

IBM开源技术微讲堂

Hyperledger Fabric v1.4 LTS

第四讲 《Peer解析》

每周四晚8点直播



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 功能介绍 架构概览
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 **“replay”**
获取往期视频地址

公众号中发送 **“报名”**，
即有机会参加Fabric线下训练营

自我介绍

李春玲

软件工程师， IBM开发实验室

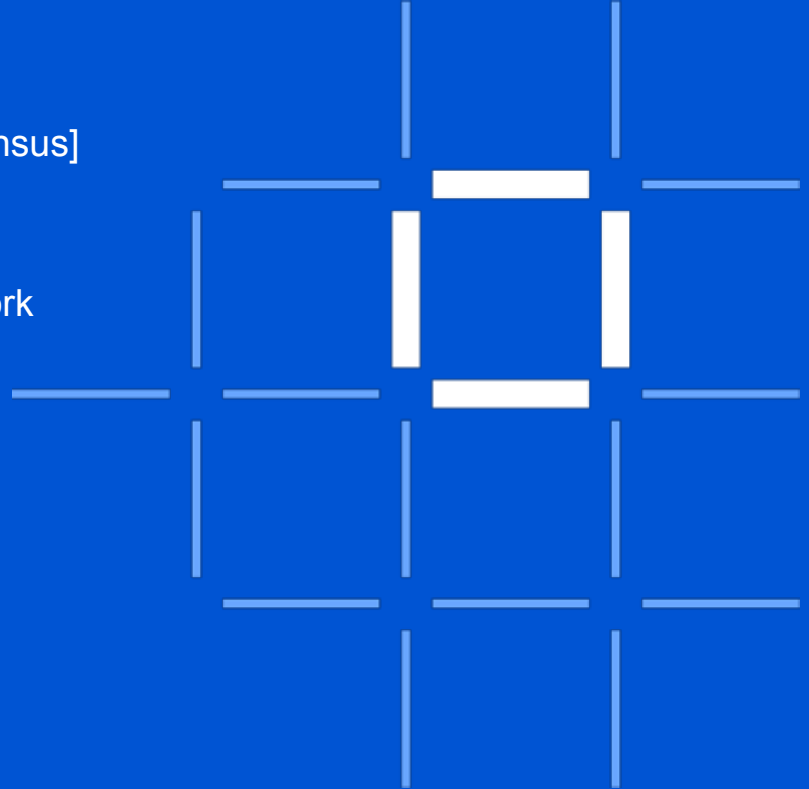
IBM Blockchain Platform 开发团队

Hyperledger开源社区爱好者



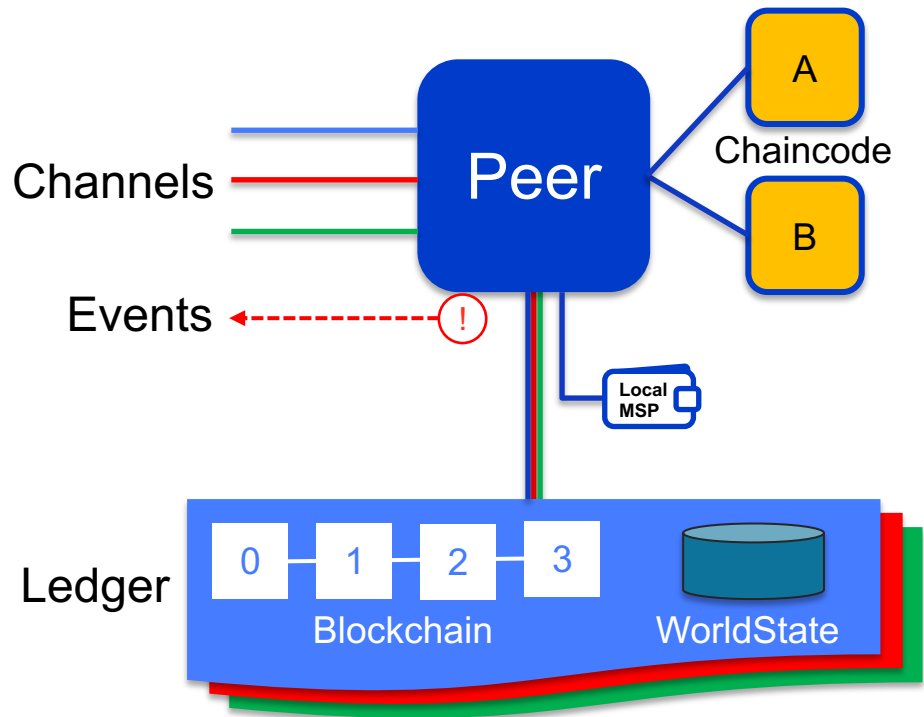
Technical Deep Dive

- [Fabric Peer in Network Consensus]
- Fabric Ledger and State DB
- Smart Contract in Fabric Network
- Gossip Protocol
- Private data to protect privacy
- Sample Distributed network deployment

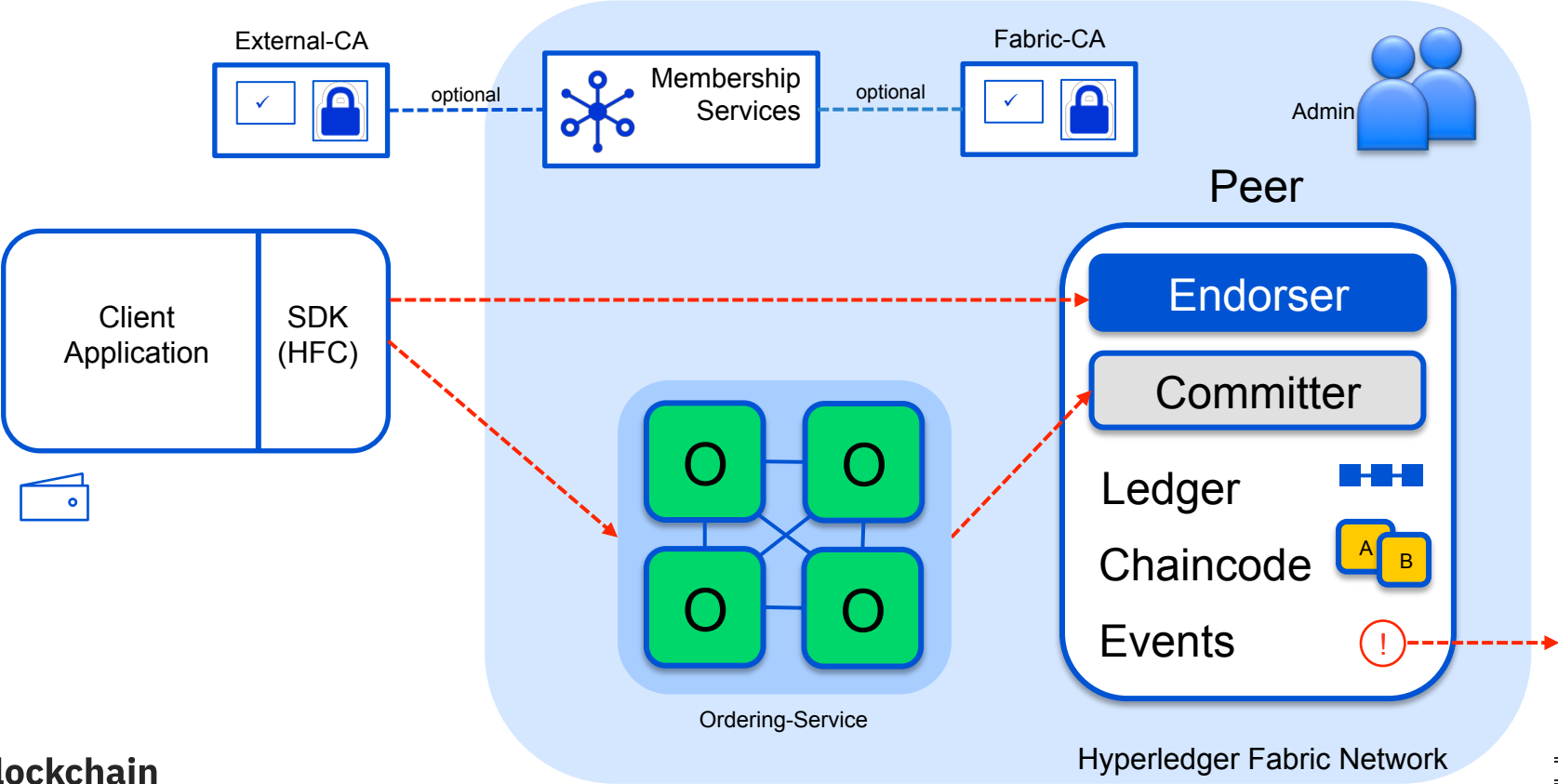
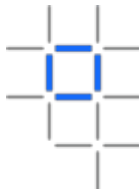


