



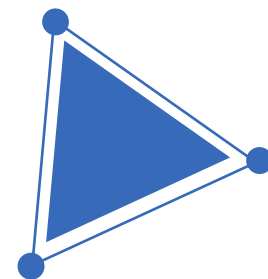
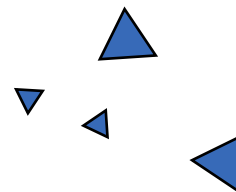
# 区块链与数字货币

浙江大学 杨小虎

2021年1月4日

# 05 超级账本技术分析

---



# 比特币之后的数字货币与区块链发展

- 竞争币、山寨币
  - Litecoin 莱特币
  - Dogecoin 狗狗币
- 新一代公共区块链平台
  - Ethereum 以太坊
  - EOS
  - .....
- 面向企业应用的区块链平台
  - R3 CEV
  - HyperLedger
  - Hyperchain

- 开发策略
  - 复制、修改开源代码
  - 全新开发
- 技术创新
  - 可编程、可开发
  - 共识算法
  - .....
- 融资创新
  - 代币发行融资（ICO）



# 超级账本 HyperLedger

- 创立于2016年，由30个创始公司成员和一套技术和组织治理机构组成。
- 是一个为了提高跨行业的区块链技术的开源合作项目。它是由Linux基金会主导的全球合作项目，包括了金融、银行、物联网、供应链、制造和科技产业的领导者。
- 宣称“自从互联网诞生以来，除了互联网本身，没有比区块链技术更广泛、更根本的革命性技术了”。
- HyperLedger Fabric，一个许可区块链平台，带有chaincode，由Digital Asset和IBM提出。
- 网站：<https://www.hyperledger.org/>



# The Hyperledger Greenhouse

## Business Blockchain Frameworks & Tools Hosted by Hyperledger



### Distributed Ledgers



Java-based  
Ethereum client



Permissionable smart  
contract machine (EVM)



Enterprise-grade DLT  
with privacy support



Decentralized identity



Mobile application focus



Permissioned & permissionless  
support; EVM transaction family

### Libraries



### Tools



### Domain-Specific



2019年



## What is Hyperledger?

Hyperledger is an open source community focused on developing a suite of stable frameworks, tools and libraries for enterprise-grade blockchain deployments.

It serves as a neutral home for various distributed ledger frameworks including Hyperledger Fabric, Sawtooth, Indy, as well as tools like Hyperledger Caliper and libraries like Hyperledger Ursa.



2020年



# HyperLedger子项目

项目名称
HyperLedger Aries
HyperLedger Avalon
HyperLedger Besu
HyperLedger Burrow
HyperLedger Cactus
HyperLedger Caliper
HyperLedger Cello
HyperLedger Explorer

项目名称
HyperLedger Fabric
HyperLedger Grid
HyperLedger Indy
HyperLedger Iroha
HyperLedger Quilt
HyperLedger Sawtooth
HyperLedger Transact
HyperLedger Ursa

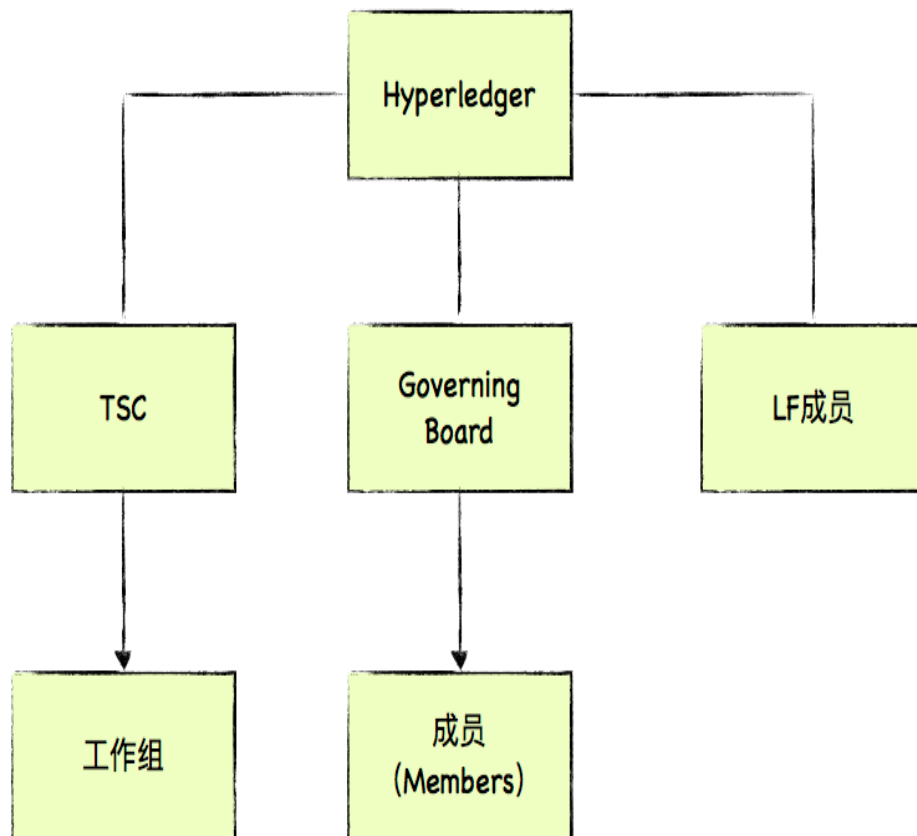


# 社区组织结构

技术委员会：（Technical Steering Committee, TSC）：负责技术相关的工作，下设多个工作组，具体带动各个项目和方向的发展；

管理董事会（Governing Board）：负责社区组织的整体决策，由超级账本会员中推选出代表；

Linux基金会（Linux Foundation, LF）：负责基金管理，协助Hyperledger社区在Linux基金会的支持下发展。



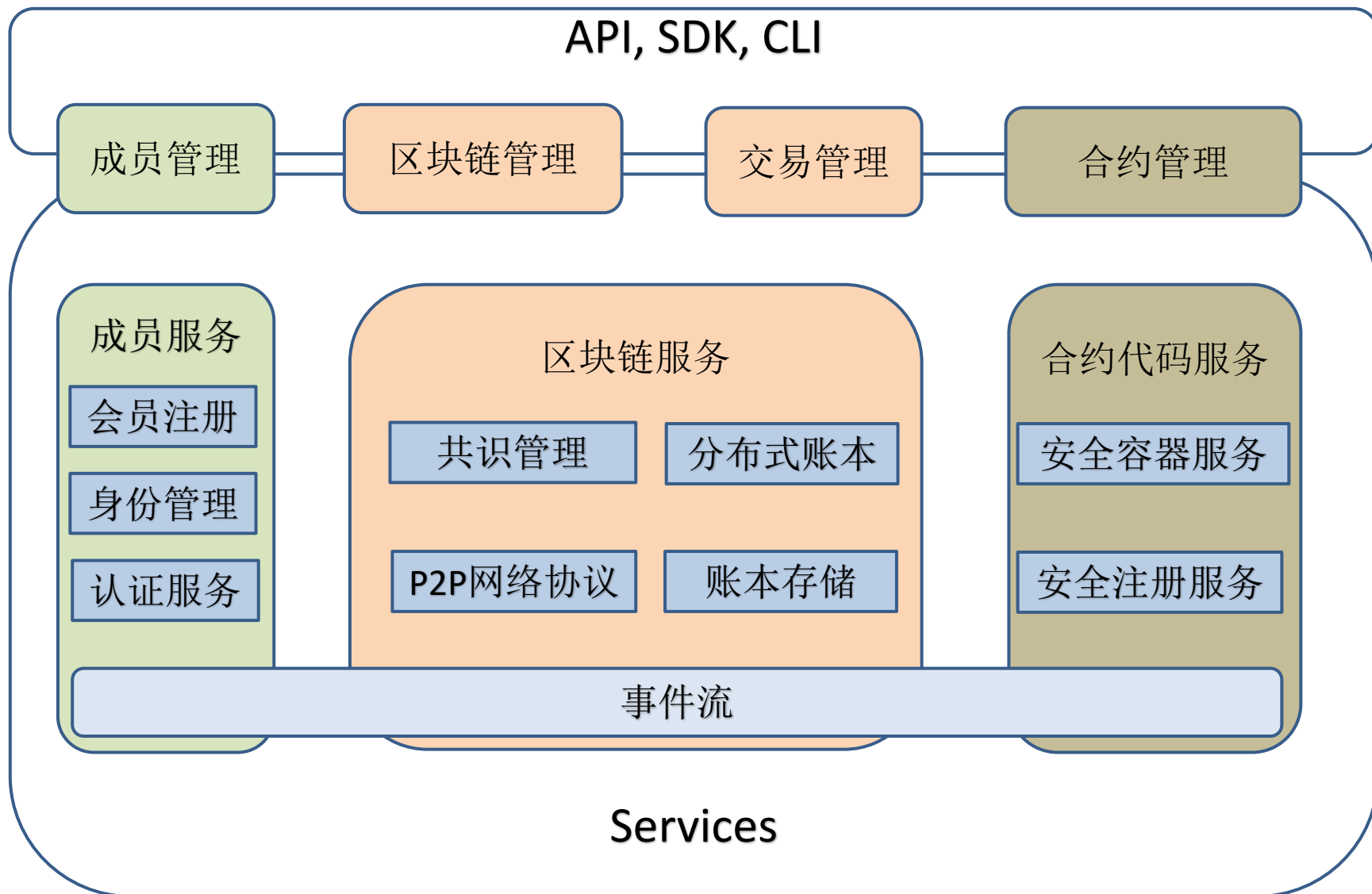


# HyperLedger Fabric

- 联盟区块链、许可区块链
- Hyperledger Fabric is intended as a foundation for developing applications or solutions with a modular architecture. Hyperledger Fabric allows components, such as **consensus and membership services, to be plug-and-play**. Its modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy.

