

# 总结与展望

路远

中国科学院软件研究所

# 本讲内容

- **回顾**

- 密码学基础
- 数据结构
- P2P网络
- 共识机制
- 智能合约
- 比特币、以太坊、超级账本

- **展望**

- 挑战与机遇

# 密码学基础

# 可忽略函数

计算安全性：计算资源受限的攻击“不可行”，  
即攻击成功的概率  $< \text{negl}(L)$ ，  
这里  $\text{negl}(L)$  代表  $L$  的**可忽略函数** (negligible function)

- **$L$  的可忽略函数：**

对于  $L$  的任意多项式函数  $\text{poly}(L)$  一定存在某个  $L_n$ ，如果  $L > L_n$ ，则：  
$$|\text{negl}(L)| < 1 / \text{poly}(L)$$

- **另一个等价的定义：**

对于所有的正整数  $c$  一定存在某个  $L_n$ ，如果  $L > L_n$ ，则：  
$$|\text{negl}(L)| < 1 / L^c$$

- **典型的可忽略函数：**

$$f(L) = 1 / 2^L$$

# 密码学哈希函数

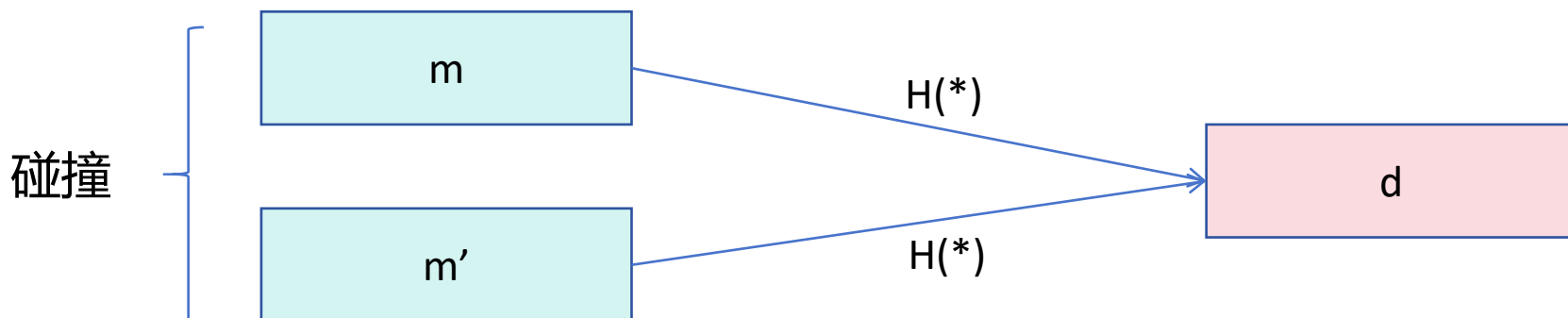
- 函数  $H: \{0,1\}^* \rightarrow \{0,1\}^K$



- $H$ 是**密码学哈希函数**（或者叫做**杂凑函数**），如果具有：

- 抗碰撞性** collision-resistance

攻击者在多项式时间内，找到两个不相同的  $m$  和  $m'$  使得  $H(m)=H(m')$  是**不可行的**



# 密码学哈希函数

- SHA256

- 用于比特币的工作量证明
- 也用于生成比特币地址

- RIPEMD160

- 主要用于生成比特币地址：  
RIPEMD160(SHA256(K))

- Keccak256

- 主要用于生成以太坊地址  
Keccak256(K)[-160:]

- SM3 (杂凑函数算法国家标准)

- 安全性及效率与SHA-256相当
- 多用于国产区块链系统

区块链常用哈希算法对比

算法名称	输入长度 (比特)	分组长度 (比特)	基本字长 (比特)	输出长度 (比特)
SHA-256	$< 2^{64}$	512	32	256
RIPEMD-160	$< 2^{64}$	512	32	160
Keccak-256	不限	1088	64	256
SM3	$< 2^{64}$	512	32	256

# 密码学哈希函数

给定密码学哈希函数H，以下哪些函数F也是密码学哈希函数？

$$F(x) = H(x) || 0000$$

$$F(x) = H(H(x))$$

$$F(x) = H(x) || H(x)$$

$$F(x) = H(0000 || x)$$

$$F(x) = H(x) \oplus H(x)$$

$$F(x) = H(x) \oplus H(\sim x)$$

# 数字签名

一个数字签名机制，通常包含 (Gen, Sign, Verify) 三个算法：

- $(pk, sk) \leftarrow \text{Gen}(1^\lambda)$ : **密钥生成算法**。输入安全参数的一元表示  $1^\lambda$ ，输出一对密钥，其中  $pk$  称为公钥或验证密钥， $sk$  称为私钥或签名密钥。
- $\sigma \leftarrow \text{Sign}(sk, m)$ : **签名算法**。输入私钥  $sk$  和消息空间  $\mathcal{M}$  中的消息  $m$ ，输出签名  $\sigma$ ，表示为  $\sigma = \text{Sign}_{sk}(m)$ 。
- $b \leftarrow \text{Vrfy}(pk, m, \sigma)$ : **签名验证算法**。输入公钥  $pk$ ，消息  $m$  和签名  $\sigma$ ，输出比特  $b$ ，取0表示签名无效，取1表示签名有效，表示为  $b = \text{Vrfy}_{pk}(m, \sigma)$ 。

比特币和以太坊都使用基于secp256k1椭圆曲线的的 ECDSA 数字签名算法，私钥长度 256 比特



# 秘密分享与拉格朗日差值

**拉格朗日差值** (以点1和点3为例) :

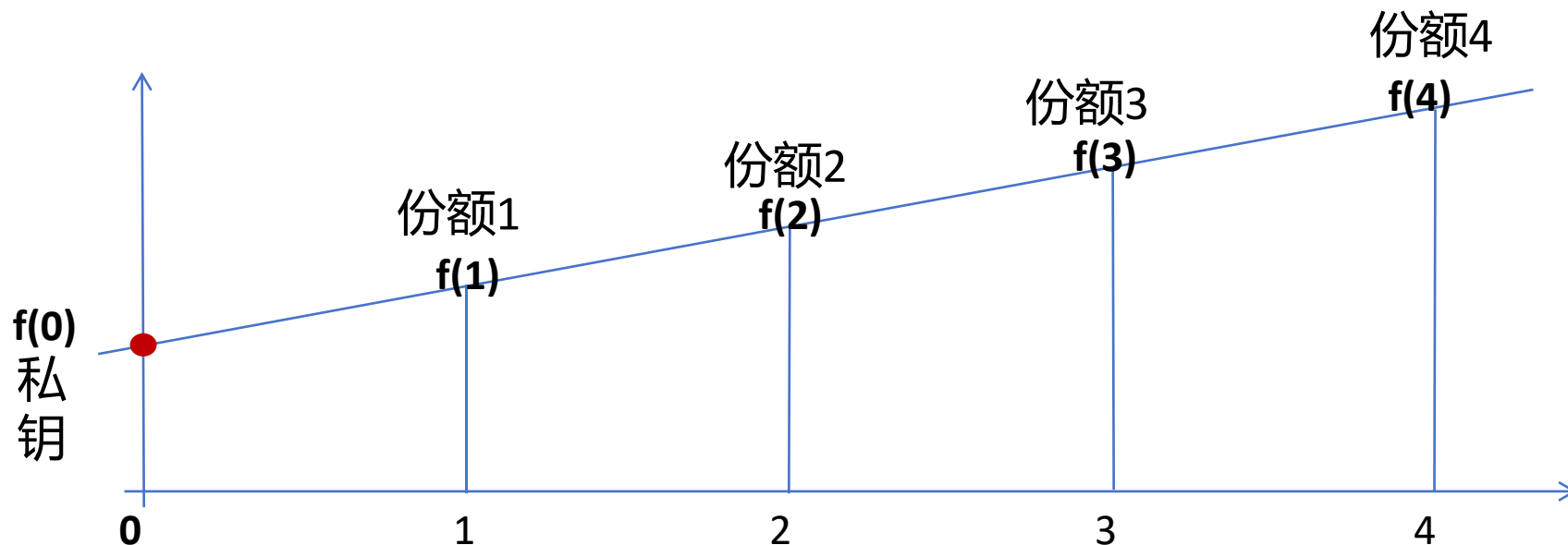
观察:

$$f_1(x) = (x-x_3)/(x_1-x_3) \Rightarrow \text{在 } x_1 \text{ 为 } 1, \text{ 在 } x_3 \text{ 为 } 0$$

$$f_3(x) = (x-x_1)/(x_3-x_1) \Rightarrow \text{在 } x_3 \text{ 为 } 1, \text{ 在 } x_1 \text{ 为 } 0$$

$f(x)$  一定可以写做  $f_1$  和  $f_3$  的线性组合地形式:

$$f(x) = f(1)*f_1(x) + f(3)*f_3(x)$$



# 助记词

- **助记词 (密钥恢复语 Secret Recovery Phrase),**

pass

original

rally

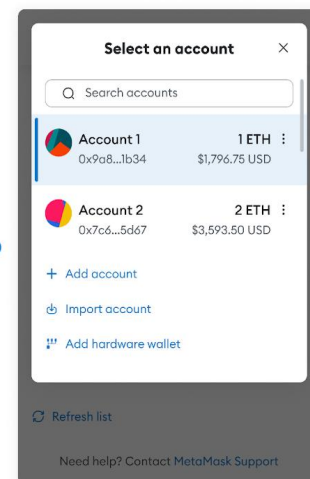
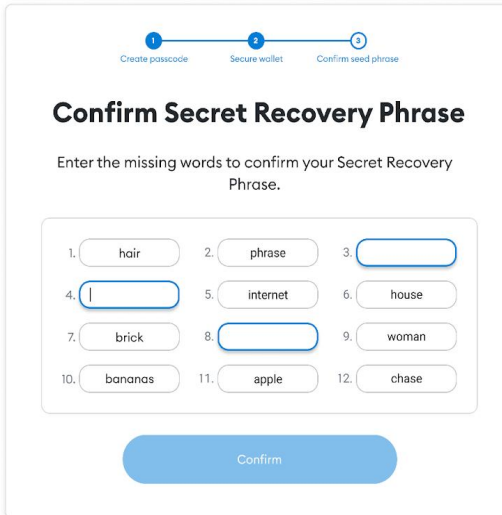
buzz

input

sustain

.....

