习题课

2021.6.10

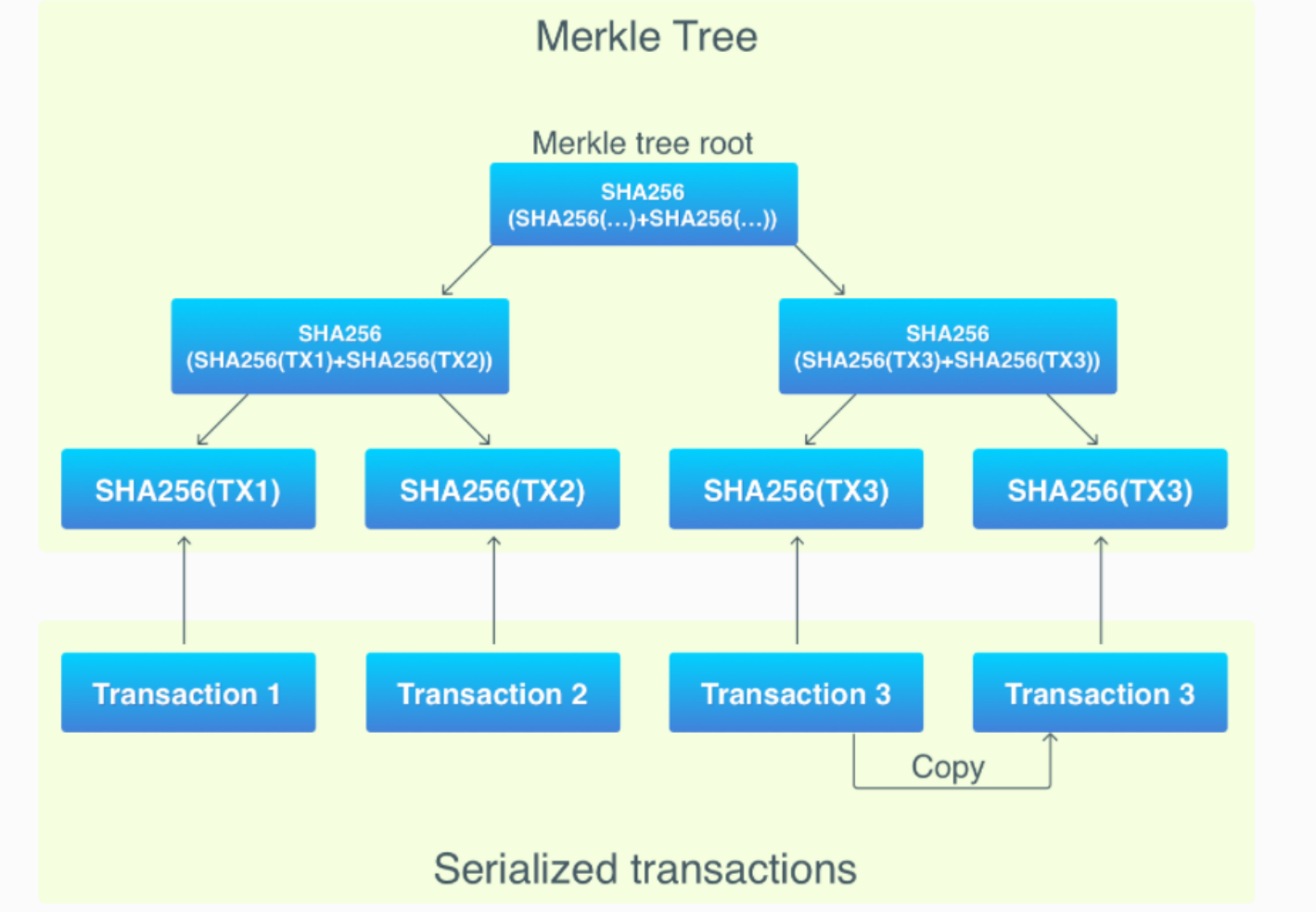
作业一

单击输入您的封面副标题

作业一

- 1.1、生成给定交易序列的二叉Merkle树。输入交易序列 ，长度为n，输出该交易序列所对应的Merkle树T。
- 1.2、生成某笔交易的Merkle树证明。输入Merkle树T，以及某笔交易的序号i，输出该交易所对应的SPV证明路径，长度为m。
- 1.3、验证SPV证明。输入Merkle树的根哈希值和SPV证明路径，长度为m，输出该SPV证明路径是否合法（true或false）。

