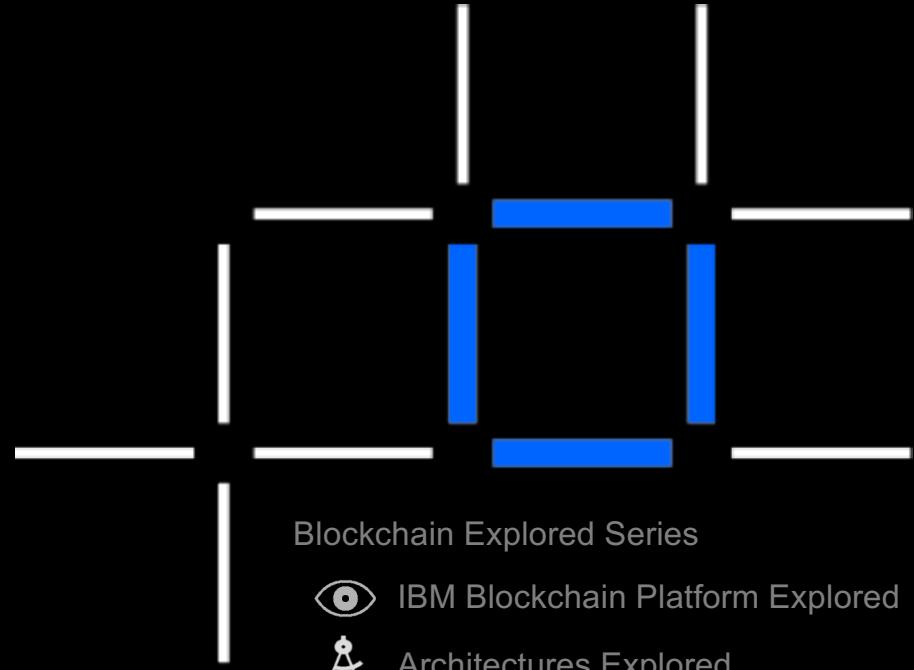


Fabric Explored

A Technical Deep-Dive on Hyperledger Fabric



Blockchain Explored Series

IBM Blockchain Platform Explored

Architectures Explored

Fabric Explored

Composer Explored

What's New

V4.4, 23 May 2018

IBM Blockchain

IBM



Project Status and
Roadmap

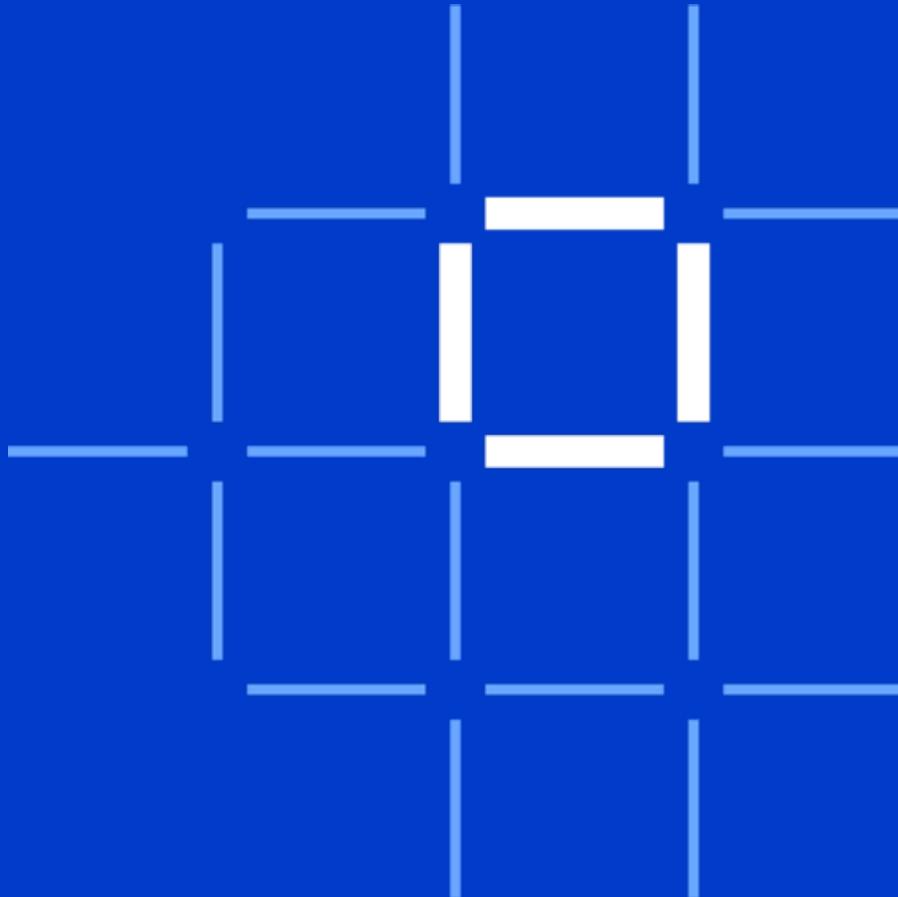


Technical Deep Dive



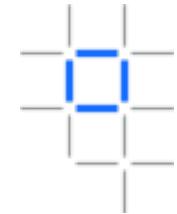
Futures

IBM Blockchain

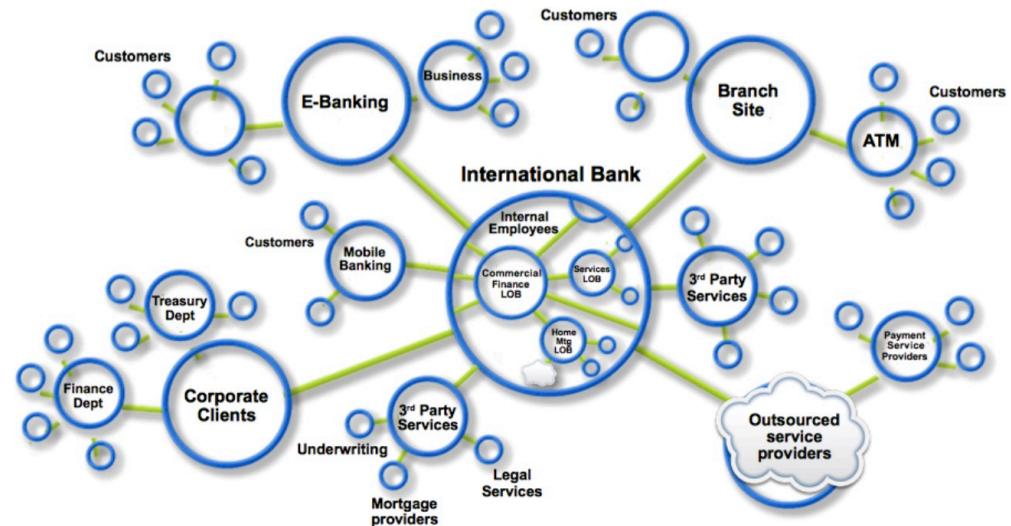


IBM

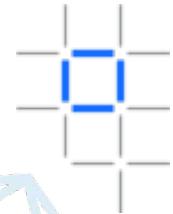
Blockchain Recap



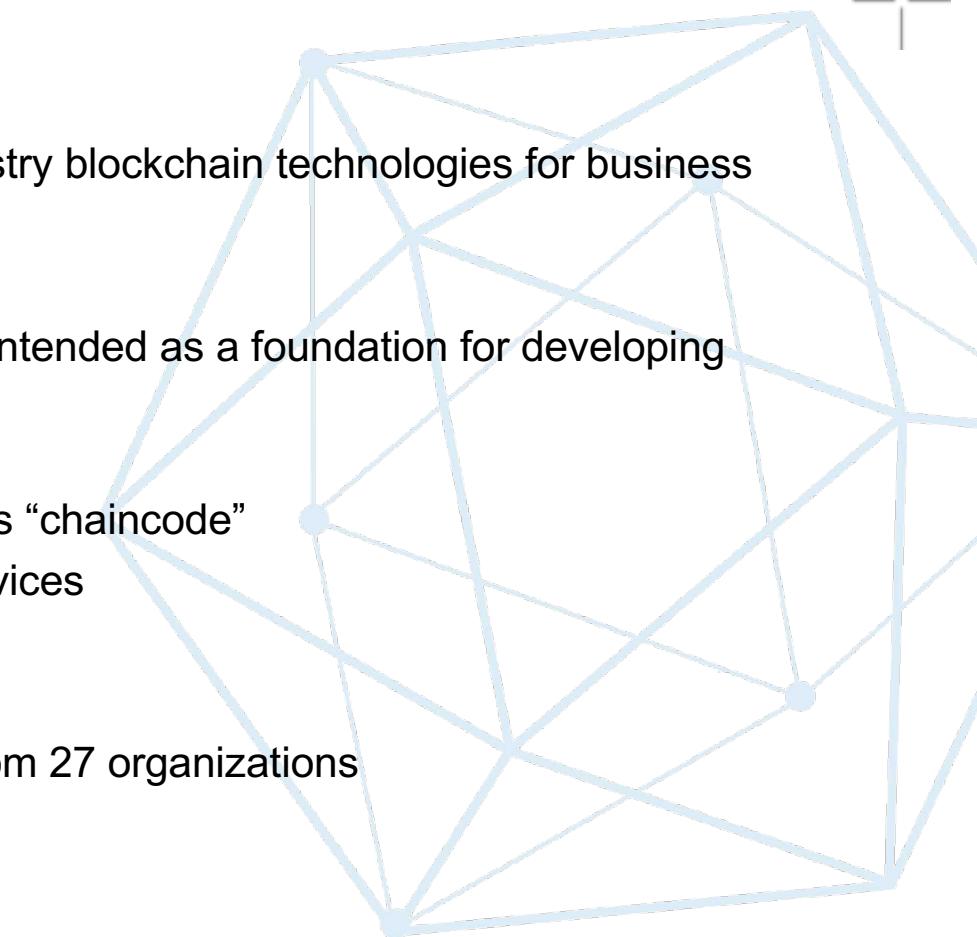
- Blockchain builds on basic business concepts
 - Business Networks connect businesses
 - Participants with Identity
 - Assets flow over business networks
 - Transactions describe asset exchange
 - Contracts underpin transactions
 - The ledger is a log of transactions
- Blockchain is a shared, replicated ledger
 - Consensus, immutability, finality, provenance



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



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

V1.1

- Node.js smart contracts
- Node.js connection profile
- Smart Contract APIs:
 - Encryption library
 - Txn submitter identity
 - Access control (using above)
- Performance & Scale
 - More orderers at scale
 - Parallel txn validation
 - CouchDB indexes
- Events
 - Per channel vs global
 - Block info minimal events
- CSR for more secure certs
- Serviceability
 - Upgrade from 1.0
- ***Technical Preview features***
 - Private channel data
 - Finer grained access control on channels (beyond orgs)
 - ZKP features (ID Mixer)
 - Java for Smart contracts

V1.2

- V1.1 Technical Preview features
 - Finalize Side DB - Private Data
 - Finalize Java chaincode
 - Finalize Fabric ACL mechanism
- Usability Features
 - e.g. Service discovery
- Technical Debt/Hygiene
 - e.g. testing frameworks
 - Parallel testing
 - More modular code
- Pluggable endorsement and validation
- State-based Endorsement
- Privacy-preserving state-based endorsement

Based on <https://wiki.hyperledger.org/projects/fabric/roadmap>

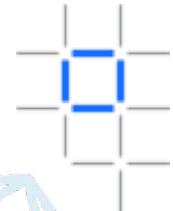
March 2017

July 2017

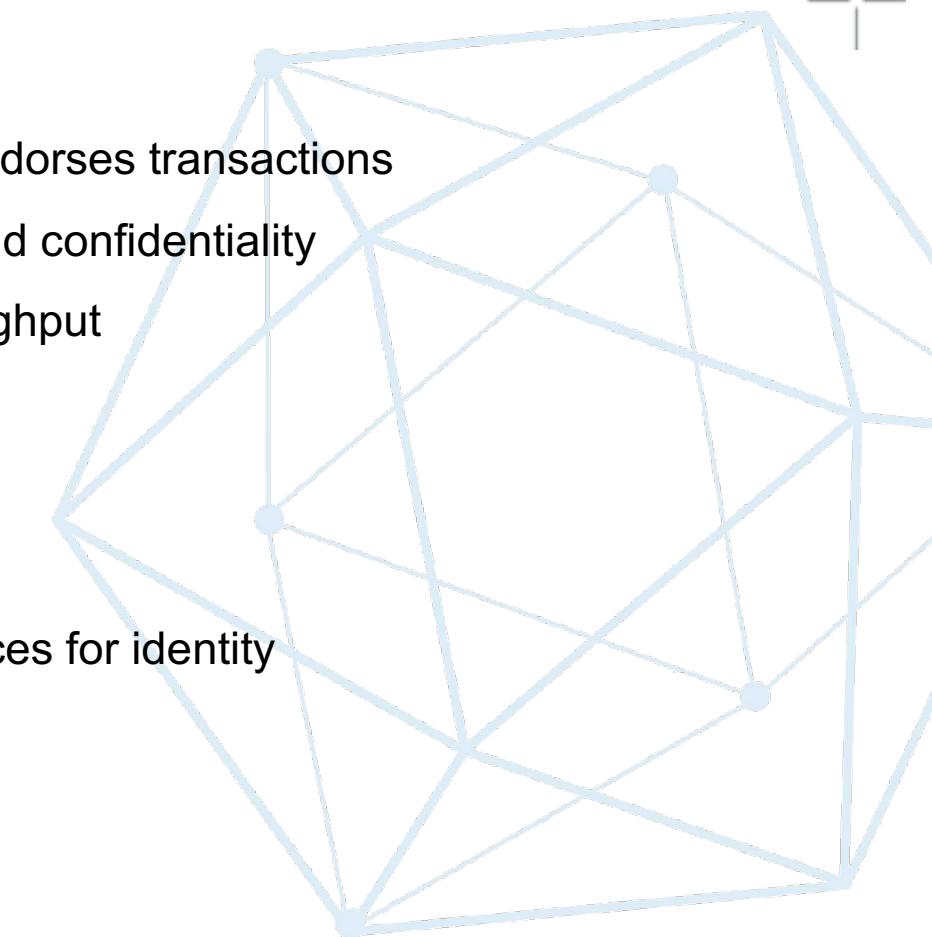
March 2018

June 2018 (quarterly)

Overview of Hyperledger Fabric v1 – Design Goals



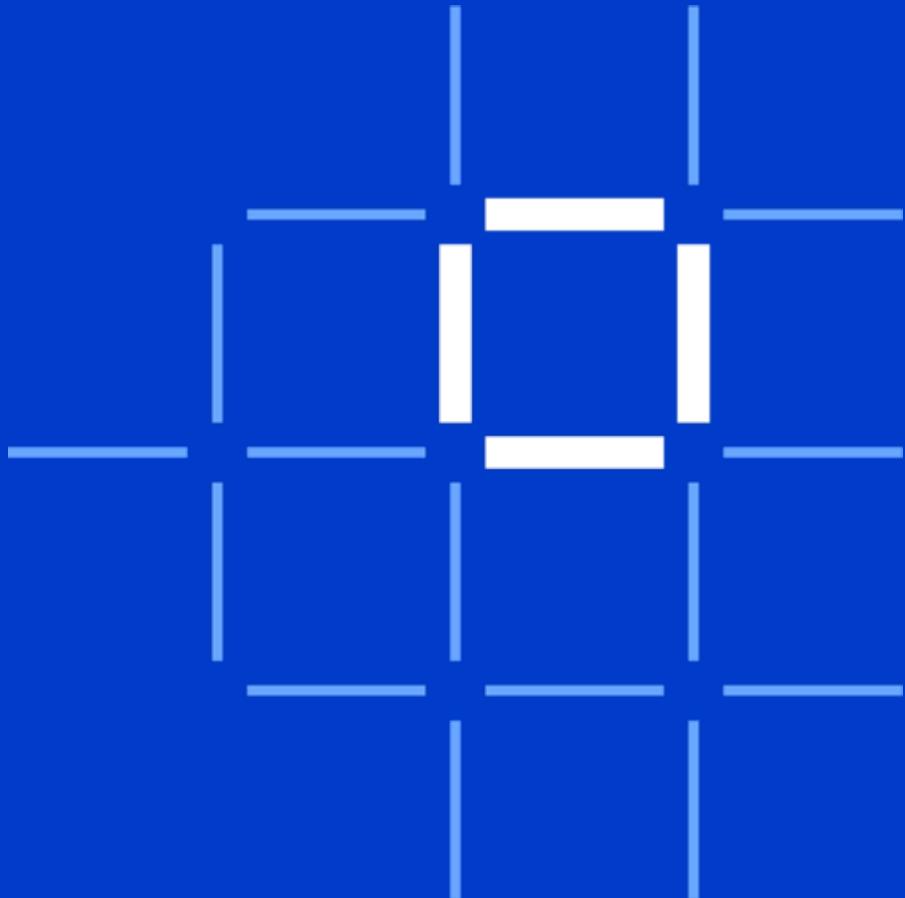
- Better reflect business processes by specifying who endorses transactions
- Support broader regulatory requirements for privacy and confidentiality
- Scale the number of participants and transaction throughput
- Eliminate non deterministic transactions
- Support rich data queries of the ledger
- Dynamically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



