深入理解以太坊

——以太坊黄皮书精简教程

杨 镇

2018/5/13

HiBlock区块链社区 HiBlock区块链课堂006期

自我介绍

HiBlock 区块链社区

- 16年的软件行业从业经验，包含：企业级信息化系统建设（架构设计、开发、测试、项目管理、产品化实施、运维）；软件过程改进（CMMI、Agile、DevOps）；企业级云计算产品架构、研发（SaaS，垂直PaaS）等。
- 10年以上的一线程序开发、代码评审、测试、部署和运维经验。
- 3年互联网创业经验。
- 2013年开始接触比特币，2016年底开始研究以太坊，2017年4月到11月利用业余时间独立翻译了【Ethereum Homestead Documentation】。
- 2017年底开始参与区块链/以太坊技术社区。
- 目前的个人工作、兴趣方向：企业级云计算产品架构、企业级区块链应用、智能合约应用、区块链/以太坊技术社区贡献。
- 个人Github：https://github.com/riversyang
- 个人中文博客：https://www.jianshu.com/u/726933951c63，（风静縠纹平@简书）

大纲

HiBlock 区块链社区

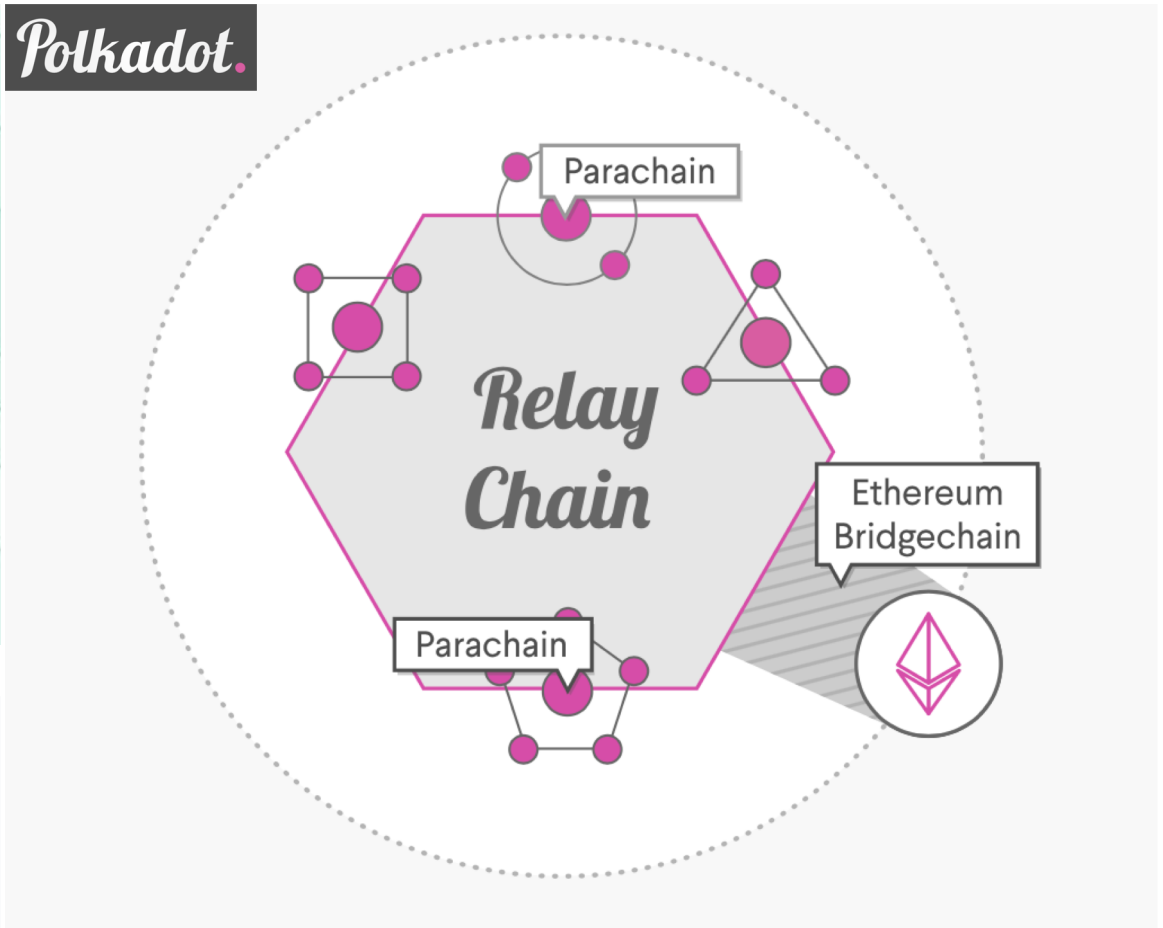
- 暖场（Warm Up）
- 区块链范式（Blockchain Paradigm）
- 以太坊数据设计概要（Overview of Ethereum Data Design）
- 交易执行（Transaction Execution）
- 合约创建（Contract Creation）
- 消息调用（Message Call）
- 执行模型（Execution Model）
- 区块定稿（Block Finalisation）
- 小结（Conclusion）

