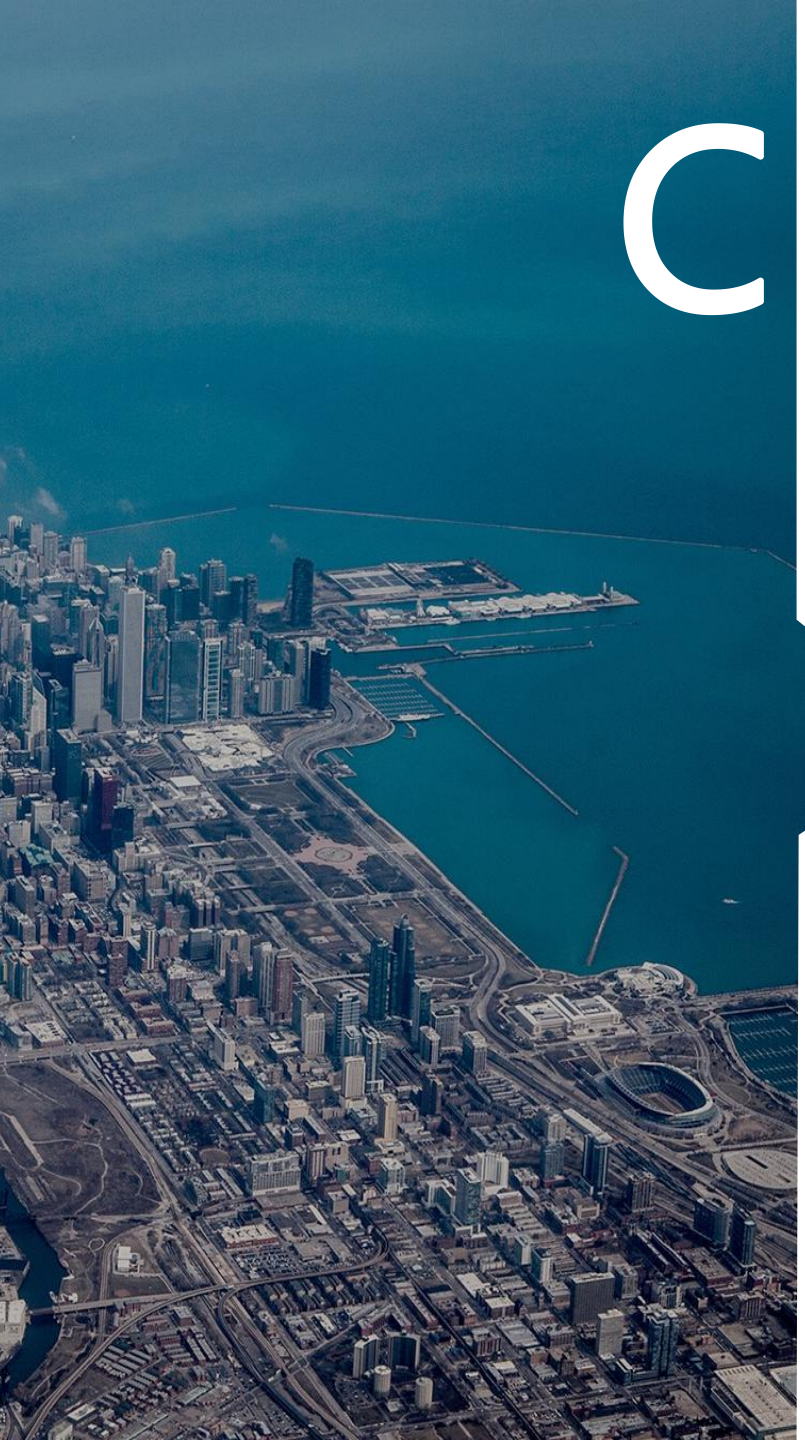
区块链技术与应用

华南理工大学　　许可　　kexu@scut.edu.cn

本课件主要内容来源于IBM

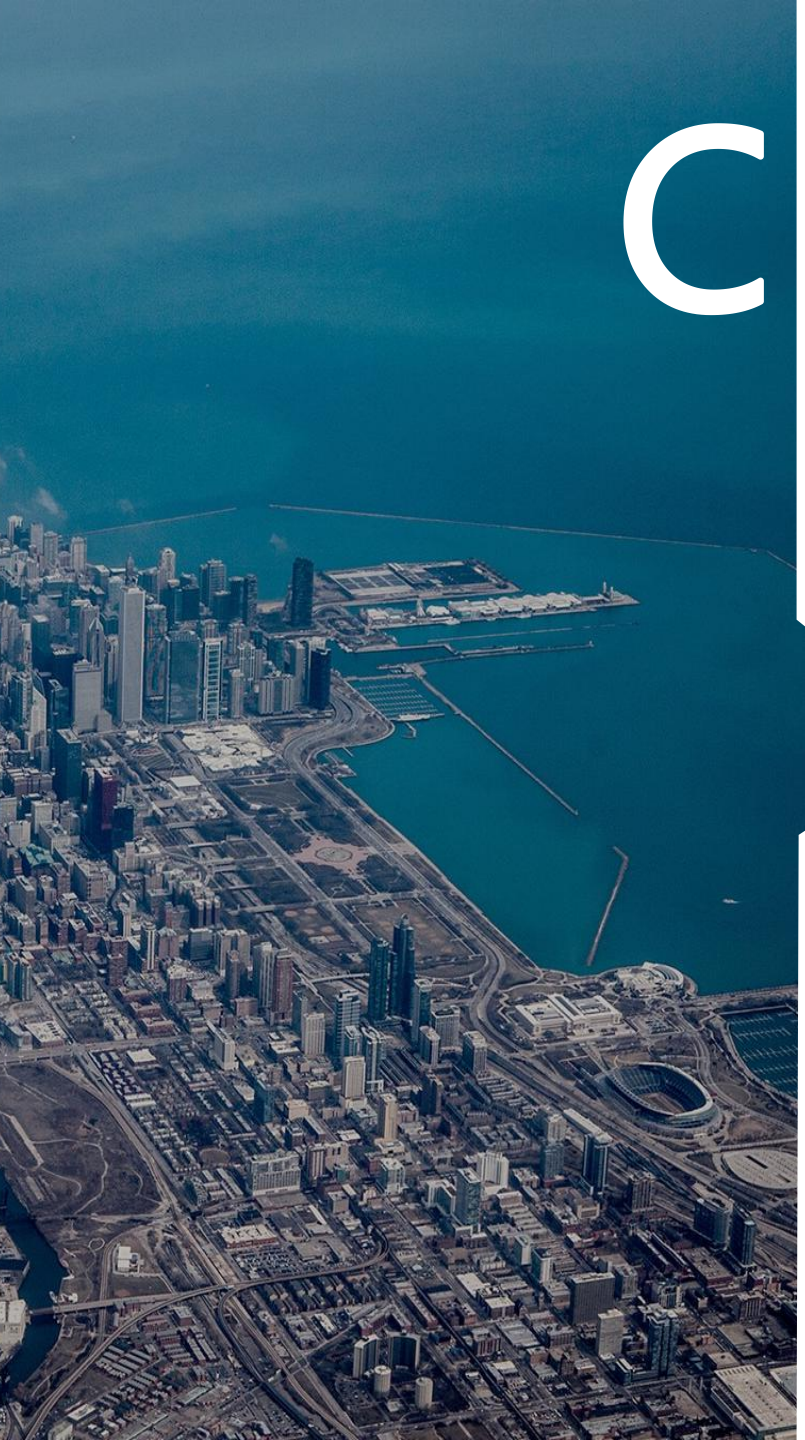
Unit 3 Fabric Peer and Orderer



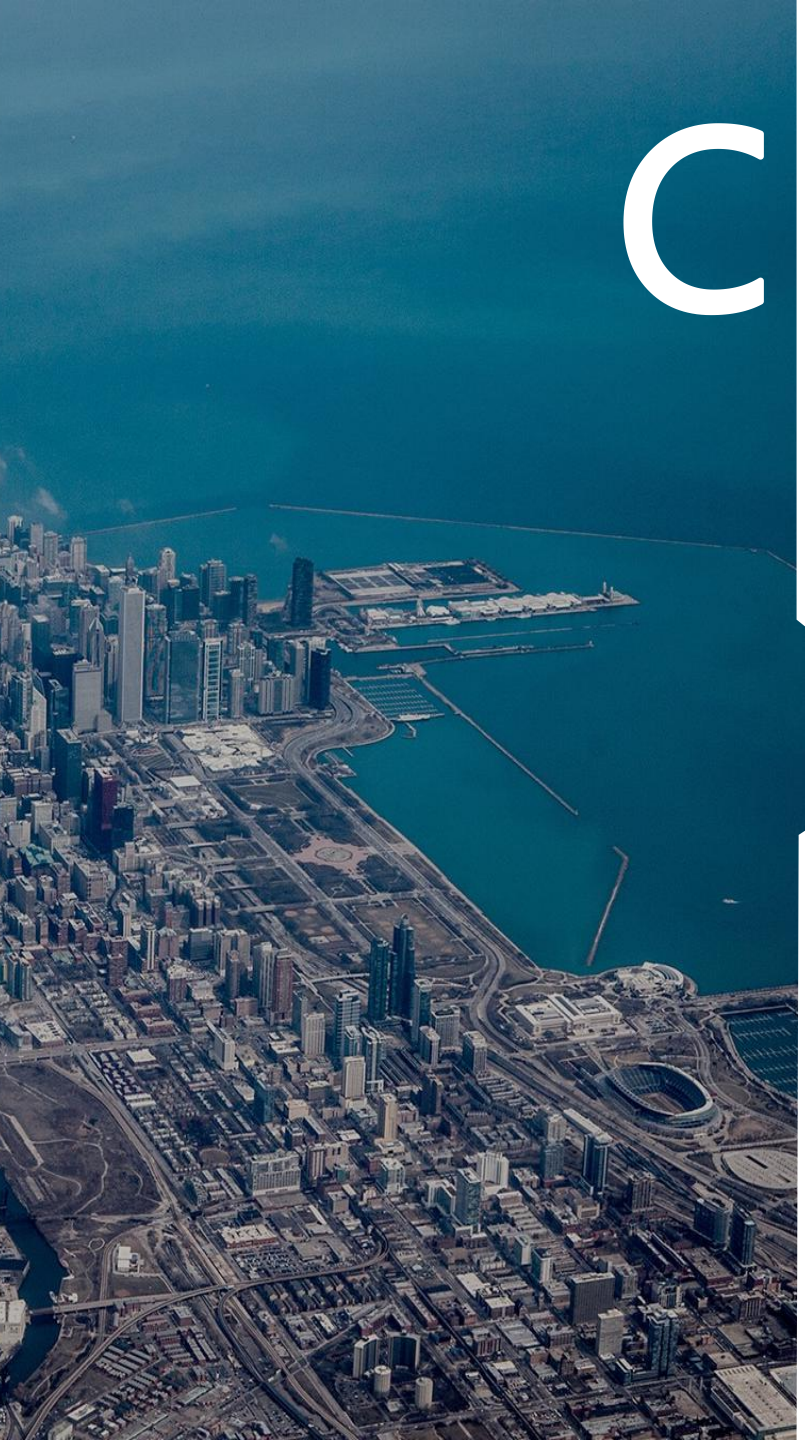
Contents

Part ONE

Fabric Peer



Contents



Contents

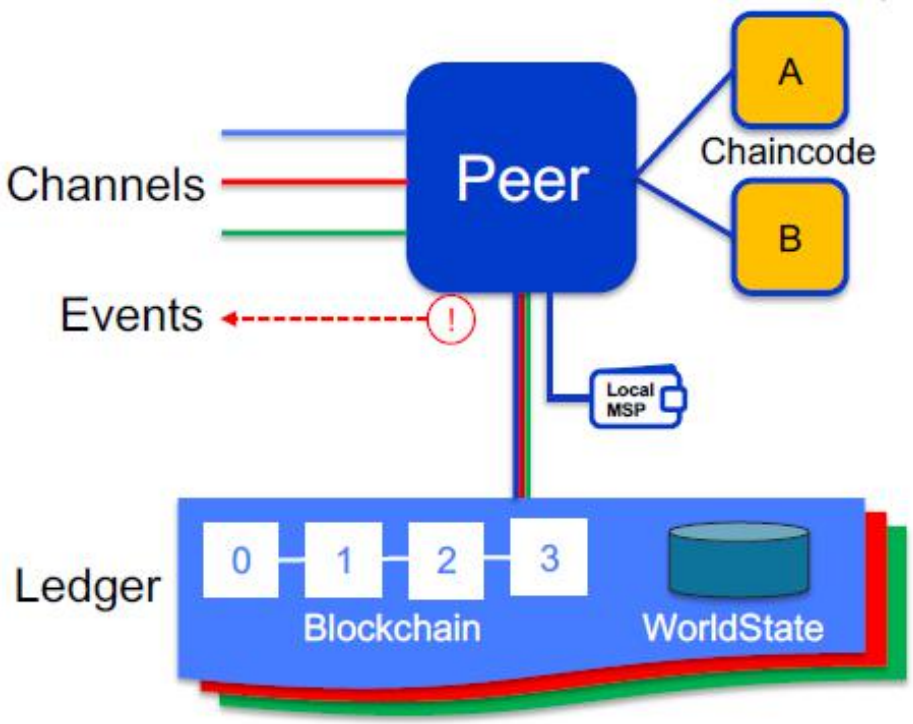
Part 1.1

Fabric Peer in Network Consensus

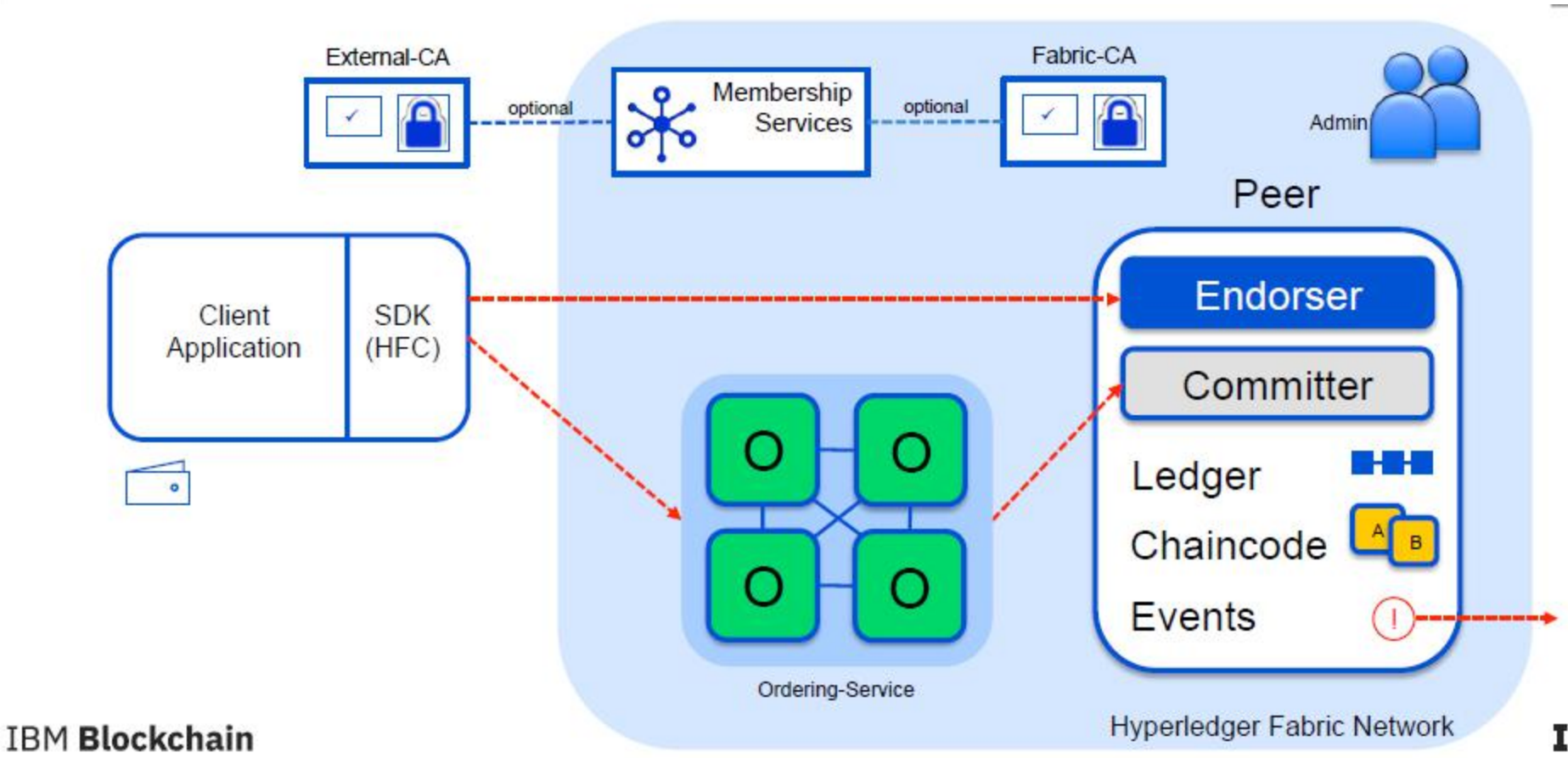
Fabric Peer

- A blockchain network is comprised primarily of a set of *peer nodes*
- Host ledgers and smart contracts

