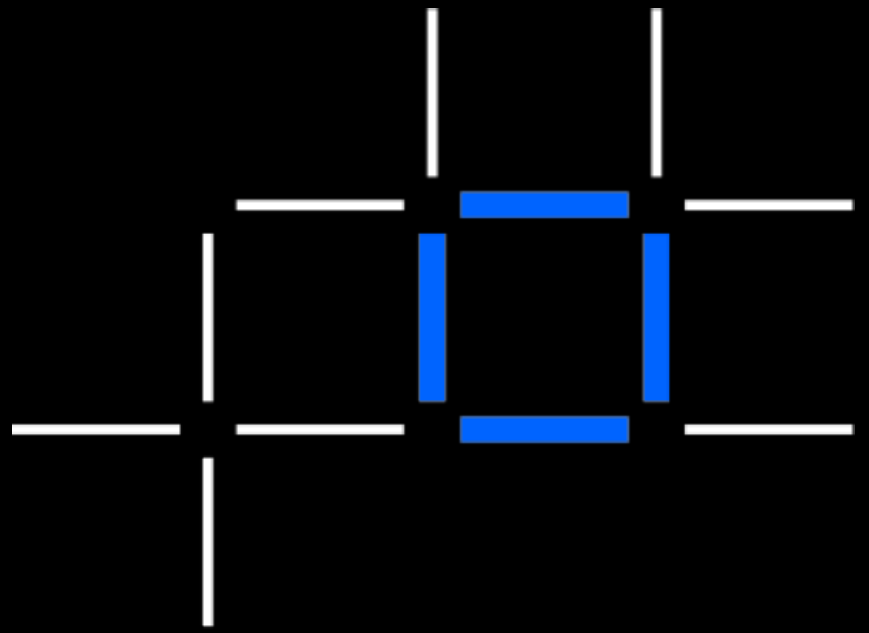
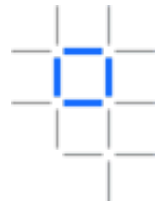
Blockchain Deep Dive

Hyperledger Fabric ”under the hood”



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

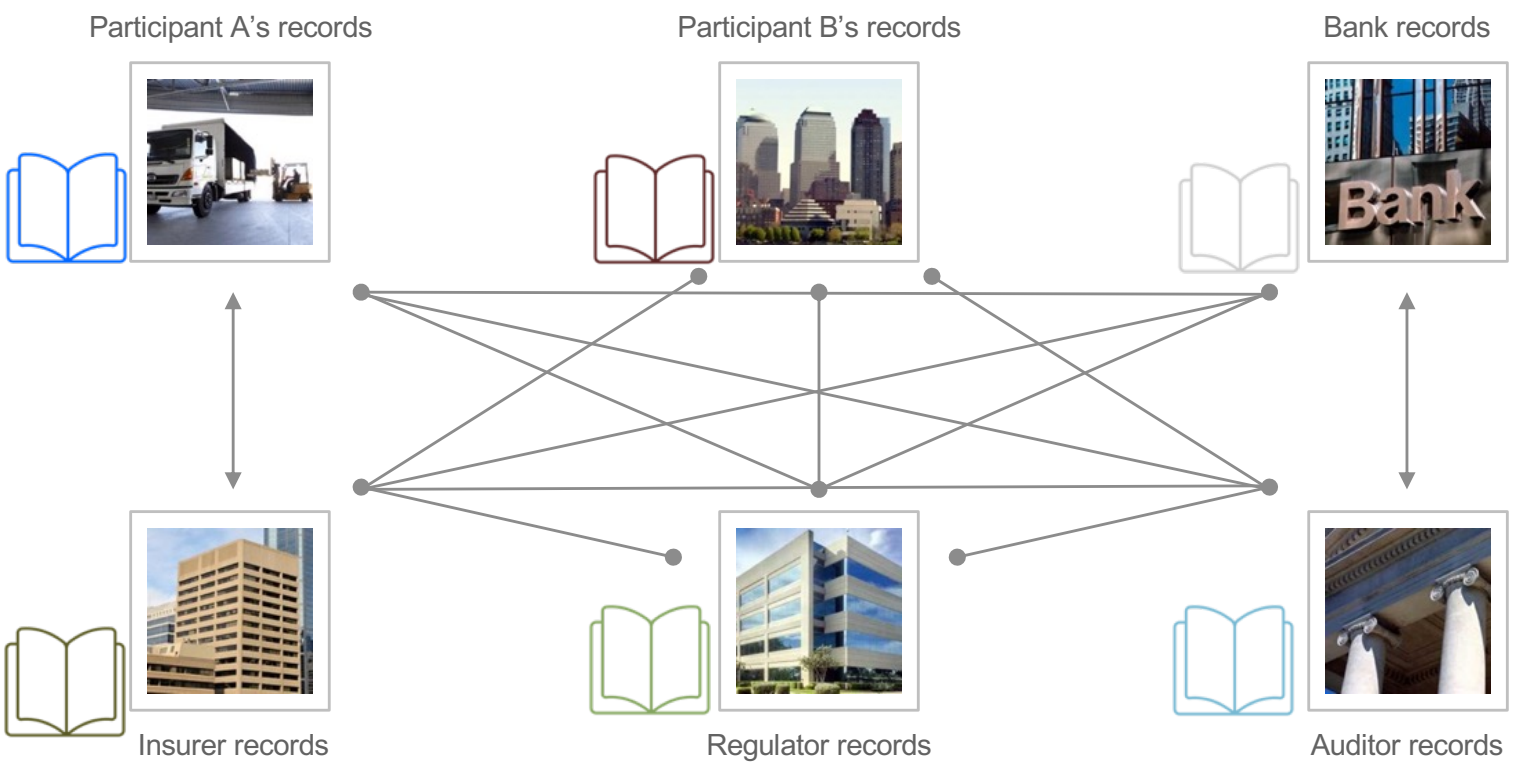
Ledgers, Transactions and Contracts



- Ledger: an important **log** of all transactions
  - Describes the inputs and outputs of the business
- Transaction: an **asset transfer** between participants
  - Matt gives a car to Dave (simple)
- Contract: the **conditions** for a transaction to occur
  - If Dave pays Matt money, then car passes from Matt to Dave (simple)
  - If car won't start, funds do not pass to Matt (as decided by third party arbitrator) (more complex)

Problem

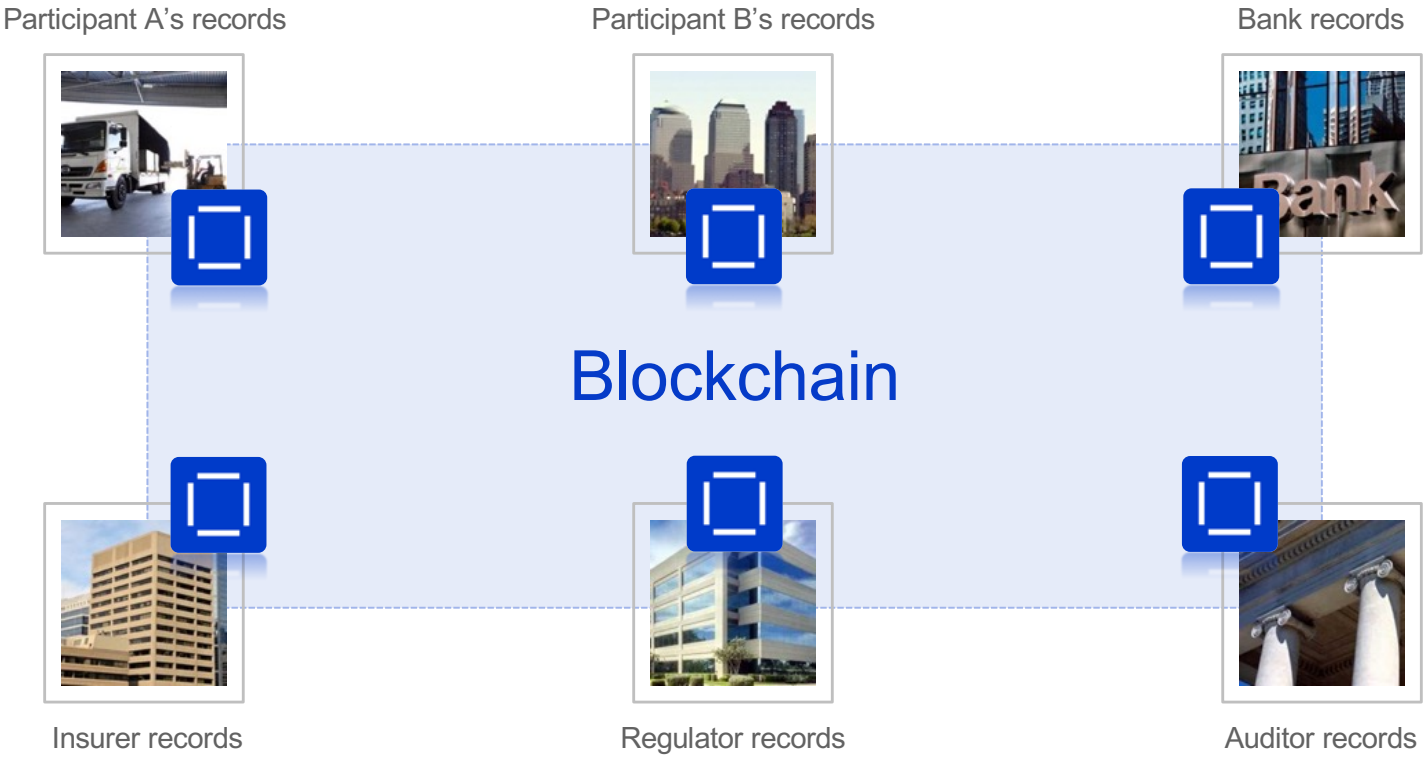
inefficient, expensive, vulnerable



Solution

A shared, replicated, permissioned ledger…

…with consensus, provenance, immutability and finality



Different types of blockchain

- All blockchains aim to provide irrefutable proof that a set of transactions occurred between participants

- Different types of blockchain exist:

  bitcoin is an example of an unpermissioned, public blockchain
  - The first blockchain application
  - Defines a shadow-currency and its ledger
  - Resource intensive

- Blockchains for business generally prioritize
  - **Assets** over cryptocurrency; **Identity** over anonymity; **Selective endorsement** over proof of work

Hyperledger and Linux Foundation

- **Hyperledger is a blockchain for business project under the Linux Foundation organization**

  https://www.hyperledger.org

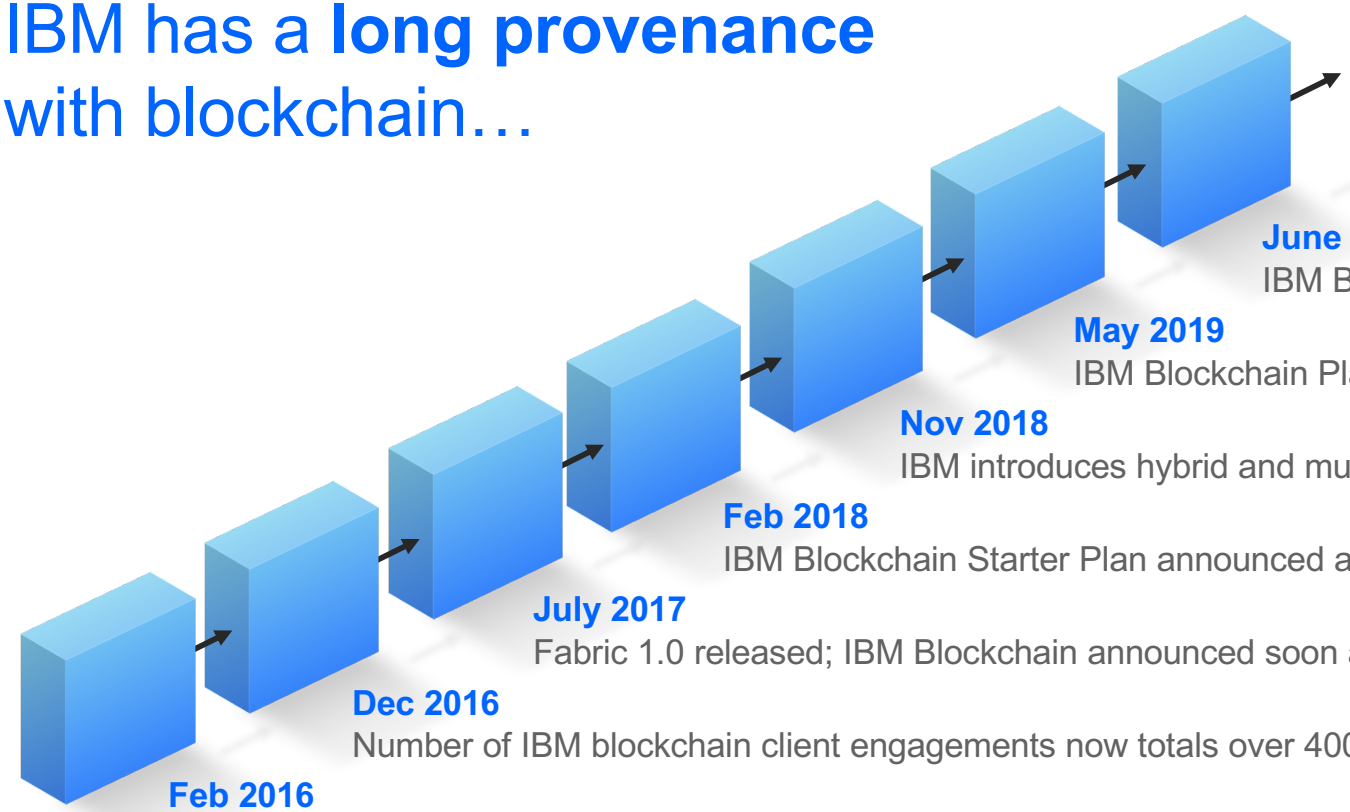
A collaborative effort created to advance cross-industry blockchain technologies for business.

- Open source, open standards and open governance model.
- Currently more than 250 members, IBM is a premier member.
- Premier, General and Associate levels of membership

- **Hyperledger contains distributed ledgers, libraries and tools**
- https://www.hyperledger.org/projects
  - IBM is a major supporter and contributor to the development of the *Hyperledger Fabric* distributed ledger framework
- **Hyperledger Fabric**
  - V1.x available since July 2017
  - Currently over 250 active developers from different organizations.
  - Focus is development of ledger, smart contracts, consensus, privacy, scalability and resilience.

https://www.hyperledger.org/projects/fabric

IBM has a **long provenance** with blockchain…



Aug 2015
IBM starts developing first prototype of blockchain technology (Open Blockchain); first client engagements

Feb 2016
IBM becomes a founding member of Linux Foundation Hyperledger; donates code and intellectual property

Dec 2016
Number of IBM blockchain client engagements now totals over 400

July 2017
Fabric 1.0 released; IBM Blockchain announced soon after

Feb 2018
IBM Blockchain Starter Plan announced at THINK

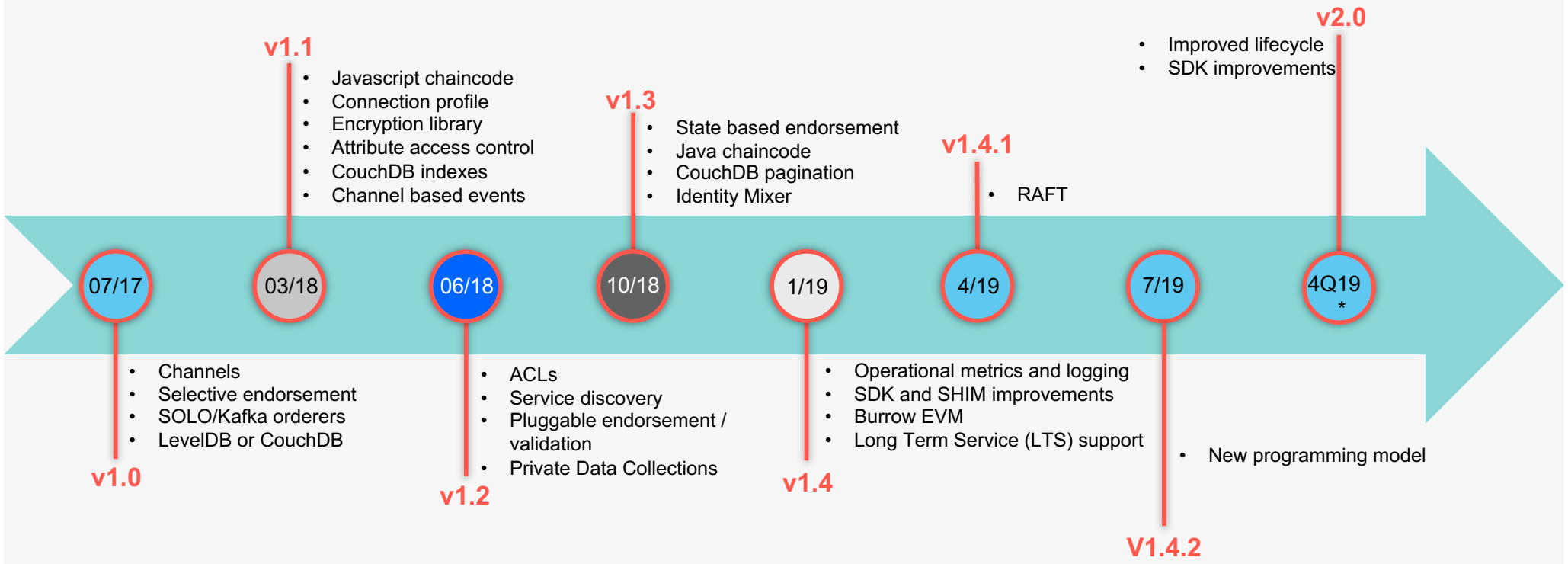
Nov 2018
IBM introduces hybrid and multi-cloud networks

May 2019
IBM Blockchain Platform for IBM Cloud released

June 2019
IBM Blockchain Platform for Multicloud released

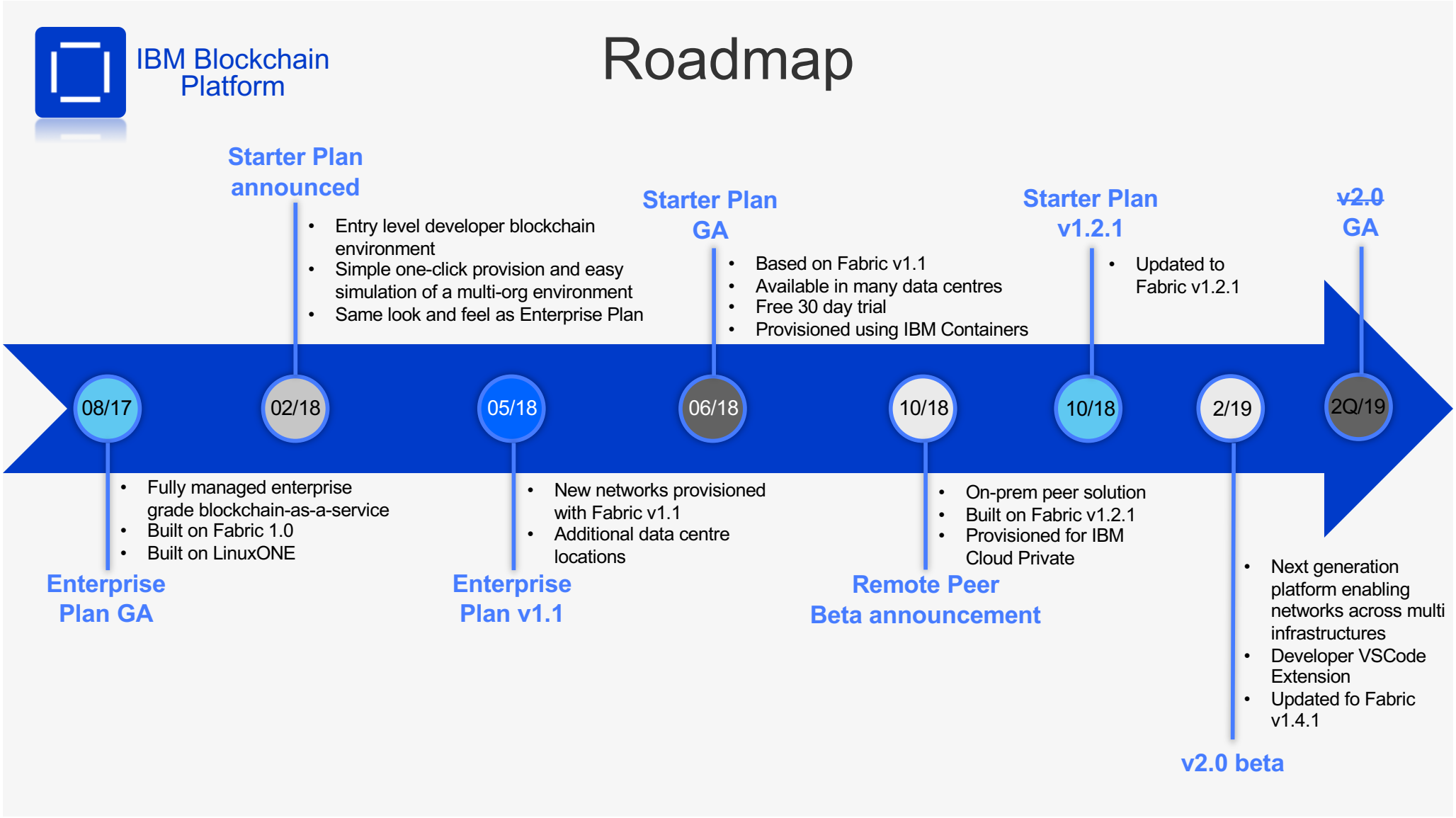
HYPERLEDGER FABRIC

Roadmap



HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

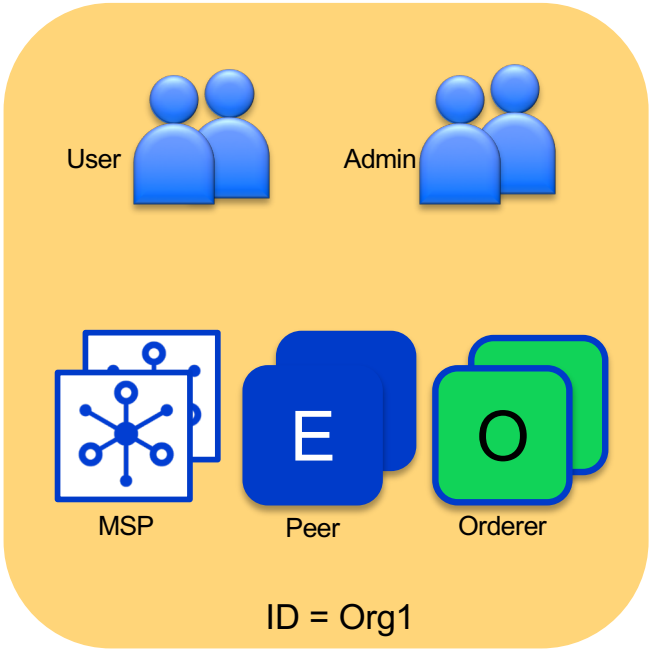
Based on https://wiki.hyperledger.org/projects/fabric/roadmap - Dates determined by the Hyperledger community - (*) Subject to change


IBM Blockchain Platform
Roadmap
Starter Plan announced
Entry level developer blockchain environment
Simple one-click provision and easy simulation of a multi-org environment
Same look and feel as Enterprise Plan
Starter Plan GA
Based on Fabric v1.1
Available in many data centres
Free 30 day trial
Provisioned using IBM Containers
Starter Plan v1.2.1
Updated to Fabric v1.2.1
~~v2.0~~ GA
08/17
02/18
05/18
06/18
10/18
10/18
2/19
2Q/19
Fully managed enterprise grade blockchain-as-a-service
Built on Fabric 1.0
Built on LinuxONE
Enterprise Plan GA
New networks provisioned with Fabric v1.1
Additional data centre locations
Enterprise Plan v1.1
On-prem peer solution
Built on Fabric v1.2.1
Provisioned for IBM Cloud Private
Remote Peer Beta announcement
Next generation platform enabling networks across multi infrastructures
Developer VSCode Extension
Updated fo Fabric v1.4.1
v2.0 beta

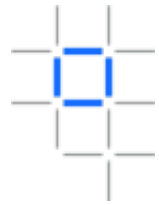
Organizations

Organizations define boundaries within a Fabric Blockchain Network

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

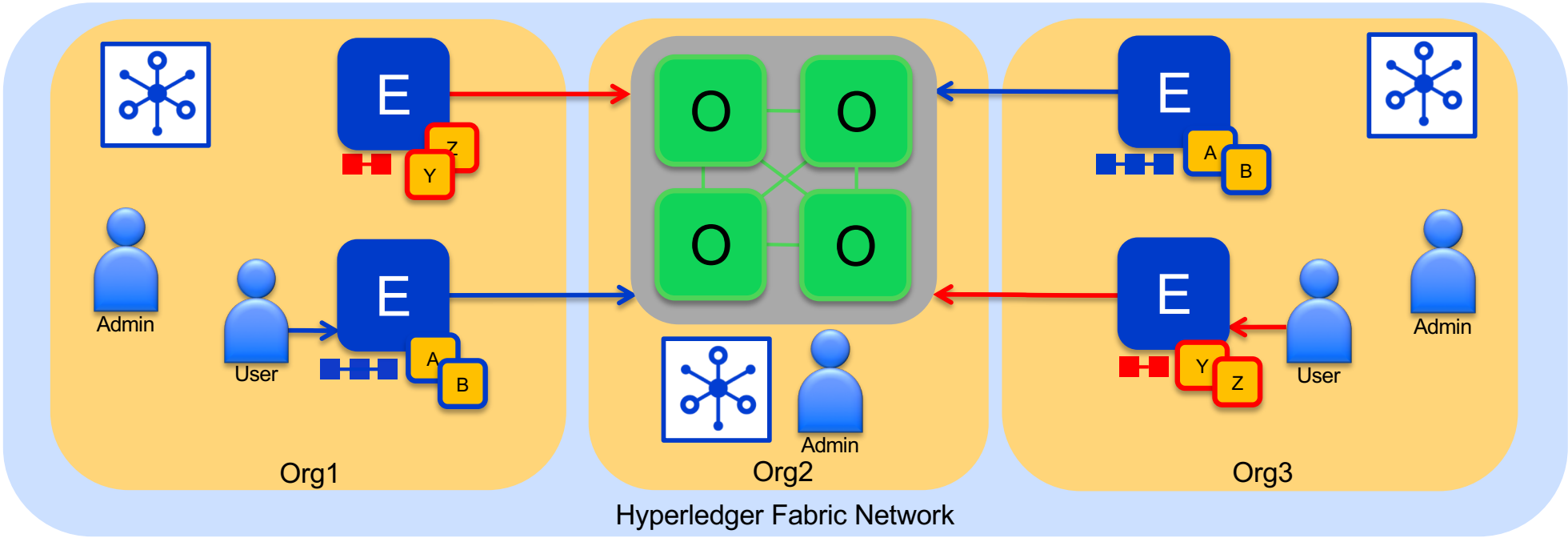


Consortium Network

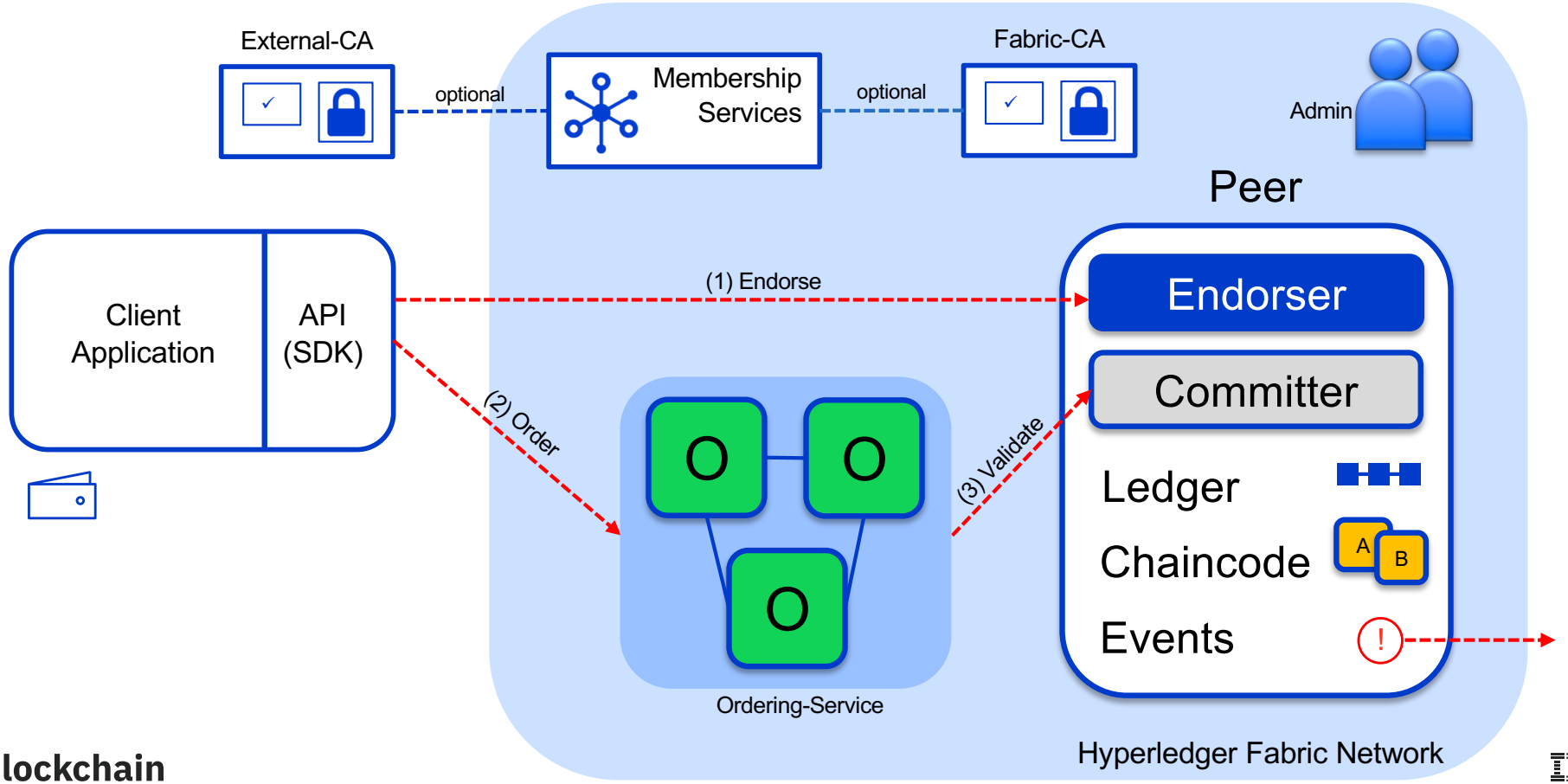


An example consortium network of 3 organizations

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



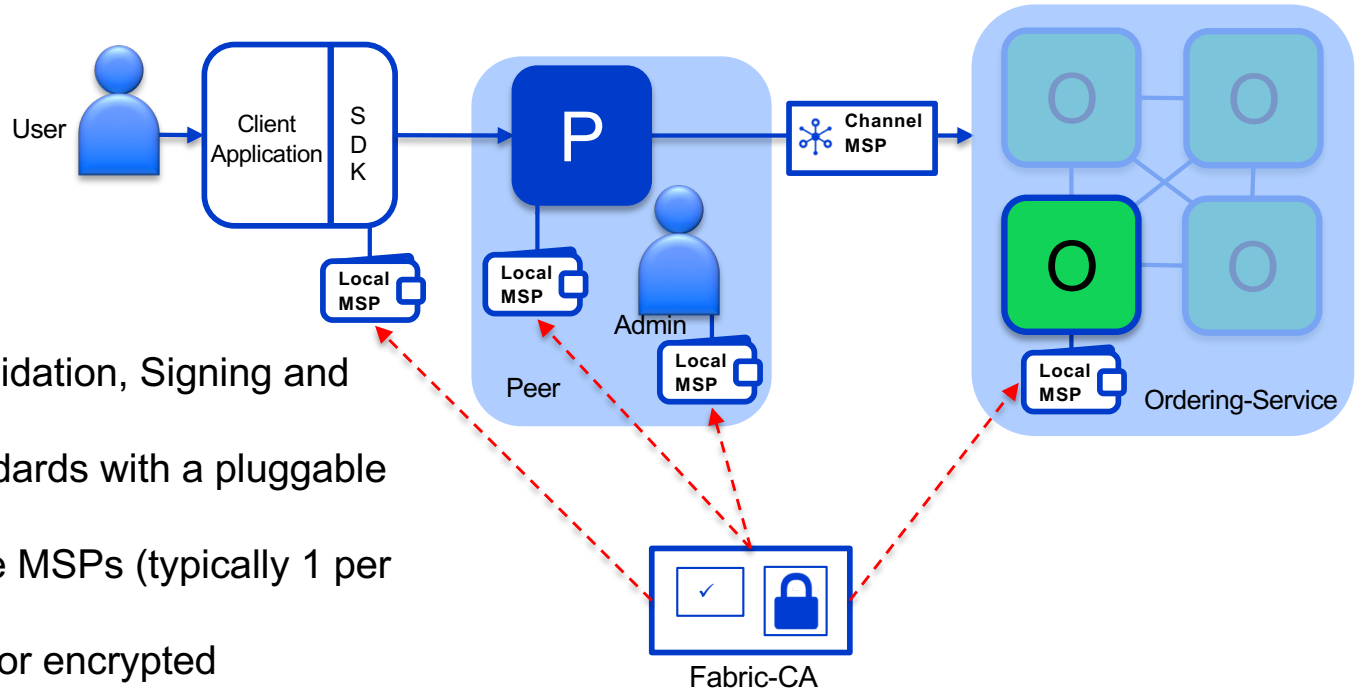
Hyperledger Fabric Architecture



Membership Services Provider - Overview

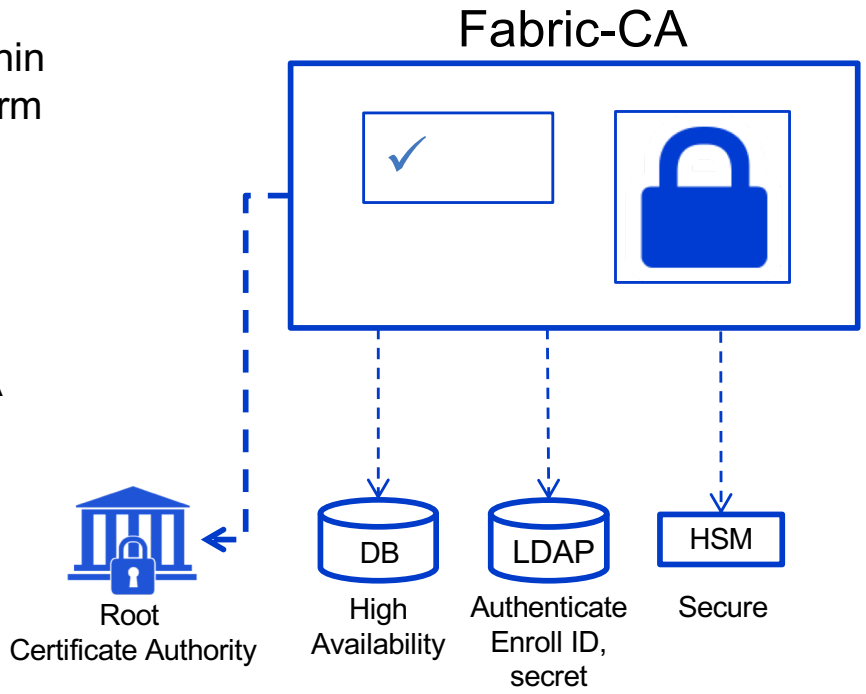
An MSP manages a set of identities within a distributed Fabric network

