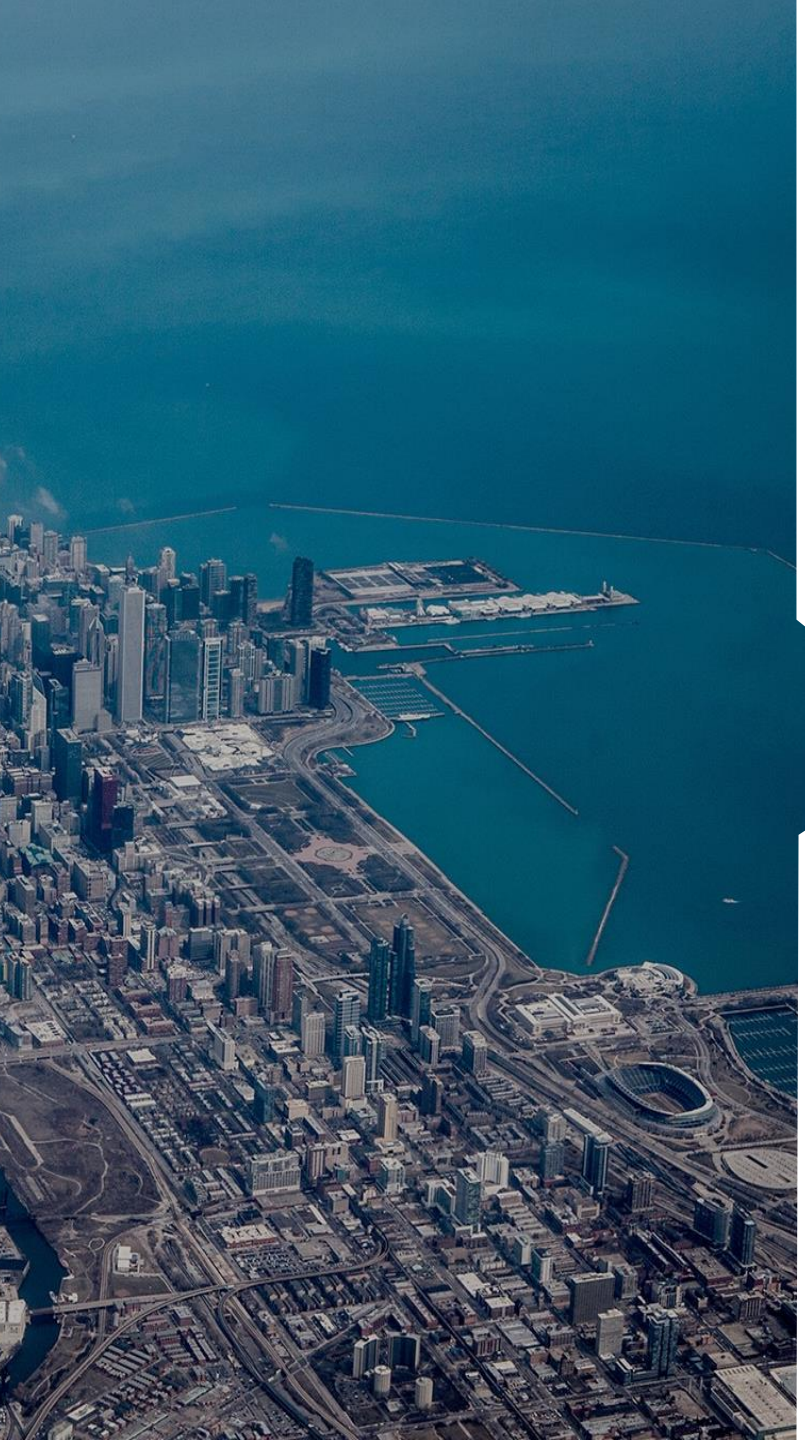
区块链技术与应用

华南理工大学　　许可　　kexu@scut.edu.cn

本课件主要内容来源于IBM

Unit 2 Hyperledger社区和 Fabric架构



1 Hyperledger社区

2 Hyperledger Fabric架构

Part ONE

Hyperledger 社区

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。

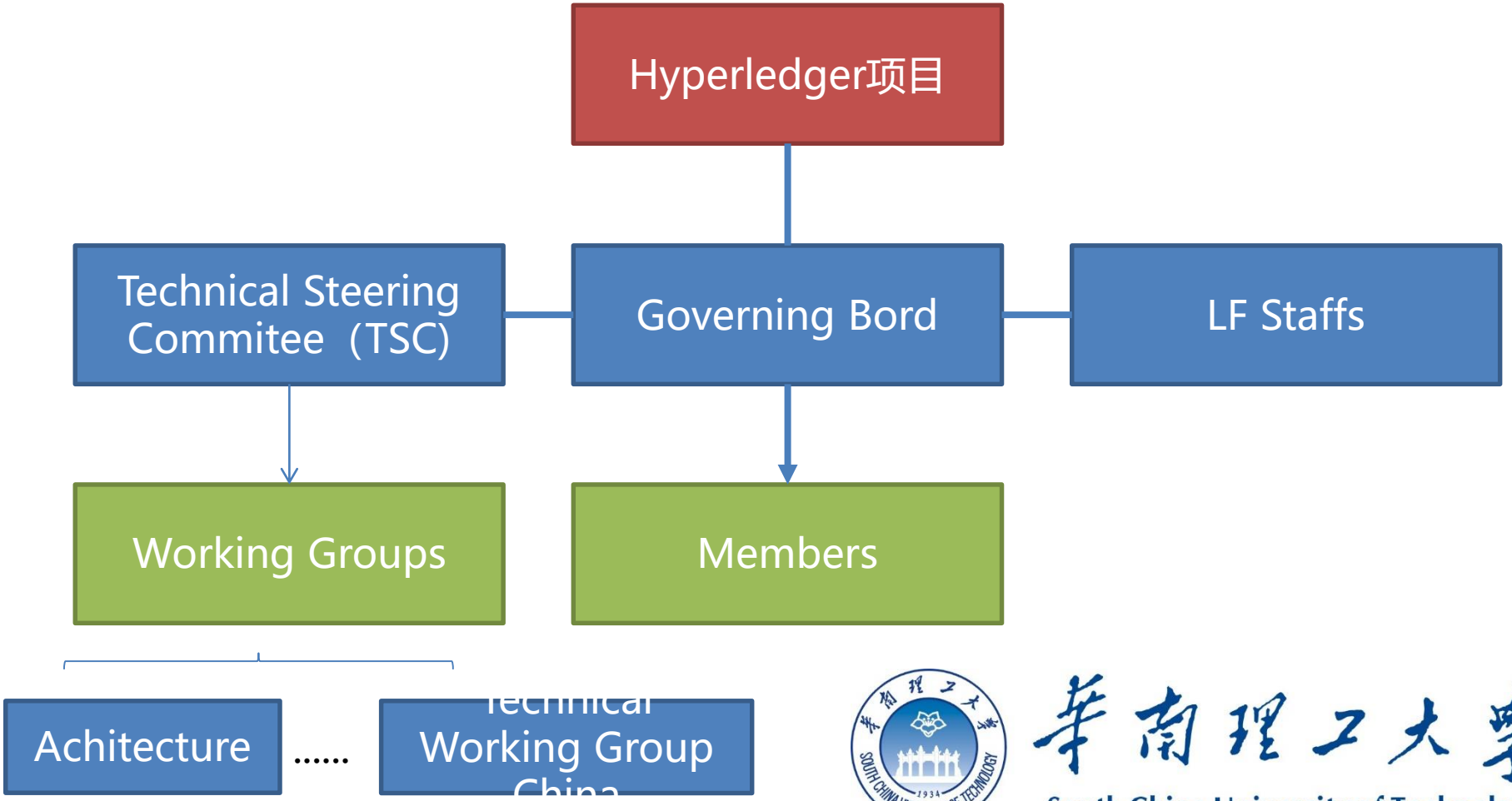
- Hyperledger项目的官方网站：hyperledger.org，Hyperledger项目的代码是托管在Gerrit上，并通过GitHub提供镜像。

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。

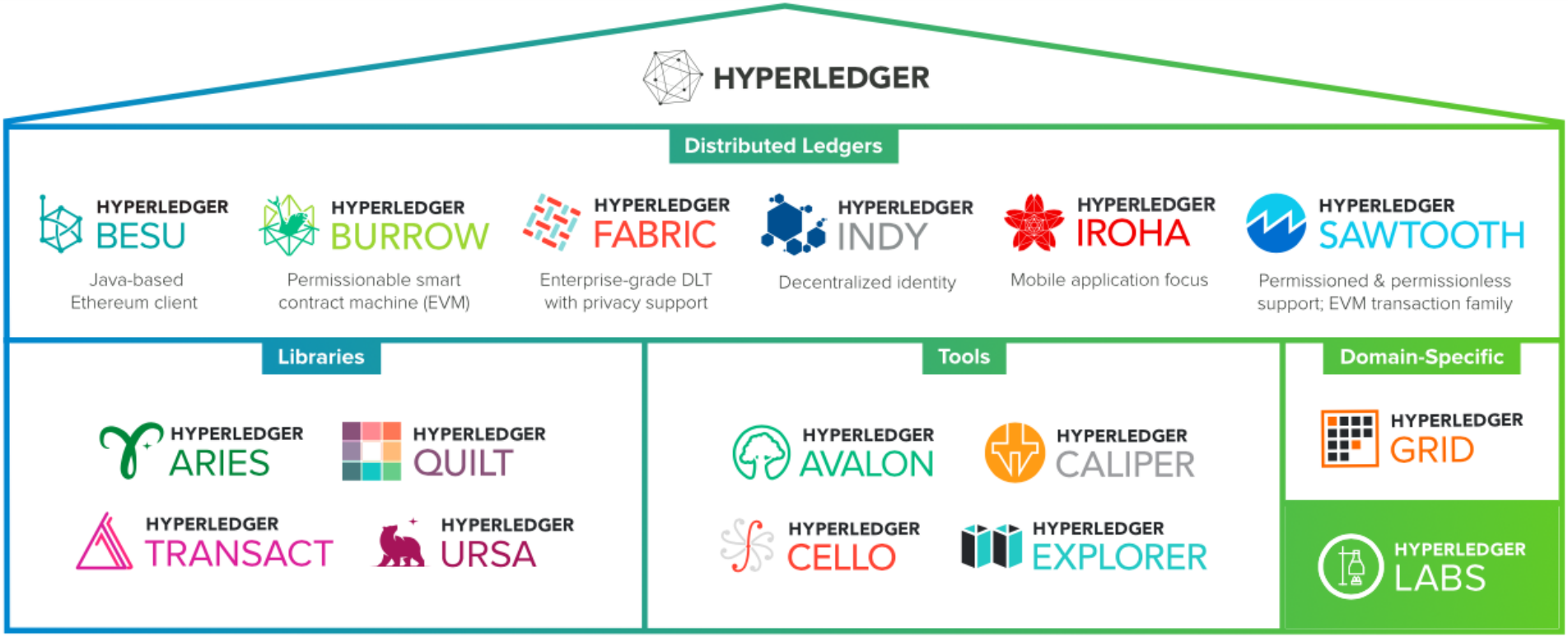
- Hyperledger是发展最快的Linux基金会项目之一，拥有超过280个成员。

