Module 5

Hyper ledger

Hyper ledger as a protocol

Fabric

Hyper ledger Fabric

Saw tooth lake

Corda

Hyper ledger

Hyperledger is an open-source project under the Linux Foundation where people can come and work on the platform to develop blockchain-related use cases.

Introduction To Hyperledger

Hyperledger provides the platform to create personalized blockchain services according to the need of business work.

Unlike other platforms for developing blockchain-based software, Hyperledger has the advantage of creating a secured and personalized blockchain network.

Example: Consider a situation when person X wants to buy medicine from person Y, who was a doctor living in another country.

- As medicine requirement is one person's private needs, they need to maintain the data confidentially.
- But Dr. Y is selling medicine in the network to so many people, in the case of the public blockchain, every transaction will get updated in the network to all the peers.
- That's where hyperledger finds its significance. In the hyperledger, the parties are directly connected and the concerned people's ledger will be updated.
- Hence providing privacy and confidentiality.

Hyperledger Technology Layers

Hyperledger uses the following key business components:

1. **Consensus layer:** It takes care of creating an agreement on the order and confirming the correctness of the set of transactions that constitute a block.
2. **Smart layer:** This layer is responsible for processing transaction requests and authorizing valid transactions.
3. **Communication layer:** It takes care of peer-to-peer message transport.
4. **Identity management services:** these are important for establishing trust on the blockchain.
5. **API:** It enables external applications and clients to interface with the blockchain

How Does Hyperledger Work?

- The membership service involved in the network validates the contract.
- The concerned two-peer has to produce a result and then sent it to the consensus cloud.
- The generated result from both the peer has to be the same in order to validate the contract.
- Once it is validated, then the transaction will happen between the affiliated peers and their ledger will be updated.
- When a business requires confidentiality and a private network for its transaction to happen without doing that in a single network, a hyperledger paves the way.

Example

- Suppose Alice decides to send the product to Bob on a hyperledger-based network.
- She looks at her app to locate her address of Bob on the network.
- The app in return queries the membership service and validates Bob’s membership.
- The hyperledger will connect both parties directly for the transaction affiliated with the deal.
- Both parties will generate the result, which has to be the same for them to get validated.
- The result will be sent to the consensus cloud to be ordered and verified.
- Once the result is validated, Bob receives the product and the transaction is committed to the ledger.

Alice Sends

Alice decides to send the product to Bob.

App Query

She looks at her app to locate the address of Bob on the network.

Membership Service

The app in return queries the membership service and validates Bob's membership.

Hyperledger

The hyperledger will connect both parties directly for the transaction affiliated with the deal.

Verification

Both the parties will generate the result, and it will be sent to the consensus cloud.

Bob Recieves

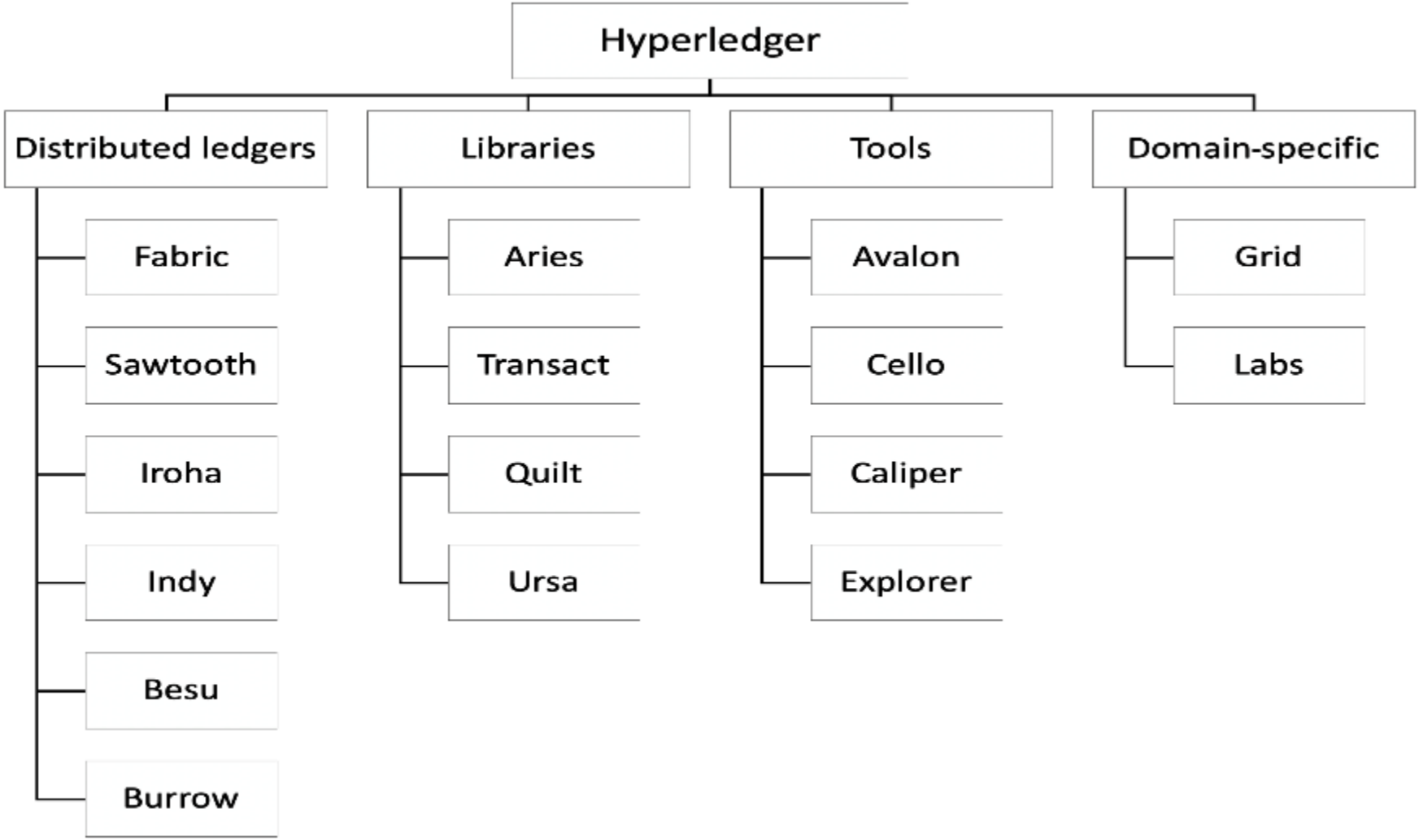
After the result is validated, Bob receives the product and the transaction is committed to the ledger.

HYPER LEDGER PROJECTS

Projects under Hyperledger

There are four categories of projects under Hyperledger. Under each category, there are multiple projects. The categories are:

- Distributed ledgers
- Libraries
- Tools
- Domain-specific

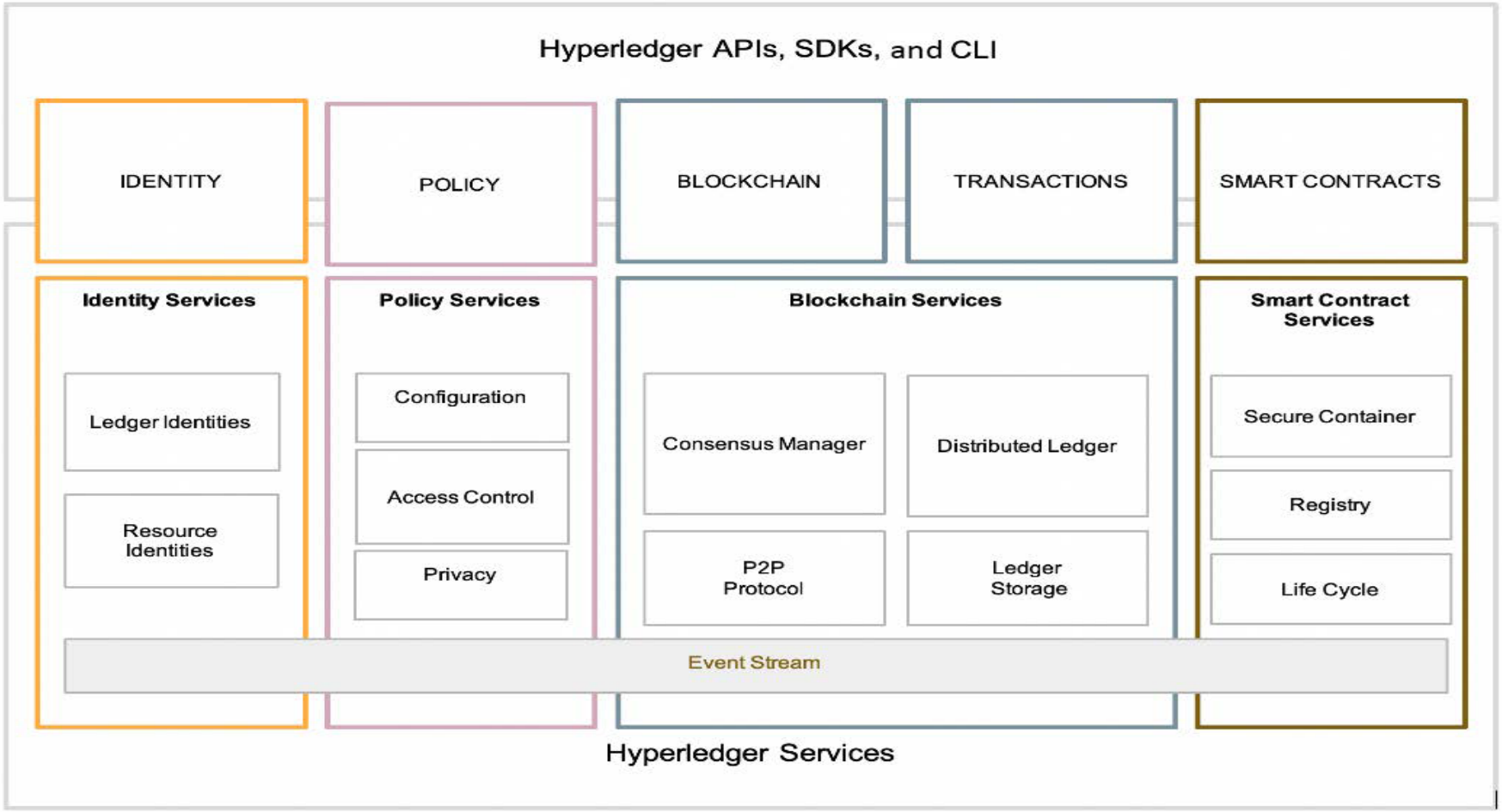

Hyperledger
Distributed ledgers
Fabric
Sawtooth
Iroha
Indy
Besu
Burrow
Libraries
Aries
Transact
Quilt
Ursa
Tools
Avalon
Cello
Caliper
Explorer
Domain-specific
Grid
Labs