暖场

Parity Technologies

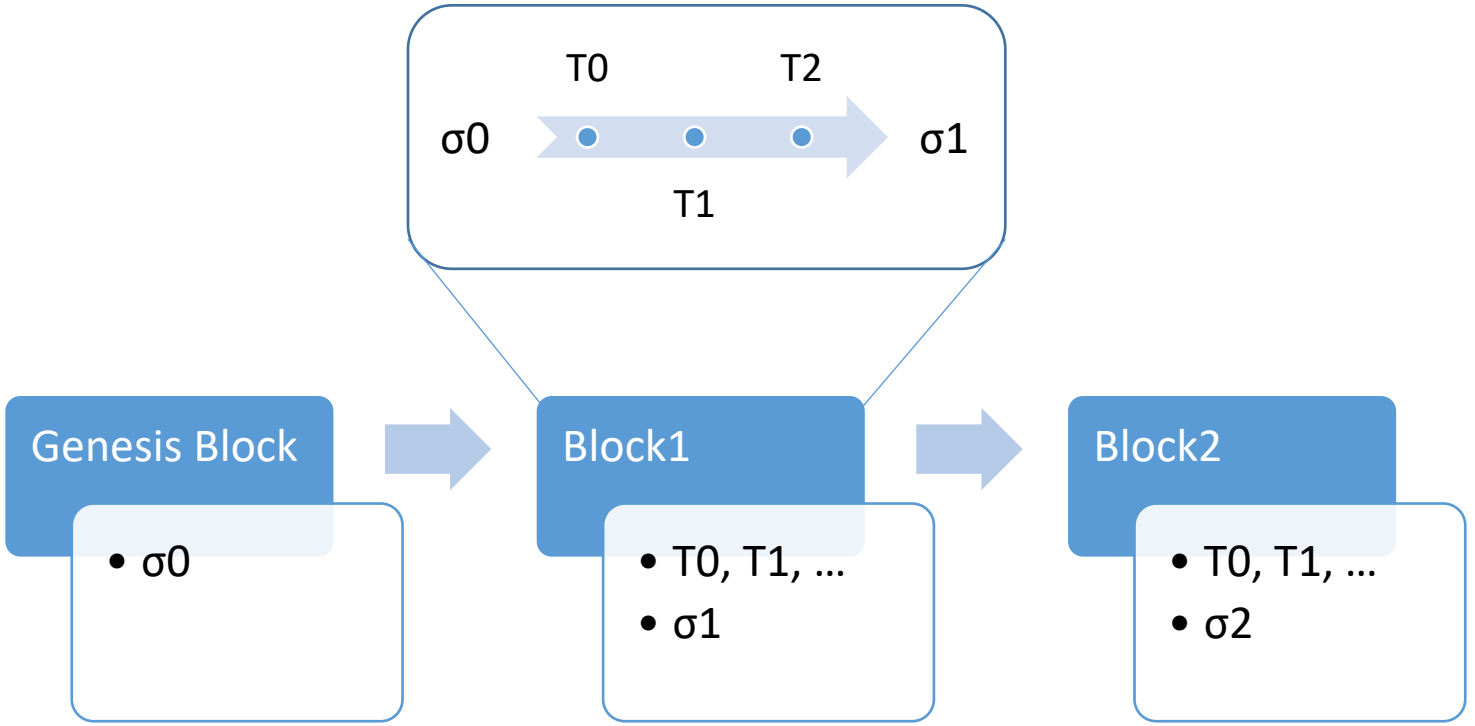
Solutions for a trust-free world

London http://paritytech.io admin@parity.io



区块链范式（Blockchain Paradigm）

$$
\begin{aligned}
(1) \quad & \boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T) \\
(2) \quad & \boldsymbol{\sigma}_{t+1} \equiv \Pi(\boldsymbol{\sigma}_t, B) \\
(3) \quad & B \equiv (..., (T_0, T_1, ...)) \\
(4) \quad & \Pi(\boldsymbol{\sigma}, B) \equiv \Omega(B, \Upsilon(\Upsilon(\boldsymbol{\sigma}, T_0), T_1)...)
\end{aligned}
$$



