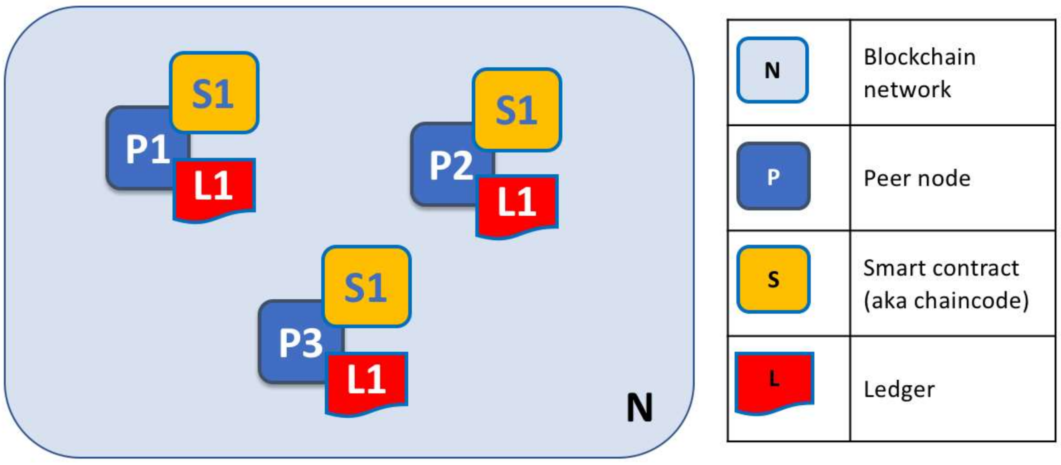
Peers

A fundamental element of a Hyperledger Fabric blockchain network is the set of *peer nodes* (or, simply, *peers*). Peers are fundamental because they manage ledgers and smart contracts. Starting in Hyperledger Fabric v2.4, peers also manage transaction proposals and endorsements by running the Fabric Gateway service. Recall that a ledger immutably records all of the transactions generated by smart contracts (which in Hyperledger Fabric are contained in *chaincode*, more on this later) and endorsed by the required organizations. Smart contracts and ledgers encapsulate the *processes* and *information*, respectively, that are shared by channel peers. These aspects of peers make them a good starting point for understanding a Fabric network.

Besides peers, other elements of a Fabric blockchain network are also important: ledgers and smart contracts, orderers, policies, channels, client applications, organizations, identities, and membership, which you can read about in their own dedicated sections. This section focuses on peers, and their relationships to these other elements in a Fabric network.



A Fabric blockchain network (above) is comprised of peers (non-ordering nodes), each of which stores and manages copies of ledgers and smart contracts. In this example, the Fabric network N consists of peers P1, P2 and P3, each of which maintains its own instance of the distributed ledger L1. P1, P2 and P3 each invoke the same chaincode, S1, to access their respective copies of the distributed ledger.

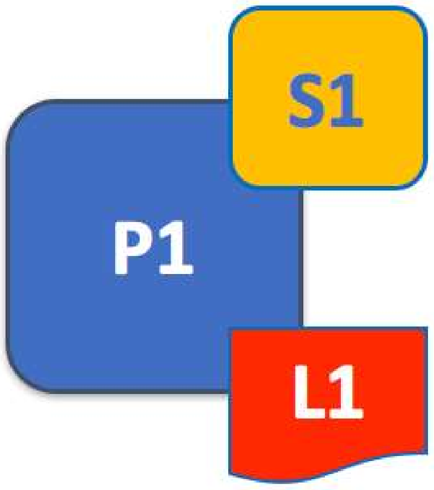
Peers are a flexible and redundant element that can be created, started, stopped, reconfigured and deleted. Peers expose a set of APIs that enable client applications to interact with the services that peers provide — the Fabric Gateway service in particular.

Chaincode terminology

Fabric implements **smart contracts** through a type of logic called **chaincode** — code that accesses the ledger — which is written using the chaincode APIs. In this topic, we use the term **chaincode**, but feel free to interpret chaincode as a **smart contract** if that concept is more familiar. To learn more about chaincode and smart contracts, check out the documentation on smart contracts and chaincode.

