Introduction to Hyperledger Fabric

Instructor: Dr. Md Sadek Ferdous

Assistant Professor, CSE, SUST

E-mail: ripul.bd@gmail.com

Outline

- Hyperledger Project
- Fabric Introduction
- Fabric Concepts

Motivations for private blockchain systems

- As blockchain tech gaining maturity, interest in applying the underlying technology of the blockchain, distributed ledger and distributed application platform to more innovative *enterprise* use cases started to grew
- However, public blockchain systems are unsuitable for many enterprise use cases
- For enterprise use, we need to consider the following requirements:
  - Participants must be identified/identifiable
    - E.g. financial transactions where Know-Your-Customer (KYC) and Anti-Money Laundering (AML) regulations must be followed
  - Networks need to be *permissioned*
  - High transaction throughput performance
  - Low latency of transaction confirmation
  - Privacy and confidentiality of transactions and data pertaining to business transactions

Hyperledger

- Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies
- It is a global collaboration, hosted by The Linux Foundation
- It including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology
  - Joining forces to develop and promote private blockchain systems

Hyperledger

- Major Hyperledger projects:
  - Fabric – initially developed by IBM which is then open-sourced, released and incubated under the Hyperledger project
  - Sawtooth – developed by Intel which is then released under the Hyperledger project
  - Burrow – a permissioned version of Ethereum released under the Hyperledger project
  - Caliper – a benchmarking platform for Hyperledger projects
  - Composer – a GUI tool to develop complex business networks
- Fabric is the most advanced project under the Hyperledger project

Hyperledger Fabric

- Hyperledger Fabric is an open source enterprise-grade permissioned distributed ledger technology (DLT) platform
- Designed for use in enterprise contexts
  - To deliver some key differentiating capabilities over other popular distributed ledger or blockchain platforms

Hyperledger Fabric

- Fabric is highly modular and configurable
- Fabric is the first distributed ledger platform to support smart contracts authored in general-purpose programming languages
  - such as Java, Go and Node.js, rather than constrained domain-specific languages (DSL)
- Fabric supports pluggable consensus protocols that enable the platform to be more effectively customised to fit particular use cases and trust models
- Fabric does not require a native cryptocurrency

Fabric: modularity

- A pluggable *ordering service* establishes consensus on the order of transactions and then broadcasts blocks to peers
- A pluggable *membership service provider* is responsible for associating entities in the network with cryptographic identities
- An optional *peer-to-peer gossip service* disseminates the blocks output by ordering service to other peers
- Smart contracts (“chaincode”) run within a container environment (e.g. Docker) for isolation
  - They can be written in standard programming languages but do not have direct access to the ledger state
- The ledger can be configured to support a variety of DBMSs
- A pluggable endorsement and validation policy enforcement that can be independently configured per application

Fabric: privacy confidentiality

- Hyperledger Fabric, being a permissioned platform, enables confidentiality through its channel architecture
- Basically, participants on a Fabric network can establish a “channel” between the subset of participants that should be granted visibility to a particular set of transactions
  - Think of this as a network overlay
- Thus, only those nodes that participate in a channel have access to the smart contract (chaincode) and data transacted, preserving the privacy and confidentiality of both

Fabric: pluggable consensus

- Since consensus is modular, its implementation can be tailored to the trust assumption of a particular deployment or solution
- This modular architecture allows the platform to rely on well-established toolkits for CFT (crash fault-tolerant) or BFT (byzantine fault-tolerant) ordering
- In the currently available releases, Fabric offers a CFT ordering service implemented with Kafka and Zookeeper
- In subsequent releases, Fabric will deliver a Raft consensus ordering service

Fabric: performance and scalability

- Fabric is quite scalable because of its permissioned model of achieving consensus
- Performance of Fabric is reported to be quite satisfactory
  - Around 3500 tx/s (https://arxiv.org/pdf/1801.10228v1.pdf)

Fabric functionalities

- Fabric relies on a concrete identity management architecture
- Hyperledger Fabric provides a membership identity service that manages user IDs and authenticates all participants on the network
- Access control lists can be used to provide additional layers of permission through authorisation of specific network operations
  - For example, a specific user ID could be permitted to invoke a chaincode application, but be blocked from deploying new chaincode

Fabric functionalities

- Smart-contract in Fabric is called chaincode
- Chaincode applications encode logic that is invoked by specific types of transactions on the channel
- Chaincode that defines parameters for a change of asset ownership, for example, ensures that all transactions that transfer ownership are subject to the same rules and requirements

Fabric model

- Fabric has been developed in six major design features:
  - Assets
  - Chaincode
  - Ledger
  - Privacy
  - Security & Membership services
  - Consensus

Fabric Assets

- Assets can range from the tangible (real estate and hardware) to the intangible (contracts and intellectual property)
- Hyperledger Fabric provides the ability to modify assets using chaincode transactions
- Assets are represented in Hyperledger Fabric as a collection of key-value pairs, with state changes recorded as transactions on a Channel ledger
- Assets can be represented in binary and/or JSON form

Fabric Chaincode

- Chaincode is software defining an asset or assets, and the transaction instructions for modifying the asset(s)
  - In other words, chaincode encodes business logic
- Chaincode enforces the rules for reading or altering key-value pairs or other state database information
- Chaincode functions execute against the ledger's current state database and are initiated through a transaction proposal
- Chaincode execution results in a set of key-value writes (write set) that can be submitted to the network and applied to the ledger on all peers

Fabric Ledger

- The ledger is the sequenced, tamper-resistant record of all state transitions in the fabric
- State transitions are a result of chaincode invocations (‘transactions’) submitted by participating parties
- Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes
- The ledger is comprised of a blockchain (‘chain’) to store the immutable, sequenced record in blocks, as well as a state database to maintain current fabric state
- There is one ledger per channel
- Each peer maintains a copy of the ledger for each channel of which they are a member

Fabric Privacy

- Hyperledger Fabric employs an immutable ledger on a per-channel basis, as well as chaincode that can manipulate and modify the current state of assets (i.e. update key-value pairs)
- A ledger exists in the scope of a channel — it can be shared across the entire network (assuming every participant is operating on one common channel)
- Or it can be privatised to include only a specific set of participants

Fabric Privacy

- To further obfuscate the data, values within chaincode can be encrypted (in part or in total)
  - using common cryptographic algorithms such as AES before sending transactions to the ordering service and appending blocks to the ledger
- Once encrypted data has been written to the ledger, it can be decrypted only by a user in possession of the corresponding key that was used to generate the cipher text

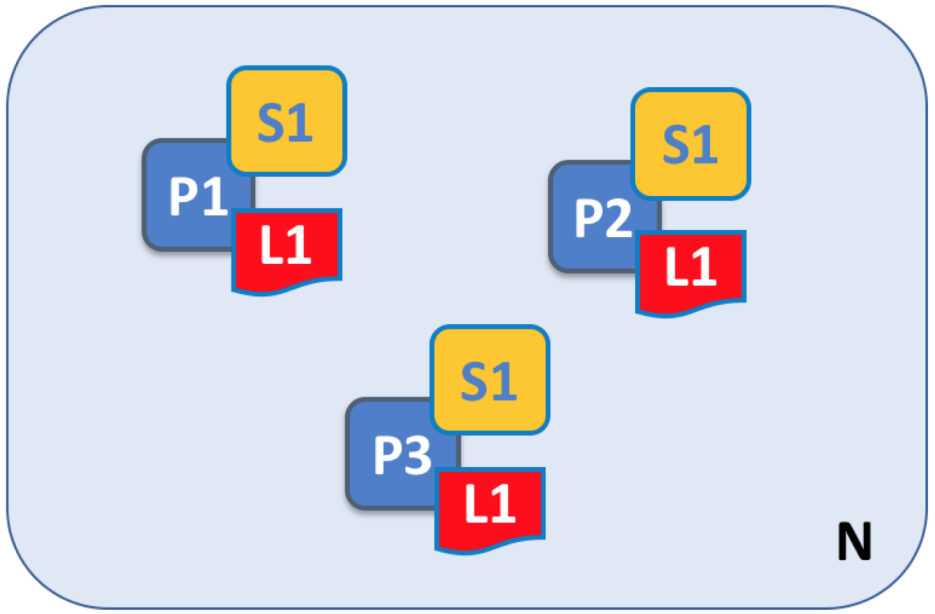
Fabric Security & Membership Services

- Hyperledger Fabric underpins a transactional network where all participants have known identities
- Public Key Infrastructure is used to generate cryptographic certificates which are tied to organisations, network components, and end users or client applications
- As a result, data access control can be manipulated and governed on the broader network and on channel levels
- This “permissioned” notion of Hyperledger Fabric, coupled with the existence and capabilities of channels, helps address scenarios where privacy and confidentiality are paramount concerns

Fabric network

- In this example, the network N consists of peers **P1**, **P2** and **P3**
- Each peer maintains their own instance of the distributed ledger **L1**
- P1, P2 and P3 use the same chaincode, **S1**, to access their copy of that distributed ledger

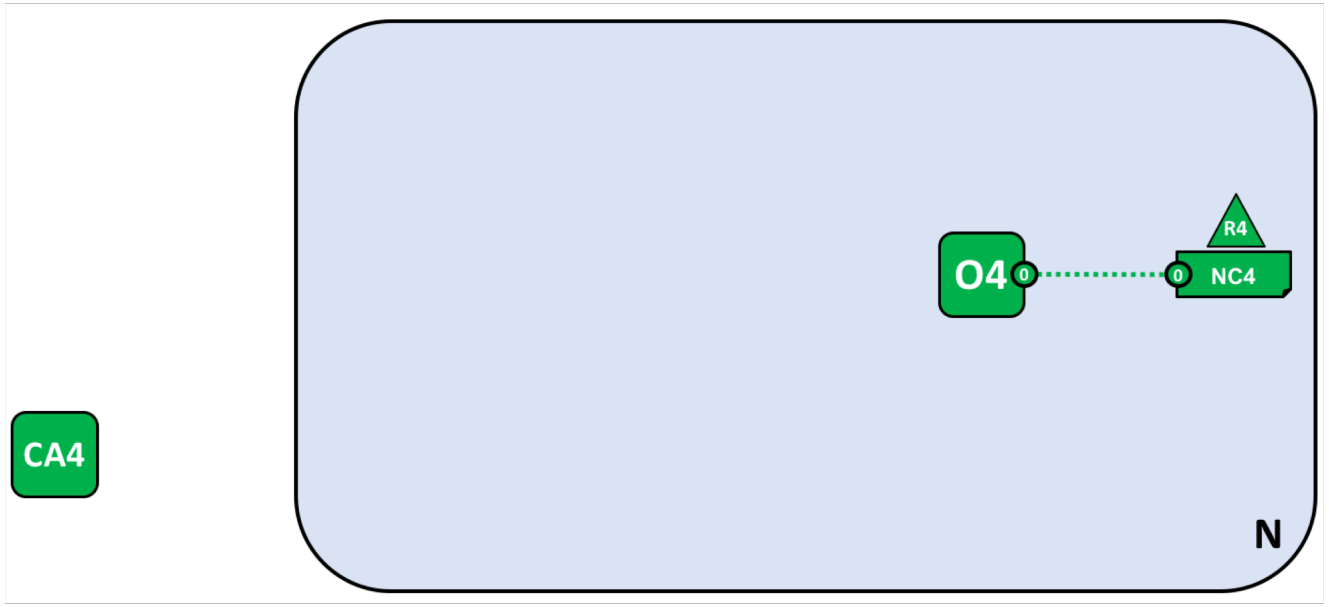
A blockchain network is comprised of peer nodes, each of which can hold copies of ledgers and copies of smart contracts



N	Blockchain network
P	Peer node
S	Smart contract (aka chaincode)
L	Ledger

Fabric network

- We start a network with the initialisation of an orderer
- Our example network, **N**, consists of
  - a single orderer **O4**
  - a single organisation **R4**
  - a certificate authority **CA4**
- **O4** is configured according to a network configuration **NC4**, which gives administrative rights to organisation **R4**
- **CA4** is used to dispense identities to the administrators and network nodes of the **R4** organisation



Fabric network: CA & PKI

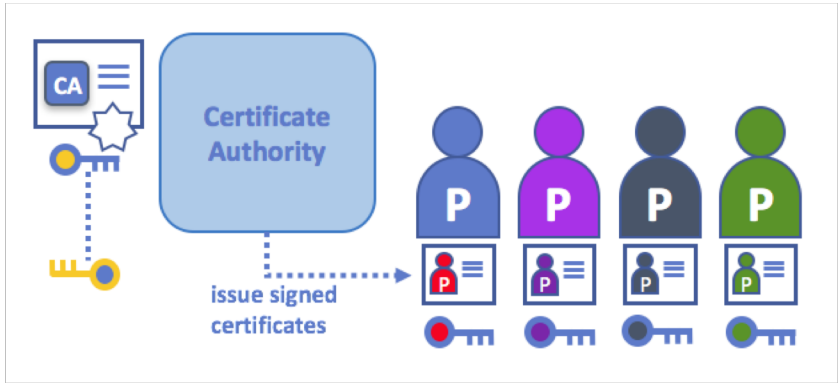
- Remember: users on a blockchain network are presented using public keys
- For any private blockchain, we need **verifiable** identities
  - It means, there must be a way to guarantee that a certain public key represents an authorised entity in a network
  - Based on this assurance we can create rules which will dictate who can participate in the blockchain network
- But, how can we ensure this association?
- We use a Trusted Third Party (**TTP**) which vouches that a certain public key belongs to a certain entity
  - In security, such TTPs are known as Certificate Authorities (**CAs**)

