IBM

Open by design™

IBM开源技术微讲堂

Hyperledger Fabric v1.4 LTS

课程安排

03/14 区块链赋能产业价值和商业模式
03/21 Hyperledger 项目概览 社区介绍
03/28 Fabric 1.4 LTS概述
04/04 Peer 解析
04/11 Orderer 解析
04/18 MSP 与 CA
04/25 应用开发指南
05/09 部署实践

欢迎关注微信公众号
“IBM开源技术”
获取更多资讯

公众号中发送“**报名**”,
即有机会参加Fabric线下训练营

IBM

Open by design™

Hyperledger Fabric v1.4 LTS概述

郭剑南
IBM软件工程师
guojiannan@cn.ibm.com
Rocket.Chat: @guoger

Outline

- Hyperledger Fabric架构概览
  - 交易处理流程
  - 数据隔离
  - 背书策略
- 应用开发
- 搭建测试网络

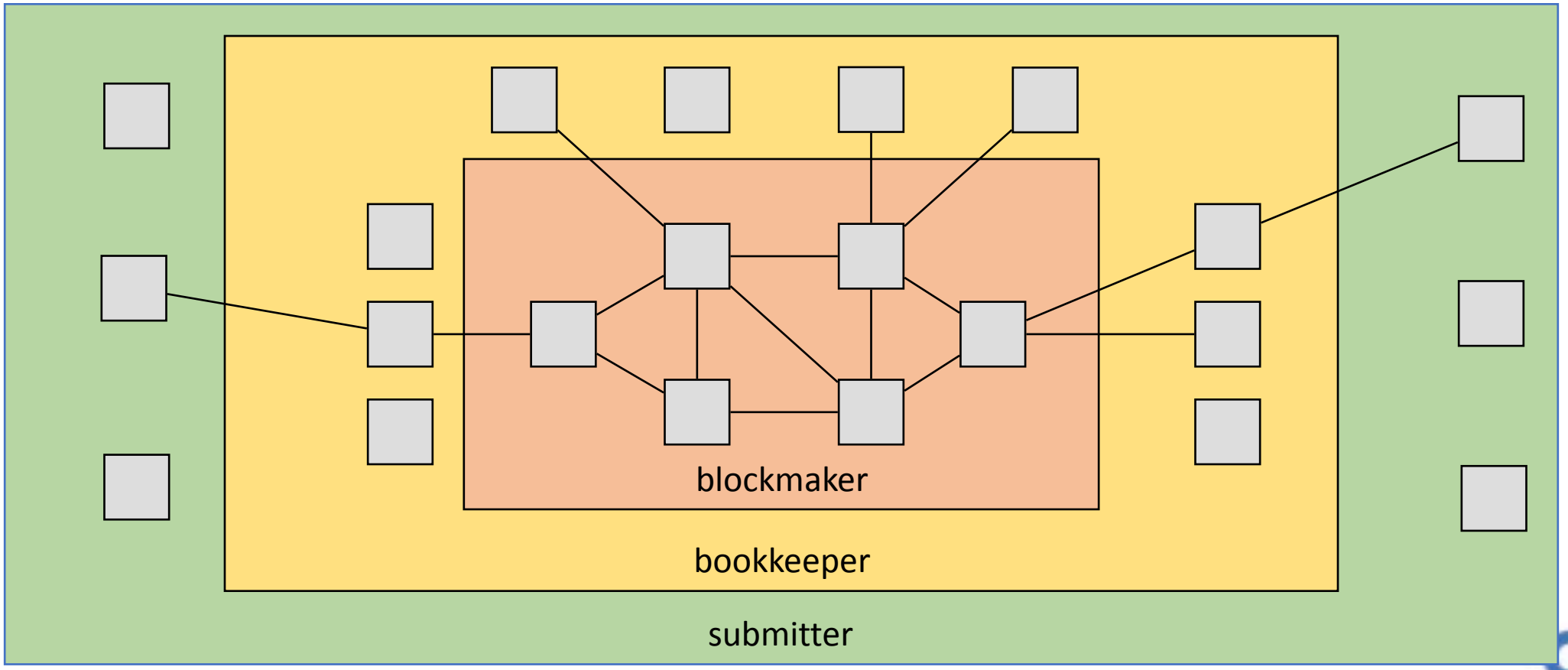
Permissioned ~~vs~~ or Permissionless

Blockchain Technologies For Business

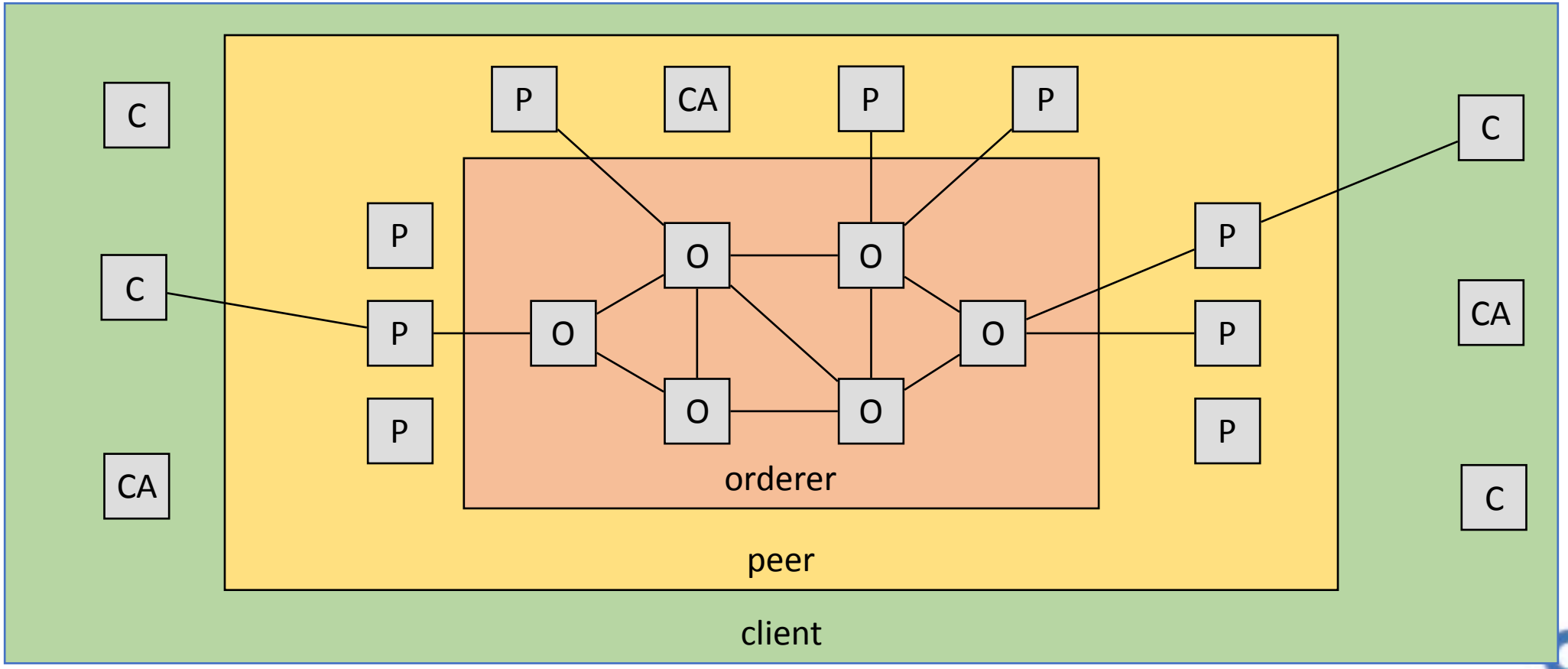
Permissioned or Permissionless

- Who can run a node?
- Who can submit transactions?
- Who can see your data?
- Who are you?
- …

