HYPERLEDGER FABRIC

A Distributed Operating System For Permissioned Blockchains

MPYANA MWAMBA MERLEC

Intelligent Blockchain Engineering Lab

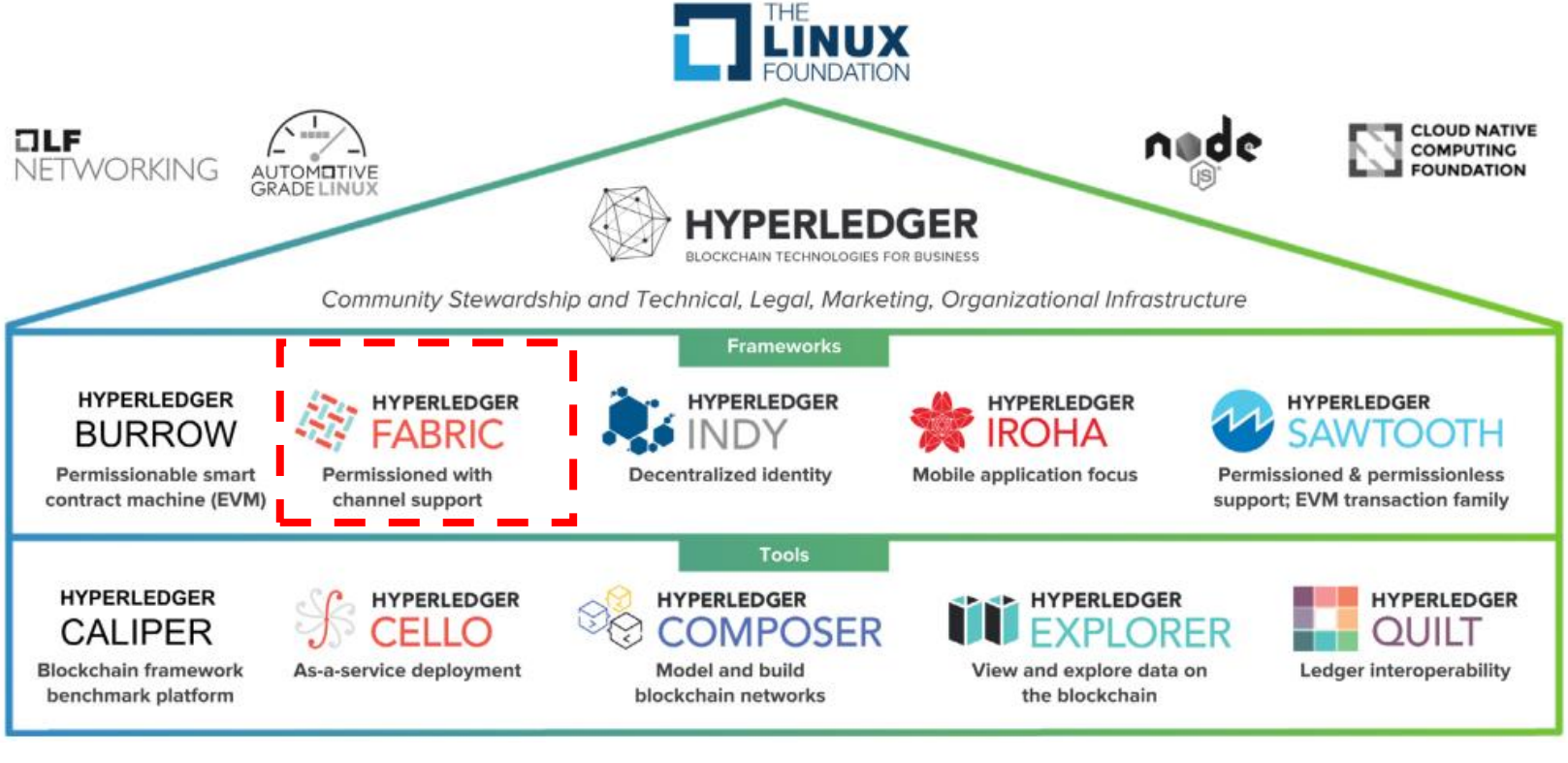
Department of Computer Science Engineering

Korea University

January 2019

Hyperledger Project Overview

- **Hyperledger Fabric** is a modular and extensible open-source system
  - For **deployment** and **operating** permissioned/private blockchains
  - One of the Hyperledger projects hosted by the Linux Foundation – www.hyperleder.org



Hyperledger Projects Umbrella Strategy by the Hyperledger Architecture Working Group (HAWG)

Introduction

- First truly extensible blockchain system for running distributed applications
  - It runs Dapps written in standard, general-purpose programming languages
  - Without systematic dependency on native cryptocurrency
  - It supports **modular consensus protocols**
    - Allowing the system to be tailored to particular use cases and trust models
  - It realizes the permissioned model using a portable notion of membership
    - Which may be integrated with industry-standard identity management (i.e. AD or LDAP)

For flexibility, Fabric introduces an entirely novel blockchain design and reinvent the way blockchains cope with nondeterminism, resource exhaustion, and performance attacks.

*Lightweight Directory Access Protocol

Introduction

- Blockchains depart for traditional ***State-Machine Replication (SMR)***
  - With Byzantine faults in important ways :

(1) Not only one, but distributed applications run concurrently

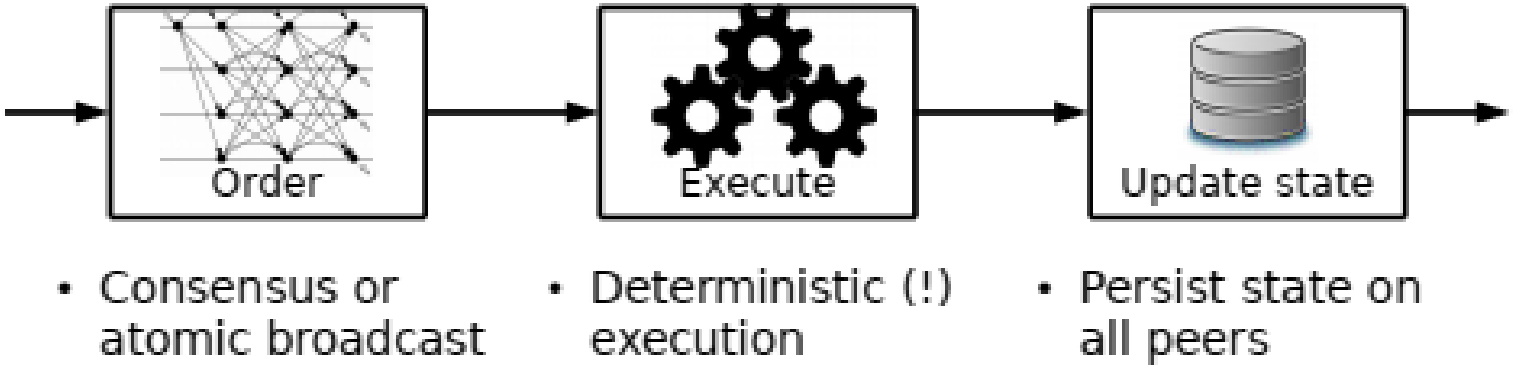
(2) Applications may be deployed dynamically and by anyone

(3) Application code is untrusted, potentially even malicious

  - To achieve these differences new designs are required !!!

Order-Execute Architecture for Blockchains

- Many existing smart-contract blockchains' implementations follow a SMR approach
  - Known as "***active replication***"



Order-Execute (OE) architecture in replicated services

(1) A protocol for consensus or atomic broadcast, first orders the transactions and propagated them to all peers

(2) Each peer executes the transaction sequentially and all transactions must be deterministic

Limitation of Order-Execute (OE) based Approach

- Prior permissioned Blockchains suffer from many limitations
  - Adoption of the ***Order-Execute Architecture (OEV)*** with an additional transaction validation step

Consensus is hard-coded within the platform

- No “one-size-fits-all” (BFT) consensus protocols

Non-flexible trust model of transaction validation

- Determined by the consensus protocol
- Cannot be adapted to smart contracts or apps’ requirements

Smart Contracts Programming Languages

- Fixes, non-standard or domain-specific language
- Obstructs wide-spread adoption
- May lead to programming errors

Sequential execution of all transactions

- Limits performance & scalability
- Complex measures are needed to prevent DoS attacks

Transactions must be deterministic

- Can be difficult to ensure programmatically

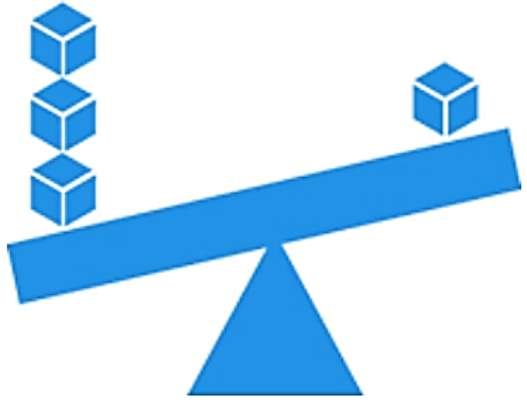
Confidentiality

- Since every SC runs on all peers
- Prohibits the dissemination of contract code and state to subset of peers

Nonfunctional Requirements

- **Performance**
  - The amount of data being shared
  - Number and location of peers
  - Latency and throughput
  - Batching characteristics
- **Security**
  - Type of data being shared, and with whom
  - How is identity achieved
  - Confidentiality of transaction queries
  - Who verifies (endorses) transactions
- **Resiliency**
  - Resource failure
  - Malicious activity
  - Non-determinism

Consider the trade-offs between performance, security, and resiliency!



Hyperledger Fabric Overview

It's a blockchain framework implementation and one of the Hyperledger projects hosted by The Linux Foundation.

- View as a **Distributed Operating System For Permissioned Blockchains**
  - Proposed as a foundation for developing blockchain applications or solutions with a modular architecture
  - It allows components, such as consensus and membership services, to be plug-and-play

Hyperledger Fabric Overview

- Fabric typically executes transactions before reaching final agreement on their order
  - Its design combines two well-known approaches to replication

Passive or Primary-backup Replication

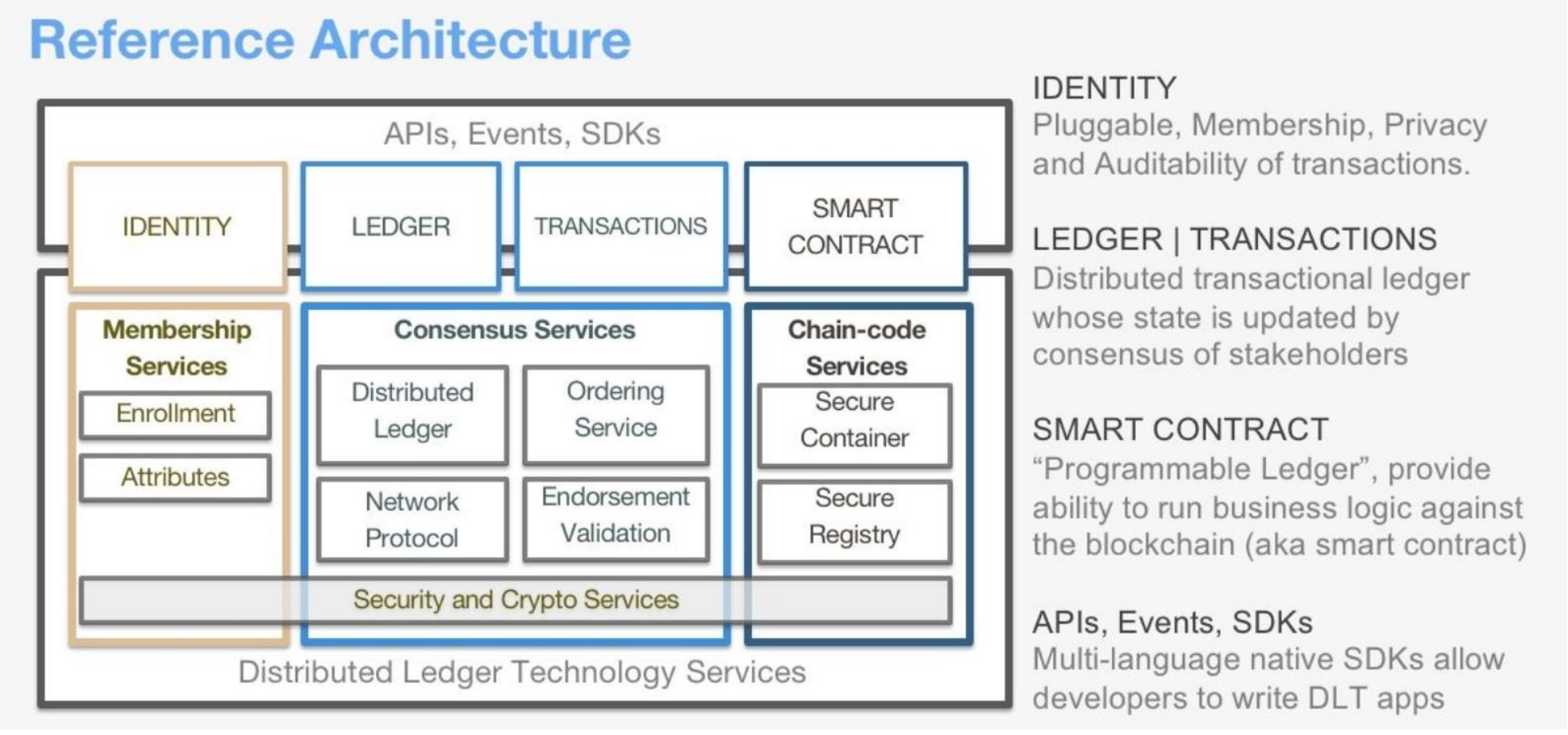
- Often used in distributed Databases
- Middleware-based asymmetric update processing
- Ported to untrusted environment with Byzantine Faults

Active Replication

- Updates the ledger state only after reaching consensus on a total order among transactions
- in a deterministic validation step executed by each peer individually
  - ❖ *Allows Fabric to respect application-specific trust assumptions according to the transaction endorsement*

In Fabric, every transaction is executed (endorsed) only by a subset of the peers, which allows for parallel execution and addresses potential non-determinism, drawing on "*execute-verify*" BFT replication.

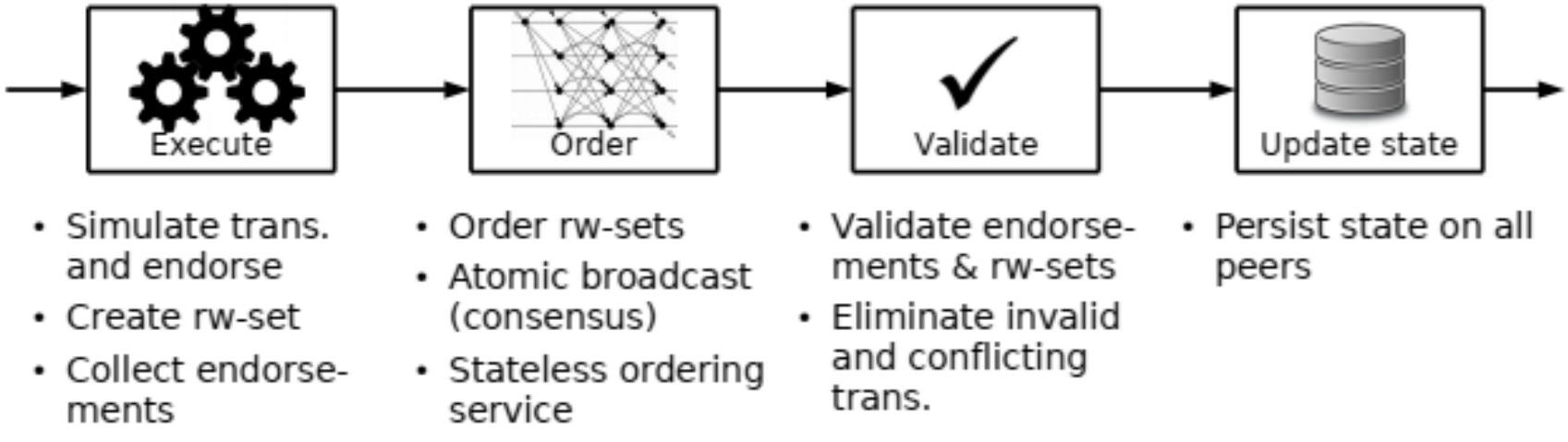
Hyperledger Fabric Reference Architecture



Source: Gaur, Nitin, et al. Hands-On Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer. Packt Publishing Ltd, 2018.