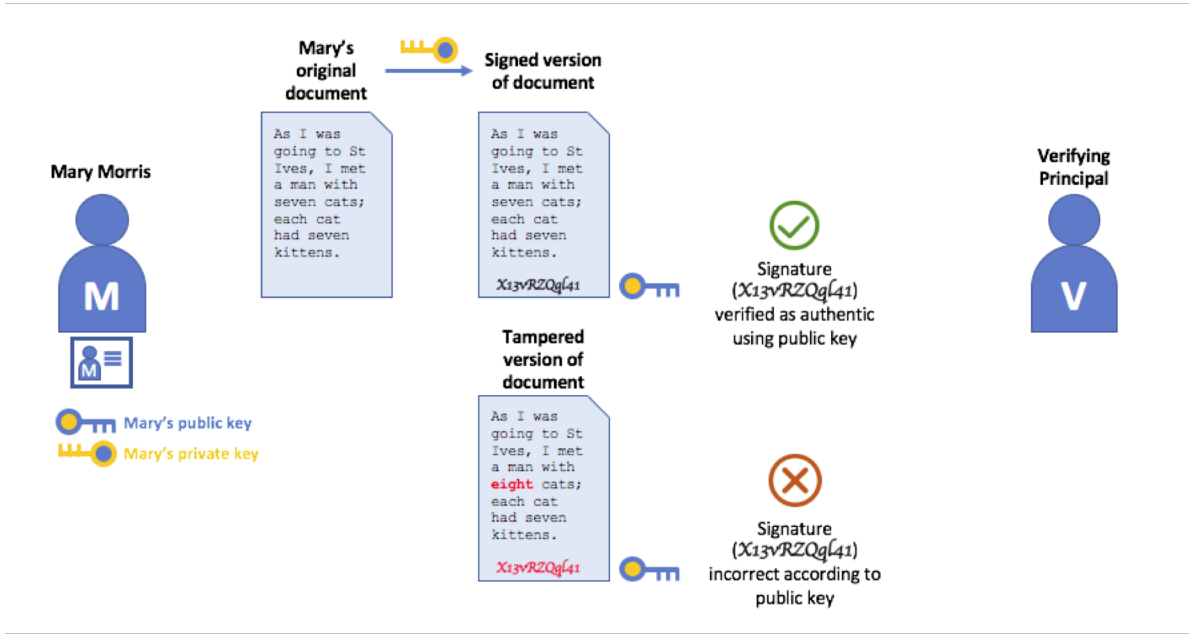
Fabric network: CA & PKI

- A CA issues a digital certificate which is used to bind a public key with an entity
- This association looks something like this:
  - “I, Mango CA, certify that Mr. Rahim belongs to Organisation O4 has this public key P_k and this is valid till 19 December, 2019”

Fabric network: CA & PKI

- Every digital certificate is then signed by the private key of the CA
- Anybody can utilise the public key of the CA to verify the signature in the certificate
- Once verified, everybody trusts the association between the public key and the entity
- If the certificate is tampered, the signature verification will fail

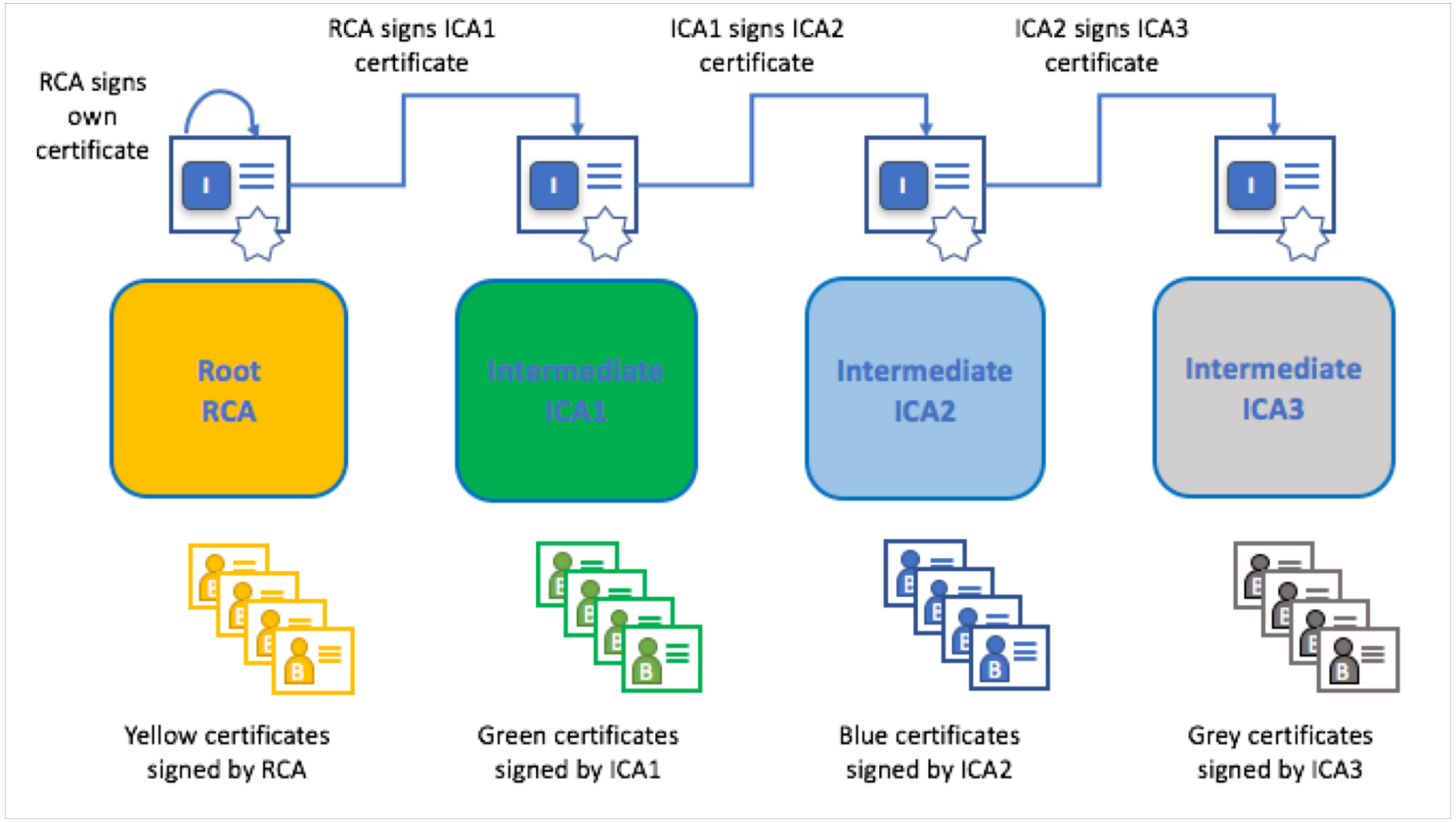




Fabric network: CA & PKI

- But, the problem is how do we get the public key of the CA to verify the signature in the certificate?
  - We used the signed certificate to get the public of an entity
  - Does it mean we will need another certificate to get a CA's public key?
  - Who will sign such certificates? How the signature verification will work?
  - It is a circular problem!
- Solution:
  - Create a hierarchy: Root CAs and Intermediate CAs
- Intermediate CAs distribute certificates to millions of users
- Root CAs distribute certificates for the intermediate Cas
- Embed the public keys of Root CAs to the OS and devices
  - In order to verify the root CA certificates
- Such a hierarchy is known as **PKI** (Public Key Infrastructure)

Fabric network: CA & PKI



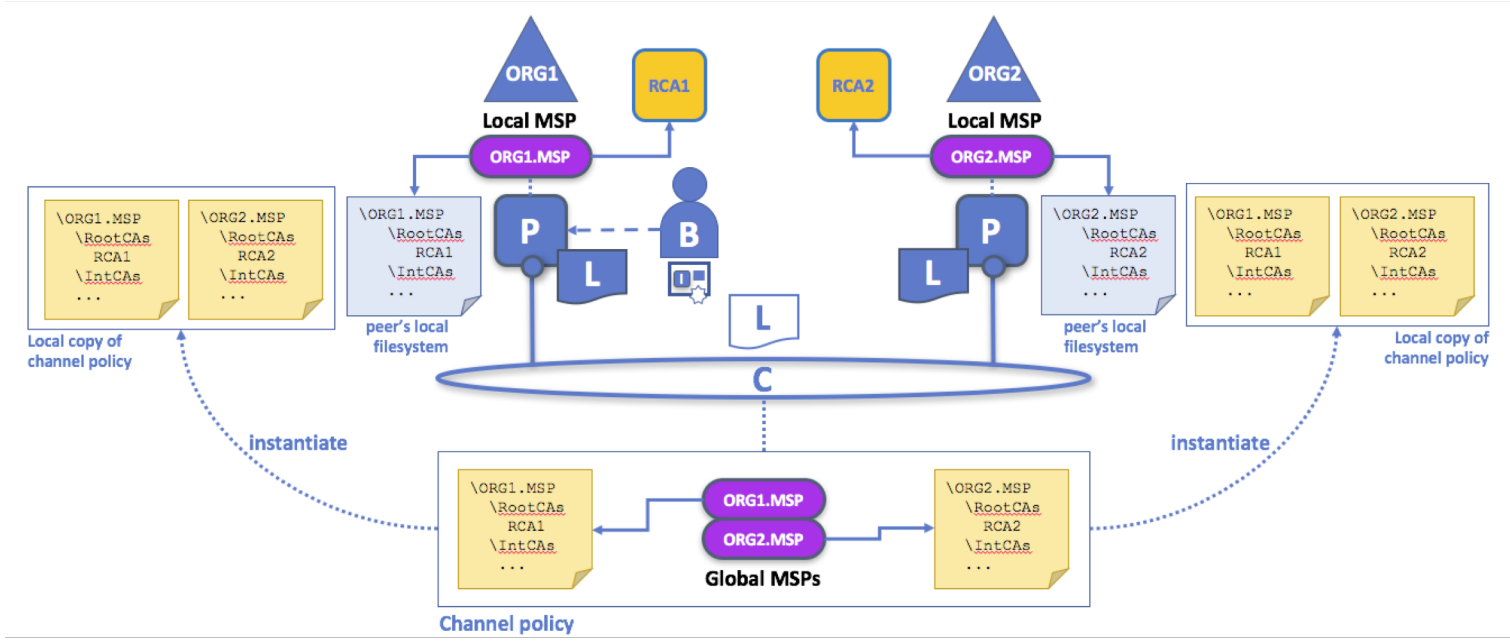
Fabric CA is a private root CA provider capable of managing digital identities of Fabric participants that have the form of **X.509** certificates for the Fabric network

Fabric network: Membership Service Provider

- A Membership Service Provider (MSP) in Fabric identifies which Root CAs and Intermediate CAs are trusted to define the members of an organisation
- It does so either by:
  - listing the identities of their members, or
  - identifying which CAs are authorised to issue valid identities for their members
- An MSP can also identify specific roles an actor might play either within the scope of the organisation the MSP represents (e.g., admins, or as members of a sub-organisation group)
  - Then It can sets the basis for defining access privileges in the context of a network and channel (e.g., channel admins, readers, writers)

Fabric network: Membership Service Provider

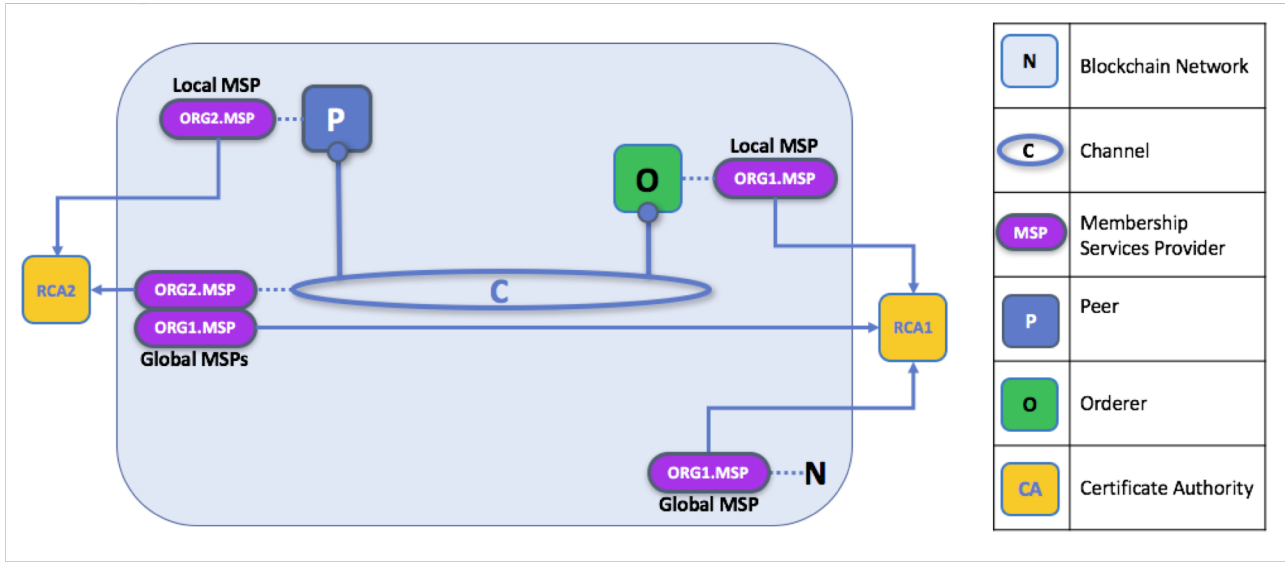
- The trust domain (e.g., the organisation) of each peer is defined by the peer's local MSP, e.g., ORG1 or ORG2
- Representation of an organisation on a channel is achieved by adding the organisation's MSP to the channel configuration
- For example, the channel of this figure is managed by both ORG1 and ORG2
- Similar principles apply for the network, orderers, and users, but these are not shown here for simplicity



Local and channel MSPs.

