WeBank

FISCO BCOS 2.0技术解析

2019年7月

FISCO 金链盟

Introduction

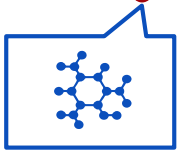
FISCO BCOS 演进路线

方案设计

2017.01-2017.03
技术选型：联盟链
基于BCOS重塑升级

平台研发

2017.04-2017.10
核心算法研发
底层架构研发



应用实践

2017.08-2017.11
完成一个应用研发
验证底层系统可用性

开源发布

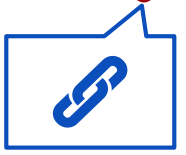
2017.12
在Github完全开源

生态建设

2017.12-～
组建千人规模的社区
数十个应用落地

举办大赛

2018.08-2018.12
近300支团队报名参赛
覆盖十几个行业领域



架构升级

2019.01-2019.03
全新2.0架构升级

FISCO BCOS 2.0

技术要求

What kind of technology do we need

- 支持快速组建联盟和建链的能力，让企业建链像建微信群一样简便
- 具备高可用的多群组能力，能处理海量服务请求
- 具有良好的联盟链治理能力，满足企业级运维管理要求
- 具有可行的隐私保护能力，能支持复杂业务需求落地
- 开源和开放，实现联盟成员之间的充分信任

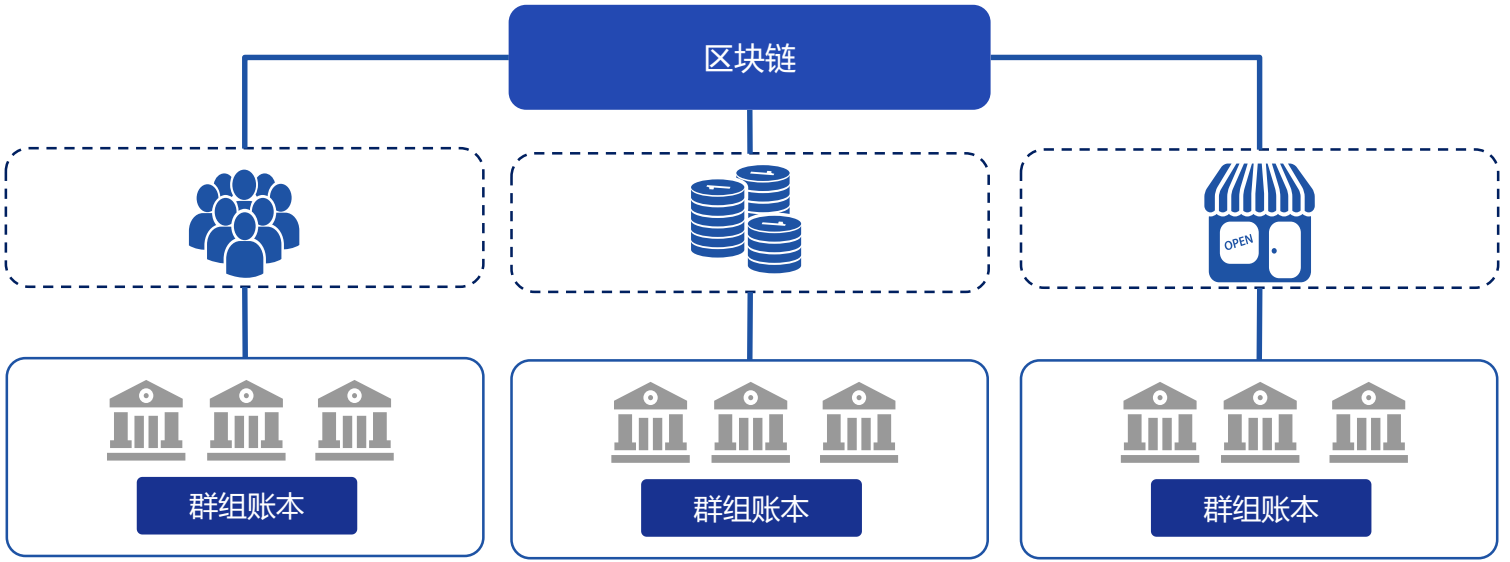
2.0 新特性

What ‘s new in FISCO BCOS 2.0

一体两翼多引擎

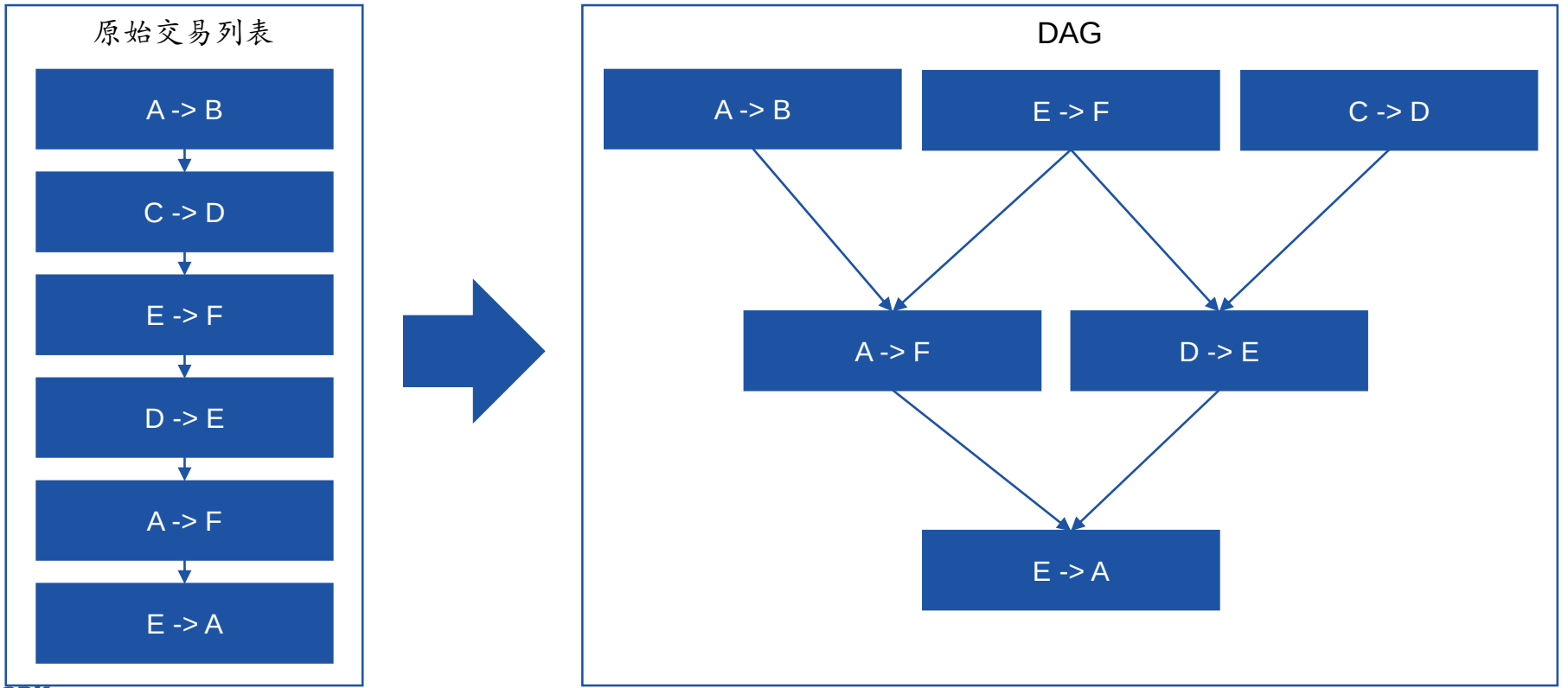
一体：群组架构，建链像建聊天群一样便利

- 在多节点组成的区块链内，部分节点通过配置，组成独立的账本
- 账本内的节点进行独立的共识，存储独立的状态
- 只需要修改配置就能组建账本，无需额外运营资源



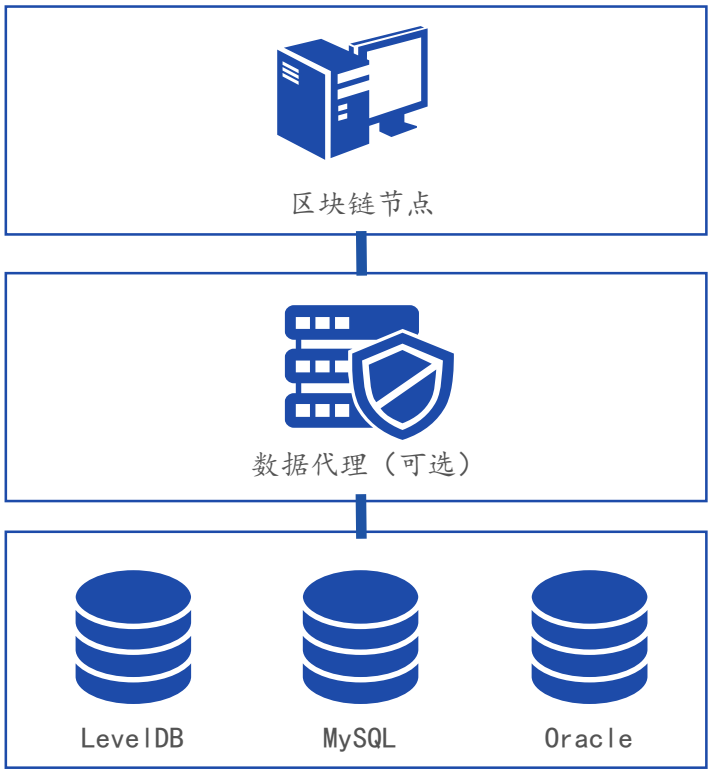
两翼：并行计算模型，让联盟链应用飞的更高

- 区块链自动识别交易的互斥资源，构建出DAG，规划一个可并行的执行路径
- 最佳情况下，性能提升可达到CPU核心数的倍数

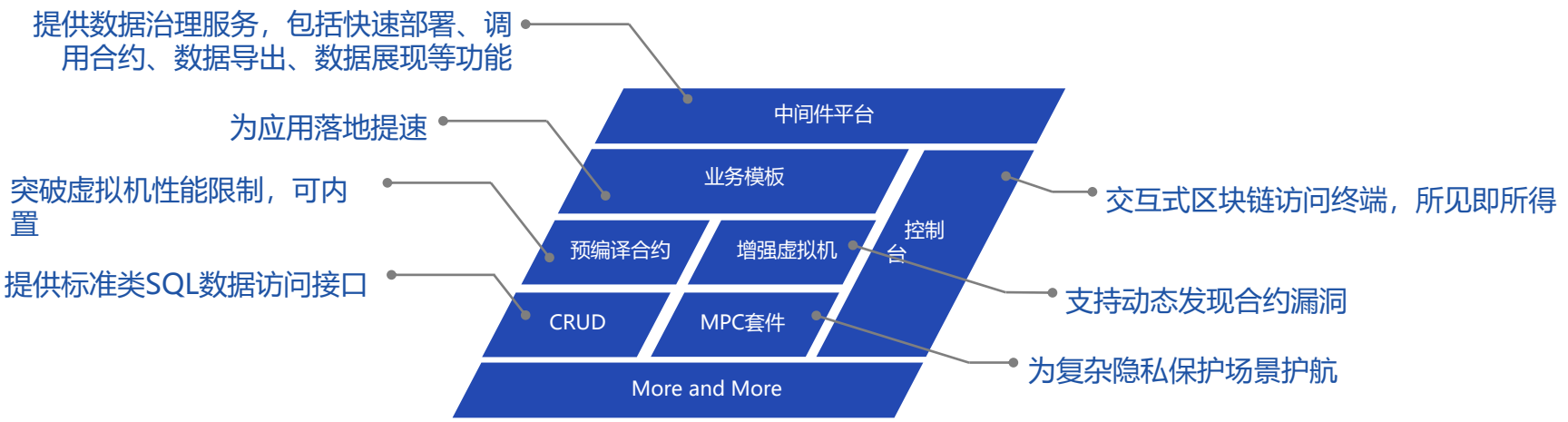


两翼：分布式存储，让联盟链应用飞得更远

- 使用分布式存储替换旧的本机存储，突破了单机磁盘容量的限制，支持横向扩展
- 分布式存储定义了标准的数据访问CRUD接口，适配多种数据存储系统
- 区块链节点既可以直接访问存储，也可以通过专用的数据代理访问存储
- 适配关系型数据（MySQL），简化数据操作
- 数据管理更方便，容易支持数据迁移/备份

