企业级联盟链技术平台

宣章炯

介绍

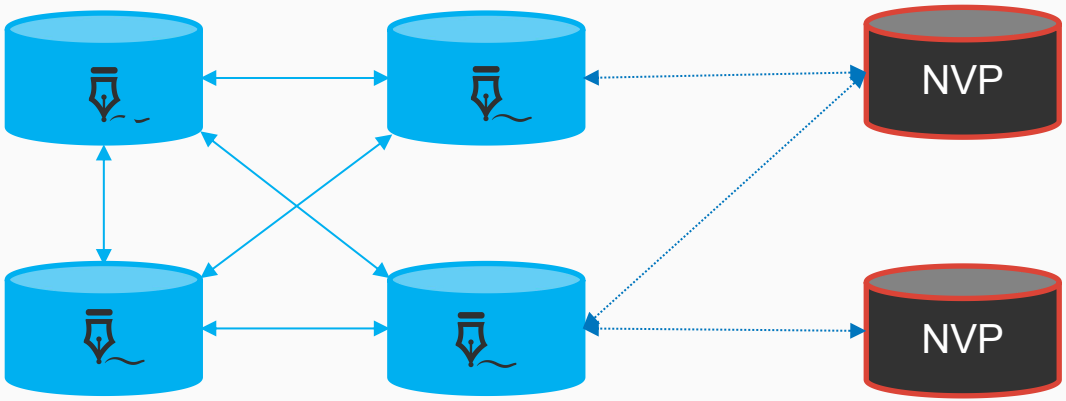
企业级区块链（也称联盟链）主要针对大型公司、政府机构和产业联盟的区块链技术需求，提供企业级的区块链网络解决方案。联盟链的各个节点通常对应一个实体的机构组织，节点的加入和退出需要经过授权。各个机构组成利益相关的联盟，共同维护区块链网络的健康运转。

与私有链和公有链不同，企业级区块链更加着眼于区块链技术的实际落地，在区块链的性能速度和安全性、隐私性保护上有着更高的要求。除此之外，企业级区块链的研发往往直接和实际业务场景相关联，更加贴近行业痛点，为企业联盟提供一套更加完善的一体化区块链解决方案。

目录

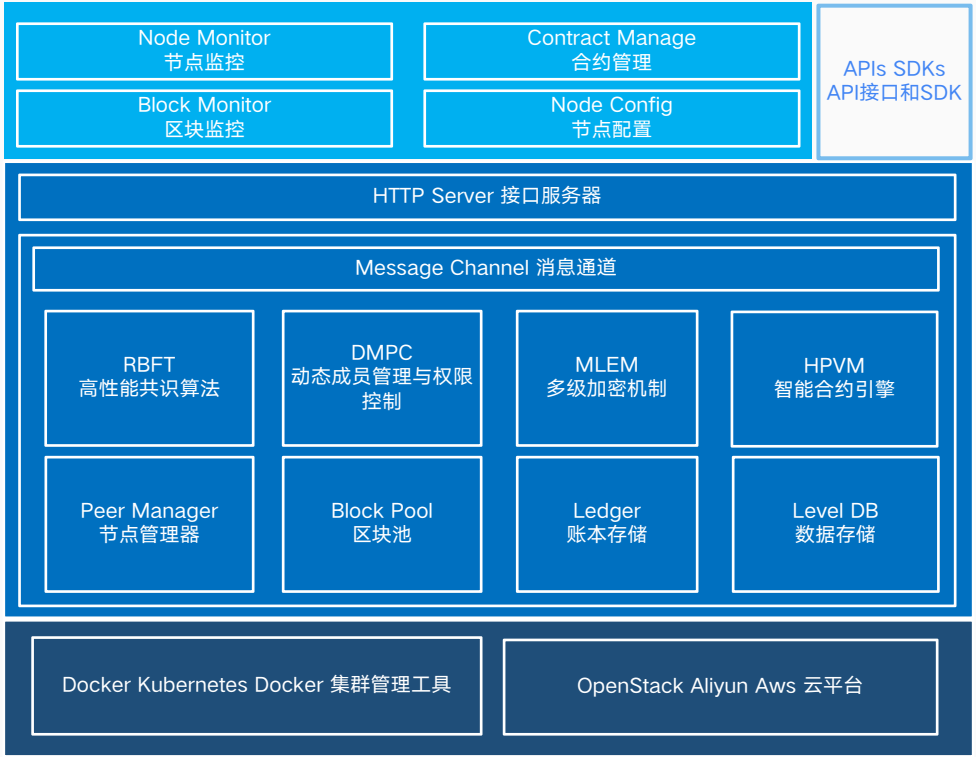
联 盟 链 系 统 整 体 架 构

平台物理集群架构



VP(Validate Peer) + NVP (Non Validate Peer)

平台逻辑架构



企业级管控平台

- web控制平台
- 区块链运行信息监控
- 合约开发者平台

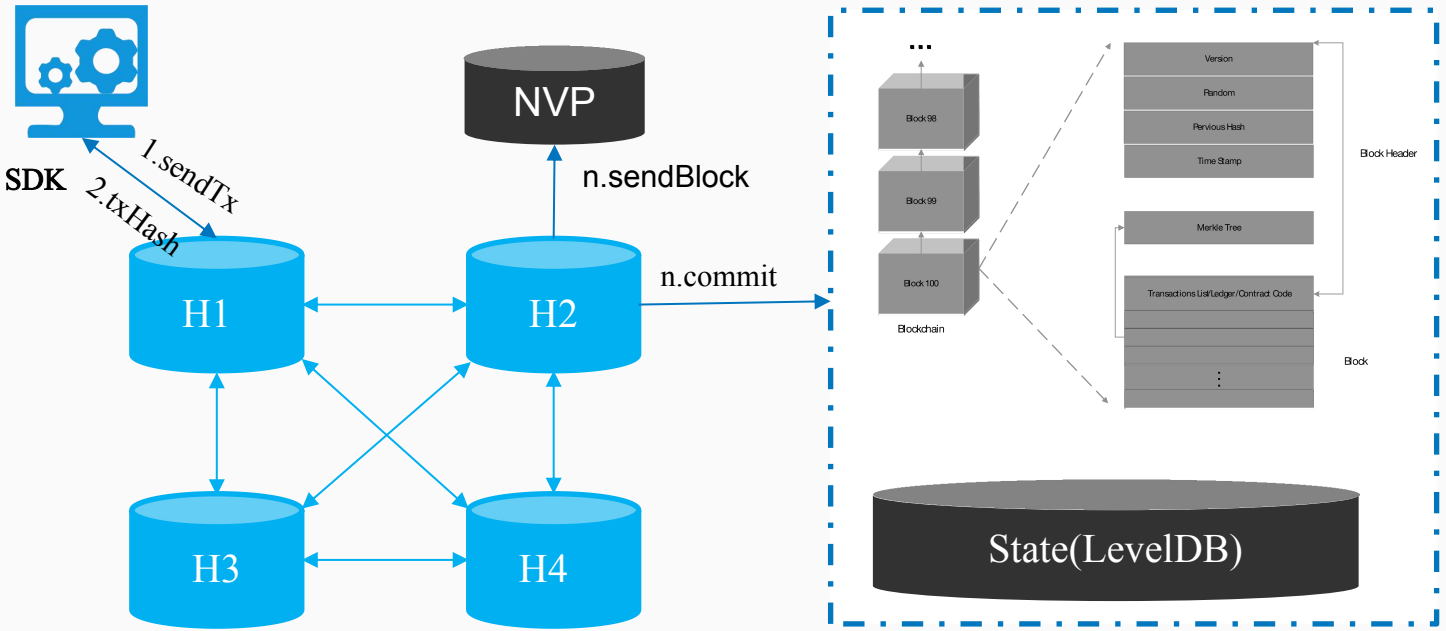
联盟链核心模块

- 高鲁棒性共识算法
- 高性能智能合约引擎
- 多级加密安全体系

系统支撑技术

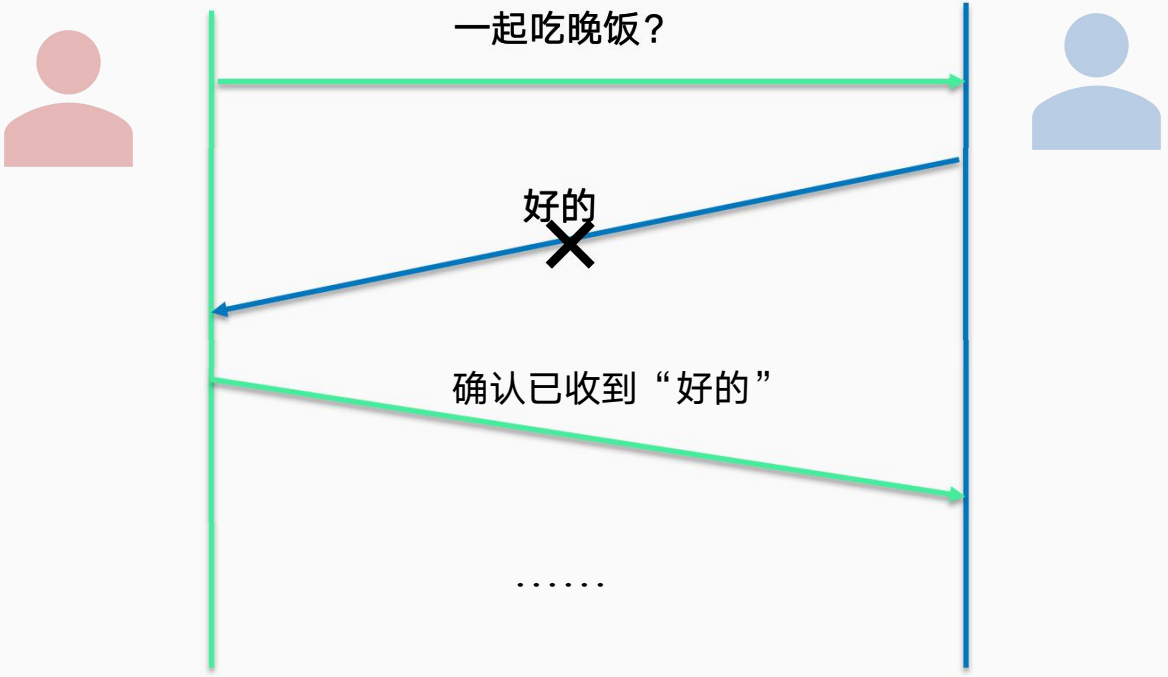
- 支持物理机和云上部署
- 支持Docker化部署

Hyperchain交易处理流程



共　　识　　算　　法

什么是共识?