Fabric network: Membership Service Provider

- ORG1 trusts identities from RCA1, whereas ORG2 trusts identities from RCA2
- Note that these are administration identities, reflecting who can administer these components
- So while ORG1 administers the network, ORG2 maintains the channel
- MSP does exist in the network definition



MSP Levels

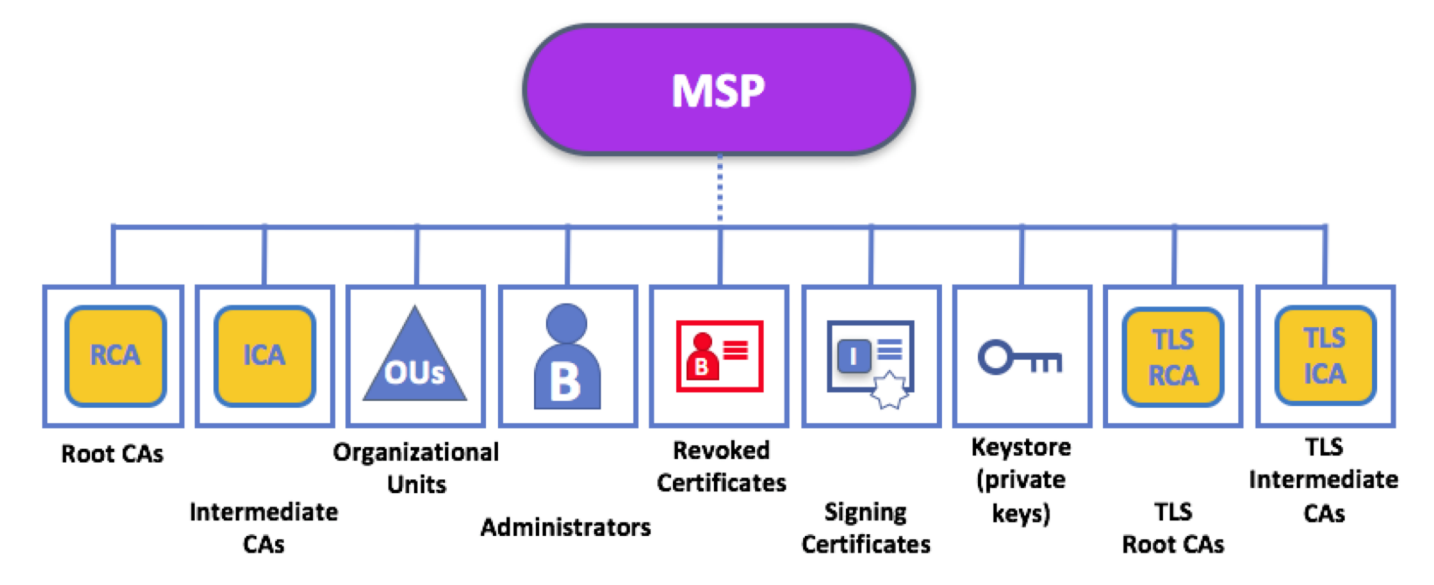
Fabric network: Membership Service Provider

- Network MSP: The configuration of a network defines who are the members in the network
  - by defining the MSPs of the participant organisations and what administrative tasks they can perform (e.g., creating a channel)
- Channel MSP: It is important for a channel to maintain the MSPs of its members separately

Fabric network: Membership Service Provider

- Peer MSP: This local MSP is defined on the file system of each peer and there is a single MSP instance for each peer
- Orderer MSP: Like a peer MSP, an orderer local MSP is also defined on the file system of the node and only applies to that node
  - Like peer nodes, orderers are also owned by a single organisation and therefore have a single MSP to list the actors or nodes it trusts

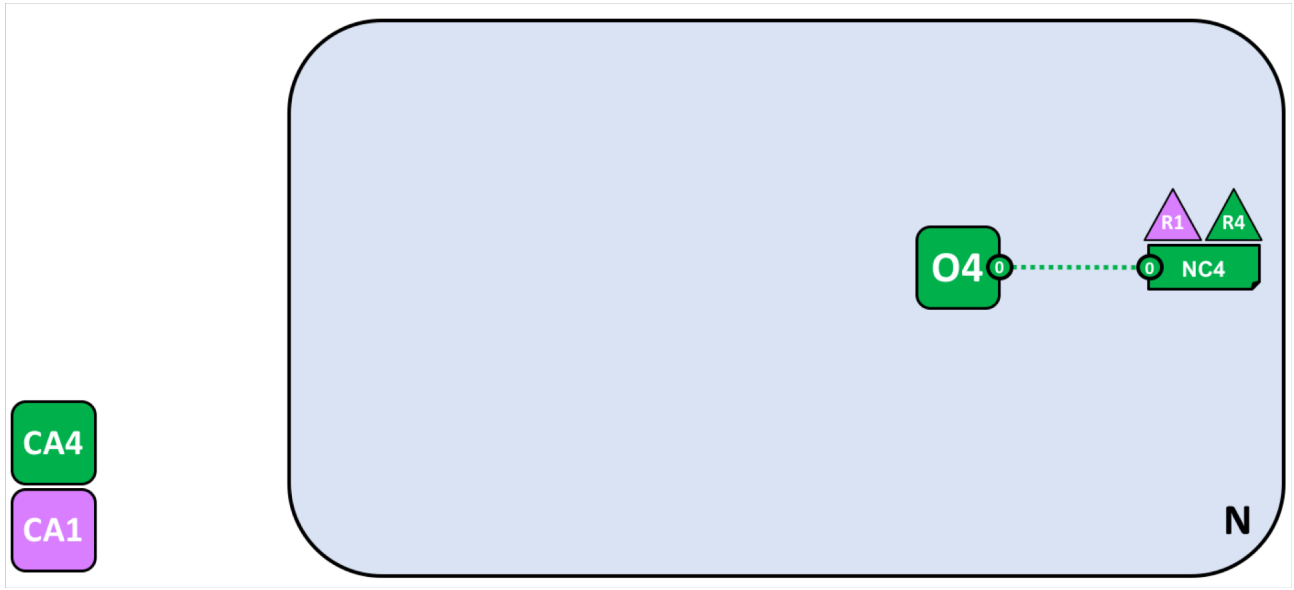
Fabric network: Membership Service Provider



The figure above shows how a local MSP is stored on a local filesystem

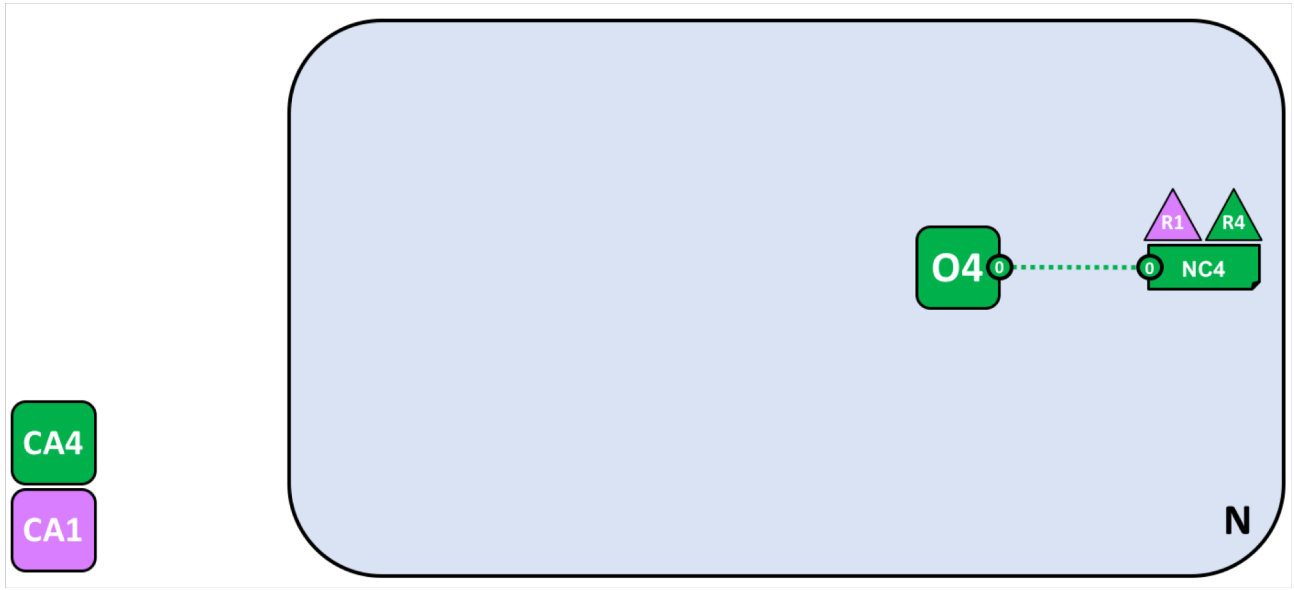
Fabric network: Membership Service Provider

- A Membership Service Provider (MSP) in Fabric identifies which Root CAs and Intermediate CAs are trusted to define the members of an organisation
- It does so either by:
  - listing the identities of their members, or
  - identifying which CAs are authorised to issue valid identities for their members



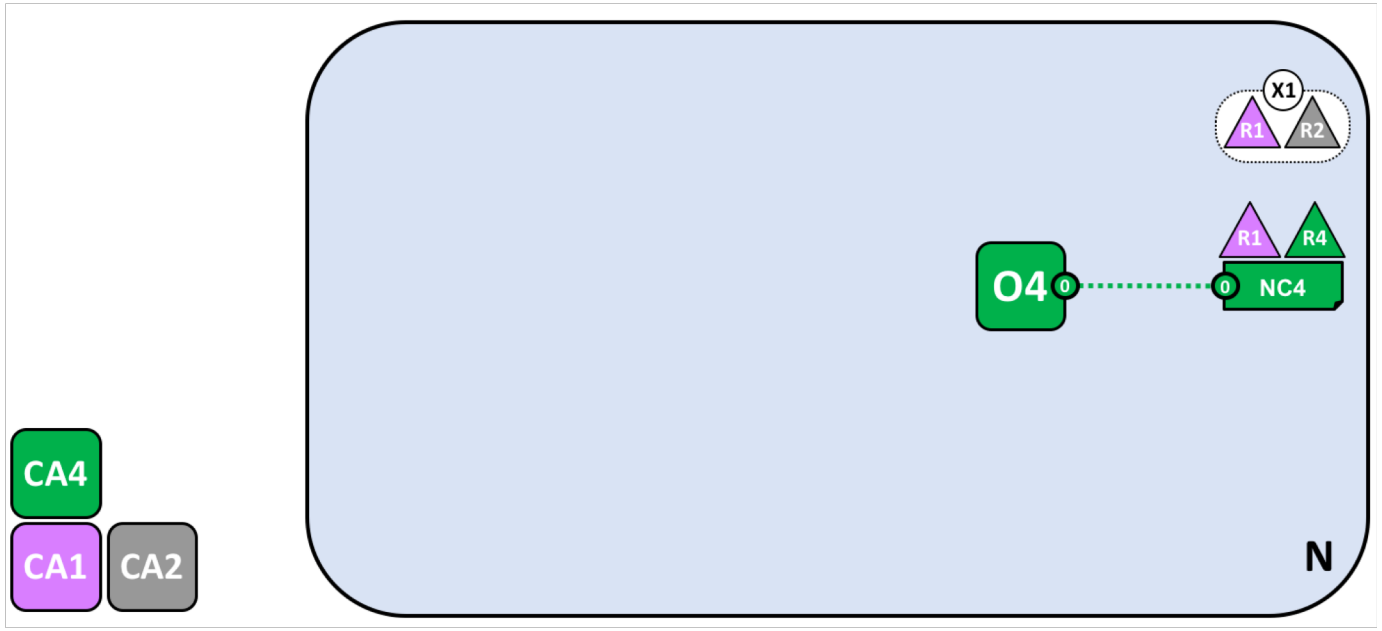
Fabric network

- In this next phase, we are going to allow organisation **R1** users to administer the network
- R4 updates the network configuration to make R1 an administrator too
- After this point R1 and R4 have equal rights over the network configuration
- Also notice that certificate authority CA1 has been added
  - it can be used to identify users from the R1



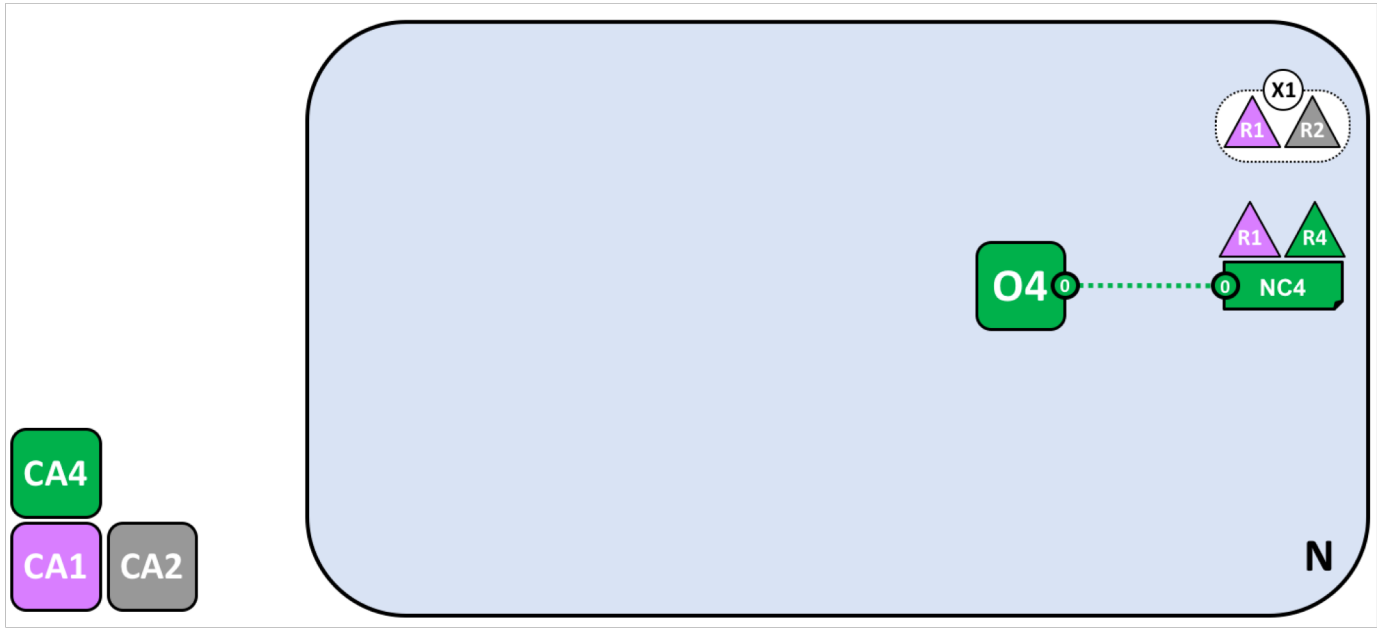
Fabric network

- Next, define a consortium
- This word literally means “a group with a shared destiny”, e.g. trying to utilise the blockchain for a particular use case
- To enable this, we need to combine a set of organisations in our blockchain network



