Module 1

Introduction to Blockchain

Syllabus

1.1 What is a blockchain, Origin of blockchain (cryptographically secure hash functions), Foundation of blockchain: Merkle trees.

1.2 Components of blockchain, Block in blockchain, Types: Public, Private, and Consortium, Consensus Protocol, Limitations and Challenges of blockchain.

1.1 INTRODUCTION

GQ. Define blockchain and explain it with a suitable example.

- By definition, blockchain is a *decentralized computation and information sharing platform* that enables *multiple authoritative domains*, who do not trust each other, to *cooperate*, *coordinate*, and *collaborate* in a rational decision making process.
- The keyword that we have in the abovementioned definition is decentralized, which is an important aspect of blockchain.
- Computation and information sharing platform is the next keyword that will help in specifying blockchain broadly to be a decentralized database, which helps in cooperation between multiple authoritative domains.
- This technology is useful when multiple parties or individuals want to cooperate with each other and they want to come to a common platform so that they can share information amongst themselves.

(1A1)**Fig. 1.1.1 : Structure of a decentralized network**

- Multiple authoritative domains do not trust each other, so what blockchain does is it combines these multiple authoritative domains in a common platform so that they can cooperate, coordinate, and collaborate in application development process or business intelligence process.
- Traditional way of sharing documents, which we do using Microsoft Word. For example, Person A wants to share some document with Person B. So, ideally what Person A will do is he/she will write the content in a Microsoft Word document and then he/she will share that document with Person B.
- Person B will first receive the document sent by Person A and then Person B will update the document with his/her content and share the document again with Person A.
- This was the traditional way of cooperation between Person A and Person B when they wanted to write something in a shared document. In this specified process, one of the main disadvantages is that Person A and Person B will not be able to simultaneously edit the document. The ideal solution where simultaneous editing of the document can be done is with the help of shared Google docs where both Person A and Person B can edit or write the document simultaneously.
- However, the problem of shared Google doc platform is that the environment is still centralized.

(1A2)**Fig. 1.1.2**

(1A3)**Fig. 1.1.3**

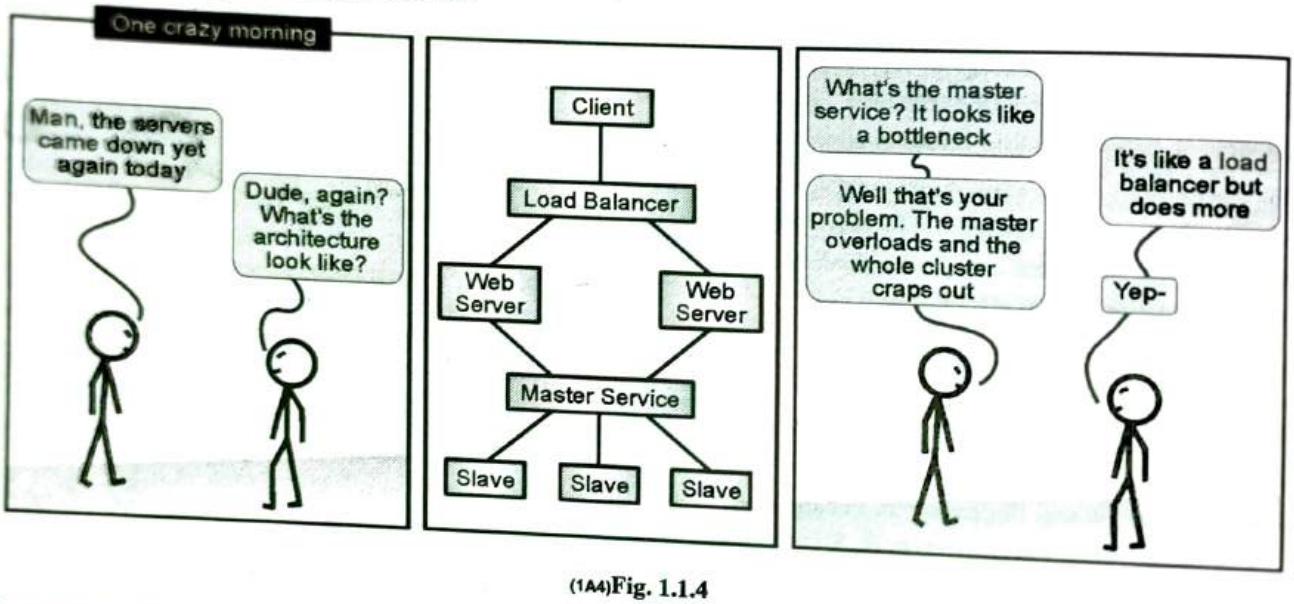
- Now, the question arises that does this type of centralized system harm or what is the disadvantage that if we use a centralized environment for cooperation during this kind of information sharing platform.

1.1.1 Problems with a Centralized System

GQ. What problems are associated with a centralized system?

GQ. Differentiate between centralized, decentralized, and distributed architecture.

- The major problem of a centralized system is that it works as a single point of failure. For example, if sufficient bandwidth is not available to load the Google doc, then a person will not be able to edit the document. You will have to wait until you connect to the Internet and load the Google doc platform. The other problem that could arise is what if Google's server or your computer crashes.

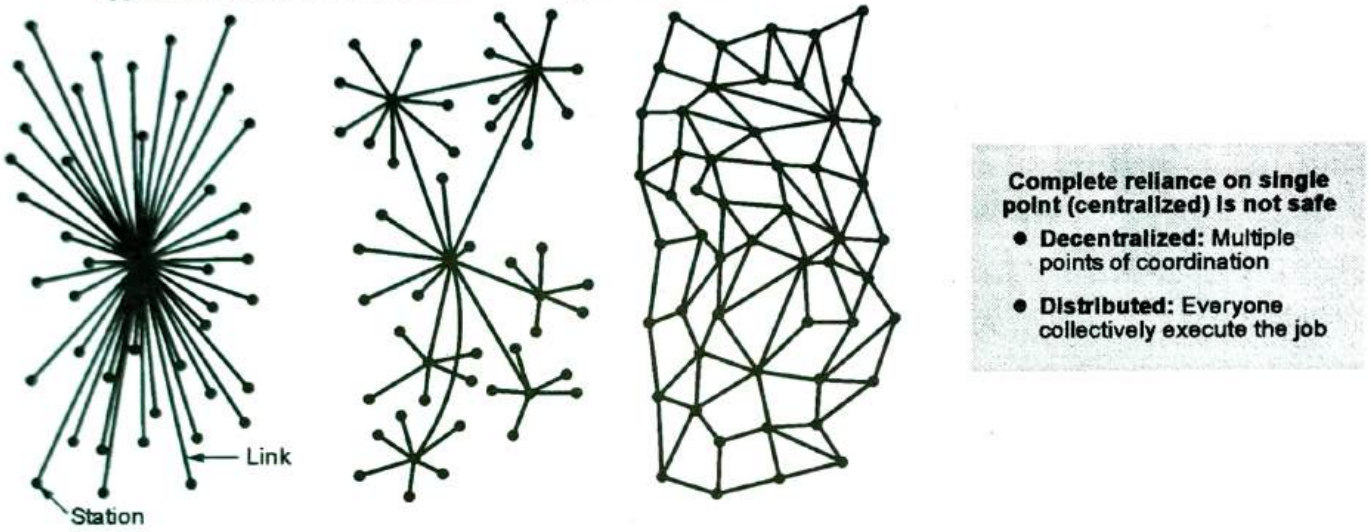


(1A4)Fig. 1.1.4

- In that case, the entire information gets lost or even if you want to take a backup, you will have to load the backup and only after loading the backup, you will be able to process it further. Because of the abovementioned reasons, it is necessary to shift from a centralized platform to a distributed platform.
- In a distributed platform, there are three different types of architecture:

1. **Centralized architecture :** There is a central coordination system and every node is connected to the central coordination system and whatever information the nodes want to share, the information shall be shared by the central coordination system. The problem with this architecture is that if the central coordination system fails, then all of the individual nodes will get disconnected.
2. **Decentralized architecture :** Here, there are few coordinators rather than a single coordinator, and all these coordinators cooperate with each other, and the individual nodes, they are connected to these coordinators. In this architecture, if a particular coordinator fails or multiple coordinators fail, then the individual nodes whose coordinators have failed can get connected to other coordinators that are working and can share the information or perform the operation with the available coordinators. In this type of architecture, the advantage is that multiple number of failures can be tolerated until the network becomes disconnected. This architecture works on top of a network and because of multiple simultaneous failure if the network gets partitioned or the network gets disconnected, then the individual nodes will not be able to cooperate with each other. This problem can be solved by a complete distributed architecture.

3. **Distributed architecture :** There is no centralized coordinator and all the nodes participate in the information sharing process or application development where they coordinate with each other and collectively develop the application or share the information amongst themselves.



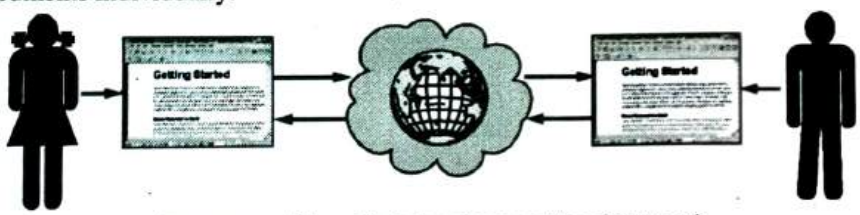
(a) Centralized (b) Decentralized (c) Distributed

(1A5)**Fig. 1.1.5**

- These are the advantages or disadvantages of a decentralized platform and distributed platform over a centralized platform. However, in the today's world, it has been observed that centralized architectures are good but are not scalable and not robust to failure.
- Therefore it is essential that we move from a centralized architecture to a decentralized architecture or distributed architecture. Blockchain is a platform that supports a decentralized platform or distributed platform where information can be shared amongst users in a trust worthy manner.

1.1.2 An Ideal Solution using Blockchain

- In a blockchain platform, Person A has his/her own copy of the document, and Person B too has his/her own copy of the document. Person A and Person B can simultaneously write to their own document and there is a network in between that has the task to ensure that information consistency is maintained between the documents with Person A and Person B who hold the documents individually.



Everyone edits on their local copy of the document - the Internet takes care of ensuring consistency

(1A6)**Fig. 1.1.6**

- This is an ideal use case where blockchain platform can be used. This platform, which is spanned over a network, will help in creating this type of coordination where Person A will keep his/her own copy of the document and Person B will keep his/her own copy of the document.

- Both of them can independently write their personal copy and then the blockchain platform will ensure that they are entering inside the document are getting synchronized with each other, and with time, both of them will be able to see the most updated copy or they will be able to update over the most updated copy.
- This is the advantage of a blockchain technology over a complete centralized architecture or an architecture where a shared copy exists between multiple parties.

Blockchain

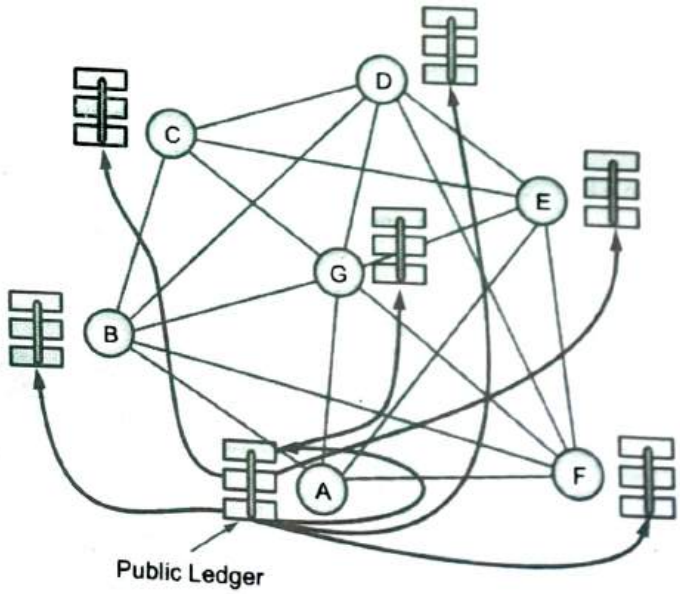
A decentralized database with strong consistency support

(1A7)**Fig. 1.1.7**

1.1.3 Simplified Architecture of a Blockchain

GQ. Explain with a suitable diagram, the simplified architecture of a blockchain.

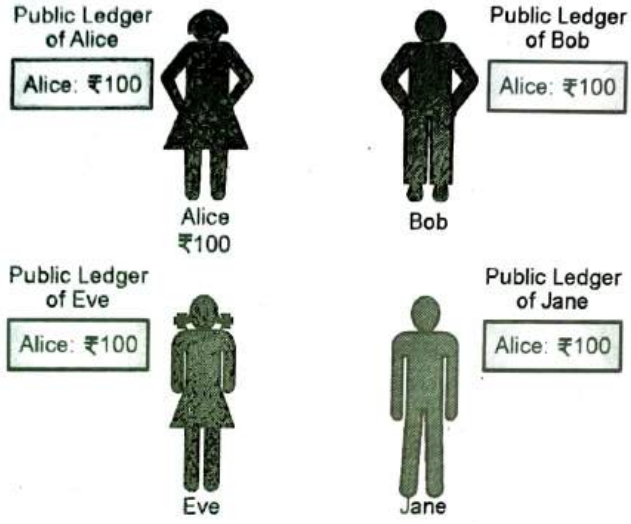
- In a typical blockchain architecture, every individual node maintains a local copy of the blockchain (i.e., a local copy of the global data sheet). The systems task is to ensure that all these individual copies are consistent with each other. Consistency means that the local copies that every node has are identical and these copies are always updated based on the global information.
- For instance, if node A wants to enter some information to its blockchain, then the entered information will get updated to all the copy of the blockchain that every node (i.e., nodes B, C, D, E, F, and G) possess. This is the architectural platform of blockchain which supports strong consistency among the local replica/local information that every node has.
- The local information is called a public ledger. A public ledger works like a database where it contains historical information which is available to everyone and this type of historical information can be utilized for future computation.
- An example of a public ledger could be a banking transaction. You are keeping the banking transactions inside a public ledger and the old transactions which are there are basically used to validate the new transactions.
- In a typical banking system, we usually maintain a passbook and the bank works like a centralized authority which stores all our transaction information and whenever an individual visits a bank and making a transaction, the bank validates everything with the centralized information that it has.



(1A8)**Fig. 1.1.8 : Simplified Architecture of a Blockchain**

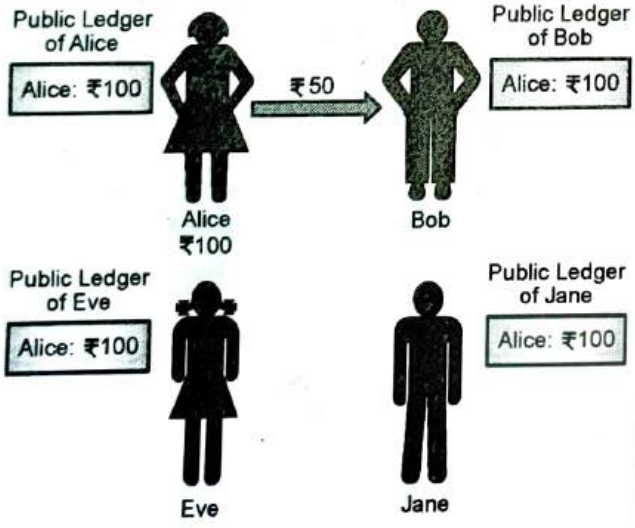
- Now, in case of a pubic ledger, there is a shift from a centralized banking system to a decentralized banking system where every individual has his/her own copy of the global transactions, which not only synchronized but also consistent, and whenever an individual attempts to make a new transaction, during that time the new transaction is validated against the old transaction that is already there inside a public ledger.

Example of a public ledger from a banking sector



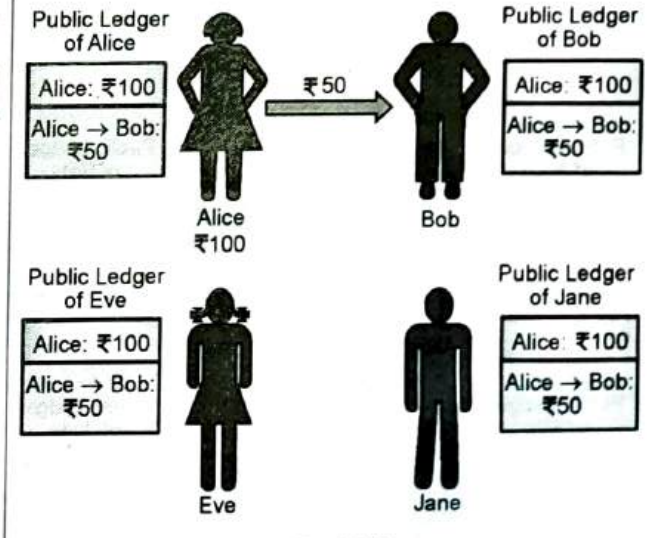
(1A9)**Fig. 1.1.9**

- There are four participants Alice, Bob, Eve, and Jane. Assume that Alice has Rs. 100 with here. The public ledger is available to Alice, Bob, Eve, and Jane, which has the initial content/information that Alice has (i.e., Rs. 100).



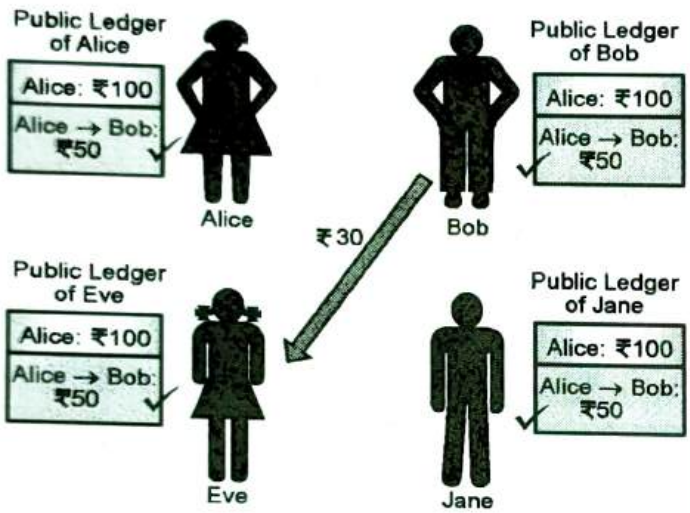
(1A10)**Fig. 1.1.10**

- Now, Alice wants to make a transaction of Rs. 50 to Bob. In this case, another information needs to be updated to all the public ledgers available with Alice, Bob, Eve, and Jane.
- Therefore, this transaction gets updated in all local copies of the public ledger.



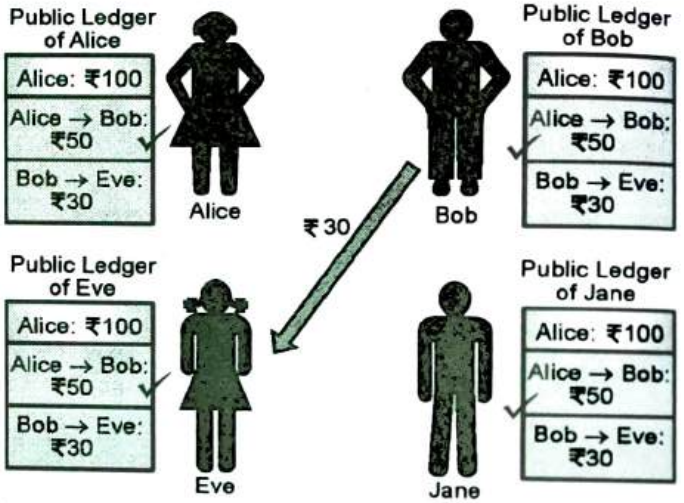
(1A11)**Fig. 1.1.11**

- Now, Bob wants to make a transaction of Rs. 30 to Eve.



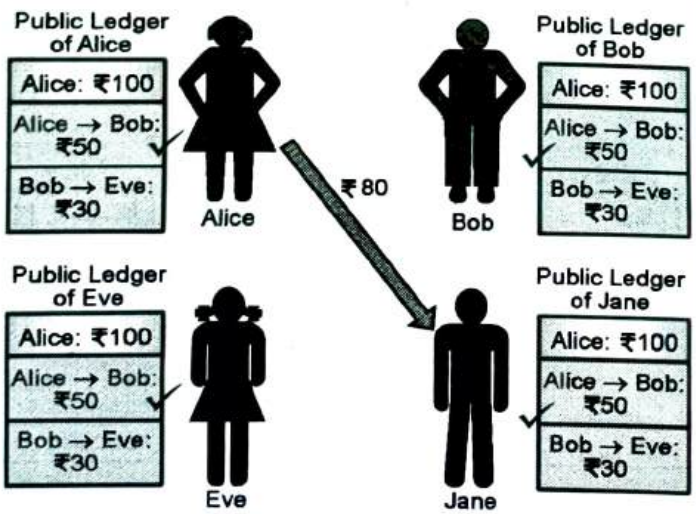
(1A12)**Fig. 1.1.12**

- So, the public ledger of Alice, Bob, Eve, and Jane gets updated with this latest information and this transaction gets updated in all local copies of the public ledger.



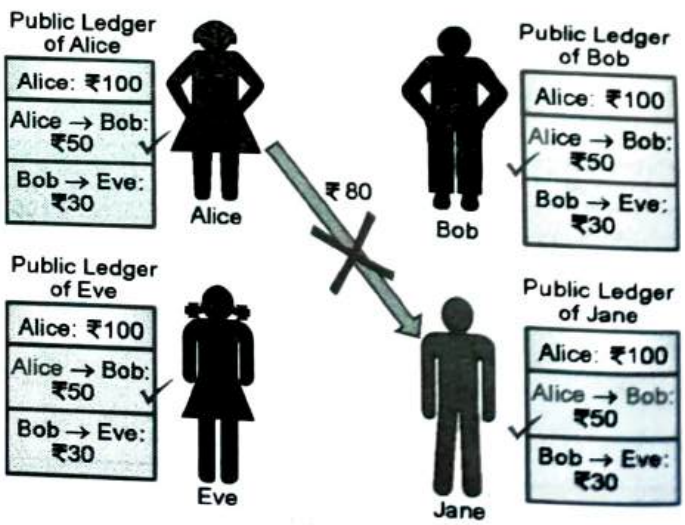
(1A13)**Fig. 1.1.13**

- Now, Alice wants to make a transaction of Rs. 80 to Jane.



(1A14)**Fig. 1.1.14**

- But, from Alice's public ledger, it is evident that she initially had Rs. 100 out of which she gave Rs. 50 to Bob. So now by combining both the transactions, she has only Rs. 50 available with her. If Alice tries to make a transaction of Rs. 80, then this particular transaction is not valid.
- Therefore, all parties will come to know from their public ledger that this transaction is not valid and it is blocked and it cannot be included in their public ledger.



(1A15) **Fig. 1.1.15**

- In this way a public ledger works in case of a decentralized system. Blockchain can be viewed as a public ledger. However, there are numerous aspects that need to be considered.
  - **Protocols for commitment :** "Ensure that every valid transaction from the clients are committed and included in the blockchain within a finite time". Whenever someone is making a new transaction, it has to be ensured that this transaction, if it is valid, then it should be committed to existing public ledgers or blockchain otherwise that entry will not be there in the blockchain. So, there should be a mechanism for validity checking of every upcoming transactions from the clients and based on those validity checks, the transaction should be accepted and added in the public ledger/blockchain or deleted or discard such transaction.
  - **Consensus :** "Ensure that local copies are consistent and updated." This is an essential aspect in the concept of blockchain. As discussed earlier, there exists a local copy of the information available to every individual parties and there is no such central platform like the bank which will maintain the consistency of the information. Therefore, consensus mechanism ensures that whatever local copy every individual party has, they are consistent/identical with each other.
  - **Security :** "Data needs to be tamper proof. Clients may act maliciously or can be compromised." The data that is inserted in a public ledger or blockchain needs to be secure and tamper proof. Because blockchain is distributed to individual parties, everyone is maintaining their local copy of their blockchain. So, it is possible that an individual might make a change in the local copy and broadcast saying that this is the updated information. But the other nodes in the network must be able to understand that whatever this individual is broadcasting is a false/tampered information and the other nodes should not accept such type of information.
  - **Privacy and Authenticity:** "Data/transactions belong to various clients; and privacy and authenticity needs to be ensured." The data that is present in the blockchain belongs to several clients, and so, it is essential that both privacy and authenticity needs to be safeguarded.

1.1.4 Formal Definition of a Blockchain

A blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way.

Essential keywords

- **Open :** Accessible to all
- **Distributed or Decentralized :** No single party control
- **Efficient :** Fast and scalable
- **Verifiable :** Everyone can check the validity of information
- **Permanent :** Information is persistent

1.2 ORIGIN OF BLOCKCHAIN

In this section, we shall look into the historical perspective of blockchain and how this concept of blockchain evolved over time and utilized into different applications.

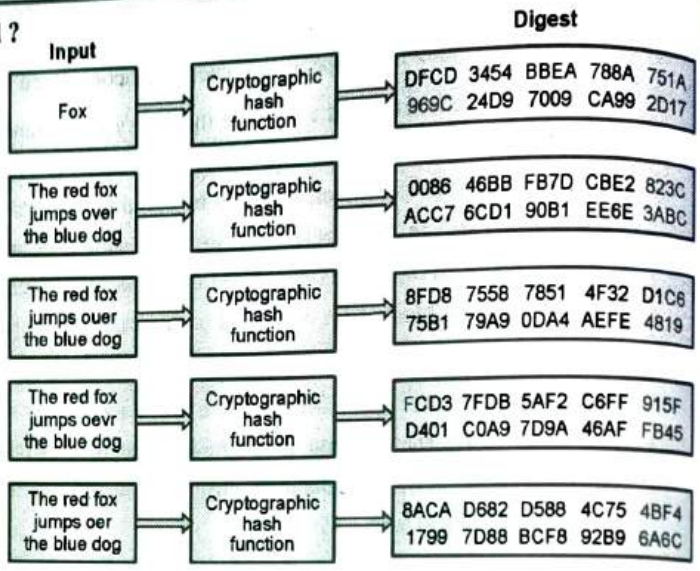
The fundamental aspects behind the evolution of blockchain are as follows:

- **Cryptographically secured hash functions**
  - Hash function : It is a function that maps any sized data to a fixed size. For example, we define a hash function

$$H(x) = x \% n,$$

  - Where x and n are integers and % is the modular (reminder after division by n) operations. x can be of any arbitrary length, but $H(x)$ lies between 0 to $n - 1$.

- **Why are they called cryptographically secured ?**
  - **Property 1:** Hash functions are called *one way* functions. *One way* means given x and n, we can compute $H(x)$, but when $H(x)$ is given, we cannot say what is the corresponding value of x (i.e., no deterministic algorithm can compute x).
  - **Property 2:** For two different x_1 and x_2, $H(x_1)$ and $H(x_2)$ should be different.
  - **Property 3 :** Avalanche effect: Let us consider x is the message and $H(x)$ is its respective message digest. Avalanche effect means a small change in x results in a significant change in $H(x)$.



(1A16)**Fig. 1.2.1**

In the above figure, the input x is "Fox" and its digest is `DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17`. In the second case, the input is "The red fox jumps over the blue dog" and its digest is `0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC`. Now, in the third case, we change "over" to "ouer" and we observe a significant change in the message digest `8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819`.

So, by simply observing the message digest, we cannot say that the original input was similar.

Similarly, in fourth and fifth cases, the message digest looks completely different.

Therefore, avalanche effect ensures that by merely looking at the digest, it is impossible to identify what was the input. Examples of cryptographic hash functions are MD5 and SHA256.

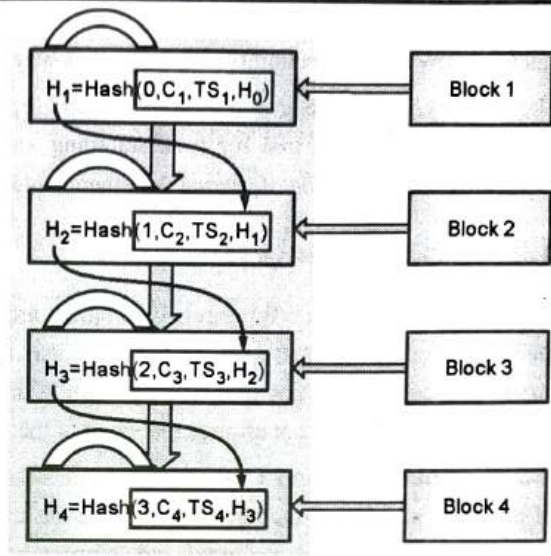
1.2.1 History of Blockchain

GQ. Explain the historical perspective of blockchain.

Cryptographically secured chain of blocks

- The first use of the concept of cryptographically secured chain of blocks came in 1991 in a paper by Harber and Stornetta. They have developed a mechanism for time stamping a digital document. In other words, you have a digital document that is edited by multiple people from time to time.
- Person 1 has created the document, Person 2 has edited the document, Person 3 has edited the document...so on and so forth. You want to maintain a list of timestamp when the document was 1st created followed by when it was edited by which person in a secure manner such that no people will be able to make a change in the time stamp value.

- This is important from the document purpose because you would like to know or see when the document was last edited or some people would claim that they haven't made changes in the document but in reality they could have made the change.
- To solve this problem, Harber and Stornetta used the concept of chain of blocks. They considered parameters like sequence number of access, client ID, timestamp, a hash value from the previous request and the entire thing is hashed to connect to the previous blocks.
- In the Fig. 1.2.2, 0 is the sequence number of access, C_1 is the client ID, TS_1 is the time stamp, H_0 is the hash value from the previous request and (0, C_1, TS_1, and H_0) are hashed and stored in H_1. H_1 is one of the components that is hashed to obtain H_2.



(1A17)**Fig. 1.2.2 : Cryptographically secured chain of blocks**

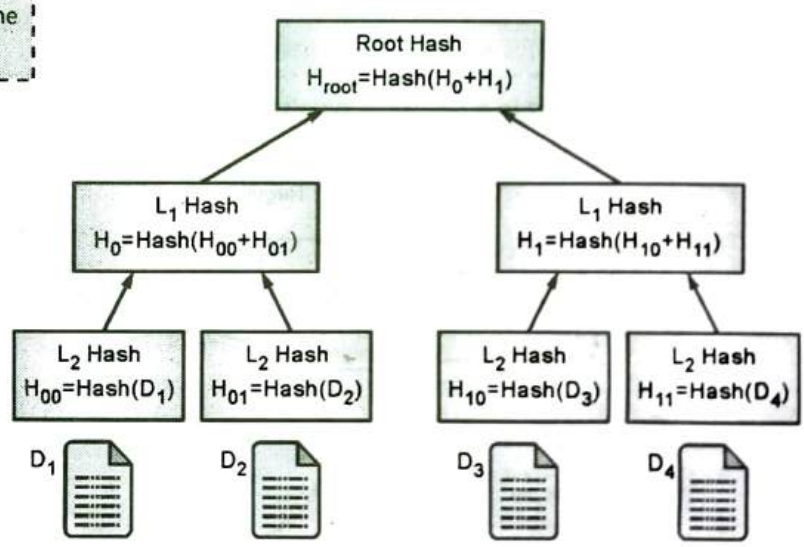
- The advantage of this hash chain is that if you want to make a change in the time stamp value (i.e., TS_1), then we need to make changes in H_2, H_3, and H_4 and people will be able to observe that values have been changed.
- This is how the concept of chain of block by connecting them with the hash function was used to cryptographically secure the time stamp value of a digital document. This is how individual hash values are helping to connect the blocks one after the other and making the blocks as tamperproof.

1.2.2 Foundation of Blockchain

GQ. What are Merkle trees ? Explain the structure of a Merkle tree.

- **Merkle Trees :** It is a tree structure where the leaf nodes will contain the hash of the document and every individual node or the intermediate node will contain the hash of the combination of the left child and the right child.

  The example of a binary Merkle tree is Fig. 1.2.3.
- The leaf node (L_2) contains the hash of the contents of the documents D_1, D_2, D_3, and D_4. The intermediate node contains the hash value of H_{00} and H_{01} and their combined hash value (i.e., H_0).



(1A18)Fig. 1.2.3 : Structure of a Merkle tree

- At root level, the root contains the combined hash (i.e., H_{root}) of its left child and the right child. This means that if any change is made in any of the documents, then the change will get reflected in their respective hash values. For instance, if any change is made in D_2, then the change will get reflected in H_{01}, H_0, and H_{root}.
- Now, if we wish to secure a number of documents together (i.e., D_1, D_2, D_3, and D_4), then we can just propagate or publicize the root value.

- If there is any change in one of the four documents, then that change will get reflected in the root value. This is how we can collectively secure a number of documents together by using this concept of Merkle tree.
- The concept of Merkle tree was used in 1992 by extending the work of Harber and Stornetta. Bayer, Harber, and Stornetta used Merkle tree for time stamping and verifying a digital document. The efficiency was improved by combining time stamping of several documents into a single block.

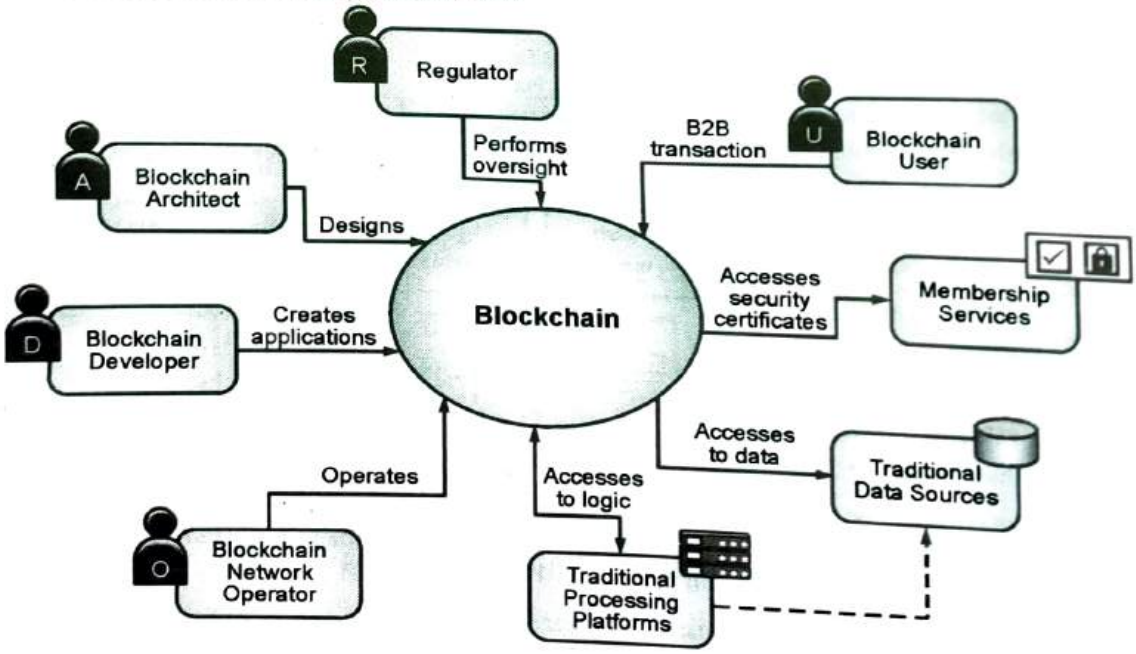
1.2.2.1 Other uses of Merkle Tree

- **Peer-to-peer networks :** Whenever data blocks are shared in a peer-to-peer network, it is essential that data blocks are received in an undamaged or in an unaltered way and other peers are not lying about a block (i.e., they are sharing a block, but that block is not an updated block). In that case if the root hash of the Merkle tree, which is the Merkle root, is shared, then Merkle root ensures that none of the documents has been altered.
- **Bitcoin implementation :** This is one of the most popular use of the Merkle tree (i.e., bitcoin implementation). It is one of the most popular cryptocurrency which forms one of the basis for blockchain technology. We want to identify that the shared information is unaltered, and no one is lying about a transaction.
- So, here too we can construct a Merkle tree of all transaction together, and if someone is denying one particular transaction, then the Merkle root will change, and by looking into the Merkle root, we would be able to validate whether a set of transactions have been transmitted from one node to another node in an unaltered way.
- **Blockchain 2.0 :** Because of the revolution in bitcoin, many mainstream companies began to use the central blockchain idea and build alternative systems for use in industry, manufacturing, supply chain, finance, governance, IoT, etc. This movement of academia, industry, and startups is termed Blockchain 2.0.

1.3 BLOCKCHAIN SOLUTION

GQ. List and explain various actors in a blockchain solution.

Who are the actors in a blockchain solution ?



(1A19) **Fig. 1.3.1 : Blockchain solution actors**

1. Blockchain Architect

- Responsible for the architecture and design of the blockchain solution.
- A blockchain architect will design how a blockchain solution is going to be built. He/she will identify what information needs to be stored, what are some of the transactions and business logic that need to be embedded onto a network, how the network itself should to be created, etc.

2. Blockchain developer

- A blockchain developer will take what has been architected and he/she will develop the actual code that will run on the blockchain network.
- The developer of applications and smart contracts that interact with the blockchain and are used by blockchain users.

3. Blockchain network operator

- A blockchain network operator manages and monitors the blockchain network.
- Each business in the network has a blockchain network operator.

4. Traditional processing platform

- There are traditional processing platforms or other systems of record that the blockchain connects to and might send or get information.
- An existing computer system which may be used by the blockchain to augment processing. The system may also need to initiate requests into the blockchain.

5. Traditional data sources

- Traditional data sources and databases are also part of the blockchain solution.
- An existing data system which may provide data to influence the behavior of smart contracts.

6. Membership services

- It is an important component. It defines or provides the identity for users to come and transact on the blockchain. For example, when you open an account in the bank, the bank gives you a username and password for login to access web services.
- The membership service will provide a digital certificate that will allow an individual to transact on the network.
- It manages different types of certificates required to run a permissioned blockchain.

7. Blockchain user

- A blockchain user will perform business transactions on the blockchain. These users could belong to multiple organizations that are participating in the blockchain network.
- He/she is a business user operating in a business network. Blockchain users interact with the blockchain using an application. However, they are unaware of the blockchain.

8. Blockchain regulator

- The regulator could be an optional actor in a blockchain solution. The regulator might have read only access onto the network where he/she might have some oversight into whether the transactions performed are legitimate or not. Whether the transactions are compliant with the policies set by the blockchain regulator.
- A blockchain regulator is responsible for the overall authority in a business network. In particular, they may require broad access to the contents of a ledger.

1.4 COMPONENTS OF BLOCKCHAIN

GQ. State and explain different components of a blockchain.

A variety of blockchain components are available in the market. Some of the major components in a blockchain solution are as follows:

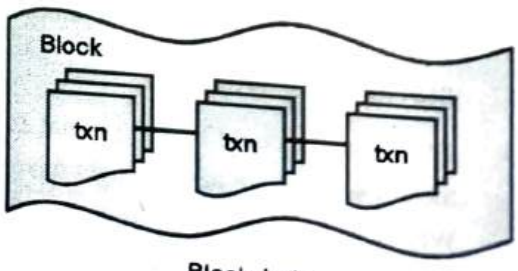
- **Ledger :** "It contains the current world state of the ledger and a blockchain of transaction invocations."

  Every node in the blockchain network will maintain a ledger of all transactions, and the transactions will maintain the state of the data that is being stored on the blockchain network. In other words, a ledger is a channel's chain and current state data which is maintained by each peer on the channel. The ledger is replicated across all nodes in the network.
- **Smart contract :** "It encapsulates business network transactions in code. Transaction invocations result in gets and sets of ledger state." The smart contract is the business logic. An individual can encode his/her own business logic as code/functions and each invocation of this function becomes a transaction on the blockchain. A smart contract is a software running on a ledger to encode assets and the truncation instructions (business logic) for modifying the assets.
- **Peer network :** A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block.
- **Consensus network :** It is a collection of network data and processing peers forming a blockchain network. It is responsible for maintaining a consistently replicated ledger.
- **Membership :** "It manages identity and transaction certificates as well as other aspects of permissioned access." A membership service provides identities for the users to transact on the blockchain.
- **Events :** "It creates notifications of significant operations on the blockchain (e.g., a new block) as well as notifications related to smart contracts. It does not include event distribution." Whenever a transaction happens on a blockchain, an individual can create an event notification. So, blockchain will specify that a particular transaction has been committed and will provide the details of the transaction, which can be used to integrate with existing systems of record. Events can also be used to trigger other transactions that might be internal to an organization.
- **System management :** The blockchain network is a distributed system that is running across multiple organizations so it requires new ways to create, change, and monitor blockchain components.
- **Wallet :** "It securely manages a user's security credentials." Each user has a digital certificate and is going to be performing transactions using the digital certificate. There needs to be a place where a user can securely store that private information. So, a digital certificate contains the private identity of an individual. He/she should not be sharing the information with anybody else, which is securely managed in a wallet.
- **System integration :** It is responsible for integrating blockchain bi-directionally with external systems. It is not a part of the blockchain, but used with it.

☞ A ledger often consists of the following data structures

1. Blockchain

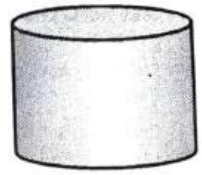
- A linked list of blocks (a hash chain)
- Each block describes a set of transactions (*i.e., the inputs to a smart contract invocation, output, identities, or certificates*)
- Hash chaining leads to certain properties of immutability (i.e., blocks cannot be tampered).



(1A20)**Fig. 1.4.1**

2. World state

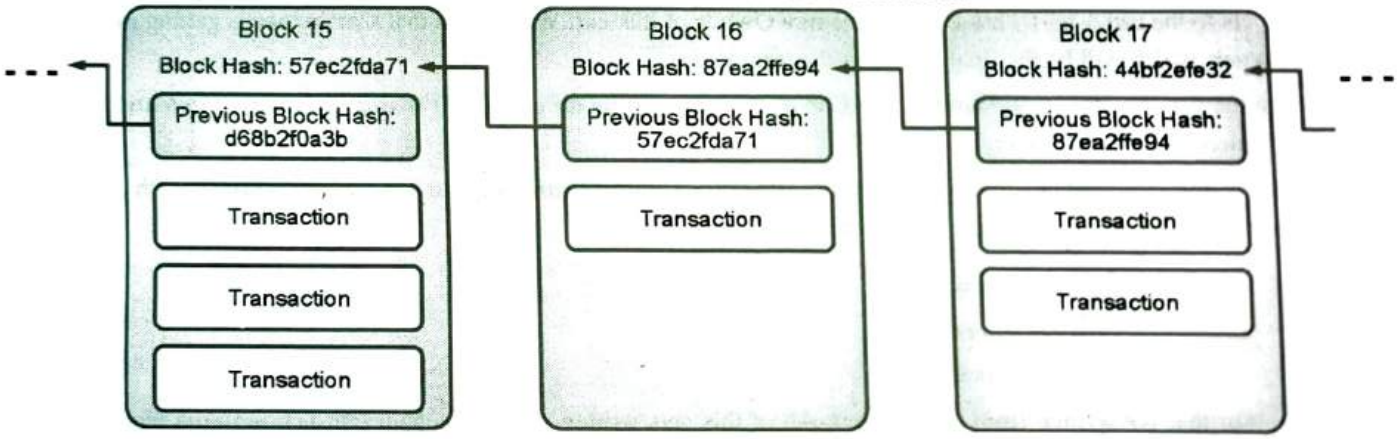
- It is the information that smart contracts can work with.
- It stores the most recent state of smart contracts (*i.e., the output of transactions*)
- In bitcoin, the world state can be thought as the *account balances.*
- It is stored in a traditional database (*e.g., key – value pair*)
- Data elements can be added, modified, deleted, all recorded as transactions on blockchain.



(1A21)**Fig. 1.4.2**

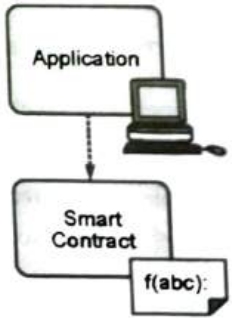
1.4.1 Block in a Blockchain in a Simplified Format

- As can be seen in the figure, every block has a hash, which is being chained together. For instance, Block 15 has a hash value **57ec2fda71**, which is getting stored in Block 16. The same process is observed in Block 16 and Block 17.
- A block is a sequence of transactions, i.e., each block can have multiple transactions in them and each transaction can specify which smart contract got invoked or which specific function got invoked and which user invoked the transaction (i.e., some identity of the user will be there in the transaction followed by other details).
- "*Each block can have zero or more transactions and some additional metadata*". Note that the first block that is getting created in the blockchain is called the genesis block, which has special information about the configuration of the network itself, who the participants are, and other configuration information. The genesis block does not have any user transactions, but other blocks in the blockchain can have one or more transactions.



(1A22)**Fig. 1.4.3 : Simplified view of a block in a blockchain**

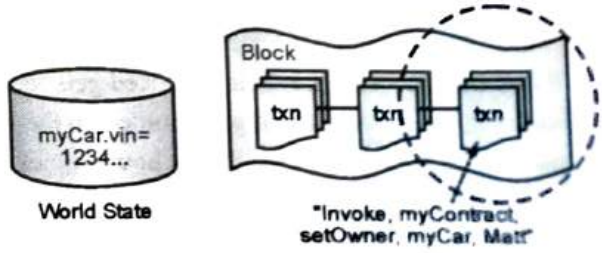
- Blocks achieve immutability because of the result of a hash function of the previous block and also because of the fact that blocks are getting added through consensus in a decentralized manner. So, not a single organization can unilaterally manipulate blockchain.



1.4.2 Ledger Example : A Change of Ownership Transaction

- The Fig. 1.4.4 shows an example of a transaction and how a ledger might work.

  As it can be seen from the Fig.1.4.4 that there is a smart contract that takes it input for instance an owner (let's say a *car*) and the owner of the car is set to *Matt.*



(1A23)**Fig. 1.4.4 : Example of a transaction & working of a ledger**

```
# Transaction input sent from application

invoke (myContract, setOwner, myCar, Matt)

...

# Smart contract implementation

setOwner(Car, newOwner)
{
    set Car.owner = newOwner
}

# World state: New Contents

myCar.vin = 1234
myCar.owner = Matt
myCar.make = Audi

...
```

- From the above code snippet, we have a *setOwner( )* where it sets the owner of a car to a new person, and the new person in this case will be *Matt*.
- The inputs to the *setOwner( )* are **Car** and the **newOwner** of that car. We can see that **Car.owner** is getting modified to **newOwner**, which will be the final output of the transaction.
- Now, in the world state, we see how the **myCar** object has been modified *(refer #World state: New Contents)*, which is getting stored in the world state.
- The transaction that contains this information *(dotted circle in the figure)* says to invoke **myContract** with *setOwner*, *myCar*, and *Matt* as the arguments. Moreover,

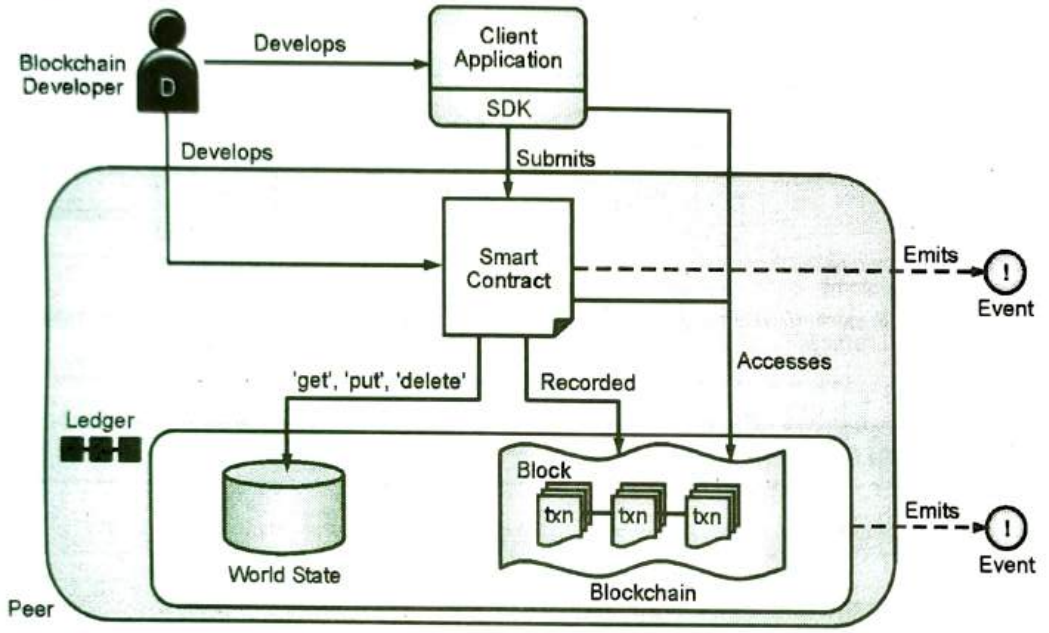
myCar . vin = 1234
myCar . owner = Matt
myCar . make = Audi

is the output that got written from the contract. All of this gets written as a blockchain transaction in the blockchain as one of the blocks.

1.4.3 How External Applications Interact with the Blockchain Ledger ?

- A blockchain developer will develop a client application that will interact with the smart contract. The developer also develops the smart contract. So, this is the business logic that runs inside the blockchain in a decentralized manner.
- In the previous example, **setOwner()** was the part of the smart contract and there is a client application that will invoke functions over a period of time. So, the client application says that Matt is now the new owner of the car and then Matt will be set as the owner. Note that the client application uses an SDK to submit transactions on to the blockchain.
- The client application can also access the blockchain directly to see whether the transactions are actually committed or also look at historic transactions.
- Every transaction that is being invoked gets recorded on the blockchain, and all the elements that are getting modified such *get*, *put*, or *delete* will get recorded on the world state, and whatever is getting modified in the world state will be a part of the transaction. So, this is how a blockchain gets constructed. All this represents a single peer, which means every peer in the network will be performing such a function/task.

- When a client application submit its information to a smart contract, the smart contract gets executed on all the nodes.
- All the nodes simultaneously update their world state and will agree that the output is actually valid and consistent throughout and then the block gets added with the legitimate transaction on to the blockchain, and once a transaction is committed on to the blockchain, it will be called as the final transaction.
- There is an event that gets emitted, as shown in the figure, which means that the transaction has been simultaneously committed on all the nodes in the blockchain. The generated events can be used to perform additional processing within an organization.

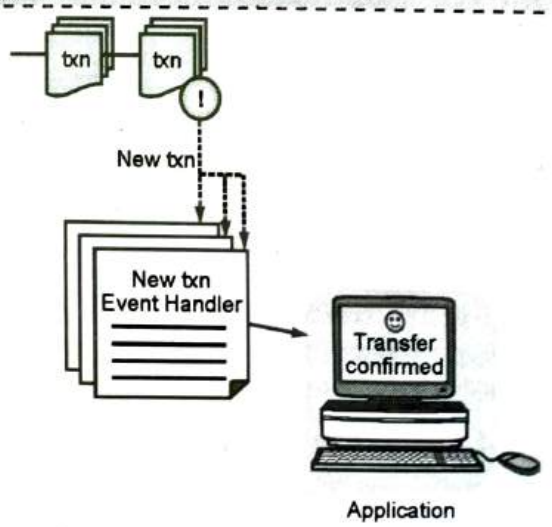


(1A24)**Fig. 1.4.5 : Interaction of an external application with the blockchain ledger**

1.4.4 Blockchain Events

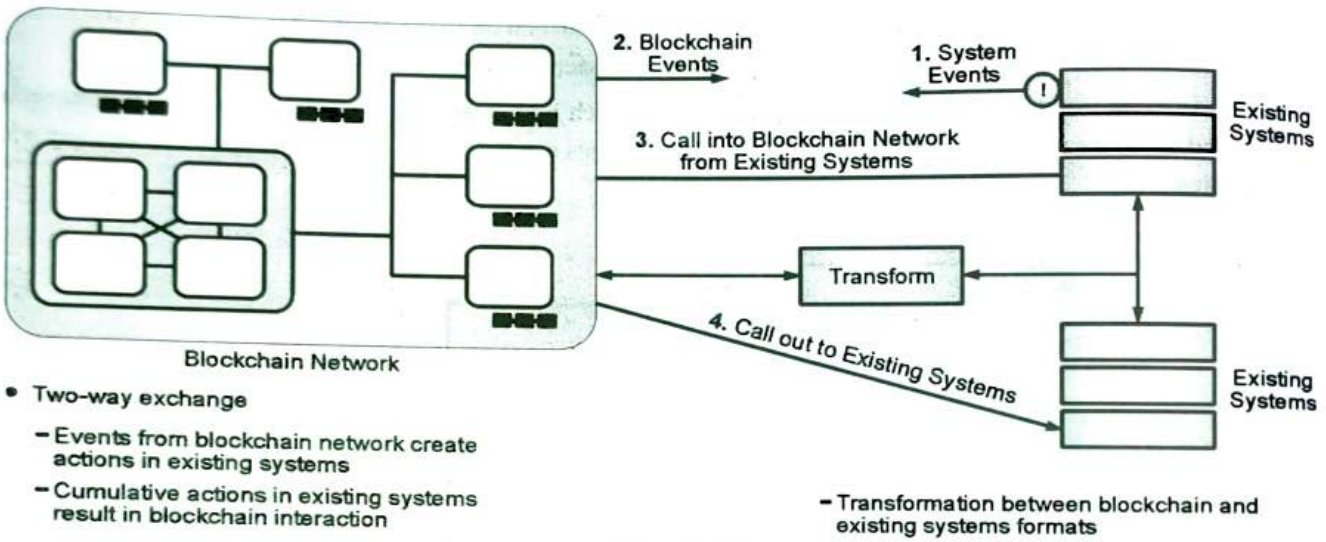
GQ. What are blockchain events ?

- In computing, an event is an occurrence that can trigger handlers (e.g., disk full, fail transfer completed, mouse clicked, message received, etc.).
- Events are important in asynchronous processing systems like blockchain.
- A blockchain can emit events that are useful to application programmers (e.g., transaction has been validated or rejected, block has been added, etc.)
- Events from external systems might also trigger a blockchain activity (e.g., exchange rate has gone down below a threshold value, temperature has risen, time period has elapsed, etc.).



(1A25)**Fig. 1.4.6 : Blockchain events**

Integrating with existing systems



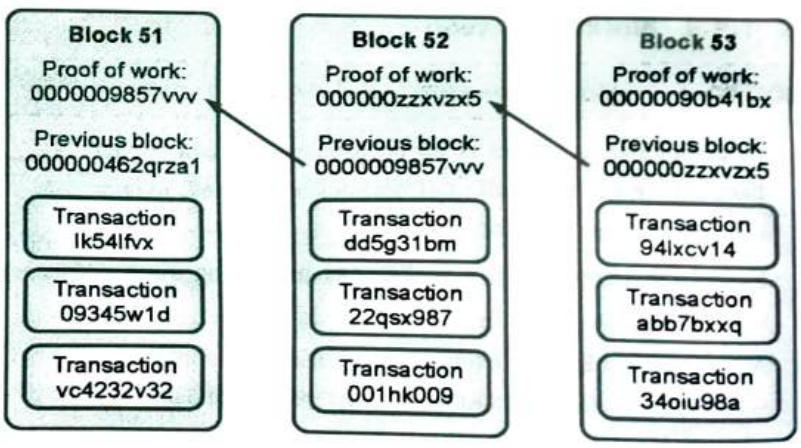
(1A26) **Fig. 1.4.7**

1.5 BLOCK IN BLOCKCHAIN

GQ. What are the various components that are present inside a block that help in forming a blockchain ?

What is there inside the block of a blockchain ?

- To put some data in the blockchain we want to secure the data and how a block is securing the data by utilizing the concept of cryptography.
- We shall take the example of bitcoin in this case to explain what all things are present inside a block and how individual blocks are getting connected.
- Block contain digitally signed and encrypted transactions that are already verified by the peers.
- Inside a block, there could be multiple transactions, which are verified by the peers. Note that these transactions are there in an encrypted format or they are digitally signed. This ensures that the participants can only view the information on the ledger that they are authorized to see.



(1A27)**Fig. 1.5.1 : Components within the block of a blockchain**

1.5.1 Structure of a Block

- A block is a container data structure that contains a series of transactions. In case of a bitcoin, a block may contain more than 500 transactions on an average. The average size of a block is approximately 1 MB as proposed by Satoshi Nakamoto in 2010. Nowadays, the size of a block may grow up to 8 MB or sometimes higher. Also, larger blocks can help in processing large number of transactions in one go. In the context of bitcoin, we shall see the structure of a bock.

- The two major components are : (i) Block header (ii) List of transactions

Block #500312

Summary	
Number Of Transactions	2580
Output Total	10,857.62500453 BTC
Estimated Transaction Volume	2,331.80756289 BTC
Transaction Fees	7.19384324 BTC
Height	500312 (Main Chain)
Timestamp	2017-12-20 20:02:40
Received Time	2017-12-20 20:02:40
Relayed By	BTC.TOP
Difficulty	1,873,105,475,221.61
Bits	402691653
Size	1093.292 kB
Weight	3992.963 kWU
Version	0x20000000
Nonce	900685155
Block Reward	12.5 BTC

Hashes	
Hash	0000000000000000000301fcfeb141088a93b77dc0d52571a1185b42526ae2fb
Previous Block	0000000000000000004b1ef0105dc1275b3adfd067aed63a43324929bed64fd7
Next Block(s)	0000000000000000000282ac9977c3d103a3c6bd873b1f7744e8d42b8329baa6
Marked Root	a89769d0487a29c73057e12d89afafa0c01e02782cba6c89b7019e5129d475cc

Transactions

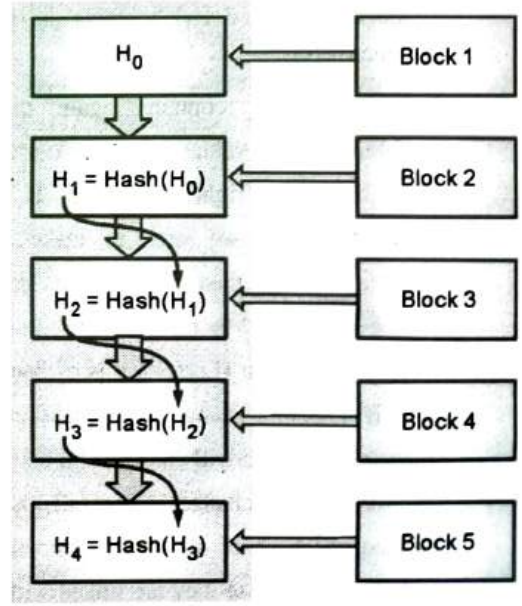
Nlkhja23hjhjjkajdfhjkjhlkjdfljhuyebdueijkuskj7 2017-12-20 20:02:40

No inputs (Newly Generated Coins) → Nlkhja23hjhjjkajdfhjkjhlkjdfljhuyebdueijkuskj7
Unable to decide output address
19.08384324 BTC
0 BTC
19.08384324 BTC

(1A28)**Fig. 1.5.2 : Example of a block header with a list of transactions**

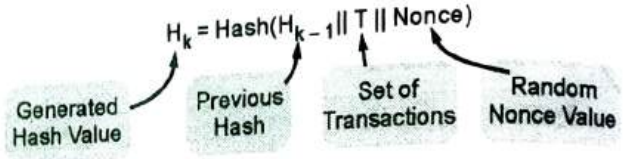
☞ (i) Block header

- In the block header, in blockchain, there exists a sequence of blocks which are connected by the hash of the previous block.
- As we can see from the Fig. 1.5.3, H_1 is connected with the hash of H_0, H_2 is connected with the hash of H_1, H_3 is connected with the hash of H_2, and H_4 is connected with the hash of H_3. This is how hash functions construct a chain kind of a structure. Inside a block header, there exists
- **Previous block hash :** This is utilized to construct the current block hash. Every block inherits from the previous block. Herein, we use previous block's hash to create the new block's hash, thereby making blockchain tamperproof. For instance, if we wish to make a change in Block 2, then H_1's value would change, and as a result, the hash values in H_2, H_3, and H_4 would also change. So, in a distributed network, some person is trying to tamper the block, then he/she has a make changes in all blocks which are there after the tampered block. We want to make this problem as complicated such that by the time some person will tamper certain blocks, new blocks will get added and he/she will never be able to reach up to the last block (*i.e., changing the hash value for the last block*). This how the blocks are made tamperproof.



