Blockchain **Fundamentals**

- Myths.
- Decentralization.
- Distributed Consensus.

Blockchain is the buzz-word now a days. There is a lot of hype about different blockchain applications, including the cryptocurrencies. Both the industries as well as the researchers are exploring the huge potential of this new domain. Blockchain is predicted to make a big revolution in the coming decades.

The Myths about Blockchain:

#01 Myth: Blockchain means Bitcoin (!!!).

Many people wrongly conflate the two. A general perception is that: Government is ‘banning’ bitcoin, so what is the use of studying Blockchain???
People generally mess up with the two terms : Blockchain and Bitcoin.

- Blockchain is the technology that is used to design Bitcoin or any other cryptocurrencies (Etherium, Dogecoin, Solana, Matic etc.)
- Bitcoin or the other alt-coins are just only a use-case of the Blockchain technology.
- As an analogy, cryptocurrencies are just like any electrical equipment being driven by the ‘electricity’ Blockchain.
- We want to learn the technology and its potent applications for the betterment of mankind – we don’t want to go into the legal issues of banning or unbanning.

#02 Myth: Anything and everything cannot be solved using a Blockchain

Blockchain technology can solve some *most complicated problems* the world is dealing with right now. However, it is not the ideal solution for all problems.

- *"Want to prevent fraud and corruption?? – Use Blockchain"* - Unfortunately this is wrong. There can be even better technology to solve the problem efficiently and economically.
- Though this technology has got contribution from various domains like the computer scientists, the cryptographers, the economists, Blockchain also has got its own limitations. So there has to be a tradeoff.

Some real-life problems where Blockchain can be beneficial:

1. Supply chain / Logistics: Real-time tracking of the company products.
2. Accounting: Financial records & transactions require accuracy across every set of books.
3. Insurance: Verify and protect identity and records of the beneficiaries.
4. Social Media: Users' personal data and information are the product for the platform.
5. Healthcare: Accurate records for every patients is vital in ensuring appropriate treatment.
6. Public Service: Manually handling and managing records in public service offices is often slow, costly and prone to risk of error.
7. Data Security: Storing data in blockchain will bring vast improvement to data security. Blockchain can enhance the safety and speed of cloud storage.

#03 Myth: Blockchain is NOT a Distributed Database.

Both blockchains and distributed databases have a similar goal of maintaining a consistent copy of a particular dataset across a number of nodes.

- Database is a data structure of organized collection of data which is able to store any new data or access any existing data.
- A distributed database is a type of database management system that stores data across multiple computers or sites that are connected by a network.

Misconception: Potentially a distributed database can be replaced with a blockchain.

- Blockchain is not at all a distributed database. It is just a kind of replicated data structure.
- Both database technology and blockchain technology have their own applications. For some applications database or distributed database might work good, while for some applications blockchain might be useful.
- We have to be careful about, what we can put on a blockchain? What is the overhead to put that particular thing on the bloackchain? Is blockchain the beneficial solution for ANY data? Example: Storing a large story book on a blockchain is not a good use-case for a blockchain.

Blockchain is not equivalent to God.

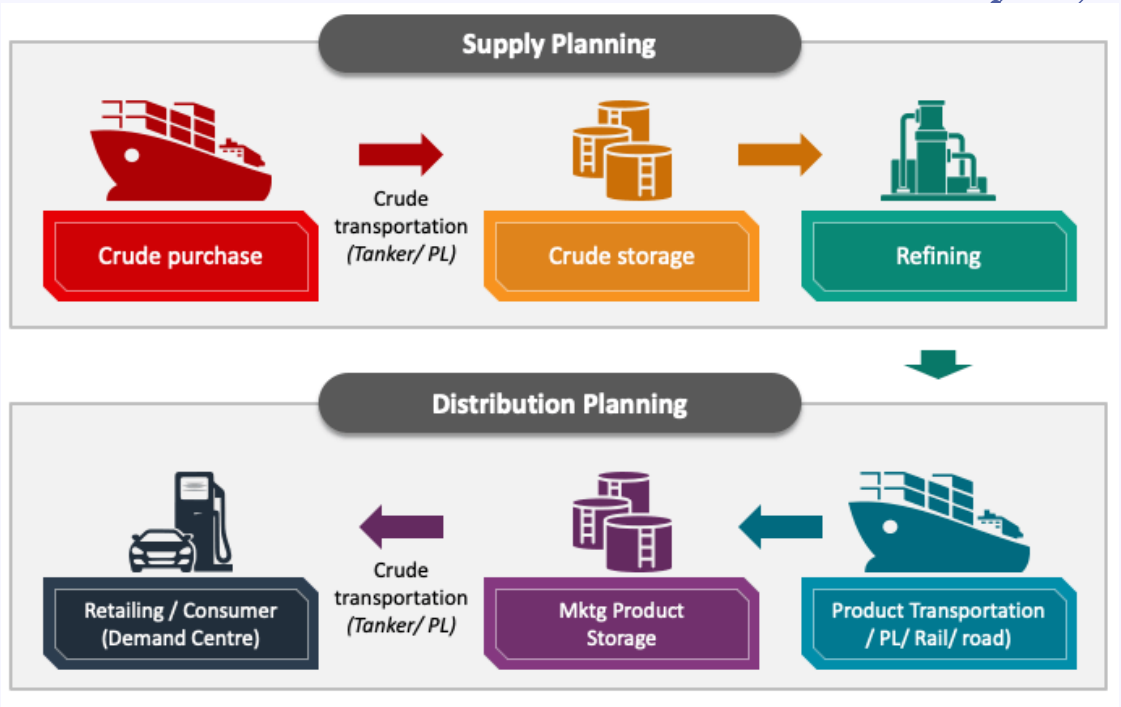
Decentralization

Why do we need decentralization?? When do we need to use this stellar technology??

A Blockchain Usecase: Supplychain Management in Petroleum Industry

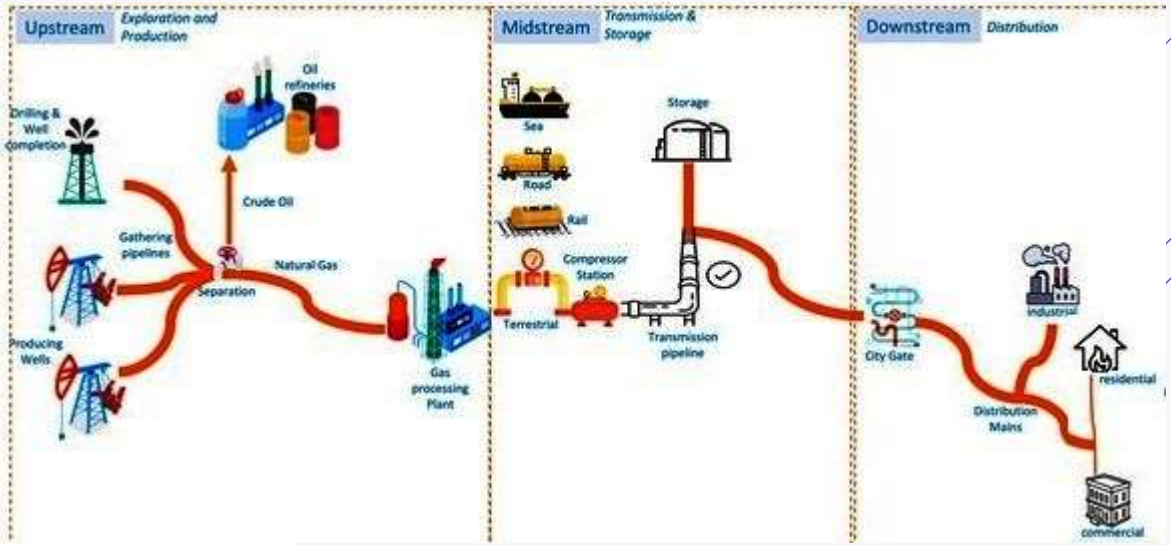
- It starts with the crude oil purchase and subsequently transported to the crude storage.
- From the storage it is send to the refineries for refining.
- After refining we get various types of products like petrol, diesel and other byproducts.
- These products are then distributed to different retailers.
- This is a simplified view of the end-to-end supplychain for the petroleum industry.

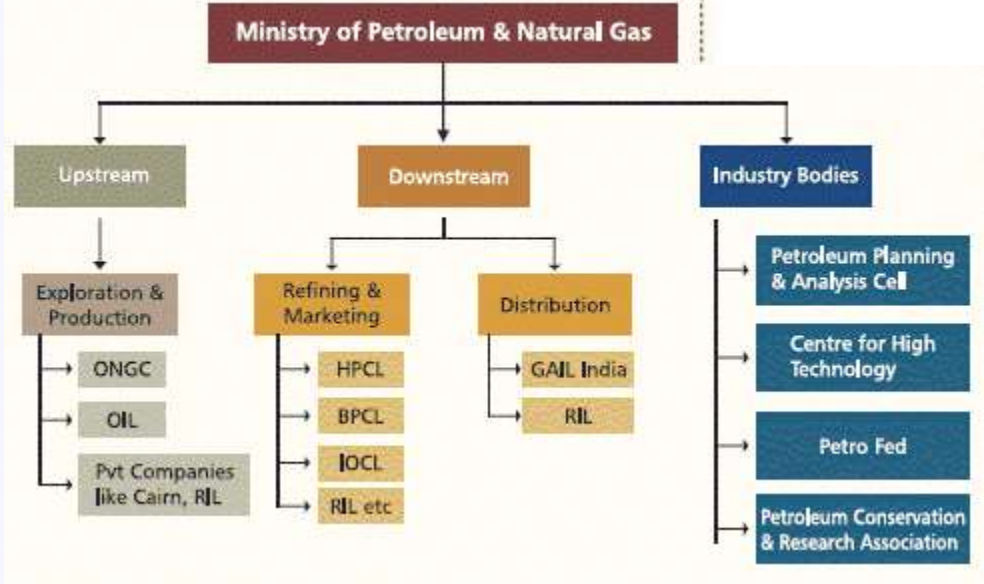
Who are the players in this Supplychain Management ??



Upstream Players:
Oil exploration and extraction from oil wells.
ONGC, Oil India Ltd. etc.

Midstream Players:
Transportation of oil & gas from refineries to distribution centers.





Downstream Players:
Refining, processing and marketing of products and byproducts of crude oil. HPCL, IOCL, GAIL, RIL etc.

Multiple industry bodies are also associated with the petroleum supplychain.
So, it is a complex system with so many players.

Requirements for a successful Supplychain Management

- Minimization of material procurement.
- Maximization of manufacturing capacity and sales.
- Meet demand numbers.
- Respond quickly to market opportunity by purchasing the production shortfall from other players i.e. cross-business among themselves.
- Objective of each production unit would be to maximize the throughput & its margin.
- Procurement would purchase the feedstock with the best yields at lowest cost.

To have a proper supplychain architecture: Strong Co-ordination is needed among the Players.

Question is : Who is going to establish this co-ordination?
Each organization is sharing a common product. Each of them wishes to maximize their profit margin. They are competitors to each other in this demand-supply.

Answer: An decentralized architecture, where they can co-ordinate, share information with each other in a successful way, for better management of this end-to-end supplychain.
Here everyone have their own governing structure, own policy of marketing/ business, but at the end of the day, they are working on a common product. So if they collaborate with each other, not only everyone is benefitted, the society as a whole gets benefitted. This is the decentralized architecture of individual players, without any central authority controlling them.

How to obtain Real-Time Information from the Stakeholders?

- **A web portal:** Containing infos about the current price, the quantity in reserve with each players, quantity of petrol and diesel produced etc. Challenges:
  - Who is going to manage the portal as there is no central authority??
  - What is the guarantee that the information submitted is correct???
  - Who is going to validate the portal information???
  - What if someone denies the information later on???

We need a decentralized solution – No one trust each other, but they should cooperate.

To solve this problem – Blockchain is the answer.

Trustless Decentralization = Blockchain

Decentralization with a Blockchain

No stakeholder trust each other, but they should cooperate for their benefit and for the benefit of the society as a whole.

For co-ordination something is needed where all the stakeholders can submit their infos and then, there should be a way to verify those infos.

Let us assume, there is a blackboard where all infos from the stakeholders of the end-to-end supplychain gets written.

- The infos are permanent on the board.
- Easy to track.
- Let the board has infinite space. So all historical data is available.
- Everyone can see all the logs & verify.
- Any change in info will be visible to all.
- The board is non-erasable. So no one can deny later.

Problem: Who will maintain the blackboard?
(no Central Authority)

What is the problem with Central Authority? Why Decentralization?

- If there is a central authority, in case of certain attacks all the infos might get lost. A perfect decentralized architecture will make the system more fail-safe.
- In case of central authority, it is easier to attack on the single point. If there is a decentralized architecture, multiple players are going to maintain the info. So it will be difficult (atleast 51% majority) for the attacker to manipulate.
- Thus, decentralization always improves security.

Possible Solution:

Let everyone maintain the same copy of the board individually and independently.