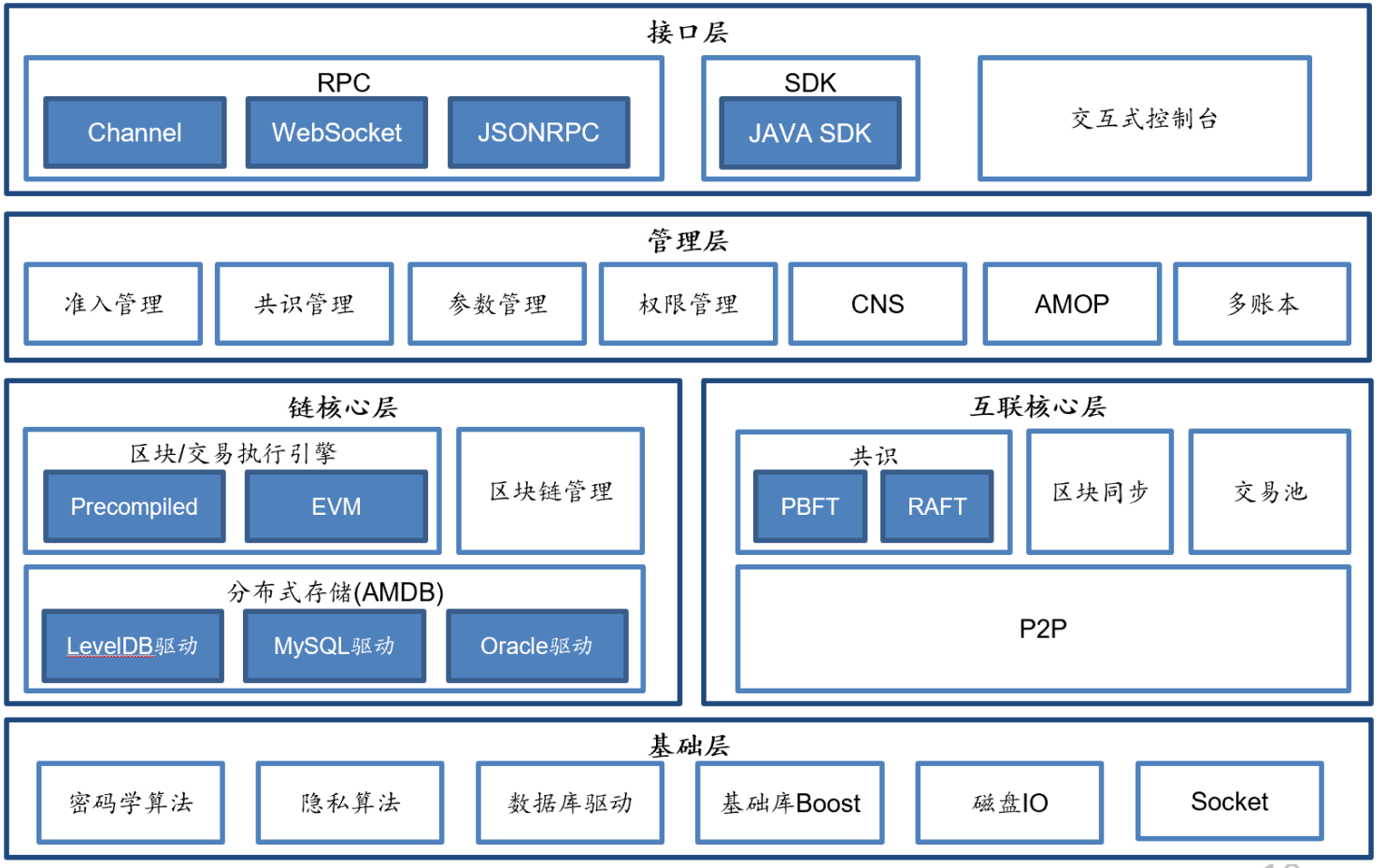


多引擎：为联盟链应用保驾护航

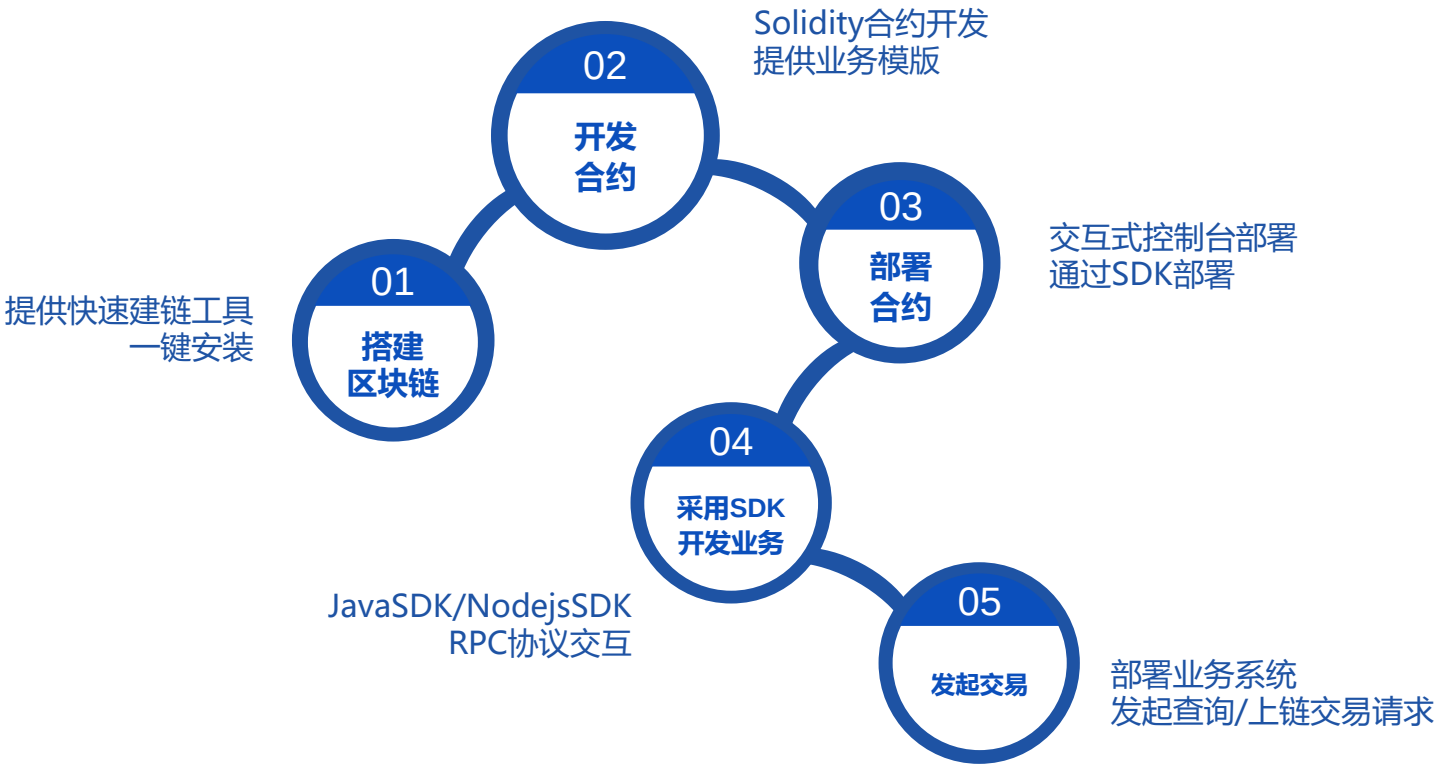


Architecture & Workflow

FISCO BCOS 逻辑架构

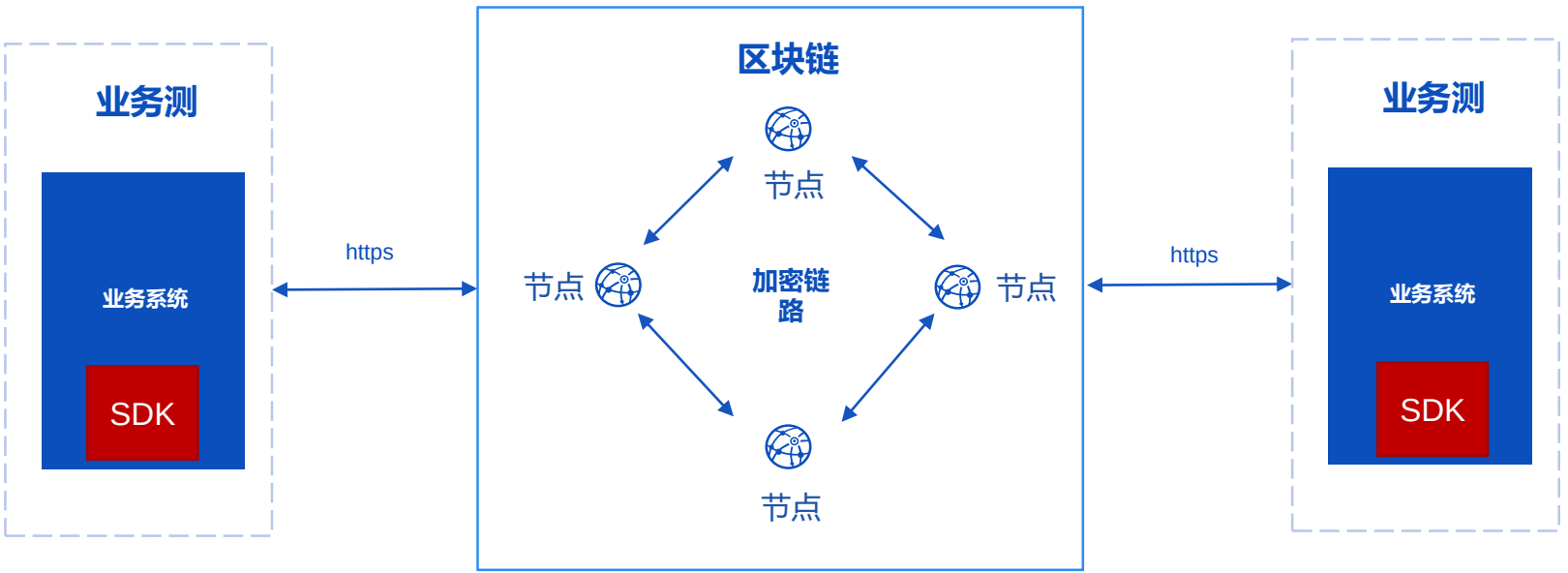


区块链业务开发全流程


02
开发
合约
Solidity合约开发
提供业务模版
01
搭建
区块链
提供快速建链工具
一键安装
03
部署
合约
交互式控制台部署
通过SDK部署
04
采用SDK
开发业务
JavaSDK/NodejsSDK
RPC协议交互
05
发起交易
部署业务系统
发起查询/上链交易请求