Fabric是一个面向企业应用的区块链软件，不是公链，不发行加密货币。





# Fabric架构

- **会员服务 (Membership Service)** : 会员注册、身份保护、内容保护、交易审计功能
- **区块链服务 (Blockchain Service)** : 节点的共识管理、账本的分布式计算、账本的存储以及节点间的P2P协议功能的实现
- **合约代码服务 (Chaincode Service)** : 提供智能合约的服务

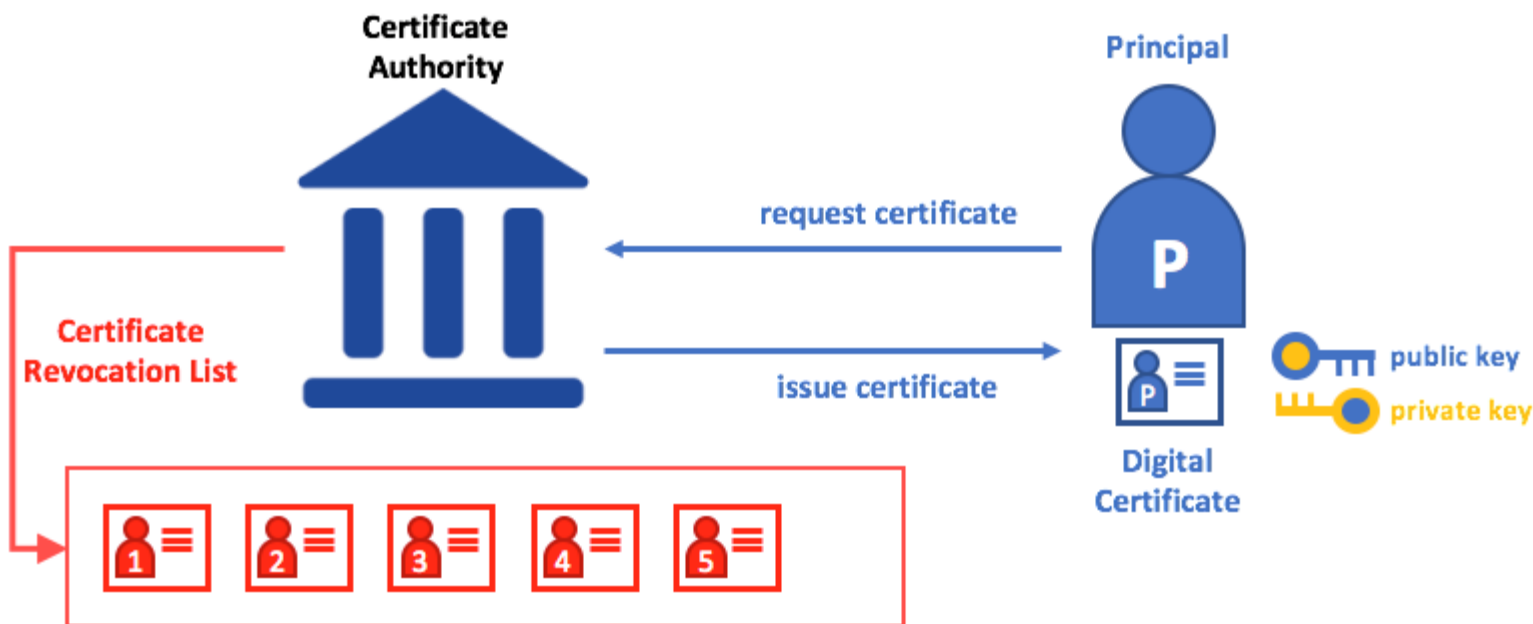


# 成员服务

- Hyperledger Fabric 支持一个交易网络，在这个网络中，所有参与者都拥有已知的身份。
- 公钥基础设施（PKI）用于生成与组织、网络组件以及终端用户或客户端应用程序相关联的加密证书。可以在更广泛的网络和通道级别上操纵和管理数据访问控制。
- 成员服务提供者（MSP）分发证书、验证证书和用户授权背后的所有加密机制和协议抽象出来。MSP可以定义它们身份概念，同样还可以定义管理(身份验证)和认证(签名生成和验证)这些身份的规则。



