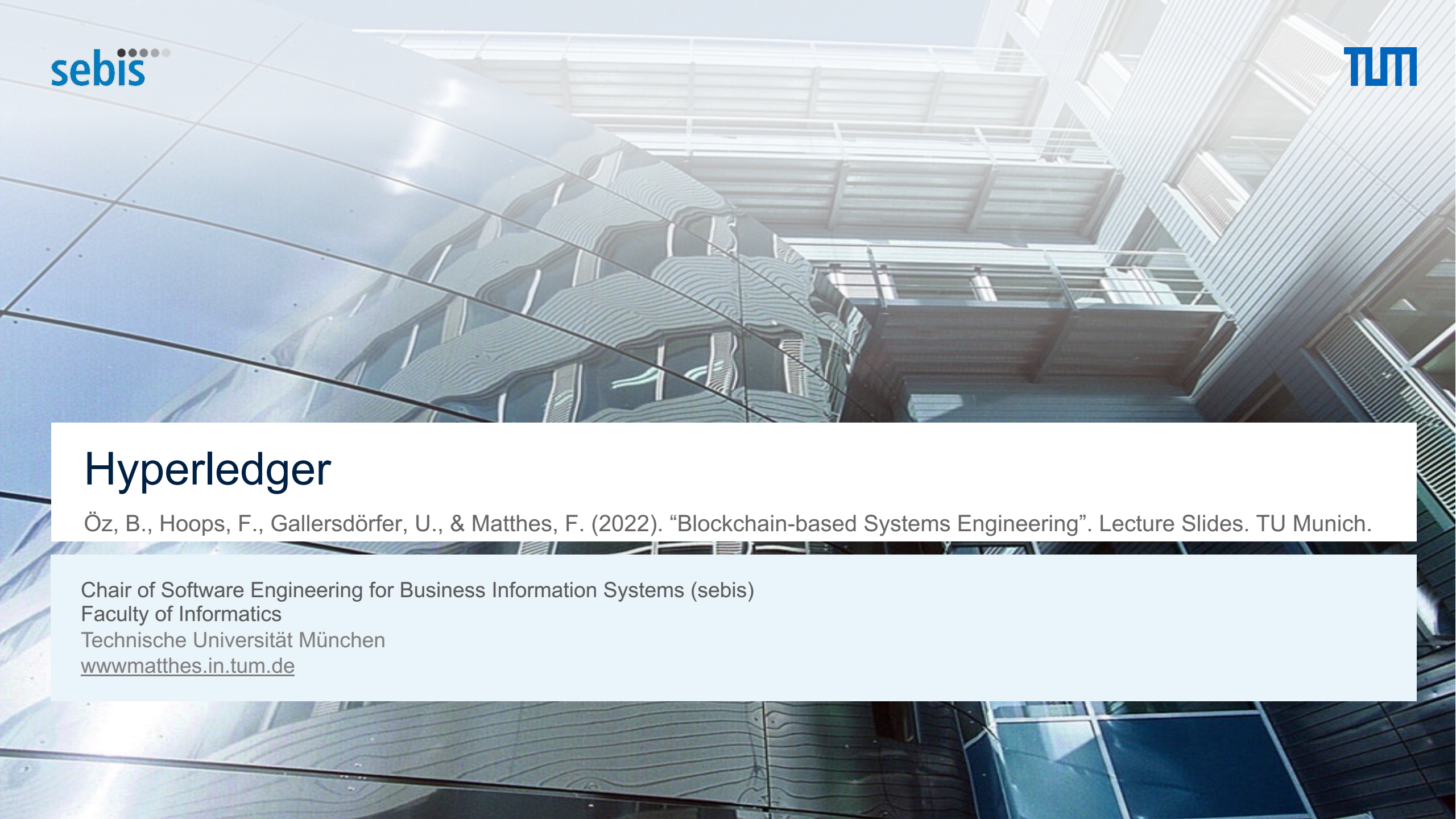


Hyperledger

Öz, B., Hoops, F., Gallersdörfer, U., & Matthes, F. (2022). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

1. Beyond Public Blockchains
- Recap
- Motivation
- Use Cases

2. The Hyperledger Project
- Vision and Mission

3. Hyperledger Fabric
- Architecture
- Membership Service Provider
- Application
- Ordering Service
- Peers
- Chaincode
- Order-Execution vs. Execution-Order
- Limitations
- Nodes and Roles
- Transaction Flow
- Channels
- Ledger
- Recent Developments in Hyperledger Fabric

Reference Architecture for a Smart Contract Platform

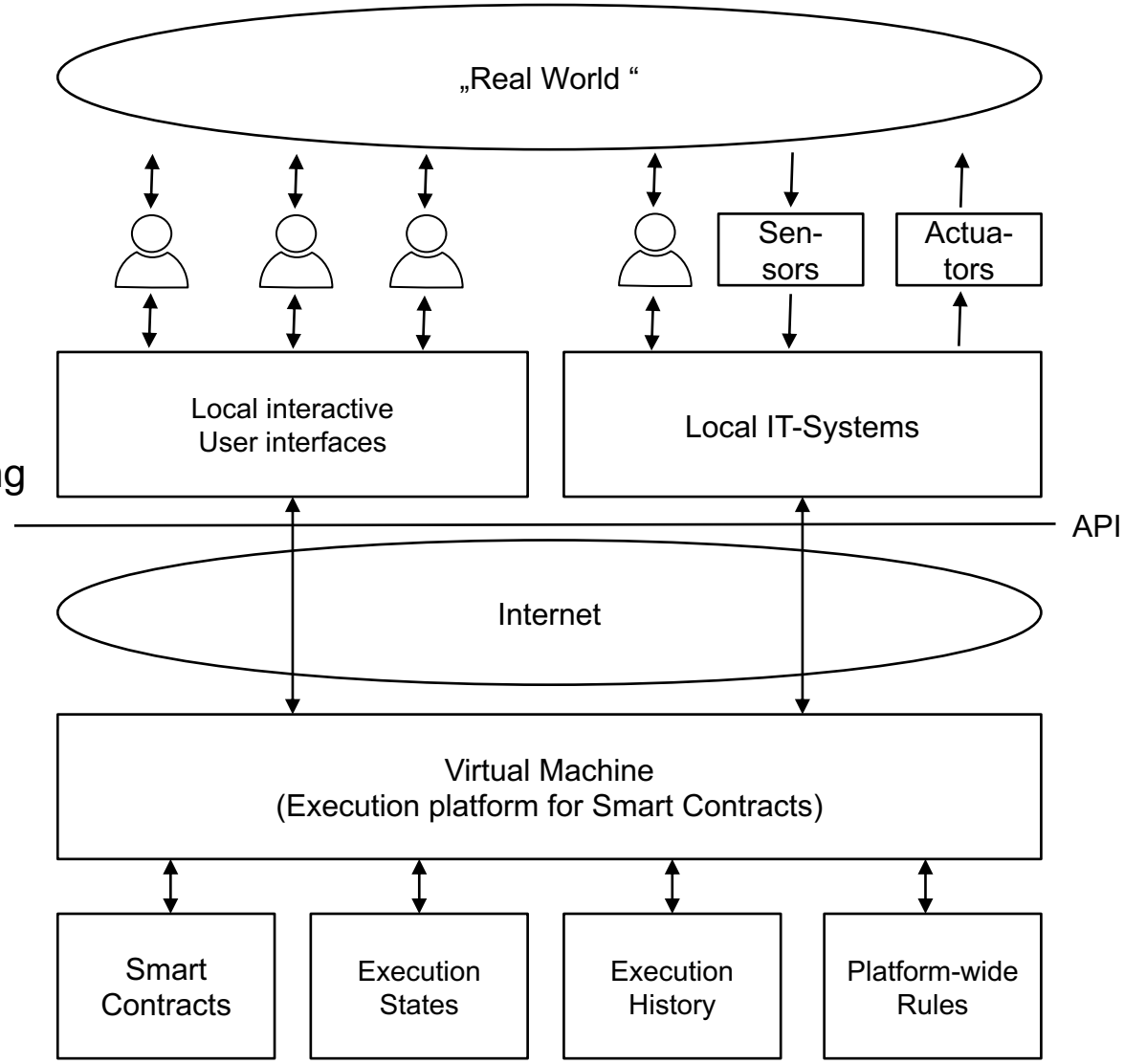
Recap - Lecture 01

Example:

Safe transfusion medicine

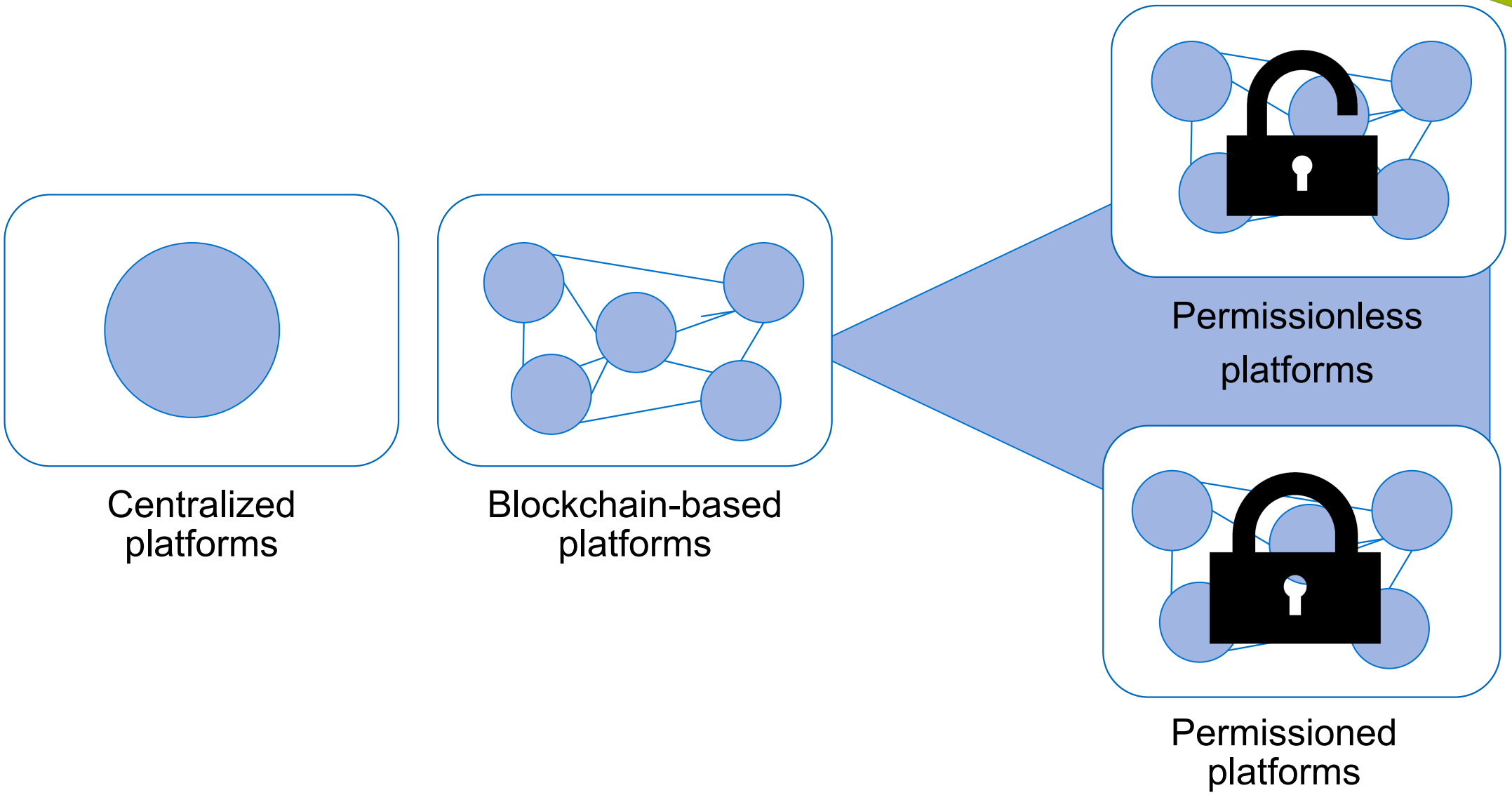
Blood product supply chain involving

- Clinics
- Labs
- Logistic companies
- Blood donor centers



Comparison to Established Centralized Solutions

Recap - Lecture 01



Permissionless Blockchain-based Platforms

Recap - Lecture 01

Opportunities

- Innovation thrust for IT solutions in the global finance system
- Lowered entry barriers for IT-savvy players with limited financial resources
- Impetus to re-evaluate established business models and economic mechanisms

Benefits

- Consensus of nodes on shared set of facts
- Traceability of the complete transaction history
- Optimized for cross-organization collaboration
- Decentralized and redundant

- (Transparency of all transaction details)
- Pseudonymity of the wallet owners
- Financial incentives for network growth

Drawbacks

- Dependency on a crypto currency
- Energy consumption
- Severely limited transaction throughput
- Risk of centralization at the network level

10 Hyperledger - Öz, B., Hoops, F., Gallersdörfer, U., & Matthes, F. (2022). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

Permissioned Blockchain-based Platforms

Recap - Lecture 01

Benefits

- Consensus of nodes on shared set of facts
- Traceability of the complete transaction history
- Optimized for cross-organization collaboration
- Decentralized and redundant

- Upward compatibility with established IT technologies
  - Identity management
  - Development tools
  - Data models & standards
  - …

Drawbacks

- Interoperability between different consortia
- System complexity
- Blockchain governance
  - Ecosystem / participants
  - Applications / contracts
  - Platform technology

Terminology

Public blockchain:
The network has open read and write access for anyone who wants to participate.