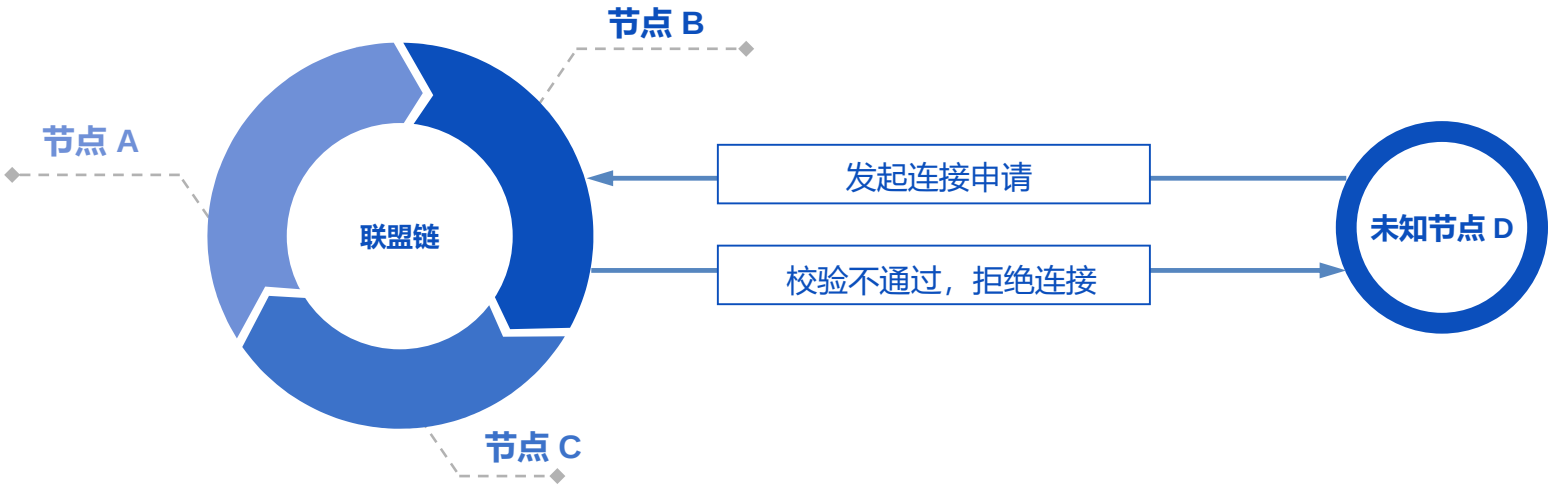
FISCO BCOS 部署架构 01

- 区块链可灵活选择在公有云/金融云，或者机构自有机房等的多种部署方式
- 尽可能部署在高质量网络，比如同一个云环境，或者与云环境建立高速通道

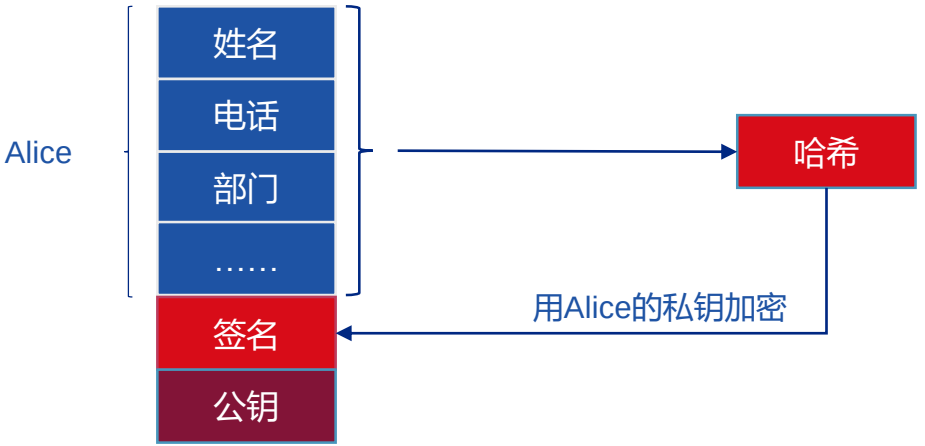


节点准入控制

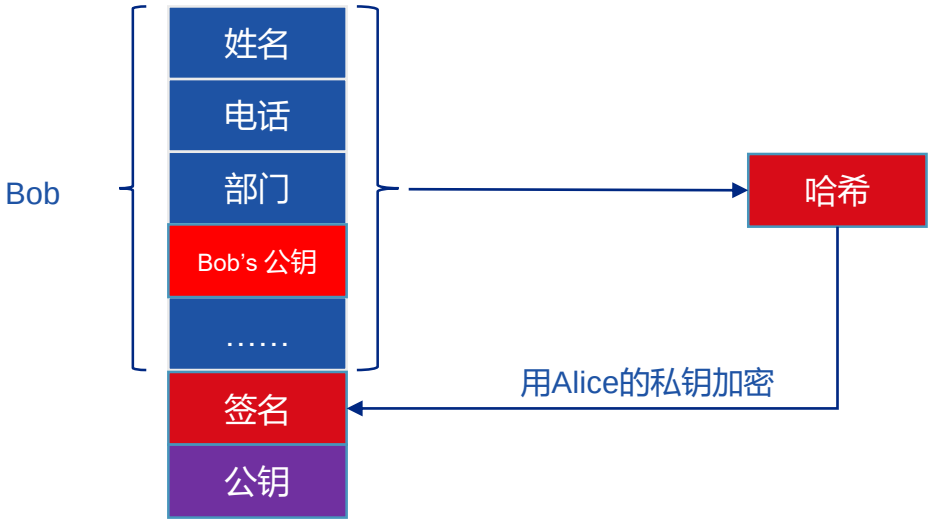
- 联盟链又称许可链，参与区块链网络的节点都需要经过准入控制
- 采用SSL通信机制，基于PKI的CA认证机制
- 节点白名单机制
- 证书黑名单机制



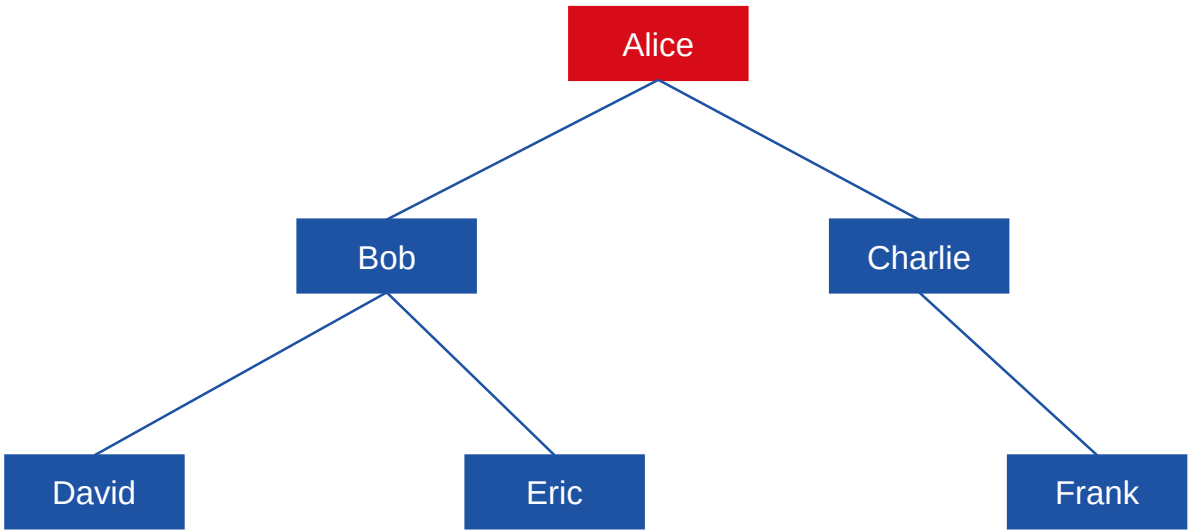
自签证书



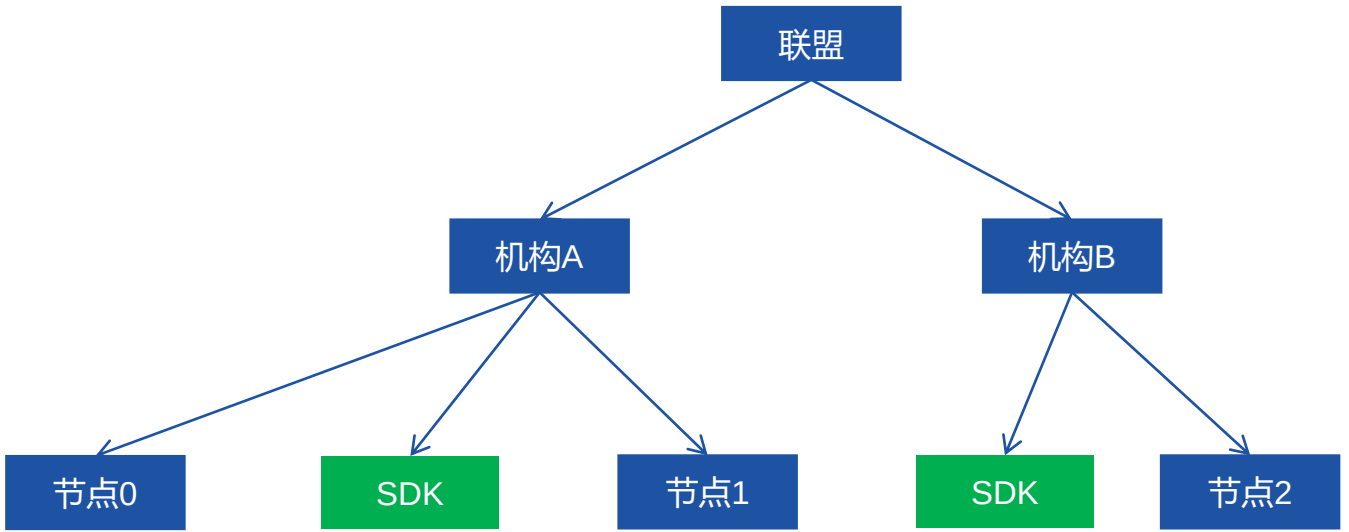
代理证书



证书链



FISCO BCOS三级证书



合约开发——Solidity 02

- FISCO BCOS 支持以太坊的Solidity语言进行智能合约开发；
- Solidity是图灵完备的高级语言，支持循环、函数调用等用法；
- Solidity拥有丰富的数据类型，支持整形、字符串、数组、Map等；
- Solidity支持继承、库引用等高级用法；
- Solidity拥有大量的参考实现；
- Solidity拥有广泛的开发者；
- 更多有关Solidity合约开发，将会在下个课时专门讲解。

```
pragma solidity ^0.4.24;

contract HelloWorld {
    string name;

    function HelloWorld() {
        name = "Hello, World!";
    }

    function get()constant returns(string) {
        return name;
    }

    function set(string n) {
        name = n;
    }
}
```

合约开发——Precompiled合约

- 采用C++编写合约逻辑，合约编译集成进FISCO BCOS底层节点
- 与Solidity合约一致的访问方式
- 调用合约不进EVM，突破EVM性能瓶颈
- 提供标准开发框架，只需继承基类，实现call接口即可

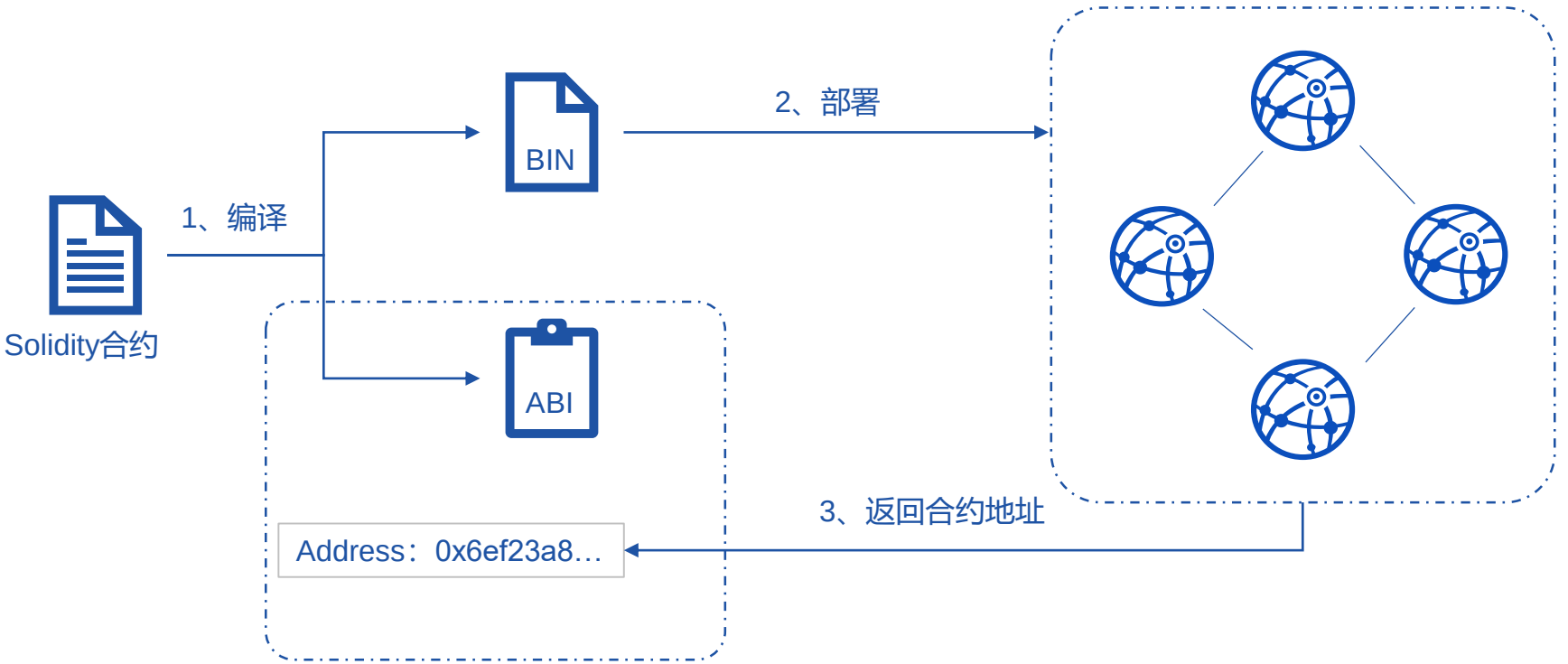
类型	Precompiled引擎	EVM引擎
寻址	固定地址，代码中定义	部署时确定
存储	数据存储在表中，与合约分离，可升级合约逻辑	合约变量和数据存储在MPT树中
执行	C++底层执行，性能更高，可实现并行	EVM虚拟机，串行执行