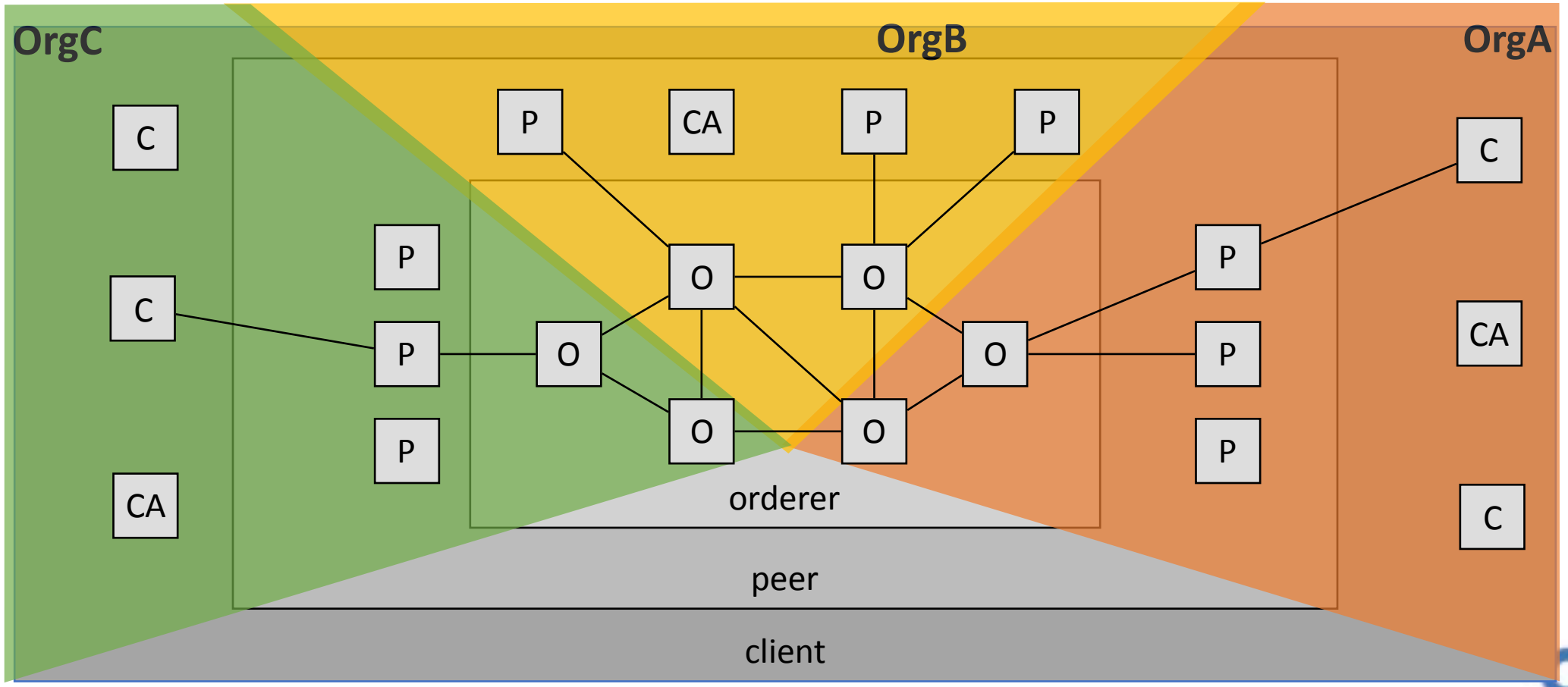
Permissioned or Permissionless



Permissioned or Permissionless



Permissioned or Permissionless



Characteristics

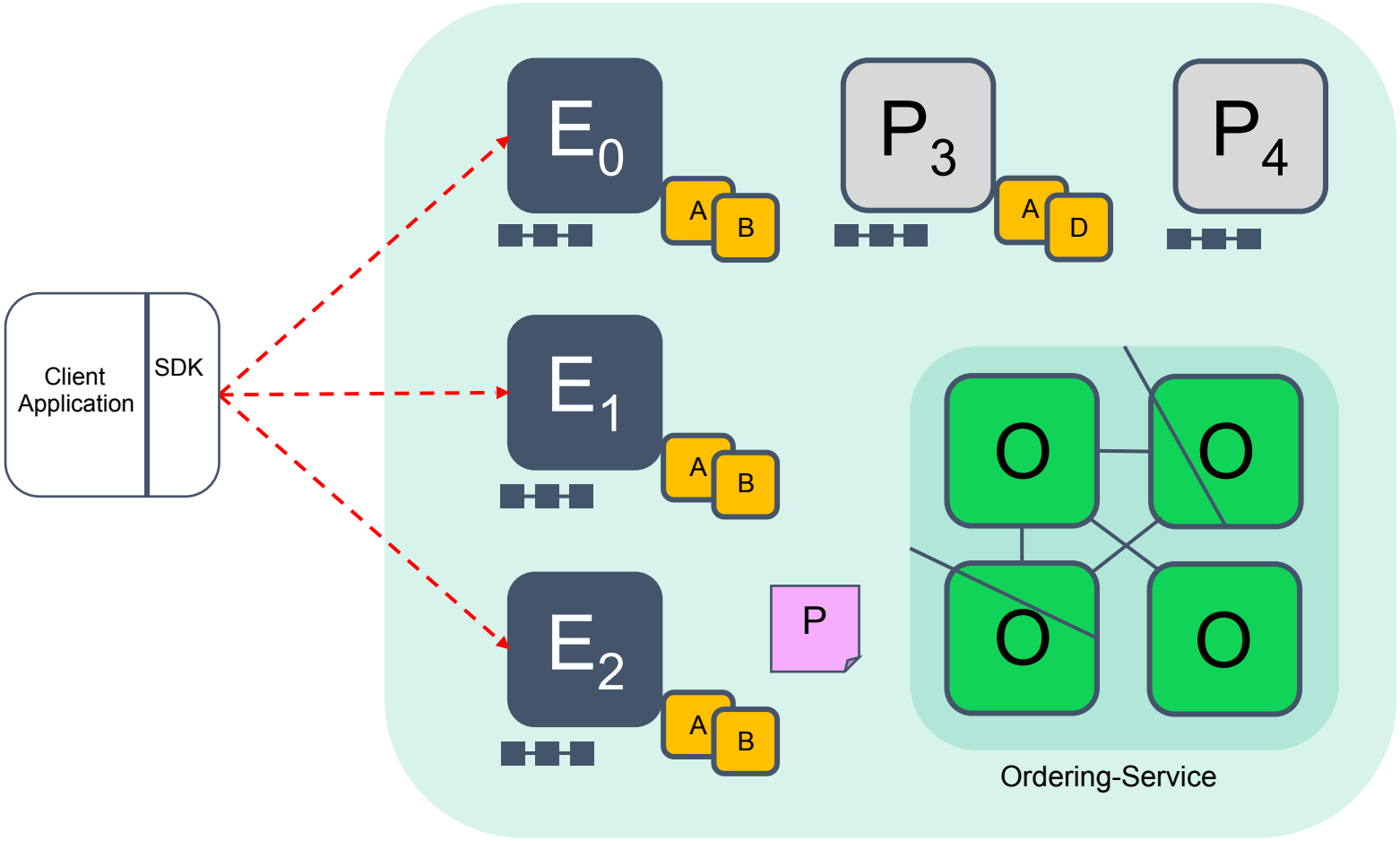
- Permissioned
- Highly modular
- Smart contracts in general purpose languages
- Pluggable consensus
- Privacy
- No “mining” or native crypto-currency required for consensus
- Execute-order-validate vs order-execute

Transaction Flow

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

- **Committing Peer**
  - Maintains ledger and state
  - Commits transactions
  - May hold smart contract (chaincode)
- **Endorsing Peer**
  - 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
- **Ordering Node**
  - 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
  - Does not hold smart contract
  - Does not hold ledger

Transaction process: Step 1/7 – Propose*



Hyperledger Fabric v1.0 Network

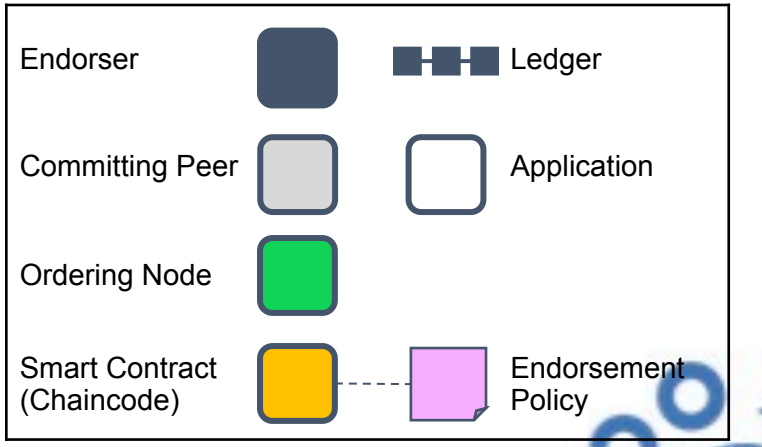
*Note: current process is based on Hyperledger Fabric v1.0.
Process and model could evolve based on the future development of Hyperledger Fabric.

1. Application proposes transaction

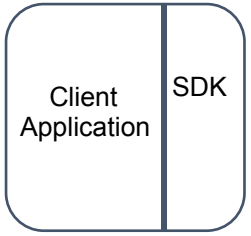
Endorsement policy:
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

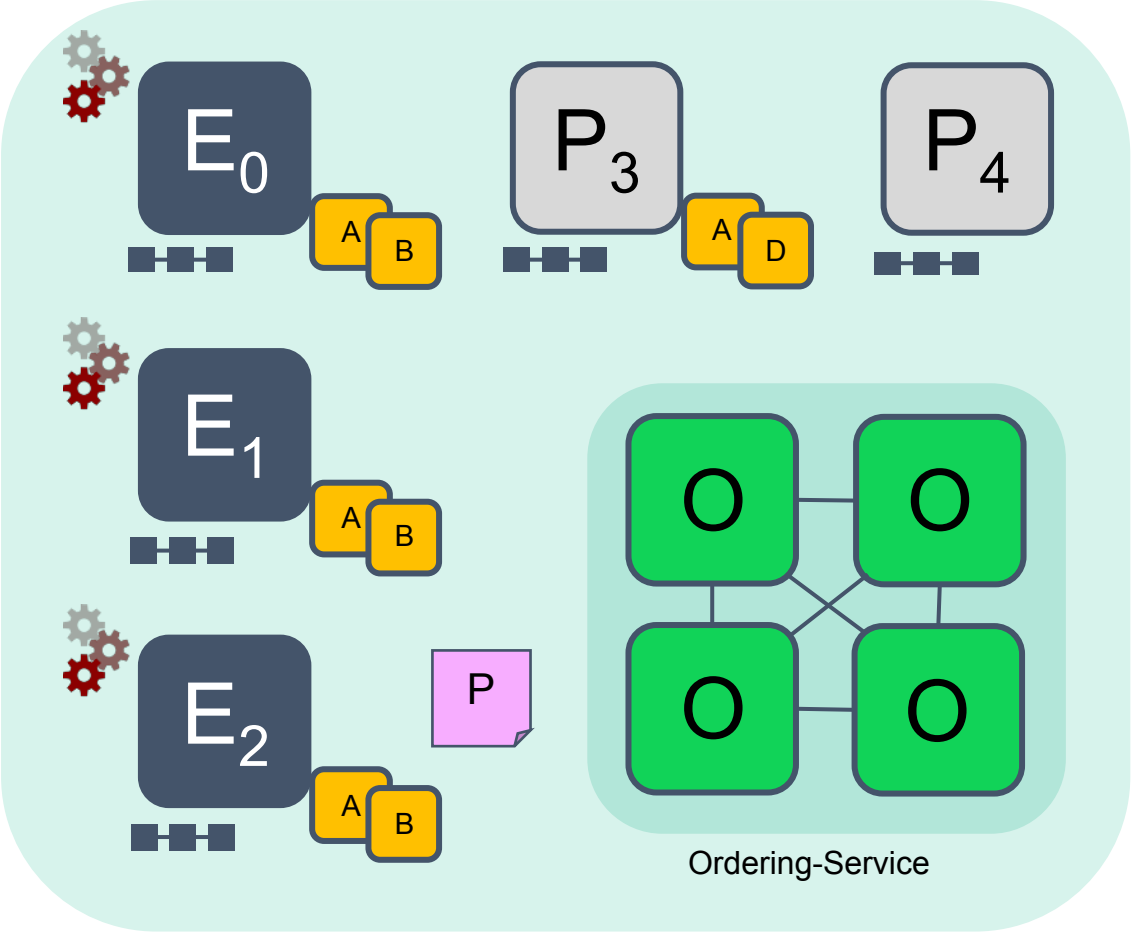
- Client application submits a transaction proposal for Smart Contract A. It must target the required peers {E_0, E_1, E_2}

