NATIONAL INSTITUTE OF TRANSPORT

Blockchain Technology Short Training

Introduction to Hyperledger Fabric

Facilitator: Dr. Cleverence Kombe (PhD)

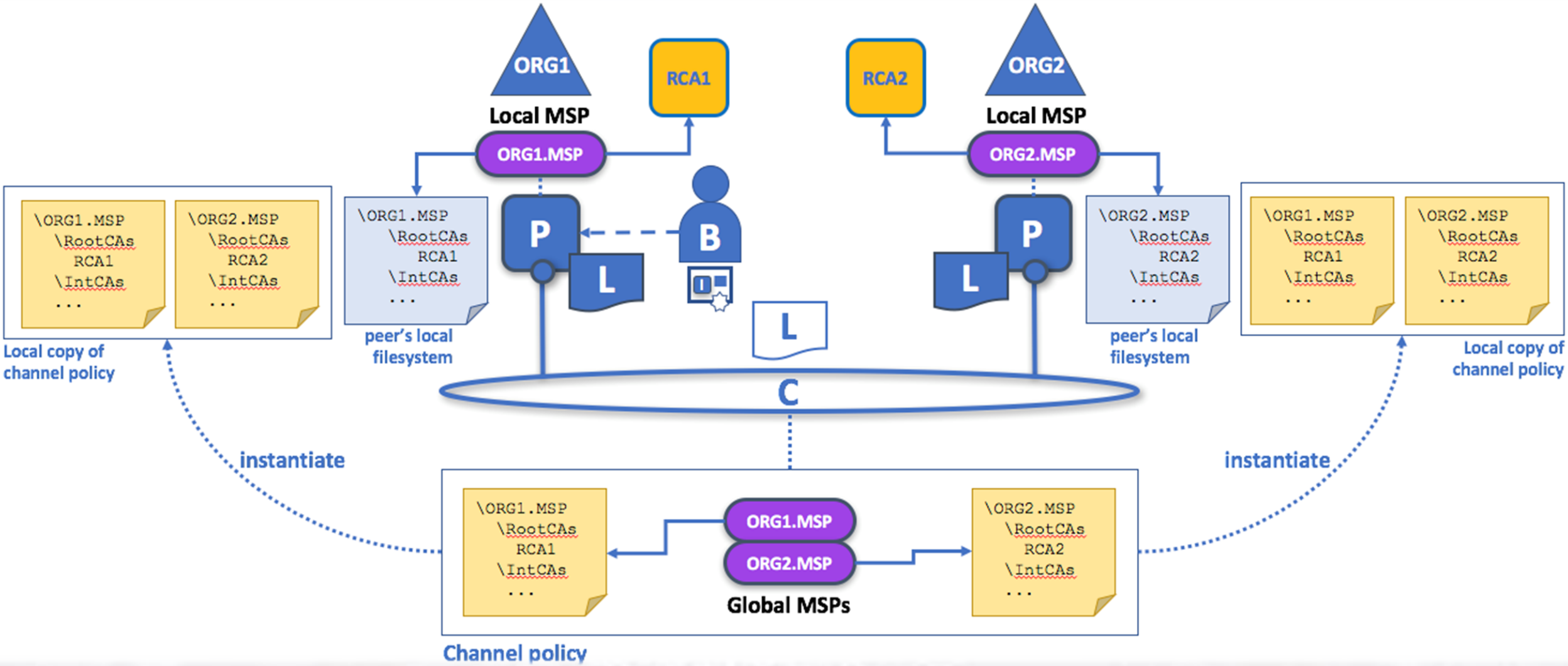
Introduction to Hyperledger | Overview

- Hyperledger Fabric reference architecture
  - *Membership Services*
  - *Transactions*
  - *Channels*
  - *Contract Services*
  - *Ledger*
  - *Client SDK*
  - *Events*
  - *Consensus*
  - *Private data*

Hyperledger Fabric Architecture | Membership Services

- This component manages the identities of participants in the network and is responsible for authentication and authorization.
- Membership Services include:
  - **Membership Service Provider (MSP):** The MSP is responsible for authenticating and authorizing network participants. It establishes an identity for each participant and provides a mechanism for verifying that identity when the participant interacts with the network. MSPs are configured to manage identities for specific organizations in the network.
  - **Certificate Authority (CA):** The CA issues digital certificates that are used by participants to authenticate their identity when they interact with the network. Each participant is issued a unique certificate that can be used to verify their identity during transactions.

Hyperledger Fabric Architecture | Membership Services



Hyperledger Fabric Architecture | Membership Services

- Suppose there is a network consisting of two organizations - **Org1** and **Org2**. Each organization has several members who need to interact with the network to perform transactions.
- The following steps describe how Membership Services would work in this scenario:
    1. Each organization establishes its own MSP to manage the identities of its members.
    2. Each member in an organization enrolls with the organization's MSP to receive a digital certificate that verifies their identity.
    3. Each organization also runs its own CA to issue digital certificates to its members.