(1A29) **Fig. 1.5.3**

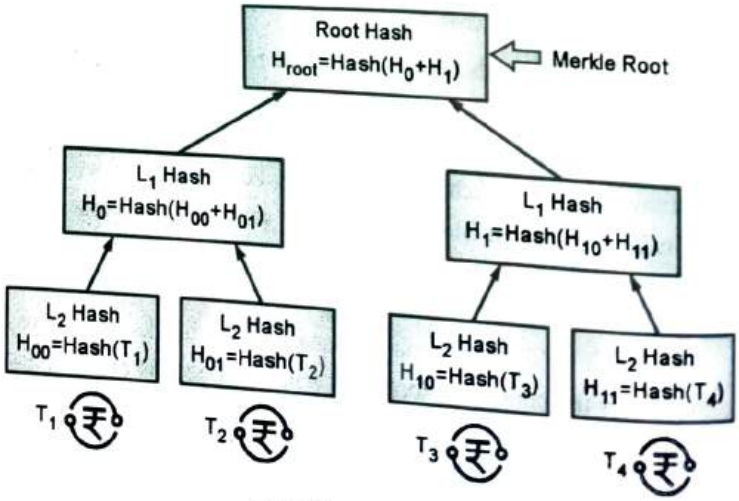
- Mining statistics used to construct the block : Mining is the mechanism to generate the hash. The mechanism needs to be complicated enough to make the blockchain tamper proof. In case of bitcoin mining, the hash function looks like



(1A30) **Fig. 1.5.4**

- The task of a miner is to determine the nonce value such that they can ensure certain difficulty on the generated hash value. For example, the complexity in bitcoin is such that whenever we are generating this hash value, we need to find out the nonce such that whatever be the value of H_k, it will have 20 zeros in its prefix or 1st 20 bits will be zero.
- By the property of the hash function, if H_k is known, then we will not be able to find out the nonce (message), but if the nonce (message) is known, then only it is possible to find out H_k. So, the miners have to change the value of nonce so that the objective is met.
- The objective is that the generated hash value will have some certain number of zeros in the beginning. The complexity of the mining algorithm depends on the number of zeros that are present at the beginning of the generated hash value. The complexity increases when more time is required to calculate the nonce value.

  The blockchain header contains the following parameters *(i.e., mining statistics)*:
  - **Timestamp :** It contains the timestamp to identify when mining has been performed.
  - **Nonce :** It provides the corresponding hash value.
  - **Difficulty :** It determines how difficult it is to obtain the nonce to meet the criteria of the complexity of having certain number of zeros at the prefix of a hash value.
- **Merkle tree root :** This stores the information or a hash value of all transactions. All transactions are organized in a Merkle tree structure. Note that the root of the Merkle tree is a verification of all the transactions.
  - In a Merkle tree, at the leaf node, there exist a hash of the transactions (i.e., T_1, T_2, T_3, and T_4). Every intermediate node (i.e., H_0 and H_1) contains the hash of the combined hash values (i.e., H_{00} and H_{01} and H_{10} and H_{11}, respectively).
  - Also, the root contains again the combined hash values of its left tree (i.e., H_0) and right tree (i.e., H_1). Interestingly, if we wish to make a change in T_3, then H_{10} would get changed followed by changes in H_1 and finally changes in H_{root}. So, it is evident that if someone changes a transaction, then the root hash will change, and once the root hash changes, then all the subsequent hash of all the blocks will get change because they are linked with each other.



(1A31) **Fig. 1.5.5 : Merkle tree root**

How does a block header look?

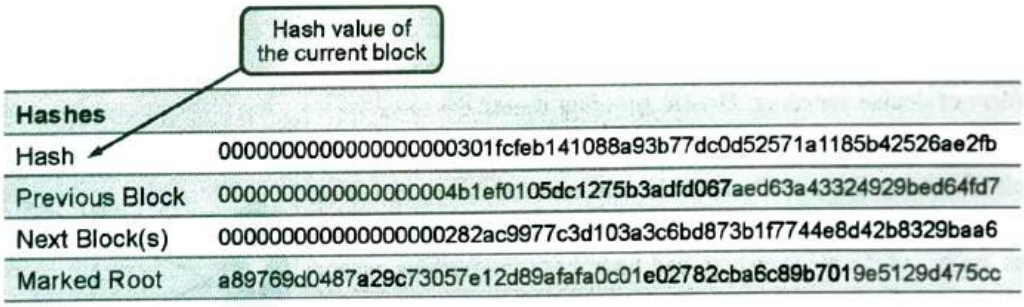
GQ. With a suitable diagram, explain the structure of a block header with a list of transactions.

Summary	
Number Of Transactions	2580
Output Total	10,857.62500453 BTC
Estimated Transaction Volume	2,331.80756289 BTC
Transaction Fees	7.19384324 BTC
Height	500312 (Main Chain)
Timestamp	2017-12-20 20:02:40
Received Time	2017-12-20 20:02:40
Relayed By	BTC.TOP
Difficulty	1,873,105,475,221.61
Bits	402691653
Size	1093.292 kB
Weight	3992.963 kWU
Version	0x20000000
Nonce	900685155
Block Reward	12.5 BTC

(1A32) **Fig. 1.5.6**

Hashes in a block header

- A block header also contains certain number of hashes, ash shown below:



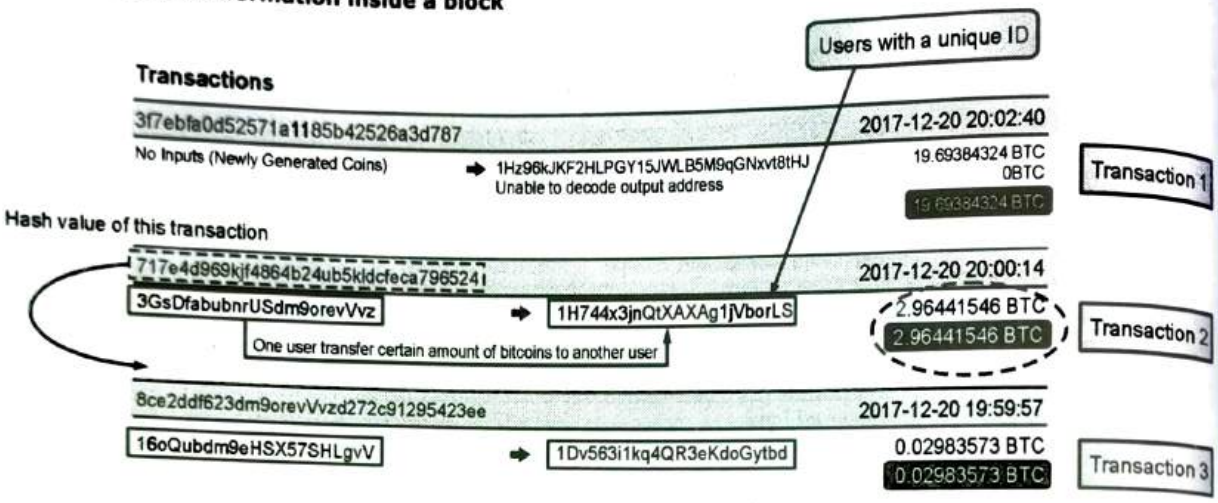
(1A33) **Fig. 1.5.7**

- If any change is made in one of the transaction, then the Merkle root gets changed. If the Merkle root changes, then the corresponding hash value gets changed. If the hash value of the current block changes, then the hash value of the next block changes.
- Ultimately, everything gets changed and so the attacker will have to change the entire blockchain. In addition, the hash algorithm that is used to compute the hash is double SHA256.

Transactions in a block

1. Transactions are organized as a Merkle tree. The Merkle root is used to construct the block hash.
2. If you change a transaction, then you need to change all the subsequent block hash.
3. ***Inference*** : The difficulty of the mining algorithm determines the toughness of tampering with a block in a blockchain.

Transaction information inside a block



(1A34)**Fig. 1.5.8 : Transaction information inside a block**

All these hash values are added in a Merkle tree and a Merkle root is added as a part of a block header.

Summary

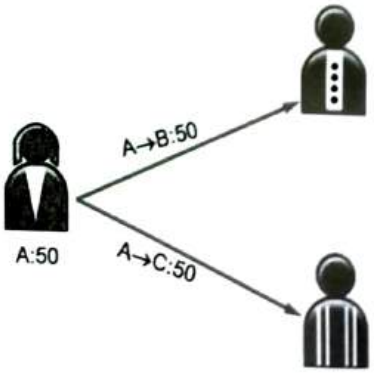
- The block contains two parts, i.e., the header and data (transactions).
- The header of a block connects the transactions, i.e., any change in any transaction will result in a change at the block header. The headers of subsequent blocks are connected in a chain, i.e., the entire blockchain needs to be updated if you want to make any change anywhere.

1.5.2 Double Spending

GQ. Explain the concept of double spending with a suitable example?

Whenever Person A is creating a transaction in a digital currency, there could be a problem of double spending. *Double spending means the same cryptocurrency is used for more than one transaction.*

- For example, Person A has a total of 50 bitcoins, and the same amount of bitcoin she has transferred to Person B and Person C. Such type of transactions are called double spending, and both these transactions cannot be valid simultaneously.
- In a centralized system like a banking agency, it is easy to validate such kind of double spending. So, when a person submits a transaction to a bank, the bank can validate whether that person is performing a double spending and take the necessary actions.



(1A35)**Fig. 1.5.9 : Double Spending**

- However, in a decentralized network, how the problem of double spending can be avoided so that an individual will not be able to make two transactions with the same cryptocurrency.

Handle double spending using blockchain

- In order to prevent double spending, we use blockchain so that the details about the transactions that are sent are forwarded to all the users in a network or as many as other computers in the network as possible. We use blockchain, which is a constantly growing chain of blocks, to contain a record of all such truncations, and this blockchain is maintained by all the peers in a cryptocurrency (i.e., bitcoin) network. So, everyone has the copy of the blockchain.

- To be accepted in a chain, transaction blocks must be validated and must include proof of work (PoW) - a computationally difficult hash generated by the mining procedure. Blockchain ensures that if any of the block is modified, then all following blocks will have to be recomputed. When multiple valid continuation to a chain of blocks appear, only the longest branch is accepted and it is then extended further (i.e., longest chain). Once a transaction is committed in the blockchain, everyone in the network can validate all transactions by using a sender's public address. This is how validation prevents double spending, especially in bitcoin.

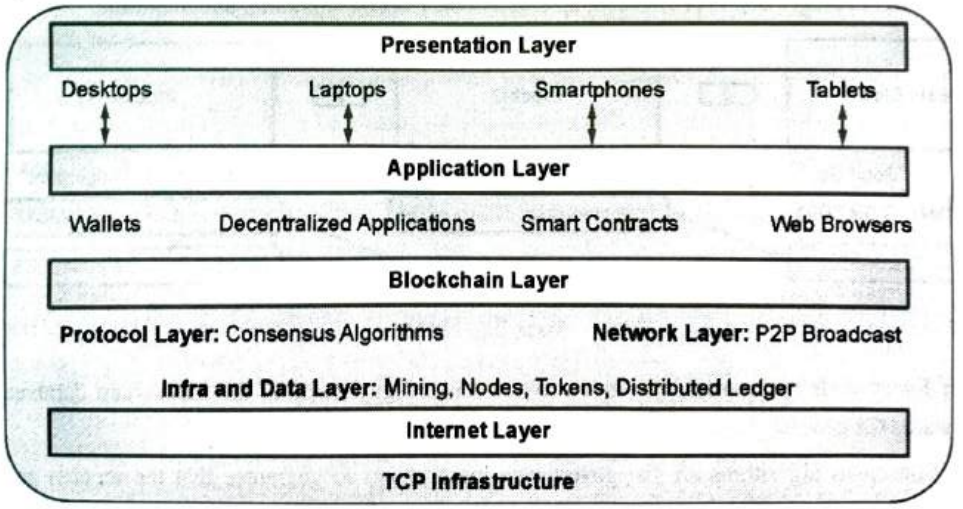
1.6 THE TECHNOLOGY

1.6.1 Blockchain Layers

GQ. Write a short note on: Blockchain layers.

A simple structure of the blockchain ecosystem is shown below. (Refer Fig. 1.6.1)

- **Presentation layer :** This layer provides access to the entire network and blockchain applications. User experience (UX) and user interface (UI) of a blockchain application begins from the presentation layer. As we all know that in today's world customer satisfaction mainly relies on ease of use and suppleness. Similarly, for the adoption of blockchain, good UX and UI designs are equally important. UX design not only enhances customer gratification and trustworthiness but also improves product usability (i.e., faster speed) and interaction between a user and the rest of the blockchain ecosystem. UI design offers blockchain visuals. It enhances the experience of a user in terms of aesthetics leading to an instinctual interface.
- **Application layer :** This layer combines business logic with user interactions. It consists of decentralized applications running on a P2P network. Decentralized applications are the ones that set communication between the presentation and blockchain layers. Decentralized applications are similar to a web application except that application programming interfaces are used to connect a website to a database. Decentralized applications connect to a blockchain via smart contracts.
- **Blockchain layer :** This layer consists of consensus algorithms, the medium, and interface for a P2P network that decides how data is placed in several packets and then transmitted between peers. It handles the mining layer, protocols that decide consensus methods and participation, nodes that execute protocols, and the distributed ledger.
- **Internet layer :** Blockchain works over the internet. This layer connects all networks together (i.e., smartphones, IoT devices, etc.).



(1A36)**Fig. 1.6.1 : Blockchain layers**

1.6.2 Advantages of Blockchain

GQ. Enlist various advantages and disadvantages of blockchain.

- Blockchain technology works on the principle of ledger data distributed to all nodes that are non-hierarchical with no single central control.Removes any single point of failure by replicating the ledger at every node in the network.
- Better communication between nodes promotes transparency and faster consensus and data synchronization.Allows for multiple entities who do not trust each other to transact directly with one another.
  - Ensures valid and accurate data.
  - Removal of middleman reduces overall transaction cost.
  - Contains a verifiable record of all transactions made that are auditable.
  - Risk of double spending is mitigated because of consensus algorithms.

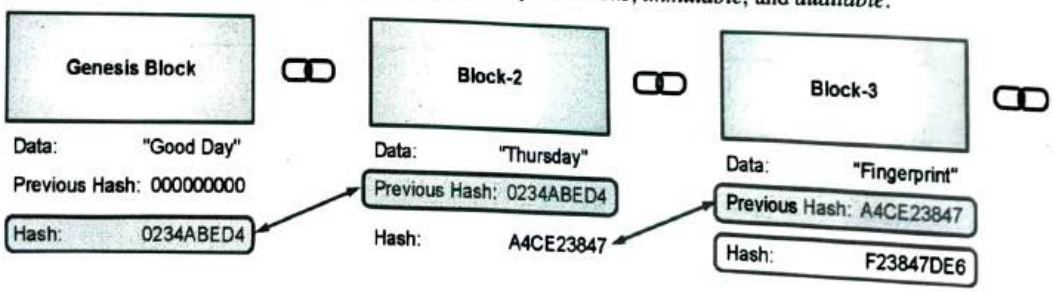
1.6.3 Disadvantages of Blockchain

- In certain cases, the traditional database may be more suitable and perform a task faster and cheaper. If a fully functional database and network are already in place, then the benefits of introducing blockchain may not produce the required returns on investment.
- Nodes with high computing power can take control of the network, thereby impacting decentralization. Data integrity is obtained at the expense of time. Every node needs to run the blockchain to verify transactions and maintain consensus.
- Enormous computing power is needed by miners leading to substantial energy consumption and wastage. Nodes may select transactions with higher rewards.
- Not every node has the ability to maintain and run a complete copy of the blockchain, thereby affecting consensus and immutability. In smaller blockchains, there is a risk of 51% attack (i.e., if a group of malicious nodes can get 51% of the mining hash rate, then the group can manipulate transactions).

1.7 BLOCKCHAIN TYPES AND CONSENSUS MECHANISM

1.7.1 Introduction

Blockchain is a digital ledger that is *distributed*, *upheld by consensus*, *immutable*, and *auditable*.



(1A37) **Fig. 1.7.1**

- **Distributed :** Every node that participates in a network has a digital copy of the blockchain database where all can contribute towards the processing of blockchain.
- **Consensus :** Consensus algorithms are the governance mechanisms to guarantee that the records are legitimate and tamper proof.

- **Immutable** : Once consensus is reached based on the validity of a transaction and recorded on a blockchain, modification or deletion is impossible. Any changes are recorded in a new block.
- **Auditable** : Immutable record tracking with timestamp permits provenance (ability to track a transaction) of an asset at each and every step.

With blockchain, the dependency of trust on third-parties is made redundant by the direct peer-to-peer handshake within the network of nodes.

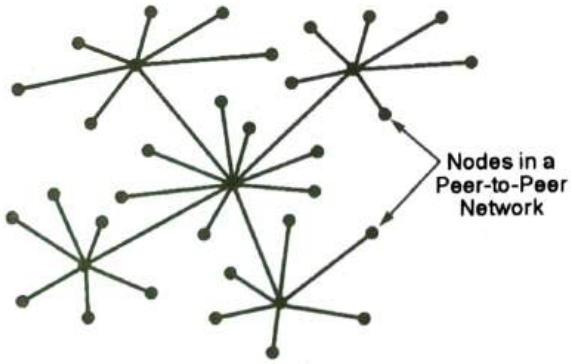
Characteristics enabling the trustless nature of the blockchain are as follows :

- **Decentralization :** Data is not only digital but also decentralized, which means that it can be shared across a network of computers without the need for a central authority for making decisions.
- **Transparency :** Data is transparent and open to everyone (i.e., anybody can view the transactions).
- **Privacy :** The identity of a user is kept private via cryptography.
- **Security:** Transactions are cryptographically secure by means of hash algorithms.

1.7.2 Decentralization and Distribution

One of the main characteristics of blockchain is decentralization (i.e., transactions are not under the control of any single party).

- A decentralized system does not rely on a single authority. Blockchain technology uses a decentralized peer-to-peer network architecture where anybody can be a node. Each and every node is equal in the hierarchy with equal access to maintain the database. In a decentralized environment, anyone can access or write into a ledger.
- Decentralization is in-built in distributed ledger technology (DLT), which becomes the core of blockchain technology.

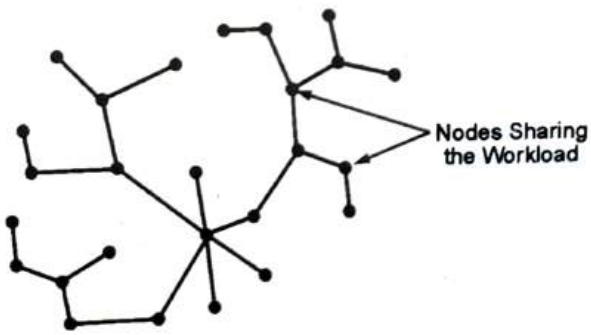


(1A38)**Fig. 1.7.2 : Decentralized System**

Distributed Ledger Technology

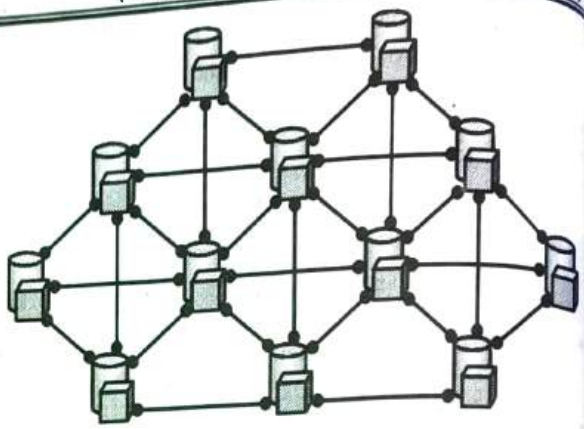
GQ. What is a distributed ledger ?

- A distributed ledger is a decentralized ledger of all the information that is recorded on the blockchain by consensus. In other words, a distributed ledger technology is defined as *a decentralized database that can securely record and share financial, physical, or electronic access across a geographical network through transparent updates of information.*
- A distributed ledger can be considered to be a database similar to an accounting ledger where financial transactions are recorded, except that it is not recorded to financial data. Herein, a transaction refers to any digital data such as text, images, or audio files.
- The database can be distributed to all locations and can be accessed by every single member present in the blockchain network, thereby ensuring incorruptibility as malicious changes cannot be made when each node has simultaneous access to all records.
- Note that a decentralized system does not be a completely distributed one. The below figure represents a decentralized network where processing work is shared with sub-nodes.



(1A39)**Fig. 1.7.3**

- Such decentralization is different from the one that is observed in blockchain. In DLT of blockchain, a database copy is available on every user's computer. This database is independent of a central authority and any changes to the ledger have to be agreed upon by all nodes.
- Such a mechanism is called as the consensus mechanism. Once consensus is attained, the database is updated to all nodes in the network. At any given point in time, information is synchronized across all nodes; so, it is referred to as distributed consensus.



(1A40)Fig. 1.7.4 : Distributed Ledger Technology

Benefits of distributed ledger technology

GQ. State several benefits of distributed ledger technology.

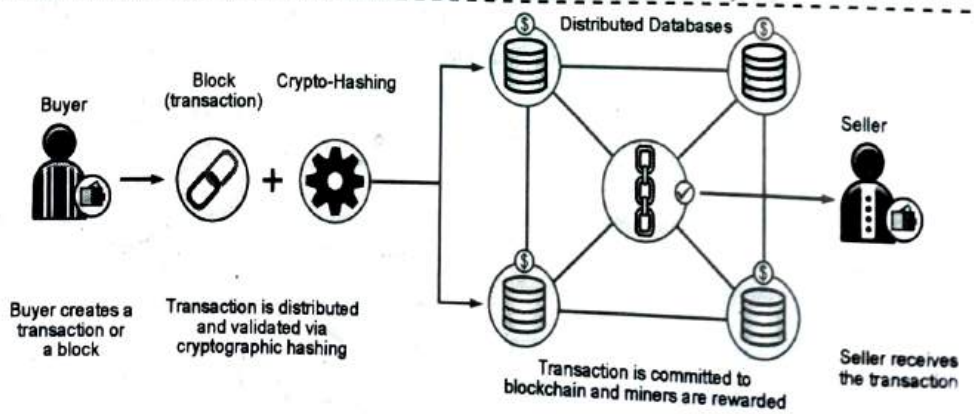
- **Transparent and secure :** As data is shared and visible to all nodes, it is not easy to make any unauthorized changes. Every node that participates in the network maintains a copy of the ledger, thereby preventing a single point of failure. Any entry has to be agreed upon by all parties making a distributed ledger secure and tamper proof.
- **Decentralized :** Unlike a centralized system, every node has control over his/her data. Decentralization offers users the power to take appropriate decisions as to what will be in his/her data record. Consensus methods help in securely adding appropriate data in a distributed ledger, thereby offering innate trust in a system.
- **Efficient :** With trustless and distributed nature of blockchain's DLT, the effort of capturing, validating, and synchronizing individual sets of information by mediators can be eliminated, thereby reducing the chances of human error and improvement of operational efficiency.
- **Cost savings :** Disintermediated systems can save on additional/bottom-line costs while realizing near-time transactions and efficiencies.

1.7.3 Types of Blockchain

GQ. Explain different types of blockchain.

GQ. What is a public blockchain? Enlist few examples of a public blockchain and state its advantages and disadvantages.

GQ. What is a private blockchain? Enlist few examples of a private blockchain and state its advantages and disadvantages.



(1A41)Fig. 1.7.5 : Public blockchain

At a glance, there are three major types of blockchain technologies.

1. **Public blockchain :** It is a permission-less distributed ledger technology where any individual can join not only join the blockchain network but also perform transactions. It is a non-restrictive version where each and every peer has a copy of the ledger (i.e., anybody can access such type of blockchain if he/she has an internet connection). One of the 1st public blockchains was bitcoin that permitted anybody connected to the internet to carry out transactions in a decentralized manner. Transaction verification takes place through consensus (i.e., proof-of-work (PoW) and proof-of-stake (PoS)). Note that the participating nodes need to perform heavy lifting such as validating transactions to make public blockchain work. If a public blockchain does not have the required peers in order to participate and solve transactions, then the blockchain becomes non-functional.

   Examples : Bitcoin and Ethereum

☞ **Advantages**

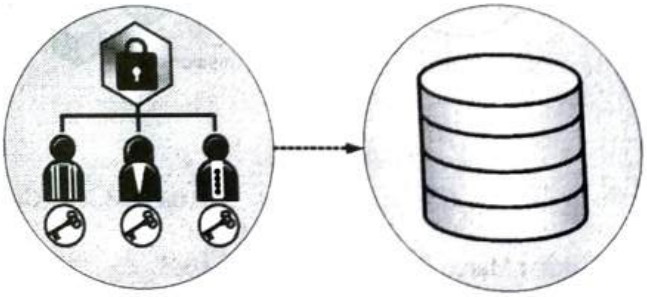
1. Anybody can join public blockchain.
2. Everyone feels incentivized to work towards the betterment of a public network.
3. It brings transparency to the entire network as the available data is accessible for verification purposes.
4. It requires no intermediaries to work.

☞ **Disadvantages**

1. Transaction speed of public blockchain is less (i.e., a transaction could be completed within a few minutes or might take several hours).
2. It suffers from scalability (i.e., the more the number of nodes that join the network, the clumsier and slow it becomes).
3. It uses the choice of consensus method (e.g., bitcoin uses PoW, which consumes a lot of energy).

2. **Private blockchain :** It is type of blockchain that works in a restrictive environment (i.e., in a closed network). It is also a permissioned blockchain that is under the control of an entity. It is suitable for organizations who only want selected participants to access a blockchain network.

   Organizations can set different parameters, such as accessibility and authorization, to the network. Private blockchain is similar to a public blockchain, but the only difference is that it allows selected participants to access the network, thereby offering such participants transparency, trust, and security. A private blockchain is usually centralized (i.e., only one authority manages the entire network).



(1A42)**Fig. 1.7.6 : : Private blockchain**

   Examples : Hyperledger Fabric, Hyperledger Sawtooth, and Corda

☞ **Advantages**

1. A private blockchain is relatively fast as compared to a public blockchain. This is because there are a few participants in the network, which means that it takes a shorter amount of time for the network to reach consensus resulting in faster transactions.
2. They are much more scalable. This is possible because only a few nodes are authorized to validate transactions. For a private blockchain, the network size does not matter. Here, centralization aspect for making a decision is of utmost importance.

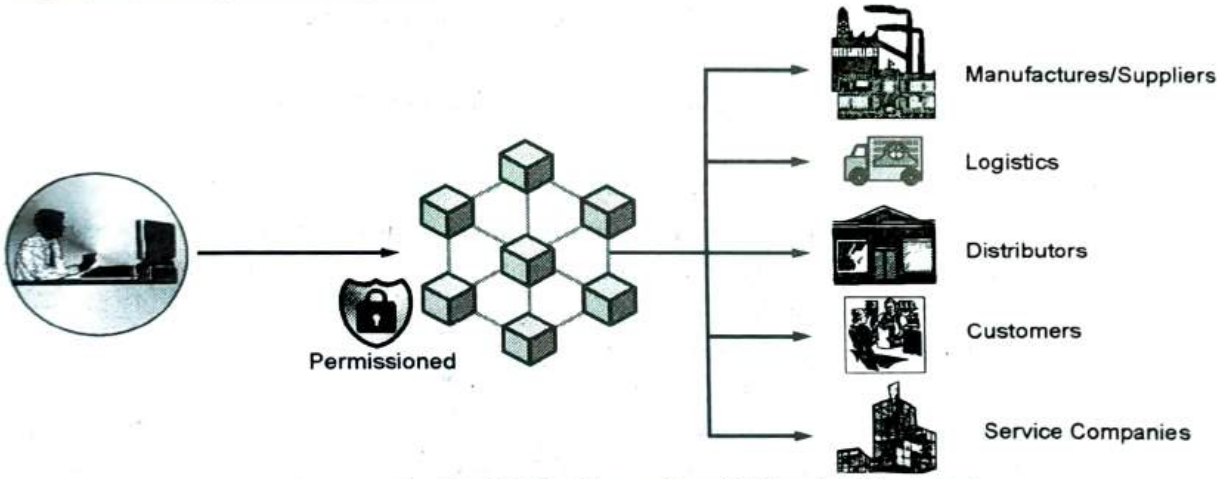
Disadvantages

1. They are not decentralized in a true sense.
2. In a private blockchain, achieving trust is difficult because centralized nodes make the last call.
3. Security could be an issue in a private blockchain. This could happen when a certain number of nodes compromise the consensus method that is utilized by the private network.

3. Consortium/Federated blockchain

GQ. What is a consortium/federated blockchain? Enlist few examples of a consortium/federated blockchain and state its advantages and disadvantages.

- It is a type of blockchain that is preferred by organizations who require both public as well as private blockchain. In a consortium/federated blockchain, certain aspects of an organizations are made public, while others remain private. Note that the consensus method in a consortium/federated blockchain is managed by preset nodes. Even though such type of blockchain is not open to mass people, it still holds a decentralized nature.
- A consortium/federated blockchain is managed by more than one organizations. As a result, there is no one single force of a centralized outcome. In consortium/federated blockchain, there exists a validator node that not only validates transactions but also initiates or receives transactions.
- The member nodes on the other hand can only receive or initiate transactions. Overall, a consortium/federated blockchain offers all the features of a private blockchain, including transparency, privacy, and efficiency, without a single party having the entire power.



(1A43)**Fig. 1.7.7 : Consortium/Federated blockchain**

Examples : Marco Polo and IBM Food Trust

Advantages

1. Better customizability and control over resources.
2. Better scalability and offers access controls.
3. More secure and efficient.

Disadvantages

1. Even though a consortium/federated blockchain is secure, the entire network can be compromised due to a member's integrity.
2. Less transparency
3. Censorship and enforced regulations might adversely affect network functionality.
4. Less anonymous compared to public and private blockchain.

Comparison between different types of blockchain

GQ. Differentiate between different types of blockchain.

Sr. No.	Parameters	Public Blockchain	Private Blockchain	Consortium/Federated Blockchain
1.	Definition	It is open to everyone and anyone can participate.	It is controlled by owners and access is limited to certain users.	It is a combination of public and private blockchain, which means some processes are kept private and others public.
2.	Cost of transactions	Costly	Not so costly	Not so costly
3.	KYC needed	No	Yes	Yes
4.	Incentive	Public blockchain incentivizes participants to grow the network.	Private blockchain is limited; hence, it has no similar incentives as that of a public blockchain.	Consortium/Federated blockchain can opt to incentivize users if they wish.
5.	Transparency	It is entirely transparent	It is transparent to only those users who are granted access	Transparency depends on how owners set rules
6.	Transaction speed	Slow	Faster than public blockchain	Fast

1.7.4 Consensus Protocol

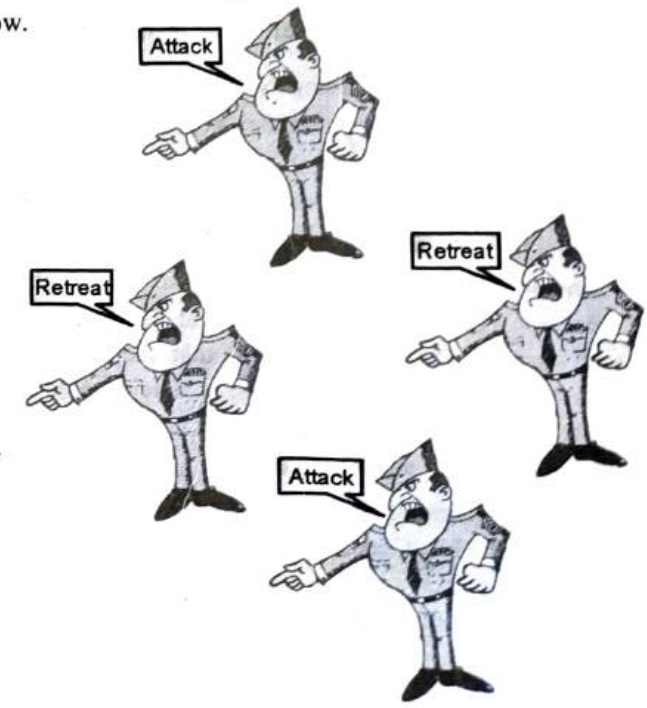
GQ. What is a consensus protocol ?

1.7.4.1 Consensus

- The concept of consensus is an interesting topic in a decentralized or a distributed network. Consensus means a procedure to reach a common agreement in a distributed or decentralized multi-agent platform.

A typical example of a consensus mechanism is shown below.

- In an army there are four generals, and they are taking some kind of important decisions, and when the generals are taking decisions in a decentralized or a distributed way *(i.e., they have their own policies to take the decision)*, they can either make an *attack* or can either *retreat* from an attack.
- In a consensus algorithm, all these generals individually express their opinion, and from their individual opinion by applying some kind of a choice function, which can be the majority of the decision in this case, the system/environment takes an appropriate decision.
- In this case, there are three generals that are making a choice to perform an attack. So, with a majority principle, the system can come to a consensus that they should make an attack collectively.
- Such type of a consensus technique is important for a message passing environment in a distributed system.



(1A44)**Fig. 1.7.8 : Consensus Mechanism**

Why consensus ?

- In a traditional or conventional distributed system, consensus is applied to ensure reliability and fault tolerance. This means that in a decentralized environment when there are multiple individual parties and they can take their own decision, then there is a possibility that some nodes, parities, or individuals are working maliciously or working as a faulty individual.
- In such cases, there is a need to come to a common decision/viewpoint. So, having a common view point in an environment where people can behave maliciously or people can crash or work as a faulty way, it is a difficult thing.
- So, under suck type of a distributed environment, the objective is to ensure reliability, i.e., to ensure correct operations in the presence of faulty individuals.

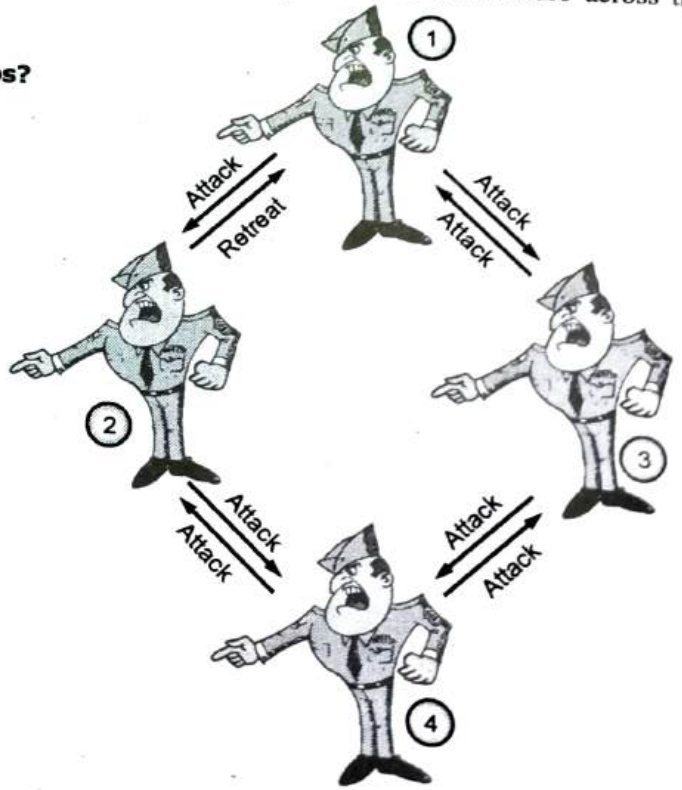
The examples for consensus in a distributed system when we consider a distributed database are as follows :

1. **Commit a transaction in a database :** Transactions are taking place from multiple points of sale, i.e., from multiple ATMs, multiple banking sectors, the transactions are coming in and during that time consensus is a crucial aspect. For instance, if an individual wants to perform a transaction from one bank branch to another account in another bank branch, then during that time, it is essential to have a common consensus by all the bank branches to decide whether it is a valid transaction or not and if the transaction is valid, then only the transaction gets committed.
2. **State machine replication :** It is an important aspect of any distributed protocol. For instance, if an individual wants to run some kind of a distributed protocol over a network, every individual node runs the current version of the protocol and they store the store the state of the protocol in different state machines; so, the entire execution part of the protocol can be represented as a state machine. Now, such a state machine needs to be replicated into multiple nodes so that every individual node can come to a common point or common output of that protocol.
3. **Distributed Clock synchronization :** Assume that you have multiple clocks in your network and every individual node tries to find out that which is the most updated/current clock and so these nodes need to make consensus amongst themselves and come to a single clock and by applying such type of a clock synchronous architecture across the network, the nodes can perform further operations.

Why consensus can be difficult in certain scenarios?

Consider a message passing system, as mentioned earlier. We have multiple generals, and they are utilizing a message passing environment to communicate their viewpoint to others. Assume that every general is making a telephone call to other general and then communicating their viewpoint to others.

- General 1 passes the information to its neighboring generals (i.e., General 2 and General 3) via a telephone call and specify his view point that they should now attack. Similarly, General 4 makes a telephonic call to General 2 and General 3 and specifies his decision that they should now perform an attack.
- However, General 2 is a malicious general who makes a malicious call to General 1 and asks for retreat whereas General 2 makes a call to General 4, General 2 says to perform an attack. In a complete distributed or a decentralized environment, the generals mist get confused.



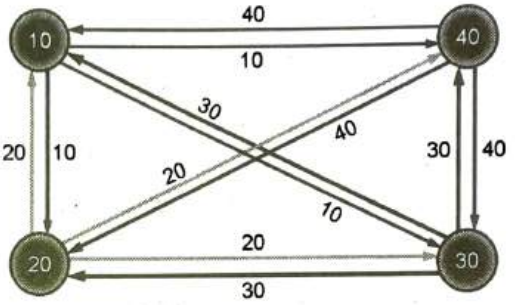
(1A45) Fig. 1.7.9

- For example, if General 2 is informing General 1 to retreat, General 1 will get confused as to what he should do, so, in a complete distributed or a decentralized environment, it would be difficult to achieve consensus over a message passing system when there are malicious nodes or after certain time the nodes begin to behave maliciously, which are called as byzantine nodes or such failures are called byzantine failures.
- In the presence of such type of byzantine failures, the system can behave maliciously or it may be difficult to achieve consensus in a distributed environment.

1.7.4.2 Distributed Consensus

GQ. What is distributed consensus ?

If there is no failure in the system, then it may be easy and trivial to reach in a consensus. A generic algorithm could be like the personal choice is broadcasted to all and then a choice function is applied saying that only that value will be considered which is the maximum of all the received values and in this way consensus can be achieved.

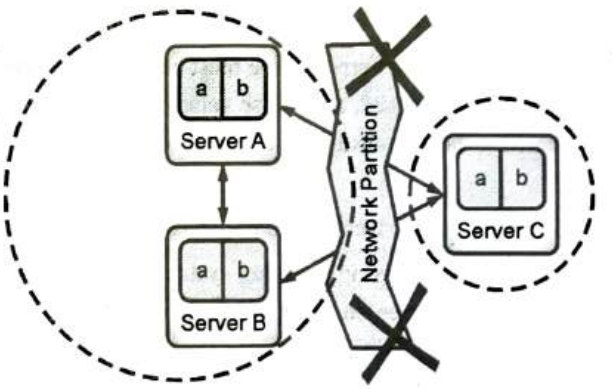


(1A46)**Fig. 1.7.10 : Distributed Consensus**

- There are four nodes. The individual nodes make a choice of 10, 20, 30, or 40, and they inform their individual choices to all other nodes in the network and whenever every node receives all the choices from all the neighbors, they can apply a choice function (*i.e., max. function*) to determine what the maximum value is. So, from the figure, it is evident that every node will reach to the value 40 if they apply the maximum function of all the received values.
- Achieving consensus in such a type of architecture is easy and straightforward. It is possible only under certain scenarios (i.e., ***the system needs to be faultless*** - there should not be any failure in the system so that every individual node can receive the message correctly and ***the system should behave in a synchronous manner*** - all messages will be received within a predefined time interval).
- So, in the above architecture, all the nodes are expected to receive all the messages from its peers in a non-failure environment within a pre-defined period of time, and every node can wait for a specific period of time, and in a synchronous system, it is expected that every node will receive the message and once the node receives the message, they can determine the maximum value from all the received values and select the maximum value for the consensus.
- However, achieving consensus can be non-trivial in case of a distributed environment due to the presence of multiple type of failures.

Typically, in a distributed system, the following types of failures are considered.

- **Crash fault :** A node suddenly crashes or becomes unavailable in the middle of a communication, and so, no message will be received from that specific node. It could be a type of a hardware or software fault due to which a node or a process that is communicating with another node or process fails.
- **Network or partitioned faults :** A network fault occurs (e.g., link failure) and the network gets partitioned. From the below figure, we can see that there are three servers A, B, and C, which are interconnected to each other.



(1A47)**Fig. 1.7.11**

- When the link between Server A and Server C fails as well as the link between Server B and Server C fails, the network gets partitioned into two halves, as shown in dotted circles, and there is no message passing /communication between the partitioned networks as the connecting link has failed.
- Such kind of partition can happen in the network, and so, it is called as a partition fault.
- This type of network failure can hamper reaching consensus. Because of partition fault, nodes in one partition become unavailable to nodes in another partition for communication, and so, general consensus cannot be attained.

Crash and network or partitioned faults take place because of hardware or software failure.

- **Byzantine faults :** It is a kind of fault that is more difficult to handle in a distributed environment. Here, a node starts behaving maliciously, and it is difficult to interpret the nature of the node. Sometimes the node could send a positive vote or sometimes it could send a negative vote. In case of the abovementioned two faults, we can identify the effects of a network fault or a crash fault. Nonetheless, the effect of a byzantine fault is difficult to guess because it completely depends on the malicious nature of the node. Sometimes a node can give a vote against the consensus or sometimes, it could give a vote in favor of the consensus. Therefore, handling byzantine nodes becomes difficult in a typical distributed system.

☞ Properties of a distributed consensus protocol

GQ. State the properties of a distributed consensus protocol.

- **Termination :** Every correct individual decides some value at the end of the consensus protocol. In other words, whosoever is the non-faulty node in the network, the non-faulty node must terminate the protocol and decide on one value, which is the correct value.
- **Validity :** If all the individuals in the node proposes the same value, then all correct individuals decide on that value.
- **Integrity :** Every correct individual decides at most one value, and the decided value must be proposed by some individual. This property ensures that the consensus value should not deviate from the values that are proposed by individual nodes in the network.
- **Agreement :** Every correct individual must agree on the same value. All the individual nodes in the network must agree on the same value. Whenever all nodes agree on the same value after termination, we declare that the system has reached consensus.

☞ Correctness of a distributed consensus protocol

- **Safety :** Correct individuals must not agree on an incorrect value (i.e., nothing bad has happened in the system). This property ensures that a node will never converge to an incorrect value or the correct individuals will never converge to an incorrect value.
- **Liveness :** Every correct value must be accepted eventually (i.e., something good will eventually happen). If some good values are proposed, then those values will be committed eventually, although there could be some time lag or delay in reaching the consensus, but after the consensus protocol terminates, a consensus value is expected to be achieved out of it.

While dealing with consensus, we need to deal with the following types of message passing systems.

- **Synchronous message passing system :** The message must be received within a predefined time interval. So, we have a strong guarantee on the message passing delay and we know in advance what can be the maximum delay of message passing for a particular network. Such type of message passing system offers simplification in designing the protocol, which means that waiting for a specific amount of delay will be worth because it is assured that within that time the message will be received. Designing a distributed system in a synchronous environment is much easy because of the

strong assumption of the message passing delay, and that is why if we wait for a certain amount of delay, it is expected that we will receive all the messages.

- **Asynchronous message passing system :** There is no upper bound on the message transmission delay or the message reception time. Here, there is no time constraint, i.e., a message can be delayed for an arbitrary period of time or the message delivery time can be arbitrarily long. So, in case of an asynchronous message passing system, we cannot wait for a finite duration to receive messages with a guaranteed probability. Designing a distributed system in an asynchronous environment is not that easy because we do not have sufficient amount of delay. Moreover, we also need to deal with the type of fault that may arise because of the asynchronous nature of the system. Achieving consensus in an asynchronous message passing system is not that easy when compared with a synchronous message passing system.
- **FLP85 (Impossibility result) :** In a purely asynchronous distributed system, the consensus problem is impossible (with a deterministic solution) to solve if in the presence of a single crash failure. However, randomized algorithms may exist.

☞ Synchronous consensus

There exists various consensus algorithms that have been explored by the distributed system community.

- Paxos
- Raft
- Byzantine fault tolerant (BFT)

☞ Consensus in an open system

Traditional distributed consensus protocols are based on

- **Message passing :** When individuals are connected over the internet
- **Shared memory :** When a common memory place is available to read and write shared variables that everyone can access.

☞ Limitations

1. Message passing requires a closed environment, i.e., everyone needs to know the identity of others.
2. Shared memory is not suitable for Internet grade computing, i.e., at which location should the shared memory be placed ?

⏭ 1.8 LIMITATIONS AND CHALLENGES OF BLOCKCHAIN

While blockchain has the potential to transform internet networks and numerous industries such as finance, insurance, healthcare, retail, media, and so on. It also faces a slew of technical and business challenges. There are some limitations to this technology that have had a significant negative impact on its mass adoption, interoperability, technical and professional support, and regulatory requirements. Some of these constraints are contradictory in the sense that some of blockchain's fundamental properties must be compromised before it can be used effectively. The anticipated limitations to blockchain application are listed in the following sections.

⏭ 1.9 BLOCKCHAIN IMPLEMENTATION - LIMITATIONS

GQ. What are the potential solutions for the scalability problem of blockchain?

The lack of a standardised protocol that makes blockchain technology accessible to all sectors is a major impediment to its use. The major limitations of blockchain can be broadly classified into the following categories.

1.9.1 Limited Scalability

- Because each node must validate the transactions executed in the system, scalability is a significant limitation in the blockchain network. As a result, the rate at which a transaction can be processed is limited.

- Researchers are continuing to work on distributed ledger technology based on blockchain-like hyperledger fabric to address scalability issues.
- Scalability, security, and decentralisation are the three components of blockchain that must be addressed. You can only determine two of the three attributes at any given time. For example :
  (1) Security and decentralisation can be implemented, but scalability must be compromised. The Bitcoin blockchain is having issues in this area.
  (2) Scalability and security can coexist, but security decentralisation must be sacrificed. BigTable and Cassandra are two examples of such decentralisation-compromising functionalities.
  (3) Although decentralisation and scalability can be prioritised, blockchain security will bear the brunt, which is undesirable for many industries. Such characteristics include private chains. Because blockchains use a consensus algorithm that requires all transactions to be verified, there is a limit to the number of transactions that can be completed in a given amount of time. There are solutions available today, such as distributed ledger technology, that allow for more transactions per second. On the other hand, it has a negative impact on the blockchain's transaction rate or speed.
- Scalability is one of the most significant drawbacks of blockchain. Most public blockchain consensus systems that operate in a decentralised manner must strike a balance between minimal network output and a high degree of centralization.
- The following is a definition of the scalability problem:
  (1) Every node on the blockchain network processes every transaction and maintains a copy of the entire state.
  (2) Because inter-node latency grows logarithmically with each additional node, the blockchain network becomes less secure as more nodes are added to its system.
  (3) As the size of the blockchain increases, more storage capacity and computational power are required to ensure that the network works at peak efficiency.
  (4) As the blockchain grows in size, only a few nodes will be able to process a block at some point, posing the possibility of reverting to centralization.
- The challenge of scaling a blockchain can be comes down to two main factors :
  (1) The blockchain itself expands in size as more people utilise it. Each node must keep a more detailed storage history. The bitcoin blockchain alone has a storage of 163 Gigabytes as of March 2018. Every node must store this amount of data, which means that anyone starting a bitcoin node must have at least this much storage space. This figure is only going to rise.
  (2) There is no incentive to be a node that manages the ledger. Miners, who receive a bitcoin block reward for each block they create and add to the chain, are the only people who have an incentive to keep the network running.
- Maintaining a node becomes more difficult and expensive for a single person as a result of these two issues. Miners may run the network, but nodes are in charge of keeping the ledger up to date.
- This creates a challenge for any blockchain; in order to scale, the network must be centralised. Developers, blockchain corporations, and research organisations all have different solutions to offer. Sharding is one solution that is currently being used.
- Each node in the sharding process is not required to handle all of the data. The entire end-to-end blockchain state can be split into separate shards, removing the need for each node to keep all of the data. Specific nodes would only have to save a subset of the state with shards.

- As a result, when transactions are initiated on the blockchain, they are only directed to the shards that they impact.. Rather than processing the entire state, each shard will only process a subset of it. However, in order to automatically establish reliable connectivity across the shards, a commanding mechanism is required.
- Some initiatives are also exploring the possibility of nodes running independently on regular off-the-shelf hardware, delivering complete decentralisation while also addressing the scalability issue.
- Experts are building an all-peer-to-peer network using Shardus (a project). The blockchain will benefit from fast throughput, low latency, and immediate finality, as well as security assurance, as a result of sharding and auto-scaling. This is supported by a proof-of-quorum algorithm and the Shardus Distributed Ledger.
- As the network grows in size, this one addresses the issues of linear scalability and state sharding.
- Some other potential solutions for scalability problem are :

(1) **Segwit (Bitcoin architecture-specific) :** Segregated witness (Segwit) is a technique for isolating transaction signatures (i.e., "witnesses") from the remaining transaction data, thereby removing some unnecessary weight from the blocks.

(2) **Increasing the block size (Bitcoin architecture-specific) :** More transactions can be stored in each block if the block size is increased. This lets the network to handle more transactions per second.

(3) **Off-chain state channels :** These state channels include mechanisms that allow interactions to occur within the blockchain rather than outside of it.

(4) **Plasma :** This solution employs a group of smart contracts that run on top of a root blockchain (ie, the main Ethereum blockchain).

(5) **Off-chain computations :** TrueBit is an example of an off-chain calculating solution that enables scalability transactions across Ethereum smart contracts. TrueBit, like state channels, relies on an external layer of the blockchain to do the heavy lifting. Off-chain computation is a notion in which all difficult mathematical operations are handled by a layer apart from the blockchain. This will not only relieve the Ethereum Blockchain of its strain, but will also lower transaction verification and processing expenses. This computing model is not used by all blockchain nodes.

Instead, only a select few will perform difficult computations in conjunction with a deposit. If the participant's solution is correct, the participant receives both the award and the deposit. Otherwise, the deposit is forfeited. Verifiers are also used to perform verification in off-chain computation. By adding a separate layer to the blockchain, this approach ensures that procedures with slow transaction speeds are executed independently. Transactions can be executed in less than a second by moving them off-chain, while information from public ledgers has a higher level of privacy.

(6) **Proof-of-stake (PoS)** : Instead of computing power, stakeholders vote with money (or, in the case of Ethereum, ether). Bitcoins and Ethereum work on the Pow case, which is a system in which difficult mathematical equations must be solved in order for the verification mechanism and transaction processing to function. As a result, the speed of transactions on Ethereum's blockchain slows. Because PoW has drawbacks, a new proof-of-stake mechanism was developed to achieve faster transactional speeds and higher levels of security.

The other major distinction between the proof-of-work and proof-of-stake models is that in POS, validators do not earn ethers but instead receive a transaction fee for their efforts. Pos will have much faster transaction speeds.

(7) **Transient blockchain :** Saito has come up with a novel solution to both issues. To avoid the growing pains of keeping large ledgers permanently, Saito devised a transitory blockchain that deletes itself at predetermined intervals known as 'Genesis Periods.' A user can keep any information on-chain by paying a small fee, which is effectively the cost of the storage that it uses. The second option presented by Saito is to provide an incentive to nodes that provide storage capacity to increase the amount of data they can store. Nodes with more storage

capacity effectively become miners themselves by having the ability to claim new Saito tokens as they are issued. The more storage capacity they provide, the better their chances of receiving a large sum of money. As a result, this is similar to the probabilistic mechanism used by Bitcoin miners to spend money in order to make money.

- The difference is that Saito, by doing work for the network, assists it in scaling. Although these technologies have brought limited improvements to date, the fundamental constraints are due to present systems' low storage and computing capabilities.
- **Hardware scalability :** The cost of hardware is rising as the popularity of blockchain grows. This raises the cost of entrance into mining, causing it to become more centralised. Aside from that, there are energy and economic concerns.

1.9.2 Limited Privacy

- Even though identities in a distributed ledger are anonymous, by looking at the transaction address, it is feasible to correlate the user identity with that address and obtain information about the user. Because anyone can create a new wallet anonymously and transact through that wallet, blockchain transactions are not tied to one's status.
- Identity verification data, like as security numbers, cannot be stored openly in public smart contracts. Another factor that is not controlled in an open, ultimately unsecured smart contract is credential management.

1.9.3 Lack of Technical Knowledge

- Despite the fact that blockchain is growing increasingly popular, most investors are having difficulty understanding the various technical concepts because there is no sufficient documentation available to assist customers.
- As a result, investors are unable to get their questions answered. Furthermore, many investors are unaware of the Initial Coin Offering (ICO), which is well-known for raising cash in the non-equity market.
- Developers will require time to understand, explore, and pilot any specialist technology in projects. Furthermore, educational establishments require time to offer new technology courses, and faculty must be trained on the subject. As a result of all of this, there is a serious lack of skilled blockchain engineers.
- This scarcity of experienced developers and workers has a negative influence on blockchain innovation. Experts are created as a result of their experience in any technology.
- According to statistics, there is a huge demand for blockchain professionals, which has increased the salary expectations of the few available in the market.

1.9.4 Security Concerns and Flaws

- Blockchain is a network of people. If more than half of the people in this network are corrupt and distort the data, the truth will be considered as a lie, and this could be one of the key causes of network failure. As a result, network users must closely monitor transactions in order to avoid data misuse and ensure security.
- As the blockchain grows in terms of nodes or blocks, corruption among participants jeopardises blockchain security. If more than half of the nodes in a blockchain network validate anything, it is considered true. As a result, if more than half of the nodes in a network approve a fraudulent transaction (false), the myth is accepted by the whole blockchain network. This is known as the 51% attack, and it is a serious security weakness in blockchain and its uses.
- The biggest security challenges of Blockchain are as follows :

  (1) **Spoofing of payment information :** When a user is sending money, malware may replace the crypto wallet address with another one. Not everyone is observant enough to double-check an address, which is typically a jumble of alphanumeric characters and symbols.

(2) **Phishing :** Exchange users may be misdirected to a phishing website, where their crypto wallet credentials may be stolen. For example, customers may receive a fake mail in their inbox informing them of a substantial upgrade to their cryptocurrency wallet. The notification may state that the user must sync his or her wallet with a newly hard-forked network; if this is not done, the user will be unable to send or receive cons.

(3) **Crypto wallet theft :** The vast majority of investors and users keep their bitcoin wallet files on their personal computers, making them vulnerable to malware attacks.

1.10 BLOCKCHAIN IMPLEMENTATION - CHALLENGES

The main problem with blockchain is a lack of knowledge about the technology. Particularly in non-banking areas, there is a general misunderstanding of how it works. In addition, the following factors must be taken into account.

1.10.1 Transaction Processing Speed

- One of the other obstacles that blockchain faces is scalability in terms of transaction processing speed. Contrary to popular belief, the most popular cryptocurrency, Bitcom, can only process an average of 7 transactions per second.
- At the same time, Visa can handle 24000 transactions per second and PayPal can handle approximately 200 transactions per second. Bitcoin Cash has a transaction rate of 50 to 60 transactions per second, which is still low when compared to Visa. Among all cryptocurrencies, Ethereum is the slowest, with only about 20 transactions per second.
- Transaction speed has been hampered by the EthereumBlockchain's growing size and complexity, and transactions can now take hours or even days to validate. The Directed Acyclic Graph (DAG) has a solution for slow transaction speeds.
- DAG has solved the problem by confirming transactions on a per-transaction basis rather than a block-by-block basis. DAG allows users to handle each other's transactions on a microscale rather than waiting for miners to execute a large block of transactions, lowering transaction fees and increasing speed.

1.10.2 Complexity

- The underlying technology of blockchain is difficult for beginners to understand because the code contains a large number of mathematical computations. Blockchain technology is still in its infancy in terms of maturity, with numerous new processes, standards, and unique technical elements.
- Another aspect of blockchain that adds to its complexity is that it is based on decentralisation and encryption as its basic backbone.As a result, various difficulties arise in its functionalities, such as transaction speed, transaction verification, and data limits that can be transferred or received.
- The expansion of the blockchain network introduces a new layer of complexity to the system. Blockchain, like any other decentralised system, is susceptible to fraudulent attacks that have the potential to corrupt the network. A robust network algorithm connects a large number of users and nodes to form a stable blockchain network.

1.10.3 Implementation and Operation Cost

- Even though blockchain provides users with reduced transaction costs, the underlying technology and implementation costs are not insignificant. Four inputs have a significant impact on the expenses of both public and private blockchain solutions, and they are fully dependent on the use case and the goals that a company achieves.
- There are other factors to consider, but the following are the most important :
  (1) Transaction volume (2) Transaction size
  (3) Node hosting methods (4) Consensus protocol

- Each of them requires a significant amount of computing power, and the blockchain grows in size as more nodes and other connections are added. All of this adds to the operating expenses.
- The development of blockchain solutions is typically an agile process, which raises costs due to the involvement of numerous Business and IT stakeholder.