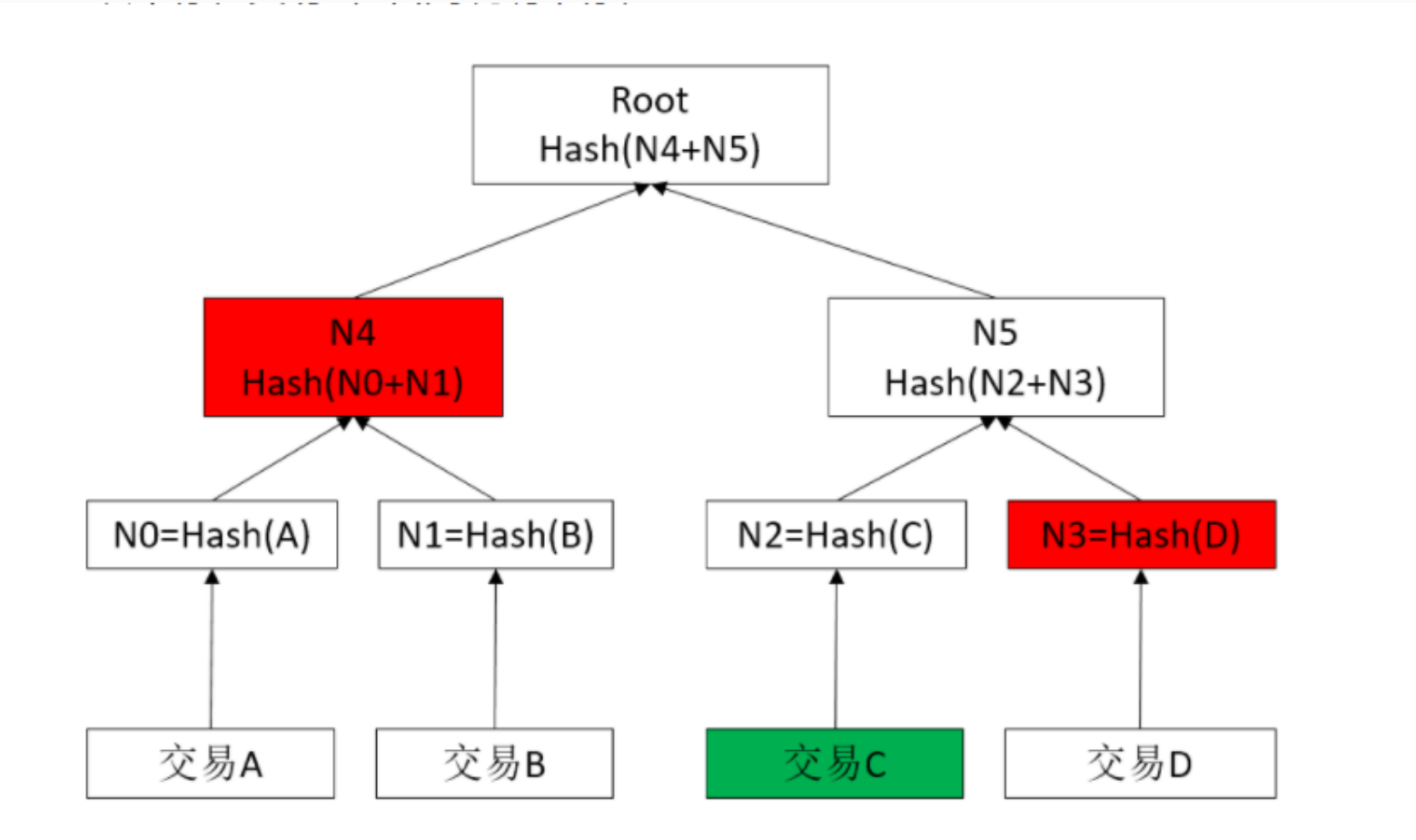
1.1 建立n笔交易的merkle树



参考代码

```
MerkleNode(Transaction)
MerkleNode(lchild,rchild)

CREATEMerkleTree(T)
    for i<- 0 to length(T)
        Nodes[i] = MerkleNode(T[i])
        //对应交易创建节点

    if length(Nodes)%2 != 0
        Nodes[length(Nodes)] = Nodes[length(Nodes)-1]

    while length(Nodes)>1

        if length(Nodes)%2 != 0
            Nodes[length(Nodes)] = Nodes[length(Nodes)-1]
        new_Nodes = []
        for i <- 0 to length(Nodes)
            new_Nodes.append(MerkleNode(Nodes[i],Nodes[i+1]))
        //左右子树生成Merkle节点
        Nodes = new_Nodes

    return Nodes[0]
```

