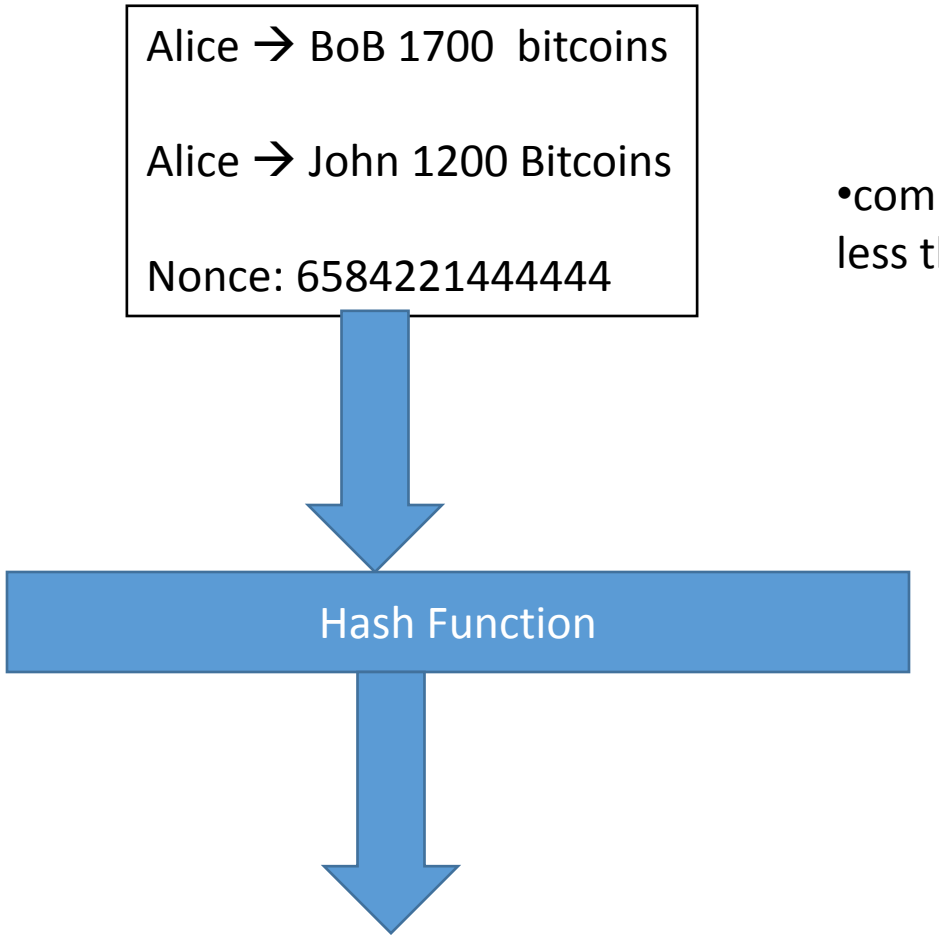
Blockchain Hash Function

Important: The validation process for blockchain transactions relies on data being encrypted using algorithmic hashing.

A hash function takes an input string (numbers, alphabets, media files) of any length and transforms it into a fixed length. The fixed bit length can vary (like 32-bit or 64-bit or 128-bit or 256-bit) depending on the hash function which is being used. The fixed-length output is called a hash. This hash is also the cryptographic byproduct of a hash algorithm. In the blockchain we normally use SHA2.

Solving the hash starts with the data available in the block header and is essentially solving a complex mathematical problem. Each block header contains a version number, a timestamp, the hash used in the previous block, the hash of the Merkle Root, the nonce, and the target hash.

Hash creation



7a22f1d90448953194eaf71477f
67ccdf406f406f42cb1628117

So to recap:

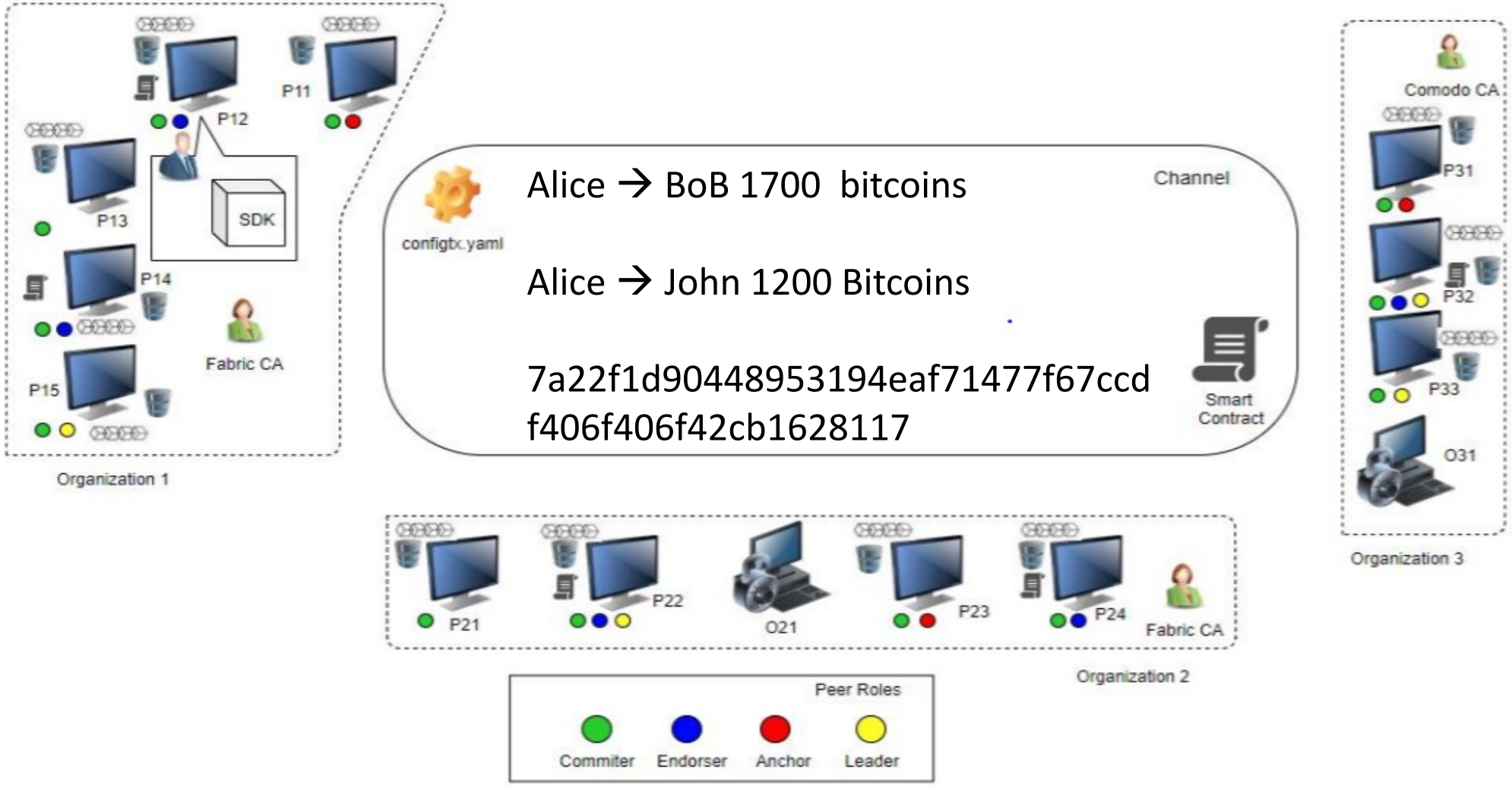
- A nonce (random number) is appended to the hash.
- The string is hashed.
- The final hash + transactions, are then submitted across the network.

- compared to the difficulty level and seen whether it's actually less than that or not.