Hyperledger Fabric Architecture | Membership Services

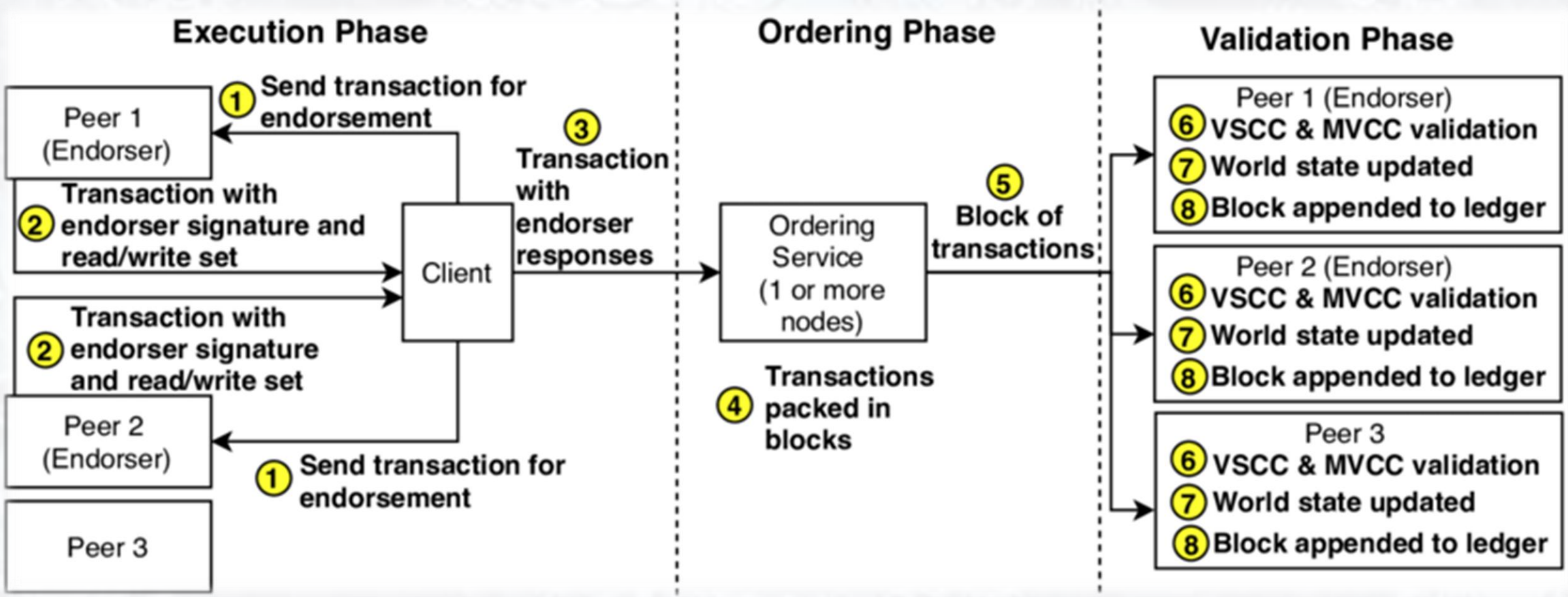
- Suppose there is a network consisting of two organizations - **Org1** and **Org2**. Each organization has several members who need to interact with the network to perform transactions.
- The following steps describe how Membership Services would work in this scenario:

4. When a member from Org1 wants to interact with the network, they present their digital certificate issued by Org1's CA to the network. The MSP for Org1 verifies the certificate and grants the member access to the network.
5. When a member from Org2 wants to interact with the network, they present their digital certificate issued by Org2's CA to the network. The MSP for Org2 verifies the certificate and grants the member access to the network.

Hyperledger Fabric Architecture | Transactions

- Transaction is a fundamental concept that represents a change in the state of the ledger.
- Transactions in Fabric are executed using smart contracts, which are also known as chaincode.
- Transactions can be initiated by a client application, which sends a transaction proposal to the endorsing peers. Once the proposal is endorsed, the transaction is submitted to the ordering service, which orders the transactions into a block and distributes the block to the peers in the network for validation and commitment to the ledger.

Hyperledger Fabric Architecture | Transactions



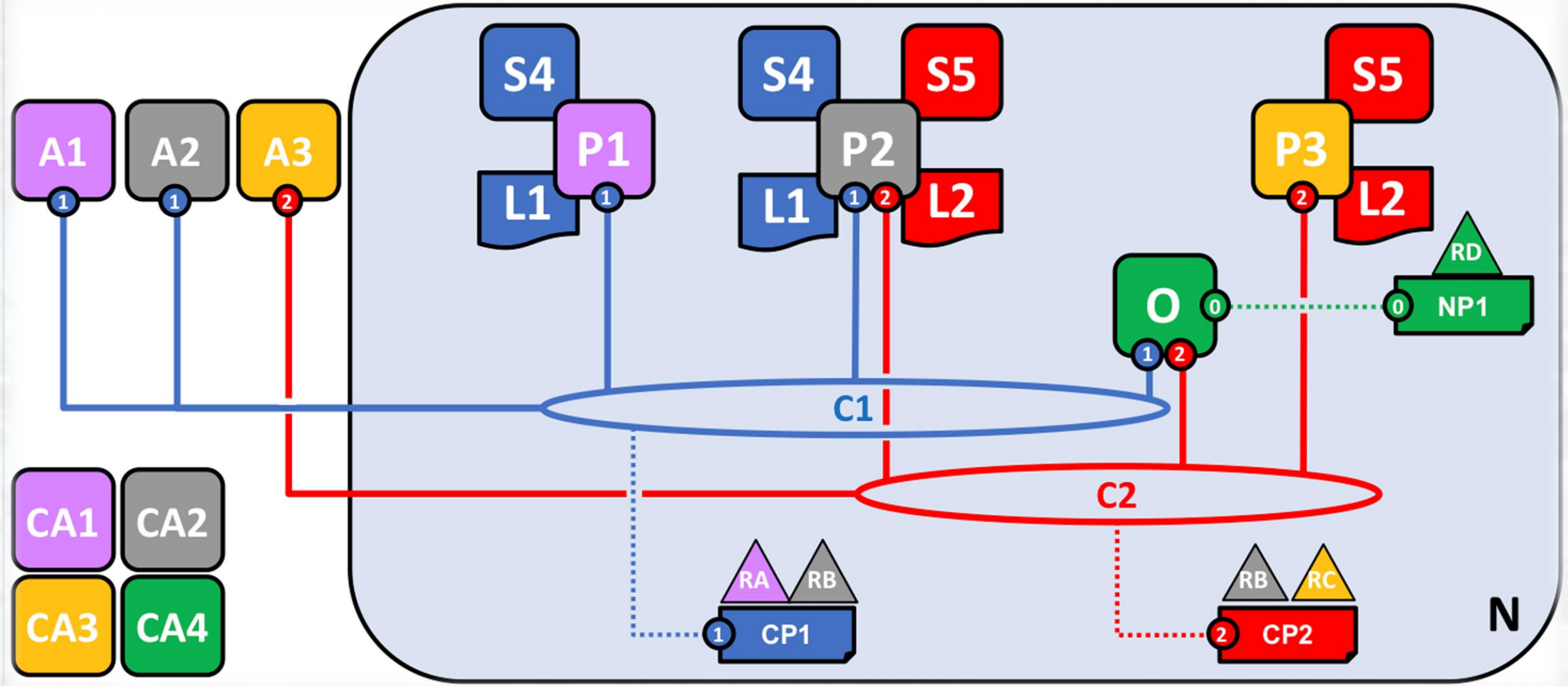
Hyperledger Fabric Architecture | Transactions

