

# Blockchain and Digital Currencies

## Lecture 8

PHBS 2024 M3

# Agenda

---

- Ethereum Tries

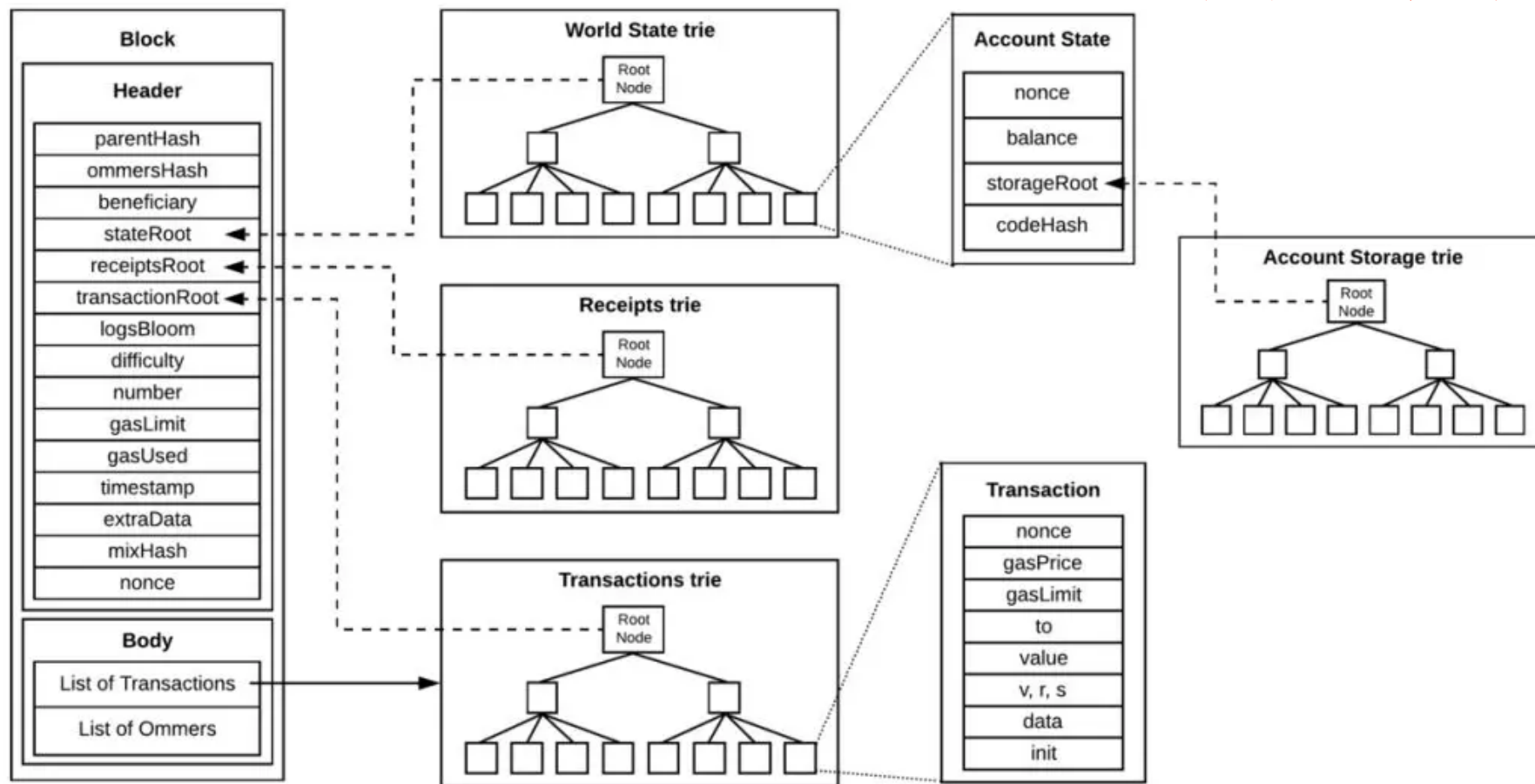
# Ethereum Tries

---

- State root
- Transaction root
- Receipt root
- Storage root

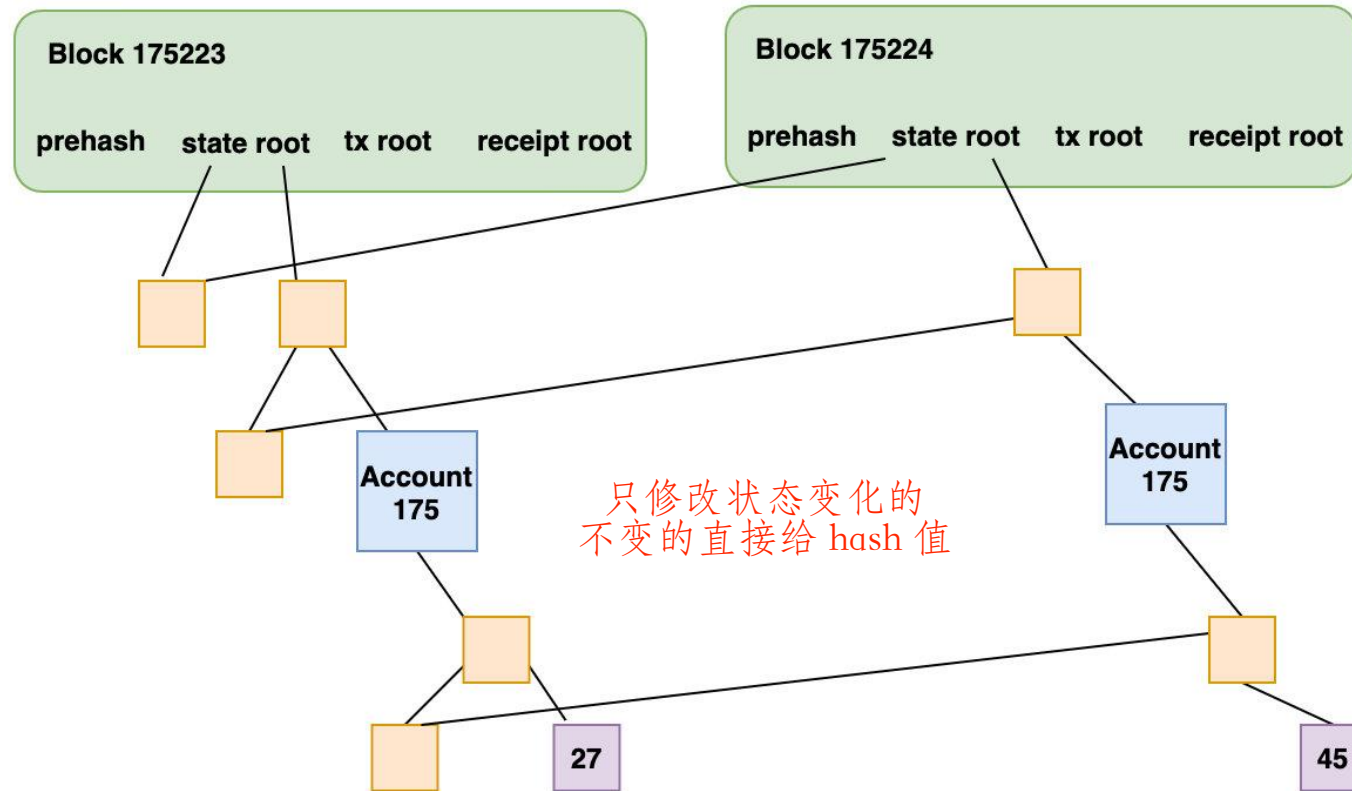
# Summary of Tries

每一个 Block 都是现实世界的一个快照



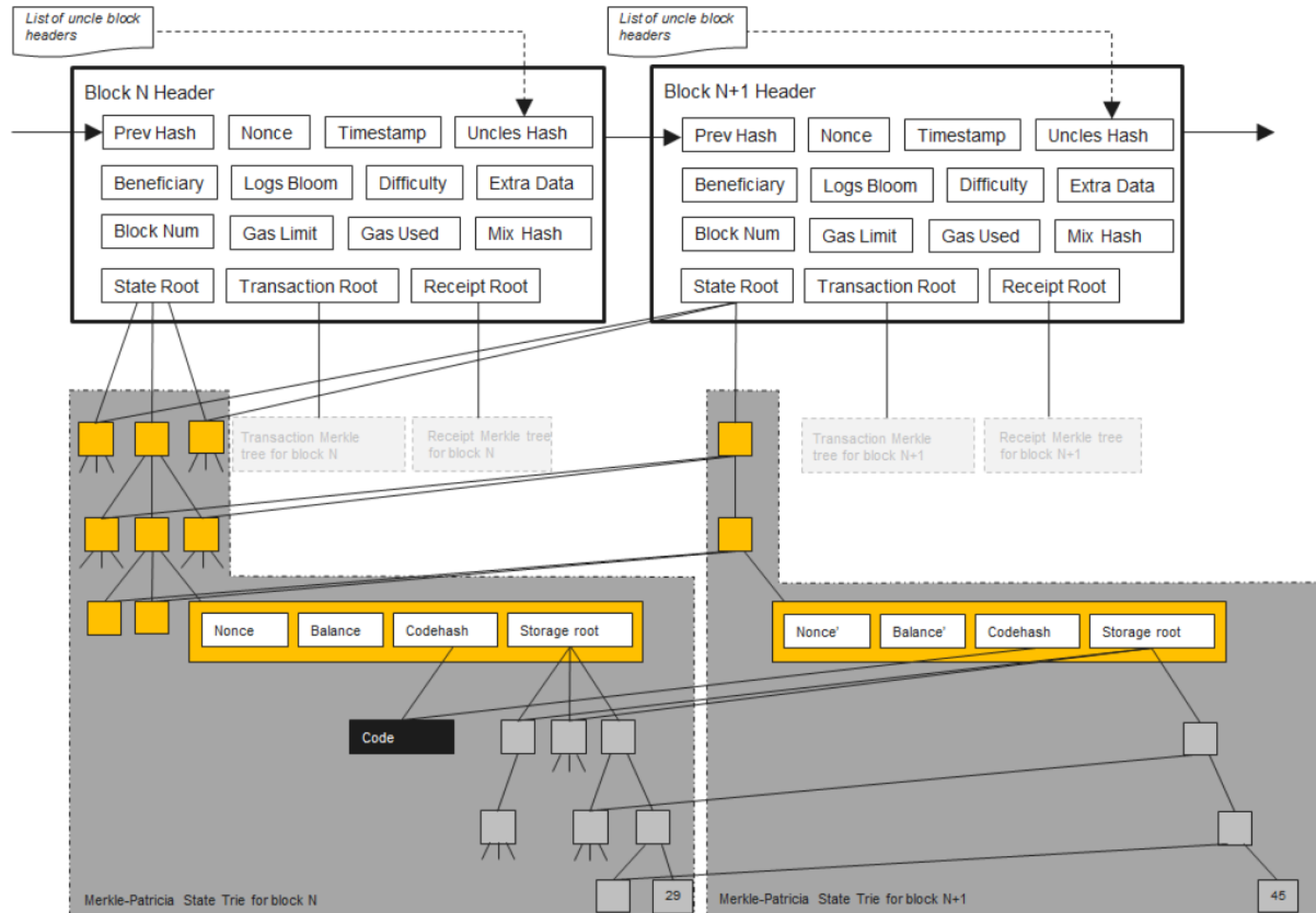
What is the purpose of the tries above?  
What are the benefits of using these tries?

