BLOCKCHAIN 3.0 – HYPERLEDGER

HYPERLEDGER

- Hyperledger is an open-source blockchain framework backed up by Linux Foundation and collaborative project that provides tools, standards and guidelines for developing permissioned blockchain solutions across various industries.

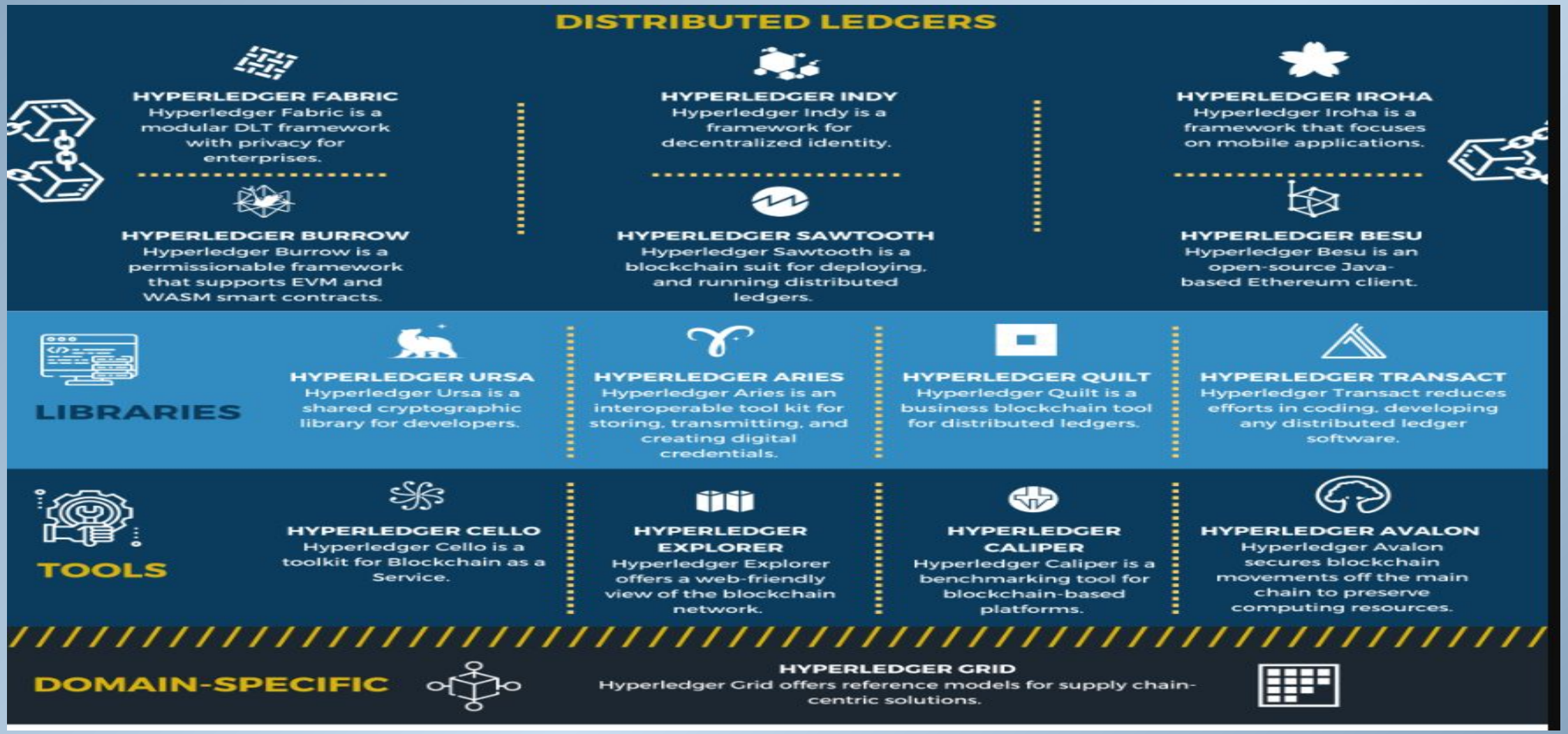
HYPERLEDGER – Key Takeaway

- Hyperledger is an open-source community focused on developing a suite of stable frameworks, tools and libraries for permissioned, enterprise-grade blockchain deployments.
- It is a global collaboration and includes member organizations that are leaders in finance, banking, Internet of Things, supply chains, manufacturing, and technology.
- Several sub-projects exist, including Hyperledger Fabric, Cello, Besu, and Caliper.

HYPERLEDGER

- It is a system that lets the users decide what they want based on their needs.
- So, a system can be built using one of the existing Hyperledger projects (consider these to be operating systems designed for specific purposes) with different modules for various needs.

HYPERLEDGER Projects



Hyperledger Indy

- Hyperledger Indy project is another popular Hyperledger project. Basically, it's a framework explicitly made for decentralized identities. More so, it comes with loads of components, toolsets, and libraries for you to use.
- The best part is that it powers users as users will get the sole ownership of their very own identities.

Main Features of Hyperledger Indy Project

- **Self-sovereignty : C**an store all your identity-based documentation on the network. Basically, all the contents will have cryptographic encryptions, proof of existence, private key and public keys and many more. Only you can alter or make changes to those documents.
- **Privacy : A** secured layer of confidentiality for protecting all your valuable documents. Everything will remain only in the ledger, and no one can even track down your identity from another link or ledger.
- **Verifiable Claims :** In many cases, you may have to show some form of identity documentation, such as a birth certificate or driving license to prove it. But in many cases, people can misuse these documents as well. Thus, you can only choose to disclose selective portions of your document to prove your identity.

Hyperledger Sawtooth

- Hyperledger Sawtooth is actually a blockchain suit for deploying, running, and building distributed ledgers.
- Many enterprises have a hard time working with the complex nature of blockchain technology.
- Sawtooth offers the perfect solution as it makes implementing blockchain solutions an easy task.

Features of Hyperledger Sawtooth

Dynamic Consensus

- Here, anyone can change a consensus in a live blockchain network. More so, you can even do it in the transaction process. But you have to choose available consensus plugins for that.

Proof of Elapsed Time (PoET)

- It’s a new type of consensus algorithm. Mainly it’s on by default, but you can change it if you need it. In reality, the work process is quite unique, and it consumes very less energy.

Transaction Families

- You can write smart contracts in any language, there will be no restriction in that.

Features of Hyperledger Sawtooth

Ethereum Contracts Compatibility

- Another best feature of Hyperledger Sawtooth is that it's 100% compatible with Ethereum smart contracts. So, if you want to use that, you can do it by plugging EVM.

Parallel Transaction Execution

- It also comes with parallel transaction execution saving you a lot of time.

Private Transactions

- You can create individual chains where you can keep all your confidential transactions easily.

Hyperledger Explorer

- This project aims to build a blockchain explorer for Hyperledger that can be used to **view and query the transactions, blocks,** and associated data from the blockchain.
- It also provides network information and the ability to interact with **chain code**.
- Currently there are two other projects that are in incubation: **Fabric chaintool, and Fabric SDK Py**. These projects are aimed at supporting **Hyperledger Fabric.**

Hyperledger Fabric

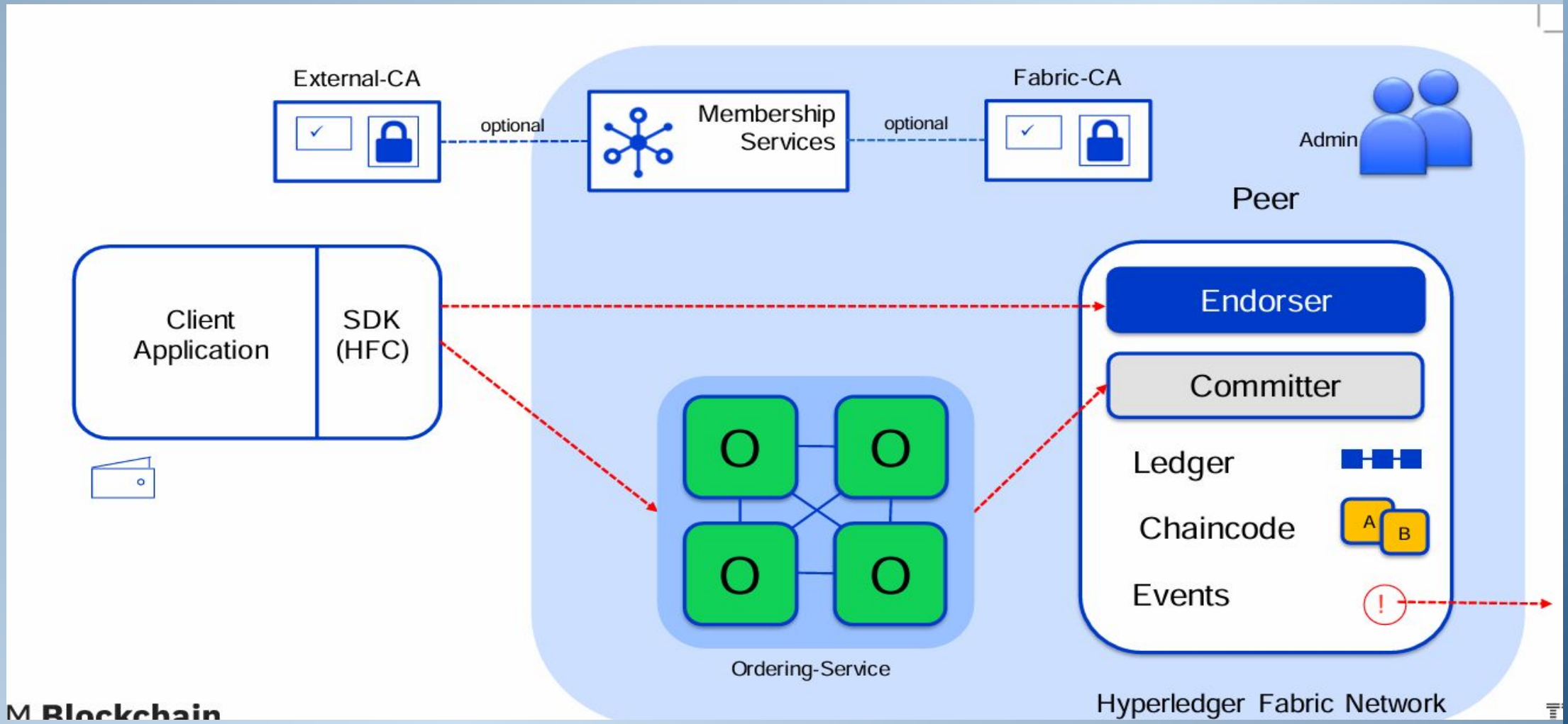
- Hyperledger Fabric project is one of the most popular modular projects under Hyperledger.
- The platform is fully optimizable. Thus, you can use it for any kind of use case.
- Furthermore, with the Hyperledger Fabric project, you'll get confidentiality, flexibility, resiliency, scalability, and many more.
- The Hyperledger Fabric project runs on the general-purpose programming language. As a result, there isn't any native token for this.