Legend



Transaction process: Step 2/7 – Execute



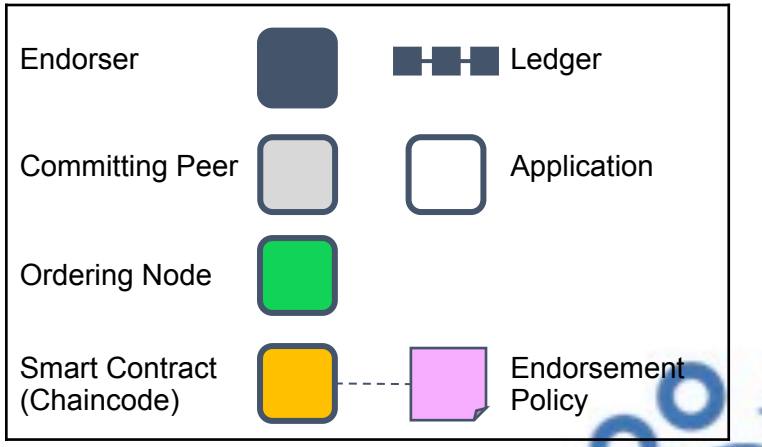


Hyperledger Fabric v1.0 Network

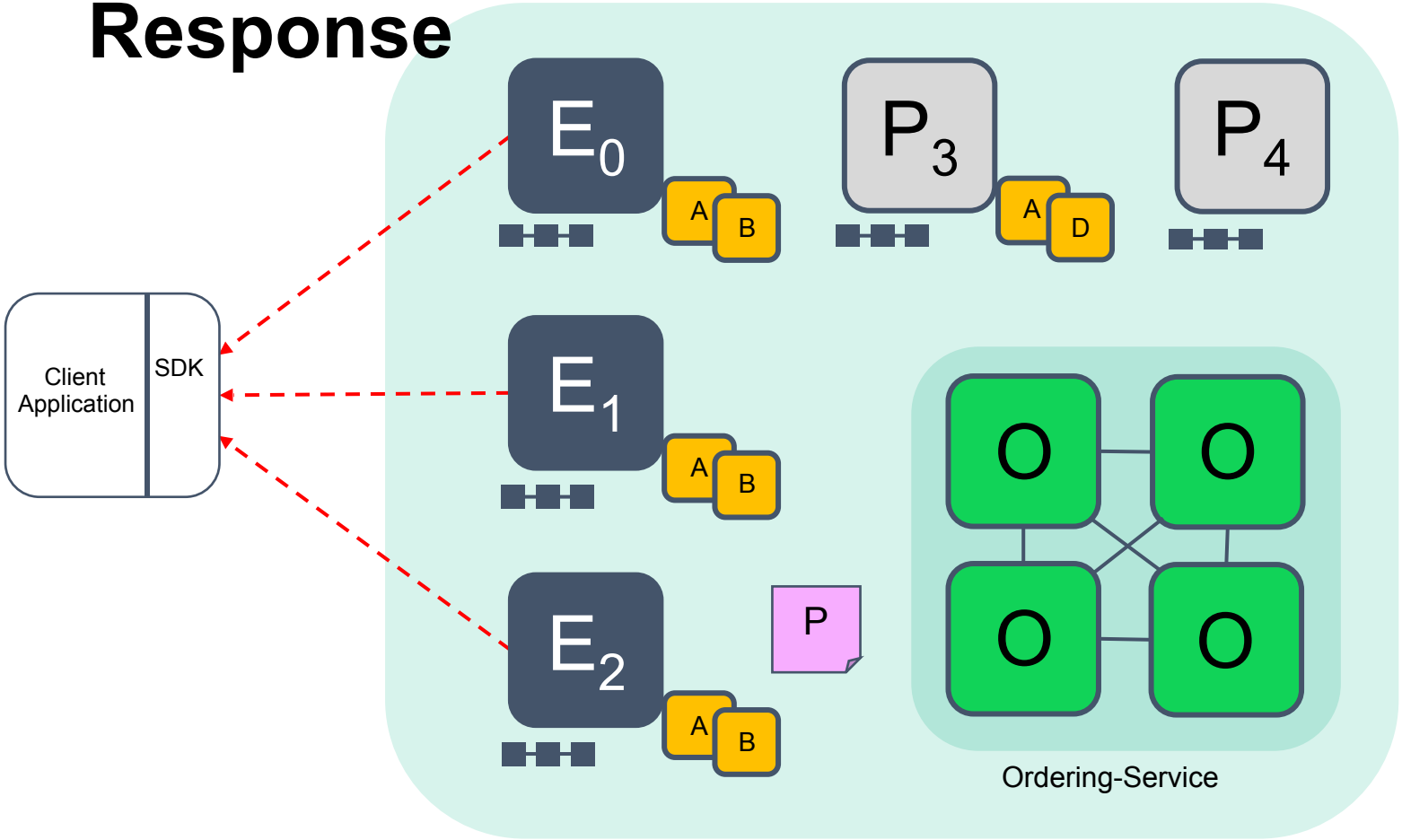
2. Endorsers Execute Proposals

- 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

Legend



Transaction process: Step 3/7 – Proposal Response

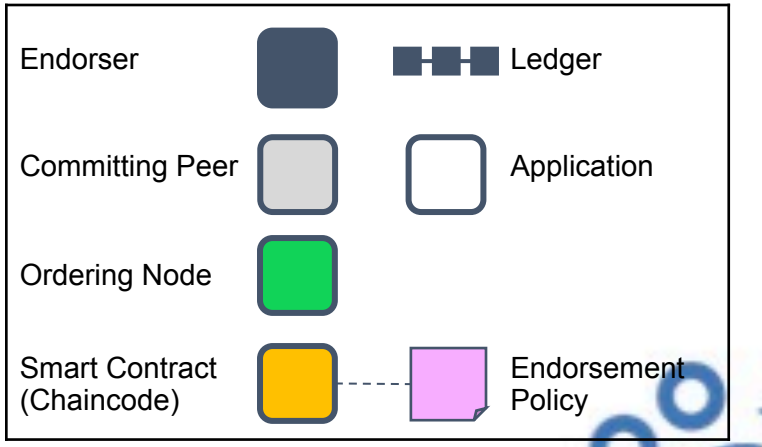


Hyperledger Fabric v1.0 Network

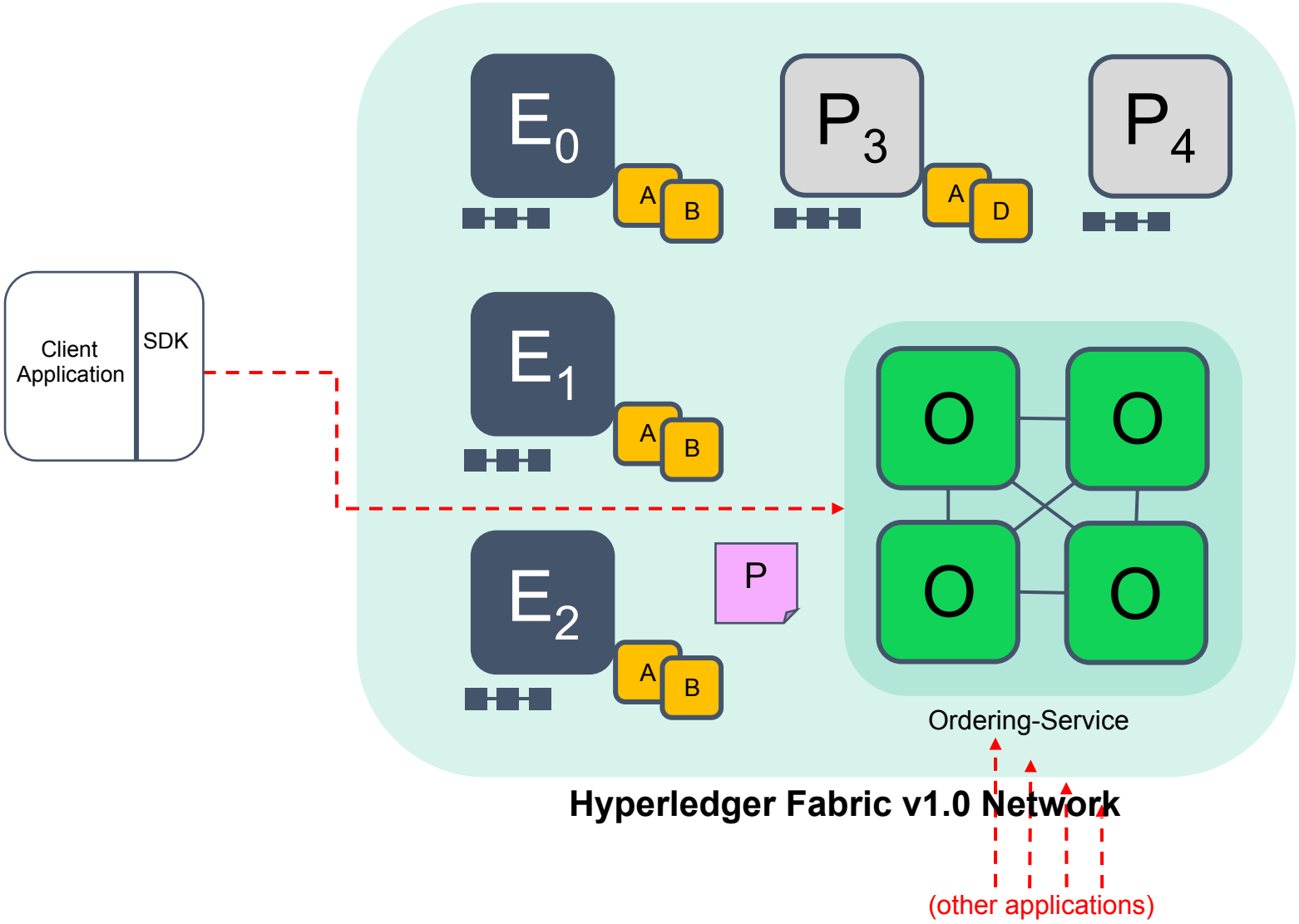
3. Application receives responses

- 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)