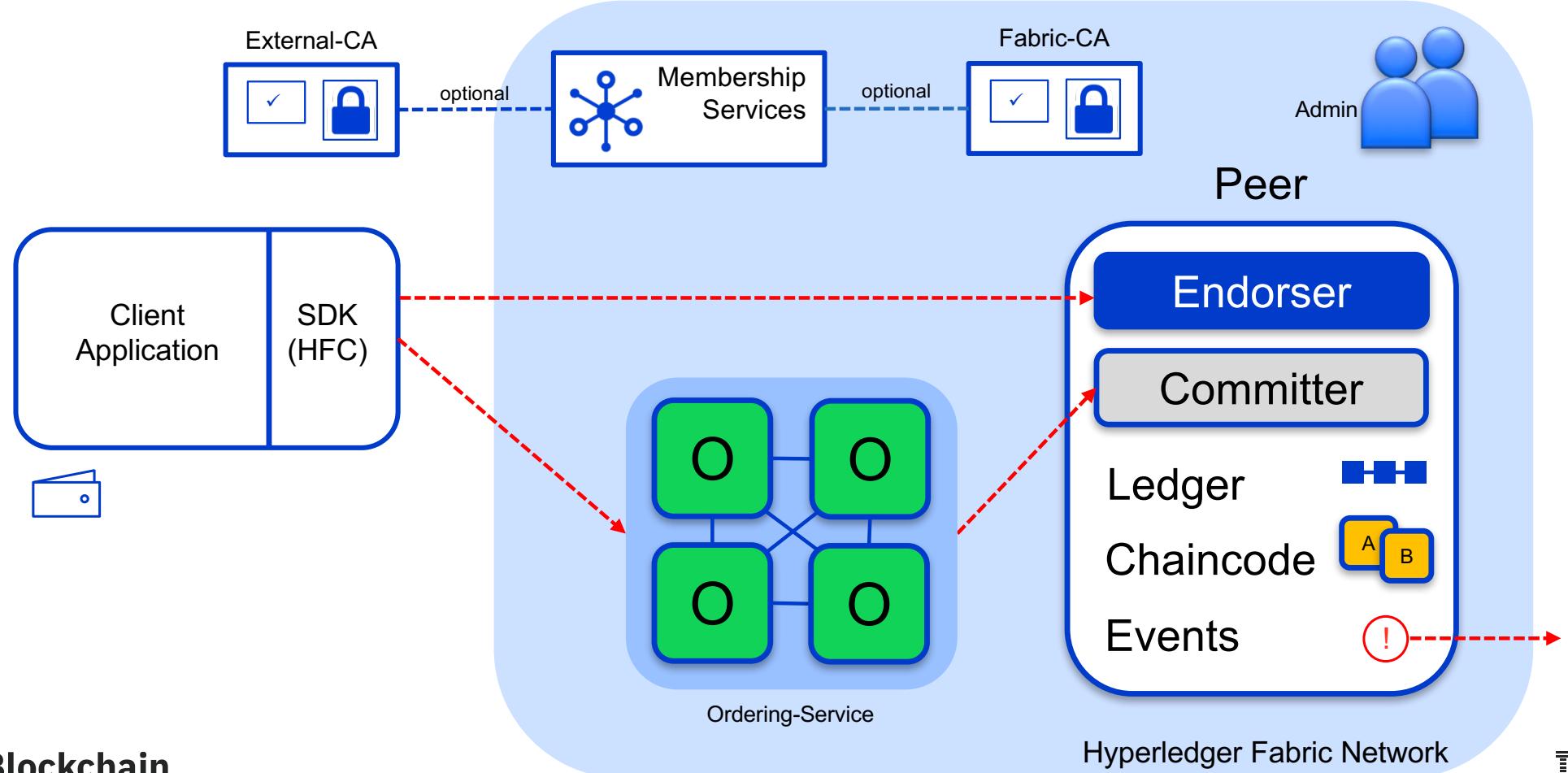
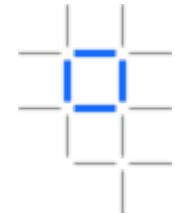


Technical Deep Dive

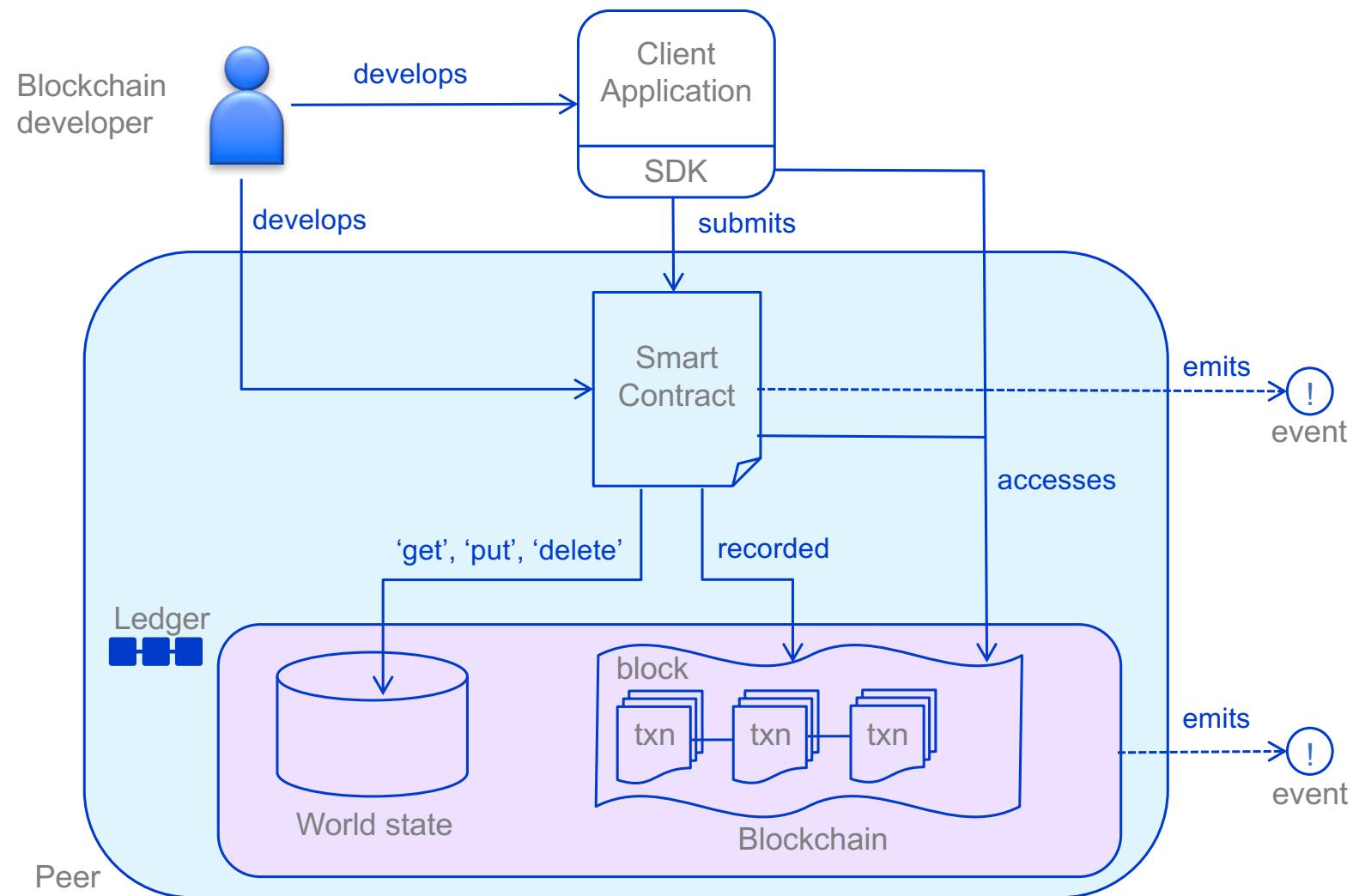
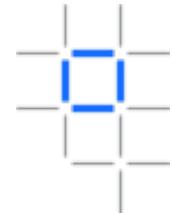
- [Architectural Overview]
- Network Consensus
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- Membership Services



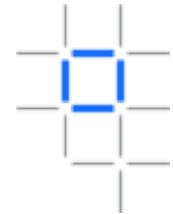
Hyperledger Fabric V1 Architecture



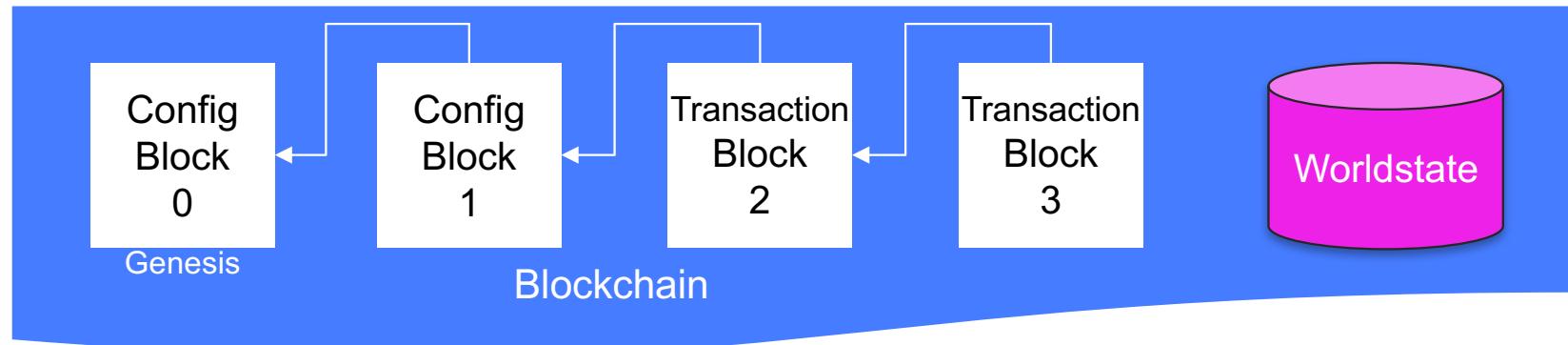
How applications interact with the ledger



Fabric Ledger



- The **Fabric ledger** is maintained by each peer and includes the **blockchain and worldstate**
- A separate ledger is maintained for each channel the peer joins
- Transaction **read/write sets** are written to the blockchain
- **Channel configurations** are also written to the blockchain
- The worldstate can be either LevelDB (default) or CouchDB
 - **LevelDB** is a simple key/value store
 - **CouchDB** is a document store that allows complex queries
- The smart contact Contract decides what is written to the worldstate



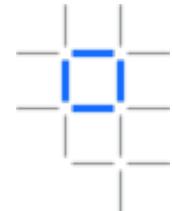


Technical Deep Dive

- Architectural Overview
- [Network Consensus]
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- Membership Services

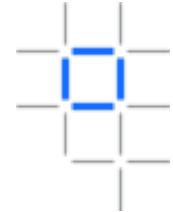


Nodes and roles

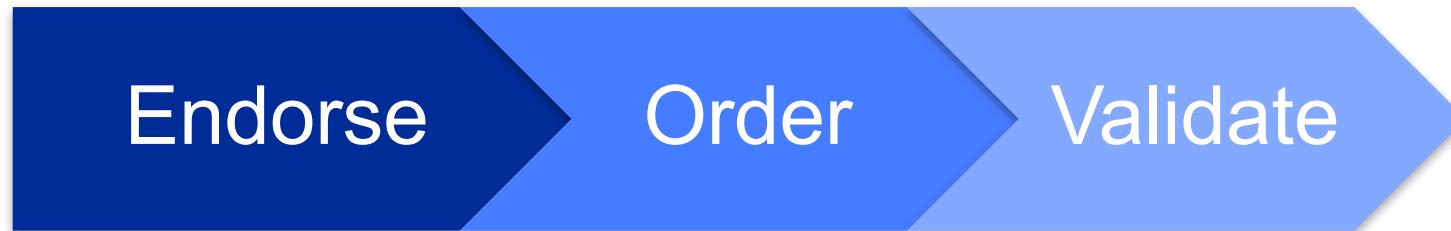


A light gray square with a blue double-line border.	Peer: Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).
A solid blue square with a blue double-line border.	Endorsing Peer: Specialized peer also endorses transactions by receiving a transaction proposal and responds by granting or denying endorsement. Must hold smart contract.
A solid green square with a blue double-line border.	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.

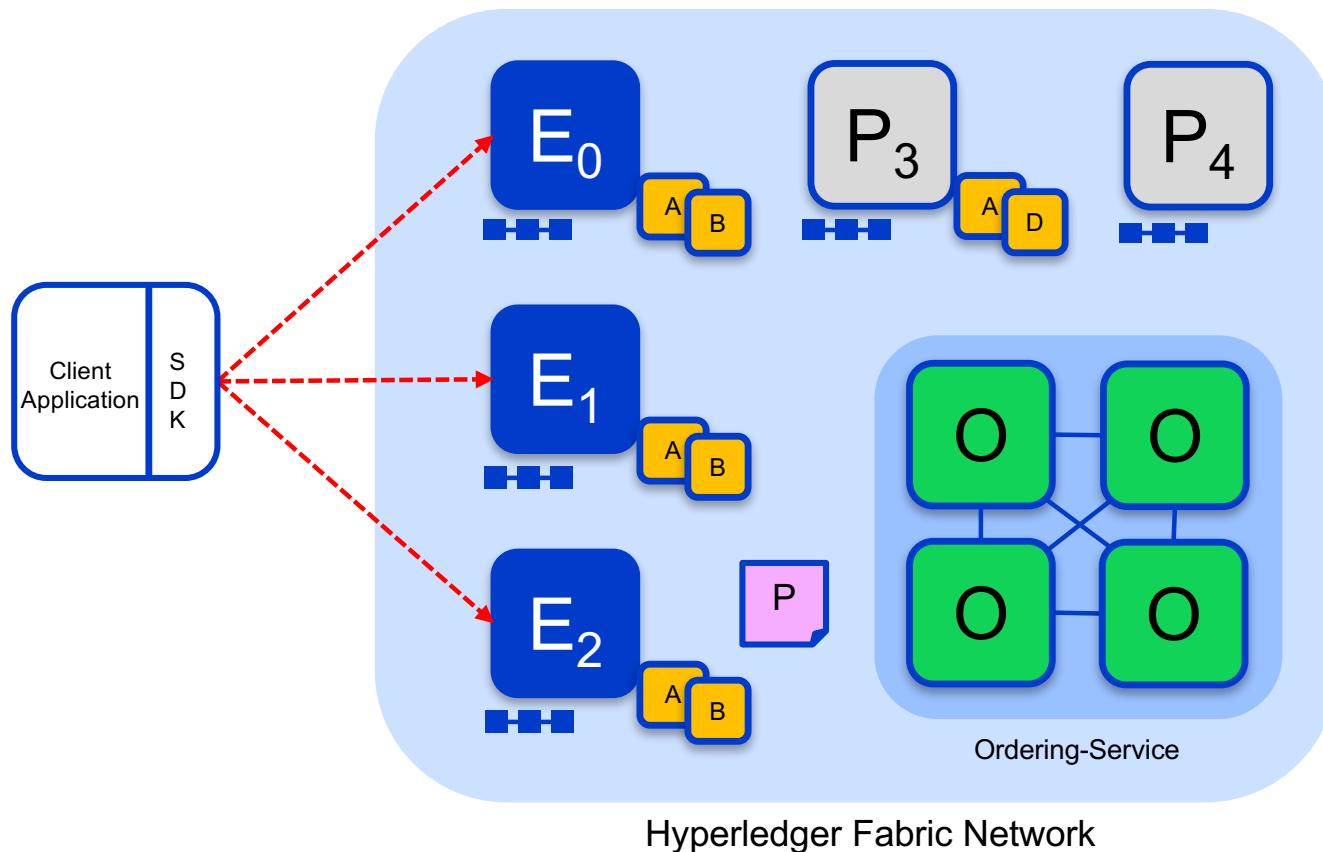
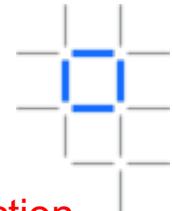
Hyperledger Fabric Consensus



Consensus is achieved using the following transaction flow:



Sample transaction: Step 1/7 – Propose transaction



Application proposes transaction

Endorsement policy:

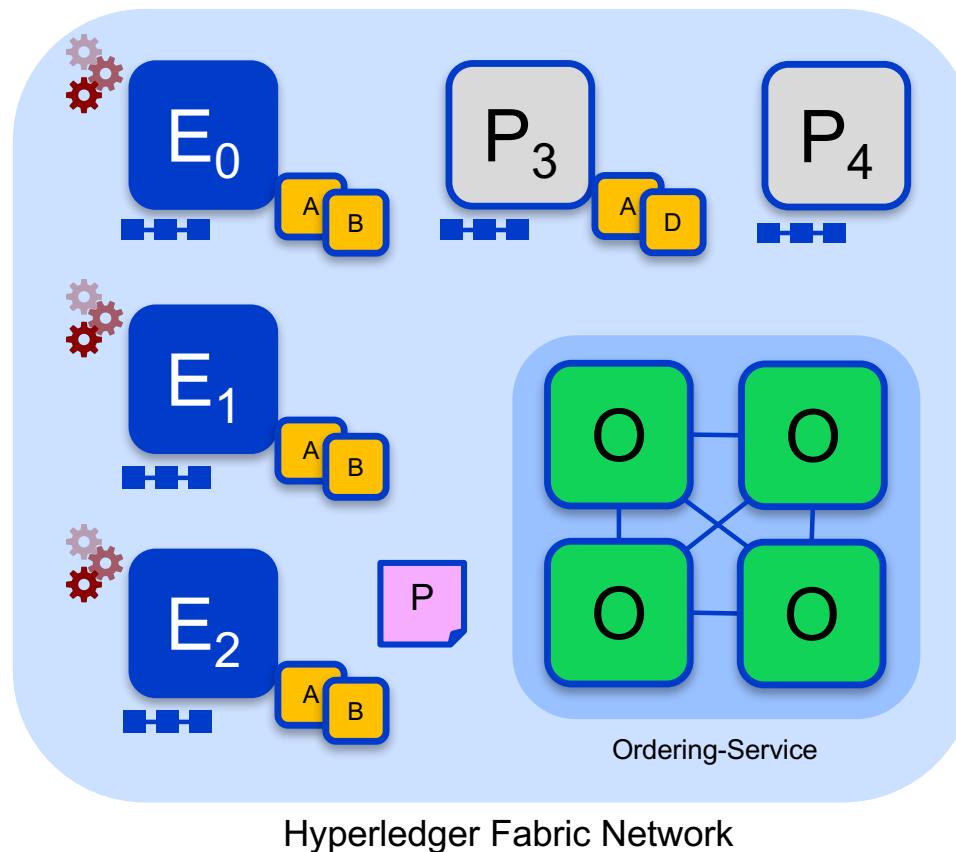
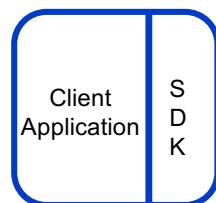
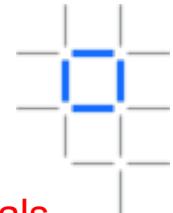
- “ E_0, E_1 and E_2 must sign”
- (P_3, P_4 are not part of the policy)

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

Key:

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

Sample transaction: Step 2/7 – Execute proposal



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.

