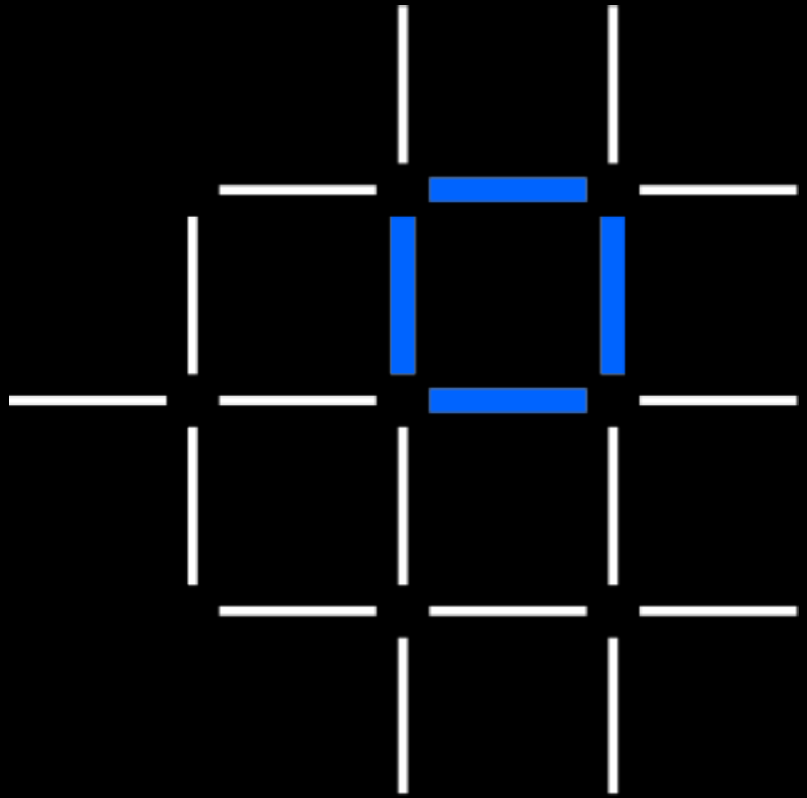
# Blockchain in-depth: Part 2

Unit 05



*IBM Skills Academy*

V1.0, July 2018

IBM **Blockchain**

IBM

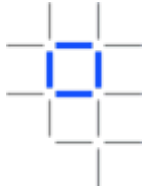
Learning objectives

Consensus overview

Hyperledger Fabric consensus

Summary

# What you should be able to do



Upon completion of this unit, you should be able to:

- Explain the need for consensus.
- Describe the role of the ordering service.
- List the steps of consensus.
- Explain how transaction endorsement and validation works in Hyperledger Fabric.

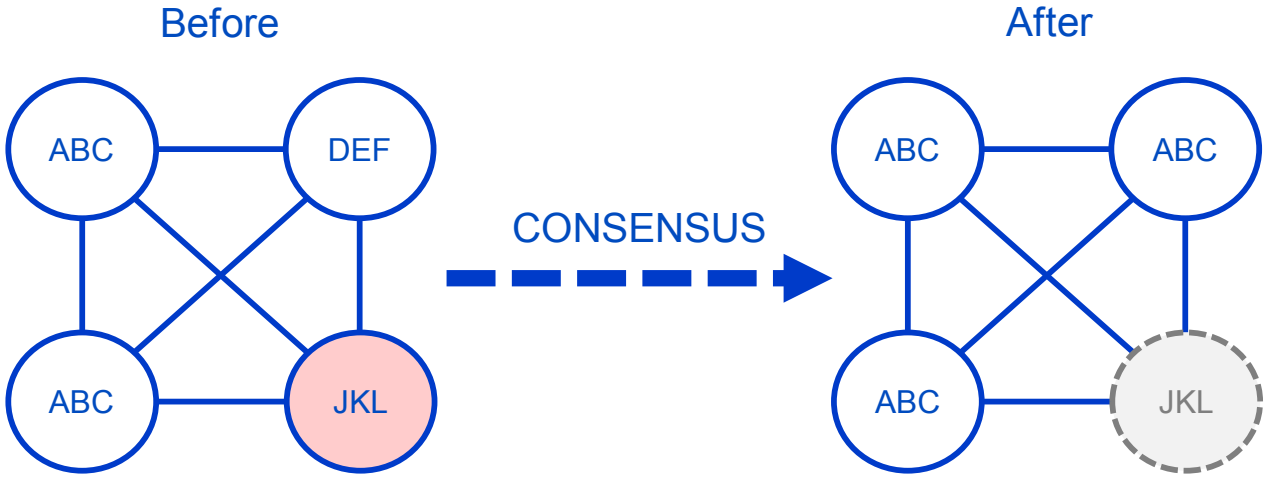
Learning objectives

Consensus overview

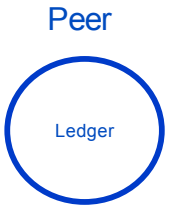
Hyperledger Fabric consensus

Summary

# Consensus: The process of maintaining a consistent ledger




- Keep all peers up-to-date.
- Fix any peers in error.
- Ignore all malicious nodes.



# Some examples of consensus algorithms


Proof of work


Proof of stake

Solo /
No-ops

Kafka /
Zookeeper

Proof of
elapsed time

Practical
Byzantine Fault
Tolerance-based

# Consensus algorithms have different strengths and weaknesses


Proof of work

Requires validators to solve difficult cryptographic puzzles.
Pros: Works in untrusted networks.
Cons: Uses much energy, and slow to confirm transactions.
Example usage: Bitcoin and Ethereum


Proof of stake

Requires validators to hold currency in escrow.
Pros: Works in untrusted networks.
Cons: Requires intrinsic (crypto)currency, and the ”Nothing at stake” problem.
Example usage: Nxt

Proof of Elapsed Time

Wait time in a trusted execution environment randomizes block generation.
Pros: Efficient.
Cons: Requires processor extensions.
Example usage: Hyperledger Sawtooth

# Consensus algorithms have different strengths and weaknesses

Solo / No-ops

Validators apply received transactions without consensus.
Pros: Quick, and suited to development.
Cons: No consensus, which can lead to divergent chains.