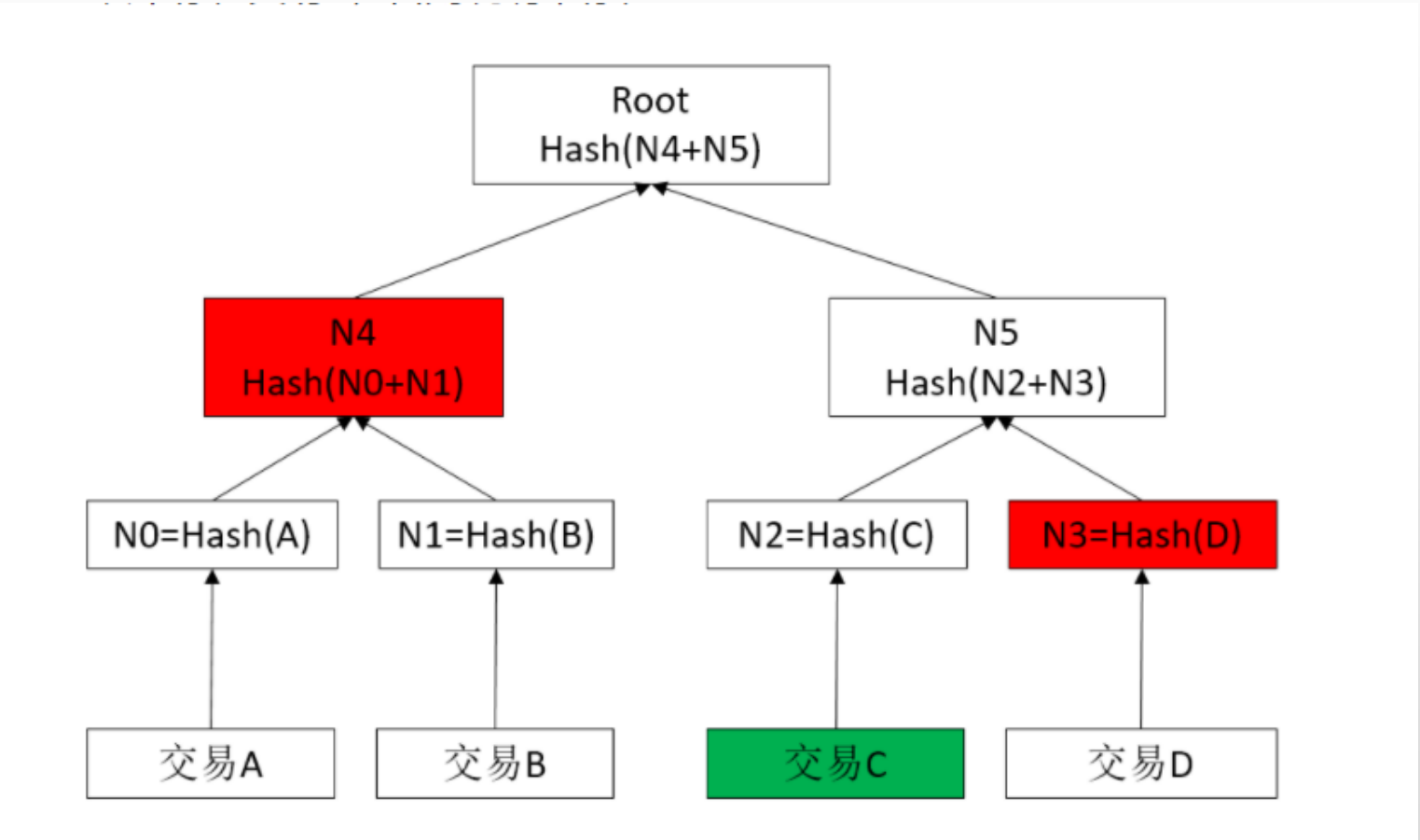
1.2 生成SPV路径


Root
Hash(N4+N5)
N4
Hash(N0+N1)
N5
Hash(N2+N3)
N0=Hash(A)
N1=Hash(B)
N2=Hash(C)
N3=Hash(D)
交易A
交易B
交易C
交易D

参考代码

```
FindNode(Transaction)

SPVPath(Tree,T)
    curr_node = FindNode(T)
    H = []

    H insert curr_node

    while curr_node != T.root
        H insert sibling(curr_node)
        curr_node = curr_node.p

    return H
```

1.3 判断SPV路径是否合法



参考代码

```
SPV_Vertify(H,T)
    curr_hash = H[0]

    for i <- 1 to length(H)

        if H[i] == H[i].p.left
            curr_hash = sha256(H[i]⊕curr_hash)
        else
            curr_hash = sha256(curr_hash⊕H[i])

    return curr_hash == T.root.data
```

作业一

- 简述比特币钱包公私钥生成原理，并给出比特币钱包地址生成的伪代码实现。（注：分别使用助记词和私钥。）

公钥加密与比特币地址

- 在比特币中，身份（identity）就是一对（或者多对）保存在电脑（或者你能够获取到的地方）上的公钥（public key）和私钥（private key）
- 所谓的比特币钱包地址，只不过是将公钥表示成人类可读的形式而已 。而比特币钱包本质上就是公私钥密钥对，他们通过公钥加密算法生成。在比特币中，谁拥有了私钥，谁就可以控制所有发送到这个公钥的币。也就是说私钥用来证明用户身份

助记词

- 比特币钱包应用很可能会为你生成一个助记词。助记词可以用来替代私钥，并且可以被用于生成私钥
- 助记词可以理解为私钥的另一种简单表示，最初是BIP39提案提出的，它有助于用户记住复杂的私钥(64位的哈希值)，并且具有与私钥相同的功能。记住64位随机数基本上是不可能的，因此助词有助于钱包用户有效地使用和支配自己的资产。助词一般由12、15、18、21个单词组成，这些单词都来自固定词库，生成顺序也是按照一定的算法生成的，所以用户没必要担忧随随便便地输入 12个单词，就会生成一个地址。
- 助记词很重要，因为能通过助记词找到私钥，还能恢复钱包。目前，大部分钱包都需要备份助记词，以恢复钱包。

公钥生成--椭圆曲线加密

- 公钥和私钥是随机的字节序列。私钥能够用于证明持币人的身份，需要有一个条件：随机算法必须生成真正随机的字节。因为没有人会想要生成一个私钥，而这个私钥意外地也被别人所有。
- 比特币使用的是 ECDSA（Elliptic Curve Digital Signature Algorithm）算法来对交易进行签名

比特币钱包公私钥生成原理

- 利用助记词生成私钥：生成一个128位随机数 → 加上对随机数做的校验4位，得到一个132位的数 → 每11位切分，得到12个二进制数 → 查BIP39定义的单词表，得到12个助记词 → 使用密钥拉伸函数，生成256bits私钥
- 利用公钥生成私钥：将私钥经过 ECDSA 算法得到，对应为椭圆曲线上一点的 X值和Y值

具体步骤

1.生成随机私钥

2.椭圆曲线算公钥

3.计算公钥地SHA-256哈希值

4.计算RIPEMD160哈希值 (唯一性哈希，160位节省空间，防止地址冲突)