合约开发——CRUD合约

- 基于Precompiled合约实现高效的CRUD存储接口
- CRUD接口可供Solidity合约调用
- 实现传统数据库方式的合约开发体验

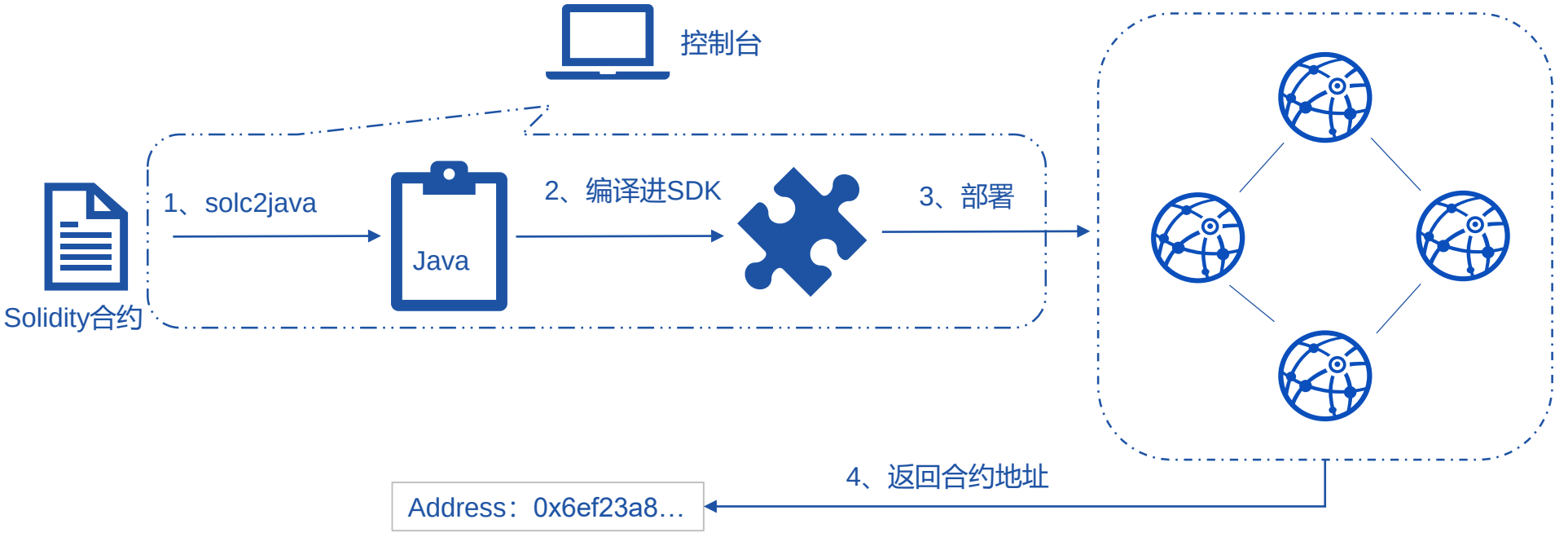
```
// 插入数据
function insert(string name, int item_id, string item_name) public returns(int) {
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("t_test");

    Entry entry = table.newEntry();
    entry.set("name", name);
    entry.set("item_id", item_id);
    entry.set("item_name", item_name);

    int count = table.insert(name, entry);
    emit insertResult(count);

    return count;
}
```

合约编译部署 03



4、根据ABI和合约地址，可以构造调用合约接口的请求

合约编译部署——采用SDK



5、根据合约地址以及生成好的Java类，可以直接通过Java调用合约

CNS

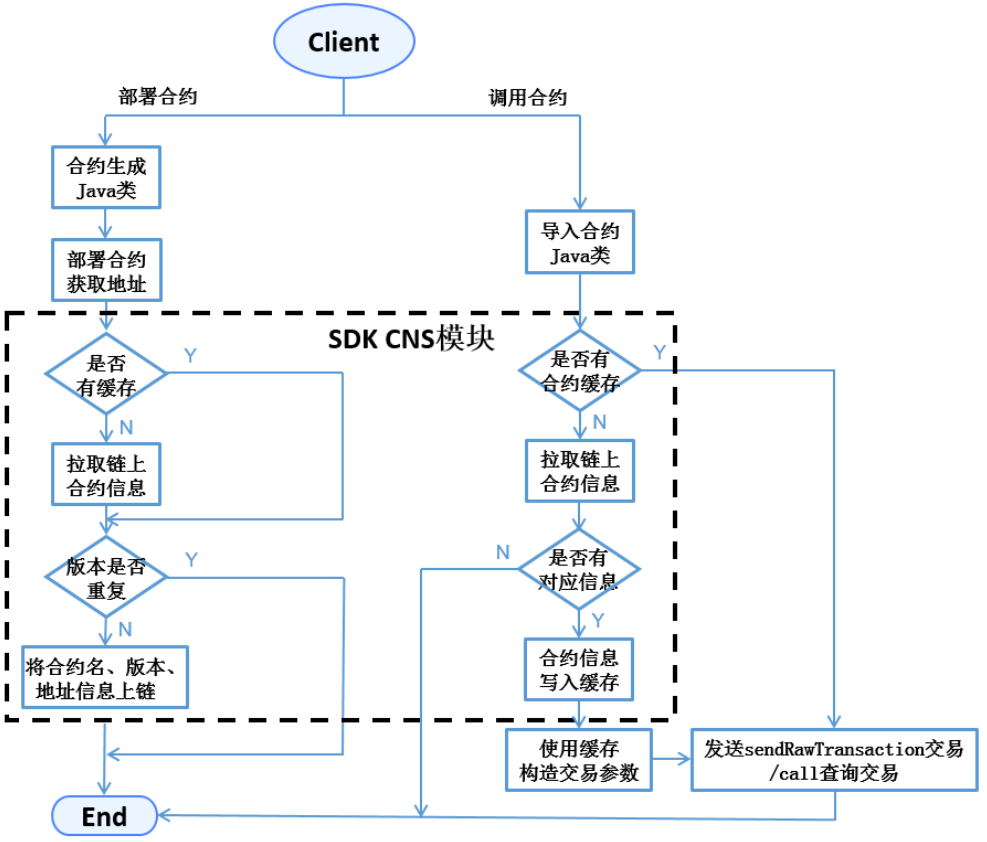
1. 合约abi为较长的JSON字符串，调用方不需直接感知；
2. 合约地址为20字节的魔数，不方便记忆，若丢失后将导致合约不可访问；
3. 合约重新部署后，一个或多个调用方都需更新合约地址；
4. 不便于进行版本管理以及合约灰度升级。

为解决以上问题，给调用者提供良好的智能合约调用体验，FISCO BCOS提出**CNS合约命名服务**。

- **CNS**（Contract Name Service）通过提供链上合约名称与合约地址映射关系的记录及相应的查询功能，方便调用者通过记忆简单的合约名来实现对链上合约的调用。
- **CNS信息**为合约名称、合约版本、合约地址和合约abi
- **CNS表**用于存储CNS信息

优势：
简化调用合约方式；
合约升级对调用者透明，支持合约灰度升级。

CNS核心流程


Client
部署合约
调用合约
合约生成Java类
部署合约获取地址
导入合约Java类
SDK CNS模块
是否有缓存
Y
N
拉取链上合约信息
版本是否重复
Y
N
将合约名、版本、地址信息上链
是否有合约缓存
Y
N
拉取链上合约信息
是否有对应信息
N
Y
合约信息写入缓存
使用缓存构造交易参数
发送sendRawTransaction交易/call查询交易
End

CNS SDK API

Field	Type	Null	Key	Expain
name	string	No	PRI	合约名称，name和version为联合主键
version	string	No		合约名称，name和version为联合主键
address	string	No		合约地址
abi	string	YES		合约abi
status	string	No		分布式存储通用字段，“0”可用“1”删除

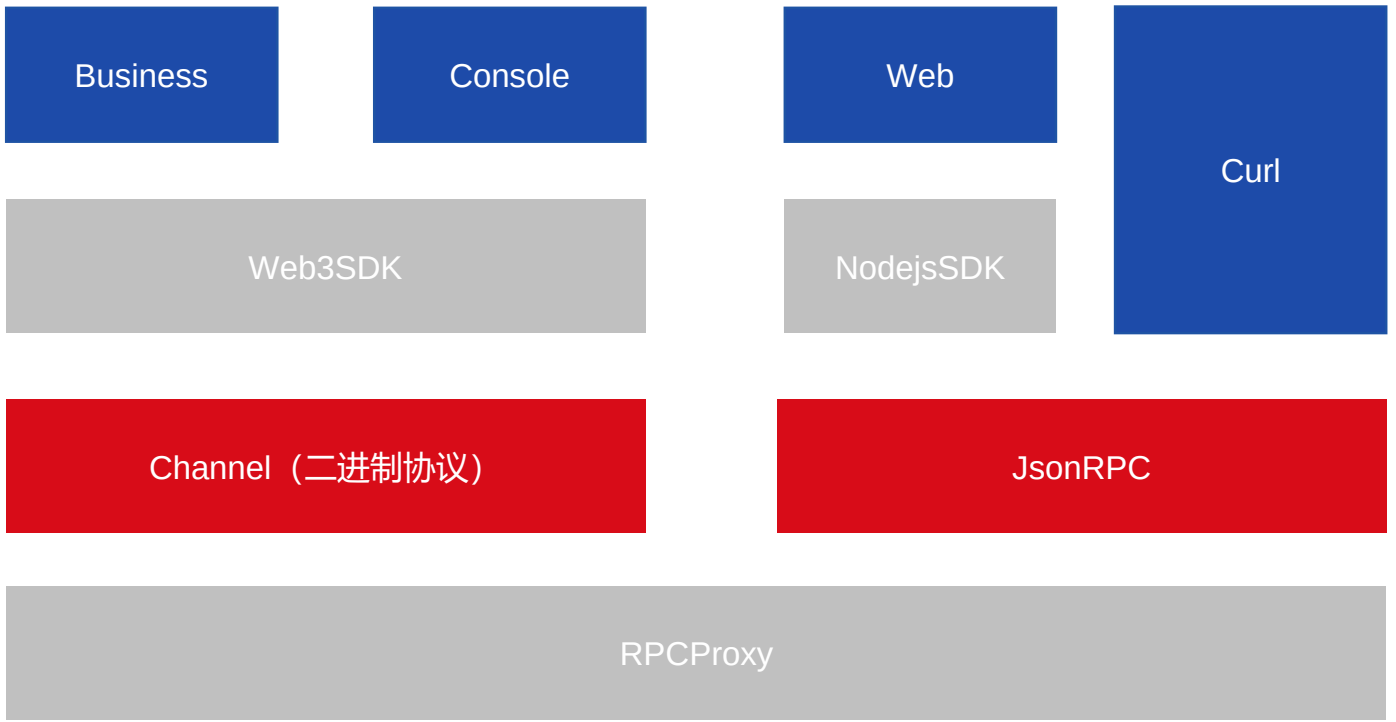
registerCns

•描述：public TransactionReceipt registerCns(String name, String version, String addr, String abi)
•功能：上链合约信息
•参数：name——合约名，version——合约版本，addr——合约地址，abi——合约abi
•返回：上链交易回执，回执中含上链结果信息及错误信息（如有）

resolve

•描述：public String resolve(String contractNameAndVersion)
•功能：基于合约名和合约版本查询合约地址
•参数：contractNameAndVersion——合约名+合约版本信息
•返回：合约地址，如无参数指定版本的合约信息，接口抛出异常
•说明：contractNameAndVersion通过:来分割合约名和合约版本，当缺少合约版本时，SDK默认调用使用合约的最新版本进行查询

区块链接入层设计 04



- 支持Json与二进制协议
- 支持短连接与长连接
- 支持多语言SDK
- 交互式终端

JsonRPC协议

RPC请求包格式示例:

```
{"jsonrpc": "2.0", "method": "getBlockNumber", "params": [1], "id": 1}
```

RPC响应包格式示例:

```
{"jsonrpc": "2.0", "result": "0x1", "id": 1}
```

getBlockNumber

返回节点指定群组内的最新区块高度

参数

- `groupID` : `unsigned int` - 群组ID

返回值

- `string` - 最新区块高度(0x开头的十六进制字符串)


```
// Request curl -X POST --data
‘{“jsonrpc”:“2.0”,“method”:“getBlockNumber”,“params”:[1],“id”:1}’
http://127.0.0.1:8545 | jq

// Result
{ "id": 1, "jsonrpc": "2.0", "result": "0x1" }
```