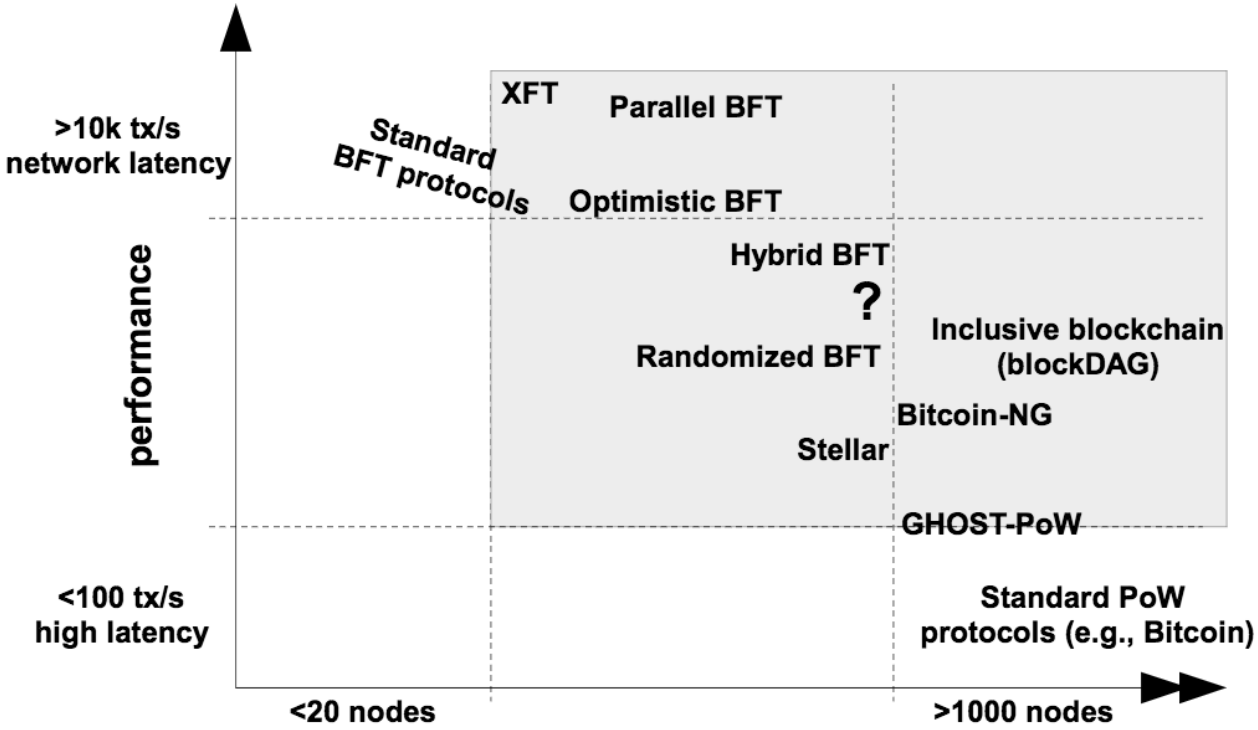
PBFT-based

Practical Byzantine Fault Tolerance (PBFT) implementations
Pros: Reasonably efficient and tolerant against malicious peers.
Cons: Validators are known and connected.

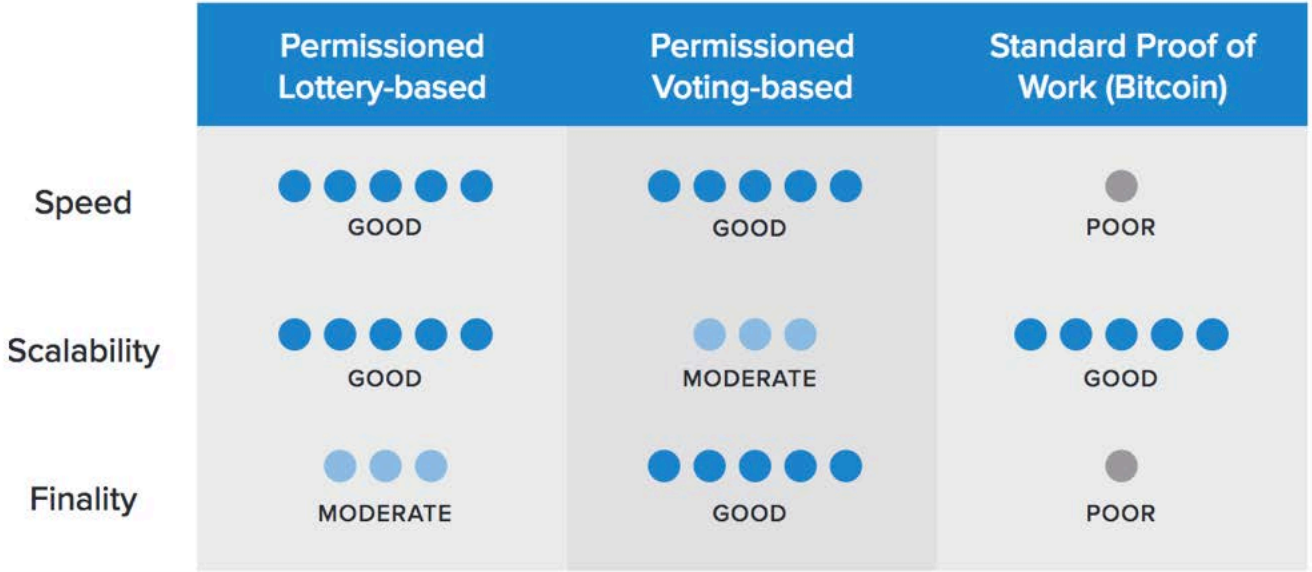
Kafka / Zookeeper

Ordering service distributes blocks to peers.
Pros: Efficient and fault-tolerant.
Cons: Does not guard against malicious activity.

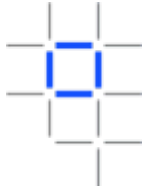
# Consensus tradeoffs: Latency versus scalability



# Consensus tradeoffs: Permissioned versus permissionless



# Consensus tradeoffs: Support for smart contracts



Smart contracts and consensus algorithms co-operate to build trust among the blockchain network. Supporting smart contracts has important implications on consensus.

Trust model:

- May be determined by the consensus protocol.
- May be driven by the requirements of the smart contracts.
- Trust in development may be required.

Confidentiality: Consensus should allow the dissemination of contract code and the state to a *subset* of peers in the network.

Performance:

- Limit execution to sequential paradigms.
- Alternatively, support parallelism.

Determinism:

- Limit the programming model to ensure that peers do not diverge.
- Alternatively, handle determinism at the consensus level.

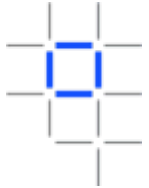
Learning objectives

Consensus overview

Hyperledger Fabric consensus

Summary

# Hyperledger consensus design principles



All Hyperledger projects follow a modular, token-neutral approach, with a focus on interoperability and security:

- Consensus may be implemented in different ways to target different network requirements.
- A generalized reference architecture for consensus is used. It can be used by any Hyperledger project.

The assumption is that business blockchain networks operate in an environment of partial trust, that is, there are no anonymous miners.

Consensus must depend on smart contracts for transaction validation.

Hyperledger frameworks reach consensus by logically separating two activities:

- Ordering of transactions
- Validating transactions

By logically separating these activities:

- Any Hyperledger framework can work with any Hyperledger consensus module.
- Separation of concerns enables better performance and scalability of the network.

# Hyperledger Fabric consensus implementation

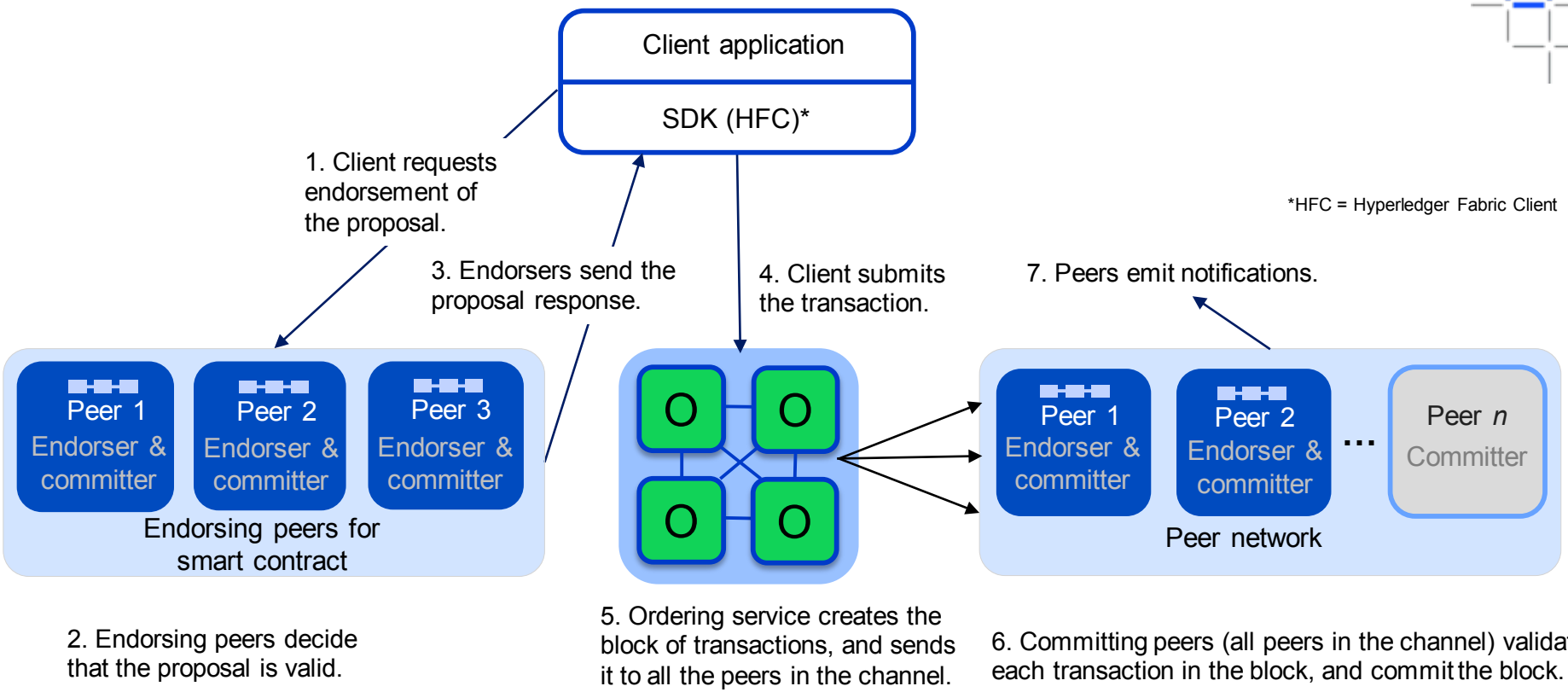
Consensus is achieved by using the following transaction flow:


- Simulate transactions.
- Collect results.
- Collect endorsements.

- Order transactions.
- Create blocks.
- Broadcast blocks.

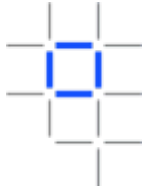
- Validate endorsements.
- Eliminate invalid transactions.
- Update the ledger.

# Transaction lifecycle

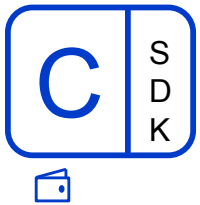


# Endorse

## Who is involved

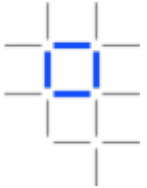


Before submitting a transaction…

| | |
|---|---|
|  | **Clients** propose the transaction to the peers and collect their endorsement signatures on a specific *policy*. |
|  | **Endorsing peers** receive a transaction proposal for endorsement, execute the proposal, and respond granting or denying endorsement. Endorsing peers must hold smart contracts. |

# Endorse

# Endorsement policies



An **endorsement policy** describes the conditions by which a transaction can be trusted. A transaction can be considered valid only if it is endorsed according to its policy. Endorsement policies are defined at chaincode instantiation on a specific channel.

Examples of policies:

- Request one signature from all three principals:

  AND(‘Manufacturer.member', 'Regulator.member’, ‘Leasing.member')

- Request one signature from either one of the two principals:

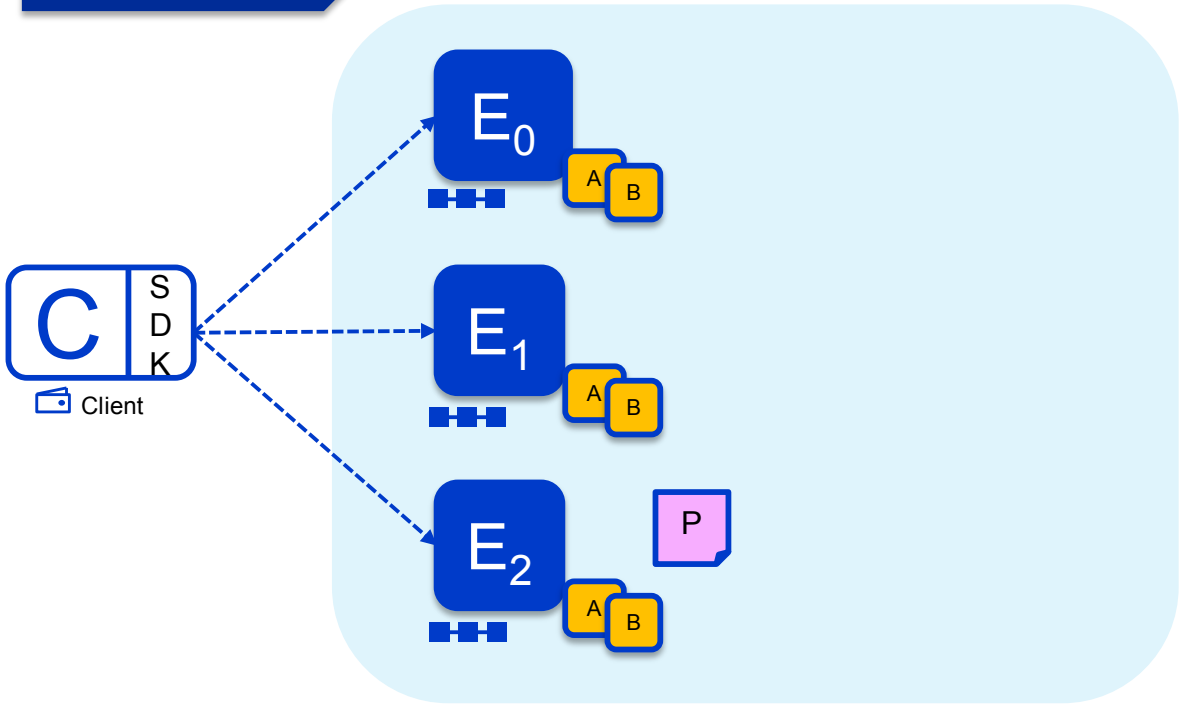
  OR(‘Manufacturer.member', ‘Leasing.member')

- Request either one signature from a member of the Manufacturer MSP, or one signature from a member of the Regulator MSP and one signature from a member of the Authority MSP:

  OR(‘Manufacturer.member', AND('Regulator.member', ‘Leasing.member'))

# Endorse

## Step 1/7: Propose a transaction

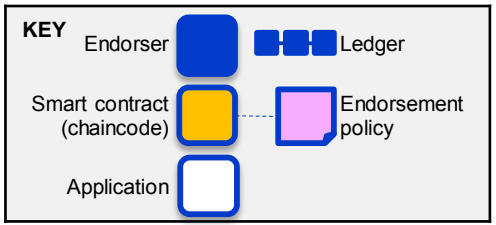



Hyperledger Fabric Network

The application proposes a transaction.