1.10.4 Storage Constraints

- Because the blockchain database is append-only and immutable, data is stored indefinitely by every full node in the network. As a result, storage remains a significant challenge for practical blockchain applications.
- The fundamental problem in achieving high transaction speeds in blockchain is reducing data and node load. At the moment, each node must store the complete data set.
- So, if the blockchain includes a system that allows nodes to retain only the data that is most frequently utilised or locally relevant for transaction processing, we can process transactions more quickly. Such methods can improve the effectiveness of blockchain, and more research into such systems is required to overcome such storage limits.

1.10.5 Lack of Governance and Standards

GQ. How does lack of governance and standards affect the blockchain ?

- Numerous blockchain networks have emerged on the market as a result of the emergence of over 2000 cryptocurrencies and thousands of applications that use distributed ledger technology. Because there are no worldwide standards for interoperability among various blockchain networks, they operate in silos.
- As a result, there are numerous protocols, coding languages, and consensus processes to enable these networks, resulting in interoperability concerns amongst blockchain networks. Ark (which uses the Smartbridges architecture) and Cosmos (which uses the Interblockchain Communication (IBC) protocol) are two solutions proposed to address interoperability difficulties.
- Decentralized blockchain systems do not have centralised control, and it is a completely trustless, open, and permissionless system in which no one is held accountable for creating and maintaining standards. At the moment, there are no global regulatory standards for blockchain applications.
- The blockchain can achieve lower total costs, more efficient consensus methods, and improved interoperability if standards are established. This has implications for both the issue of onboarding new developers and blockchain security concerns.
- Because blockchain technology is based on a distributed ledger, centralised databases will always outperform blockchain networks. When a new block is added to the network, it must execute all of the processes, just like a traditional database.
- It must, however, take care of new processes to ensure that performance is not jeopardised. Some of the processes that must be completed are as follows :

  (1) **Signature verification :** Every transaction on the blockchain must be digitally signed, and it must be authenticated using a public or private key. The validation process as a whole is time-consuming and complicated.

  (2) **Redundancy :** In a blockchain network, each node must travel and process each intermediate node separately in order to reach the target node. Nodes in a centralised database system, on the other hand, can be handled in parallel without relying on any other nodes. This redundancy has a direct impact on the performance of the blockchain.

(3) **Attaining consensus :** With a blockchain, it is critical that a common consensus is obtained for each transaction completed in the blockchain network. The back-and-forth interactions required to reach consensus can consume a significant amount of time and resources, depending on the size of the network and the number of blocks or nodes in a blockchain.

(4) **Alternative consensus algorithms :** Two alternative consensus procedures are delegated proof-of-stake (DPoS) and practical byzantine fault tolerance (pBFT). These two alternative consensus mechanisms are more appropriate for non-financial blockchains. As of now, these methods necessitate additional research and development for a variety of application situations.

(5) **Delegated proof-of-stake (DPoS) :** A small number of delegates will maintain the ledger in this approach, and users will vote for them using a real-time algorithm. The network's throughput is significantly higher due to the fewer active nodes. Because each node is paid through inflation, it makes sense to keep a large data centre for network support. The disadvantage of the DPOS is that it is centralised. This means greater network control and less ability to be transparent and open. As a result, if a user can buy more votes, they can easily control and change the network rules to their benefit.

- Every node in the pBFT algorithm authenticates the block's state in turn. PBFT is capable of performing a variety of functions very efficiently, resulting in higher throughput and lower resource utilisation costs.
- The disadvantage of pBFT is that it is easily vulnerable to Sybil attacks due to the low cost of adding new network nodes. Fraudsters may be able to establish more nodes and attack the entire chain if they reach 33% of the network.
- In contrast to the Bitcoin network, which requires 50% of the network to gain control, pBFT requires only 33% of the network to gain control, making the pBFT more vulnerable to takeover.

1.10.6 Lack of Formal Contract Verification

Formal verification of smart contracts is a huge unsolved topic because it necessitates a definition of "formal proof." Formal verification is used to ensure that the programme operates as expected.

1.10.7 Energy and Resource Consumption

- The blockchain's size and number of transactions grow as more nodes are added, necessitating the need for more resources and infrastructure to operate. Miners must validate each block in a blockchain.
- The time and effort required by miners to validate the network grows dramatically as the blockchain grows in size. To validate transactions, miners must generate a large number of mathematical solutions, which requires a significant amount of processing power.
- The average cost of running the blockchain is also significant due to the robust hardware architecture required because each node has extreme fault tolerance to provide zero downtime and high levels of network and data security. All of this necessitates considerable energy consumption.

1.10.8 Simplified Mining

- Mining for a common blockchain cryptocurrency, such as bitcoin, has created an oligopoly problem. The vast majority of mining power is concentrated in a few mining pools that split revenue. This directly contradicts the fundamental principle of bitcoin and cryptocurrency decentralisation.
- Ethereum is based on the proof-of-work (PoW) algorithm. Only when the validation conditions are met are the blocks created and connected to the Ethereum network. This method ensures that each transaction is legal. Despite the fact that this PoW technique secures transactions and transfers, it has some limitations.

- This strategy provides no benefit to miners, and there is no way to detect fraudulent activity or transactions in the blockchain using this method. DAG is attempting to address this issue by defining key responsibilities for blockchain transactions, but there is still a long way to go. Miners lack the authority to manage the blockchain in this manner.

1.10.9 Human Errors

- Manual data entry errors will result in out-of-date information or data inconsistency in the database. Thus, data must be validated before entering the system and confirmed after entering the system.
- The phrase "garbage in, garbage out" refers to data in the context of the blockchain. Another common cause of system failure is human error.
- Though we can think of the blockchain as a single database with each block serving as a storage stack, the date appended to the database is entered by humans in the first mile.
- If this data is to be error-free, it must be correctly fed at the entry point. However, no method or system exists to validate the data that is entered to be processed within the blockchain network. As a result, trust is an issue, because incorrect data entered into a blockchain could contaminate the data of the entire system.

Chapter Ends...

❑❑❑

Module 2

Cryptocurrency

Syllabus

2.1 Cryptocurrency: Bitcoin, Altcoin, and Tokens (Utility and Security), Cryptocurrency wallets: Hot and cold wallets, Cryptocurrency usage, Transactions in Blockchain, UTXO and double spending problem

2.2 Bitcoin blockchain: Consensus in Bitcoin, Proof-of-Work (PoW), Proof-of-Burn (PoB), Proof-of-Stake (PoS), and Proof-of-Elapsed Time (PoET), Life of a miner, Mining difficulty, Mining pool and its methods

2.1 CRYPTOCURRENCY – BITCOIN, ALTCOIN, TOKEN

2.1.1 Introduction

- The term cryptocurrency originated from the words *crypto* and *currency*. Crypto means obscured or hidden and currency means money. The amalgamation of both these terms refers to all an encrypted decentralized digital currencies. Cryptocurrencies are digital in nature that are designed to be faster, cheaper, and highly reliable. It is an open source encrypted ledger of transactions built on blockchain technology. It is created via mining and users can transact directly with each other over the Internet.
- In the world of cryptocurrency, the importance of banks to store more and facilitate transactions is redundant. A cryptocurrency system is resilient against fraud as cryptocurrency users can not only record but also verify each other's transactions. The acceptance of all the nodes participating in a transaction is a must before the transaction is committed onto a ledger. A ledger that is made up of digital transactions is public and secure and doesn't require any financial institution such as banks to offer trust; so, it is referred to as a trustless system.
- Cryptocurrency is still emerging and is considered as an investment instead of a currency. Trading on cryptocurrency exchanges is gaining importance despite its volatile prices. Bitcoin, Ether, Litecoin are some of the examples of popular cryptocurrencies.

2.1.2 Bitcoin and the Cryptocurrency

- By definition, bitcoin is a decentralized digital currency that enables instant payments to anyone and anywhere in the world.It is a cross-country payment system, which was one of the primary objectives behind the development of bitcoin. Another objective was to have a cross-country transaction support such that no government or organization will have a control over it.
- The essential property of bitcoin is its decentralized architecture (i.e., no central authority), which helps in obtaining a system where no one has an external control over this currency system. It works over a complete peer-to-peer network and it supports different levels of securities so that the entire system becomes tamper proof.

In bitcoin, there are two broad operations

- **Transaction management :** Transferring of bitcoins from one user to another.
- **Money issuance :** Regulate the monetary base of bitcoin (i.e., economical aspects of a digital currency needs to be ensured).

Bitcoin basics : Creation of coins

- There needs to be a controlled supply of money (i.e., the currency value needs to be limited in order to have its value) For instance, if a lot of currency is put in the system, then the value of that particular currency will get gradually reduced. Any such maliciously generated currency needs to be rejected and the actual currency needs to be accepted that is flowing in the system.The new bitcoins need to be generated like a normal currency. The new bitcoins are generated during the bitcoin mining time. So, each time a user discovers a new block (i.e., mining), the system generates a new bitcoin and the person who is investing his/her power, system power, and time to generate the new bock by participating in the mining procedure receives the newly generated bitcoin.
- As mentioned earlier, the flow/regulation of bitcoin needs to be managed appropriately. So, in case of a standard bitcoin architecture, the rate of block creation is adjusted over every 2016 blocks and the aim is to have a constant two-week adjustment period.The number of bitcoins generated per block is set to decrease geometrically with a 50% reduction for every 210,000 blocks, or approximately 4 years. This reduces, with time, the amount of bitcoins generated per block. The theoretical limit for generating total bitcoins is slightly less than 21 million.

- So, as the time progresses, the miner shall receive less reward. But, the question that arises is that how the mining fee can be paid or how the incentives are provided to the miners for participating in the mining procedure? Because the reason for a miner to participate in the mining procedure is to get some bitcoin. In the mining process, the miners have to solve some mathematical puzzle by investing time and system power. If they are successful, then they are rewarded with some bitcoin, but as the bitcoin value will decrease with time and approximately 21 million bitcoins are generated, the system will not generate any new bitcoin; so, the miners will not get paid from the network for participating in the mining procedure. In such a case, their incentive would be to increase the transaction fee or acquire more transaction fee from the participating user to pay the miners for participating in the mining procedure.

☞ Projected bitcoins w.r.t time

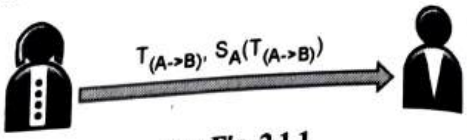
Date reached	Block	Reward Era	BTC/block	Year (estimate)	Start BTC	BTC Added	End BTC	BTC increase	End BTC % of Limit
2009-01-03	0	1	50.00	2009	0	2625000	252500	Infinite	12.500%
2010-04-212	52500	1	50.00	2010	2625000	2525000	5250000	100.00%	25.000%
2011-01-28	105000	1	50.00	2011*	5250000	2525000	7875000	50.00%	37.500%
2011-12-14	157500	1	50.00	2012	7875000	2625000	10500000	33.33%	50.000%
2012-11-28	210000	2	25.00	2013	10500000	1312500	11812500	12.50%	56.250%
2013-10-09	252500	2	25.00	2014	11812500	1312500	13125000	11.11%	62.500%
2014-08-11	315000	2	25.00	2015	13125000	1312500	14437500	10.00%	68.750%
2015-07-29	367500	2	25.00	2016	14437500	1312500	15750000	9.09%	75.000%
2016-07-09	420000	3	12.50	2016	15750000	656250	16406250	4.17%	78.125%
2017-06-23	472500	3	12.50	2018	16406250	656250	17062500	4.00%	81.250%
	525000	3	12.50	2019	17062500	656250	17718750	3.85%	84.375%
	577500	3	12.50	2020	17718750	353250	18375000	3.70%	87.500%
	630000	4	6.25	2021	18375000	328125	18703125	1.79%	89.063%
	682500	4	6.25	2022	18703125	328125	19031250	1.75%	90.625%
	735000	4	6.25	2023	19031250	328125	19359375	1.72%	92.188%
	787500	4	6.25	2024	19359375	328125	19687500	1.69%	93.750%

- From the above table, we can see that if a new block is generated, then we would get 50.00 BTC for every block. As the time progressed, the value reduced to 25.00 BTC for every block. Furthermore, the value reduced to 12.50 BTC for every block and then it gradually reduced to 6.25 BTC for every block.
- So, this is how the reward for generating a new block will get reduced over time. The above table provides an indication about how much time is required to completely generate a targeted bitcoin (i.e., the bitcoin reward will nearly get close to zero). Also, once this count reaches to 21 million bitcoins in the network, no further reward will be provided from the system.

☞ Sending payments in the bitcoin network

- Whenever an individual sends a payment in the bitcoin network, we need to ensure that any other person cannot spend the bitcoin owned by one person.
- For instance, Person B cannot spend Person A's bitcoins by creating transactions in his/her name. Bitcoin uses public key cryptography to make and verify digital signatures. So, each person who is participating in the bitcoin network has one or more addresses with an associated pair of public and private keys.

- This means that every user may have one or more address based on the bitcoin wallet, but every bitcoin address will get associated with a pair of public and private key.
- **Person A wants to transfer some bitcoin to Person B.**
  - Person A can sign the transaction with his/her private key.
  - Anyone can validate the transaction with Person A's public key.



(1A48)**Fig. 2.1.1**

Steps to be followed by Person A if he/she wants to transfer a set of bitcoins to Person B

- Person B sends his/her address (i.e., cryptographically generated) to person A.
- Next, Person A adds Person B's address and the amount of bitcoin that needs to be transferred in a *transaction* message.
- Person A signs his/her transaction with his/her private key and announces the public key where anyone can validate the transaction. Then, Person A broadcasts the transaction in the bitcoin network for all to see that transaction.

☞ **Double Spending (Refer section 1.5.2)**

Bitcoin anonymity

- Bitcoin is permissionless where there is no need to setup any account or there is no need of any email address, username, or password to login to a bitcoin wallet. This means that an individual can join anytime in the network. When an individual joins in a network, he/she need not register the public and the private keys.
- The wallet can generate them for the users. There is a bitcoin address that is used for transaction. Note that this bitcoin address is not the username or identity (i.e., they do not carry the identity of a particular user; they are some type of an anonymous address through which it is difficult to guess who is the actual user). A single user can have more than single addresses as well.

☞ **How the bitcoin addresses are generated ?**

A bitcoin address mathematically corresponds to a public key based on elliptic curve digital signature algorithm (ECDSA), i.e., the digital signature algorithm used in bitcoin.

The algorithm is something like this that you have generated the public key. So, once the public key is generated. For example, Person A has generated the public key, we then apply a hash function of 160 bit. After applying the hash function, we extract the first few bits from the hash and use it as the address. A sample bitcoin address looks like this "*1PHYrmdJ22MKbJevpb3MBNpVckjZHt89hz*". By looking into this address, it is difficult to guess who the actual user is. As mentioned earlier, each person can have many such addresses, each with its own balance. Therefore, it becomes difficult to know which person owns what amount. So, this particularly prevents the anonymity of bitcoin transactions.

2.1.3 Cryptocurrency Basics

2.1.3.1 What is Cryptocurrency ?

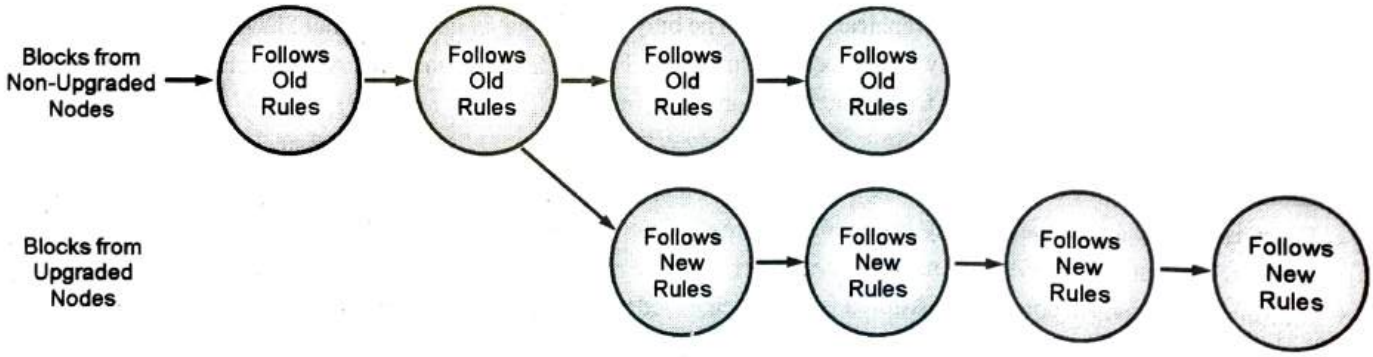
GQ. What is cryptocurrency ? State its characteristics.

- A cryptocurrency is a digital asset that is used as a medium of exchange in blockchain. Some of the popular cryptocurrencies are bitcoin and ether of Ethereum. Since the creation of 1st decentralized cryptocurrency Bitcoin, thousands of alternative digital currencies, referred to as altcoins or coins and tokens have emerged. According to CoinMarketCap as of October 2019, there are approximately 3000 active cryptocurrencies.
- Cryptocurrency has given rise to a new form of raising capital for start-up ventures called initial coin offering (ICO), which is much faster than traditional fund-raising models.

- Cryptocurrency and blockchain projects have open-source code to foster trust and safety. Developers can copy the open-source software and modify it to build compatible applications. This process of duplication from an existing blockchain is called forking.
- A fork creates an alternative version of a blockchain. Forking is performed to apply upgrades or new governance rules to a network *(i.e., implementing new features and functionalities in the blockchain network).*
- **There are two types of fork :**
  - **Hard fork :** It is a radical change to a network's protocol that makes previously invalid blocks and transactions valid, or vice-versa. A hard fork requires all nodes to upgrade to the latest version of the protocol software. In other words, a hard fork refers to a radical change to the protocol of a blockchain network that effectively results in two branches, *one that follows the previous protocol and one that follows the new version.* In a hard fork, holders of tokens in the original blockchain will be granted tokens in the new fork as well, but miners must choose which blockchain to continue verifying. A hard fork can occur in any blockchain, and not only Bitcoin (i.e., Bitcoin –Bitcoin Cash fork; Ethereum Classic - Ethereum fork). The Fig. 2.1.2 shows that nodes that are not upgraded reject the new rules, which creates a divergence, or hard fork, in the blockchain.

Other hard fork examples

- Bitcoin hard fork in 2018
- **New coin :** Bitcoin cash
- **New feature :** Block size increased from 1 MB to 8 MB
- **Purpose :** To allow for more transactions to be processed, thereby reducing fees that users pay and minimizing bottleneck of a bitcoin network as usage increased.

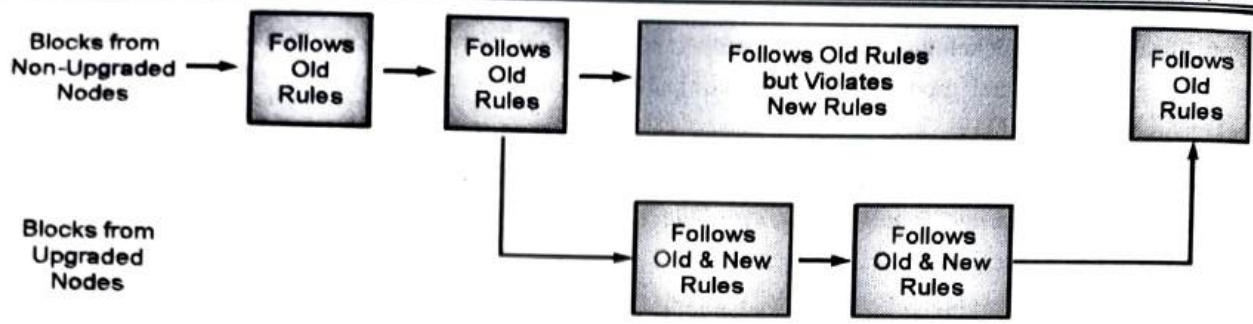


(1A63)**Fig. 2.1.2 : Hard fork**

- **Soft fork :** It is a change to the software protocol where only previously valid transaction blocks are made invalid. Because old nodes will recognize the new blocks as valid, a soft fork is backward compatible. This type of fork requires only a majority of the miners upgrading to enforce the new rules as opposed to a hard fork that requires all nodes to upgrade and agree on the new version (e.g., Bitcoin protocol upgraded to SegWit).

☞ **Other soft fork examples**

- **Bitcoin improvement proposal 66 (BIP-66) :** A soft fork on bitcoins signature validation.
- **Pays to script hash (P2SH) :** A soft fork that enabled multi-signature addresses in a bitcoin network.



(1A64)**Fig. 2.1.3 : Soft fork**

However, there exist original cryptocurrency blockchains like Ethereum and Ripple, amongst others, which have their codebase. Ethereum has brought in a new category of software applications called decentralized applications (DApps) and smart contract protocols.

2.1.3.2 Characteristics of Cryptocurrencies

Cryptocurrencies have distinct characteristics, which set them apart from other traditional fiat currencies.

- **Decentralized :** Cryptocurrencies are not controlled by any bank. All nodes in the network work together in mining or processing a transaction. Decentralization means no central body can control or influence a cryptocurrency's value.
- **Form of existence :** Cryptocurrencies only exist in a digital form. They are not tangible and are essentially developed by software code and cryptographic algorithms.
- **Limited supply :** The supply of cryptocurrencies are limited. The maximum supply of cryptos that can ever be generated or mined is defined when the genesis block is created.
- **Global access :** Because of the decentralized nature of blockchain, anybody can access and transact using cryptocurrencies despite their geographical location. The only requirement is that they should have internet connection. Cryptocurrencies do not have any cross-border restrictions. The processing times of cryptocurrencies is fast and the transaction fee is also very low as there is no third-party intervention.
- **Anonymity and transparency :** Whenever a standard transaction is performed, personal details are stored in traditional databases, and such transactions can be monitored, tracked, and retrieved by central authorities as and when required. There is no privacy and anonymity followed. Conversely, in cryptocurrencies, a transaction is linked to an individual's address and not the individuals name or any other personal information. This is how anonymity is maintained. In addition, every transaction record is stored in a blockchain ledger making it transparent and verifiable to everyone.
- **Impossible to duplicate :** Cryptocurrency encryption and consensus protocols make forging cryptocurrency impossible.
- **Irreversible :** Cryptocurrency transactions are irreversible in nature. Once a transaction is recorded and verified, it is irreversible and cannot be modified. This promotes trust, integrity, and auditability. It also means that if a cryptocurrency is sent to an incorrect address, then it is lost forever.

2.1.3.3 Cryptocurrency Wallets

Storage vehicles for cryptocurrencies are referred to as wallets. A cryptocurrency wallet, sometimes known as a digital wallet, is a software program that stores a user's private and public keys enabling him/her to transact crypto assets. It is a management system that interacts with various blockchains to permit users to send and receive digital currency and monitor their balance. Cryptocurrencies are stored immutably on a blockchain network using a user's public key. This key is used by other wallets to send funds to the user's wallet address. Nonetheless, private key is required if users want to spend cryptocurrency from their address.

On the basis of their storage, cryptocurrency wallets are classified into two types

- **Hot wallets :** It is designed for online everyday transactions. It is always connected to the internet and hence it becomes a strong candidate for hackers. Thus, it is not advisable to use a hot wallet for long-term storage. They are free, easy to set up, and convenient to use. Examples of hot wallets include exchange wallets, web wallets, and software wallets like Exodus.
- **Categories of hot wallets**
  - **Desktop wallets :** They can be downloaded and installed in a desktop/laptop. As they are stored locally, a user has complete control of the wallet. They possess features like built-in exchange platforms, multi-currency support, etc. The drawback of desktop wallets is that they are dependent on security features installed in an individual's computer. If an individual's system is infected by a virus, then there is a risk to loose all funds. Examples of desktop wallets are Bitcoin Core, Electrum, and Exodus.
  - **Mobile wallets :** They are designed to operate on smartphones. They can be easily carried wherever an individual goes and makes purchases from merchants who accept cryptocurrencies via QR codes. Vulnerability to malicious applications and viruses is more compared to desktop wallets. Security features like wall encryption is needed to be installed on mobile phones and sufficient backup is required for private keys if a mobile phone is stolen or damaged. Examples of mobile wallets are Coinbase Wallet, Trust Wallet, and Mycelium.
  - **Online wallets :** They are also called as web wallets that run on cloud and hence a user need not download or install any application. They can be accessed from anywhere via a web browser. Online wallets are hosted and managed by third-parties and so the chances of susceptibility are more. Examples of online wallets are MyEtherWallet, GreenAddress, and MetaMask.
  - **Cold wallets :** It is a digital wallet this is not connected to the internet. They are not free. As they are available offline, they are more secure and used for long-term storage of cryptocurrencies. Examples of cold wallets include paper wallets and hardware wallets.
- **Categories of cold wallets**
  - **Hardware wallets**

    They are physical and electronic devices that use a random number generator to produce public/private key this is stored in a device. It is connected to the internet whenever a user needs to send/receive payments and disconnects once a transaction is executed. Transactions are confirmed via private keys that are saved offline. Even though hardware wallets are more secure than hot wallets, they are difficult to access and less user-friendly. Examples of hardware wallets are Trezor and Ledger.
  - **Paper wallets**

    They are a piece of paper with a crypto address and its private key is printed in the form of a QR code. The QR codes can be scanned to execute cryptocurrency transactions. They are secured as paper wallets cannot be hacked easily. The disadvantage of paper wallets is that it contains only a single public/private pair, and hence, they can be used only once for the entire amount in a transaction. They need to be kept safe from water and fire.

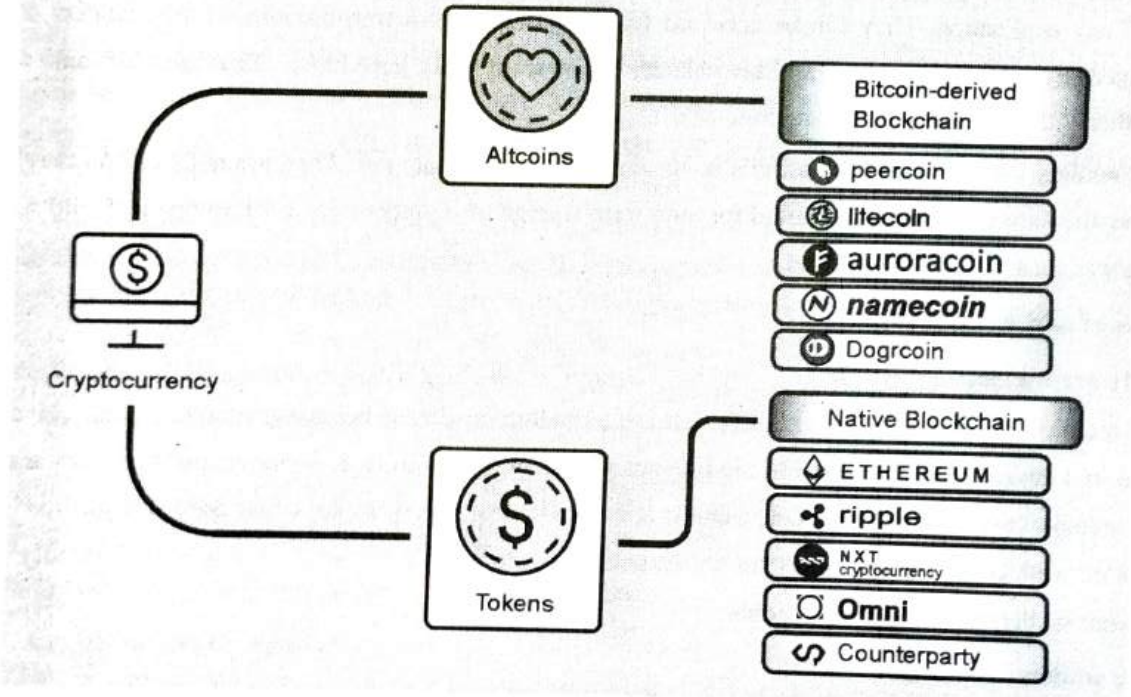
2.1.3.4 Differentiation between hot wallets and cold wallets

GQ. Differentiation between hot wallets and cold wallets.

Sr. No.	Parameters	Hot wallets	Cold wallets
1.	Purpose	Suggested for short-term storage. Well-situated for everyday transactions	Suggested for long-term storage
2.	Connectivity	Online storage; requires internet	Offline storage; doesn't require internet
3.	Cost	Free	Needs to be purchased
4.	Security	Vulnerable to cyber-attacks and malware	Impervious to cyber-attacks and malware as no internet is needed. But, it requires physical protection from loss or breakage
5.	Accessibility	Easy to access	Difficult to access
6	Categories	Desktop, mobile, and online wallets	Hardware and paper wallets

2.1.4 Types of Cryptocurrency

GQ. State and explain different types of cryptocurrency.



(1A65)**Fig. 2.1.17 : Types of cryptocurrency**

1. **Altcoins :** The term altcoins is often used incorrectly, and is somewhat misleading. It is used to refer to all cryptocurrencies that are not bitcoin, but not all cryptocurrencies that are not bitcoin are not technically coins. Altcoins (*actually referring to non-bitcoin coins*) are based on bitcoin's open-sourced, original platform with changes to its

underlying codes, thereby creating a brand new coin with a different set of features. Coins will typically differ in block production speed, the total supply of coins in circulation, transaction processing speed, and other features.

The main features that are common to all altcoins are as follows:

- They are peer-to-peer digital currencies that involves a mining process.
- They possess their independent blockchain
- They possess characteristics of money (i.e., they have limited supply and typically meant to operate only as a means of payment).

2. **Tokens :** They are cryptocurrencies that have their own native blockchain. The most prominent examples of tokens are Ethereum, Ripple, and Cardano. Tokens will operate on top of their native blockchain to facilitate the creation of decentralized applications. Tokens provide a way not only to define a protocol but to fund the operating expenses required to host it as a service. Tokens can represent any assets that are tradable from commodities to loyalty points to even other cryptocurrencies.

Tokens are broadly classified into two types:

- **Utility tokens :** They offer a token holder the right to use a product or service. Utility tokens can be redeemed in future to access a product or services of an organization. They are free of restrictions as they are not built to function as an investment. The token holder might possess certain voting rights within the ecosystem. Nonetheless, the token holder will not have any decisioning power in the direction that an organization can move.
- **Security tokens :** They are also referred to as equity tokens that enable a token holder to a particular share or equity in an organization. In other words, they can be considered as a digitized and tokenized form of traditional securities. Security tokens are viewed as an investment opportunity as they represent the percentage of ownership of an organization such as voting rights and decision making power. Also, security tokens are subject to regulatory restrictions.

2.1.5 Differentiation between Altcoins and Tokens

GQ. Distinguish between altcoins and tokens.

Sr. No.	Parameters	Altcoins	Utility tokens	Security tokens
1.	Foundation	They are built from scratch via an original code or open-source code of an existing cryptocurrency.	They are built from templates of an existing blockchain via smart contracts.	They are built from templates of an existing blockchain via smart contracts.
2.	Intention	Means of payment	To fund blockchain projects	To fund blockchain projects
3.	Use	Acts as a currency or cash	Signifies the right to use a product or service in the future.	Acts as an investment product and represents the shares of a company.
4.	Value	Dictated by market supply and demand.	Used as a utility and not linked to a company's performance. A user acquires voting and other usage rights within a specific blockchain it operates.	Gives partial ownership to the token holders. Its value is directly linked to a company's valuation.

Sr. No.	Parameters	Altcoins	Utility tokens	Security tokens
5.	Security	Protection as strong as the strength of the blockchain network protocols and user diligence.	Vulnerable to ICO scams and frauds.	Strict regulations aim to protect investors from scams and frauds.
6.	Regulation	Not regulated by any bank or central authority.	Unregulated	Regulated by a financial market authority.

2.1.6 Cryptocurrency Usage

GQ. Write a short note on : Cryptocurrency usage.

- **Ecosystem players :** A blockchain is only as strong as its community. If the players in the community are honest and engaged, then only it leads to a robust and sustainable ecosystem. Various players that constitute a blockchain ecosystem are as follows :
  - **Developers :** Blockchain developers develop, deploy, and maintain blockchain protocols, applications, and interfaces. Different functionality levels (i.e., consensus design) are designed by them. They also work on decentralized applications and smart contracts that run on blockchain. They not only monitor but also maintain blockchain as per needs and requirements.
  - **Miners :** They validate new transactions and record them on blockchain. A miner who can mine a specific block can be fixed based on a particular consensus mechanism. Sometimes, miners compete for privileges. They put forth their effort and computing power for solving a cryptographic problem and are awarded with block rewards or transaction fees.
  - **Users :** He/she uses the blockchain/cryptocurrency for the purpose for which it is designed. Note that a user can also be a miner, validator, or investor. Users are epitomized as nodes that possess a copy of the ledger for initiating, creating, or creating transactions.
  - **Merchants :** They are business entities that accept cryptocurrencies as a form of payment for their goods or services.
  - **Traders :** They are sometimes also called as brokers. They buy or sell cryptocurrencies based on price movements. They use a specific exchange method that is designed to compare and identify fixed cheapest exchange rates between tokens.
- ***Crypto mining :*** Miners generate wealth through mining. A miner needs to have some level of technical knowledge in setting up computing software. Blockchains differ in the mining systems that they use. Nonetheless, there exists some form of consensus algorithm and an incentive system. For instance, bitcoin uses PoW consensus mechanism. A successful miner fetches 6.25 BTC per block as a reward. For Ethereum, it is 2 ETH and additional fees per block. Miners use computing power to validate as well as produce a block. Miners can be of two types: solo miners and pool miners.
  - **Solo miners :** Here, the miner works independently to obtain a block.
  - **Pool miners :** Here, a group of miners join collectively and make use of their computational resources over a network for fast processing of a hash function.

  CPU/GPU, application specific integrated circuit (ASIC), and cloud mining are the different types of mining based on the processors used by a miner.

- **Airdrop :** It is a promotional activity aimed at spreading awareness among a blockchain community. Airdrop is a distribution event where a blockchain project distributes free tokens to wallet addresses for creating a market so that investors get attracted and begin investing in the project.
  - Benefits of Airdrop
    - Marketing and hype
    - Rewarding loyalty supports and investors
    - Wider distribution of tokens
- **Coin burning :** It is the process of permanently removing coins that are circulating in order to reduce the total supply. As airdrop is used to increase user awareness and engagement by offering free coins/tokens, coin burning is carried out to reduce the supply of coins so that the value of existing coins can be increased. A coin is considered to be burned when it is sent to an unspendable public address, which does not have an operable private key.
- The transaction is transparent on the blockchain network for any node to confirm that the coin was sent to the address, but the address does not belong to anyone and holds no practical importance. This is referred to as proof of burn (PoB) where anyone can check and verify that the coins were legally destroyed or burned.
- Coin burn takes place when a user calls the burn() function with the amount of coin he/she wants to burn. A smart contract then verifies whether the user has the stated amount or not. If the stated amount is present, then the defined amount of coins are burned. The reasons for performing coin burning are as follows:
  - Reward investors
  - Destroy coins that are not sold
  - Decentralize mining opportunities via PoB
- **Investing and trading :** It is the easiest way for people to generate cryptocurrency. The fundamental principal followed is *buy low and sell high.* Investors look for long-term profits whereas traders are focused on short-term goals. Unlike the stock market, cryptocurrency market is volatile where rise and falls occur in extremely short intervals of time. Hence, there is a huge possibility of making huge profits or having huge loss.
- **Cryptocurrency safety :** In fiat/traditional currency where banks are the protectors, if a bank account gets hacked or data is stolen or compromised, then it is the bank official's onus to catch hold of the wrongdoer. On the other hand, in the world of cryptocurrency, an individual is the sole owner. Transactions once executed cannot be reversed. Verification is performed via private keys, and if private keys are compromised/money is sent to an incorrect address, then that money is lost forever. Hence, safety measures are a must while storing or performing transactions in cryptocurrencies.

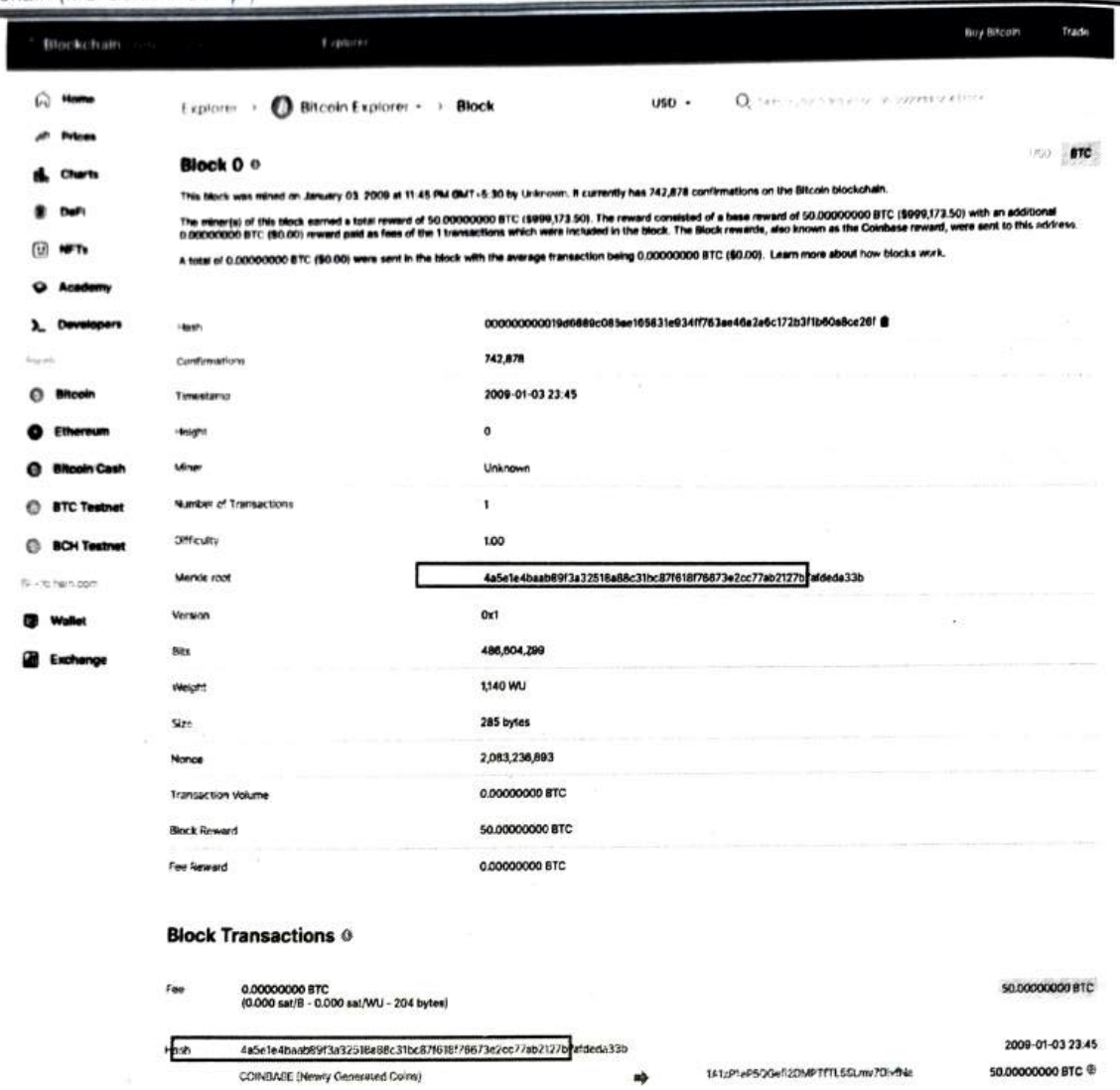
2.2 TRANSACTIONS IN BLOCKCHAIN

GQ. Explain transactions in blockchain.

Transactions are the most important part of any blockchain network. Essentially, every other part of a blockchain exits to ensure that transactions happen successfully and securely on an ongoing basis.

How transactions look like in blockchain?

Let us examine a bitcoin transaction on **Blockchain.com**(https://www.blockchain.com/explorer)



The first block (i.e., Block 0) is the genesis block. This block was created by Satoshi Nakamoto to introduce bitcoin.

☞ Components in a block

- **Hash :** It is the first component that acts as a unique identifier for the block. It is a 256-bit hash that is generated based on all the transactions in the block.
- **Confirmations :** They are a successful act of hashing a transaction and adding it to the blockchain through mining. They indicate how many times the block (i.e., Block 0) is verified in the network.
- **Timestamp :** It shows when the block was added after it was confirmed for the first time.
- **Height :** It represents the number of blocks connected on the blockchain, which is current 0, as shown in the figure.
- **Miner :** It is the name of the miner who confirms the transactions in Block 0. From the figure, it is evident that the miner here is *unknown*. However, by clicking on the miner name, information such as *address (i.e., unique identifier used to identify a particular address), total number of transactions this address has participated in, total amount received from this address, total amount sent from this addressover time*, and *current balance of this address*about the miner can be obtained.

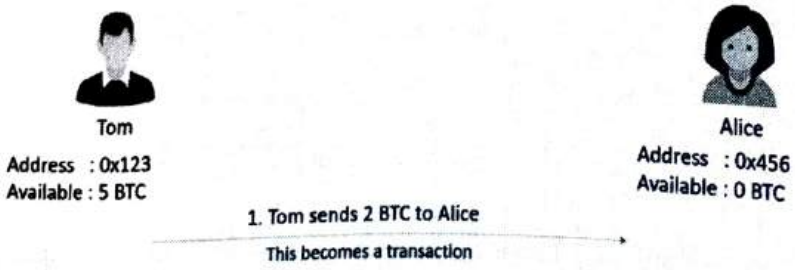
- **Number of transactions :** It shows the total number of transactions in the block, which is 1 for Block 0. In bitcoin, each block can have up to 1 MB of data.
- **Difficulty :** It indicates how hard it is to find the current hash to the block. The difficulty began with this baseline level but has increased significantly over the years and is constantly adjusting to keep time to create new blocks around 10 min.
- **Merkle root :** It is described as the root node of a Merkle tree, which is the descendant of all the hashed pairs of the tree. In other words, the Merkle root represents the hash of all the hashes of all the transactions in the block. Since there is only 1 transaction specified, the Merkle root and the transaction hash both are same.
- **Version :** It refers to the number of related protocol proposals underway.
- **Bits :** It represents a subunit of bitcoin where 1,000,000 bits (i.e., one million bits) is equivalent to 1 bitcoin.
- **Weight :** It is a measurement that is used to compare the transactions in relation to the size of the block limit.
- **Nonce :** It is a random value that can be adjusted into the correct hash, which is found using PoW.
- **Transaction Volume :** It indicates how many bitcoins were included in the block.
- **Block Reward :** It is the reward for the miner who calculated the hash for the block, and the initial reward was set to 50 BTC. Currently, its value is about 500000 USD.
- **Fee Reward :** It is the transaction fee rewarded to the miner.

2.2.1 Unspent Transaction Output (UTXO)

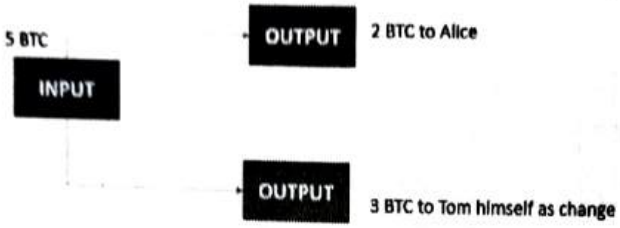
GQ. Write a short note on UTXO.

- UTXO is a transaction model that is used in bitcoin.

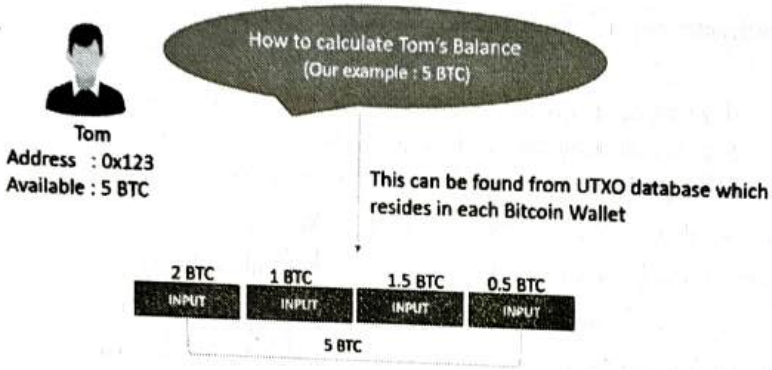
What happens when a bitcoin transaction happens?



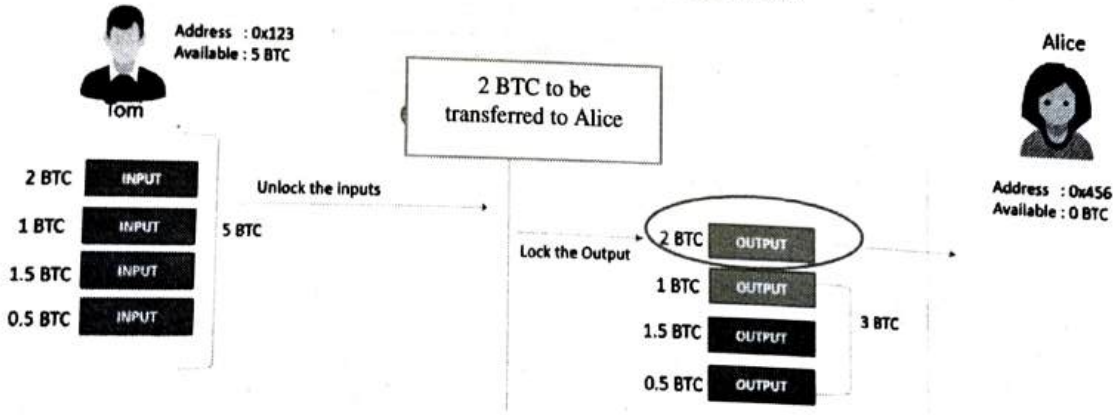
- From the figure, it is evident that Tom and Alice have their own addresses wherein Tom has 5 BTC and Alice has 0 BTC.
- Now, Tom sends 2 BTC to Alice, which becomes a transaction.



- In bitcoin world, the input is 5 BTC. The output is 2 BTC, which is planned to send to Alice by Tom. So, the remaining balance is 3 BTC, which has to remain with Tom.
- Herein, the protocol that is followed is that the remaining BTC (i.e., 3 BTC) would be sent back to Tom himself.



- Tom has 5 BTC and his address is 0x123. Now, it is essential to understand that how Tom has received 5 BTC. In bitcoin world, for every address, there will be corresponding inputs in the blockchain, which is stored in UTXO.
- For instance, 5 BTC received by Tom is the summation of 4 inputs (i.e., 2 BTC, 1 BTC, 1.5 BTC, and 0.5 BTC).
- So, when Tom opens his crypto wallet, these UTX inputs would be calculated and summed up together and then the balance for the same would be provided as the output and shown in the wallet.



- From the figure, it is evident that 2 BTC Tom is trying to send to Alice. Now, when Tom decides to send 2 BTC to Alice, it has to be from any one of the inputs (i.e., 2 BTC, 1 BTC, 1.5 BTC, or 0.5 BTC). In other words, Tom will have

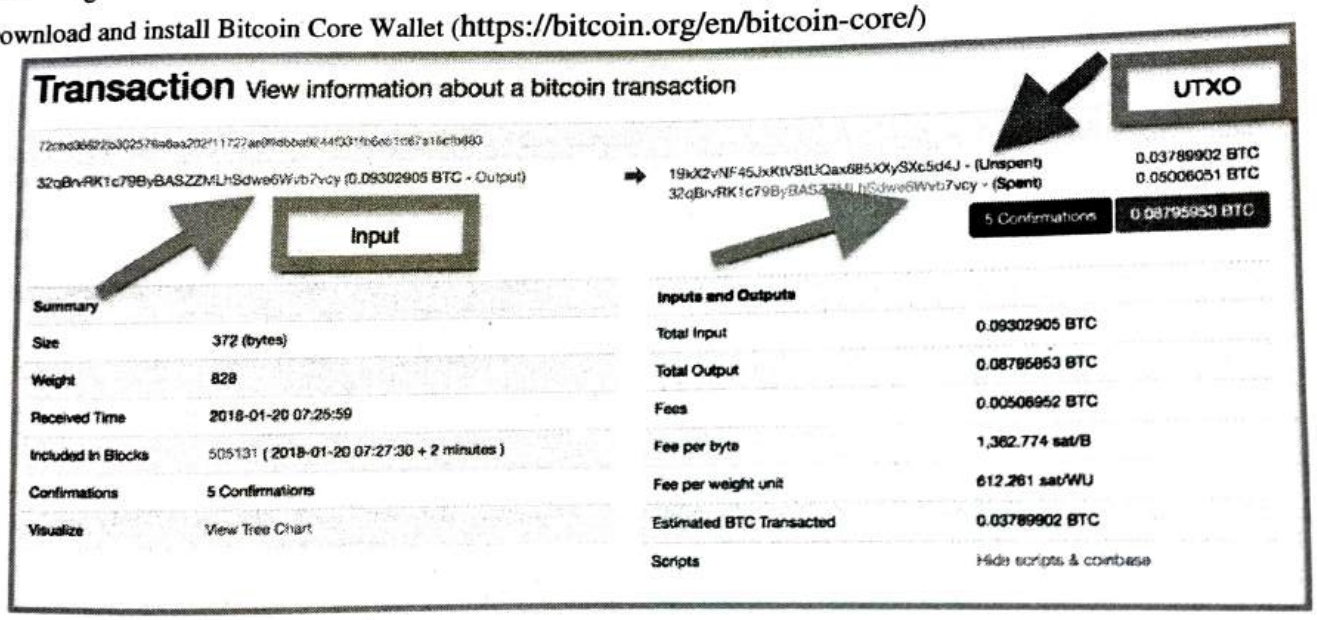
to unlock the inputs and lock the output (i.e., 2 BTC), and the remaining 3 BTC will have to be sent back to Tom. Finally, 2 BTC will go to Alice as a transaction. This is known as the UTXO model.

☞ Structure of a transaction

Version	Input counter	Inputs	Output counter	Ouputs	Lock timestamp

- Thus, a structure of a transaction includes:
  - Version Number
  - Input counter *(i.e., how many inputs are used to send 2 BTC)*
  - Inputs *(i.e., Tom has to prove that he owns 5 BTC)*
  - Output counter *(i.e., how many people is Tom trying to send. Herein, it is just one person)*
  - Outputs *(i.e., the amount that Tom is trying to send, which is 2 BTC)*
  - Lock Timestamp *(i.e., it specifies at what time the transaction is getting initiated)*
- In this way, a transaction gets initiated and then propagated to several nodes (miners). The miners then take these transactions and verify whether Tom actually possesses 5 BTC or not or was there a double spending performed by him. Such things will be validated as part of mining.

Download and install Bitcoin Core Wallet (https://bitcoin.org/en/bitcoin-core/)



Sample structure of UTXO in Bitcoin

⏭ 2.3 PUBLIC BLOCKCHAIN SYSTEM

GQ. What is a public blockchain? State some characteristics of public blockchain.

2.3.1 Introduction

Blockchain is one of the top ten strategic technologies of 2019. As per International Data Corporation (IDC), global expenditure on blockchain solutions is anticipated to reach 9.7 billion dollars in 2021. Blockchain can either be public or private based on whether a network is open or accessible to anyone with an Internet connection. In public blockchain,

anyone can join a network, and the users in a private blockchain are unidentified. Blockchain users can download a copy of the ledger, initiate, broadcast, or mine blocks.

☞ Public Blockchain

In a public blockchain, one doesn't need any permission to initiate or read/access a transaction, or participate in a consensus process in order to create a block. There is anonymity maintained in a public blockchain with the help of high cryptographic protocols.

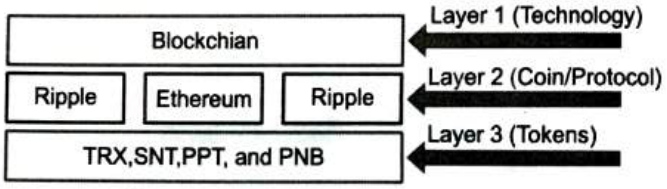
Some of the main characteristics of a public blockchain are as follows :

- **Type of organization :** Public
- **Approach :** Open and visible to all
- **Operations :** Anyone can read, initiate, or receive transactions
- **Immutability :** Secured via hashing
- **Trust :** It is a trust-free system
- **Users :** Anonymous
- **Type of network :** Decentralized
- **Verification :** Any node can participate in the consensus process to validate transactions and create a block
- **Speed of transactions :** Slow
- **Energy consumption :** Very high
- **Scalability :** It has limited scalability, which mainly depends on the growth of the network (i.e., the number of nodes in a network). Other factors could be bandwidth, storage, and an exponential increase in computational power.

2.3.2 Blockchain layers

GQ. Explain the relationship between different blockchain layers.

Relationship between blockchain, cryptocurrency, protocols, and tokens can be comprehended with the help of different layers in a blockchain.



(1B1)**Fig. 2.3.1 : Blockchain layers**

- **Layer 1 :** Blockchain is the technology at the back of a variety of cryptocurrencies such as Bitcoin, Ethereum, Ripple, etc.
- **Layer 2 :** This layer not only defines several coins but also focuses on the protocol of the respective currencies. A protocol is a set of rules that permit people to communicate or transact data with each other to not only reach consensus but also validate transactions within a given network. The protocols used are HTTP, HTTPS, and TCP/IP. Protocols in a blockchain define various consensus and signature mechanisms, use of public keys, and rules related to cryptography.
- **Layer 3 :** It comprises tokens. Tokens serve as the symbol for an underlying contract. For instance, Ethereum has 100s of tokens (i.e., TRX, SNT, PPT, and PNB). Bitcoin and Ripple have no tokens, and hence, they are unable to create smart contracts.

2.3.3 Popular Public Blockchain

- The two famous public blockchain are Ethereum and Bitcoin. Bitcoin the first blockchain created is a public permissionless blockchain. It was mainly designed to create a decentralized digital currency that is free from government regulations where individuals can confidentially trade with each other without any intermediaries. People became aware of blockchain only because of bitcoin. Bitcoin is a cryptocurrency that is used in many countries, and there is a possibility that bitcoin might determine the economic future of many nations.
- It is the fundamental form of cryptocurrency and first of its kind. After bitcoin, the second largest cryptocurrency is Ethereum, which was launched in 2015. Services such as voting, records of real-estate transfer, and social networking

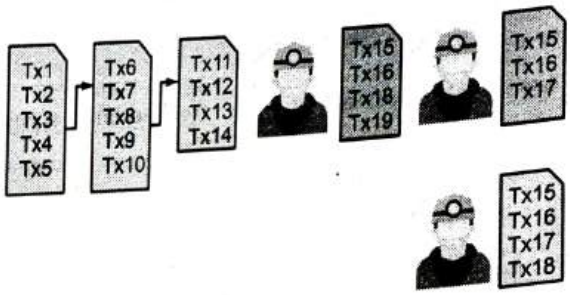
sites are based on dedicated servers that control most of the data we upload. Blockchain technology can be used to decentralize such services. Ethereum permits the creation of decentralized and open-source solutions for sectors like crowdsourcing and voting for democracy.

2.3.4 The Bitcoin Blockchain

GQ. Elaborate on the concept of "Consensus in Bitcoin".

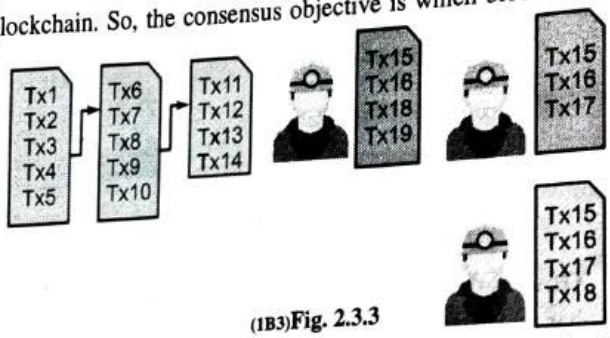
Consensus in bitcoin

- The objective of the consensus algorithm in bitcoin is to add a new block to an existing blockchain. Now, there could be multiple miners in a bitcoin network and individual miners can propose their new block based on a transaction that they have heard.
- So, one miner hears the transaction Tx15, Tx16, Tx18, and Tx19, whereas another miner hears the transaction Tx15, Tx16, and Tx17. Note that it is not necessary that every miner will propose the same block.
- It is like whatever transactions the miner has heard of till now (i.e., the time where the last block has been added in the blockchain), the miners can include all those transactions in the new block provided the total size of the block doesn't exceed certain threshold.



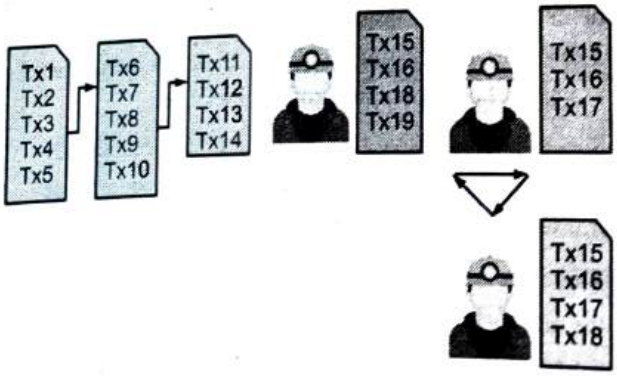
(1B2)Fig. 2.3.2

- There are three miners. They have three individual blocks, which have been proposed by them, and the objective here is to decide which block needs to be added to an existing blockchain. So, the consensus objective is which block do we add next out of the three blocks proposed by the miner.
- The challenge here is that the miners do not know each other because it is an open and a permissionless network, and thus, anyone can participate in the network as a miner and they can collect the transactions coming from the clients and can propose a new block. Under such a scenario, we need to ensure that the miners come under a common consensus.



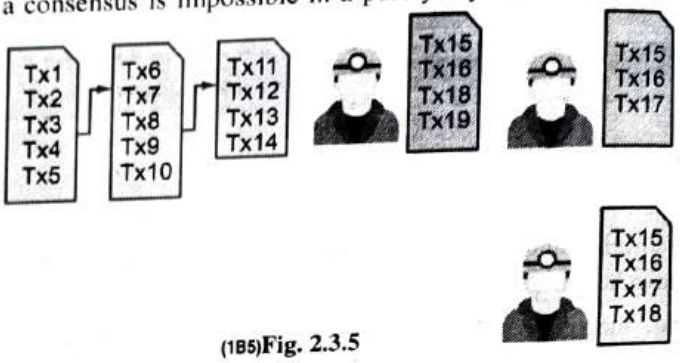
(1B3)Fig. 2.3.3

- One of the possible solutions that is closer to the traditional distributed consensus algorithm where every miner who is proposing new blocks, they will broadcast the information in the network and wait for a certain amount of time and see whether they are receiving any other blocks from other miners and based on that they will apply a choice function to select one of the blocks.
- However, there exists a problem in this mechanism. The problem here is that when we say that a miner will wait for a certain duration, it is technically infeasible under such an environment. The reason is that the miners do not know each other and they are unaware how many other miners are there in the network who will be proposing a new block.