- 在metamask钱包中，用户的私钥从2048个短语中随机选取的12个助记词导出

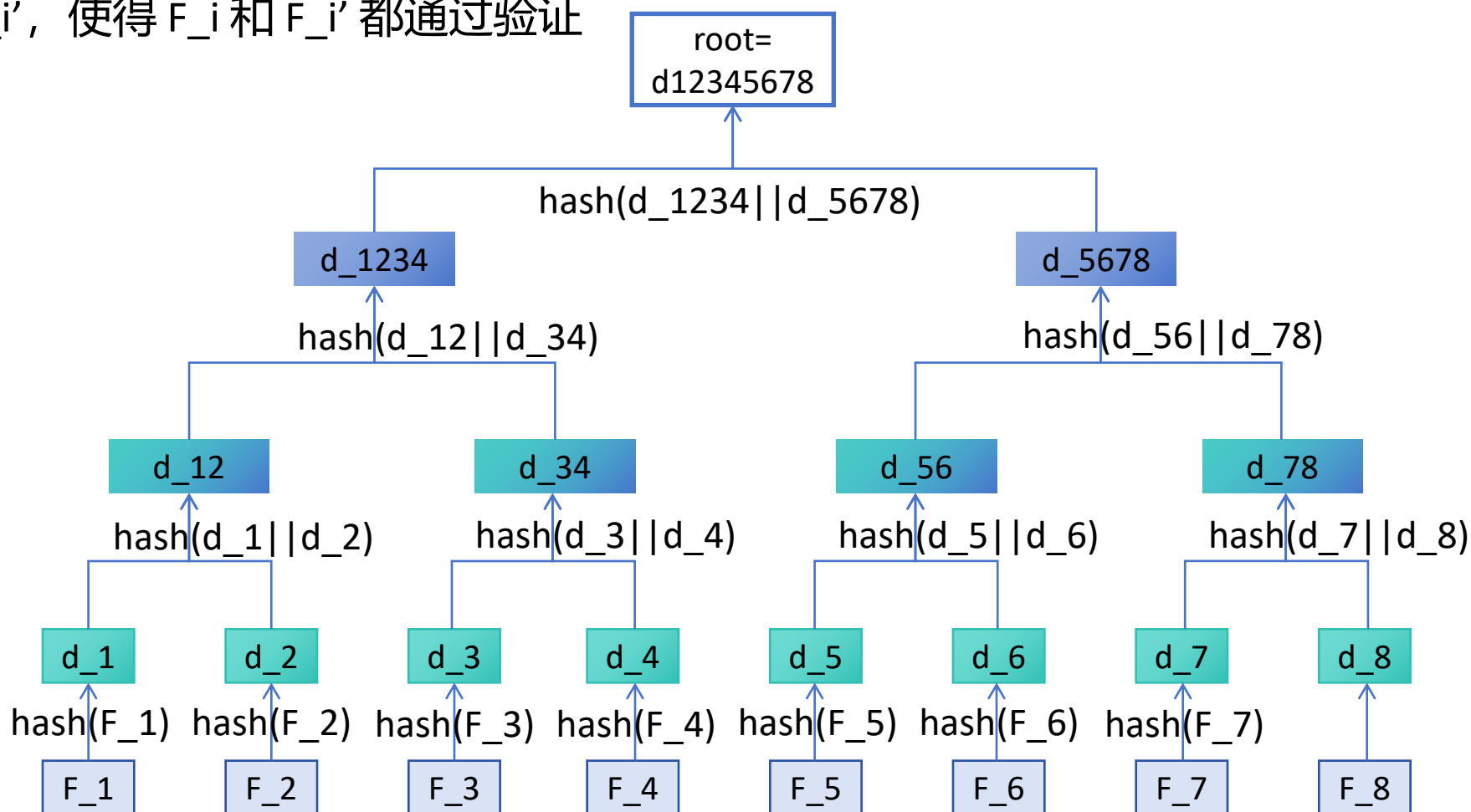


- 导出密钥的方式成为**基于口令的密钥导出函数**  
Password-Based Key Derivation Function 2

# 数据结构

# 默克尔树

**Merkle Tree 的安全性：位置绑定性**（向量承诺）—— 对一个承诺了N个位置的Merkle tree root, 对每个位置i, 攻击者难以可行地找到不同的  $F_i$  和  $F_i'$ , 使得  $F_i$  和  $F_i'$  都通过验证



# 布隆过滤器

以太坊的 Bloom filter:

- 2048个比特的数组 (注意  $2048 = 2^{11}$ )
- 哈希函数需要 11 比特, 不是直接使用 Keccak256, 而是使用了:
  - $H_1$  = Keccak256 的最低双字节的最低11位  
=> 得到第一个 0-2047 之间的index
  - $H_2$  = Keccak256 的次最低双字节的最低11位  
=> 得到第一个 0-2047 之间的index
  - $H_3$  = Keccak256 的次次最低双字节的最低11位  
=> 得到第一个 0-2047 之间的index
- 错误率:
  - 只有false positive(错误接受一个不在集合中的元素), 不存在 false negative
  - $m=2048, k=3$ :
    - $n = 100 \rightarrow \sim 0,25\%$
    - $n = 200 \rightarrow \sim 1,64\%$

# 比特币区块结构

## Nakamoto 作为设计者在 创世区块 (Genesis) 里做了什么?

01000000 - version

[illegible]

3BA3EDFD7A7B12B27AC72C3E67768F617FC81BC3888A51323A9FB8AA4B1E5E4A - merkle root

## 29AB5F49 - timestamp

FFFF001D - difficulty (0x00ffff00)

1DAC2B7C - nonce

### 01 - number of transactions

- 只写了一笔 无法使用的 交易!
- 除了给自己 0 BTC, 还给大家留了一段话:  
“The Times 03/Jan/2009 Chancellor on  
brink of second bailout for banks”

## 2009年1月3日，财政大臣正处于实施第二轮 银行紧急援助的边缘

(《泰晤士报》当天的头条新闻标题)



# 比特币区块结构

## 以比特币区块头

- 每个Bitcoin区块头中有6个字段的元数据
- 一个pre hash、五个字段的其他元数据

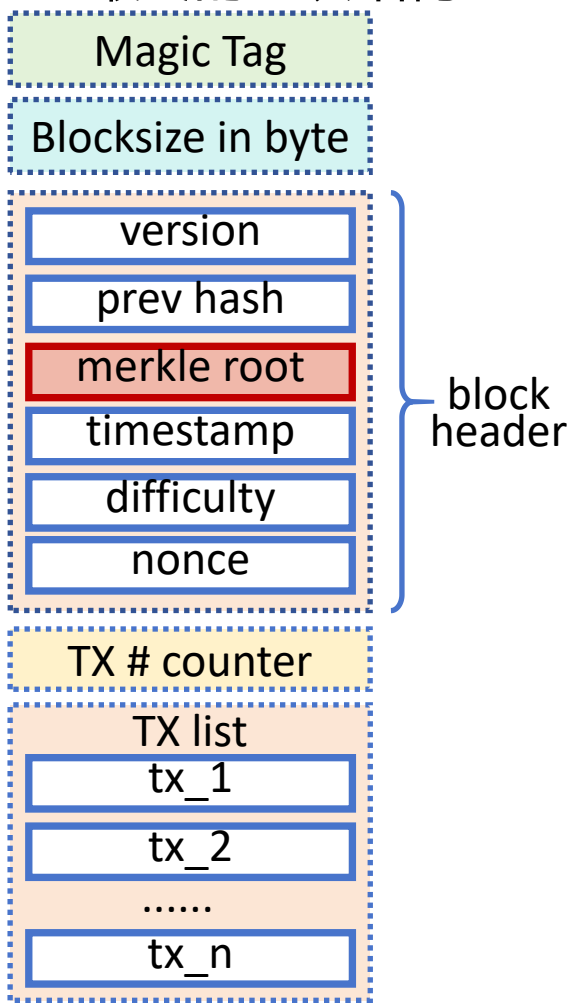


Version	区块的版本号，和当前使用的软件版本有关	4字节
Pre_hash	前一个区块头的256比特hash值	32字节
Merkle_root	本区块中所有交易构造的默克尔树的树根hash值	32字节
Timestamp	UNIX格式时间戳	4字节
Difficulty	压缩形式的PoW目标难度	4字节
Nonce	32比特的数字，计算PoW时使用	4字节

