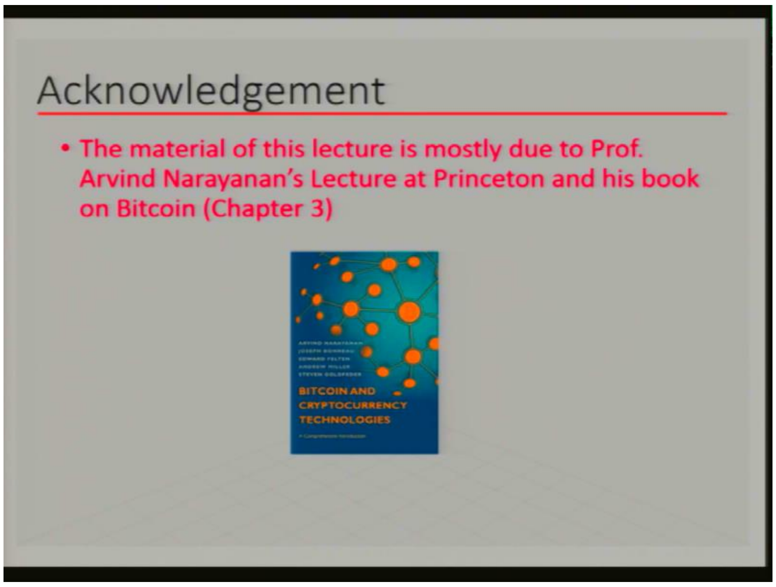
Introduction to Blockchain Technology and Applications

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

Lecture No. 09
Blockchain Technology and Applications

Welcome back to another session of blockchain technology and applications. In the last few sessions we have learned a lot about bitcoin consensus mechanism. But we have not seen the details of transactions and how the transactions are validated by the different participants in the blockchain. And we also have not seen the how the blocks are validated. So, today, we will talk about the blockchain, bitcoin blockchain mechanics.

(Refer Slide Time: 00:53)



Acknowledgement

- The material of this lecture is mostly due to Prof. Arvind Narayanan's Lecture at Princeton and his book on Bitcoin (Chapter 3)

BITCOIN AND
CRYPTOCURRENCY
TECHNOLOGIES

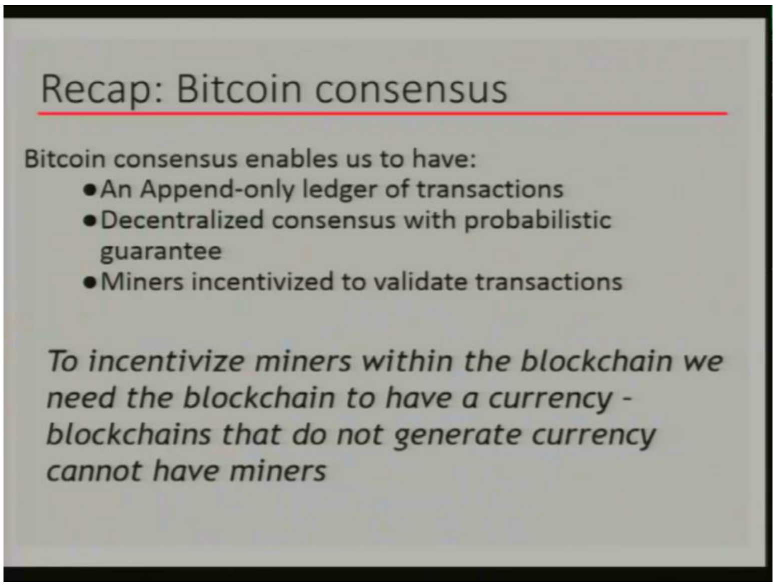
So, today's talk is entirely taken from the lectures and the book by Professor Arvind Narayanan of Princeton University. And it is a very interesting book and if you are interested in learning all about Bitcoin blockchain and other alternative crypto currencies, then this is a very good book, I would recommend that to you.

(Refer Slide Time: 01:12)

Mechanics of Bitcoin

But we will be taking only selected parts of the book that is necessary for you to understand the basic mechanics of the Bitcoin blockchain. So the consensus that we have seen allows us to have an append only ledger of transactions. And the consensus is decentralized.

(Refer Slide Time: 01:44)



Recap: Bitcoin consensus

Bitcoin consensus enables us to have:

- An Append-only ledger of transactions
- Decentralized consensus with probabilistic guarantee
- Miners incentivized to validate transactions

To incentivize miners within the blockchain we need the blockchain to have a currency - blockchains that do not generate currency cannot have miners

It is a distributed algorithm, but we overcome the impossibility results in the presence of adversaries or in the presence of malicious users. Because we are using probabilistic guarantee and eventual consistency also we are using incentive mechanism and incentive mechanism has 2 purposes. One is what would I do go and do the validation of your transaction? So that is solved by giving incentives.