Examples:
Ethereum, Monero, Bitcoin, Litecoin

vs

Private blockchain:
Read and write access to the network is limited to a certain set of entities. These networks are (so far) also always permissioned (see following slide). Therefore, private blockchains usually achieve a much higher throughput rate compared to public chains. Private blockchains are a very centralized approach to blockchains and use cases have yet to be found where such solutions are superior to conventional systems.

Examples:
Monax, MultiChain, C-Chain, Quorum

Terminology (cont.)

Permissionless blockchain:
The network is completely open for anyone who wants to participate in reaching consensus. Currently, a highly used Sybil control mechanism is Proof-of-Work (PoW). Network participants are usually financially incentivized by a native currency (e.g., Ether or Bitcoin).

Examples:
Ethereum, Monero, Bitcoin, Litecoin

vs

Permissioned/consortium blockchain:
Participation in the consensus mechanism is restricted to a certain set of entities. This allows the use of more centralized Sybil control mechanisms (e.g., Proof of Authority). Therefore, permissioned blockchains usually achieve a much higher throughput rate compared to permissionless chains.

Examples:
Ripple, Hyperledger Fabric

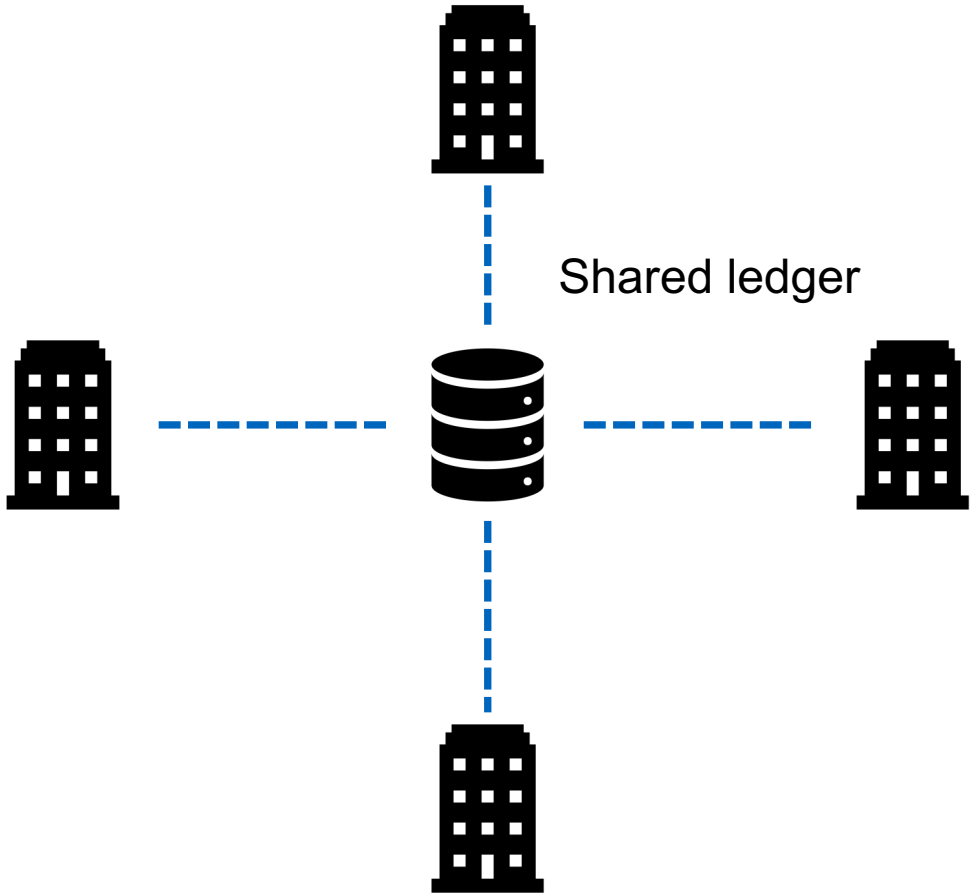
IBM: The difference between public and private Blockchain

Motivation for Permissioned Blockchains

The problem:
Multiple parties that do not fully trust each other need to share information in a tamper proof and secure way.

Challenges:
Having a cost effective and secure data structure.

- Who owns and operates the ledger?
- Who pays for the shared data structure?
- What are allowed operations on the shared data structure?
- What is the process of adding new parties?
- How is consensus achieved?



Business Requirements for Blockchain-based Systems

Privacy
The network must be secured via authentication and authorization mechanisms. Only involved parties should be able to issue transactions and participate in the network.

Smart contracts
The platform should support the execution of business logic on which the participating parties have agreed on.

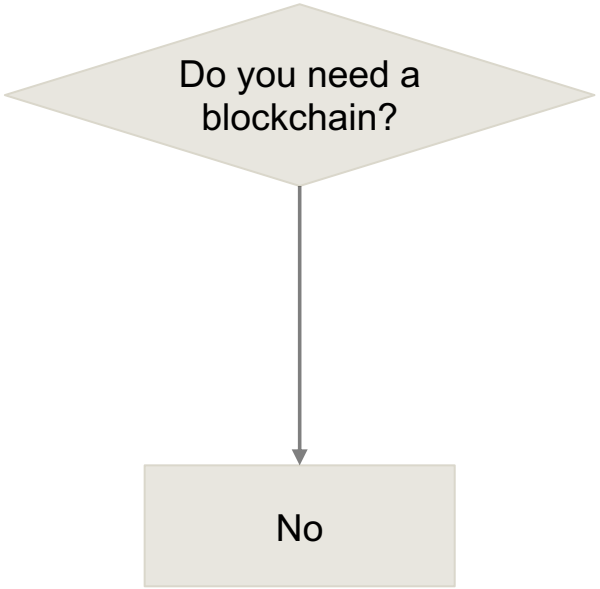
Trust
All transactions that are issued on the network must be verified and valid to a set of rules on which the participating parties agreed on.

Single source of truth
An immutable ledger that keeps track of all transactions. Participating parties must be accountable for transactions they issue on the network.

When to Use Blockchain

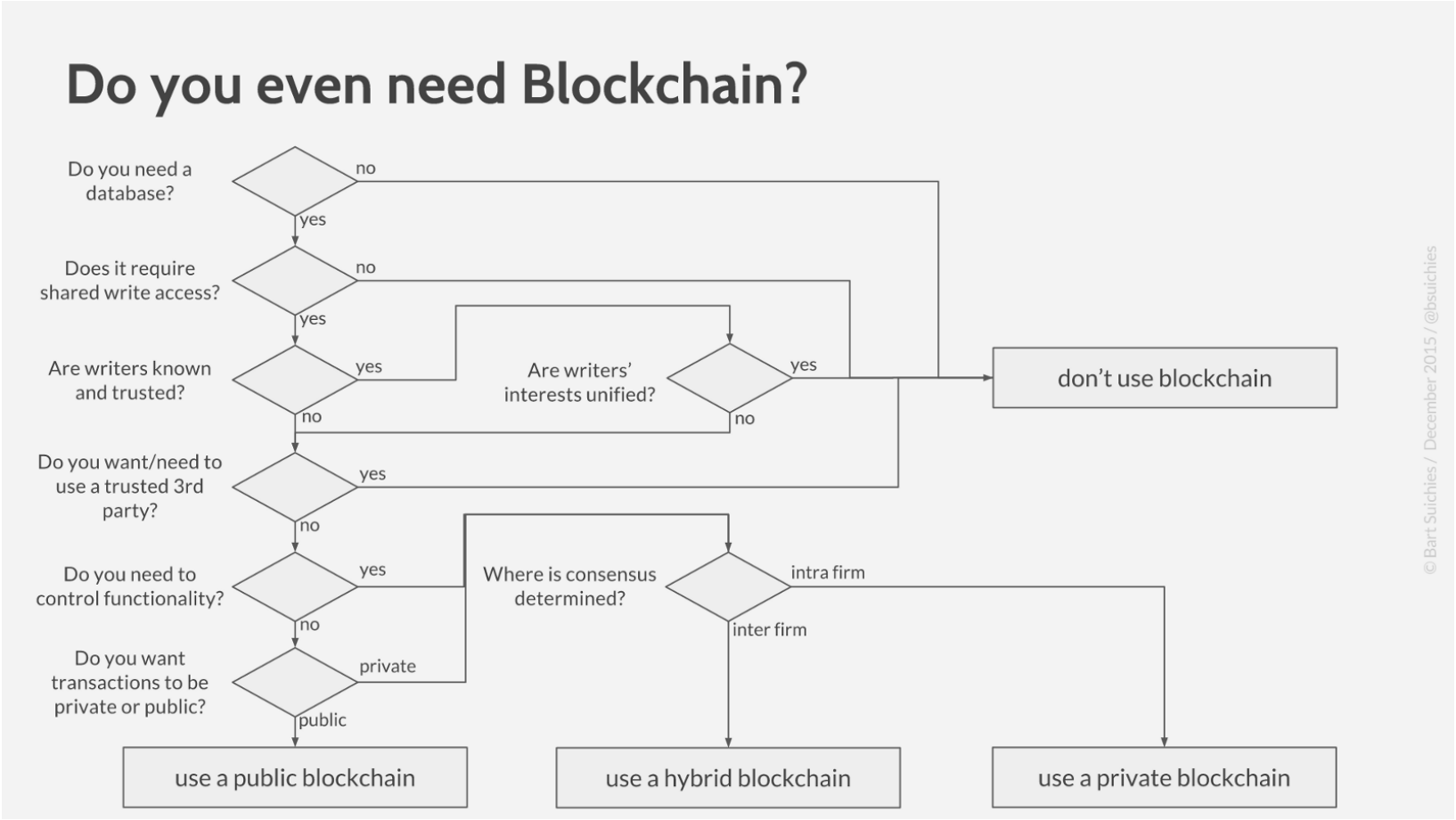
Blockchain is one of the most hyped technology trends nowadays and many enterprises are trying to leverage the technology. However, for finding a use case for a blockchain-based solution it is necessary to understand the technology and the use case well.

Therefore, several individuals and institutions came up with different models for making a decision whether a blockchain-based solution makes sense or not.



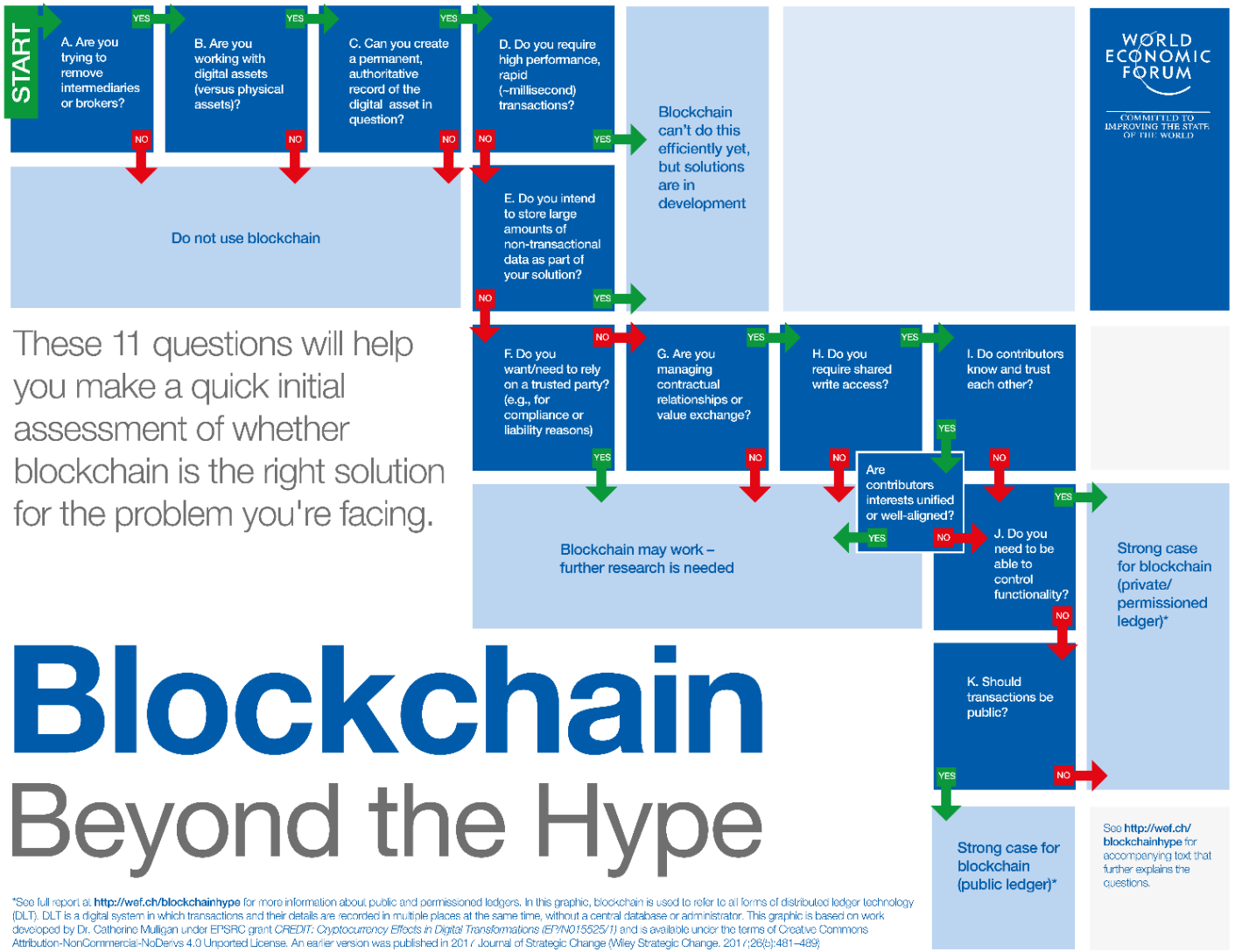
(Joke) Model by Dave Birch (https://twitter.com/dgwbirch?lang=de)