Fabric Peer

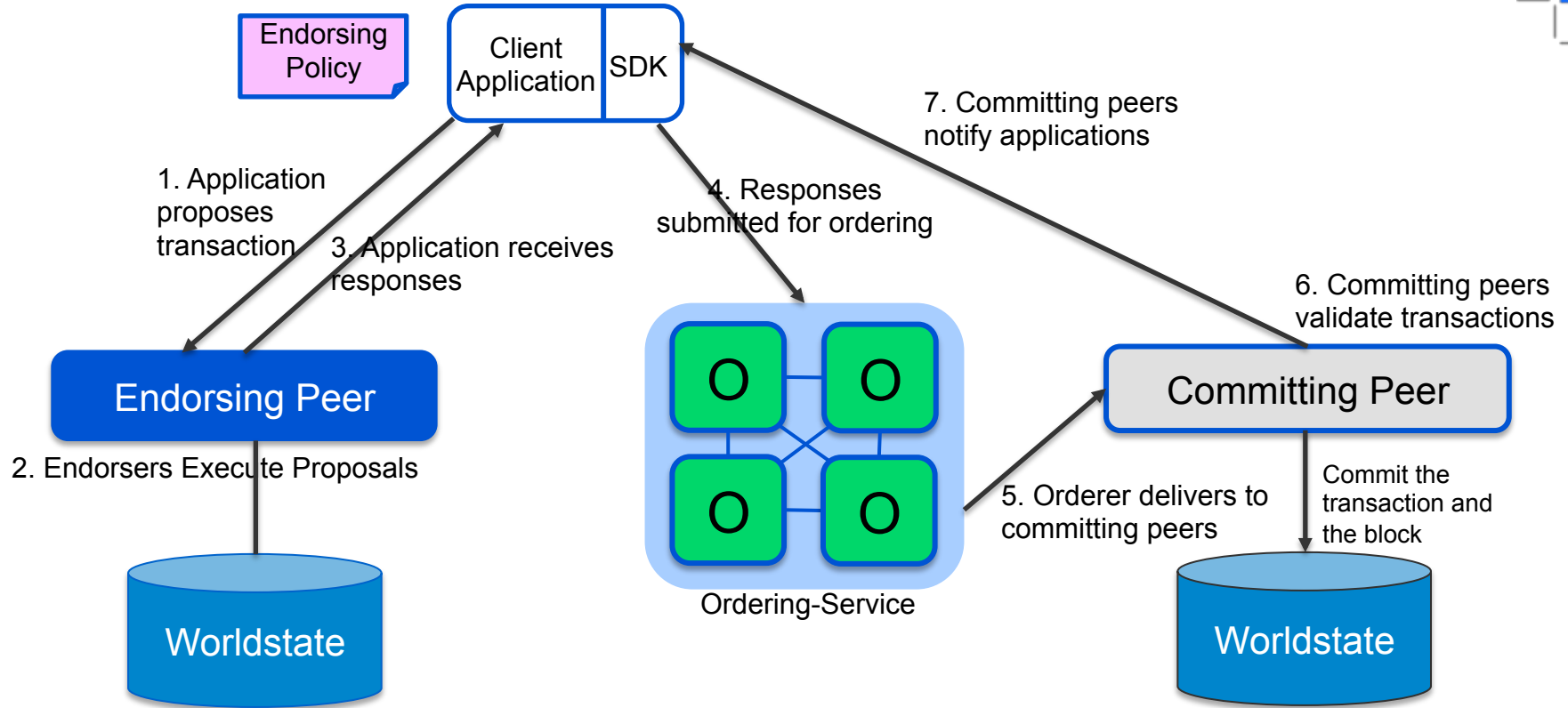
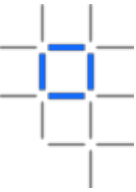
- A blockchain network is comprised primarily of a set of *peer nodes*
- Host *ledgers* and *smart contracts*
- Each peer:
 - Connects to one or more *channels*
 - Maintains one or more *ledgers* per channel
 - Maintains *installed chaincode*
 - Manages *runtime docker containers* for *instantiated chaincode*
 - Has a local MSP (Membership Services Provider) that provides *crypto material*
 - *Emits events* to the client application



Hyperledger Fabric V1.x Architecture

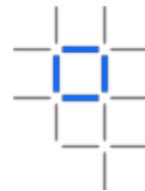


Hyperledger Fabric V1.x Transaction Flow



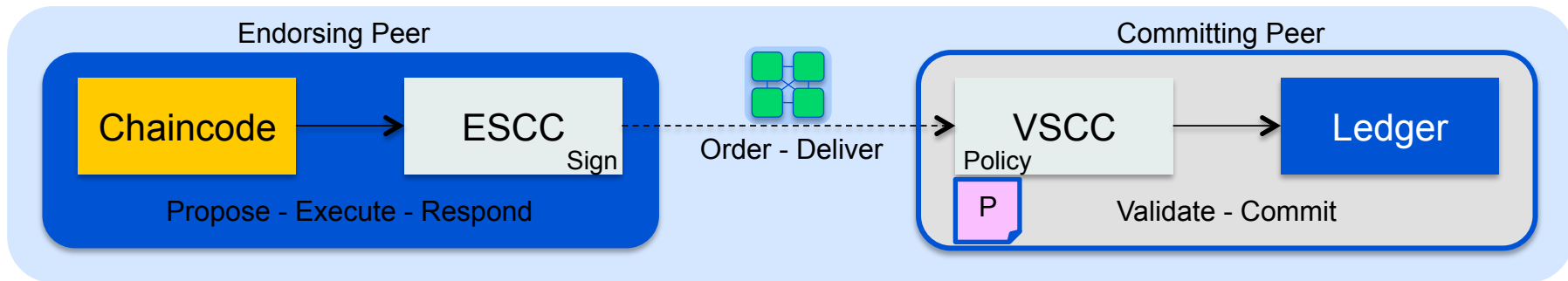
Endorsing Peer: endorses transactions by receiving a transaction proposal and responds by granting or denying endorsement
Committing Peer: maintains ledger and state

Endorsement Policies

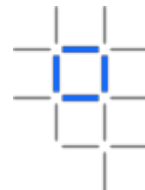


An endorsement policy **describes the conditions** by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

- 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 Policy 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...])**

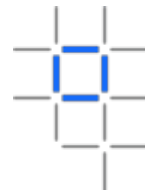
Where **EXPR** is either AND, OR or OutOf and **E** is either a principal or nested EXPR

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 Policy Examples



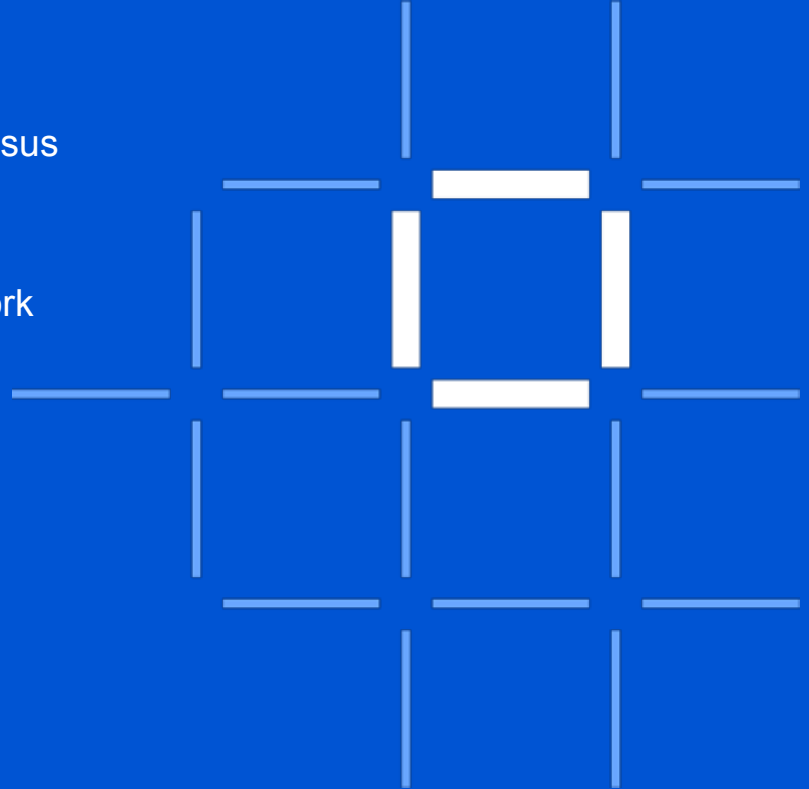
Examples of policies:

- 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'))`

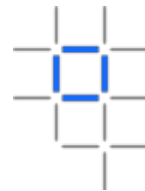


Technical Deep Dive

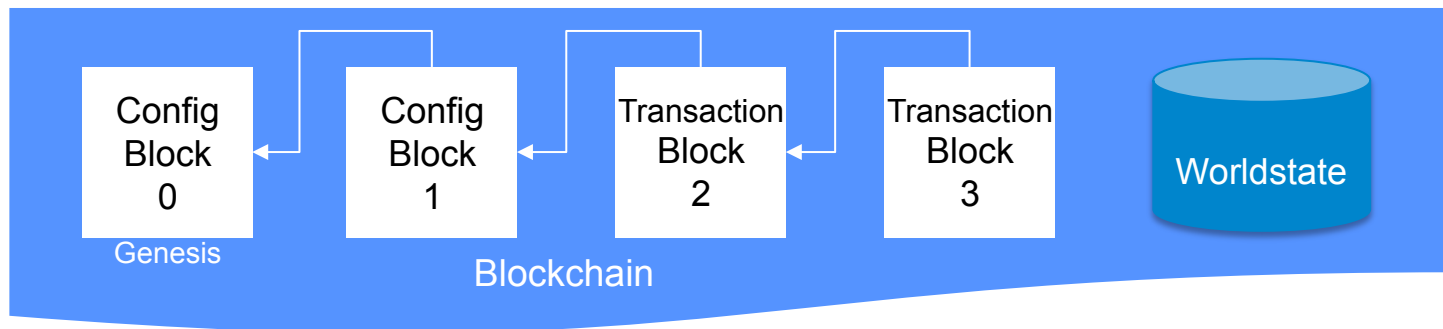
- Fabric Peer in Network Consensus
- [Fabric Ledger and State DB]
- Smart Contract in Fabric Network
- Gossip Protocol
- Private data to protect privacy
- Sample Distributed network deployment