The other problem, as we discussed is that there might be malicious adversaries as participants because this is a sort of pseudo anonymous network and it is very easy to enter the network. So therefore, we use some mechanism for incentivizing the users to behave according to rules. So, one thing that you must note here is that Bitcoin blockchain is basically a blockchain for crypto currency creation and crypto currency transactions.

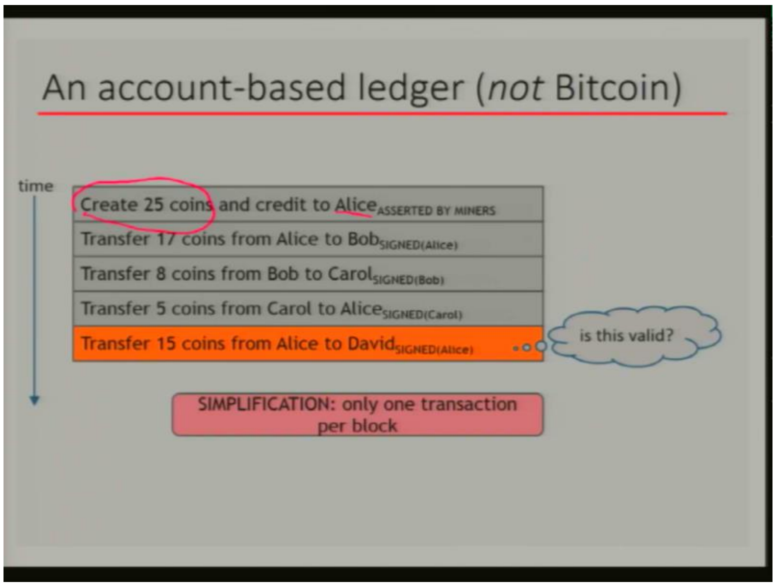
So if you have a blockchain and we will see many blockchains that did not have a native crypto currency, and if they did not have native crypto currency, such incentive mechanism would not be available to us and there, we have to worry about other ways to solve the problem of validating transactions, validating blocks and also creating the blocks that does not make the blockchain go in all possible directions and keep the consensus branch most longest branch. And therefore, make the blockchain integrity intact.

(Refer Slide Time: 03:30)

Bitcoin transactions

So, what we mean by bitcoin mechanics. Bitcoin mechanics mostly is about how transactions are done, what are the data structures that are used for denoting a transaction and what are their purposes? Why do we need those kind of increase in the blockchain transaction structure and also how people, the miners that are or other members of the participant other participants validate the transactions and also we will look at the blocks and block validation. So first let us look at the bitcoin transactions.

(Refer Slide Time: 04:13)



An account-based ledger (*not* Bitcoin)

time

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

is this valid?

SIMPLIFICATION: only one transaction per block

So first thing one would think having been used to the accounts based ledger is that one might consider an account based transaction and an account based transaction will look like this that suppose I have a transaction that creates 25 coins and I say that who it goes to, and this is asserted by the miners who create this 25 coins. But then let us say Alice transfers 17 coins to Bob and science digitally sign the transaction.

And then Bob transactions transacts 8 coins to Carol and signed by Bob, and then 5 coins are given by Carol, back to Alice. Now how many coins now Alice possess. So there are 25 coins in the beginning which Alice got, she gives 17 to Bob, so she had 8, and then the next transaction does not involve her. But then the transaction after that gives 5 coins to her so she had 8 and now she has 5 more so she has 13.

But in order to know that Alice now has 13 we have to go back to the for very first transaction and work our way down in order to see where Alice got a positive inflow and where she got a you know outflow. And then we can also consider these as if these are the blocks. So now, let us say at this point, Alice wants to transfer 15 coins to David, so all the participants need to change the validity of this.