共识相关概念

节点（Node）：系统中一个的工作单元

消息模型：

- 消息传递（Message Passing）
- 消息丢失（Message Loss）
- 可变消息延迟（Variable Message Delay）

状态复制

共识定义

系统中有n个节点，其中最多有f个节点可能崩溃，也就是说，最少有n-f个节点是好的。节点i从一个输入值v_i开始。所有节点必须要从输入值中选择一个值（决策值），并且满足下面的条件：

■一致性（Agreement）：所有好的节点决策值必定相同。

■可终止性（Termination）：所有好节点在有限的时间内结束决策过程。

■有效性（Validity）：选择出的决策值必须是某个节点的输入值。

两阶段协议

阶段一：
1：客户端向服务器请求锁
阶段二：
2：if 客户端成功获得了所有服务器的锁：
3：　　客户端以可靠方式向每个服务器发送指令，随即释放锁
4：else
5：该客户端释放已获得的锁
6：该客户端等待一段时间，重新进入阶段一

Paxos

阶段一

（1）Proposer选择一个提案编号N，然后向半数以上的Acceptor发送编号为N的Prepare请求

（2）如果一个Acceptor收到一个编号为N的Prepare请求，且N大于该Acceptor已经响应过的所有Prepare请求的编号，那么它就会将它已经批准过的编号最大的提案作为响应反馈给Proposer，同时该Acceptor承诺不再接受任何编号小于N的提案。

阶段二

（1）如果Proposer收到来自半数以上Acceptor对其发出的编号为N的Prepare请求的响应，那么它就会发送一个针对[N,V]提案的Accept请求给半数以上的Acceptor。注意：V就是收到的响应中编号最大的提案的value，如果响应中不包含任何提案，那么V就由Proposer自己决定。

（2）如果Acceptor收到一个针对编号为N的提案的Accept请求，只要该Acceptor没有对编号大于N的Prepare请求做出过响应，它就接受该提案。

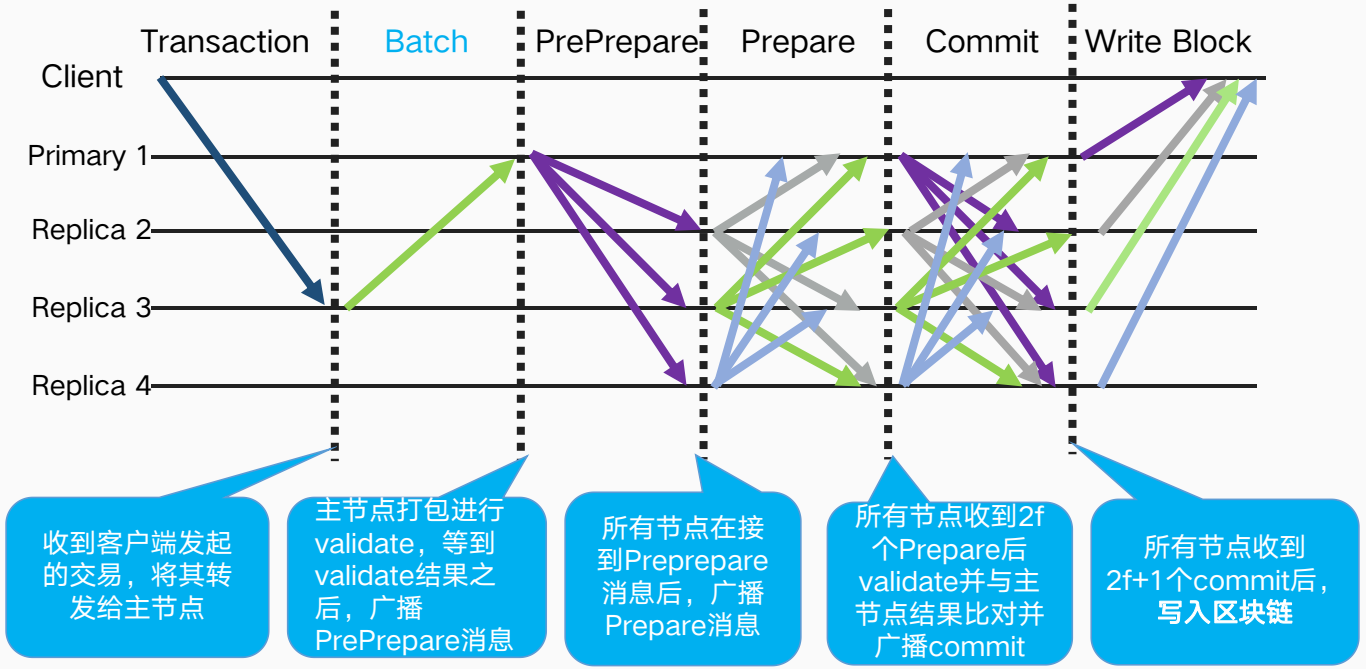
拜占庭将军问题

拜占庭问题最早由 Leslie Lamport 等学者于 1982 年在论文《The Byzantine Generals Problem》中正式提出：

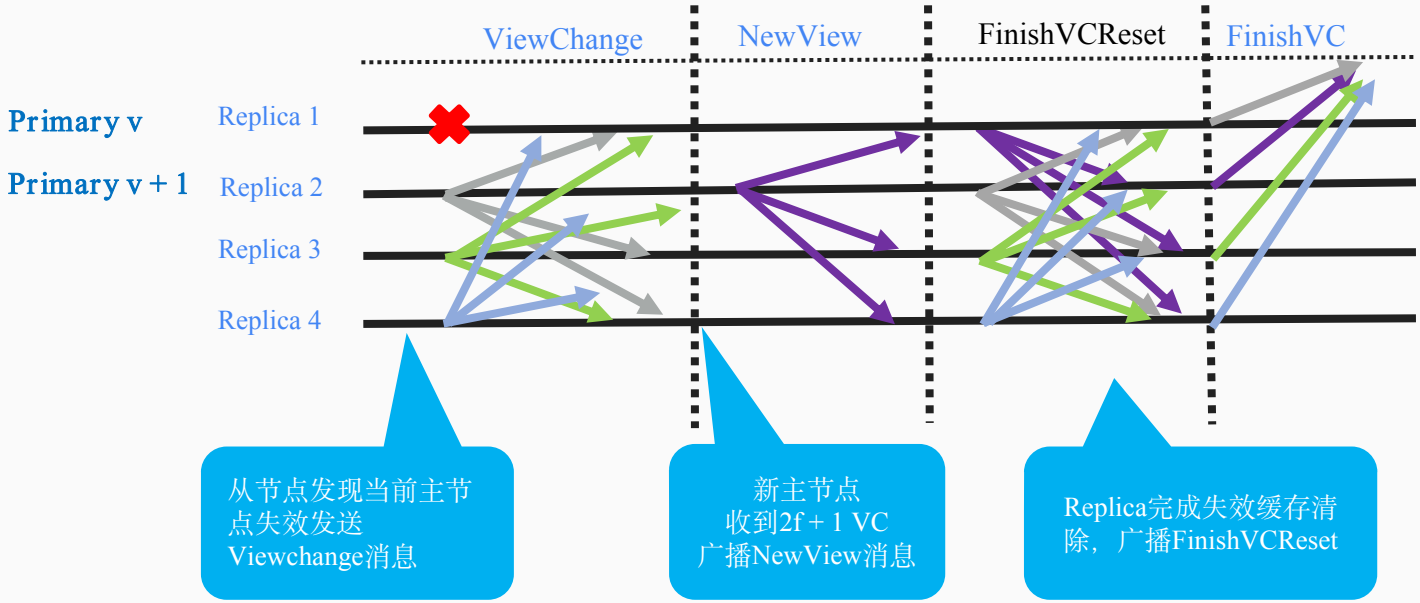
一组拜占庭将军分别各率领一支军队共同围困一座城市。为了简化问题，将各支军队的行动策略限定为进攻或撤离两种。因为部分军队进攻部分军队撤离可能会造成灾难性后果，因此各位将军必须通过投票来达成一致策略，即所有军队一起进攻或所有军队一起撤离。因为各位将军分处城市不同方向，他们只能通过信使互相联系。在投票过程中每位将军都将自己投票给进攻还是撤退的信息通过信使分别通知其他所有将军，这样一来每位将军根据自己的投票和其他所有将军送来的信息就可以知道共同的投票结果而决定行动策略。

忠诚的将军能通过多数决定来决定他们的战略。

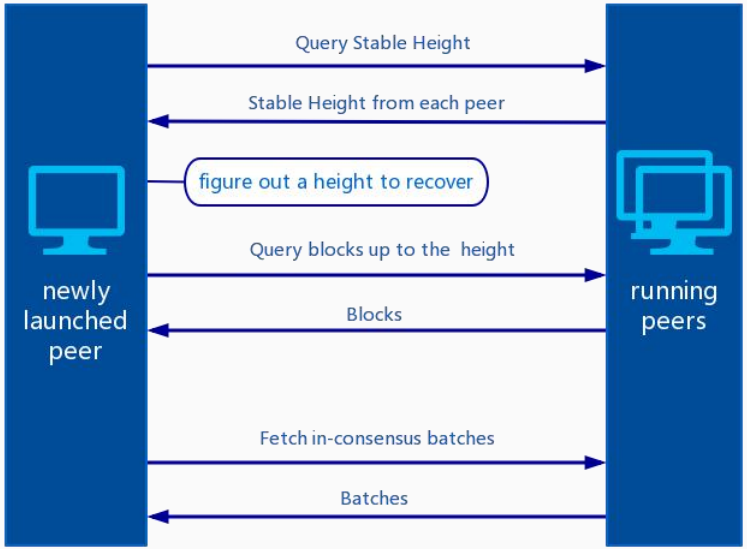
RBFT常规流程


Transaction
Batch
PrePrepare
Prepare
Commit
Write Block
Client
Primary 1
Replica 2
Replica 3
Replica 4
收到客户端发起的交易，将其转发给主节点
主节点打包进行validate，等到validate结果之后，广播PrePrepare消息
所有节点在接到Preprepare消息后，广播Prepare消息
所有节点收到2f个Prepare后validate并与主节点结果比对并广播commit
所有节点收到2f+1个commit后，写入区块链