Execute-Order-Validate Scheme

- Fabric architecture follows a novel ***Execute-Order-Validate*** paradigm
  - For distributed execution of untrusted code in an untrusted environment
  - Separates transactions flow into 3 steps that can be run on different entities in a system



Execute-Order-Validate architecture of Fabric

Adaptive Consensus Protocols

- Fabric is the first blockchain system that introduced **pluggable consensus**
  - Previously, all blockchain systems, permissioned or not, came with a hard-coded consensus protocol

Proof of work

Require validators to solve difficult cryptographic puzzles.

PROS:
- Works in public/untrusted networks

CONS:
- High energy consumption,
- Slow transactions confirmation

e.g. : Bitcoin, Ethereum, EOS

Proof of stake

Require validators to hold currency in escrow

PROS:
- Works in public/untrusted networks

CONS:
- Requires intrinsic (crypto)currency
- "Noting at stake" problem

e.g. : Corda, PeerCoin, Nxt

Solo

Validators apply received transactions without consensus

PROS:
- Very Quick; suitable for development

CONS:
- Nos consensus (single node, development)
- Can lead to divergent chains

e.g. : Hyperledger Fabric V1

PBFT-based

Practical Byzantine Fault-Tolerance imple mentations

PROs:
- Reasonable efficient
- Tolerant against malicious peers *(n > 3f peers)*

CONS:
- Validators should be known and totally connected

e.g. : Hyperledger Fabric V 0.6

Proof of Elapsed Time

Wait time in a trusted execution environment randomizes block generation

PROS:
- Efficient

CONS:
- Currently tailored towards one vendor

e.g. : Hyperledger Sawtooth

Kafka/ Zookeeper

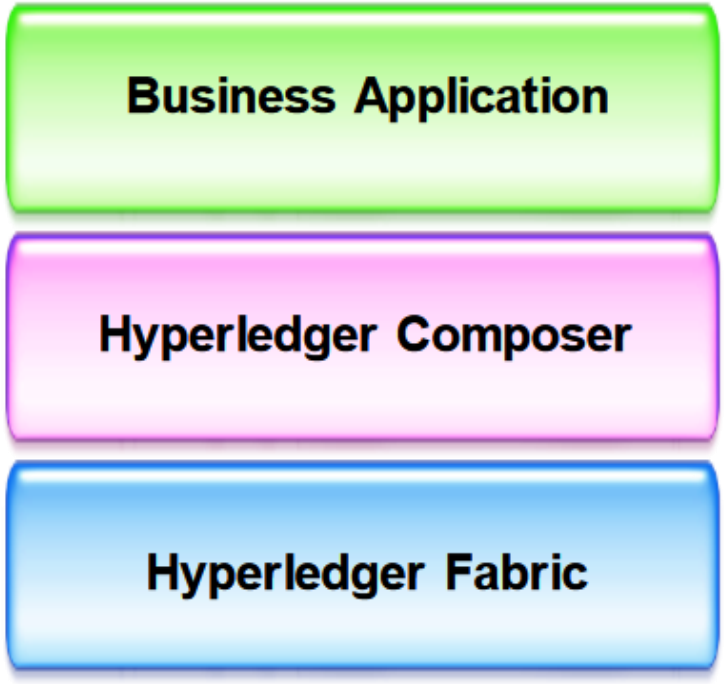
Ordering service distributes blocks to peers

PROS:
- Efficient and Crash Fault tolerant

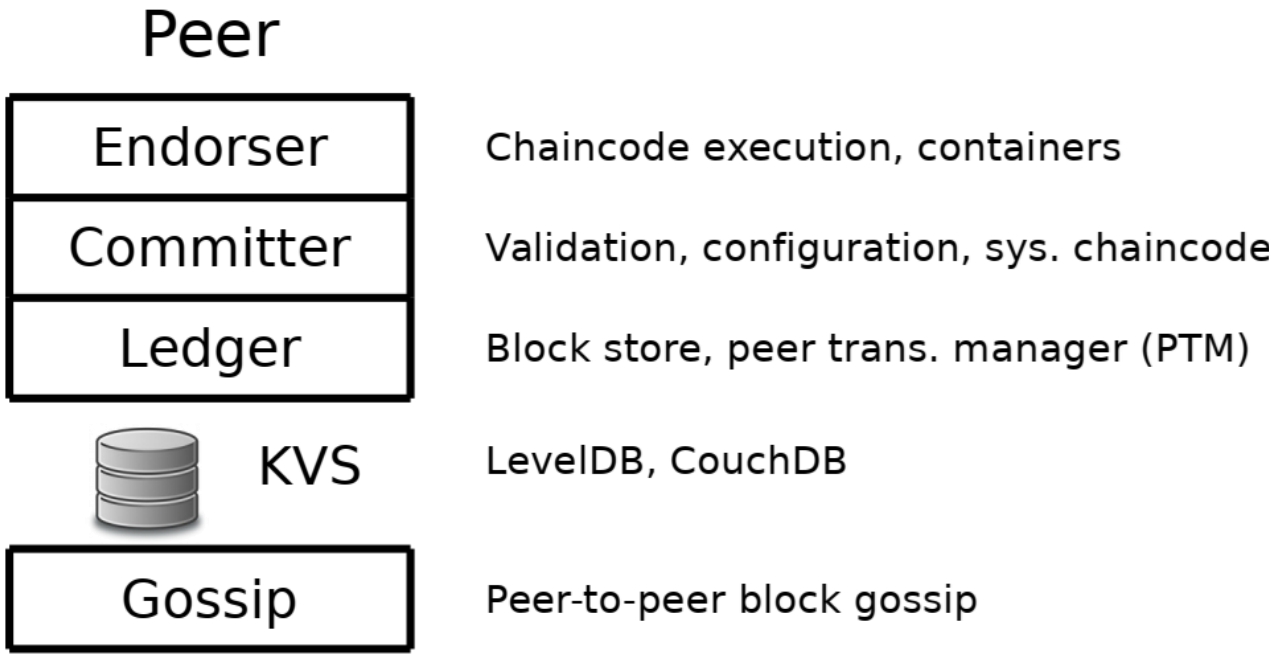
CONS:
- Doesn't guard against malicious activity

e.g. : Hyperledger Fabric V1

Hyperledger Fabric – Peer Node



Hyperledger Platform Layer Architecture



Peer Node Layer Architecture

*KVS – Key Value Store

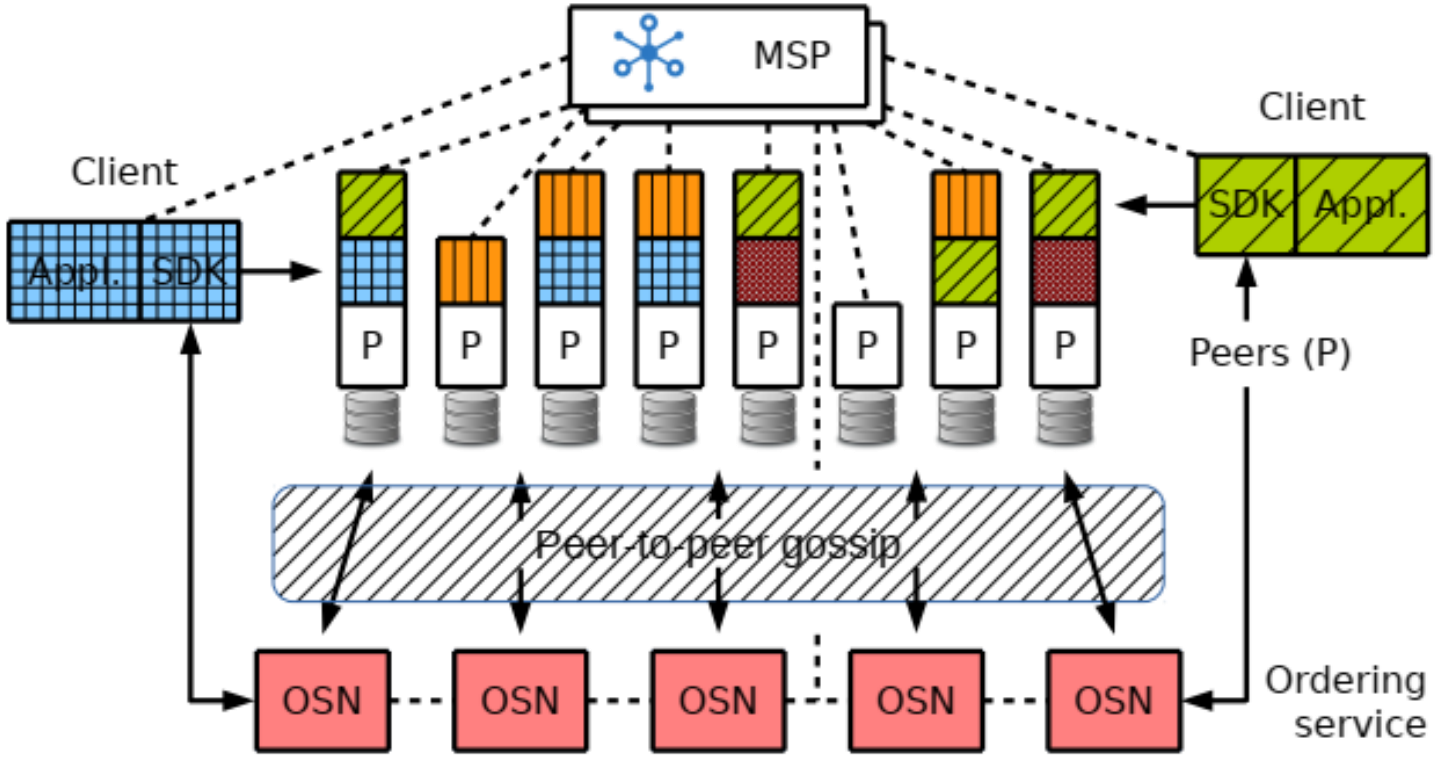
Fabric Architecture Overview

- Fabric architecture novelty lies on a **hybrid replication** design approach
  - mixes passive replication in Byzantine model and execute-order-validate paradigm
  - Contains modular building blocks for each of the following components:

Ordering Service	▪ Automatically broadcasts state updates to peers and establishes consensus on the order of transactions.
Membership Service (MSP)	▪ Responsible for associating peers with cryptographic identities. ▪ It maintains the permissioned nature of Fabric.
P2P Gossip Service	▪ Disseminates the blocks output by ordering service to all peers.
Smart Contracts	▪ SCs in Fabric run within container environment for isolation. ▪ Can be written in standard programming languages ▪ Don't have direct access to the ledger state
Ledger	▪ Maintained by each peer locally in form of the append-only blockchain and as a snapshot of the most recent state in a key-value store

Fabric Architecture Overview

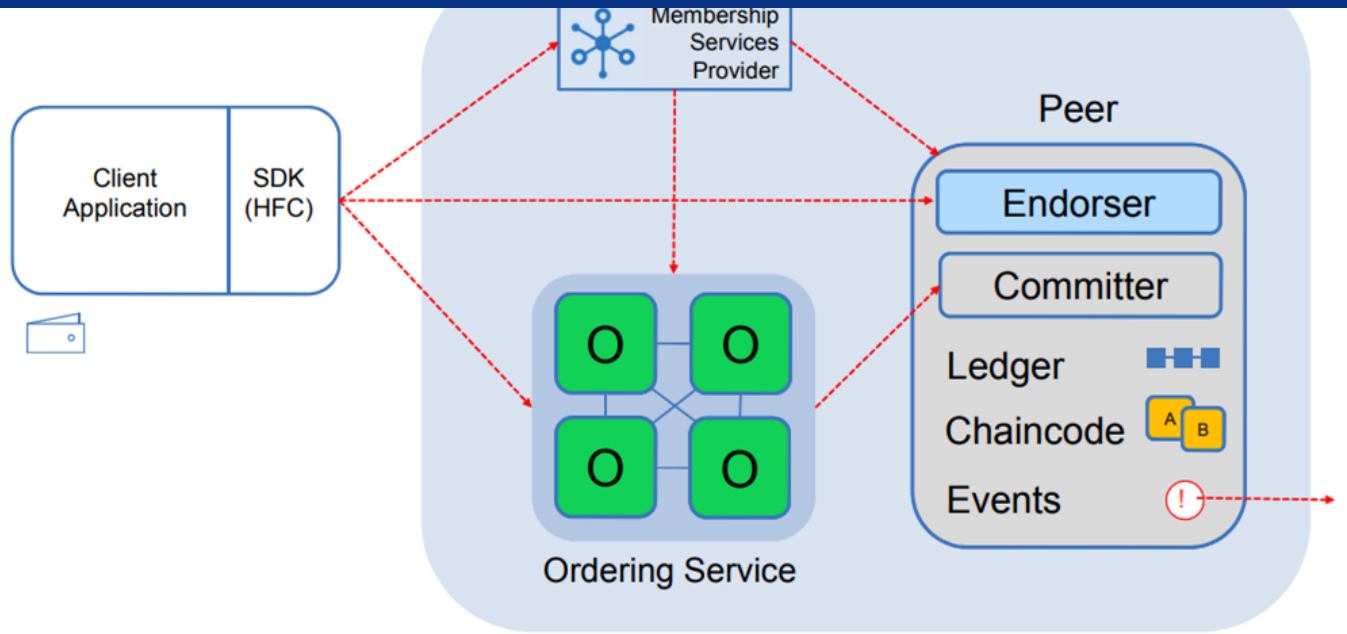
- Fabric architecture novelty lies on a ***hybrid replication*** design approach
  - mixes passive replication in Byzantine model and *execute-order-validate* paradigm



A Fabric Network with Federated MSPs and Running Multiple (differently shared and colored) Chaincodes, Selectively installed on peers according to policy

Source: Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018.