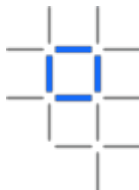
Fabric Ledger



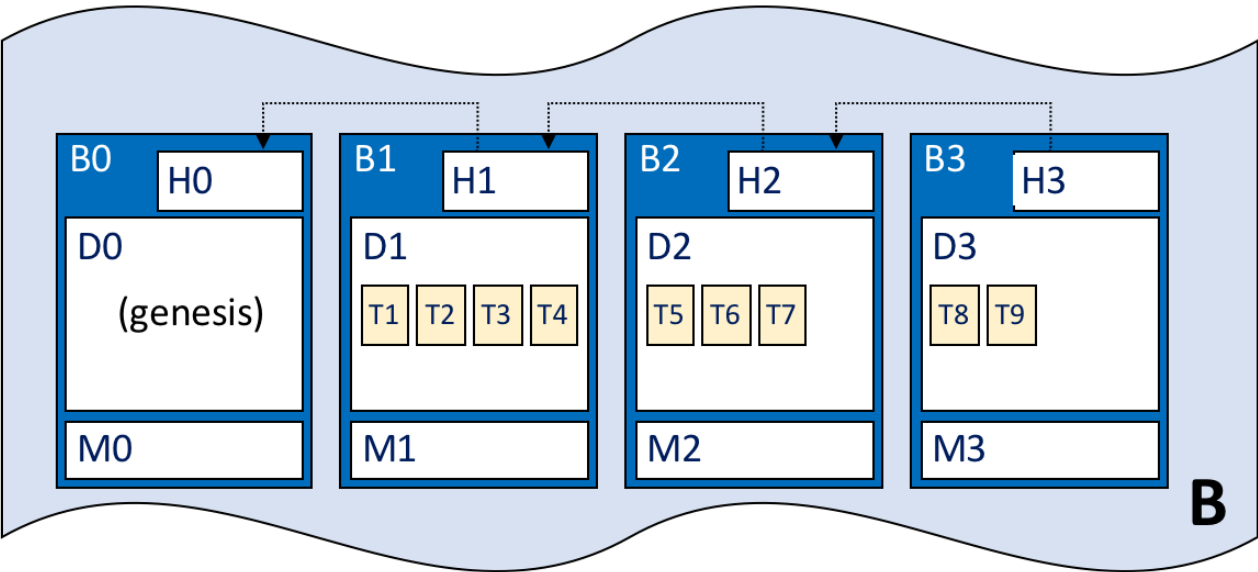
- The **Fabric ledger** is sequenced, tamper-resistant record of all **state** transitions
- Blockchain
 - Channel configurations
 - Immutable, sequenced record in blocks
- World state
 - Maintain current state



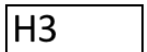


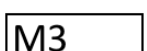
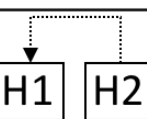


Block chain

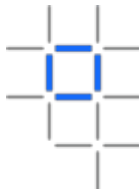


An **historical record** of the facts about how these objects arrived at their current states.

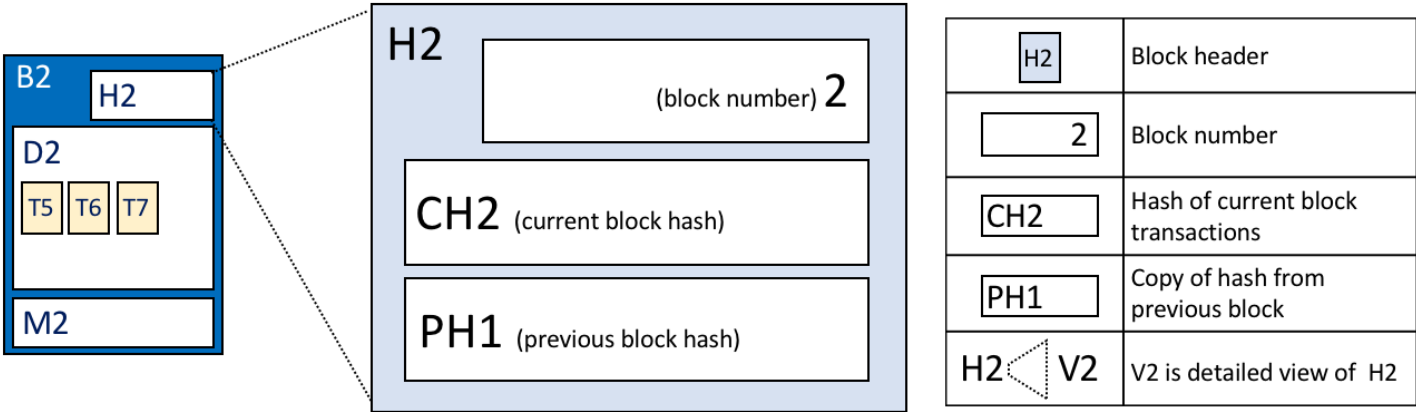


 B	Blockchain
 B1	Block
 H3	Block header
 D1	Block data
 T5	Transaction
 M3	Block metadata
	H2 is chained to H1

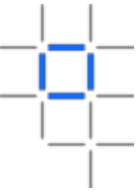
Block(s)



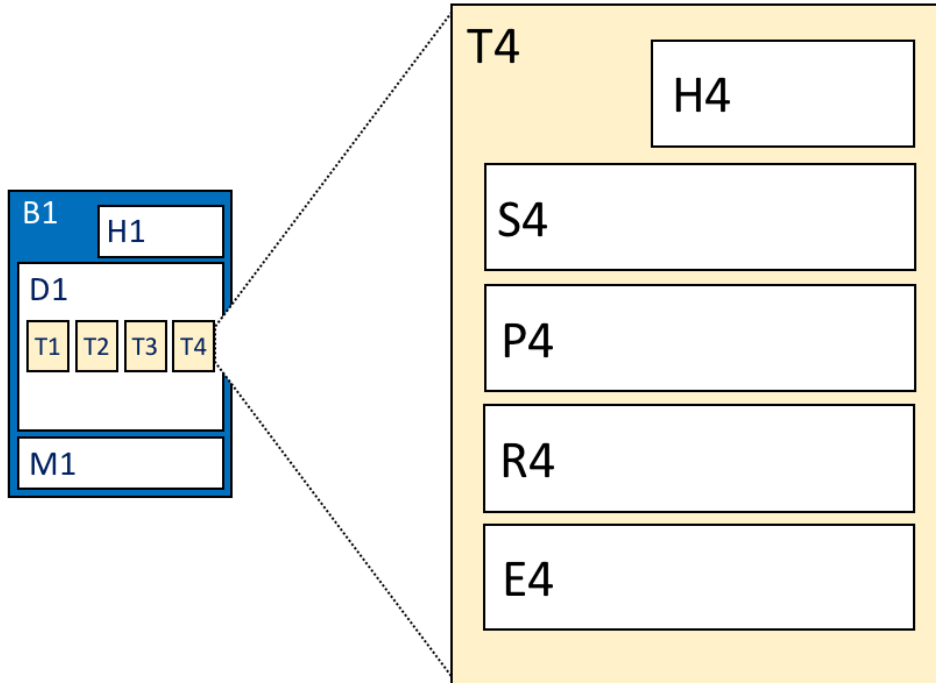
- Block header
 - Block number
 - Current Block Hash
 - Previous Block Hash
- Block Data
- Block Metadata



Transactions

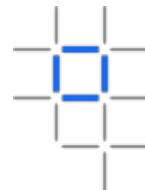


Captures some **essential metadata** about the transaction

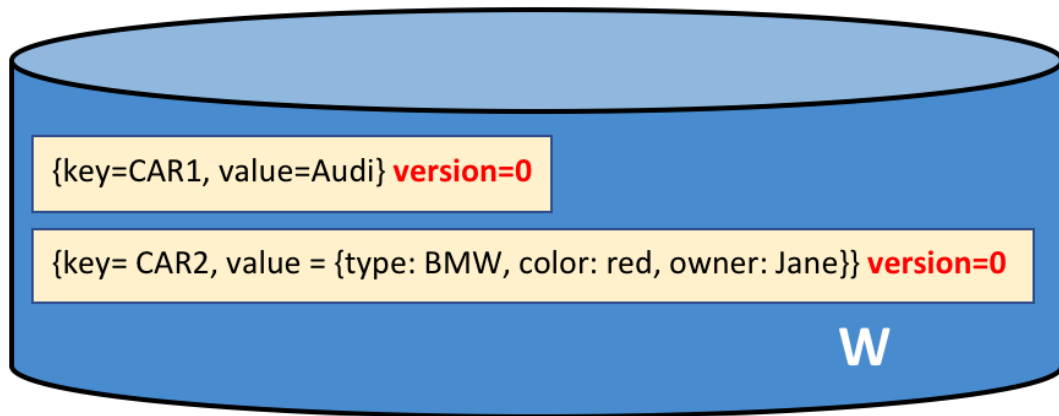



T4	Transaction
H4	Header
S4	Signature
P4	Proposal
R4	Response
E4	Endorsements
T4 ◁ V4	V4 is detailed view of T4

