

# *Bitcoin*

## *An Introduction*

Oct. 18, 2018

李其柄

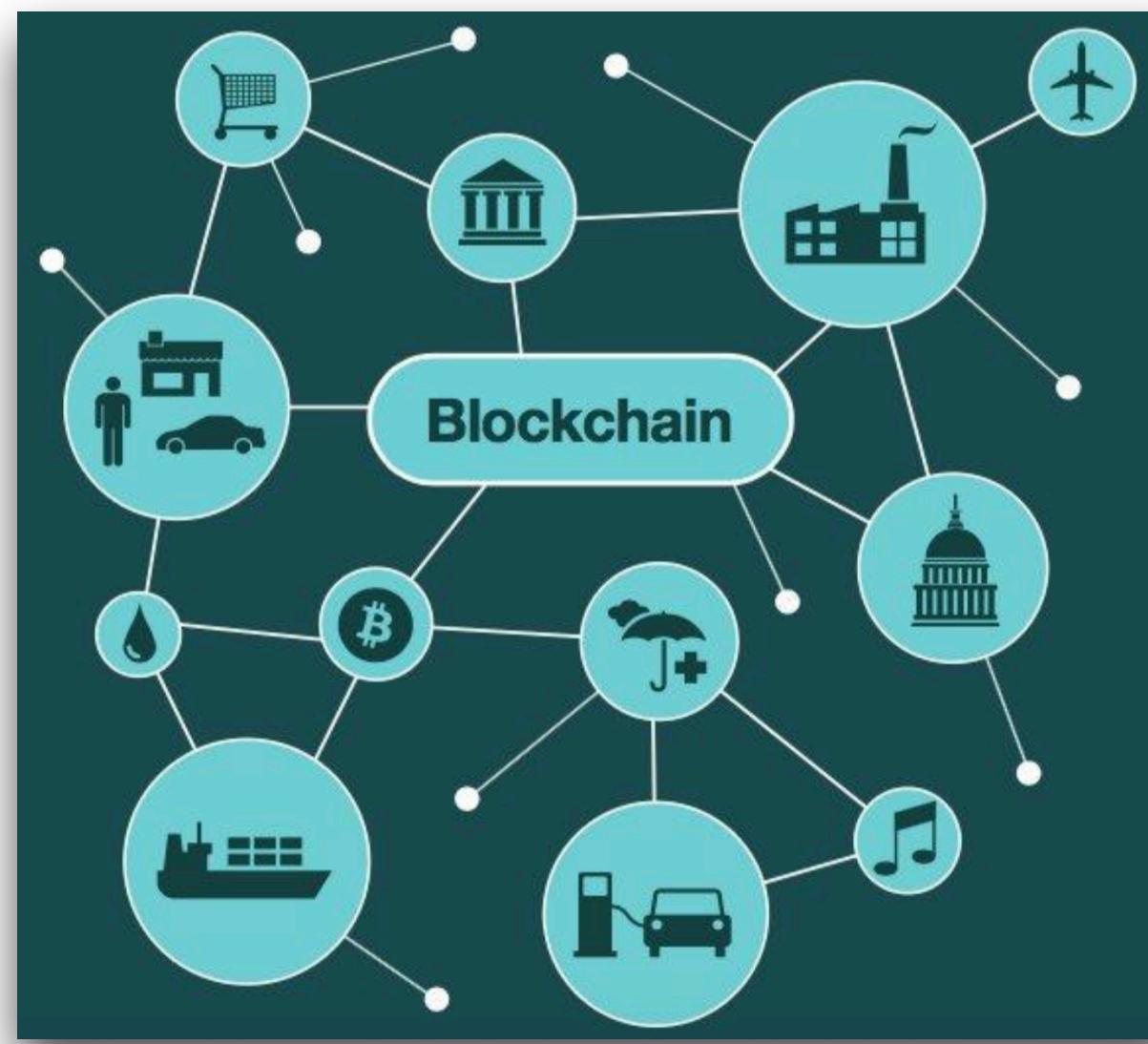
qblee@zju.edu.cn





## Bitcoin

### A peer-to-Peer Electronic Cash System

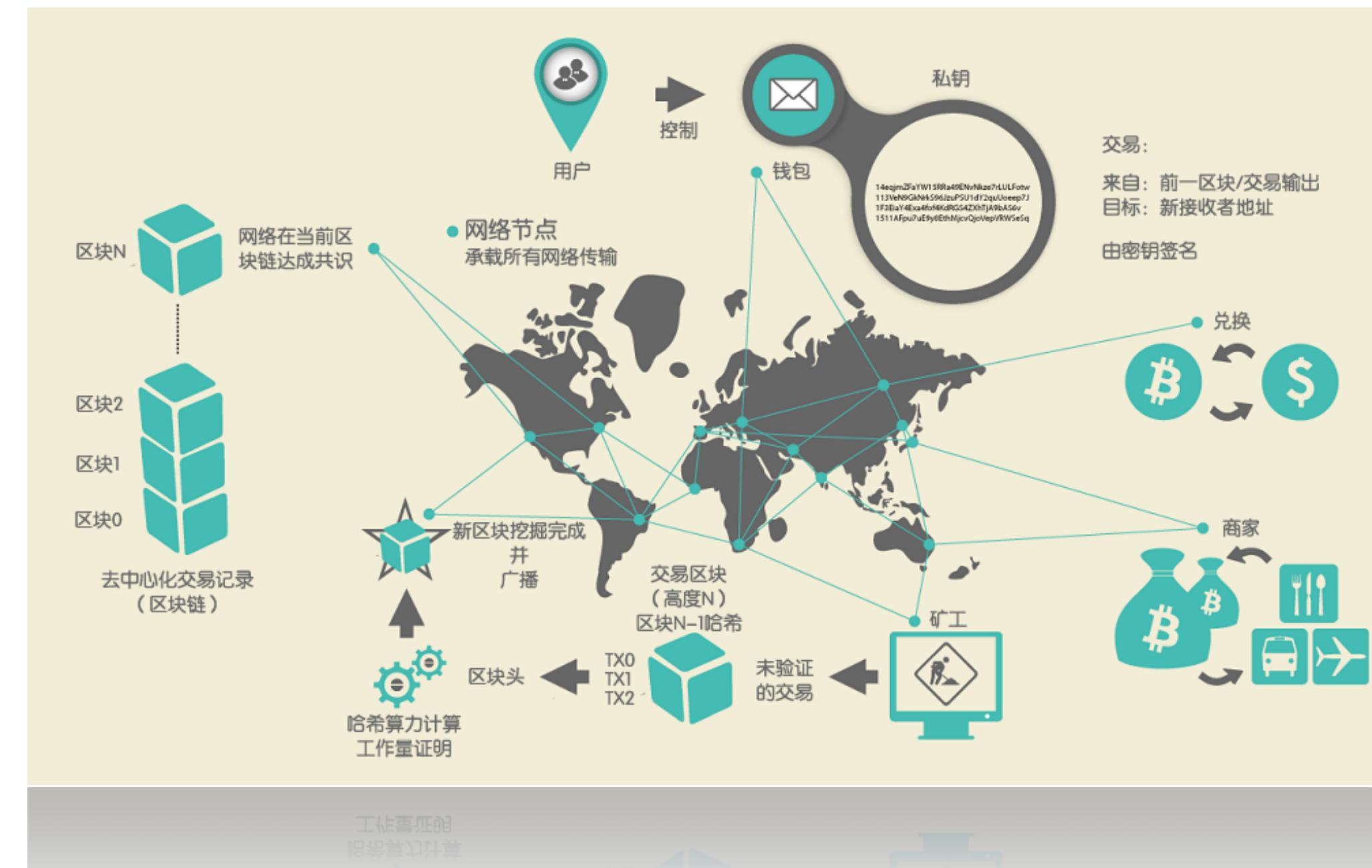


制造信任的机器，价值传输网络

- ① 我可以相信钱是真实的，不是假的吗？
- ② 我可以相信数字金钱只能花一次(双重支付)吗？
- ③ 我可以确定没有人能够声称这笔钱属于他们而不是我吗？

——《精通比特币》

### 用户、交易和矿工



### 组合创新

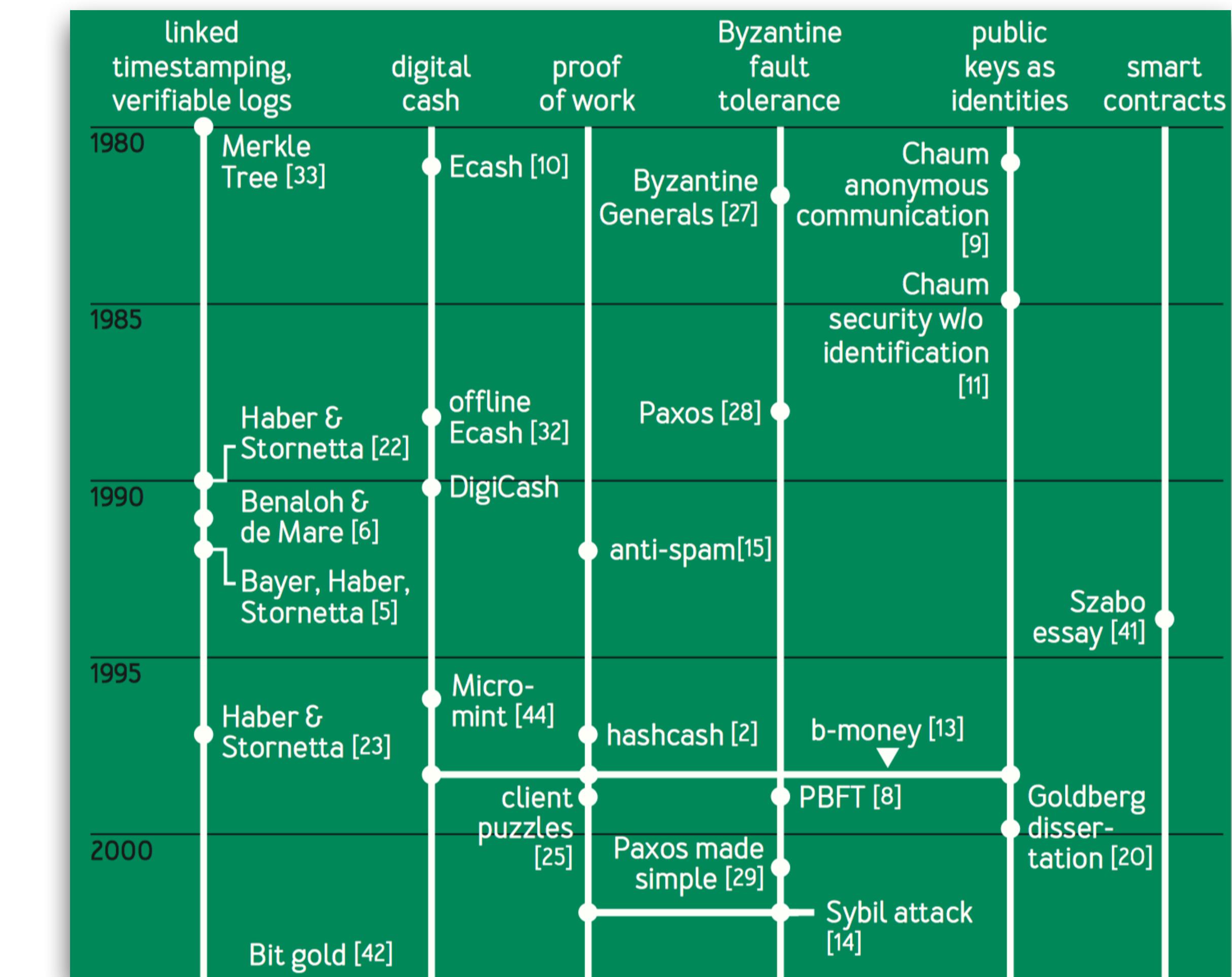
- A. 去中心化的对等网络
- B. 公共交易总帐(区块链)
- C. 交易验证和货币发行的一套规则  
(密码学和激励设计)
- D. 全球去中心化共识机制 (POW)

电子**现金**系统，解决双重支付，区块链即公共总帐本  
不需要中央权威机构清算结算，降低成本

## Academic Pedigree

组合创新的典范—比特币的非凡和成功之处不在于它处于任何组件研究的前沿，而在于它整合了许多不相关领域的旧创意。

- ① Digital cash (David Chaum)
- ② The LEDGER, UTXO
- ③ Linked timestamping
- ④ Merkle trees
- ⑤ Peer-to-peer network
- ⑥ Byzantine fault tolerance
- ⑦ Proof-of-work (hashcash)
- ⑧ Public keys as identities
- ⑨ Mechanism Design



Nakamoto's true leap of **insight**

—the specific, complex way in which the underlying components are put together.