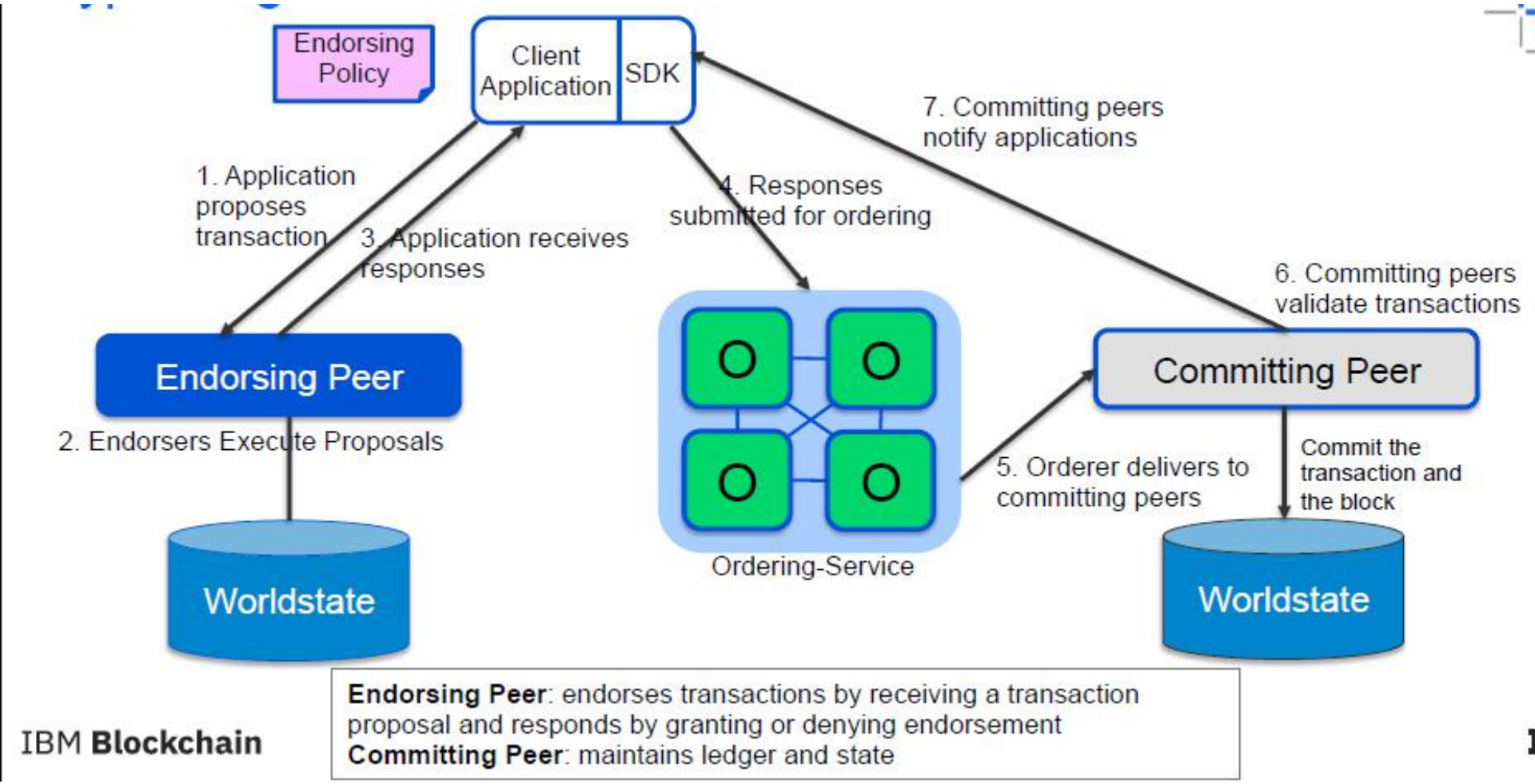
- Each peer:
  - Connects to one or more channels
  - Maintains one or more ledgers per channel
  - Maintains installed chaincode
  - Manages runtime docker containers for instantiated chaincode
  - Has a local MSP (Membership Services Provider) that provides crypto material
  - Emits events to the client application



Hyperledger Fabric V1.x Architecture



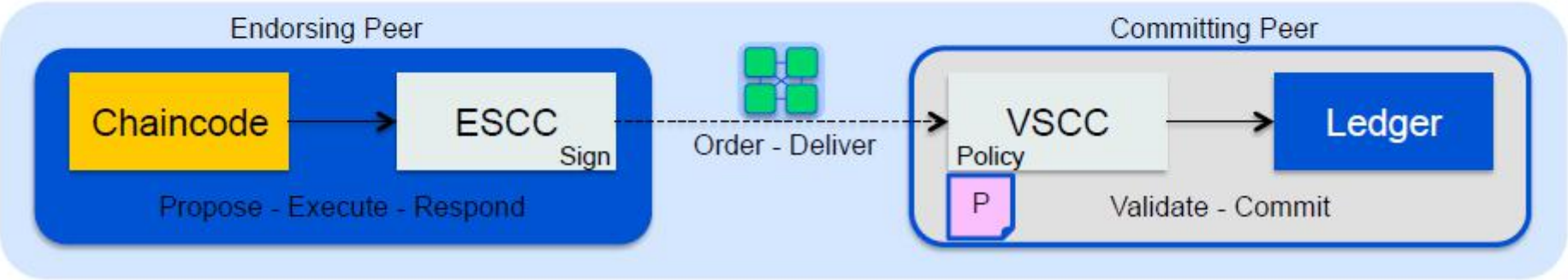
Hyperledger Fabric V1.x Transaction Flow



Endorsement Policies

An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

- Each chaincode is deployed with an Endorsement Policy
- **ESCC** (Endorsement System ChainCode) signs the proposal response on the endorsing peer
- **VSCC** (Validation System ChainCode) validates the endorsements



Endorsement Policies Syntax

```
$ peer chaincode instantiate
-C mychannel
-n mycc
-v 1.0
-p chaincode_example02
-c '{"Args":["init","a", "100", "b","200"]}'
-P "AND('Org1MSP.member')"
```

Instantiate the chaincode mycc on channel mychannel with the policy AND('Org1MSP.member')

Policy Syntax: EXPR(E[, E...])

Where EXPR is either AND, OR or OutOf and E is either a principal or nested EXPR

Principal Syntax: MSP.ROLE

Supported roles are: member, admin, client, peer

Where MSP is the MSP ID, and ROLE is either “member” or “admin”

Endorsement Policies Examples

Examples of policies:

- Request 1 signature from all three principals
  - AND('Org1.member', 'Org2.member', 'Org3.member')
- Request 1 signature from either one of the two principals
  - OR('Org1.member', 'Org2.member')
- Request either one signature from a member of the Org1 MSP or (1 signature from a member of the Org2 MSP and 1 signature from a member of the Org3 MSP)
  - OR('Org1.member', AND('Org2.member', 'Org3.member'))

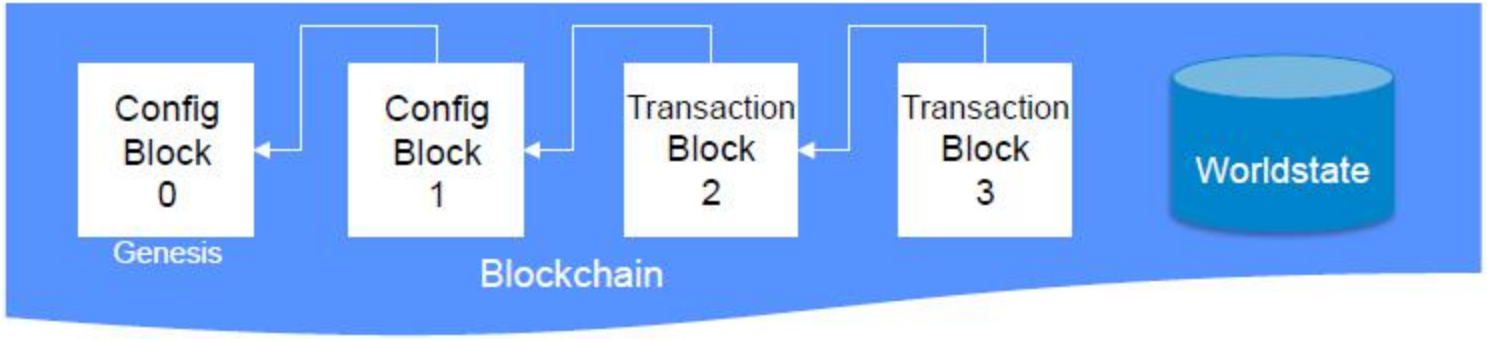
Part 1.2

Fabric Ledger and State DB

Fabric Ledger

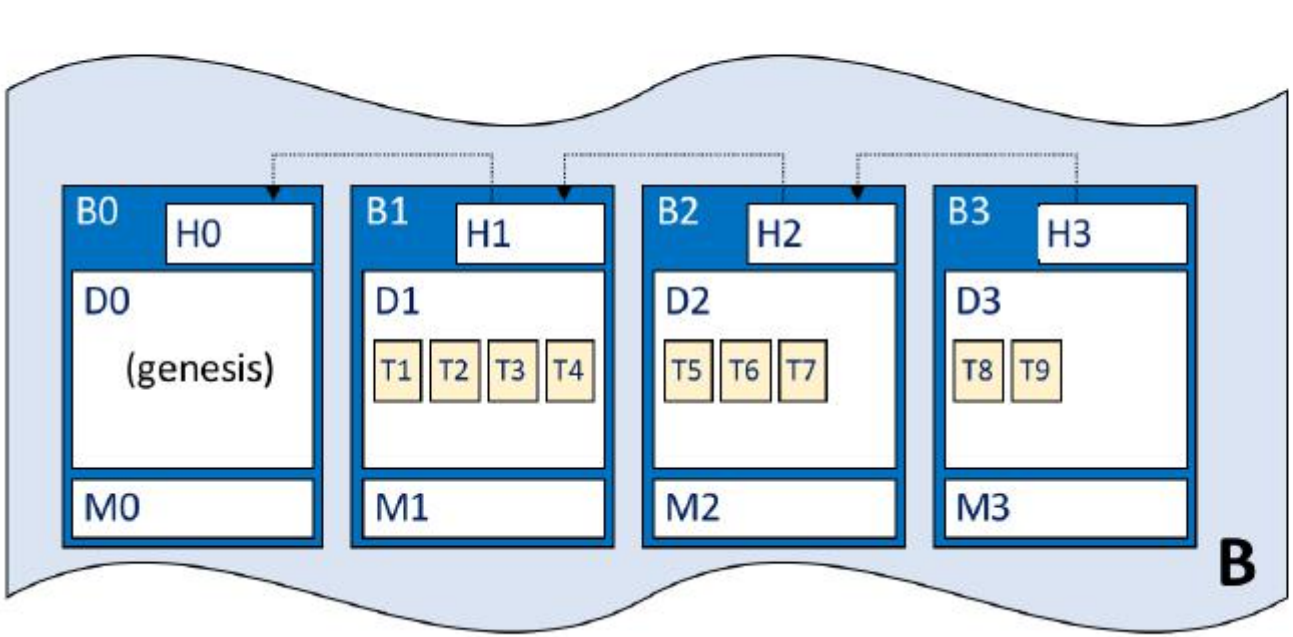
- The Fabric ledger is sequenced, tamper-resistant record of all **state** transitions

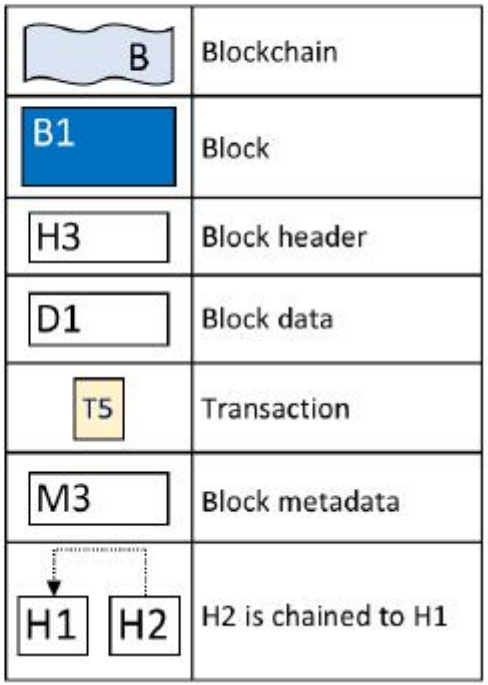
- Blockchain
  - Channel configurations
  - Immutable, sequenced record in blocks
- World state
  - Maintain current state



Block Chain

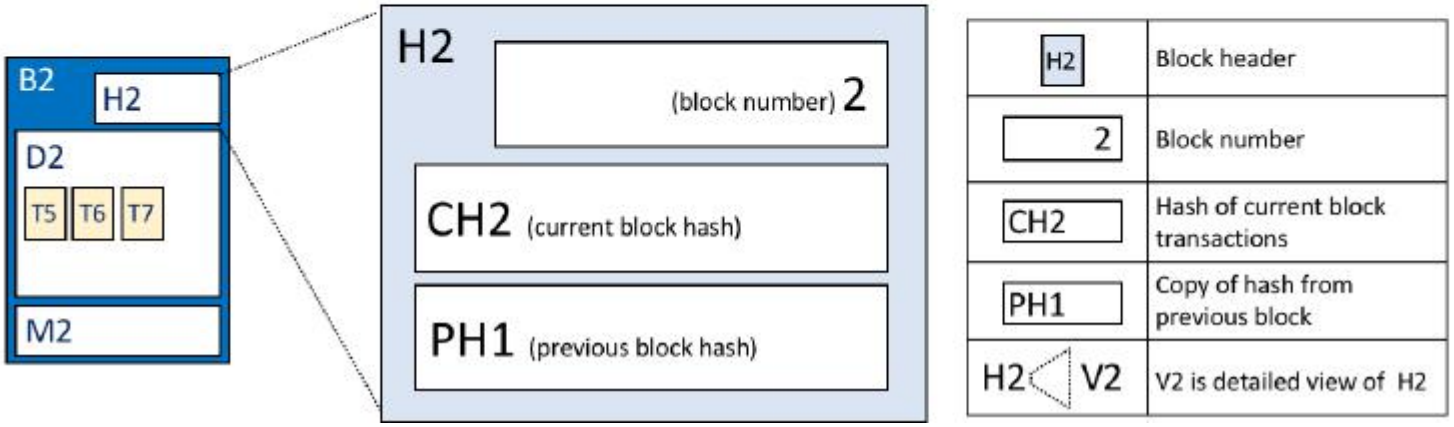
An historical record of the facts about how these objects arrived at their current states.





