以太坊概述

——数据结构 与 共识机制

吴嘉婧 副教授

中山大学 计算机学院

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

以太坊简介

- 比特币和以太坊是两种最主要的加密货币
- 比特币是区块链1.0，以太坊是区块链2.0
- 以太坊基于比特币的一些运行问题进行了改进
  - 出块时间15s左右，减小交易延迟
  - mining puzzle对内存要求高（限制ASIC使用）
  - 使用PoS，代替PoW
- 第一个支持智能合约的区块链系统

回顾：传统合约

在了解智能合约前，先回顾传统合约：

- 传统合约如果没有定量标准，将无法正常执行
- 当双方有分歧，需要有可信任的公证人。
- 现实社会中合约需要通过政府、司法手段来维护。

自动化维度	条件满足，但交易未必会继续
主客观维度	公证人的主观意识会影响合约规则
执行时间维度	整个合约执行过程繁琐，浪费时间
违约惩罚维度	一方违约，未必会收到惩罚， 难以追责

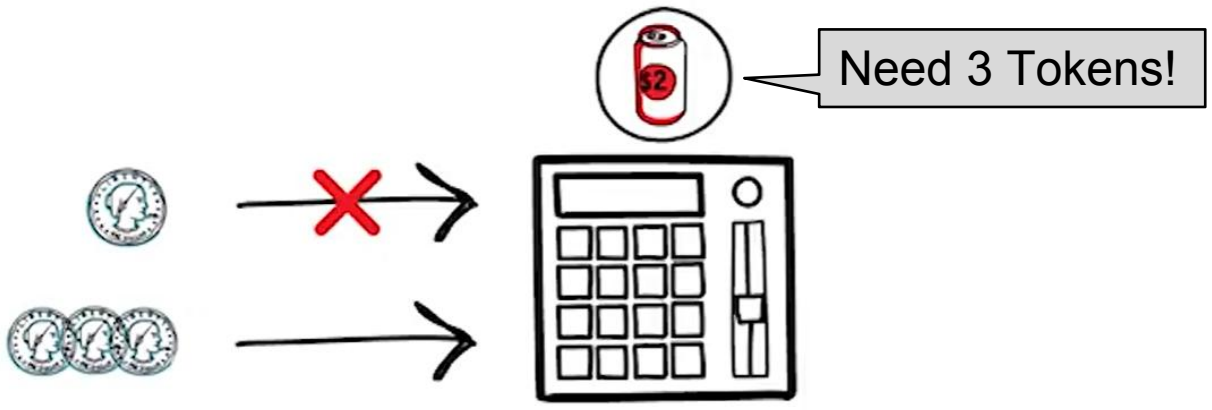
传统合约会受到各种维度的影响

智能合约

- 为了解决传统合约的弊端，尼克·萨博提出智能合约。
  - 一般认为，智能合约指能够自动执行合约条款的计算机程序，其概念由尼克·萨博在1994年提出，具有事件驱动、价值转移、自动执行等特性。

智能合约

- 将自动售货机视为智能合约
  - 事件驱动：合约以投币等动作作为输入，触发其动作执行；
  - 价值转移：外部以钱币为输入，合约输出饮料、食品等商品，完成了价值的交换或转移；
  - 自动执行：这一履约行为是完全自动的，不需要人在其中干预（投币动作除外）



智能合约

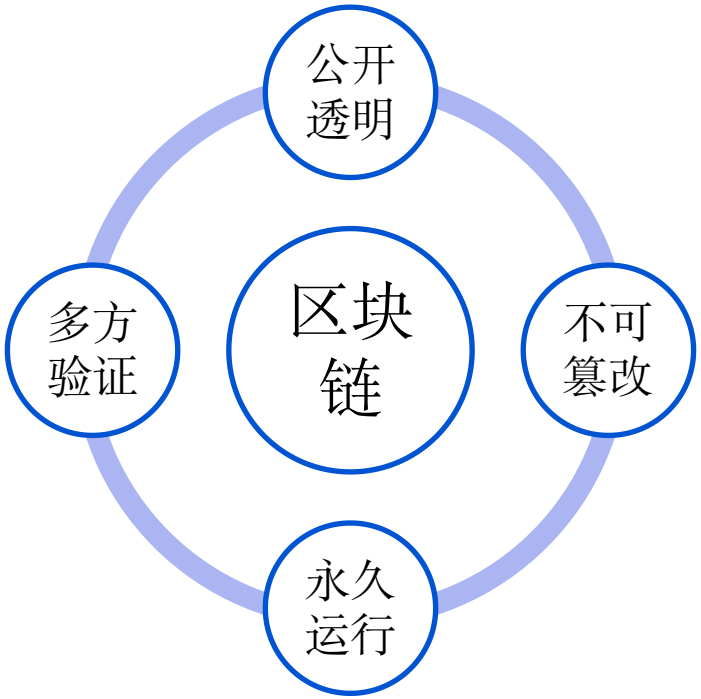
- 智能合约依赖于环境的可靠性
  - 售货机这一“智能合约”依赖的就是机器，其执行的可靠性除了合约本身，还依赖于执行环境的可靠性，一旦执行环境出错，则合约的执行也将出错

智能合约

- 中心化环境的弊端
  - 容易受到篡改：可以人为地修改自动售货机的程序，使其免费出售商品
  - 出错后难以追溯恢复：一旦售货机程序被篡改，篡改的源头是往往无法追溯的，因为恶意篡改人已经掌握了整个机器的控制权

智能合约

- 应用在区块链上的智能合约
  - 区块链是一种能使多方间达成状态一致的有效手段，那么将智能合约应用到区块链上，就能使得智能合约具备更高的可靠性。



智能合约

- 以太坊智能合约
  - 以太坊在多个节点组成的点对点网络中，维护共同的区块链数据，通过区块链上的交易来进行智能合约的创建、调用、结束等操作。
  - 由于多个节点所维护的区块链状态是一致的，因此，多个节点上所运行的智能合约的过程和结果也是一致的。

智能合约的出现解决了传统合约的信任问题，大幅降低了信任成本。

Code is law!

以太坊简介

- 第一个支持智能合约的区块链系统
- 使用Ether作为加密数字货币
- 是生态社区最活跃的区块链系统，出现了大量去中心化自治组织（DAOs）和去中心化应用（Dapps），促进了区块链技术在发币以外的应用
  - 超过30,000个github开源项目
  - 超过 3000 个基于以太坊的 Dapp

以太坊诞生和发展

2015 年 7 月 → Frontier（前沿）阶段
- 仅开发者测试使用

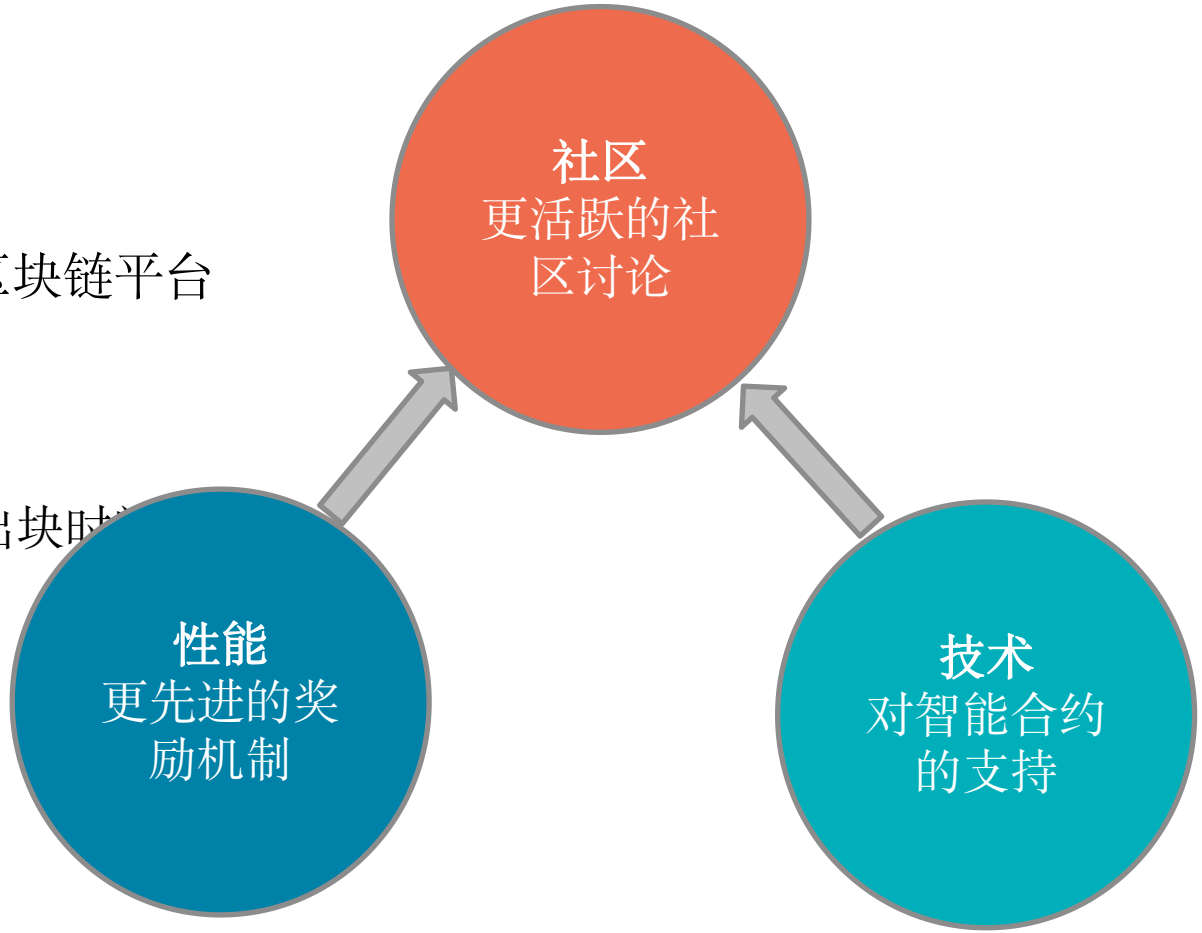
2016 年 3 月 → Homestead（家园）阶段
- 提供图形界面，提升安全性和平稳性

2017 年 10 月 → Metropolis（大都会）阶段
- 在智能合约执行性能、以太坊经济政策等方面进行了优化

至今未完成 → Serenity（宁静）阶段
- 预期将共识机制从工作量证明(Proof of Work，简称 PoW)转向 PoS，使持币者也能有一定的决定权

以太坊与比特币对比

- 技术：
  - 支持智能合约
  - 通用的Dapp的底层区块链平台
  - 账户模型
- 性能：
  - 增加叔块奖励，减少出块时
  - Ghost共识机制
- 社区：
  - 更加活跃的社区
  - 超过35k的开源项目



以太坊特色与应用

- 溯源存证
- 数字资产发行和流通
- 数据共享
- 多种应用Dapp

表 3.1　以太坊当前最流行的十个区块链应用

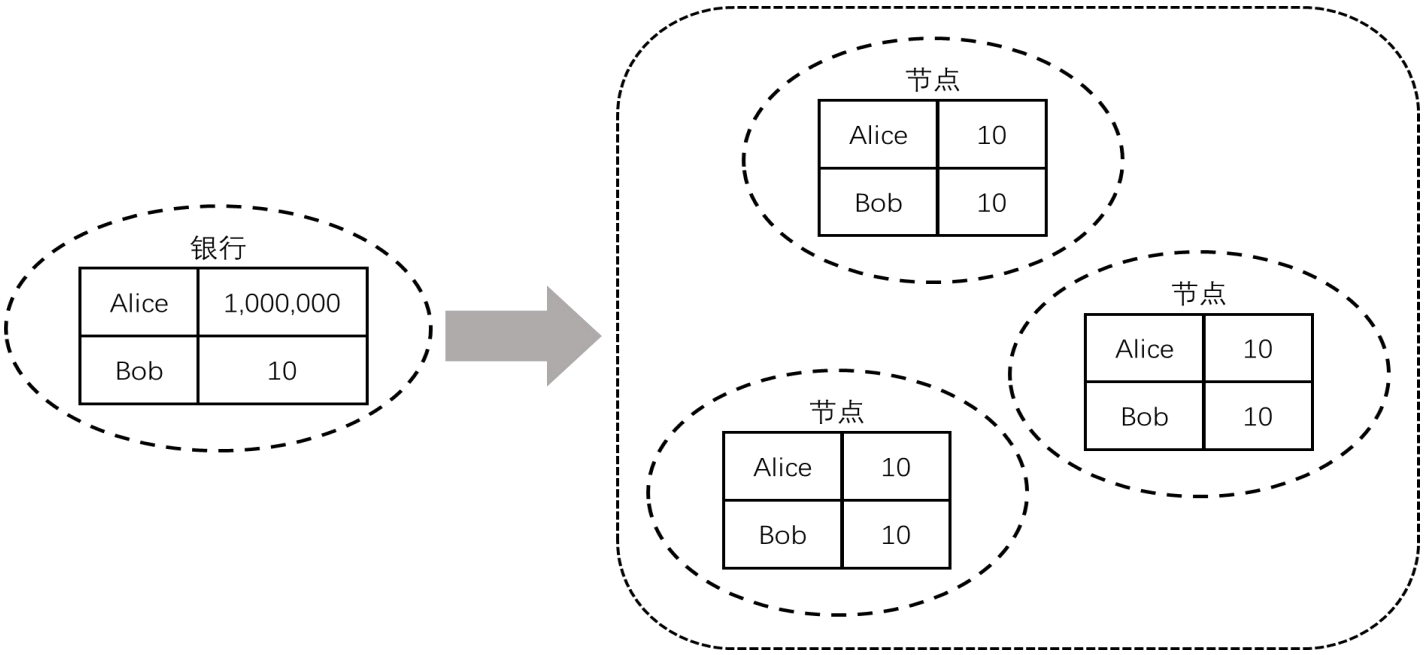
名称	类别	功能描述
MakerDAO	金融	去中心化借贷应用
Chainlink	安全	去中心化预言机
KyberNetwork	交换	代币交换协议
Status	钱包	DApp 浏览器发行的代币
My Crypto Heroes	游戏	角色扮演游戏
Uniswap	交换	代币交换协议
Axie Infinity	游戏	宠物养成游戏
Synthetix	金融	去中心化合成资产平台
Basic Attention Token	钱包	Brave 浏览器的激励代币
Knight Story	游戏	角色扮演游戏

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

以太坊账户模型

- 背景：
  - 地址与账户：类似银行系统
  - 区别：账本状态在所有区块链节点分布式存储



以太坊的账户模型

以太坊账户模型

- 地址的生成：
  - 计算椭圆曲线下的私钥与公钥

```
7C7FC7CCECC1FEE3B3E4AC63ADD864BD786696C20CA15683A269AA9AD8639443
 (私钥)
FBD0A98CEF180CDAA3FC2B64A4CA56FBC8E8AF5CC74E820DA34EB9DAB0D46FF2
E3858707929D0A9BEACF5533C22BF02D34E0A6EE4F1A19C1CEAC4FC4251810A6
 (公钥)
```