Bart Suichies Model



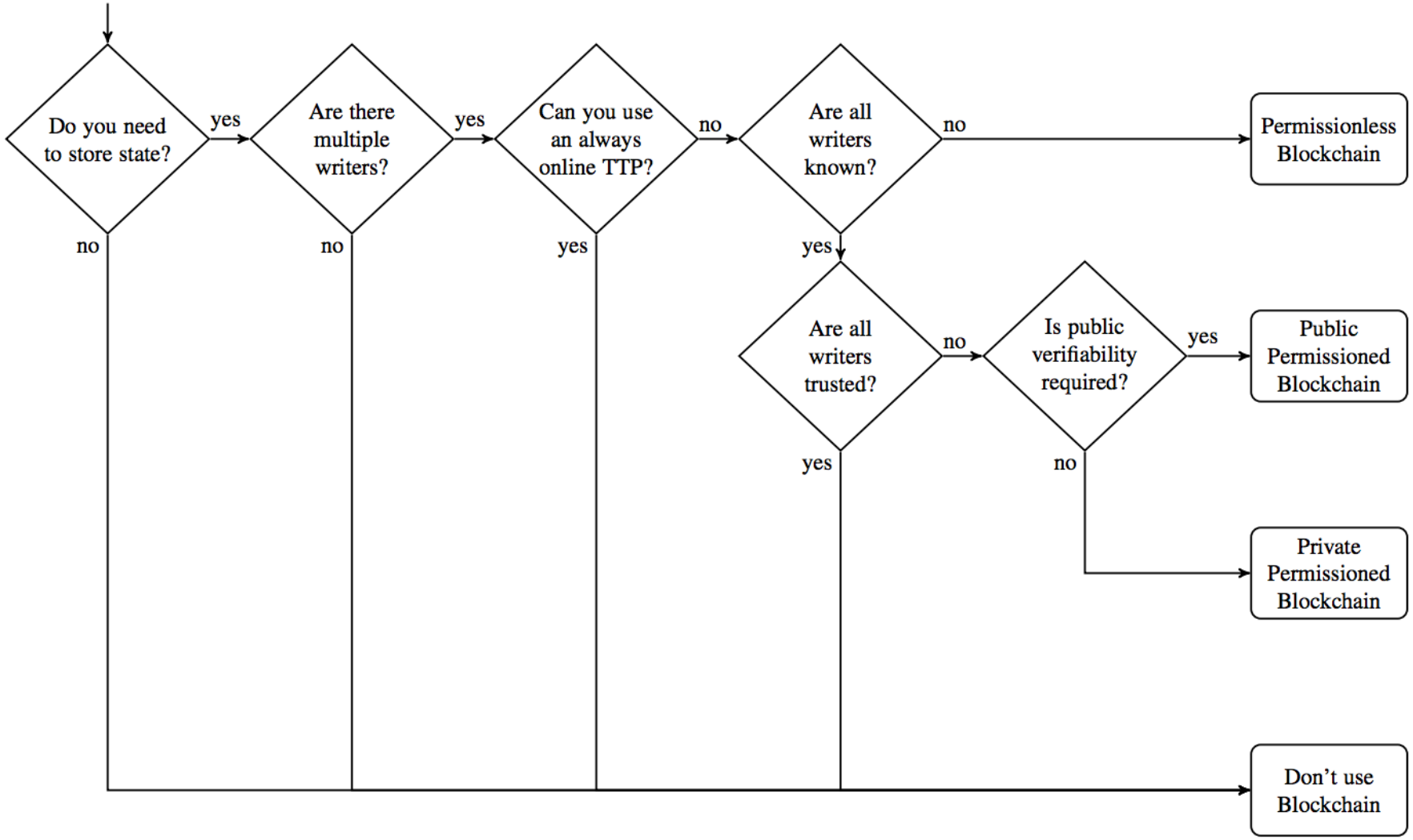
Source: https://medium.com/@bsuichies/why-blockchain-must-die-in-2016-e992774c03b4

Model by the World Economic Forum



Source: https://www.weforum.org/agenda/2018/04/questions-blockchain-toolkit-right-for-business

Model by Karl Wüst and Arthur Gervais



Wüst, Karl, and Arthur Gervais. "Do you need a Blockchain?." *IACR Cryptology ePrint Archive* 2017 (2017): 375.

Outline

1. Beyond Public Blockchains
- Recap
- Motivation
- Use Cases

2. The Hyperledger Project
- Vision and Mission

3. Hyperledger Fabric
- Architecture
- Membership Service Provider
- Application
- Ordering Service
- Peers
- Chaincode
- Order-Execution vs. Execution-Order
- Limitations
- Nodes and Roles
- Transaction Flow
- Channels
- Ledger
- Recent Developments in Hyperledger Fabric

Hyperledger as a Consortium DLT project

HYPERLEDGER

Hyperledger is an umbrella project hosted by the Linux foundation, initiated in 2015. The goal is to create open-source software that enables institutions to set up a shared and trustless ledger databases for collaboration. The most prominent project is Fabric, which can be used to set up a permissioned blockchain infrastructure with the capability of executing smart contracts. Fabric was originally initiated by IBM and Digital Asset.

Screenshot taken from https://www.hyperledger.org/home_aug17-2

Vision and Mission

"[Building] a global, cross-industry community of communities advancing business blockchain technologies." [1]

The vision behind Hyperledger is that there won't and can't be a one-fits-all blockchain solution. This is because of the various different use cases and requirements on the network and consensus. Based on their vision statement, the Hyperledger foundations predicts a future where multiple different public and private blockchain solutions co-exist. Each of them tailored to specific use cases.

"In this environment, Hyperledger serves as a trusted source of innovative, quality-driven open-source software development, creating modular components and platforms. Hyperledger is incubating and promoting enterprise-grade, open-source business blockchain technologies, on top of which anyone can set up apps to meet their business needs." [1]

[1] The Linux foundation: https://www.linuxfoundation.org/projects/case-studies/hyperledger/

Outline

1. Beyond Public Blockchains
- Recap
- Motivation
- Use Cases

2. The Hyperledger Project
- Vision and Mission

3. Hyperledger Fabric
- Architecture
- Membership Service Provider
- Application
- Ordering Service
- Peers
- Chaincode
- Order-Execution vs. Execution-Order
- Limitations
- Nodes and Roles
- Transaction Flow
- Channels
- Ledger
- Recent Developments in Hyperledger Fabric

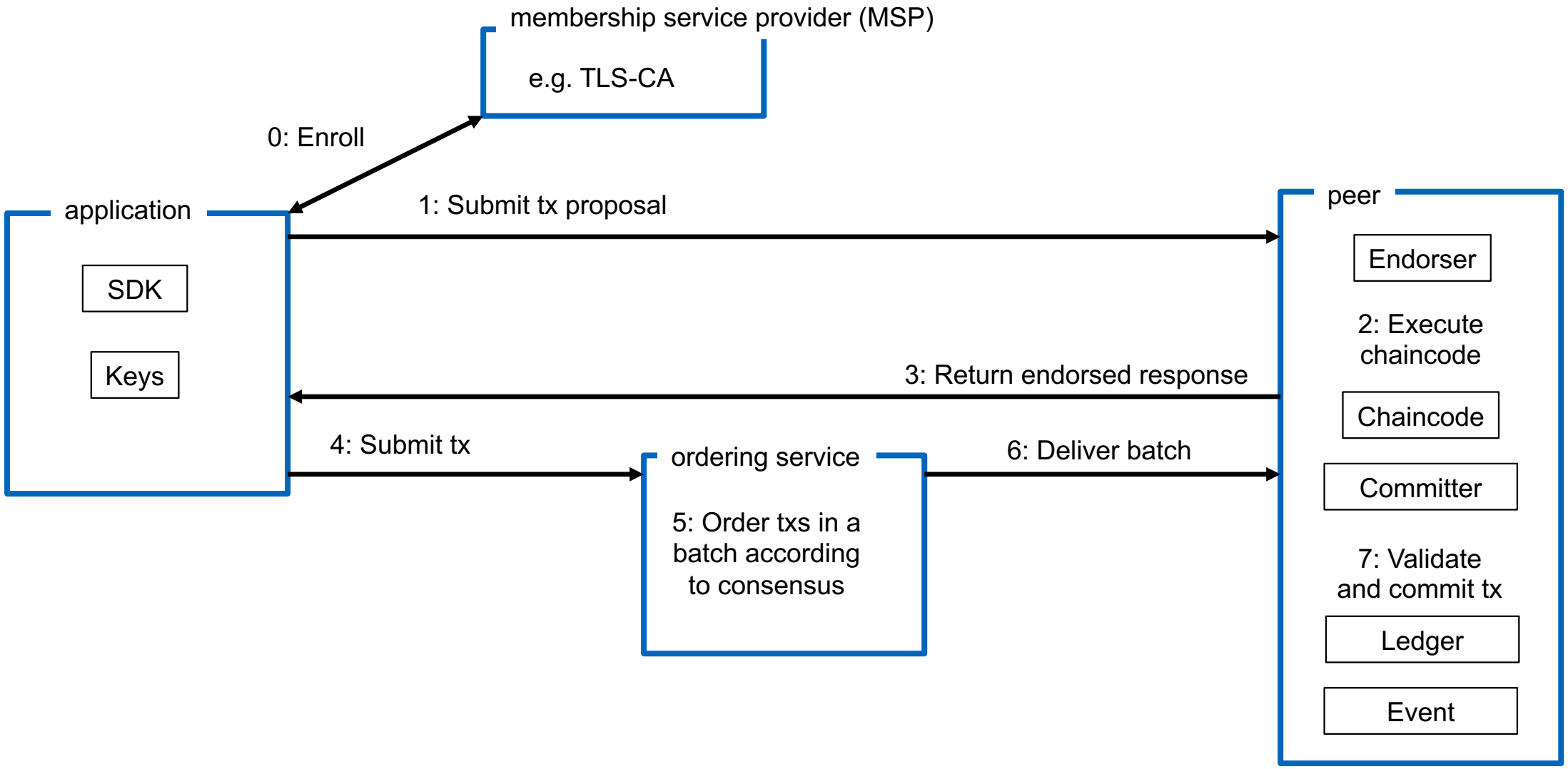
Hyperledger Fabric

Fabric is currently considered as the most sophisticated and used framework in the Hyperledger ecosystem. The current version 2.2.5 is considered stable and ready for productive use. Fabric is highly modular and built with a focus on flexibility and scalability.

Features:

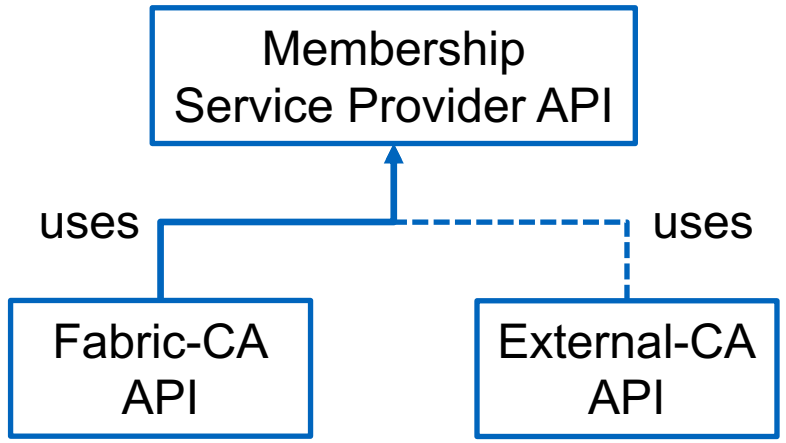
- **Permissioned network:** Fabric is designed as platform to set up permissioned blockchain networks. The framework uses a “Membership Service Provider” (MSP) to enroll new nodes into the network.
- **Channels and multi-ledger:** A core feature of Fabric are so-called channels to establish connections between peers. Each member of a channel maintains a copy of the channel’s specific ledger. Like in other blockchain networks, the ledger contains a sequenced list of all state transitions (transactions).
- **Chaincode:** Fabric allows network operators (system chaincode) and developers (application chaincode) to define certain business logic for a whole channel or for particular transactions.