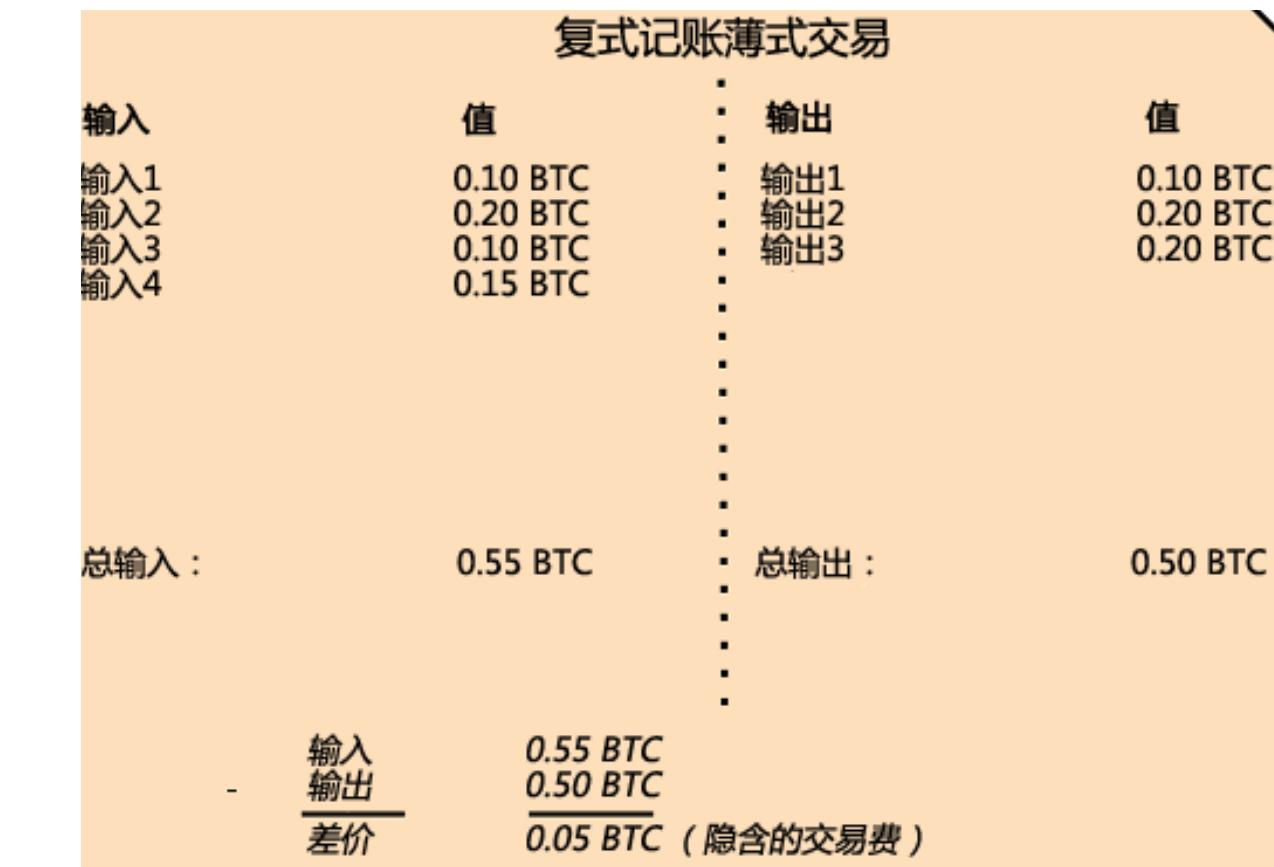


## The LEDGER

**总账本是由相互不信任的一组参与者共同维护的全局数据结构：**

- ① 可以添加新的交易
- ② 不能删除、修改或对已有交易重新排序

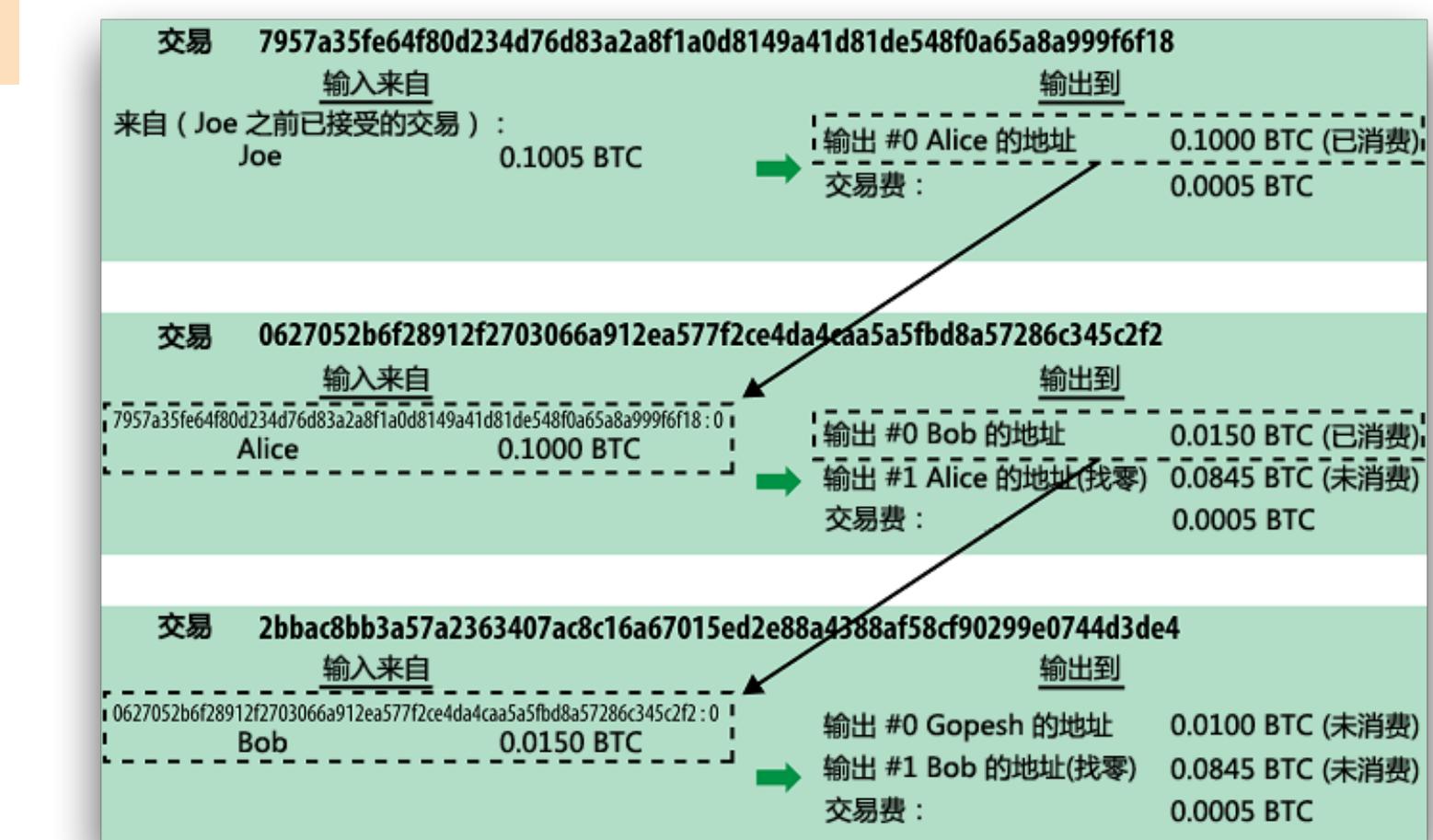
-> 不可变(不可篡改)，但未涉及双重支付



价值的组合与分割  
执行交易需要交易费  
(每千字节)

交易输入是之前几笔交易的输出(**UTXO**)  
-> 基于地址的所有权链

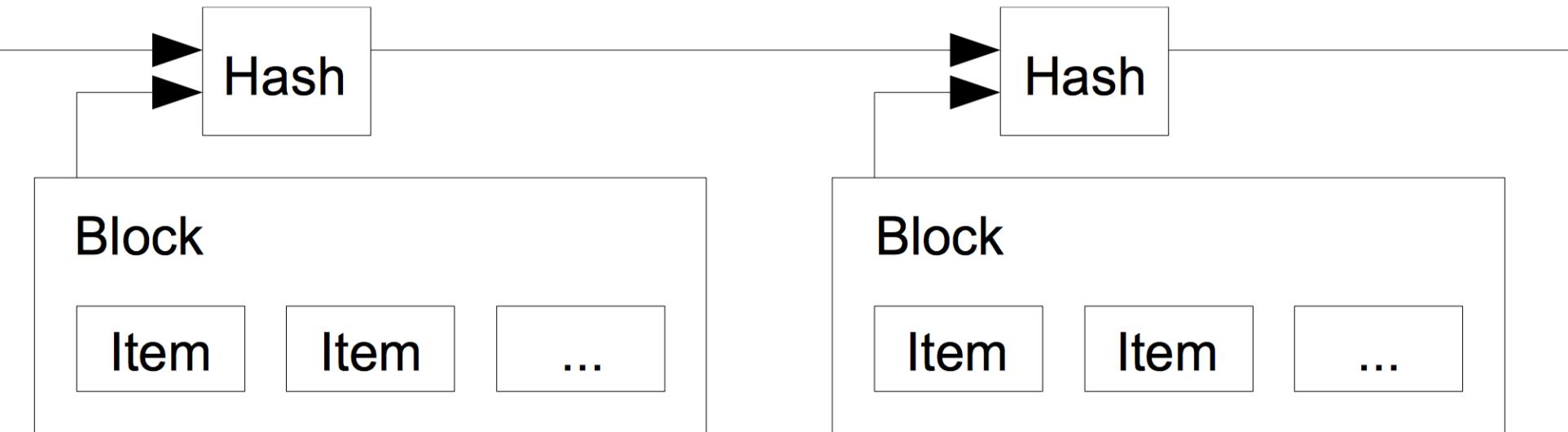
**UTXO**是不能再分割的、被所有者锁住的可用货币  
钱包扫描区块链，获取用户的所有**UTXO**算余额