Fabric Architecture Overview



Hyperledger Fabric V1.0 System Architecture

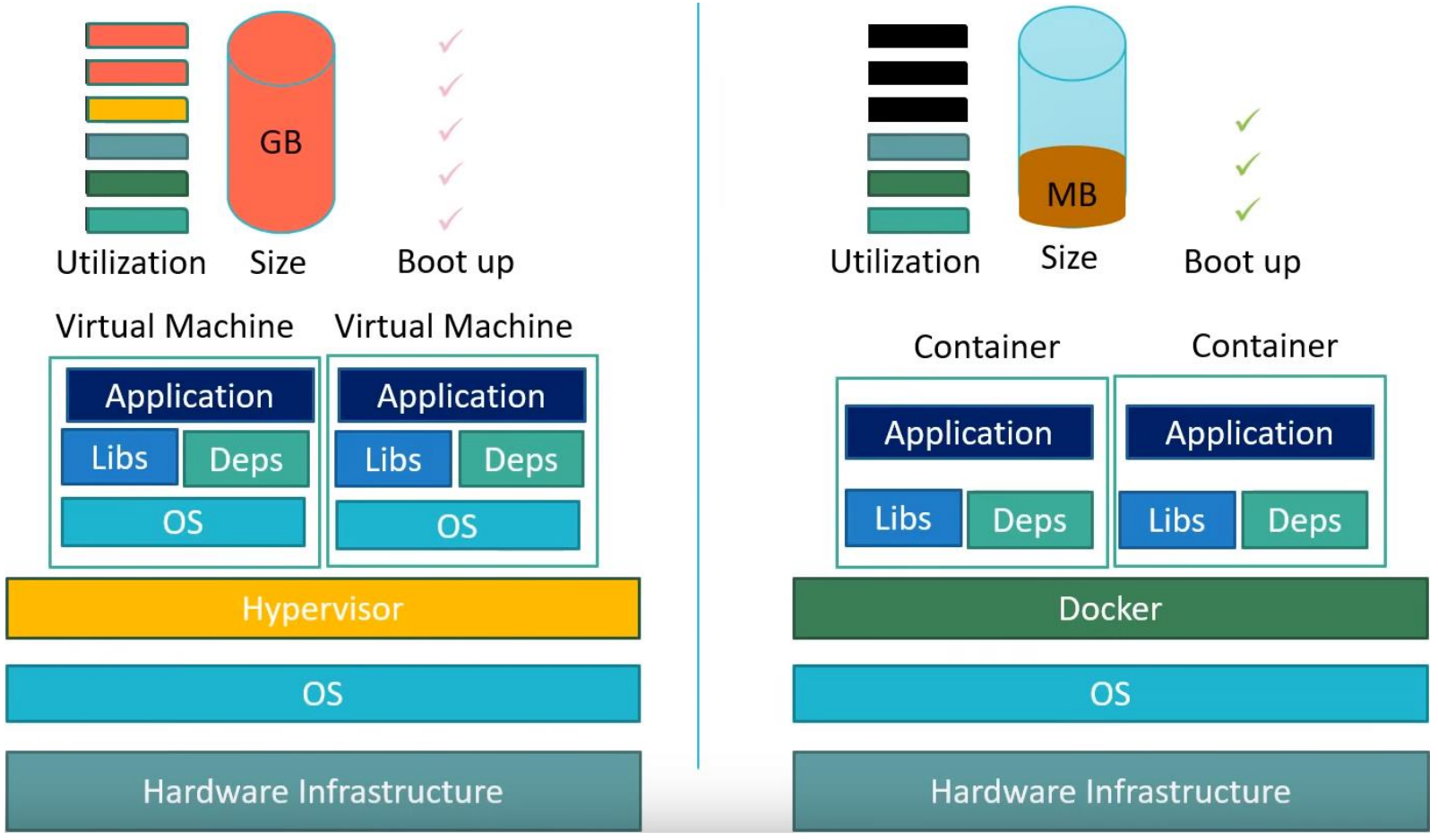
Events – Creates notifications of significant operations on the blockchain (e.g. a new block), as well as notifications related to smart contracts

	Committing Peer: Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).
	Endorsing Peer: Specialized committing peer that receives a transaction proposal for endorsement, responds granting or denying endorsement. Must hold smart contract
	Ordering Nodes (service): Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes. Does not hold smart contract. Does not hold ledger.

*HFC – Hyperledger Fabric Client

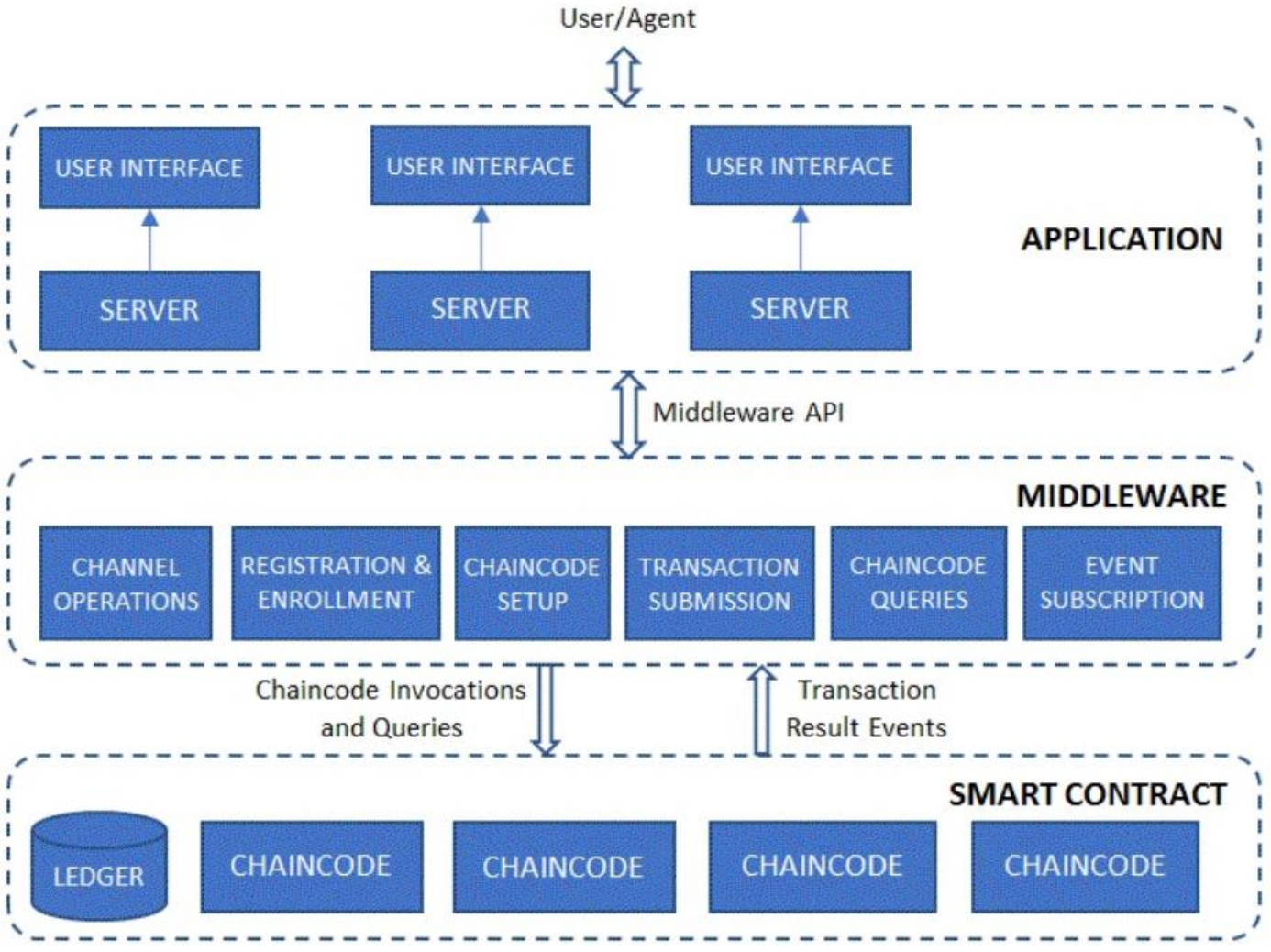
Chaincode Execution Environment

- **Chaincode** (a.k.a Smart Contract) Fabric run within container environment for isolation



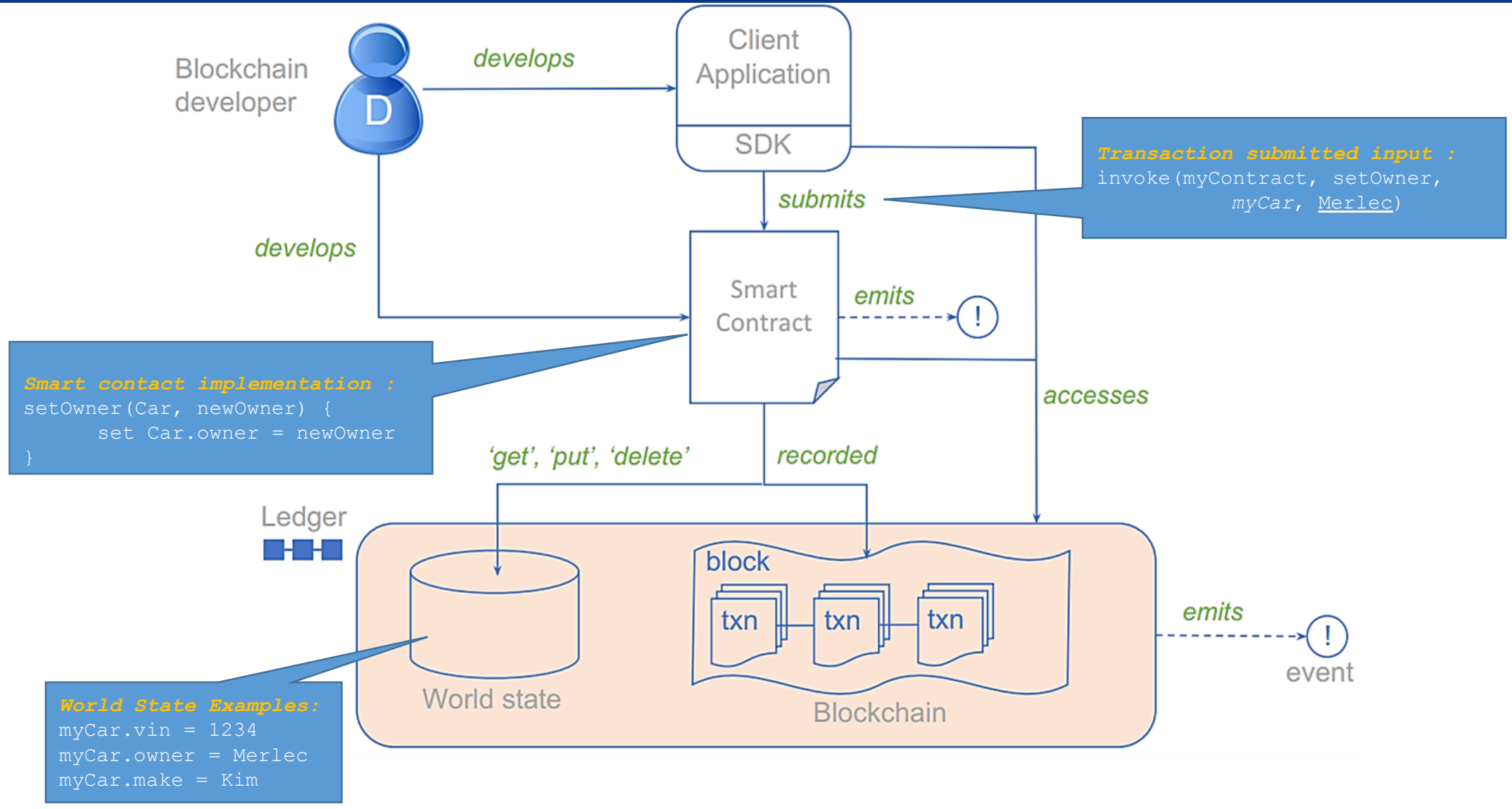
Source: https://youtu.be/rmf04yII2K0

Typical 3-Layer Architecture of a Fabric App.

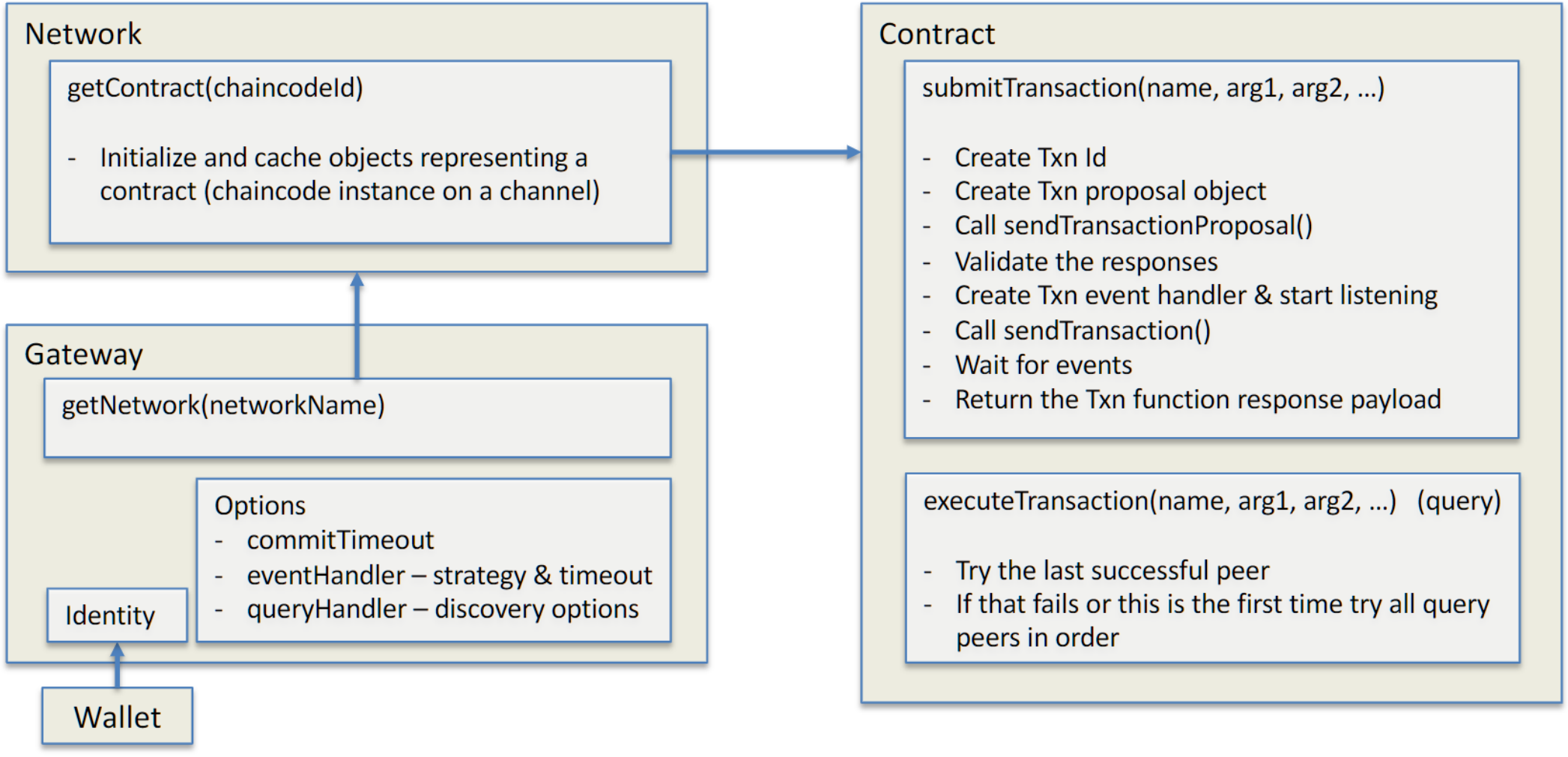


Source: Gaur, Nitin, et al. Hands-On Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer. Packt Publishing Ltd, 2018.