发送交易（调用合约上链） 05

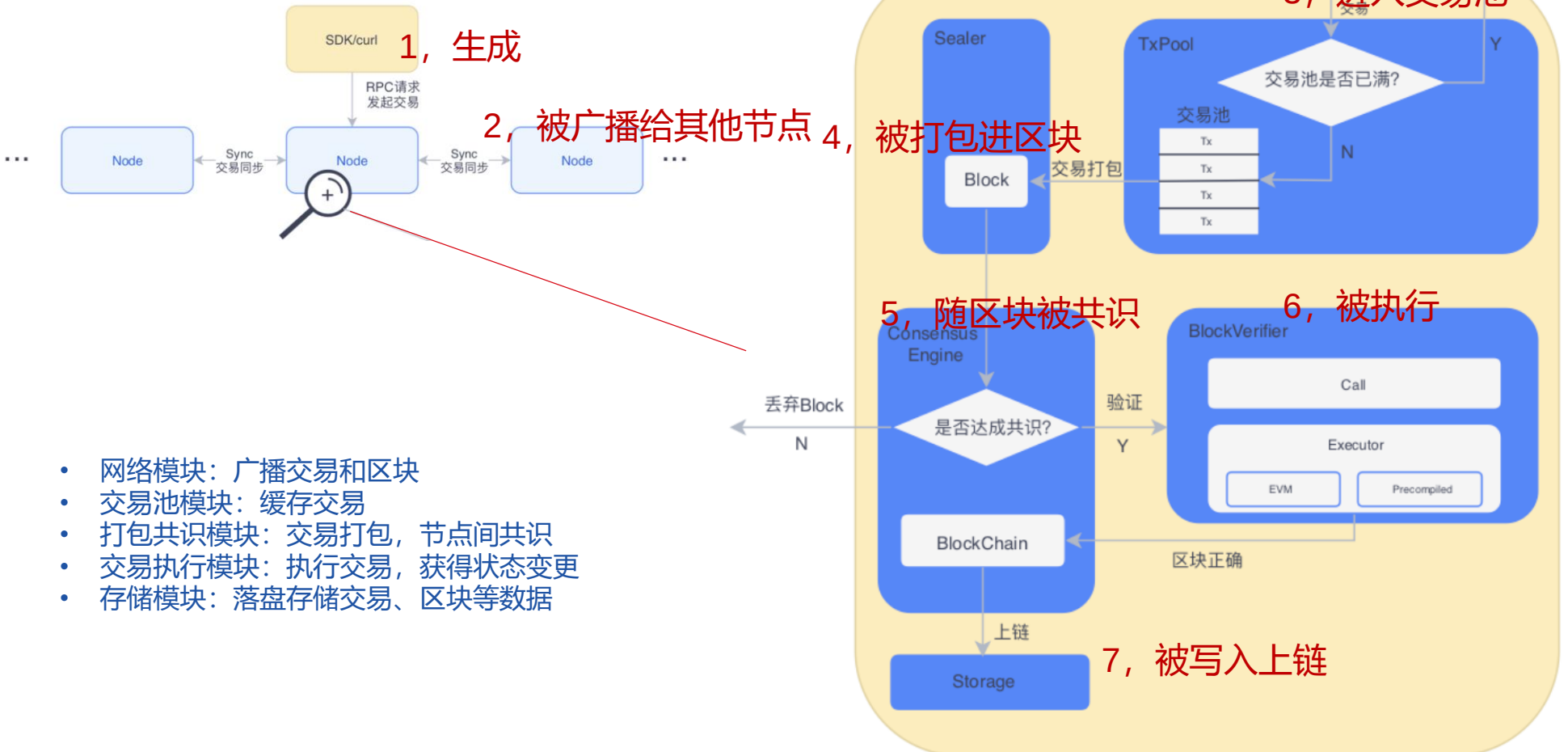
- 获取合约地址（上一步部署合约获得）
- 确定调用的接口
- 确定需要传入的接口参数
- 构造一个交易（如右边数据结构）
- 对交易进行签名
- 对交易进行编码
- 调用RPC接口，发送交易请求给区块链
- 获取交易Hash值
- 根据交易Hash值，异步查询区块链获取交易回执
- 根据交易回执，判断交易完成状态

采用SDK，2-8将全部封装好，用户无感知

交易数据结构

name	type	description
type	enum	交易类型，表明该交易是创建合约还是调用合约交易，初始为空合
nonce	u256	消息发送方提供的随机数，用于唯一标识交易
value	u256	转账数额，目前去币化的FISCO BCOS不使用该字段
receiveAddress	h160	交易接收方地址，type为创建合约时该地址为0x0
gasPrice	u256	本次交易的gas的单价，FISCO BCOS中为固定值300000000
gas	u256	本次交易允许最多消耗的gas数量，FISCO BCOS可配置该值
data	vector< byte >	与交易相关的数据，或者是创建合约时的初始化参数
chainId	u256	记录本次交易所属的链信息/业务信息
groupId	u256	记录本次交易所属的群组
extraData	vector< byte >	预留字段，记录交易信息，内部使用"#"分割信息
vrs	SignatureStruct	交易发送方对交易7字段RLP编码后的哈希值签名生成的数据
hashWith	h256	交易结构所有字段（含签名信息）RLP编码后的哈希值
sender	h160	交易发送方地址，基于vrs生成
blockLimit	u256	交易生命周期，该交易最晚被处理的块高，FISCO BCOS新增字段
importTime	u256	交易进入交易池的unix时间戳，FISCO BCOS新增字段
rpcCallback	function	交易出块后RPC回调，FISCO BCOS新增字段

交易生命周期

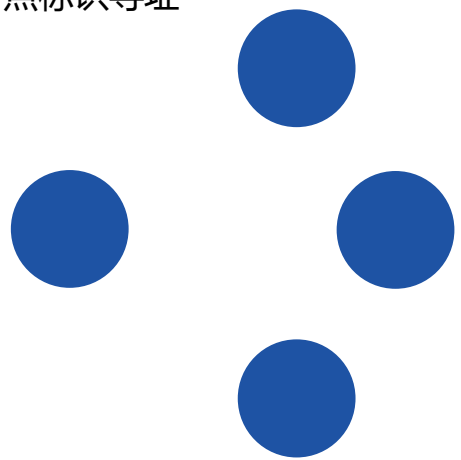


- 网络模块：广播交易和区块
- 交易池模块：缓存交易
- 打包共识模块：交易打包，节点间共识
- 交易执行模块：执行交易，获得状态变更
- 存储模块：落盘存储交易、区块等数据