总账本记录系统发生的所有交易，对所有参与者开放

在互联网这样参与者互不信任的场景下，怎样建立一个被所有人信赖的总账本？

## Timestamp Server



**文档创建者声明一个创建时间、文档的时间戳和前一个广播文档，并签名。递归地，形成一个倒退链。**

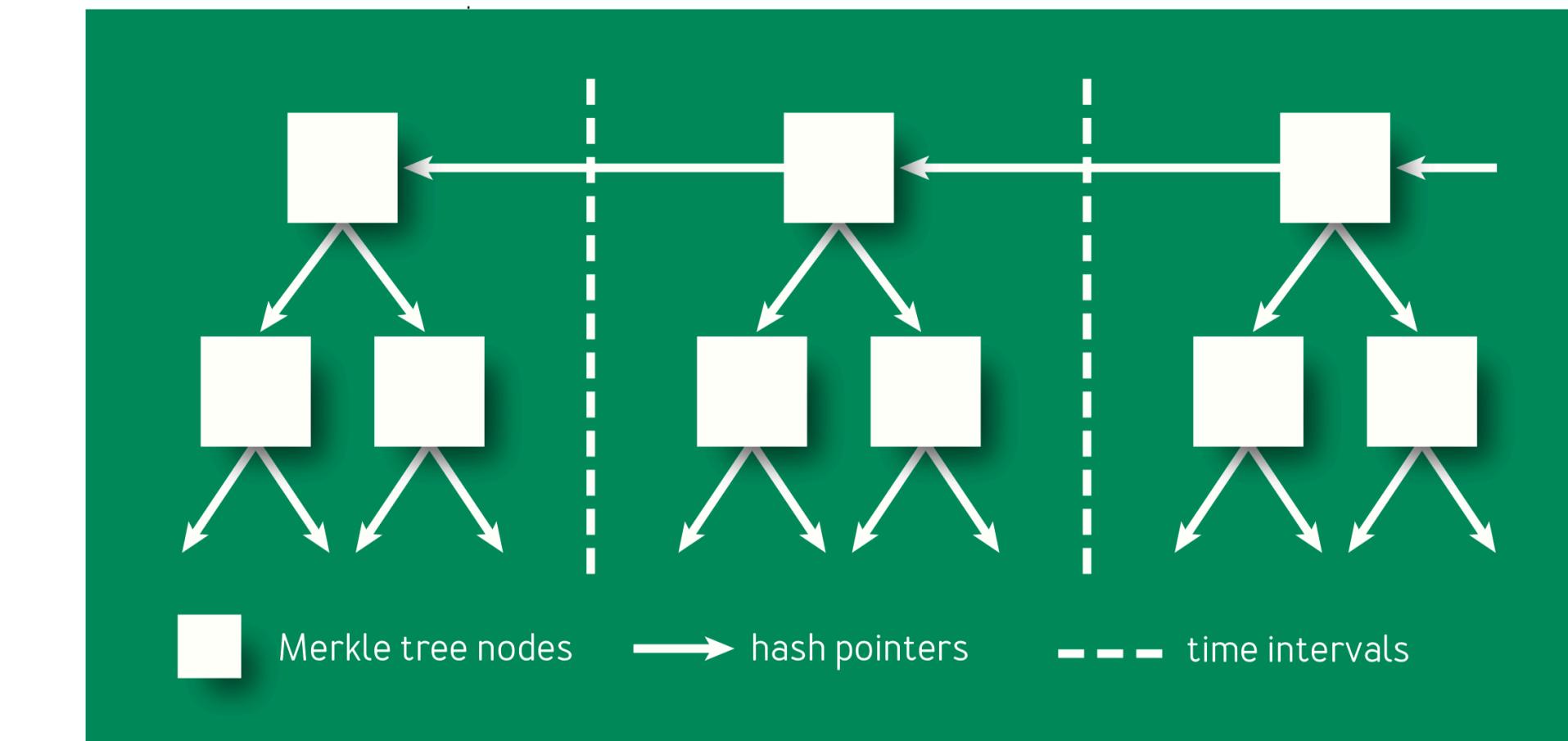
ps: 外部用户不能改变文档时间戳，因为创建者签名；  
创建者不能在不改变整个跟随链的情况下改变时间戳。  
->满足总帐本的基本数据结构要求

**Linked timestamping:** 用来确定文件(专利、商业合同)是在或**不迟于某个时间点**创建的  
但依靠可信服务提供商 (检查时间戳声明是否合理)，而Bitcoin则不需要依赖中心

## 具体实现: Merkle trees

对全局状态的部分进行高效的验证(**BTC SPV**)

- ① 最新块的哈希作为摘要。知道最新哈希，就可以从不受信任的来源下载总账本，并验证是否改变；
- ② 某人可以简单有效地向你证明某个特定的交易包含在总账本中。该交易的路径+后续块的哈希。



叶节点是交易，每个内部节点由两个指针组成。



## Transactions

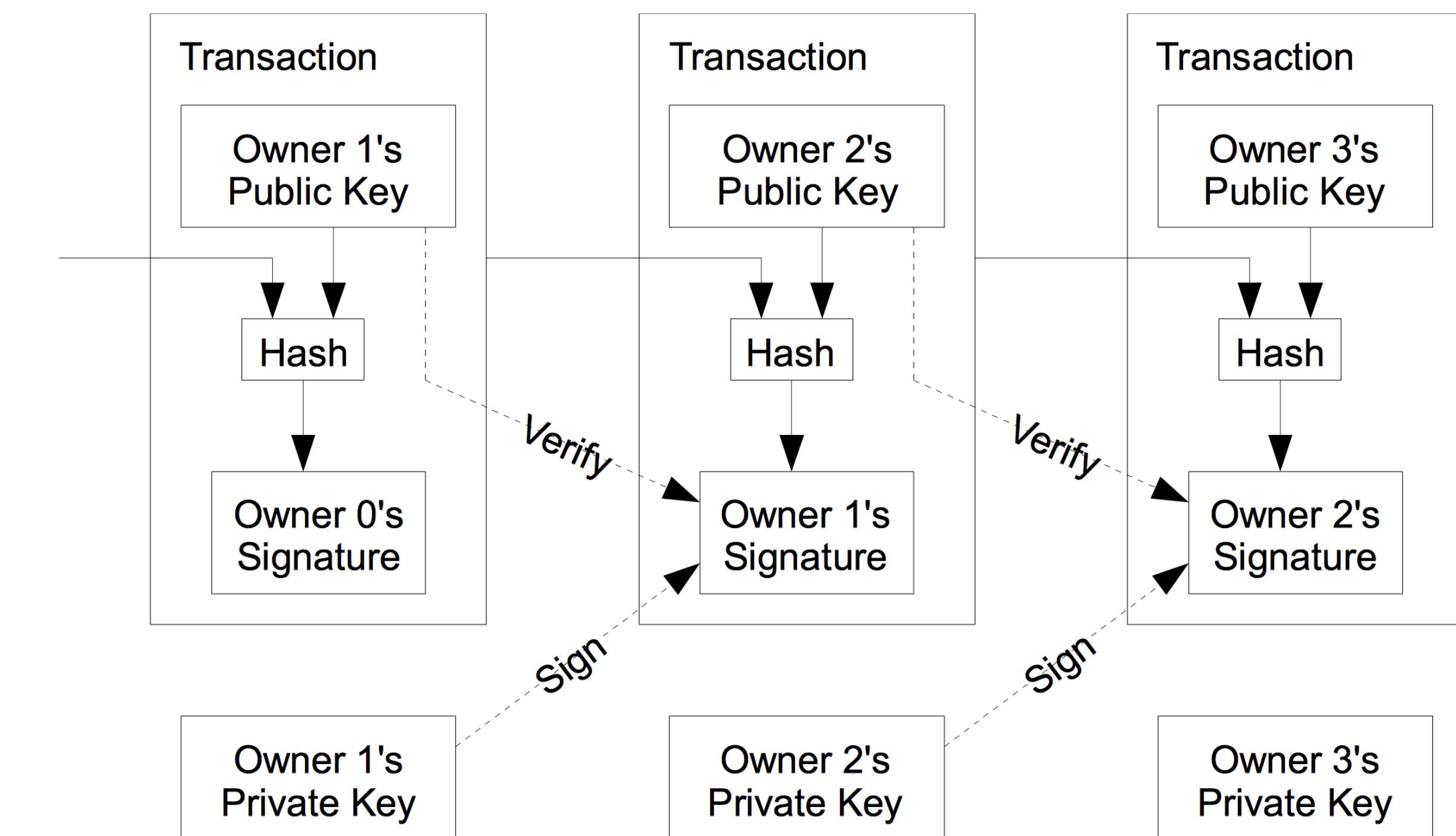
**锁定脚本: 把输出锁在特定的比特币地址(设定条件)**

**解锁脚本: 用私钥解锁解锁未消费的输出(满足条件)**

**→ 智能合约的雏形, 图灵非完备, 无循环**

通过对前一次交易(包含电子货币)和下一位拥有者的公钥(Public key) 签署一个随机散列的数字签名, 并将这个签名附在交易的末尾, 这枚货币就发送给了下一位所有者。

收款人希望能确保之前的所有者没有对更早发生的交易实施签名(**双重支付, 开放环境下**)。例如O2希望O1发给他的电子货币没有用在之前的其他交易当中(未消费)



造币厂 (mint) 回收验证-再发行

一笔交易一旦被足够多的后续区块确认(6个), 就成为总账簿中的有效交易



## Script

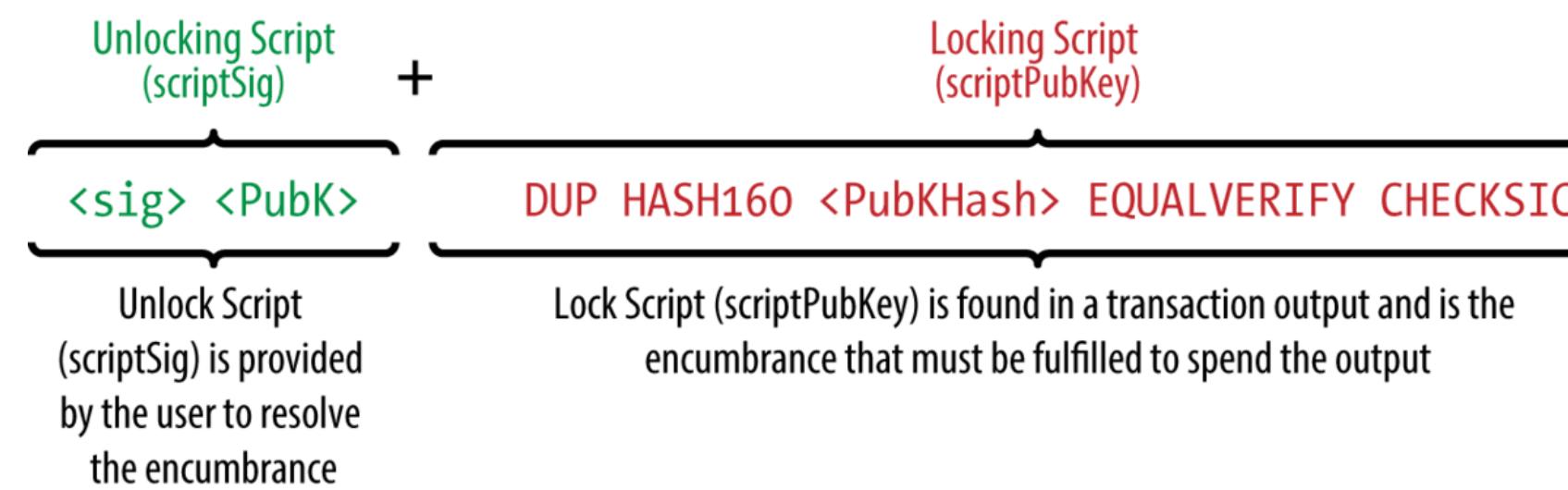
在比特币里  
没有具体的  
货币，没有  
发送者，没  
有接受者

```
{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe...",
      "vout": 0,
      "scriptSig": "3045022100884d142d86652a3f...",
      "sequence": 4294967295
    },
    "vout": [
      {
        "value": 0.01500000,
        "scriptPubKey": "OP_DUP OP_HASH160..."
      },
      {
        "value": 0.08450000,
        "scriptPubKey": "OP_DUP OP_HASH160..."
      }
    ]
  ]
}
```