Architecture and Responsibilities



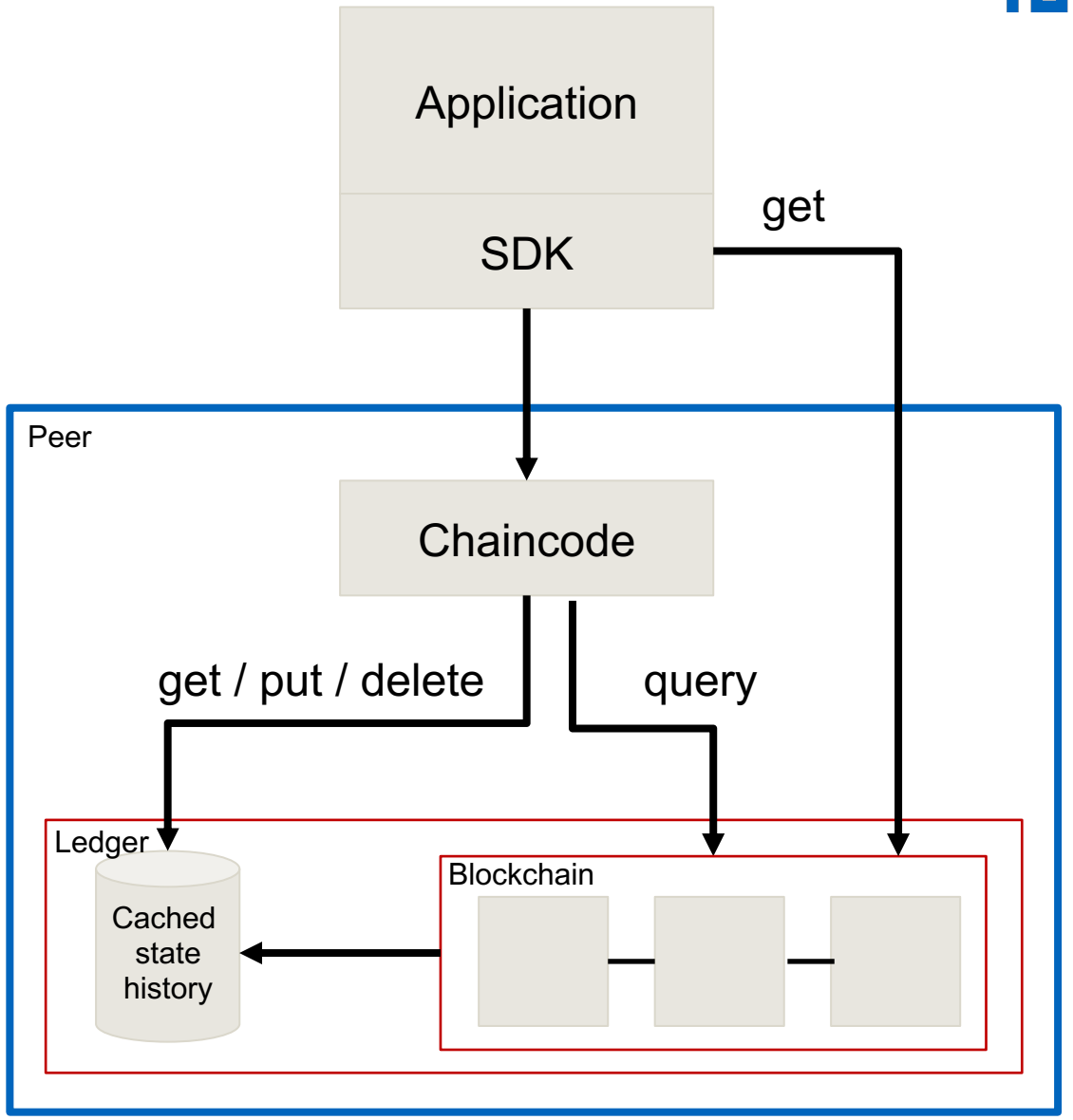
Membership Service Provider (MSP)

- The membership service provider has a pluggable interface that supports a range of credential architectures. A common external certificate authority (CA) is TLS-CA (https certificates).
- Fabric implements it's own CA which is called Fabric-CA and used by default.
- The MSP is responsible for only letting authorized clients join the network. It governs the identity for applications, users, peers and the ordering service.
- Takes care of the the access control in the whole network. It ensures that only authorized clients can join certain channels and execute chaincode.



Applications

- Currently, applications are programs (written primarily in JavaScript, Java, or Go), that invoke calls on smart contracts.
- Applications are very similar to DApps in the Ethereum ecosystem. They usually handle the UI and enable an end user to use the blockchain.
- Applications submit transactions to the network and invoke chaincode calls.
- Communicate directly with peers



Ordering Service

- The ordering service consists of one or more ordering nodes. The nodes do not store any chaincode, nor do they hold the ledger.
- The ordering service enforces the network policy and takes care of the authentication and authorization of the peers.
- Applications must submit transactions to an ordering service node. The node then collects transactions and orders them sequentially. The transactions are then put into a new block and delivered to all peers of a specific channel.
- The blocks are then broadcasted to the peers that hold the chaincode and the ledger. Before the transactions are committed, the peers validate it.
- Consensus is reached when the endorsement of a transaction is accepted, the ordering was successful and the execution was valid and committed.

Peer

- A peer is comparable to full-node in Ethereum or Bitcoin. It maintains the state of the ledger and executes transactions.

- Multiple roles for peers exist:
  - **Committing peer**
    Maintains the state of the blockchain and commits transactions. It verifies endorsements and validate the results of a transaction.

  - **Endorsing peer**
    An endorsing peer is always also a committing peer. The difference is that an endorsing peer additionally takes transaction proposals and executes them to create endorsements.

- The number of peers in a network is not limited and depends on the operating consortium.

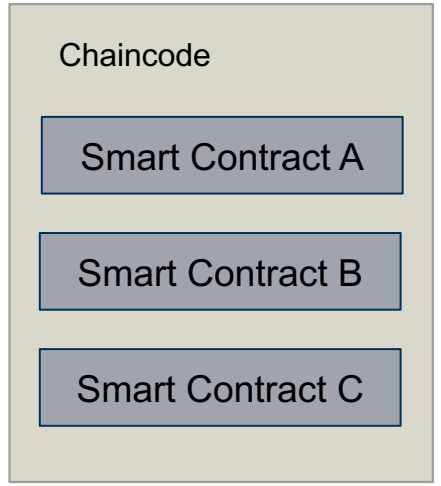
- Chaincode supports the emission of **events**, e.g. when a certain transaction happened. Peers are responsible for handling and delivering such events to subscribers (applications), publish / subscribe architecture pattern.

Chaincode and Smart Contracts

Hyperledger Fabric uses the terms ”smart contracts” and ”chaincode”. While smart contracts refer to the transaction logic of a business object, whereas a chaincode is a **technical container of a group of related smart contracts** for installation and instantiation.

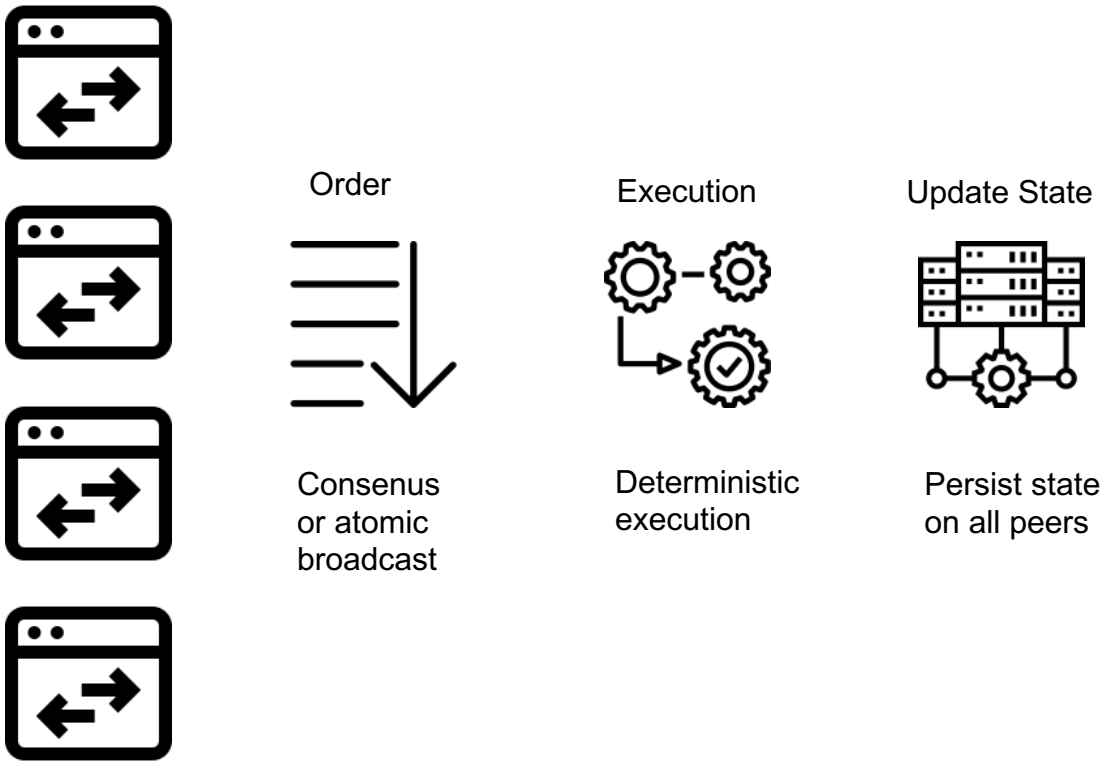
Basically, chaincode is a program written in JavaScript or Java that is installed on a node. A chaincode program allows to write business logic that changes the state of the ledger.

In the past, developing native chaincode for Fabric was very inconvenient. With recent versions, an external chaincode launcher was introduced, making working with Fabric in any capacity (e.g., development, testing, production) significantly easier. Several old tools and pipelines (e.g., Hyperledger Composer) are now obsolete and development is much more comparable to Solidity development.



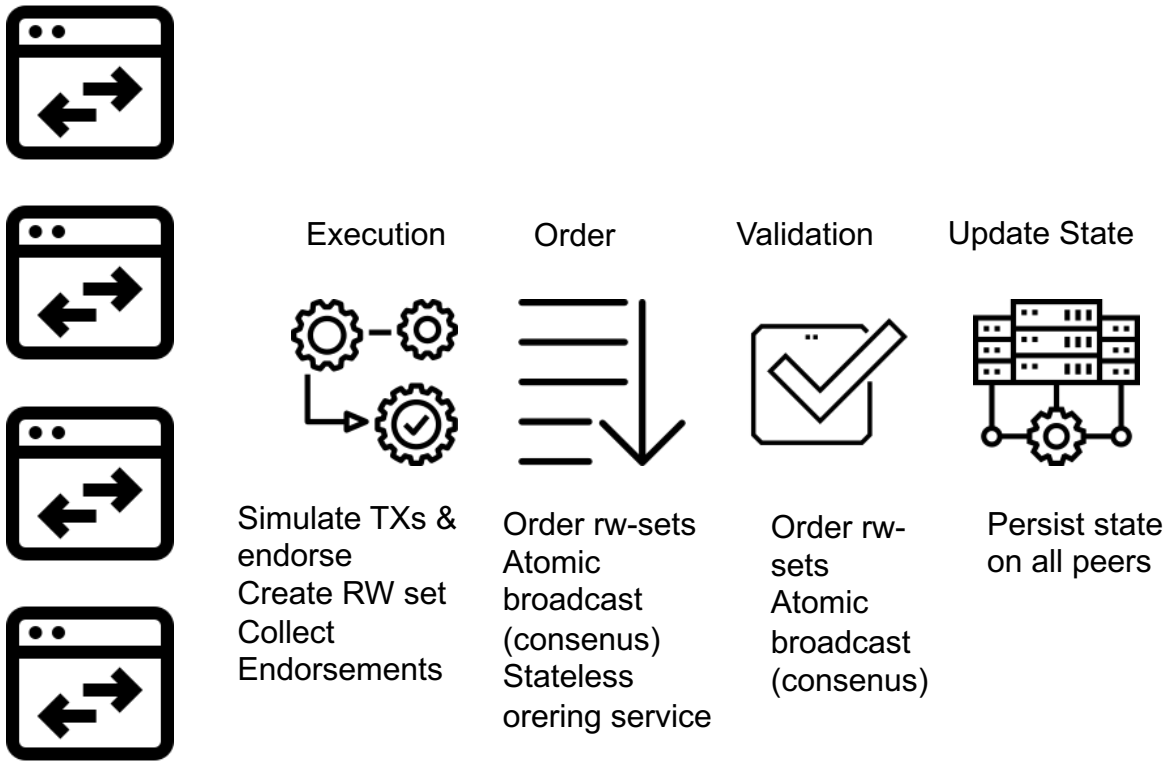
Software Architecture Order-Execution vs. Execution-Order-Validate

Order-Execution



Execution-Order-Validate

Parallel execution of unrelated operations



Limitations of Order-Execution Architecture

Sequential execution on all peers
Executing the transactions sequentially on all peers **limits** the effective **throughput** that can be achieved by the blockchain. In particular, since the throughput is inversely proportional to the execution latency, this may become a performance bottleneck for all but the simplest smart contracts.

