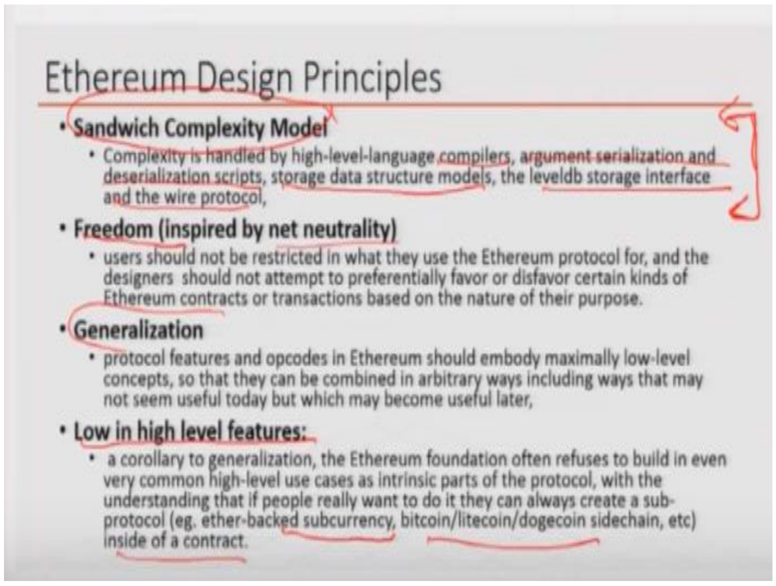
Introduction to Blockchain Technology & Applications

Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 15

Welcome back to the blockchain technology and applications.

(Refer Slide Time: 00:21)



Ethereum Design Principles

- **Sandwich Complexity Model**
  - Complexity is handled by high-level-language compilers, argument serialization and deserialization scripts, storage data structure models, the leveldb storage interface and the wire protocol,
- **Freedom (inspired by net neutrality)**
  - users should not be restricted in what they use the Ethereum protocol for, and the designers should not attempt to preferentially favor or disfavor certain kinds of Ethereum contracts or transactions based on the nature of their purpose.
- **Generalization**
  - protocol features and opcodes in Ethereum should embody maximally low-level concepts, so that they can be combined in arbitrary ways including ways that may not seem useful today but which may become useful later,
- **Low in high level features:**
  - a corollary to generalization, the Ethereum foundation often refuses to build in even very common high-level use cases as intrinsic parts of the protocol, with the understanding that if people really want to do it they can always create a sub-protocol (eg. ether-backed subcurrency, bitcoin/litecoin/dogecoin sidechain, etc) inside of a contract.

We were talking about dapps or Ethereum based applications that are interacted with by users through web interface and the underlying technology of smart contracts and blockchain can be hidden. Now as I said, that will give you some indicators as to where to go and learn more about it towards the end of this lecture.

But before that, I want to point out a certain few other things about Ethereum which Ethereum foundation so these are from the documents of white paper of Ethereum Foundation, which basically explains their choice of the design of Ethereum which is quite different from bitcoin, although both are currency, cryptocurrency block chains and both are permissionless blockchain and both are pseudo anonymous.

And both have the notion of transactions and mining and hash puzzles and everything. But still they have very different design principles on which the entire design and architecture of Ethereum was done. And this is done because the Vitalik Buterin who is the originator of Ethereum was using bitcoin blockchain for a while, and then he realized that there are certain things that are not very easy for with bitcoin.

And there are certain design principles that he did not like of bitcoin and therefore he is explaining here and I think you should know that, because one of the goal of this course is to actually give you more conceptual handle on the blockchain technology then nitty gritty programming practices of blockchain. So you should be able to explain to somebody who is asking what is the difference between Ethereum and bitcoin.

Of course, you can talk about smart contracts you can talk about Solidity which is Turing-complete language. But there are certain other things more higher level principles that are at play here. So Ethereum has this notion of what they call Sandwich Complexity Model. So sandwich basically means that it has three layers right. So top layer and bottom layer are bread and then middle layer has something like a meat or some vegetables or something.

So here also the idea is that all the complexity is in the top layer and the bottom layer and then the middle layer is kept as simple as possible. So complexity is handled at the high level with the language, Solidity language and the language compiler argument serialization and deserialization scripts, that is when you have complex data structures that are part of your arguments in a function called smart contract, how they are serialized because they have to be traversing the network, right.