- 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。

华南理工大学
South China University of Technology

Hyperledger社区

The Hyperledger Greenhouse（超级账本温室）



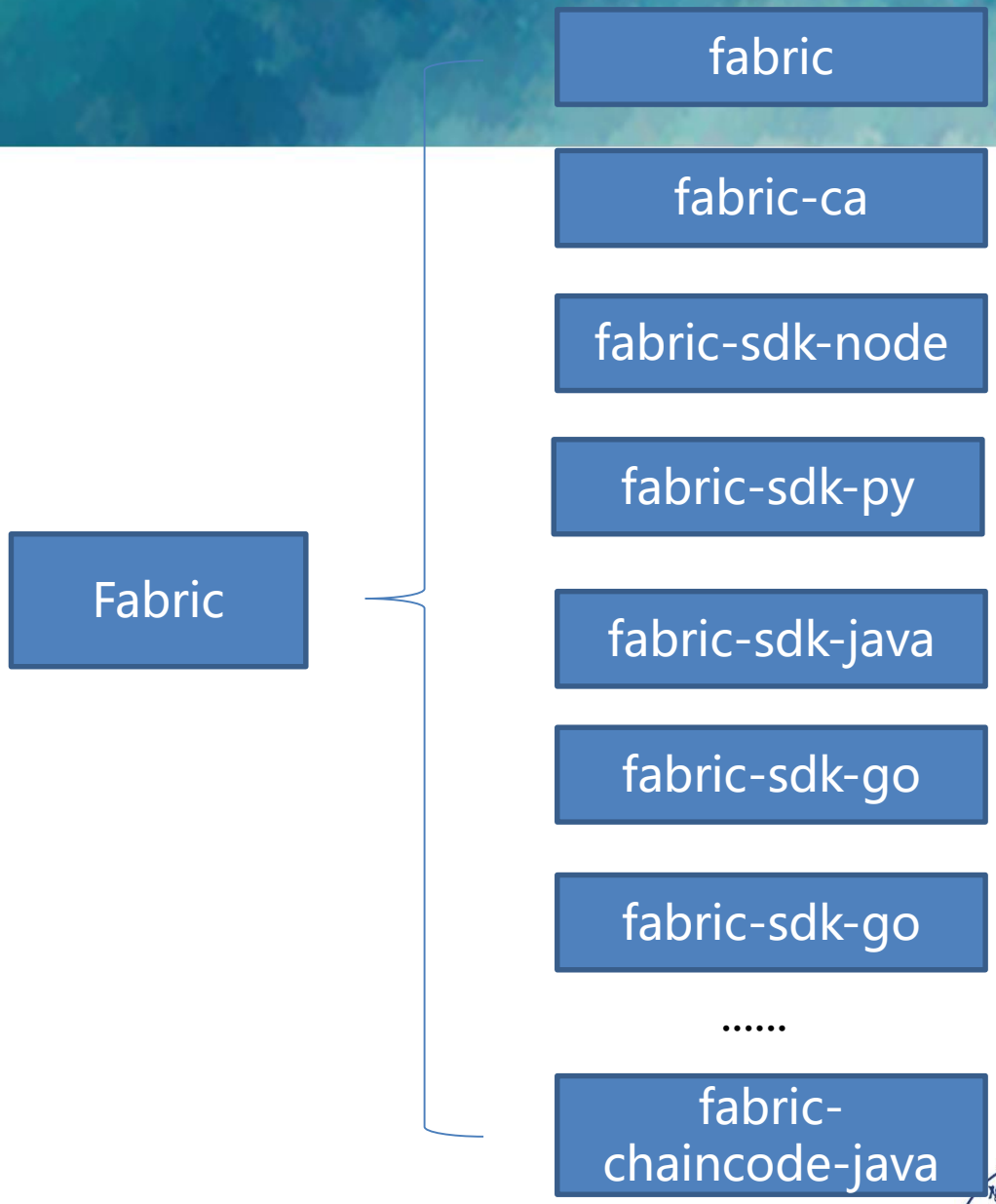
Hyperledger社区

The Hyperledger Greenhouse
分布式账本框架类子项目

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，最新版本为v2.0
- Fabric包括Fabric CA、Fabric SDK（包括Node.Js、Python和Java等语言）和fabric-api等子项目


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

Hyperledger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）

Hyperledger社区

Hyperledger框架类子项目

- Sawtooth（锯齿湖）由Intel等企业于2016年4月提交。项目的Github地址：https://github.com/hyperledger/sawtooth-core
- Sawtooth项目的定位是分布式账本平台，实现了低能耗的支持全新的基于硬件芯片的共识机制Proof of Elapsed Time（PoET），并支持交易族（Transaction Family），提供了一些典型的交易模版，方便用户快速开发应用
- Sawtooth基于Python语言实现
- Sawtooth支持联盟链和非联盟链的部署

Hyperledger社区

Hyperledger框架类子项目

- Iroha由Soramitsu和日立等企业于2016年10月提交。项目的Github地址：https://github.com/hyperledger/iroha
- Iroha项目的定位是分布式账本平台。其设计上类似Fabric，同时考虑了移动端和Web端的需求，有助于超级账本社区中的移动应用（mobile applications）发展
- Iroha基于C++语言实现，包括iroha-android、iroha-ios等模块

Hyperledger社区

Hyperledger框架类子项目

- Burrow由Monax、Intel等企业于2017年4月提交。项目的Github地址：https://github.com/hyperledger/burrow
- Burrow 是Hyperledger中第一个源于以太坊框架的项目
- 是一个以太坊衍生代码库，提供以太坊虚拟机（EVM）的支持，实现支持高效交易的带权限的区块链平台
- Burrow基于Golang语言实现的。

Hyperledger社区

Hyperledger框架类子项目

- Indy项目由Sovrin基金会牵头进行开发，致力于打造一个基于区块链和分布式账本技术的数字中心管理平台。该平台支持去中心化身份认证(decentralized identity)的分布式账本，支持跨区块链和跨应用的操作，可实现全球化的身份管理。该项目于2017年3月底加入Hyperledger。

- 举例：在传统的社会中每个人可能拥有多个身份，公司员工、车主、房屋所有人等。这些身份都存在于各自独立的系统中，互相之间没有联系。Indy试图将这些身份信息统一处理，提供一套独立于任何系统的数字身份账本来解决身份统一管理的问题。

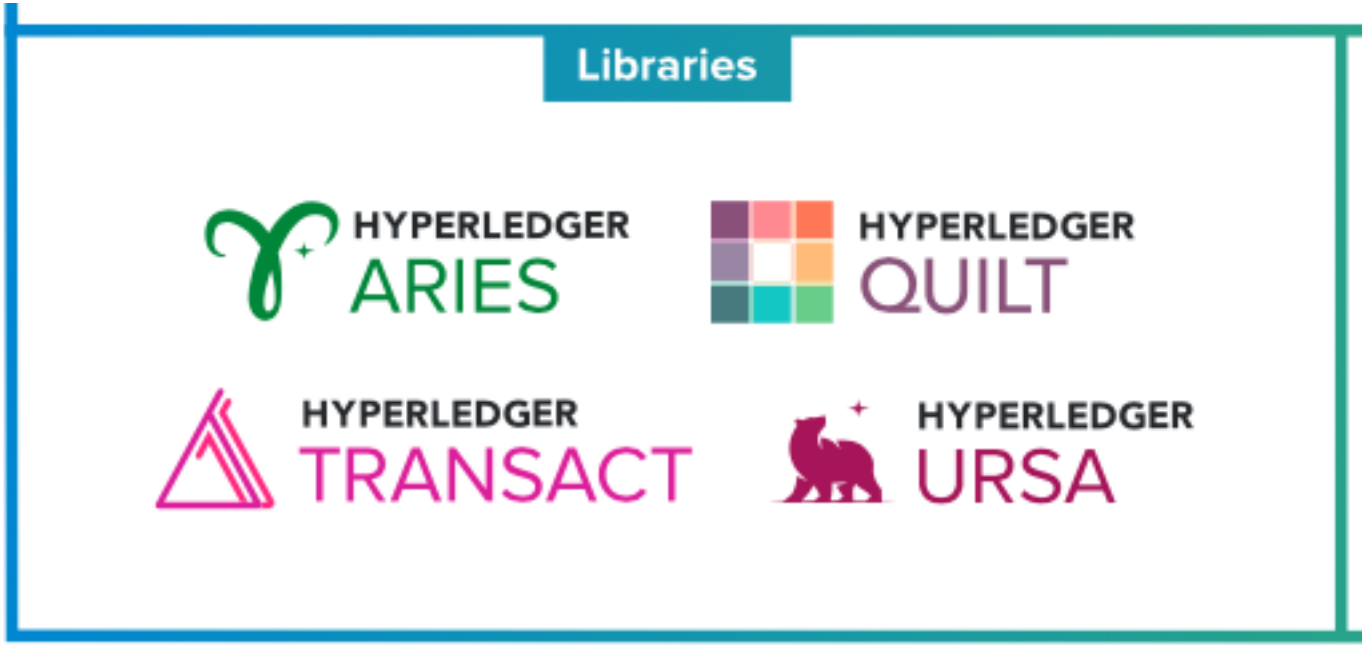
Hyperledger社区

Hyperledger框架类子项目

- Besu子项目于2019年9月加入Hyperleger，项目的Github地址：https://github.com/hyperledger/besu
- Besu是一个基于Java的以太坊客户机，也是提交给超级账本的第一个可以在公链上运行的区块链项目。
- Besu的愿景是为以太坊主网和企业级客户提供了最可靠、具有可扩展性和易用的平台
- Besu包含了PoW、PoA（Proof of Authority权威证明）等在内的几个共识算法，并且拥有专门为联盟链环境中使用而设计的全面的安全方案。

Hyperledger社区

The Hyperledger Greenhouse
程序库子项目



Hyperledger社区

Hyperledger程序库类子项目

- URSA项目在2018年11月开源，其Github地址：https://github.com/hyperledger/ursa
- 目标是：实现密码学成果的工具库。它提供一个可信任，易使用的方式简化和整合加密库，以便这个加密软件库能以一种可互操作的方式用于基于分布式账本技术的项目。
- Ursa 基于Rust语言实现，包含两个模块：base crypto 和 Z-mix

Hyperledger社区

- Ursa 项目拥有一个包括模块化签名和对称密钥原语（primitive）功能的综合库，开发人员可以通过配置使用不同的加密方案，而无需修改代码。同时，Ursa还将包括新一代的加密技术，包括基于双线性对的（pairing-based）签名，阈值（threshold）签名和聚合（aggregate）签名。 此外，还包括像 SNARK这样的零知识（zero-knowledg）原语。

- Ursa使得每个项目可以在无需理解基础数学算法的情况下，轻松使用各种可变的加密算法。

Hyperledger社区

Hyperledger程序库类子项目

- Quilt由日本的NTT Data在2017年10月贡献至社区。项目的Github地址：https://github.com/hyperledger/quilt
- Quilt是一种支付协议，主要应用于Hyperledger下面的不同区块链产品之间进行价值的传递和转换
- Quilt本质上是Interledger Protocol （ILP）协议的Java实现。
- ILP定义了分布式账本与分布式账本之间、传统账本与分布式账本之间的交互过程

Hyperledger社区

Hyperledger程序库类子项目

- Aries子项目于2019年5月加入Hyperleger，项目的Github地址 https://github.com/hyperledger/aries
- 它是一个共享的、可重用的、可互操作的基于区块链的身份管理工具包。
- 它是基于区块链的点对点交互的基础设施。
- 目标和愿景是：提供一种为实现安全通信构建可互操作且可验证的资质证明的方法，在未来完全替代纸质身份证件

华南理工大学
South China University of Technology

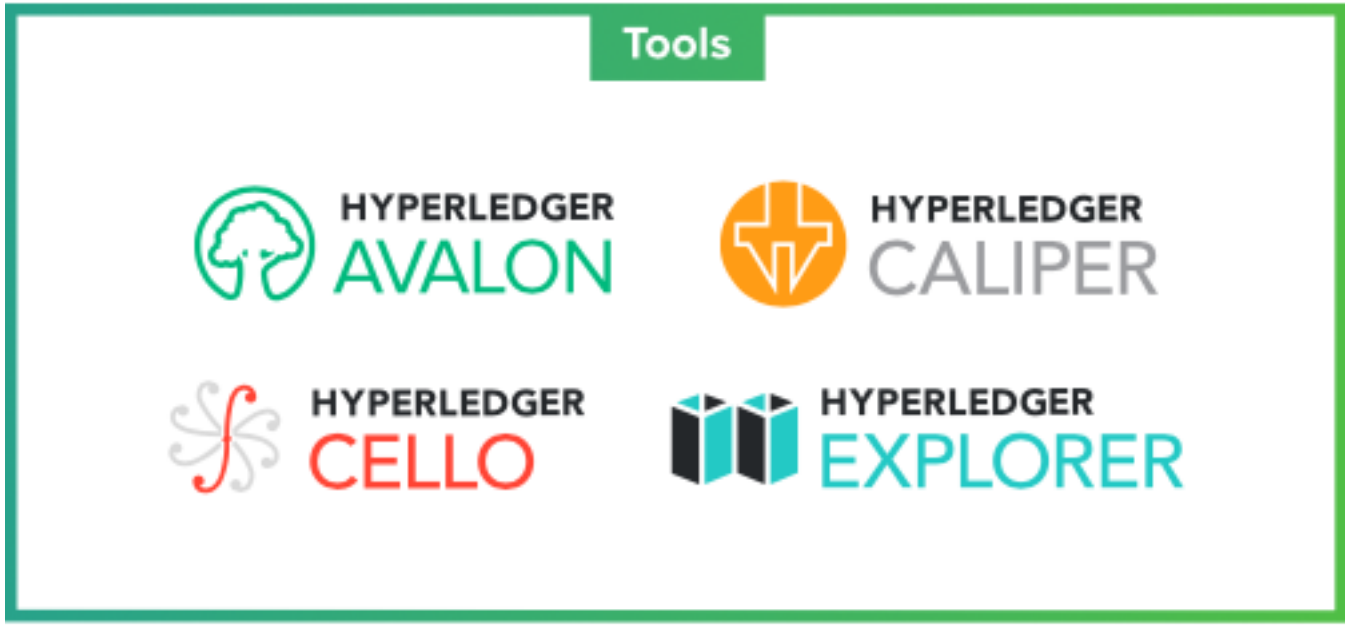
Hyperledger社区

Hyperledger程序库类子项目

- Transact于2019年6月贡献至社区。项目的地址：https://crates.io/crates/transact
- Hyperledger Transact提供了一个与平台无关的程序库——“智能合约引擎”，作为实现处理智能合约的处理器或解释器，可减少编写分布式账本软件的开发工作量，提升区块链网络的可兼容性。

