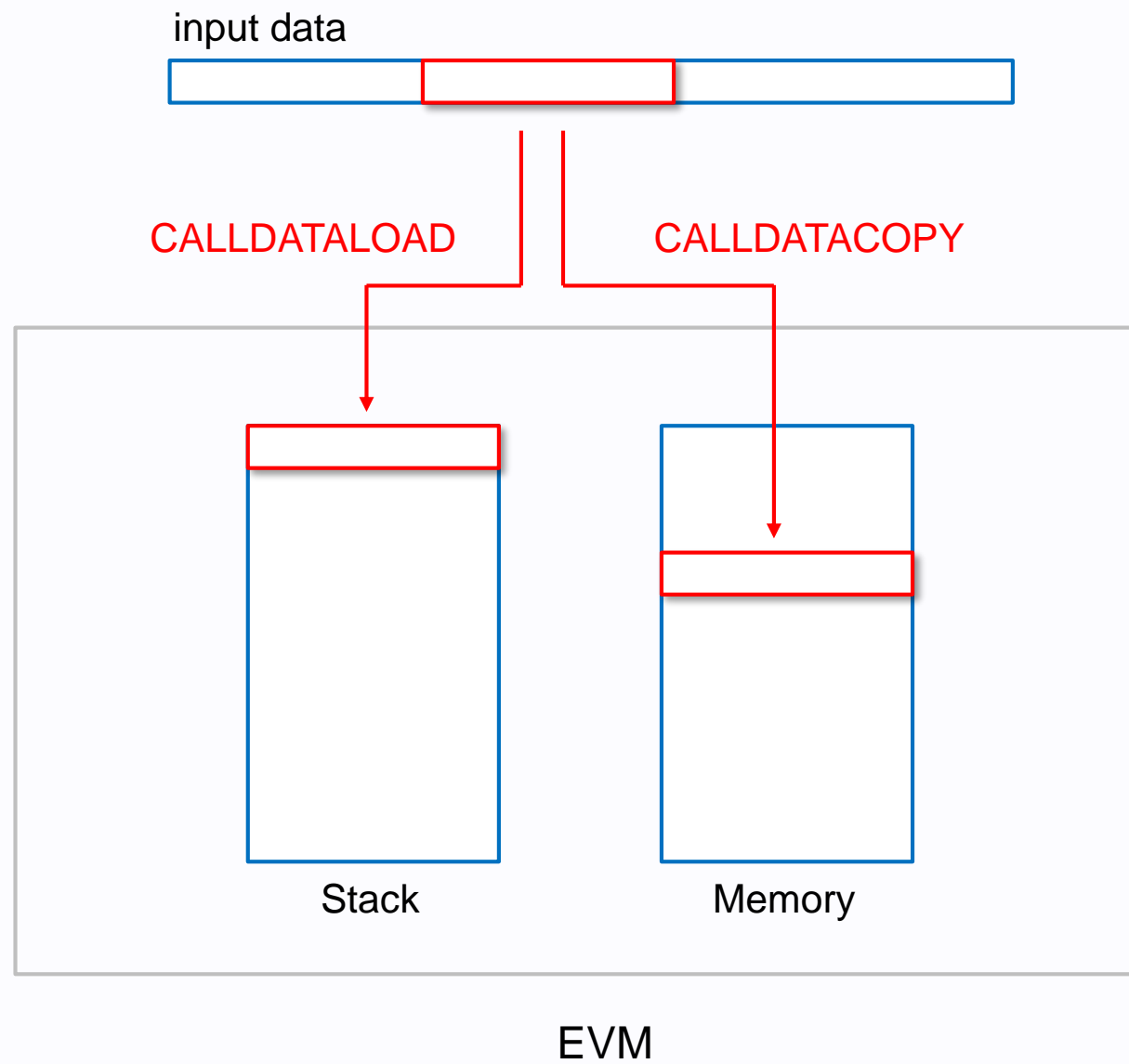
Input and output

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.