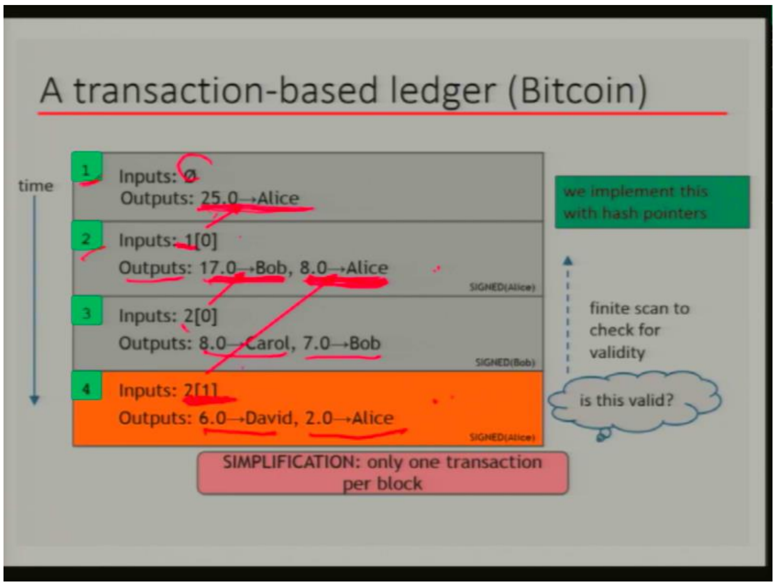
And in order to check the validity, they may have to go back to the first transaction where Alice at 25, then they, find that next transaction, Alice gives 17. So she has 8, then next one she does, it does not involve her, and the next one, she gets 5, so she has 13. So she cannot give 15 coins to Alice. So it is an invalid transaction. So in order to decide that, we have to go through all the transactions that happened before.

And if you consider each of this transaction as a block, we are basically going all the way to the genesis block to find out whether a transaction is valid. Now, this is going to be very difficult. Now, as we saw in our prototype blockchain that we actually discussed and given size on that, in that we were cashing this balance information. So every time we validate a transaction, we update the balance and that is also possible to do.

But that requires a lot of more work for every participant because every participant needs to validate these transactions. So therefore, if they have to cache all this information, then they have to do a lot of work. And then there are a few other complications because sometimes the blockchain will diverged a little bit and then they converge back to the main consensus chain. So, therefore, those things becomes a little bit non deterministic.

So that is not a very good solution to do. In our example, that was left to us exercise. There, we were only one party that was running the entire program, it was not replicated all over the places. So there was no non determinism involved. But when you have, you know, 1000s of participants involved and each of them are independently doing all this validation of transactions and also keeping track of the balance there would be some inconsistency creeping in and therefore, there could be some problem. So, therefore, this is not what bitcoin does.

(Refer Slide Time: 08:11)



A transaction-based ledger (Bitcoin)

time

1 Inputs: Ø
Outputs: 25.0→Alice

2 Inputs: 1[0]
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)

3 Inputs: 2[0]
Outputs: 8.0→Carol, 7.0→Bob
SIGNED(Bob)

4 Inputs: 2[1]
Outputs: 6.0→David, 2.0→Alice
SIGNED(Alice)

we implement this with hash pointers

finite scan to check for validity

is this valid?

SIMPLIFICATION: only one transaction per block

So, what bitcoin does is actually a transaction based ledger. So they only keep track of transactions not the account balances. So, the first transaction for example, this is a genesis transaction and what is happening is that there is no input to this transaction, but the output says that give 25 coins to Alice. Now, at this point, the next transaction only Alice can use. Now, what Alice has to do now is that she has to refer to the transaction that gave her the 25 coins.

So, transactions are numbered so, these are transaction IDs. So, she refers to the transaction ID 1 and then in this case there are only there is only 1 output so the outputs are and inputs are basically can be thought of as Alice. So, she refers to the first entry of the output of transaction 1. So this means I am inputting for from transaction 1 and only way I can input something from the transaction 1 is from the outputs of the transaction.

So the output number is 0, so there was only one output here. So, output numbers 0 is 25 coins going to Alice. So, and then she says the, my output has 2 entries. So, this is the entry number 0 and this is the entry number one where in the entry number 0. I give 17 to Bob and give 8 to myself to a different address. Remember the change address concept. So, Alice creates a change address which is different from her original address.

And then she gives that money to herself because otherwise there was a replay attack that we discussed before. So, now the next time there is a transaction. So, in this case Bob is making the transaction. So, Bob is refers to transaction number 2 and entry numbers 0 in the outputs of the transaction number 2, which basically gave him gave him 17 coins. So out of the 17 coins he gives 8 to Carol and then gives to one of his change addresses 7 coins.

So therefore, now Bob has given a casual 8 and himself back 7. Now in this one Alice again mo wants to make a transaction. Now she has transaction she got money in transaction 2 she cannot use transaction one anymore, because transaction 1 output has been spent. So transaction 2 is the only a transaction where she got money. So from herself only but she got money there. So she refers to the entry number 1 of the outputs of transaction 2.