Hyperledger社区

The Hyperledger Greenhouse
工具类子项目



Hyperledger社区

Hyperledger工具类子项目

- Avalon子项目于2019年10月提交，项目的Github地址： https://github.com/hyperledger/avalon
- Avalon 使用“可信计算”维护数据的弹性和完整性。
- 其核心是提供一种受信的计算服务（TCS，Trusted Compute Service），支持受信执行环境（TEE，Trusted Execution Environmen）、零知识证明（ZKP，Zero Knowledge Proofs）和多方计算（MPC，Multi-Party Compute）。

Hyperledger社区

Hyperledger工具类子项目

- Cello由IBM团队于2017年1月底提交。项目的Github地址：https://github.com/hyperledger/cello
- Cello的定位是区块链集成管理平台，同时提供区块链即服务（BaaS），实现区块链环境的快速部署，以及对区块链平台的运行时管理。
- 使用Cello，管理员可以轻松部署和管理多条区块链；应用开发者可以专注于应用开发，而无需关心如何底层平台的管理和维护。
- Cello主要的开发语言为Python和JavaScript等，底层支持包括裸机、虚拟机、容器等多种基础架构

Hyperledger社区

Hyperledger工具类子项目

- Explorer由Intel、DTCC、IBM等企业于2016年8月提交。项目的Github地址：https://github.com/hyperledger/blockchain-explorer
- Explorer的定位是区块链平台的浏览器。提供Web操作界面，用户可以快速查看底层区块链平台的运行信息
- 实现的语言是Node.js

Hyperledger社区

Hyperledger工具类子项目

- Caliper由华为于2018年贡献。项目的Github地址：https://github.com/hyperledger/caliper
- Caliper是一个区块链测试工具用于测评区块链性能
- 基于Node.js语言编写

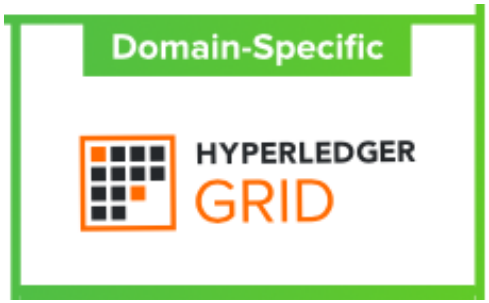
Hyperledger社区

Hyperledger工具类子项目

- Composer由IBM团队于2017年3月底提交。项目的Github地址：https://github.com/hyperledger/composer
- Composer提供了一个Hyperledger Fabric的开发辅助框架，可以简化Fabric应用程序的创建、部署和使用。
- 通过Composer，开发人员可以使用Javascript定义应用逻辑，再加上资源、参与者、交易等模型和访问规则，生成Hyperledger Fabric支持的链码。
- Composer主要基于Node.js语言开发

Hyperledger社区

The Hyperledger Greenhouse
特定领域类子项目



Hyperledger社区

Hyperledger特定领域类子项目

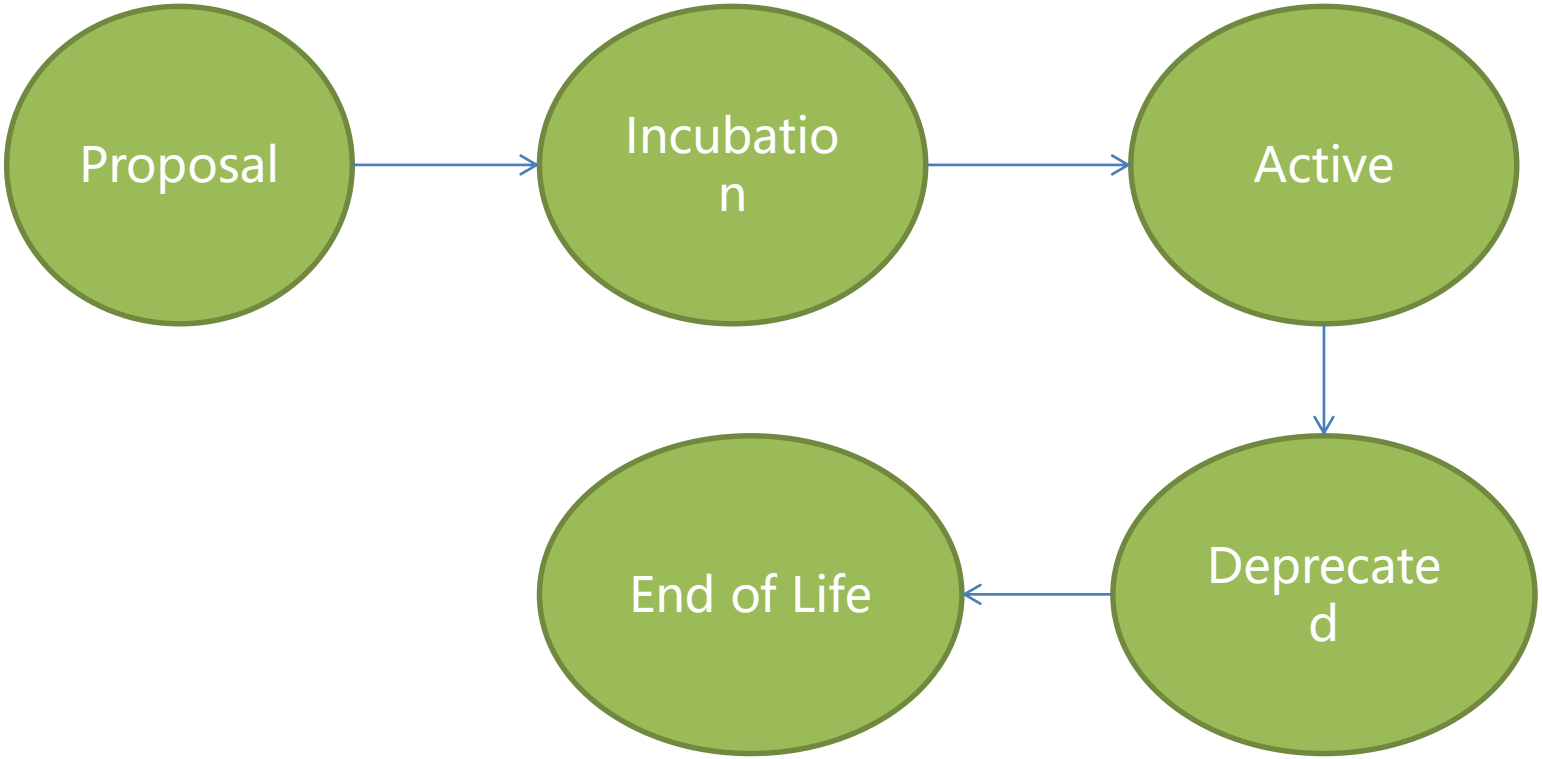
WebAssembly-based project for building supply chain solutions

- Grid项目由Cargill、Intel等企业于2018年12月提交。项目的Github地址：https://github.com/hyperledger/grid
- Grid是一个构建包含分布式分类账组件的供应链解决方案的平台。它提供了一套不断增长的工具，可以加速供应链智能合约和客户机接口的开发。
- Grid主要基于Python语言实现
- 将解决食品安全，可追溯性和贸易结算等问题。