How Applications interact with the ledger



Network, Gateway and Smart Contracts

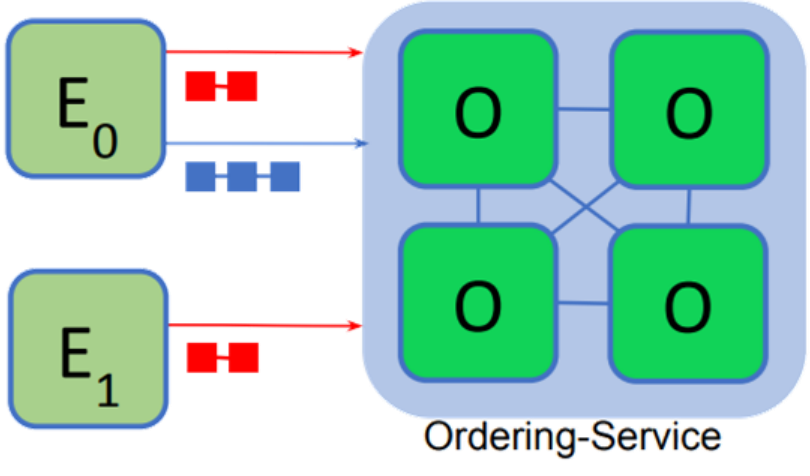


https://jira.hyperledger.org/projects/FABN/issues/FABN-692?filter=allissues

IBM

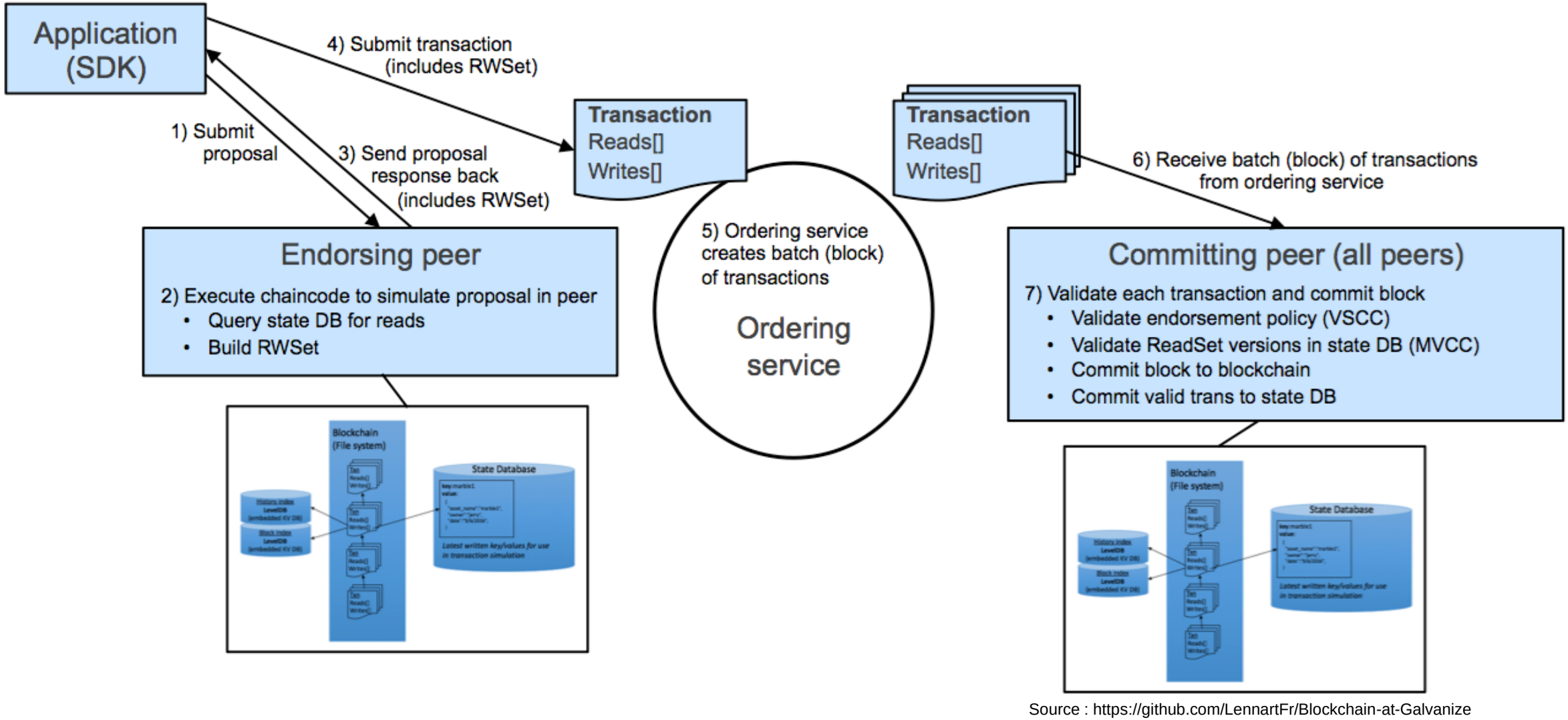
Channels

Separate channels isolate transactions on different ledgers



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

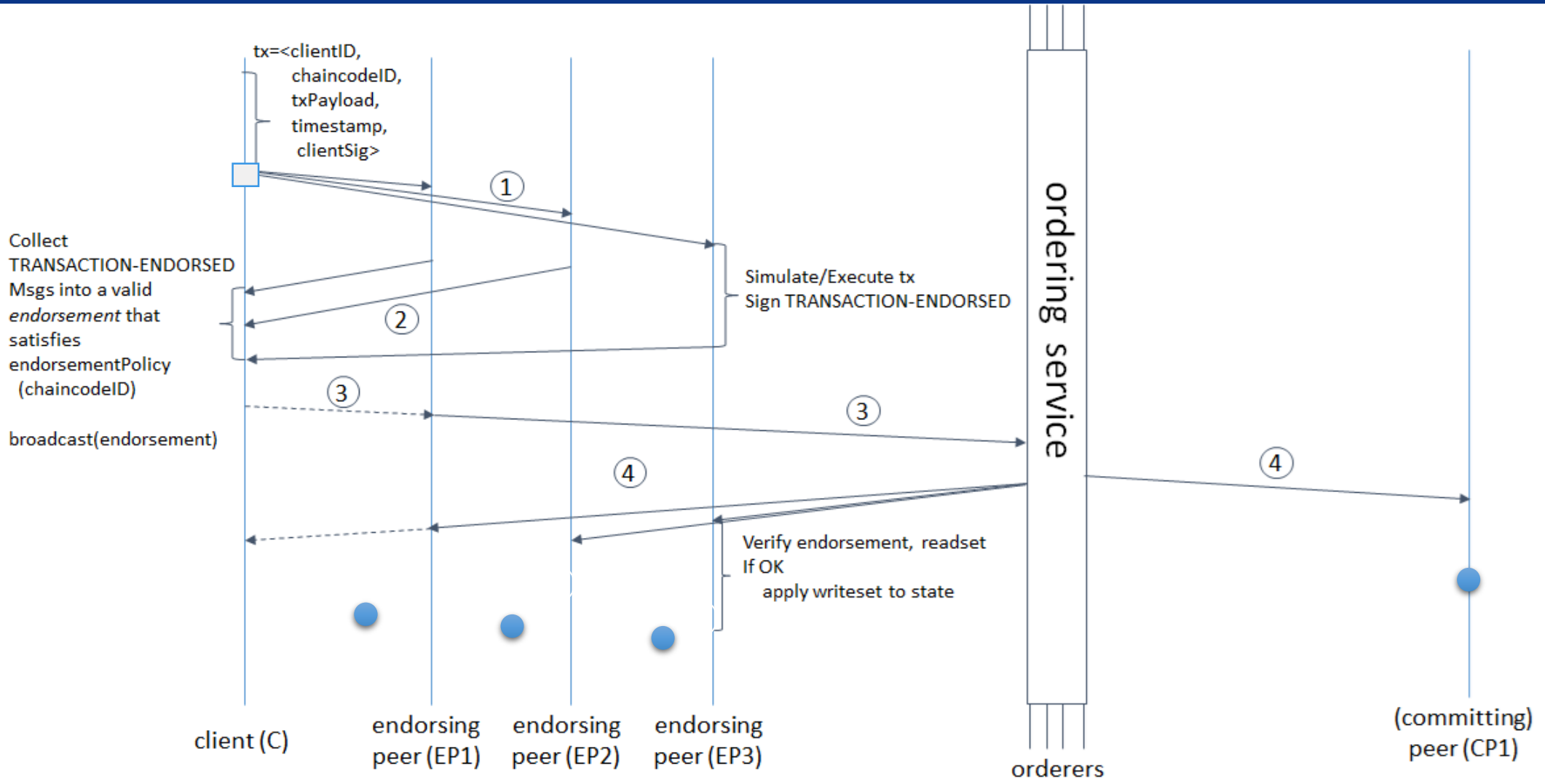
Hyperledger Fabric Working Principle



Source : https://github.com/LennartFr/Blockchain-at-Galvanize

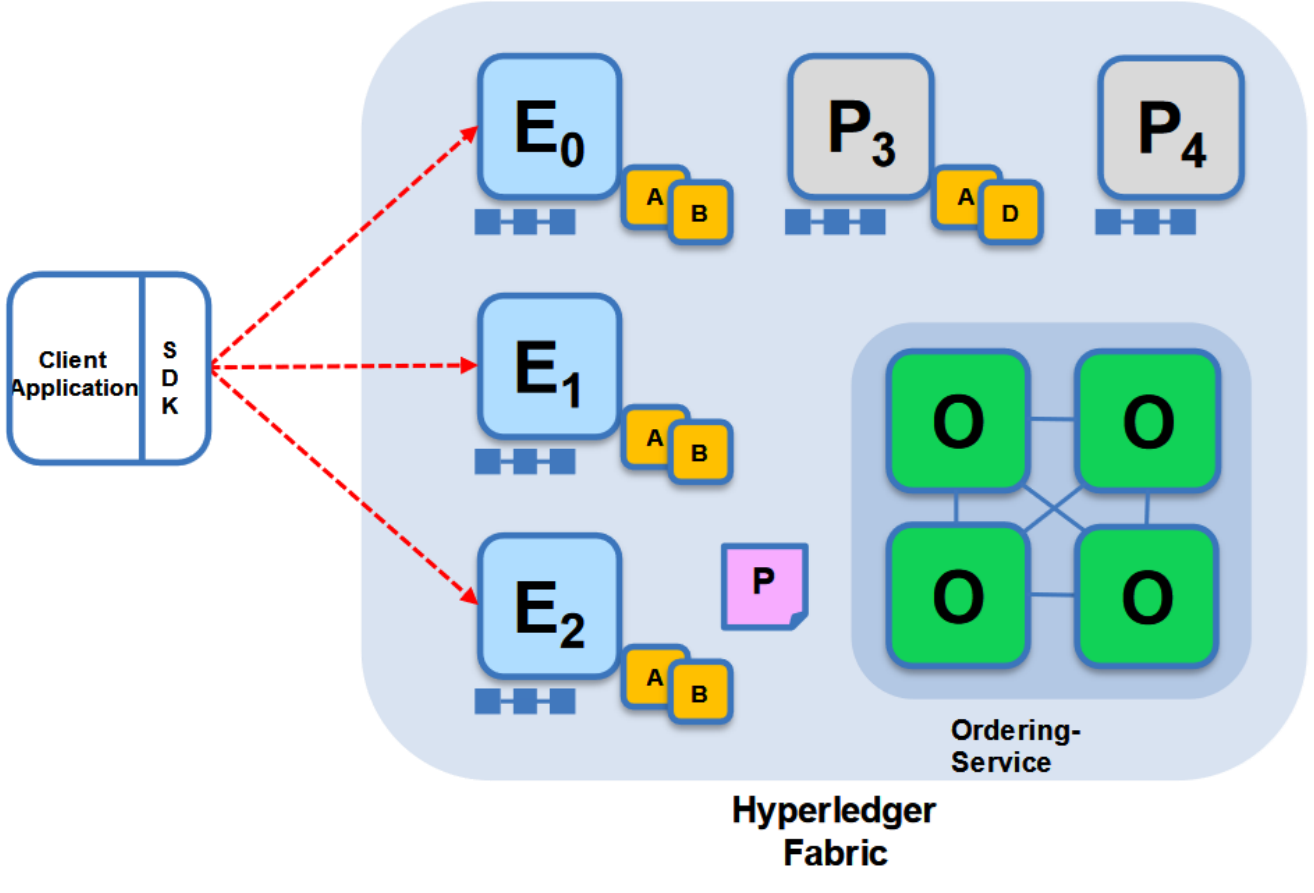
Execution Phase in Detail

Execution Phase



Fabric High Level Transaction Flow

Step 1/7 – Proposal Transaction

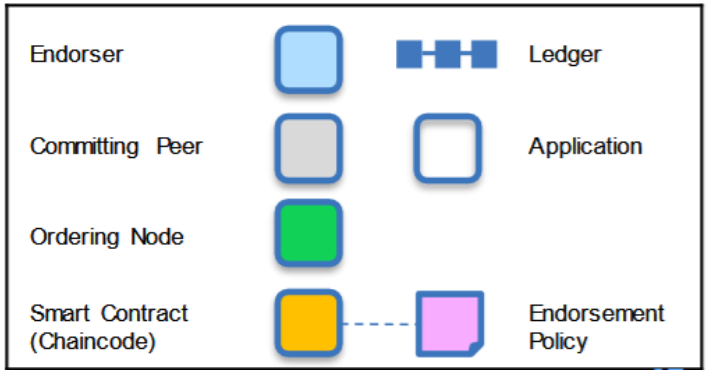


Application proposes transaction

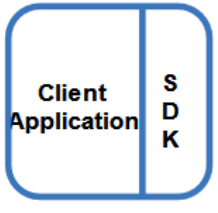
Endorsement policy:
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

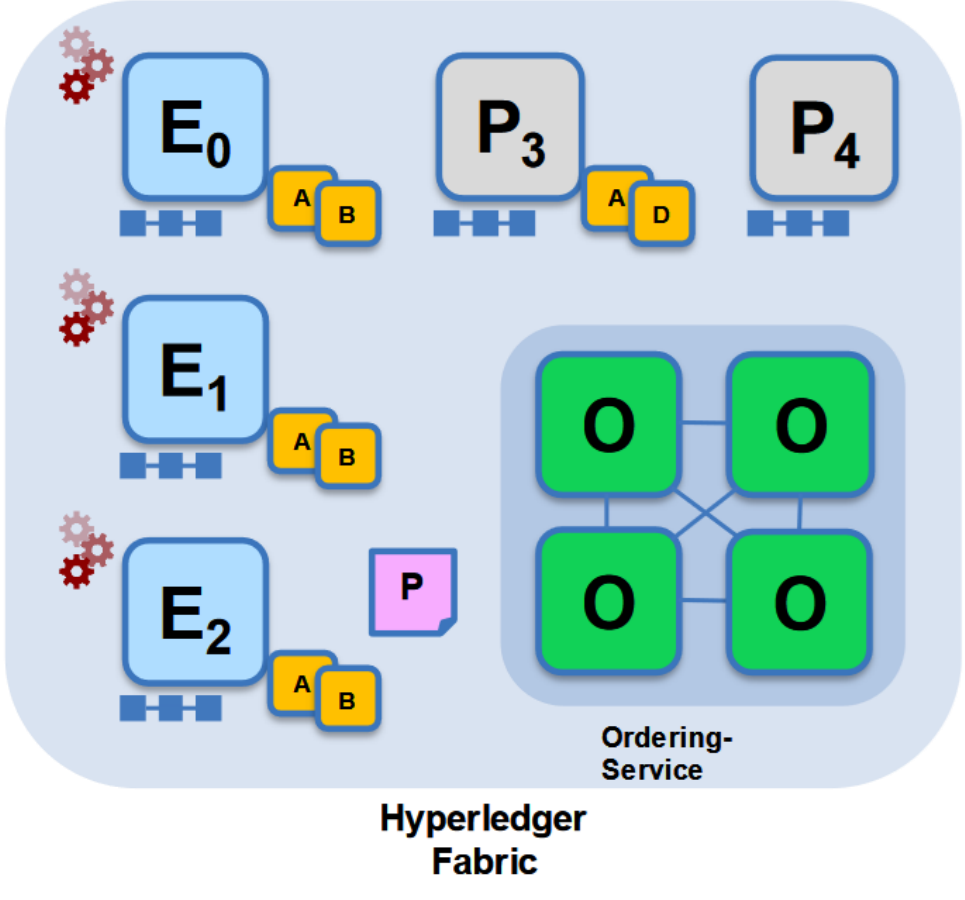
Client application submits a transaction proposal for Smart Contract A. It must target the required peers {E_0, E_1, E_2}.