So therefore, then there is need for serialization but the user does not have to do these things themselves. There are already scripts, which do the serialization, deserialization automatically. So user will use it like a normal function called storage data structure models. And then the leveldb storage interface at the bottom layer. That is where the information is stored.

And the wire protocol that is the peer to peer protocol that is used for interacting between the nodes in the blockchain. So that is the idea of Sandwich Complexity Model. So this is not something that the bitcoin used, right? So bitcoin has kind of a monolithic complexity and therefore the user, like who puts transactions have to you know, know a lot more the working of the bitcoin mechanism in order to write complex scripts.

Here, the programming model is very much like a regular program, like when you write regular programs, you do not think of like the computer architecture, the registers and program stack and all that stuff when you write your program. Similarly, in Solidity when you write smart contracts, you do not think of the underlying architecture microarchitecture of the EVM and all that issues.

So that complexity is kind of hidden from you. So there is another idea of freedom or net neutrality. So if you have noticed in bitcoin the, it is very biased towards only bitcoin transaction. So we talked about how you can use bitcoin to actually store some data permanently or timestamp some data permanently, but usually those are not favored. So the way the bitcoin has been designed is kind of singularly meant for the bitcoin transactions.

So what they are they decided is that we should not restrict the users so that they have that the system or the platform preferentially disfavors a particular type of transactions or particular type of applications. So it is kind of net neutrality inspired idea that any kind of application it could be cryptocurrency application, it could be ecommerce application, it could be just data storage.

Like land record, where you are trying to guarantee the integrity of the data, or it could be a grade management system. All that stuff can be actually managed in this with equal difficulty or equal simplicity. So that is the freedom principle. Generalizing principle is that all the protocol features, and opcodes in Ethereum, are actually low level concepts. So they do not create high level opcodes, which do very high level that an opcode that basically it says, do something very complex.

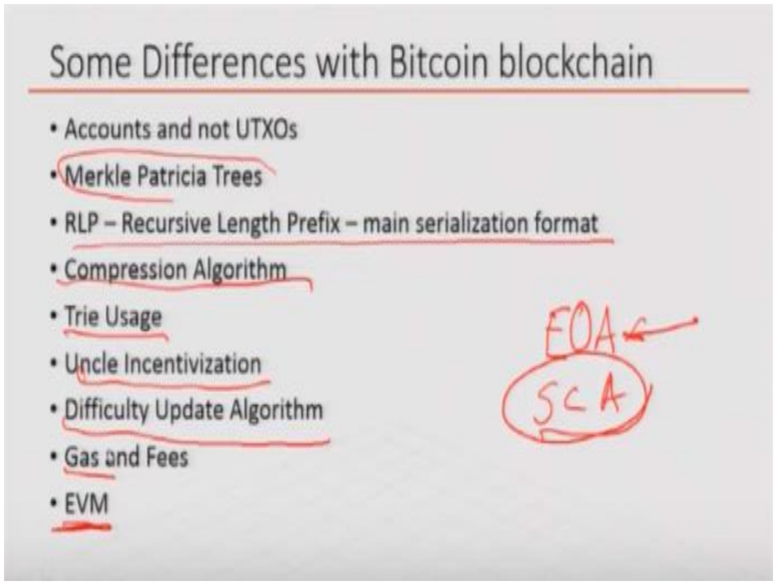
So everything is of very low level, simple opcodes semantics are very simple low level ideas. So one can actually use them in arbitrary ways. So create very complex functionality. So more it is like, you know like, if you want to do something very complex, assembly level programming might be more flexible for you. Then let us say you are using some high level programming language like a functional language.

And then you have to you know do a lot of trickery to make it do something complex and something that is not meant for by the language designers. Here, they did not do that choice and it is very generalizable. And therefore, as a principle, as a corollary to this principle of generalization, they have very few or almost no high level features. And they expect that if you want to create high level features, you can create protocols on top of Ethereum protocol.

So you can make ether-backed subcurrency, you can define your own tokens. And like this is how the ECOS are done. And you can also make other kinds of coins and etc., using sub contracts. So that flexibility is there. So four principles that design principles is that a complexity is kept, you know out of the users purview. The freedom in the sense that it does not disfavor a particular type of production or favor a particular type of application on top of Ethereum.

It is generalizable and generalizability is probably the core to this freedom and because it is basically uses all opcode semantics are very low level concept based and not high level and then therefore the as a corollary it is low in high level features. So remember, this very high level design principles that drove the creation of Ethereum and the design and architecture of Ethereum.