华南理工大学
South China University of Technology

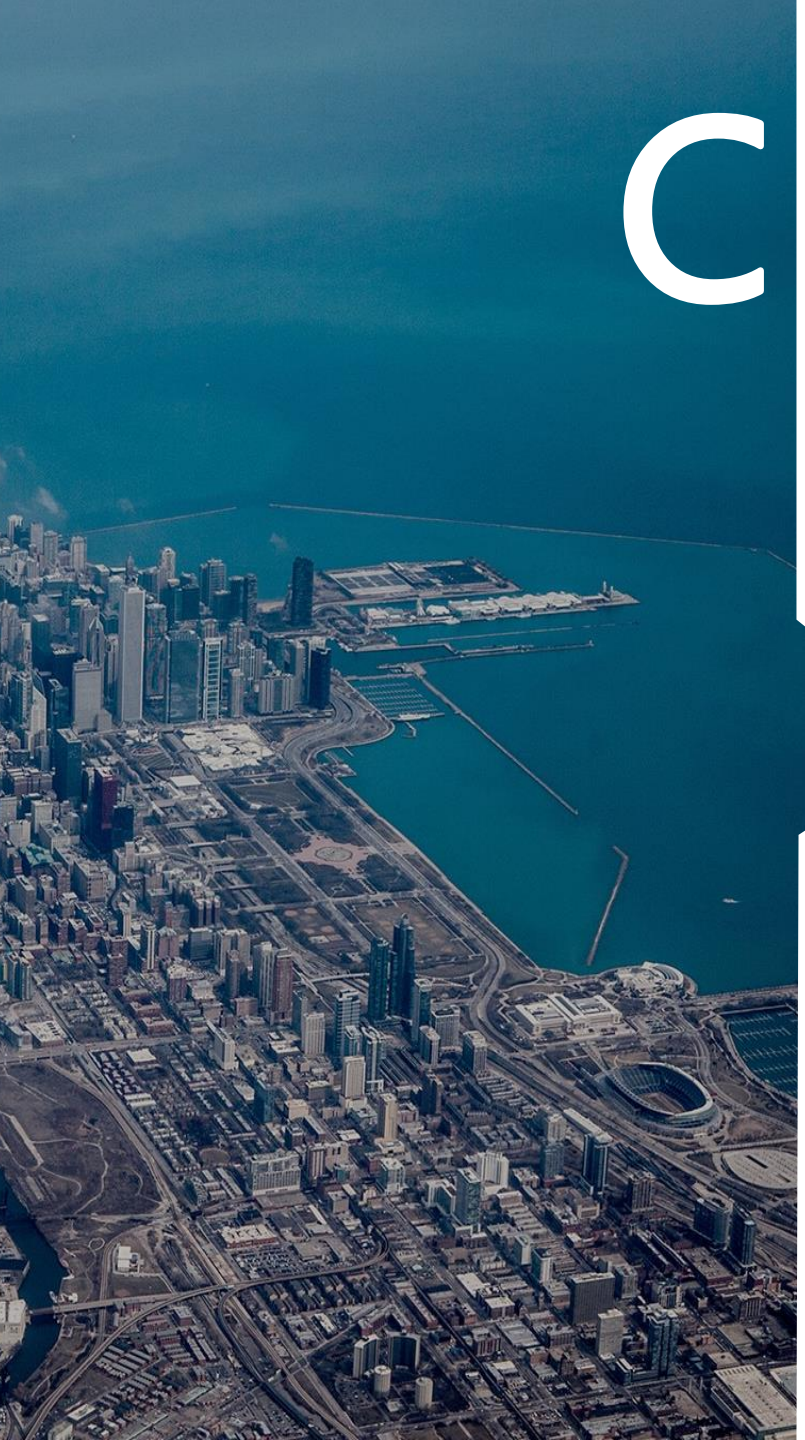
Hyperledger社区

- Hyperledger项目的管理：（生命周期管理模式）



Part TWO

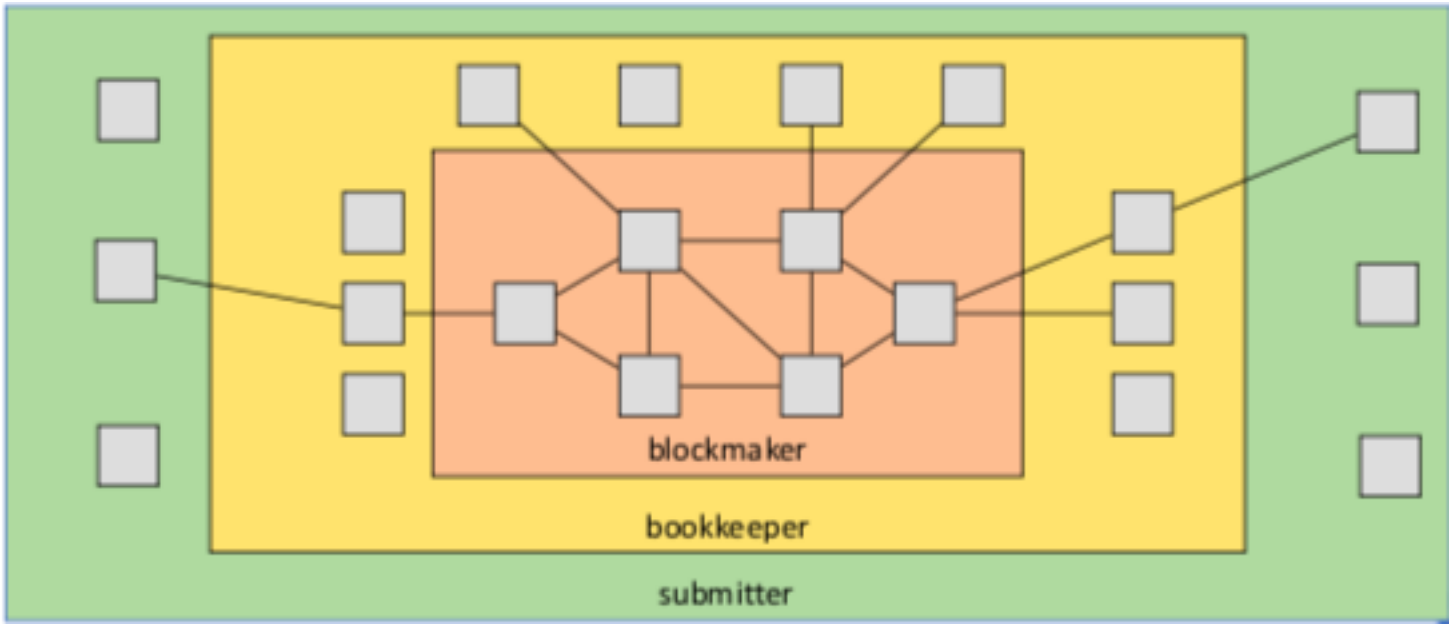
Hyperledger Fabric 架构



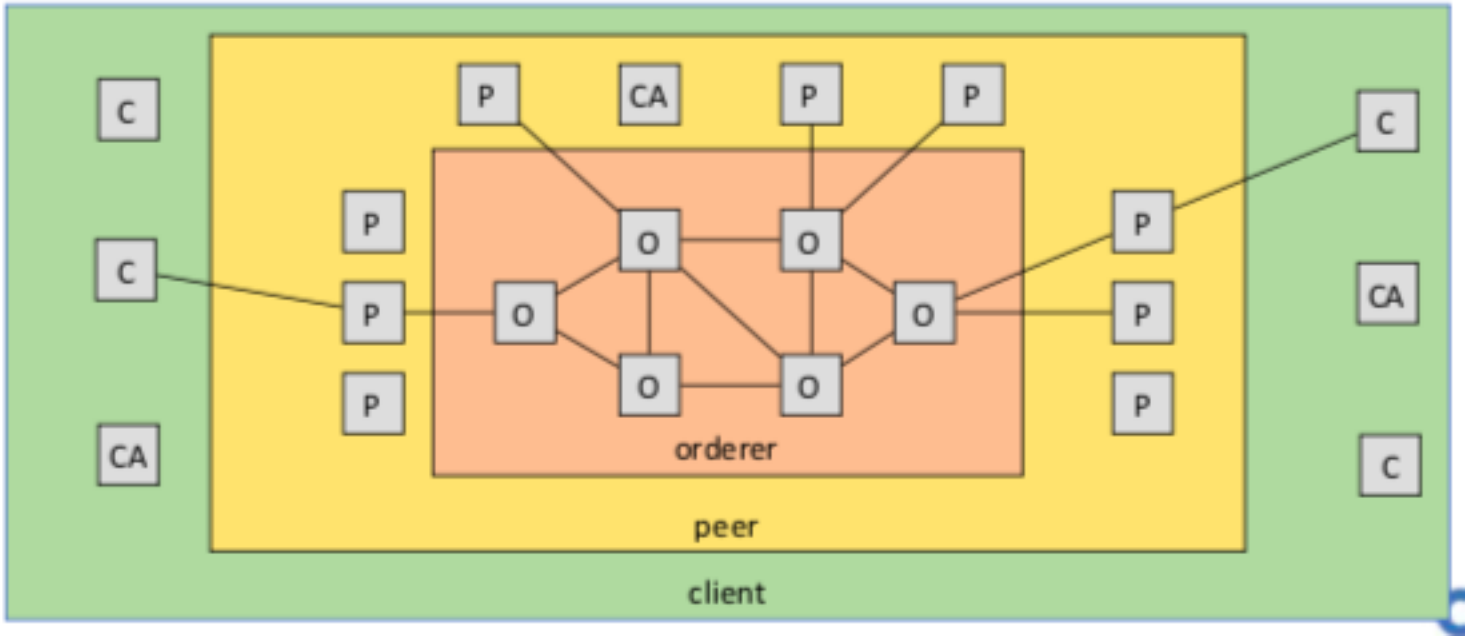
Contents

Part 2.1

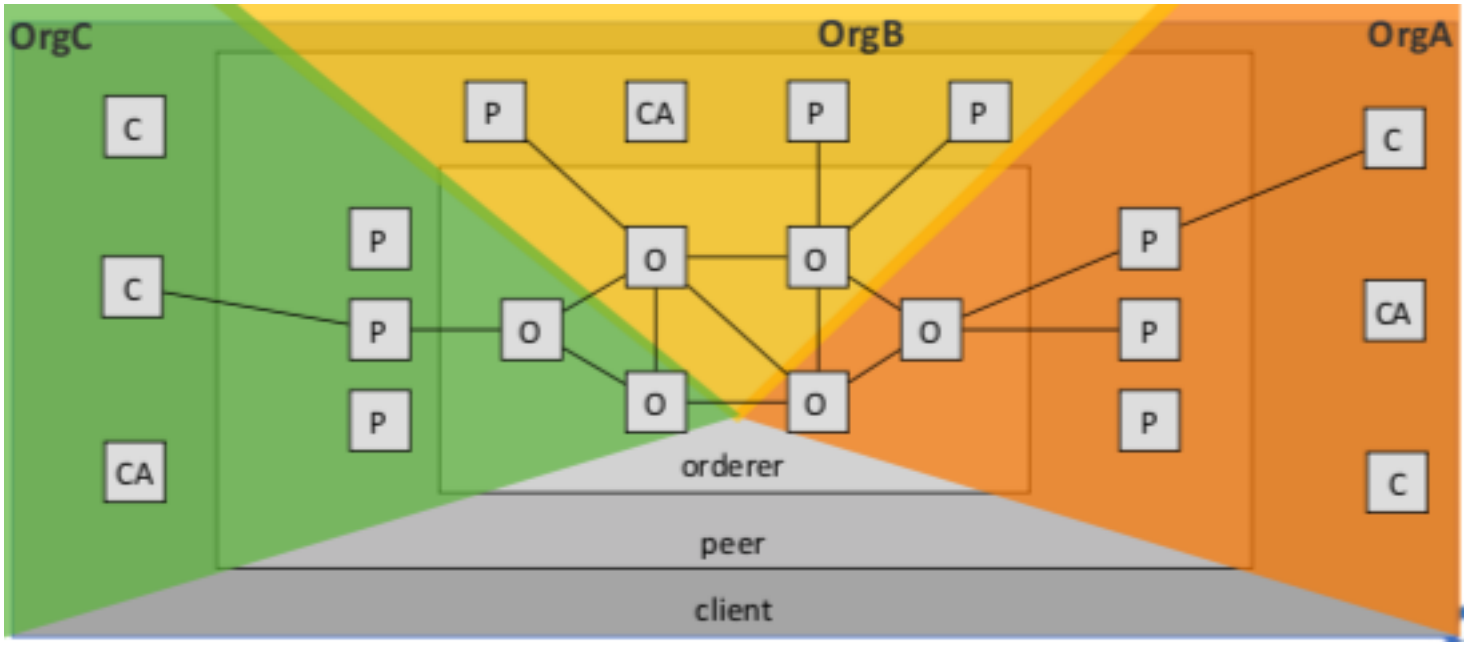
公有链和联盟链结构中的三类不同的角色



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

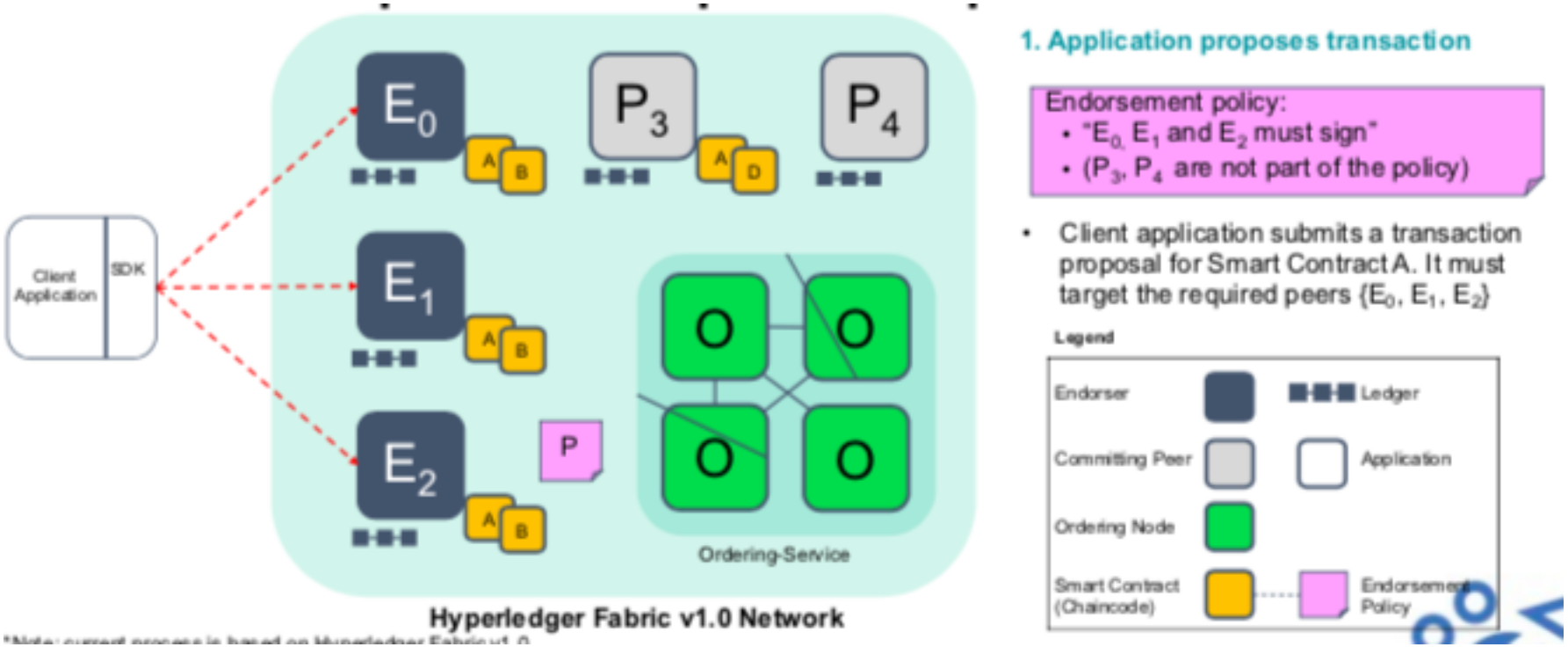
- **Committing Peer （提交节点）**
- ✓ Maintains ledger and state
- ✓ Commits transactions
- ✓ May hold smart contract (chaincode)

