

**Advanced Distributed Systems**  
**Professor Smruti R. Sarangi**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Delhi**  
**Lecture 27**  
**Ethereum**

(Refer Slide Time: 0:17)

# Ethereum

Prof. Smruti R. Sarangi  
IIT Delhi



1

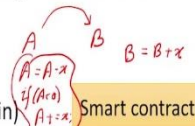
Welcome to the lecture on Ethereum. So, Ethereum is, well, it is not that new as a blockchain, but at least it is newer than Bitcoin 5 years newer. So, Ethereum is nowadays being used for a lot of other applications other than cryptocurrencies. So, at least as of 2022, when I am recording this video Ethereum has become a default mechanism for all FinTech transactions.

(Refer Slide Time: 0:47)

## History of Ethereum



- Dwork and Naor **proposed** the idea of "proof of work" way back in 1992
- Vishnumurthy et al. [2003] used PoW to **secure** a currency. The basic idea is to make it very hard to generate fake currency.
- Bitcoin came in 2008 [Satoshi Nakamoto]
- We have seen a **deluge** of cryptocurrencies after that
- Ethereum came in 2013 (first **proposed** by Vitalik Buterin)
- It is much more than a digital currency. It allows users to run arbitrary code on a **shared** global state (allows all kinds of financial instruments)
- Second largest cryptocurrency in terms of market cap in 2018 (just behind Bitcoin)
- Extremely **popular** as of 2022. Adopted to create a large number of **financial** products. Also used to create Non Fungible Tokens (NFTs) for managing digital art and collectibles.



2

So, a brief look at the history Dwork and Naor proposed the idea of proof of work, which was way back in 1992, where the basic concept behind proof of work was that, to you just cannot be a passive participant in an algorithm. So, when you say that a certain transaction is valid, or let us say a certain account has a certain balance, you need to do some amount of work before you can actually say that. So, what does this ensure this basically ensures that if 51% of all the nodes, or 51% of the work that is done is honest work, you can guarantee that the blockchain even without permissions will still work.

So, it took some time it took 11 years for Vishnu Murthy and his group to use the proof of work to secure a currency in the sense that from 1992 to 2003, the idea of using this for cryptocurrencies had not gained a lot of traction. Then Bitcoin came in 2008, Satoshi Nakamoto. But again, that is not the name of real person. It is a pseudonym. And then we have seen a deluge of cryptocurrencies as we talk. There are a lot of cryptocurrencies people actively buy them, sell them, buy futures contracts, hedge on them, and so on. So, Ethereum came in 2013, it had a different design philosophy.

So, unlike being made by one person, it was initially proposed by Vitalik Buterin. But then many people like Gavin wood, and so on, all of them came in. So, now it is a big group that actually manages Ethereum. It is much more than a digital currency, if you think about it, it has generalized the idea of a digital currency. So, when we transfer some money from one account to another, so we have account A and account B. So, when we transfer some money here, effectively we are computing the new balance  $(A - x)$  where A is the amount of money transferred, and  $B = B + x$ .

But of course, it is not as simple as this, because then we are seeing that if the new balance is less than 0, which means the transaction is not valid, then we will restore it to what it used to be and the transaction is not valid. So, if you see, for transferring a small amount, we actually need to run a piece of code like this.

So, even a basic money transfer is actually the execution of a code. And this code can be generalized in the sense, you can say that look, I will transfer 10 units, in the case of Ethereum. It is 10 ethers if the balance is less than 50. Otherwise, I will transfer 5 you could say that, so all such pieces of code are known as smart contracts.

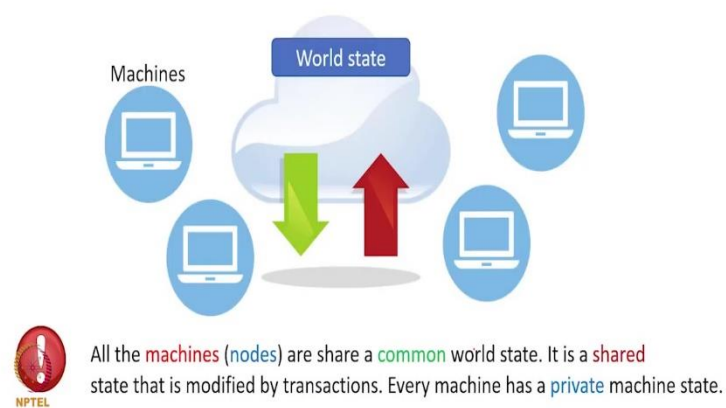
So, that is why Ethereum at least as of today is much more than a digital currency. Because it allows users to run arbitrary code on a shared global state. What is a shared global state? It is

the state of all the accounts that is the shared global state as of 2018, it is the second largest cryptocurrency in terms of market capitalization. And now of course, I do not have the 2022 numbers, but as of today, it is by far, the most popular after Bitcoin. And it is being used for doing a lot of other things as well, like create financial products, including managing digital art and collectibles. So, this also can be stored on the blockchain.

So, this idea over here is non-fungible tokens, so we will see what they are as we keep discussing, but before I go further, I would like to say that the entire discussion on consensus. The entire discussion on Bitcoin and blockchains is a prerequisite for this chapter. So, kindly go back in the playlist, look at all the discussion on consensus, bitcoins and blockchains and then kindly come back because otherwise you will not be able to follow the material.

(Refer Slide Time: 5:21)

## Machine State and World State



So, the idea is we have a world state, which as I said, is a state of all the accounts. So, in unlike Bitcoin, which did not have the notion of accounts, Ethereum has the notion of accounts. And all of these accounts together comprise or form the world state. Then of course, we have multiple machines. So, each machine then has a state of its own and local state, which is the machine state, but mind you any modification that is ever done, this change is done to the world state. Unlike a traditional cloud-based system, the world state is actually not present, in any dedicated cloud machine instead, either the entire world state or parts of it are replicated across all the machines.

So, as we know, in all blockchain based systems, each machine and essentially contains the entire state, that is probably one reason why blockchains were not popular in the earlier days,

in a prior to 2015. Because machines did not have the kind of storage to at least store the world state. But now this has changed. So, every machine will have its private state with a private state is valid insofar as the scope of the currently executing transaction. So, now, let us move on.

(Refer Slide Time: 6:48)

## Basic Concepts



- The blockchain **starts** with a genesis block.
- There are two kinds of **functions** applied to the world state
  - **State transition** functions (transfer from account *A* to account *B*)
  - Account creation transactions  $B \cdot b_d + x$  *message*
- The built-in **currency** is in Ether ( $\text{Ξ}$ ). The lowest denomination is a Wei.

Multiplier	Name
1	Wei
$10^{12}$	Szabo
$10^{15}$	Finney
$10^{18}$	Ether



4

So, the basic concepts in an Ethereum like this Ethereum is a blockchain. The blockchain starts with a Genesis block. So, Blockchain is like a linked list, it starts with a Genesis block. So, Bitcoin was the same as well and we grouped transactions into blocks. So, the blockchain pretty much contains all the transactions that are happening in the Ethereum ecosystem. So, as we have discussed, every machine maintains some state which is either a slice of the world state and some internal private machine state as well, there are two kinds of functions that each machine can apply on to the world state.

So, the first kind of state transition functions as we have seen, which is basically transfer some money from account A to account B. So, that is one kind of a state transition function. So, this is basically changing the state of the other world, why? Because we are changing the state of account A and account B, because we are changing the balances. So, this is a state transition function, the other is an account creation transaction. So, in this case, what is happening is that we are creating a new account and every new account also has some associated code with it in the sense that every state transition is simulated as a message call.