Key:



Step 2/7 – Execute Proposal





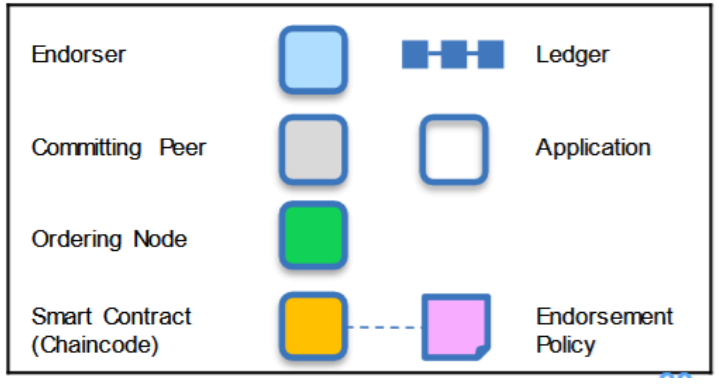
Endorsers Execute Proposals

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

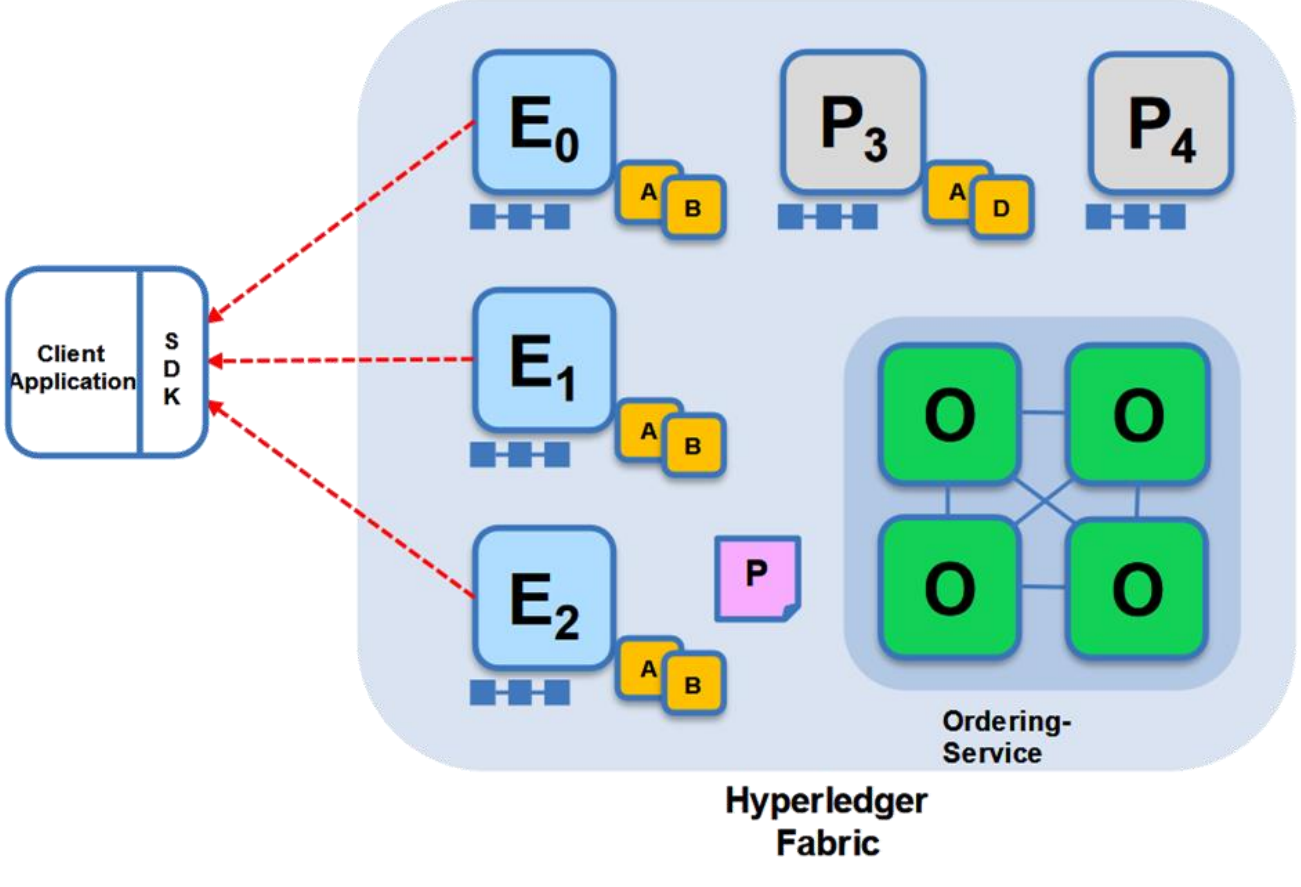
Each execution will capture the set of Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed and encrypted.

Key:



Step 3/7 – Proposal Response

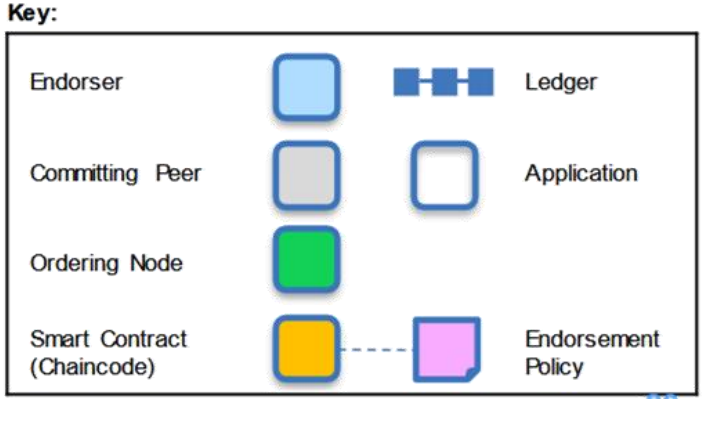


Application receives responses

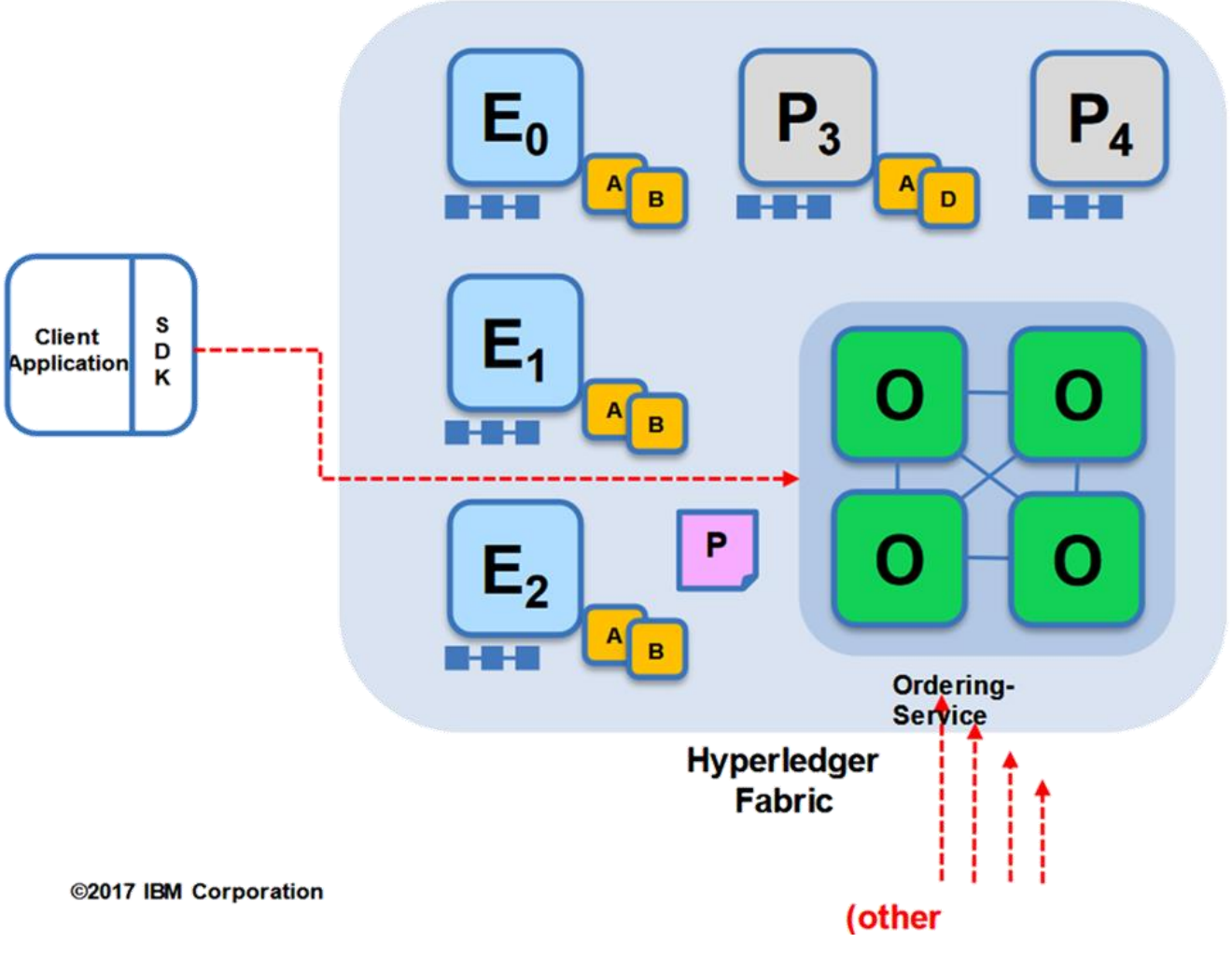
RW sets are asynchronously returned to application.

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.



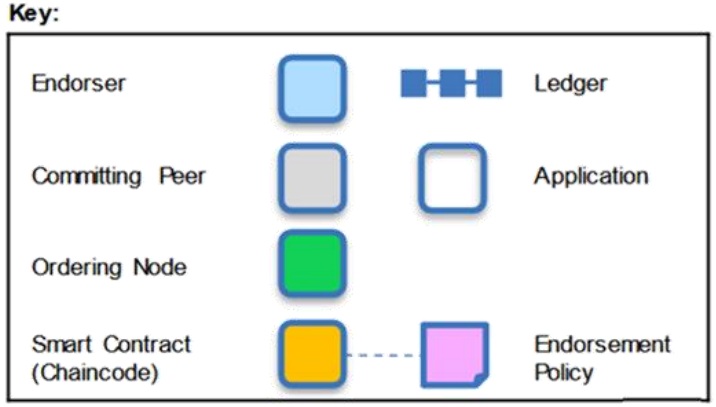
Step 4/7 – Order Transaction



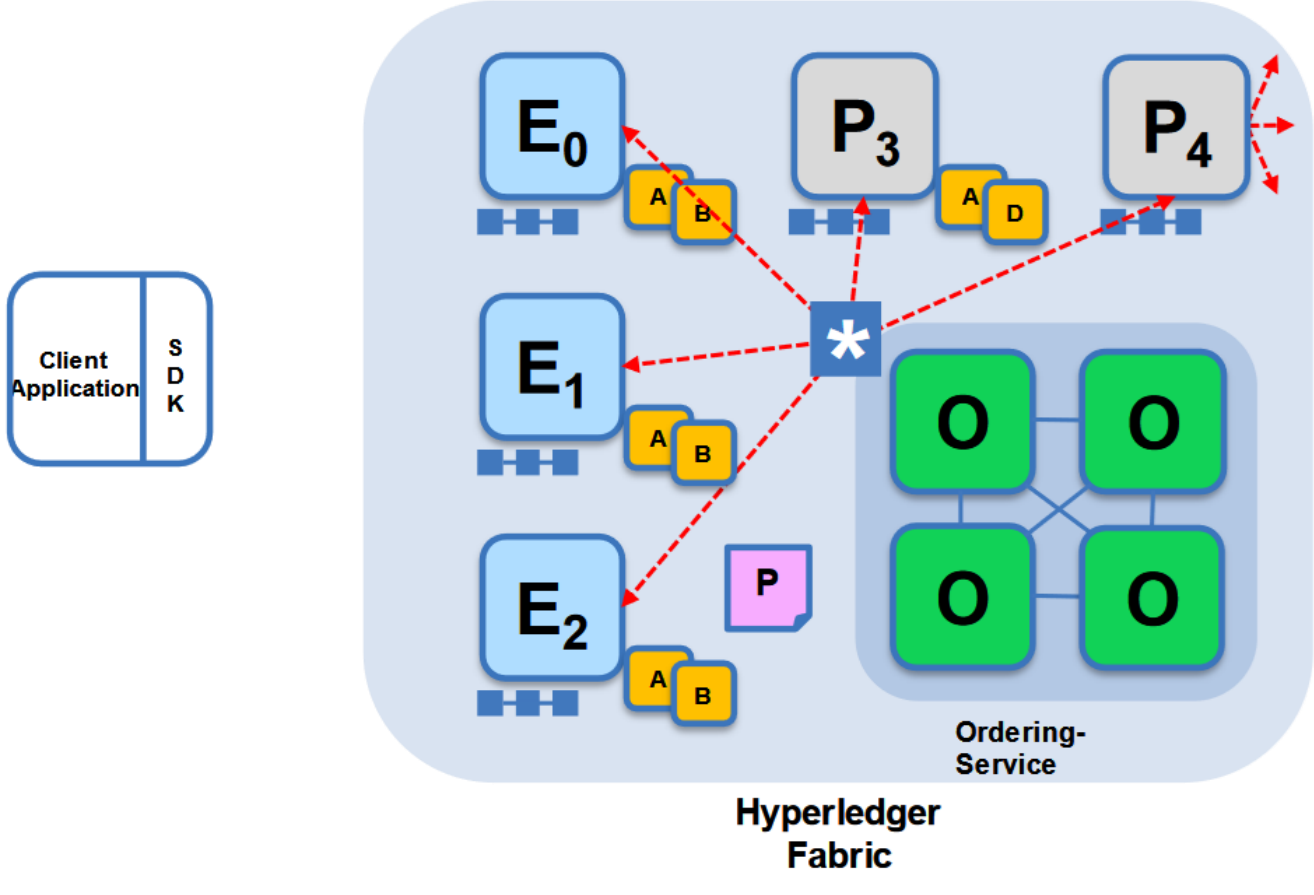
Application submits responses for ordering

Application submits responses as a transaction to be ordered.