RBFT视图变更

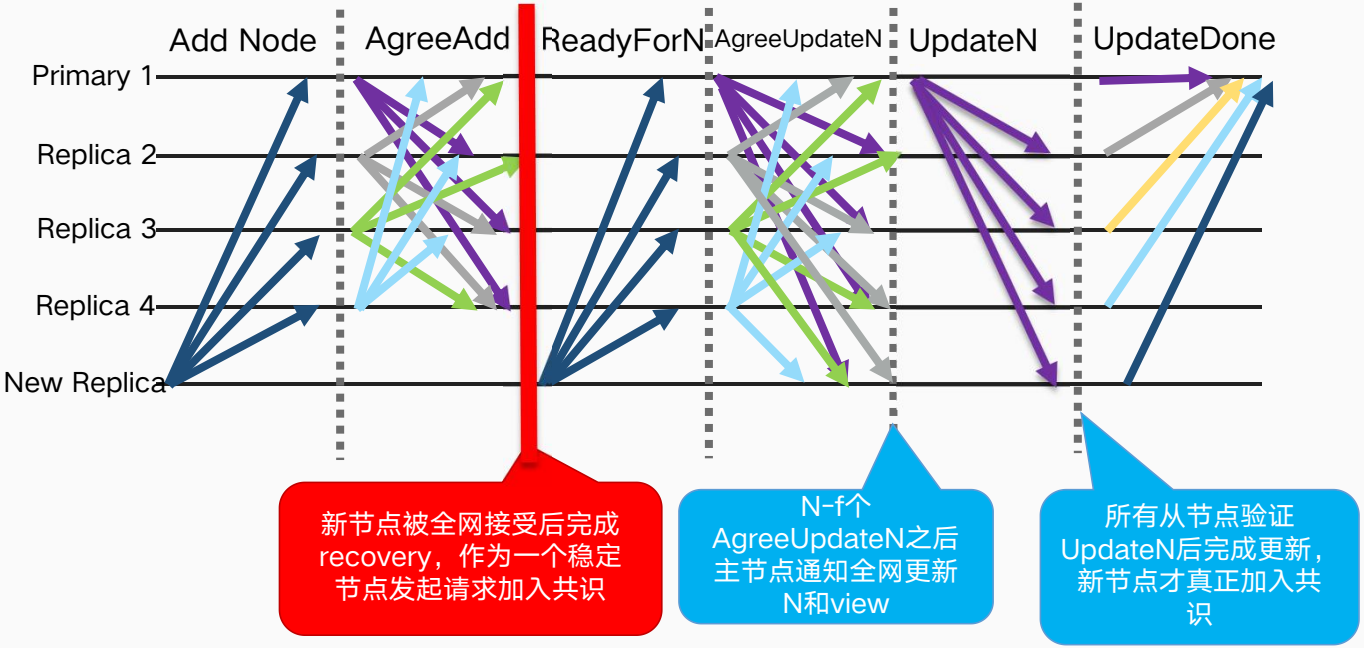

ViewChange
NewView
FinishVCReset
FinishVC
Primary v
Primary v + 1
Replica 1
Replica 2
Replica 3
Replica 4
从节点发现当前主节点失效发送Viewchange消息
新主节点收到2f + 1 VC广播NewView消息
Replica完成失效缓存清除，广播FinishVCReset

RBFT崩溃恢复机制



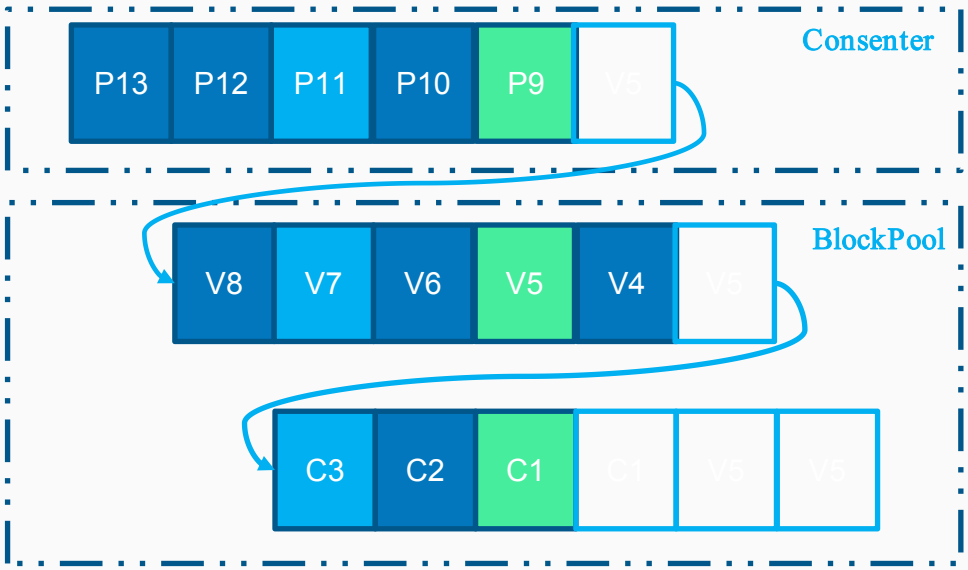
- ✓ **节点重启时**，recovery机制能自动检测节点并自主更新自身状态，recovery同时也是动态增删的基础
- ✓ **当一个节点发生ViewChange而无响应时（意味着拜占庭行为）**，recovery机制可以进行自我恢复
- ✓ **Recovery机制的存在大大地增强了共识模块的可用性与鲁棒性**

RBFT成员准入


Add Node
AgreeAdd
ReadyForN
AgreeUpdateN
UpdateN
UpdateDone
Primary 1
Replica 2
Replica 3
Replica 4
New Replica
新节点被全网接受后完成recovery，作为一个稳定节点发起请求加入共识
N-f个AgreeUpdateN之后主节点通知全网更新N和view
所有从节点验证UpdateN后完成更新，新节点才真正加入共识

合 约 执 行 引 擎

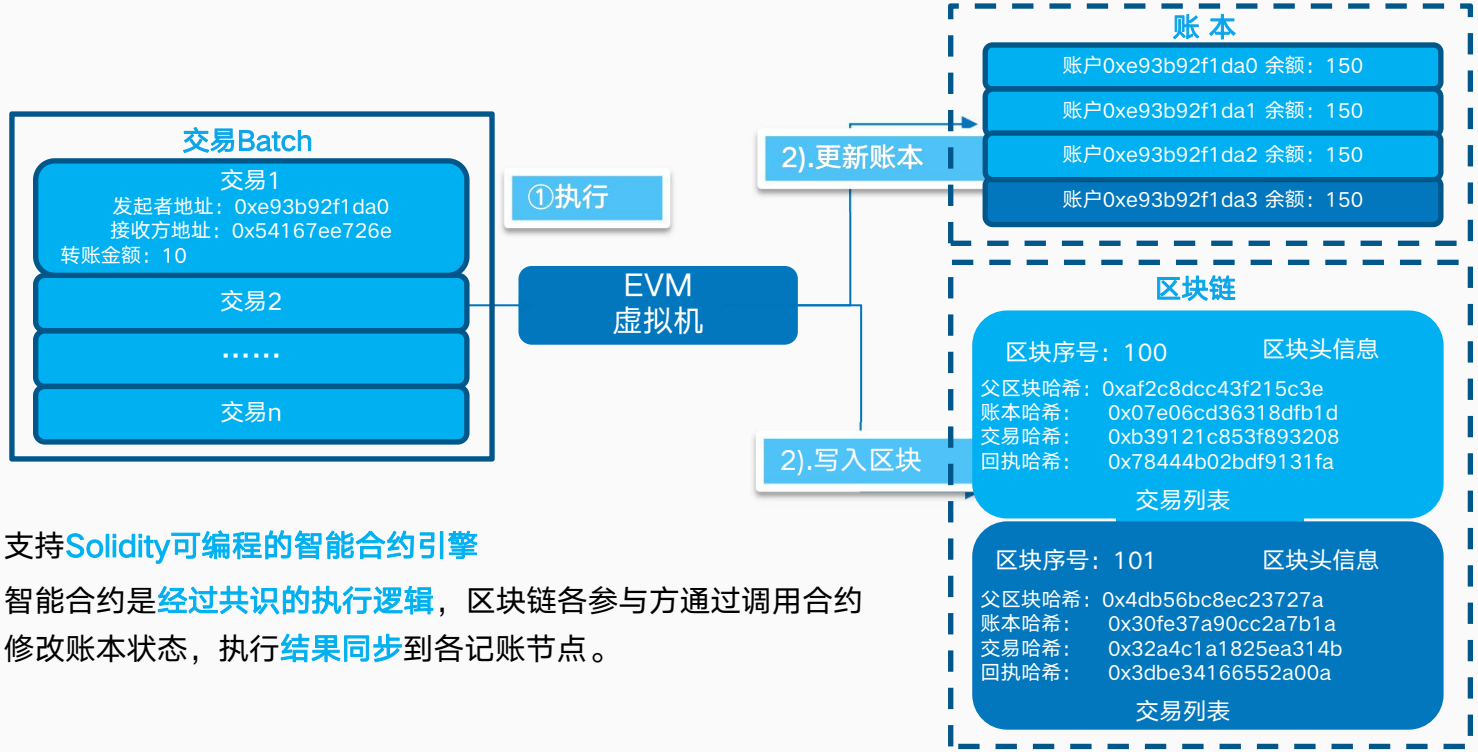
执行引擎调度框架



基于多级流水线的调度执行框架

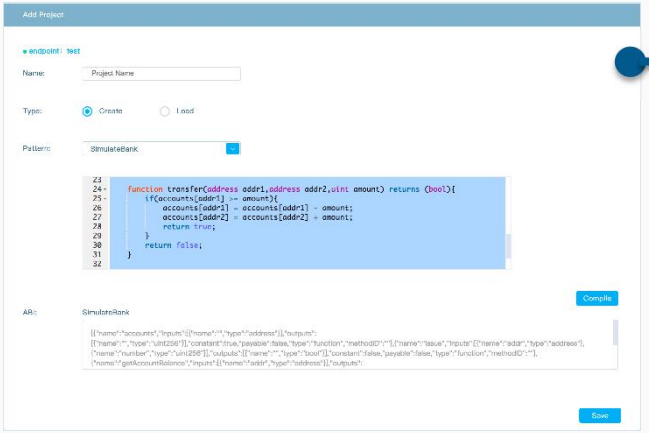
- 共识队列P系列
- 执行队列V系列
- 存储队列C系列

执行引擎合约引擎



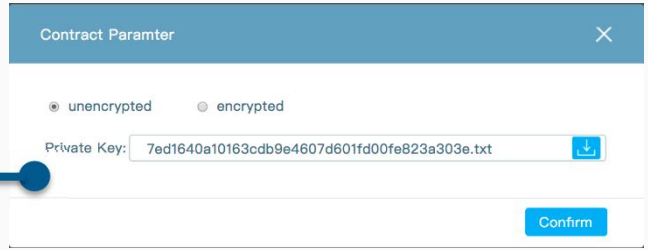
- 支持Solidity可编程的智能合约引擎
- 智能合约是经过共识的执行逻辑，区块链各参与方通过调用合约修改账本状态，执行结果同步到各记账节点。