(1B4)Fig. 2.3.4

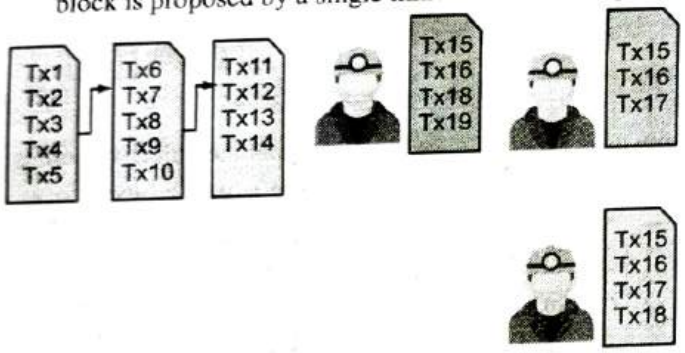
- The other reason is that the internet is inherently asynchronous, which means that whenever a miner will propose a block, there is a possibility of an arbitrary delay in transmitting new blocks. So, in such an environment, having a consensus based on a broadcasting algorithm by applying traditional distributed system mechanism may not be feasible.
- The impossibility result/theorem states that coming to a consensus is impossible in a purely asynchronous distributed network in the presence of a single failure.
- In this particular architecture, even if there is a miner who is a malicious miner then other miners may not be able to collect the blocks that are coming from legitimate miners and having a consensus by applying such type of traditional distributed algorithm based on message passing or broadcasting is infeasible in this scenario.



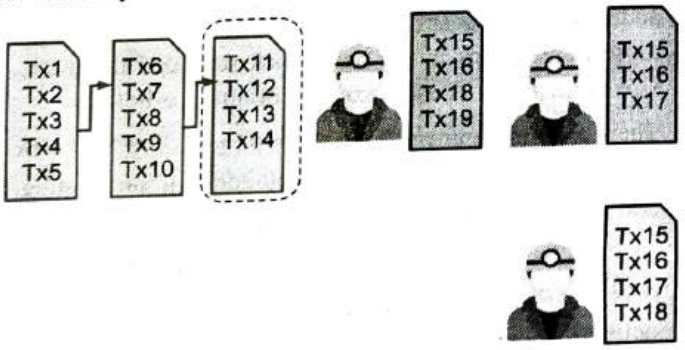
(1B5)**Fig. 2.3.5**

There are two observation from the above mentioned scenario

- **Observation 1 :** In case of a blockchain environment, any valid block can be accepted. A valid block means a block with all valid transactions. So, we can accept any of the proposed blocks if it is valid. There could be a possibility that a block is proposed by a single miner and not multiple miners. The only condition is it should be valid.

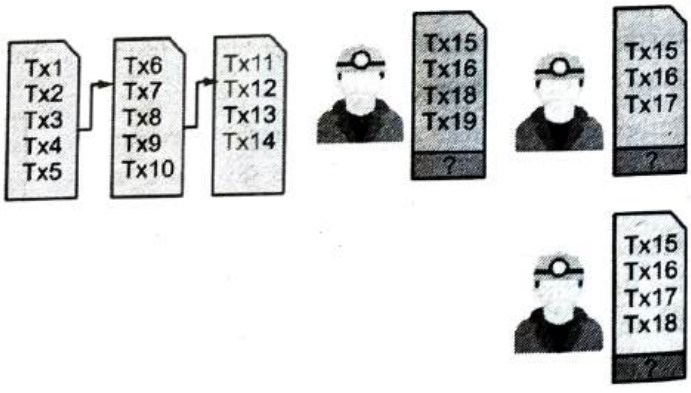


(1B6)**Fig. 2.3.6**



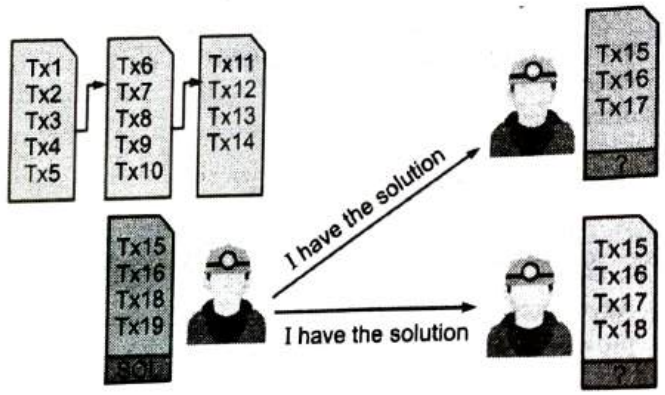
(1B7)**Fig. 2.3.7**

- **Observation 2 :** The entire protocol can run in rounds. It means that once a valid block has been accepted/the system has reached consensus from that point a round starts, the miners start getting the transactions and then they can find out that which are the transactions that are not already committed in the blockchain and based on that they can propose the new block.
- In the above Fig. 2.3.7, the last block in the blockchain has transactions Tx11, Tx12, Tx13, and Tx14. So, all the miners know that this particular block has been committed in the blockchain.
- Since they know that this transaction has been already committed, so there is no need to include those transactions again in a new block.
- Therefore, they can propose a new block by excluding the transactions that have already been committed in the blockchain.



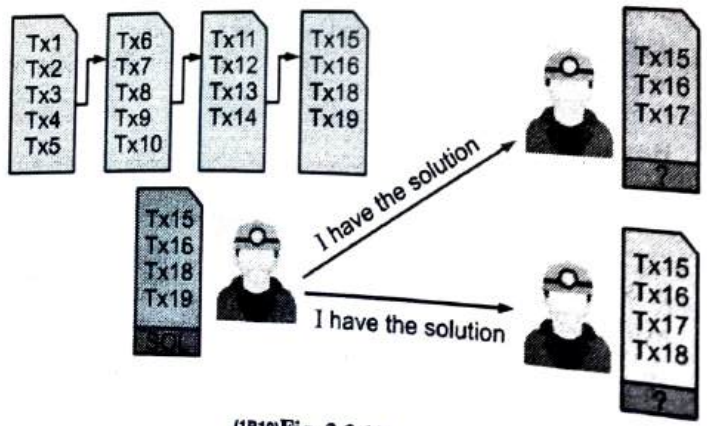
(1B8)**Fig. 2.3.8**

- So, in this way, once a particular commitment has been done (i.e., once a block is added to a particular blockchain), the miners can try to figure out which are the next transactions that need to be added to the blockchain and they can propose the new block.
- Based on the two observations, there is a possibility of designing a solution in the following manner where every miner tires to independently solve a challenge. The challenge is provided by the network. The miners will independently try to solve a challenge, and the block is accepted for the miner who can *prove first* that the challenge has been solved.
- The keyword *prove first* is important here because it implies that every individual miners are trying to obtain a solution for a challenge that is imposed by the network. The miner who is able to solve the problem/challenge first, they add the solution of the problem to the existing block and add that block to the blockchain.
- Such type of an algorithm can be called as *proof-of-work* algorithm. The term proof comes from the fact that the miner who is proposing the solution, he/she was able to prove that he/she has found out the solution to the problem.
- So, when a miner identifies that there is a solution, he/she proposes the block to the other miners stating that he/she has already received the solution, and because of this other miners stop finding solution to this challenge/problem and start finding new blocks from the transactions that are coming from the clients. Note that this particular communication can work asynchronously.
- Whenever a miner is advertising a solution to other miners in the network, the miners can work for mining the previous block during that time. Say, if we consider a miner who is trying to find out the solution for a given challenge and when the miner identifies that a new block has been added in the blockchain during that time if the miner finds out that the newly added block has certain set of transactions that are a part of his/her block, which implies that there is an overlapping of transactions between the block that the miner is trying to mine and the newly added block in the blockchain.



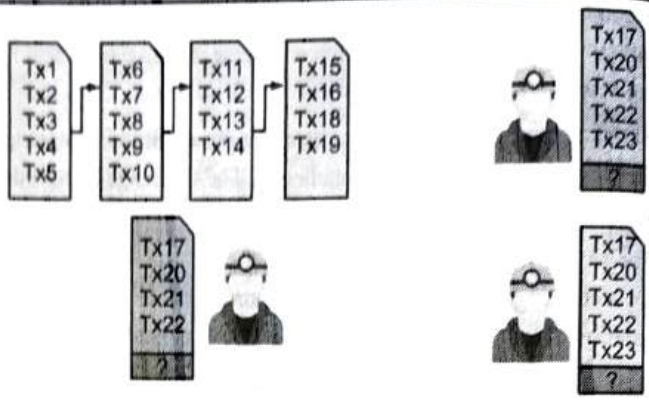
(1B9)**Fig. 2.3.9**

- In such cases, the miner can stop mining at that point, accept the most updated blockchain, and then start finding out blocks with new transactions that are coming from the clients. Once the solution has been obtained, then the block is added to the existing blockchain. Whenever the updated blockchain is propagated to all the miners in the network, the other miners were also trying to mine the same block where Tx18 and Tx19 were there.
- So, from the Fig. 2.3.10 it is evident that one of the miners was trying to mine a block where Tx15, Tx16, and Tx17 were there, and the second miner was also trying to mine a block where Tx15, Tx16, Tx17, and Tx18 were there.



(1B10)**Fig. 2.3.10**

- Now the new block gets added to the blockchain and the miners can find out that in the newly added block Tx15, Tx16, Tx18, and Tx19 are there (i.e., committed), but Tx17 has not be included in the block (i.e., not committed).
- So, in the next round every miner can include Tx17 and at the same time the other transactions that they have heard within a time duration can be added, and the miners can begin mining those specific blocks.



(1B11)**Fig. 2.3.11**

- Thus, the same process repeats in rounds and the miners can try to find out what can be the solution for a particular challenge. In order to generate the challenge, bitcoin relies on a mechanism called proof-of-work (PoW). PoW is an economic measure to deter service abuses by requiring some work from a service requester (usually processing time by a computer).
- Say, if an individual is requesting for a service, then the individual has to show to the service provider that the individual has spent significant interest on the service that he/she is requesting for. The idea came from Dwork and Naor (1992) to combat junk/spam emails. Some work needs to be done to send a valid email. An attacker would be discouraged to send junk/spam emails.

☞ Proof-of-work (PoW) features

GQ. What is proof-of-work (PoW)? How cryptographic hash can act as a good indicator for PoW ?

- **Asymmetry :** The work must be moderately hard but feasible for a service requester. On the other hand, at the service provider side, the work must be each to check/validate.
- This is how service requesters will get discouraged to forge the work, but service providers can easily check the validity of the work.

☞ How cryptographic hash can act as a good indicator for PoW?

- The puzzle friendliness property of a cryptographic hash function states that given X and Y, determine k such that $Y = Hash(X \parallel k)$. So, if X and Y are known, then it is very difficult to find k. Some kind of brute force/random mechanism would be needed to find k. However, once k is found, then one can easily verify the challenge.
- Such concept was used in hashcash, which was proposed by Adam Back (2002). It was a PoW mechanism that can be added with an email as a good will token. In other words, it means that only legitimate email senders will be interested in generating the hashcash, whereas spam email generators will be discouraged to generate the hashcash.

☞ Hashcash PoW

GQ. What do you mean by Hashcash PoW ? Explain with an example.

- It is a textual encoding of a hashcash stamp that is included in an email header. This acts as a proof that a sender has expended a modest amount of CPU time calculating the stamp before sending an email. It is doubtful that the sender is a spammer.
- Conversely, an email receiver can verify the hashcash easily by applying certain hash function. But any change in the header also requires changes in the hashcash because the hashcash is generated from the header. So, if an individual wants to modify certain fields inside an email header, the hashcash needs to be regenerated where brute force is the only way to find a new hashcash. This is how an attacker will be discouraged to tamper the email header.

☞ Example of a hashcash at the sender's side

The hashcash that is specified in an email header is as follows :

X-Hashcash

1:20:210827: vivian27@gmail.com::0000000272727v270891rsv:2228

- 1 indicates the version number (i.e., hashcash version 1). 20 represents the number of zero bits that need to be present in the hashed code (i.e., a hash function needs to be generated where the 1st twenty bits need to be zero). So, this is the challenge which is given to an email sender.
- The email sender has to find out the hash value where the 1st twenty bits are zero. Next, the date is specified in YYMMDD format followed by an email ID (resource). Then a random number (a string of random characters) is included in the hashcash followed by a counter value.
- The counter value acts like a nonce. For an email sender, all the fields are fixed. However, the only field that can be changed by the email sender is the counter field. Thus, the email sender needs to check by providing different counter value that from which counter value it can find the expected hash value where the objective is to have 20 zeros at the prefix.
- In the sender side, the hashcash is constructed in the following manner:

1:20:210827:vivian27@gmail.com::<hash>:<counter>

- After adding individual fields in the header, the sender initializes the counter value to a random number. Once the sender has initialized the counter value to a random number, then the sender computes the hash of the header using 160 bit SHA-1 algorithm. If the 1st 20 bits of the hash are all zeros, then the hash is accepted else the counter value needs to be changed.

☞ Example of a hashcash at the receiver's side

- The recipient needs to check that the date should be within two days as per hashcash version 1. So, if an email spammer/attacker is changing the date, then he/she needs to recompute the hashcash. It also checks the email address (i.e., the email address in the hashcash and the email address in the header are same or not).
- The recipient also needs to check the random string, which should not be used repeatedly within a certain duration so that replay attack can be prevented (i.e., the attacker has cached the previous hashcash and by caching the previous hashcash with every junk email he/she is sending the previous hashcash). So, the random string helps in preventing replay attack where every time an email is sent, there is a need to change the ransom string.
- Once all the fields in the hashcash have been check, then it is necessary to validate the hashcash, which can be done by using 160 bit SHA-1 algorithm and if it is possible to find that the first 20 bits are zeros, then only the email is accepted else it is discarded.
- Analysis of hashcash PoW
- Because 160 bit SHA-1 hash function is being utilized, we can have 2^{160} possible hashcash values. Since we require 20 bits in the beginning, 2^{140} possible hash values would be remaining, which satisfy the criteria that the 1st 20 bits are zeros and the remaining 140 bits can either be zero or one.
- So, the chance of randomly selecting a header with 20 zero bits at the prefix is 1 in 2^{20}. This means that if one can try with 2^{20} different hash functions with a high probability, then he/she can get a hash value. On the other hand, the recipient would require approximately 2 microsecond to validate a hashcash.

Exercise

Visit www.hashcash.org

- Download the source and try with different number of zero bit targets. Increase the number of targeted zero bits at the hash prefix, say from 20 to 2020, at a step of 100, and observe the time to compute the hash.
- Use **shalsum** (in Linux) to compute the SHA-1 checksum of the obtained hashcash values from the above exercise. Observe the amount of time that was needed to validate a hashcash.

Bitcoin PoW system OR Nakamoto consensus system

GQ. Discuss about bitcoin PoW system.

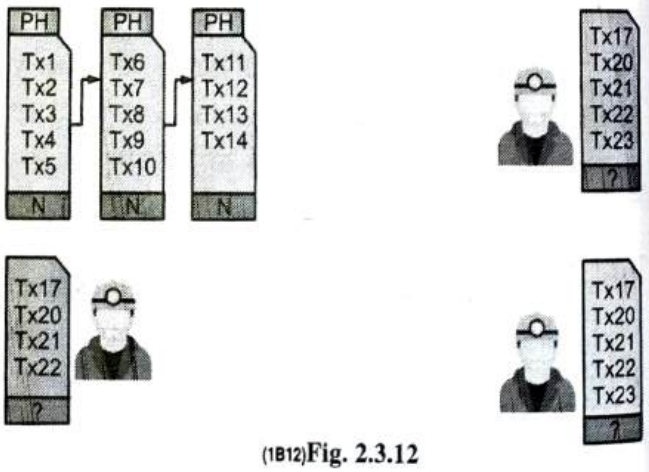
- The bitcoin PoW system is based on the hashcash PoW system. So, the miners who are the special nodes in a bitcoin network who participate in the consensus procedure, they need to give a proof that they have done some work before proposing a new block. If the miners can successfully complete that work, then they are able to submit the block as a part of the existing longest chain of the blockchain.
- However, the attacker will be discouraged to propose a new block or make a change in the existing block because they will have to do the entire work of the blockchain, which is computationally difficult in a generic environment.

Methodology for bitcoin PoW system

- In bitcoin PoW system, there exists a blockchain. According to the architecture of blockchain, every block is connected to the previous block with a hash value.
- PH represents the hash value of the previous block. There exists a nonce value (N), which is present in every block. We need to include one of the three existing blocks that are proposed in the blockchain. So, every miner will try to find a nonce value, which will satisfy certain hash equation, as shown below.

  BH = Hash(PH:MR:N)

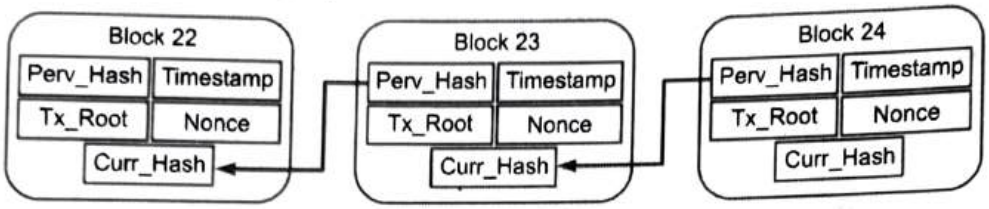
- PH is the previous block hash along with the Merkle root (MR) of a transaction. Note that all transactions are arranged in the form of a Merkle tree, which is a hash-based architecture, and the root of the Merkle tree contains the root hash, which is known as the Merkle root. N represents the nonce value.
- Block hash (BH) value has been given as a challenge. The challenge is that BH should have certain zeros (difficulty) at the beginning/prefix. The miners will try with various values of nonce to find out the block hash value for a certain difficulty level (i.e., certain number of zeros at the prefix of the hash value).
- Every individual miner will try to find out the nonce value, and the miner who will be able to find out the nonce value for his/her own block, then he/she will be able to include the block as a part of the existing blockchain.



(1B12)**Fig. 2.3.12**

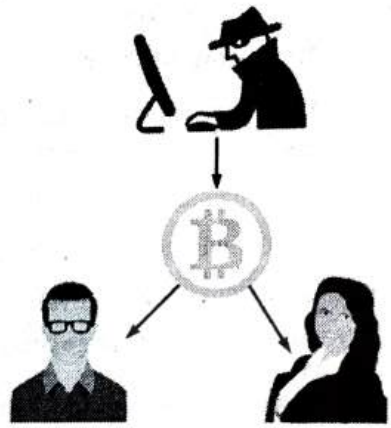
- This is how the PoW system works in bitcoin to ensure consensus by utilizing a challenge response-based system where the challenge is that the nonce value has to be obtained for a block hash with a certain difficulty level.
- Most implementations of bitcoin PoW use double SHA256 Hash function. The miners collect transactions for approximately 10 minutes (default setup) and start mining the PoW. The probability of getting a PoW is low, i.e., it is difficult to interpret which miner will be able to generate the block. No miners will be able to handle the bitcoin network single handedly.

☞ Why bitcoin PoW system is tamperproof ?



(1B13)**Fig. 2.3.13**

- Fig. 2.3.13 shows the broad architecture of a block. As we know that every individual node needs to find out the nonce so that the Curr_Hash [Block 24] satisfies certain difficulty (i.e., the minimum number of zeros should be there in the prefix). The Curr_Hash value of one block is included as a Prev_Hash value of the other block. If an attacker wants to make some changes in one block, then they have to perform more work compared to the collective work of all the blocks in the blockchain.
- So, every block in the chain has been obtained by doing some work by individual miners where the miners have obtained the nonce value based on the difficulty level. Thus, the attacker will have to do more work to tamper the blockchain, which is difficult with the current hardware.



(1B14)**Fig. 2.3.14**

☞ PoW solves double spending problem

GQ. How PoW solves the problem of double spending ?

- Double spending problem can sometimes work as an attack too. It nothing but the successful use of the same fund twice. An attacker is trying to transfer the same bitcoin to two different persons almost at the same instance of time. For instance, a transaction is simultaneously generated with BTC30 to both Person A and Person B.
- Bitcoin by using the PoW mechanism tries to solve the double spending problem. The transactions that are entered in the block are irreversible (computationally impractical to modify), and every transaction can be validated against the transactions that are already existing in the blockchain.
- There can be certain attacks that could happen on a bitcoin PoW system such as Sybil attacks.

GQ. Explain Sybil attack and denial of service (DoS) attack.

- **Sybil attacks :** Here, an attacker attempts to fill a network with clients under his/her control. If the attacker can fill the network with clients under his/her control, then the attacker can actually control or get a monopoly over the network and such clients can perform different kinds of actions based on the instructions received from an attacker such as they can refuse to relay valid blocks, they can only relay the blocks that are generated by the attackers, which could lead to double spending. Thus, in a Sybil attack, the attacker can include multiple such nodes in the network who can collectively compromise the PoW mechanism.
- **Possible solution :** To diversify the connections (i.e., bitcoin allows outbound connection to one IP per/16 (a.b.0.0) IP address). For instance, if you have an IP address series of 192.16.10.0/16, then in this entire network,one can have atmost one pair. If the network is diversified and if the attacker generates multiple false miners, the attacker will generate then within the same network so all the false miners will be clustered within the same subnet.

- **Denial of service (DoS) attack :** Here, an attacker sends a lot of data to a particular node and because of this, the node will not be able to process normal bitcoin transactions.
- **Possible solutions :**
  - Do not forward orphaned blocks.
  - Do not forward double spend transactions.
  - Do not forward same block or transactions.
  - Disconnect peers who send too many messages/transactions.
  - Restrict the block size to 1 MB.
  - Restrict the size of bitcoin script up to ten thousand bytes.

☞ Breaking bitcoin PoW

Bitcoin PoW is computationally difficult to break, but not impossible. Attackers can deploy high power servers to do more work than the total work of the blockchain.

☞ Monopoly problem in bitcoin PoW system

PoW depends on computing resources that are made available to a miner. Miners that have more resources are highly probable to complete a specific work and can gradually control the entire network.

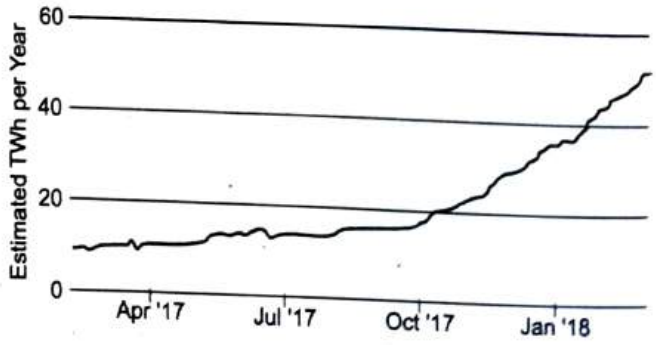
However, monopoly can increase over time.

- Miners will get less reward over time.
- Users will get discouraged to join as a miner.
- Few miners with large computing resources may get control over a network.

☞ PoW power consumption

GQ. Why was proof-of-stake (PoS) introduced ?

- Another limitation of the PoW system is power consumption. The PoW system relies on the amount of power, which is being consumed. Whenever a huge amount of computational resources have been deployed in the network, it is expected that those computational resources are making using of a huge amount of power to generate hash functions.
- The following graph shows the amount of power consumptions with the bitcoin network. An exponential growth is seen in the power consumption due to bitcoin mining.
- In order to handle monopoly and power consumption problems, proof-of-stake (PoS) was introduced.
- PoS was proposed in 2011 by a member in bitcoin forum. The general idea was to make a transition from a PoW-based system to a PoS-based system when bitcoins are being widely distributed.
- Difference between PoW and PoS



(1B15)Fig. 2.3.15

Sr. No.	PoW	PoS
1.	Probability of mining a block depends on the work done by a miner (i.e., if a miner has huge computing resources)	Amount of bitcoin that a miner holds, i.e., if a miner holds 1% of the bitcoin, then he/she can mine 1% of the PoS blocks.

PoS-based system offers increased protection, which means that executing an attack is expensive (i.e., more bitcoins are needed). Moreover, there is reduced incentive for an attack because for generating an attack a huge amount of bitcoins are needed, and spending those bitcoins to perform an attack will result in loss (if unsuccessful), which in turn will affect an attacker. Conclusively, this means that attacks are more expensive in a PoS-based system.

☞ Proof-of-burn (PoB)

GQ. What is proof-of-burn (PoB)

- PoB is another consensus mechanism where miners should show some proof that they have burned some coins. This means that miners have to send these coins to a verifiably unspendable address. Note that PoB is as expensive as PoW, but no external resources are used other than burned coins.
- In PoB, virtual or digital resources need to be spent, whereas in PoW, real resources are needed. PoB works by burning PoW mined cryptocurrencies. Also, PoB is power efficient as physical hardware is not used and digital currencies are used.

☞ Difference between PoW, PoS, and PoB

GQ. Differentiate between PoW, PoS, and PoB.

Sr. No.	PoW	PoS	PoB
1.	Do some work to mine a new block	Obtains sufficient stake to mine a new block	Burn some bitcoins to mine a new block
2.	Consumes physical resources (i.e., CPU power and time)	No external resources are consumed, but participate in transactions	Consumes virtual or digital resources
3.	Power hungry	Power efficient	Power efficient

☞ Proof of elapsed time (PoET)

GQ. Explain the concept of proof of elapsed time (PoET).

- It is another consensus algorithm that was proposed by Intel as a part of Hyperledger Sawtooth, which is a blockchain platform for building distributed ledger applications.
- In PoET, each participant in a blockchain network waits for a random amount of time. The first participant to finish becomes the leader for the new block. In other words, a randomization amongst miners is made. So, the miner who will be able to complete the random waiting first, he/she will be allowed to propose the new block.
- *However, the challenge is the how will one verify that the proposer has really waited for a random amount of time?*
- For verification, Intel uses a special CPU instruction set, i.e., Intel Software Guard Extension (SGX), which is a trusted execution platform. Moreover, the trusted code is kept private to the rest of the application. Also, the specialized hardware provides an attestation that the trusted code has been set up correctly.

☞ Life of a miner

GQ. Explain the procedure of mining bitcoins by a miner.

- The task of a miner is to validate transactions and construct a new block. Once they have constructed a new block, then they employ their hash power to vote on consensus (i.e., to identify who will be finishing the work first and propose the block as a new block) and commit the transactions with a new block and add the new block to an existing blockchain, and then store and broadcast the new blockchain to the peers. So, this is now the entire blockchain gets propagated in the entire network.
- The entire procedure of mining bitcoins has six steps :
  1. Join the bitcoin network and listen for transactions and validate the proposed transactions that are coming from the client.
  2. To listen for new blocks, i.e., validate new blocks and re-broadcast a new block when it is proposed.
  3. Collect the transactions for a pre-defined time and construct a new block.
  4. The miner needs to participate in the mining procedure where nonce value needs to be obtained to make a new block valid.
  5. Broadcast the new block to the peers. If everyone accepts it, then it is a part of the main chain.
  6. Earn a reward for participating in the mining procedure.

☞ Mining difficulty

GQ. How is the difficulty level of a block computed ?

- It is defined as a measure of how difficult it is to find a hash below a given target. A bitcoin network has a global block difficulty. The difficulty level changes for every 2016 blocks.
- The desired rate is that one block should be generated for every 10 minutes. Thus, it will take two weeks to generate 2016 blocks. The difficulty level is readjusted every two weeks. Note that the change in difficulty is in proportion to the amount of time over or under two weeks the previous 2016 blocks took to find.
- For every two weeks, the difficulty level is computed as follows :

$$\text{Current difficulty} = \text{Previous difficulty} * \frac{\text{(Two weeks in milliseconds)}}{\text{(Milliseconds to mine last 2016 blocks)}}$$

- The same formula can be applied after 2016 blocks have been generated. If less than two weeks' time is taken to generate 2016 blocks, then current difficulty needs to be increased.
- On the other hand, if more than two weeks' time is taken to generate 2016 blocks, then current difficulty needs to be reduced. This is how the bitcoin network dynamically changes the difficulty level.

☞ How difficulty level is related to the hash rate?

GQ. How difficulty level is related to the hash rate ?

- The hash is a random number between 0 and -1. So, to find a block, the hash must be less than a given target.
- The offset for difficulty level 1 is $0xffff * 2^{208}$, which means that out of the 256 blocks the initial 48 bits should be zero and the remaining bits need to be one or zero.

- Accordingly, the offset for difficulty D is $\frac{0xffff * 2^{208}}{D}$. The expected number of hashes we need to calculate to find a block with difficulty D is $\frac{D * 2^{256}}{0xffff * 2^{208}}$.
- The difficulty level dynamically changes the hash rate. If more difficulty is provided, then more number of hashes need to be produced to obtain the resultant target.

☞ Mining pool

GQ. What is a mining pool? State its methods along with advantages and disadvantages.

- The idea of a mining pool is that several miners are combined together, i.e., the resources available from multiple miners are combined and then the miners are asked to generate the hash in a distributed way.
- So, every miner will begin the generation of hash for a block and will share their processing power over the network to mine a new block. Moreover, the reward is split proportionally to the amount of work that is done by each and every miner.
- In a mining pool, there could be hundreds or thousands of miners who communicate with each other through some special protocols.
- Let us assume that B is the block reward minus pool fee and p is the probability of finding a block in a shared attempt $\left(p = \frac{1}{D}\right)$ where D is the difficulty level of the block.

☞ Mining pool methods

1. **Pay per share :** It represents an instant and guaranteed payout to a miner. Miners are paid from a pool's existing balance. Share of a miner is $R = B \times p$. Miners receive almost equal payment and the risk lies in the hands of the pool operator.
2. **Proportional :** Here, miners earn share until the pool finds a block (i.e., end of a mining round). $R = B \times \frac{n}{N}$ where n is the amount of a miner's own share, whereas N is the amount of all shares in the round. Here, Payments are made once a pool finds out a block
3. **Pay per last N share :** It is similar to proportional. Here, a miner's reward is calculated on the basis of N last shares. In this method, miners obtain more profit for a short round.

☞ Advantages of mining pools

1. Small miners can participate
2. Predictable mining

☞ Disadvantages of mining pools

1. Leads to centralization
2. Discourages miners for running a complete mining procedure

Chapter Ends...

❑❑❑

Module 3

Programming for Blockchain

Syllabus

3.1 Introduction to Smart Contracts, Types of Smart Contracts, Structure of a Smart Contract, Smart Contract Approaches, Limitations of Smart Contracts

3.2 Introduction to Programming: Solidity Programming – Basics, functions, Visibility and Activity Qualifiers, Address and Address Payable, Bytes and Enums, Arrays-Fixed and Dynamic Arrays, Special Arrays-Bytes and strings, Struct, Mapping, Inheritance, Error handling

3.3 Case Study – Voting Contract App, Preparing for smart contract development **(Self-study)**

3.1 SMART CONTRACTS

GQ. What is a smart contract ? Explain its characteristics.

3.1.1 Introduction

- The term was coined by Nick Szabo (a computer scientist and cryptographer) in 1996. Szabo claimed that smart contacts (i.e., whenever we are establishing smart contracts between multiple parties) can be realized with the help of a public ledger. Thus, blockchain can also be a pioneering technology to realize smart contracts.
- A smart contract basically provides a decentralized platform which can be utilized to avoid the intermediaries (i.e., middleman) in a contract. When two persons are coming to a contract, there is a legal advisor who is controling such type of a contract. So, with the help of such type of blockchain environment, intermediaries/middleman can be avoided.

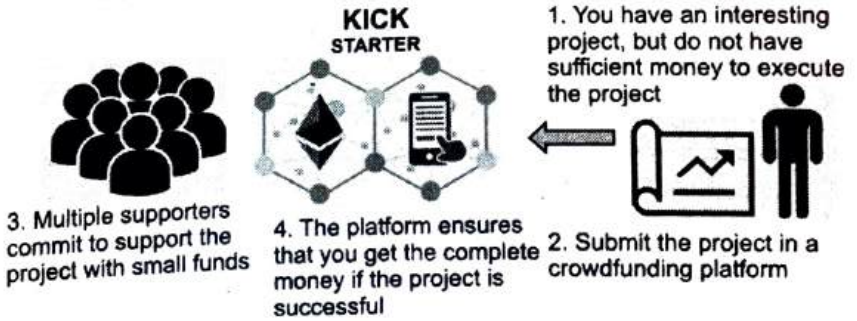
3.1.2 Smart Contract

Definition : A smart contract is an automated computerized protocol used for digitally facilitating, verifying, or enforcing the negotiation or performance of a legal contract by avoiding intermediates and directly validating the contract over a decentralized platform.

Contracts in a centralized platform by means of crowdfunding

GQ. How crowdfunding platforms can be managed using smart contracts ?

- Kickstarter is a crowd funding company the uses the following principle. For instance, you (or a group of people) want to execute some kind of an interesting project but there is no sufficient amount of money or fund. An individual or a group of people what they do is they submit the project to a crowd funding company like Kickstarter to acquire funds. Now, there are multiple supporters who can support with some small funds for a particular project.
- The project that is submitted by an individual or a group of people could be supported by an individual (i.e., offering the entire fund) or a group of supporters by offering small funds. The task of Kickstarter platform is to ensure that when certain milestone of a project is achieved, an individual or a group of people is eligible to receive the fund.
- Kickstarter ensures that whenever a supporter is providing the fund, the fund is received to an intended project as and when the project completes certain milestone (i.e., project executers are getting the fund). However, if the project is not completed successfully or in between the project gets scrapped, then the fund is sent back to the supporter.



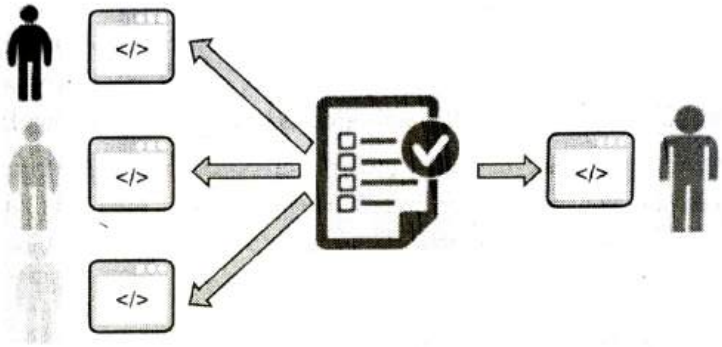
(1B20)**Fig. 3.1.1 : Crowdfunding**

- In this type of an architecture, the relationship of trust is essential on the crowdfunding platform between the product team and the supporters. The product team expects the money to be get paid based on the project progress (i.e., whenever a particular milestone for a project is reached Kickstarter should provide the fund).

- The supporters expect the money to go to the right project and if the project is scrapped in between, they will get back their money. Here, the crowdfunding platform (i.e., Kickstarter) acts as a middleman that takes a huge amount of money from the product team as well as the supporters to manage the entire process*(as there is a huge amount of risk factor that is involved like if the project is left half completed or the supporter would not support the project further)*, which is indeed a problem in a centralized platform.

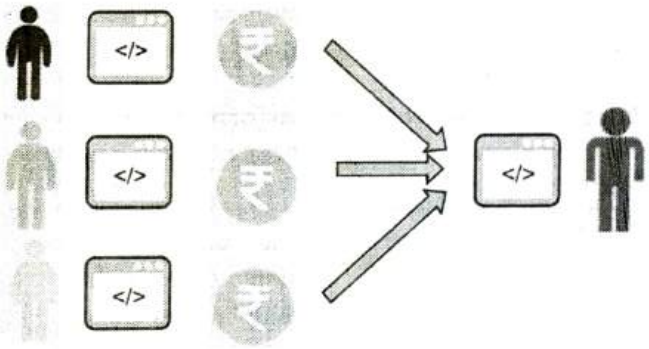
☞ How crowdfunding platforms can be managed using smart contracts ?

- There are a list of supporters on one side and on the other side there is a product team.
- The contract between the supporters and the product team is written in a code, which is available to all the stakeholders (i.e., to the supporters as well as the product team).



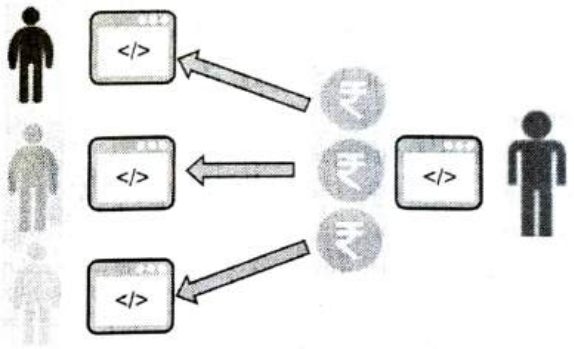
(1B21)**Fig. 3.1.2**

- Now, everyone can verify the contract. Also, if we put the contact inside a blockchain, then everyone will be able to verify that contract but no one will be able to tamper it.
- So, this gives an interesting idea that such type of a smart contract platform can be realized using a blockchain.



(1B22) **Fig. 3.1.3**

- If certain goals of the project are reached, then the code automatically transfers the money from supporters to the production team.
- If the project goals (contracts) fail, then the code can transfer the money back to the supporters.



(1B23) **Fig. 3.1.4**

- In this example, instead of placing a transaction or data inside the blockchain, we are placing the code which will be automatically verified by every stakeholders. They will not be able to tamper the code. They will not be able to deny the code in between, but as and when the code runs by verifying the events or actions that have been executed, the contract can get executed over time and fulfill the initial agreements that have been made.

☞ Characteristics of a Smart Contract

The characteristics of a smart contract are as follows :

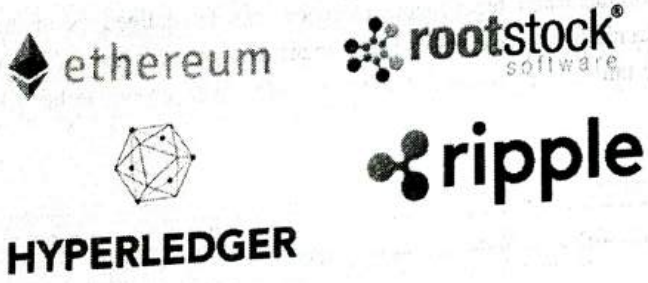
- **Immutable :** No party will be able to change the contract once it is fixed and written to the public ledger (i.e., the Blockchain).
- **Distributed :** There is no need of a middleman like Kickstarter who handles all the risk. The code will get automatically executed. If the promise is not fulfilled, then automatically the code will execute some steps based on the contracts. In other words,all the steps of the contract can be validated by every participating party, i.e., no one can claim later that the contract was not validated.

☞ Blockchain is a suitable platform for executing a smart contract because of the following reasons

- Based on the blockchain architecture, blocks are immutable.
- Information is open (i.e., everyone can check and validate the information inside a blockchain).

☞ Various types of smart contract platforms

GQ. Elaborate on different types of smart contracts.



(1B24)**Fig. 3.1.5 : Types of smart contract platforms**

☞ Types of Smart contracts

Smart contracts can be categorized into four types based on the applications :

- **Smart legal contracts :** These contracts possess several legal contract templates. They simply execute the contracts as per the templates used.
- **Decentralized applications :** They are also called as DApps. They run point-to-point network of computers instead of a single computer. It is not essential that DApps should run on top of a blockchain network.DApps are blockchain-enabled websites. On the other hand, a smart contract is what permits a DApp to connect it to a blockchain. DApp covers from end to end (i.e., both front end and back end). If a decentralized application on a smart contract system needs to be created, then several smart contracts must be combined and third-party systems have to be relied upon for the front-end.
- **Distributed autonomous organization (DAO) :** It is an organization exemplified by logic encoded as a transparent program controlled by shareholders and not influenced by a central authority. The financial transaction records and program logics of DAO are maintained on a blockchain. DAOs make use of blockchain technology to maintain a secure advanced record in the form of a digital ledger (DL) to trace financial transactions over the Internet. DAOs make use of distributed databases and is tamperproof as it involves timestamps, which excludes the presence of a trusted third-party in a financial transaction. DAO is an entity that is independently present on the Internet. However, it requires human intervention to perform certain tasks.

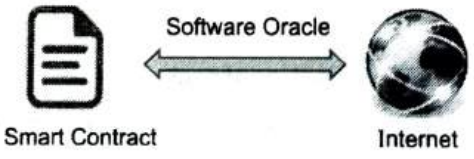
☞ Characteristics of Independent DAOs

- **Tokens of transactions :** DAOs require tokens that are used for rewarding certain activities. Funding occurs directly by a specific organization that creates the tokens.
- **Independent :** The code, once deployed, is not under the control of its creators and cannot be influenced by outsiders. Thus, DAOs are open source, which offer a transparent and indestructible system.
- **Consensus :** Most of the stakeholders must agree on the same decision for not only withdrawing but also moving funds from DAOs.
- **Contractors :** The task of building a product, writing a code, or developing a hardware do not lie in the hands of DAOs. They need to hire a contractor, and so, contractors get appointed through voting.
- **Proposals :** In DAOs, the principle way of taking decisions is through proposals. To avoid overloading of the network with proposals, DAOs require monetary deposit, which discourages people from spamming the network.
- **Voting :** Voting takes place only after submitting the proposal. DAOs permit people to exchange economic values with others through various activities like investing, money raising, lending, and borrowing without any intermediaries.
- **Smart contracting devices :** Any failure in IoT on the Internet will result in the leak of personal data. A security breach occurs primarily in authentication, connection, and transaction. With the help of blockchain that manages and access data from IoT devices, a hacker has to bypass an additional layer of security in addition to some robust encryption standards that are available. Moreover, a single point of failure cannot take place because of the decentralized nature.
- In the context of smart contracts and blockchain, an oracle can be defined as an agent that verifies real-world occurrences and submits details to a blockchain so that smart contracts can use it. For example, fund managers can release funds only when certain conditions are satisfied/met. Moreover, an oracle has to sign on a smart contract for releasing funds.

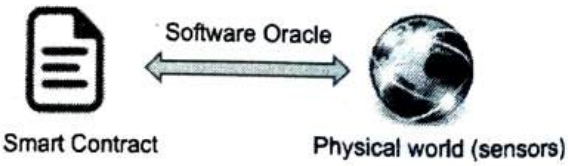
☞ Types of Oracles

GQ. What is an oracle? State and explain different types of oracles.

1. **Software Oracles :** They manage information that is commonly available on the Internet. For instance, flight details, commodity prices, etc. Such data is usually acquired from online sources (i.e., airways sites and e-commerce sites). Software oracles fetches relevant information and sends it to a smart contract.
2. **Hardware Oracles :** Certain smart contracts require input from the physical word (i.e., sensors), which is offered by hardware oracles. For instance, hardware oracles can be used to track a car's details, such as date, time, speed, and direction, when it is crossing a specific junction through sensors. Another example could be the use of RFID sensors for keeping a track of truck movements that carry goods.
3. **Inbound and Outbound Oracles :** Inbound oracles offer a smart contact with data from external world (i.e., the use of an automatic purchase order for an item in an inventory management system when that item's stock value hits a certain threshold value), whereas outbound oracles allows a smart contract to send data to the outside world.



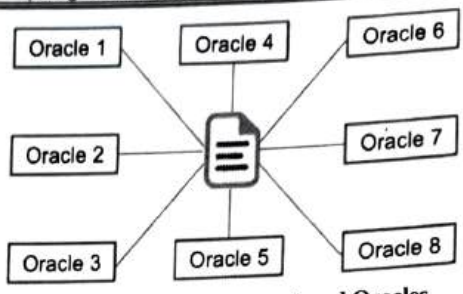
(1B25)**Fig. 3.1.6 : Software Oracle**



(1B26)**Fig. 3.1.7 : Hardware Oracle**

(1B27)**Fig. 3.1.8 : Inbound and Outbound Oracles**

4. **Consensus-based Oracles :** The prediction market relies heavily on oracle for predicting future outcomes. Depending on a single source of information is not advisable and risky too. In order to avoid market manipulation, prediction markets implement a rating system for oracles. An amalgamation of different oracles is used for improving security.



(1B28)**Fig. 3.1.9 : Consensus-based Oracles**

3.2 SMART CONTRACTS IN ETHEREUM

GQ. Write a short note on: Smart contracts in Ethereum.

- A contract is a legal record that unites two parties to perform transactions either immediately or at a later instance. Example of this would be an individual getting to a contract with an insurance company to secure his/her wellness, a person investing in a property that is belonging to someone else, and an organization attempting to sell its shares to some other organizations.
- On the other hand, a smart contract is the one that is executed, deployed, and implemented in an Ethereum environment. Smart contracts result in the digitization of legal contracts. Smart contracts store data. The data that is stored in smart contracts could be used to describe facts, associations, accounts, or other information to execute logic to obtain real-world contracts.
- A smart contract can be thought of as a small program that consists of functions/methods. With the help of a smart contract, an instance can be created and functions/methods can be invoked for viewing and updating contract data in conjunction with logic implementation.
- Smart contracts enable blockchain to be useful in finance and trade, supply chain, banking, etc. Assume that there is crowd funding (*means to raise money/funds from individuals or a group of individuals*) taking place to build/develop a huge project. The project can be raised with the following features :
  - **Project :** Crowdfunding for a huge project
  - **Amount :** 10,000 US dollars
  - **Duration :** 60 days
- **Decision :** If the specified amount is raised within the given duration, then the funds/money would be given for developing the huge project. Nonetheless, if the specified amount is not raised in the given duration, then the funds/money would go back to the respective donor.

- **Step 1 :** Crowdfunding project launched via Ethereum.
- **Step 2 :** As this is public funding, the platform is open to all *(i.e., anybody can join the network, mine, or buy ethers).*
- **Step 3 :** Stop further funding for the project.
- **Step 4 :** If the crowdfunding project meets the specified amount, then transfer the fund/money to the organization. If not, then revert the funds to the respective donors and ensure that the right amount is transferred to the right owner.

3.3 SMART CONTRACTS IN INDUSTRY

GQ. State and explain some use cases of smart contracts in industry.

Smart contracts in blockchain are a new technology that appears to be influencing industries such as finance, healthcare, supply chain and logistics, retail, government, and so on. Some of the use cases are detailed shortly below.

Healthcare industry

- There is a massive opportunity for blockchain revolution to lead digital transformation in the healthcare industry. There are numerous applications for this technology, ranging from medical records to pharmaceutical supply chains to smart contracts for payment distribution.
- Here are some examples of how blockchain and smart contracts can impact the way healthcare services are delivered. Every healthcare system is reliant on medical records. Medical records, on the other hand, tend to become more detailed and involved with each doctor's or medical consultant's visit. It is difficult for healthcare providers to obtain access to these records because each hospital, clinic, or doctor's office stores them differently.
- Individual businesses, such as medical chain and medicos, have devised solutions to this issue. The goal is to provide patients control over their entire medical history and to provide a single point of contact for patients and physicians, bringing security to health record access. Authorized parties can access, receive, and modify their records using the Smart Contract functionality. These documentation changes are verified and authenticated by the blockchain, making them more secure than traditional medical records.
- Prescriptions can be shared and managed with a proper timestamp that cannot be tampered with. This increases the safety of medical records. Physical prescriptions are not required to be carried because they can be accessed by physicians, patients, hospital or clinic staff, and others with appropriate access. This solution, when combined with big data and artificial intelligence, has the potential to predict and manage epidemics, lead to more informed research, and characterize our efforts toward better health.

Manufacturing and supply chain

- From its inception, the supply chain industry has maintained the highest levels of product safety, security, and stability. Because pharmaceutical manufacturing and distribution partners are spread across multiple countries and regions, it is difficult to track and trace the information of each drug or medical device. Furthermore, different countries have different drug laws. Because real-time tracking is currently unavailable, records can be easily altered.
- Furthermore, non-legitimate vendors contribute to the drug black market by fabricating records. Blockchain smart contracts can be used to secure and transparently monitor a supply chain. This can be used to investigate time delays and human errors, which are both extremely costly to the pharmaceutical industry.
- It can be used to track costs, labour, and even waste throughout the supply chain. It can be used to verify the authenticity of a product and track it back to its source in order to combat the counterfeit drug market, which costs the industry approximately $200 billion USD per year.
- Blockpharma and Modus, for example, are working to improve logistics efficiency. Modum, in particular, works in accordance with EU laws, which require proof that the medicine has not been subjected to conditions such as unusually high or low temperatures, which could compromise its quality.
- Sensors that monitor the climatic conditions of the products while they are in transit and physically record them on the blockchain, which will be programmed with smart contracts, will be included in these solutions.

☞ Banking and financial service industry

- **Syndicated loans :** Banks and financial institutions are increasingly using smart contracts to manage standard loans. Consequently, syndicated loans, in which multiple lenders provide loans to multiple borrowers on the same loan terms, stand to benefit significantly from the use of smart contracts. All steps, including syndication, diligence, underwriting, and servicing of syndicated loans, can be completed more quickly using smart contracts. With multiple entities involved in syndicated loans, smart contracts' on-chain/off-chain information makes it much easier to build relationships, identities, and maintain security. Smart contracts thus act to drive effectiveness and efficiency in this scenario. In this scenario, smart contracts act to drive effectiveness and efficiency.
- **Securities :** Private companies can benefit from smart contracts to simplify capitalization table management. Currently, the security's issuer must transfer title to a number of guardians, each of whom will have sub-custodians until it reaches the investor. Various other challenges faced by the securities segment are listed below :

1. Securities are paper-based
2. Manual company registration procedures
3. Increased cost
4. Increased counterparty risk
5. Increased latency.

The following are the advantages of using smart contracts logic :

1. Security is delivered to an investor directly from an issuer.
2. As a result of securities on a distributed ledger, entire workflows have been digitized.
3. Private security markets provide benefits more quickly than public securities markets.
4. The cryptographic signature of a ledger entry replaces the state's seal on paper stock certificates.
5. While issuers will need to know who owns their securities, some buy-side firms will keep this information confidential.

☞ Trade and finance

- Smart contracts facilitate the smooth transfer of goods across borders by enabling smart credit and trade payment initiation, as well as increased liquidity of finance-related assets. Smart contracts provide a risk-mitigated payment method. It also increases the efficiency of all parties involved in the process, including buyers, suppliers, and financial institutions. The current challenges we face are :
  - Obtaining a letter of credit is a time-consuming and expensive process.
  - Because of paperwork and physical document handling, there is various coordination among multiple parties.
  - These manual de-linked processes may result in fraud and duplication of funding.

Using smart contracts logic includes the following benefits :

- Automated compliance and tracking of letter of credit conditions allows for quick payment and approval commencement.
- Enhanced efficiency in creating, changing, and supporting trade, title, and transportation-related contract arrangements.
- Financial asset liquidity has increased as a result of the simplified transport process and a reduction in fraud.

Some of the considerations while implementing smart contracts in trade and finance are :

- Standard smart contract templates for each industry should be implemented, and guidelines should be put in place to ensure wider acceptance and adoption.
- Legal implications for potential smart contract execution fall-out must be decided and addressed, particularly for defaults and dispute resolution.
- The legal implications of potential smart contract execution fallout must be decided and addressed, particularly in terms of defaults and dispute resolution.

☞ Derivatives

- Because most over-the-counter (OTC) derivatives have asset servicing handled independently by each counter party, post-trading has many redundant and time-consuming processes.
- The vast majority of transaction contracts are paper-based and include terms, trade agreements, and/or post-trade confirmations. Smart contracts can improve post-trade processes by eliminating duplicate processes performed by each counter party for record keeping and verifying trades, as well as executing appropriate trade levels and other lifecycle events.
- Implementing standard rules and regulations in smart contracts optimizes OTC derivatives post-trade processing. Some of the considerations while building smart contracts for derivatives are:
  - To manage large-scale protocol changes to current contracts as a result of regulatory reform, contract changes, or other unforeseen events, proper governance must be established.
  - For OTC derivatives, proper agreement on lifecycle events is required (e.g., an external source of data).
  - Oracles required to feed smart contracts with information to/from the blockchain network should be integrated and governed.

☞ Smart contracts in other industries

- There are many other use cases of smart contracts in other industries, as given below:
  - **Legal :** Smart contracts stored in a blockchain track contract parties, terms, transfer of ownership, and delivery of goods or services in the absence of the disadvantageous need for a legal intervention.
  - **Government :** Smart contracts on a blockchain provides promise as a system to maintain personal identifying information, criminal histories and "e-citizenship" authorized by biometrics.
  - **Food :** The blockchain concept may be utilized to trace a product's origin, batch, processing, expiration, storage conditions, and shipment in the food supply chain business.
  - **Insurance :** When autonomous vehicles such as driverless automobiles and smart devices communicate status updates with insurance carriers via the blockchain, overall costs are reduced because there is no need for auditing and verifying data.
  - **Education :** Universities and educational institutions can use blockchain to record grade or credentials data related to tests, degrees, and transcripts and to provide such data to the government or firms looking to hire students.

☞ Travel and hospitality

Hotel guests and airline passengers can use blockchain to maintain their authenticated "single travel ID." This will aid them in the processing of travel documents, identity cards, loyalty programmes, personal preferences, and payment information.

⏭ 3.4 INTRODUCTION TO PROGRAMMING : SOLIDITY PROGRAMMING

- Smart contacts can be written in several programming languages like Serpent, Vyper, Lisp like language (LLL), and Solidity. Solidity a Turning complete and general purpose language—allows for programming smart contracts in a high-level, object-oriented language. The introduction of solidity based on Ethereum Virtual Machine (EVM) makes it easy to deploy with blockchain.
- Solidity programming is similar to JavaScript. It is a case sensitive and object-oriented programming (OOP) language. Even though it is anOOP language, it supports limited features (i.e., data types of variables should be defined and known at compile time). Also, function name and variable name should be written in the same manner as they are defined for successful execution.

- Solidity programs are written and savedwith a *.sol*extension, and statements are terminated in solidity programming with a semicolon (;). Any text editor can be used to open and read solidity programming files.
- A solidity programming file broadly consists of the following structure :

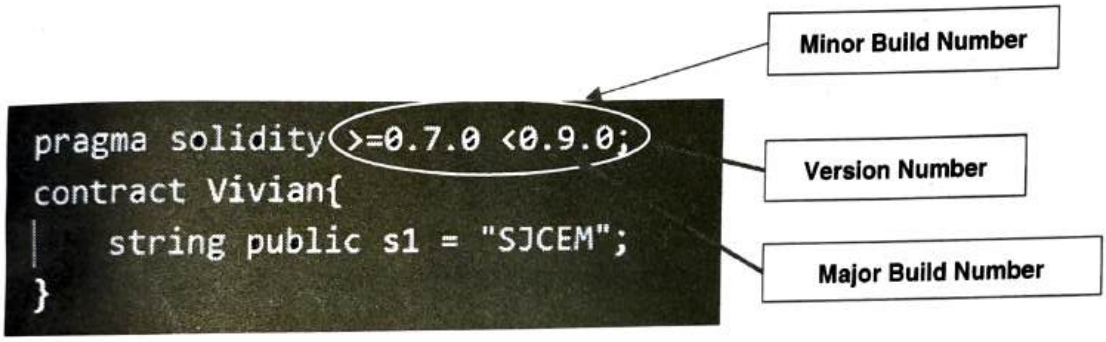
1. Pragma

It is the 1st line of code that is generally written within any solidity file. *Pragma* usually specifies which version of the compiler has to be used for a solidity file. The syntax for *pragma* is

```
pragma solidity <<version number>>;
```

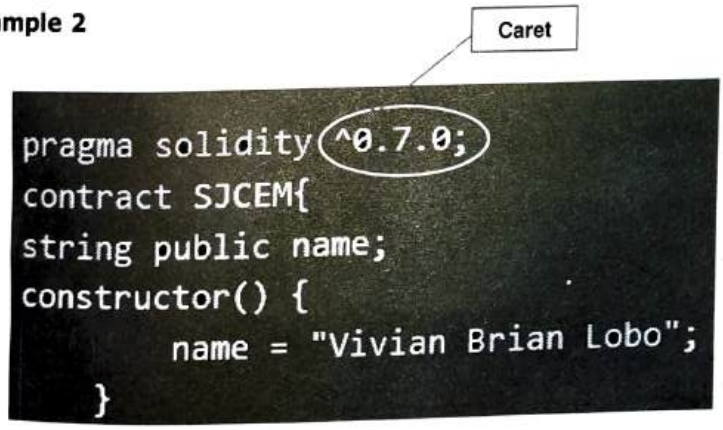
Using *pragma*, one can select a desired compiler version and get your target code executed accordingly.

☞ **Example 1**



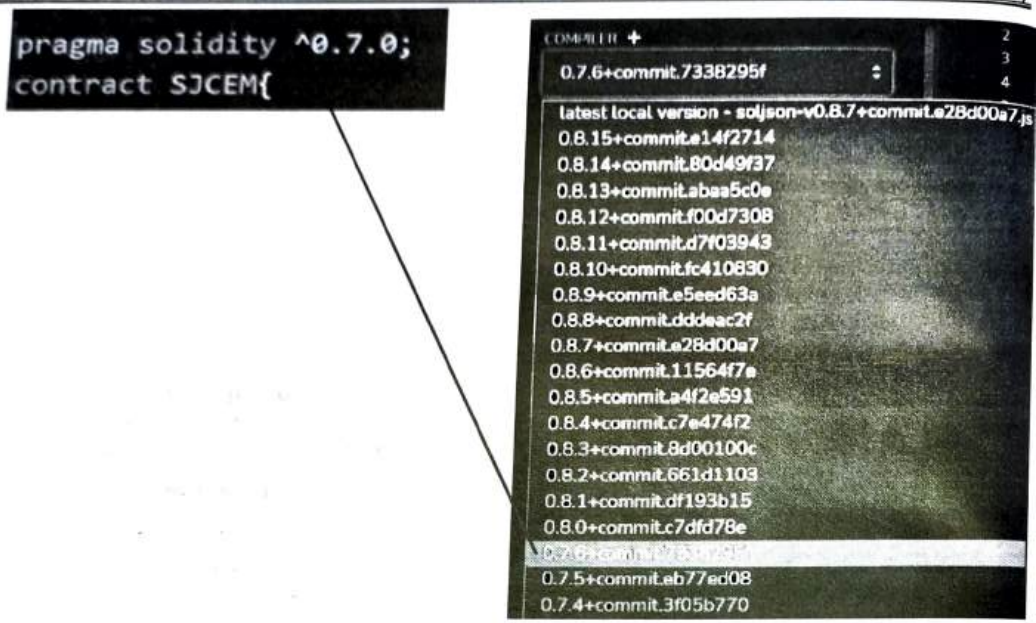
The version number comprises a major and minor build number wherein the major build number here is 9 and the minor build number here is 0. Here, >=0.7.0 <0.9.0 indicates that one can use any solidity compiler having version number that is greater than equal to 0.7.0 and less than 0.9.0.