- An implementation of blockchain technology that is intended as a foundation for developing blockchain applications for the enterprise
- Key characteristics:– Permissioned– Highly modular
- Pluggable consensus, ledger, membership services, endorsement and validation
- Smart contracts in general purpose languages– Privacy– No “mining” or native crypto-currency required for consensus
- Execute-order-validate vs order-execute

Features of Hyperledger Fabric project

- **The Modularity:** It comes with a modular design. So, you can just plug in any feature and just start using it. There are some modules already available to use, such as ordering services, membership providers, gossip services, smart contracts, etc.
- **Smart Contracts:** In the Hyperledger Fabric project, smart contracts are called **chaincode**. In reality, it's a bit different than typical smart contracts as it can get all the functions it needs form the blockchain network. More so, it follows three steps – **execution, ordering, and validation**.
- **Pluggable Hyperledger Fabric Consensus Protocols :** This is by far the best features of Fabric. All the consensus mechanism within the system is completely pluggable.

Hyperledger Fabric Architecture



Hyperledger Fabric Architecture

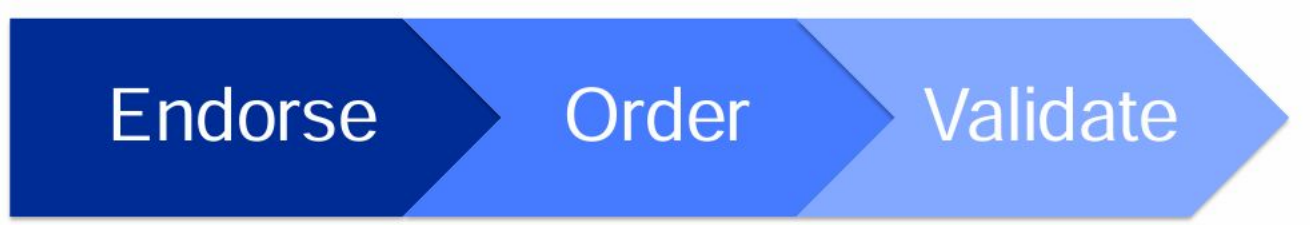
- The reference architecture consists of two main components:
  - Hyperledger services
  - Hyperledger APIs, SDKs, andCLI.
- Hyperledger services provide various services such as identity services, policy services, blockchain services and smart contract services.
- Hyperledger APIs, SDKs, and CLIs **provide an interface into blockchain services** via appropriate application programming interfaces, software development kits, or command line interfaces.

Nodes and roles

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

Hyperledger Fabric Consensus

- Consensus is achieved using the following transaction flow:

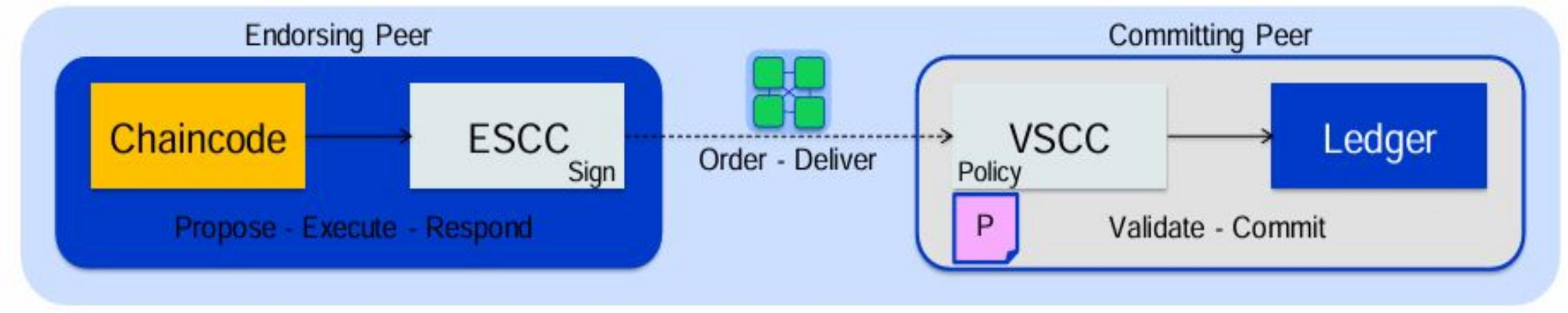


IDENTITIES AND POLICIES

Endorsement Policies

- An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.
  - Each chaincode is deployed with an Endorsement Policy
  - ESCC(Endorsement System Chaincode) signs the proposal response on the endorsing peer
  - VSCC(Validation System Chaincode) validates the endorsement

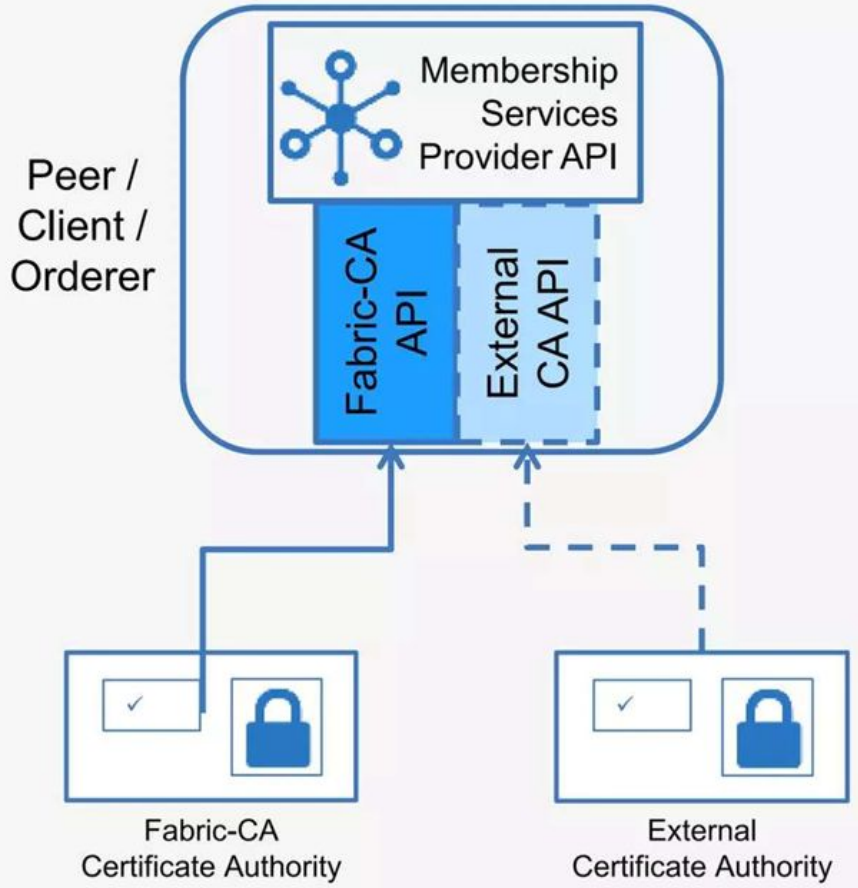
Endorsement Policies



Endorsement Policy Examples

- Examples of policies:
- Request 1 signature from all three principals
  - AND('Org1.member', 'Org2.member', 'Org3.member')
- Request 1 signature from either one of the two principals
  - OR('Org1.member', 'Org2.member')
- Request either one signature from a member of the Org1 MSP or (1 signature from a member of the Org2 MSP and 1 signature from a member of the Org3 MSP)
  - OR('Org1.member', AND('Org2.member', 'Org3.member')