# Information Saved in State Tries



The state root is the hash of the MPT of all accounts.  
A new block only contains modifications caused by transactions.  
Why such a structure? -- to support rollback

# A More Comprehensive View



# Block Header Information

---

```
69 // Header represents a block header in the Ethereum blockchain.
70 type Header struct {
71     ParentHash common.Hash    `json:"parentHash"      gencodec:"required"`
72     UncleHash   common.Hash    `json:"sha3Uncles"      gencodec:"required"`
73     Coinbase    common.Address `json:"miner"           gencodec:"required"`
74     Root         common.Hash    `json:"stateRoot"       gencodec:"required"`
75     TxHash       common.Hash    `json:"transactionsRoot" gencodec:"required"`
76     ReceiptHash  common.Hash    `json:"receiptsRoot"    gencodec:"required"`
77     Bloom        Bloom          `json:"logsBloom"       gencodec:"required"`
78     Difficulty   *big.Int       `json:"difficulty"      gencodec:"required"`
79     Number       *big.Int       `json:"number"          gencodec:"required"`
80     GasLimit     uint64         `json:"gasLimit"        gencodec:"required"`
81     GasUsed      uint64         `json:"gasUsed"         gencodec:"required"`
82     Time        *big.Int       `json:"timestamp"       gencodec:"required"`
83     Extra        []byte         `json:"extraData"       gencodec:"required"`
84     MixDigest    common.Hash    `json:"mixHash"         gencodec:"required"`
85     Nonce        BlockNonce     `json:"nonce"           gencodec:"required"`
86 }
```

# Block Header Information

```
69 // Header represents a block header in the Ethereum blockchain.
70 type Header struct {
71     ParentHash common.Hash `json:"parentHash" gencodec:"required"`
72     UncleHash   common.Hash `json:"sha3Uncles" gencodec:"required"`
73     Coinbase    common.Address `json:"miner" gencodec:"required"`
74     Root         common.Hash `json:"stateRoot" gencodec:"required"`
75     TxHash       common.Hash `json:"transactionsRoot" gencodec:"required"`
76     ReceiptHash  common.Hash `json:"receiptsRoot" gencodec:"required"`
77     Bloom        Bloom `json:"logsBloom" gencodec:"required"`
78     Difficulty   *big.Int `json:"difficulty" gencodec:"required"`
79     Number       *big.Int `json:"number" gencodec:"required"`
80     GasLimit     uint64 `json:"gasLimit" gencodec:"required"`
81     GasUsed      uint64 `json:"gasUsed" gencodec:"required"`
82     Time        *big.Int `json:"timestamp" gencodec:"required"`
83     Extra        []byte `json:"extraData" gencodec:"required"`
84     MixDigest    common.Hash `json:"mixHash" gencodec:"required"`
85     Nonce        BlockNonce `json:"nonce" gencodec:"required"`
86 }
```

叔父区块的哈希(后  
续进行详细讲解)

父区块的哈希(前一个区块的哈希值)

矿工地址 → Coinbase

三棵树的  
根哈希值 → Root, TxHash, ReceiptHash

挖矿难度 → Difficulty

布隆过滤器(用于查询, 和收据树相关)

汽油费相关 → GasLimit, GasUsed

区块大致产生时间 → Time

和挖矿过程相关 → Nonce

<https://blog.csdn.net/qbq/v4r156959d>



# Block Information

```
144 // Block represents an entire block in the Ethereum blockchain.
145 type Block struct {
146     header      *Header
147     uncles       []*Header
148     transactions Transactions
149
150     // caches
151     hash atomic.Value
152     size atomic.Value
153
154     // Td is used by package core to store the total difficulty
155     // of the chain up to and including the block.
156     td *big.Int
157
158     // These fields are used by package eth to track
159     // inter-peer block relay.
160     ReceivedAt time.Time
161     ReceivedFrom interface{}
162 }
```

Published block struct

<https://blog.gcpd.com>

Complete block struct

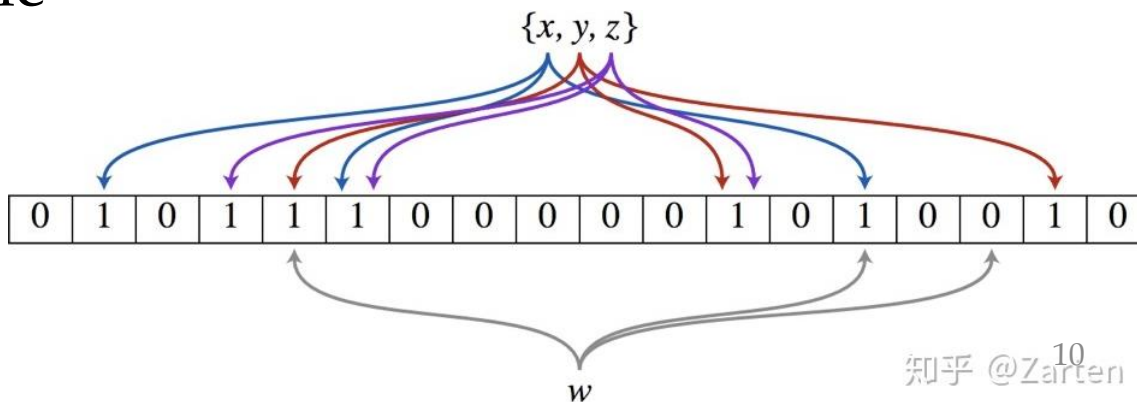
```
177 // "external" block encoding. used for eth protocol, etc.
178 type extblock struct {
179     Header *Header
180     Txs     []*Transaction
181     Uncles  []*Header
182 }
```

# Bloom Filter

- A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set.
  - Easy to check the **non-existence** of elements
  - False positive matches are possible
  - Very hard to delete elements

如果给出 100% 不在，那就肯定不在  
但是判断在只能是 maybe

因为判断地址是很重要的操作必须要快速准确



# Bloom Filter

- Demonstration from <https://lmlib.github.io/bloomfilter-tutorial/>
- Easy to check the existence of elements but very hard to delete

The base data structure of a Bloom filter is a **Bit Vector**. Here's a small one we'll use to demonstrate:

|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

It's easier to see what that means than explain it, so enter some strings and see how the bit vector changes. Fnv and Murmur are two simple hash functions:

Enter a string:

fnv: 13  
murmur: 10

Your set: [ab, abc, abcd, bc, bc, ab, abc]

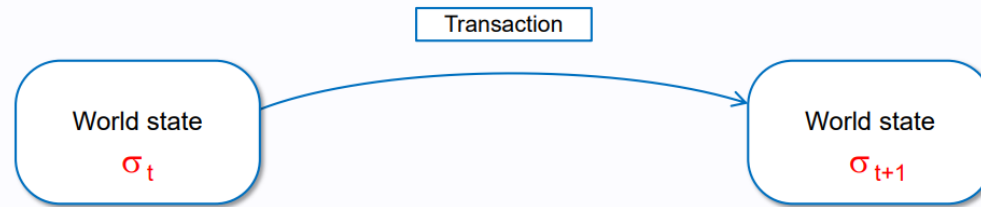
What might be an application of this data structure?

# World State Revisited

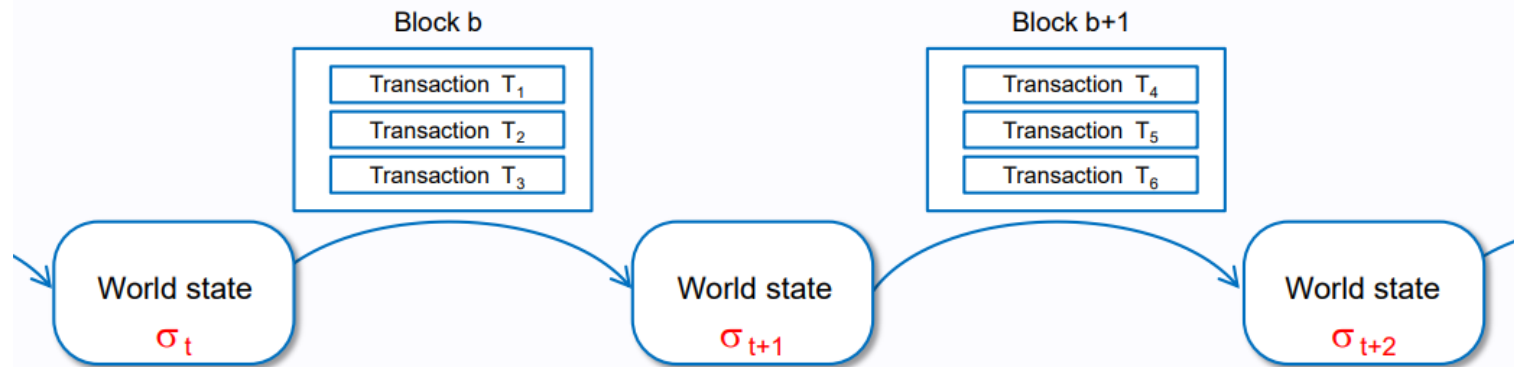
---

- The following slides are from a very good reference:
- [https://takenobu-hs.github.io/downloads/ethereum\\_evm\\_illustrated.pdf](https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf)