Ledgers and Chaincode

Now let's look at a peer in a little more detail. We can see that it's the peer that hosts both the ledger and chaincode, in addition to services such as Fabric Gateway. More precisely, the peer hosts *instances* of the ledger, and *instances* of chaincode, because blockchain requires consistent replicas of data and smart contracts across peers in a channel. This design provides a deliberate redundancy in a Fabric network — avoiding single points of failure and providing consistent, current ledgers. We'll learn more about the distributed and decentralized nature of a Fabric blockchain network later in this section.

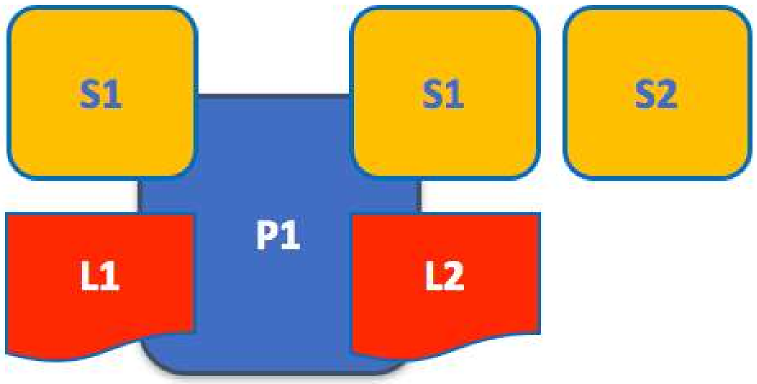


A peer hosts instances of ledgers and instances of chaincodes (above). In this example, P1 hosts an instance of ledger L1 and an instance of chaincode S1. There can be many ledgers and chaincodes hosted on any individual peer.

Because a peer is a *host* for ledgers, chaincodes and services, client applications and administrators must interact with a peer to access these resources. That's why peers are considered the fundamental building blocks of a Fabric network. We'll see later how ledgers get created, and how chaincodes get installed, on peers.

Multiple Ledgers

A peer is capable of hosting more than one ledger, which is useful because it allows for a flexible system design where a single peer can belong to multiple channels in a Fabric network. In the simplest configuration, a peer hosts a single ledger, and therefore belongs to a single channel. But it is not uncommon for a peer to host multiple ledgers because it belongs to multiple channels.

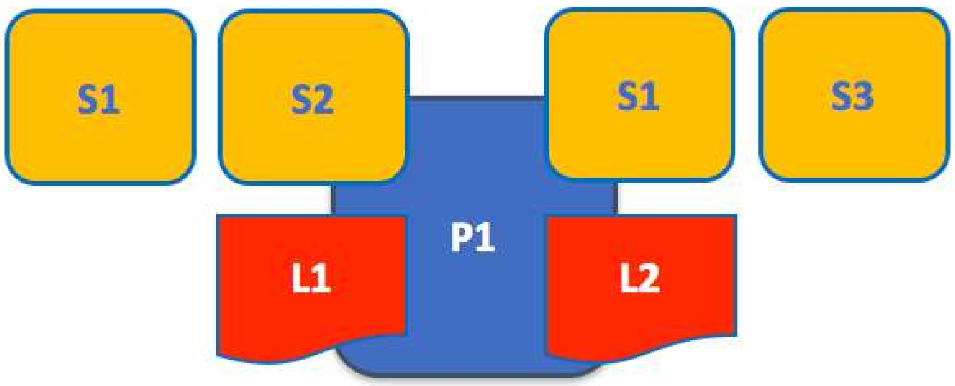


A peer hosting multiple ledgers (above). Peers host one or more ledgers and the chaincodes that access them. In this example, peer P1 hosts ledgers L1 and L2. Ledger L1 is accessed using chaincode S1; Ledger L2 can be accessed using either chaincode S1 or S2.

Chaincodes installed on a peer can query or update (write to) a ledger. Note that peers also host special *system chaincodes* which relate to the overall configuration of the Fabric network.

Multiple Chaincodes

A chaincode is instantiated on a single channel. Each channel (and ledger) can have multiple chaincodes that interact with it.



An example of a peer hosting multiple chaincodes (above). Each ledger can have many chaincodes which access it. In this example, peer P1 hosts ledgers L1 and L2, where L1 is accessed by chaincodes S1 and S2, and L2 is accessed by S1 and S3. S1 can access both L1 and L2.

We'll see a little later why the concept of **channels** in Fabric is important to hosting multiple ledgers and multiple chaincodes on a peer.