Membership and Access Control

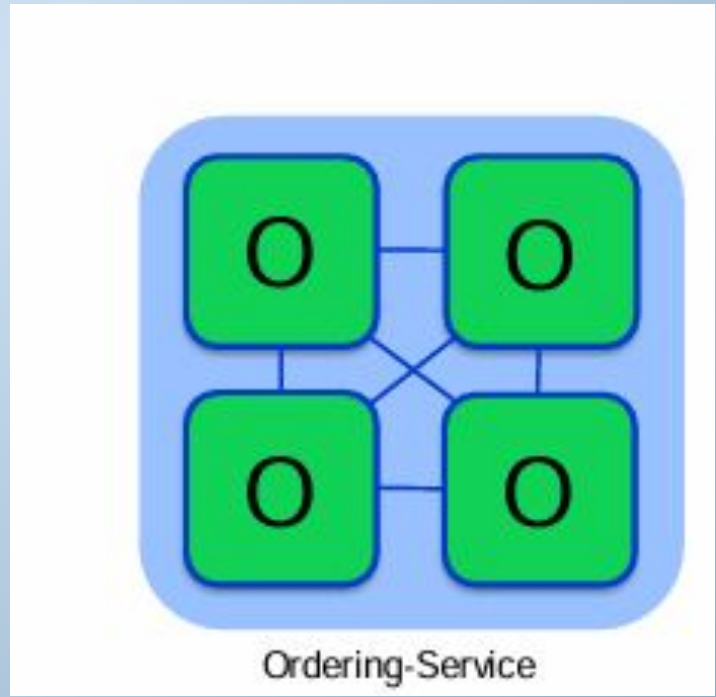


Membership Services Provider API

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

Ordering Service

- The ordering service packages transactions into blocks to be delivered to peers. Communication with the service is via channels.
- Different configuration options for the ordering service include:
- – Solo • Single node for development
- – Kafka : Crash fault tolerant consensus
- 3 nodes minimum
- Odd number of nodes recommended
- – Raft : Crash fault tolerant
- • In-process consensus, no extra dependency
- • More decentralized, orderer nodes can be spread



Consensus: Solo, Kafka, RAFT

- **SOLO:** It is the Hyperledger Fabric ordering mechanism most typically used by developers experimenting with Hyperledger Fabric networks.
- SOLO involves a single ordering node. In this, the transactions are ordered in chronological order to form a block.

Consensus: Solo, Kafka, RAFT

- **Kafka:** It is the Hyperledger Fabric ordering mechanism that is, recommended for production use.
- This ordering mechanism utilizes Apache Kafka, an open-source stream processing platform that provides a unified, high-throughput, low- latency platform for handling real-time data feeds.
- In this case, the data consists of endorsed transactions and RW sets. The Kafka mechanism provides a crash fault-tolerant solution to ordering service.

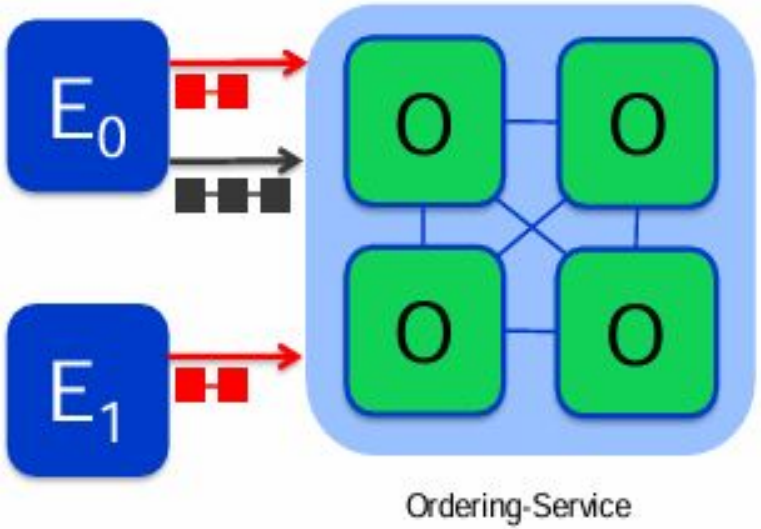
Consensus: Solo, Kafka, RAFT

- RAFT: Raft is a modern, reliable and relatively less complicated distributed consensus algorithm that is frequently used in modern software solutions such as Consul, etc, RabbitMQ and so on.
- Raft is the first step toward Fabric's development of a byzantine fault tolerant (BFT) ordering service.

Feature	Solo	Kafka	Raft
Type	Single-node ordering	Crash Fault Tolerant (CFT)	Crash Fault Tolerant (CFT)
Architecture	Single orderer node	Multiple orderers + external Kafka cluster	Multiple orderers (internal Raft cluster)
External Dependencies	None	Requires Kafka and Zookeeper	No external dependencies
Fault Tolerance	None (single point of failure)	Tolerates node failures using Kafka brokers	Tolerates node failures using built-in Raft consensus
Scalability	Very limited	Scales well with multiple brokers	Scales well, simpler than Kafka
Performance	Fast (single node)	High throughput	High throughput
Ease of Setup	Easiest	Complex (requires Kafka/Zookeeper setup)	Moderate (built into Fabric)
Use Case	Development / Testing only	Production (Fabric v1.0–v1.3)	Production (Fabric v1.4+ and v2.x)
Leader Election	No leader (single node)	Kafka manages partitions	Raft elects leader among orderers automatically
Dynamic Membership	No	No (restart required)	Yes (can add/remove orderers dynamically)

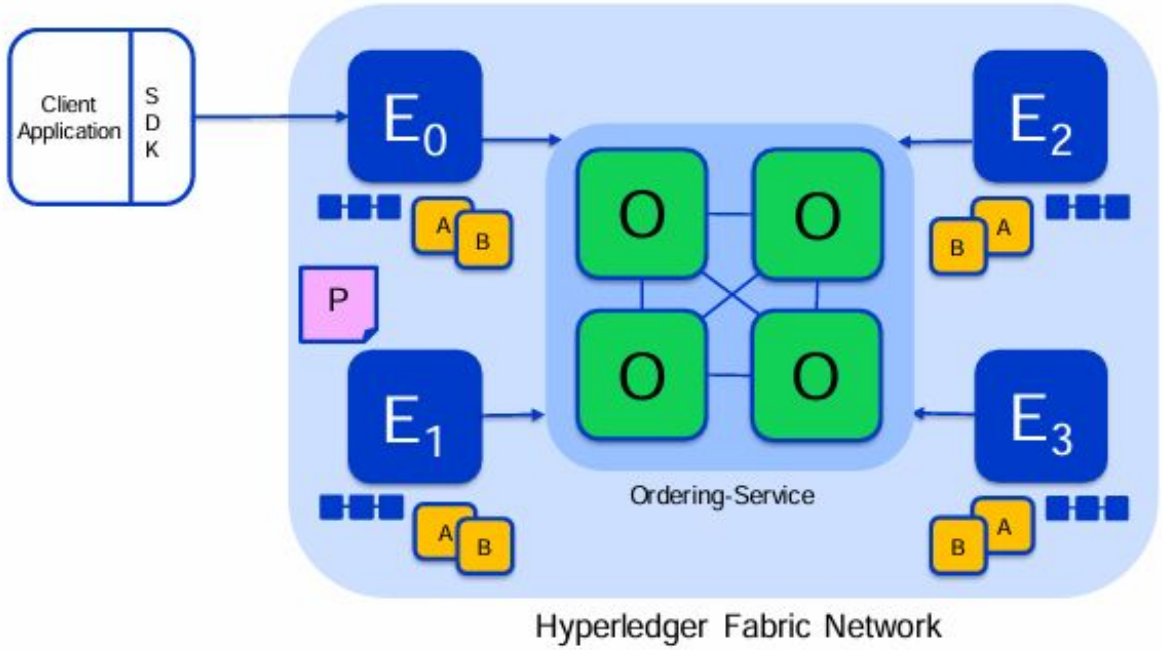
Channels

- Channels provide privacy between different ledgers

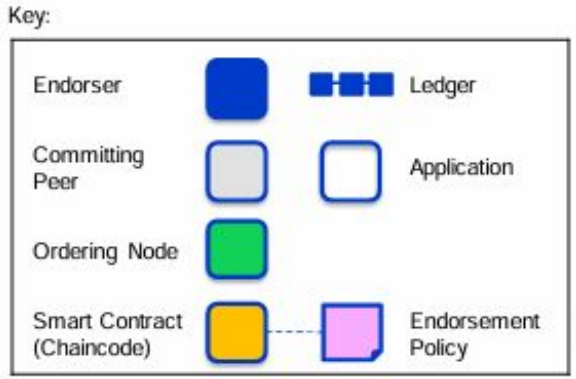


– Ledgers exist in the scope of a channel
  - Channels can be shared across an entire network of peers
  - Channels can be permissioned for a specific set of participants
– Chaincode is installed on peers to access the worldstate
– Chaincode is instantiated on specific channels
– Peers can participate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