Alice → BoB 1700 bitcoins

Alice → John 1200 Bitcoins

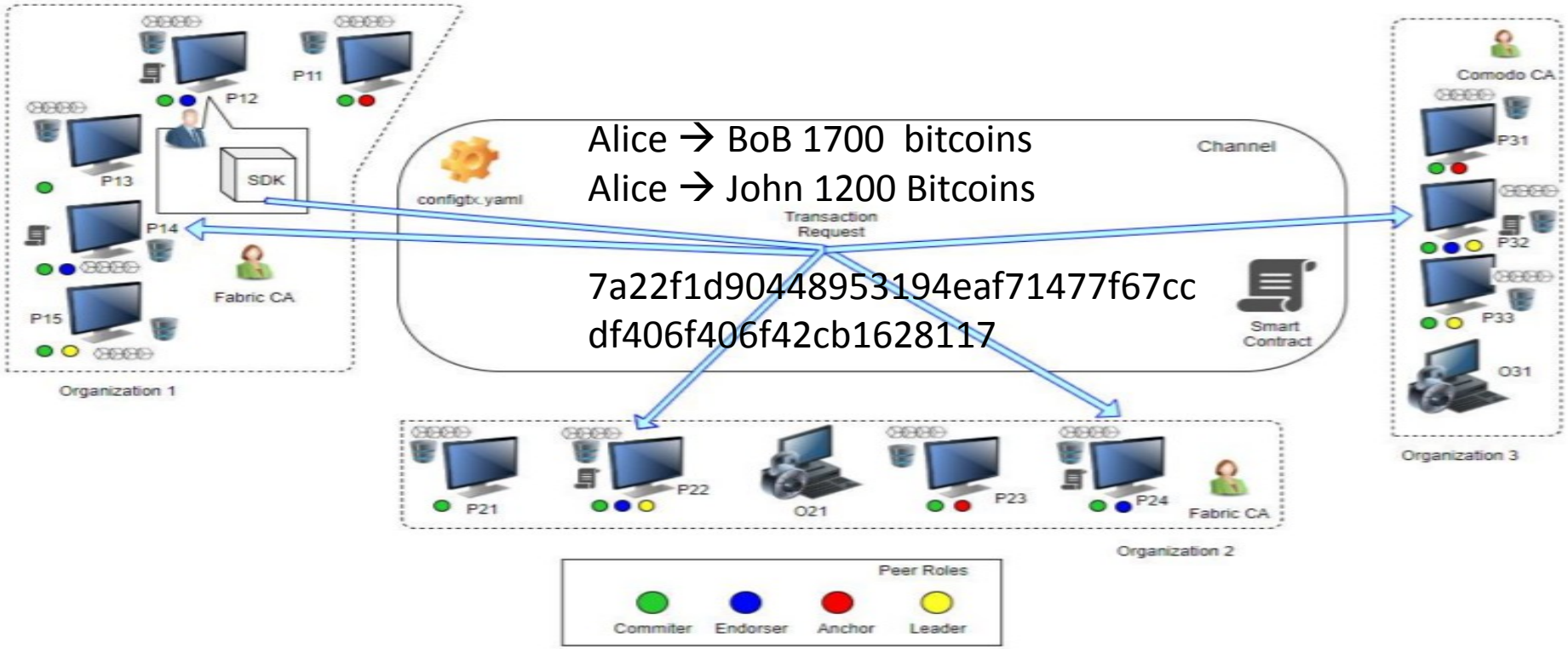
7a22f1d90448953194eaf71477f67ccdf406f406f42cb1628117


P11
P12
P13
SDK
P14
Fabric CA
P15
Organization 1
configtx.yaml
Alice → BoB 1700 bitcoins
Alice → John 1200 Bitcoins
7a22f1d90448953194eaf71477f67ccd
f406f406f42cb1628117
Channel
Smart
Contract
Comodo CA
P31
P32
P33
O31
Organization 3
P21
P22
O21
P23
P24
Fabric CA
Organization 2
Peer Roles
Commiter
Endorser
Anchor
Leader