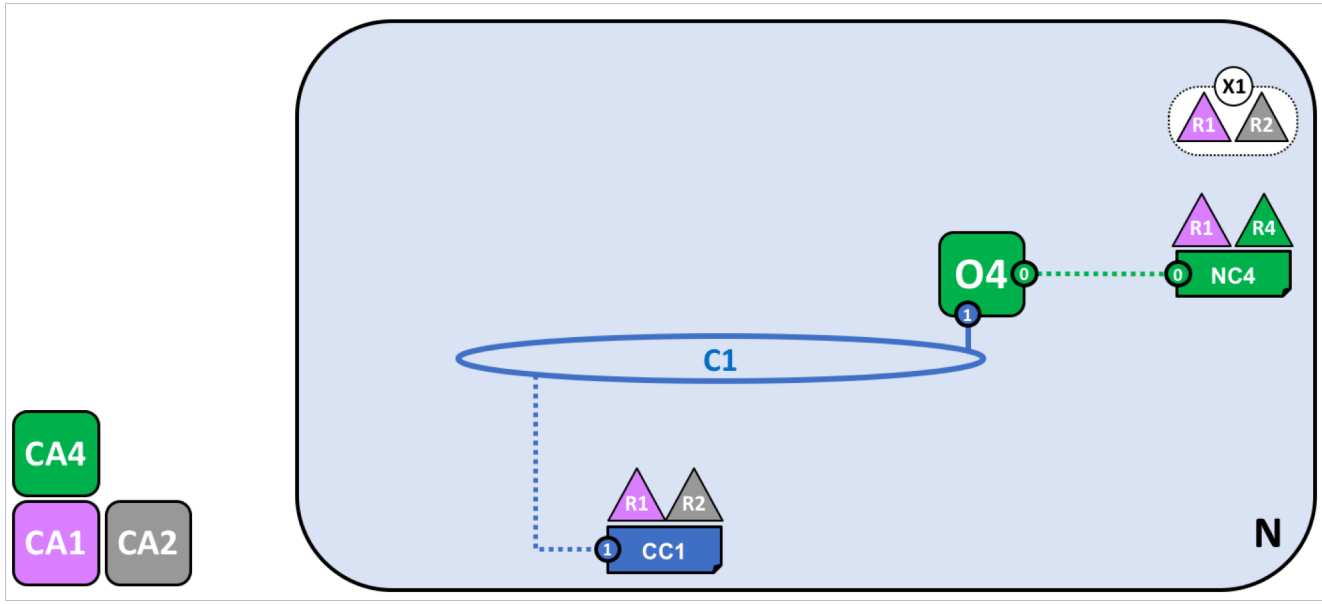
Fabric network

- A network administrator defines a consortium X1 that contains two members
  - the organisations R1 and R2
- This consortium definition is stored in the network configuration NC4
- Because of the way NC4 is configured, only R1 or R4 can create a new consortia
- CA1 and CA2 are the respective CAs for these organisations



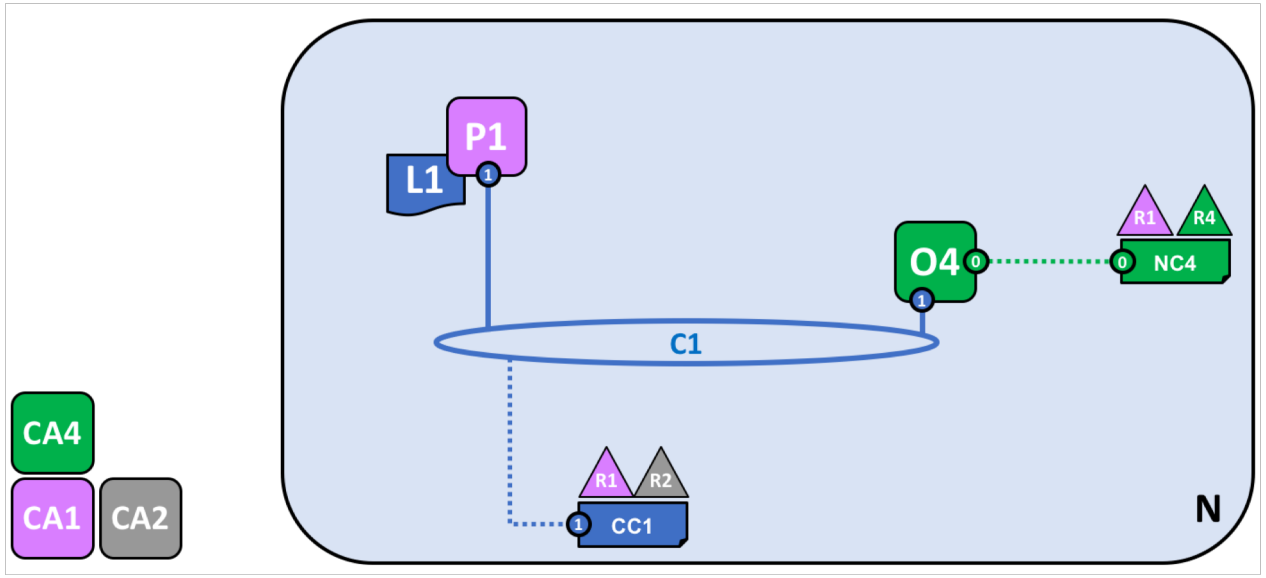
Fabric network

- A channel is a primary communications mechanism by which the members of a consortium can communicate with each other
- A channel **C1** has been created for R1 and R2 using the consortium definition **X1**
- The channel is governed by a channel configuration **CC1**, completely separate to the network configuration
- **CC1** is managed by R1 and R2 who have equal rights over C1
- R4 has no rights in CC1 whatsoever
  - As it is not a part of the consortium



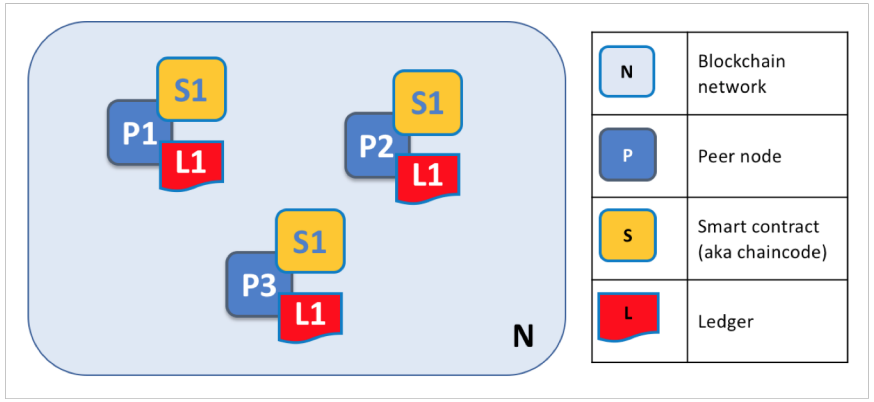
Fabric network

- Next, a peer node P1 has joined the channel C1
- P1 physically hosts a copy of the ledger L1
- P1 and O4 can communicate with each other using channel C1



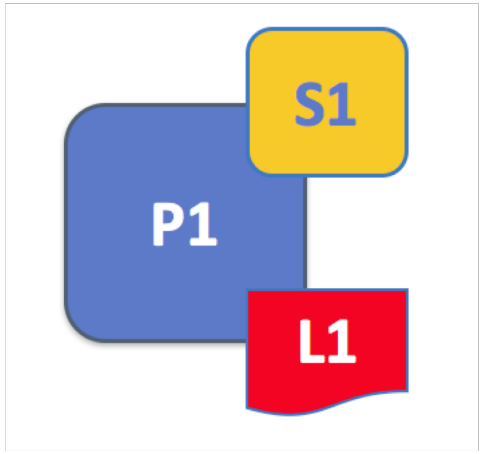
Fabric network: peers

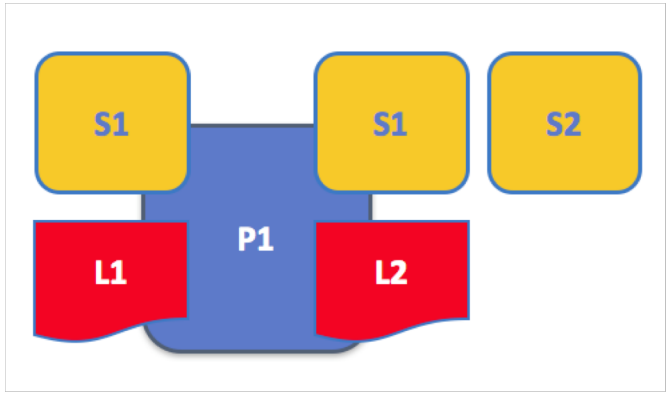
- Next, we will add Peers in the network
- A blockchain network is comprised primarily of a set of peer nodes (or, simply, peers)
- Peers are a fundamental element of the network because they host ledgers and smart contracts
- In this example, the network N consists of peers P1, P2 and P3, each of which maintain their own instance of the distributed ledger L1
- P1, P2 and P3 use the same chaincode, S1, to access their copy of that distributed ledger



Fabric network: peers

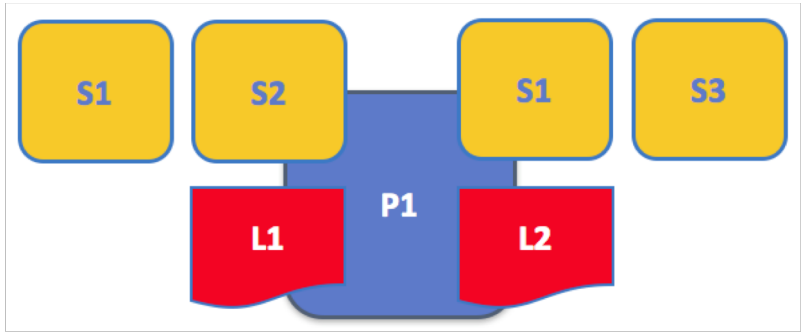
- In this example, P1 hosts an instance of ledger L1 and an instance of chaincode S1
- There can be many ledgers and chaincodes hosted on an individual peer
- In this example, we can see that the peer P1 hosts ledgers L1 and L2
- Ledger L1 is accessed using chaincode S1
- Ledger L2 on the other hand can be accessed using chaincodes S1 and S2





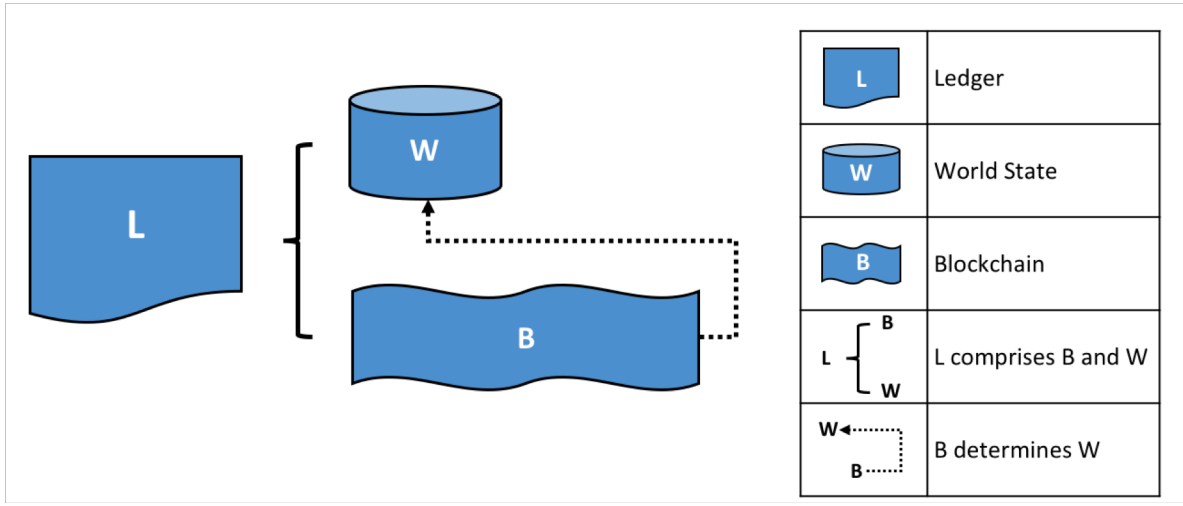
Fabric network: peers

- Each ledger can have many chaincodes which access it
- In this example, we can see that peer P1 hosts ledgers L1 and L2, where L1 is accessed by chaincodes S1 and S2, and L2 is accessed by S1 and S3
- We can see that S1 can access both L1 and L2