World State

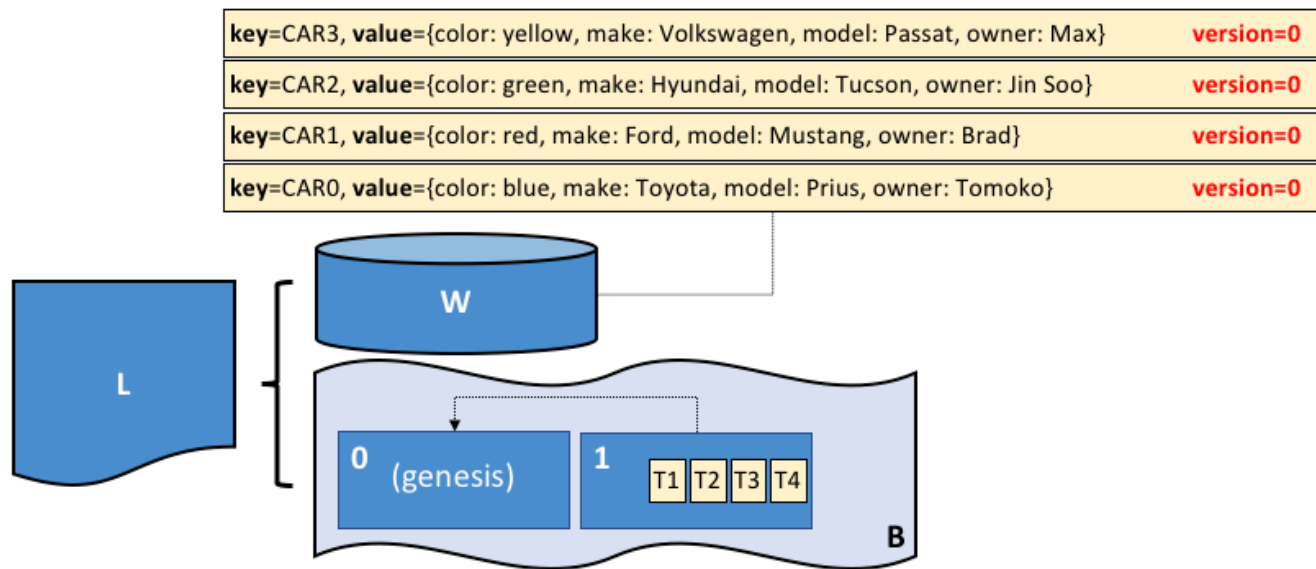
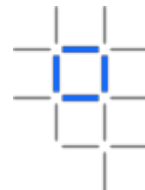


Holds **current state** of a set of business objects



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

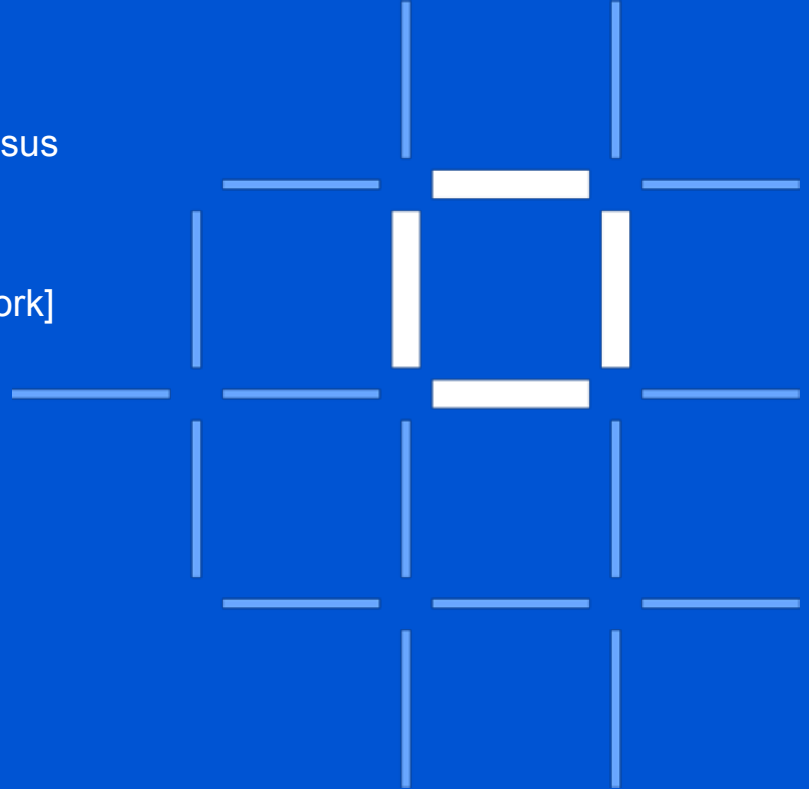
Example Ledger: fabcar



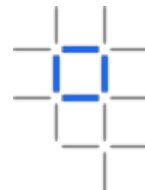


Technical Deep Dive

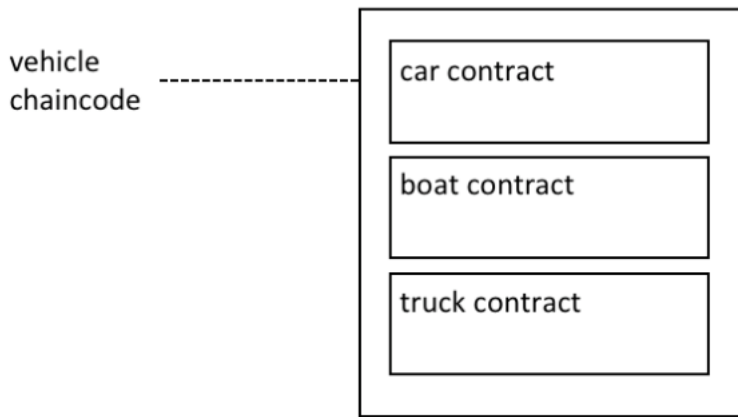
- Fabric Peer in Network Consensus
- Fabric Ledger and State DB
- [Smart Contract in Fabric Network]
- Gossip Protocol
- Private data to protect privacy
- Sample Distributed network deployment



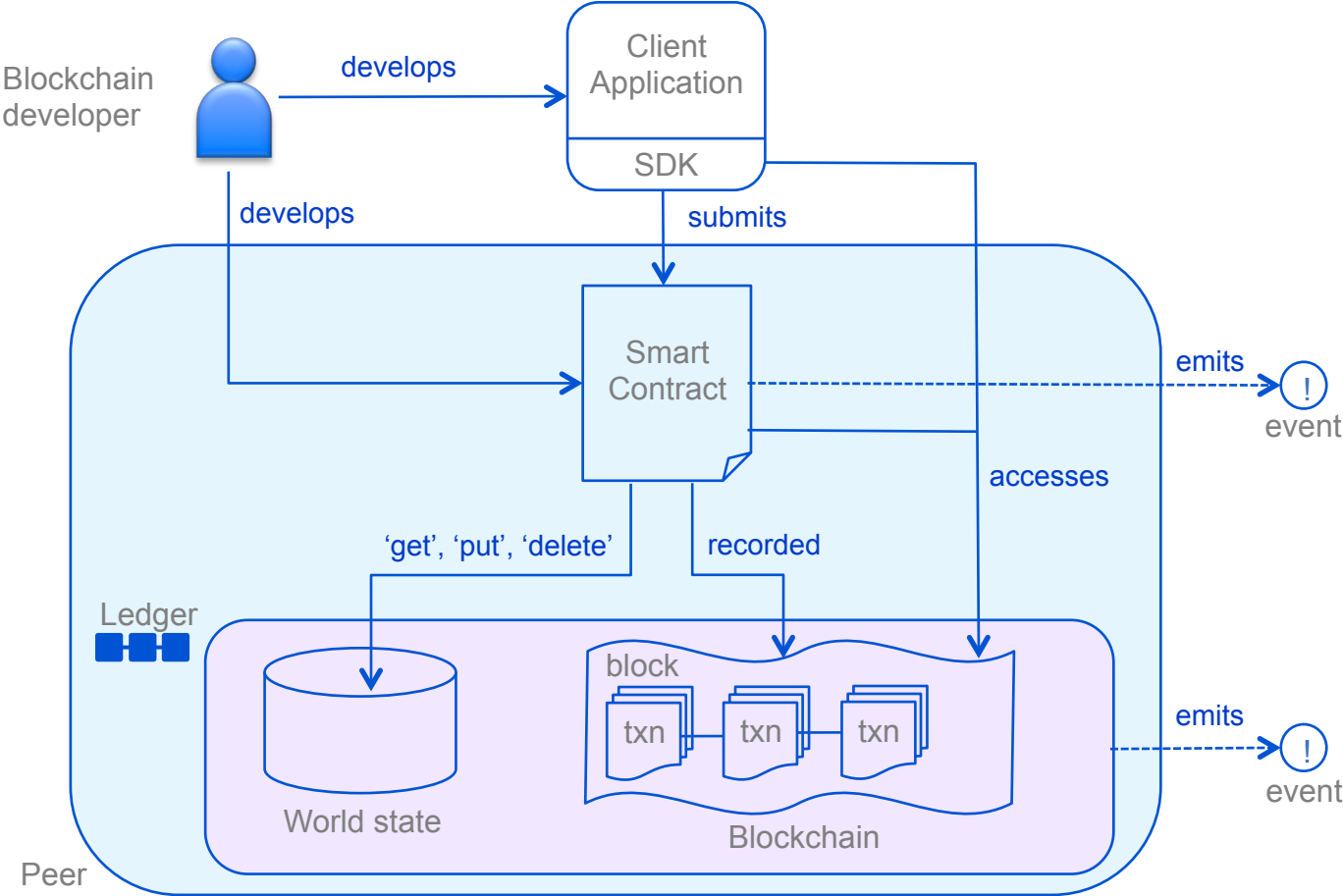
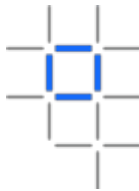
Smart Contract & chaincode



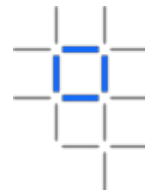
- Smart Contract
 - Heart of a blockchain network
 - Defines the rules between different organizations in executable code
 - Generate transactions that are recoded on the ledger
 - packaged into a chaincode
- Chaincode
 - Can package multiple smart contracts
 - Smart contracts are available to applications when a chain code is deployed