- 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

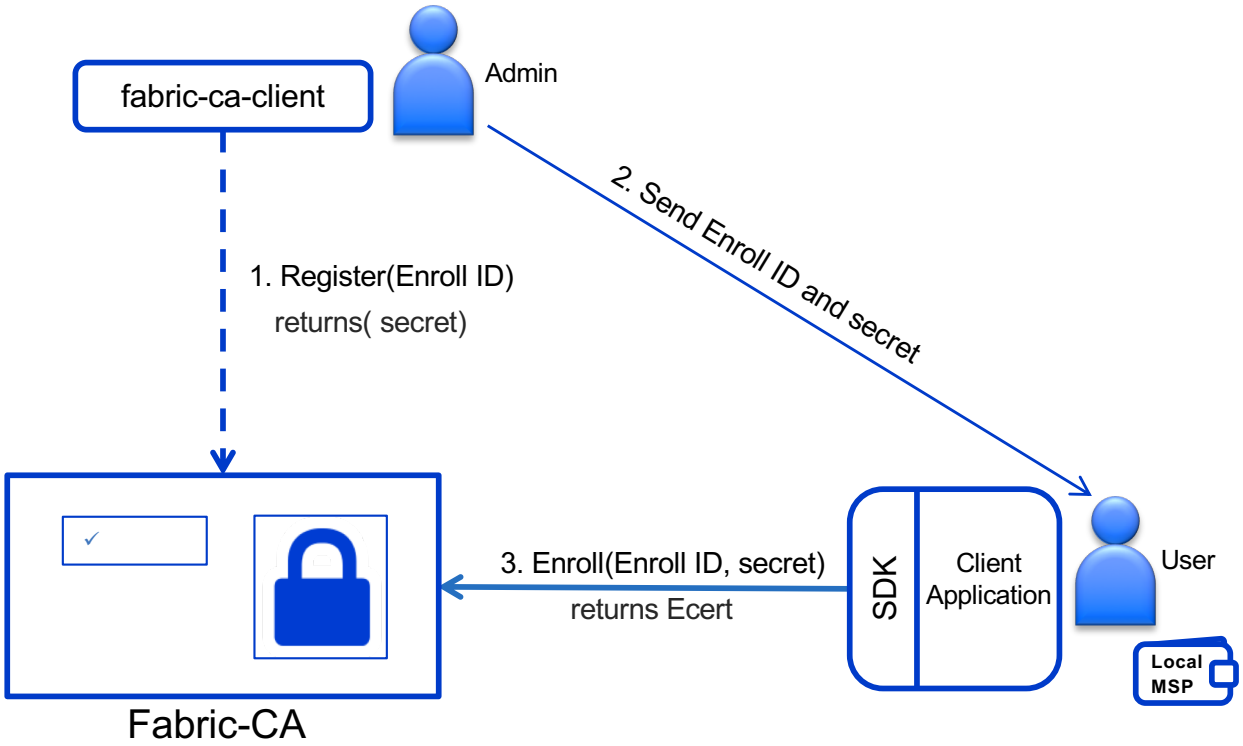


Fabric-CA

- 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



New User Registration and Enrollment



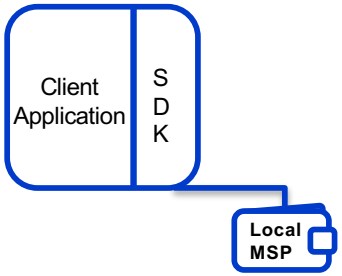
- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- Additional offline registration and enrollment options available

User Identities

Each client application has a local MSP to store user identities

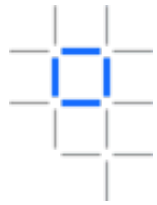
- Each local MSP includes:
  - **Keystore**
    - **Private key** for signing transactions
  - **Signcert**
    - **Public x.509 certificate**

- 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

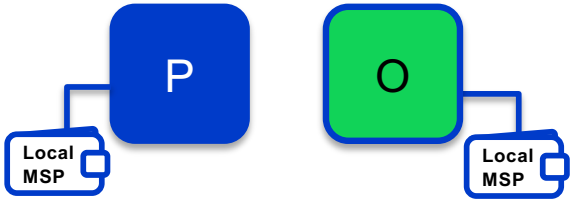
Peer and Orderer Identities

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)

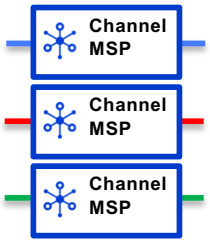


peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>

Channel MSP information

Channels include additional organisational MSP information

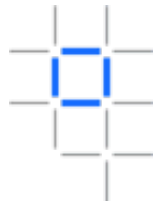
- Determines which orderers or peers can join the channel
- Determines client applications read or write access to the channel
- Stored in configuration blocks in the ledger
- Each channel MSP includes:
  - **admincerts**
    - Any public certificates for administrators
  - **cacerts**
    - The CA public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



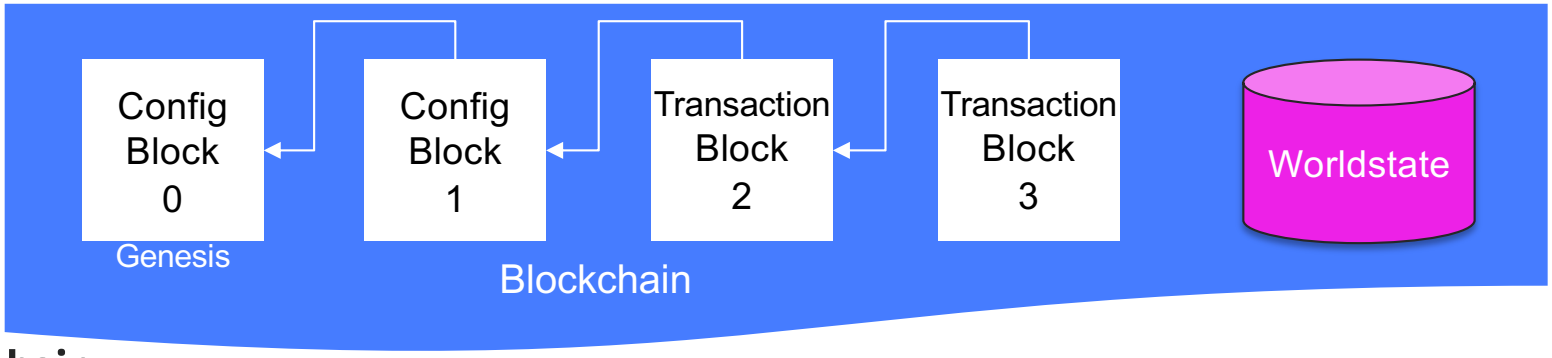
Channels

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

Fabric Ledger

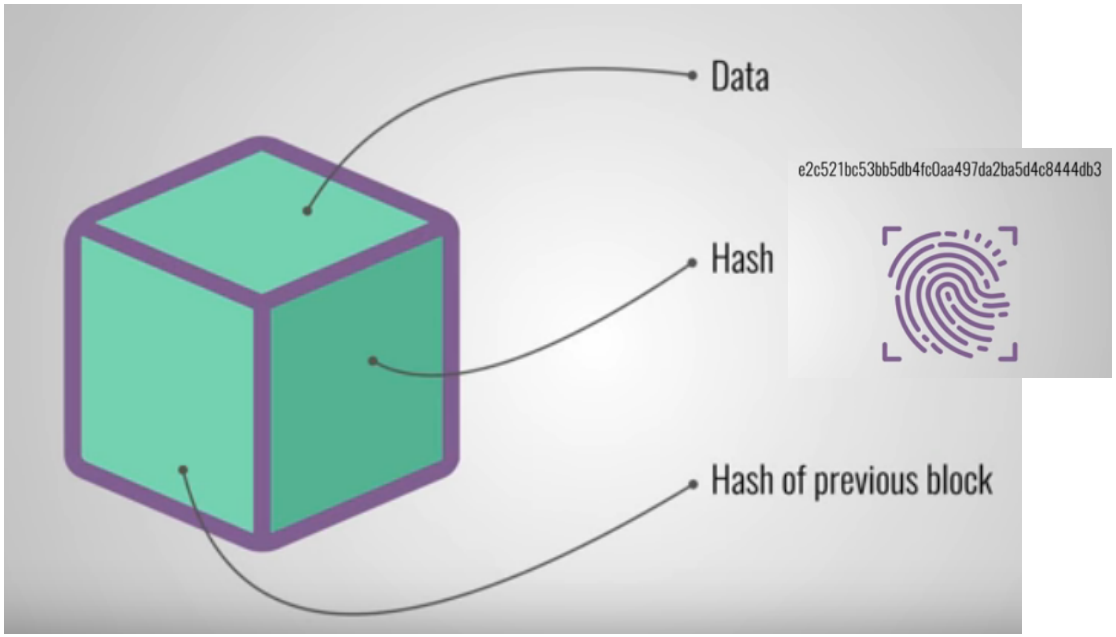


- The Fabric ledger is maintained by each peer and includes the blockchain and worldstate
- A separate ledger is maintained for each channel the peer joins
- Transaction read/write sets are written to the blockchain
- Channel configurations are also written to the blockchain
- The worldstate can be either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

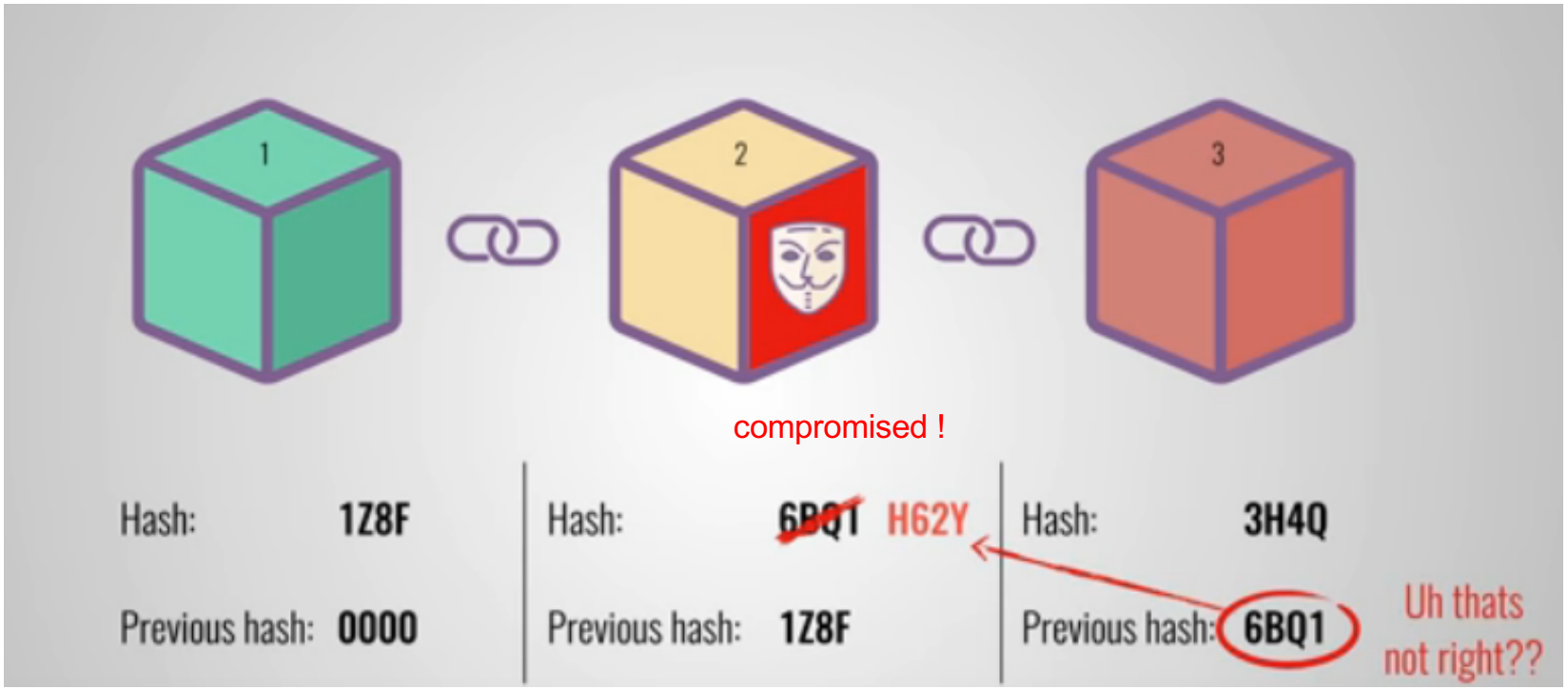


Blocks in the shared ledger

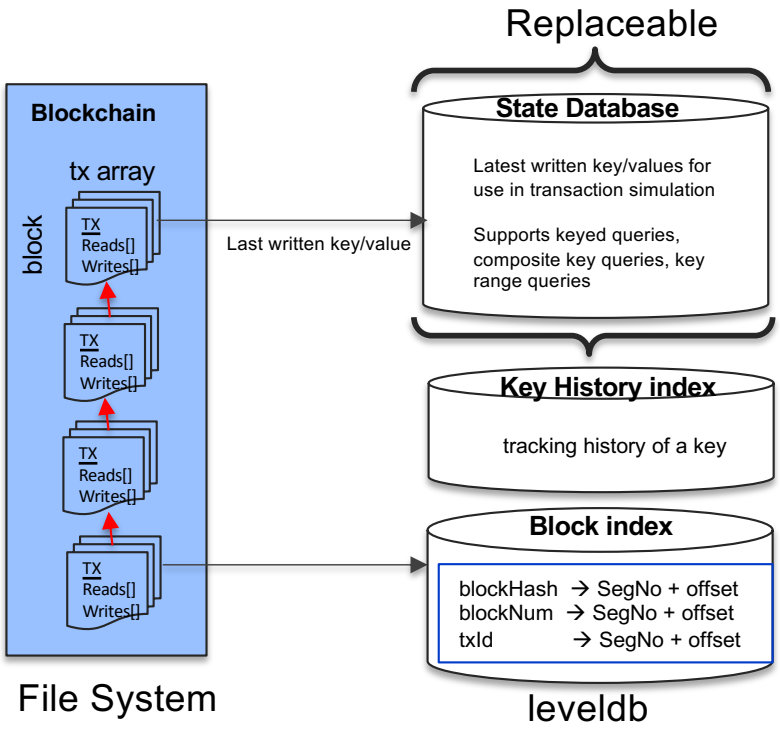
- contain the data to be stored in the blockchain
- the hash of the data
- hash of previous block



Immutability of blocks

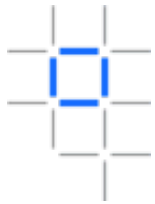


Fabric Ledger

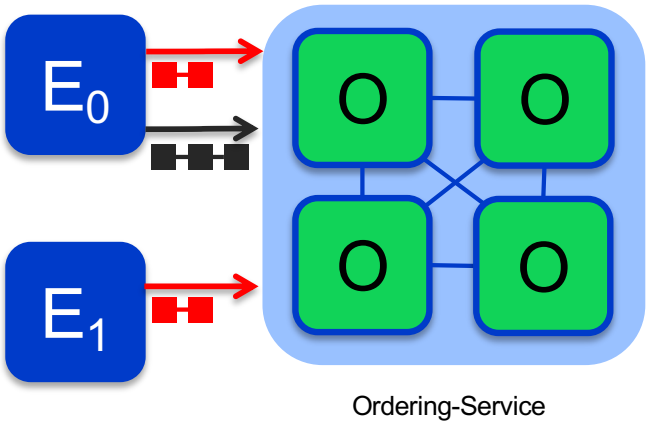


CouchDB (external option) adds rich data queries

Channels

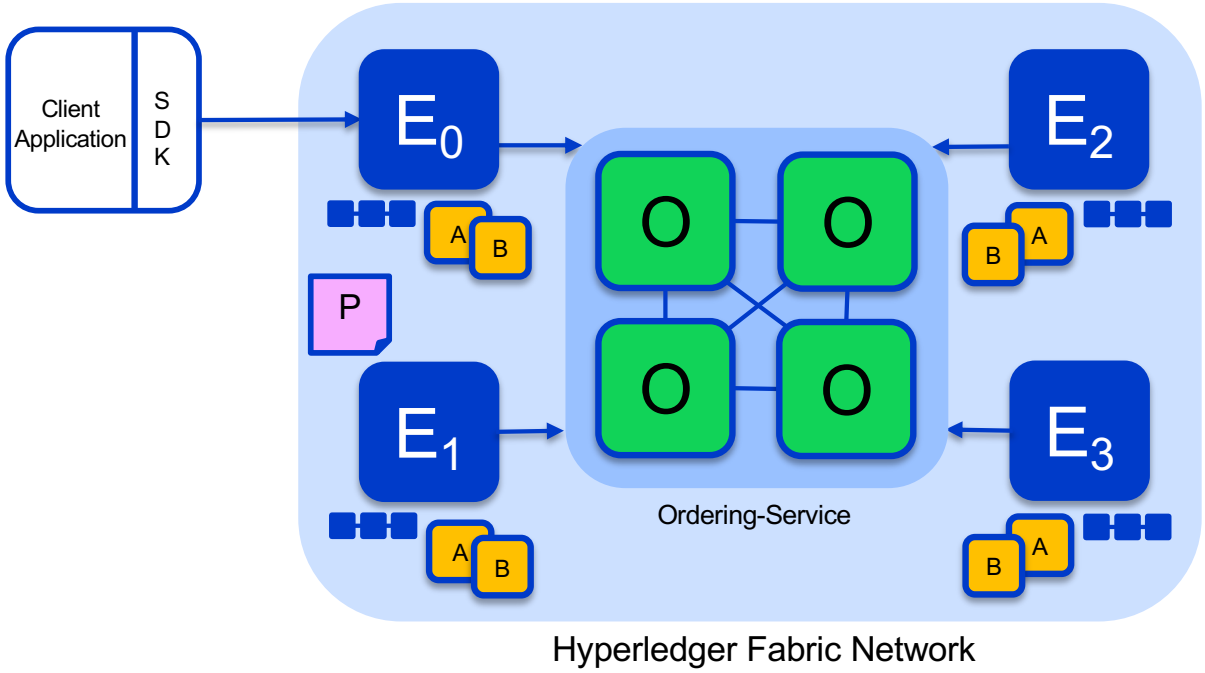


Channels provide privacy between different ledgers

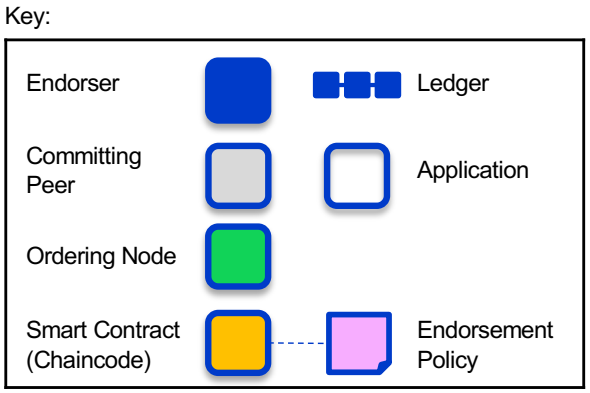


- Ledgers exist in the scope of a channel
  - Channels can be shared across an entire network of peers
  - Channels can be permissioned for a specific set of participants
- Chaincode is installed on peers to access the worldstate
- Chaincode is instantiated on specific channels
- Peers can participate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

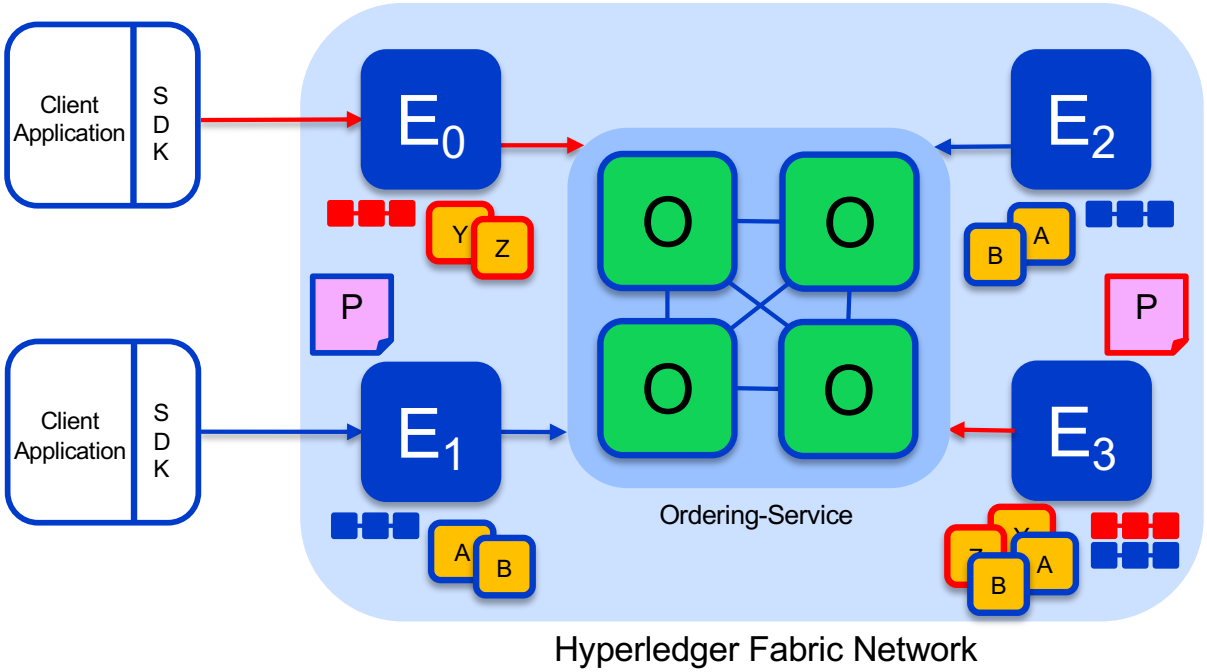
Single Channel Network



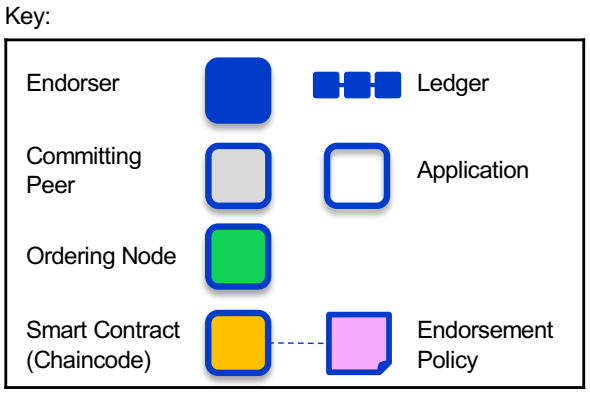
- All peers connect to the same channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3



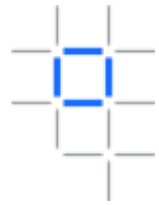
Multi Channel Network



- 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

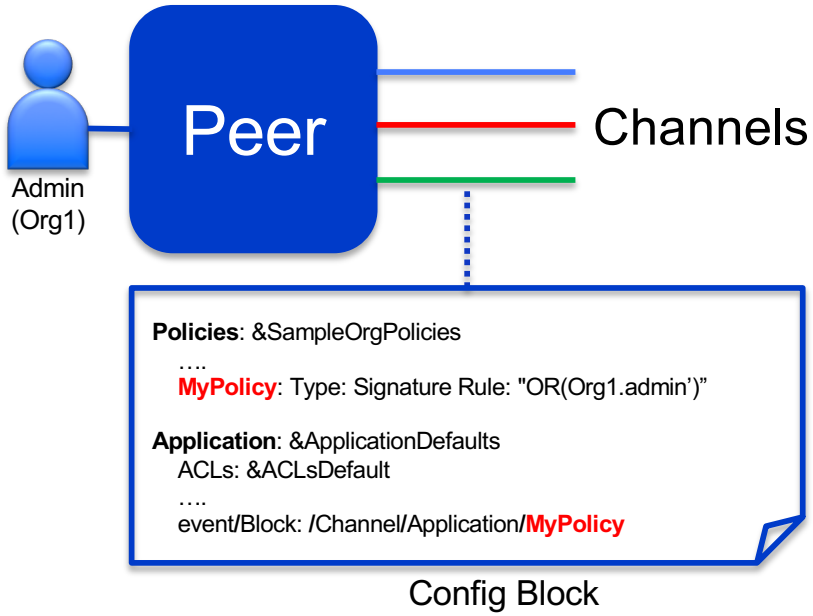


ACL mechanism per channel



Support policy based access control for peer functions per channel

- Access control defined for channel and peer resources:
  - User / System chaincode
  - Events stream
- Policies specify identities and include defaults for:
  - Readers
  - Writers
  - Admins
- Policies can be either:
  - Signature : Specific user type in org
  - ImplicitMeta : “All/Any/Majority” signature types
- Custom policies can be configured for ACLs

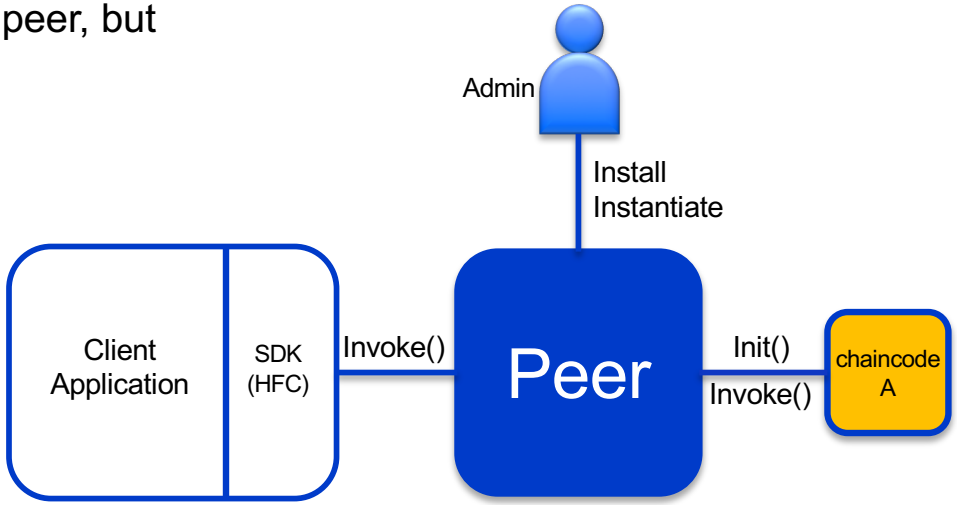


https://jira.hyperledger.org/browse/FAB-8727

Chaincode

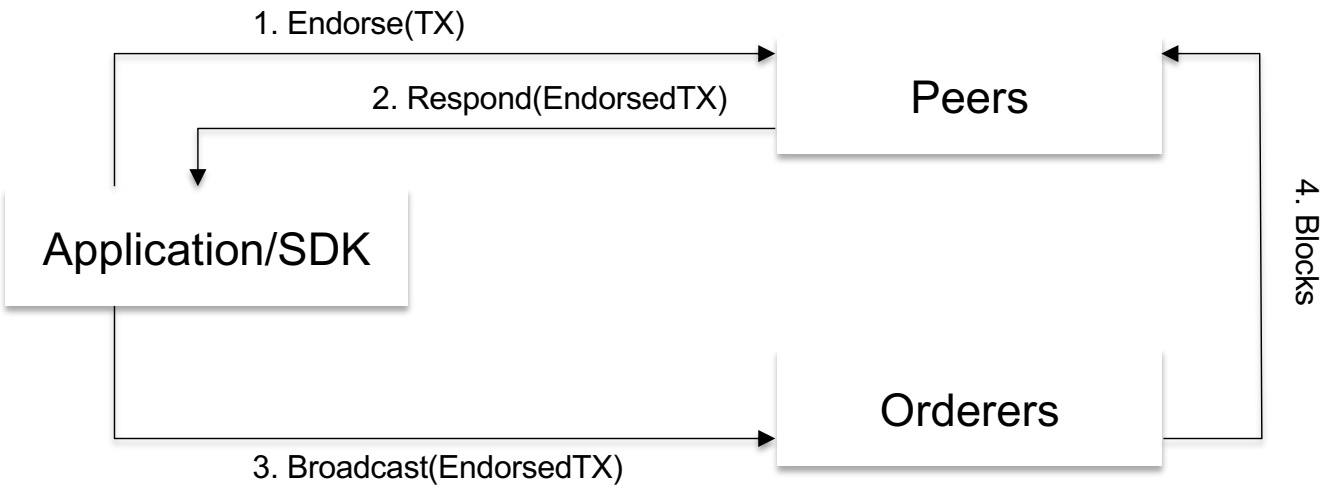
Chaincode (Smart Contract) contains business logic deployed to peers

- Installed on peers and instantiated on channels
- Run in secured docker images separate to the peer
- Interact with the world state through the Fabric shim interface
- Each chaincode has its own scoped world state
- Needs to be present to endorse a transaction from a peer, but does not need to be present to commit transactions
- Language support for:
  - Golang
  - Javascript / Typescript
  - Java
- Implements:
  - Init() - Called on instantiate and upgrade
  - Invoke() – Called from client application



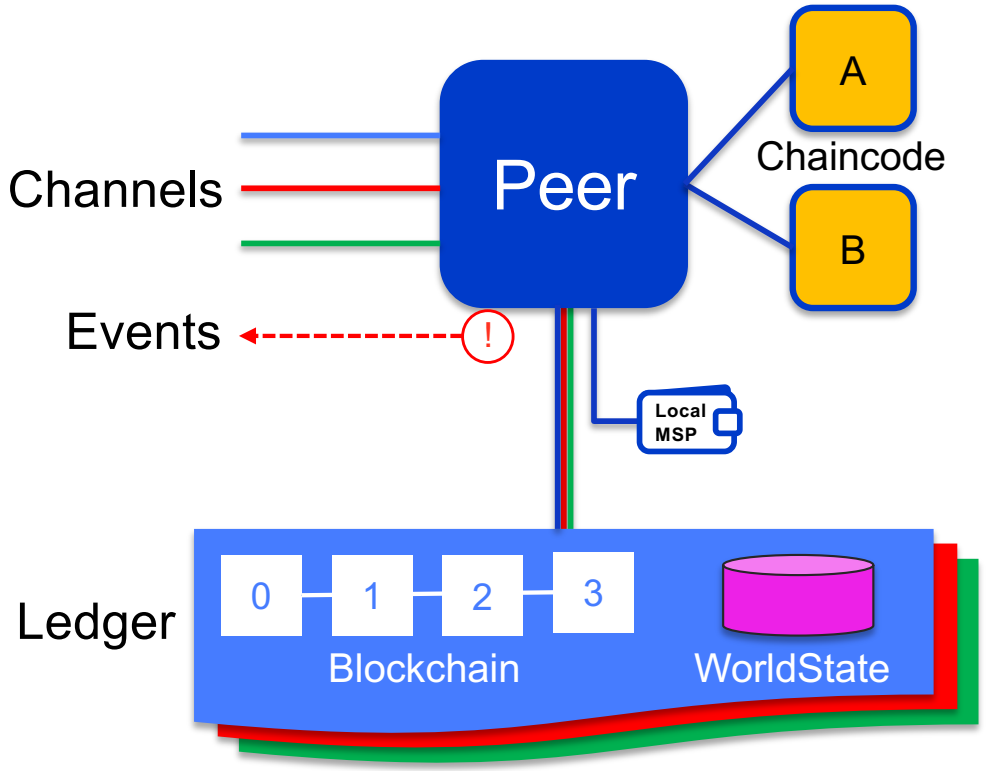
https://jira.hyperledger.org/browse/FAB-8063

Transaction data flow



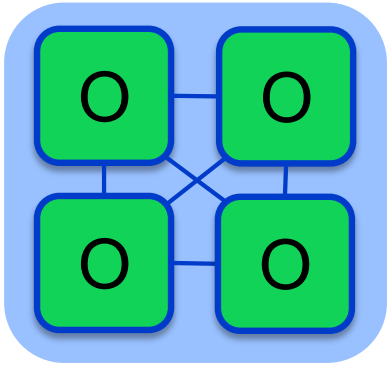
Fabric Peer

- Each peer:
  - Connects to one or more channels
  - Maintains one ledger 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