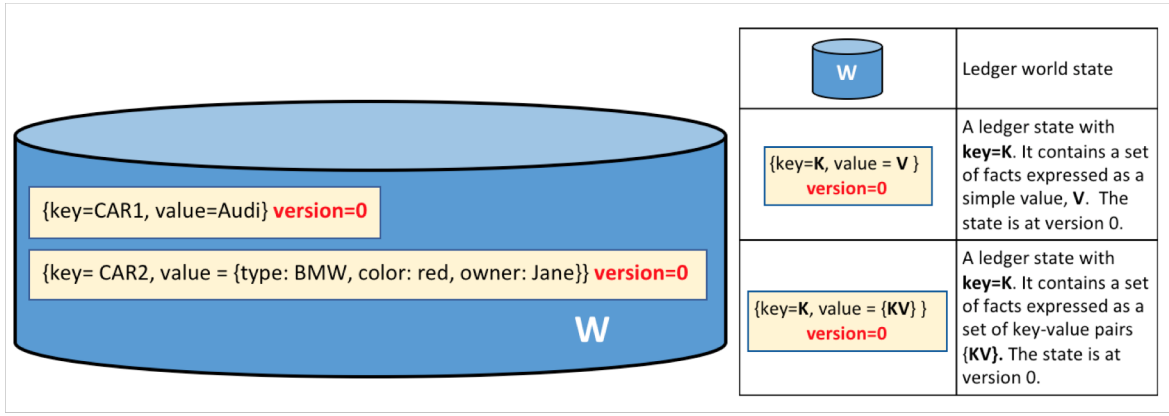
Fabric network: ledger

- In Hyperledger Fabric, a ledger consists of two distinct, though related, parts
  - a world state and a blockchain
- Each of these represents a set of facts about a set of business objects
- A **world state** is a database that holds a cache of the **current values** of a set of ledger states
- The world state makes it easy for a program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log
- Ledger states are, by default, expressed as **key-value** pairs



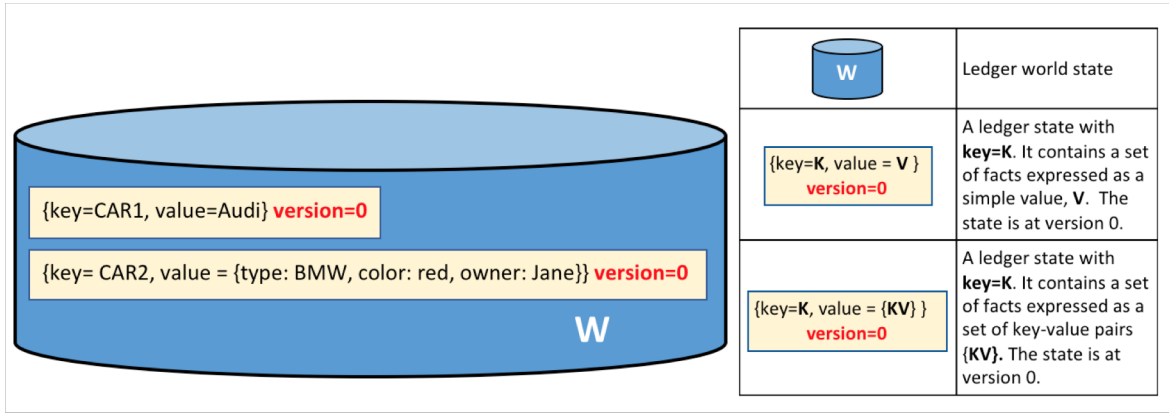
Fabric network: ledger

- A ledger world state containing two states
- The first state is: key=CAR1 and value=Audi
  - Here, the value is simple
- The second state has a more complex value:
  - key=CAR2 and value={model:BMW, color=red, owner=Jane}
- Both states are at version 0
- The version number is incremented as the states are updated



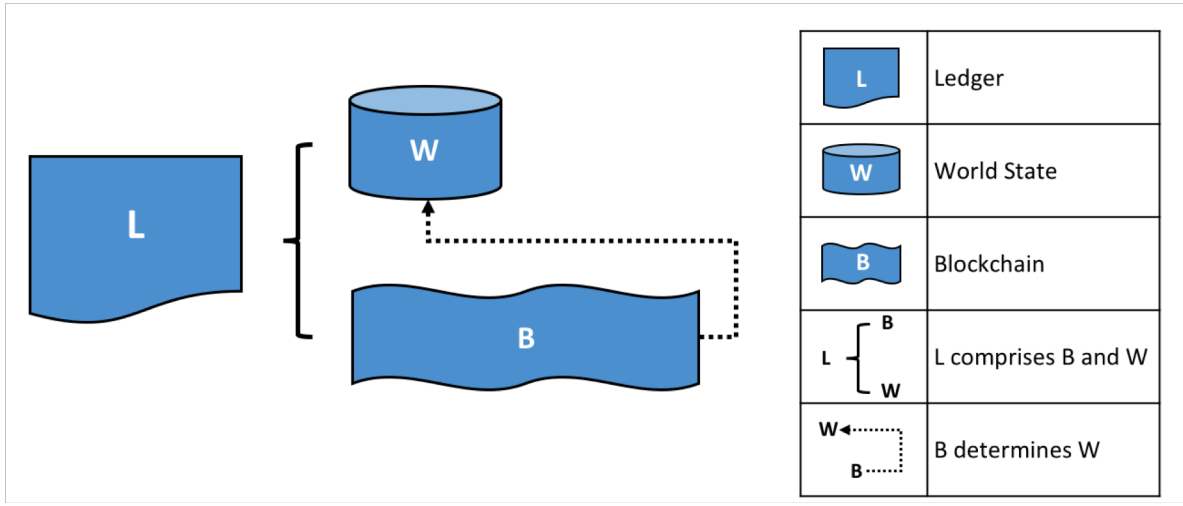
Fabric network: ledger

- At the initial stage, when a ledger is first created, the world state is empty
- When a transaction is created and then is then recorded in the blockchain
  - The world state is updated and recorded in the database



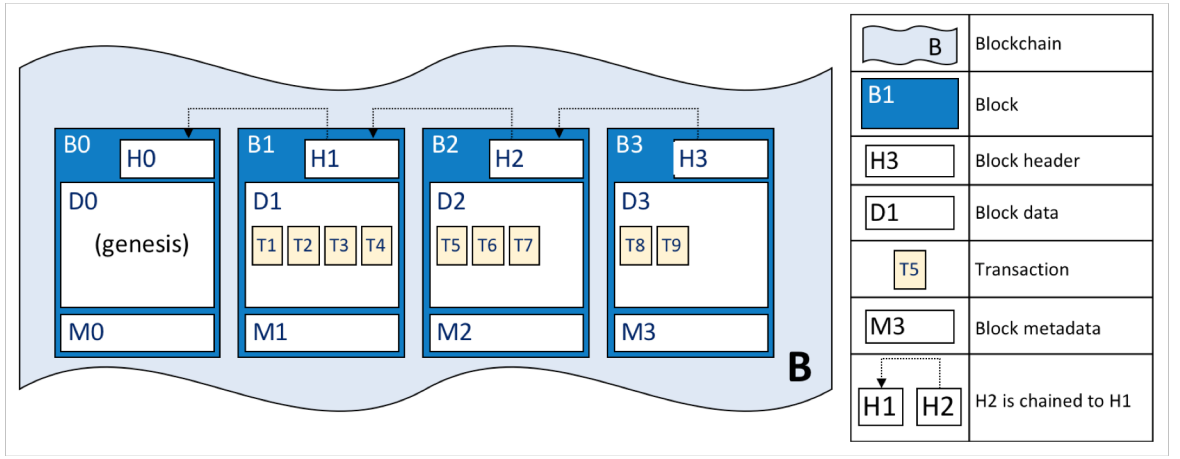
Fabric network: ledger

- The second part is the **blockchain**
- A blockchain is a transaction log that records all the changes that have resulted in the current the world state
- Transactions are collected inside blocks that are appended to the blockchain
- This enables you to understand the history of changes that have resulted in the current world state
- The blockchain data structure is very different to the world state because once written, it cannot be modified
  - it is **immutable**
- We can also say that world state W is derived from blockchain B



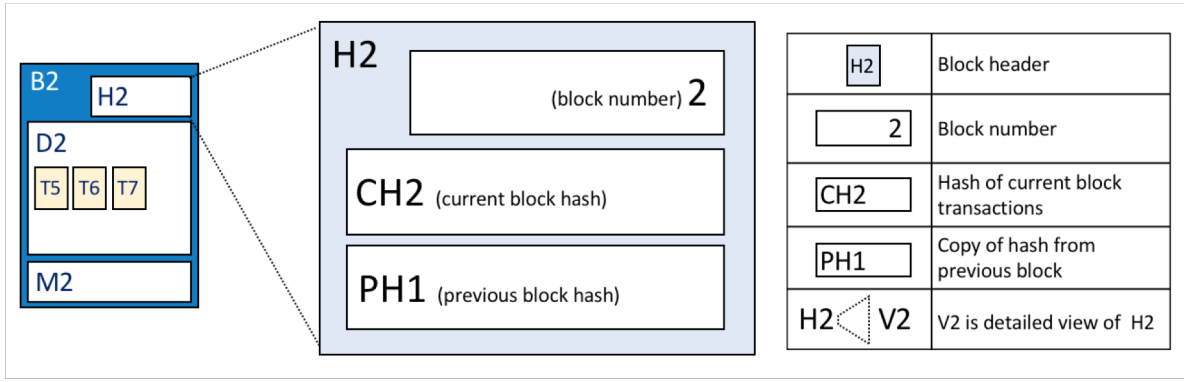
Fabric network: ledger

- A blockchain B containing blocks B0, B1, B2, B3
- B0 is the first block in the blockchain, the genesis block
- We can see that block B2 has a block data D2 which contains all its transactions: T5, T6, T7
- B2 has a block header H2
  - which contains a cryptographic hash of all the transactions in D2 as well as with the equivalent hash from the previous block B1
- This creates the chain among the blocks



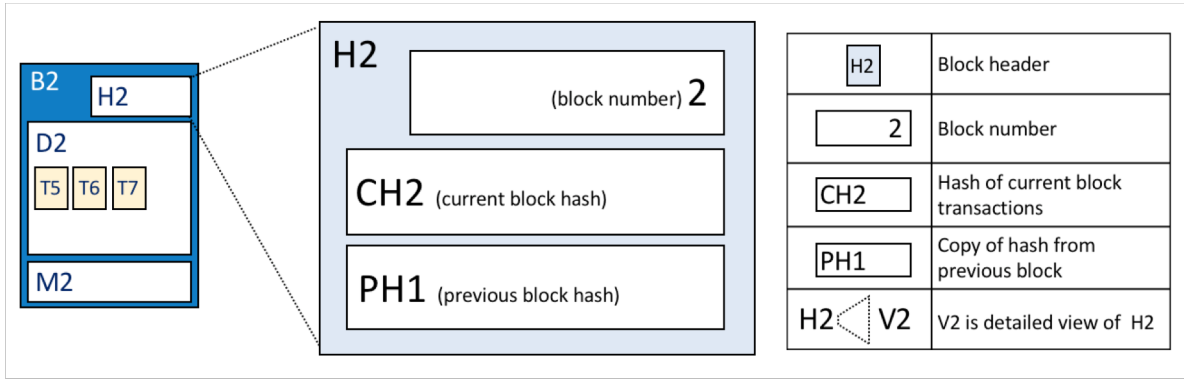
Fabric network: ledger

- Block Header: This section comprises three fields, written when a block is created
  - Block number: An integer starting at 0 (the genesis block), and increased by 1 for every new block appended to the blockchain
  - Current Block Hash: The hash of all the transactions contained in the current block
  - Previous Block Hash: A copy of the hash from the previous block in the blockchain
- These fields are internally derived by cryptographically hashing the block data
- They ensure that each and every block is inextricably linked to its neighbour, leading to an immutable ledger



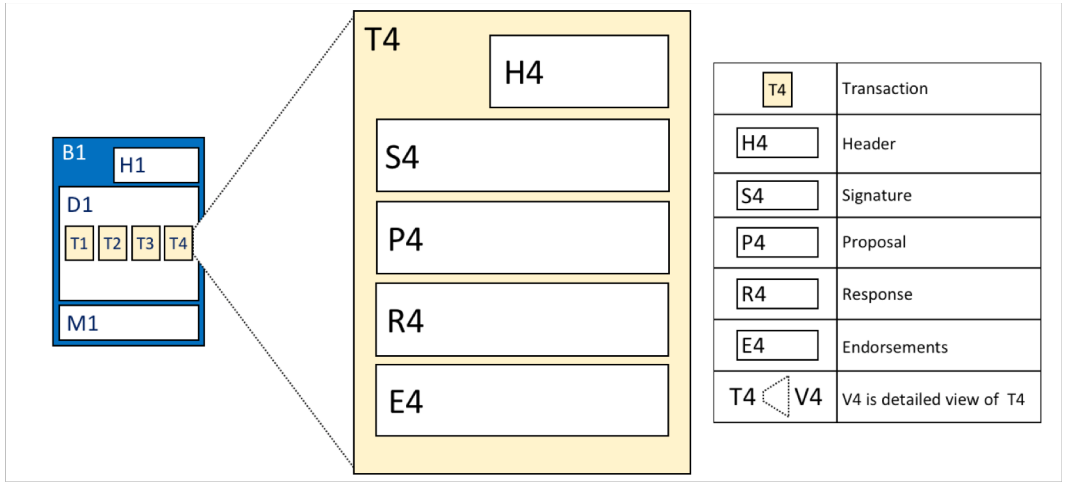
Fabric network: ledger

- Block Data
  - This section contains a list of transactions arranged in order
  - It is written when the block is created by the ordering service
- Block Metadata
  - This section contains the time when the block was written, as well as the certificate, public key and signature of the block writer
- Such metadata also contains a valid/invalid indicator for every transaction



