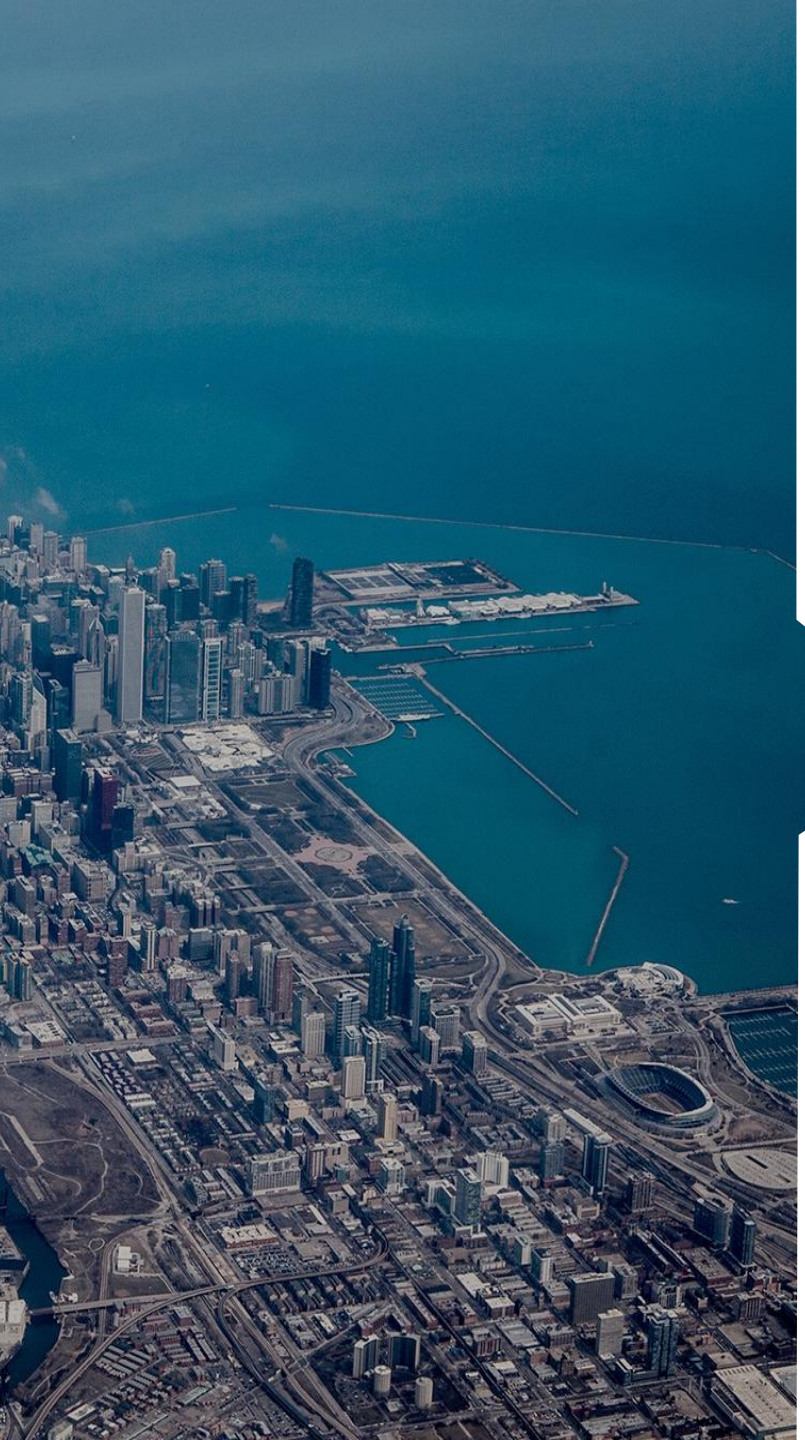
区块链技术与应用

华南理工大学　　许可　　kexu@scut.edu.cn

本课件主要内容来源于IBM



1 Hyperledger社区

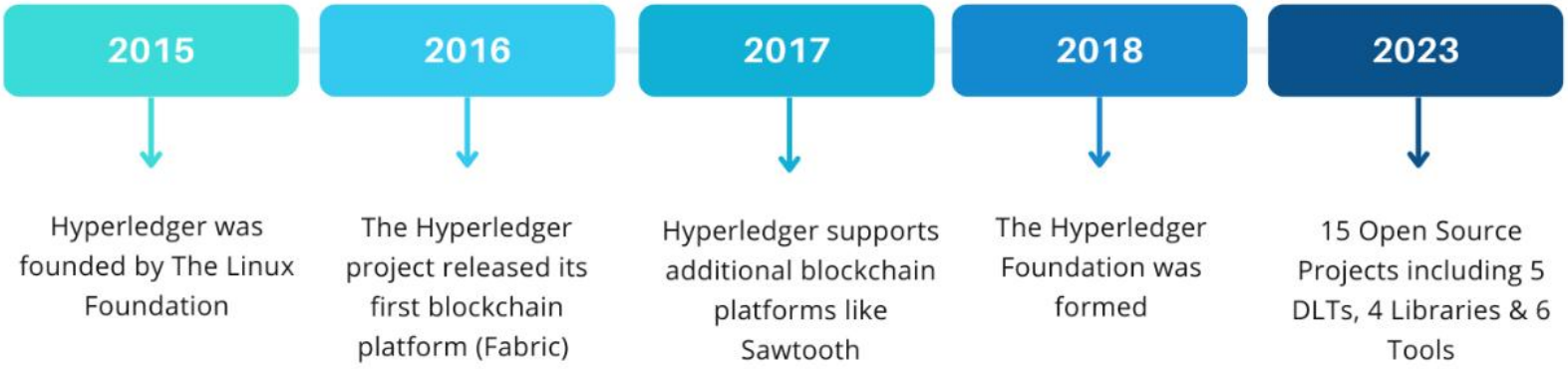
2 Hyperledger Fabric架构

Part ONE

Hyperledger 社区

Hyperledger社区

History of Hyperledger



Hyperledger Timeline

The Hyperledger project was founded in 2015 by the Linux Foundation. The project aimed to develop an open-source blockchain platform that could be used by businesses and enterprises to build decentralized applications. The project was named Hyperledger, and it quickly gained momentum within the blockchain community.

In 2016, the Hyperledger project released its first blockchain platform, called Fabric. Fabric, a framework that IBM first proposed, was designed to be a modular and scalable blockchain platform that could be used to build a wide range of decentralized applications. Fabric was built using the consensus mechanism, Byzantine Fault Tolerance, which allowed it to be highly secure and reliable.

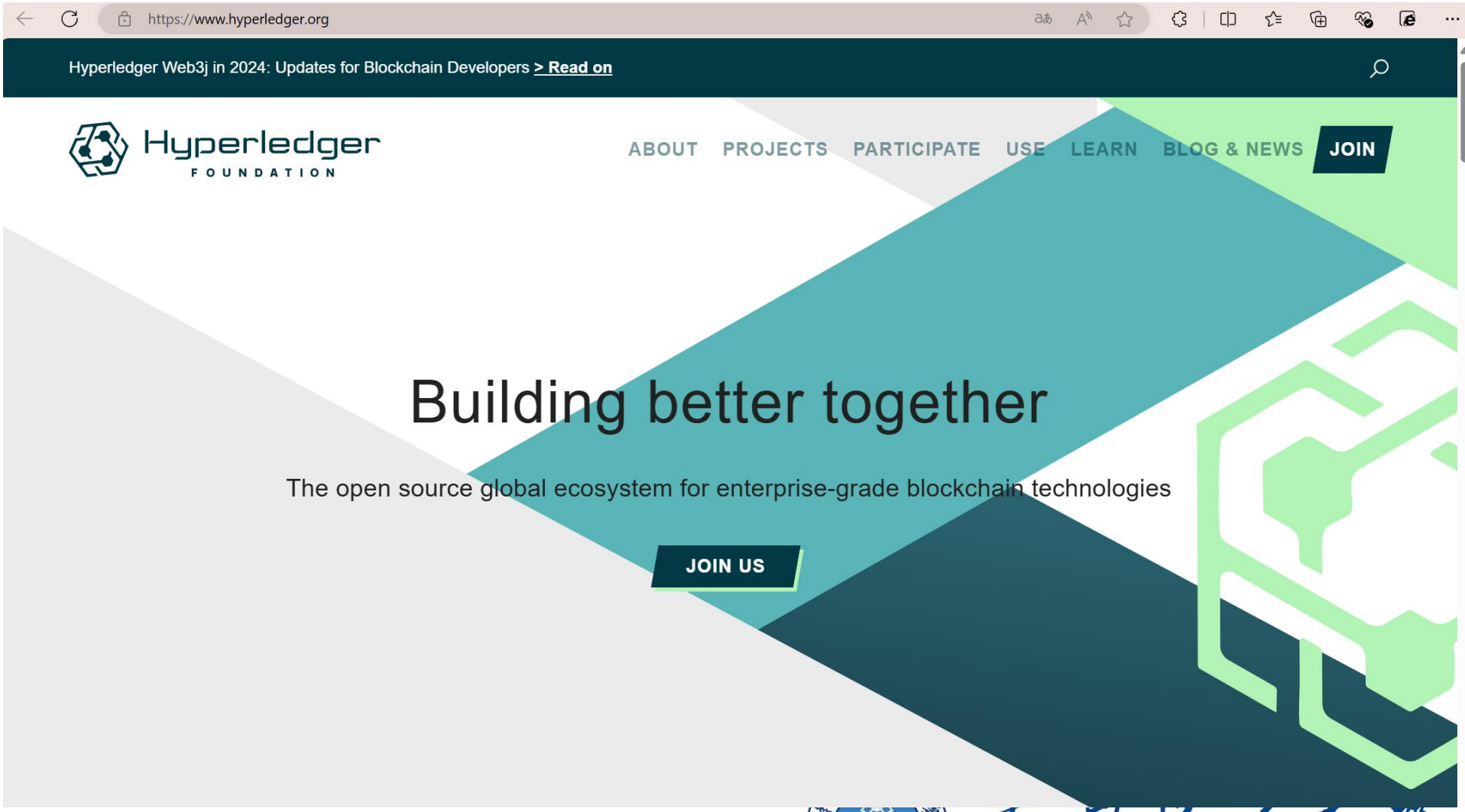
Hyperledger社区

- 2015年12月，由Linux基金会牵头，20余家初始企业成员（包括IBM、Intel等）共同宣布Hyperledger（超级账本）联合项目成立。

- 超级账本是开放开源的社区，其定位是开源的分布式账本项目（DLT），着力为部署企业级区块链而开发一系列稳定的框架，工具和程序库。

- Hyperledger项目的官方网站：hyperledger.org，Hyperledger项目的代码是托管在Gerrit上，并通过GitHub提供镜像。

Hyperledger社区



South China University of Technology

Hyperledger社区

Hyperledger
FOUNDATION

+Open source

+Open development

+Open governance

= the future of decentralized tech

South China University of Technology

Hyperledger社区

Hyperledger Foundation Goals

Create enterprise grade, open source, distributed ledger frameworks and code bases to support business transactions

Provide neutral, open, and community-driven infrastructure supported by technical and business governance

Build technical communities to develop blockchain and shared ledger POCs, use cases, field trails and deployments

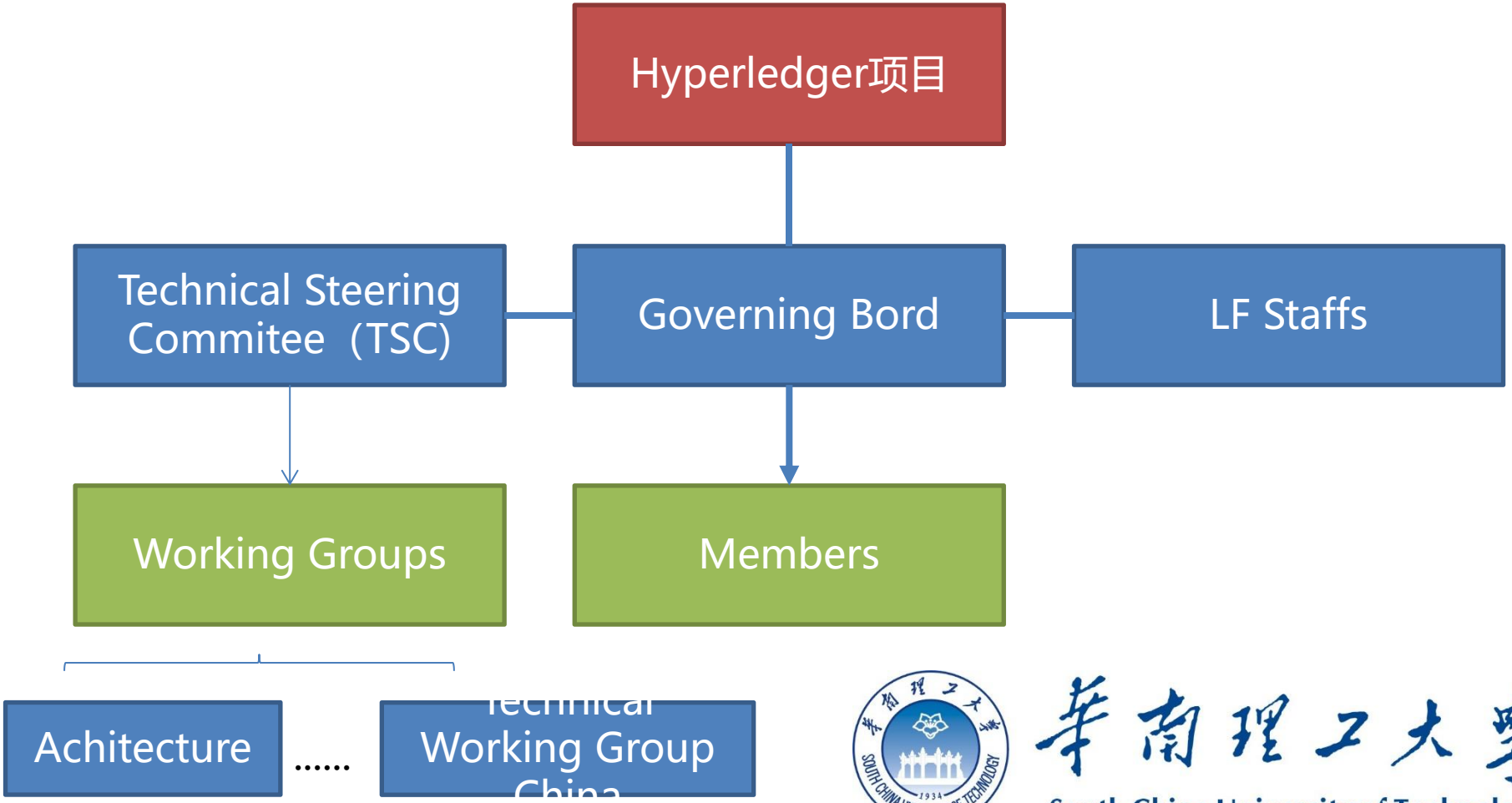
Educate the public about the market opportunity for blockchain technology