☞ **Example 2**



The ^ character is called a *caret*, which is optional but plays a significant role in deciding the version number of a solidity compiler.

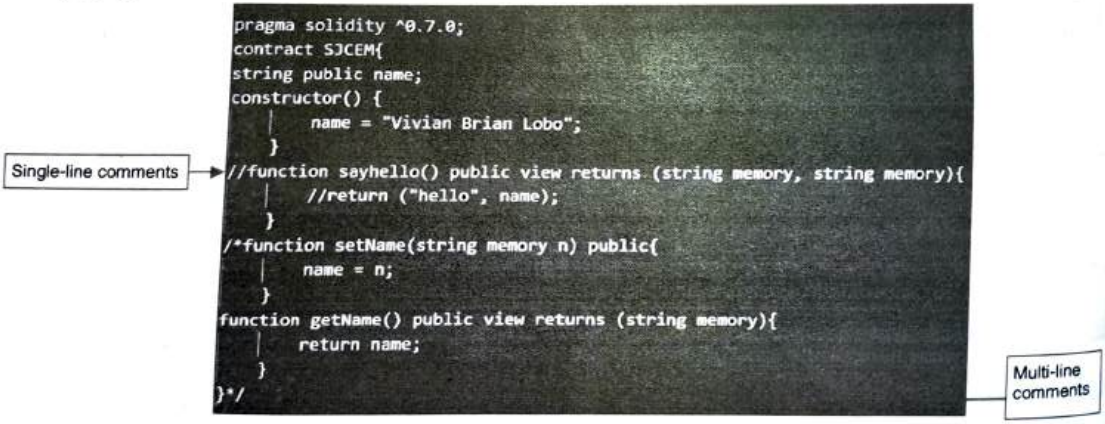
Note : Caret refers to the latest version within a specified major build number (i.e., ^0.7.0 refers to the major build number with 0.7.6 being the latest version).



- In a similar manner, for ^0.8.0, the latest version would be 0.8.15.
- The caret will not target any other major build apartfrom the one that is provided.
- The solidity file will compile only with a compiler with 7 asthe major build. It will not compile with any other majorbuild.

2. Comments

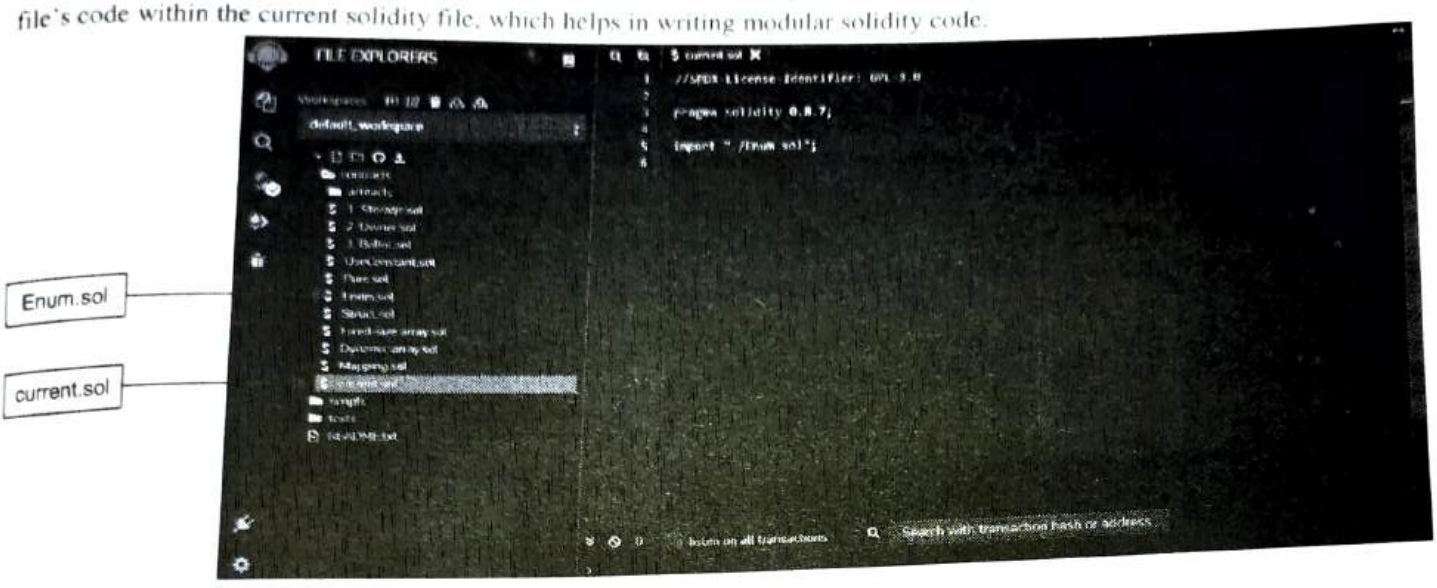
Like any other programming language, solidity offers the provision to add single-line (//) and multi-line comments (/*...*/).




```
pragma solidity ^0.7.0;
contract SJCEM{
string public name;
constructor() {
    name = "Vivian Brian Lobo";
}
//function sayhello() public view returns (string memory, string memory){
    //return ("hello", name);
}
/*function setName(string memory n) public{
    name = n;
}
function getName() public view returns (string memory){
    return name;
}
}*/
```

☞ 3. Import

With the help of *import* keyword, other solidity files can be imported. This keyword helps in accessing other solidity file's code within the current solidity file, which helps in writing modular solidity code.



For instance, *current.sol* is the solidity file wherein *Enum.sol* file needs to be imported, which can be done by using *import* keyword.

➡ **Note :** While using **http://remix.ethereum.org/** and writing code in solidity, it is essential that the license identifier (i.e., **//SPDX-License-Identifier: GPL-3.0**) should be written before specifying the version number of the solidity compiler.

☞ Basics

Structure of a smart contract

- The primary goal of solidity programming is to write smart contracts for Ethereum. Smart contracts are the basic unit of deployment and execution for Ethereum Virtual Machines (EVMs). Smart contracts mainly consists of two important aspects, i.e., *variables and functions*.
- A smart contract consists of state variables, structure definitions, modifier definitions, event declarations, enumeration definitions, and function definitions

State variable

- A variable that is declared in a contract that is not inside any function is called a state variable.
- It stores the current address of the contract. A state variable's allotted memory is statically assigned, meaning that it cannot change while the contract is in effect.
- State variables can possess qualifiers such as *internal*, *private*, *public*, and *constant*.
  - ***Internal :*** If nothing is specified to a state variable, then by default, it is termed to be internal. This means that a variable can be accessed inside the current contract. Accessibility from outside the contract is not permitted. However, such variables can be viewed from outside.
  - ***private :*** It is similar to *internal* but stringent. Private state variables can only be used in contracts declaring them. Moreover, they cannot be used even within derived contracts.
  - ***public :*** With the help of *public*, state variables can be accessed directly.
  - ***constant :*** Using *constant* makes state variables unalterable.

Functions

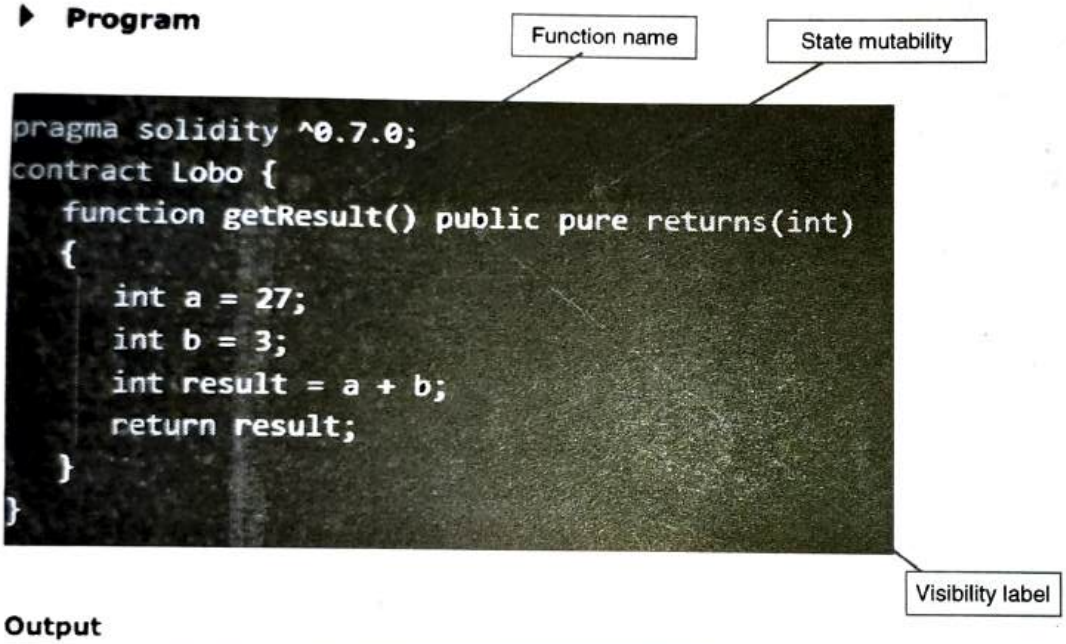
Functions play a crucial role in any programming language, and the same is applicable for solidity programming. Ethereum maintains a state variable's current state and executes transactions to change values in state variables. When a function in a smart contract is invoked, it results in transaction creation.

- Functions are the means to not only read but also write values from/to state variables.
- On-demand execution of code can be performed through functions by simply calling them.
- Functions can accept arguments or parameters,execute its logic, and optionally return values to the caller.

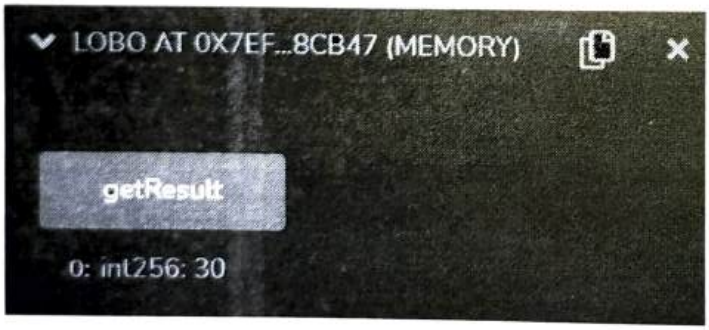
In solidity programming, the most common way of defining a function is by using the keyword ***function*** followed by a function name, a list of arguments or parameters, and a set of statements.

Example 1

Program

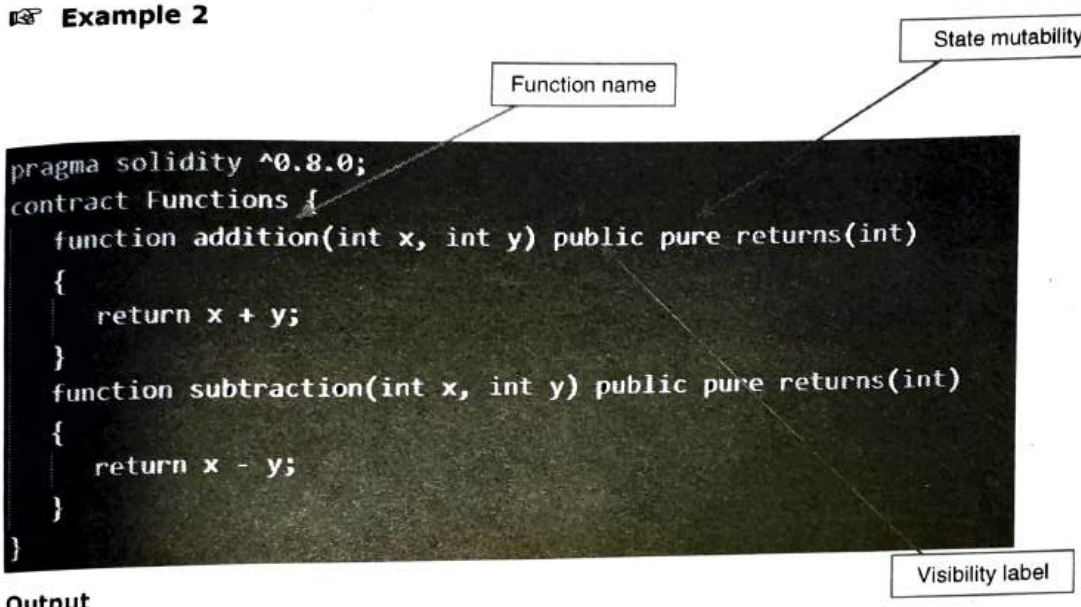


Output

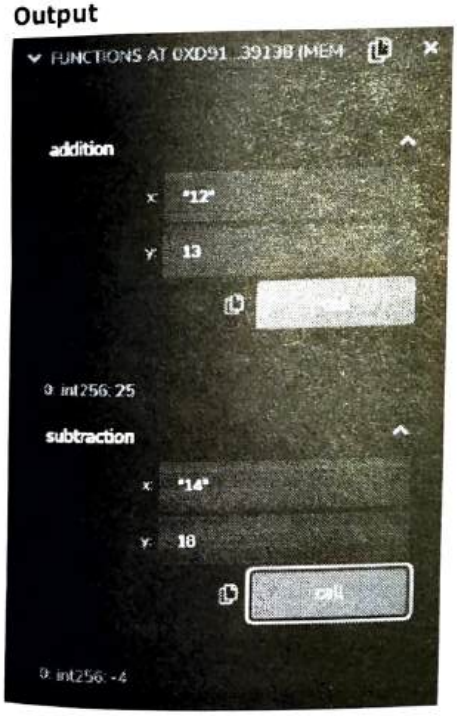


The above program is a simple example that defines a function called *getResult()* that takes no arguments or parameters.

☞ Example 2



Output



The above program is a simple example that define functions *addition()* and *subtraction()* that takes two integer arguments or parameters (i.e., *x* and *y*).

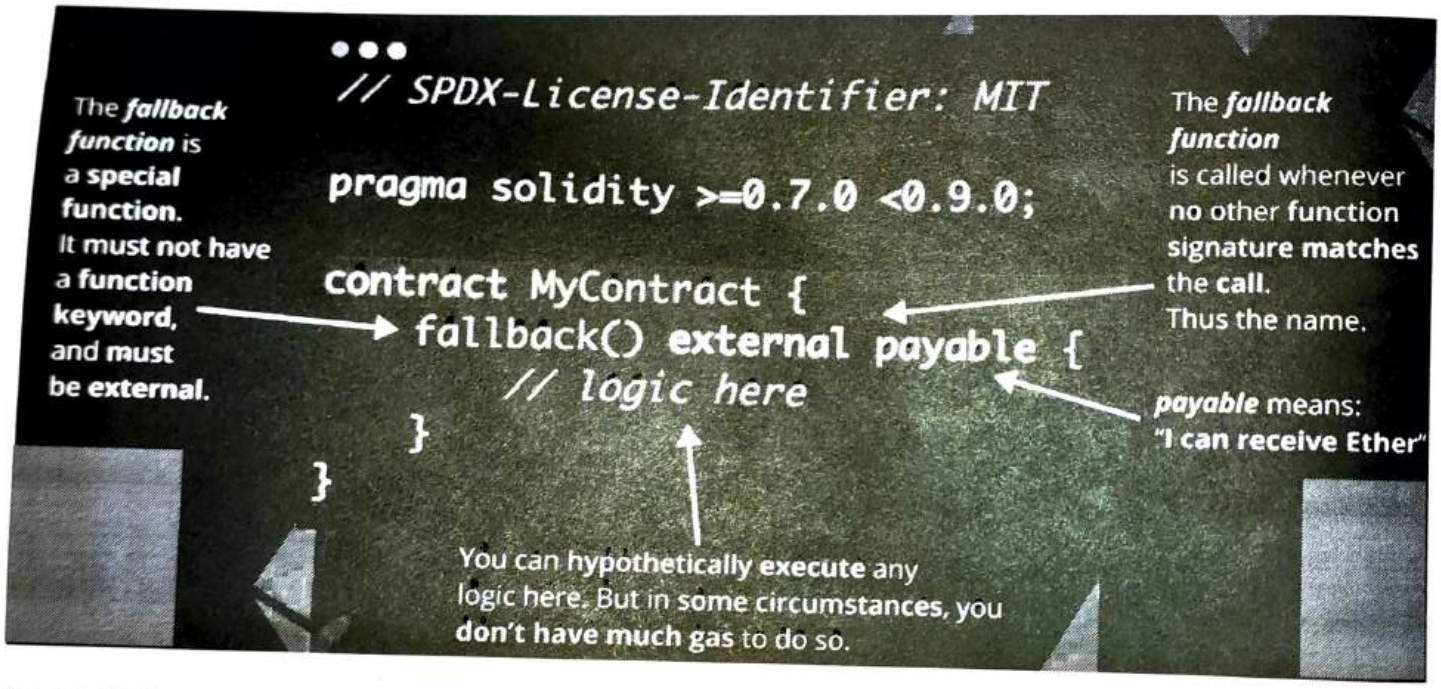
Fall back function

- If none of the other functions match the function identification or no data was provided with the function call, the solidity fallback function is called. A contract can only have one unnamed function, which is executed anytime the contract gets plainether without any data.
- To receive ether and add it to the contract's total balance, the fallback function must be marked payable. If such a function does not exist, then the contract will fail to receive anether via ordinary transactions and will throw an error.

☞ Properties

1. It has no name or arguments.
2. If it is not marked *payable*, then the contract will throw an exception if it receives plain ether without data.
3. It cannot return anything
4. It is also executed if the caller meant to call a function that is not available
5. It is mandatory to mark it external.

Example



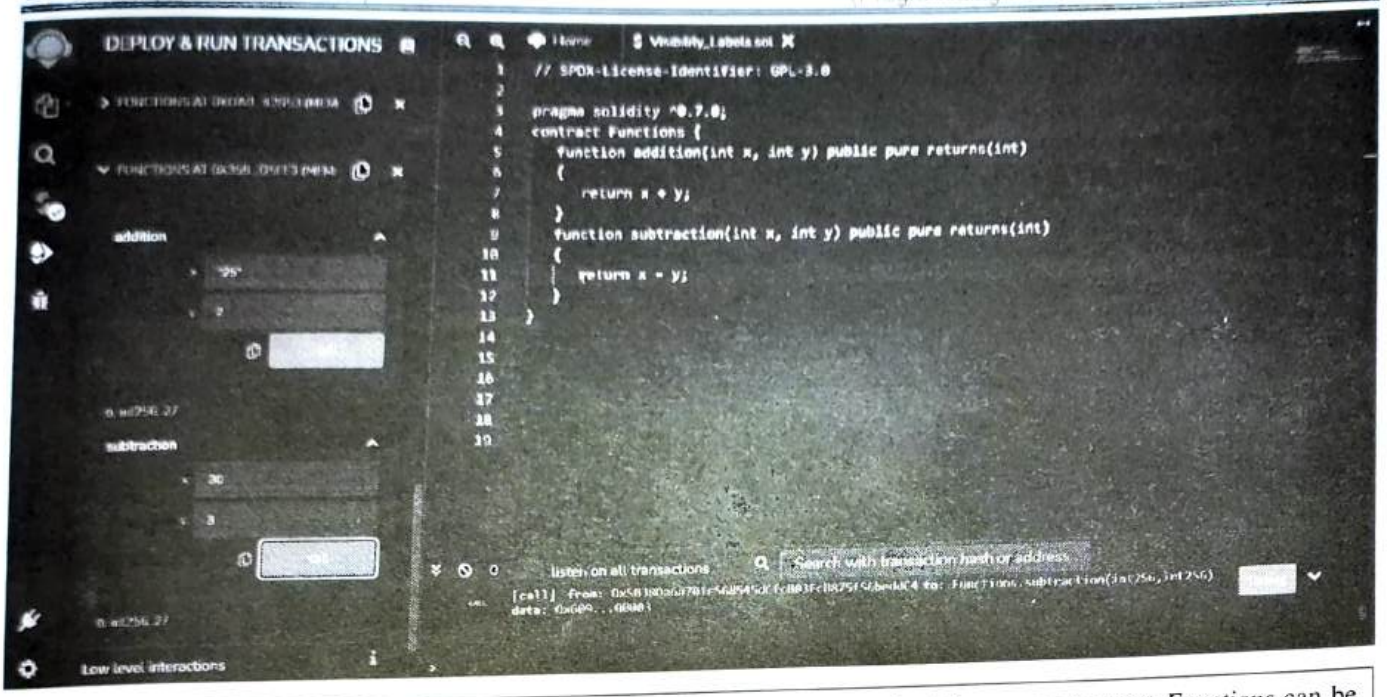
☞ Visibility and Activity Qualifiers

- There exists several quantifiers that affect not only the behavior of a function but also its execution. Functions possess visibility quantifiers and quantifiers pertaining to what actions/activities can be executed inside a function (i.e., activity quantifiers).
- Data can be returned in functions using the keyword *return*.

☞ Visibility Qualifiers

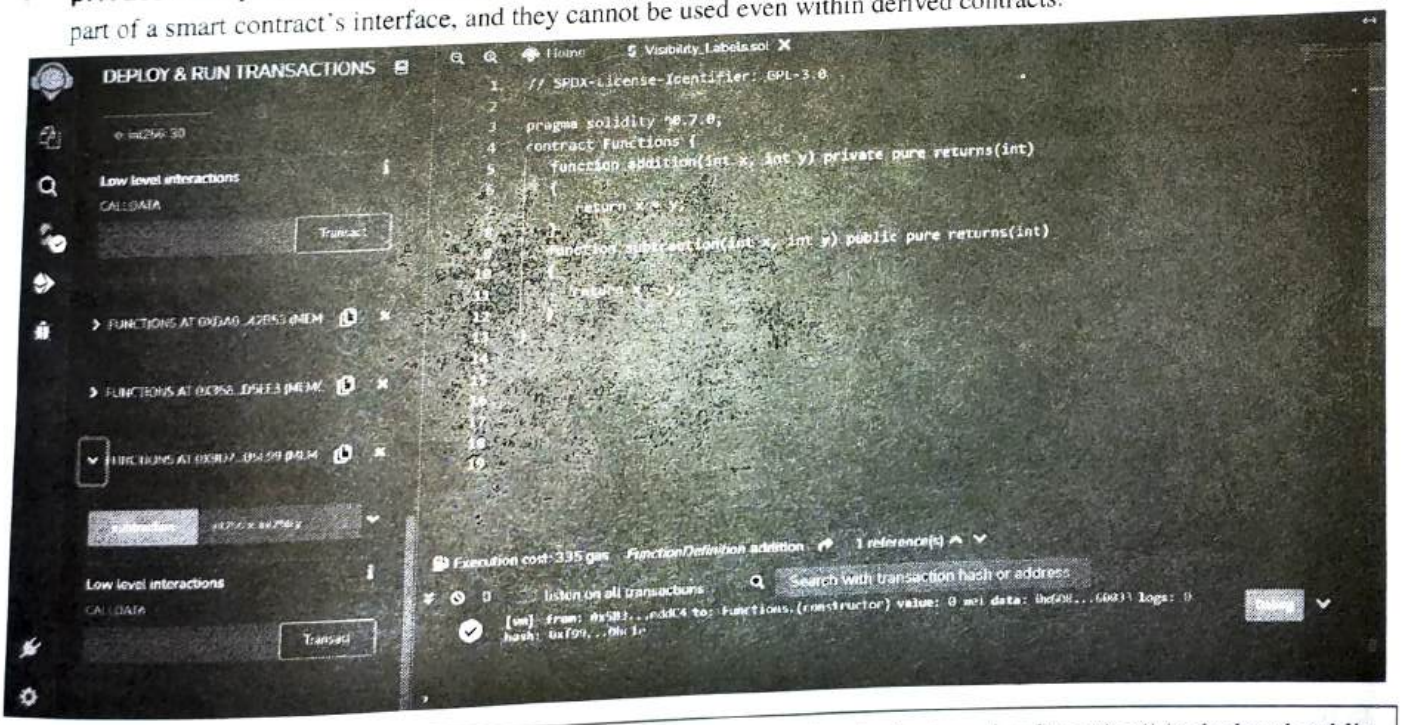
A function's visibility can be any one of the following :

- **public :** It makes a function access directly from outside (i.e., externally). Public functions become a part of the smart contract interface and can be called from within as well as from outside.



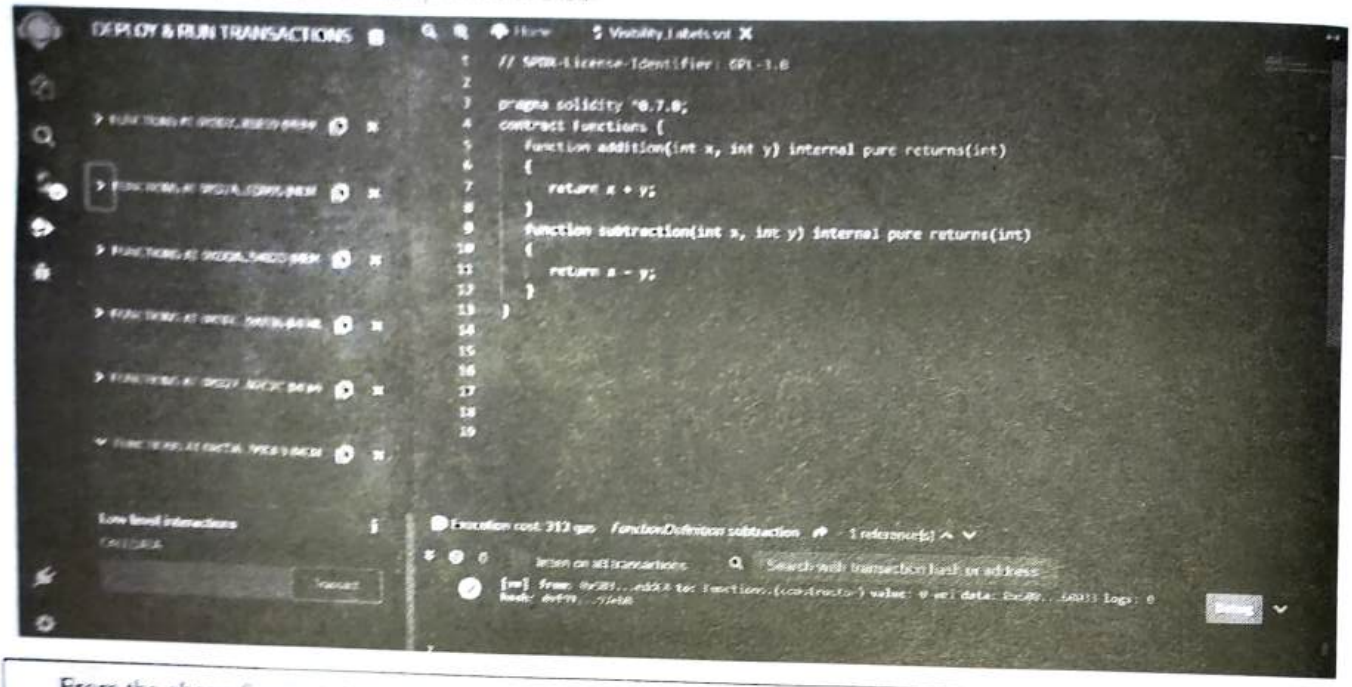
From the above figure, it is evident that functions *addition()* and *subtraction()* in a smart contract *Functions* can be accessed externally after it has been successfully deployed.

- **private :** They can be used only if a contract is explicitly declared with a keyword *private*. Private functions are not a part of a smart contract's interface, and they cannot be used even within derived contracts.



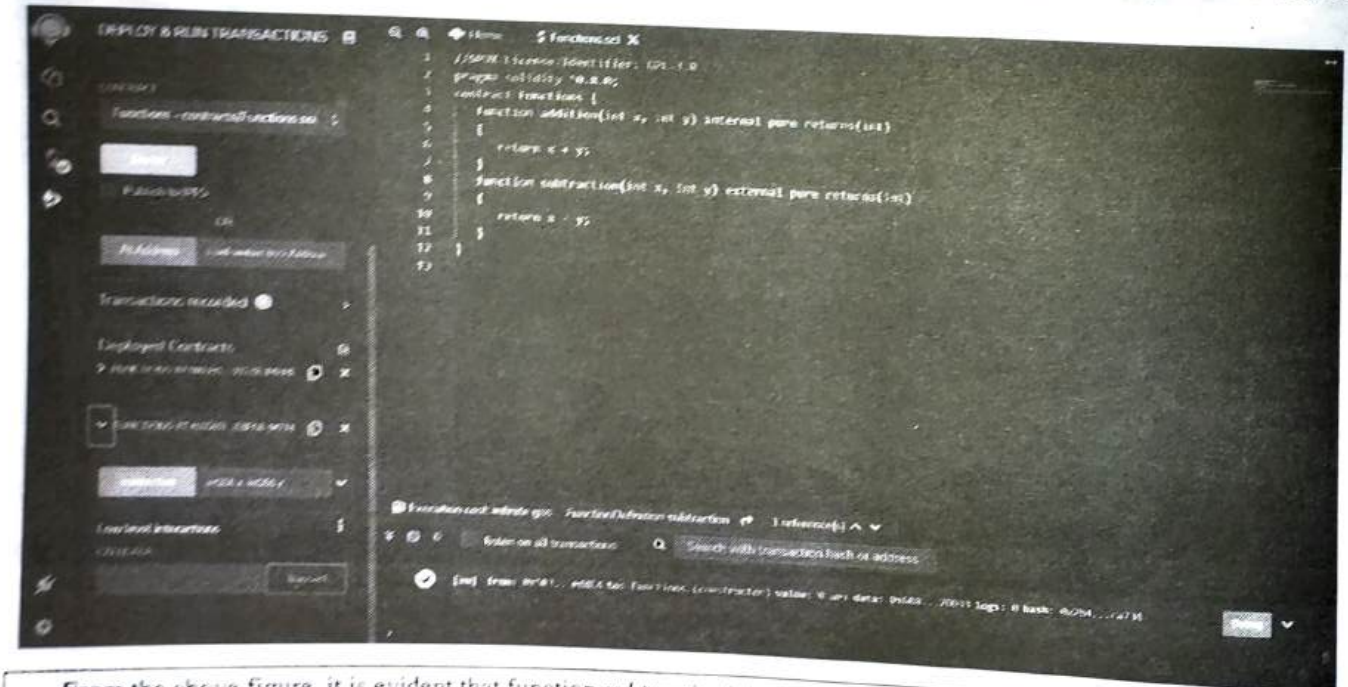
From the above figure, it is evident that function *addition()* is declared *private* and *subtraction()* is declared public. Thus, only *subtract()* can be accessed at the interface of the smart contract.

- **internal** : This function can only be used within the smart contract and any contract that inherits from it. Such functions cannot be accessed from outside and are not part of a smart contract's interface. Note that if nothing is specified, then by default *internal* qualifier is used.



From the above figure, it is evident that both functions *addition()* and *subtraction()* are declared internal, and so, they cannot be accessed from outside.

- **external** : It makes a function access directly from outside but not inside. Such functions become part of the contracts interface.

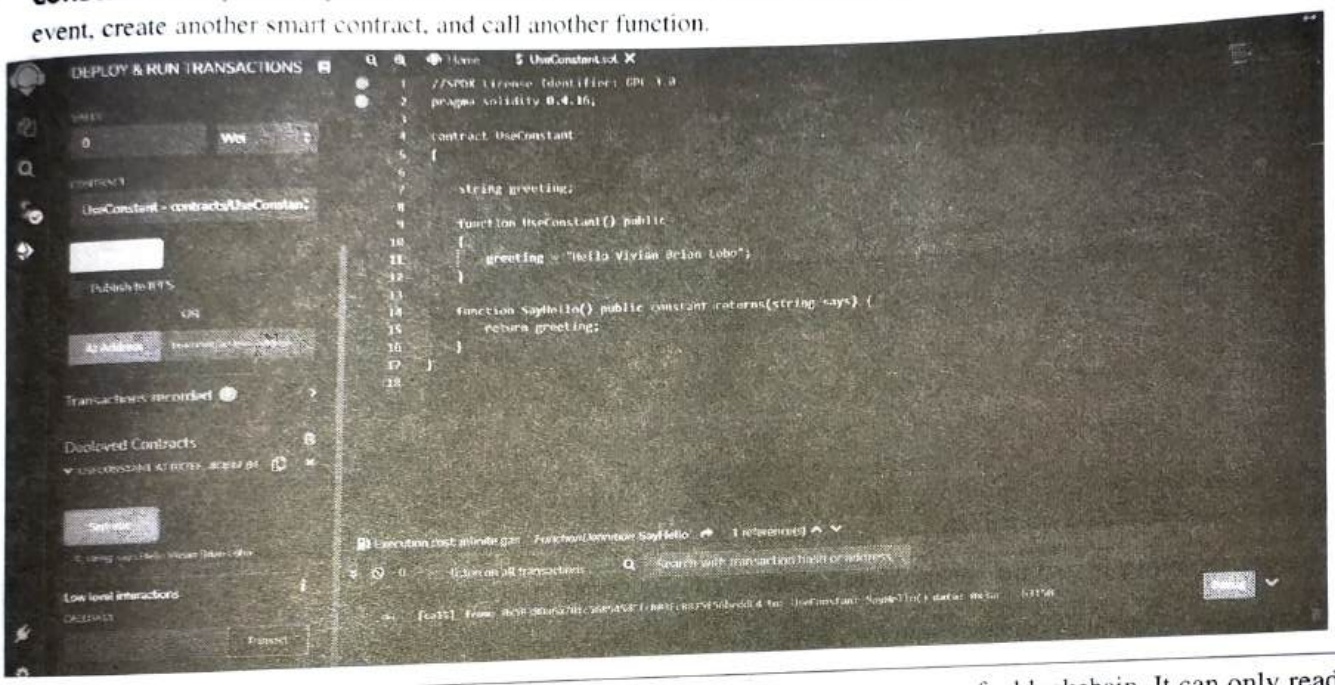


From the above figure, it is evident that function *subtraction()* is declared external, which is accessible from the smart contract's interface, whereas function *addition()* cannot be accessed.

Activity Qualifiers

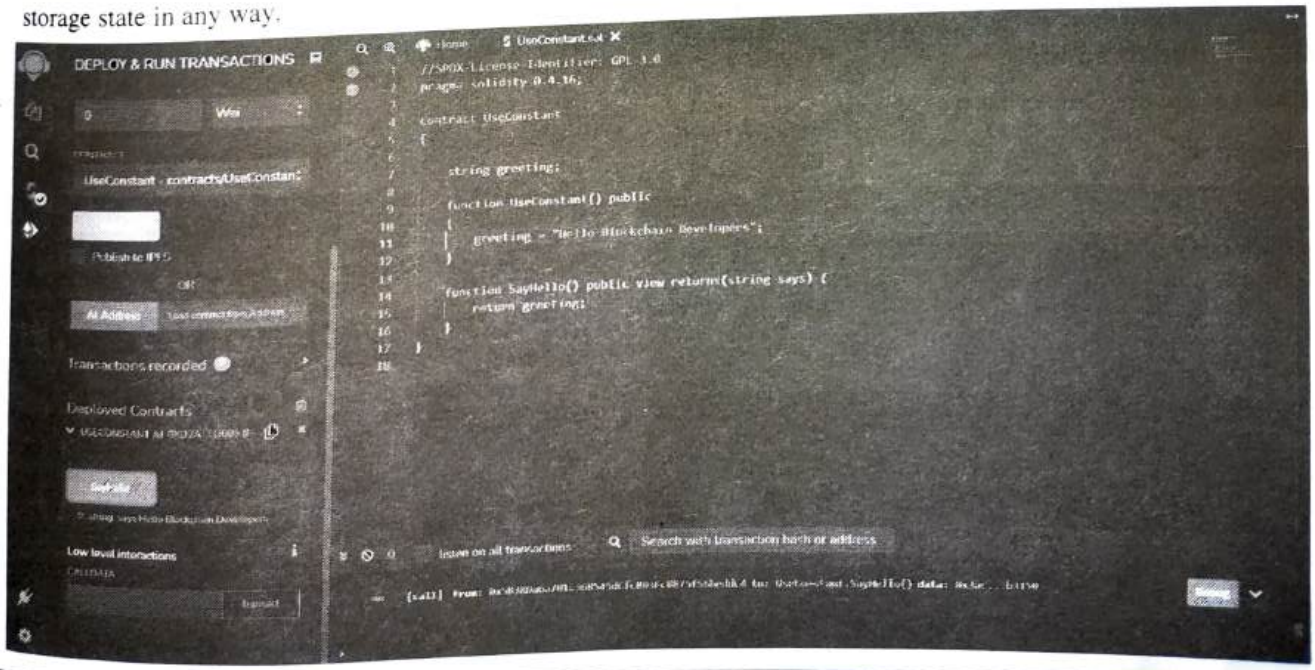
A function's behavior can be changed through the following activity qualifiers :

- **constant :** They can only read state variables and return back to the caller but cannot modify any variable, invoke an event, create another smart contract, and call another function.



From the above figure, it is evident that function *SayHello()* cannot modify the state of a blockchain. It can only read the state variable in this case.

- **view :** It can be considered asa replacement or an alias for *constant*. It indicates that a function will not modify the storage state in any way.



Here, the function *SayHello()* is denoted by an activity qualifier *view*.

- **pure :** They are even stricter than *view*. It can neither make any changes to state variables nor read them. They cannot make any calculations that use any state variables as its component.

Here, the function *getSum()* does not read or modify any state variables, and so, it is considered *pure*.

- **payable :** Functions declared with the *payable* keyword can only accept Ether from a caller. The call will fail in case Ether is not provided by a sender. A function can only accept an Ether if it is marked as *payable*.

☞ Address and Address Payable

The distinction between address and address payable was introduced in Solidity version 0.5.0. The idea was to make the distinction between addresses that can receive money, and those who can't (used for other purposes). Simply speaking, an address payable can receive Ether, while a plain address cannot.

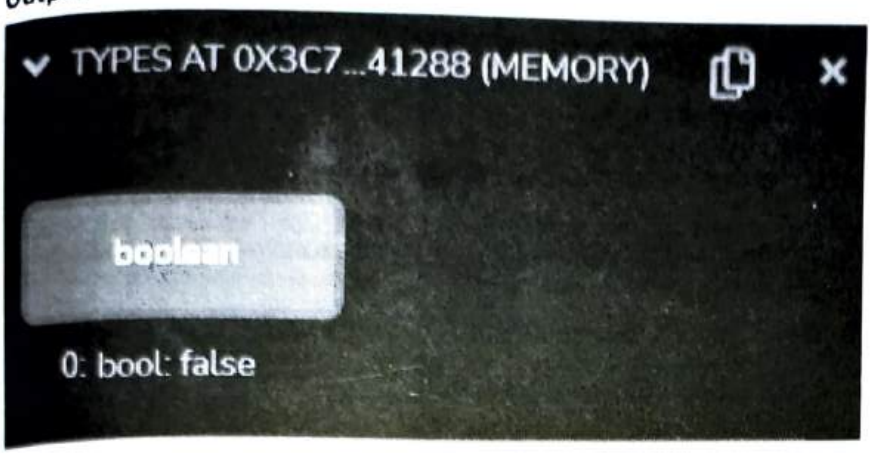
☞ Data types in solidity

- **bool :** It can hold true or false as its value.

▶ Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^ 0.8.7;

// Creating a contract
contract Types {

    // Initializing Bool variable
    bool public boolean = false;


}
```

Output



- **uint :** They are unsigned integers that can have only 0 and +ve values only.

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^ 0.8.7;

// Creating a contract
contract Types {

// Initializing Unsigned Integer variable
    uint public int_var = 60313;

}
```

Output

- **int :** Theyare signed integers that can hold +ve as well as –ve values.

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^ 0.8.7;

// Creating a contract
contract Types {

// Initializing Integer variable
    int32 public int_var = -60313;

}
```

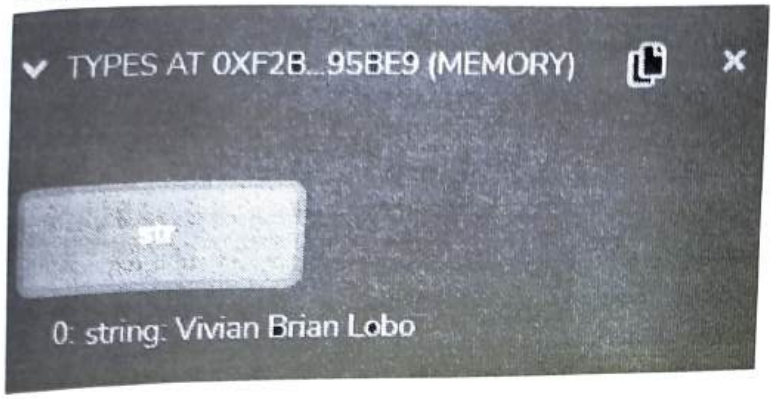
Output

- **string :** A combination of characters is known as strings. In solidity programming, the keyword **string** is used to declare a string.

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^ 0.8.7;

// Creating a contract
contract Types {

//  Initializing String variable
    string public str = "Vivian Brian Lobo";

}
```

Output



- **bytes :** It refers to 8 bit signed integers. Everything in memory is stored in bits consisting of binary values—0 and 1. Solidity provides the byte data type to store information in binary format. In general, programming languages have a single data type for representing bytes. However, solidity offers different types of byte datatype. It provides data types in the range from bytes1 to bytes32 to represent varying byte lengths. These are called fixed-sized byte arrays and are implemented as value types. The bytes1 data type represents 1 byte and bytes2 represents 2 bytes. The default value for byte is 0x00, and it gets initialized with this value. Solidity also has a byte type that is an alias to bytes1.
- A byte can be assigned character values as follows :

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^ 0.8.7;

// Creating a contract
contract Types {

// Initializing Byte variable
   bytes1 public b = "v";

}
```

Output

- *In a similar manner, we have*
- *bytes1 public vv = 0x65;* *// A byte can be assigned byte values in hexadecimal format*
- *bytes1 public rr = 28;* *// A byte can be assigned integer values in decimal format*
- *bytes1 public ss = -22;* *// A byte can be assigned negative integer values in decimal format*

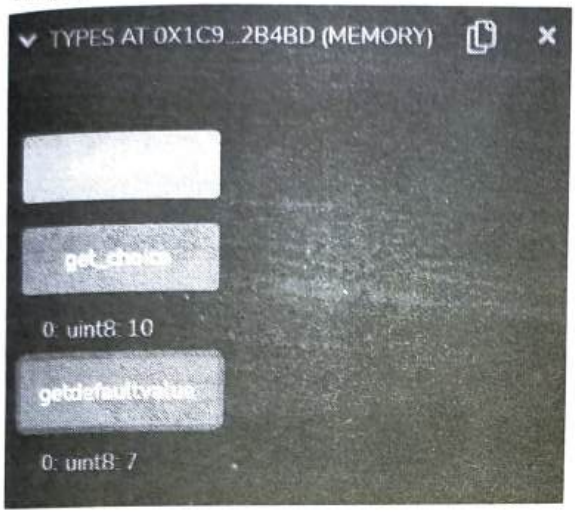
- **Enums :** They are a technique to create user-defined data types and are generally used to provide names for integral constants, which improves the readability and maintenance of a smart contract. Enums are predetermined values that limit a variable to one of a small number of options. Integer numbers beginning at zero are used to represent options.Enums can also have a default value. The number of defects in a code can be decreased by using enums.

Program

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.7;

contract Types {
  enum months_of_a_year   // Creating an enumerator
  {
  January, February, March, April, May, June, July, August, September, October, November, December
  }

  months_of_a_year month;   // Declaring variables of type enumerator
  months_of_a_year choice;

  months_of_a_year constant default_value = months_of_a_year.August;   // Setting a default value

  function set_value() public
  {
  choice = months_of_a_year.November;
  } // Defining a function to set value of choice

  function get_choice() public view returns (months_of_a_year)
  {
  return choice;
  } // Defining a function to return value of choice

  function getdefaultvalue(
  ) public pure returns(months_of_a_year) {
    return default_value;
  } // Defining function to return default value
}
```

In the example, the contract *Types* consist of an enumerator months_of_a_year, and functions are defined to *set* and *get* a variable's value of the type enumerator.

Output



☞ Arrays : Fixed and Dynamic Arrays

- **Arrays :** A fixed collection of identically typed elements are stored in arrays. Each element in an array has a unique place known as an index.
- An array of a specific size needs to be declared, and then we need to put the elements in the array and make use of the index to retrieve the array's elements rather than establishing several separate variables of the same type. Arrays in solidity can either be fixed or dynamic in terms of its size. The first element in an array is represented by the lowest index, and the last element is represented by the highest index.

1. **Fixed-size array :** An array's size ought to be known in advance. The total number of elements must not be greater than an array's size. If an array's size is not supplied, then an array of sufficient size needs to be built to accommodate the initialization.

▸ Program

```
pragma solidity ^0.8.7;

contract Types {

    // Declaring state variables of type array
    uint[7] data1_fixed_size;

    // Defining function to add values to an array
    function array_example() public returns (
    int[5] memory, uint[7] memory){

        int[5] memory data
        = [int(150), -163, 277, -128, 901];
        data1_fixed_size
        = [uint(101), 202, 303, 404, 505, 606, 450];

        return (data, data1_fixed_size);
    }
}
```

In the example, the contract *Types* are created to demonstrate how to declare and initialize *fixed-size arrays*.

Output

```
decoded output                {
                                "0": "int256[5]: 150,-163,277,-128,901",
                                "1": "uint256[7]: 101,202,303,404,505,606,450"
                              }
```

2. **Dynamic array :** When an array is declared, its size is not known in advance. An array's size changes when new elements are added, and it will be calculated during runtime.

Program

```
pragma solidity 0.8.7;

contract Types {

  // Declaring state variable of type array. One is fixed-size and the other is dynamic array
  uint[] data = [101, 202, 303, 404, 505];
  int[] data1_dynamic_array;

  // Defining function to assign values to dynamic array
  function dynamic_array() public returns(
  uint[] memory, int[] memory){

    data1_dynamic_array
    = [int(-607), 707, -807, 907, -1007, -1207, 1407];
    return (data, data1_dynamic_array);
  }
}
```

In the example, the contract *Types* are created to demonstrate how to declare and initialize *dynamic arrays*.

Output

```
decoded output                {
                                "0": "uint256[]: 101,202,303,404,505",
                                "1": "int256[]: -607,707,-807,907,-1007,-1207,1407"
                              }
```

Special Dynamically Sized Arrays

Solidity provides two special arrays:

- **bytes array :** It is a dynamic array that can hold any number of bytes(i.e., they can hold an arbitrary length of raw byte data). As bytesis treated as an array in solidity programming, it can have a length of zero and one can append a byte to the end.
- **string array :** It is a dynamic array of UTF-8 data. *string* in solidity programming does not provide functions to obtain string length or perform concatenation or comparison between two strings. A string can be converted to a byte array using *bytes(<string>)*.

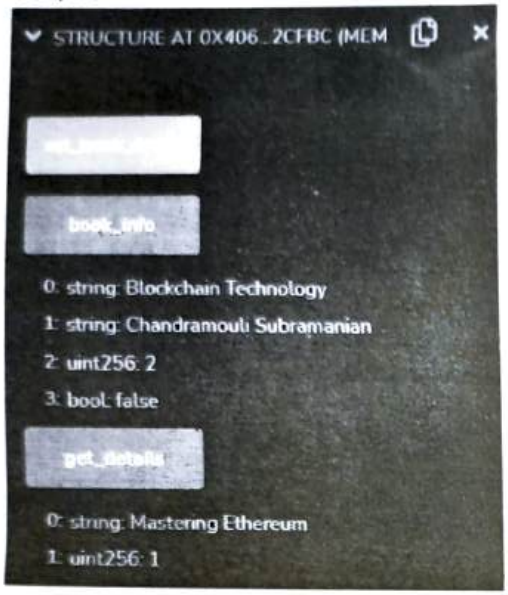
- **struct :** Solidity allows an individual to create complicated datatypes that have multiple properties, which can be done by using struct.One can define his/her own datatype by creating a struct. Structs are helpful for collecting related data into one category. Structures may be imported into one contract from another after being declared outside of it. It often serves as a record representation. The struct keyword, which generates a new data type, is used to define a structure. The "dot operator" is used to delineate the space between the struct variable and the element one wants to access when accessing any element of the structure. To define the variable of structure data type, structure name is used.

Program

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.7;

contract Structure {

// Declaring a structure
struct Book {
  string name; string writter; uint id; bool available;
}

// Declaring a structure object
Book book1;

// Assigning values to the fields for the structure object book2
Book book2 = Book("Blockchain Technology", "Chandramouli Subramanian", 2, false);

// Defining a function to set values for the fields for structure book1
function set_book_detail() public {
  book1 = Book("Mastering Ethereum", "Andreas M. Antonopoulos Dr.Gavin Wood", 1, true);
}

// Defining function to print book2 details
function book_info()public view returns (string memory, string memory, uint, bool) {
    return(book2.name, book2.writter, book2.id, book2.available);
  }

// Defining function to print book1 details
function get_details(
) public view returns (string memory, uint) {
  return (book1.name, book1.id);
}
}
```

In the example, the contract *Structure* consists of a structure Book, and functions are defined to *set* and *get* values of the elements of the structure.

Output



- **Mapping :** They store the data in the form of key–value pair. A key can be any of built-in data types, although reference types are not permitted, and the value can be of any type. Mostly, mappings are used to link the specific Ethereum address to the relevant value type. They act like a hash table or dictionary in any other programming language.Mapping is defined as any other variable type, which accepts a key type and a value type.
  - ***key type :*** It can be any built-in types as well as bytes and string. No reference type or complex objects are allowed.
  - ***value type :*** It can be any type.

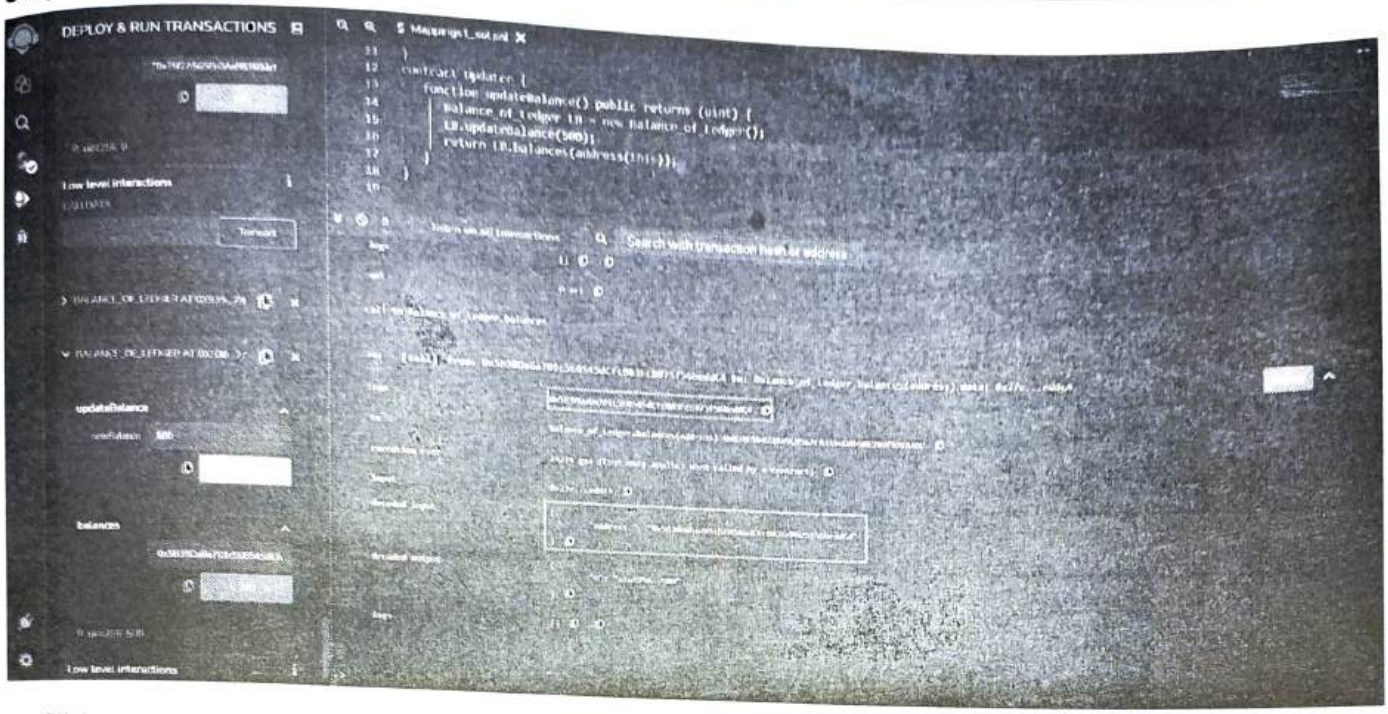
Program

key

value

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.7;

contract Balance_of_Ledger {
    mapping(address => uint) public balances;

    function updateBalance(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
contract Updater {
    function updateBalance() public returns (uint) {
        Balance_of_Ledger LB = new Balance_of_Ledger();
        LB.updateBalance(500);
        return LB.balances(address(this));
    }
}
```

Output



Click on *updateBalance* button to set the value as 500 and then in the *balances* button add *from* address and click on *call* button, will show the **decoded output** as

```
decoded output
{
        "0": "uint256: 500"
}
```

☞ Inheritance

- One of the most important features of the object-oriented programming language is inheritance.
- It is a method of extending a program's functionality by separating the code, reducing dependency, and increasing the re-usability of existing code.
- Solidity supports smart contract inheritance, allowing multiple contracts to be inherited into a single contract.
- A base contract is the contract from which other contracts inherit features, whereas a derived contract is the contract that inherits the features.
- They are simply known as parent–child contracts. In solidity programming, inheritance is restricted to public and internal modifiers.
- Solidity has the "is" keyword to inherit the base/parent contract to the derived/child contract.

Difference types of inheritance

- **Single inheritance**
  - In this type of inheritance, the functions and variables of one parent/base contract are inherited to only one child/derived contract.

Program

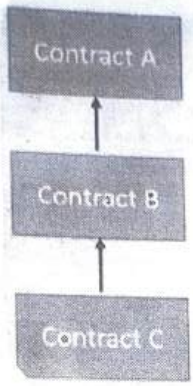
```
// Defining calling contract
contract caller {

    // creating child/derived contract object
    derived d = new derived();

    // Defining function to call
    // setValue and getValue functions
    function testInheritance(
    ) public returns (uint) {
        d.setValue();
        return d.getValue();
    }
}
```

```
//SPDX-License-Identifier: GPL 3.0

pragma solidity 0.8.7;

// Defining parent/base contract
contract base{

    // Declaring internal state variable
    uint internal addition;

    // Defining external function to set value of internal state variable sum
    function setValue() external {
        uint a = 25;
        uint b = 2;
        addition = a + b;
    }
}

// Defining child/derived contract
contract derived is base{

    // Defining external function to return value of internal state variable sum
    function getValue(
    ) external view returns(uint) {
        return addition;
    }
}
```

Output

```
decoded output                {
                                  "0": "uint256: 27"
                              }
```

- **Multilevel inheritance**
  - It is similar to single inheritance, but the difference is that it has levels of the parent–child relationship. The child contract derived from a parent also serves as a parent to the contract derived from it.




```
contract A {
  ..............
}

contract B is A {
  ..............
}

contract C is B {
  ..............
}
```

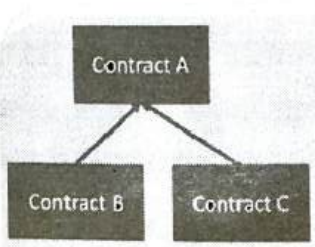
Program

```
//SPDX License Identifier: GPL 3.0

pragma solidity >=0.4.22 <0.6.0;

// Defining parent/base contract A
contract A {
    string internal x;
    string a = "Block";
    string b = "Chain";

    // Defining external function to return concatenated string
    function getA() external{
        x = string(abi.encodePacked(a, b));
    }
}

// Defining child/derived contract B
// inheriting parent/base contract A
contract B is A {

    string public y;
    string c = "Technology";

    // Defining external function to return concatenated string
    function getB() external payable returns(string memory){
        y = string(abi.encodePacked(x, c));
    }
}

// Defining child/derived contract C
// inheriting parent/base contract A
contract C is B {

    // Defining external function
    // returning concatenated string
    // generated in child/derived contract B
    function getC() external view returns(string memory){
        return y;
    }
}

// Defining calling contract
contract caller {

    // Creating object of child/derived contract C
    C cc = new C();

    // Defining public function to return final concatenated string
    function testInheritance(
    ) public returns (
    string memory) {
        cc.getA();
        cc.getB();
        return cc.getC();
    }
}
```

Output

```
decoded output  {
                    "0": "string: BlockChainTechnology"
                }
```

- **Hierarchical inheritance**
  - In this type of inheritance, a parent contract has more than one child contract. It is commonly used when the same functionality must be used in multiple places.




```
contract A {
    .....
}
contract B is A {
    .....
}
contract C is A {
    .....
}
```

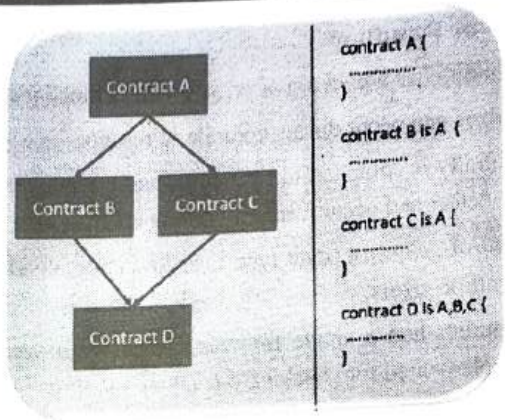
Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;

contract A {

    string internal X;
    function get_a_value() external
    {
        X = "Blockchain and Blockchain Lab";
    }

    string internal Y;
    function get_a_values() external
    {
        Y = "Solidity Programming in Blockchain";
    }


} contract B is A {
      function get_a_value_1() external view returns(string memory)
      {
          return X;
      }
} contract C is A {
    function get_a_values_2() external view returns(string memory)
    {
        return Y;
    }

}
contract caller {
    B obj_b = new B();
    C obj_c = new C();

    function testInheritance() public view returns (string memory, string memory) {
        return (obj_b.get_a_value_1(), obj_c.get_a_values_2());
    }
}
```

Output

```
decoded output        {
                        "0": "string: Blockchain and Blockchain Lab"
                        "1": "string: Solidity Programming in Blockchain"
```

- **Multiple inheritance**
  - In this type of inheritance, a single contract can be inherited from multiple contracts.
  - A parent contract can have multiple children, whereas a child contract can have multiple parents.



Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;

contract A {

    uint internal a;

    function getA(uint value) external {
        a = value;
    }
}

contract B {

    uint internal b;

    function getB(uint value) external{
        b = value;
    }
}

contract C is A, B {

    function getValueOfSum() external view returns(uint) {
        return a + b;
    }
}

contract caller {

    C obj_c = new C();

    function testInheritance() public returns(uint) {
        obj_c.getA(25);
        obj_c.getB(25);
        return obj_c.getValueOfSum();
    }
}
```

Output

```
decoded input                {
                                "uint256 value": "50"
                             }
```

Error Handling

- Solidity has a plethora of error handling functions.
- Errors can occur during compile or runtime.
- Solidity is compiled to byte code, and a syntax error check occurs at compile time, whereas runtime errors are difficult to detect and occur primarily during contract execution.
- Out-of-gas error, data type overflow error, divide by zero error, array-out-of-index error, and so on are examples of runtime errors.
- Solidity had a single throw statement until version 4.10; so to handle errors, multiple if...else statements must be implemented for checking the values and throwing errors, which consumes more gas.
- After version 4.10, new error handling constructs *require, assert, and revert* statements were added, and the throw function was made absolute.

