

# 区块链系统入门

Nanyan@2022/12



# 区块链系统极简概述

区块链是一种分布式账本技术，本质就是要解决  
去中心化、去信任中介的问题。



# 本质概要

源起: 2008 [中本聪](#) 《[Bitcoin: A Peer-to-Peer Electronic Cash System](#)》=>

2009 BTC网络运行

比特币: 去中心化的, 在零信任的环境下运行的电子货币系统。

区块链: 本质就是要解决去中心化、去信任中介的问题。

去中心化、去信任中介

How?

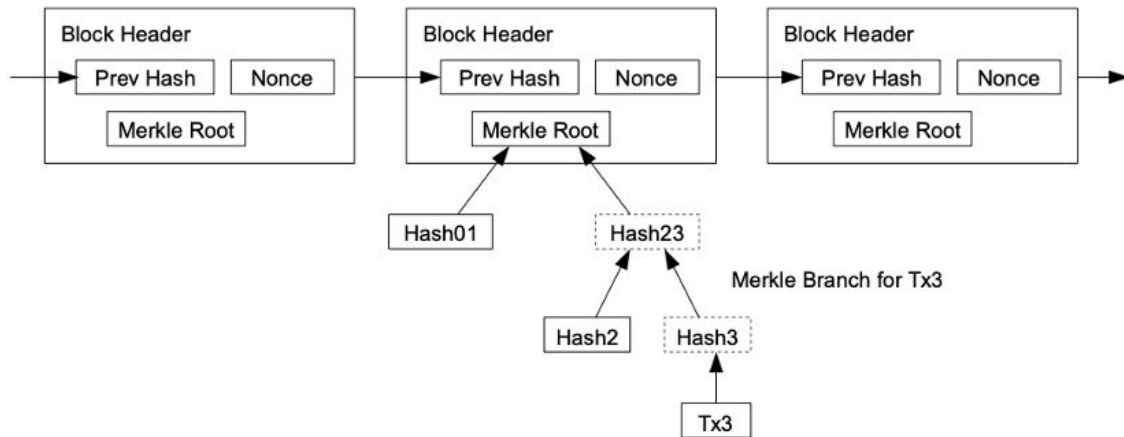
开放开源  
多数共识  
Code is law!

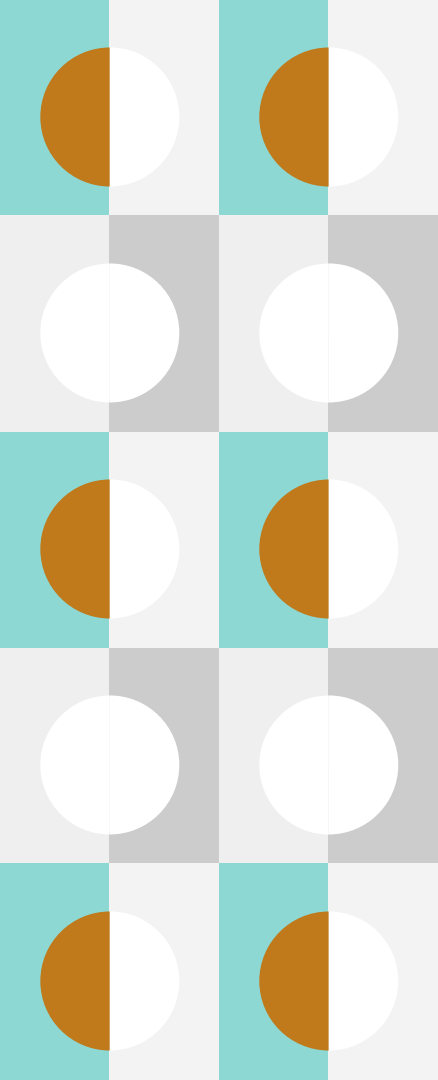
非对称加密  
数字签名  
单向消息摘要  
Merkle tree

以“块”为号  
链式追溯  
不可篡改

P2P网络  
无中心  
无准入

Longest Proof-of-Work Chain

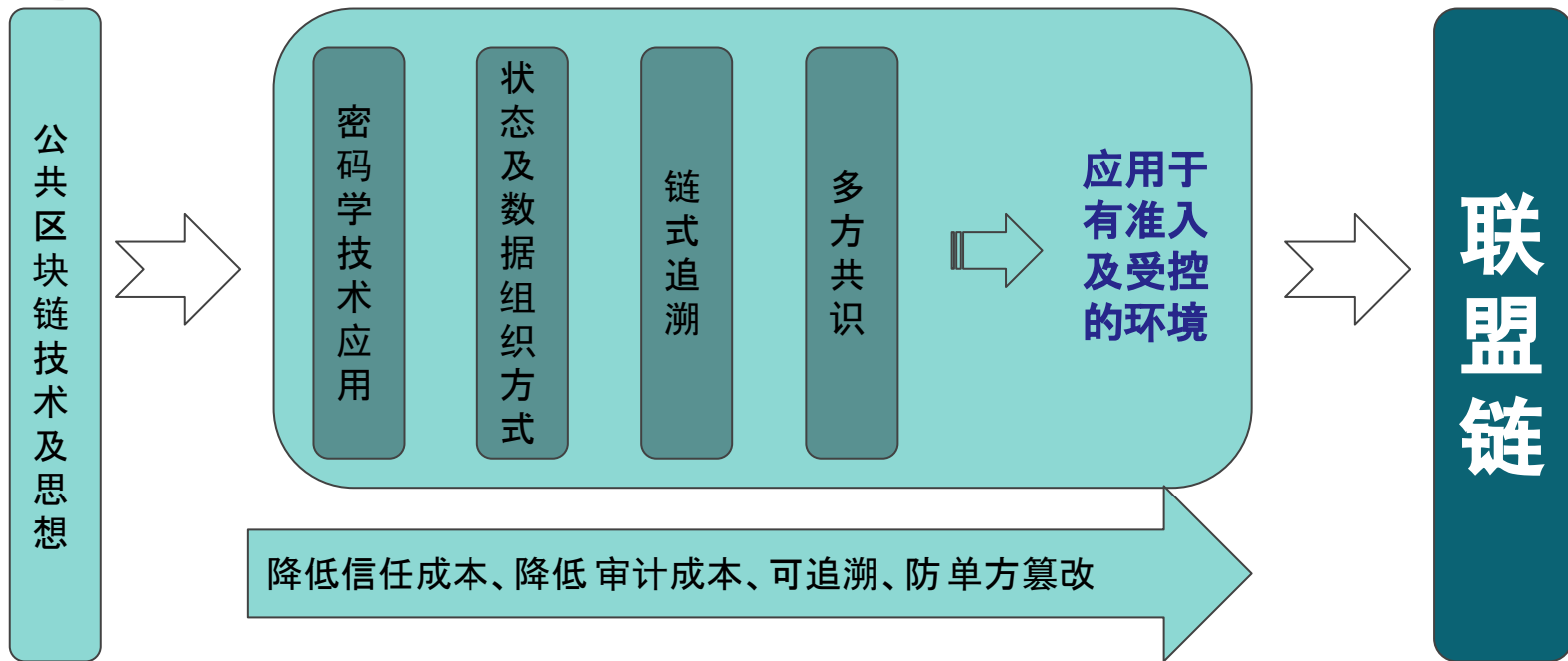




区块链系统就是一堆独立参与者在互不信任的前提下，通过p2p网络，依据某种共识机制实现对一个全局账本进行一致性记账，并以链式结构对账本历史进行记录和追溯的，去中心化、开放开源、抗审查的系统。