Hyperledger Projects

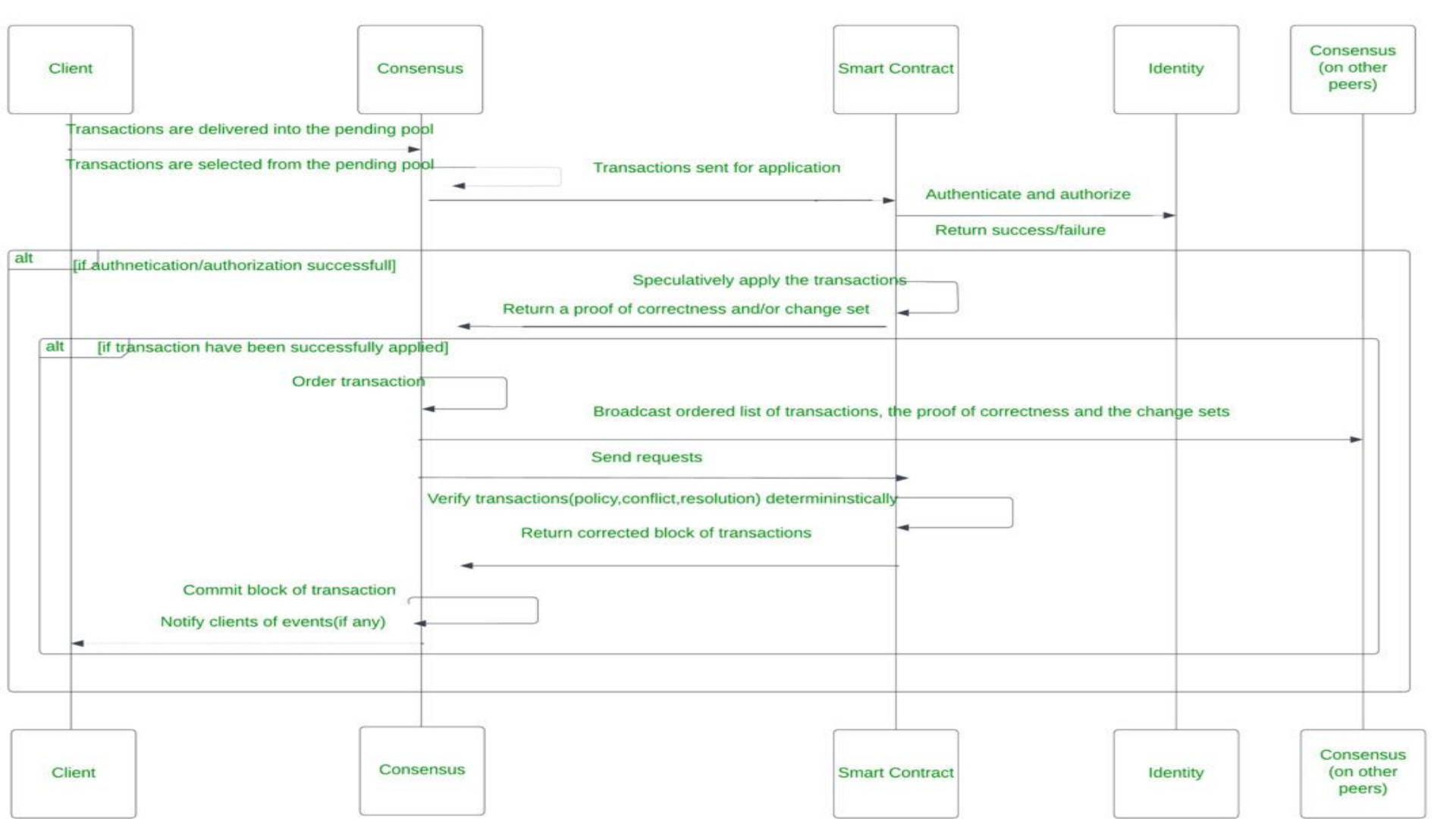
1. **Hyperledger Fabric:** Hyperledger Fabric is intended as a foundation for developing applications and solutions with modular architecture. It provides many benefits like permissioned networks, confidential transactions, etc.
2. **Hyperledger Sawtooth:** It is an open-source project and used as an enterprise-level blockchain system used for creating and operating distributed ledger applications. Hyperledger sawtooth supports a variety of consensus algorithms like PBFT, and PoET.
3. **Hyperledger Indy:** It is a project that is made for decentralized identity. It offers lots of libraries, tools, and reusable components for creating decentralized identities.
4. **Hyperledger Iroha:** It is a blockchain platform designed for infrastructure projects that need distributed ledger technology. It is used to build identity management platforms like national IDs. It can integrate with Linux, macOS, and Windows platforms.
5. **Hyperledger Burrow:** It is a framework for executing smart contracts in permissioned blockchains. The goal of Hyperledger burrow is to facilitate cross-industry applications for smart contracts. It is built around the BFT consensus algorithm.

6. **Hyperledger Caliper:** It is a blockchain benchmark tool that allows users to measure the performance of a blockchain implementation with a set of predefined use cases. It will produce reports containing a number of performance indicators to serve as a reference when using the blockchain solutions like Hyperledger Burrow, Hyperledger Fabric, Hyperledger Iroha, and so on.
7. **Hyperledger Cello:** It serves as an operational dashboard for Blockchain that reduces the effort required for creating, managing, and using blockchains. It provides an operational console for managing blockchain efficiently.
8. **Hyperledger Explorer:** It is a user-friendly web application tool that is used to view, invoke, deploy, or query blocks, associated data, and network information stored in the ledger. It is regarded as an easy way that allows users to view the necessary network information of the blockchain.
9. **Hyperledger Besu:** It is an Ethereum client designed to be enterprise-friendly for both public and private blockchain network use cases. It offers many useful features like EVM, several proof-of-authority protocols, a privacy transaction manager to ensure the privacy of transactions, etc.

Hyperledger reference architecture



Hyperledger Architecture


Client
Consensus
Smart Contract
Identity
Consensus (on other peers)
Transactions are delivered into the pending pool
Transactions are selected from the pending pool
Transactions sent for application
Authenticate and authorize
Return success/failure
alt
[if authnetication/authorization successfull]
Speculatively apply the transactions
Return a proof of correctness and/or change set
alt
[if transaction have been successfully applied]
Order transaction
Broadcast ordered list of transactions, the proof of correctness and the change sets
Send requests
Verify transactions(policy,conflict,resolution) determininstically
Return corrected block of transactions
Commit block of transaction
Notify clients of events(if any)
Client
Consensus
Smart Contract
Identity
Consensus (on other peers)

In the preceding diagram, starting from the left, we see that we have five top-level components that provide various services. The first is identity, which provides authorization, identification, and authentication services under membership services. Then, we have the policy component, which provides policy services.

After this, we see blockchain and transactions, which consist of the Consensus Manager, Distributed Ledger, the network P2P Protocol, and Ledger Storage services. The consensus manager ensures that the ledger is updateable only through consensus among the participants of the blockchain network.

Finally, we have the smart contracts layer, which provides chaincode services in Hyperledger and makes use of Secure Container technology to host smart contracts. Chaincode can be considered as the Hyperledger equivalent of smart contracts.

Hyperledger contains various elements, as described here

- Consensus: These services are responsible for facilitating the agreement process between the participants on the blockchain network. Consensus is required to make sure that the order and state of transactions are validated and agreed upon in the blockchain network.
- Smart contracts: These services are responsible for implementing business logic as per the requirements of the users. Transactions are processed based on the logic defined in the smart contracts that reside on the blockchain.
- Communication: This component is responsible for message transmission and exchange between the nodes on the blockchain network

Security and crypto: These services are responsible for providing the capability to allow various cryptographic algorithms or modules to provide privacy, confidentiality, and non-repudiation services.

• Data store: This component provides an ability to use different data stores for storing the state of the ledger. This means that data stores are also pluggable, allowing the usage of any database backend, such as couchdb or goleveldb.

• Policy services: This set of services provides the ability to manage the different policies required for the blockchain network. This includes endorsement policy and consensus policy.