- **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

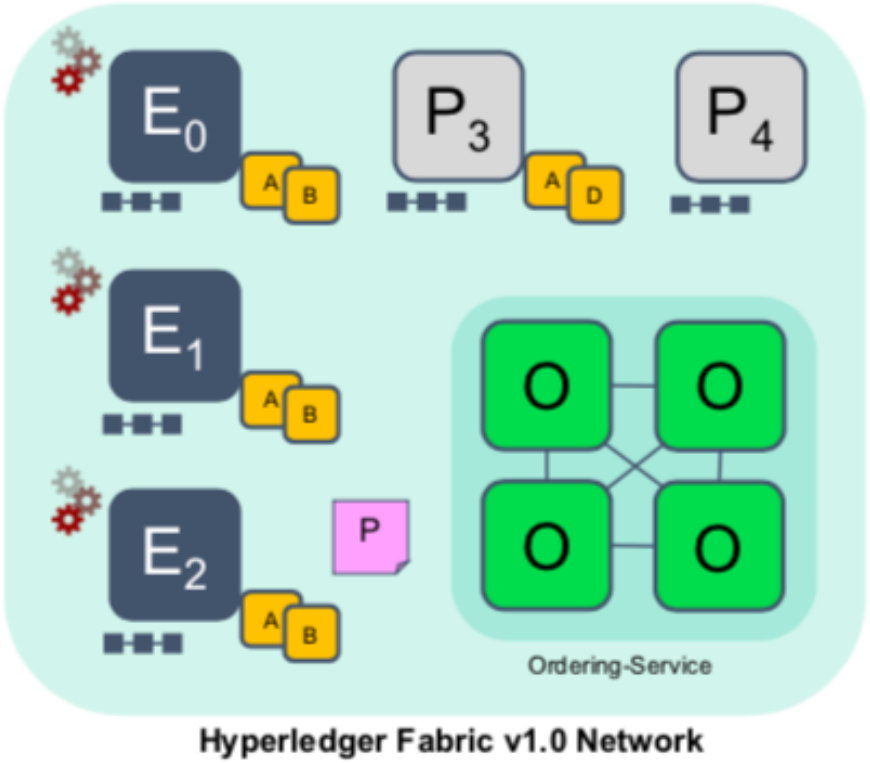


交易提案（Proposal）包含如下5个要素：

（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）

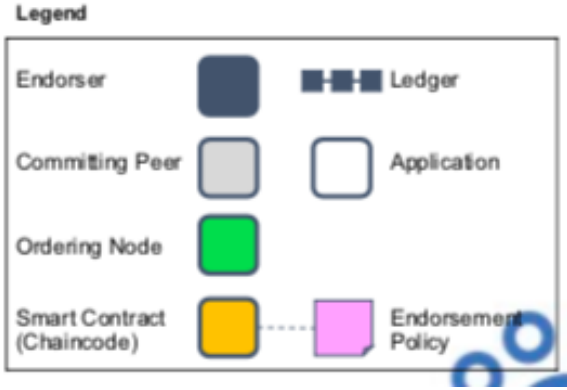
Transaction process: Step 2/7- Execute



Hyperledger Fabric v1.0 Network

2. Endorsers Execute Proposals

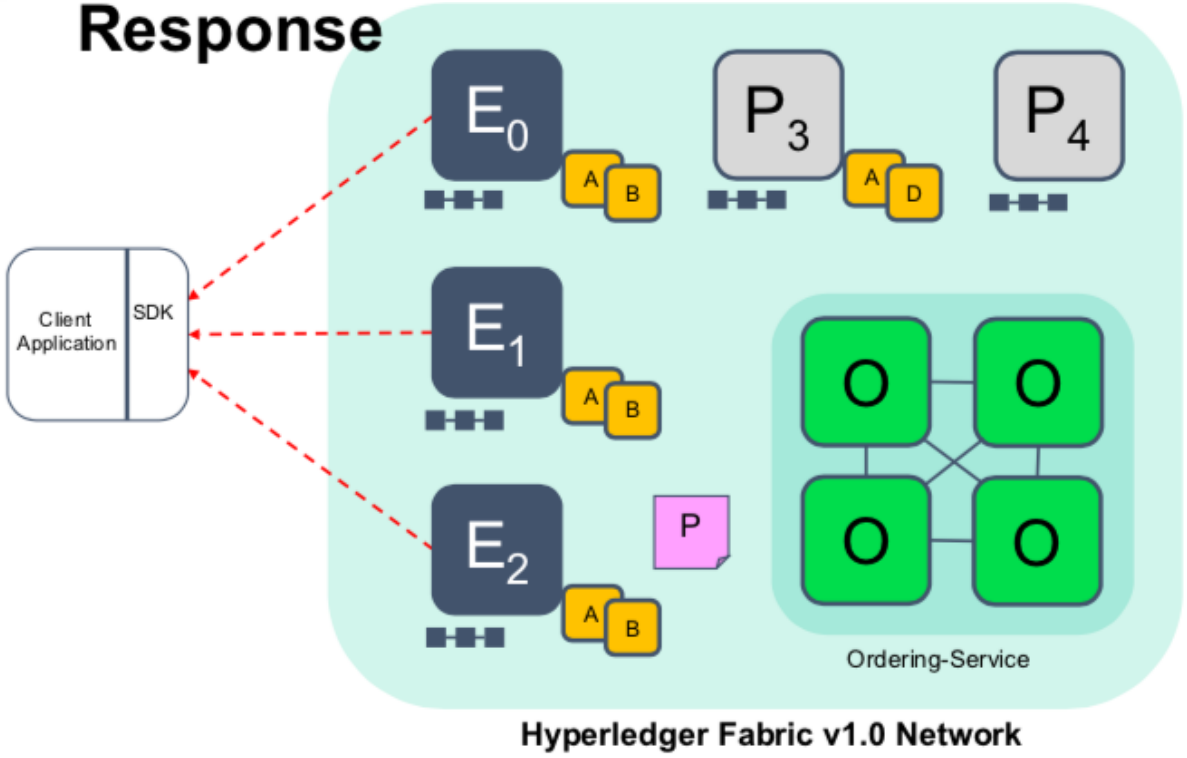
- E_0, E_1 & E_2 will each execute the proposed transaction. None of these executions will update the ledger
- Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric
- Transactions can be signed & encrypted



读集：Key=balance，value=100，version=1

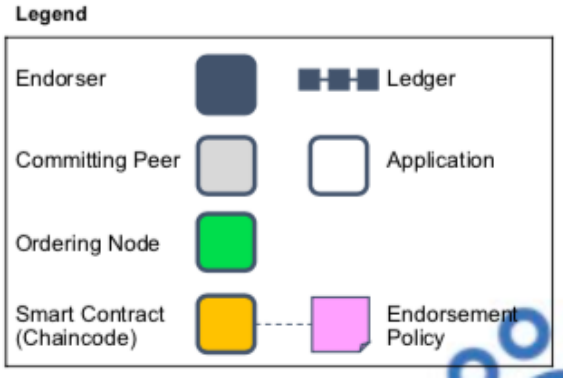
写集：Key=balance，value=90，version=2

Transaction process: Step 3/7- Proposal response

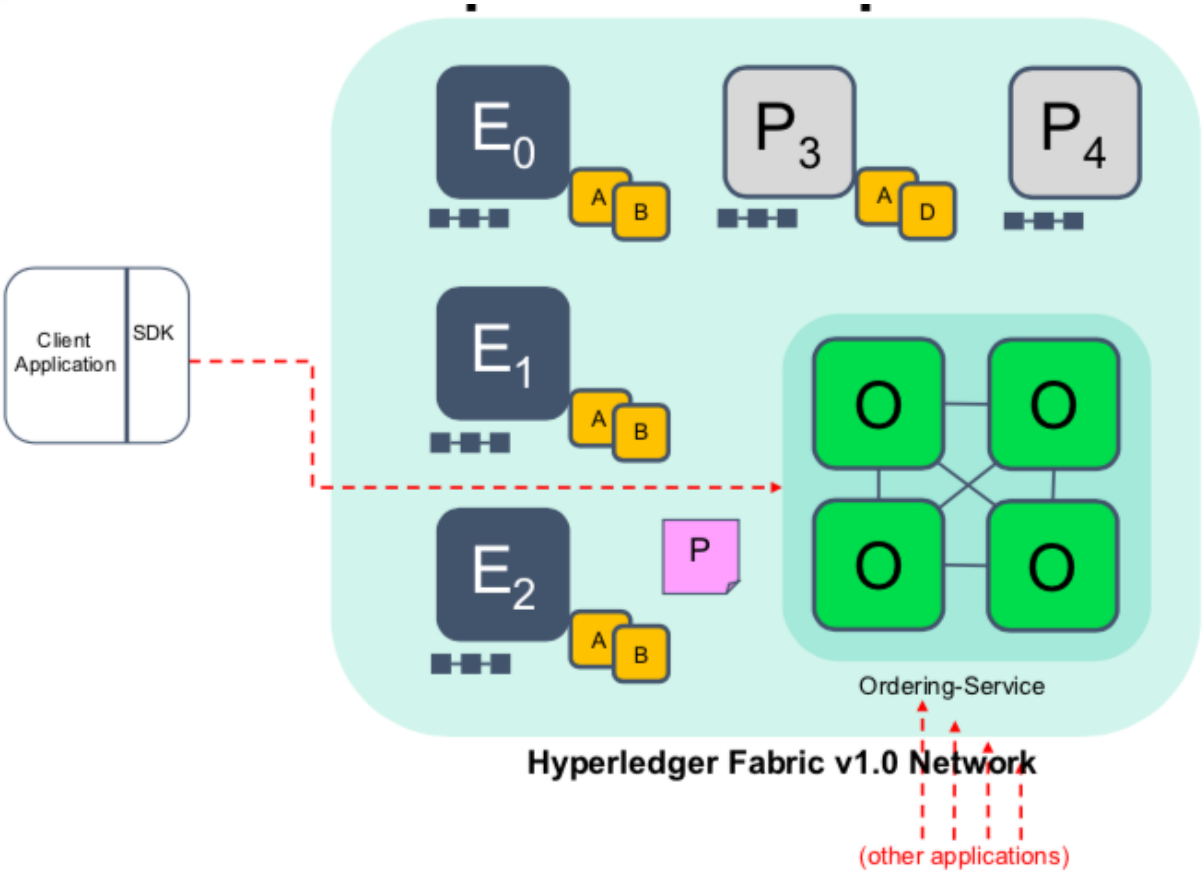


3. Application receives responses

- 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)