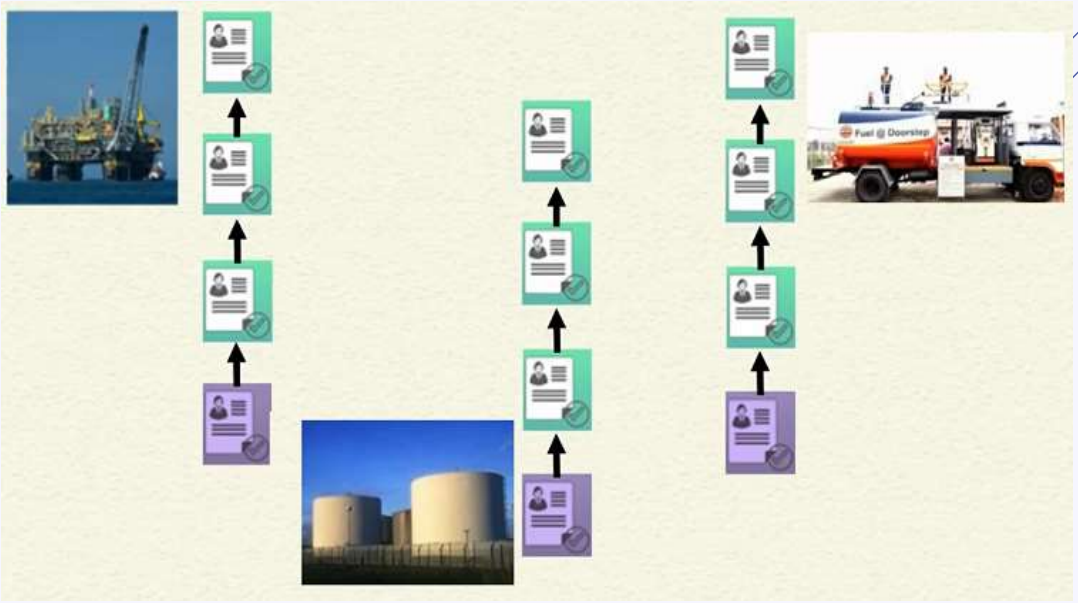
- There is no central database. Everyone has a copy of the same data.
- These data has to be consistent. Same copy at the same instance of time.

This is known as
Distributed Consensus Problem.

Let us replace these blackboards with a particular data-structure called Blockchain.

Blockchain: A decentralized, immutable, append-only, ever growing chain of data. Data once added, cannot be deleted or modified from this public ledger.

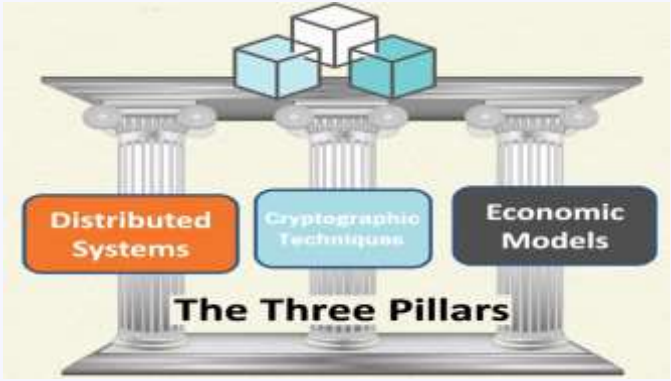
- Each player process the data of the blockchain locally for any info.
- New information is added to the chain in the form of new Blocks. The entire data-structure is a chain of blocks, hence blockchain.
- The information is transparent to everyone – so everyone can verify and validate at any point from their local copy.



Distributed Systems for Decentralization

A distributed system is one that uses networked computers to accomplish a common task. Many networked computers are used to distribute the job.

The idea of blockchain borrows concept from different domains to form up a technology in order to solve different practical problems.

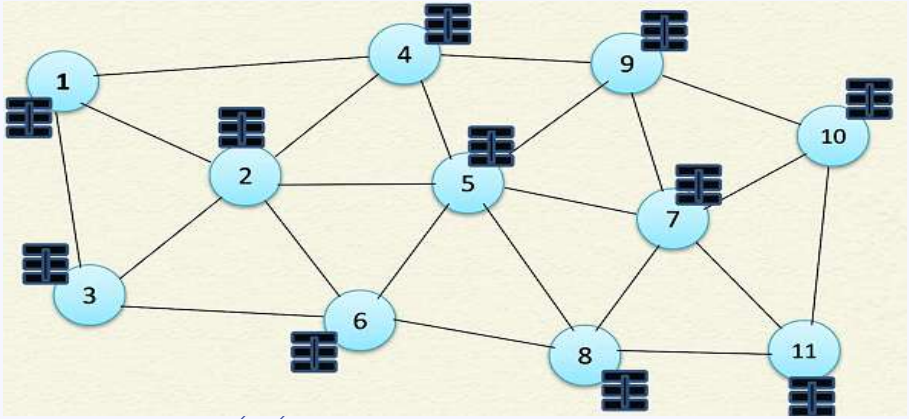


The 3 important pillars that are there behind the design of blockchain architecture are:

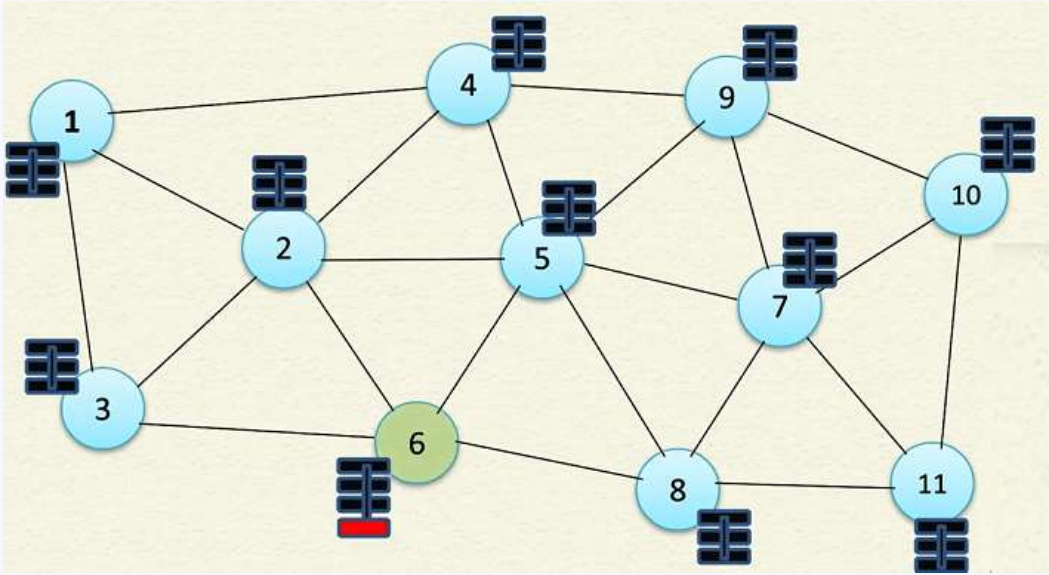
1) Distributed Systems
2) Cryptographic Techniques
3) Economic Models

The core problem:

- Data structure/ chain of blocks maintained in all the nodes.
- Every node maintains their local copy of blockchain.
- Let node 6 adds up a new block in the blockchain.



- All other nodes in the network need to agree to this new block.
- They need to add this block to their copy of the blockchain.



This is an old problem in distributed system, where someone is proposing something and all other peers need to collectively decide whether to agree on the proposal or not, is termed as: the Classical Distributed Consensus Problem.

Distributed Consensus Example:

1st Case: Let 4 friends wants an eat out collectively. They need to decide whether to go for pizza or burger. Someone becomes the leader, calls up the rest 3 to get their choices, decides where to go and again informing the choice to others.

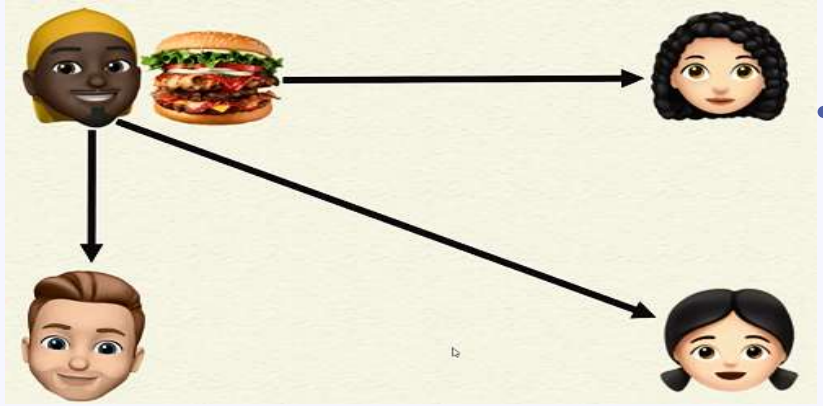
Here all the other friends are relying on the leader that he is unbiased and true. That is, there is no fraudulent behavior by the leader. But in blockchain no peer trust each other.

2nd Case: How can we reach a consensus in a trustless set-up??

Everyone has their own choice of pizza or burger. Everyone make a call to others informing his/her choice. Thus everyone gets information from others. Take a majority voting & decide.

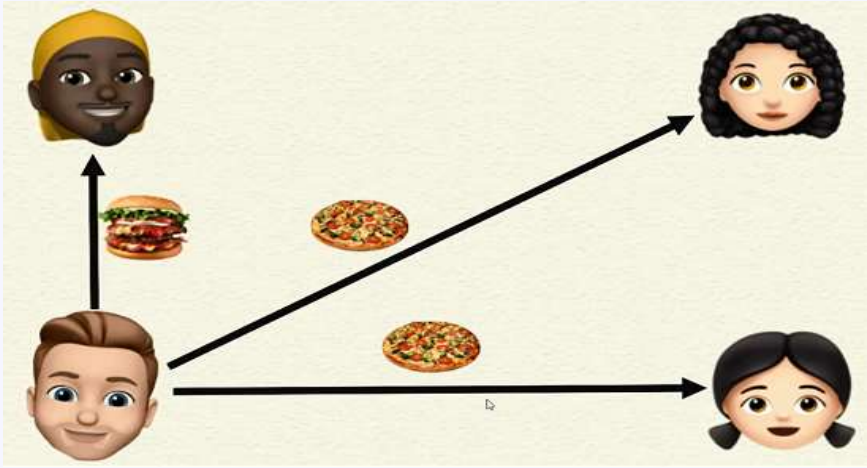
Thus everybody comes to a consensus in a distributed set-up. No leader is required. But……

- Less number of calls in a centralized solution compared to distributed set-up.
- One or more can show Byzantine behavior. Then what???



One friend, while calling 2 others, say ‘I want pizza’ but calling the 3rd friend says ‘I want a burger’. This is known as **Byzantine behavior**.

We have to design consensus in the distributed set-up, where no one trust each other, in the presence of this Byzantine behavior. How? will study..



Concept: Synchronous & Asynchronous Communication

The key difference between synchronous and asynchronous communication is synchronous communications are scheduled, real-time interactions by phone, video, or in-person. Asynchronous communication happens on your own time and doesn’t need scheduling.

While synchronous communication is instantaneous (phone call), there is an expected lag in asynchronous communication like sms, email etc.

Practical communication channels, particularly over the internet, are more biased towards the asynchronous mode of communication.

FLP Impossibility Theorem:

In 1985, Michael Fischer, Nancy Lynch and Michael Paterson (hence FLP) published a seminal paper, which became known as the **FLP Impossibility theorem**.

- *Consensus is impossible in a fully asynchronous system even with a single crash fault.*

Failures in distributed systems are known as faults. Broadly they are of two types: Crash fault and Byzantine fault. Crash faults can be software fault or hardware fault. The node stop responding in crash fault. Byzantine fault occurs when a Byzantine node communicate different info to different nodes. Crash faults are more easier to handle.

- *Cannot ensure Safety and Liveness **together**.*

Safety (a.k.a. Finality or Agreement): A block being committed should remain part of the final blockchain that is accepted by the distributed network as the true block.

Liveness (a.k.a. Guaranteed Termination): All nodes (that have not failed) eventually decide. The output will be produced within a finite amount of time. Thus there will be an eventual termination of the consensus algorithm.

FLP theorem says, it is impossible to support safety and liveness together. In practical cases, priority is given more to ‘safety’ over ‘liveness’ in-order to get consensus.

Consensus algorithms: Paxos, Raft, Byzantine Fault Tolerance, PBFT etc..

The Birthday Paradox:

• In a room of just 23 people there's a 50-50 chance of at least two people having the same birthday.

• In a room of 75 there's a 99.9% chance of at least two people matching.

• We randomly take one person out of the group of N people. This person obviously has 365^1 possible birthdates denoted as A (considering non-leap year). Let's pick another person, with birthdate B. There are 365^2 possible birthdates they can have. What is the probability the A is different from B?

$$P = \underbrace{\frac{365}{365}}_{A} \cdot \underbrace{\frac{364}{365}}_{B}$$

P = 0.9972.

Thus, the probability that these two people share a birthdate is 1 - P = 0.0027.

• Assume we have 3 people. There are 365^3 possible birthdays they can have. All three has different birthdate is given by $P = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365}$ P = 0.9912.

Thus, the probability of a shared birthdate is 1 - P = 0.0088.

• Lets calculate the probability of choosing randomly 23 people (365^{23} possible birthdays) so that no one of them shares a birthdate. $P = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{343}{365}$ P = 0.4927.

Similarly, to find the probability of a shared birthdate we need to calculate 1 – P = 0.5073.