Promote our community of communities taking a toolkit approach with many platforms and frameworks

South China University of Technology

Hyperledger社区

- Hyperledger社区的组织架构



Hyperledger社区

- Hyperledger的愿景是：为透明、公开、去中心化的企业级分布式账本技术提供开源参考实现，并推动区块链和分布式账本相关协议、规范和标准的发展。

- Hyperledger是发展最快的Linux基金会项目之一，拥有超过280个成员。

- Hyperledger项目由面向不同目的和场景的子项目构成。它们共同遵循以下原则：（1）重视模块化设计；（2）重视代码可读性；（3）可持续的演化路线。

Hyperledger社区

Hyperledger社区

The Hyperledger Greenhouse（2023超级账本温室）

Hyperledger社区

Hyperledger FOUNDATION

ABOUT PROJECTS PARTICIPATE USE LEARN BLOG & NEWS JOIN

All Projects

The community is working on other types of projects that include distributed ledgers, tools, libraries and domain-specific projects. There are also a number of experimental development efforts being worked on in >> Hyperledger Labs, a place where community members can easily collaborate with others on new ideas.

Hyperledger SOLANG

PROJECTS | INCUBATING | SOLIDITY COMPILER

Solang is a Solidity compiler written in rust which uses llvm as the compiler backend. Solang can compile Solidity for Solana and Substrate.

READ MORE

Hyperledger IROHA

PROJECTS | GRADUATED | GENERAL-PURPOSE DLT

A developer friendly blockchain platform written in C++, aimed at helping businesses and financial institutions manage digital assets.

READ MORE

Hyperledger INDY

PROJECTS | GRADUATED | DECENTRALIZED IDENTITY | VERIFIABLE DATA REGISTRY DLT

A distributed ledger that provides tools, libraries, and reusable components for creating and using independent digital identities.

READ MORE

Hyperledger FIREFLY

PROJECTS | GRADUATED | CONNECTIVITY/INTEGRATION GATEWAY

Hyperledger FireFly is the first open source Supernode: a complete stack for enterprises to build and scale secure Web3 applications. Learn more!

READ MORE

Hyperledger社区

Hyperledger FABRIC

PROJECTS | GRADUATED | GENERAL-PURPOSE DLT

FEATURED HYPERLEDGER PROJECTS

Hyperledger Fabric - A blockchain framework implementation intended as a foundation for developing applications or solutions with a modular architecture.

READ MORE

Hyperledger CELLO

PROJECTS | INCUBATING | DEPLOYMENT AUTOMATION

A blockchain module toolkit designed to bring the on-demand "as-a-service" deployment model to the blockchain ecosystem.

READ MORE

Hyperledger CALIPER

PROJECTS | INCUBATING | PERFORMANCE BENCHMARKING

A blockchain benchmark tool designed to allow users to measure the performance of a specific blockchain implementation.

READ MORE

Hyperledger CACTI

PROJECTS | GRADUATED | CROSS-CHAIN INTEROPERABILITY

FEATURED HYPERLEDGER PROJECTS

Hyperledger Cacti is a blockchain integration tool designed to allow users to securely integrate different blockchains.

READ MORE

Hyperledger社区

Hyperledger BEVEL

PROJECTS | INCUBATING | DEPLOYMENT AUTOMATION

A blockchain framework implementation intended as a foundation for developing applications or solutions with a modular architecture.

READ MORE

Hyperledger BESU

PROJECTS | GRADUATED | GENERAL-PURPOSE DLT

FEATURED HYPERLEDGER PROJECTS

Hyperledger Besu is an open source Ethereum client developed under the Apache 2.0 license and written in Java.

READ MORE

Hyperledger ARIES

PROJECTS | GRADUATED | DECENTRALIZED IDENTITY

VERIFIABLE CREDENTIALS | FEATURE

FEATURED HYPERLEDGER PROJECTS

Hyperledger Aries is a shared, reusable, interoperable tool kit for developing critical digital credentials solutions.

READ MORE

Hyperledger ANONCREDS

PROJECTS | INCUBATING | DECENTRALIZED IDENTITY

VERIFIABLE CREDENTIALS

Hyperledger AnonCreds - short for “Anonymous Credentials”- is the most commonly used Verifiable Credential (VC) format in the world.

READ MORE

Hyperledger社区

PROJECTS | INCUBATING | ETHEREUM INTEGRATION LIBRARY

Hyperledger Web3j is a highly modular, reactive, type safe Java and Android library for working with Smart Contracts and integrating with clients (nodes) on the Ethereum network.

READ MORE

PROJECTS | DECENTRALIZED IDENTITY | VERIFIABLE CREDENTIALS

FEATURED HYPERLEDGER PROJECTS

Hyperledger Identus provides components to develop decentralized identity solutions that adhere to widely recognized self-sovereign identity (SSI) standards.

READ MORE

Hyperledger社区

Hyperledger框架类子项目

Hyperledger Foundation Projects

HYPERLEDGER FABRIC

Hyperledger FABRIC

PROJECTS | GRADUATED | GENERAL-PURPOSE DLT | FEATURED HYPERLEDGER PROJECTS

Type: Distributed ledger software

Hyperledger Fabric serves as the foundation for developing applications or solutions with a modular architecture. It allows interchangeable components, including consensus and membership services, enabling a plug-and-play environment.

Hyperledger Fabric is designed to meet diverse industry needs. Additionally, it offers a unique approach to consensus that facilitates scalable performance while maintaining privacy.

Hyperledger社区

Hyperledger框架类子项目

- Fabric最早加入Hyperledger项目，由IBM、DAH等企业于2015年底提交。项目的Github地址： https://github.com/hyperledger/fabric
- Fabric项目的定位是面向企业的分布式账本平台，创新性地引入了权限管理支持，设计上支持可插拔，可扩展，是首个面向联盟链场景的开源项目
- Fabric基于Golang语言实现
- Fabric包括Fabric CA、Fabric SDK（包括Node.Js、Python和Java等语言）和fabric-api等子项目

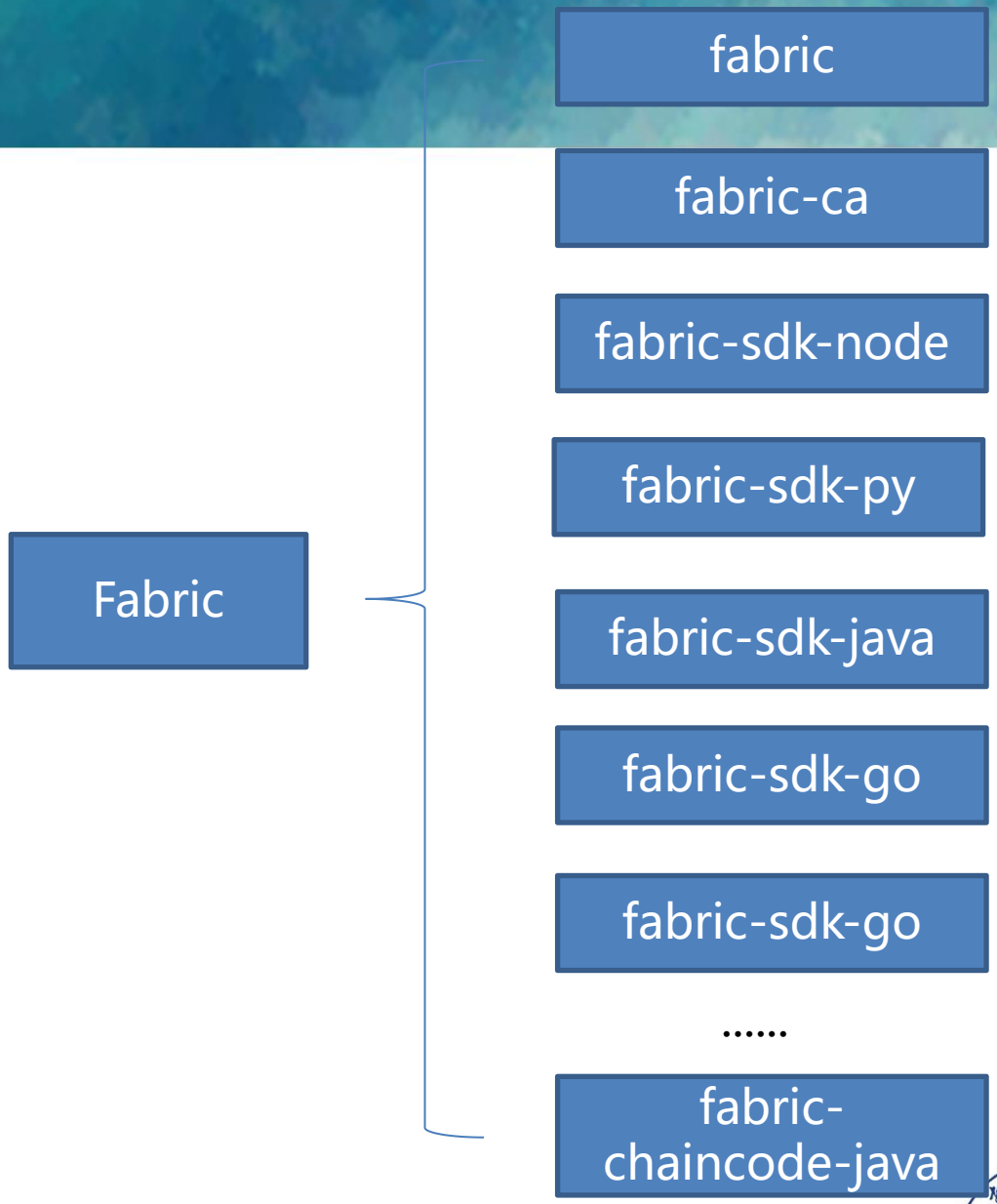
Hyperledger Fabric特点

Hyperledger ≠ Fabric

Fabric ∈ Hyperledger

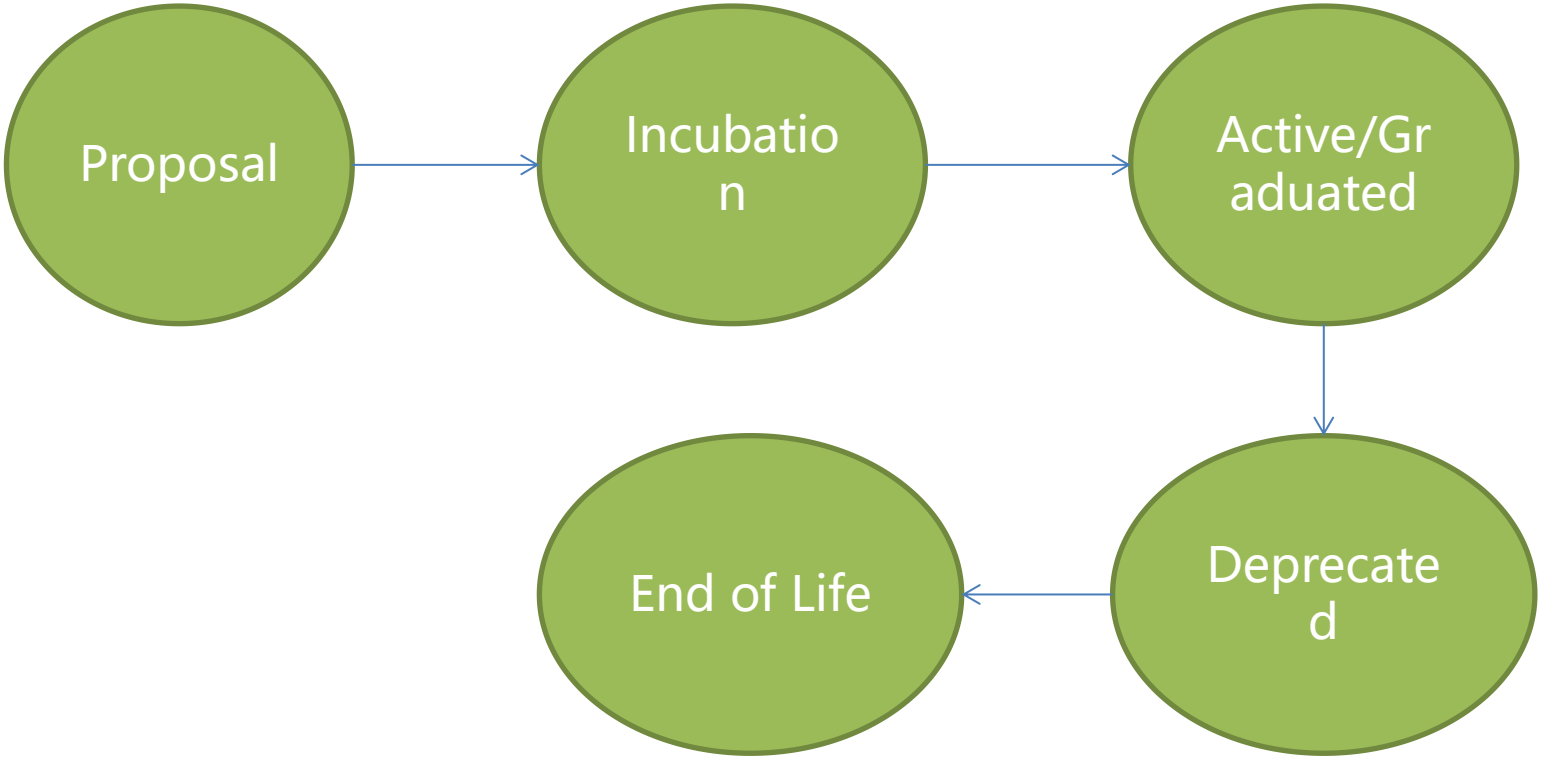
Hyperledger Fabric 是 Hyperledger 的核心项目。

- 本质 → 分布式共享账本
- 目标 → 成为开发应用和解决方案的基础
- 设计 → 模块化架构（组件可根据需要灵活配置，插入即用）


Fabric
fabric
fabric-ca
fabric-sdk-node
fabric-sdk-py
fabric-sdk-java
fabric-sdk-go
fabric-sdk-go
.....
fabric-chaincode-java