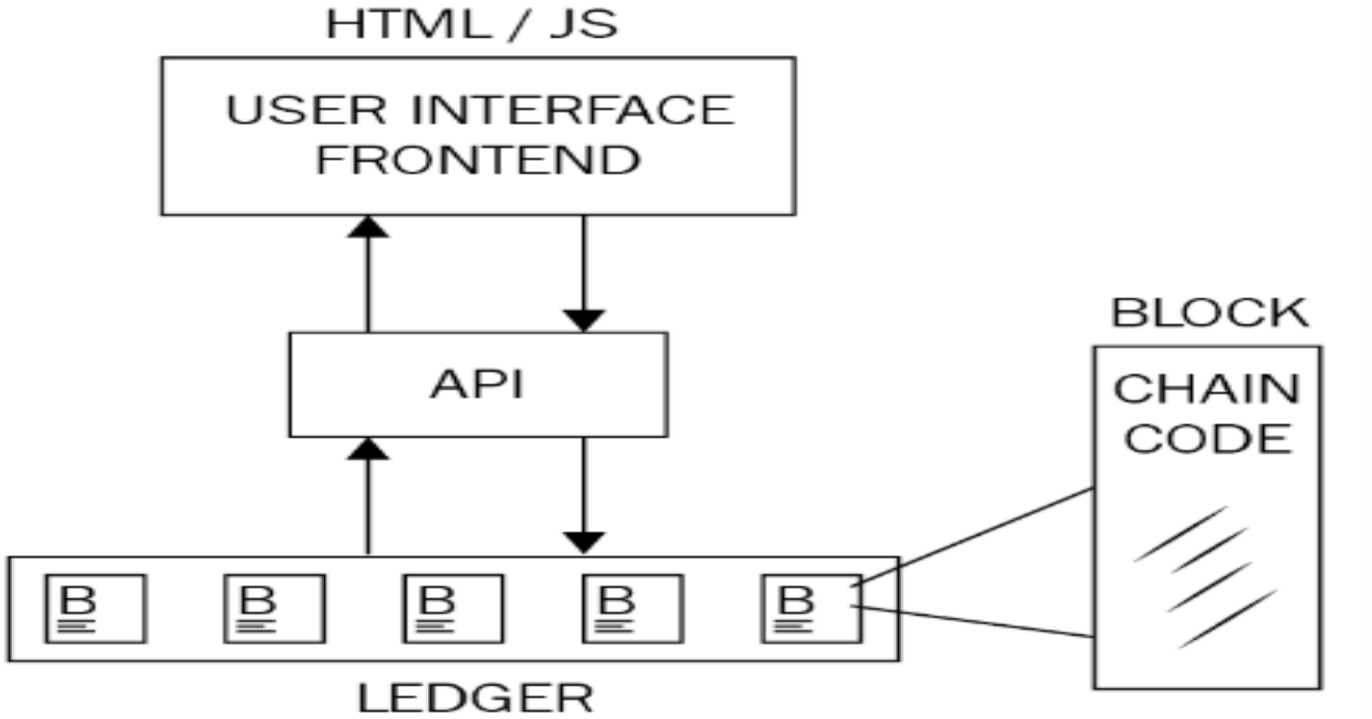
• APIs and SDKs: This component allows clients and applications to interact with the blockchain. An SDK is used to provide mechanisms to deploy and execute chaincode, query blocks, and monitor events on the blockchain.

Fabric

Applications on blockchain

A typical application on Fabric is simply composed of a user interface, usually written in JavaScript/HTML, that interacts with the backend chaincode (smart contract) stored on the ledger via an API layer:

Figure 17.4: A typical Fabric application



Hyperledger provides various APIs and command-line interfaces to enable interaction with the ledger.

These APIs include interfaces for identity, transactions, chaincode, ledger, network, storage, and events.

Hyperledger Fabric

- Hyperledger Fabric is led by IBM. Hyperledger Fabric is a plug-and-play blockchain technology implementation with a configurable degree of permissions. The Linux Foundation manages this private and confidential blockchain framework.
- Hyperledger Fabric is a scalable, flexible architecture-based platform for distributed ledger applications. It is designed to allow for pluggable versions of various aspects and to manage the financial ecosystem’s complexities and intricacies

. Hyperledger Sawtooth

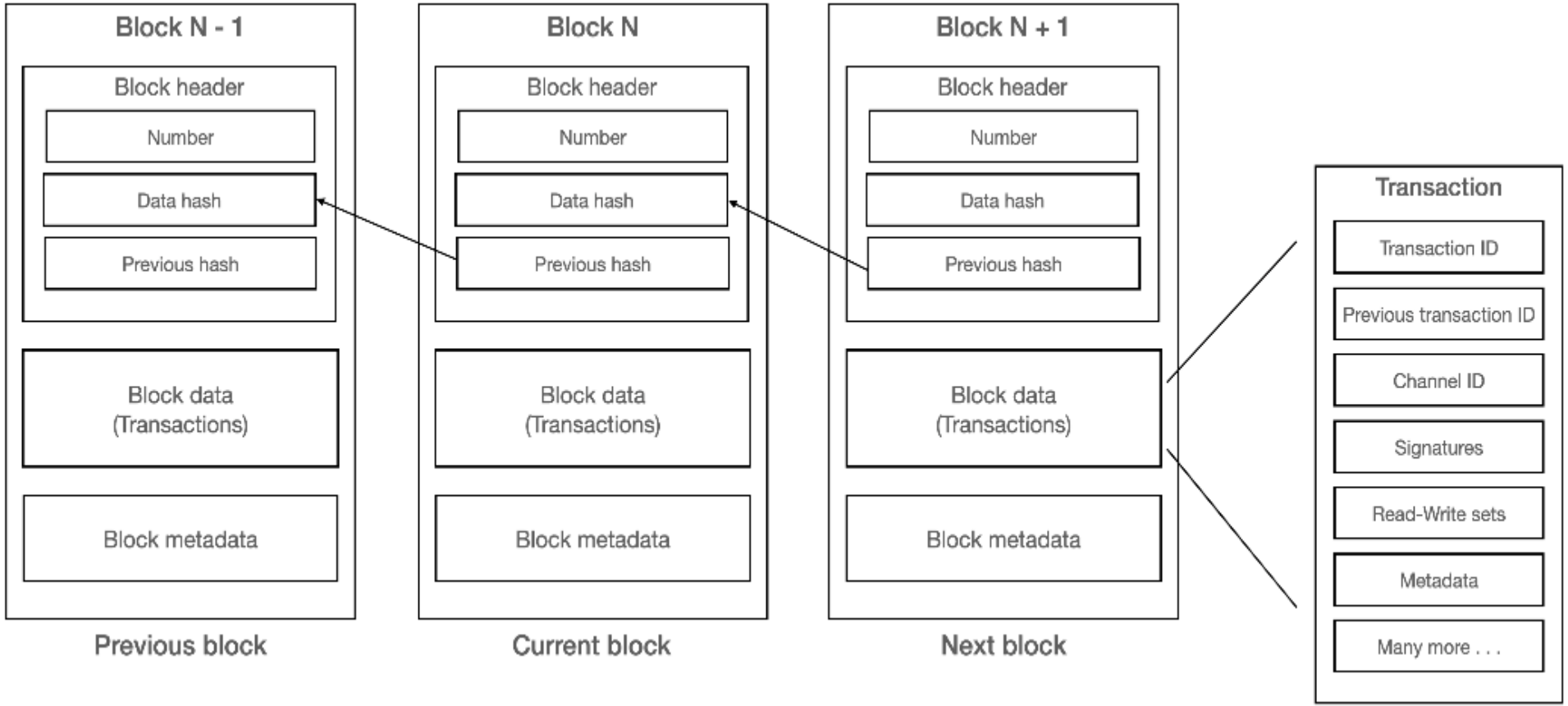
- Hyperledger Sawtooth is a flexible platform for developing, implementing, and running distributed ledgers, and it features a revolutionary consensus mechanism known as Proof of Elapsed Time. This consensus is suitable for large distributed validator groups while consuming few resources.
- Intel is working on this project. Hyperledger Sawtooth is an open-source industrial blockchain-as-a-service platform that can run customized smart contracts without requiring knowledge of the core system’s underlying design

corda

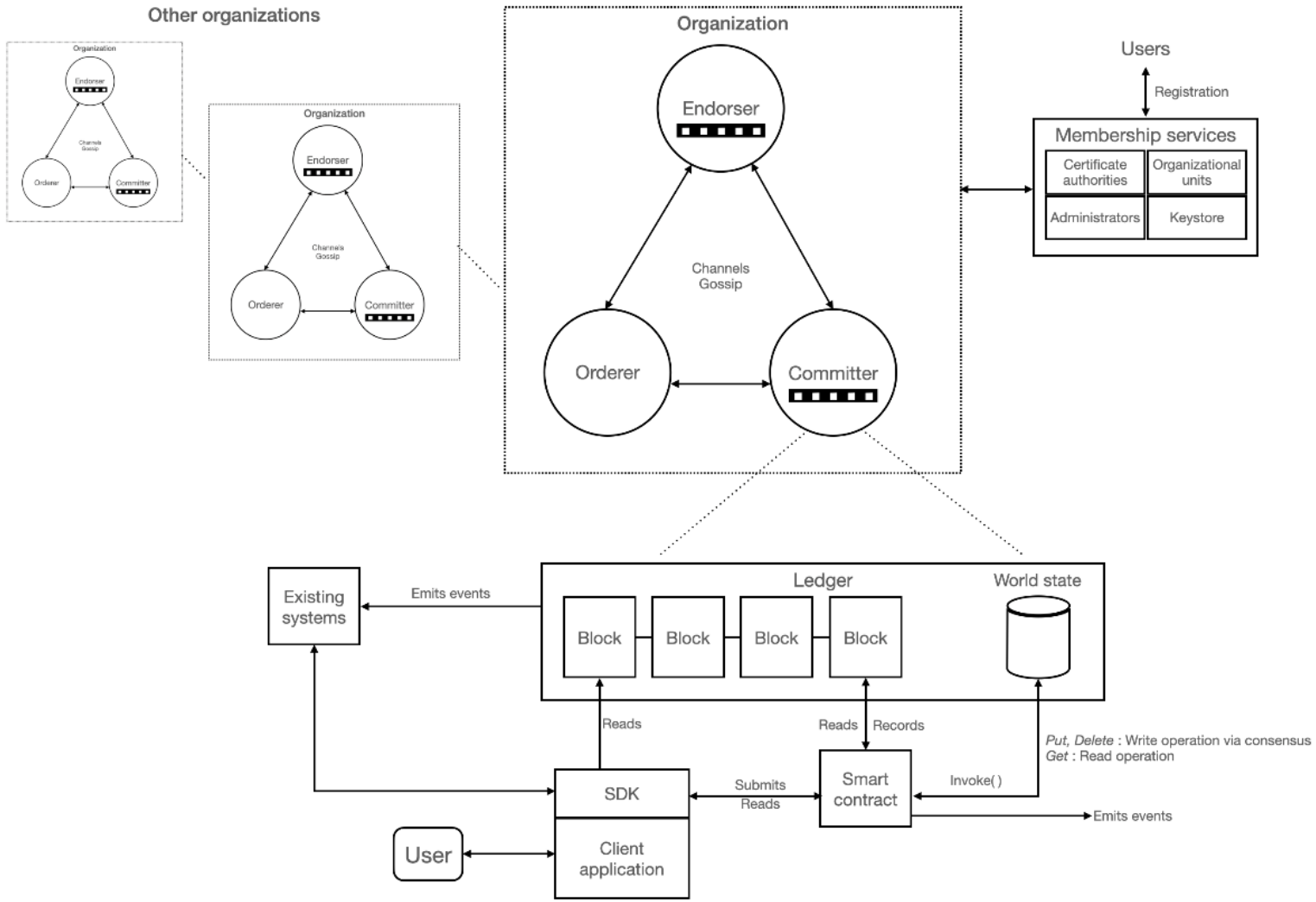
Hyper Ledger Fabric

- **Hyperledger Fabric, or Fabric** for short, is the contribution made initially by IBM and Digital Assets to the Hyperledger project.
- This contribution aims to enable a *modular, open, and flexible* approach toward building blockchain networks.
- Various functions in the fabric are *pluggable*, and it also allows the use of any language to develop smart contracts. This functionality is possible because it is based on container technology *(Docker), which can host any language.*
- **Chaincode** is sandboxed in a secure container, which includes a secure operating system, the chaincode language, runtime environment, and SDKs for Go, Java, and Node.js.