Require

- The 'require' statement declares the prerequisites for running a function, i.e., the constraints that must be met before executing a code. It takes a single argument and after evaluation returns a Boolean value.
- It also has a custom string message option. If false, then an exception is thrown, and the execution is terminated. The unused gas is returned to the caller, and the state is reset to its original setting.

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;

contract Require_Error_handling {

    // Defining function to check input
    function checkInput(uint input) public pure returns(string memory){
        require(input >= 0, "invalid uint");
        require(input <= 255, "invalid uint");

        return "Input is uint";
    }

    // Defining function to use require statement
    function Odd(uint input) public pure returns(bool){
        require(input % 2 != 0);
        return true;
    }
}
```

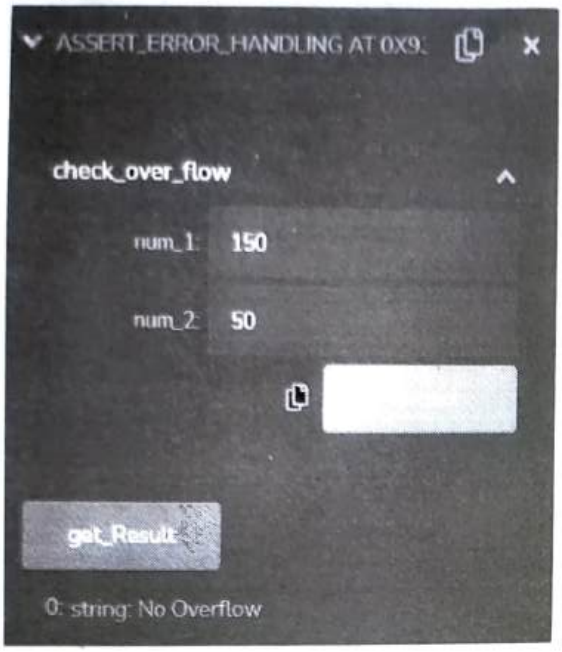
Output

- **Assert :** Its syntax is similar to the 'require' statement and returns a Boolean value. Depending on the return value, a programme will either continue or throw an exception. Instead of returning unused gas, the assert statement consumes the entire gas supply and then returns the state to its original state. Before executing the contract, *assert* is used to validate the current state and function conditions.

Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;

// Creating a contract
contract Assert_Error_Handling {

    bool result;

    // Defining a function to check condition
    function check_over_flow(uint num_1, uint num_2) public {
        uint sum = num_1 + num_2;
        assert(sum<=255);
        result = true;
    }

    // Defining a function to print result of assert statement
    function get_Result() public view returns(string memory){
        if(result == true){
            return "No Overflow";
        }
        else{
            return "There is an overflow";
        }
    }
}
```

Output



Revert

- This is comparable to the 'require' statement. It does not evaluate any condition or rely on any state or statement. It is used to throw exceptions, show errors, and revert a function call. This statement includes a string message that describes the problem with the exception's information.
- A revert statement implies that an exception is thrown, the unused gas is returned, and the state is reset to its original state. Revert is used to handle the same types of exceptions that require does, but with slightly more complex logic.

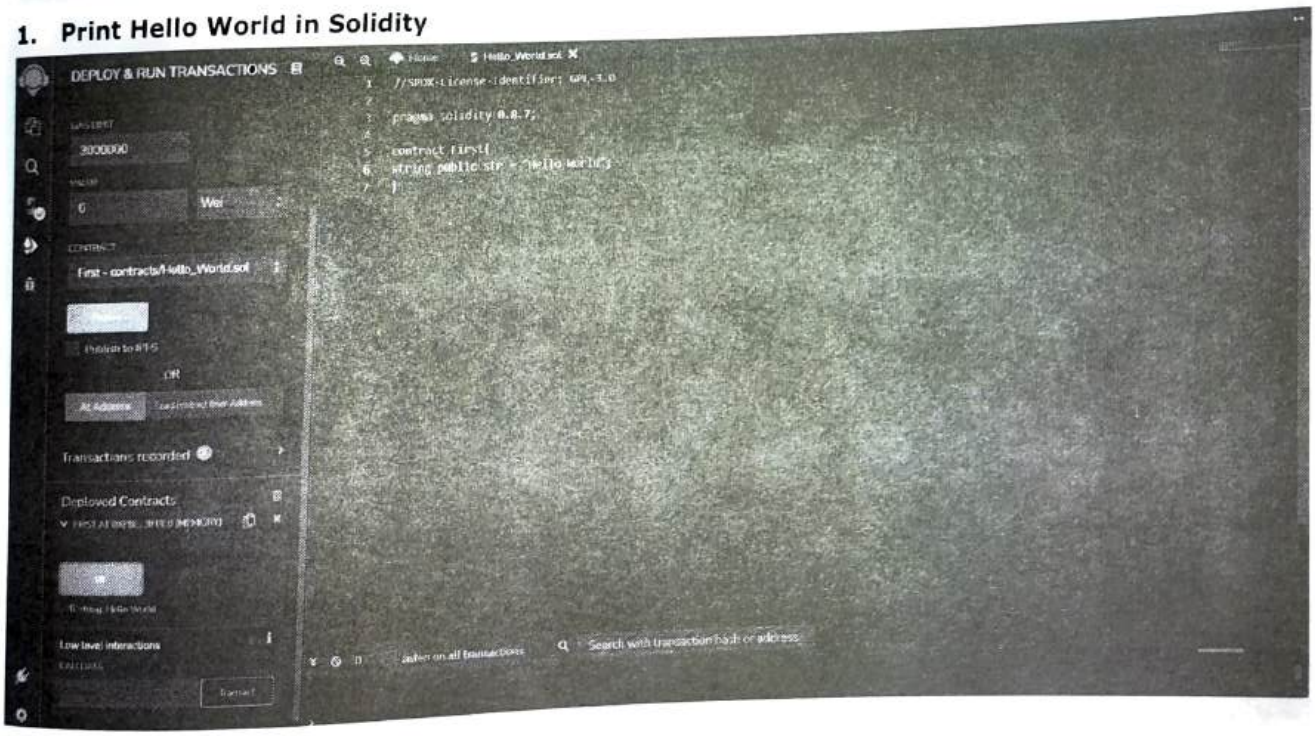
Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;


contract Revert_Error_Handling {

// Defining a function to check condition
function check_over_flow(uint num_1, uint num_2) public pure returns(string memory, uint)
{
        uint total = num_1 + num_2;
        if(total < 0 || total > 255){
            revert(" There exists an overflow");
        }
        else{
            return ("There is no overflow", total);
        }

    }
}
```

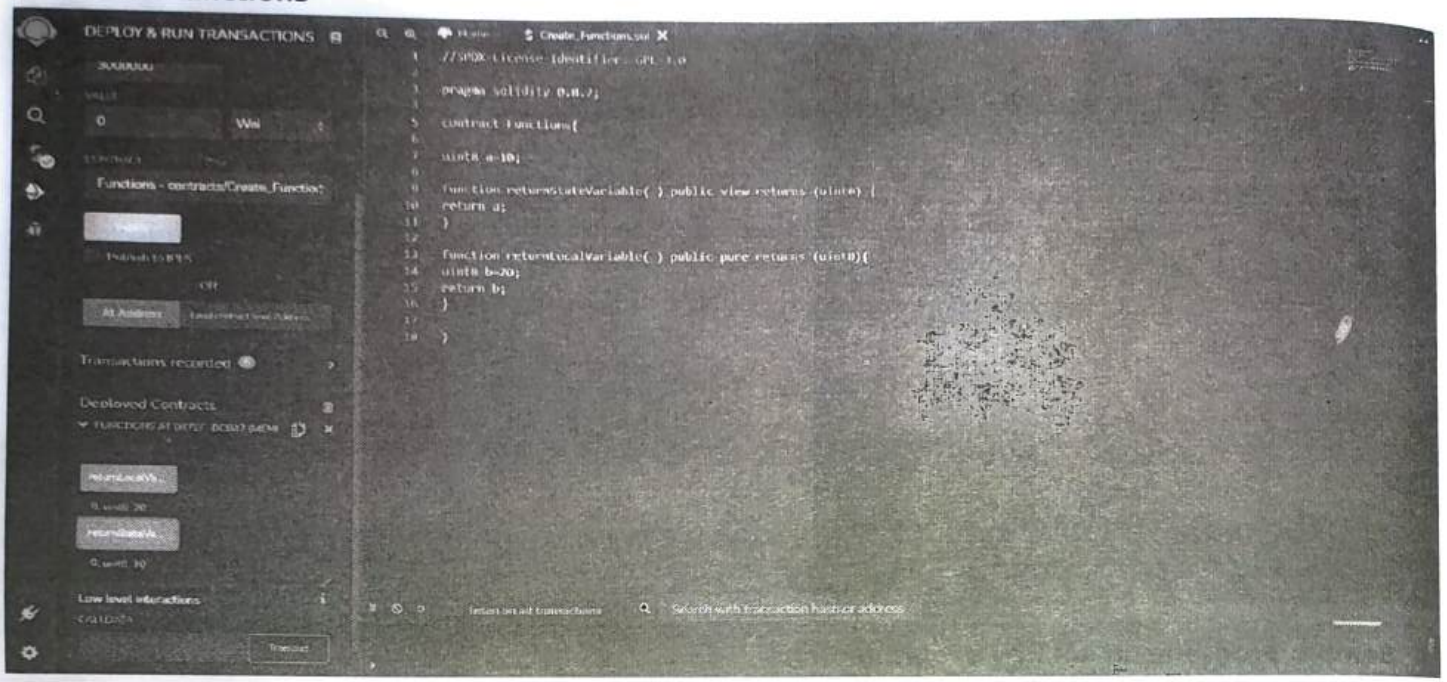
Output

Sample programs in solidity programming

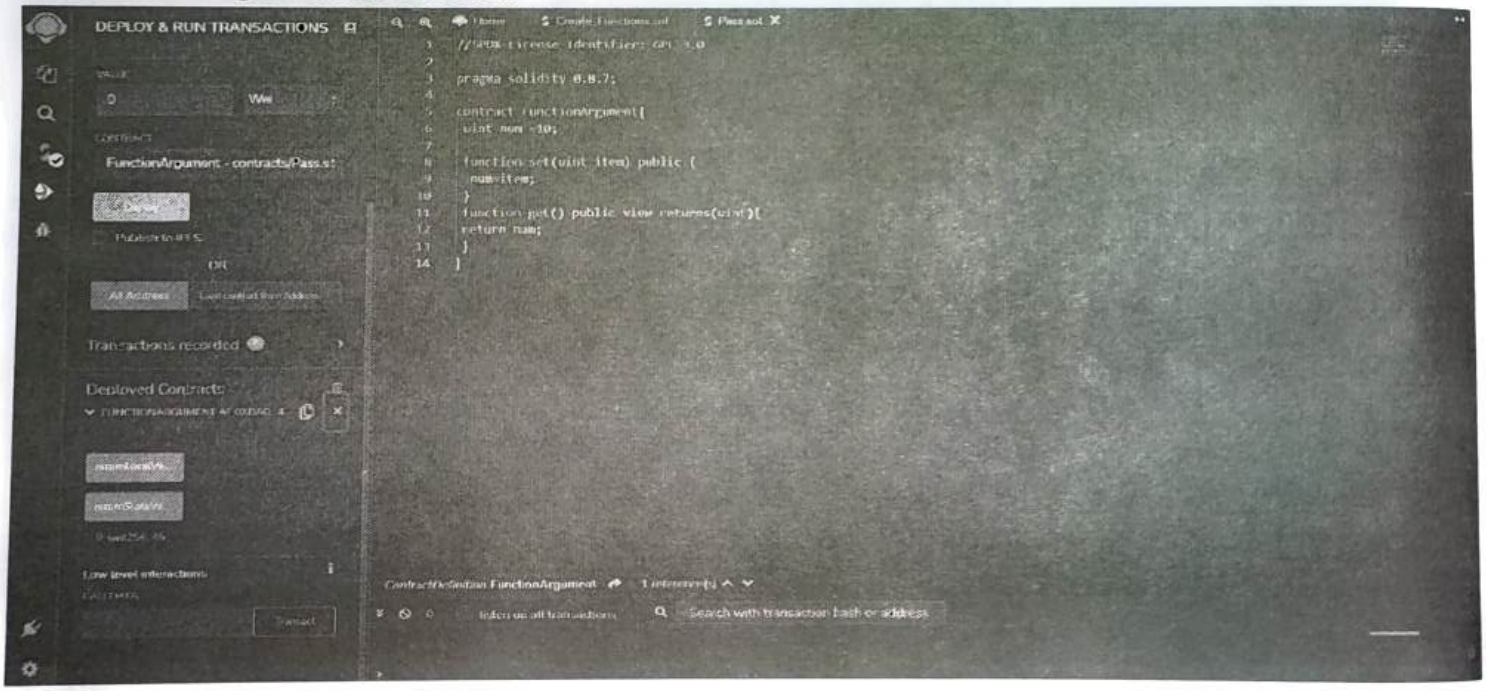
1. Print Hello World in Solidity



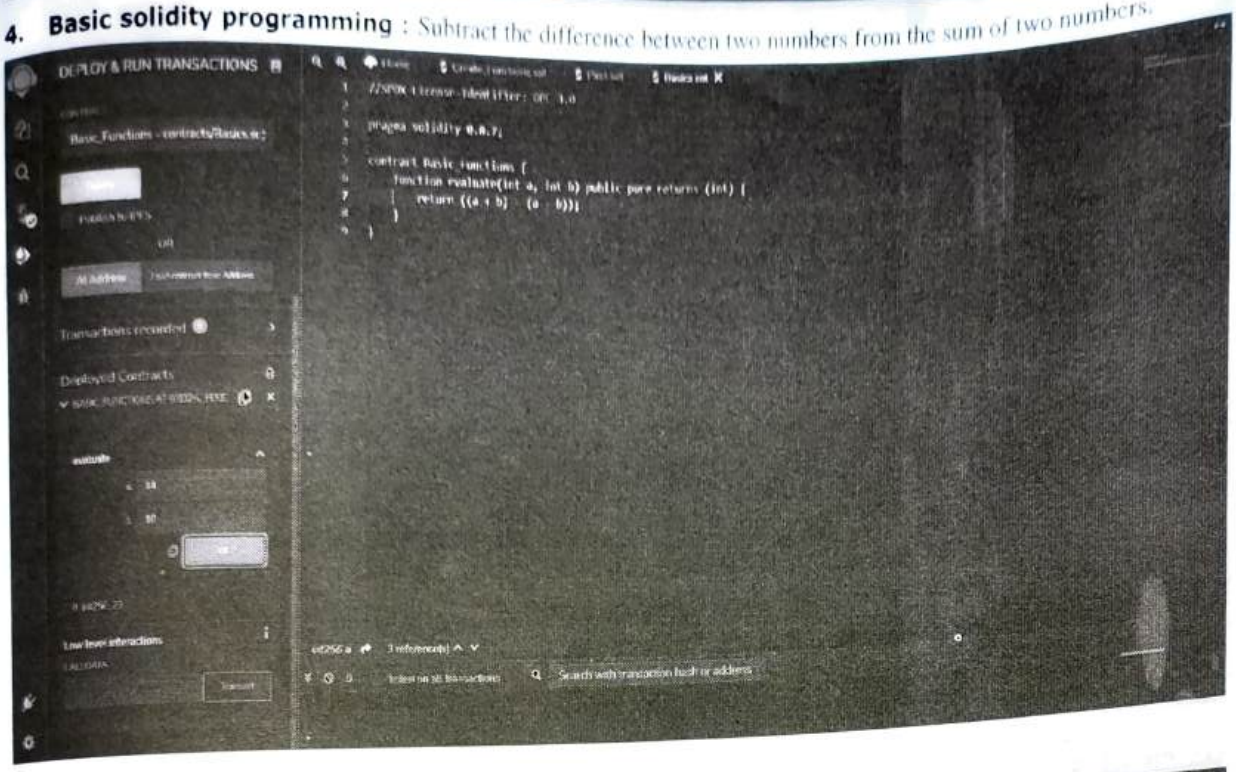
2. Create functions



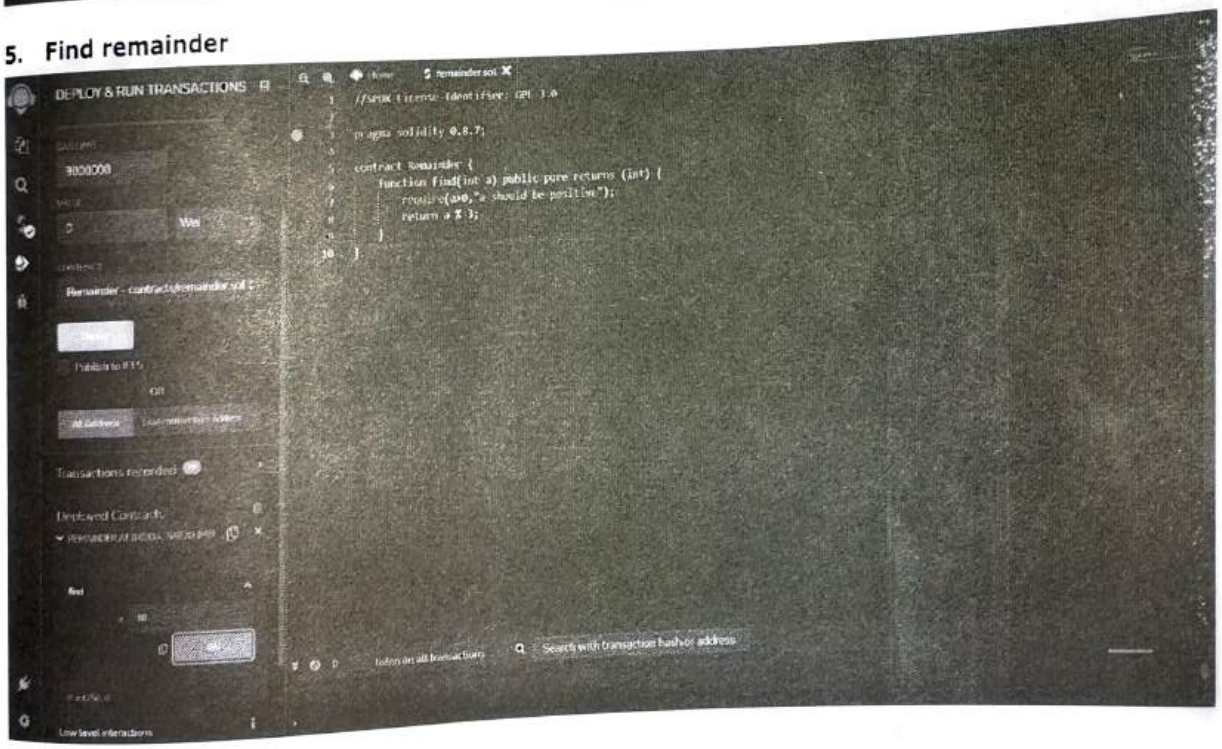
3. Pass an argument to function



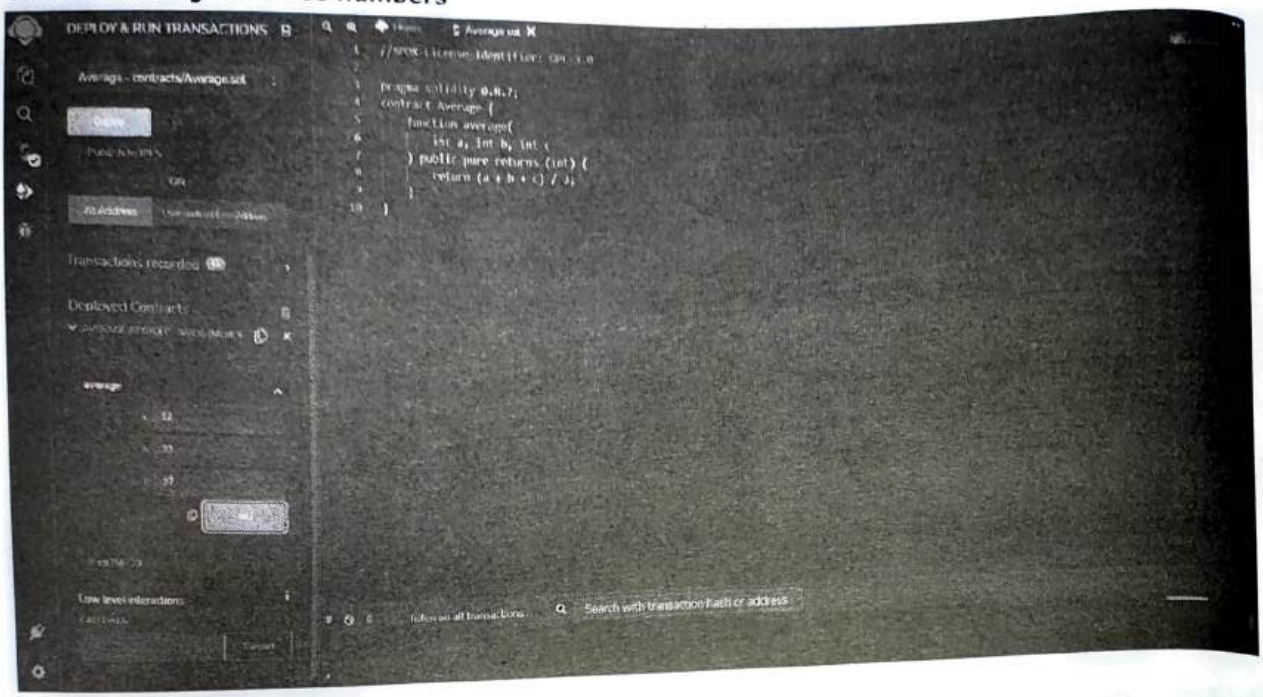
4. **Basic solidity programming** : Subtract the difference between two numbers from the sum of two numbers.



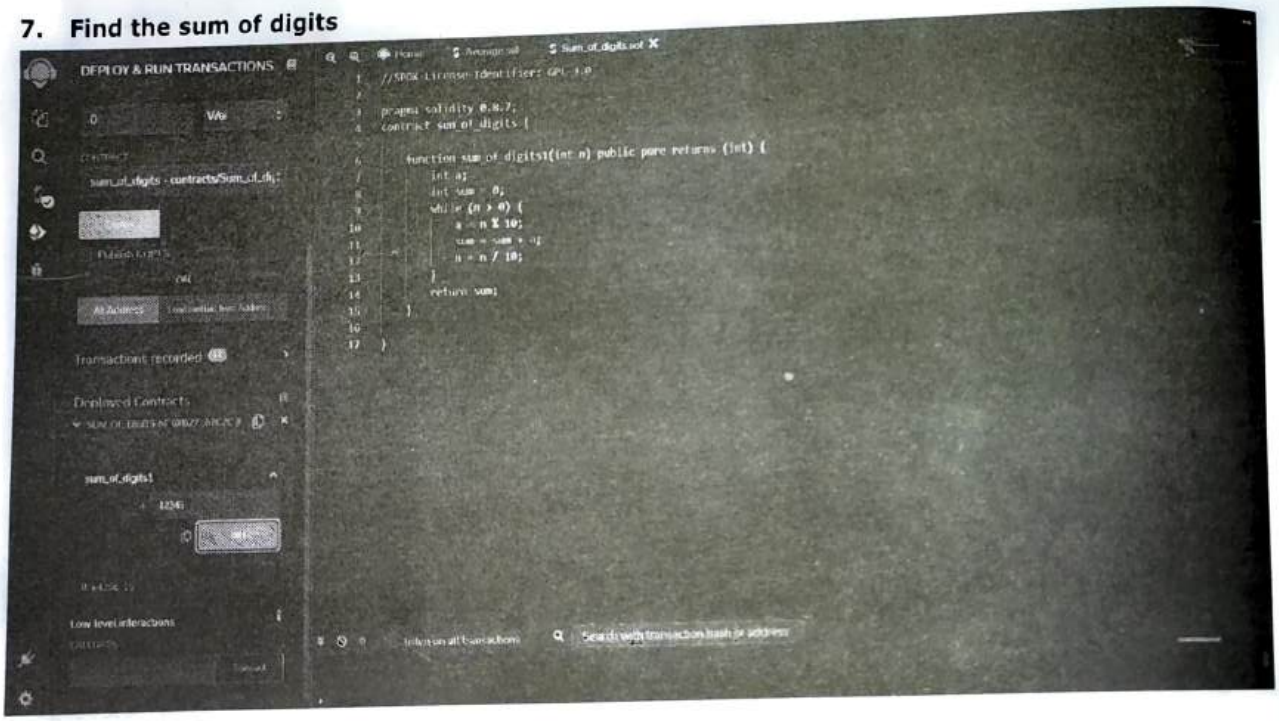
5. **Find remainder**



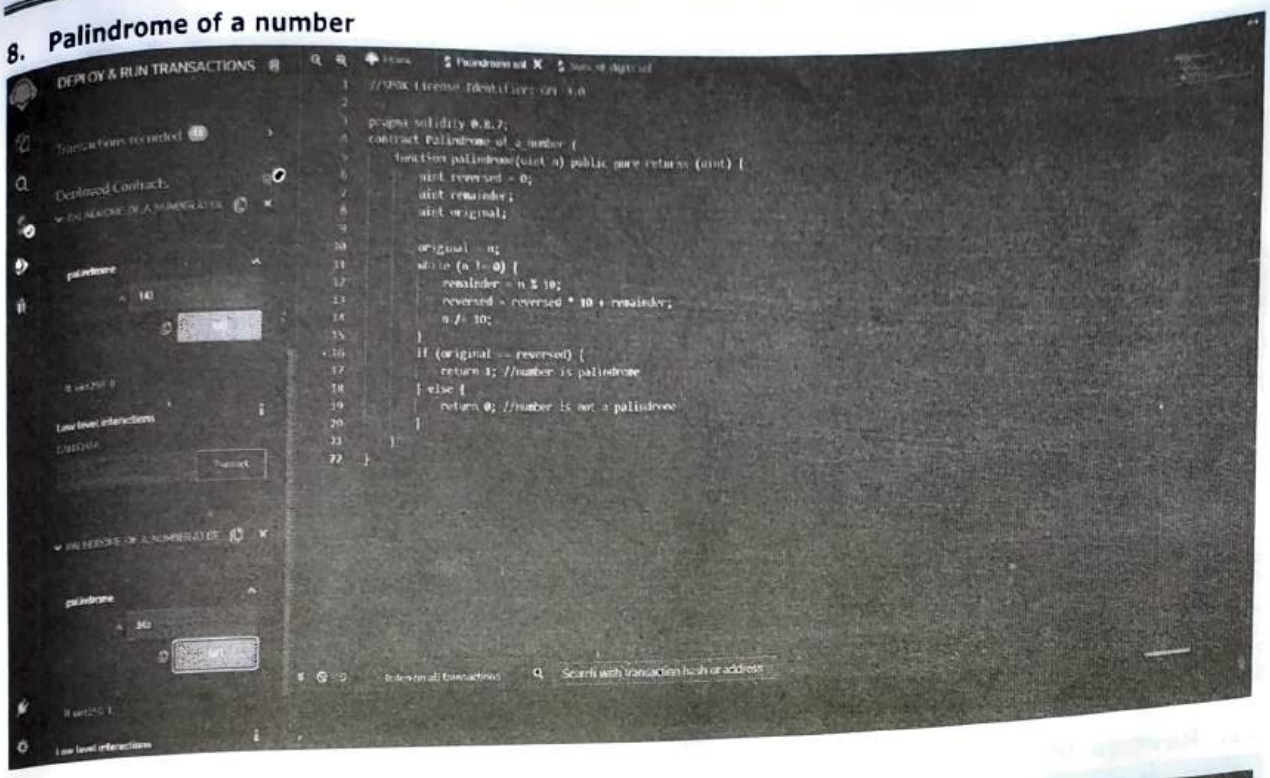
6. Find average of three numbers



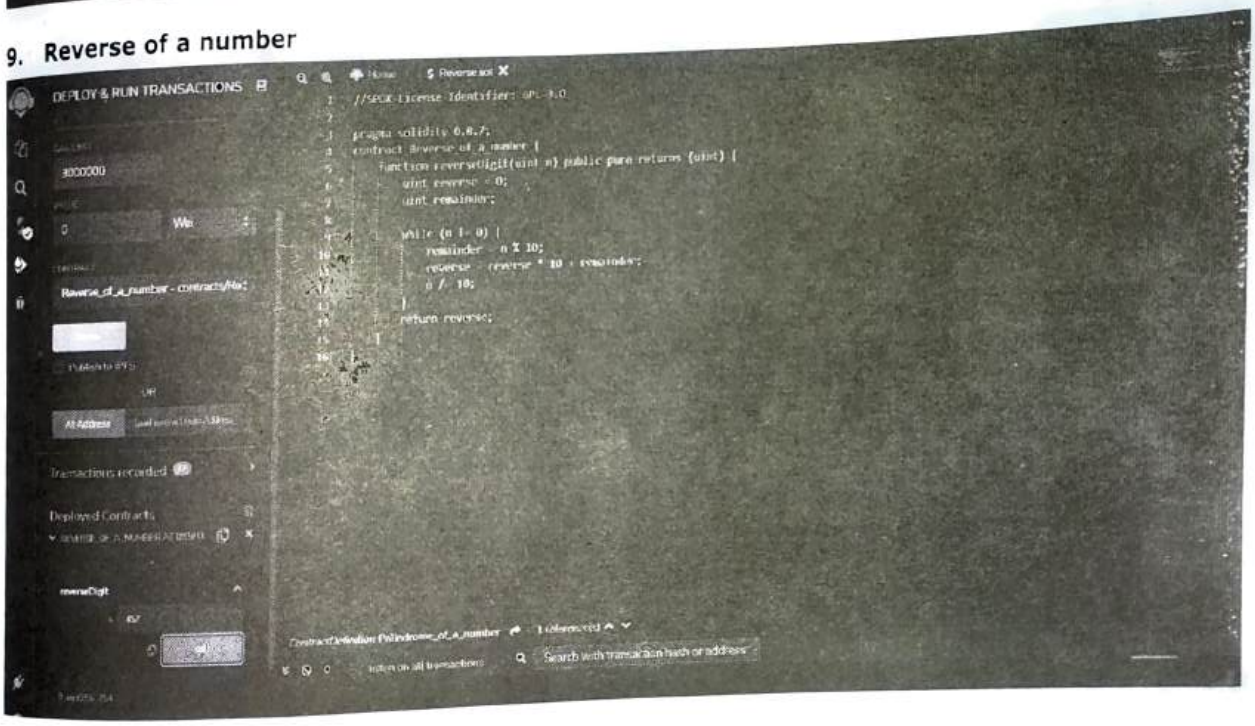
7. Find the sum of digits



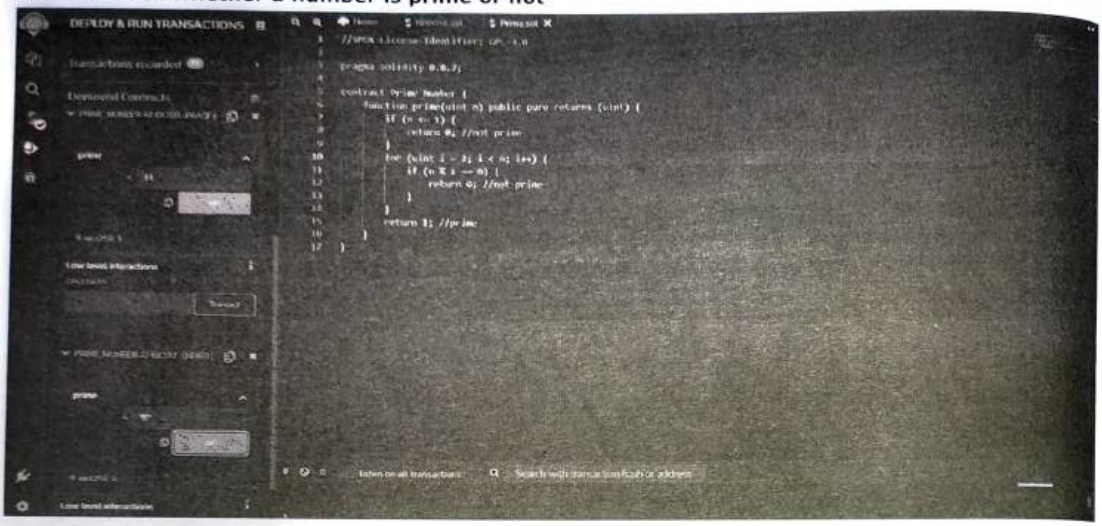
8. Palindrome of a number



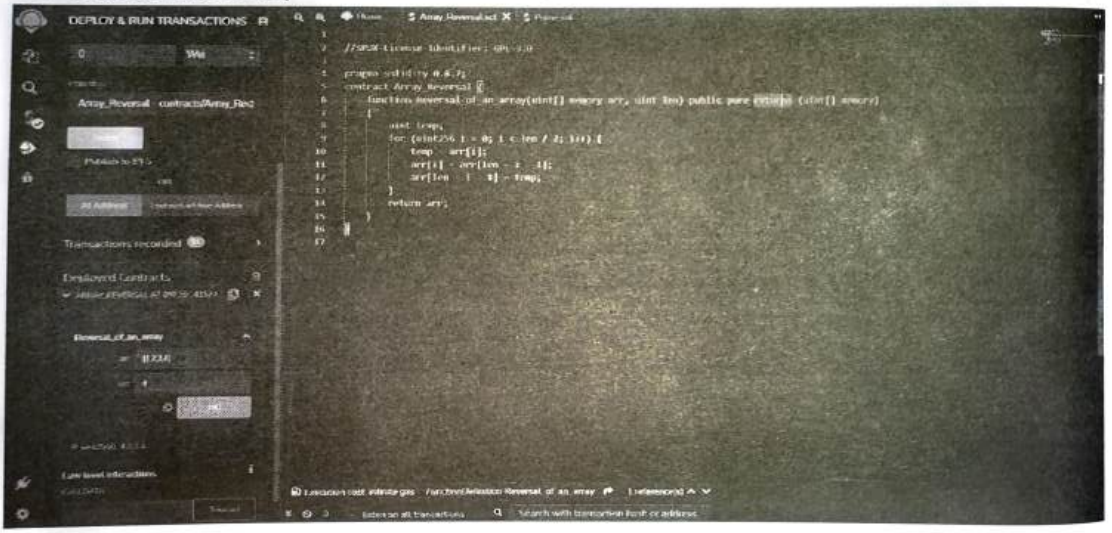
9. Reverse of a number



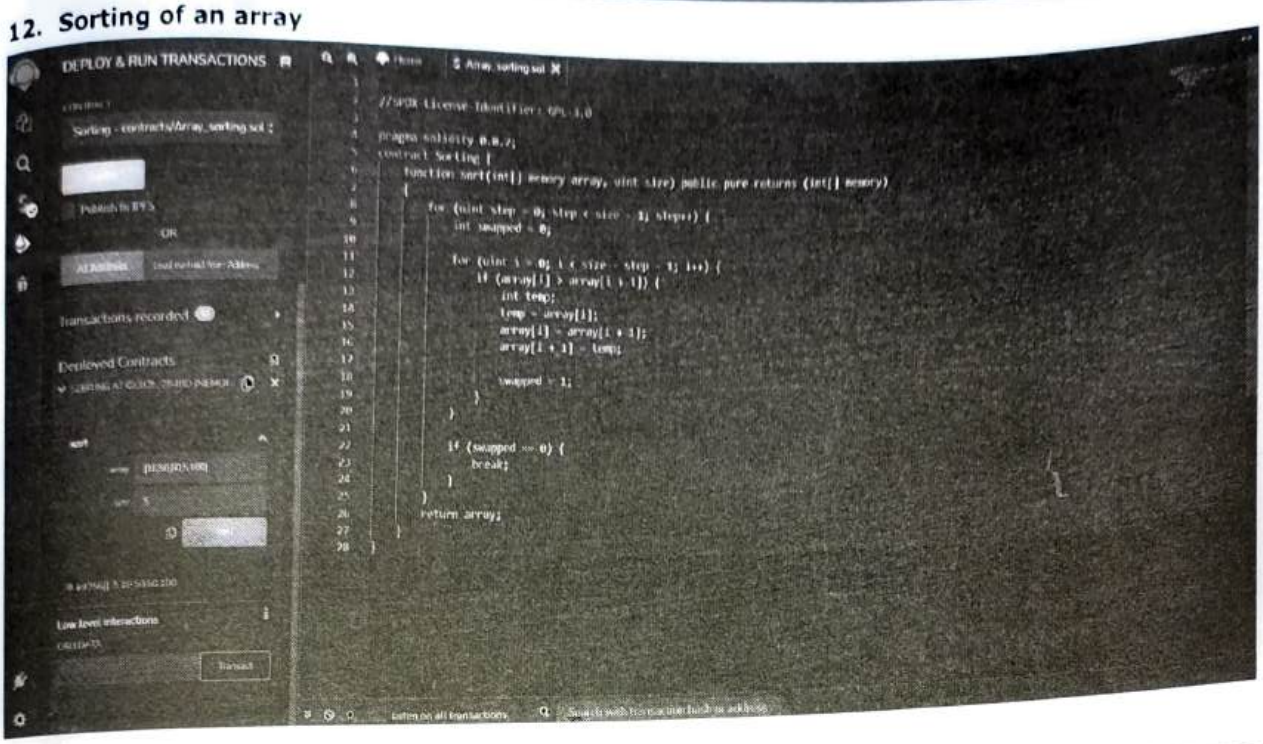
10. To check whether a number is prime or not



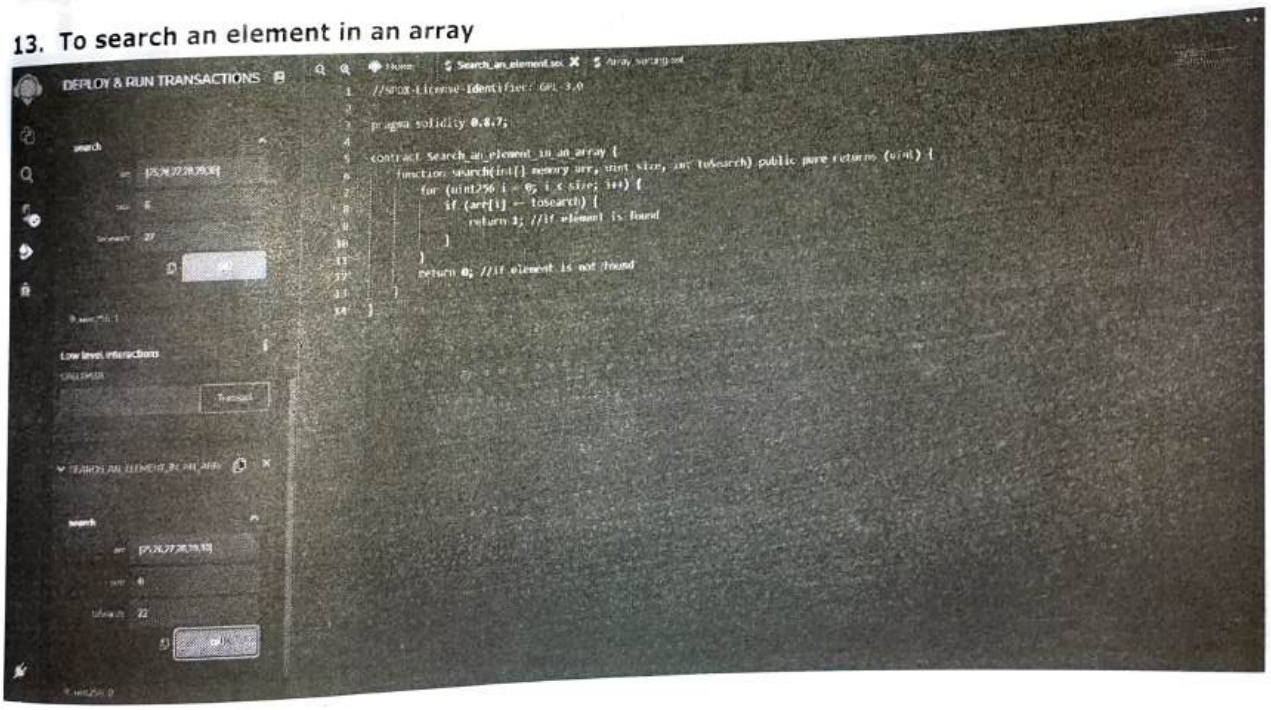
11. Reversal of an array



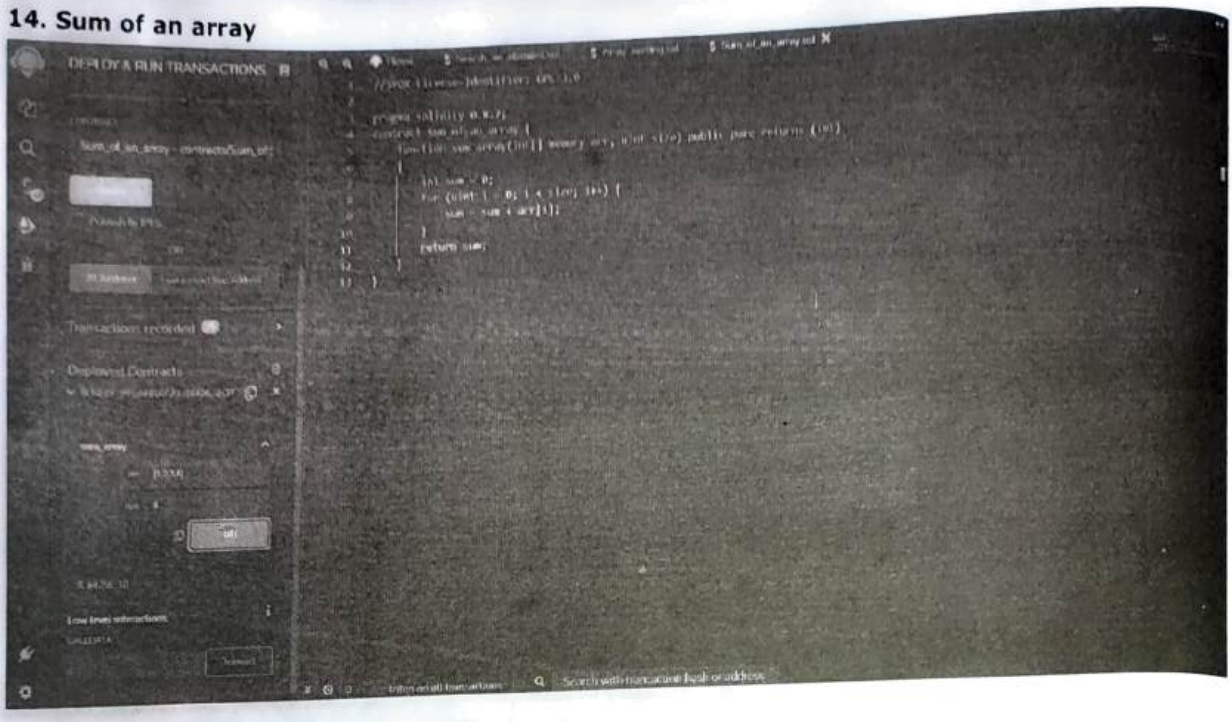
12. Sorting of an array



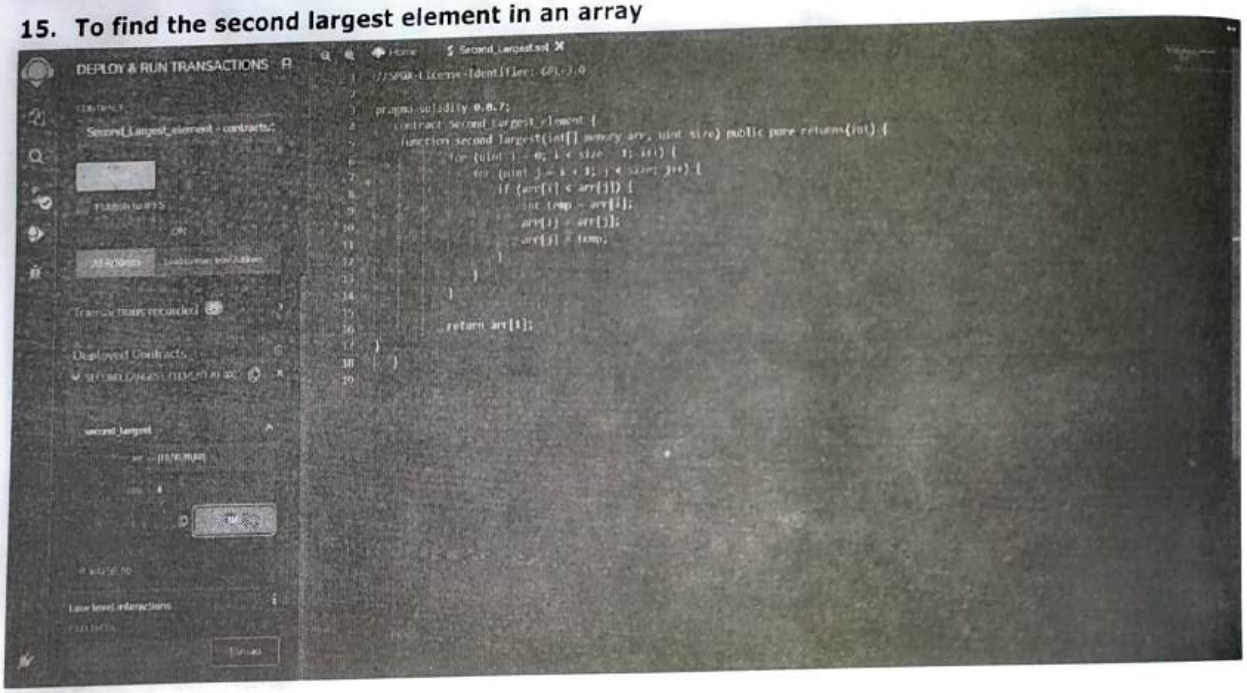
13. To search an element in an array



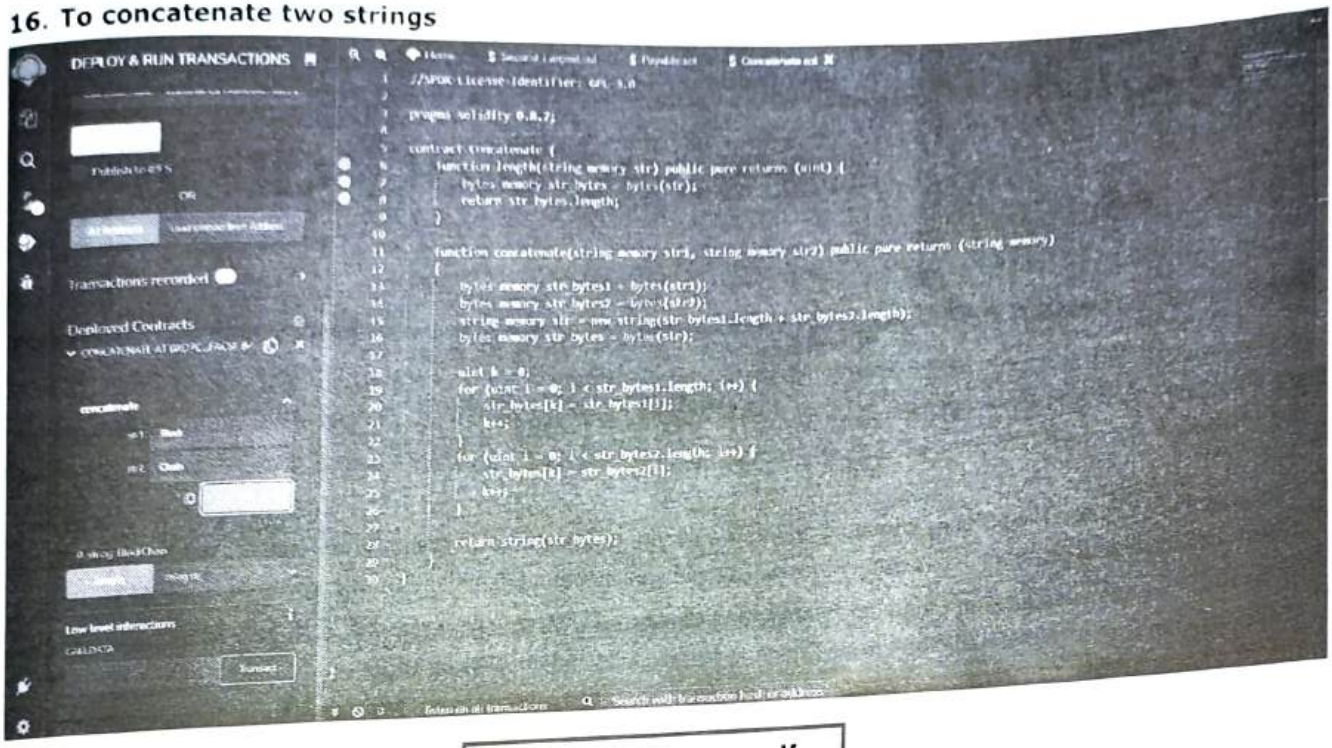
14. Sum of an array



15. To find the second largest element in an array



16. To concatenate two strings



Try it out for yourself

1. There is a series S where the next term is the sum of pervious three terms. Given the first three terms of the series a, b, and c, respectively, you have to output the n^{th} term of the series.

   S(n) = a for n=1

   S(n) = b for n=2

   S(n) = c for n=3

   S(n) = S(n-1) + S(n-2) + S(n-3) for n>3

   Create a function n_th_term(uint n, uint a, uint b, uint c) where n is the n^{th} term to find and a, b, and c are the three terms of the series.

2. ***X raised to Y :*** Create a function power(uint x, uint y). This power() will calculate x raised to the power of y and return it.

3. Create a function even(array, length of array). This even() will take two arguments, i.e., a dynamic uint type array and length of the array. The even() will multiply each element of array with 2.

4. Create a function distinct(array, length of array). This distinct() will take two arguments, i.e., a dynamic uint type array and length of the array. The distinct() will return the number of distinct elements in an array.

5. Find the sum of the series $1 + x + x^2 + x^3 + \ldots + x^n$. Create a function expression(x, n).The expression() will find the sum of the above expression.

6. Create a function hcf(num1, num2). This hcf() will take two arguments uint type number1 and number2.The hcf() will find the of number1 and number2.

Chapter Ends...

❑❑❑

Module 4

Public Blockchain

Syllabus

Introduction to Public Blockchain, Ethereum and its Components, Mining in Ethereum, Ethereum Virtual Machine (EVM), Transaction, Accounts, Architecture and Workflow, Comparison between Bitcoin and Ethereum

Types of test-networks used in Ethereum, Transferring Ethers using Metamask, Mist Wallet, Ethereum frameworks, Case study of Ganache for Ethereum blockchain. Exploring etherscan.io and ether block structure

4.1 PUBLIC BLOCKCHAIN SYSTEM

GQ. What is a public blockchain ? State some characteristics of public blockchain.

4.1.1 Introduction

Blockchain is one of the top ten strategic technologies of 2019. As per International Data Corporation (IDC), global expenditure on blockchain solutions is anticipated to reach 9.7 billion dollars in 2021. Blockchain can either be public or private based on whether a network is open or accessible to anyone with an Internet connection. In public blockchain, anyone can join a network, and the users in a private blockchain are unidentified. Blockchain users can download a copy of the ledger, initiate, broadcast, or mine blocks.

Public Blockchain

In a public blockchain, one doesn't need any permission to initiate or read/access a transaction, or participate in a consensus process in order to create a block. There is anonymity maintained in a public blockchain with the help of high cryptographic protocols.

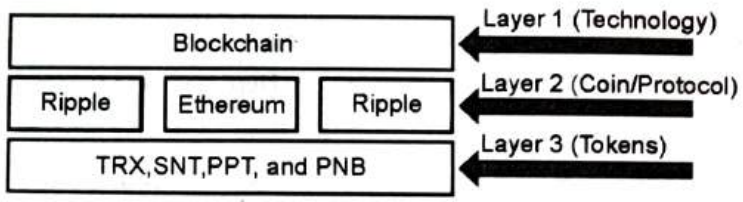
Some of the main characteristics of a public blockchain are as follows :

- **Type of organization :** Public
- **Approach :** Open and visible to all
- **Operations :** Anyone can read, initiate, or receive transactions
- **Immutability :** Secured via hashing
- **Trust :** It is a trust-free system
- **Users :** Anonymous
- **Type of network :** Decentralized
- **Verification :** Any node can participate in the consensus process to validate transactions and create a block
- **Speed of transactions :** Slow
- **Energy consumption :** Very high
- **Scalability :** It has limited scalability, which mainly depends on the growth of the network (i.e., the number of nodes in a network). Other factors could be bandwidth, storage, and an exponential increase in computational power.

4.1.2 Blockchain layers

GQ. Explain the relationship between different blockchain layers.

Relationship between blockchain, cryptocurrency, protocols, and tokens can be comprehended with the help of different layers in a blockchain.



(1B1)**Fig. 4.1.1 : Blockchain layers**

- **Layer 1 :** Blockchain is the technology at the back of a variety of cryptocurrencies such as Bitcoin, Ethereum, Ripple, etc.
- **Layer 2 :** This layer not only defines several coins but also focuses on the protocol of the respective currencies. A protocol is a set of rules that permit people to communicate or transact data with each other to not only reach consensus but also validate transactions within a given network. The protocols used are HTTP, HTTPS, and TCP/IP. Protocols in a blockchain define various consensus and signature mechanisms, use of public keys, and rules related to cryptography.

- **Layer 3 :** It comprises tokens. Tokens serve as the symbol for an underlying contract. For instance, Ethereum has 100s of tokens (i.e., TRX, SNT, PPT, and PNB). Bitcoin and Ripple have no tokens, and hence, they are unable to create smart contracts.

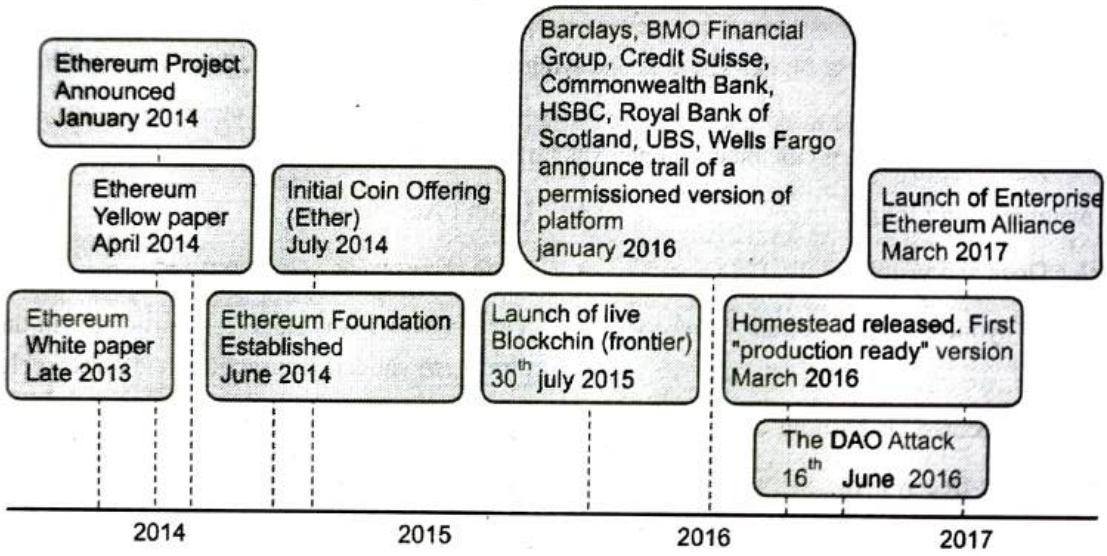
4.1.3 Ethereum Blockchain

4.1.3.1 Brief History of Ethereum

GQ. Explain in brief the history of Ethereum.

Ethereum could be a "do it yourself" platform for most decentralized programs/applications (i.e., DApps). Ethereum platform includes several computers that are completely decentralized. Once a program is setup in an Ethereum system, the computers (nodes) are likely to make sure it implements as composed. Ethereum's programming terminology, i.e., solidity programming, could be used to publish smart contracts, which can be used as a logic to execute DApps.

Ethereum blockchain is a design comprising multiple components that interact and function with each other.



(1B16)**Fig. 4.1.2 : Brief history of Ethereum**

Miner and mining nodes

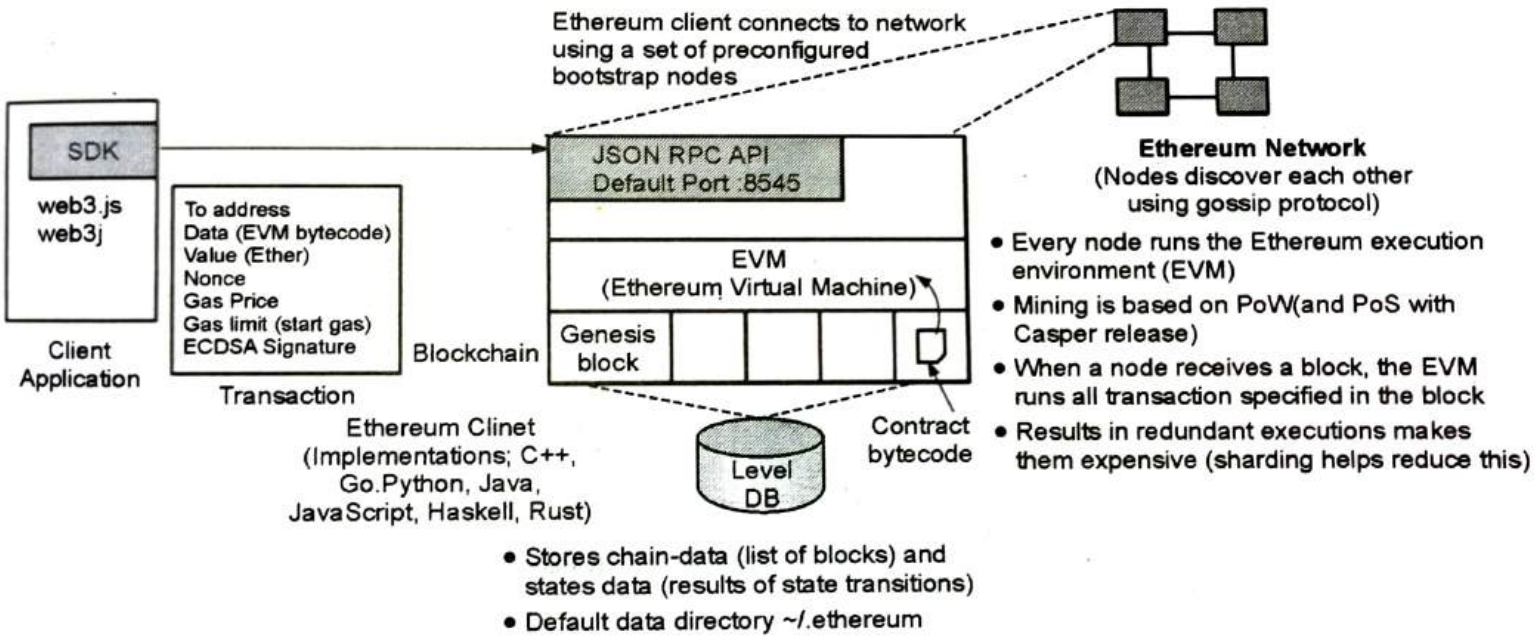
- A miner is responsible for writing a transaction to an Ethereum series. He/she receives a reward when paper transactions are added to a ledger. Here, miners receive two types of rewards, namely benefits for writing a block into a series and accumulative gas price from all transactions in the block.
- Every miner needs to have a backup of the transactions, which includes features like code and other things that exist within the transactions. Data is not stored directly in the blockchain as it is too big. Data is stored in a distributed hash table like InterPlanetary File System (IPFS).
- IPFS provides a hash or content address for the entire content, which helps in retrieving a file from a data storage system. Thus, hash points of the data are stored in the blockchain. There are many miners available in a blockchain network who compete and try to write transactions.
- A single miner can write a block into the ledger, whereas the other miners may not be able to write the block in an existing blockchain, and the conclusion of a miner who will who will eventually compose the block is a challenge. The problem is awarded to each node and every miner attempts to solve the same problem available computing power.

- Lastly, the miner who solves the problem is allowed to write the block comprising transactions to the ledger and receives fiveethers (5 ETH) as a reward. In an Ethereum network, there are two types of nodes:
  - **Mining nodes :** Nodes that belong to miners.
  - **Ethereum virtual machines (EVMs) :** They have a unique code attached to it, which is called as a smart contract.

4.1.3.2 Architecture of Ethereum

GQ. Describe the architecture of Ethereum.

- Every Ethereum node runs an EVM and all the smart contracts run within this virtual machine. Client applications can connect to an Ethereum client using a web3j SDK. Transactions have a specific format.
- It has "to" address (i.e., a recipient's address), data pertaining to the smart contracts, Ether value that will be transferred to the recipient, nonce, gas, gas limit, and the signature by a transactor (i.e., ECDSA signature). So the user who is sending the transaction will sign the transaction and send it to blockchain.
- Note that Ethereum will transfer ether as well as execute a smart contract function and that smart contract can hold data on the blockchain.

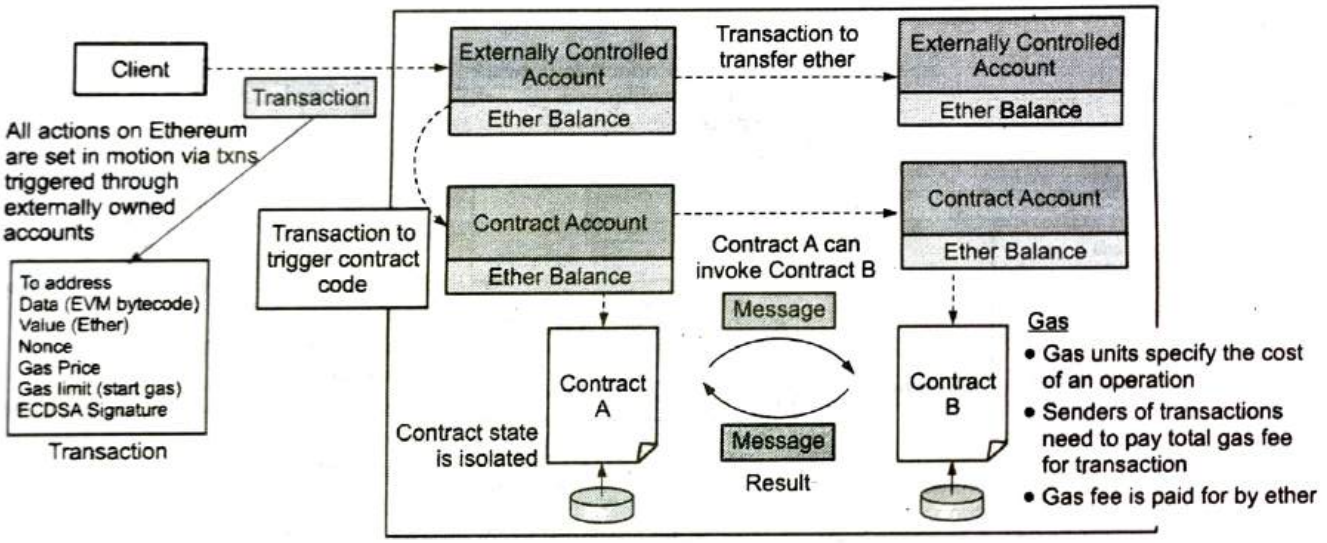


(1B17)**Fig. 4.1.3 : Ethereum Architecture**

- There exists an Ethereum network of many such nodes that interact with each other and these nodes are the ones that run consensus and determine which transaction blocks should be added next to the blockchain. The consensus is performed through PoW similar to bitcoin.
- When a node receives a block, an Ethereum client will execute all transactions in the block in a sequence and it will then update its ledger once the block is added. It is possible that the client receives the block from multiple people (i.e., redundant executions), but sharding would reduce this problem.
- The database that is used by Ethereum is level DB, which has a key and value store available to smart contracts to store state information. Moreover, the contract bytecode is stored on the blockchain. The state data and list of blocks are stored on the blockchain.

4.1.3.3 Account Types, Gas and Transactions

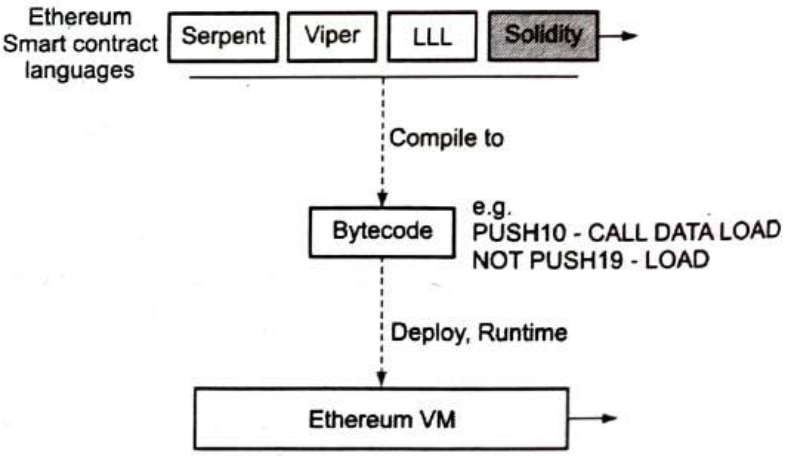
- Ethereum has two types of accounts, i.e., externally controlled account and contract account. Ether balance is stored in the externally controlled account (i.e., a user owns this account). It is possible to transfer ether from one externally controlled account to another externally controlled account.
- Every smart contract will have a contract account and it has an address that is similar to an externally controlled account. Note that a contract account specifically refers to a smart contract and it does not refer to an externally controlled account. For transactions to have an externally controlled account as well as a contract account, ethers can be transferred from one account to the other.
- A specific smart contract can be invoked specified at a particular address and that smart contract will run a particular function defined in the smart contract and that function can update state information on blockchain. Apart from invoking one contract, other contracts can also be invoked. There can be a transfer of balances as well. Invocation of smart contract functions incurs a transaction cost.
- As this is a public permissionless system, there is a need to ensure that people do not spam the network with invalid transactions, so in Ethereum, there is a something called as a gas. So for each transaction, some gas needs to be spent. Until the gas price is paid, a transaction cannot be executed. Every transaction will have a specific gas price. Also, a gas limit can be set so that there is no depletion of gas. The gas fee is paid using ethers. The gas price is borne by the sender of the transaction.



(1B18)**Fig. 4.1.4 : Account types, gas, and transactions**

4.1.4 Ethereum Smart Contracts

- Ethereum allows smart contracts to be written in different languages like Serpent, Viper, LLL, and Solidity. The most popular one is solidity. Solidity is an object-oriented high-level language for writing EVM-based smart contracts. It has a syntax similar to JavaScript. It is executed inside an EVM.
- So, the languages are compiled into a standardized bytecode, and the bytecode runs in an EVM. It offers a support for multiple inheritance. It also supports complex user-defined types through structs.



(1B19)**Fig. 4.1.5 : Ethereum smart contracts**

- The programming languages that are defined for Ethereum smart contracts is to reflect the limitations of EVM. For example, basic string manipulation is not supported in a language, but it gets supported through libraries.
- There exits a browser-based IDE with an integrated compiler, and solidity runtime environment can run without any server-side components.
- One of the main aspects in Ethereum blockchain is "*code is law*", which means that there is no auditability or governance or authority who will oversee/administer the code in the blockchain. Whatever the code is, it will be executed in a decentralized manner.

☞ Ethereum Virtual Machine

GQ. What is an Ethereum virtual machine (EVM) ?

- It is a 256 bit virtual machine that is Turing complete, and the computation is intrinsically bounded through gas. It allows execution of EVM bytecode. EVM defines approximately 70 OPCODEs. The set of OPCODEs are interpreted by an interpreter. There exists a JIT interpreter the compiles the bytecode into manageable datatypes and structures.
- Gas is defined for all 70 OPCODEs, and the gas value is set based on the difficulty of the operation (e.g., IF, AND, MUTIPLY, and SEND), which is paid using ether.

[Example of a solidity code]