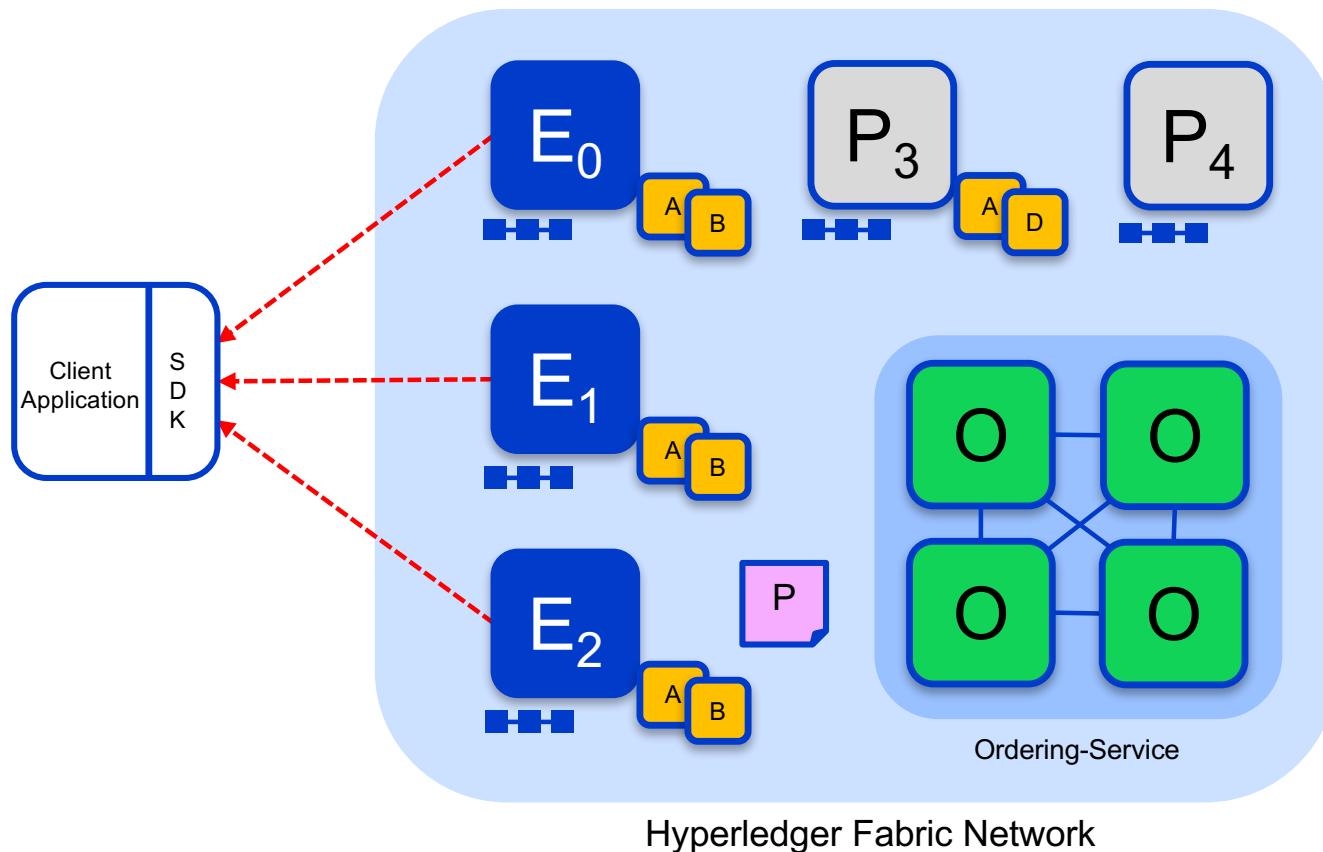
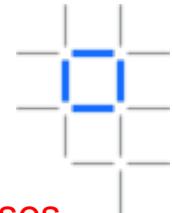
Transactions can be signed & encrypted

Key:

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



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

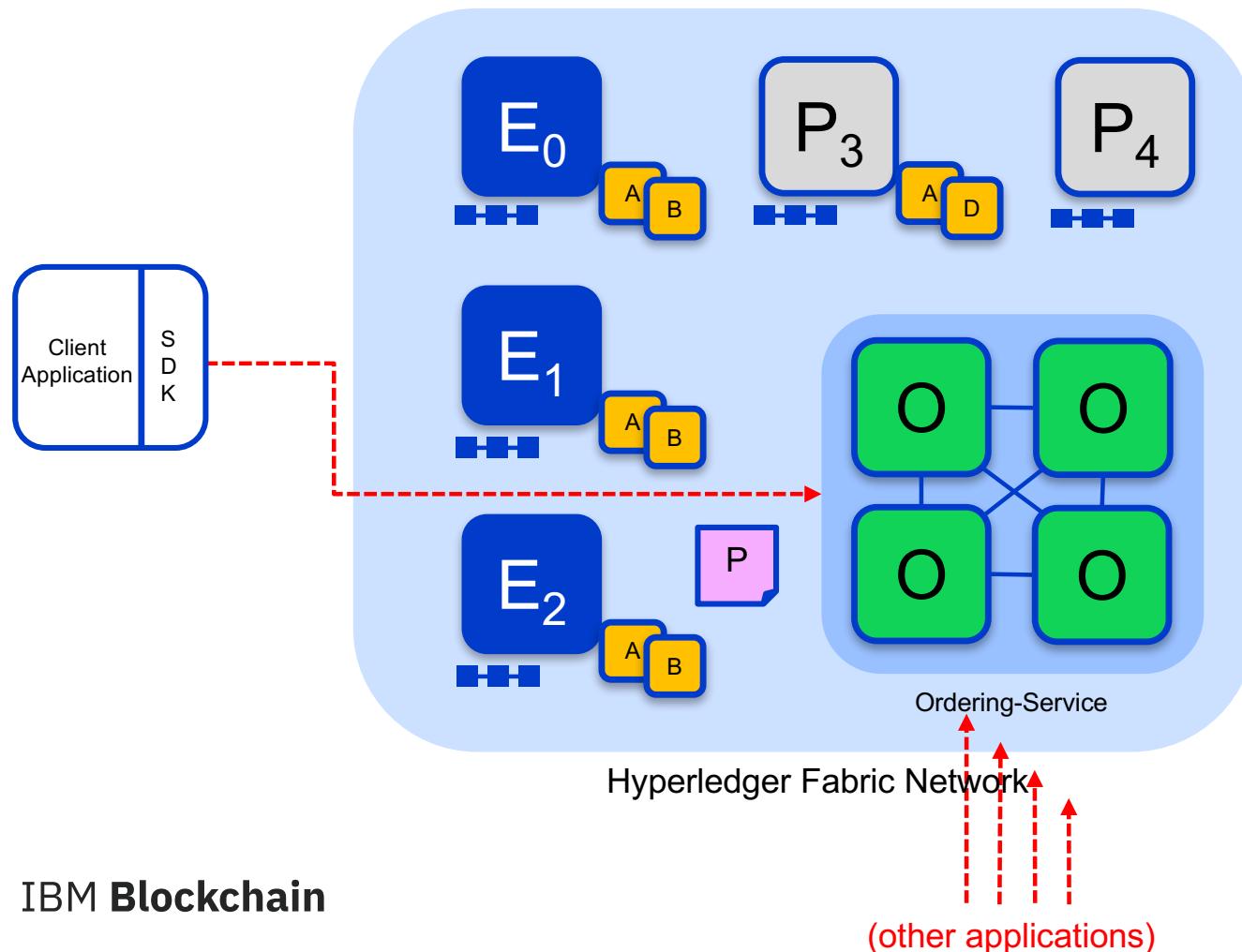
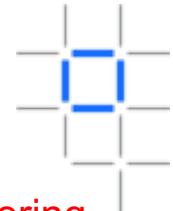
(This information will be checked much later in the consensus process)

Key:

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



Sample transaction: Step 4/7 – Order Transaction



Responses submitted for ordering

Application submits responses as a transaction to be ordered.

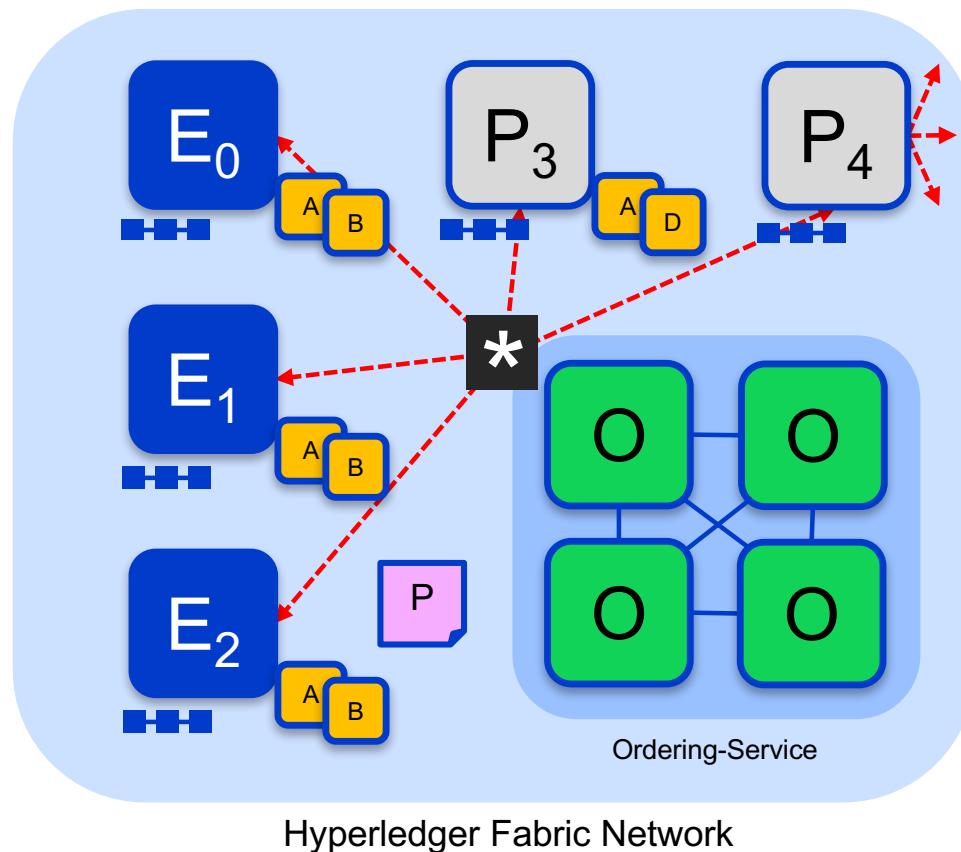
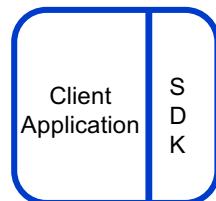
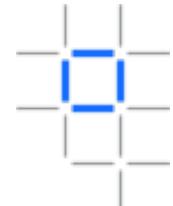
Ordering happens across the fabric in parallel with transactions submitted by other applications

Key:

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



Sample transaction: Step 5/7 – Deliver Transaction



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:

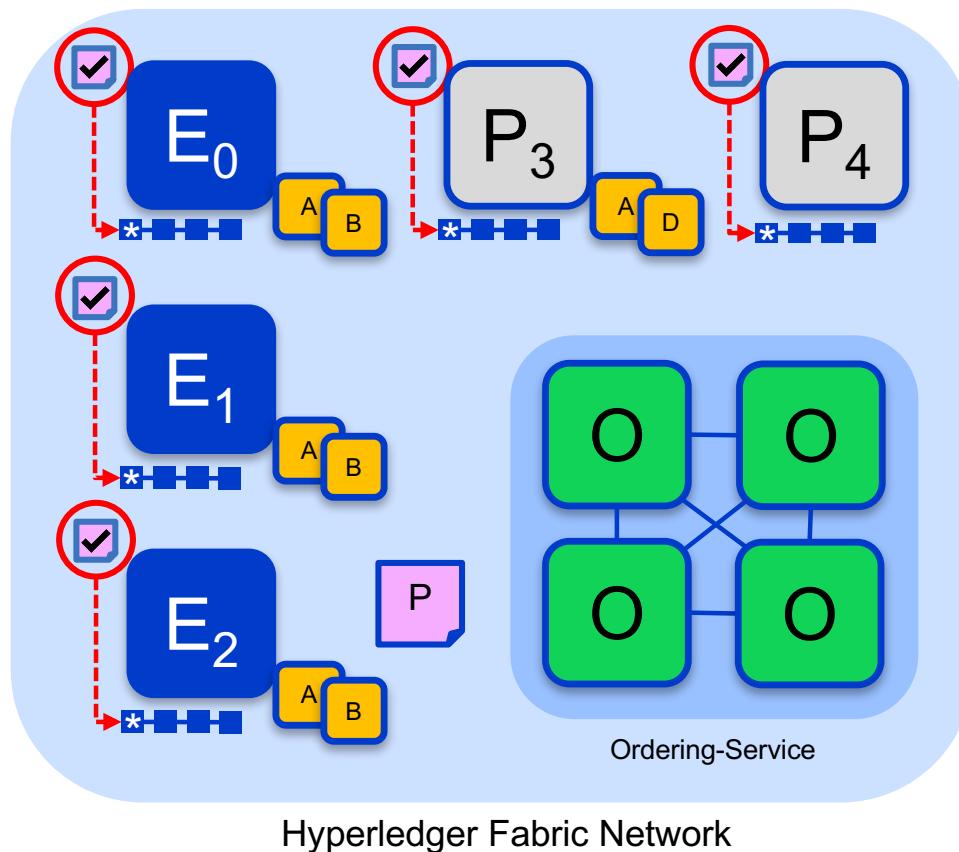
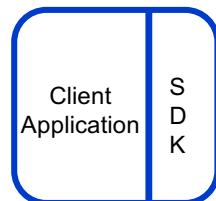
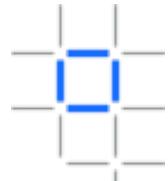
- SOLO (Single node, development)
- Kafka (Crash fault tolerance)

Key:

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



Sample transaction: Step 6/7 – Validate Transaction



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

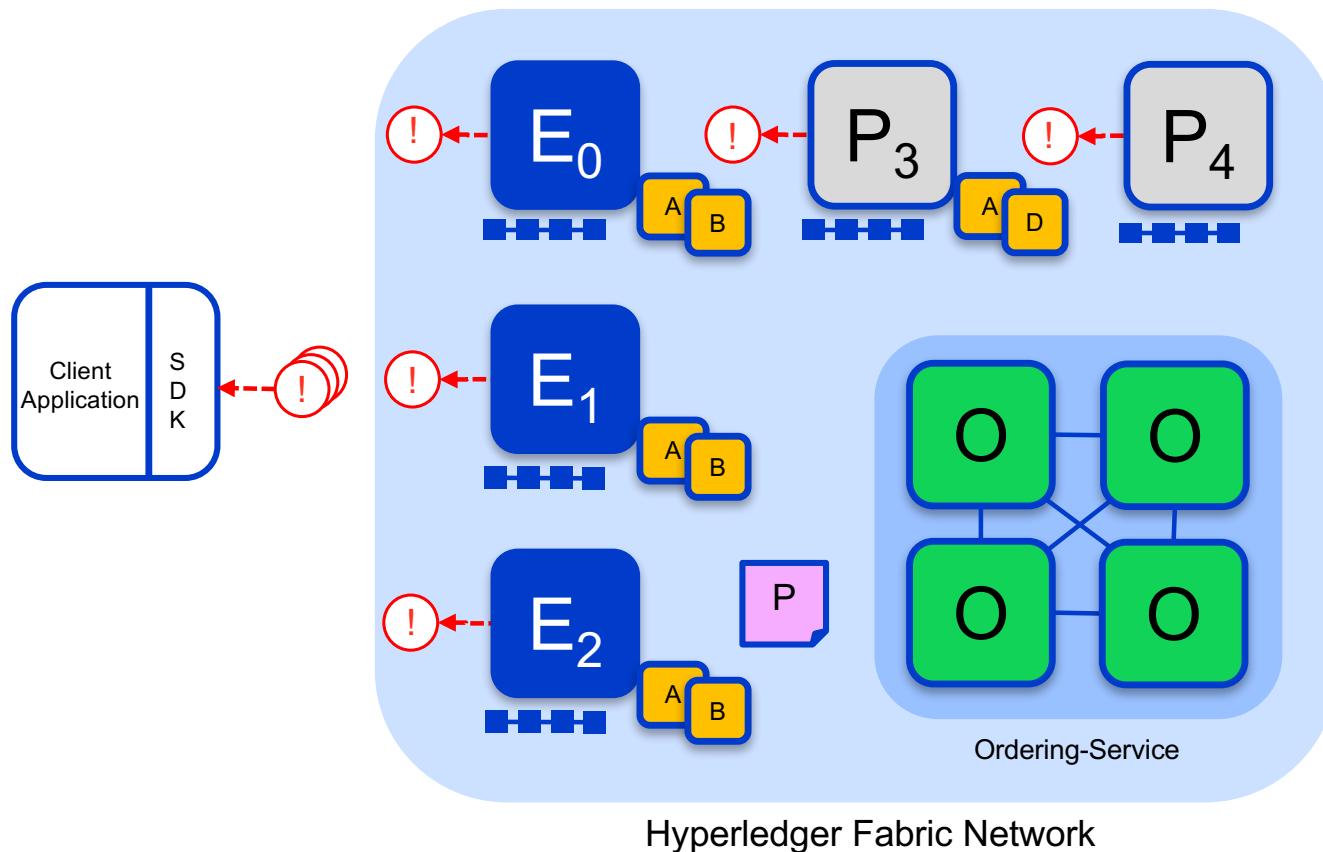
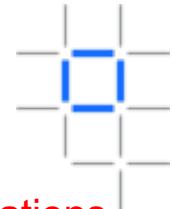
Invalid transactions are also retained on the ledger but do not update world state

Key:

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



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:

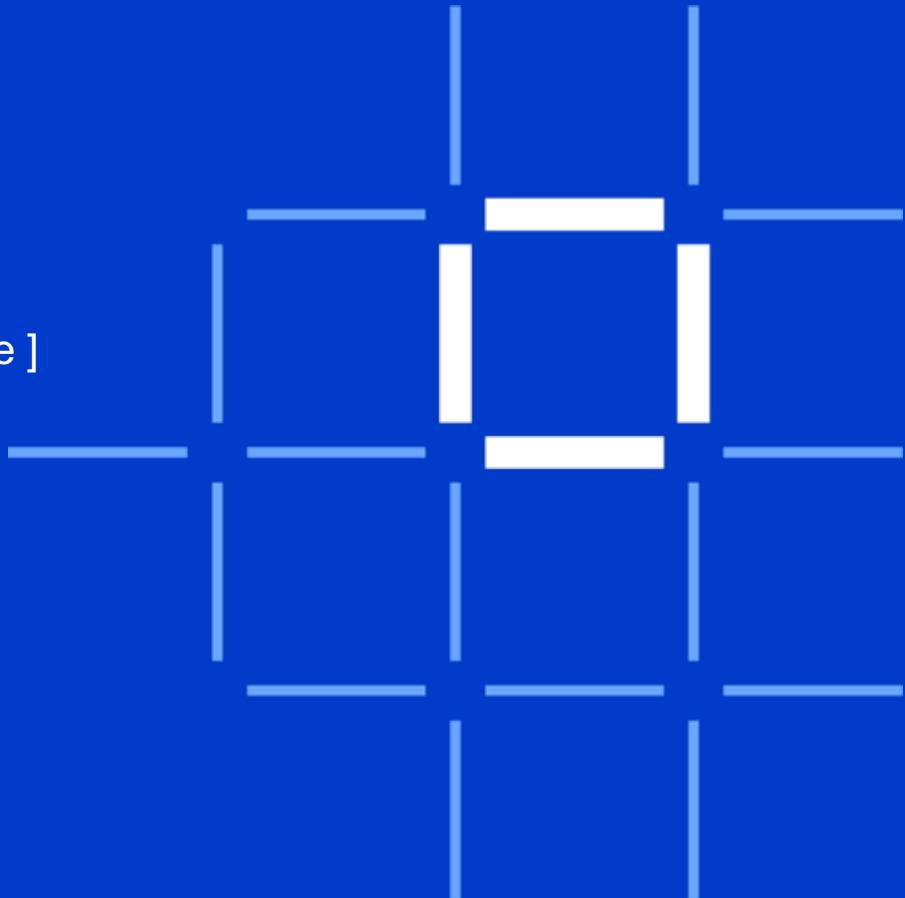
Endorser		Ledger
Committing Peer		Application
Ordering Node		
Smart Contract (Chaincode)		Endorsement Policy



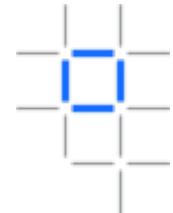


Technical Deep Dive

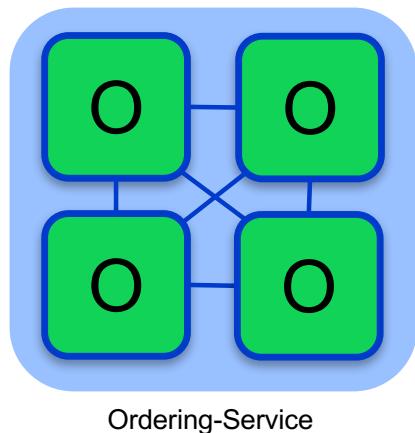
- Architectural Overview
- Network Consensus
- [Channels and Ordering Service]
- Components
- Network setup
- Endorsement Policies
- Membership Services



Ordering Service



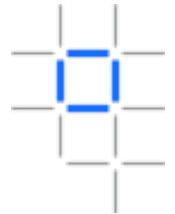
The ordering service packages transactions into blocks to be delivered to peers. Communication with the service is via channels.