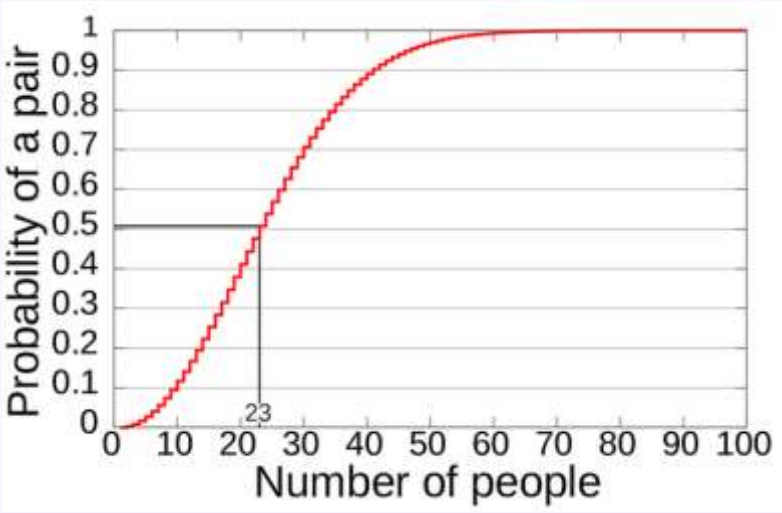
We can generalize this with the pigeonhole principle :

$$P = \frac{365 \cdot 364 \cdot \ldots \cdot (365-n+1)}{365^n} = \frac{365!}{365^n \cdot (365-n)!}$$

Another approach:

$$P = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{343}{365}$$ or

$$P = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{22}{365}\right)$$



Thus, with n=365, the general expression for least k persons with no birthdate collision is:

$$P = (1 - 1/n)\left(1 - \frac{2}{n}\right)\left(1 - \frac{3}{n}\right) \ldots\ldots (1 - (k-1)/n)$$

From Taylor series $e^x \approx 1 + x$ hence $e^{-\frac{1}{n}} \approx 1 - \frac{1}{n}$

Replacing we get $P = e^{(-1-2-3\ldots.-k+1)/n} = e^{-k(k-1)/2n}$

What is the no. of people(k) required to have at least a 50% chance of repetition?

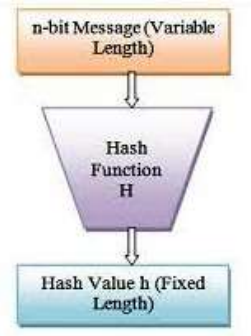
We solve $\frac{1}{2} = e^{-k(k-1)/2n}$. Taking log both side $-\ln(2) = -k(k-1)/2n \Rightarrow k(k-1) = 2nln(2)$

Taking approximation (k-1) ≈k, we get $k^2 = 2n*\ln(2)$. Or **$k \approx 1.1774\sqrt{n}$**.

For the shared birthday case; we see that with n=365, required minimum k becomes 22.49 or approximately 23 people.

Why Paradox? It is a ‘paradox’ because out brain finds it difficult to handle the compounding power of exponents.

Cryptographic Primitives

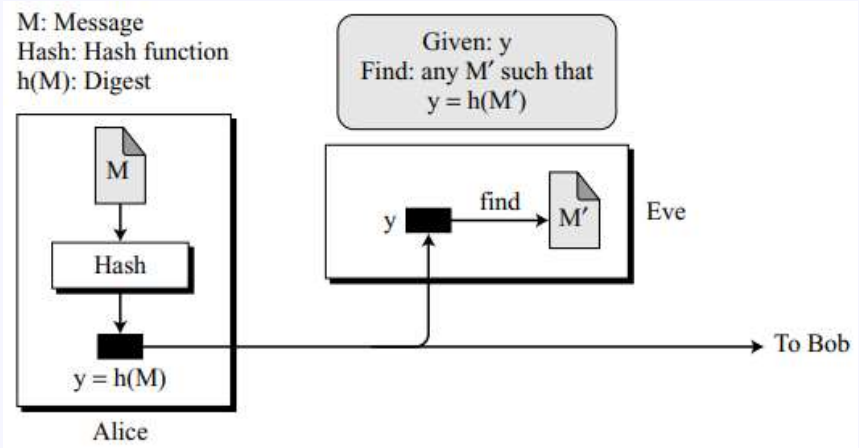


Cryptographic Hash Functions:

A hash function accepts a variable-size message M as input and produces a fixed-size message digest H(M) as output.

- The hash code is also referred to as a **message digest** or hash value.
- Change to any bit/bits in the message results in a change to the hash code.

A cryptographic hash function must satisfy three criteria: **preimage resistance**, **second preimage resistance**, and **collision resistance**.



●Preimage Resistance:

●A cryptographic hash function must be preimage resistant. Given a hash function h and y = h(M), it must be extremely difficult for Eve to find any message, M′, such that y = h(M′).

●If the hash function is not preimage resistant, Eve can intercept the digest h(M) and create a message M′. Eve can then send M′ to Bob pretending it is M.

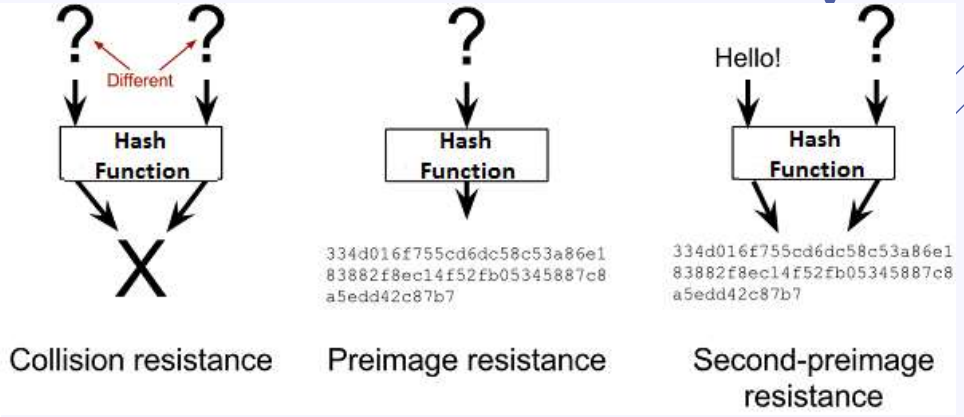
●It is the one-way property. It is easy to generate a code given a message, but virtually impossible to reverse i.e. generate a message given a code.

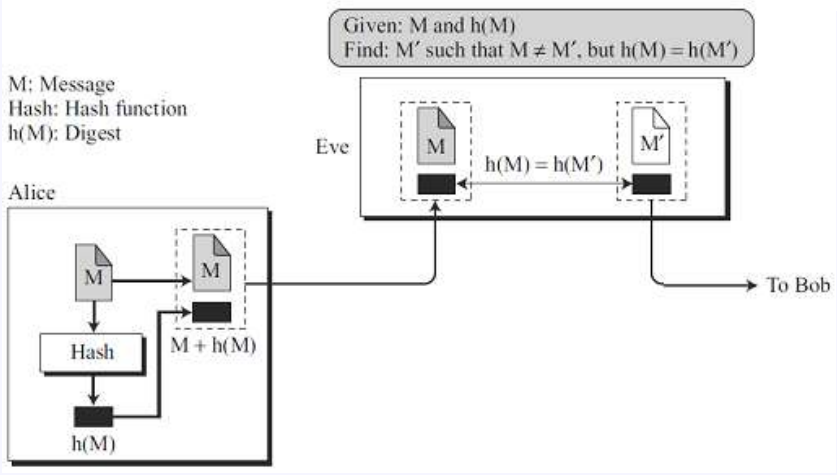
● Second Preimage Resistance:

It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage x ′ ≠ x such that h(x) = h(x′).

● It guarantees that an alternative message hashing to the same value as a given message cannot be found.

● It is known as **weak collision property**.





● Collision Resistance:

It is computationally infeasible to find any two distinct inputs (x, x′) which hash to the same output, i.e., such that h(x) = h(x′). This is **strong collision property**.

● Collision resistance implies second-preimage resistance of hash functions.

If any two messages produce the same message digest, it is called a **collision**. Message digest algorithms usually produce a message digest having a length of 128 bits or 160 bits or 256 bits. This means that the chances of any two message digests being the same are one in 2^{128} or 2^{160} or 2^{256}, respectively.

- Clearly, this seems possible only in theory, but extremely rare in practice.
- The birthday attack is a method to find collisions in a cryptographic hash function. Having the same birthday is the analogue of a “collision” in a hash function.
- Remember: If there are 23 people in a room, chances are that more than 50% of the people will share the same birthday.
- If a message digest uses 64-bit keys then after trying 2^{32} transactions, an attacker can expect that for two different messages, he may get the same message digests. In general, for a given message, if we can compute up to N different message digests then we can expect the first collision after the number of message digests computed exceeds the square root of N.

Commonly used Hash Functions:

There are many types of hash functions, each with its own strengths and weaknesses.

- **MD5 (Message Digest 5)**: A widely-used cryptographic hash function that produces a 128-bit hash value.
- **CRC (Cyclic Redundancy Check)**: A non-cryptographic hash function used primarily for error detection.
- **SHA (Secure Hash Algorithm):** SHA is a family of cryptographic hash functions designed by the National Security Agency (NSA) in the United States. The most widely used SHA algorithms are SHA-1, SHA-2, and SHA-3.

SHA-2 is a family of hash functions that includes SHA-224, SHA-256, SHA-384, and SHA-512. These functions produce hash values of 224, 256, 384, and 512 bits, respectively.

Parameters for the Various Versions of SHA

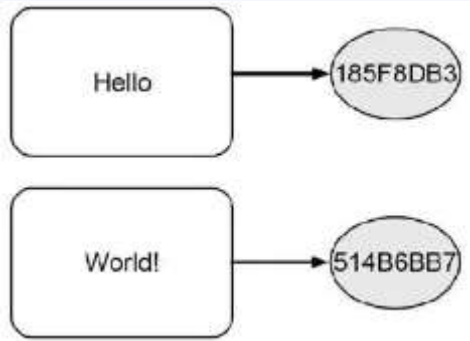
Parameter	*SHA-1*	*SHA-256*	*SHA-384*	*SHA-512*
Message digest size (in bits)	160	256	384	512
Message size (in bits)	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size (in bits)	512	512	1024	1024
Word size (in bits)	32	32	64	64
Steps in algorithm	80	64	80	80

Patterns of Hashing Data:

- **Independent hashing.**
- **Repeated hashing.**
- **Combined hashing.**
- **Sequential hashing.**
- **Hierarchical hashing.**

Independent hashing.

The Hash function transforms each input data separately

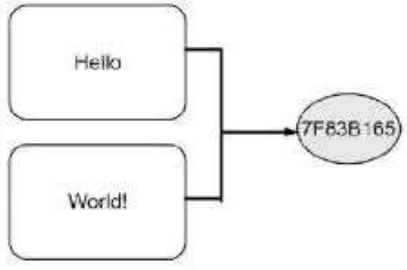


Repeated hashing.

The Hash functions transforms input data into a hash value and again this hash value is given as input and produces another output hash value.

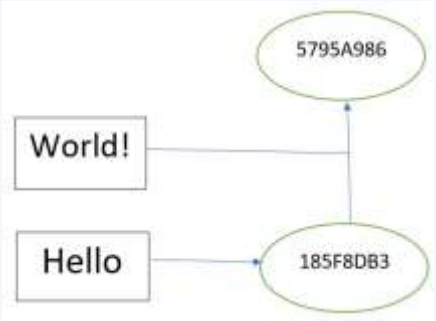
Combined hashing.

It enables to produce singular hash value for more than one chunk of data.



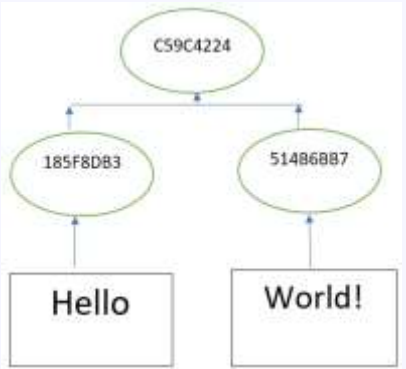
Sequential hashing.

Sequential hashing creates an update of a hash value as soon as new data appears using combined and repeated hashing simultaneously.



Hierarchical hashing

Hierarchical hashing uses combined hashing to create pairs of hash values which enable the creation of hierarchy.