- Each transaction in Fabric has a unique identifier called a transaction ID. This ID is generated by the client application that initiates the transaction, and it is used to track the progress of the transaction as it moves through the network.
- Fabric supports multiple transaction types, including read-only transactions and read-write transactions.
- Read-only transactions are used to query the state of the ledger, while read-write transactions are used to modify the state of the ledger.

Hyperledger Fabric Architecture | Channels

- Channels in Hyperledger Fabric are a mechanism for organizing the members of a network into subgroups, where each subgroup can share information and transact with each other.
- Channels allow for private communication and transactions between specific parties, without the need to share information or transactions with the entire network.
- Channels in Hyperledger Fabric can be used to create private sub-networks between different organizations or members, and can be used to ensure that only specific members have access to certain transactions or data.
- This is useful for networks with multiple participants, where some data or transactions may be sensitive or proprietary, and must be kept private.

Hyperledger Fabric Architecture | Channels



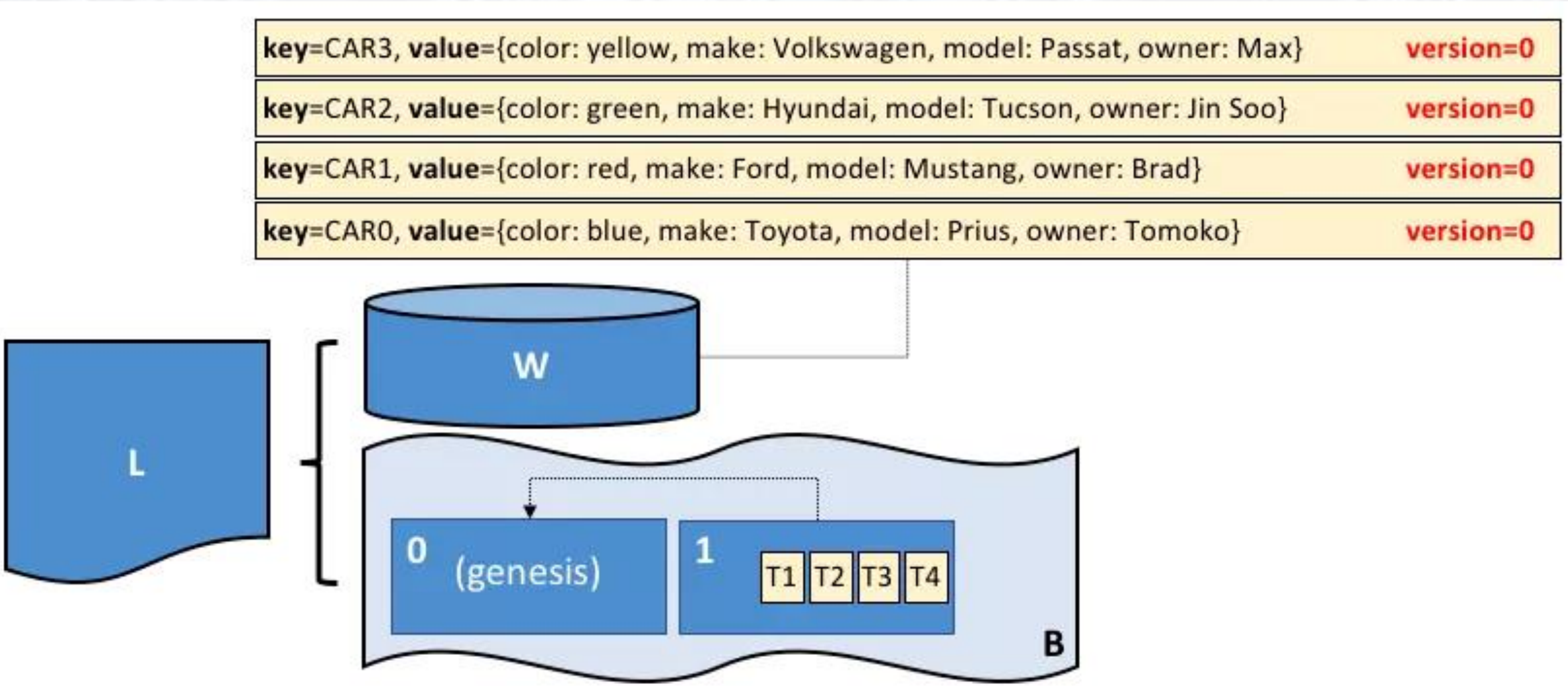
Hyperledger Fabric Architecture | Channels