以太坊数据设计概要——基础算法

HiBlock 区块链社区

RLP（ Recursive Length Prefix ）

- 一种可以对任意结构的二进制数据（字节数组）进行序列化的编码算法。

HP（ Hex-Prefix Encoding ）

- 一种可以将任意数量的半字节数据（ nibble ）编码为字节数组的有效方法。

Trie（ Modified Merkle Patricia Tree ）

- 一种可以存储“键值对”映射数据集合的可变数据结构。

以太坊数据设计概要——Recursive Length Prefix

假定 x 为字节数组，RLP(x) 为 x 的 RLP 编码结果

且 s(x) 为以 RLP(x) 为元素的元组，即 s(x) = RLP(x0) . RLP(x1) . …

用 len(x) 表示 x 的字节数长度，则有 len(s(x)) = len(RLP(x0)) + len(RLP(x1)) + …

那么 RLP 编码规则可以定义如下：

条件	RLP 前缀字节	RLP 编码结果
len(x) == 1 且 x < 128	x 数值范围：[0x00, 0x7f]	RLP(x) = x
len(x) < 56	128 + len(x) 数值范围：[0x80, 0xb7]	RLP(x) = 128 + len(x) . x
len(x) >= 56	183 + len(len(x)) 数值范围：[0xb8, 0xbf]	RLP(x) = 183 + len(len(x)) . len(x) . x
len(s(x)) < 56	192 + len(s(x)) 数值范围：[0xc0, 0xf7]	RLP(s(x)) = 192 + len(s(x)) . s(x)
len(s(x)) >= 56	248 + len(len(s(x))) 数值范围：[0xf8, 0xff]	RLP(s(x)) = 248 + len(len(s(x))) . len(s(x)) . s(x)

以太坊数据设计概要——Hex-Prefix Encoding

HiBlock 区块链社区

$$\mathtt{HP}(\mathbf{x}, t) : \mathbf{x} \in \mathbb{Y} \equiv \begin{cases} (16f(t), 16\mathbf{x}[0] + \mathbf{x}[1], 16\mathbf{x}[2] + \mathbf{x}[3], ...) & \text{if } \|\mathbf{x}\| \text{ is even} \\ (16(f(t) + 1) + \mathbf{x}[0], 16\mathbf{x}[1] + \mathbf{x}[2], 16\mathbf{x}[3] + \mathbf{x}[4], ...) & \text{otherwise} \end{cases}$$

$$f(t) \equiv \begin{cases} 2 & \text{if } t \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

hex char	bits	node type partial	path length
0	0000	extension	even
1	0001	extension	odd
2	0010	terminating (leaf)	even
3	0011	terminating (leaf)	odd

- <0x012345, flag=0> -> 0x00012345
- <0x12345, flag=0> -> 0x112345
- <0x0f1cb8, flag=2> -> 0x200f1cb8
- <0xf1cb8, flag=2> -> 0x3f1cb8

以太坊数据设计概要——Trie 的示例

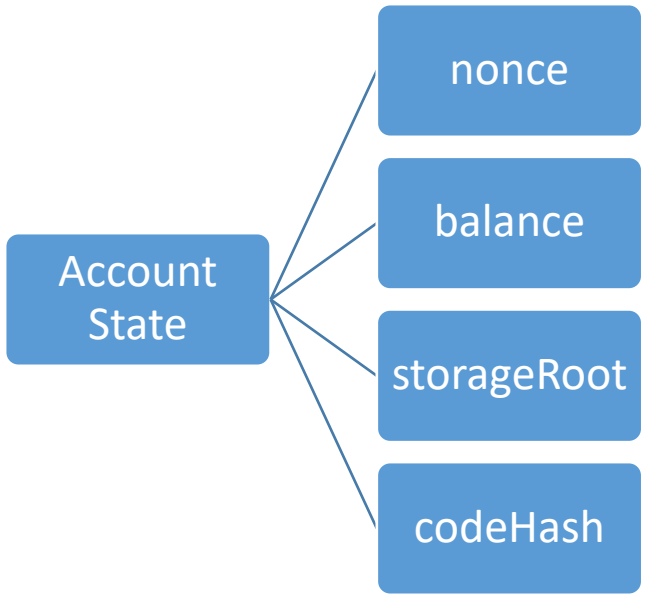
HiBlock 区块链社区

{ ('do', 'verb'), ('dog', 'puppy'), ('doge', 'coin'), ('horse', 'stallion') }

- <64 6f> : 'verb'
- <64 6f 67> : 'puppy'
- <64 6f 67 65> : 'coin'
- <68 6f 72 73 65> : 'stallion'

• rootHash:	[<16>, hashA]	Extension
• hashA:	[<>, <>, <>, <>, hashB, <>, <>, <>, hashC, <>, <>, <>, <>, <>, <>, <>, <>]	Branch
• hashB:	[<00 6f>, hashD]	Extension
• hashC:	[<20 6f 72 73 65>, 'stallion']	Leaf
• hashD:	[<>, <>, <>, <>, <>, <>, hashE, <>, <>, <>, <>, <>, <>, <>, <>, <>, 'verb']	Branch
• hashE:	[<17>, hashF]	Extension
• hashF:	[<>, <>, <>, <>, <>, <>, hashG, <>, <>, <>, <>, <>, <>, <>, <>, <>, 'puppy']	Branch
• hashG:	[<35>, 'coin']	Leaf