Blockchain and transaction structure



- **Block header** consists of three fields, namely **Number, Data hash, and Previous hash.**
- **Block data** contains an ordered list of transactions**. A Transaction** is made up of multiple fields such as **Transaction ID, Previous transaction ID, Channel ID, Signatures, Read-Write sets, Metadata,** and many more.
- **Block metadata** mainly consists of the creator identity, time of creation, and relevant signatures.


Other organizations
Organization
Endorser
Channels Gossip
Orderer
Committer
Organization
Endorser
Channels Gossip
Orderer
Committer
Organization
Endorser
Channels Gossip
Orderer
Committer
Users
Registration
Membership services
Certificate authorities
Organizational units
Administrators
Keystore
Existing systems
Emits events
Ledger
World state
Block
Block
Block
Block
Reads
Reads
Records
Put, Delete : Write operation via consensus
Get : Read operation
SDK
Submits
Reads
Smart contract
Invoke( )
Emits events
User
Client application

A high-level overview of a Hyperledger Fabric network

- The preceding diagram shows that peers (shown center-top) communicate with each other and each node has a copy of blockchain.
- In the top-right corner, the membership services are shown, which validate and authenticate peers on the network by using a CA.
- At the bottom of the diagram, a magnified view of blockchain is shown whereby existing systems can produce events for the blockchain and can also listen for the blockchain events, which can then optionally trigger an action.
- At the bottom right-hand side, a user's interaction with the application is shown. The application in turn interacts with the smart contract via the Invoke()method, and smart contracts can query or update the state of the blockchain.

The application model

- Any blockchain application for Hyperledger Fabric follows the **MVC-B** architecture. This is based on the popular MVC design pattern. Components in this model are **View, Control, Model** (data model), and **Blockchain:**
- **View logic:** This is concerned with the user interface. It can be a desktop, web application, or mobile frontend.
- **Control logic:** This is the orchestrator between the user interface, data model, and APIs.
- **Data model**: This model is used to manage the off-chain data.
- **Blockchain logic:** This is used to manage the blockchain via the controller and the data model via transactions.

Consensus in Hyperledger Fabric

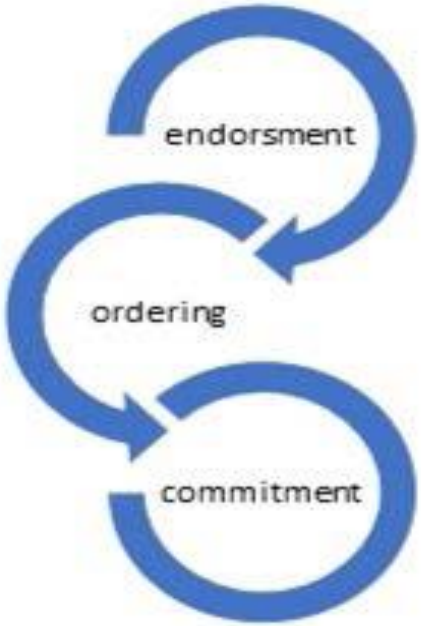
The consensus mechanism in Hyperledger Fabric consists of three steps:

- **Transaction endorsement:** This process endorses the transactions by simulating the transaction execution process.

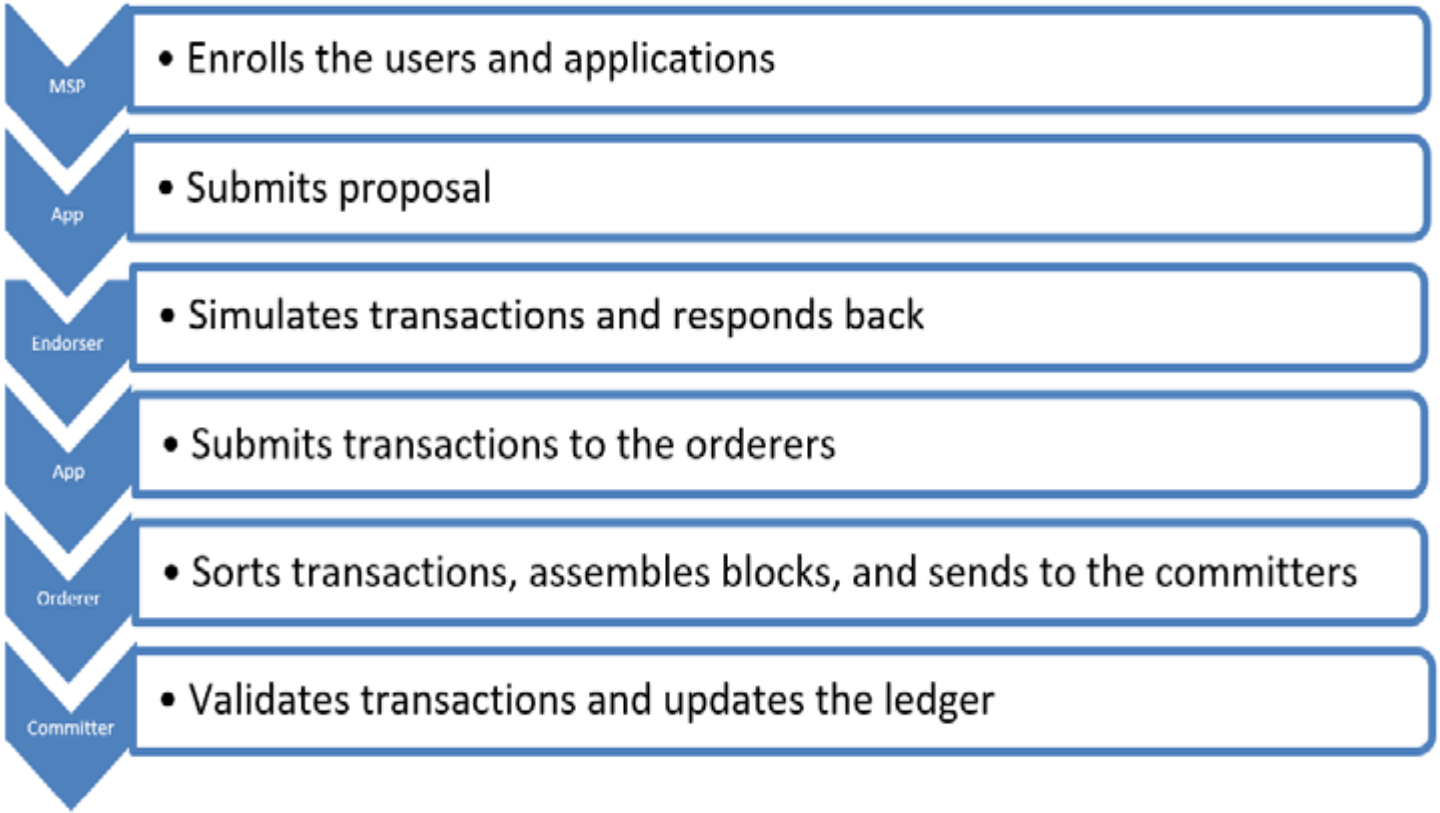
- **Ordering:** This is a service provided by the cluster of orderers, which takes endorsed transactions and decides on a sequence in which the transactions will be written to the ledger.

- **Validation and commitment:** This process is executed by committing peers, which first validate the transactions received from the orderers and then commit that transaction to the ledger.

The consensus flow



The transaction lifecycle in Hyperledger Fabric



The steps are described in detail

1. Transaction proposal by clients and sent to endorsing peers on the distributed ledger network. All clients need to be enrolled via membership services before they can propose transactions.

2. The transaction is simulated by endorsers, which generates a **read-write (RW) set.** This is achieved by executing the chaincode, but instead of updating the ledger, only an RW set depicting any reads or updates to the ledger is created.
3. The endorsed transaction is sent back to the application.