5.加入版本号

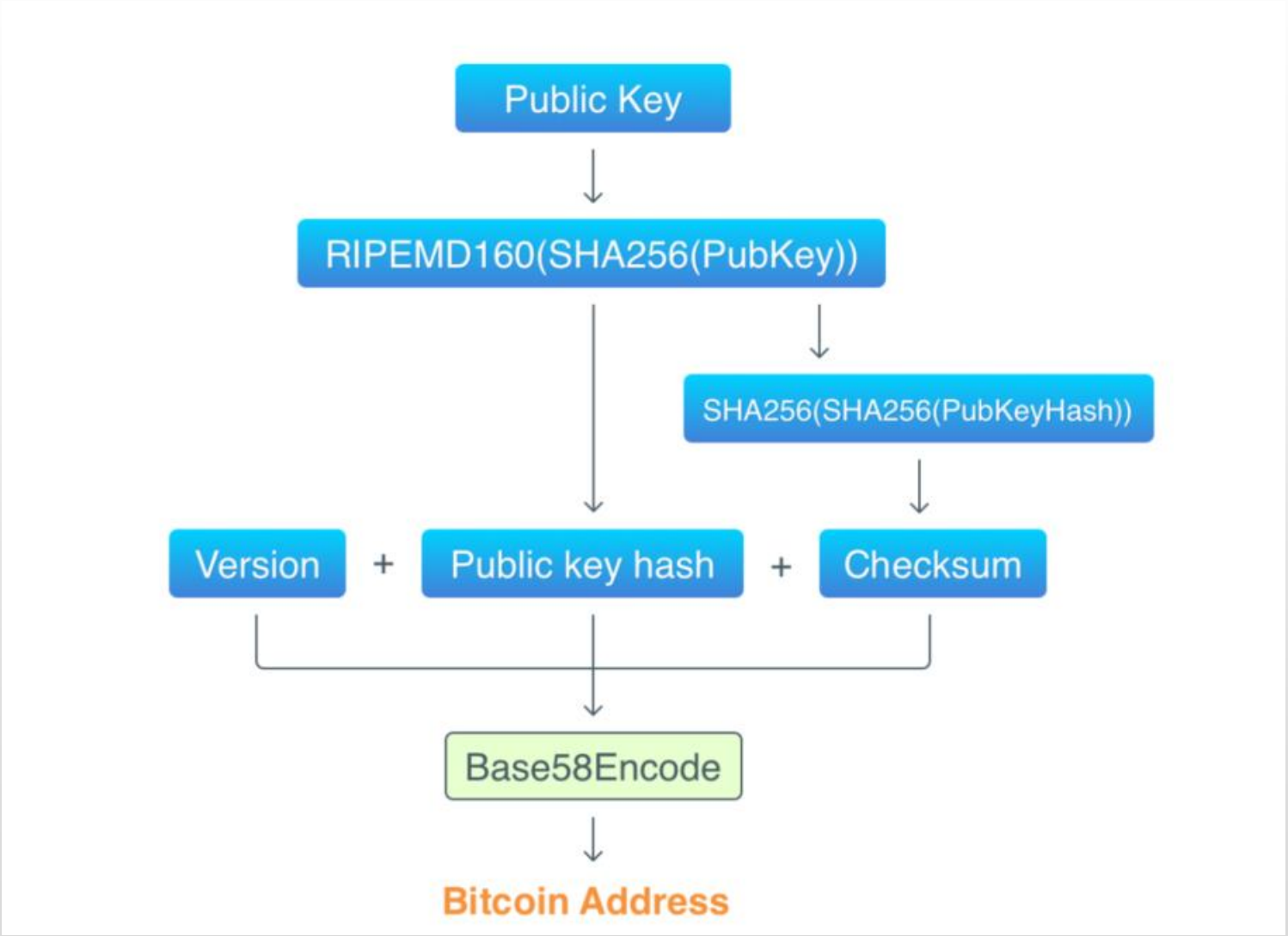
6.计算SHA256哈希值

7.获取校验和

8.合并5，7的结果

9.Base58变换地址

比特币钱包地址生成示意图