Ordering Service

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

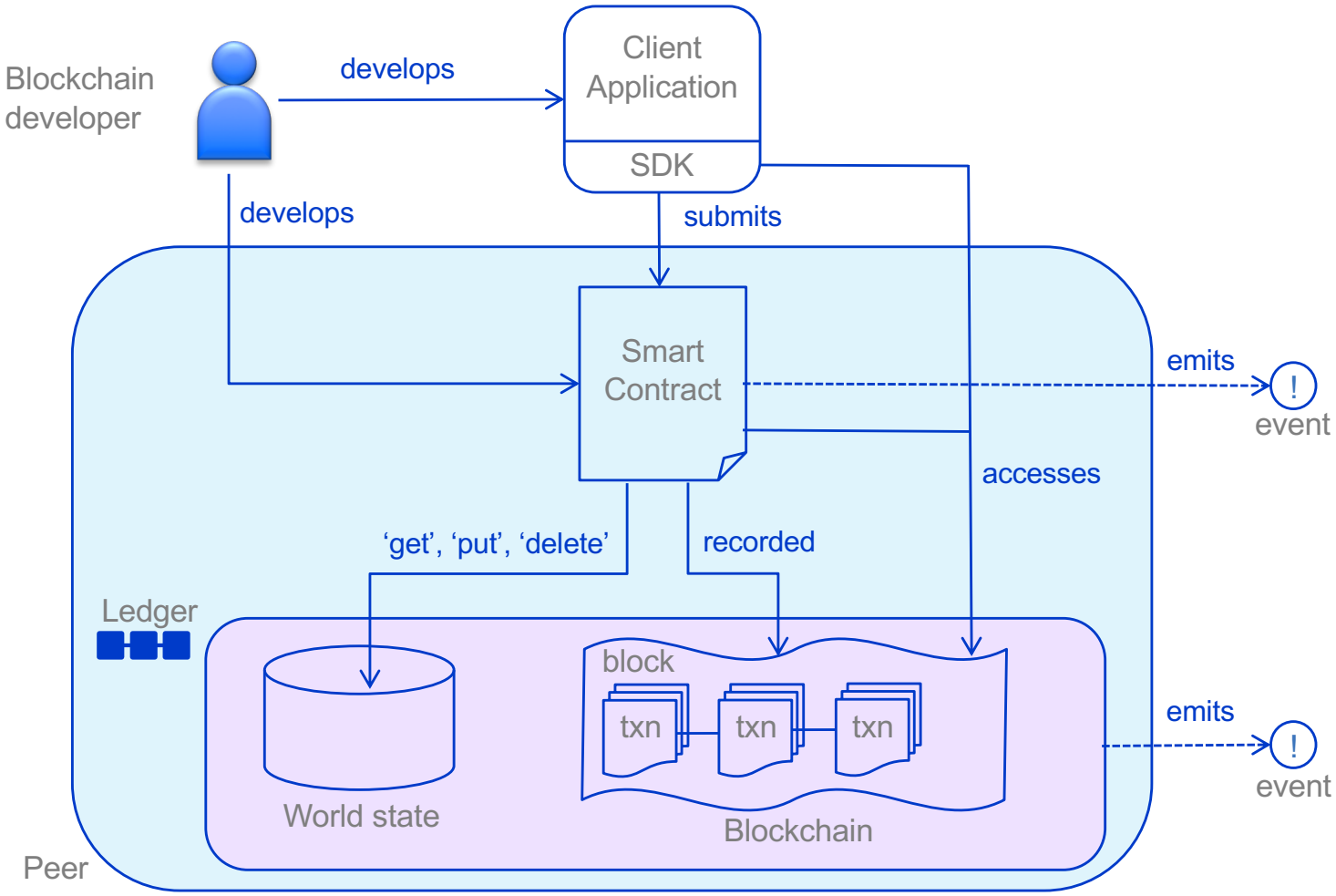


Ordering-Service

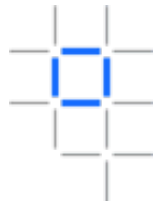
Different configuration options for the ordering service include:

- SOLO
  - Single node for development
- Kafka : Crash fault tolerant consensus
  - Minimum recommendation: 3 Orderer nodes, 4 Kafka nodes, 3 Zookeeper nodes
- Raft: Crash fault tolerant consensus
  - New with Hyperledger Fabric 1.4.1
  - Recommended

How applications interact with the ledger

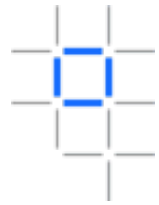


Nodes and roles



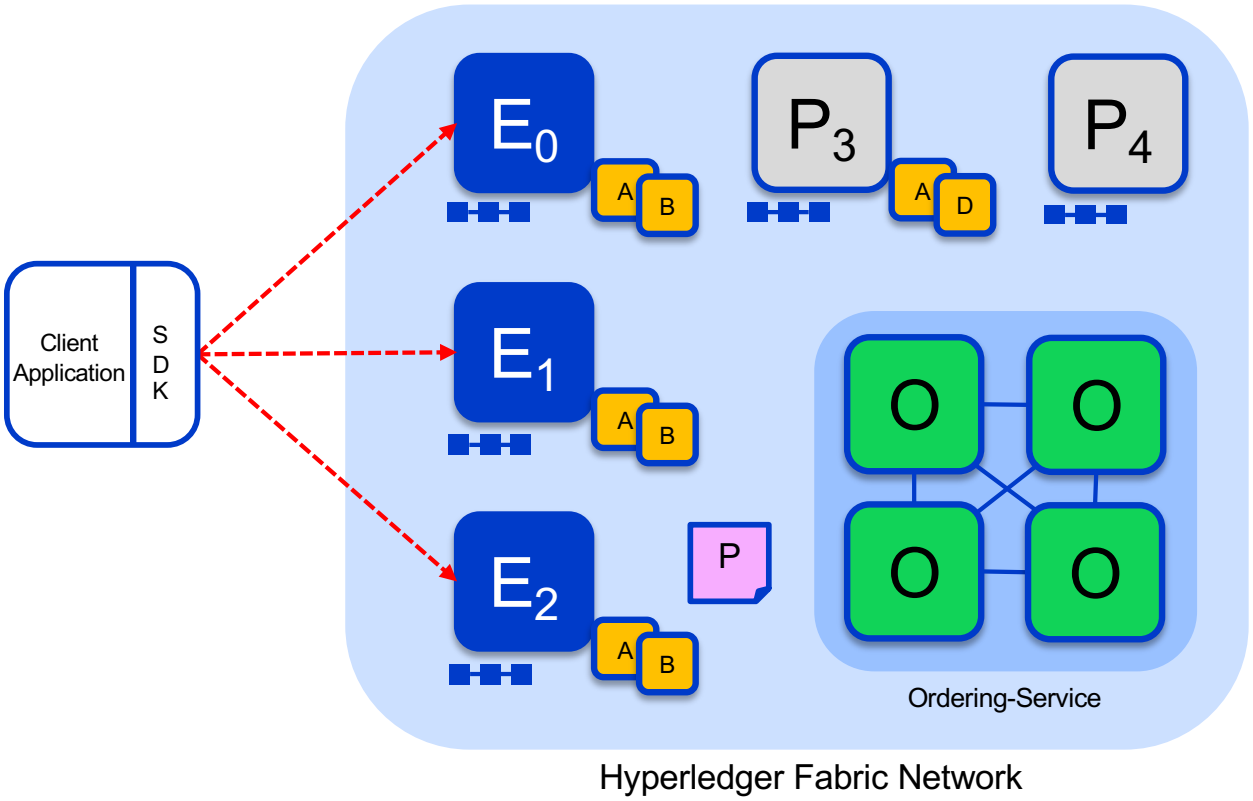
P	**Peer:** Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).
E	**Endorsing Peer:** Specialized peer also endorses transactions by receiving a transaction proposal and responds by granting or denying endorsement. Must hold smart contract.
O	**Ordering Node:** Approves 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

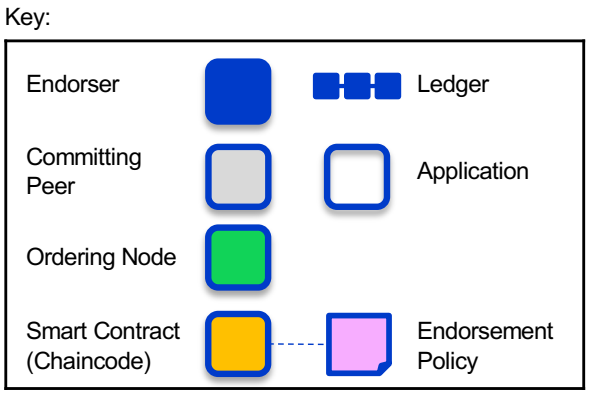


Hyperledger Fabric Network

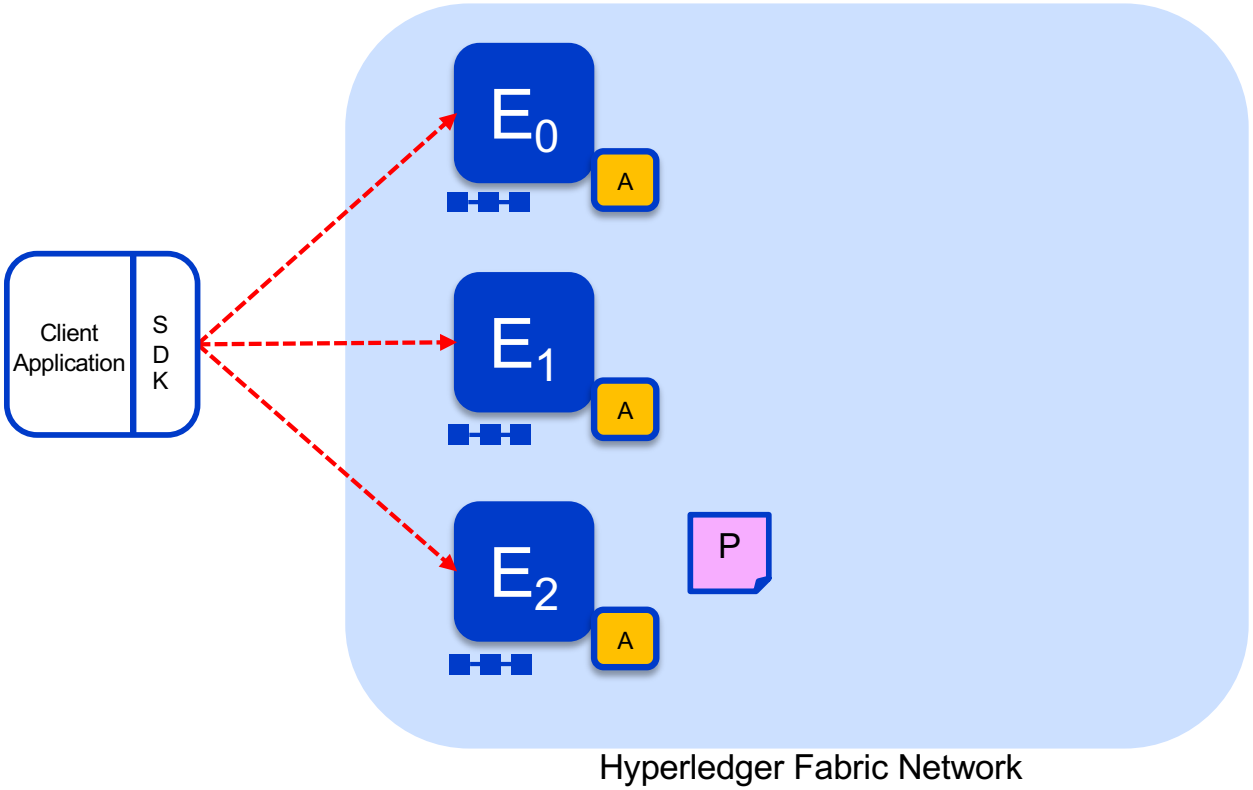
Application proposes transaction

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}



Sample transaction: Step 1/7 – Propose transaction



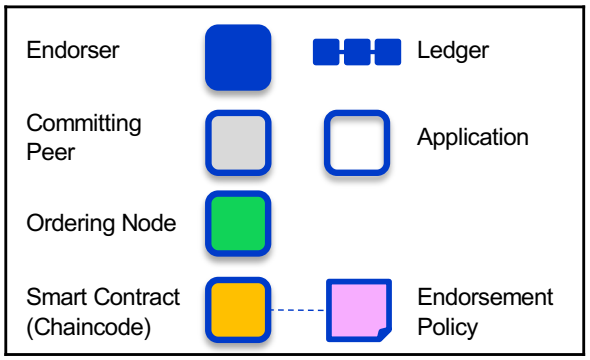
Hyperledger Fabric Network

Application proposes transaction

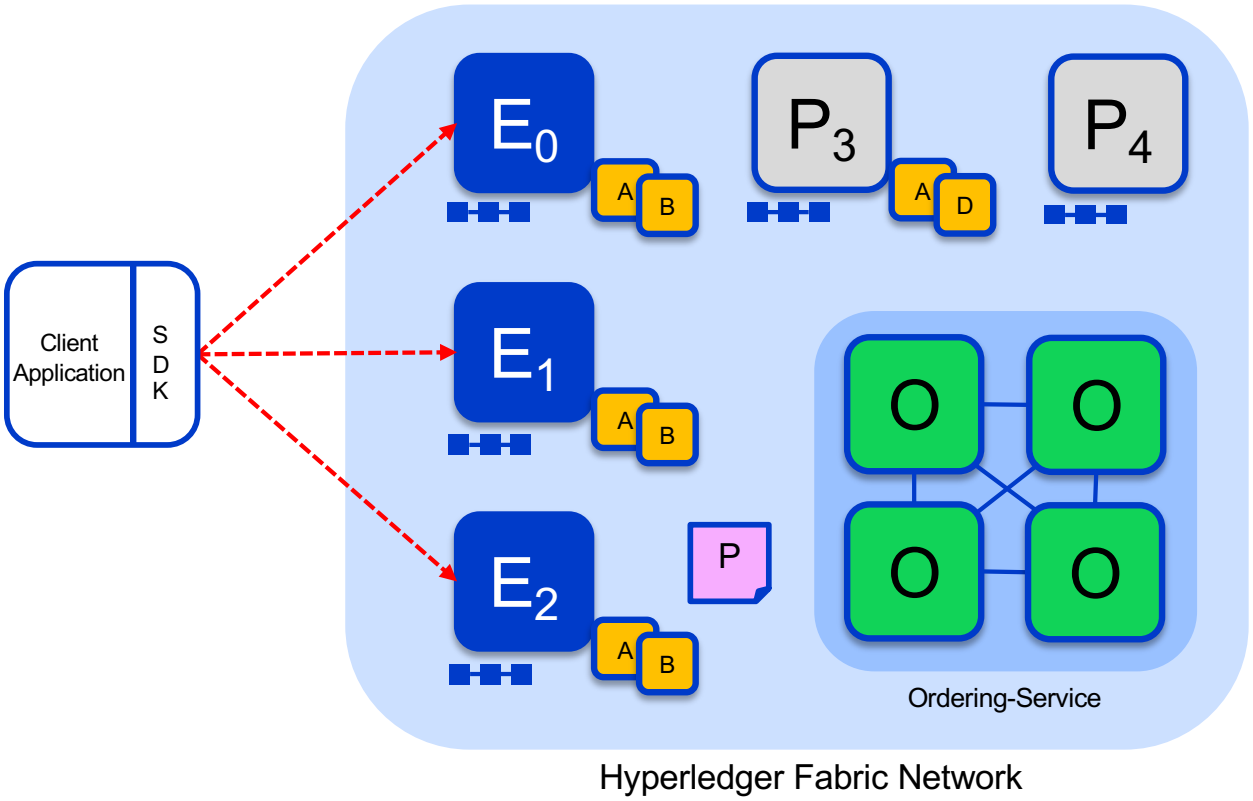
Endorsement policy:
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

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 1/7 – Propose transaction

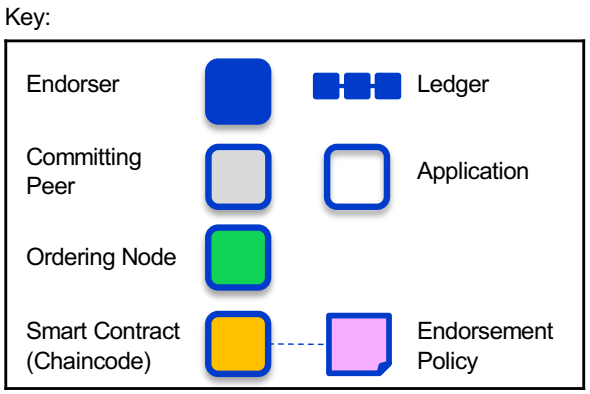


Hyperledger Fabric Network

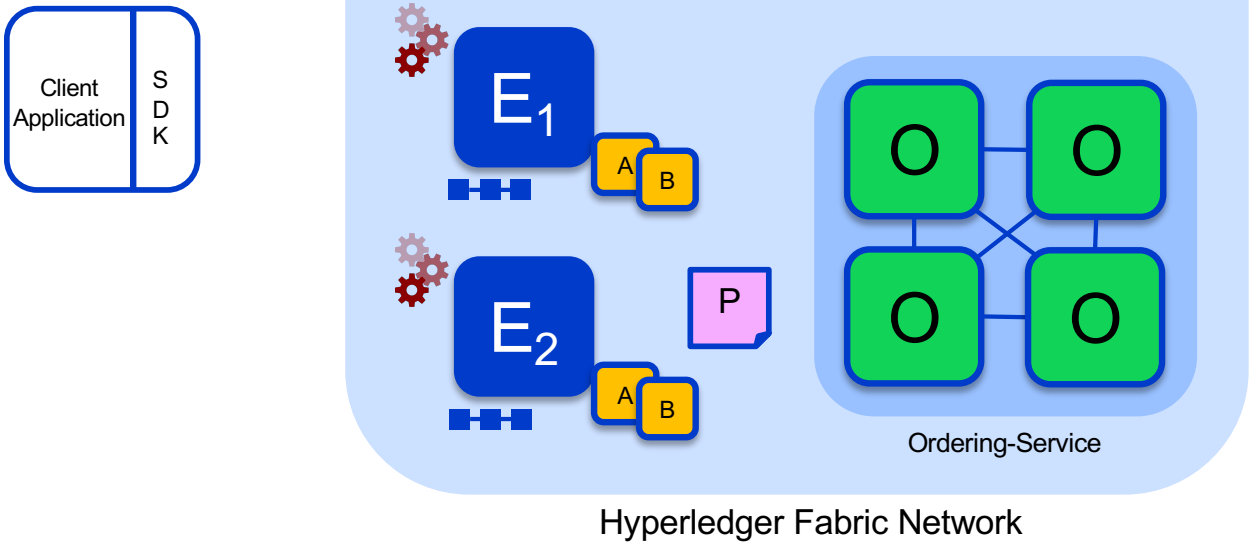
Application proposes transaction

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}



Sample transaction: Step 2/7 – Execute proposal



Hyperledger Fabric Network

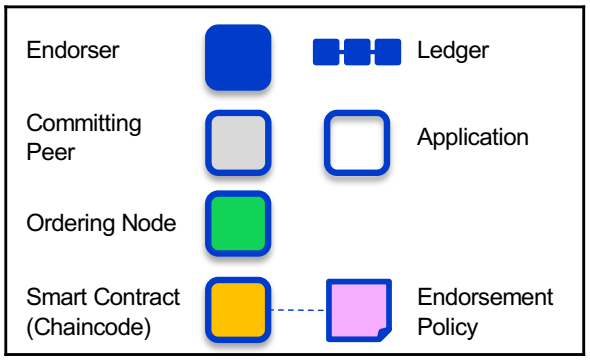
Endorsers Execute Proposals

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

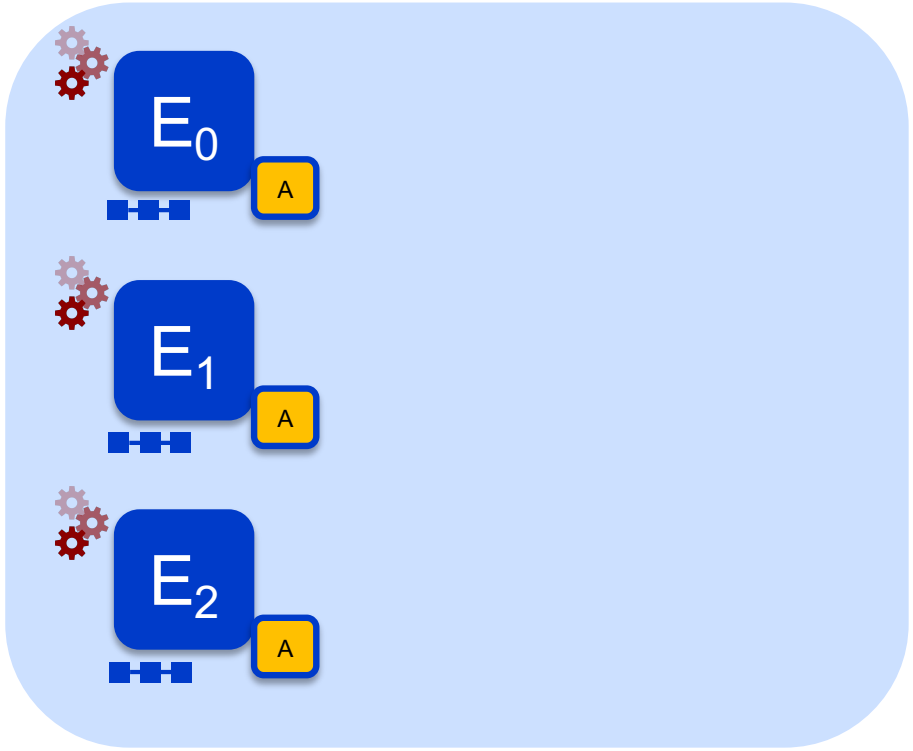
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



Sample transaction: Step 2/7 – Execute proposal



Hyperledger Fabric Network

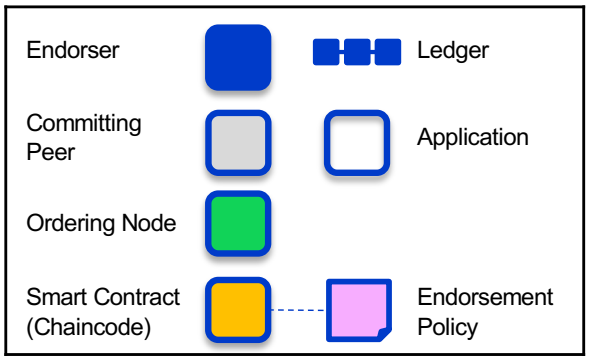
Endorsers Execute Proposals

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

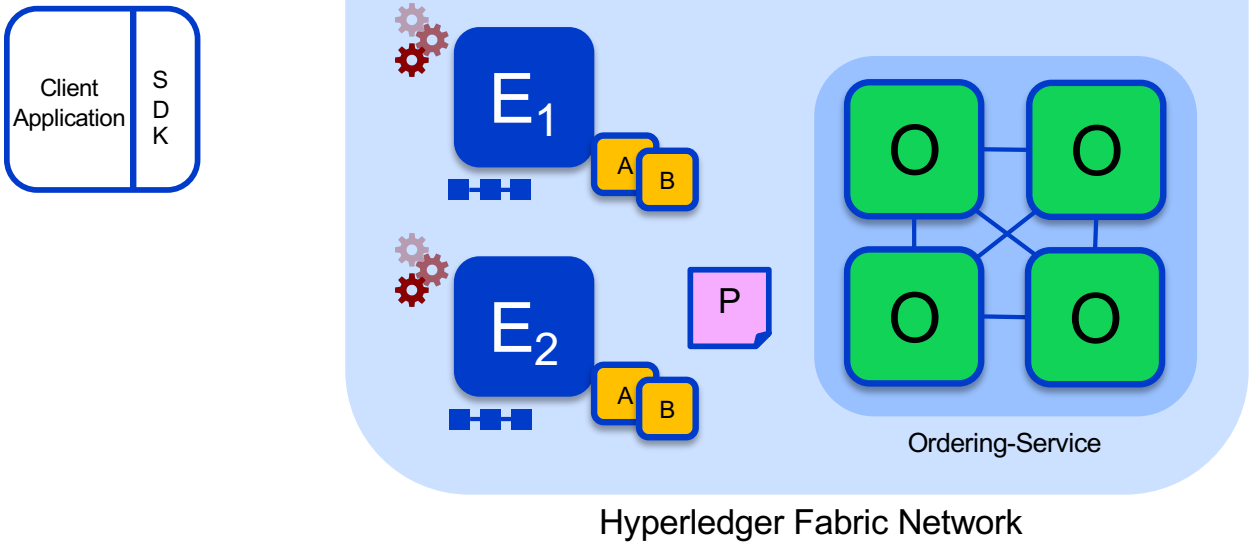
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



Sample transaction: Step 2/7 – Execute proposal



Hyperledger Fabric Network

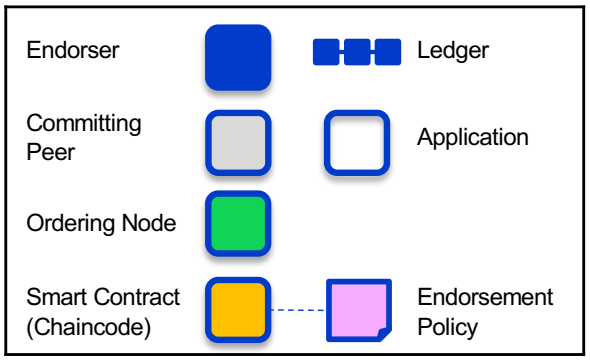
Endorsers Execute Proposals

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

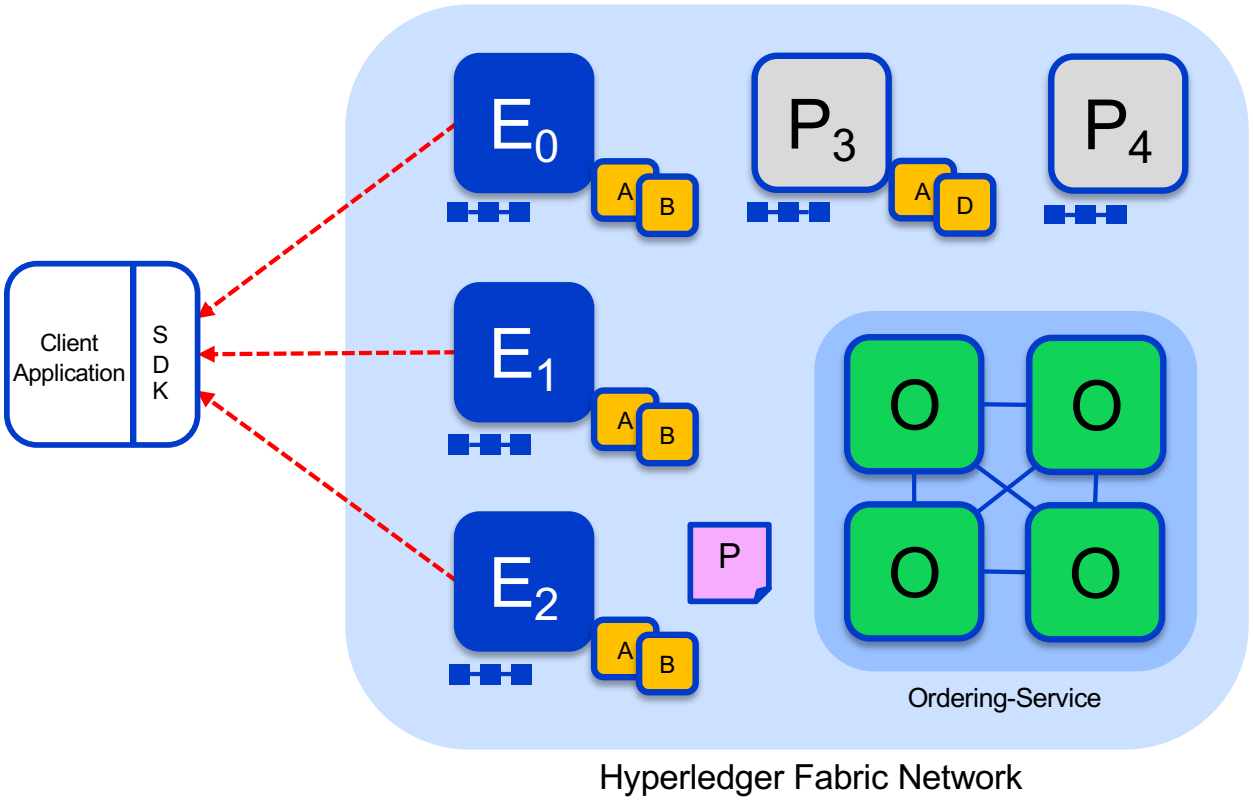
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed & encrypted

Key:



Sample transaction: Step 3/7 – Proposal Response



Hyperledger Fabric Network

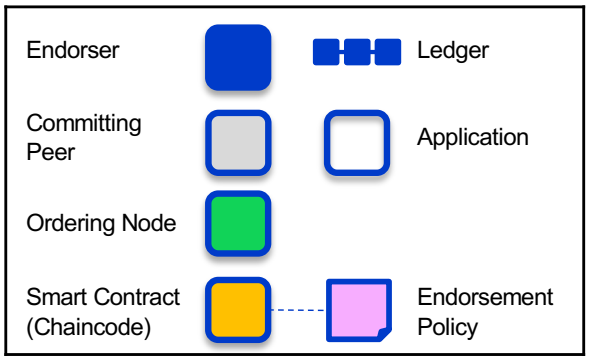
Application receives responses

RW sets are asynchronously returned to application

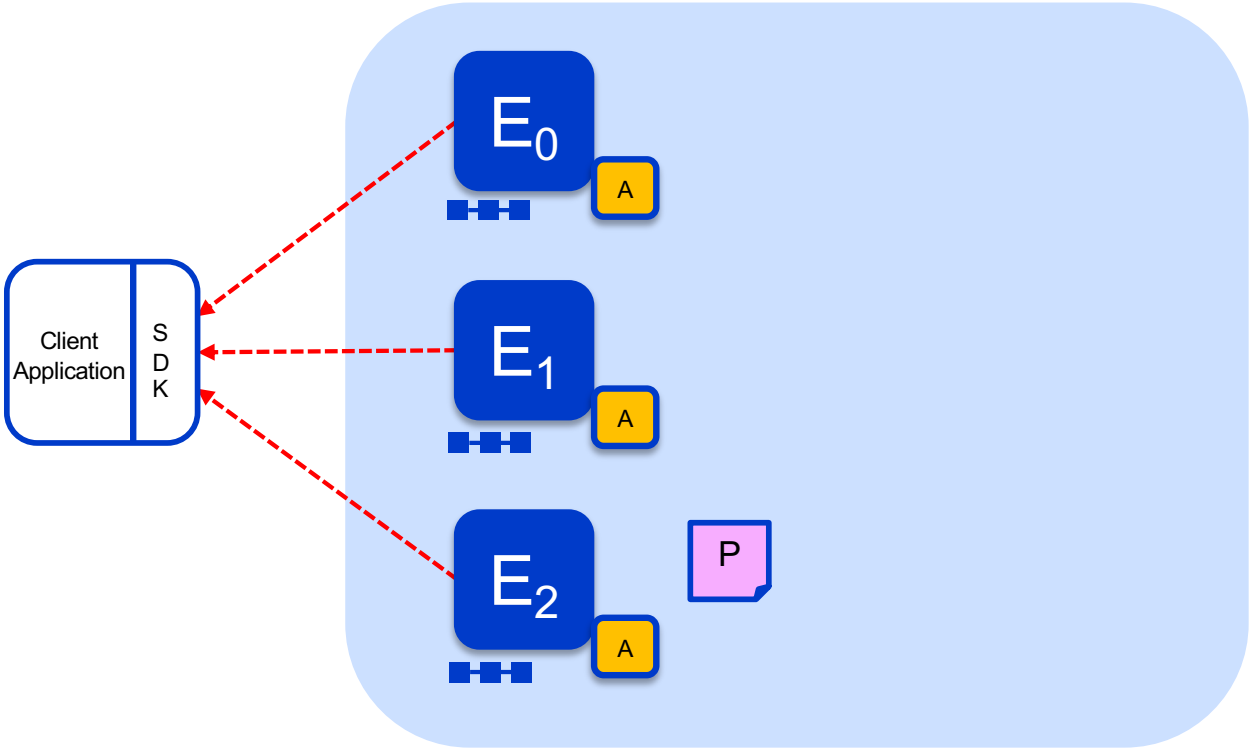
The RW sets are signed by each endorser, and also includes each record version number

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

Key:



Sample transaction: Step 3/7 – Proposal Response



Hyperledger Fabric Network

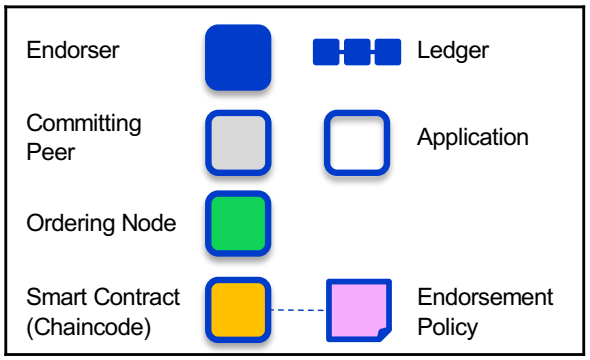
Application receives responses

RW sets are asynchronously returned to application

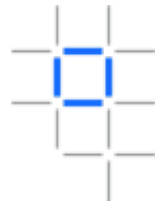
The RW sets are signed by each endorser, and also includes each record version number

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

Key:



Non-determinism in Blockchain



- Blockchain is a distributed processing system
  - Smart contracts are run multiple times and in multiple places
  - As we will see, smart contracts need to run deterministically in order for consensus to work
    - Particularly when updating the world state
- It's particularly difficult to achieve determinism with off-chain processing
  - Implement oracle services that are guaranteed to be consistent for a given transaction, or
  - Detect duplicates for a transaction in the blockchain, middleware or external system

random()

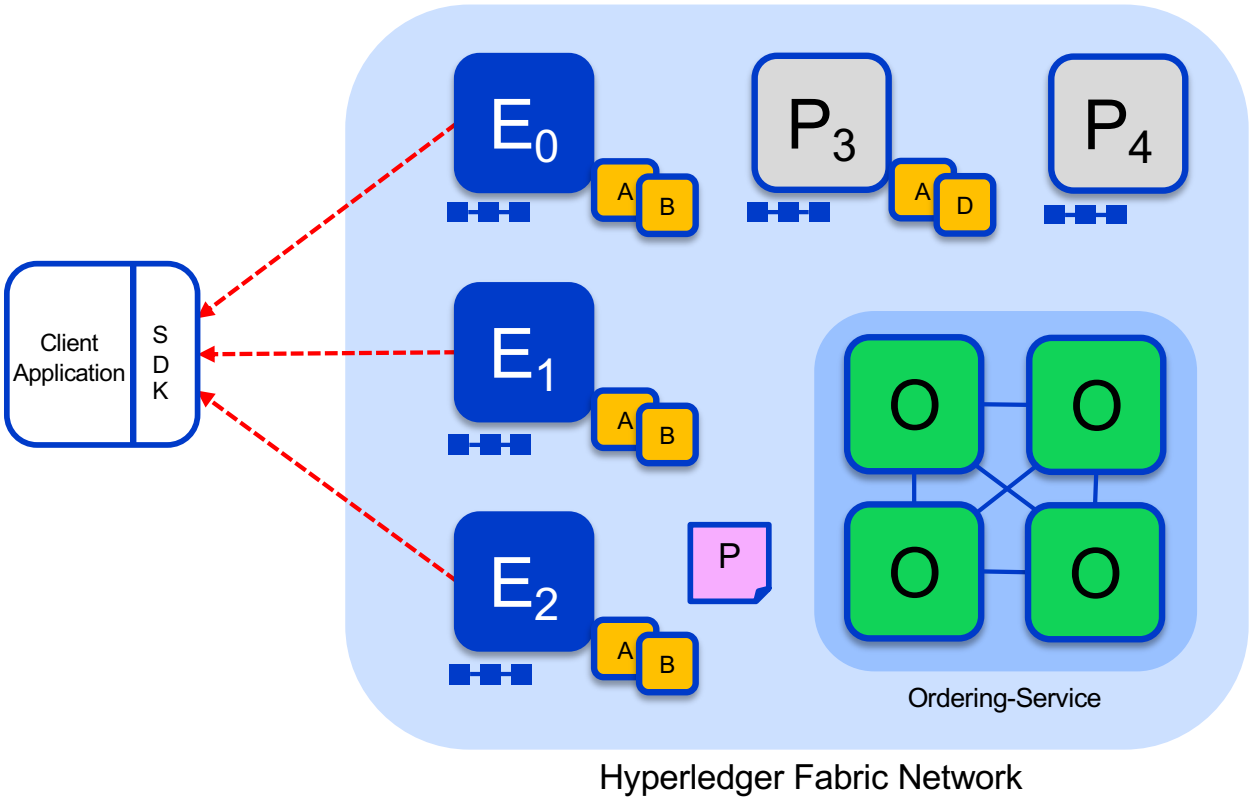
getExchangeRate()

getDateTime()

getTemperature()

incrementValue inExternalSystem(…)