Only endorsing peers execute Proposed TXs

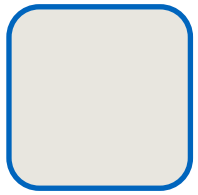
Non-deterministic code
Agreement on the same state is an essential property of current blockchain systems. A single non-deterministic operation could have a devastating effect. This issue is addressed by programming blockchains in domain-specific languages like Solidity in Ethereum. Those languages are usually not as expressive and require additional learning by the programmer.

Applications written in general purpose PL need to follow endorsement policies. If RW-sets differ → Rejection by endorsing peers

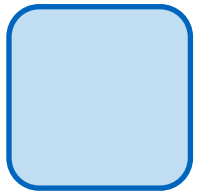
Confidentiality of execution
It often suffices to propagate the same state to all peers instead of running the same code everywhere. Because everyone has to execute everything in the Order-Execution cycle, inherent confidentiality is not provided. Confidentiality can only be achieved through additional data encryption leading to considerable overhead.

Only a few trusted endorsing peers execute the TXs against the chaincode, the rest just updates the state (RW set)

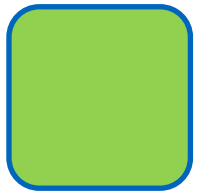
Nodes and Roles



Committing peer
Maintains the state of the blockchain and commits transactions. It verifies endorsements and validate the results of a transaction.



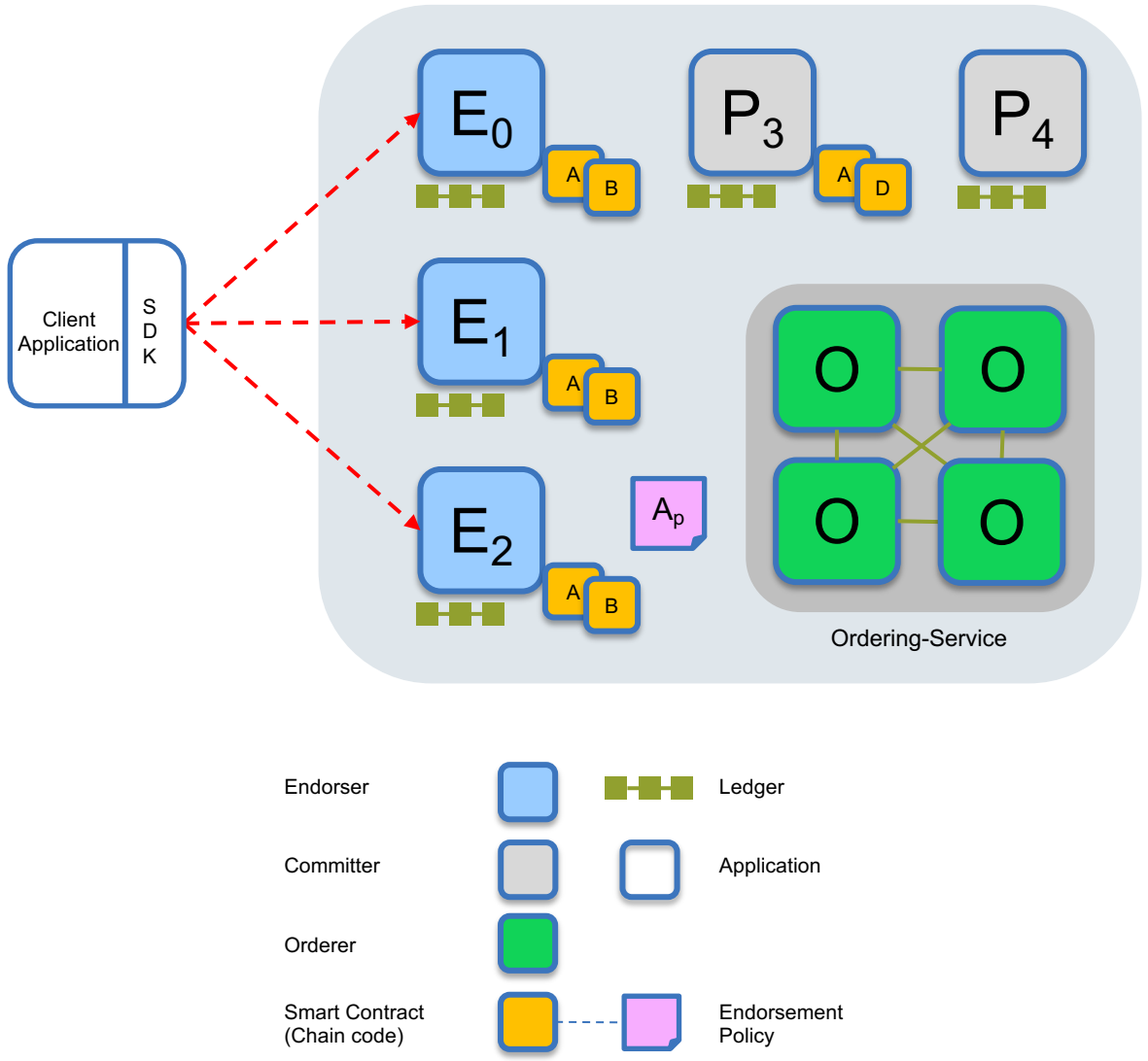
Endorsing peer
An endorsing peer is always also a committing peer. The difference is that an endorsing peer additionally takes transaction proposals and executes them to create endorsements.



Ordering service node
Applications must submit transactions to an ordering service node. The node then collects transactions and orders them sequentially. The transactions are then put into a new block and delivered to all peers of a specific channel. Does neither hold ledger nor chaincode.

Transaction Flow (Step 1)

Application Submits Transaction Proposal



Application creates a transaction proposal for **chaincode A** and sends it to all peers that are part of the endorsement policy $\mathbf{A_p}$

Endorsement Policy $\mathbf{A_p}$

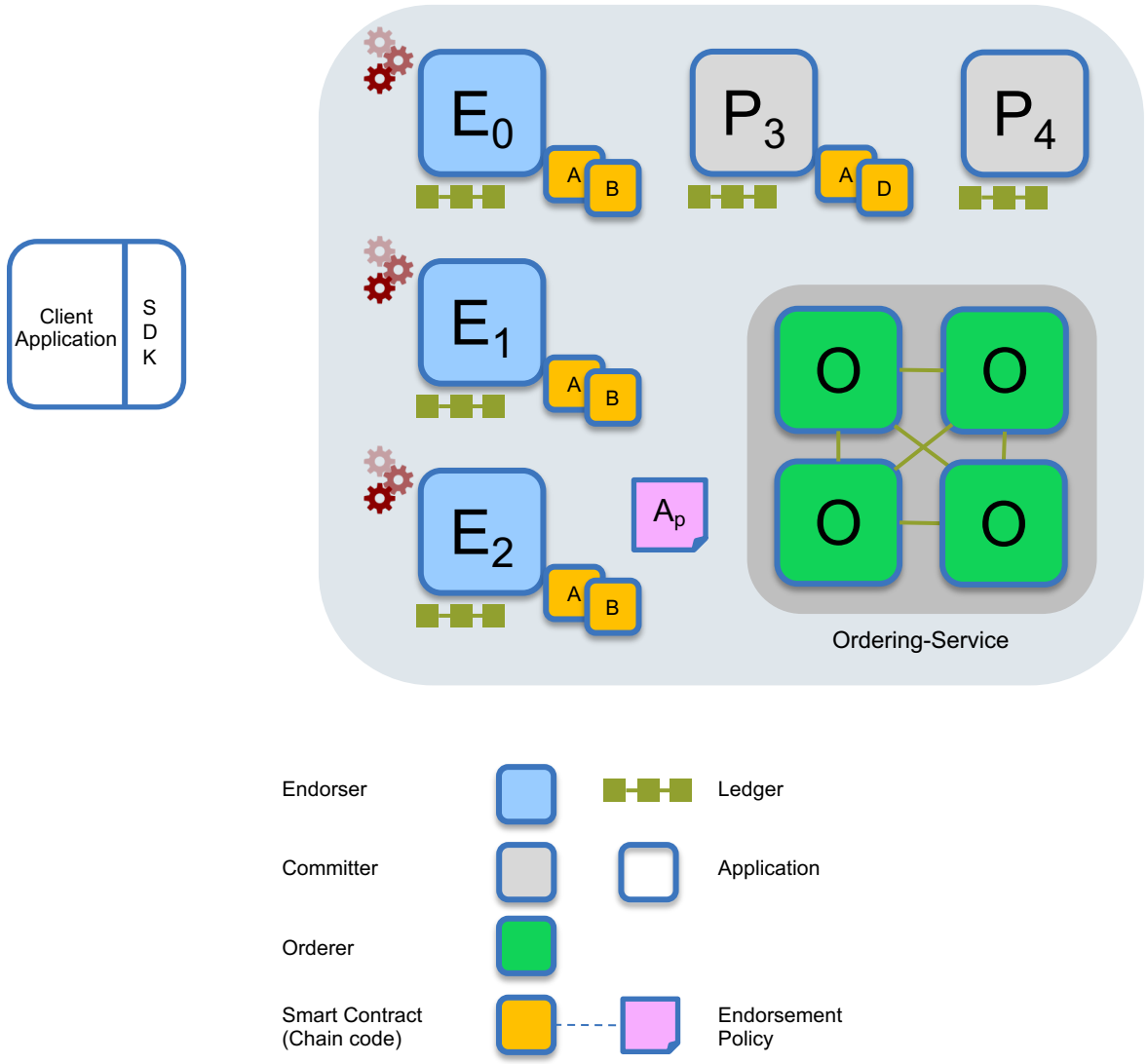
- $\mathbf{E_0}$, $\mathbf{E_1}$ and $\mathbf{E_2}$ must sign the transaction
- ($\mathbf{P_3}$, $\mathbf{P_4}$ are not part of the policy)

Since only the peers $\mathbf{E_0}$, $\mathbf{E_1}$ and $\mathbf{E_2}$ are part of the endorsement policy $\mathbf{A_p}$, it is not required to send the transaction proposal to $\mathbf{P_3}$ and $\mathbf{P_4}$

Source: https://www.hyperledger.org/resources/universities#educators

Transaction Flow (Step 2)

Endorsing Peers Execute the Transaction Proposal



E_0, **E_1** and **E_2** will each simulate the execution of the *proposed* transaction from the application.

None of these executions will update the ledger.

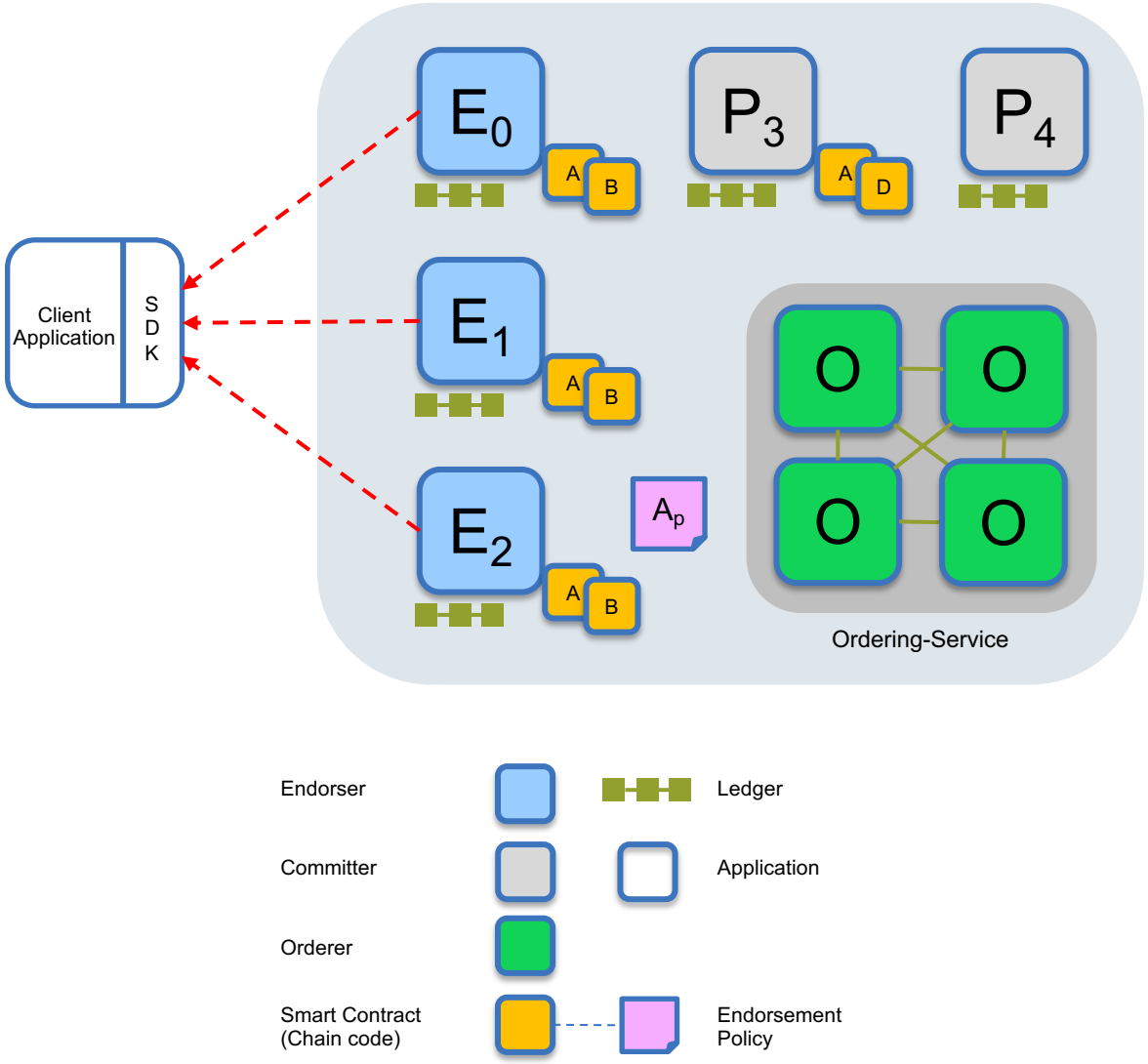
The **simulation will be used to capture** the read and write operations on the ledger. After the transaction is executed, each peer will have a generated **read/write set** (RW set)

```
<TxReadWriteSet>
  <NsReadWriteSet name="chaincode1">
    <ReadSet>
      <read key="K1", version="1">
      <read key="K2", version="1">
    </ReadSet>
    <WriteSet>
      <write key="K1", value="V1">
      <write key="K3", value="V2">
      <write key="K4", isDelete="true">
    </WriteSet>
  </NsReadWriteSet>
<TxReadWriteSet>
```