多模网络设计

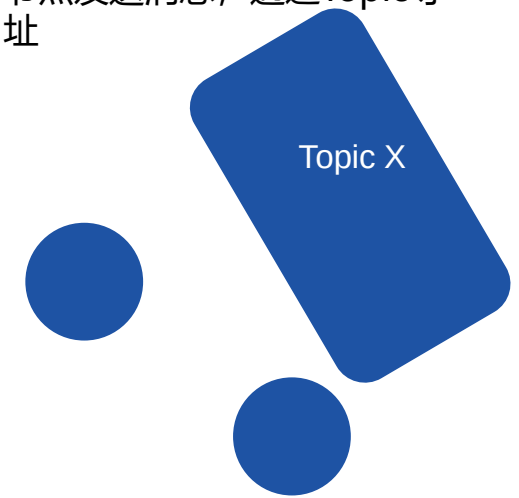
单播

单个区块链节点向单个区块链节点发送消息，通过区块链节点标识寻址



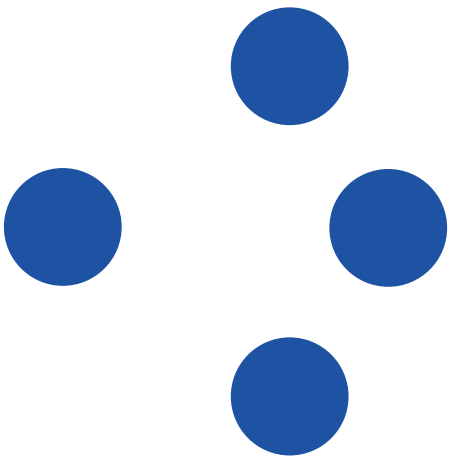
组播

单个区块链节点向一组区块链节点发送消息，通过Topic寻址

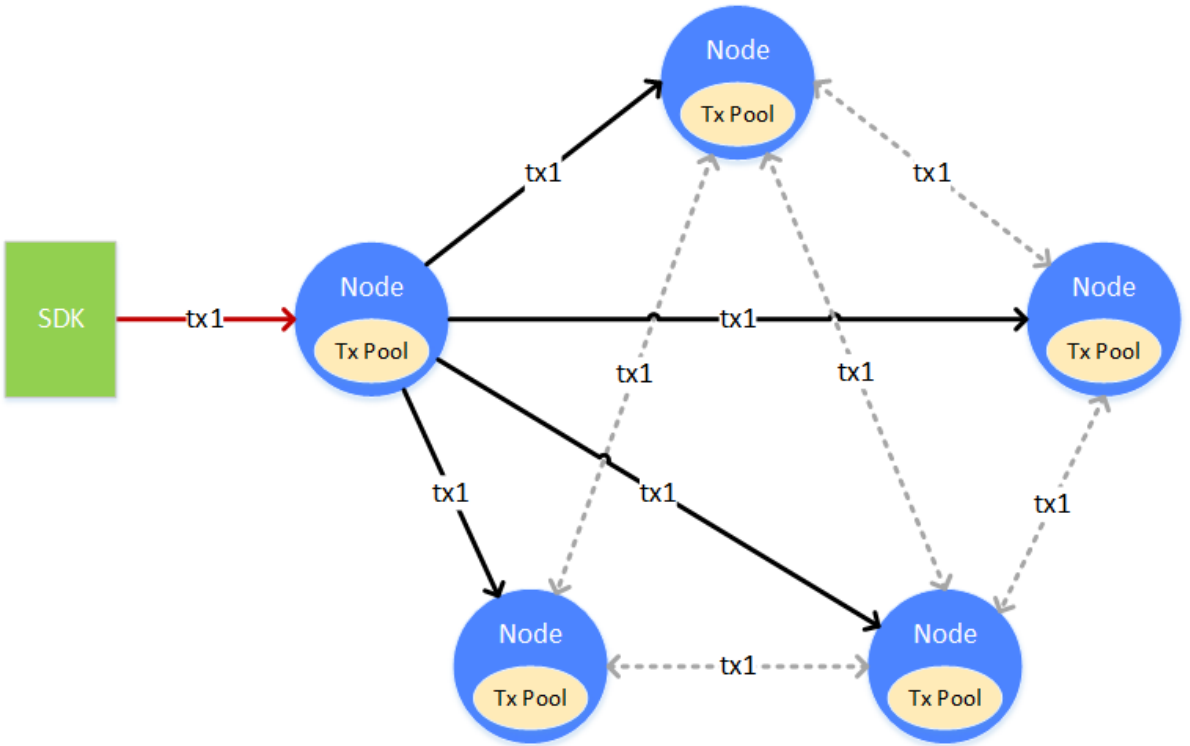


广播

单个区块链节点所有节点发送消息

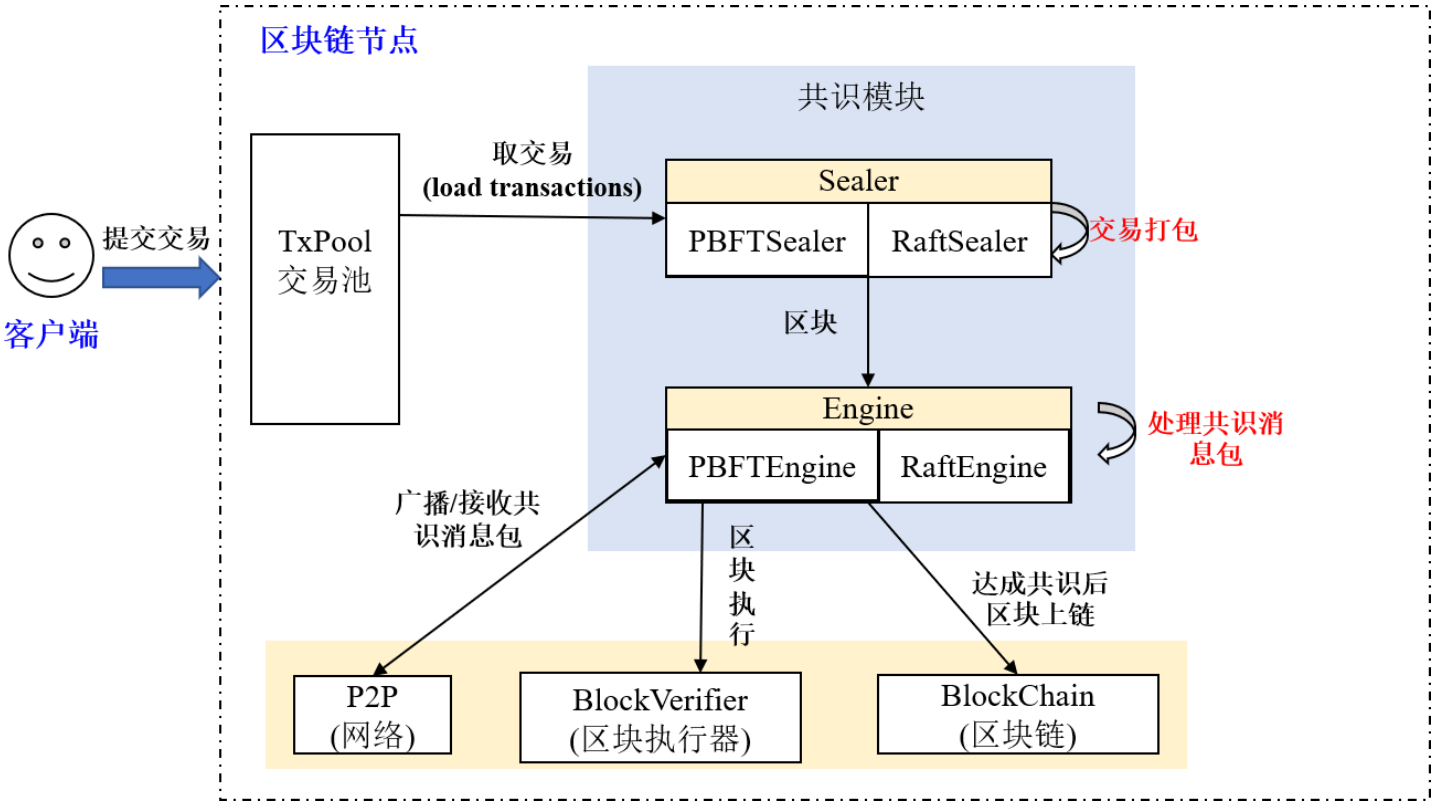


交易同步



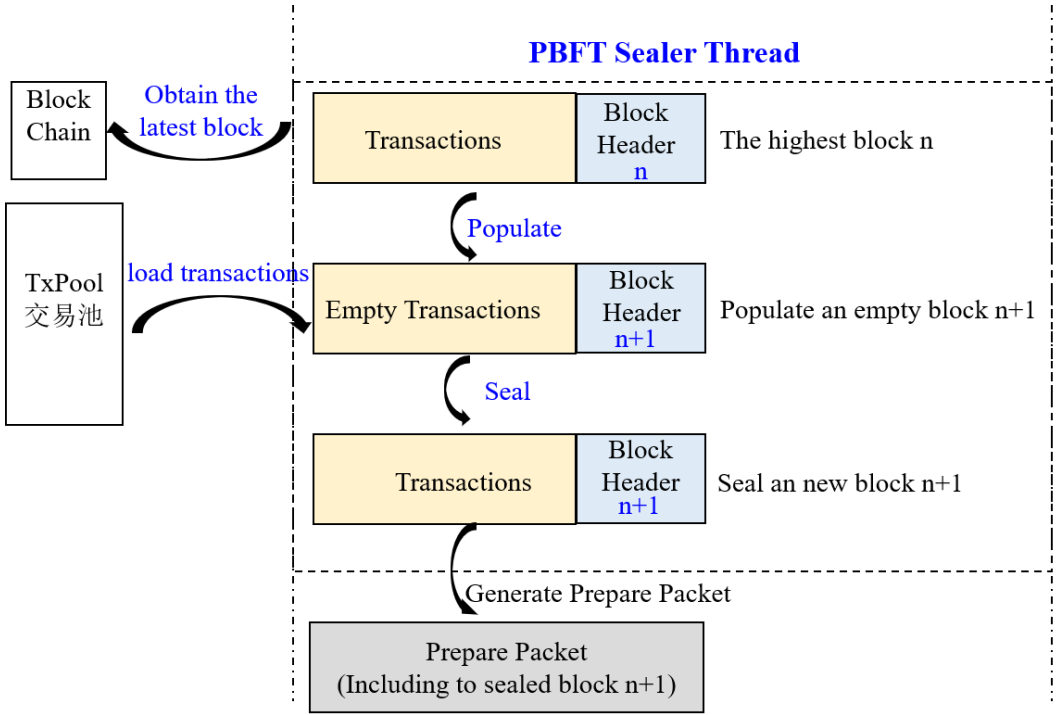
- 对于SDK来的交易，广播给所有的节点
- 对于其它节点广播来的交易，随机选择25%的节点再次广播
- 一条交易在一个节点上，只广播一次，当收到了重复的交易，不会进行二次广播

交易打包共识框架



- 抽象共识流程，统一打包共识框架，支持扩展任意共识算法
- 集成支持PBFT和Raft两种打包共识算法

交易打包（以PBFT为例）



1. **产生新的空块**: 通过区块链(BlockChain)获取当前最高块，并基于最高块产生新空块

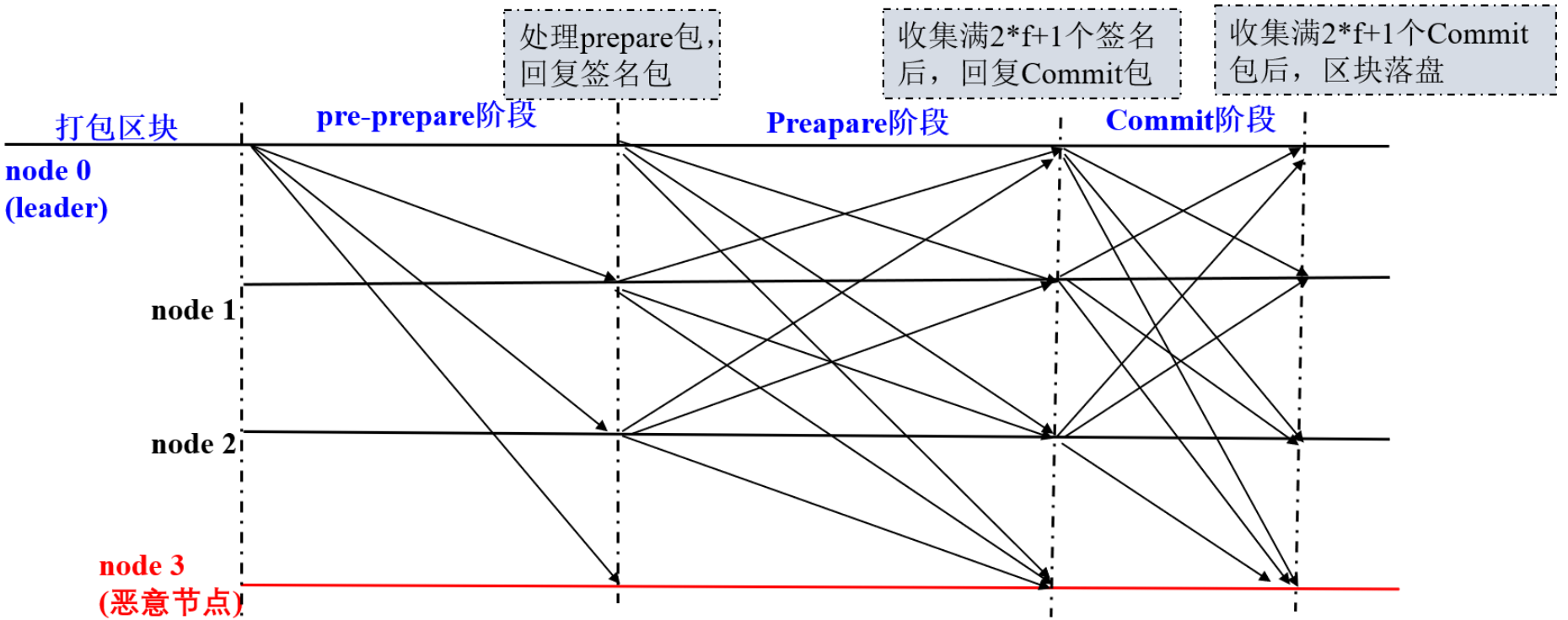
2. **从交易池打包交易**: 产生新空块后，从交易池中获取交易，并将获取的交易插入到产生的新区块中；

3. **组装新区块**: Sealer线程打包到交易后，将新区块的打包者(Sealer字段)置为自己索引，并根据打包的交易计算出所有交易的transactionRoot；

4. **产生Prepare包**: 将组装的新区块编码到Prepare包内，通过PBFTEngine线程广播给组内所有共识节点，其他共识节点收到Prepare包后，开始进行三阶段共识

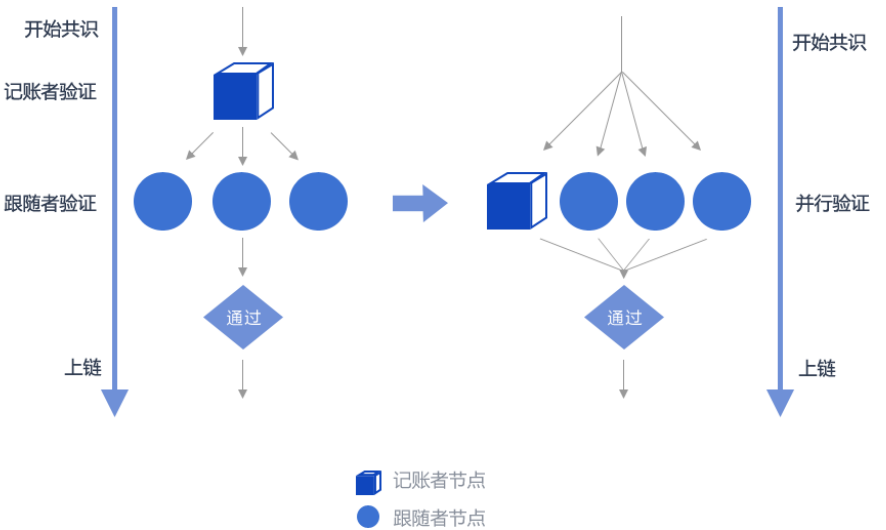
PBFT共识算法

- 广播模型，三次广播，容错1/3节点故障



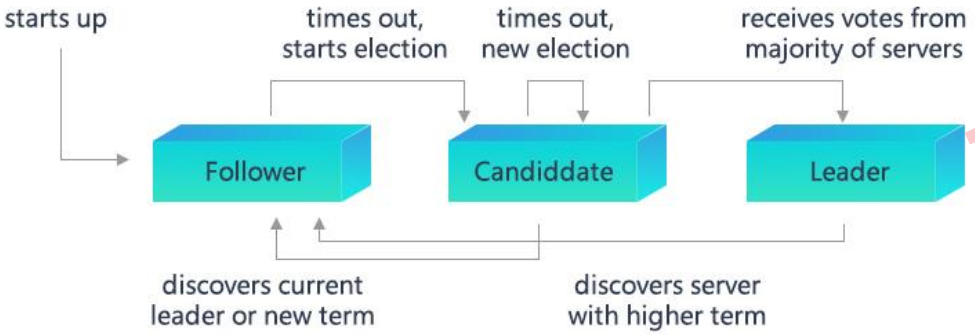
高性能的PBFT算法

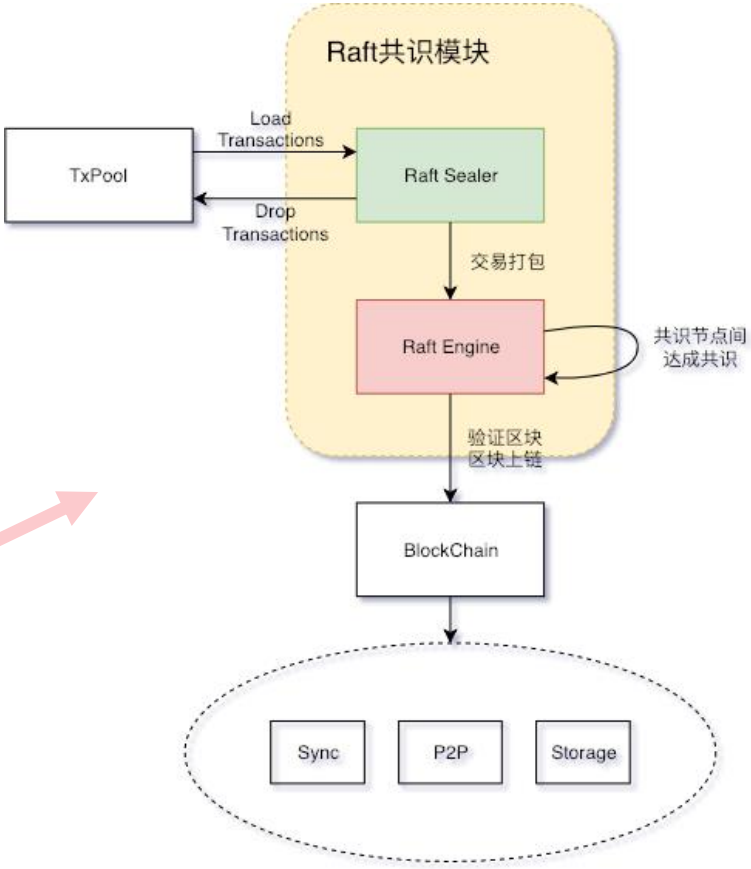
- 记账者与跟随者同时并行计算，大幅提升交易处理速度
- 不出空块，减少存储量，加快同步速度
- 加速记账节点的互相检测，异常时可快速切换到下一个记账者