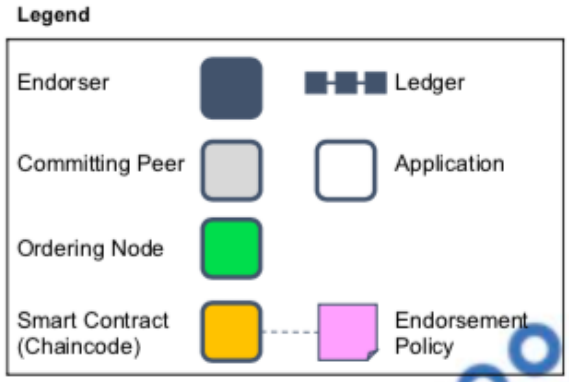


Transaction process: Step 4/7- Order Transaction

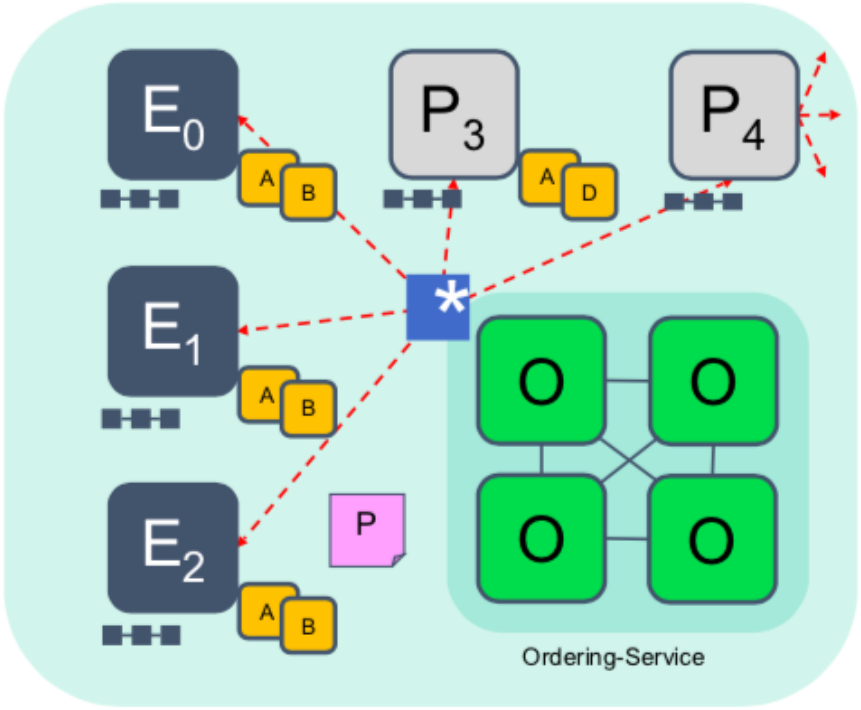


4. Responses submitted for ordering

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



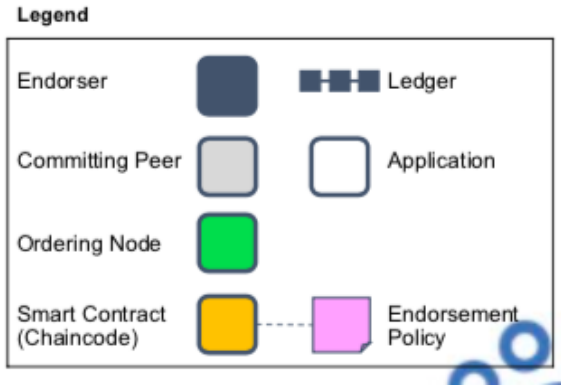
Transaction process: Step 5/7- Deliver



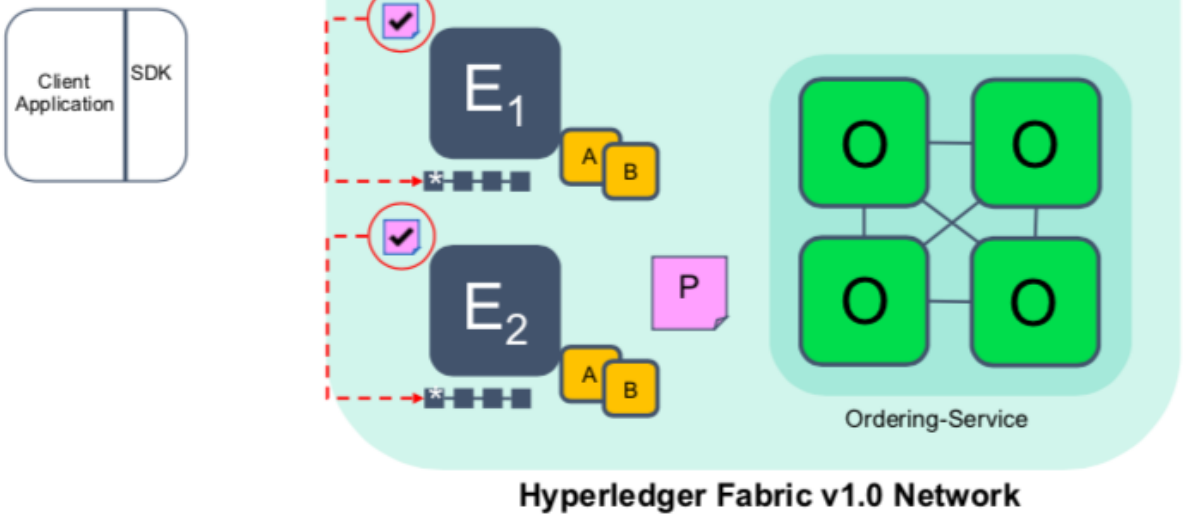
Hyperledger Fabric v1.0 Network

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)



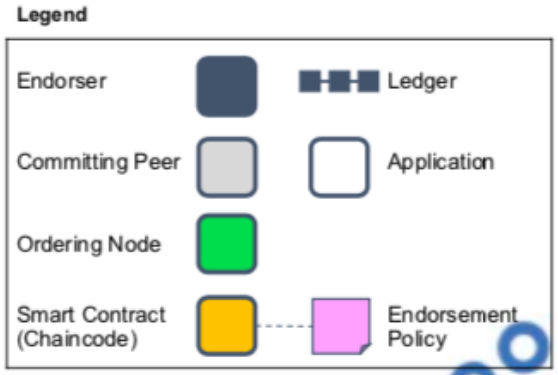
Transaction process: Step 6/7- Validate



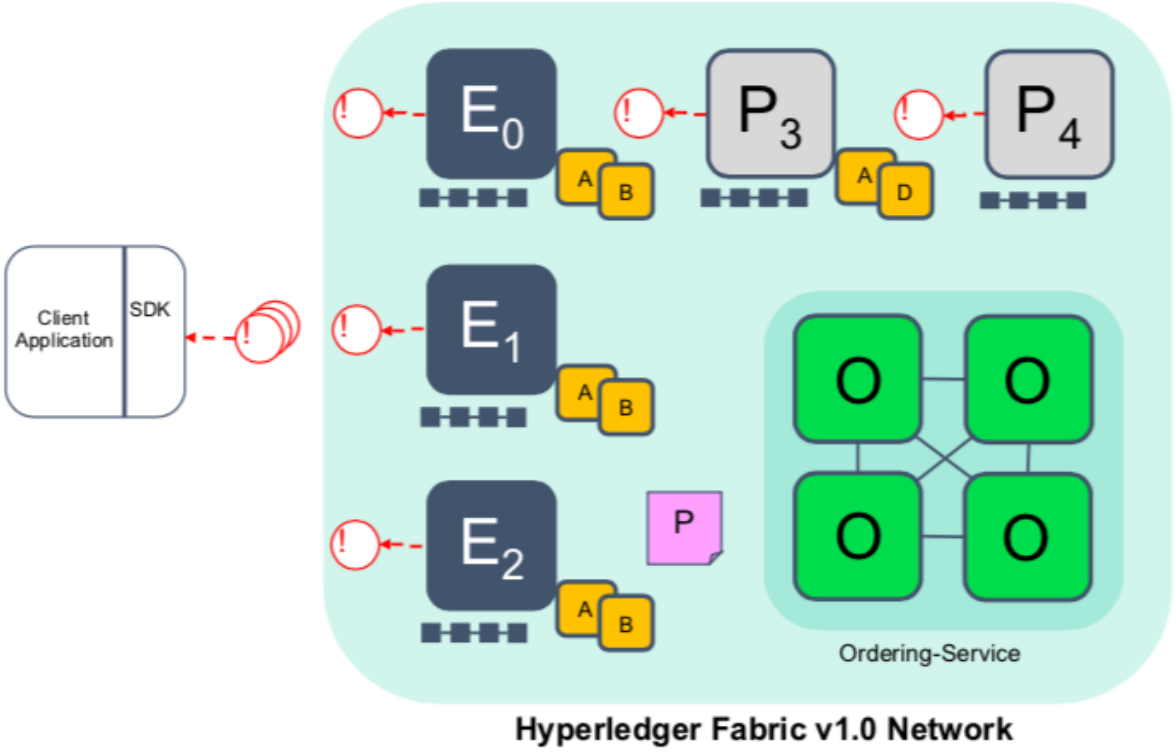
Hyperledger Fabric v1.0 Network

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



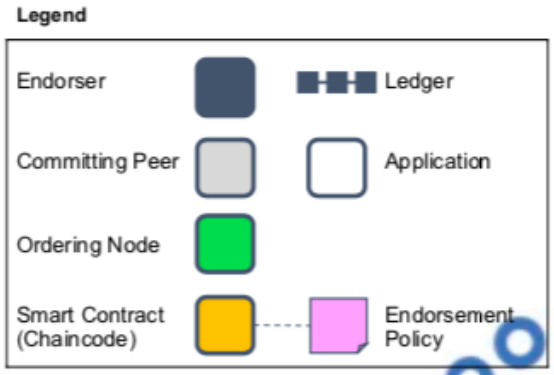
Transaction process: Step 7/7- Notify



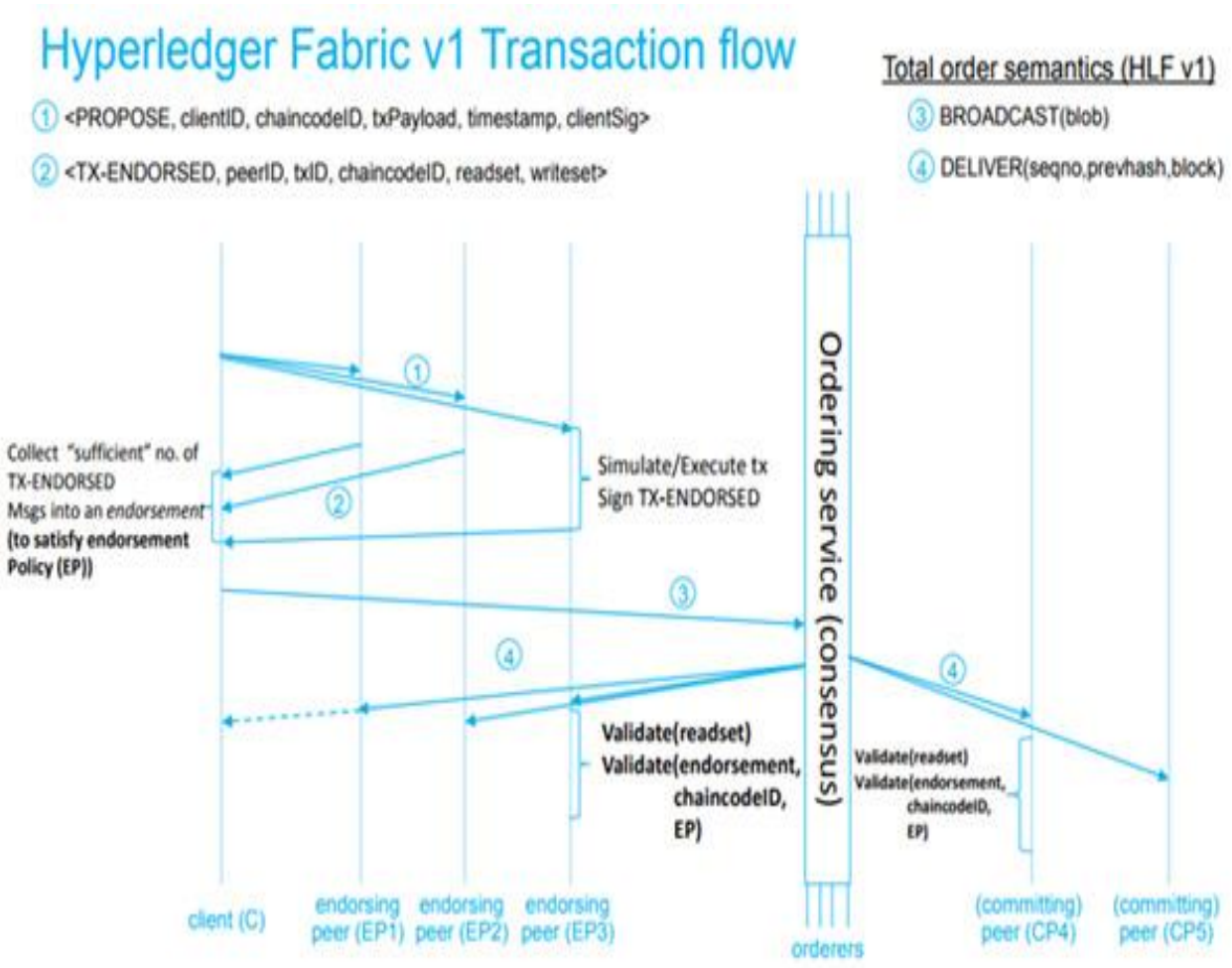
Hyperledger Fabric v1.0 Network

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。

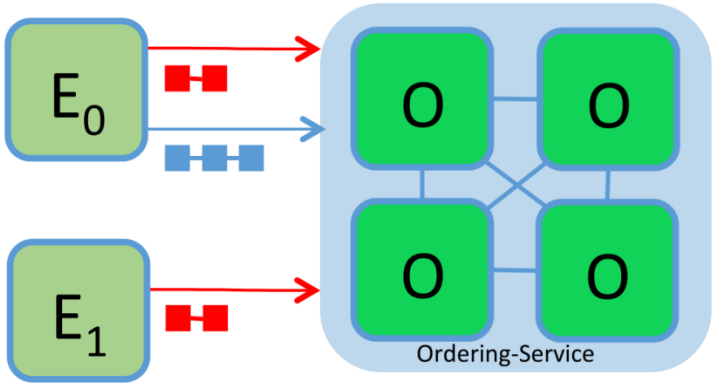
Fabric对双花问题的解决

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?