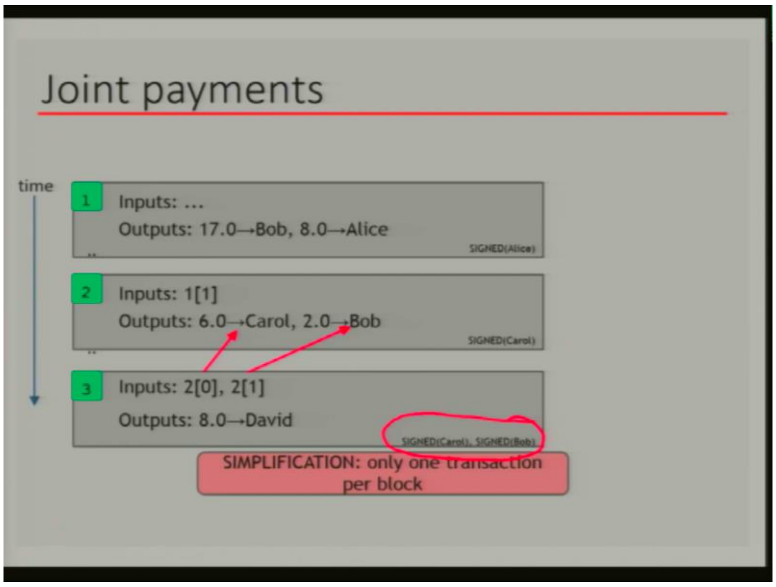
So that that is indexed with one, and there she gives 6 to David and then to change address. So this is the way bitcoin does this transaction keeps track of the transactions. And note that in this case, we want to see whether this transaction is valid, all we need to do is to look at the transactions she refers to as a source transaction those transaction is this. And we see that in that source transaction in the output number 1, she got 8.

So that is all we need to check. So now you see that we did not have to go all the way back to all the transactions, we also have to check whether Alice has used this transaction 2 index 1 anywhere between this transaction and this transaction that also has to be checked. So therefore we have to only check a fraction of the transactions and not all the transactions all the way up to the Genesis transaction.

So that is actually the Genesis, it is not called the Genesis transaction. It is called the coinbase transaction. So coinbase transaction or transactions in which we create some coins, and those transactions did not require any inputs, coins are created in those transactions. So those are called coin based transactions. I was calling it intentionally Genesis transactions but it is not really a Genesis transaction. Genesis block is a concept not Genesis transaction.

And Genesis block means the first block of the blockchain in which the blockchain started. So we will talk about that later. But what we also do is that we create hash pointers to these different transactions so that we can easily trace back when somebody refers to a transaction, we can quickly refer back to the transaction that we are interested in.

(Refer Slide Time: 12:45)



Joint payments

time

1 Inputs: ...
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)

2 Inputs: 1[1]
Outputs: 6.0→Carol, 2.0→Bob
SIGNED(Carol)

3 Inputs: 2[0], 2[1]
Outputs: 8.0→David
SIGNED(Carol), SIGNED(Bob)

SIMPLIFICATION: only one transaction per block

Now, other thing we can do with this mechanism is that suppose I have multiple change addresses on many transactions. I got different number of points into same number of points, it does not matter. I got some points on multiple chain of change addresses. But it is very difficult for me to keep track of all those change addresses and how much I have in all those.

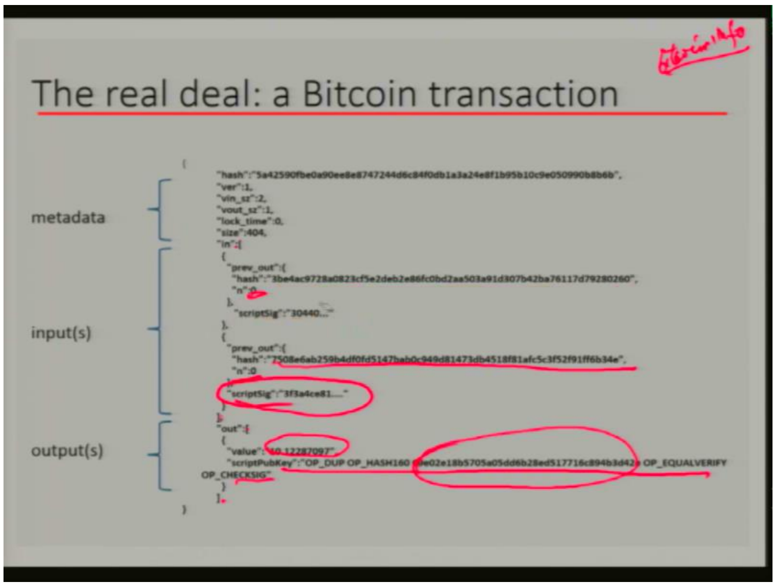
So time to time I may want to merge those change addresses, the contents of those change addresses into a single address so that that is what is called a margin.

And that is what we can do using like to input transaction so one from here and one from here. So, you see that the Bob can consolidate so here Bob got 17 in this transaction and Bob got 2 in this transaction. So she he should ideally have 19. So he actually can create that by putting in a transaction where he refers to transaction number 1, output 0 and transaction number 2, output 1 and then she says that give 19 to me.