-----  
Nanyan 2022

# 联盟链



本次介绍将只讨论公共区块链的内容

# 部分知名公链简介

# 比特币

匿名者[中本聪](#)创建，是区块链的鼻祖。

是加密货币系统，具有极强的金融属性，而这也是它的唯一目的。

基于PoW+最长链原则，解决了记账权以及双花问题。

基于难度值控制区块生产速度，约10分钟/区块；2016区块调整一次难度值。

基于增发的区块奖励+交易手续费 激励矿工，区块奖励每4年减半。  
BTC总额上限2100万。



# 比特币--PoW Pros and Cons

## Pros

- one-cpu-one-vote
- 真正去中心化、无准入

## Cons

- 能源消耗巨大
- 51%攻击问题
- 效率低, 交易确认时间长

# 比特币

- 账号基于椭圆曲线密码学, 采用secp256k1曲线;
- 私钥即一切; 公钥推导出地址代表链上身份;
- 余额采用UTXO模型, 转账需引用之前交易的输出, 即收集未花费交易输出作为当前交易输入

```
Tx1
{ Inputs:[...]
  Outputs:[
    {address: A, Amount: 10},
    {...}
  ]
}
```

```
Tx2
{ Inputs:[...]
  Outputs:[
    {address: A, Amount: 20},
    {...}
  ]
}
```

```
Tx3
{ Inputs:[
  {Tx1,IndexOfTx1},
  {Tx2,IndexOfTx2}
]
  Outputs:[
    {Address: B, Amount: 25},
    {Address: A, Amount: 5}
  ]
}
```

# 以太坊 Ethereum

- 2013 [Vitalik Buterin](#) “一个图灵完备的可编程和通用区块链”
- [Gavin Wood](#) 加入, 2015 以太坊公链启动
- Ethash: 加入内存要求的PoW, 之后于2022/9/15 成功转PoS ([The Merge](#))
- Account 账户模块, 全局统一状态, MPT结构



# 以太坊 Ethereum

Account 账户模块, 全局统一状态

图灵完备智能合约编程语言 Solidity

以太坊虚拟机 EVM

图灵完备的可编程和通用区块链

区块链 2.0

公链之王

# 以太坊 Ethereum

## 以太坊扩容生态

应用繁荣, 性能瓶颈愈加显现

区块链扩容成为以太坊第一要务

### 二层扩容

- zk-rollup
- op-rollup
- 状态通道

### 侧链类扩容

- 侧链
- Plasma Chain
- Validium

### 需求溢出, EVM兼容链

- BSC
- HECO
- ...

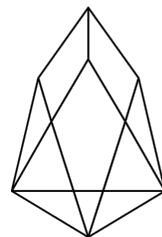
### 异构公链兼容 EVM “蹭流量”

- EVMOS
- Tron
- Avalanche C-chain
- ...

## 部分其他较知名公链



SOLANA



EOS



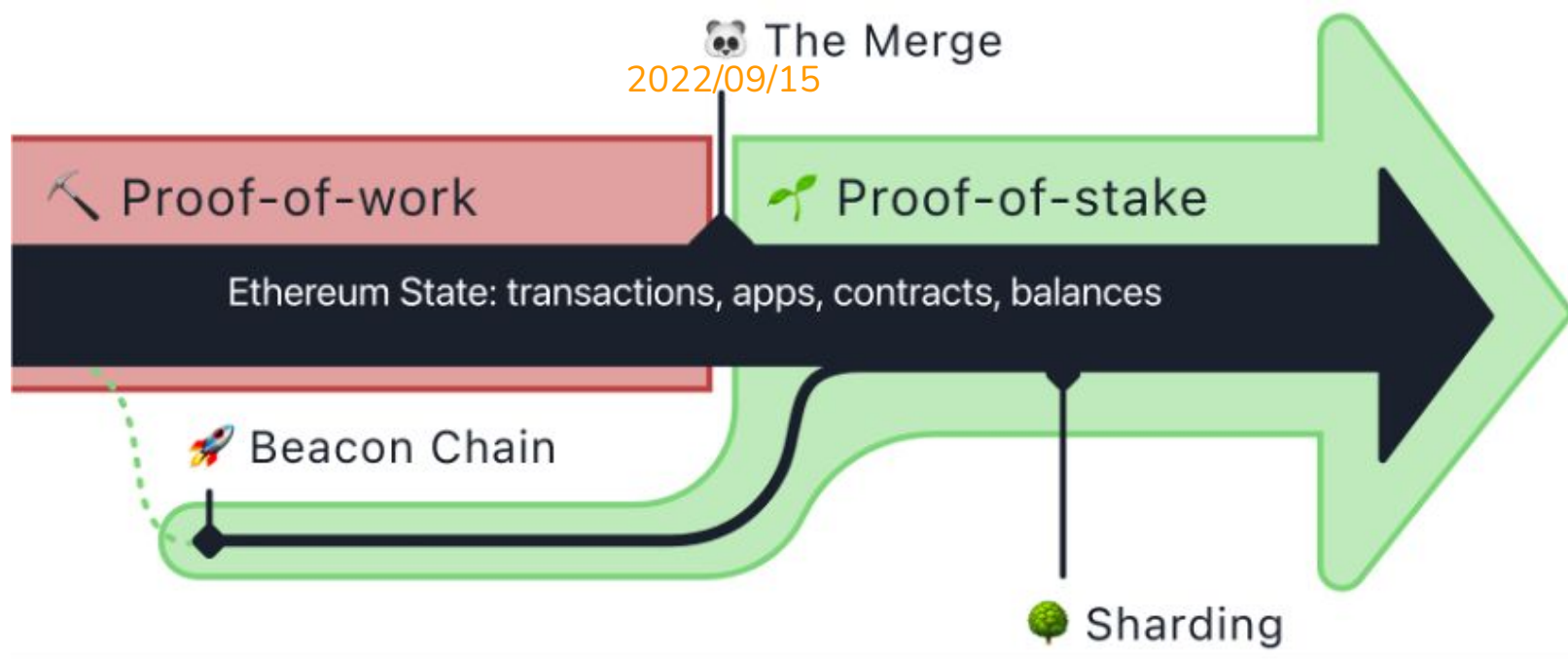
AVALANCHE





# 以太坊

# The Merge: 从PoW到PoS





# The Merge: 从PoW到PoS

Beacon chain

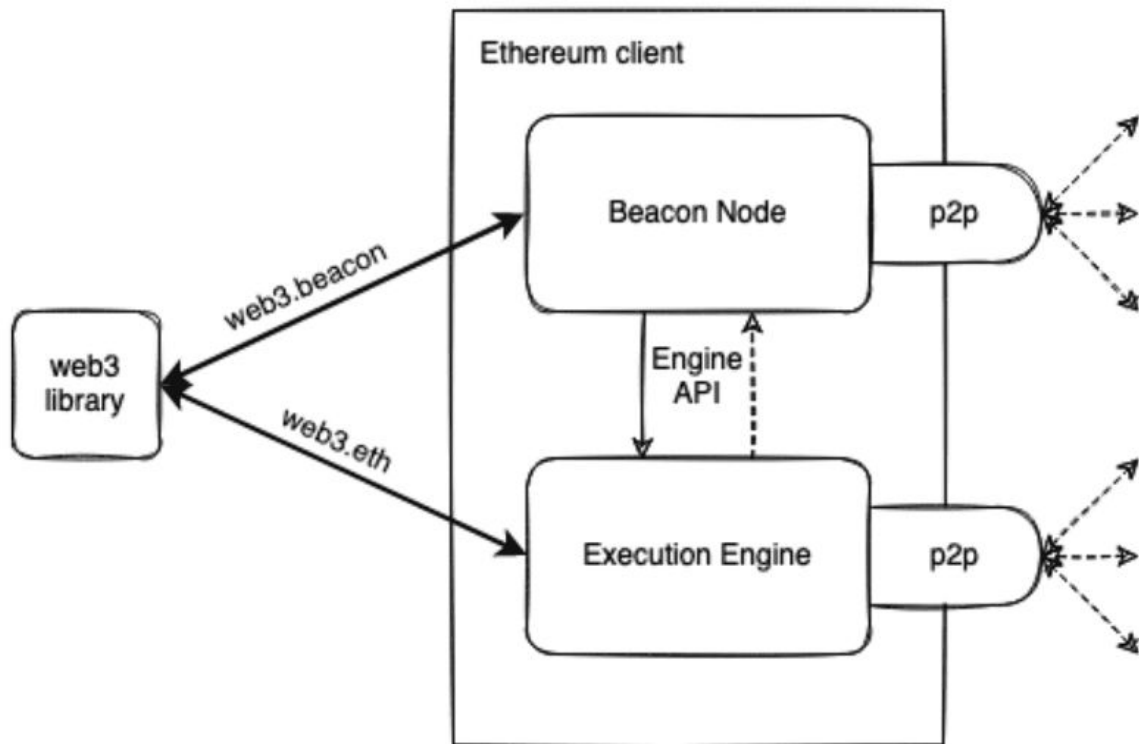


Execution Shard (ETH1)

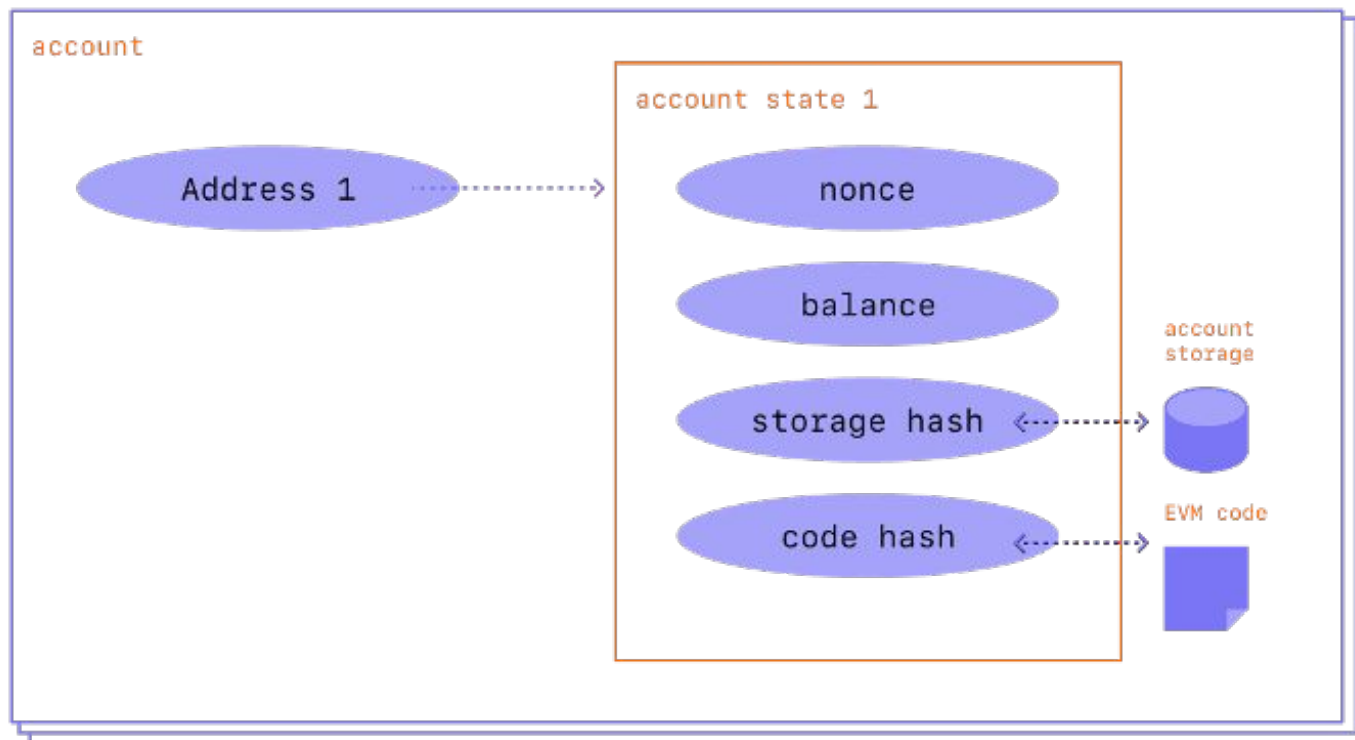
1. Validator抵押管理、提议及验证共识层区块(共识层区块包含执行层区块完整数据, 不包含执行层状态)、投票等共识相关业务;
2. BP调用执行客户端构建执行层区块数据;调用执行客户端验证执行区块、更新执行层链;
3. 共识层p2p网络通信, (含区块传播)

1. 接收共识客户端提供的执行层区块数据, 验证执行层区块, 维护执行层链的状态, 即负责EVM虚拟机;
2. 执行层txpool
3. 执行层p2p网络(含交易、状态传播);
4. 供共识客户端调用的共识接口。

# The Merge: 从PoW到PoS



# 核心概念 - 账号





# 核心概念 - 账号

## 账号类型

### EOA

1. 接收、持有和发送 ETH 和 token
2. 与已部署的智能合约进行交互
3. 创建帐户是免费的
4. 可以发起交易
5. EOA之间只能进行 ETH 转账

### Contract

1. 接收、持有和发送 ETH 和 token
2. 与已部署的智能合约进行交互
3. 创建合约存在成本, 因为需要使用网络存储空间
4. 只能在收到交易时发送(内部)交易
5. 从EOA向合约帐户发起的交易能触发可执行多种操作的代码, 例如转移代币甚至创建新合约
6. 合约账户没有私钥, 它们被智能合约代码所控制



## 核心概念 - 交易

- 交易是由帐户发出, 带密码学签名的指令, 用于更新以太坊网络的状态。
- 交易将在整个网络中广播;
- 交易需要支付费用, 并且必须被包含在有效区块中才可以生效(被执行)。



## 核心概念 - 交易

- to – 接收地址(若为EOA, 交易将只是普通转账;若为合约帐户, 交易将执行合约代码)
- signature – 发送者的用私钥对交易所做数字签名。
- nonce - 为防交易重复执行, 以太坊的每个账号都记录了一个连续单调递增的 nonce值, 交易中携带nonce值用于区分同一个账号发起的不同交易。
- value – 发送人向接收人转移的以太币金额(单位:WEI)
- data(input) – 可包括任意数据的可选字段;对于要执行合约代码的交易, 此字段值应该是按ABI编码格式对要调用函数及其入参编码后的数据
- gasLimit – 交易可以消耗的最大数量的燃料单位。
- maxPriorityFeePerGas - 作为验证者小费所愿意支付的最大价格
- maxFeePerGas - 愿意为交易支付的最大gas价格(包括 baseFeePerGas 和 maxPriorityFeePerGas)



## 核心概念 - 交易

```
{  
  "nonce": "0x0",  
  "maxFeePerGas": "0x1234",  
  "maxPriorityFeePerGas": "0x1234",  
  "gas": "0x5555",  
  "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",  
  "value": "0x1234",  
  "input": "0xabcd",  
  "v": "0x26",  
  "r": "0x223a7c9bcf5531c99be5ea7082183816eb20cfe0bbc322e97cc5c7f71ab8b20e",  
  "s": "0x2aadee6b34b45bb15bc42d9c09de4a6754e7000908da72d48cc7704971491663",  
  "hash": "0xeba2df809e7a612a0a0d444ccfa5c839624bdc00dd29e3340d46df3870f8a30e"  
}
```



# 核心概念 - 交易

## 交易类型

1. 常规交易: 从一个EOA帐户到另一个EOA帐户的交易。
2. 合约部署交易: 没有“to”地址的交易, 数据字段用于合约代码。
3. 执行合约: 与已部署的智能合约进行交互的交易。在这种情况下, “to”地址是智能合约地址。





# 核心概念 - 交易

## TYPED TRANSACTION ENVELOPE 交易

### 1. 最初交易编码形式:

RLP([nonce, gasPrice, gasLimit, to, value, data, v, r, s])

### 2. [EIP-2718: 类型化交易封套](#)定义了交易类型, 是未来交易类型的“封套”。

EIP-2718:

TransactionType || TransactionPayload

- TransactionType - 一个在 0 到 0x7f 之间的数字, 总共为 128 种可能的交易类型。
- TransactionPayload - 由交易类型定义的任意字节数组。

```
// Transaction types.  
const (  
    LegacyTxType = iota  
    AccessListTxType  
    DynamicFeeTxType  
)
```



## 核心概念 - 交易收据

交易被包含在区块中之后(经过EVM执行后), 将产生一个对应的收据 receipt, 该收据记录了交易执行结果的一些信息, 包括交易是否“成功”, 消耗了多少gas等, 如果是跟智能合约相关的交易, 还可能包含合约代码内定义及记录的“事件日志”。

```

1  {
2    "blockHash": "0xc4fad2e9821ec7c7f899316d9b0cce5586792e563d4c0966a0be6b3b3e529970",
3    "blockNumber": "0xf4bfea",
4    "contractAddress": null,
5    "cumulativeGasUsed": "0x809ca9",
6    "effectiveGasPrice": "0x28aa03fa2",
7    "from": "0x0da808d51f07ab111fbbcd62c40b898d68bb4211",
8    "gasUsed": "0xb41d",
9    "logs": [
10     {
11       "address": "0xdac17f958d2ee523a2206206994597c13d831ec7",
12       "topics": [
13         "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
14         "0x000000000000000000000000da808d51f07ab111fbbcd62c40b898d68bb4211",
15         "0x000000000000000000000000f3e50c433e87f1127d3f3d9bd39ad845c898ee73"
16       ],
17       "data": "0x0000000000000000000000000000000000000000000000000000000000000000177825f00",
18       "blockNumber": "0xf4bfea",
19       "transactionHash": "0xcce32933c791636c66e5a27c1470323c3ff3315dbd7ea8134cee7907212d3f06",
20       "transactionIndex": "0x6f",
21       "blockHash": "0xc4fad2e9821ec7c7f899316d9b0cce5586792e563d4c0966a0be6b3b3e529970",
22       "logIndex": "0xb4",
23       "removed": false
24     }
25   ],
26   "logsBloom": "0x00000000000000000000000000000000000000000000000000000000000000001000000000000000010000000",
27   "status": "0x1",
28   "to": "0xdac17f958d2ee523a2206206994597c13d831ec7",
29   "transactionHash": "0xcce32933c791636c66e5a27c1470323c3ff3315dbd7ea8134cee7907212d3f06",
30   "transactionIndex": "0x6f",
31   "type": "0x2"
32 }

```



## 核心概念 - 交易费用

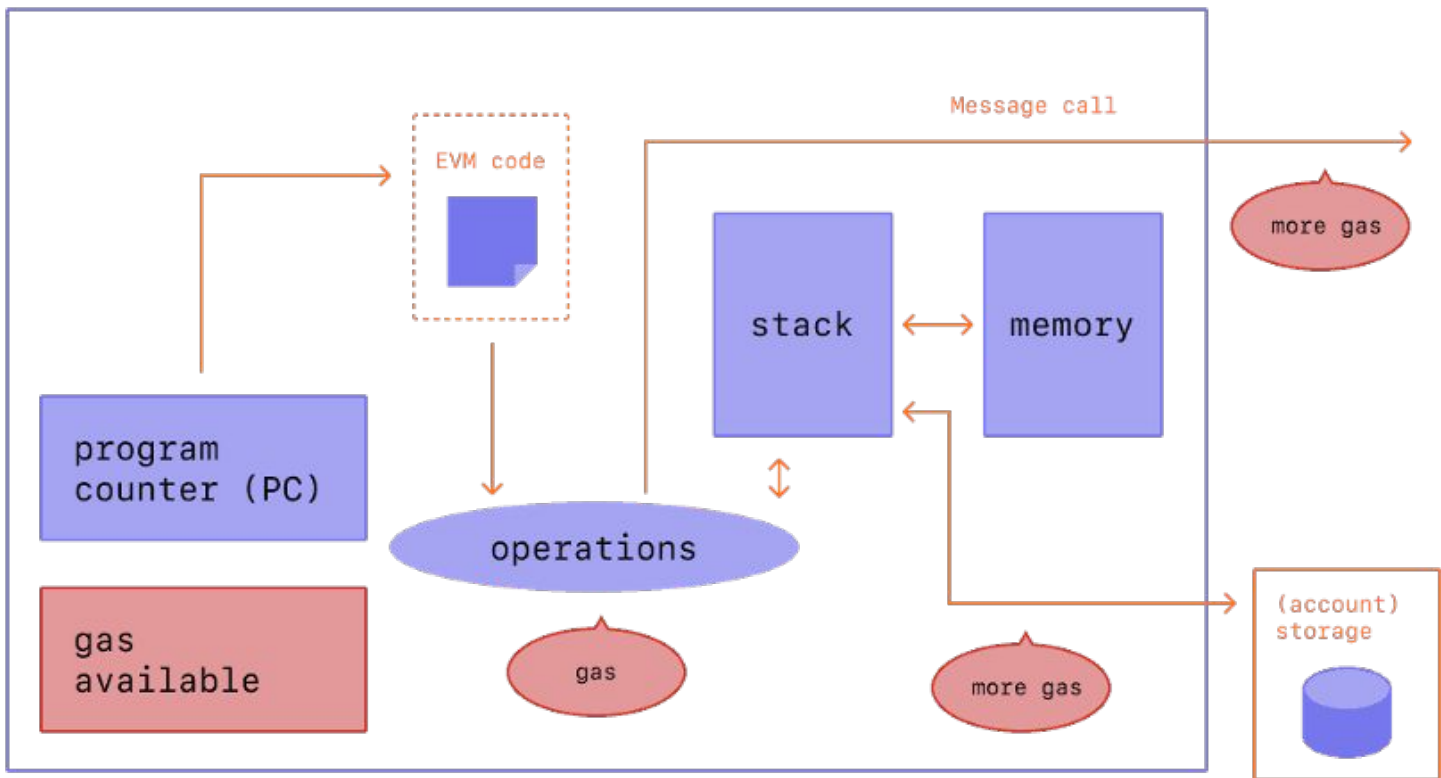
- 交易需要消耗 gas。
- Gas 是指在以太坊网络上执行特定操作所需消耗的资源计量。
- 以太坊定义了复杂的gas计量机制，使得EVM中执行的每一个操作，都尽量对应其资源消耗成本，从而最大限度预防各种DoS攻击。
- 交易中的gasLimit，设置了用户愿意为交易所消耗的最大gas数量。若是跟智能合约交互的交易并且gasLimit小于交易实际所需要消耗的gas数量，则交易会以“失败”的状态被打包进区块中。

EOA之间转账：  
21000 gas

常见ERC20转账：  
5万 gas 左右

DEX类Swap：  
可能 10万~30万 gas

## 核心概念 - 交易费用





## 核心概念 - 交易费用

**交易费用 = 交易消耗的gas \* gas价格**

- gas消耗只与交易本身业务有关；
- 通过gas定价机制，实现交易竞价执行的市场机制。

### **After EIP-1559:**

For LegacyTxType(set gasPrice):

If gasPrice >= baseFeePerGas

Then fee = gasUsed\*gasPrice;

and

tip = gasUsed\*(gasPrice -  
baseFeePerGas)

For DynamicFeeTxType(set maxFeePerGas and  
maxPriorityFeePerGas):

If maxFeePerGas >= baseFeePerGas

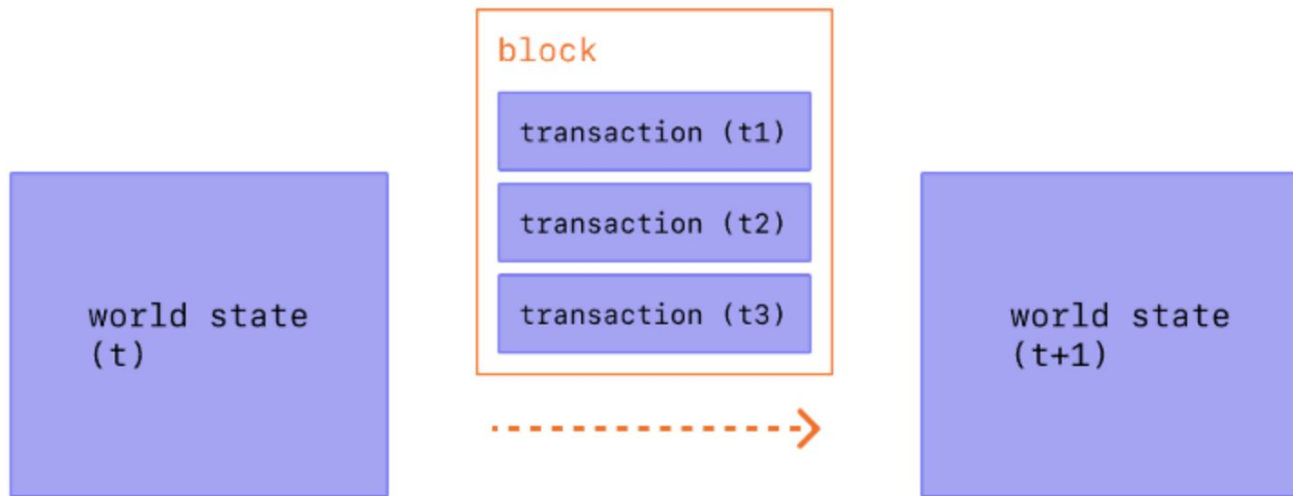
Then

tip = gasUsed \* MIN(maxFeePerGas-baseFeePerGas,  
maxPriorityFeePerGas);

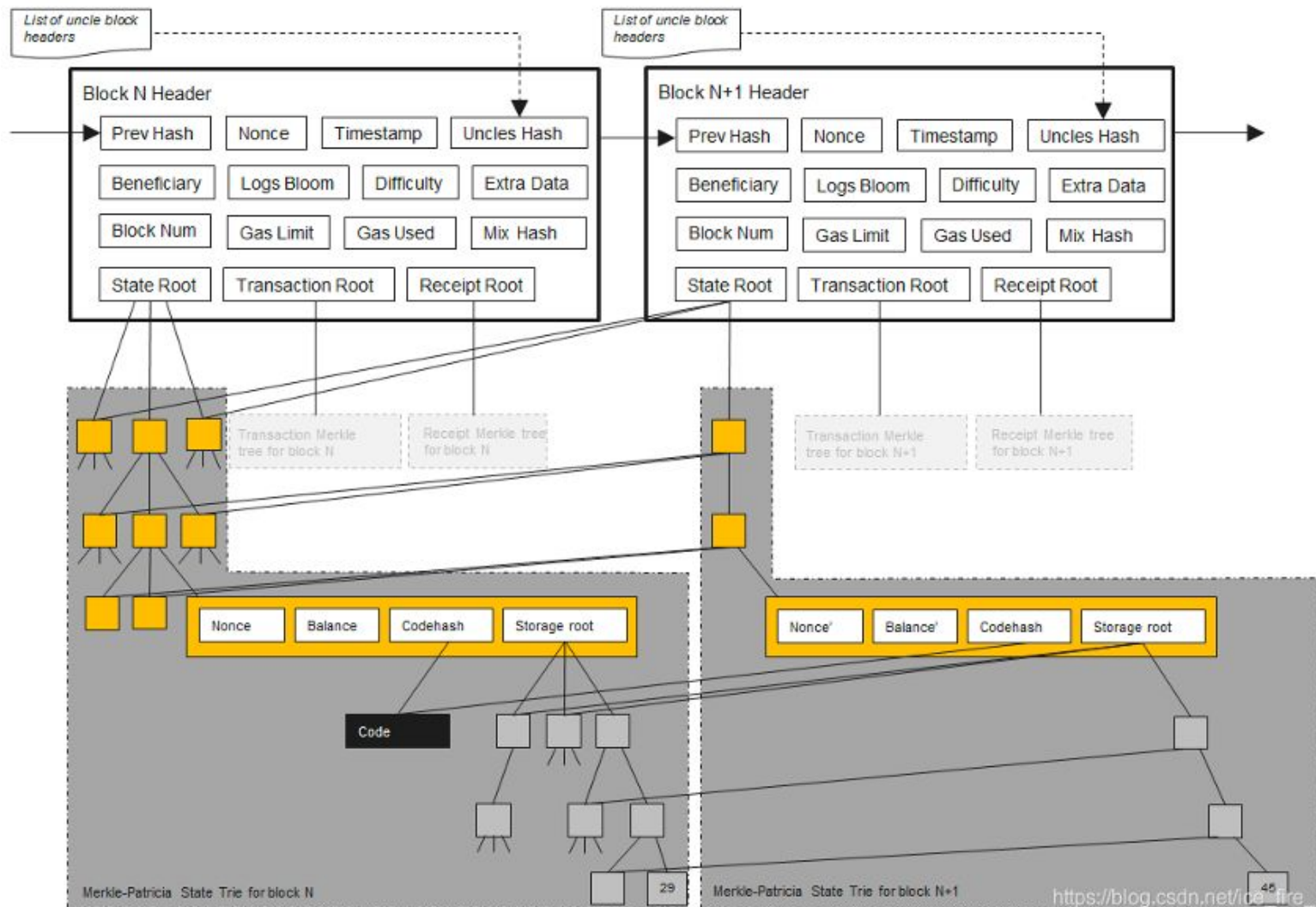
fee = tip + gasUsed \* baseFeePerGas;

## 核心概念 - 区块 Block

以太坊具有典型的区块链结构，以区块Block 为单位组织状态转移记录即交易，并在前一个区块形成的状态的基础上，通过应用当前区块的交易完成状态转移，形成当前区块的状态。因此，每一个区块都对应着一个世界状态。



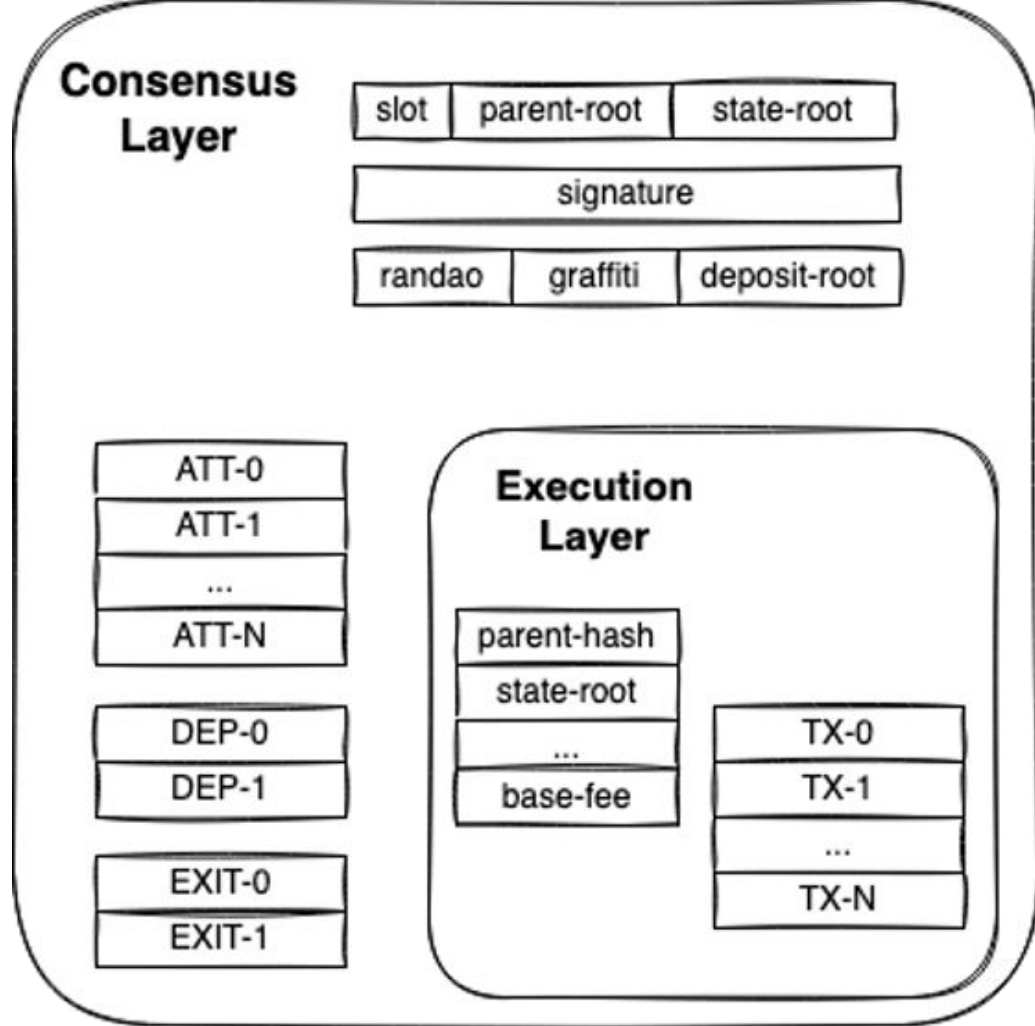
Before the merge:







After the merge:

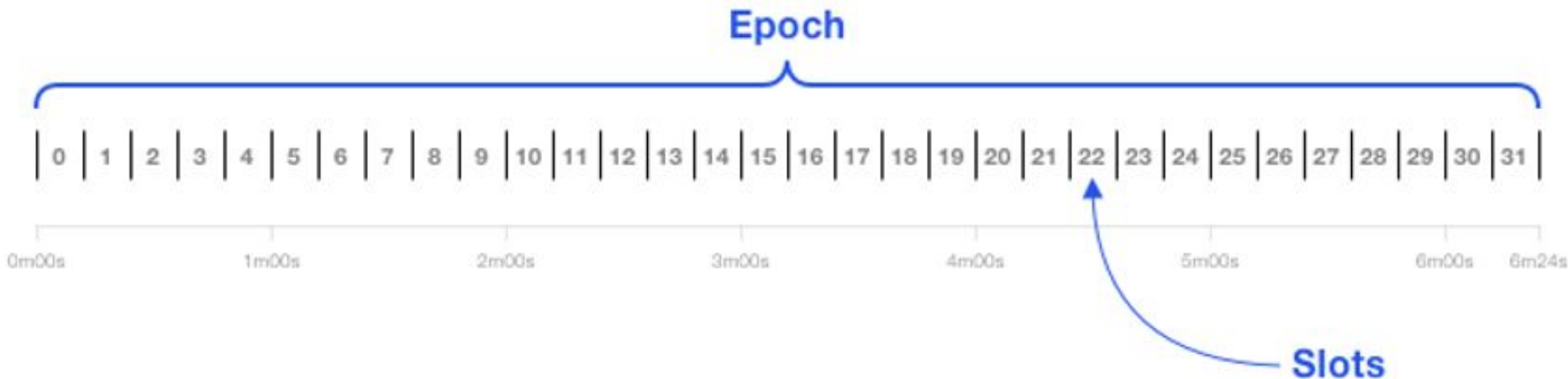




# 核心概念 - 以太坊PoS基本概念

## Slots and Epochs

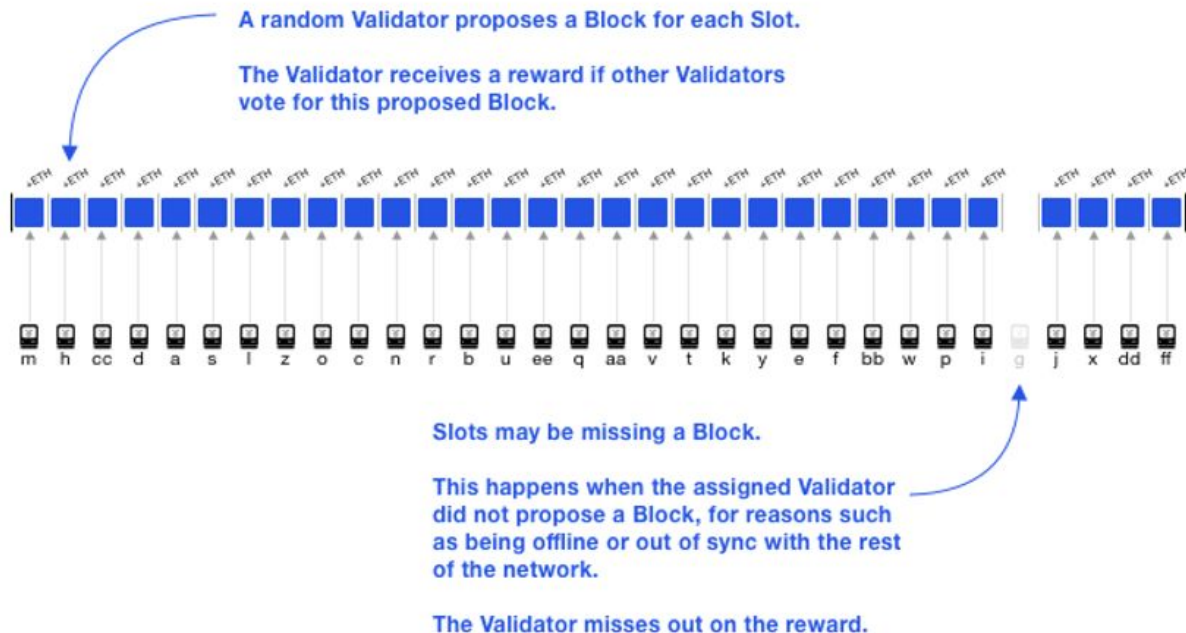
12 secs/slot, 32slots/epoch



# 核心概念 - 以太坊PoS基本概念

## Validators and Attestations

在每个slot, 会有一个 block  
proposer 提议一个区块, 然后其他 validator会对区块进行(事后)验证及投票。



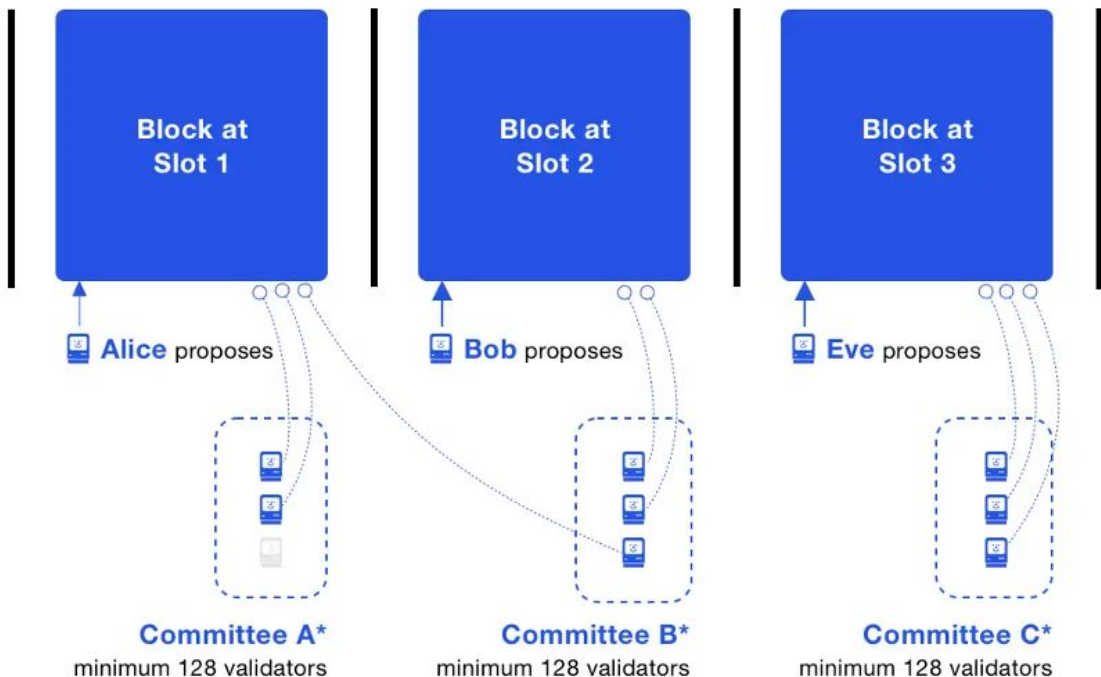
# 核心概念

## - 以太坊PoS

### 基本概念

## Committees

以太坊为每个 slot 随机选出至少 128个validator 组成的委员会，一个或多个委员会负责该slot的见证投票。



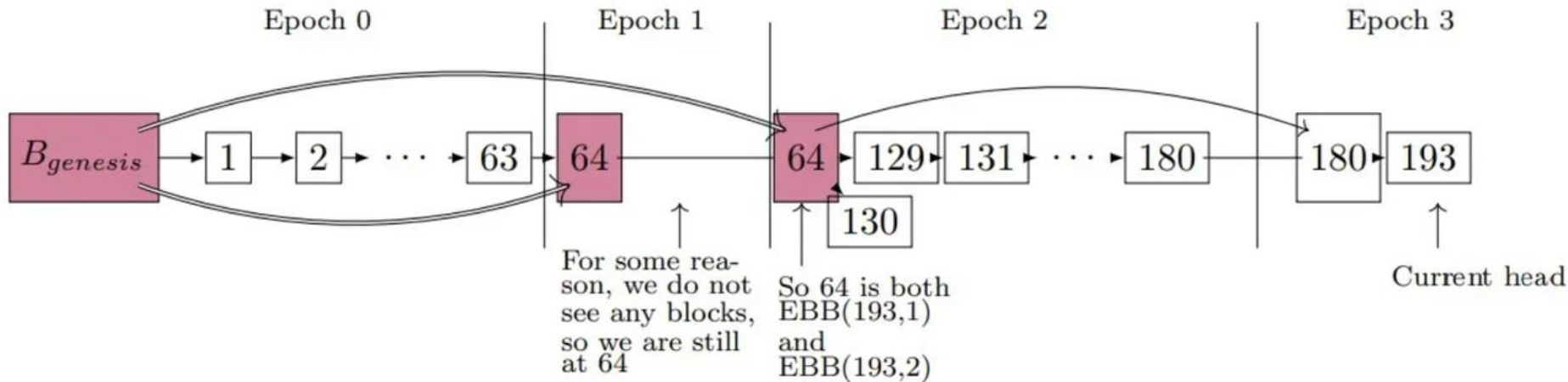
**Validators in the committees are supposed to attest to what they believe the head of the blockchain is**

*\*Note there can be more than one committee per slot.*

# 核心概念 - 以太坊PoS基本概念

## Beacon Chain Checkpoints

每个epoch的第一个 slot, 为 checkpoint。网络中的所有活跃validator, 都会按照一个叫 Casper FFG 的算法额外对检查点进行投票。



Checkpoints for a scenario where epochs contain 64 slots.



# 核心概念 - 以太坊PoS基本概念

## Finality

在区块链中, Finality是指交易上链之后达到一个(很可能是概率性的)最终确定的状态, 即我们可以安全地认为交易不会被回滚的状态。

PoW: 区块确认数

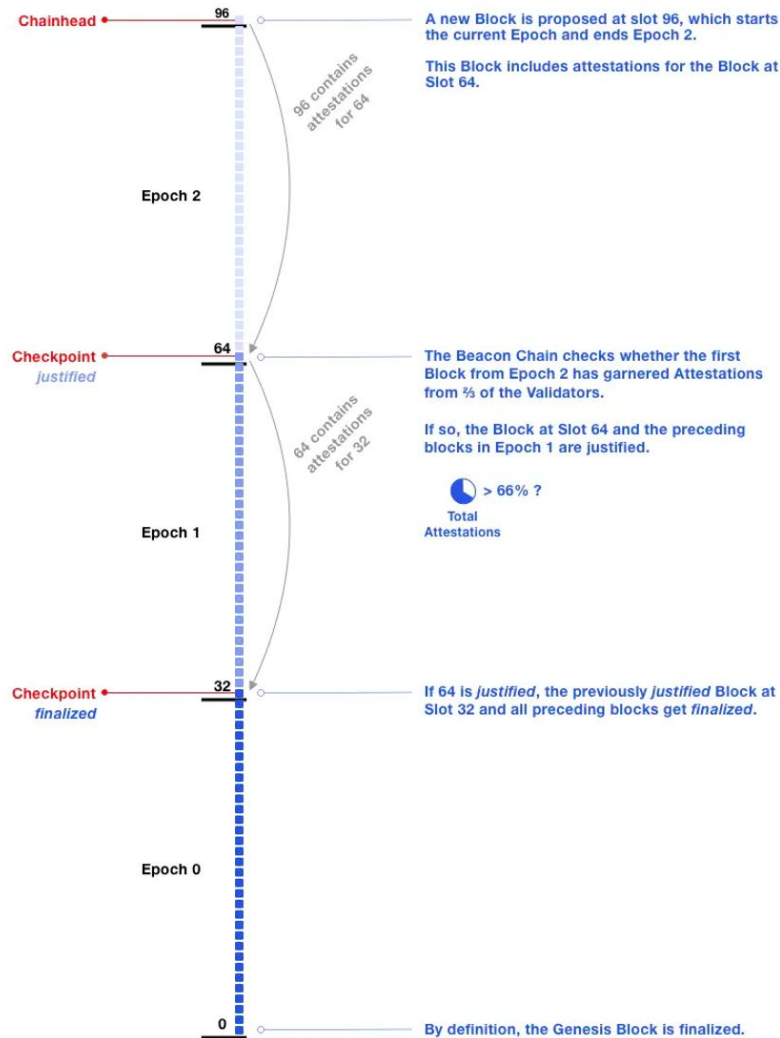
PoS+BFT: 即时确认

以太坊 LMD GHOST + Casper FFG: ??

# 核心概念 - 以太坊PoS基

## Finality in Ethereum

- 若检查点B获得2/3多数投票则变成 justified;
- 若B是 justified, 且紧随其后的检查点也变成了 justified, 此时B升级为 finalized。
- 正常情况下, 检查点会在 2个epoch之后 finalized, 大概 12.8 分钟。假设一笔交易被包含在一个 epoch 中间的区块, 则它最快需 2.5 个 epoch才能 finalized, 大概 16分钟。





# 核心概念 – 以太坊PoS基本概念

## Finality in Ethereum

理论上应该等区块所在的epoch finalized之后，才应该将交易状态最终确认。

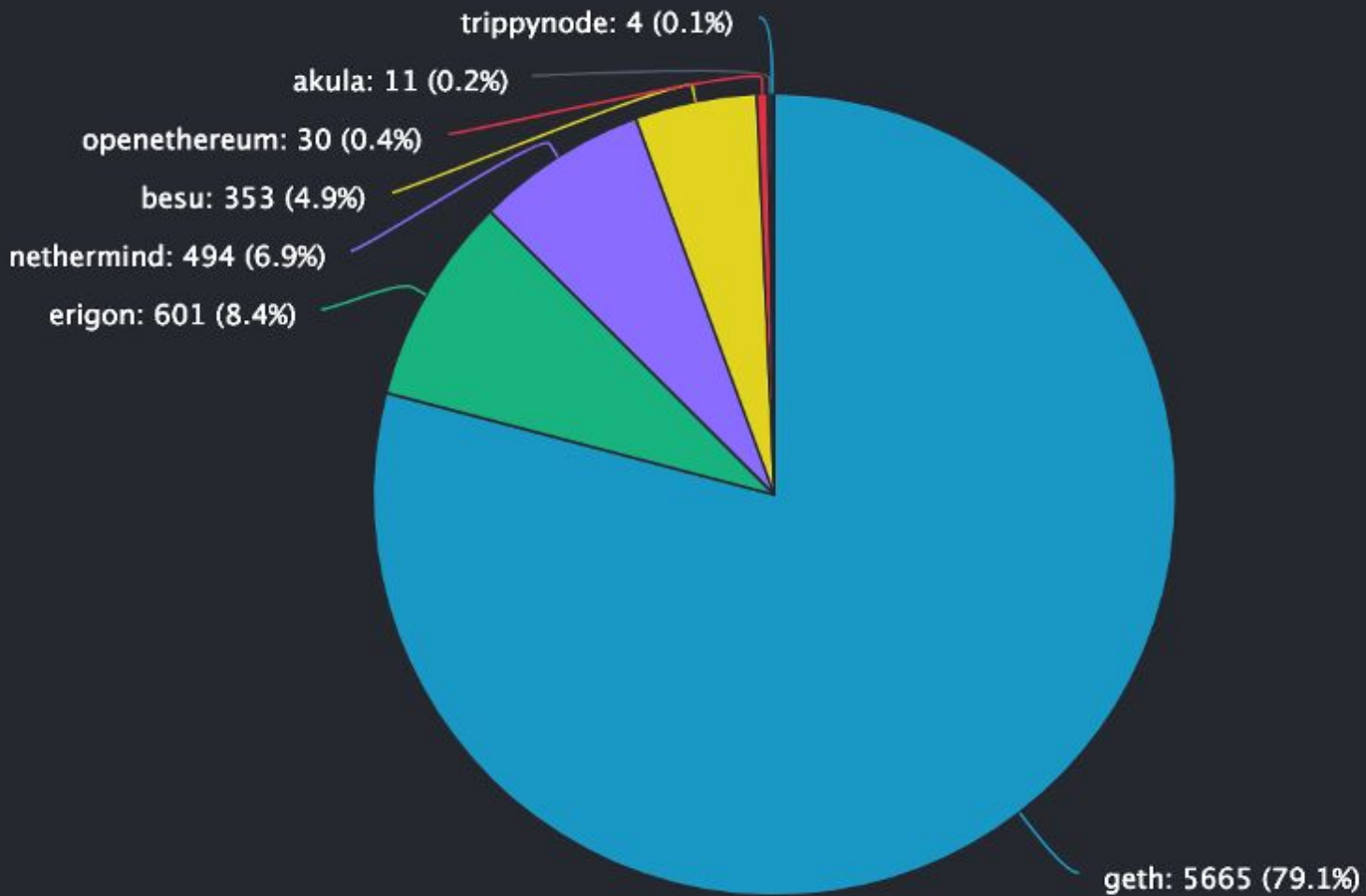
但实际上，由于不大可能存在足够强大的“作恶”势力，当前很多交易所都没有按以太坊PoS的Finality规则进行交易的最终确认，而是几乎还沿用了原来的“区块确认数”概念，  
如 Binance 还是12区块确认，OKX则是 30区块确认(12区块就到账)





# Clients and nodes – Execution clients

Client	Language	Operating systems	Networks	Sync strategies	State pruning
<a href="#">Geth</a>	Go	Linux, Windows, macOS	Mainnet, Sepolia, Görli, Ropsten, Rinkeby	Snap, Full	Archive, Pruned
<a href="#">Erigon</a>	Go	Linux, Windows, macOS	Mainnet, Sepolia, Görli, Rinkeby, Ropsten, and more	Full	Archive, Pruned
<a href="#">Nethermind</a>	C#, .NET	Linux, Windows, macOS	Mainnet, Sepolia, Görli, Ropsten, Rinkeby, and more	Snap (without serving), Fast, Full	Archive, Pruned
<a href="#">Besu</a>	Java	Linux, Windows, macOS	Mainnet, Sepolia, Görli, Ropsten, Rinkeby, and more	Fast, Full	Archive, Pruned





# Clients and nodes – Consensus clients

Client	Language	Operating systems	Networks
<a href="#">Lighthouse</a>	Rust	Linux, Windows, macOS	Beacon Chain, Goerli, Pyrmont, Sepolia, Ropsten, and more
<a href="#">Prysm</a>	Go	Linux, Windows, macOS	Beacon Chain, Gnosis, Goerli, Pyrmont, Sepolia, Ropsten, and more
<a href="#">Lodestar</a>	TypeScript	Linux, Windows, macOS	Beacon Chain, Goerli, Sepolia, Ropsten, and more
<a href="#">Nimbus</a>	Nim	Linux, Windows, macOS	Beacon Chain, Goerli, Sepolia, Ropsten, and more
<a href="#">Teku</a>	Java	Linux, Windows, macOS	Beacon Chain, Gnosis, Goerli, Sepolia, Ropsten, and more



## Clients and nodes – Consensus clients

Prysm - 41.86%



Lighthouse - 36.84%



Teku - 17.73%



Nimbus - 3.33%



Lodestar - 0.24%



Uncertain - 0%



Other - 0%



Data provided by [Sigma Prime's Blockprint](#) — updated daily.

Data may not be 100% accurate. ([Read more](#))

**Data source ([read more](#)):**



Sigma Prime's Blockprint



Miga Labs



# Clients and nodes

## – Execution node types

- **Full node** 全节点: 保存所有的区块、交易、收据数据 + 最新若干(128个)区块的完整世界状态。是最常见的节点类型, 可以完成绝大部分场景需求。
- **Light node** 轻节点: 只验证和保存所有的区块头数据。从网络中的其他节点查询相应数据以及相应的默克尔证明, 然后基于本地区块头上的状态树根 验证数据的正确性。
- **Archive node** 归档节点: full node data + all state tree。archive节点可以查询历史上任何区块高度上的状态, 可以跟踪任何区块高度上的交易 (以便分析交易执行过程, debug)。



# Clients and nodes

## – Execution node sync mode

- **Full sync** 全同步: 从网络其他节点同步所有的区块及交易数据, 然后本地重放每一个区块并验证每一个区块。最安全, 最慢。
- **Fast sync** 快速同步: 同步区块头数据并依据共识规则验证区块头, 若无误则认为区块合法, 然后单独同步区块体(交易、收据)并根据区块头中的交易哈希 和 收据哈希 进行验证, 无误则保存。然后, 在接近最新区块时, 同步世界状态树, 状态树同步完成后切换成 full sync模式。快, 安全性稍弱。曾经的主流。



# Clients and nodes

## – Execution node sync mode

- **Snap sync** 快照同步: 这里的snap 指snapshot, 而这里的snapshot是指以太坊大概在2021年成熟应用的一种扁平化状态存储结构, 以这种结构缓存世界态, 可以大大提高对状态的读速度。而snap sync就是基于这种snapshot结构, 整体上类似于快速同步, 只是在同步世界状态树的时候, 基于snapshot结构进行同步, 然后本地重建状态树。这种同步方式, 要求网络中有大量的节点建好了snapshot结构。当前, snap sync 取代 fast sync称为大部分全节点的选择。
- **Light sync** 轻同步: 用于轻节点, 只会同步并验证区块头, 不同步区块体 (交易、收据), 也没有最新的完整的世界状态树。



# 以太坊客户端实操

<https://github.com/0xcoolface/ethernode-operation>





# Geth交互接口

## 接口传输协议

- http
- Websocket
- 进程内通信管道 IPC



# Geth交互接口

## 接口传输协议

- JSON-RPC
- JavaScript Console 交互终端
- 各语言对JSON-RPC接口的封装,  
如JavaScript API
- GraphQL



# 链上交互核心流程

<https://github.com/0xcoolface/backend-dapp-demo>

- 构造交易
- 估算gas
- 签名及广播交易
- 主动查询交易结果 || 订阅机制, 被动实时接受通知 [geth pubsub](#)
- 进阶: 交易失败问题排查

[https://geth.ethereum.org/docs/rpc/ns-debug#debug\\_traceTransaction](https://geth.ethereum.org/docs/rpc/ns-debug#debug_traceTransaction)

<https://geth.ethereum.org/docs/evm-tracing/builtin-tracers>

谢谢！