(Refer Slide Time: 08:48)



Some Differences with Bitcoin blockchain

- Accounts and not UTXOs
- Merkle Patricia Trees
- RLP – Recursive Length Prefix – main serialization format
- Compression Algorithm
- Trie Usage
- Uncle Incentivization
- Difficulty Update Algorithm
- Gas and Fees
- EVM

So another difference that is interesting in case of you know difference between bitcoin blockchain and Ethereum is in bitcoin there is no concept of real account. Everything is kept track of by unspent transactions, right. So whenever you give some money to another person, you have your digital signature on it. And then whenever you want to use that money, you have to use that as an input transaction.

And then you create output transactions. So these UTXOs are the ones which are not yet used. So this UTXOs are kept track of. So that is kind of the state of the current state of the blockchain. So in Ethereum, however, we do not keep track of UTXOs, we have the notion of accounts. So every account the accounts are two types. One is called EOS or external user accounts.

And another is the smart contract accounts, right. So externally owned accounts and smart contract accounts. So these accounts have their addresses, they have ether balances. The balance could be zero, or the balance could be positive, but they have balances. So when you send money to somebody, let us say a real person with an externally owned account, then that balance will go up.

If the person spends money from its balance, then the balance will go down. Similarly, the subcontract accounts also can keep ether as balance and can send ether to externally on the accounts or to other smart contracts for getting things done. So therefore, they are the notion of accounts and not UTXOs. In Ethereum, we saw that the transactions are kept as Merkle trees. In here, we see a little different concept.

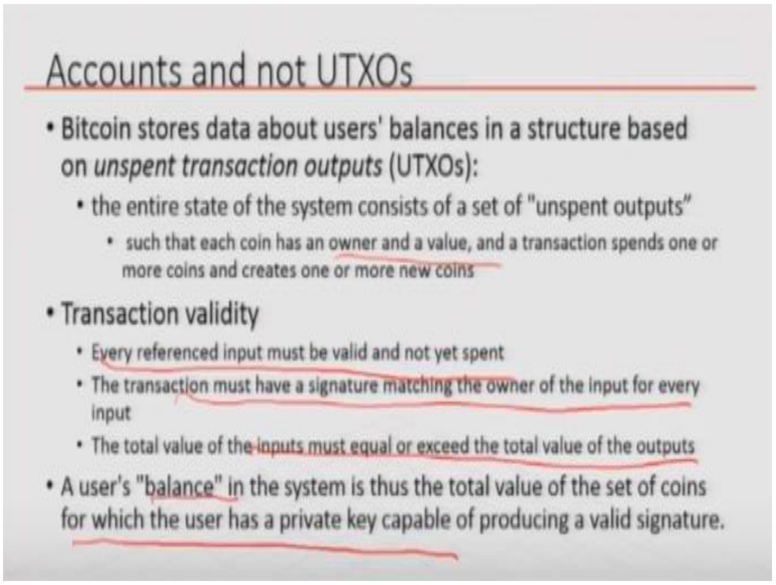
They use something called Merkle Patricia trees and we will show you what this means. They have a different serialization format called Recursive Length Prefix. And they have a compression algorithm. They use cries, which is a particular data structure on which the Patricia trees are based.

And they have this notion of Uncle incentivization which basically means that you know whenever so your parents in case of bitcoin blockchain, but there is no notion of uncle because uncle is actually a branch that has been abandoned after the race condition that happens when multiple branches starts coming up. But eventually everybody starts building on top of the longest branch.

So the other blocks that would be called uncle or grand uncle, they now have no meaning. But here we have the incentivization for uncles because then if two of you are making the next block, eventually one of you will become uncle and then there would be some ether that you will get for becoming uncle. So Difficulty Update Algorithm that bitcoin also has but it is a different algorithm than bitcoin.

Bitcoin also changes the difficulty of the hash puzzle. This one also does that. And the notion of gas is completely not there in bitcoin. Bitcoin has notion of transaction fees. Here we pay gas. And because we pay gas, we always have to pay fees. And then the Ethereum has this notion of EVM which runs bytecode and which runs Turing-complete bytecode which of course, is one of the major differences.