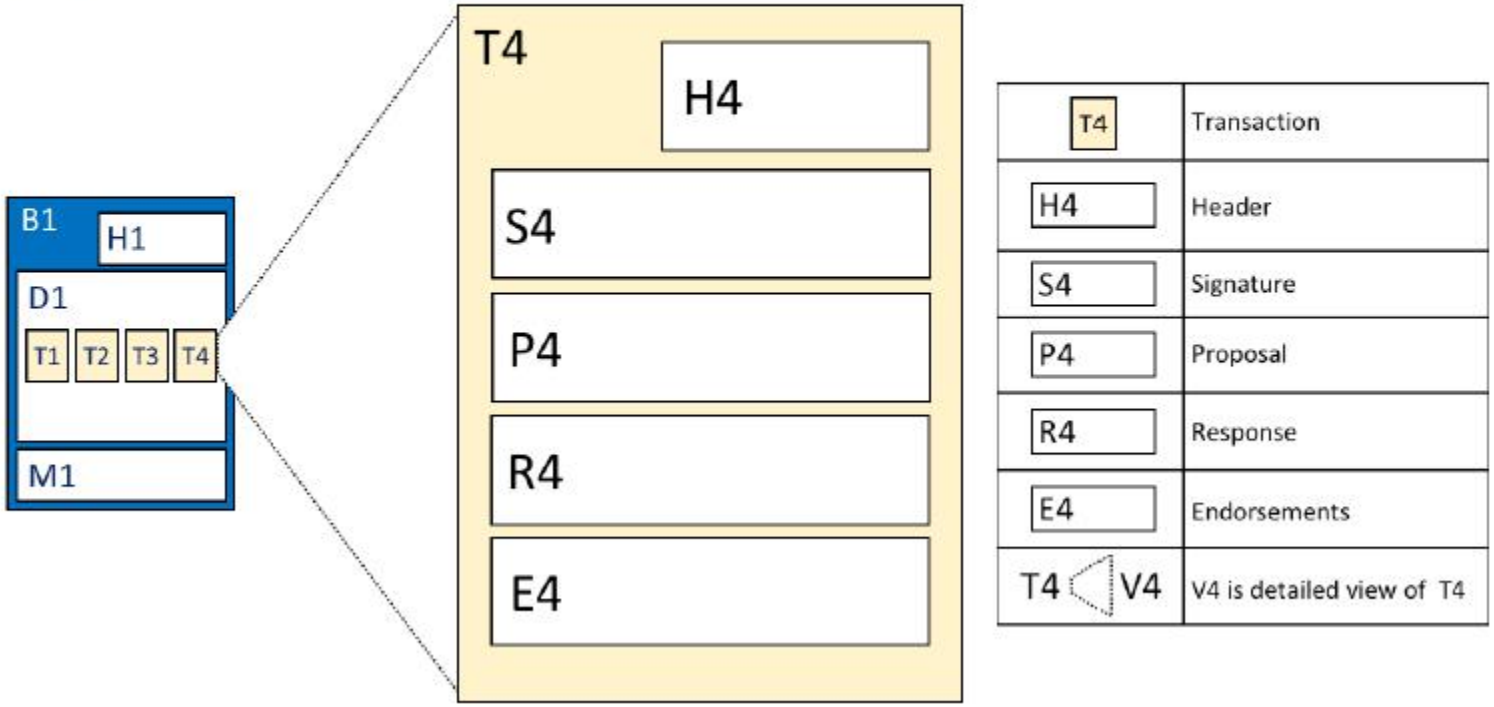
Block Chain

- Block header
  - Block number
  - Current Block Hash
  - Previous Block Hash
- Block Data
- Block Metadata



Transactions

Captures some essential metadata about the transaction

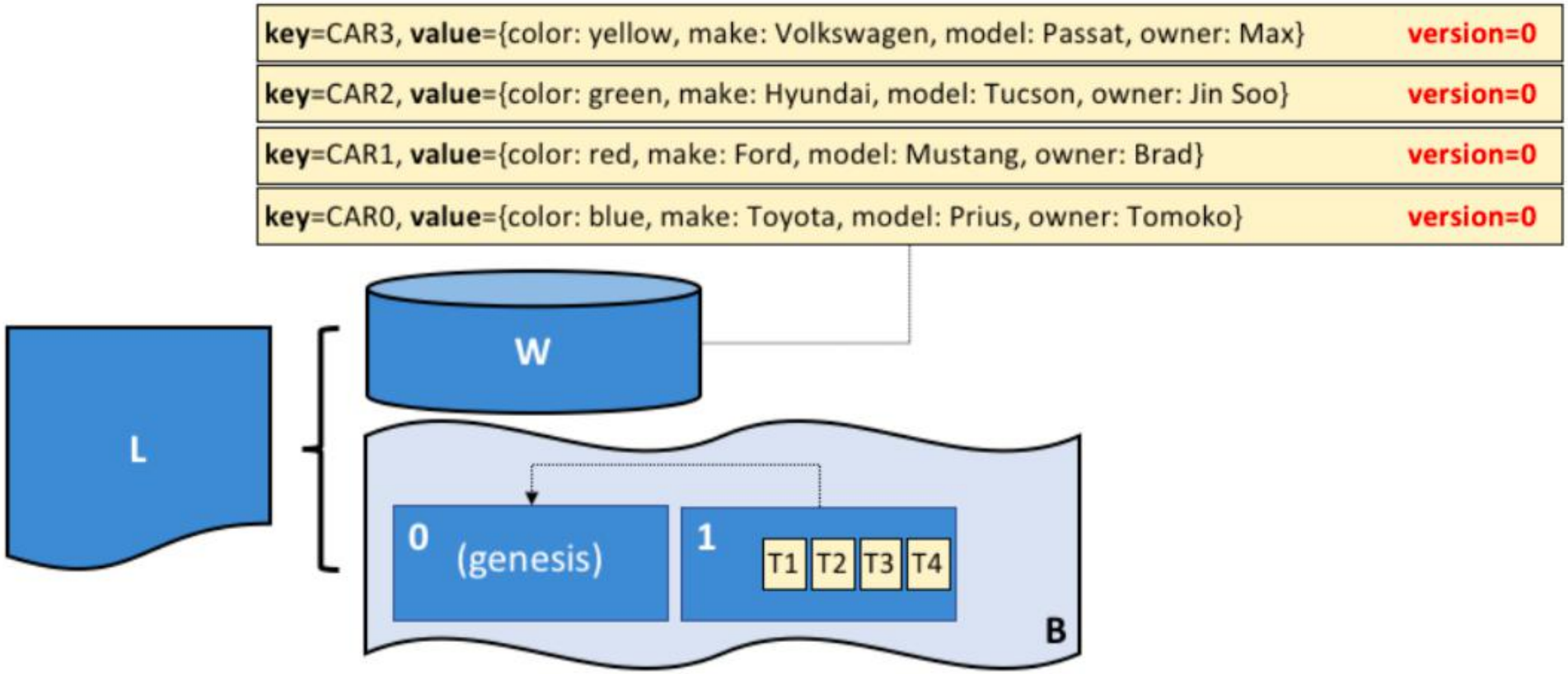


World State

Holds current state of a set of business objects

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

Example Ledger: fabcar



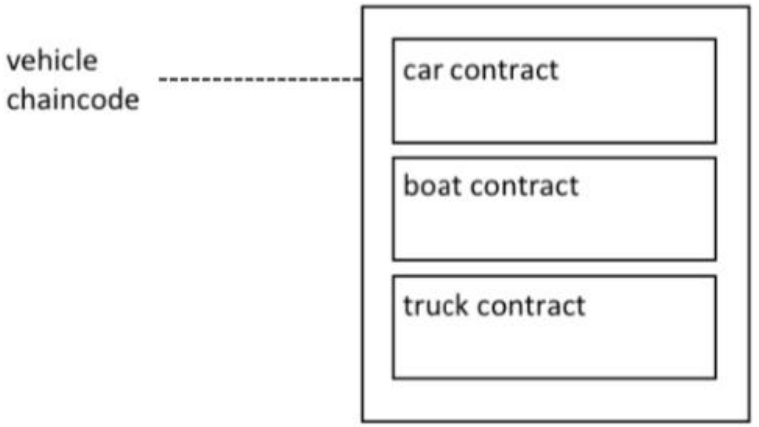
Part 1.3

Smart Contract

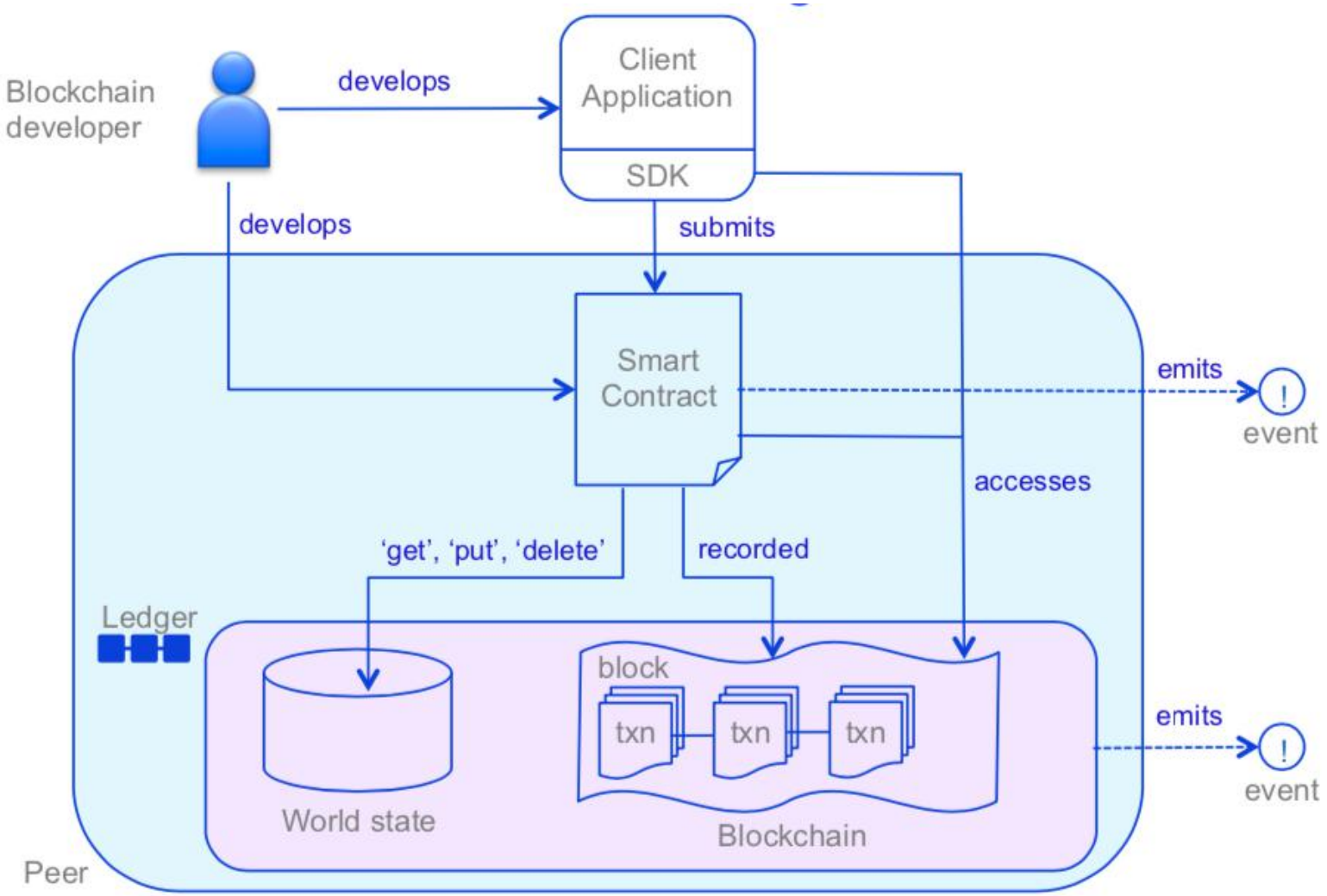
Smart Contract & chaincode

- Smart Contract
  - Heart of a blockchain network
  - Defines the rules between different organizations in executable code
  - Generate transactions that are recoded on the ledger
  - packaged into a chaincode

- Chaincode
  - Can package multiple smart contracts
  - Smart contacts are available to applications when a chain code is deployed



How Smart Contract interact with the legder



Smart Contract Example -SimpleSet

- Add the Go import statements

```
import (
    "fmt"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)
// SimpleChaincode example simple Chaincode implementation
type SimpleChaincode struct {
}
```

- Initializing

```
// Init is called during chaincode instantiation to initialize any data.

func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
    // Initialize the chaincode
    A = args[0]
    Aval, err = strconv.Atoi(args[1])
    B = args[2]
    Bval, err = strconv.Atoi(args[3])
    fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)
    // Write the state to the ledger
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
    return shim.Success(nil)
}
```

Smart Contract Example -SimpleSet

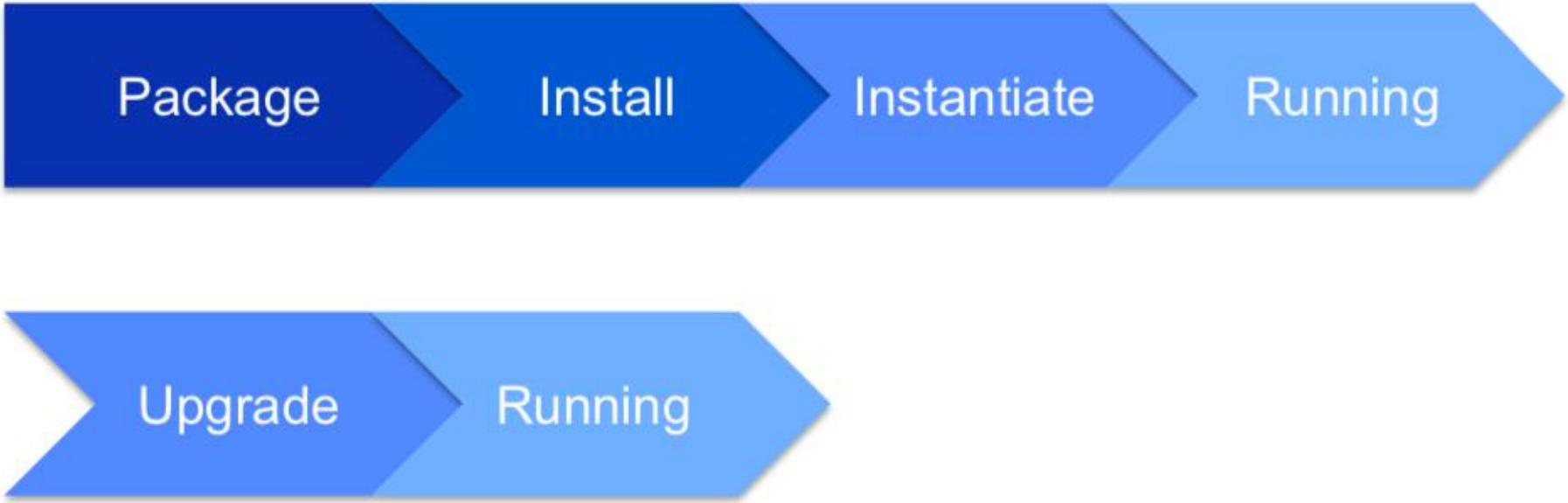
- Invoking

```
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {

    function, args := stub.GetFunctionAndParameters()
    if function == "invoke" {
        // Make payment of X units from A to B
        return t.invoke(stub, args)
    } else if function == "delete" {
        // Deletes an entity from its state
        return t.delete(stub, args)
    } else if function == "query" {
        // the old "Query" is now implemtned in invoke
        return t.query(stub, args)
    }

    return shim.Error("Invalid invoke function name. Expecting \"invoke\"
\"delete\" \"query\"")
}
```

