Explained Composed Solutions Architected **Explored**

IBM

Blockchain
Explored

V3.02, 25 April 2017

Contents

HYPERLEDGER FABRIC

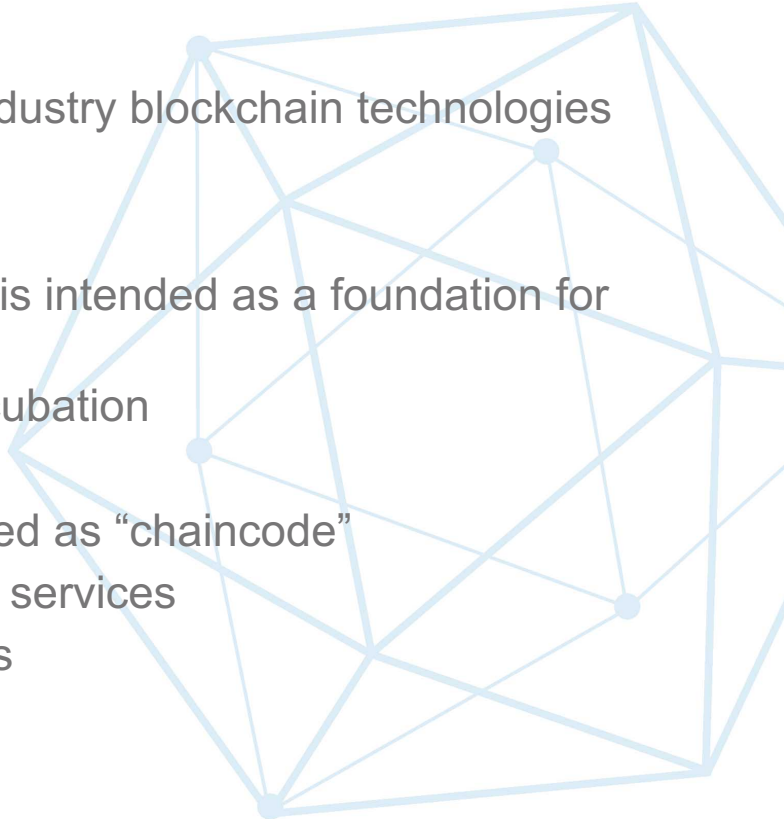
- Project status and roadmap
- Technical Deep Dive

What is Hyperledger Fabric

- **Linux Foundation Hyperledger**
  - A collaborative effort created to advance cross-industry blockchain technologies

- **Hyperledger Fabric**
  - An implementation of blockchain technology that is intended as a foundation for developing blockchain applications
  - The first Hyperledger project to graduate from incubation
  - Key technical features:
    - A shared ledger and smart contracts implemented as “chaincode”
    - Privacy and permissioning through membership services
    - Modular architecture and flexible hosting options

- IBM is one contributor to Hyperledger Fabric



Hyperledger Fabric Roadmap

Hack Fest docker images

- **60 participants tested**
- Basic v1 architecture in place
- Add / Remove Peers
- Channels
- Node SDK
- Go Chaincode
- Ordering Solo
- Fabric CA

V1 Alpha *

- Docker images
- Tooling to bootstrap network
- Fabric CA or bring your own
- Java and Node 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
- Java Chaincode
- 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 Next *

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

2016/17 December | March | June | Future

Connect-a-thon

- 11 companies in Australia, Hungary, UK, US East Coast, US West Coast, Canada dynamically added peers and traded assets

Connect-a-cloud

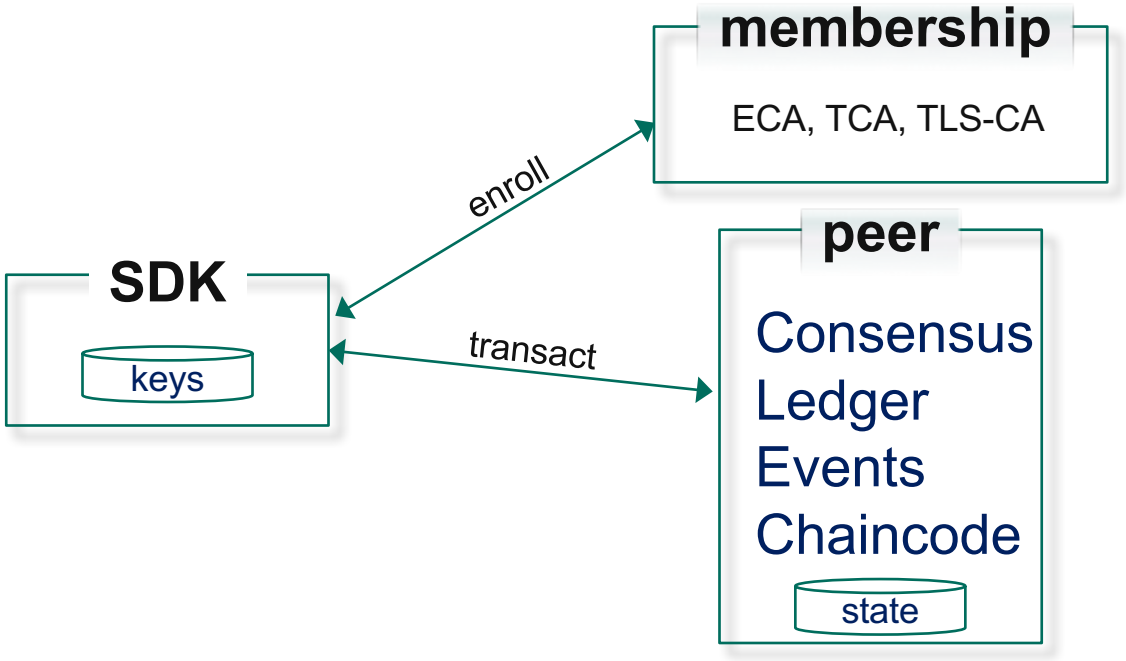
- Dynamically connecting OEM hosted cloud environments to trade assets

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

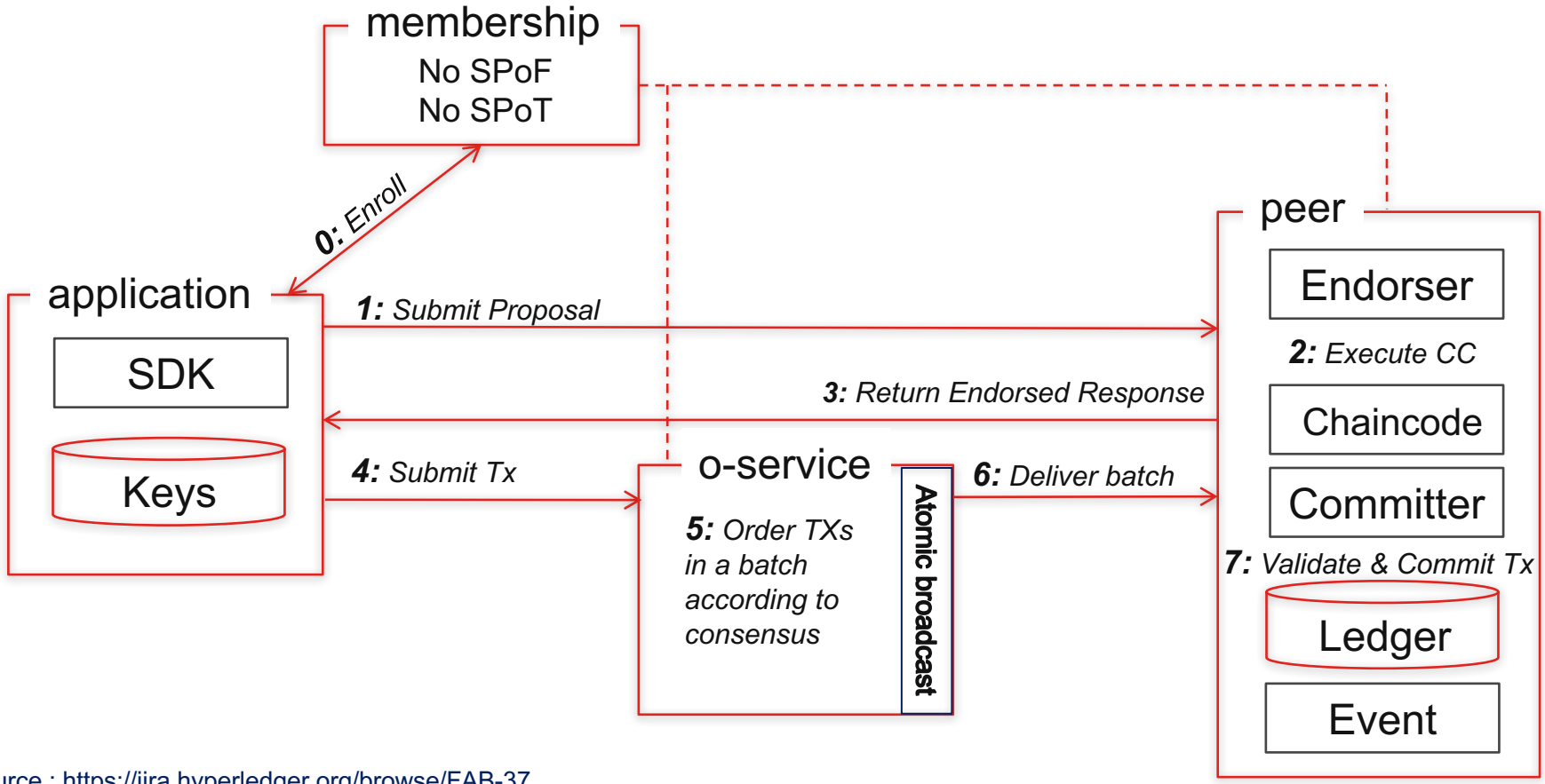
*** Dates for Alpha, Beta, and GA are determined by Hyperledger community and are currently proposals.**

Proposed Alpha detailed content:
https://wiki.hyperledger.org/projects/proposedv1alphacontent

Architecture of Hyperledger Fabric v0.6



Architecture of Hyperledger Fabric v1



Source : https://jira.hyperledger.org/browse/FAB-37

Overview of Hyperledger Fabric v1 – Lessons Learned

- Better reflect business processes by specifying who endorses transactions
- Support broader regulatory requirements for privacy and confidentiality
- Scale the number of participants and transaction throughput
- Eliminate non deterministic transactions
- Support rich data queries of the ledger
- Dynamically upgrade fabric and chaincode
- Support for multiple credential and cryptographic services for identity
- Support for ”bring your own identity”