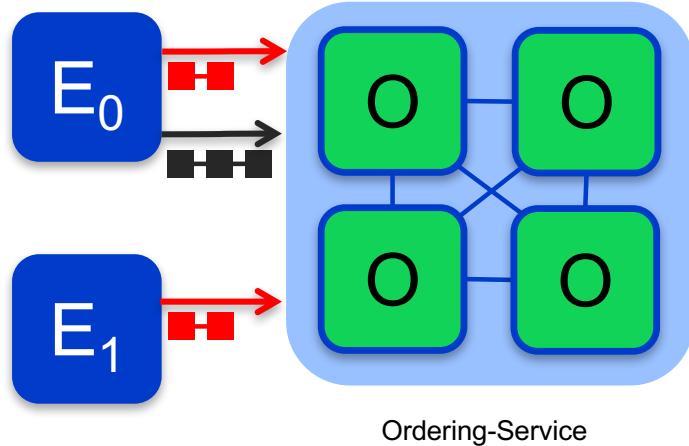
Different configuration options for the ordering service include:

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

Channels

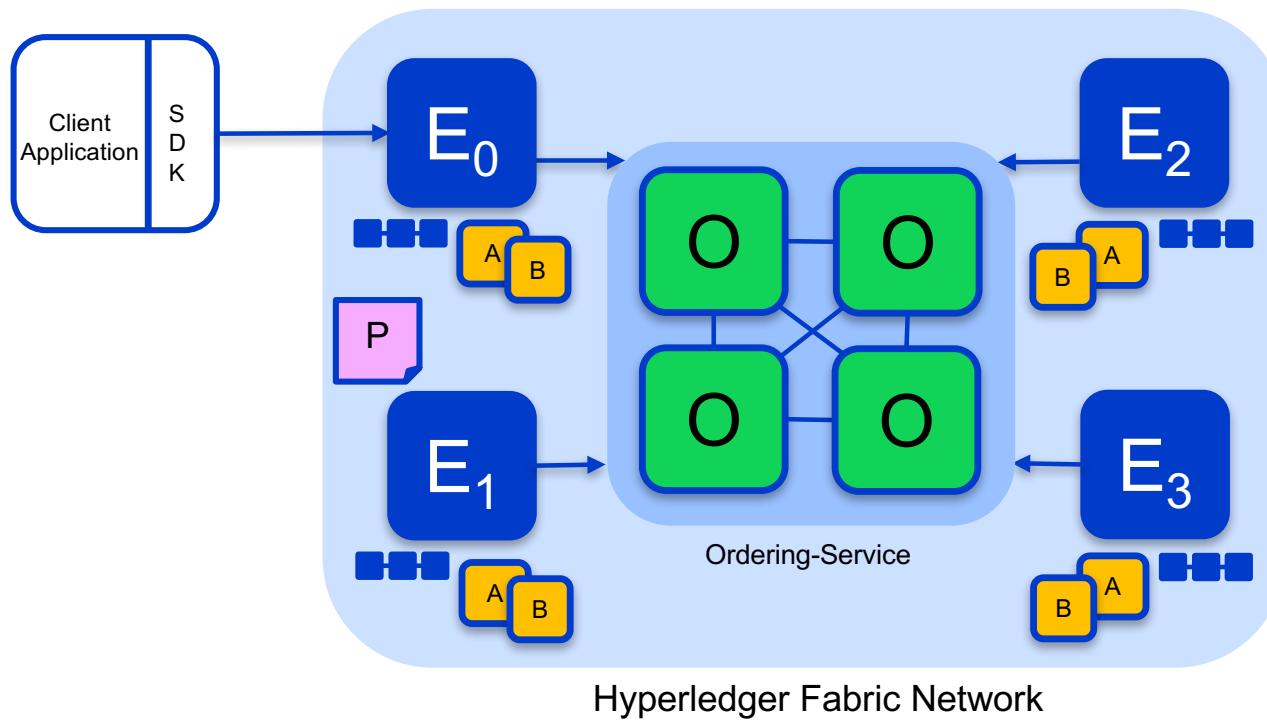
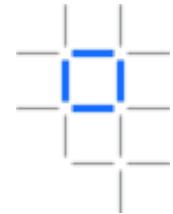


Channels provide privacy between different ledgers



- 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 specific channels
- Peers can participate in multiple channels
- Concurrent execution for performance and scalability

Single Channel Network

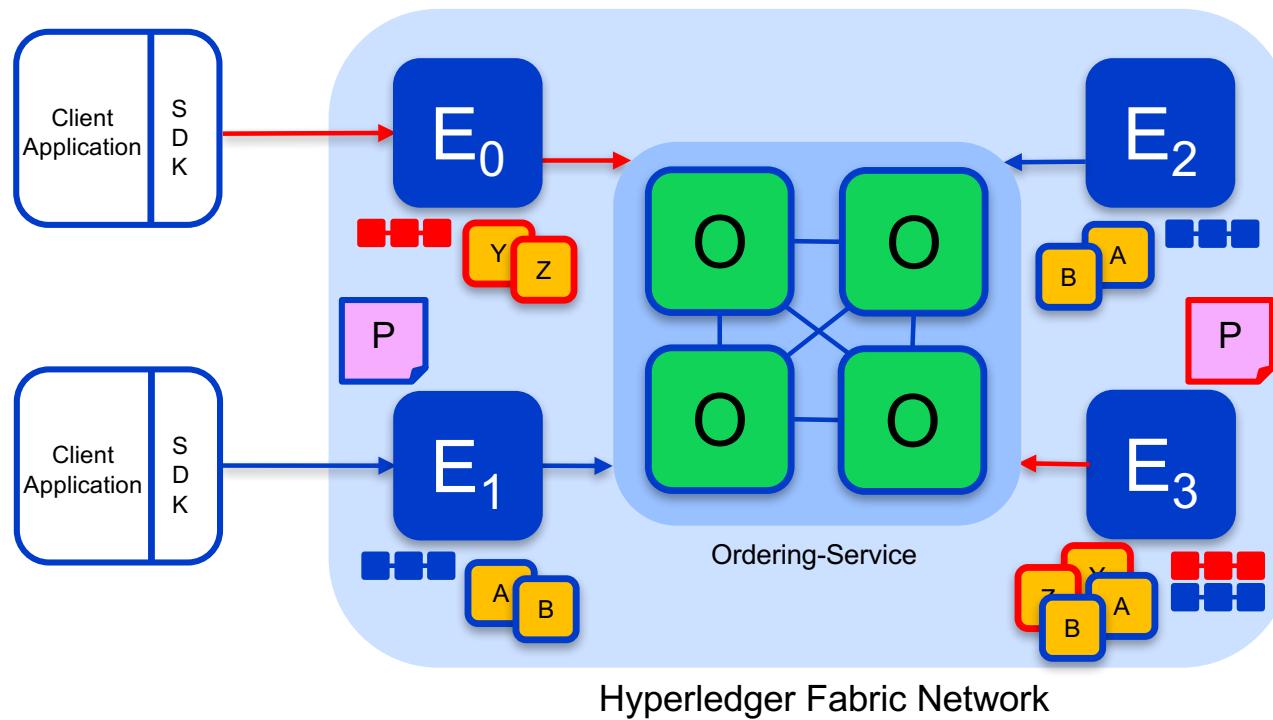
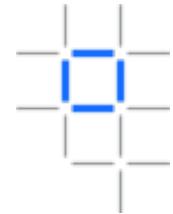


- Similar to v0.6 PBFT model
- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0 , E_1 , E_2 and E_3

Key:

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

Multi Channel Network



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

Key:

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



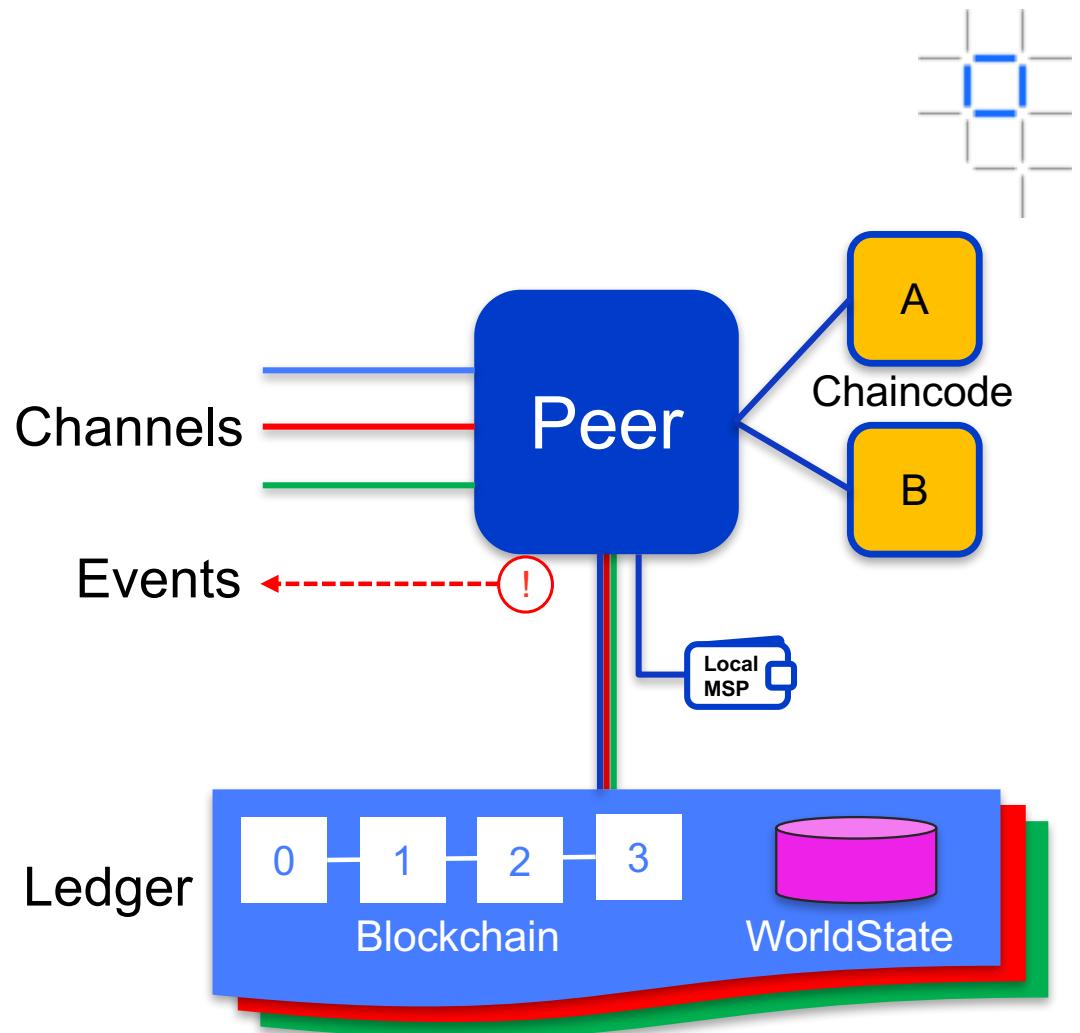
Technical Deep Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- [Components]
- Network setup
- Endorsement Policies
- Membership Services

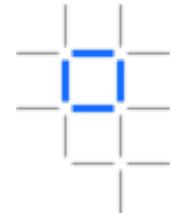


Fabric Peer

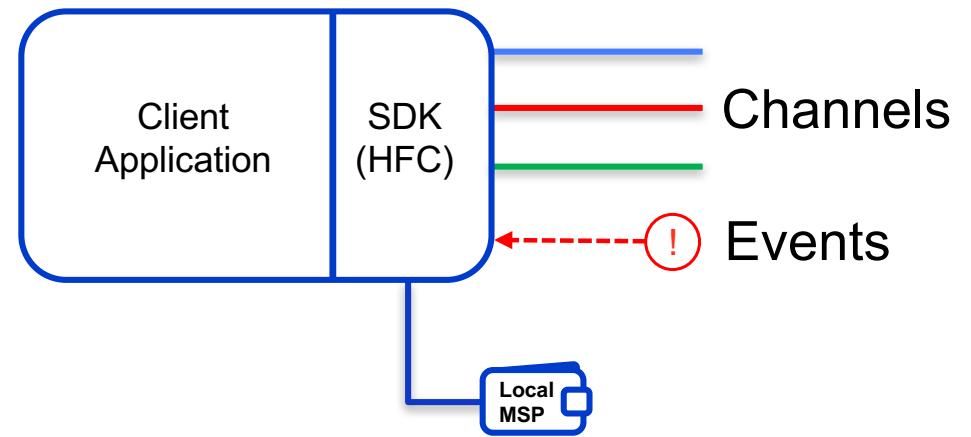
- Each peer:
 - Connects to one or more **channels**
 - Maintains one or more **ledgers** per channel
 - Maintains **installed chaincode**
 - Manages **runtime docker containers** for **instantiated chaincode**
 - Chaincode is instantiated on a channel
 - Runtime docker container shared by channels with same chaincode instantiated (no state stored in container)
 - Has a local MSP (Membership Services Provider) that provides **crypto material**
 - **Emits events** to the client application

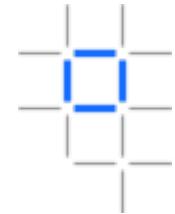


Client Application



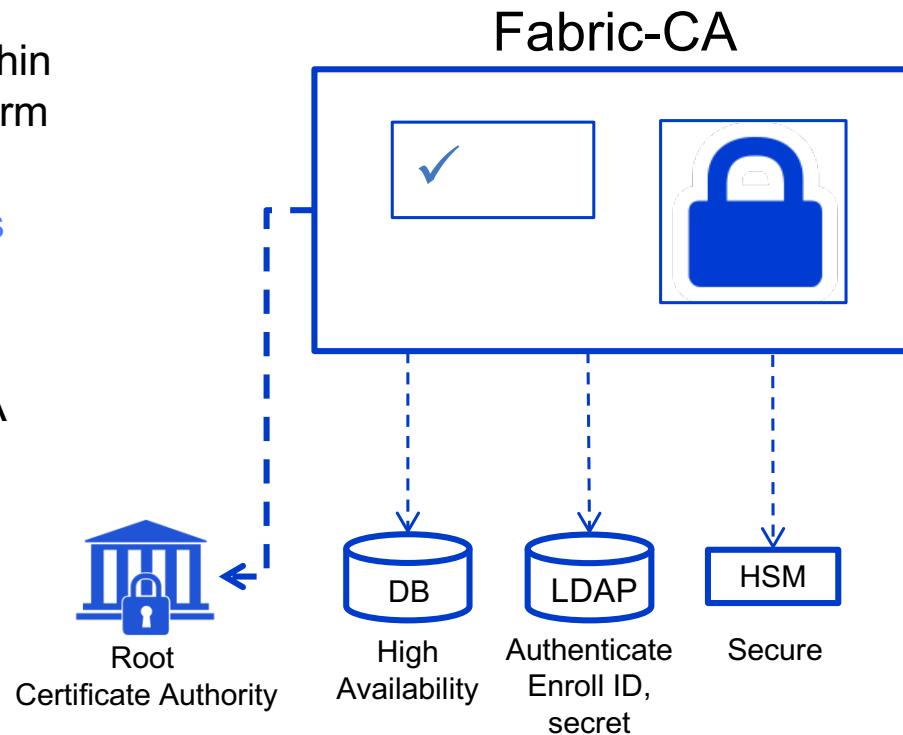
- Each client application uses Fabric SDK to:
 - Connects over channels to one or more peers
 - Connects over channels to one or more orderer nodes
 - Receives events from peers
 - Local MSP provides client [crypto material](#)
- Client can be written in different languages
(Node.js, Go, Java, Python?)