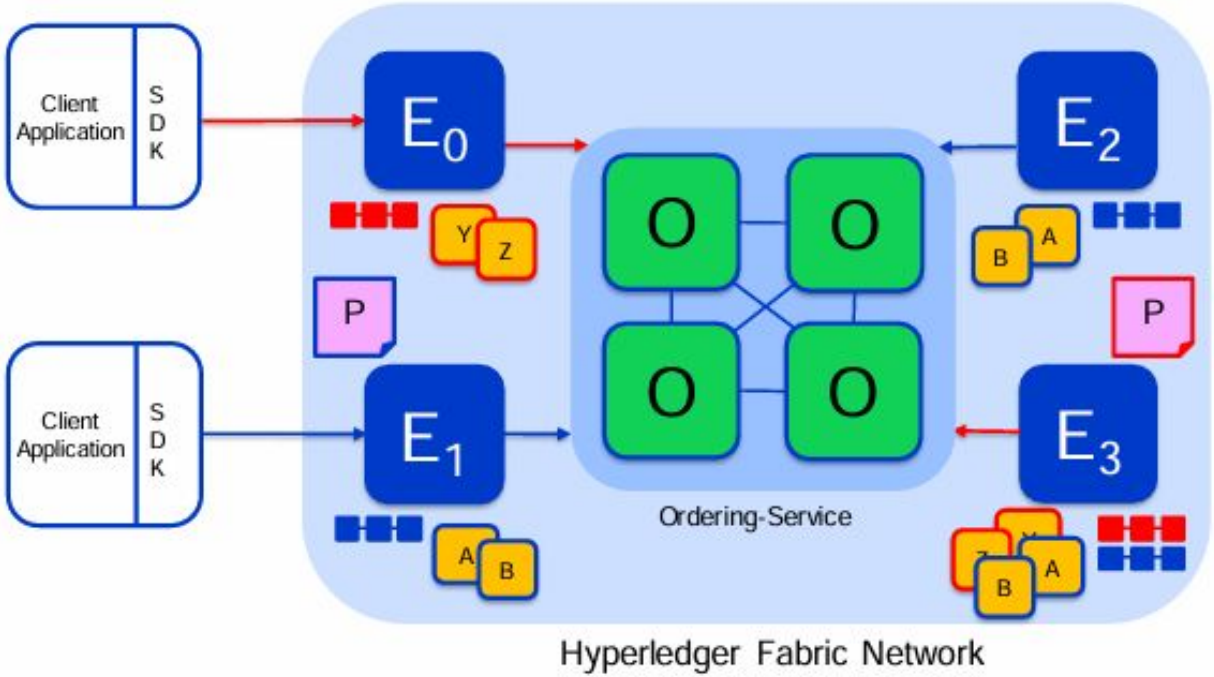
Single Channel Network



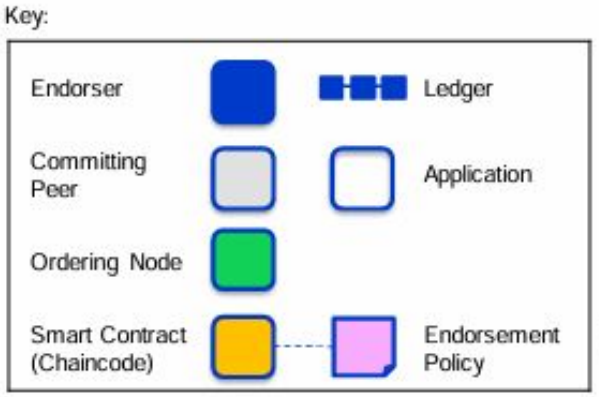
- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3



Multi Channel 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

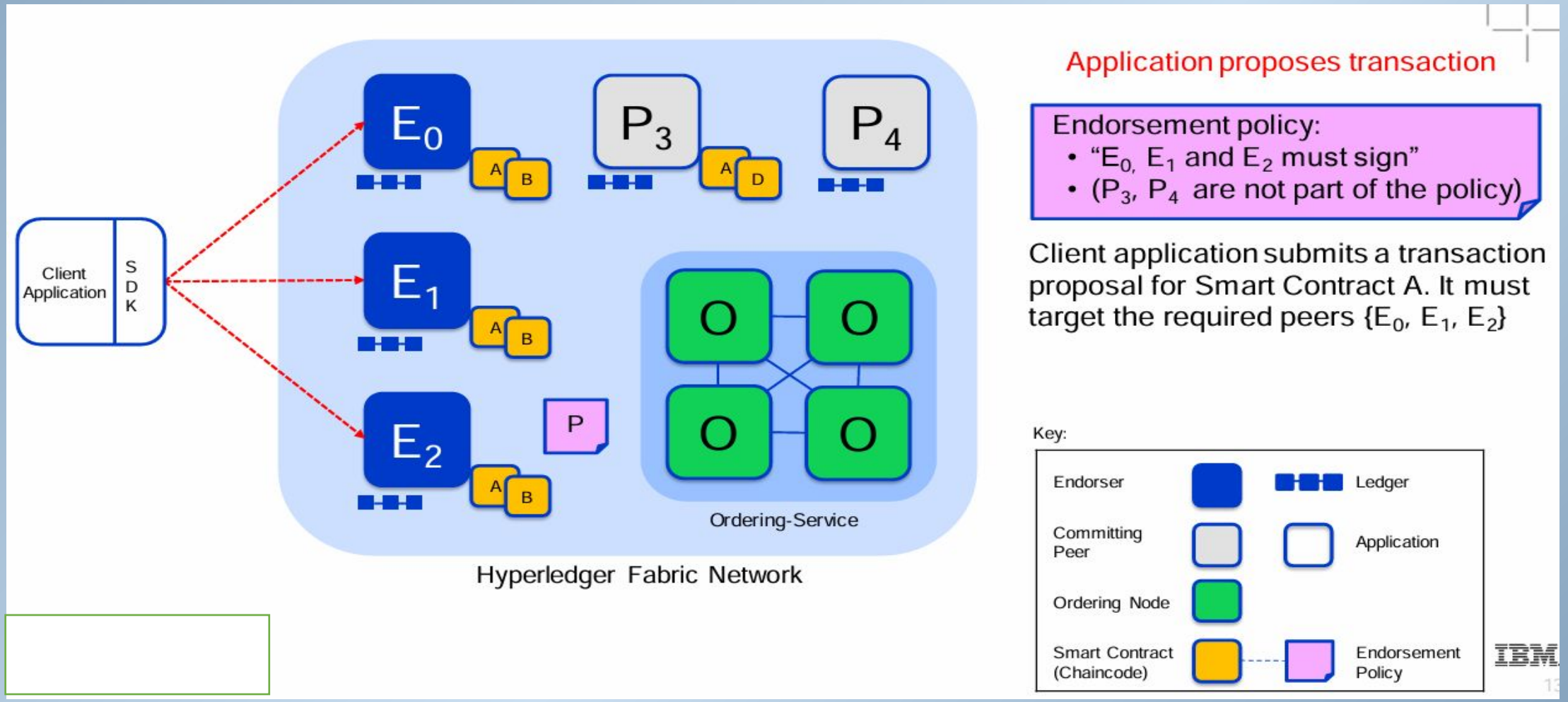


IBM **Blockchain**

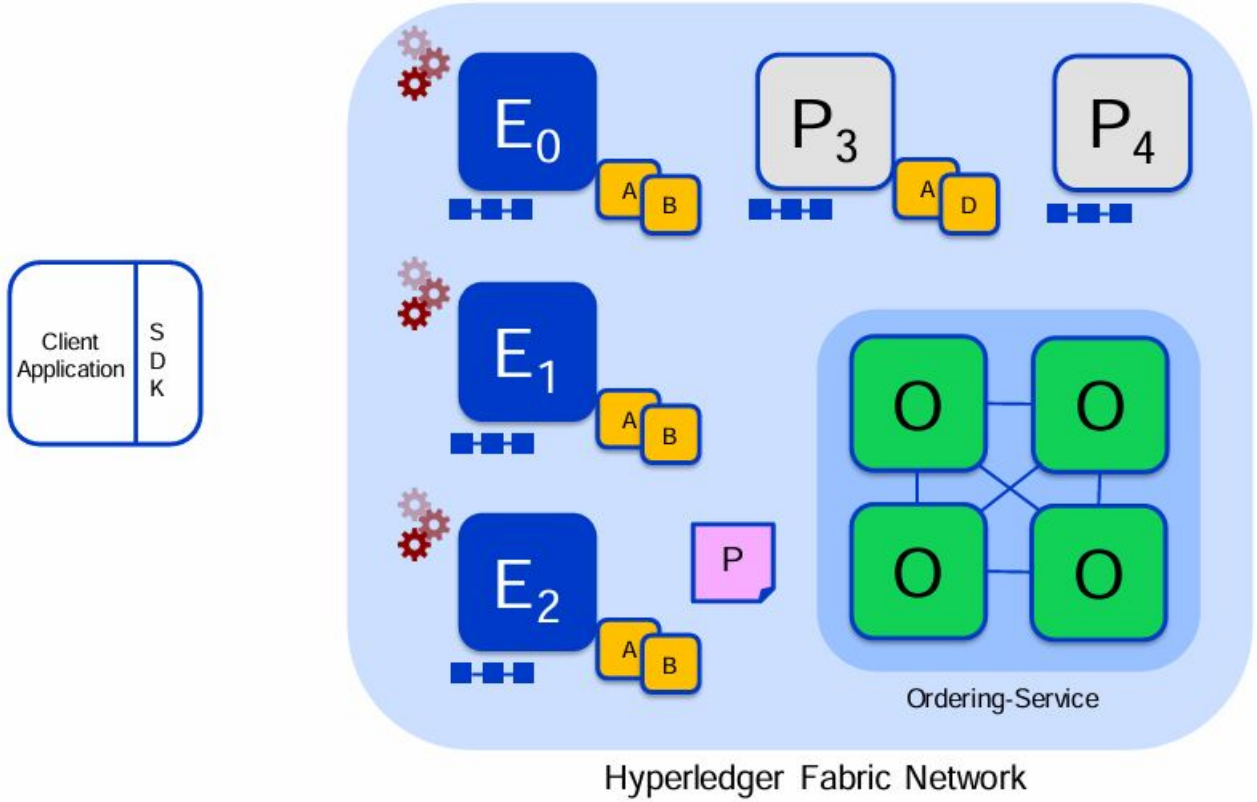
IBM

Feature	**Single Channel**	**Multi-Channel**
Definition	One shared channel for all organizations.	Multiple channels, each with a subset of organizations.
Ledger	One shared ledger for all participants.	Separate ledger per channel (isolated ledgers).
Data Visibility	All members see all transactions.	Only members of a specific channel can see its transactions.
Privacy	Low — every org on the channel can read data.	High — channels isolate data among organizations.
Smart Contracts (Chaincode)	One chaincode instance runs for all members.	Each channel can have its own chaincode instance and version.
Use Case	When all organizations can share all data (e.g., consortium where data transparency is required).	When different sets of organizations need to transact privately or maintain different ledgers.
Management Overhead	Low — easier to set up and maintain.	Higher — each channel requires separate setup, policies, and management.
Scalability	Easier scaling, but limited data separation.	More flexible and secure, but adds operational complexity.
Example Scenario	A supply chain where all parties share the same data.	A supply chain where manufacturers and suppliers share one channel, and suppliers and retailers share another.