Source: https://www.hyperledger.org/resources/universities#educators

Transaction Flow (Step 3)

Read/Write Set is Signed and Returned to the Application

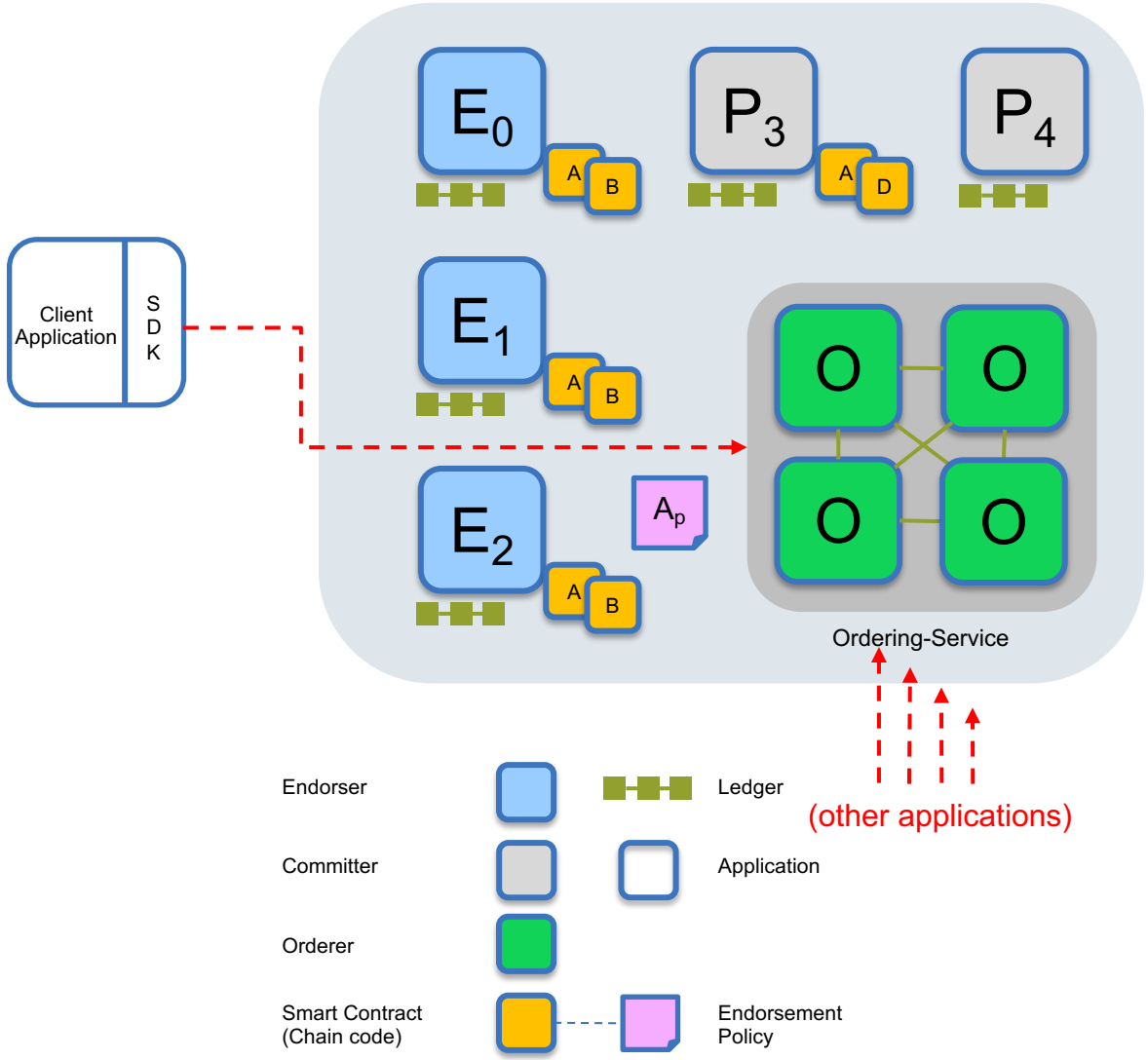


E_0, **E_1** and **E_2** will each sign their generated read/write set and return it to the application that invoked the transaction.

Source: https://www.hyperledger.org/resources/universities#educators

Transaction Flow (Step 4)

Signed Responses are Sent to the Ordering Service



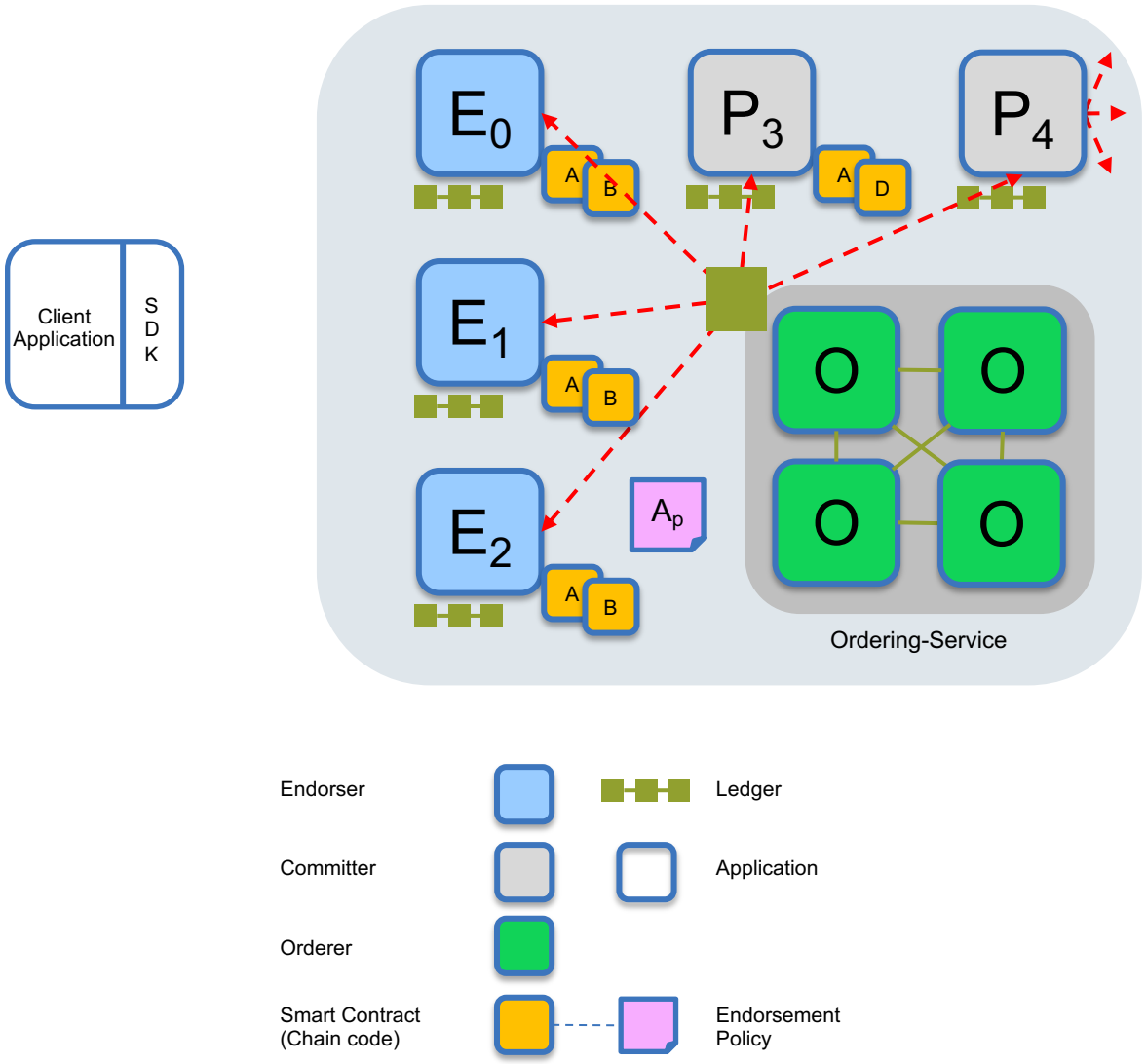
The application submits the signed responses from $\mathbf{E_0}$, $\mathbf{E_1}$ and $\mathbf{E_2}$ to the ordering service.

The ordering service is responsible to order all transactions from all applications in the network. The service tries to **serialize** the incoming transactions.

Source: https://www.hyperledger.org/resources/universities#educators

Transaction Flow (Step 5)

Ordering Service Distributes New Block to all Committing Peer



The ordering service creates a new block based on the incoming transactions. The block will then be broadcasted to all committing peers in the channel.

Currently, the ordering service supports three different ordering algorithms:

- SOLO (single node, development)
- Kafka (blocks map to topics)
- Raft (crash fault tolerant (CFT), follows a "leader and follower" model)

Source: https://www.hyperledger.org/resources/universities - educators

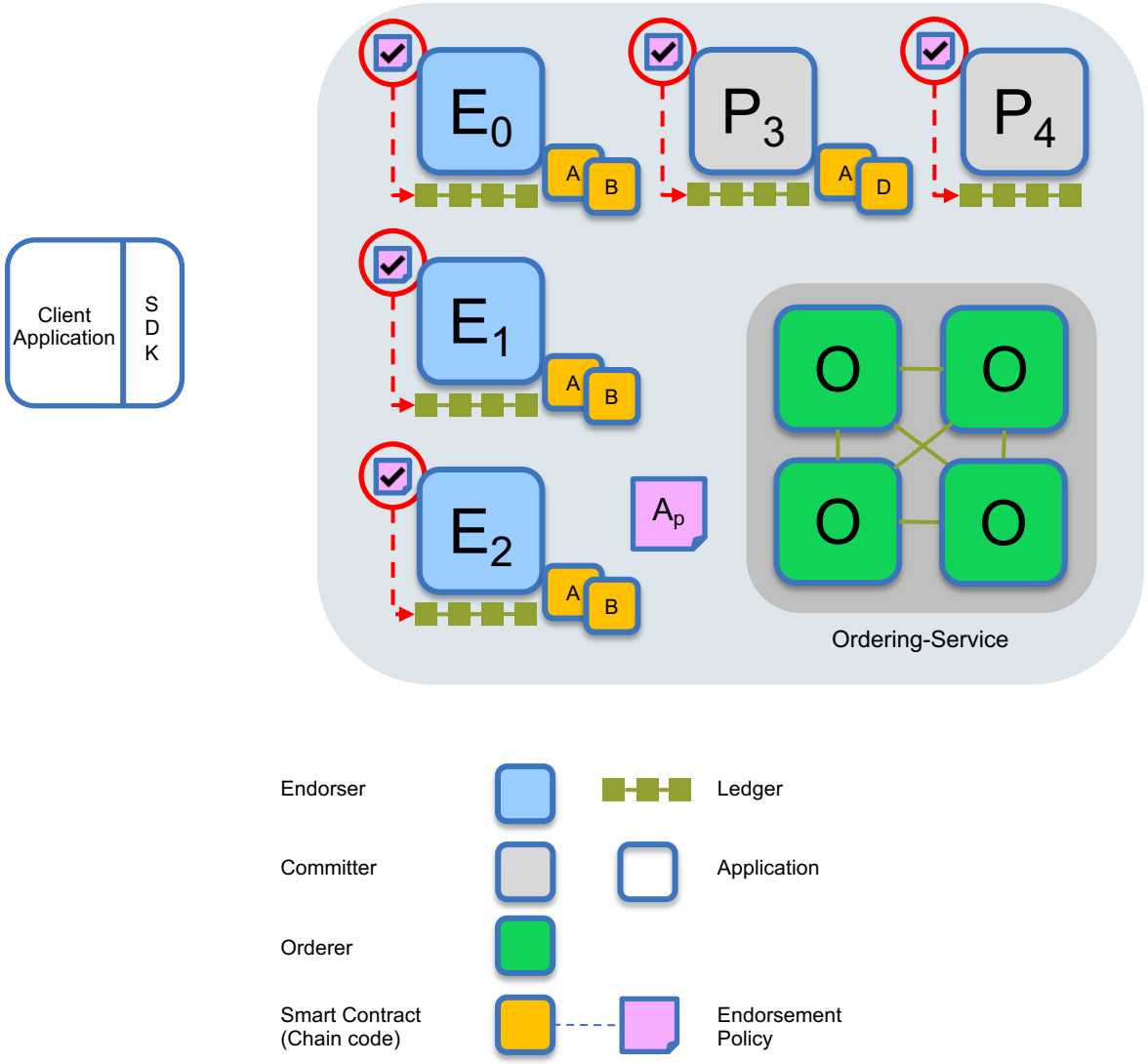
Raft

- Raft is a crash fault tolerant ordering service.
- It uses a “leader and follower” model, in which a leader is dynamically elected among the ordering nodes in a channel, and that leader replicates messages to the follower nodes.
  - Because the system can sustain the loss of nodes, including leader nodes, as long as there is a majority of ordering nodes remaining, Raft is said to be “crash fault tolerant” (CFT). In other words, if there are three nodes in a channel, it can withstand the loss of one node.
- Raft is the first step toward Fabric’s development of a byzantine fault tolerant (BFT) ordering service.