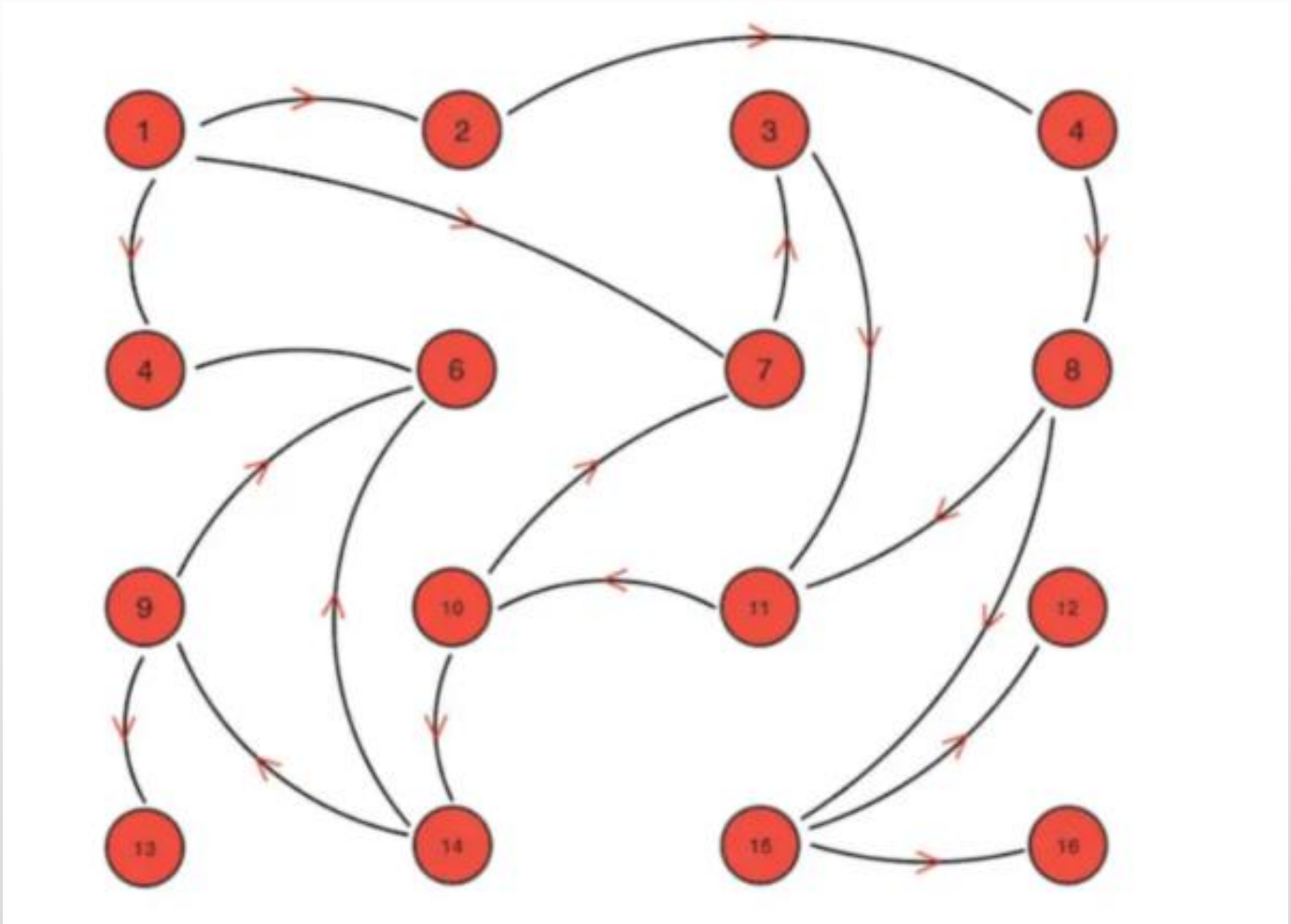


作业一

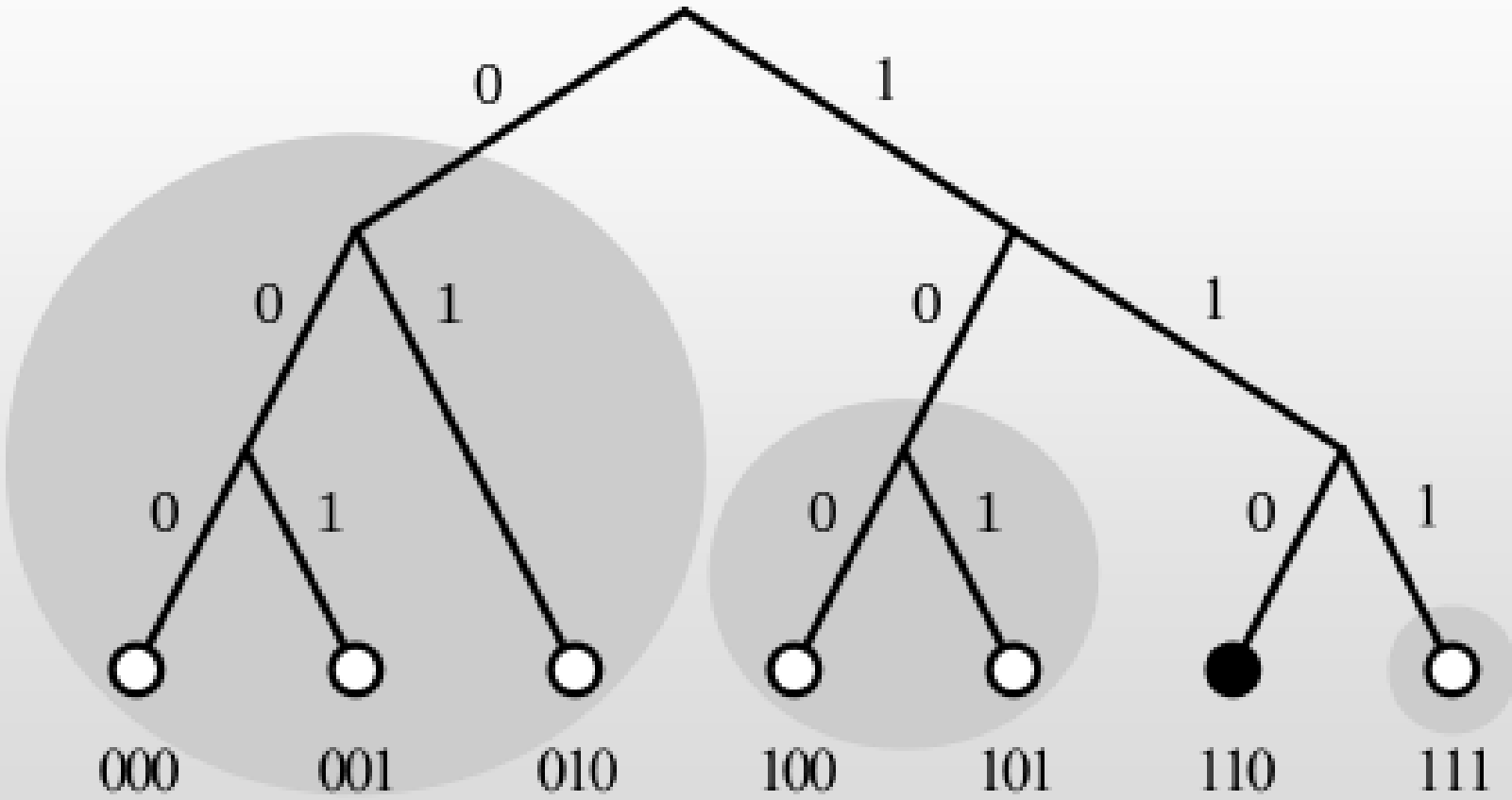
- 简述区块链 P2P 网络协议中Gossip（比特币）和Kademlia（以太坊）原理，给出相关示意图，并加以比较。