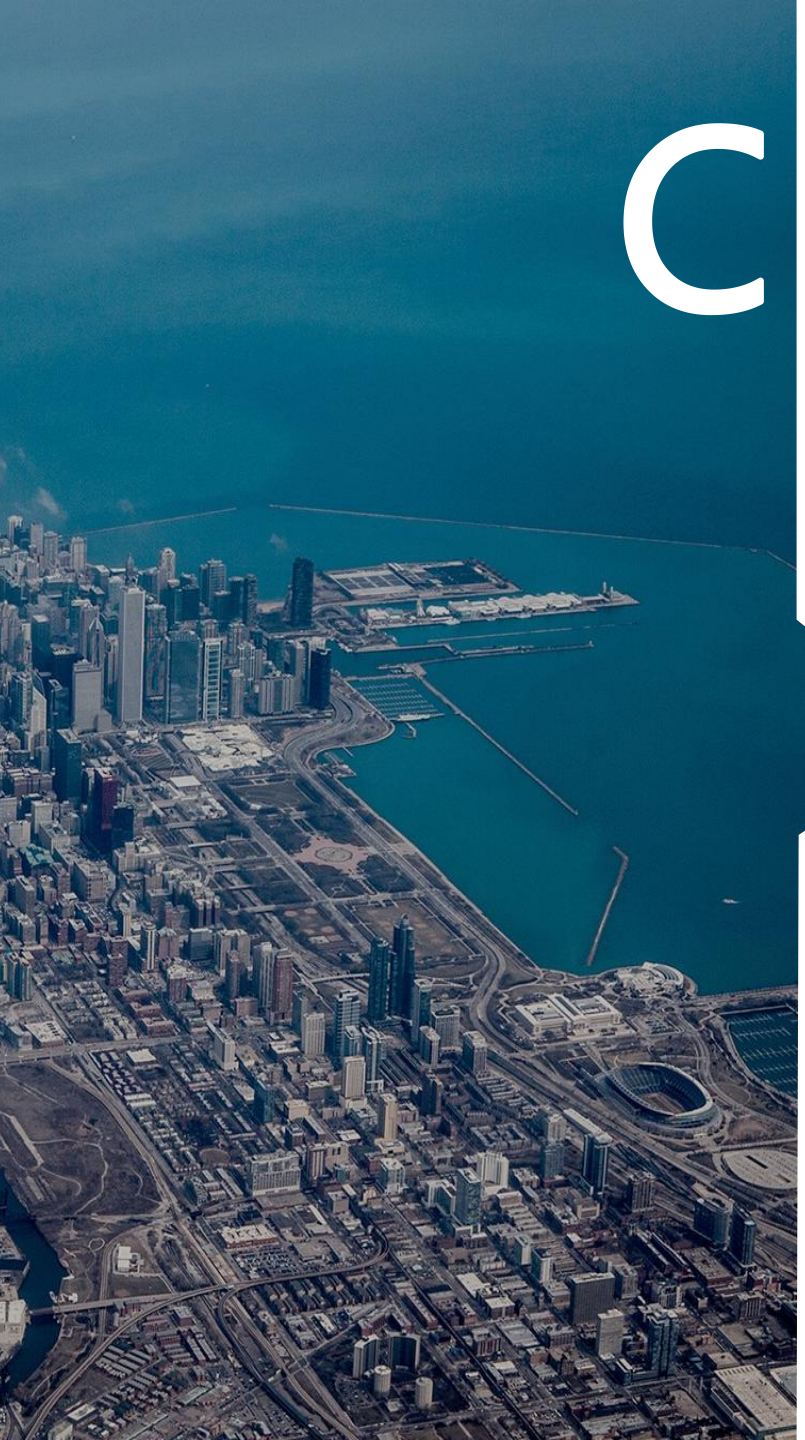
Hyperledger社区

- Hyperledger项目的管理： （生命周期管理模式）



Part TWO

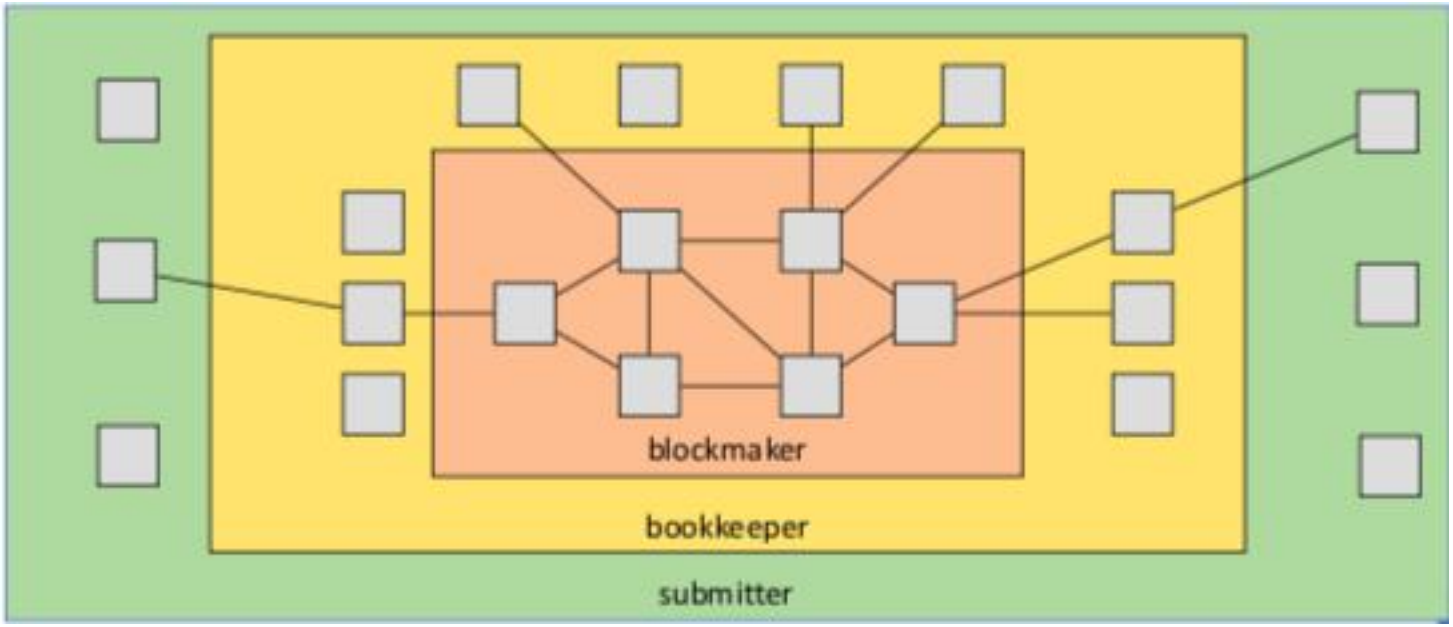
Hyperledger Fabric 架构



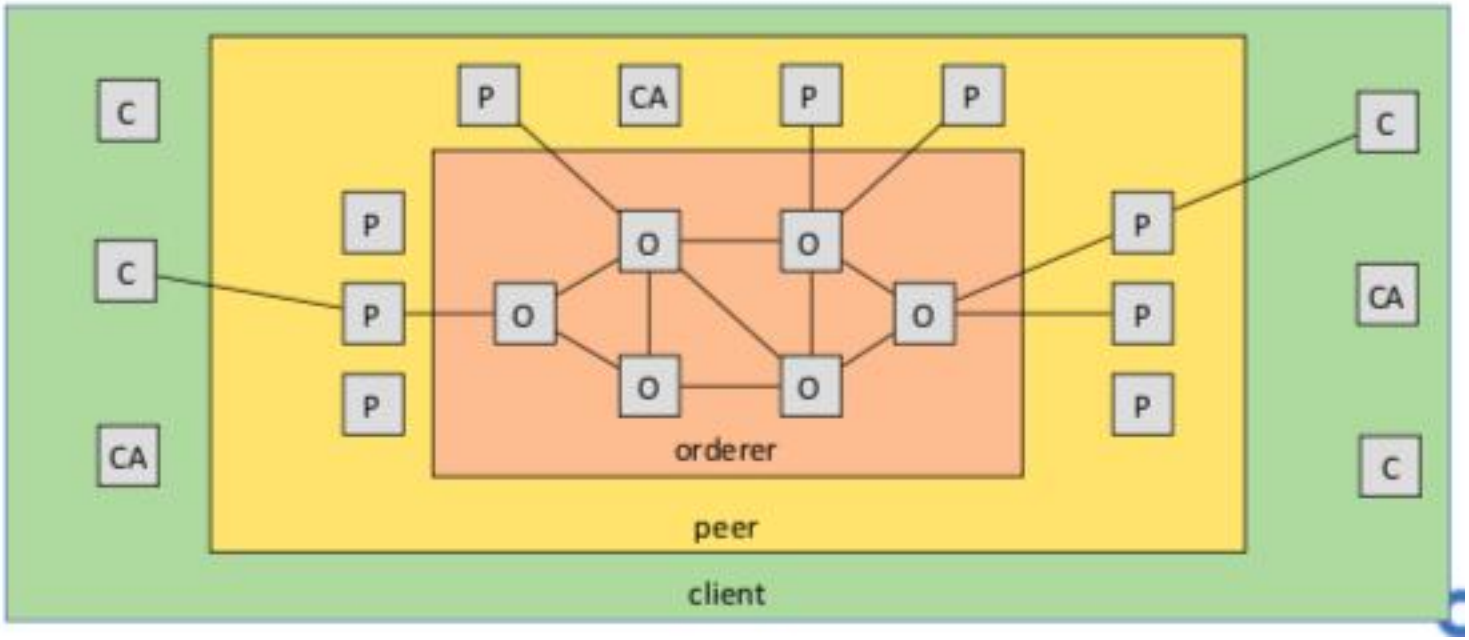
Contents

Part 2.1

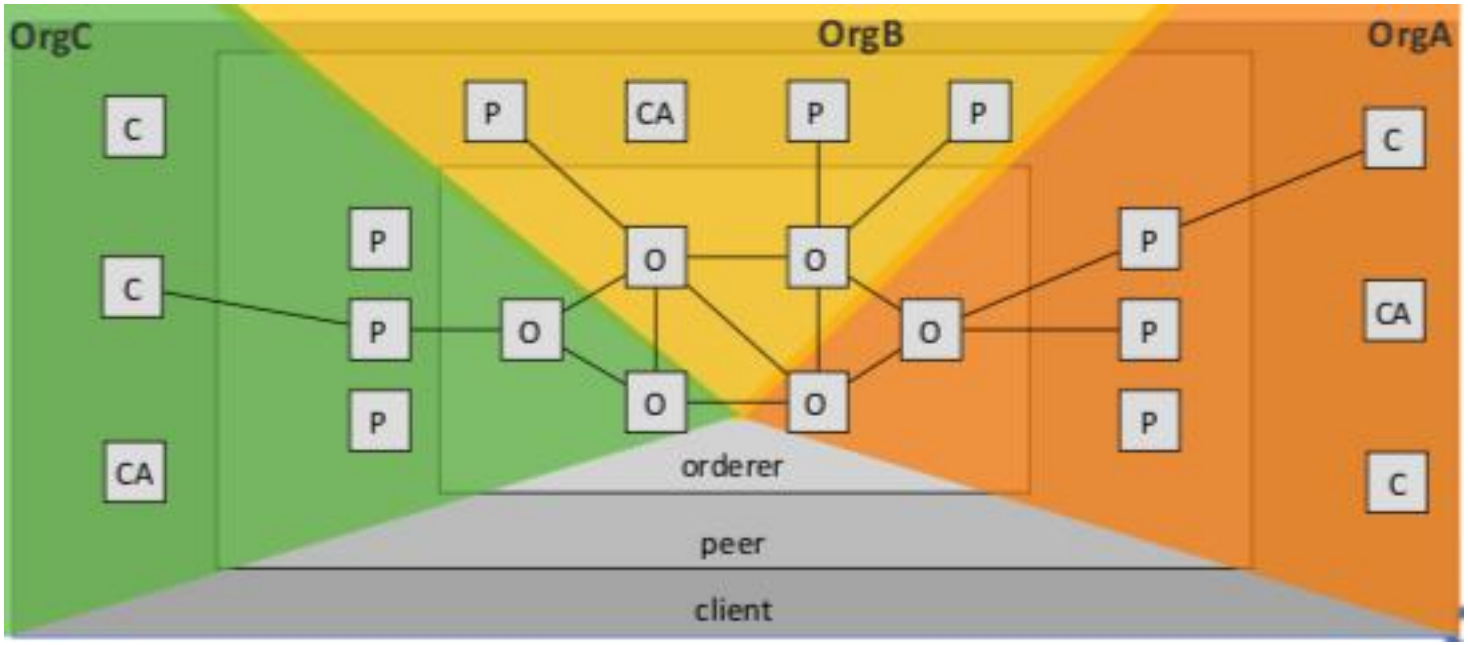
公有链和联盟链结构中的三类不同的角色



映射到Fabric



映射到Fabric（Organization）



Fabric Characteristics

- Permissioned
- Highly modular
- Smart contract in general purpose language
- Pluggable consensus
- Privacy
- No “mining” or native crypto-currency required for consensus
- Execute-order-validate vs order-execute