- Channels in Hyperledger Fabric work by creating a separate, private blockchain for each channel. Each channel has its own set of members and its own ledger, which is separate from the main blockchain.
- Transactions that are sent over a channel are only visible to the members of that channel, and are not visible to the other members of the network.
- Channels are created and managed by the members of the network, and can be configured with different access controls and permissions.
- This allows the members of a channel to determine who can access the channel, and what data or transactions are visible to different members.

Hyperledger Fabric Architecture | Ledger

- The Hyperledger Fabric Ledger is a key component of the Fabric architecture that serves as the system of record for all transactions and state changes that occur in the network.
- It serves as a tamper-evident record of all transactions that have taken place on the blockchain network.
- The ledger can be thought of as a database that stores all of the state changes that have occurred on the network.
- Each state change is recorded as a transaction and stored in a block that is then added to the chain of previous blocks, creating an immutable record of the network's history.

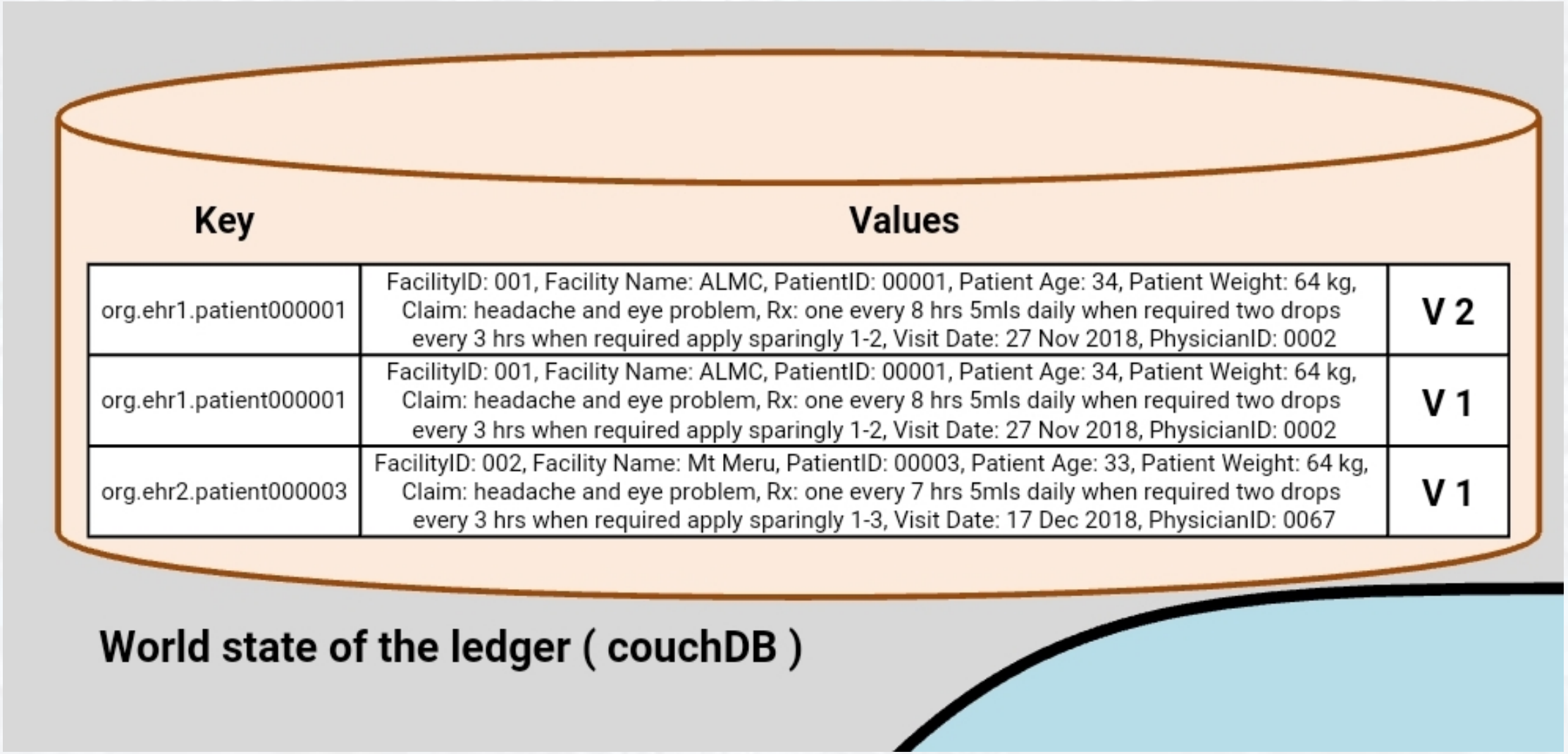
Hyperledger Fabric Architecture | Ledger



Hyperledger Fabric Architecture | Ledger

- The ledger is divided into two parts: **the world state** and **the transaction log**.
  - The World State is a database that contains the current state of the blockchain network.
  - It is a key-value store where each key corresponds to an asset or entity, and the value represents the current state of that asset or entity.
  - The World State is used to maintain the current state of the network, and it is updated every time a new transaction is added to the blockchain.