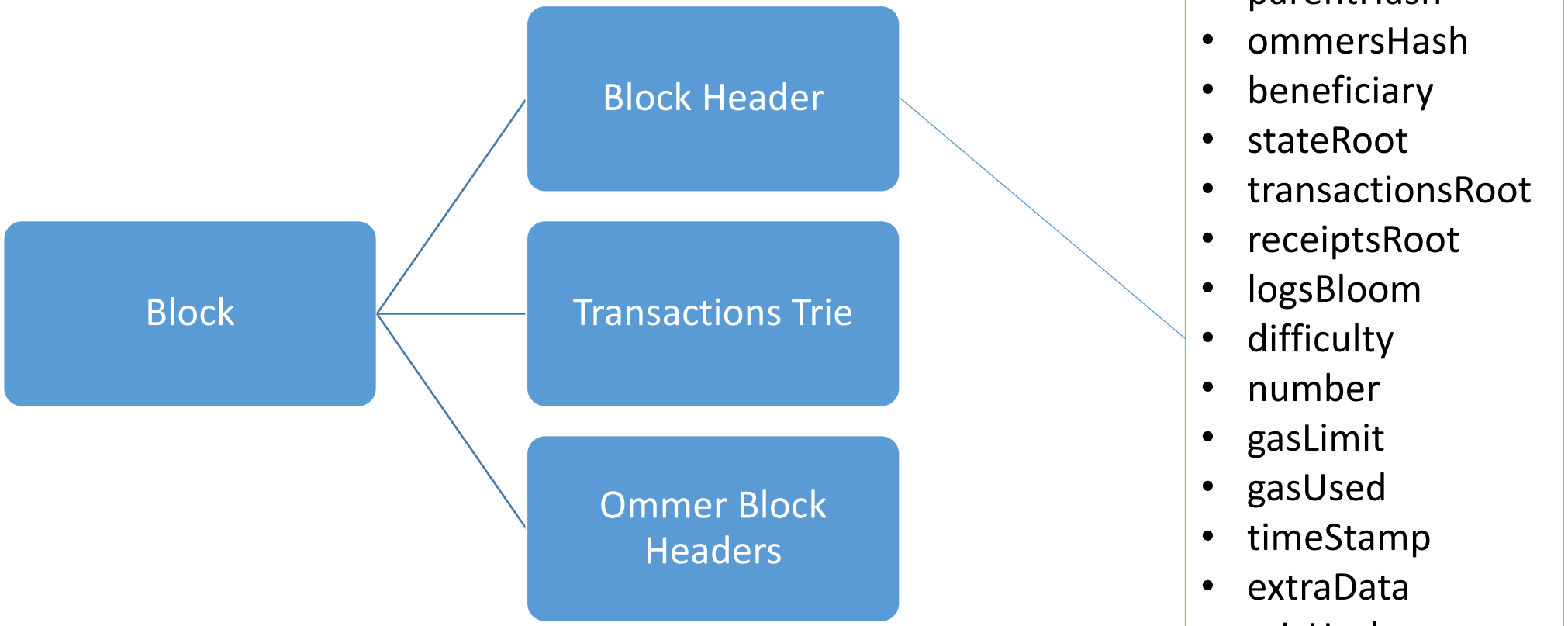
以太坊数据设计概要——State Trie



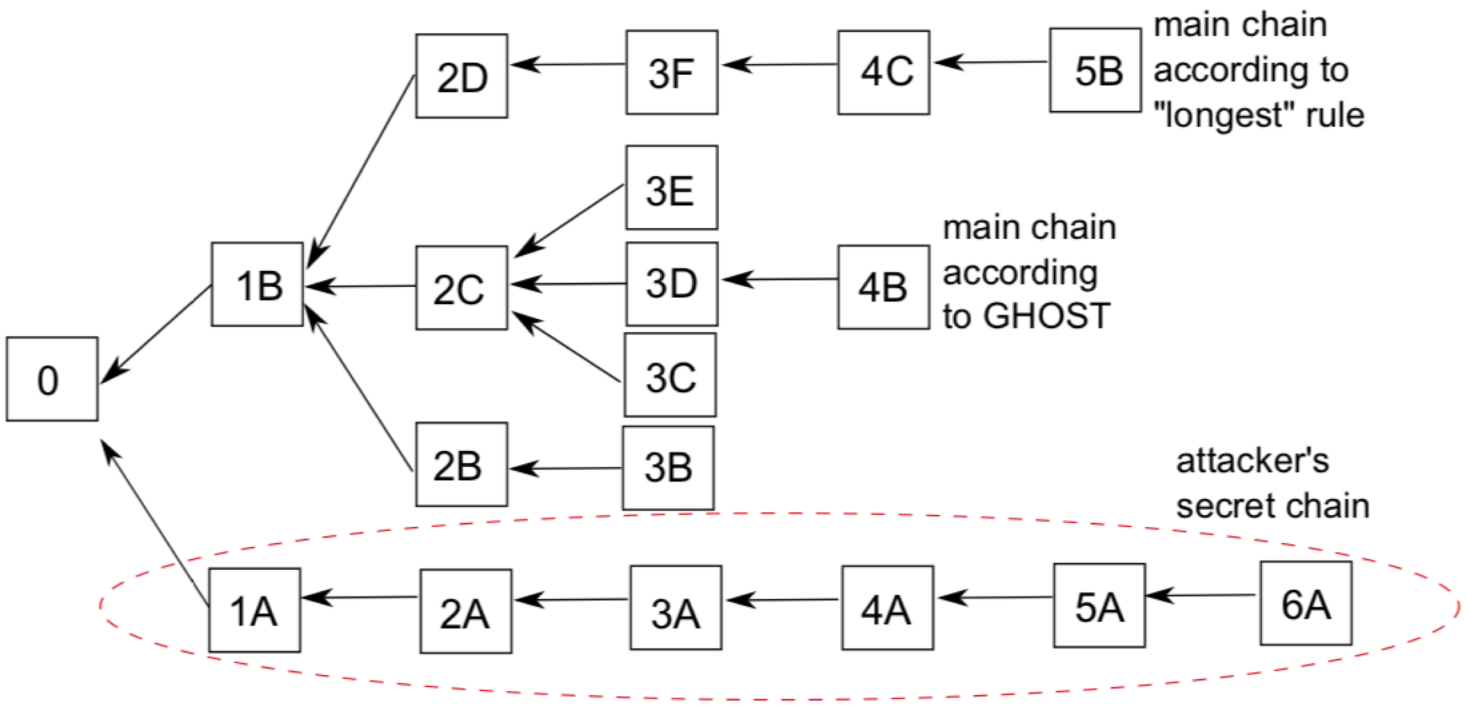
State Trie （ World State ）

- 全局唯一的状态树，存储的是 <sha3(address), rlp(accountState)> 的集合。
- 账户的 storageRoot 表示另一个全局唯一的合约存储状态树的根节点哈希，基于这个合约存储状态树（ Storage Trie ），可以通过地址、存储位置、区块号取得相应的合约存储数据（ web3.eth.getStorageAt ）。
- State Trie 和 Storage Trie 实际存储在各节点（客户端）中，数据由节点根据区块数据生成（更新），不会经网络传输。
- 以太坊网络中的各节点（客户端）需要维护一个 Trie Database，每个区块对应一个特定的 State Trie 版本（和 Storage Trie 版本）。基于 Trie Database，可以将整个树的数据恢复到特定的状态（某个区块定稿之后的状态）。