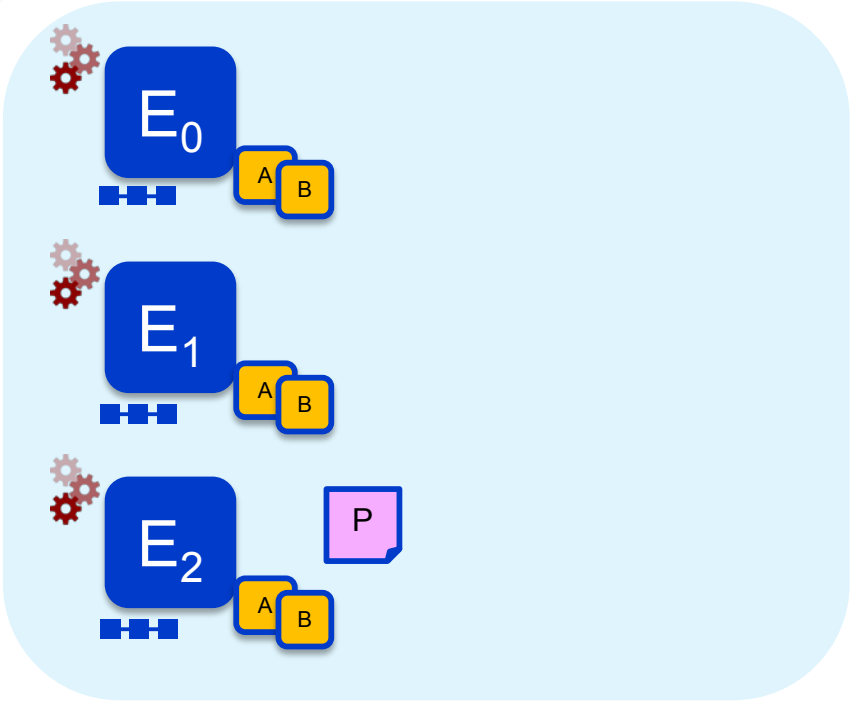
Endorsement policy:
- "$E_0$, E1, and $E_2$ must sign"

The client application submits a transaction proposal for smart contract A. It must target the required peers {$E_0$, $E_1$, and $E_2$}.



# Endorse

## Step 2/7: Execute the proposal



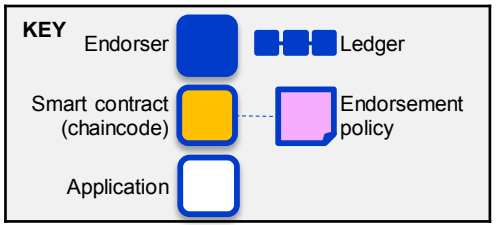

Hyperledger Fabric Network

C SDK Client

The endorsers execute the proposals.

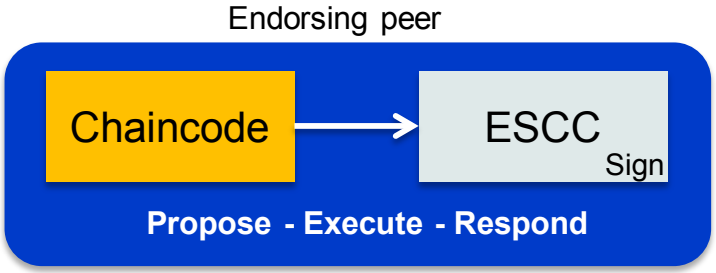
$E_0$, $E_1$, and $E_2$ each execute the proposed transaction. None of these executions update the ledger.

Transactions can be signed and encrypted.



# Endorse

## Inside the endorsing peer



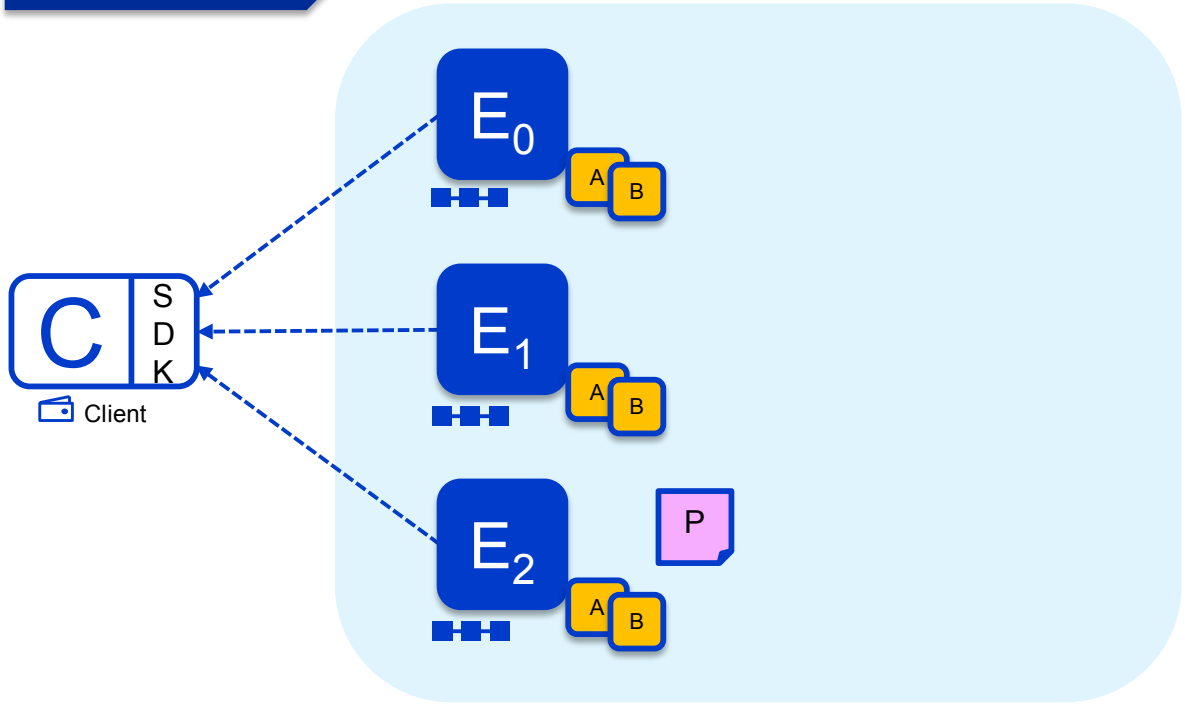

Each execution captures the set of read and written data, which is called the RW set, which now flows in Hyperledger Fabric.

Endorsement System Chaincode (**ESCC**) signs the proposal response on the endorsing peer.

The RW sets are signed by each endorser, and also include each record's version number.

# Endorse

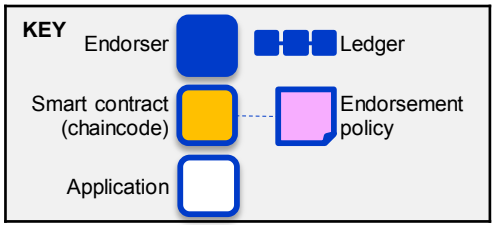
## Step 3/7: Proposal response



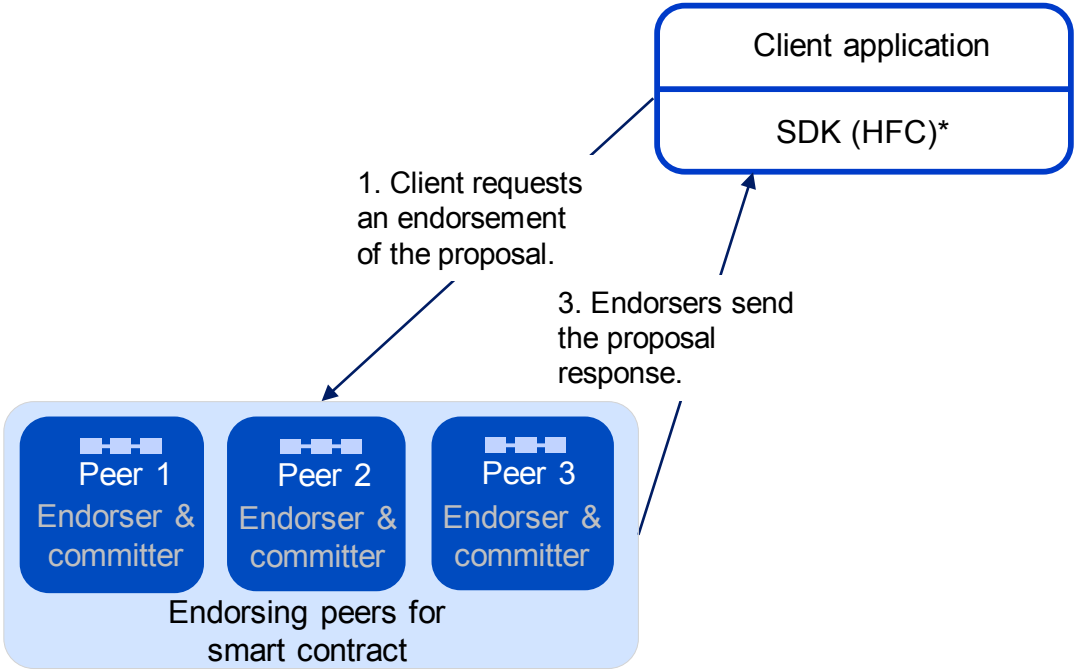

Hyperledger Fabric Network

The application receives responses.

