

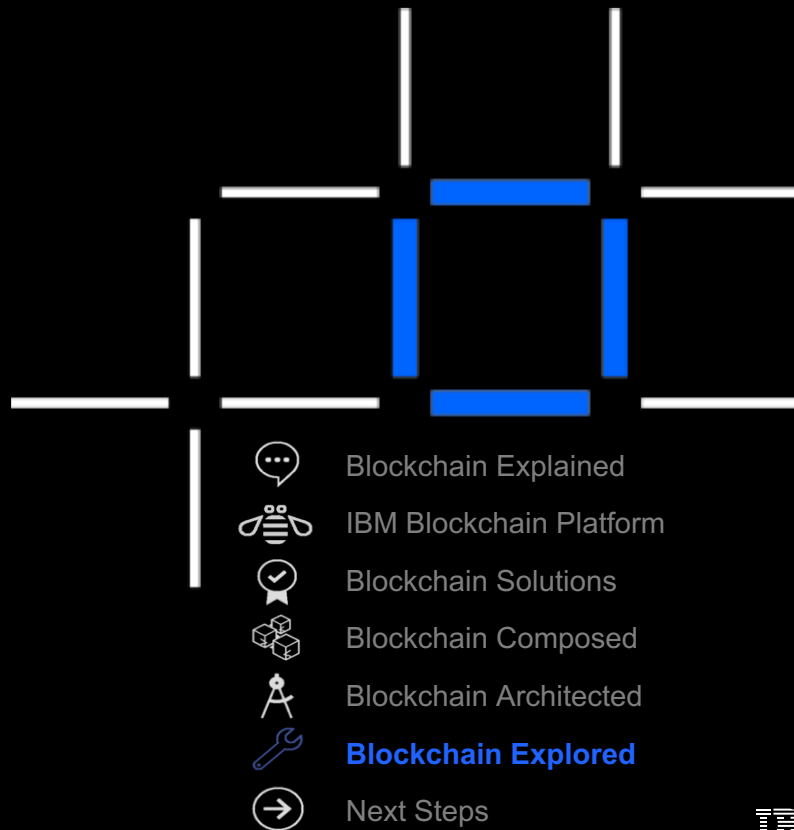
Blockchain Explored

A Technical Deep-Dive on Hyperledger Fabric V1

Barry Silliman
IBM Z Blockchain Enablement
Washington Systems Center
silliman@us.ibm.com

Presented on February 14, 2018

IBM Blockchain



IBM



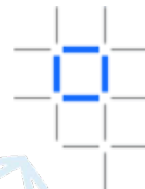
Project Status and
Roadmap



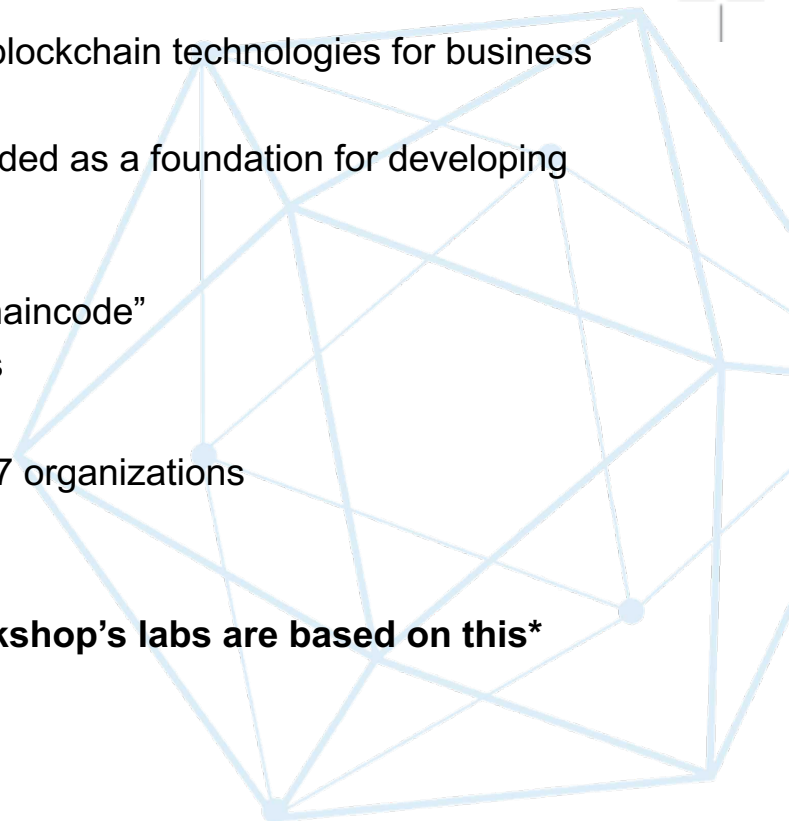
Technical Deep Dive



What is Hyperledger Fabric?



- Linux Foundation Hyperledger
 - A collaborative effort created to advance cross-industry blockchain technologies for business
- Hyperledger Fabric
 - An implementation of blockchain technology that is intended as a foundation for developing blockchain applications
 - Key technical features:
 - A shared ledger and smart contracts implemented as “chaincode”
 - Privacy and permissioning through membership services
 - Modular architecture and flexible hosting options
- V1.0 released July 2017: contributions by 159 engineers from 27 organizations
 - IBM is one of the contributors to Hyperledger Fabric
- V 1.1 release targeted for end of March 2018
 - Most current code is v1.1.0-alpha ***The rest of the workshop’s labs are based on this***



Hyperledger Fabric Roadmap

V1 Alpha

- Docker images
- Tooling to bootstrap network
- Fabric CA or bring your own
- Java and Node.js SDKs
- Ordering Services - Solo and Kafka
- Endorsement policy
- Level DB and Couch DB
- Block dissemination across peers via Gossip

V1 GA

- Hardening, usability, serviceability, load, operability and stress test
- Chaincode ACL
- Chaincode packaging & LCI
- Pluggable crypto
- HSM support
- Consumability of configuration
- Next gen bootstrap tool (config update)
- Config transaction lifecycle
- Eventing security
- Cross Channel Query
- Peer management APIs
- Documentation

V 1.1 *

- Node.js chaincode
- Node.js connection profile
- Provide an encryption library
- Trigger events per channel
- Enhanced CC attribute access control
- Orderer horizontal scaling improvements
- *Preview of*
 - Private channel data
 - Finer grained access control on channels
 - Identity mixer

V Next *

- SBFT
- Archive and pruning
- System Chaincode extensions
- Application crypto library
- Dynamic service discovery
- REST wrapper
- Python SDK
- Java Chaincode
- Side DB for private data
- Identity Mixer

March 2017

July 2017

Q1 2018

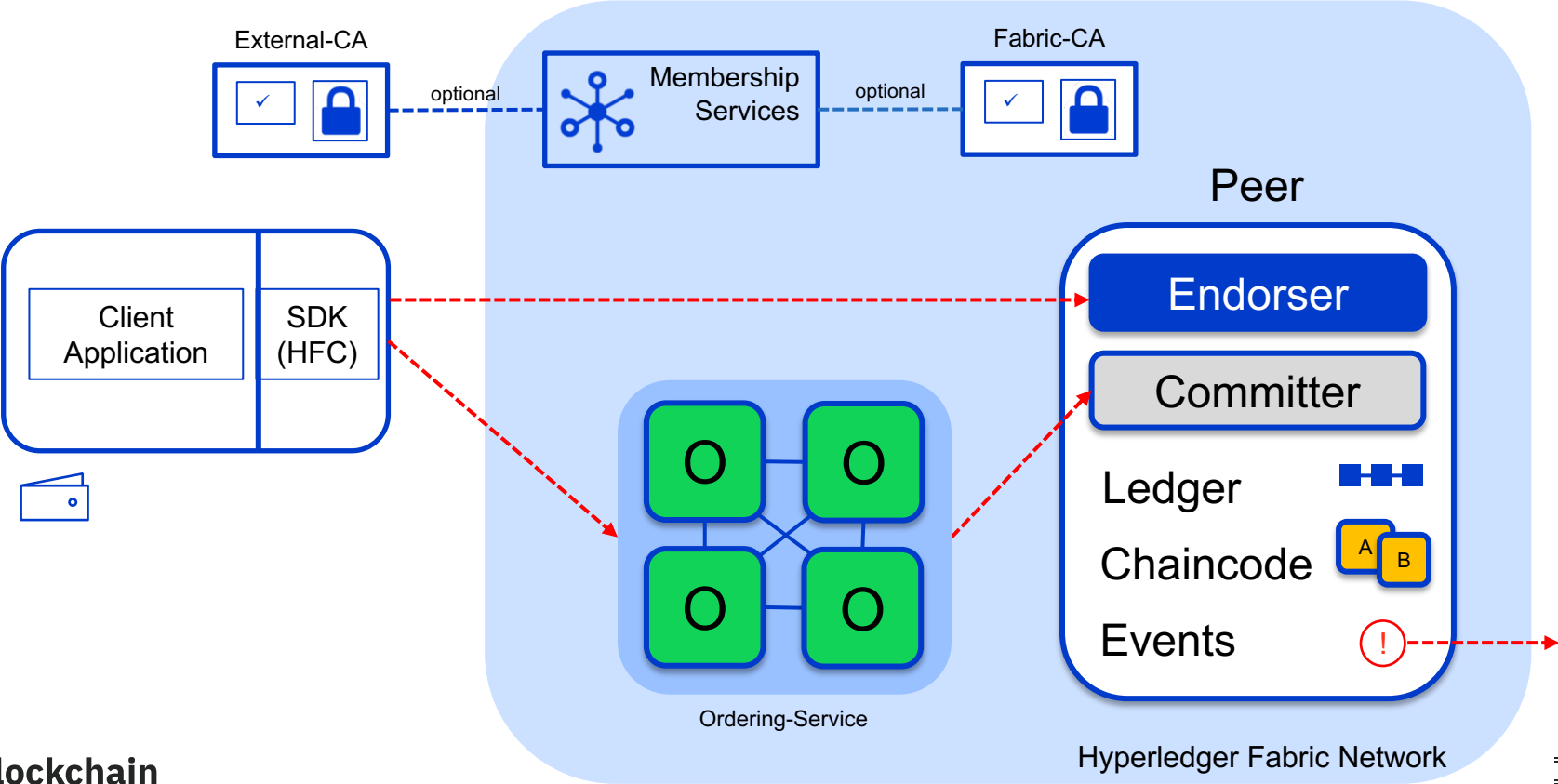
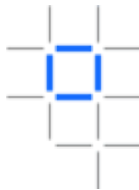
Future



HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

* Dates for content and releases are determined by the Hyperledger community

Hyperledger Fabric V1 Architecture

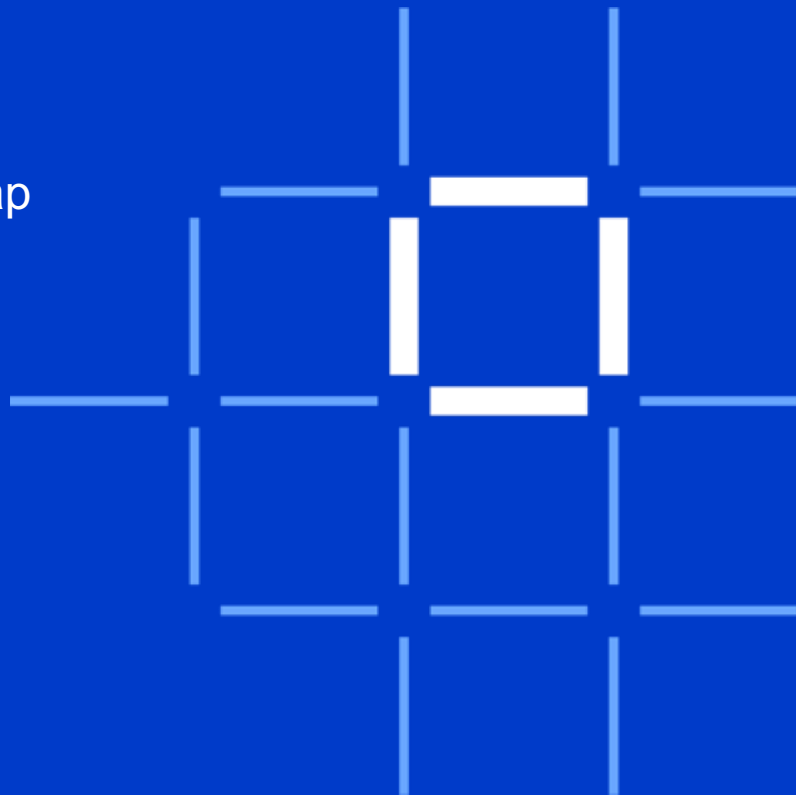




Project Status and Roadmap



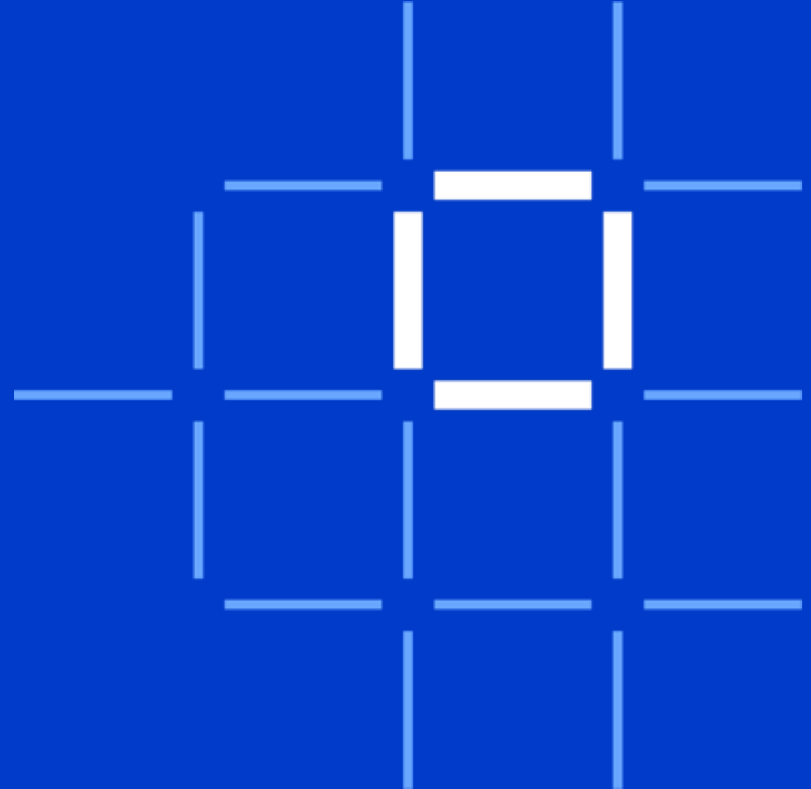
Technical Deep Dive



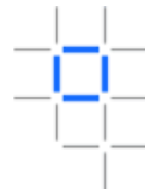





Technical Deep Dive

- [Network Consensus]
- Channels and Ordering Service
- Network setup
- Endorsement Policies
- Permissioned ledger access
- Pluggable world-state

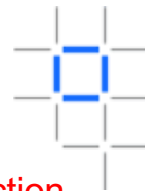


Nodes and roles



	Committing Peer: Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).
	Endorsing Peer: Specialized committing peer that receives a transaction proposal for endorsement, responds granting or denying endorsement. Must hold smart contract
	Ordering Node: Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes. Does not hold smart contract. Does not hold ledger.

Sample transaction: Step 1/7 – Propose transaction



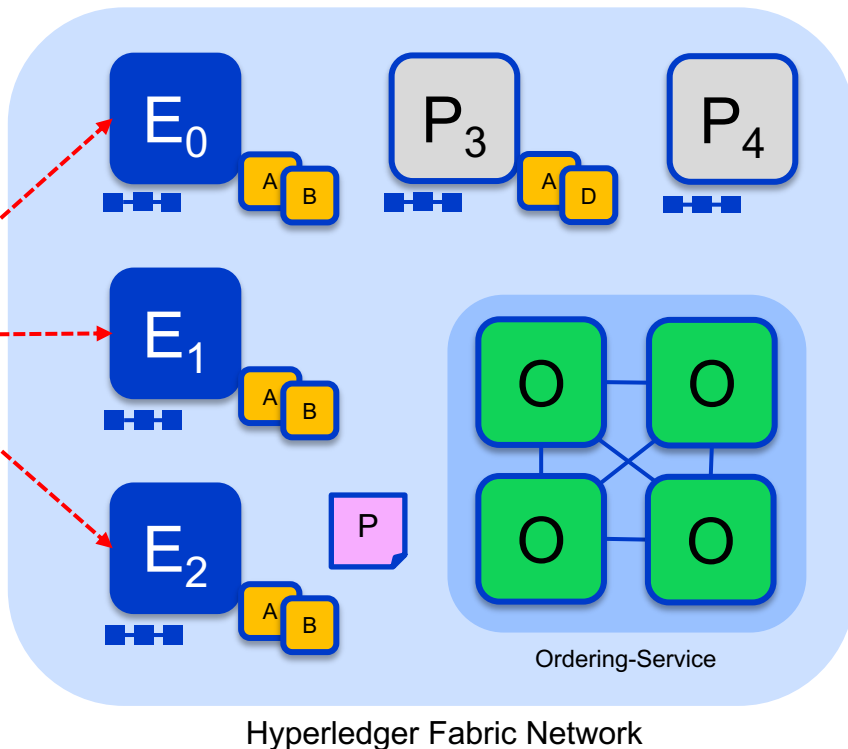
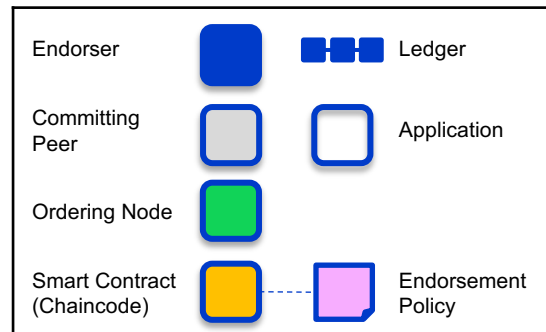
Application proposes transaction

Endorsement policy:

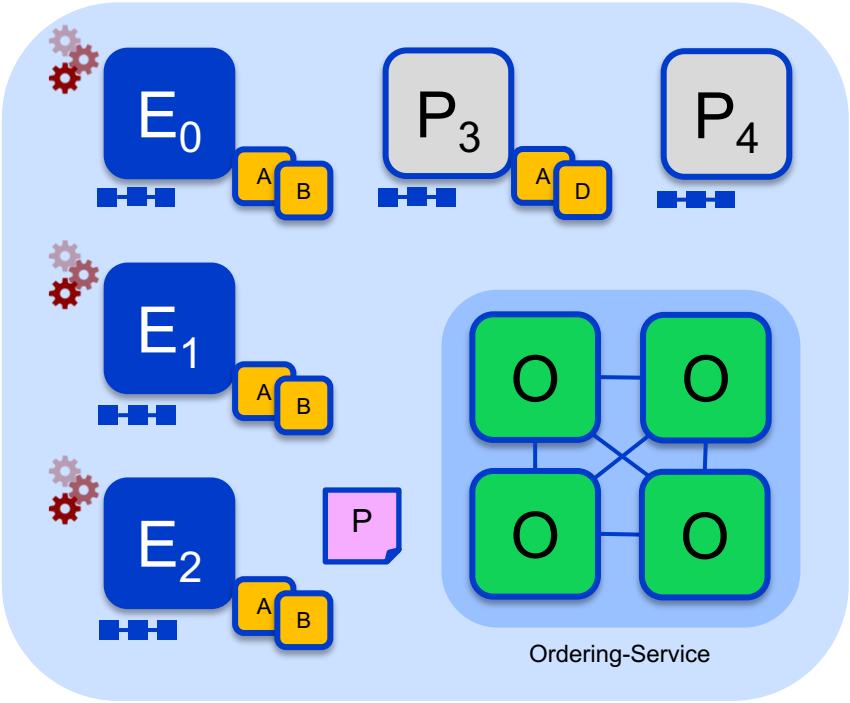
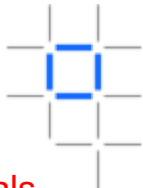
- “E₀, E₁ and E₂ must sign”
- (P₃, P₄ are not part of the policy)

Client application submits a transaction proposal for Smart Contract A. It must target the required peers {E₀, E₁, E₂}

Key:



Sample transaction: Step 2/7 – Execute proposal



Hyperledger Fabric Network

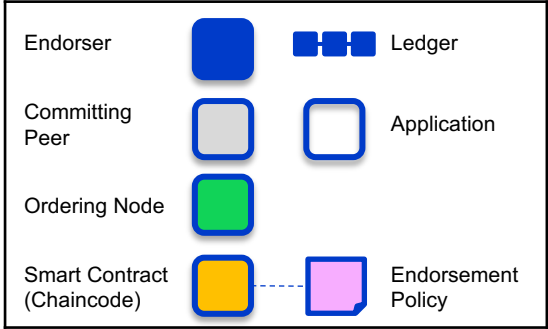
Endorsers Execute Proposals

E₀, E₁ & E₂ will each execute the proposed transaction. None of these executions will update the ledger

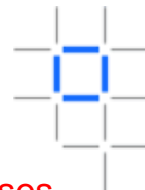
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