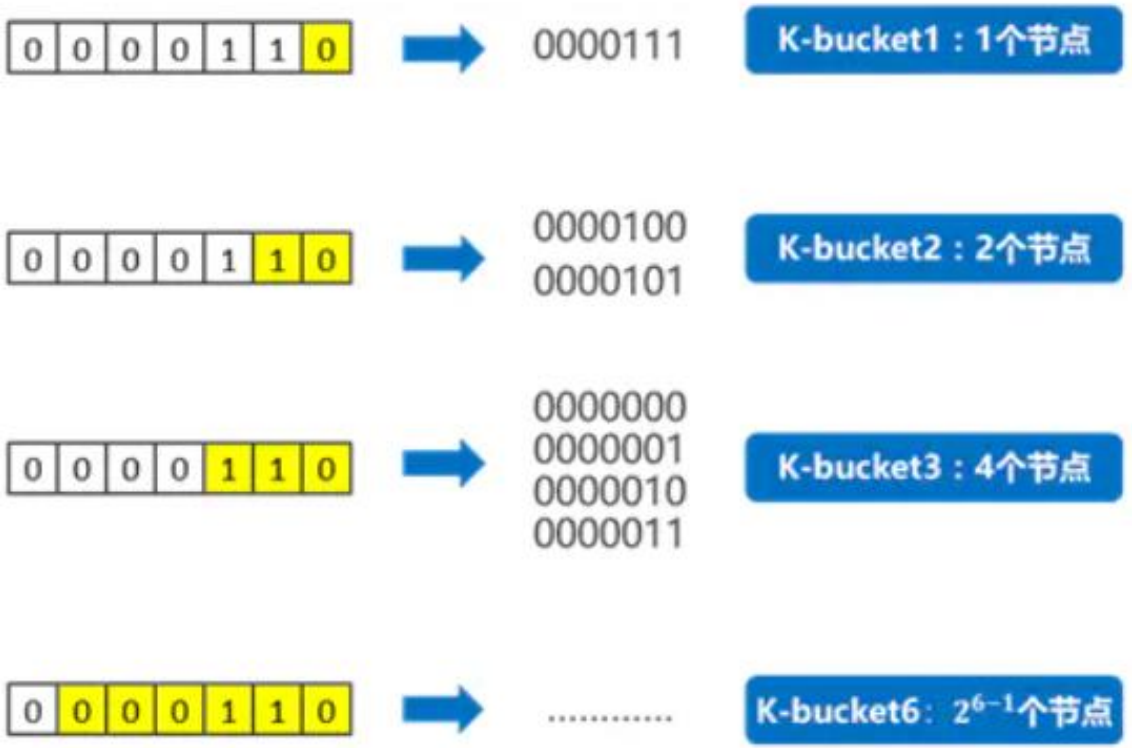
Gossip

- （1）Gossip 是周期性的散播消息，把周期限定为 1 秒
- （2）被感染节点随机选择 k 个邻接节点（fan-out）散播消息
- （3）每次散播消息都选择尚未发送过的节点进行散播
- （4）收到消息的节点不再往发送节点散播，比如 A -> B，那么 B 进行散播的时候，不再发给 A。
- 这里一共有 16 个节点，节点 1 为初始被感染节点，通过 Gossip 过程，最终所有节点都被感染：