合约生命周期

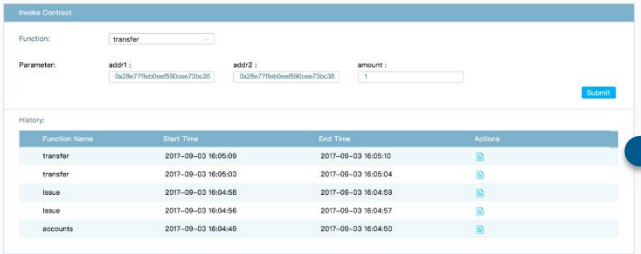


创建或升级合约

合约部署



加载已部署合约



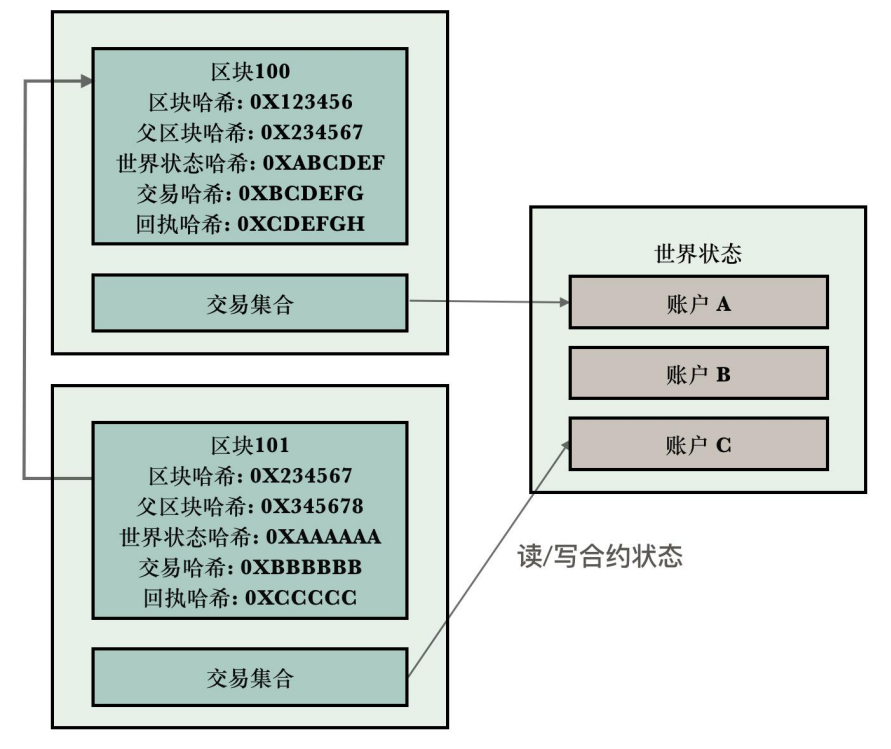
合约调用

账　　本　　存　　储

账本设计

- UTXO模型
  - 以资产为核心建模
  - 以锁脚本记录所有者
  - Bitcoin
- 账户模型
  - 以所有者为核心建模
  - 以账户余额记录资产数量
  - Ethereum/Fabric/Hyperchain

账户体系



特点

1 区块链中保存交易流水

区块以链式结构进行存储，每个区块中包含一批交易

2 世界状态中保存账户状态

世界状态中保存所有账户的状态数据

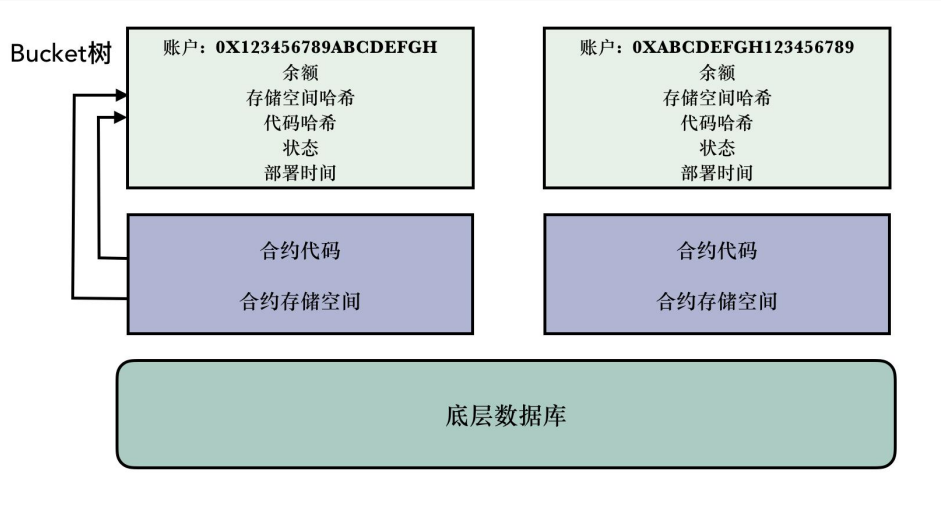
账户可以分为：合约账户及非合约账户

3 交易执行时改变账户状态

交易在虚拟机中执行的过程中，可以读写合约的状态变量

每执行一笔交易，即意味着世界状态进行了一次状态变更

账户体系



特点

1 账户对象模型

采用了与比特币有别的对象模型来描述账户

2 bucket树进行存储空间的哈希计算

合约的状态变量存储在存储空间中

利用bucket树对状态变量进行哈希计算

3 底层KV数据库

账户的所有相关数据，包括账户元数据，状态变量，bucket树数据，都持久化在底层的KV数据库中

数据类型

1 静态类型

- bool
- bytes(1-32), byte是bytes1的别名
- uint/int(8-256), uint/int分别是uint256和int256的别名
- address(20 bytes)
- ……

2 动态类型

- map
- 动态长度数组(string, bytes等)

数据类型-数组

1 定长数组

```
uint [5] A = [1,2,3,4,5];
function modifyArray() public {
    A[0] = 10;
    A.push(6);
    A.length = 10
}
```

定长数组不可改变长度，也没有push操作。但是可以通过索引值去修改里面的值。

2 不定长数组

不定长数组可以改变长度，可以进行push，也可以通过索引去修改里面的值。

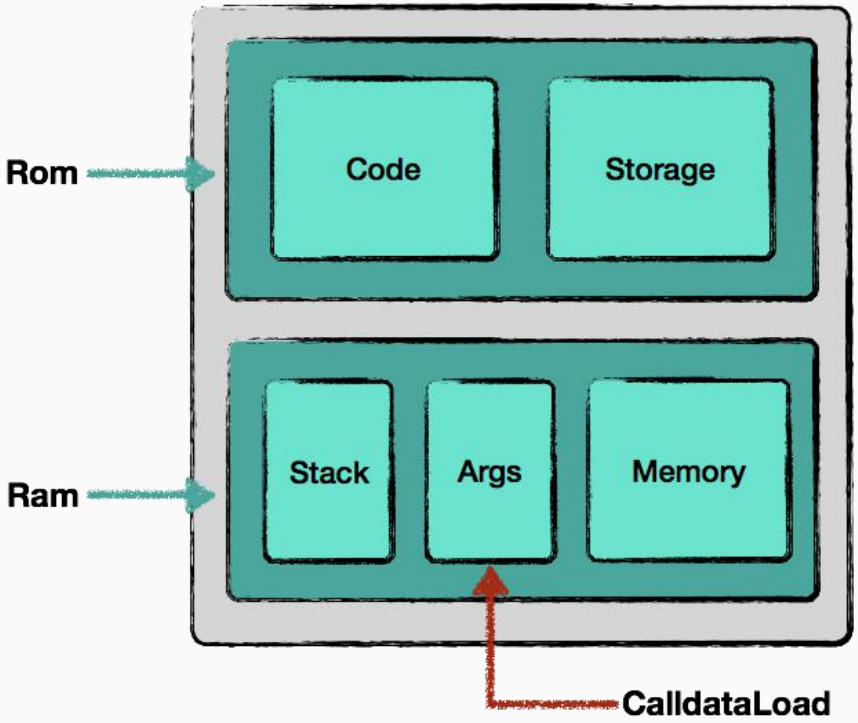
数据类型-Map

- 无法直接从map变量中获取长度值。
- Key为空时，返回0x0，不报错。
- 无法直接遍历和返回。
- 删除map中的某一个元素直接置为0x0

合约存储结构-问题

- map为什么不能获取长度和直接遍历，为什么不能直接返回？
- 升级合约，为什么变量需要定义在已有变量后面？

合约存储结构



合约存储结构-Storage结构

0x000000...000	slot(32 bytes)
0x000000...001	slot(32 bytes)
0x000000...002	slot(32 bytes)

- Storage中每一条数据的Key和Value大小都是32字节。
- 在Storage中取数据的时候，都是以一个slot为单位完成操作。

问题：

•那一个变量在Storage中存储的索引值是怎么计算的？

•变量本身是怎么编码的？

合约存储结构-变量索引规则

- 变量索引的生成是发生在合约的编译阶段。编译器根据变量的类型分别应用不同的规则生成索引。
- solidity中变量有两个大类型，分别是静态变量和动态变量，这两种类型变量的索引规则也是不同的。