# 比特币区块结构

## 区块

- 最终的区块结构



Magic Tag	在比特币的主网中，永远设置为0xD9B4BEF9，标记为区块链主网的区块数据	4字节
Block size	本区块还剩余的字节数量	4字节
Block header	Version、Pre_hash、Merkle_root、Timestamp、Difficulty、Nonce等 6个项目	80字节
TX # counter	可变长度的正整数，表示区块中交易的数量	1-9 字节
TX list	本区块中交易的列表	-

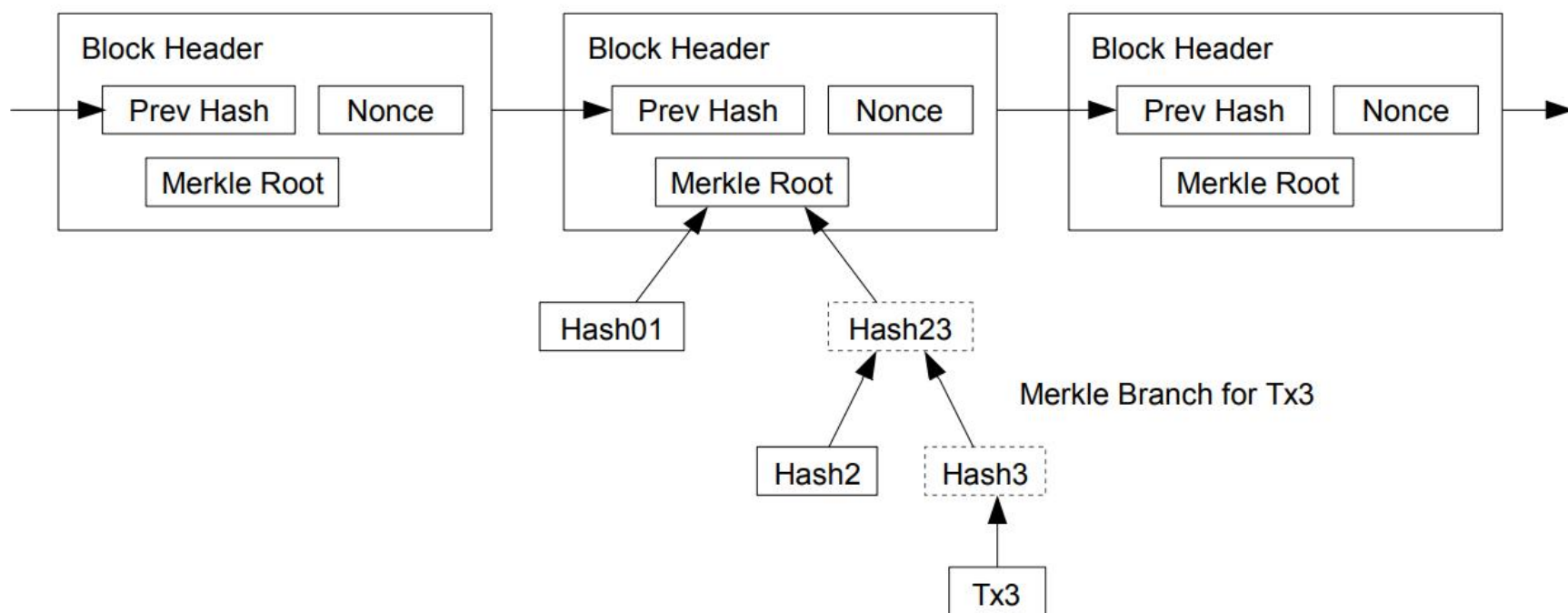


# 比特币 SPV

无法下载和存储所有区块的轻量节点，如何验证某笔交易存在于区块链：

- 正确的区块头 + 有效的 Merkle tree proof

Longest Proof-of-Work Chain



# 比特币生成交易

每个区块的第一笔交易都是特殊的，称为**生成交易**（coinbase transaction），奖励给该区块的生成者指定的比特币地址(或公钥)！

区块111111：  
50个BTC



coinbase  
的奖励变少了？

区块222222：  
25个BTC

区块444444：  
12.5个BTC

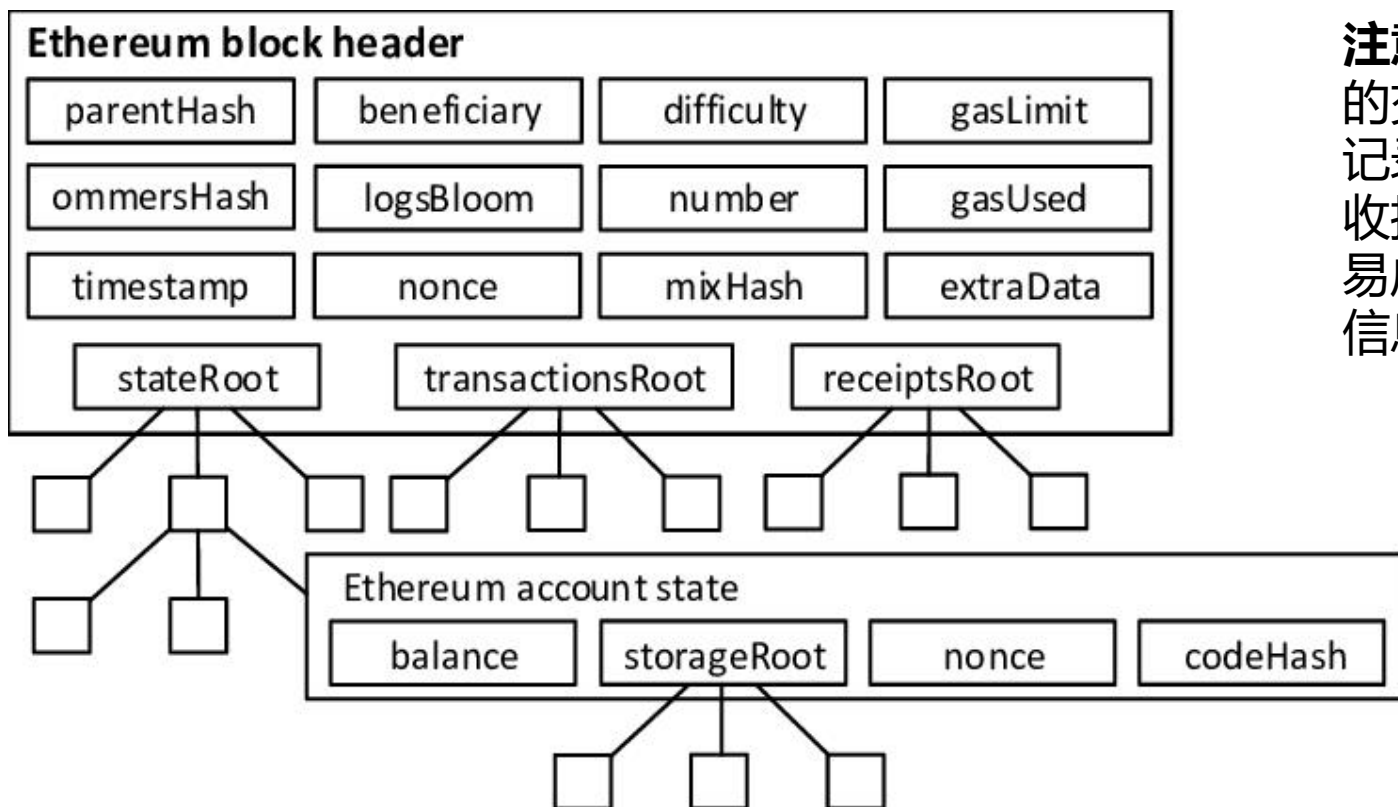
区块666666：  
6.25个BTC

**Nakamoto的规定：**  
每当比特币区块链产生  
210,000个区块时，区块  
奖励将减半。  
总量最多2100个BTC。

# 以太坊的区块结构 (23年前的PoW版)

PoW时代的以太坊区块结构，和比特币的几大不同：

- 三个哈希树：分别承诺全局的**账户状态**、本区块的**交易**、本区块的交易**收据**(日志)
- 布隆过滤器：提供区块中日志事件的快速搜索方法
- 叔块指针：包含某些分叉的哈希值