以太坊数据设计概要——区块



彩蛋——Greedy Heaviest-Observed Sub-Tree



以太坊数据设计概要——Transaction 和 Receipt

HiBlock 区块链社区

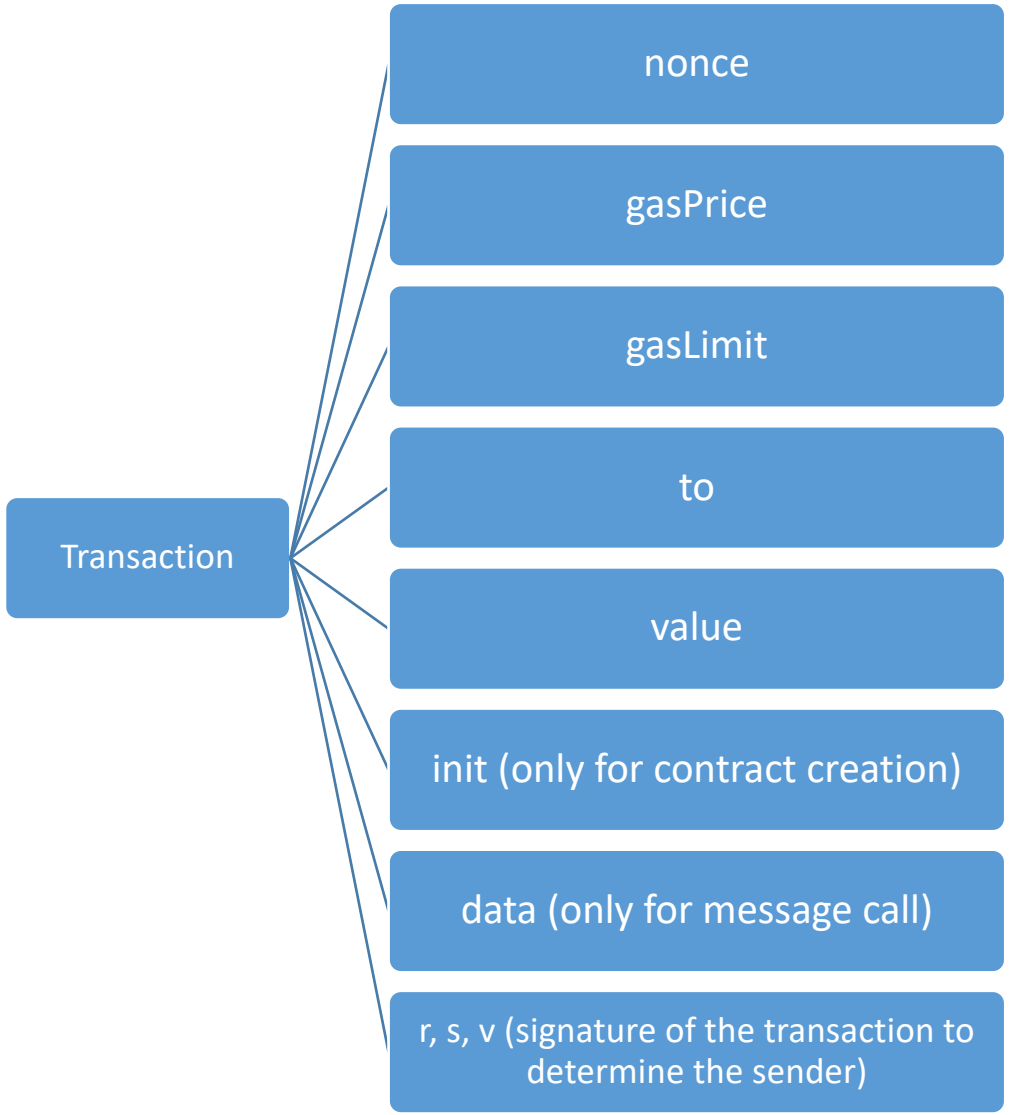