Sample transaction: Step 3/7 – Proposal Response



Hyperledger Fabric Network

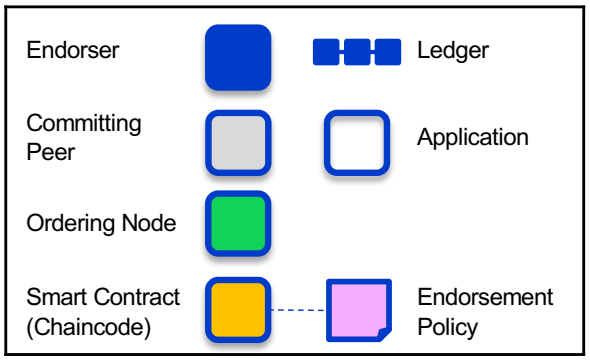
Application receives responses

RW sets are asynchronously returned to application

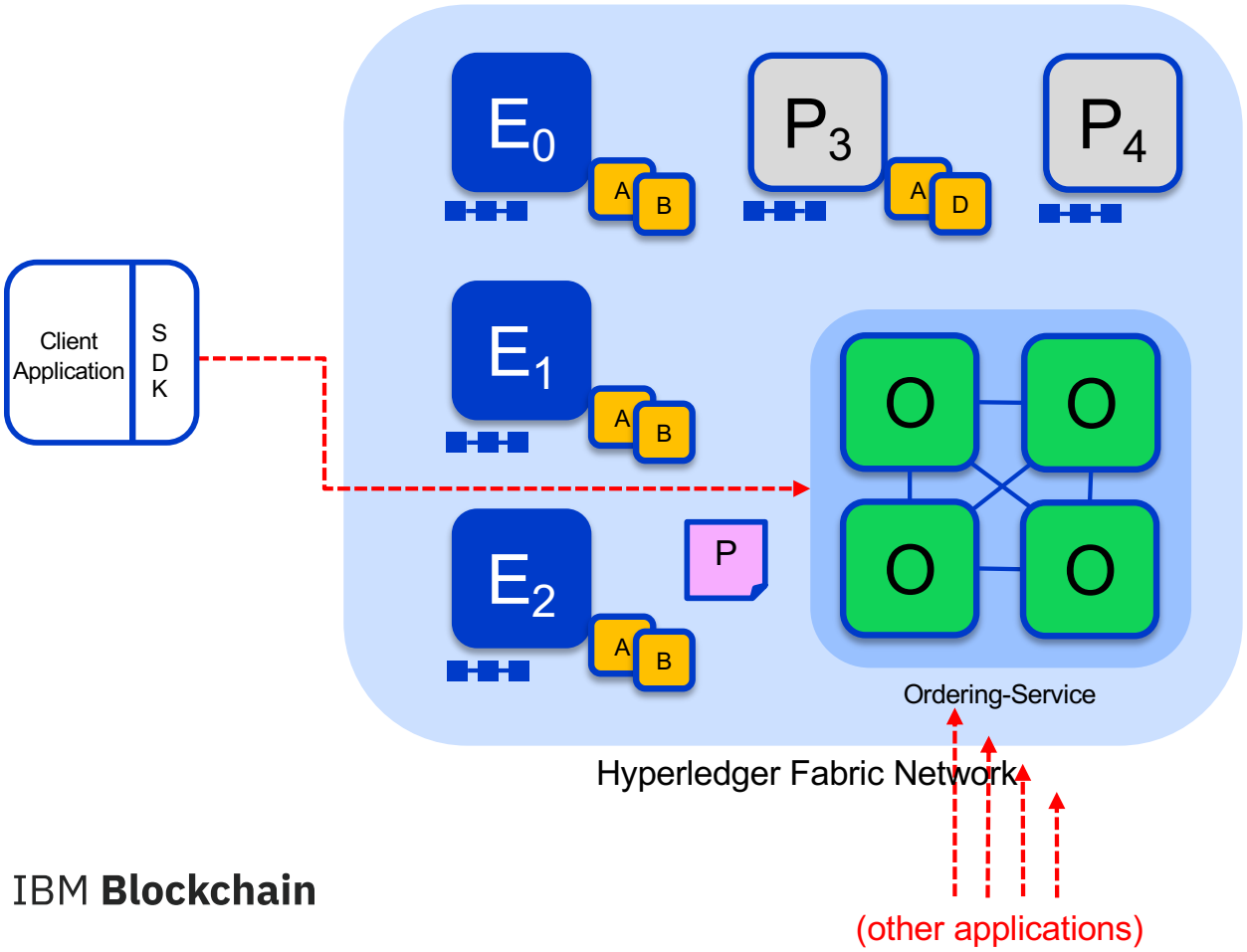
The RW sets are signed by each endorser, and also includes each record version number

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

Key:



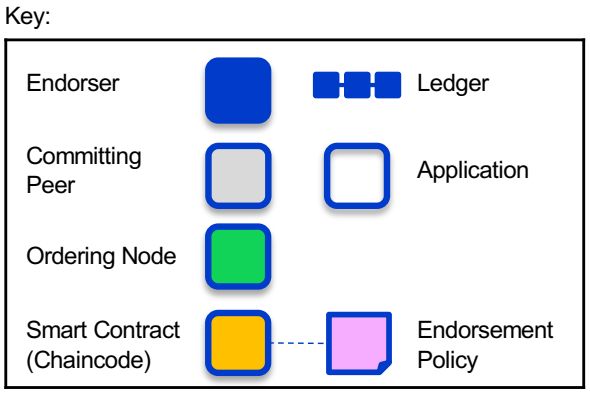
Sample transaction: Step 4/7 – Order Transaction



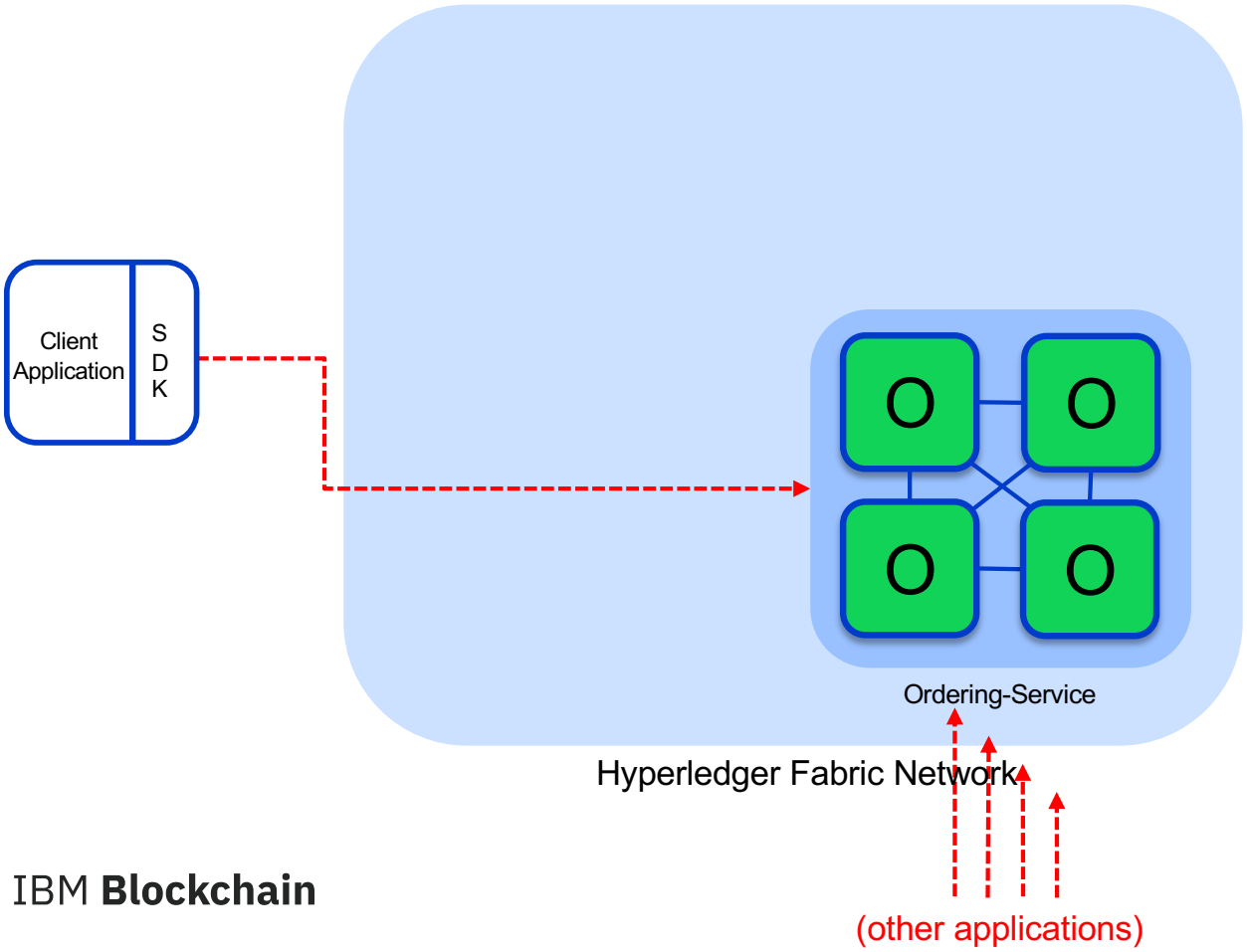
Responses submitted for ordering

Application submits responses as a transaction to be ordered.

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



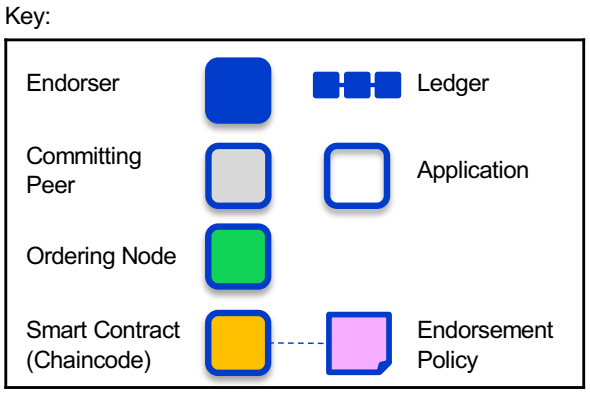
Sample transaction: Step 4/7 – Order Transaction



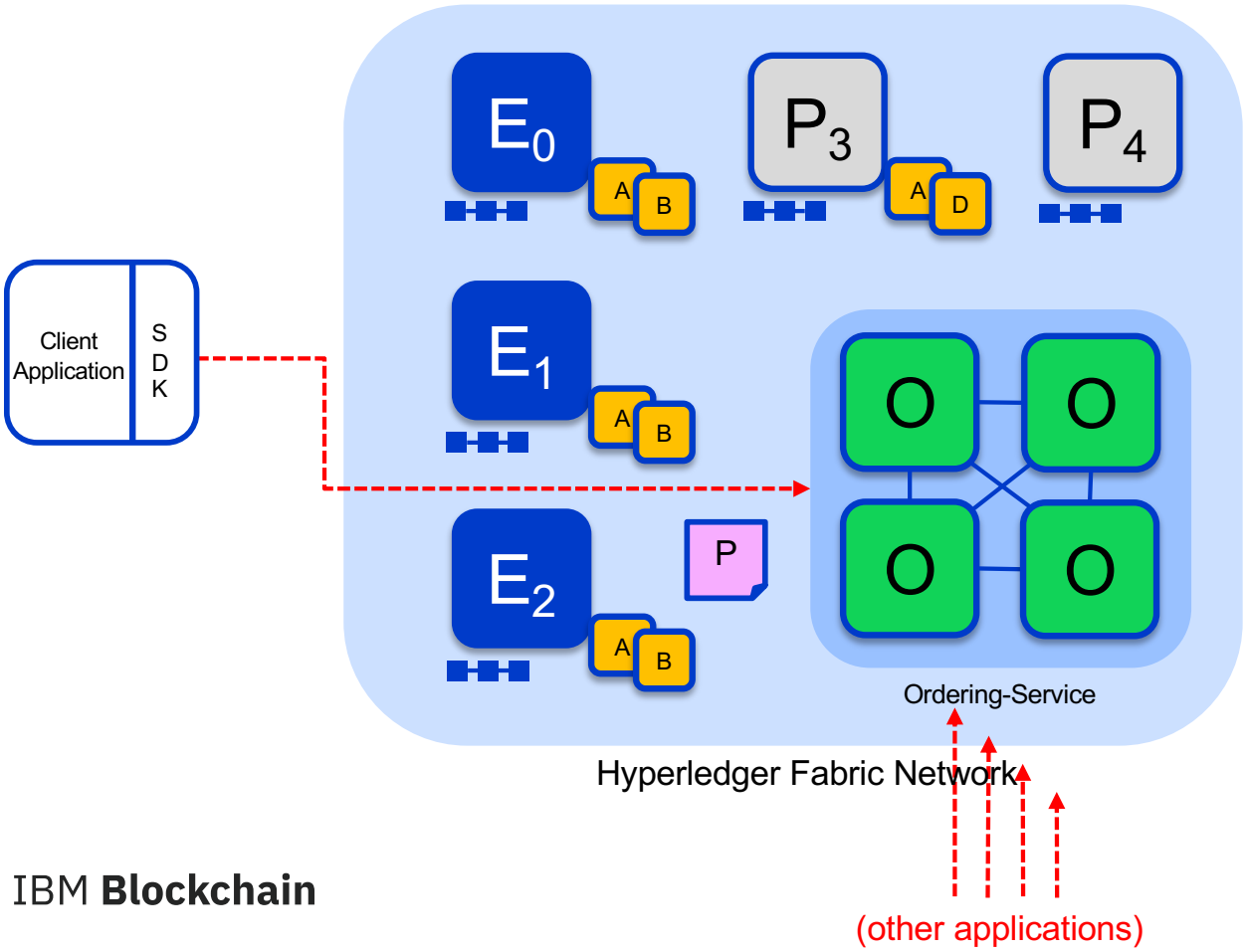
Responses submitted for ordering

Application submits responses as a transaction to be ordered.

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



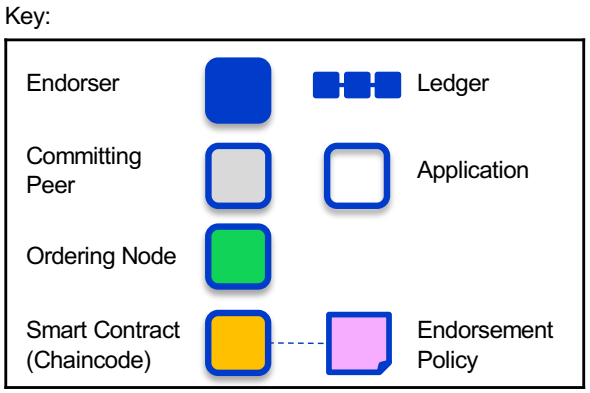
Sample transaction: Step 4/7 – Order Transaction



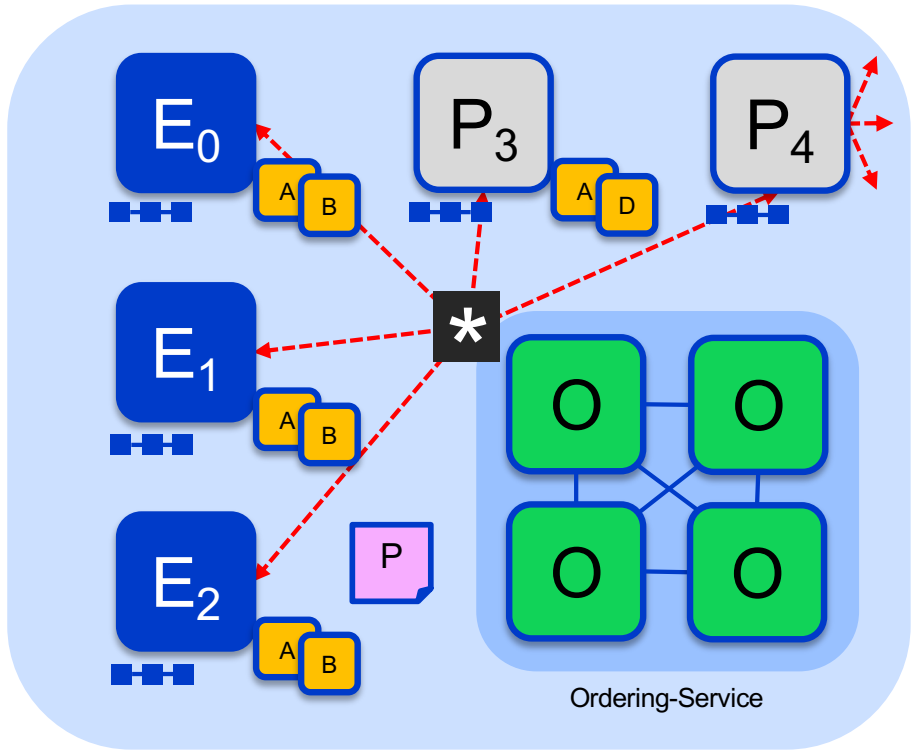
Responses submitted for ordering

Application submits responses as a transaction to be ordered.

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



Sample transaction: Step 5/7 – Deliver Transaction



Hyperledger Fabric Network

Client Application | SDK

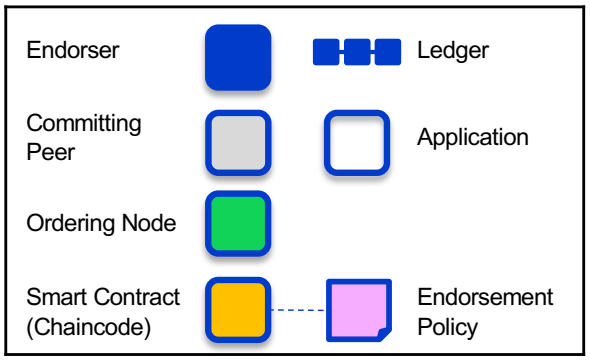
Orderer delivers to committing peers

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

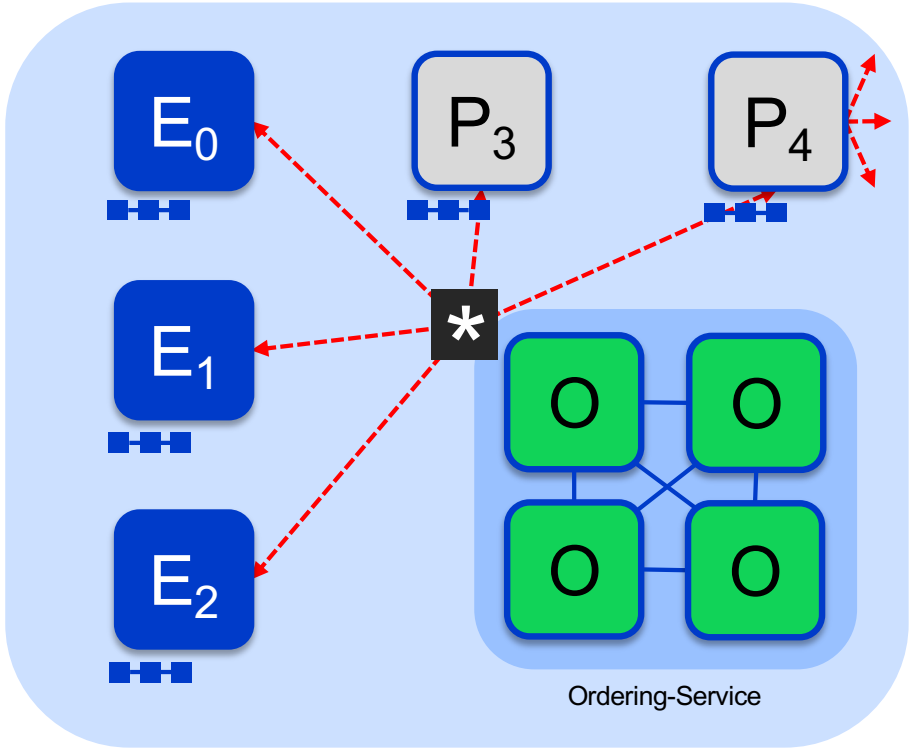
Different ordering algorithms available:

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

Key:



Sample transaction: Step 5/7 – Deliver Transaction



Hyperledger Fabric Network

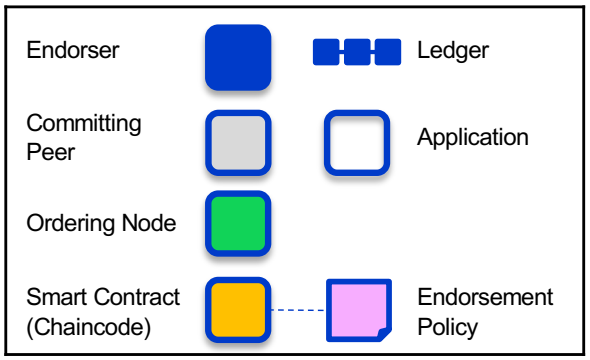
Orderer delivers to committing peers

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

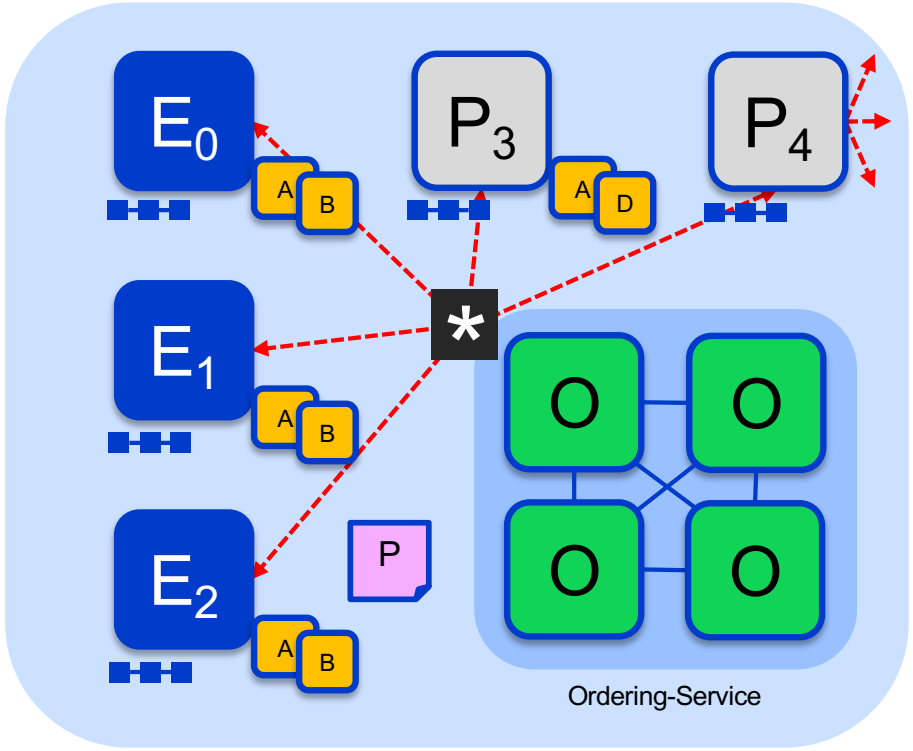
Different ordering algorithms available:

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

Key:



Sample transaction: Step 5/7 – Deliver Transaction



Hyperledger Fabric Network

Client Application | SDK

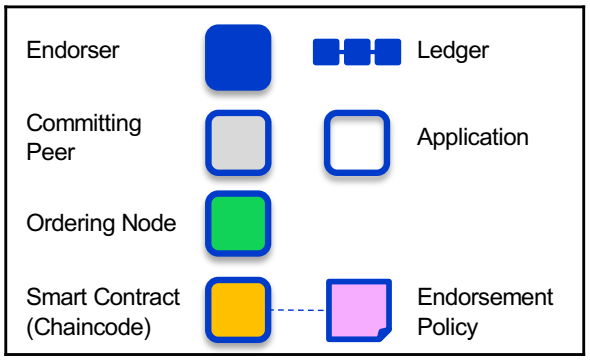
Orderer delivers to committing peers

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

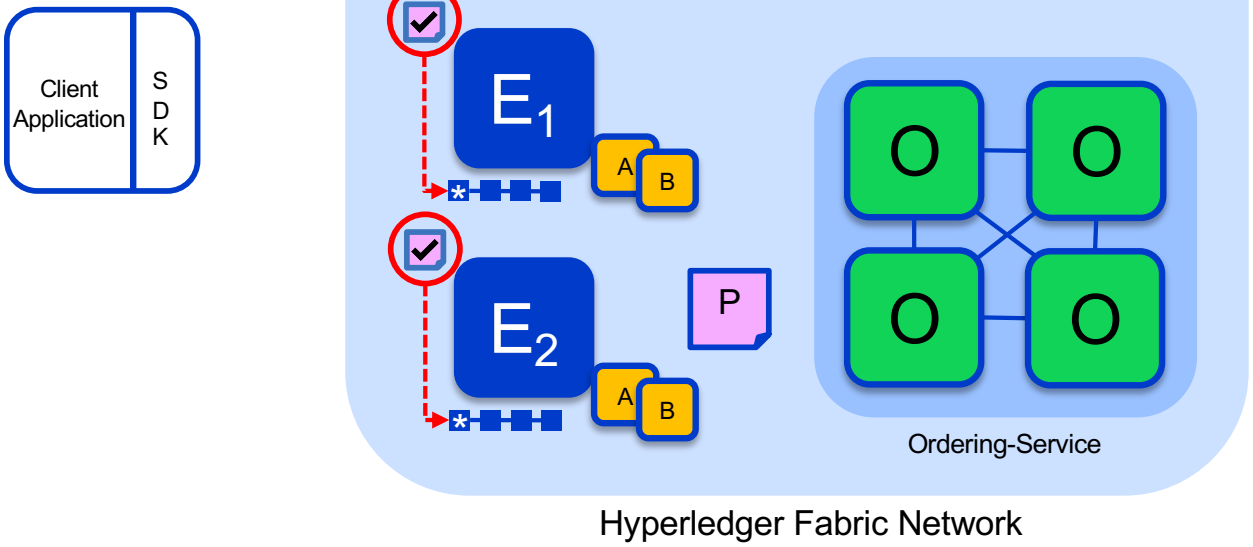
Different ordering algorithms available:

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

Key:



Sample transaction: Step 6/7 – Validate Transaction



Hyperledger Fabric Network

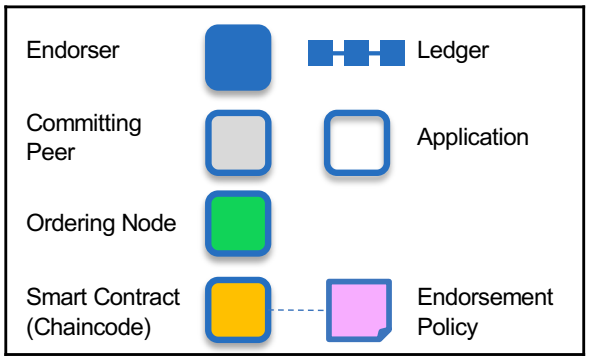
Committing peers validate transactions

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

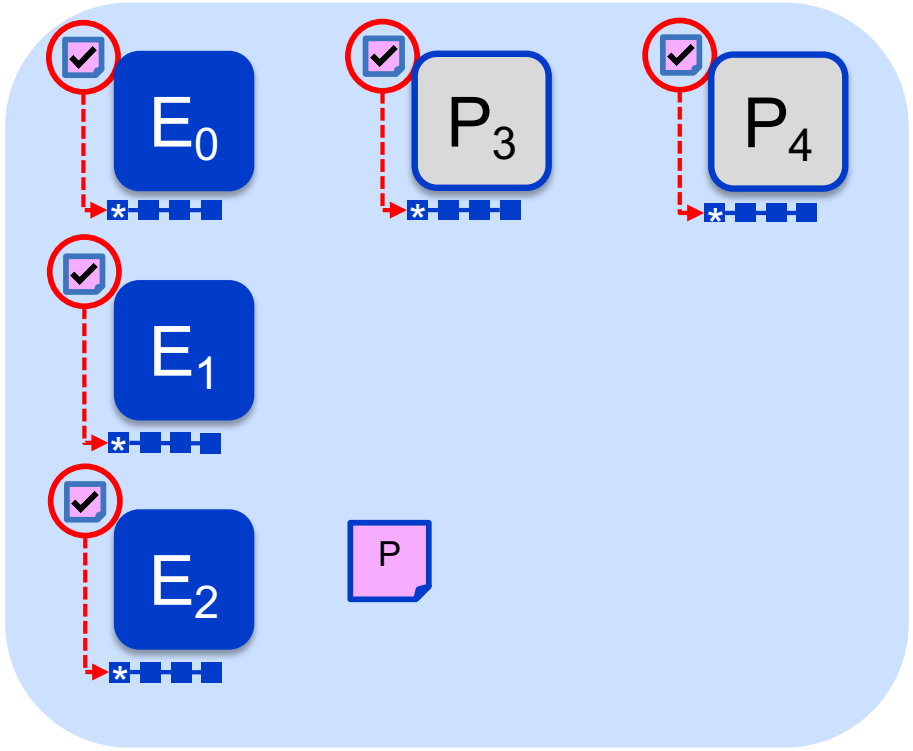
Validated transactions are applied to the world state and retained on the ledger

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

Key:



Sample transaction: Step 6/7 – Validate Transaction



Hyperledger Fabric Network

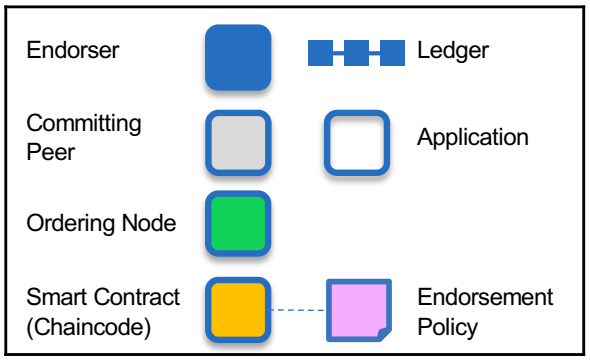
Committing peers validate transactions

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

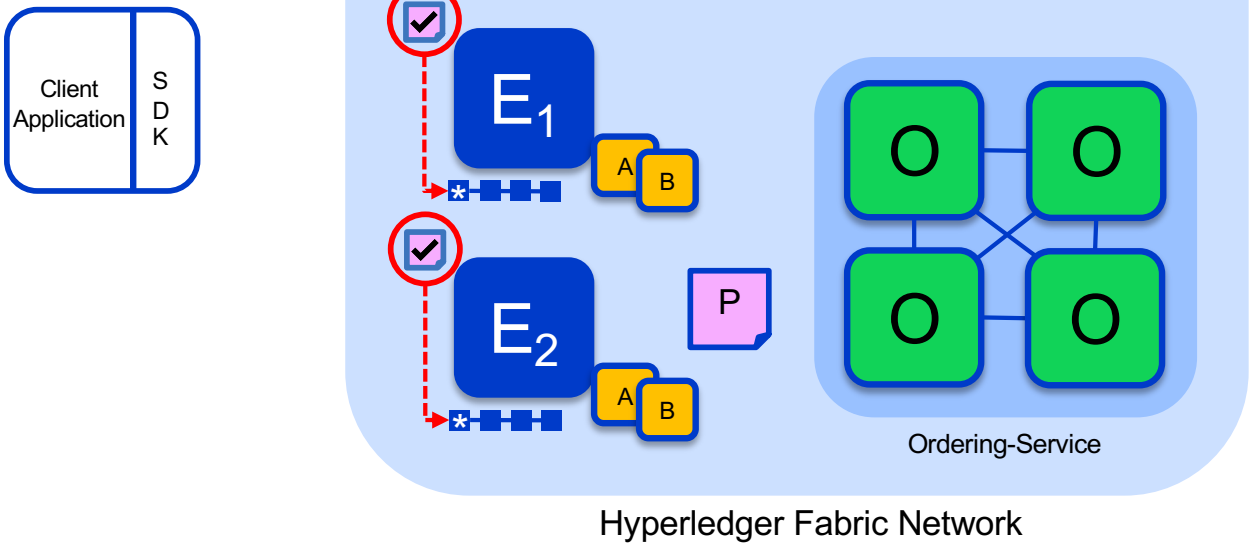
Validated transactions are applied to the world state and retained on the ledger

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

Key:



Sample transaction: Step 6/7 – Validate Transaction



Hyperledger Fabric Network

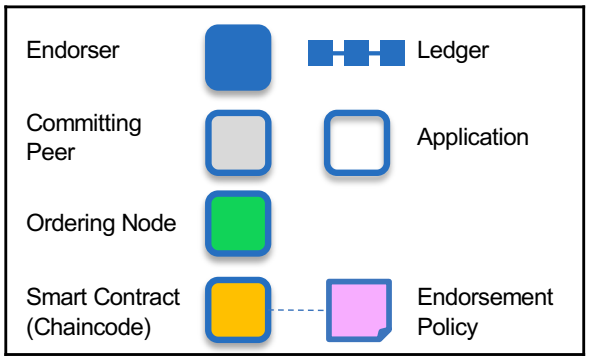
Committing peers validate transactions

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

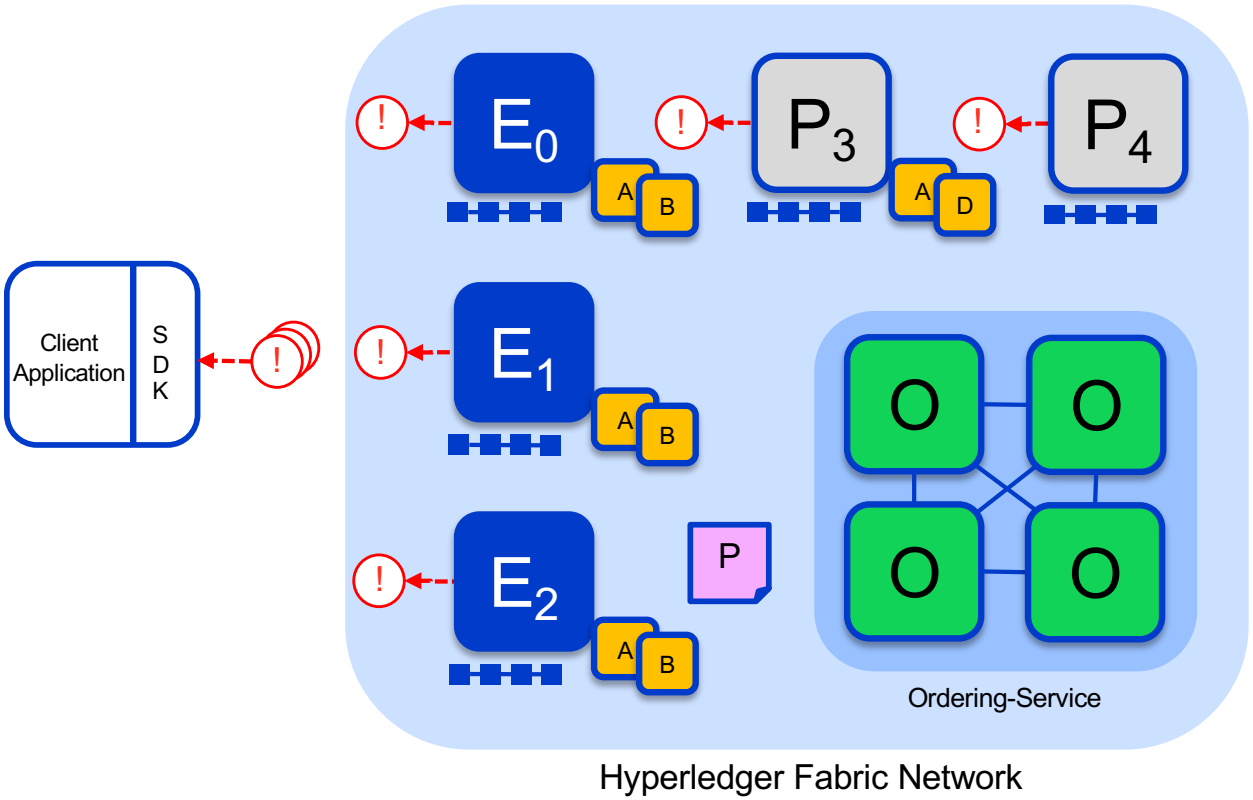
Validated transactions are applied to the world state and retained on the ledger

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

Key:



Sample transaction: Step 7/7 – Notify Transaction

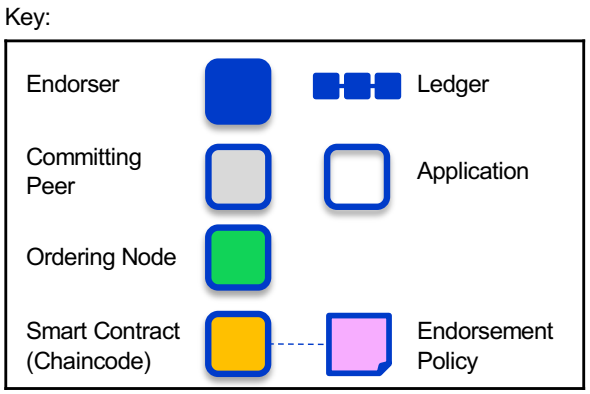


Hyperledger Fabric Network

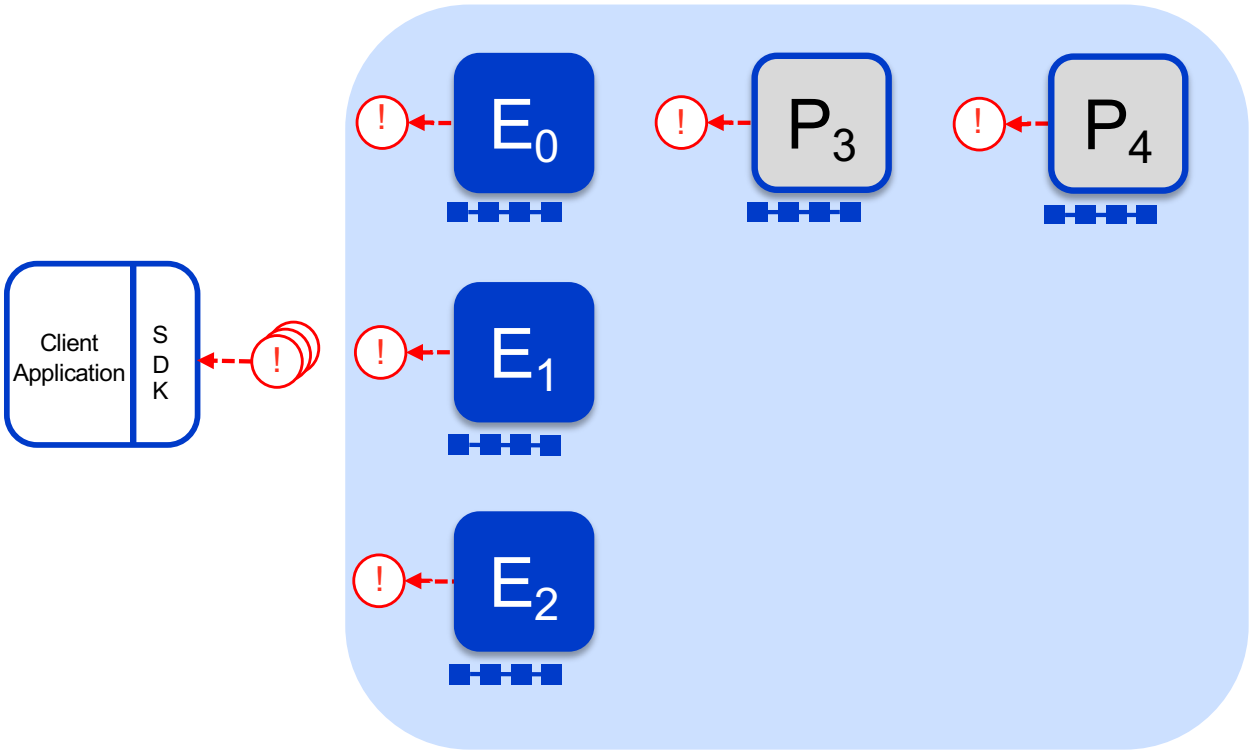
Committing peers notify applications

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



Sample transaction: Step 7/7 – Notify Transaction



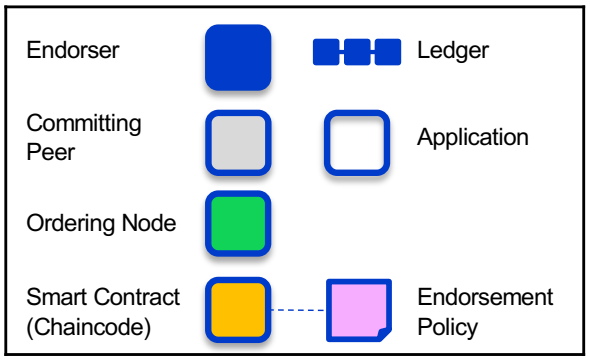
Hyperledger Fabric Network

Committing peers notify applications

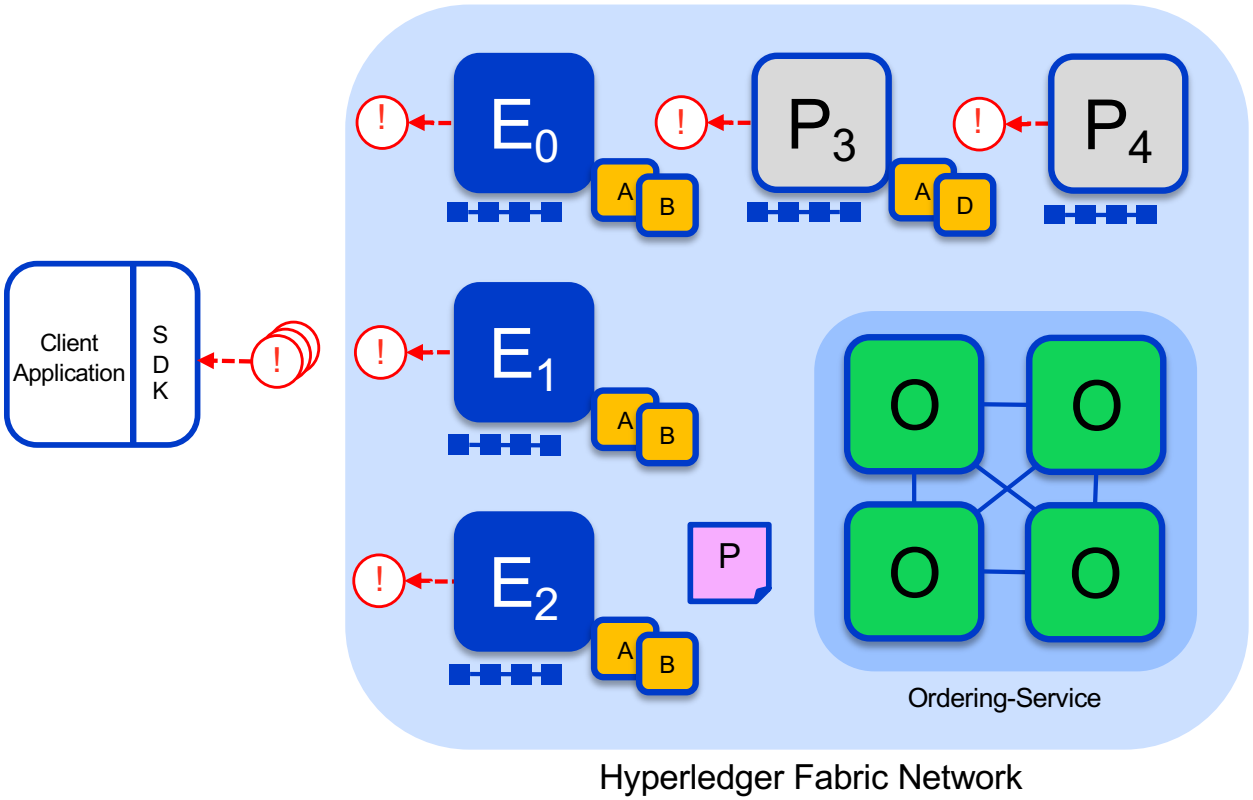
Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger

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

Key:



Sample transaction: Step 7/7 – Notify Transaction

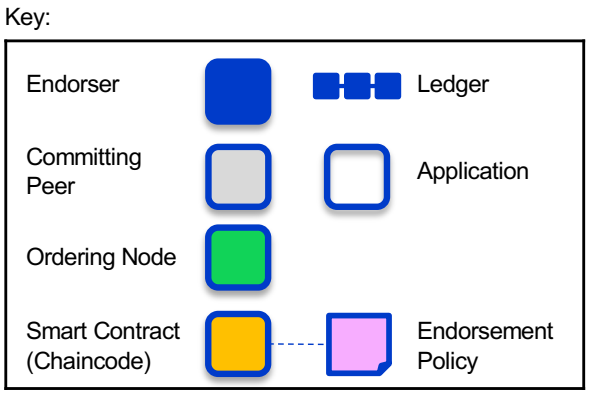


Hyperledger Fabric Network

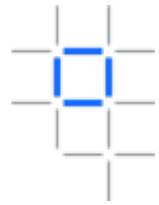
Committing peers notify applications

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

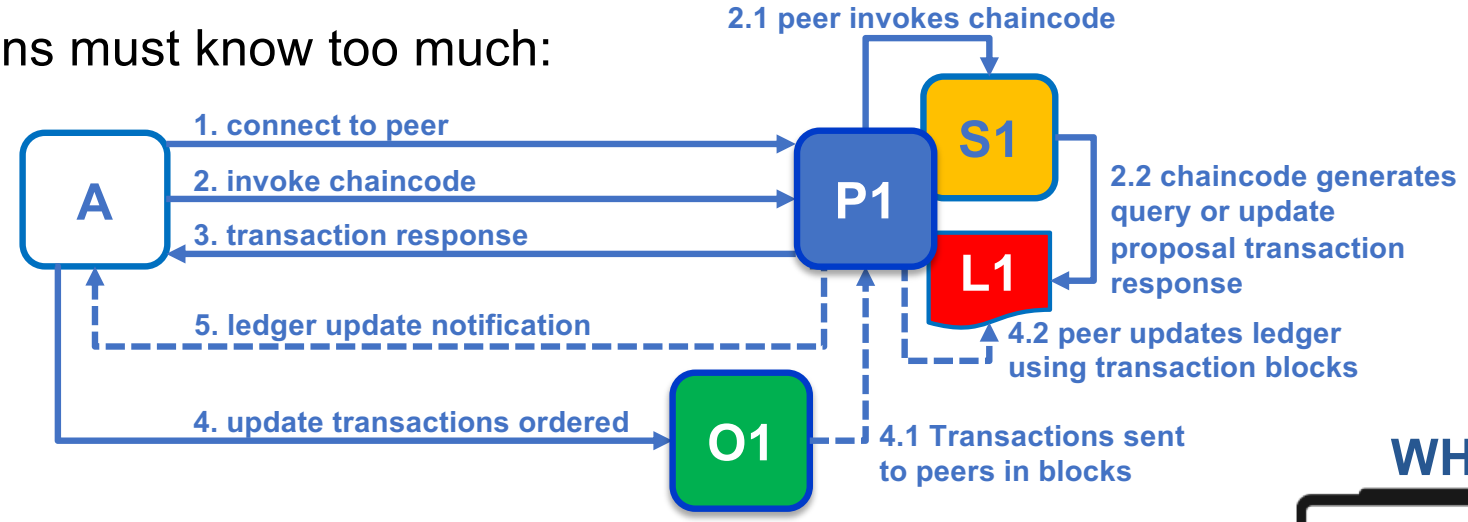
Applications will be notified by each peer to which they are connected

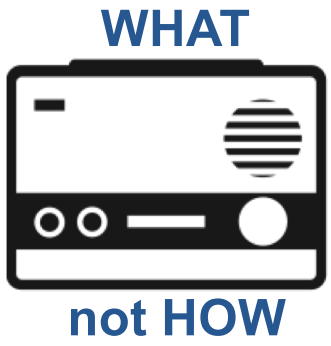


The application programming challenge



Applications must know too much:





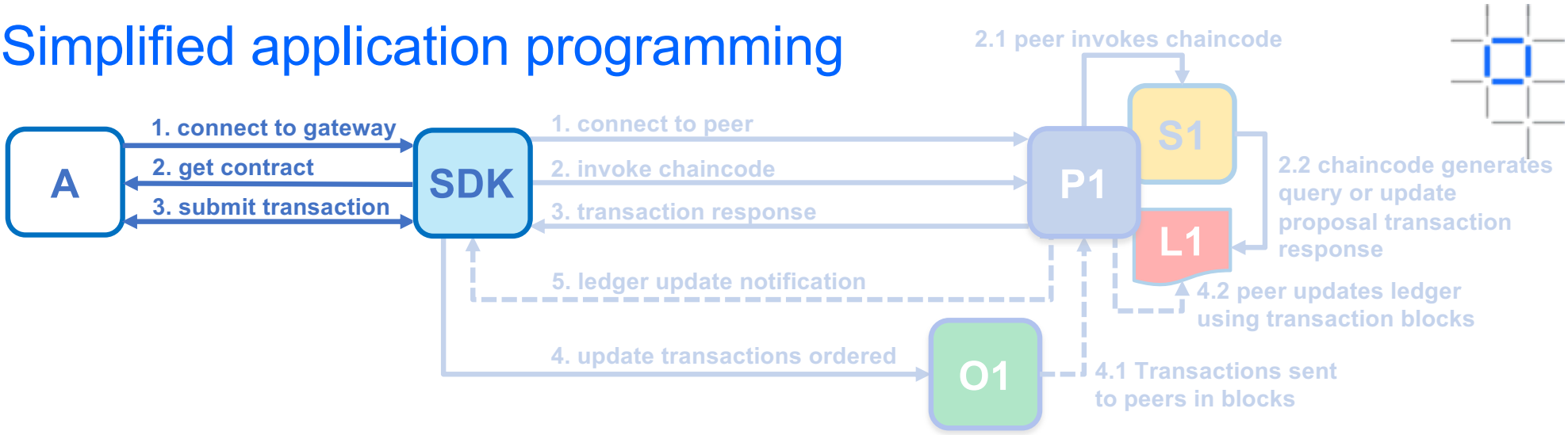
- **WHO** are the endorsing organizations (Endorsement policy)
- **WHERE** are the endorsing organizations (Peers)
- **HOW** to get transactions validated (Ordering)

Simplify for the developer

- Applications should only need to know **WHAT** system does – not **WHO, WHERE** or **HOW** it does it
- Allow developers to concentrate on their **business logic**

IBM

Simplified application programming



Enhanced SDK to simplify the developer experience

- Application has a **client connection policy**
- **Contract** metaphor for **business logic**
- Simply **submit** a transaction to the network
- Built-in **fault tolerance** (policy driven)
- **Pluggable handlers** for queries and events
- Sensible **defaults** for most cases

https://jira.hyperledger.org/browse/FABN-692

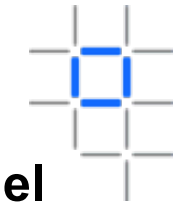
```
// A gateway defines the peers used to access Fabric networks
const gateway = new Gateway();
try {
  // Connect to gateway using application specified parameters
  await gateway.connect(connectionProfile, connectionOptions);

  // Access PaperNet network
  const network = await gateway.getNetwork('mychannel');

  // Get addressability to commercial paper contract
  const contract = await network.getContract('papercontract',
```

Simplified Smart Contract programming

- A '**contract**' can now be implemented as a **class** - the methods in the class are the transaction functions
- Each transaction proposal from the client will automatically invoke that named transaction function - **no dispatcher required**
- Arguments are **automatically extracted** from the transaction proposal and **passed** into the transaction function
- Client applications can **discover** the list of transaction functions available for a deployed contract
- Contracts are deployed in **exactly the same way** as Fabric chaincode, using the **same APIs or CLIs**
- Support for Javascript/Typescript and Java; future support expected for Go

```
// Fabric smart contract classes
const { Contract, Context } = require('fabric-contract-api');

/**
 * Define commercial paper smart contract by extending Fabric Contract class
 */
class CommercialPaperContract extends Contract {

    constructor() {
        // Unique namespace when multiple contracts per chaincode file
        super('org.papernet.commercialpaper');
    }

    async instantiate(ctx) {
        // Instantiate to perform any setup of the ledger that might be required.
    }

    async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime, faceValue) {
        // Issue commercial paper
    }

    async buy(ctx, issuer, paperNumber, currentOwner, newOwner, price, purchaseDateTime) {
        // Buy commercial paper
    }

    async redeem(ctx, issuer, paperNumber, redeemingOwner, redeemDateTime) {
        // Redeem commercial paper
    }
}
```

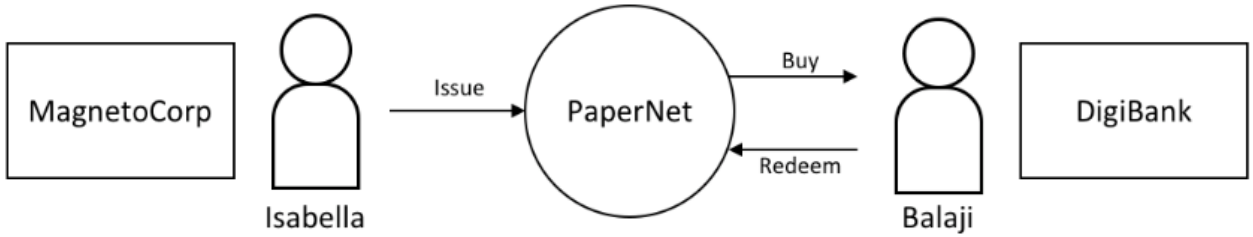
New Commercial Paper tutorial



Built using the new Hyperledger Fabric v1.4 simplified programming model

Description and examples of:

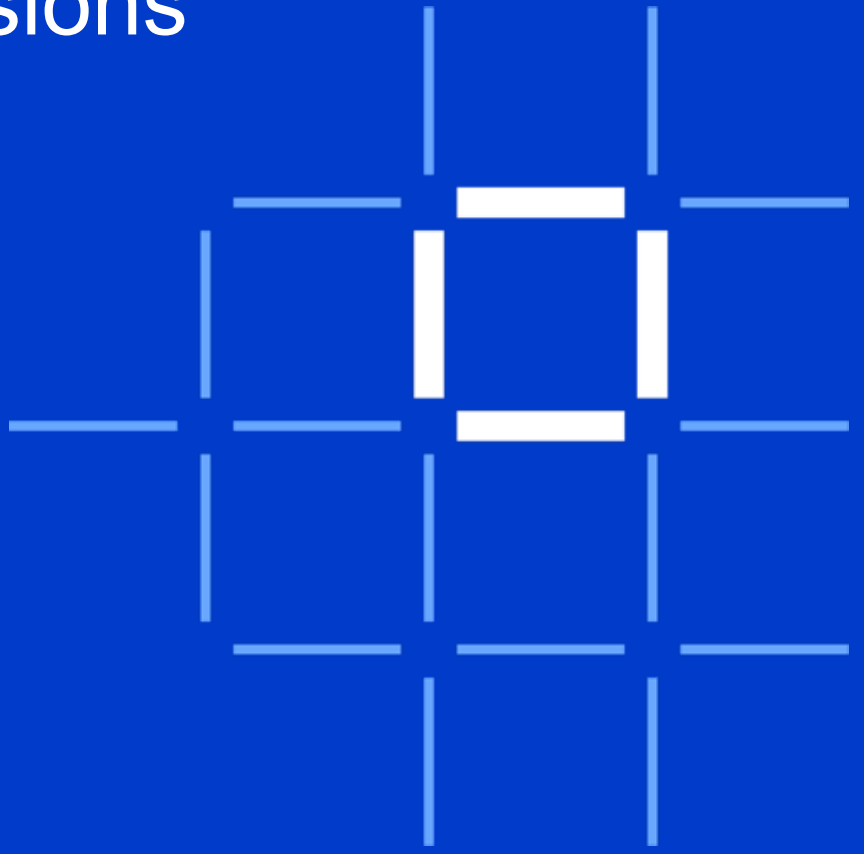
- Smart Contract
- Client application
- Wallet and user identity management
- Install and run



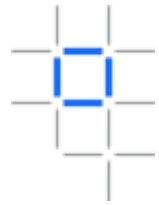
IBM

Hyperledger Fabric Versions

- ☐ **v1.4.1** : Released April 2019
- ☐ **v1.4** : Released January 2019
- ☐ **v1.3** : Released October 2018
- ☐ **v1.2** : Released June 2018
- ☑ **v1.1** : Released March 2018

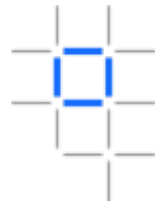


Fabric 1.1 new features overview



- **Rolling Upgrade Support**
  - Allows components of the blockchain network to be updated independently
- **Channel Events**
  - Peers now deliver events per channel
- **Couch DB Indexes**
  - Indexes can be packaged with chaincode to improve query performance
- **Chaincode - Node.js**
  - Node.js chaincode support in base Fabric for 1.1
- **Client Application – Common connection profile**
  - Includes all blockchain network end-points and connection parameters
- **Application Level Encryption**
  - Fabric includes an encryption library for use by chaincode
- **Transport Layer Security (TLS)**
  - All communications within a Hyperledger Fabric network can be secured using mutual TLS
- **Attribute Based Access Control**
  - Include identity attributes in enrollment certificates for chaincode

Rolling Upgrade Support



Allows components of the blockchain network to be updated independently

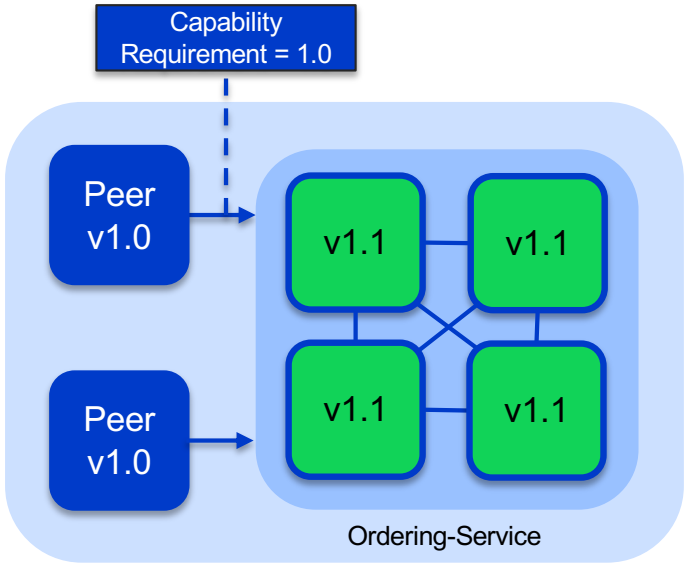
New "**capability requirements**" configuration in the channel determines the feature version level

Separately configure:

- **Channel** – Capabilities of orderers and peers defined in the channel group
- **Orderer** – Capabilities of orderers only
- **Application** – Capabilities of peers only

Steps to upgrade a network (can be done in parallel):

- Orderers, Peers, Fabric-CAs
- Client SDK
- Enable 1.1 capability requirement
- Kafka
- Chaincode



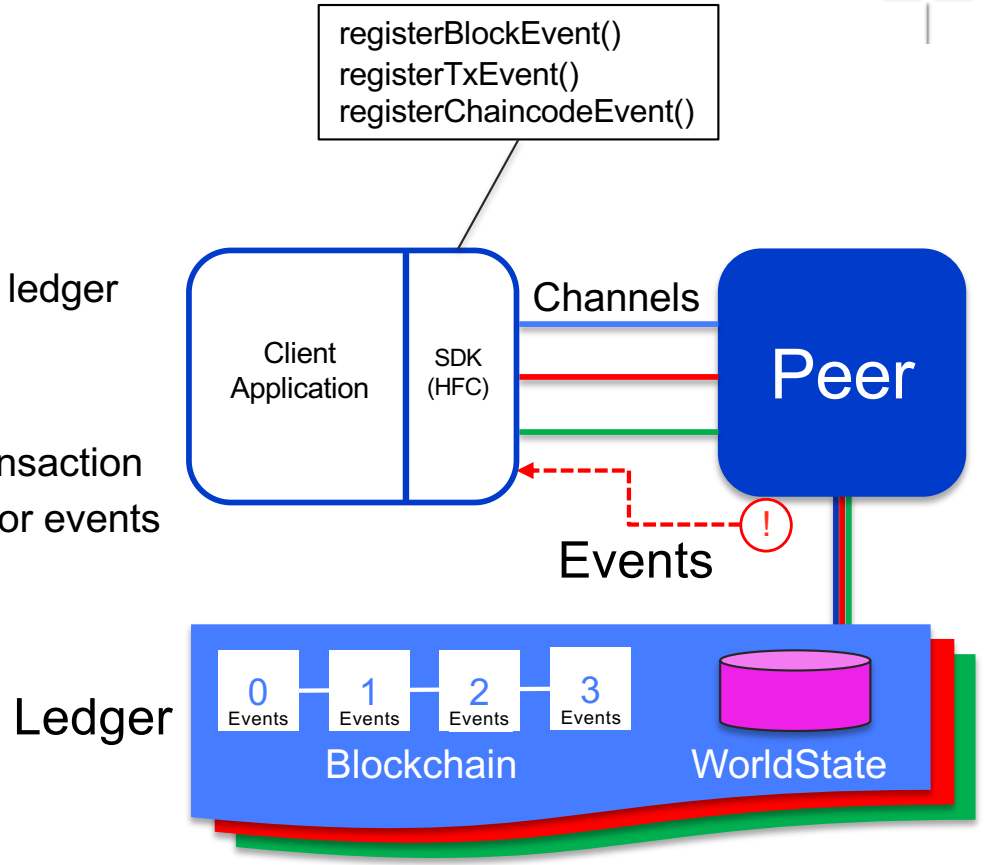
Hyperledger Fabric Network

https://jira.hyperledger.org/browse/FAB-5556

Channel Events

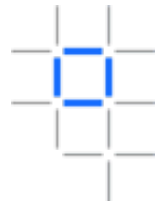
Peers now deliver events per channel

- Rearchitected in Fabric 1.1
- Events captured during endorsement and stored in the ledger
- Peers can replay past events
- Applications can request old events to catch-up
- Events can include the entire block or be filtered by transaction
- Channel MSP defines which applications can register for events
- Applications register listeners for:
  - Block creation events
  - Transaction events
  - Custom chaincode events



https://jira.hyperledger.org/browse/FAB-6911 + https://jira.hyperledger.org/browse/FAB-6911

CouchDB Indexes



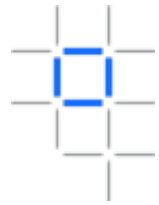
Indexes can be packaged with chaincode to improve query performance

- Indexes packaged alongside chaincode in the following directory:
  - */META-INF/statedb/couchdb/indexes*
- Each index must be defined separately in its own .json file
- When chaincode is first installed, the index is deployed to CouchDB on instantiation
- Once the index is deployed it will automatically be used by CouchDB

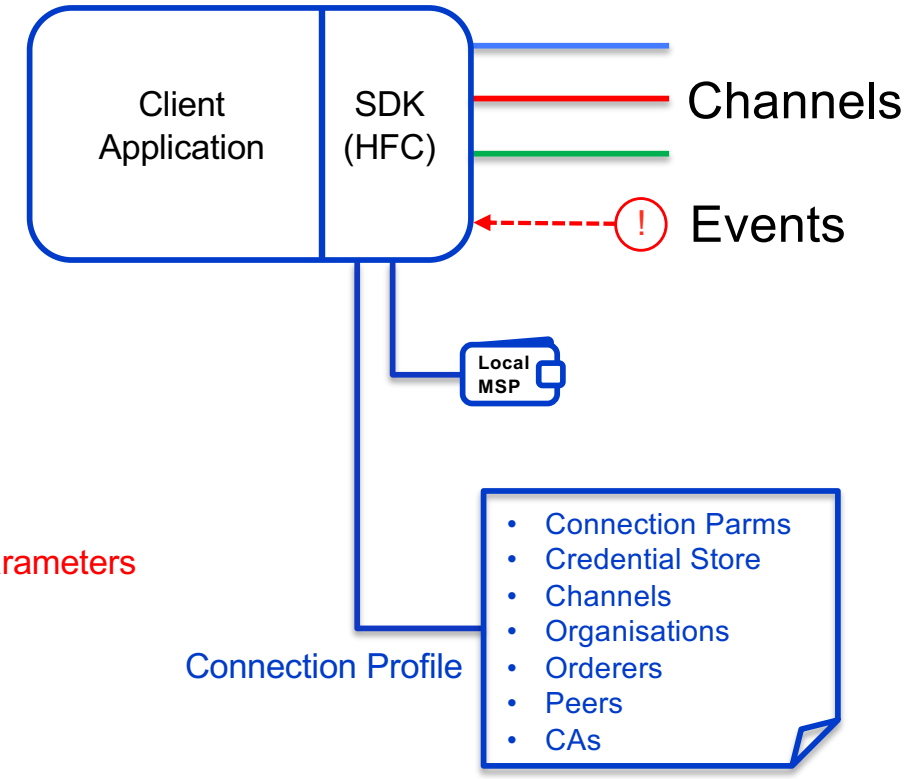
```
{
    "index": {
        "fields": ["docType", "owner"]
    },
    "ddoc": "indexOwnerDoc",
    "name": "indexOwner",
    "type": "json"
}
```


https://jira.hyperledger.org/browse/FAB-3067

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
- Supported Languages
  - Node.js
  - Java
  - Golang
  - Python (future)
- Common Connection Profile (New in Fabric 1.1)
  - Includes all blockchain network end-points and connection parameters
  - Simplifies coding in the Fabric-SDK
  - Used to create a Business Network Card in Composer
  - Can be coded in yaml or JSON



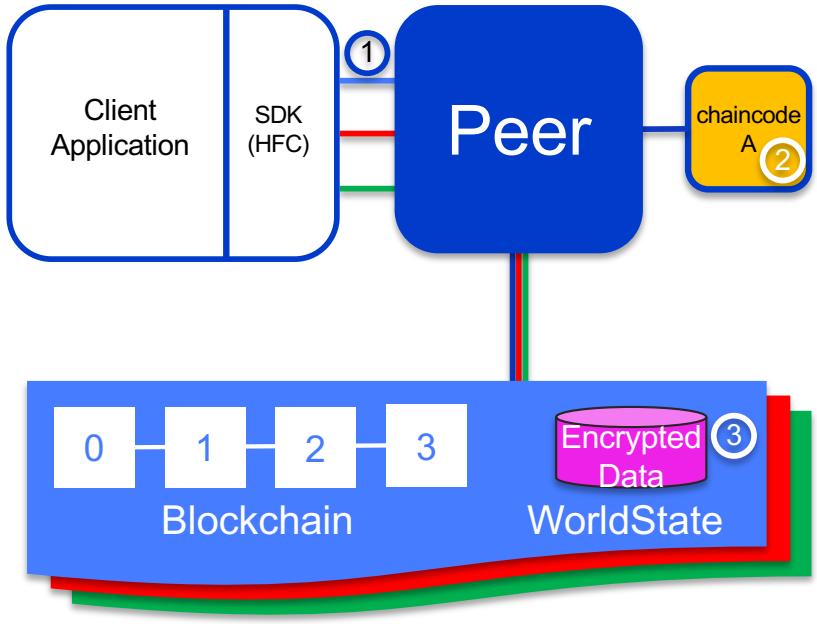
https://jira.hyperledger.org/browse/FAB-5363

Application Level Encryption

Encrypt/Decrypt and sign data in chaincode

- Fabric includes an encryption library for use by chaincode
- New chaincode “entities” API provides interface to:
  - Encrypt
  - Decrypt
  - Sign (ECDSA)
  - Verify
- Pass cryptographic key(s) to chaincode via **transient-data** field
- Support for Initialisation Vector (IV) allowing multiple endorsers to calculate same cypher text

1. Pass unencrypted data and keys to endorser
2. Chaincode encrypts data to put in worldstate
3. Encrypted data stored in worldstate

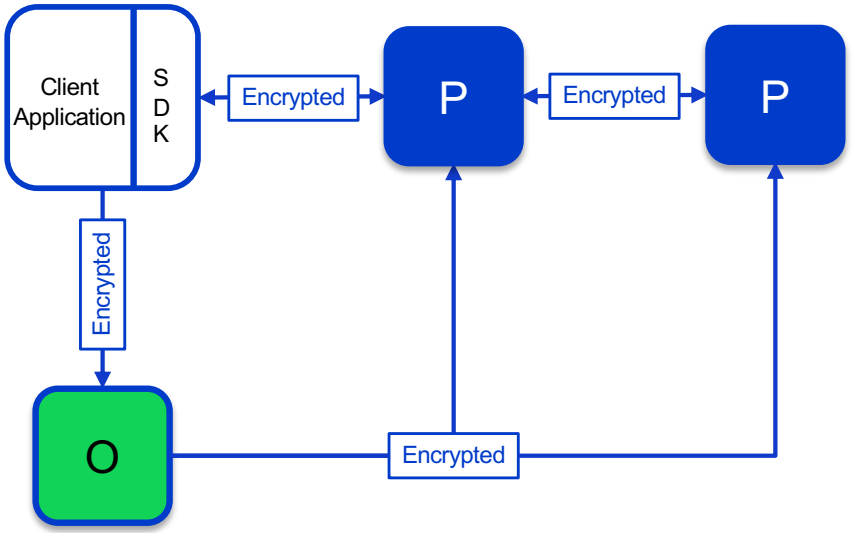


https://jira.hyperledger.org/browse/FAB-830

Transport Layer Security

All communications within a Hyperledger Fabric network can be secured using TLS

- Peers and Orderers are both TLS Servers and TLS Clients
- Applications and commands are TLS Clients
- Fabric 1.0.x support for TLS Server authentication
- Fabric 1.1 supports mutual TLS (client authentication)



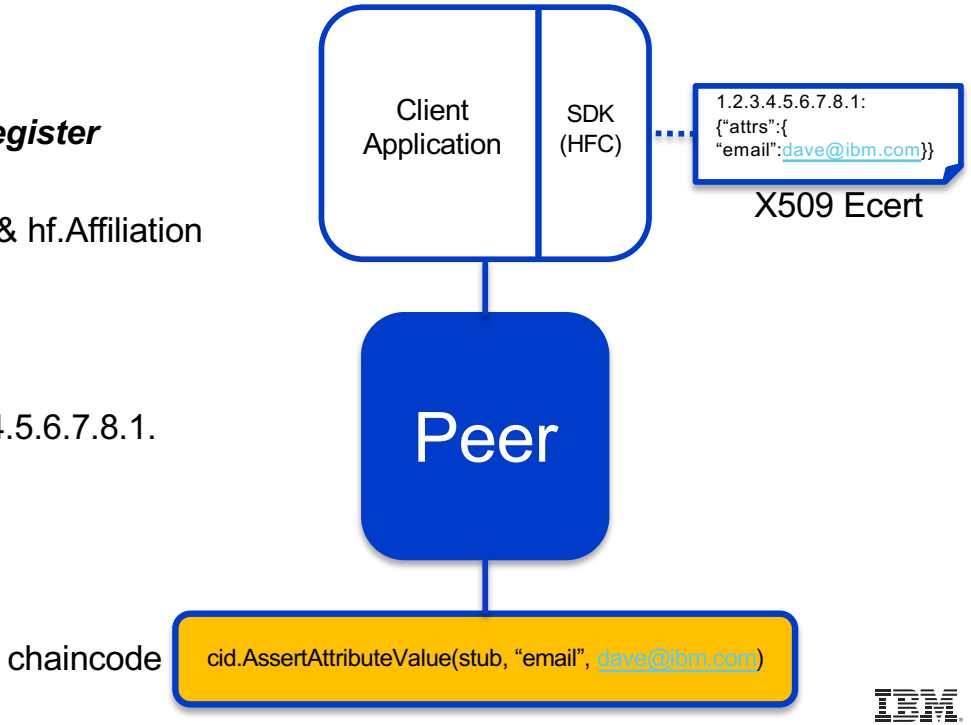
https://jira.hyperledger.org/browse/FAB-6715

Attribute-Based Access Control

Include identity attributes in enrollment certificates for chaincode

- Include attributes in **X509 enrollment certificates** (Ecerts)
- Defined as name/value pairs: email=dave@ibm.com
- Define mandatory and optional attributes with ***fabric-ca-client register***
- Specify attribute values with ***fabric-ca-client enroll***
- Ecerts automatically include attributes: hf.EnrollmentID, hf.Type & hf.Affiliation
- API provided by Client Identity chaincode Library:
  - cid.GetAttributeValue(stub, "attr1")
  - cid.AssertAttributeValue(stub, "myapp.admin", "true")
- Stored as an extension in the Ecert with an ASN.1 OID of 1.2.3.4.5.6.7.8.1.

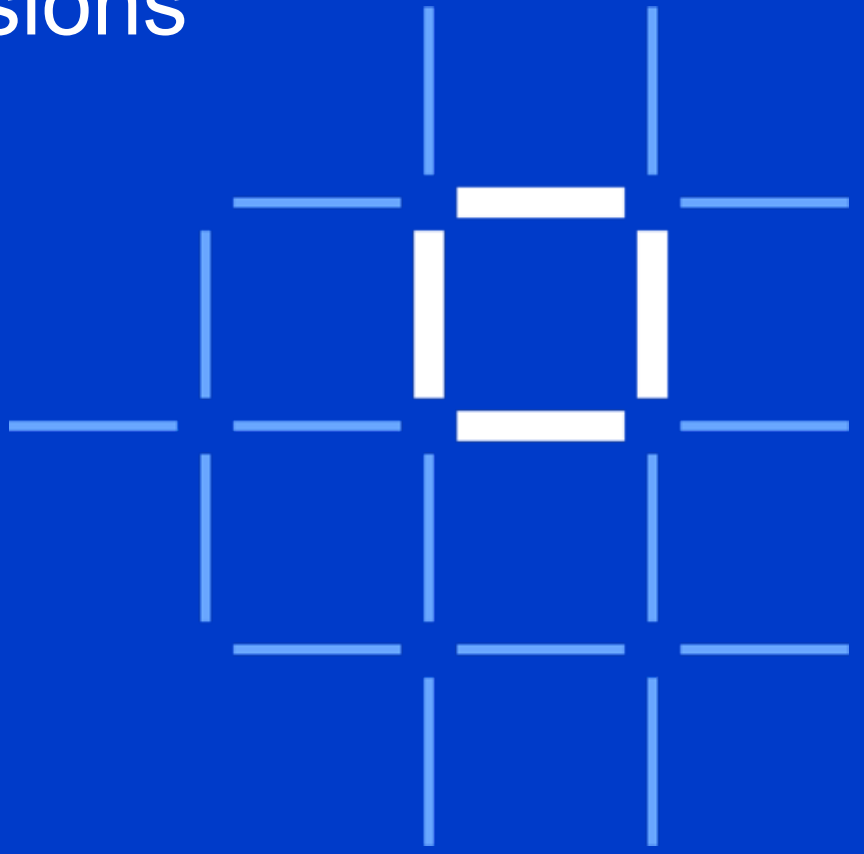
```
1.2.3.4.5.6.7.8.1:
    {"attrs":{"attr1":"val1"}}
```



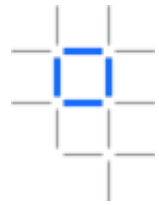
https://jira.hyperledger.org/browse/FAB-5346 + https://jira.hyperledger.org/browse/FAB-6089

Hyperledger Fabric Versions

- ☐ **v1.4.1** : Released April 2019
- ☐ **v1.4** : Released January 2019
- ☐ **v1.3** : Released October 2018
- ☑ **v1.2** : Released June 2018
- ☐ **v1.1** : Released March 2018



Private Data Collections

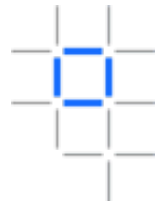


Allows data to be private to only a set of authorized peers

Fabric 1.0 & 1.1	Fabric 1.2
• 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)
	• Complements existing Fabric channel architecture
	• Policy defines which peers have private data

https://jira.hyperledger.org/browse/FAB-8718

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.

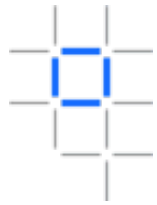
2. **Hashes of private data** included in transaction proposal for evidence and validation.
    1. Peers not in the collection policy and the Orderer only have hashes.

3. **Peers maintain** both a public worldstate and a private data store

4. **Private data** held in a transient store between endorsement and validation.

https://jira.hyperledger.org/browse/FAB-1151

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

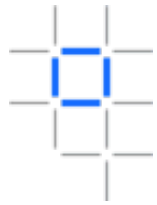
Collection: Marble Private Details
- Private Write Set
- Price

Policy: Org1

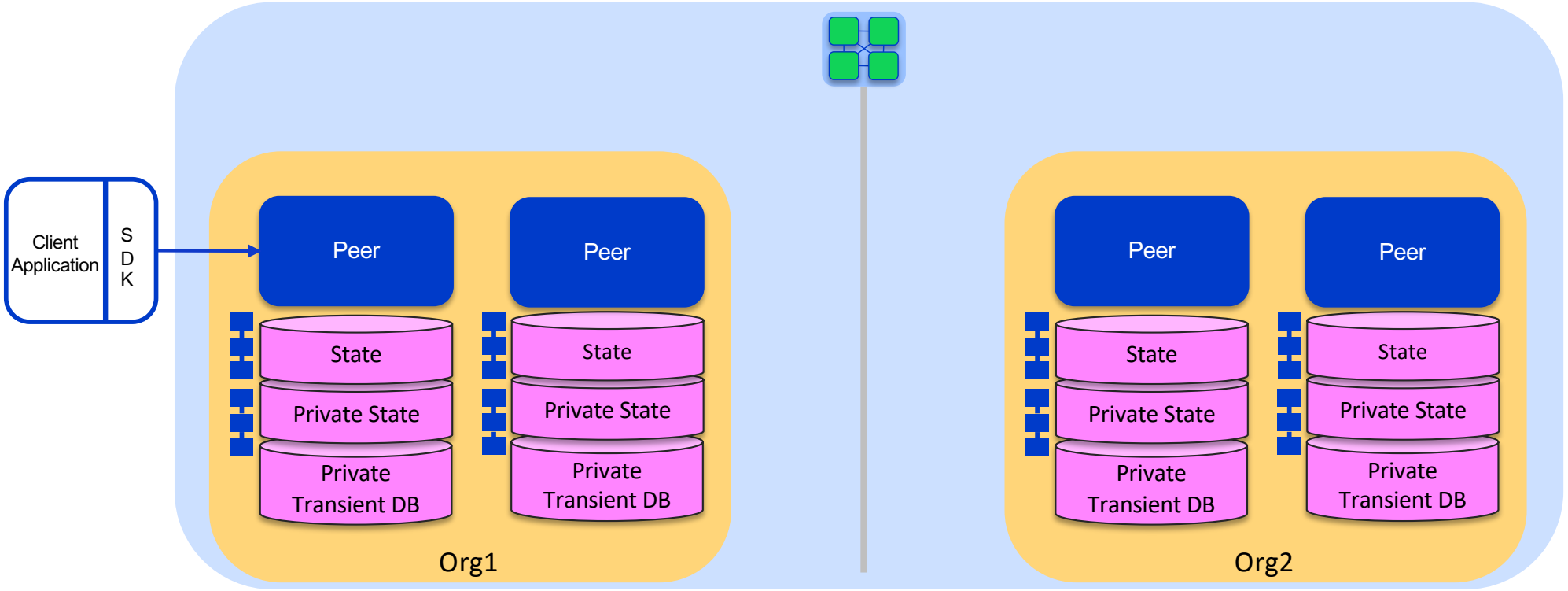
"requiredPeerCount": 1,
"maxPeerCount": 1,
"blockToLive":3

Collection: Marbles Private Details
- Private data for subset of channel peers
- Goes to subset of peers only

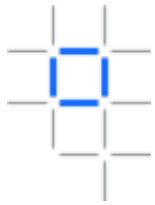
Step 1: Propose Transaction



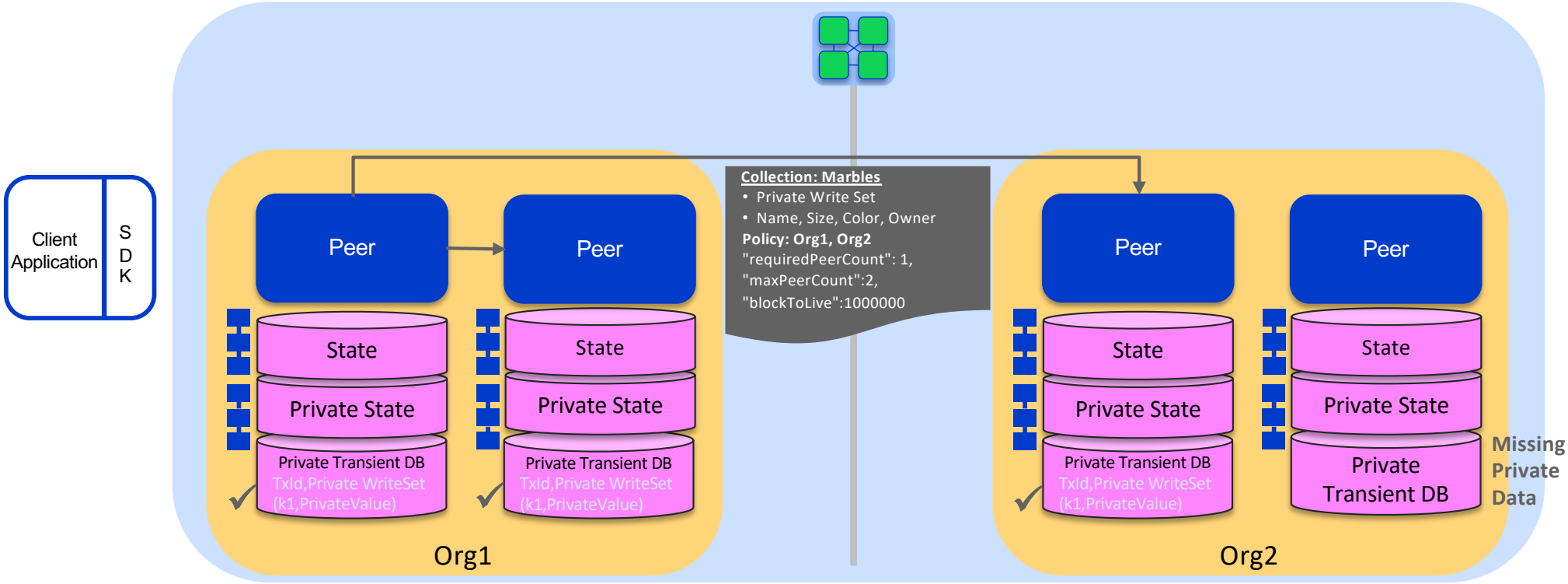
Client sends proposal to endorsing peer(s)



Step 2a: Execute Proposal and Distribute 1st Collection

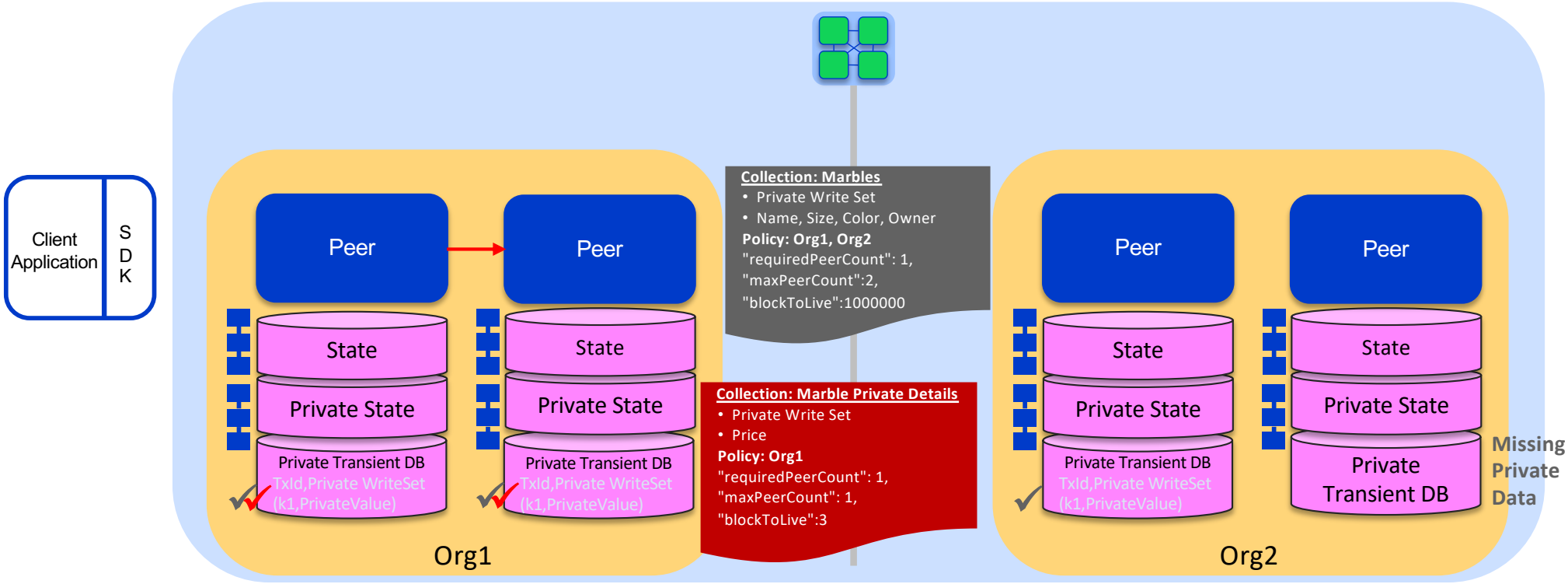


Endorsing peer simulates transaction and distributes **marbles collection** data based on policy



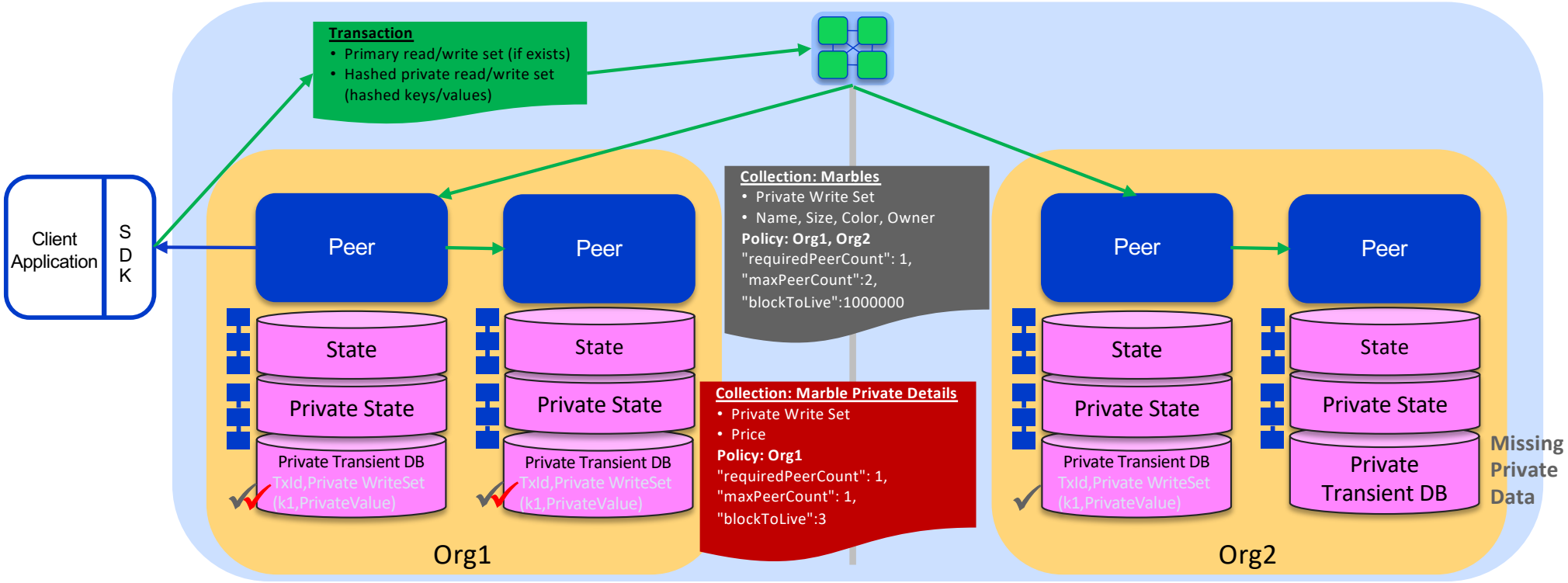
Step 2b: Distribute 2nd Collection

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



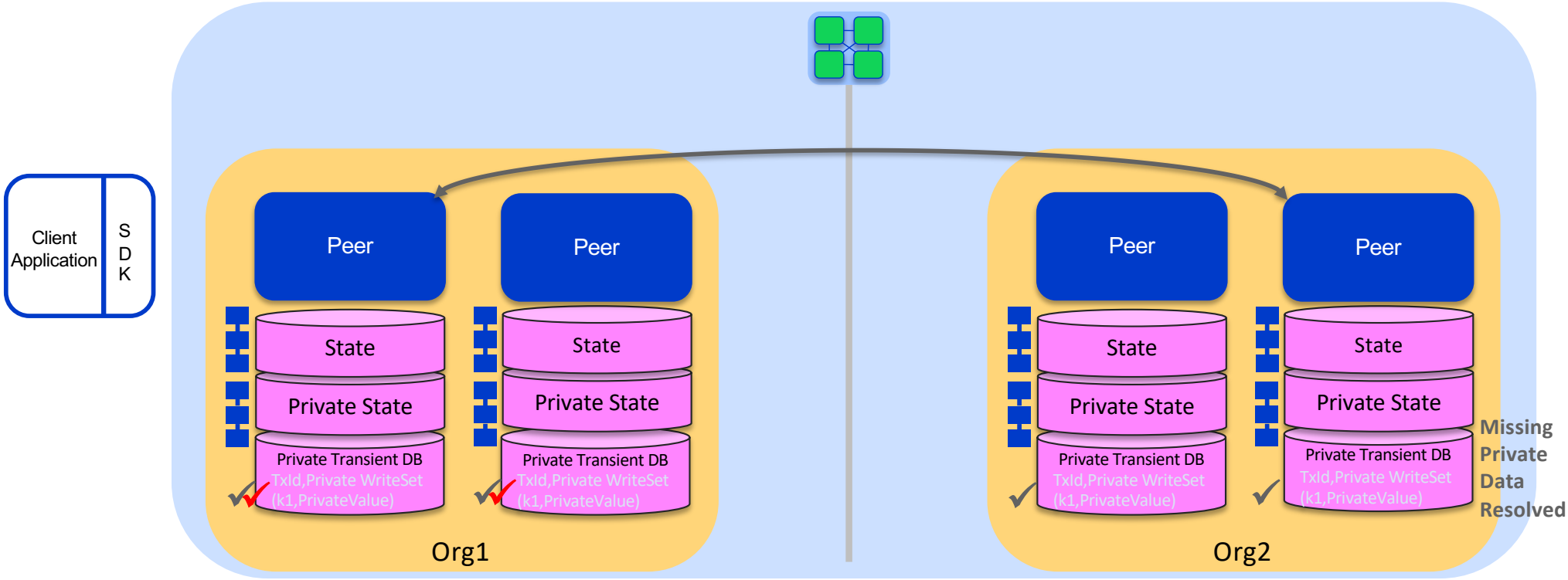
Step 3: Proposal Response / Order / Deliver

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



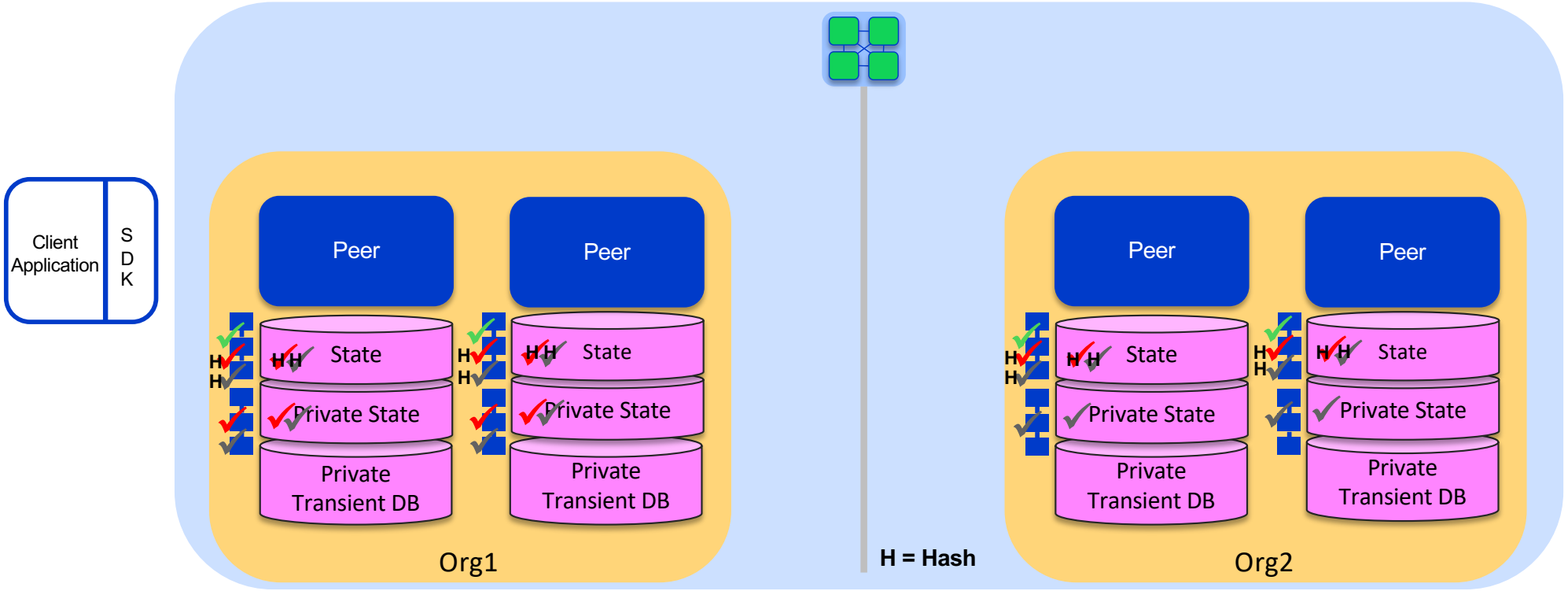
Step 4: Validate Transaction

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

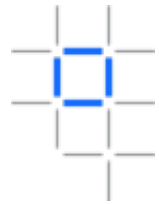


Step 5: Commit

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.



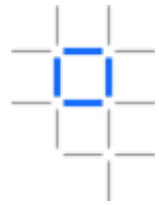
Channels vs Private Data



- Organizations that do not participate in a channel have no knowledge of the transactions in that channel
- Organizations in a channel that are not members of a private data collection know that these “insiders” are transacting, they just do not know the data involved
- Your business use case requirements may favor one approach over the other
- Ordering service receives the transactions in a channel
- Ordering service does not receive private data
  - It receives only the hashes of the private data
  - Private data itself is shared among authorized peers for peer-to-peer gossip

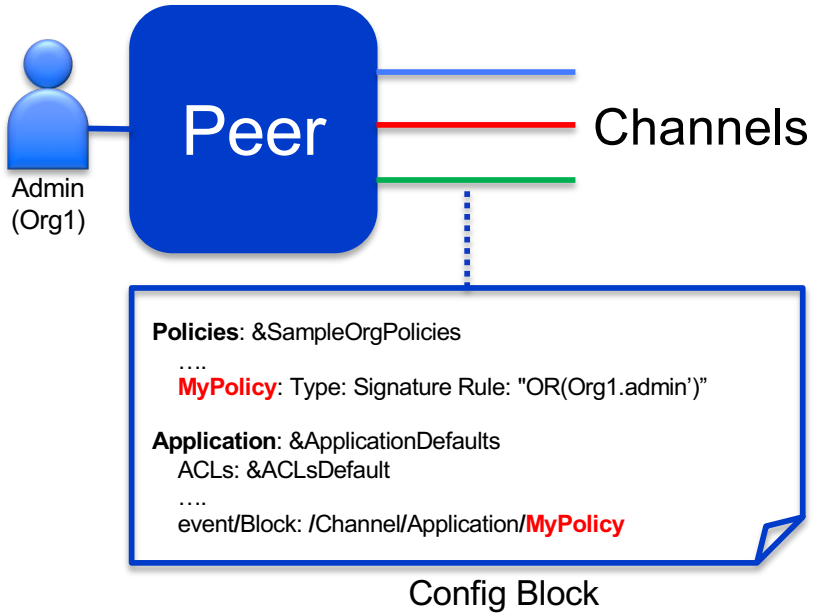
https://jira.hyperledger.org/browse/FAB-1151

ACL mechanism per channel



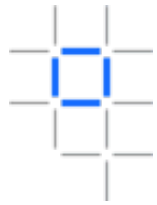
Support policy based access control for peer functions per channel

- Access control defined for channel and peer resources:
  - User / System chaincode
  - Events stream
- Policies specify identities and include defaults for:
  - Readers
  - Writers
  - Admins
- Policies can be either:
  - Signature : Specific user type in org
  - ImplicitMeta : “All/Any/Majority” signature types
- Custom policies can be configured for ACLs



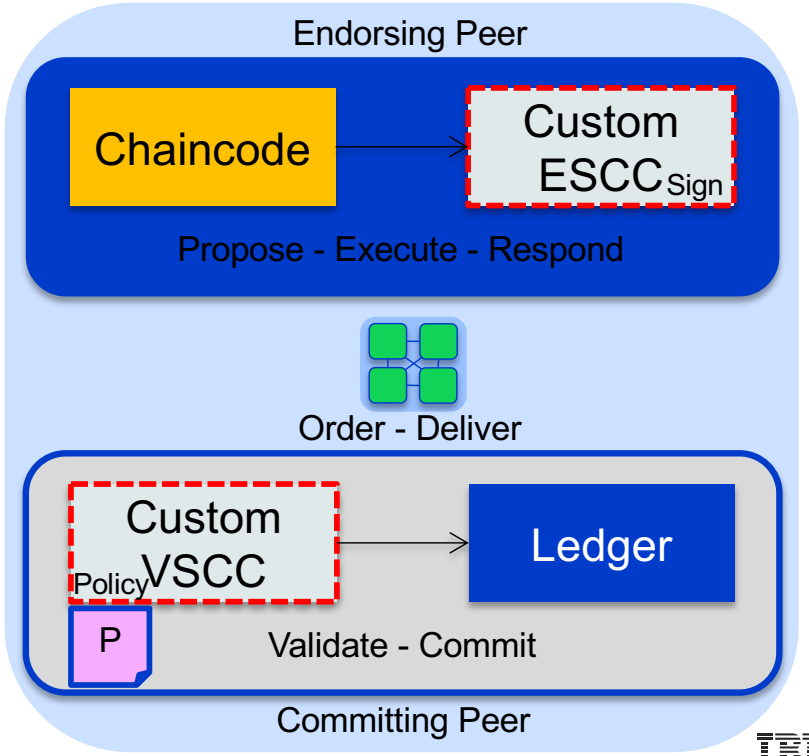
https://jira.hyperledger.org/browse/FAB-8727

Pluggable endorsement and validation



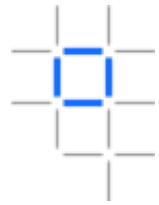
Support for custom transaction endorsement and validation plugins

- Supports alternative transaction models for: State based endorsement, UTXO etc
- No need to recompile peer, **core.yaml** specifies additional golang plugins
- Support for custom:
  - **ESCC** : Endorsement System Chaincode
  - **VSCC** : Validation System Chaincode
  - **QSCC** : Query System Chaincode
  - **CSCC** : Configuration System Chaincode
  - **LSCC** : Lifecycle System Chaincode
- Chaincode associated with custom ESCC and VSCC at instantiation



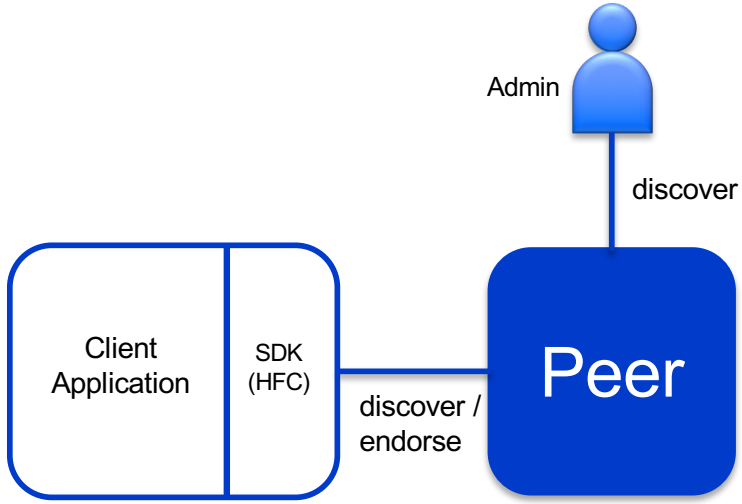
https://jira.hyperledger.org/browse/FAB-8779

Service Discovery



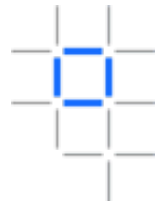
Applications can dynamically query peers to discover network service information

- Network metadata is shared between peers over GOSSIP
- Peers dynamically compute the following:
  - Configuration : MSP for all orgs in a channel
  - Peers : Peers that have joined a channel
  - Endorsers : Endorses for a specific channel/chaincode
- SDK sends dynamic query to peer to establish service connection information (including: endorsement policy, peers endpoints, TLS, CA and orderer endpoints).
- Administrator uses **discover** CLI to discover service information



https://jira.hyperledger.org/browse/FAB-8779

Further Information



Private Data Collections

- https://hyperledger-fabric.readthedocs.io/en/release-1.2/private-data/private-data.html

ACL mechanism per channel

- https://hyperledger-fabric.readthedocs.io/en/release-1.2/access_control.html

Pluggable endorsement and validation

- https://hyperledger-fabric.readthedocs.io/en/release-1.2/pluggable_endorsement_and_validation.html