Fabric Transaction Flow

Everything behind

```
sender_balance-=10; receiver_balance+=10
```

Fabric Basic Three Roles

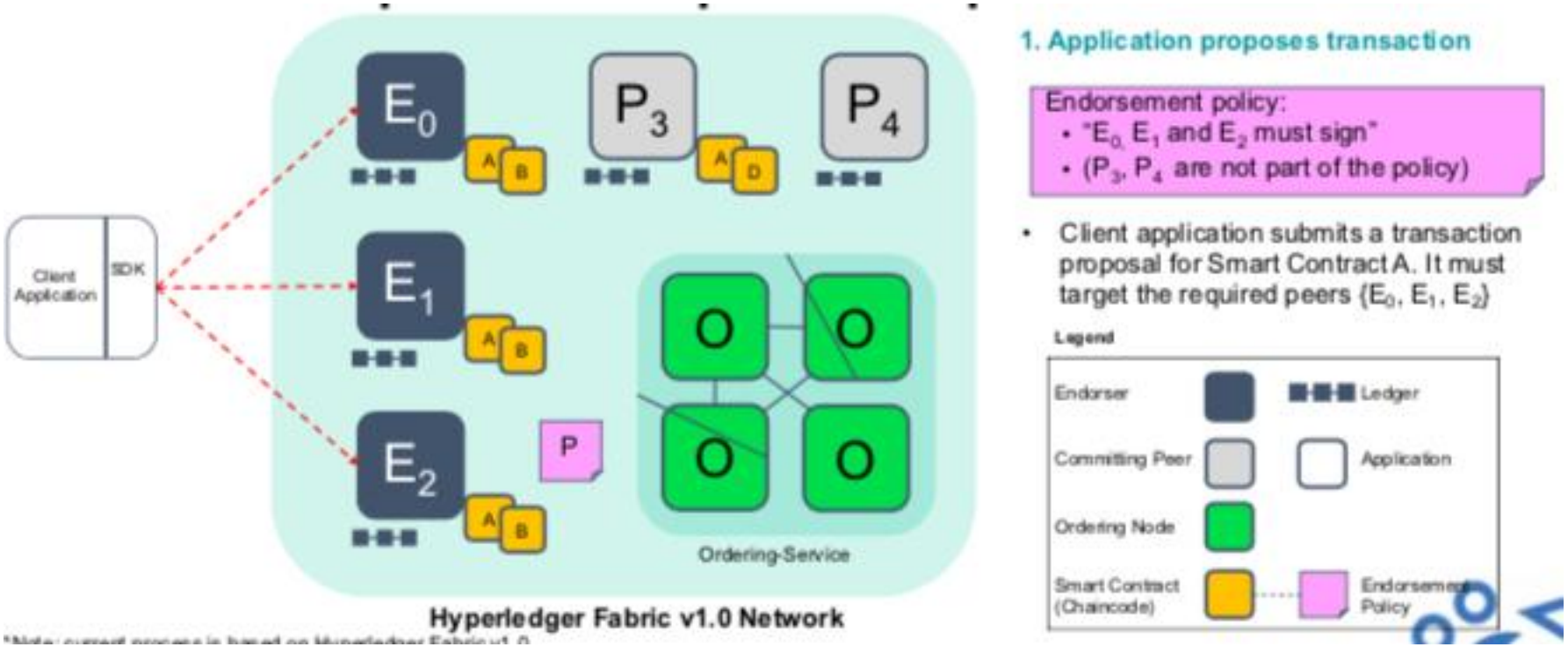
- **Committing Peer （提交节点）**
- ✓ Maintains ledger and state
- ✓ Commits transactions
- ✓ May hold smart contract (chaincode)

- **Endorsing peer （背书节点）**
- ✓ Receives a transaction proposal for endorsement, responds granting or denying endorsement
- ✓ Must hold smart contract
- ✓ Verifies that its content obeys a given smart contract
- ✓ Endorser “signs” the contract

Fabric Basic Three Roles

- **Ordering Node（排序节点）**
- ✓ Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes
- ✓ Controls what goes in the ledger making sure that the ledger is consistent
- ✓ Dose not hold smart contract
- ✓ Dose not hold ledger

Transaction process: Step 1/7- Propose

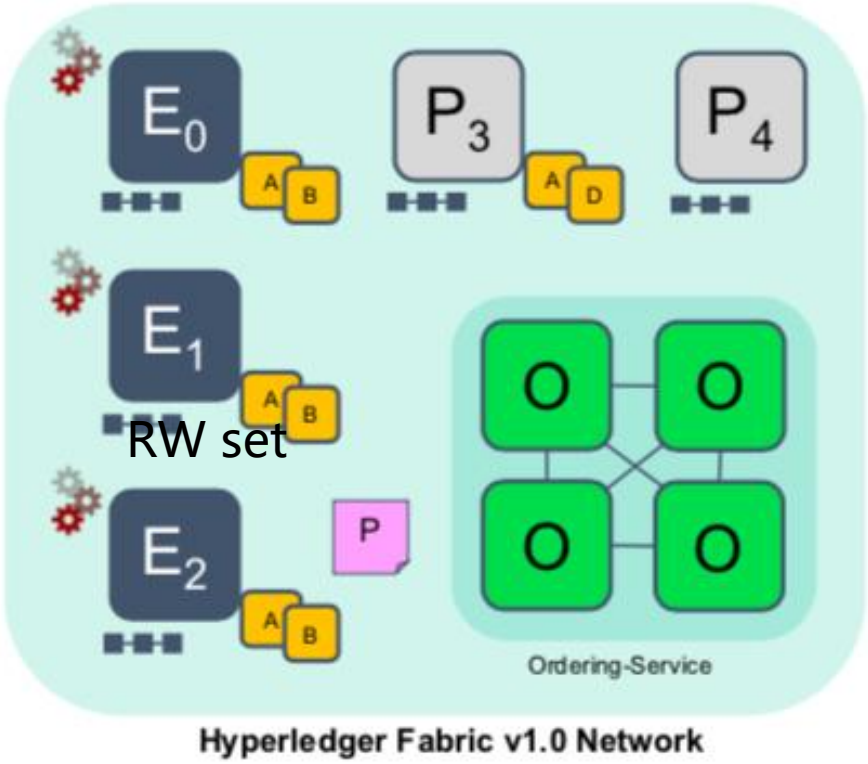


交易提案（Proposal）包含如下5个要素：

（1）channelID，通道信息；（2）chaincodeID：要执行的链码信息；

（3）timestamp：时间戳；（4）sign：客户端的签名；（5）txPayload：提交的事务本身所包含的内容（要调用的链码的函数及相应参数、调用的相关属性）

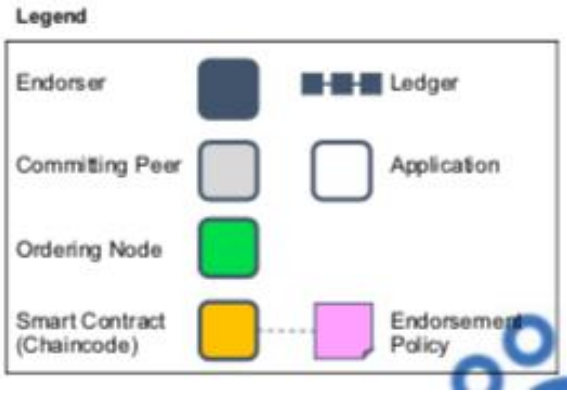
Transaction process: Step 2/7- Execute



Hyperledger Fabric v1.0 Network

2. Endorsers Execute Proposals

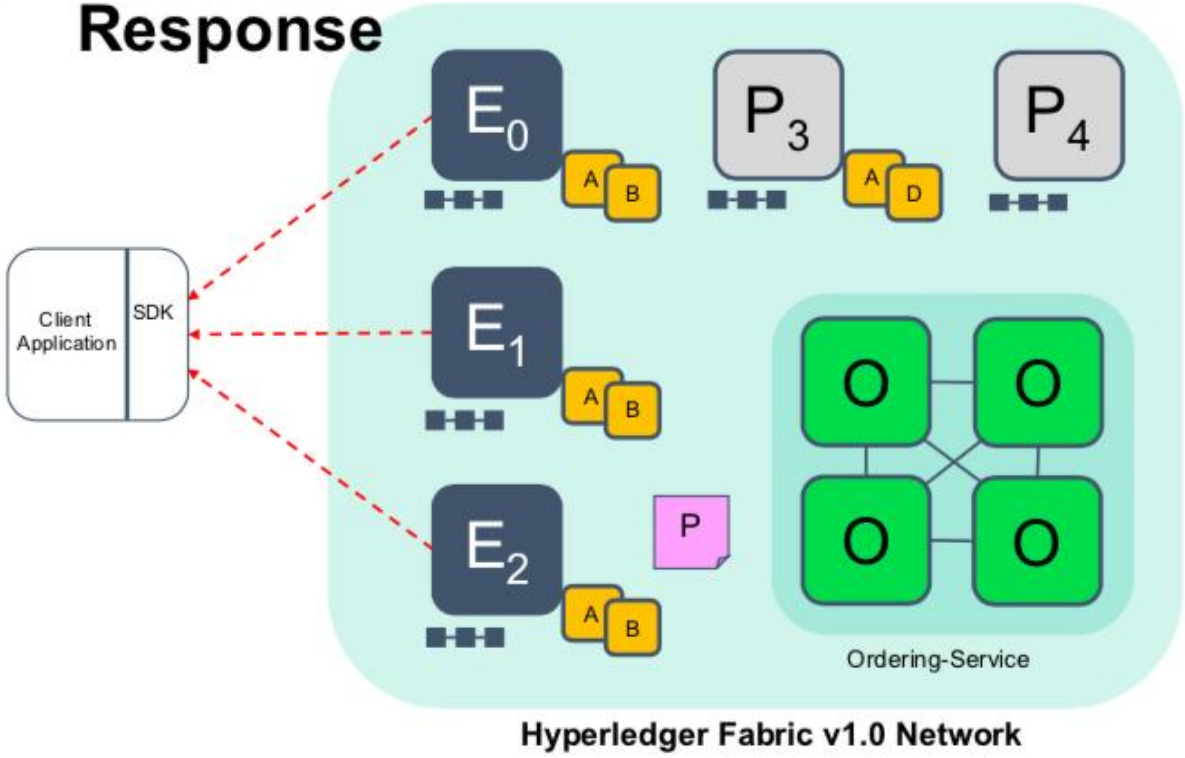
- E_0, E_1 & E_2 will each execute the proposed transaction. None of these executions will update the ledger
- Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric
- Transactions can be signed & encrypted



RW set:

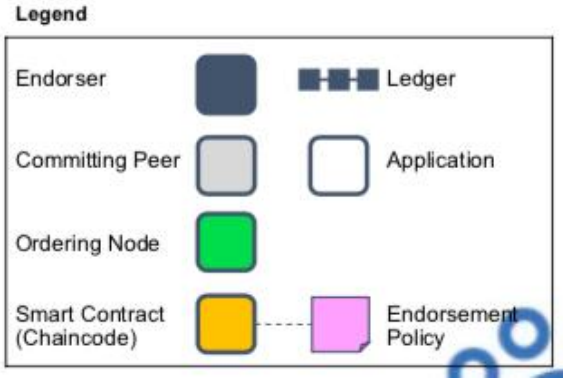
- (Read Set) 读集：Key=balance，value=100，version=1
- (Write Set) 写集：Key=balance，value=90，version=2