Source https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html#raft

Transaction Flow (Step 6)

Committing Peers Validate the Transaction



All committing peers in the channel validate the transaction (read/write set) according to the endorsement policy of the chaincode A.

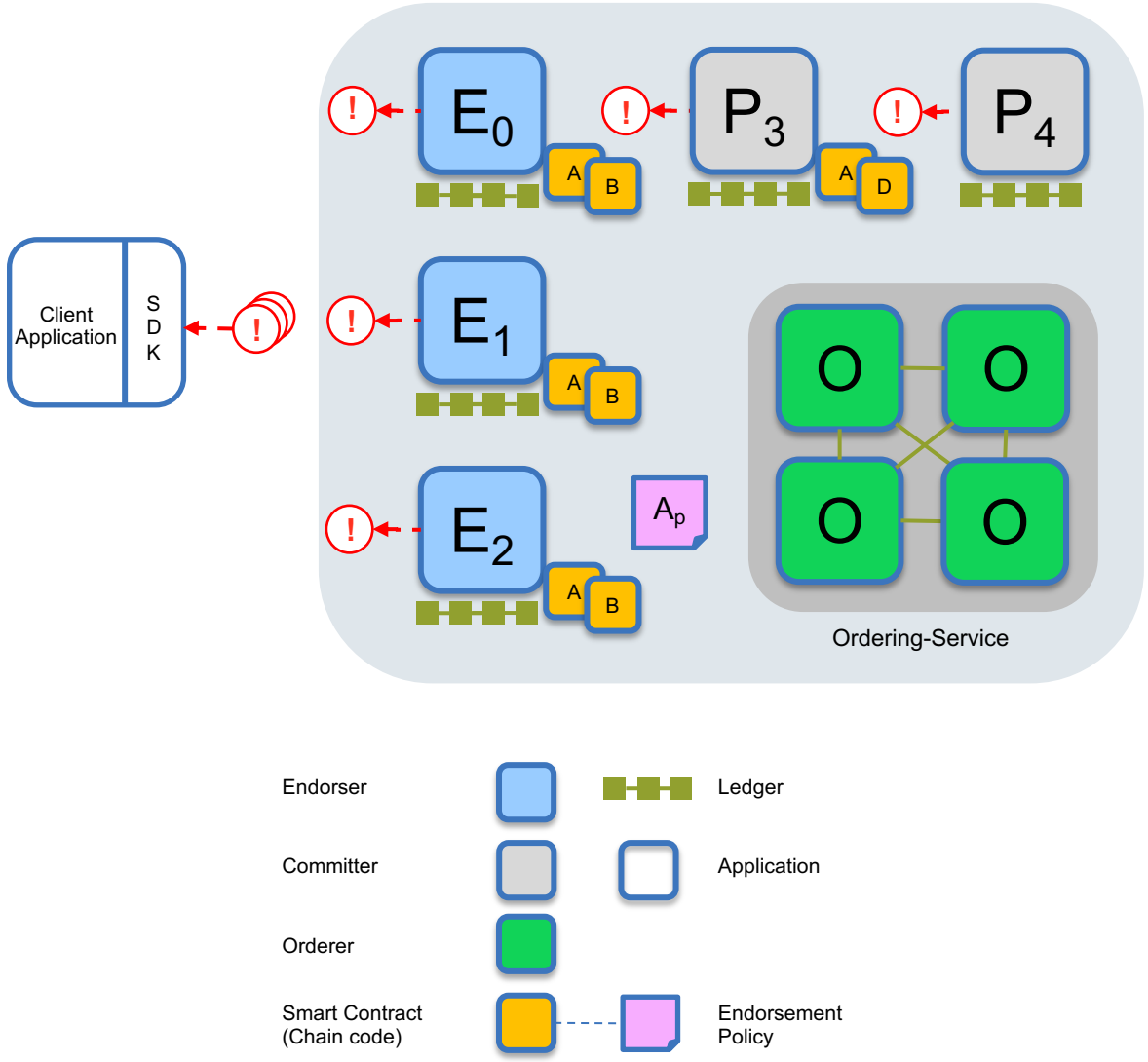
If the transaction is valid, the **read** and write set is written to the ledger and added as a new block to the blockchain.

The databases used for caching are updated with the new state information accordingly.

Source: https://www.hyperledger.org/resources/universities#educators

Transaction Flow (Step 7)

Notify Application

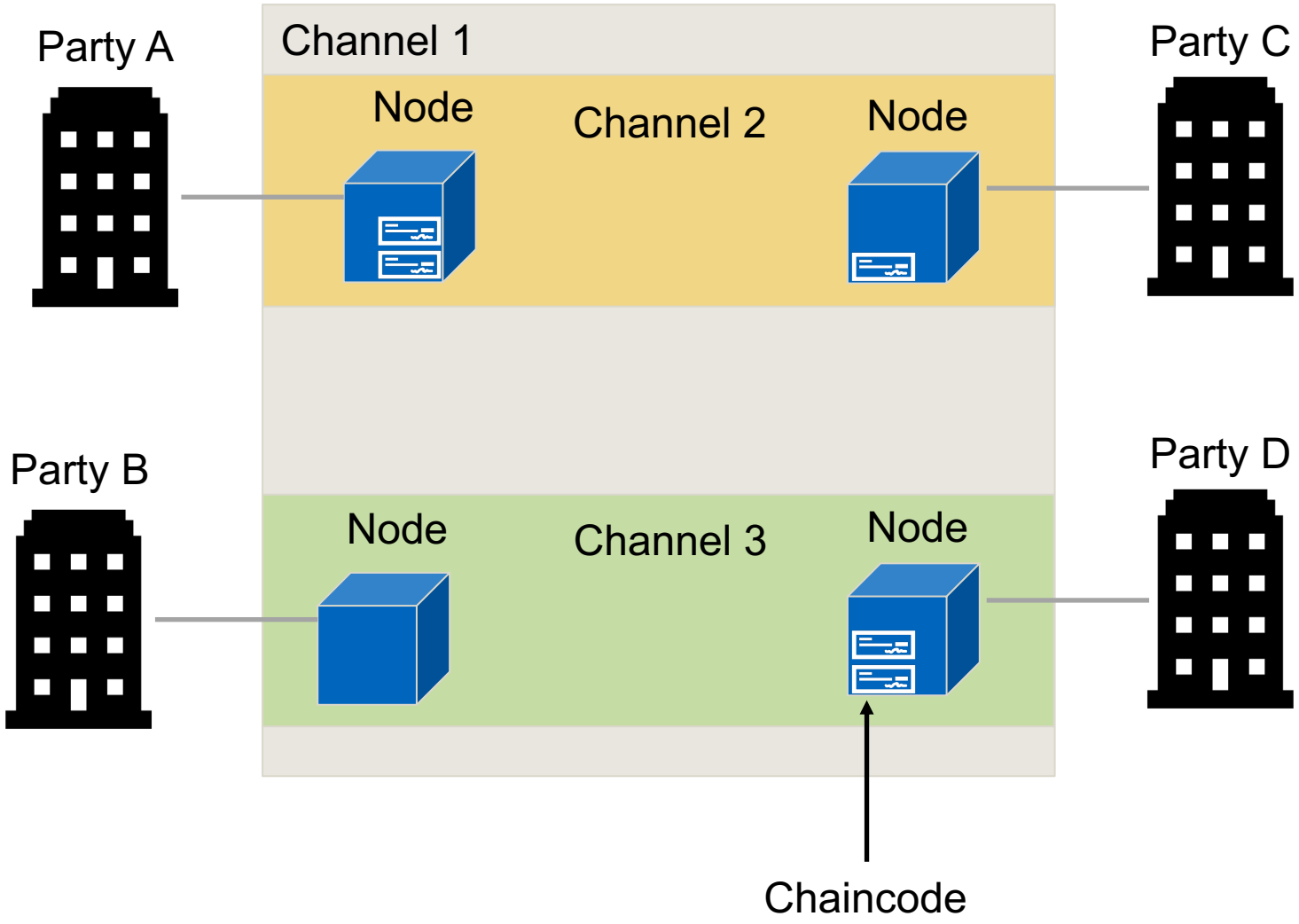


Applications can register to be notified by the peers once the transaction is done. The peers will emit an event that indicates if the transaction succeeded or failed.

Applications can also subscribe to state changes of the ledger, i.e. a connected peer will then notify them if new blocks are added to the chain.

Source: https://www.hyperledger.org/resources/universities#educators

Channels



Channels (cont.)

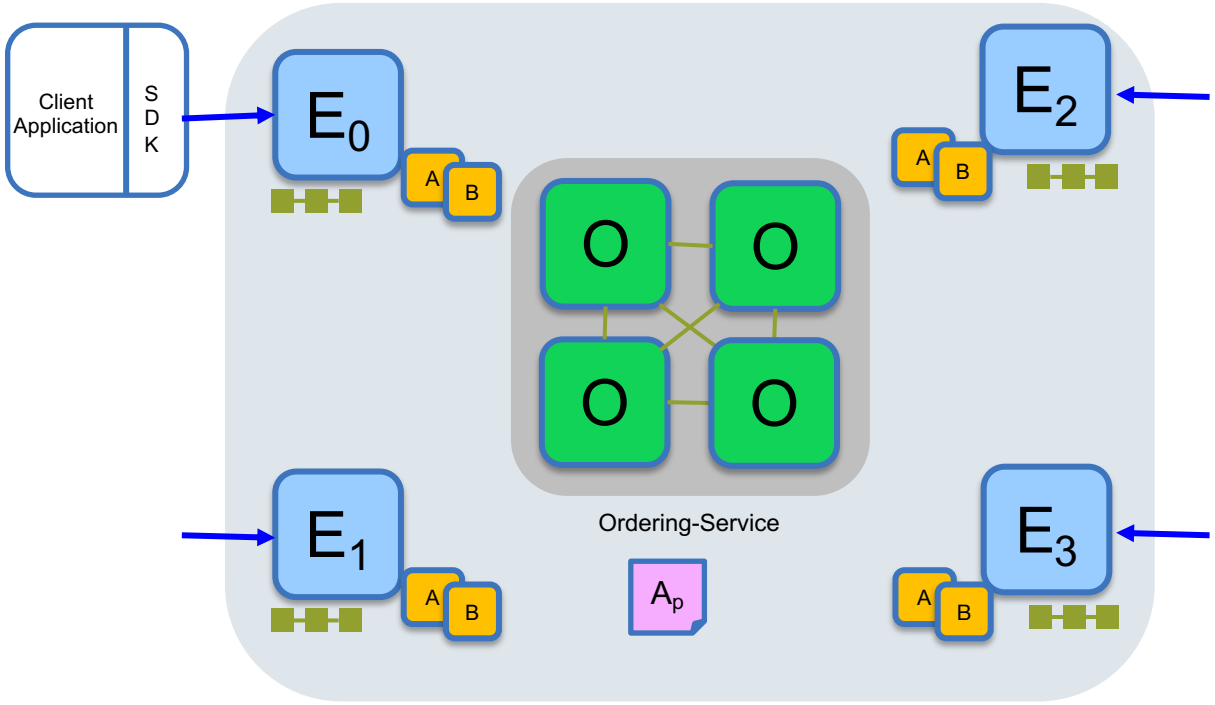
A **channel is a separate blockchain**.

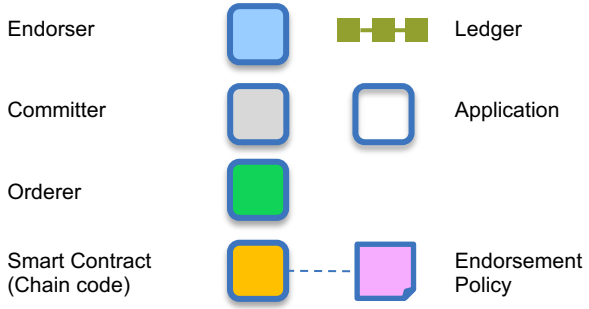
This blockchain is only managed by a subset of all available nodes as defined by the membership service provider.