# 成员服务：CA体系

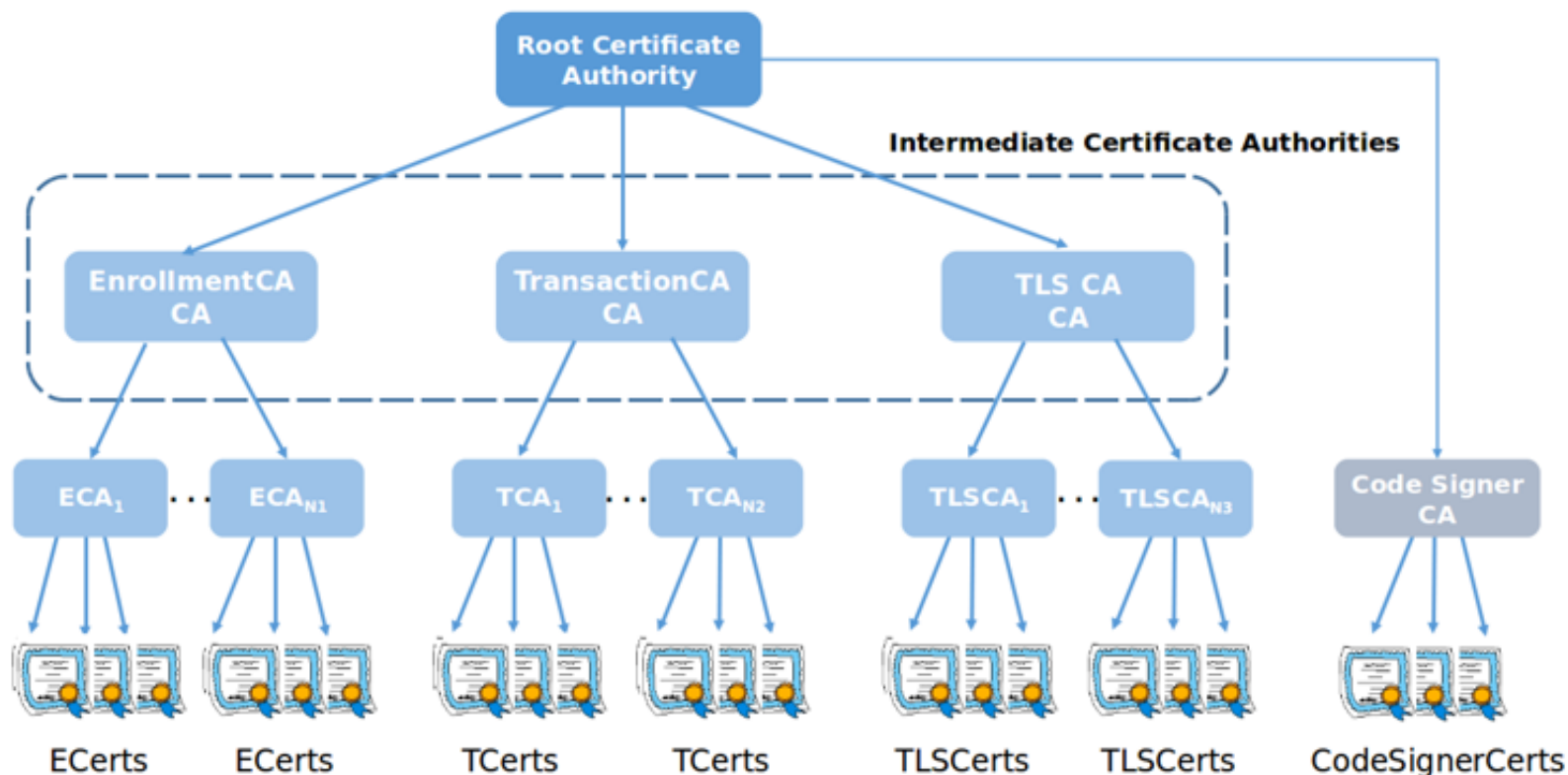


PKI 由向各方（如服务的用户、服务提供者）发布数字证书的证书授权中心组成，各方使用证书在与其环境交换的消息中对自己进行身份验证。



# 成员服务：CA体系

## Public Key Infrastructure - Hierarchy



Fabric有一个独立的Fabric CA模块，用来管理证书服务，专门进行证书发放、管理和身份验证，当然Fabric也允许第三方CA机构的接入。

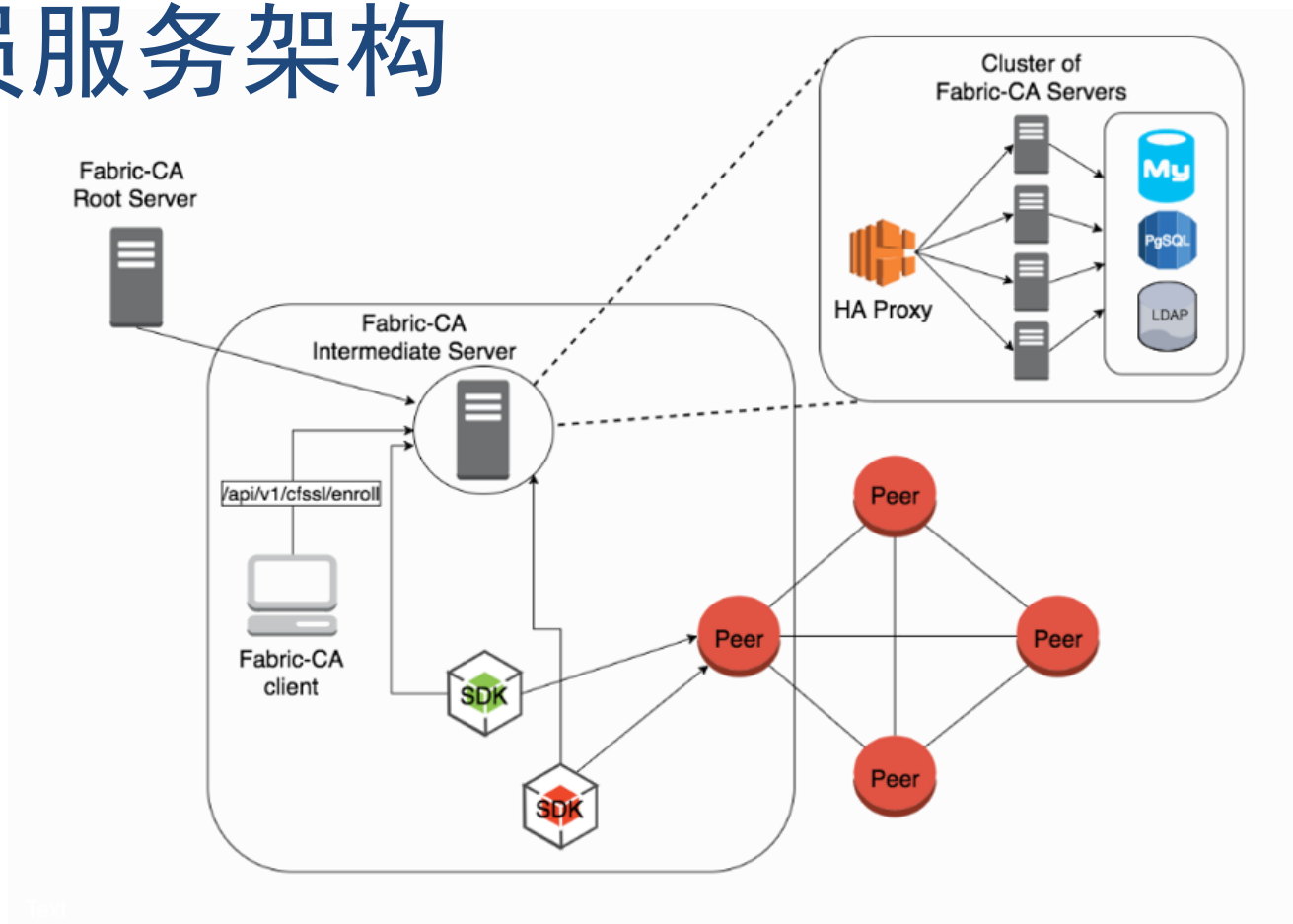


# 成员服务：CA证书

- 一个自签名 (X.509) CA 证书列表来组成信任根 (*root of trust*)
- 一个X.509证书列表来代表证书验证时需要考虑的中间证书，这些证书应该由某一个信任根颁发；中间证书是可选的参数
- 一个X.509证书列表，并拥有从某一信任根起可验证的 CA 证书路径，来代表该MSP的管理员证书；拥有管理员证书则代表拥有申请改变该MSP配置的权力(例如，根CA、中间CA)
- 一个组织单位列表，此列表应出现在该MSP的有效成员的X.509证书中；这是一个可选的配置参数，举例来说，可用于多组织使用相同信任根和中间CA，并给其成员预留OU信息
- 一个证书撤销列表(CRLs)，其中每一个对应一个列出的(根或中间)MSP CA；这是一个可选参数
- 一个自签(X.509)证书列表，用来组成TLS证书的信任根(*TLS root of trust*)
- 一个X.509证书列表来代表证书验证时需要考虑的TLS中间证书，这些证书应该由某一个TLS信任根颁发；TLS中间证书是可选的参数



# 成员服务架构



Fabric CA提供客户端和SDK两种方式与CA进行交互，每个Fabric CA都有一个根CA或者中间CA。为了保证CA的安全性，中间CA可以采用集群方式搭建。





# 区块链服务

区块链服务用于维护分布式账本。区块链服务包括P2P协议、分布式账本和共识机制管理。

- 1. P2P协议** Fabric网络中，Peer和Orderer采用gRPC（Google RPC）对外提供远程服务，供客户端进行调用。网络中的节点之间通过Gossip 协议来进行状态同步和分发。
- 2. 共识机制** Fabric允许根据实际业务需要选择合适的共识机制，目前支持SOLO、Kafka、Raft三种共识机制。
- 3. 分布式账本** 分布式账本包括两个组件：世界状态（world state）、事务日志，分布式账本是世界状态数据库和事务日志历史记录的组合。世界状态（world state）组件记录的是最新的分布式账本状态，事务日志组件记录的是世界状态的更新历史。
- 4. 账本存储** 支持LevelDB（由Google公司研发的键值对嵌入式数据库管理系统）和CouchDB（由Apache软件基金会开发的一个面向文档的开源数据库管理系统）。



# Fabric的节点

- **Client**: 最终用户, 至少连接的一个peer节点或一个Orderer节点, 一般只保存与自己有关的账户数据
- **Orderer**: **编排节点**, 接收包含背书签名的交易, 对未打包的交易进行排序生成区块, 并广播给Peer节点
- **Peer**: **对等节点**, 负责通过执行链码 (chaincode) 实现对账本的读写操作, 所有的Peer节点都是**提交节点 (Committer)**, 负责维护状态数据和账本的副本
- **Endorser**: **背书节点**, 部分Peer节点根据背书策略的设定会执行交易并对结果进行签名的背书, 充当了背书节点 (Endorser) 的角色。背书节点是动态的角色, 每个链码在实例化的时候都会设置背书策略, 指定哪些节点对交易背书后才是有效的。只有在应用程序向节点发起交易背书请求的时候该Peer节点才是背书节点, 否则它就是普通的记账节点。



# Fabric 的节点

- 所有的 Peer 节点都是一样的，Peer 节点能够担当多个角色。
- **提交节点**（Committer），每个 Peer 节点都是一个提交节点。他们会接收生成的区块，在这些区块被验证之后会以附加的方式提交到 Peer 节点的账本副本中。
- **背书节点**（Endorser），每个安装了智能合约的 Peer 节点都可以作为一个背书节点。然而，想要成为一个真正的背书节点，节点上的智能合约必须要被客户端应用使用，来生成一个被签名的交易响应。
- 智能合约的背书策略明确了在交易被接受并且记录到提交节点的账本之前，需要哪些组织的 Peer 节点为交易签名。



# Fabric的节点

- Peer 节点还可以担任的两种其他的角色:
- **主节点**。当组织在通道中具有多个 Peer 节点的时候，会有一个主节点，它负责将交易从排序节点分发到该组织中其他的提交节点。
- **锚节点**。如果一个 Peer 节点需要同另一个组织的 Peer 节点通信的话，它可以使用对方组织通道配置中定义的锚节点。一个组织可以拥有0个或者多个锚节点，并且一个锚节点能够帮助很多不同的跨组织间的通信。

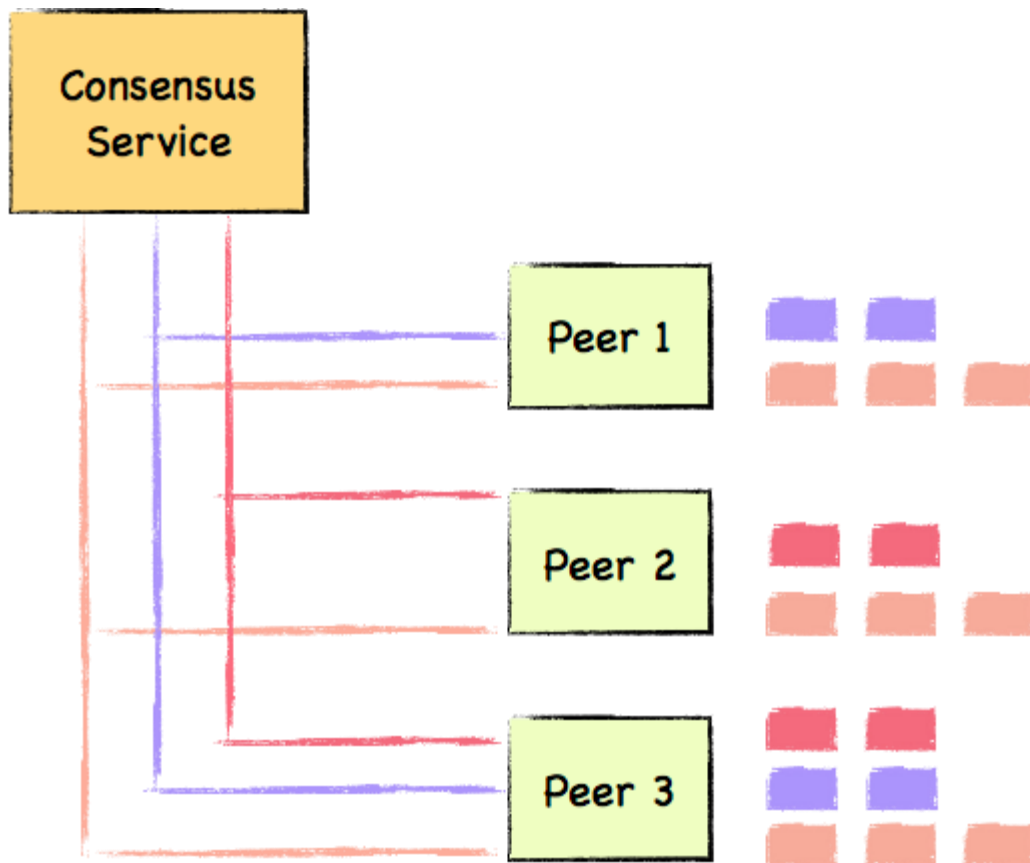




# Fabric的通道和多链

- Channel：通道，是构建在Fabric区块链网络上的由若干个节点所组成的一条链，实现了数据的私有和保密。
- 一个Peer节点至少接入一个通道，可以接入多个通道。每个通道拥有自己单独的账本，且仅对通道成员节点共享。
- Orderer节点拥有的通道成为系统通道，该通道账本拥有网络上所有账本数据。

