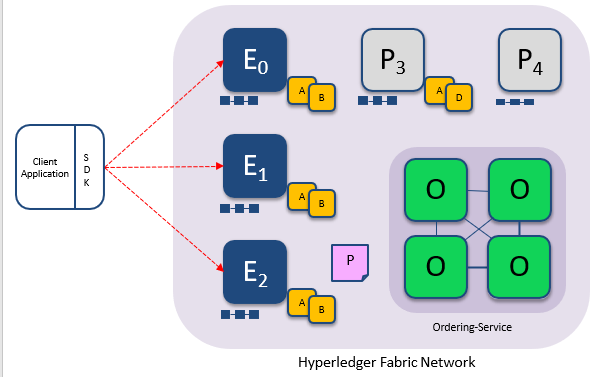
**Lecture 4-9 Notes**

**Hyperledger Fabric V1 Architecture**

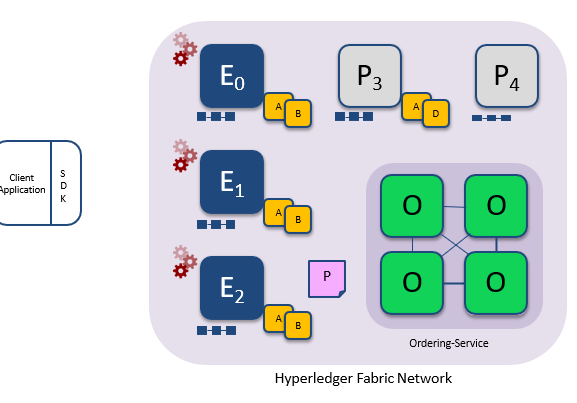
**Step by step Process**

**Step 1/7: Propose Transaction**

****

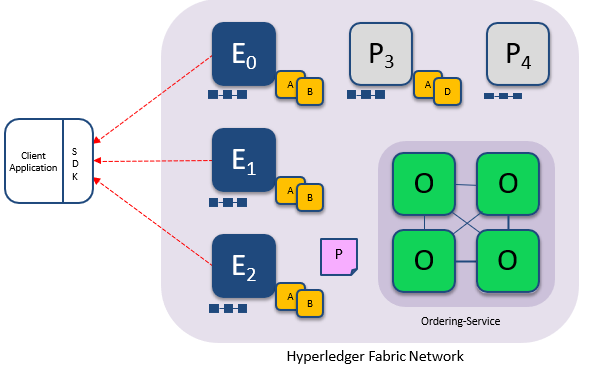
* Step 1 of the whole transaction flow is what we call proposing at transaction.
* A client application or a user is going to propose a particular transaction. It will say I am this user, I have this identity on the network, I want to invoke this particular function of smart contract, here is my transaction, here are the inputs to that transactions.
* This client application will send to multiple endorsing peer. So, E0 E1 and E2 in above figure are all endorsing peer they will all execute this transaction.

**Step 2/7: Execute Proposed Transaction**



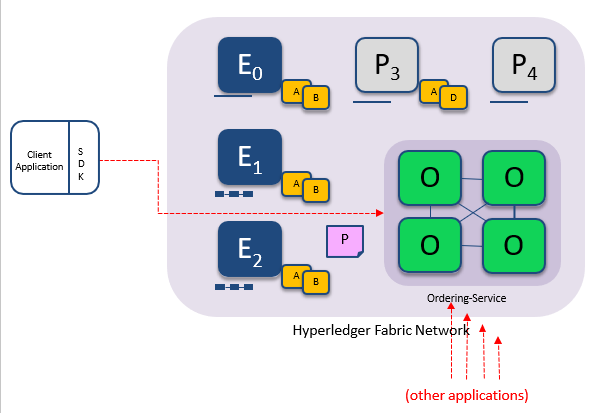
* The step 2 is each of these endorses have to execute this transaction. E0 E1 and E2 are going to execute these transactions and each execution will capture the read and write set.
* The read write set is collected for that particular transaction and each endorser will find on it, it can be encrypted, so all these communications between parties are encrypted for security purposes.

**Step 3/7: Proposal Response**



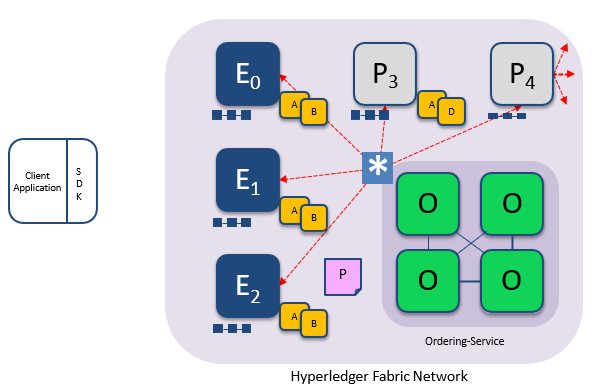
The third step of the flow is for the endorsers to communicate fact that this is the output they see along with their signature. The application or the SDK or the client is now gone receive these read write sets; and all of these are happening in parallel. It is possible that E0 and E1 respond first, whereas E2 takes little bit more time to come back with the response.

**Step 4/7: Order Transaction**



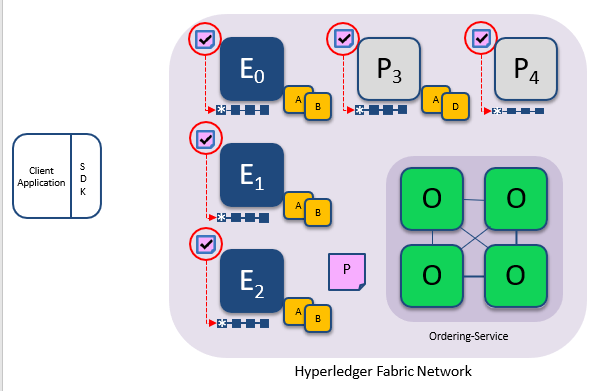
* The next step is ordering, once the client has received some sufficient endorsements that it thinks satisfies the endorsement policy. In this case it needed signatures from E0 E1 E2 it has got responses from all 3 of them, it is going to then submit the transaction to the ordering service.
* Such submission to ordering service can be happening from multiple users multiple client applications simultaneously across the network. Now any one node will not know who all submitted transactions, who all submitted other transactions at the same time.
* It is the ordering service that will determine that is what is shown here other applications are all submitting transactions to the ordering service and the ordering service is when going to determine how to order these transactions and it will then make sure everyone sees the same order across the network that is the ordering part of it of the equation.

**Step 5/7: Deliver Transaction**



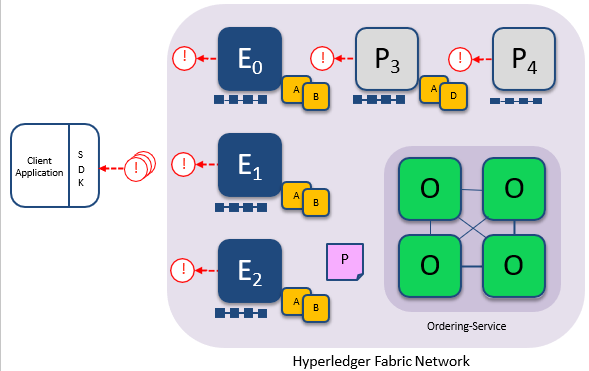
* Once the ordering service is determined an order, it is going to deliver that order set of ordered set of transactions to all the peers right in the network.
* This ordered set of transactions is what we are going to call a block, each block the ordering service forms a block and communicates that with all the peers.
* To say there are the 100 transactions which I have ordered all of you include this block in your block chain. So, these 100 transactions are now part of the block and they are communicated to all and it is possible that these peers themselves are architected in a way that there is a hierarchy.
* From the above diagram may be P4 is part of one organization which has other peers that are also that it also has as committing peers.
* This P4 can then communicate to other peer in its network whereas, it only P4 communicates with the order.

**Step 6/7: Validate Transaction**



* It is possible that one peer in the organization takes responsibility for communicating with other peers with the ordering service, but internally all the other peers can then create a copy of the block.
* Now how do you see ordering service form this ordered set of transactions, this is pluggable component in hyper ledger fabric you can use any ordering service.
* There is rich literature of over 30 years where various consensus algorithms have been designed for ordering and for consensus.
* There are a few that are implemented by hyper ledger fabric today one is just a solo order, think of it has there is a dictator who is just a single node that is going to dictate what the order should be. And right now.
* That single node implements of FIFO order first in first out, whatever I see first that transactions is going to be ordered first that is the SOLO order.
* The second ordering service that has been implemented is Kafka ordering service. Kafka is very popular event management service by Apache it is open source again and internally Kafka implements notion of consensus and that notion of consensus is a crash fault tolerant consensus.

**Step 7/7: Notify Transaction**



* The final step is now all the peers are going to commit the set of valid transactions they gone commit the block, they add that block into the block chain and each of them will emit events.
* The events itself in hyper ledger fabric you can get multiple kinds of events for the block itself you can get an event, saying this block of transactions was committed.
* For each transaction in the block whether it is valid or invalid for both of them you can get specific transaction level events and it is also possible for client applications to subscribe to specific transactions.
* For instance, the client application can only subscribe to the transaction that it has submitted. It might not be interested in getting notifications for the other transactions.

**Key Benefits of the Transaction Flow**

* what is the reason for having the endorsement first then a ordering then a validation?
* If you look at many of the existing block chain platforms, they actually have the thing inverted around they first do an ordering and then do the execution of that execution and validation of the transactions.
* It is an order and then execute is the traditional way in which many people have built block chain applications.
* Whereas hyper ledger fabric for very specific reasons we took a deviation from that who do the execution first and then ordering and validation and some other reasons are as follows.
* The notion of endorsements gives you a way for you to specify who in the system needs to validate a particular transaction.
* The second reason is to eliminate non-deterministic transactions, non-deterministic transactions can actually be play havoc in a block chain system what can happen.
* If it is non-deterministic because of that fact that these transactions are executed by all the peers or multiple peers in the network.
* If they all come up with the different answers then you can it can lead to inconsistent state across the network and we want all the nodes to maintain the same consisting state at any point of time.
* Let say I have a 20 node network, it is possible that there are 4 sets of 5 nodes it is 20 let say it is divided as 4 sets of 5 nodes, each of these 5 nodes can be executing a different transaction parallelly. Nodes 1 to 5 may be executing 1 transaction, nodes 6 to 10 may be executing one of the transaction and all of these transaction executions can happen in parallel and they can all be submitted to the ordering service in parallel.
* If the endorsement policy says I want 5 nodes to accept this transaction. Based on the endorsement policy you can use that to also scale in terms of transaction through put, you can have multiple parallel transactions going through the system simultaneously. So, that was also reason for us to adopt this first execute then order and then validate ok.

**Some of the components, some of the principles, concepts and Hyperledger fabric**

**Ordering Services**

* The ordering service as I mentioned is responsible for ordering a sequence of transactions into blocks and it go it is going to deliver sequence of totally ordered transactions to all the pears in the network.
* The communication with the service is via channels, there are different configuration options possible for the ordering service and the ordering service needs to be set up before you adopt the blockchain itself.
* the ordering service and the details of its configuration has to be included in the geneses block of the block chain.
* There are two ordering services that are implemented today in hyperledger fabric.
* SOLO ordering service, it is a single node, it is meant for development purposes and it is easy to setup and you do not have to worry about some of the distributed descent life’s aspects of the ordering service and this single node will just do a FIFO order. first in first out whichever transactions it sees first, it will order that first.
* Kafka consensus is a crash fault tolerance consensus it needs at least three nodes at a very minimum and we recommend an odd number of nodes in the network so that we can do a majority consensus. That is the ordering service and it needs to be setup before you setup everything else; anything in the network.