(Refer Slide Time: 12:42)



Accounts and not UTXOs

- Bitcoin stores data about users' balances in a structure based on *unspent transaction outputs* (UTXOs):
  - the entire state of the system consists of a set of "unspent outputs"
    - such that each coin has an owner and a value, and a transaction spends one or more coins and creates one or more new coins
- Transaction validity
  - Every referenced input must be valid and not yet spent
  - The transaction must have a signature matching the owner of the input for every input
  - The total value of the inputs must equal or exceed the total value of the outputs
- A user's "balance" in the system is thus the total value of the set of coins for which the user has a private key capable of producing a valid signature.

So something to be said about this notion of accounts versus UTXOs. So bitcoin stores user’s balances in a structure based on unspent transaction outputs or UTXOs. So the state of the system is a set of unspent outputs. So whenever you try to make a transaction, the input transaction should come from the UTXOs, right. Because then you know that there is no double spending attempt. So each coin has a particular owner.

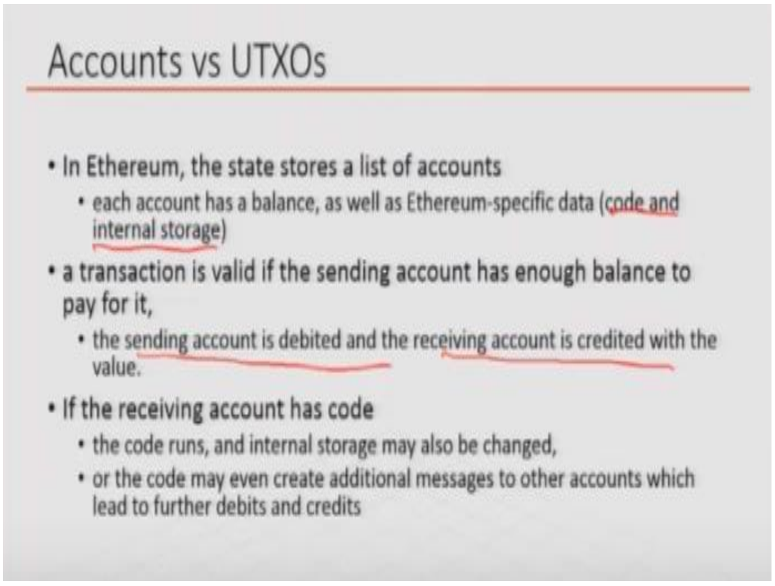
And as we know that in bitcoin, when you create a coin, when you either mine a coin, you become the owner, and that mining transaction is associated with the, your ownership, right? So when you want to spend that coin, you have to use that transaction in which you mined the coin as an input transaction. And then you have to, if you are giving the money to somebody, there will be output transactions.

And you have to give the balance to also a different address as we know because otherwise there will be replay attacks. So each coin therefore has a owner and a value and transaction spends one or more coins and creates one or more new coins in bitcoin. And the idea is that every referenced input must be valid and not yet spent.

So it should be from the UTXO. And the transaction must have a signature matching the owner for every input and total value of input must be equal or exceed the total value of outputs. If the inputs exceed the total value of outputs, the difference between input and output total will be lost forever. The user’s balance is the total value of the set of coins for which user has a private key capable of producing valid signature.

So the balance is based on the UTXOs for which it has a valid private key. So that is the bitcoin way.

(Refer Slide Time: 14:49)



Accounts vs UTXOs

- In Ethereum, the state stores a list of accounts
  - each account has a balance, as well as Ethereum-specific data (code and internal storage)
- a transaction is valid if the sending account has enough balance to pay for it,
  - the sending account is debited and the receiving account is credited with the value.
- If the receiving account has code
  - the code runs, and internal storage may also be changed,
  - or the code may even create additional messages to other accounts which lead to further debits and credits

In Ethereum, the state stores a list of accounts. And at any point, the current state has a list of accounts which are active and they have balance. So this data that is associated with the account is balance and the code if it is a smart contract and internal storage, right. So various kind of other information, the program state basically because you when you execute the program, you may change some state variables of the program.

So there will be those as well. Now a transaction is valid if the sending account has enough balance to pay for it. So it will what it will check is that for the account, it will check whether the balance is enough. It does not care whether that money came through another which transaction say. So in case of bitcoin, you identify exactly which transaction gave you that money. That becomes your input.