合约存储结构 – 静态变量

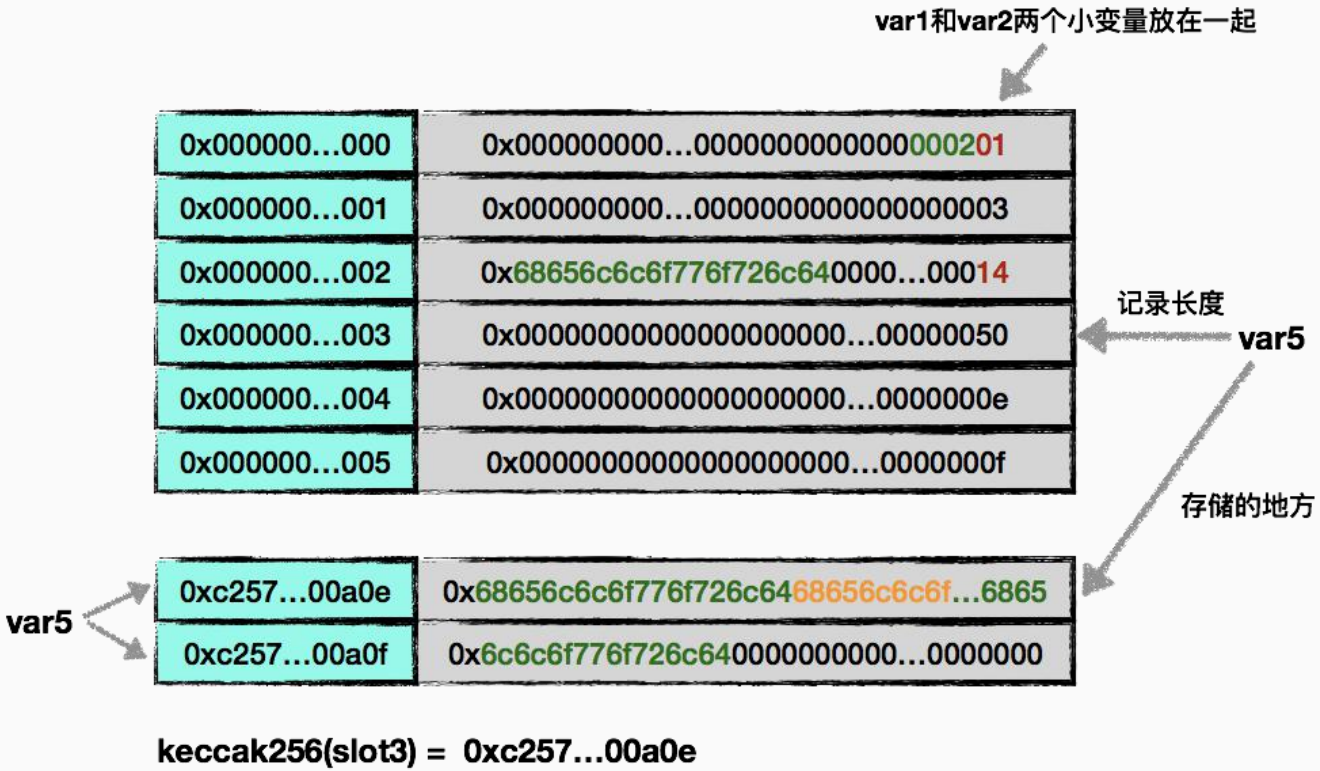
1. 静态变量就是变量大小固定的变量。

2. 所有静态变量被连续地存放在Storage中，存放的顺序和变量在合约中定义的顺序一致。

3. 一些比较小的静态变量，会被压缩存储到一个slot中。

4. 静态变量采用右端对齐进行存储。

合约存储结构 – 静态变量例子



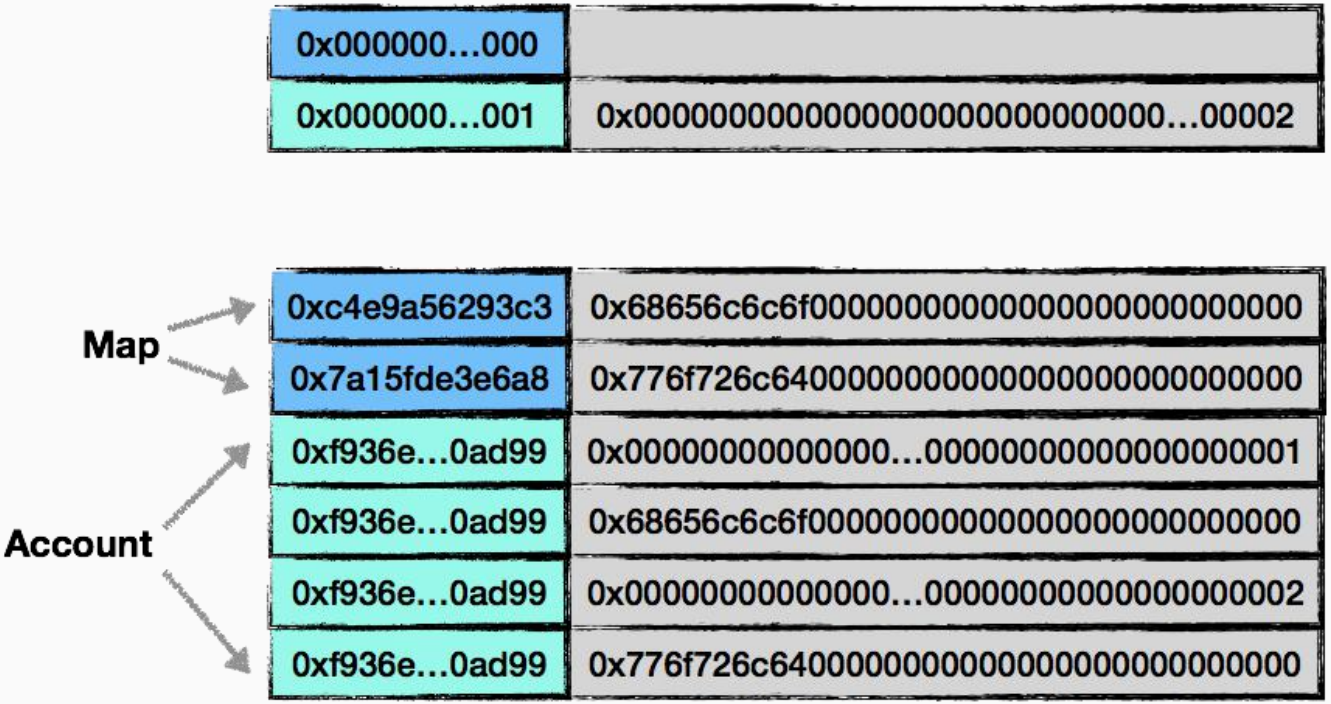
合约存储结构 – 动态变量

- 动态变量主要有两种：map和动态长度数组(string，bytes等)
- Storage中会为map变量本身分配一个slot，不过这个slot不存储任何数据，而map中的数据被散列存储在Storage中。
- 由于无法确认map将要存储的数据大小，solidity采用哈希算法来计算map中每一条数据在Storage中的存储索引值。

合约存储结构 – 动态变量例子

```
contract Dynamic {

        struct Account {
                uint8 id;
                bytes32 name;
        }

        mapping(uint => bytes32) var1;
        Account[] account;

        function Dynamic() {
                var1[0] = "hello";
                var1[1] = "world";
                account.push(Account(1, "hello"));
                account.push(Account(2, "world"));
        }
}
```

合约存储结构 – 动态变量例子



预编译合约

0x000000...001	Ecrecover	用户签名反解用户公钥
0x000000...002	Sha256	Sha256哈希计算
0x000000...003	Ripemd160	Ripemd160 哈希计算
0x000000...004	Memcpy	字符串拷贝

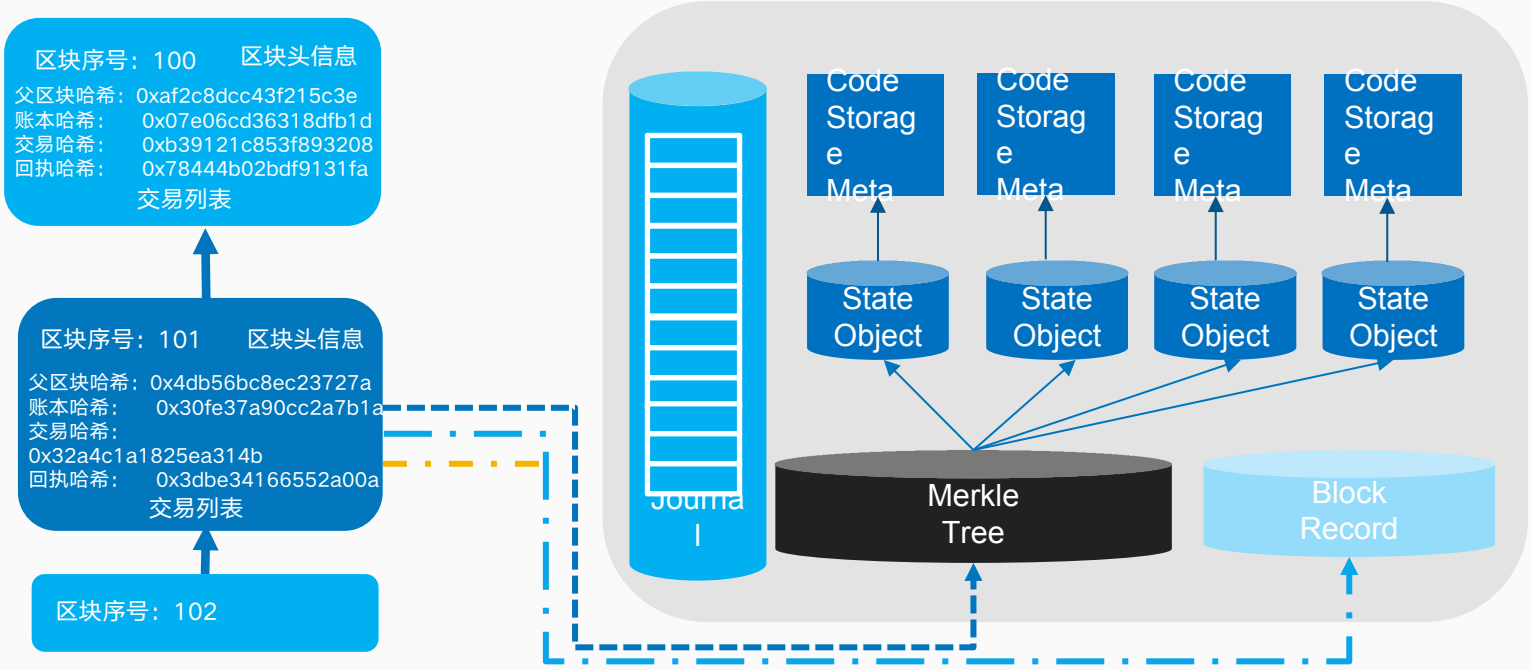
https://github.com/ethereum/go-ethereum/blob/release/1.7/core/vm/contracts.go