How smart contract interact with the ledger



Smart Contract Example – SimpleSet



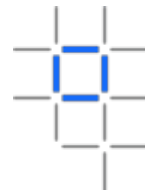
- Add the Go import statements

```
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



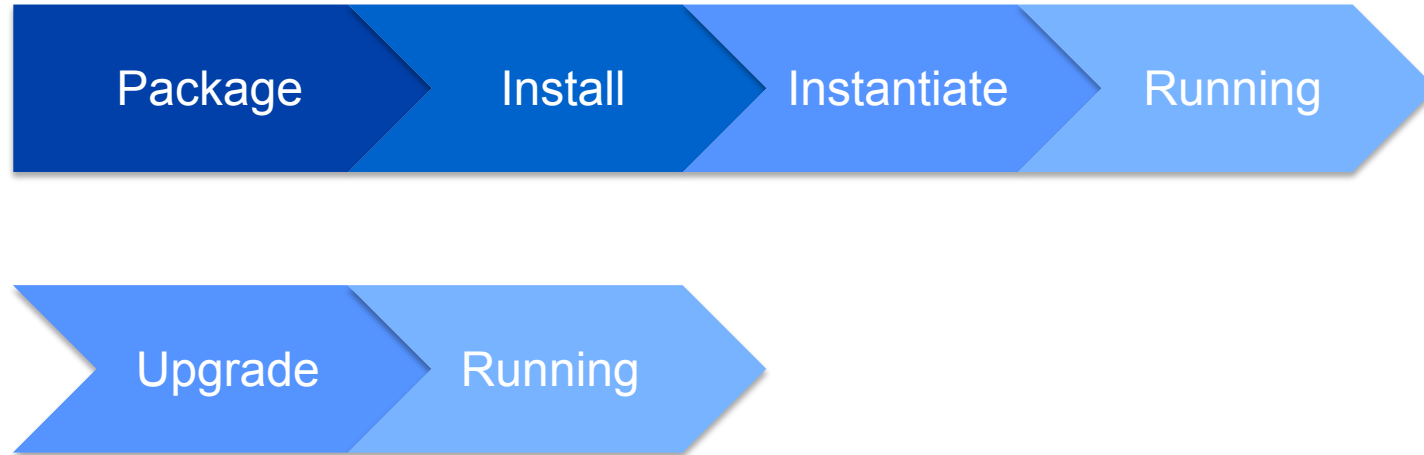
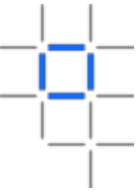
- Invoking

```
// 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 {

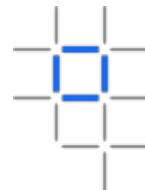
    function, args := stub.GetFunctionAndParameters()
    if function == "invoke" {
        // Make payment of X units from A to B
        return t.invoke(stub, args)
    } else if function == "delete" {
        // Deletes an entity from its state
        return t.delete(stub, args)
    } else if function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

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

Chaincode Lifecycle



Chaincode Lifecycle – Packaging



- 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

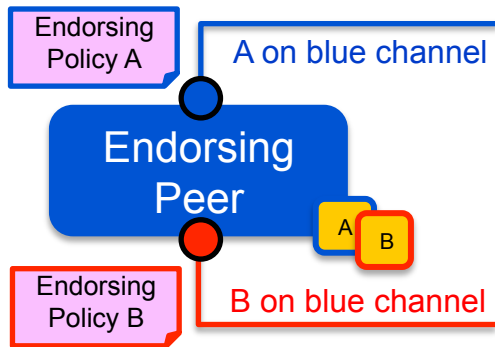
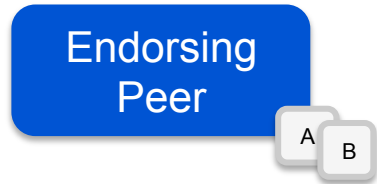
- Installs chaincode on a peer node
- Multiple chaincodes could be installed on a peer node
- Must install the chaincode on **each endorsing peer node** of a channel
- Example

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

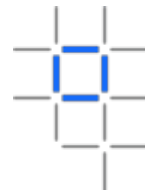
- Instantiate chaincode

- 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

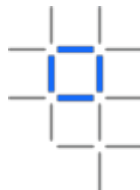
```
peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'  
peer chaincode invoke -o order-url -C mychannel -n mycc -c '{"Args":["invoke","a","b","10"]}'
```

- 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

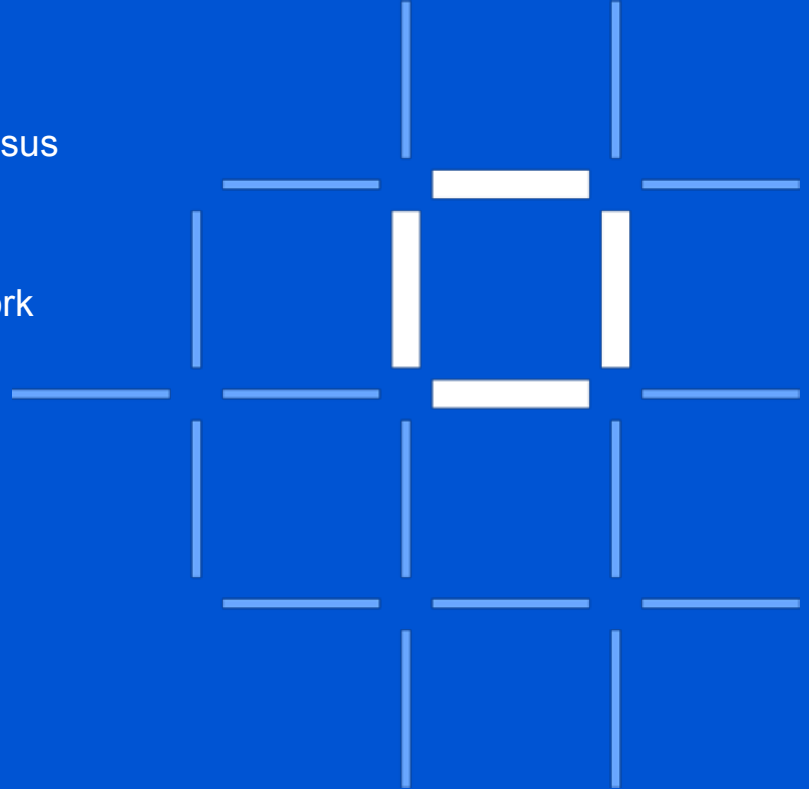


- Runs **within the peer process** rather than in an isolated container like normal chaincode
- Implement a number of system behaviors
- **LSCC**(Lifecycle system chaincode)
 - handles lifecycle requests of application chaincodes
- **CSCC**(Configuration system chaincode)
 - handles channel configuration on the peer side
- **QSCC**(Query system chaincode)
 - provides ledger query APIs such as getting blocks and transactions

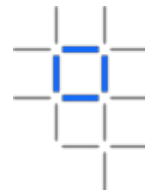


Technical Deep Dive

- Fabric Peer in Network Consensus
- Fabric Ledger and State DB
- Smart Contract in Fabric Network
- [Gossip Protocol]
- Private data to protect privacy
- Sample Distributed network deployment

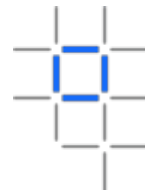


Functions of Gossip Protocol



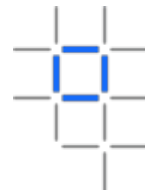
- Manages **peer discovery** and **channel membership**
- **Disseminates ledger data** across all peers on a channel
- Allowing **peer-to-peer state transfer** update of ledger data for new peers.

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

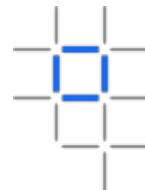
```
peer:  
  # Gossip related configuration  
  gossip: useLeaderElection: false  
  orgLeader: true
```

- 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

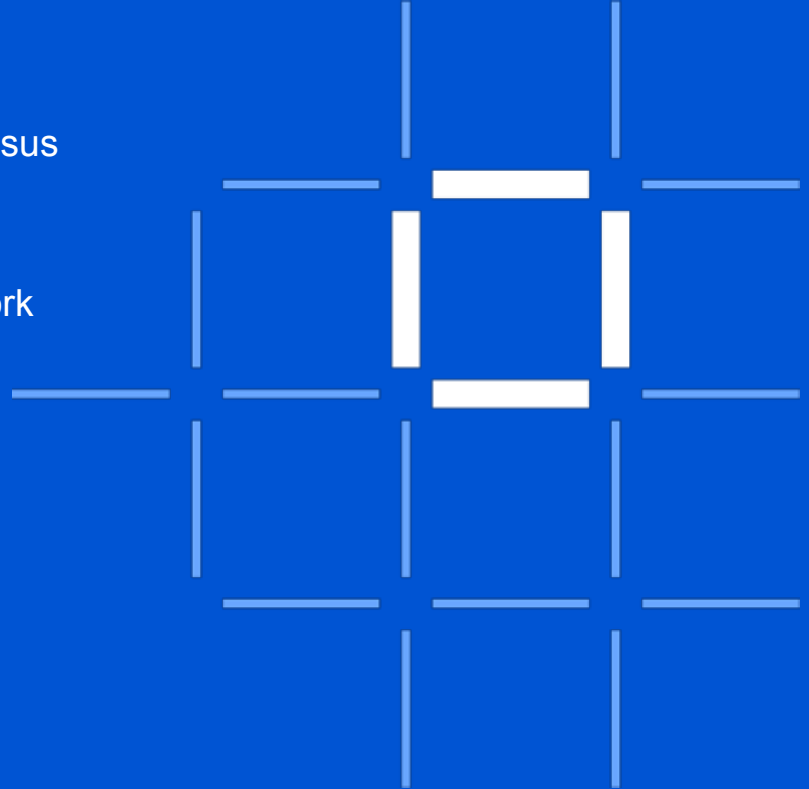


- Online peers indicate their availability by continually broadcasting “alive” messages
- Peers maintain channel membership by collecting these alive messages
- Peers receives/handle messages, and forward the received messages automatically as well
- Each peer continually pulls blocks from other peers 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