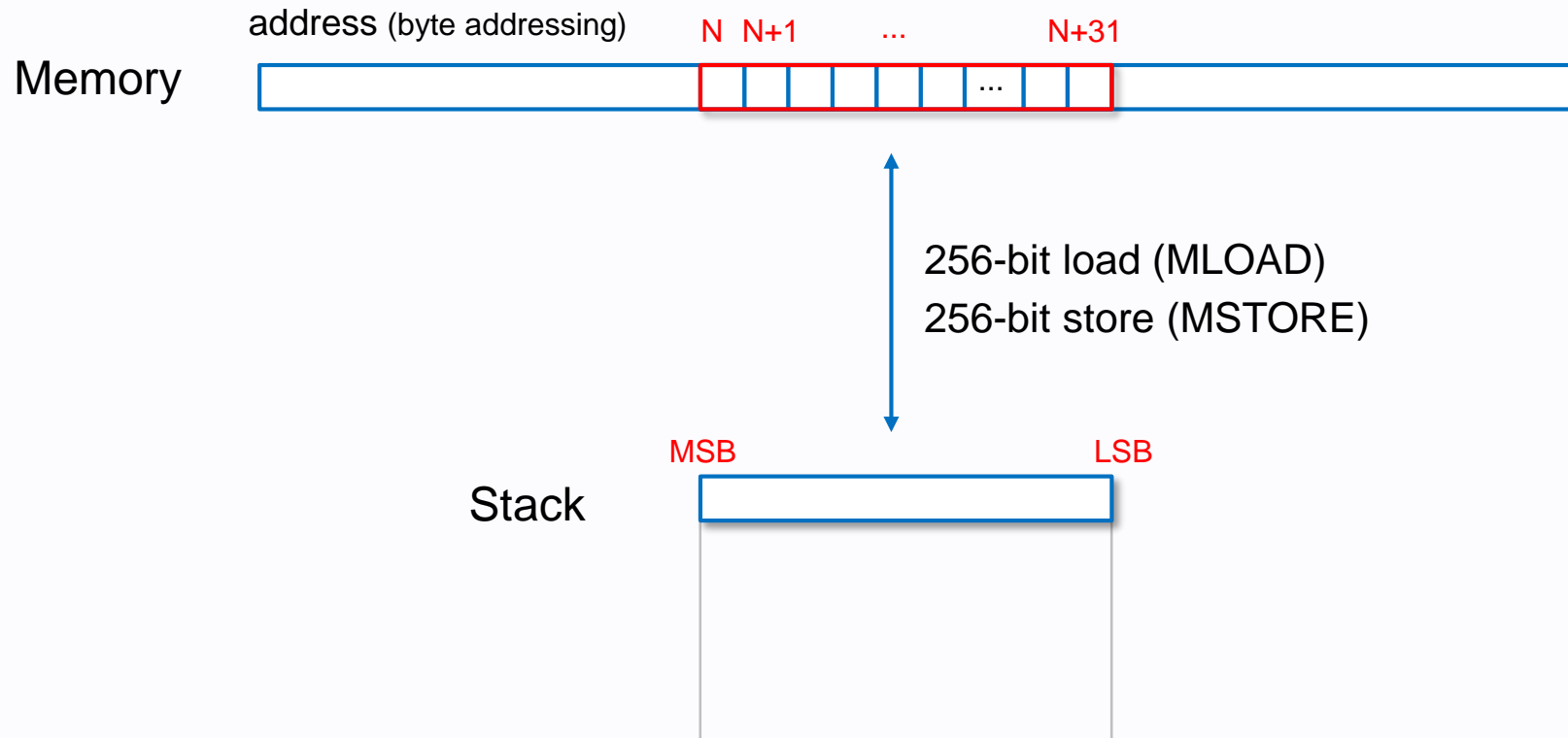
Instructions for input data



2. Virtual machine

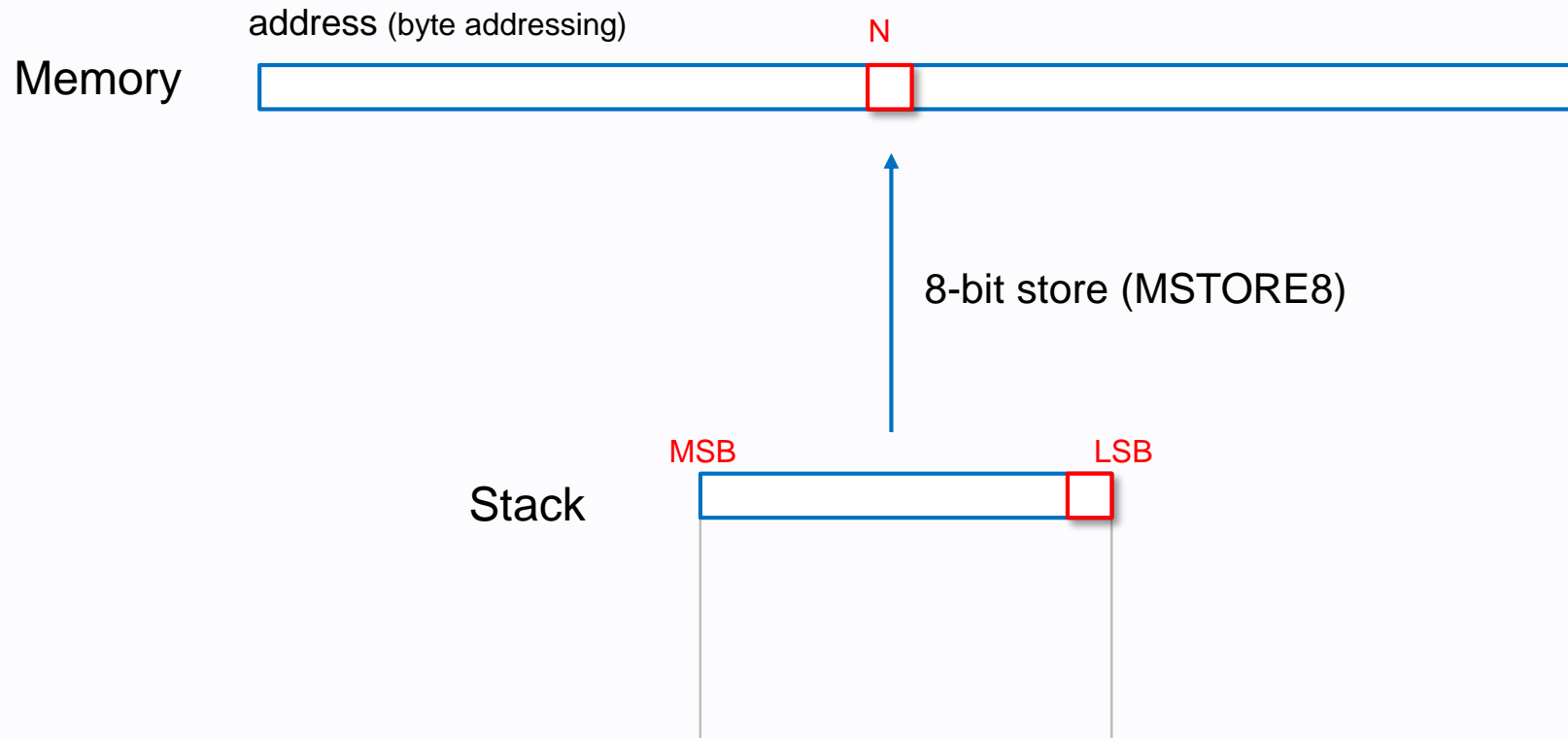
Byte order

Endian for Memory



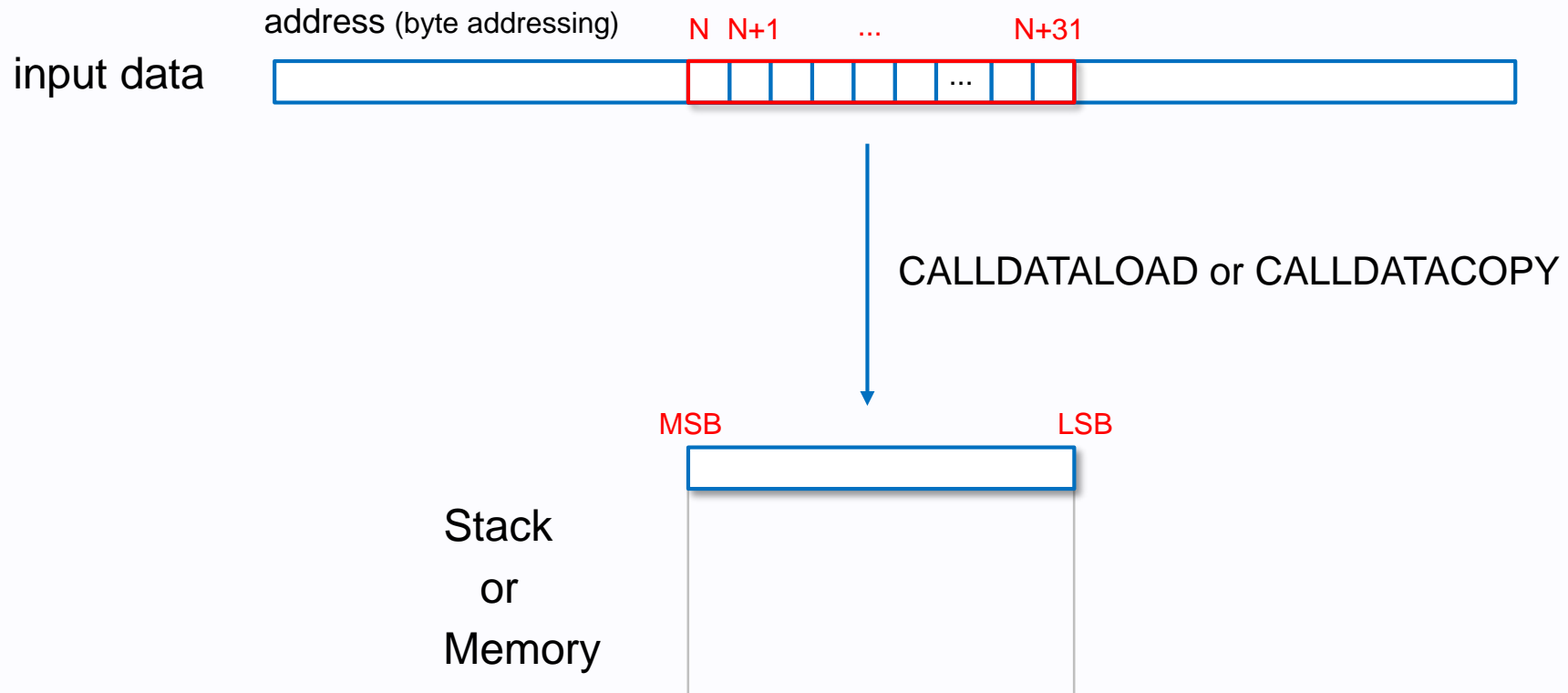
EVM is big endian order (network byte order).

Endian for Memory



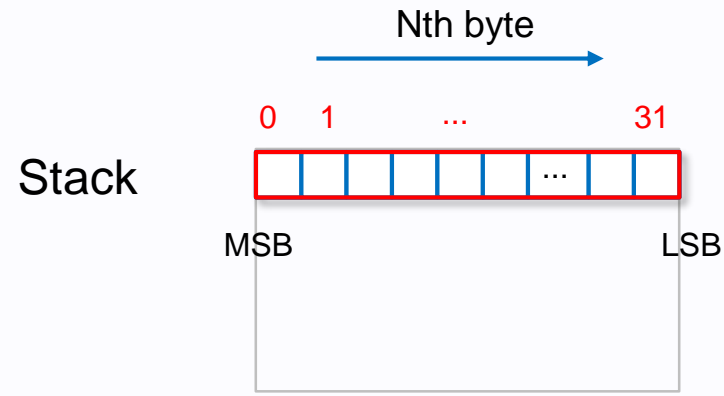
EVM is big endian order (network byte order).