4. The endorsed transactions and RW sets are submitted to the ordering service by the application.

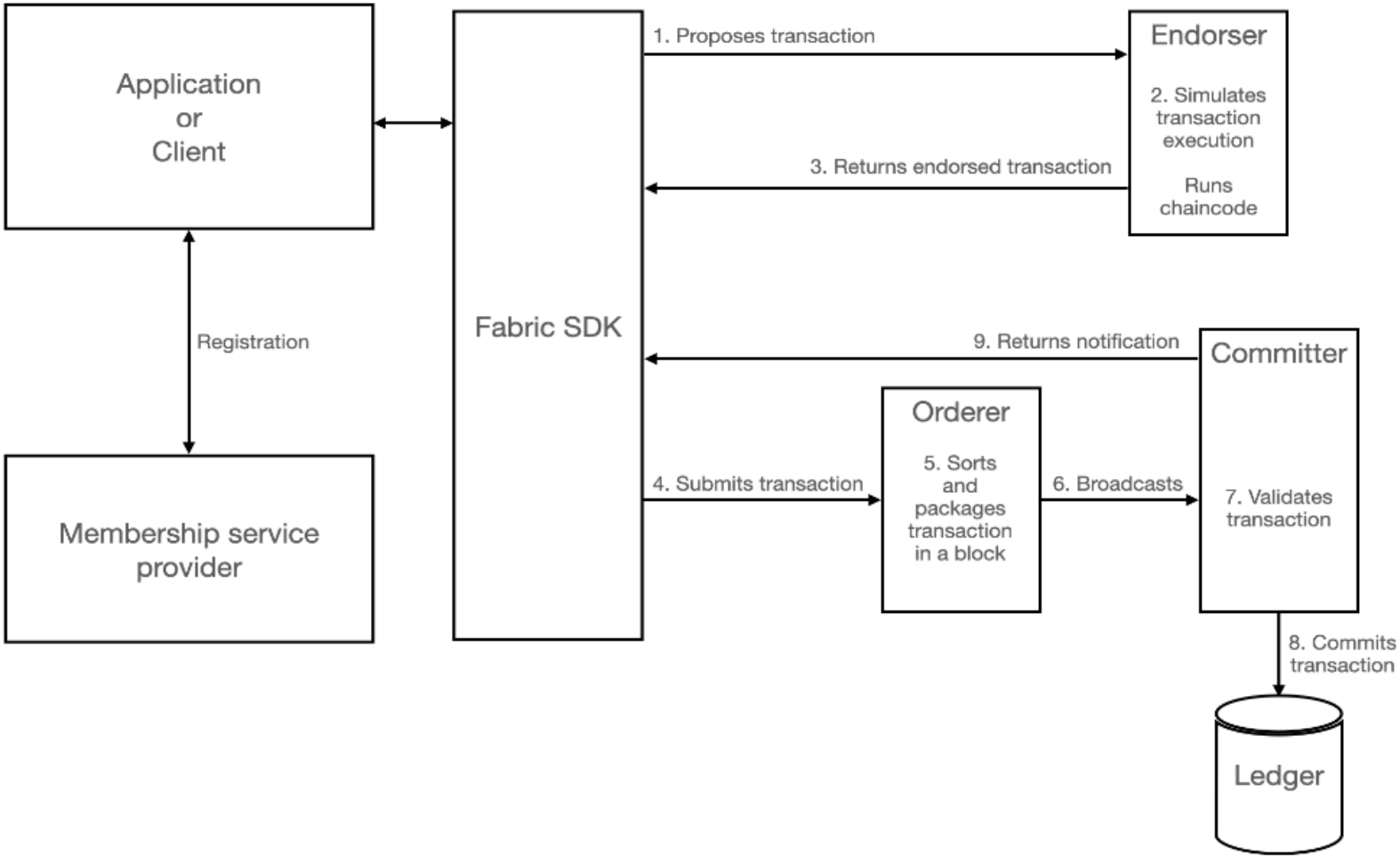
5. The ordering service assembles all endorsed transactions and RW sets in order into a block and sorts them by channel ID.

6. The ordering service broadcasts the assembled block to all committing peers.

7. Committing peers validate the transactions.

8. Committing peers update the ledger.

9. Finally, notification of success or failure of the transaction by committing peers is sent back to the clients/applications.


Application
or
Client
Registration
Membership service
provider
Fabric SDK
1. Proposes transaction
Endorser
2. Simulates
transaction
execution
Runs
chaincode
3. Returns endorsed transaction
9. Returns notification
Committer
Orderer
5. Sorts
and
packages
transaction
in a block
4. Submits transaction
6. Broadcasts
7. Validates
transaction
8. Commits
transaction
Ledger

Hyperledger Sawtooth

- Sawtooth is an enterprise-grade distributed ledger that can run in both permissioned and nonpermissioned modes.

- Sawtooth has several new / novel features:

1. **Modular design:** The design of Sawtooth can be viewed as a *layered architecture* where transaction processors manage the application business logic and, on another layer, validators handle the verification and consensus on transactions
2. **Parallel transaction execution:** parallel scheduler *splits the transactions into parallel flows*, which results in increased performance due to the parallel execution of transactions.

3. **Global state agreement:** Sawtooth is a *cryptographically verifiable distributed ledger* with global state agreement.
4. **Dynamic and pluggable consensus algorithms:**. A very interesting feature of Sawtooth is that *consensus mechanisms can be changed while a blockchain is running*.
5. **Multi-language support:** Blockchain applications in Sawtooth can be *written using various different languages*. Sawtooth also includes SDKs for different languages, such as JavaScript, Python, and Go.
6. **Enhanced event mechanism:** Sawtooth supports the *creation and broadcasting of events where applications can subscribe* to blockchain-related events or application-specific events.
7. **On-chain governance:** Sawtooth allows the *on-chain storage, management, and application* of configuration settings and permissions, which makes it easier for all clients to refer to a *single source of configuration.*

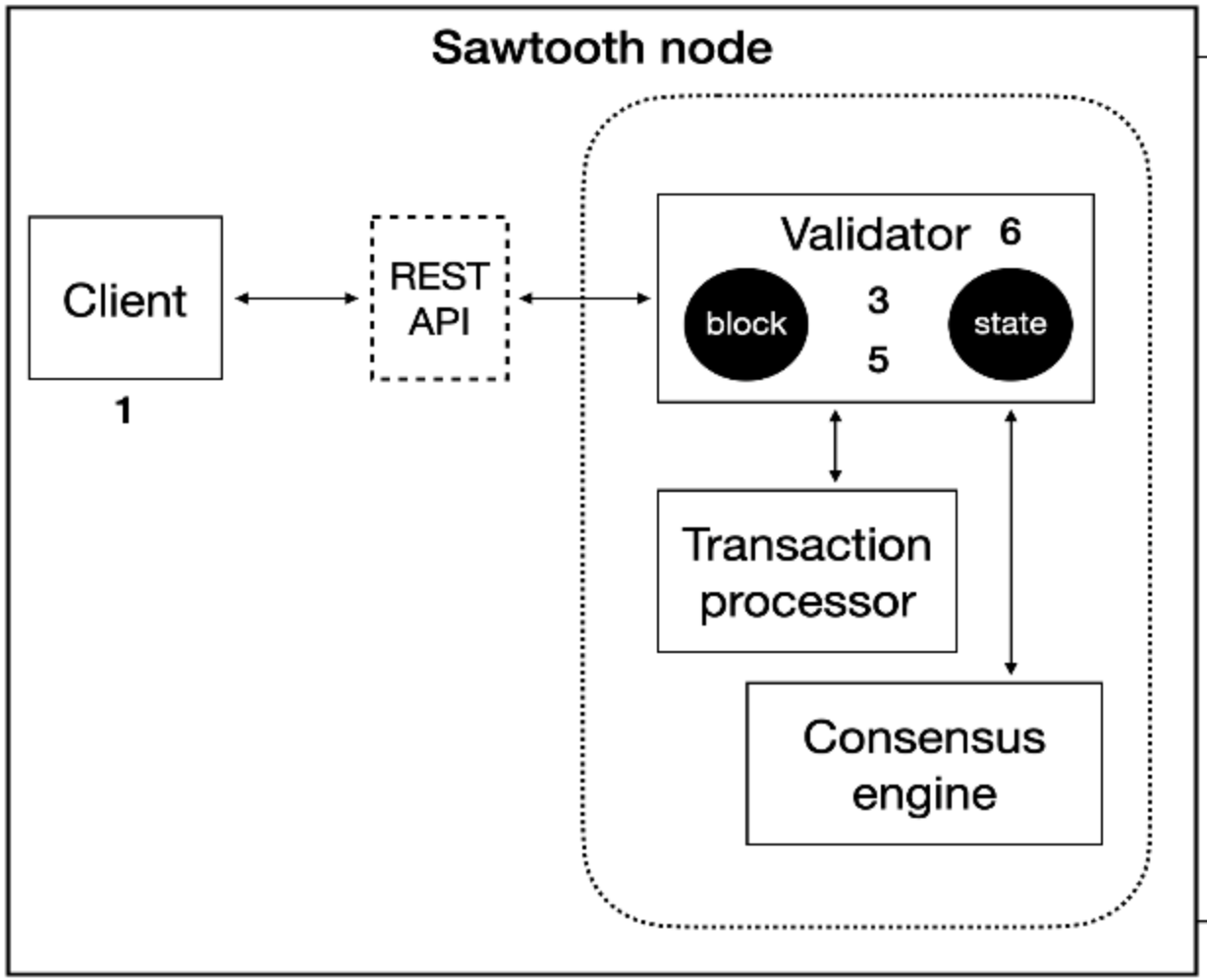
8. **Interoperability:** Sawtooth *supports integration with other blockchain technologies*. For example, it introduced a project named seth, which allows EVM-based smart contracts to be deployed on Sawtooth.

9. **Consensus in Sawtooth:** Sawtooth supports several types of algorithms. A novel mechanism introduced with Sawtooth is called **PoET.**
PoET is a *trusted, executed environment-based lottery function* that elects a leader randomly based on the time a node has waited for a block proposal.

Transaction lifecycle

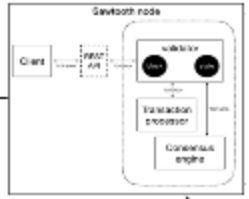
Transaction is a function that, when executed, results in changing the state of the blockchain.

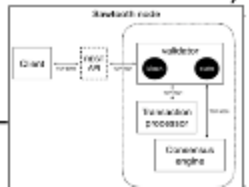
1. A *client submits a transaction* to a validator. (by utilizing the REST API)
2. After the transaction is submitted, it is *propagated across the validator network*.
3. One of the *validators is elected as a leader*, which then creates a candidate block and publishes it on the network.
4. This *candidate block is propagated* across the entire validator network.
5. When validators receive this *block*, it is *validated*. In addition, *transaction processors validate and execute all transactions* present in the candidate block.
6. Once the *block* is validated and verified, it *is written in its respective local storage and the state is updated* accordingly.