Fabric Gateway service

Starting in Hyperledger Fabric v2.4, the Fabric Gateway service is installed and enabled on each peer by default. The gateway service, as opposed to the client application (in Fabric v2.3 and earlier), manages transaction proposals and endorsements on the peer. The Gateway SDKs (v1.0.0 for Go, Node and Java) incorporate this peer-centric model of transaction processing, enabling simplified application development. Client applications developed with Fabric v2.3 or earlier SDKs will continue to run in Fabric v2.4.

Applications and Peers

We're now going to show how client applications interact with peers, and more specifically, with the Fabric Gateway service running on peers, to access the ledger. Ledger-queries involve a simple dialogue between an application and a peer, while ledger-updates (writes) involve additional steps.

A client application connects to the Fabric Gateway service on a peer in order to access ledgers and chaincodes. Starting in Fabric v2.4, the Gateway SDKs (v1.x) make this easy for programmers. The APIs enable applications, via the gateway, to submit transaction proposals (which invokes chaincode), request endorsement, receive events, and forward endorsed transactions to the ordering service.

Through a peer connection on the gateway, applications can run chaincodes to query or update the ledger. The result of a ledger query transaction is returned with simple processing, whereas a ledger update (write) involves a more complex workflow between applications, peers and orderers. Let's investigate this ledger update process in detail.

Peers, in conjunction with orderers, ensure that the ledger is kept consistent and current on every peer in a channel. The following sequence, in three phases, describes how interactions between a client application, the gateway service running on a peer, orderer nodes and additional peers update the ledger.

Attention: The three transaction phases which follow explain the internal methods of how Fabric manages transactions. The Fabric Gateway SDKs implement these phases seamlessly; developers only need to use a Gateway SDK (1.x).

Phase 1 - Transaction Proposal and Endorsement

Phase 1 of a ledger update (write) consists of transaction proposal submission, execution and endorsement:

a) **Transaction proposal** — The client application (A1) submits a signed transaction proposal by connecting to the gateway service on P1. A1 must either delegate the selection of endorsing organizations to the gateway service or explicitly identify the organizations required for endorsement.

b) **Transaction execution** — The gateway service selects P1, or another peer in its organization, to execute the transaction. The selected peer executes the chaincode (S1) specified in the proposal, generates a proposal response (containing the read-write set). The selected peer signs the proposal response and returns it to the gateway.

c) **Transaction endorsement** — The gateway repeats transaction execution (b) for each organization required by the chaincode (smart contract) endorsement policies. The gateway service collects the signed proposal responses and creates a transaction envelope — which it returns to the client (SDK) for signing.

Phase 2 - Transaction Submission and Ordering

Phase 2 of a ledger update consists of transaction submission and ordering into blocks:

a) **Transaction submission** — The client (SDK) sends the signed transaction envelope to the gateway service. The gateway forwards the envelope to an ordering node and returns a success message to the client.

b) **Transaction ordering** — The ordering node (O1) verifies the signature, and the ordering service orders the transaction, and packages it with other ordered transactions into blocks. The ordering service then distributes the block to all peers in the channel for validation and commitment to the ledger.

Phase 3 - Transaction Validation and Commitment

Phase 3 of a ledger update consists of transaction validation, ledger commitment and a commit event:

a) **Transaction validation** — Each peer checks that the client signature on the transaction envelope matches the signature on the original transaction proposal. Each peer also checks that all read-write sets and status responses are equivalent (i.e. the endorsements from all peers match) and that the endorsements satisfy the endorsement policies. Each peer then marks each transaction as valid or invalid for commitment to the ledger.

b) **Transaction commitment** — Each peer commits the ordered block of transactions to the channel ledger (L1). The commit is an immutable ledger update (write) to the channel ledger. The world state (essentially, the sum of all valid transactions) of the channel is updated with results of valid transactions only.

c) **Commit event** — Each peer that commits to the ledger sends the client a commit status event with proof of the ledger update.