**解锁脚本**

**锁定脚本/脚本公钥**

### P2PKH: 对公钥哈希的付款

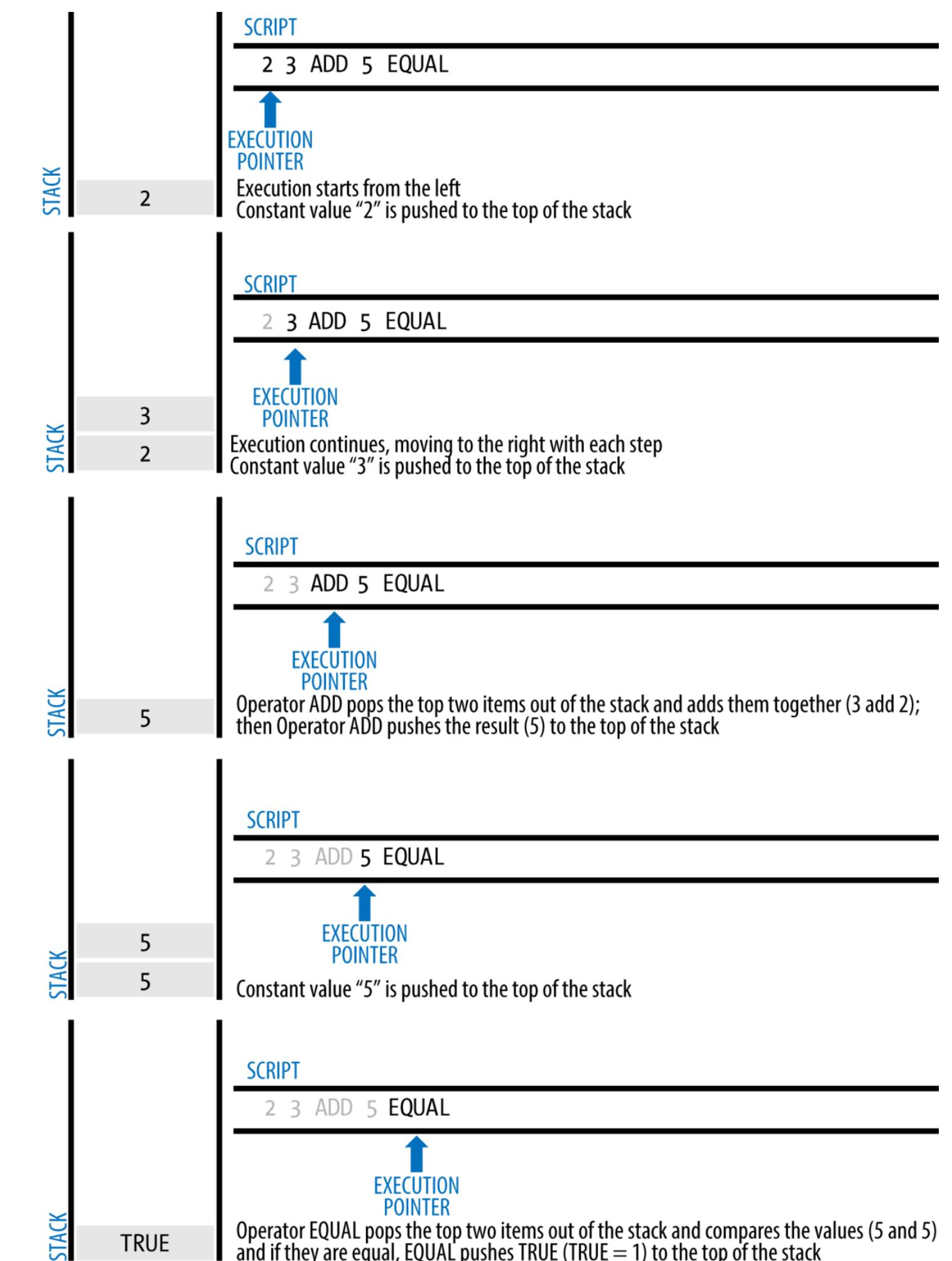


Unlock Script (scriptSig) is provided by the user to resolve the encumbrance

Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

使用部分算术操作码脚本作为锁定脚本：  
3 OP\_ADD 5 OP\_EQUAL  
对应的解锁脚本为：2

### 脚本执行堆栈



交易的输出会被创建成为一个包含这笔数额的脚本的形式  
只能被引入这个脚本的一个解答后才能兑换

## Proof-of-Work

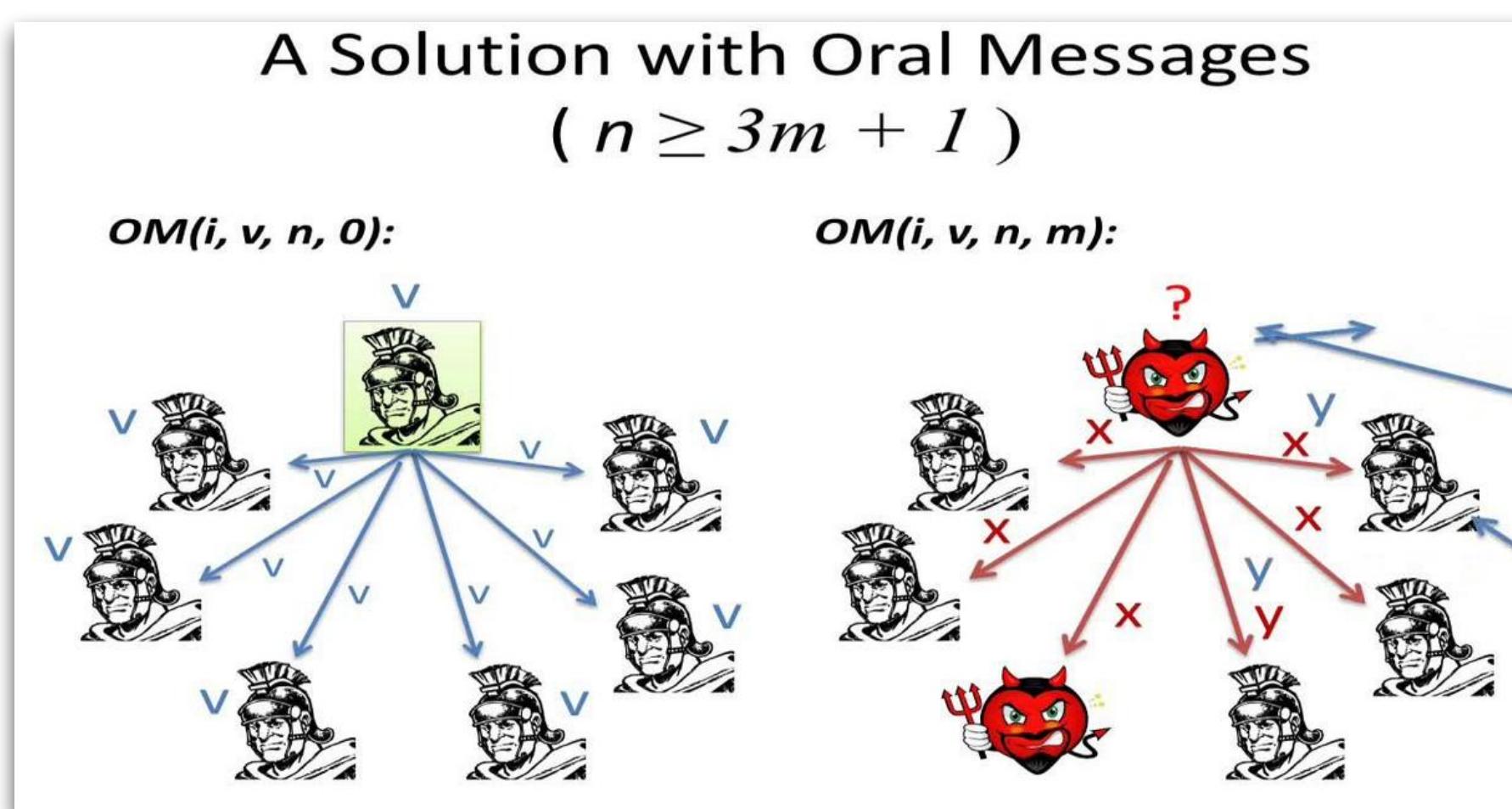
## Concurrency

容错分布式计算(恶意制作的行为和自然发生错误, 如不可靠的网络)

拜占庭将军问题 ->

状态复制(状态是余额, 交易是转换),

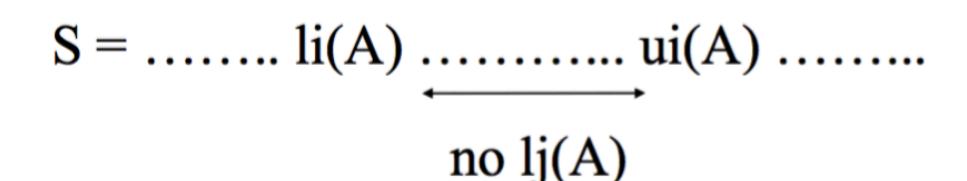
PBFT(联盟链, 节点少) vs Paxos and Raft



**Database(中心化帐本或不涉及作恶的分布式帐本)  
concurrency control for transactions**  
(事务的一致性协议, 规定读写顺序)

❖ Rule #1: Well-formed transactions  
 $T_i: \dots li(A) \dots pi(A) \dots ui(A) \dots$

❖ Rule #2: Legal scheduler



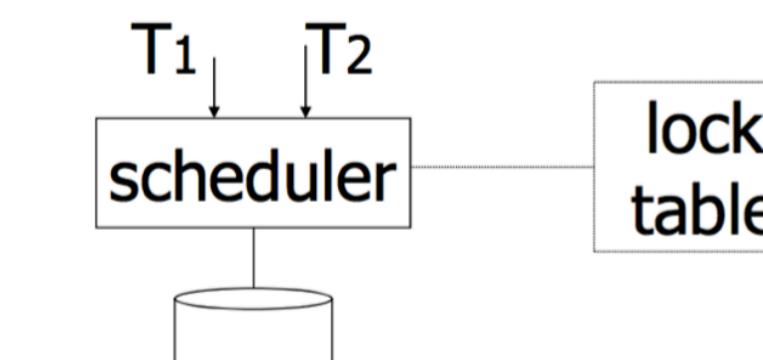
### serializable schedules

(一套加锁规则, 依赖中心调度)

Two new actions:

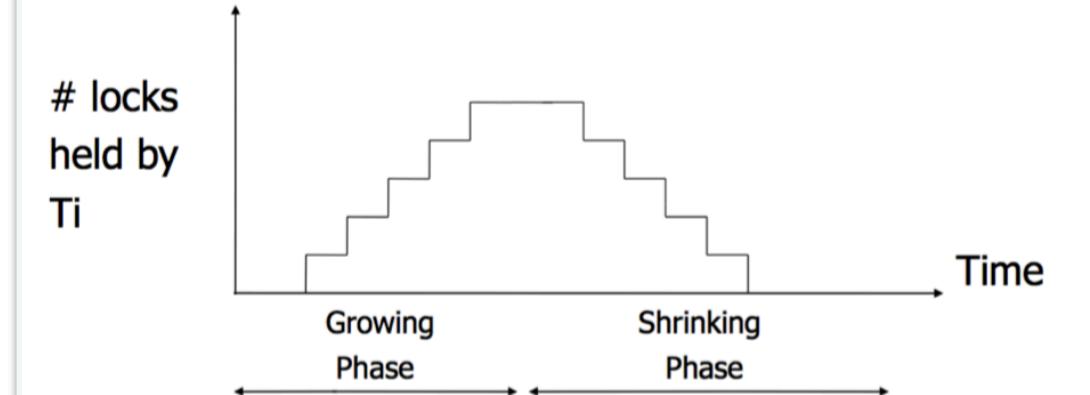
lock (exclusive):  $li(A)$

unlock:  $ui_i(A)$



### Rule #3 Two phase locking (2PL) for transactions

$T_i = \dots \xrightarrow{\quad} li(A) \xleftarrow{\quad} ui(A) \xrightarrow{\quad}$   
no unlocks                                    no locks