Fabric-CA

- 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



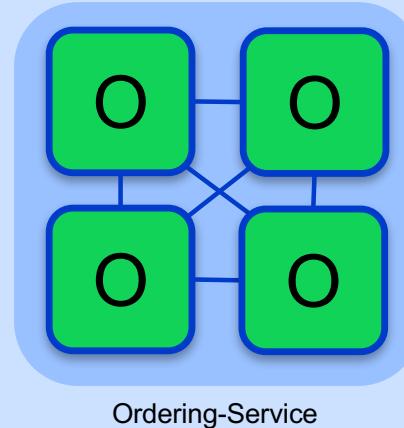
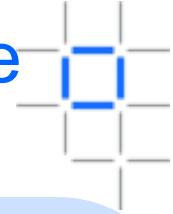


Technical Deep Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- Components
- [Network setup]
- Endorsement Policies
- Membership Services



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

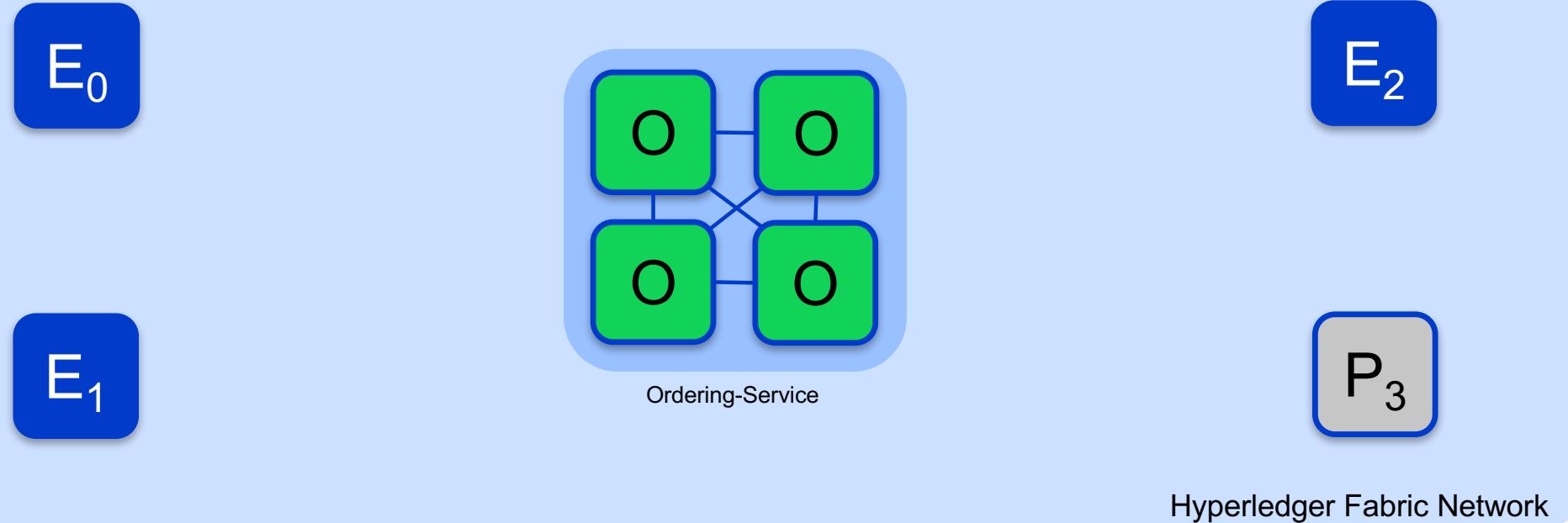
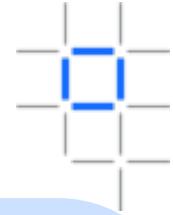


Hyperledger Fabric Network

An Ordering Service is configured and started for the network:

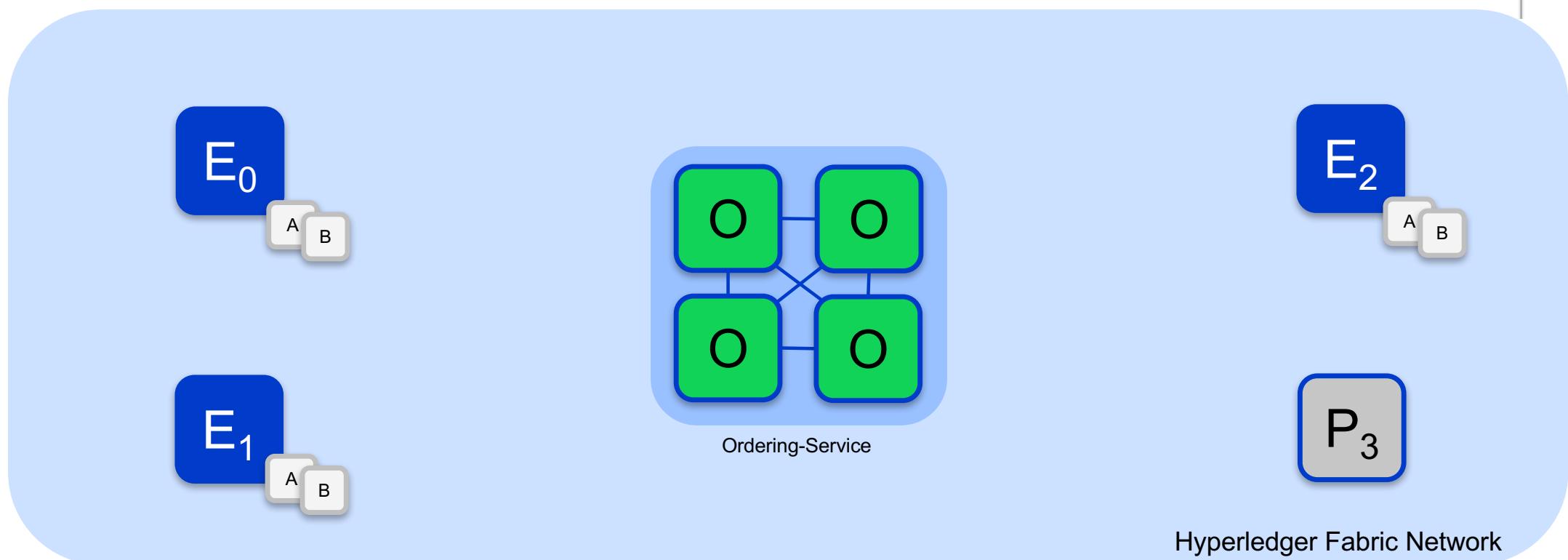
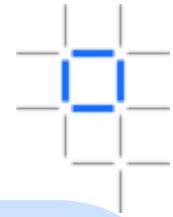
\$ docker-compose [-f orderer.yml] ...

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



A peer is configured and started for each Endorser or Committer in the network:
\$ peer node start ...

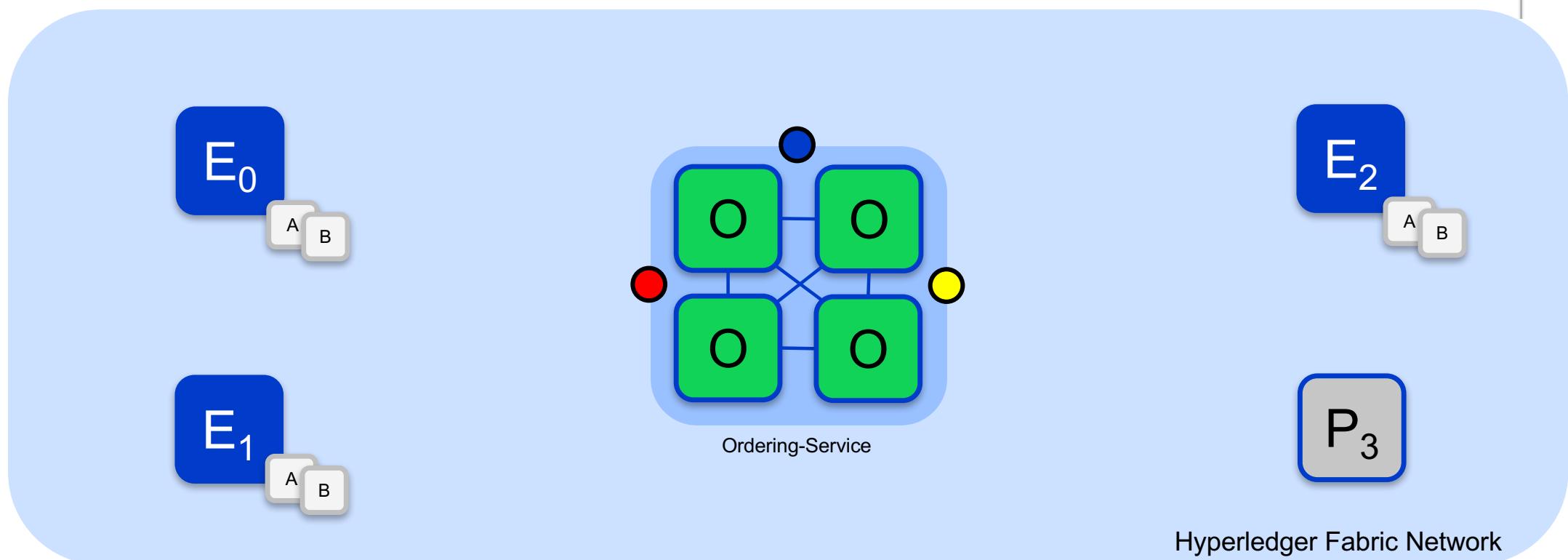
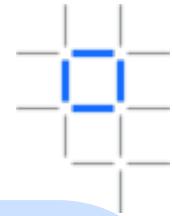
Bootstrap Network (3/6) - Install Chaincode



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

\$ peer chaincode install ...

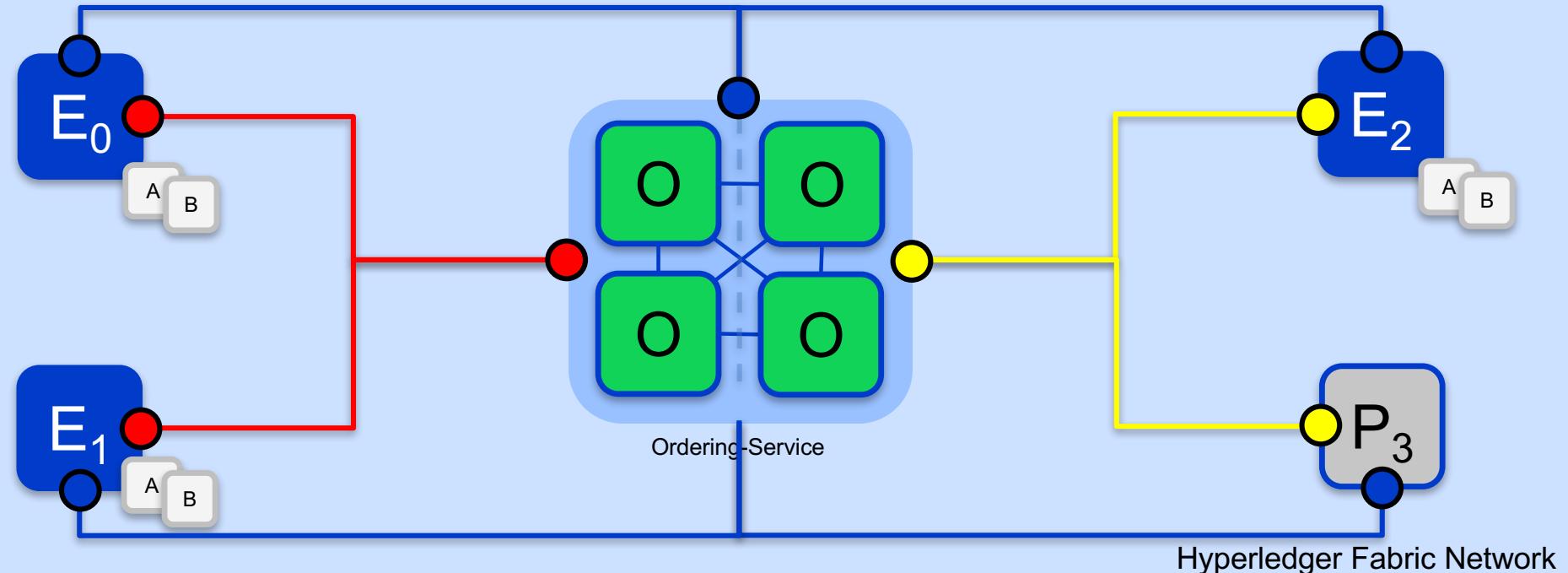
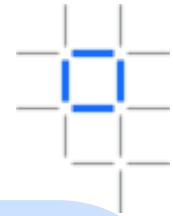
Bootstrap Network (4/6) – Create Channels



Channels are created on the ordering service:

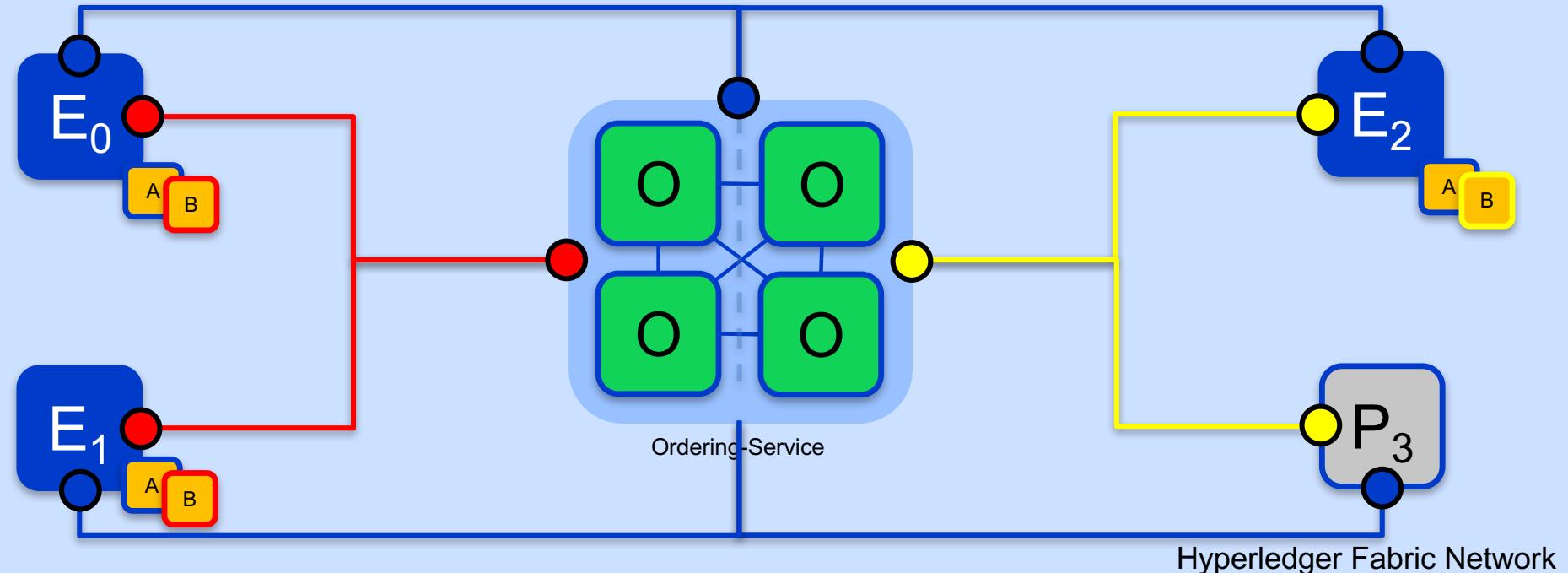
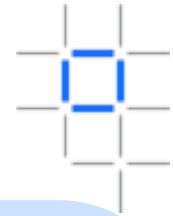
\$ peer channel create -o [orderer] ...

Bootstrap Network (5/6) – Join Channels



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

Bootstrap Network (6/6) – Instantiate Chaincode



Peers finally instantiate the Chaincode on the channels they want to transact on:
\$ peer chaincode instantiate ... –P ‘policy’

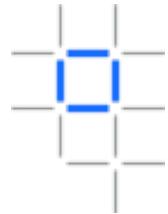


Technical Deep Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- Components
- Network setup
- [Endorsement Policies]
- Membership Services

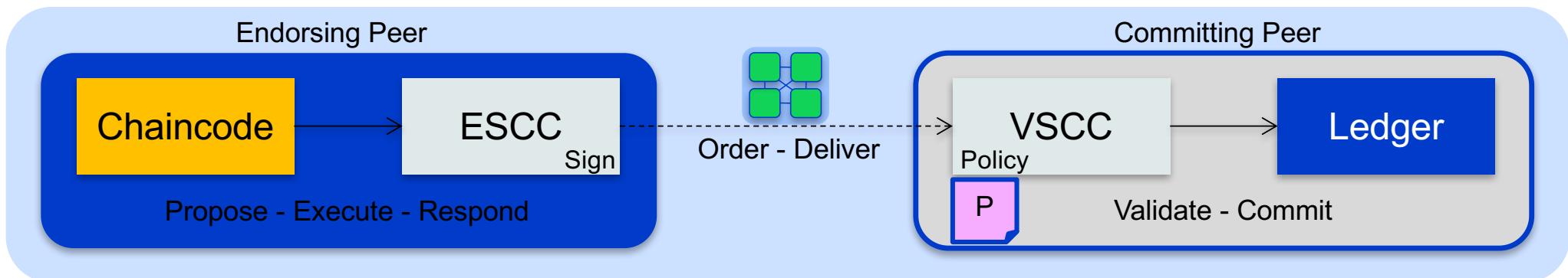


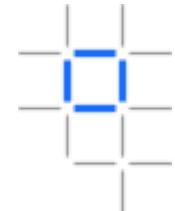
Endorsement Policies



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\[, E...\]\)](#)

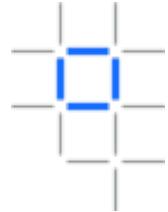
Where [EXPR](#) is either AND or OR and [E](#) is either a principal or nested EXPR

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



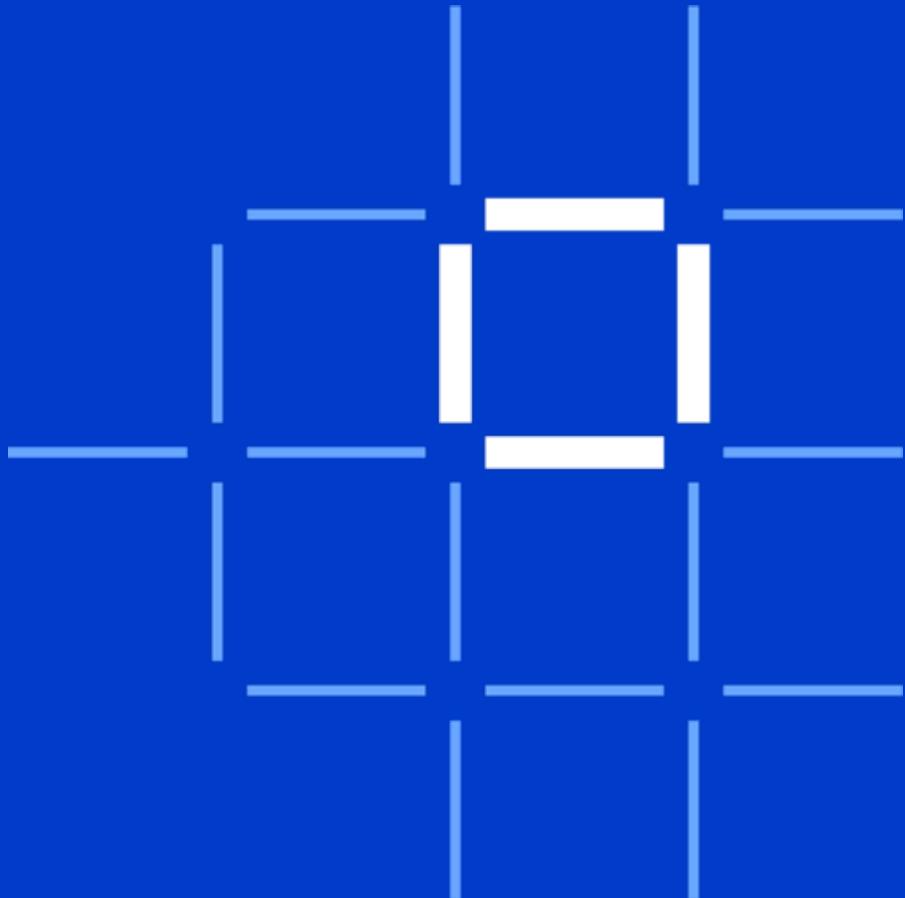
Examples of policies:

- 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'))`

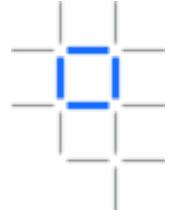


Technical Deep Dive

- Architectural Overview
- Network Consensus
- Channels and Ordering Service
- Components
- Network setup
- Endorsement Policies
- [Membership Services]

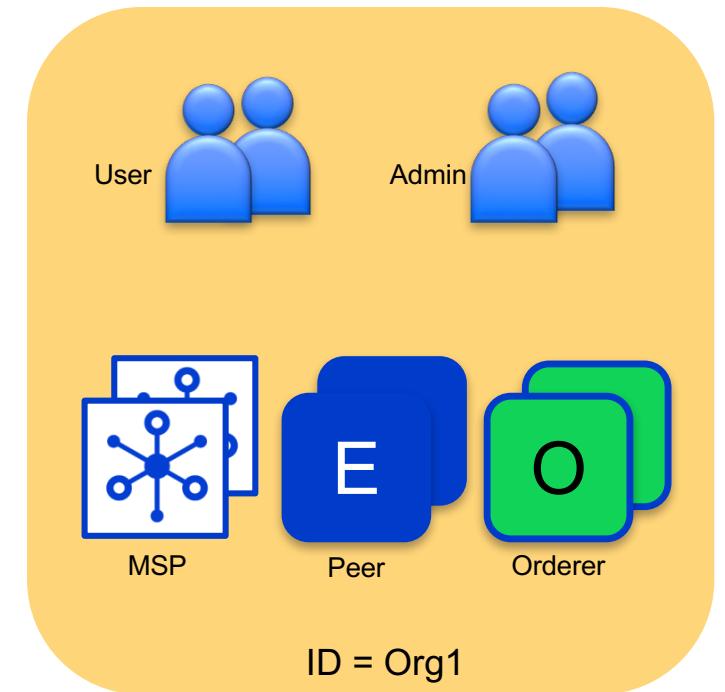


Organisations

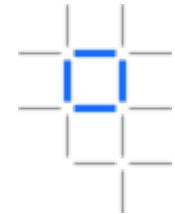


Organisations define boundaries within a Fabric Blockchain Network

- 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

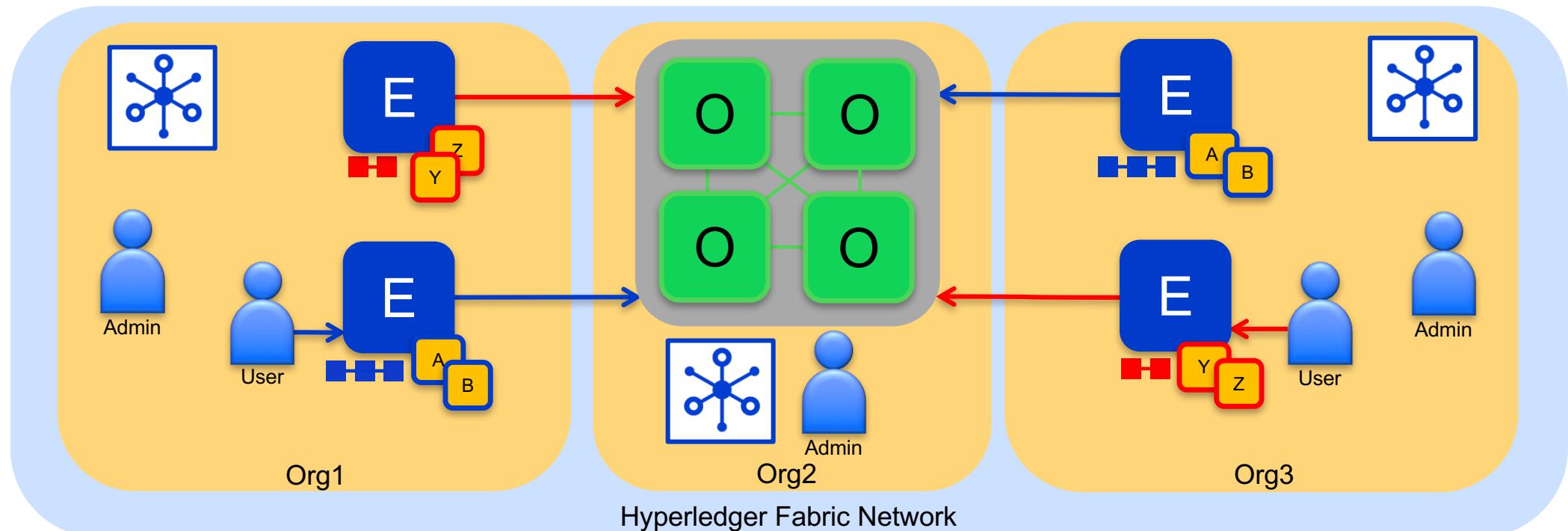


Consortium Network

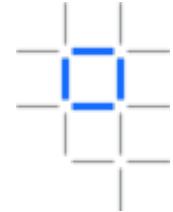


An example consortium network of 3 organisations

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

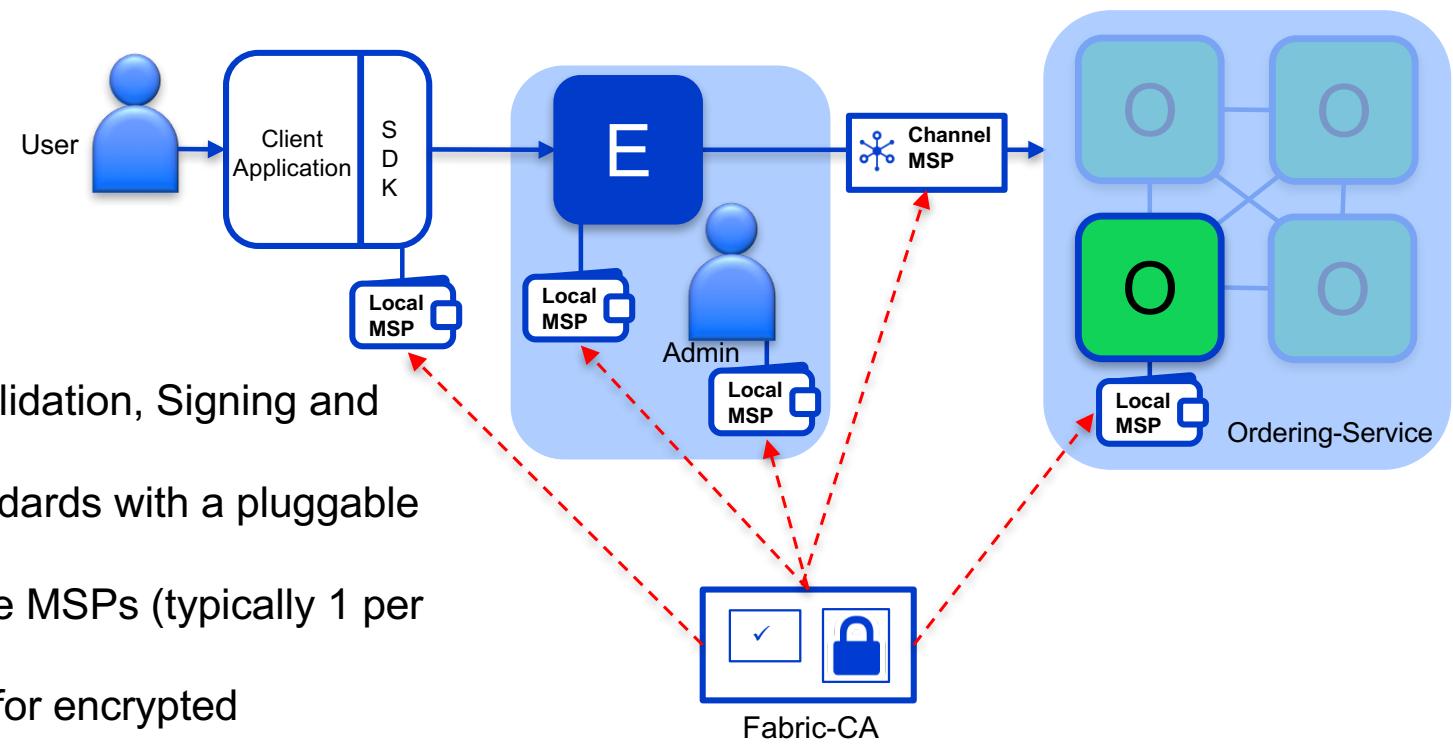


Membership Services Provider - Overview

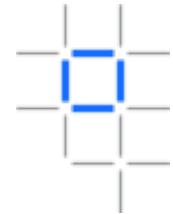


A MSP manages a set of identities within a distributed Fabric network

- 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

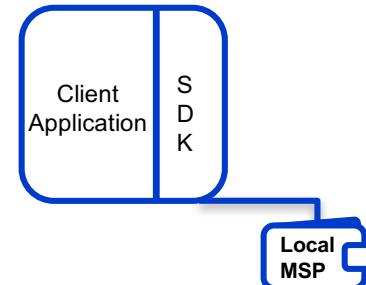


User Identities

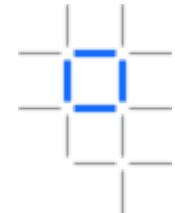


Each client application has a local MSP to store user identities

- Each local MSP includes:
 - **Keystore**
 - **Private key** for signing transactions
 - **Signcert**
 - **Public x.509 certificate**
- May also include TLS credentials
- Can be backed by a Hardware Security Module (HSM)



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



Admin Identities

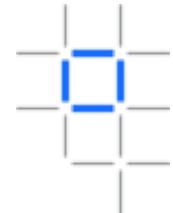
Each Administrator has a local MSP to store their identity

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

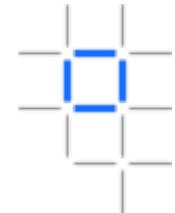


Each peer and orderer has a local MSP

- 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)



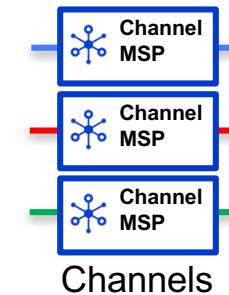
peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>



Channel MSP information

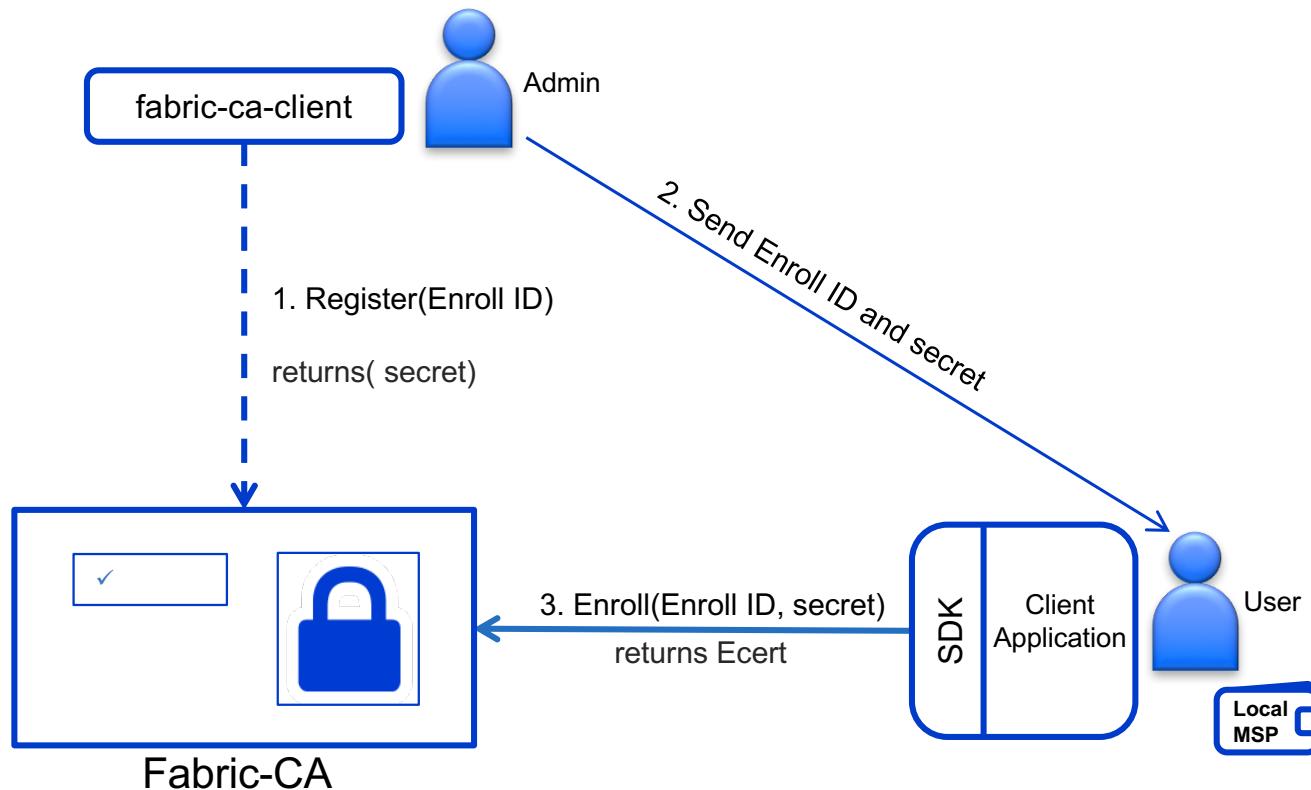
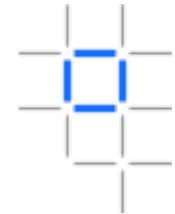
Channels include additional organisational MSP information

- 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



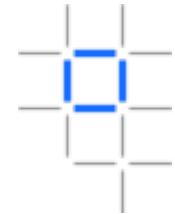
ID = MSP1	
admincerts	admin.org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
crls	<list of revoked admin certificates>

New User Registration and Enrollment



Registration and Enrollment

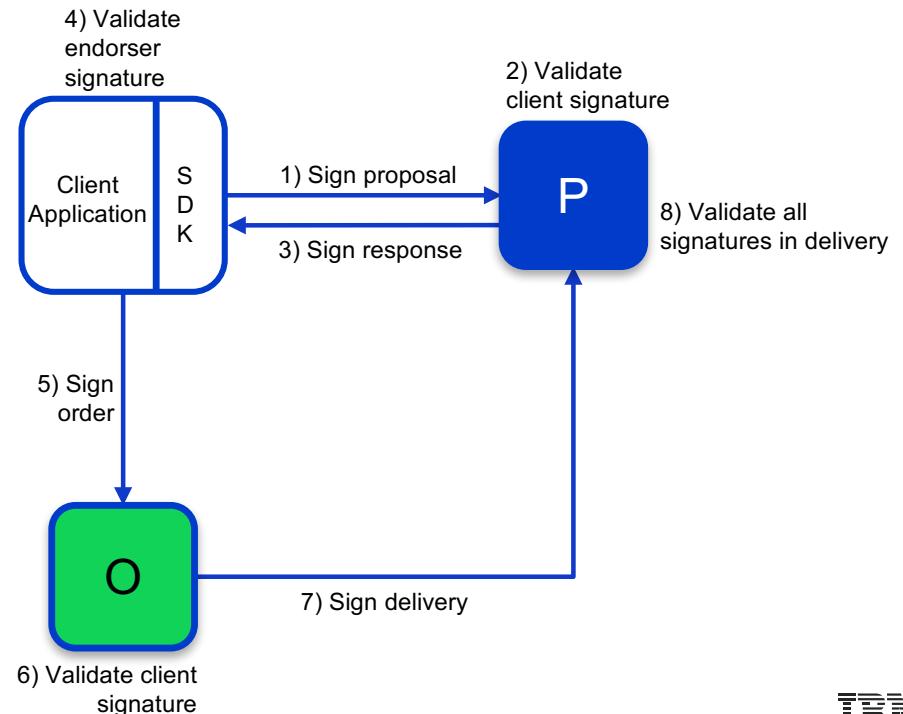
- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- Additional offline registration and enrollment options available



Transaction Signing

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



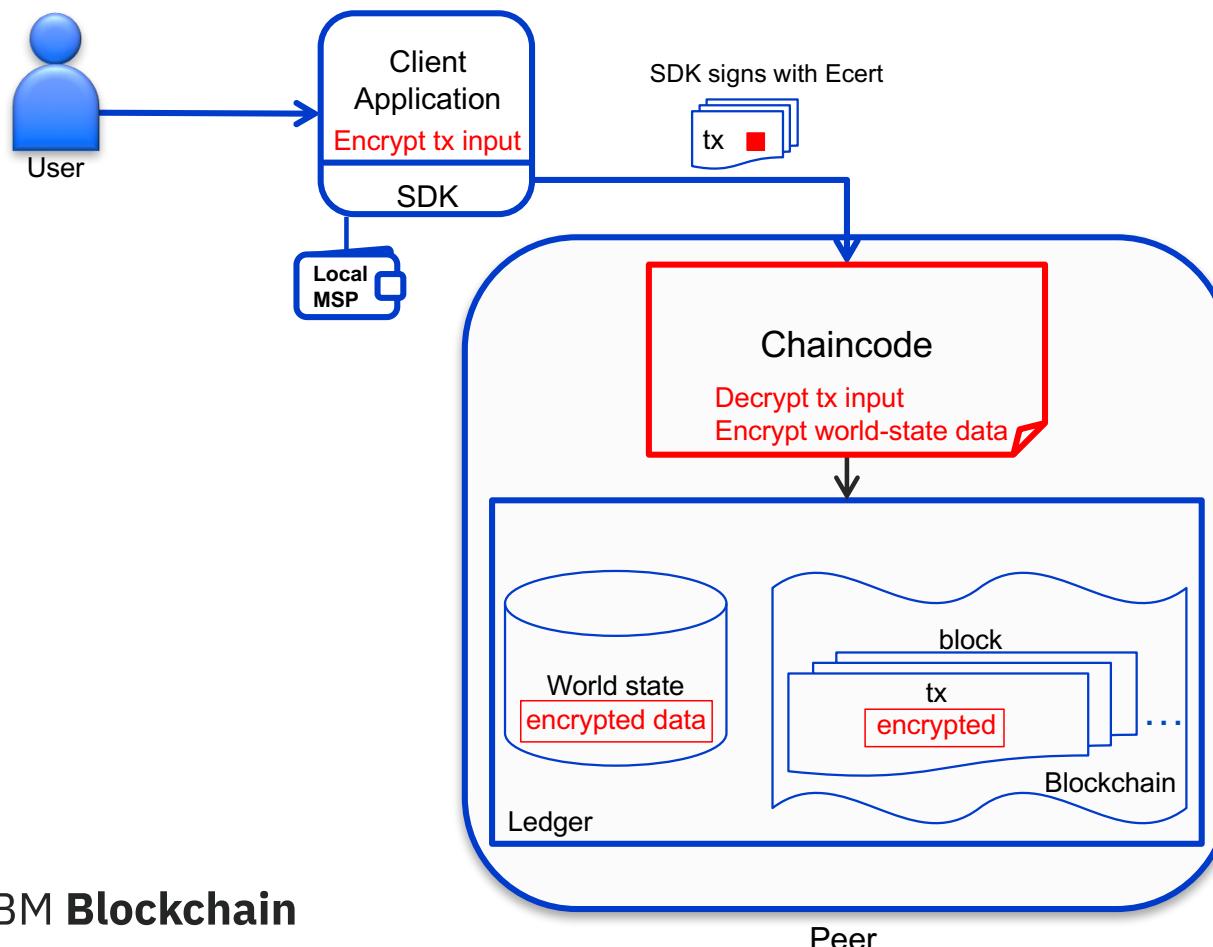
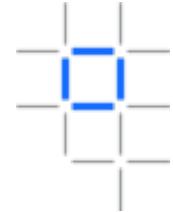


Futures

- [Encryption]
- Collections (SideDB)



Application Level Encryption (FAB-830)



Data Encryption

Handled in the application domain.