Transaction process: Step 3/7- Proposal response

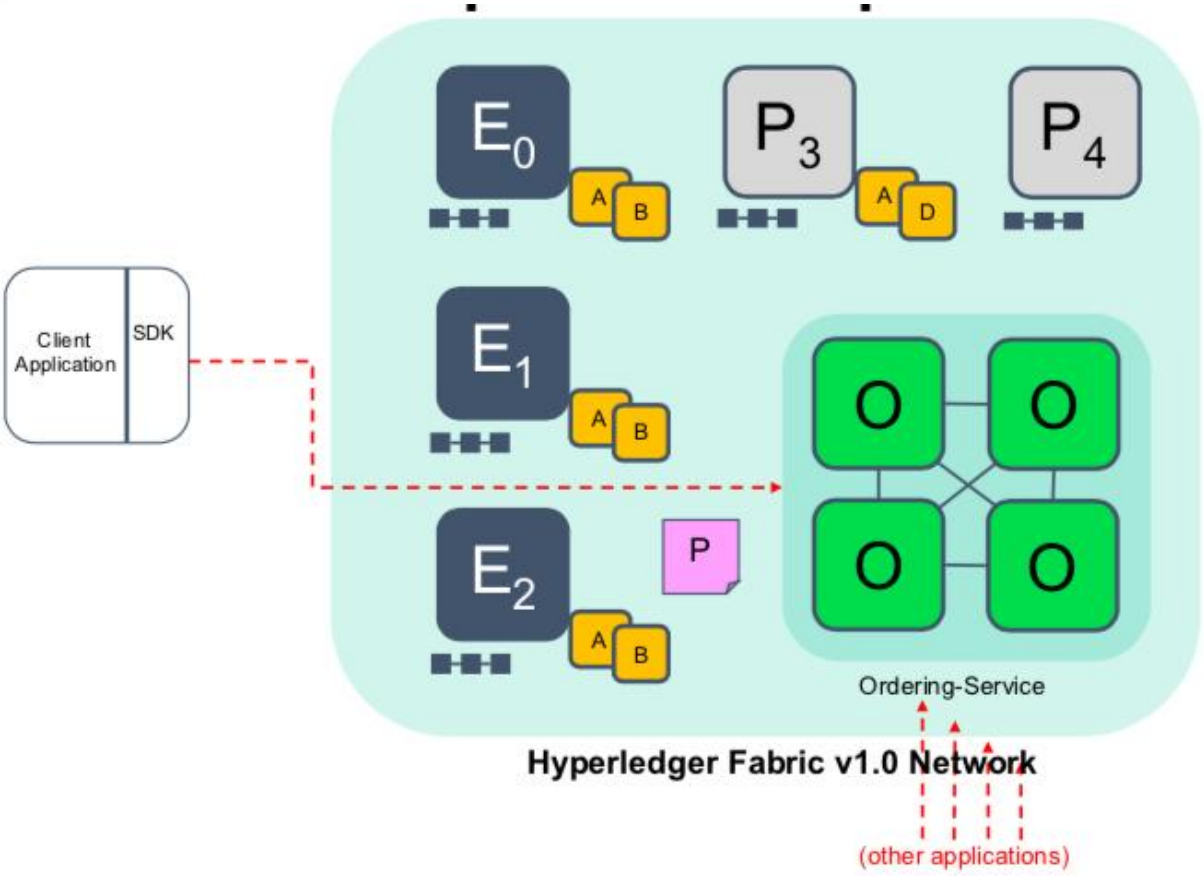


3. Application receives responses

- RW sets are asynchronously returned to application
- The RW sets are signed by each endorser, and also includes each record version number
- (This information will be checked later in the consensus process)

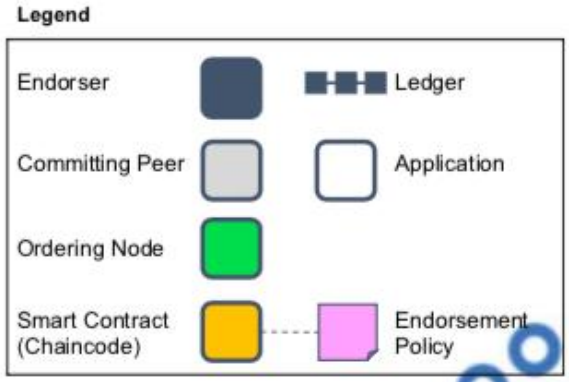


Transaction process: Step 4/7- Order Transaction

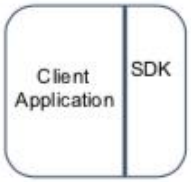


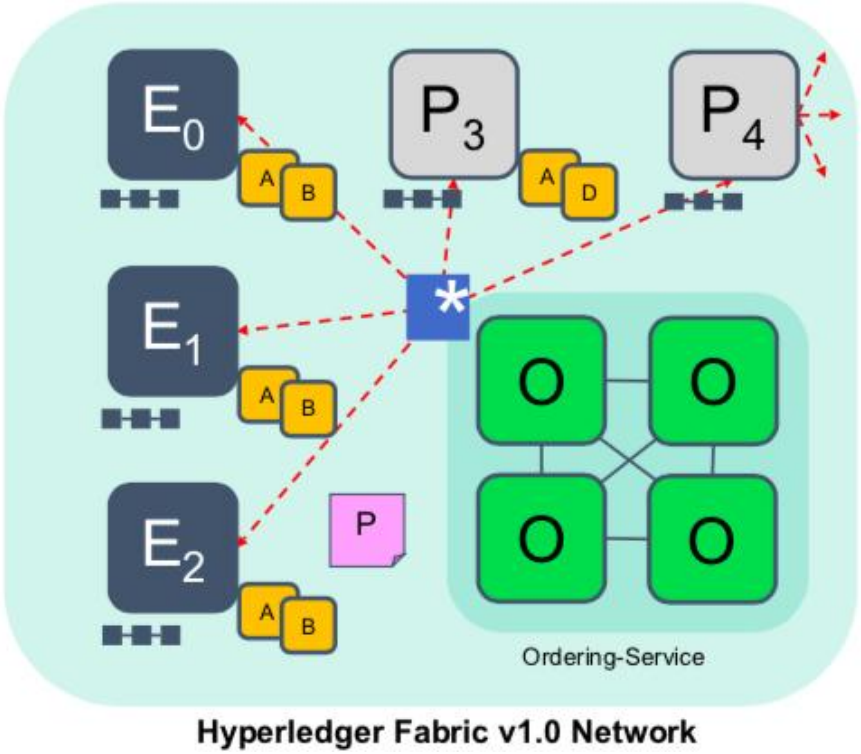
4. Responses submitted for ordering

- Application submits responses as a transaction to be ordered
- Ordering happens across the fabric in parallel with transactions submitted by other applications



Transaction process: Step 5/7- Deliver

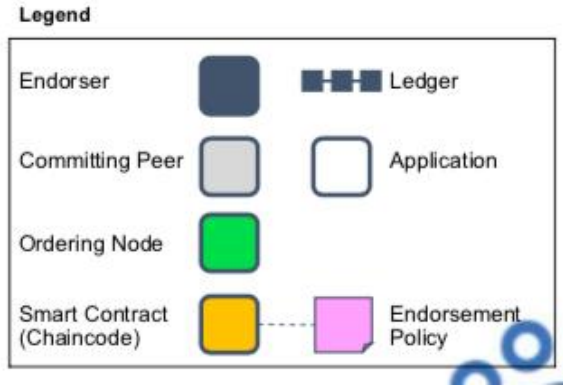




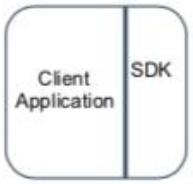
Hyperledger Fabric v1.0 Network

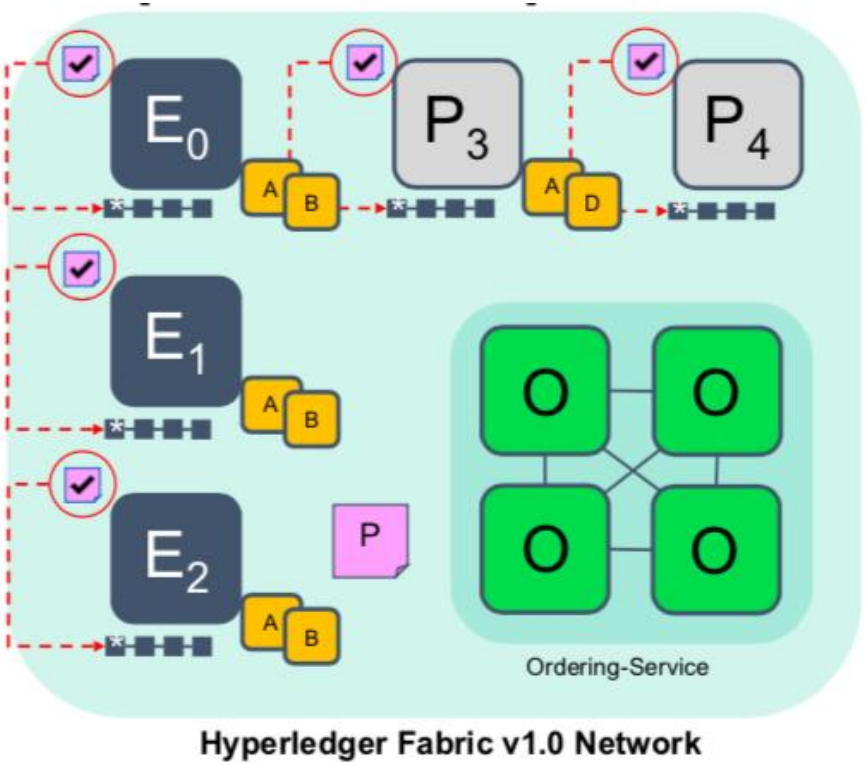
5. Orderer delivers to committing peers

- Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown)
- Different ordering algorithms available:
  - Solo (Single node, development)
  - Kafka (Crash fault tolerance)



Transaction process: Step 6/7- Validate

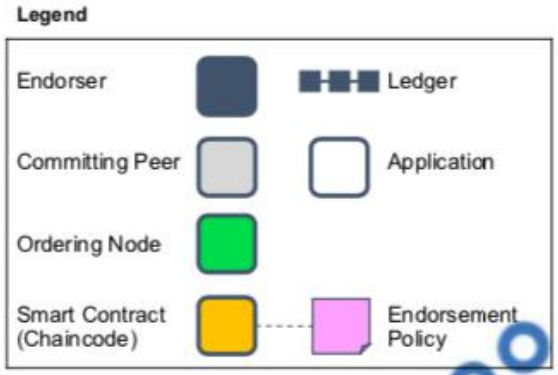




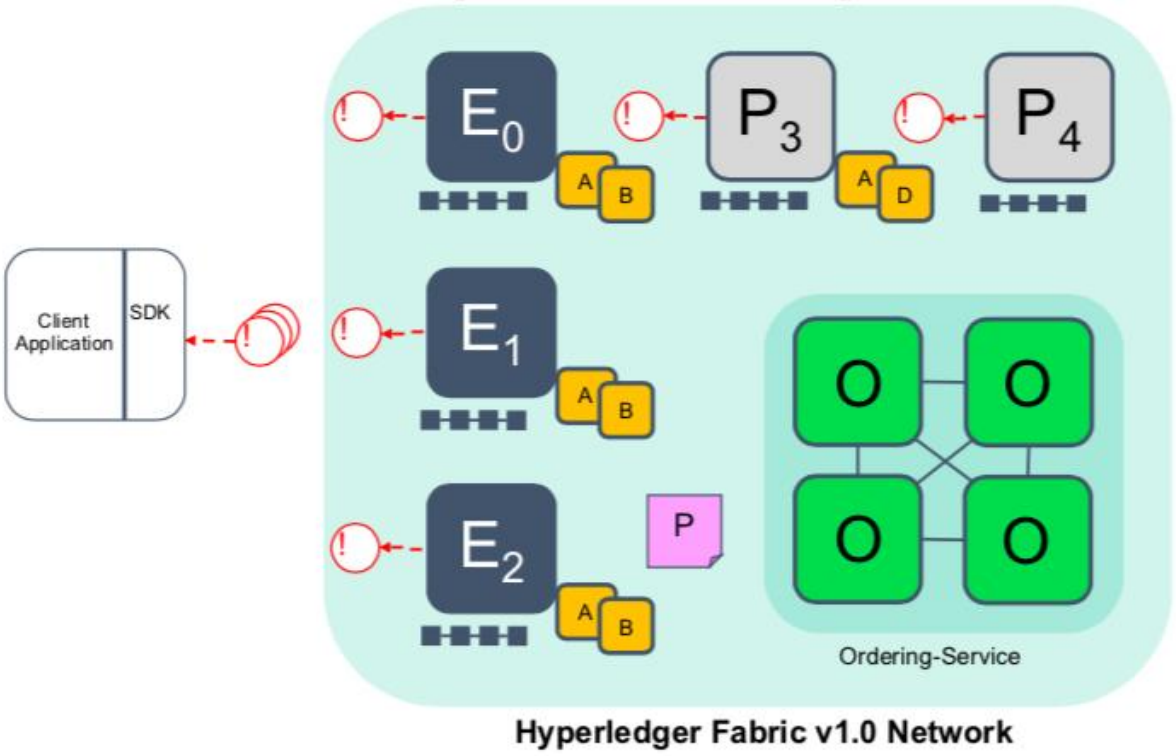
Hyperledger Fabric v1.0 Network

6. Committing peers validate transactions

- Every committing peer validates against the endorsement policy. Also check RW sets are still valid for current world state.
- Validated transactions are applied to the world state and retained on the ledger
- Invalid transactions are also retained on the ledger but do not update world state