Transactions Trie

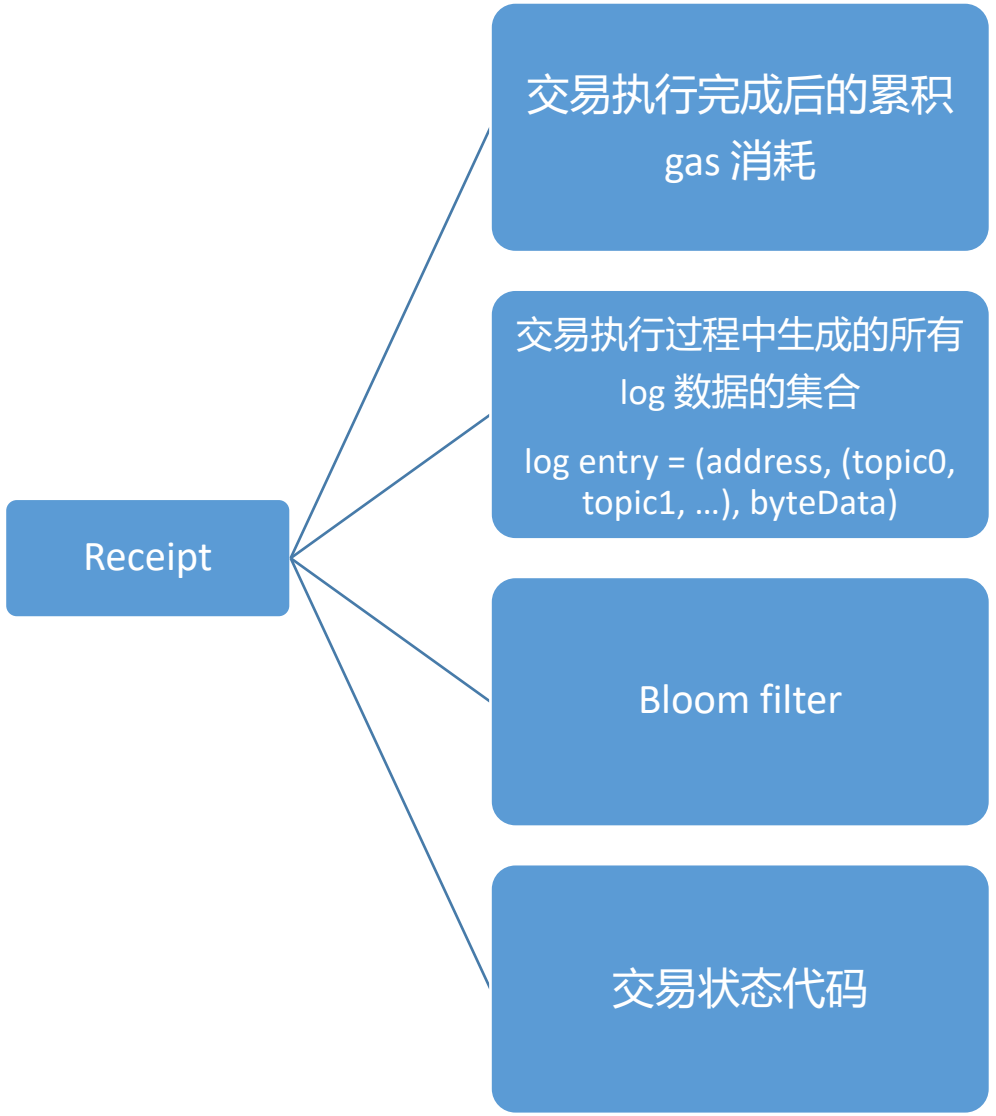
- 每个区块有独立的交易树，存储的是 <rlp(transactionIndex), transactionData> 的集合
- 数据存储在区块中，会由记账矿工将具体数据作为区块数据的一部分向网络广播