预编译合约 – Ecrecover

Ecrecover

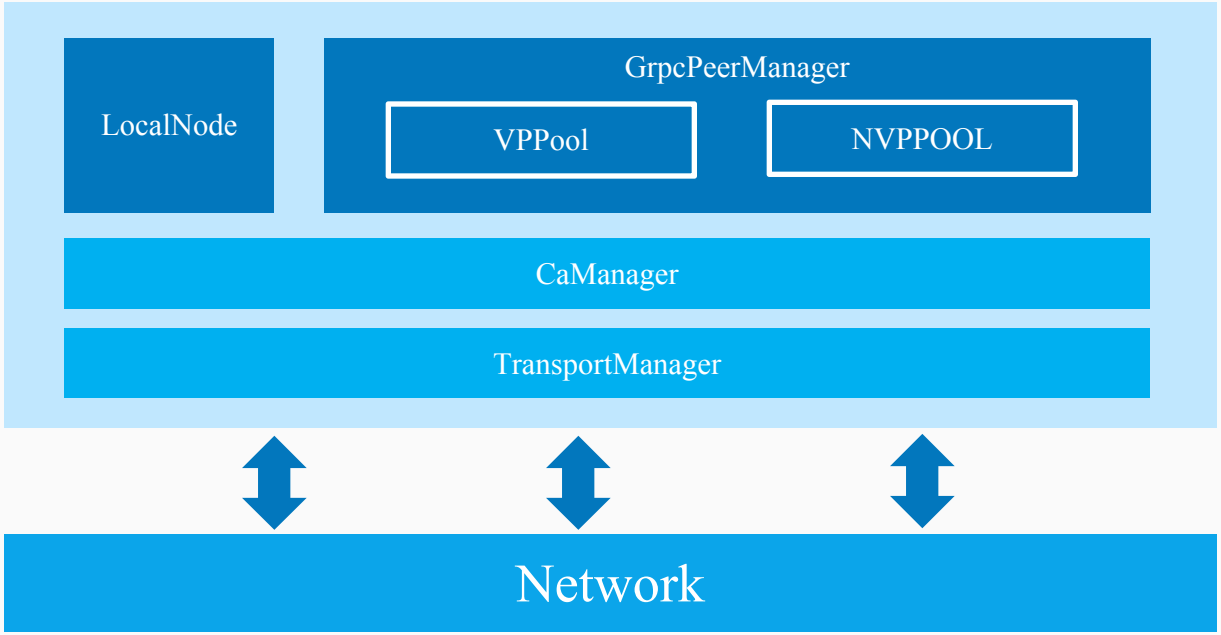
```
function verify(bytes32 msgHash, bytes32 r, bytes32 s, uint8 v, address addr)
    internal returns (bool) {
        address addrv = ecrecover(msgHash,v,r,s);
        if(addr == addrv){
            return true;
        }else{
            return false;
        }

    }
```

存储区块链+世界状态

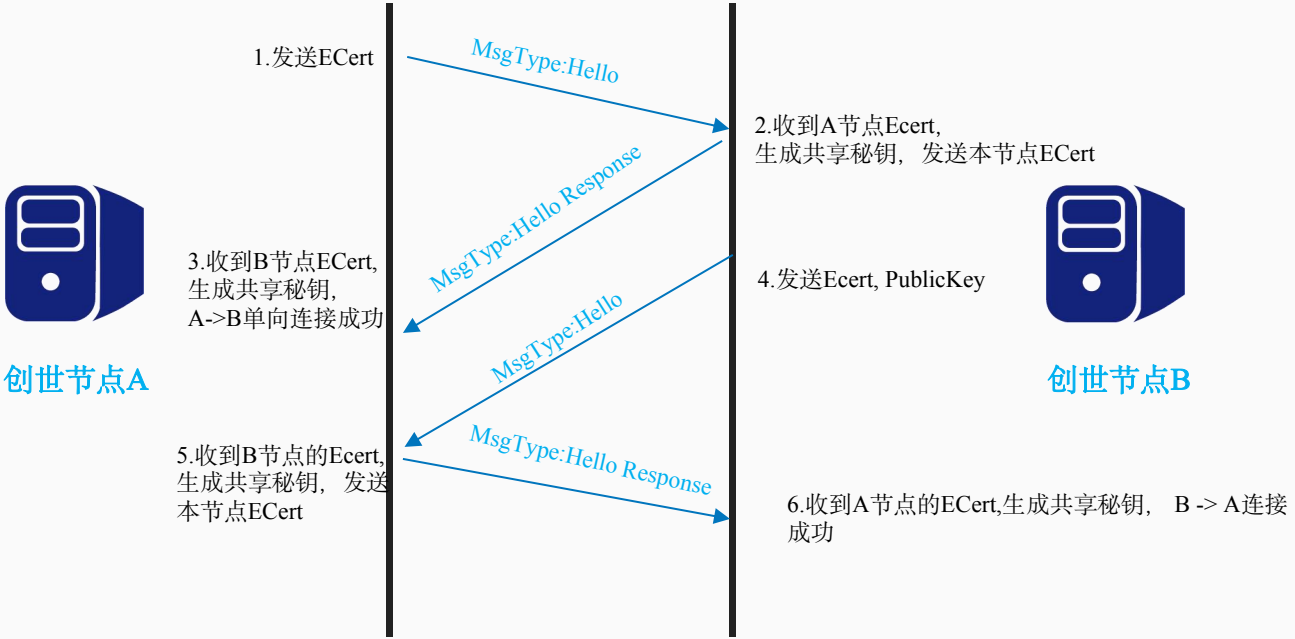

区块序号：100
区块头信息
父区块哈希：0xaf2c8dcc43f215c3e
账本哈希：0x07e06cd36318dfb1d
交易哈希：0xb39121c853f893208
回执哈希：0x78444b02bdf9131fa
交易列表
区块序号：101
区块头信息
父区块哈希：0x4db56bc8ec23727a
账本哈希：0x30fe37a90cc2a7b1a
交易哈希：
0x32a4c1a1825ea314b
回执哈希：0x3dbe34166552a00a
交易列表
区块序号：102
Journal
Code Storage Meta
Code Storage Meta
Code Storage Meta
Code Storage Meta
State Object
State Object
State Object
State Object
Merkle Tree
Block Record

P 2 P 网 络

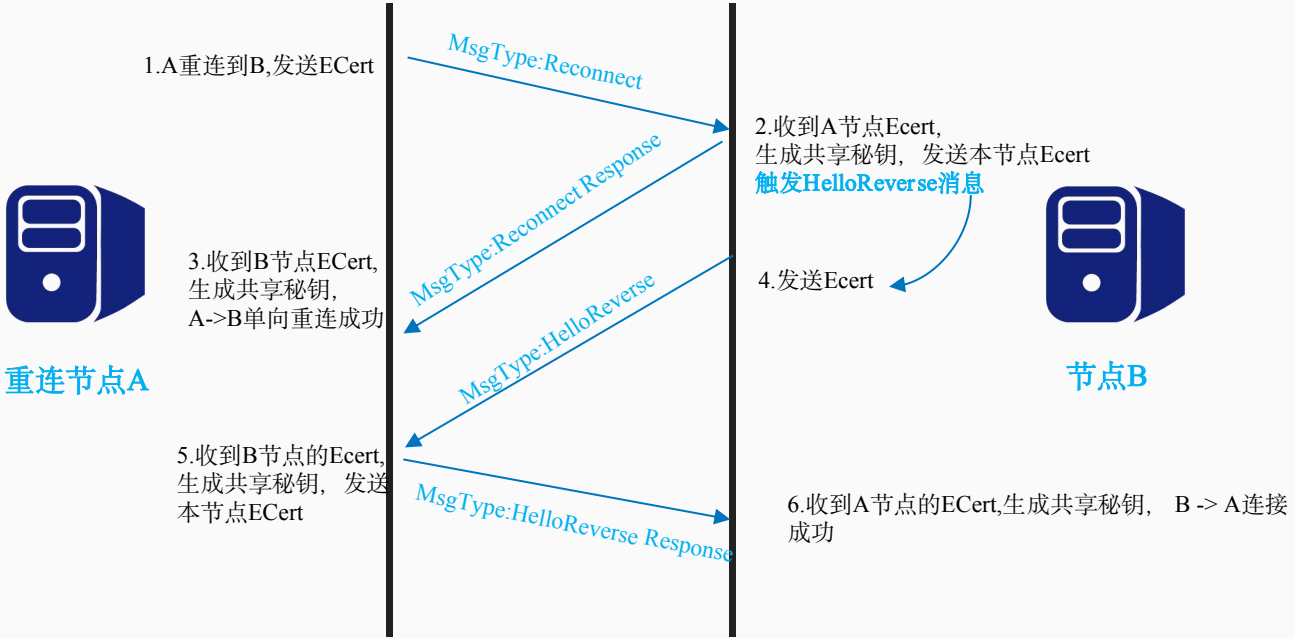
网络P2P


LocalNode
GrpcPeerManager
VPPool
NVPPOOL
CaManager
TransportManager
Network

网络创世节点

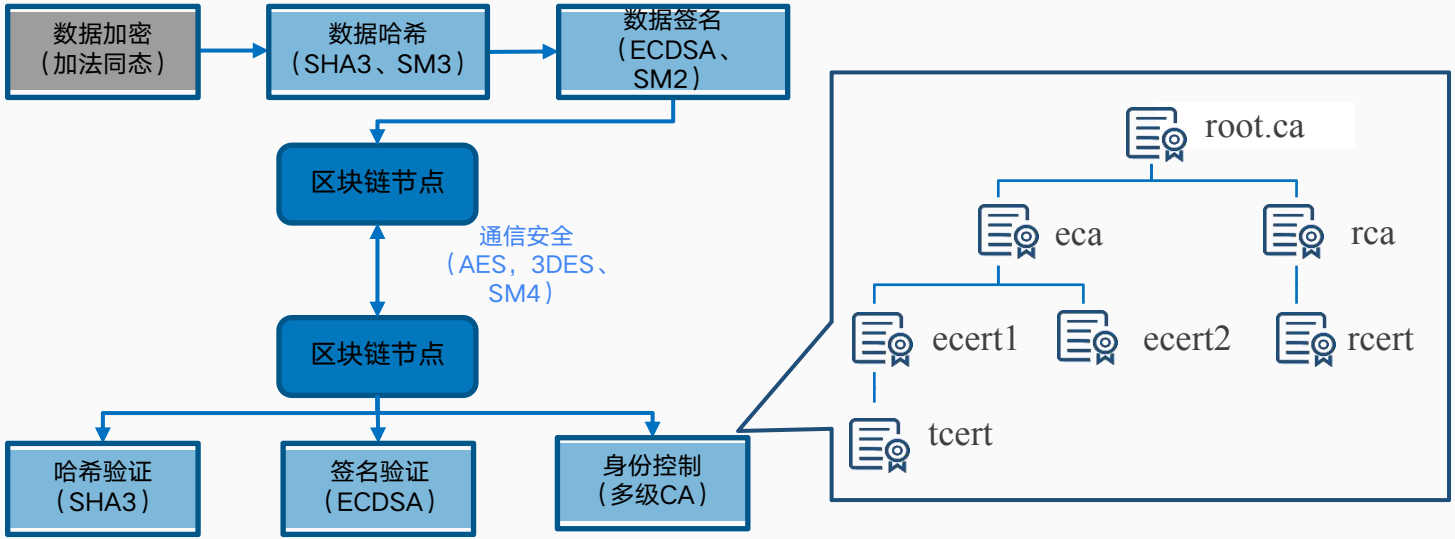

1.发送ECert
MsgType:Hello
2.收到A节点Ecert,
生成共享秘钥，发送本节点ECert
MsgType:Hello Response
3.收到B节点ECert,
生成共享秘钥,
A->B单向连接成功
4.发送Ecert, PublicKey
MsgType:Hello
MsgType:Hello Response
5.收到B节点的Ecert,
生成共享秘钥，发送
本节点ECert
6.收到A节点的ECert,生成共享秘钥， B -> A连接
成功
创世节点A
创世节点B