Sawtooth node
Client
1
REST API
Validator
6
block
3
5
state
Transaction processor
Consensus engine

2

4





Components

Validator

- The validator is the component that is responsible for *validating batches of transactions*.

- It combines transactions into blocks and *maintains consensus* on the Sawtooth blockchain network.

- In addition, validators also perform *coordination functions between clients, transaction processors, and other validators* on the network.

- Validators *communicat*e using the *gossip protocol*.

REST API

- REST API is responsible for providing a *communication interface between clients and validators*.

Client

- The client is responsible for implementing the client logic.

- It is responsible for the following functions:
  - Transaction creation and payload generation
  - The handling of events in the blockchain and application, such as error handling, notifications, and blockchain state updates
  - Displaying data to the users

State

- The state in a blockchain is represented as a *log of all changes made to the blockchain database* since the start of the blockchain network or genesis block.
- The **global state,** on the other hand, represents the *agreement on the state of the blockchain between all nodes* on the network.

Transaction processors

- Transaction processors are responsible for *validating transactions and updating the state*.

- There is always a *transaction family associated with a transaction processor*.

Transaction processors are responsible for:

- Transaction validation
- Block creation
- Business logic or transaction rules

Transaction families (sawtooth applications)

- A transaction family is *created by decomposing the logic layer into a set of rules and a composition layer* for a specific domain.
- The key idea is that business logic is composed within transaction families, which provides a more *secure and powerful way to build smart contracts*.
- Transaction families contain the domain-specific rules and another layer that allows the creation of transactions for that domain.
- Another way of looking at it is that transaction families are a combination of a data model and a transaction language that implements a logic layer for a specific domain.
- The data model represents the current state of the blockchain (ledger), whereas the transaction language modifies the state of the ledger.

Transaction families

- A **traditional smart contract paradigm** provides a solution that is based on a *general-purpose instruction set for all domains*.

  For example, in the case of Ethereum, a set of opcodes has been developed for the EVM that can be used to build smart contracts to address any type of requirements for any industry.
- It is *not very secure* as it provides a single interface into the ledger.

- Vulnerabilities that were found and exploited recently by hackers **DAO** hack and **Denial of Services (DoS)**

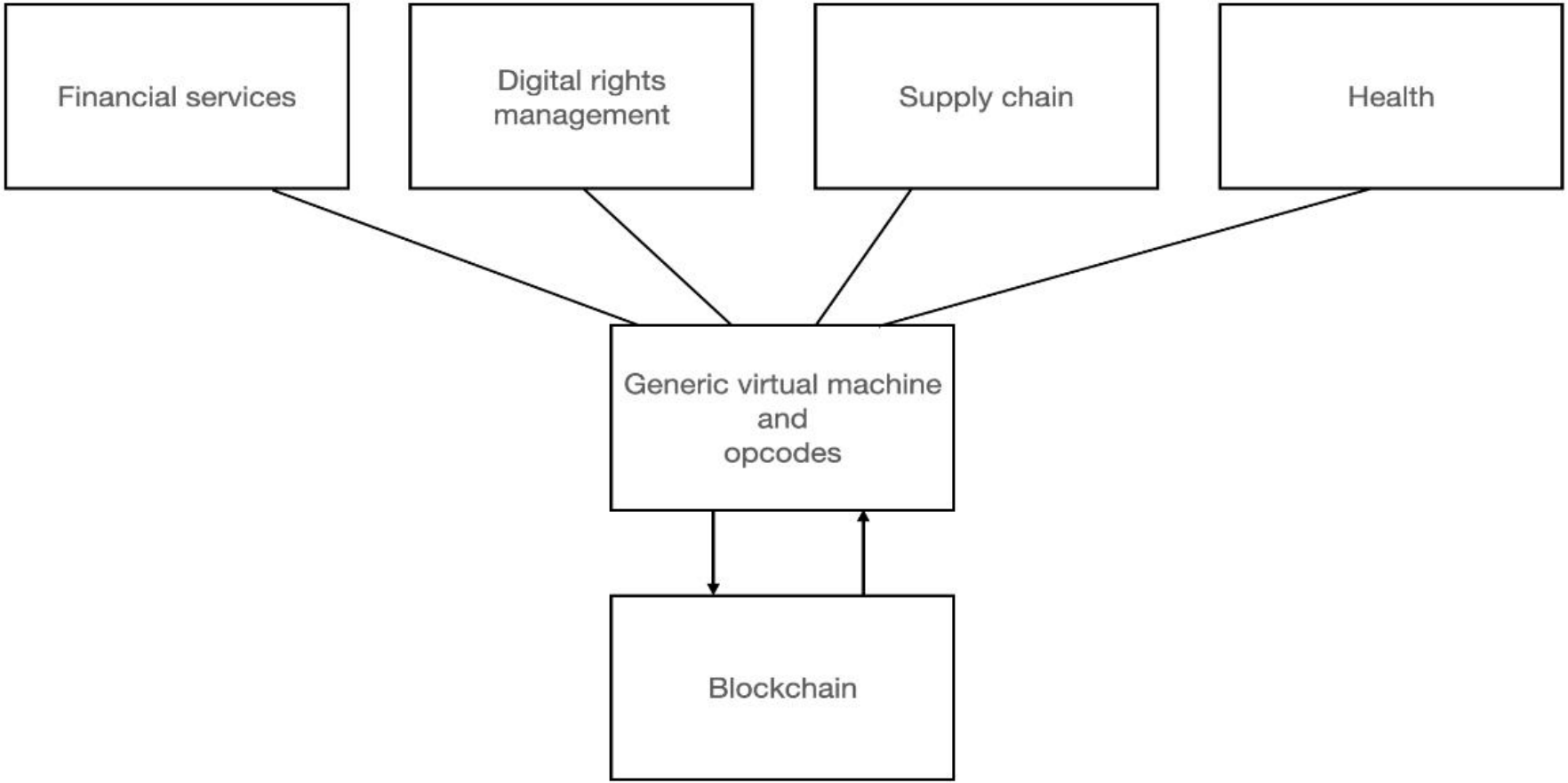
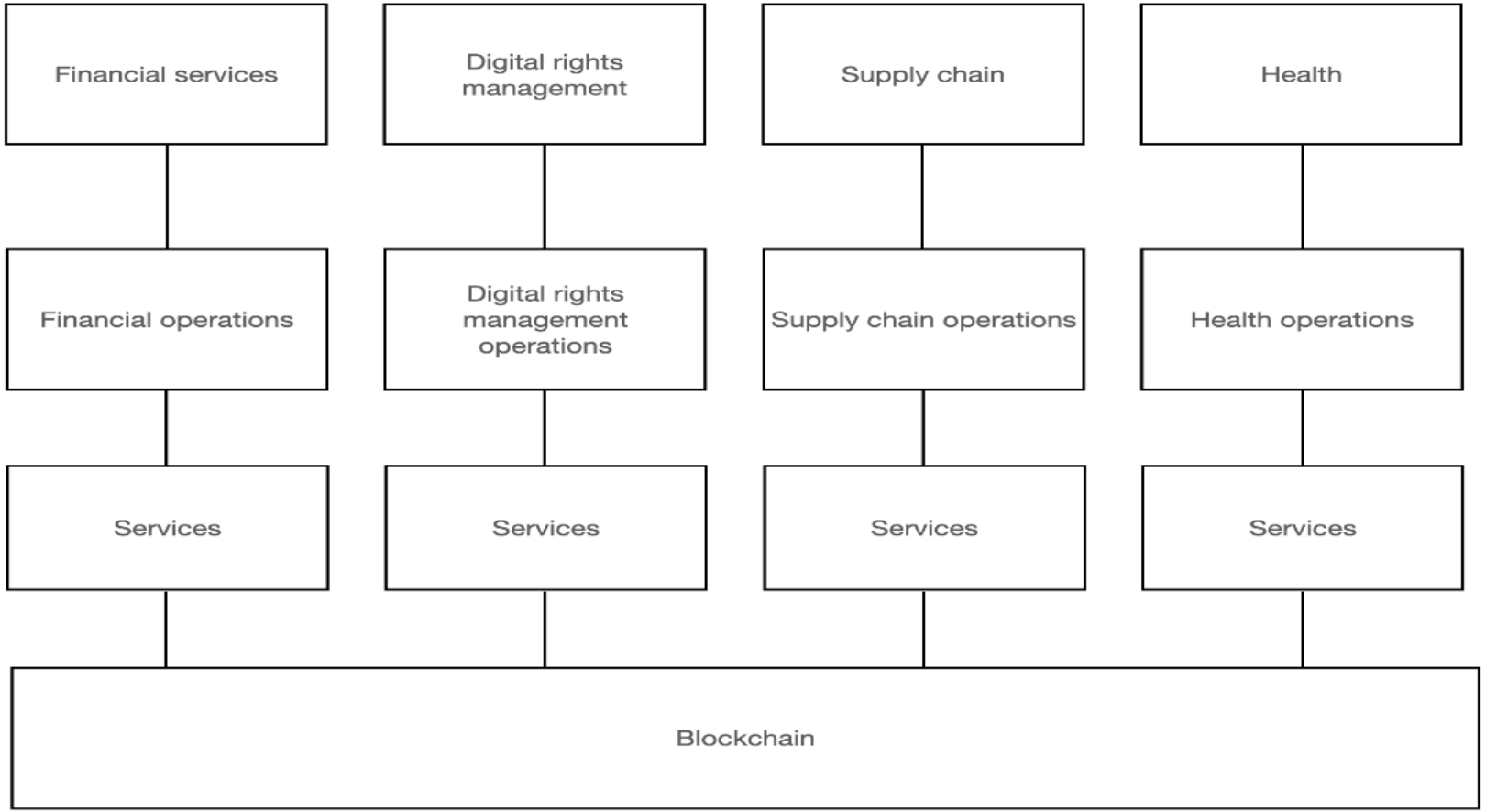


Fig: The traditional smart contract paradigm


Financial services
Digital rights management
Supply chain
Health
Financial operations
Digital rights management operations
Supply chain operations
Health operations
Services
Services
Services
Services
Blockchain

- The following diagram represents this model, where each specific domain, such as **Financial services, Digital rights management (DRM), Supply chain, and the Health industry,** has its own logic layer comprised of operations and services specific to that domain.