分布式帐本容易分叉, 难达到一致性(网络延迟或攻击者->开放环境下的双花), CAP不可能定理(CP)  
一些节点认为A块是最新的, 一些节点认为B块是最新的 (依靠链式时间戳无法解决分叉 by Mike\_1998)



## Proof-of-Work

### POW



电子邮件收件人只会处理**附带证明**(发件人执行的计算工作量)的电子邮件，同时工作量证明**特定于电子邮件以及收件人**。

- ① 防止一篇邮件能发给多个收件人
- ② 防止向同一个收件人发送多个邮件
- ③ ps: 不特定的话，成本和一对一发送是一样的

ps: 需要给权威机构开**后门**，例如开放一个发送邮件不产生成本的邮件列表。在不做工作量证明的情况下解决问题，因为他们起邮件中转作用。

① 阻止垃圾邮件(Cynthia Dwork, Moni Naor 1992)，**puzzle属性**: 难计算，易验证

② Hashcash(Adam Back 1997)，不是电子现金，因为它没有防止双花的保护，不能对等交换

## Hashcash—基于哈希的工作量证明



ADAM BACK

不需要中心权威(反政府)，基于哈希函数的特性：

- ① 找到哈希到特定输出的输入的唯一方法是尝试各种输入(枚举)。
- ② 找到哈希到另一输出的输入的唯一方法是再次枚举进行哈希。

## Sybil-对抗网络

**女巫攻击**: 对等网络中，单一节点具有多个身份标识，削弱了冗余备份的作用

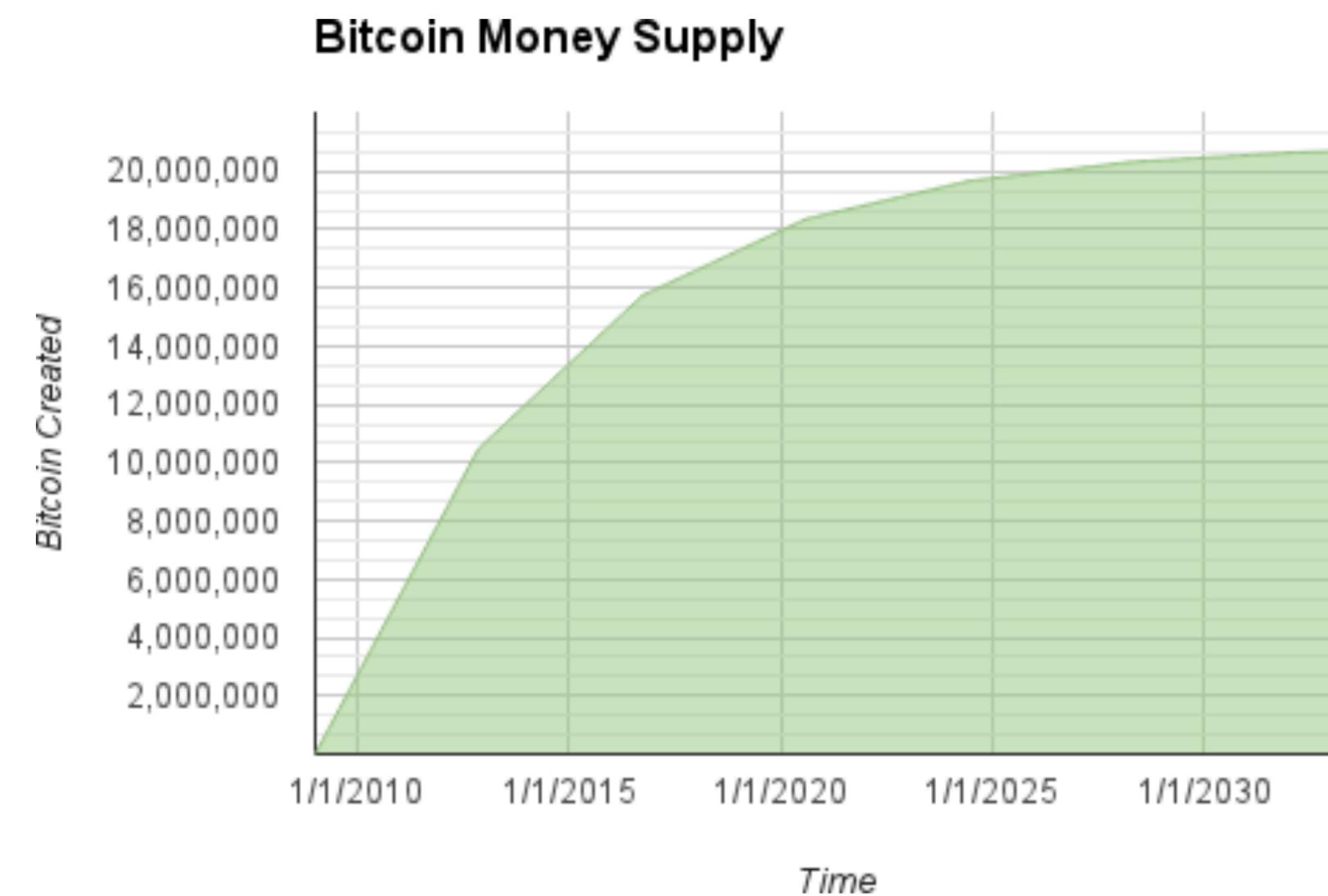
使用POW来消除节点身份问题，**不再需要验证身份**，开放的对等网络可以运行BFT协议(有数学证明，上下界)。BTC用了这个思路，激励诚实。

- ① 求解hashcash难题，恶意节点(多个身份)无法来得及解多题，但优势大
- ② 后续论文(2005)指出诚实的节点应该声称自己有足够的计算力建立多个身份



## Proof-of-Work

### Incentive



每开采210,000个块，大约耗时4年，货币发行速率降低50%。第一个四年中，每个区块创造出50个新比特币。

**不再需要验证身份，但需要激励节点诚实的完成工作**

(前面提到的工作，要求诚实节点声称自己有足够的计算力)

中本聪认为，没必要假设行为是诚实的，因为**行为是被激励的**（博弈论、行为经济学、机制设计）。进一步解决了问题：什么激励节点执行昂贵的工作量证明计算？**数字货币(digital currency)**

- ① 时间戳和拜占庭协议的研究没有触及**节点激励**问题，直到2005年，也没有使用工作量证明来**消除节点身份**问题
- ② 电子货币和工作量证明之间的双环困境。hashcash、b-money 和bit gold没有吸收**一致性(共识)**算法的思想来解决双花问题
- ③ BTC避免了工作量即现金(**proof-of-work-as-cash**)，实现了难题解决和经济价值的两倍脱钩(难度调整和挖矿奖励递减)

**矿工挖矿(竞争求解Puzzle)，赚取交易费 + 挖矿奖励(Coinbase交易, 10个确认?)**

类似于耗费资源去挖掘金矿, ps: CPU的时间和电力消耗就是消耗的资源



## Proof-of-Work

## Consensus

### 4种独立过程：

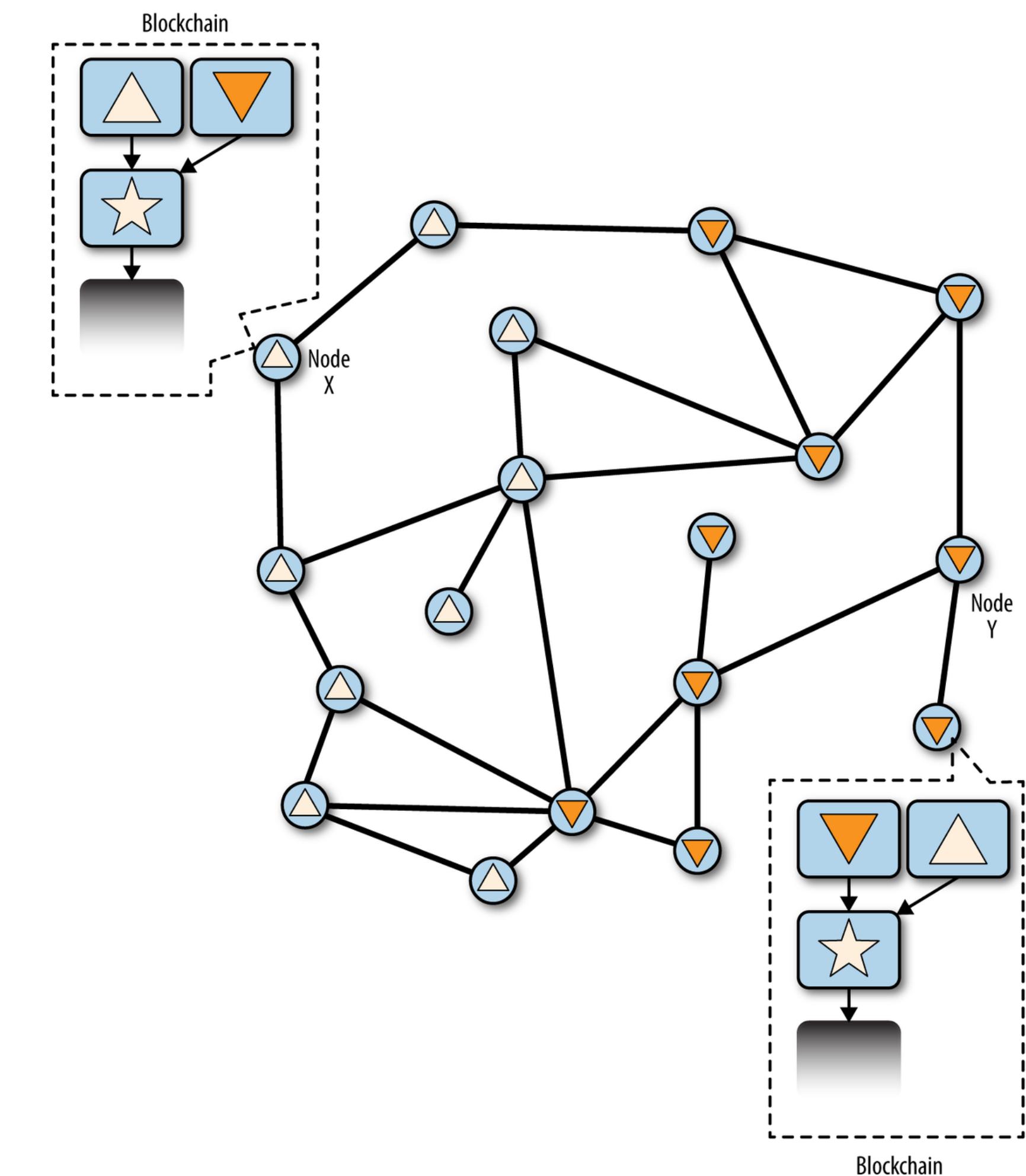
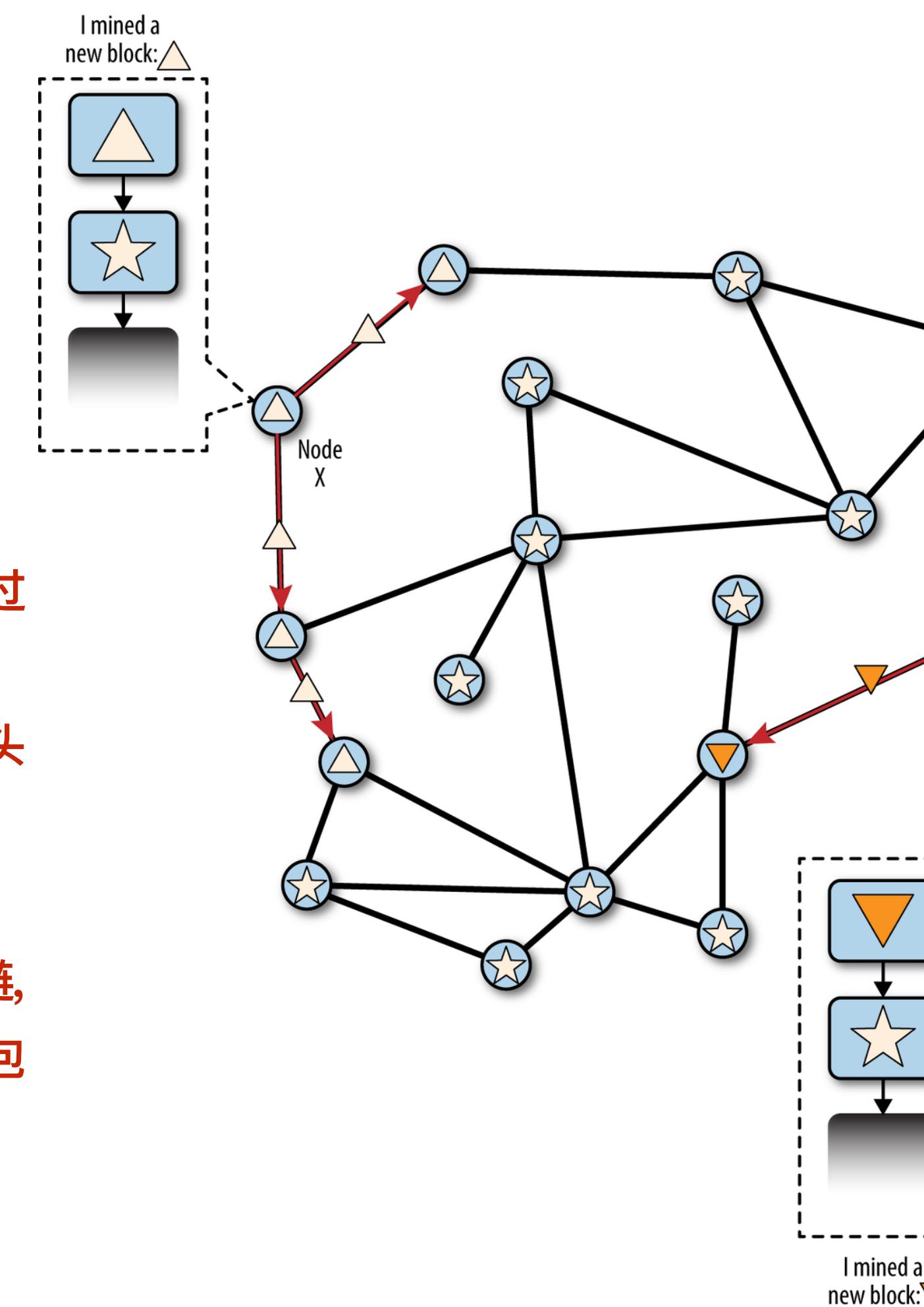
- ① **交易验证**: 每个全节点依据综合标准对每个交易进行独立验证
  - ② **挖矿出块**: 通过完成工作量证明算法的验算，挖矿节点将交易记录独立打包进新区块
  - ③ **新块组装**: 每个节点独立的对新区块进行校验并组装进区块链 (e.g., 验证第一个交易是coinbase)
  - ④ **最长链选择**: 每个节点对区块链进行独立选择，在工作量证明机制下选择累计工作量最大的区块链
- 
- A. 不诚实的矿工所产生的区块将被拒绝，不但失去奖励，也浪费了本可以求解问题的电费
  - B. 对51%算力者(攻击者)，按照规则行事更有利可图，能拥有更多BTC，而不是破坏系统使自身财富受损