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



Step 5/7 – Deliver Transaction

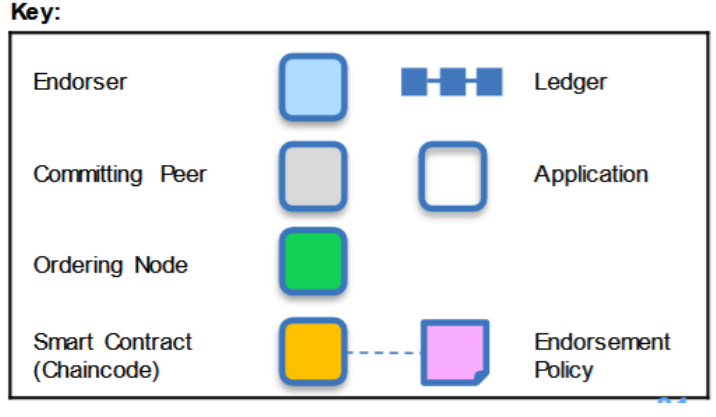


Orderer delivers to all committing peers

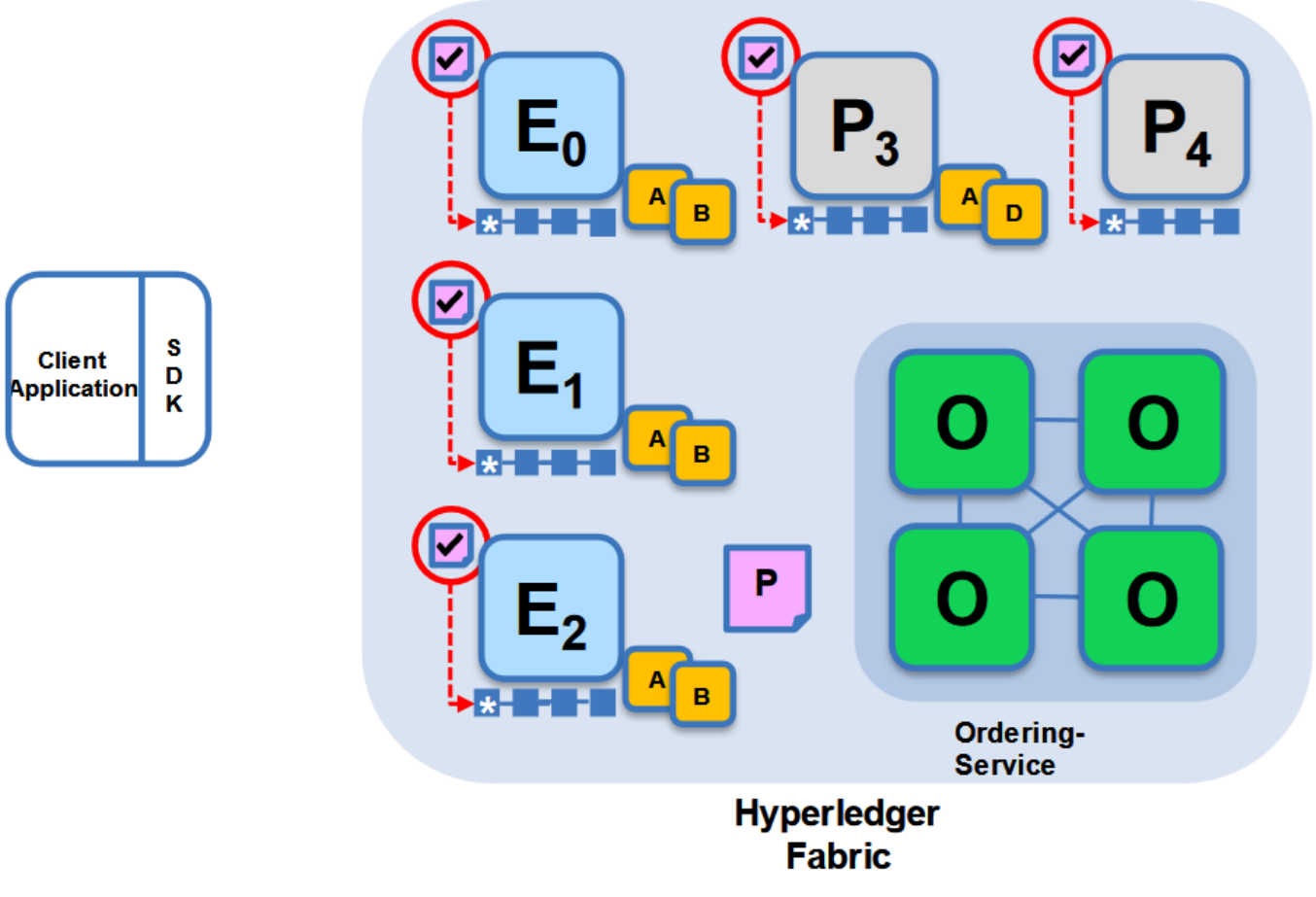
Ordering service collects transactions into proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown).

Different ordering algorithms available:

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



Step 6/7 – Validate Transaction

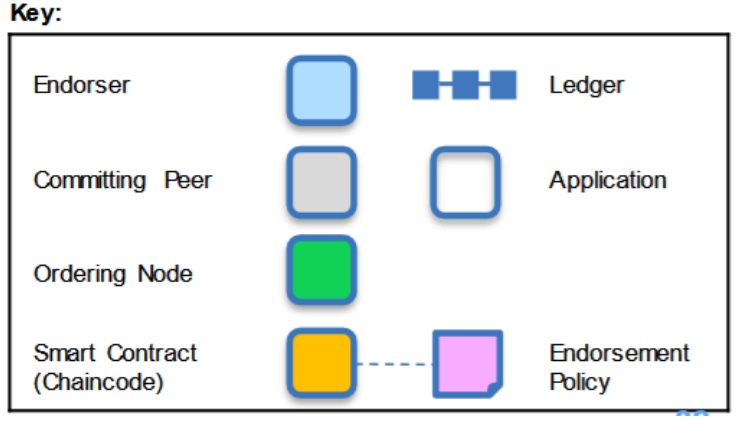


Committing peers validate transactions

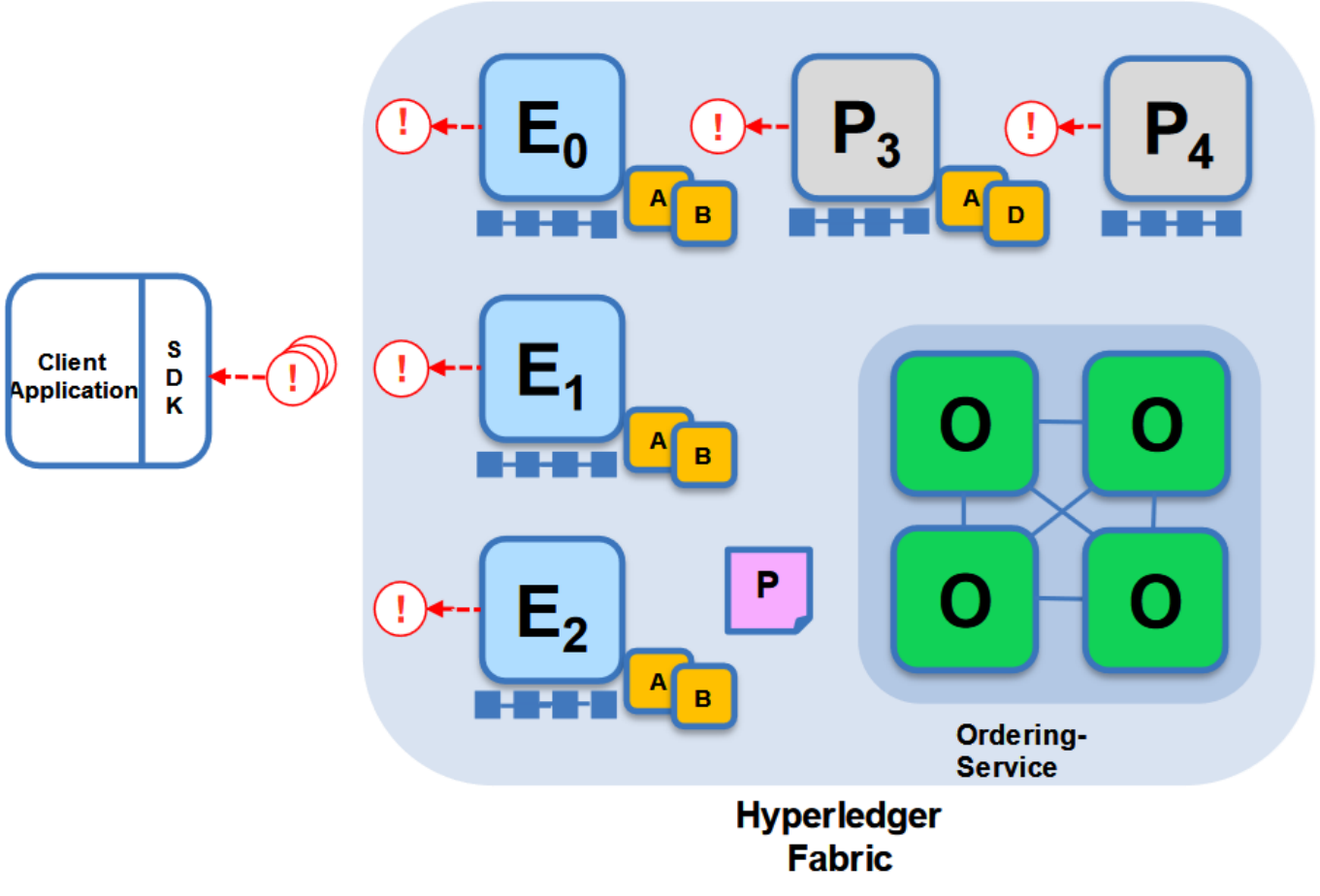
Every committing peer validates against the endorsement policy. Also check RW sets are still valid for current world state.