Chaincode Lifecycle



Chaincode Lifecycle - Package

- Packaging
  - ChaincodeDeploymentSpec (CDS) - the source code, the name, and version of the chaincode
  - An instantiation policy, expressed as endorsement policies
  - A set of signatures by the entities that “own” the chaincode

- Example

```
peer chaincode package -n mycc -p github.com/hyperledger/fabric-samples/
chaincode/abstore/go -v 1.0 -s -S -i "AND('OrgA.admin')" ccpack.out

peer chaincode signpackage ccpack.out signedccpack.out
```

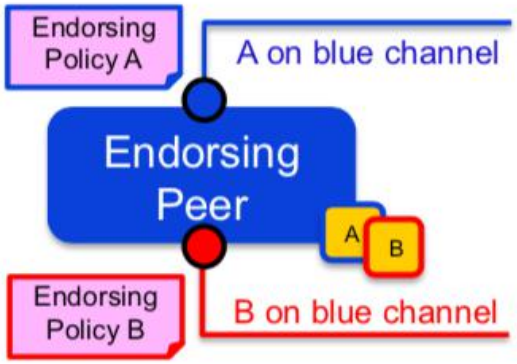
Chaincode Lifecycle – Install & Instantiate

- Installing chaincode
  - Installs chaincode on a peer node
  - Multiple chaincodes could be installed on a peer node
  - Must install the chaincode on each endorsing peer node of a channel
  - Example

```
peer chaincode install ccpack.out
```

- Instantiate chaincode
  - Create and initialize a chaincode on a channel
  - Sets up the endorsement policy during instantiation
  - Example

```
peer chaincode instantiate -n mycc -v 1.0 -c '{"Args":["a",
"100", "b", "200"]}' -P "AND ('Org1.member','Org2.member')"
```



Chaincode Lifecycle – Running & Upgrade

- Running
  - Application/Client submits a transaction
  - Smart contracts handles the transaction, update the ledger and return a response
  - Application/Client receives the response
  - Example

```
peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
peer chaincode invoke -o order-url  -C mychannel -n mycc -c '{"Args":["invoke","a","b","10"]}'
```

- Upgrade
  - A chaincode may be upgraded any time by changing its version
  - Prior to upgrade, the new version of the chaincode must be installed on the required endorsers
  - Similar to the instantiate transaction, only affects one channel at a time

```
peer chaincode upgrade -C mychannel -n mycc -v 1.0 -c '{"Args":
["a","100","b","200"]}'
```

System Chaincode

- Runs within the peer process rather than in an isolated container like normal chaincode
- Implement a number of system behaviors

- **LSCC**(Lifecycle system chaincode)
  - handles lifecycle requests of application chaincodes
- **CSCC**(Configuration system chaincode)
  - handles channel configuration on the peer side
- **QSCC**(Query system chaincode)
  - provides ledger query APIs such as getting blocks and transactions

Part 1.4

Gossip Protocol

Functions of Gossip Protocol

- Manages peer discovery and channel membership
- Disseminates ledger data across all peers on a channel
- Allowing peer-to-peer state transfer update of ledger data for new peers.

Leader Peer & Anchor Peer

- Leader Peer
  - Connect to the ordering service and pull out new blocks
  - Distribute transactions to the other committing peers in the organization
  - Allow one or more leader peers in an organization
  - Leader Peer election
    - Static
    - Dynamic
- Anchor Peer
  - Used by gossip to make sure peers in different organizations know about each other

Leader Election

- Static
  - A system administrator manually configures a peer in an organization to be the leader
  - Can define one or more peers within an organization as leader peers

```
peer:
    # Gossip related configuration
    gossip: useLeaderElection: false
    orgLeader: true
```

- Dynamic
  - Peers execute a leader election procedure to select one leader in an organization
  - A dynamically elected leader sends heartbeat messages to the rest of the peers as an evidence of liveness

```
peer:
    # Gossip related configuration
    gossip:
        useLeaderElection: true
        orgLeader: false
        election:
            leaderAliveThreshold: 10s
```

Gossip Messaging

- Online peers indicate their availability by continually broadcasting “alive” messages
- Peers maintain channel membership by collecting these alive messages

- Peers receives/handle messages, and forward the received messages automatically as well
- Each peer continually pulls blocks from other peers on the channel, in order to repair its own state if discrepancies are identified

- Peers on one channel cannot message or share information on any other channel

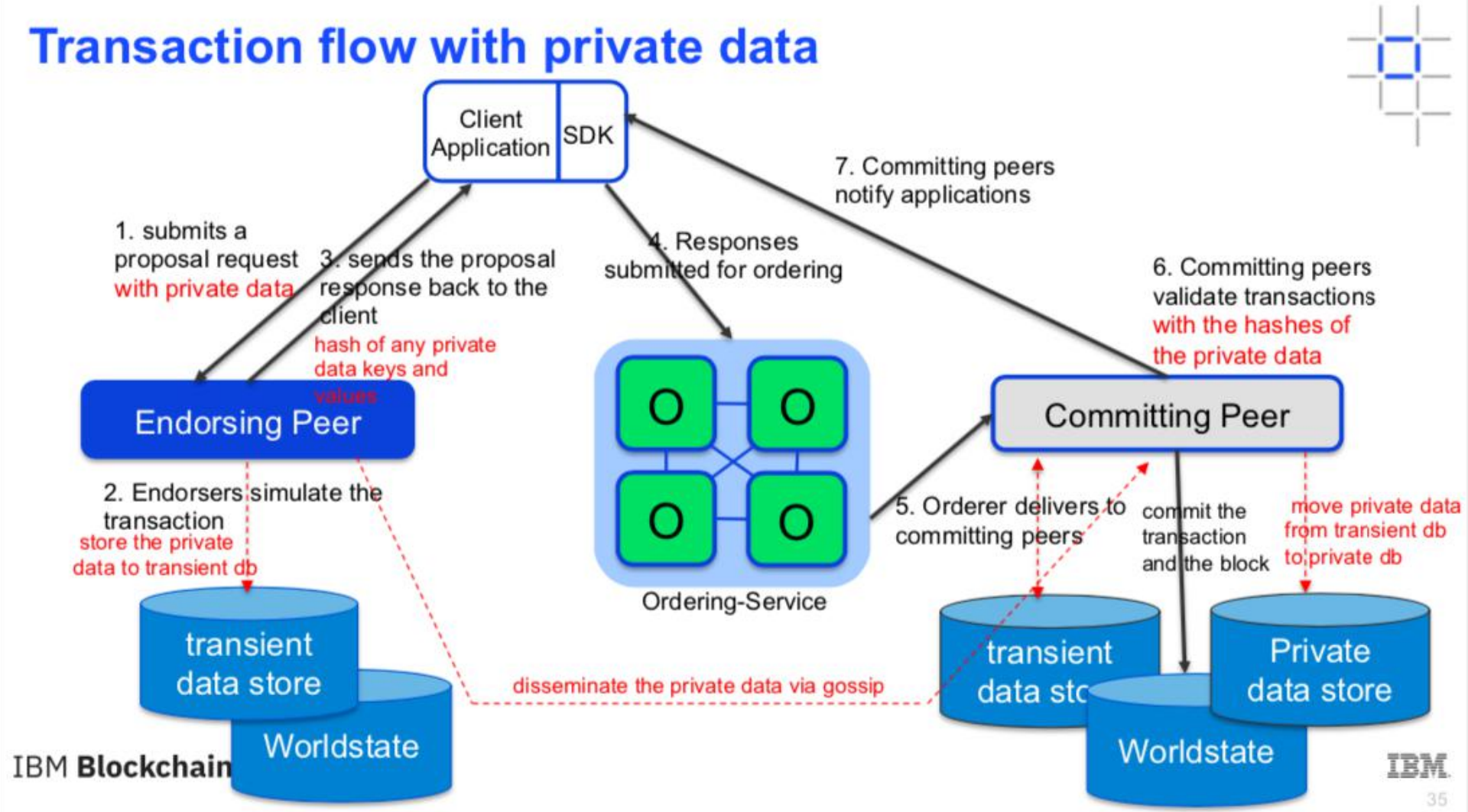
Part 1.5

Private Data

Private Data

- Confidential data that is stored in a private database on each authorized peer
- Private data collection policy to define authorized peers
- Ordering service does not see the private data
- Sent peer-to-peer via gossip protocol

Transaction flow with private data



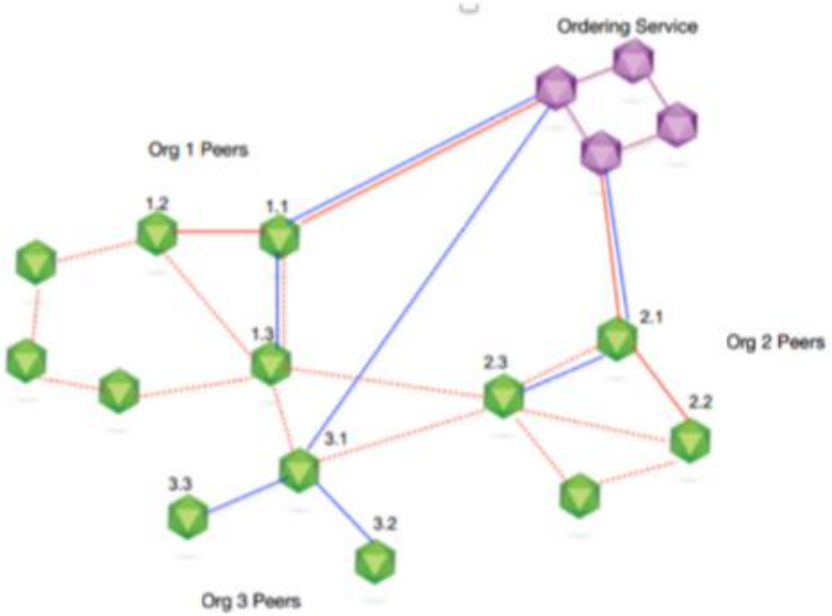
Part

1.6

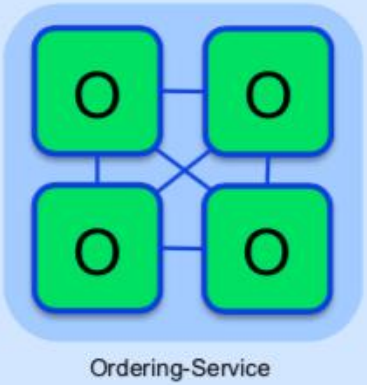
Sample Distributed network deployment

What is a Hyperledger blockchain network?

- Multiple organizations as a consortium to form the network
- Governed by policies agreed by the organizations
- Provide ledger and smart contract service



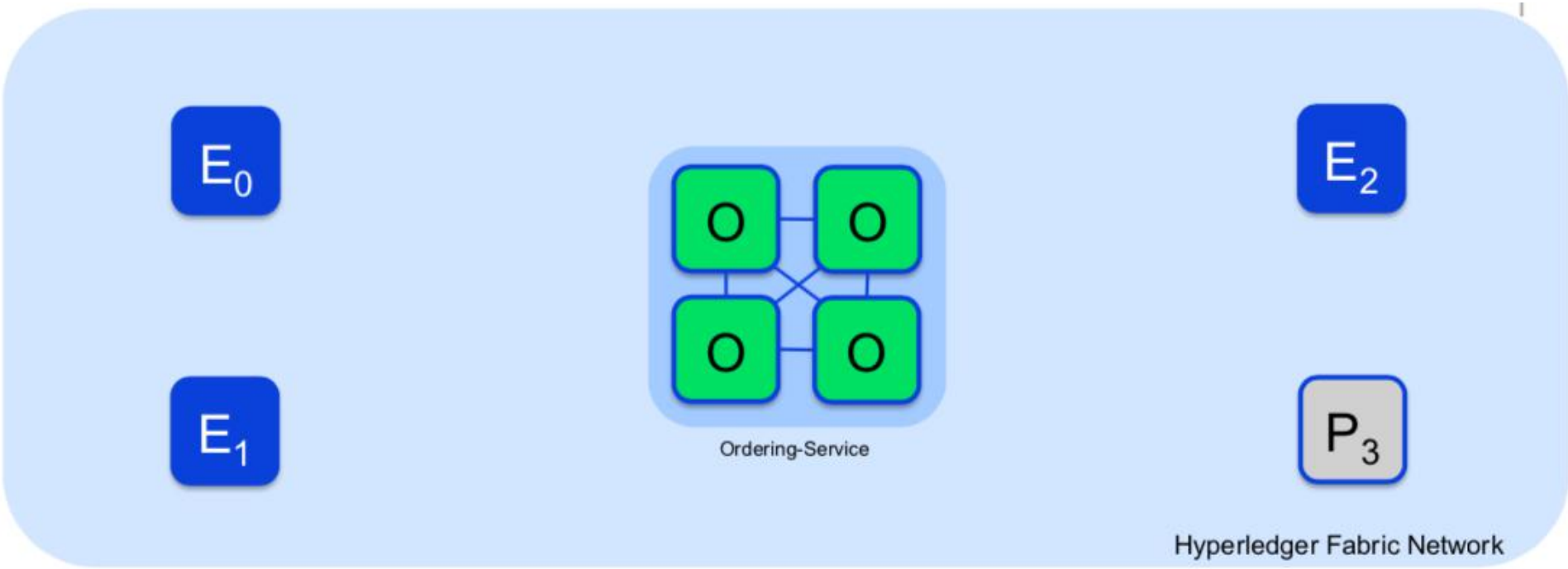
Bootstrap Network (1/6) - Configure & Start Ordering Service



An Ordering Service is configured and started for the network:

$ docker-compose [-f orderer.yml] ...