# World State Revisited



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

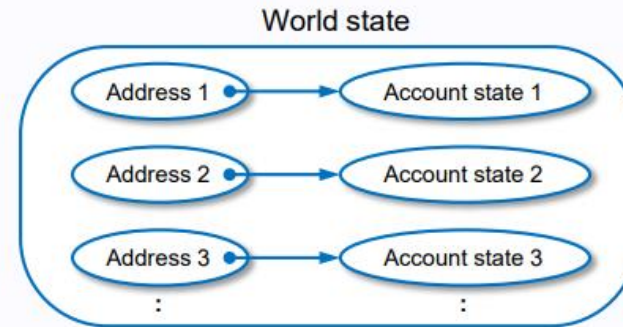


Block b 是和新的  
world state 存在一起的  
以方便验证

# Different Views of World State

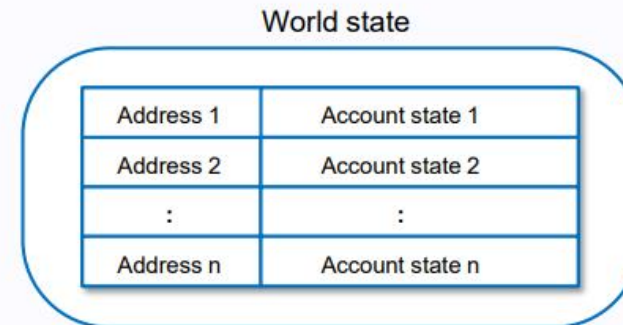
Conceptual view

Mapping view



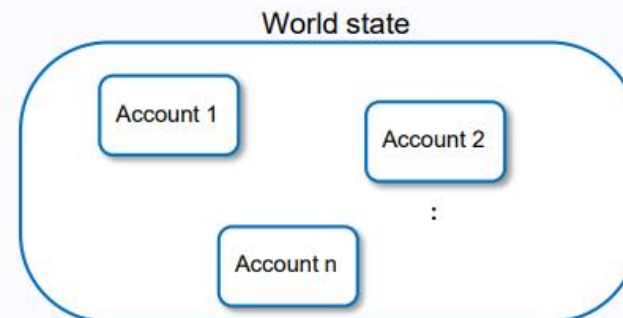
Database view

Table view



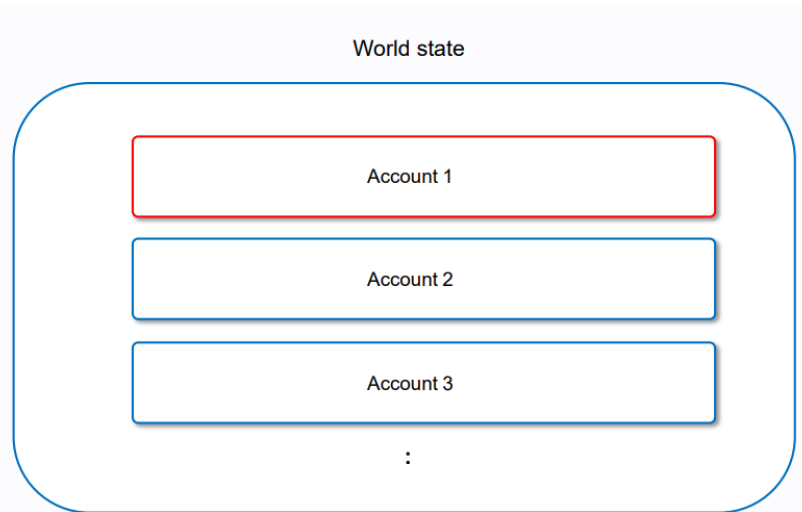
Coding view

Object view



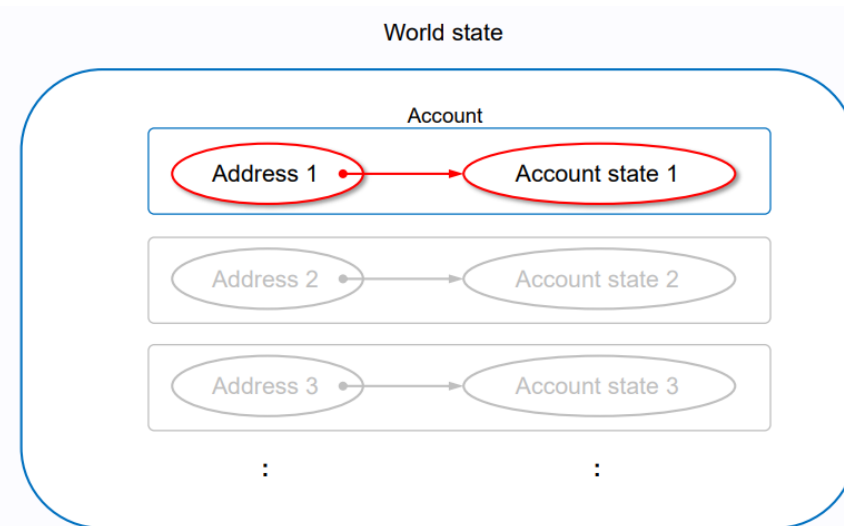
# Accounts

## Coding view



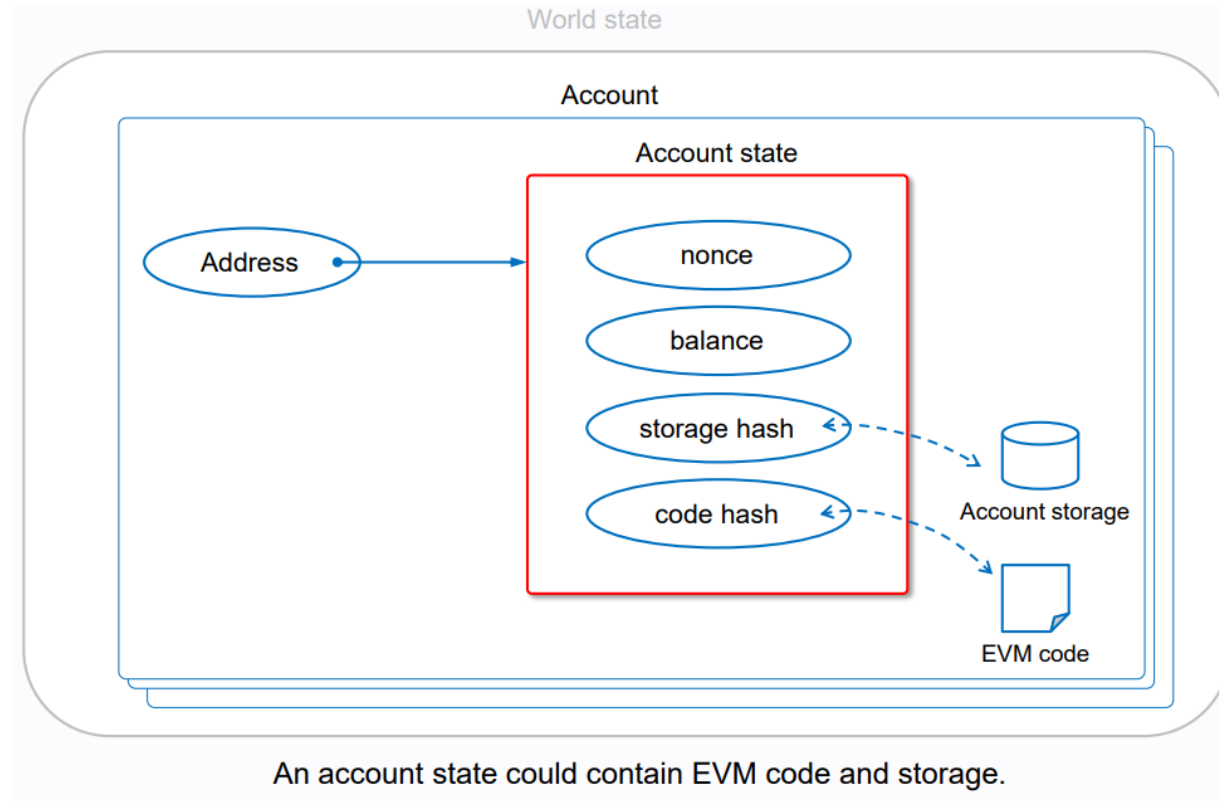
An account is an object in the world state.

## Conceptual view



An account is a mapping between address and account state.