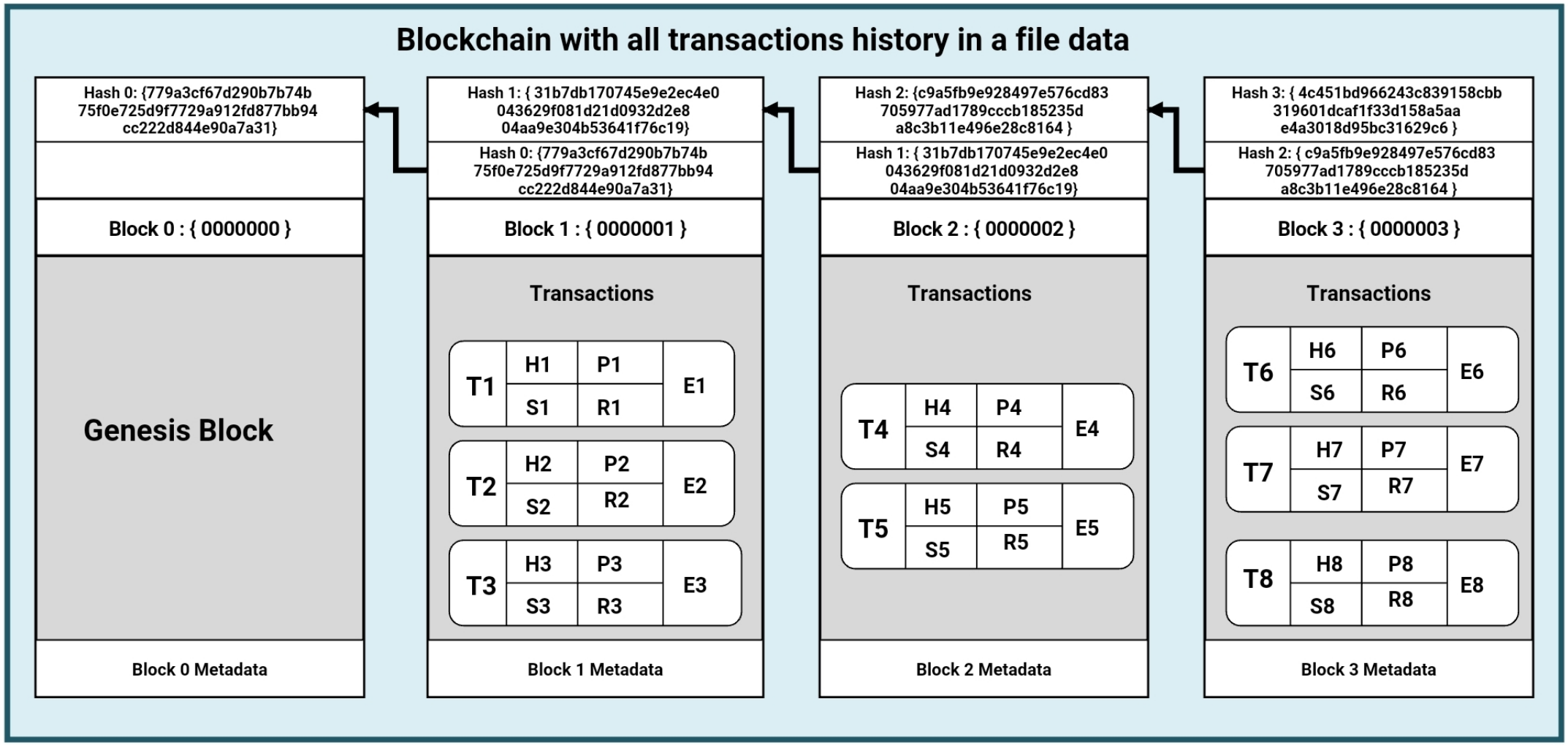
Hyperledger Fabric Architecture | Ledger



Hyperledger Fabric Architecture | Ledger

- The ledger is divided into two parts: **the world state** and **the transaction log**.
  - The Transaction Log, also known as the blockchain, is a tamper-proof record of all the transactions that have been executed in the network.
  - It is composed of a series of blocks, with each block containing a batch of transactions.
  - Each block is linked to the previous block, forming a chain of blocks, hence the name blockchain.

Hyperledger Fabric Architecture | Ledger



Hyperledger Fabric Architecture | Contract Services

- Contract Services in Hyperledger Fabric are responsible for executing chaincode, also known as smart contracts. Chaincode is written in a specific programming language, and it defines the logic that will be executed on the network.
- The chaincode is a program that runs on a set of peers and is responsible for handling transactions that are specific to an application.
- In Fabric, chaincode can be implemented using the Go, JavaScript, or Java programming languages. This provides developers with the flexibility to choose the language that is most suitable for their use case.

Hyperledger Fabric Architecture | Contract Services

- Chaincode can be executed on a single peer or multiple peers in a network.
- Hyperledger Fabric uses containerization technology to deploy and execute chaincode.
- Each chaincode is deployed in a separate Docker container. This allows for easy deployment and upgrade of chaincode without disrupting the rest of the network.

Hyperledger Fabric Architecture | Contract Services

- The **context** and **contract classes** in Hyperledger Fabric work together to enable the execution of smart contracts on a blockchain network.
- The **context class** provides the infrastructure and tools necessary for the smart contract to interact with the Hyperledger Fabric blockchain, while
- The **contract class** implements the business logic that defines the behavior of the smart contract.