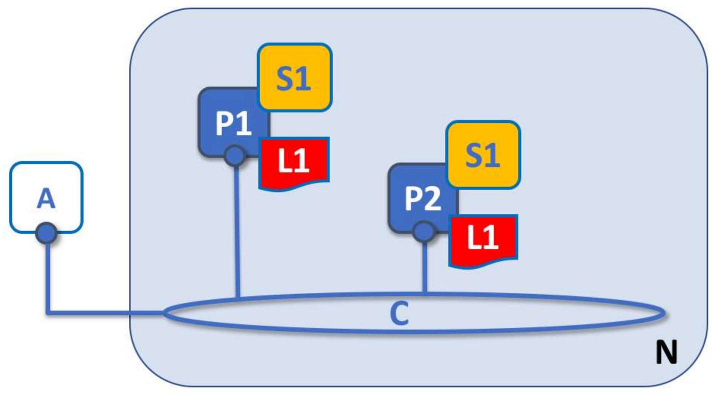
Note: Fabric v2.3 SDKs, which embed transaction proposal and endorsement functionality in the client application, remain supported in Fabric v2.4. Refer to the v2.3 Applications and Peers topic for details.

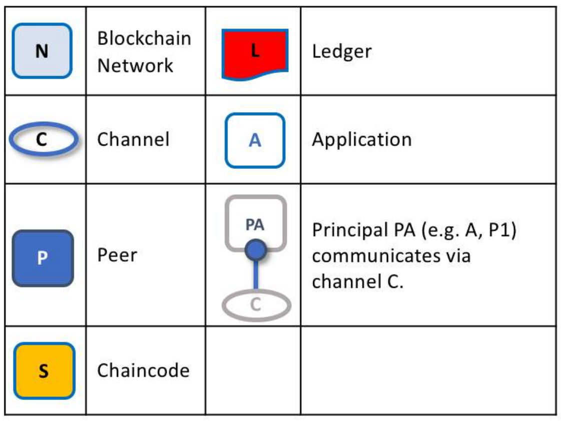
Peers and Channels

It is worth spending some time understanding how peers interact with each other, and with applications, on a *channel* — a mechanism by which components within a Fabric blockchain network communicate and transact *privately*.

Channel components include peer nodes, orderer nodes and applications, and by joining a channel, they agree to collaborate to collectively manage and share identical copies of the ledger. Conceptually, you can compare channels to groups of friends; a person might have several groups of friends with each group participating in different activities. These groups might be entirely separate (a group of work friends as compared to a group of hobby friends), or there can be some crossover membership between them. Nevertheless, each friend group is its own entity, with specific rules (or expectations) that establish and maintain membership.

Channel membership works the same way as in other groups; any one peer may belong to several channels and maintain a ledger and chaincodes specific to each channel. Or a peer may belong to only a single channel, and therefore have only one set of rules to follow.





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		

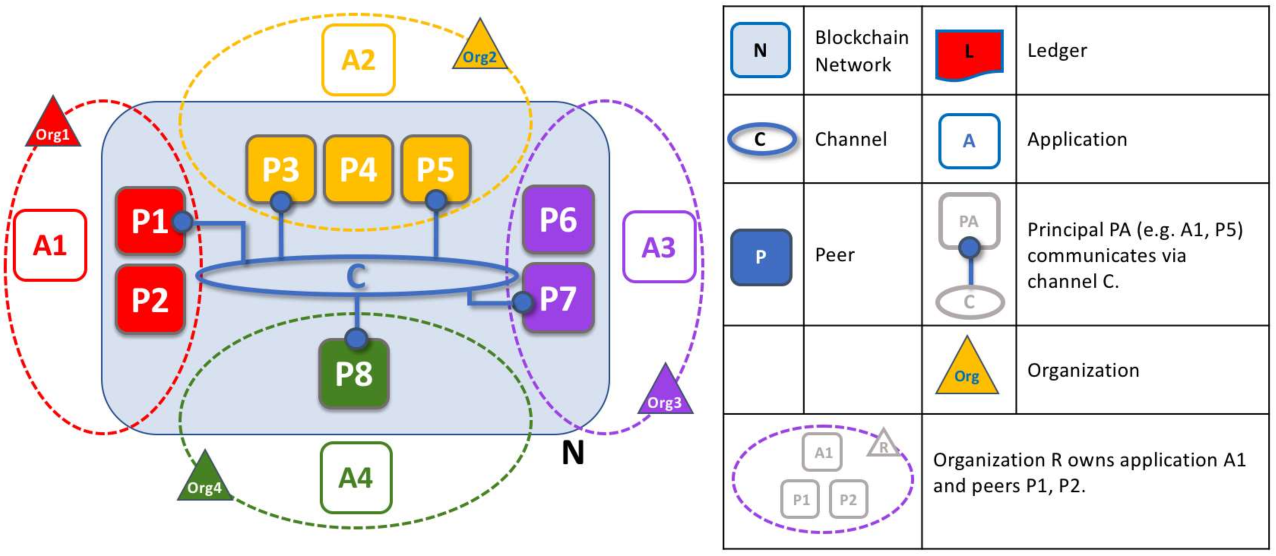
*Channels allow a specific set of applications and peers (and organizations) to communicate with each other on a Fabric blockchain network. In the example above, application A communicates with peers P1 and P2, through the gateway service, on channel C. The channel is a pathway for communications between specific applications and peers.