Endian for input data

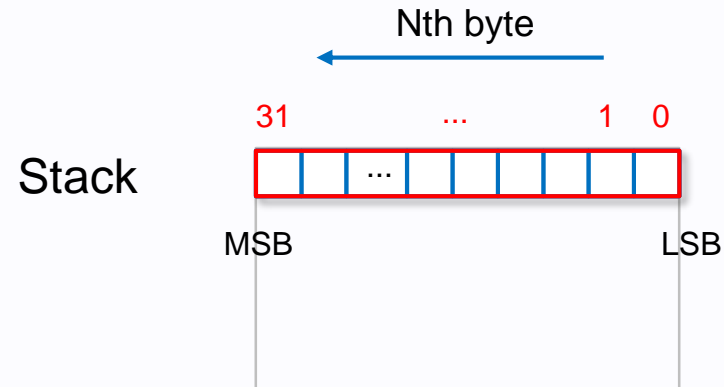


EVM is big endian order (network byte order).

Byte order of BYTE and SIGNEXTEND instruction



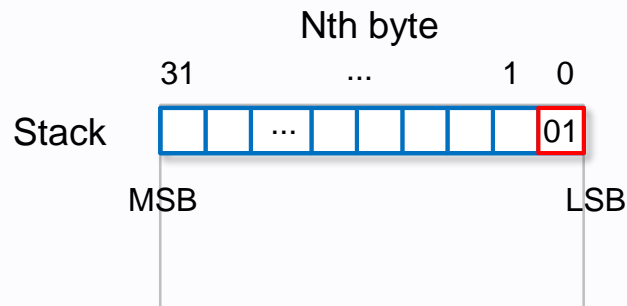
BYTE instruction counts from MSB.



SIGNEXTEND instruction counts from LSB.

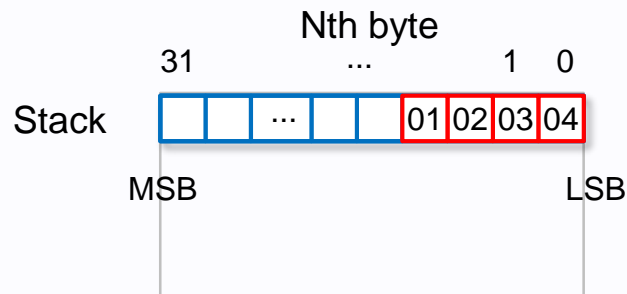
Byte order of PUSH instructions

PUSH1 0x01



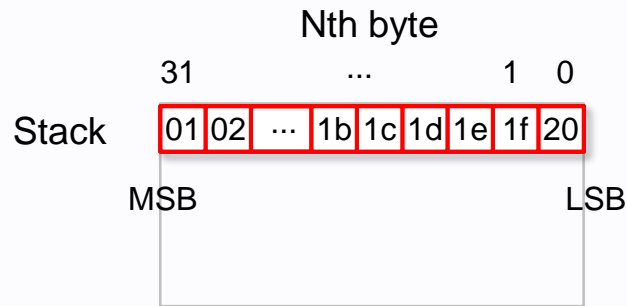
right-aligned, big endian

PUSH4 0x01020304



right-aligned, big endian

PUSH32 0x0102...1f20



right-aligned, big endian

2. Virtual machine

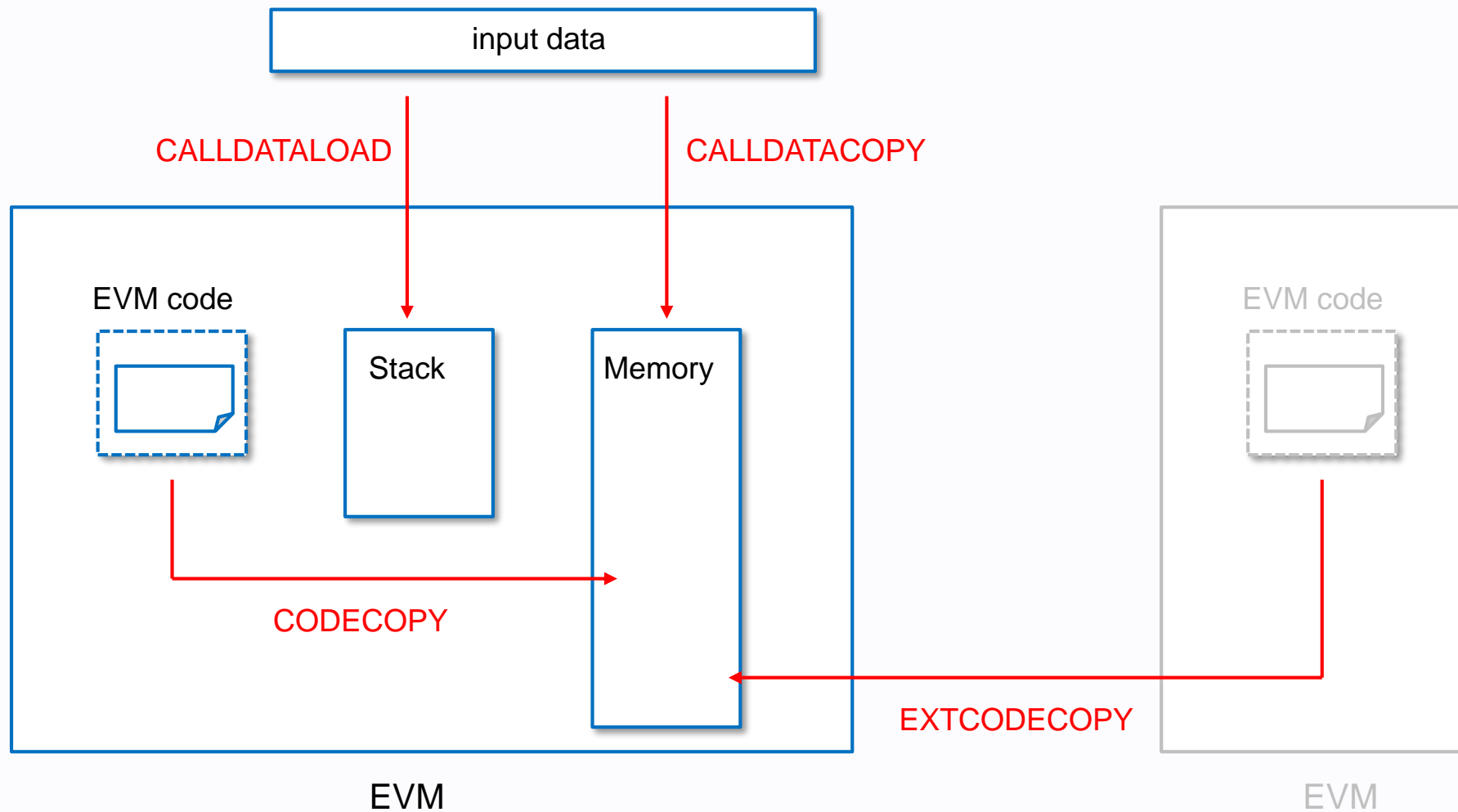
Instruction set

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
- * ...

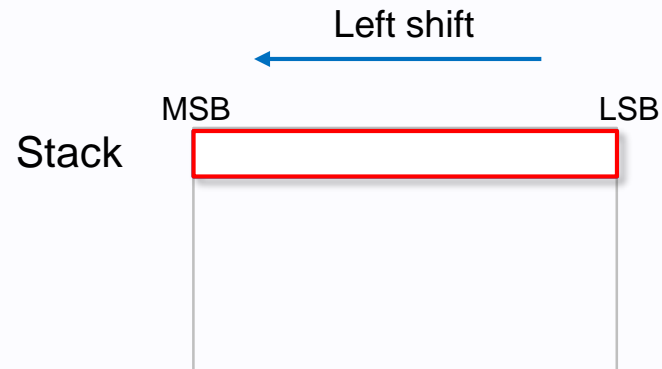
WIP

Copy of code and input data



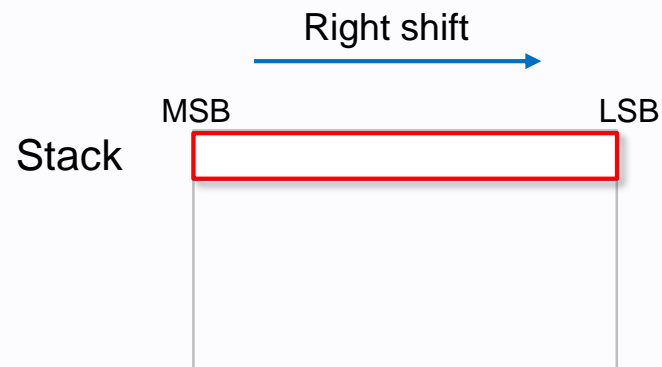
There are several copy instructions for inter spaces.