RW sets are asynchronously returned to the application.



# Transaction lifecycle: Endorse

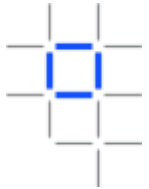



*HFC = Hyperledger Fabric Client

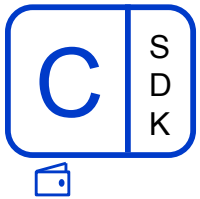
Endorse

# Order

## Who is involved

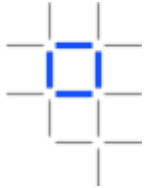


When submitting a transaction:

| | |
|---|---|
|  | **Clients** submit the transactions to orderers to broadcast them to the peers. |
|  | **Orderers** or **ordering-service nodes** receive transactions from clients, form blocks, and deliver them to the peers. |

# Order

## The ordering service



The ordering service packages transactions into blocks to be delivered to peers while ensuring the following:

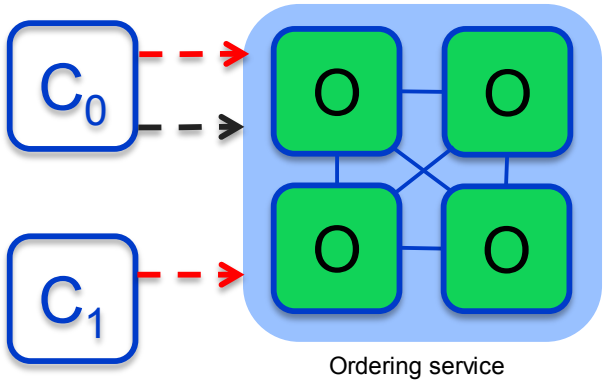
- Agreement: A block is delivered with the same sequence number to all peers.
- Hash chain integrity: For all peers, the current block contains the hash of the previous one.
- Sequential Delivery: Each block is delivered sequentially to every peer. No block is skipped or missed.
- No transaction creation: Blocks are composed only of transactions that are broadcast by clients.
- Eventual validity: If a client broadcasts a transaction, it eventually is delivered in a block.

The ordering service is run by specialized nodes that:

- Do not hold smart contracts.
- Do not hold endorsement policies.
- Do not need to store the world state to work.
- Do not update the ledger by themselves.
- Can see all transactions by default, but do not need to inspect the details of a transaction.

The ordering service may perform access control to check whether clients are allowed to broadcast a transaction on the channel.

# The ordering service: Consensus protocols

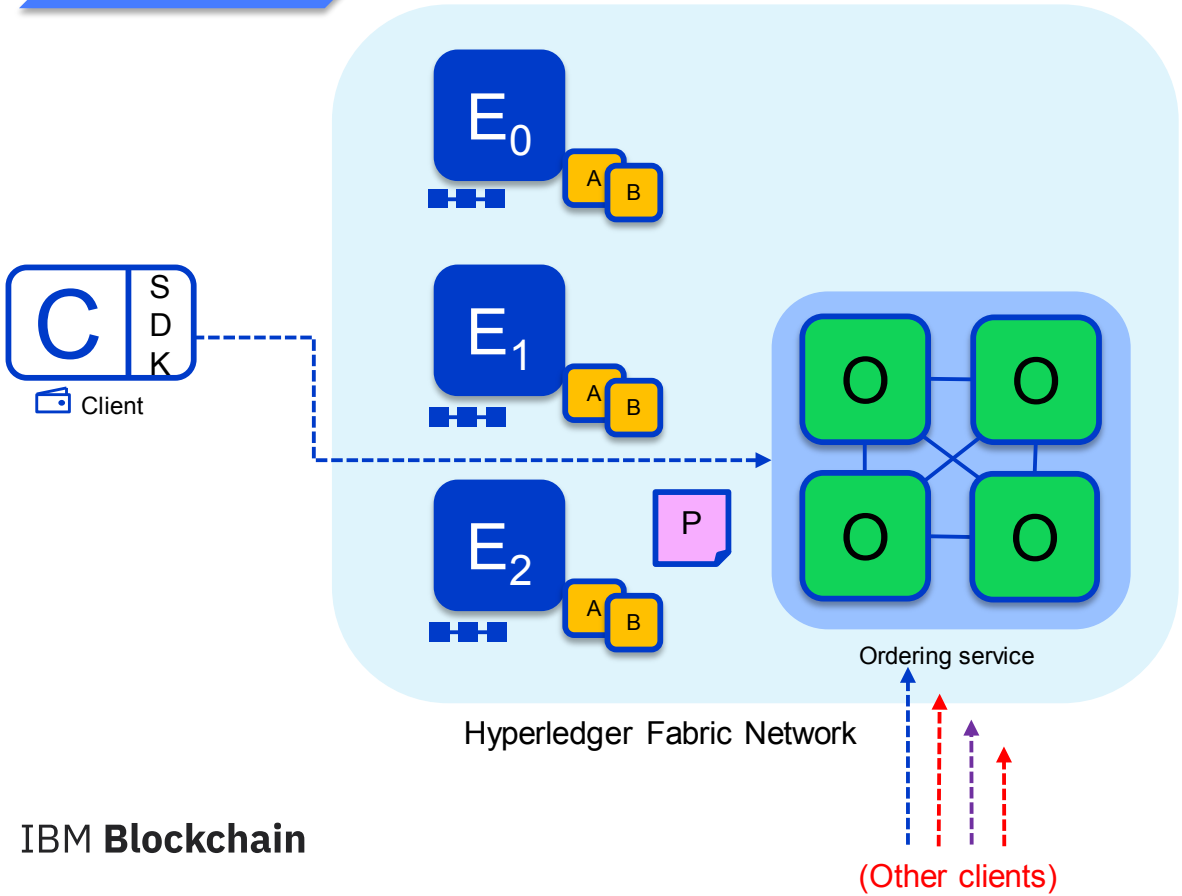



Different configuration options for the ordering service include:

- SOLO: Single node for development
- Kafka: Crash fault tolerant consensus:
  - Three nodes minimum.
  - Odd number of nodes are preferred.

# Order

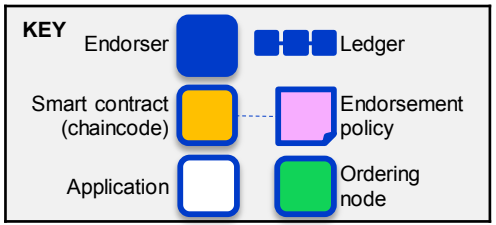
## Step 4/7: Order transaction




Responses are submitted for ordering.