Sample transaction: Step 3/7 – Proposal Response



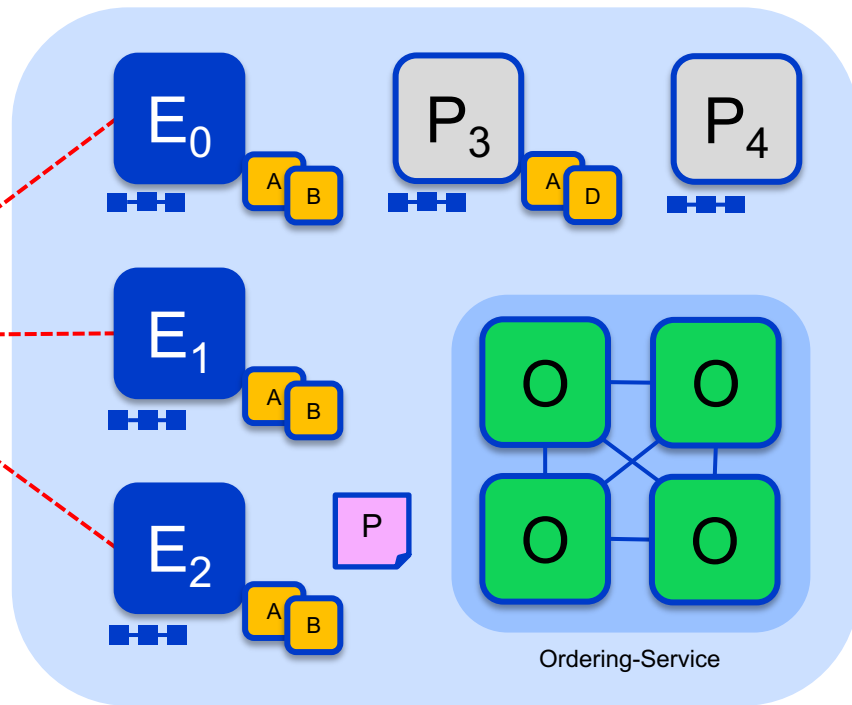
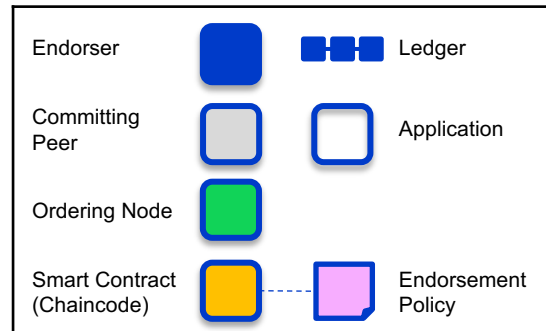
Application receives responses

RW sets are asynchronously returned to application

The RW sets are signed by each endorser, and also includes each record version number

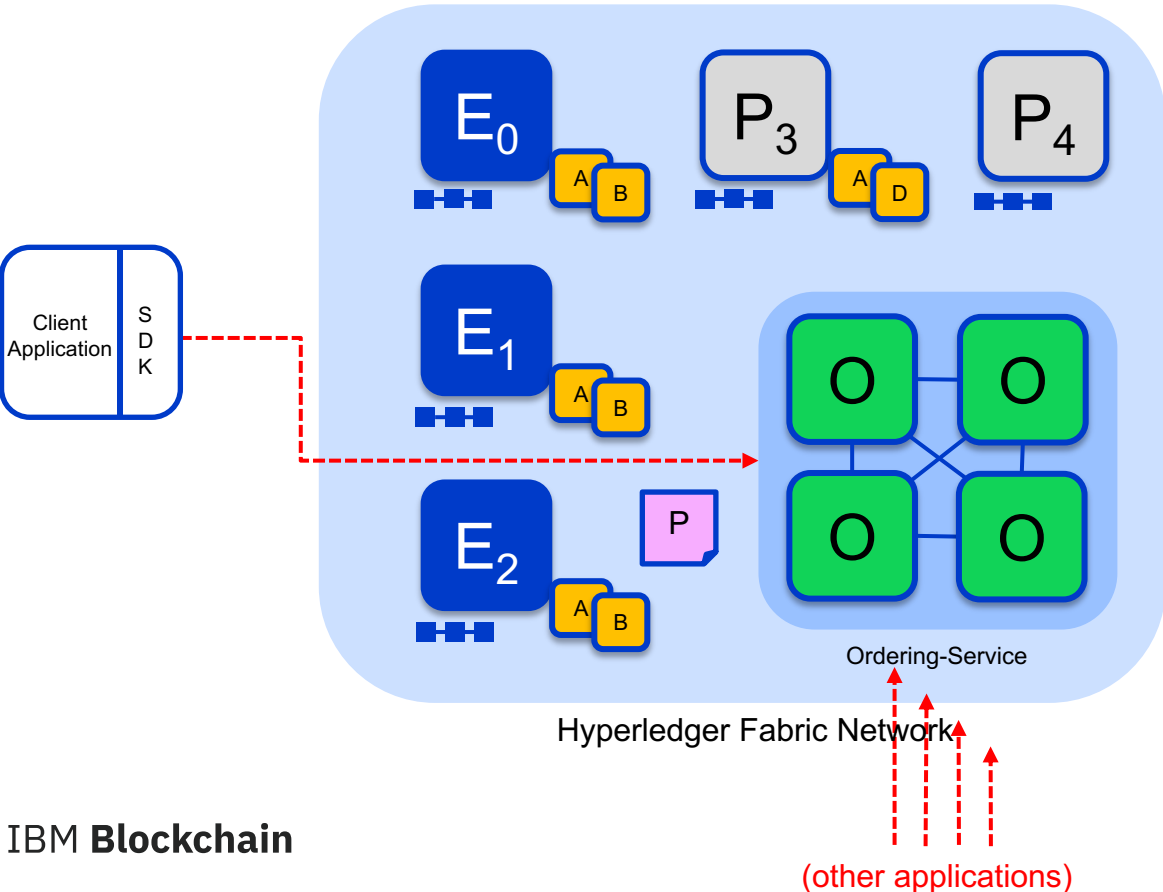
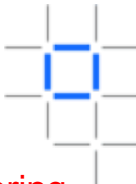
(This information will be checked much later in the consensus process)

Key:



Hyperledger Fabric Network

Sample transaction: Step 4/7 – Order Transaction



Responses submitted for ordering

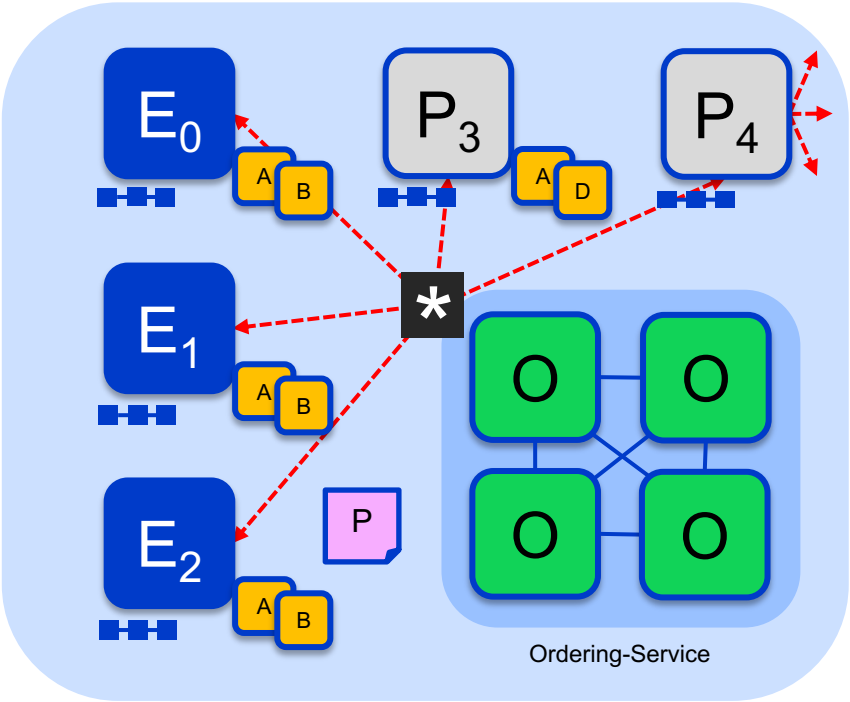
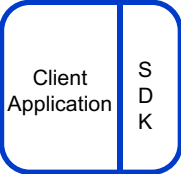
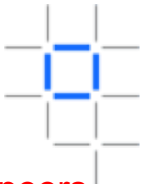
Application submits responses as a transaction to be ordered.

Ordering happens across the fabric in parallel with transactions submitted by other applications

Key:

Endorsor			Ledger
Committing Peer			Application
Ordering Node			
Smart Contract (Chaincode)			Endorsement Policy

Sample transaction: Step 5/7 – Deliver Transaction



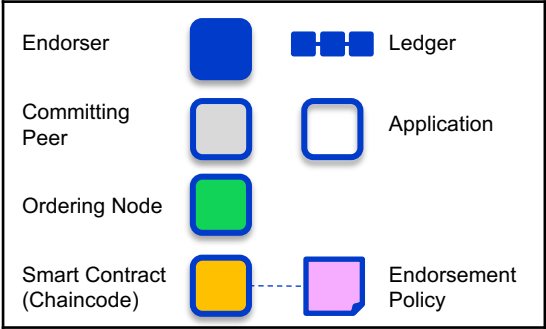
Hyperledger Fabric Network

Orderer delivers to committing peers

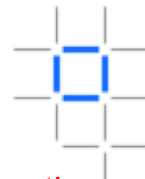
Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown)

- Different ordering algorithms available:
- SOLO (Single node, development)
 - Kafka (Crash fault tolerance)

Key:



Sample transaction: Step 6/7 – Validate Transaction



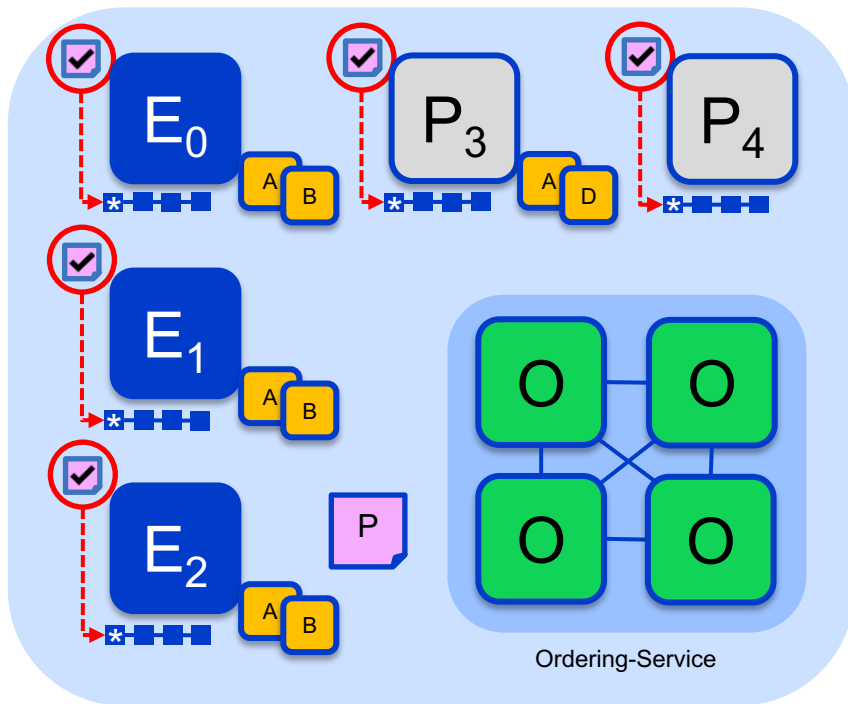
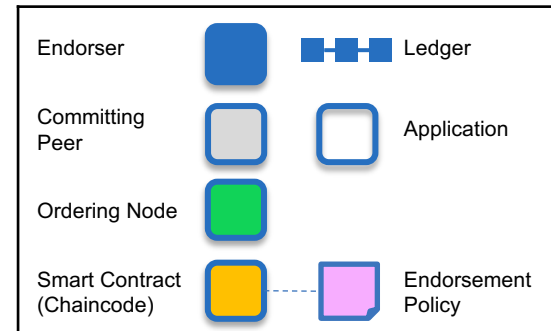
All peers validate (commit) transactions

Every peer has the committing role and validates against the endorsement policy, and checks that RW sets are still valid for current world state

Validated transactions are applied to the world state and retained on the ledger

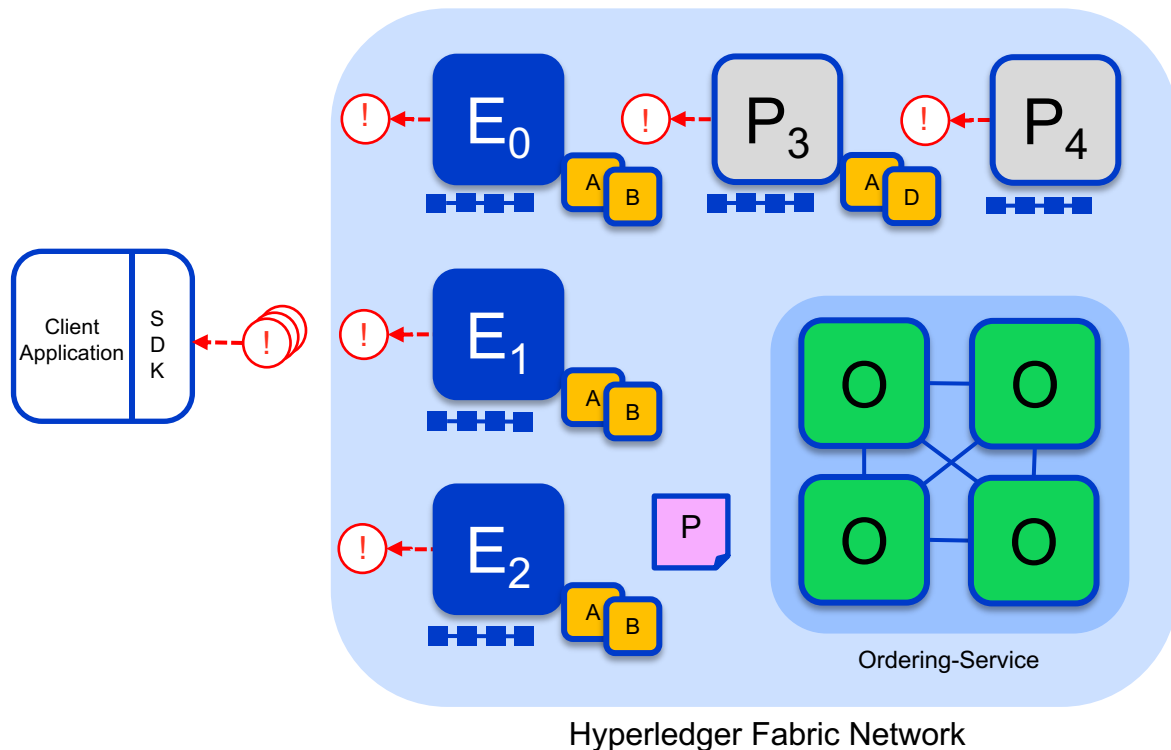
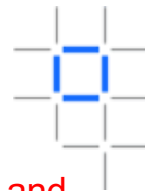
Invalid transactions are also retained on the ledger but do not update world state

Key:



Hyperledger Fabric Network

Sample transaction: Step 7/7 – Notify Transaction

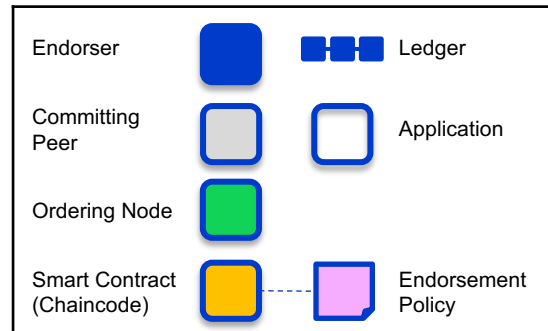


Committing peers emit block and transaction events

Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger

Applications will be notified by each peer to which they are connected

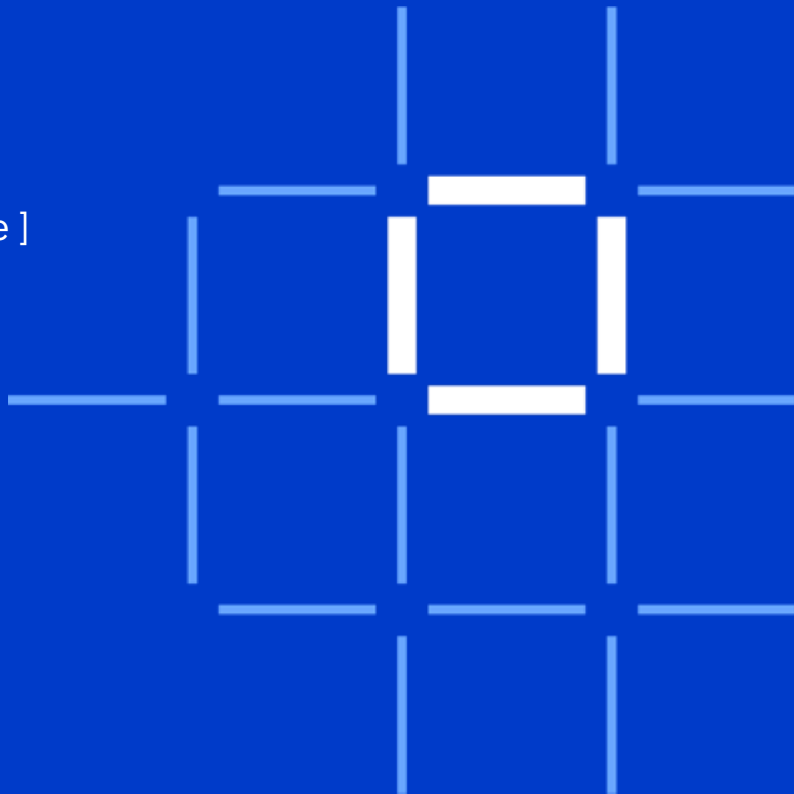
Key:



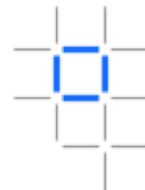


Technical Deep Dive

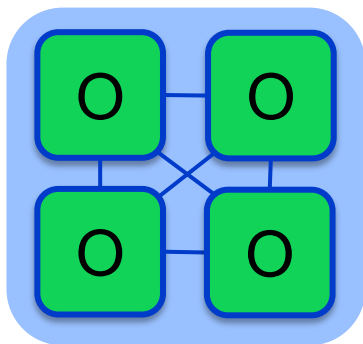
- Network Consensus
- [Channels and Ordering Service]
- Network setup
- Endorsement Policies
- Permissioned ledger access
- Pluggable world-state



Ordering Service



The ordering service packages transactions into blocks to be delivered to peers. Communication with the service is via channels.



Ordering-Service

Different configuration options for the ordering service include:

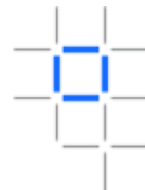
– **SOLO**

- Single node for development

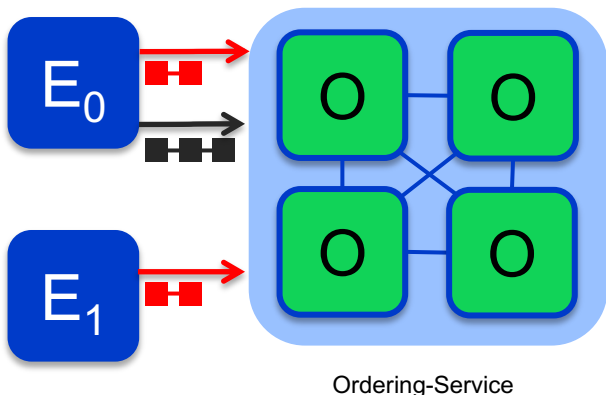
– **Kafka** : Crash fault tolerant consensus

- 3 nodes minimum
- Odd number of nodes recommended

Channels

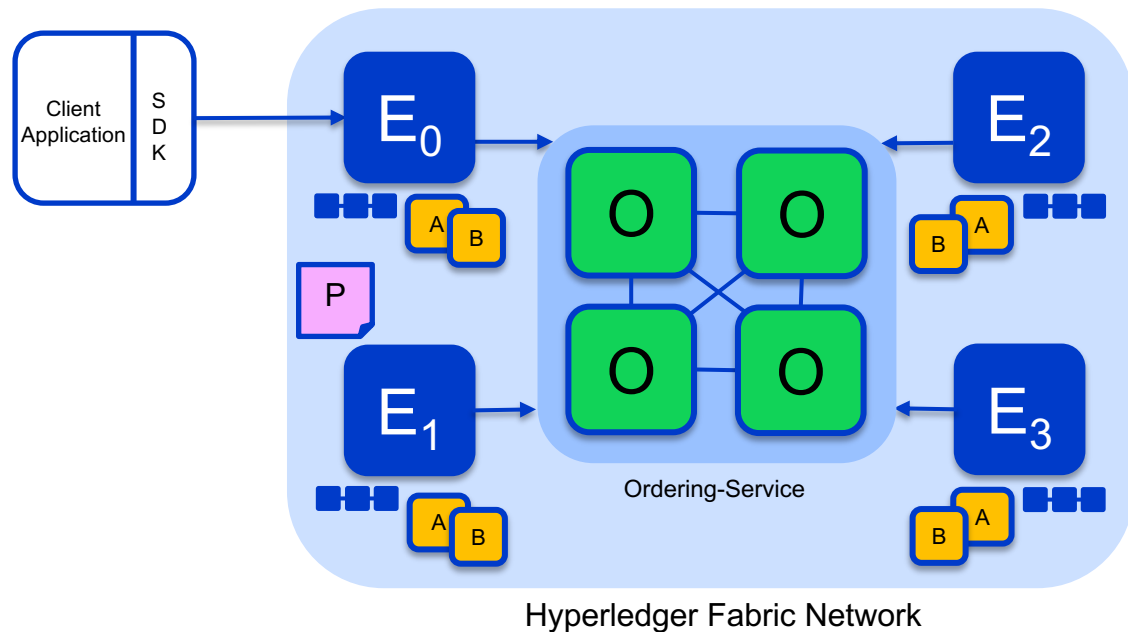
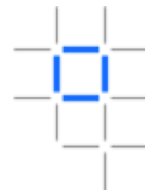