```
pragma solidity ^0.4.4;
import "KVStore.sol";
contract CustomerManager{
  KVStore private kvStore;
  struct Customer{
    byte32 customerId;
    byte32 name;
  }
  mapping(byte32 => Customer)customers;
  modifier customerNotExist(byte32 customerId){
    If (hasCustomerId(customerId)) throw;
    _;
  }
  event NewCustomer(byte32 indexed customerId, byte32 name);
  function createNewCustomer(byte32 customerId, byte32 name)
           customerNotExists(customerId) {
    //_
    Customers[customerId] = Customer(customerId, name)
    NewCustomer(customerId,name);
  }
  function getCustomer(byte32 customerId) constant
           returns (byte32 name, byte32 customerId ) {}
}
```

Reference to another smart contract

Modifier functions:
Ensure execution proceeds only if
Pre-conditions are met

Define Transaction Event

File Transaction Event

Constant keyword
Signifies read operations

Indicate return types (and values)

☞ Comparison of Bitcoin and Ethereum

GQ. Compare Bitcoin and Ethereum.

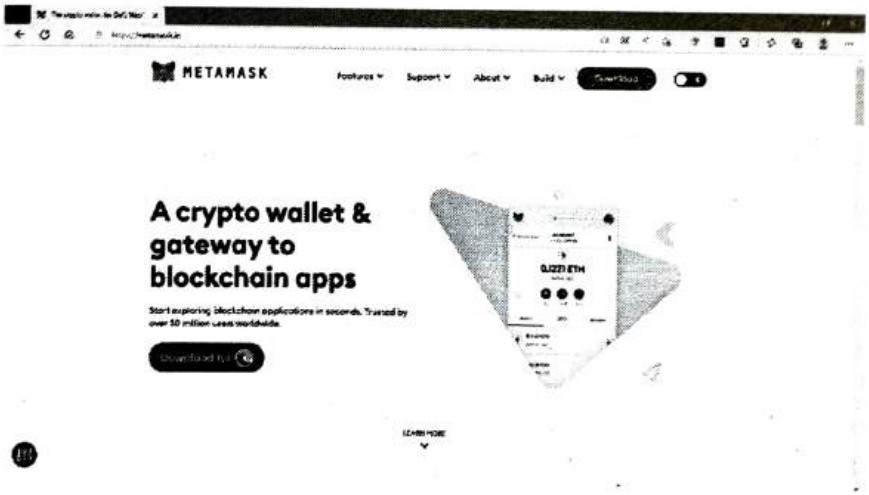
Sr. No.	Parameters	Bitcoin	Ethereum
1.	Founder	Satoshi Nakamoto	Vitalik Buterin and others
2.	Purpose	Cryptocurrency	Network Software
3.	Release date	January 2009	July 2015
4.	Average block time	~10 minutes	~15 seconds
5.	Coin symbol	BTC	ETH
6.	Tokens	Not available	Available
7.	Monetary policy	Hardcoded	Non-hardcoded
8.	Emission rate	Halving policy followed	Occasional
9.	Backward compatibility	Available	Not available
10.	Block limitation	1 MB per block	No limit

4.2 METAMASK

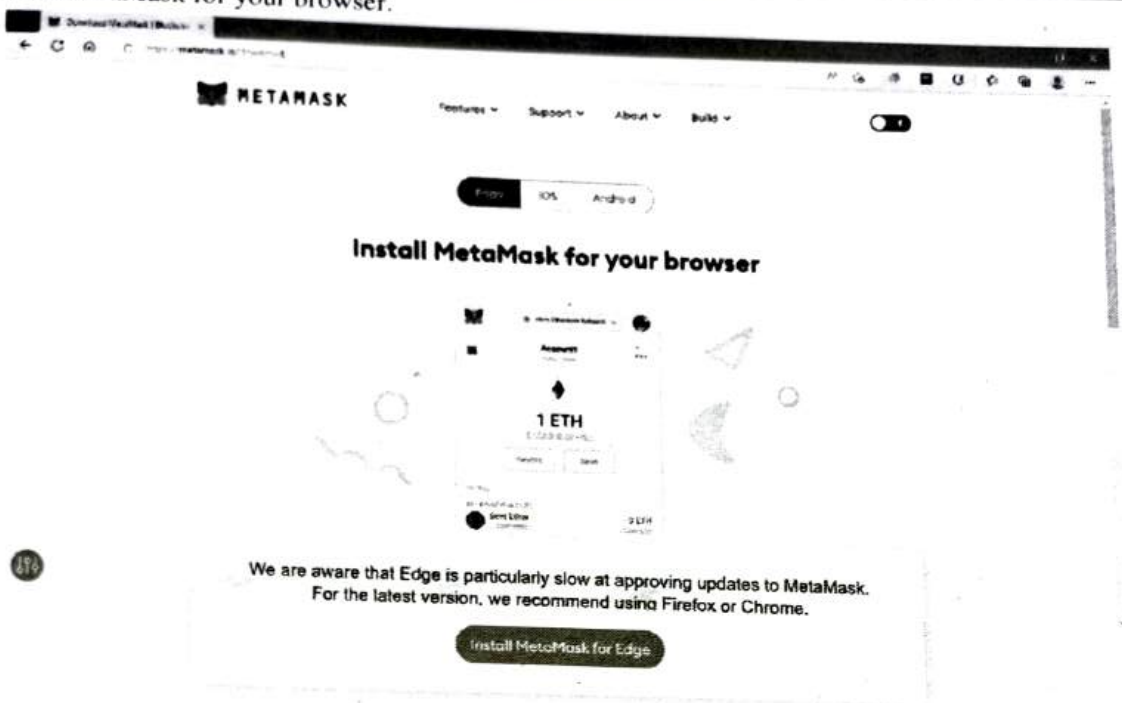
Metamask is a browser plugin that hold an Ethereum wallet and connects the computer with the Ethereum network. Metamask can connect and network with other providers who offer free ethers (i.e., test ethers) to accounts created in wallets of Metamask. For instance, an account created in Metamask can be connected with an external network named Ropsten Test Network, which can then inject free ethers (ETH) to a corresponding account.

Setting up Metamask account

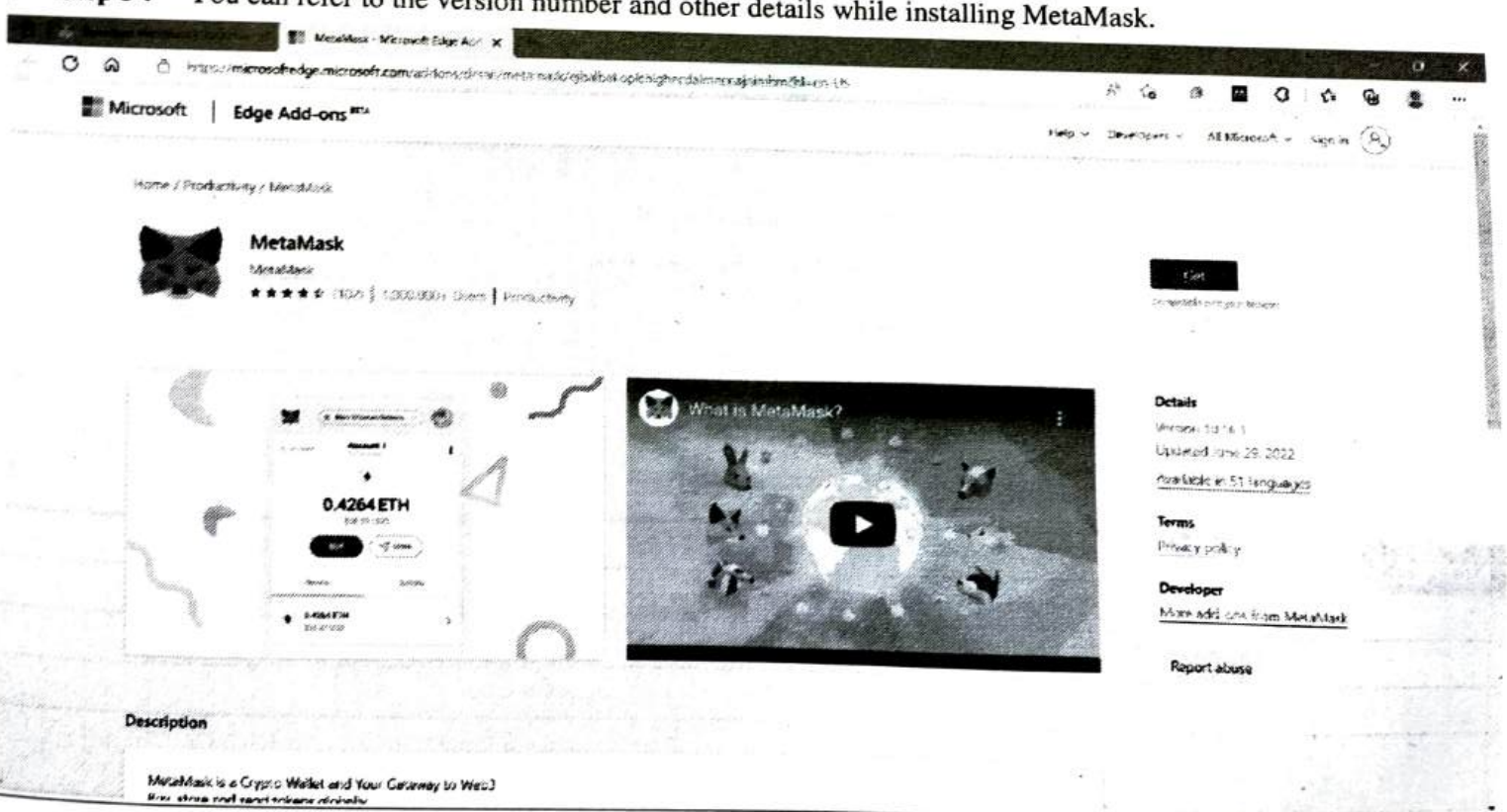
- **Step 1 :** Go to https://metamask.io/ and download MetaMask, which is a crypto wallet and gateway to blockchain applications



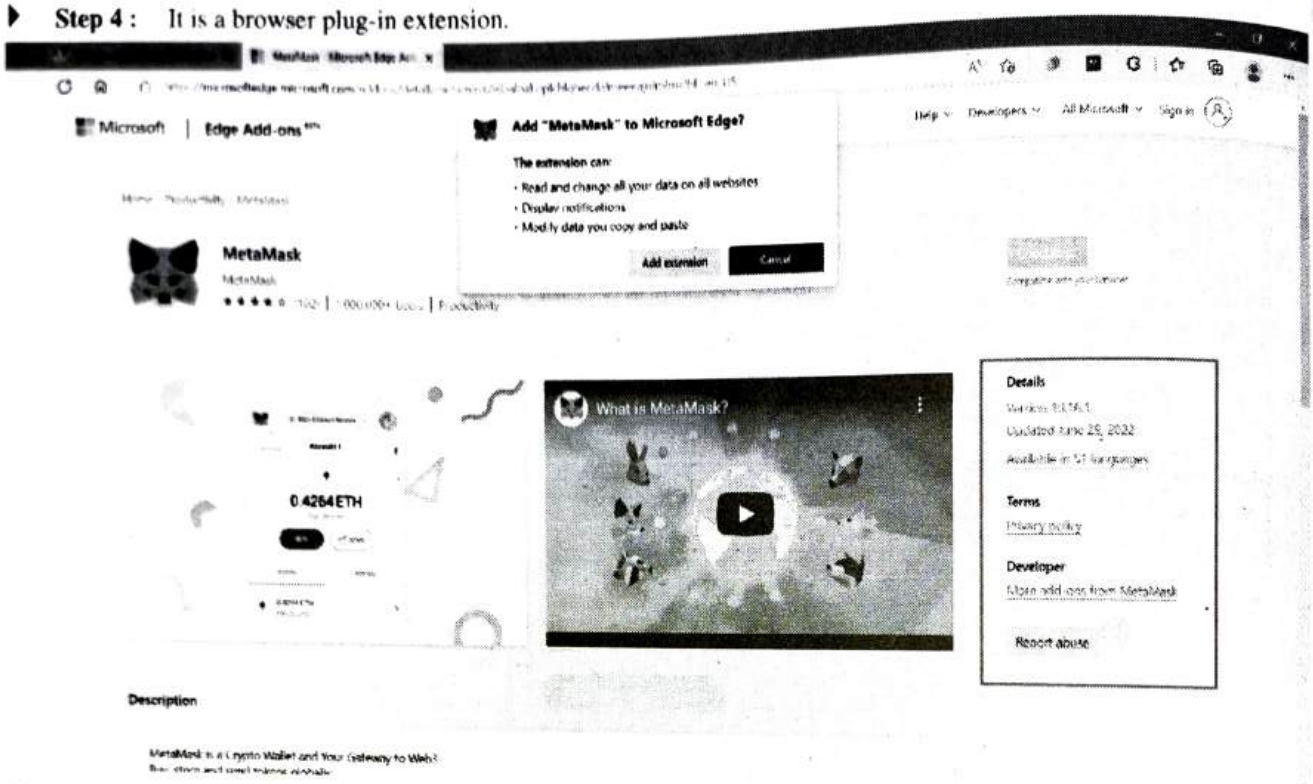
- **Step 2 :** Install MetaMask for your browser.



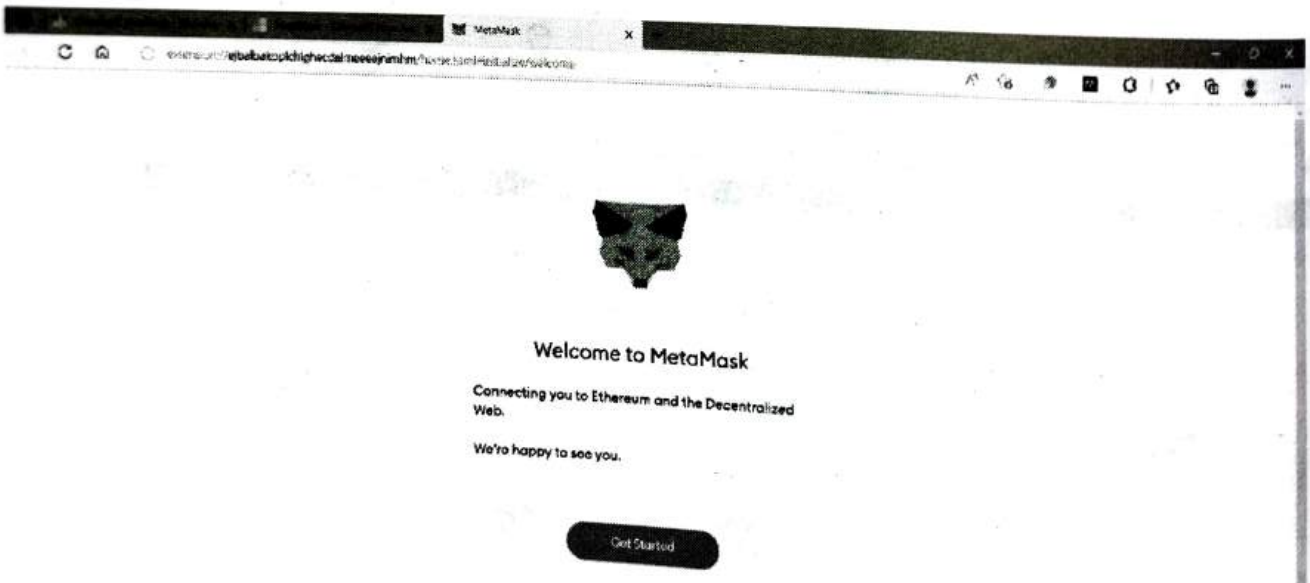
- **Step 3 :** You can refer to the version number and other details while installing MetaMask.



- **Step 4 :** It is a browser plug-in extension.

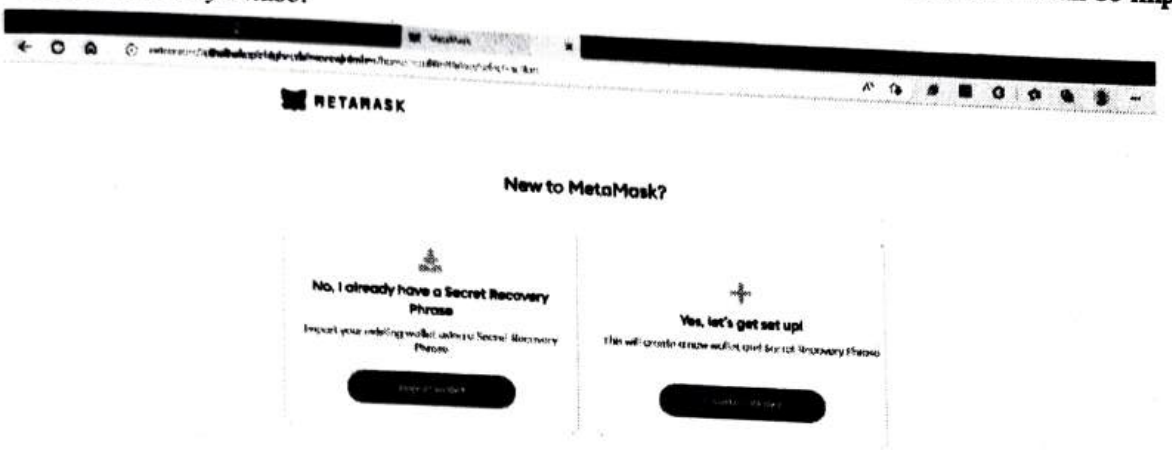


- **Step 5 :** MetaMask has been successfully installed, which connects us to Ethereum and the decentralized web.

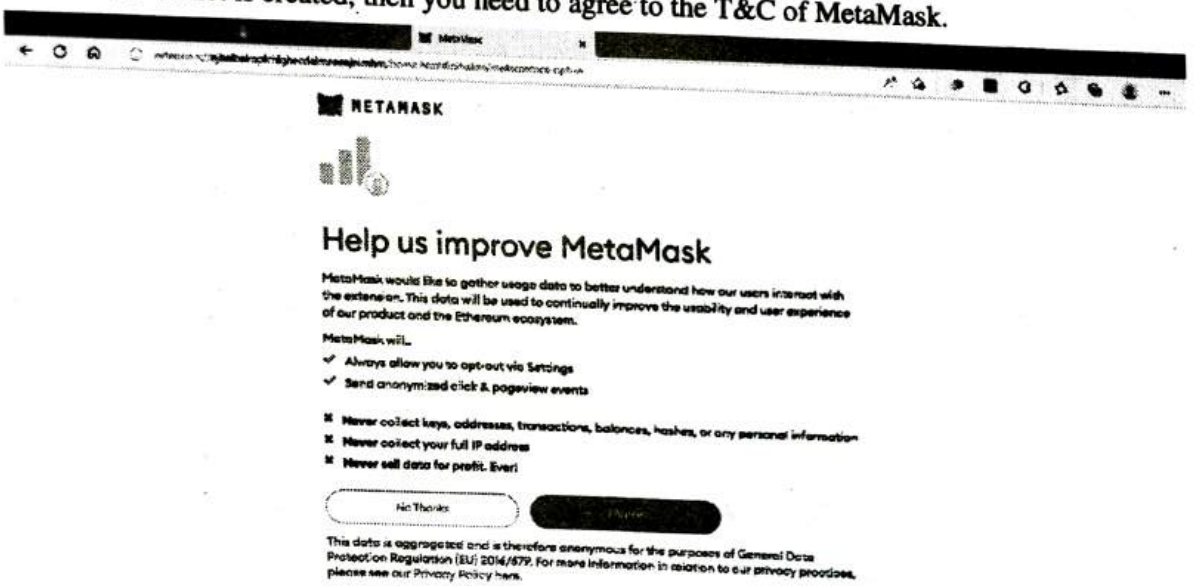


NOTES

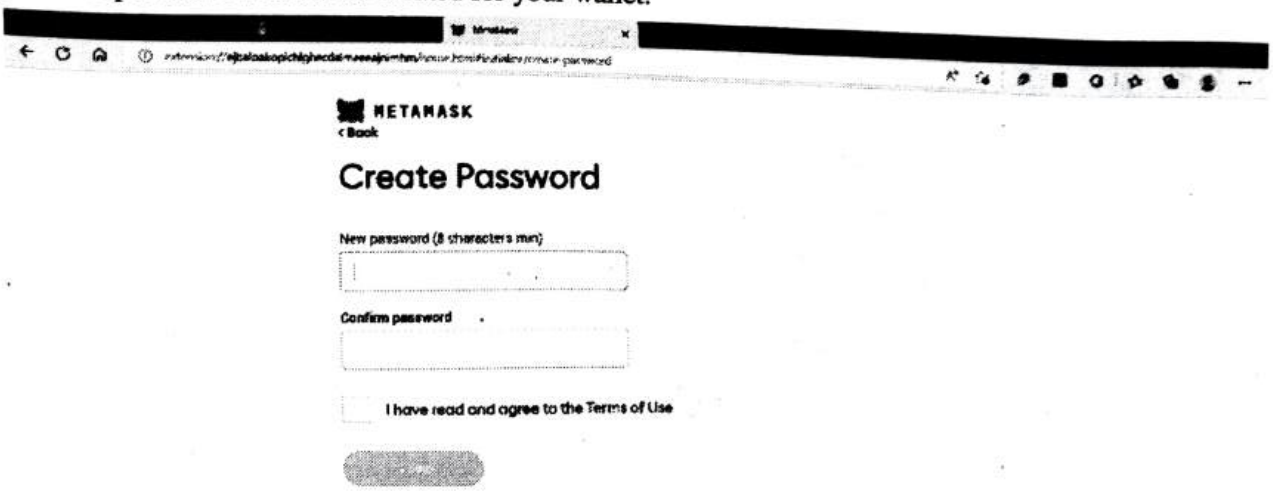
- **Step 6 :** If you do not have a wallet, then you need to create one. If you have a wallet, then it can be imported via a *Secret Recovery Phase*.



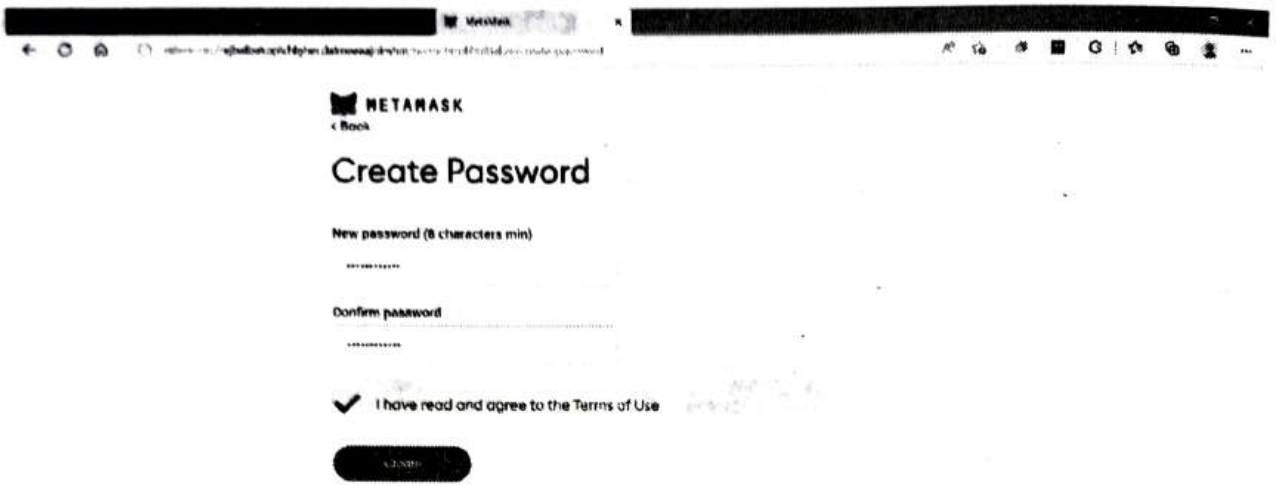
- **Step 7 :** Once the wallet is created, then you need to agree to the T&C of MetaMask.



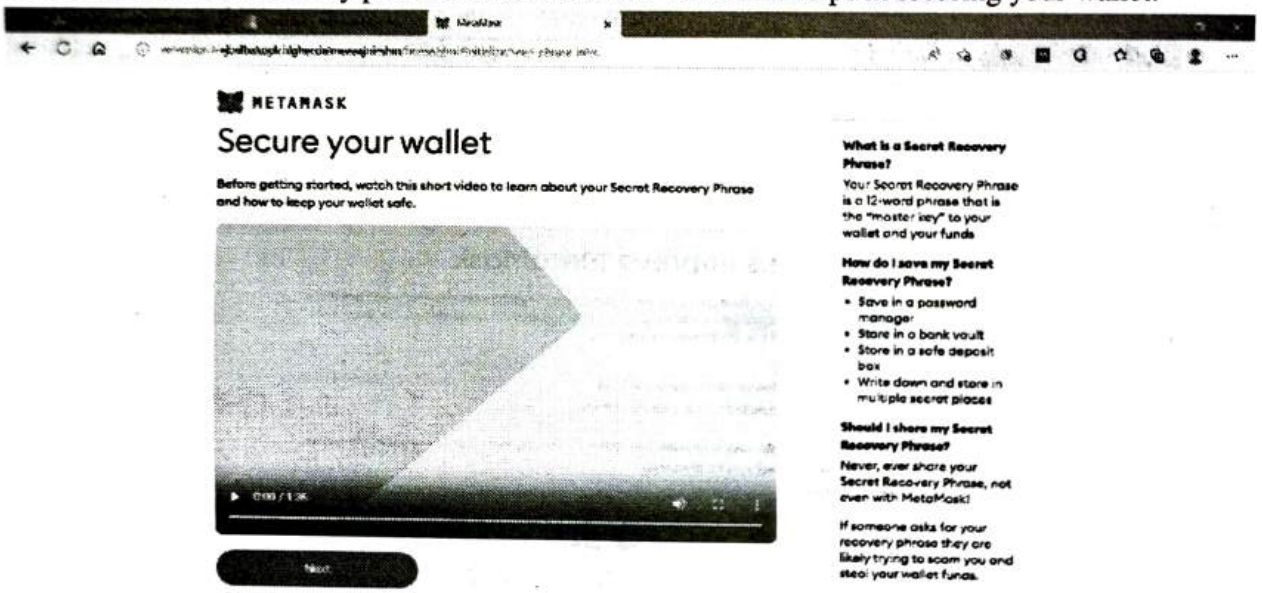
- **Step 8 :** A password needs to be created for your wallet.



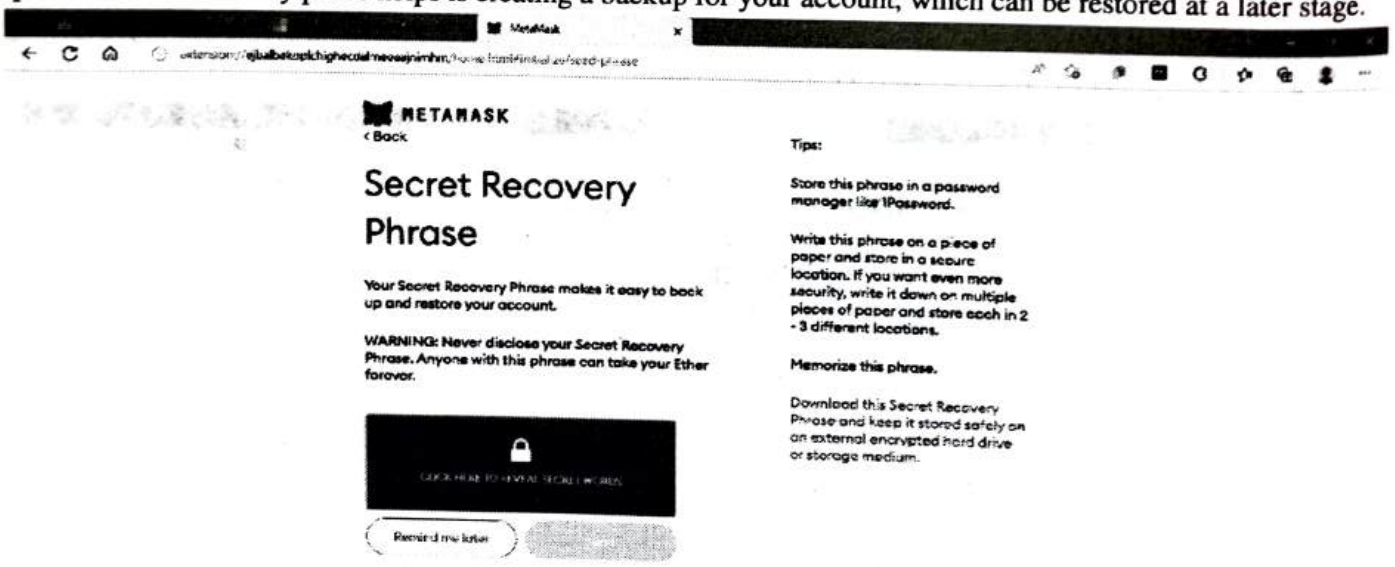
- **Step 9 :** Conformation of the password and agreement to the 'Terms of Use'.



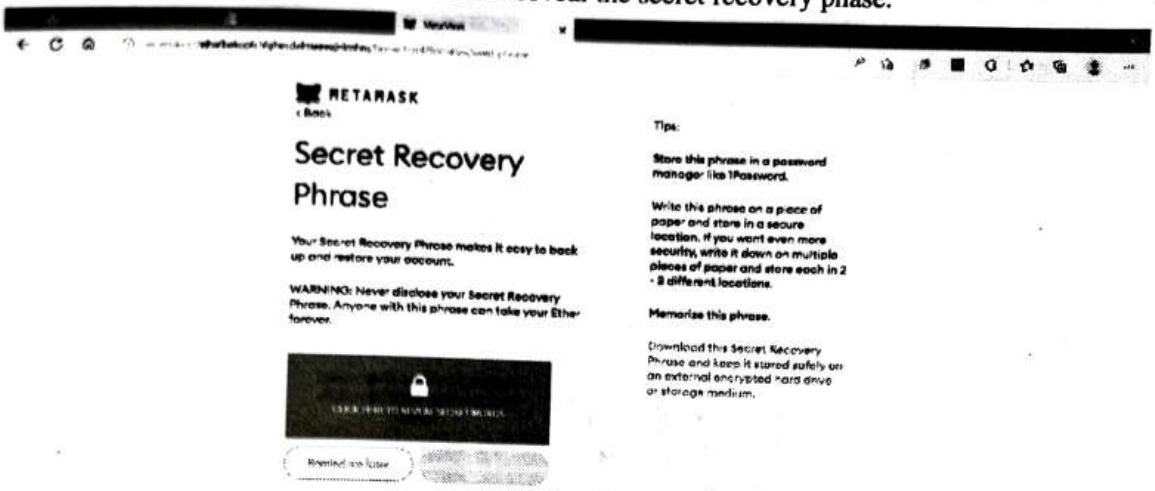
- **Step 10 :** Details about secret recovery phase can be seen in the video that helps in securing your wallet.



- **Step 11 :** Secret recovery phase helps is creating a backup for your account, which can be restored at a later stage.

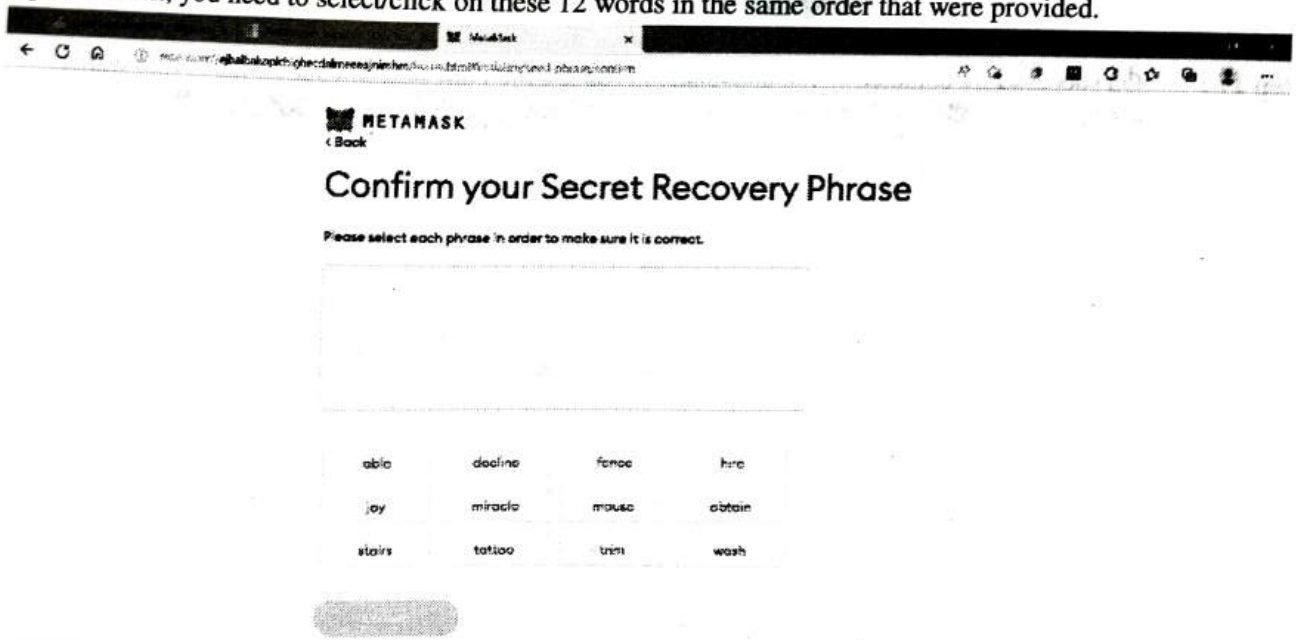


Step 12 : It is generally recommended not to share/reveal the secret recovery phase.



Step 13 : You will see 12 words, which you need to remember in the same order as provided.

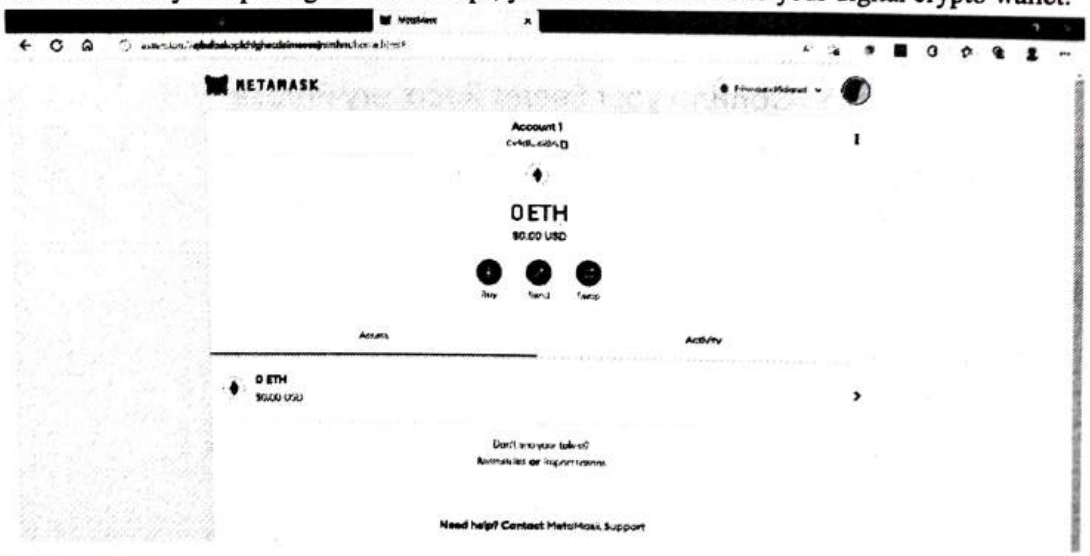
Step 14 : Next, you need to select/click on these 12 words in the same order that were provided.



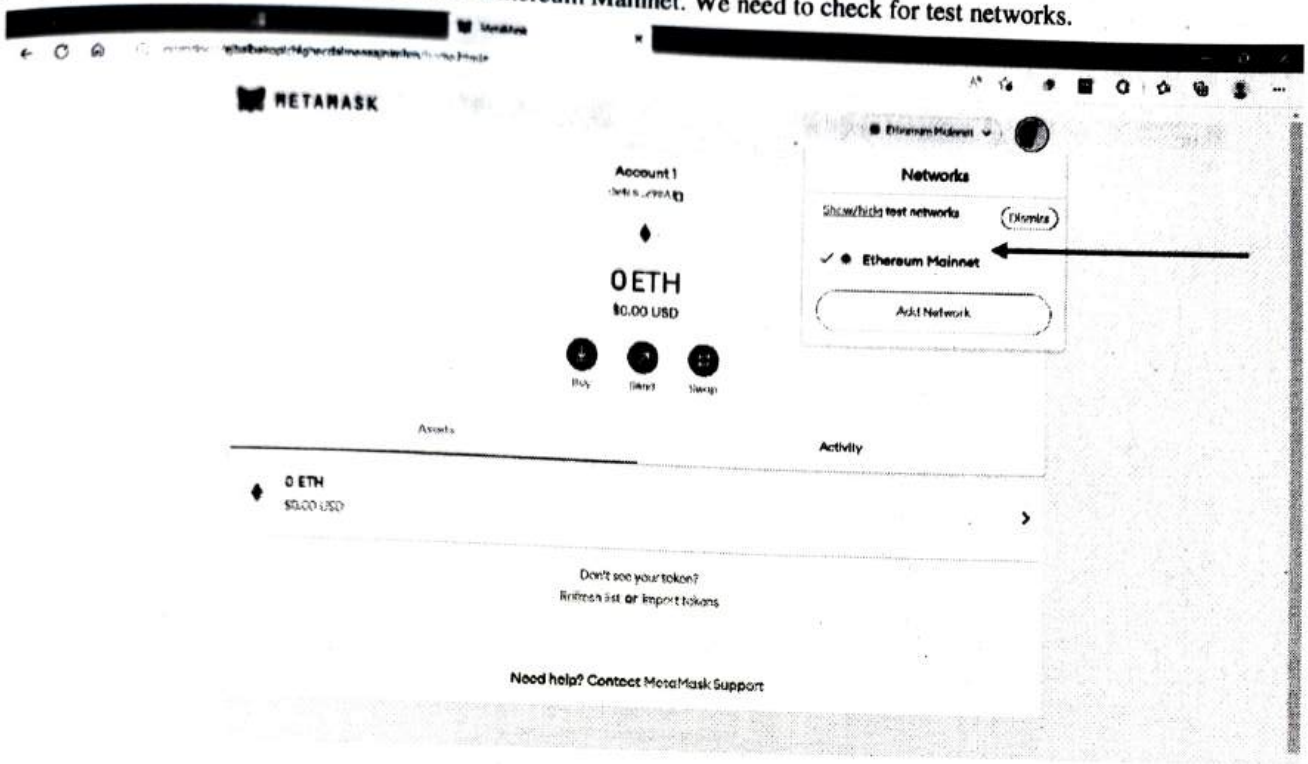
- **Step 15 :** Once you have selected/clicked on the same order, then you need to click on the conform button.

- **Step 16 :** If found correct, you can proceed further.

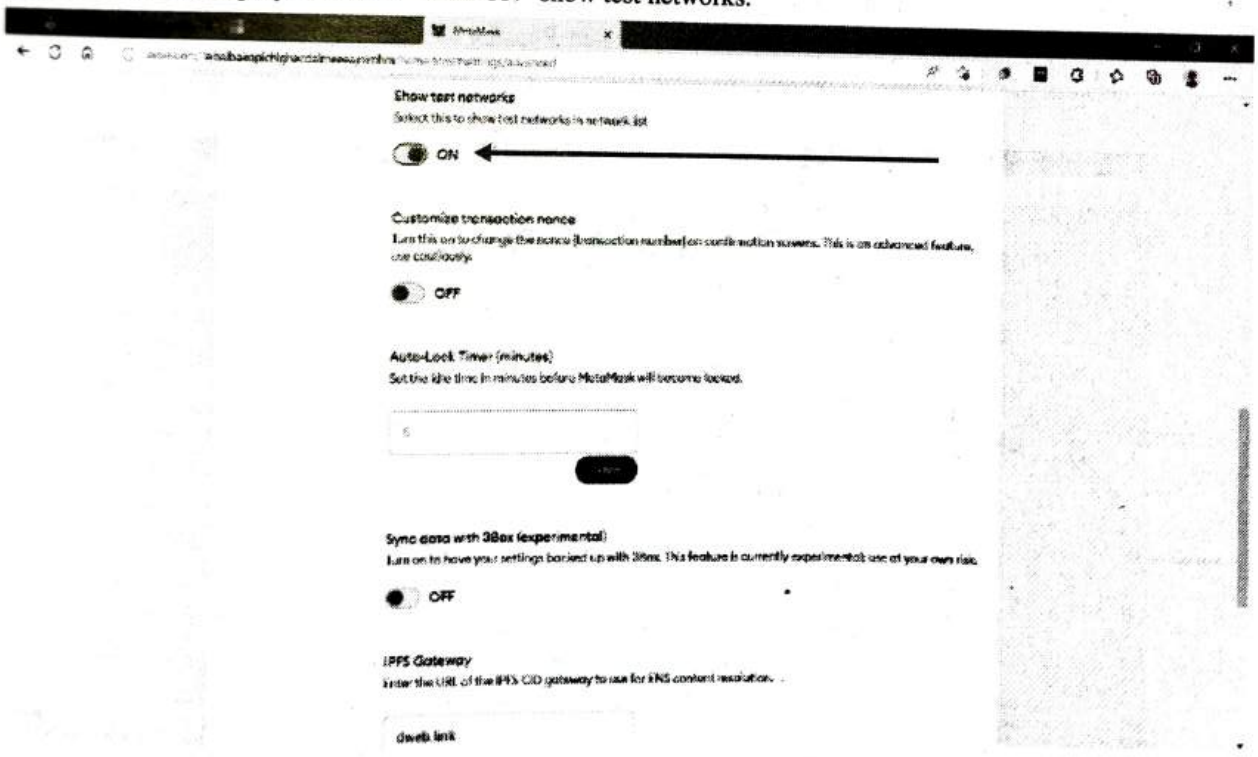
- **Step 17 :** On successfully completing the above steps, you are then directed to your digital crypto wallet.



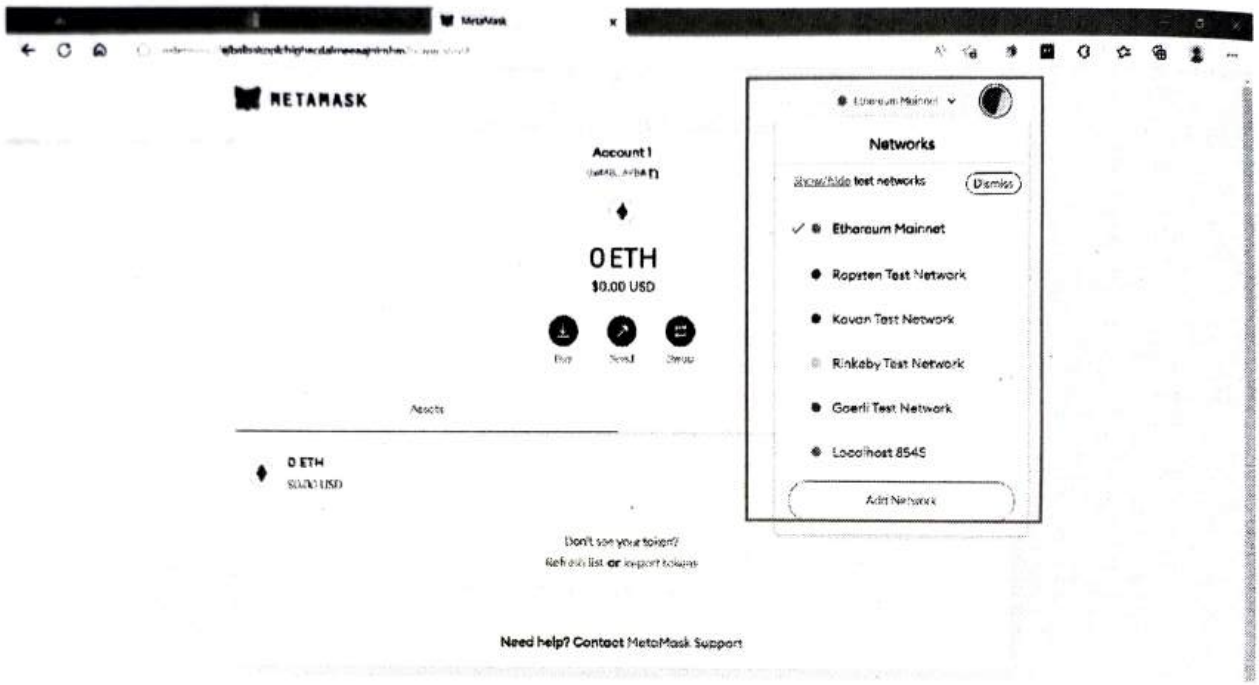
Step 18 : As you can see, we are on the Ethereum Mainnet. We need to check for test networks.



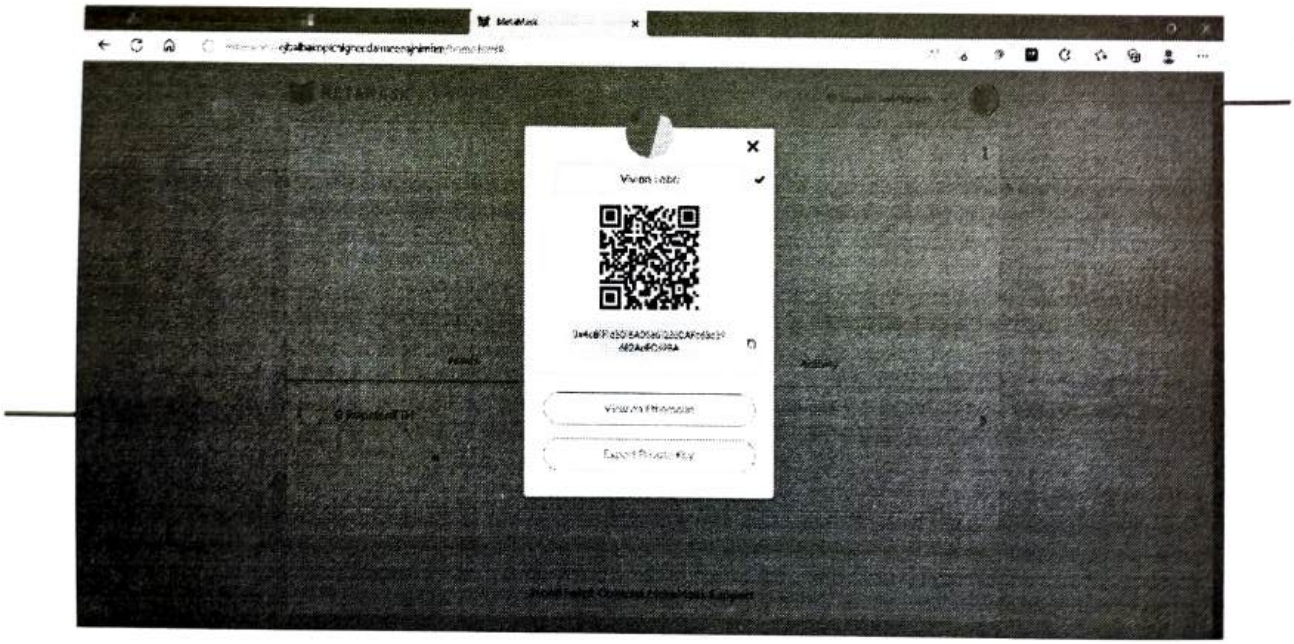
Step 19 : Under settings, you need to "Turn ON" show test networks.



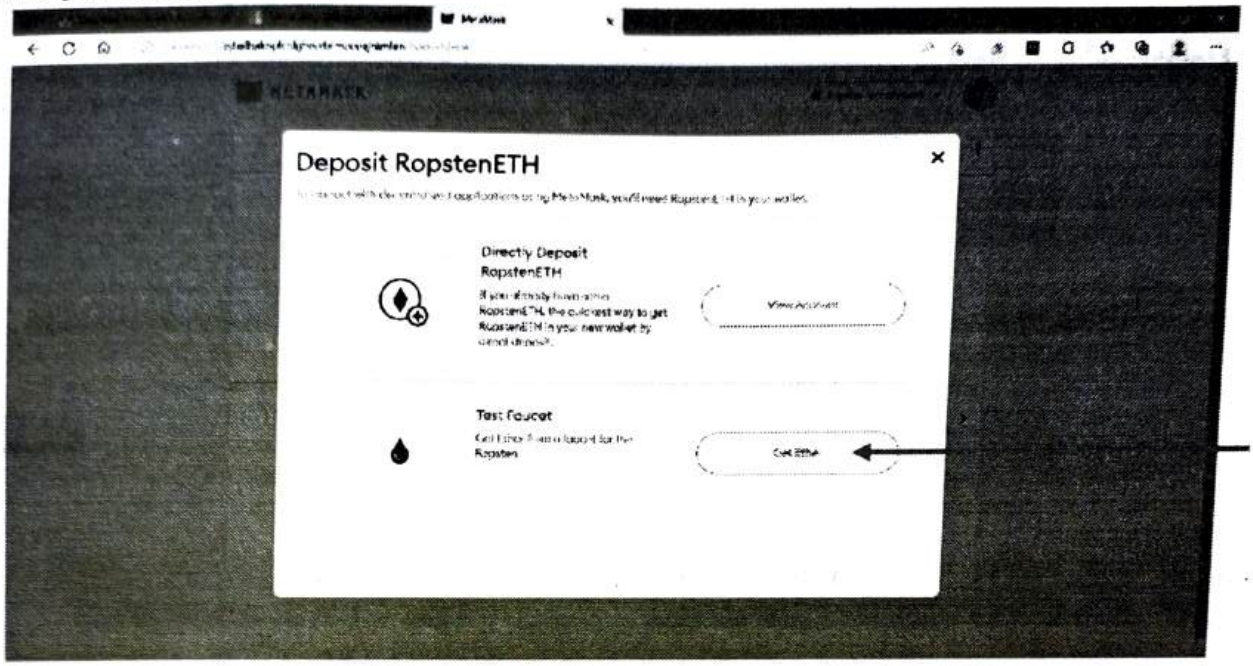
- **Step 20 :** Once the test networks have been 'Turned ON', then you will be able to see all test networks such as Ropsten, Kovan, Rinkeby, and Goerli.



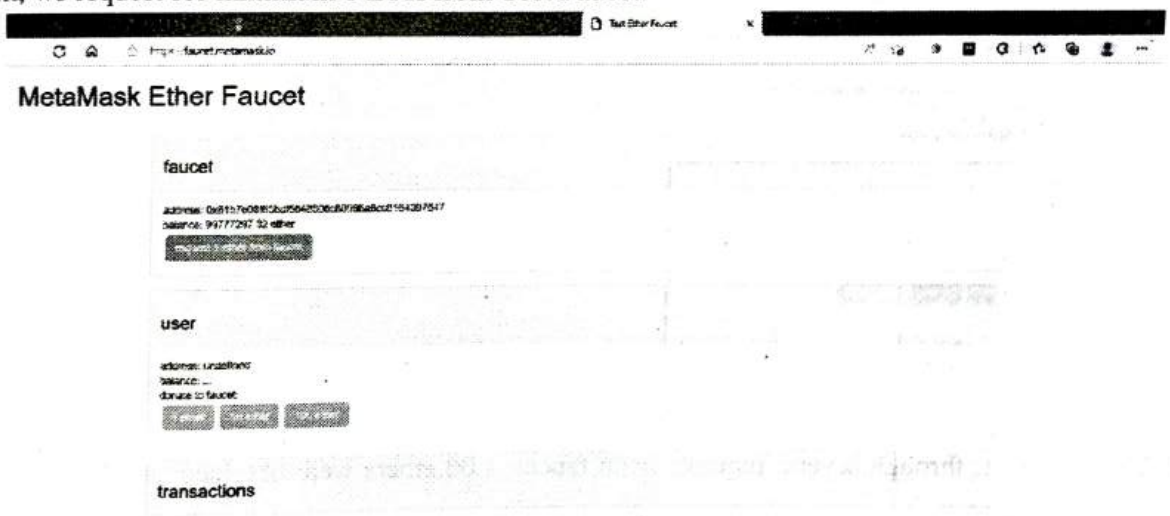
- **Step 21 :** On clicking on the icon, you can edit on the account name as well as you can see the account number (i.e., the public address). Note that I have selected Ropsten as the test network. You can select any test network of your choice. At present, there are 0 RopstenETH. We need to hunt for some ethers.



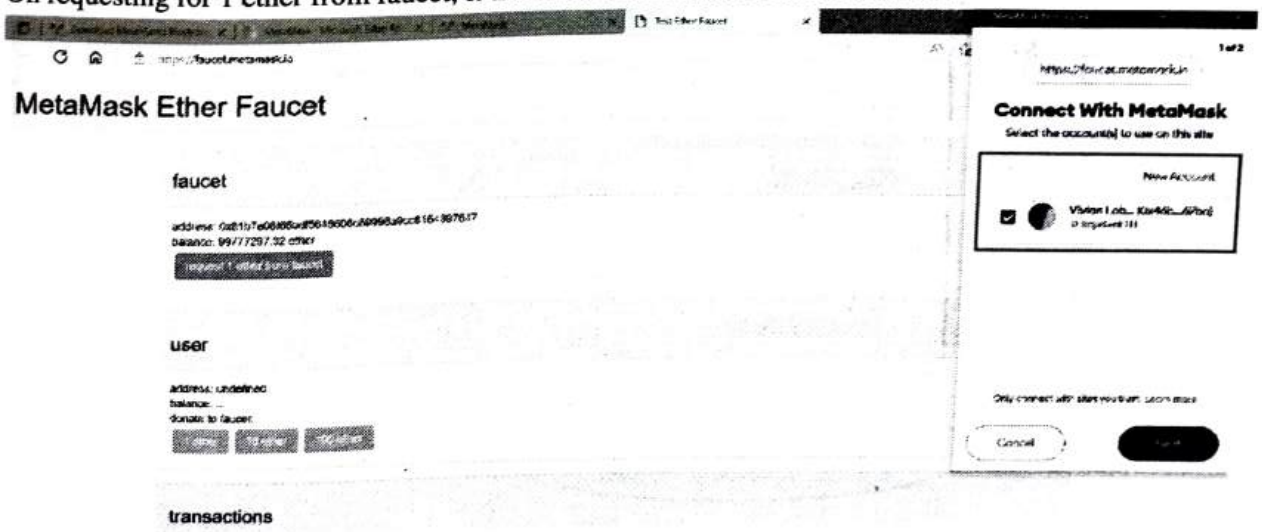
Step 22 : On clicking on 0 RopstenETH, a window opens, that offers **Deposit RopstenETH** or **Test Faucet**(i.e., get Ether [test ethers]).



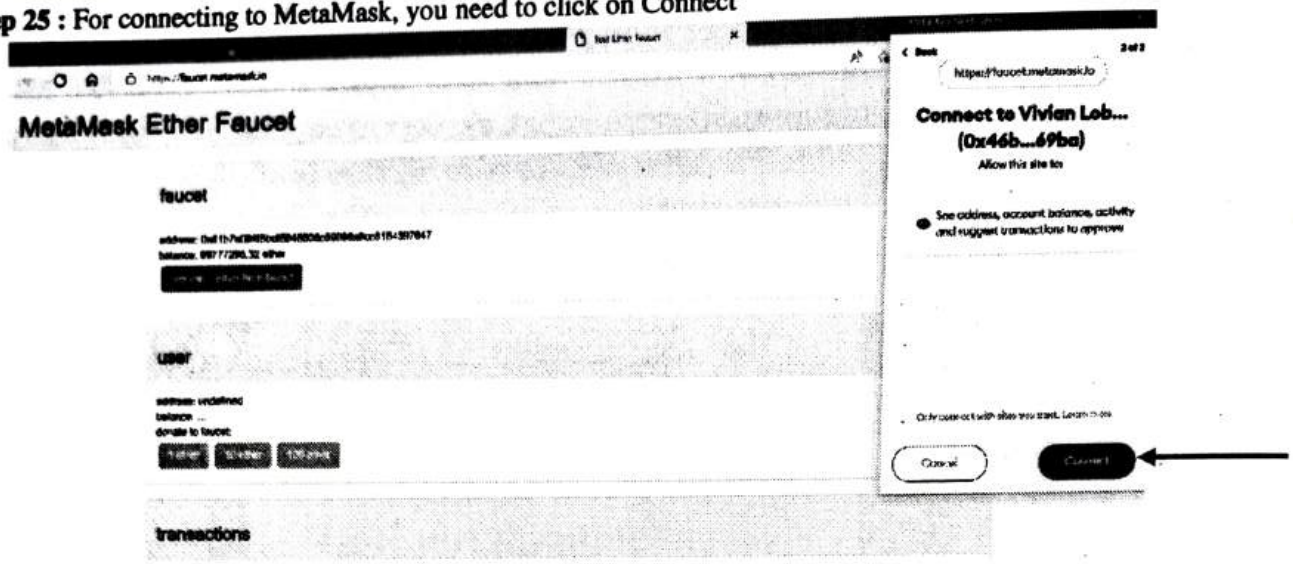
Step 23 : Next, we request for minimum 1 ETH from Test Faucet.



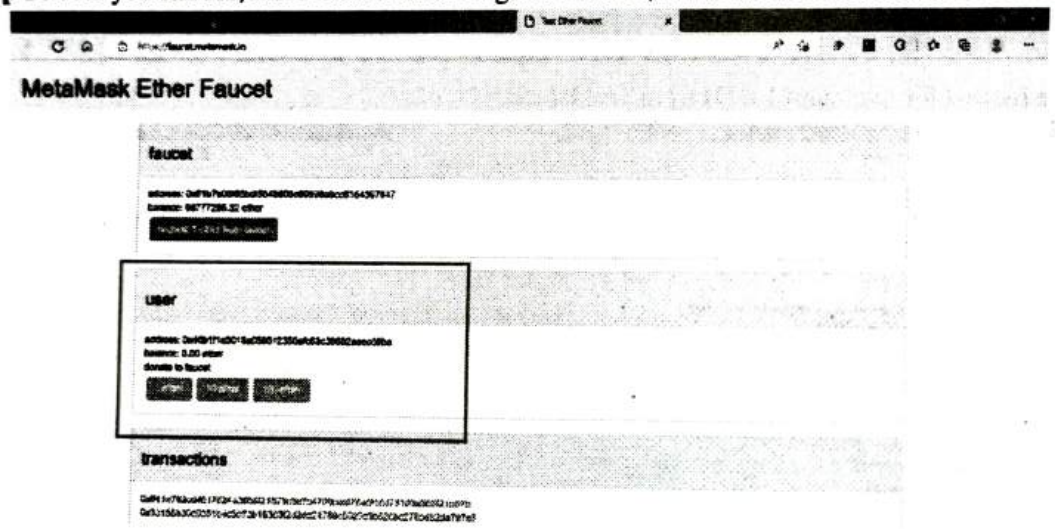
Step 24 : On requesting for 1 ether from faucet, it tries to connect with MetaMask wallet.



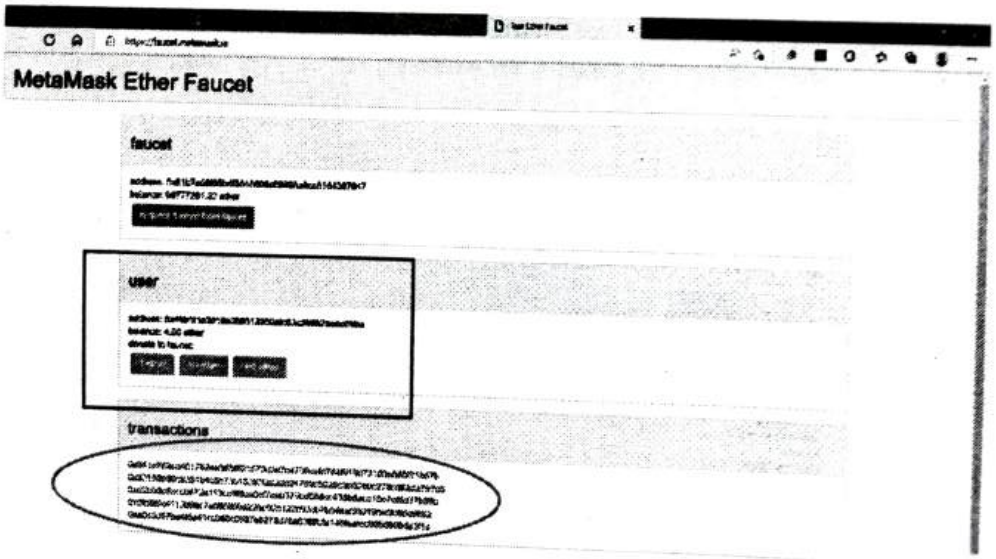
- **Step 25 :** For connecting to MetaMask, you need to click on Connect



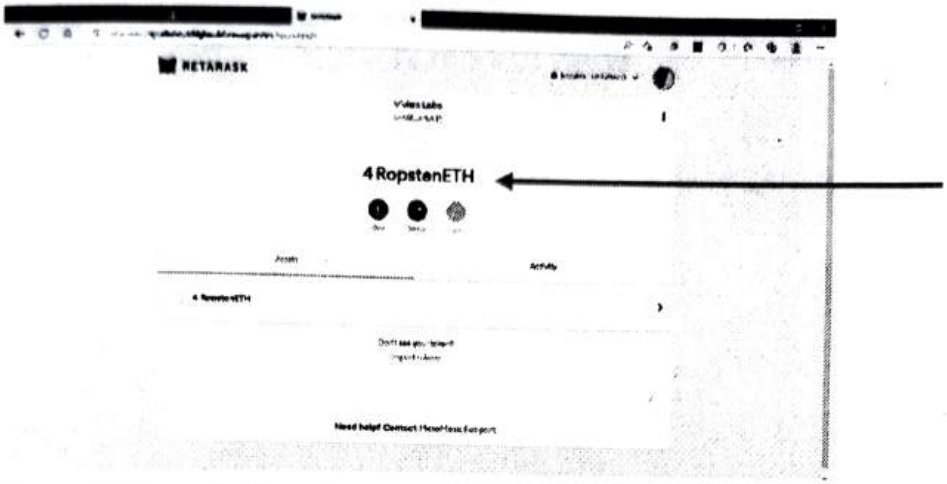
- **Step 26 :** As you can see, there is 0.00 ether for given address, which is the account that is created.



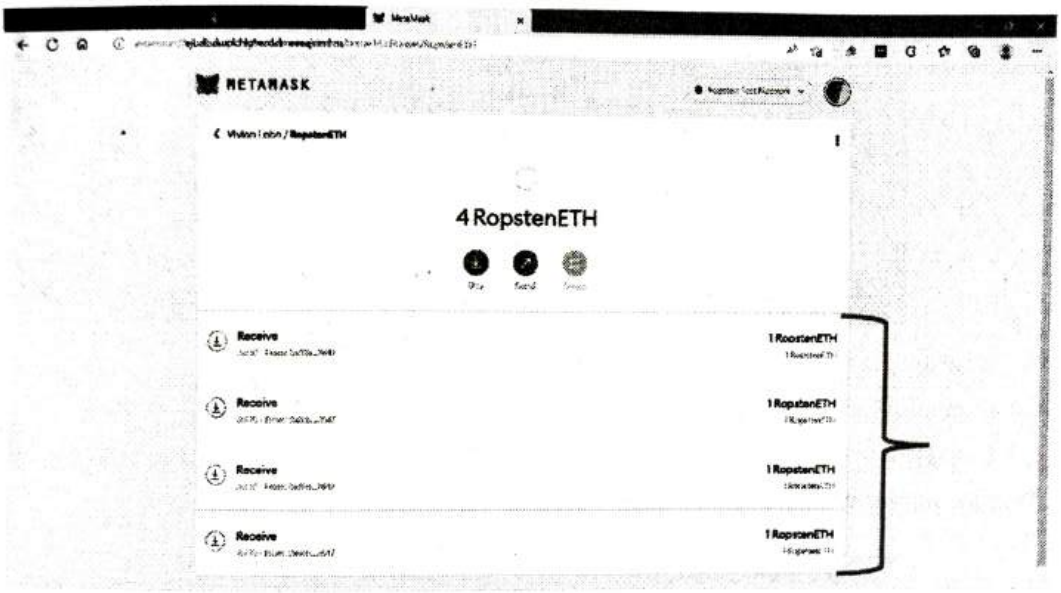
- **Step 27 :** However, through several requests from faucet, 4.00 ethers were obtained and the transaction address can also be seen.



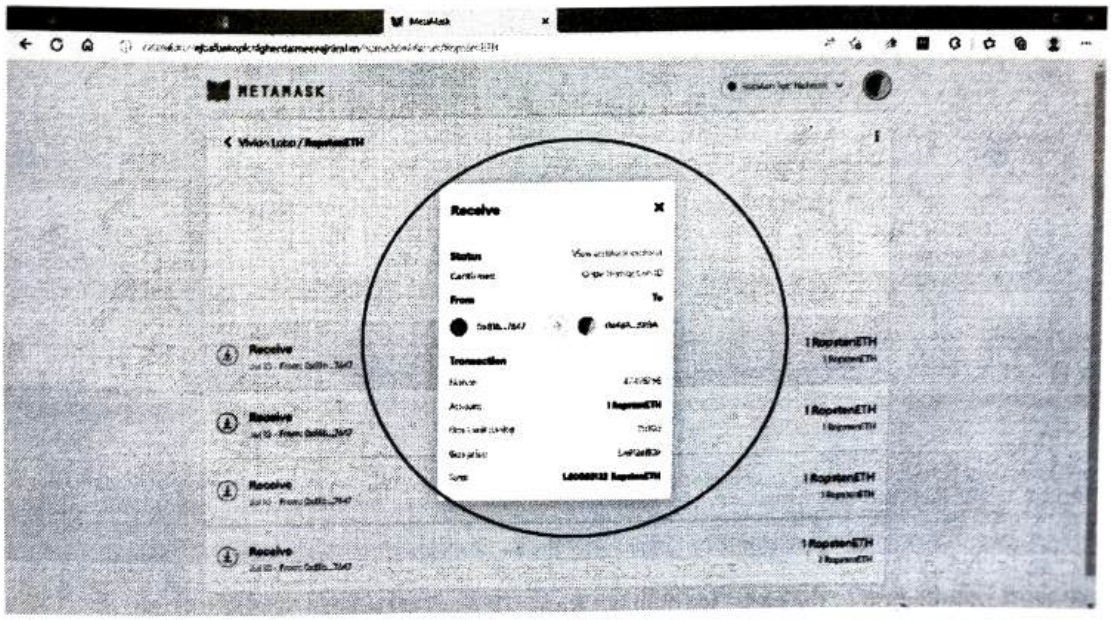
Step 28 : As you can see, 4 RopstenETH were received.



Step 29 : Details of each RopstenETH received can be seen below.



Step 30 : On clicking on each RopstenETH, the following details can be found.

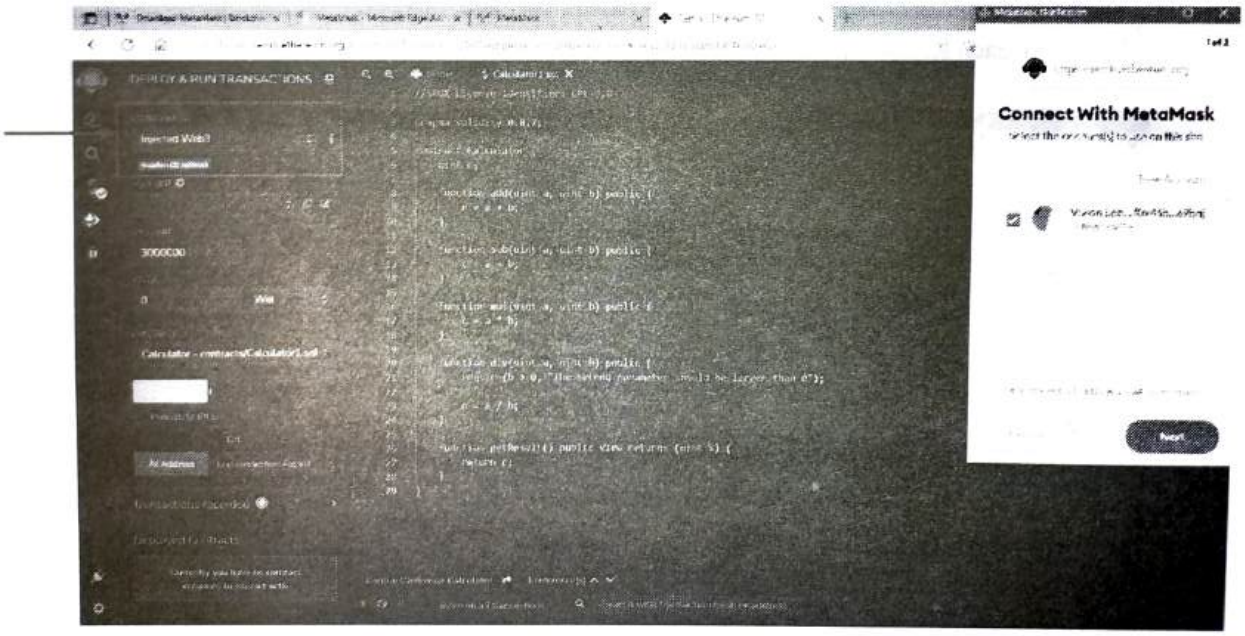


Step 31 : Writing and compiling the code in Remix IDE

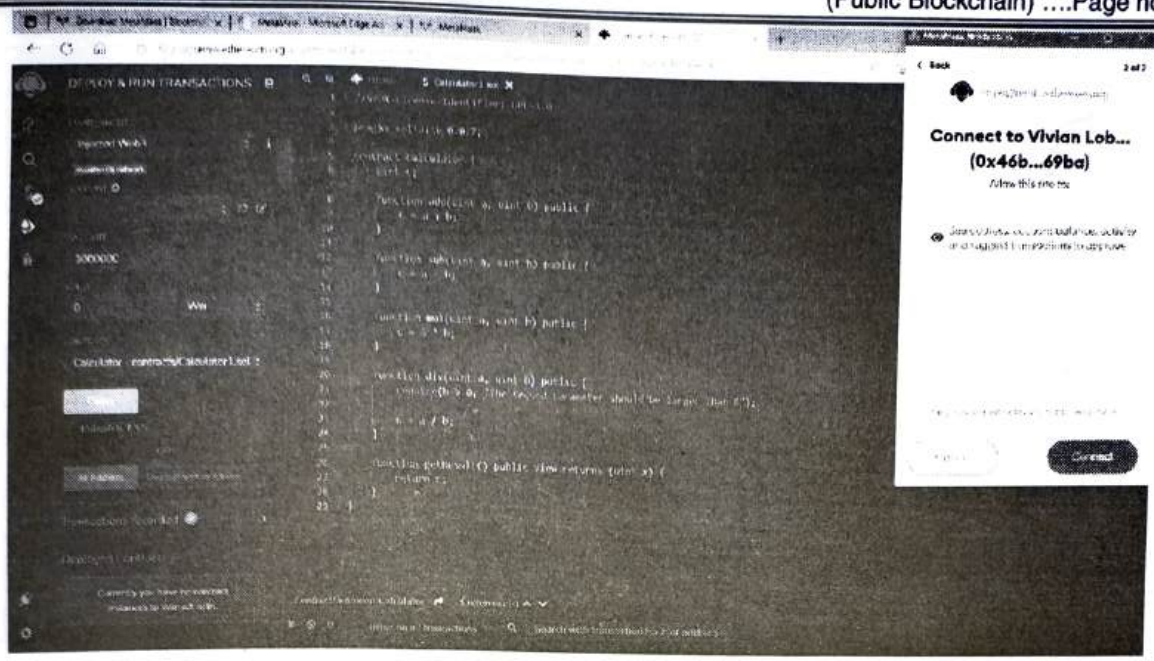
- Calculator Program