  - 对公钥使用KECCAK256哈希算法，计算得到一个64位的16进制哈希值（256个比特长）

```
DDC0D707E349E9B726925710238661F085A338F04B0C7C956A796B57018151F0
```

  - 截取这个哈希值的最后40位作为一个以太坊地址。

```
238661F085A338F04B0C7C956A796B57018151F0
```

以太坊账户模型

- 账户结构：
  - 用户账户结构保存了用户地址对应账户的数据信息
  - **余额**：记录了当前地址持有的以太币的数额，单位是Wei
  - **Nonce**：记录了这个地址创建以来累计发起的交易次数

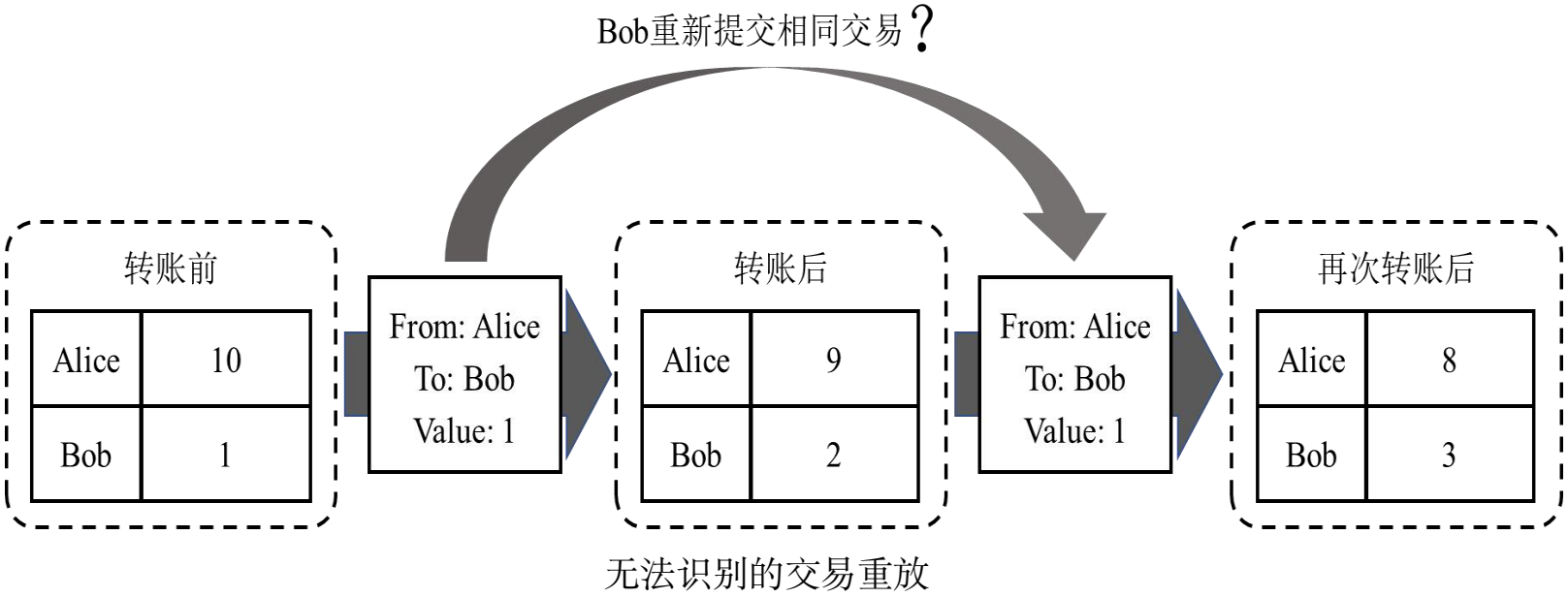
以太坊账户结构

Nonce的作用

- 交易重放问题：
  - 在以太坊的模型中，交易的合法性检验在于转账发起者的账户余额
  - 如果没有其他手段来使得发起过的交易失效，那么这个交易将可以被无限次的重新发起而不需要发起者的同意，因为发起者的签名对于交易始终是有效的。

Nonce的作用

- 交易重放问题：
  - 如何判断这笔交易是Alice继续向Bob转账，还是在没有Alice同意的情况下进行的一次恶意的攻击行为？

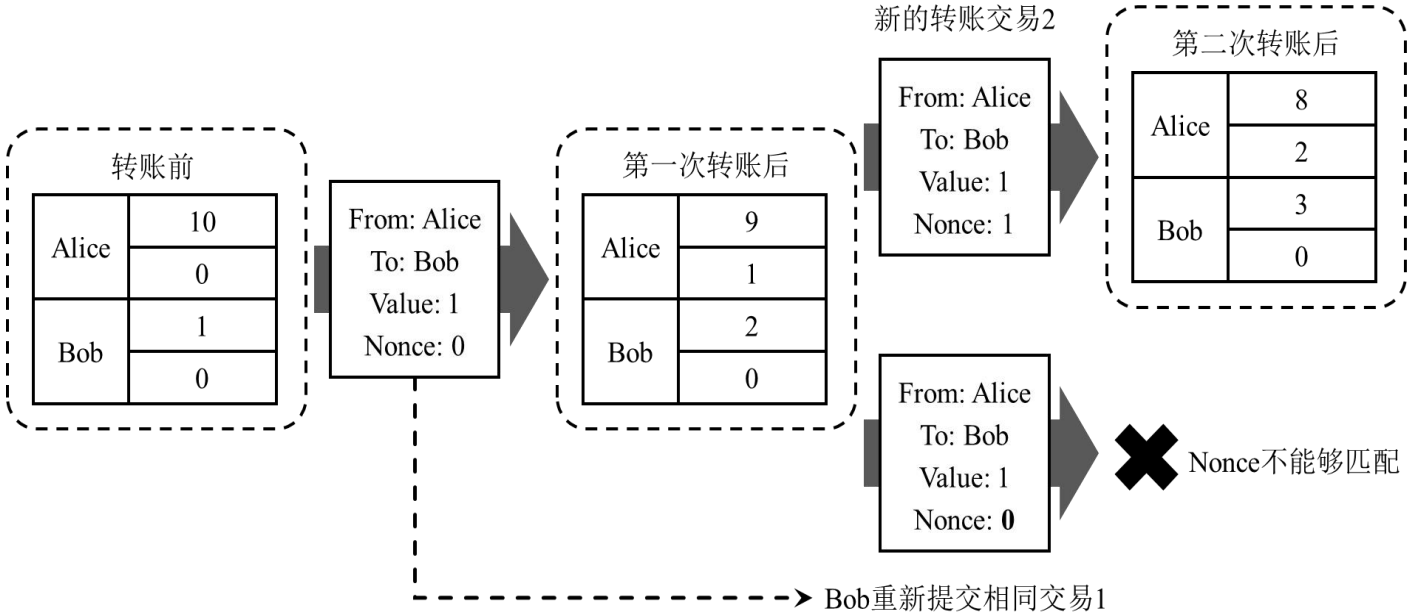


Nonce的作用

- Nonce计数器：计算交易次数和序列：
  - 只有账户的Nonce和交易的Nonce能够对应的情况下，交易才是合法的
  - 当一个交易执行完毕之后，账户的Nonce值增加，原本执行完毕的交易中的Nonce值就无法与现在账户的Nonce值匹配
  - 修改对应的Nonce值，意味着原有交易的签名失效，需要发起者的重新签名

Nonce的作用

- 实例：Nonce计数器
  - Alice向Bob转账之前，Alice的Nonce值为0
  - 当Alice向Bob转账1 ETH后，Alice的Nonce值变成1
  - Nonce值为1并且具有合法签名的交易2将会是一个合法的交易
  - 对于重新提交的交易1而言，此时的交易记录的Nonce值依旧为0



Nonce值防止重复交易

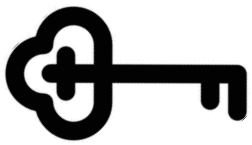
Nonce的作用

- Nonce的其他用途
  - 控制账户发起的交易的顺序，从而实现一些相对复杂的功能。
  - 通过重复提交一个相同Nonce值的交易来使得一个已经提交但是尚未被确认的交易变得不合法，从而实现一定程度的撤销功能。

以太坊账户

- 定义以太坊的两种账户：

外部账户
(Externally Owned Accounts)



有账户余额
无代码
能触发交易
(转账或执行智能合约)
由私钥控制

Nonce
Balance
CodeHash
StorageRoot

合约账户
(Contract Accounts)
<code>
<code>
<code>

有账户余额
有代码
能被触发执行智能合约代码
在智能合约创建后自动执行

思考1：以太坊账户

- 比特币的UTXOs模型更有利于隐私保护，那为什么以太坊还要使用账户模型？
  - 以太坊之所以要使用账户模型，是为了支持智能合约，要求参与者要有稳定的身份。比如智能合约可以实现一些金融衍生品（financial derivative），利于参与者进行投资。

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构
- 智能合约

以太坊状态

- 定义以太坊的账户状态：
  - Nonce：累计发起的交易次数
  - Balance：账户余额
  - CodeHash：智能合约代码的哈希值
  - StorageRoot：合约存储树的根节点哈希值（维护智能合约状态）
- 合约存储树：
  - 合约账户下的存储也是一个映射表，它记录了从存储地址到存储值的一个映射关系
  - 在合约账户的数据结构中存储了这个映射表的哈希值，这个哈希值被称作存储根（Storage Root），它同样是由一棵MPT（Merkle Patricia Trie）来维护和计算

思考2：以太坊状态如何维护

- 系统的全节点需要维护账户状态，如果只用哈希表（Hash Table）来实现账户地址到状态的映射（addr -> state），有什么弊端？
  - 提示：如果用哈希表，如何提供证明某个账户有多少钱？如何实现对状态一致性的共识？

答：如果要证明某个账户有多少钱，需要将哈希表的内容组织成一个Merkle Tree, 计算出根哈希值，将根哈希值存储在Block Header中（参考比特币）。只要根哈希值正确，就能保证Merkle Tree不被篡改。

但是，当一个新的区块发布，执行了新的交易必然导致哈希表的内容发生变化，则每次发布新的区块都需要将哈希表的内容重新组织一次Merkle Tree, 计算代价太大（以太坊账户数量多达上千万个）。而实际上真正改变的账户状态只是一小部分。

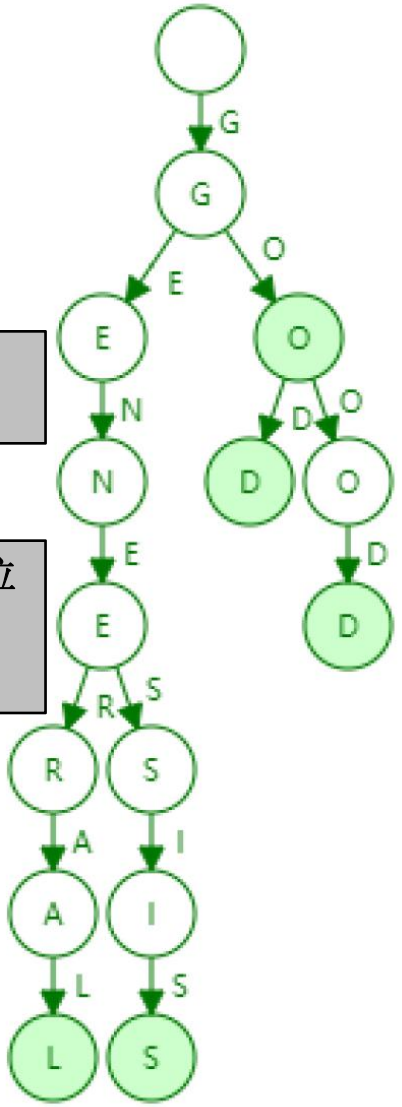
另外，利用哈希表构建的Merkle Tree不利于更新（新增一个账户要重构Merkle Tree）。

回顾：Trie 字典树

- Trie 来源于 Retrieval 单词。
- Trie：前缀树/字典树，是一种有序树
- 一个节点的全部子孙有相同前缀
- 分叉数目取决于Key的元素取值范围。
- Trie的查找效率取决于Key长度
- 单词不同就不会出现碰撞。
- 更新效率高，只需要局部更新。
- 缺点：存储浪费。

以太坊地址，用16进制为0~f

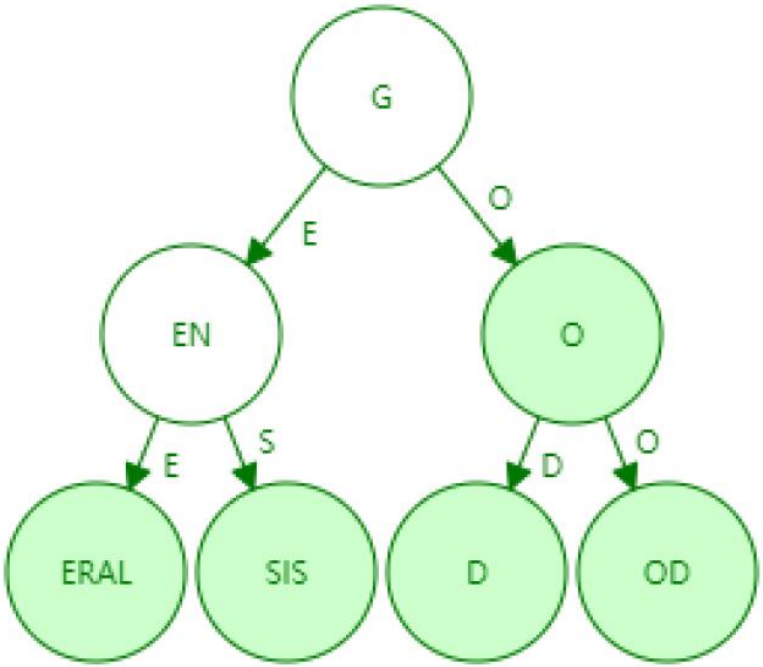
以太坊地址160位，对应40个16进制数。



General
Genesis
Go
God
Good

回顾：压缩前缀树

- Patricia Trie：对路径进行压缩，树的高度大大缩短，访问内存的次数变小，效率提高。
  - 键值路径比较稀疏的时候，压缩效果比较好（2^{160}远大于以太坊地址总数）