Channels provide privacy between different ledgers



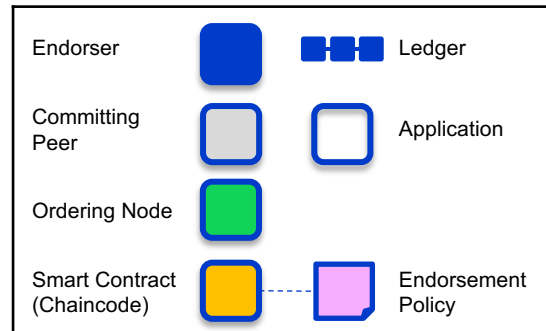
- Ledgers exist in the scope of a channel
 - Channels can be shared across an entire network of peers
 - Channels can be permissioned for a specific set of participants
- Chaincode is **installed** on peers to access the worldstate
- Chaincode is **instantiated** on a specific channel
- Peers can participate in multiple channels
- Concurrent execution for performance and scalability

Single Channel Network

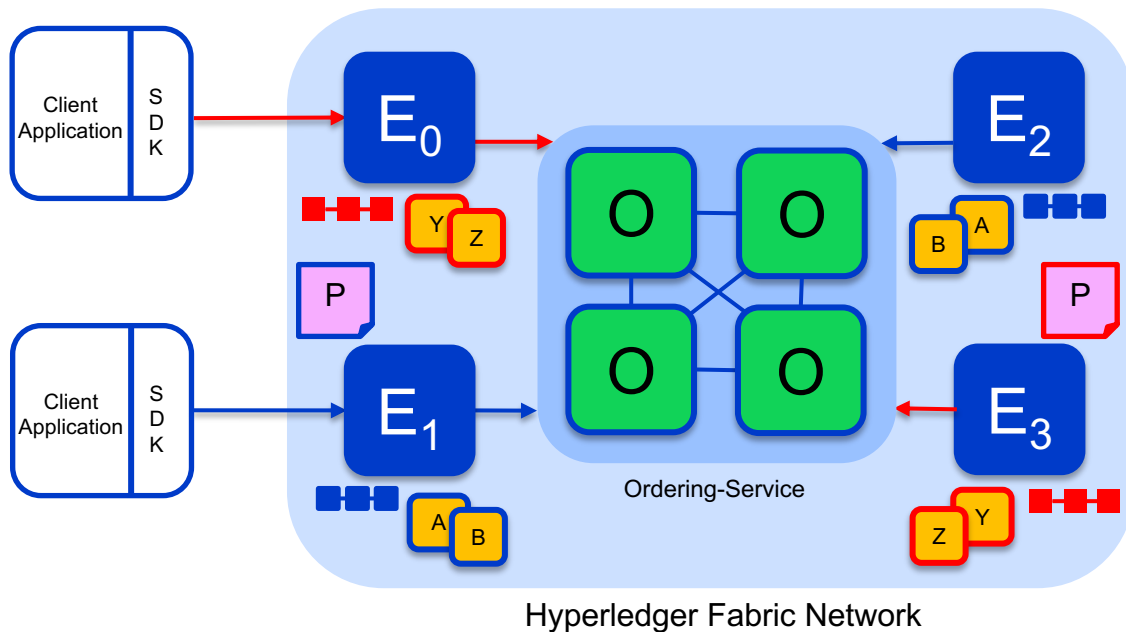
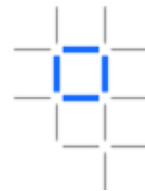


- All peers connect to the same system channel (**blue**).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3

Key:

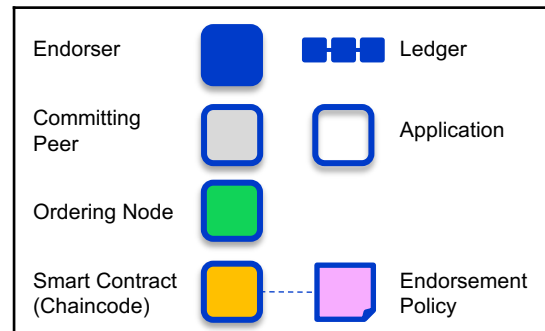


Multi Channel Network



- Peers E_0 and E_3 connect to the **red** channel for chaincodes **Y** and **Z**
- Peers E_1 and E_2 connect to the **blue** channel for chaincodes **A** and **B**

Key:



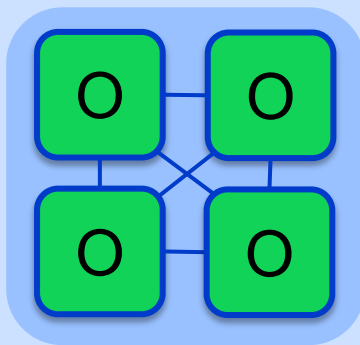
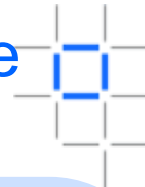


Technical Deep Dive

- Network Consensus
- Channels and Ordering Service
- [Network setup]
- Endorsement Policies
- Permissioned ledger access
- Pluggable world-state



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



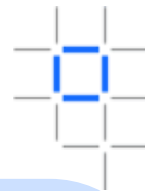
Ordering-Service

Hyperledger Fabric Network

An Ordering Service is configured and started for the network:

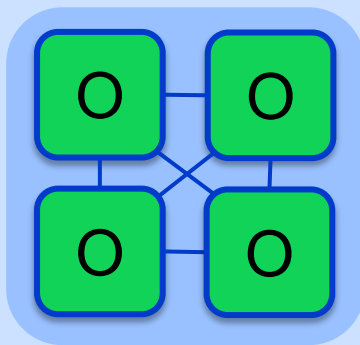
\$ docker-compose [-f orderer.yml] ...

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



E_0

E_1



Ordering-Service

E_2

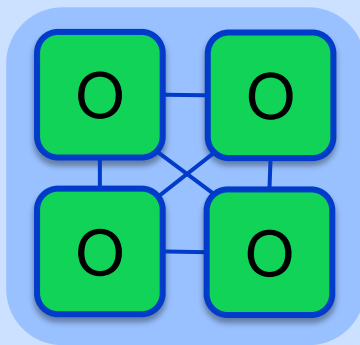
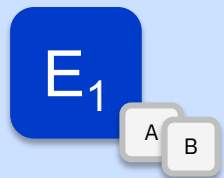
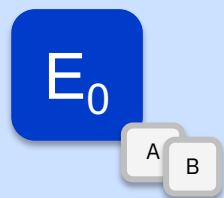
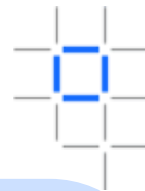
E_3

Hyperledger Fabric Network

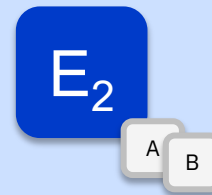
A peer is configured and started for each Endorser or Committer in the network:

\$ peer node start ...

Bootstrap Network (3/6) - Install Chaincode



Ordering-Service

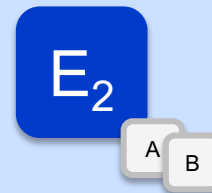
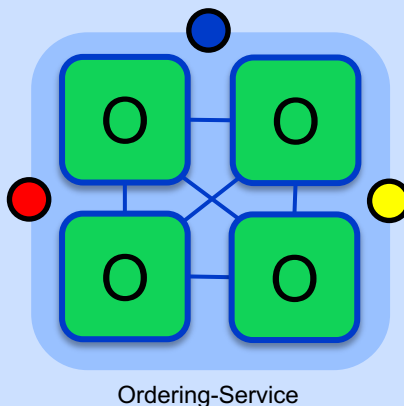
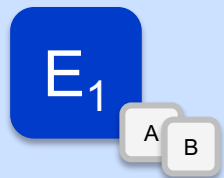
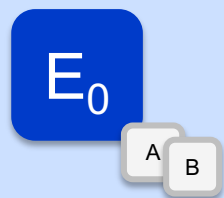
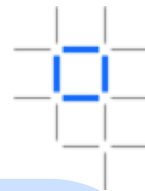


Hyperledger Fabric Network

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

\$ peer chaincode install ...