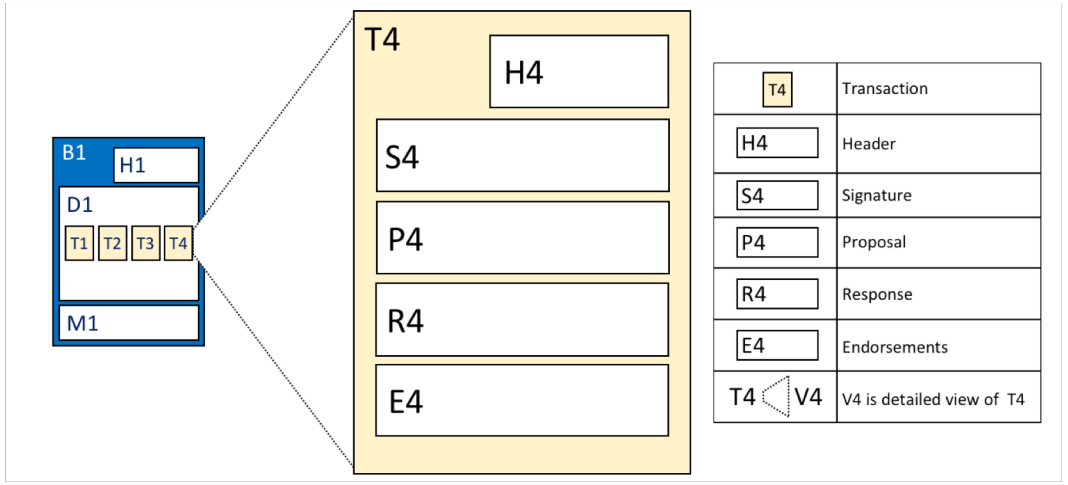
Fabric network: ledger

- Header: This section, illustrated by H4, captures some essential metadata about the transaction
  - for example, the name of the relevant chaincode, and its version
- Signature: This section, illustrated by S4, contains a cryptographic signature, created by the client application
  - This field is used to check that the transaction details have not been tampered with, as it requires the application's private key to generate it
- Proposal: This field, illustrated by P4, encodes the input parameters supplied by an application to the smart contract which creates the proposed ledger update

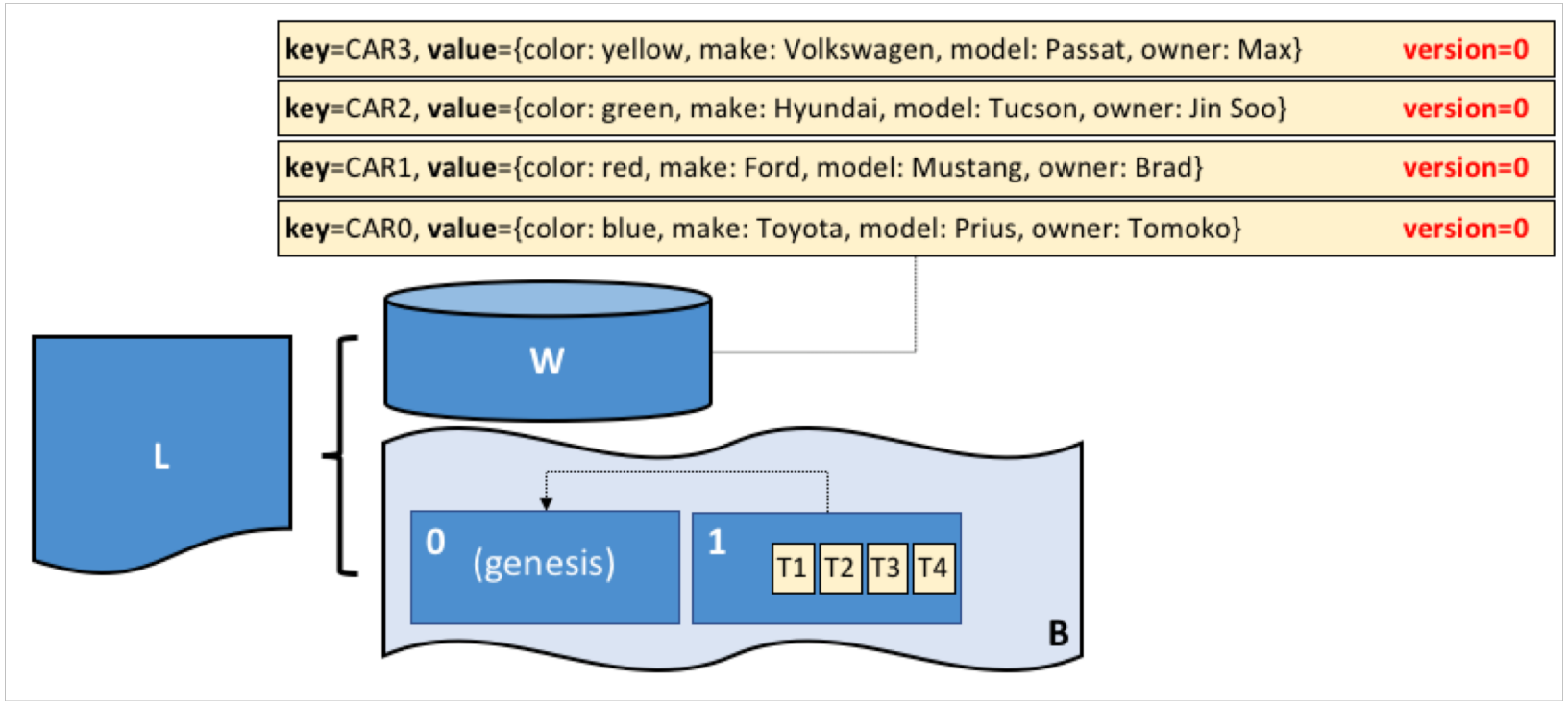


Fabric network: ledger

- Response: This section, illustrated by R4, captures the before and after values of the world state, as a Read Write set (RW-set)
  - It's the output of a smart contract
- If the transaction is successfully validated, it will be applied to the ledger to update the world state
- Endorsements: As shown in E4, this is a list of signed transaction responses from each required organisation sufficient to satisfy the endorsement policy (explained later)

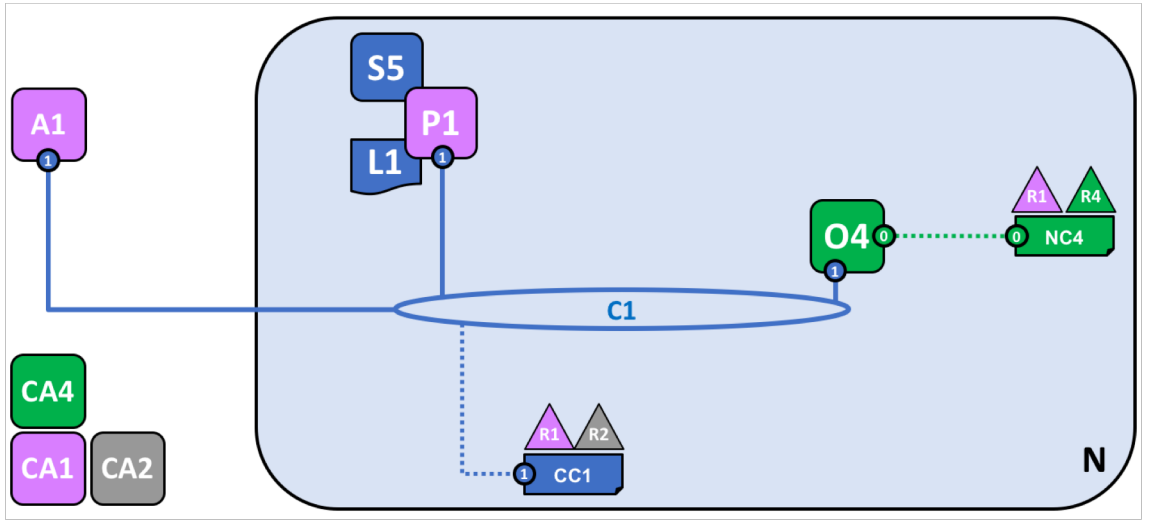


Fabric network: ledger



Fabric network

- Next, we add client applications to consume some of the services provided by workhorse of the ledger, the peer!
- For this, a smart contract S5 has been installed onto P1
- Client application A1 in organisation R1 can use S5 to access the ledger L1 via peer node P1
- A1, P1 and O4 are all joined to channel C1, i.e. they can all make use of the communication facilities provided by that channel



Fabric network: Installing a smart contract

- After a smart contract S5 has been developed, an administrator in organisation R1 must install it onto peer node P1
- This is a straightforward operation; after it has occurred, P1 has full knowledge of S5
- Specifically, P1 can see the implementation logic of S5
  - the program code that it uses to access the ledger L1

Fabric network: Instantiation of a smart contract

- However, just because P1 has installed S5, the other components connected to channel C1 are unaware of it
  - it must first be instantiated on channel C1
- In our example, which only has a single peer node P1, an administrator in organisation R1 must instantiate S5 on channel C1 using P1
- After instantiation, every component on channel C1 is aware of the existence of S5
- In our example it means that S5 can now be invoked by client application A1!

Fabric network: Endorsement policy

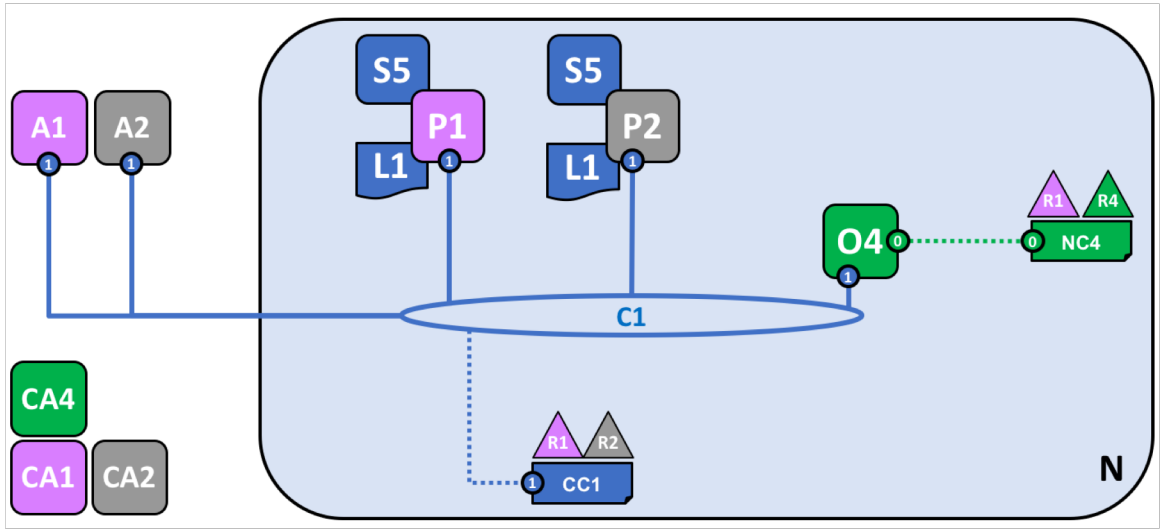
- The most important piece of additional information supplied at instantiation is an endorsement policy
- It describes which organisations must approve transactions before they will be accepted by other organisations onto their copy of the ledger
- In our sample network, transactions can be only be accepted onto ledger L1 if R1 or R2 endorse them

Fabric network: Endorsement policy

- Once a smart contract has been installed on a peer node and instantiated on a channel it can be invoked by a client application
- Client applications do this by sending transaction proposals to peers owned by the organisations specified by the smart contract endorsement policy
- The transaction proposal serves as input to the smart contract, which uses it to generate an endorsed transaction response
- The response is returned by the peer node to the client application

Fabric network

- The network has grown through the addition of infrastructure from organisation R2
- Specifically, R2 has added peer node P2, which hosts a copy of ledger L1, and chaincode S5
- P2 has also joined channel C1, has application A2
- A2 and P2 are identified using certificates from CA2
- All of this means that both applications A1 and A2 can invoke S5 on C1 either using peer node P1 or P2



Fabric network: Peer roles

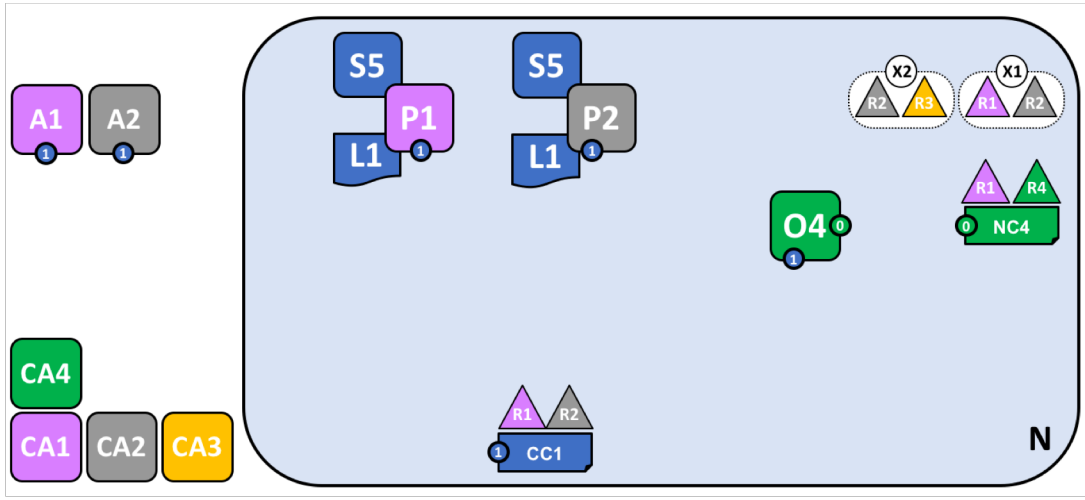
- Each peer can take two roles:
- *Committing peer:*
  - Every peer node in a channel is a committing peer
  - It receives blocks of generated transactions, which are subsequently validated before they are committed to the peer node's copy of the ledger as an append operation
- *Endorsing peer:* Every peer with a smart contract *can* be an endorsing peer if it has a smart contract installed
  - However, to actually *be* an endorsing peer, the smart contract on the peer must be used by a client application to generate a digitally signed transaction response
  - The term *endorsing peer* is an explicit reference to this fact