- Separate channels isolate transactions on different ledgers
- Consensus takes place within a channel by members of the channel
  - Other members on the network are not allowed to access the channel and will not see transactions on the channel
- A chaincode may be deployed on multiple channels with each instance isolated within its channel
- Peers can participate on multiple channels

➔ Concurrent execution for performance and scalability

Single Channel Network





All peers connected to the same channel (**blue**)

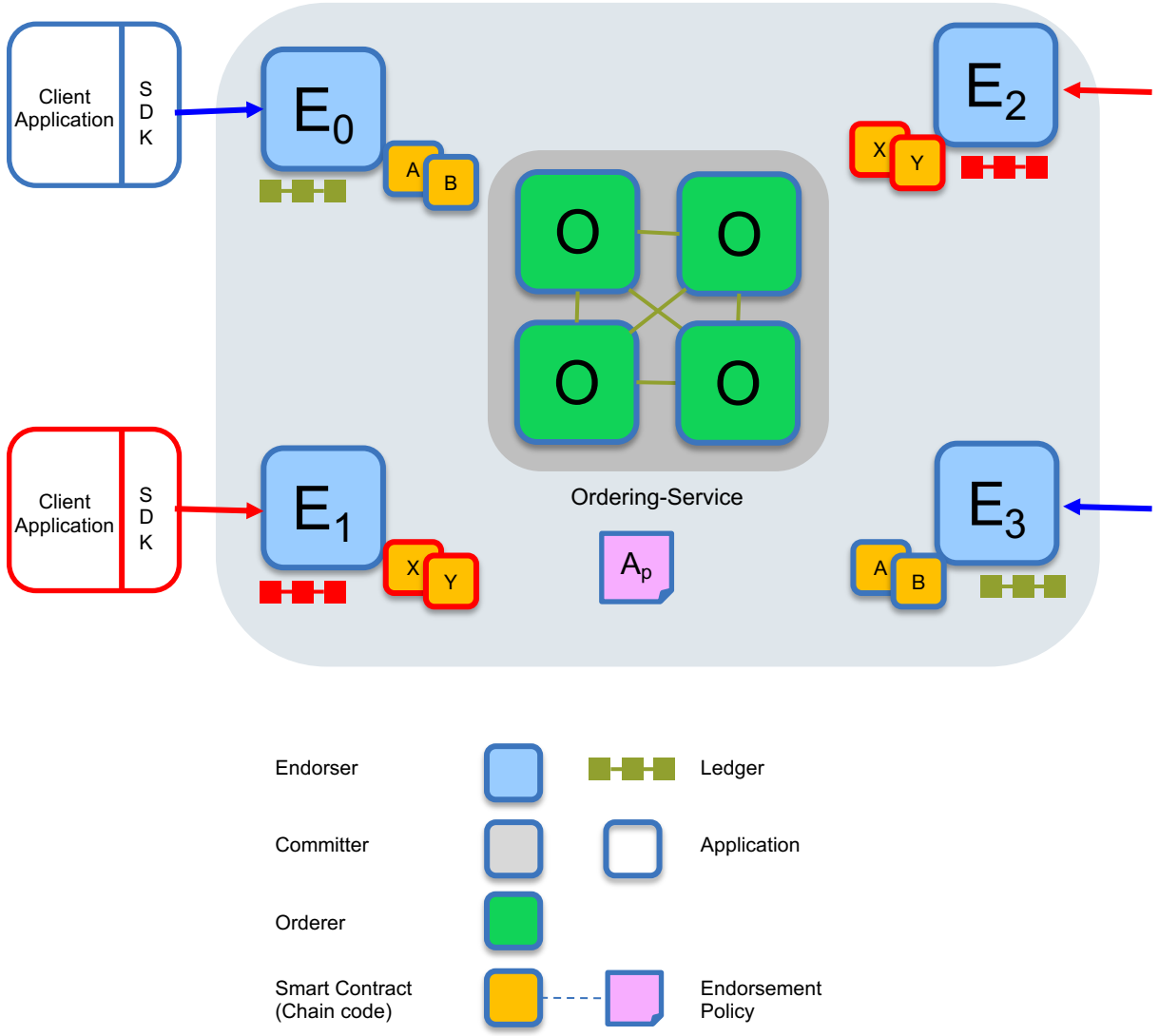
All peers have the same chaincode and maintain the same ledger

Endorsements by peers **E_0, E_1, E_2**, and **E_3**

A single channel network is similar to traditional, public blockchain network where the whole world state is shared with all participating nodes.

Source: https://www.hyperledger.org/resources/universities#educators

Multi Channel Network



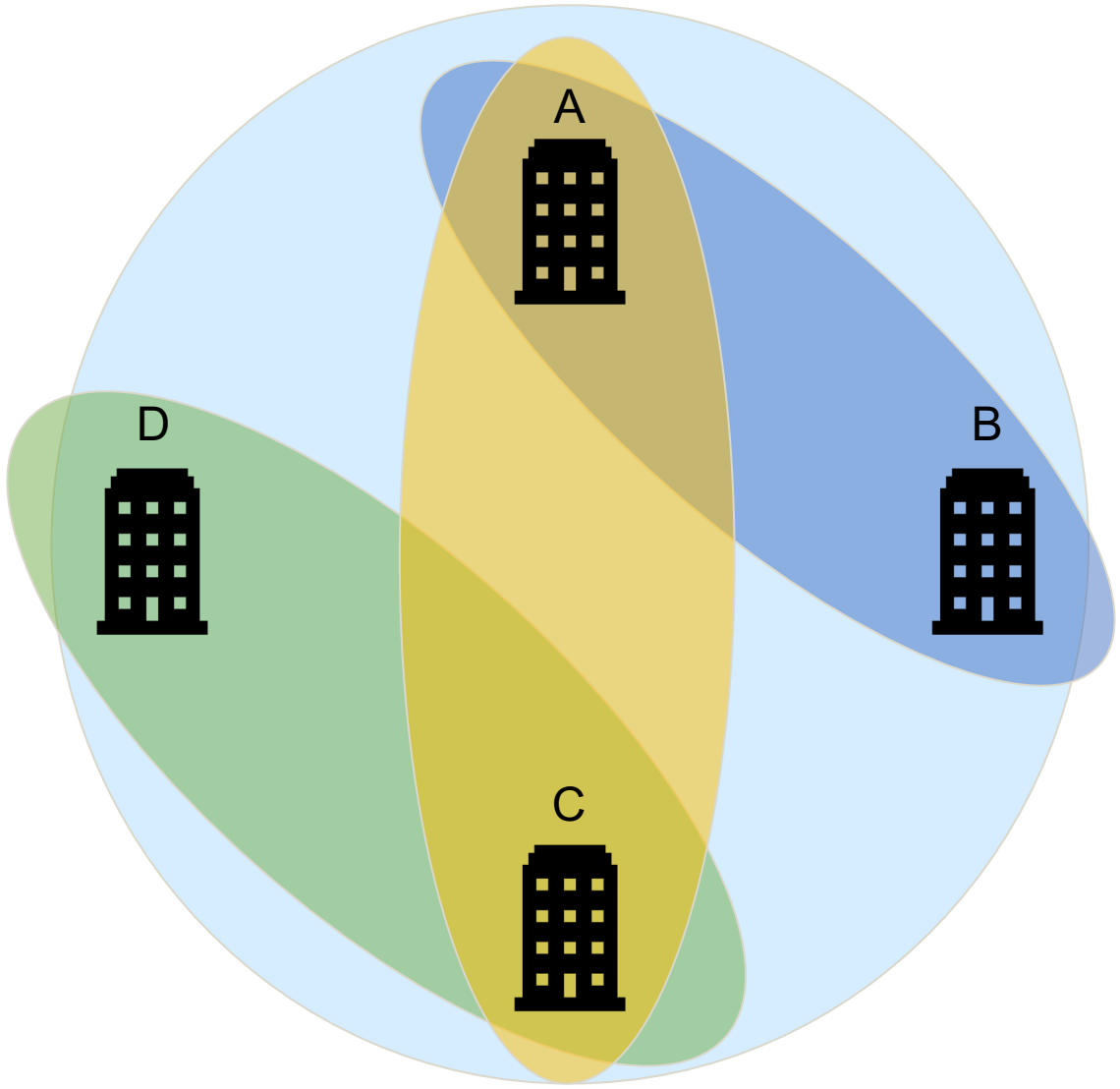
Peers $\mathbf{E_1}$ and $\mathbf{E_2}$ connect to the **red** channel for **X** and **Y** chaincodes.

Peers $\mathbf{E_0}$ and $\mathbf{E_3}$ connect to the **blue** channel for **A** and **B** chaincodes.

Fabric support multi channel networks. Each peer only shares the ledger with nodes that are in the same channel. Smart contracts also operate on a channel basis and are not globally available.

Source: https://www.hyperledger.org/resources/universities#educators

Example of a Multi Channel Network



A ledger that is shared among all consortium members

A ledger that is shared between A and B (C and D cannot access it)

A ledger that is shared between C and D (A and B cannot access it)

A ledger that is shared between A and C (B and D cannot access it)

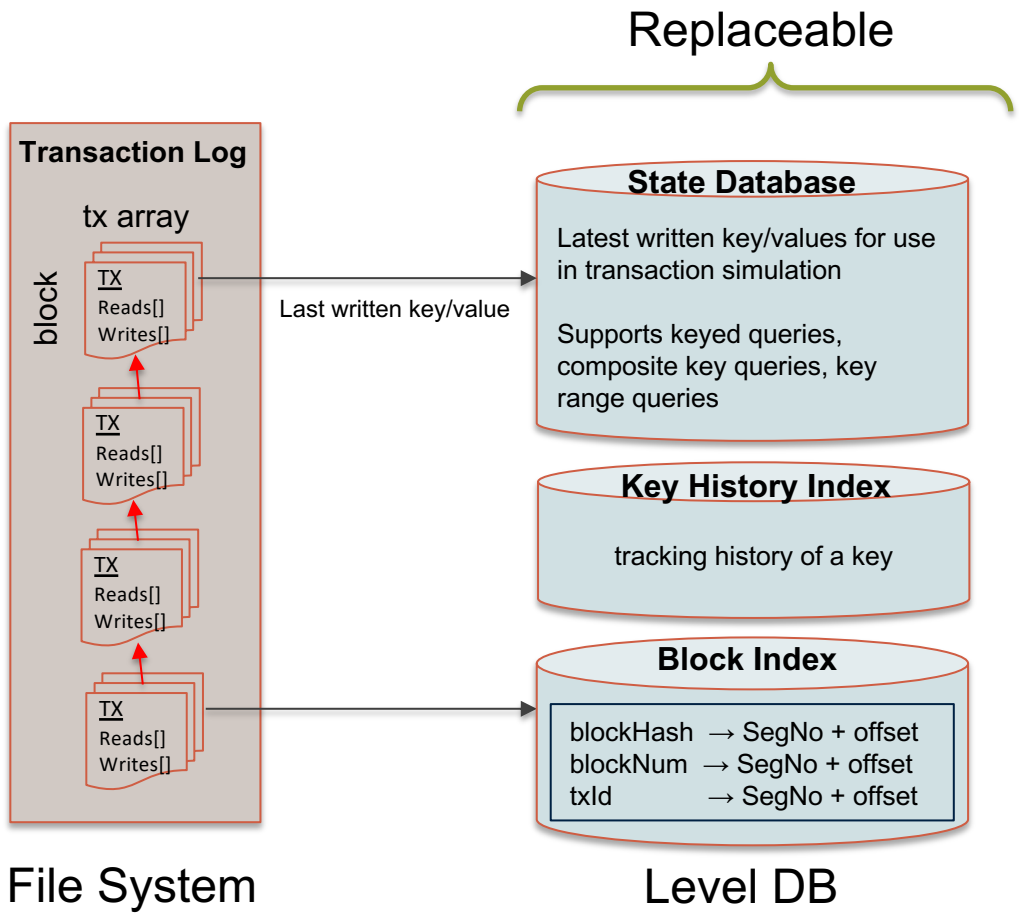
Ledger

A block in Fabric consists of transactions that define reads and writes on the shared state data.

The blockchain is saved on the file system, just like in Ethereum and most other blockchain systems.

Reading from the blockchain can be slow when the blockchain on the file system has to be scanned.

Fabric uses a database that contains the current state of the ledger. This provides peers a very efficient data structure to retrieve ledger information.



Source: https://www.hyperledger.org/resources/universities#educators

Recent Developments in Hyperledger Fabric

While the core principles have stayed the same since Fabric 1.0, many additional features and user experience improvements have been introduced. Here is an overview of just some of them:

Fabric Gateway

- Provides an abstraction layer (a simplified API) that clients can use for easy interaction with the blockchain

Channel Administration Improvements

- Now it is possible to remove peers from a channel
- Easier joining of new peers to a channel by loading a snapshot form an existing peer

Smart Contract Administration Improvements

- Multi-party agreement on chaincode parameters and chaincode updates
- Policy changes without repackaging chaincode and reinstalling it
- Installing multiple chaincodes packaged together as one package

If you are interested in working with Fabric, the documentation has become a great place to start.

Source: https://hyperledger-fabric.readthedocs.io/en/release-2.4/whatsnew.html - what-s-new-in-hyperledger-fabric-v2-0-v2-1-v2-2