Legend

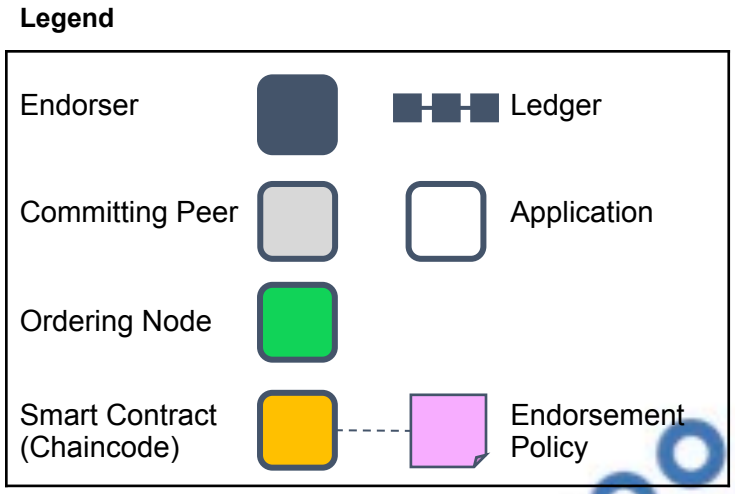


Transaction process: Step 4/7 – Order Transaction

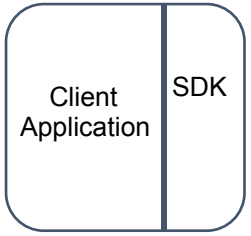


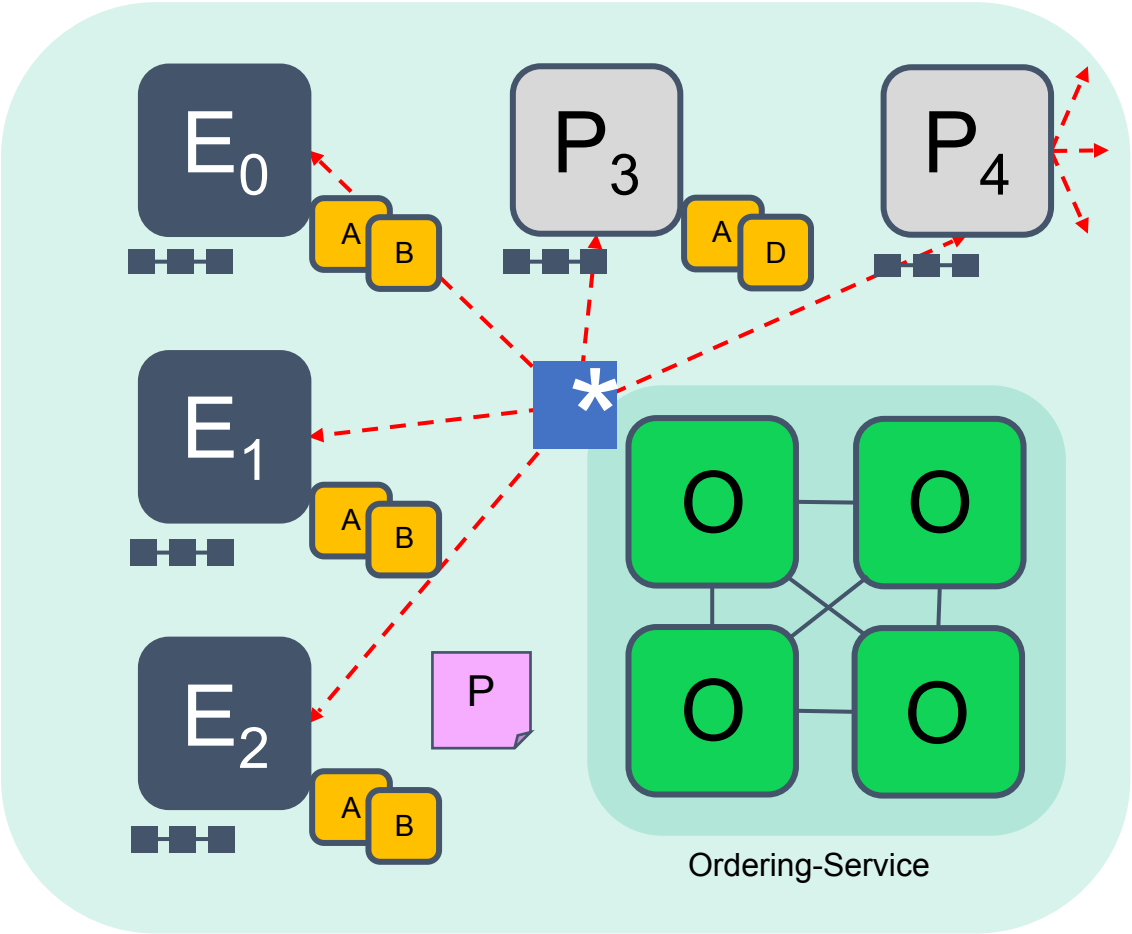
4. Responses submitted for ordering

- 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



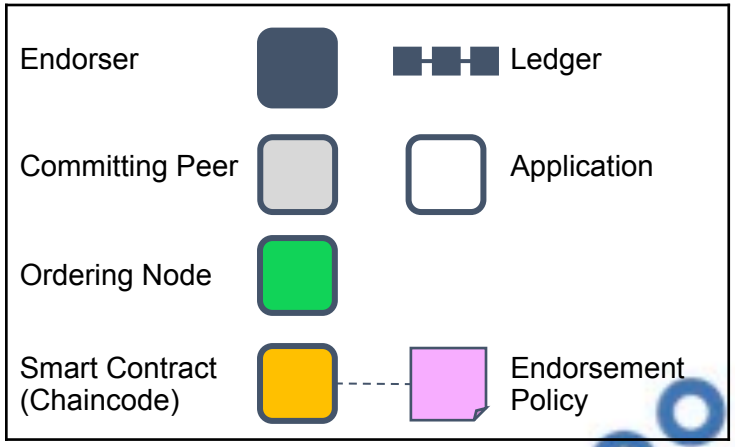


Hyperledger Fabric v1.0 Network

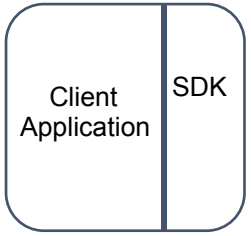
5. Orderer delivers to committing peers

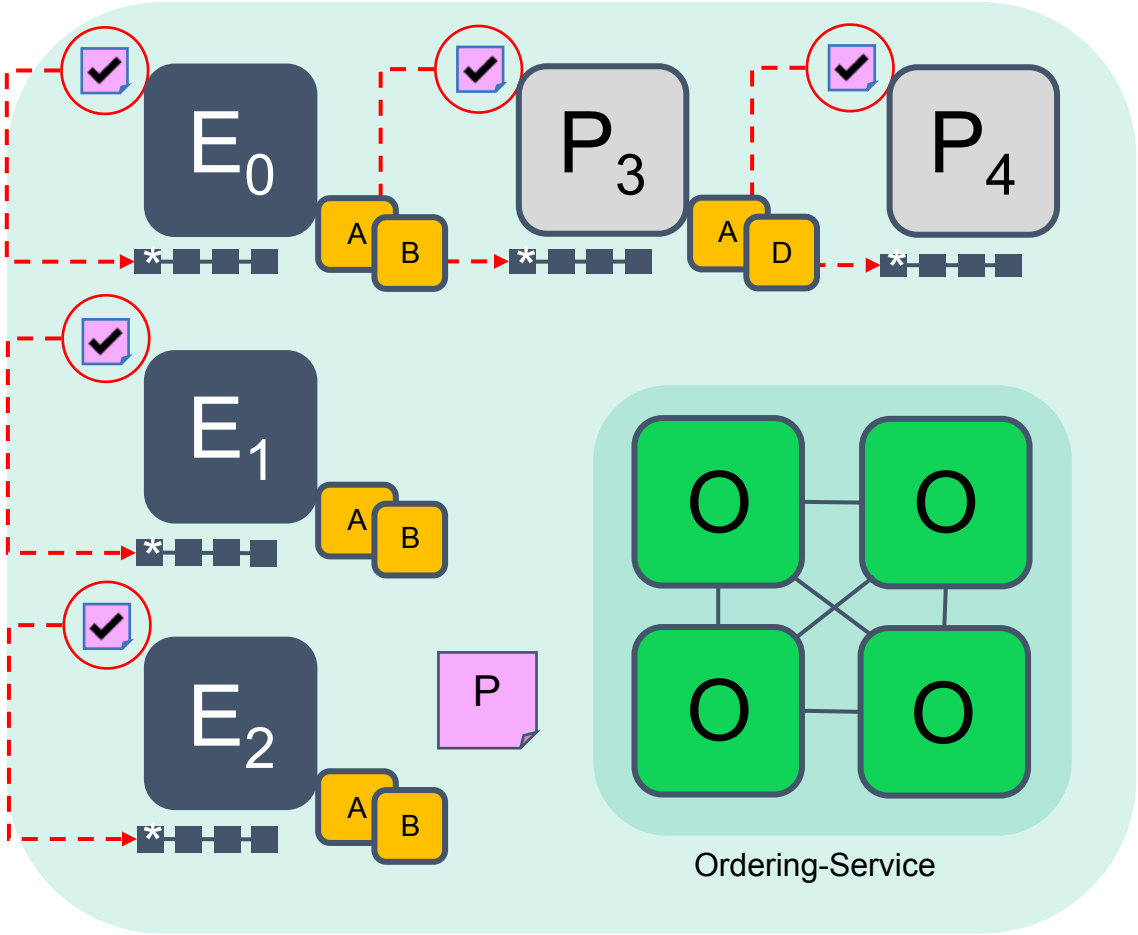
- Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown)
- Different ordering algorithms available:
  - Solo (Single node, development)
  - Kafka (Crash fault tolerance)

Legend



Transaction process: Step 6/7 – Validate



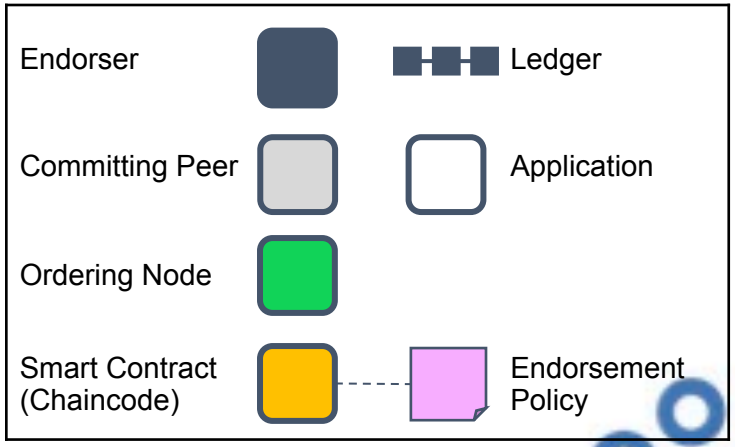


Hyperledger Fabric v1.0 Network

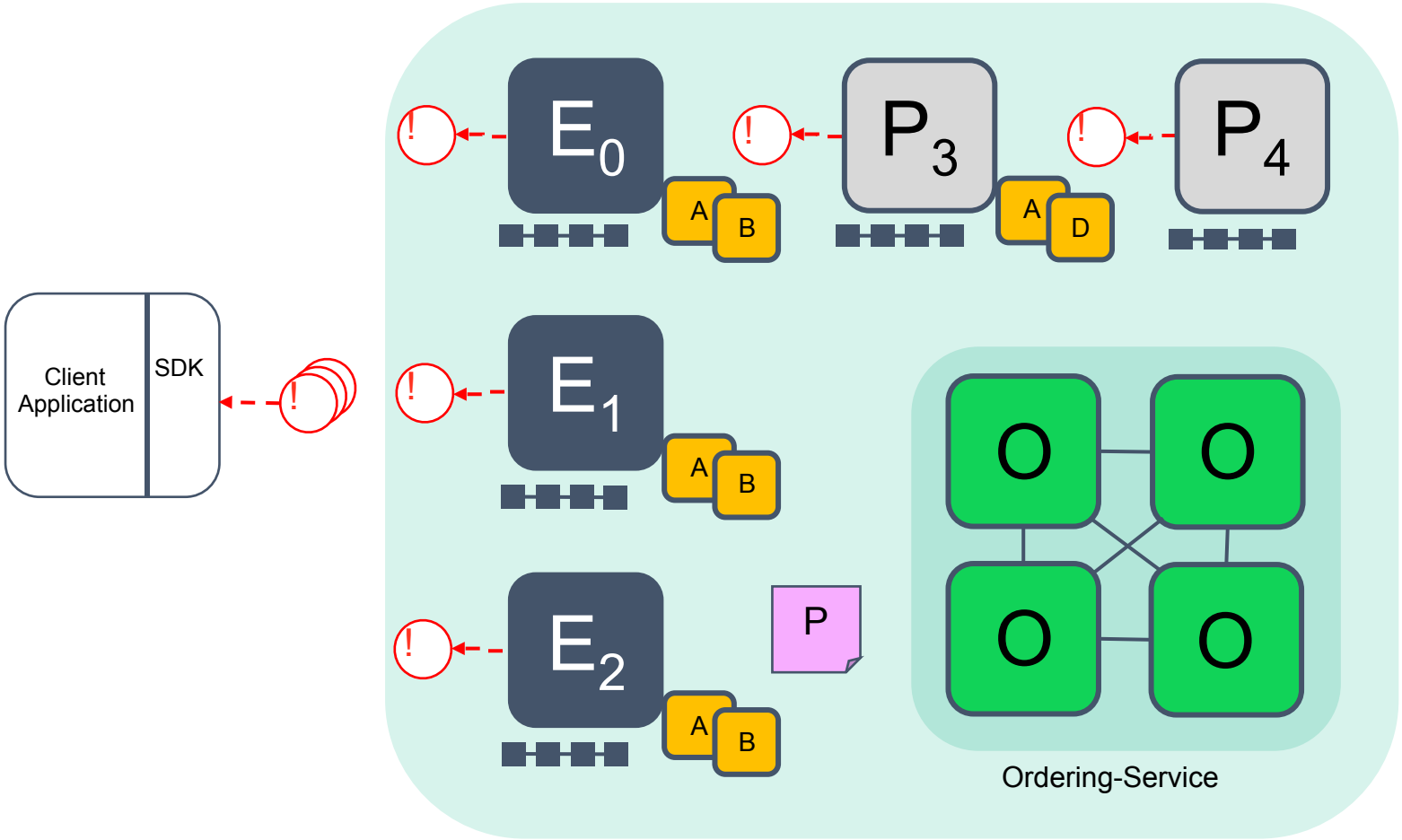
6. Committing peers validate transactions