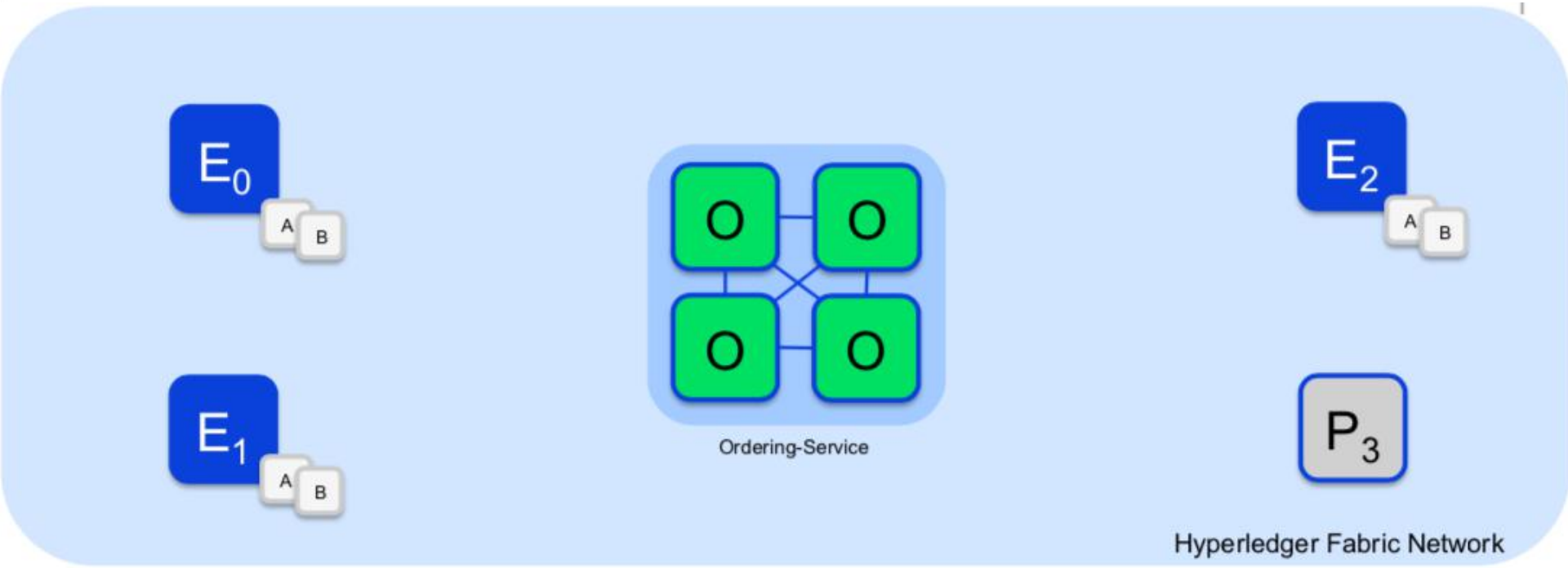
Bootstrap Network (2/6) - Configure and Start Peer Nodes



A peer is configured and started for each Endorser or Committer in the network:

$ peer node start ...

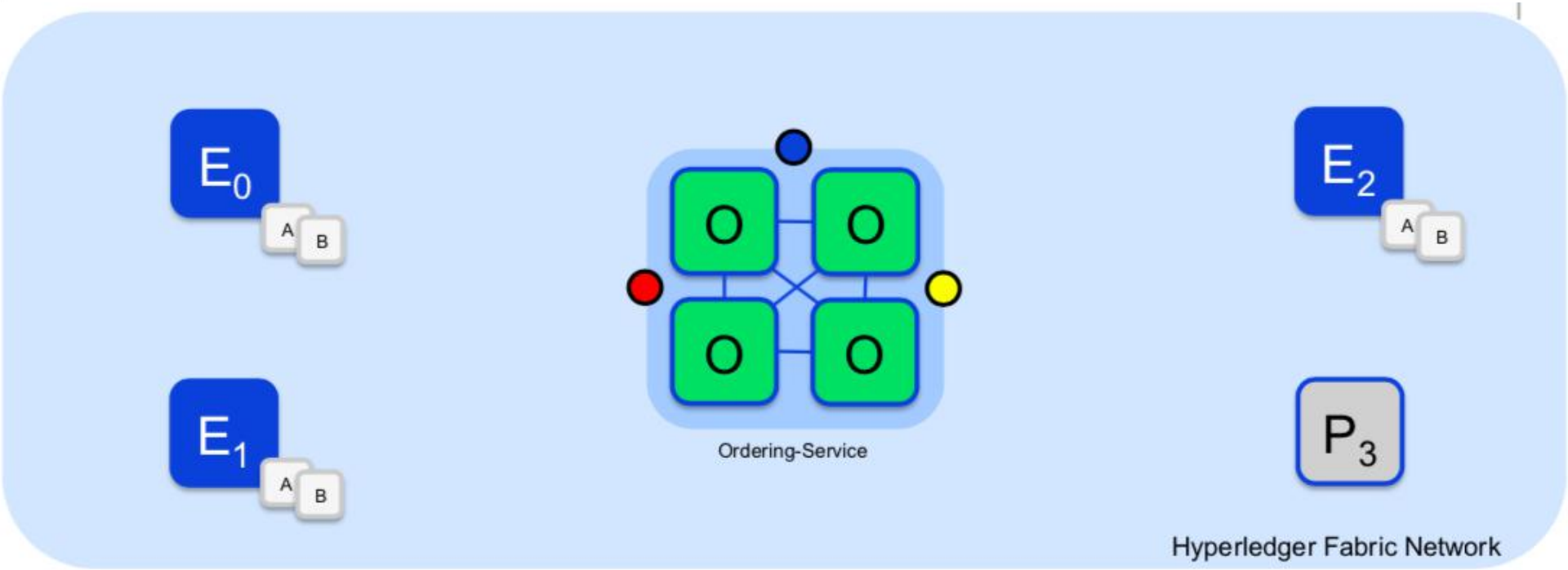
Bootstrap Network (3/6) - Install Chaincode



Chaincode is installed onto each Endorsing Peer that needs to execute it:

$ peer chaincode install ...

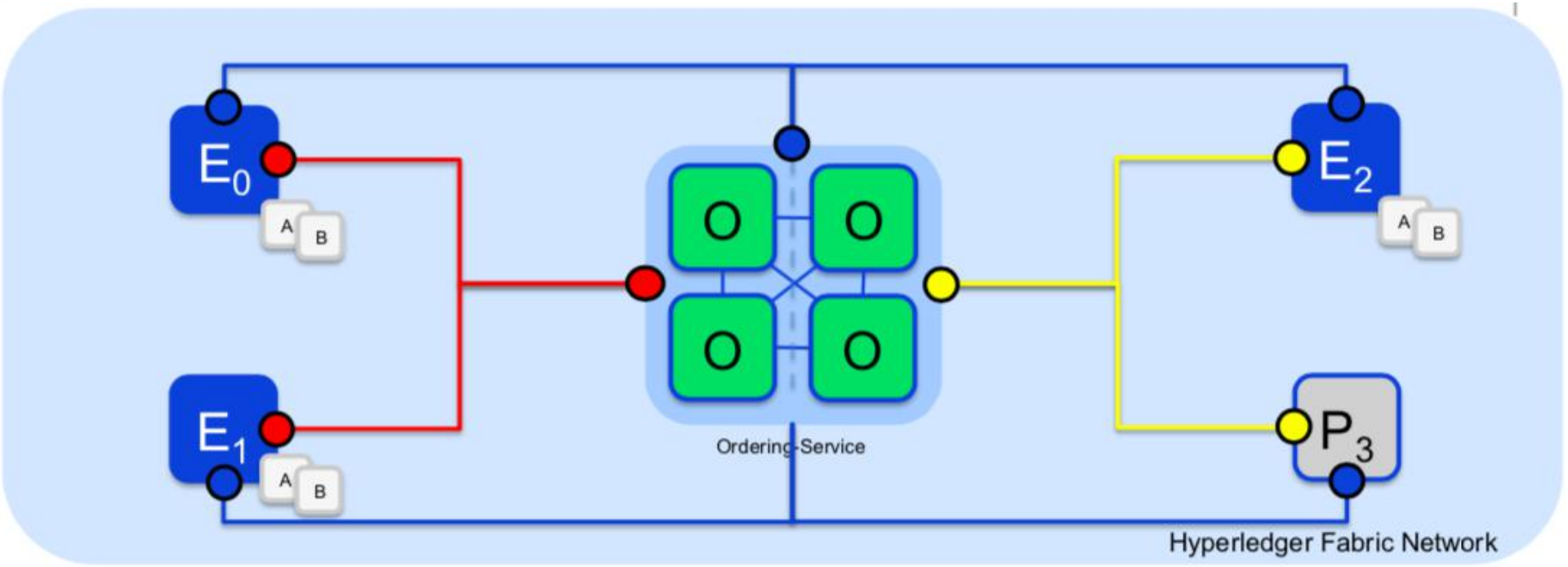
Bootstrap Network (4/6) – Create Channels



Channels are created on the ordering service:

$ peer channel create –o [orderer] ...

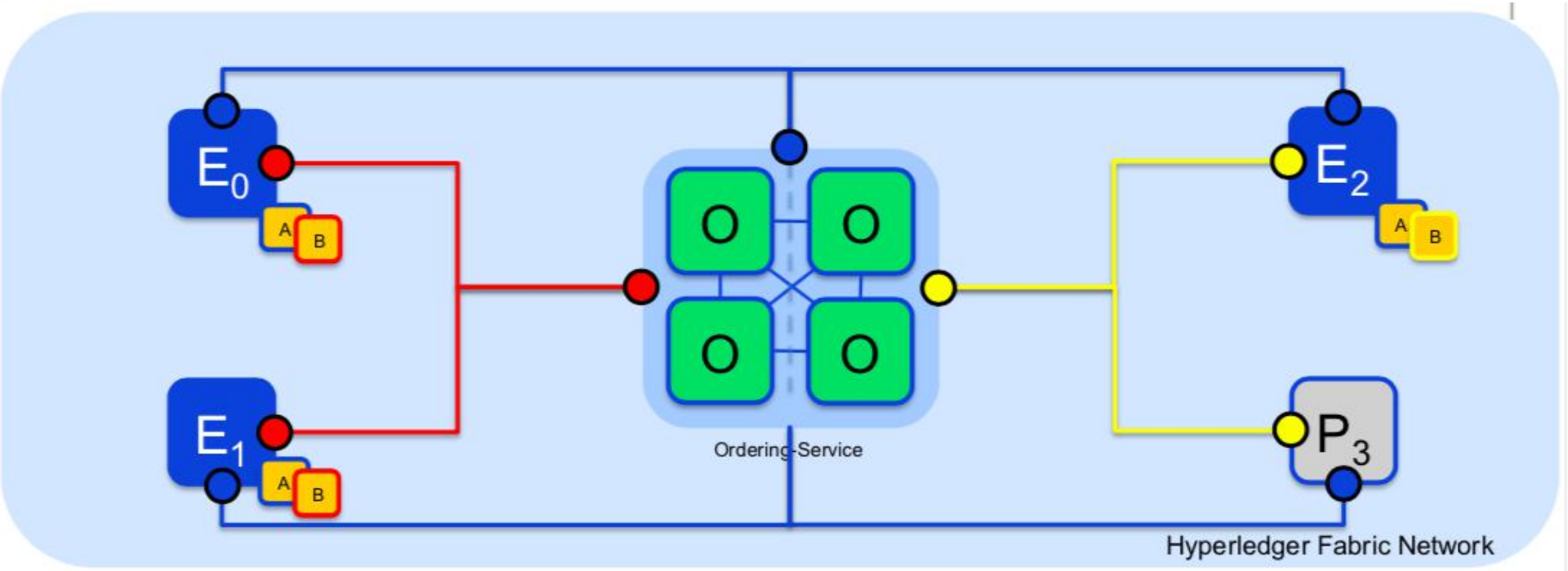
Bootstrap Network (5/6) – Join Channels



Peers that are permissioned can then join the channels they want to transact on:

$ peer channel join ...

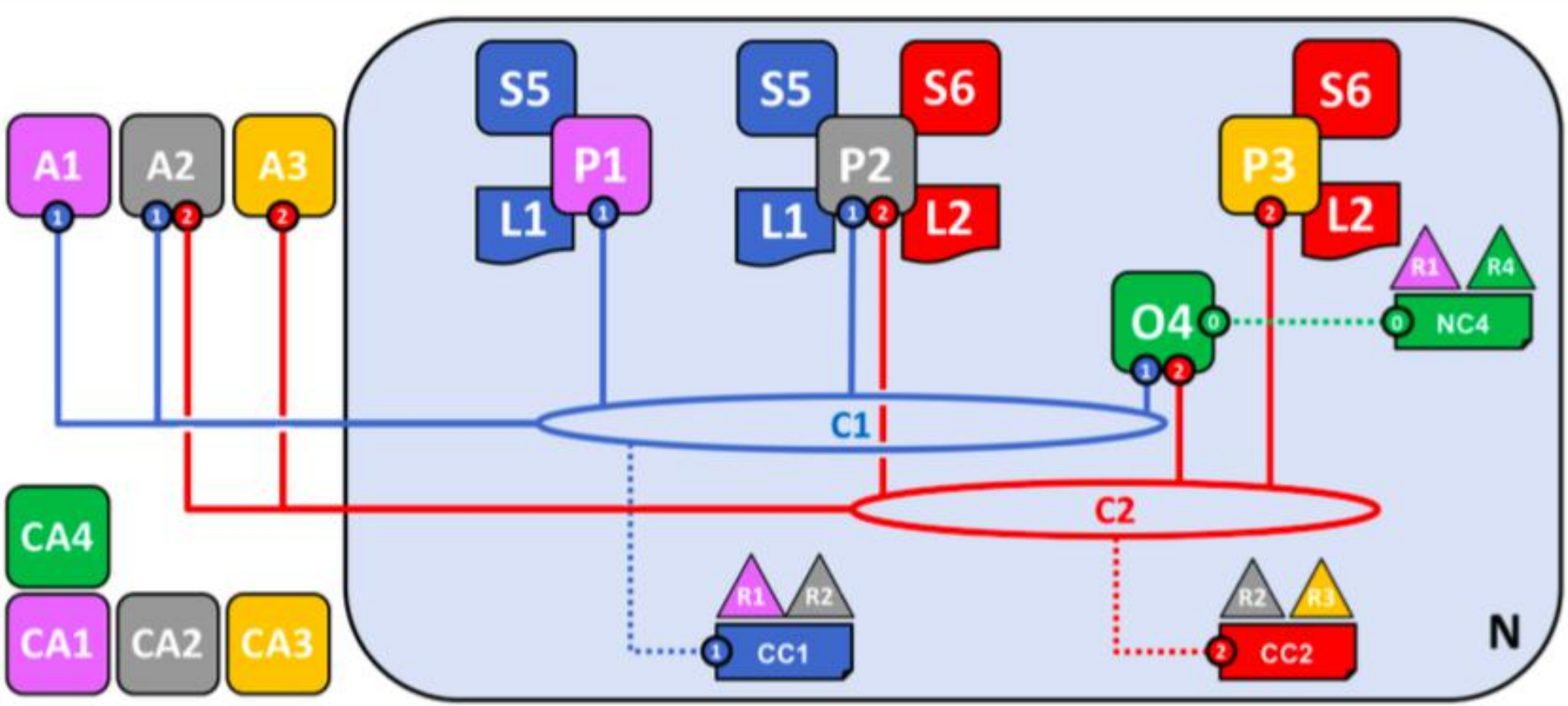
Bootstrap Network (6/6) – Instantiate Chaincode



Peers finally instantiate the Chaincode on the channels they want to transact on:

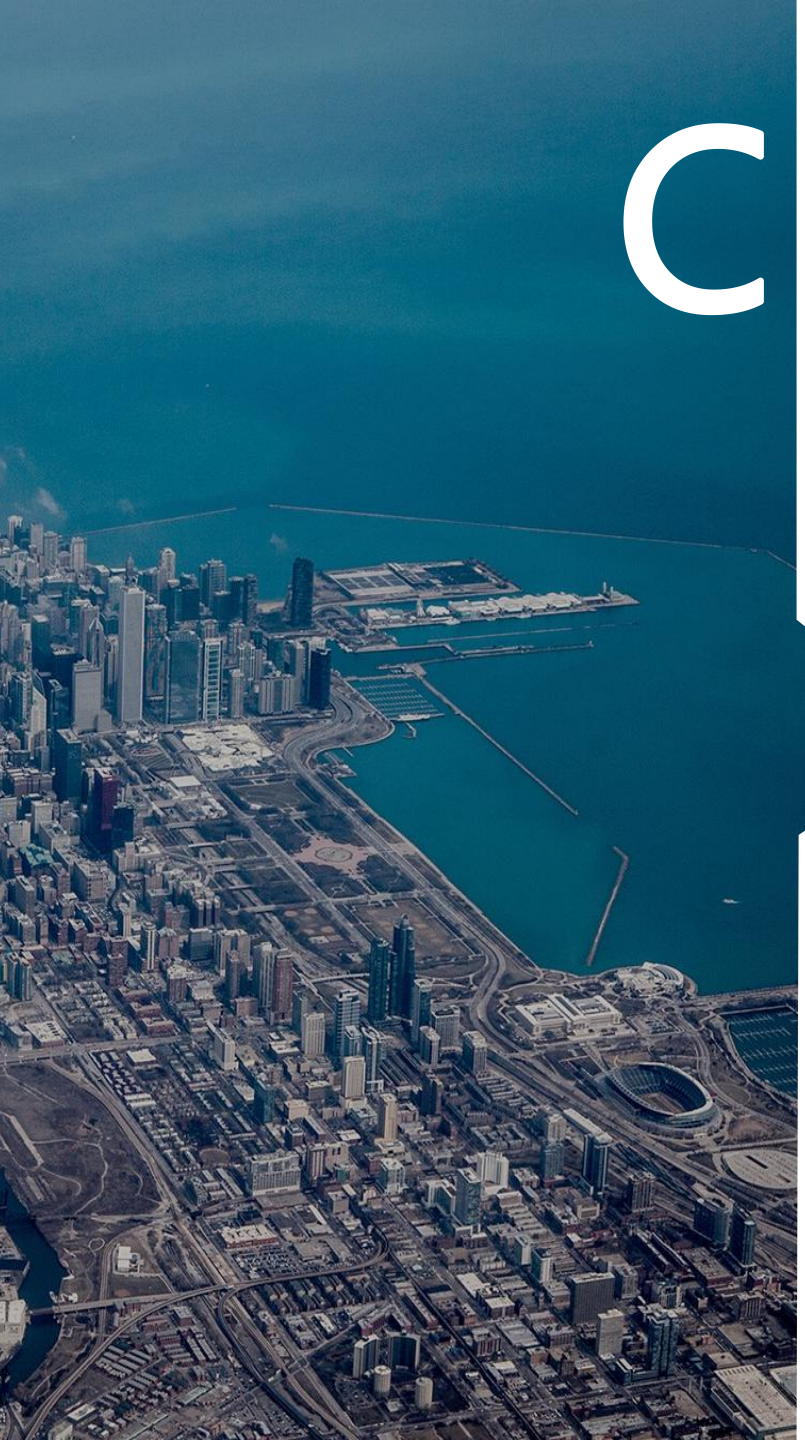
$ peer chaincode instantiate ... –P 'policy'

Sample network with multiple orgs/channels



Part Two

Fabric Orderer

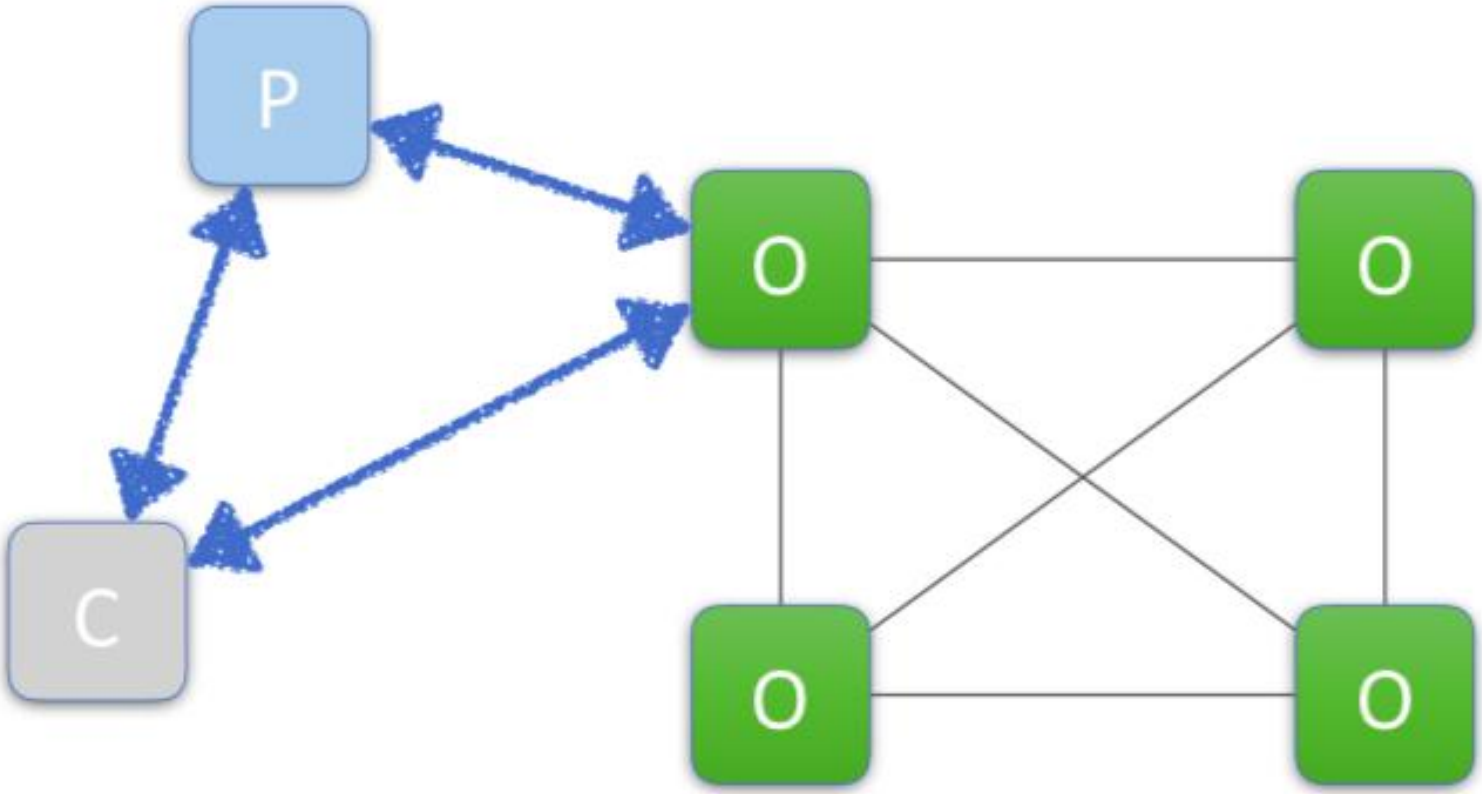


Contents

Part 2.1

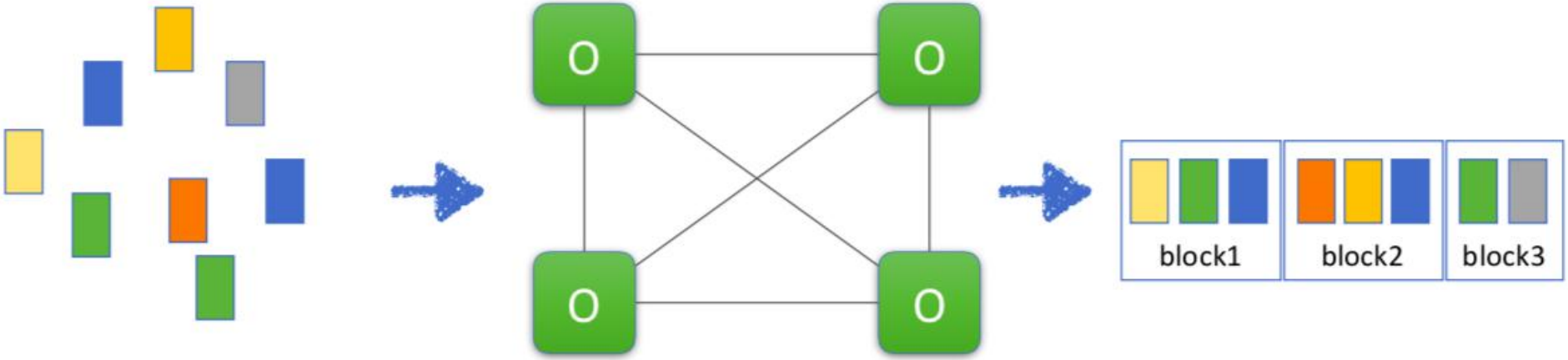
Atomic Broadcast (Total Order)