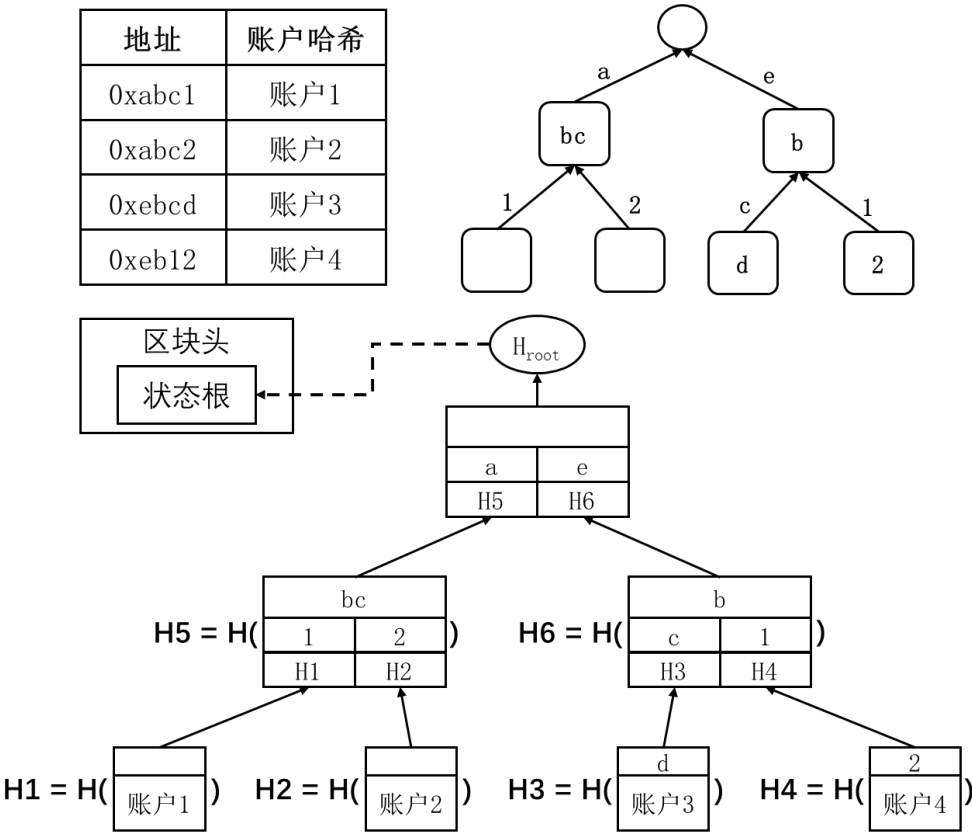
General
Genesis
Go
God
Good

Merkle Patricia Trie

- 背景
  - 如果把单词和箭头都换成哈希值，然后计算每个中间节点的哈希值，便得到了MPT
- MPT的构造
  - 按照所有数据的地址（或者键值）来构建一棵压缩前缀树
    - 由于地址是以16进制为编码的，我们使用0123456789abcdef作为每一个编码的单元
  - 按照构建得到的压缩前缀树，从叶子节点开始，逐步计算每一层的哈希值，并将其汇合到父节点中，与Merkle Tree的计算过程类似

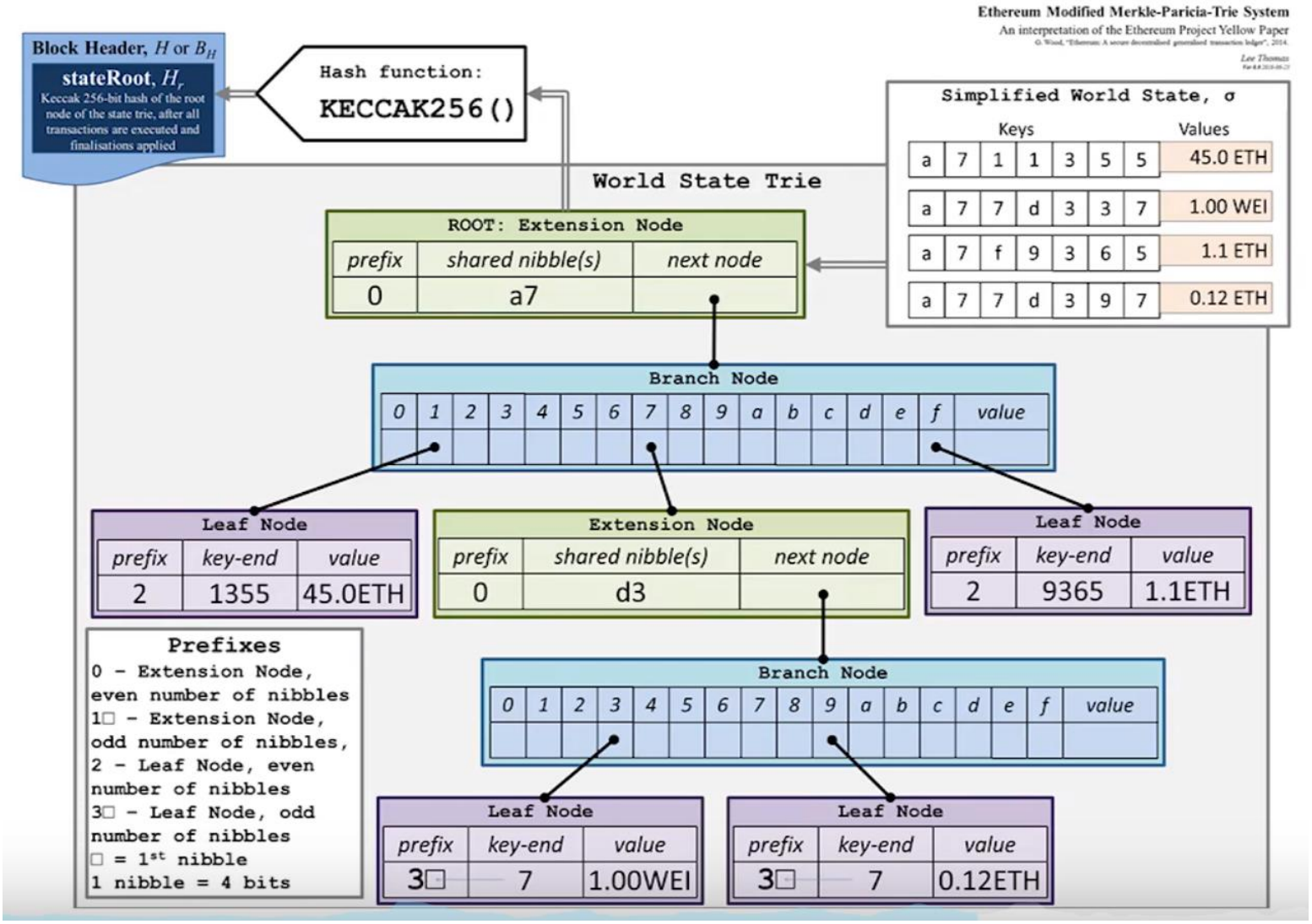
Merkle Patricia Trie

- 实例：根哈希值的计算
  - 计算叶子节点的哈希值
    - 空前缀加上数据1的哈希值，得到哈希值H1 = H(“” +账户1哈希)。H2、H3、H4的计算类似。
  - 计算父节点哈希值
    - 将前缀和子分支的哈希值一同进行哈希，得到哈希值H5 = H(bc + H1 + H2 + “” + … + “”)，注意其他子分支2~f不存在，值都是空。
  - 逐层计算得到根哈希值H_{root}
    - 根哈希值H_{root}=H(“” + “” + … + “” + H5 + “” + … + “” + H6 + “”)，注意到这个节点的公共前缀为空 。



MPT的构建

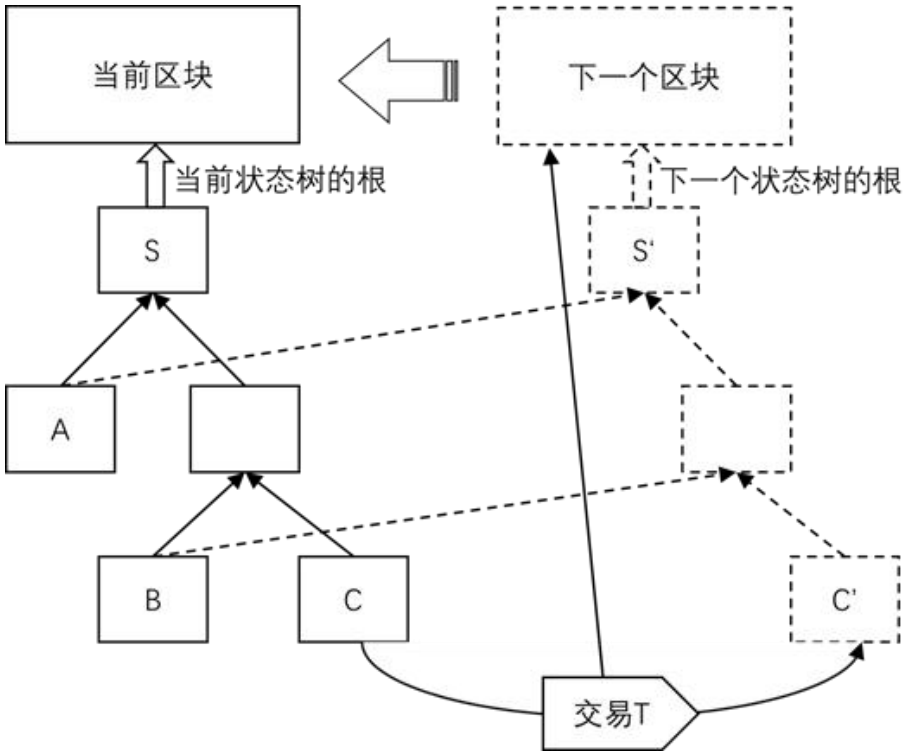
状态树



- 以太坊用来记录各个账号的状态的树，树的键是账户地址，值是账户的详细信息。
- 以太坊采用Modified Merkle Patricia Trie，引入三种节点。这里假设地址长度是7位。

状态树

- 实例：以太坊中状态树的变换
  - 当前的区块的世界观中有状态为A、B和C三个账户
  - 经过交易T之后，账户C的状态变成了C’，计算下一个状态树的根S’
  - 以太坊将交易T和状态树根S’记录在下一个区块之中：



以太坊中状态树的变换

局部更新：只是改变对应分支的状态。

状态树

- 当新区快形成时，区块链节点是生成一个新的状态树，而不是修改原来的状态树。其中大部分节点的内容是共享的。目的是有利于解决分叉时，对状态的回滚。
  - 与比特币不同，比特币简单的转账可以通过简单的反向推算。
  - 而以太坊有智能合约，实现的功能很复杂，很难推算出之前的状态。所以为了回滚，必须保存之前的状态。

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

收据 (Transaction Receipts)

- 定义
  - 收据是对应交易的数据结构，代表了交易执行的一些中间状态的写入 和 交易的执行结果 等信息

- “收据”的内容
  - 以太坊的智能合约向虚拟机输出的一些执行日志
  - 智能合约运行的 Gas 信息。
  - 单个交易执行完毕后以太坊的状态根
  - 一个交易创建智能合约的时候，如果执行成功会把新建合约的地址写到一个收据之中

收据

Receipt数据结构

```
45  // Receipt represents the results of a transaction.
46  type Receipt struct {
47          // Consensus fields
48          PostState         []byte `json:"root"`
49          Status            uint64 `json:"status"`
50          CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`
51          Bloom             Bloom  `json:"logsBloom"         gencodec:"required"`
52          Logs              []*Log `json:"logs"              gencodec:"required"`
53
54          // Implementation fields (don't reorder!)
55          TxHash          common.Hash    `json:"transactionHash" gencodec:"required"`
56          ContractAddress common.Address `json:"contractAddress"`
57          GasUsed         uint64         `json:"gasUsed" gencodec:"required"`
58  }
```

交易树和收据树

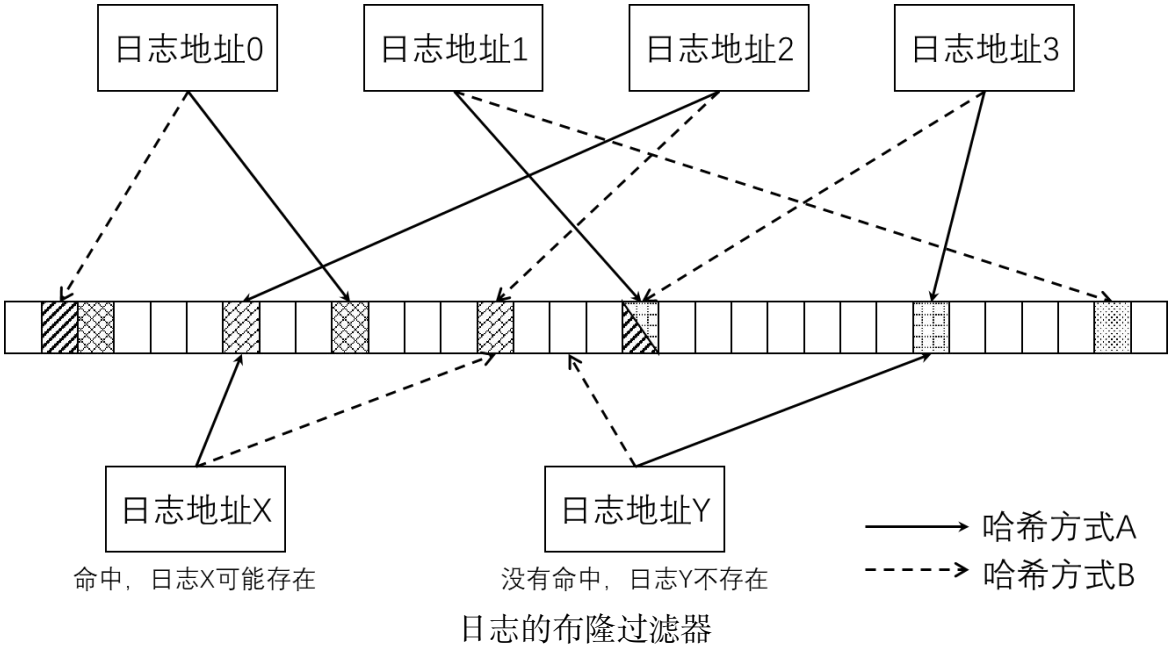
- 交易树和收据树
  - 与比特币中的Merkle Tree类似的，对于区块中的所有交易和交易对应的收据，都可以使用MPT进行组织和证明
  - MPT的构建不再是通过账户地址来进行，而是通过交易或者收据在区块中的序号来构建MPT
  - 收据树和交易的信息一一对应。主要是考虑到智能合约的执行比较复杂，收据树可以有利于快速查询，证明交易结果。
- 为什么使用MPT，而不是普通的Merkle Tree?
  - 代码复用：以太坊都用MPT， 代码统一，利于管理
  - 查找效率高。

Bloom Filter

- Bloom Filter （布隆过滤器）的用途
  - 以太坊中通过布隆过滤器对收据的日志进行索引。比如：
    - Client 找到过去十天和某个智能合约有关的交易
    - Client 找到过去十天当中符合某种类型的所有事件（e.g., 众筹事件 or 发行新币的事件）
- 优点
  - 布隆过滤器可以用于检索一个值是否在一个集合中。在容忍一定的误识别率的条件下，它有着远超过一般算法的空间效率和时间效率。
- 原理
  - 通过多个哈希函数将键值映射到位图之中，并在位图中合并集合中所有键值的映射结果。
  - 对于一个键值，如果经过同样的哈希函数映射之后，出现了在位图中没有出现的标记位，那么这个键值必定不存在于集合之中。