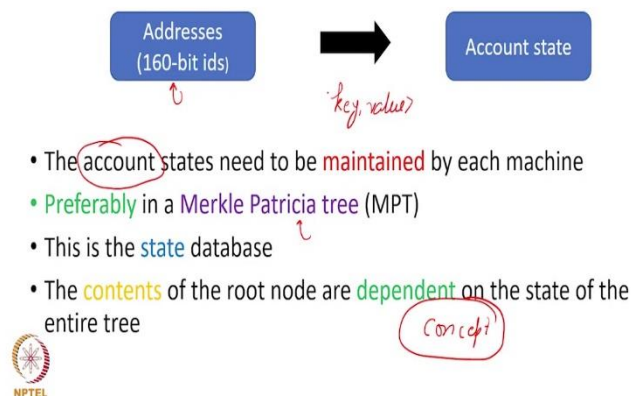
So, in this case, if let us say account A wants to add some money into account B, it will send a message to account B and account B will have some code associated with it, which will process

the message and this code was initially made a part of account B by the account creation transaction. So, that code will see that, new money is coming in what do I need to do this is what I need to do in this case. So, basically, let us say  $B.bd + = x$  where  $x$  is the amount. So, Ethereum has a built-in currency the same way we had, currency in Bitcoin.

The lowest denomination is a Wei and many a time Wei actually the protocol relies on the lowest denomination, which is a way one way and then of course, as you can see, we have some very large denominations, which are actually in a much, much larger so,  $10^{12}$  Wei's. So,  $10^{12}$  Wei's is a Szabo,  $10^{15}$  Wei's is a Finney say incidentally, they are all founding members of Ethereum and  $10^{18}$  Wei's is an Ether. So, we will see that the term ether is being used quite a bit later on.

(Refer Slide Time: 9:48)

## World State



So, the word state is like this. The word state if you think about it is a simple mapping, where every account, every account is represented by its address, which is 160-bit ids. And that is mapped to the state of the account or the account state. So, you have an account number and an account state. And this can be extended to be a generic mechanism. It is not the case that every account needs to have some amount of money or ethers in this case, but it could contain anything, it could contain digital art, for instance, it could contain insurance products, it could contain, anything that you want.

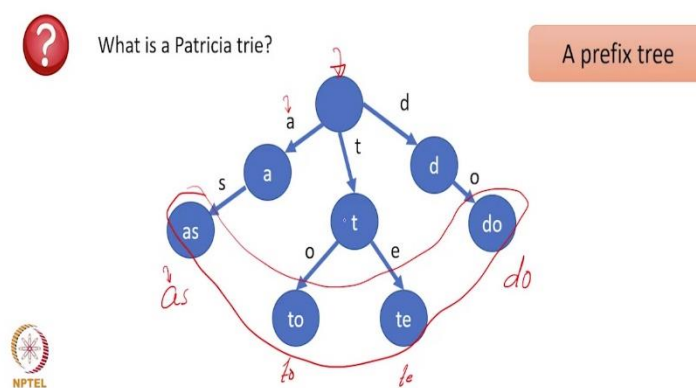
So, here, the key point is that the account states have to be maintained by each machine in the sense every machine has to maintain a database. So, this is quite similar to Bitcoin. But it is just that Bitcoin did not have the notion of accounts, Ethereum has the notion of accounts.

Furthermore, the protocol suggests a data structure, the Merkle Patricia tree, which we will look at, to maintain the state database, which is nothing but a large database of key value pairs, basically.

And so, we will have similar data structures, we had a Merkle tree and Bitcoin, if you think about it, and we will have something similar over here. So, I said the contents of the root node are dependent on the state of the entire tree. So, we will use this as the same concept, but we will use this as a common concept for looking at the account states of Ethereum. But essentially, the way that we perceive the world states of Ethereum are a set of key value pairs.

(Refer Slide Time: 11:34)

### What is a Merkle Patricia Tree (or Trie)?



So, what is a Merkle Patricia tree, or it is also called a Trie? So, let us understand what is a Trie? So, a Trie or a Patricia Trie is a data structure, which is used to store a bunch of strings. So, as you can see, let us see if the bunch of strings that I have are as 'to', and let us say 'te' and 'do' the way that I would actually store them is that I will start from the root node over here, which is over here, and create a prefix tree. So, the prefix tree so we will look at the first letter over here, match that with the first letter over here, if a and a match, then we come here and then again s and s match. So, we come here, and similarly, we look at 'to' and 'te'.

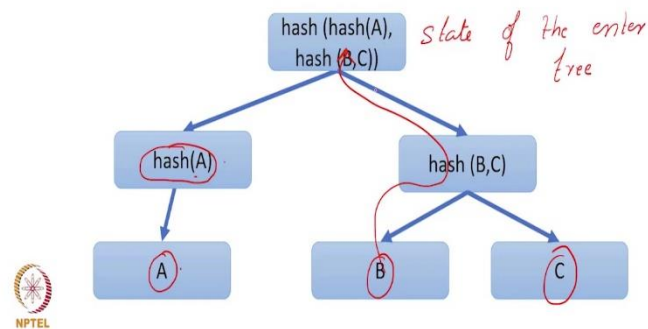
So, what we do is that we create a prefix tree over here. And so basically, the idea is that the Trie is a representation of the prefix tree where if you want to find if a given string is there or not, then all that we do is we start with the first letter and we traverse it. Optionally, this can be used to store key value pairs, how do we do that? Well, we use the same idea we treat the keys as strings, all the strings are stored in the prefix tree like this and the values are other leaves.

So, the values are where, a full string for a key terminates. So, the values are at the leaves over here. So, as you can see a Merkle Patricia Tree can be used to store key value pairs in this fashion.

(Refer Slide Time: 13:21)

## Merkle Tree

- A tree of hashes
- The root represents the state of the entire tree.



What is a Merkle tree? So, we have seen this in the case of Bitcoin as well. So, it is basically used to store a hash of a bunch of data not one data, but a large amount of data, we want to store a hash of it. So, that if there is a small change, and there are n data items, it will take order log n time to propagate the updates.

So, in this case, what we do is that we take a data item with value A, we hash the item. So, we call it hash (A). And similarly, we create a tree like this where the individual data items are B and C, we create a hash value hash (B), (C), and the root basically, so what happens is every node contains the hash of its children.

So, this node contains the hash of (A), hash (B), C contains the hash of its children and the root furthermore, will contain the hash of the concatenated value of hash of (A) and hash of (D), C. So, in this sense, the contents of the root represent the state of the entire tree. So, this is not a Patricia Tree this is a tree the earlier one was a Patricia Tree. So, this is the state of the entire tree and end state of the entire tree is represented over here in the root the contents of the roots if anything changes, then of course, it will be very easy to propagate the new change to the root in roughly order log n time.

(Refer Slide Time: 14:53)

### Merkle Patricia Tree (MPT)

- The **leaf** nodes are **<key, value>** pairs
- The **key** can be the account id or the transaction id (or hashes of them)
- The **key** is used to **traverse** the MPT tree (*Patricia tree*)
- There are two **kinds** of internal nodes: branch node and extension node
  - We **traverse** the MPT tree in terms of **nibbles** (4-bit quantities)
  - A **branch node** can have 16 children ( $2^4 = 16$ )
  - The MPT **discourages** nodes with a **single child**. An extension node **collapses** a **chain of nodes** (stores all the nibbles beyond the position of the extension node)
- The **full** Merkle tree is traversed (**DFS**) once to create a **root hash** and then the root hash is **updated** incrementally.