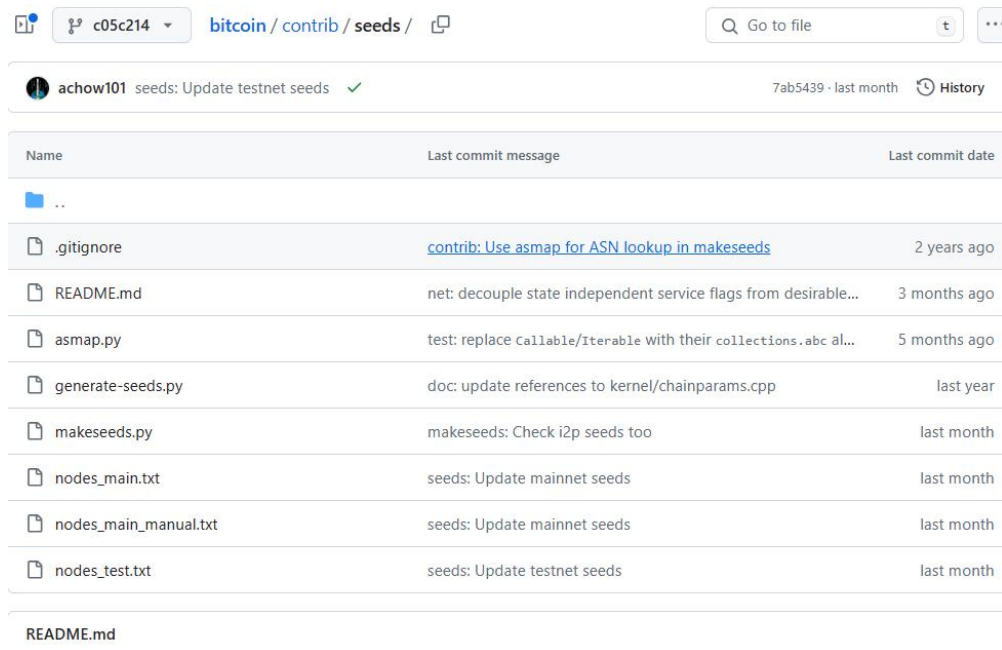
**注意：**执行错误的交易也可能会记录在交易树，收据树会记录交易成功或失败的信息

# P2P网络

# 节点入网与发现

节点第一次启动时：

- 先尝试查询**种子DNS**服务 (种子DNS网址的域名地址写入在比特币软件中)
- 等待**种子DNS**返回目前网络中仍然允许 incoming 连接的节点**IP地址列表**
- 向 IP 列表中的节点**查询更多活跃节点的IP地址**
- 更新活跃节点的IP地址
- 随机连接足够数量的活跃节点



The screenshot shows the GitHub repository page for 'bitcoin/contrib/seeds'. At the top, there's a search bar and a commit history section for user 'achow101' with the commit message 'seeds: Update testnet seeds'. Below this is a table with columns 'Name', 'Last commit message', and 'Last commit date'. The table lists several files: '..', '.gitignore', 'README.md', 'asmap.py', 'generate-seeds.py', 'makeseeds.py', 'nodes\_main.txt', 'nodes\_main\_manual.txt', and 'nodes\_test.txt'. Each file has a corresponding commit message and date. Below the table, there's a section for 'README.md'.

Name	Last commit message	Last commit date
..		
.gitignore	<a href="#">contrib: Use asmap for ASN lookup in makeseeds</a>	2 years ago
README.md	net: decouple state independent service flags from desirable...	3 months ago
asmap.py	test: replace Callable/Iterable with their collections.abc al...	5 months ago
generate-seeds.py	doc: update references to kernel/chainparams.cpp	last year
makeseeds.py	makeseeds: Check i2p seeds too	last month
nodes_main.txt	seeds: Update mainnet seeds	last month
nodes_main_manual.txt	seeds: Update mainnet seeds	last month
nodes_test.txt	seeds: Update testnet seeds	last month

README.md

## Seeds

Utility to generate the seeds.txt list that is compiled into the client (see [src/chainparamsseeds.h](#) and other utilities in [contrib/seeds](#)).

Be sure to update `PATTERN_AGENT` in `makeseeds.py` to include the current version, and remove old versions as necessary (at a minimum when `SeedsServiceFlags()` changes its default return value, as those are the services which seeds are added to addman with).

The seeds compiled into the release are created from sipa's DNS seed and AS map data. Run the following commands from the `/contrib/seeds` directory:

```
curl https://bitcoin.sipa.be/seeds.txt.gz | gzip -dc > seeds_main.txt
curl https://bitcoin.sipa.be/asmap-filled.dat > asmap-filled.dat
python3 makeseeds.py -a asmap-filled.dat -s seeds_main.txt > nodes_main.txt
cat nodes_main_manual.txt >> nodes_main.txt
python3 generate-seeds.py . > ../../src/chainparamsseeds.h
```

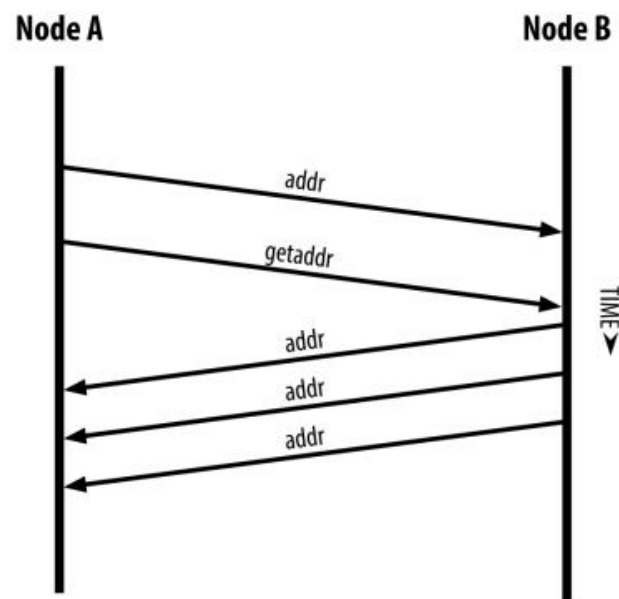
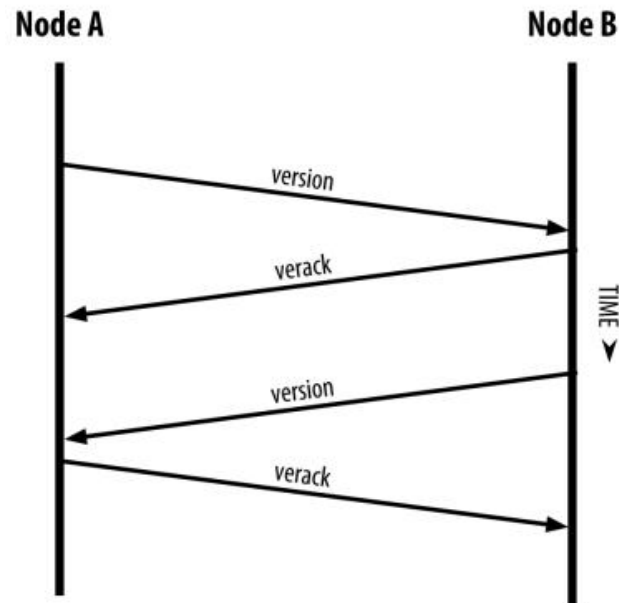
# 节点入网与发现

- **outbound连接建立**

- 启动节点发送 version 消息给种子
- DNS 返回的 IP 地址
- 被连接节点返回 verack, 也发送 version 消息
- 返回 verack 消息
- 默认允许11个outbound连接  
(曾经默认允许8个outbound连接)

- **节点发现**

- 连接完成后, 发送 addr 消息 (包含自己接受 inbound 连接的 IP 和端口等参数)
- 再发送 getaddr 消息给被连接节点, 查询活跃节点的 inbound 连接参数
- 被连接节点返回addr



# 区块下载

- 初始区块下载 (Initial Block Download, IBD)

## 区块优先策略

Message	From→To	Payload
<a href="#">“getblocks”</a>	IBD→Sync	One or more header hashes
<a href="#">“inv”</a>	Sync→IBD	Up to 500 block inventories (unique identifiers)
<a href="#">“getdata”</a>	IBD→Sync	One or more block inventories
<a href="#">“block”</a>	Sync→IBD	One serialized block

## 区块头优先策略

Message	From→To	Payload
<a href="#">“getheaders”</a>	IBD→Sync	One or more header hashes
<a href="#">“headers”</a>	Sync→IBD	Up to 2,000 block headers
<a href="#">“getdata”</a>	IBD→Many	One or more block inventories derived from header hashes
<a href="#">“block”</a>	Many→IBD	One serialized block