Channel

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



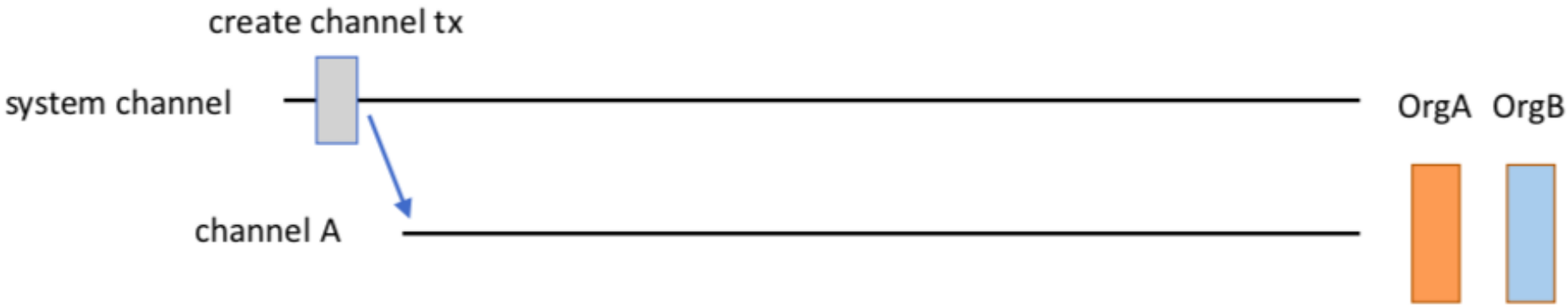
- Enables chaincode privacy
  - Chaincode deployed to certain nodes
- Messages partitioned into separate channels
  - Transactions stored depending on node and channel
- Nodes can connect to one or more channels