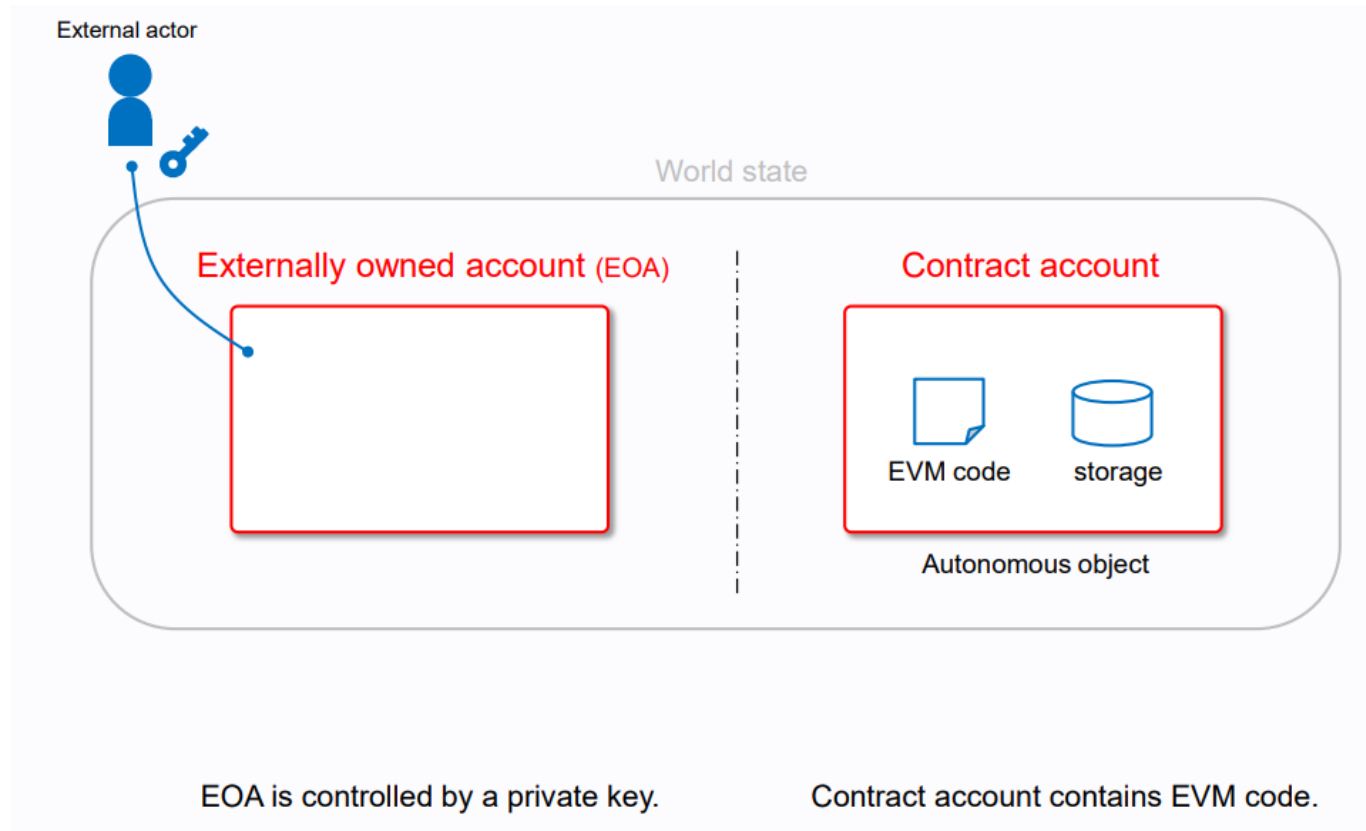
# Account State



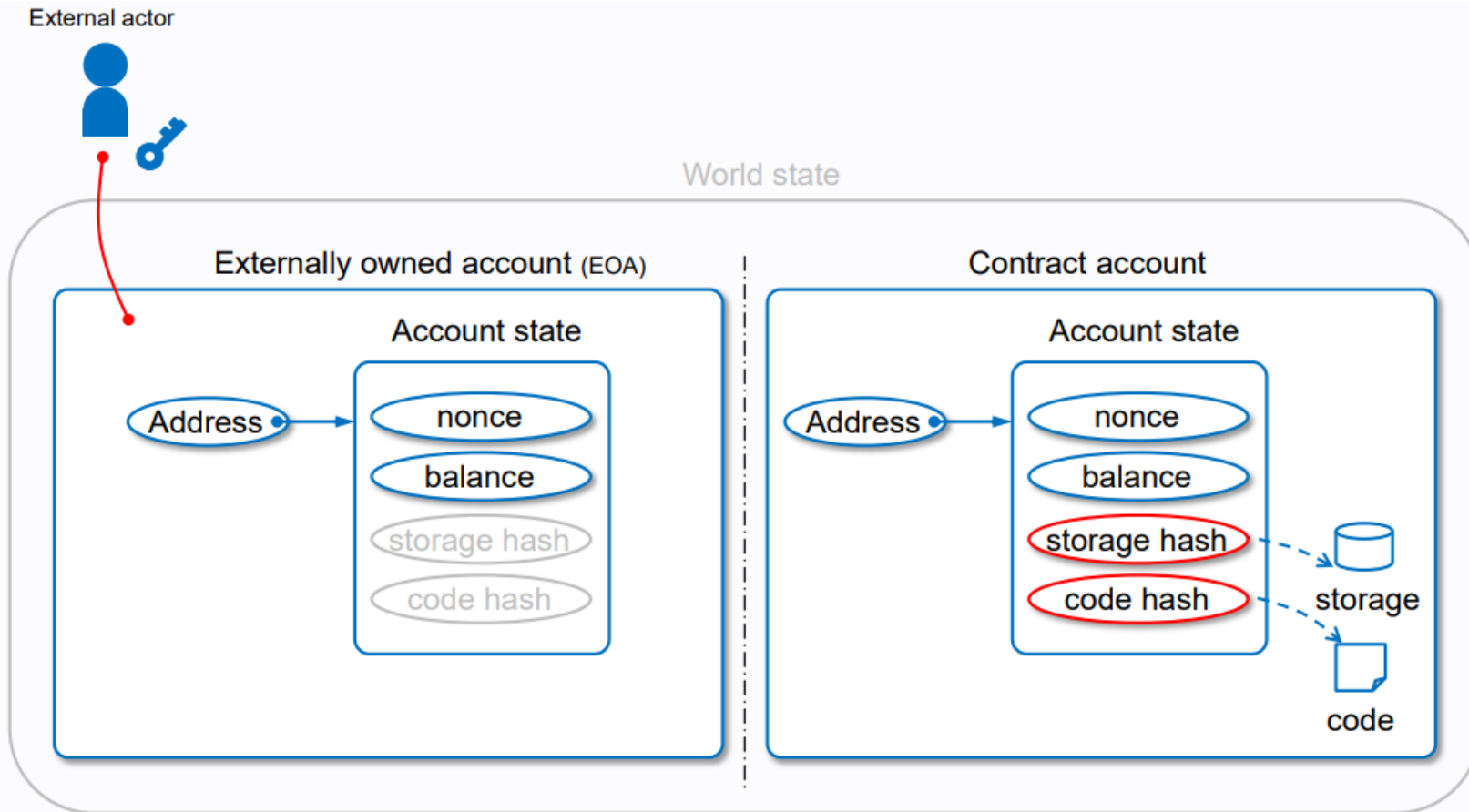
Only applicable for contract accounts!



# Accounts Revisited



# Accounts: EOA vs. Contract

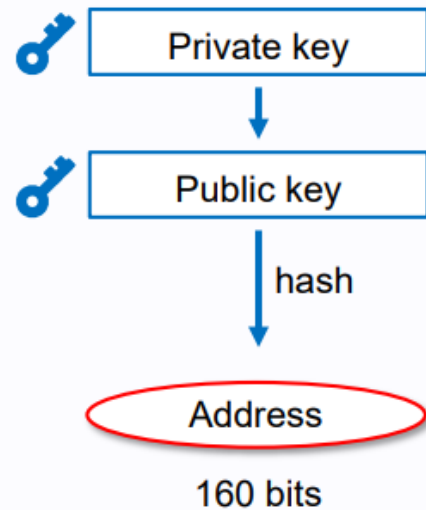


EOA is controlled by a private key.  
EOA cannot contain EVM code.

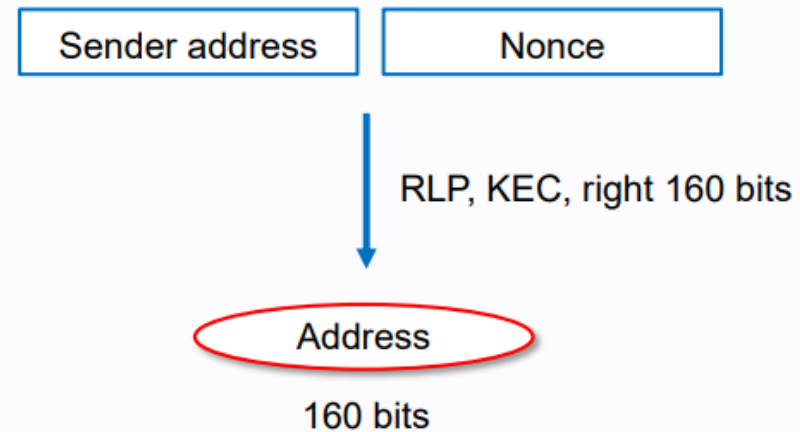
Contract contains EVM code.  
Contract is controlled by EVM code.