Remember requirements in the input transactions, these are called the input transactions to this transaction and these are this is an output of this transaction. So this is an input of this transaction, is an input of this transaction and this is the output. So output can be can be also multiple and input can be also multiple. And that is what we are seeing here. So now we can also do joint payments. That is if I did not have enough coins myself, and but I am under friend together decide that we will be using our coins together to buy something, we are allowed to do that.

So the joint transaction looks like this. So let us say I have 2 transactions here. Carol has 6 Bob has to and they together decide to give to David so they can put in a joint transaction. But remember that they have to both sign this transaction because Bob or Carol, themselves by themselves did not have the right to spend both of these outputs. So they have to do a joint signing we will see how that works in a little bit.

(Refer Slide Time: 15:03)



The real deal: a Bitcoin transaction

metadata

input(s)

output(s)

```
{
    "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":0,
    "size":404,
    "in":[
     {
      "prev_out":{
       "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
       "n":0
      },
      "scriptSig":"30440..."
     },
     {
      "prev_out":{
       "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
       "n":0
      },
      "scriptSig":"3f3a4ce81...."
     }
    ],
    "out":[
     {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY
  OP_CHECKSIG"
     }
    ],
}
```

Now in reality, so, we give you the schematics of what the transactions look like when in reality this is what happens this is the structure of a transaction and if you actually go into like you know some real browser this bitcoin dot info for example, and then you can browse transactions and you will see things like this. So, what the transaction in the transaction structure what we have are the following.

So, we have metadata. So, this is the metadata this is the data about the transaction, this is these are the inputs. So you see that the input is an array, so this array is denoted by this and finished by this and then output is also an array in this case, I this in this particular transaction, there is only 1 output and I see there are 2 inputs. So in the Meta data, what we have is the hash of the transaction.

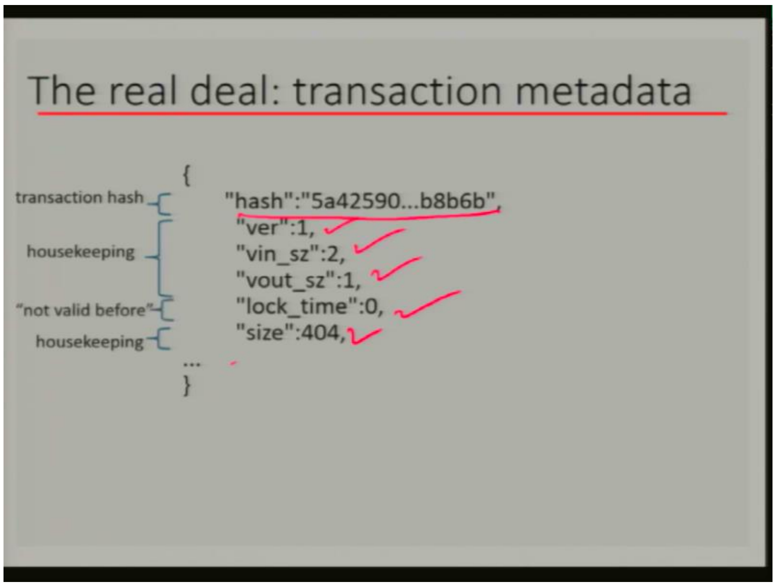
And then we have the version. This is the Bitcoin version. This says how many inputs are there. So this is the input size, this is the output size. So this is 2 and 1, there is something called lock time and we will discuss very soon. What lock time means so for now, we can ignore it accept the fact that lock time zero means this transaction can be done immediately. So that is what lock time zero means, the size of the transaction.

And then in the input, we have the hash of the transaction that is the input. And in a zero here means that this is the, zeroeth output of the transaction. And then there is a signature. Now, how this signature is done. We will we will defer that discussion for later a little bit later, but this is kind of like a signature you can first now assume, similarly the second output also has the transaction hash for the previous second previous transaction.

And then this is the, zeroeth output of the transaction and here is sort of the signature in the output, you say, what is the value of the value that you are sending. And this basically says the address of the person whose account is getting the funding the getting the money. Now, here, you see that there is a lot of things, not only just account number, the account number is basically somewhere here.

But the rest of it seems like, you know, sort of like an assembly program. And we will see what that is in a little bit. So that is what the transaction structure looks like. So you get some idea about what this looks like.

