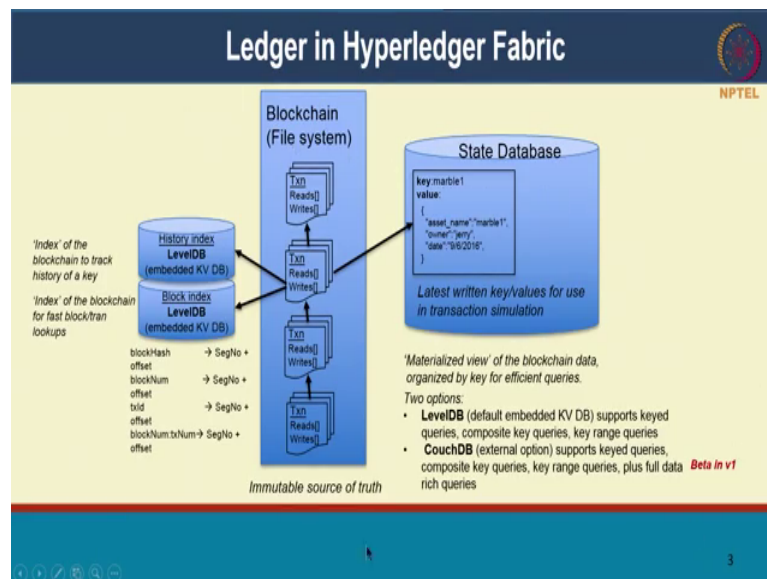


**Blockchains Architecture, Design and Use Cases**  
**Prof. Sandip Chakraborty**  
**Prof. Praveen Jayachandran**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 46**  
**Blockchain Security – III (Fabric SideDB)**

Hello everyone. Welcome back to the next lecture of our Architecture Design and Use Cases Course. We have been going through different notions of security and privacy. So, last lecture we looked at what are what is transaction privacy, data privacy, user privacy and what are the different constructs one could use to achieve privacy. And now we going to look at one other construct, a very important construct in hyperledger fabric that allows you to achieve data privacy. So, we look at that. So, it is called side DB.

(Refer Slide Time: 00:43)



So, before we get to side DB itself let us look at just the ledger aspect of hyperledger fabric, right.

So, what are the things we are storing on the blockchain on the ledger? So, we are going to store the blockchain itself. So, this is be chain link of blocks with transactions in each block and apart from that we have a state database that holds the data's that is handled by these smart contracts. So, each smart contract can have its own private data store and in that data store you can store information that it seeks to keep immutable on the

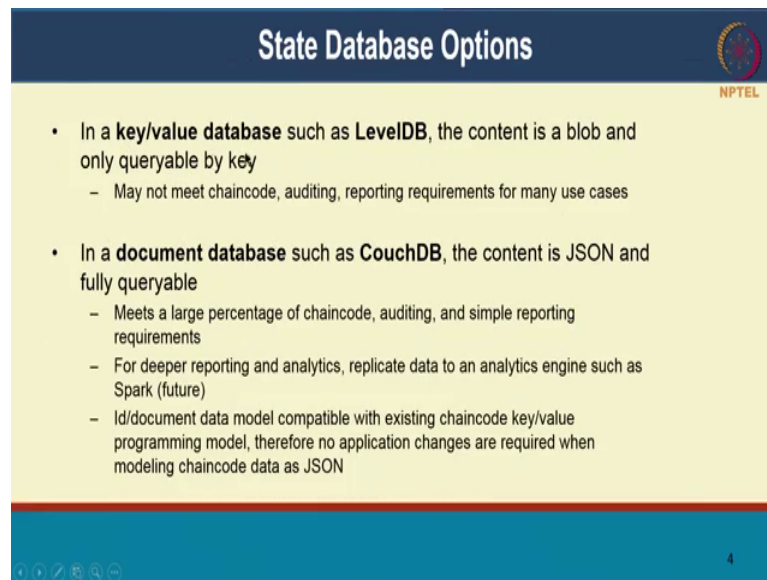
blockchain. And these transactions are referring to data in this state database as part of the smart contract invocations.

Now, on top of the state database there be couple of a few index indices that are created, one is a history index which says which transaction is in which block and a history of all such transactions over time and there is also block index, right. All these are stored, the indexes are stored on the state database and that is a level DB for fabric. And for the state database itself there are two options, one is you could use today at least there are two implementations; hyperledger fabric gives you a pluggable interface for your database definition.