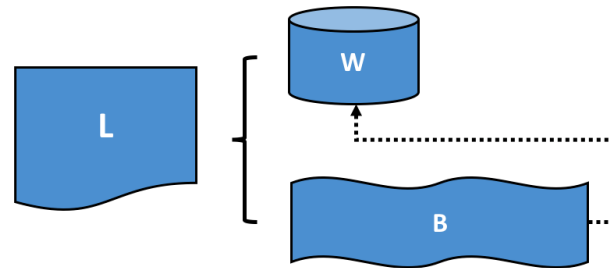









- 通道：通道提供一种通讯机制，将Peer和Orderer连接在一起，形成一个个具有保密性的通讯链路（虚拟）
- Fabric的区块链网络缺省包含一个账本（称为：系统账本）和一个通道；
- 子账本可以被创建，并绑定到一个通道

# Fabric账本

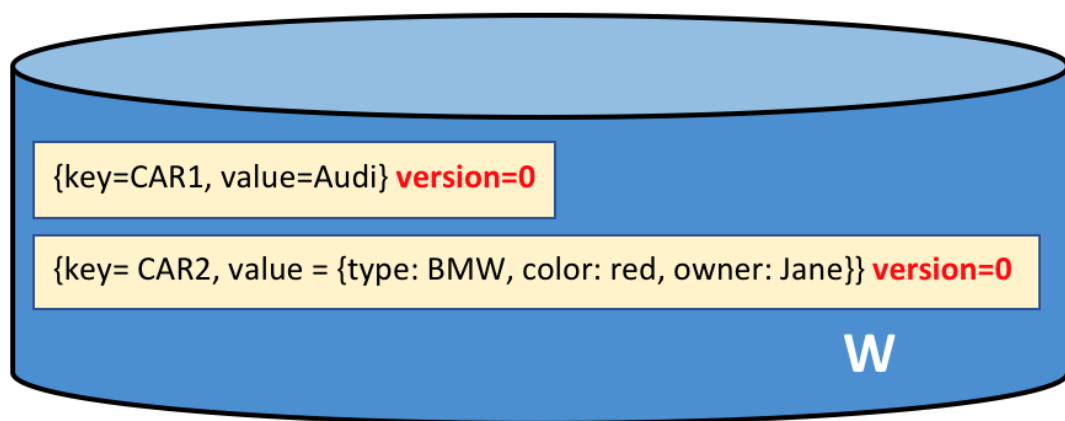
- Hyperledger Fabric 中的账本由“**世界状态**”和“**区块链**”这两部分组成，它们彼此不同但却相互关联。二者都代表了与业务对象有关的一些事实。
- 世界状态**是一个数据库，它存储了一组账本状态的**当前值**。默认情况下，账本状态是以**键值对**的方式来表示的。因为我们可以创建、更新和删除状态，所以**世界状态能够频繁更改**。
- 区块链**是交易的日志，它记录了促成当前世界状态的所有改变。交易被收集在附加到区块链的区块中，能帮助我们理解所有促成当前世界状态的改变的历史。一旦把数据写入区块链，就无法修改，它是**不可篡改**的。


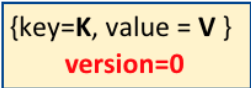
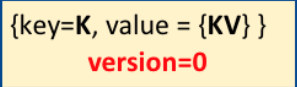


	Ledger
	World State
	Blockchain
	L comprises B and W
	B determines W



# 世界状态



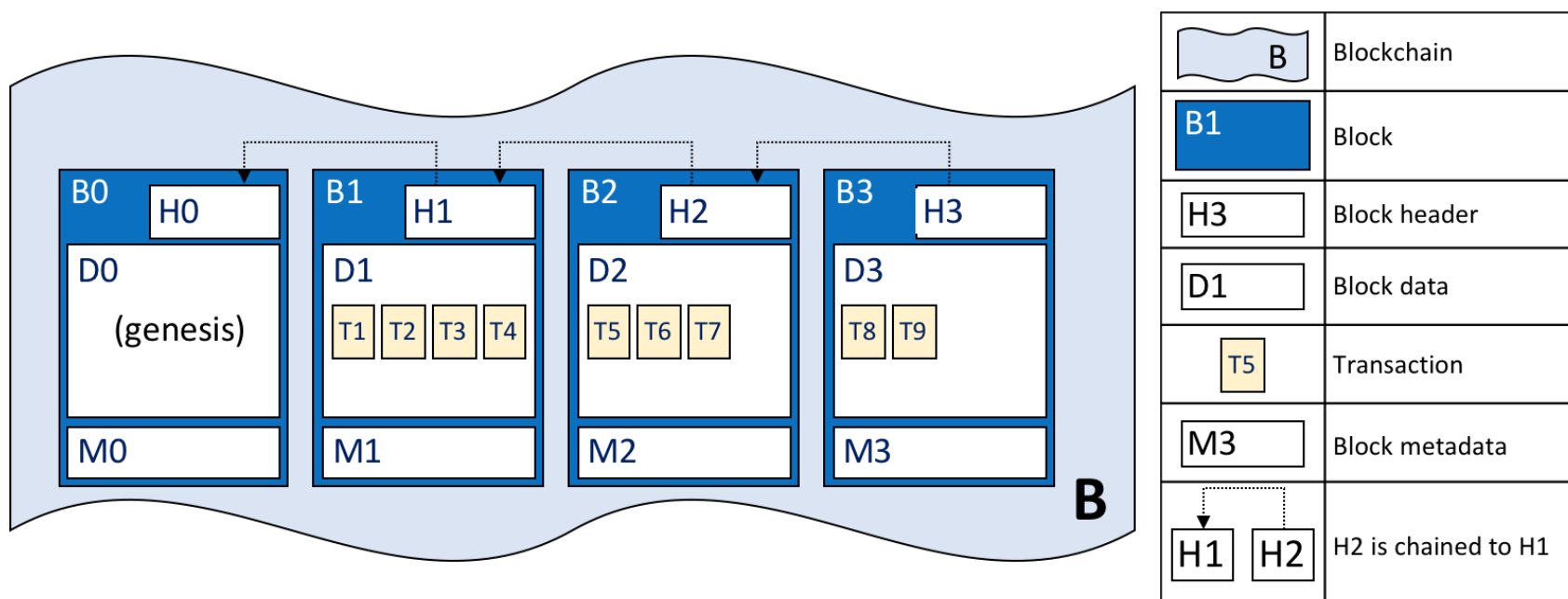
	Ledger world state
	A ledger state with <b>key=K</b> . It contains a set of facts expressed as a simple value, <b>V</b> . The state is at version 0.
	A ledger state with <b>key=K</b> . It contains a set of facts expressed as a set of key-value pairs <b>{KV}</b> . The state is at version 0.

账本状态记录了一组与特定业务对象有关的事实。

采用key-value数据库实现，如LevelDB, CouchDB等。



# 区块链结构

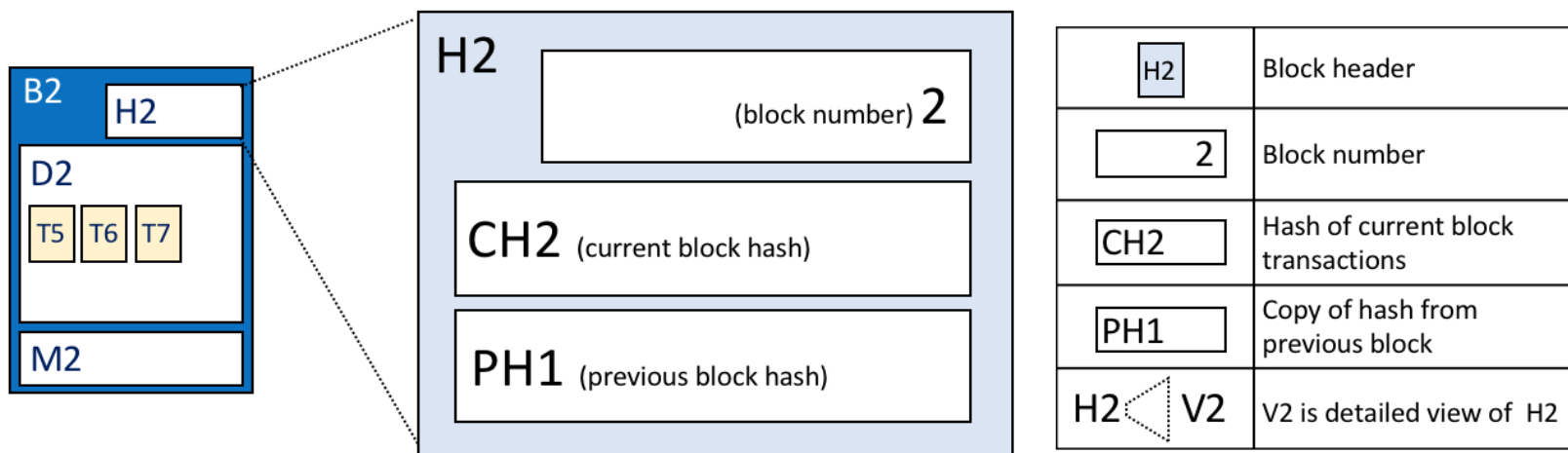


- 区块头
- 区块数据
- 区块元数据

区块数据用文件系统存储



# 区块头



- 区块编号：编号从0（初始区块）开始，每增加一个新区块加1。
- 当前区块的哈希值：当前区块中包含的所有交易的哈希值。
- 前一个区块头的哈希值：区块链中前一个区块头的哈希值。