(Refer Slide Time: 17:46)



The real deal: transaction metadata

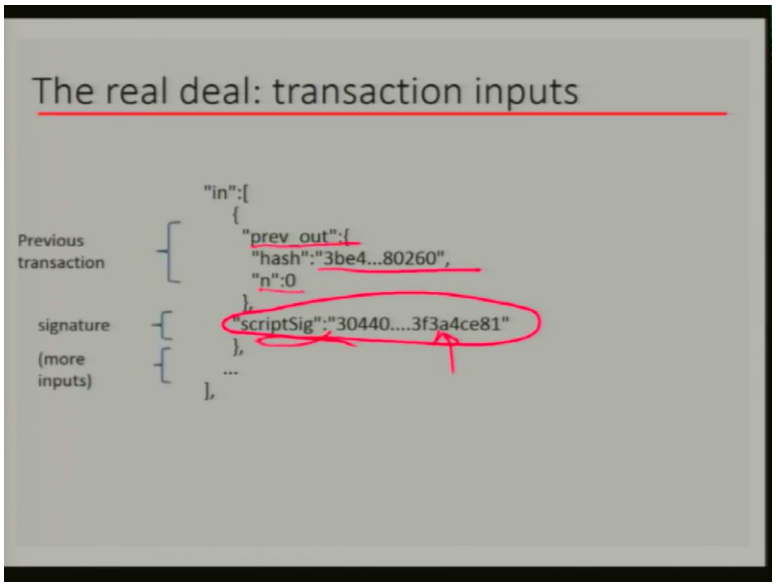
transaction hash
housekeeping
"not valid before"
housekeeping

```
{
    "hash":"5a42590...b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":0,
    "size":404,
...
}
```

Now, let us look at this metadata and other inputs and outputs in a little bit detail. So, as I said that the metadata part has hash of the transaction, the version number, the size of the input array, the size of the output array, the lock time zero means it is an immediate transaction and the size of the transaction and there may be some other data that might be there. So that is a Meta data. So all of you know Meta data means that it is a data about the data.

So this is the data about the transaction. So this is what kind of says a housekeeping information. And this is the transaction hash. And lock time gives you when the transaction becomes valid, zero means it becomes valid immediately.

(Refer Slide Time: 18:38)



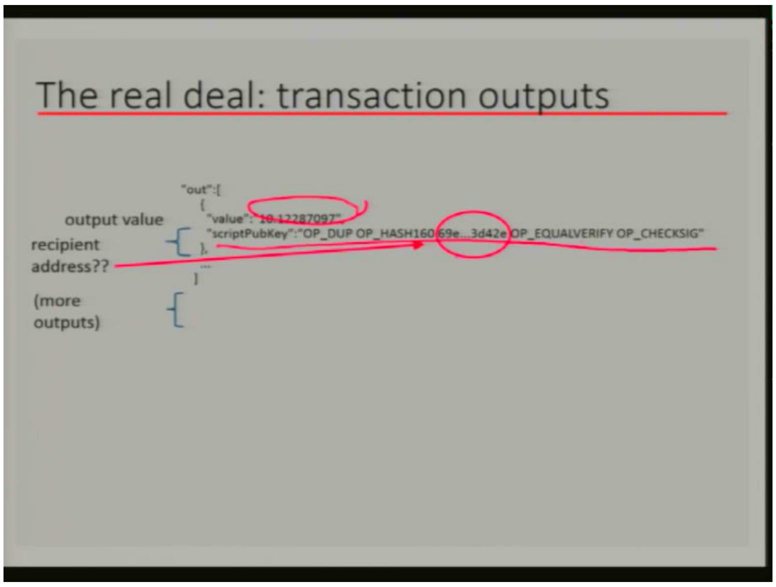
The real deal: transaction inputs

Previous transaction
signature
(more inputs)

```
"in":[
    {
      "prev_out":{
        "hash":"3be4...80260",
        "n":0
      },
      "scriptSig":"30440....3f3a4ce81"
    },
    ...
],
```

The input looks like this. So you have the previous output transactions hash here. So and then this is the index of the output of the transaction. And here is the signature. And we will we will see what that means. This is actually a public key. And this is a, you know, script that we will understand in a little bit.

(Refer Slide Time: 19:05)



The real deal: transaction outputs

```
                        "out":[
                          {
      output value          "value": "10.12287097",
recipient             {     "scriptPubKey":"OP_DUP OP_HASH160 69e...3d42e OP_EQUALVERIFY OP_CHECKSIG"
address??                 },
                          ...
                        ]
(more                 {
outputs)
```

So the output also has the value of the output, how much how many coins are going and, here it says who is going to be created for that amount. And now, as we said that this kind of looks strange, but will very soon understand what it means. And this part of that is actually a hash value is the account number, but we will understand very soon.