- This makes the logic layer both restrictive and powerful at the same time.

- Transaction families ensure that operations related to only the required domain are present in the control logic, thus removing the possibility of executing needless, arbitrary, and potentially harmful operations

CORDA

- Corda is not a blockchain by definition because it *does not use blocks for batching transactions*.

- Still, it is a *distributed ledger*, and *provides all benefits that a blockchain can*.

- It was developed initially for the *financial industry* to solve issues arising from each organization managing its own local ledgers.

- Corda has two implementations, **Corda enterprise** and **Corda open source.** Both are *interoperable and compatible* with one another, and have the *same features*.

- **Corda enterprise** is a commercial version of the Corda platform targeting *business requirements* such as privacy, security, and performance.

- Also, it includes *Corda firewall, high availability nodes and notaries*, and support for *hardware security modules*.

- The **Corda source code** is available at *https://github.com/corda/corda*. It is written in a language called **Kotlin,** which is a statically typed language targeting the **Java Virtual Machine**(**JVM).**

Architecture

The main components of the Corda platform include the **Corda network, state objects (**contract code and legal prose**), transactions, consensus,** and **flow**s.

➢ **1) Corda network**

- The Corda network is defined as a fully connected graph. It is a *permissioned network* that provides *direct P2P communication* on a *"need to know"* basis.

- The communication occurs directly on a point to point basis between interested parties.

- Peers or nodes communicate using the *AMPQ serialization protocol*.

- A service called *network map service* is responsible for publishing a list of peers.

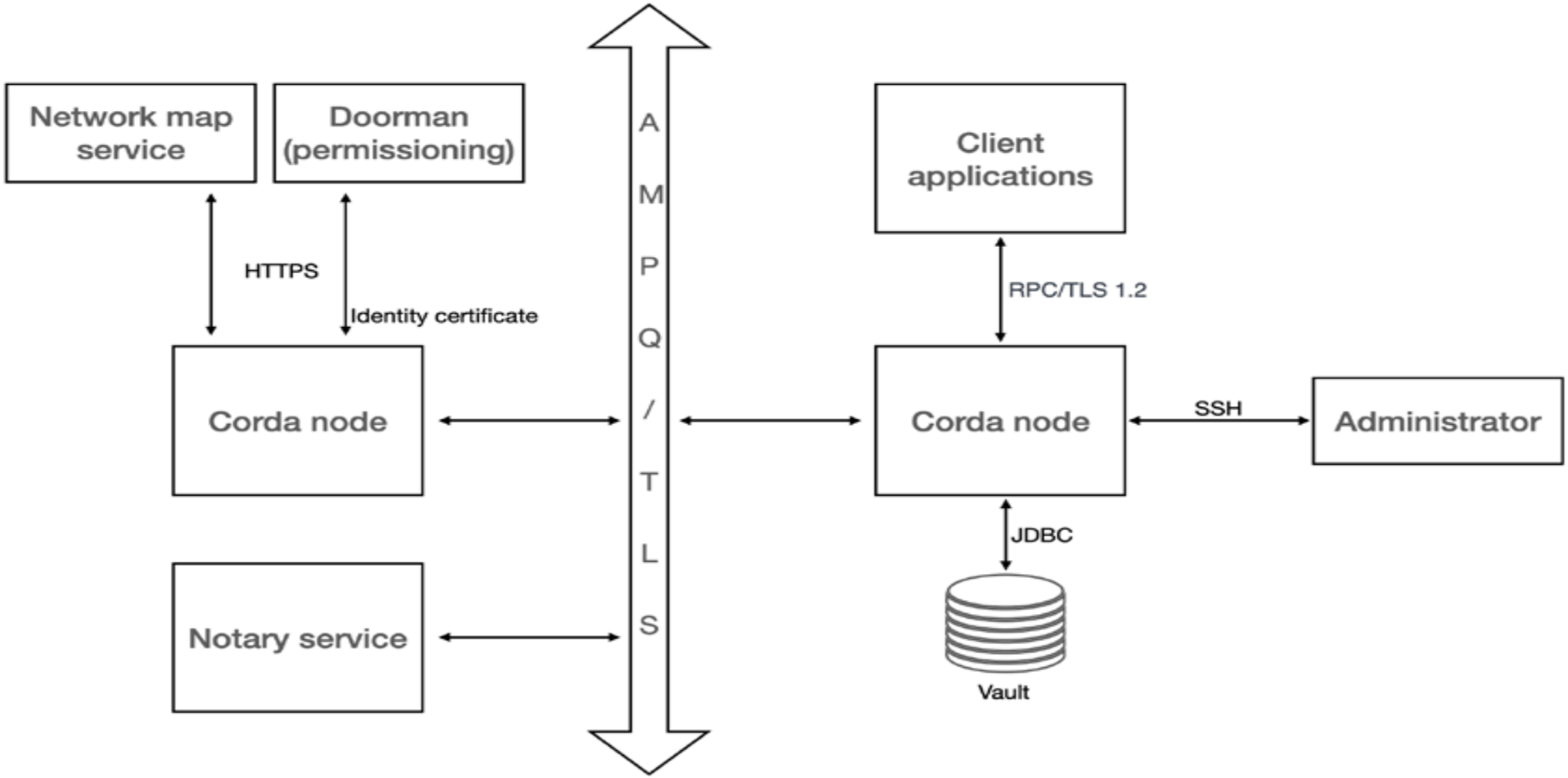


Figure : Corda high-level network architecture

2) State objects

- State objects represent the *smallest unit of data that constitute a financial agreement.*

- State objects *refer* to the *contract code and legal prose*.

➢ **Legal prose** is optional and provides legal binding to the contract.
➢ **contract code** is required to provide a state transition mechanism for the node, according to the business logic defined in the contract code.

- State objects contain a *data structure* that represents the *current state* of the object. A state object can be either *current (live) or historic (no longer valid).*

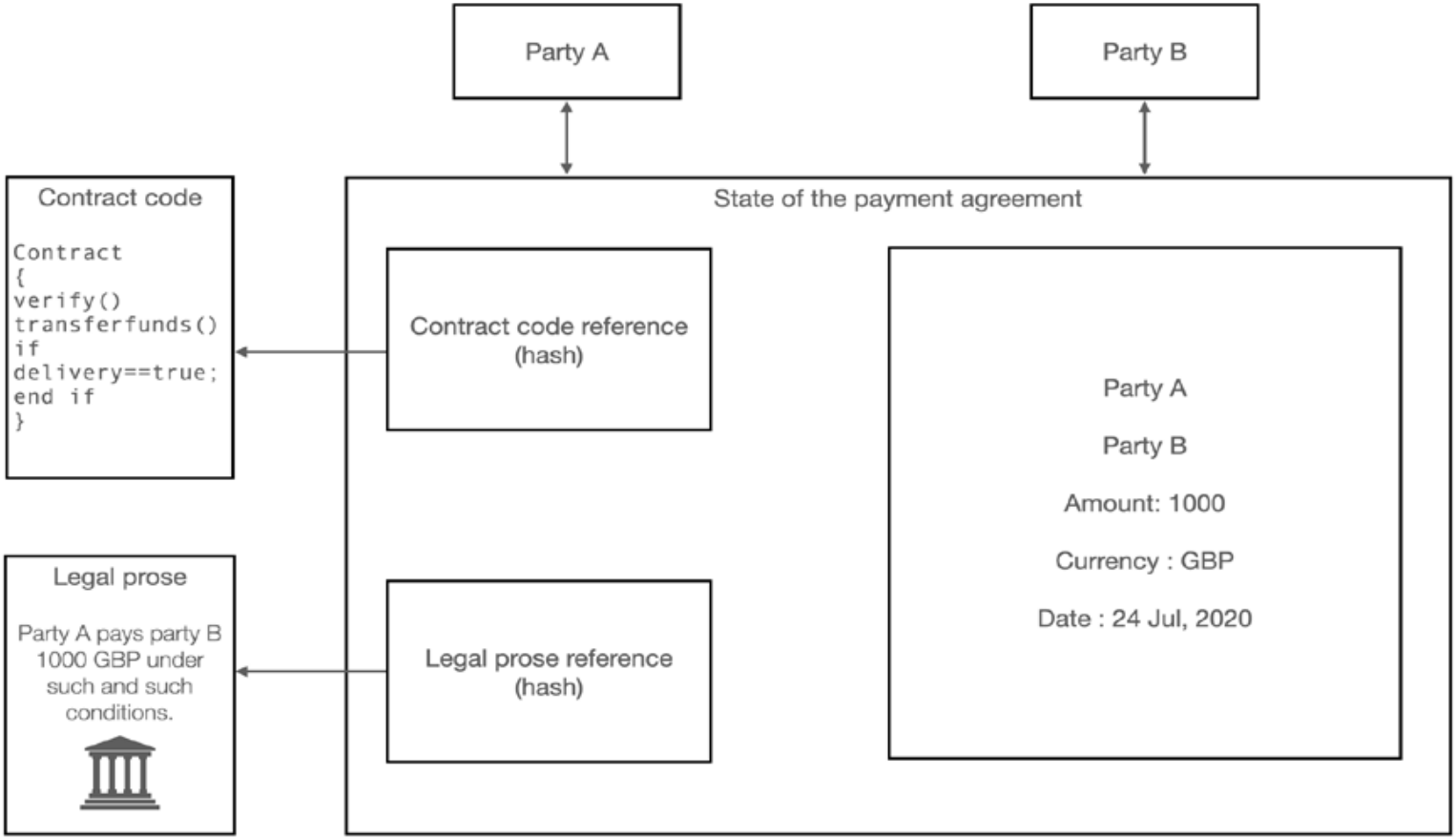


Figure 20.6: An example state object

- For example, in the following diagram, a state object represents the current state of the object.