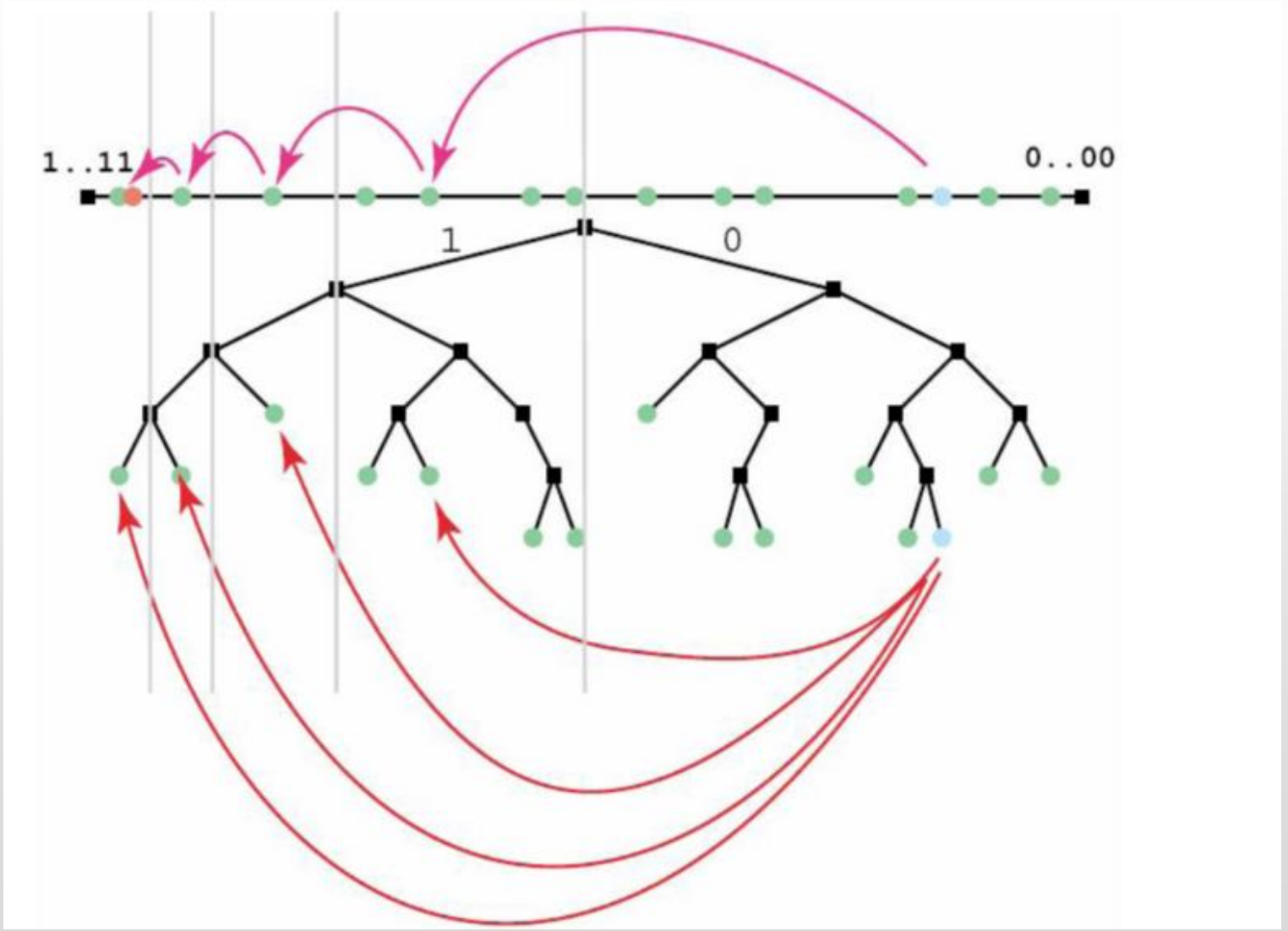

1
2
3
4
4
6
7
8
9
10
11
12
13
14
15
16

Kademlia



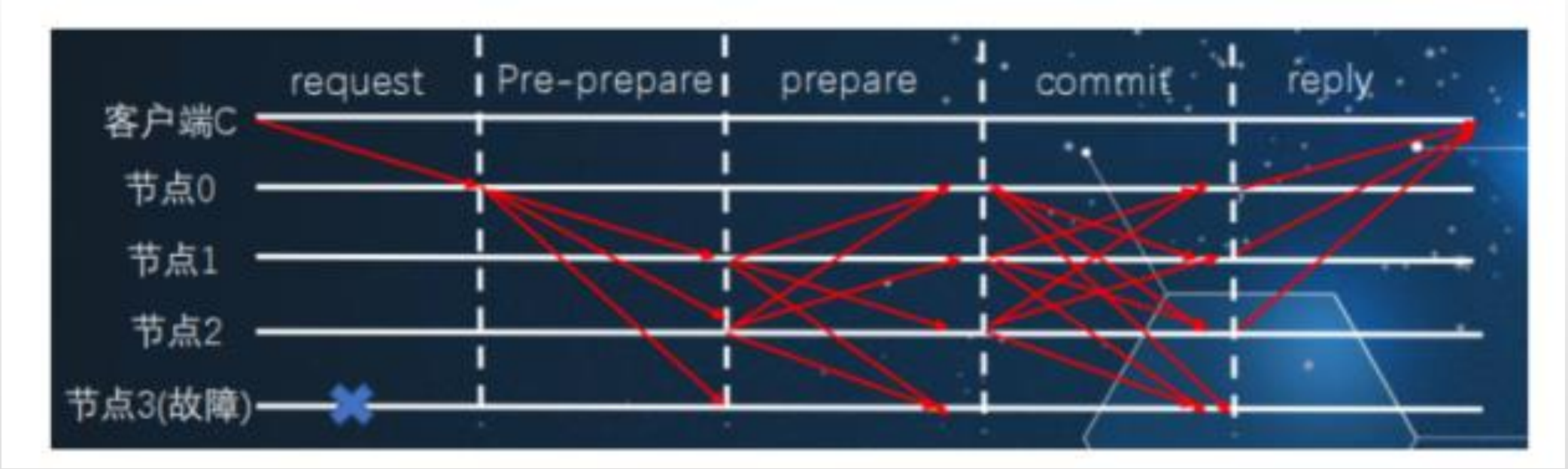


按位数区分k-bucket


1..11
0..00
1
0

区别

- 使用Gossip协议的比特币主网的 P2P 网络是无结构的，基于网络路由器以随机熵扩散的形式传播，但以太坊的 P2P 网络是有结构的，将整个网络拓扑组织成如一个二叉前缀树，每个 NodeID 会映射到二叉树上的某个叶子。
- 比特币网络的结构明显容易理解，实现起来也相对容易得多，而以太坊网络引入了异或距离、二叉前缀树、K-桶等，结构上复杂不少，但在节点路由上的确会比比特币快很多。
- 这种查找算法相比 gossip 减少了信息在网络中的泛滥传播，能更好地抵御 DoS


request
Pre-prepare
prepare
commit
reply
客户端C
节点0
节点1
节点2
节点3(故障)

实验一

```
type Block struct {
        Timestamp     int64  // 时间戳
        Data          [][]byte //数据
        PrevBlockHash []byte //前一个区块对应哈希
        Hash          []byte //当前区块数据对应哈希值
        Nonce         int //随机数
}
```

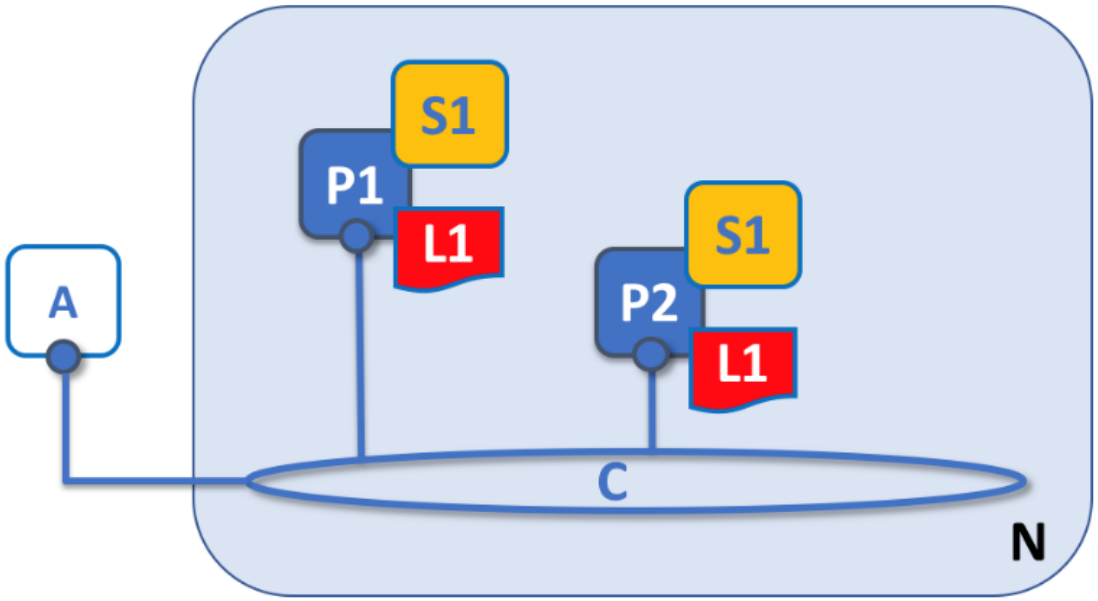
区块链操作

```
err = db.Update(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(blocksBucket))

        if b == nil {
                fmt.Println("No existing blockchain found. Creating a new one...")
                genesis := NewGenesisBlock()

                b, err := tx.CreateBucket([]byte(blocksBucket))
                if err != nil {
                        log.Panic(err)
                }

                err = b.Put(genesis.Hash, genesis.Serialize())
                if err != nil {
                        log.Panic(err)
                }

                err = b.Put([]byte("l"), genesis.Hash)
                if err != nil {
                        log.Panic(err)
                }
                tip = genesis.Hash
        } else {
                tip = b.Get([]byte("l"))
        }

        return nil
})
```