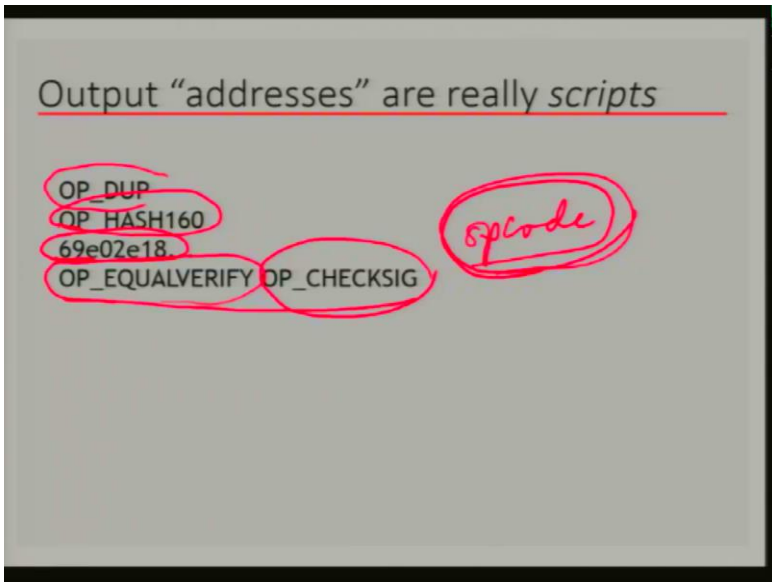
(Refer Slide Time: 19:34)

Bitcoin scripts

So bitcoin scripts is what those things are so those were mysterious things that you saw, like what scripting and in our in the output it says script pub key, and in the input, it says, crypt

SIG. These are actually what we call bitcoin scripts. So bitcoin script is actually kind of like an assembly program, except that assembly programs are meant for real Hardware machines, but in this case this script language actually has an idealized machine on which it is supposed to be supposed to be executed. And this is a stack based machine on which this is executed. So let us see how that works.

(Refer Slide Time: 20:16)



Output "addresses" are really *scripts*

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

So, first of all, like any assembly code, it has top codes. So it has popped up for example, is not is an opcode which says the duplicate something that is on top of a stack, so we have a stack based model of computation. So, data remain data is pushed into the stack. And then at that point, if we get this instruction on top, then we create a duplication of the top of the stack on top. So if there is x on the top of the stack at the time.

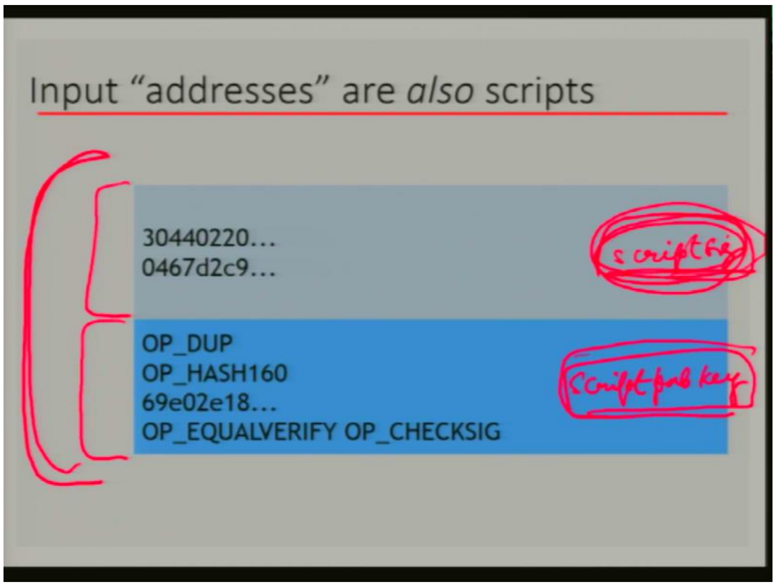
I encountered this instruction, then I will write another X on top of it of hash basically hashes, the whatever on the top of the stack, so, it basically does a 2 layer hashing first it does SHA256 and then it does a 160 bit hash the output is a 160 bit hash. And here is some data. We saw in this case, we said that this was actually a hash value, but it could be any data. And then this says, op equal verify object SIG.

So, op equal verify a bit verifies that the top 2 entries on the stack are equal. And then object SIG basically checks whether the signature checks out. So if you have the signature, and if you have the public key, then if you apply object SIG, then it will verify that the signature is according to the is decipherable with the public key that means it is it was created by the private key have the corresponding public key.

So these are called off opcodes of the language. So, what we are seeing here is that there is a language and this language is called a Bitcoin script language and this language is very simple and it has the semantics of this language is stack based which means that we will assume that way the programs written in this language are executed is by reading one instruction and then if the instruction is data, then push it onto the stack.