Today there are two implementations, one is there level DB which is an embedded key value database and it supports basic functionality, right. So, it embeds supports key queries. So, I can give you a particular key ask for the value. It allows composite key queries and also allows range queries. So, give me all keys from a to b, right in that range. This is a basic very simple key value store.

The slightly more sophisticated one is couch DB which is really a document store. It supports, so basically each key is really a document, it can be a (Refer Time: 02:32) on document and it supports key queries, composite queries, range queries plus full data rich query. So, instance in the level DB case the data is really a block, but in couch DB you can actually have a schema and ask specific queries about data within a particular document itself or a particular key.

(Refer Slide Time: 02:54)



### State Database Options

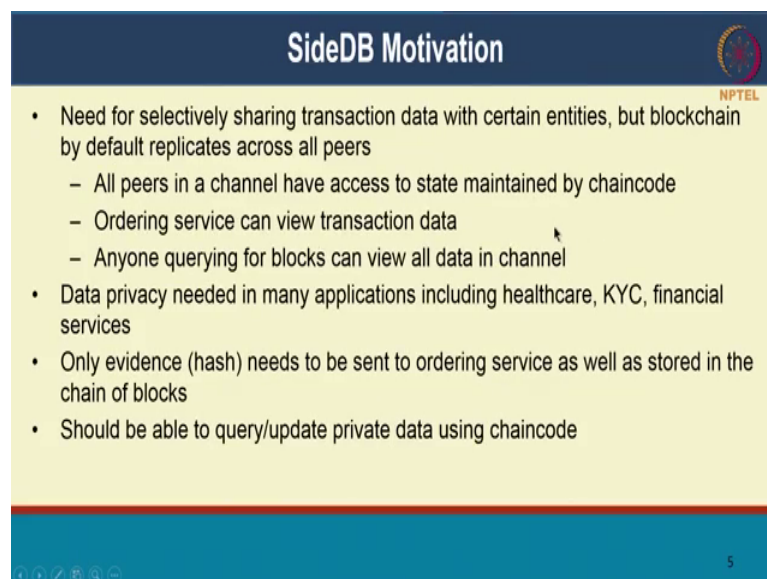
- In a **key/value database** such as **LevelDB**, the content is a blob and only queryable by **key**
  - May not meet chaincode, auditing, reporting requirements for many use cases
- In a **document database** such as **CouchDB**, the content is JSON and fully queryable
  - Meets a large percentage of chaincode, auditing, and simple reporting requirements
  - For deeper reporting and analytics, replicate data to an analytics engine such as Spark (future)
  - Id/document data model compatible with existing chaincode key/value programming model, therefore no application changes are required when modeling chaincode data as JSON

NPTEL

4

So, this is what I mentioned. So, with level DB it is a very simple key value store. So, all you can just do is get the entire value for a particular key, a few additional composite key in rich query capabilities. But these may not be sufficient for many applications, right. So, if you want to do more sophisticated query kept queries in your chaincode, it is really not this may not be sufficient, right. Be, might what you use a document store such as couch DB, where you can do a much more richer queries on the blockchain and you can also get richer reporting and analytics performed on the chain using chaincode, ok.

(Refer Slide Time: 03:34)



### SideDB Motivation

- Need for selectively sharing transaction data with certain entities, but blockchain by default replicates across all peers
  - All peers in a channel have access to state maintained by chaincode
  - Ordering service can view transaction data
  - Anyone querying for blocks can view all data in channel
- Data privacy needed in many applications including healthcare, KYC, financial services
- Only evidence (hash) needs to be sent to ordering service as well as stored in the chain of blocks
- Should be able to query/update private data using chaincode