Hyperledger Fabric Architecture | Contract Services

- The **context class** in Hyperledger Fabric represents the runtime context for a smart contract. It is responsible for providing access to the ledger, transaction data, and other services necessary for executing the smart contract.
- The **context class** contains various properties and methods that enable the smart contract to interact with the underlying blockchain network.
- These include functions for accessing the ledger state, handling events, and managing transactions. The context class acts as a bridge between the smart contract and the blockchain network.

Hyperledger Fabric Architecture | Contract Services

- The **contract class** in Hyperledger Fabric represents the actual smart contract logic. It contains the business rules and algorithms that define how the smart contract behaves.
- The **contract class** is responsible for processing transactions, updating the ledger state, and performing other operations based on the input provided by the context class.
- The contract class defines various functions that can be invoked by external clients to interact with the smart contract. These functions typically take inputs in the form of JSON objects and return results in a similar format.

Hyperledger Fabric Architecture | Contract Services

```
1   const { Context } = require('fabric-contract-api');
2
3   class MyContext extends Context {
4     constructor() {
5       super();
6     }
7
8     async getAsset(key) {
9       return await this.stub.getState(key);
10    }
11
12    async putAsset(key, value) {
13      return await this.stub.putState(key, Buffer.from(JSON.stringify(value)));
14    }
15  }
16
```

Hyperledger Fabric Architecture | Contract Services

```
1   const { Contract } = require('fabric-contract-api');
2
3   class MyContract extends Contract {
4     constructor() {
5       super('mycontract');
6     }
7
8     async createAsset(ctx, key, value) {
9       await ctx.putAsset(key, value);
10    }
11
12    async readAsset(ctx, key) {
13      const asset = await ctx.getAsset(key);
14      return JSON.parse(asset.toString());
15    }
16  }
17
```

Hyperledger Fabric Architecture | Client SDK

- Hyperledger Fabric's Client SDK provides a collection of APIs and libraries that enable client applications to interact with the network. The SDK supports several programming languages, including Go, Node.js, and Java.
- Let us discuss about **Node.js Client SDK**.
- Node.js is a popular JavaScript runtime that is widely used for developing server-side applications. It is a lightweight and efficient platform that is well-suited for building blockchain applications.
- Hyperledger Fabric provides a Node.js SDK that makes it easy to interact with the network using JavaScript.

Hyperledger Fabric Architecture | Client SDK

- The Node.js SDK provides a set of APIs for performing various operations on the network, including submitting transactions, querying the ledger, and listening for events.
- The SDK also provides a set of utility functions for working with cryptographic keys and certificates, which are required for secure communication with the network.
- To use the Node.js SDK, you first need to install it using the npm package manager. Here's an example of how to install the SDK:

```
npm install fabric-network
```

Hyperledger Fabric Architecture | Client SDK

- Once you have installed the SDK, you can use it to connect to the network and perform various operations. Here's an example of how to connect to the network using the SDK:

```
const { Gateway, Wallets } = require('fabric-network');
const path = require('path');

async function main() {
    const walletPath = path.join(process.cwd(), 'wallet');
    const wallet = await Wallets.newFileSystemWallet(walletPath);
    const gateway = new Gateway();
    const connectionProfilePath = path.resolve(__dirname, '..',
'connection.json');
```

Hyperledger Fabric Architecture | Client SDK

- Once you have installed the SDK, you can use it to connect to the network and perform various operations. Here's an example of how to connect to the network using the SDK:

```
const connectionOptions = {
        identity: 'user1',
        wallet: wallet,
        discovery: { enabled: true, asLocalhost: true }
    };
    await gateway.connect(connectionProfilePath,
connectionOptions);
    const network = await gateway.getNetwork('mychannel');
    const contract = network.getContract('mychaincode');
```

Hyperledger Fabric Architecture | Client SDK

- Once you have installed the SDK, you can use it to connect to the network and perform various operations. Here's an example of how to connect to the network using the SDK:

```
const contract = network.getContract('mychaincode');
    const result = await contract.submitTransaction('createAsset',
'asset1', 'blue', '10', 'Tom');
    console.log(result.toString());
    await gateway.disconnect();
}
main();
```

Hyperledger Fabric Architecture | Events

- In Hyperledger Fabric, events are notifications that are triggered when certain conditions are met on the blockchain. These conditions can be related to transactions, blocks, or other aspects of the blockchain.
- Events are used to signal important changes in the network, and they allow client applications to receive real-time updates about the state of the blockchain.
- Events are implemented using a publish/subscribe model. Clients subscribe to specific events using a topic, and the blockchain network sends notifications to all subscribers when the event occurs. This enables clients to stay informed about changes in the blockchain, without the need for constant polling or other inefficient methods.

Hyperledger Fabric Architecture | Events

- Events can be used for a variety of purposes, such as triggering actions in response to certain transactions or providing real-time updates to user interfaces.
- For example, a client application might subscribe to an event that is triggered when a new block is added to the blockchain.
- When the event is triggered, the client can update its user interface to reflect the new state of the blockchain.