Service Discovery

- https://hyperledger-fabric.readthedocs.io/en/release-1.2/discovery-overview.html
- https://hyperledger-fabric.readthedocs.io/en/release-1.2/discovery-cli.html

New tutorials

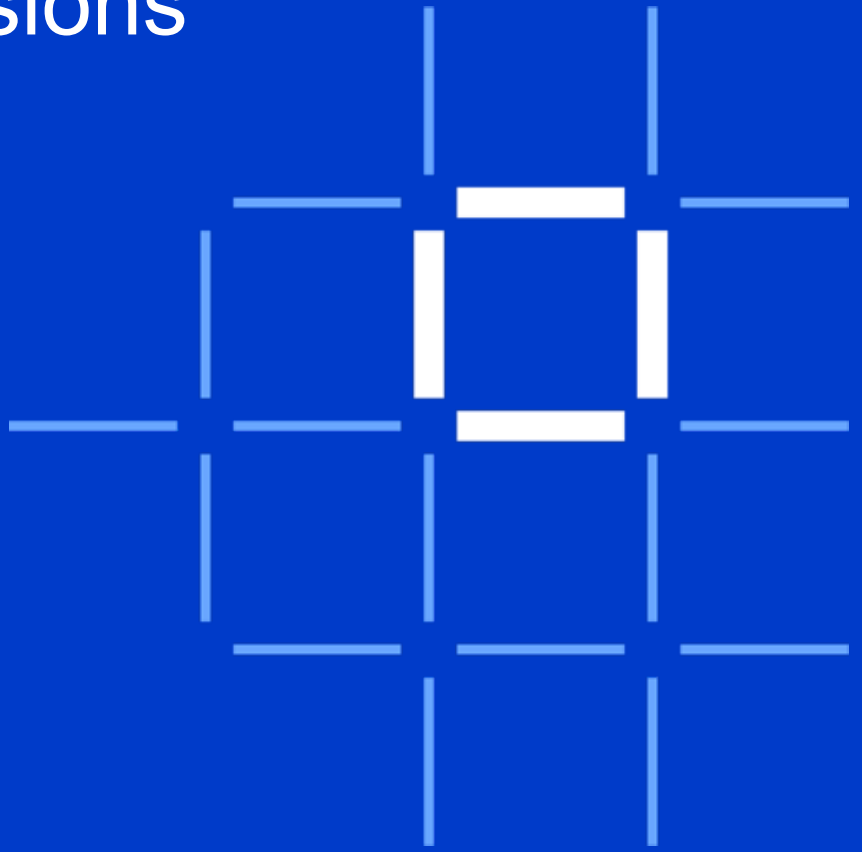
- https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatsnew.html#new-tutorials

New documentation concepts

- https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatsnew.html#new-conceptual-documentation

Hyperledger Fabric Versions

- ☐ **v1.4.1** : Released April 2019
- ☐ **v1.4** : Released January 2019
- ☑ **v1.3** : Released October 2018
- ☐ **v1.2** : Released June 2018
- ☐ **v1.1** : Released March 2018

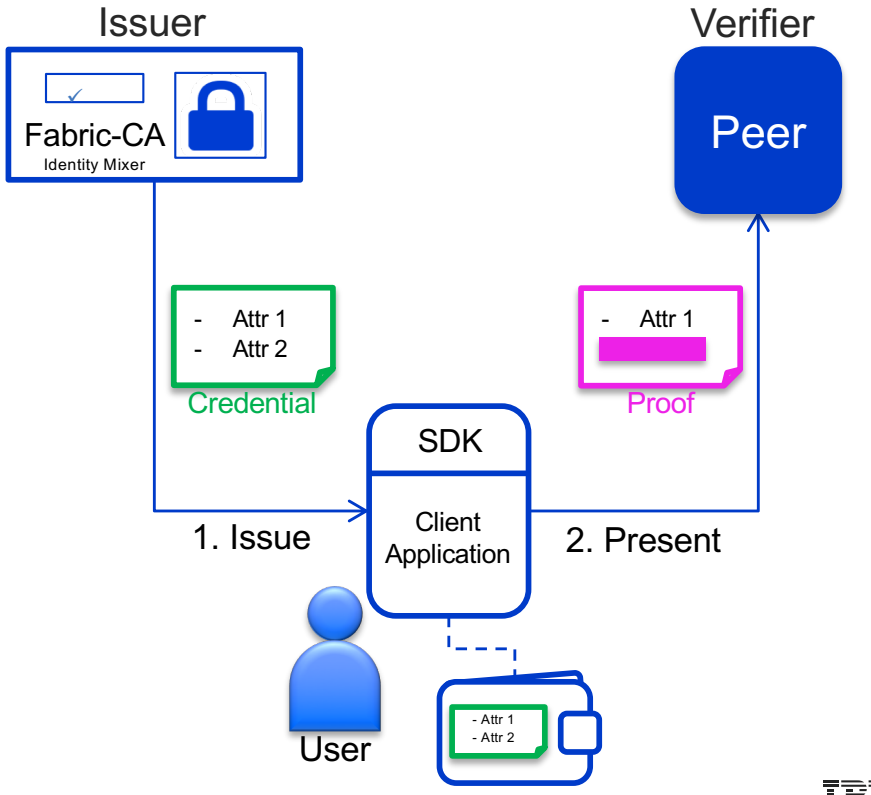


Identity Mixer

Fabric includes Identity Mixer for anonymity/unlinkability of a user's identity

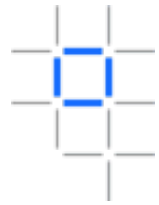
Support for a user's identity to be hidden but verifiable, and selective disclosure of user attributes through zero-knowledge-proofs

- **Issuer:** Issues user's identity in the form of an identity mixer *credential*. Includes all attributes associated with the user.
- **User:** Generates *proofs* from their *credential* with selectively disclosed attributes using zero-knowledge-proof. These *proofs* do not disclose the user's true identity and are unlinkable as each proof is different.
- **Verifier:** Verifies the *proof* based on the public certificate of the issuer.



https://jira.hyperledger.org/browse/FAB-10120

Notes – Identity Mixer



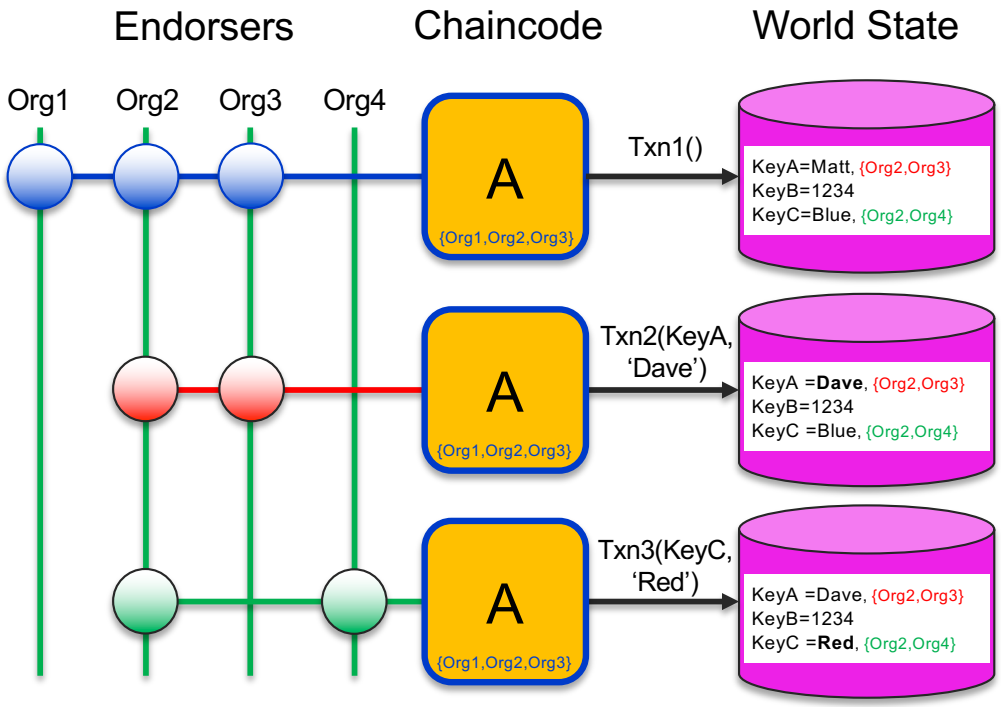
- Identity Mixer technology integrated into:
  - Fabric-CA for the issuer
  - Fabric Java SDK for the user
  - MSP for the verifier
- A credential is analogous to an enrolment certificate
- A proof is analogous to a transaction certificate
- Supported attributes in v1.3:
  - OU (affiliation)
  - Role
- Neither the issuer or verifier can link a proof to a user
- Future support includes:
  - Additional SDKs
  - Custom attributes
  - Additional roles such as a peer creating a proof
  - Revocation of credentials
  - Auditor
  - Predicates
- **idemixgen** is a new tool for development/test to issue credentials
- Identity mixer video: https://www.youtube.com/watch?v=ZVItp_LqGgw

Key-level endorsement policies

Set endorsement policies for individual key-level state changes

Fabric supports endorsement policies for individual key-level changes as well as the default chaincode level

- Default endorsement policy applies to all state within the chaincode:
  - **Txn1()** endorsement from {Org1,Org2,Org3}
- Alternative endorsement policies for keys:
  - **Txn2()** endorsement from {Org2,Org3}
  - **Txn3()** endorsement from {Org2,Org4}
- First key-level endorsement change requires endorsement from chaincode level endorsement policy to maintain trust



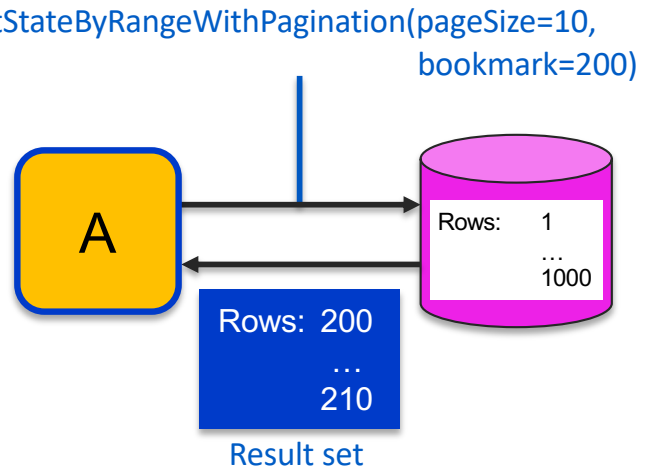
https://jira.hyperledger.org/browse/FAB-8812

CouchDB query support for pagination

Improved query support for performance and scalability

Chaincode specifies the starting point and number of rows to return in a query result set

- Specify *pageSize* for the number of rows to return, and *bookmark* for the location of the first row
- Supported on several shim API calls such as *GetStateByRangeWithPagination()* etc
- Bookmark can be reused so chaincode can iterate queries over a large amount of data



https://jira.hyperledger.org/browse/FAB-2809

Support for Ethereum's Virtual Machine

Enables developers to migrate and create new Ethereum DApps for Fabric

- Burrow is Apache-2.0 license(#)
- Hyperledger Fabric user chaincode wraps the Burrow EVM implementation
- Ethereum based smart contracts deployed via the user chaincode wrapper
- Interact with deployed smart contracts using Fabric CLI and SDK
- Ethereum client support, e.g. Web3.js, Remix IDE, via the Ethereum JSON-RPC API
- Ethereum's Contract Accounts accessible within the EVM chaincode namespace
- Gas limit set arbitrarily high

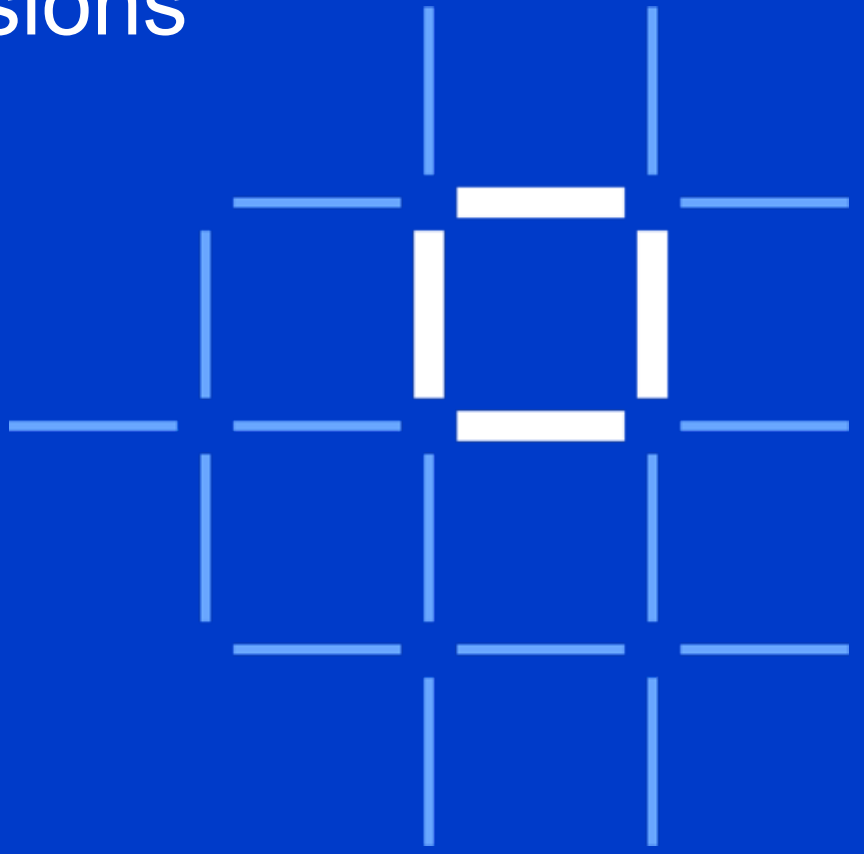
Blog by Swetha Repakula: https://www.hyperledger.org/blog/2018/10/26/hyperledger-fabric-now-supports-ethereum
Technical Documentation: https://github.com/hyperledger/fabric-chaincode-evm/blob/master/examples/EVM_Smart_Contracts.md
Design: https://docs.google.com/document/d/1xZfdtFiIFvHI7UZAze2xbm5hhgaDHGRBMusozKfpOck/edit#heading=h.xf6b8qykmhxw

(#) Ethereum's VM is LGPL-3.0

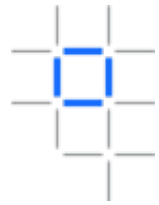
https://jira.hyperledger.org/browse/FAB-10273 and https://jira.hyperledger.org/browse/FAB-8078

Hyperledger Fabric Versions

- ☐ **v1.4.1** : Released April 2019
- ☑ **v1.4** : Released January 2019
- ☐ **v1.3** : Released October 2018
- ☐ **v1.2** : Released June 2018
- ☐ **v1.1** : Released March 2018



Long Term Support (LTS)



Existing release policy

- Provide bug fix (patch) releases for the most recent major or minor release until the next major or minor release is published. This continues to be the default policy.

Hyperledger Fabric v1.4 LTS policy

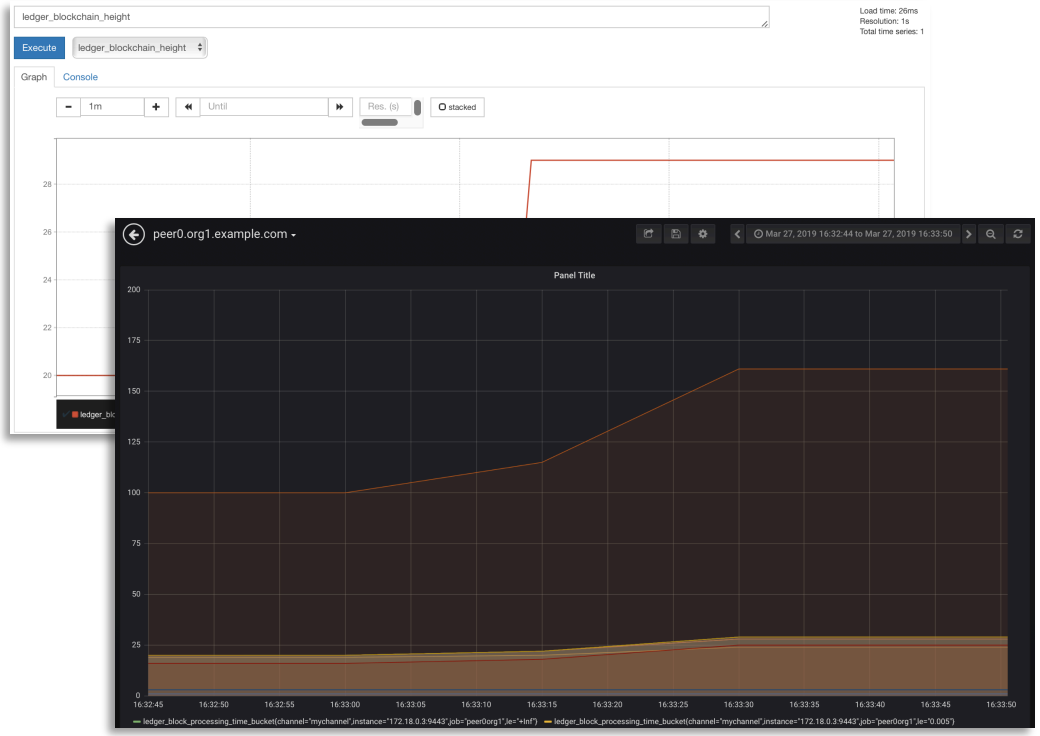
- Hyperledger Fabric maintainers have pledged to provide bug fixes for a period of one year from release date of v1.4. This will likely result in multiple fixes bundled into a series of patch releases (v1.4.1, v1.4.2, …).

Operational Metrics and Logging

A new operations service emits logging, health and metrics data

Both **peers** and **orderers** can be configured to emit the following data:

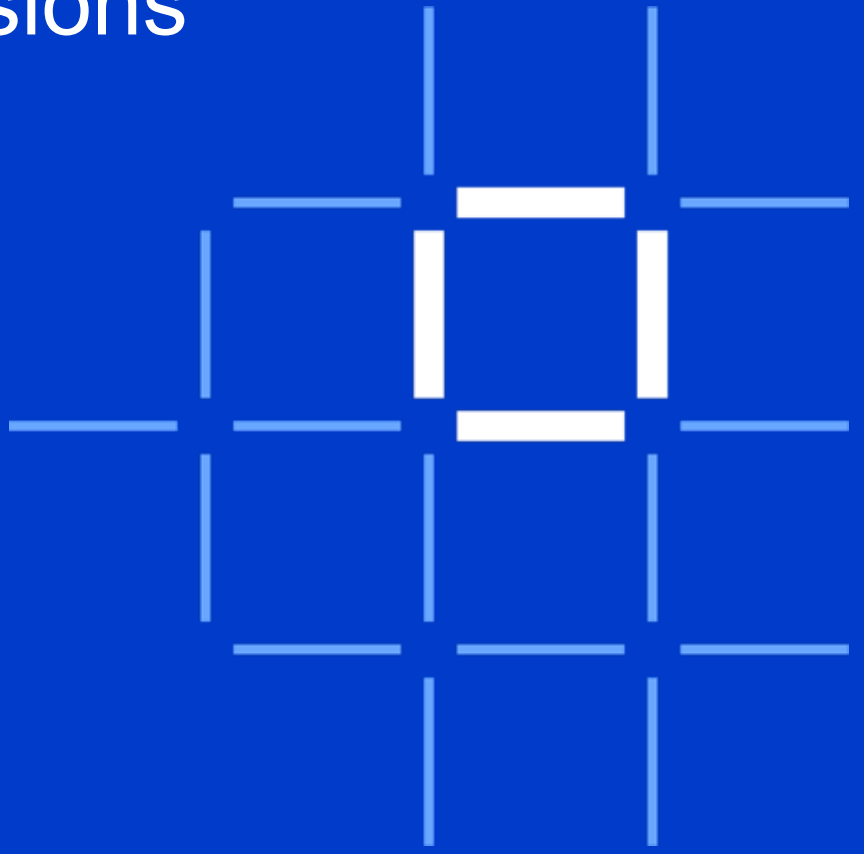
- **Logging info:** Enables operators to dynamically get and set component logging levels.
- **Health info:** Operators and container orchestration services can dynamically check the health of a component.
- **Metrics data:** A range of monitoring data from a component can be collected and aggregated by **Prometheus** or **StatsD**. This can then be visualized in tools such as **Grafana**.



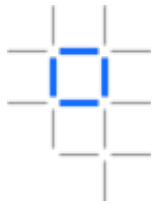
https://jira.hyperledger.org/browse/FAB-3388
https://jira.hyperledger.org/browse/FAB-10851
https://jira.hyperledger.org/browse/FAB-12265

Hyperledger Fabric Versions

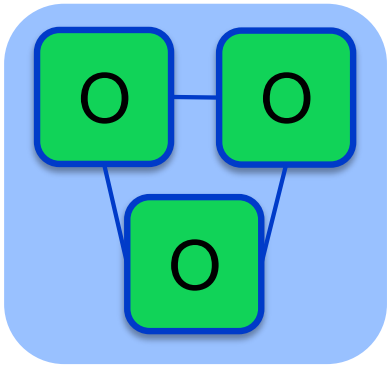
- ☑ **v1.4.1** : Released April 2019
- ☐ **v1.4** : Released January 2019
- ☐ **v1.3** : Released October 2018
- ☐ **v1.2** : Released June 2018
- ☐ **v1.1** : Released March 2018



Ordering Service



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

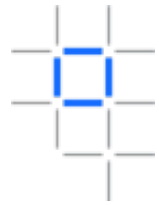


Ordering-Service

Fabric v1.4.1 has the following options for the ordering service:

- Raft (recommended)
  - **Crash fault tolerant** (minimum of 3 nodes)
  - **No additional dependencies** required
  - Can be run as single node for development
  - Works more efficiently than Kafka over geographically diverse networks, allowing a **distributed ordering service**
- Solo
  - Single node for development
- Kafka
  - Crash fault tolerant
  - Requires Kafka and Zookeeper

Raft Consensus

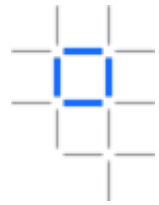


Choose Raft as the consensus protocol for the ordering service

- Raft stands for Reliable,Replicated,Redundant and fault-tolerant
- It is a leader-based protocol
- Hyperledger Fabric embeds the *etcd/raft* binaries from https://coreos.com/etcd/
- Requires no further external dependencies
- Each channel has its own instance of the Raft protocol
- A Raft cluster is the collection of nodes running the Raft protocol
- An orderer node can be part of multiple Raft clusters
- Orderer nodes can be geographically dispersed when using Raft
- All orderer nodes must be part of the main ordering service system channel

https://jira.hyperledger.org/browse/FAB-6135

Raft based orderer node roles



Each orderer node will assume one of three Raft roles:

L **Leader**

- Receives updates (transactions) and sends blocks of ordered transactions to known follower nodes until replaced by a new leader.

F **Follower**

- Follower nodes receive updates (blocks) from the current leader, and leader requests from any candidates.

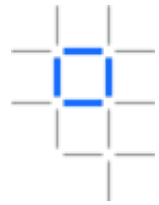
C **Candidate**

- A follower becomes a candidate after an election timeout occurs. A candidate node requests votes for becoming a leader from other followers.

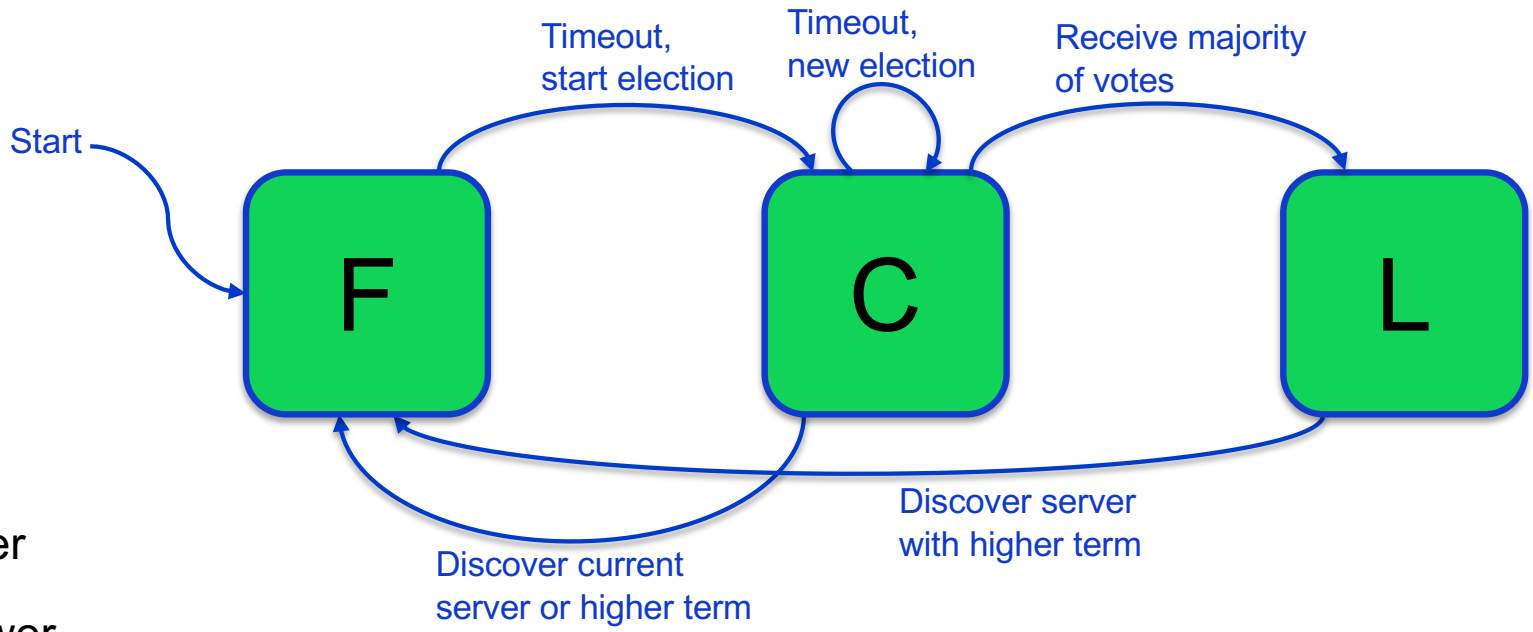
Hyperledger Fabric includes the https://coreos.com/etcd/ Raft implementation allowing orderer nodes to be a leader, follower or candidate.

IBM

Orderer nodes change of role



This shows the events that cause a node to change role within Raft



L Leader

F Follower

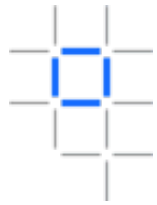
C Candidate

https://raft.github.io/slides/raftuserstudy2013.pdf

IBM

Raft Leader Election

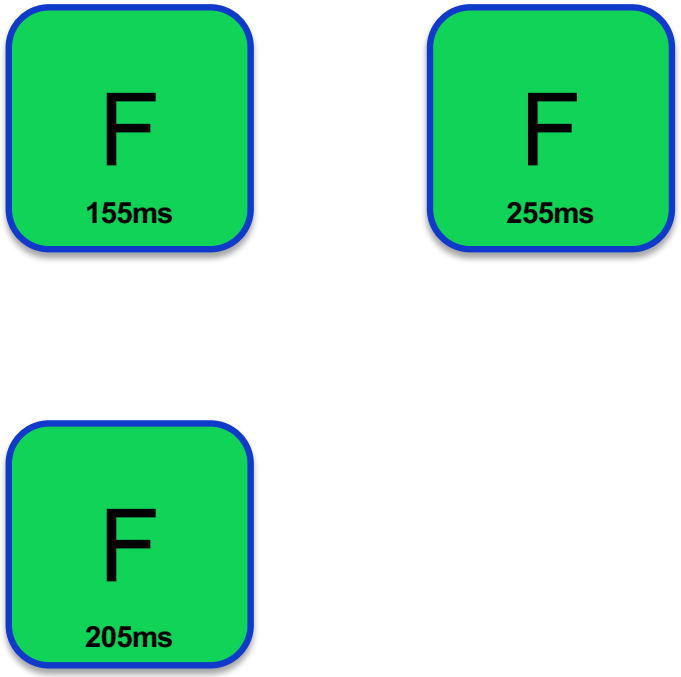
Leader Election: Step 1 – Follower node times out



All nodes start as a **follower** with a randomized election timeout value

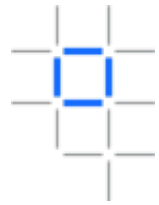
Details:

- Randomized timeout from a fixed interval (e.g.150ms – 300ms)
- First node to timeout becomes a **candidate** node



IBM

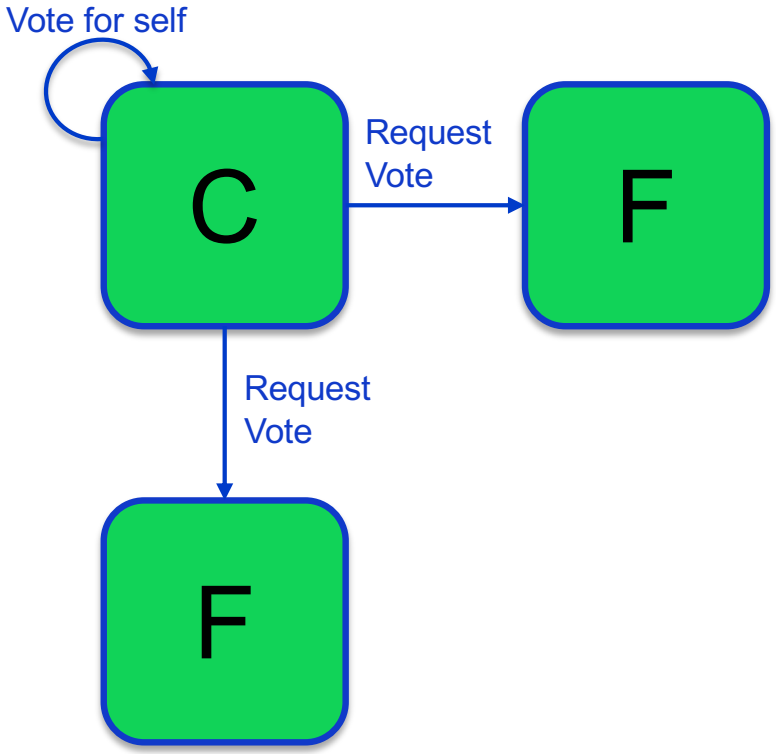
Leader Election: Step 2 – Candidate node requests votes



The **candidate** node requests votes from all known **follower** nodes

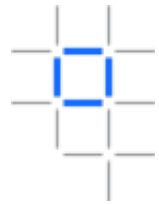
Details:

- The **candidate** node first votes for itself
- RequestVote RPCs are then sent to all **follower** nodes



IBM

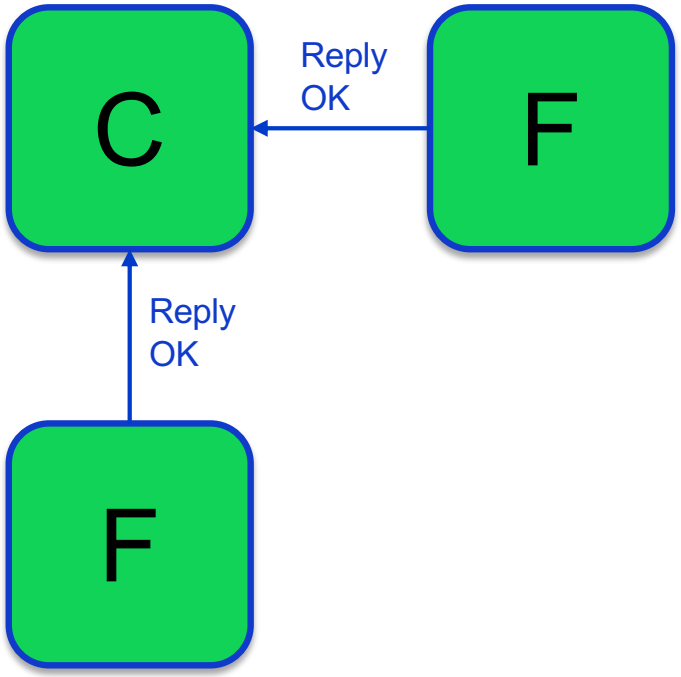
Leader Election: Step 3 – Reply to vote request



Follower nodes reply to the **candidate's** request to vote

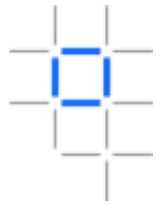
Details:

- If the **follower** hasn't voted before in the current term it replies with OK
- If the **follower** has voted before it replies with an error



IBM

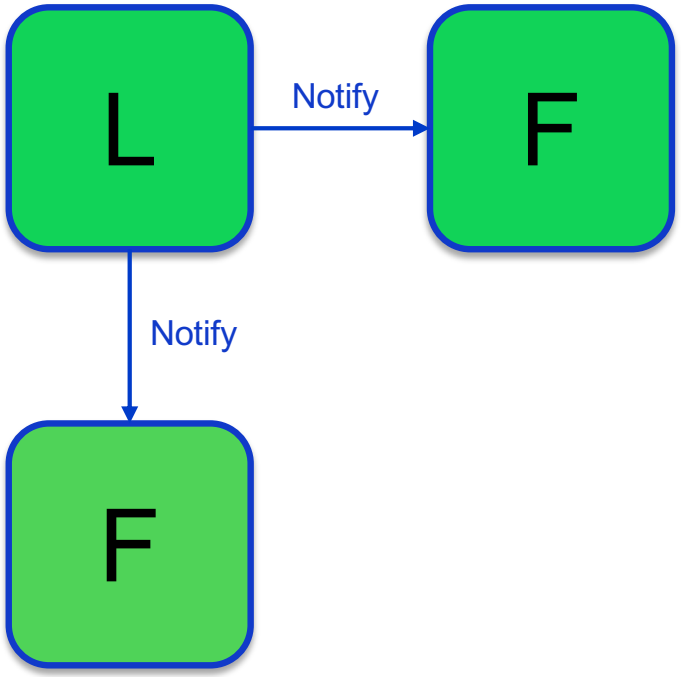
Leader Election: Step 4 – New leader voted



The **candidate** node becomes the **leader**

Details:

- If the **candidate** receives a quorum (majority) of votes then it becomes the new **leader**
- The new **leader** notifies other nodes
- The previous **leader** and any other **candidates** return to being **followers**