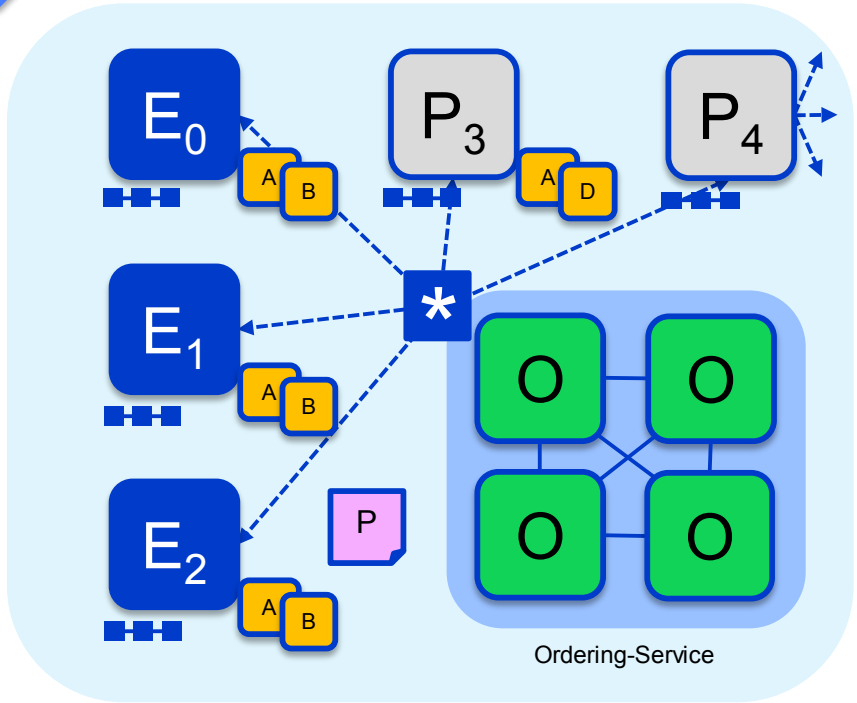
The client submits endorsed responses as a transaction to be ordered.

Ordering happens across the Hyperledger Fabric in parallel with transactions that are submitted by other applications.



# Order

## Step 5/7: Deliver the transaction



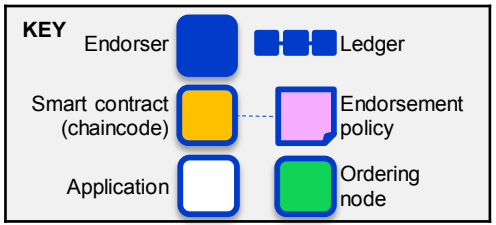

Hyperledger Fabric Network

C SDK Client

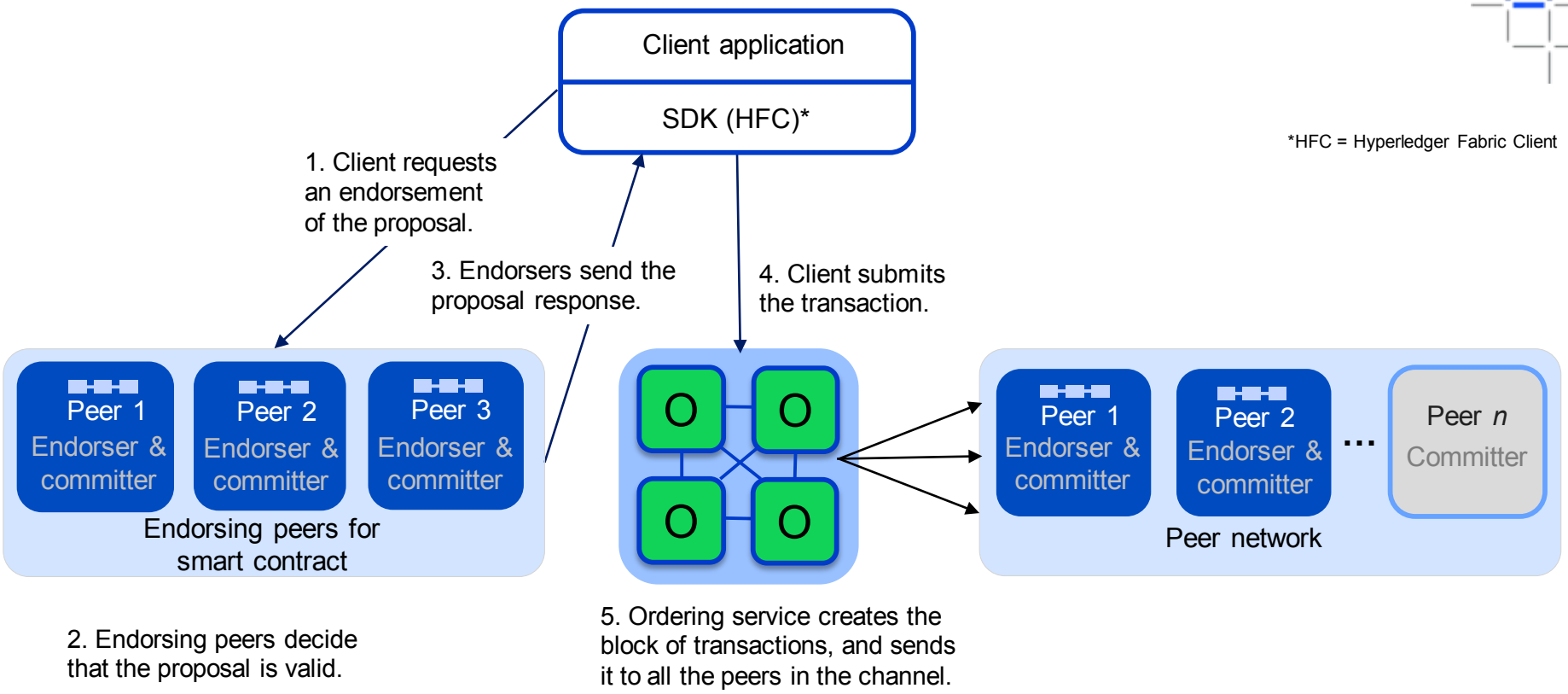
The orderer delivers to the committing peers.

The ordering service collects transactions for a channel into proposed blocks for distribution to peers. Blocks are delivered on a channel basis.

Peers can deliver to other peers in a hierarchy (not shown).



# Transaction lifecycle: Order




*HFC = Hyperledger Fabric Client

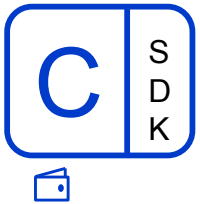
# Validate

## Who is involved

When receiving a new block…

| | |
|---|---|
|  | **Committing peers** (including endorsing peers) run validation and update their copy of the blockchain and world state.<br>Committing peers may hold smart contracts, but they do not execute them at this stage. |

...after transaction validation…

| | |
|---|---|
|  | **Clients** who registered are notified about any new blocks and transactions that are delivered on the channel. |

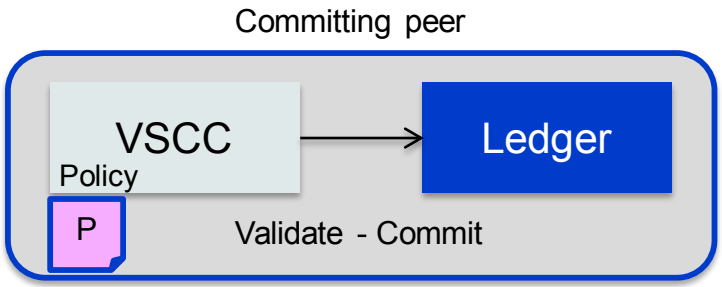
# Validate

## Inside the committing peer

On every committing peer, Validation System Chaincode (**VSCC**):

- Validates the endorsements against the endorsement policy.
- Checks that the RW sets are still valid for current world state.