# Address of Accounts <sup>Text</sup>

## Externally owned account (EOA)

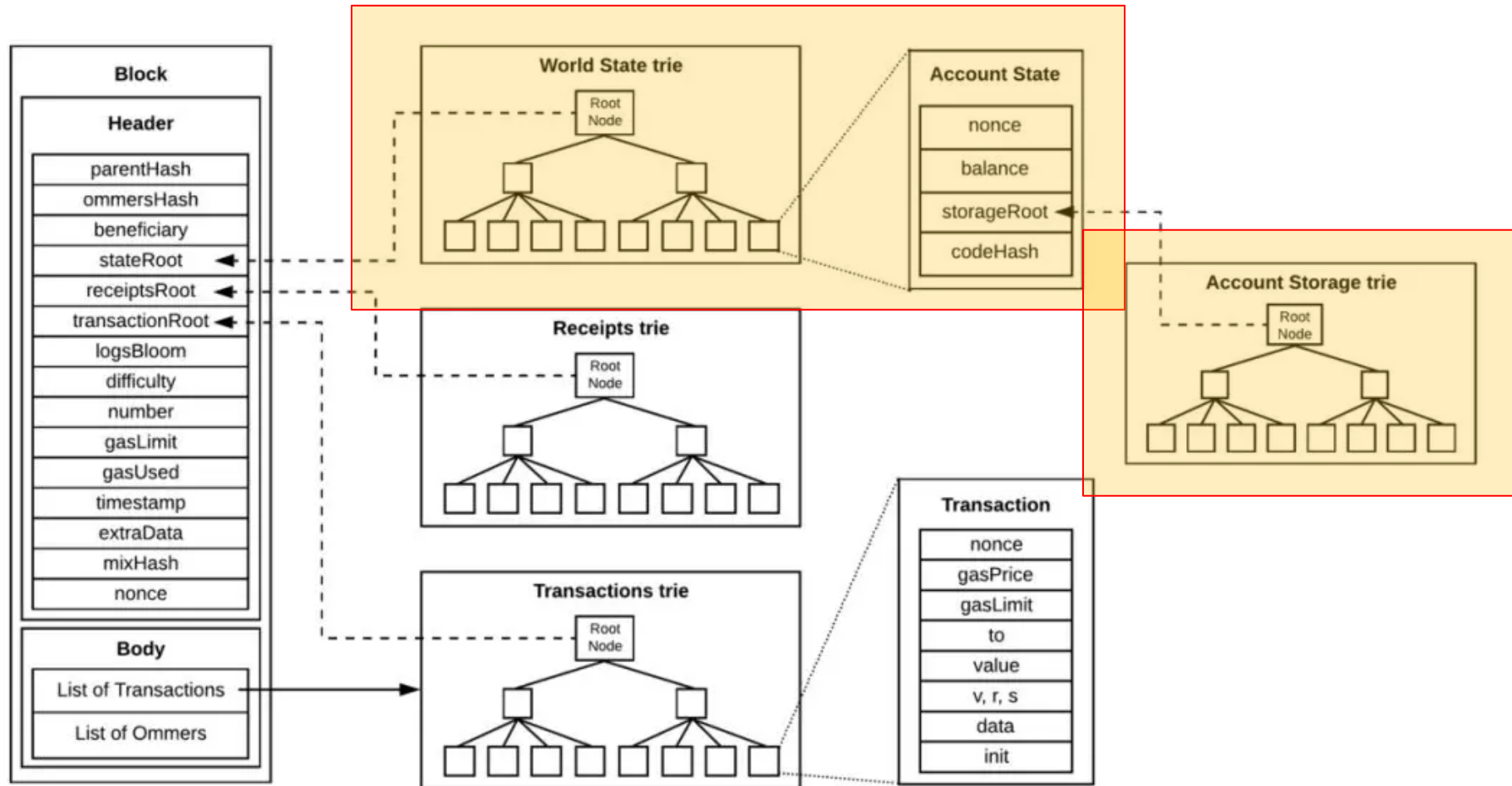


## Contract account



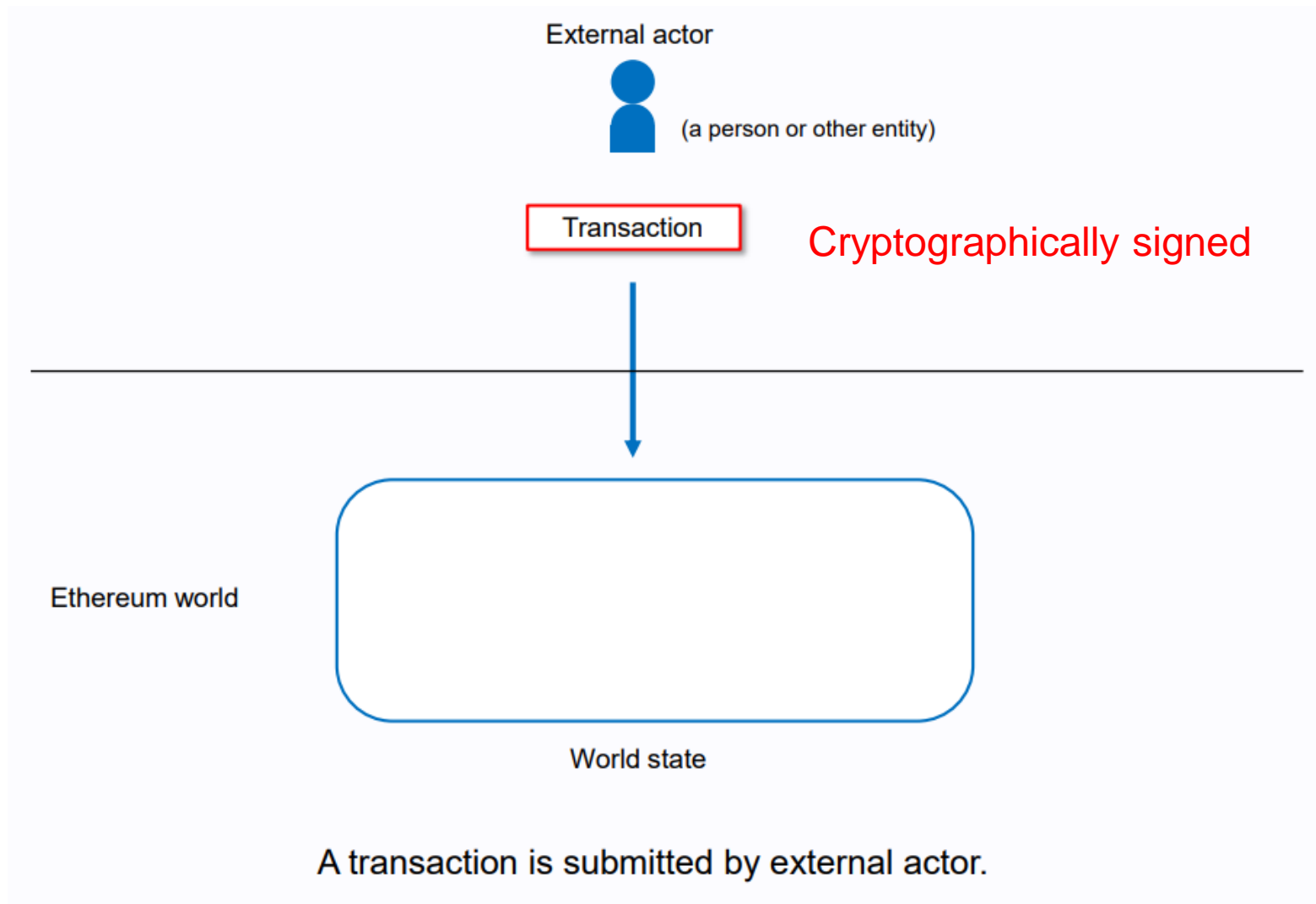
A 160-bit code used for identifying accounts.

# Summary of Tries Revisited

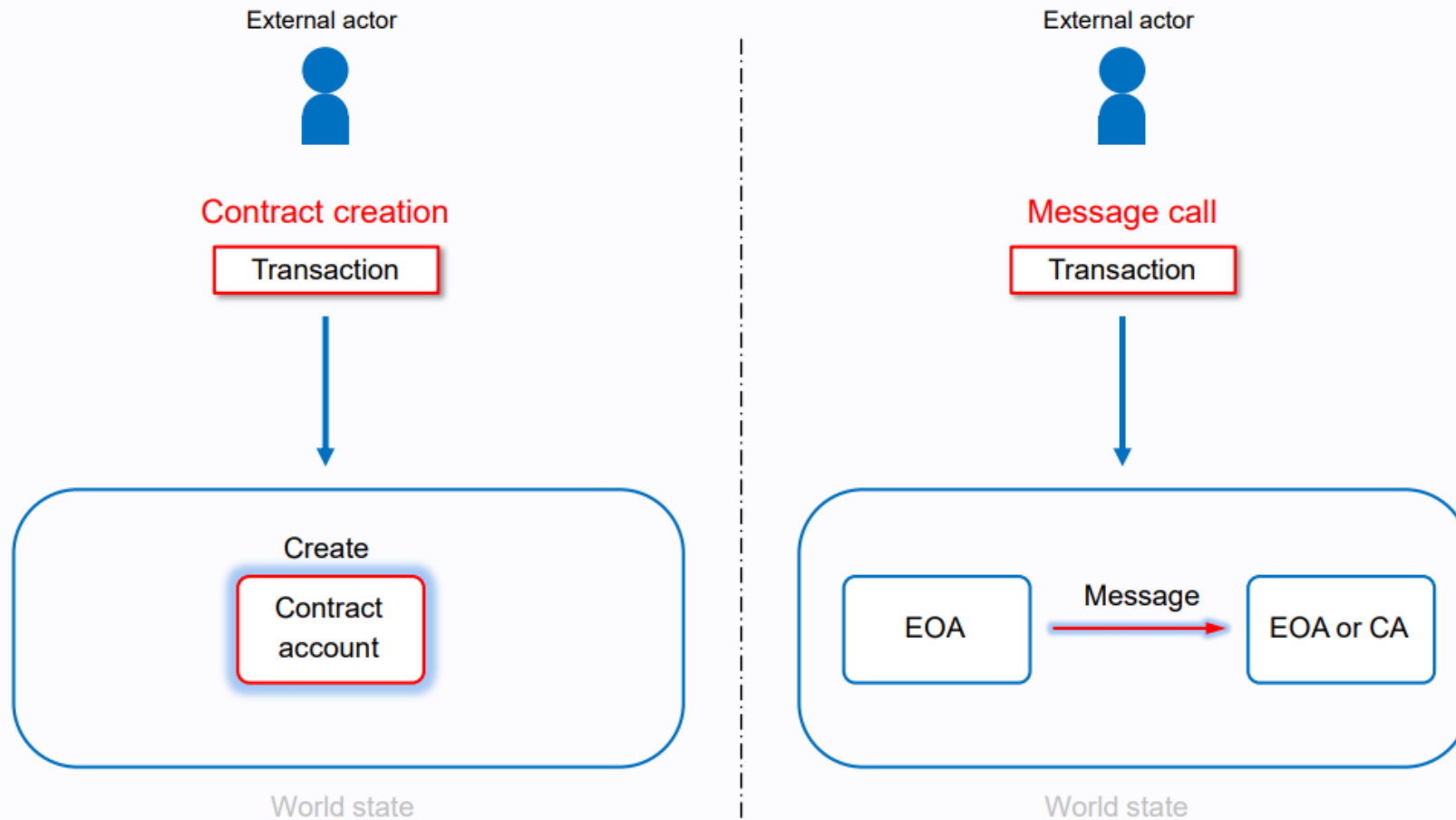


What might be a good data structure for Account Storage trie?