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

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



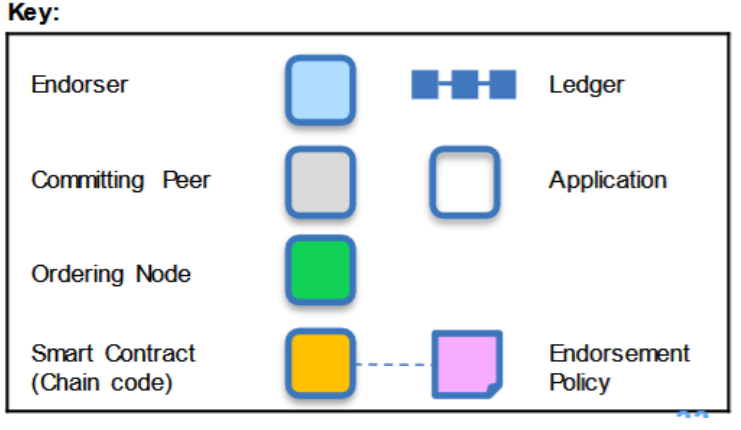
Step 7/7 – Notify Transaction



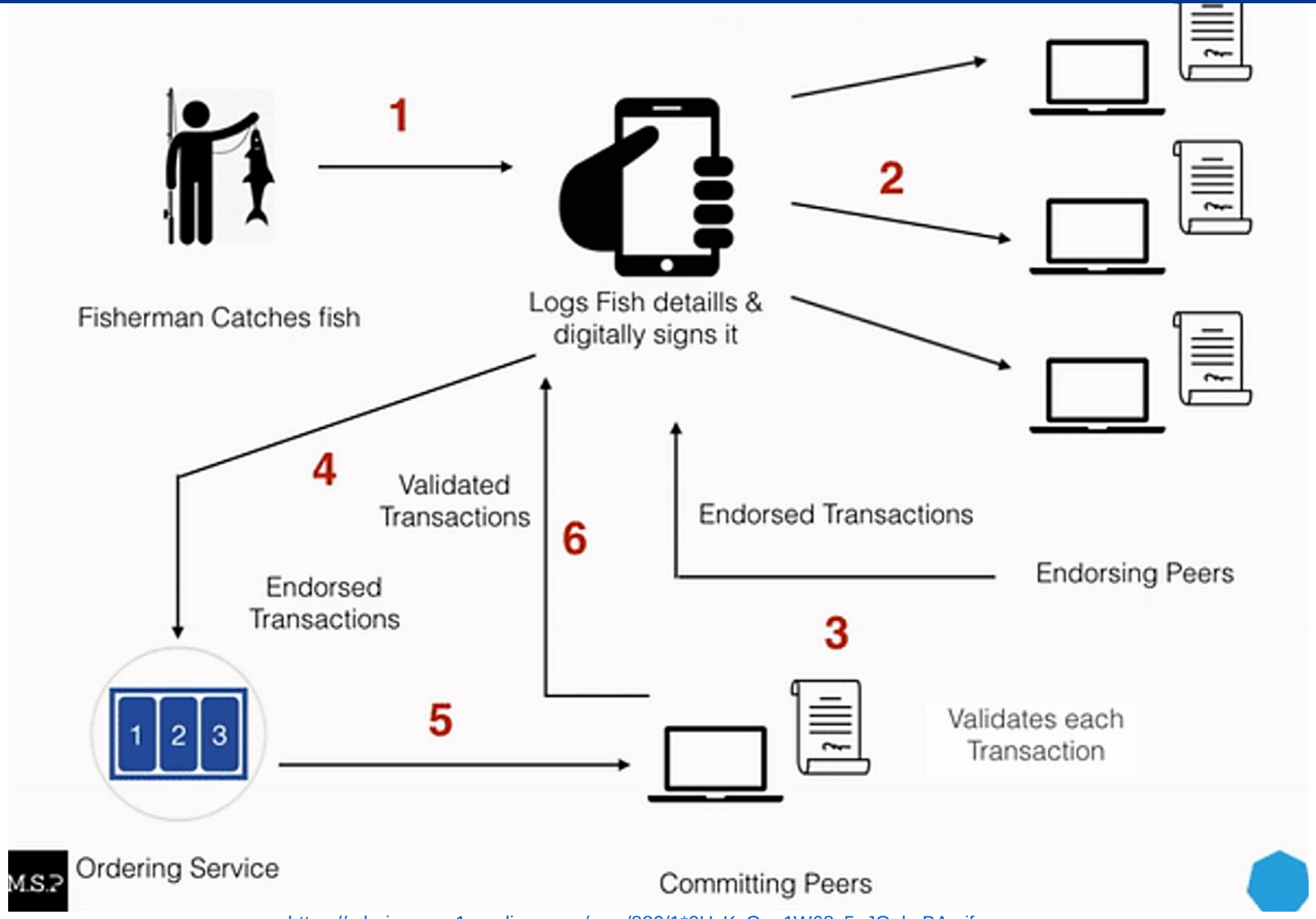
Committing peers notify applications

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

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

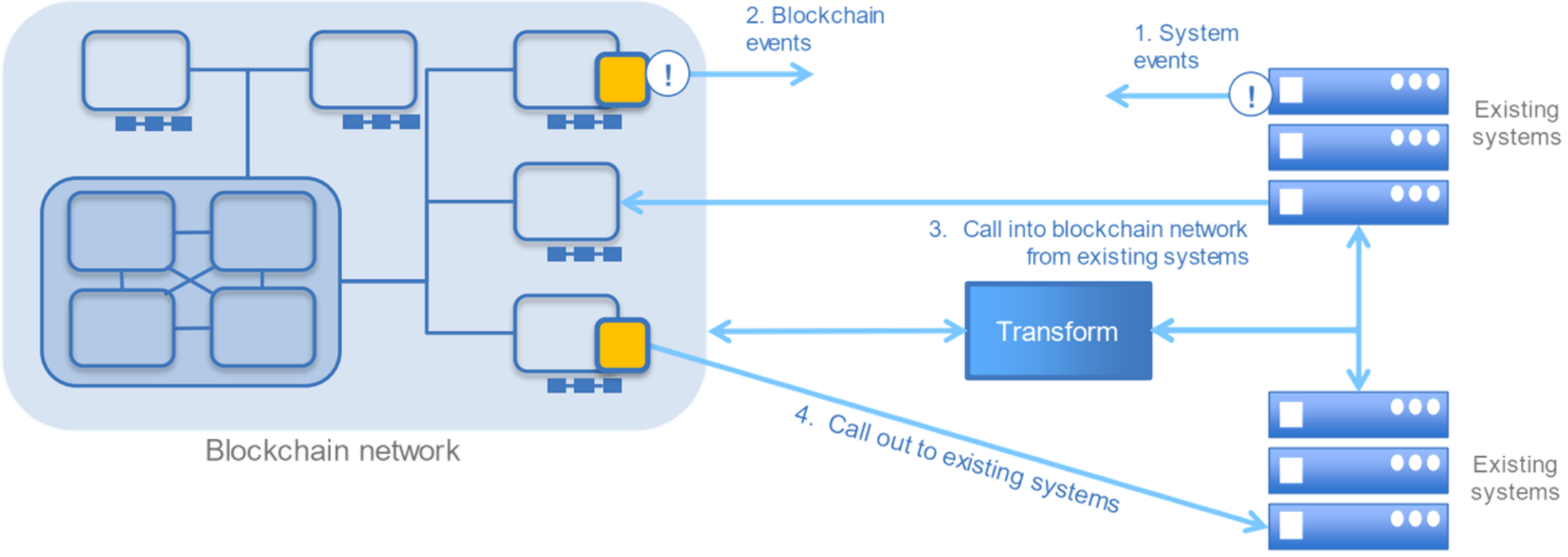


Hyperledger Fabric Transaction Flow illustration



https://cdn-images-1.medium.com/max/880/1*0HyKsOag1W68z5aJOahcBA.gif

Hyperledger Integration with Existing Systems



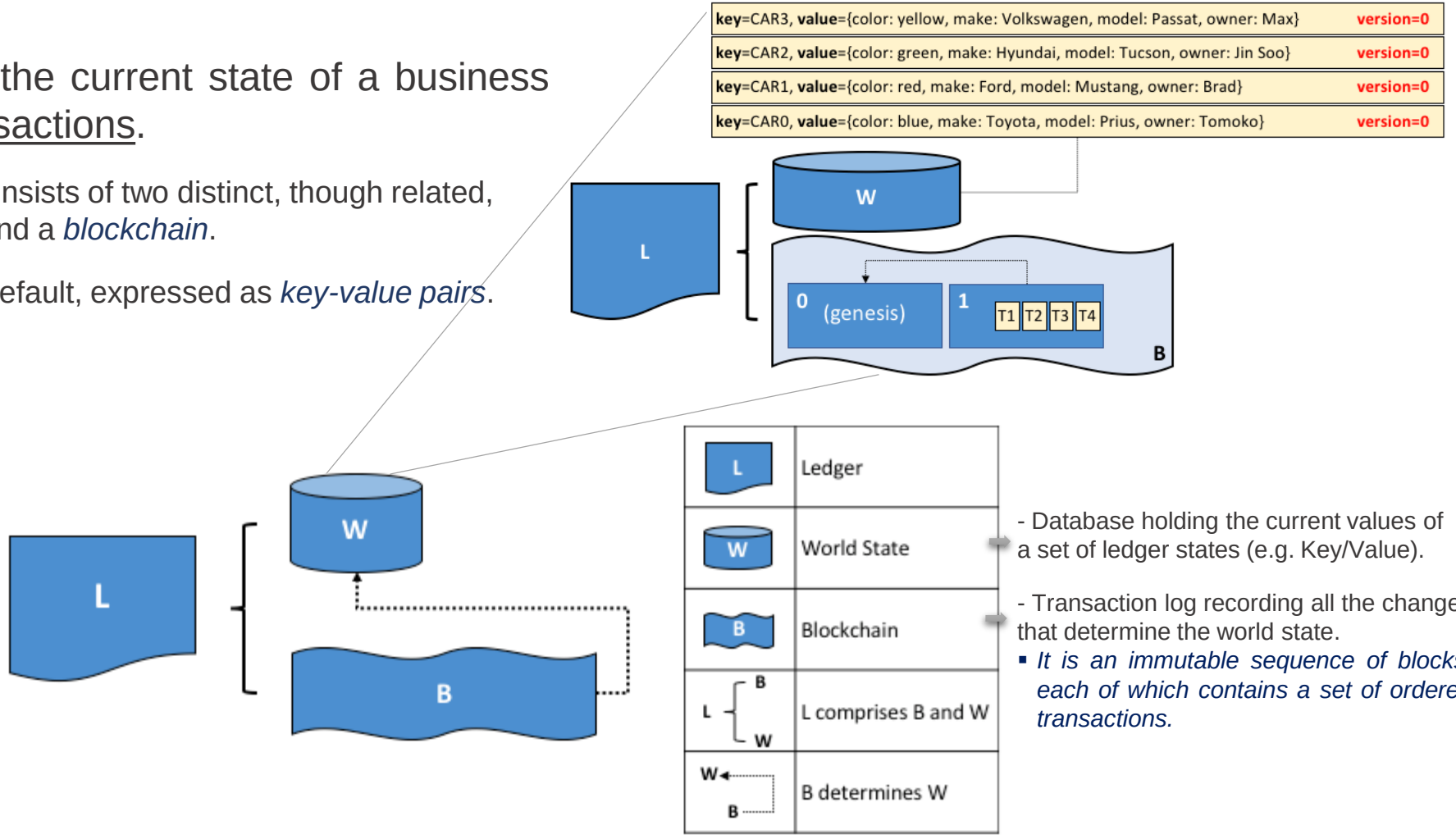
Blockchain Network Integration with Existing Systems — Source: IBM Code Tech Talk

Hyperledger Blockchain Overview

Blockchain Ledger Structure

A ***ledger*** contains the current state of a business as a journal of transactions.

- A blockchain ledger consists of two distinct, though related, parts – *a world state* and a *blockchain*.
- Ledger states are by default, expressed as *key-value pairs*.

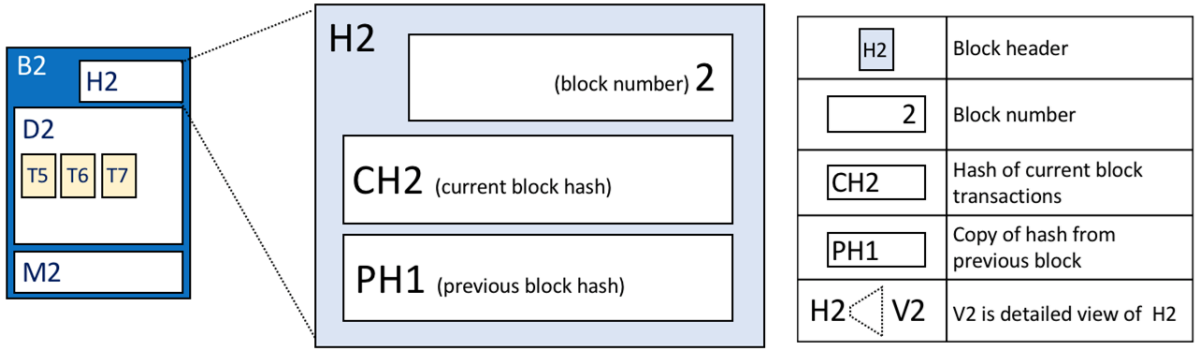


- Database holding the current values of a set of ledger states (e.g. Key/Value).

- Transaction log recording all the changes that determine the world state.
  - *It is an immutable sequence of blocks, each of which contains a set of ordered transactions.*

Source : http://hyperledger-fabric.readthedocs.io/en/release-1.2/ledger/ledger.html

Block Structure



Block Header

- ***Block number***: An integer starting at 0 (the genesis block)
  - Increased by 1 for every new block appended to the BC
- ***Current Block Hash***: contains the hash of all the transactions.
- ***Previous Block Hash***: a copy of the hash of previous block in the chain.

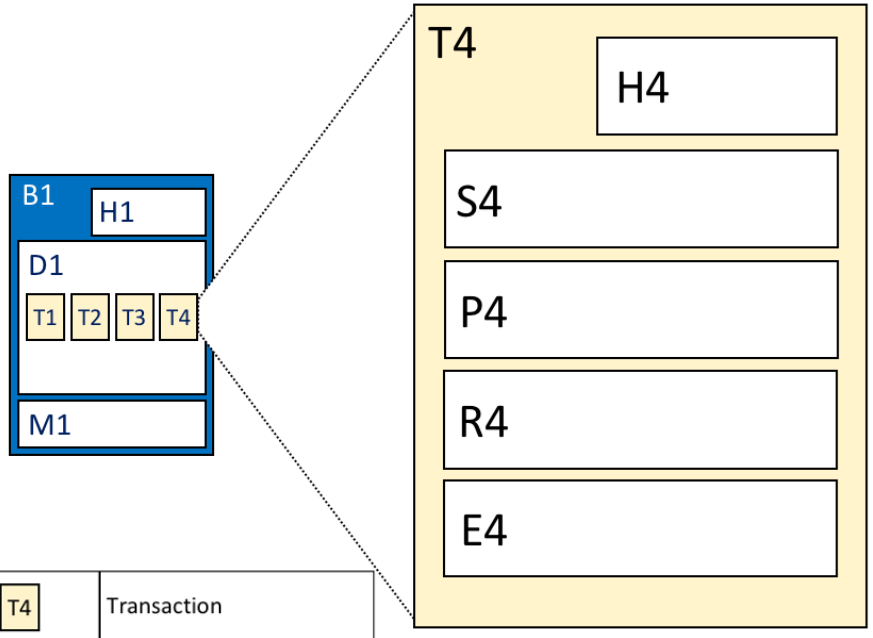
Block Data : contains a list of transactions arranged in order.

- It is written when the block is created

Block Metadata: contains the time when the block was written

- the certificate, public key and signature of the block writer

Transaction Structure



T4	Transaction
H4	Header
S4	Signature
P4	Proposal
R4	Response
E4	Endorsements
T4 ◁ V4	V4 is detailed view of T4

A transaction captures changes to the world state. It contains following fields :

Header : captures some essential metadata about the transaction.

- e.g. the name of the relevant chaincode and its version.

Signature: contains a cryptographic signature, created by the client app.

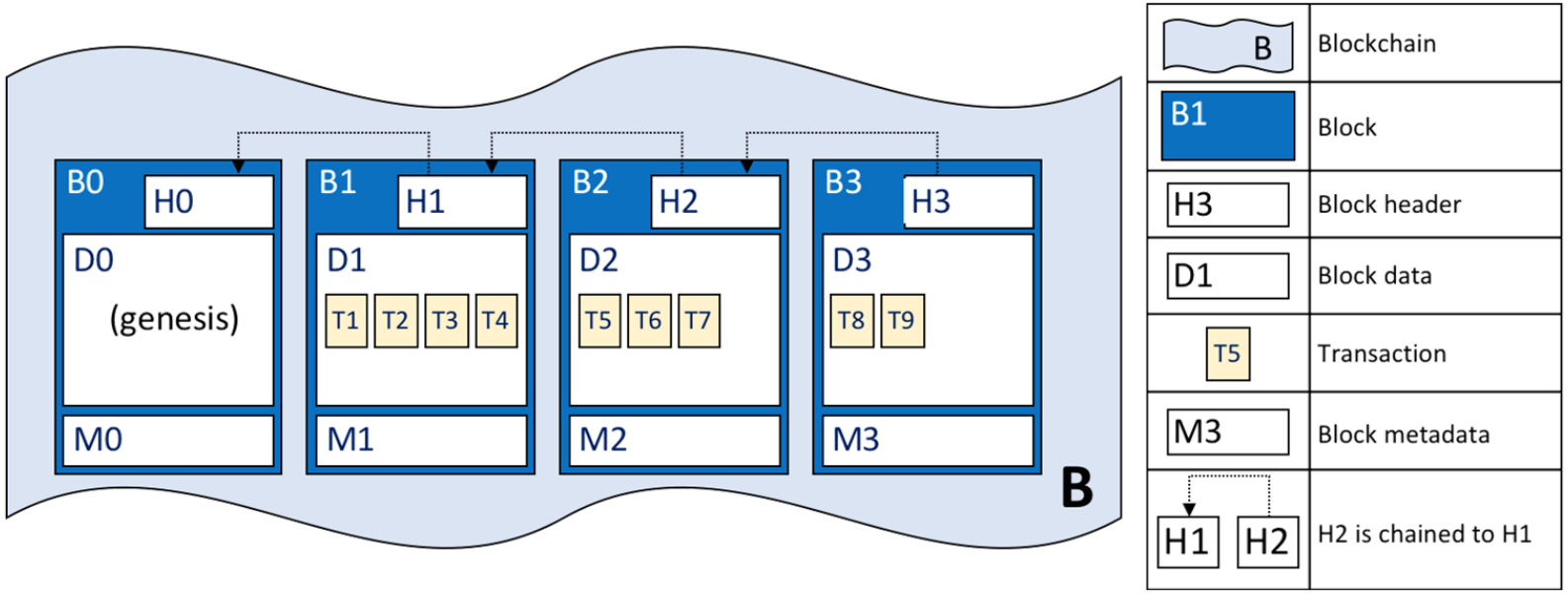
- It is used to check that the transaction details have not been tampered with, as it requires the application's private key to generate it.

Proposal: encodes input parameters supplied by an app. to the chaincode which creates the proposed ledger update.

Response: captures before and after values of the world state, as a *Read Write set (RW-set)*.

Endorsements: is a list of signed transaction responses from each required organization sufficient to satisfy the endorsement policy.