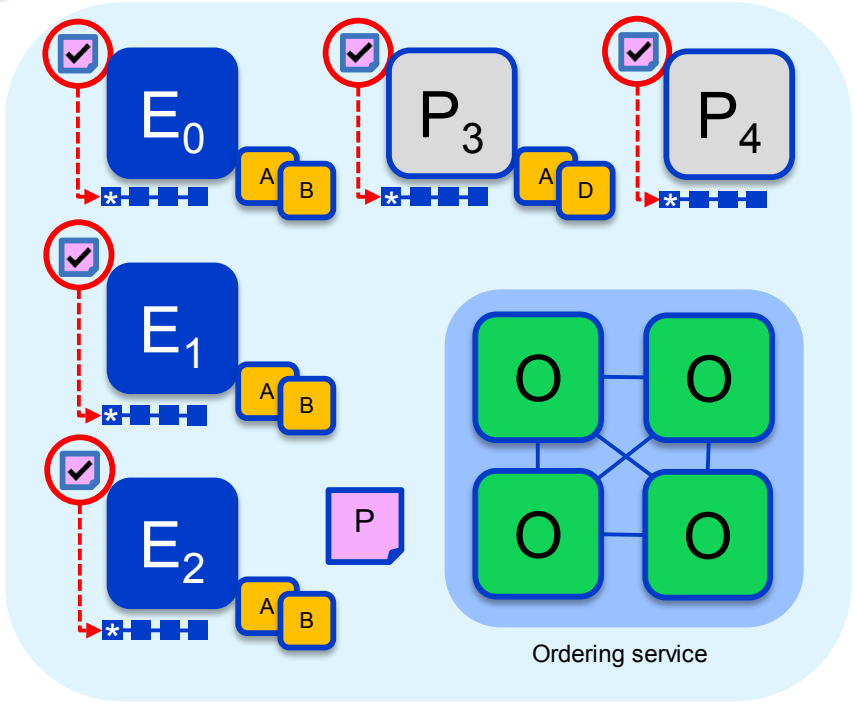
Validated transactions are applied to the world state and retained on the ledger.
Invalid transactions are also retained on the ledger, but do not update world state.



# Validate

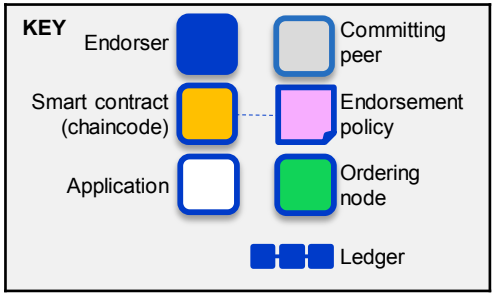
## Step 6/7: Validate the transaction

Committing peers validate transactions.



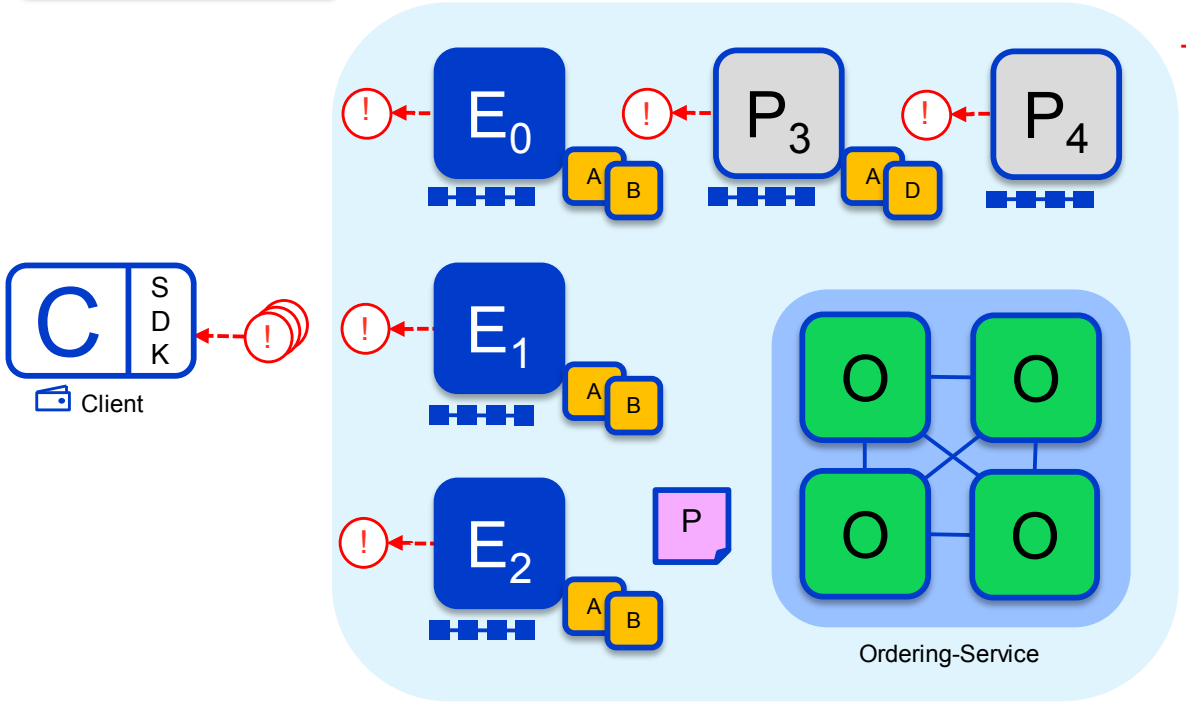

Hyperledger Fabric Network

C SDK Client



# Validate

## Step 7/7: Notify the transaction

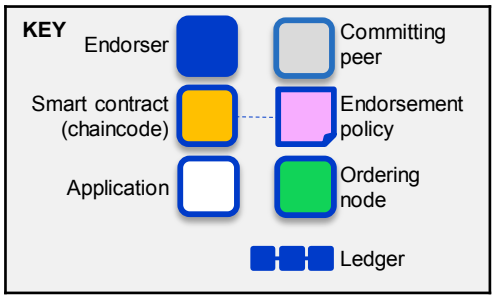



Hyperledger Fabric Network

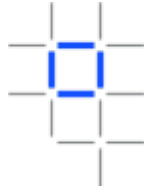
The committing peers notify the applications.

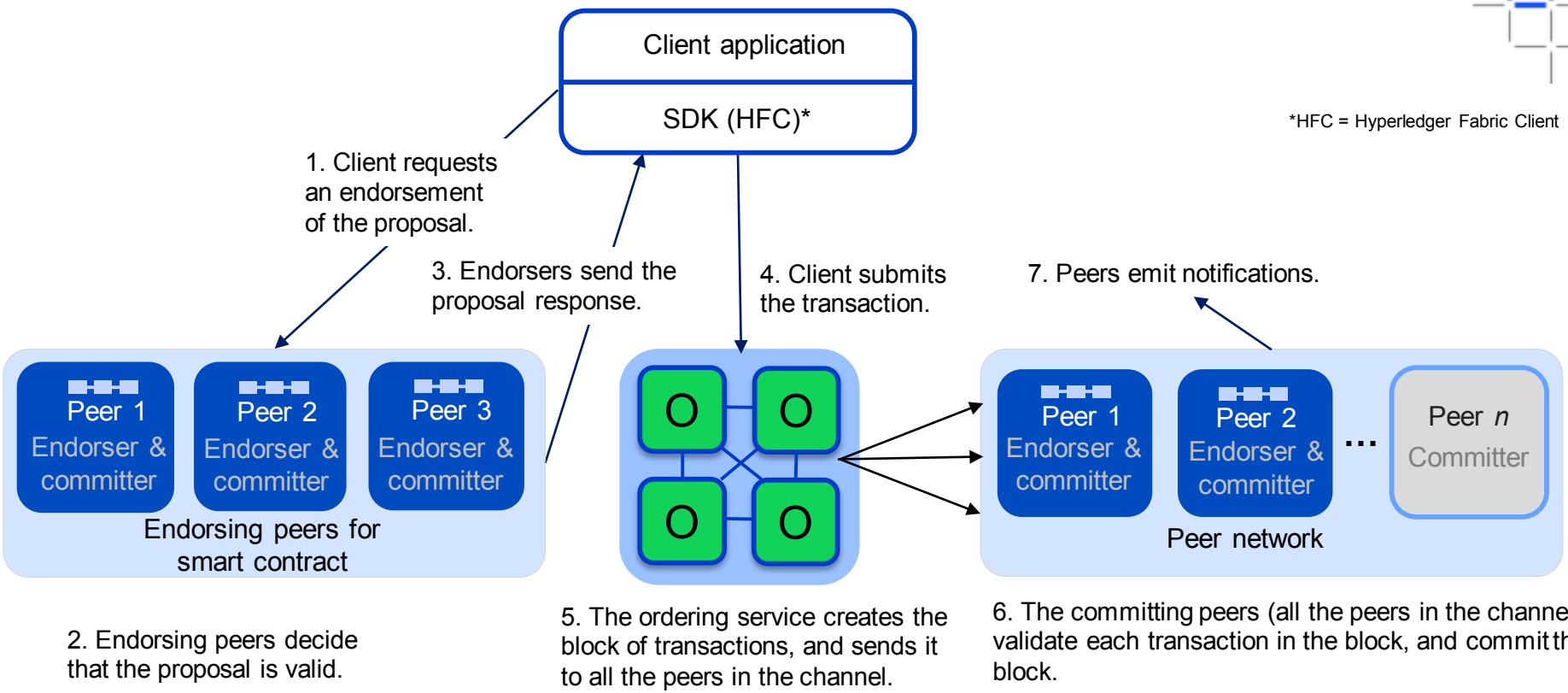
Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger.