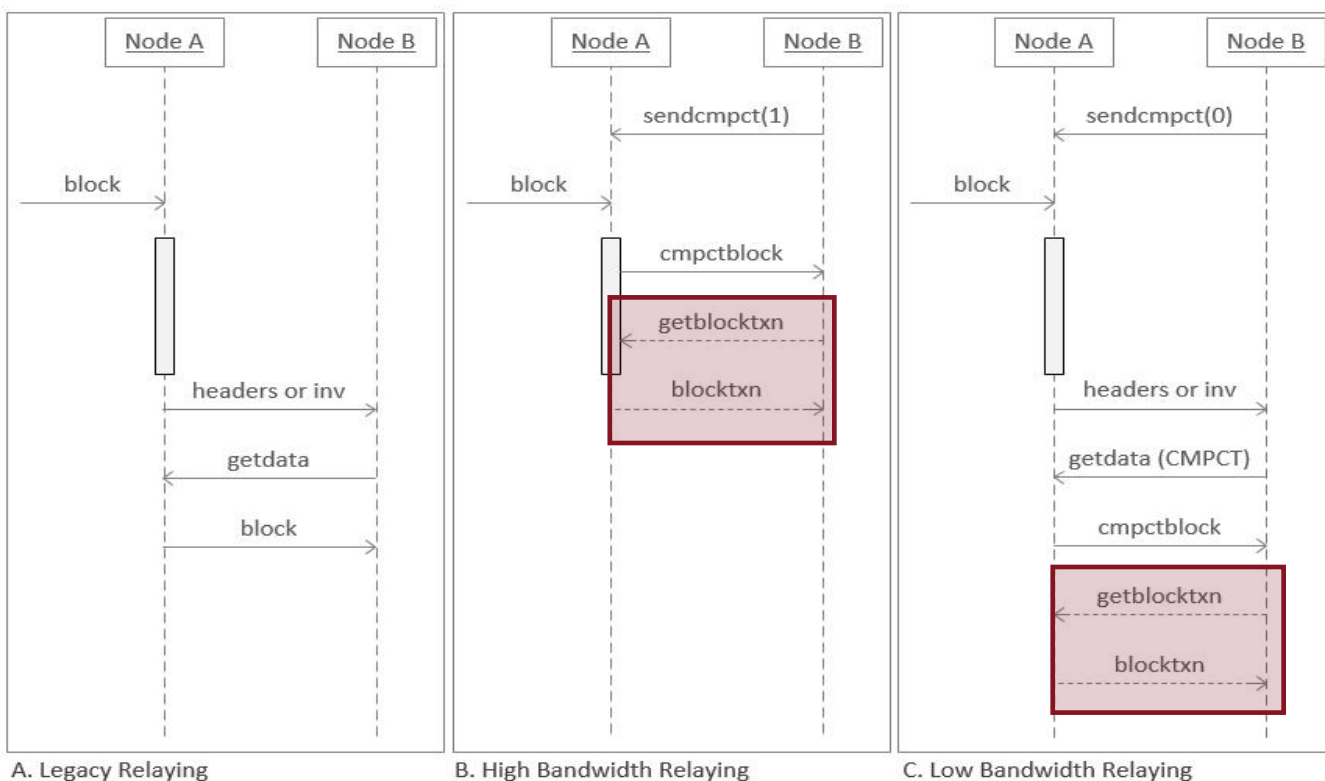
# 区块扩散

- 新区块的发现者/中继节点，有四种可能的推送区块的方式：
  - a) 使用 block 消息直接推送完整的新区块
  - b) 使用 inv 消息先推送新区块的哈希值，再等待：
    - b.1) 来自区块优先节点的 getdata 消息，推送 block 消息
    - b.2) 来自块头优先节点的 getheaders 消息，推送 headers 消息
  - c) 向块头优先节点直接发送 headers 消息 (跳过 inv 消息)  
**注意：**在握手阶段，块头优先节点会通过 sendheaders 指明自己的区块下载策略。
  - d) 压缩区块中继 (Compact Block Relay) BIP-152



# 区块扩散

- 新区块的发现者/中继节点，有四种可能的推送区块的方式：
  - d) 压缩区块中继 (Compact Block Relay) BIP-152  
交易的下载是可选 (只下载交易池中没收到过的交易)



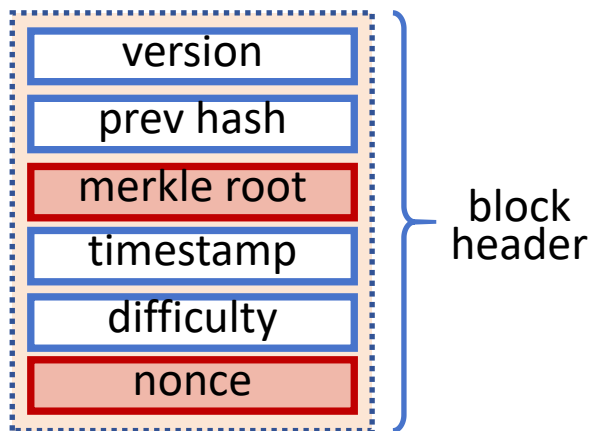
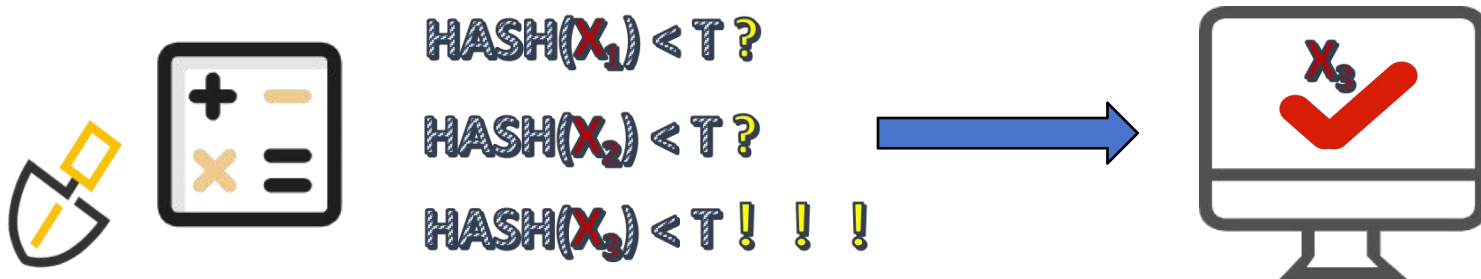
# 共识机制

# PoW谜题

**POW**: 创建一个区块不是无代价的, 需要寻找一个区块头、使其哈希值小于难度目标:

$\text{Hash}(\text{block\_header}) < \text{目标难度target}$ , 即,

$\text{SHA256}(\text{SHA256}(\text{Version} || \text{Pre\_hash} || \text{Merkle\_root} || \text{Timestamp} || \text{Difficulty} || \text{Nonce})) < \text{target}$

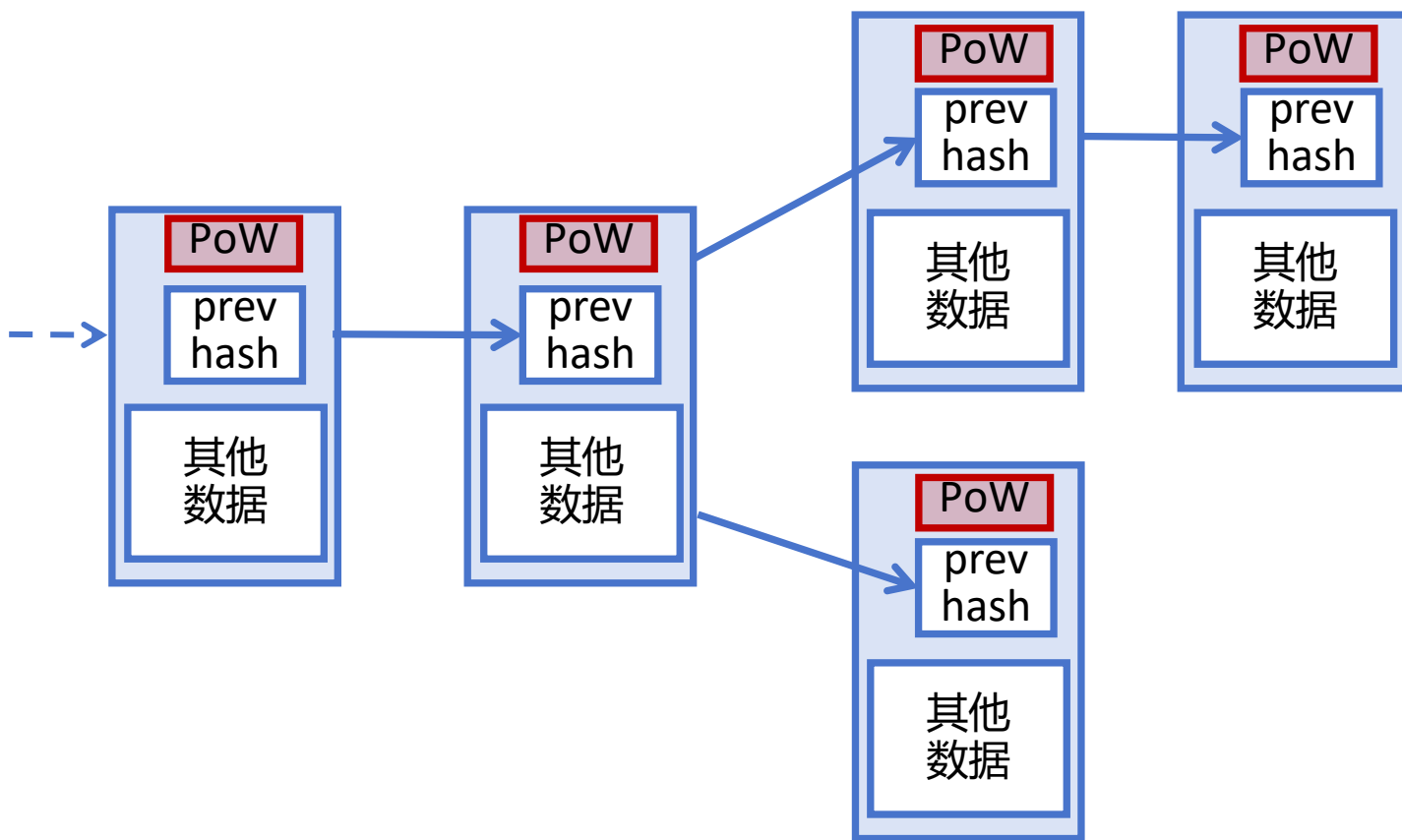


- version, prev\_hash, difficulty 是固定的
- timestamp 能够变化的位数有限 (需在当前时间附近)
- nonce 有32比特的变化空间
- merkle\_root 可以通过修改 coinbase 交易变化, 有256比特的变化空间
- target 是由 difficulty 算出