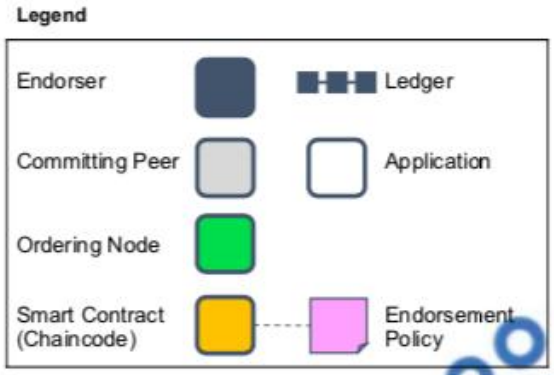
Transaction process: Step 7/7- Notify



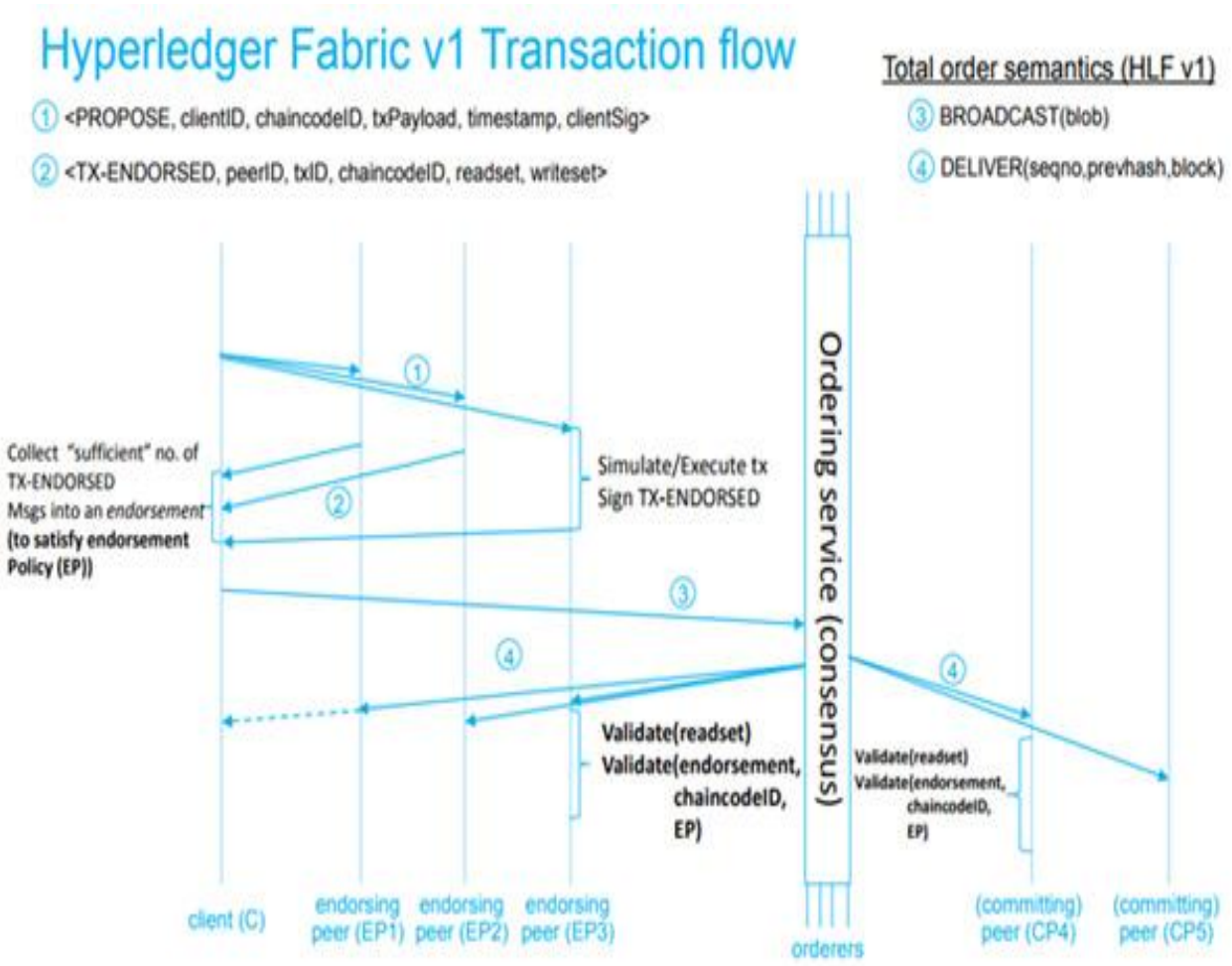
Hyperledger Fabric v1.0 Network

7. Committing peers notify applications

- Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger (event trigger)
- Applications will be notified by each peer to which they are connected



Transaction process（总结）



1. 客户端构造交易提案，并向一个或多个背书节点发送背书请求；
2. 背书节点执行chaincode模拟交易，但并不将结果提交到本地，只是将结果返回给应用；
3. 客户端收集到所有背书节点的结果后，将结果广播给orderers（共识服务）；
4. 共识排序，生成新区块，通过消息通道将block发布给peer节点，各个peer节点验证交易，并提交到本地账本。

Fabric对双花问题的解决

区块链的“双花”问题：

- “双花”，又名“双重支付”，即一笔钱被花了两次或者两次以上。

- A只有20元钱，它向B转账20元，也向C转账20元。Fabric如何避免双花?

Fabric对双花问题的解决

区块链的“双花”问题：

- 客户A分别发送一笔给b转20元，以及给C转20元的交易提案，背书节点会进行验证和产生读写集，在背书节点这一步，两个交易提案都能通过，产生的读集和写集都是：

读集：Key=balance，value=20，version=1

写集：Key=balance，value=0，version=2

背书节点为两份读写集签名后返回client ，client将两个交易提交到order节点，由于order是依据收到交易的时间顺序对交易进行全排序，因此，一定会有其中一个交易排在前面，譬如说，是a转给B 20元这个交易排在前面

Fabric对双花问题的解决

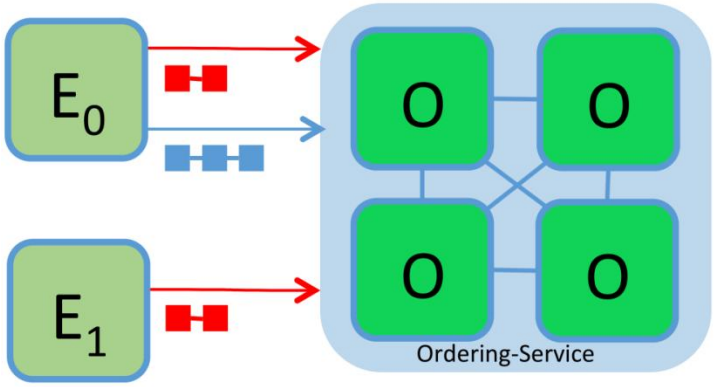
区块链的“双花”问题：

提交节点会接收新区块并对交易内容进行一一校验，首先，他会校验a转给B20元这个交易生成的读写集，Key=balance，value=0，version=2，没有问题，验证通过，更新账本，a给b转帐20元成功。

随后，他会拿到之后的各个交易进行验证，将再次校验a转给C20元这个交易生成的读写集，此时发现问题，由于之前账本已经更新balance的version为version=2了，现在正在校验的还是version=2，导致版本号不符，第二笔交易拒绝，因此，A转给c20元失败。于是，防止简单的双花问题。

Channel

Nodes send/receive messages to the ordering-service via channels.



- Enables chaincode privacy
  - Chaincode deployed to certain nodes
- Messages partitioned into separate channels
  - Transactions stored depending on node and channel
- Nodes can connect to one or more channels

通道是节点与单个ordere节点或odere集群之间建立的一个具有保密性的通信链路！！！

Channel

总结：

1、在Fabric 的网络中，可能同时存在多个彼此隔离的通道，每个通道包含一条私有的区块链和一个私有账本，通道中可以实例化一个或多个链码，以操作区块链上的数据。由此可见，Fabric 是以通道为基础的多链多账本系统。

2、每个通道中有唯一的账本，由通道中所有成员共同维护着这个账本，每个确认节点上都保存了它所属通道的账本的一个副本，因而是分布式账本。对账本的访问需要通过链码实现对账本键值对的增加、删除、更新和查询等的操作。

Channel

总结：

3、通道由排序服务管理。在创建通道的时候，需要定义它的成员和组织、锚节点（anchor peer）和排序服务的节点，一条和通道对应的区块链结构也同时生成，用于记录账本的交易，通道的初始配置信息记录在区块链的创世块（第一个区块）中。通道的配置信息可以用增加一个新的配置区块来更改。

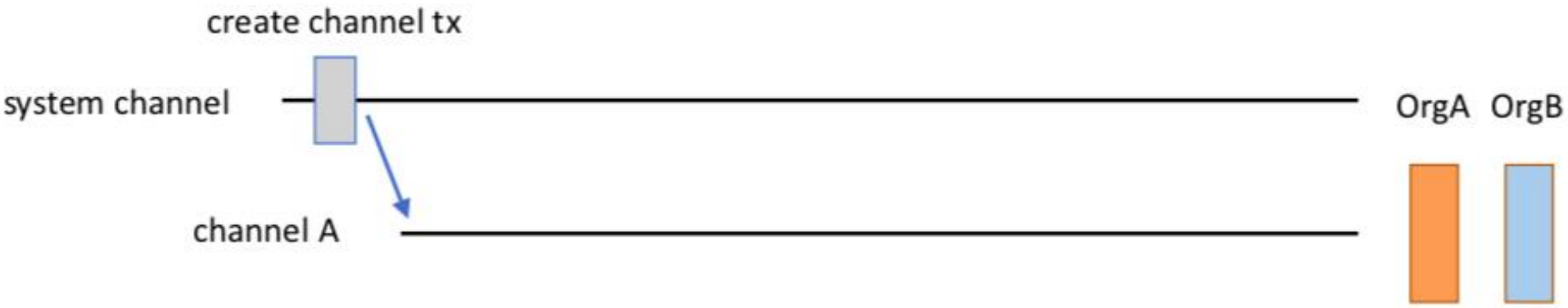
4、每个组织可有多个节点加入同一个通道，这些节点中可以指定一个锚节点（或多个锚节点做备份）。锚节点代表本组织与其他组织的节点交互，从而发现通道中的所有节点。另外，同一组织的节点会选举或指定主导节点（leading peer），主导节点负责接收从排序服务发来的区块，然后转发给本组织的其他节点。主导节点可以通过特定的算法选出，因此保证了在节点数量不断变动的情况下仍能维持整个网络的稳定性。