Bootstrap Network (4/6) – Create Channels

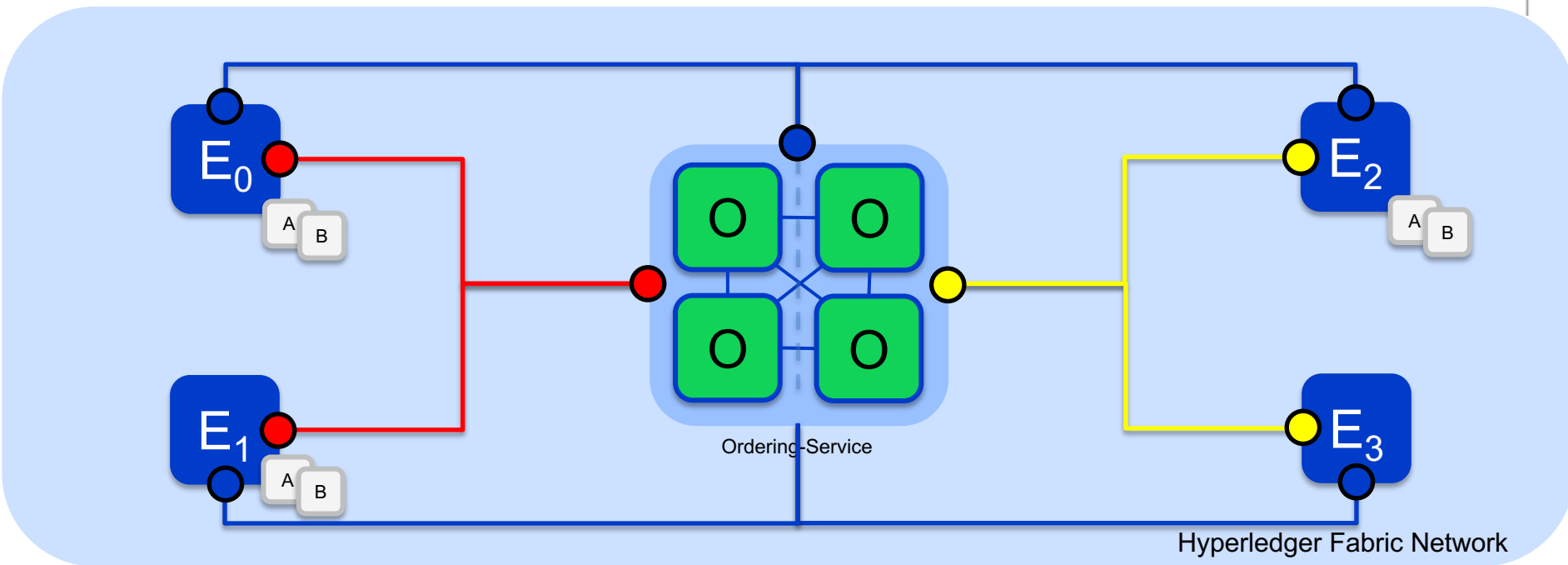
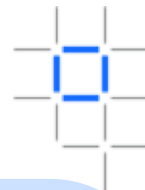


Hyperledger Fabric Network

Channels are created on the ordering service:

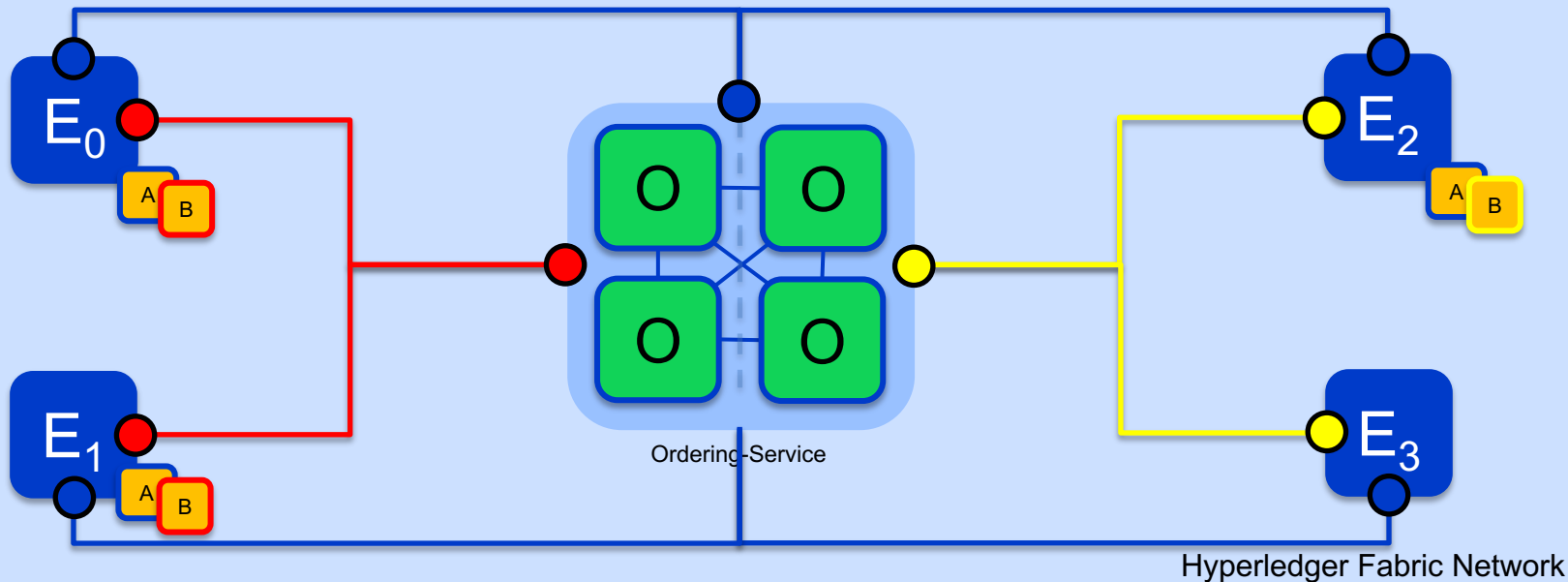
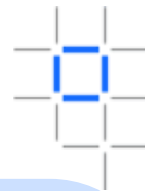
\$ peer channel create -o [orderer] ...

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'

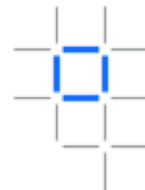


Technical Deep Dive

- Network Consensus
- Channels and Ordering Service
- Network setup
- [Endorsement Policies]
- Permissioned ledger access
- Pluggable world-state

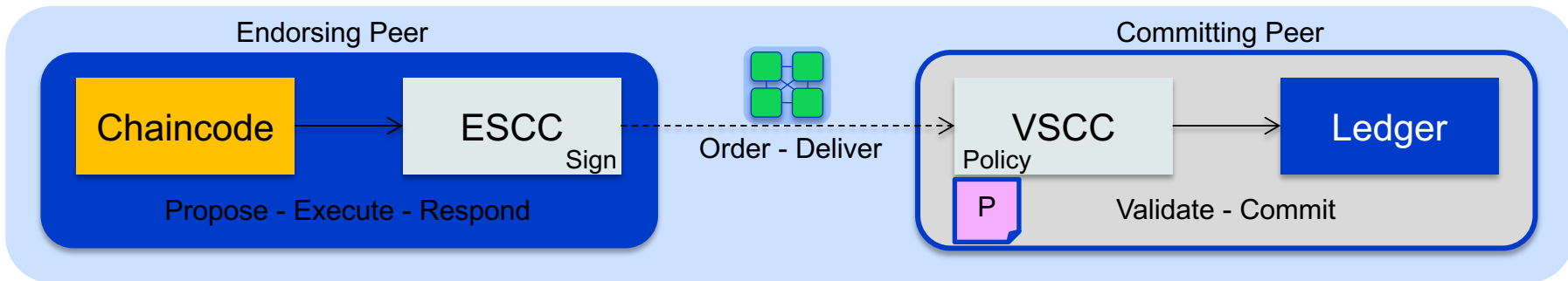


Endorsement Policies

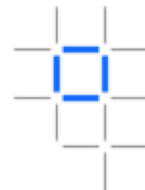


An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

- 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...])**

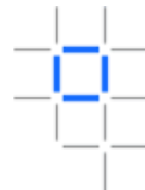
Where **EXPR** is either AND or OR and **E** is either a principal or nested EXPR

Principal Syntax: **MSP.ROLE**

Supported roles are: member and admin

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

Endorsement Policy Examples



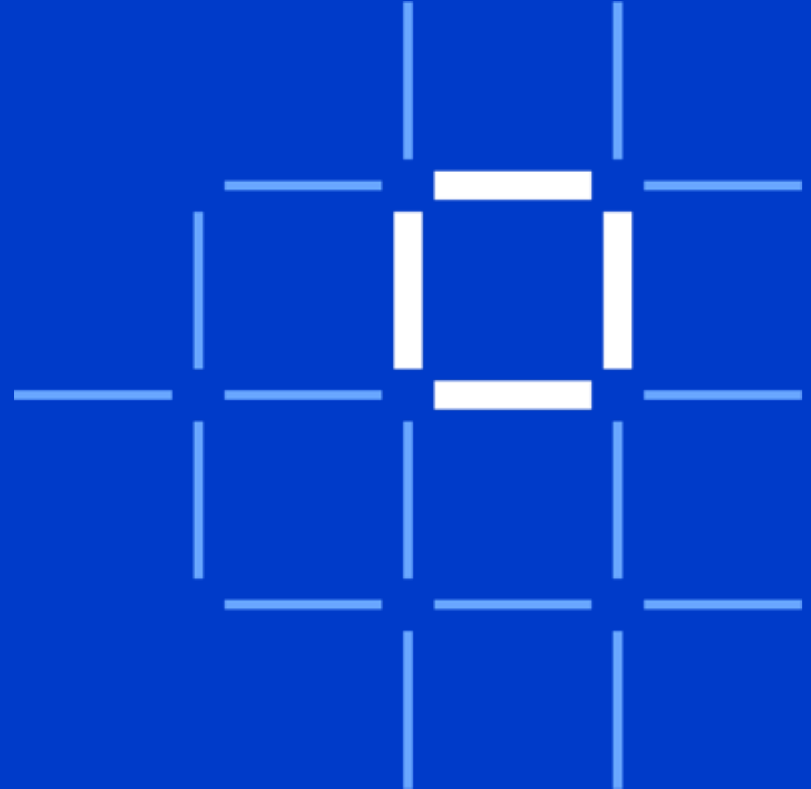
Examples of policies:

- 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'))`

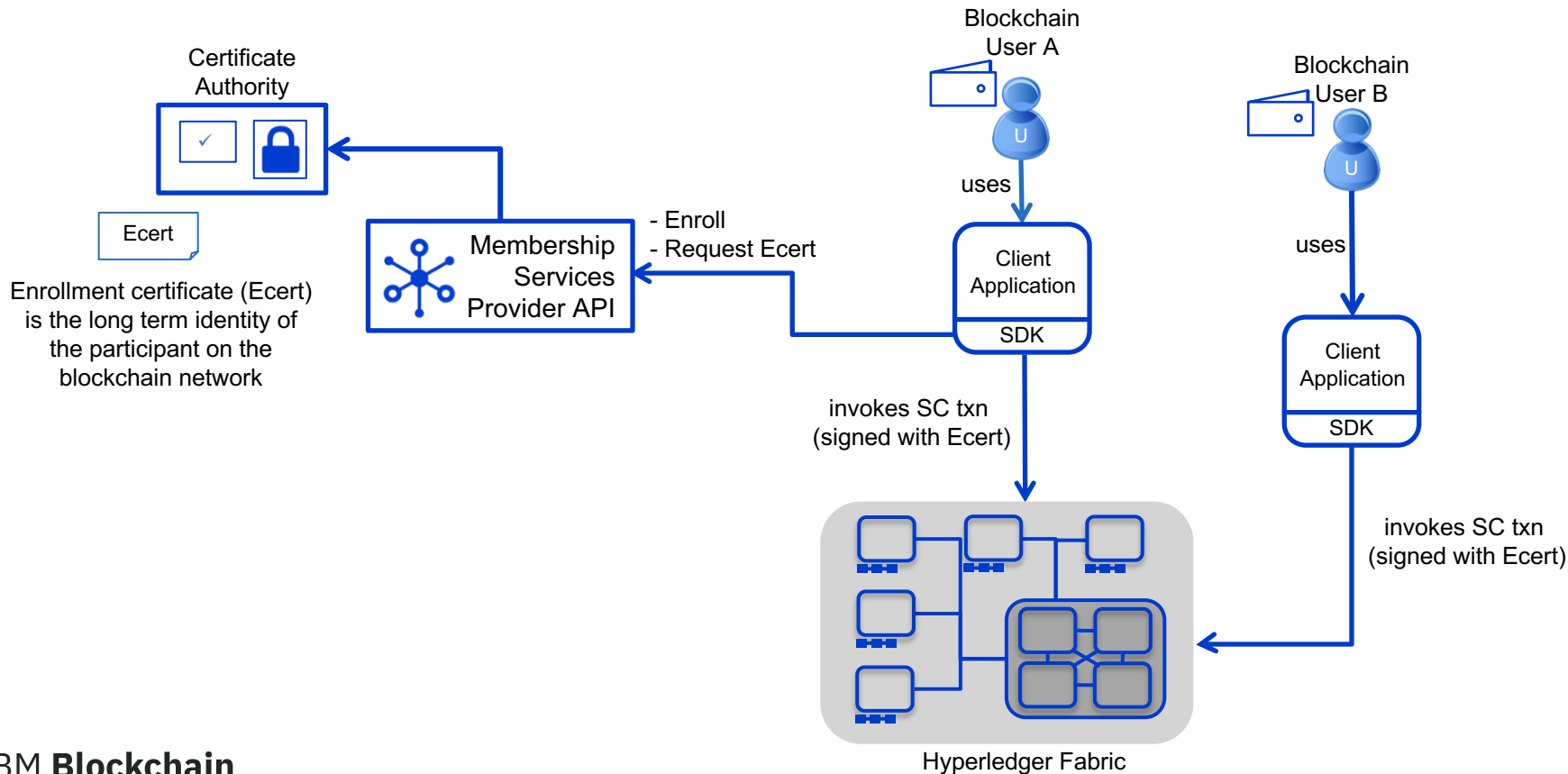
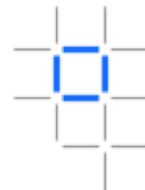


Technical Deep Dive

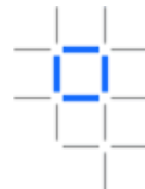
- Network Consensus
- Channels and Ordering Service
- Network setup
- Endorsement Policies
- [Permissioned ledger access]
- Pluggable world-state



Membership Services Overview

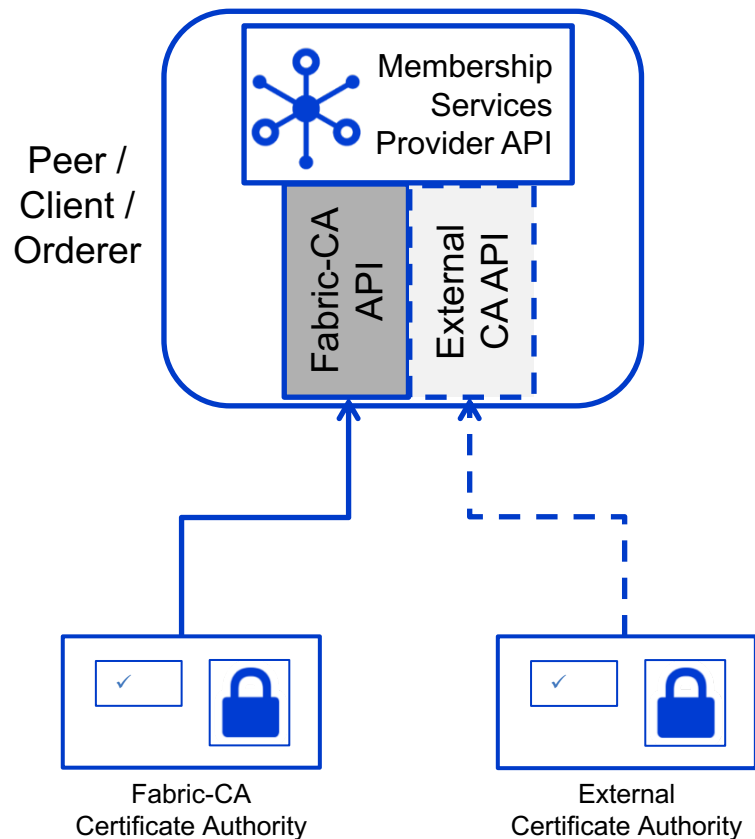
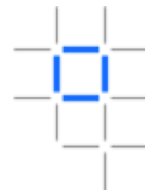


Transaction and Identity Privacy



- Enrollment Certificates, Ecerts
 - Long term identity
 - Can be obtained offline, bring-your-own-identity
- Permissioned Interactions
 - Users sign with their Ecert
- Membership Services
 - Abstract layer to credential providers

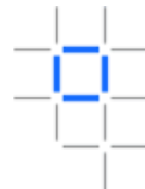
Membership Services Provider API



Membership Services Provider API

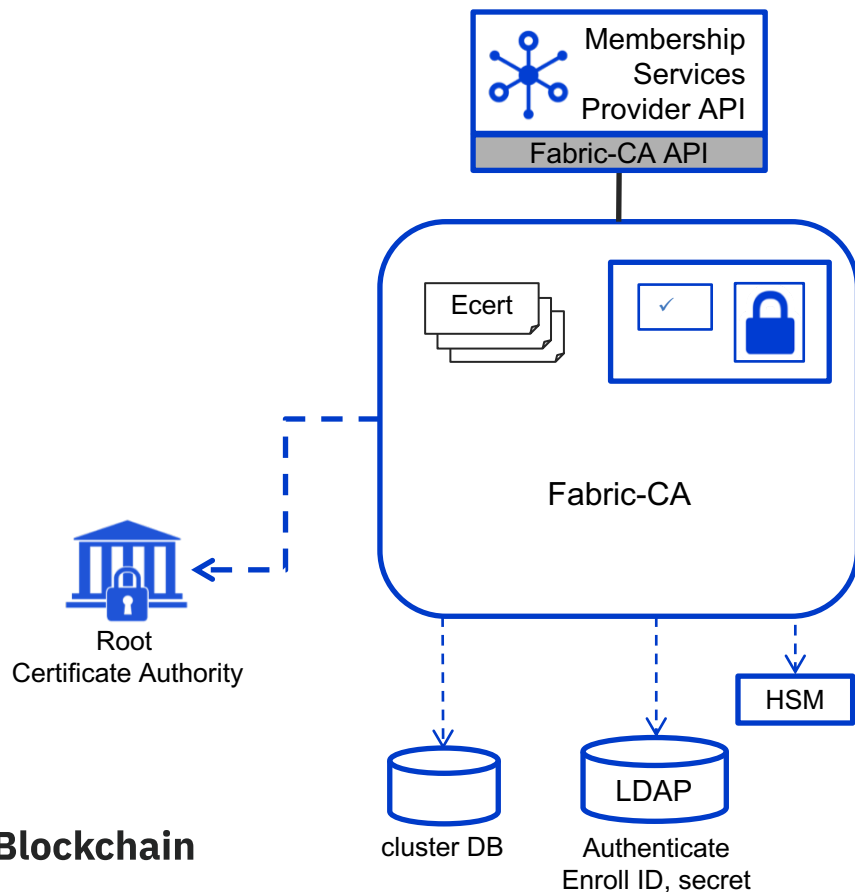
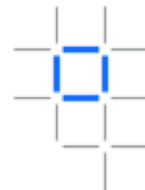
- Pluggable interface supporting a range of credential architectures
- Default implementation calls Fabric-CA.
- Governs identity for Peers and Users.
- Provides:
 - User authentication
 - User credential validation
 - Signature generation and verification
 - Optional credential issuance
- Additional offline enrollment options possible (eg File System).

Membership Services Provider (MSP)



- An abstraction to represent a membership authority and its operations on issuing and management of Hyperledger Fabric membership credentials in a modular & pluggable way
 - Allows for the co-existence of a variety of credential management architectures
 - Allows for easy organizational separation in credential management/administration operations according to business rules at a technical level
 - Potential to smoothly easily support different standards and membership implementations
 - Easy and straight-forward interface that the core can understand
- Described by a generic interface to cover:
 - User credential validation
 - User (anonymous but traceable) authentication: signature generation and verification
 - User attribute authentication: attribute ownership proof generation, and verification
 - (optionally) User credential issue

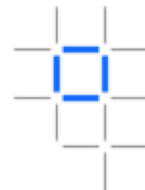
Fabric-CA Details



Fabric-CA

- Default implementation of the Membership Services Provider Interface.
- Issues Ecerts (long-term identity)
- Supports clustering for HA characteristics
- Supports LDAP for user authentication
- Supports HSM

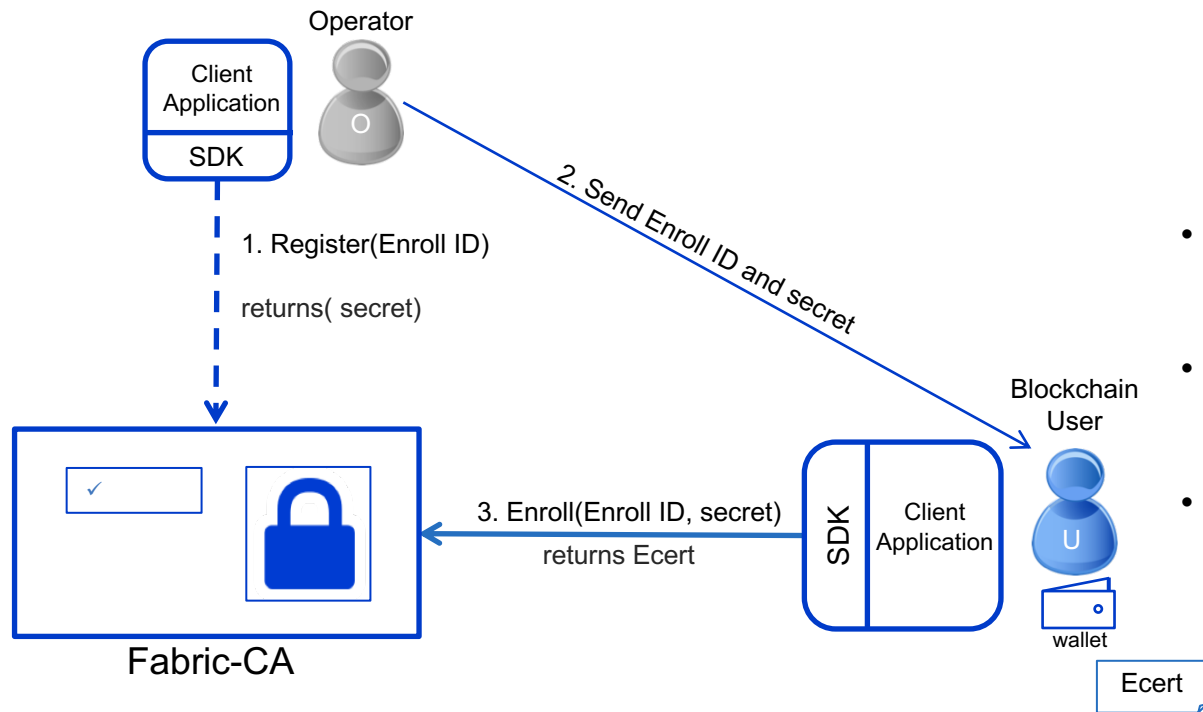
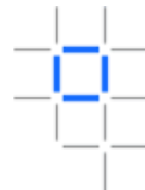
Fabric-CA