Bloom Filter

- 实例：在进行哈希方式B之后，若日志地址Y的哈希出现了新的标记位，那么日志地址Y必然不存在于原有四个日志地址之中
- 性质：可以保证某个元素一定不在集合里，但不能保证元素一定在集合中。即会误报，不会漏报。
- 好处：可以快速过滤，迅速定位， 再从对应区块中相求全节点发布进一步的信息。



日志的布隆过滤器

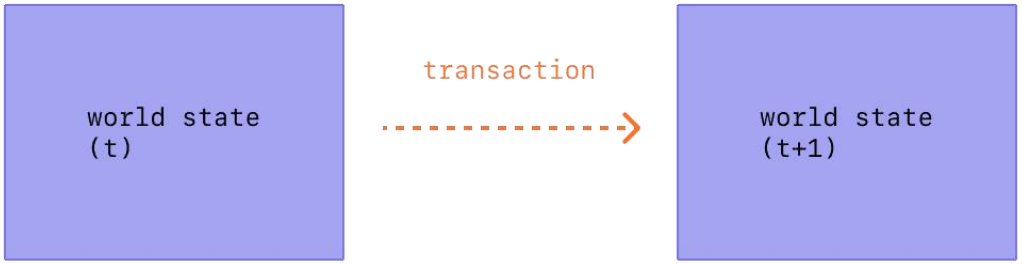
Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

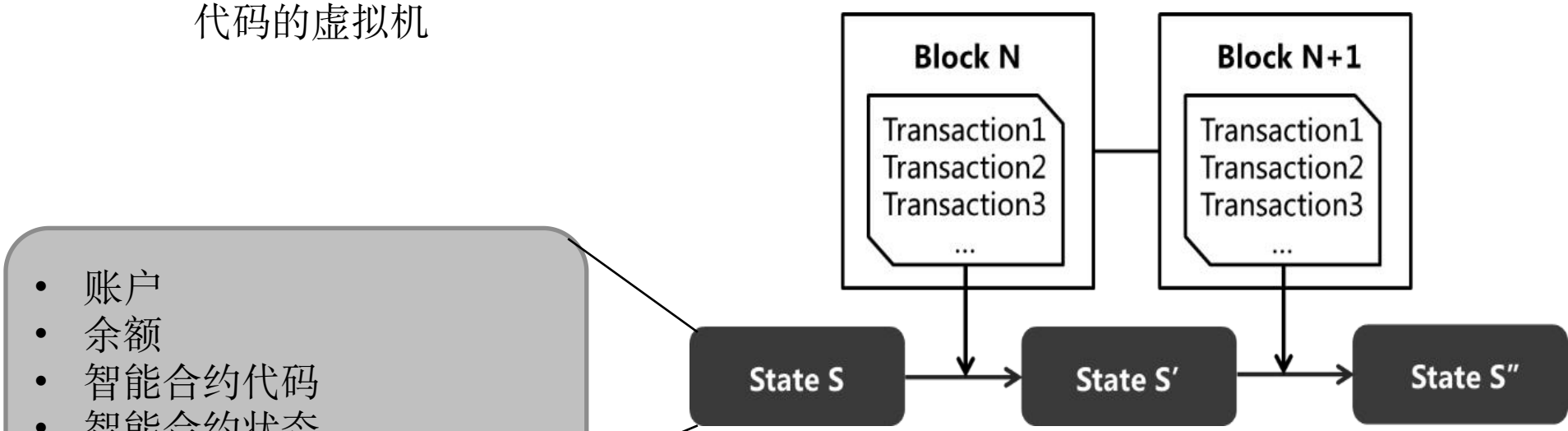
以太坊基本架构与原理——状态模型

- 背景
  - 在账户模型中，用户的余额通过地址上的账户数据来表示，具体为账户数据结构中的一个余额的数值
  - 在转账交易的过程中，通过转账预先定义好的语义，在发起者的账户中减去交易中定义好的转账金额，在接受者的账户中增加相应的金额
  - 可以把账户的余额泛化成一种账户的状态，而把转账交易当作是改变状态的一个方法
- 状态转移：由交易来驱动的状态机
  - 在区块N执行前状态为S，经过区块N的交易进行状态转换后，转换为状态S'，再经过区块N+1的转换后，转换为状态S''

状态转移



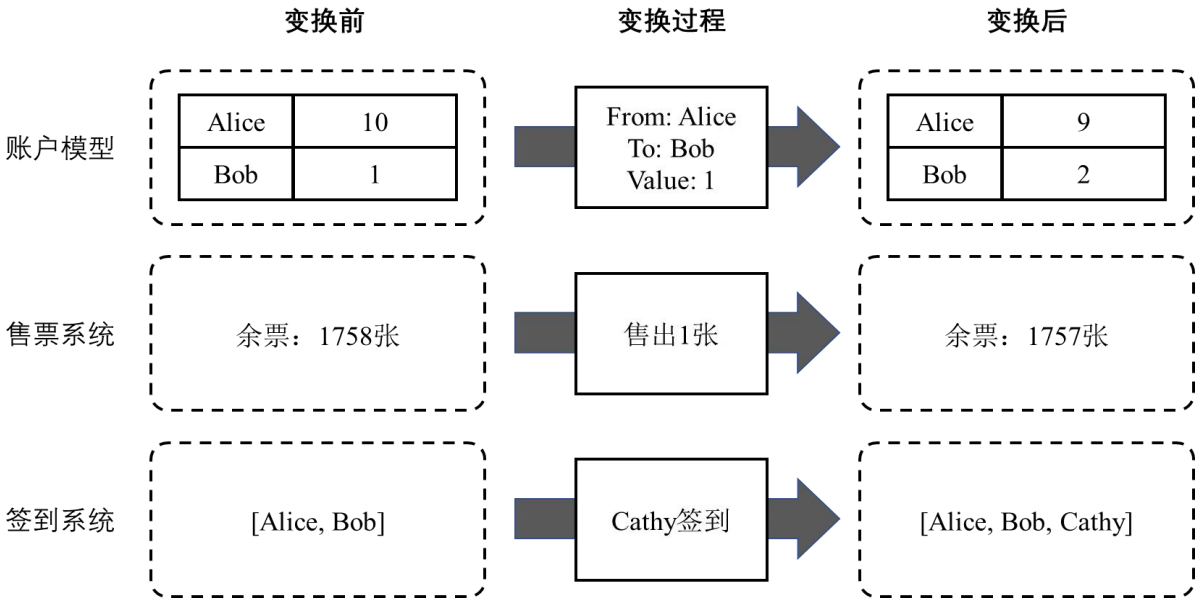
- 以太坊状态转移：
  - 智能合约：作用于该状态机转换的代码
  - 以太坊虚拟机（Ethereum Virtual Machine，简称EVM）：执行状态转换代码的虚拟机



以太坊中的状态转换模型示意图

状态转移

- 只要约定好的变换规则是固定的，就可以不断地通过变换规则不停地作用到原有状态上产生新的状态
- 对于有着相同的初始状态的参与者们来说，经过完全相同的变换过程，最后必然得到完全相同的当前状态。

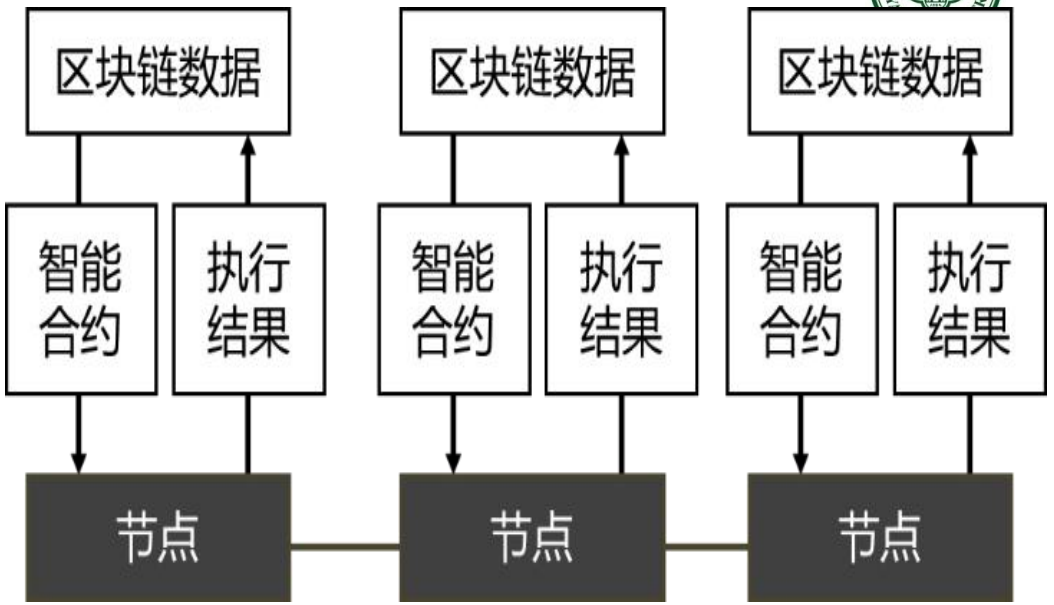


三种系统下的状态变换例子

以太坊基本架构及原理

- 以太坊状态转移模型的共识问题
  - 相比比特币的UTXO模型，状态转换模型虽然使得智能合约的各种变量存储、传参等变得更加灵活，但也带来了多方共识上的困难，如发生分叉时的处理
  - 在以太坊的状态转换模型中，如果发生分叉，需要回到分叉时的状态，重新验证另一条分支上的区块
  - 以太坊状态的存储采用了状态树结构，其根哈希记录在区块头中，记为stateRoot，从而使得状态能够在全网得到共识确认，并在分叉时能够快速回滚。

以太坊简易架构图

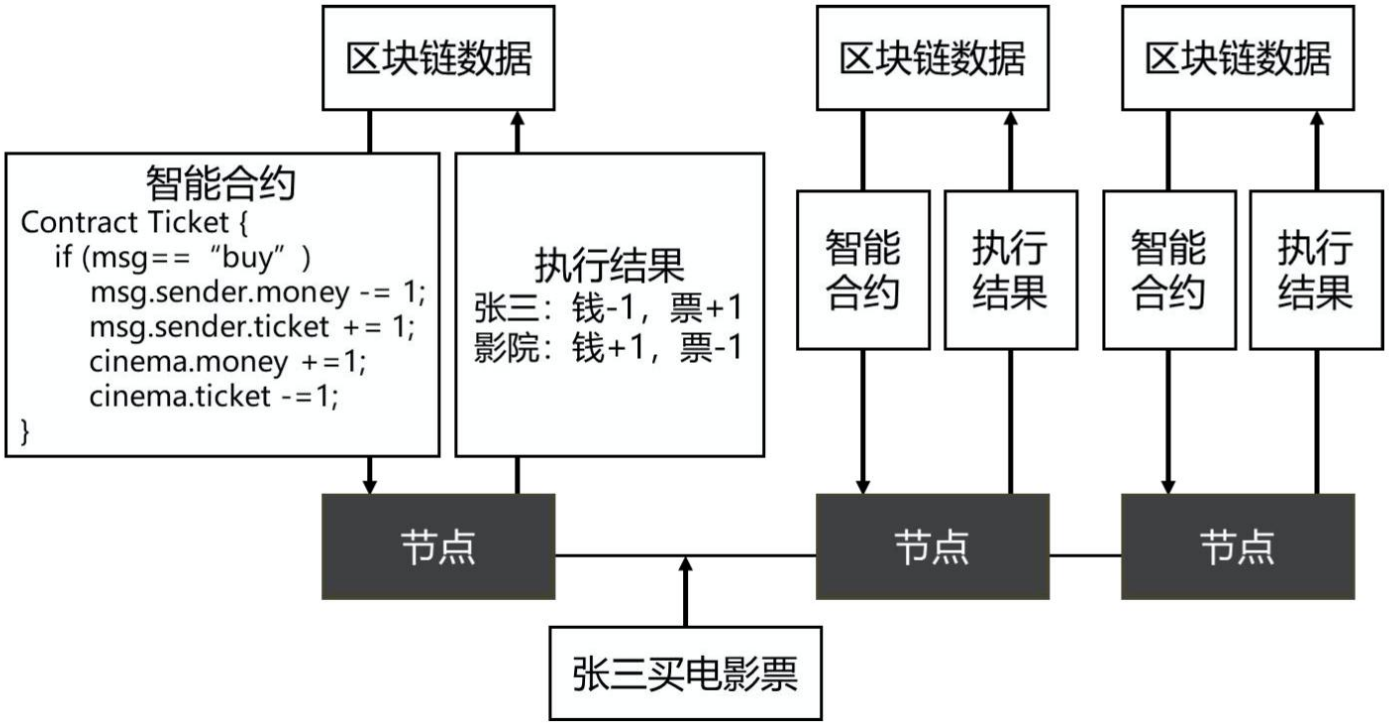


- 定义
  - 区块链数据
    - 以太坊区块链，
    - 状态数据，收据数据等
  - 智能合约
    - 存储在以太坊上的一段代码
  - 节点
    - 保存有区块链数据的节点
    - 每个节点独立的维护区块链数据
    - 节点间采用共识机制达成共识，维护全网的状态一致

以太坊基本原理及例子

- 交易执行概述

1. 每个节点独立维护数据
2. 节点独立地在EVM中执行合约
3. 将执行结果写回区块链数据
4. 节点之间执行共识机制，达成共识



以太坊基本原理及例子

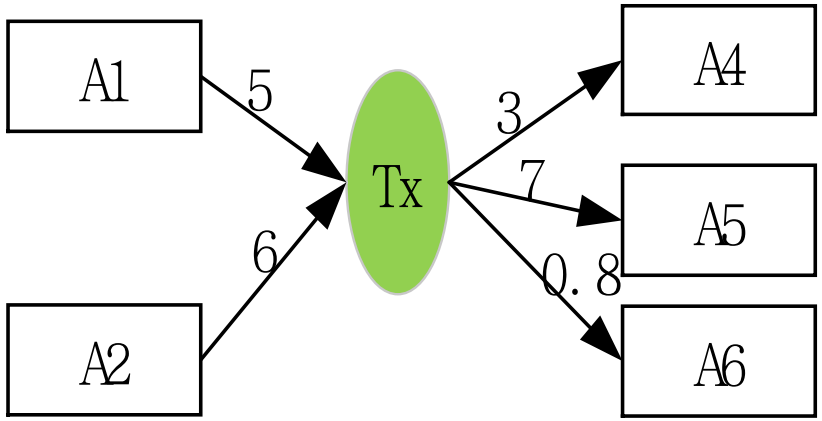
- 实例：交易执行具体过程
  1. 每个节点独立维护数据，这些节点可以是张三、李四、赵四、影院主管等人
  2. “张三买电影票”的交易在所有节点中被独立验证
  3. 以太坊虚拟机的执行结果将以某种方式写回到区块链数据中，比如张三的余额、影院的余票等。每个区块中会保留一段摘要，这段摘要为执行完区块中交易后以太坊状态的stateRoot，任一子状态的不同都将导致stateRoot的不同
  4. 如果张三控制的节点受到非法攻击或篡改，则执行结果及区块链数据将与网络中其他节点（如李四、赵四、影院的节点）不符，无法参与到网络的下一步共识中