BTC去中心化共识由所有网络节点的**4种独立过程相互作用**而产生

## Proof-of-Work

### Fork

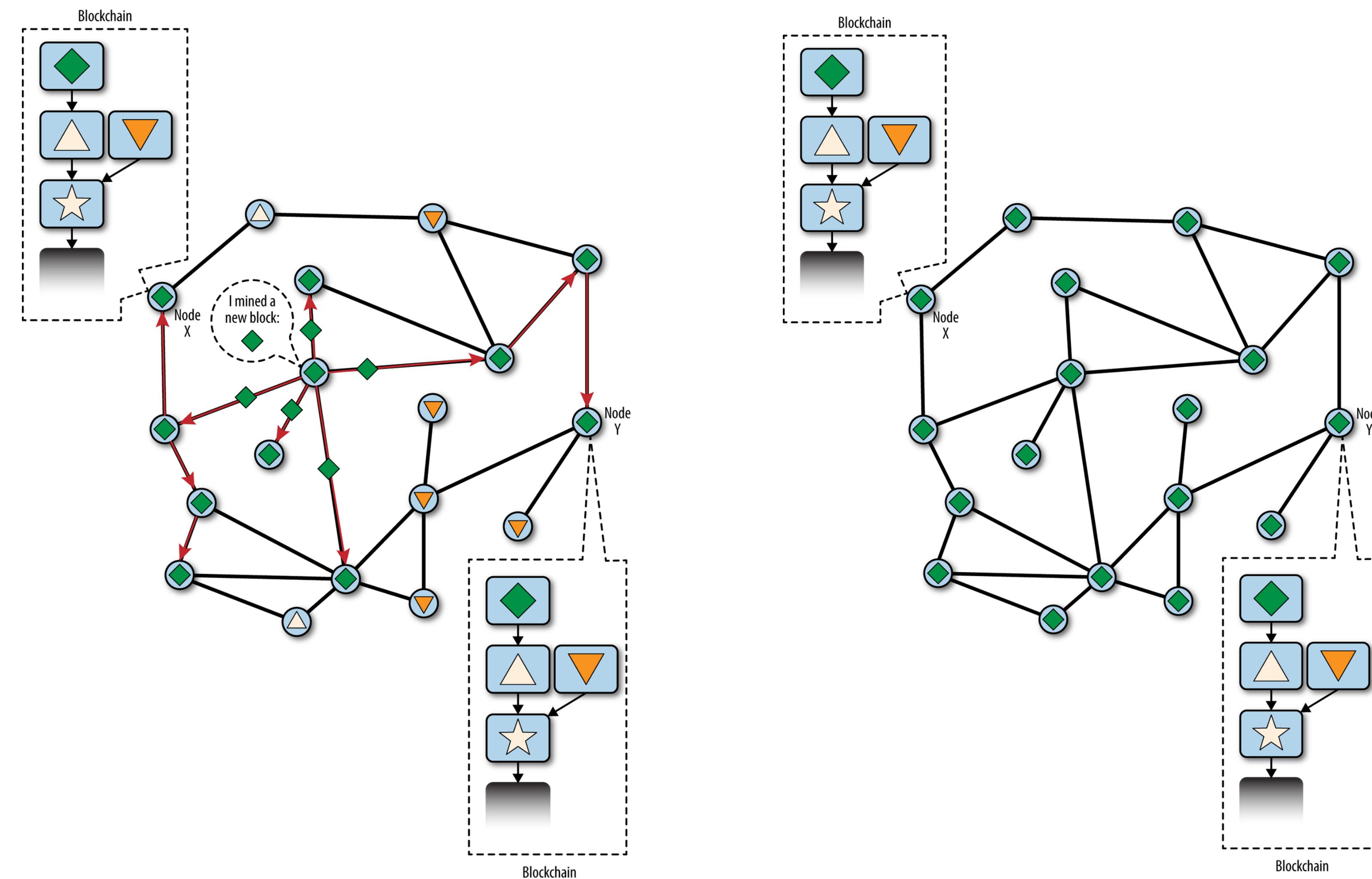
- ① 收到新区块，验证有效后，通过“*previous hash*”找父区块；
- ② 如果是链接到主链的，放弃手头的工作(因为累积工作量最大)，立即开始计算下一区块；
- ③ 保留备用链，未来可能超过主链，保证那时候该节点具有重新打包所需的信息



分叉即在不同节点间发生的临时差异，类似git全量存储，维护主链、备用链、孤立链(没有找到父区块)  
每个节点总是选择并尝试延长代表累计了最大工作量证明(最长)的区块链 (greatest cumulative work chain).