SHA-256:

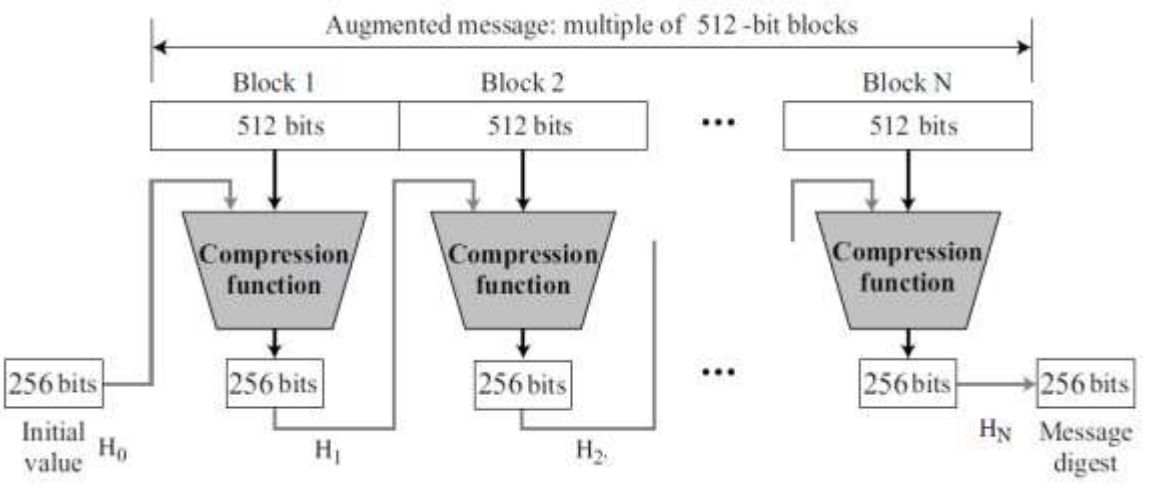
SHA-256 is a popular hashing algorithm used in Bitcoin encryption in 2009. Since then, SHA-256 has been adopted by a number of different blockchain projects, including several coins created from forks of the original Bitcoin source code.

- Hash functions are used to connect the ‘blocks’ in a ‘chain’ in a tamper-proof way.
- SHA-256 like the others in the SHA family of algorithms, is based on the Merkle-Damgard scheme.

SHA-256 Algorithm (Merkle-Damgard Scheme):

The Merkle-Damgard scheme is an iterated hash compression function that is collision resistant. The scheme uses the following steps:

1) The message length and padding are appended to the original message to create an augmented message (multiple of 512-bit blocks), that can be evenly divided into blocks of n = 512 bits.
2) The message is then considered as N blocks, each of 512 bits. We call the digest created at N iterations H_1, H_2,…, H_N.



3) Before starting the iteration, the digest H_0 is set to a fixed value, normally called IV (initial value or initial vector).
4) The compression function at each iteration operates on H_{i-1} and message M_i to create a new H_i. In other words, we have $H_i = f(H_{i-1}, M_i)$, where f is the compression function.
5) H_N is the cryptographic hash function of the original message, that is, h(M).

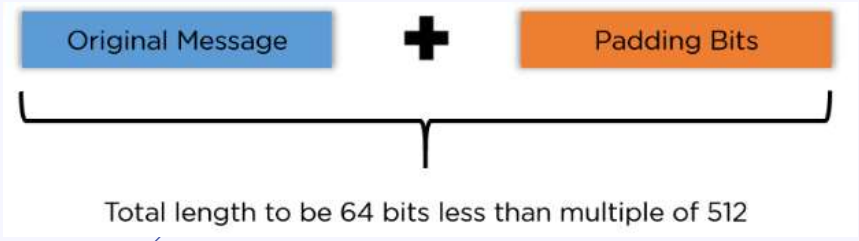
Steps in SHA-256 Algorithm:

Step #1: Appending bits

The first step involves preprocessing the input message to make it compatible with the hash function. It can be divided into two substeps:

- **Padding Bits:**

The total length of the message must be a multiple of 512. In this step, we append bits to the end of the message such that the final length of the message must be 64 bits less than a multiple of 512. The formula depicts this step: $m + p = (n * 512) - 64$.
where m = length of the message, p = length of the padding, and n = a constant.
The first bit that we append is 1 followed by all 0 bits.

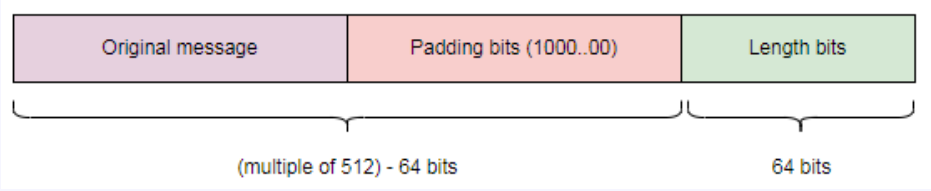


• **Padding Length**

After padding bits are added, the next step is to calculate the original length of the message, and add it to the end of the message, after padding. How is this done?

• The length of the message is calculated, excluding the padding bits (i.e. it is the length before the padding bits were added). For instance, if the original message consisted of 1000 bits, and we added a padding of 472 bits to make the length of the message 64 bits less than 1536 (a multiple of 512), the length is considered 1000, and not 1472, for the purpose of this step.

• This length of the original message is now expressed as a 64-bit value, and these 64 bits are appended to the end of the original message + padding. If the length of the message exceeds 2^{64} bits (i.e. 64 bits are not enough to represent the length, which is possible in the case of a really long message), we use only the low-order 64 bits of the length. That is, in effect, we calculate the length mod 2^{64} in that case.



Step #2: Buffer initialization

Before we begin carrying out computations on the message, we need to initialize some buffer values. The default eight buffer values are 8 32-bit words derived from square roots of first 8 prime numbers.

A = 0x6a09e667 **C** = 0x3c6ef372 **E** = 0x510e527f **G** = 0x1f83d9ab

B = 0xbb67ae85 **D** = 0xa54ff53a **F** = 0x9b05688c **H** = 0x5be0cd19

Moreover, we have to initialize an array containing 64 constants, denoted by *k*. These 64 32-bit words are cube roots of first 64 prime numbers.

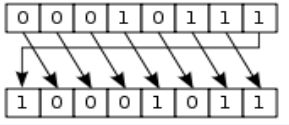
```
k[0..63] :=
   0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

Some Operation Concepts:

i) **Shift Right Operation**: [$SHR^n(x) = x >> n$]. Right shift of each bit by n. Initial n bits are replaced by 0s. Example: $SHR^1(10100111) = 01010011$. 

, $SHR^3(1011) = 0001$.

ii) **Rotate Right Operation:** [$ROTR^n(x)$]. A bitwise rotate right is a right shift where the overflow is added to the left side instead of dropping. $ROTR^1(00010111) = 10001011$



iii) **Addition Modulo 2^{32}**: [$Z = (X+Y) \bmod 2^{32}$]. This restricts the result of addition to 32 bits.

Some Logical Functions:

i) $\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$

```
x:        00000000000000000111111111111111
          --------------------------------
ROTR 7:   11111110000000000000000001111111
ROTR 18:  00001111111111111100000000000000 XOR
 SHR 3:   00000000000000000001111111111111 XOR
          --------------------------------
σ0(x):    11110001111111111100011110000000
```

ii) $\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

```
x:        00000000000000000011111111111111
          --------------------------------
ROTR 17:  00011111111111111000000000000000
ROTR 19:  00000111111111111110000000000000 XOR
 SHR 10:  00000000000000000000000000001111 XOR
          --------------------------------
σ1(x):    00011000000000000110000000001111
```

iii) $\sum_0^{\{256\}}(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$

```
x:        00000000000000000011111111111111
          --------------------------------
ROTR 2:   11000000000000000000111111111111
ROTR 13:  11111111111110000000000000000001 XOR
ROTR 22:  00000000111111111111110000000000 XOR
          --------------------------------
Σ0(x):    00111111000001111111001111111110
```

iv) $\sum_1^{\{256\}}(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$