Fabric network: Peer roles

- Each peer can be of two types:
- *Leader peer*
  - When an organisation has multiple peers in a channel, a leader peer is a node which takes responsibility for distributing transactions from the orderer to the other committing peers in the organisation
  - A peer can choose to participate in static or dynamic leadership selection
- *Anchor peer*
  - If a peer needs to communicate with a peer in another organisation, then it can use one of the **anchor peers** defined in the channel configuration for that organisation
  - An organisation can have zero or more anchor peers defined for it, and an anchor peer can help with many different cross-organisation communication scenarios

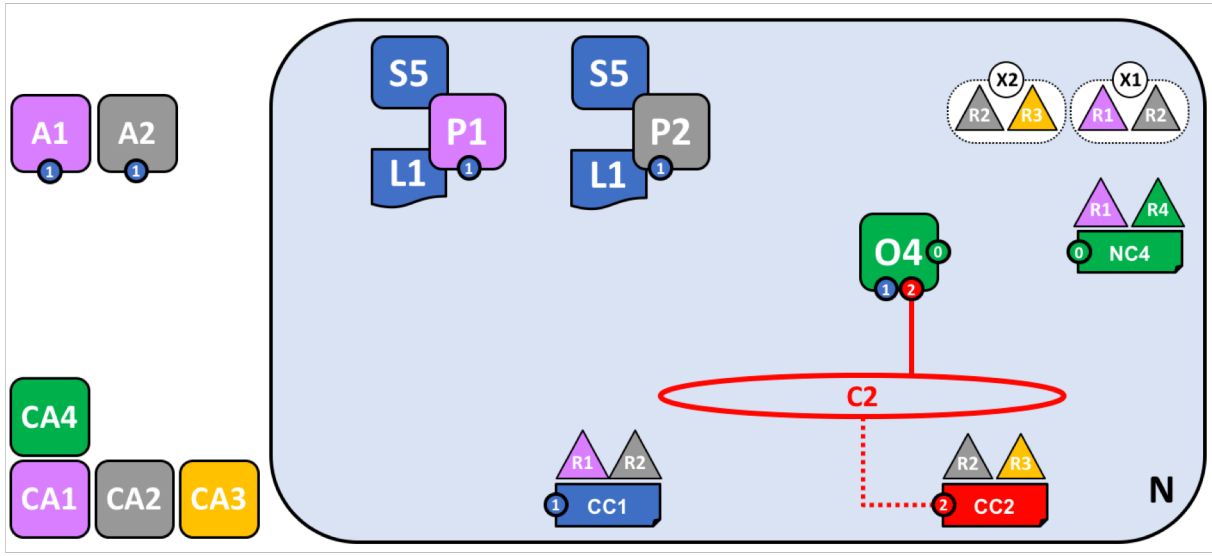
Fabric network

- In this next phase of network development, we introduce organisation **R3**
- A network administrator from organisation R1 or R4 has added a new consortium definition, X2, which includes organisations R2 and R3
- We’re going to give organisations R2 and R3 a separate application channel for the new consortium which allows them to transact with each other
- This application channel will be completely separate to that previously defined, so that R2 and R3 transactions can be kept private to them



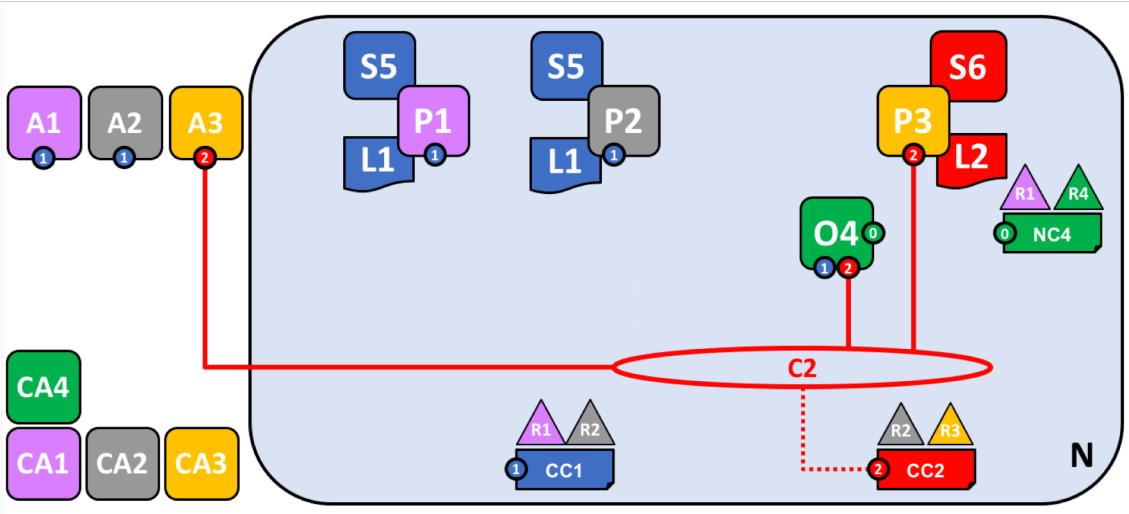
Fabric network

- A new channel C2 has been created for R2 and R3 using consortium definition X2
- The channel has a channel configuration CC2, completely separate to the network configuration NC4, and the channel configuration CC2
- Channel C2 is managed by R2 and R3 who have equal rights over C2 as defined by a policy in CC2
- R1 and R4 have no rights defined in CC2 whatsoever



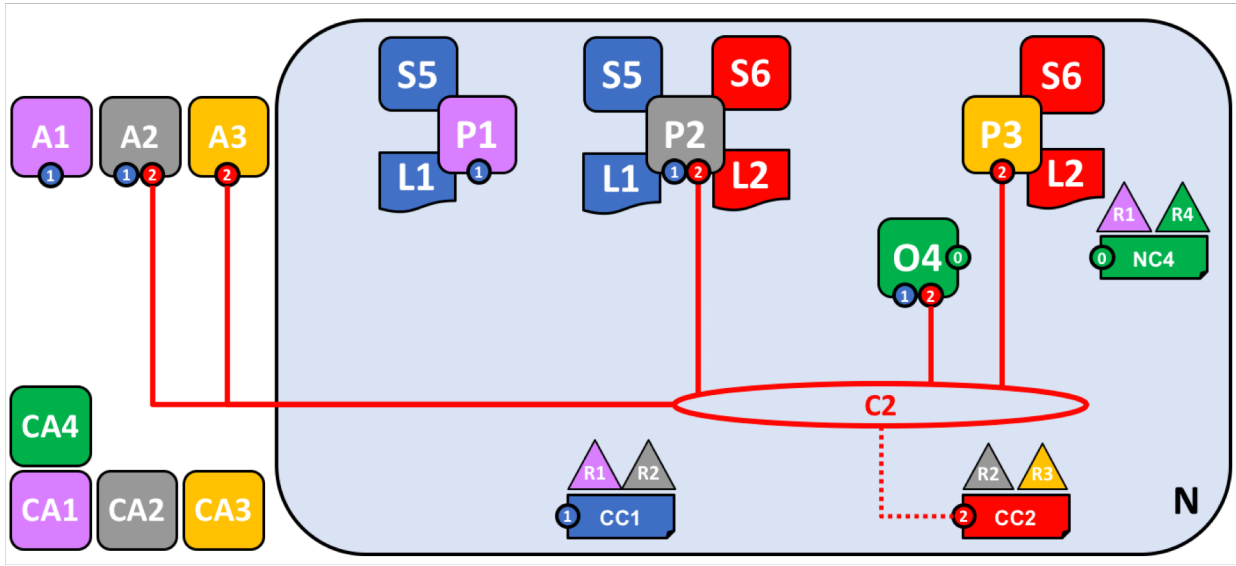
Fabric network

- Peer P3 and its corresponding application A3 have been added to the network
- Client applications A1 and A2 can use channel C1 for communication with peers P1 and P2, and ordering service O4
- Client applications A3 can use channel C2 for communication with peer P3 and ordering service O4
- Ordering service O4 can make use of the communication services of channels C1 and C2
- Channel configuration CC1 applies to channel C1, CC2 applies to channel C2



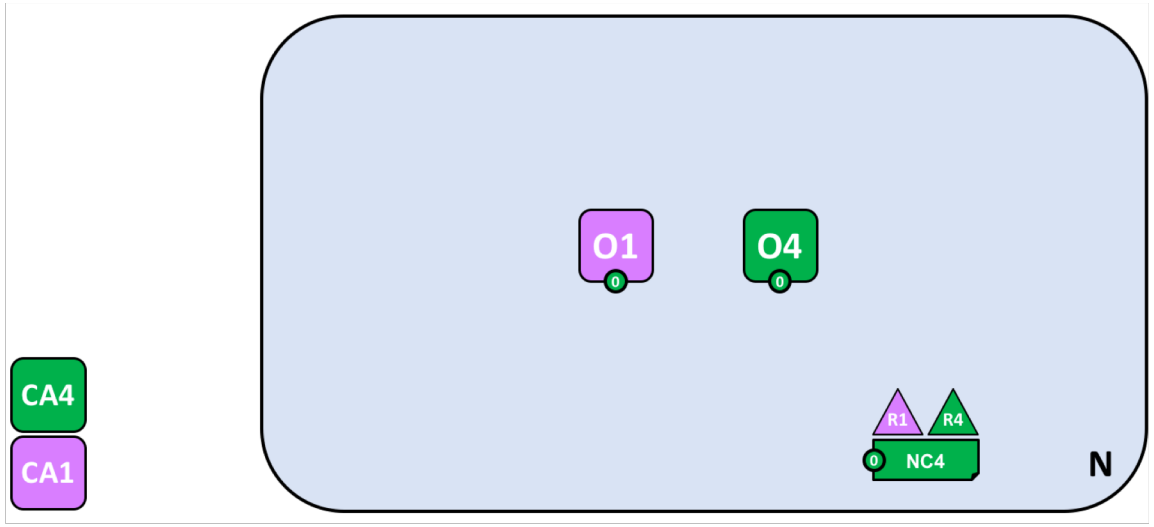
Fabric network

- P2 and A2 have also been added to C2
- Client applications A1 can use channel C1 for communication with peers P1 and P2, and ordering service O4
- Client application A2 can use channel C1 for communication with peers P1 and P2 and channel C2 for communication with peers P2 and P3 and ordering service O4
- Client application A3 can use channel C2 for communication with peer P3 and P2 and ordering service O4
- Ordering service O4 can make use of the communication services of channels C1 and C2
- Channel configuration CC1 applies to channel C1, CC2 applies to channel C2



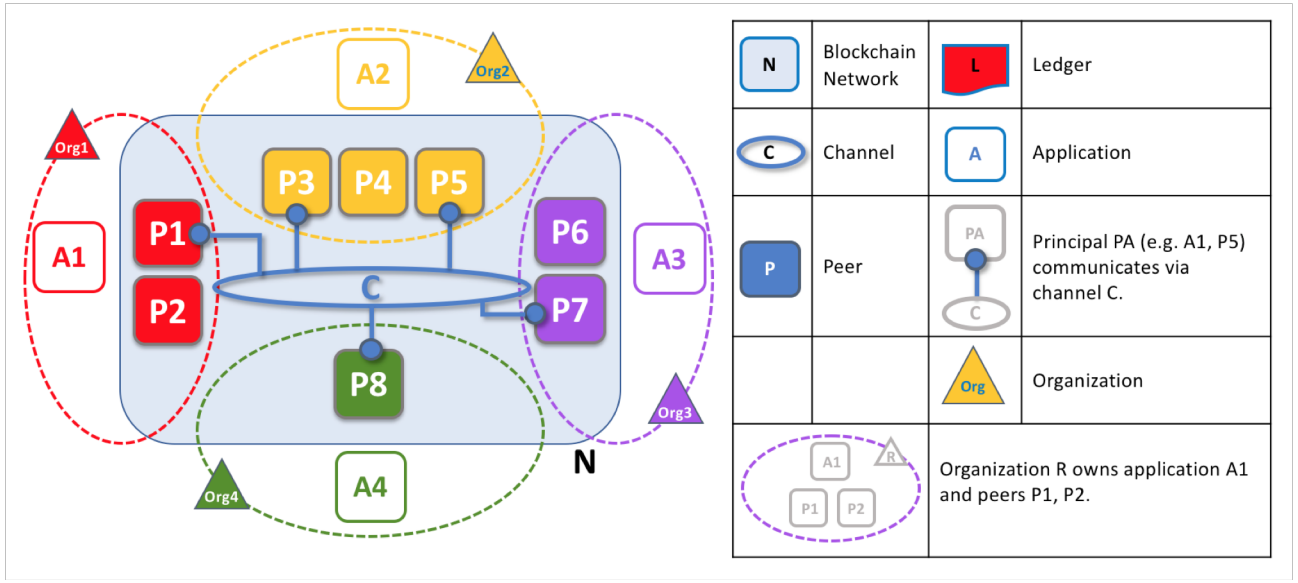
Fabric network

- A multi-organisation ordering service
- The ordering service comprises ordering service nodes O1 and O4
- O1 is provided by organisation R1 and node O4 is provided by organisation R4
- The network configuration NC4 defines network resource permissions for actors from both organisations R1 and R4



Fabric network

- In this example, we see four organisations contributing eight peers to form a network
- The channel C connects five of these peers in the network N — P1, P3, P5, P7 and P8
- The other peers owned by these organisations have not been joined to this channel, but are typically joined to at least one other channel
- Applications that have been developed by a particular organisation will connect to their own organisation's peers as well as those of different organisations
- For simplicity, an orderer node is not shown in this diagram