- 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

Legend



Transaction process: Step 7/7 – Notify

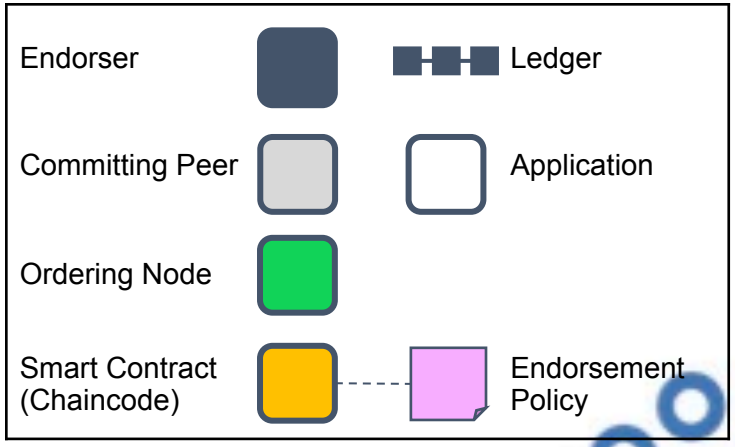


Hyperledger Fabric v1.0 Network

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

Legend



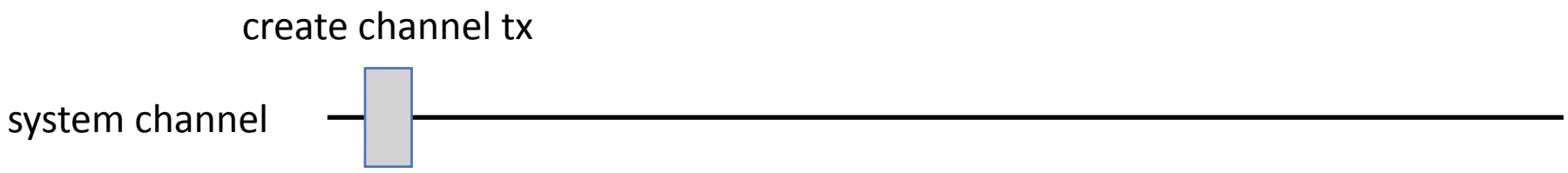
Channel

Data Isolation != Sharding

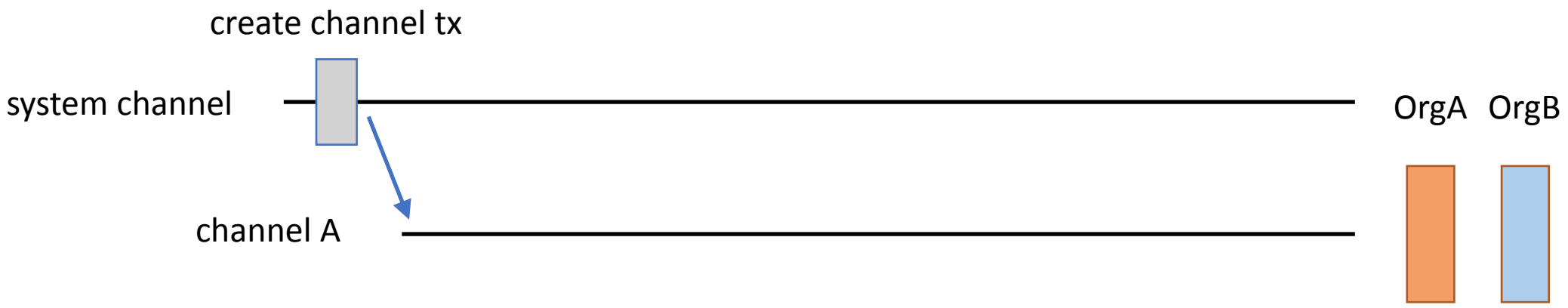
Channel

system channel

Channel



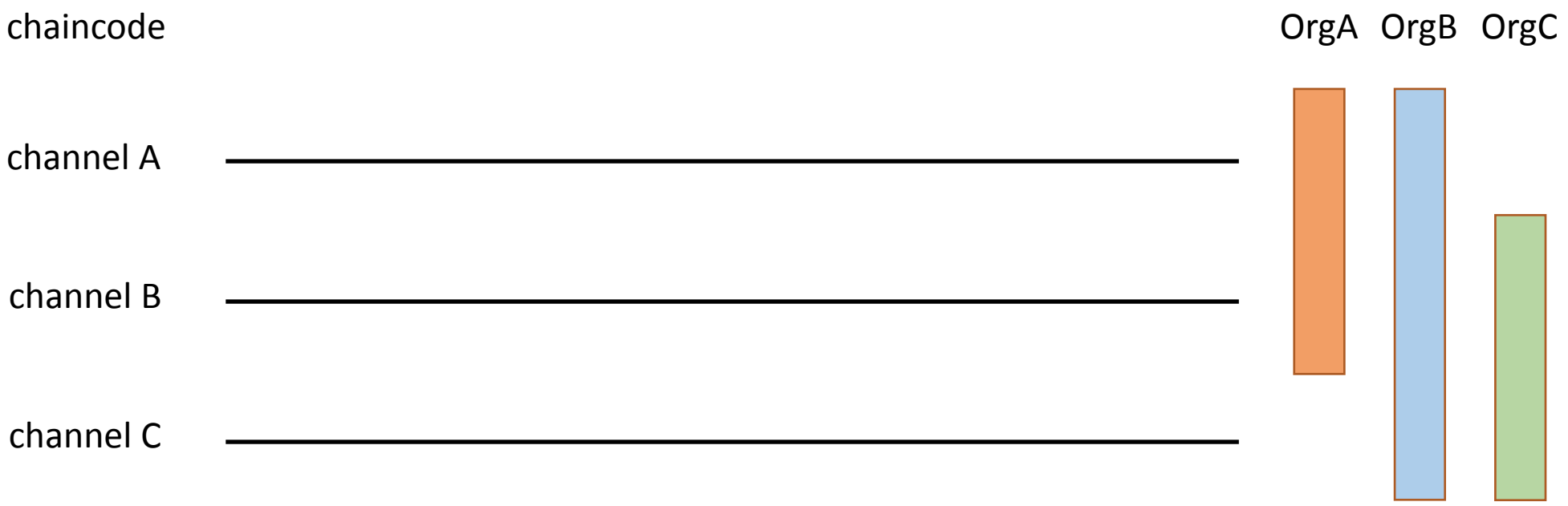
Channel



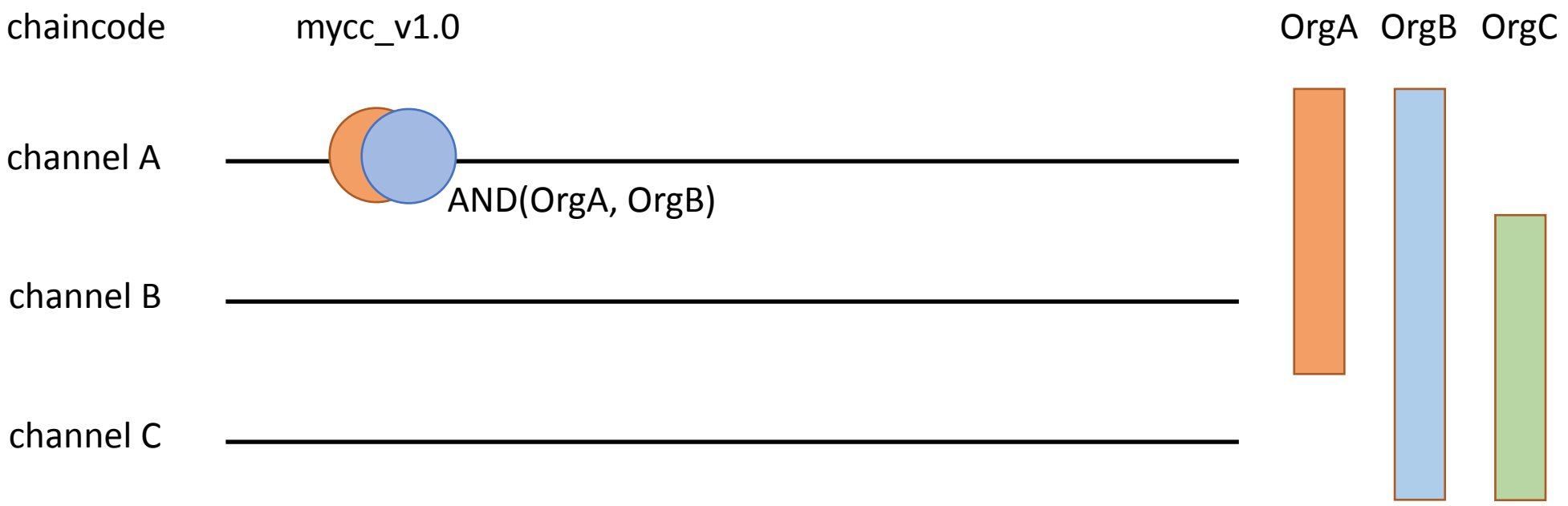
Channel


create channel tx
system channel
OrgA
OrgB
OrgC
channel A
channel B
channel C

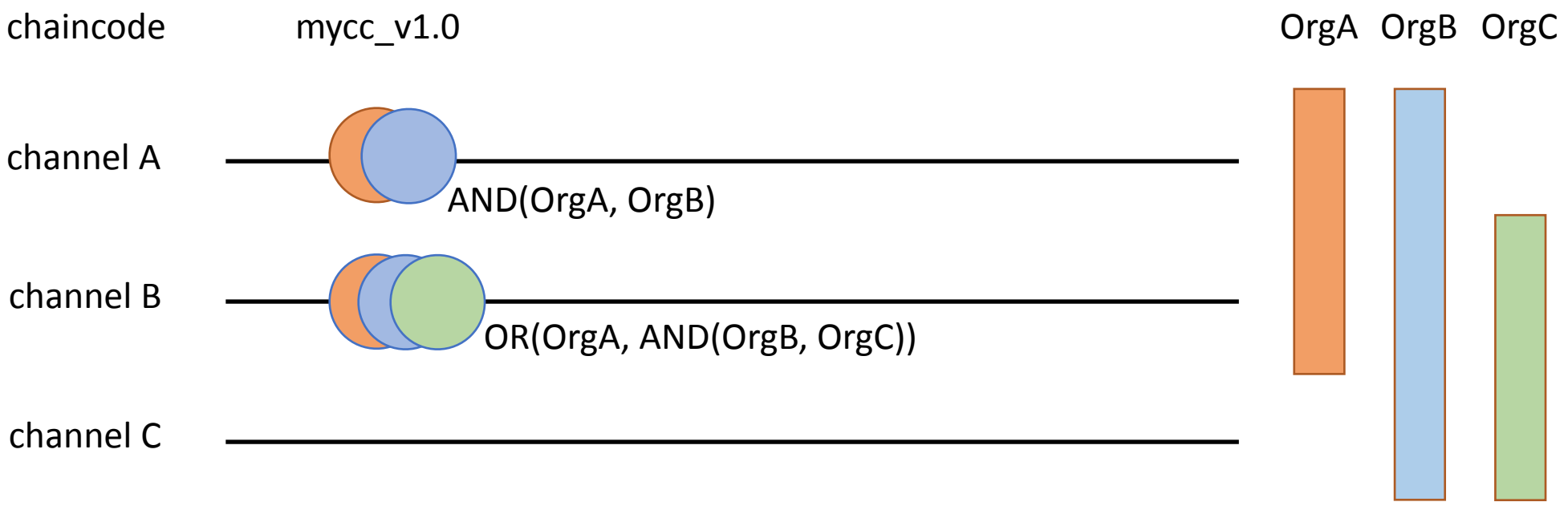
Channel



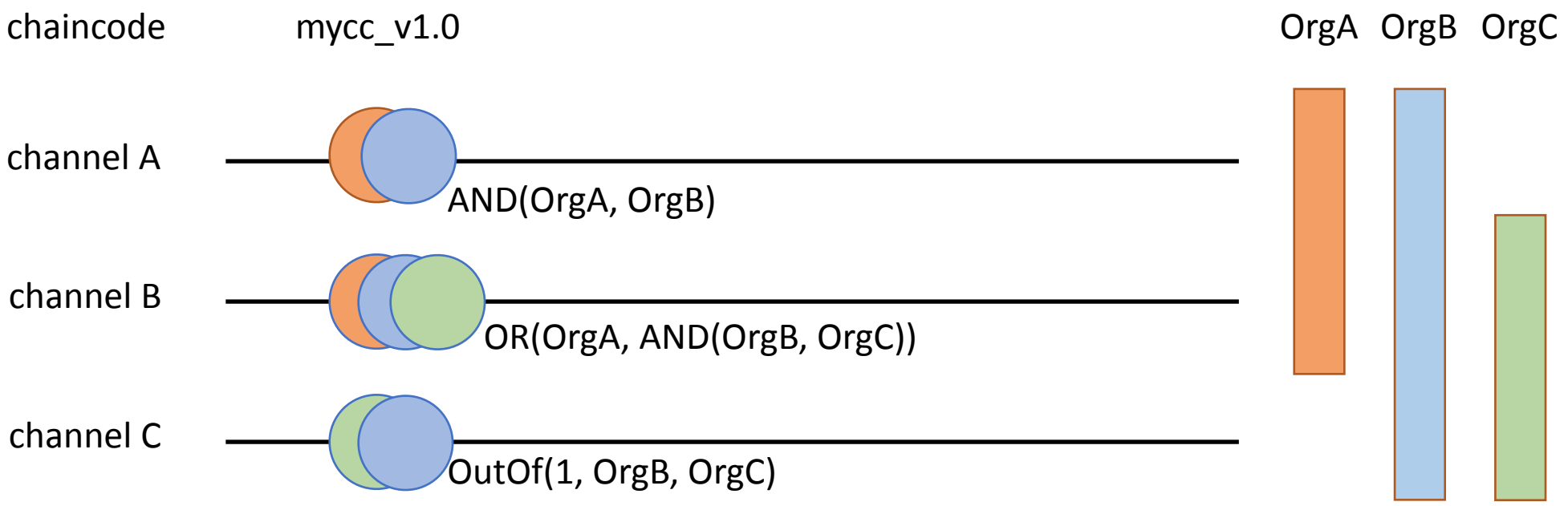
Channel


chaincode