NPTEL

5

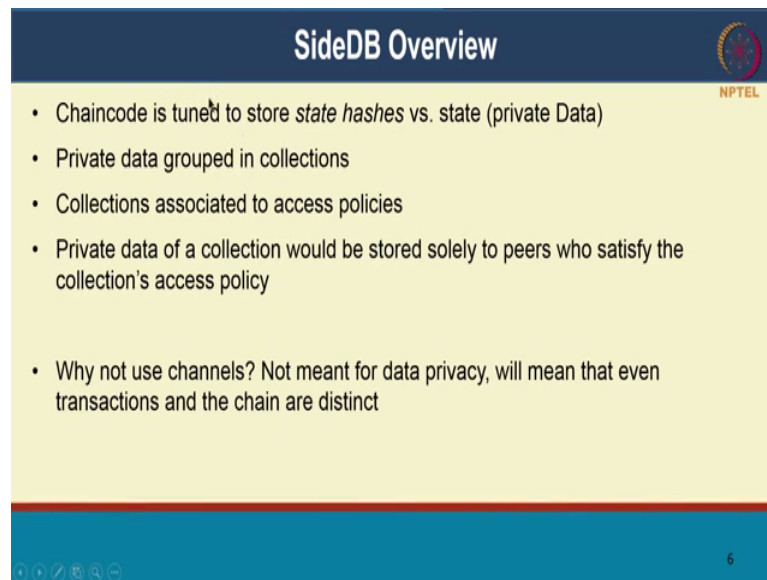
Now, coming to why a side DB, right. So, what we what many we have seen many applications ask for a many industries cases ask is the ability to be able to selectively share particular pieces of data with only certain entities and blockchain by default replicates all the data across all the peers. So, even if you have channels all the entities all the participants of channels will see all the data.

So, suppose you have some data that you consider private that you want to you want to share only with a subset of participants within a channel, right. So, that data you want a keep private and only that is a three out of 10 people should see the data no one else, right. How do you do that? The side DB provides that kind of capability where I can for each for each data element I can specify who are the other peers who will see the data. So, it gives you privacy even within a channel so that only subset of the peers see it. It also give you the privacy with respect to the ordering service. So, even the ordering service will not see that private data they will only see hash, right.

And anyone querying just the blocks of transactions just looking at the blockchain itself will also not see the private data. So, across the peers the blockchain, as well as the order we can ensure data privacy of specific private data. And this says it comes in handy for many applications where audit requirements are there is a stringent compliance and audit requirements and there are also regulatory requirements. So, such as healthcare, KYC, insurance and many financial services use cases also.

So, the way it does that is only the evidence of the private data maybe a hash is going to be is going to be on the transaction and will be seen by the ordering service. So, all the other information is kept private to a subset of the authorised peers, ok.

(Refer Slide Time: 05:33)



### SideDB Overview

- Chaincode is tuned to store *state hashes* vs. state (private Data)
- Private data grouped in collections
- Collections associated to access policies
- Private data of a collection would be stored solely to peers who satisfy the collection's access policy
- Why not use channels? Not meant for data privacy, will mean that even transactions and the chain are distinct

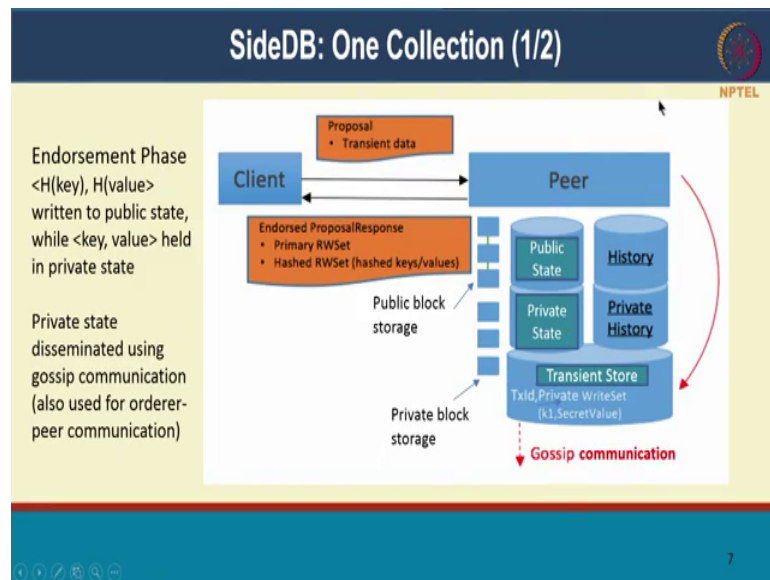
6

So, the chain code is going to store both public data as well as private data and the private data will only be with authorised entities only the hash of the private data gets onto the transaction. Now, the private data is it is possible to group them into collections. So, collections as an notion where I can say collection a has these 3 peers, collection b has these 5 peers. So, I can create those collections and I can have data within those collections.

So, if a particular data element is in a collection that data element will only be seen by participant of the collection and just like channels it is also possible for the membership of the collections to change over time, right. So, collections are again associated with access policies. So, you can define who can read the data, who can write data. So, those access policies can be defined for each collection and the as I said the private data is only stored on the peers who satisfy that access policy.