**Channels**

* The next notion is the notion of channels and the channels what it provides you is the notion of privacy or transaction privacy between different ledgers.
* The same set of peer nodes E0, E1 can actually be part of multiple block chain ledgers, each of them can have a different ordering service if they want to; it can have same as well and they are all going to maintain a different scoped ledger. So, each block chain is going to maintain the same ledger.
* It can have different applications with the different set of peers executing on each channel and the notion of ledgers that we talked about exist in the scope of a channel.
* Each channel has its own ledger and the data in that ledger is available for the peers in that channel, all the peers in the channel can see the data maintained by the ledger of that channel and it is available to all this.
* let us say a consortium of 20 banks that come together, there is a one channel that has all the 20 participants amongst themselves. But some sub set may be 2 banks or 5 banks might come together to form a private channel that only they have knowledge of; the other nodes in the network who are not permissioned as part of that channel will not even know the existence of that channel. So, that sort of privacy is possible, and the notion of channels also segregates chain code.

1. **Single Channel Network**

* Couple of examples. Here is a network with just a single channel, as before a client application is going to submit a transaction to the network, Say contacts E0, E1, E2, E3 are the peers or the endorsing peers in this network, there is an ordering service and there is a exactly one channel in this system and all the peers connects to that channel. So, this is the blue channel here.
* This channel is going to maintain sequence of blocks or a sequence of transactions and that is what is represented here. Now, A and B are smart contracts. So, these are chain codes that are deployed. So, these chain codes are all installed on every node in the network and they are also instantiated on that channel, on the blue channel.
* When our transaction is submitted by the client application to the network, let us say it is invoking a particular function in smart contract A, then all the 4 peers E0, E1, E2, E3 they all have a copy of that function, they are also going to be executing the same function across all the nodes and then of course, they will go through ordering and the consensus and then final validation.

1. **Multichannel Network**

* A multi-channel network, now there could be two different applications one application is transacting on the red channel, where as another application is transacting on the blue channel and these applications may not even know of the existence of the other channel.
* In this example, E 0 and E 3 are only part of the red channel right and they are only aware of smart contracts that are or chain codes that are deployed on that channel. E 1 and E 3 are only part of the blue channel and they do not know of existence of the red. So, this is the two completely disjoined channels in this network and in this example, they are actually using the same ordering service.
* It is possible that even the ordering service is separate across the channels, it is also possible that the same ordering service is ordering for both channels and the and the ordering for the two channels are kept separate.
* The transactions on one channel are only ordered for that channel, the transactions on the other channels are kept separate.
* Now the endorsement policy on these two channels can also be different or the same per smart contract basis, for the smart contract Y on the red channel, I can define an endorsement policy. For the smart contract A on the blue channel, I can define a different endorsement policy.

**Fabric peer**

* Peer: each peer can connect to one or more channels, these channels are completely disjoined from each other, they are private.
* one channel’s information is not mixed with another and so, in this case, the there are three different channels; there is a blue, there is a red and there is a green.
* The three different channels that this particular peer connects to and each channel has its own ledger and its own block chain and they are all kept separate.
* The actual process for the chain code; let us say that is chain code A and chain code B, these chain code are processes, they run inside actually docker containers, you can package them as this processes or you can package them as docker containers.

**Client Application**

* The client application, it can also receive events from multiple channels and the SDK needs to be connected to again local membership service provider that has the crypto materials for the users using that applications, based on the channels the client might also have to communicate with one or more ordering services.
* The blue and red might be using one ordering service, green might be using another ordering service. The client application has to be aware of the ordering services and all this information is available in the genesis block.
* Peers can enter and leave channels dynamically and the fact that the way peers entering and leaving the channel are also recorded as transactions on the block chains, they are called channel configuration transactions and they are also recorded on the block chain again in a decentralized manner.
* The client the application itself can be written in any language, but the client that connects with the block chain; there are multiple SDKs that we provide, we provide SDK in Node js, Java and Python and you can use that to connect with the blockchain.

**Fabric Certificate Authority**

* The fabric certificate authority, this is one implementation of certificate authority that issues identities for users who users or components transacting on a blockchain.
* This could be for users, for peers, for ordering services all of them have identities on the blockchain.
* A fabric CA can be used to issue all of those identities, there are many ways in which you can implement the certificate authority.
* The fabric CA is attached to root certificate authority and it can have multiple hierarchical level. It can have intermediate certificate authorities that link to the root CA and so on.
* It has database for high availability, it can also support high availability characteristics, It can be connected to other identity mechanisms like LDAP and so on.
* The keys that are issued by the fabric CA has stored securing in a hardware service module.