实验二

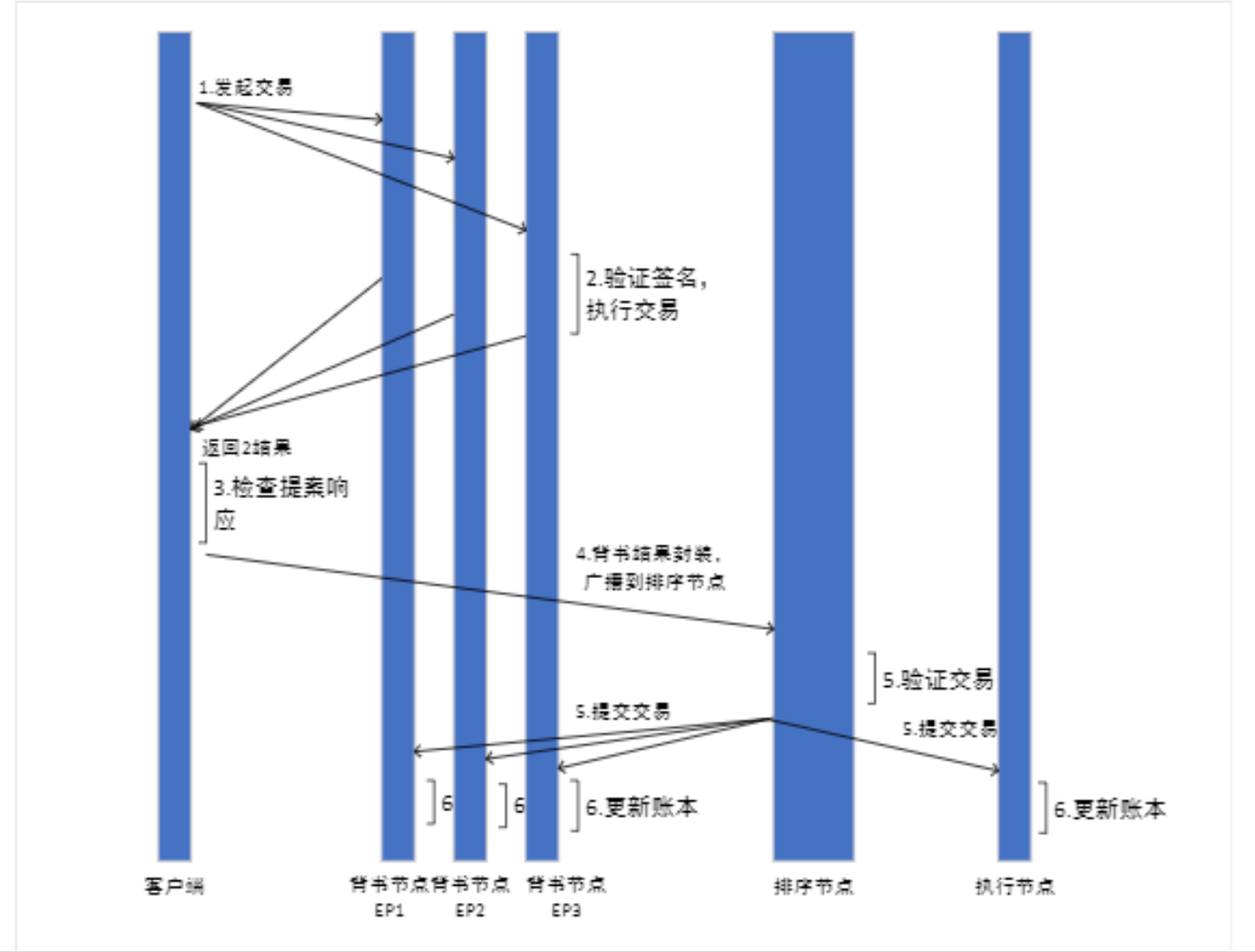
- 从区块链中获取一些公开的数据，对应本次实验我们需要获取上一个区块哈希值(32位)，当前区块数据对应哈希（32位），时间戳，区块难度，随机数。对应数据直接进行合并的操作来进行合并。
- 添加计数器，作为随机数。计算器从0开始基础，每个回合+1
- 对于上述的数据来进行一个哈希的操作。
- 判断结果是否满足计算的条件：
- 如果符合，则得到了满足结果。
- 如果没有符合，从2开始重新直接2、3、4步骤。

fabric上链码部署相关的逻辑



N	Blockchain Network	L	Ledger
C	Channel	A	Application
P	Peer	PA C	Principal PA (e.g. A, P1) communicates via channel C.
S	Chaincode		

交易流程


1.发起交易
2.验证签名，
执行交易
返回2结果
3.检查提案响
应
4.背书结果封装，
广播到排序节点
5.验证交易
5.提交交易
5.提交交易
6
6
6.更新账本
6.更新账本
客户端
背书节点
EP1
背书节点
EP2
背书节点
EP3
排序节点
执行节点

链码安装主要逻辑

- 1.打包链码
- 2.安装链码
- 3.定义链码（倍数策略等）
- 4.调用链码