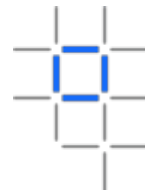


Technical Deep Dive

- Fabric Peer in Network Consensus
- Fabric Ledger and State DB
- Smart Contract in Fabric Network
- Gossip Protocol
- [Private data to protect privacy]
- Sample Distributed network deployment

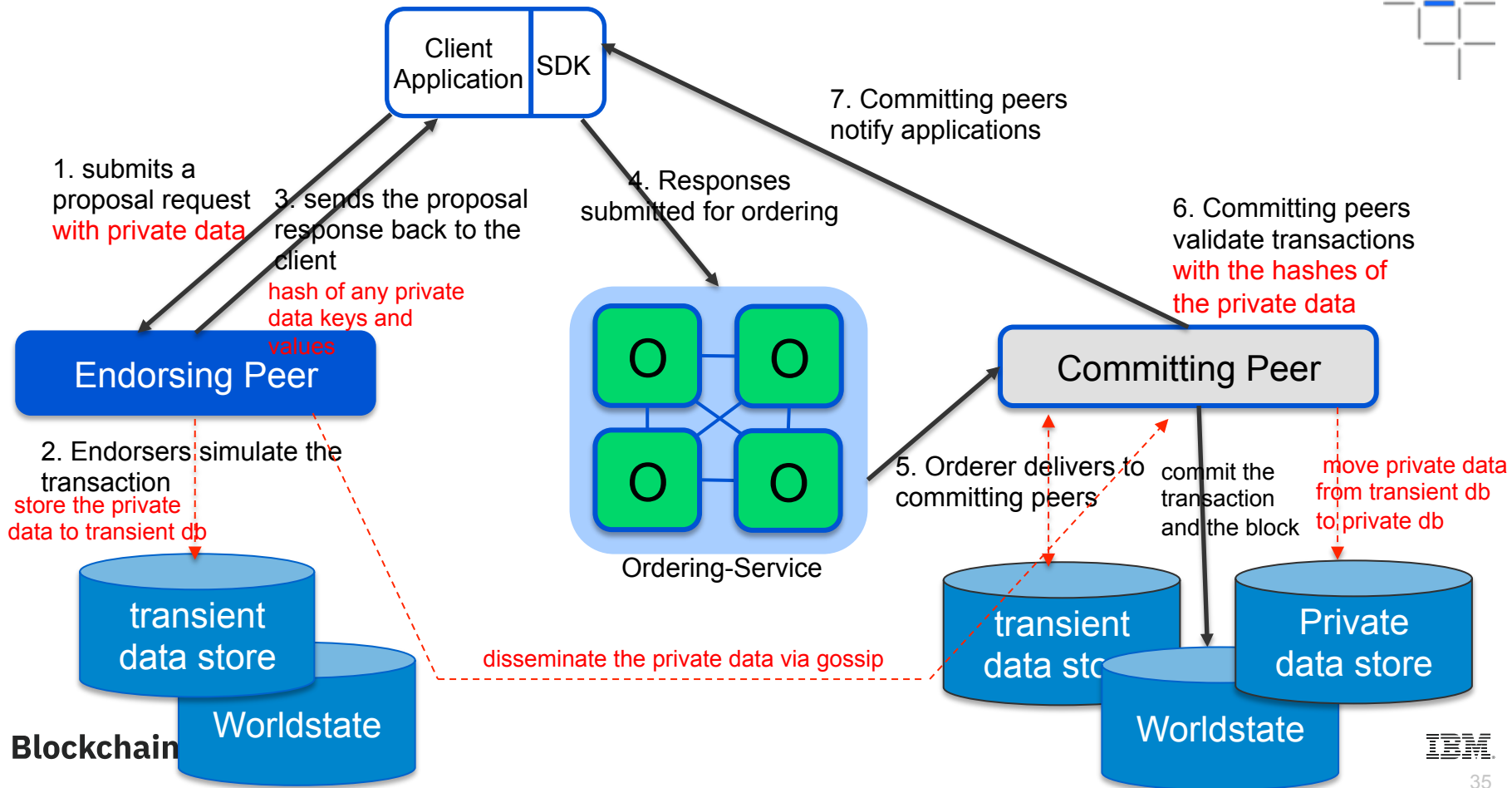
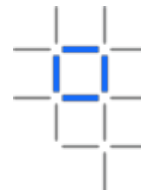


Private Data

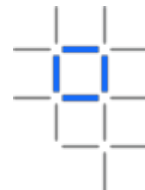


- Confidential data that is [stored in a private database](#) on each authorized peer
- Private data collection policy to define authorized peers
- Ordering service does not see the private data
- Sent peer-to-peer [via gossip protocol](#)

Transaction flow with private data

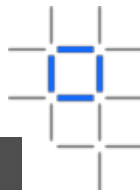


An example to explain private data collection



```
[
  {
    "name": "collectionMarbles",
    "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive": 1000000,
    "memberOnlyRead": true
  },
  {
    "name": "collectionMarblePrivateDetails",
    "policy": "OR('Org1MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive": 3,
    "memberOnlyRead": true
  }
]
```

How to save private data



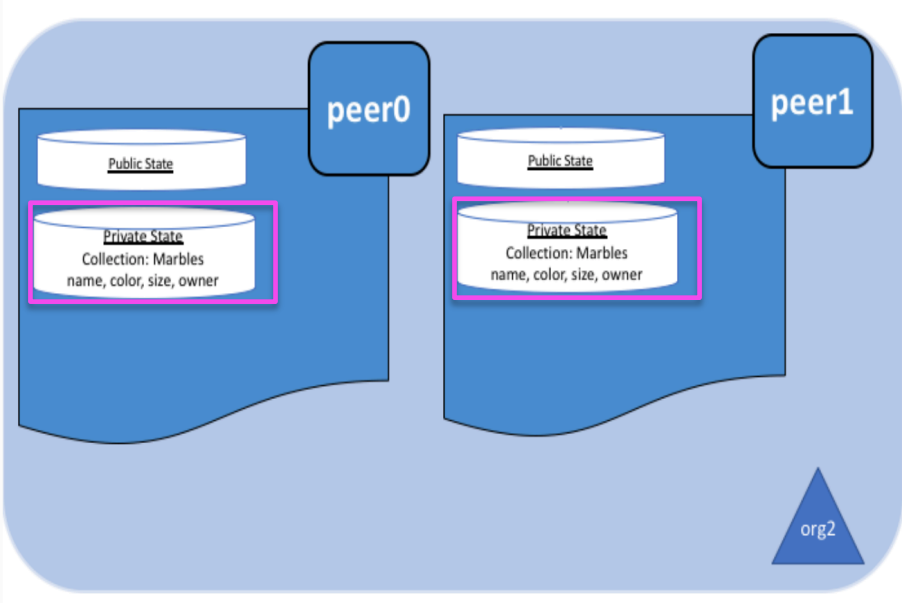
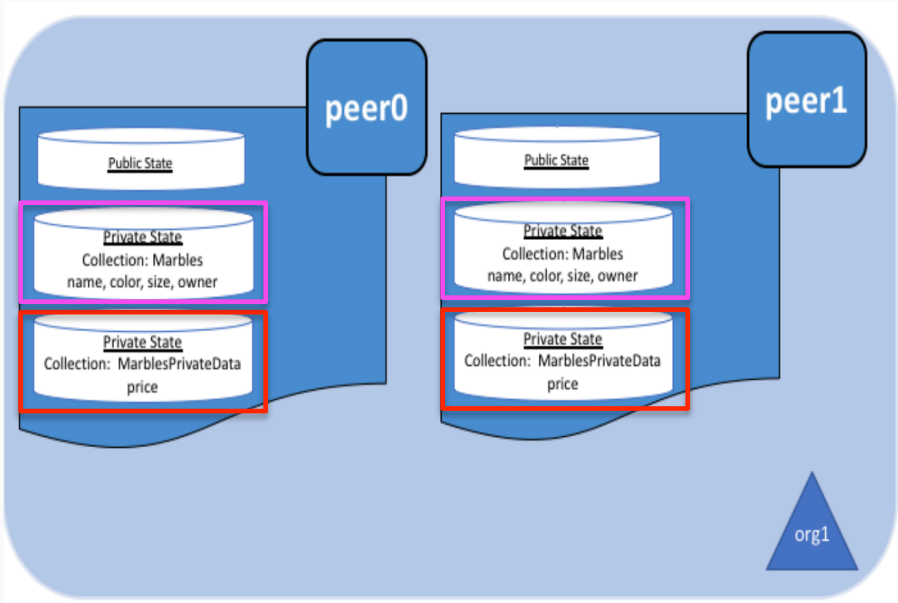
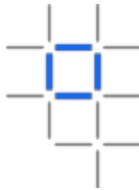
```
// ==== Create marble object, marshal to JSON, and save to state ====
marble := &marble{
    ObjectType: "marble",
    Name: marbleInput.Name,
    Color: marbleInput.Color,
    Size: marbleInput.Size,
    Owner: marbleInput.Owner,
}

// === Save marble to state ===
err = stub.PutPrivateData("collectionMarbles", marbleInput.Name, marbleJSONasBytes)

// ==== Create marble private details object with price, marshal to JSON, and save to
state ====
marblePrivateDetails := &marblePrivateDetails{
    ObjectType: "marblePrivateDetails",
    Name: marbleInput.Name,
    Price: marbleInput.Price,
}

marblePrivateDetailsBytes, err := json.Marshal(marblePrivateDetails)
err = stub.PutPrivateData("collectionMarblePrivateDetails", marbleInput.Name,
marblePrivateDetailsBytes)
```