mycc_v1.0
OrgA
OrgB
OrgC
channel A
AND(OrgA, OrgB)
channel B
channel C

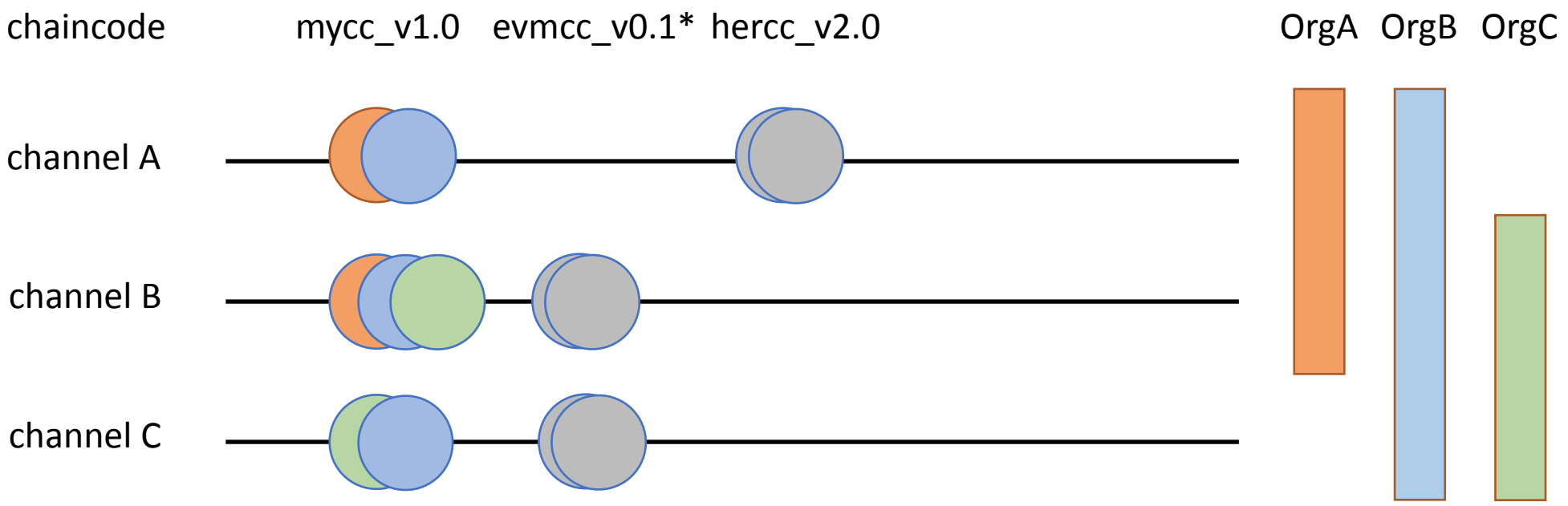
Channel


chaincode
mycc_v1.0
OrgA
OrgB
OrgC
channel A
AND(OrgA, OrgB)
channel B
OR(OrgA, AND(OrgB, OrgC))
channel C

Channel


chaincode
mycc_v1.0
OrgA
OrgB
OrgC
channel A
AND(OrgA, OrgB)
channel B
OR(OrgA, AND(OrgB, OrgC))
channel C
OutOf(1, OrgB, OrgC)

Channel



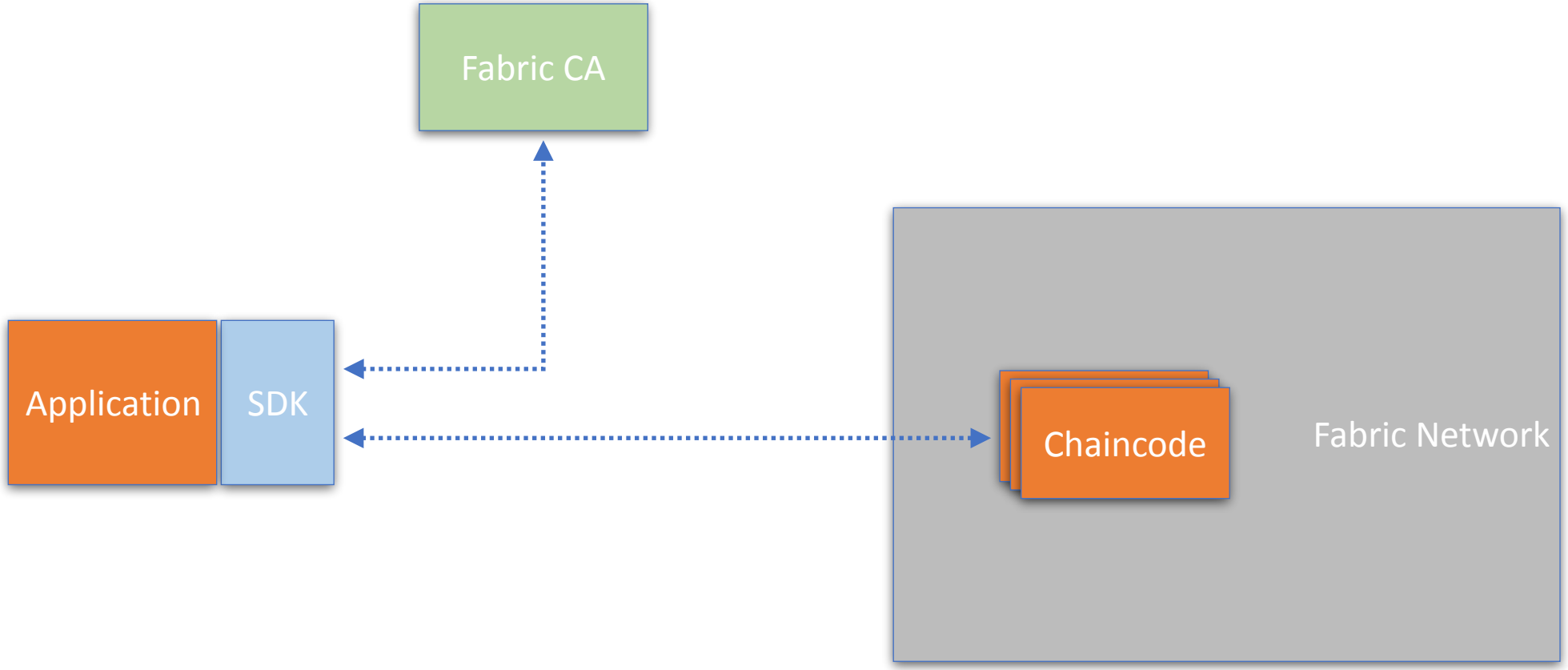
** github.com/hyperledger/fabric-chaincode-evm*

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)
- …

Develop Applications

Develop Applications

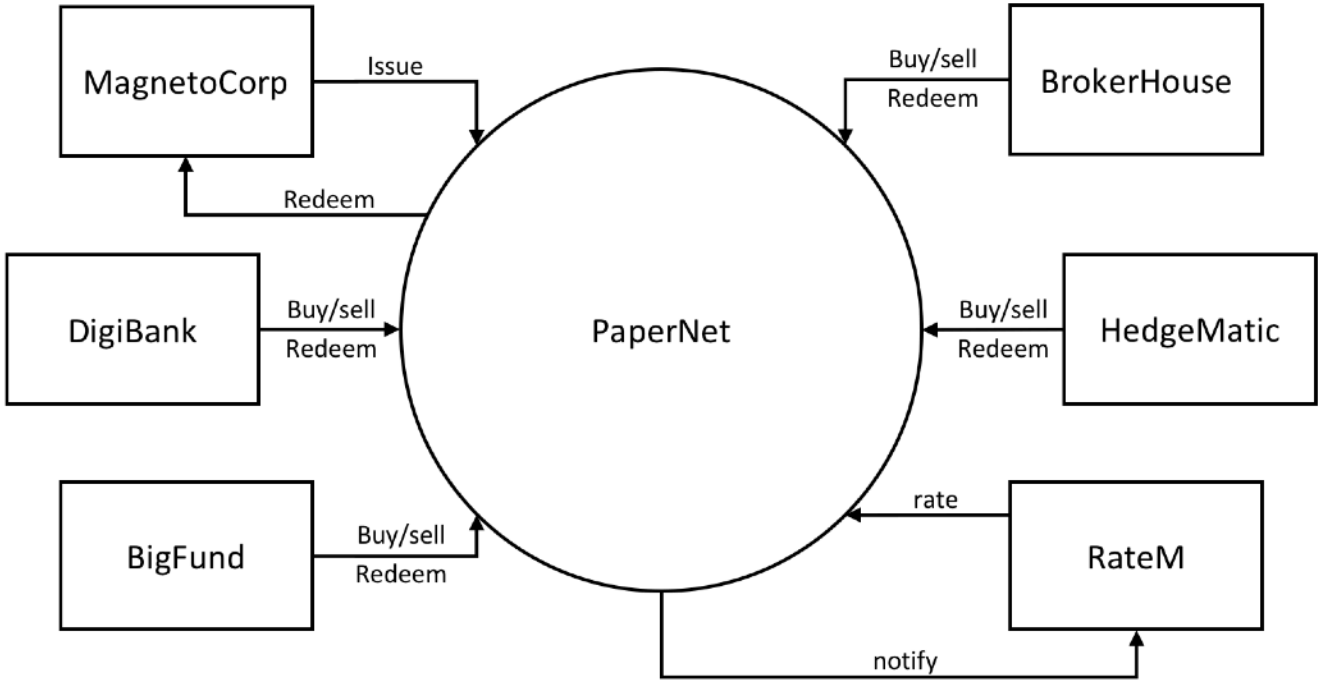


Develop Applications

```
// 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