```
x:        00000000000000000011111111111111
          --------------------------------
ROTR 6:   11111100000000000000000011111111
ROTR 11:  11111111111000000000000000000111 XOR
ROTR 25:  00000000000111111111111110000000 XOR
          --------------------------------
Σ1(x):    00000011111111111111111101111000
```

v) Conditional Function:

$$\mathrm{Ch}(x, y, z) = (x \wedge y) \oplus (\overline{x} \wedge z)$$

∧ is **logical AND operation.**

vi) Majority Function: It takes three corresponding bits in three buffers (x, y, and z) and calculates

$$\mathrm{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

Step #3: Parsing the Padded Message:

Each 512 bit message block is subdivided into 16 32-bit words (512 /32 = 16), collectively called the "Message Schedule" denoted as $M_0^{(i)}, M_1^{(i)} \ldots, M_{15}^{(i)}$

- We have to expand message schedule such that there are a total 64 32-bit words.

Step #4: Expand Message Schedule:

- Create 48 32-bit words using the formulae:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

$M_t^{(i)}$ = 32-bit word from the 512-bit message block.

W_t = 32-bit word created by the formula.

Remember:

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

+ is Addition Modulo 2^{32}.

Step #5: Hash Computation

5 a) Initialize working variables:

In this step we use eight working variables a,b,c ... h. Those variables will be used to update the hash value in one step of the iteration.

- In the first iteration, a,b,c ... h are initialized with constants.
- In each subsequent iteration, a,b ... h are initialized with the hash values H_0 ... H_7 from the previous iteration.

5 b) Compute the working variables:

- In this step, we perform 64 iterations t = 0,1 ... 63.
- In each iteration, we shift the values of the working variables to the right, so that h = g, g = f, f = e and so on.
- We compute values of working variables a and e as follows:

$$e = d + T_1$$
$$a = T_1 + T_2.$$

For $t = 0$ to 63:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

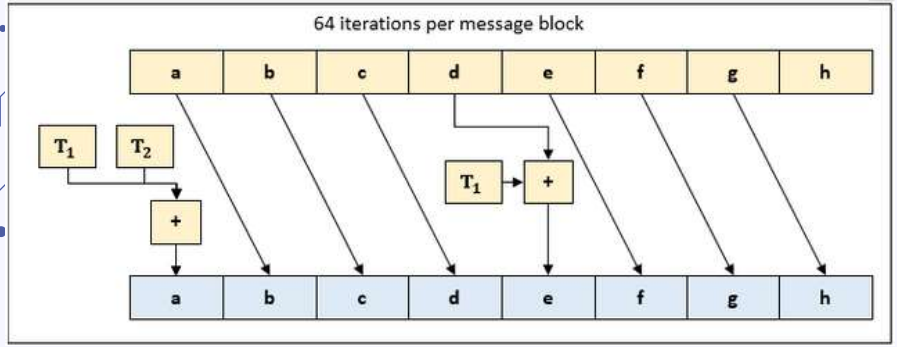
$$b = a$$

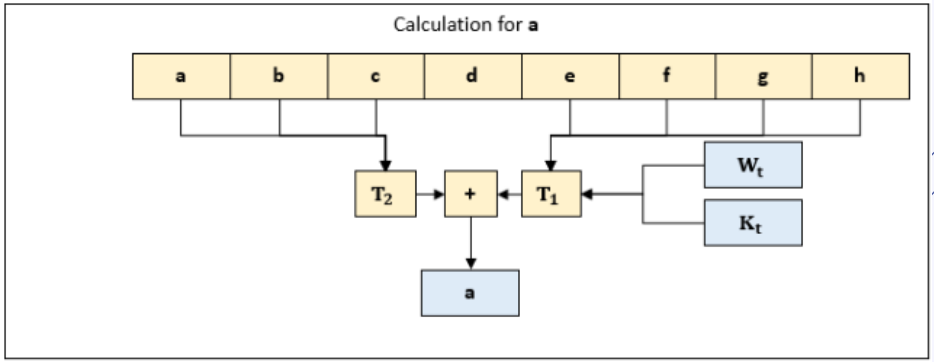
$$a = T_1 + T_2$$

}

where:

- T_1 = temporary word 1
- T_2 = temporary word 2
- $a...h$ = working variables
- $\sum_0^{\{256\}}(a)$ = uppercase sigma 0 func. on a
- $\sum_1^{\{256\}}(e)$ = uppercase sigma 1 func. on e
- $Ch(e,f,g)$ = choice func. on e, f, g
- $K_t^{\{256\}}$ = constants
- W_t = words in message schedule
- $Maj(a,b,c)$ = majority func. on a, b, c





T_1 and T_2 are temporary values computed. T_1 and T_2 serve the purpose of mixing the message into the hash value.

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$

For 64 words and 64 constants repeat the iterations.

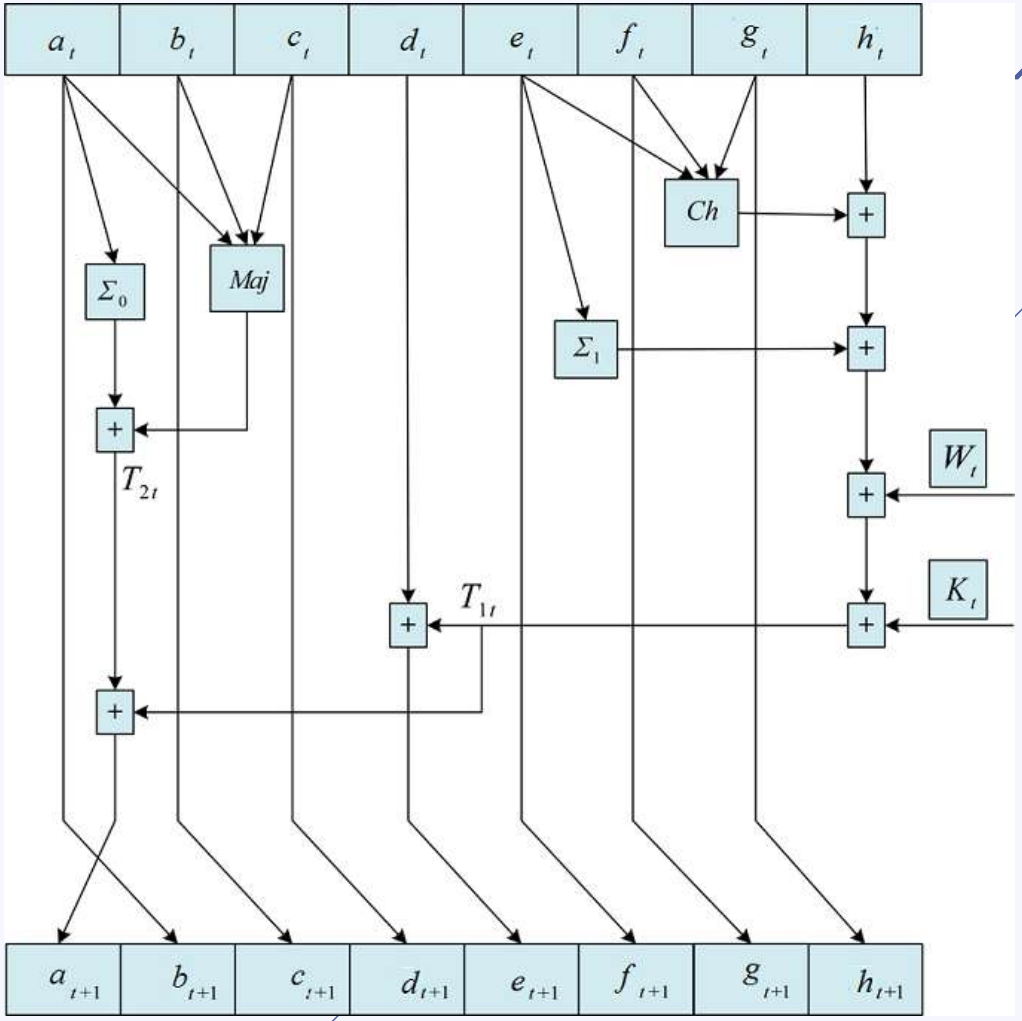
5 c) Update hash values:

After computing the working variables a,b ... h, the new intermediate hash values are calculated. This is done by a simple addition to of a,b ... h to the hash values of the previous iteration.

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

5 d) Finalize the hash value:

After running one iteration for each message block, the hash values H_0 ... H_7 are converted into Hex and concatenated into one 256-bit hash value.



Security Properties Of Hash Functions

The below properties aim to give an idea of what's required for a secure cryptographic hash function.

- **Hiding property** (or Pre-image resistance): Just knowing H(x), you cannot deduct x. However, this only holds up if message x is not chosen from a "likely" set of values. "Likely" here means that x cannot be realistically guessed by an attacker.
- **Collision Free:** A collision occurs if for two different messages the same hash value is calculated. "Collision free" means that it is infeasible to find two messages x and y, with x != y, so that H(x) = H(y). Meaning it should be near impossible that two different messages result in the same hash value.
- **Puzzle friendliness,** especially relevant for cryptocurrencies:

•Suppose we know two values x and y. x is part of the hash function input message, y is the hash function output.

•A hash function is puzzle friendly if it is infeasible to find a third value k, so that H(k | x) = y. k | x means just concatenation of k and x (e.g. if k = 1100 and x = 0011 then k | x = 1100 0011).

•We assume k to be a random value from a very spread-out set of values.

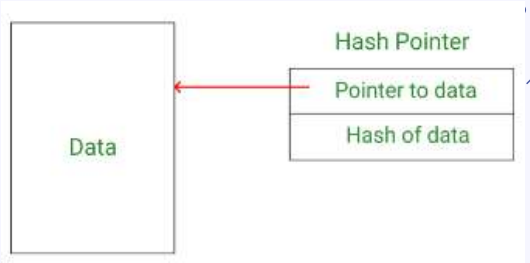
• Puzzle friendliness broadly means, with the data what value (nonce) we have to concatenate, so that the hash of the concatenate value will result in certain value which will have certain properties we would like to impose (restrictions).

•This property is used to construct the Bitcoin proof-of-work mining puzzle. Here a miner has to iterate over k (x is given in this scenario, it is a hash of the block that's being mined), and tries to find a value y that is in a certain range. For bitcoin, y needs to be smaller than a specific value determined roughly every two weeks by the Bitcoin software.

Hash Pointers:

• A regular pointer stores the memory address of data. With this pointer, the data can be accessed easily.
• On the other hand, a cryptographic hash pointer is a pointer to where data is stored and the cryptographic hash of the data is also stored.

With the hash pointer, we can:
• Retrieve the information.
• Check that the information has not been modified (by computing the message digest and then matching the digest with the stored hash value).



Hashing Technique for Searching Purpose:

1) Linear Search: In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. The time complexity of linear search is **O(n).**

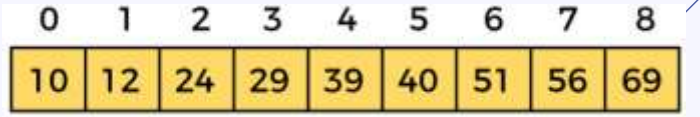
Let the elements of an array are -

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

Let the element to be searched is **K = 41.** The algorithm will return the index of the element matched. The worst-case time complexity of linear search is **O(n=6).**

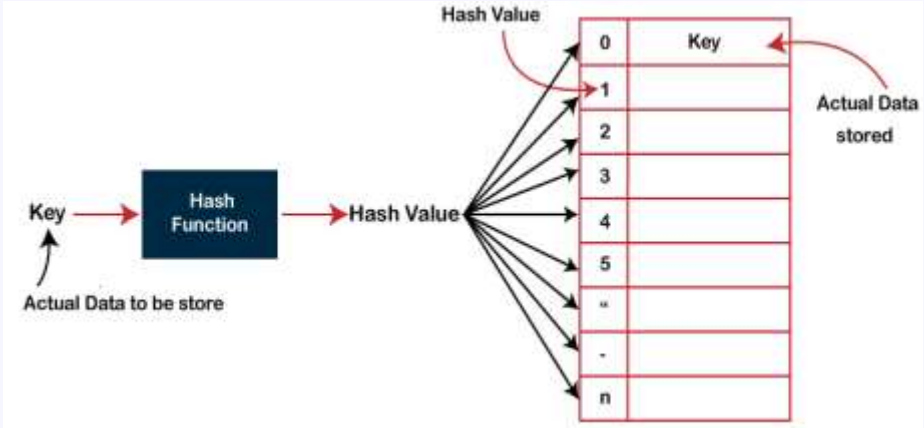
2) Binary Search: Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list we must ensure that the list is sorted (ascending order).
In the binary search, list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

The worst-case time complexity of Binary search is **O(log n).** Thus it is faster**.**



3) Hashing: Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is O(1). In Hashing technique, the hash table and hash function are used. Using the hash function, we can calculate the address at which the value can be stored.

If the key is given, then the algorithm computes the index at which the value would be stored.
index = hash (key).
If the key value is 6 and the size of the hash table is 10. When we apply the hash function to key 6 then the index would be: h(6) = 6%10 = 6.
The index is 6 at which the value is stored.
Thus, time complexity is O(1).



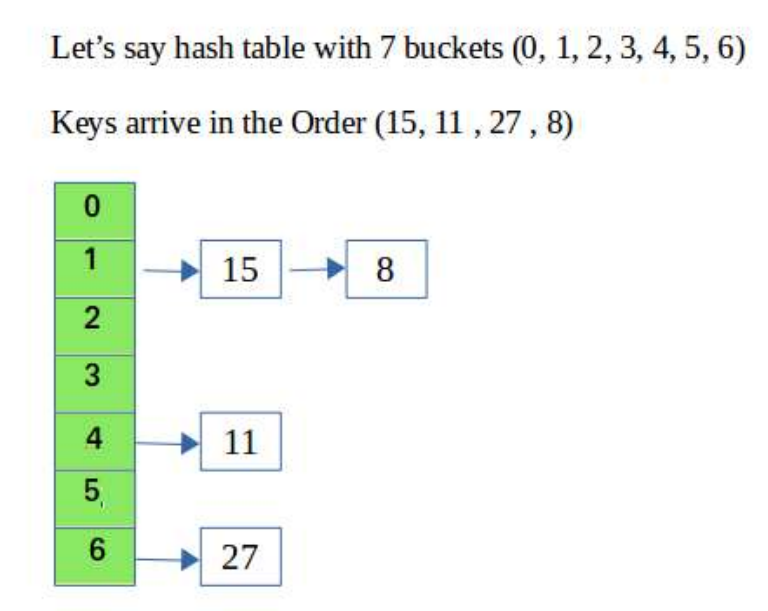
Hash Collision:

When the two different values have the same value, then the problem occurs between the two values, known as a collision. In the above example, the value is stored at index 6. If the key value is 26, then the index would be: h(26) = 26%10 = 6. Two values stored at the same index, leads to the collision problem. To resolve these collisions, collision techniques used are:

- Open Hashing: It is also known as closed addressing.
- Closed Hashing: It is also known as open addressing.

Open Hashing

- In Open Hashing, one of the methods used to resolve the collision is known as a chaining method.
- When multiple elements are hashed into the same slot index, then these elements are inserted into a singly-linked list which is known as a chain.
- It means, if two different elements have the same hash value then we store both the elements in the same linked list one after the other.
- We can use a key K to search in the linked list by just linearly traversing.



Closed Hashing : Linear Probing

In Linear probing, when the collision occurs by mapping a new key to the cell already occupied by another key, then linear probing technique searches for the closest free locations and adds a new key to that empty cell.

In this case, searching is performed sequentially, starting from the position where the collision occurs till the empty cell is not found.

Problem with LP is that, it may cause clustering of elements.

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Add the keys 10, 5, and 15 to the previous table .

Hash key = key % table size

2 = 10 % 8

5 = 5 % 8

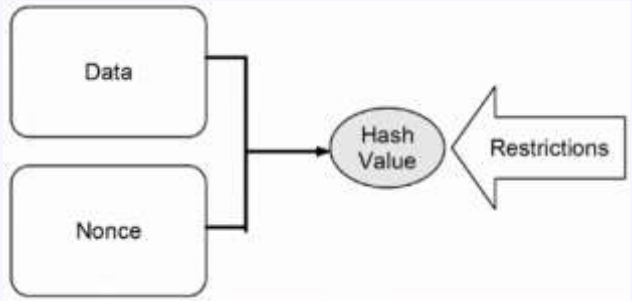
7 = 15 % 8

[0]	72
[1]	15
[2]	18
[3]	43
[4]	36
[5]	10
[6]	6
[7]	5

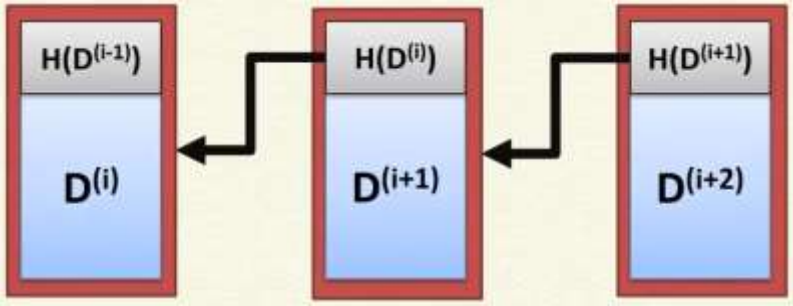
How to make Tampering a Hash Chain Computationally Challenging:

- A Nonce (number used once) is a random number that is appended to a block of data before it is hashed.
- It is a number added to a data block, when rehashed, meets the difficulty level restrictions.
- The nonce is the number that blockchain miners are solving for.
- When the solution is found, the blockchain miner that solves it is given the block reward.

www.blockchain-basics.com/HashPuzzle.html



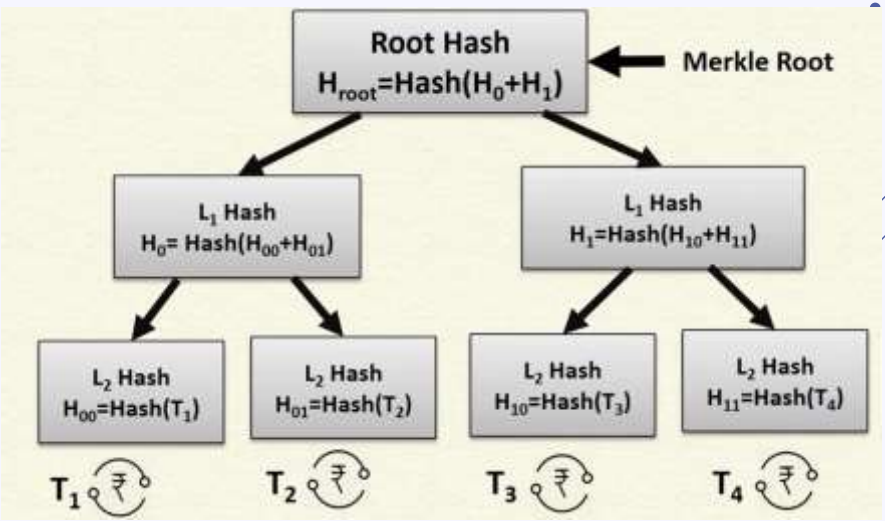
Tampering Detection from Hash Pointers - Hash Chain :



- The hash pointer will be more difficult to compute if the nonce is added., Example: hash of the data + nonce will have some restrictions (like 5 leading zeros or 10 leading zeros etc.).
- The difficulty in computing the hash pointer not only ensures puzzle-friendliness, but it also makes it extremely difficult for the attacker to change it.
- It will depend upon how much computational power the attacker possesses.
- In Hashchain, a data block contains a hash which is the hash of the previous block. Thus every data block will be linked to the previous data block through the hash pointer that points to the previous block.
- If the attacker wants to change data in D(i), he has to change hash pointer H(D(i)). But the hash pointer H(D(i+1)) depends upon both D(i+1) & H(D(i)). So it becomes computationally impossible for the attacker to change(update) the hash pointers(with restrictions) of all successive blocks and remain undetected.

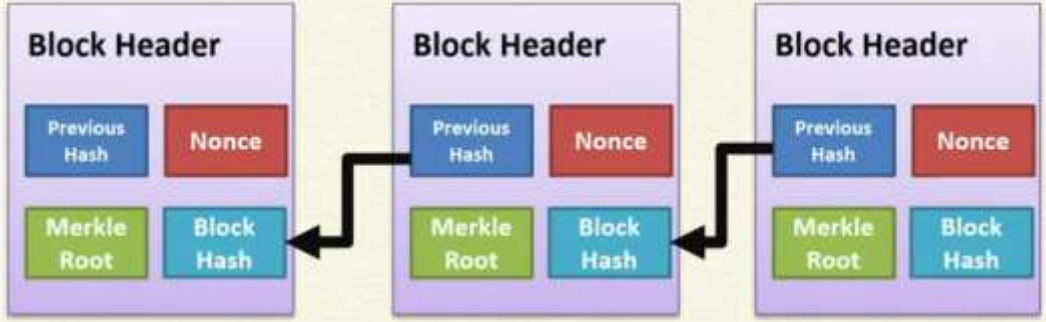
Merkle Tree (Hash Tree):

- The Merkle Tree is a binary tree. They're used to encrypt blockchain data more efficiently and securely.
- The hash of every transactions (Ti) is calculated and are stored in the nodes.
- Each Leaf node(L2) stores the hash of the transactions (data).
- One level above, the leaf nodes(L1) stores the hash of the concatenation of two L2 hashes.
- The two hashes (**Hash0** and **Hash1**) are then hashed again to produce the Root Hash or the Merkle Root.

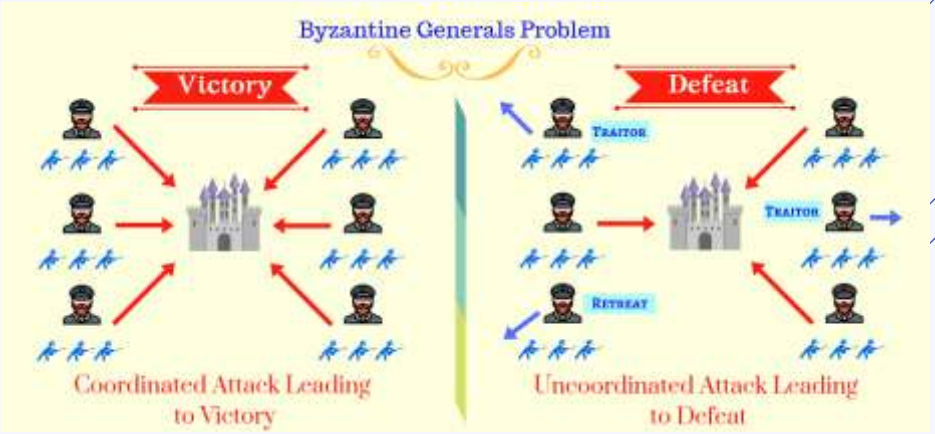


Blockchain as a Hashchain :

Each block in the blockchain will consist of a Block Header, Previous Hash, Merkle Root of the transactions or data, Hash of the block itself, the Nonce and the data. The next block will connect to the current block via the hash pointer and so on. Thus, a blockchain will be like a hashchain, where different blocks will contain different informations in a tamperproof manner.