Blockchain Structure



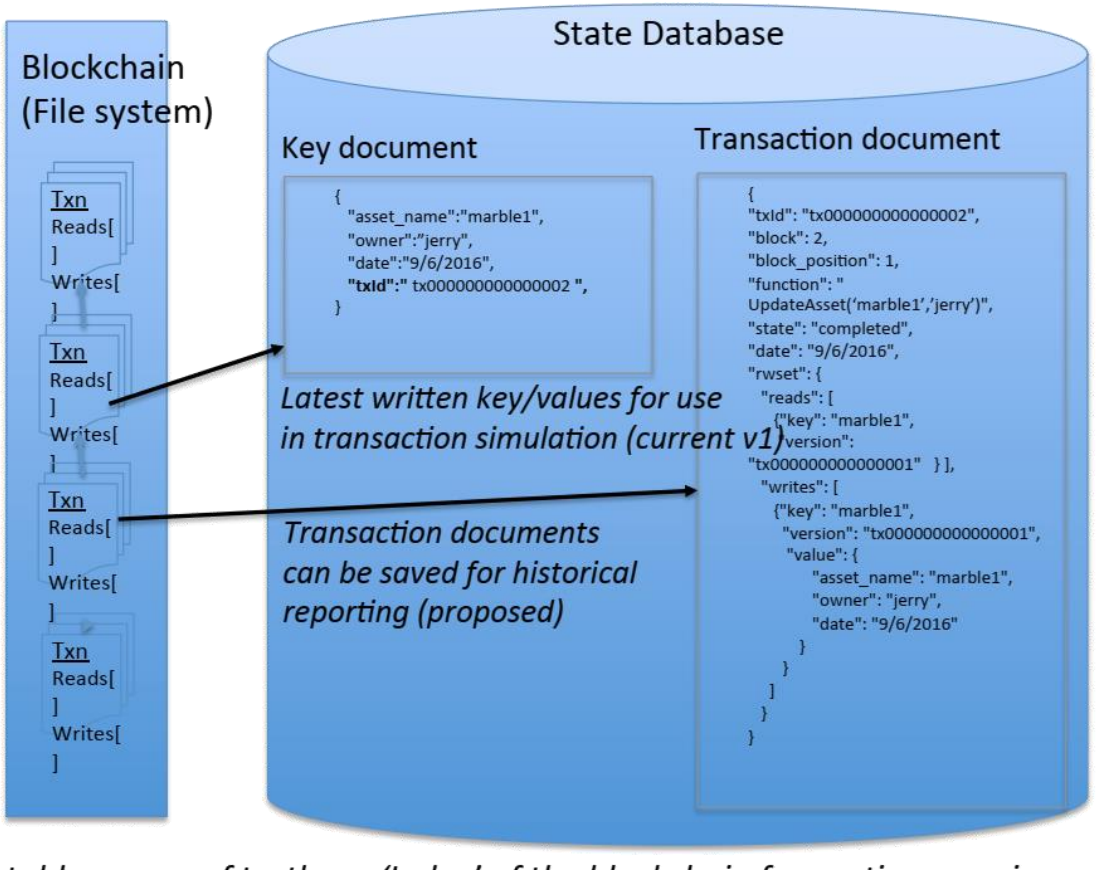
Blockchain is a transaction log, structured as interlinked blocks,

- where each block contains a sequence of transactions,
- each of which represents a query or update to the world state.

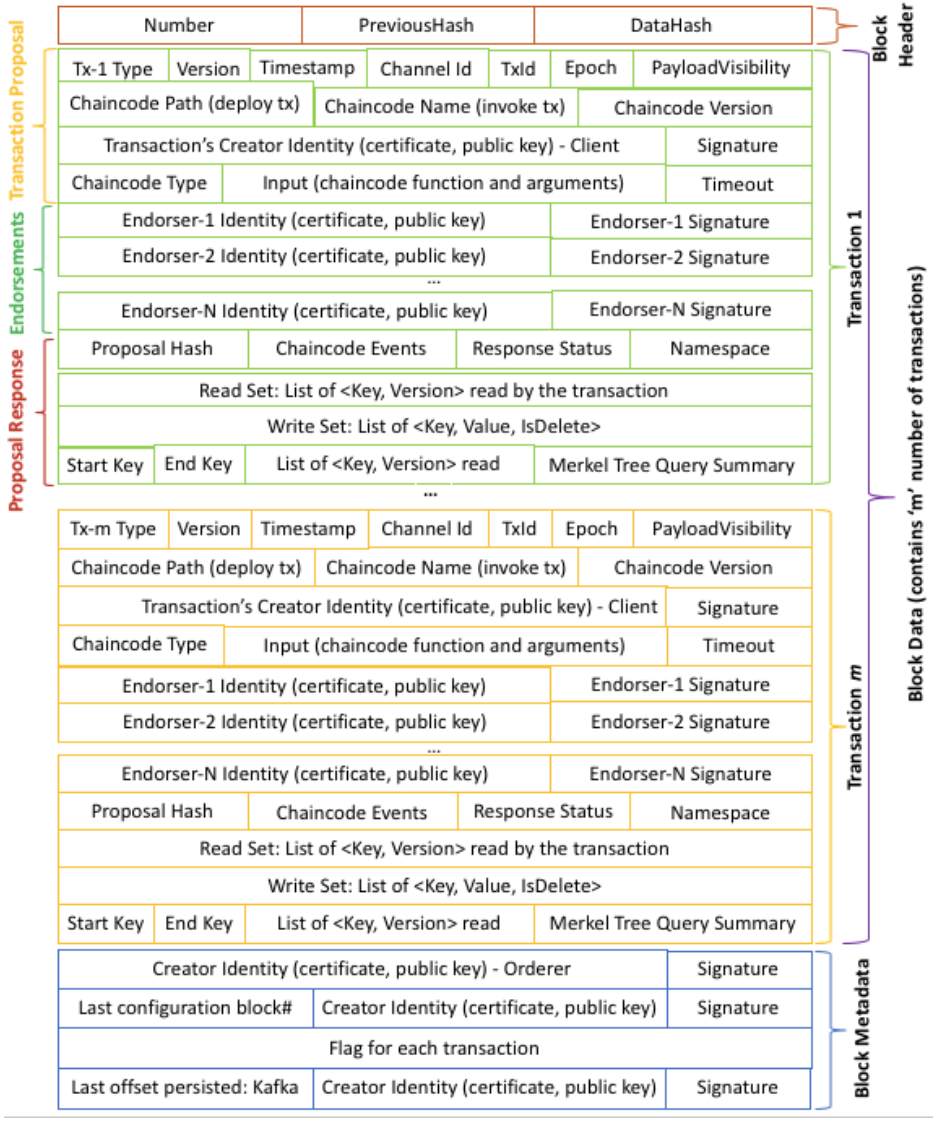
Blockchain Ledger Structure

The ledger is maintained by each peer node in a file system.

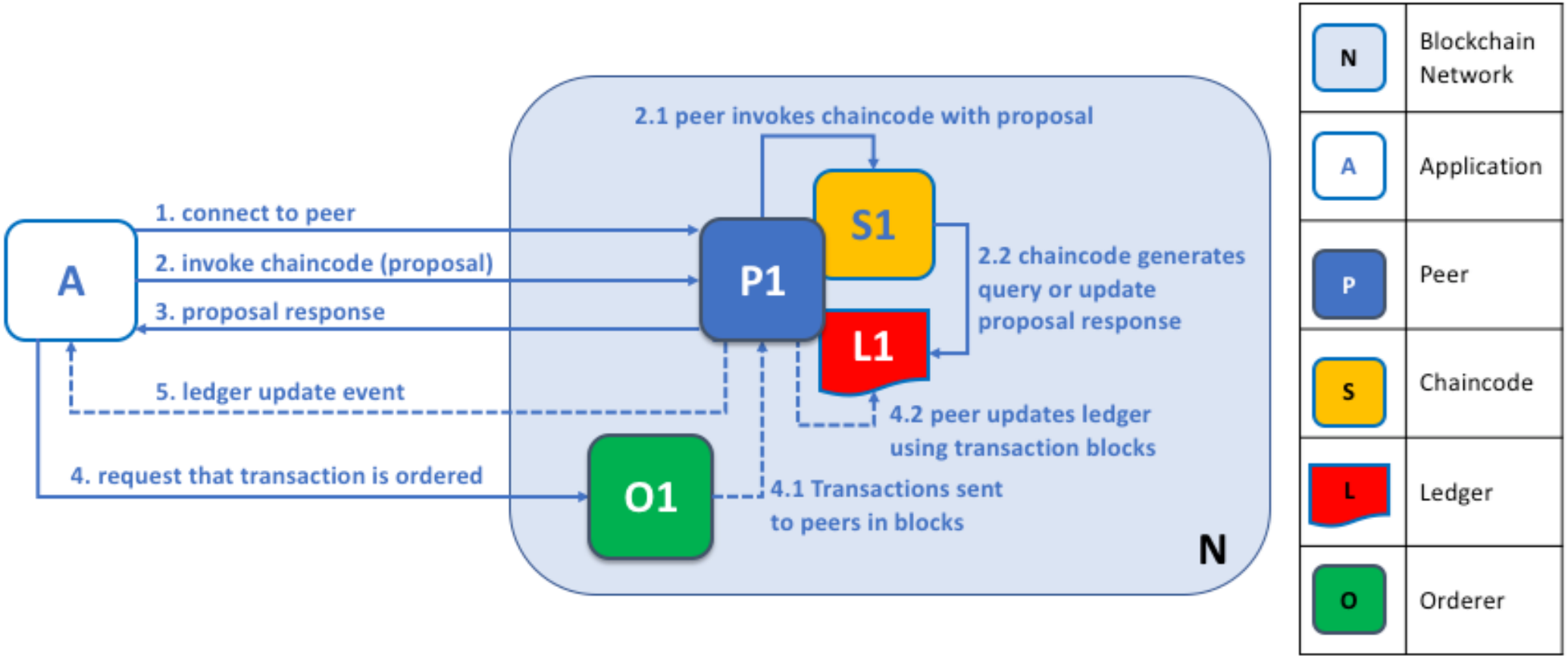
- **The chaincode is allowed access to the shared state by well-defined ledger APIs.**



Immutable source of truth *'Index' of the blockchain for runtime queries*



Interactions between Applications and Peers



Applications and Peers

Source: https://hyperledger-fabric.readthedocs.io

Fabric Development Roadmap

Filter Results: ALL Fabric 2.0 Epics Development

Key	Summary	P	Current Status
FAB-11237	Chaincode lifecycle - 2.0 improvements	↑	Main scenarios working e2e, continue dev and test on other scenarios.
FAB-10889	Chaincode lifecycle - implicit org-specific collections (enables lifecycle 'voting')	↑	Main scenarios working e2e, continue dev and test on other scenarios.
FAB-6135	Raft Consensus (etcd/raft)	↑	Main scenarios in system test, also backport to release-1.4. Development completing for migration.
FAB-10851	Serviceability - Monitor health for Fabric runtime components	↑	Liveness and docker check in v1.4. gRPC, CouchDB, Kafka, CA health checks in v2.0.
FAB-3388	Serviceability - Operational Metrics for Fabric runtime components	↑	Metrics added in v1.4. Gossip, endorser, CA metrics added in v2.0.

1–5 of 20 **1** 2 3 4

Filter Results: ALL Fabric 2.1 Epics Development

Key	Summary	P	Current Status
FAB-13584	Chaincode refactoring	↑	
FAB-10711	Enhanced Concurrency Control	↑	Design reviewed.
FAB-7593	Private data - Local collections	↑	
FAB-12430	Programming model - Inversion of control for Node.js smart contracts	↑	
FAB-13754	Programming model - Java chaincode	↑	Defer until broader chaincode refactoring is complete.

1–5 of 15 **1** 2 3

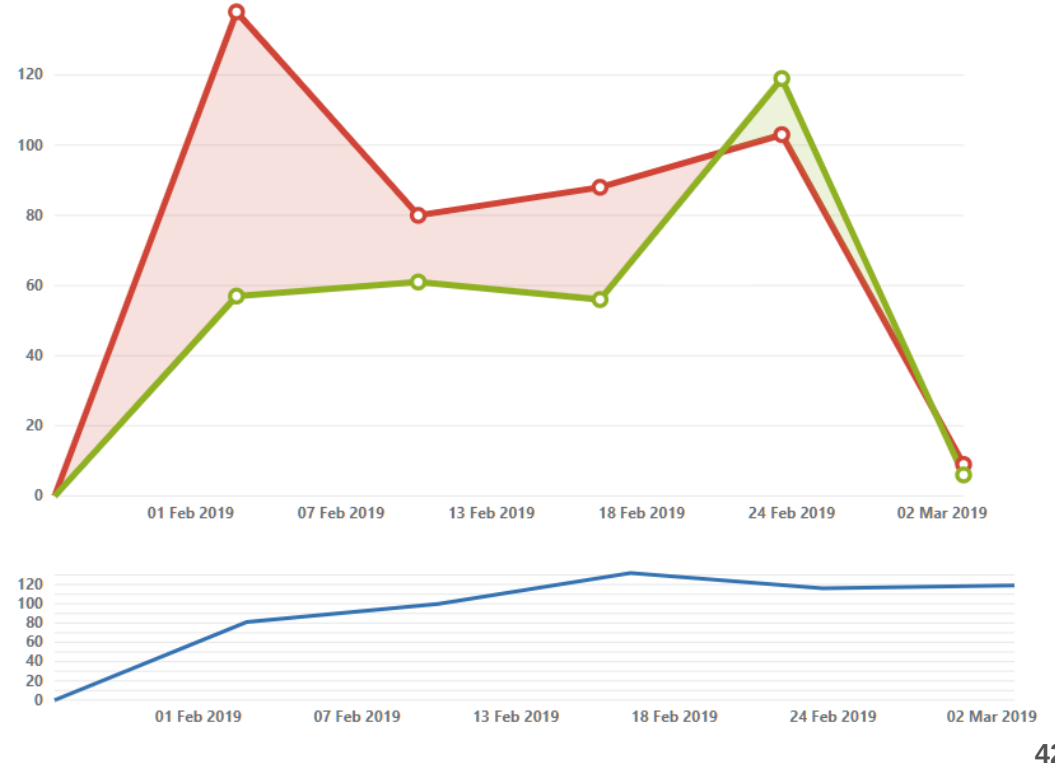
Introduction

If you need an account to JIRA please create a Linux Foundation ID (LFID) then log in to JIRA with that account. This account will be used for all hyperledger.org development resources

For other issues you may email helpdesk@hyperledger.org.

Jira Road Map: Next 30 Days (Until 27/Mar/19)

Created vs. Resolved Chart: Fabric



References

[1] Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018. https://dl.acm.org/citation.cfm?id=3190538

[2] Cachin, Christian. "Architecture of the Hyperledger blockchain fabric." *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. 2016.

[3] Design Philosophy and ConsensusBob Dill, David Smits, Zero to Blockchain, *IBM Redbooks course*, 2017

[4] LinuxFoundationX: LFS171x Blockchain for Business - An Introduction to Hyperledger Technologies, edx MOOC 2017

[5] Hyperledger Architecture, Volume 1: Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

[6] Hyperledger Architecture, Volume 2: Smart Contracts - https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf

[7] https://www.hyperledger.org/resources/publications#presentations

[8] http://hyperledger-fabric.readthedocs.io/en/master/index.html | https://hyperledger-fabric.readthedocs.io/en/release-1.2/membership/membership.html

[9] https://developer.ibm.com/code/wp-content/uploads/sites/118/2017/09/Marbles_BlockChain_Tech_Talk1.pdf

[10] https://www.altoros.com/blog/hyperledger-fabric-v1-0-to-bring-improved-transactions-and-a-pluggable-data-store/

[11] https://github.com/yeasy/hyperledger_code_fabric/blob/master/README.md

[12] https://fabric-sdk-node.github.io/

[13] https://developer.ibm.com/blockchain/ | https://www.ibm.com/developerworks/cloud/library/cl-blockchain-basics-intro-bluemix-trs/

Thank you!

Q & A

mlecjm@korea.ac.kr

mlecjm