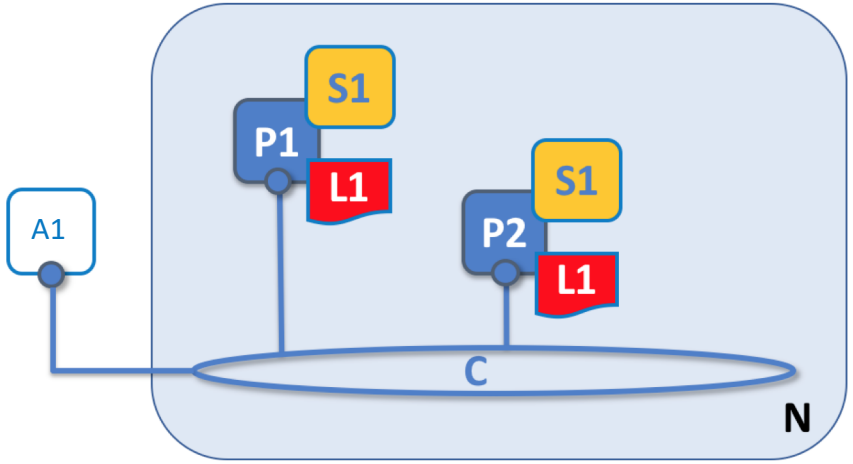


Peers in a blockchain network with multiple organisations

Fabric protocol flow

- The scenario includes two clients, A and B, who are buying and selling radishes
- They each have a peer (Peer A and Peer B respectively) on the network through which they send their transactions and interact with the ledger

Fabric protocol flow



N	Blockchain Network	L	Ledger
C	Channel	A	Application
P	Peer	PA / C	Principal PA (e.g. A, P1) communicates via channel C.
S	Chaincode		

Fabric protocol flow

- Assumption:
  - A channel is set up and running
  - The application user has registered and enrolled with the organisation's certificate authority (CA) and received back necessary cryptographic material, which is used to authenticate to the network
  - The chaincode is installed on the peers and instantiated on the channel
  - An endorsement policy has also been set for this chaincode, stating that both Peer A and Peer B must endorse any transaction

Fabric protocol flow

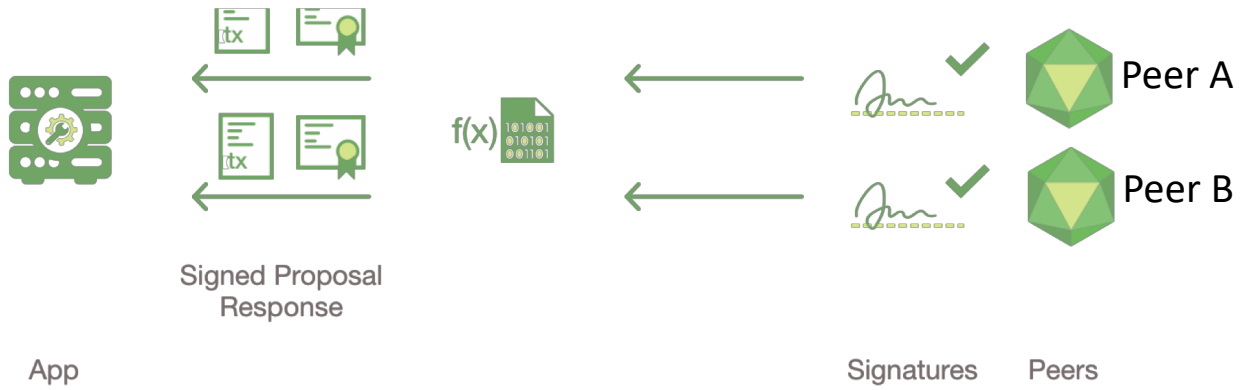
- Client A is sending a request to purchase radishes
- The request targets Peer A and Peer B
  - who are respectively representative of Client A and Client B
- The endorsement policy states that both peers must endorse any transaction, therefore the request goes to Peer A and Peer B

Fabric protocol flow

- Next, the transaction proposal is constructed
- An application leveraging a supported SDK (Node, Java, Python) utilises one of the available APIs which generates a transaction proposal
- The proposal is a request to invoke a chaincode function so that data can be read and/or written to the ledger (i.e. write new key value pairs for the assets)
- The SDK creates the transaction proposal in a proper format and takes the user's cryptographic credentials to produce a unique signature for this transaction proposal

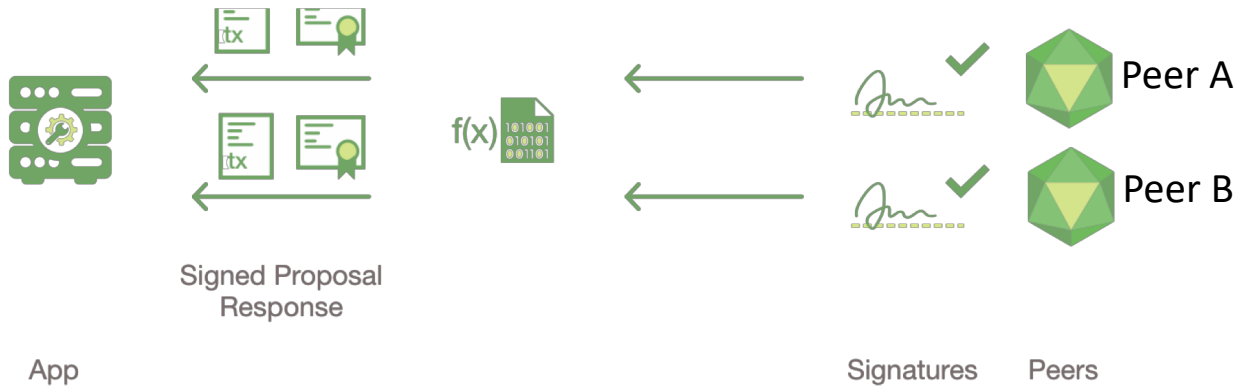
Fabric protocol flow

- The endorsing peers verify
  i. that the transaction proposal is well formed
  ii. it has not been submitted already in the past (replay-attack protection),
  iii. the signature is valid (using MSP), and
  iv. that the submitter (Client A, in the example) is properly authorised to perform the proposed operation on that channel (namely, each endorsing peer ensures that the submitter satisfies the channel's *endorsing* policy)



Fabric protocol flow

- The endorsing peers take the transaction proposal inputs as arguments to the invoked chaincode's function
- The chaincode is then executed against the current state database to produce transaction results including a response value, read set, and write set
- No updates are made to the ledger at this point
- The set of these values, along with the endorsing peer's signature is passed back as a "proposal response" to the SDK and the application which parses the payload



Fabric protocol flow

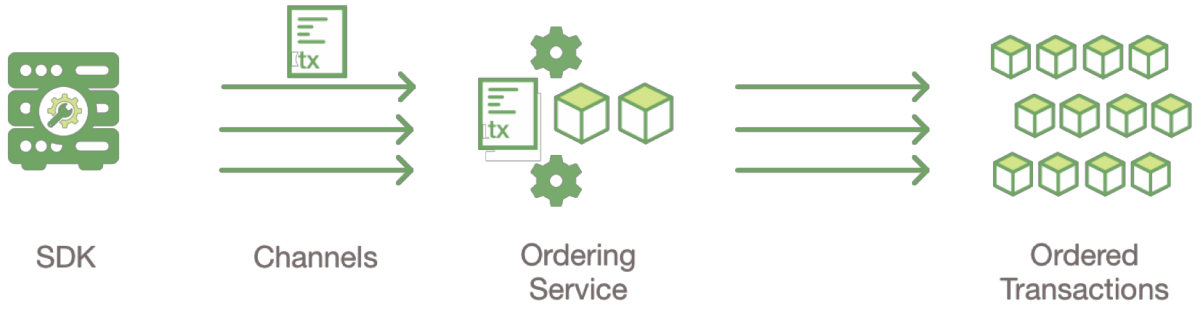
- The application verifies the endorsing peer signatures and compares the proposal responses to determine if the proposal responses are the same
- If the chaincode only queried the ledger
  - the application would inspect the query response and would typically not submit the transaction to Ordering Service

Fabric protocol flow

- If the client application intends to submit the transaction to Ordering Service to update the ledger, the application determines if the specified endorsement policy has been fulfilled before submitting (i.e. did both Peer A and Peer B both endorse)
- If both did not endorse, the transaction request will be discarded
- This checking at the application is optional
- Since the endorsement policy will still be enforced by peers and upheld at the commit validation phase

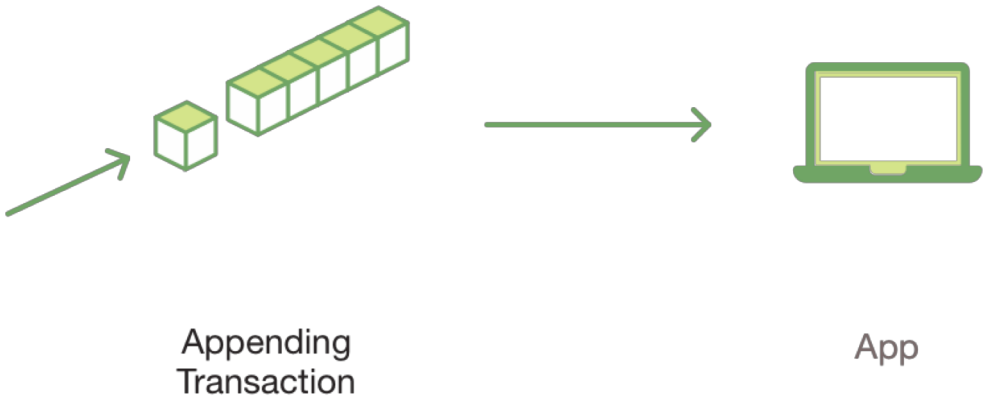
Fabric protocol flow

- The application “broadcasts” the transaction proposal and response within a “transaction message” to the Ordering Service
- The transaction will contain the read/write sets, the endorsing peers signatures and the Channel ID
- The Ordering Service does not need to inspect the entire content of a transaction in order to perform its operation
- It simply receives transactions from all channels in the network, orders them chronologically by channel, and creates blocks of transactions per channel

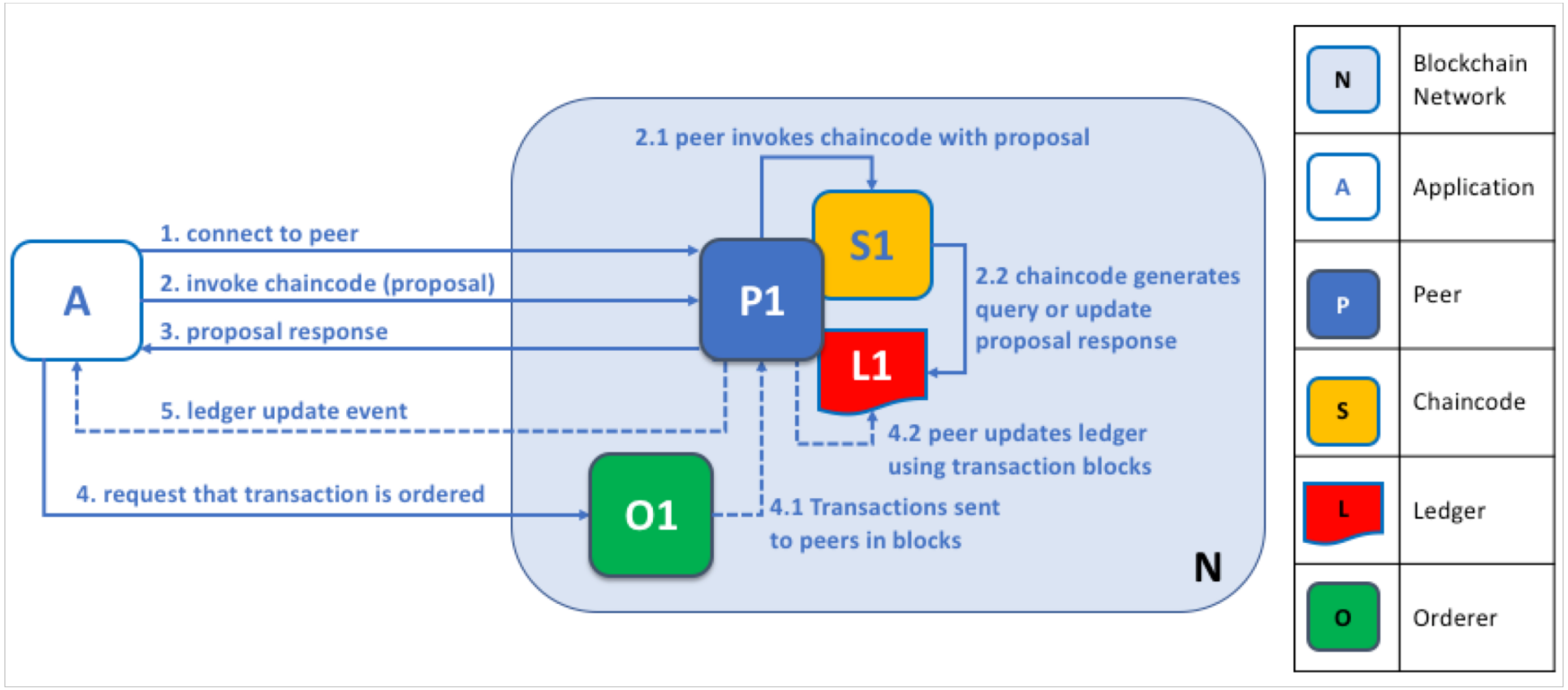


Fabric protocol flow

- Each peer appends the block to the channel's chain, and for each valid transaction the write sets are committed to current state database
- An event is emitted, to notify the client application that the transaction (invocation) has been immutably appended to the chain
  - as well as notification of whether the transaction was validated or invalidated



Fabric protocol flow



References

- https://www.hyperledger.org/
- https://hyperledger-fabric.readthedocs.io/en/release-1.4/