Fabric Channel Running method

system channel

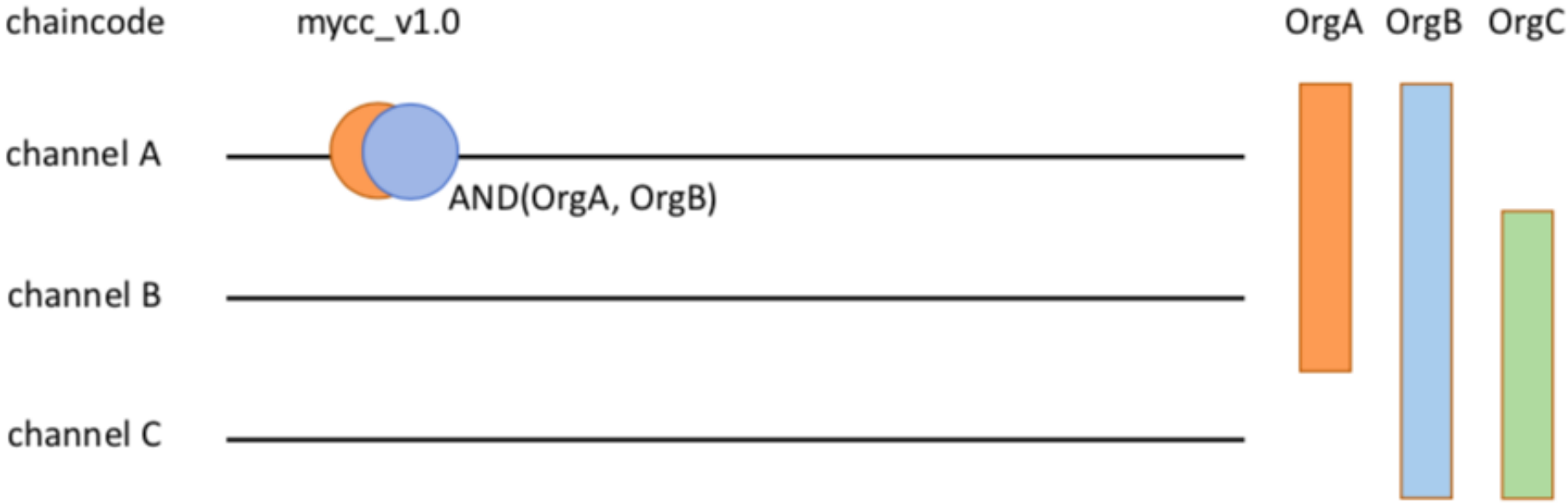
Fabric Channel Running method

Fabric Channel Running method

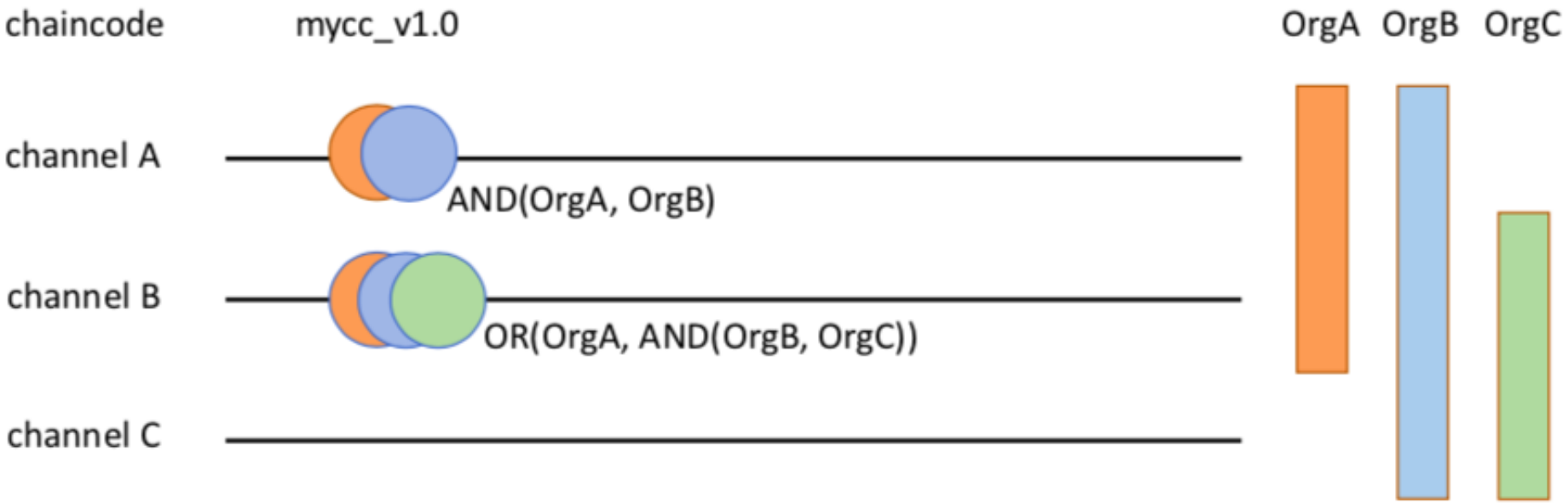


Fabric Channel Running method

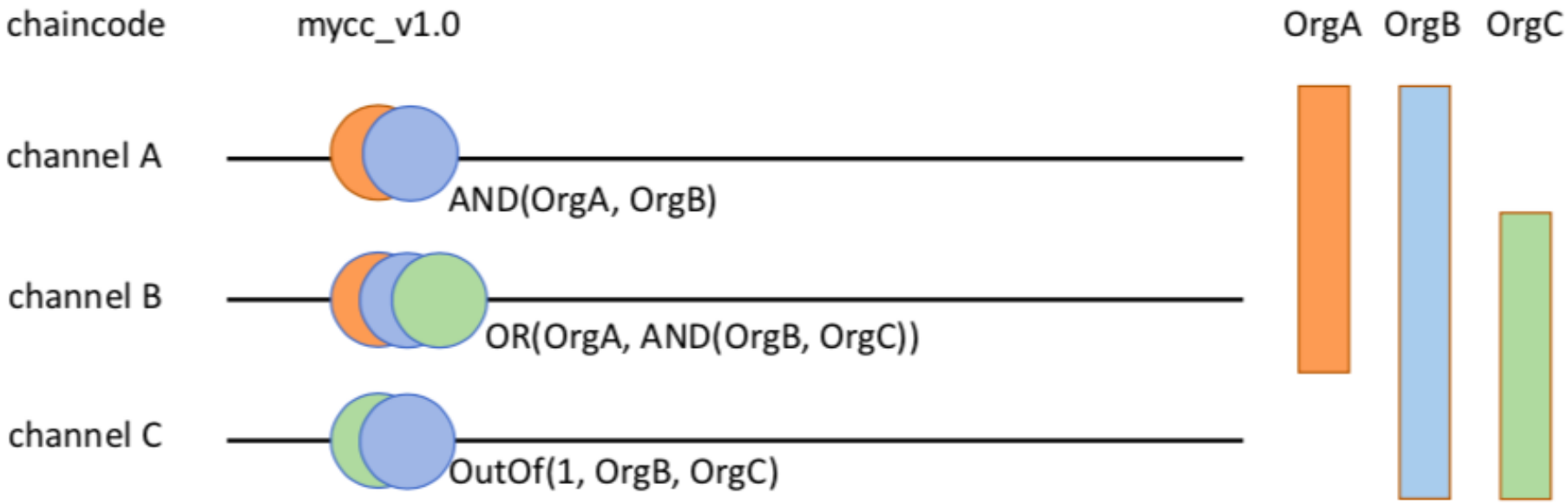
Fabric Channel Running method



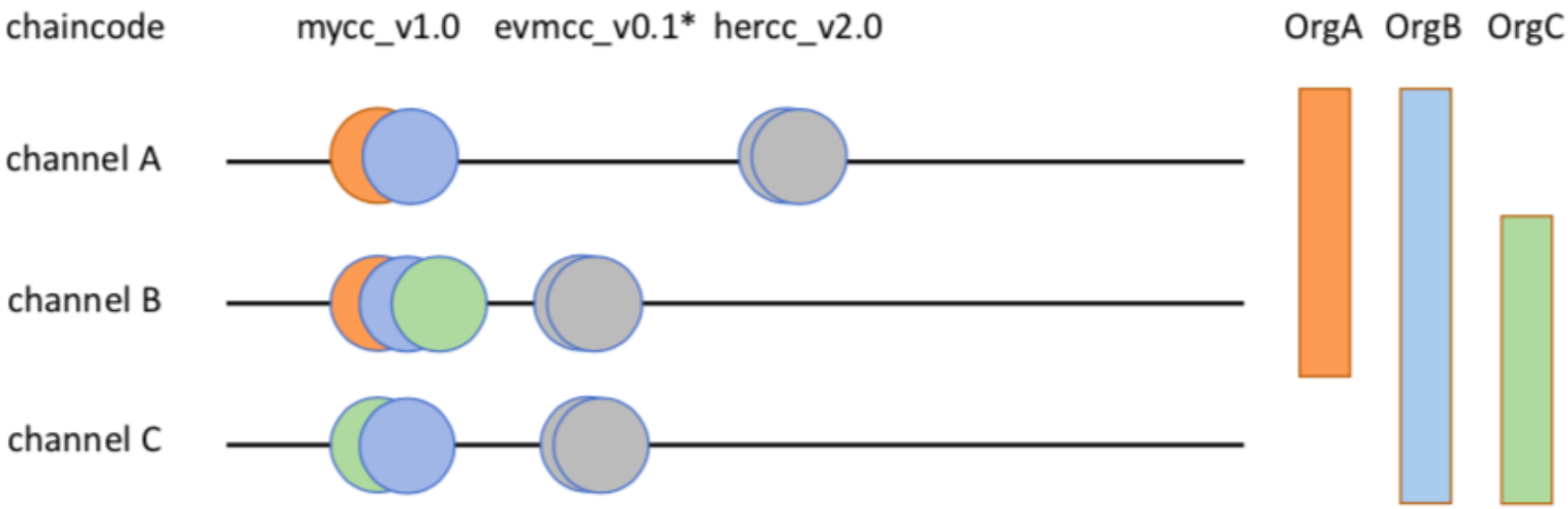
Fabric Channel Running method



Fabric Channel Running method



Fabric Channel Running method

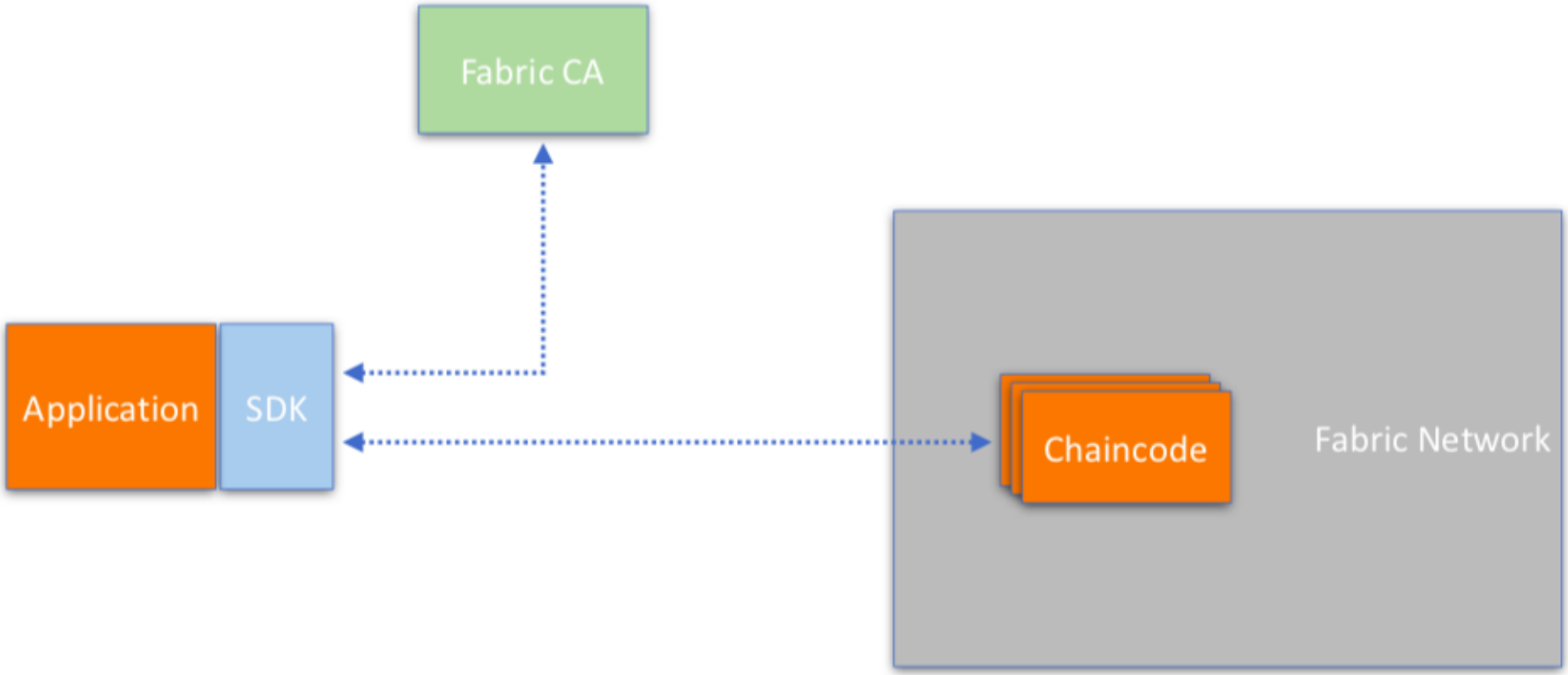


Coming soon

- 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

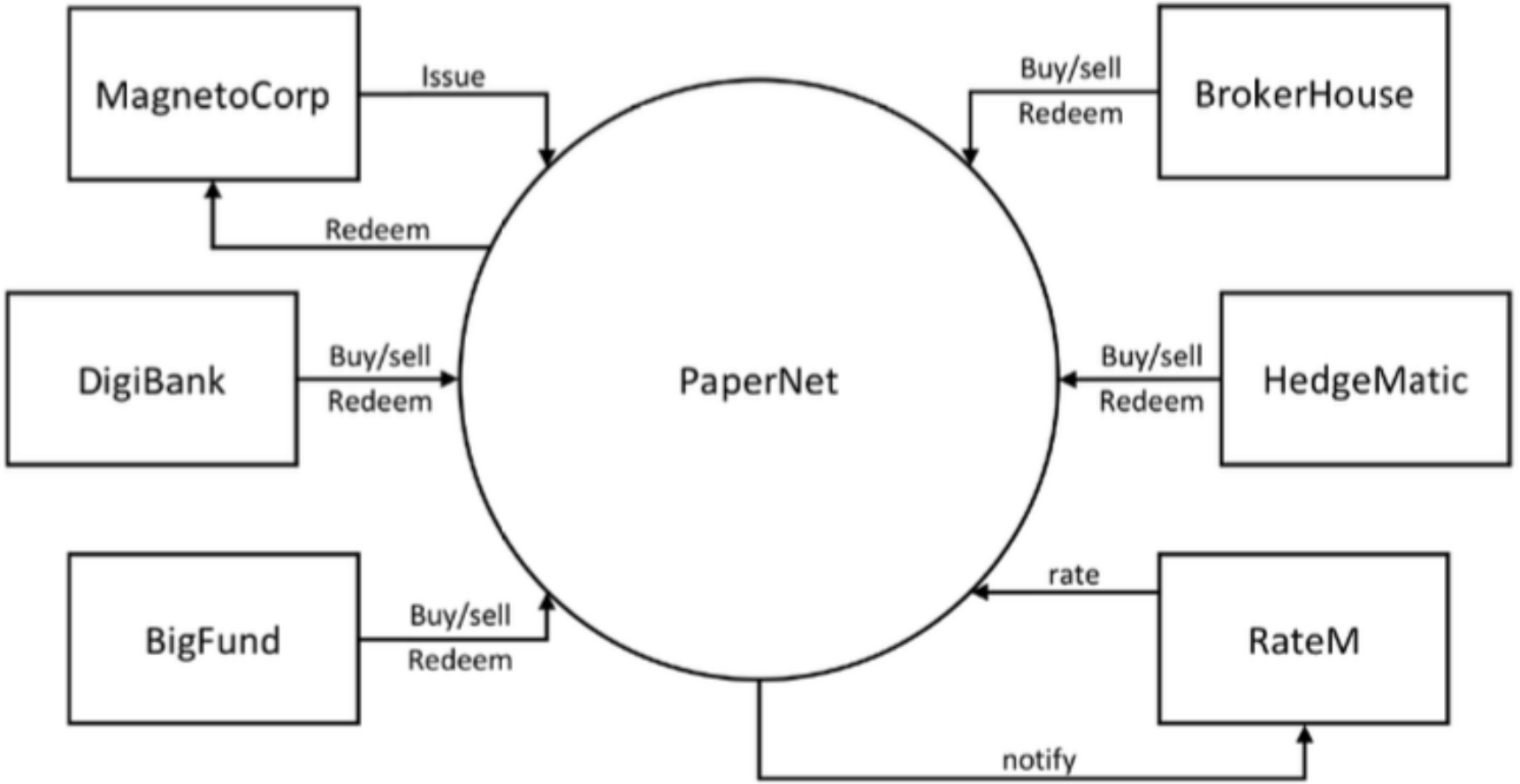


Develop Applications

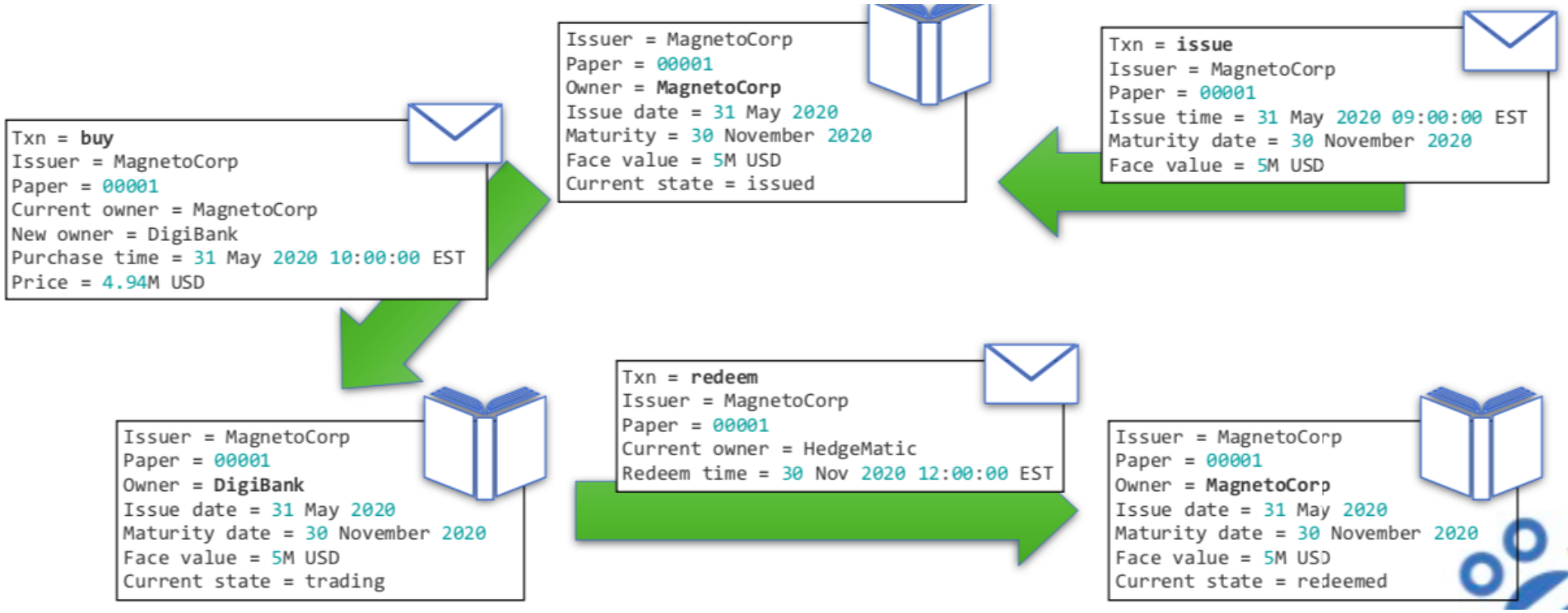

```
// ChaincodeStubInterface is used by deployable chaincode apps to access and
// modify their ledgers
type ChaincodeStubInterface interface {
	// GetArgs returns the arguments intended for the chaincode Init and Invoke
	// as an array of byte arrays.
	GetArgs() [][]byte

	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

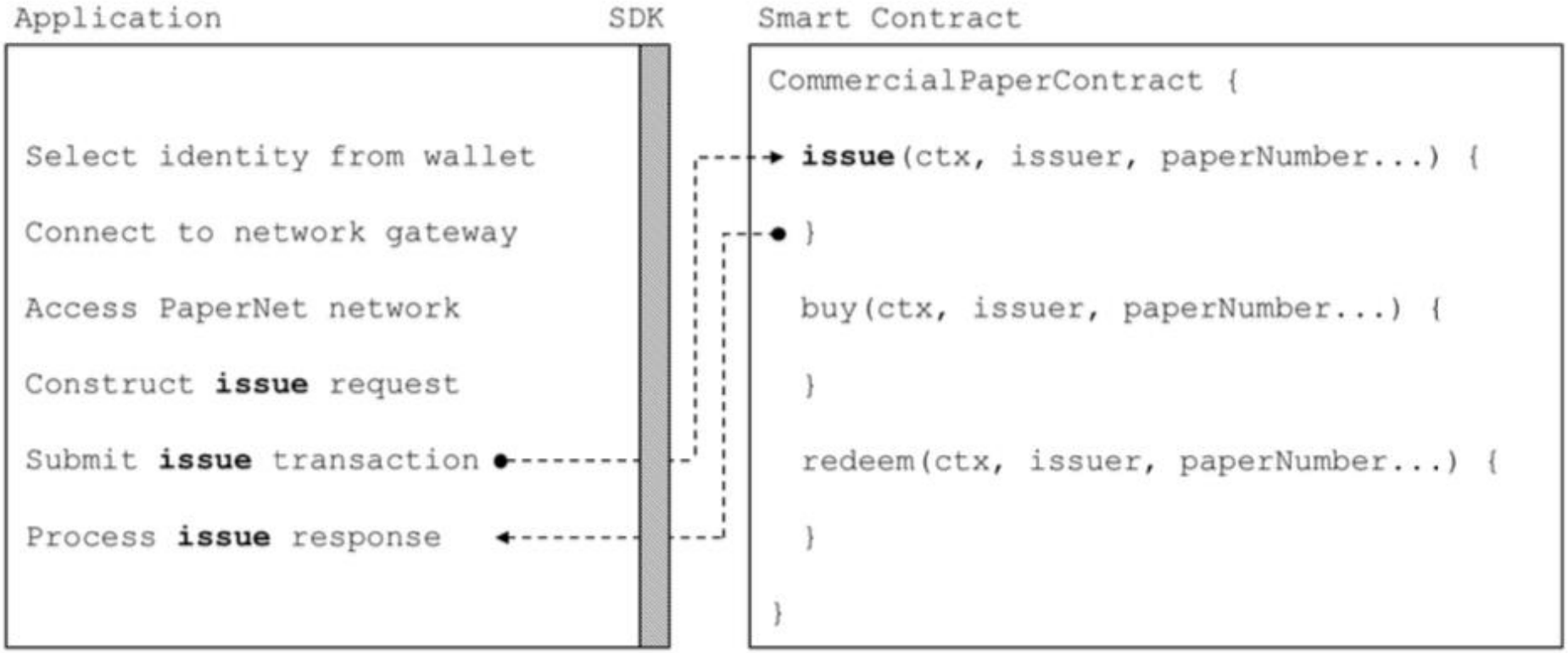
key: org.papaernet.paperMagnetoCorp00001

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



Develop Applications-last but not least

- Chaincode namespace
- Endorsement policies
- Connection Profile
- Idendities
- ...

Thanks!