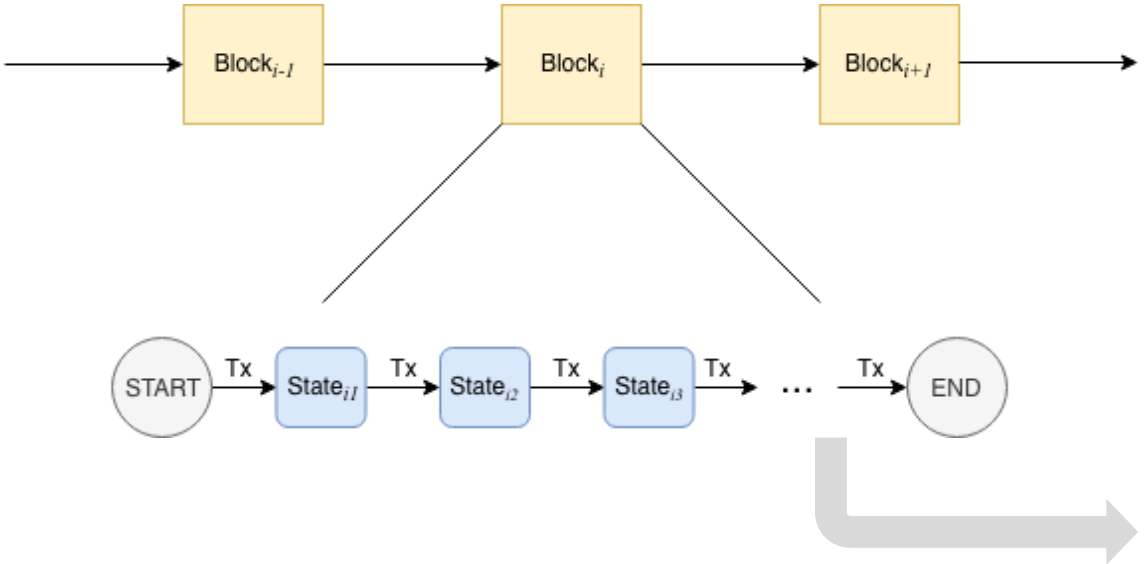
Raft共识算法

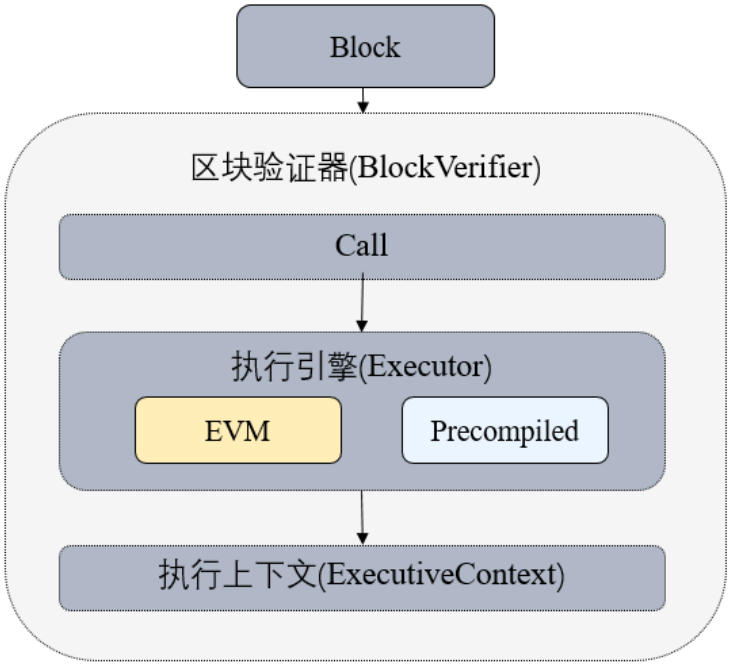
- 选定一个节点作为Leader，Leader负责打包区块
- 其他节点同步Leader区块
- 支持节点状态切换，容错1/2节点故障





交易执行器架构

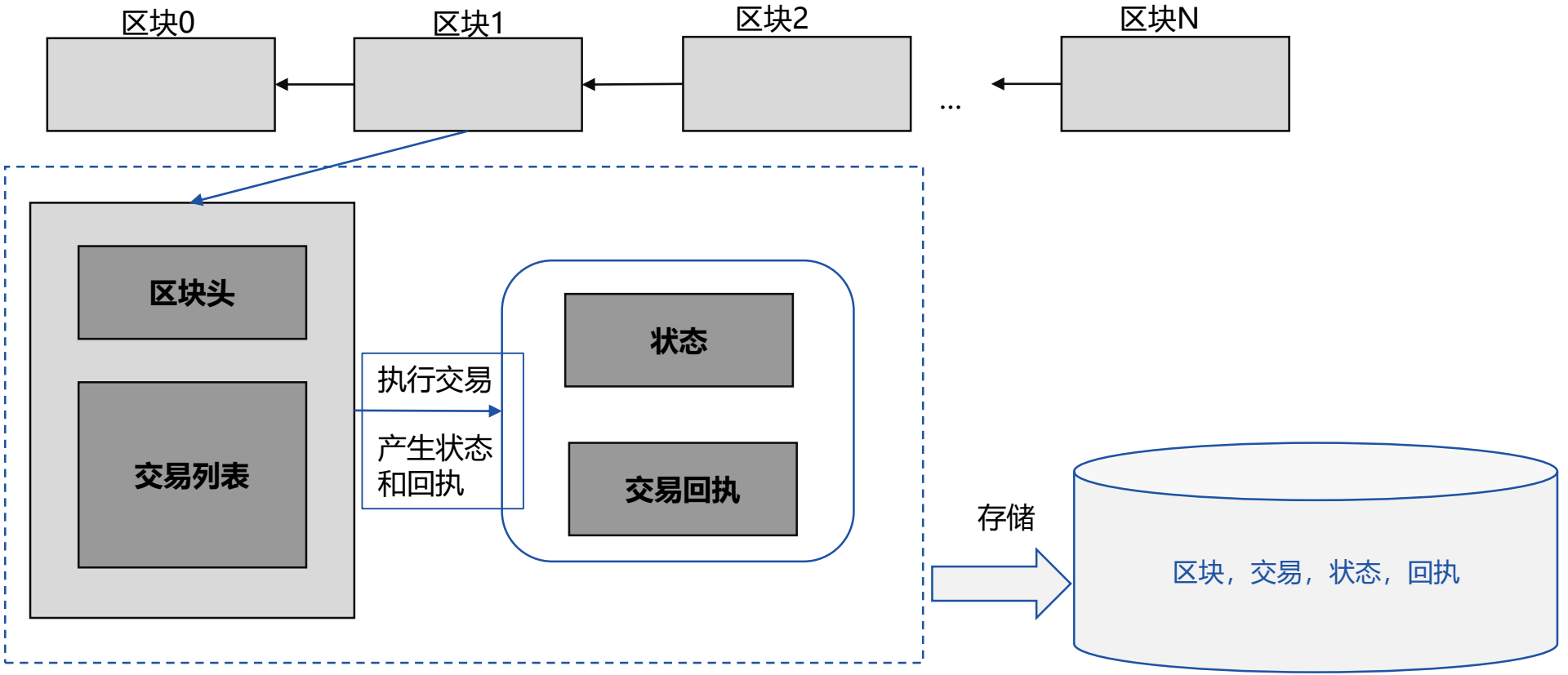




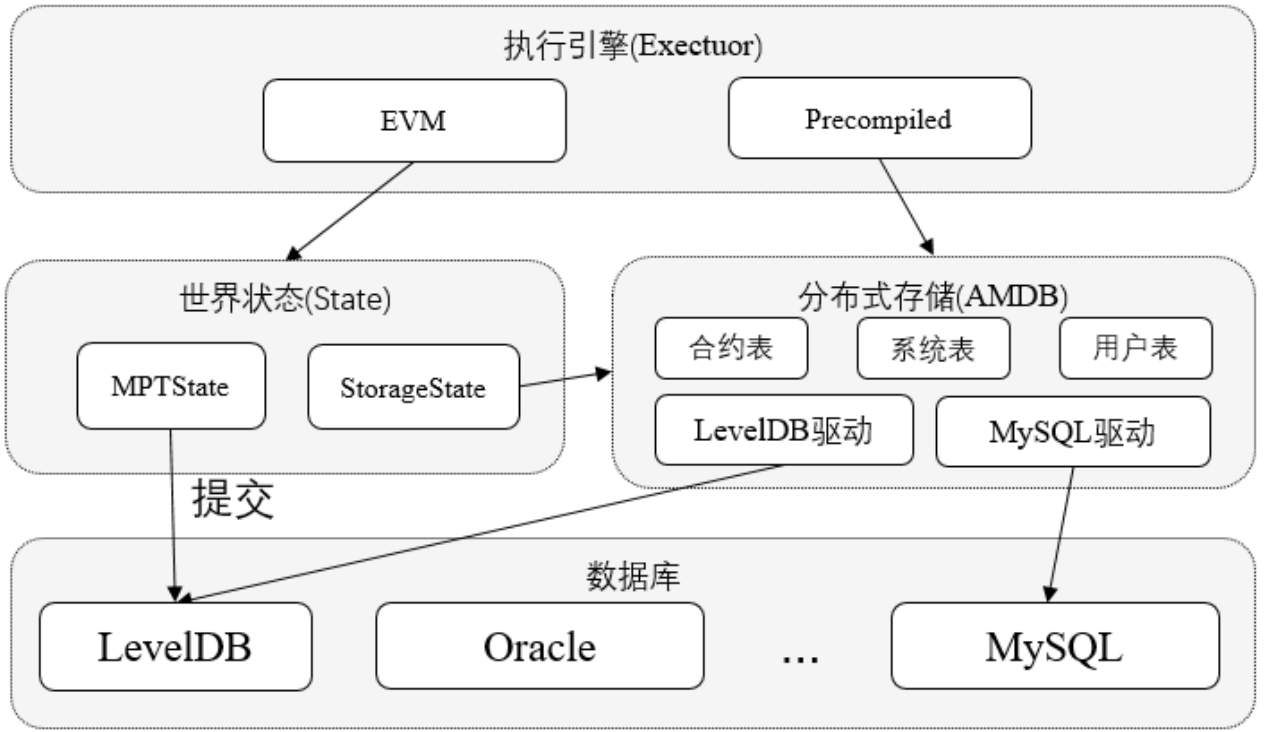
- 抽象交易执行处理框架，支持多引擎
- 集成支持EVM和Precompiled引擎

What's new in FISCO-BCOS 2.0?