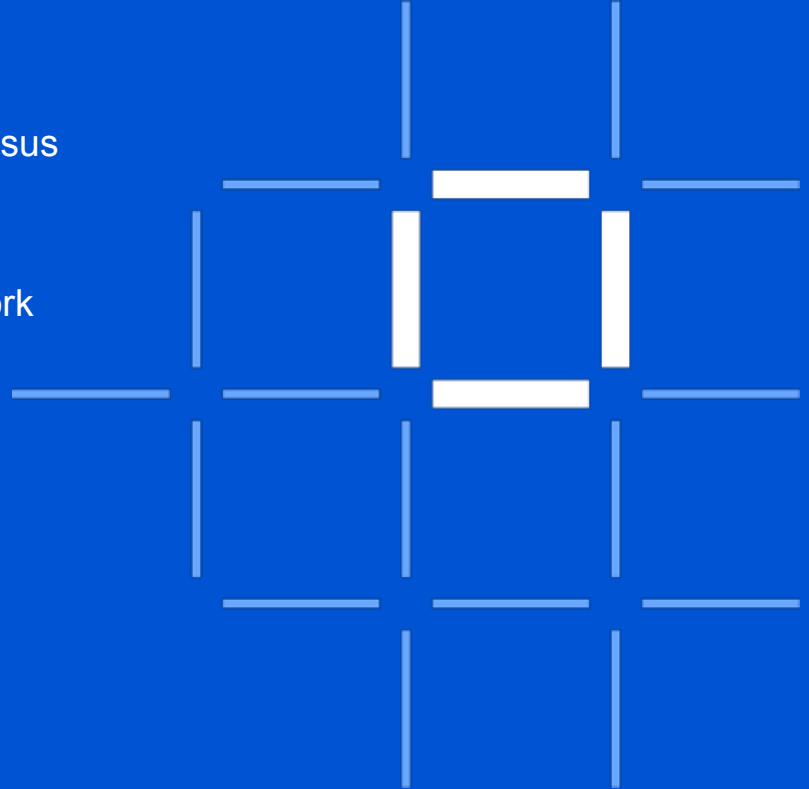
Private Data DB



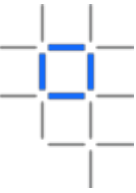


Technical Deep Dive

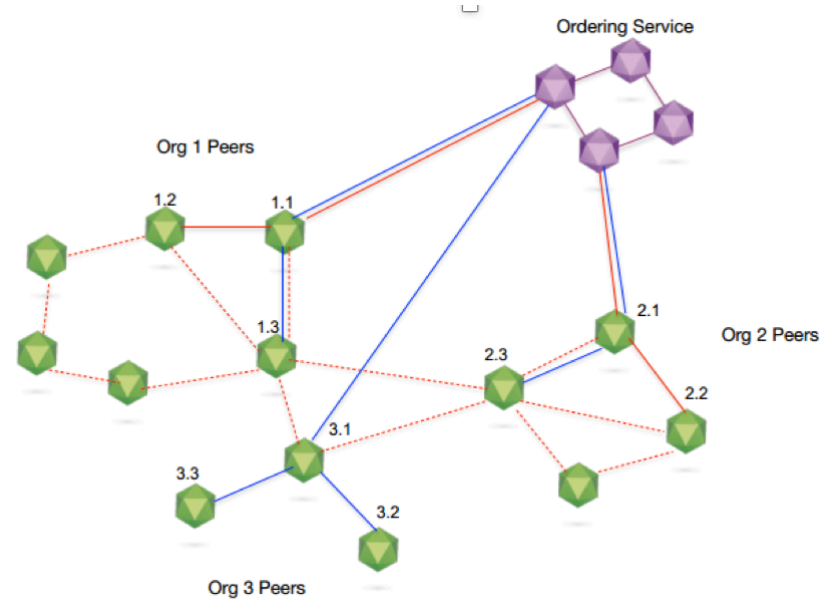
- Fabric Peer in Network Consensus
- Fabric Ledger and State DB
- Smart Contract in Fabric Network
- Gossip Protocol
- Private data to protect privacy
- [Sample Distributed network deployment]



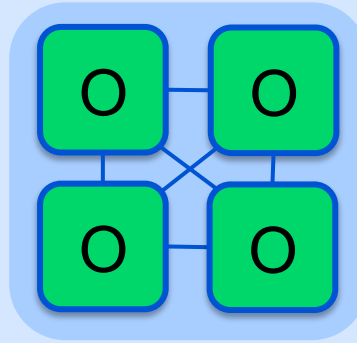
What is a Hyperledger blockchain network?



- 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



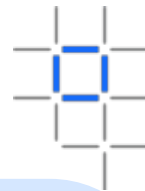
Ordering-Service

Hyperledger Fabric Network

An Ordering Service is configured and started for the network:

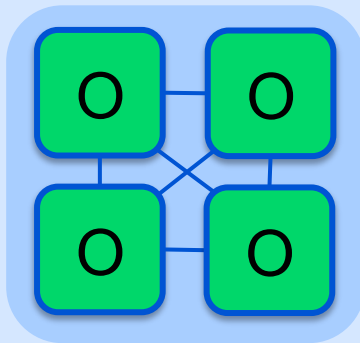
\$ docker-compose [-f orderer.yml] ...

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



E_0

E_1



Ordering-Service

E_2

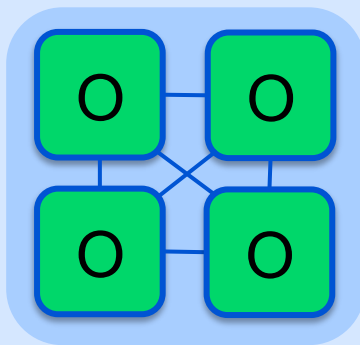
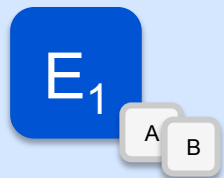
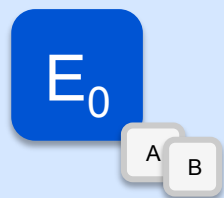
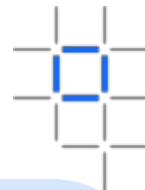
P_3

Hyperledger Fabric Network

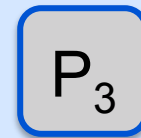
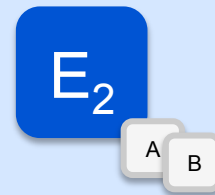
A peer is configured and started for each Endorser or Committer in the network:

\$ peer node start ...

Bootstrap Network (3/6) - Install Chaincode



Ordering-Service

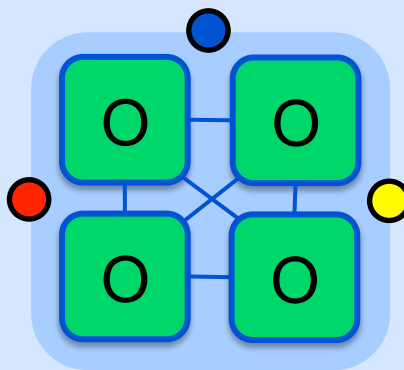
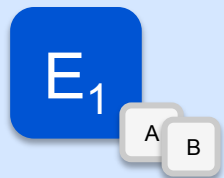
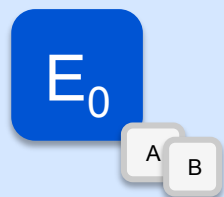
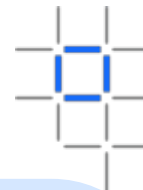


Hyperledger Fabric Network

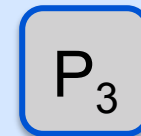
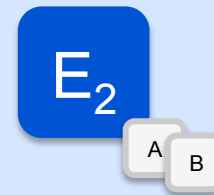
Chaincode is installed onto each Endorsing Peer that needs to execute it:

\$ peer chaincode install ...