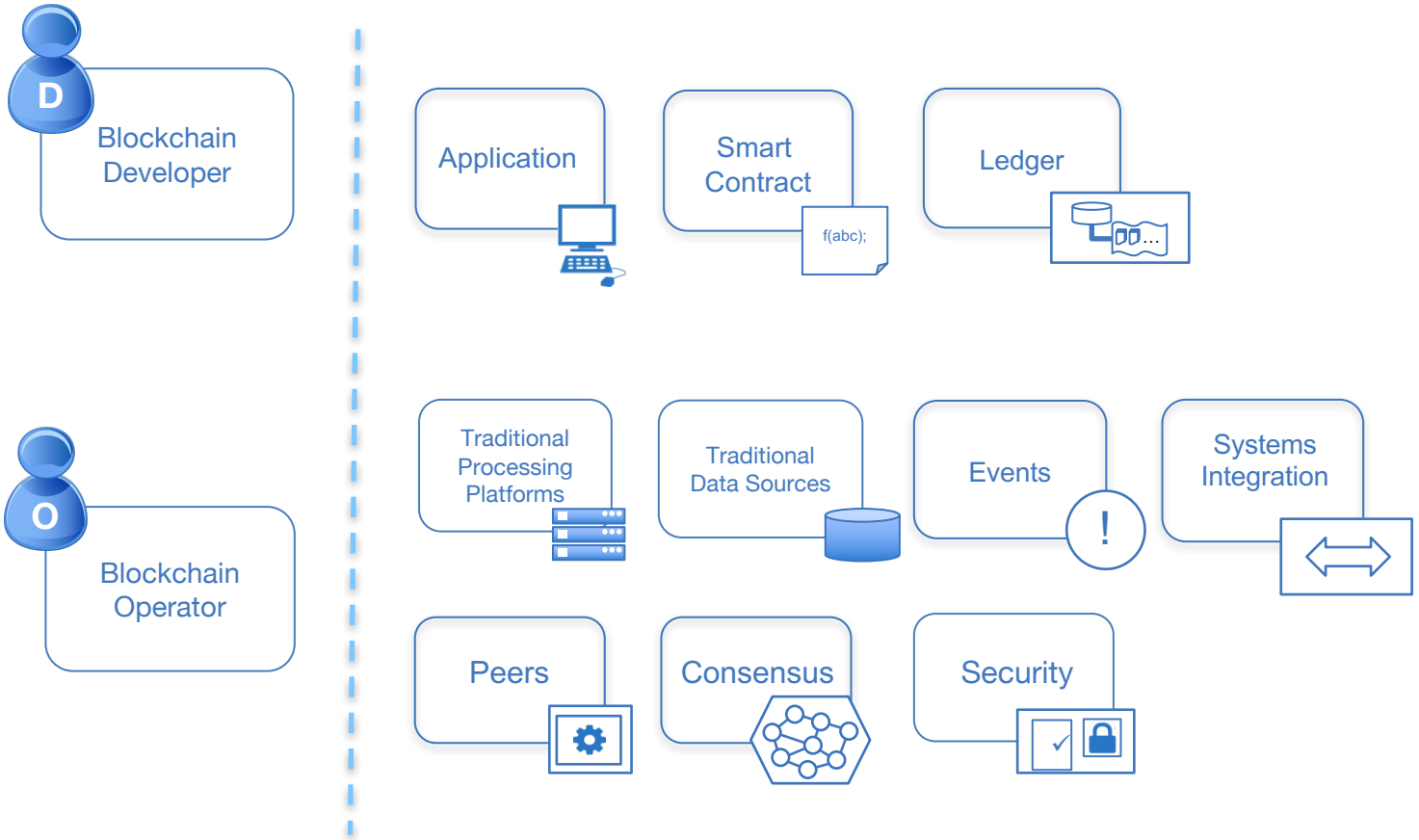
Contents

HYPERLEDGER FABRIC

Project status and roadmap

Technical Deep Dive

Recall key blockchain concepts



Hyperledger Fabric V1 - Deep Dive Topics

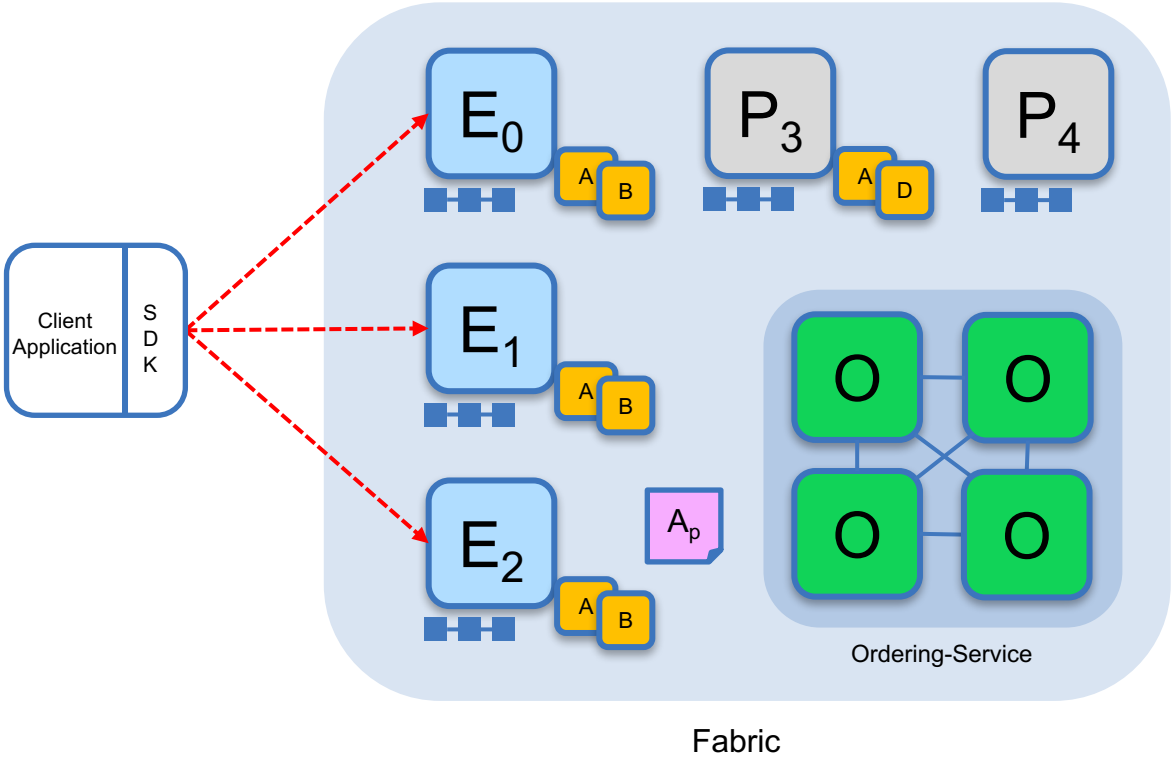
- Network Consensus
- Channels and Ordering Service
- Fabric network setup
- Endorsement Policies
- Permissioned ledger access
- Pluggable world-state

Network Consensus

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 Nodes (service)**: 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

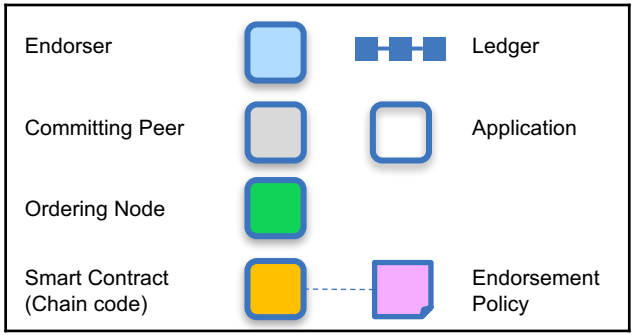


Application proposes transaction

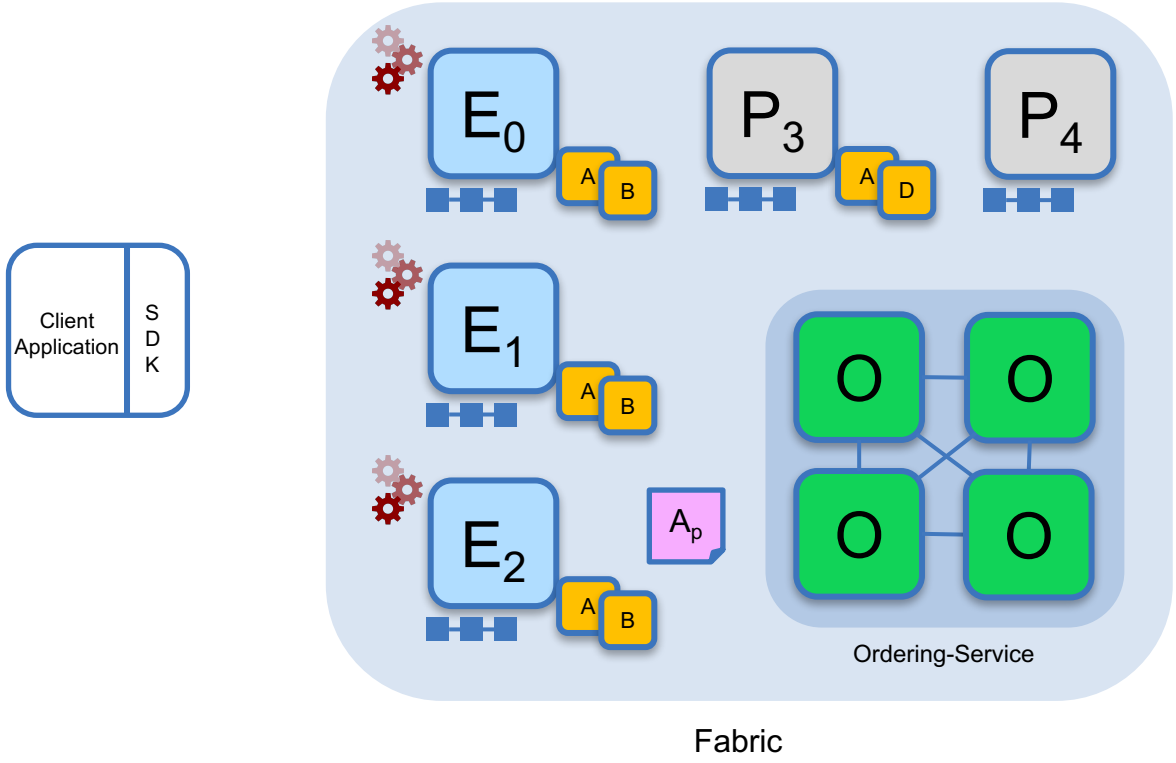
Endorsement policy:
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

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

Key:



Sample transaction: Step 2/7 – Execute proposal



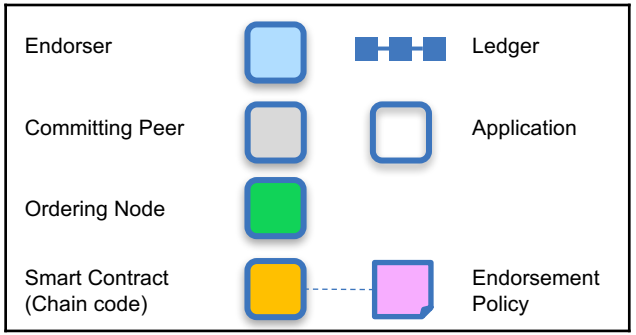
Endorsers Execute Proposals

E_0, E_1 & E_2 will each execute the *proposed* transaction. None of these executions will update the ledger

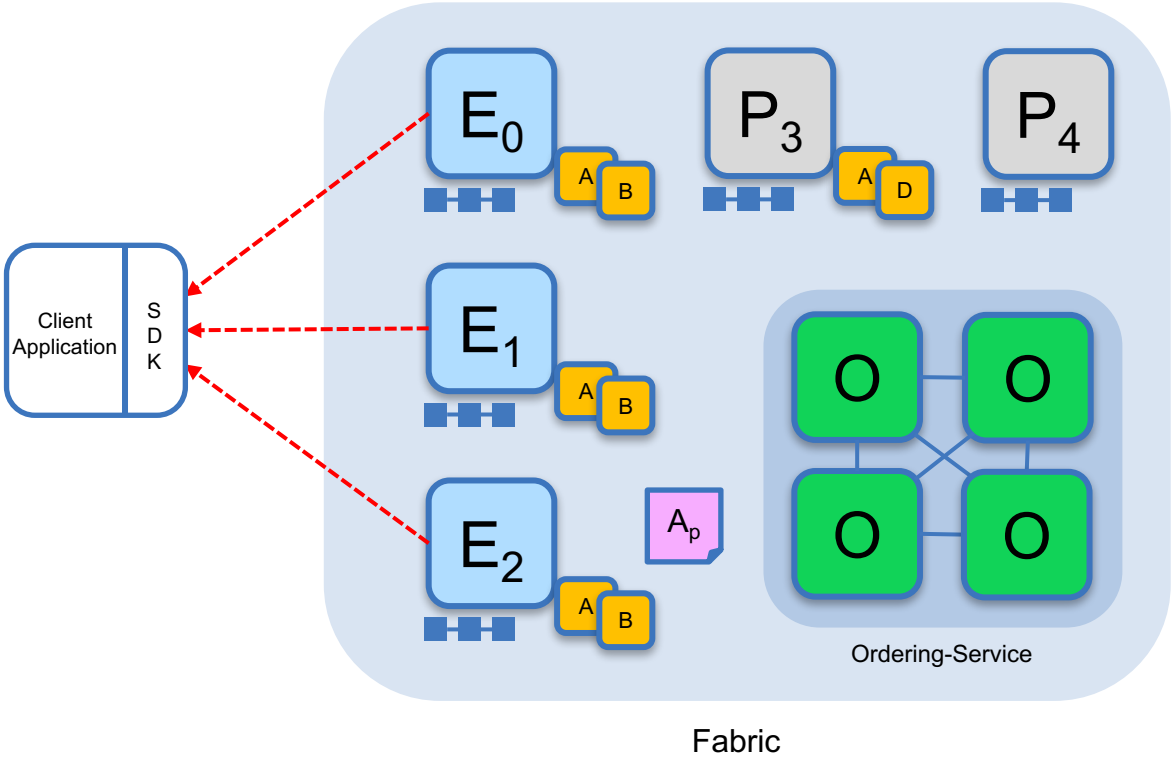
Each execution will capture the set of **R**ead and **W**ritten data, called **RW sets,** which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



Sample transaction: Step 3/7 – Proposal Response



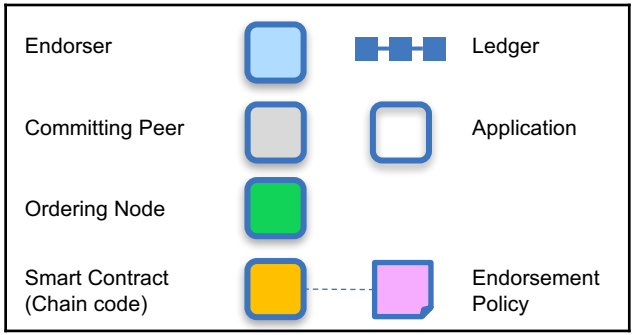
Application receives responses

RW sets are asynchronously returned to application

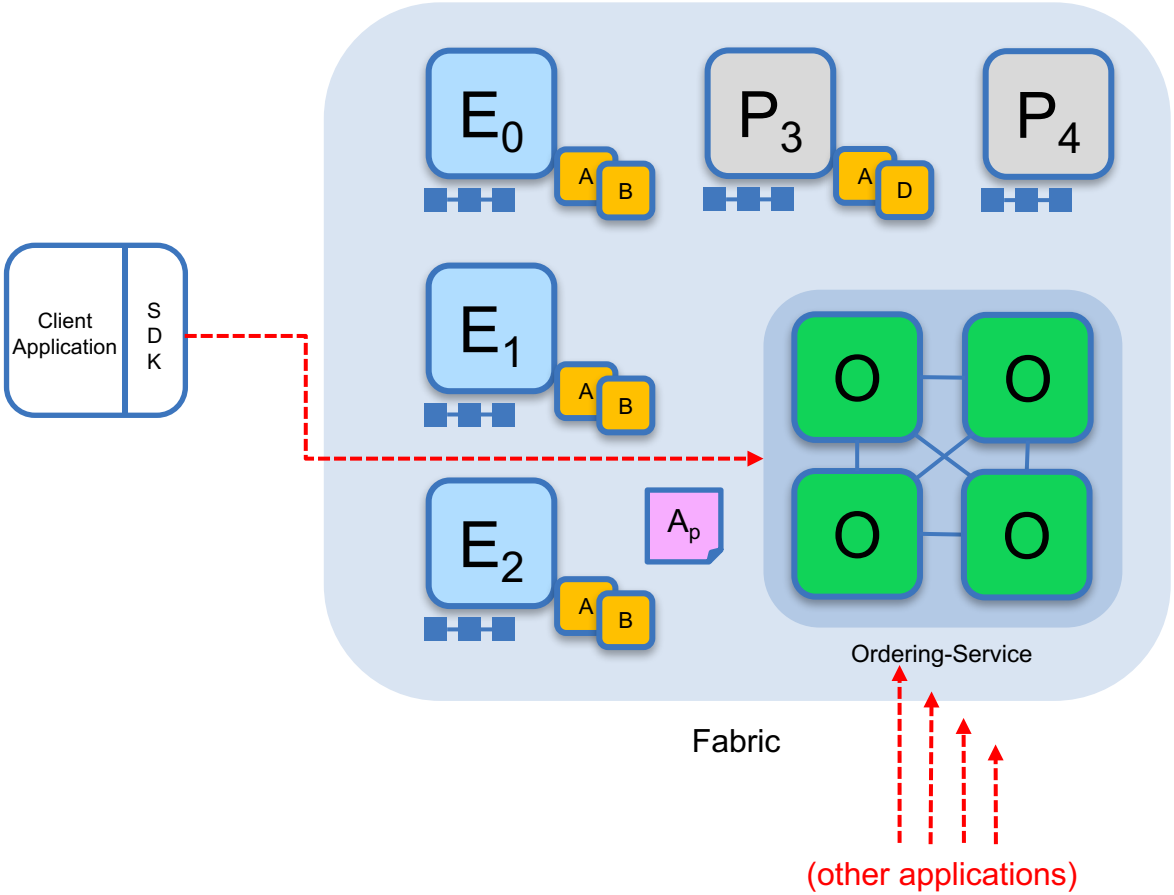
The RW sets are signed by each endorser, and also includes each record version number

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

Key:



Sample transaction: Step 4/7 – Order Transaction

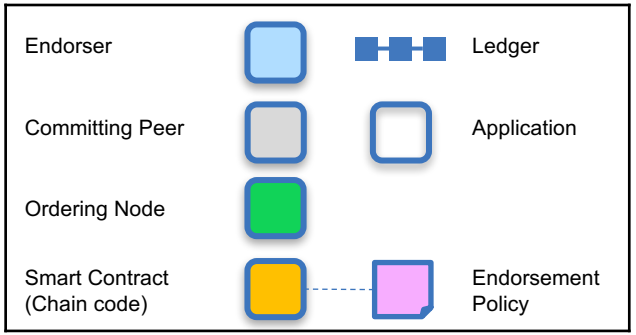


Application submits responses for ordering

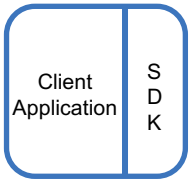
Application submits responses as a **transaction** to be ordered.

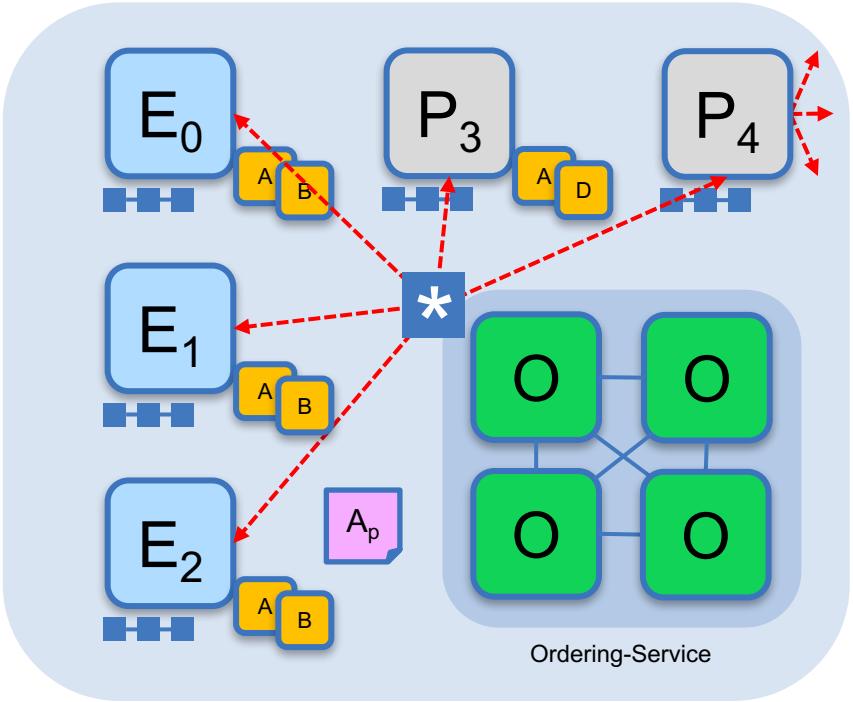
Ordering happens across the fabric in parallel with transactions submitted by other applications

Key:



Sample transaction: Step 5/7 – Deliver Transaction





Fabric

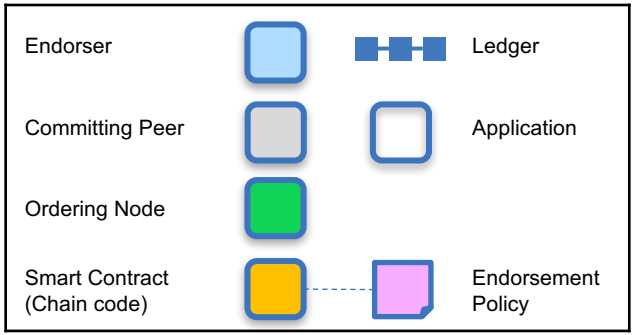
Orderer delivers to all committing peers

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

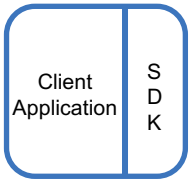
Different ordering algorithms available:

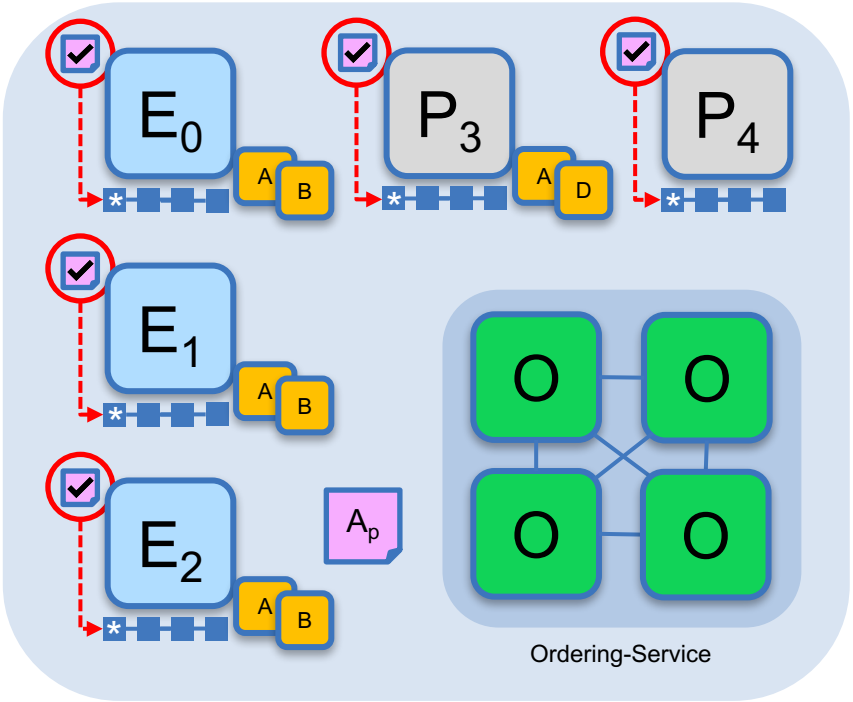
- SOLO (Single node, development)
- Kafka (Crash fault tolerance)
- SBFT (Byzantine fault tolerance)

Key:



Sample transaction: Step 6/7 – Validate Transaction





Fabric

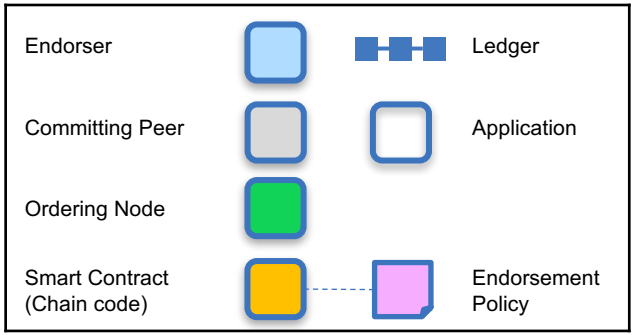
Committing peers validate transactions

Every committing peer validates against the endorsement policy. Also check RW sets are still valid for current world state

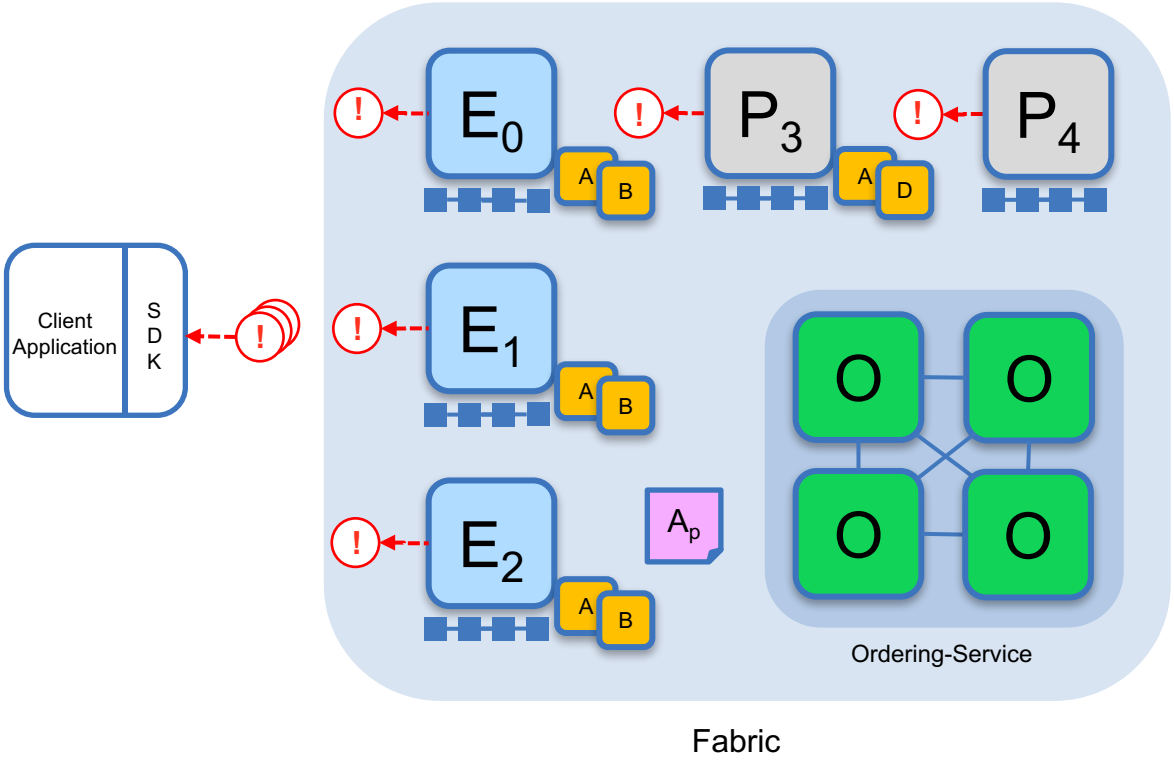
Validated transactions are applied to the world state and retained on the ledger

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

Key:



Sample transaction: Step 7/7 – Notify Transaction

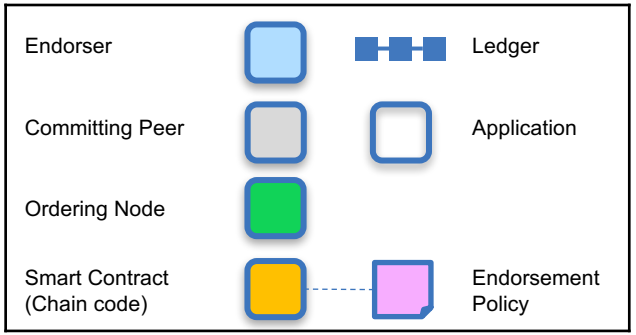


Committing peers notify applications

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

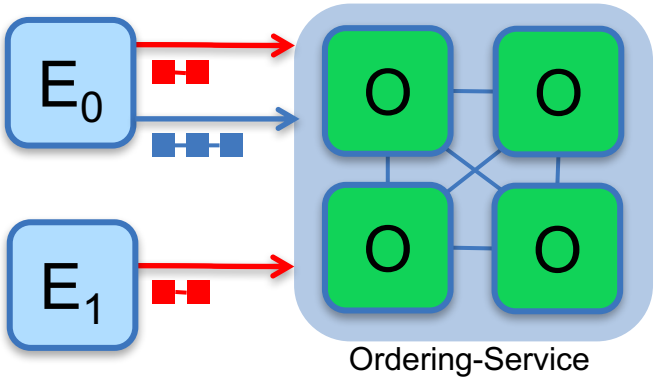
Key:



Channels and Ordering Service

Channels

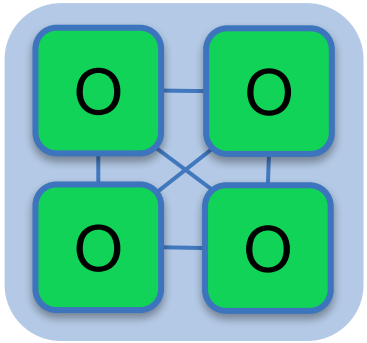
Separate channels isolate transactions on different ledgers



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

Ordering Service

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

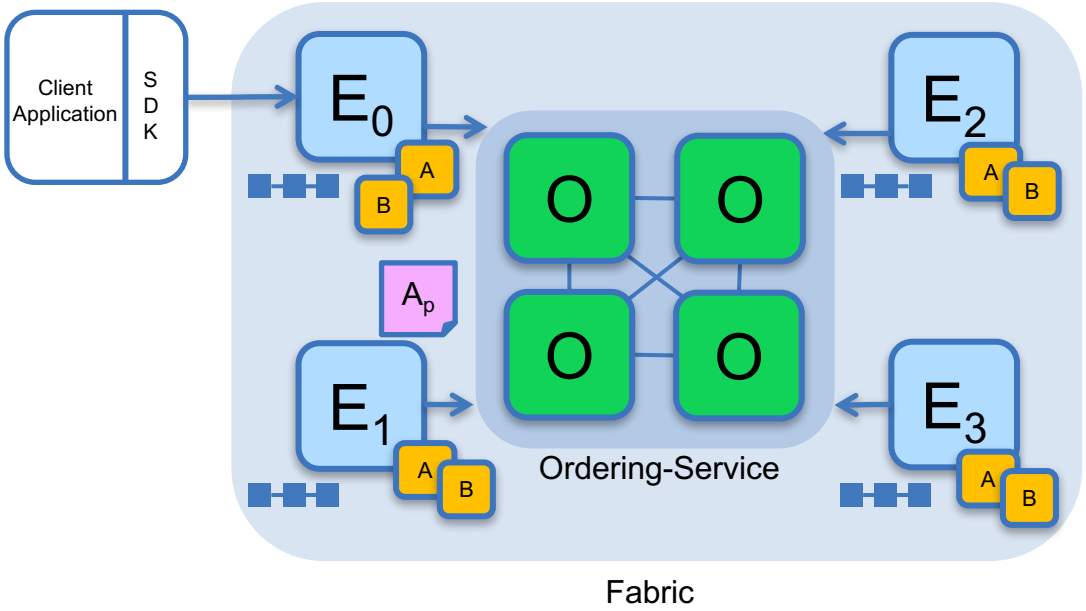


Ordering-Service

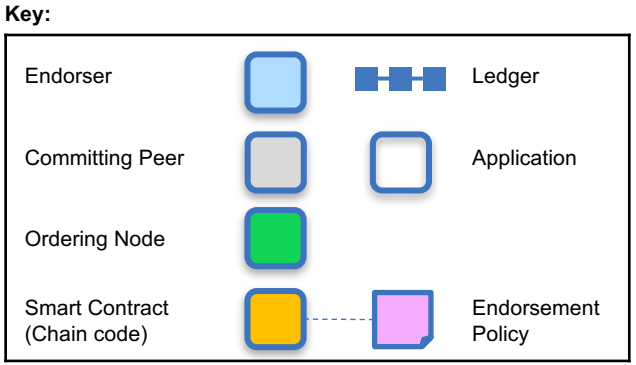
Different configuration options for the ordering service include:

- **SOLO**
  - Single node for development
- **Kafka :** Crash fault tolerant consensus
  - 3:n nodes minimum
  - Odd number of nodes recommended
- **SBFT :** Byzantine fault tolerant consensus
  - 4:n nodes minimum

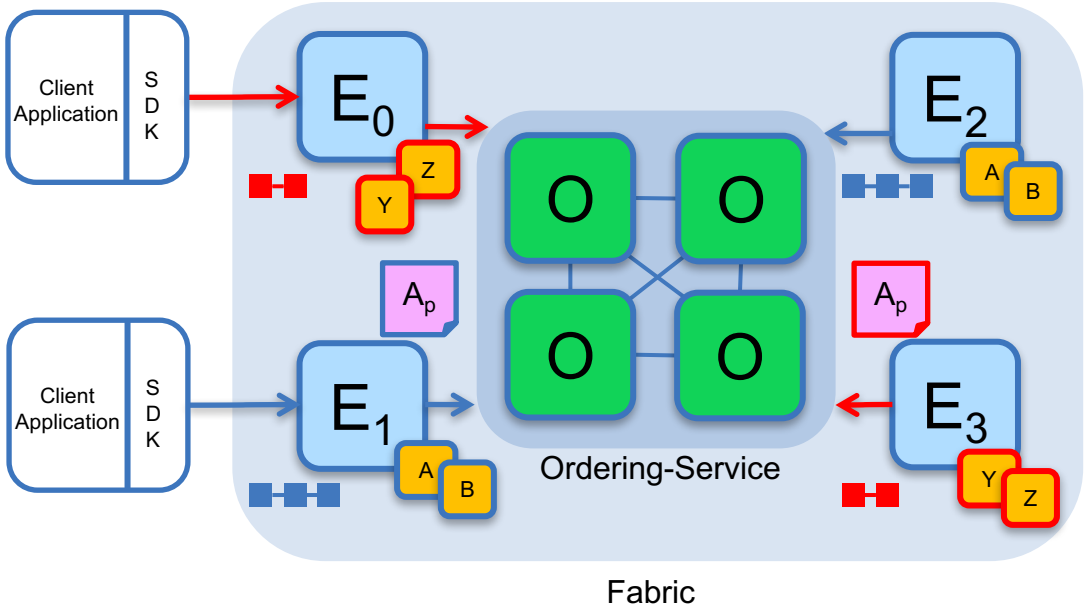
Single Channel Endorsement



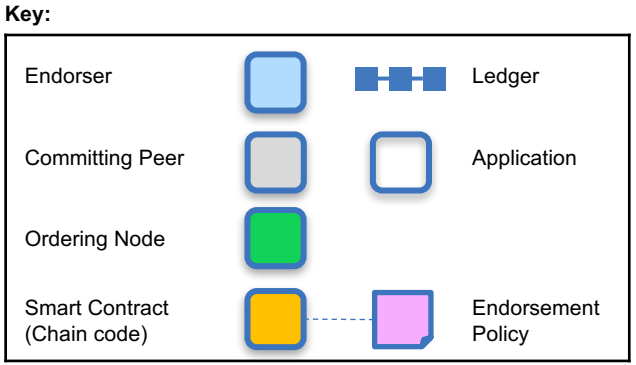
- Similar to 0.6 PBFT model
- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3



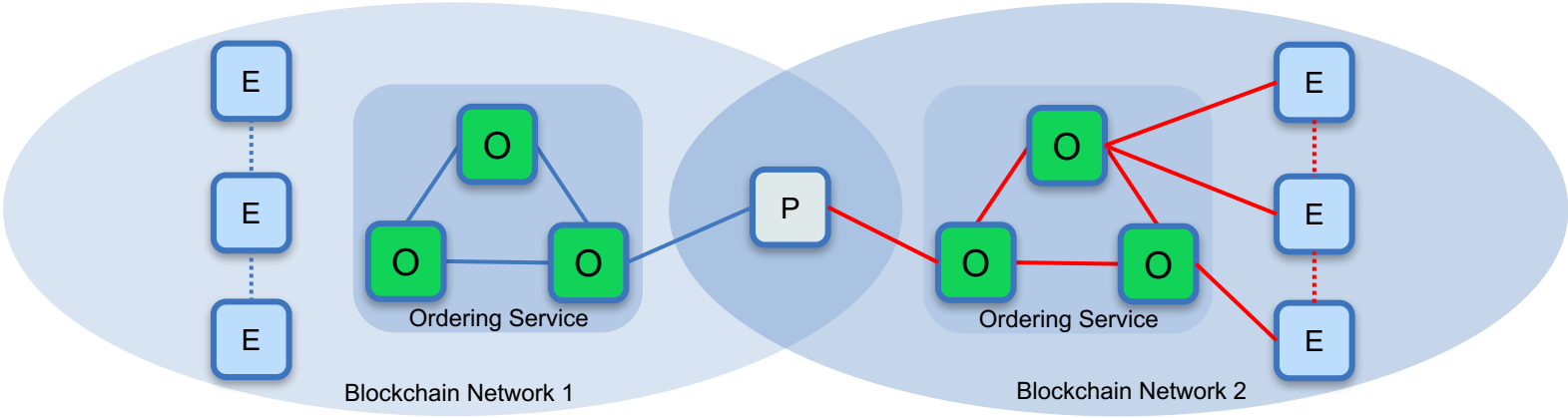
Multi Channel Endorsement



- 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**



Future 2 network topology



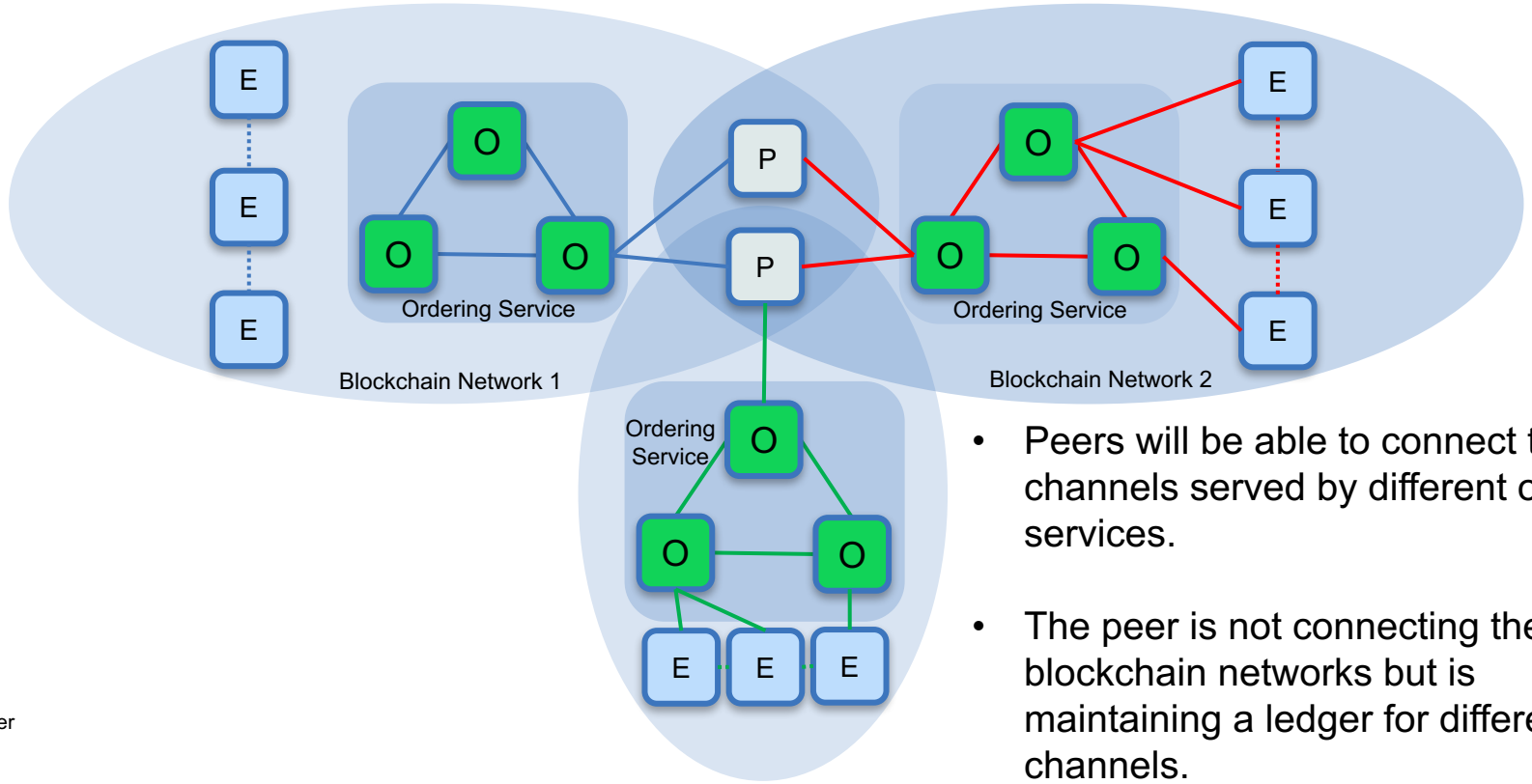
- Peers will be able to connect to channels served by different ordering services.
- The peer is not connecting the three blockchain networks but is maintaining a ledger for different channels.

Peer

Endorser

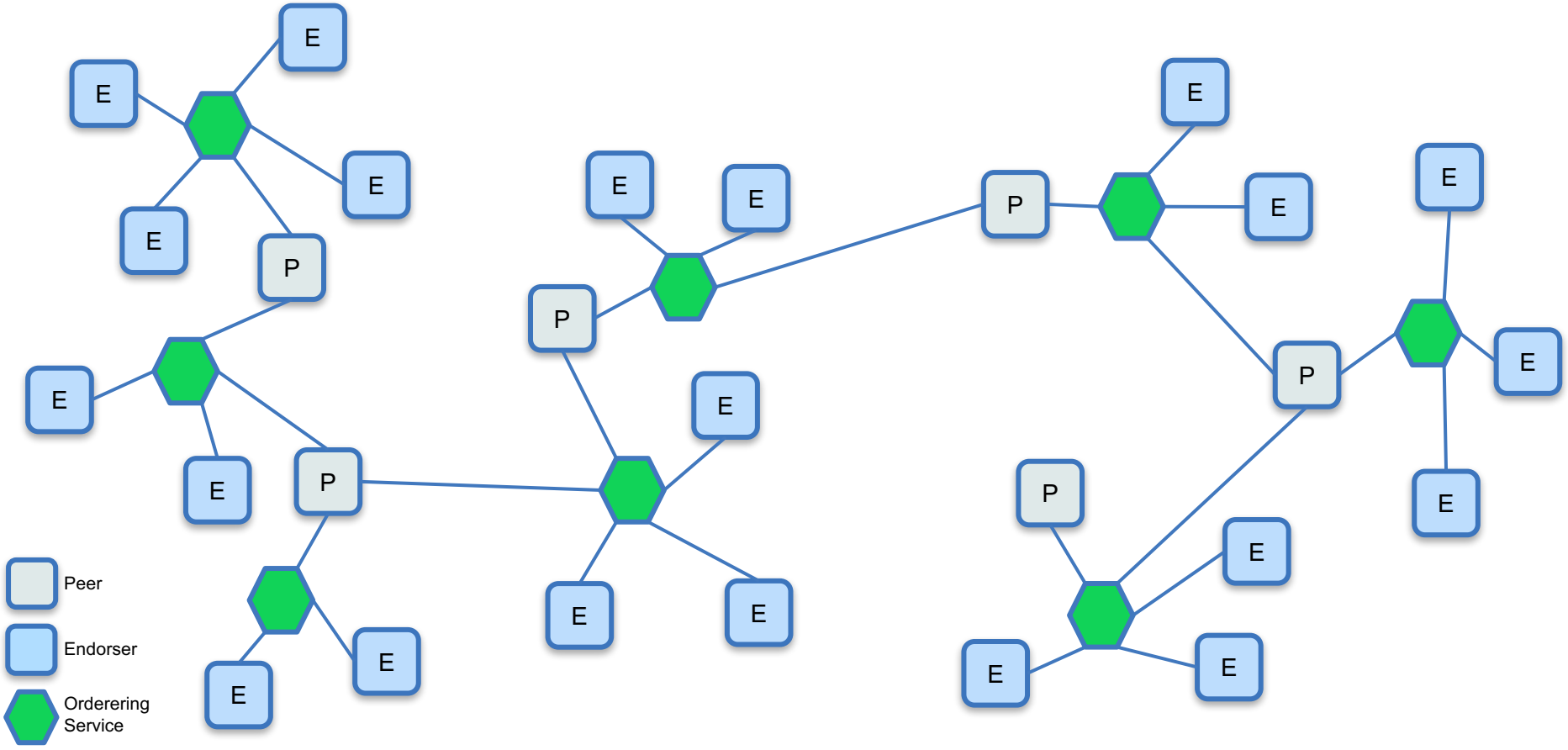
Orderer

Future 3 network topology



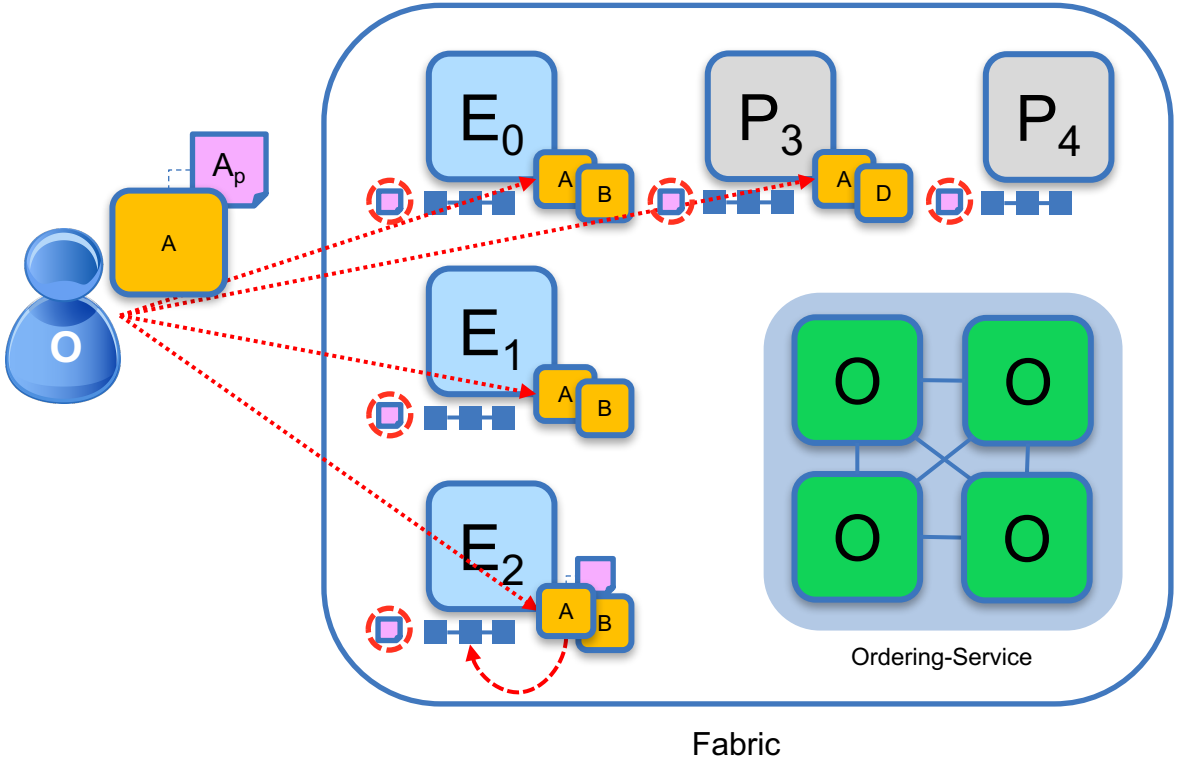
- Peers will be able to connect to channels served by different ordering services.
- The peer is not connecting the three blockchain networks but is maintaining a ledger for different channels.

Future Network of Networks



Fabric Network Setup

Installing and instantiating smart contract chain-code



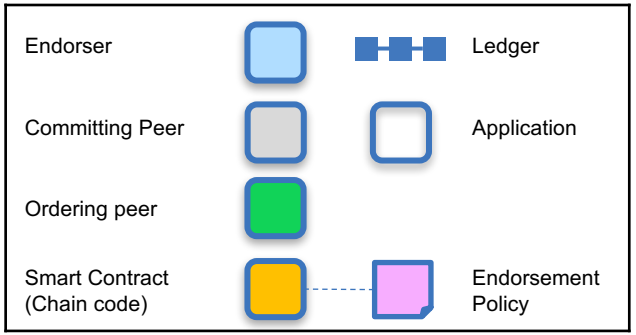
Operator installs then instantiates

Operator **installs** smart contracts with endorsement policies to appropriate peers: **E_0**, **E_1**, **E_2**, **P_3**, and not **P_4**

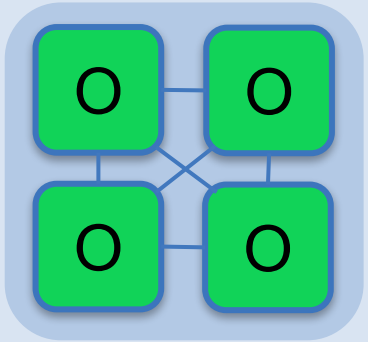
Operator **instantiates** smart contract on given channel. One-time initialization

Policy subsequently available to all peers on channel, e.g. including **P_4**

Key:



Bootstrapping the Network (1/6) – Configure & start Ordering Service

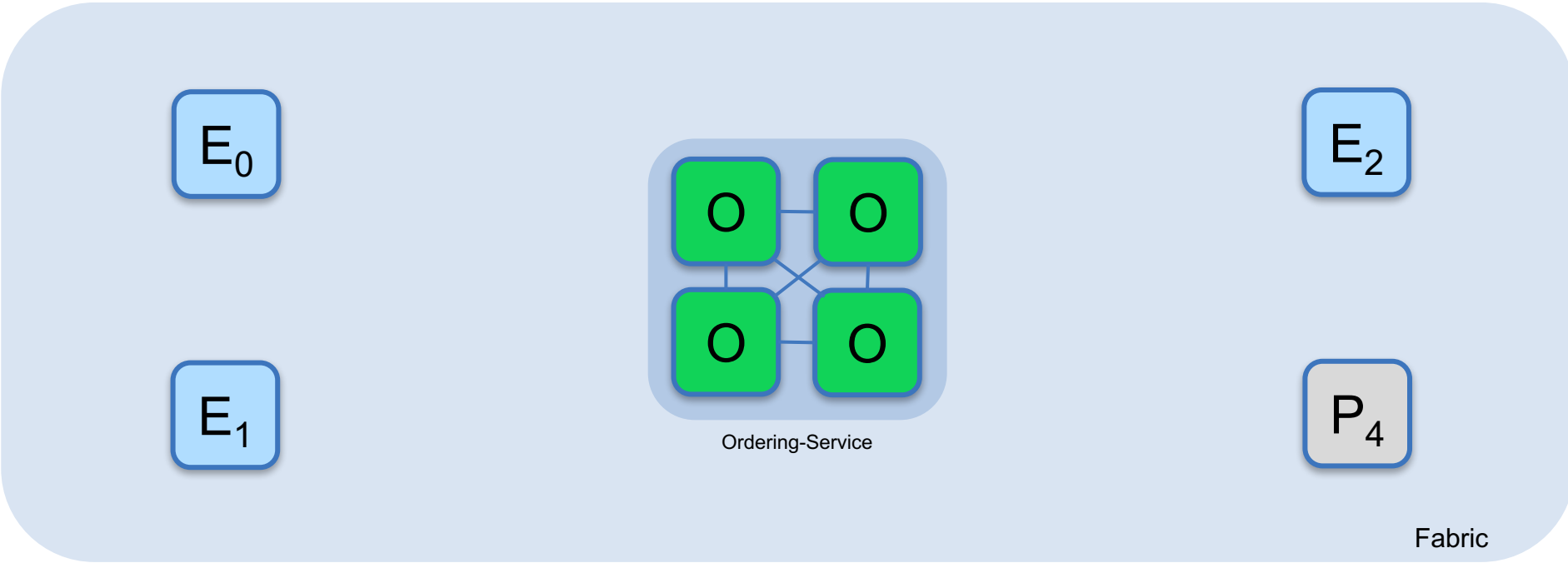


Ordering-Service

Fabric

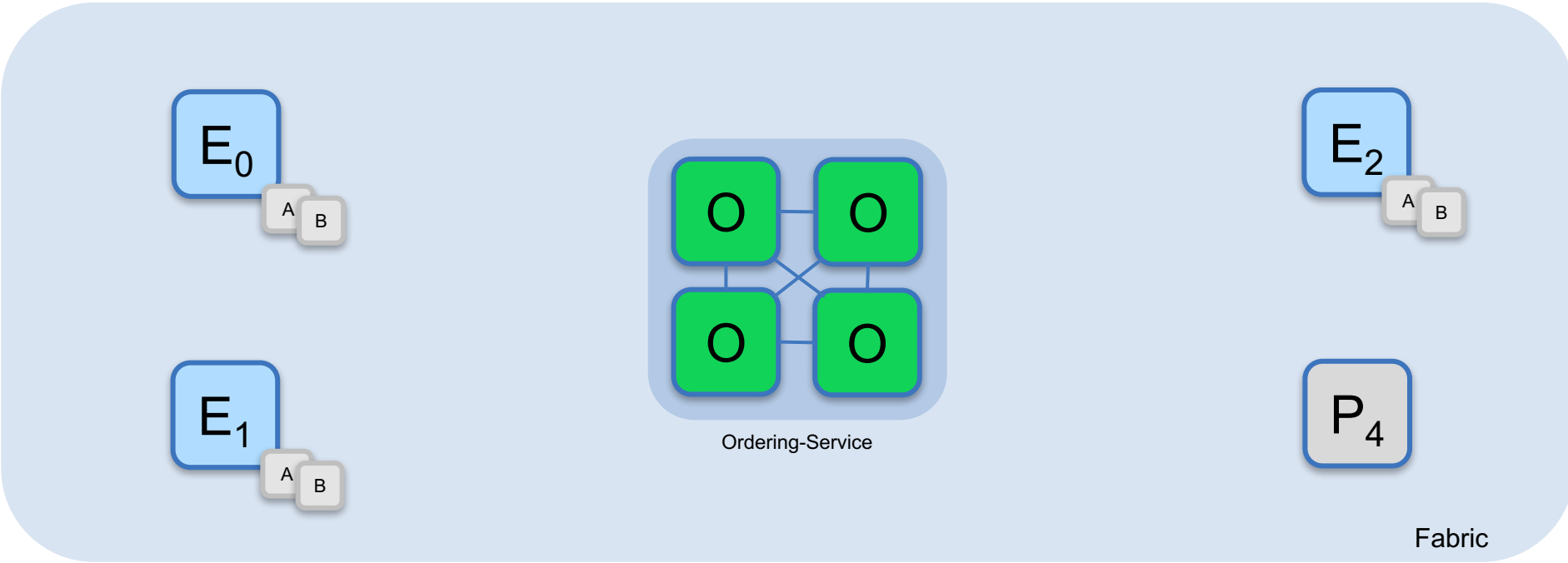
- An Ordering Service is **configured** and started for other network peers to use
  `$ docker-compose [-f orderer.yml] ...`

Bootstrapping the 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 ...`

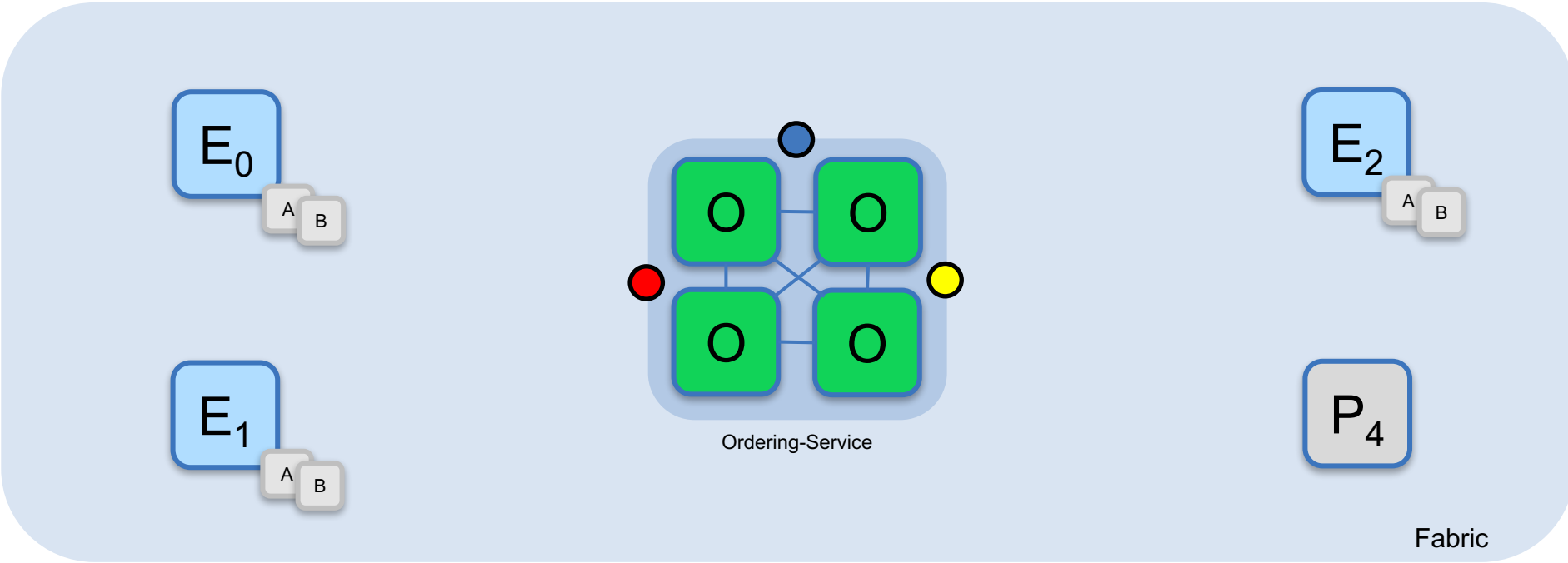
Bootstrapping the Network (3/4) – Install Chaincode



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

```
$ peer chaincode install ...
```

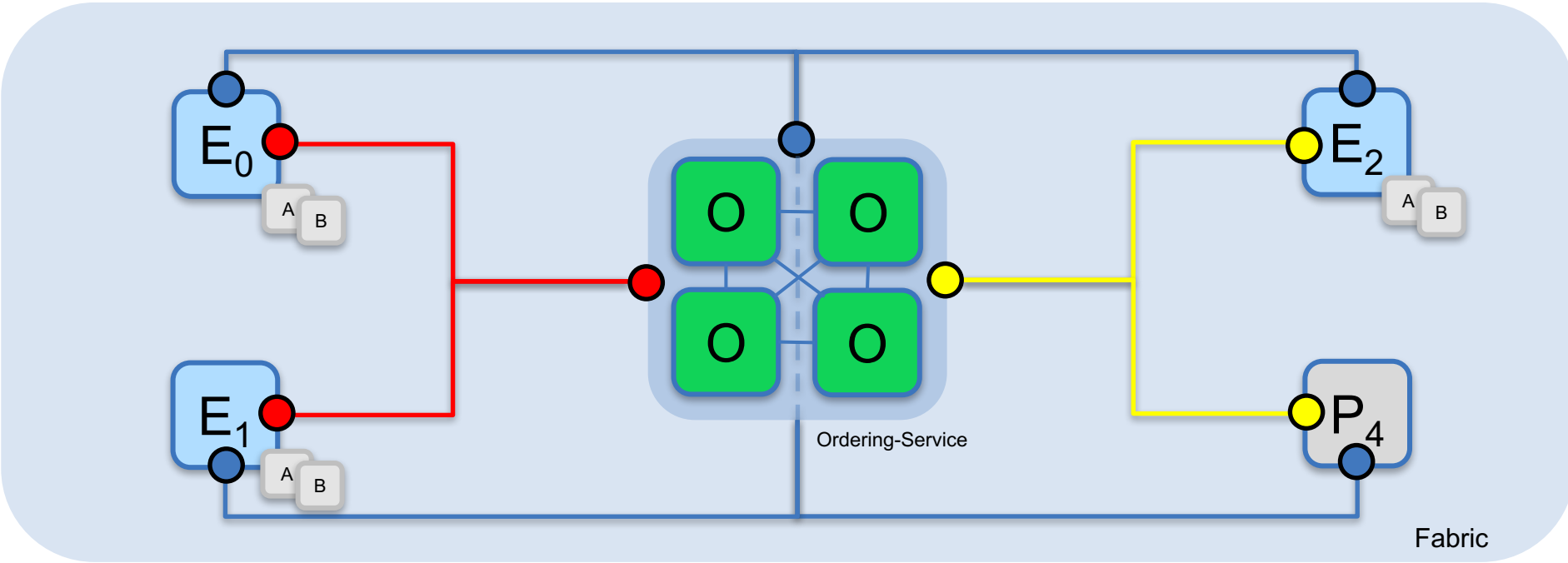
Bootstrapping the Network (4/6) – Create Channels



- Channels are **created** on the ordering service

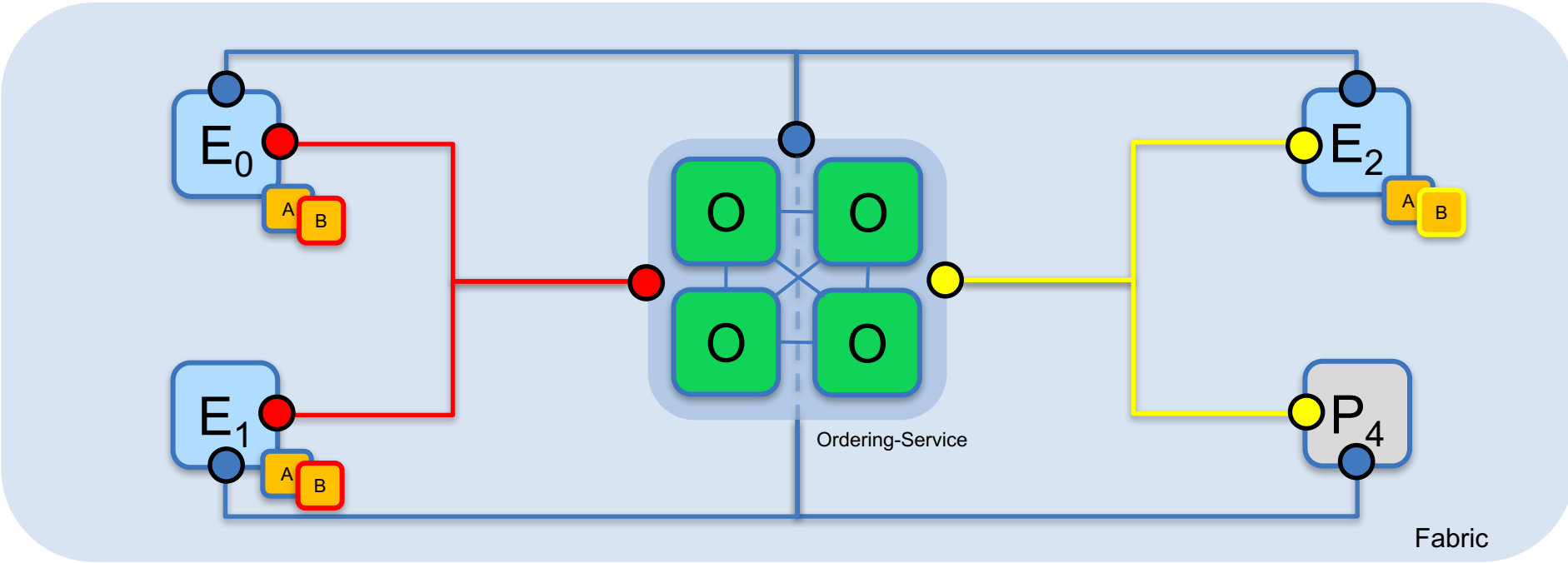
```
$ peer channel create -o [orderer] ...
```

Bootstrapping the Network (5/6) – Join Channels



- Peers that are permissioned can then **join** the channels they want to transact on
  `$ peer channel join ...`

Bootstrapping the Network (6/6) – Instantiate Chaincode



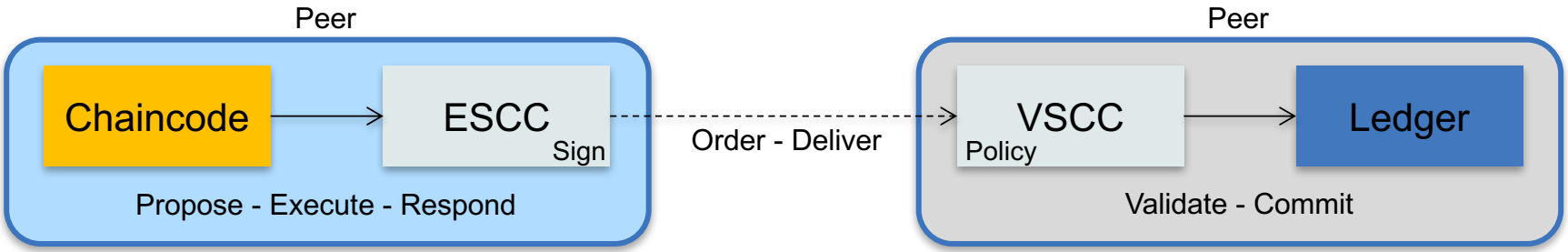
- Peers finally **instantiate** the Chaincode on the channels they want to transact on
  `$ peer channel instantiate ... -P 'policy'`
- Once instantiated a Chaincode is live and can process transaction requests
- Endorsement Policy is specified at instantiation time

Endorsement Policies

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 associated with an Endorsement Policy
- Default implementation: Simple declarative language for the 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')“
```

This command instantiates 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 required, and **ROLE** is either “member” or “admin”.

Endorsement Policy Examples

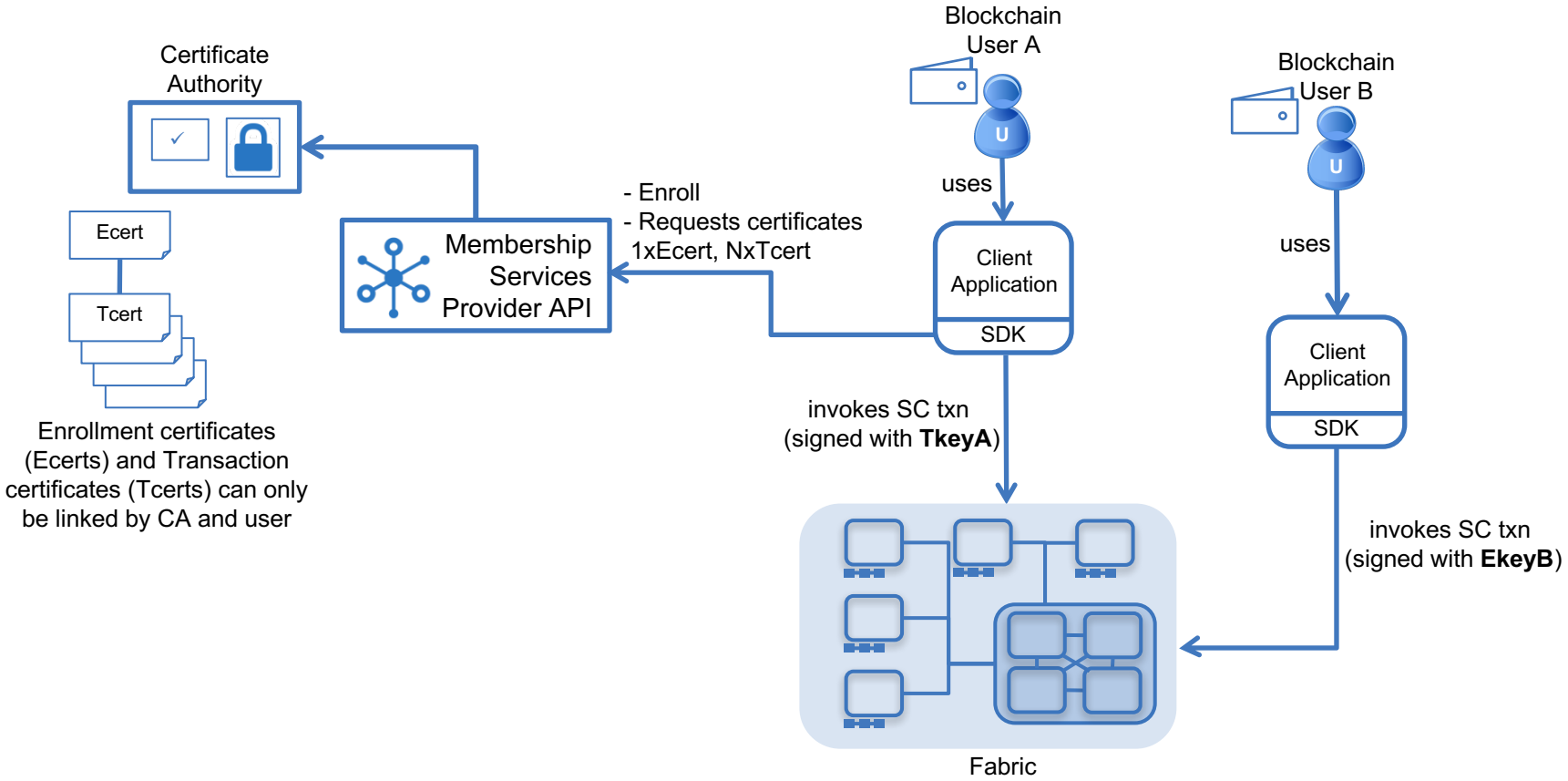
Examples of policies:

- 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'))

Permissioned Ledger Access

Transaction and identity privacy

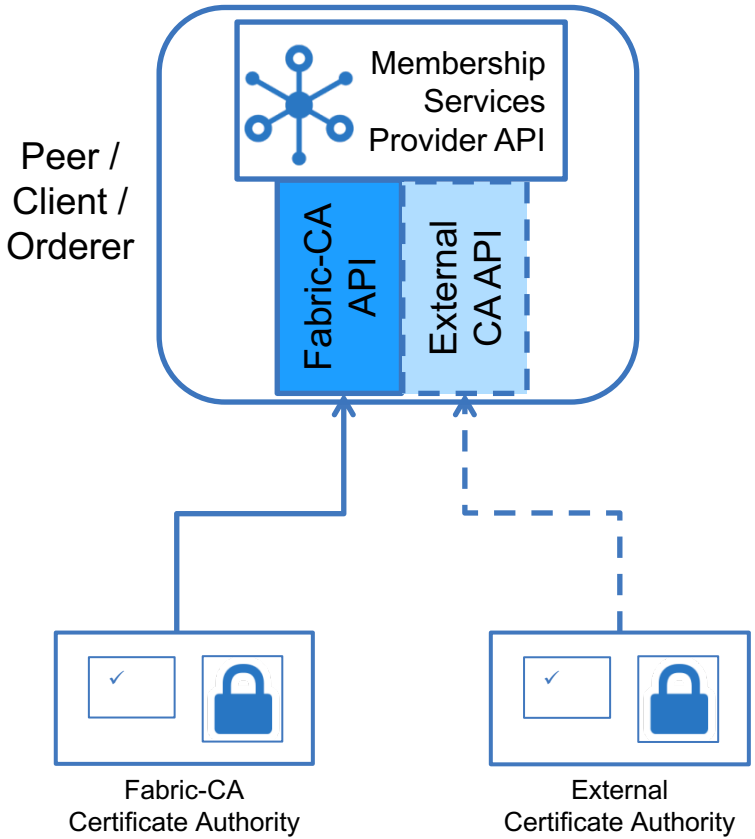
Membership Services Overview



Transaction and Identity Privacy

- Enrollment Certificates, Ecerts
  - Long term identity
  - Can be obtained offline, bring-your-own-identity
- Transaction Certificates, Tcerts
  - Disposable certificates, typically used once, requested from Transaction CA
  - Tcert derived from long term identity - Enrollment Certificate, Ecert
  - Only Transaction CA can link Ecert and Tcert
- Permissioned Interactions
  - Users sign with either Ecerts or Tcerts
- Membership Services
  - Abstract layer to credential providers

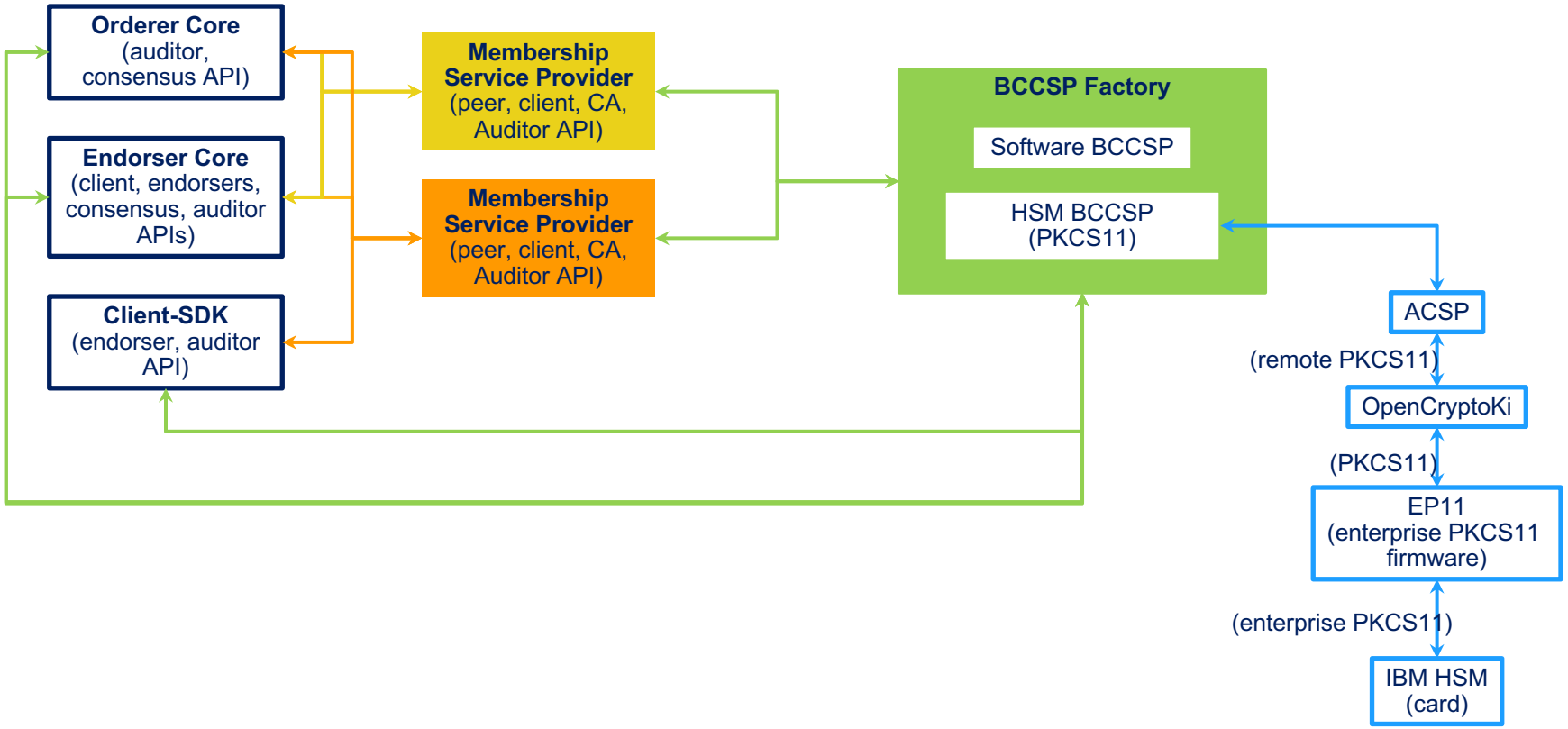
Membership Services Provider API



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).

MSP and BCCSP (Modularity and Decentralisation)



Membership Services Provider (MSP)

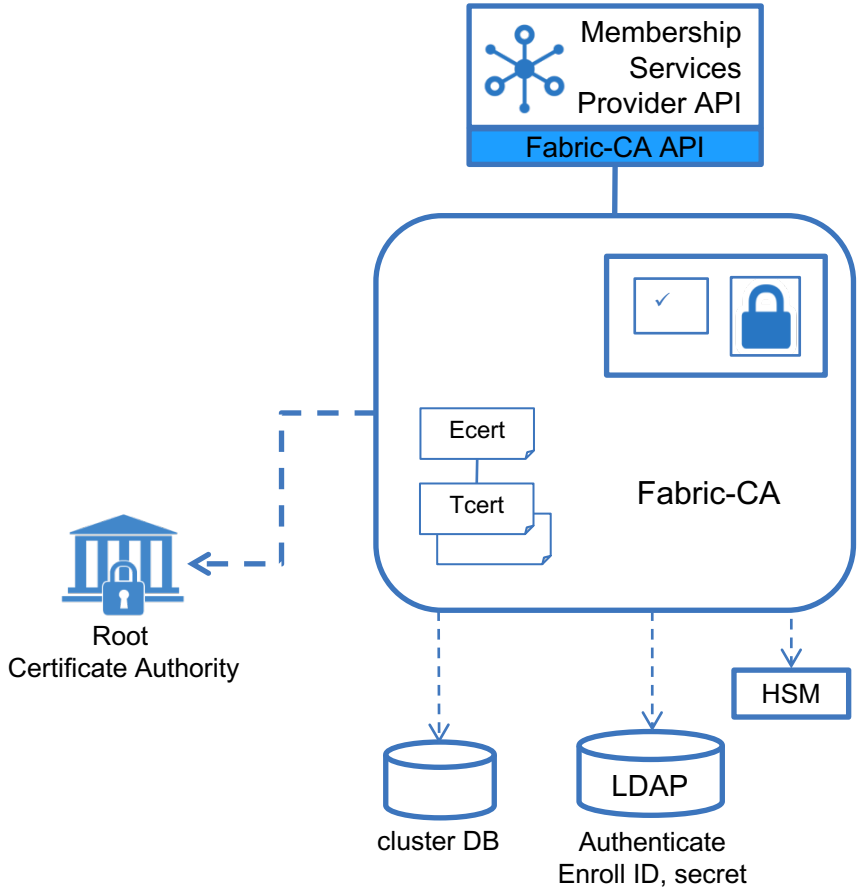
- An abstraction to represent a membership authority and its operations on issuing and management of fabric membership credentials in a modular & pluggable way
  - Allows for the co-existence of a variety of credential management architectures
  - Allows for easy organizational separation in credential 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

Blockchain Crypto Service Provider (BCCSP)

- Pluggable implementation of cryptographic standards and algorithms.
- Pluggability
  - alternate implementations of crypto interface can be used within the HPL/fabric code, **without modifying** the core
- Support for Multiple CSPs
  - Easy addition of more types of CSPs, e.g., of different HSM types
  - Enable the use of different CSP on different system components transparently
- International Standards Support
  - E.g., via a new/separate CSP
  - Interoperability among standards is not necessarily guaranteed

Fabric-CA Details



Fabric-CA

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

Fabric-CA

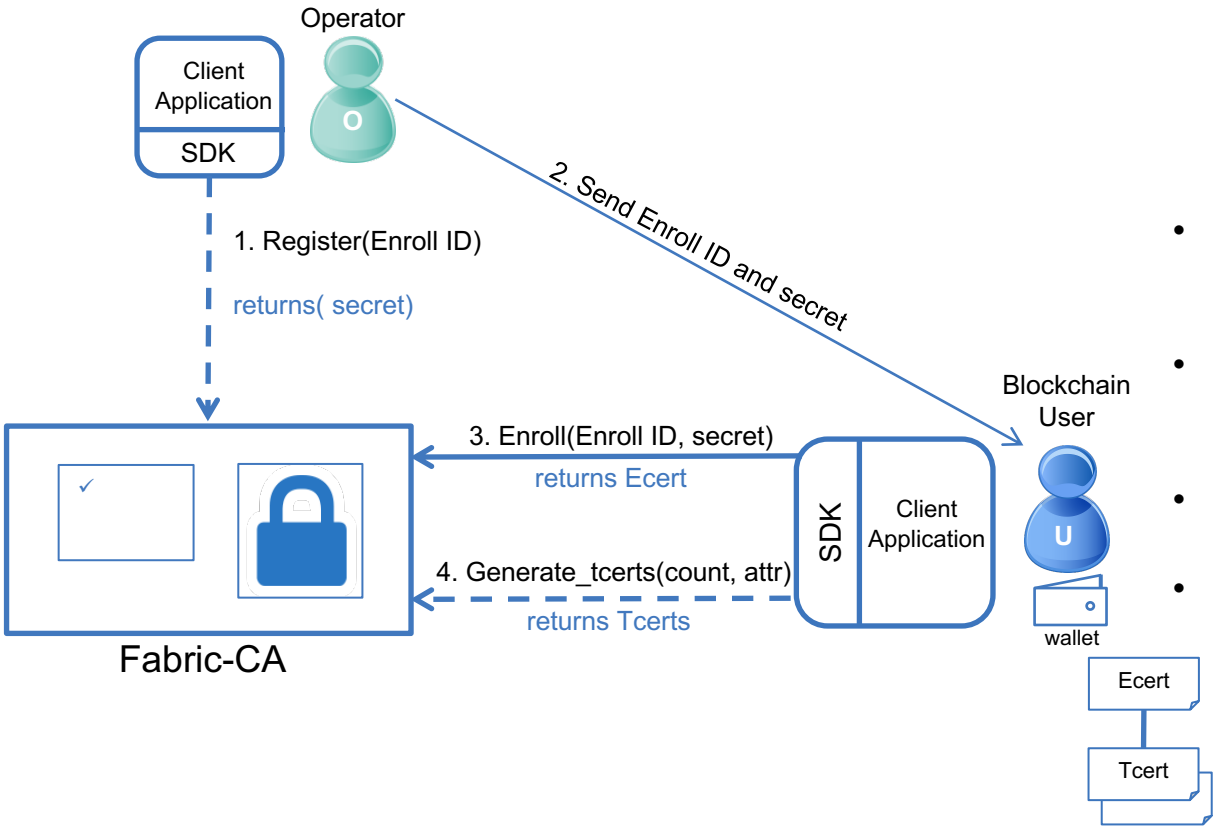
Certificate Authority

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

Fabric leverages it

- On the peer side for client certificate authentication
- For peer authentication
- For client-authentication, when anonymity is not required

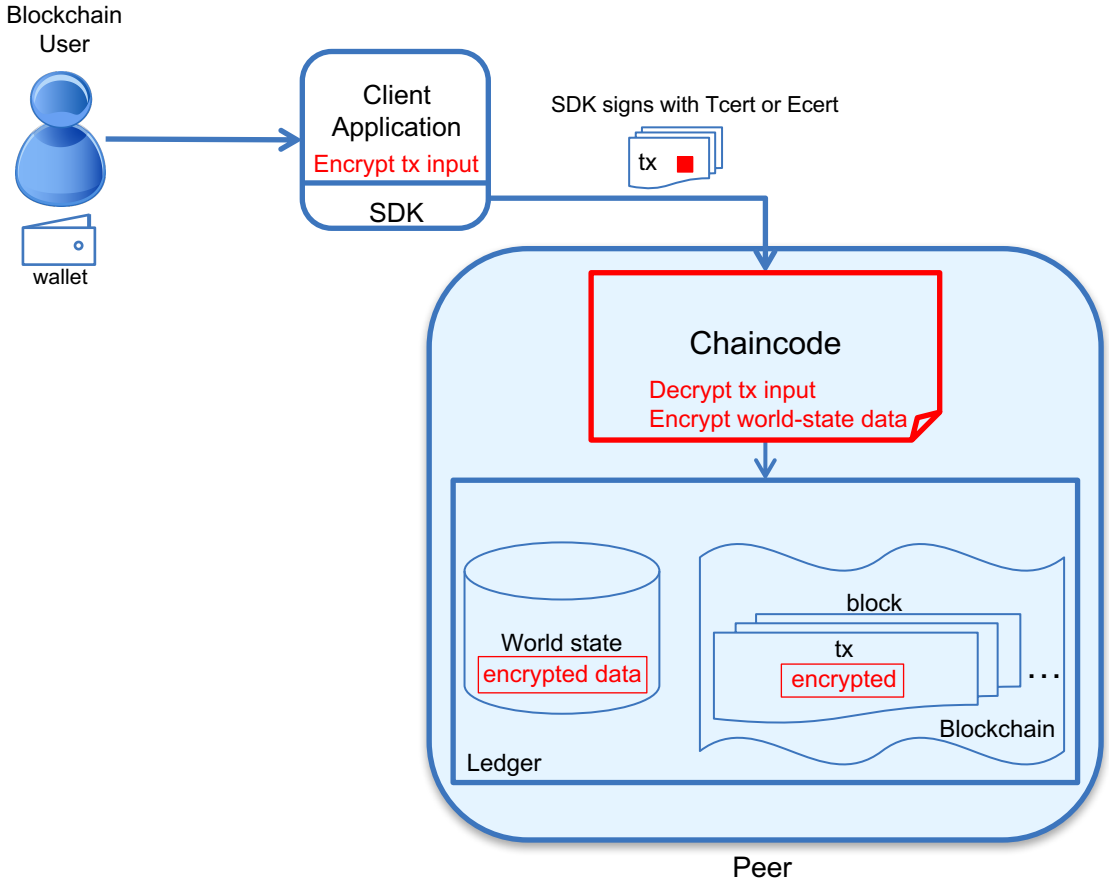
New User Registration and Enrollment



Registration and Enrollment

- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- User requests Tcerts in batches
- 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).

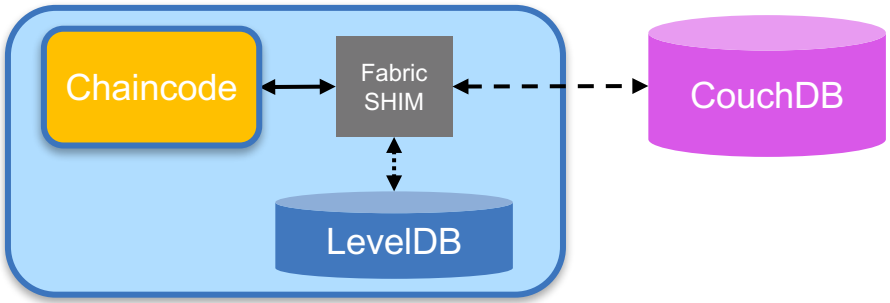
*Encryption of application chaincode requires additional development of system chaincode.

Pluggable world state

How is data managed on the ledger?

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



Summary and Next Steps

Summary and Next Steps

- Apply shared ledgers and smart contracts to your Business Network
- Think about your participants, assets and business processes
- Spend time thinking about realistic business use cases
- Get some hands-on experience with the technology
- Start with a First Project
- IBM can help with your journey

Thank you!

www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org