# 最长链规则

最长链规则 (longest chain rule):

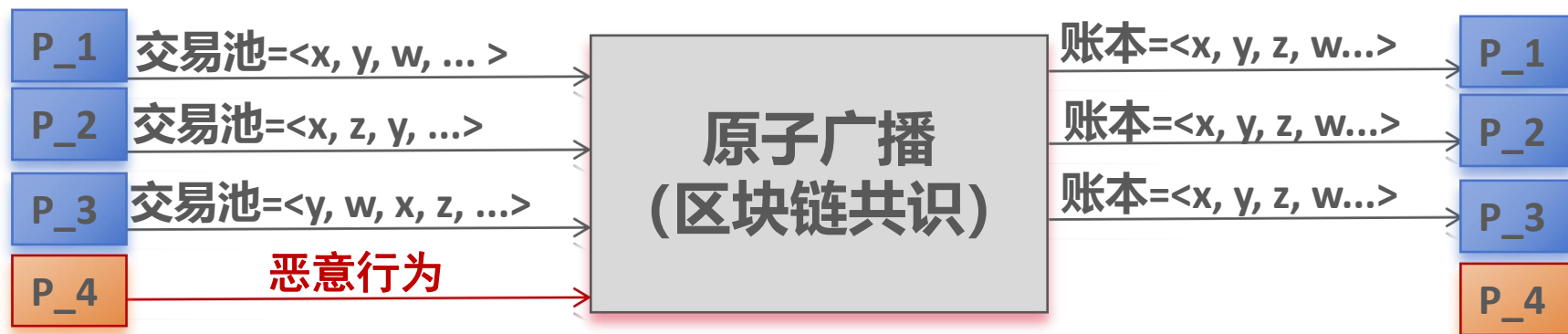
诚实节点的“挖矿规则”：当遇到分叉时，选择更长 (累计工作难度更多) 的分支开始寻找满足难度规则的区块



# 共识机制的安全性质

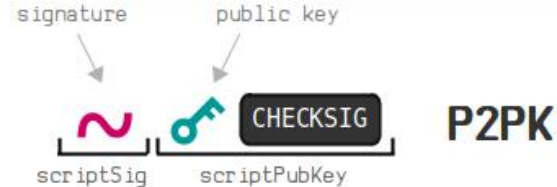
■ 区块链需要一类持续运行的共识协议，满足：

- 一致性 Safety/Consistency  
=> 保证全网各节点的“交易日志”最终一致
- 活性 Liveness  
=> 保证用户提交的交易最终被“交易日志”接受

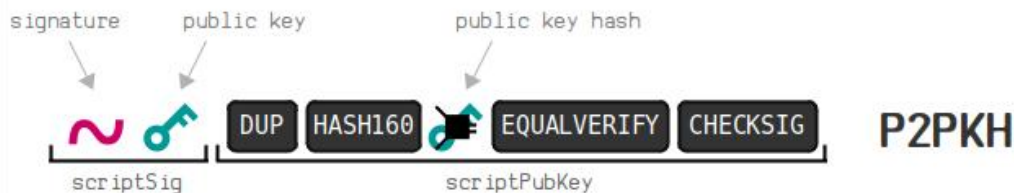


# 智能合约

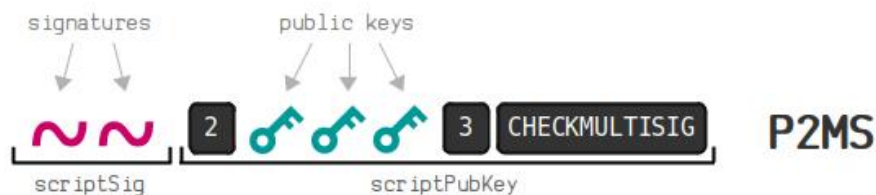
# 比特币经典脚本



P2PK



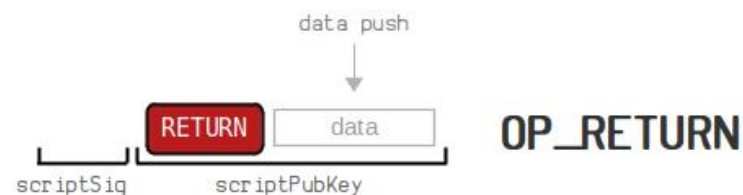
P2PKH



P2MS



P2SH



OP\_RETURN

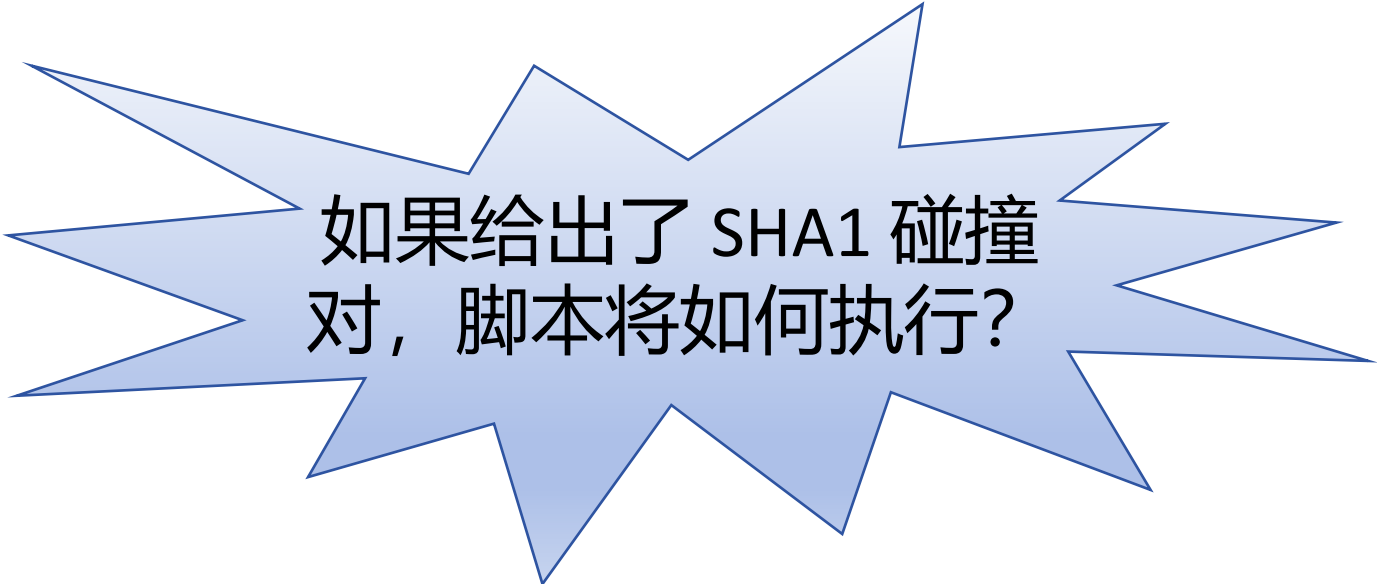
- P2PK (Pay To Public Key)
- P2PKH (Pay To Pubkey Hash)
- P2MS (Pay To Multisig)
- P2SH (Pay To Script Hash)
- OP\_RETURN

# 一些比特币脚本举例

- 一个奖励 SHA1 碰撞对的输出脚本:

```
OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP  
OP_SHA1 OP_EQUAL
```

有效的输入脚本 => 给出 SHA1 碰撞对



如果给出了 SHA1 碰撞对，脚本将如何执行？

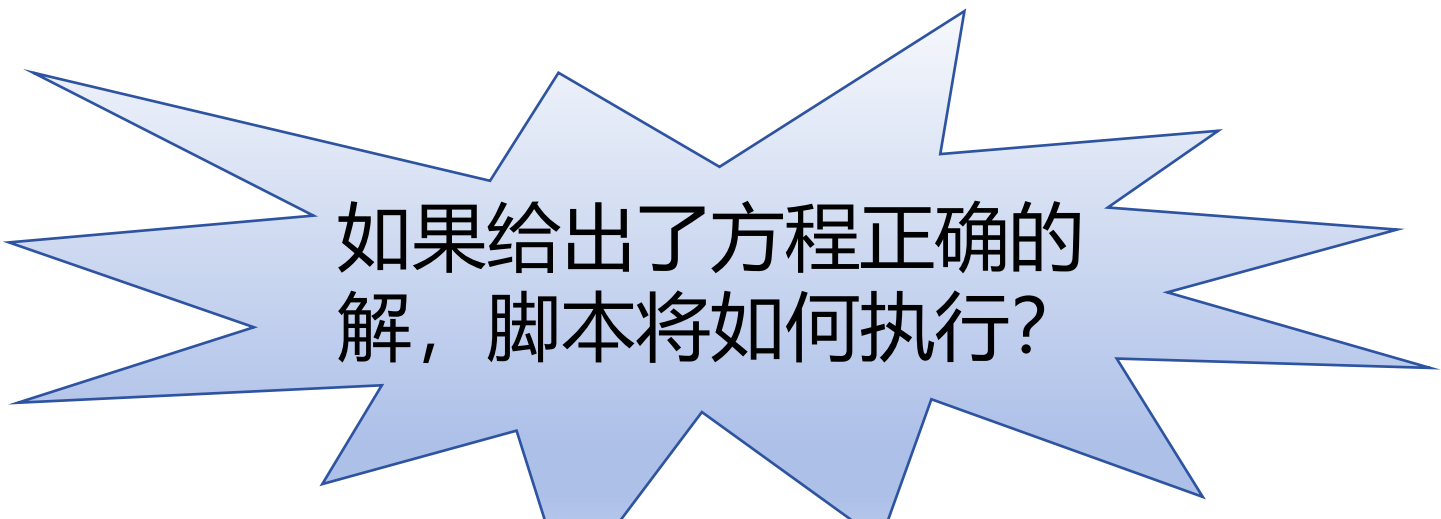


# 一些比特币脚本举例

- 一个奖励 线性方程  $3x + 7 = 13$  根的输出脚本:

OP\_DUP OP\_DUP 7 OP\_ADD OP\_ADD OP\_ADD 13 OP\_EQUAL

有效的输入脚本 => 给出 方程  $3x + 7 = 13$  的根  $x=2$



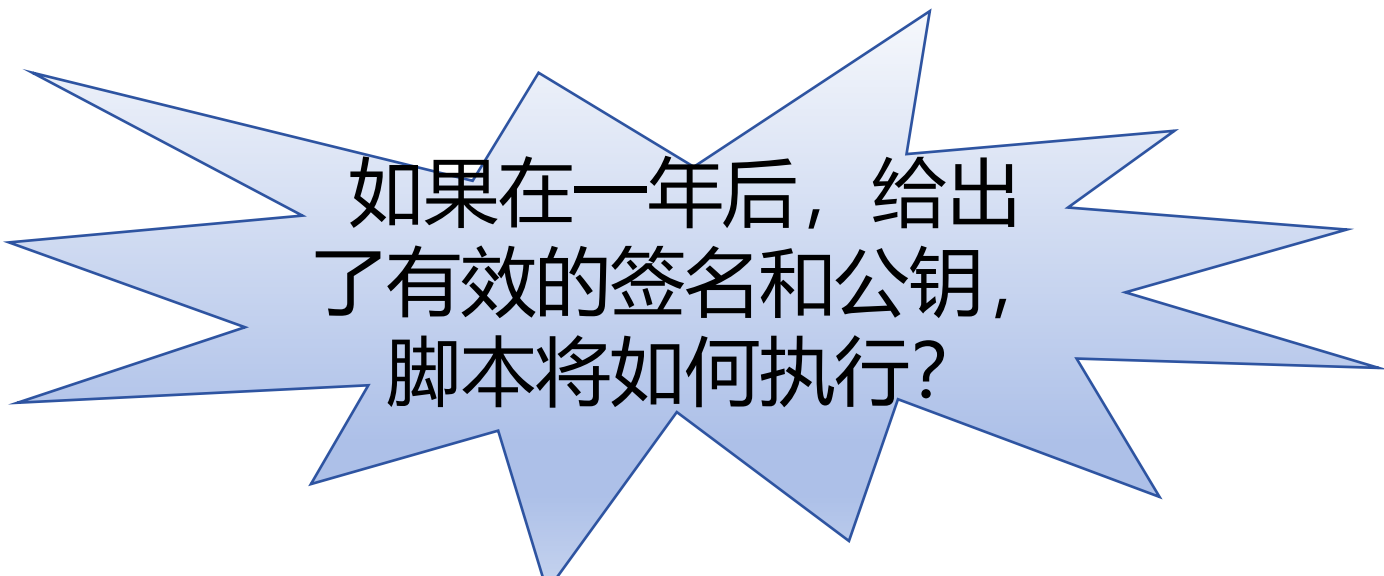
如果给出了方程正确的解，脚本将如何执行？

# 一些比特币脚本举例

- 第二年才可以花费的P2PKH输出脚本:

```
<NextYear> OP_CHECKLOCKTIMEVERIFY OP_DROP  
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

有效的输入脚本: <signature> <pubKey>



如果在一年后，给出了有效的签名和公钥，脚本将如何执行？

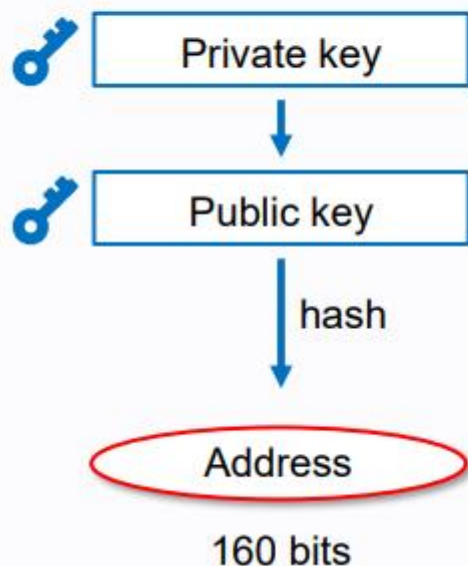
# 以太坊账户

- 以太坊有两类账户

- **外部拥有账户** (普通账户) **EOA**: 由用户的私钥控制, 状态包括余额信息等
- **智能合约账户 CA**: 地址由创建者的普通账户地址导出, 包含可执行代码、变量状态、余额等信息

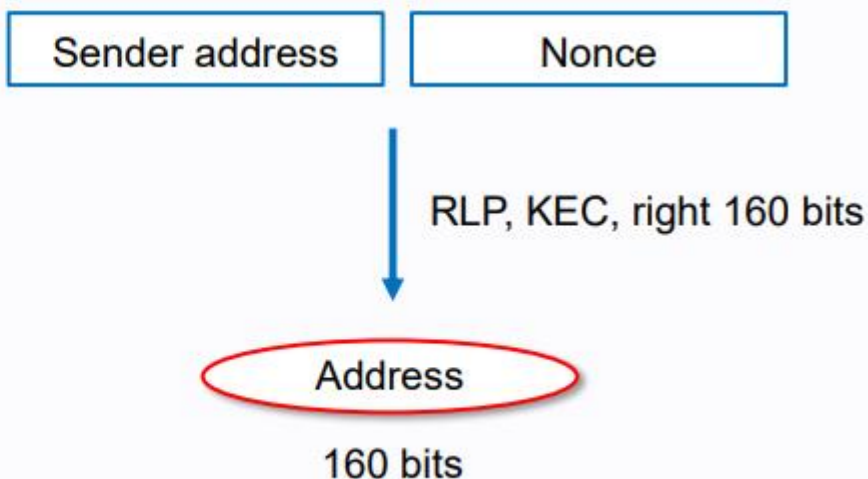
Externally owned account (EOA)

外部拥有账户地址由密钥导出



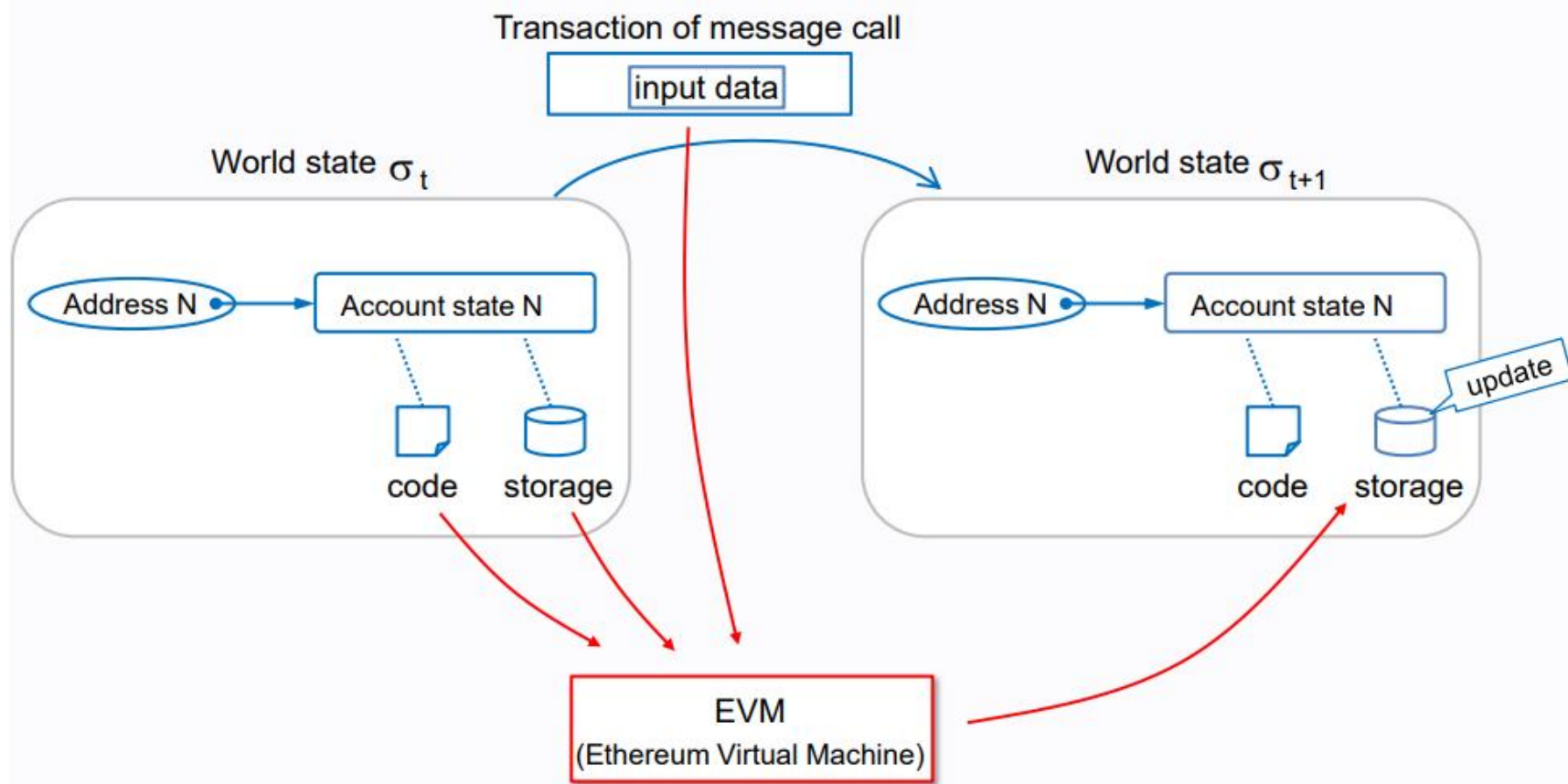
Contract account

合约地址由创建者的普通账户地址导出



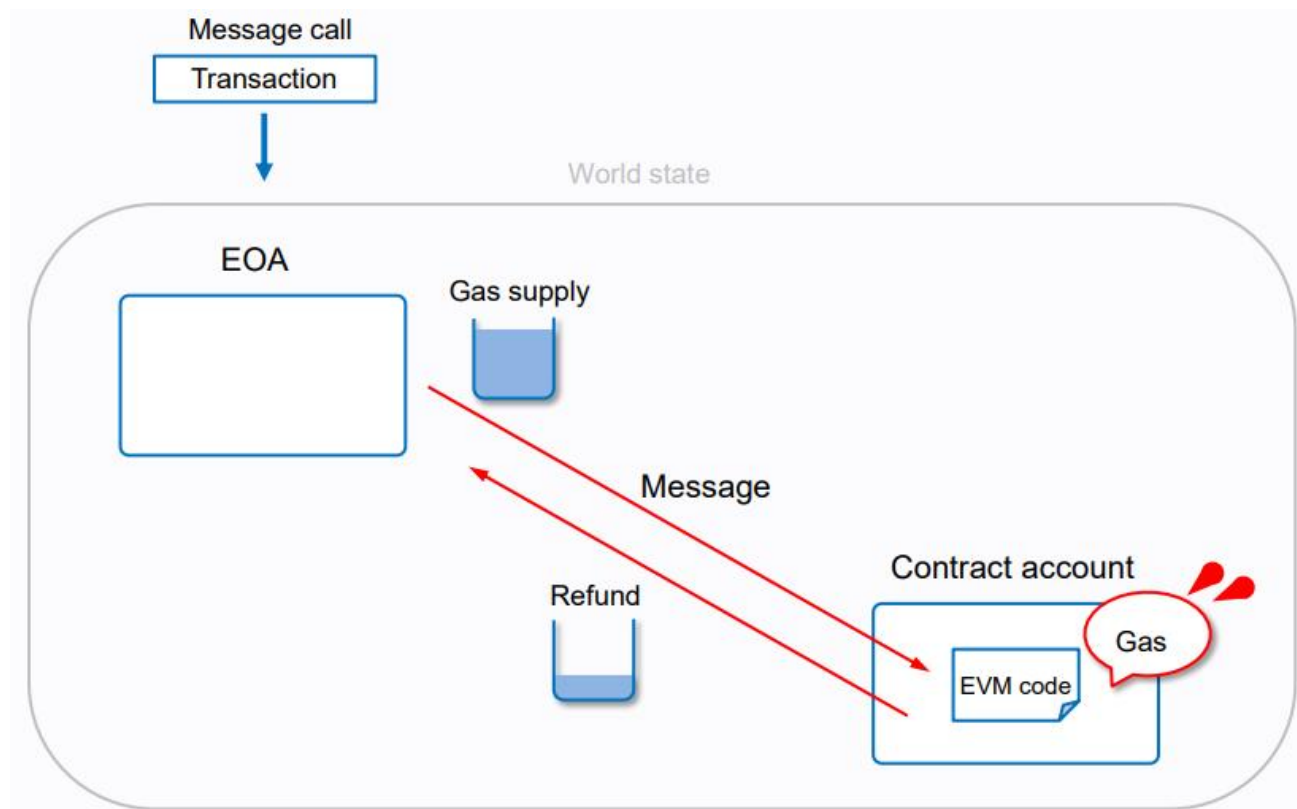
# 以太坊虚拟机 EVM

- 以太坊中的智能合约代码需要按照提前规定的以太坊虚拟机执行
  - 栈机、256-bit字长、**确定性执行**、汇编型代码



# EVM 燃料机制

- 调用合约的外部拥有账户 (EOA) 负责支付 燃料费
- 如果 燃料费 有剩余, 会返回给支付 燃料费 的 EOA
- 如果 燃料费 不足, “燃料” 计数器会抛出 out-of-gas 异常



比特币、以太坊、超级账本

# 比特币存储目录