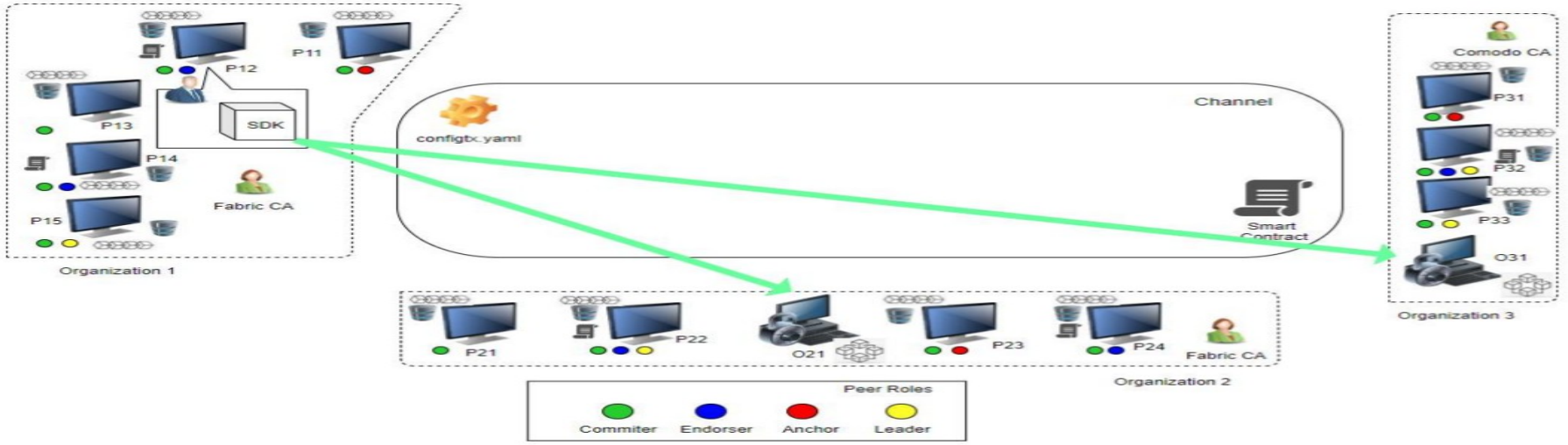
Endorsing peers verify signature & execute the transaction

1) The miners trying to find the write nonce

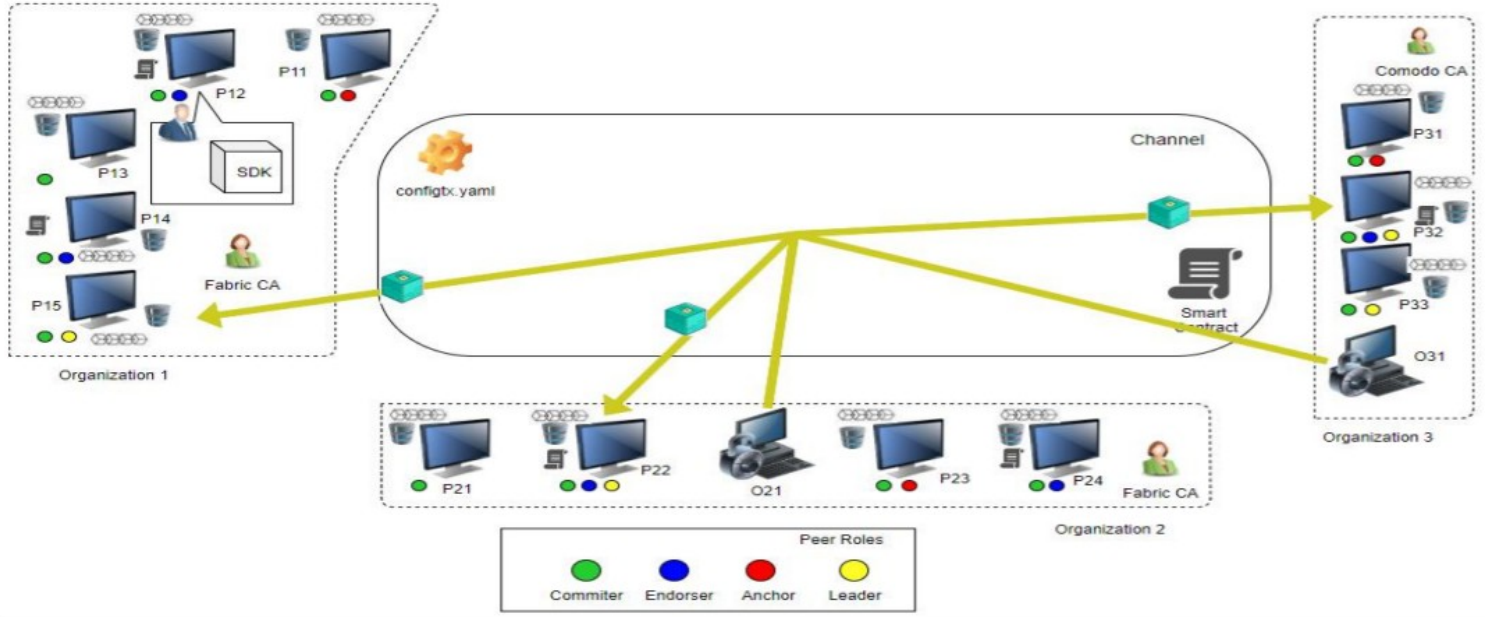
2) Mining is like a game, you solve the puzzle and you get rewards. Setting difficulty makes that puzzle much harder to solve and hence more time-consuming.