Hyperledger Fabric Architecture | Events

- Example:

```
    // Subscribe to an event
    const eventHub =
network.getChannel().newChannelEventHub('localhost:7051');
    eventHub.registerChaincodeEvent('mychaincode',
'updateAsset', (event, blockNum, txid, status) => {
        console.log(`Received event:
${event.payload.toString()} on block ${blockNum} with txid
${txid} and status ${status}`);
    });
```

Hyperledger Fabric Architecture | Events

- Example:

```
        // Submit a transaction that triggers the event
        await contract.submitTransaction('updateAsset',
'asset1', 'blue', '10', 'Tom');

        await gateway.disconnect();
}
main();
```

Hyperledger Fabric Architecture | Consensus

- Consensus is the process by which the network agrees on the current state of the blockchain. In a blockchain network, each node maintains a copy of the ledger, and updates are propagated across the network as transactions are added to the ledger.
- Consensus is necessary to ensure that all nodes have a consistent view of the state of the network, and to prevent inconsistencies and potential attacks.
- Hyperledger Fabric supports pluggable consensus mechanisms, which means that the network can use different algorithms to achieve consensus. Each consensus algorithm has its own strengths and weaknesses, and the choice of algorithm depends on the specific requirements of the network.

Hyperledger Fabric Architecture | Consensus

- Here are some examples of consensus algorithms that can be used in Hyperledger Fabric:
  - **Kafka:** This is a high-throughput distributed messaging system that is often used in big data applications.
  - Kafka can be used as a consensus mechanism in Hyperledger Fabric, where it provides ordering of transactions and ensures that all nodes in the network have the same view of the ledger.
  - Kafka uses "leader and follower" concept, in which transactions are replicated from the leader node to the follower nodes. In the event the leader node goes down, one of the followers becomes the leader and ordering can continue, ensuring fault tolerance.

Hyperledger Fabric Architecture | Consensus

- Here are some examples of consensus algorithms that can be used in Hyperledger Fabric:
  - **PBFT (Practical Byzantine Fault Tolerance):** This is a consensus algorithm that is designed to be tolerant of Byzantine faults, which are faults that can cause nodes to behave in arbitrary and malicious ways.
  - PBFT is a well-established consensus algorithm that is used in many blockchain networks, and it provides strong security guarantees.

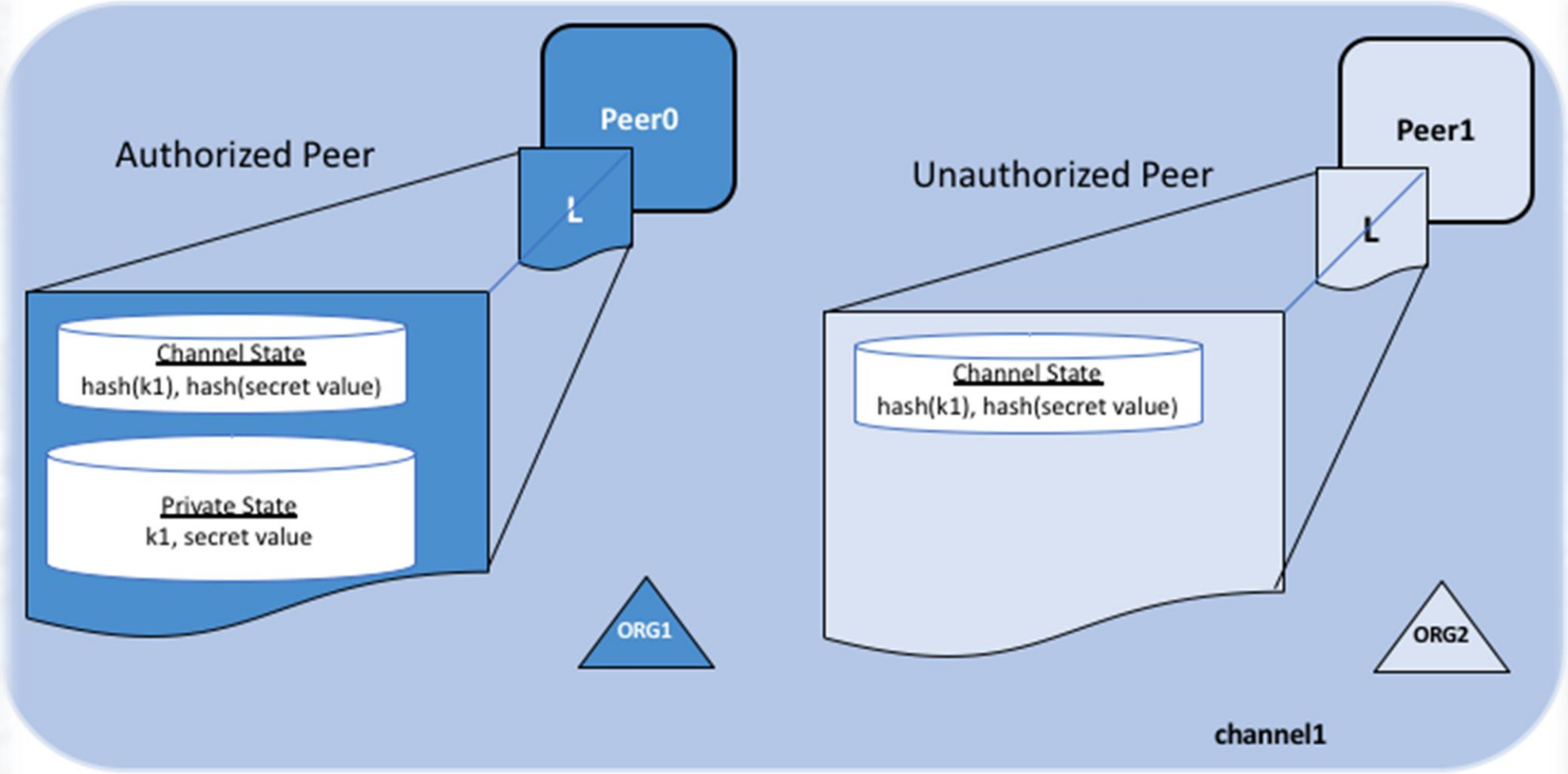
Hyperledger Fabric Architecture | Consensus

- Here are some examples of consensus algorithms that can be used in Hyperledger Fabric:
  - **Raft:** This is a consensus algorithm that is often used in distributed systems. In Raft, each node maintains a replicated state machine, and nodes communicate with each other to agree on the current state of the system.
  - Raft follows a "leader and follower" model, where a leader node is elected (per channel) and its decisions are replicated by the followers.
  - Raft ordering services should be easier to set up and manage than Kafka-based ordering services, and their design allows different organizations to contribute nodes to a distributed ordering service.