So, again the distinction from channel channels is that within a channel everyone sees all data, but side DB is a construct whereby you can prevent certain private information to be seen by to not be seen by all parties in a channel it will be seen by only a subset. The private data is also not in the transaction in not in the block of transactions, it is also not seen by the order. So, the unauthorised entities even if they are authorised within a channel they only see the hash they will not see the private data. So, how does this all work? So, let us take just one collection, right.

(Refer Slide Time: 07:15)

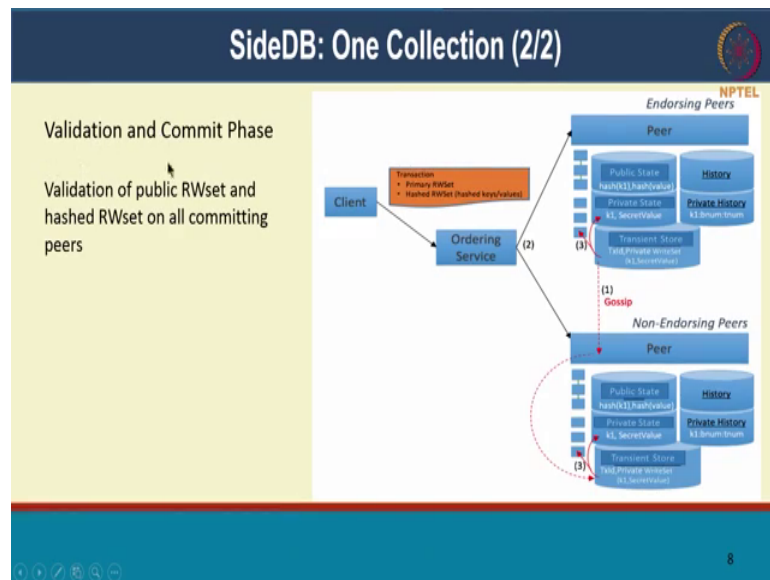


So, let us say there is an endorsement and that endorsement is going to have two parts to it. This is basically a transaction proposal. They will have two parts: one is a primary read-write side. So, what this means is this is the public information that anyone in the channel can see. So, this is really the read set of data elements that are read and written by the chaincode, which is public. That is the primary read-write set.

Apart from that, this is going to be private data in the collection. That is the private read-write set, and for that, only a hash of that private read-write set is captured in the endorsement itself in the transaction. So, in the endorsement phase, only the hash of the key and hash of value is written to the public state. The actual key-comma-value of the private information is held in a separate database, which is called the side DB. So, in this case, there is public state information and there is a private state, and these are held separately.

The private state is then disseminated using gossip. The public state is part of the transaction, and all the peers will get the transaction, but the private information is then distributed using gossip in fabric.

(Refer Slide Time: 08:35)

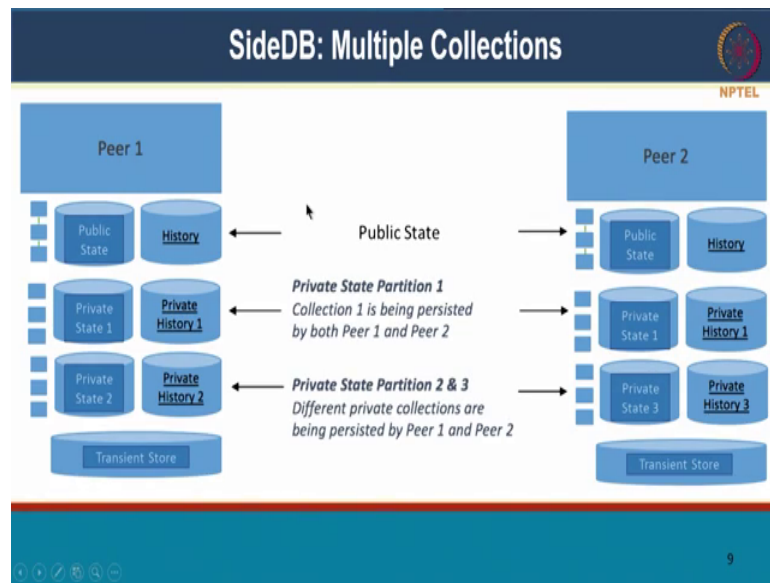


What happens after that? So, now you written the hash of the key and hash of value into the transaction so that make sure allows immutability and there is a private data set that only a subset of the pees in the channel have. Now how does validation take place? Now, the validation of course, of the public read write set happens as before, apart from that the hash read write set on all the committing peers they will also check the key, right sorry they will also check the version.