Shift by MUL, DIV and SDIV



$$\text{MUL } m \ (2^n) == m \ll n$$

Left shift is represented by MUL instruction.



$$\text{DIV } m \ (2^n) == m \gg n$$

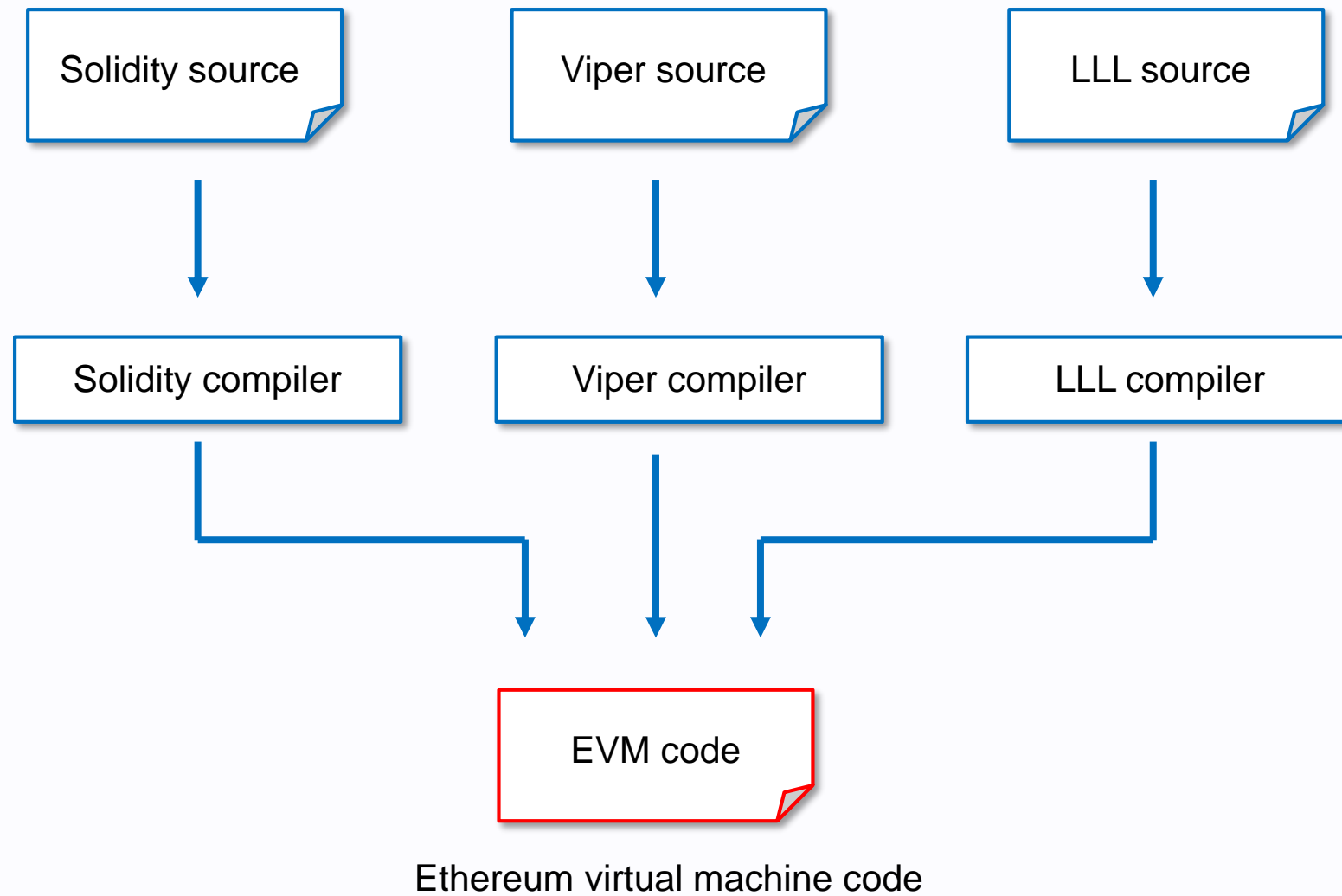
DIV for logical right shift
SDIV for arithmetic right shift

Right shift is represented by DIV and SDIV instruction.

2. Virtual machine

Miscellaneous

EVM code generation



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, ...)

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 {
```

Block header

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

```
}
```

Transaction

(go-ethereum version 1.8)

[core/types/transaction.go]

Transaction

```
type txdata struct {
    AccountNonce uint64          `json:"nonce" gencodec:"required"`
    Price         *big.Int                    `json:"gasPrice" gencodec:"required"`
    GasLimit      uint64                    `json:"gas" gencodec:"required"`
    Recipient     *common.Address `json:"to" rlp:"nil"`
    // nil means contract creation

    Amount *big.Int `json:"value" gencodec:"required"`
    Payload []byte   `json:"input" gencodec:"required"`

    // 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 (Wei)

input data

World state

(go-ethereum version 1.8)

[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  
  
    :
```

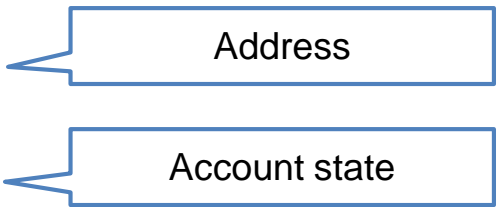
Mapping for
Address to Account state

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)  
}
```



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

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)}  
}
```



[core/vm/memory.go]

```
type Memory struct {  
    store      []byte  
    lastGasCost uint64  
}  
  
func NewMemory() *Memory {  
    return &Memory{}  
}
```



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
}
```

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
}
```

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()
    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 {
        :
    }
}
```

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(),  
:  
    )
```

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

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

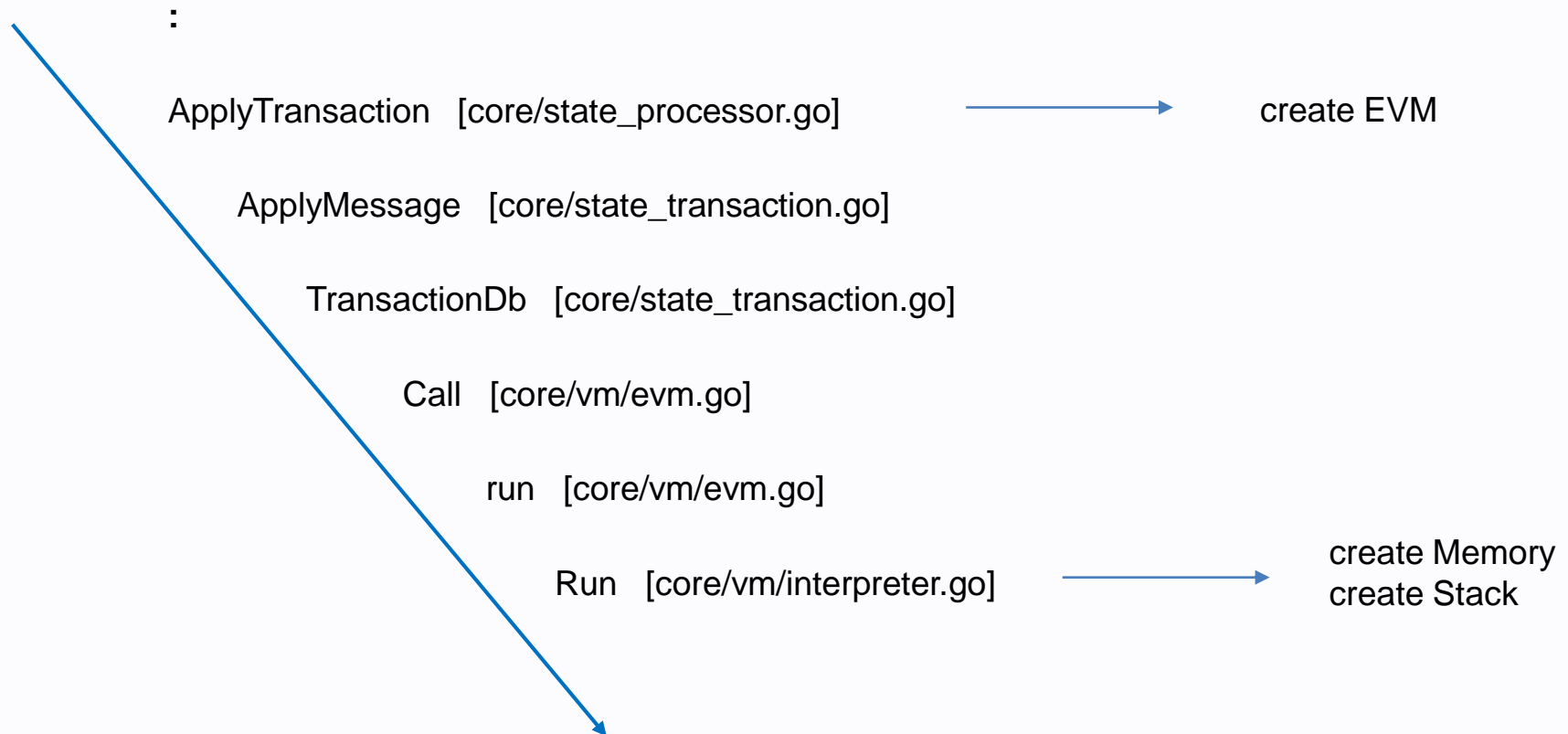
added instruction:
DELEGATECALL

[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),
    }
)
```