We see that channels don't exist in the same way that peers do — it's more accurate to think of a channel as a logical structure that is formed by a collection of physical peers. *It is vital to understand this point — peers provide the control point for access to, and management of, channels.*

Peers and Organizations

Now that you understand peers and their relationship to ledgers, chaincodes and channels, you'll be able to see how multiple organizations come together to form a blockchain network.

Fabric blockchain networks are administered by a collection of organizations rather than a single organization. Peers are central to how this kind of distributed network is built because they are owned by — and are the network connection points for — these organizations.


A2
Org2
Org1
A1
P1
P2
P3
P4
P5
P6
P7
A3
C
P8
Org3
Org4
A4
N
Blockchain Network
Ledger
Channel
Application
Peer
Principal PA (e.g. A1, P5) communicates via channel C.
Organization
Organization R owns application A1 and peers P1, P2.

*The example above shows organizations and their peers in a Fabric blockchain network. We see four organizations contributing a total of eight peers to form a network. Channel C connects five of these peers in the network N — P1, P3, P5, P7 and P8. The other peers owned by these organizations have not joined channel C, but typically join at least one other channel. Applications developed by an organization connect to peers, via the Fabric Gateway service, in the same organization as well as peers in other organizations on a channel.

It is important to notice what happens during the formation of a Fabric blockchain network. *The network is both formed and managed by the multiple organizations that contribute resources to it.* Peers are the resources that we're discussing in this topic, but organizations provide other resources too, such as chaincode and ordering service nodes. There's a principle at work here — the network literally does not exist without organizations contributing their individual resources to the collective network. Moreover, the network grows as resources are provided by collaborating organizations, increasing the resiliency and security of the network.

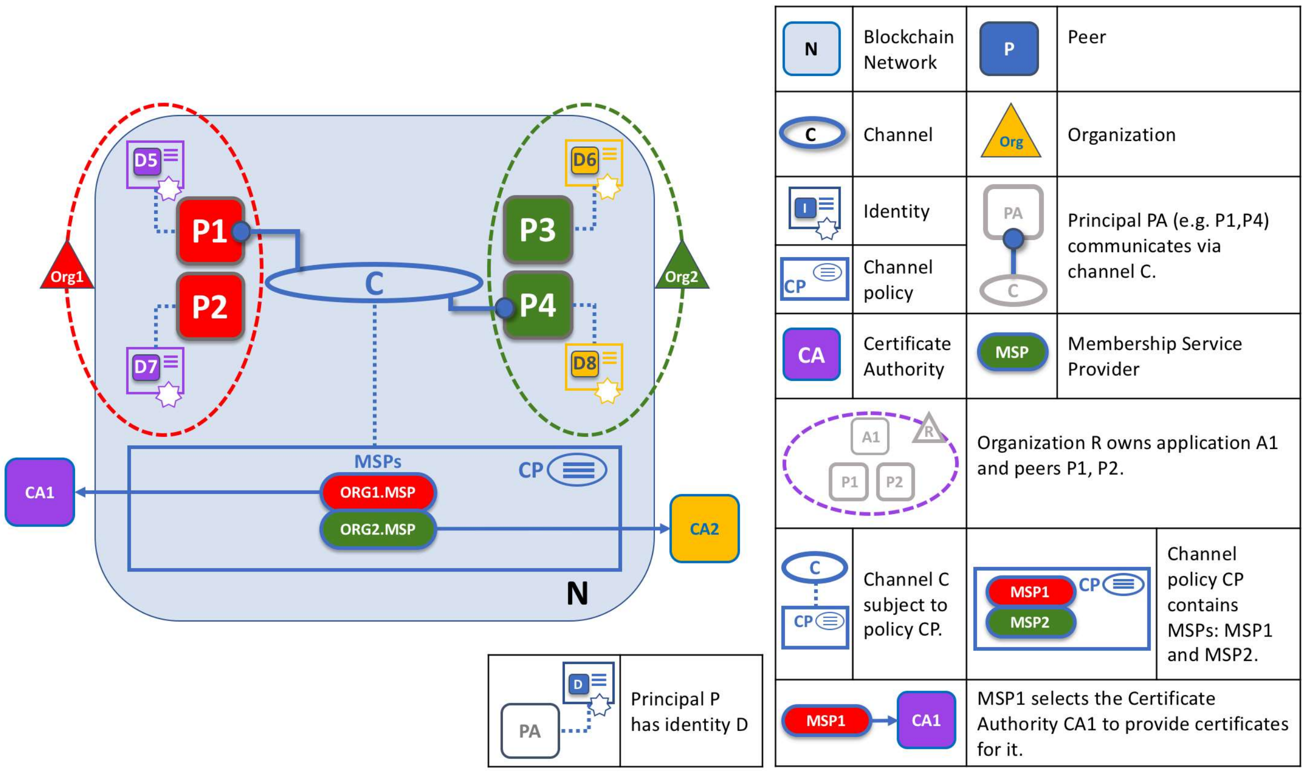
You can see that (other than the ordering service, to some degree) there are no centralized resources — in the previous diagram, the network **N** would not exist if the organizations did not contribute their peers and other resources. Moreover, the network does not depend on any individual organization — it continues to exist despite departures as long as its membership meets the self-defined requirements of the network. This is at the heart of what it means for a network to be decentralized; all of its member organizations share and contribute equally.

Applications used by organizations, as in the previous diagram, may or may not be the same, because each organization can decide how it wants to use data on the ledger. Both application and presentation logic can therefore vary across organizations, even though all peers host an equivalent copy of the ledger.

Peers and Identity

Now that you've seen how peers from different organizations come together to form a blockchain network, it's worth spending a few moments understanding how peers get assigned to organizations by their administrators.

Peers have an identity assigned to them via a digital certificate from a particular certificate authority. You can read a lot more about how X.509 digital certificates work elsewhere in this guide, but for now, think of a digital certificate as like an ID card that provides verifiable information about a peer. *Each and every peer in the network is assigned a digital certificate by an administrator from its owning organization.*



D5
D6
P1
P3
Org1
C
Org2
P2
P4
D7
D8
MSPs
CP
CA1
ORG1.MSP
ORG2.MSP
CA2
N
D
PA
Principal P has identity D
N
Blockchain Network
P
Peer
C
Channel
Org
Organization
I
Identity
PA
C
Principal PA (e.g. P1,P4) communicates via channel C.
CP
Channel policy
CA
Certificate Authority
MSP
Membership Service Provider
A1
R
P1
P2
Organization R owns application A1 and peers P1, P2.
C
CP
Channel C subject to policy CP.
MSP1
CP
MSP2
Channel policy CP contains MSPs: MSP1 and MSP2.
MSP1
CA1
MSP1 selects the Certificate Authority CA1 to provide certificates for it.

When a peer connects to a channel, its digital certificate identifies its owning organization via a channel MSP. In the example above, P1 and P2 have identities issued by a certificate authority (CA1). Channel C determines, from a policy in its channel configuration, that identities from CA1 should be associated with Org1 using ORG1.MSP. Similarly, P3 and P4 are identified by ORG2.MSP as part of Org2.

Whenever a peer connects to a Fabric network channel, *a policy in the channel configuration uses the peer's identity to determine its rights.* The mapping of identity to organization is provided by a component called a *Membership Service Provider* (MSP) — which determines how a peer gets assigned to a specific role in a particular organization and accordingly, gains authorized access to resources. Moreover, a peer can be owned only by a single organization, and is therefore associated with a single MSP. Think of an MSP as linking an individual identity with a particular organizational role in a Fabric network.

Peers, and *anything that interacts with a Fabric network, acquire their organizational identity from their digital certificate and an MSP*. Peers, applications, end users, administrators and orderers must each have an identity, and an associated MSP, to interact with the network. Any entity that interacts with a blockchain network using an identity is known as a *principal.* Note that identity is distinct from where the peer is physically located — it could reside in the cloud, or in a data center owned by one of the organizations, or on a local machine — the digital certificate associated with the peer is what identifies it as owned by a particular organization. In the previous example diagram, P3 could be hosted in Org1's data center, but as long as the digital certificate associated with it is issued by CA2, it is owned by Org2.

Peers and Orderers

We've seen that peers form the basis for a Fabric blockchain network, hosting ledgers and smart contracts which can be queried and updated by peer-connected applications. The mechanism by which applications and peers interact with each other to keep the ledger current and consistent across a channel is mediated by special nodes called *orderers*. It's to these orderer nodes that we now turn our attention.

A ledger update transaction is different from a query transaction because a single peer cannot, on its own, update the ledger — that requires the consent of other peers in the network, a process known as *consensus*. When the peers required to approve the transaction do so, and the transaction is committed to the ledger, the gateway service notifies the applications that the ledger has been updated. Peers and orderers work together to manage consensus.

Orderers and the Three Phases

Previously, in the Applications and Peers topic, we saw how a client application that requests a ledger update initiates a three-phase process which, with help from the Fabric Gateway service, ensures that all peers in a Fabric network maintain current and consistent copies of the ledger. As described, orderers are also involved; the following sections take a closer look at the three-phase process from the perspective of ordering nodes and peers (i.e. the ordering service and the gateway service).

Phase 1: Transaction Proposal

Phase 1 of the transaction workflow involves an interaction between a client application, peers and orderers. In Phase 1, the client application initiates a request to the Fabric Gateway service to evaluate a transaction proposal.

The target peer, selected by the client application, executes the transaction by invoking chaincode — this step can be described as simulating the transaction, because it runs the transaction without any effect on the ledger. The peer then returns its transaction result to the client.

The gateway service also forwards the transaction proposal to the required *endorsing peers* (based on the endorsement policies), which also execute the transaction and return their results to the peer. The gateway service collects all responses, and if they collectively satisfy the endorsement policies, forwards the transaction to the ordering service.

If the proposed transaction would write to a private data collection, (as *transient* data) the client application must explicitly specify the organizations required for endorsement.

Note that a peer endorses a proposal response by adding its digital signature, and signing the entire payload using its private key. This endorsement can be subsequently used to prove that this organization's peer generated a particular response.

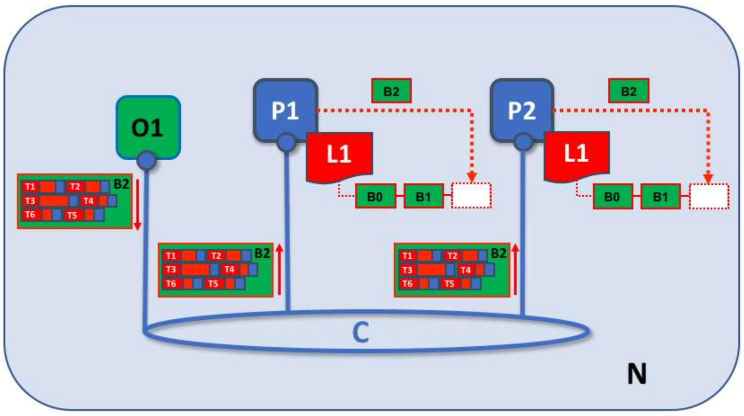
Phase 2: Transaction Ordering

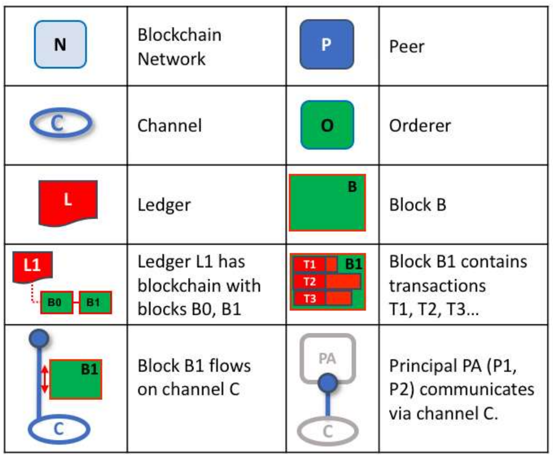
The second phase of the transaction workflow is the ordering and packaging phase. The ordering service (running on orderer nodes) receives transactions containing signed and endorsed proposal responses, from one or more applications via the gateway service, and orders and packages the transactions into blocks. These are the blocks (which are also ordered) — consisting of endorsed and ordered transactions — that make up a Fabric blockchain ledger.

For details about the ordering and packaging phase, see the conceptual information about the ordering phase.

Phase 3: Validation and Commitment

The third and final phase of the transaction workflow is the distribution of ordered transactions from orderers to peers. Each peer then validates each transaction, in the correct order, and ensures that each transaction has been consistently endorsed by all required organizations. Only then does the peer commit the block to its copy of the channel ledger.





N	Blockchain Network	P	Peer
C	Channel	O	Orderer
L	Ledger	B	Block B
L1 B0 B1	Ledger L1 has blockchain with blocks B0, B1	T1 T2 T3 B1	Block B1 contains transactions T1, T2, T3...
B1 C	Block B1 flows on channel C	PA C	Principal PA (P1, P2) communicates via channel C.

An orderer node distributes ordered blocks to peers for validation and commitment. In this example above, orderer O1 distributes block B2 to peers P1 and P2. Peer P1 processes block B2, resulting in a new block being added to ledger L1 on P1. In parallel, peer P2 processes block B2, resulting in a new block being added to ledger L1 on P2. Once complete, the ledger L1 has been consistently updated on peers P1 and P2, and the gateway service informs the relevant applications that the transaction has been committed.

Phase 3 begins with the orderer distributing blocks to all peers connected to it. Peers are connected to orderers on channels such that when a new block is generated, all of the peers connected to the orderer will be sent a copy of the new block. Each peer will process this block independently, but in exactly the same way as every other peer on the channel. In this way, we'll see that the ledger can be kept consistent. It's also worth noting that not every peer needs to be connected to an orderer — peers can cascade blocks to other peers using the **gossip** protocol, who also can process them independently. But let's leave that discussion to another time!

Upon receipt of a block, a peer will process each transaction in the sequence specified in the block. For each transaction, the peer will verify that the transaction has been endorsed by the required organizations according to the *endorsement policies* for the chaincode which generated the transaction. For example, some transactions may only need to be endorsed by a single organization, whereas others may require multiple endorsements to be valid. This process of validation verifies that all relevant organizations have generated the same outcome or result.

If a transaction has been endorsed correctly, the peer will attempt to apply it to the ledger. To do this, a peer must perform a ledger consistency check to verify that the current state of the ledger is compatible with the state of the ledger when the proposed update was generated. This may not always be possible, even when the transaction has been fully endorsed. For example, another transaction may have updated the same asset in the ledger such that the transaction update is no longer valid and therefore can no longer be applied. In this way, the ledger is kept consistent across each peer in the channel because they each follow the same rules for validation.

After a peer has successfully validated each individual transaction, it updates the ledger. Failed transactions are not applied to the ledger, but they are retained for audit purposes, as are successful transactions. This means that peer blocks are almost exactly the same as the blocks received from the orderer, except for a valid or invalid indicator on each transaction in the block.

Note that phase 3 does not require running chaincode — this is done only during phase 1, and that's important. It means that chaincodes only have to be available on endorsing nodes, rather than throughout the blockchain network. This keeps the logic of the chaincode confidential to endorsing organizations only. This is in contrast to the output of the chaincodes (transaction proposal responses), which are shared with every peer in the channel, whether or not they endorsed the transaction. This specialization of endorsing peers is designed to help scalability and confidentiality.

Finally, each time a block is committed to a peer's ledger, that peer generates an *event*. *Block events* include the full block content, while *block transaction events* include summary information only, such as whether each transaction in the block has been validated or invalidated. *Chaincode* events that the chaincode execution has produced can also be published at this time. Applications can register for these event types for notification. The commit event notification concludes the third and final phase of the transaction workflow.

In summary, phase 3 sees the transaction blocks which are generated by the orderer validated and consistently applied to the ledger. The strict ordering of transactions into blocks allows each peer to validate that transaction updates are consistently applied across the channel.

Orderers and Consensus

This entire transaction workflow process is called *consensus*, because peers have to collectively reach agreement on the order and content of transactions, with mediation from orderers. Consensus is a multi-step process and ledger updates only occur when the process completes successfully — which can happen at slightly different times on different peers. Think of the orderers as nodes which sequence and distribute proposed ledger updates for peers to endorse and add to the ledger.

Peers Summary

That's it! We've now finished our detailed tour of peers and their interactions with other Fabric components, including client applications, chaincodes and orderers. We've seen that peers are in many ways the most fundamental element of Hyperledger Fabric architecture — they form the network, host chaincodes and the ledger, handle transaction proposals and responses, and keep the ledger updated and consistent with validated and endorsed transactions.