IBM

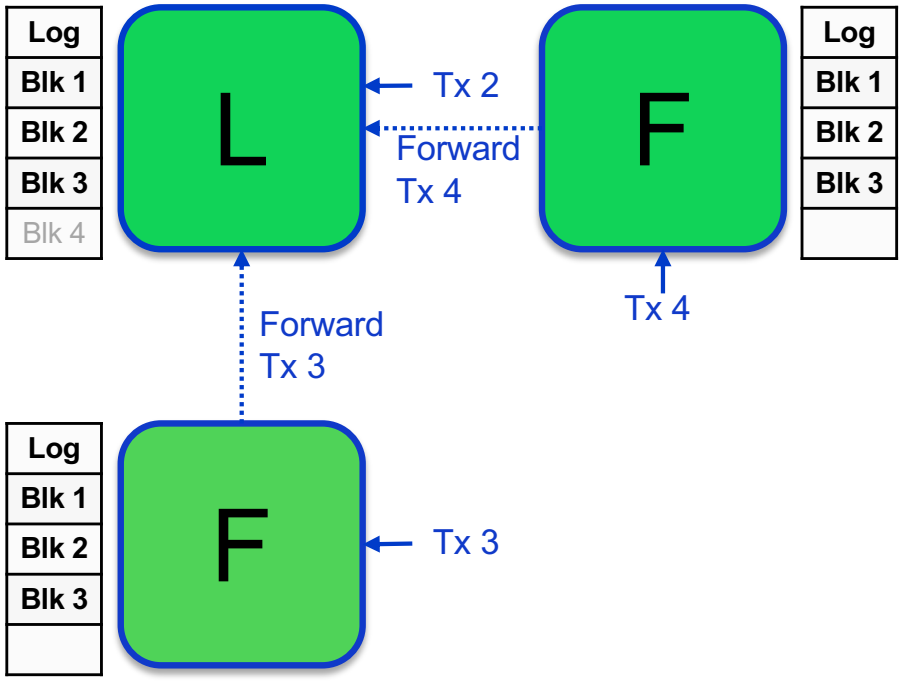
Raft Log Updates

Log Updates: Step 1 – Transactions received and forwarded

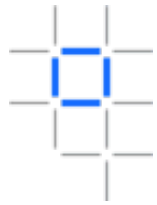
Nodes receive transactions, and **followers** forward transactions to the **leader**

Details:

1. Nodes forward transactions(Tx) to the **leader**
2. The **leader** decides the order of transactions in the next block(Blk)
3. The **leader** adds a new uncommitted log entry to its log containing the the new block of transactions



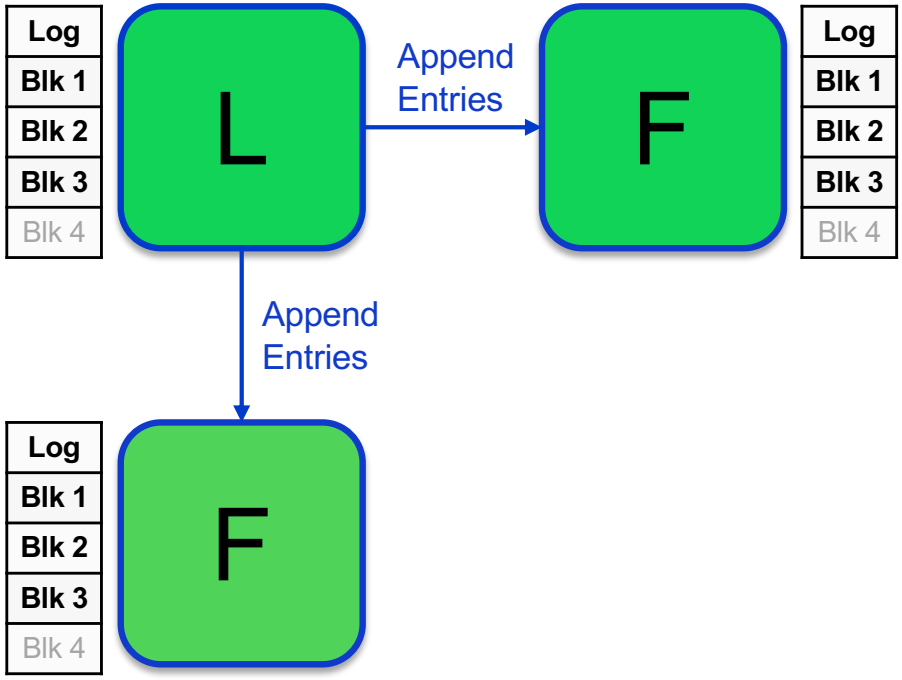
Log Updates: Step 2 – Leader replicates uncommitted log



The **leader** sends the uncommitted log entry to all **followers**

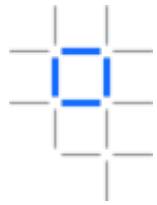
Details:

- At the next heartbeat, the **leader** replicates the new log entry to **follower** nodes
- These are not yet committed by either the **leader** or any **followers**
- **Followers** should all now have a copy of the new entry in their log



IBM

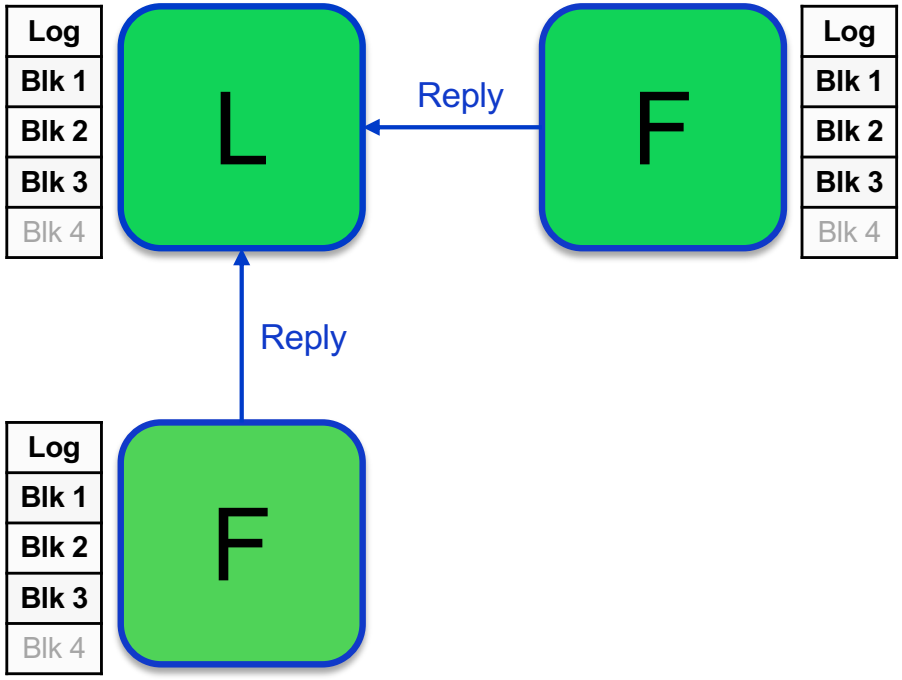
Log Updates: Step 3 – Followers reply to the leader



Followers reply to the **leader** to say they have the new log entry

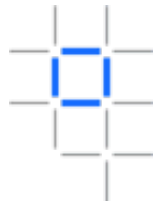
Details:

- Each **follower** replies to the **leader** to confirm they have the new uncommitted log entry, and are ready to commit the changes
- The **leader** needs the majority of **followers** to reply



IBM

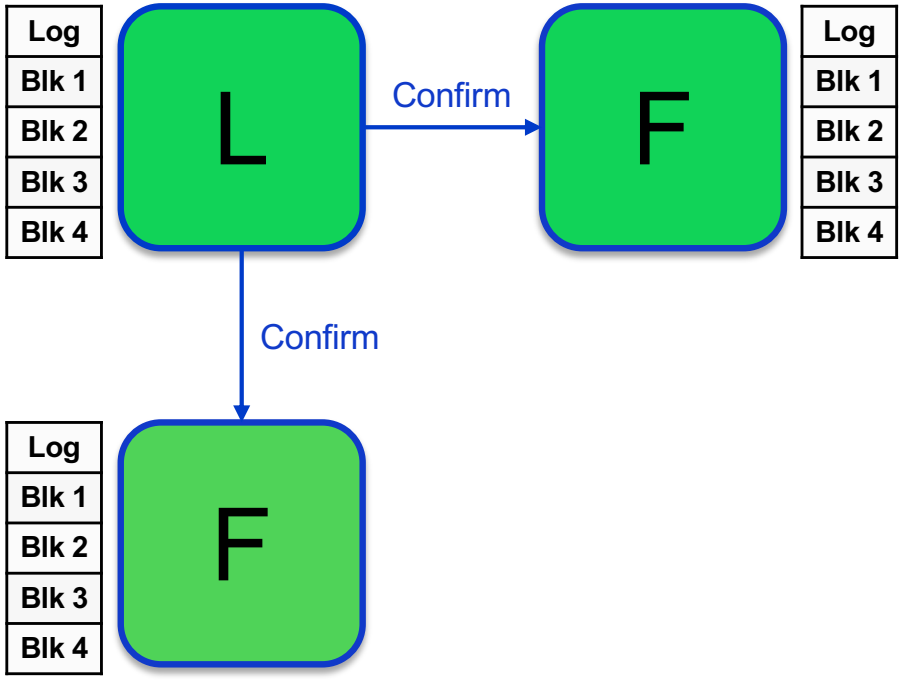
Log Updates: Step 4 – Leader confirms log updates



The **Leader** confirms to all **follower** nodes to commit log entries

Details:

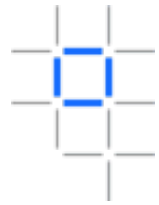
- The **leader** sends a confirmation to all **follower** nodes to confirm that the log entries can be committed
- Each node forwards the commited log entry to all connected **peers** for validation (execute->order->validate)



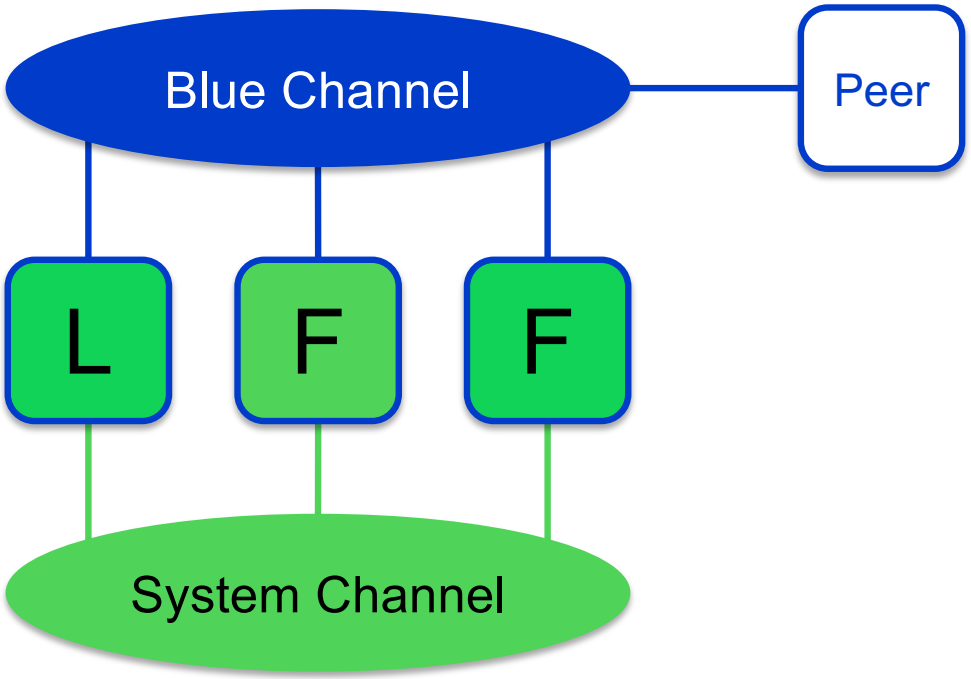
IBM

Example topologies

Raft ordering service with single channel

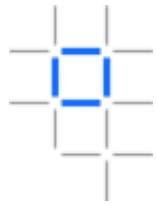


A single user channel (blue) provided by a 3 node Raft based ordering service

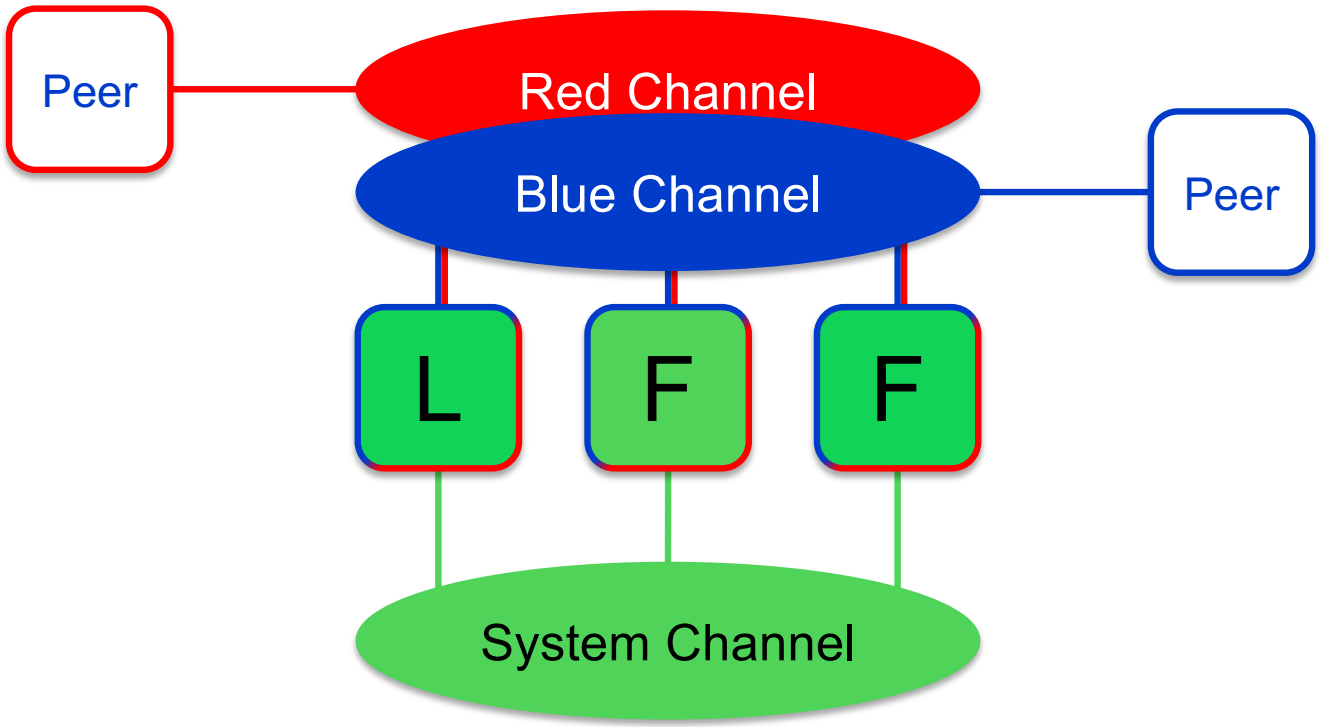


IBM

Raft ordering service with two channels

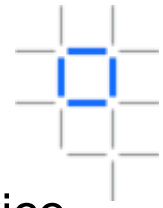


Two user channels provided by the same 3 node Raft based ordering service



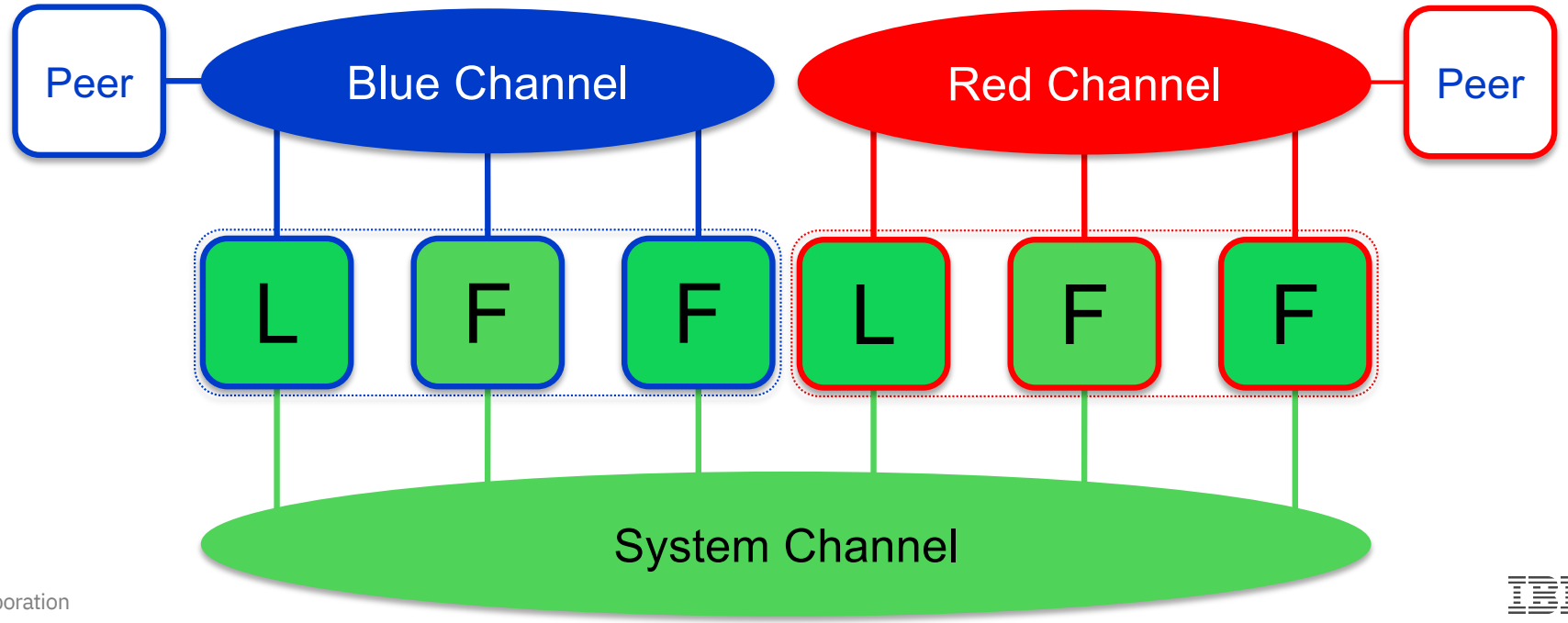
IBM

Raft ordering service with two channels



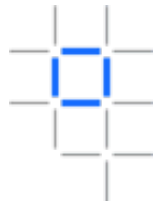
Two user channels provided by different Raft clusters for the same ordering service

System channel is shared by all orderer nodes in a single ordering service



IBM

High Availability and Disaster Recovery



The ordering service is easy to configure for HA and DR

Configuration

- It is recommended that each Raft cluster (a.k.a consenter set) has an odd number of orderer nodes
- This ensures that during a network split, one half of the ordering service should continue ordering transactions as it will have a majority of follower nodes

High Availability

- Achieved by provisioning more than 1 orderer node in each site, with each node running on a different server

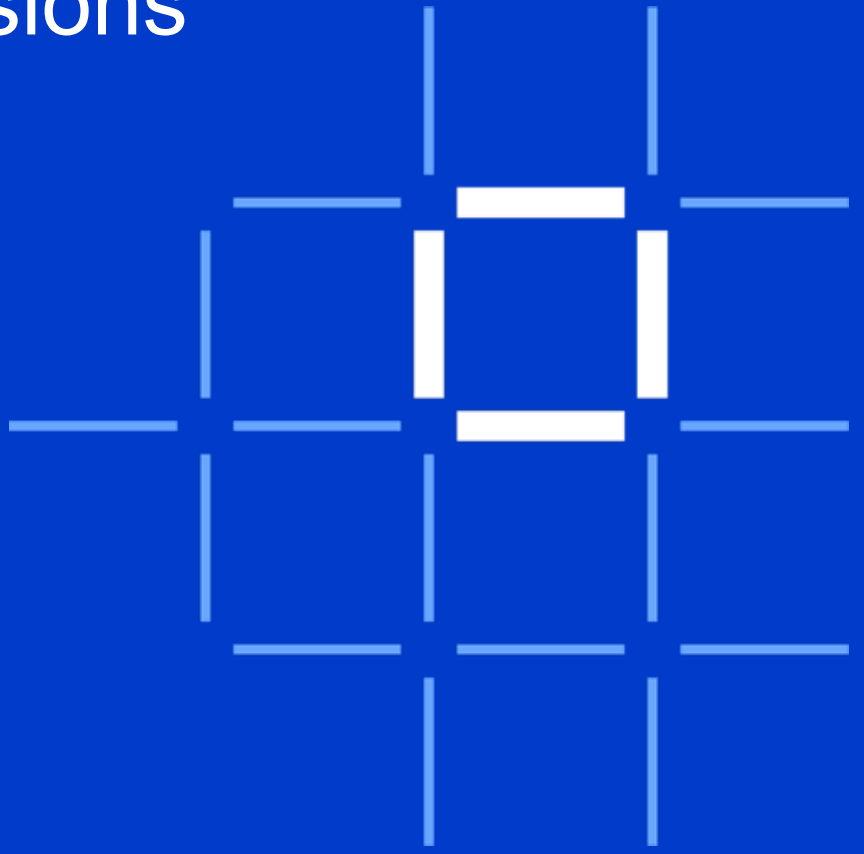
Disaster Recovery

- It is recommended that the ordering service spans at least 3 sites
- This ensures that in the event of any 1 site going offline a majority of follower nodes will continue to run in the remaining sites

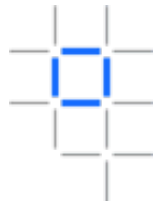
IBM

Hyperledger Fabric Versions

- ☐ **v1.4.1** : Released April 2019
- ☐ **v1.4** : Released January 2019
- ☐ **v1.3** : Released October 2018
- ☐ **v1.2** : Released June 2018
- ☐ **v1.1** : Released March 2018
- ☑ **Future**



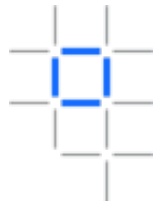
Future - Chaincode Lifecycle improvements(*)



Decentralized chaincode governance and remove problematic LSCC custom validation
This is a pre-req for post order execution

- The new v2.0 chaincode lifecycle introduces decentralized governance for chaincode, with a new process for installing a chaincode on your peers and starting it on a channel. The new Fabric chaincode lifecycle allows multiple organizations to come to agreement on the parameters of a chaincode, such as the chaincode endorsement policy, before it can be used to interact with the ledger. Specifically, the new model offers several improvements over the previous lifecycle:
    - **Multiple organizations must agree to the parameters of a chaincode:** Previous versions of Fabric, one organization had the ability to set parameters of a chaincode (for instance the endorsement policy) for all other channel members. The new Fabric chaincode lifecycle is more flexible since it supports both centralized trust models (such as that of the previous lifecycle model) as well as decentralized models requiring a sufficient number of organizations to agree on an endorsement policy before it goes into effect.

Future - Chaincode Lifecycle improvements(*)



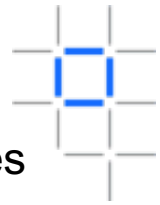
- **Safer chaincode upgrade process:** For chaincode lifecycle in previous versions of Fabric, the upgrade transaction could be issued by a single organization, creating a risk for a channel member that had not yet installed the new chaincode. The new model allows for a chaincode to be upgraded only after a sufficient number of organizations have approved the upgrade.

- **Easier endorsement policy updates:** Fabric chaincode lifecycle will allow you to change an endorsement policy without having to repackage or reinstall the chaincode. Users will also be able to take advantage of a new default policy that requires endorsement from a majority of members on the channel. This policy will be updated automatically when organizations are added or removed from the channel.

- **Inspectable chaincode packages:** Chaincode is packaged in easily readable tar files. This makes it easier to inspect the chaincode package and coordinate installation across multiple organizations.

- **Start multiple chaincodes on a channel using one package:** The previous lifecycle defined each chaincode on the channel using a name and version that was specified when the chaincode package was installed. You will be able to use a single chaincode package and deploy it multiple times with different names on the same or different channel.

Peer and chaincode native cloud support(*)



Enable chaincode to run on Kubernetes cloud deployments following security best practices and conformance

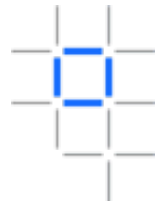
Currently the hosting peer builds the chaincode Docker image and launches the container during the instantiate transaction. This model causes nonconformance with standard security practices in certain production deployments:

- Endorsing peers must run in privileged mode to build and launch Docker chaincode images
- Chaincode container can't be provisioned and scaled by cloud native tools (eg K8s)
- Chaincode package contains source code that can be confidential due to sensitive business logic
- Dependence on peer to build and launch chaincode which doesn't work well with CI/CD process

Future, focus on resolving these issues with 2 phases of development:

- Remove the elevated privilege configuration from the peer container enabling it to be deployed like any typical microservices in cloud environment
- Enable auto provisioning and elastic scaling of chaincode container with cloud native tools

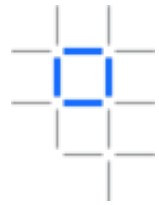
Future – Validation refactoring(*)



Refactor the validator/committer for simplicity, maintainability, performance, and extensible to support various processing models. This is a pre-req for post-order execution

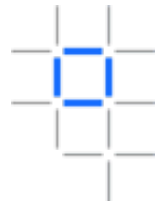
- **High throughput:** the validation component should strive to make use of all the available computation resources.
- **Testability:** it should be possible to unit-test the validator component without having to set up peer artifacts such as the ledger or the MSP.
- **Increase test coverage**.
- **Create a validation pipeline:** structuring the validation/committer pipeline as a set of processing elements connected in series has several advantages:
  - It paves the way for low-latency validation if in the future processing of block n+1 may commence before processing of block n is complete (the benefits of this approach are described here https://arxiv.org/abs/1808.08406)
  - It keeps the code simpler: understanding, coding and testing a component (e.g. the component that does signature checking) doesn't require understanding or changing another component
  - It makes it easy to extend the validation logic by adding a new stage in the pipeline (e.g. a new component that filters transactions based on a new logic, or that handles a specific new type of transaction

Future – Validation refactoring(*) cont…



- **Avoid capability switches in the validation path:** instead of performing capability checks inside the validation code, keep different validation logics, one for each release that modifies the validation logic in substantial ways. The right validation logic should be called based on the channel capabilities. This should go beyond what was done for FAB-11552.
- **Minimize unmarshalling:** the current validation component repeatedly performs unmarshalling to extract the same fields. Fields relevant for validation should be unmarshalled only once as part of the input validation step.
- **Avoid complex synchronization if possible:** state-based endorsement requires synchronization mechanisms to detect dependencies and ensure the correct validation scheduling; we should attempt to only parallelize tasks that are not constrained by dependencies such as signature verification, while performing others sequentially. It's a matter of balancing performance with correctness.
- **Plan for the hooks for enhanced concurrency control (FAB-10711):** enhanced concurrency control requires the ability to further mutate the read-write set of a transaction based on the richer set of operations requested at chaincode-simulation time. We should plan for a step in the validation/committing pipeline where such a hook may be enabled.

Future - Further data privacy enhancements(*)



Private data – Org specific collections (FAB-10889)

- Useful for storing state specific to a single org

Private data – Local collections (FAB-7593)

- Client determines private data dissemination dynamically instead of using static private data collection policies

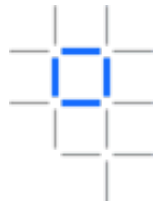
Zero-knowledge proof

- Privacy-preserving asset transfer with public verifiability

State based endorsement – Privacy preserving (FAB-8820)

- Hides state based endorsement policy

Future – Additional features(*)



Performance

- Add a state database cache to significantly improve the performance of endorsement and validation transaction phases.

Post order refactoring

- The ability to perform post-order commit-time operations (e.g. addition/subtraction, range checks) so that collisions on endorsed read/write sets can be avoided in high TPS systems.

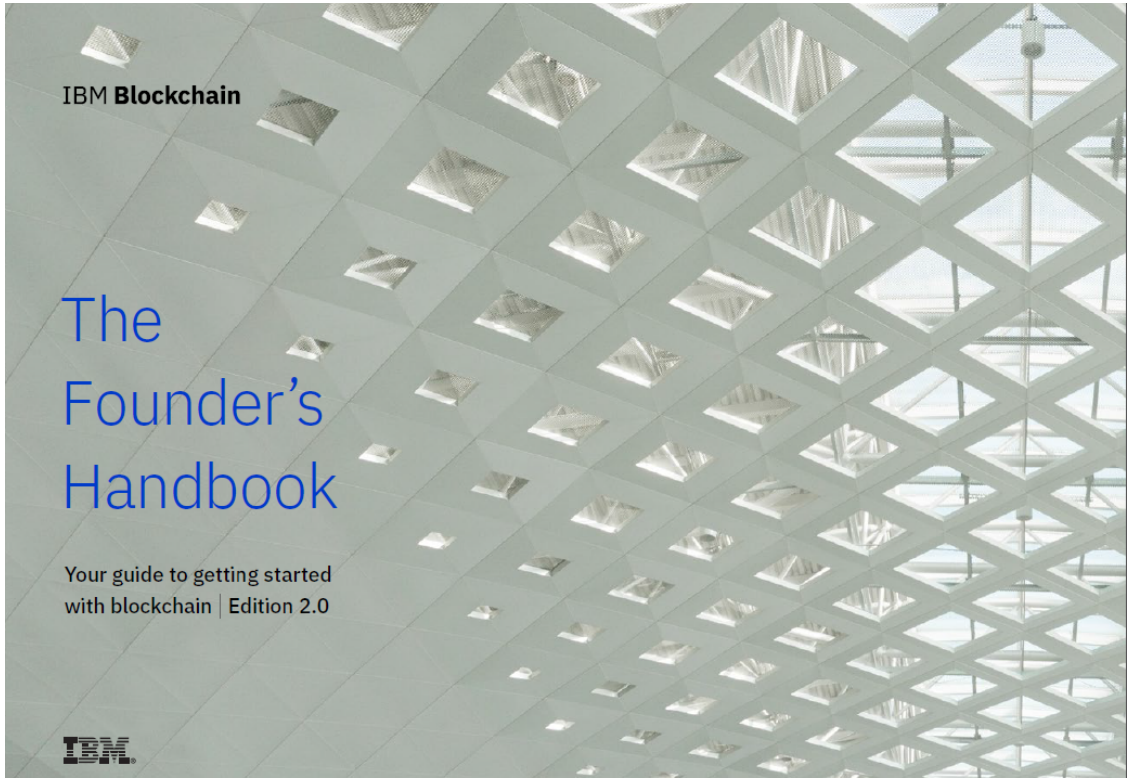
Checkpoint and Archive

- **Checkpoint** - Peers dump global state and consent on a global channel state hash (e.g. nightly or weekly).
- Checkpoint block with global state hash gets submitted to ordering.
- Peers and orderers can optionally prune/archive blocks prior to the checkpoint block.
- With a checkpointed global state, new peers can join a channel from the **checkpointed state** and **latest checkpoint block**, without reprocessing every prior block.

BFT orderer consensus algorithm

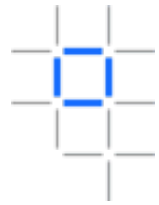
- Performant version in process

IBM Blockchain handbook



https://public.dhe.ibm.com/common/ssi/ecm/28/en/28014128usen/the-founders-handbook-edition-2_28014128USEN.pdf

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

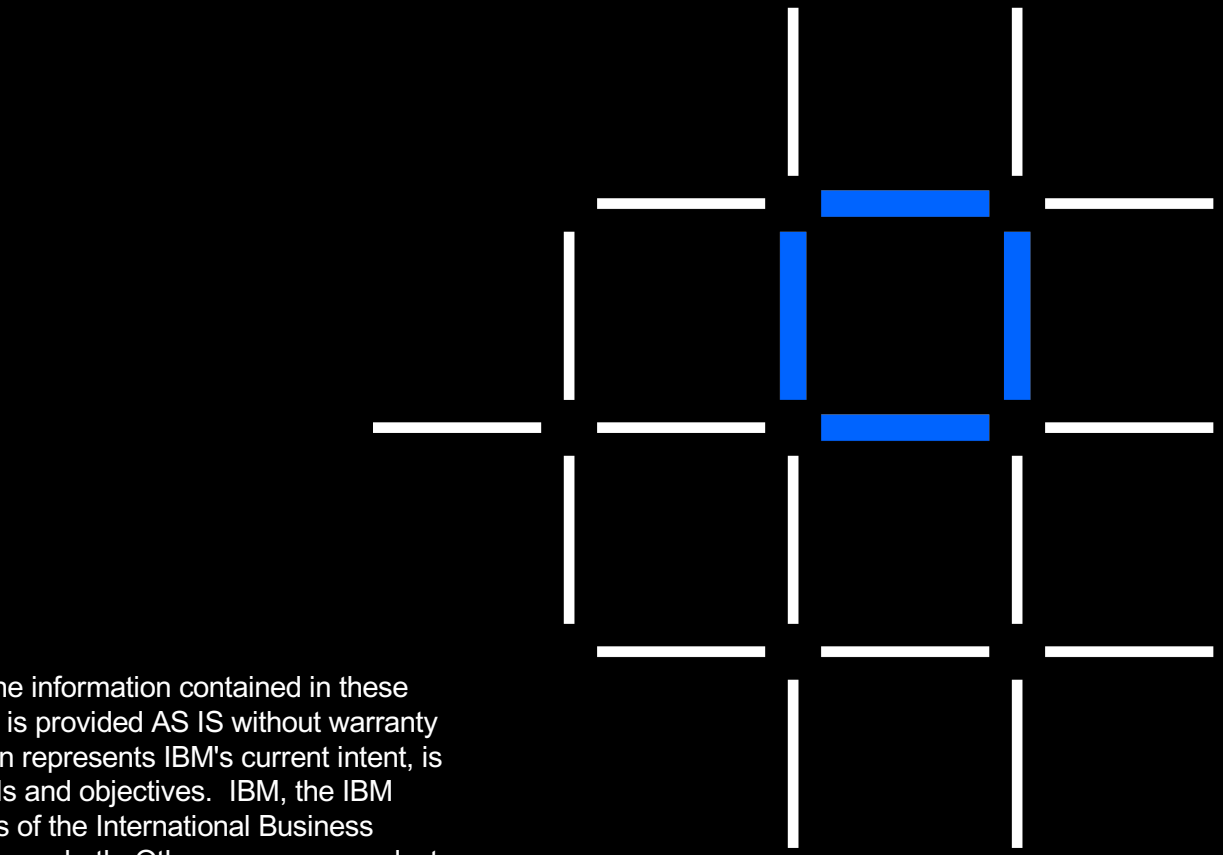
Barry Silliman
silliman@us.ibm.com

IBM **Blockchain**

www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org



IBM


IBM