```
$ tree ~/.bitcoin/regtest/
```

```
├── banlist.dat
├── blocks
│   ├── blk00000.dat Block数据(磁盘文件)
│   ├── index Block元数据(levelDB)
│   │   ├── 000005.ldb
│   │   ├── 000006.log
│   │   ├── CURRENT
│   │   ├── LOCK
│   │   └── MANIFEST-000004
│   └── rev00000.dat UTXO随区块的增删
├── chainstate UTXO集合 (levelDB)
│   ├── 000005.ldb
│   ├── 000006.log
│   ├── CURRENT
│   ├── LOCK
│   └── MANIFEST-000004
├── debug.log
├── fee_estimates.dat
├── indexes
│   └── txindex
│       ├── 000003.log
│       ├── CURRENT
│       ├── LOCK
│       └── MANIFEST-000002
├── mempool.dat
├── peers.dat
├── wallets
│   ├── db.log
│   └── wallet.dat
└── ...
```

.dat 文件存储序列化后的二进制数据结构;

.ldb .log ... => levelDB 数据库

# 比特币 vs 以太坊

	比特币	以太坊(PoS升级前)
哈希函数	SHA256、ripemd160	Keccak256 (NIST SHA3标准变体)
数字签名	签名算法: ECDSA 椭圆曲线: secp256k1	签名算法: ECDSA 椭圆曲线: secp256k1
传输与通信	P2P网络	P2P网络
共识机制	PoW	PoW (目前已经升级 PoS)
挖矿难度	目标为平均10分钟一个区块	目标为12-15秒一个区块
区块奖励	基础奖励每210000个区块减半	原始设计不含有减半机制, 目前引入了更复杂的发行量控制机制
状态是否在链上认证?	UTXO集合不在链上认证	所有的状态通过默克尔基数树在链上认证
区块是否包含叔块指针?	否, 仅指向前一个区块	是, 如果前一个区块有分叉 (即存在叔块), 使用叔块指针包含叔块
记账模型	UTXO	账户
支持图灵完备的链上计算?	仅支持功能受限的少数脚本, 无法实现图灵完备的智能合约	支持 (接近) 图灵完备的智能合约 (仍受限于燃料上限)



# Fabric 交易执行流程

## 执行-排序-验证 v.s. 排序-执行

- **优势:**

- 交易执行并行化
- 支持通用的非确定性编程语言 (Go、node.js、Java)
- 交易执行的chaincode插件化
- 排序过程的共识机制可插拔

- **问题:**

- 排序后的交易可能存在冲突 => 损害有效吞吐量
- readset 检验仍是顺序的、且I/O bound => 限制交易执行速度

**思考:**

HL Fabric 的交易执行流程是最佳实践么？

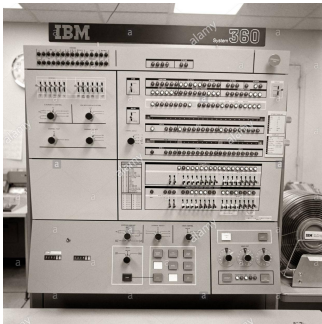
展望

# 应用现状

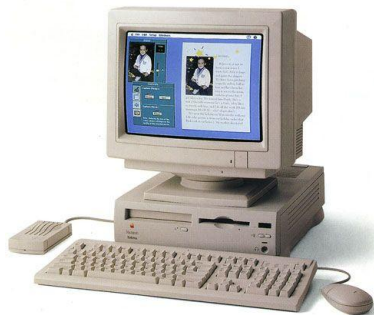
Number of startups is more than number of experts

The technology is still in its infant stage – many grand challenges remain

It is like Internet 20 years ago



1970s



1980s



2000s-now



2020s ???

# 应用前景

Every full node in the Internet will execute the contract codified into ledger, and enforce autonomous payment.

The limitation is  
your  
imagination

.....

# 挑战与机遇

Blockchain is like Internet 20 years ago, there are many technical challenges:

scalable & secure consensus,  
external oracles,  
cross-chain communication,  
semantically verifiable smart contract,  
privacy & anonymity,  
lightweight client for IoT,  
use-case oriented protocol design...

***And also, great chances!!!***

**Be part of the  
history?**

谢谢