Atomic Broadcast （Total Order）

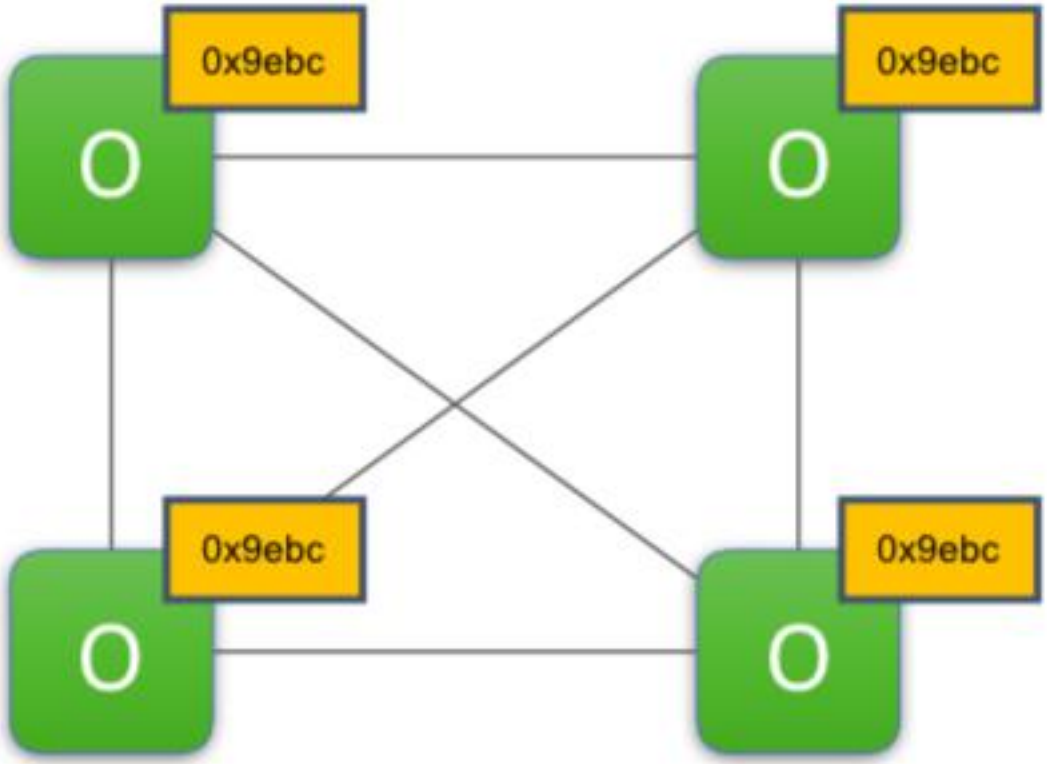


Execute-Order-Validate

Atomic Broadcast （Total Order）



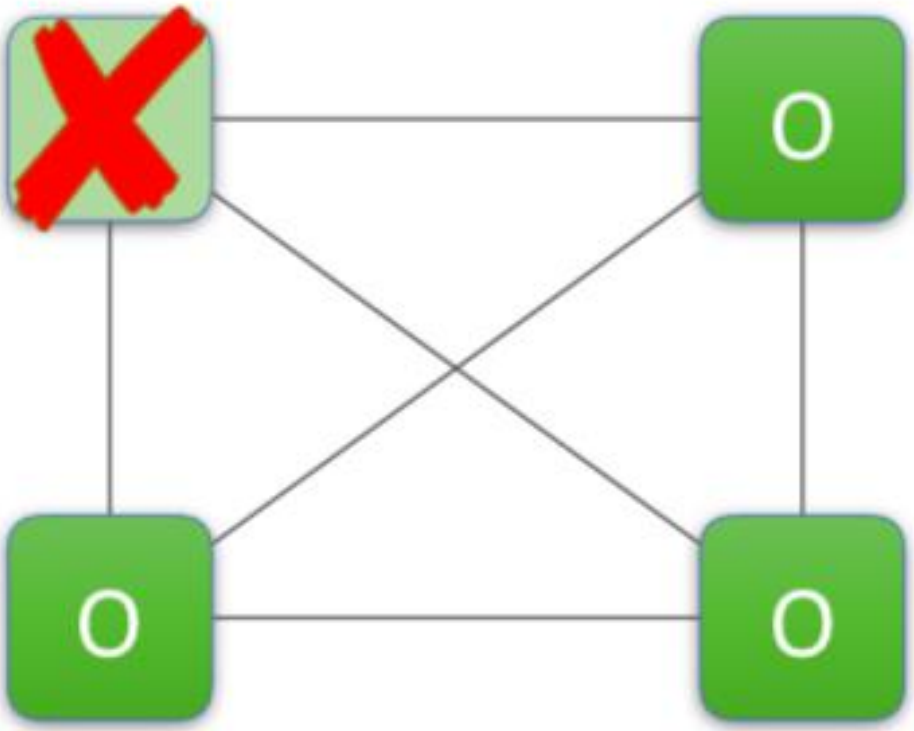
Atomic Broadcast （Total Order）



Produce identical blocks

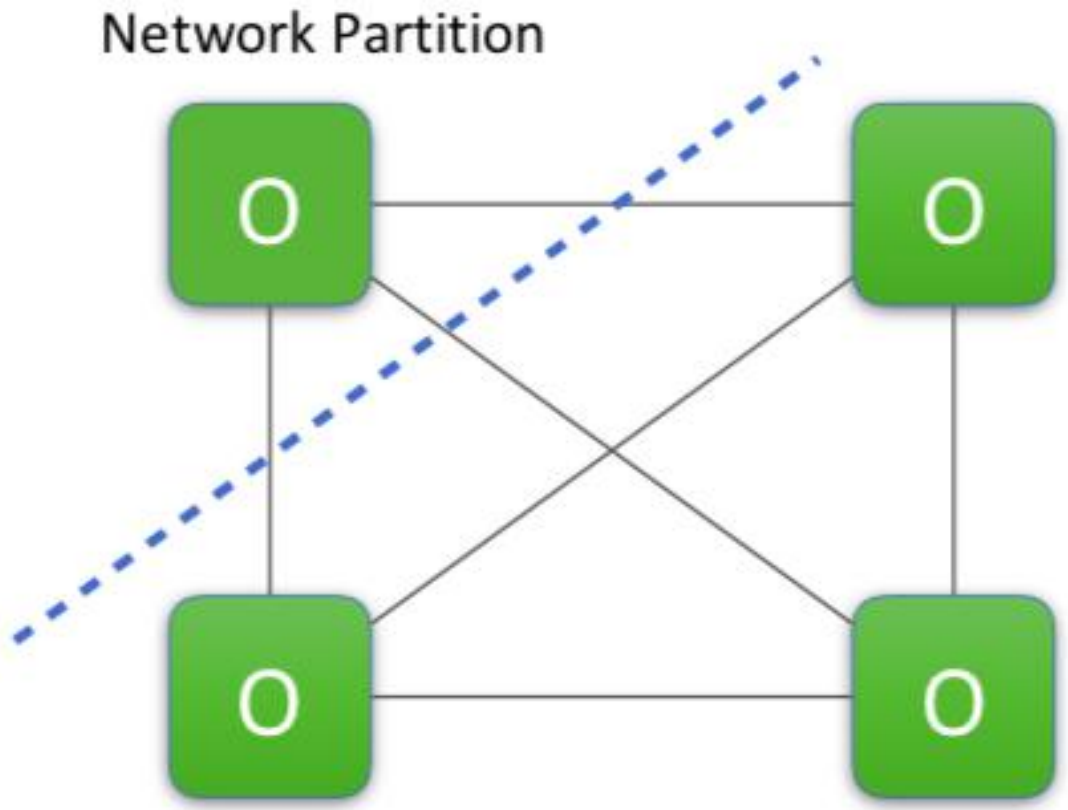
Atomic Broadcast （Total Order）

Fail-Recovery



Crash Fault Tolerance

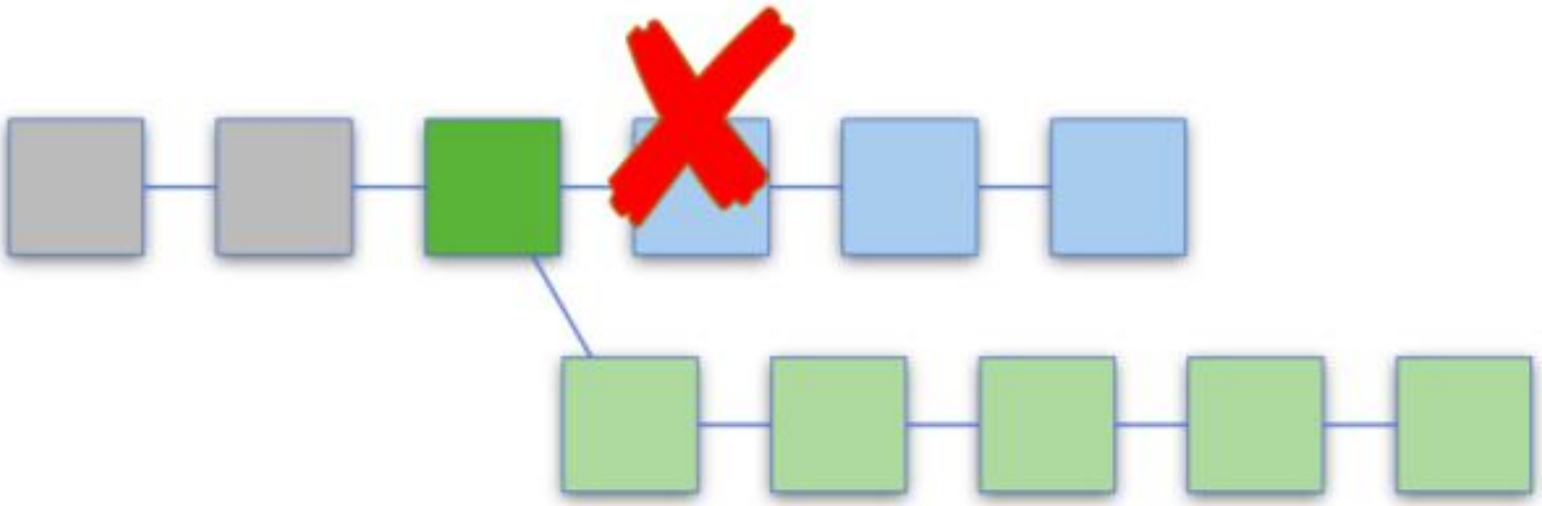
Atomic Broadcast （Total Order）



Crash Fault Tolerance

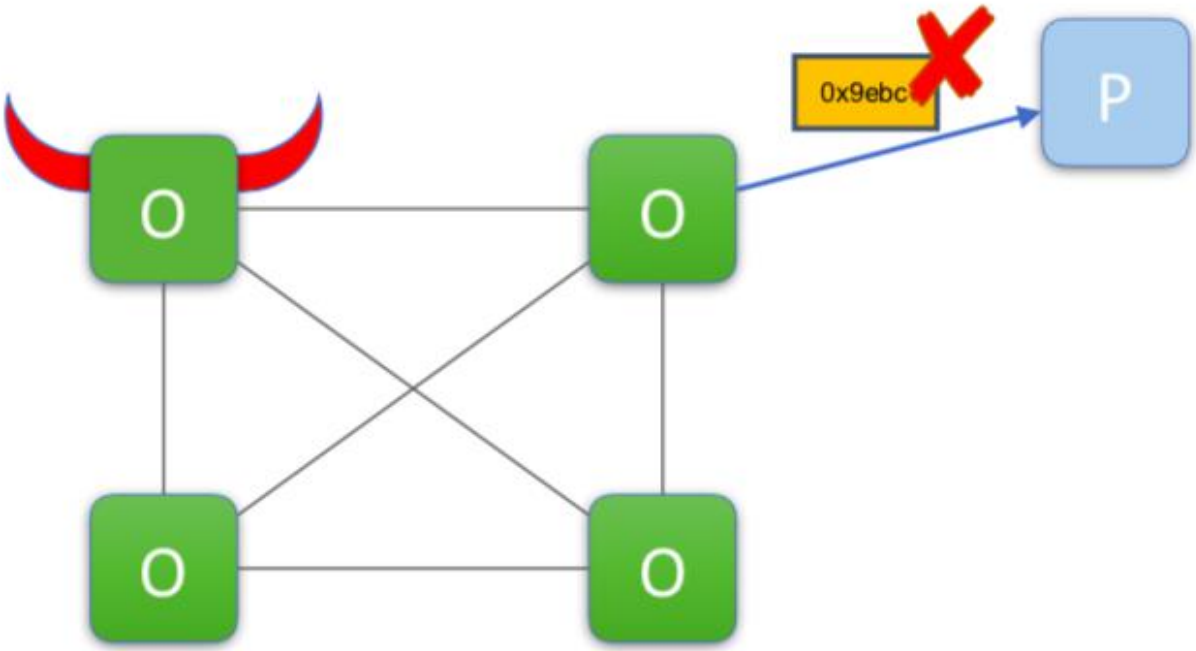
Atomic Broadcast (Total Order)

Strong Consistency (Finality, no temporary fork)



Byzantine Fault Tolerance

CFT Orderer != CFT Fabric



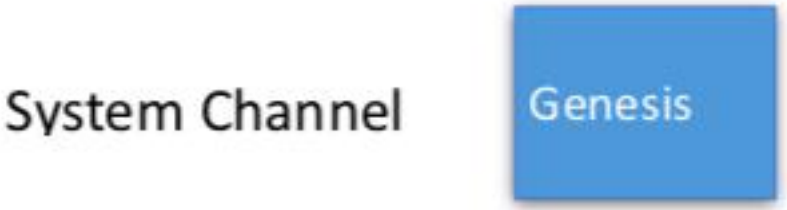
Block Cutting

- BatchSize
  - MaxMessageCount
  - AbsoluteMaxBytes
  - PreferredMaxBytes
- BatchTimeout
  - Timeout

Part 2.3

Channels

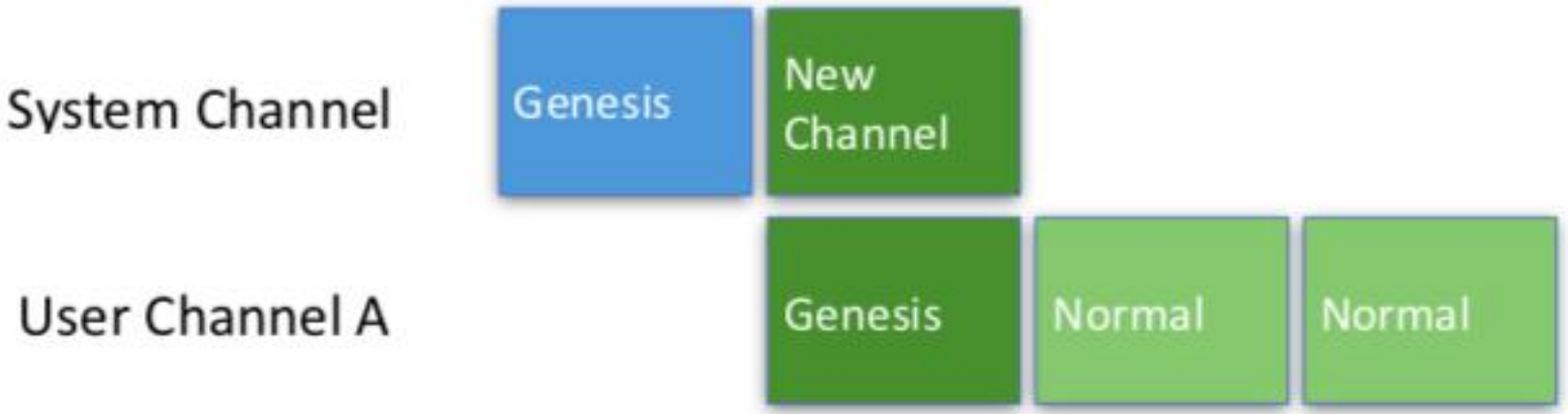
Channels



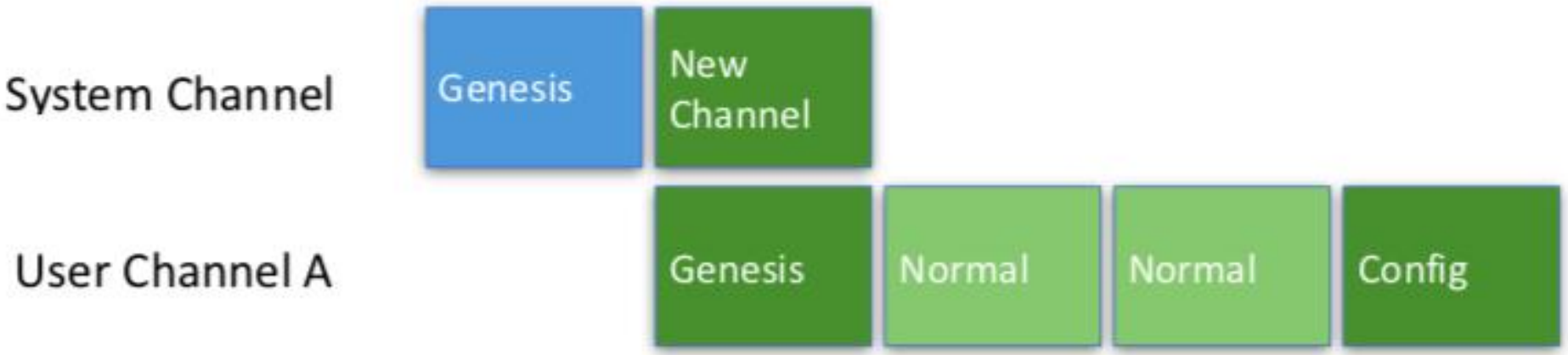
Channels

Channels

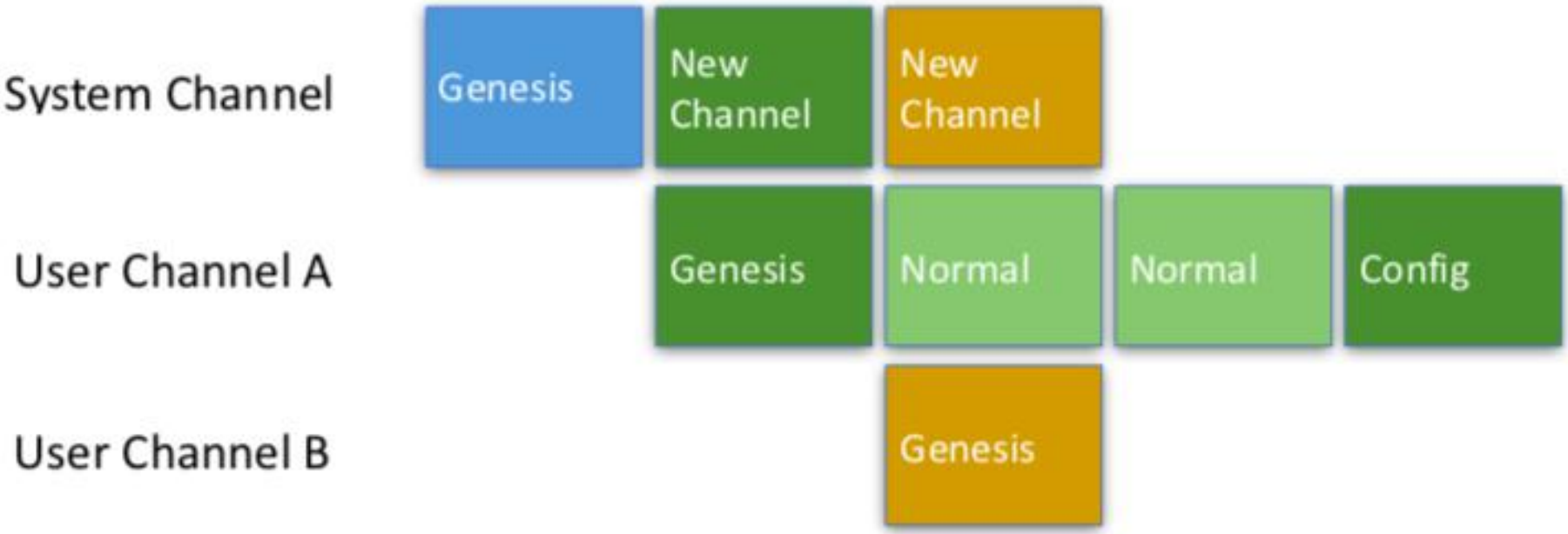
Channels


System Channel
Genesis
New Channel
User Channel A
Genesis
Normal
Normal

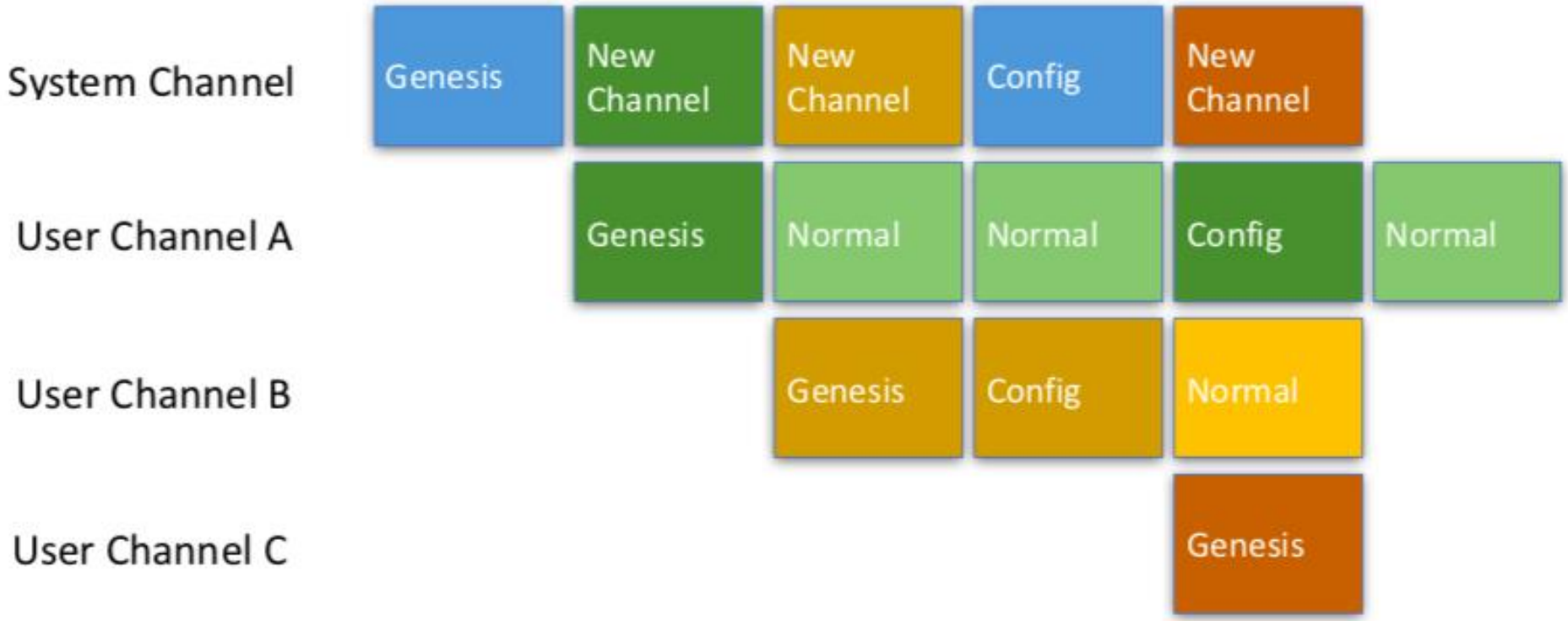
Channels


System Channel
Genesis
New Channel
User Channel A
Genesis
Normal
Normal
Config

Channels


System Channel
Genesis
New Channel
New Channel
User Channel A
Genesis
Normal
Normal
Config
User Channel B
Genesis