NPTEL

So, the Merkle Patricia tree is like a union of both the Merkle tree and a Patricia try Trei. The leaf nodes are the key value pairs, and they can be used to store anything. For instance, the key can be the account id or the transaction id, or even hashes of them. But the key point is that whatever we consider the key anyway, the MPT tree does not care. The keys used to traverse the MPT tree the same way you would traverse a Patricia Trei exactly in the same way you would traverse the tree. So, that is what the key is used for. And we traverse it a nibble at a time. So, let us see if it is an 80 bit key for instance, then we divide it into 4 into 20 nibbles what is nibble? Nibble is 4 bits.

So, what we do is that we divided into nibbles, we will have 20 such nibbles. And it will basically be a 20-level tree and will traverse it and nibble at a time. And every node because we are dealing with in terms of nibbles will have 16 possible children. So, every Branch node in the tree will have 16 possible children over here. But there is a catch. So, the catch is that if we look at it, who wants it with a level tree that is too much. So, many a time a large part of the value space will be sparse. In the value space is sparse. The MPT construction and Ethereum discourages nodes with a single child.

So, it discourages any kind of an internal structure like this. Is a much better idea that if this is the internal structure, then this actually points to this point where this is a fused node. So, the fuse node basically will indicate what are the values additional values of nibbles in this in a chain over here, for example, let us let us consider in the hash values. So, if this is A, and let us say this is 5, and this is 6, and 7. So, we will basically create one fuse node, where we will



store the fact that well, we enter it using the AH. And all the children for this position have 5. And then of course, and after that we can have branches.

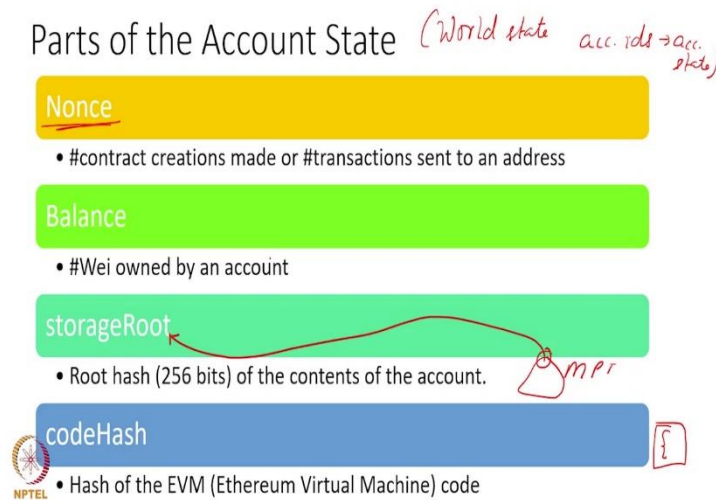
So, this fuse node is called an extension node, which is collapsing the chain of nodes into a single. So, what we will have after this is we will have this as an extension node, we will store 5 over here. And then again, we will have two of these children over here 6 and 7. So, the value space is expected to be reasonably sparse.

So, that is why we will have many of these extension nodes in the MPT tree, which will reduce the storage and as well as the traversal time substantially. So, this was a Trei part, this was a Trei aspect, the PT part of MPT tree. But also, what we do is that we have the same idea that an internal node stores the hash of its children, so on and so forth.

So, we create what is called a root hash, which is basically the hash stored at the root. And the way this is done is that we traverse this. So, basically, we, in this case, we treat the Patricia Trei as a Merkle tree, we traverse it using DFS, we compute the hash, every parent computes the hash of its children. And then finally, we have the root hash. So, the advantage of doing this is that if let us say at any point, there is a change, then incrementally, we can update the root hash, we do not have to traverse the full tree. So, we just compute the new hash for the internal node and kind of propagate it up.

So, this is why we have a Merkle Patricia tree where it is easy to add a node it is easy to search for a node as you can see, so this acts like a search tree as well. It is kind of compressed because of these extension nodes. And furthermore, we maintain internal hashes as well as the root hash. So, this completes the definition of the Merkle Patricia tree or the MPT tree. So, the MPT tree is quite important in Ethereum because it is used to store many, many things.

(Refer Slide Time: 19:13)



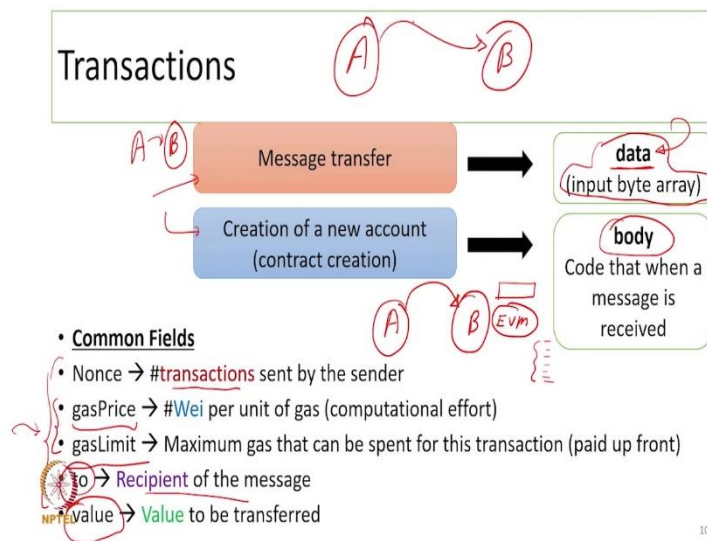
So, what are the things are given the account, mapping it to the account state is anyway what the world state is. Which is nothing but a mapping from account ids, 168 account ids to the account state. So, of course, we stored several things as a part of that count state. So, the word nonce will appear many times in Ethereum. And every time it will mean a different thing unfortunately.

So, the account state the nonce basically means the number of transactions that have been sent to an address either that is this the number of transactions that have been sent from this account, or the number of times in this account, you have created a new contract a contract is a piece of code basically, balance that is in a number of way owned by the account storage root.

So, the storage root is where we actually store the state of the account. And this is again stored as an MPT tree. So, the root hash of MPT tree is known as the storage root. And so, this stores the contents of the account or several accounts, the code hash is nothing but we take all the code that is associated with the account.

So, this code is also known as a smart contract, but we will refer it to as just code in this presentation. So, this code is anyway targeted towards something called Ethereum virtual machine, which is like a virtual machine, which is meant to run Ethereum code regardless of the platform. So, the hash of that is known as the code hash.

(Refer Slide Time: 20:58)



So, how do transactions work? So, the way that transactions work is that a message is transferred from one account to the other and the other account processes it. So, what is there in the message? Well, we can have, structured data and unstructured data. So, the unstructured data the field is data, where, you can send anything function arguments, anything that you want, in absolutely anything that you want, can be sent over here as a part of the message transfer, which is the data field of the message transfer. So, as we have discussed earlier, there are two kinds of transactions in Ethereum.

There is a message transfer, and we can also create a new account or contract creation. In that case, what we will have to do is, so since every account will have some code associated with it, what does this code do? Well, the code is invoked when a message is received. So, when the account receives a message could be any message could be credit yourself, debit yourself could be anything. In that case, the code that runs is known as the body. And this code has to be part of the account creation or account initialization process. There are a few more fields, so we have seen nonce, so nonce here is also the means the same thing.