Fabric Channel Running method

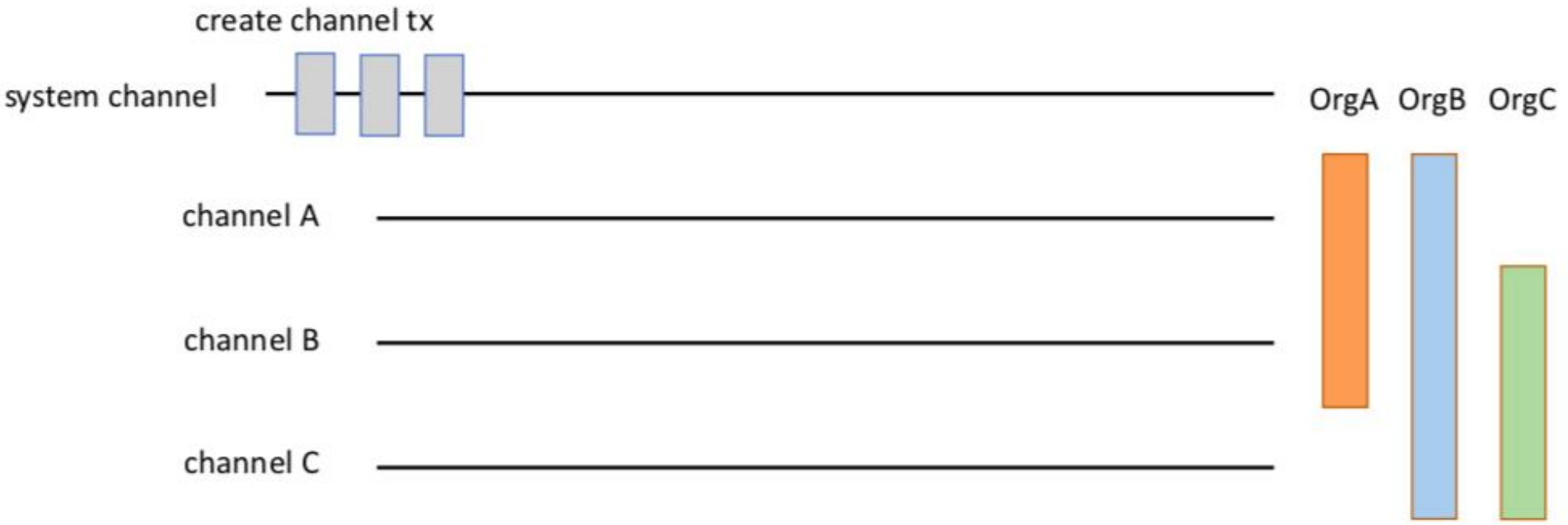
system channel

Fabric Channel Running method

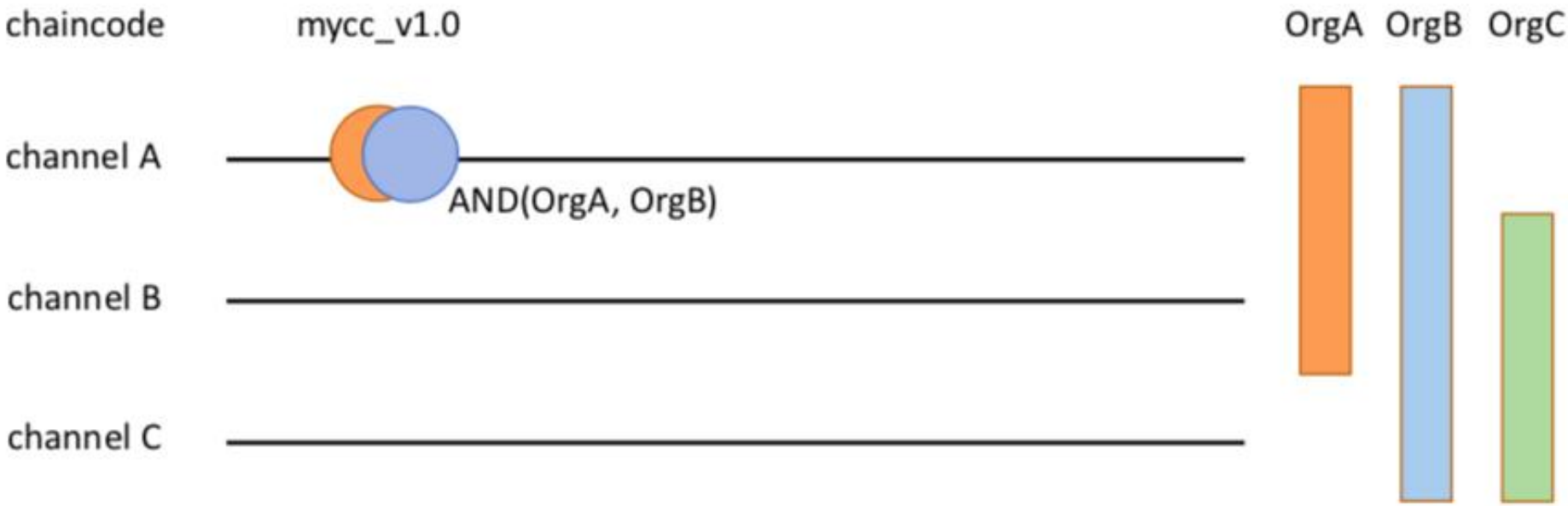
Fabric Channel Running method



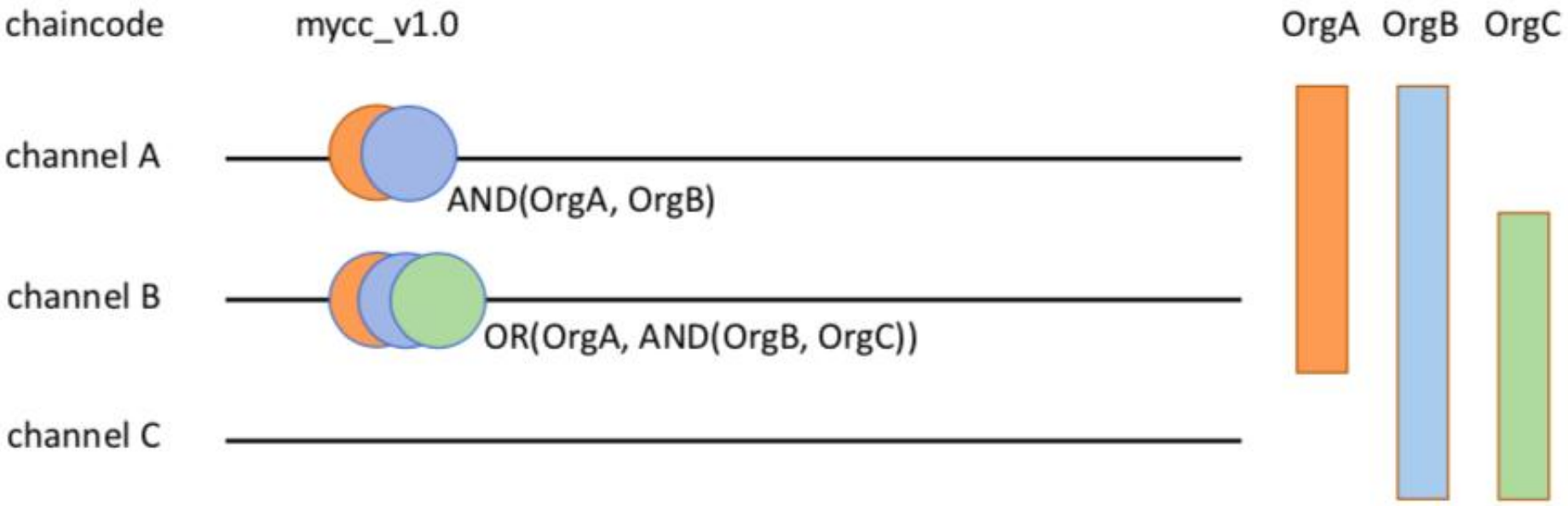
Fabric Channel Running method



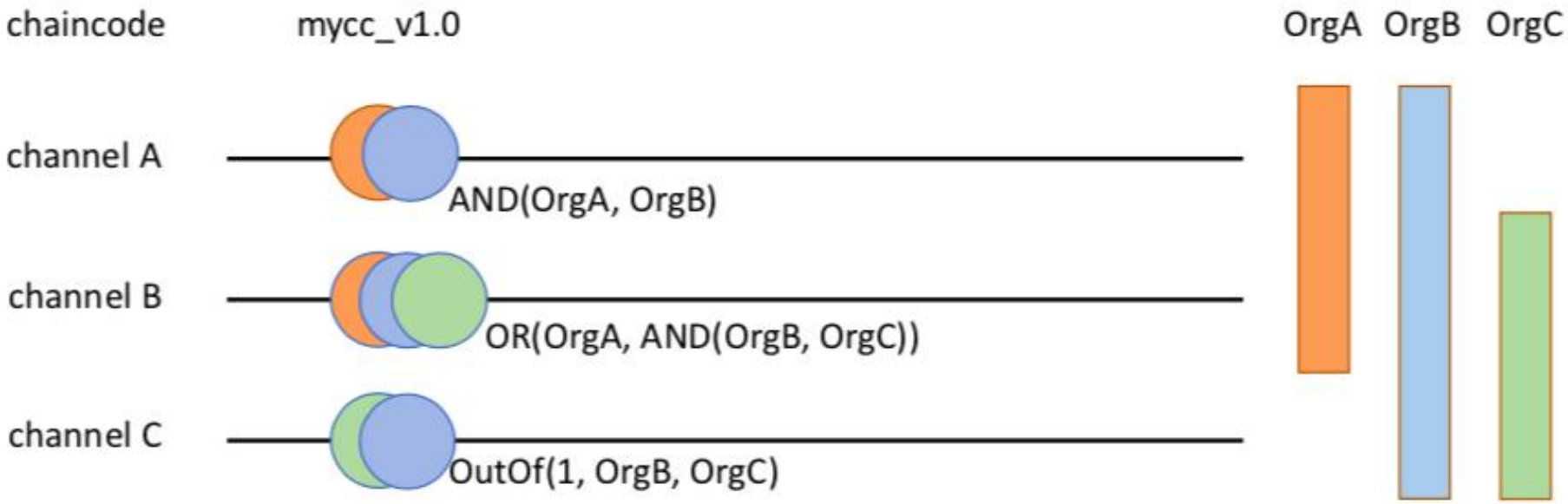
Fabric Channel Running method



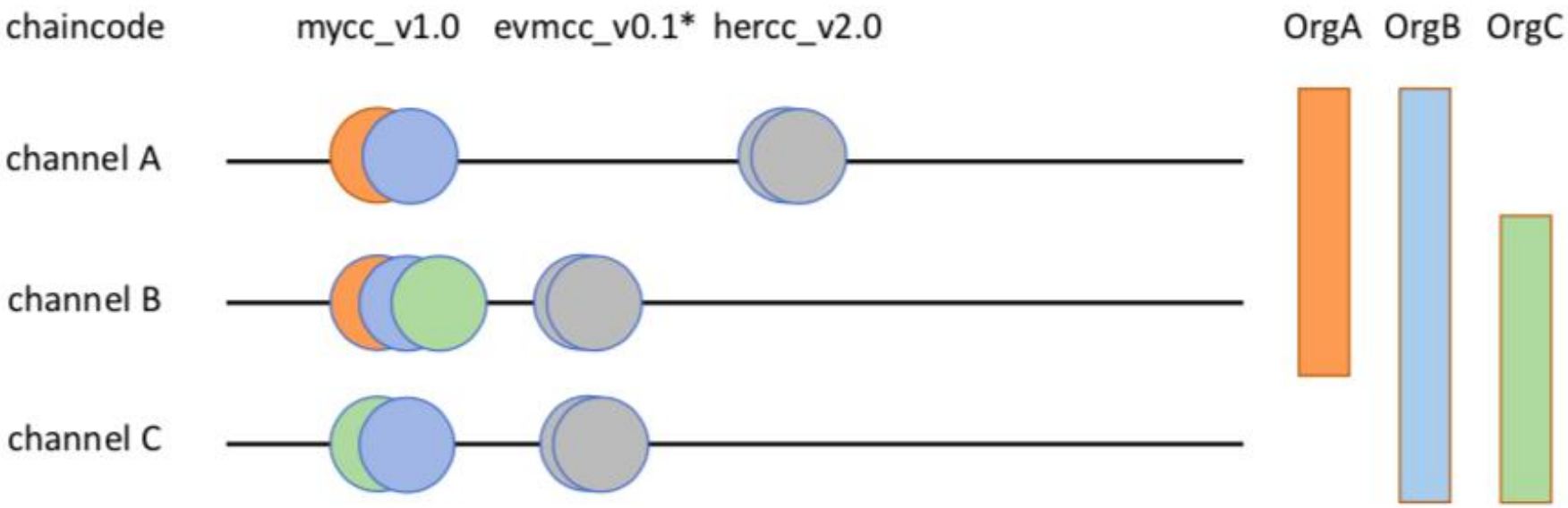
Fabric Channel Running method



Fabric Channel Running method



Fabric Channel Running method

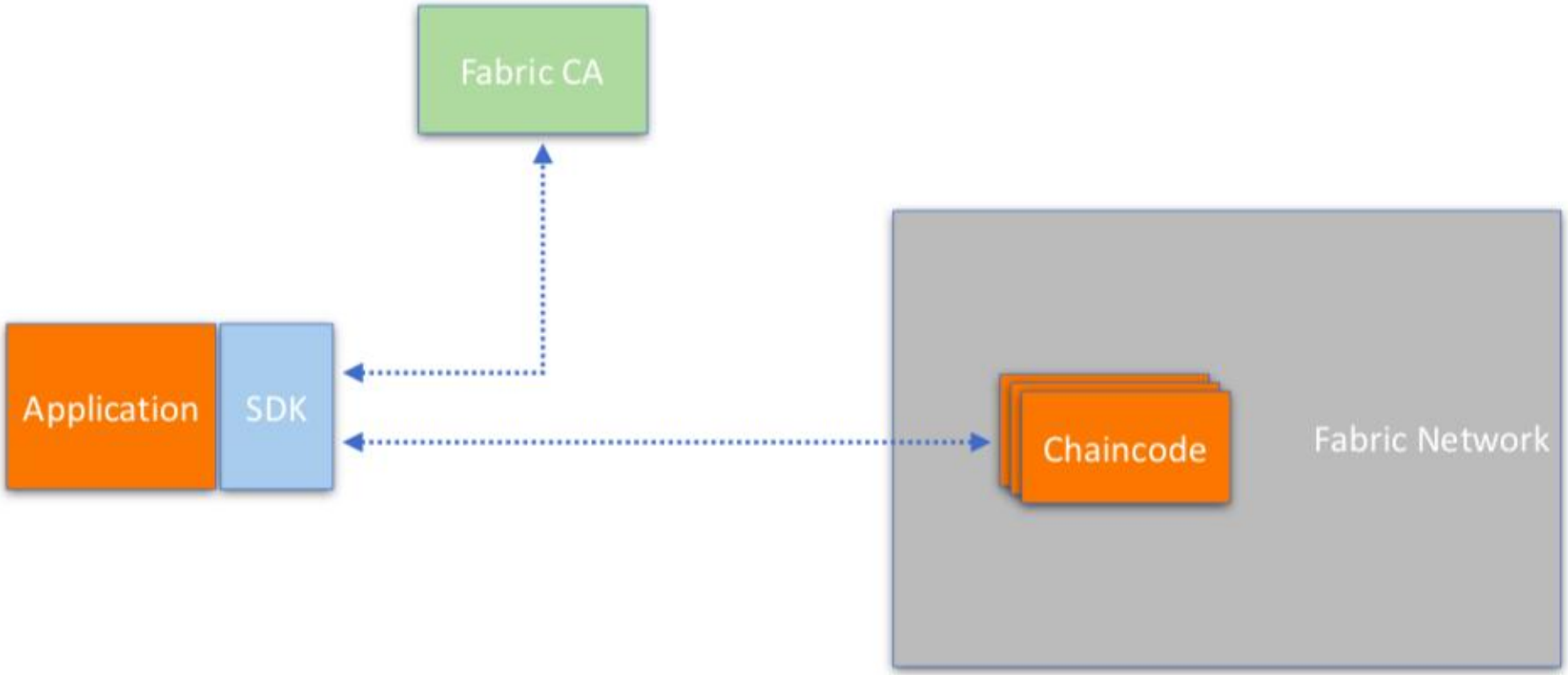


Coming soon

- Private Data
- Pluggable E/V Chaincode
- Attribute-based Access Control
- State Based Endorsement
- Connection profiles
- Service Discovery
- Metrics
- EVM Support (Run Ethereum smart contracts on Fabric)
- Orderer consensus (Solo/Kafka/Raft)
- ...