Outline

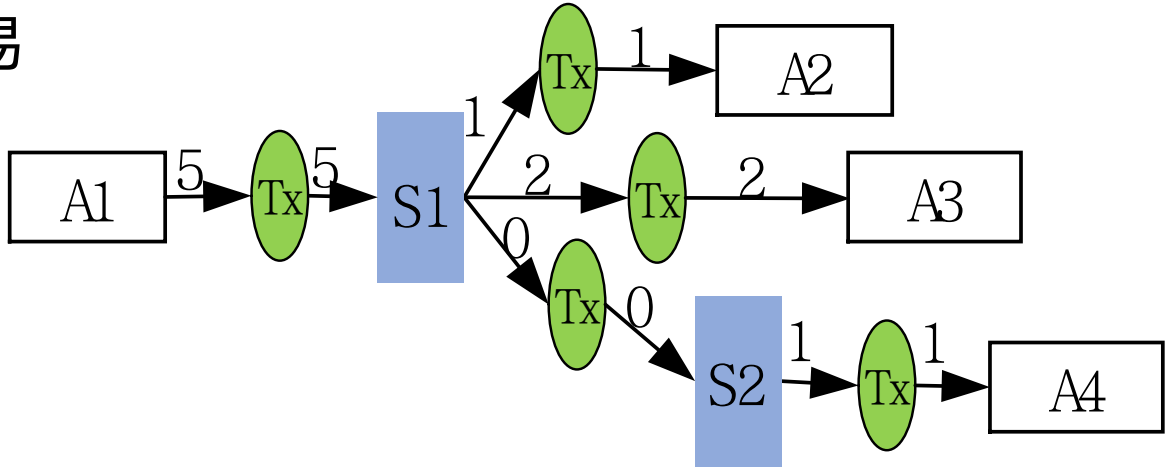
- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

以太坊和比特币

典型的比特币交易

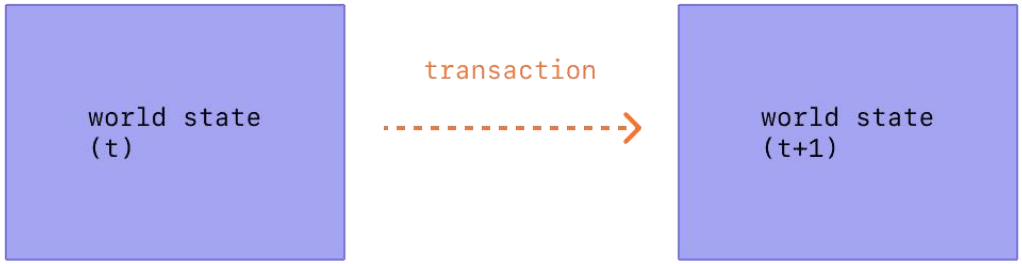


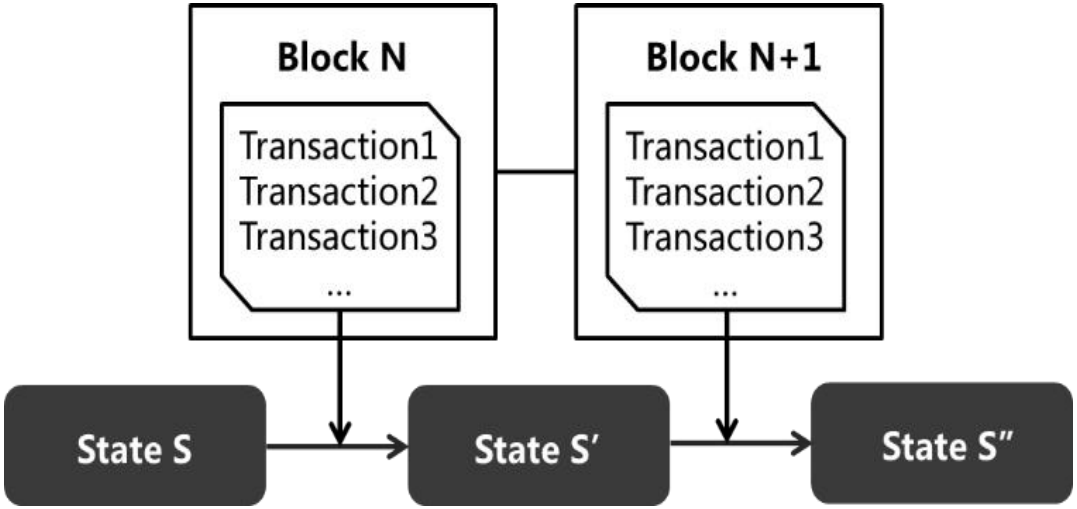
典型的智能合约交易



状态转移

- 以太坊状态转移:





以太坊中的状态转换模型示意图

交易内容

- 背景
  - 在以太坊中，交易承载了账户转账和合约创建、调用合约等功能
  - TX 数据的内容更为复杂，大体上可以粗略的分为三个部分，即基本的交易，驱动智能合约和交易的签名


```
{
  "from": "0x1923f626bb8dc025849e00f99c25fe2b2f7fb0db",
  "gas": "0x55555",
  "maxFeePerGas": "0x1234",
  "maxPriorityFeePerGas": "0x1234",
  "input": "0xabcd",
  "nonce": "0x0",
  "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
  "value": "0x1234"
}
```

交易内容 (cont.)

（1）基本交易内容

- from：交易发送者的地址。发送者地址可以通过合约的签名信息<r, s, v>计算得到，实现上交易的数据结构中并不会存储发送者地址
- to：交易接收者的地址。在进行转账时是接受转账金额的地址，在创建合约时设置为0x0000...000，在调用合约时则是合约的地址
- value：交易的金额，单位是Wei。Wei是以太币最小单位，我们常说的1个以太币是单位Ether，1 Ether =10^18 Wei

```
{
    from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "10",
    nonce: "0",
    value: "10000000000”
}
```

交易内容 (cont.)

（2）驱动智能合约

- input data：交易附带数据，传递创建合约的代码和构造函数，或调用合约的函数及参数
- nonce：交易发送者累计发出的交易数量，用于区分一个账户的不同交易及顺序
- gasPrice：发送者支付给矿工的gas的价格，用于实现从Gas到以太坊货币单位的转换，从而计算使用的Gas的总价格
- gasLimit：该交易允许消耗的最大的gas，用于解决智能合约不能停机的问题

```
{
    from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "10",
    nonce: "0",
    value: "10000000000”
}
```

交易内容 (cont.)

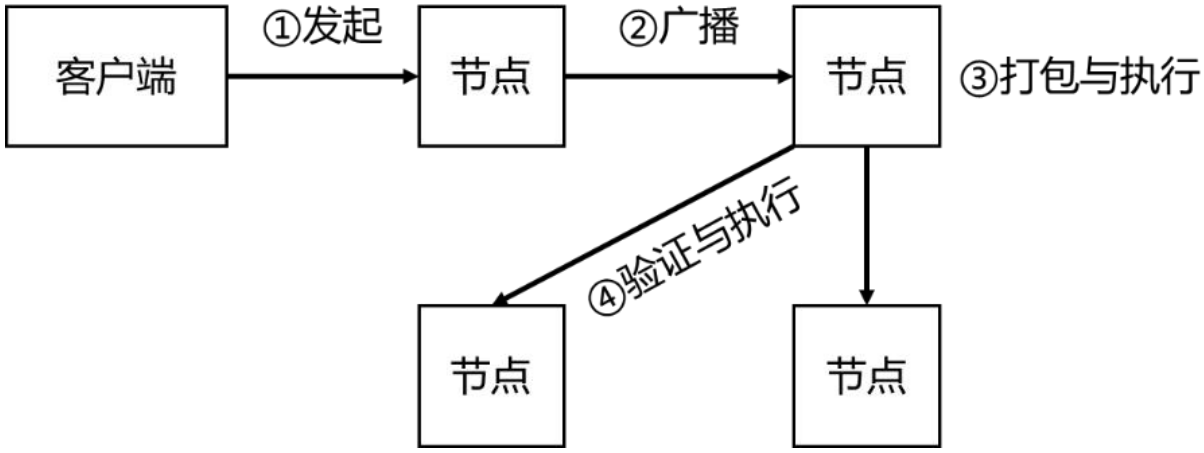
- (3) 交易的签名
  - hash：由以上字段生成的哈希值，也作为交易的id。
  - r、s、v：用于ECDSA验证的参数，由发送者的私钥对交易的哈希做数字签名生成，用于确认转账的合法性。

```
{
    "nonce": "0x0",
    "maxFeePerGas": "0x1234",
    "maxPriorityFeePerGas": "0x1234",
    "gas": "0x55555",
    "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
    "value": "0x1234",
    "input": "0xabcd",

    "v": "0x26",
     “r”: "0x223a7c9bcf5531c99be5ea7082183816eb20cfe0bbc322e97cc5c7f71ab8b20e",
    "s": "0x2aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc7704971491663",
    "hash": "0xeba2df809e7a612a0a0d444ccfa5c839624bdc00dd29e3340d46df3870f8a30e"
}
```

交易的周期

- 以太坊交易周期
  - 发起
  - 广播
  - 打包与执行
  - 验证与执行



以太坊交易周期

交易发起

- 设定交易内容
  - 用户在本地的以太坊钱包软件中选择要发送交易的地址（from），输入目标地址（to）、金额（value）、是否部署或调用合约（data）、手续费单价（gasPrice）等
- 发起交易
  - 用户确认发送至以太坊节点，多个用户各自保有钱包私钥，而通过同一个以太坊节点广播交易
- 以太坊钱包软件补充内容
  - 以太坊钱包软件将自动为用户得出交易所需的燃料上限（gasLimit），并给出用户地址对应的nonce值，最后使用私钥得到r、s、v，最终将交易序列化后发送到网络中
  - 部分客户端中，gasLimit与nonce也可以是用户自定义的。

交易广播

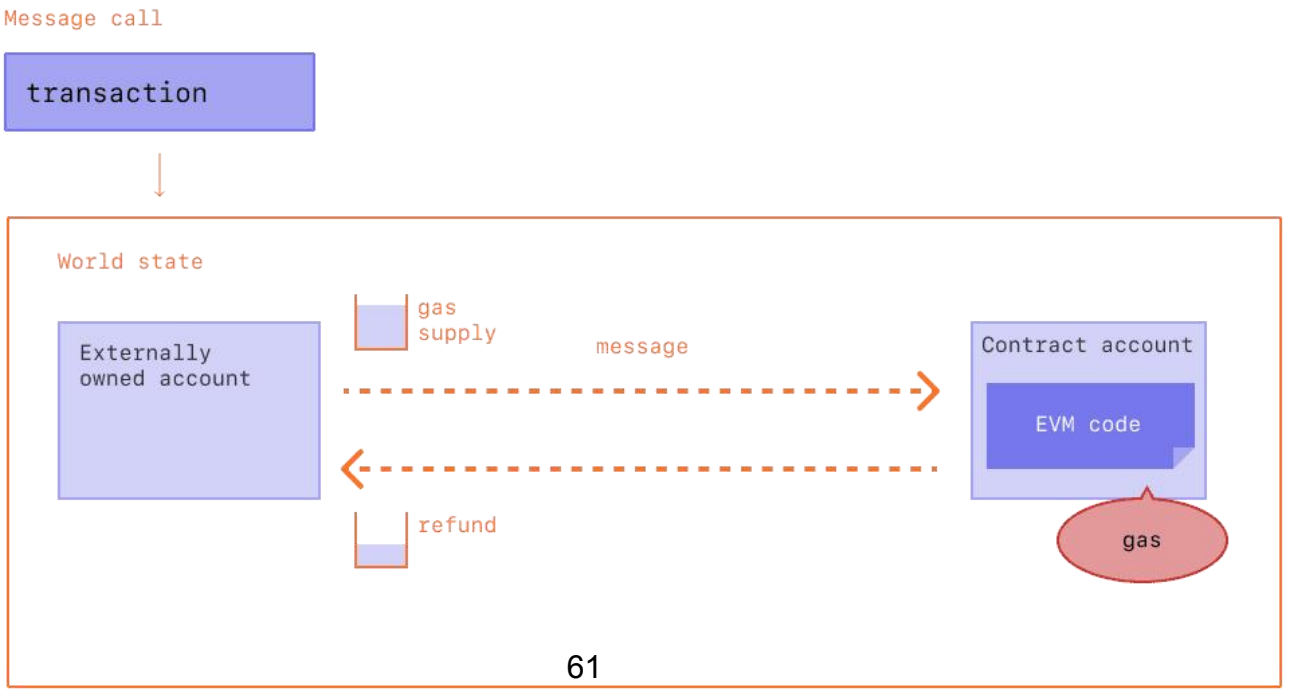
- 节点收到（或自己发起）交易后，对交易进行验证
- 节点验证交易为合法交易后，将交易加入节点的交易池中
- 节点验证交易通过后，除了加入节点的交易池中，还会根据P2P网络广播的策略向相邻节点继续广播该交易

交易打包和执行

- 打包和执行：
  - 交易进入内存池后，具有挖矿功能的全节点，开始打包下一个区块
  - 节点将交易打包时，对交易进行逐个执行，每笔以太坊交易都是对以太坊状态的修改
  - 在所有需要打包的交易执行后，交易、状态以及收据的信息也会打包到区块中
  - 记账节点在打包交易并获得合法的区块后，将区块（包含交易数据）广播到网络中的相邻节点

交易打包和执行

- 交易类型：根据 to 值的不同，将交易分为3种执行类型
  - 创建合约交易
  - 调用合约交易
  - 普通转账交易



交易打包和执行

- 普通转账交易：to 为外部账户 的交易
  - 直接将以太币金额从 from 转到 to

```
{
    from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "10",
    nonce: "0",
    value: "10000000000”
}
```

交易打包和执行

- 创建合约交易： to 为空 的交易
  - EVM 根据 from 值及 nonce 值生成合约地址
  - 执行 data 中对应的智能合约代码
  - 将合约 EVM 代码存储到合约地址中

```
{
    from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    to: " ",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "10",
    nonce: "0",
    value: "10000000000”
}
```