区块链数据

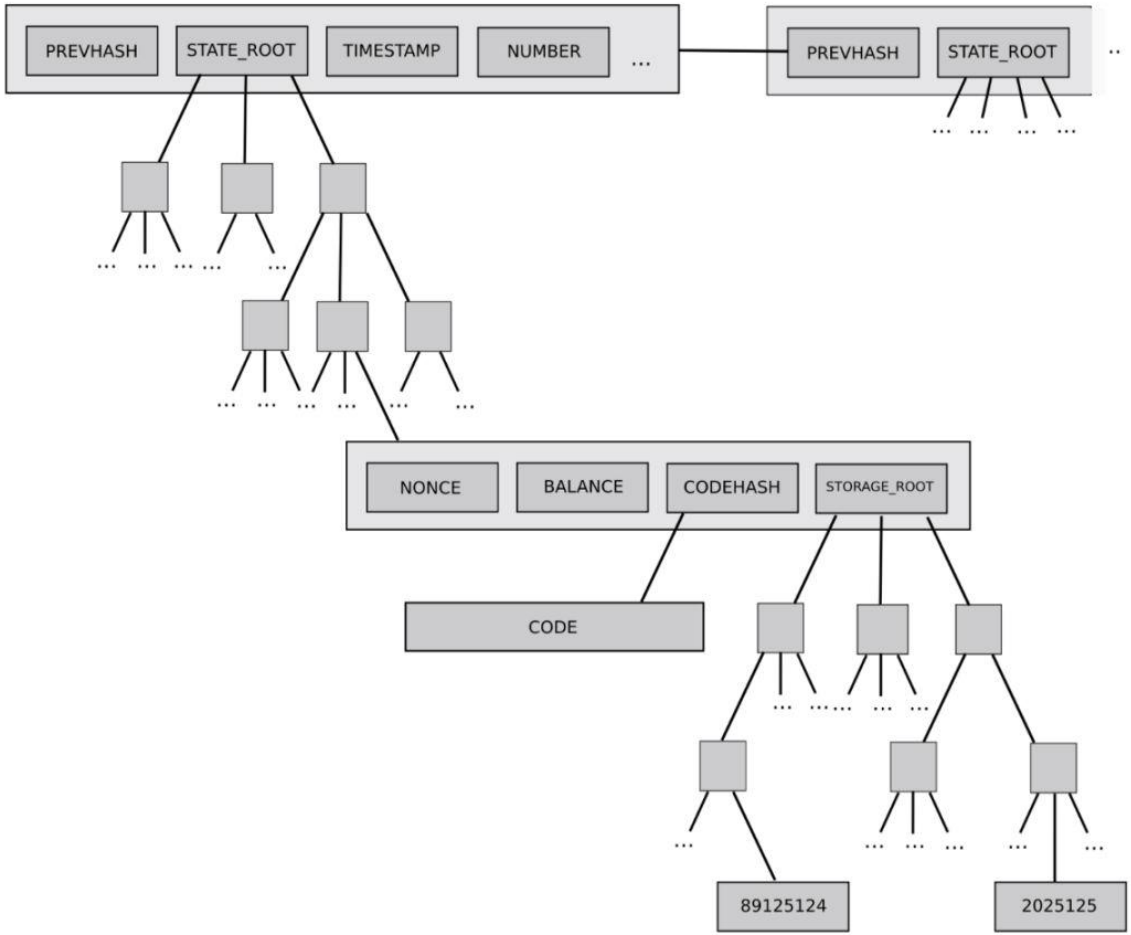


分布式存储架构设计



- 通过抽象表结构，实现了SQL和NoSQL的统一
- 通过驱动层，实现可适配多种底层数据库引擎
- 根据场景需要，提供多种数据存储模式

树结构——MPTState



- 存储全局历史状态，追溯方便
- Merkle树结构，Hash计算开销很大，存储消耗很大
- 随着数据增大，MPT存储访问效率下降很快

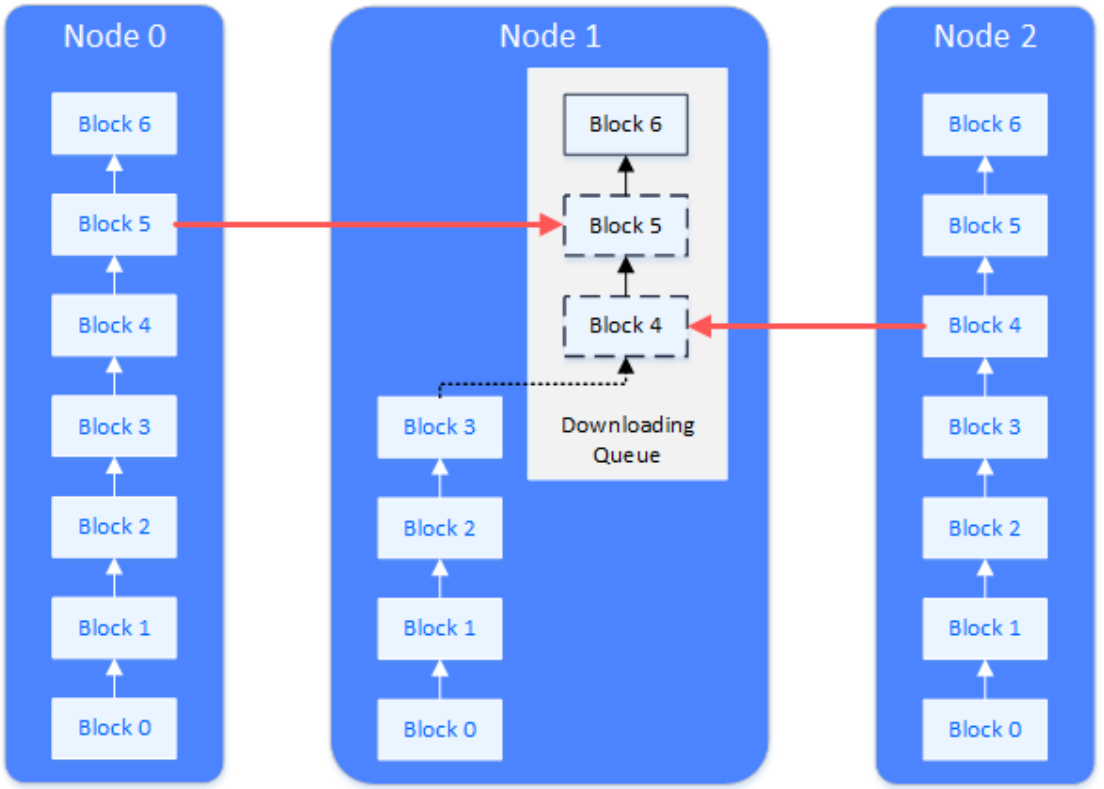
表结构——StorageState

_contract_data_+Address+_作为表名，表中存储外部账户相关信息，表结构如下：

key*	value
balance	账户余额（可选）
nonce	账户计数
code	智能合约代码（可选）
codeHash	智能合约代码HASH（可选）
alive	账户是否有效

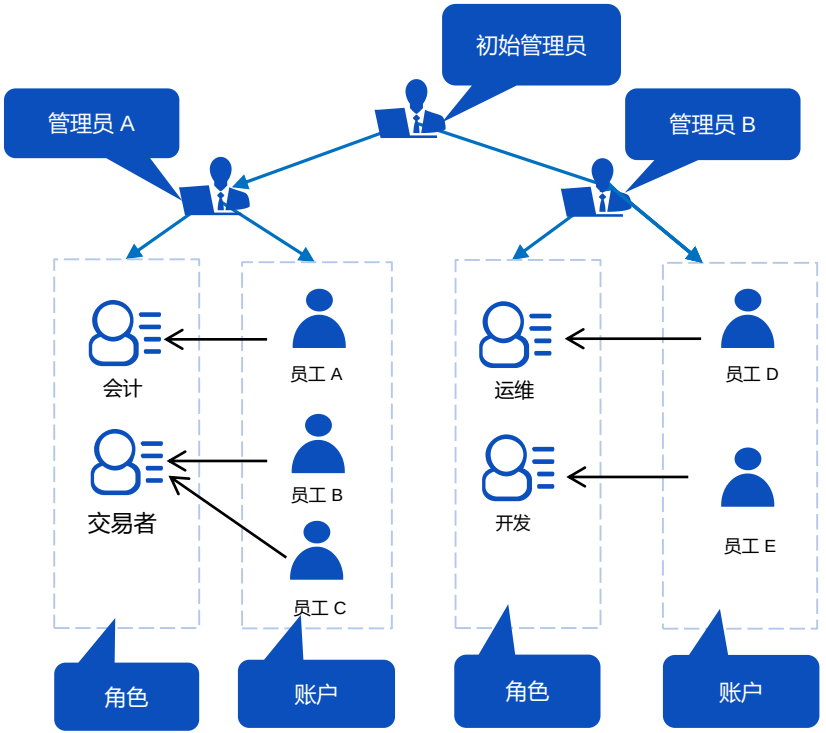
- 性能更高，更适合业务使用，数据管理更简便
- 牺牲了一定的追溯便利性

区块同步



- 节点之间实时同步更新状态，感知链上最新区块高度
- 状态落后将启动区块同步流程
- 并行下载，极速同步到最新状态

权限控制设计



- 开发，管理，交易，审计，运维等多种角色
- 给角色规划分配不同的权限
- 一个账户对应到某一种角色
- 拒绝未授权的访问
- 可以审计各账户发起的操作

权限控制实现

- 权限控制的最小粒度为表，基于外部账号进行控制。
- 使用白名单机制，未配置权限的表，默认完全放开，即所有外部账户均有读写权限。
- 权限设置利用权限表（_sys_table_access_）。权限表中设置表名和外部账户地址，则表明该账户对该表有读写权限，设置之外的账户对该表仅有读权限。
- 采用控制台，可以方便地进行权限控制的设置

设置权限示例

进入账户1登录的控制台，设置外部账号1拥有部署合约和创建用户表的权限。

```
[group:1]> grantDeployAndCreateManager 0xf1585b8d0e08a0a00fff662e24d67ba95a438256
{
        "code":0,
        "msg":"success"
}
```

```
[group:1]> listDeployAndCreateManager
-------------------------------------------------------------------------------------------------
|                    address                    |                enable_num
| 0xf1585b8d0e08a0a00fff662e24d67ba95a438256  |                     1
-------------------------------------------------------------------------------------------------
```

合约部署示例

- **外部账号1部署合约：**

```
$ ./run.sh 1 1 deploy

tx.origin = 0xf1585b8d0e08a0a00fff662e24d67ba95a438256

deploy contract address: 0x31877d5864125b3aa3a5ae60022274d1a4130d00
deploy contract successful!
```

外部账号1部署合约成功，有权限部署合约。

- **外部账号2部署合约：**

```
$ ./run.sh 2 1 deploy

tx.origin = 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d

non-authorized to deploy contracts!
```

相关配置

- 配置文件位于nodeX/conf目录下；
- 群组系统配置文件group.x.genesis
- 群组可变配置文件group.x.ini

配置详情

群组系统配置文件group.x.genesis:

- group ID
- 共识配置
  - ✓ 共识算法
  - ✓ 最大交易数量
  - ✓ 群组共识节点
- 存储配置
  - ✓ levelDB：type=levelDB
  - ✓ 使用mysql：type=external 并且topic=DB
- state配置
  - ✓ 支持MPT
  - ✓ 和storage（推荐）
- gax_limit，最大可消耗gas

配置详情 II

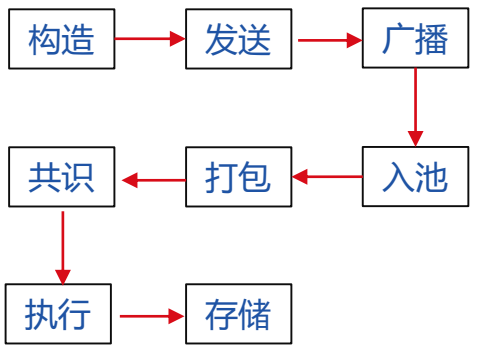
群组可变配置文件group.x.ini:

- ttl：收到共识消息后最大转发次数（ttl-1），仅对PBFT有效；
- tx_pool：交易池大小
- enable_parallel：是否使用并行计算
- enable_dynamic_block_size：是否允许动态调整区块包含交易数量

本课程回顾

FISCO BCOS的总体架构设计

交易生命周期



FISCO BCOS核心模块

更多FISCO BCOS相关内容

- 合约开发详细教程，以及最佳实践
- 性能方面的优化——并行计算模型
- 安全方面的设计——多维度安全体系
- 隐私保护方面的研究——零知识证明，群签名，环签名，同态加密等
- 系统易用性的设计——外围工具
- 联盟链治理——监控，运维，数据治理等
- 平台化解决方案——WeBase
- 分布式身份管理方案——WeIdentity
- 分布式消息中间件——WeEvent
- 应用案例

微众银行/金链盟区块链开源产品：Please Star

- **WeIdentity：实体身份标识 x 可信数据交换**
  - https://github.com/WeBankFinTech/WeIdentity
- **WeEvent：基于区块链的事件驱动架构**
  - https://github.com/WeBankFinTech/WeEvent
- **WeBASE：区块链中间件平台**
  - https://github.com/WeBankFinTech/WeBASE
- **FISCO-BCOS：**
  - https://github.com/FISCO-BCOS/FISCO-BCOS

WeBank

谢谢！