网络重连


1.A重连到B,发送ECert
MsgType:Reconnect
2.收到A节点Ecert,
生成共享秘钥，发送本节点Ecert
触发HelloReverse消息
MsgType:Reconnect Response
3.收到B节点ECert,
生成共享秘钥,
A->B单向重连成功
4.发送Ecert
MsgType:HelloReverse
重连节点A
节点B
5.收到B节点的Ecert,
生成共享秘钥，发送
本节点ECert
MsgType:HelloReverse Response
6.收到A节点的ECert,生成共享秘钥， B -> A连接
成功

多 级 加 密 体 系

安全多级加密体系


数据加密
（加法同态）
数据哈希
（SHA3、SM3）
数据签名
（ECDSA、
SM2）
区块链节点
通信安全
（AES，3DES、
SM4）
区块链节点
哈希验证
（SHA3）
签名验证
（ECDSA）
身份控制
（多级CA）
root.ca
eca
rca
ecert1
ecert2
rcert
tcert

加密体系

对称加密	非对称加密	密码杂凑	密钥交换
des/3des	rsa	sha1、md5	Diffie–Hellman (DH)
aes	ecdsa	sha3 (Keccak256)	ECDH
sm4	sm2	sm3	国密DH

对称加密与非对称加密

名称	密钥管理	安全性	速度
对称算法	比较难,不适合互联网,一般用于内部系统	中	快好几个数量级(软件加解密速度至少快100倍,每秒可以加解密数M比特数据),适合大数据量的加解密处理
非对称算法	密钥容易管理	高	慢,适合小数据量加解密或数据签名

对称加密(hts模块)

名称	密钥长度	运算速度	安全性	资源消耗
DES	56位	较快	低	中
3DES	112位或168位	慢	中	高
AES	128、192、256位	快	高	低

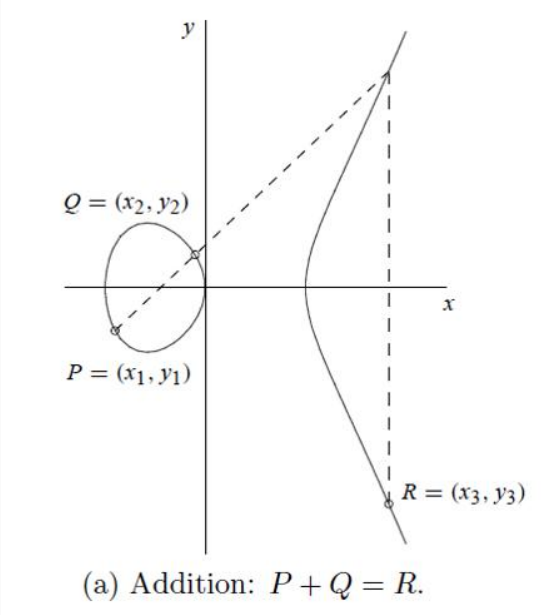
非对称加密

名称	安全性(取决于密钥长度)	运算速度	资源消耗
RSA	高	慢	高
ECDSA	高	快	低(计算量小,存储空间占用小,带宽要求低)

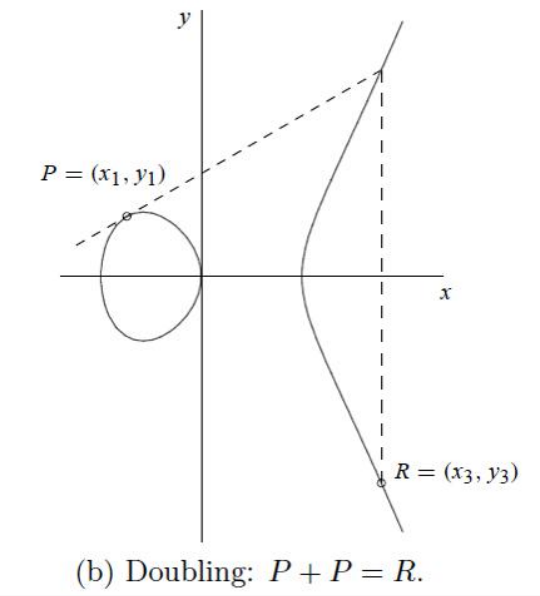
基本原则：私钥==>公钥;公钥=/=>私钥

1.RSA是一个它的安全性依赖于因式分解是缓慢的和乘法的快速的密码方案。
2.ECDSA是一个基于加法阶数难求问题的密码方案。

椭圆曲线



(a) Addition: $P + Q = R$.

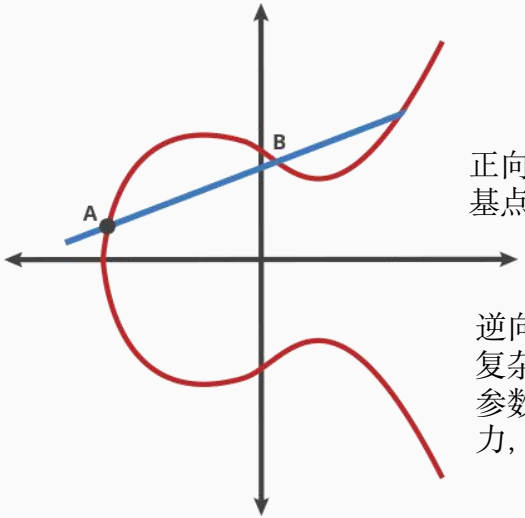


(b) Doubling: $P + P = R$.

公钥==>私钥:公钥=/=>私钥

公私钥对结构

```
// PublicKey represents an ECDSA public key.
type PublicKey struct {
        elliptic.Curve
        X, Y *big.Int
}

// PrivateKey represents a ECDSA private key.
type PrivateKey struct {
        PublicKey
        D *big.Int
}
```

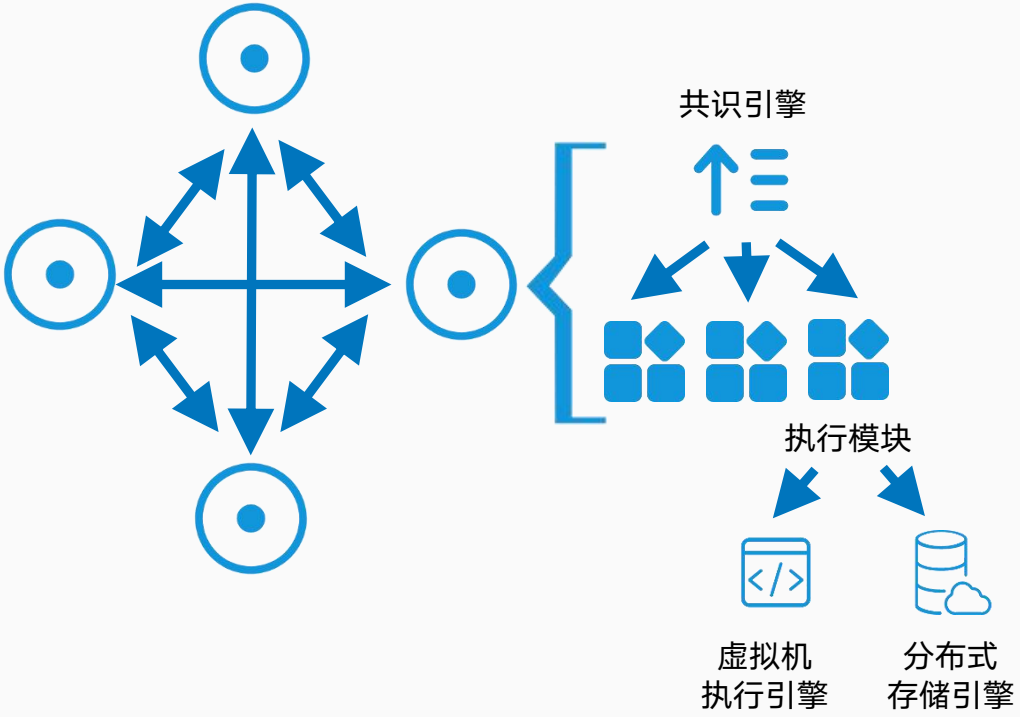


正向计算的定义很简单，私钥为 $K \in [0, n)$;
基点为点P;公钥点Q定义为K个P相加:$Q = \underbrace{P + \widehat{P + \cdots + P)}}_{K}$ 。

逆向计算，由椭圆曲线公钥求解私钥的最有效算法复杂度为 $O(\sqrt{p})$,其中p是阶数n的最大素因子。当参数选的足够好让p>2^{160}时，以目前的计算能力，攻破椭圆曲线是不现实的