Part 2.2

Develop Applications

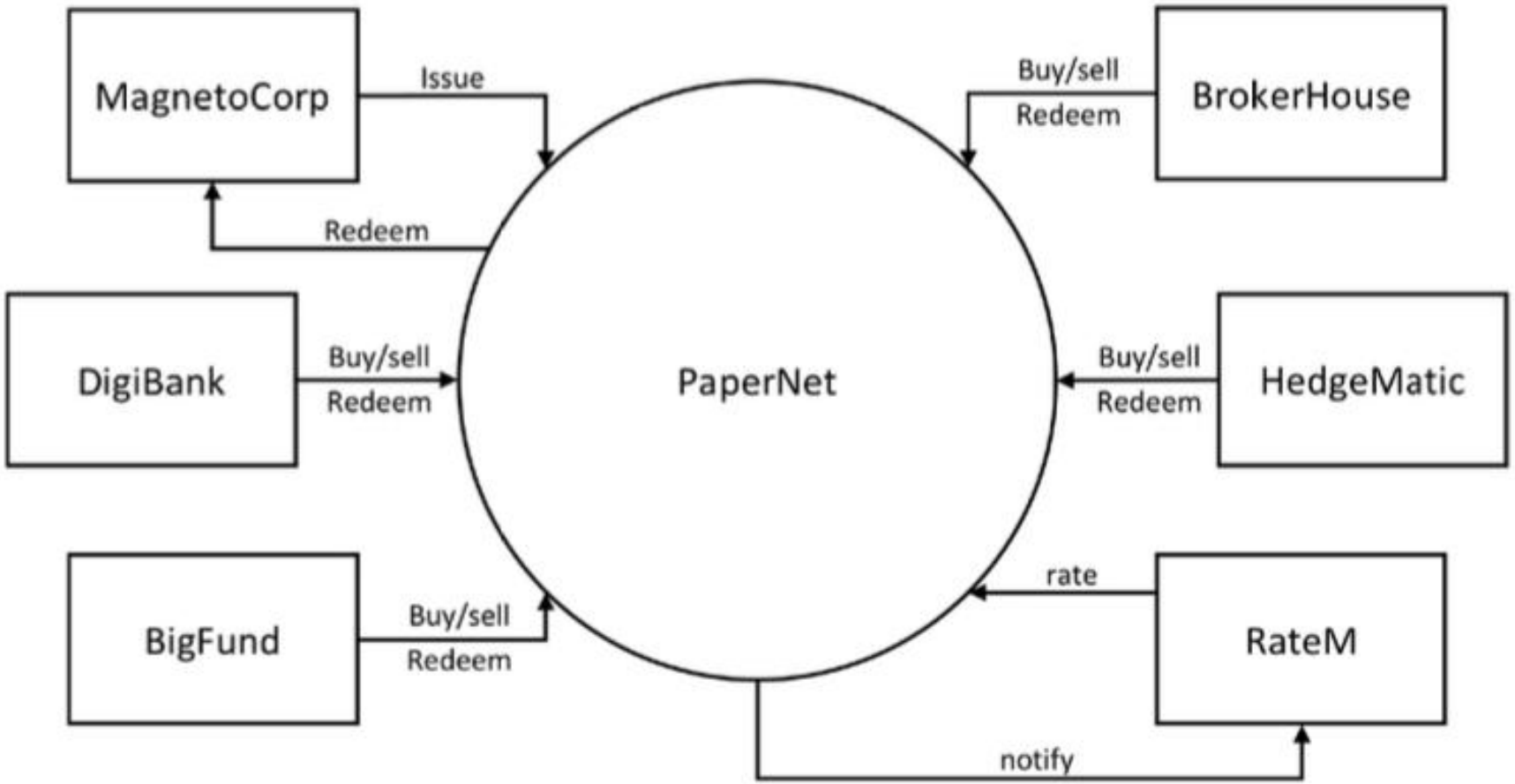


Develop Applications

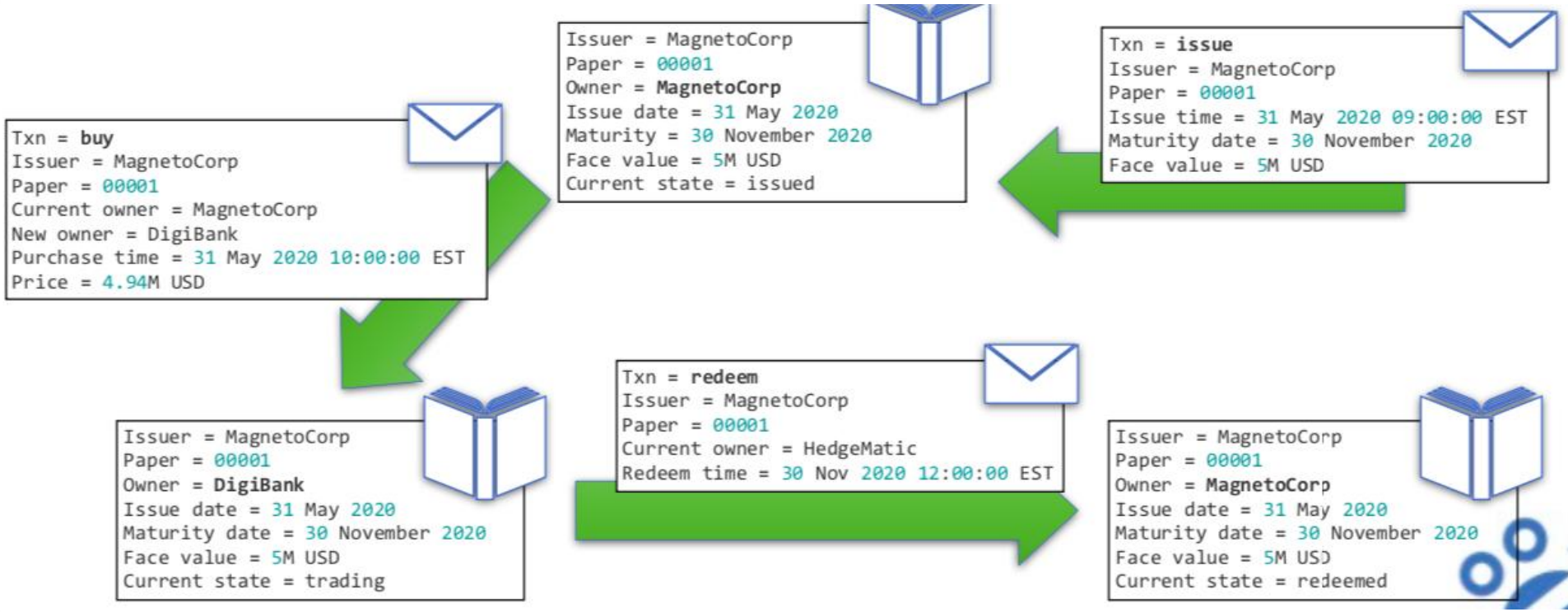

```
// ChaincodeStubInterface is used by deployable chaincode apps to access and
// modify their ledgers
type ChaincodeStubInterface interface {
    // GetArgs returns the arguments intended for the chaincode Init and Invoke
    // as an array of byte arrays.
    GetArgs() [][]byte

    InvokeChaincode(chaincodeName string, args [][]byte, channel string) pb.Response

    // GetState returns the value of the specified `key` from the
    // ledger. Note that GetState doesn't read data from the writeset, which
    // has not been committed to the ledger. In other words, GetState doesn't
    // consider data modified by PutState that has not been committed.
    // If the key does not exist in the state database, (nil, nil) is returned.
    GetState(key string) ([]byte, error)

    // PutState puts the specified `key` and `value` into the transaction's
    // writeset as a data-write proposal. PutState doesn't effect the ledger
    // until the transaction is validated and successfully committed.
    // Simple keys must not be an empty string and must not start with null
    // character (0x00), in order to avoid range query collisions with
    // composite keys, which internally get prefixed with 0x00 as composite
    // key namespace.
    PutState(key string, value []byte) error
```

core/chaincode/shim/interfaces.go

Develop Applications-1 Scenario



Develop Applications-2 Lifecycle



Develop Applications-3 Data Structure

commercial paper: MagnetoCorp paper 00004

Issuer :	Paper:	Owner:	Issue date:	Maturity date:	Face value:	Current state:
MagnetoCorp	00004	DigiBank	31 August 2020	31 March 2021	5m USD	issued

commercial paper list: org.papernet.paper

add

Issuer :	Paper:	Owner:	Issue date:	Maturity date:	Face value:	Current state:
MagnetoCorp	00001	DigiBank	31 May 2020	31 December 2020	5m USD	trading
MagnetoCorp	00002	BigFund	30 June 2020	31 January 2021	5m USD	trading
MagnetoCorp	00003	BrokerHouse	31 July 2020	28 February 2021	5m USD	trading

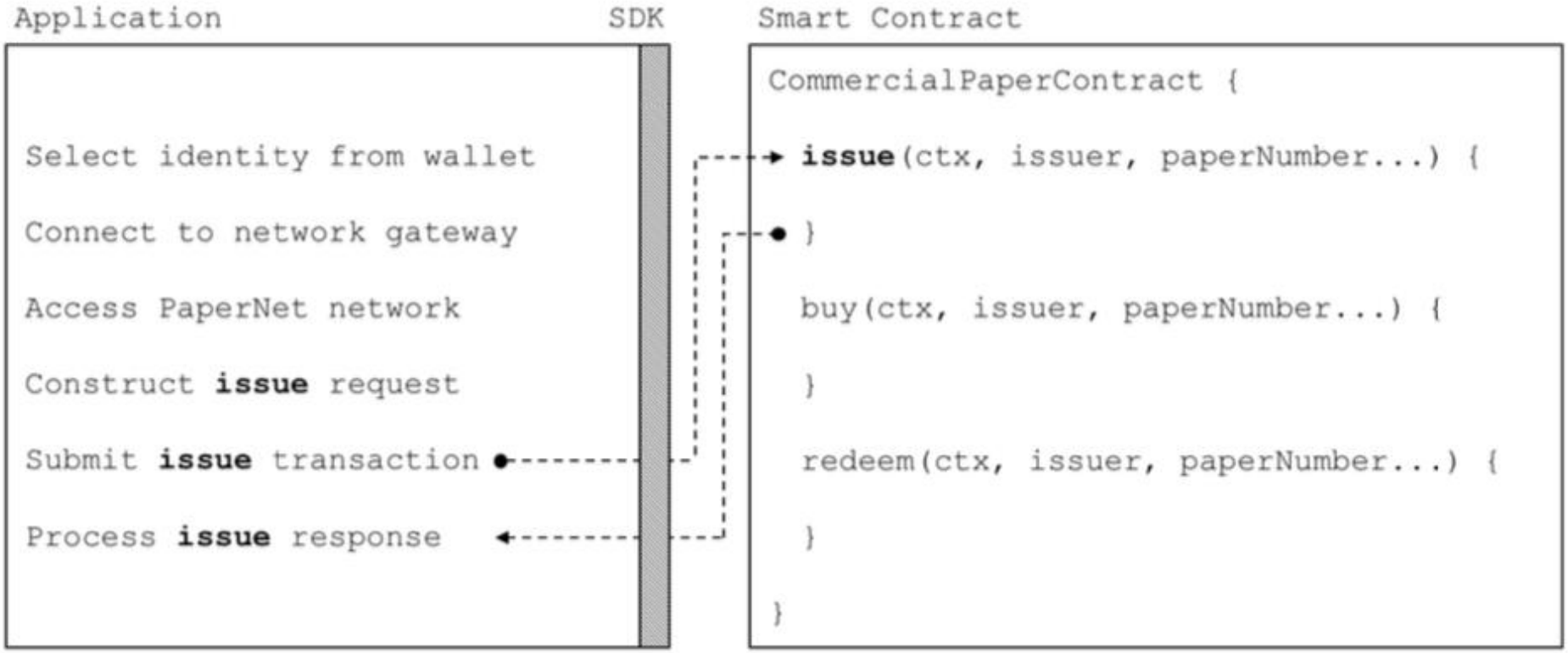
key: org.papaernet.paperMagnetoCorp00001

key	value
org.papernet.paperMagnetoCorp00001	**Issuer :** MagnetoCorp, **Paper:** 00001, **Owner:** DigiBank, **Issue date:** 31 May 2020, **Maturity date:** 31 December 2020, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00002	**Issuer :** MagnetoCorp, **Paper:** 00002, **Owner:** BigFund, **Issue date:**, 30 June 2020, **Maturity date:** 31 January 2021, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00003	**Issuer :** MagnetoCorp, **Paper:** 00003, **Owner:** BrokerHouse, **Issue date:** 31 July 2020,, **Maturity date:** 28 February 2021, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00004	**Issuer :** MagnetoCorp, **Paper:** 00004, **Owner:** DigiBank, **Issue date:** 31 August 2020, **Maturity date:** 31 March 2021, **Face value:** 5m USD, **Current state:** issued

Develop Applications-4 Smart contract

```
55      /**
56       * Issue commercial paper
57       *
58       * @param {Context} ctx the transaction context
59       * @param {String} issuer commercial paper issuer
60       * @param {Integer} paperNumber paper number for this issuer
61       * @param {String} issueDateTime paper issue date
62       * @param {String} maturityDateTime paper maturity date
63       * @param {Integer} faceValue face value of paper
64      */
65      async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime, faceValue) {
66
67          // create an instance of the paper
68          let paper = CommercialPaper.createInstance(issuer, paperNumber, issueDateTime, maturityDateTime, faceValue);
69
70          // Smart contract, rather than paper, moves paper into ISSUED state
71          paper.setIssued();
72
73          // Newly issued paper is owned by the issuer
74          paper.setOwner(issuer);
75
76          // Add the paper to the list of all similar commercial papers in the ledger world state
77          await ctx.paperList.addPaper(paper);
78
79          // Must return a serialized paper to caller of smart contract
80          return paper.toBuffer();
81      }
82
```

Develop Applications-5 Frontend



Thanks!