Bootstrap of EVM in Geth

(go-ethereum version 1.8)



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
```

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:
```

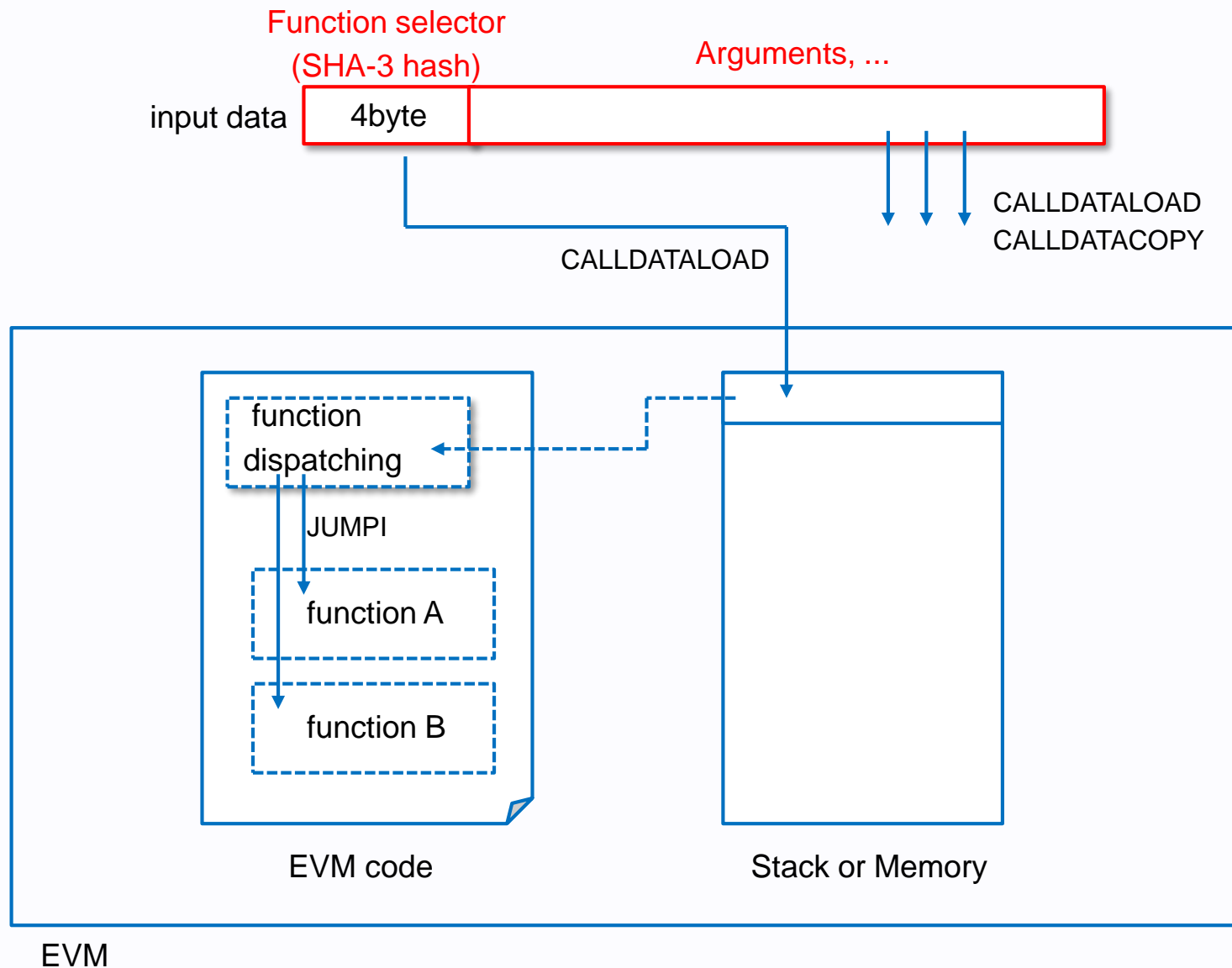
```
00000000  0000000000000000000000000000000000000000000000000000000000000003
```