Certificate Authority

- Issues Ecerts and manages renewal and revocation
- Supports:
 - Clustering for HA characteristics
 - LDAP server for registration and enrollment
 - Hardware Security Modules

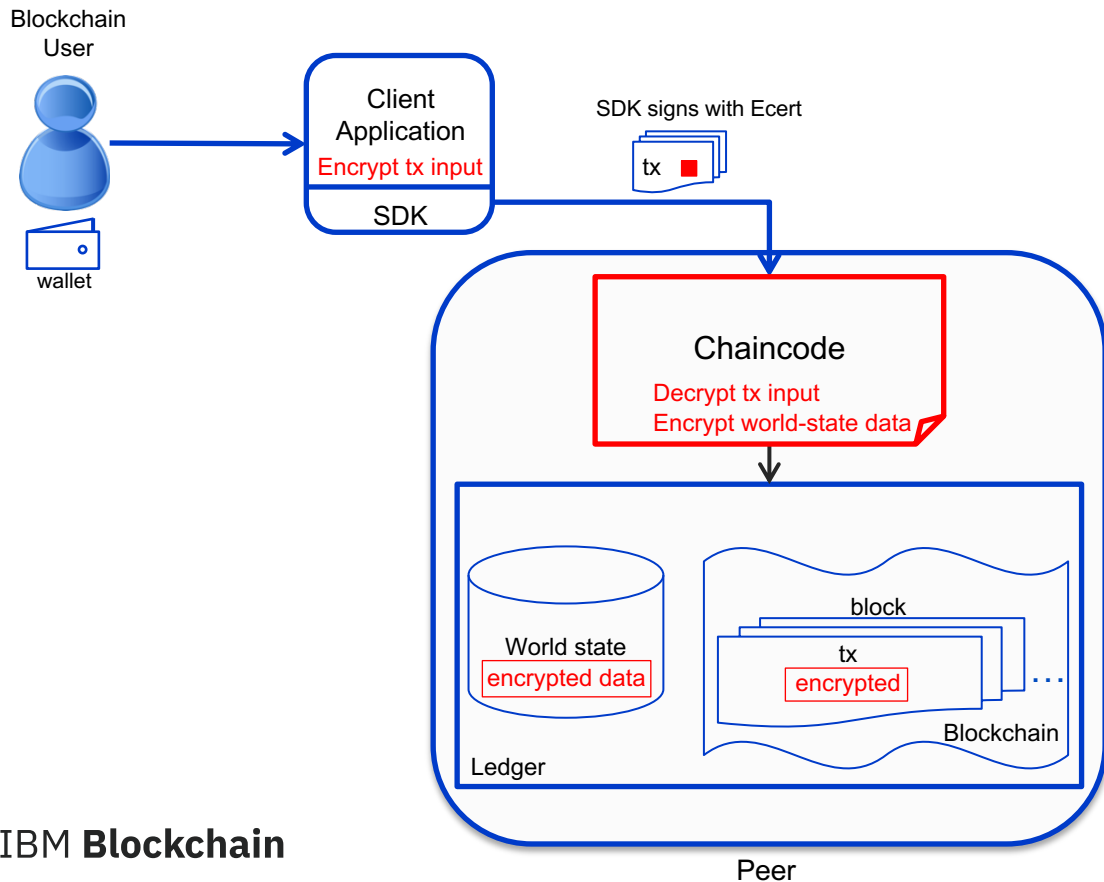
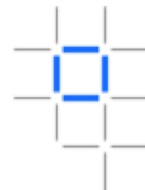
New User Registration and Enrollment



Registration and Enrollment

- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- Additional offline registration and enrollment options available

Application Level Encryption



Data Encryption

Handled in the application domain.

Multiple options for encrypting:

- Transaction Data
- Chaincode*
- World-State data

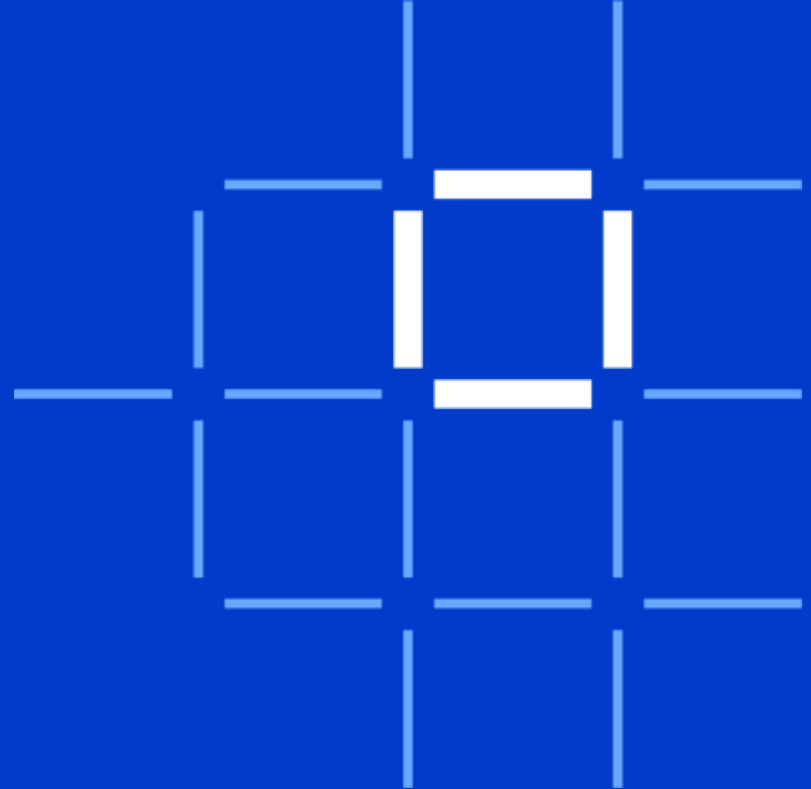
Chaincode optionally deployed with cryptographic material, or receive it in the transaction from the client application using the transient data field (not stored on the ledger).

*Encryption of application chaincode requires additional development of system chaincode.

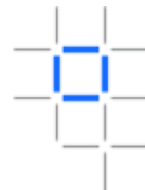


Technical Deep Dive

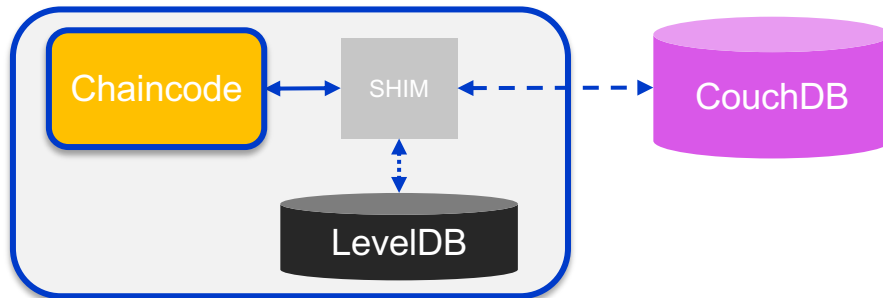
- Network Consensus
- Channels and Ordering Service
- Network setup
- Endorsement Policies
- Permissioned ledger access
- [Pluggable world-state]



WorldState Database



- Pluggable worldstate database
- Default embedded key/value implementation using LevelDB
 - Support for keyed queries, but cannot query on value
- Support for Apache CouchDB
 - Full query support on key and value (JSON documents)
 - Meets a large range of chaincode, auditing, and reporting requirements
 - Will support reporting and analytics via data replication to an analytics engine such as Spark (future)
 - Id/document data model compatible with existing chaincode key/value programming model



Thank you

Barry Silliman

silliman@us.ibm.com

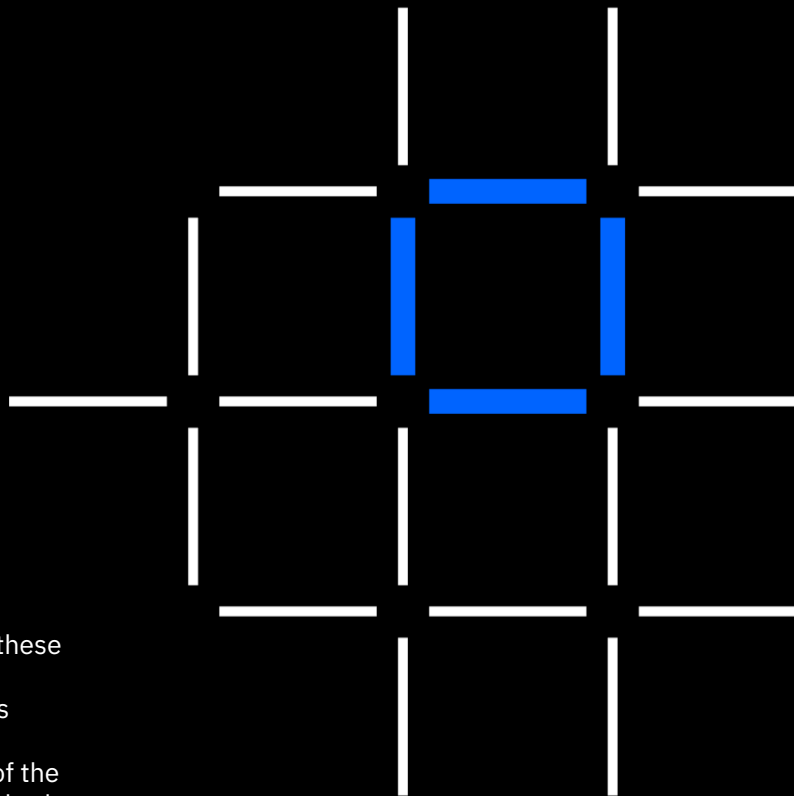
IBM Blockchain

www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org

© Copyright IBM Corporation 2017. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represents only goals and objectives. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.



IBM