•If not, then the nonce is changed and the process repeats again.
•The miners responsible for this are rewarded with bitcoins.
• Each validated block contains a block hash that represents the work done by the miner.
•The nodes will verify the validity of the outcome and the miners node is rewarded with the block reward



Disseminate the block to leader peers



So to recap:

- Its is impossible to add new block into the main chain without first finding a valid nonce.
- The nodes will verify the validity of the outcome and the miners node is rewarded with the block reward.
- Miners need to try and guess a pseudo random number (nonce).
- The nonce, (the random number) is calculated independently , billions of times per second by each mining hardware system.
- These mining computers calculate billions of nonces per second. There is no manual about this, there are completely automated operation.

Hashing and data structures

A data structure is a specialized way of storing data. There are two data structure properties that are critical if you want to understand how a blockchain works. They are:
1.Pointers.
2.Linked Lists.

Pointers

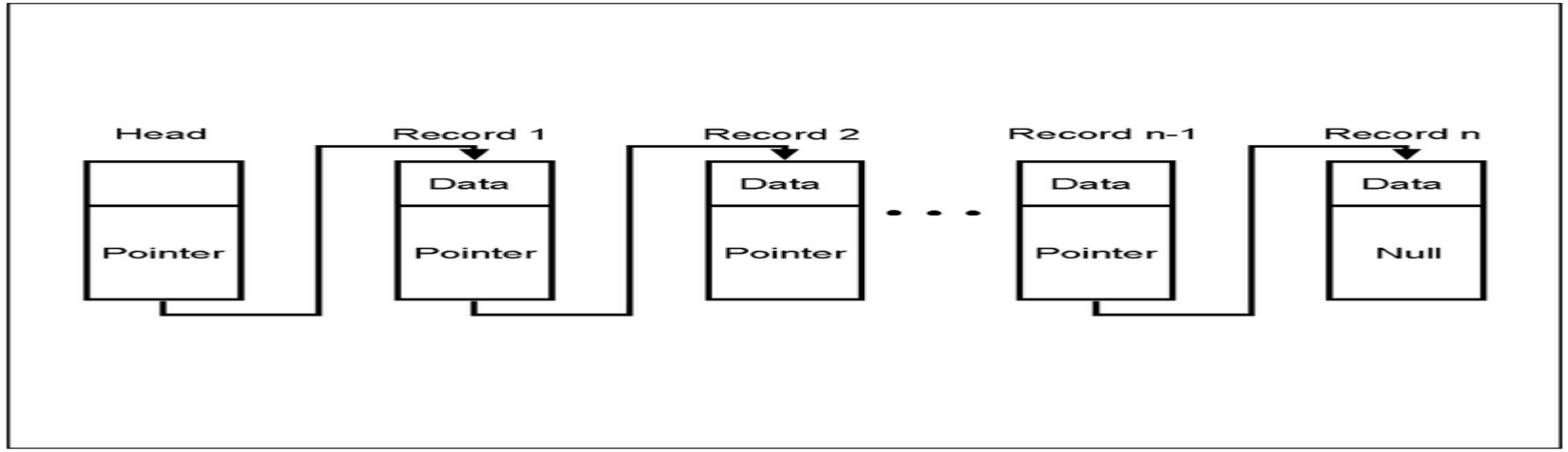
Pointers are variables in programming which stores the address of another variable. Usually normal variables in any programming language store data.
Eg. int a = 10, means that there is a variable “a” which stores integer values. In this case, it is storing an integer value which is 10. This is a normal variable.
Pointers, however, instead of storing values will store addresses of other variables. Which is why they are called pointers, because they are literally pointing towards the location of other variables.