# 区块数据和区块元数据

- **区块数据**

- 这部分包含了一个有序的交易列表。区块数据是在排序服务创建区块时被写入的。

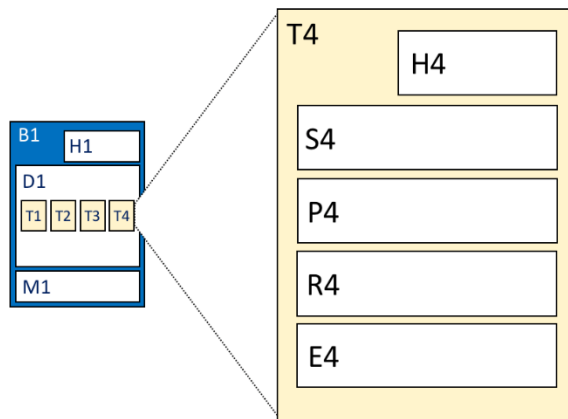
- **区块元数据**

- 这个部分包含了区块被写入的时间，还有区块写入者的证书、公钥以及签名。随后，区块的提交者也会为每一笔交易添加一个有效或无效的标记，但由于这一信息与区块同时产生，所以它不会被包含在哈希中。



# 交易的结构

- **头**, H4, 记录了关于交易的一些重要元数据, 比如, 相关链码的名字以及版本。
- **签名**, S4, 包含了一个由客户端应用程序创建的加密签名。该字段是用来检查交易细节是否未经篡改, 交易签名的生成要用到应用程序的私钥。
- **提案**, P4, 负责对应用程序供给智能合约的输入参数进行编码, 随后该智能合约生成提案账本更新。在智能合约运行时, 这个提案提供了一套输入参数, 这些参数同当前的世界状态一起决定了新的账本世界状态。
- **响应**, R4, 以读写集 (RW-set) 的形式记录下世界状态之前和之后的值。交易响应是智能合约的输出, 如果交易验证成功, 那么该交易会被应用到账本上, 从而更新世界状态。
- **背书**, E4, 是一组签名交易响应, 这些签名都来自背书策略规定的相关组织, 并且这些组织的数量必须满足背书策略的要求。

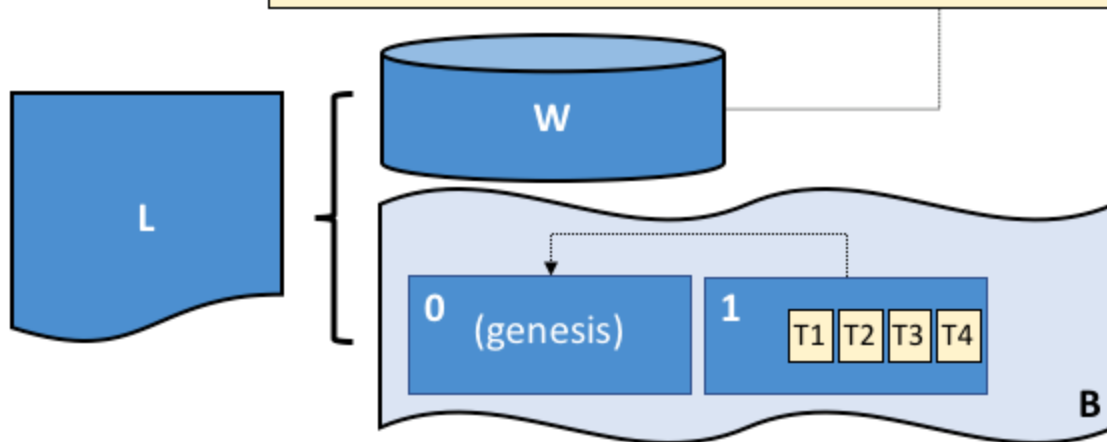


T4	Transaction
H4	Header
S4	Signature
P4	Proposal
R4	Response
E4	Endorsements
T4 ◀ V4	V4 is detailed view of T4



# 账本举例

<b>key</b> =CAR3, <b>value</b> ={color: yellow, make: Volkswagen, model: Passat, owner: Max}	<b>version</b> =0
<b>key</b> =CAR2, <b>value</b> ={color: green, make: Hyundai, model: Tucson, owner: Jin Soo}	<b>version</b> =0
<b>key</b> =CAR1, <b>value</b> ={color: red, make: Ford, model: Mustang, owner: Brad}	<b>version</b> =0
<b>key</b> =CAR0, <b>value</b> ={color: blue, make: Toyota, model: Prius, owner: Tomoko}	<b>version</b> =0



## 链码 (chaincode)

- Fabric没有像以太坊那样有一个自己的虚拟机EVM，而是利用Docker容器，在宿主机上创建一个虚拟环境来运行链上代码，因此链上代码可以是宿主机可以编译执行的各种语言代码。
- 链上代码是图灵完备的，为了防止链上代码执行陷入死循环，为代码执行设置一个计时器，如果运行时间超过预定时间，就杀掉该进程。



# 智能合约示例

Seller Organization

ORG1

**application:**

```
seller = ORG1;  
buyer = ORG2;  
transfer(CAR1, seller, buyer);
```

**car** contract:

```
query(car):  
  get(car);  
  return car;
```

```
transfer(car, buyer, seller):  
  get(car);  
  car.owner = buyer;  
  put(car);  
  return car;
```

```
update(car, properties):  
  get(car);  
  car.colour = properties.colour;  
  put(car);  
  return car;
```

Buyer Organization

ORG2

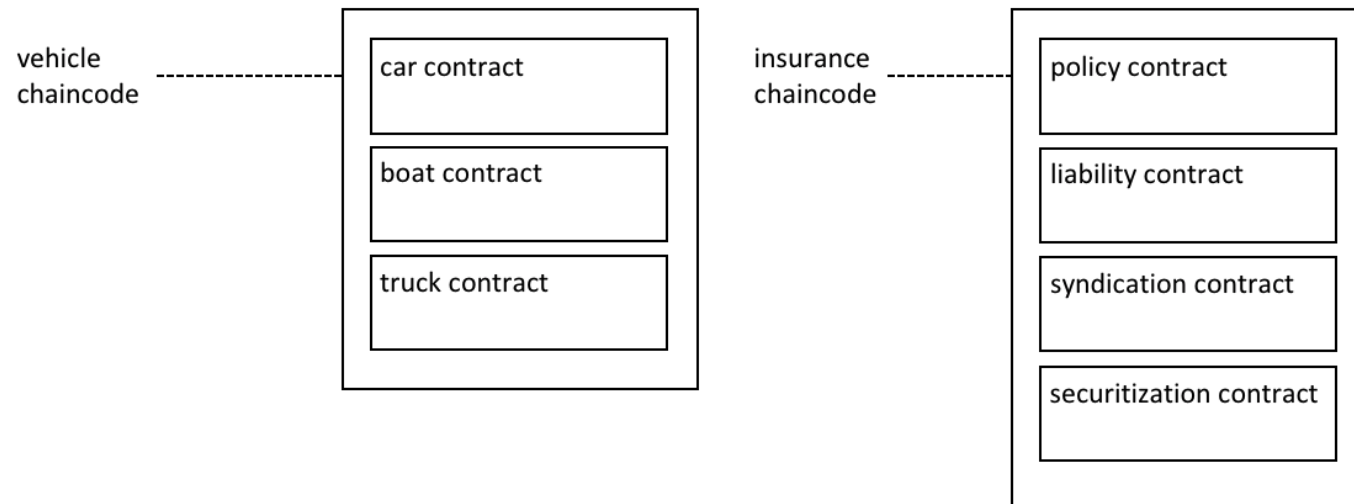
**application:**

```
seller = ORG2;  
buyer = ORG1;  
transfer(CAR2, seller, buyer);
```





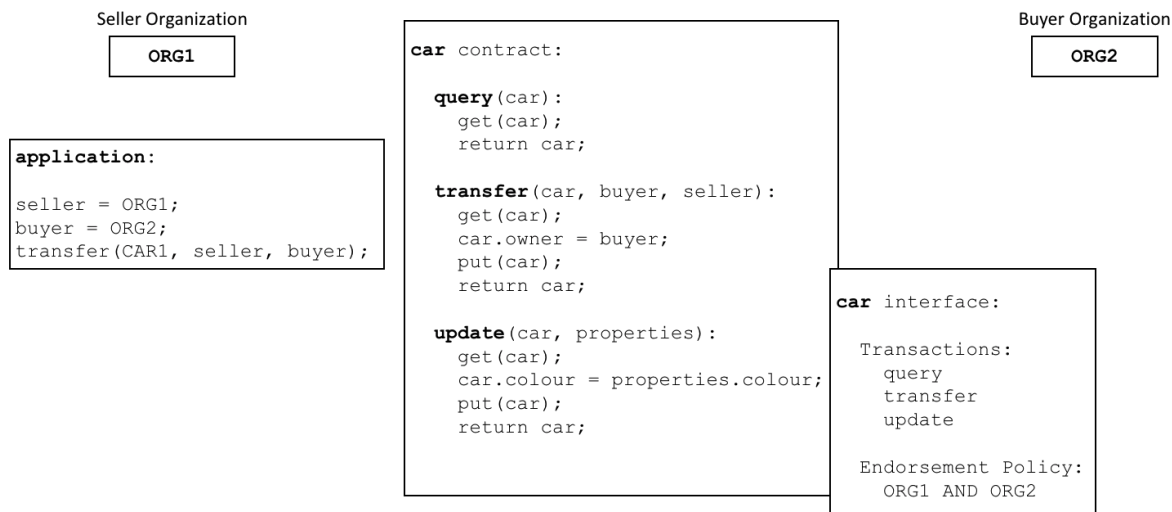
# 智能合约与链码



一个链码可以包含多个智能合约



# 背书



每个链码都有一个背书策略与之相关联，该背书策略适用于此链码中定义的所有智能合约。背书策略非常重要，它指明了区块链网络中哪些组织必须对一个给定的智能合约所生成的交易进行签名，以此来宣布该交易有效。

背书策略是 Hyperledger Fabric 与以太坊（Ethereum）或比特币（Bitcoin）等其他区块链的区别所在。在这些区块链系统中，网络上的任何节点都可以生成有效的交易。而 Hyperledger Fabric 更真实地模拟了现实世界；交易必须由 Fabric 网络中受信任的组织验证。



# 交易验证

- 智能合约提取一组名为**交易提案**的输入参数，并将其与程序逻辑结合起来使用以读写账本。对世界状态的更改被捕获为**交易提案响应**（或简称**交易响应**），该响应包含一个**读写集**，其中既含有已读取的状态，也含有还未书写的新状态（如果交易有效的话）。注意，在执行智能合约时**世界状态没有更新**！
- 一项交易被分发给网络中的所有节点，各节点通过两个阶段对其进行**验证**。首先，根据背书策略检查交易，确保该交易已被足够的组织签署。其次，继续检查交易，以确保当该交易在受到背书节点签名时它的交易读集与世界状态的当前值匹配，并且中间过程中没有被更新。如果一个交易通过了这两个测试，它就被标记为**有效**。所有交易，不管是**有效的**还是**无效的**，都会被添加到区块链历史中，但是仅**有效的**交易才会更新世界状态。



# 智能合约与账本

- 智能合约以编程方式访问账本两个不同的部分：一个是**区块链**（记录所有交易的历史，且记录不可篡改），另一个是**世界状态**（保存这些状态当前值的缓存，是经常需要用到的对象的当前值）。
- 智能合约主要在世界状态中将状态**写入**（put）、**读取**（get）和**删除**（delete），还可以查询不可篡改的区块链交易记录。
  - **读取（get）** 操作一般代表的是查询，目的是获取关于交易对象当前状态的信息。
  - **写入（put）** 操作通常生成一个新的业务对象或者对账本世界状态中现有的业务对象进行修改。
  - **删除（delete）** 操作代表的是将一个业务对象从账本的当前状态中移除，但不从账本的历史中移除。



# Fabric的共识

- 共识不仅仅是就交易顺序达成一致。
- 共识被定义为组成区块的一组交易的正确性的闭环验证。当区块中交易的顺序和结果满足明确的策略标准检查时，最终会达成共识。
- Fabric 通过其在整个交易流程中的基本角色，从提案和背书到排序、验证和提交，突出了这种区别。
- Fabric 的设计依赖于确定性的共识算法，账本不会像其他分布式的以及无需许可的区块链中那样产生分叉。

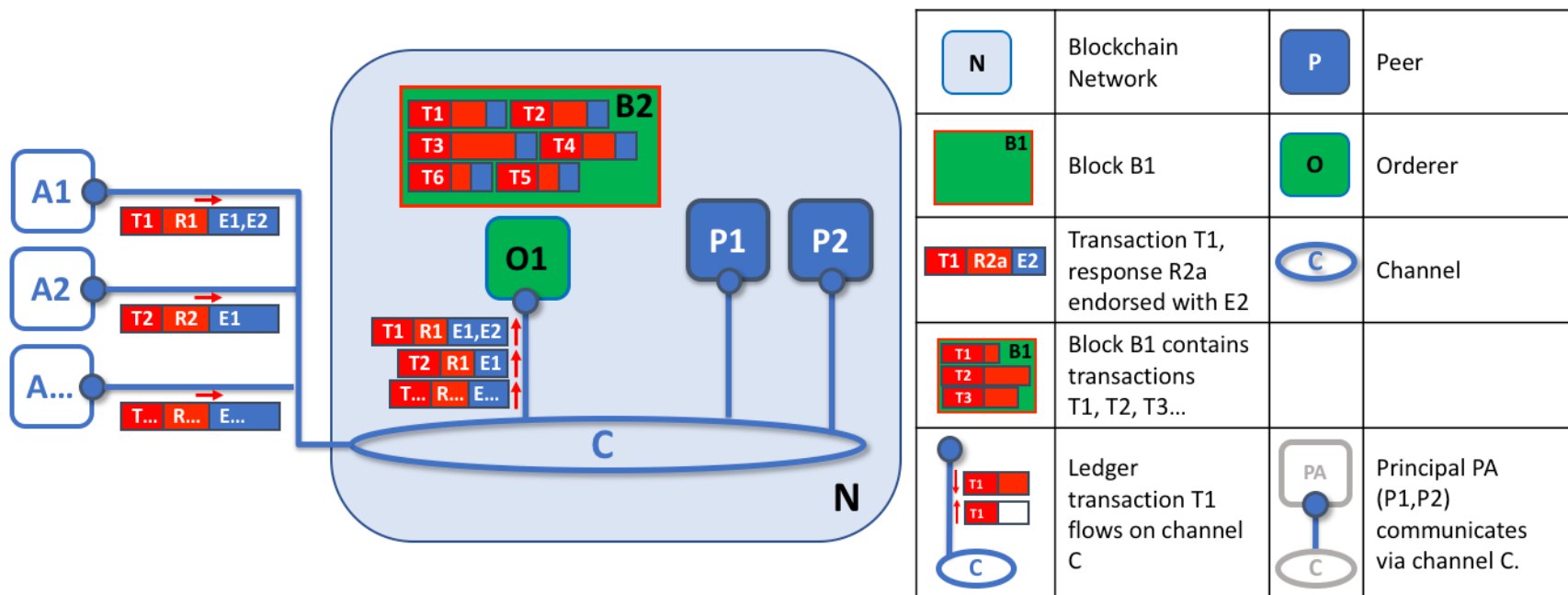


# 排序服务流程：1. 提案

- 客户端应用程序将交易提案发送给一组节点，这些节点将调用智能合约来生成一个账本更新提案，然后背书该结果。
- 背书节点此时不将提案中的更新应用于其账本副本。相反，背书节点将向客户端应用程序返回一个提案响应。
- 已背书的交易提案最终将在第二阶段经过排序生成区块，然后在第三阶段分发给所有节点进行最终验证和提交。



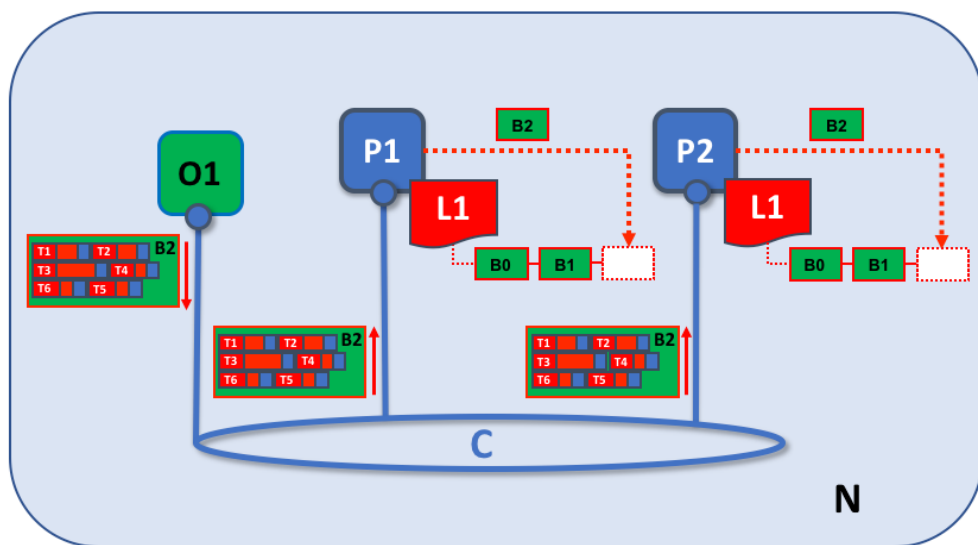
## 排序服务流程：2. 交易排序并打包成区块













注：虽然 Peer 节点执行智能合约并处理交易，而排序节点不会这样做。到达排序节点的每个授权交易都被机械地打包在一个区块中，排序节点不判断交易的内容。



# 排序服务流程： 3. 验证和提交



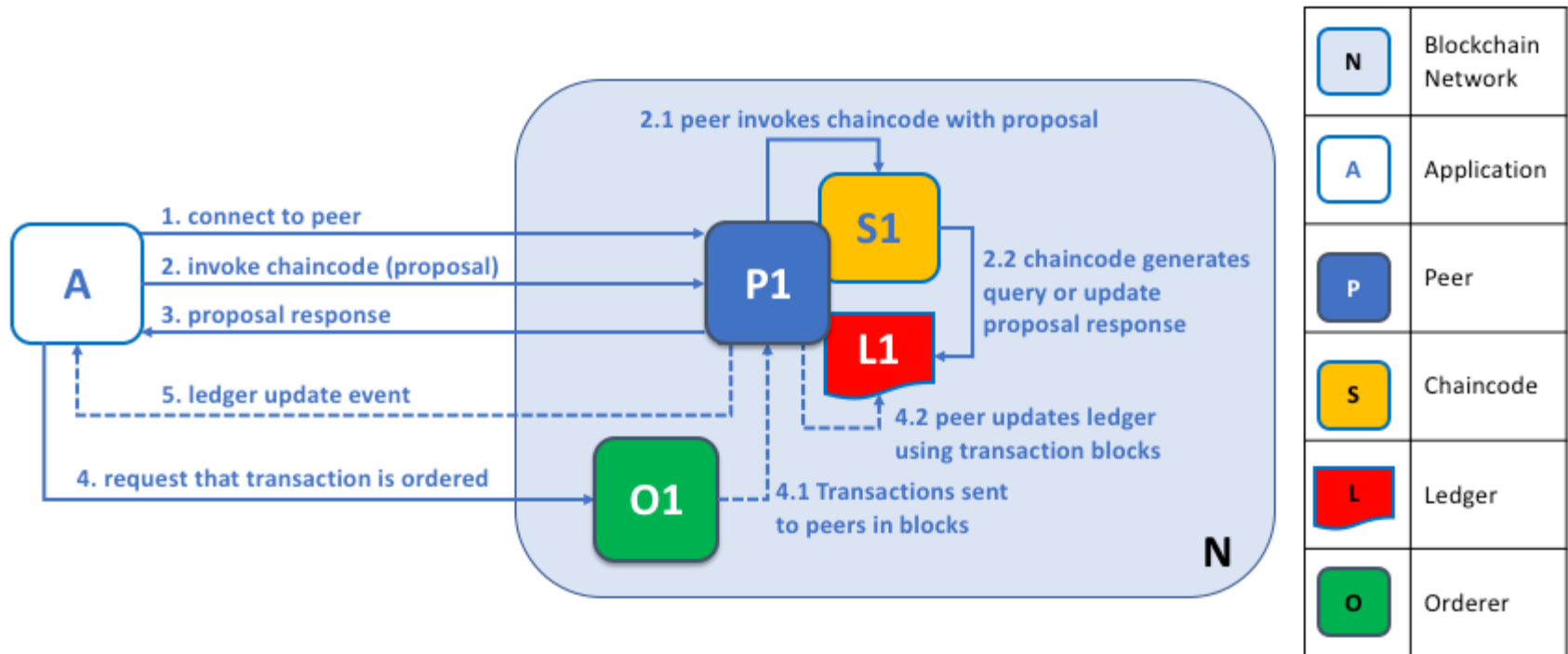
	Blockchain Network		Peer
	Channel		Orderer
	Ledger		Block B
	Ledger L1 has blockchain with blocks B0, B1		Block B1 contains transactions T1, T2, T3...
	Block B1 flows on channel C		Principal PA (P1, P2) communicates via channel C.

排序节点将区块分发给连接到它的所有 Peer 节点开始。并不是每个 Peer 节点都需要连接到一个排序节点，Peer 节点可以使用 gossip 协议 将区块关联到其他节点。

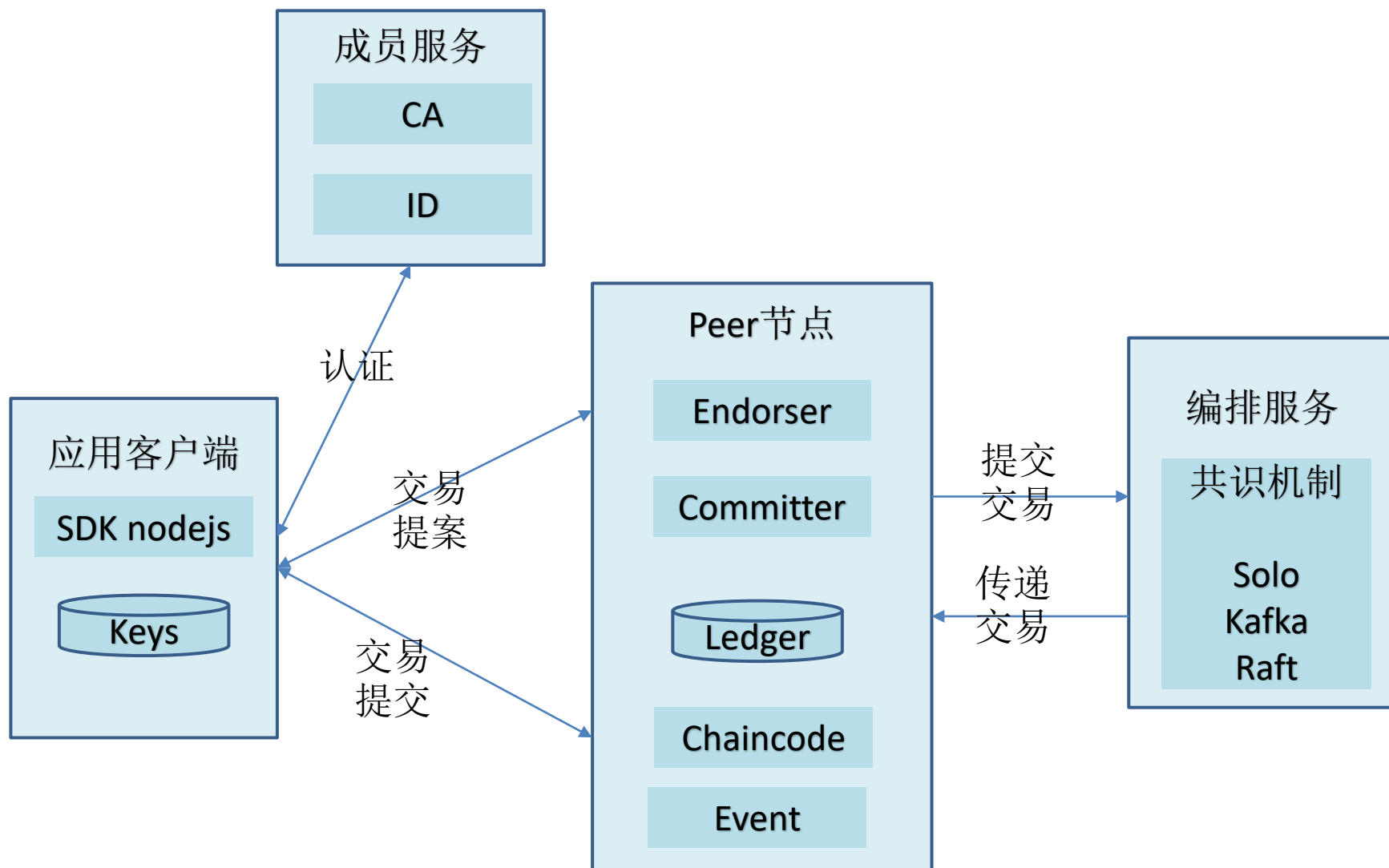




# Fabric 交易流程



# Fabric 运行时架构



# Fabric的排序服务

- Fabric将排序服务设计成了可插拔模块，可以根据不同的应用场景配置不同的共识选项。目前提供了3种模式实现：Solo、Kafka和Raft。
- Solo是一种部署在单个节点上的简单时序服务，它只支持单链和单通道。
- Kafka是一种支持多通道分区的集群共识服务，可以支持CFT（Crash Faults Tolerance）。它容忍部分节点宕机失效，但是不能容忍恶意节点。
- Raft是一个分布式崩溃故障容错共识算法，它可以保证在系统中部分节点出现非拜占庭故障的情况下，系统依然可以处理客户端的请求。在 $2n+1$ 个节点中，最多 $n$ 个节点发生崩溃故障。