Receipts Trie

- 每个区块有独立的收据树，存储的是 <rlp(transactionIndex), receiptData> 的集合
- 数据存储在各个节点（客户端）中，是一次性的数据，不会更新，也不会向网络广播。

以太坊数据设计概要——Transaction 和 Receipt





交易执行（Transaction Execution）

前置条件：

- 交易数据是合法的 RLP 格式
- 交易签名有效
- 交易的 nonce 有效
- gas 上限不小于实际要使用的 gas 数量
- 发送者账户的 balance 不小于实际的 gas * gasPrice
- gas 上限与当前区块中已经使用的 gas 数量之和，不大于当前区块的 gasLimit

实际执行：

- 发送者账户的 nonce + 1
- 发送者账户的 balance 减去实际的 gas * gasPrice
- 执行合约代码
- 返还剩余 gas 给发送者
- 交易费支付给矿工
- 删除自毁账户列表和代码执行中接触过的空账户

交易子状态：

- 自毁账户列表
- 交易日志列表
- 接触过的账户列表
- 交易返还的 gas

合约创建（Contract Creation）

输入参数：

- 发送者
- 原始交易发送者
- 可用的 gas
- gasPrice
- endowment
- EVM 初始化代码（字节数组）
- 调用栈深度
- 修改状态的许可

几点说明：

- 合约创建是由 EVM 代码完成的，在这个代码执行过程中可以更改当前账户的 storage，也可以再创建合约或者再做消息调用。
- EVM 初始化代码，是“生成新合约的代码的代码”；新合约的代码，是保存在 EVM 的 ROM 中的，也就是保存在各个节点（客户端）中。
- EVM 初始化代码执行中出现任何异常，都将回退所有变更；也就是说，要么带着 endowment 成功创建合约，要么什么都不会发生。
- 初始化代码如果没有返回合约代码，或者用 EVM 的 STOP 指令停止执行的话，会产生一个僵尸账户，账户中收到的 endowment 会被永久锁定。

消息调用（Message Call）

HiBlock 区块链社区

输入参数：

- 发送者
- 原始交易发送者
- 接受者
- 执行代码的账户
- 可用的 gas
- gasPrice
- value
- 函数调用输入数据
- 调用栈深度
- 修改状态的许可

几点说明：

- 消息调用会有一个输出数据。
- 合约函数代码执行中出现任何异常，都将回退所有变更。
- 当通过 DELEGATECALL 调用合约函数时，可以保持调用上下文（即合约函数调用中的发送者和 value 保持与调用者相同），这使合约代码的重用成为可能。
- 消息调用中可以使用 8 个“预编译”合约（合约地址为 1 到 8）：椭圆曲线公钥恢复函数、SHA2 256 位哈希函数、RIPEMD 160 位哈希函数、标识函数、任意精度的模幂运算、椭圆曲线加法、椭圆曲线纯量乘法和椭圆曲线配对检查。

执行模型（Execution Model）

基本概念：

- EVM 是“准”图灵机
- EVM 的基础存取单位是“字”（Word）
- EVM 是基于栈（Stack）的架构
- 存储（Storage）
- 内存（Memory）
- 所有合约代码保存在 ROM 中
- EVM 指令的 gas 消耗是严格定义的
- 使用内存也是要消耗 gas 的，且是 JIT 结算

机器状态：

- 可用的 gas
- 程序计数器
- 内存的内容
- 内存中激活的“字”数
- 栈的内容

几点说明：

- 栈的最大深度是 1024（“字”）
- 无法用指令直接触发异常停止
- 执行的过程就是基于执行环境（上下文）递归（迭代）地执行合约代码，直到状态机达到异常停止或正常停止状态