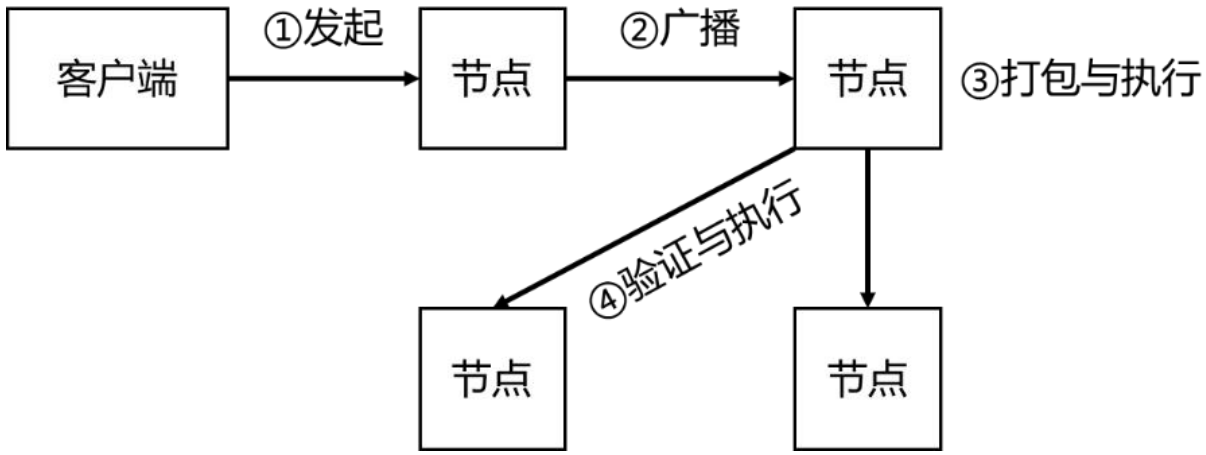
交易打包和执行

- 调用合约交易：to 为合约账户 的交易
  - EVM 将从世界状态中获取 to 地址中存储的 EVM 代码
    - to 地址中存储的是合约本身
  - 执行交易的 data 字段中包含的代码
    - data 中则包含了调用合约的相应函数及其参数
  - 本质上讲，对合约的调用，是对合约状态的修改。

```
{
    from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
    to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "10",
    nonce: "0",
    value: "10000000000”
}
```

交易验证和执行

- 没有获得记账权的节点，在收到广播的区块后，对区块进行合法性的验证，并进行交易的执行



Outline

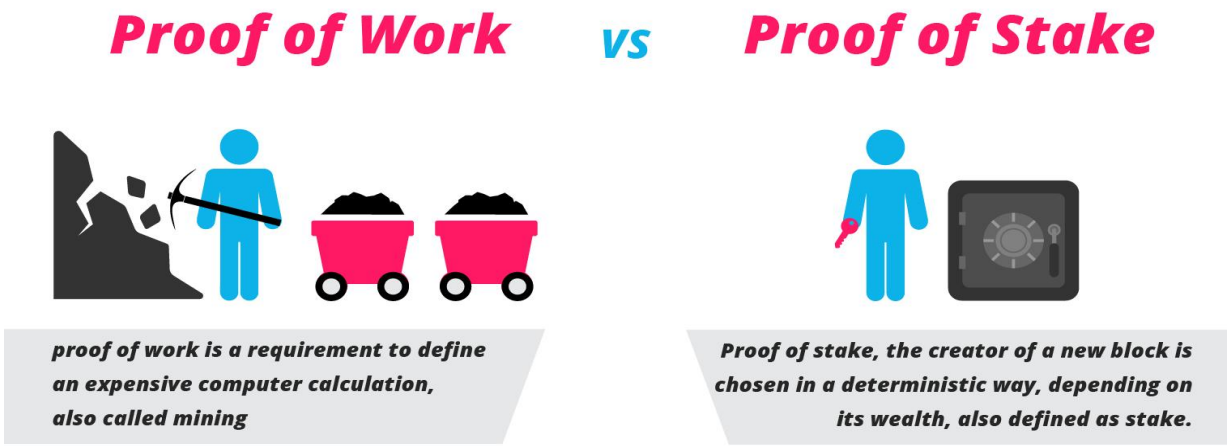
- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

以太坊的 共识机制

- 基于 PoW 机制
  - 增加的特性：ASIC集成电路 resistance: memory hard puzzle
  - 因为以太坊采用的哈希算法是 ethash 算法，需要频繁对内存访问（ASIC resistance)。更换了哈希函数（比特币使用的是SHA256 函数，所以可以使用ASIC芯片挖BTC)
  - 抑制硬件挖矿，好还是不好?
    - 正反两方面影响：整体算力减弱了，虽然节约了资源，但是攻击的成本下降了
- 将来过渡为 PoS (Proof-of-Stake)

以太坊将来的共识机制：PoS

- **Proof-of-Stake 权益证明**
  - 最初由 Sunny King 2012年在论文“PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake”中提出，这种机制通过计算你持有币数占总币数的百分比，包括你占有币数的时间来决定你获得本次记账权利的概率。**持有越多，获得记账权力概率越大**

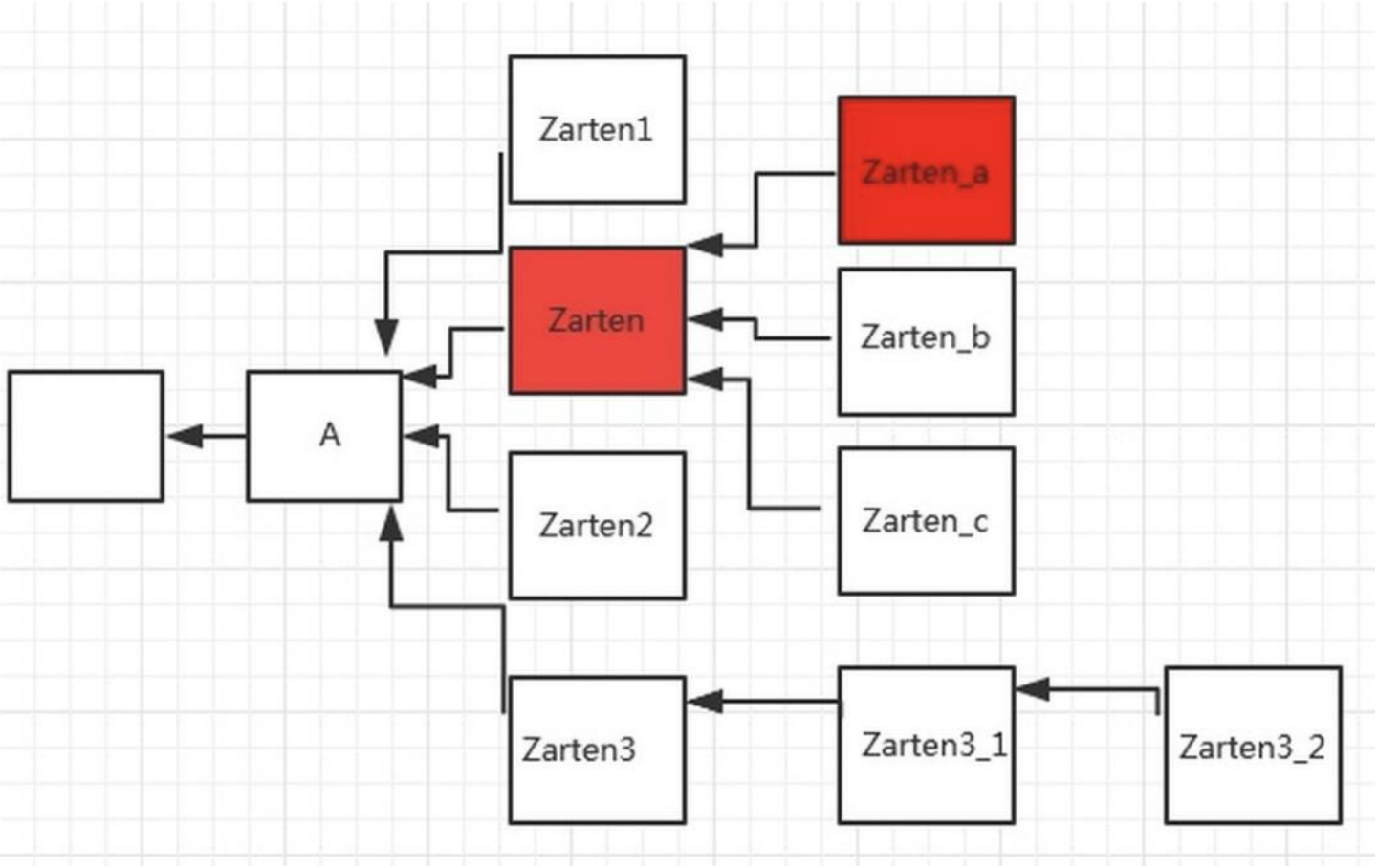


- 代表币种：如 Qtum 币，以太坊也部分采用POS机制
- 优点：相比 PoW 缩短了达成共识时间；节省能源
- 缺点：容易分叉；易中心化（马太效应）

新 PoW 共识：以太坊的分叉

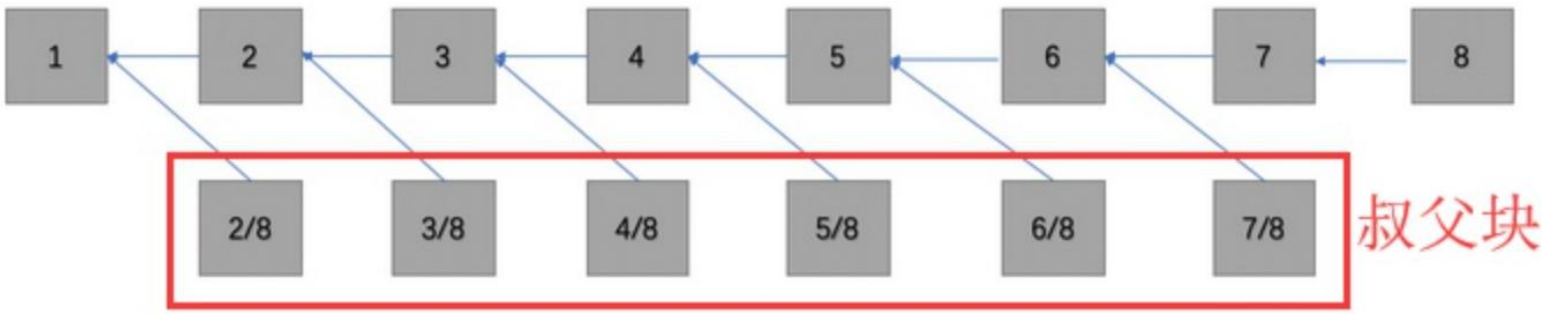
- 以太坊的出块时间被定为15s左右。
- 在15秒的时间内，一个新发布的区块很很大可能还没有扩散到整个区块链网络，导致以太坊的分叉是常态
  - 分叉上的块被废弃，打击积极性
  - 安全上的考虑，有什么坏处？
- 为了鼓励分叉的合并，以太坊的设计中引入了 Ghost 协议
  - GHOST (Greedy Heaviest Observed SubTree)
  - 由Yonatan Sompolinsky 和 Aviv Zohar在2013年12月引入的创新

GHOST 规则


Zarten1
Zarten_a
Zarten
Zarten_b
A
Zarten2
Zarten_c
Zarten3
Zarten3_1
Zarten3_2

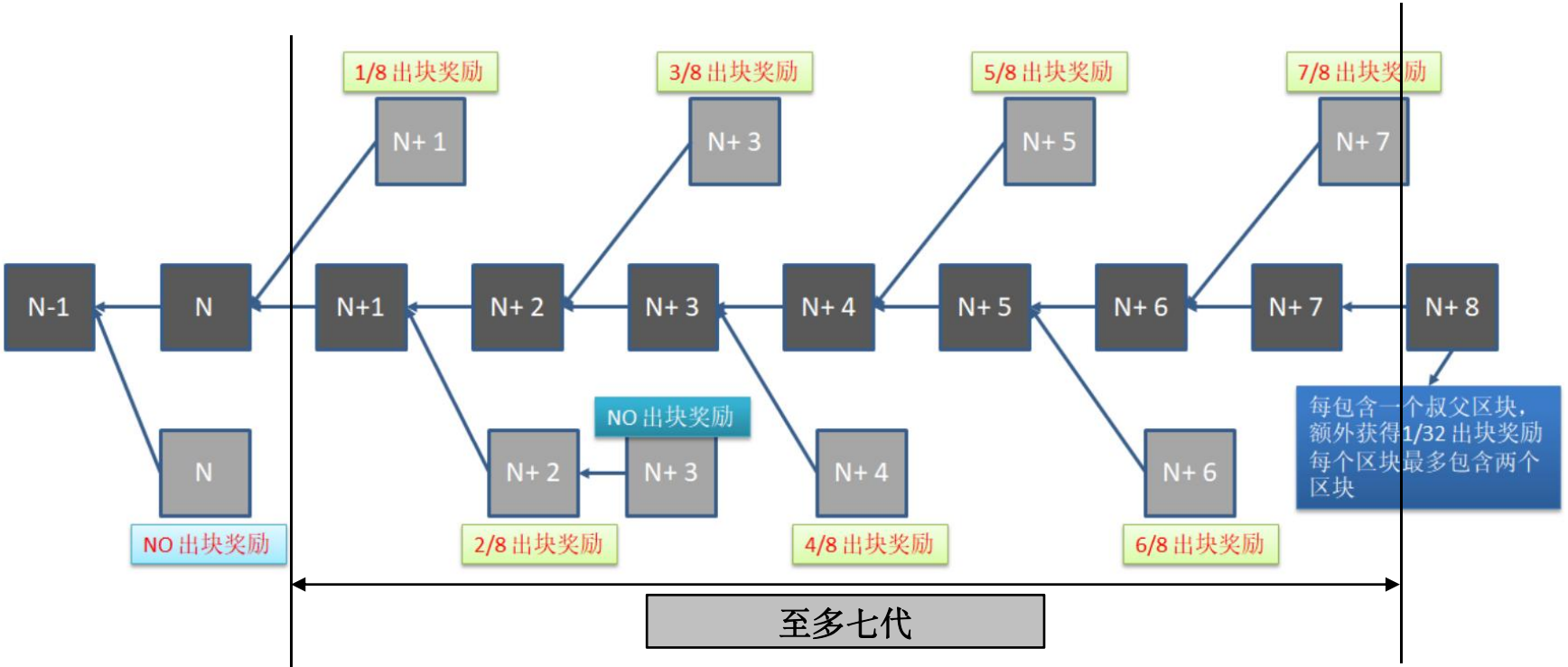
叔块

- 定义
  - 以太坊定义了不在主链但被主链区块记录的满足难度的区块为叔块（uncle block）。
  - 在以太坊中叔父不是严格意义上的叔父，在以太坊中规定在当前区块的7代以内有共同祖先的都可以认为是叔父块。
- 作用
  - 在尽可能减少两个相邻区块产生时间的条件下，尽量收缩和统一整个区块链的主链，同时通过叔块的激励来维护矿工的积极性。