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



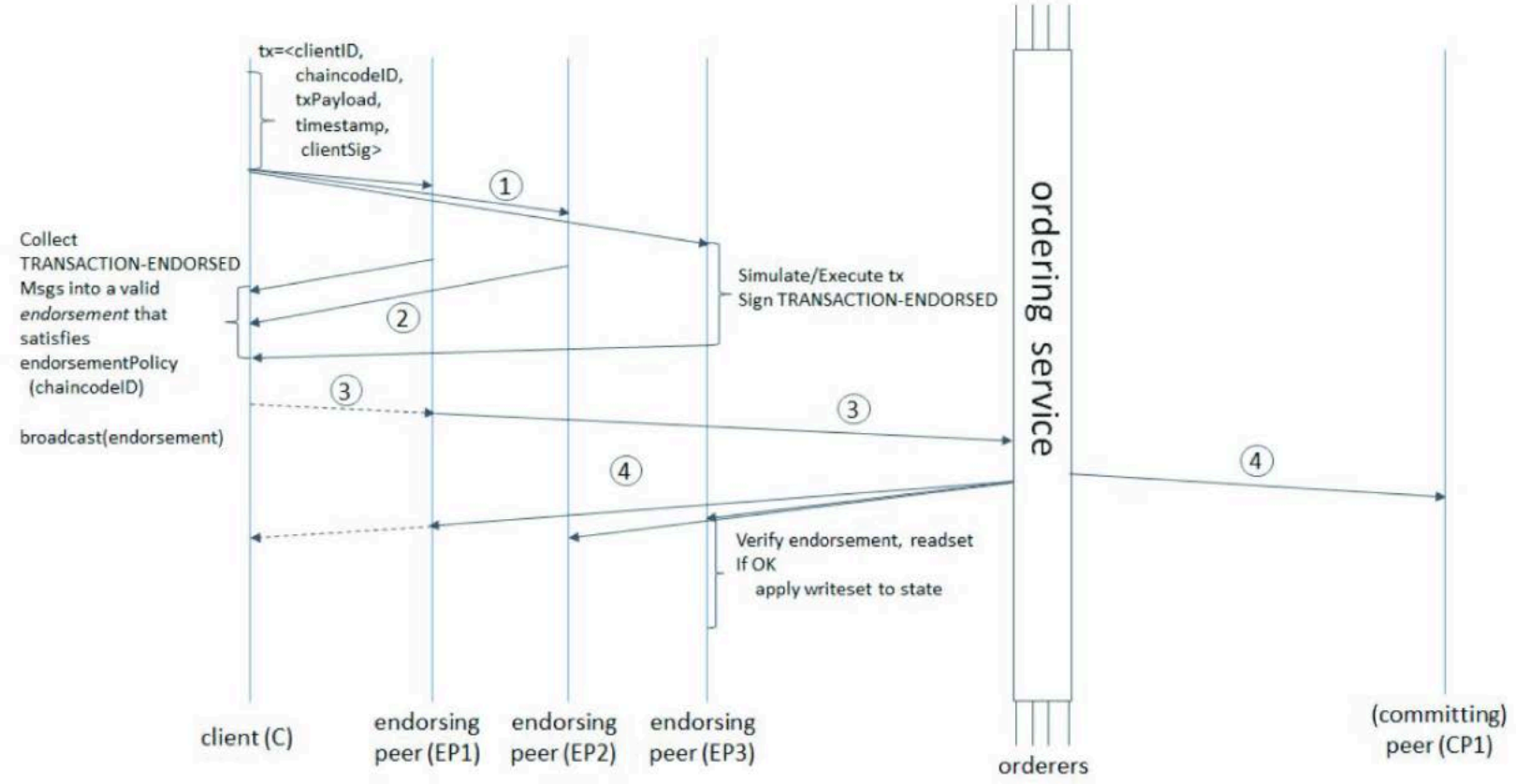
# Transaction lifecycle: Validate






Endorse | Order | Validate

# Hyperledger Fabric consensus: Putting it all together






https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

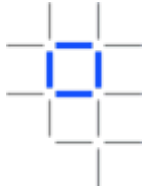
Learning objectives

Consensus Overview

Hyperledger Fabric consensus

Summary

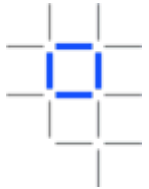
# Unit summary



This unit described consensus algorithms and the Hyperledger Fabric approach to consensus:

- The objective of consensus algorithms is to maintain a consistent ledger.
- Different consensus algorithms have pros and cons; there is no “one-size-fits-all”.
- The Hyperledger approach to consensus is modular and focuses on enterprise blockchain.
- The key steps in Hyperledger Fabric Consensus are endorse, order, and validate.
- The key elements that are involved in a transaction are clients, peers, orderers, smart contract, and endorsement policies.
- Transactions are executed by peers only after the transaction proposals are submitted for endorsement.
- Specialized nodes take care of ordering transactions and creating blocks.
- Validation is based on a read/write set and endorsement signatures.

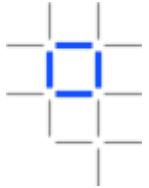
# Exercise objectives



During this exercise, you will apply Hyperledger Fabric network concepts to:

- Start a sample.
- Review the existing organizations and channels.
- Add an organization and peer.
- Add a peer to an existing channel.
- Install and instantiate the Hyperledger Fabric chaincode.
- Review the changes by querying the existing data on the ledger before adding the new peer.
- [Optional] Deploy the whole network on IBM Cloud by using Kubernetes.

# References



For more information about the topics that are covered in this unit, see the following resources:

- Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains
- Hyperledger Architecture Paper, Volume 1
- http://hyperledger-fabric.readthedocs.io/en/release-1.1
- https://developer.ibm.com/courses/all/ibm-blockchain-foundation-developer/
- https://developer.ibm.com/academic/resources/blockchain-educator-guide/
- https://allquantor.at/blockchainbib/pdf/vukolic2015quest.pdf
- https://arxiv.org/abs/1801.10228v1.pdf
- https://www.youtube.com/watch?v=8kRc2895uMY
- https://www.youtube.com/watch?v=DqtzxJP6Y9k
- https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

# Thank you.

*IBM Skills Academy*

IBM **Blockchain**

www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org