Byzantine Generals' Problem:



- The Byzantine Generals Problem is a game theory problem.
- A group of generals attacks a fortress; every general has an army and surrounds a fort from one side.
- Every general can either attack or retreat.
- It has to be a coordinated attack or retreat to incur minimum losses.
- Thus, a consensus is held, and the majority decision is implemented.
- This consensus is formed after following the following steps:
    1) Every general sends their own choice to all other generals.
    2) After receiving the choice of all generals, every general calculates the votes in favor of attacking and retreating.
    3) If the majority is in favor of retreat, then they retreat; otherwise, they attack.
- However, any message they transmit or receive could have been intercepted or deceptively sent by Byzantium's defenders, the generals have no secure communication channels with one another.

The Problem: Additionally, suppose there is a traitor general who sends a retreat message to half generals and an attack message to the other half generals.

- Then half of the generals may end up attacking while the other half will retreat, causing the army to lose.
- The generals must find a way to reach a consensus despite the possibility of deception and betrayal.

Issues with Physical Currencies

Fiat money is legal tender, but it has no intrinsic value. In essence, it has value because the authorities that issued it say it does.

1) The fiat currency note has to be carried along with and hence may get defected or torn down which has to be exchanged at the bank.
2) It might get stolen and whoever gets it….it becomes his money.
3) The central authority have the control in order to establish a monopoly.

Cryptocurrency:

An automated payment system having the properties:

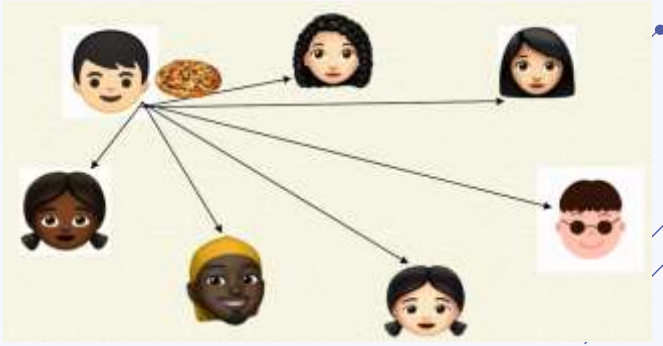
- **Inability** of the third parties to determine payee, time of payments made by individuals.
- **Ability to show** the proof of payment anytime.
- **Ability to stop** the use of payment media reported stolen.

Evolution of Cryptocurrencies: Digital Money

The first cryptocurrency was eCash, by David Chaum in 1983.

- Money is stored in the computer – digitally signed by the bank.
- Use a concept of “blind signature” to make the payment anonymous – the content of a message is kept disguised before it is signed.
- It was not so popular and ultimately failed.

Distributed Consensus : Limitation



Let us revisit the distributed consensus problem example again. Here each member makes a phone call to intimate his/her choice and a decision is taken on majority voting.
But the main point is that, everyone has to make a phone call. Thus the algorithm is based on **Message Passing**. Thus identity of the members is needed for message passing.
Hence, the classical distributed consensus algorithm works good only within closed system. But blockchain is a open network, anyone can join and anyone can leave at any point of time. So message passing will not work in open system as the identity of the participants are anonymous.

Bitcoin Proof of Work (PoW) : An Open Consensus

To solve this problem in open system, in 2008 Satoshi Nakamoto floated the pioneer whitepaper “Bitcoin: A Peer-to-peer Electronic Cash System”. It gave an idea to reach consensus in open network by avoiding the idea of message passing.

Consensus in an Open Network : Puzzle Solving

Problem: One of the members become the leader. He/she verifies the transactions to be committed to the block. But, how to elect the leader when no-one knows each other???

Solution: The idea of puzzle solving will elect the leader. Let the network itself elect the leader. How???

- The network throws a complex puzzle asking everyone to solve it. The member who could solve it will be elected as the leader.
- But what sort of puzzle he has to solve? It has to be a very difficult puzzle, as multiple member can solve a simple puzzle simultaneously. It will take time to solve, but ultimately it will elect the leader.
- Another important property of the puzzle is: although it is difficult to solve it, it should be easy to verify it.

- When the leader has solved the puzzle, other members can easily verify whether the leader has actually solved the puzzle or not. Thus, **Solving the puzzle is complex** but **Verification is easy**.
- Different member can solve the puzzle at different rounds. Thus different people will get elected in rounds.
- Let y=Hash(x||N). If (y,x) is given, the puzzle is to determine N, the nonce. This is a hard problem.
- The person who solves the N (nonce) first becomes the leader for that round. This is Proof of Work.
- Hash Chain + Puzzle solving as Proof + Coin Mining in open P2P setup was proposed by Nakamoto.
- Proof of Work (PoW) or Nakamoto Consensus gives more emphasis to “**Liveness”** rather than “**Safety**”. Thus, participants may agree on a transaction that is not the final one in the chain.

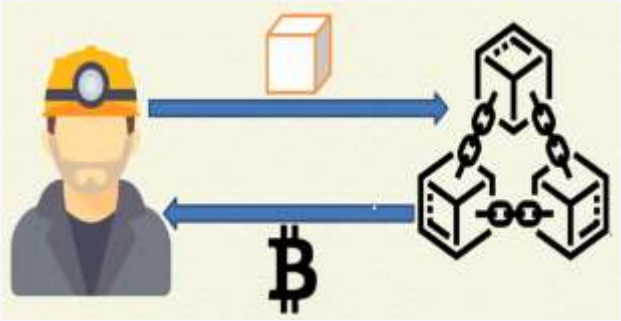
Bitcoin Mining

Solved the complex puzzle requires a lot of computational power. Then why any participant will solve the puzzle without having any benefit? What's the kind of reward or incentive they are going to get?

- There are special nodes (with miner application), called the **Miners.**
- Miners "hears" the transactions that are floated in the network. They collect those, combine those transactions and form a block. This block is needed to be added to the blockchain.
- Miners propose new blocks – solve the puzzle (find the nonce corresponding to a target block hash), and add the solution as a **proof** of solving the challenge.
- Solving the challenge needs some work to be done. Hence **Proof of Work (PoW).**
- Other miners & normal participants can easily verify this 'work done' from the block added to the chain.
- Competition for the next block begins. As there is no deterministic algo to solve the puzzle. Everyone has to try randomly. Hence, its more likely different person wins in different rounds. So single miner do not gets control over the network every time.

What is the reward?

Earn Bitcoin as a reward by successfully solving the puzzle and adding a block to the chain.
Hence, It is a perfectly people-driven financial network.
But, wherefrom these Bitcoins are coming ????



The Economics behind the reward.

The Miners while getting reward for successfully committing a block to the Bitcoin network, actually produces new Bitcoin in the system. It is similar to minting new coins.
As it is generating more Bitcoins into the system, so it is known as Mining.

The Price of Bitcoin:

Bitcoin value has increased drastically over time:

- May 2010 : < $0.01
- April 2014: $340 - $530.
- December 2017: $13800
- November 2021: $68000
- August 2023: $29500

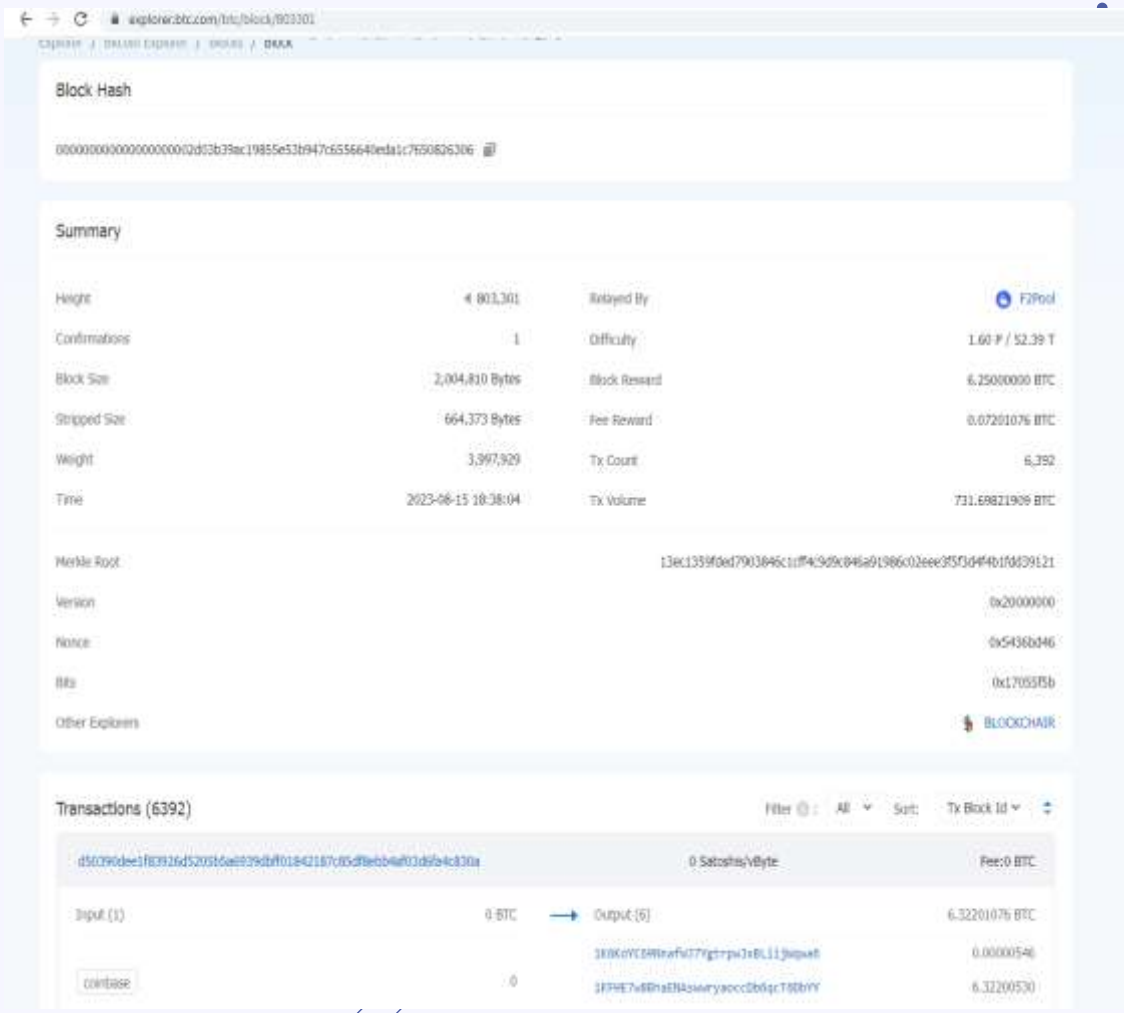
Blockchain 1.0: Distributed Ledger

•Distributed ledger technology (DLT) is a digital system for recording the transaction of assets in which the transactions and their details are recorded in multiple places at the same time. Unlike traditional databases, distributed ledgers have no central data store or administration functionality.
•It is the technological infrastructure and protocols that allow simultaneous access, validation, and record updating across a networked database.
• The transparency of DLT provides a high level of trust among the participants and practically eliminates the chance of fraudulent activities occurring in the ledger.

Structure of a Bitcoin Block

- A block is a container data structure that contains a series of transactions.
- In Bitcoin, a block may contain more than 500 transactions on average. The average size of a block is around 1 MB.
- It may grow up to 8 MB or sometimes higher.
- Larger blocks can help in processing large number of transactions at one go, but it requires longer time for verification and propagation.
- It consists of two components:
  - **i) Block Header**
  - **ii) List of Transactions.**
- Here 803301 indicates the block no.
- The 1st block is known as the Genesis Block.

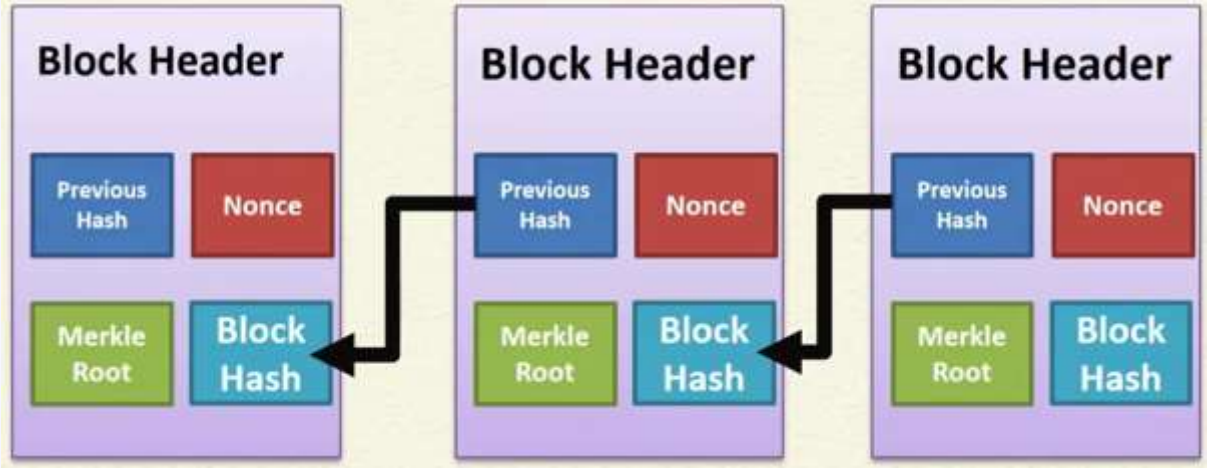
Ref: btc.com/btc/blocks



Block Header

It contains:

- Metadata about the block : 1) Previous block hash. 2) Mining statistics used to construct the block. 3) Merkle tree root.
- Previous block hash: Every block inherits from the previous block to make the blockchain **tamper proof**.
- Mining statistics contains – timestamp , nonce and difficulty.



Bitcoin Scripting Language

Let Alice has 100 BTC and wants to transfer 40 BTC to Bob. How Bob will verify that the transaction actually originated from Alice?

•A and B are bitcoin addresses that are generated from the respective public keys of Alice and Bob.

•Alice will send her Public Key along with the Signature – so that Bob can verify. Transaction is signed with Private Key of Alice.

•Bob can decrypt the signature with the public key of Alice and verify that it is indeed from Alice.

•In actual, this transaction will occur through **Bitcoin Scripts**.

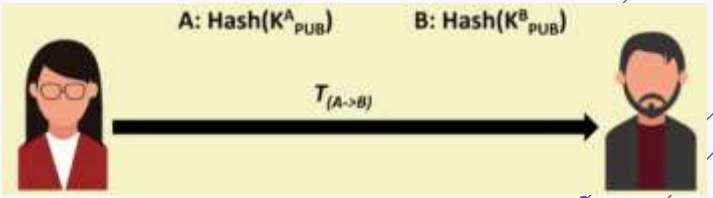
• Bitcoin indeed transfers scripts instead of the signature and the public key.

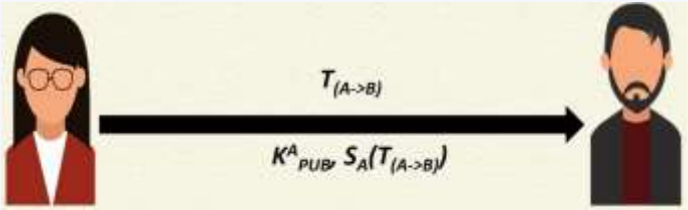
• Bob can spend the BTCs to Tom, only if both the scripts return TRUE after execution.

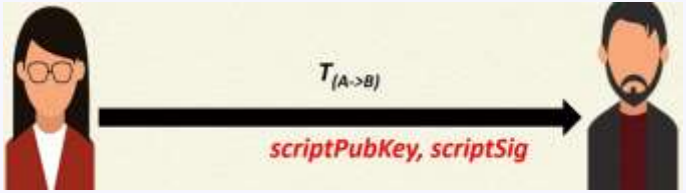
• It should not be the case as Alice is willing to send 40 BTC to Jack & somehow Bob gets hold of it & is going to send to Tom.

• Bitcoin scripts are simple, compact, stack-based and processed left to right. (FORTH like language).

• Bitcoin scripts are **Not Turing Complete** (no Loops). It is not capable of solving any type of problem such as **turing machines**.







Pay-to-Public-Key-Hash (P2PKH)

The majority of transactions processed on the bitcoin network occurs with a Pay-to-Public-Key-Hash or "P2PKH" script.

These outputs contain a locking script that locks the output to a public key hash, more commonly known as a bitcoin address.

The complete script consists of two sections, the unlocking script (**scriptSig**) and the locking script (**scriptPubKey**). The locking script is from the transaction output that is being spent, while the unlocking script is included in the transaction input that is spending the output.

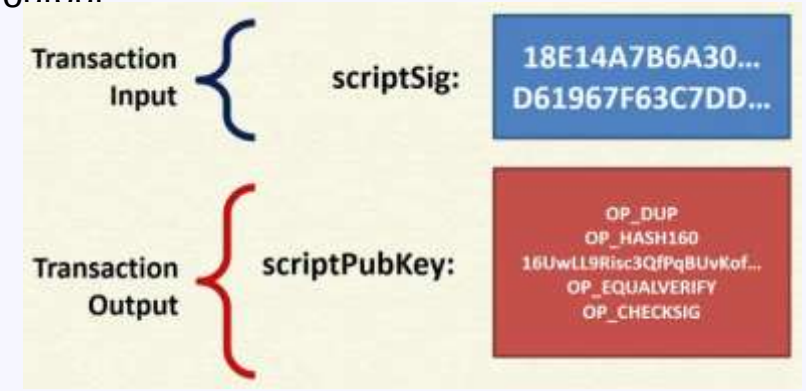
The OP_CODES that are used in the operation are :

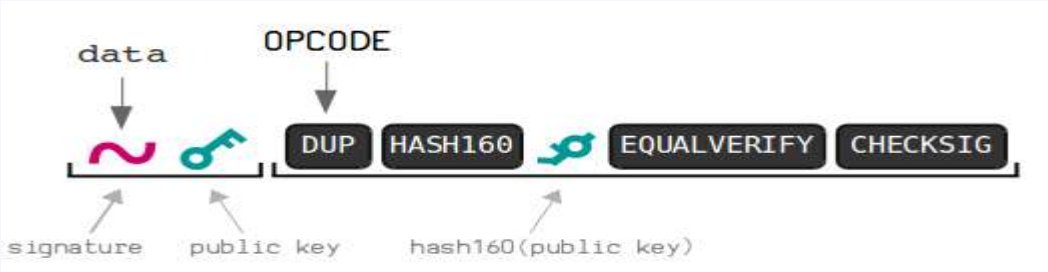
OP_DUP: Duplicate the item on the top stack.

OP_HASH160: The input is encoded twice: first with SHA-256 and then with RIPEMD-160.

OP_EQUALVERIFY: Verify that the data entered is correct and valid.

OP_CHECKSIG: The outputs, inputs, and script of the entire transaction are summarized in a hash. It verifies the signature with the public key.



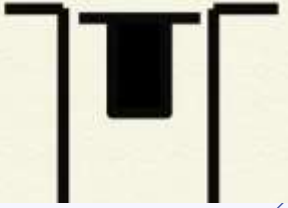


Transaction output : Alice sending 40 BTC to Bob.
Transaction input: Bob is claiming those 40 BTC. He can pay some or full of those to say Tom.

• The stack is initially empty. Both the scripts are combined – Input followed by output.

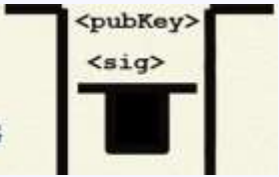
• Top two items are pushed into the stack, one after another.