Multiple options for encrypting:

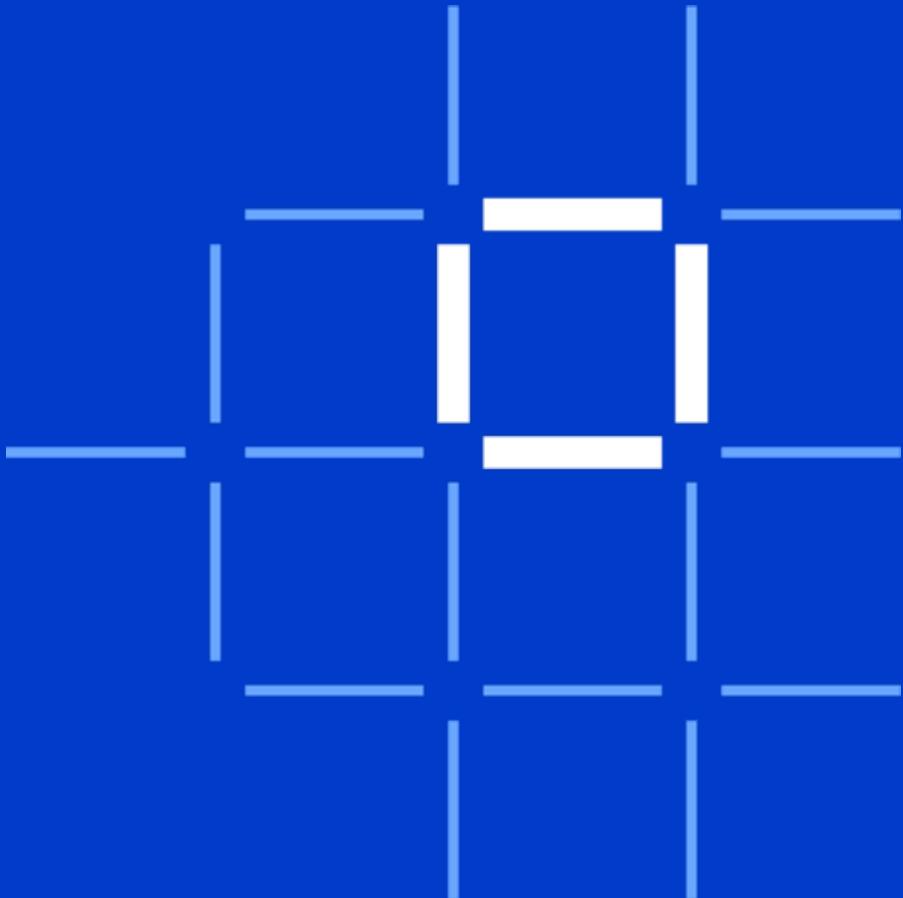
- Transaction Data
- Chaincode
- World-State data

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).

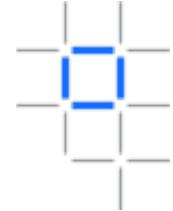


Futures

- Encryption
- [Collections (SideDB)]



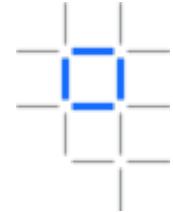
Private Data Collections (FAB-1151)



Allows data to be private to only a set of authorized peers

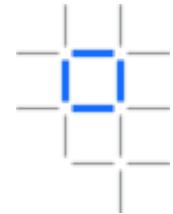
Fabric 1.0	Fabric 1.1
• Data privacy across channels only	→ Data privacy within a channel
• Transaction proposal and worldstate read/write sets visible to all peers connected to a channel	→ Transaction proposal and worldstate read/write sets available to only permissioned peers
• Ordering service has access to transactions including the read/write sets	→ Ordering service has only evidence of transactions (hashes) <ul style="list-style-type: none">• Complements Fabric 1.0 channel architecture• Policy defines which peers have private data

Private Data Collections - Explained



1. Private data:
 1. Excluded from transactions by being sent as ‘transient data’ to endorsing peers.
 2. Shared peer-to-peer with only peers defined in the collection policy.
2. Hashes of private data included in transaction proposal for evidence and validation.
 1. Peers/Orderers not in the collection policy have only hashes.
3. Peers maintain both a public worldstate and private worldstate.
4. Private data held in a transient store between endorsement and validation.

Private Data Collections – Marble Scenario



Privacy Requirements:

- No marble data should go through ordering service as part of a transaction
- All peers have access to general marble information
 - *Name, Size, Color, Owner*
- Only a subset of peers have access to marble *pricing* information

Transaction

- Primary read/write set (if exists)
- Hashed private read/write set (hashed keys/values)

Transaction

- Public channel data
- Goes to all orderers/peers

Collection: Marbles

- Private Write Set
 - Name, Size, Color, Owner
- Policy: Org1, Org2**
"requiredPeerCount": 1,
"maxPeerCount": 2,
"blockToLive": 1000000

Collection: Marbles

- Private data for channel peers
- Goes to all peers but not orderers

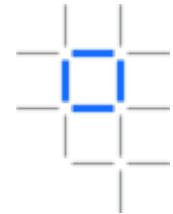
Collection: Marble Private Details

- Private Write Set
 - Price
- Policy: Org1**
"requiredPeerCount": 1,
"maxPeerCount": 1,
"blockToLive": 3

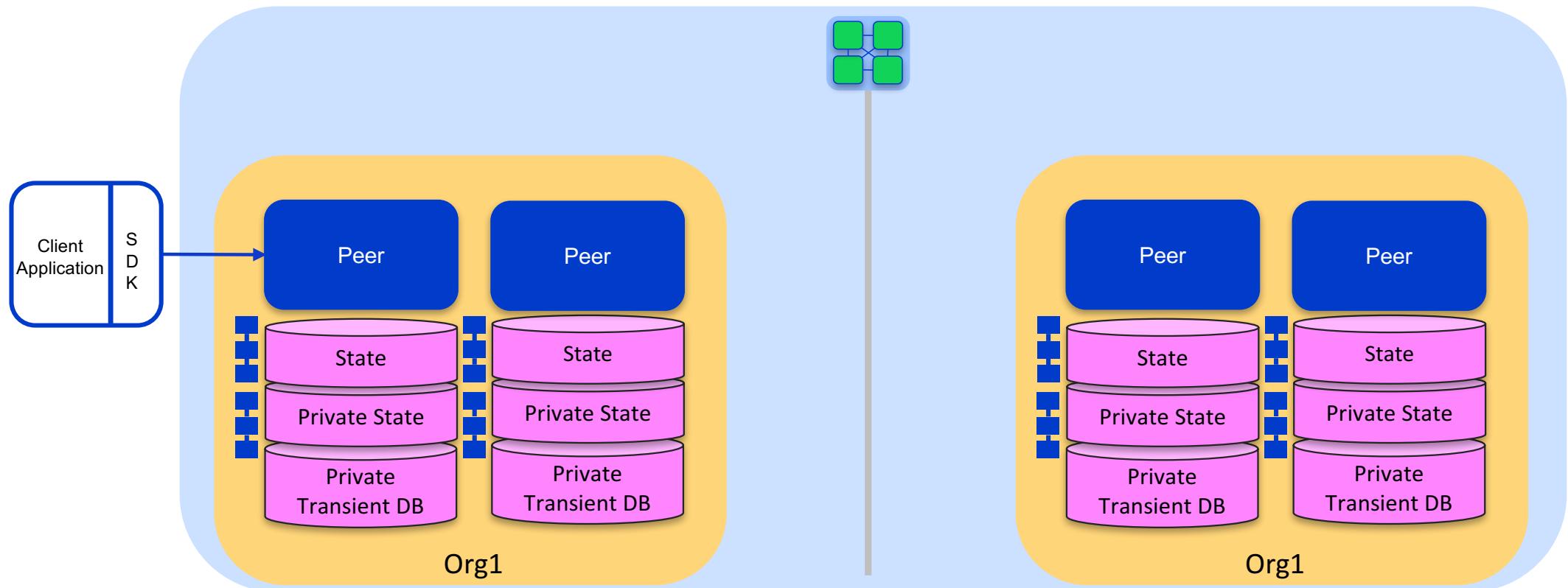
Collection: Marbles Private Details

- Private data for subset of channel peers
- Goes to subset of peers only

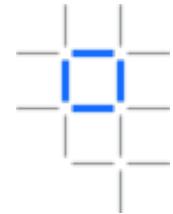
Step 1: Propose Transaction



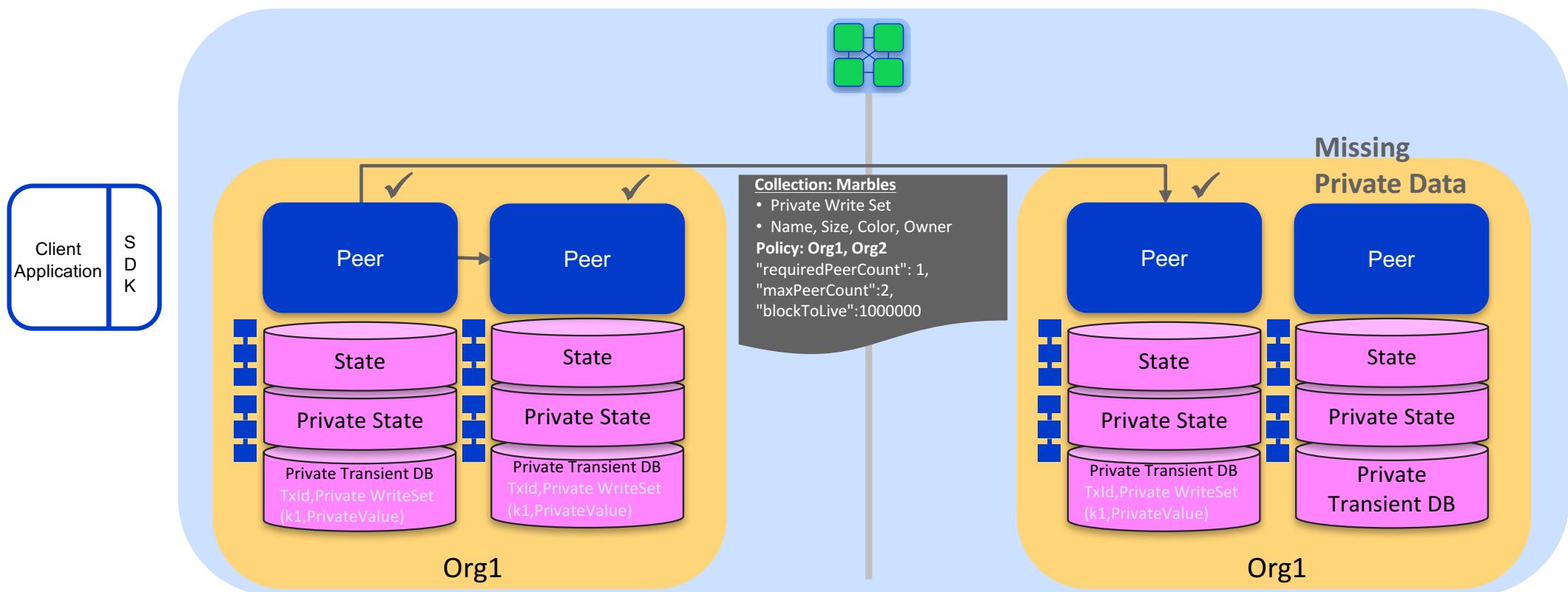
Client sends proposal to endorsing peer(s)



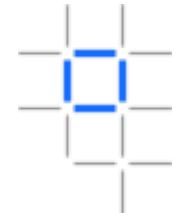
Step 2a: Execute Proposal and Distribute 1st Collection



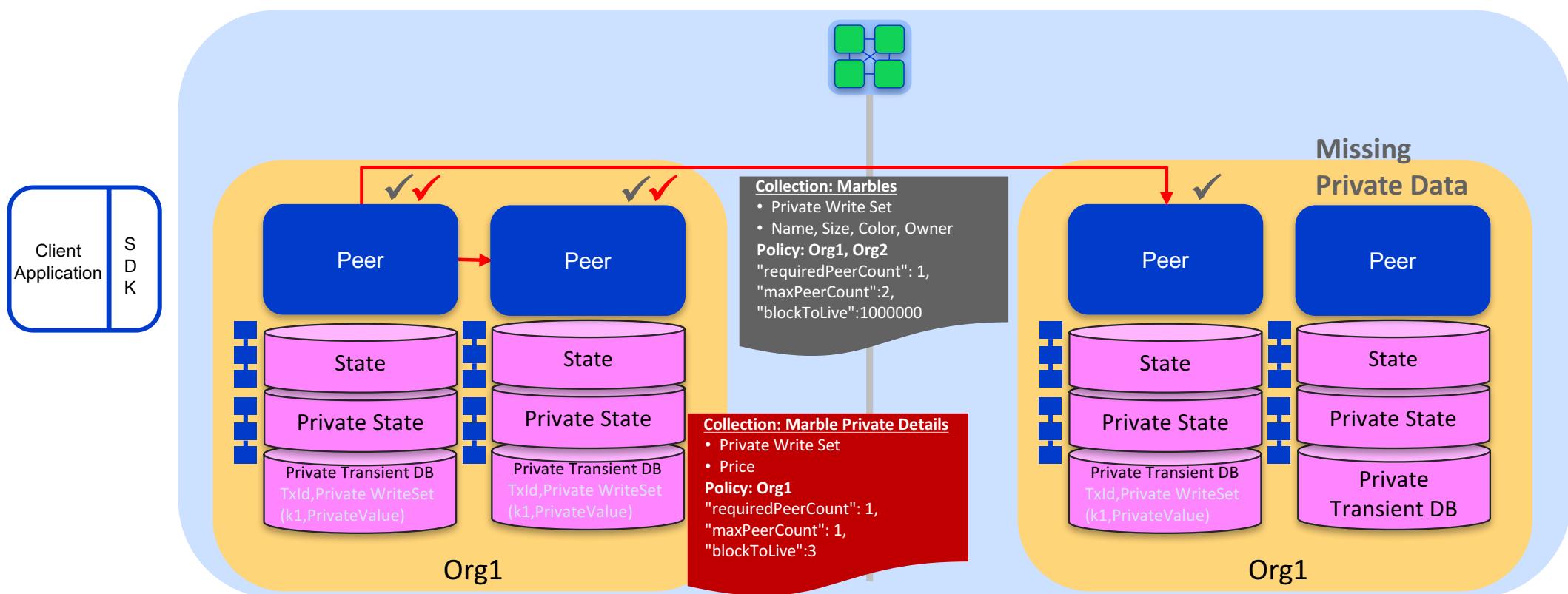
Endorsing peer simulates transaction and distributes **marbles collection** data based on policy



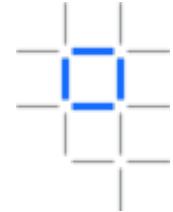
Step 2b: Distribute 2nd Collection



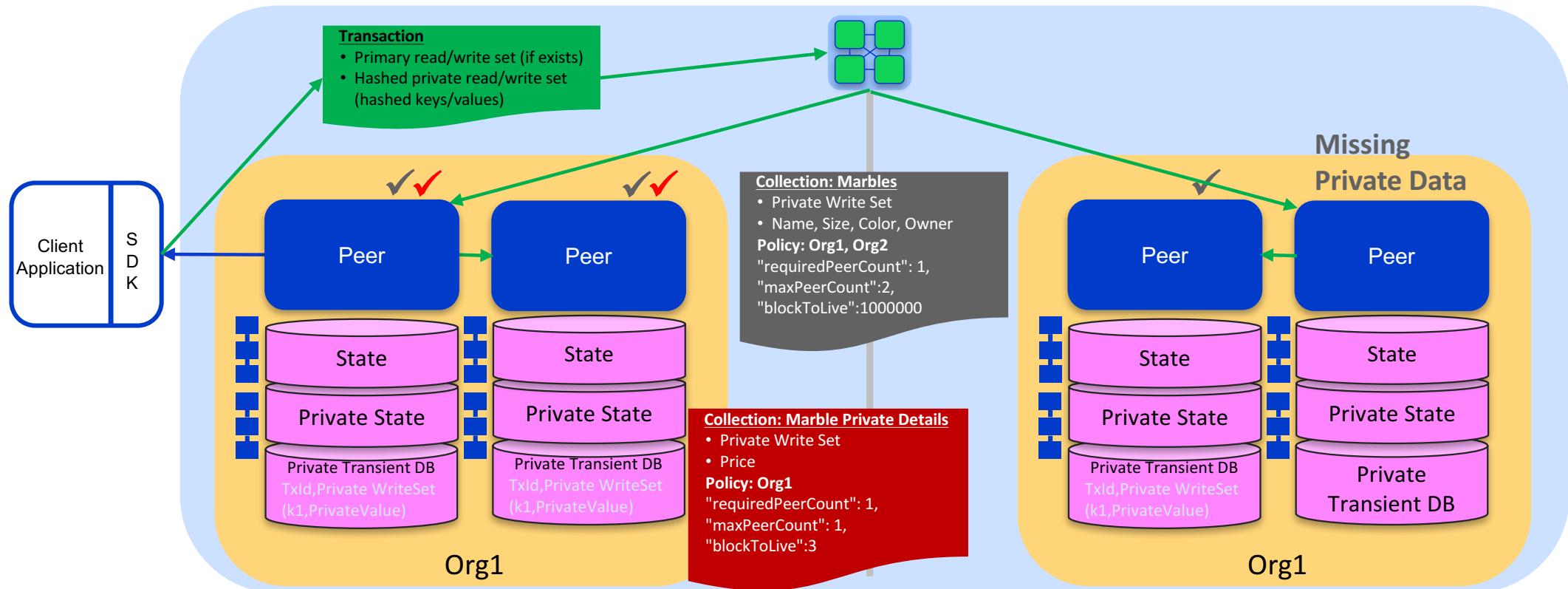
Endorsing peer distributes **marbles private details collection** data based on policy