Linked Lists

A linked list is one of the most important items in data structures. This is what a linked list looks like:

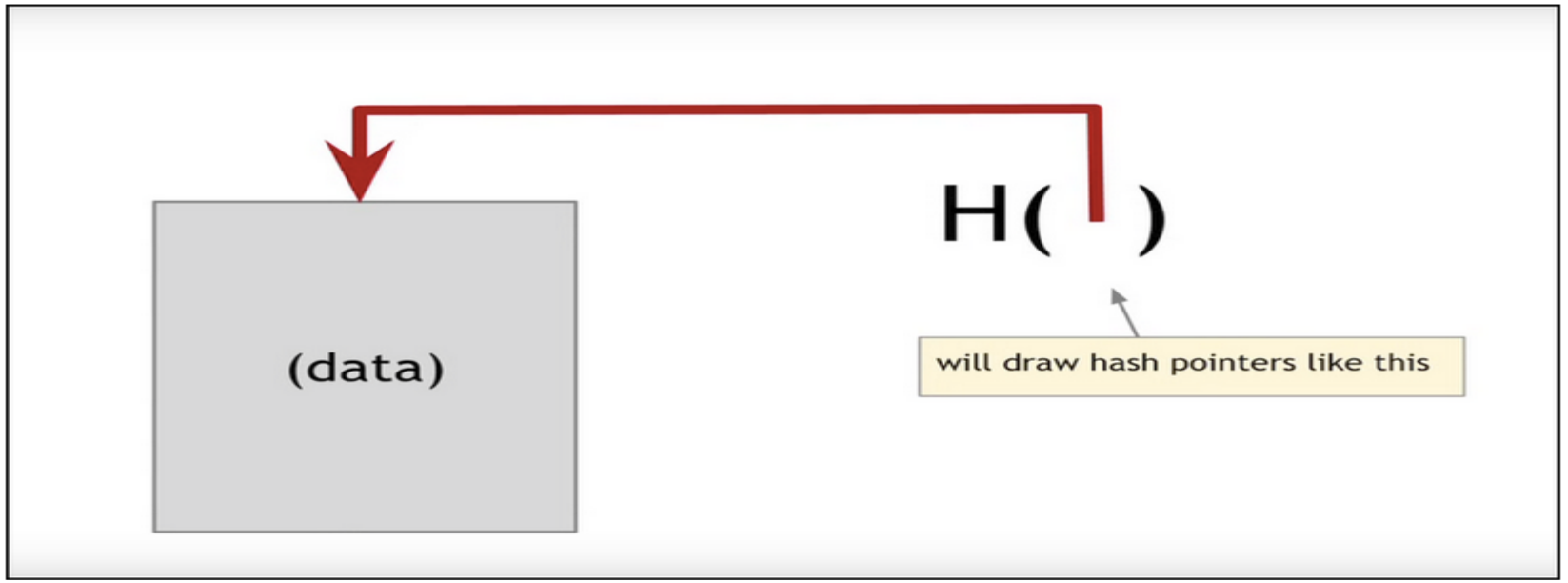
Linked Lists

A linked list is one of the most important items in data structures. This is what a linked list looks like:

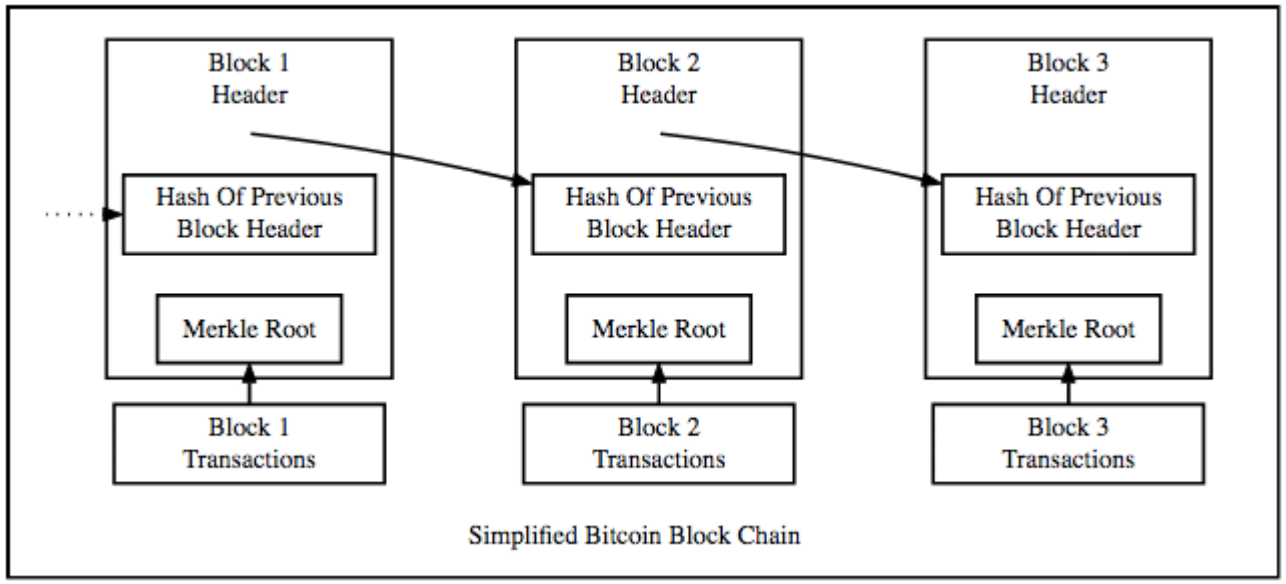


It is a sequence of blocks, each containing data that is linked to the next block via a pointer. The pointer variable, in this case, contains the address of the next node in it and hence the connection is made. The last node, as you can see, has a null pointer which means that it has no value.

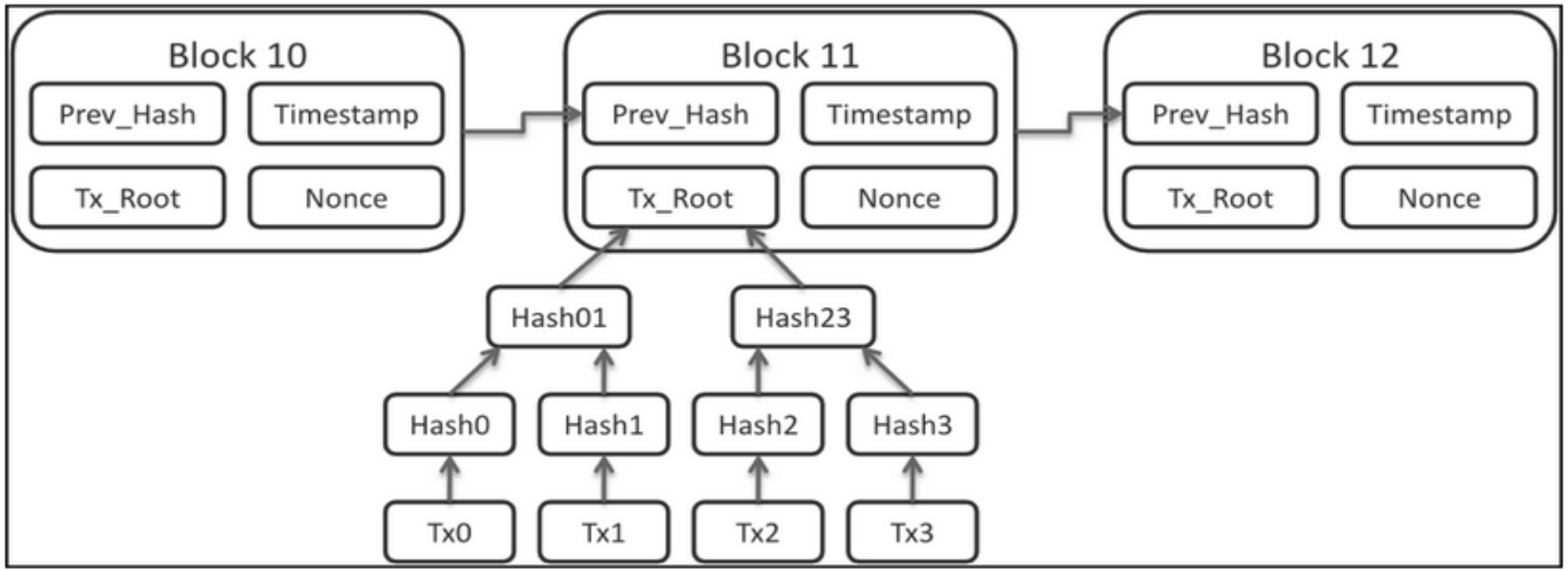
One important thing to note here, the pointer inside each block contains the address of the next block. That is how the pointing is achieved. Now you might be asking what does that means for the first block in the list? Where does the pointer of the first block stay?
The first block is called the “genesis block” and its pointer lies out in the system itself. It sort of looks like this:



If you are wondering what the “hash pointer” means, we will get there in a bit. As you may have guessed by now, this is what the structure of the blockchain is based on. A blockchain is basically a linked list. Let’s see what the blockchain structure looks like:



The blockchain is a linked list that contains data and a hash pointer that points to its previous block, hence creating the chain.