But as I have said, we will see the term nonce appeared in several places, and it will mean different things. So, in this case, it is a number of transactions sent by the sender. So, we have two very interesting things in Ethereum. So, let me explain the motivation behind them. So, what is the key point? The key point is that everything in Ethereum is being monitored as code execution, including transferring money. Because, as I have just said, transferring money is not that simple, basically, because you will have to run a small program at the sender side, a small program at the receiver side.

So, the issue is that now we need to provide an Ethereum virtual machine to run the code as well as the language for the code. So, when you are running the code, the code may go into an infinite loop, there may be faults to the code, all kinds of issues that happen with code. See, even though the language that runs on the Ethereum virtual machine is Turing complete, we still want to charge a premium for the amount of code that executes in a sense, we do not want people to write very heavy code. So, that will slow down the entire Ethereum system. So, we define the notion of gas.

So, gas actually stands for gasoline, so they call petrol gasoline in the United States. So, basically, per computational effort are known as gas, the number of Wei we need to spend. So, what happens is that as code keeps on executing, depending upon the instructions, you are executing, how much memory and storage you are using, gas continuously gets deducted.

So, the idea here is to basically create code, which will not use a lot of gas, then there is a gas limit. So, the idea the gas limit is that look, if you have said that you will use 20 units of gas, you cannot cross that crossing, that basically means you may be moving towards an infinite loop. And we do not want the system to hang.

So, the moment you run out of gas, the transaction stops. Furthermore, for the node that is initiating the transaction, it needs to actually pay. So, the idea is that if you want to initiate a transaction, and you want Ethereum to accept the transaction, you have to pay the gas price, multiplied with the number of gas units that are actually spent.

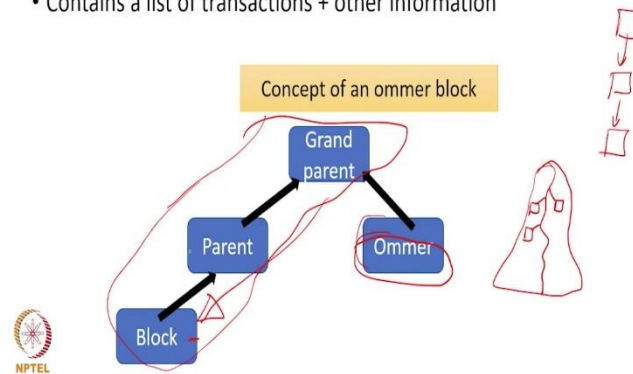
But since an upfront payment is being made, and at the moment, we do not know how much of gas will be used. So, whatever is the maximum gas amount that you are seeing, multiply that with a gas price paid upfront, later on a refund will be issued, if less gas is used, then we have the standard fields to is the recipient of the message.

And because Ethereum is still has its cryptocurrency DNA, it does have value as a common field that you are referring fair, you are kind of using it to transfer value. But if the Ethereum blockchain is being used to implement something that is not a cryptocurrency, then we need to use the data field for arbitrary data can be specified. So, the key idea, again, is that we have accounts, we have messages that are sent from accounts. And clearly in the case of currency, these are credit debit kind of transactions. But as I said, Ethereum scope goes well beyond that.

(Refer Slide Time: 25:49)

## The Block

- Contains a list of transactions + other information



So, let us now come to the block. So, as we have seen in Bitcoin, a block basically contains a bunch of transactions can contains a list of transactions, along with other information. So, here is the fun part, unlike Bitcoin, that actually always store a linked list in the Ethereum is going to store a tag. So, you basically store the fact that, well, this could be the longest chain. So, we will again discuss what is the longest chain in some detail. But it is very well possible that at some point of time, other nodes tried to add their blocks. But they were not successful in the sense that they were not part of the longest chain.

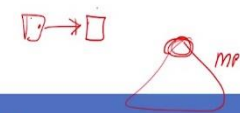
So, Bitcoin used to just discard them. But in this case, we do not discard them. In case you lose, you still get a consolation prize. So, what you do is that Ethereum kind of maintains a truncated tag, where this could be the longest chain, so we will define what the longest chain means at a later point. But again, there will be a few of these blocks, which were not a part of the longest chain, but did get added at some point.

So, what we will do is, we will remember them, and we will compensate them for the effort that was put in. So, such blocks are known as ommer. So, ommer is basically a gender-neutral term for uncle or aunt. And so, we will refer to ommer as uncle. So, if you see if this is the block, then an ommer is basically a sibling of its parent, it is still not on the chain that is validated, it is still not on the blockchain. But the fact is that an unsuccessful attempt was made to add the ommer to the grandparent, but since the parent one, the ommer got kind of thrown out.

But the good thing here is that once this block is successfully mined and added to the chain, we will kind of be nice to the ommers and give them a small amount. The small amount of currency will be given to them, kind of like a consolation prize. So, this has advantages in terms of preventing starvation and so on.

(Refer Slide Time: 28:03)

### List of Fields in a Block



Field	Meaning
parentHash	Hash of the parent's block header
ommersHash	Hash of the list of ommers
beneficiary	Who gets the fees for successfully mining this block
stateRoot	Root hash of the account state trie
transactionsRoot	Root hash of the trie (that stores all the transactions in the block)
receiptsRoot	Root hash of the trie that has all transaction receipts
logsBloom	Bloom filter for all the log topics in the transaction logs

*Handwritten notes:* "debugging" with arrows pointing to stateRoot, transactionsRoot, and receiptsRoot; "log topic" with an arrow pointing to logsBloom; "MPT" with a diagram of a Merkle tree structure above the table.

So, what are the list of fields in blocks? Of course, will have a bunch of transactions, but we have a lot of other things as well. So, of course, we have a hash of the parents block headers. So, this is what makes the blockchain and blockchain because every node in the blockchain contains the hash of its previous nodes. If this is the node, this is the spirit.

The moment you store a hash of this, this becomes a blockchain. We also maintain a list of ommers. So, we store the hash of the list of ommers. A beneficiary in a sense, if you successfully mined the block who gets the fees. Then we will have a bunch of MPT trees, so we will actually have three.

So, the MPT trees over here are known as State Route transactions route and receipts route. The State Route basically means the root hash of the account state, as simple as that, that if I take the world state, and then I take its root hash, then I am linking a block with the root hash of the world state to indicate that, once all the transactions for this block are processed, this is the world state. And given that the world state itself is stored as an MPT, it is very easy to kind of get a handle to the world state which is the root hash of the trial. Then Furthermore, a block contains a bunch of transactions, so which are also stored as a Trei.

Where, again, the key is the transaction ID and the value is the content of the transaction. So, in this case, we have this so mind you, this was there in Bitcoin as well, in a slightly different form, but the idea here is the same. So, the idea here is that the machine stores the state in the sense they store an exact list of transactions and the world state and so on.

But what actually is there in a block is basically the root hashes. Once a transaction executes at exit the code finishes executing, it will generate some return value and some logs. So, that together is known as the receipt that is again stored as a Trie, which is basically again the key value here is the transaction id and the transaction receipt. So, the root hash of that is also stored.

Furthermore, what happens is if you actually think about it in an Ethereum account a lot of code runs. So, it is possible that there may be bugs in the code, maybe it did not run correctly, maybe it had issues, exceptional conditions and so on. So, in this case, we would like to store logs. So, logs again are defined by the code, but essentially what you store is you store a log topic and some sort of a checkpoint on the machine state such that you can kind of go back and debug. So, this is stored in a data structure called a Bloom filter. So, we will discuss what is a Bloom filter in some detail in a subsequent slide.