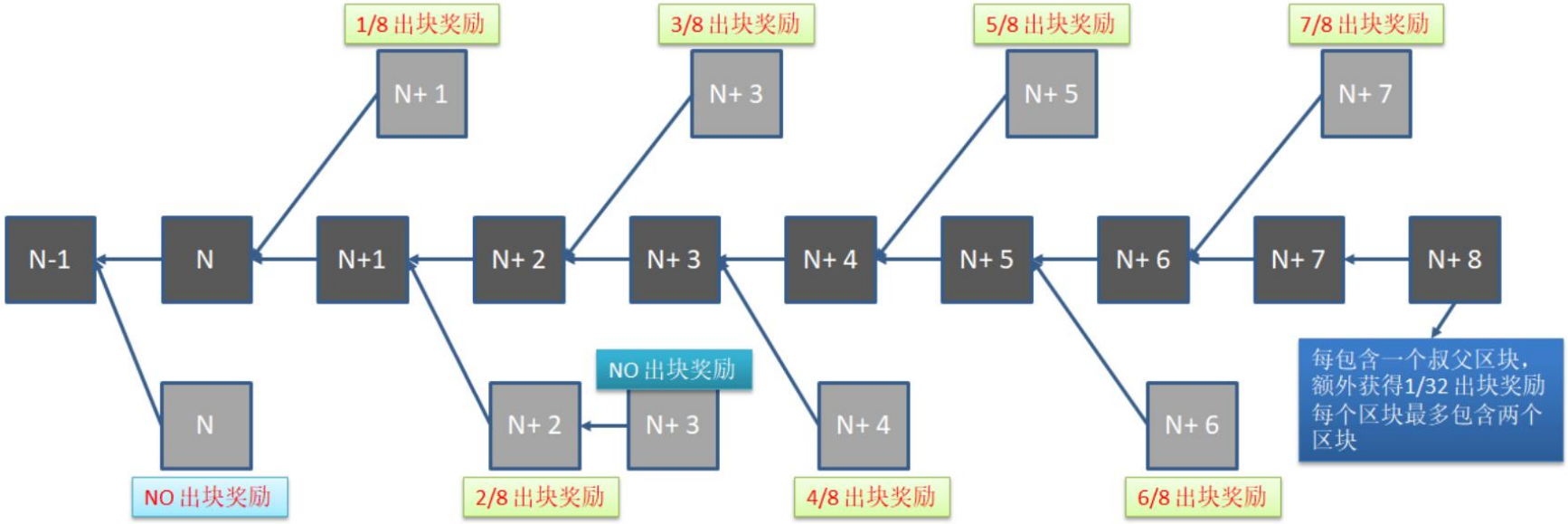
叔块特点

- 某个区块最多只能接纳2个叔父块，也可以不接纳任何叔父块
- 叔父块必须是区块的前2层～前7层的祖先的直接子块
- 被接纳过的叔父块不能再重复接纳了
- 接纳了n个叔父块的区块，可以获得出块奖励的n*1/32
- 被引用了的叔父块，随着距离越远，得到的奖励递减1/8

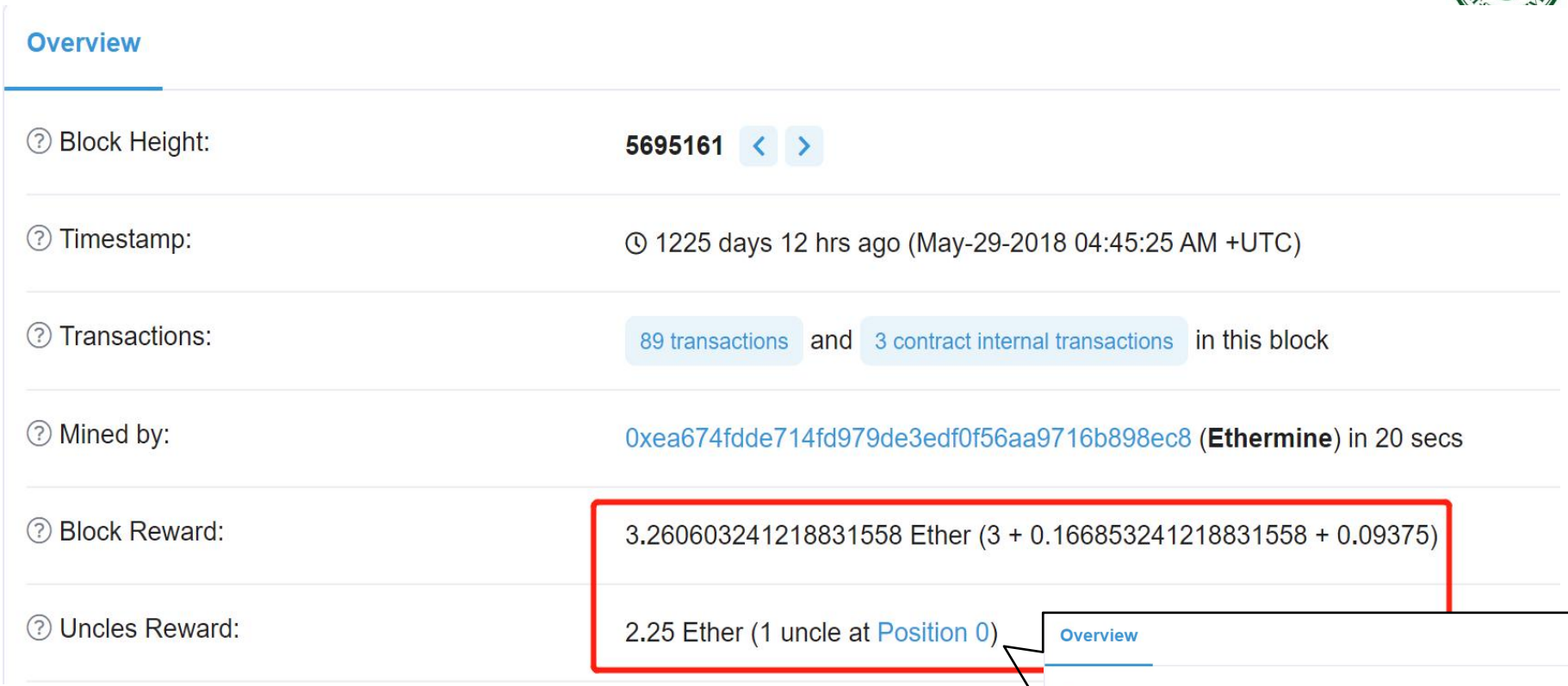


叔块

- Question: 叔父区块中的交易会被执行吗?
  - 不需要。
  - 很可能叔父区块与当前区块之间的交易是存在冲突的交易，实际上以太坊中不会执行叔父区块中的交易，
  - 而只检查这个叔父是不是符合挖矿难度要求的合法区块。



叔块奖励例子#1



原始奖励：3
手续费：0.166
一个叔块，额外奖励：3* (1/32) = 0.09375
叔块Height = 5695159，奖励：3*（6/8）=2.25

source: https://cn.etherscan.com/block/5695161

叔块奖励例子#2

Overview

Block Height:	**5695150**
Timestamp:	1225 days 12 hrs ago (May-29-2018 04:41:51 AM +UTC)
Transactions:	160 transactions and 9 contract internal transactions in this block
Mined by:	0xea674fdde714fd979de3edf0f56aa9716b898ec8 (**Ethermine**) in 14 secs
Block Reward:	3.31510552614492296 Ether (3 + 0.12760552614492296 + 0.1875)
Uncles Reward:	4.875 Ether (2 uncles at Position 0, Position 1)

Uncle Height:	**5695149**
Uncle Position:	0
Block Height:	5695150

Uncle Height:	**5695148**
Uncle Position:	1
Block Height:	5695150

原始奖励：3
手续费：0.127
2个叔块，额外奖励：3* (1/32)*2 = 0.1875
叔块奖励：3*（7/8 + 6/8）=4.875

source: https://cn.etherscan.com/block/5695150

以太币的总供应量

- 创世区块包含大部分以太币（约7200W）。
- “挖矿挖的再努力，关键还是不能输在起跑线上”。

Ether Distribution Overview

Genesis (60M Crowdsale 12M Other):	72,009,990.50 Ether
+ Mining Block Rewards:	42,862,582.34 Ether
+ Mining Uncle Rewards:	2,933,324.63 Ether
+ Eth2 Staking Rewards:	274,011.61 Ether
- Burnt Fees:	434,091.46 Ether
= Current Total Supply	**117,645,817.62 Ether**

Data Source: Total Ether Supply API

Price per Ether

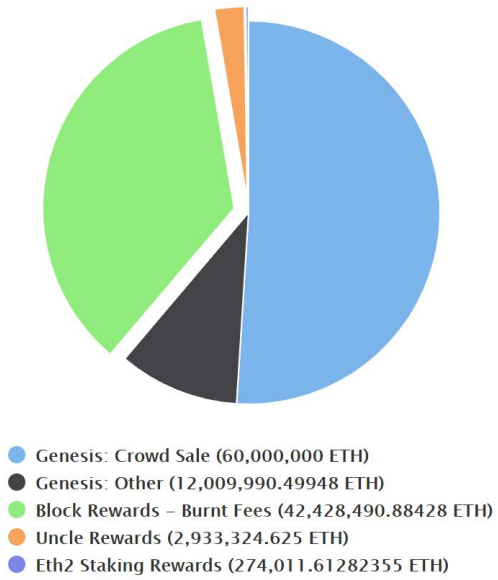
In USD:	$3,440.31
In BTC	0.0685

Data Source: CryptoCompare

117,645,817.62
Total Ether Supply

$404,738,082,822.00
Market Capitalization

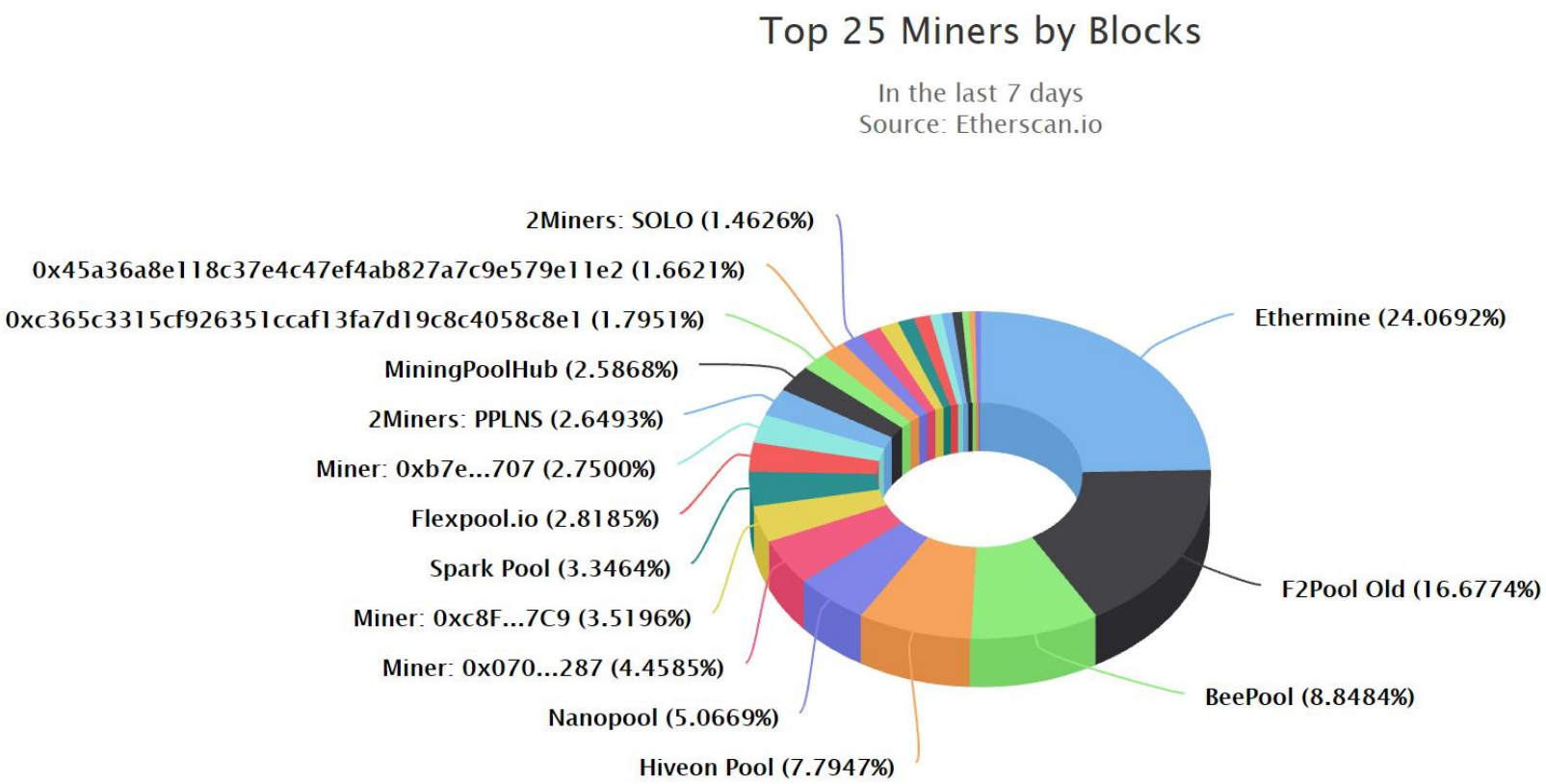
Breakdown by Supply Types



source: https://cn.etherscan.com/stat/supply，2021.10.6

以太坊的算力分布

- 挖矿的集中化程度也很高。



最大的25个以太坊矿池所占的算力比重

source: https://cn.etherscan.com/stat/miner?range=7&blocktype=blocks， 2021.10.6

以太币价格

- 2021年大涨

以太币价格随时间变化

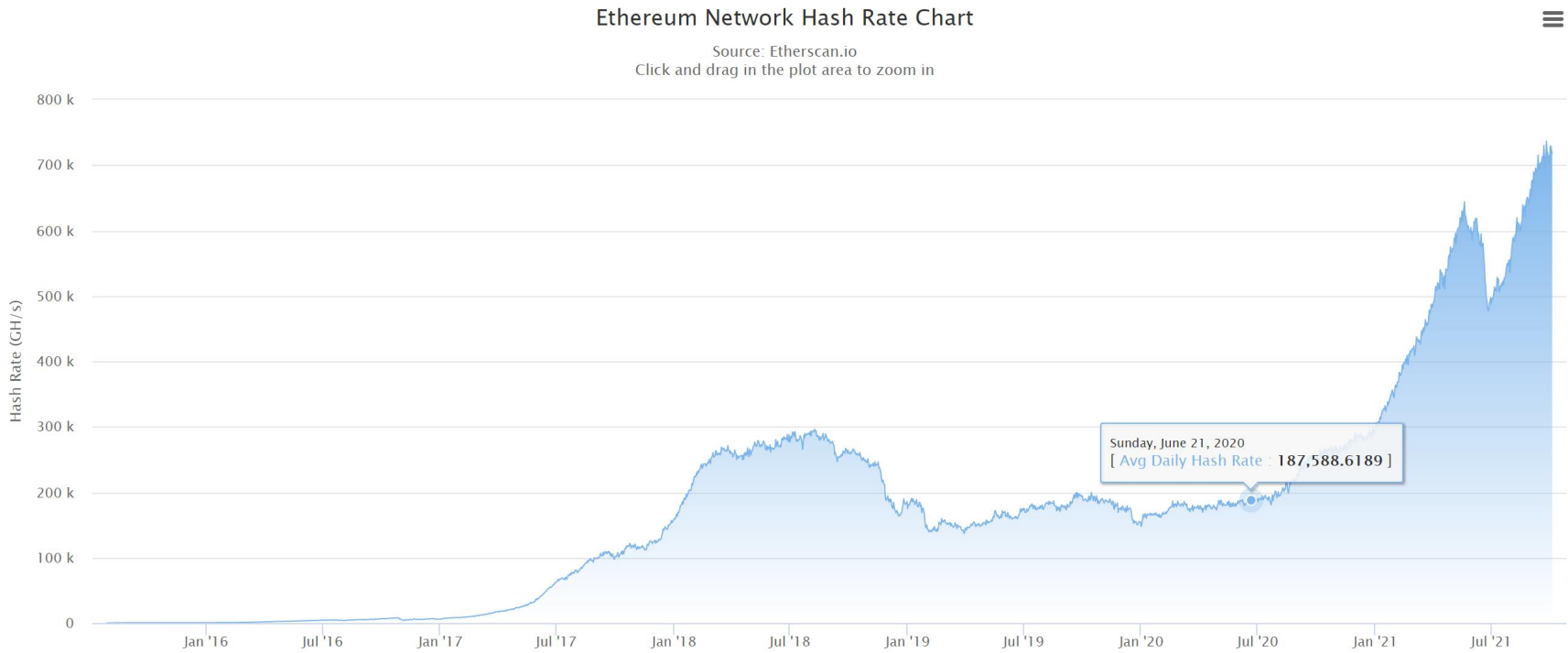
source: https://cn.etherscan.com/chart/etherprice , 2021.10.6