区块定稿（Block Finalisation）

HiBlock 区块链社区

Ommer Validation

- 区块头为合法区块头，且父区块为当前区块的6代及以内祖先区块；最多包含 2 个 ommer header。

Transaction Validation

- 区块头的 gasUsed 必须与区块内所有交易的 gas 消耗之和相等。

Reward Application

- 当前区块的 beneficiary 和最多 2 个 ommer 区块的 beneficiary 都会得到区块奖励。

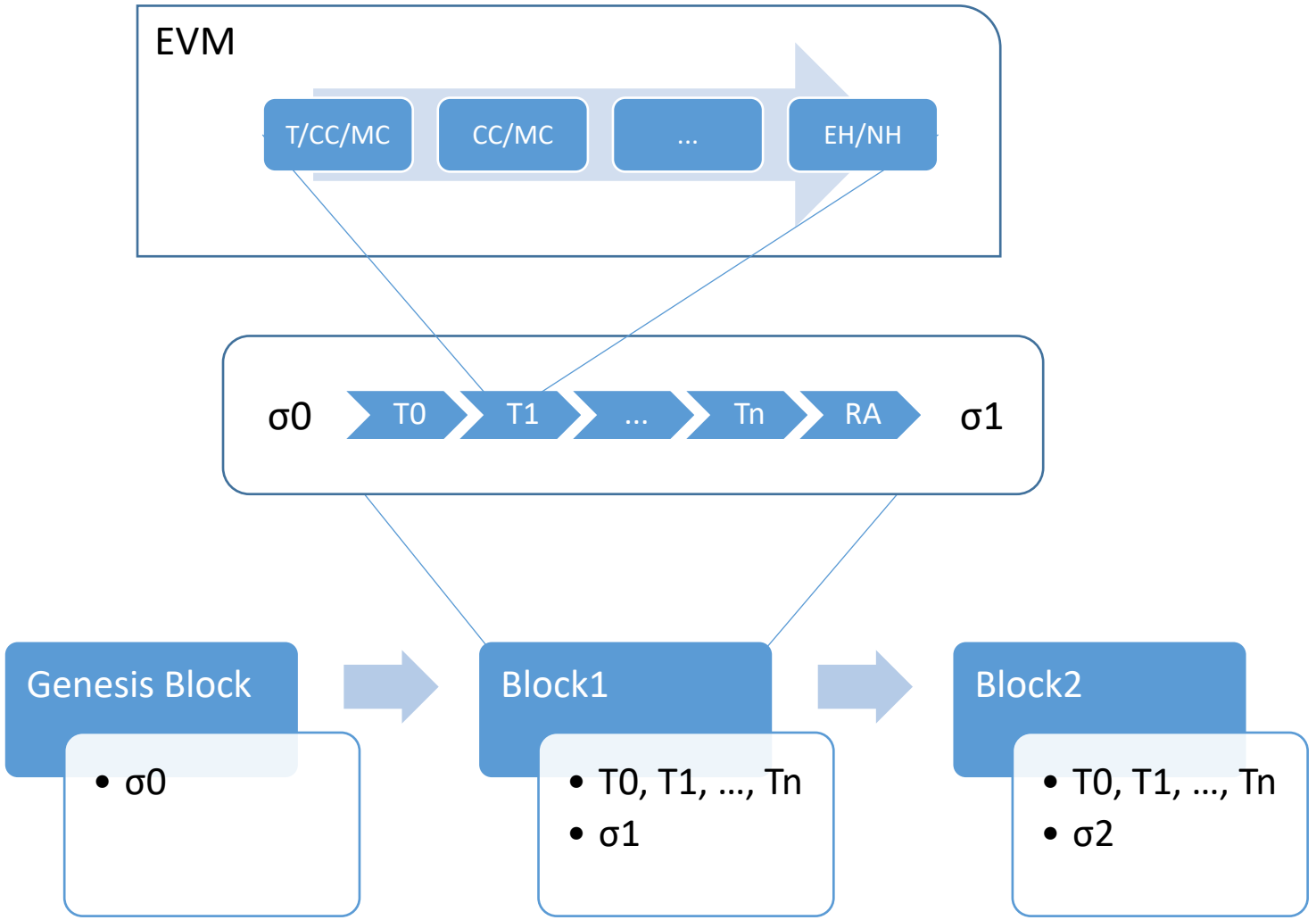
State Validation

- 基于当前区块内的所有交易数据、区块奖励计算当前区块的 stateRoot。

Mining Proof-of-Work

- 使用 Ethash 计算当前区块的 nonce 和 mixHash。

小结（Conclusion）



σ: World State
T: Transaction
RA: Reward Application
CC: Contract Creation
MC: Message Call
EH: Exceptional Halting
NH: Normal Halting

HiBlock
区块链社区

扫码关注HiBlock公众号
一起学习区块链技术

谢谢！