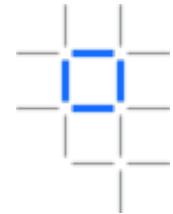
Step 3: Proposal Response / Order / Deliver



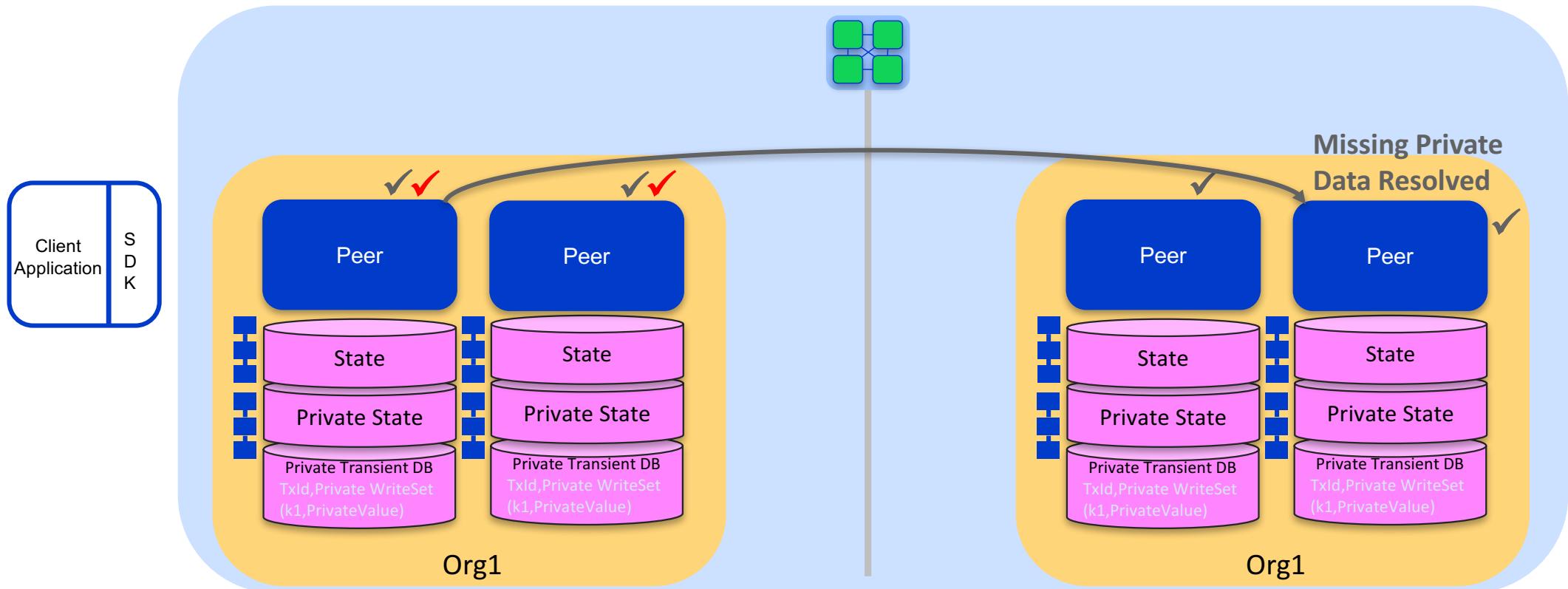
Proposal response sent back to client, which then sends the proposal to the ordering service for delivery to all peers



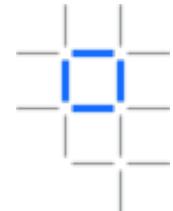
Step 4: Validate Transaction



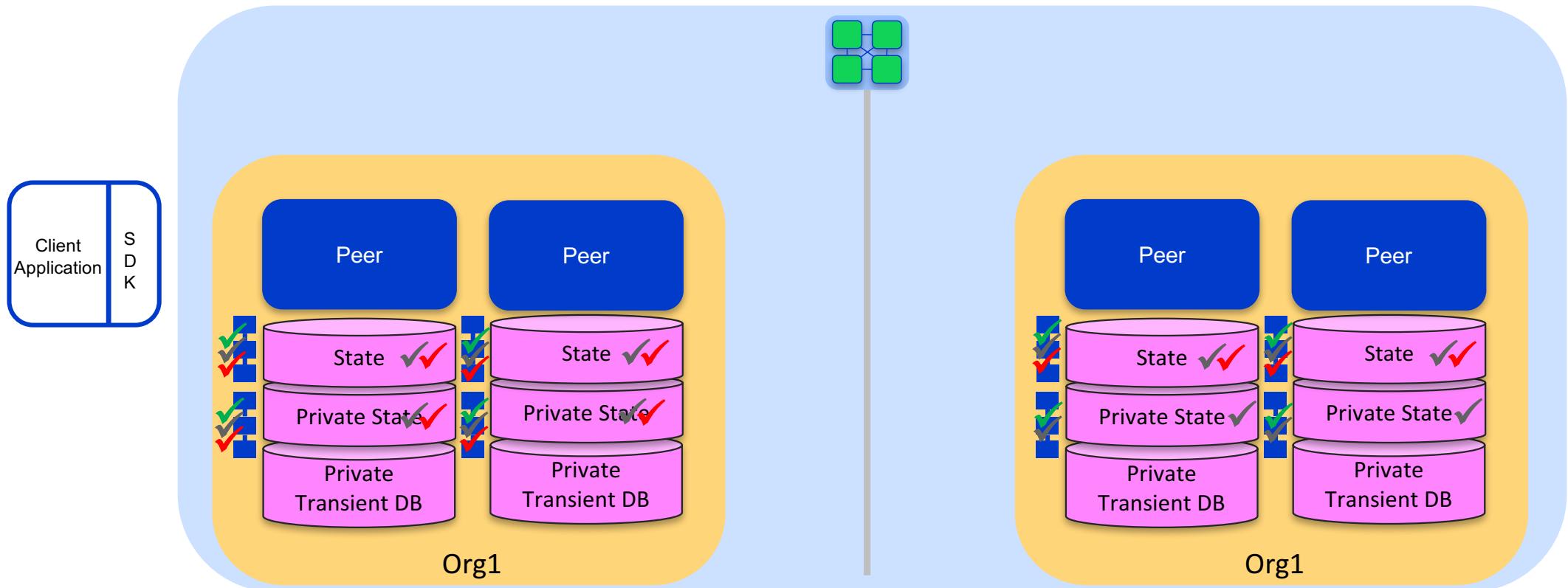
Peers validate transactions. Private data validated against hashes. Missing private data resolved with pull requests from other peers.



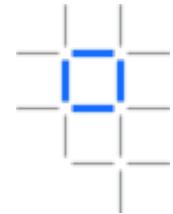
Step 5: Commit



- 1) Commit private data to private state db. 2) Commit hashes to public state db.
- 3) Commit public block and private write set storage. 4) Delete transient data.

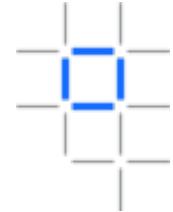


Summary and Next Steps



- 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

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>

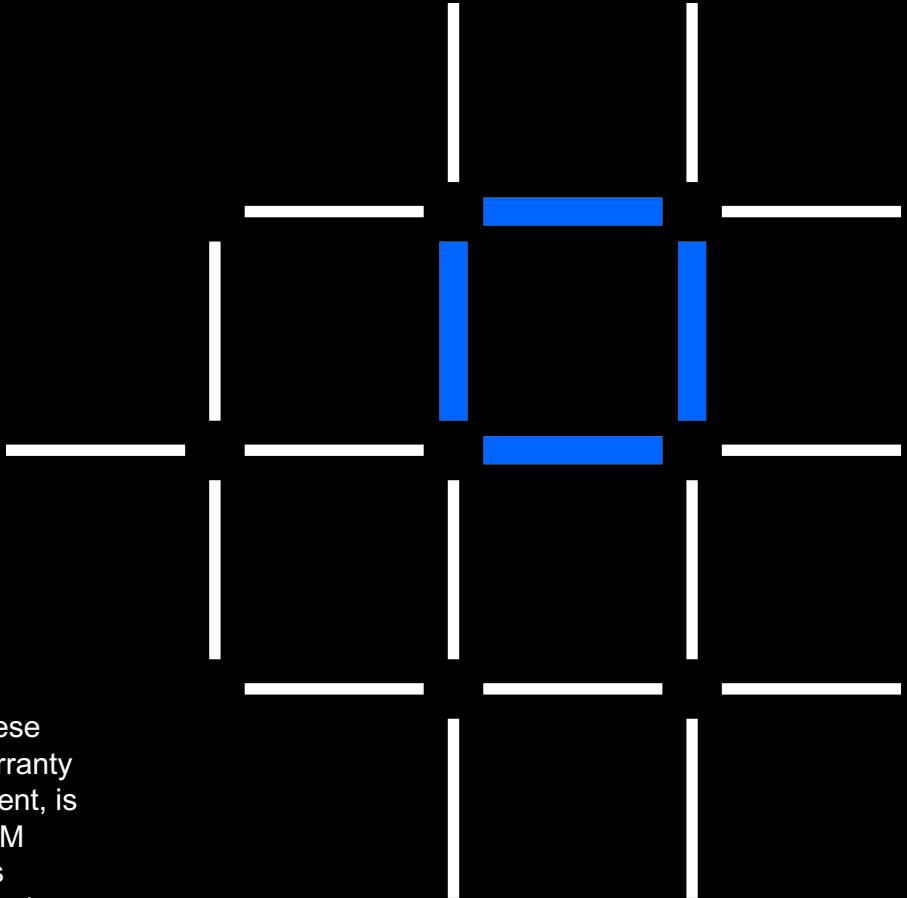
Thank you

IBM Blockchain

www.ibm.com/blockchain

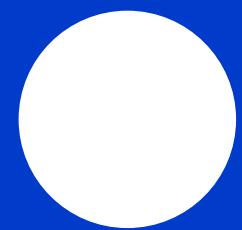
developer.ibm.com/blockchain

www.hyperledger.org

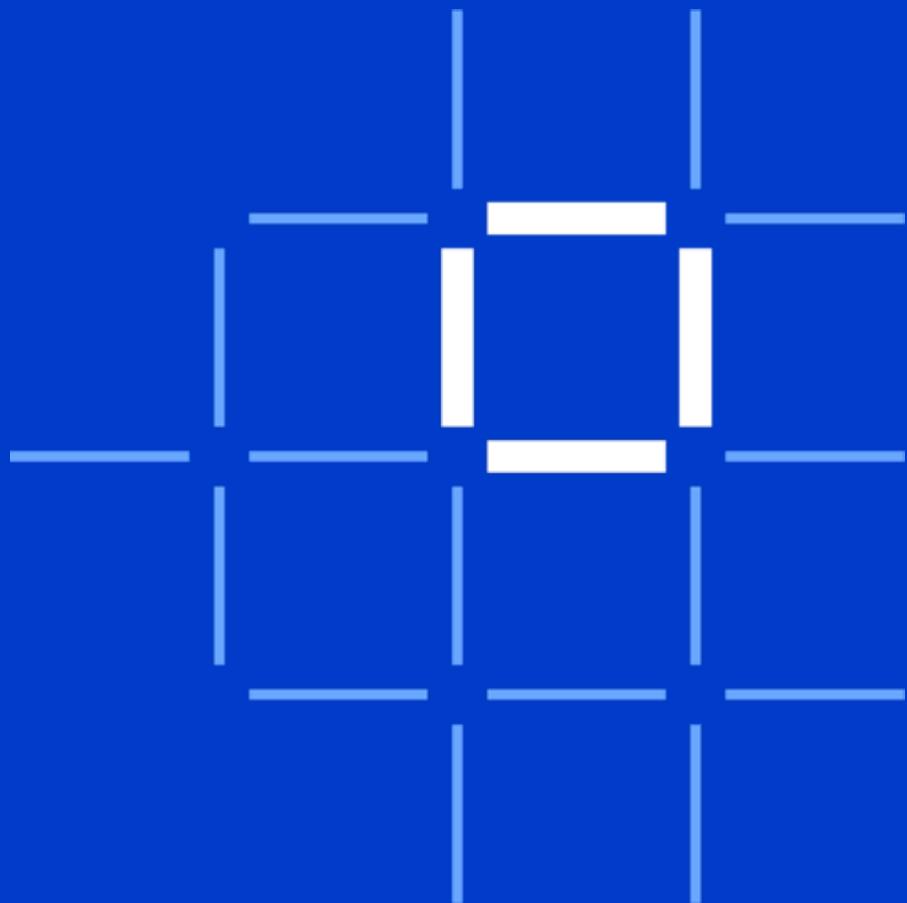


© Copyright IBM Corporation 2017. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or 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.

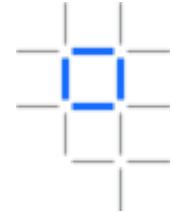
IBM



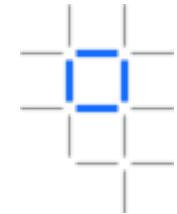
Appendix



Fabric Commands



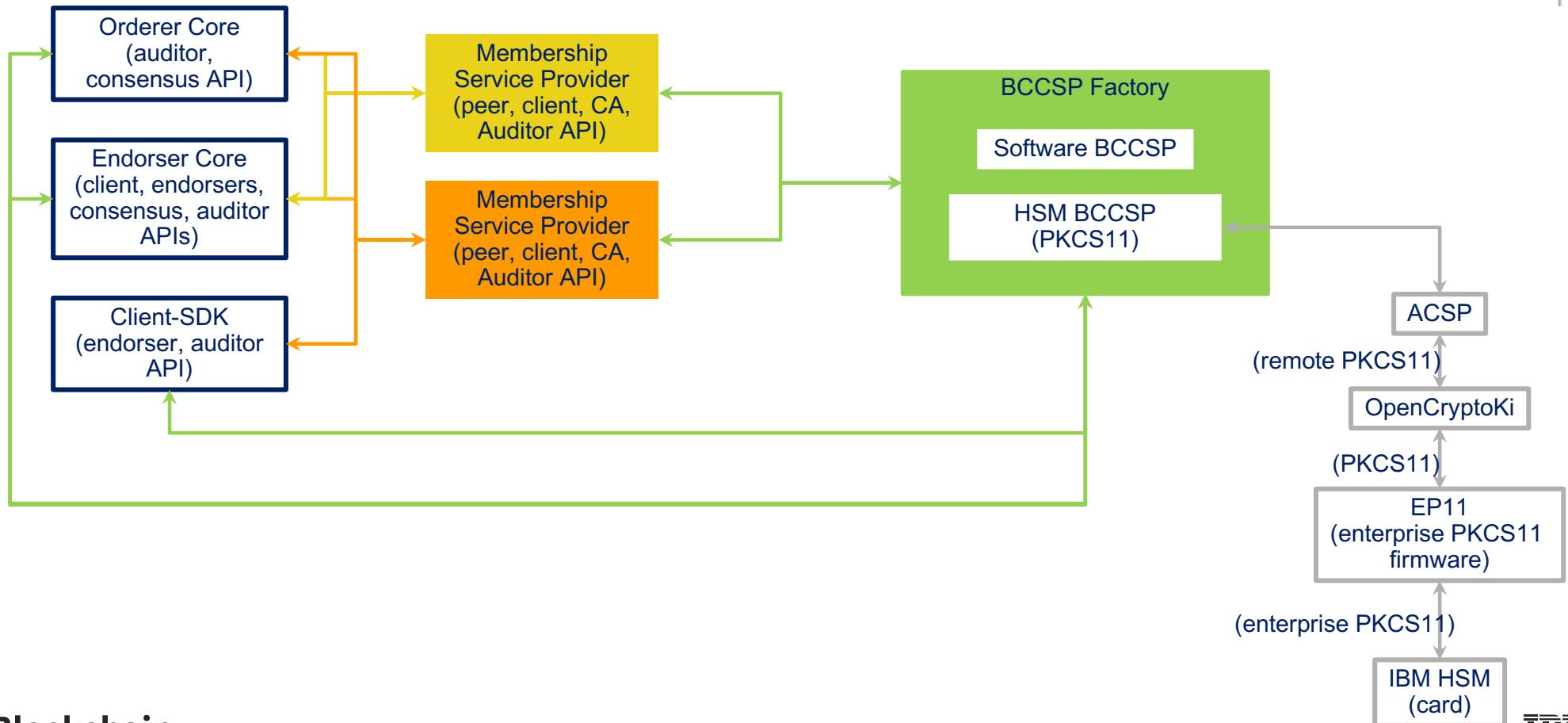
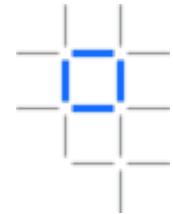
- Fabric has the following commands:
 - **peer** ... (For operating and configuring a peer)
 - **peer chaincode** ... (Manages chaincode on the peer)
 - **peer channel** ... (Manages channels on the peer)
 - **peer node** ... (Manages the peer)
 - **peer version** (Returns the peer version)
 - **cryptogen** ... (Utility for generating crypto material)
 - **configtxgen** ... (Creates configuration data such as the genesis block)
 - **configtxlator** ... (Utility for generating channel configurations)
 - **fabric-ca-client** ... (Manages identities)
 - **fabric-ca-server** ... (Manages the Fabric-CA server)



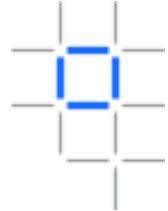
Configuration Detail

Path	MSP ID	Attributes	Attributes
config -> channel_group -> groups -> application -> groups	Org1MSP	mod_policy	Admins
		policies -> Admins	mod_policy
			Admins
			policy -> value -> identities
			Org1MSP, Admin
			Policy -> value -> rule
			n_out_of
		Policies -> Readers	Mod_policy
			Admins
			Policy -> value -> identities
			Org1MSP
			Policy -> value -> rule
			N_out_of
		Policies -> Writers	Mod_policy
			Admins
			Policy -> value -> identities
			Org1MSP
			Policy -> value -> rule
			N_out_of

MSP and BCCSP (Modularity and Decentralisation)



Blockchain Crypto Service Provider (BCCSP)



- Pluggable implementation of cryptographic standards and algorithms.
- **Pluggability**
 - alternate implementations of crypto interface can be used within the Hyperledger 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