```
Txn = issue
Issuer = MagnetoCorp
Paper = 00001
Issue time = 31 May 2020 09:00:00 EST
Maturity date = 30 November 2020
Face value = 5M USD
```

```
Issuer = MagnetoCorp
Paper = 00001
Owner = MagnetoCorp
Issue date = 31 May 2020
Maturity = 30 November 2020
Face value = 5M USD
Current state = issued
```

```
Txn = buy
Issuer = MagnetoCorp
Paper = 00001
Current owner = MagnetoCorp
New owner = DigiBank
Purchase time = 31 May 2020 10:00:00 EST
Price = 4.94M USD
```

```
Issuer = MagnetoCorp
Paper = 00001
Owner = DigiBank
Issue date = 31 May 2020
Maturity date = 30 November 2020
Face value = 5M USD
Current state = trading
```

```
Txn = redeem
Issuer = MagnetoCorp
Paper = 00001
Current owner = HedgeMatic
Redeem time = 30 Nov 2020 12:00:00 EST
```

```
Issuer = MagnetoCorp
Paper = 00001
Owner = MagnetoCorp
Issue date = 31 May 2020
Maturity date = 30 November 2020
Face value = 5M USD
Current state = redeemed
```

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

add

commercial paper list: org.papernet.paper

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

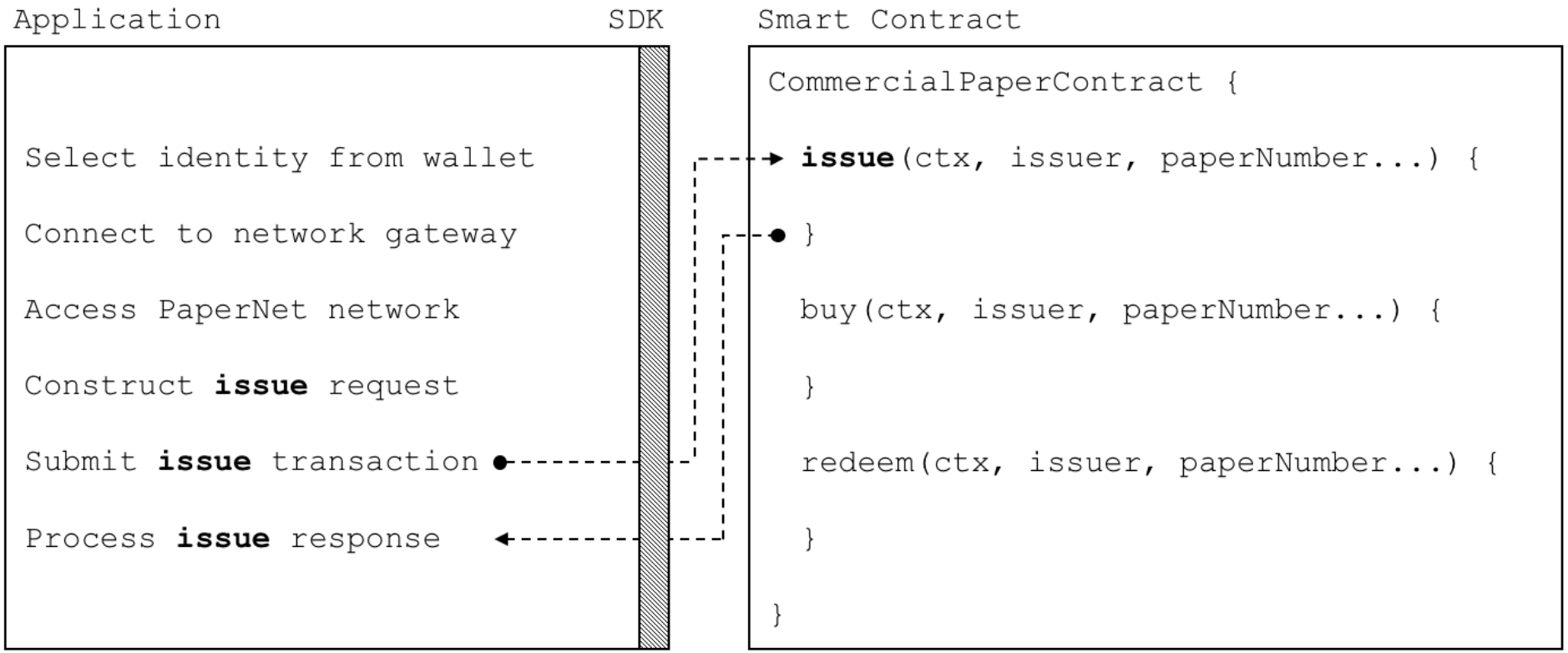
key: org.papaernet.paperMagnetoCorp00001

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
- …

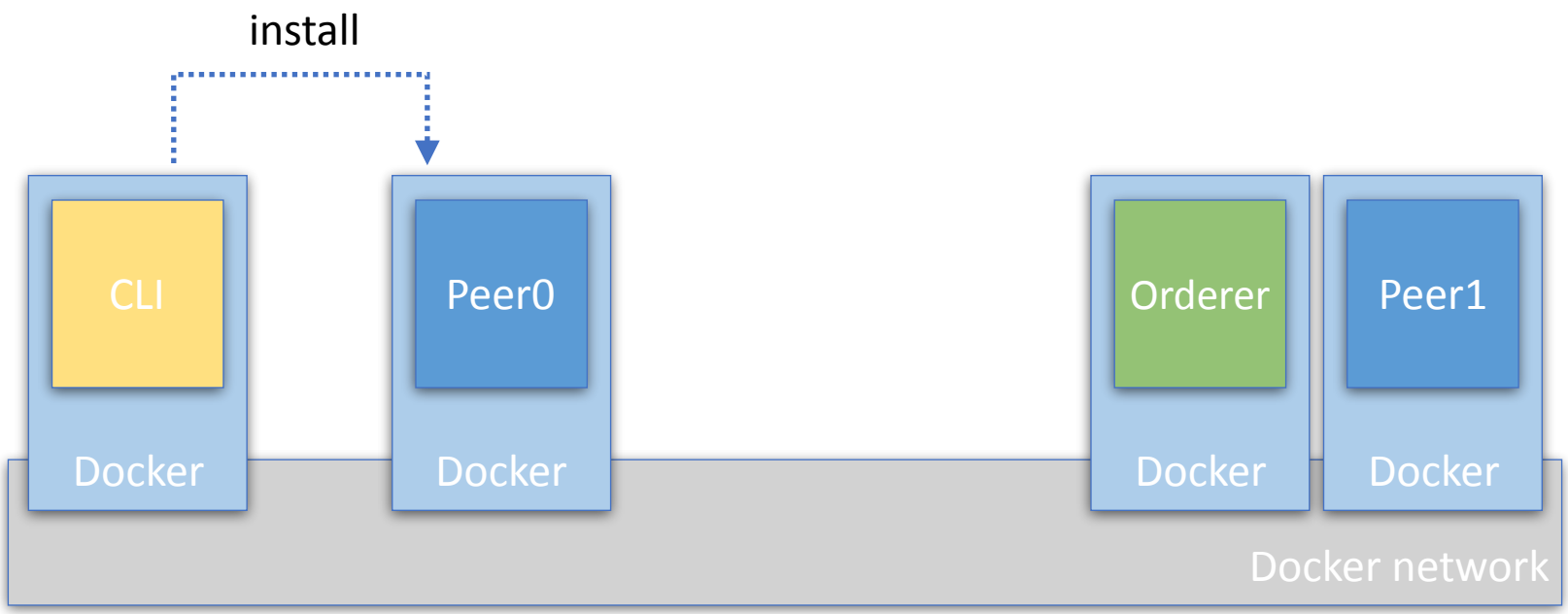
Build Your First Network

Build Your First Network

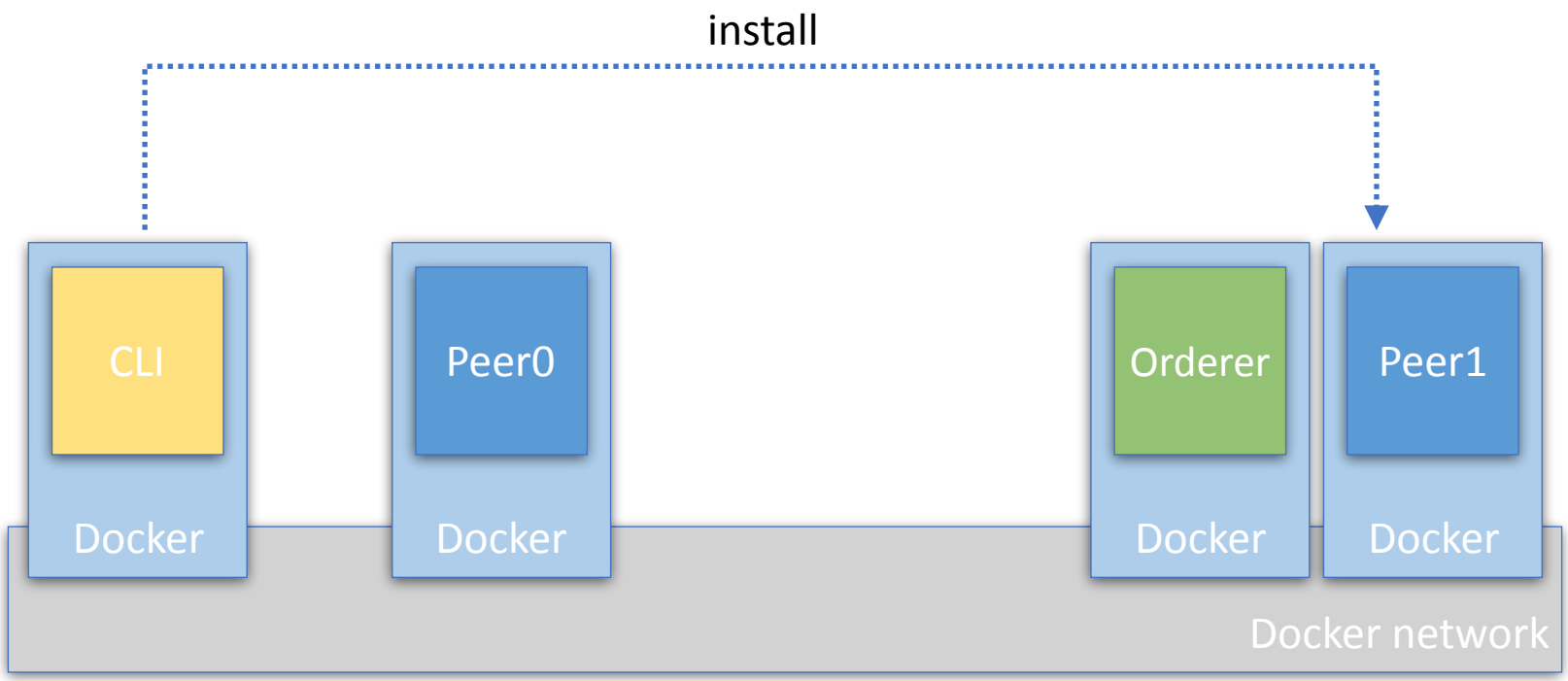
```
ubuntu@oti-server1 → first-network git:(release-1.4)  ./byfn.sh -h
Usage:
  byfn.sh <mode> [-c <channel name>] [-t <timeout>] [-d <delay>] [-f <docker-compose-file>] [-s <dbtype>] [-l <language>] [-o <consensus-type>] [-i <imagetag>] [-v]
    <mode> - one of 'up', 'down', 'restart', 'generate' or 'upgrade'
      - 'up' - bring up the network with docker-compose up
      - 'down' - clear the network with docker-compose down
      - 'restart' - restart the network
      - 'generate' - generate required certificates and genesis block
      - 'upgrade'  - upgrade the network from version 1.3.x to 1.4.0
    -c <channel name> - channel name to use (defaults to "mychannel")
    -t <timeout> - CLI timeout duration in seconds (defaults to 10)
    -d <delay> - delay duration in seconds (defaults to 3)
    -f <docker-compose-file> - specify which docker-compose file use (defaults to docker-compose-cli.yaml)
    -s <dbtype> - the database backend to use: goleveldb (default) or couchdb
    -l <language> - the chaincode language: golang (default) or node
    -o <consensus-type> - the consensus-type of the ordering service: solo (default), kafka, or etcdraft
    -i <imagetag> - the tag to be used to launch the network (defaults to "latest")
    -v - verbose mode
  byfn.sh -h (print this message)