A block header contains:

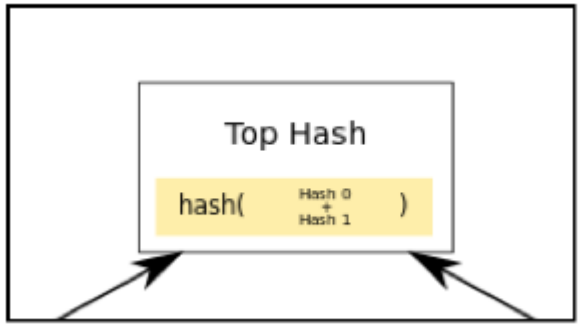
- Version: The block version number.
- Time: the current timestamp.
- Hash of the previous block.
- Nonce (more on this later).
- Hash of the Merkle Root.
- The current difficulty target. (More on this later).

let’s focus on the Hash of the Merkle Root. But before that, we need to understand what a Merkle Tree is.

What is a Merkle Tree?

The above diagram shows what a Merkle tree looks like. In a Merkle tree, each non-leaf node is the hash of the values of their child nodes.
Leaf Node: The leaf nodes are the nodes in the lowest tier of the tree. So wrt the diagram above, the leaf nodes will be L1, L2, L3 and L4.

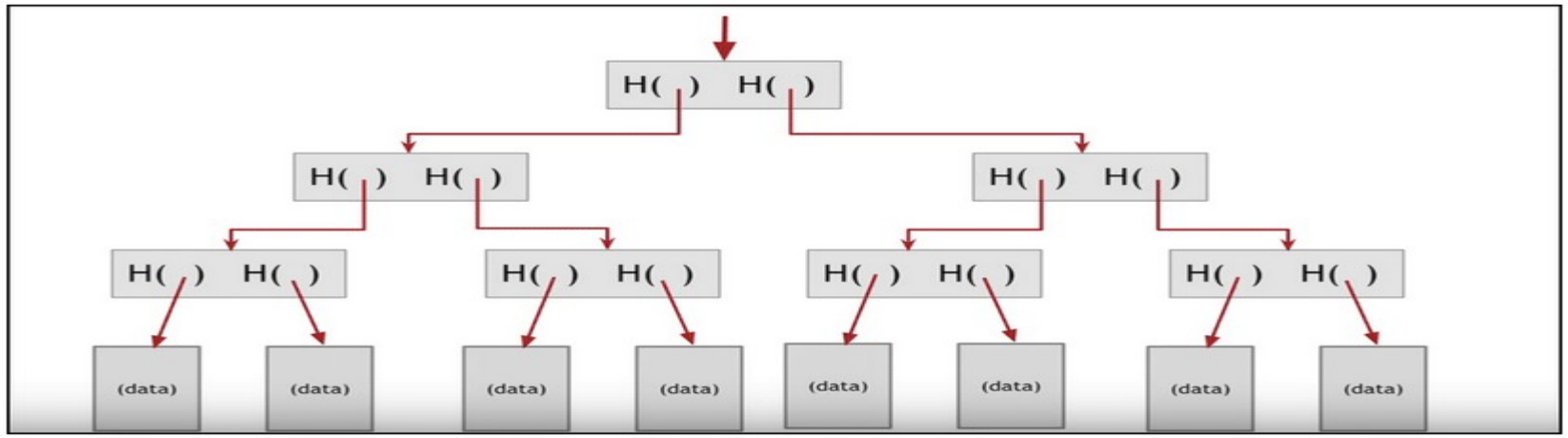
Child Nodes: For a node, the nodes below its tier which are feeding into it are its child nodes. Wrt the diagram, the nodes labeled “Hash 0-0” and “Hash 0-1” are the child nodes of the node labeled “Hash 0”.
Root Node: The single node on the highest tier labeled “Top Hash” is the root node.



So what does a Merkle Tree have to do with blockchains?

Each block contains thousands and thousands of transactions. It will be very time inefficient to store all the data inside each block as a series. Doing so will make finding any particular transaction extremely cumbersome and time-consuming. If you use a Merkle tree, however, you will greatly cut down the time required to find out whether a particular transaction belongs in that block or not.

Let's see this in an example. Consider the following Merkle tree:



Now suppose I want to find out whether this particular data belongs in the block or not: