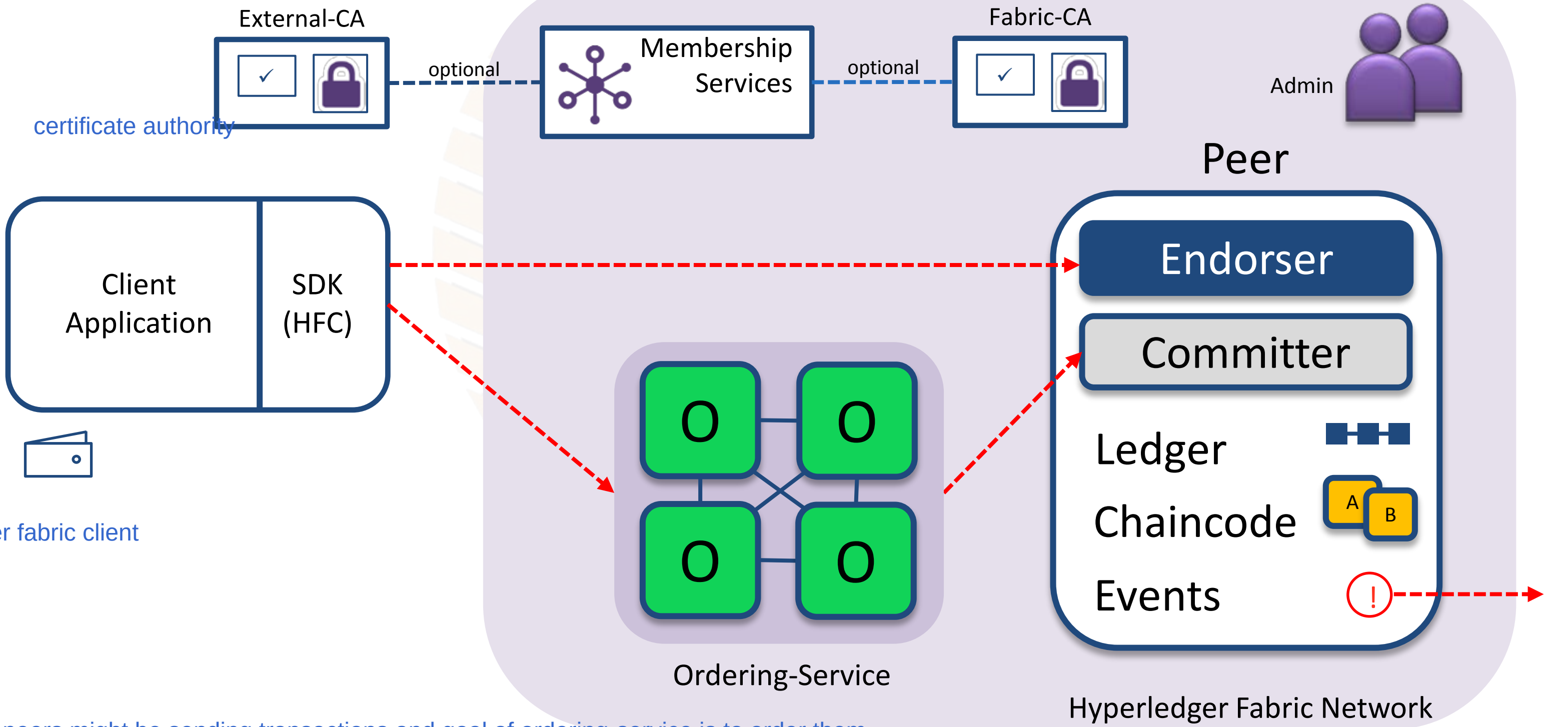
Hyperledger Fabric V1 Architecture

User can login with his signature and can also register to get signature and we have both public key and private key



certificate authority

Hyperledger fabric client

different peers might be sending transactions and goal of ordering-service is to order them

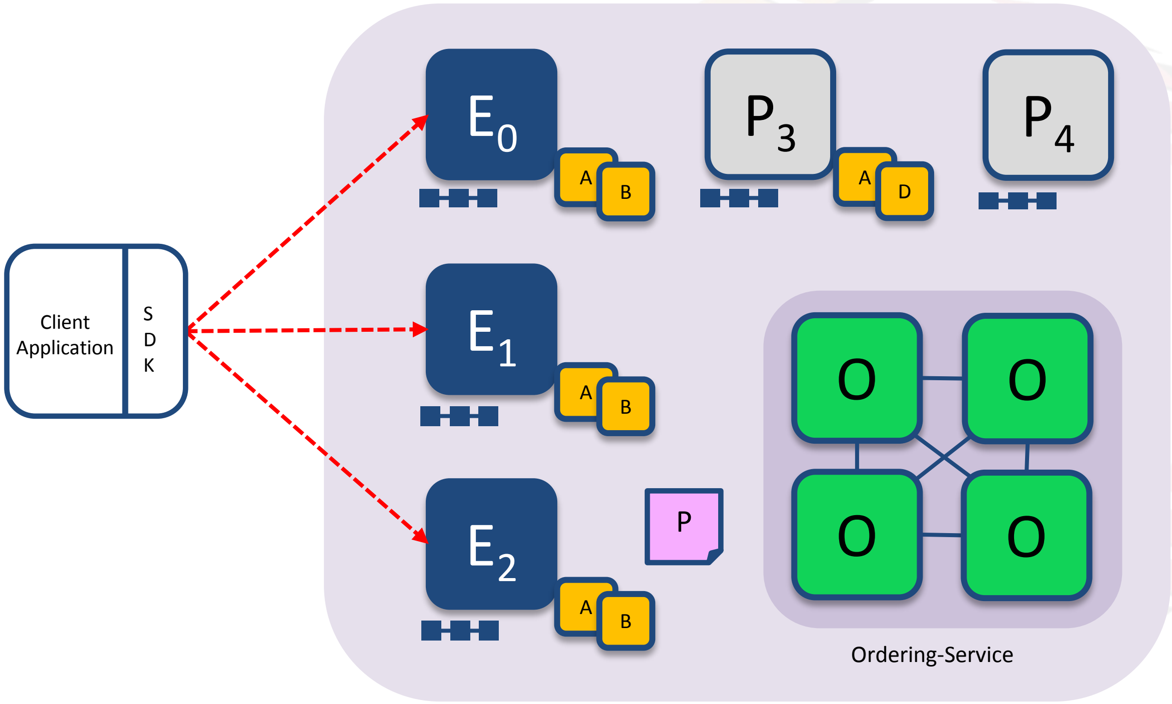
Nodes and Roles

	Committing Peer: Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).
	it executes the transactions **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.

Transaction Flow

Consensus is achieved using the following transaction flow:

Step 1/7: Propose Transaction



Hyperledger Fabric Network

Application proposes transaction

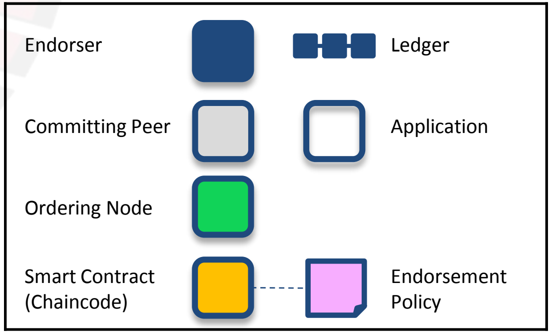
endorsement policy is predefined by smart contract

Endorsement policy:

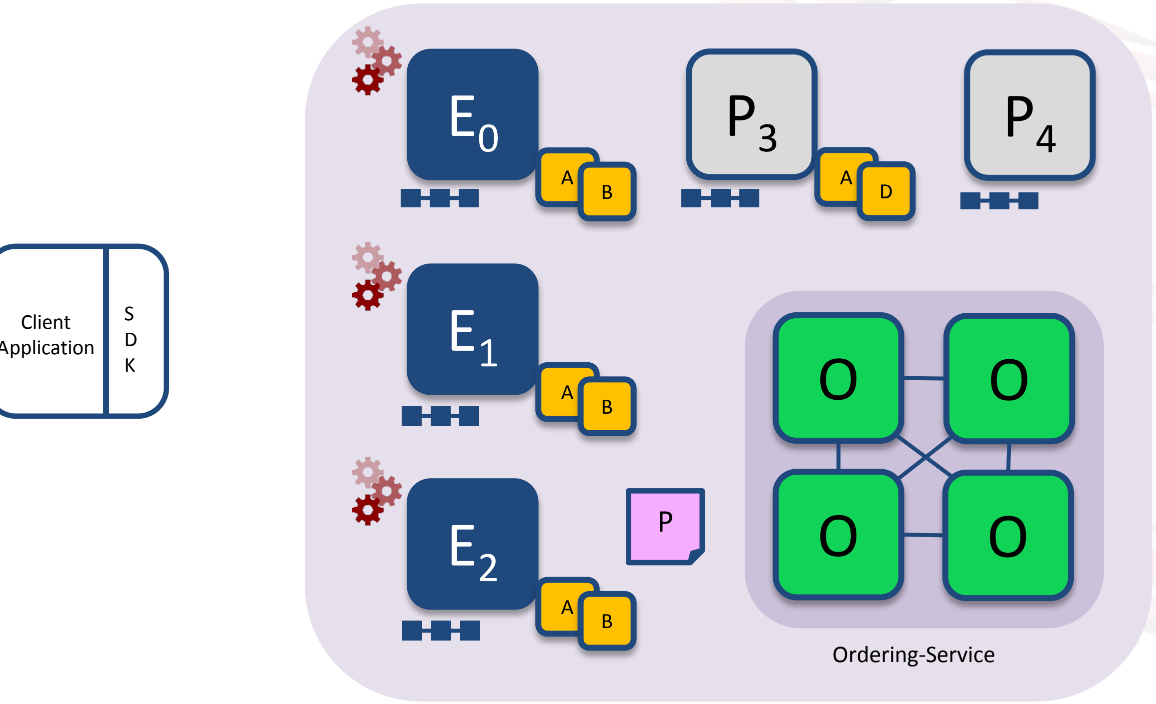
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

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

Key:



Step 2/7: Execute Proposed Transaction



Hyperledger Fabric Network

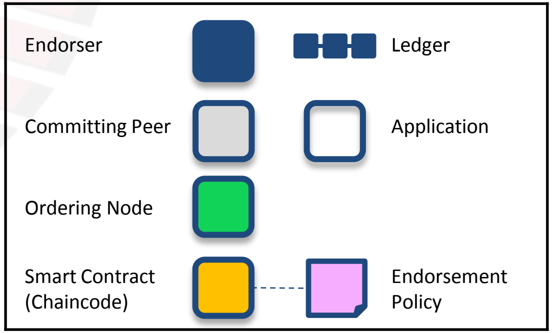
Endorsers Execute Proposals

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

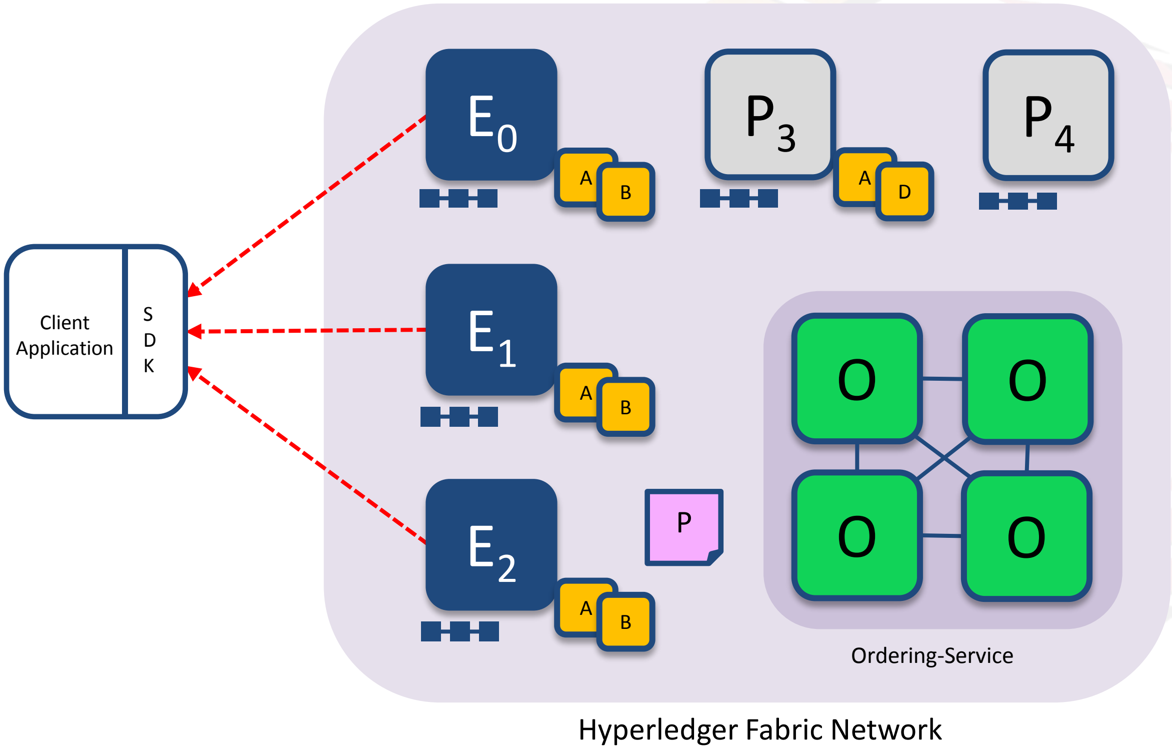
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



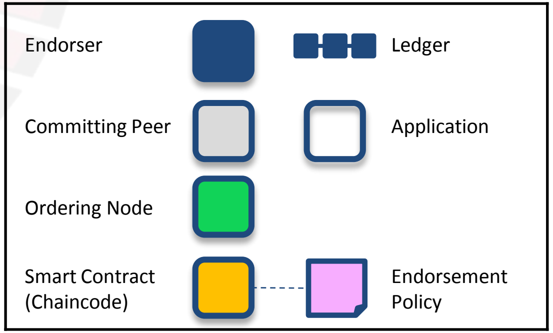
Step 3/7: Proposal Response



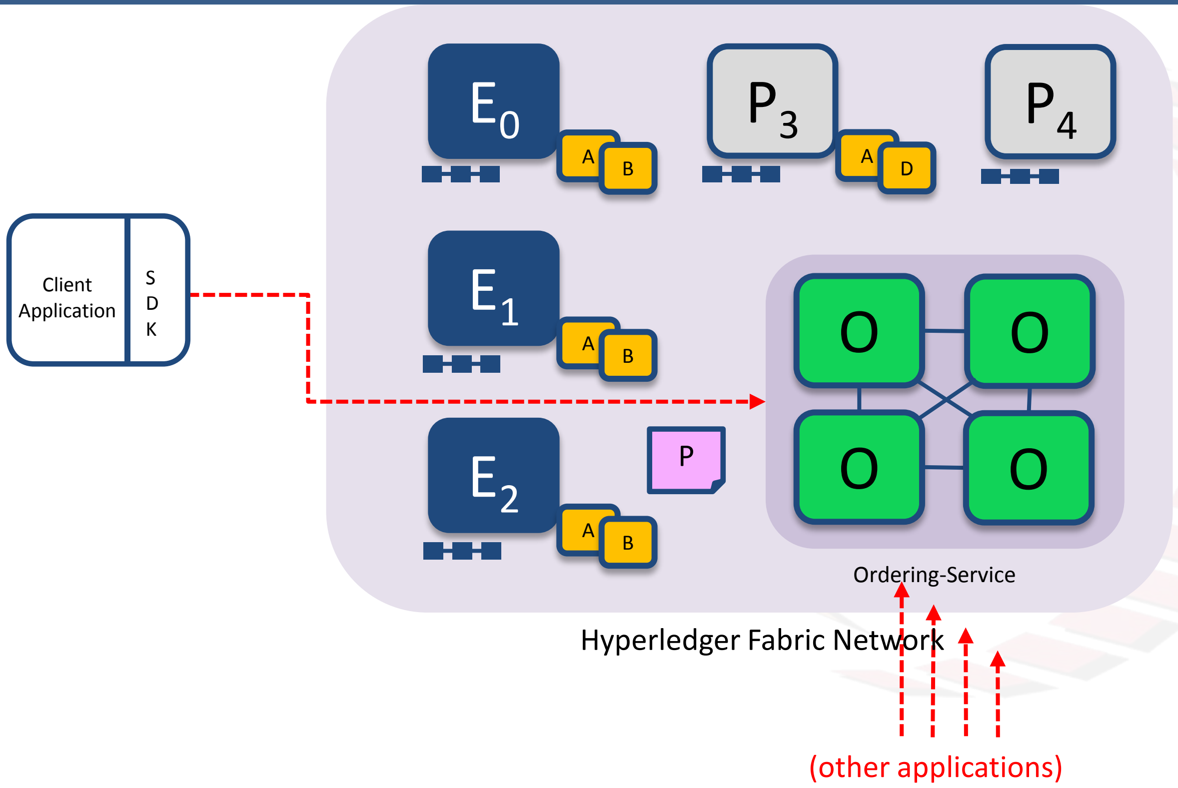
Application receives responses

Read-Write 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:



Step 4/7: Order Transaction

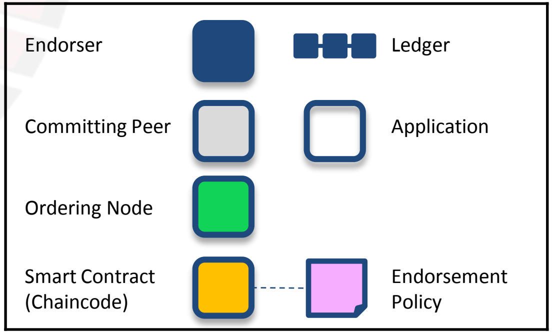


Responses submitted for ordering

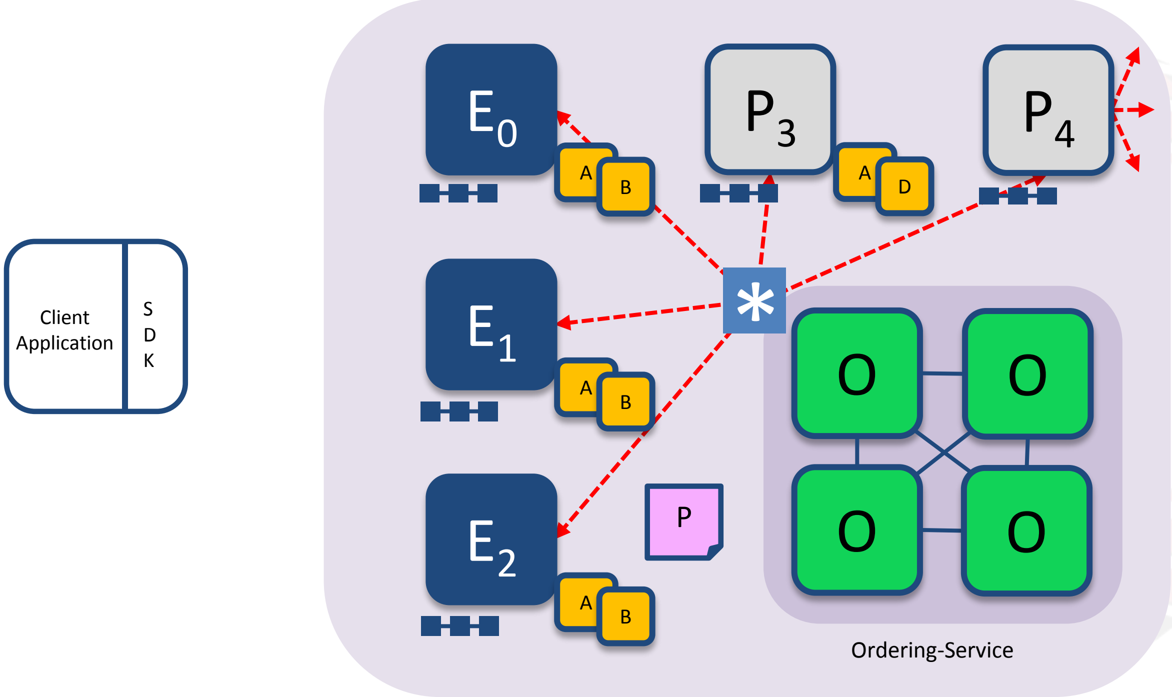
Application submits responses as a transaction to be ordered.

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

Key:



Step 5/7: Deliver Transaction



Hyperledger Fabric Network

Orderer delivers to committing peers

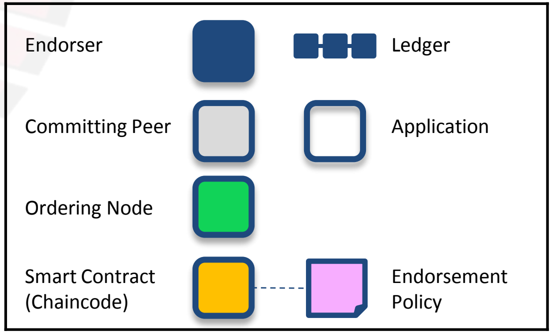
Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown)

Different ordering algorithms available:

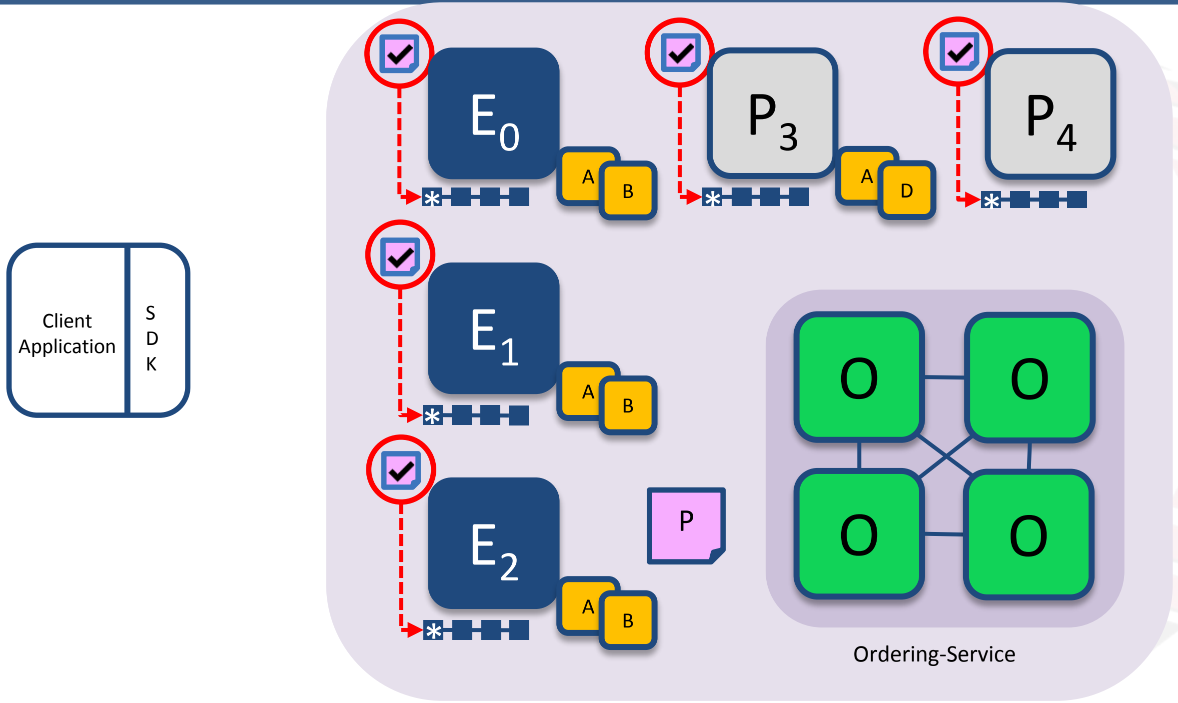
- SOLO (Single node, development)
- Kafka (Crash fault tolerance)

Kafka requires minimum 3 nodes

Key:



Step 6/7: Validate Transaction

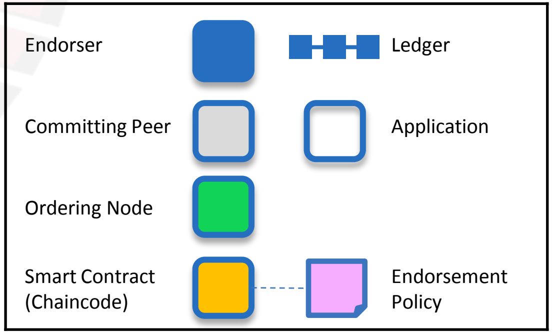


Hyperledger Fabric Network

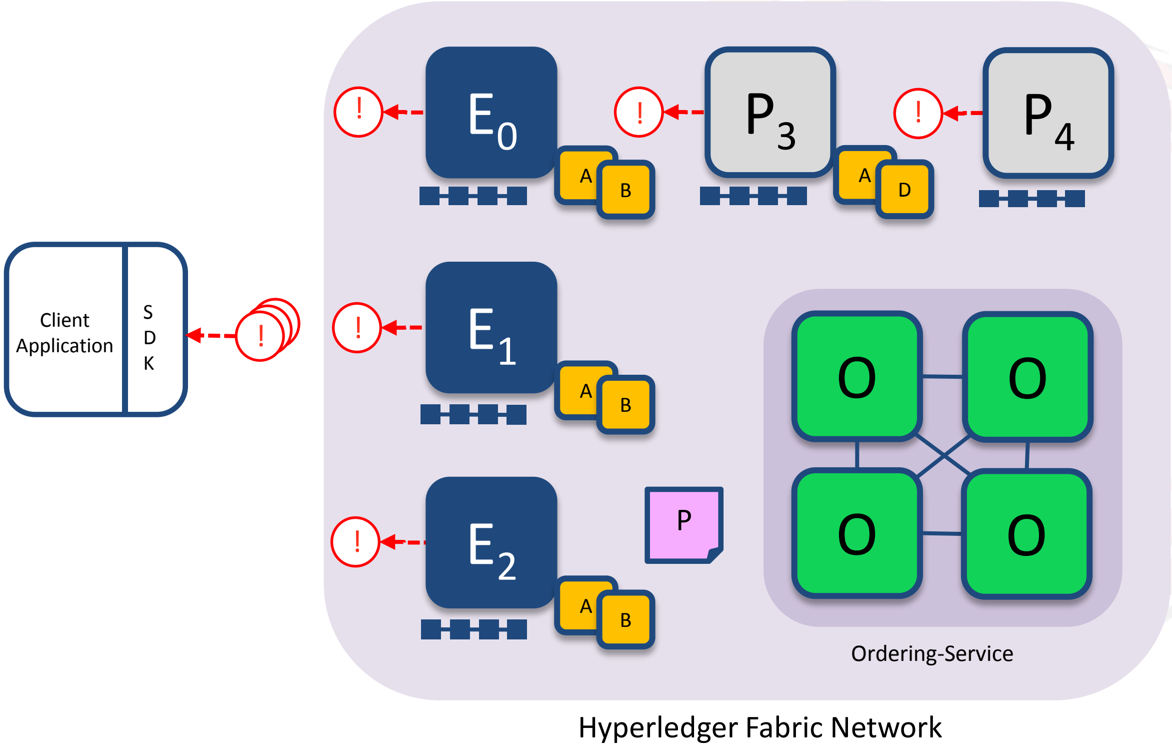
Committing peers validate transactions

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:



Step 7/7: Notify Transaction

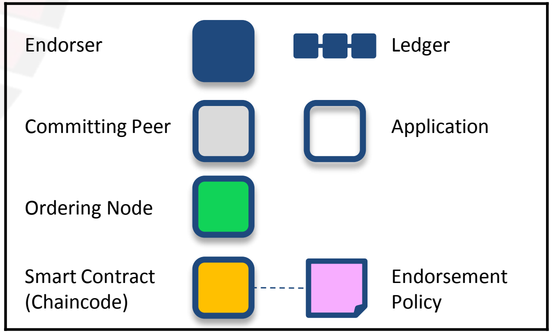


Committing peers notify applications

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:



Key Benefits of the Transaction Flow

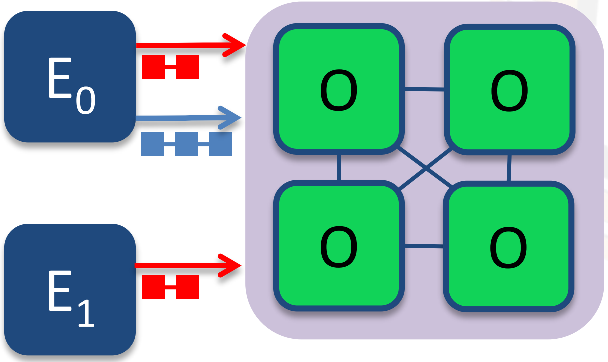
- Better reflect business processes by specifying who endorses transactions

- Eliminate non deterministic transactions

- Scale the number of participants and transaction throughput

Channels

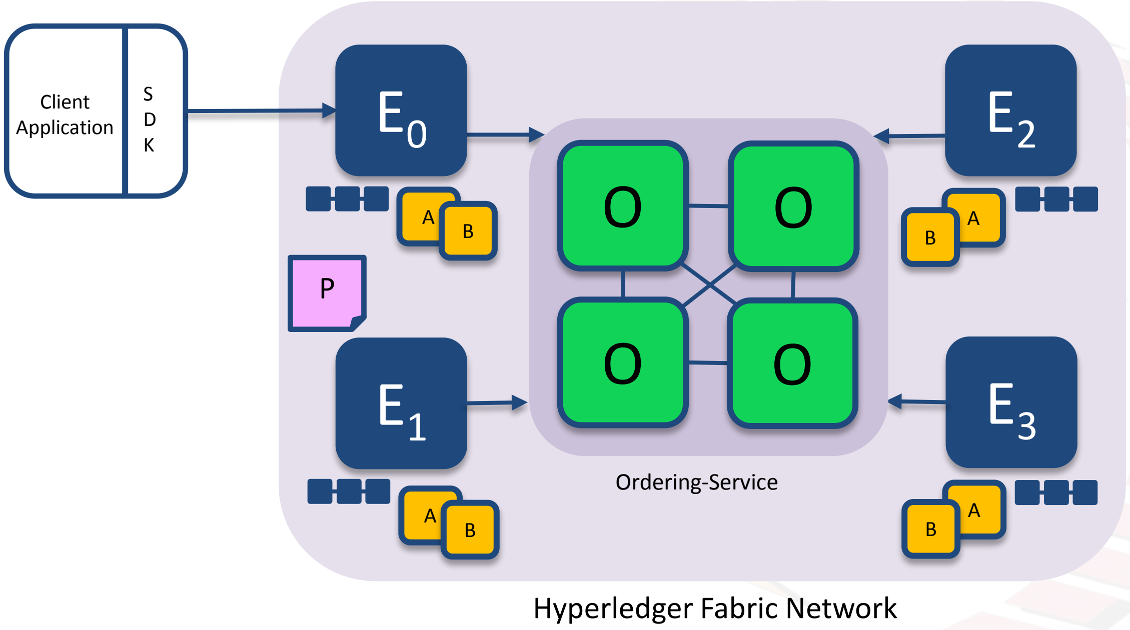
Channels provide privacy between different ledgers



Ordering-Service

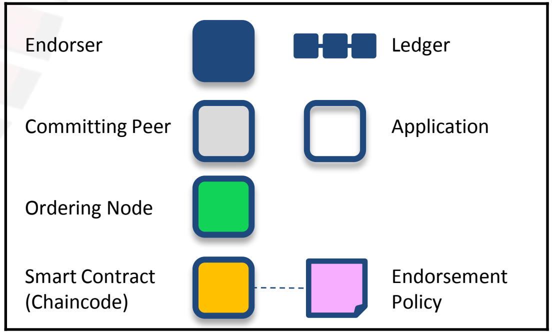
- Ledgers exist in the scope of a channel
  - Channels can be shared across an entire network of peers
  - Channels can be permissioned for a specific set of participants
- Chaincode is installed on peers to access the worldstate
- Chaincode is instantiated on specific channel
- Peers can participate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

Single Channel Network

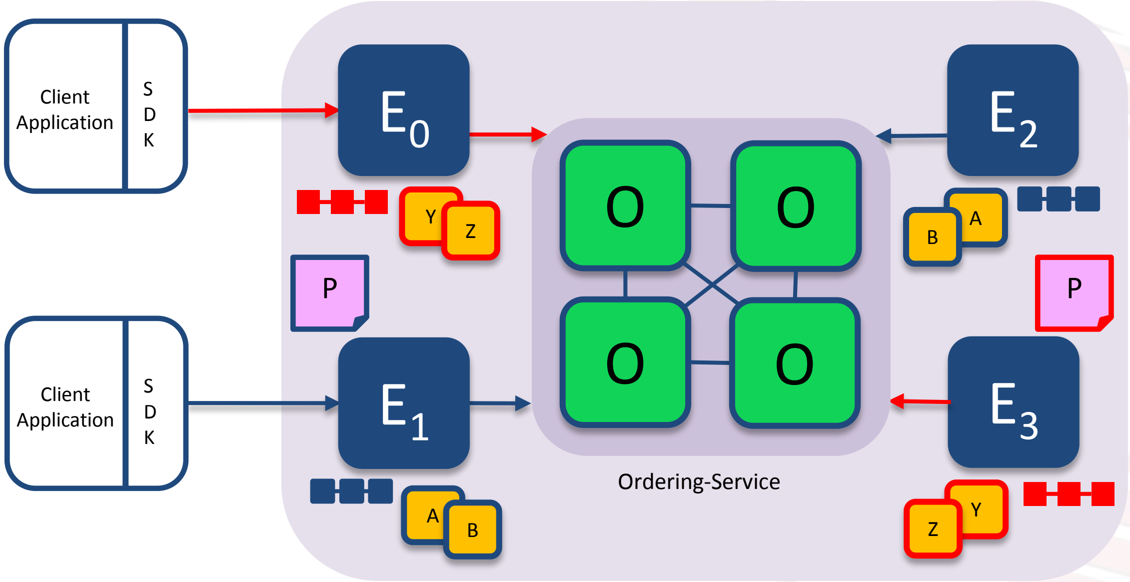


- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3

Key:



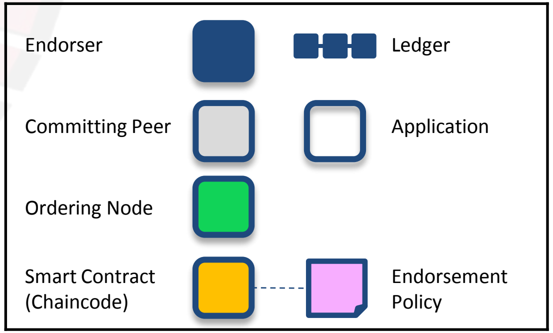
Multi-Channel Network



Hyperledger Fabric Network

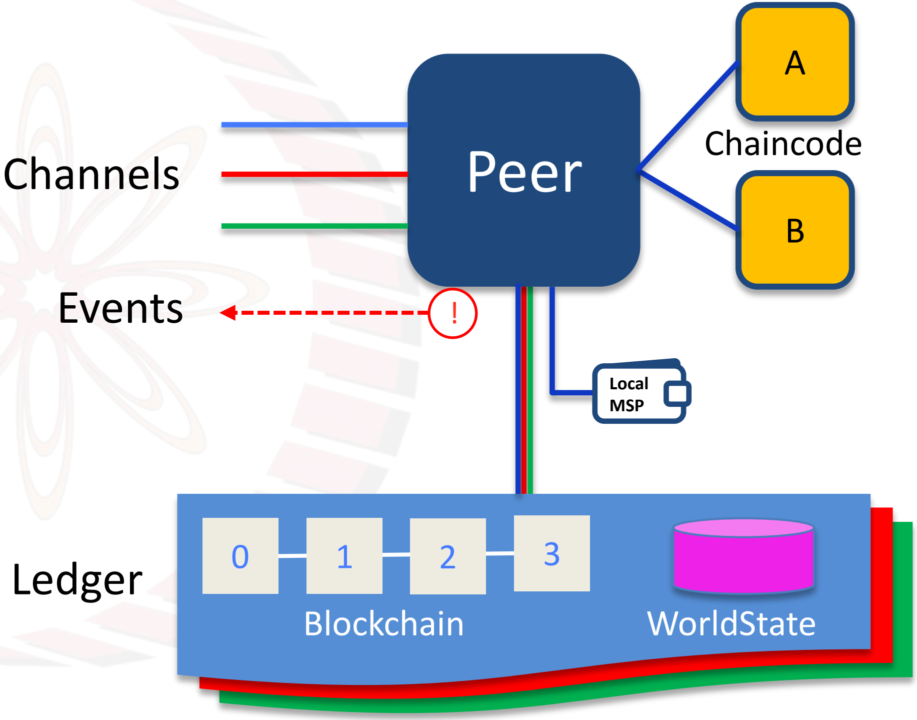
- Peers E_0 and E_3 connect to the red channel for chaincodes Y and Z
- Peers E_1 and E_2 connect to the blue channel for chaincodes A and B

Key:



Fabric Peer

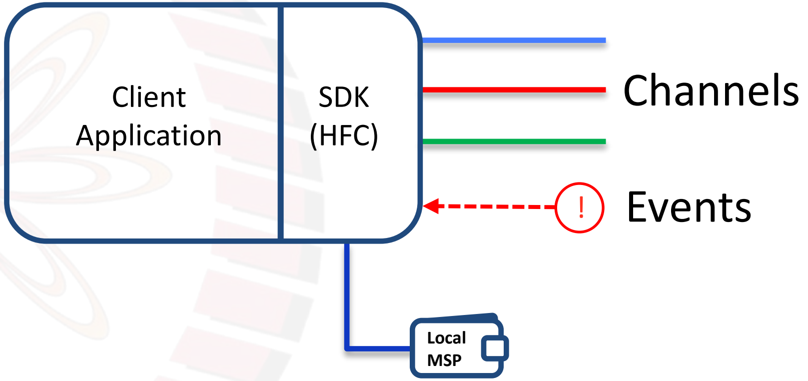
- Each peer:
  - Connects to one or more channels
  - Maintains one or more ledgers for each channel
  - Chaincodes are instantiated in separate docker containers
  - Chaincodes are shared across channels (no state is stored in chaincode container)
  - Local MSP (Membership Services Provider) provides crypto material
  - Emits events to the client application



Client Application

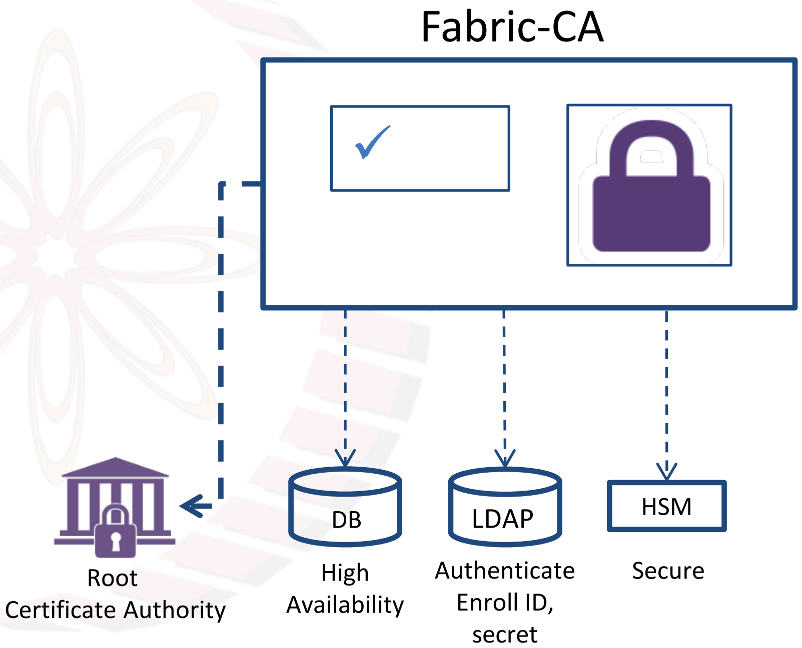
- Each client application uses Fabric SDK to:
  - Connects over channels to one or more peers
  - Connects over channels to one or more orderer nodes
  - Receives events from peers
  - Local MSP provides client crypto material

  - Client can be written in different languages (Node.js, Go, Java, Python?)



Fabric Certificate Authority

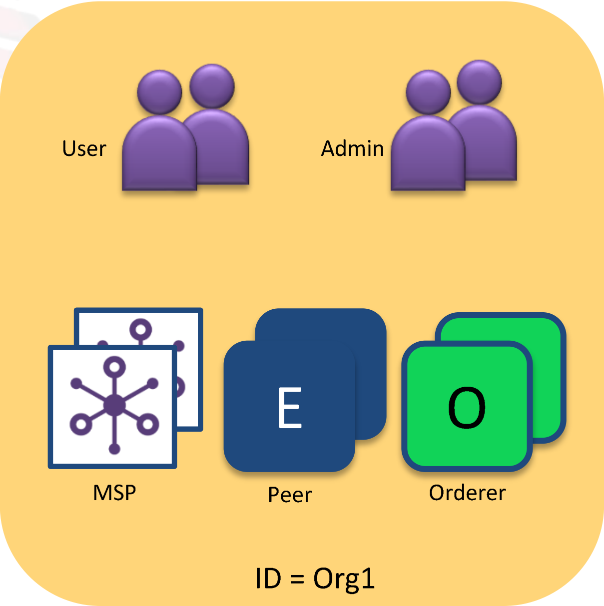
- Default (optional) Certificate Authority within Fabric network for issuing Ecerts (long-term identity)
- Supports clustering for HA characteristics
- Supports LDAP for user authentication
- Supports HSM for security
- Can be configured as an intermediate CA



Organisations

Organisations define boundaries within a Fabric Blockchain Network

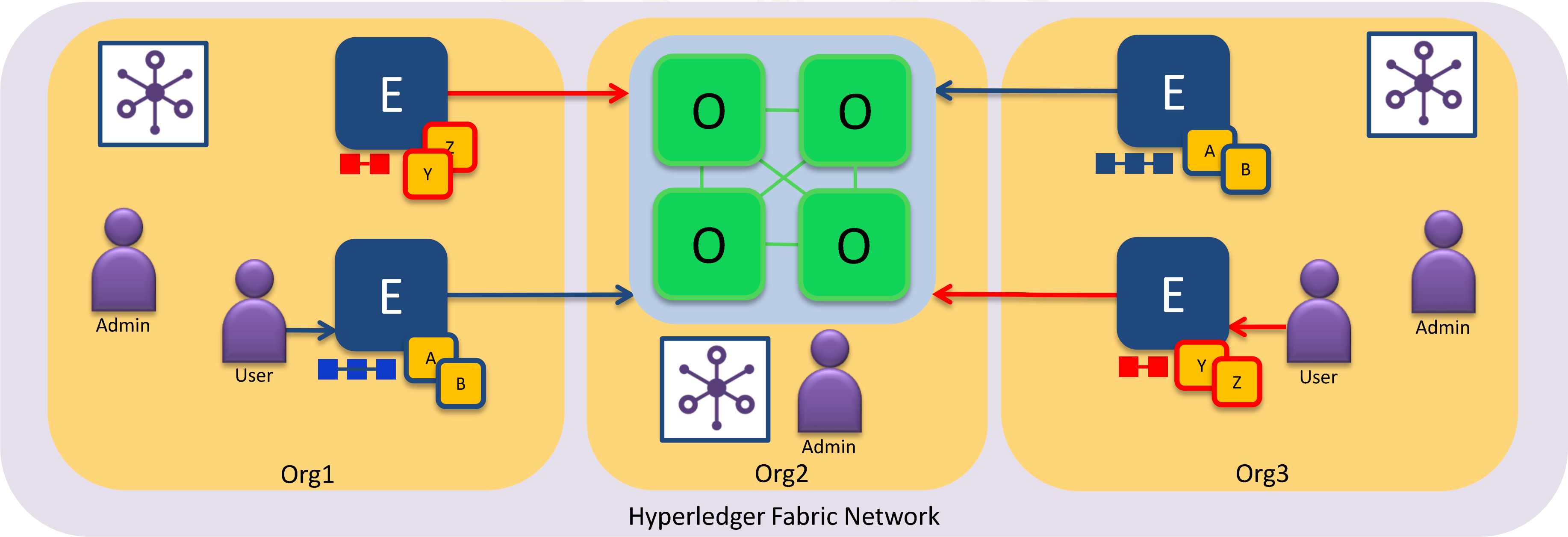
- Each organisation defines:
  - Membership Services Provider (MSP) for identities
  - Administrator(s)
  - Users
  - Peers
  - Orderers (optional)
- A network can include many organisations representing a consortium
- Each organisation has an ID



Consortium Network

An example consortium network of 3 organisations

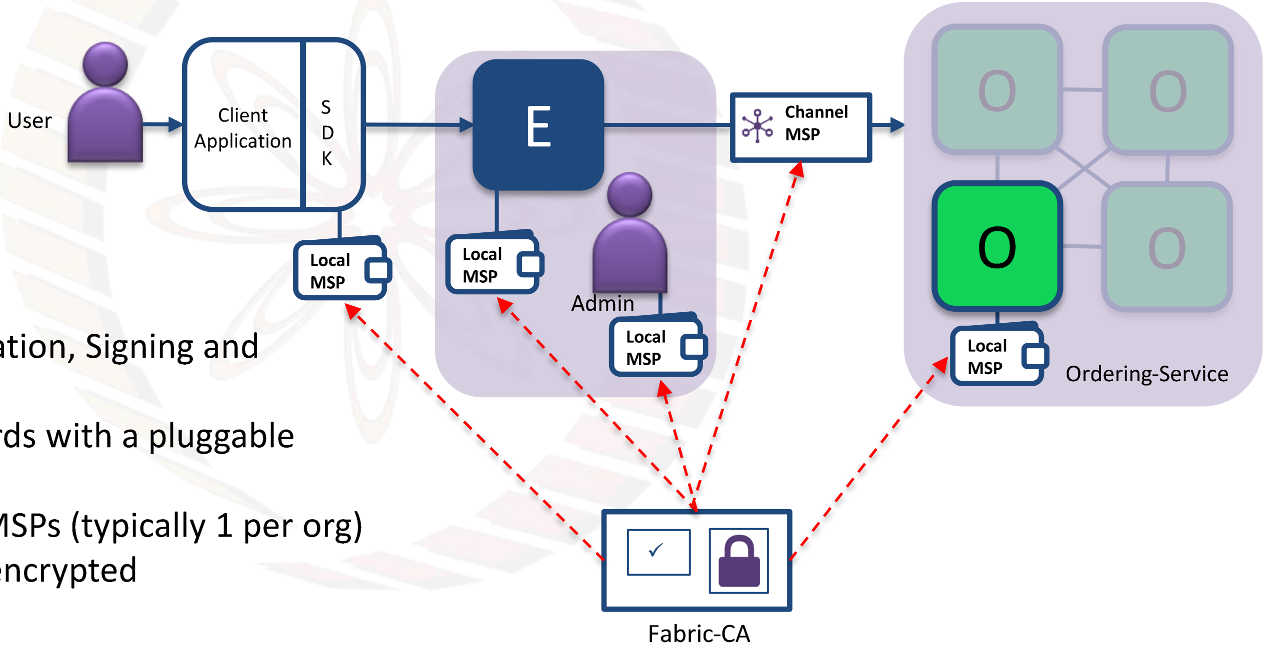
- Orgs 1 and 3 run peers
- Org 2 provides the ordering service only



Membership Service Provider (MSP) - Overview

A MSP manages a set of identities within a distributed Fabric network

- Provides identity for:
  - Peers and Orderers
  - Client Applications
  - Administrators
- Identities can be issued by:
  - Fabric-CA
  - An external CA
- Provides: Authentication, Validation, Signing and Issuance
- Supports different crypto standards with a pluggable interface
- A network can include multiple MSPs (typically 1 per org)
- Includes TLS crypto material for encrypted communications



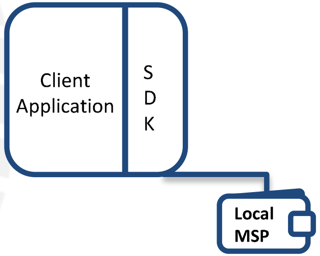
Transport Layer Security (TLS)

- Cryptographic protocols that provide communications security over a computer network
- Provides privacy and data integrity
- Symmetric cryptography is used to encrypt the data transmitted (privacy)
- Public-key cryptography is used to authenticate the identities of the communicating parties
- Include message integrity check to prevent loss or alteration of the data
- All component communication in Fabric secured using TLS (client-peer, peer-peer, peer-orderer, orderer-orderer)

User Identities

Each client application has a local MSP to store user identities

- Each local MSP includes:
  - **Keystore**
    - **Private key** for signing transactions
  - **Signcert**
    - **Public x.509 certificate**



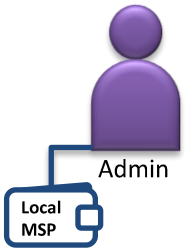
user@org1.example.com	
keystore	<private key>
signcert	user@org1.example.com-cert.pem

- May also include TLS credentials
- Can be backed by a Hardware Security Module (HSM)

Admin Identities

Each Administrator has a local MSP to store their identity

- Each local MSP includes:
  - **Keystore**
    - **Private key** for signing transactions
  - **Signcert**
    - **Public x.509 certificate**



- May also include Transport Layer Security (TLS) credentials
- Can be backed by a Hardware Security Module (HSM)

admin@org1.example.com	
keystore	<private key>
signcert	admin@org1.example.com-cert.pem

Peer and Orderer Identities

Each peer and orderer has a local MSP

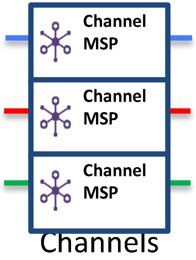
- Each local MSP includes:
  - **keystore**
    - **Private key** for signing transactions
  - **signcert**
    - **Public x.509 certificate**
- In addition Peer/Orderer MSPs identify authorized administrators:
  - **admincerts**
    - List of **administrator certificates**
  - **cacerts**
    - The **CA public cert** for verification
  - **crls**
    - List of **revoked certificates**
- Peers and Orderers also receive channel MSP info
- Can be backed by a Hardware Security Module (HSM)

peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>

Channel MSP Information

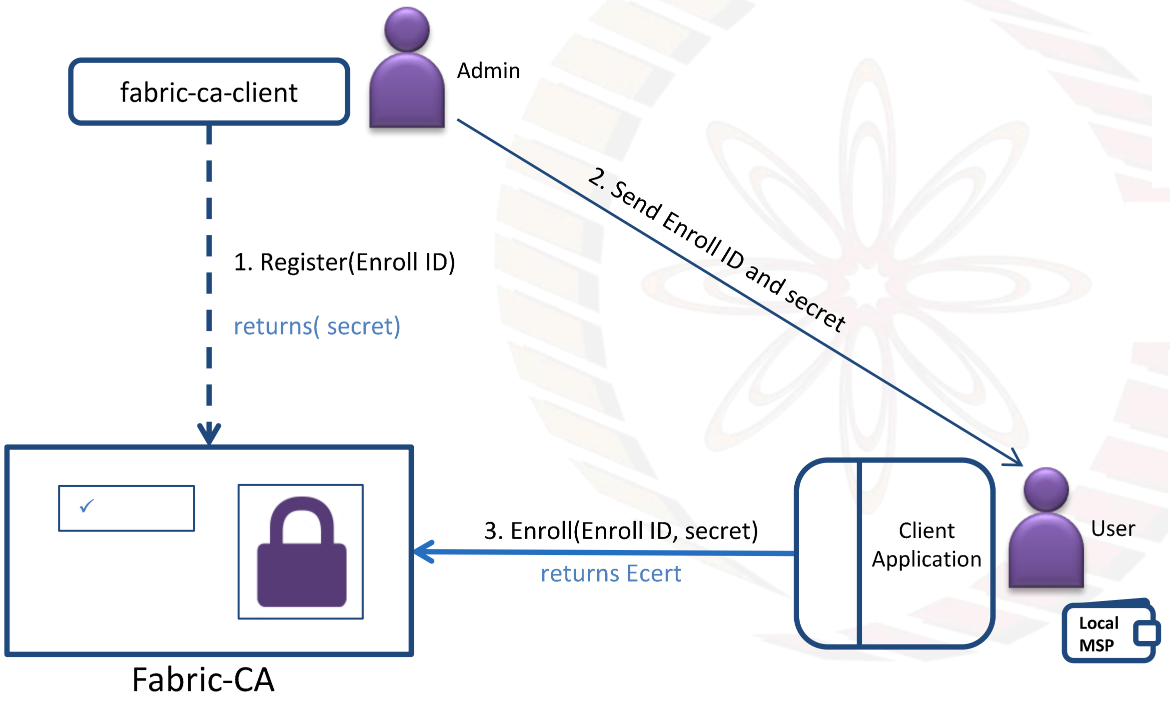
Channels include additional organisational MSP information

- Determines which orderers or peers can join the channel
- Determines client applications read or write access to the channel
- Stored in configuration blocks in the ledger
- Each channel MSP includes:
  - **admincerts**
    - Any public certificates for administrators
  - **cacerts**
    - The CA public certificate for this MSP
  - **crls**
    - List of revoked certificates
- Does not include any private keys for identity



ID = MSP1	
admincerts	admin.org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
crls	<list of revoked admin certificates>

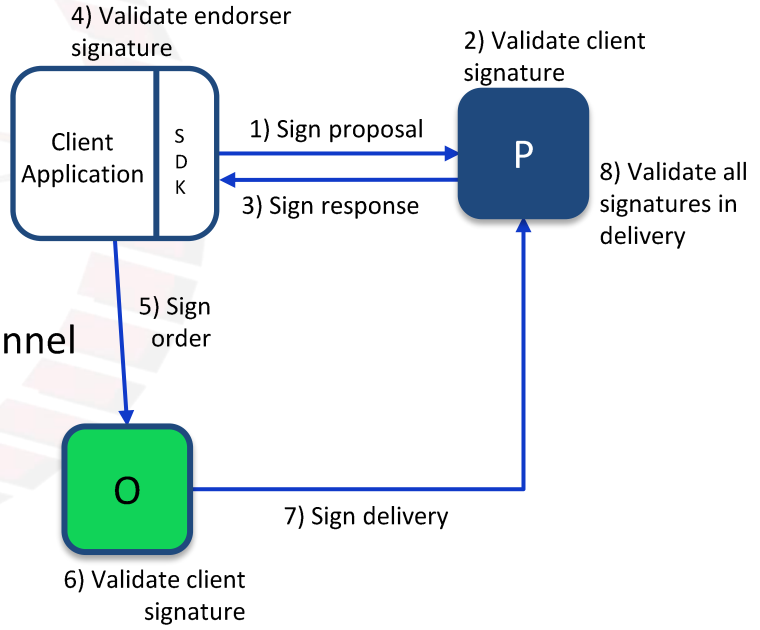
New User Registration and Enrollment



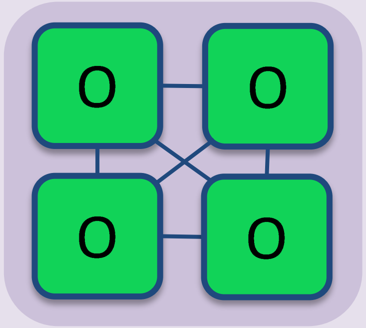
Transaction Signing

All transactions within a Hyperledger Fabric network are signed by permissioned actors, and those signatures are validated

- Actors sign transactions with their enrolment private key
  - Stored in their local MSP
- Components validate transactions and certificates
  - Root CA certificates and CRLs stored in local MSP
  - Root CA certificates and CRLs stored in Org MSP in channel



Step 1/6: Configure & Start Ordering Service



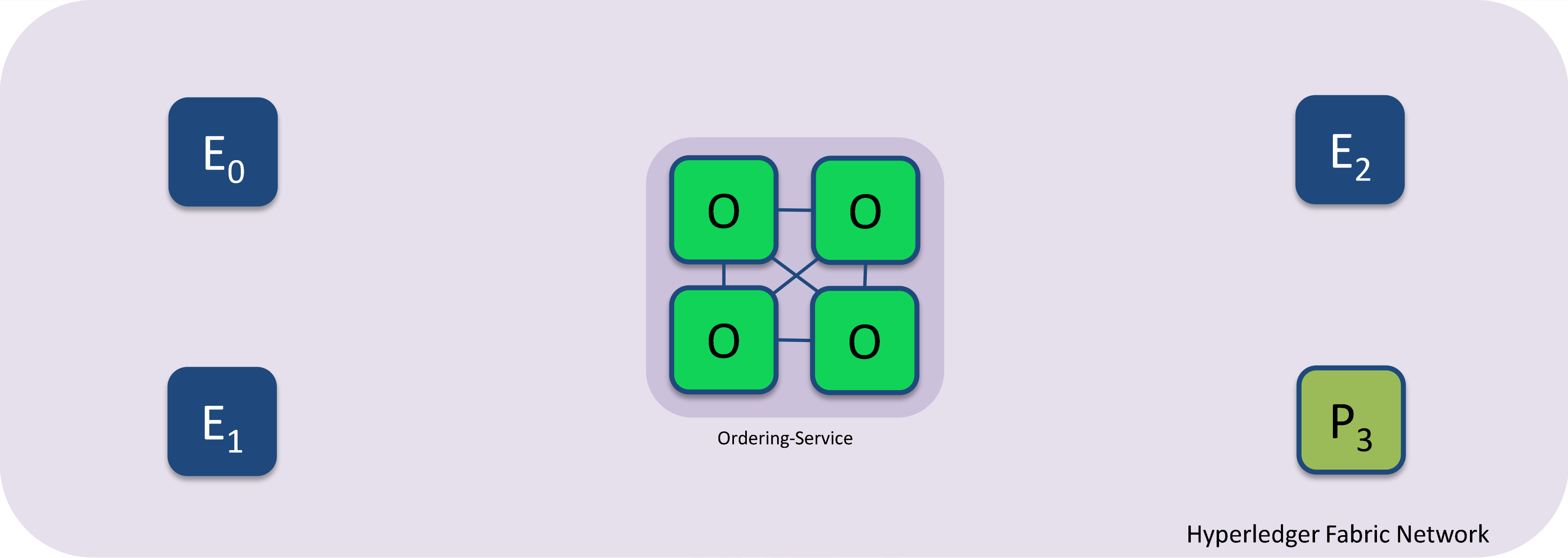
Ordering-Service

Hyperledger Fabric Network

An Ordering Service is configured and started for the network:

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

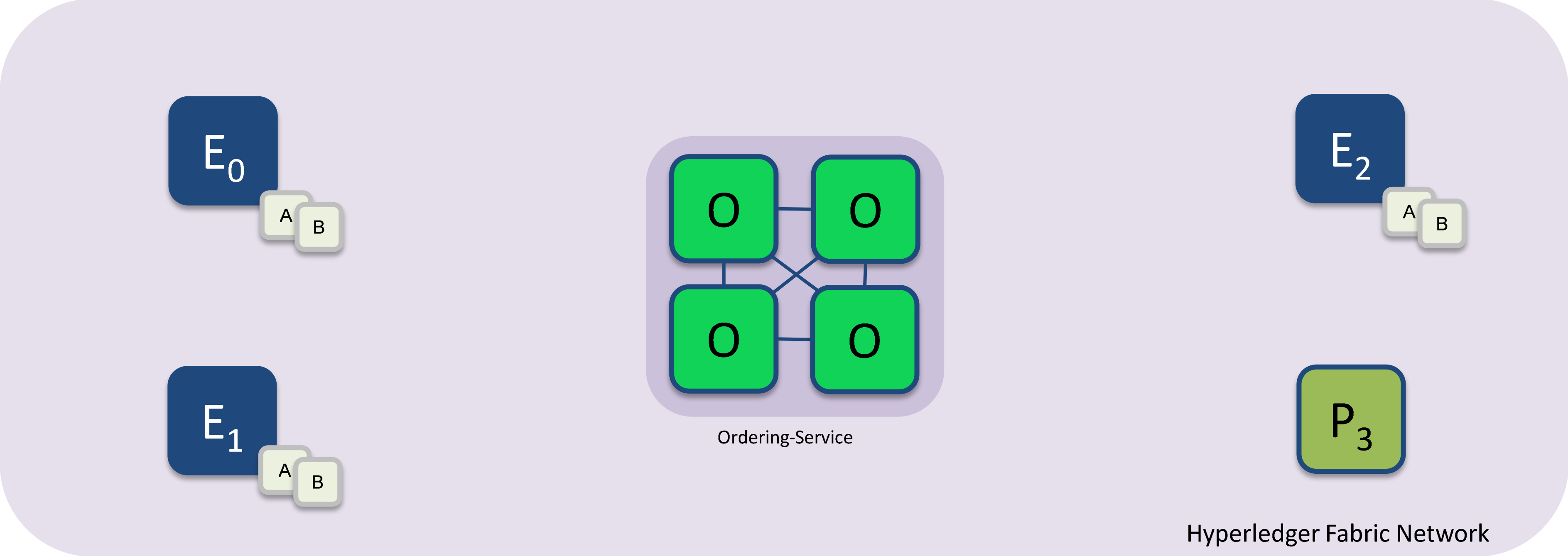
Step 2/6: Configure and Start Peer Nodes



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

$ peer node start ...

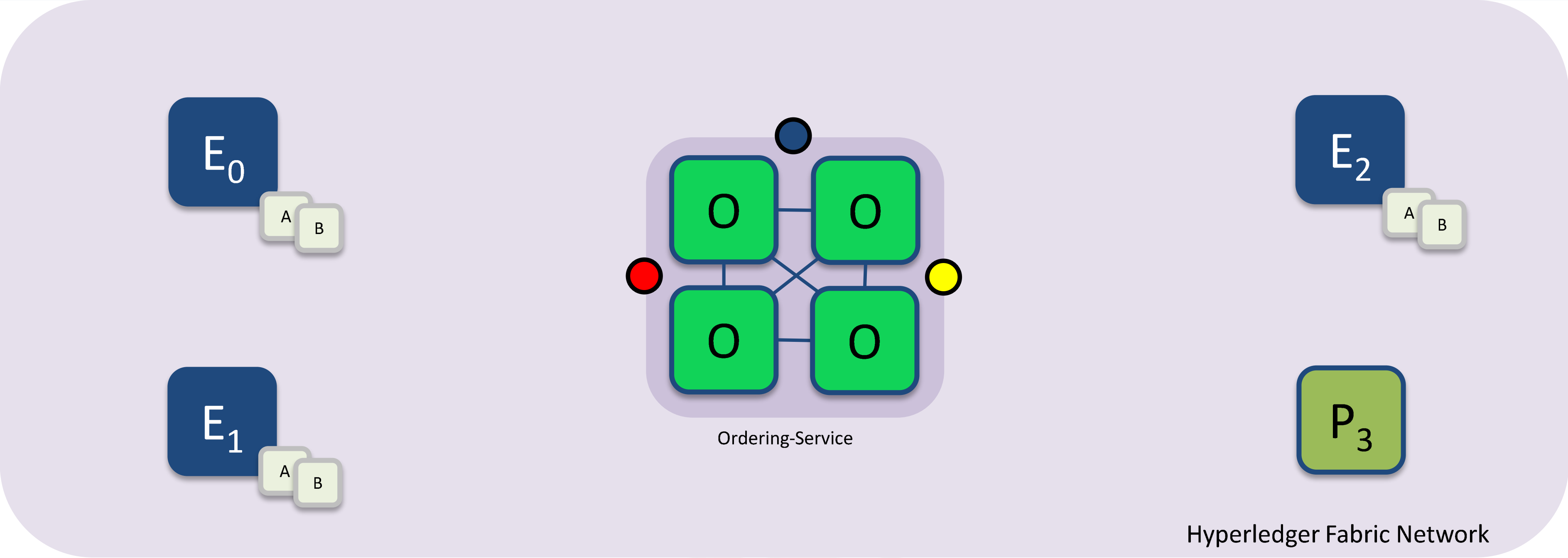
Step 3/6: Install Chaincode



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

$ peer chaincode install ...

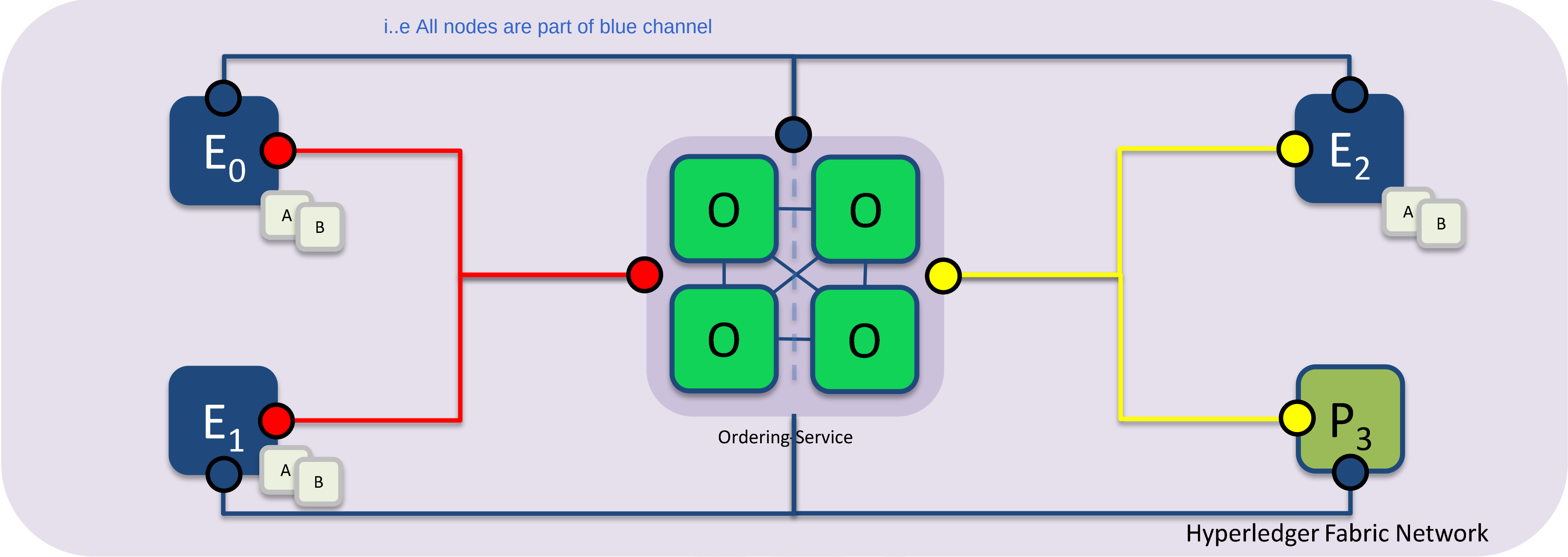
Step 4/6: Create Channels



Channels are created on the ordering service:

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

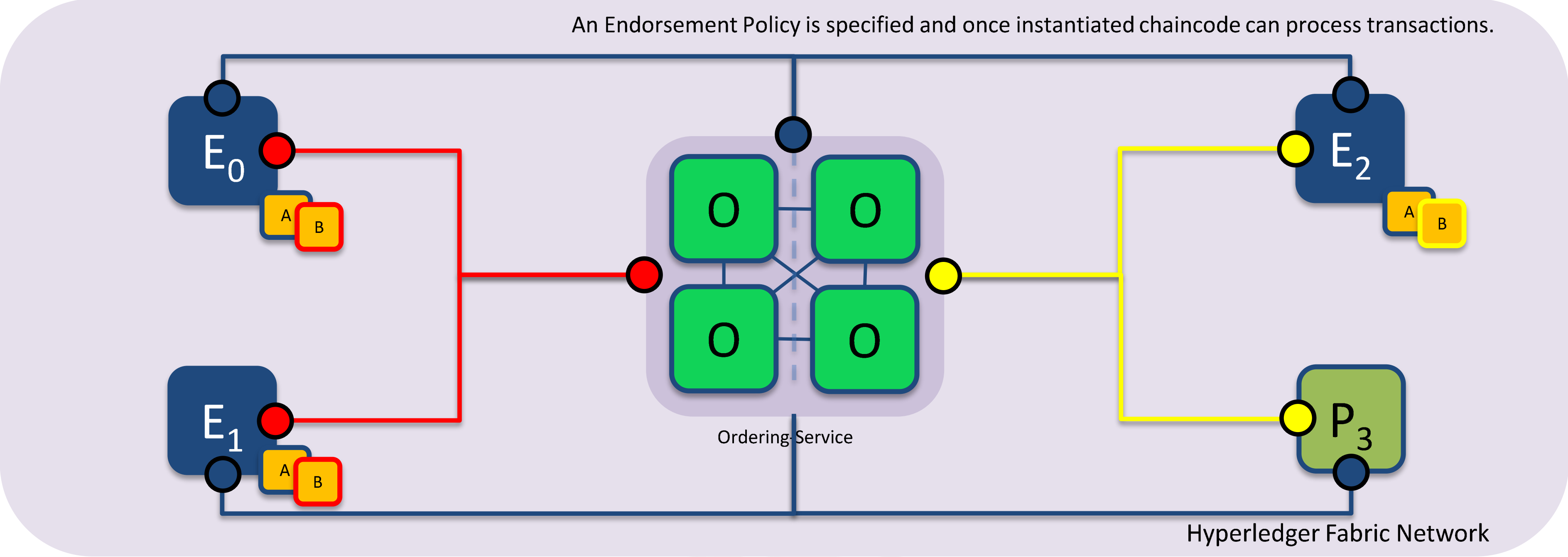
Join Channels



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

$ peer channel join ...

Step 6/6: Instantiate Chaincode in Channel

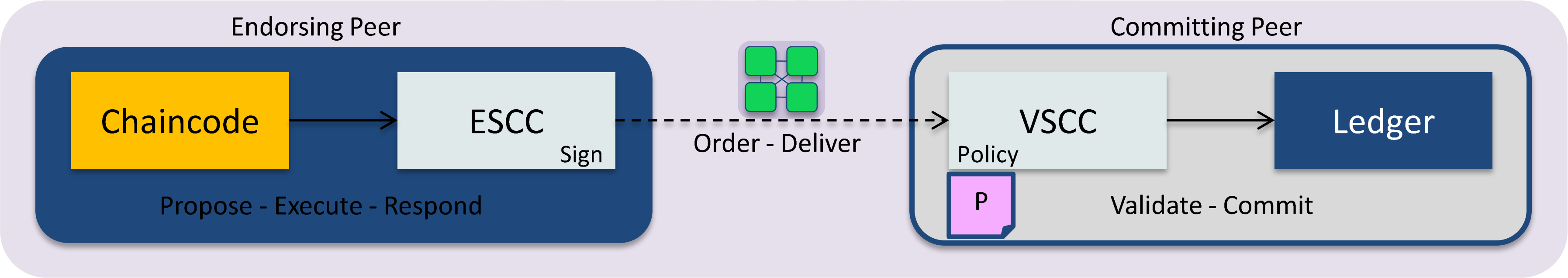


Peers finally instantiate the Chaincode on the channels they want to transact on:

$ peer chaincode instantiate … –P 'policy'

Endorsement Policies

- 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')

i.e. any member of Org1 can sign this transaction

-p refering to this file

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”

N-out-of-K policy specification also possible (e.g., 3 out of 5 peers in the channel must endorse)

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'))