- In this case, it is a simple mock agreement between **Party A and Party B** where Party A has paid Party B **1,000 GBP**(Great Britain Pound – form of currency in UK). This represents the current state of the object;

- however, the referred contract code can change the state via transactions.

- State objects can be thought of as a state machine, which is consumed by transactions to create updated state objects:

3)Transactions

Corda uses a *Bitcoin-style UTXO-based model* for its transaction processing.

- can have single, multiple, or no inputs, and single or multiple outputs.
- All transactions are digitally signed.
- Corda has no concept of mining because it does not use blocks to arrange transactions in a blockchain. Instead, notary services are used to provide a temporal ordering of transactions.
- In Corda, new transaction types can be developed using JVM bytecode, which makes it very flexible and powerful.

4) Consensus

- *The general idea is that the transactions are evaluated for their uniqueness* by the notary service and, if they are unique (that is, unique transaction inputs), they are *signed by consensus services as valid*.

- Various consensus *algorithms* like **PBFT or Raft** can be used by notaries to reach consensus.

- There are *two types of consensus* in Corda:
  - validity consensus
  - uniqueness consensus.

4.1)Validity consensus

This ensures that:

- The contract of *every input and output state accepts the proposed transaction.*

- The transaction *possesses all required signatures*

  it validates all transactions behind this transaction that ultimately resulted in the creation of the inputs of the proposed transactions. This is called ***"walking the chain".***

4.2) Uniqueness consensus

- The first mechanism is concerned with the validation of the transaction, ensuring that all required signatures are available and the states are appropriate.

- The second mechanism is a means *to detect double-spend attacks* and ensures that a *transaction has not already been spent* and is *unique*.

- Uniqueness consensus is only *checked by a notary service*.

5) Flows

- Flows in Corda are a novel idea that *allows the development of decentralized workflows*

- All communication on the Corda network is handled by these flows

CorDapps

- A **CorDapp (Corda Distributed Application**) is a ***distributed application that runs on a Corda network.***

- It allows *nodes* on the network to *reach an agreement regarding updates to the ledger.*

- There are three main components in a Corda smart contract, as follows:
  - **Executable code:** which defines the validation logic to *validate changes to the state objects.*
  - **State objects:** represent the *current state of a contract* and can either be consumed by a transaction or produced (created) by a transaction.
  - **Commands:** are used to *describe the operational and verification data* that defines how a transaction can be verified.

Components

1. Nodes

A **node** is a JVM runtime with a unique identity on the network.

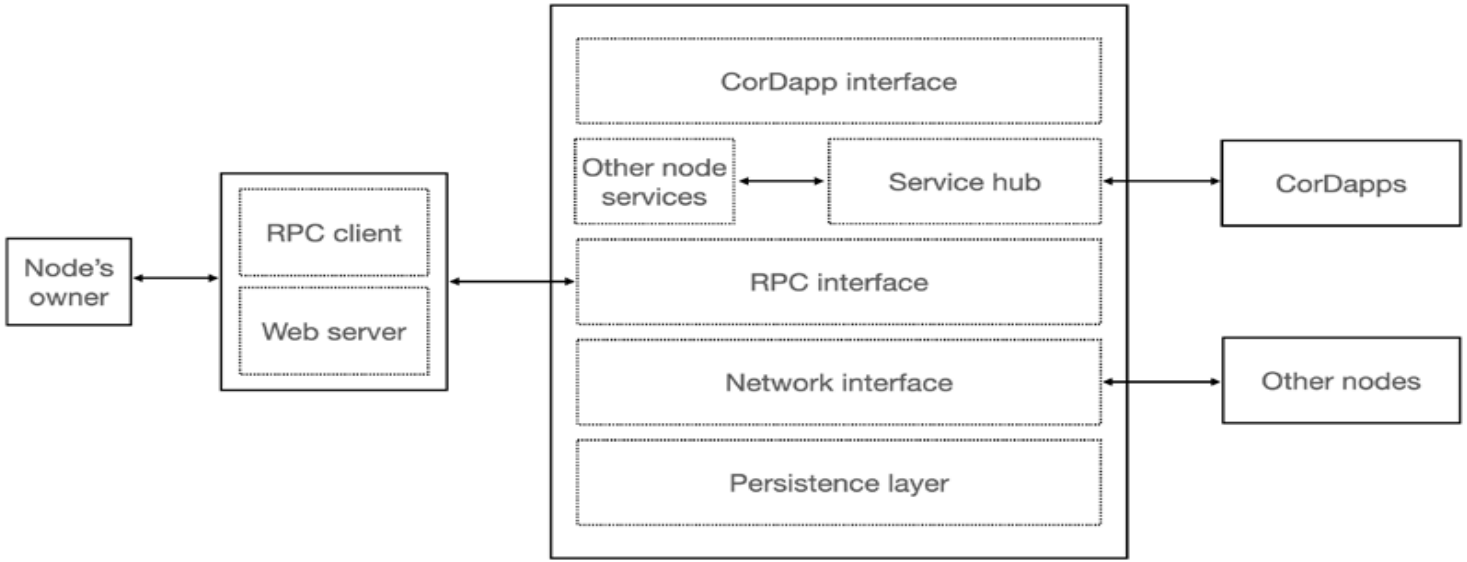


Figure 20.7: Simplified Corda node architecture

key elements of a Corda node

- A **Persistence layer,** which consists of a vault and storage service. It is responsible for *storing data in a local SQL database*.
- A **Network interface,** which *enables communication with other nodes* as part of running a flow.
- An **RPC interface** is responsible for providing interface for *interaction with the node's owner via RPC clients and/or web servers*.
- A **Service hub** enables *flows in the node to call other services* (utilities in a node) such as storage and vault services.
- A **CorDapp interface and provider,** which allows the *extending of a node with CorDapps.*

Transaction flow

The flow process can be summarized as follows:

1. An HTTP request via API or RPC call is made directly to flows in Corda. This is a request to start a transaction.
2. Start a flow. Flows contain the business logic.
3. Perform initial steps such as transaction building and verification.
4. Obtain counterparty signatures.
5. Sign the transaction and check signatures.
6. Obtain notarization/finality.
7. Record the transaction and store it in persistent storage and Vaults. (Vaults are similar to wallets, and store their data in a standard relational database and, as such, can be queried by using standard SQL)
8. Respond via an HTTP response through API.

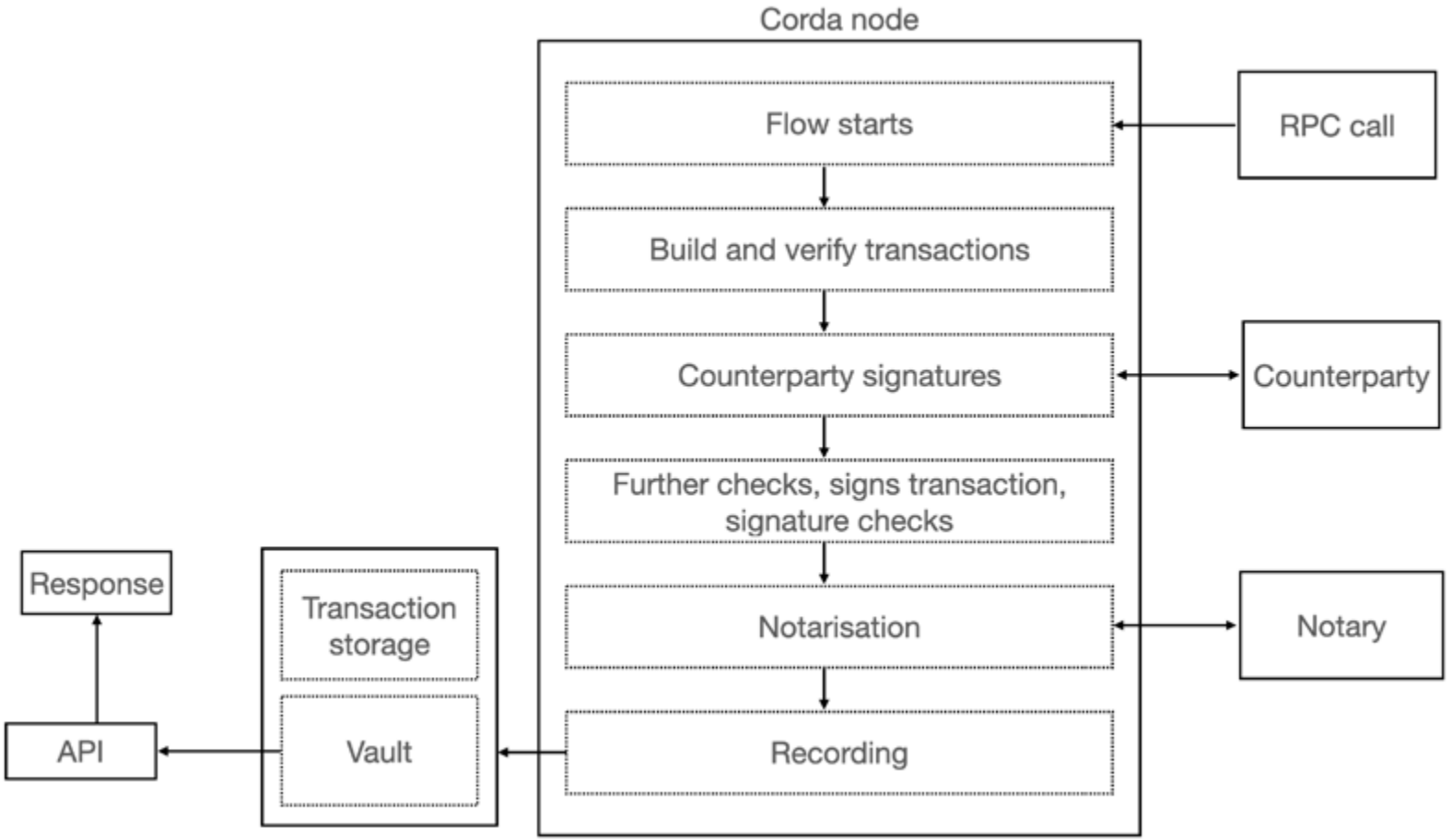


Figure 20.8: Node high-level flow