```
scriptPubKey: OP_DUP
OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

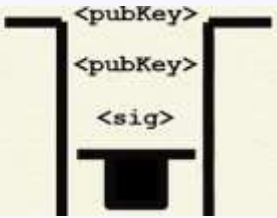


```
<sig> <pubKey> OP_DUP OP_HASH160
<pubKeyHash> OP_EQUALVERIFY
OP_CHECKSIG
```

```
<sig> <pubKey> OP_DUP
OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
```

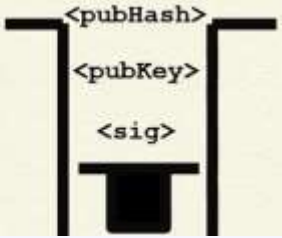



```
OP_DUP OP_HASH160
<pubKeyHash> OP_EQUALVERIFY
OP_CHECKSIG
```



• Top stack item is duplicated by OP_DUP. <pubKey> was there at the top of the stack, hence it will be duplicated.

```
OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
```



• Top stack item is hashed (RIPEMD-160 hashing). Hash of the duplicate <pubKey> will be generated.

The constant <pubKeyHash> is pushed in the stack.
This constant is provided by Alice as the BTC wallet address of Bob, to which she is sending 40 BTC.

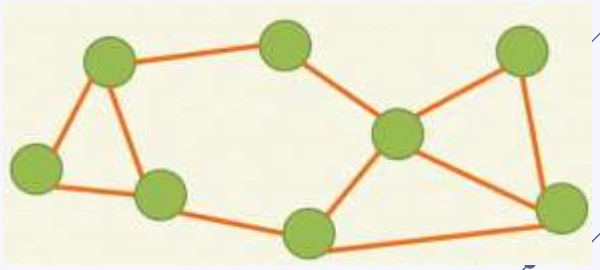
Equality is checked between the top two items in the stack. If TRUE, then only the transaction can proceed.
<pubHash> generated from hashing the <pubKey> of Bob will be checked with <pubKeyHash> provided by Alice.

Signature is checked based on the top two stack items. When TRUE, it proves the identity of Bob.
In this step Bob ensures that the <pubKey> is indeed his ownership.

Example: https://explorer.btc.com/btc/transaction/d707e4e7b6dfd24ad0853e259bf38a4d6ae07d5c2f1fd5097d999d1bbef5821c

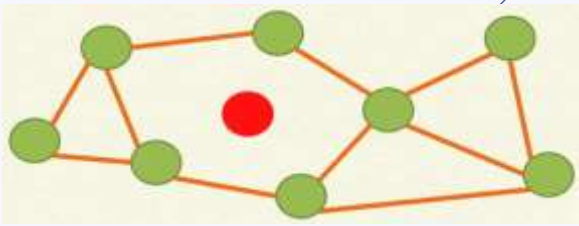
Bitcoin P2P Network

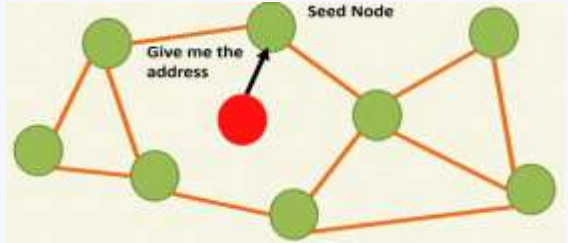
• An ad-hoc network (no fixed topology) with random topology. Bitcoin protocol runs over TCP.
• All nodes (users) in the bitcoin network are treated equally.
• New nodes can join any time. Non-responding nodes are removed after 3 hours.