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

Appendix A

Solidity ABI

Solidity Application Binary Interface

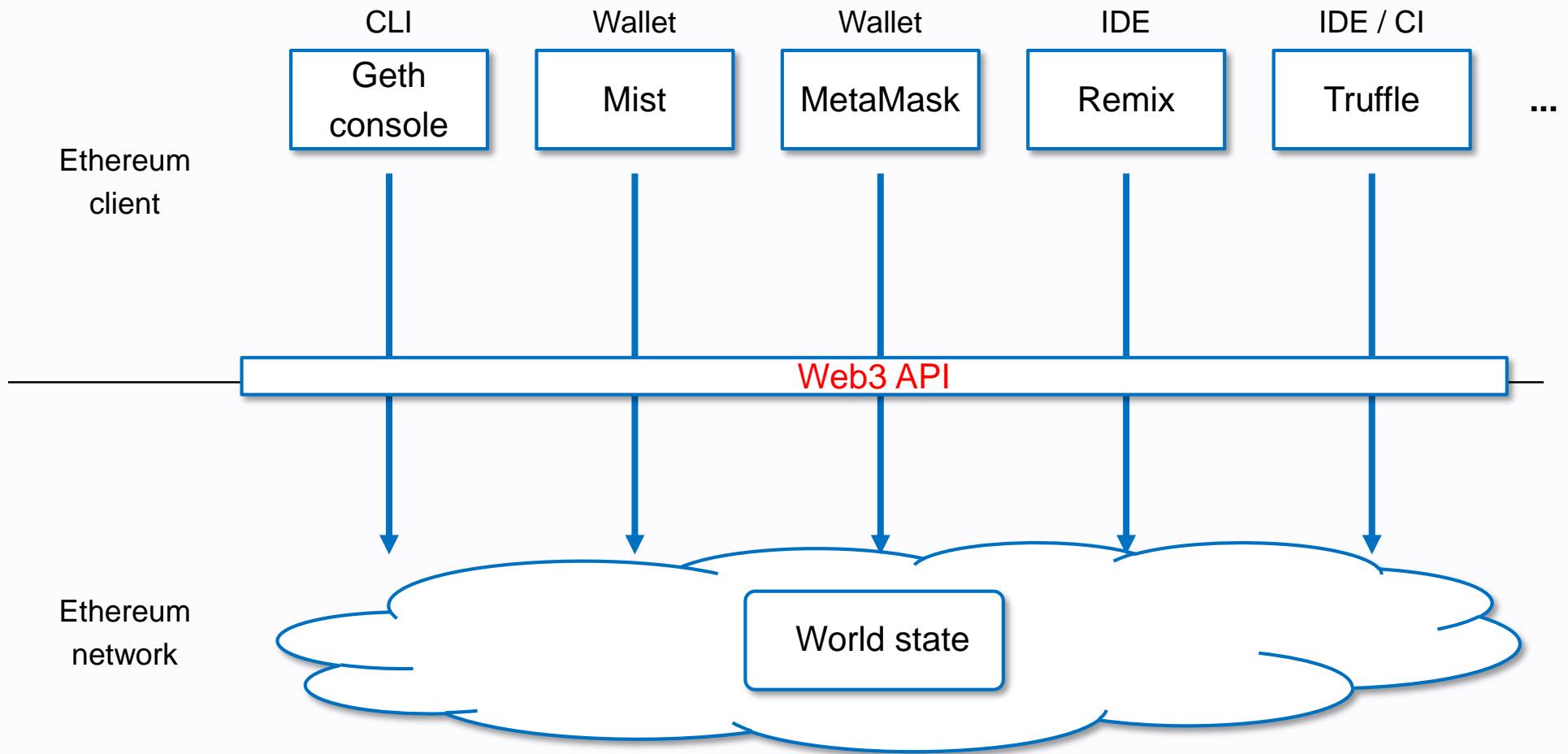


Appendix B

Appendix B

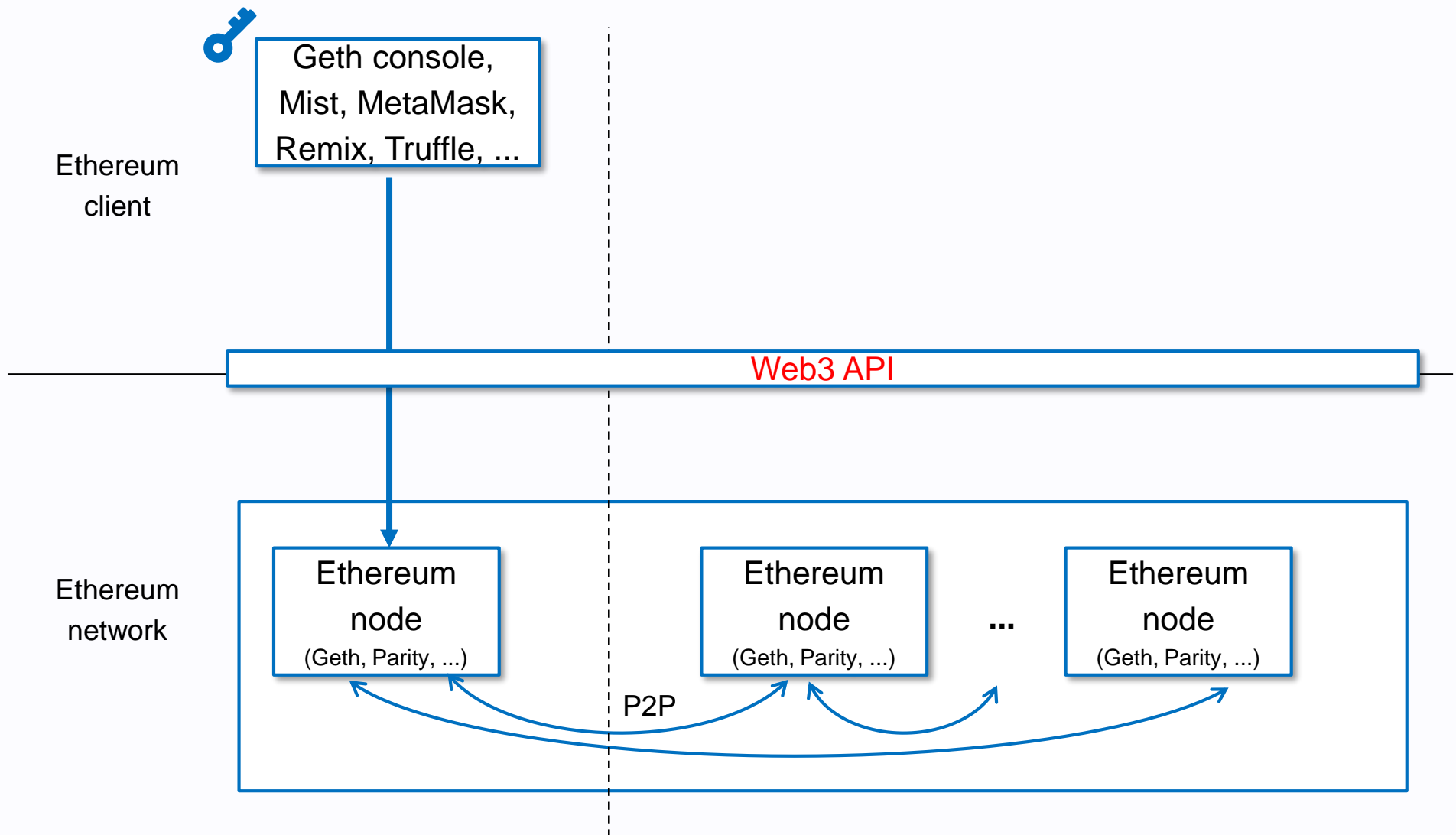
Web3 API

Web3 API and client



Ethereum clients access to Ethereum network via Web3 API.

Web3 API and client

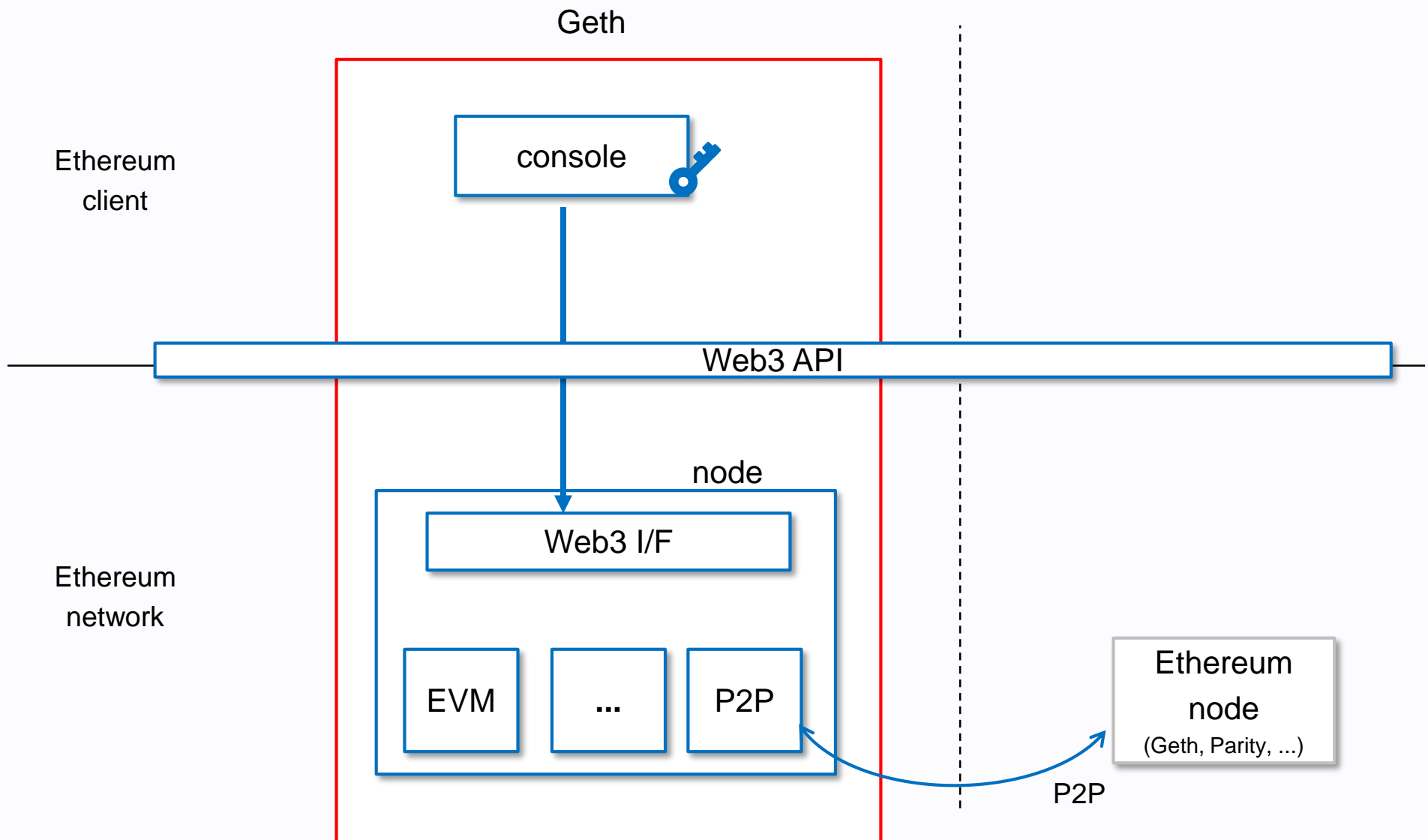


Ethereum clients access to Ethereum network via Web3 API.