Transaction Validation: Sample transaction: Step 1/7 – Propose transaction



Sample transaction: Step 2/7 – Execute proposal



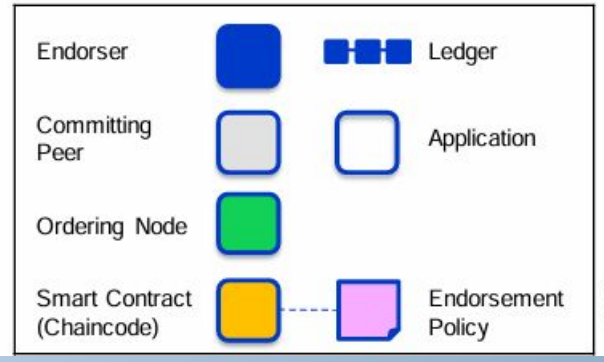
Endorsers Execute Proposals

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

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

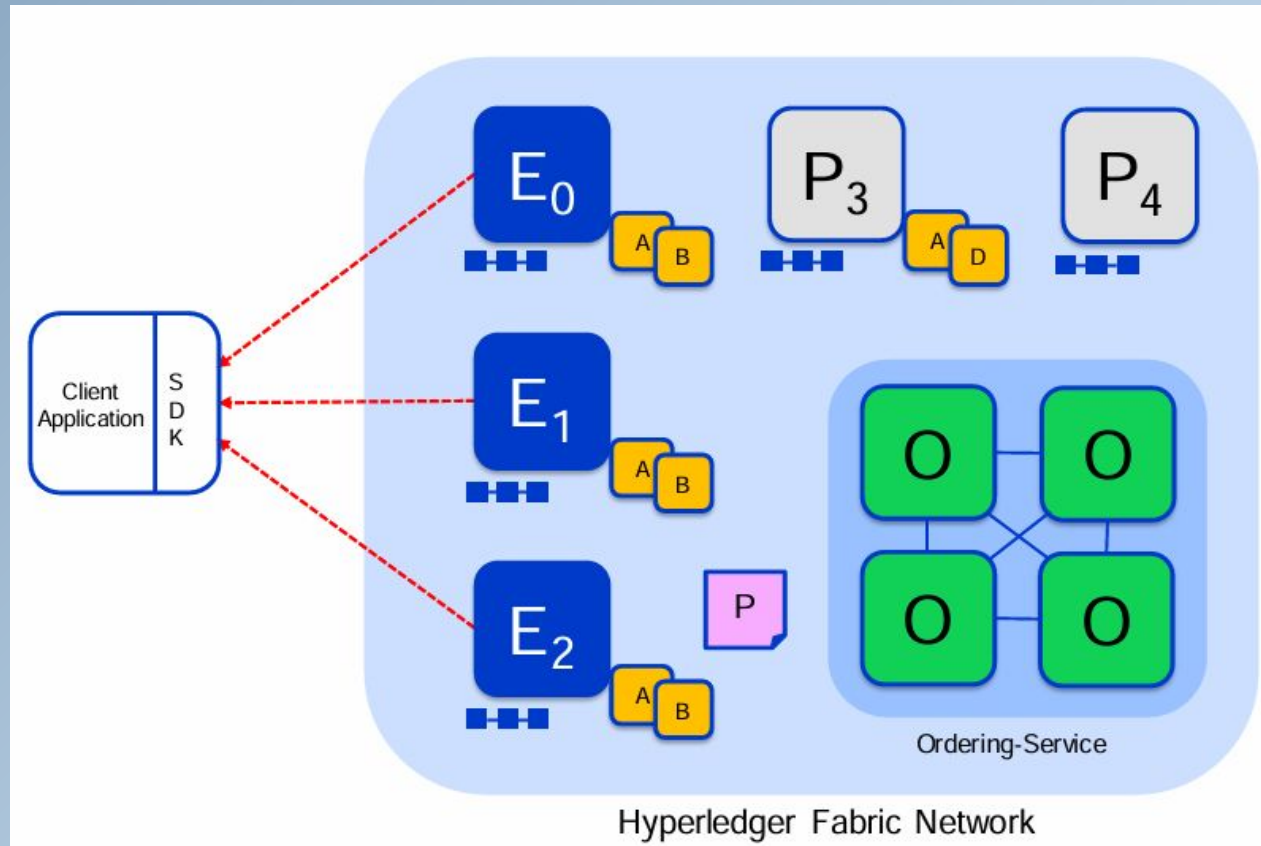
Transactions can be signed & encrypted

Key:



IBM

Sample transaction: Step 3/7 – Proposal Response

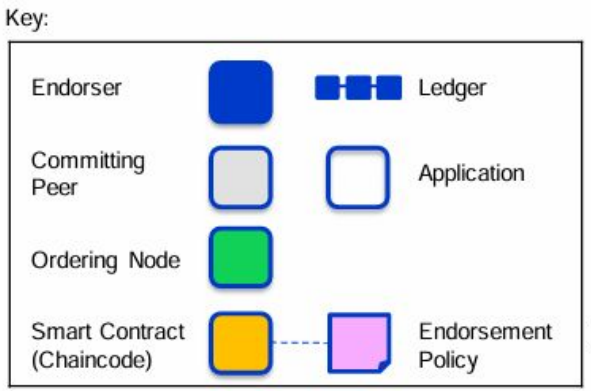


Application receives responses

RW sets are asynchronously returned to application

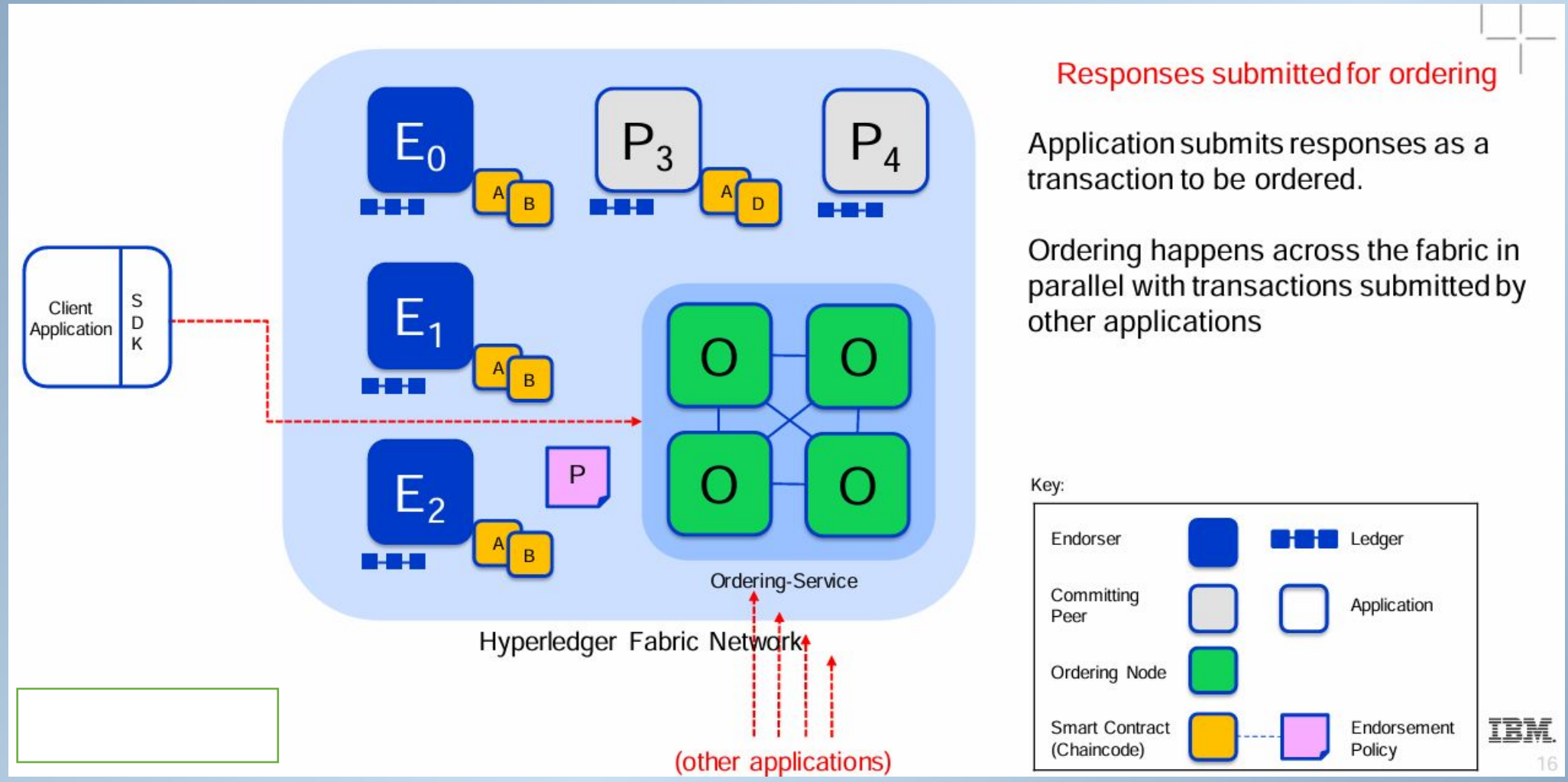
The RW sets are signed by each endorser, and also includes each record version number

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

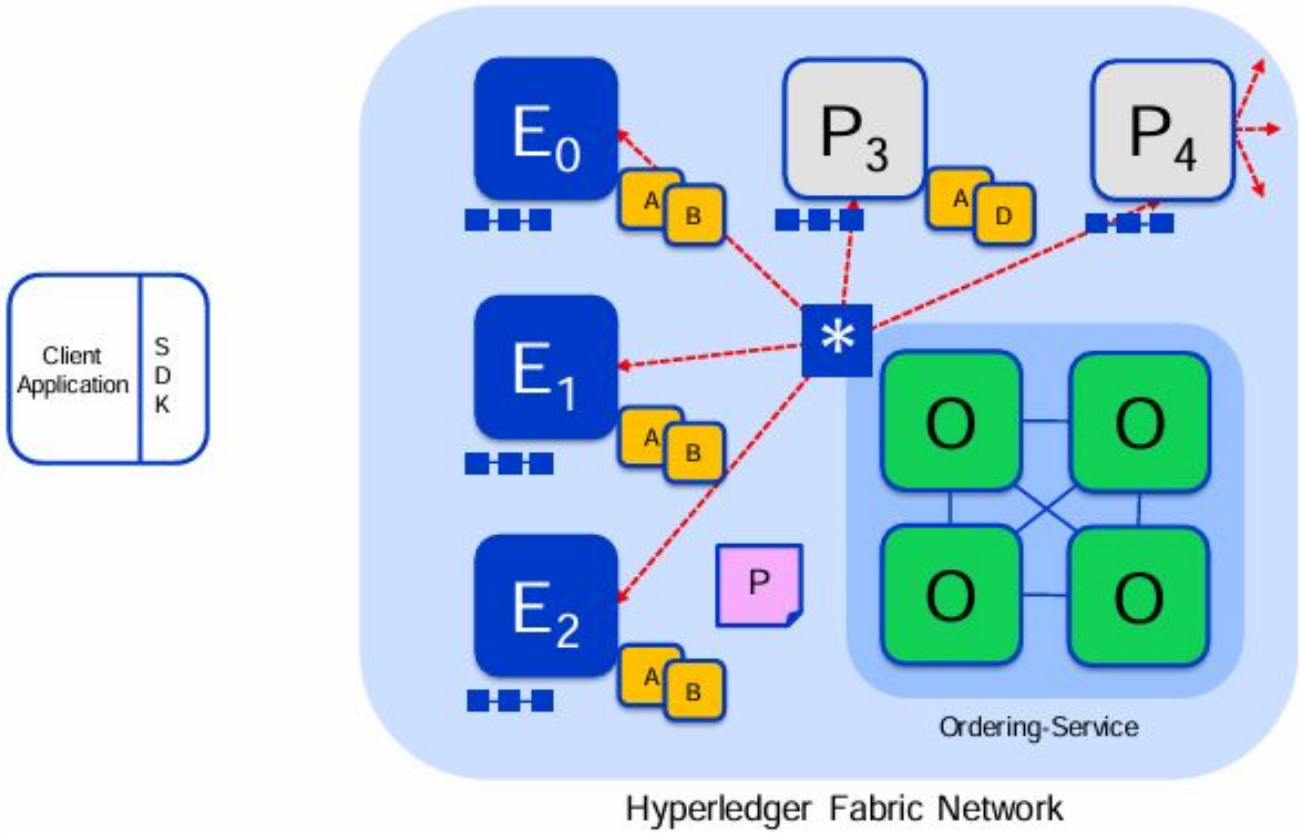


IBM

Sample transaction: Step 4/7 – Order Transaction



Sample transaction: Step 5/7 – Deliver Transaction

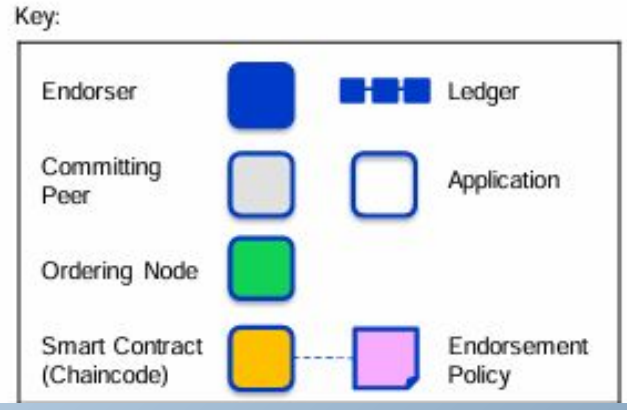


Orderer delivers to committing peers

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

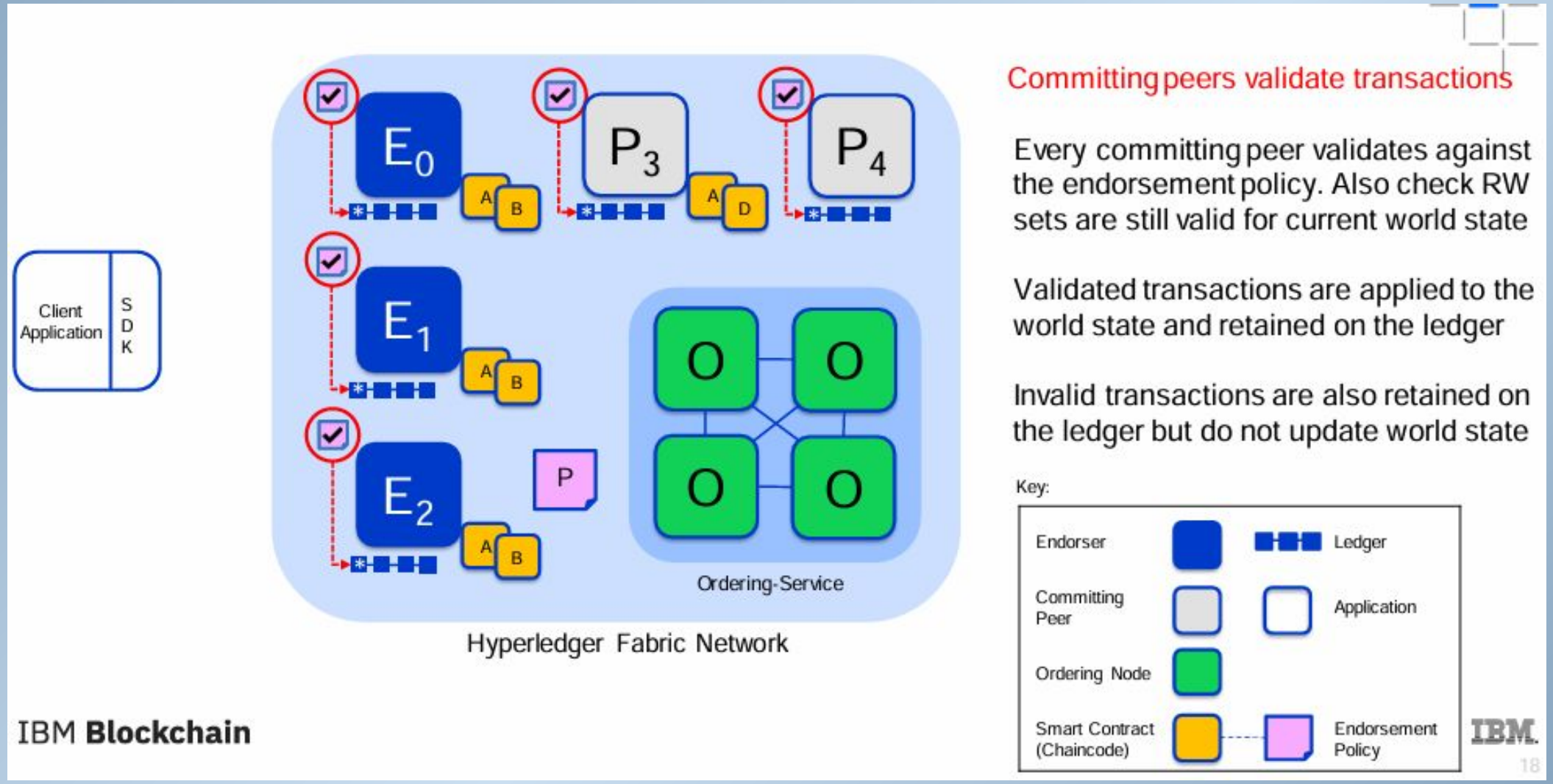
Different ordering algorithms available:

- SOLO (Single node, development)
- Kafka (Crash fault tolerant)
- Raft (Crash fault tolerant)

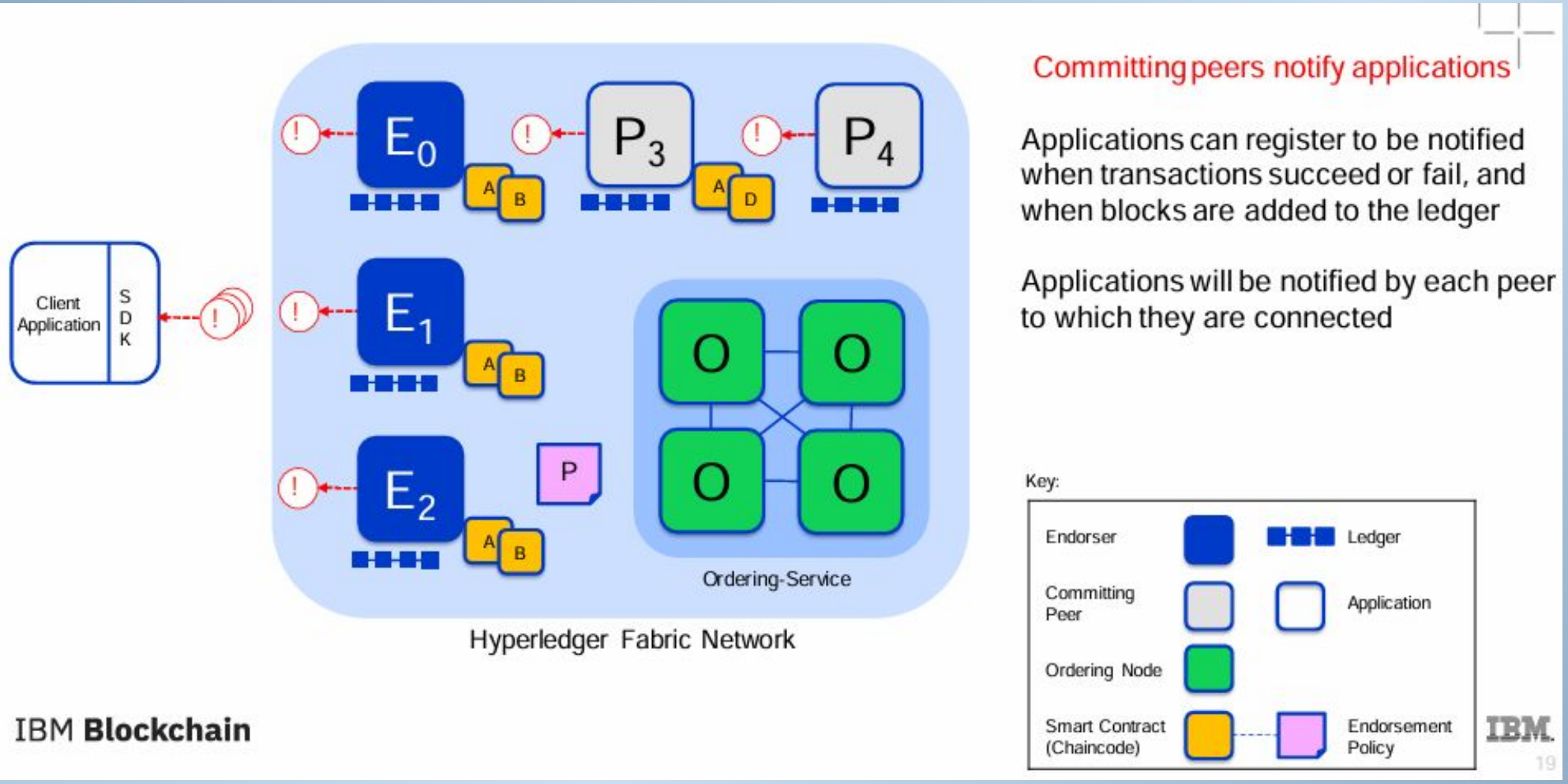


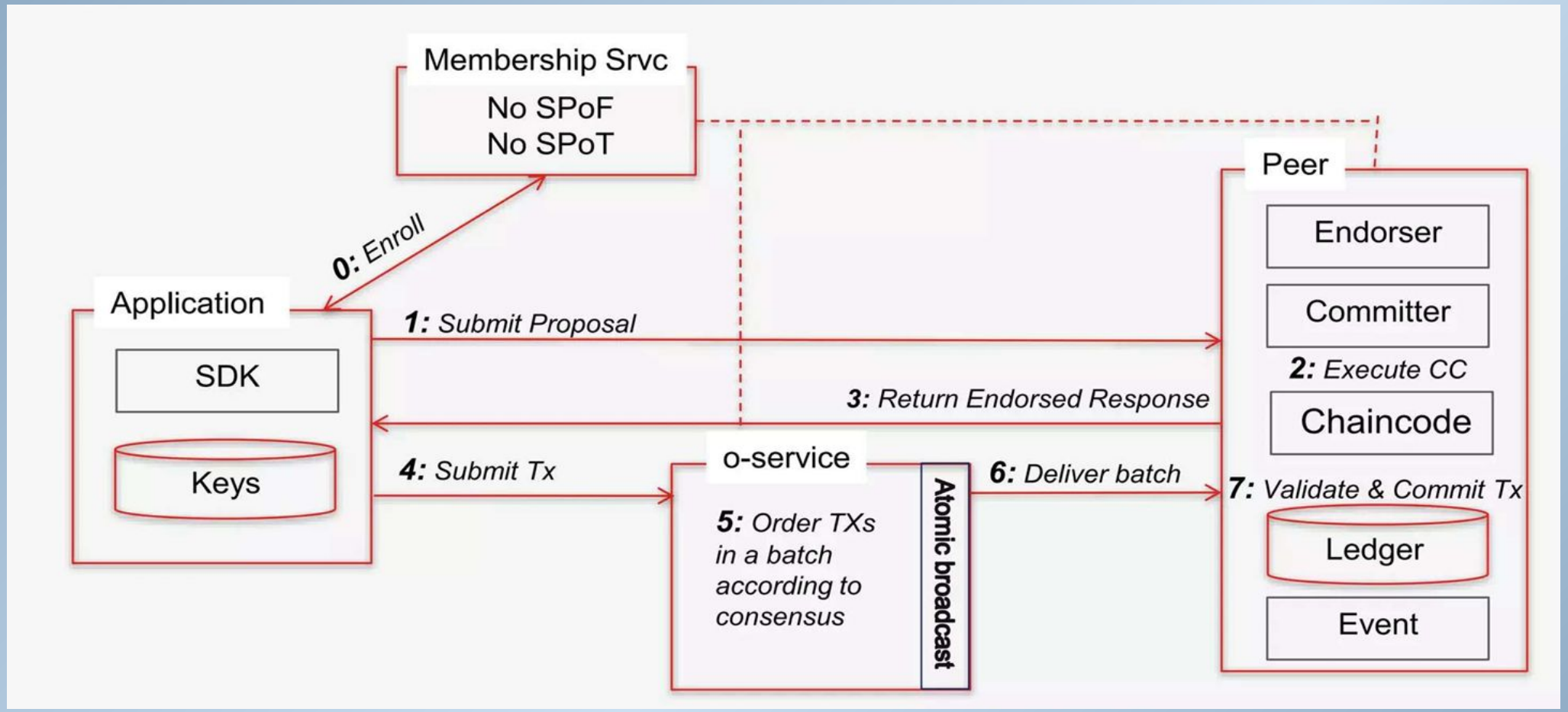
IBM

Sample transaction: Step 6/7 – Validate Transaction



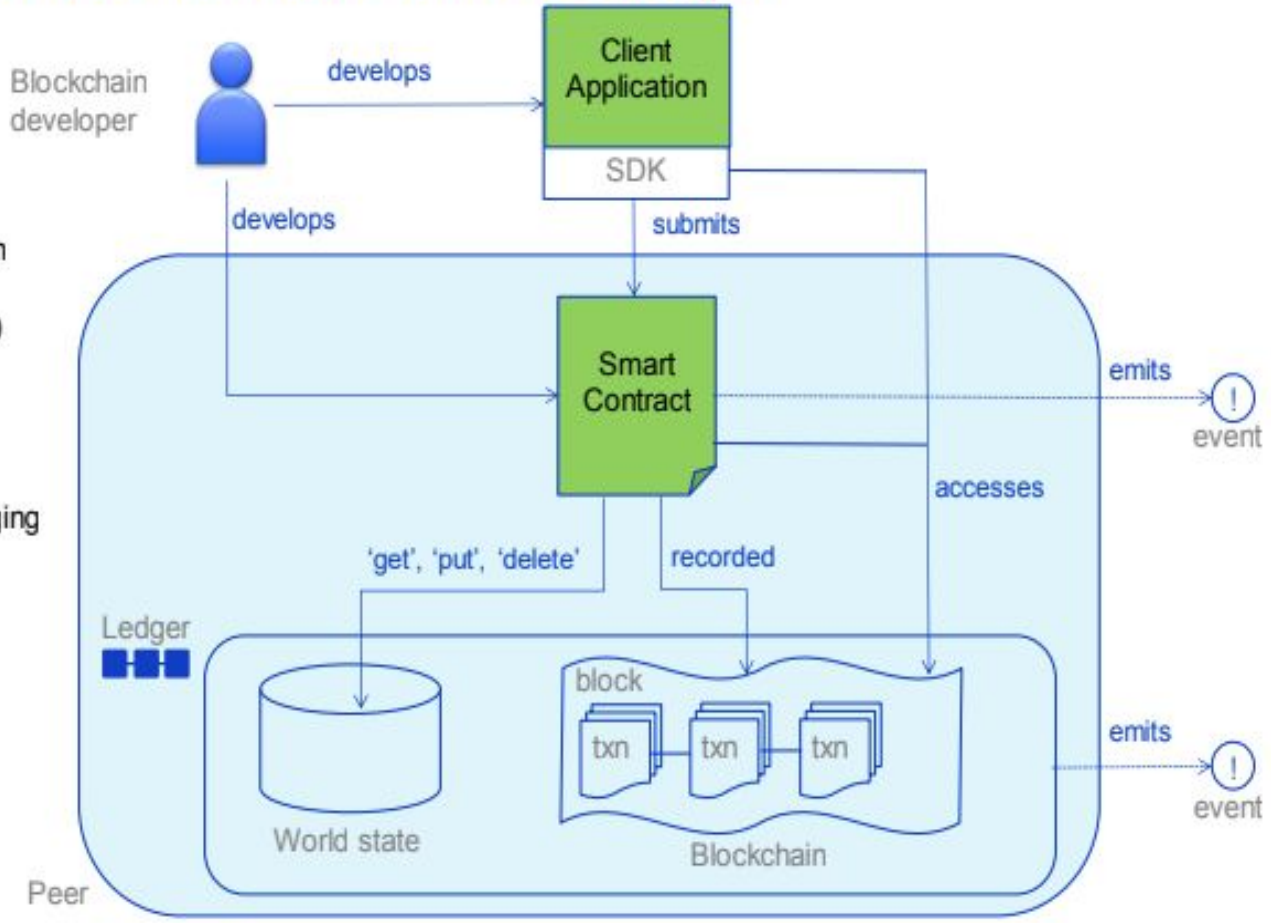
Sample transaction: Step 7/7 – Notify Transaction