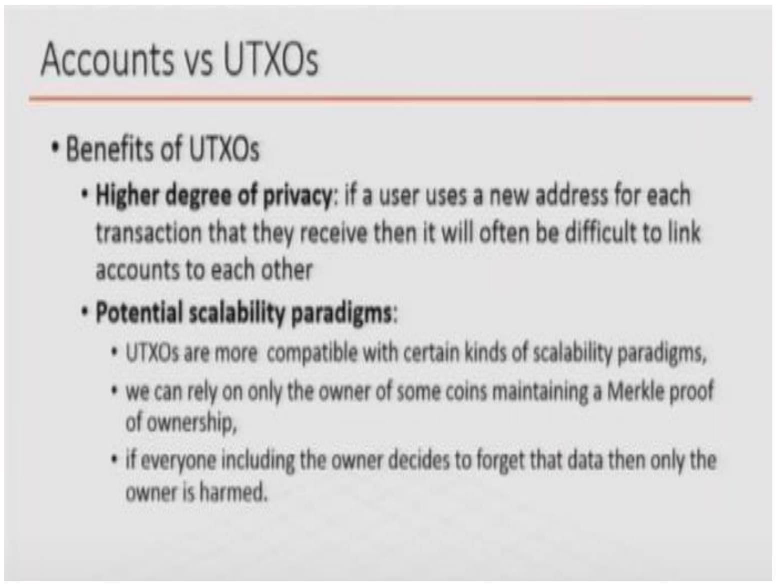
And when you create an output transaction giving money to somebody else, then you are creating a new coin that you are giving to somebody else and that somebody else then we will have that unspent transaction until it spends it. Now in this case, I might have multiple transactions, let us say one for five ether, one for two ether, one for one ether, then I will have an 8 ether balance.

And this money becomes these coins are fungible in the sense that they are replaceable with each other. I do not know which of these eight coins came from the first transaction which came from the second transaction, which came from the third transaction. The only thing that happens is the sending account is debited and the receiving account is credited with the value.

If the receiving account has a code, then it might do something else after it receives the ether, depending on what function was called to send the ether and it may run the code and there may be some state change also. For example, it might say that some flag may become true to false or false to true or it might actually have some other information that changes that can happen.

And even these there may be a chain of transactions that may be created. Because once this smart contract gets your ether, it might actually create other transactions to other contracts, send some part of the money to another account all kinds of chain series of transaction will may also ensue when you do that.

(Refer Slide Time: 17:28)



Accounts vs UTXOs

- Benefits of UTXOs
  - **Higher degree of privacy**: if a user uses a new address for each transaction that they receive then it will often be difficult to link accounts to each other
  - **Potential scalability paradigms**:
    - UTXOs are more compatible with certain kinds of scalability paradigms,
    - we can rely on only the owner of some coins maintaining a Merkle proof of ownership,
    - if everyone including the owner decides to forget that data then only the owner is harmed.

So benefit of UTXO is that it gives you a higher degree of privacy. So because remember in UTXO, every time you let us say you have 25 bitcoin, and you are giving somebody 5 bitcoin, but you give the 20 bitcoins to a new address. And then when you want to spend that 20 bitcoin, then you have to use that transaction with the new addresses private key to unlock the 20 coins.

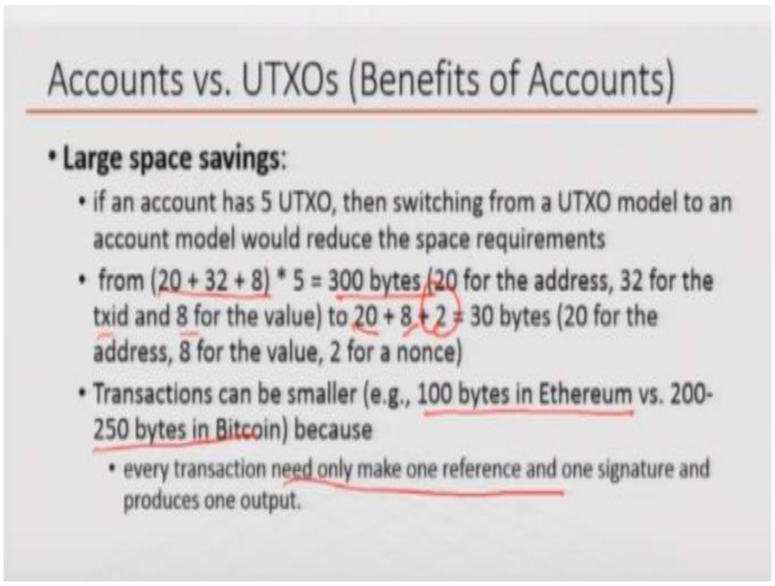
And you will let us say you give another 5 to somebody else and then a 50 and you give to a new address. So addresses proliferate and therefore, when somebody collects all the data from the past transactions blocks and try to figure out who is who and all that stuff, it becomes more difficult because same guy will have many addresses. Whereas, if you have an account, account has a given address.