以太坊算力

- 2021年算力也大涨

Highest Avg Hash Rate of **736,674.3172 GH/s** was recorded on Saturday, September 25, 2021

Lowest Avg Hash Rate of **11.5297 GH/s** was recorded on Thursday, July 30, 2015



以太币算力随时间变化

source: https://cn.etherscan.com/chart/hashrate， 2021.10.6

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

难度调整

$$D(H) \equiv \begin{cases} D_0 & \text{if} \quad H_{\mathrm{i}} = 0 \\ \max\left(D_0, P(H)_{H_{\mathrm{d}}} + x \times \varsigma_2\right) + \epsilon & \text{otherwise} \end{cases}$$

where:

$$D_0 \equiv 131072 \tag{42}$$

- $D(H)$是本区块的难度，由基础部分$P(H)_{Hd} + x \times \varsigma_2$和难度炸弹部分$\epsilon$相加得到。
  - $P(H)_{Hd}$为父区块的难度，每个区块的难度都是在父区块难度的基础上进行调整。
  - $x \times \varsigma_2$ 用于自适应调节出块难度，维持稳定的出块速度。
  - ϵ 表示设定的难度炸弹。
- 基础部分有下界，为最小值$D_0 = 131072$。

难度调整

- 自适应难度调整：$x \times \varsigma_2$

$$x \equiv \left\lfloor \frac{P(H)_{H_{\mathrm{d}}}}{2048} \right\rfloor \tag{43}$$

$$\varsigma_2 \equiv \max\left(y - \left\lfloor \frac{H_{\mathrm{s}} - P(H)_{H_{\mathrm{s}}}}{9} \right\rfloor, -99\right) \tag{44}$$

  - x是调整的单位，ς_2为调整的系数。
  - y和父区块的uncle数有关。如果父区块中包括了uncle，则y为2，否则为1。
    - 父块包含uncle时难度会大一个单位，因为包含uncle时新发行的货币量大，需要适当提高难度以保持货币发行量稳定。
  - 难度降低的上界设置为-99，主要是应对被黑客攻击或其他目前想不到的黑天鹅事件。

难度调整

$$(43) \qquad x \equiv \left\lfloor \frac{P(H)_{H_d}}{2048} \right\rfloor$$

$$(44) \qquad \varsigma_2 \equiv \max\left(y - \left\lfloor \frac{H_s - P(H)_{H_s}}{9} \right\rfloor, -99\right)$$

- H_s是本区块的时间戳，$P(H)_{H_s}$是父区块的时间戳，均以秒为单位，并规定$H_s > P(H)_{H_s}$。
  - 该部分是稳定出块速度的最重要部分：出块时间过短则调大难度，出块时间过长则调小难度。
- 以父块不带uncle的情况$(y = 1)$为例：
  - 出块时间在[1,8]之间，出块时间过短，难度调大一个单位。
  - 出块时间在[9,17]之间，出块时间可以接受，难度保持不变。
  - 相差时间在[18,26]之间，出块时间过长，难度调小一个单位。
  - …

难度炸弹

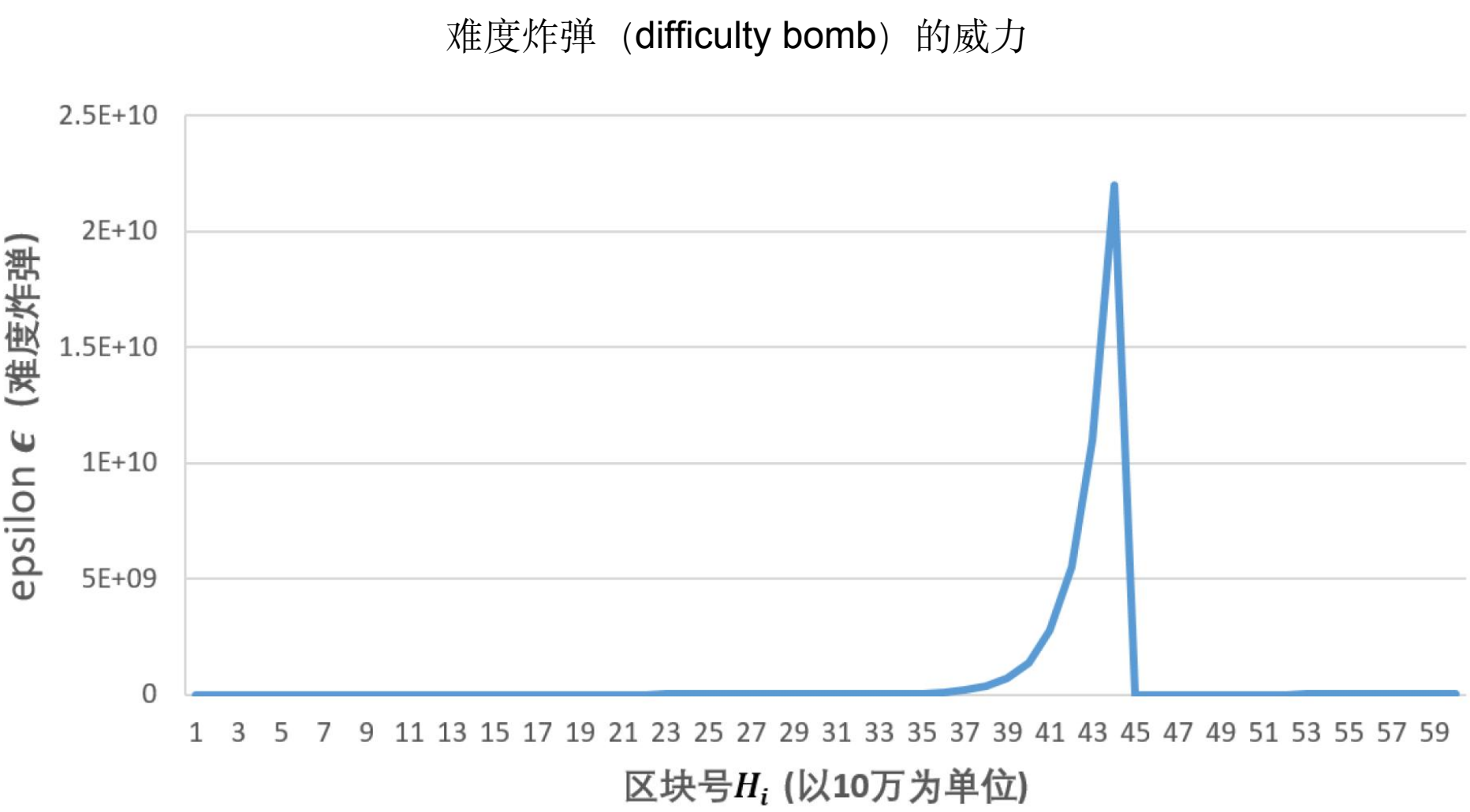
难度炸弹

$$\epsilon \equiv \left\lfloor 2^{\left\lfloor H_{\mathrm{i}}' \div 100000 \right\rfloor - 2} \right\rfloor$$

$$H_{\mathrm{i}}' \equiv \max(H_{\mathrm{i}} - 3000000, 0)$$

- ϵ每十万个块扩大一倍，是2的指数函数，到了后期增长非常快，这就是难度“炸弹”的由来。
- 设置难度炸弹的原因是要降低迁移到PoS协议时发生fork的风险：到时挖矿难度非常大，所以矿工有意愿迁移到PoS协议。
- H_i'称为fake block number，由真正的block number H_i减少三百万得到。这样做的原因是低估了PoS协议的开发难度，需要延长大概一年半的时间(EIP100)。

难度炸弹


难度炸弹（difficulty bomb）的威力
2.5E+10
2E+10
1.5E+10
1E+10
5E+09
0
epsilon ϵ（难度炸弹）
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
区块号H_i (以10万为单位)

以太坊发展的四个阶段

2015 年 7 月 → Frontier（前沿）阶段
- 仅开发者测试使用

2016 年 3 月 → Homestead（家园）阶段
- 提供图形界面，提升安全性和平稳性

2017 年 10 月 → Metropolis（大都会）阶段
- 又分为Byzantium和Constantinople两个子阶段
- 难度炸弹的回调发生在Byzantium这个子阶段，在EIP（Ethereum Improvement Proposal）中决定
- 同时把block reward从5个ETH降为3个ETH

至今未完成 → Serenity（宁静）阶段
- 预期将共识机制从工作量证明(Proof of Work，简称 PoW)转向 PoS，使持币者也能有一定的决定权

难度炸弹的调整对出块的影响

- 早期的难度调整，主要是为了稳定出块时间。
- 出块时间大幅度增长，是因为难度炸弹的效应。
- 调整后总的出块时间又恢复到原来的水平。

source: https://cn.etherscan.com/chart/blocktime ， 2021.10.6

Outline

- 以太坊概述
- 以太坊的账户模型
- 以太坊的状态树
- 以太坊的交易树、收据树、Bloom Filter
- 以太坊基本架构及原理
- 以太坊交易
- 以太坊共识机制
- 以太坊挖矿难度调整
- 以太坊区块结构

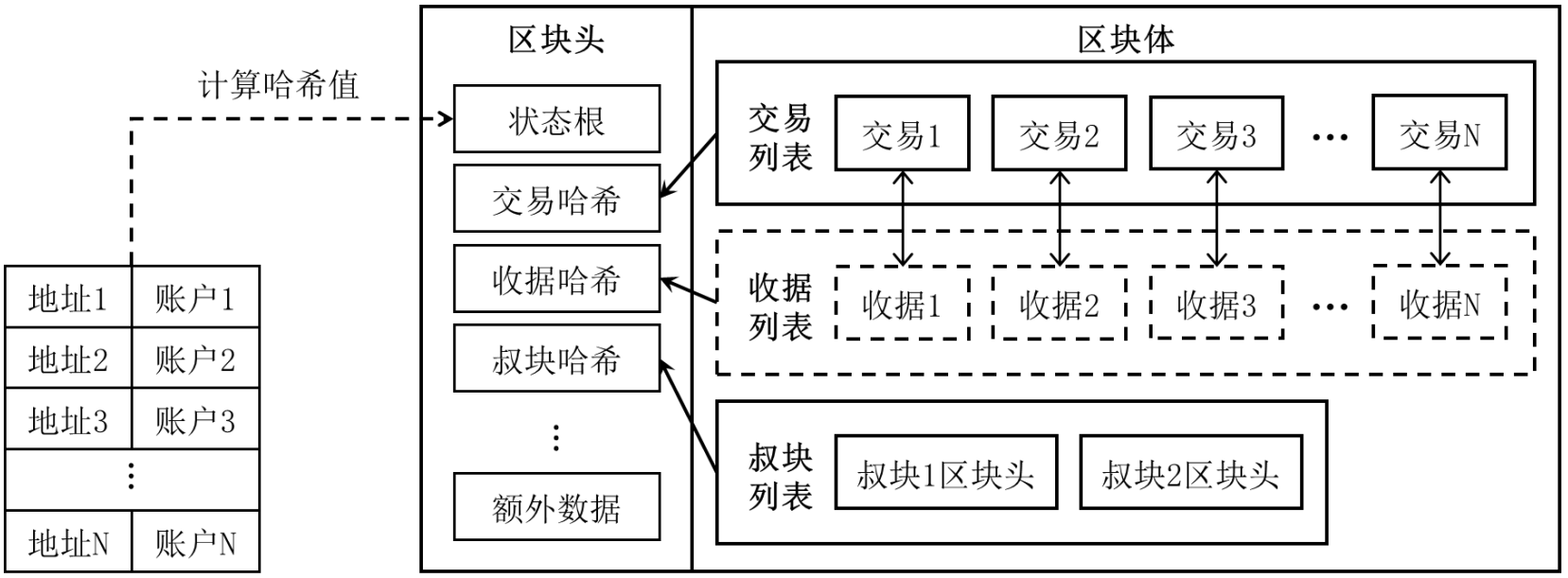
区块头

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

- 父块哈希
- 叔块哈希
- 矿工地址
- 状态树哈希
- 交易树哈希
- 收据树哈希
- Bloom Filter
- 挖矿难度
- 区块号
- Gas上限
- 所有交易gas之和
- 区块时间戳
- 可变长度字段
- 工作量证明摘要
- 挖矿Nonce值

以太坊区块链的 区块数据结构

- 以太坊的区块
  - 打包一批执行后的交易
  - 它的数据结构同样分成了区块头和区块体。



以太坊区块链的 区块数据结构

- 以太坊区块头内容
  - 记录以太坊状态的状态根
  - 交易列表、收据列表和叔块列表对应的哈希值
  - 最长不超过100KB的额外数据

- 以太坊区块体内容
  - 交易组成的交易列表
  - 由交易执行信息组成的收据列表
  - 用于改进以太坊共识过程的叔块列表