# Transactions

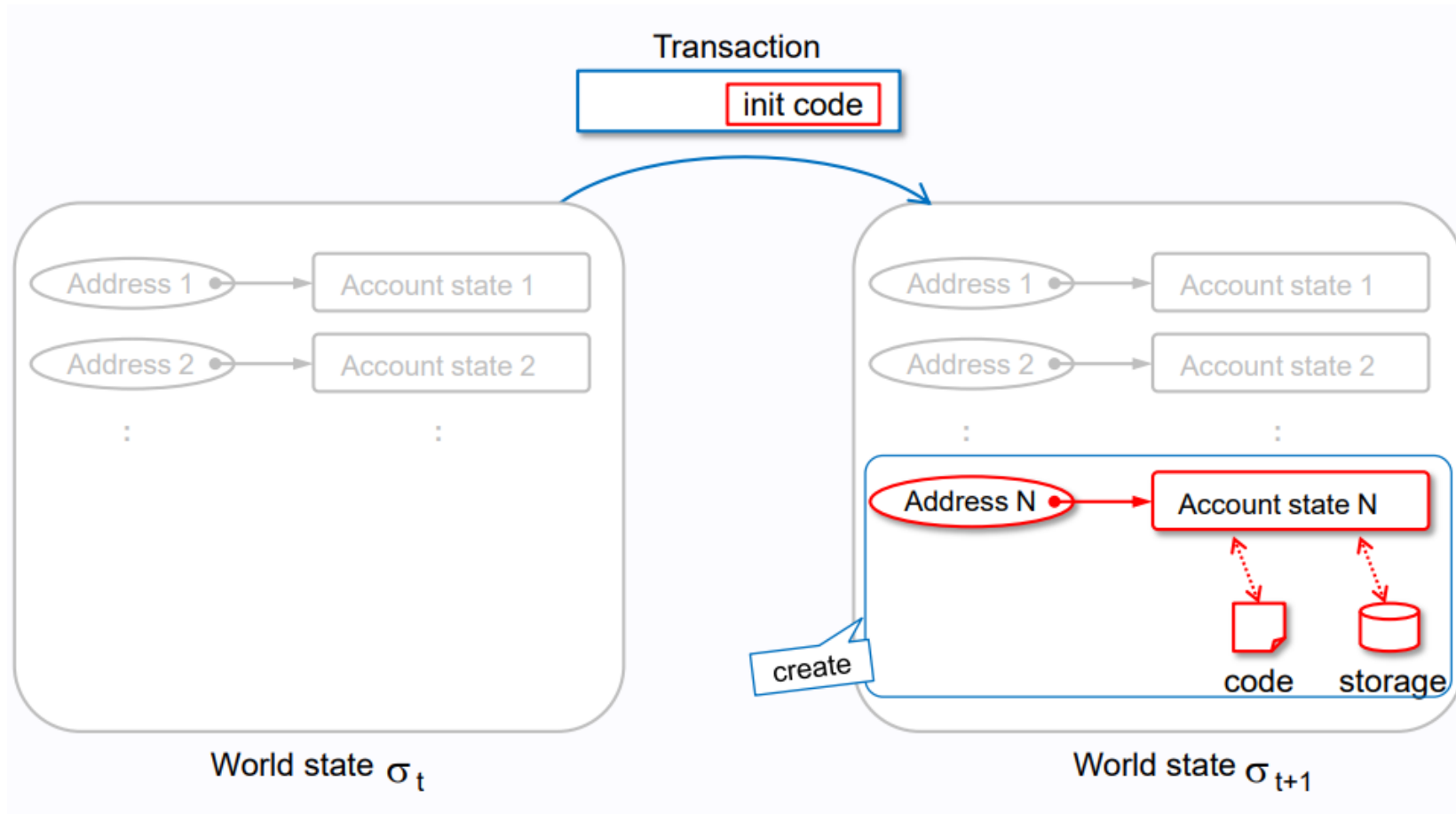


# Special Transactions for Smart Contracts



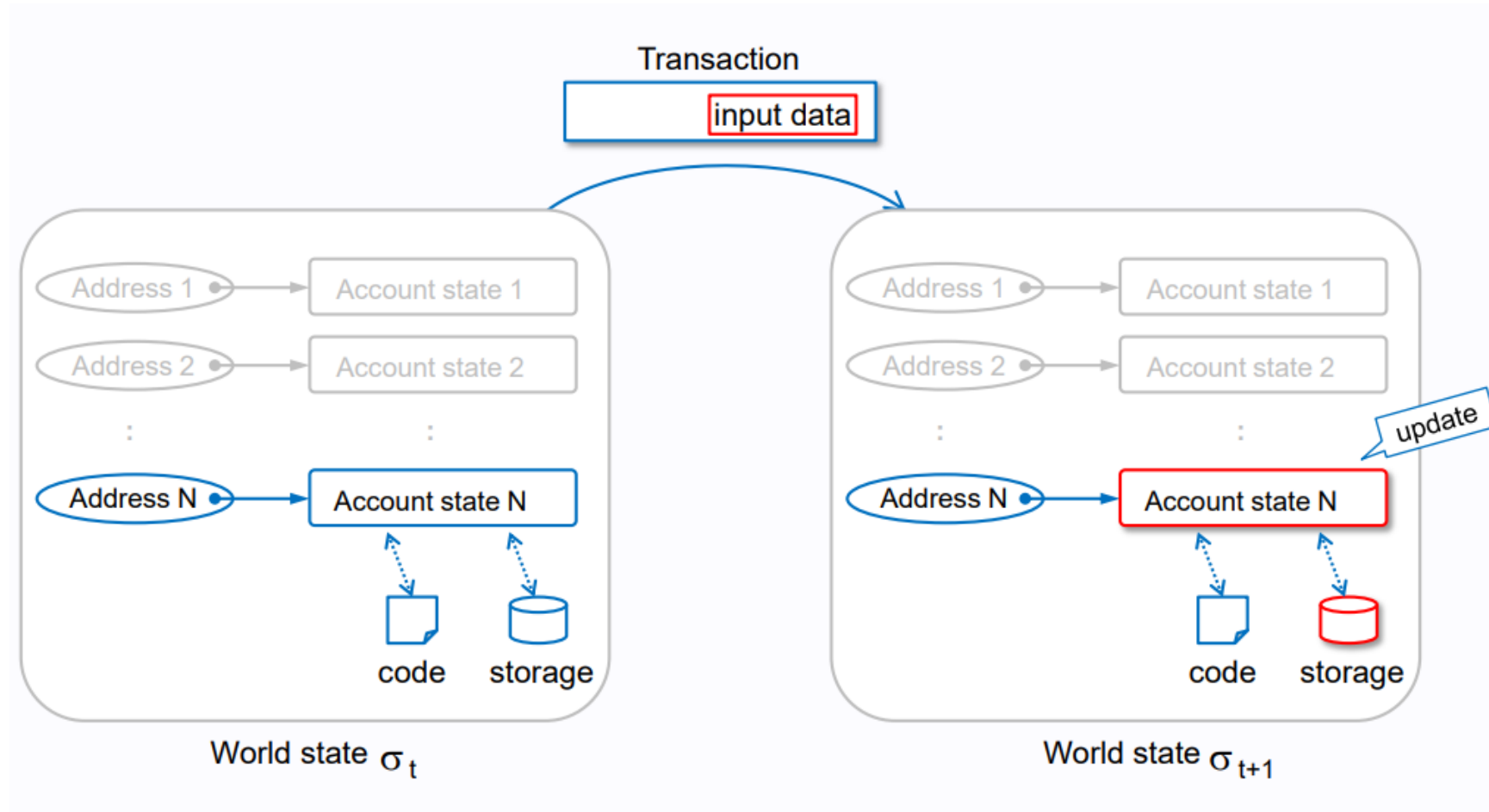
There are two practical types of transaction, contract creation and message call.

# Contract Creation



Creating a computer/server on the internet

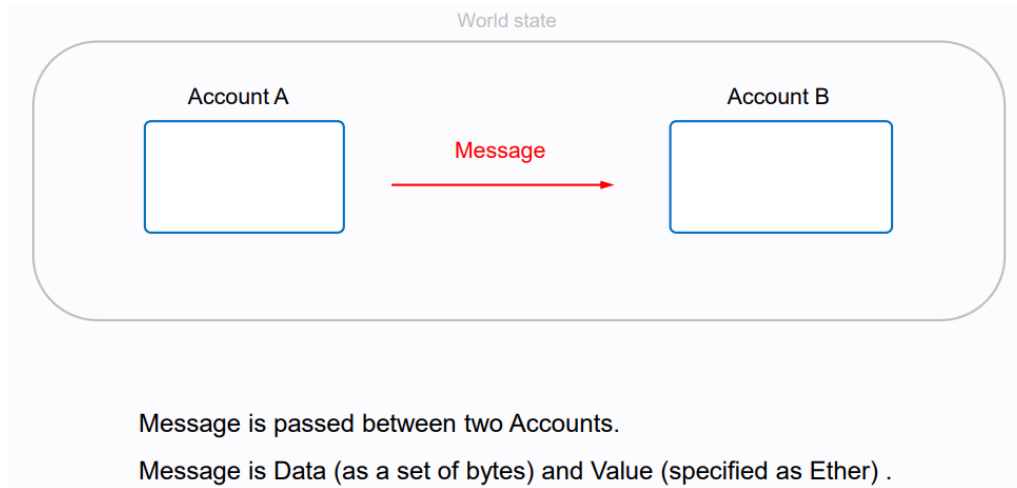
# Message Call to Invoke Contract Code



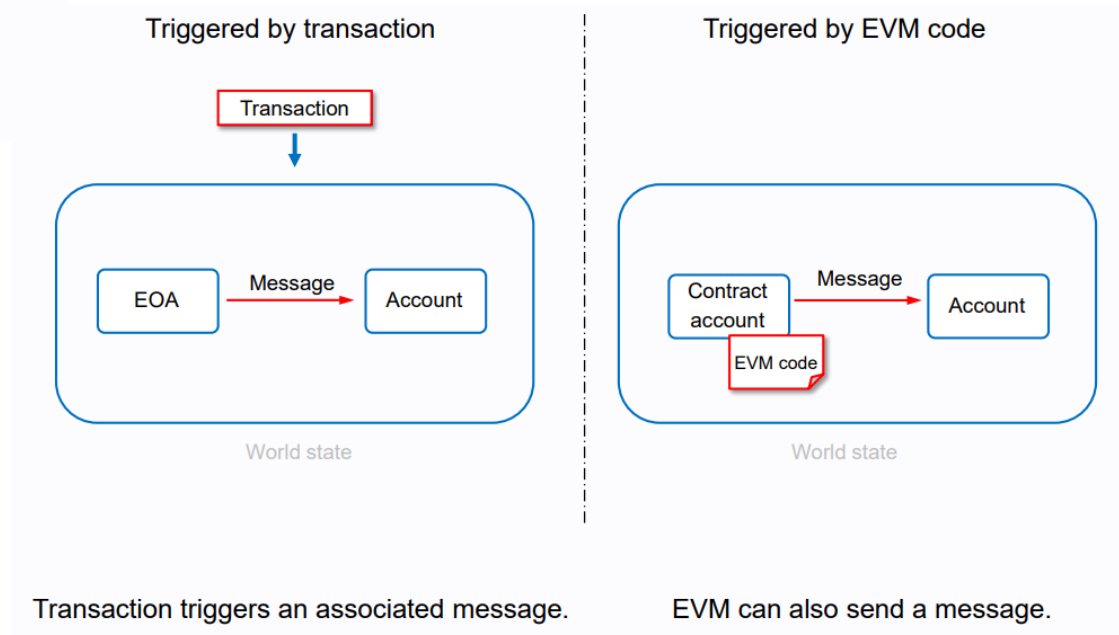
Running a computer on the internet to update the world state