But the key idea is that some debugging and checkpoint information is stored in the block, because the block is after all associated with executing code. So, it is important for us to understand if something went wrong, where did it go wrong, and store enough debugging information including, checkpoints and watch points. So, this information is stored.

(Refer Slide Time: 31:28)

### List of Fields in a Block – II 1 2 3

Field	Meaning
<u>difficulty</u>	mining difficulty level of the block (function of the timestamp and difficulty of the previous block)
<u>number</u>	Depth of the block (starting from the genesis block)
gasLimit, gasUsed	Total allowed gas and the amount of gas used
timestamp	Unix time() value at the block's inception
extraData	Max. 32 bytes of additional meta-data
mixHash	32 bytes of pseudorandom data (used for mining)
nonce	A 8-byte quantity (used for mining)

*mining PoW*

We have few more fields and all of these fields trust me are very important in a block. So, difficulties this is related to the proof of work concept in Ethereum. So, I will not discuss a lot about this now, because we will get some amount of time later on to discuss what is difficulty? Let us say that it indicates how difficult it is to mine the block or mining the block basically means adding it successfully to the chain.

So, we will see that this is a varying quantity. And it is dependent on the previous block and the timestamp and so on. But let us discuss this later. The number is important it provides the depth of the block starting from the Genesis block. So, if the Genesis block is numbered one, the next one is two, three, and so on.

So, that is how you decide the chain. It is a blockchain. So, every block has to have a number starting from the beginning. Gas limit we have discussed is the total amount of gas that can be used. And gas uses the amount of gas that we have used already. So, gas use has to be less than equal to gas limit. Timestamp is the time value at which the block was created. It is a Unix time value at the block's inception. In addition, we could have additional meta data because as I have argued Ethereum can be used for all kinds of applications many a time the application would need some additional data of its own. So, that feature has been provided.

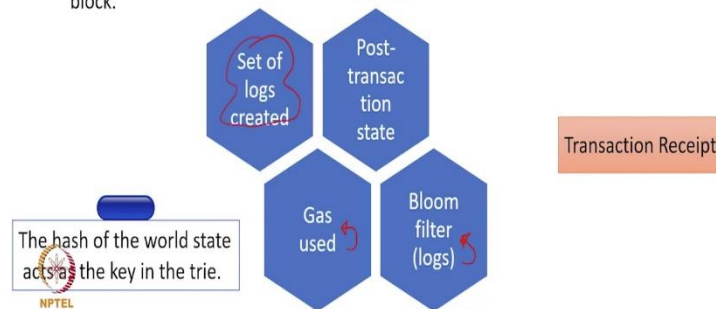
Then we have two more parameters mix hash and nonce. So, in this case, nonce is not the number of transactions and it is something totally different, we are still using the same term unfortunately. So, both of these are used to mine the block. So, both of these are used in block mining, in computing the proof of work, so we will discuss this later. But kindly remember these things as mix hash and nonce to be discussed later. The mix hash by the way, is a 32-byte quantity. And a nonce is in the 8 byte or 64-bit quantity.



(Refer Slide Time: 33:43)

## Transaction Receipts

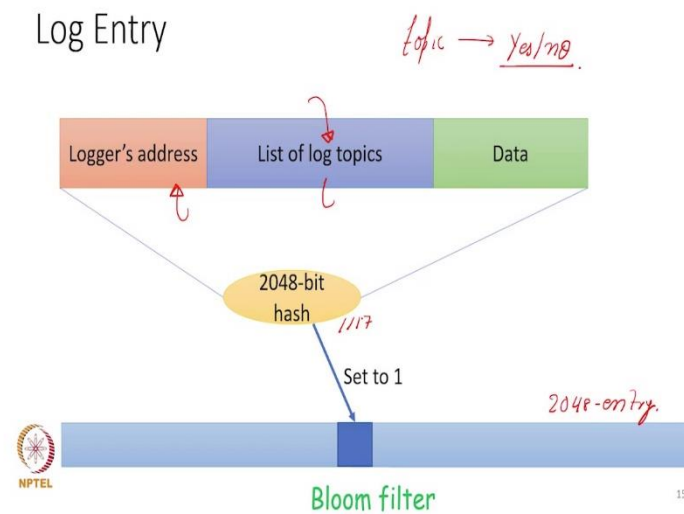
- Every transaction **modifies** the world state and (**optionally**) produces an **output**
- The **transaction** outputs (receipts) are **stored** in a trie. The root hash is stored in the block.



So, let us now come to some things that we have discussed but decided to talk later. So, we will discuss transaction receipts and the Bloom filter. So, as we have seen, every transaction modifies the world state and may optionally produce an output. For example, you may ask a Ethereum to compute the result of some function. For example, find out how many accounts are there whose balance is greater than 100 ethers, for instance. So, that can come out here. So, the outputs the transactions, the receipts are again stored in a Trie. And the root hash of that Trie is stored in the block. So, let us look at what are the transaction receipts that we look at.

So, that will have the set of logs that were created for debugging the post transaction state in the sense after the transaction, what was this world state. The gas that we used, and a dedicated data structure called a Bloom filter to store the logs. So, the debugging logs. So, in this case, the hash of the world state will act as the key in the Trie to basically find out the transaction receipt. So, you say that look, if this is the world state after a transaction, give the receipt to me. And the receipt will include all of these things.

(Refer Slide Time: 35:06)



Let us not look at a log entry. So, log entry is kind of interesting. And the key part of the log entry is that this is what is produced by machines while they execute transactions. So, is there later on and Ethereum client can query the log entries and find out if certain logs were generated or let us say for a certain log topic, the log Topic could be, your printing a message writes a message for what was actually presented, what was actually printed.

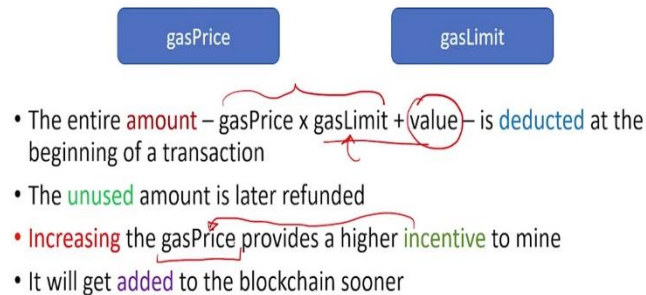
So, this could be done. So, the log entry would consist of three fields, who is the logger? What are the list of log topics? And what is the log data? So, what we do is that if let us say we want to search, you know to have some sort of a data structure to give us a yes, no answer with regards to whether a log topic is there or not.

So, whether a log topic is there or not, basically, yes, no answer. Use an indexing data structure known as the Bloom filter. So, what we will do is that we will compute a 2048-bit hash of this entire thing using a hashing algorithm. And we will take a 2048 entry array, where each entry is 1 bit.

And so, let us say the value of this is 1117. We will go to the 1117 entry over here and set it to 1 that would basically indicate that for these exact contents, I hash it and then in this entry is 1. So, what is the probability that another, set of logs are hashing to the same entry? Well, that is 1 by 2048 which is fine, the number of transactions is limited 32 to 64 in a block, then this is the risk that we can take. And so, this will basically tell us that for let us say a given log, is it present in this block or not? So, bloom filtered even otherwise is a very standard data structure, which is used to find out whether a certain piece of information is there or not.