## Proof-of-Work

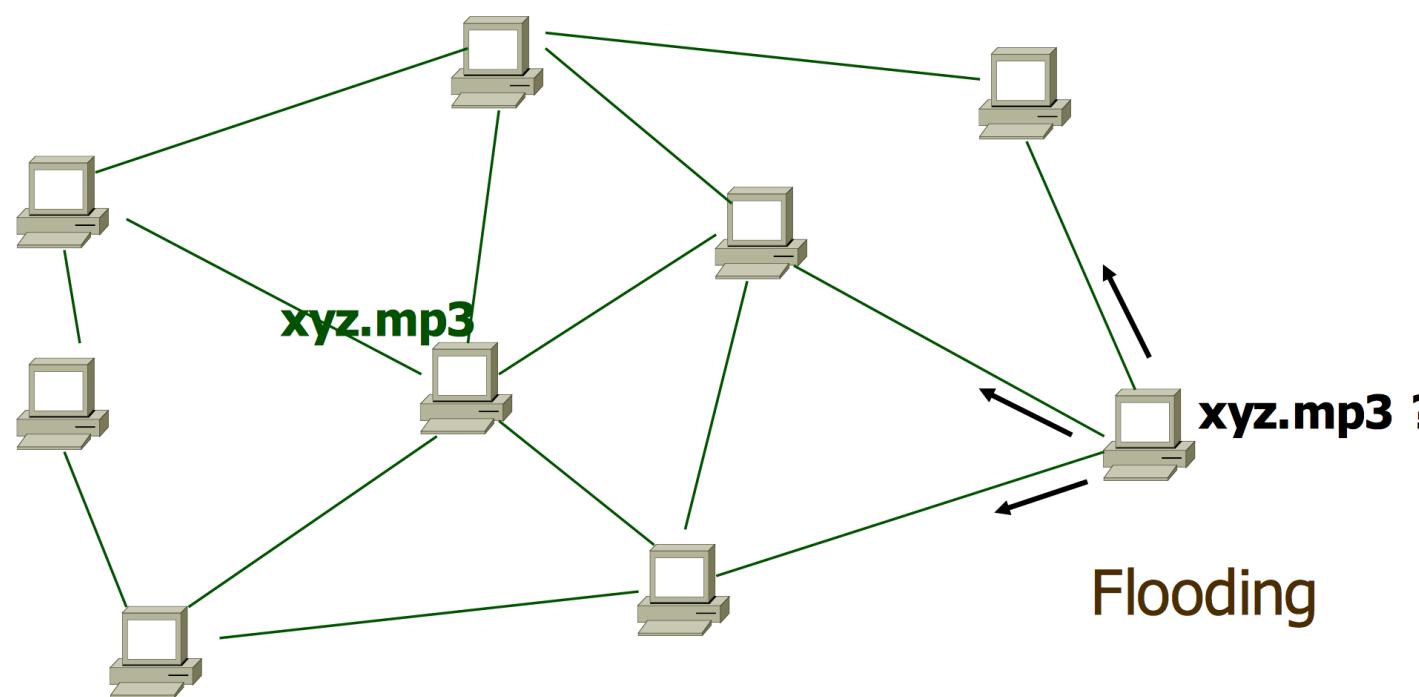
## Fork



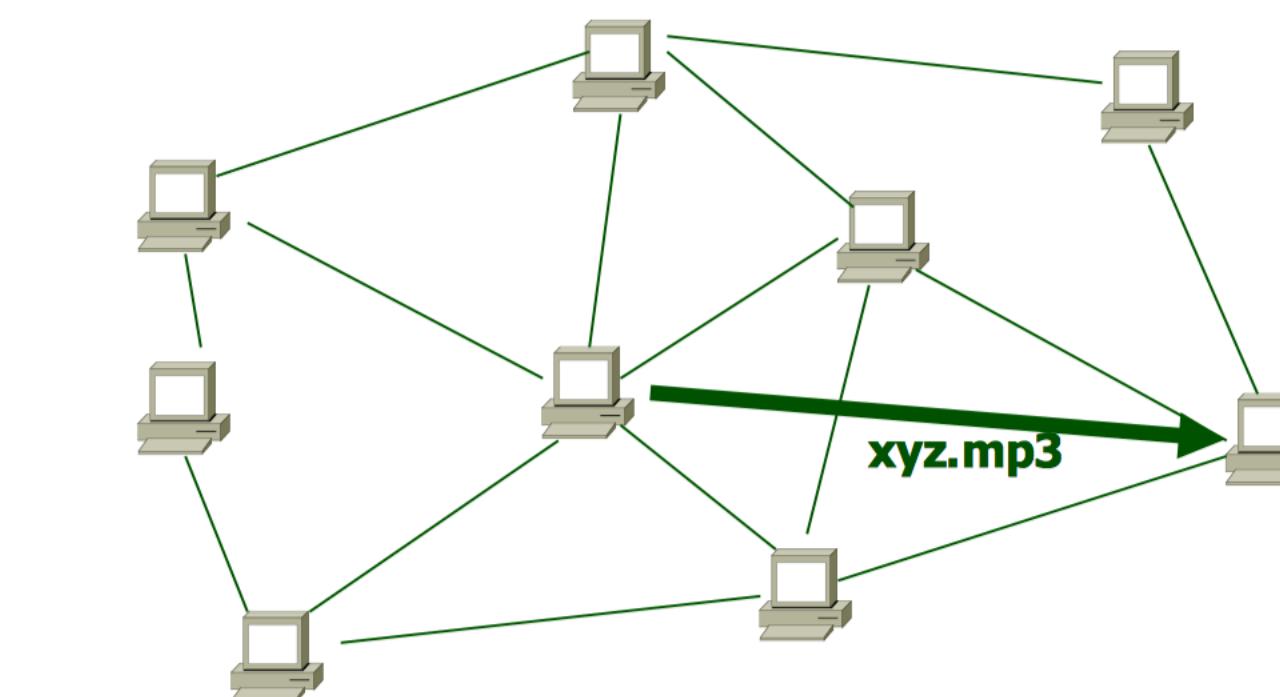
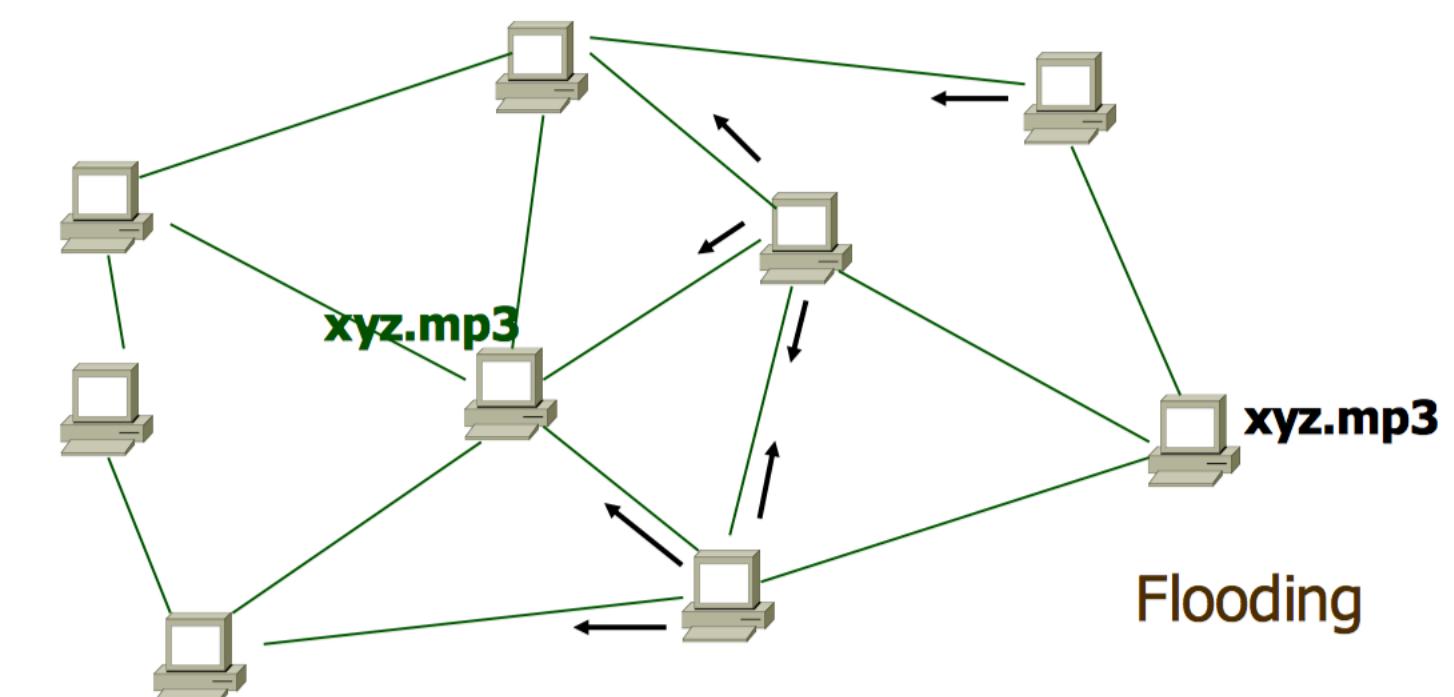
节点将新的区块添加到备用链，同时比较备用链与主链的难度