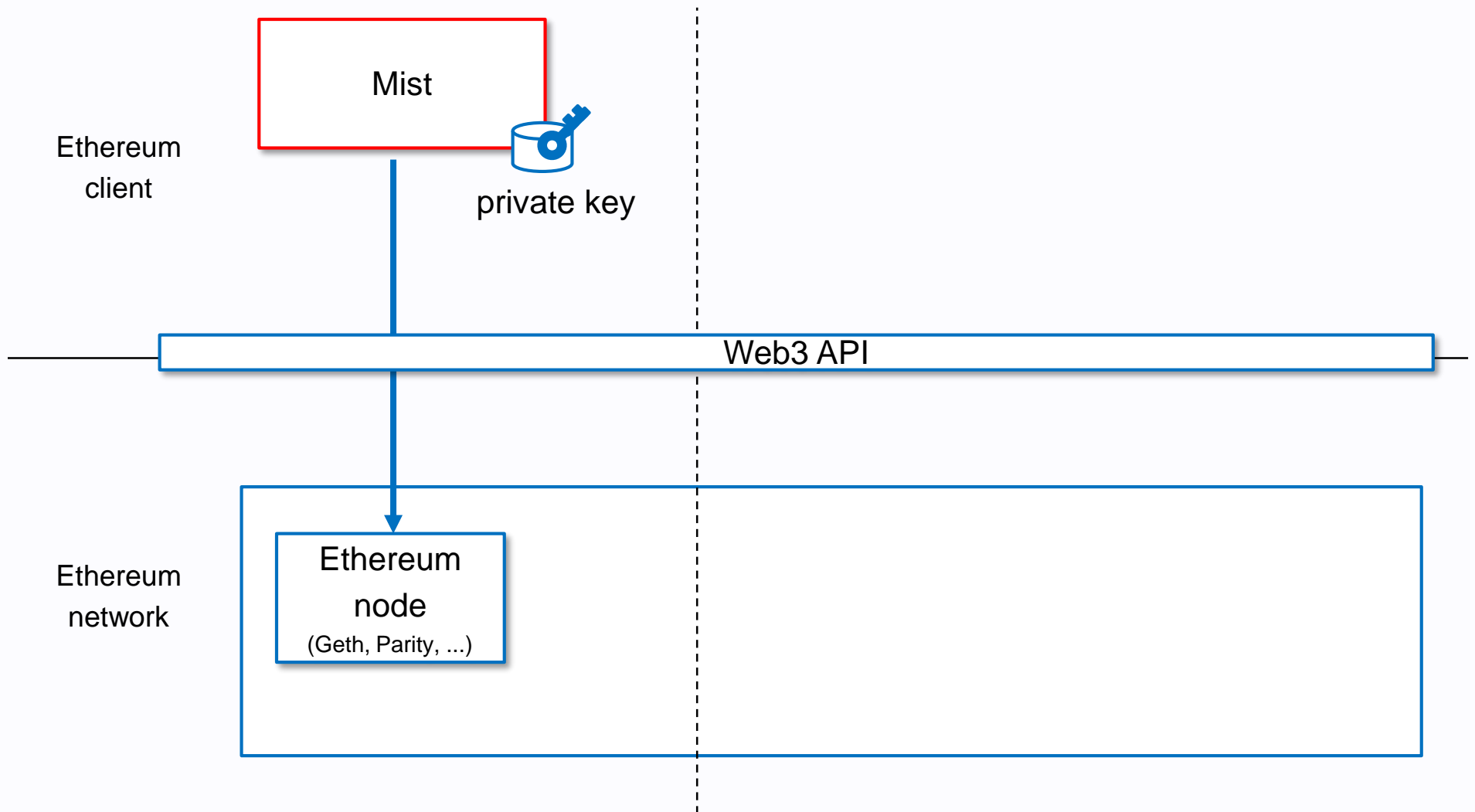
Appendix B

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

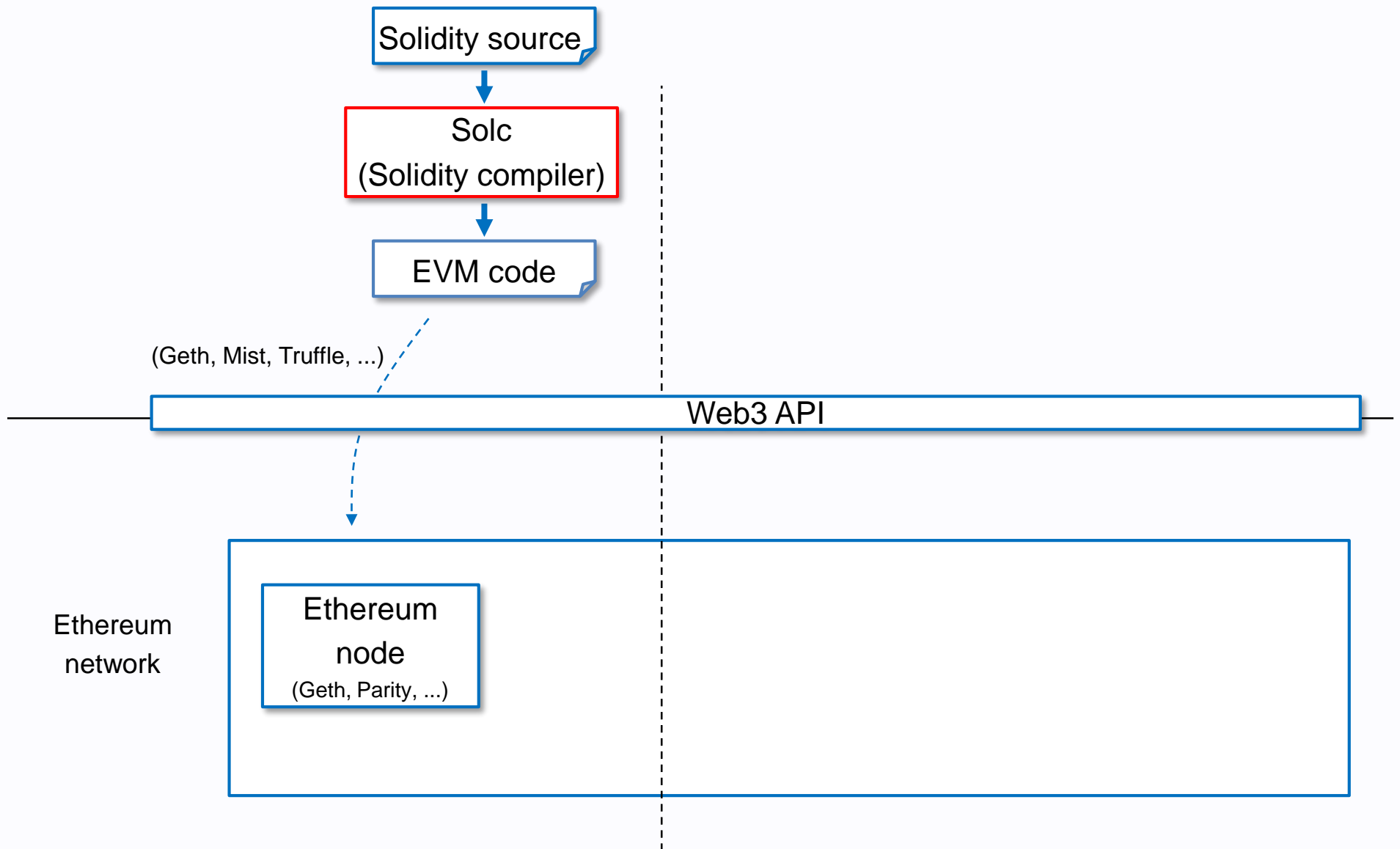
Geth



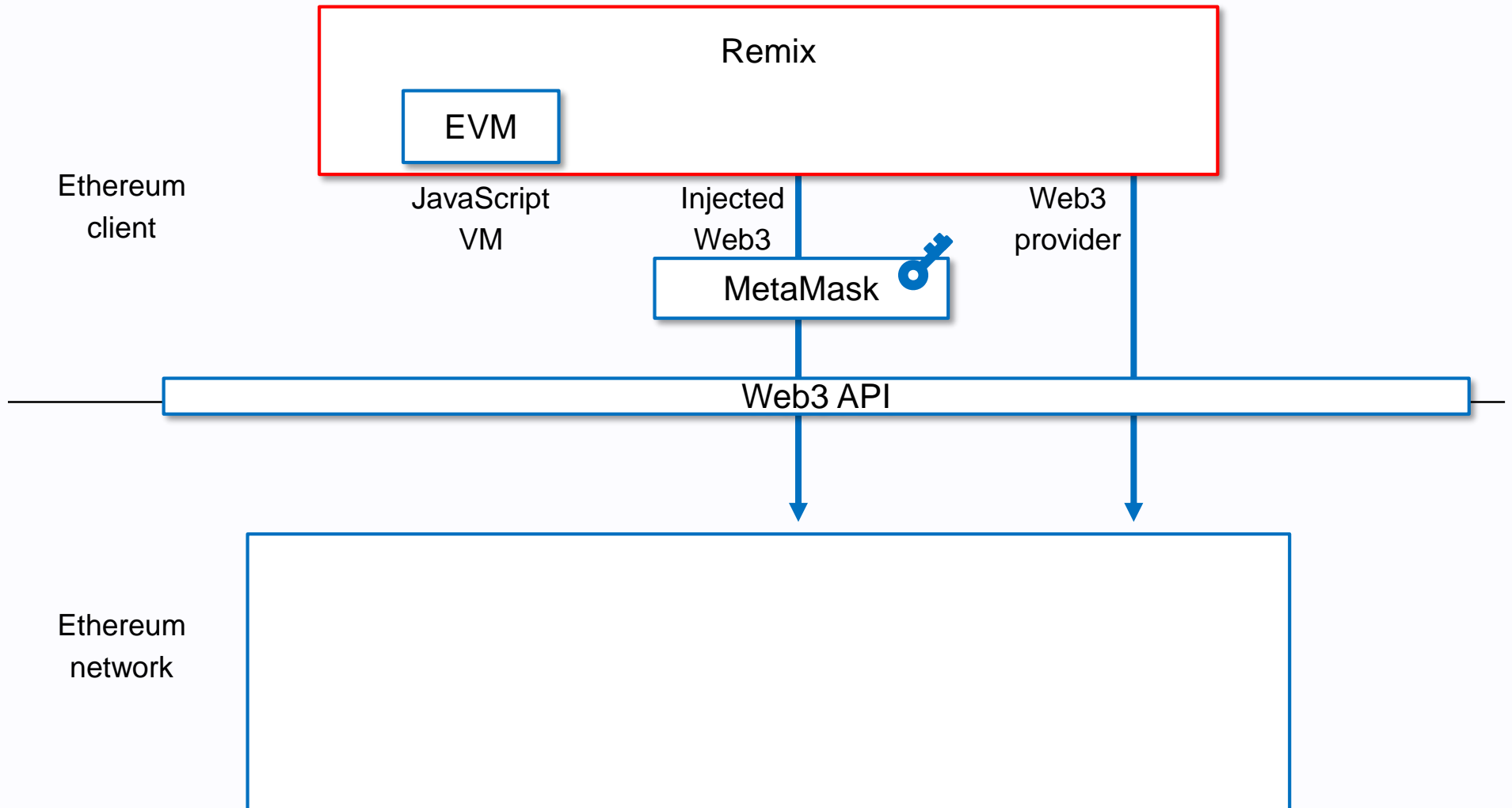
Mist



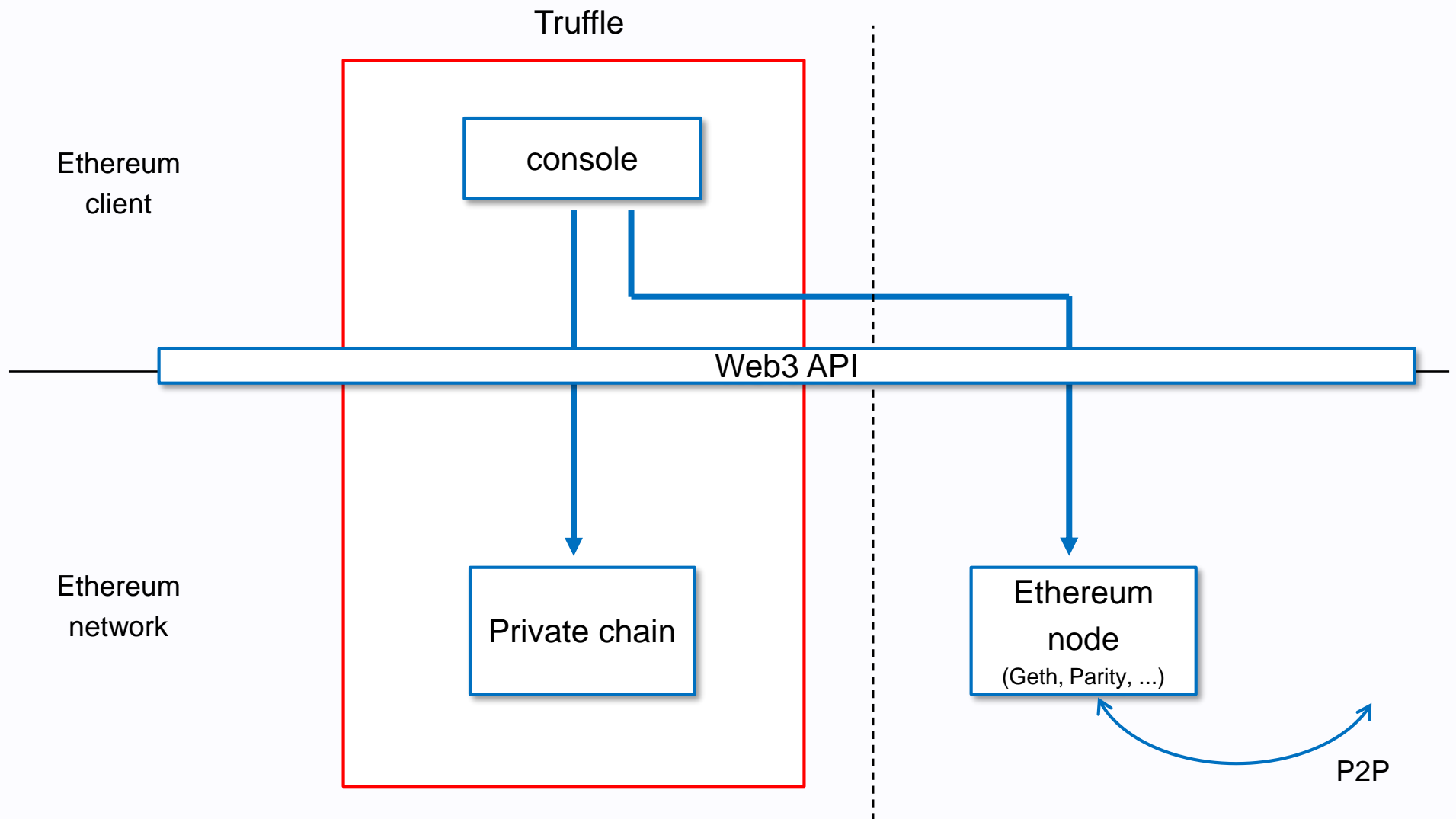
Solc



Remix



Truffle



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>

- [E9] ethereum/wiki Subtleties
<https://github.com/ethereum/wiki/wiki/Subtleties#exceptional-conditions>

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>