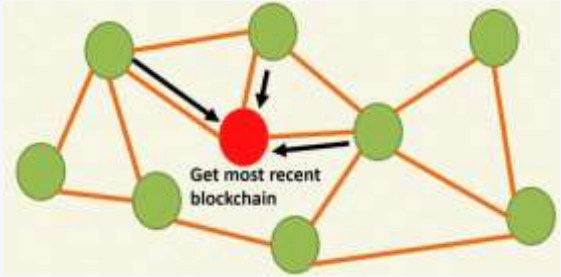


Joining in a Bitcoin P2P Network

• Joining in a Bitcoin P2P network implies downloading the installation script. Execute the script to become a potent node in the network.
• It will ask a Seed Node for the address to join the network. Seed node will send the address list to this new node.
• In Bitcoin network, a typical node do not communicate with all the other nodes. It will communicate with its neighboring (peer) nodes only. These neighbors will communicate with other peers and so on to the entire network.
• The new node will then ask and get the most recent blockchain. It gets at per the thousands of existing nodes. It is now ready to start new transaction.
• Let the new node is of Alice, and she want to send 10 BTC to Bob.
• Alice broadcasts this transaction in the bitcoin network.







• Alice will be sending this transaction info to her peers. The peers will send it to their peers and so on. This is **Transaction Flooding** in the bitcoin network. The protocol is known as **Gossip protocol**.

• Every node will now validate the transaction. Then they will flood the transaction in the network.

• In a typical case, any peer node will get the transaction info directly from Alice and also from other peers. To avoid duplication, it will not add the same transaction again, as if "I have already seen the Trx.".

• The transactions will be considered as valid and should be relayed:

i) No conflict.

ii) No double spending.

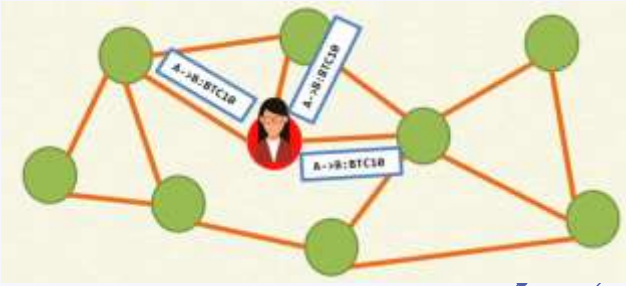
• The node will accept the first transaction it has heard. Different nodes will have different transaction lists at any point of time.

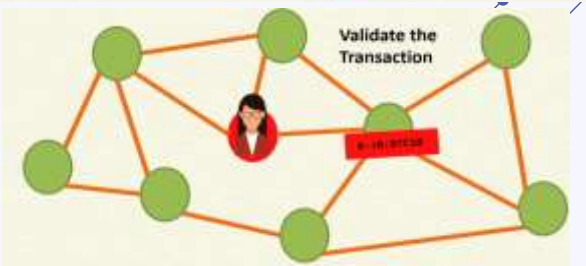
• The miner nodes will collect all the flooded transactions, and starts mining a new block to add to the bitcoin network.

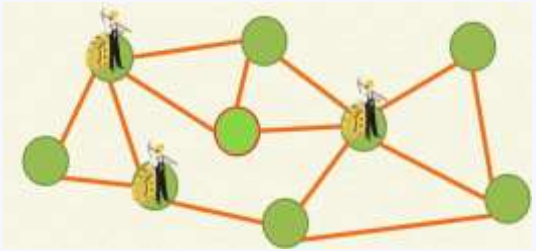
• The miner who solves the puzzle first (find the nonce), will generate a new block. He will get the block reward for his PoW. The other miners effort will go waste. They will again compete for a new block.

• The wining miner will now flood the blockchain with the new block to get added with the other peer nodes and beyond.

• The other miners will then remove any duplicate transaction from it's cache that has been included in this new block & will add fresh trnxs.





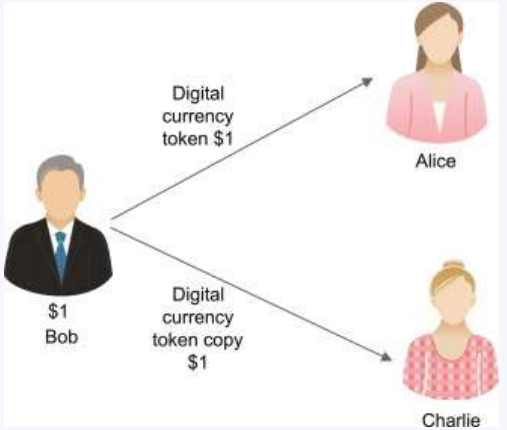


Creation of Bitcoins – Properties.

- **Controlled Supply:** There is a limited supply of bitcoins that can ever exist, with a total cap of **21 million**.
- Must be limited for the currency to have value.
- Currently, around 19 million bitcoins have been mined and are in circulation, leaving approximately 2 million left to be mined.
- BTCs are generated during the mining as reward – each time a miner discovers a new block.
- The mining reward for each block of transactions is currently 6.25 BTC, but this amount is halved approximately **every four years** in a process called a **halving event**.
- The last halving event occurred in June 2024, and the next one is expected to occur in 2028.
- It's estimated that all bitcoins will be mined by the year **2140**, at which point the last block reward will be released.

Double Spending

- Double spending means spending the same money (digital currency) twice.
- Since digital currencies are nothing but files, a malicious user can create multiple copies of the same currency file and can use it in multiple places.
- Double spending can never arise physically. It can happen in online transactions.
- Suppose a user B wants to avail of services from ‘A’ and ‘C’.
- B first made a digital transaction with ‘A’.
- The copy of the cryptocurrency is stored on the B’s computer.
- So B uses the same cryptocurrency to pay ‘C’.

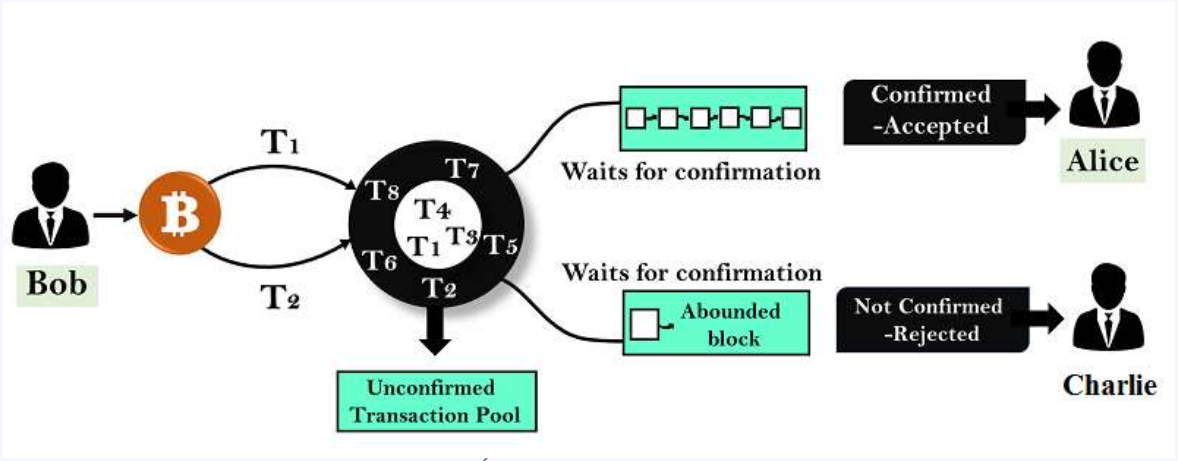


• Now both A and C have the illusion that the money has been credited since the transactions were not confirmed by the miners.
• This is the case of double spending.

How Bitcoin handles Double Spending?

• Bitcoin handles the double-spending problem by implementing a confirmation mechanism and maintaining a universal ledger called blockchain.
• Let Bob have 1 BTC and try to spend it twice. Bob made the 1 BTC transaction to Alice. Again, Bob send the same 1 BTC transaction to Charlie.
• Both transactions go into the pool of unconfirmed transactions where many unconfirmed transactions are stored already.

• Now, whichever transaction first got confirmations and was verified by miners, will be valid.
• Another transaction which could not get enough confirmations will be pulled out from the network.



Zero-Knowledge Proof (ZKP)

"A zero-knowledge (ZK) proof is a cryptographic protocol that enables one person (the prover) to convince another (the verifier) that a particular claim is true without disclosing any details about the claim itself".

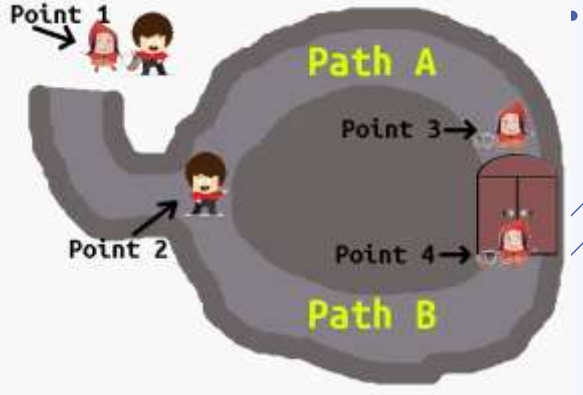
Zero-knowledge proofs must satisfy three properties:

- **Completeness:** if the statement is true, an honest verifier will be convinced by an honest prover.
- **Soundness:** if the statement is false, no dishonest prover can convince the honest verifier. The proof systems are truthful and do not allow cheating.
- **Zero-Knowledge:** if the statement is true, no verifier learns anything other than the fact that the statement is true

Example: Ali Baba Cave

- Let a circular cave containing a door that unlocks only with a specific secret code. Alice claims to know the code, and wants to prove this to Bob, without revealing anything about the secret code.
- Alice enters the cave, and walks up to Point 3 or 4. From there, she can return to the entrance via two paths, Path A and Path B.
- Once she is at one of the points, Bob comes to the entrance (Point 2) and calls for her to arrive there via one of the paths, which he chooses on random.

• Depending on which side of the door Alice is, she always has a 50% chance of being able to return to Bob without passing through the door.
• Otherwise, she needs to know and use the secret code.
• If this exercise is carried out multiple times, the probability that Alice exits through the same path selected by Bob without knowing the code progressively reduces.
• Thus, Bob get convinced that Alice actually knows the secret code and to this end there was no need to share the actual code.



Zero-Knowledge Proof (ZKP) in Blockchain

• ZKP has immense potential in a wide variety of applications where sensitive information is required, such as providing proof of password, proof of identity, and proof of membership.
• It can allow authentication with untrusted or unidentified parties over an untrusted communication channel.
• Cryptocurrencies like Monero and ZCash make use of ZKPs to maintain a high level of user and transaction privacy for their users

Types of Zero Knowledge Proof (ZKP):

There are two forms of zero knowledge proof (ZKP):

i) Interactive ZKP.

ii) Non-Interactive ZKP.

In interactive ZKP, the prover has to perform a series of actions to convince the verifier of a certain fact. For example, in the Alibaba cave example, Alice has to come out of any of the two paths inside the cave to prove to Bob. So everytime there needs to be interaction between Alice and Bob or any other verifier to whom Alice has to prove. Transactions has to be verifiable by every nodes in the blockchain.

Problems with traditional Interactive ZKP and remedy:

- Interactive ZKP might be expensive in communication.
- Proof size might be very large, resulting in blockchain size explosion and hence not scalable.
- Only designated verifiers are able to verify the proof, not compatible with 'public' ledge concept.

Therefore we need few more properties to make ZKP compatible with the public concept of blockchain:

- **Non-interactive:** The prover only sends one message to the verifier.
- **Publicly verifiable:** Everyone can verify the proof without any secrets.
- **Succinct:** The proof should be very short. No scalability issue.

Zk-SNARKs is one kind of ZKP that satisfies all these requirements.

- Completeness.
- Non- interactive
- Zero-knowledge
- Publicly verifiable
- Soundness.
- Succinct.

Zk-SNARKS : for anonymity preservation in Blockchain

The acronym zk-SNARK stands for **Zero-Knowledge Succinct Non-Interactive Argument of Knowledge** and refers to a proof construction where one can prove possession of certain information, e.g., a secret key, without revealing that information, and without any interaction between the prover and verifier.

- "Succinct" zero-knowledge proofs can be verified within a few milliseconds, with a proof length of only a few hundred bytes even for statements about programs that are very large.
- The mathematical basis of zk-SNARKS is complex.
- Nonetheless, proofs of this type allow one party to demonstrate not only that a particular bit of information exists, but also that the party in question has awareness of that information.
- In the case of Zcash, zk-SNARKs can be verified nearly instantly, and the protocol does not require any interaction between the prover and the verifier.