Channels


System Channel
Genesis
New Channel
New Channel
Config
New Channel
User Channel A
Genesis
Normal
Normal
Config
Normal
User Channel B
Genesis
Config
Normal
User Channel C
Genesis

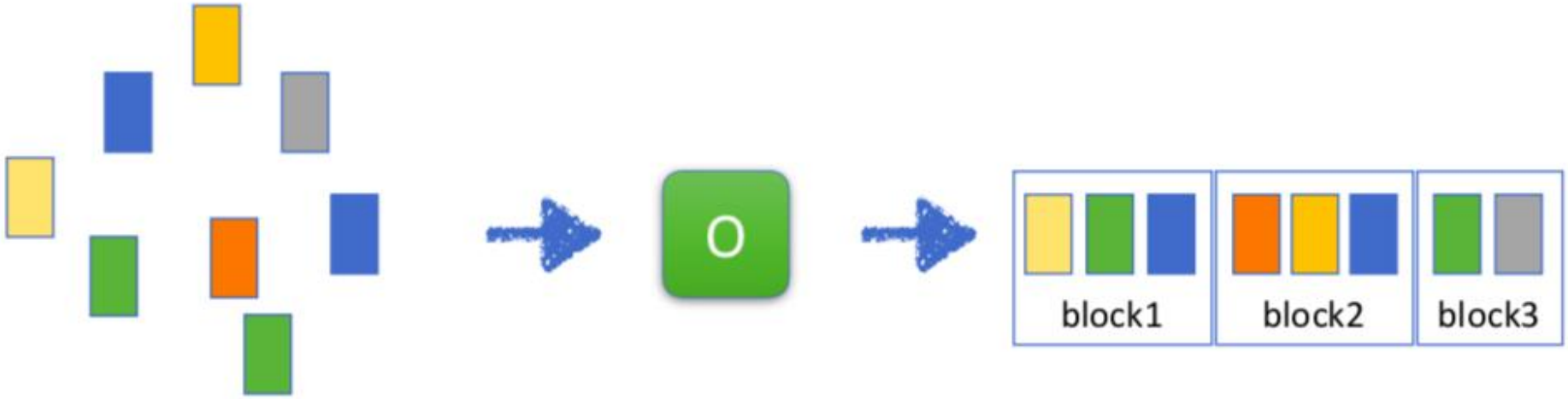
Part 2.3

Solo/Kafka/Raft

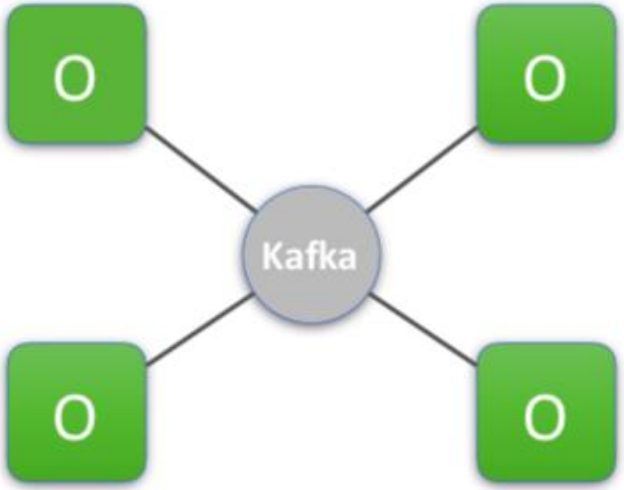
Consensus

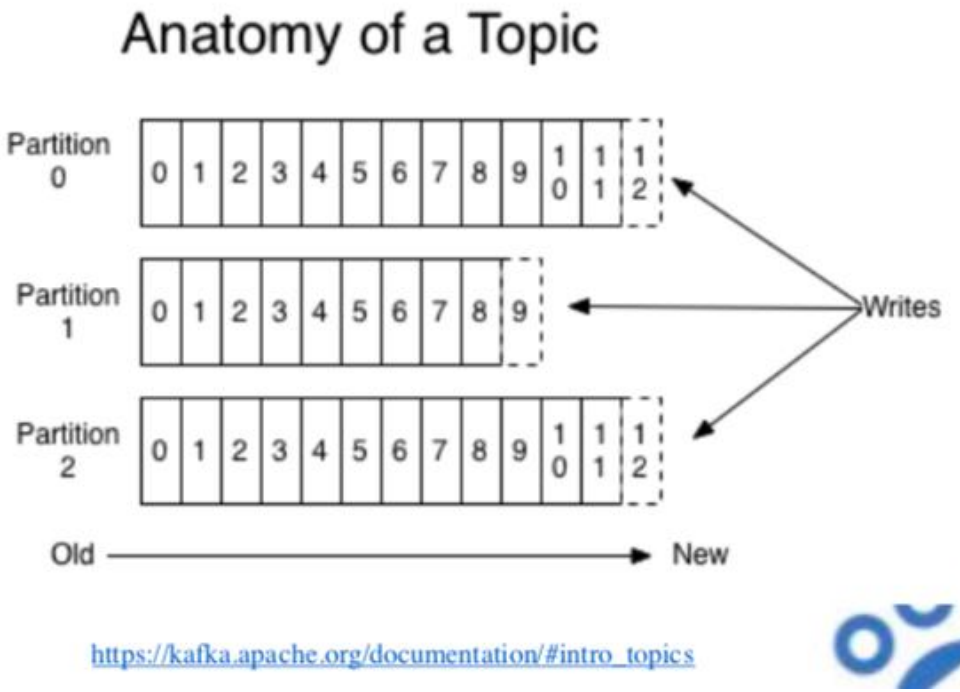
- Solo
- Kafka
- Raft
- BFT

Solo



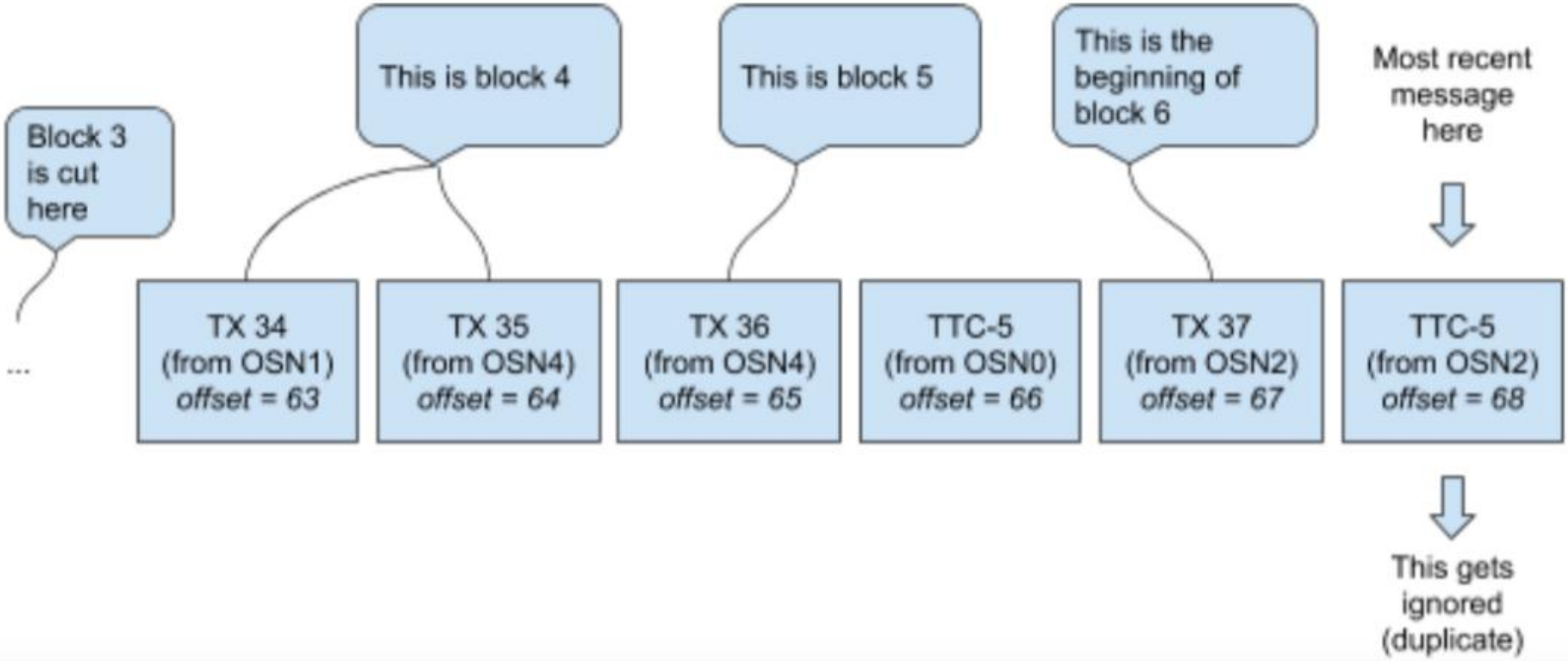
Kafka





https://kafka.apache.org/documentation/#intro_topics

Kafka

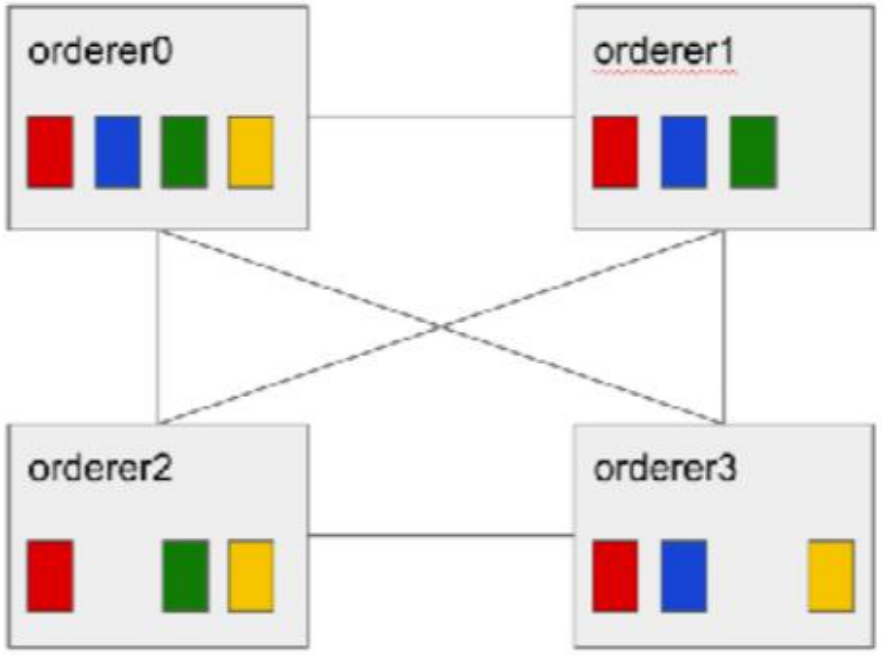

Block 3 is cut here
...
This is block 4
This is block 5
This is the beginning of block 6
Most recent message here
TX 34 (from OSN1) offset = 63
TX 35 (from OSN4) offset = 64
TX 36 (from OSN4) offset = 65
TTC-5 (from OSN0) offset = 66
TX 37 (from OSN2) offset = 67
TTC-5 (from OSN2) offset = 68
This gets ignored (duplicate)

Raft

- Based on Etcd/raft library
- No Kafka/Zookeeper dependency (easier to operate)
- Necessary communication layer built for future use
- Each channel runs its own Raft instance
- A channel can run on a subset of orderers
- All orderers should belong to system channel
- Nodes are identified by TLS cert
- Support Migrating from Kafka to Raft

Raft

Each channel has its own raft cluster

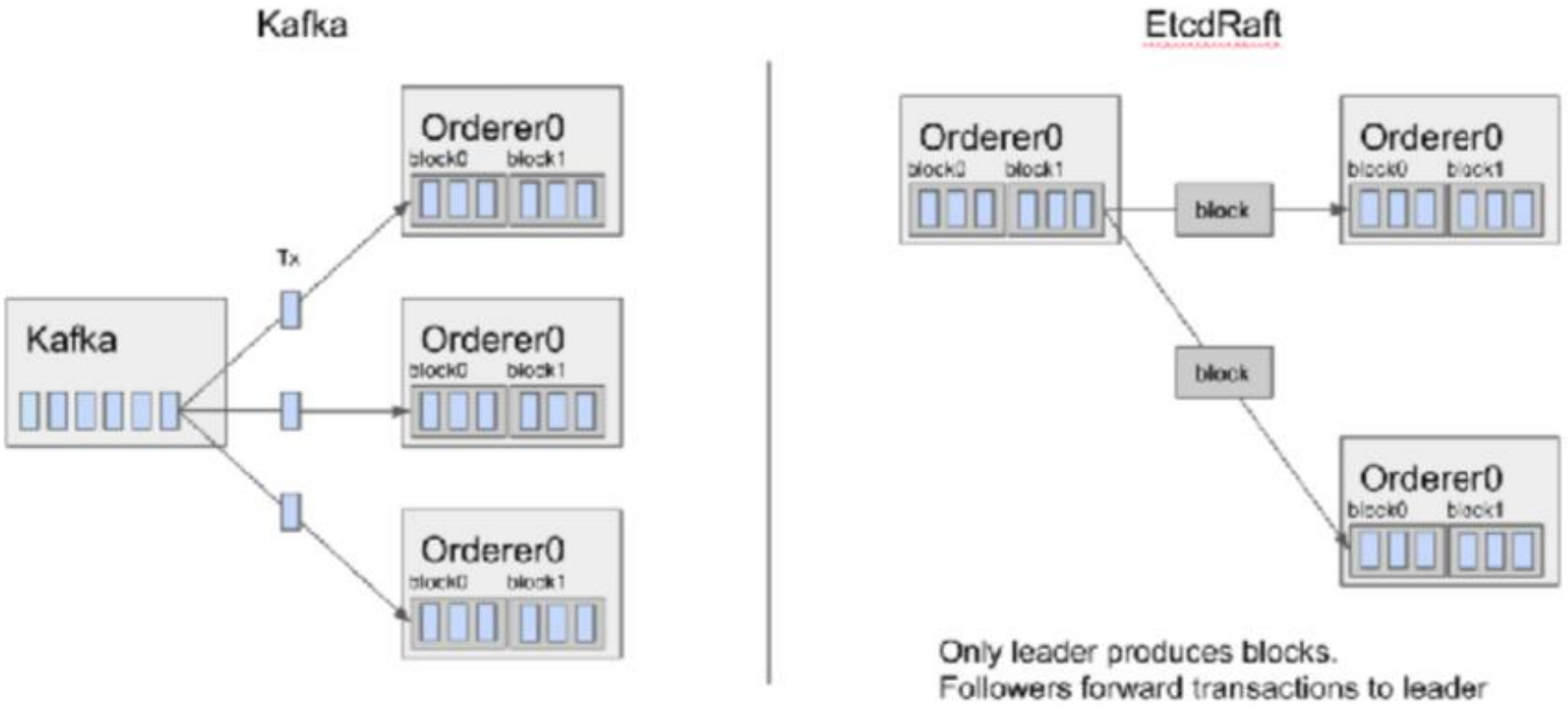


* system channel runs on every orderer

system channel
channel A
channel B
channel C

Raft

Consent on blocks, instead of envelopes



Thanks!