So therefore, and then you make a transaction to you start with 25 ether, you give 5 to somebody, then you give another 5 to somebody. So all those things will be easily tied to you. However, obviously the account is now, account address is actually a public key in some hash form. Therefore, that does not mean that you will be completely you know deanonymized.

But in bitcoin, it is even harder to put together all this transaction to attribute all these activities to you. And also UTXOs are more compatible with certain kind of scalability. And the owner of the coin maintains the Merkle proof of the ownership which basically means that it has the signature, it has the ability to do the signature and it also has the transaction history, the hash of the root of the transaction history, which is called a Merkle proof of ownership.

And if the owner decides to forget that data, then only the owner is harmed. In case of an account, it is if you forget, then anybody who is supposed to get some activities or everybody who has the ability to make transactions on that smart contract in which the ether balance is there, then they cannot none of them can use the ether if that is forgotten.

(Refer Slide Time: 20:00)



Accounts vs. UTXOs (Benefits of Accounts)

- **Large space savings:**
  - if an account has 5 UTXO, then switching from a UTXO model to an account model would reduce the space requirements
  - from (20 + 32 + 8) * 5 = 300 bytes (20 for the address, 32 for the txid and 8 for the value) to 20 + 8 + 2 = 30 bytes (20 for the address, 8 for the value, 2 for a nonce)
  - Transactions can be smaller (e.g., 100 bytes in Ethereum vs. 200-250 bytes in Bitcoin) because
    - every transaction need only make one reference and one signature and produces one output.

However, for accounts there is a lot of space saving. So if an account if suppose your wallet has 5 UTXOs associated with it. I mean you only know because you have your wallet has all the corresponding private keys. So you know the other guys may not know that they all these UTXOs belong to you because they have different public key associated with them.

But you as an account owner would know that I have this 5 UTXOs because I have this 5 keys and they are unspent. So however to keep that information on the blockchain you need about you need 300 bytes 20 for an address 32 for the transaction ID and 8 for the value. So these are bytes and that times 5 for each of the UTXO you need 60. So 60 times 5, 300.

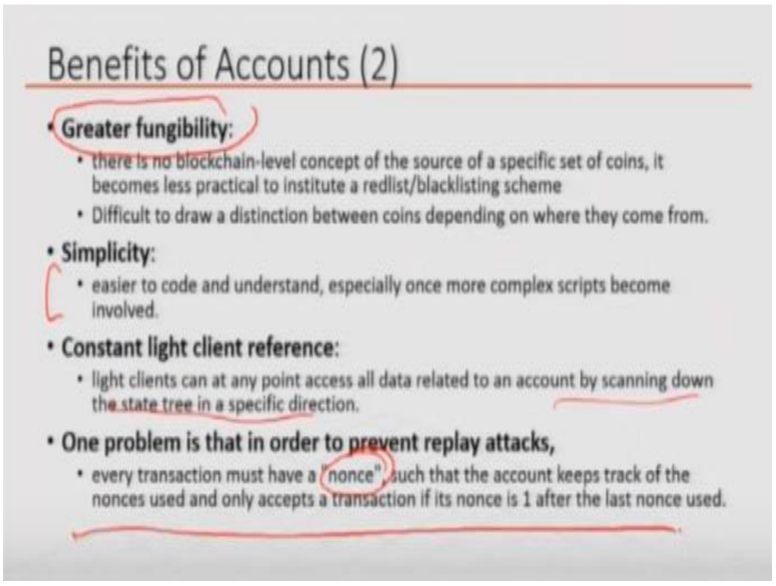
Whereas in case of an account, you do not have to make it 5 times, right, because it is a single account. So you need to know the address. You need to know the value. And you do not need to remember the transaction ID that brought that value in. And then you need 2 bytes for a nonce. Now why is nonce there? Because the problem with the UTXOs are used because we want to stop replay attacks, right?