(Refer Slide Time: 37:19)

## ? Gas and Payment

- 
- The diagram shows two blue boxes labeled 'gasPrice' and 'gasLimit'. Below them is a formula:  $\text{amount} - \text{gasPrice} \times \text{gasLimit} + \text{value}$ . A red bracket underlines the entire formula, and a red circle highlights the '+ value' part. Below the formula is a list of three bullet points:
- The entire **amount** –  $\text{gasPrice} \times \text{gasLimit} + \text{value}$  – is **deducted** at the beginning of a transaction
  - The **unused** amount is later refunded
  - **Increasing** the **gasPrice** provides a higher **incentive** to mine
  - It will get **added** to the blockchain sooner



16

Now, let us come to gas and payment. So, we have seen gas price and gas limit. So, what happens is that so this will, why was this done? The reason that this was done was to ensure that nobody monopolizes the CPU's in Ethereum, because every transaction in a Ethereum has to be validated, which means you have to run the code, we do not get into infinite loops. So, that is why the entire amount, which is the gas price X with the maximum amount of gas that can be used for this transaction + value is this + the value is not the given value + the value that is going to be transferred, transferred out of the account.

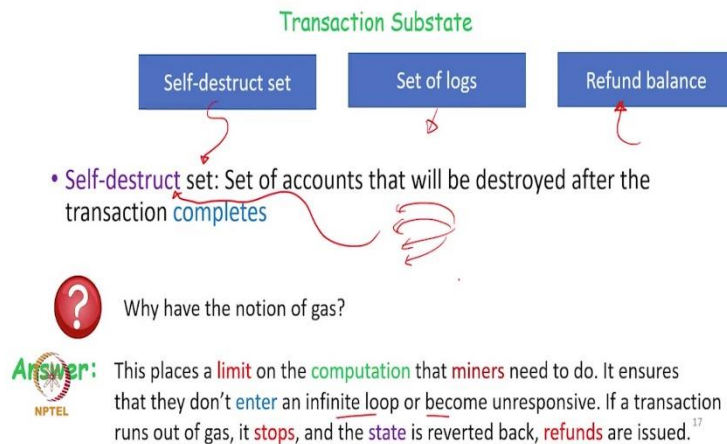
This entire amount is deducted upfront at the beginning of the transaction. Such that, we always have enough money and gas with us. And let us say we did not spend gas limit, but we spent gas limit by two, the unused amount is later refunded. And given the fact that we are in a blockchain kind of setting, the implicit trust is there.

So, we will discuss trustworthiness issues. But the main advantage of using Ethereum is that as long as a majority of the nodes are honest, you will ensure that you will get back the refund. Clearly, if the transaction initiative increases the gas price, it will provide a much higher incentive to mine. Why?

Because the incentive that miners get has been tied to the gas price. So, higher is the gas price more will miners actually mined the transaction and make it a part of a block. So, this basically means that if you want your transaction to get in, you better pay for it. And how do you pay for it by increasing the gas price that will ensure it will get added to the blockchain sooner.

(Refer Slide Time: 39:15)

## Transaction Execution



Now, let us come to the transaction execution. So, what we have seen every transaction maintains a sub state. Along with its execution state, it maintains a few other things. So, what it maintains is a self-destruct set, which is a set of accounts that will be destroyed after the transaction completes.

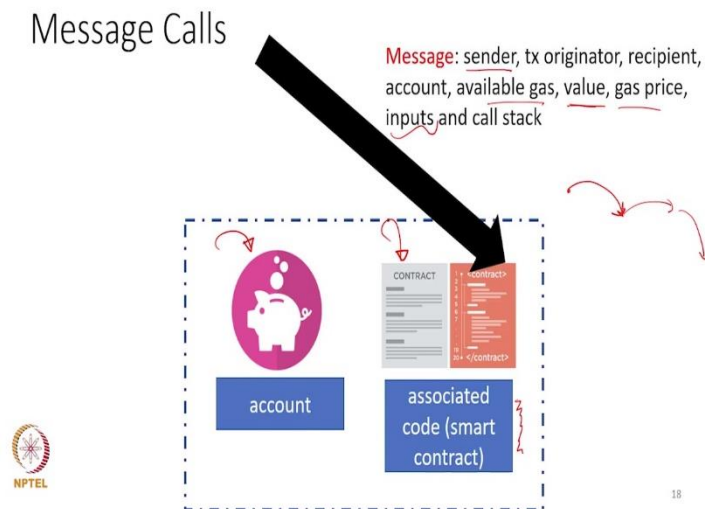
So, most of the time, a transaction may say that look, let me finish. And after I am done, you kindly destroy this account, or you will go and destroy some of their accounts. So, for example, one such transaction could be that go to all the accounts of students in a school, transfer all their money back to the school because that could be their safety deposit, and it could be the graduating batch.

So, then we destroy their account. So, other accounts will then enter the self-destruct set. We have seen the set of logs that could be generated and how to index them with a Bloom filter. And of course, the refund balance, which is the amount of users the amount of money virtual money that needs to go back to the transaction.

So, why have this notion of gas? Well, as I said, it places a limit on the computation that miners need to do. Because every node in Ethereum, or at least the majority have to successfully mined the block. Mined, the block means validate the block, validate all the transactions in the block and run the code associated with them.

So, system remains responsive does not enter an infinite loop. And, if you run out of gas, the state needs to be reverted back because it is transactions is an all or nothing transaction. So, refunds are issued. And because we are in an Ethereum, like setup, it is guaranteed that refunds will always be issued correctly.

(Refer Slide Time: 41:09)



Now, let us come to the basic idea of message calls, how does that work? So along with every account, there is some amount of code with the account. And, you could have one function or you could have multiple functions, but we will at least have one. So, that is associated code with that account, which is known as a smart contract.

So, why is it a smart contract as opposed to a dumb contract? Well, a dumb contract is something that you can say that, you can decide to kind of restrain yourself from the contract. But in this case, what happens is that something gets added to the chain, only if a majority of nodes agree. So, once an account has been added, and its associated smart contract has been added, the moment a message goes to it, the code of the smart contract has to execute. Because again, you are assuming that a majority of the nodes are honest.

So, they will do this on their own. And it will ensure that all contracts are honored. That is why it is a smart contract. So, the message will contain what will contain the usual stuff, the sender, the originator of the transaction, the recipient, the available gas, the value that needs to be transferred the gas price, additional inputs and the call stack. Why the call stack? Because it is possible that one account can send a message to another in response, a few more message calls can be made. So, we will have a call stack in this fashion and the call stacks of course after a finite depth, bounded depth.

(Refer Slide Time: 42:44)

## Execution Model

- The Ethereum Virtual Machine (EVM) is a Turing-complete machine. However, the number of **computations** is bounded by the **amount of available gas**.
- **Word size: 32 bytes**
- Maximum stack size: 1024
- **Independent** word array (for **storage**)
- Code **stored** in a ROM (read-only memory) region



19

So, all the code that we have been talking about will execute on the Ethereum virtual machine, or EVM, which per se is a Turing complete machine. So, Turing completeness is something that is studied in computer science, which means it is as powerful as a Turing machine. In formal terms, it can execute anything. But there is an important caveat over here, it is a bounded Turing machine, because we have the notion of gas, so we cannot have an unlimited amount of computation.

So, the word size over here in the Ethereum virtual machine is 32 bytes. So, that is our minimum granularity in memory. The maximum stack size is 1024 entries, it is not very large. So, it is not meant for in a huge computation. For storage, we have an independent array of words for each word is 32 bytes. And the code as such is stored in a read only memory or a ROM region. So, it is not a classical ROM machine, but the code is stored in a place so the code cannot be dynamically modified. So, it is not a just kind of code, the code is static, it does not change.

(Refer Slide Time: 43:59)

## When is Gas Charged?



- Fees **intrinsic** to the operation (different operations need different amounts of **gas**)
- **Payment** for message calls and **contract** creation
- **Increase** in the usage of memory
- **Note:** storage usage fees are **refunded** (once storage is **freed**)



20

So, when is gas charged. So, what happens is that we have some fees, which are intrinsic to the operation. So, which means that for every instruction in the code, so, the code basically is executed instruction by instruction, every instruction has a certain gas price in ways. So, as we keep on executing, we keep on accumulating, yes.

Furthermore, if there are some big calls like message calls or contract creation calls of that type, then what we have is that we have a much bigger payment in terms of gas along with that, if he ever asked for more memory more than what is initially given or provided, we ask for an increase in the amount of memory that is provided to us, then also additional gas is charged. The issue with storage is kind of tricky. The reason is that we use more storage of course, we need to pay for it, but again if we kind of free storage, the amount is refunded back.

(Refer Slide Time: 45:11)

## Overview of Execution

- The EVM is a **regular** sandboxed **execution** environment, which is **isolated** from the host.
  - It **stops** upon an exception including when the gas used exceeds the gas limit
  - Machine state → (gas available, program counter, memory contents, active number of words in memory, stack contents)
- 32-byte



21

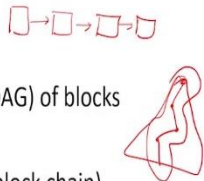
So, overview of the execution, well the execution there is nothing spectacular about that. The EVM is a regular sandboxed execution environment, which is kind of isolated from the host. So, that ensures the safety and security whenever there is an exception of any time the EVM stops and an exception could also be when the gas used exceeds the gas limit then also you stop.

Machine state as I said is maintained machine state in this case would be the amount of available gas, the current program counter of the code, the contents of the memory write number of inactive words in memory and so on. So, we measure everything in terms of 32-byte words and of course, the contents of the stack. So, that is the machine state and once the transaction is done, the machine state reverts back to null.





(Refer Slide Time: 46:12)

### Block Tree to Blockchain



- Unlike Bitcoin, Ethereum **maintains** a tree (DAG) of blocks
- The list of ommers is **maintained**
- There is no **confusion** about the chain (real block chain)
  - It is the path with the maximum **cumulative** difficulty
  - This is the **hardest** to mine
- Rewards
  - A block's **reward** is 5 Ether (**credited** to the beneficiary address)
  - Two **ommers** are maintained (**credited** 1/32th of the block **reward**)



22

In any blockchain, the most important question is how do we ensure that all of us agree on the fact that there is a single chain? So, this is in effect, solving the consensus problem with potentially faulty nodes which could be, faulty in the byzantine sense. Since, they could be malicious. How do we ensure that?

So, unlike Bitcoin, what we are seeing is Ethereum maintains a tree, kind of a DAG of blocks. We also maintain a list of ommers. So, that is also maintained. So, we also maintain, uncles and great uncles and so on, but still there is no confusion in terms of the real blockchain, which is a path in the tree. The reason is, it is the path with the maximum cumulative difficulty and we are guaranteed that there will be only one such part.

This is clearly the hardest to mine. And given the fact that this involves the maximum amount of proof of work, this is the hardest part and nobody will have any doubts on the fact that this is the hardest, and this has required the maximum amount of computational effort. So, it is possible to prove that if you have 50 + epsilon percent, it is either colloquially called 51% of the computational power is honest, then there will be no disagreement on this path. So, what are the rewards? So, in Bitcoin we had a reward for miners. But a problem with Bitcoin was the maximum amount of bitcoin that can be generated was fixed, in this case it is not fixed, it can be very high.

So, blocks reward is 5 ether, if which you had actually see is a lot given the fact that the ether is  $10^{18}$  Wei. It is credited the address of the beneficiary. And as I mentioned earlier, the ommers also get a consolation prize. So, two ommers are maintained and they are the ones who get the

consolation prize, which is 1 by 32 th each of the block reward. So, this is what they get. Typically, when a block is successfully mined, the ommers also get something.

(Refer Slide Time: 48:34)

## Proof of Work (PoW) in Ethereum

- Should be **accessible** to as many people as possible
- Should not be easily **solvable** using a custom ASIC
- Should not be **possible** to write an efficient **parallel program** to compute the PoW
- A general-purpose **computation** is "ASIC hard"
- PoW algorithm

Ethash



23

So, now, let us come to the proof of work. So, in any blockchain kind of system, the proof of work computation is kind of its key. So, what was observed in Bitcoin? So, you need to understand that, Ethereum came much later. So, it had enough time to actually see what is wrong in Bitcoin. So, some of the things in or some other design principles for the proof of work function was that in as many it should be accessible to as many people you should not use a custom algorithm or something that, somebody with a certain hardware can only do.

Furthermore, what people started doing with Bitcoin is that they started fabricating their own custom ASIC and FPGS. So, the way that the designers wanted to generate a function is that it should be ASIC hard in the sense that there is no great benefit of using an ASIC. So, friend is the benefit of using a custom chip come when you can break it down into many, many parallel components. But in general, it should involve the algorithm was made in such a way that it has a lot of sequential steps, it is very hard to efficiently paralyze it. Furthermore, it had a lot of memory accesses.

So, which also ensured that you would need a large amount of memory. And so larger systems are unlikely to help, because nobody would have that much of memory. So, this idea was that you make it general purpose, ensure that we do not you do not have adequate parallelism that will make the algorithm ASIC hard. So, the proof of work algorithm that came out of all of this was called Ethash.

(Refer Slide Time: 50:21)

## Ethash algorithm

1. Compute a seed for a block by scanning all the block headers (can be sped up with checkpoints)
2. seed → cache (pseudorandom data, 16 MB) → dataset
3. The dataset is 4-5 GB (updated every 30-100k blocks)
4. Mix the header, nonce, and a random slice of the dataset (128 bytes at a time) *mix Hash, nonce*
5. Compute multiple rounds (step (4) repeated multiple times). Final output: a compressed digest
6. Finally, verify if digest ≤ threshold. *0... < threshold*
  1. The threshold is a function of  $(2^{256} / \text{difficulty})$  →  $C$



A nonce that leads to condition (6) getting satisfied is the PoW.

24

So, the idea of Ethash like this, that we come, we scan all the block headers, so you do not have to scan it from the Genesis block but you can scan it from a checkpoint, and then we kind of hash those values, and you compute a seed. So, the seed is what creates the dependency between the previous blocks on the blockchain and the block that you want to mine and insert. So, this still involves ordered one amount of work. From the seed, we generate a cache of pseudo random data, which is typically 16 megabytes. So, you can generate this from the seed kind of run a pseudo random number generator, generate all this data.

And from this, you generate a data set, for each this is not a subset, the data set uses this and the data set is large. So, I stand corrected, the data set is large, it is several gigabytes. And this is what makes it hard to run a parallel program or run this on an ASIC or run this on a system with a lot of memory because mining was made difficult. Say every 30 to 100,000 blocks. So, this keeps on changing, evolving in a certain sense, the data set is updated, so the data set is updated. But otherwise, it remains constant for, reasonably large chunk. So, the data set is of the order of gigabytes.