If it is an actual instruction opcode then we take the things on top of the stack, maybe the only the top entry or top 2 entry or top 3 entry depending on the instruction and then we execute on that data that instruction and then we whatever it is the result we remove the entries that on which you apply the apply the instruction and then you push the result on top of the stack if there is a result. So that is how the semantics of this language works.

(Refer Slide Time: 23:05)



Input "addresses" are *also* scripts

```
30440220...
0467d2c9...
```

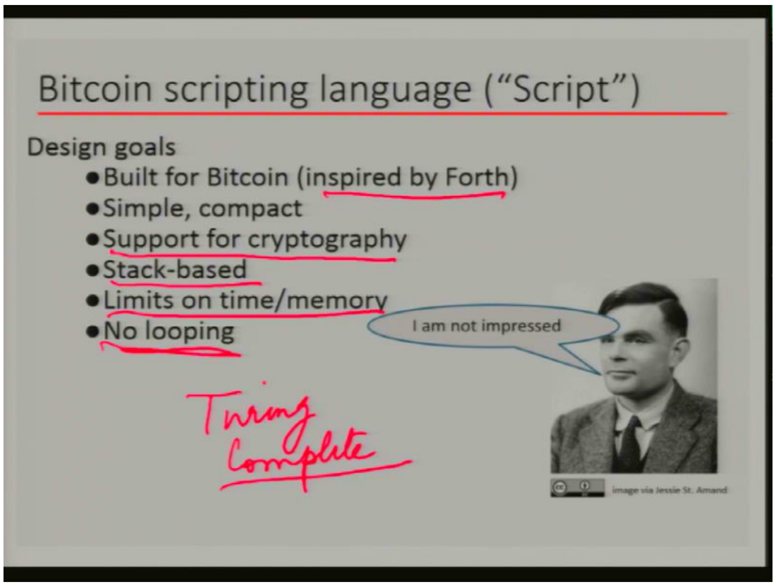
```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

So it turns out that when we say scripts rig that actually is a script. So we would think that in the input transaction, so when I use a transaction as my input, so what I am doing is that I am looking at a previous transaction in which I was given some money. And I am going to sign that the thing that I am now signing to use that money, so it is called redeeming the input transaction, the output of the previous transaction.

So to redeem, I need to give a signature but instead of a plain signature, I give a script. Similarly, in the output, I saw that I am supposed to only say, whose account which should go to, but instead of giving just the account number as a hash of a public key, I have to write a script and that is called a Script pub key. So we will see how this, this thing works. And, then whoever is going to validate the transaction has to put script on top of this script.

So script is basically a set of instructions and data values. And this is also a set of instruction and data value. So I concatenate them. So I have first these instructions, and then I have this instructions, I get a longer script. And then I have to execute that script. And in a stack based system, and when I execute, if the execution goes to well, then it is all fine. The transaction validates if the execution throws an error that means something did not work out. That means either this script has a wrong information or this script has the wrong information, and therefore the transaction is invalidated. So that is the idea of this scripts.

(Refer Slide Time: 24:52)



Bitcoin scripting language ("Script")

Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based
- Limits on time/memory
- No looping

I am not impressed
Turing Complete
image via Jessie St. Amand

So the goal of the bitcoin scripting language is actually to have a simple, compact way of validating the transactions and we want to keep the machinery to validate the transactions. Very, very simple. That is why the language is also very, very simple. It has instructions to support cryptography. So it has single instruction to compute a hash, it has single instruction to validate a signature. So this is actually in that sense, the instructions are pretty powerful, but it is tag based.

And therefore an every instruction is executed in just once. And therefore the time and memory is basically linear in the size of this script. And therefore it is pretty limited. Because we did not want to see the scripts are written by somebody else, because when I are validating a transaction, the transaction script SIG, or script pub key, is written by the person who actually put the transaction in now is the malicious user.

And he had a time consuming instruction there, then you all the validators will actually be getting into a huge time consuming computation and they will delay or validating all other transactions. So, we want the language such that it is not possible to get into an infinite loop or very, expensive computation. So, this language has no loops in the in its structure and therefore, it is not turing complete.

So, Turing complete languages like all programming languages that we normally use are called turing complete languages. So, if you have taken a course in theory of computation, you probably know what that means, but turing complete languages can design or can express very, complex functions, whereas, this kind of a very, simple language With no looping cannot express any arbitrary functions and therefore, and that is by design.

So that we can transaction validation cannot be attacked by a malicious user by putting in a transaction where he puts a script which goes into an infinite loop and things like that. And it is inspired by previously available language called forth. We did not need to worry about that language, but we have to understand the very basics of this language. Now, why are we talking about this language that come when we come back? We will talk about that.

And we will be actually discussing the concept of smart contracts in this context and then you will understand why even though we are not interested in crypto currency, we are talking about bitcoin scripts. So, we will come back.