So we actually, for every transaction we create new addresses for the balance and therefore replay attacks cannot work. But in case of accounts, you have to use some kind of a way to distinguish every transactions. So there is a notion of a nonce. Nonce is a once only number. So it is a it may be initially a random number. So this actually is incremented and therefore, you cannot replay it.

So transactions can also be smaller, so 100 bytes for Ethereum where 200 to 250 bytes in bitcoin. And because every transaction need to make only one reference and one signature, whereas in case of bitcoin every transaction requests all the let us say you want to spend 25 bitcoins. One single UTXO which has given you 25. Let us say you have 5 of them. So 5 UTXOs will be your input and each of them will have their own transaction ID and address and the amount and digital signature.

So it will require a lot more information per transaction. Whereas in this case, in Ethereum, you only have to say the account address the nonce and therefore you will need less amount of bits to do the to formulate the transaction.

(Refer Slide Time: 22:58)



Benefits of Accounts (2)

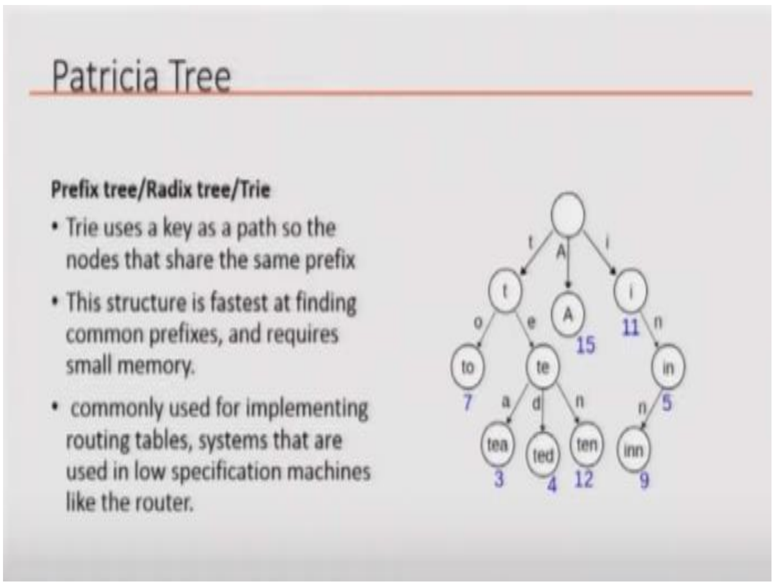
- **Greater fungibility:**
  - there is no blockchain-level concept of the source of a specific set of coins, it becomes less practical to institute a redlist/blacklisting scheme
  - Difficult to draw a distinction between coins depending on where they come from.
- **Simplicity:**
  - easier to code and understand, especially once more complex scripts become involved.
- **Constant light client reference:**
  - light clients can at any point access all data related to an account by scanning down the state tree in a specific direction.
- **One problem is that in order to prevent replay attacks,**
  - every transaction must have a "nonce", such that the account keeps track of the nonces used and only accepts a transaction if its nonce is 1 after the last nonce used.

So the benefits of accounts is greater fungibility, which we already talked about. That is all the different sources are all put together. So you do not have to remember that this came from this and this came from that. So you cannot blacklist or red list a coin. You cannot say that this coin came through wrong type of sender.

So this UTXO now get banned. You cannot do anything like that, because all these things are fungible unless you want to ban the entire account, you cannot really do anything. So coins are not distinguishable which transaction they came from. Of course, accounts are much simpler to code and understand because you know the notion of accounts from your banking and other kinds of activities.

And the light clients at any point can access all data related to an account by scanning down the state tree in a specific direction. And the replay attacks are prevented through the use of nonce. So we already talked about that each transaction is associated with a nonce. So that basically takes us to the end of the discussion on accounts.

(Refer Slide Time: 24:13)



Patricia Tree

Prefix tree/Radix tree/Trie

- Trie uses a key as a path so the nodes that share the same prefix
- This structure is fastest at finding common prefixes, and requires small memory.
- commonly used for implementing routing tables, systems that are used in low specification machines like the router.

t
A
i
t
A
15
i
11
o
e
n
to
7
te
in
5
a
d
n
n
tea
3
ted
4
ten
12
inn
9

So when we come back, we will talk about the some of the data structures used in Ethereum, as well as we will talk about the, you know give you some pointers as to how you can start making yourself more familiar with making, creating smart contracts and Solidity language and so on.