Membership Srvc
No SPoF
No SPoT
0: Enroll
Application
SDK
Keys
1: Submit Proposal
3: Return Endorsed Response
4: Submit Tx
o-service
5: Order TXs in a batch according to consensus
Atomic broadcast
6: Deliver batch
Peer
Endorser
Committer
2: Execute CC
Chaincode
7: Validate & Commit Tx
Ledger
Event

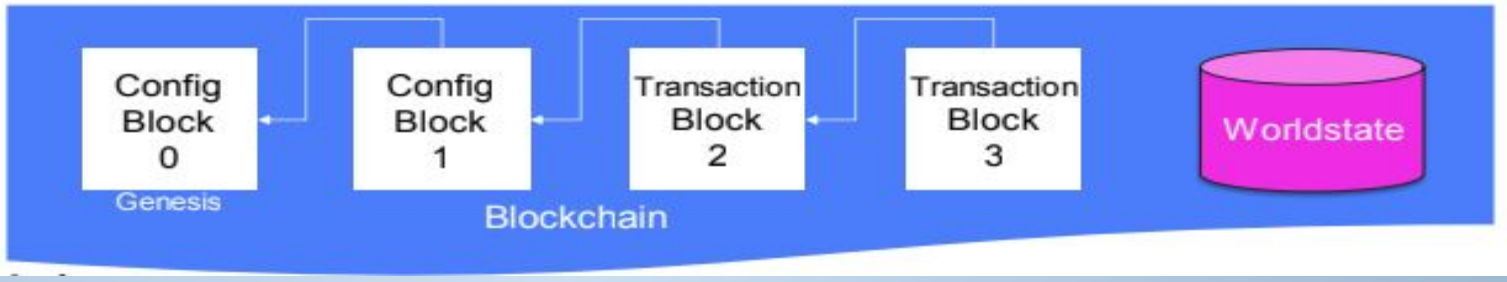
How applications interact with the ledger

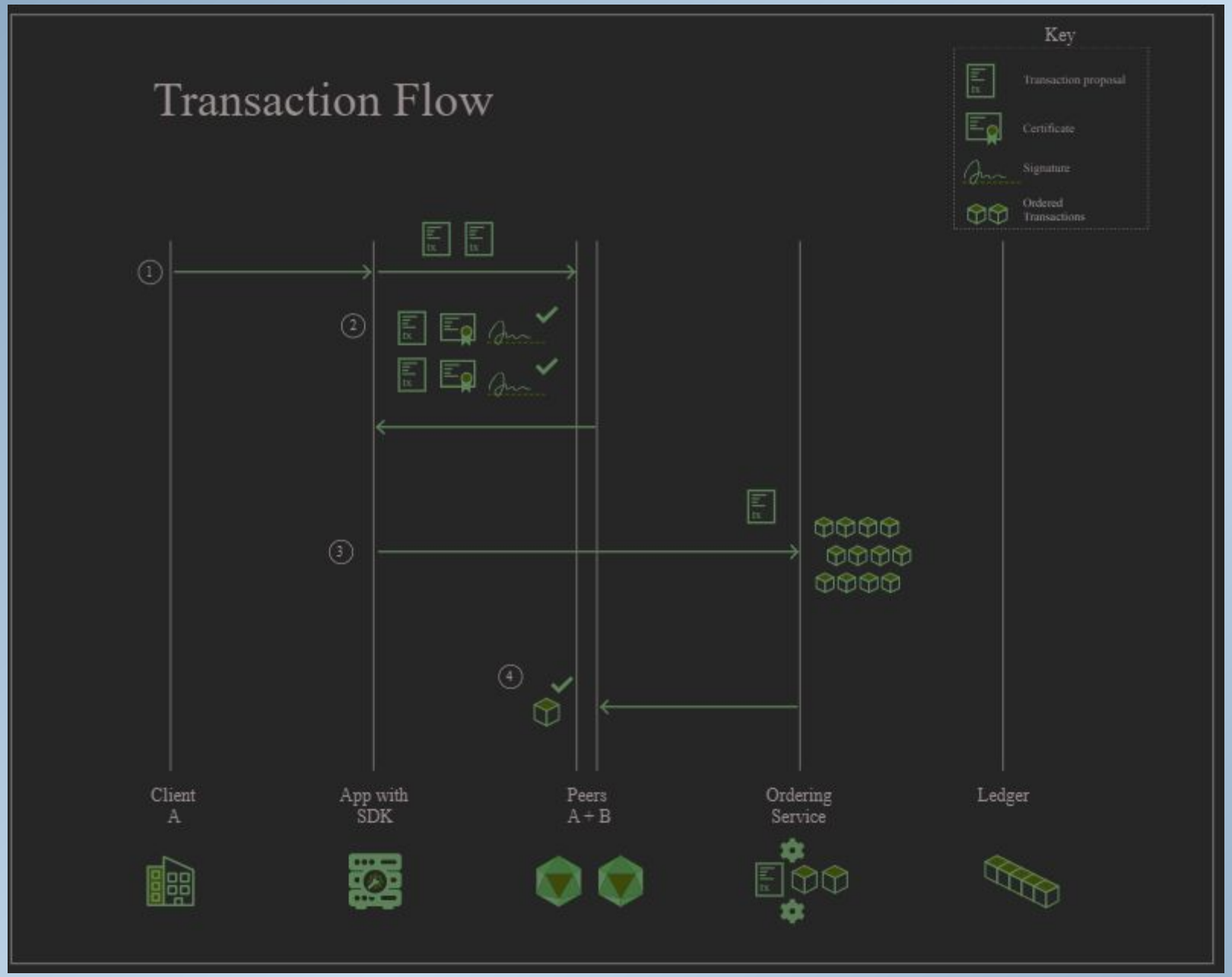
1. Client Application in using Hyperledger Fabric Client (HFC) SDK
2. Smart Contract implemented using chaincode – managing the World state



Fabric Ledger

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




Transaction Flow
Key
Transaction proposal
Certificate
Signature
Ordered Transactions
1
2
3
4
Client A
App with SDK
Peers A + B
Ordering Service
Ledger

Transaction flow

- On a Hyperledger Fabric network, the flow of data for queries and transactions is initiated by a client-side application by submitting a transaction request to a peer on a channel. The initial flow of data across the network is common to both queries and transactions:

1. Using APIs available in the SDK, a client application signs and submits a transaction proposal to the appropriate endorsing peers on the specified channel. This initial transaction proposal is a **request** for endorsement.
2. Each peer on the channel verifies the identity and authority of the submitting client, and (if valid) runs the specified chain code against the supplied inputs. Based on the transaction results and the endorsement policy for the invoked chain code, each peer returns a signed YES or NO response to the application. Each signed YES response is an **endorsement** of the transaction.
3. At this point in the transaction flow, the process diverges for **queries and transactions.** If the proposal called a query function in the chain code, the application returns the data to the client. If the proposal called a function in the chain code to update the ledger, the application continues with the following steps:

Transaction flow

4. The application forwards the transaction, which includes the read/write set and endorsements, to the **ordering service**.
5. The transaction is then relayed to the ordering service. All channel peers validate each transaction in the block by applying the chain code-specific Validation Policy and running a Concurrency Control Version Check.
    - Any transaction that fail the validation process is marked as invalid in the block, and the block is appended to the channel's ledger.
    - All valid transactions update the state database accordingly with the modified key/value pairs.

- The **gossip data dissemination protocol** continually broadcasts ledger data across the channel to ensure synchronized ledgers among peers.
- For more information, see Gossip data dissemination protocol in Hyperledger Fabric documentation.

Ordering service

- In other distributed blockchains, such as Ethereum and Bitcoin, there is no central authority that orders transactions and sends them out to peers.
- Hyperledger Fabric, the blockchain that the IBM Blockchain Platform is based on, work differently. It features a node called an **orderer**.
- Orderers also enforce basic access control for channels, restricting who can read and write data to them, and who can configure them.
- Remember that who is authorized to modify a configuration element in a channel.
- Fabric's design relies on **deterministic** consensus algorithms, any block validated by the peer is guaranteed to be final and correct.
- Ledgers cannot fork the way they do in many other distributed and permissionless blockchain

Ordering service

Orderers are key components in a network because they perform a few essential functions:

- It literally **orders** the blocks of transactions that are sent to the peers to be written to their ledgers, and this process is called "ordering". If these transactions were instead bundled and ordered at the peers themselves, it would increase the possibility of one peer writing a transaction to its ledger where another peer did not, creating a state fork.
- It maintains the **orderer system channel**, the place where the **consortium**, the list of peer organizations permitted to create channels, resides.
- It performs important identity validation checks. For example, if an organization tries to create a channel when it is not a member of the orderer's consortium, the request will be denied.
- Orderers also validate against behaviors in transaction channels, such as the permissions for changing a channel configuration.
- Hyperledger Fabric currently uses an implementation of the Raft protocol for its ordering service. For more information about the ordering service, see The Ordering Service.