技　术　前　沿

可扩展性——微服务架构



可扩展性

- 共识引擎与执行模块隔离
- 执行引擎规模可扩展
- 充分体现多Namespace的性能优势

自适应共识算法

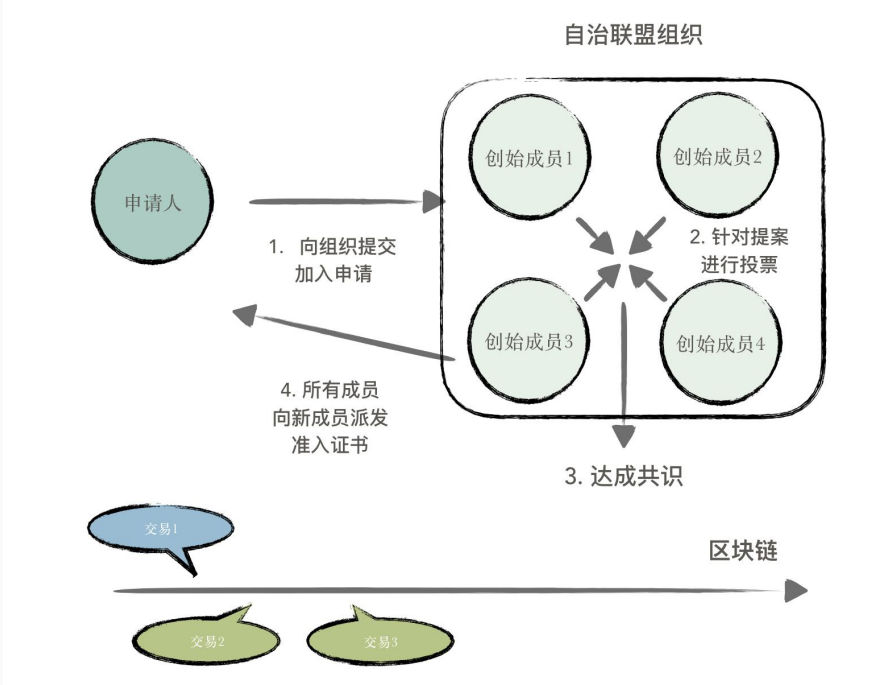
- 支持多类型的共识算法
- 支持不同共识引擎的动态切换

海量数据存储

- 支持多类型存储引擎进行数据存储
- 支持分布式存储引擎存储海量数据

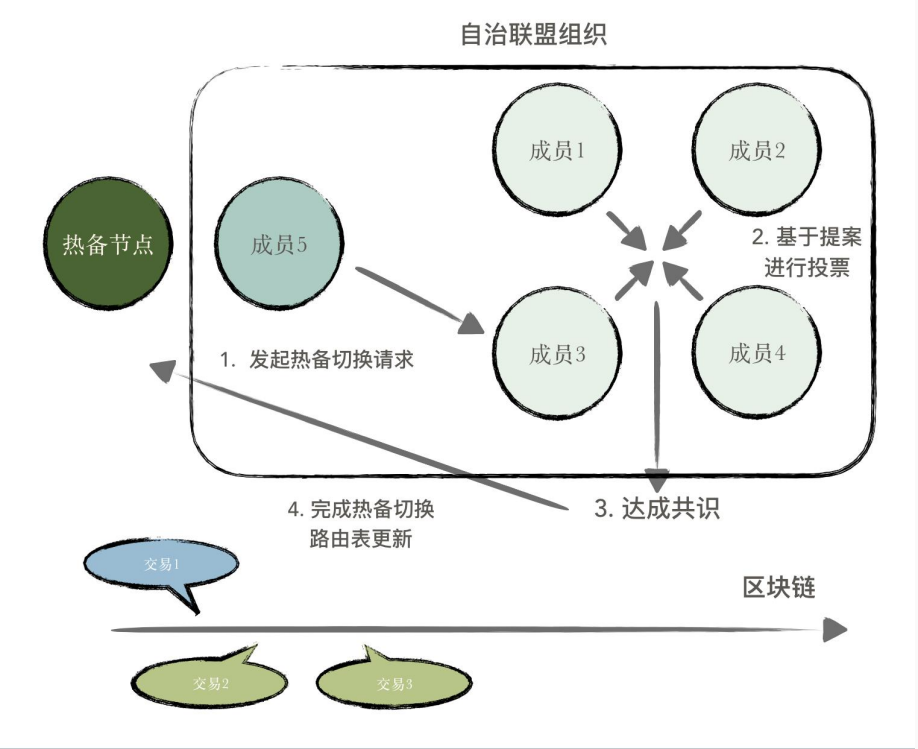
联盟自治：成员管理

- 联盟链成员可以通过ACO来进行**区块链网络管理**
- 新成员可以通过向ACO提交提案申请**加入**区块链网络
- 旧成员可以通过向ACO提交提案申请**退出**区块链网络
- 成员变动需要获得自治联盟组织**绝大多数**的成员同意
- 准入证书采用自颁发形式，形成**去中心化**的Ca体系



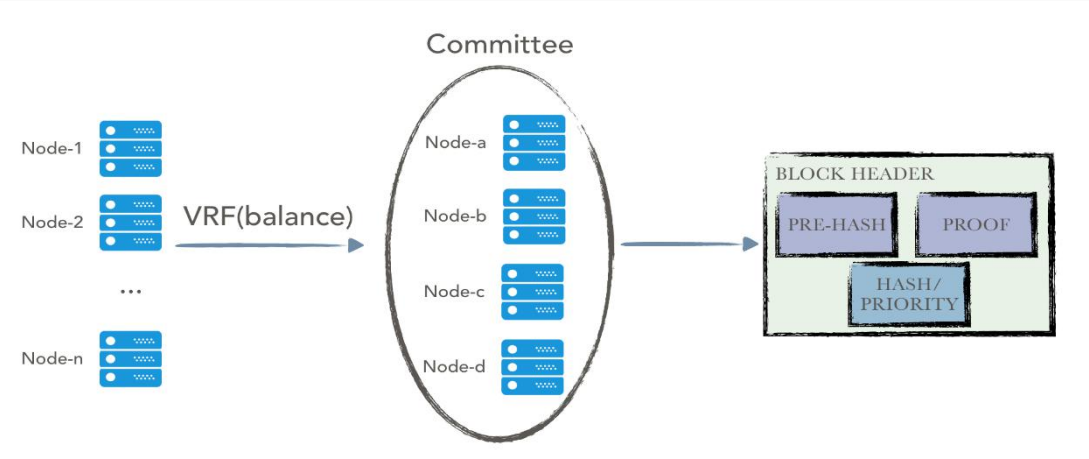
联盟自治：热备切换

- 联盟链成员可以通过ACO来进行**动态热备切换**
- 故障节点可以**主动发起热备替换请求**，将热备节点与自身进行切换；
- 其他节点检测到连接故障，可以根据预定义规则**协助故障节点进行热备切换**；
- 自治联盟组织绝大多数成员同意该提案，进行**路由表替换**，完成热备切换；



两类共识机制结合（Algorand）

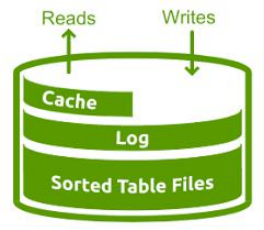
原理：节点先通过VRF算法选举出一批节点作为committee成员节点，这些节点会通过gossip广播将区块分发给所有节点，其他节点验证该节点的committee合法性之后Byzantine-Agreement区块确认。



VRF函数的特点：

1. 选举结果不可预知性
2. 选举结果的数量确定性

FPGA硬件加速



应用方向

1 密码学全系列加速

- SM国家安全系列
- ECC AES NIST系列
- NTRU 次时代系列

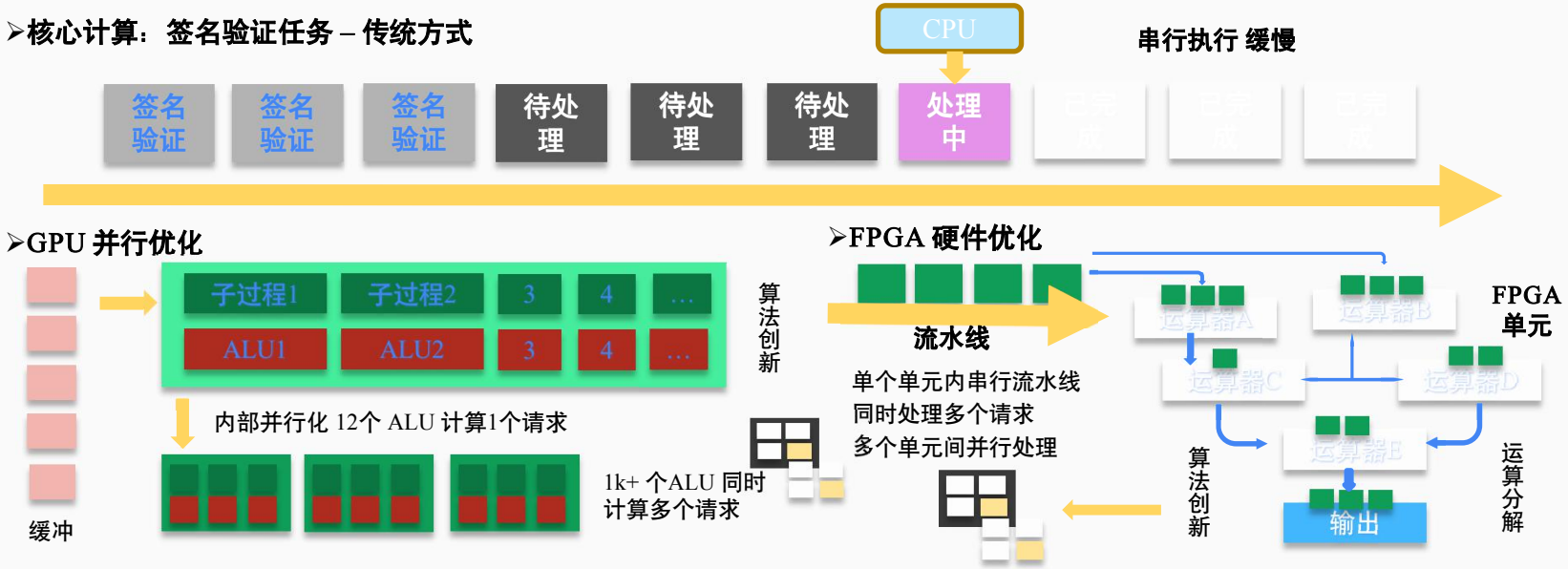
2 存储引擎加速

- LevelDB compaction过程

3 未公开的Intel技术

- FPGA on Master Board via FSB
- AAL & AAS /SKX Framework

性能提升：硬件（GPU/FPGA）加速



小结

本章对企业级联盟链相关技术做了深入解读，有助于读者深入理解联盟链底层实现原理。首先，介绍了联盟链逻辑架构及交流流程；之后，对共识、智能合约执行引擎、账本存储、网络、加密等核心模块算法及底层原理进行介绍；最后，介绍了目前联盟链前沿技术的研究和探索。