So, if you would go back to the fields in a block, you would realize that there were two fields that we had discussed. And we had not, we have not said much about it, we just say that we use it in the proof of work these two when a mix hash and the nonce. So, what we do is that we take the header of the block, we compute a random nonce, we take a random slice of the data set. So, this random slice of the data set is actually the next hash, and the data set is this big thing that we generated. So, what do we do, we take a random slice of the data set and makes hash comes out of it.

We generate a random nonce, we take all three quantities, the header, the nonce, and the random slice, then what we do is we compute multiple rounds. So, we do these multiple times. And every time we compute a hash, and the final output is like a hash of hashes. It is a compressed digest. Where we have taken the data and we have kind of mixed it several times. In each iteration, we have, of course, use the same random nonce, but we have taken different slices of the data set. So, this is important to understand that 5 is essentially running Step 4 several times, the nonce remains the same for the entire process.

But the random slice changes. So, by changing the random slice several times, keeping the nonce the same, you finally arrive at a compressed digest. So, those who are familiar with cryptographic and hashing algorithms, will realize that they also work in a similar way, which is that, if you keep on data, keep on mixing and kind of permuting it for several rounds. So, this is kind of similar in principle. But of course, the way it works is quite different. But the output is that finally we kind of have a master hash that I am calling the conference digest.

So, how do you verify that this is indeed the proof of work, the way that you verify is the digest has to be  $\leq$  a threshold and the threshold is a function of is a linear function of  $2^{256} / \text{difficulty}$ . So, we have seen the difficulty before. So, now just look at it higher is the difficulty, lower is the threshold, the threshold is lower and the digest is kind of randomly distributed, it will be harder for any random nonce to lead to a digest where this will be satisfied.

If the difficulty is very, very high, the threshold will be very, very low. And the threshold is very low, then the probability of finding a nonce whose digest will be between let us say zero and the threshold, that is going to be quite difficult. And the function is so complex, it is quite hard to kind of work backwards, where you fixed the value of a digest and work backwards and find out what is the value of a nonce. So, pretty much we have to guess random values of a nonce and you have to be lucky. So that is the key idea.

So, the point is that the difficulty basically determines how long on an average it takes to mine a block. Mining a block means what? Finding the mixer hash and nonce, such that this condition is being satisfied. So, the way that the Ethereum algorithms Ethereum designers have coded their system. It is that the difficulty gradually increases with time.

And the reason is quite simple. The reason is that over time computational capability increases. So, our processors get faster, we get more memory on chip, and our memory also becomes faster. So, the same difficulty that we had, we cannot have it again, otherwise it will become

very easy to mined blocks. And the easier it is to mined blocks, the harder it is to ensure that we still have the view of a single chain.

So, that is why there is a need to keep the difficulty high. So, the difficulty also keeps increasing with time. So, the difficulty, in fact of a block is a function of the difficulty of its previous block. So, it increases. Verifying, as you can see is quite easy. Any nonce that leads to condition 6 getting satisfied, which is this. Essentially satisfies the proof of work, we will have many such nonsense that, as long as we can find a nonce, and of course, we will have the mix hash value. As long as this is these holds, we know that we are done.

(Refer Slide Time: 56:41)

## Types of Smart Contracts

- Data Feed
  - Access to the external world within the Ethereum ecosystem
  - News feeds, current time, etc.
- Random Numbers
  - Use the internal PoW mechanisms to generate pseudorandom numbers
- Cryptocurrency
  - Unlike Bitcoin, Ethereum has built-in support for accounts
  - This makes it a default choice for cryptocurrency and other kinds of financial instruments



25

## Ethash algorithm

1. Compute a seed for a block by scanning all the block headers (can be sped up with checkpoints)
2. seed → cache (pseudorandom data, 16 MB) → dataset
3. The dataset is 4-5 GB (updated every 30-100k blocks)
4. Mix the header, nonce, and a random slice of the dataset (128 bytes at a time) → mix Hash, nonce
5. Compute multiple rounds (step (4) repeated multiple times). Final output: a compressed digest
6. Finally, verify if  $\text{digest} \leq \text{threshold}$ .
  1. The threshold is a function of  $(2^{256} / \text{difficulty}) \rightarrow C$



A nonce that leads to condition (6) getting satisfied is the PoW.

24

So, as you many of you know, would have seen in this case, the verification is kind of probabilistic. So, it is not giving any party any distinct advantage, unless, of course, it can. So,

you can in principle, take many values of nonsense and run it in parallel. So, that of course, can be done. But the thing is that it is still so hard because for even running one instance of it, you need a lot of resources. And it is not possible to run either maybe more than two or three on a single ASIC because we will run out of area. But of course, if I have a supercomputer, I can do that. But then again, the algorithm is biased towards parties that have a lot of computational power.

So, we are not seeing that, 51 percent of the parties have to be honest, you are seeing 51 percent of the holders of the 51 percent of the computational power has to be honest. Types of smart contracts. So, when we can have a lot of smart contracts, so of course, let me start from the bottom we could have cryptocurrency. Ethereum the nice thing is it has built in support for accounts. So, it makes it a default choice not only for currency, but anything which is tradable.

We can use the internal proof of work mechanisms to generate random pseudo random numbers can easily be done with the mix hash. We can use it as a data provenance mechanism in the sense that we can use it to provide news feeds, current time and so on, and everybody would know that a given node has provided something and you can always, later on verify it. So, if it turns out to be wrong, we can kind of discard a certain node. So, the good thing with the smart contract and blockchain mechanism is that it is the responsibility of the community to enforce that contracts world contracts are executed, and kind of the world state is updated, honestly, subject to this 51 percent criteria.

(Refer Slide Time: 58:55)

## Scalability

- We can **maintain** periodic checkpoints
  - There is no need to store all the blocks and all the transactions within a block
- **Periodically** checkpoint the world state trie
- **Inactive** nodes could be thrown out
- Create a **hierarchical** structure
  - Smaller light-weight chains (sharding)
  - Support **parallel** blockchains (subject to **limits**)



So, now, sorry, I missed this slide on scalability. So, the issue is that the rate of transaction commit as we have seen in block in Bitcoin is quite slow. So, it takes 10 times 16 minutes to finalize the transaction in Ethereum it is in seconds, two minutes. But we can still make it faster. So, what we can do is we do not have to store all the blocks and transactions we can store checkpoints, we have seen that.

We can periodically checkpoint the world state tree in the sense if I am a new node, I want to verify that the world state is correct. I do not have to start from the Genesis block, I can start from the previous checkpoint. Inactive nodes, nodes that are not active can be thrown out, you can of course create a hierarchical structure where we divide the entire network into sub networks each network compute its own limited chain.

So, these are known as shard. So, this approach is known as sharding. And then these lightweight chains are finally joined. So, this is possible we can create a hierarchical setup to do it this is indeed possible with small level lightweight chains are made into a bigger chain. So, of course, all of this is possible. So, this has rendered Ethereum quite practical, quite scalable and quite popular as of today.

(Refer Slide Time: 60:15)

## References

- Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* 151.2014 (2014): 1-32.
- Dameron, Micah. "Beigepaper: an ethereum technical specification." *Ethereum Project Beige Paper* (2018). Dameron, M. (2018). Beigepaper: an ethereum technical report



27

These are the references so the main references of course Ethereum project yellow paper. So, this is always a document in progress. So, currently our proof of work is kind of computational proof of work. So, Ethereum is moving to Ethereum to which will be proof of stake-based proof of work. So, once that comes again, maybe another video is due from my side. So, the yellow paper is basically for implementers. It is reasonably technical. If you want something lighter, you can go to the beige paper, which is an easier read.