Bootstrap Network (4/6) – Create Channels



Ordering-Service

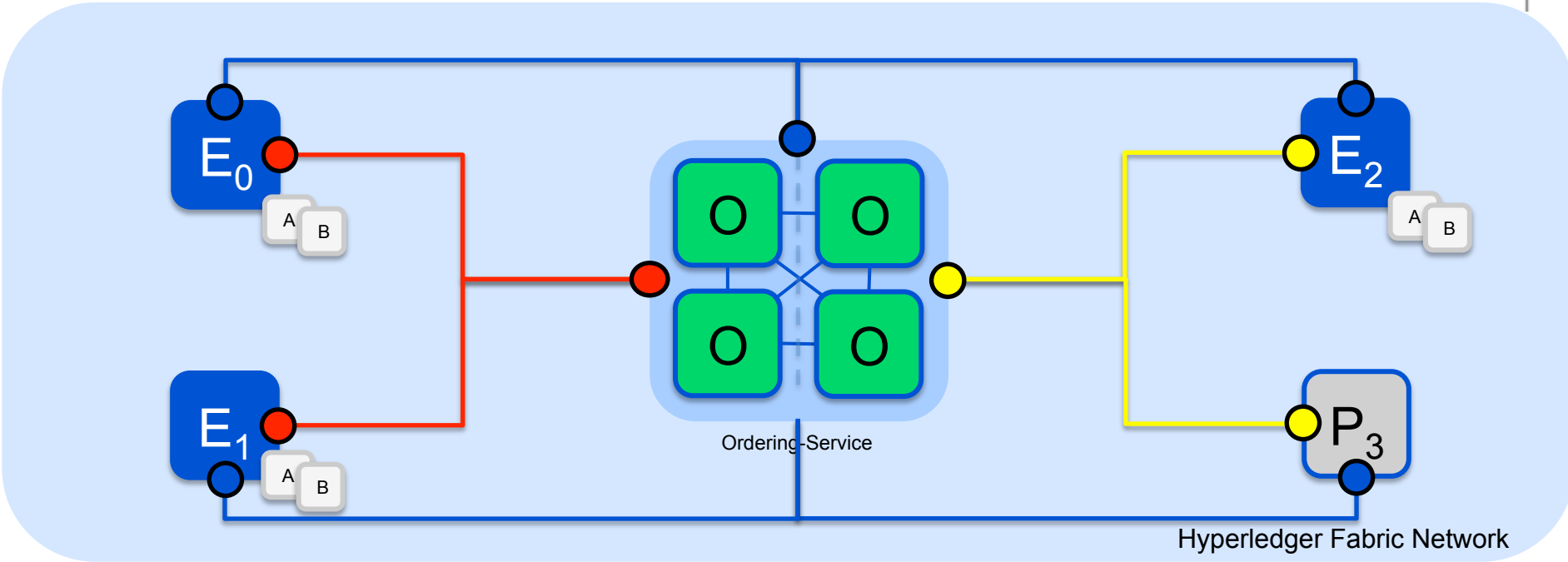
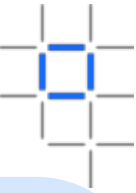


Hyperledger Fabric Network

Channels are created on the ordering service:

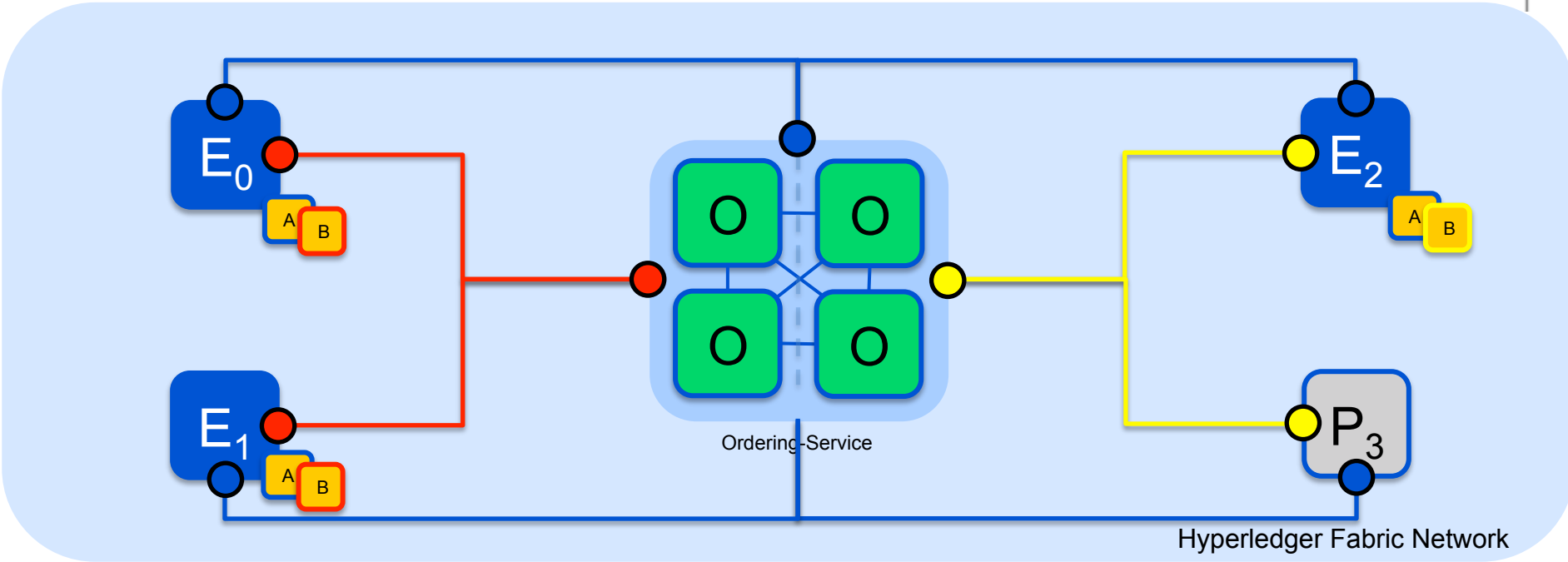
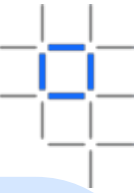
\$ peer channel create -o [orderer] ...

Bootstrap Network (5/6) – Join Channels



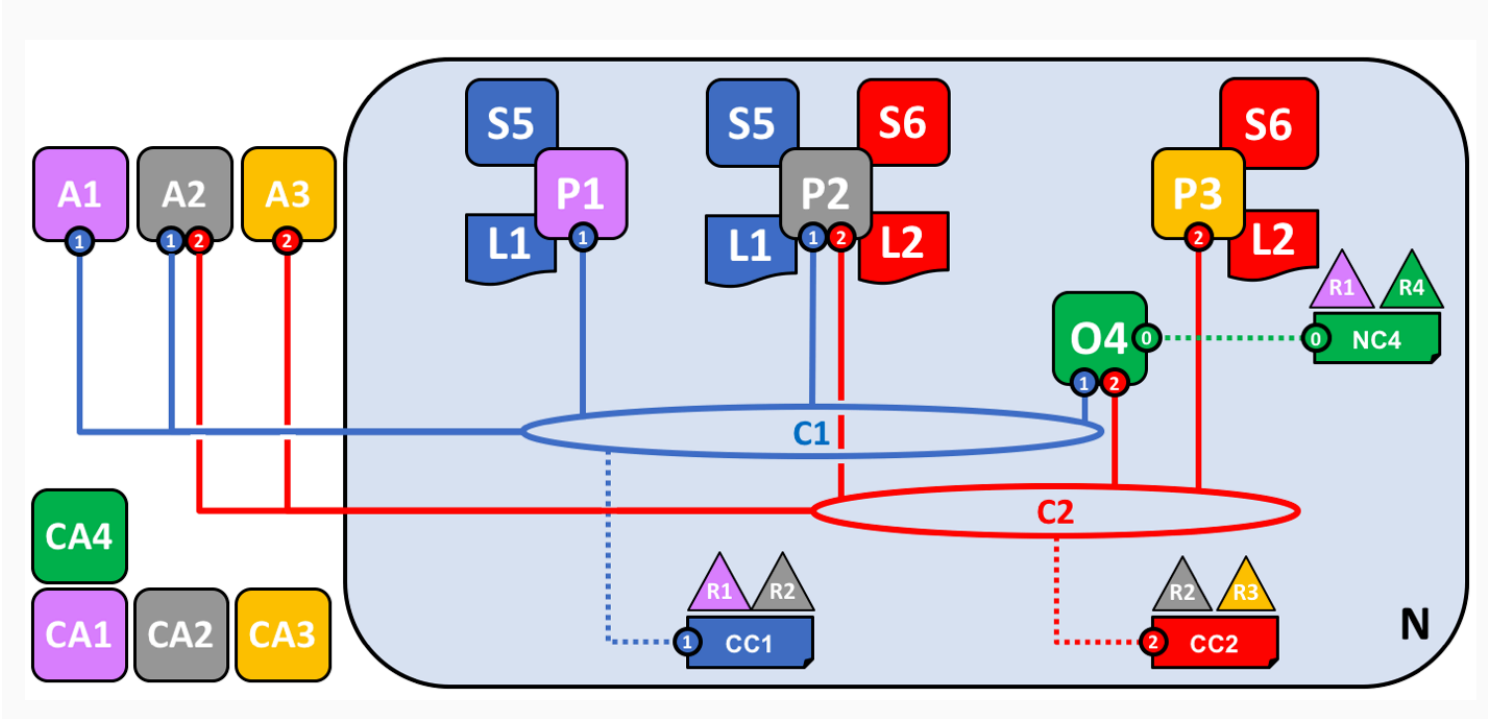
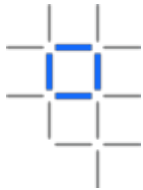
Peers that are permitted can then join the channels they want to transact on:
\$ peer channel join ...

Bootstrap Network (6/6) – Instantiate Chaincode

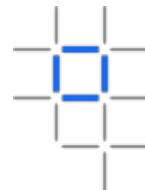


Peers finally instantiate the Chaincode on the channels they want to transact on:
\$ peer chaincode instantiate ... -P 'policy'

Sample network with multiple orgs/channels



Further Hyperledger Fabric Information



- Project Home: <https://www.hyperledger.org/projects/fabric>
- GitHub Repo: <https://github.com/hyperledger/fabric>
- Latest Docs: <https://hyperledger-fabric.readthedocs.io/en/latest/>
- Community Chat: <https://chat.hyperledger.org/channel/fabric>
- Project Wiki: <https://wiki.hyperledger.org/projects/fabric>
- Design Docs: <https://wiki.hyperledger.org/community/fabric-design-docs>

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 功能介绍 架构概览
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 **“replay”**
获取往期视频地址

公众号中发送 **“报名”**，
即有机会参加Fabric线下训练营