Typically, one would first generate the required certificates and
genesis block, then bring up the network. e.g.:

        byfn.sh generate -c mychannel
        byfn.sh up -c mychannel -s couchdb
        byfn.sh up -c mychannel -s couchdb -i 1.4.0
        byfn.sh up -l node
        byfn.sh down -c mychannel
        byfn.sh upgrade -c mychannel

Taking all defaults:
        byfn.sh generate
        byfn.sh up
        byfn.sh down
```

Build Your First Network

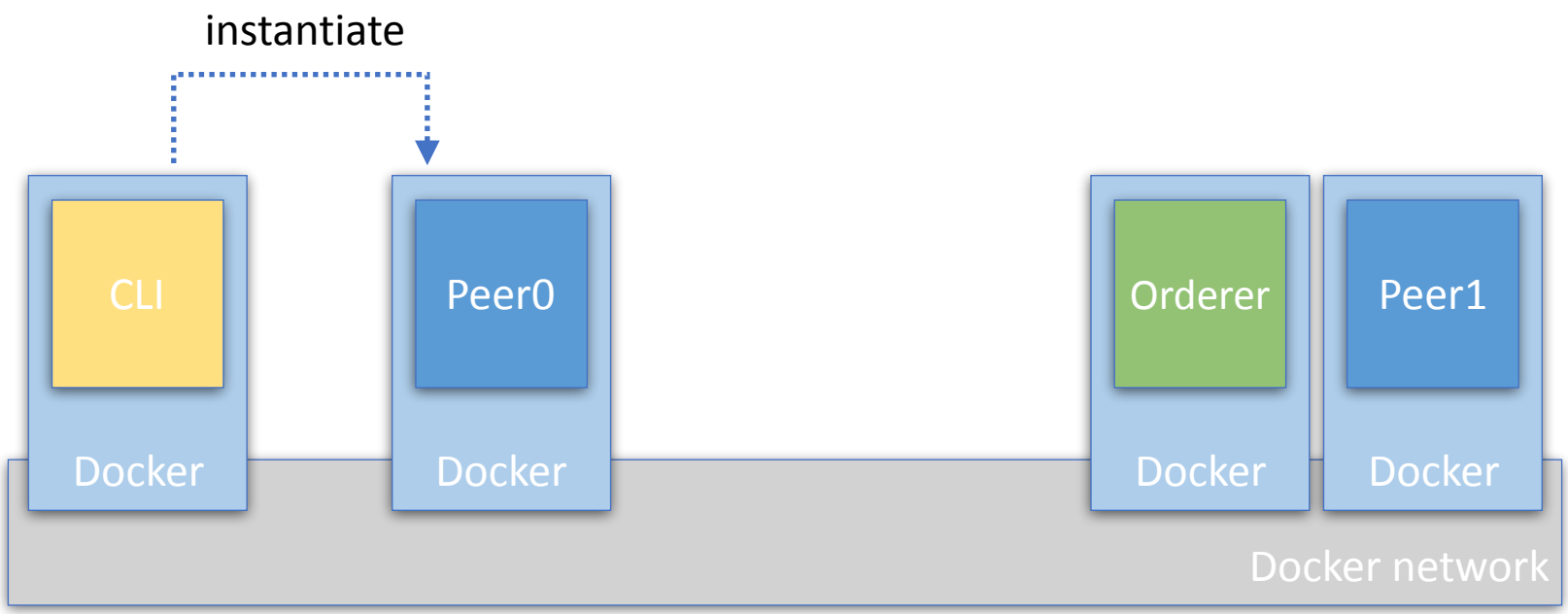
Build Your First Network



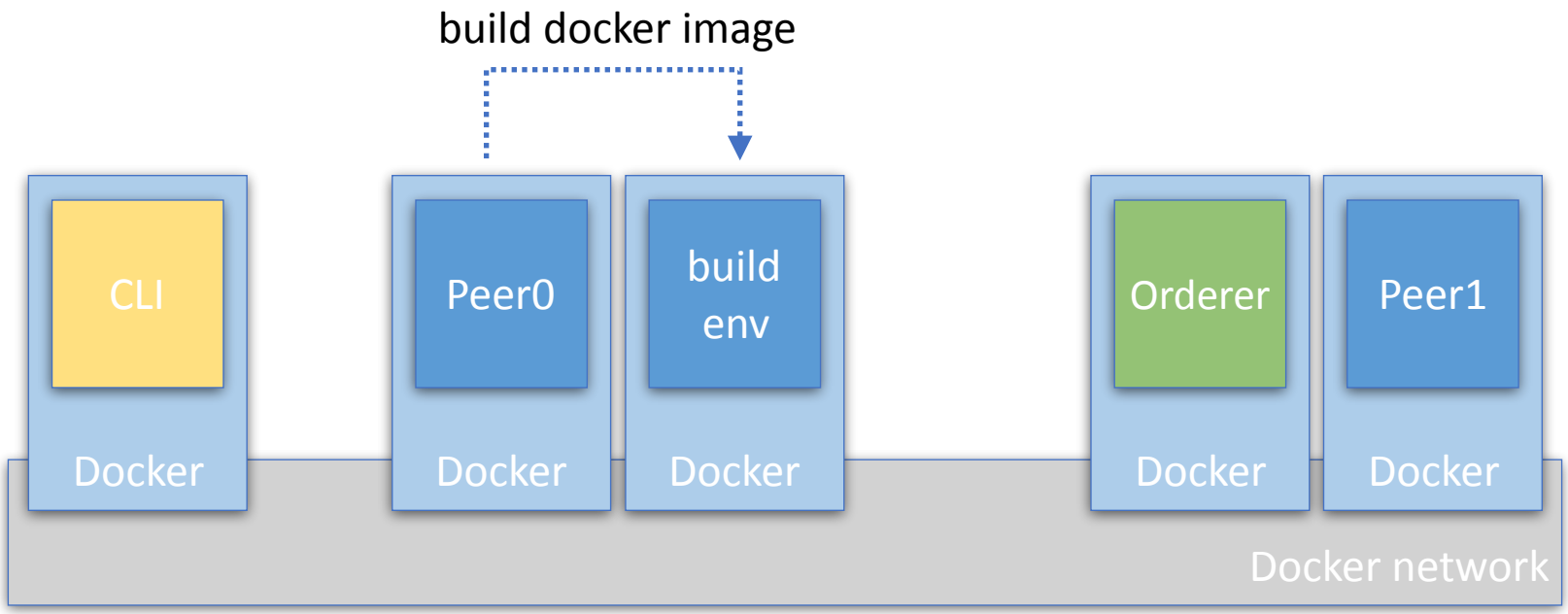
Build Your First Network



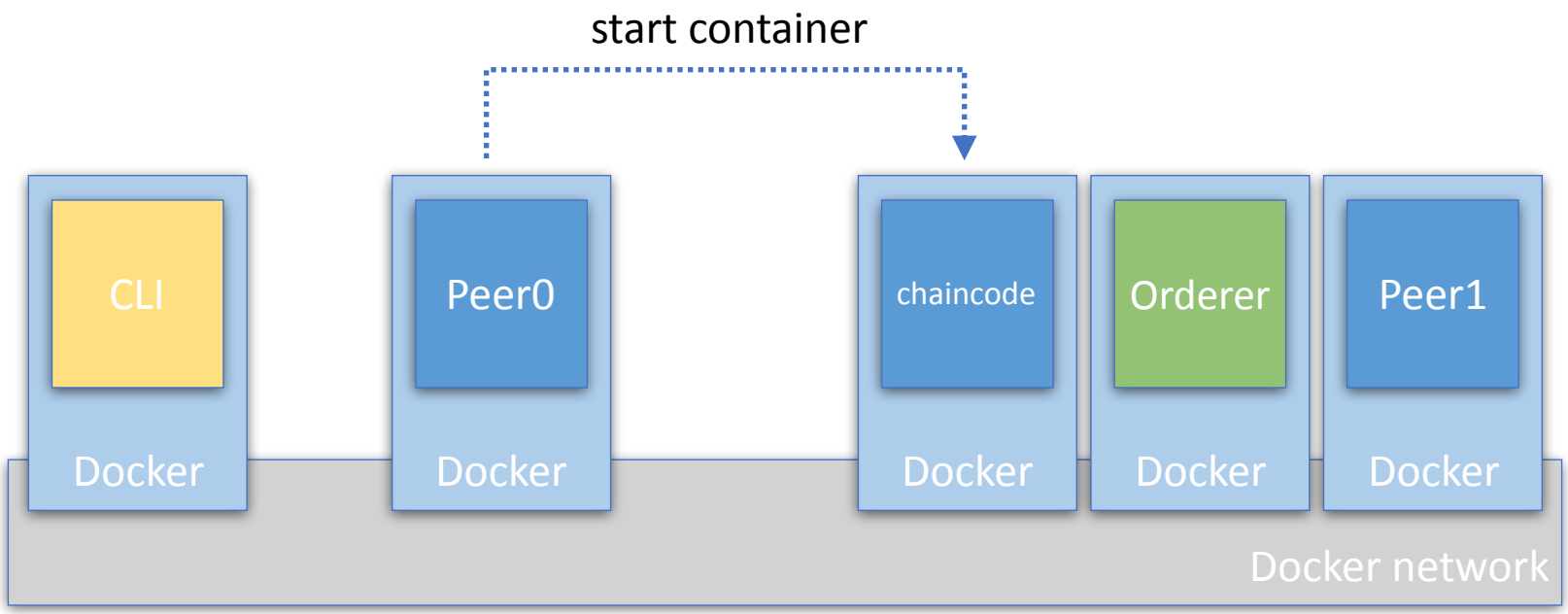
Build Your First Network



Build Your First Network



Build Your First Network



IBM

Open by design™

未完待续...

课程安排

03/14 区块链赋能产业价值和商业模式
03/21 Hyperledger 项目概览 社区介绍
03/28 Fabric 1.4 LTS概述
04/04 Peer 解析
04/11 Orderer 解析
04/18 MSP 与 CA
04/25 应用开发指南
05/09 部署实践

欢迎关注微信公众号
“IBM开源技术”
获取更多资讯

公众号中发送“**报名**”,
即有机会参加Fabric线下训练营