如果备用链比主链积累了更多的难度，节点将切换备用链作为其新的主链

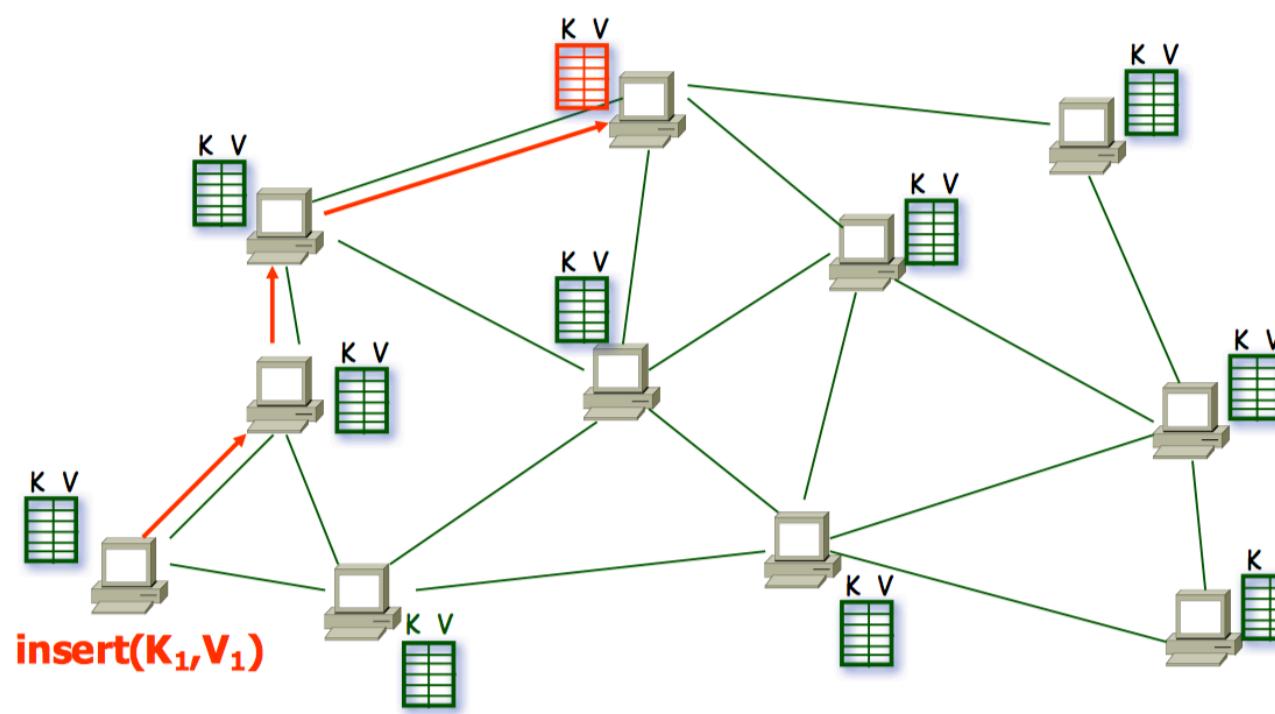
## P2P Network



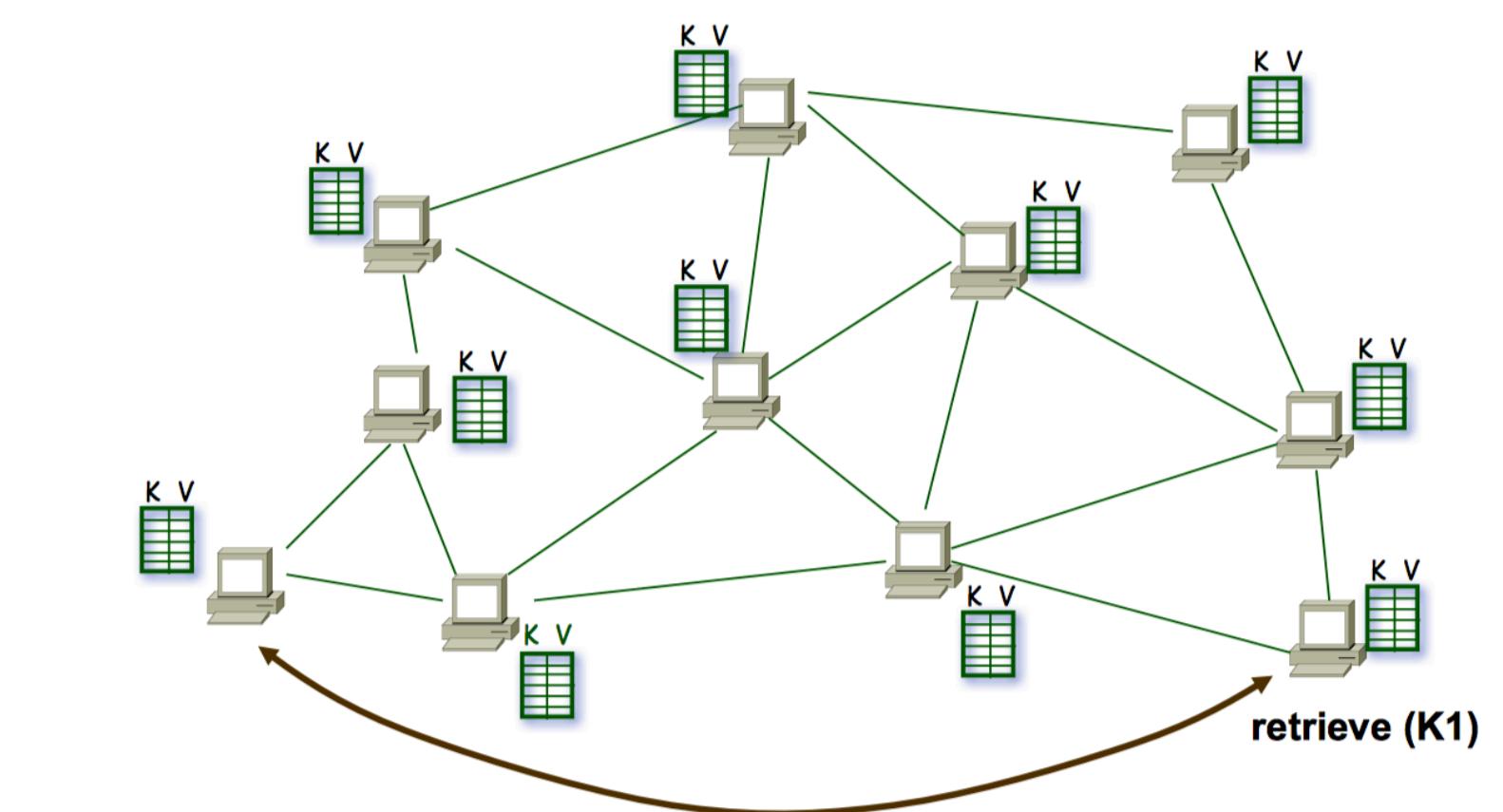
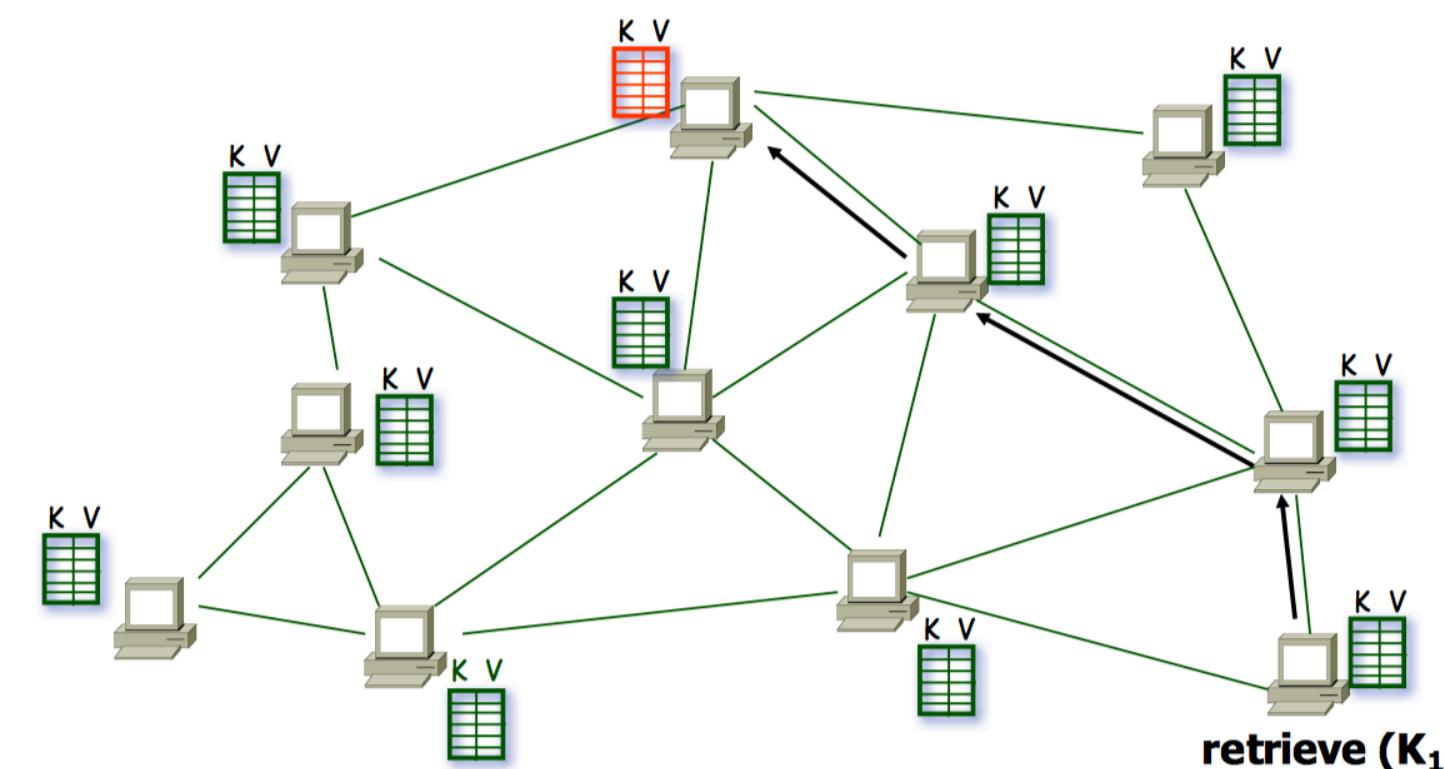
## Flooding on Overlays(泛洪)



K是文件标识, V是路由信息



## DHT(分布式哈希表)



## Gnutella、Freenet、DHT

Chord: Consistent Hashing(一致性哈希)、Finger Tables(具体实现, 节点加入和离开)



## Chain

```
class Blockchain(object):
    def __init__(self):
        self.current_transactions = []
        self.chain = []
        # Create the genesis block
        self.new_block()

    def new_block(self):
        # Creates a new Block and adds it to the chain
        pass

    def new_transaction(self):
        # Adds a new transaction to the list of transactions
        pass

    def last_block(self):
        return self.chain[-1]

    def hash(block):
        """
        生成块的 SHA-256 hash值
        """
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

提供两份代码， proof-of-work(PPT内容)和blockchain\_imp(自学)

- ① 区块链是一个不断延伸的基于哈希的工作量证明的链条
- ② 由区块(时间戳+交易集合)构成的不可变、有序的链结构



## Transaction

```
class Blockchain(object):
    ...
    def new_transaction(self, sender, recipient, amount):
        """
        生成新交易信息，信息将加入到下一个待挖的区块中
        :param sender: <str> Address of the Sender
        :param recipient: <str> Address of the Recipient
        :param amount: <int> Amount
        :return: <int> The index of the Block that will hold this transaction
        """

        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1
```

交易数据结构：发送地址、接受地址和金额，接收到的交易放入下一区块(未打包)

**ps:** 未涉及到Merkle树和基本的交易验证(e.g., 余额是否够)



## Block

```
block = {
    'index': 1,
    'timestamp': 1506057125.900785,
    'transactions': [
        {
            'sender': "8527147fe1f5426f9dd545de4b27ee00",
            'recipient': "a77f5cd fa2934df3954a5c7c7da5df1f",
            'amount': 5,
        }
    ],
    'proof': 324984774000,
    'previous_hash': "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7"
}
```

### 区块数据结构(head)

少了Merkel树根哈希

```
class Blockchain(object):
    ...
    def new_block(self, proof, previous_hash=None):
        """
        生成新块
        :param proof: <int> The proof given by the Proof of Work
        :param previous_hash: (Optional) <str> Hash of previous Block
        :return: <dict> New Block
        """

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        # Reset the current list of transactions
        self.current_transactions = []

        self.chain.append(block)
        return block
```

生成新块：高度+时间戳+交易+上一个区块哈希 +工作量证明 (+随机数Nonce+Merkel树根哈希)

添加新块到区块链，清空交易缓存 (**ps**: 实际中需要缓存交易池，找最长链时分叉、共识)



## POW

假设整数  $x$  乘以另一整数  $y$  的积的 Hash 必须以 0 结尾。

设变量  $x = 5$ , 求  $y$  的值?

```
from hashlib import sha256
x = 5
y = 0 # y未知
while sha256(f'{x*y}'.encode()).hexdigest()[-1] != "0":
    y += 1
print(f'The solution is y = {y}')
```

$\rightarrow y=21$

hash( $5 * 21$ ) = 1253e9373e...5e3600155e860

```
def proof_of_work(self, last_proof):
    """
    简单的工作量证明:
    - 查找一个  $p'$  使得  $\text{hash}(pp')$  以4个0开头
    -  $p$  是上一个块的证明,  $p'$  是当前的证明
    :param last_proof: <int>
    :return: <int>
    """

    proof = 0
    while valid_proof(last_proof, proof) is False:
        proof += 1

    return proof
```

```
@staticmethod
def valid_proof(last_proof, proof):
```

枚举  
ps: 验证只需调用一次valid\_proof()

验证证明: 是否 $\text{hash}(last\_proof, proof)$ 以4个0开头?

```
:param last_proof: <int> Previous Proof
:param proof: <int> Current Proof
:return: <bool> True if correct, False if not.
"""

guess = f'{last_proof}{proof}'.encode()
guess_hash = hashlib.sha256(guess).hexdigest()
return guess_hash[:4] == "0000"
```

**Proof-of-Work求解puzzle, 找出符合特定条件(前面N个0+随机数)的数字**  
**核心思想是要求这个数字很难计算出来(枚举), 但容易验证**



## POW

```

app = Flask(__name__)
blockchain = Blockchain()

@app.route('/mine', methods=['GET'])
def mine():
    return "We'll mine a new Block"

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    return "We'll add a new transaction"

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

Flask是轻量 Web 应用框架，将网络请求映射到 Python 函数

Flask 服务器将扮演区块链网络中的一个节点

```

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

```

```

        required = ['sender', 'recipient', 'amount']
        if not all(k in values for k in required):
            return 'Missing values', 400

```

```

        index = blockchain.new_transaction(values['sender'],
                                            values['recipient'], values['amount'])

```

```

        response = {'message': f'added to Block {index}'}
        return jsonify(response), 201

```

```

@app.route('/mine', methods=['GET'])
def mine():
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)
    # 新块奖励，发送者为 "0" 表明是新挖出的币
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )
    block = blockchain.new_block(proof)
    response = {...}
    return jsonify(response), 200

```

- ① /transactions/new: 创建一个交易并添加到区块
- ② /mine: 告诉挖矿节点去挖掘新的区块
- ③ /chain: 返回节点的整个区块链



## Demo

## 1. 查看初始区块链:

<http://localhost:5000/chain>

```
"chain": [
  {
    "index": 1,
    "previous_hash": "1",
    "proof": 100,
    "timestamp": 1506280650.770839,
    "transactions": []
  },
]
```

## 2. 用http post发送交易

调用/transactions/new接口

```
$ curl -X POST -H "Content-Type: application/json" -d '{
  "sender": "d4ee26eee15148ee92c6cd394edd974e",
  "recipient": "someone-other-address",
  "amount": 5
}' "http://localhost:5000/transactions/new"
```

## 3. 挖矿，把新交易打包到新区块

<http://localhost:5000/mine>

## 4. 查看更新后的区块链:

<http://localhost:5000/chain>

```
"chain": [
  {
    "index": 1,
    "previous_hash": "1",
    "proof": 100,
    "timestamp": 1506280650.770839,
    "transactions": []
  },
  {
    "index": 2,
    "previous_hash": "c099bc...fb7",
    "proof": 35293,
    "timestamp": 1506280664.717925,
    "transactions": [
      {
        "amount": 1,
        "recipient": "8bbcb347e0634905b0cac7955bae152b",
        "sender": "0"
      }
    ]
  }
],
```

单挖矿节点demo，用端口5000运行  
python pow.py -p 5000



## Consensus

### 添加相邻挖矿节点的路由

```

def register_node(self, address):
    """
    param address: Eg. 'http://192.168.0.5:5000'
    :return: None
    """

    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
    .....

def valid_chain(self, chain):
    """
    Determine if a given blockchain is valid
    """

    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        # Check that the hash of the block is correct
        if block['previous_hash'] != self.hash(last_block):
            return False

        if not self.valid_proof(last_block['proof'], block['proof']):
            return False

        last_block = block
        current_index += 1

    return True

```

### 递归验证其他链的工作量

### 共识规则，选择网络中的最长链

```

def resolve_conflicts(self):
    """
    :return: <bool> True 如果链被取代, 否则为False
    """

    neighbours = self.nodes
    new_chain = None
    max_length = len(self.chain)

    # Grab and verify the chains from all the nodes in our network
    for node in neighbours:
        response = requests.get(f'http://{node}/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            # Check if the length is longer and the chain is valid
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

    if new_chain:
        self.chain = new_chain
        return True

    return False

```

- ① 让一个节点知道它相邻的节点(实际用P2P实现)
- ② 规定最长的、有效的链才是最终的链



## Consensus

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

### @用http post将5001链的路由信息告知5000:

```
$ curl -X POST -H "Content-Type: application/json" -d
'{"nodes": ["http://localhost:5001"]}' "http://localhost:5000/nodes/register"
```

### 同步区块数据

<http://localhost:5000/nodes/resolve>

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

- ① /nodes/register: 接收URL形式的新节点列表
- ② /nodes/resolve: 执行一致性算法，确保节点拥有最长链

# THANK FOR YOUR ATTENTION

qblee@zju.edu.cn