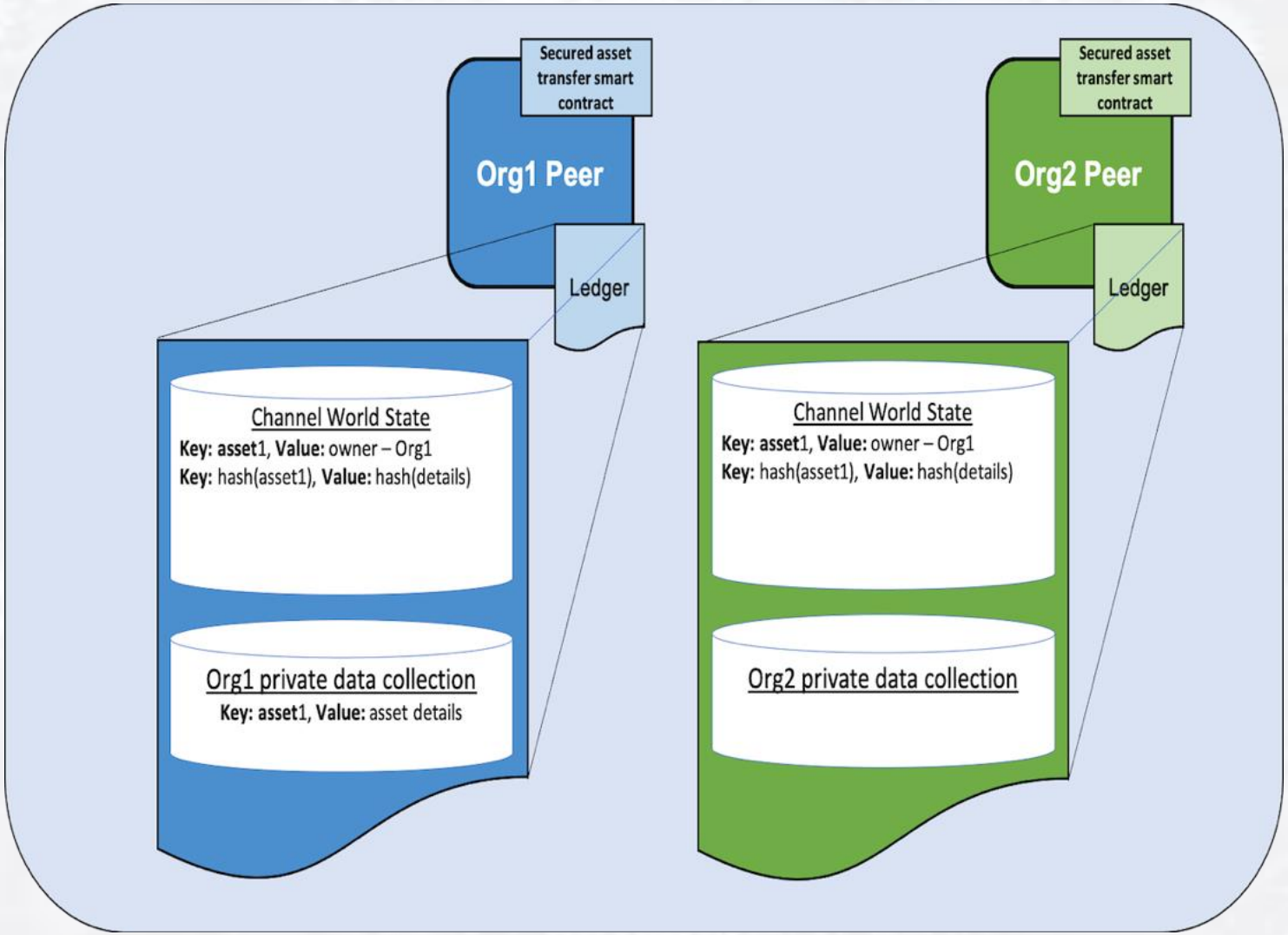
Hyperledger Fabric Architecture | Private Data

- Hyperledger Fabric provides a feature called private data that allows network participants to keep certain data private from other participants. This can be particularly important in business networks where certain data needs to be kept confidential, such as financial data or personally identifiable information.
- Private data in Hyperledger Fabric is achieved by encrypting the data and storing it on a private data collection, which is only accessible to authorized parties.
- The private data collection is stored on the peer nodes that are authorized to access the data, and it is not replicated to other peer nodes. This means that the data is kept private and is only visible to the authorized parties.

Hyperledger Fabric Architecture | Private Data



Hyperledger Fabric Architecture | Private Data



Introduction to Blockchain | Summary

►In this topic, we discussed:

- Hyperledger Fabric reference architecture
    - Membership Services
    - Transactions
    - Channels
    - Contract Services
    - Ledger
    - Client SDK
    - Events
    - Consensus
    - Private data


THANK
YOU