So, there is a hash of the key and as a particular version you can check that version is not duplicated, right. So, no two transactions in a particular block are reading the same information. So, that is a transaction a, reads a particular version for a private data and modifies it in some ways it is going to be, going to be part of the write set. So, tries to modify the data. If transaction b, had the same version it means that its read stale information because this is the previous transaction in the block that is my modify the private information. If transaction b, tries to modify the same element in the same block then that will be rejected because the version on the hash of the key can be checked, right. If both of them are in version 1.0 then the second one will get rejected.

So, that is really almost like a the same validation that fabric does on state variables, this is similar to not allowing double spending that is the same account to spans cannot be made from that, right. So, that really is how validation also happens. The validation happens on the hash of the key rather than the key itself.

(Refer Slide Time: 10:20)



Now, likewise just as we had one collection it is possible to have multiple collections. So, you can have different collections with different subsets of peers and there all held as separate state information. So, that is private state 1, private state 2 and in this example collection one is persisted between peer 1 and peer 2. And there is a partitions 2 and 3 they are persisted by just one of the peers, right. So, or private state 2 resides only in peer 1, private state 3 resides only in peer 2. So, they are they are bifurcate, right. And the only the hash of those values get into the public state, ok.

(Refer Slide Time: 11:05)

The slide is titled 'Define Collections for each Chaincode and Channel'. It contains two main bullet points: 'Define collections during chaincode deployment' and 'Using channel configuration'. The first bullet point has two sub-points: 'Need to have a special peer command to specify collections per chaincode per peer.' and 'Can easily add/remove collections to existing chaincode'. The second bullet point has two sub-points: 'Easy to configure collection' and 'To add/remove collections on demand need channel reconfiguration message'. The NPTEL logo is in the top right corner, and the number 10 is in the bottom right corner.

- Define collections during chaincode deployment
  - Need to have a special peer command to specify collections per chaincode per peer.
  - Can easily add/remove collections to existing chaincode
- Using channel configuration
  - Easy to configure collection
  - To add/remove collections on demand need channel reconfiguration message



So, these collections, so how does the life cycle work? The collections can be defined at the time of chaincode deployment. So, I can say these are the collections these data elements when you are writing into the when the when the chain code is going to write this state information into the ledger it will specify whether this is public or whether it is a private collection. And its possible to easily add or remove collections over the life cycle of the chain code, over time you can add new collections or remove collections. You can also using the channel configuration transaction you can also configure transactions. So, you can add new members to a collection you can remove members from a collection, so all that is possible, right. So, all that is happens through channel reconfiguration, ok.

(Refer Slide Time: 11:52)

The slide is titled "Fun Reading" and features the NPTEL logo in the top right corner. It contains two bullet points:

- Hyperledger Fabric documentation, Ledger: <http://hyperledger-fabric.readthedocs.io/en/release-1.0/ledger.html>
- Hyperledger Fabric SideDB: <https://jira.hyperledger.org/secure/attachment/12720/PrivacyEnabledLedger20171022.pptx>

The slide footer includes navigation icons and the number 11.

So, that is a really nutshell the whole state DB notion of how subset of entities within a channel can really hold private data just amongst them, that data is shared with others through gossip with the other authorised entities through gossip. What is there in the public blockchain itself is just hash of that private data, right. So, in some sense immutability is guaranteed by the platform itself. So, they no party can modify this private data. So, the other people who have the private data will be able to detect that because the hash is on the public chain. And double spends also avoided because validation can be performed using version information on the hash of the key, right. The key itself is not revealed only the hash of the key is in the public state.

So, to get more details you can check out the fabric documentation on ledger, gives you good details and also working example of how side DB works. So, this really helps provide data privacy amongst sub set of participants within a within a channel and also with respect to the ordering service ok.

So, that ends this lecture on using side DB as a construct for privacy. So, with this I think you seen a range of constructs for providing security and privacy in blockchain platforms. And this is a very hot area for research and an innovation, there is lot of action going on in both academy as well as industry in the space.

With that thanks a lot. I will see you in the next lecture.