# 容错共识对比

## 拜占庭容错（BFT）

- 出错节点
  - 崩溃节点
  - 恶意节点
- 节点总数 $3n+1$ 时，出错节点数不超过 $n$
- 算法：PBFT、Hotstuff

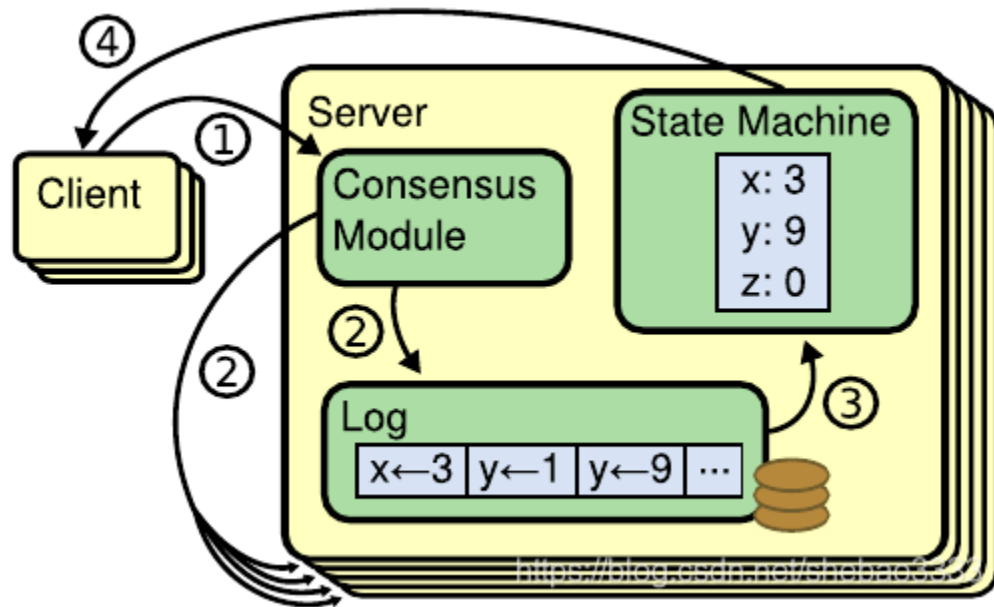
## 崩溃容错（CFT）

- 出错节点
  - 崩溃节点
- 节点总数 $2n+1$ 时，出错节点数不超过 $n$
- 实现：Raft、Kafka

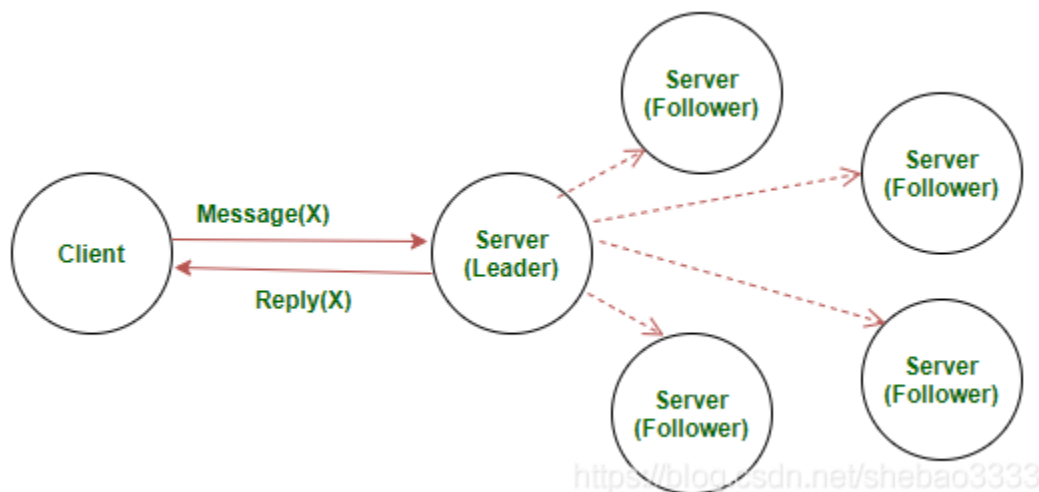


# Raft协议

Raft是一个管理复制日志（Replicated Log）的共识算法，复制日志是复制状态机（RSM：Replicated State Machine）的组成部分。



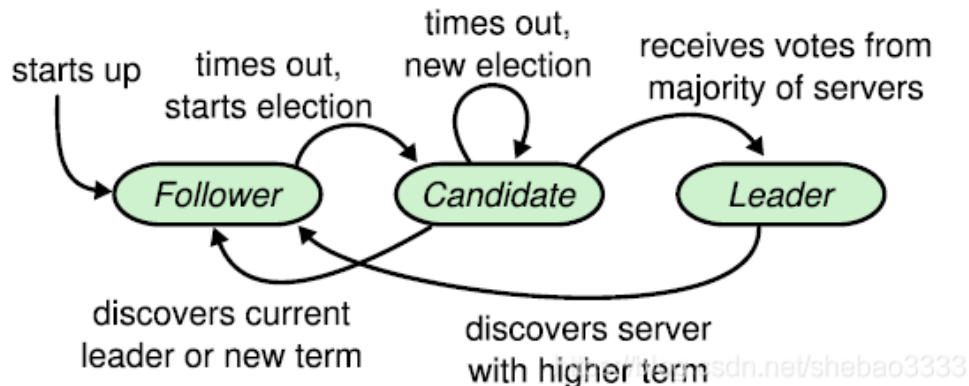
# 复制状态机工作流程



1. 客户端向主导节点（Leader Node）发送包含命令的请求。
2. 主导节点将收到的请求追加到其日志中，并将该请求发送给所有的 跟随节点（Follower Node）。跟随节点也会将该请求追加到自身的日志中并返回一个确认消息
3. 一旦主导节点收到大部分跟随节点的确认消息，就会将命令日志提交给其管理的状态机。一旦主导节点提交了日志，跟随节点也会将日志提交给自身管理的状态机
4. 主导节点向客户端返回响应结果



# 节点三个状态



- 跟随状态：初始情况下，所有的节点都处于跟随状态，也就是都是跟随节点。一旦某个跟随节点没有正常通信，它就转换为候选状态（Candidate），也就是成为一个候选节点。跟随节点的日志可以被主导节点重写。
- 候选状态：处于候选状态的节点会发起选举，如果它收到集群中大多数成员的投票认可，就转换为主导状态。
- 主导状态：处理客户端请求并确保所有的跟随节点具有相同的日志副本。主导节点不可以重写其自身的日志。
- 如果候选节点发现已经选出了主导节点，它就会退回到跟随状态。同样，如果主导节点发现另一个主导节点的任期（Term）值更高，它也会退回到跟随状态。
- 任期（Term）是一个单调递增的整数值，用来标识主导节点的管理周期。每个任期都从选举开始，直到下一个任期之前。



# Raft主导节点的选举

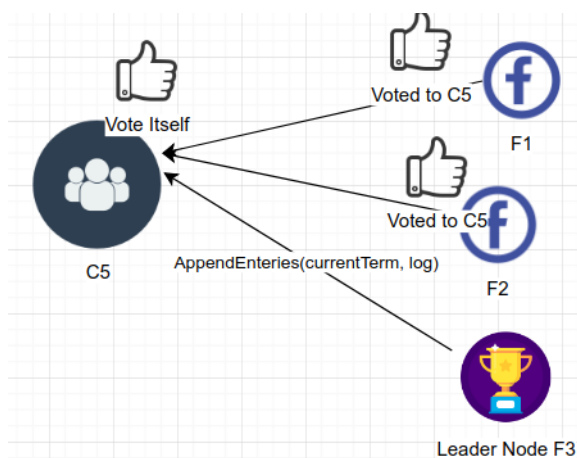
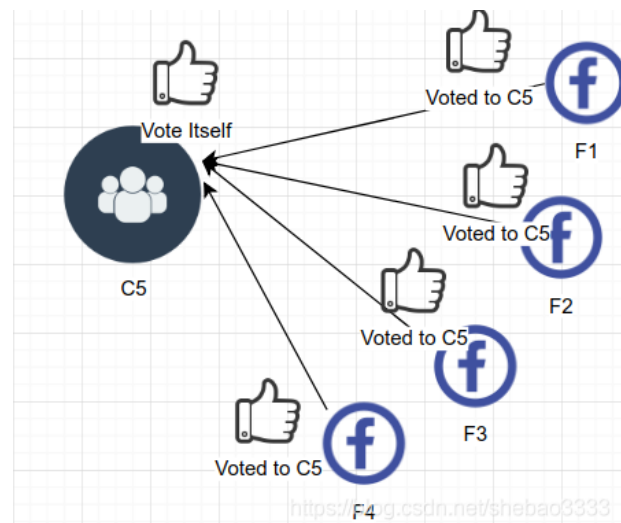
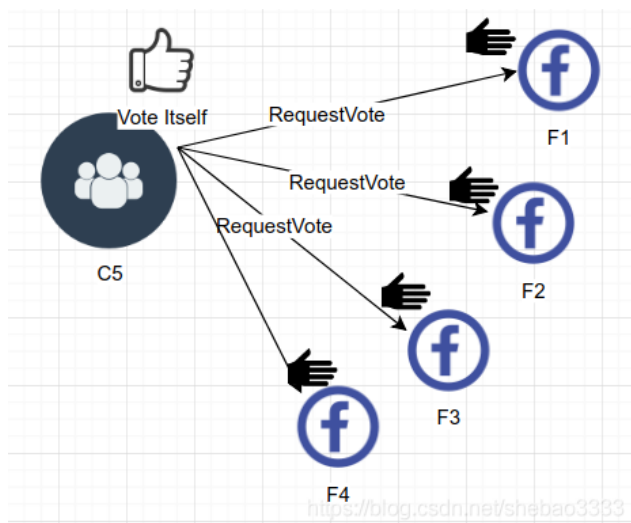


Raft使用心跳机制来出发主导节点的选举。当节点启动后进入跟随状态，只要它从主导节点或候选节点收到有效的RPC心跳消息，就会保持在跟随状态。主导节点会周期性发送心跳消息（没有日志项的AppendEntries RPC消息）给所有的跟随节点来维持其主导地位。如果某个跟随节点在一段时间内没有收到心跳消息，就发生选举超时事件，该节点就认为目前没有主导节点并发起选举来选出新的主导节点。





# Raft主导节点的选举



If current term of C5 > current Term of L3,  
C5 remain leader and L3 request will be discarded  
if not then C5 become follower and L3 will be leader.



# Fabric Raft排序节点工作流程

1. 交易自动路由到通道的当前主导节点
2. 主导节点验证交易后，将收到的交易传入区块切割模块，创建候选区块
3. 产生新的区块，主导排序节点将其应用于本地的Raft有限状态机（FSM）
4. 有限状态机将尝试复制到足够数量的排序节点，以便提交区块
5. 区块被写入接收节点的本地账本

每个通道都会运行Raft协议的单独实例。换句话说，有N个通道的网络，就有N个Raft集群，每个Raft集群都有自己的主导排序节点。



# 作业3

- 分析Fabric源代码, 说明Fabric中交易从产生到记入账本的每个环节及相关的数据结构。
- 提交格式: word文件
- 文件名: 姓名+学号+作业3.docx
- 提交邮箱: [sherlin@zju.edu.cn](mailto:sherlin@zju.edu.cn)
- 提交截止日期: 2021年1月11日12点



# 深入阅读

- HyperLedger 技术文档
- [https://hyperledger-fabric.readthedocs.io/zh\\_CN/release-2.2/whatis.html](https://hyperledger-fabric.readthedocs.io/zh_CN/release-2.2/whatis.html)



谢谢！