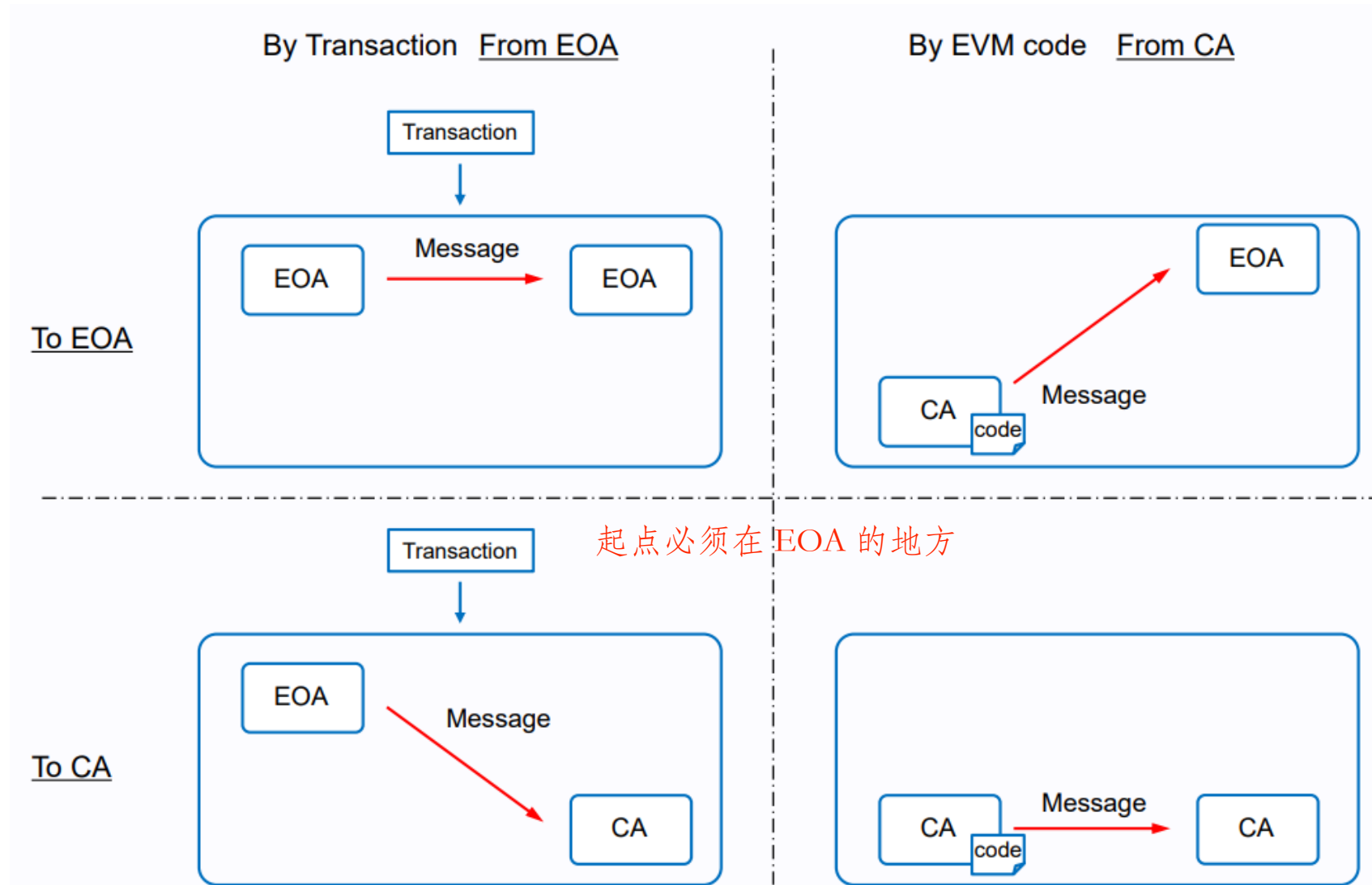
# Messages



The simplest one is  
money transfer  
transaction

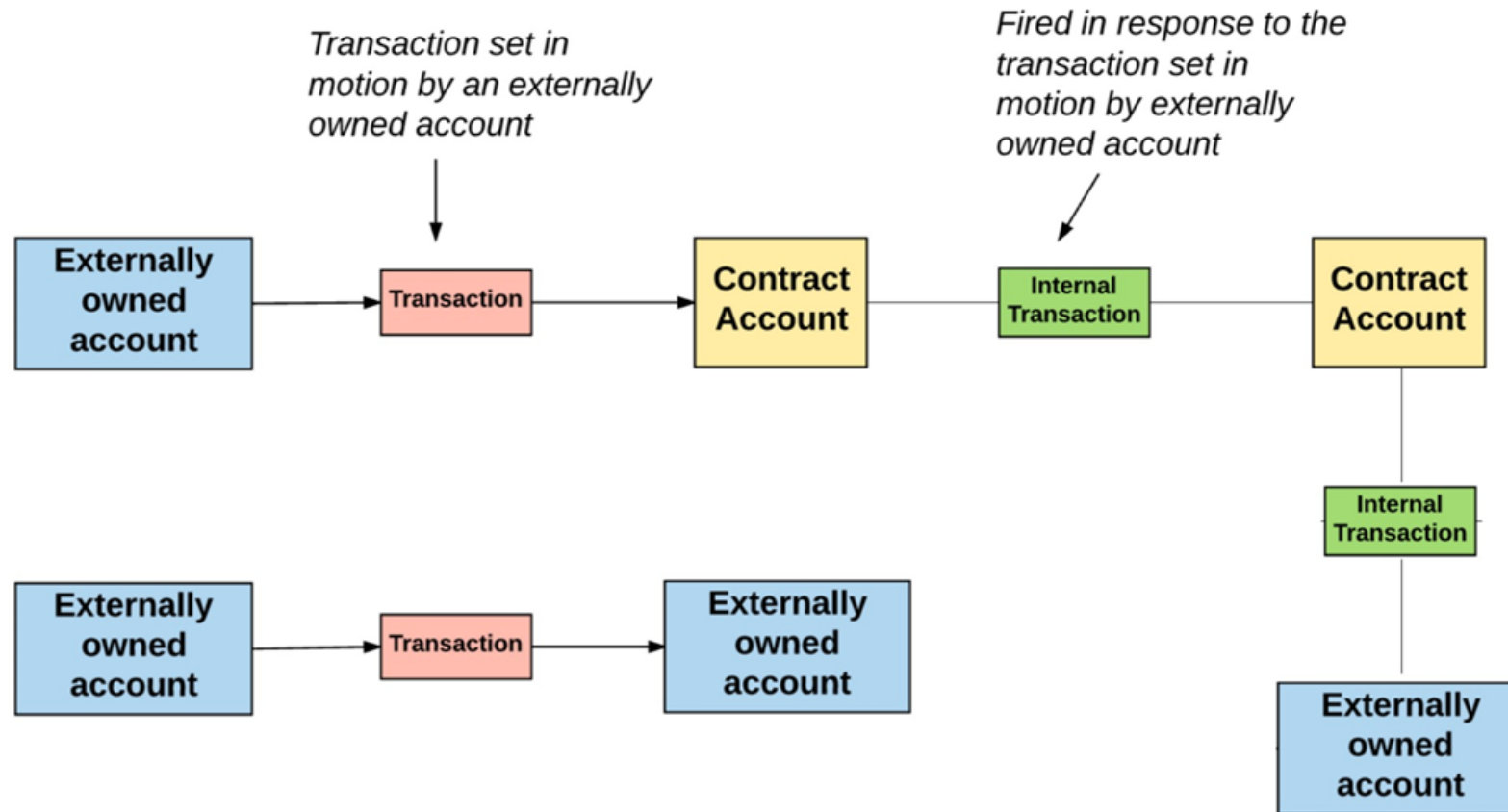


# Message Calling



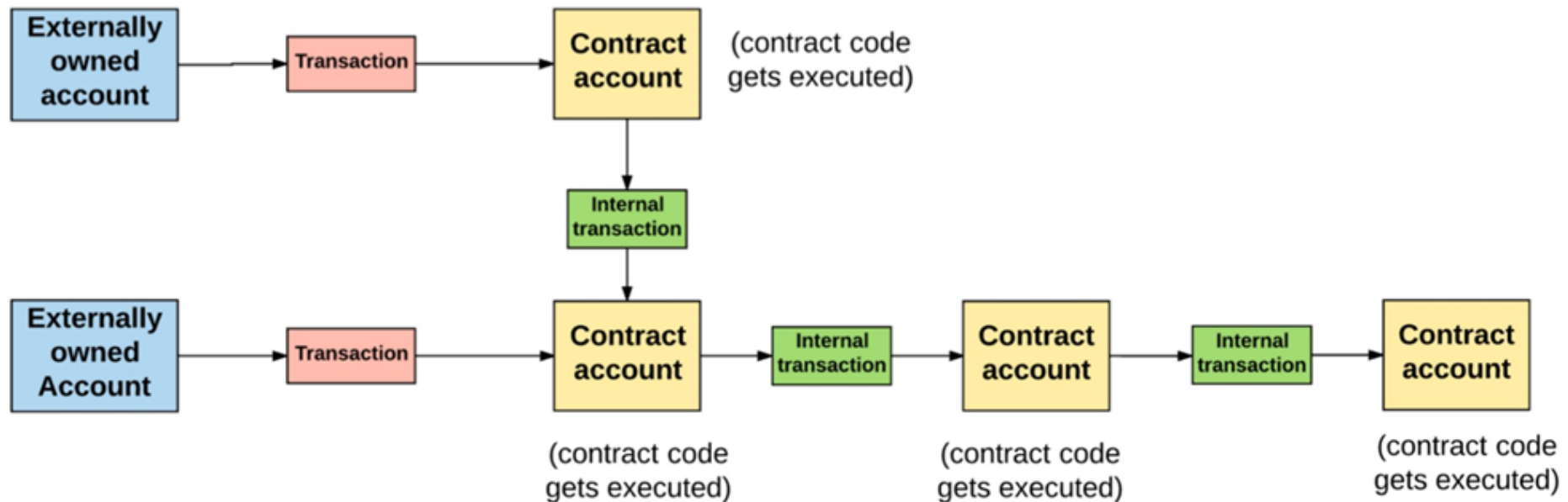
# Message Calling Chain

---



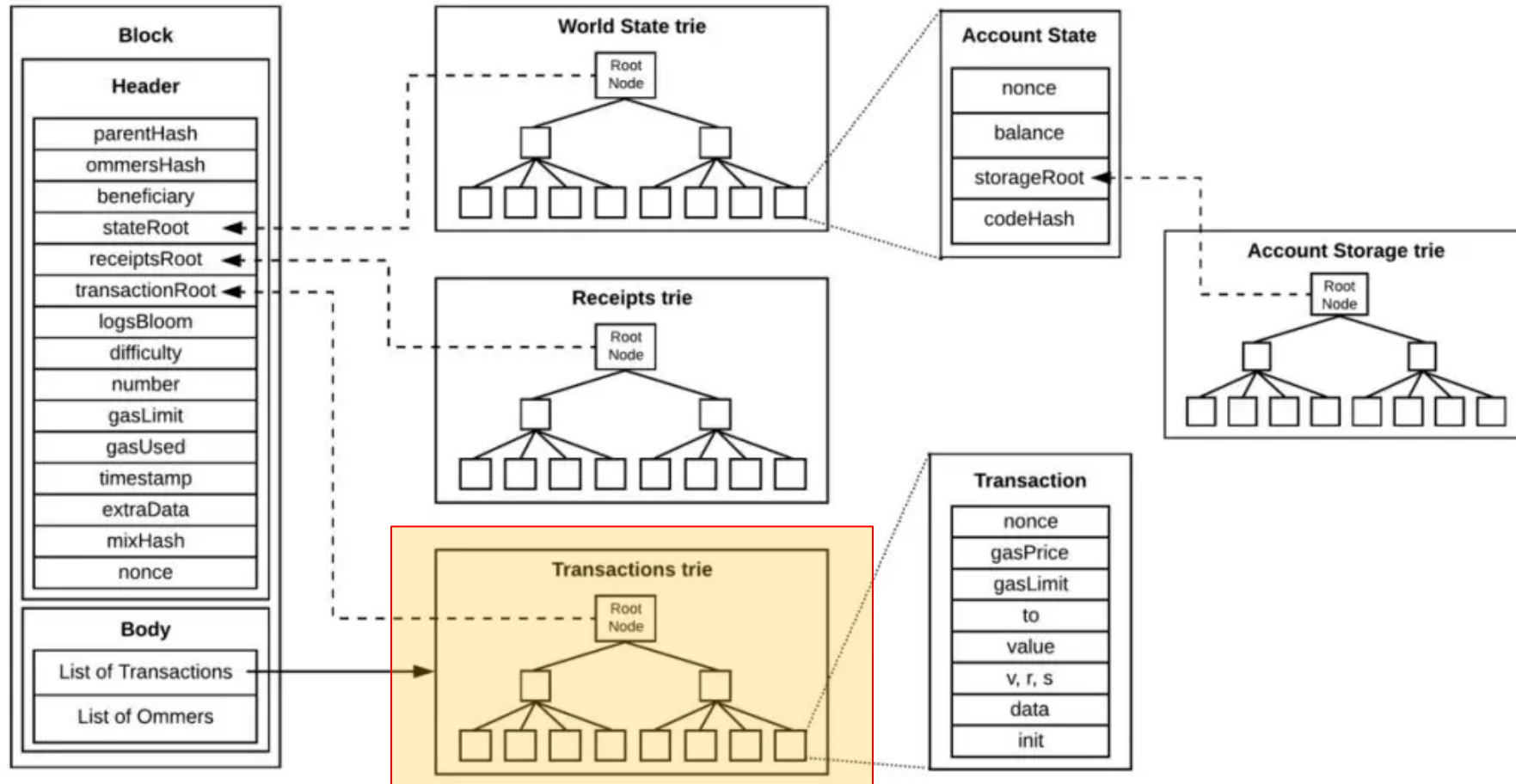
# Another Calling Chain Example

---



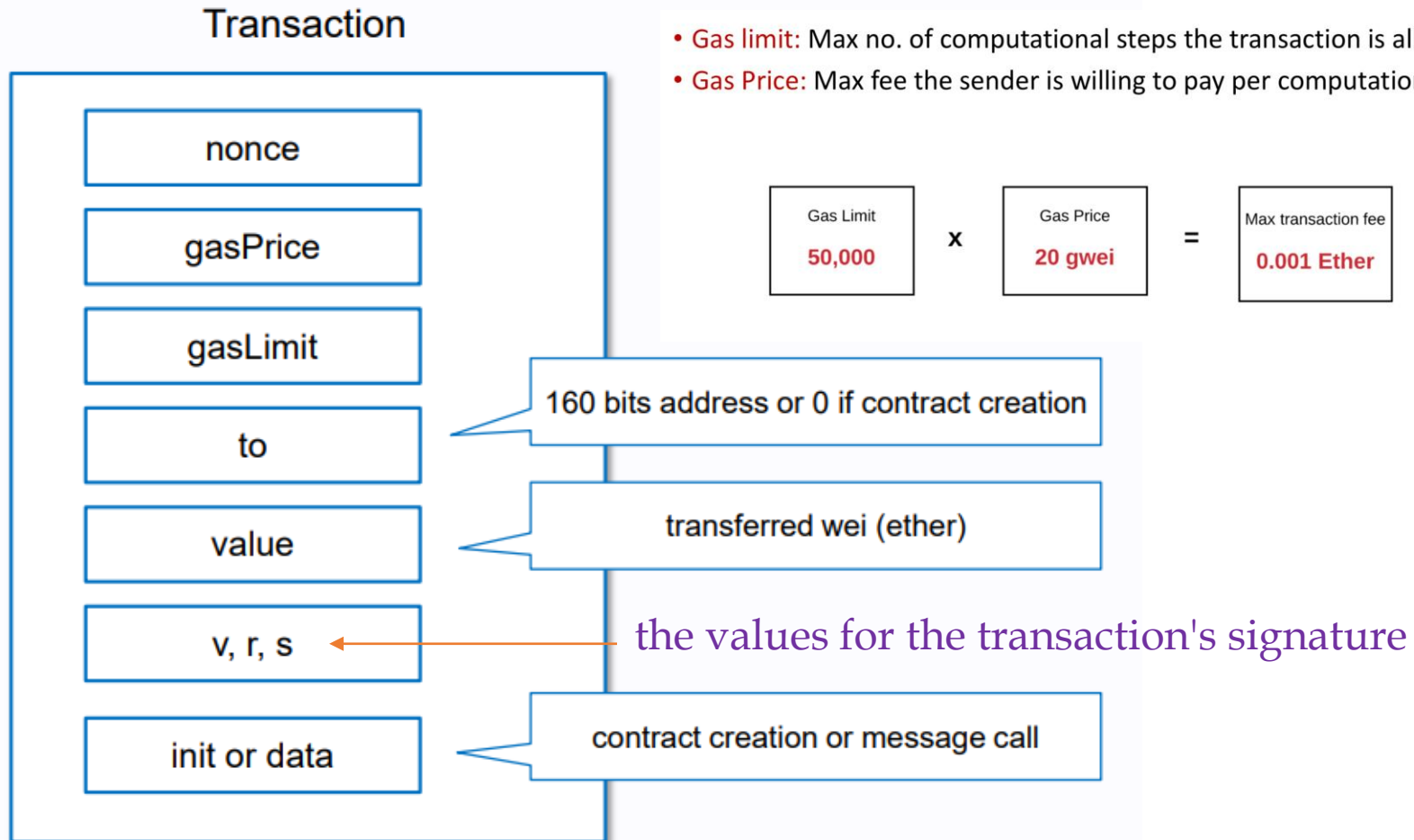
Can you think an application of this design? – Multisig?

# Summary of Tries Revisited



Note that transaction tries independent across different blocks.

# Fields of Transactions



# Receipt Trie

---

- Why do we need receipt trie?
  - Will state trie be sufficient?
  - Will state trie + transaction trie be sufficient?
- Transaction receipt tries record **transaction outcome**
  - [details in section 4.4.1 of the [yellow paper](http://gavwood.com/paper.pdf)]:  
<http://gavwood.com/paper.pdf>
  - ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER (EIP-150 REVISION)

# Receipt Trie Parameters

---

- Transaction receipt tries record includes:
  - post-transaction state,
  - the cumulative gas used,
  - the set of logs created through execution of the transaction, and
  - the Bloom filter composed from information in those logs
- Transaction receipt tries always accompany transaction tries
  - Therefore independent across different blocks
  - Useful to collectively handle frequent query cases conveniently



# Quiz: Which Trie to Use?

---

- Different tries are used under different scenarios:
  - What is the current balance of a given account? (S)
  - Does a given account exist? (S)
  - Given a transaction, figure out whether it has been included in a particular block? (T)
  - Tell me all instances of an event of type X (eg. a crowdfunding contract reaching its goal) emitted by a particular address in the past 30 days (R: through the set of logs stored in receipt tries)