```
//SPDX-License-Identifier: GPL-3.0

pragma solidity 0.8.7;

contract calculator {
    uint c;

    function add(uint a, uint b) public {
        c = a + b;
    }

    function sub(uint a, uint b) public {
        c = a - b;
    }

    function mul(uint a, uint b) public {
        c = a * b;
    }

    function div(uint a, uint b) public {
        require(b > 0, "The second parameter should be larger than 0");

        c = a / b;
    }

    function getResult() public view returns (uint x) {
        return c;
    }
}
```

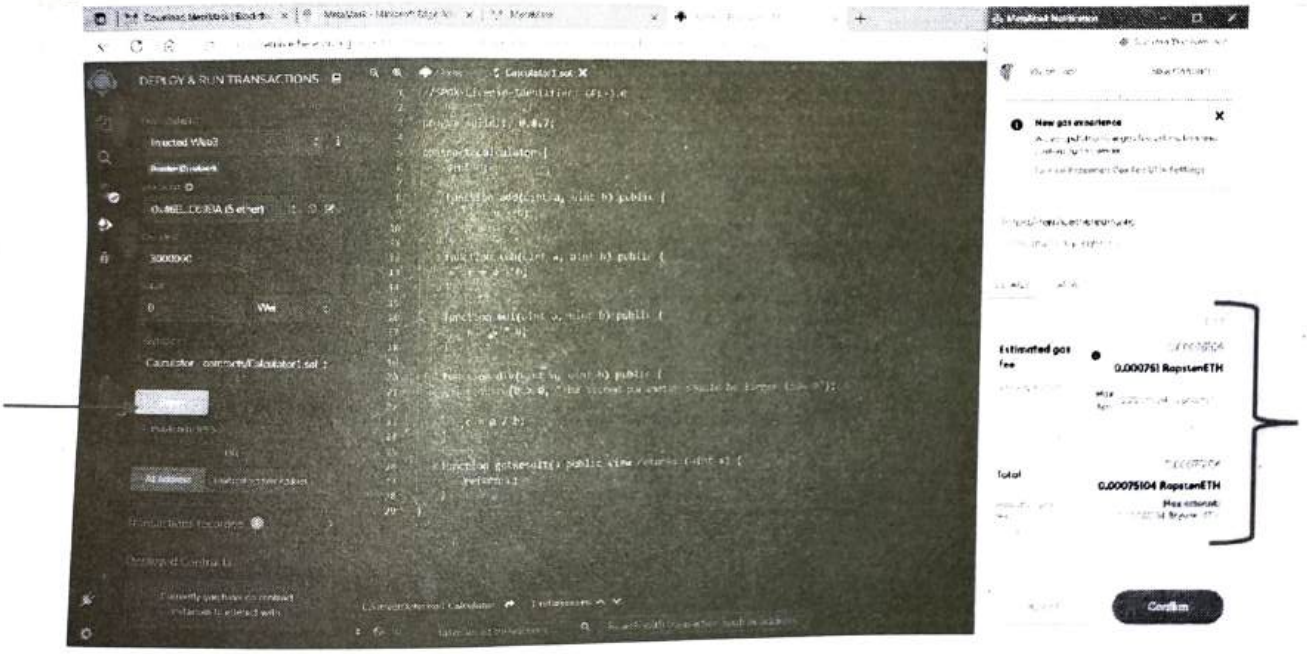
- The above code is for a simple calculator that performs basic arithmetic operations.
- The code is written in Remix IDE.
- Once the code is written, the environment needs to be changed from JavaScript VM (London)to Injected Web3. This in turn will lead to connect with MetaMask wallet.



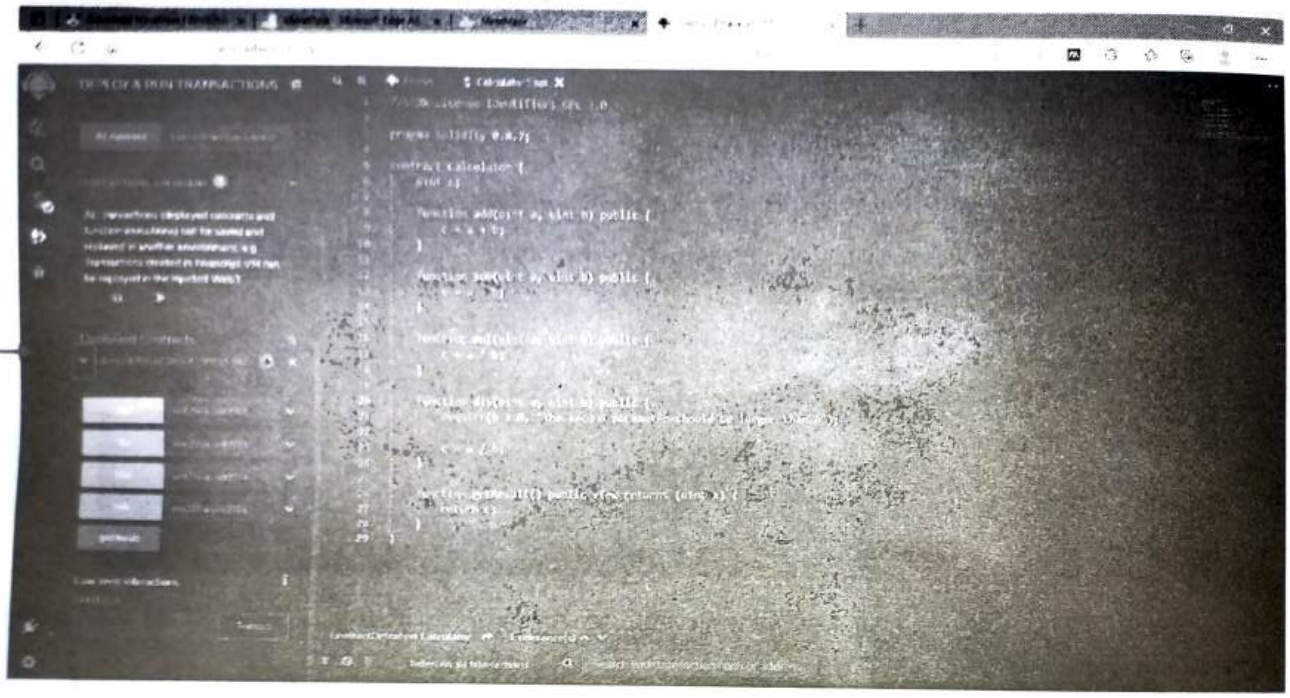
- On successful compilation of the code, we connect Remix IDE to MetaMask wallet.



Once connected, there exists an Estimated Gas Fee that shows how much RopstenETH would be used out of 4 RopstenETH. Click on confirm.



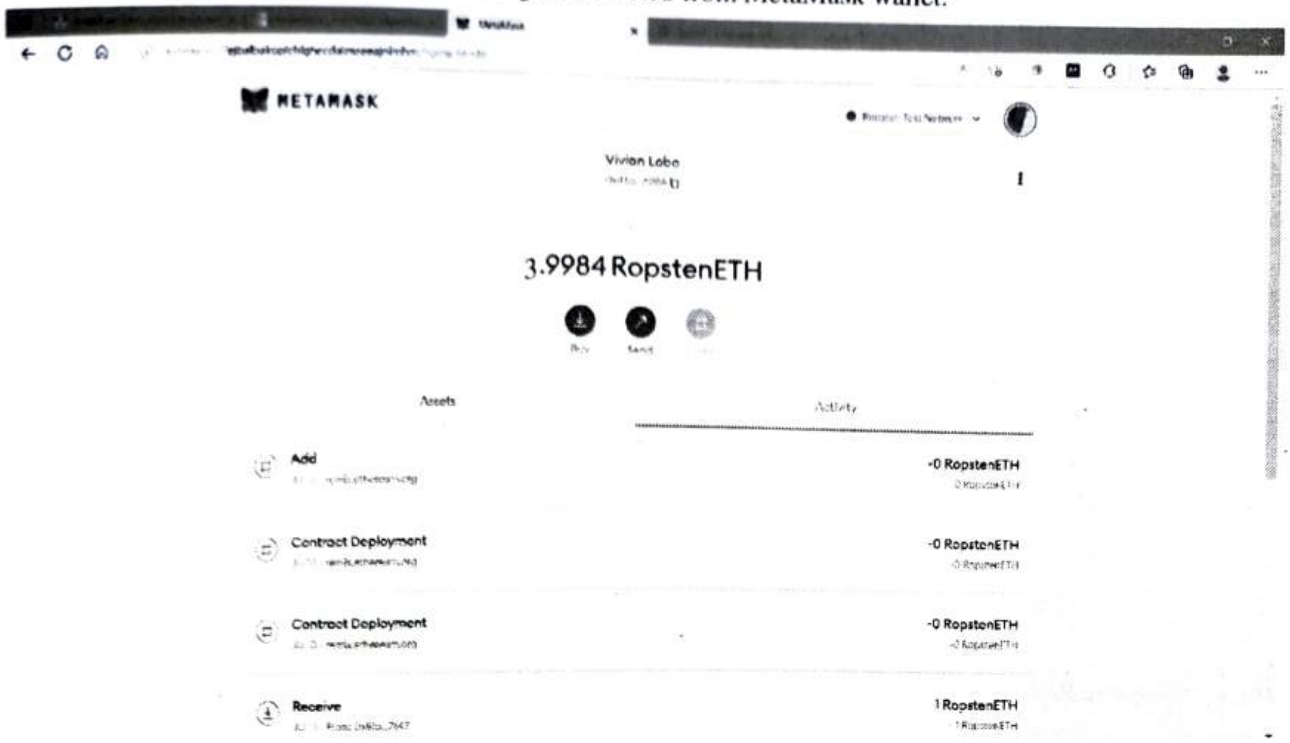
Next, click on "deploy" contact, which will result in the following output.



- When we consider two numbers (i.e., 12 and 10) and attempt to perform addition operation, we need to click on transact. On clicking on transact, the MetaMask wallet gets connected.

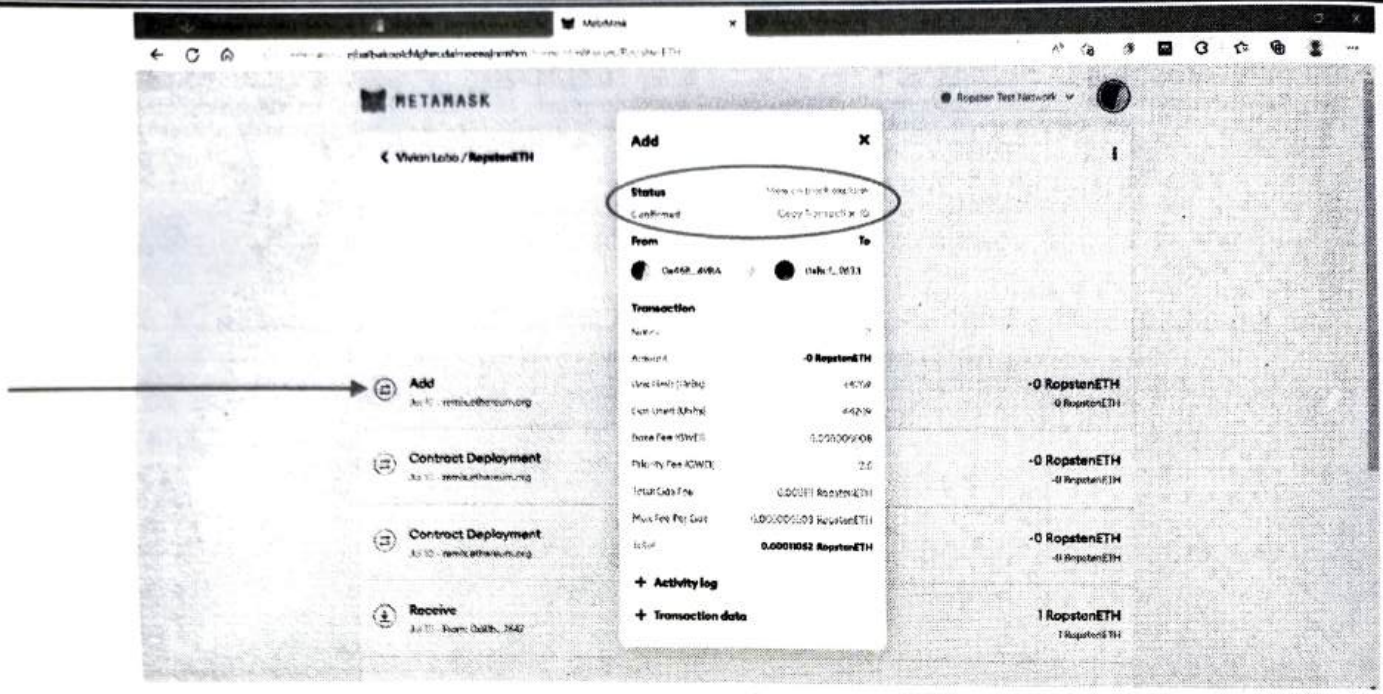
NOTES

For every transaction, a certain amount of fee gets deducted from MetaMask wallet.

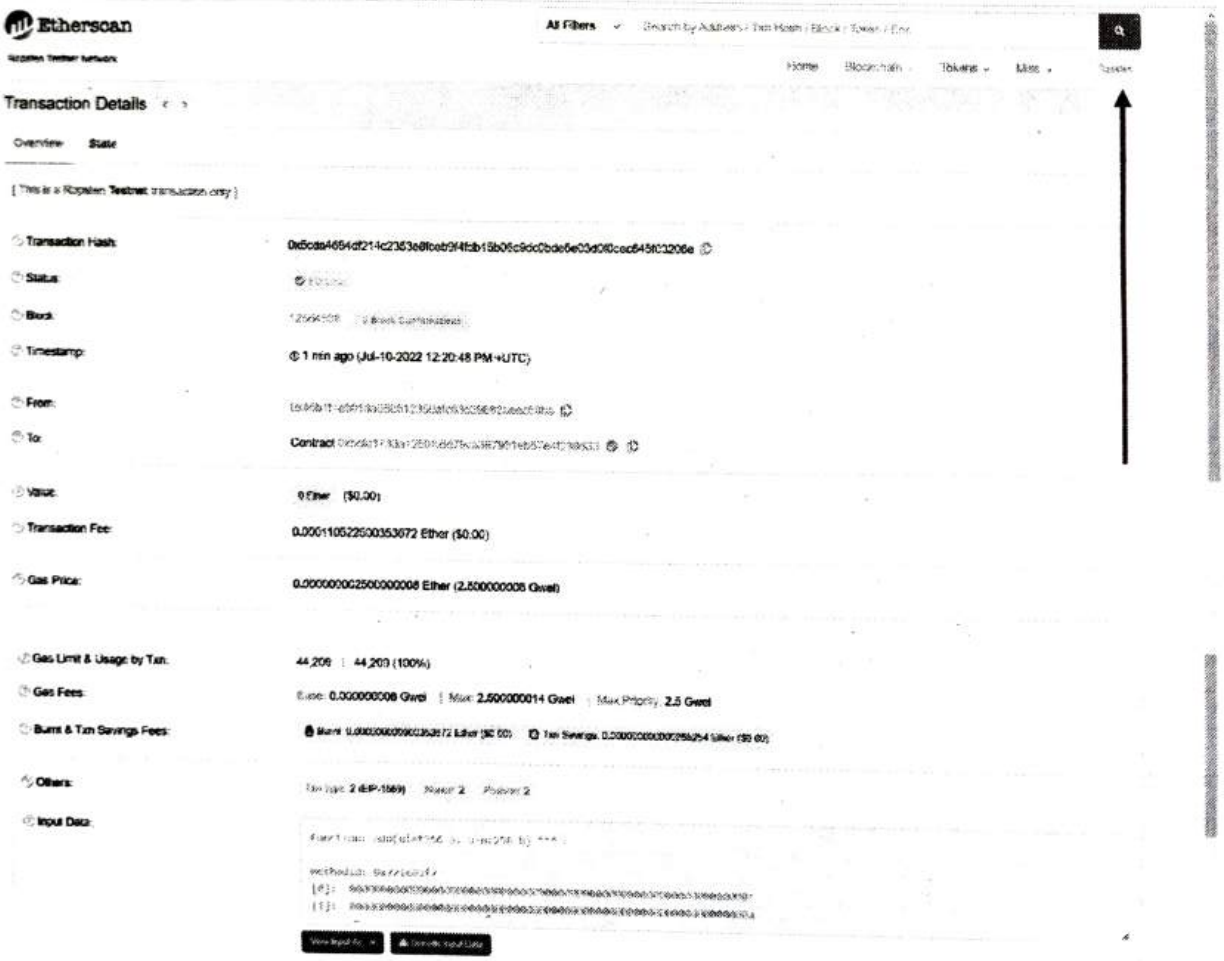


Next, the result of addition of two numbers can be seen in the *getResult()*.

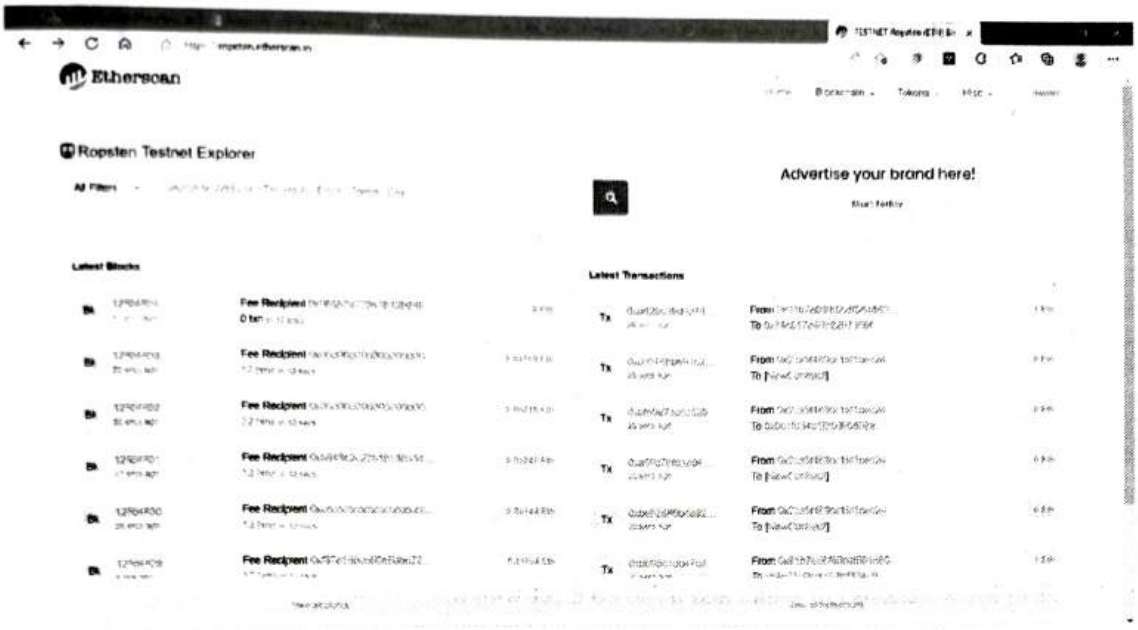
Whether the transaction is successful or not can be seen by clicking on Add, as shown below. Also, the status of the transaction can be seen on Block Explorer as well as Transaction ID can be seen.



- Also, on Etherscan for Ropsten Test Network, the transaction details can be seen.



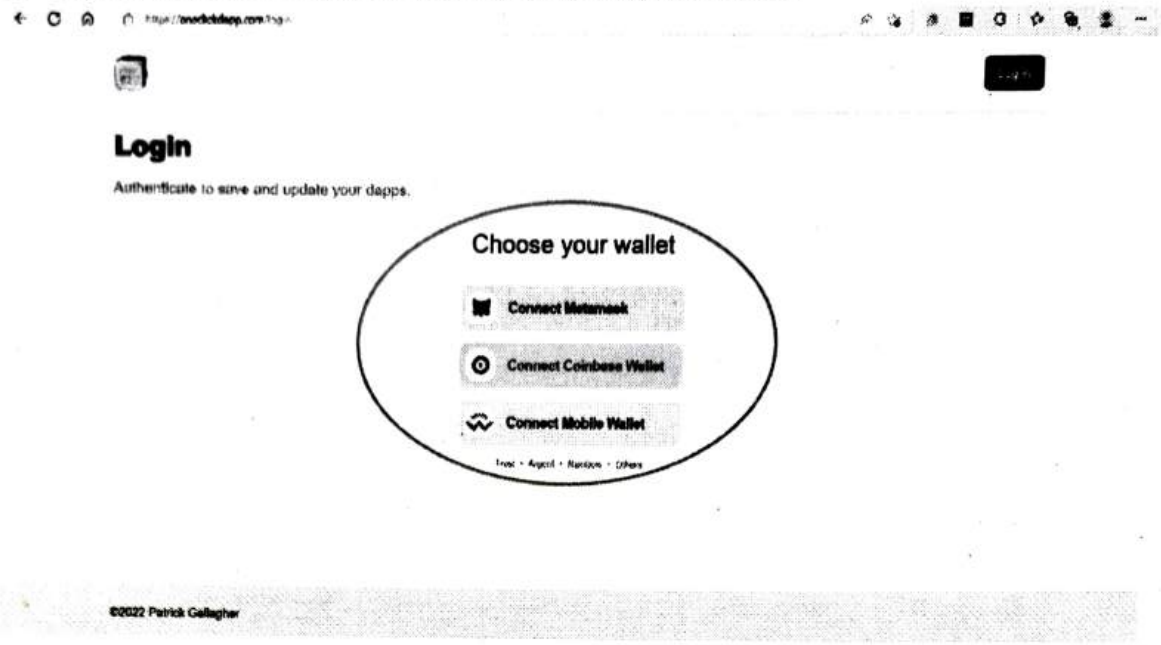
On clicking on Ropsten, we can see entire Ropsten Testnet Explorer, which includes latest blocks and latest transactions.



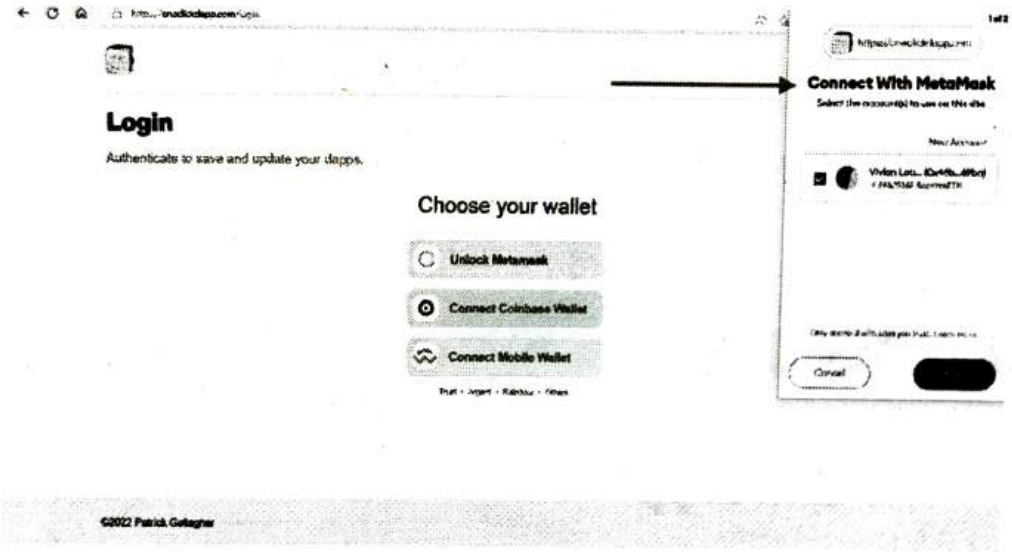
Creating a simple application using ***One Click Dapp*** (https://oneclickdapp.com/). One Click Dapp turns any smart contract into a decentralized application (dapp).

Step 1 : With the help of One Click Dapp, a decentralized application can be developed. First, we need to "Log in" to Once Click Dapp.

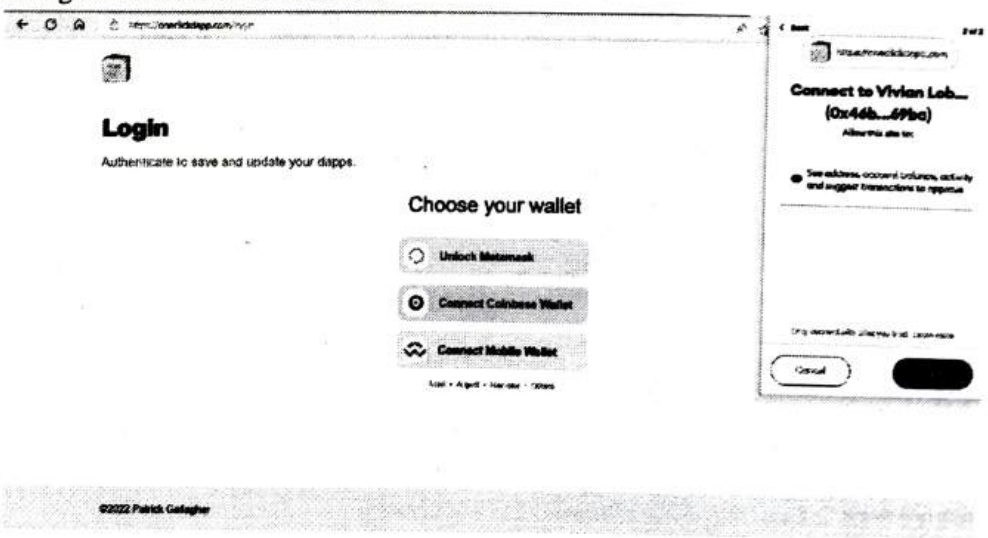
- **Step 2 :** After "Log in", we need to choose a wallet from the available wallets.



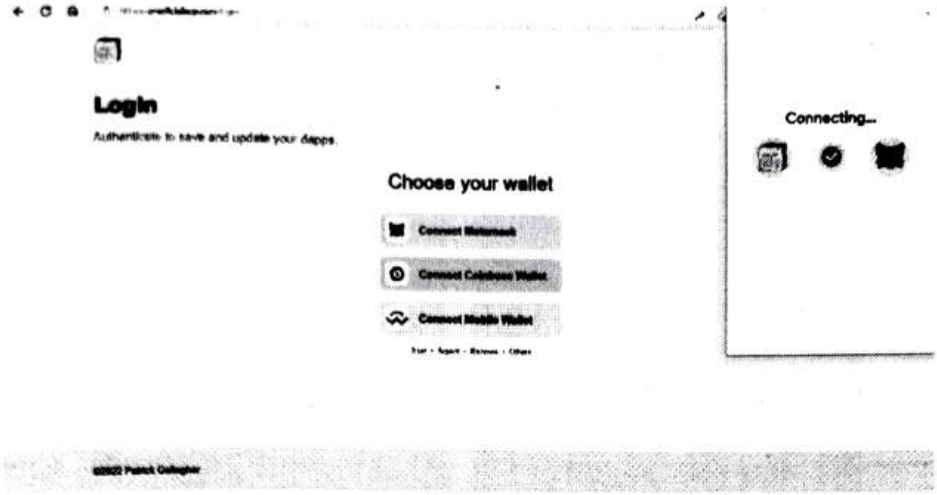
- **Step 3 :** We select MetaMask as the wallet and try to connect with it.



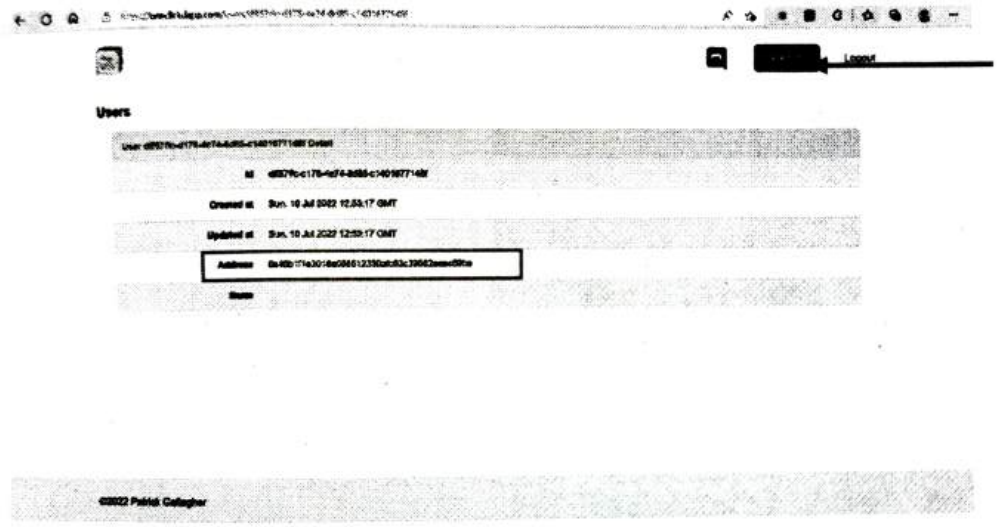
- **Step 4 :** Waiting for connection to establish.



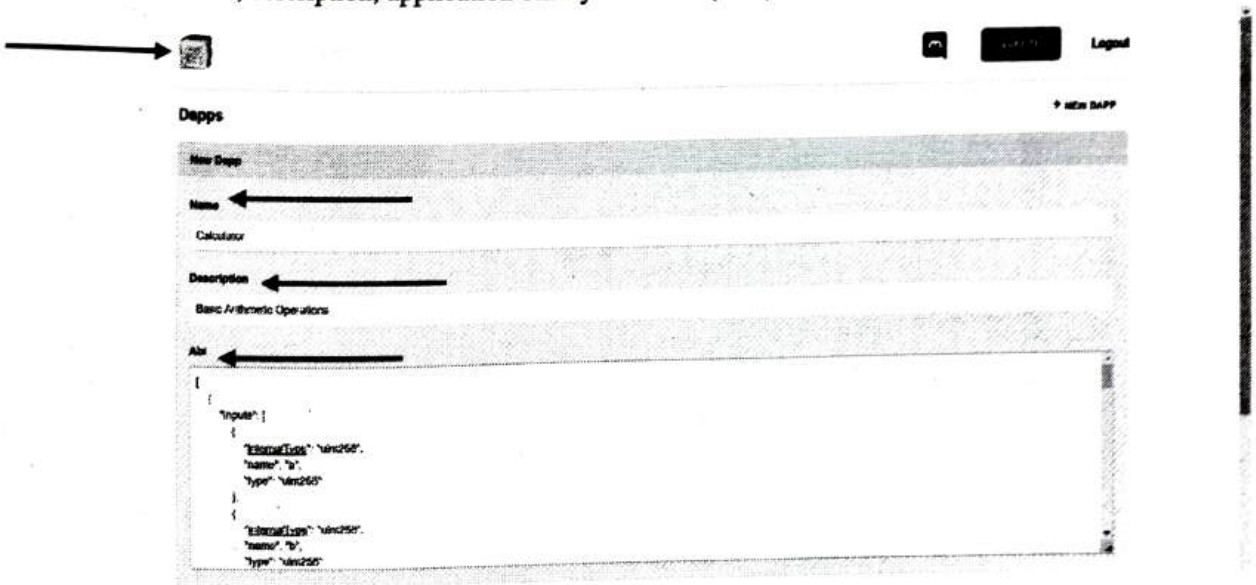
Step 5 : One Click Dapp and MetaMask getting connected.



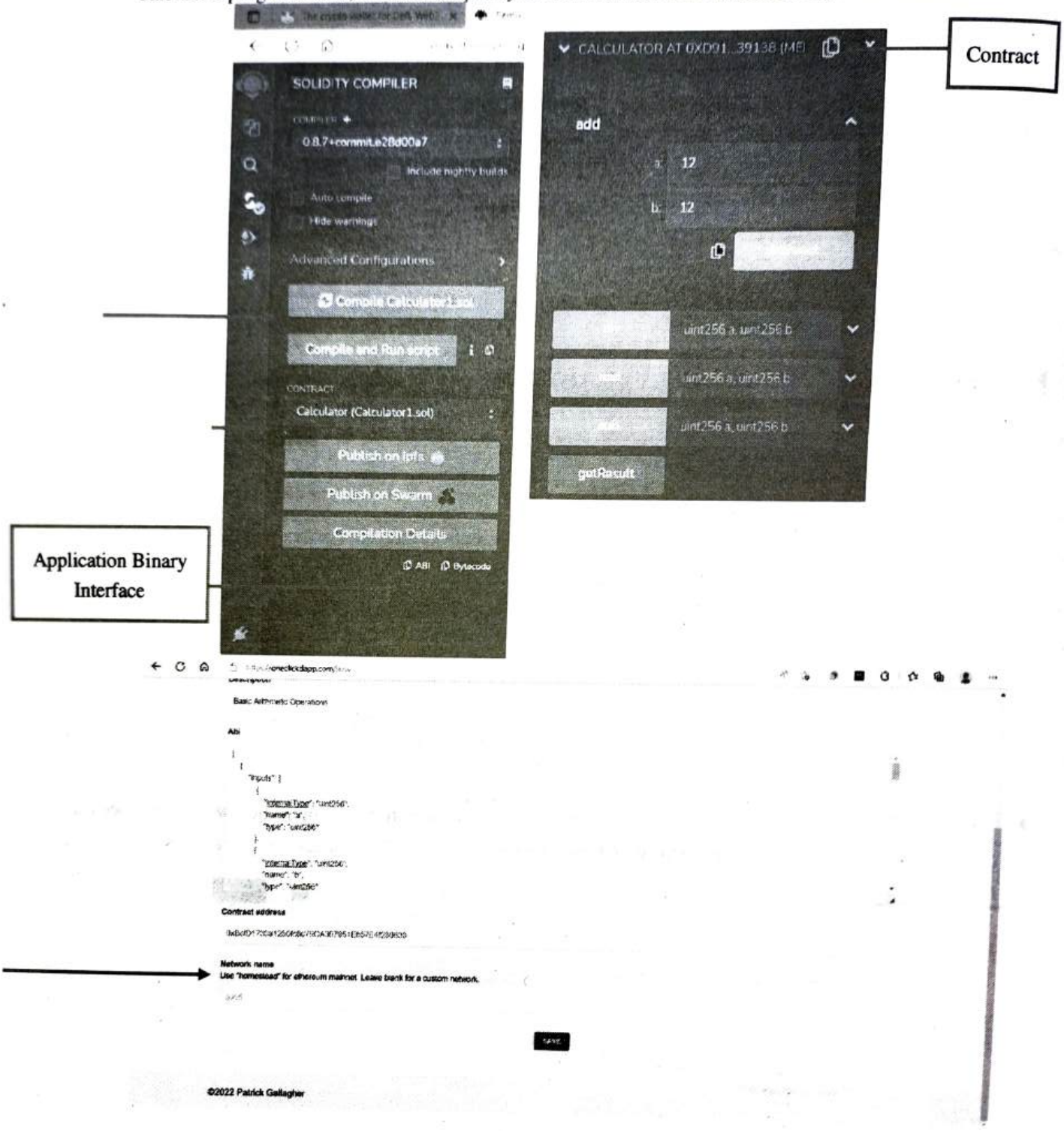
Step 6 : MetaMask wallet address (i.e., public key) can be seen after successful connection.

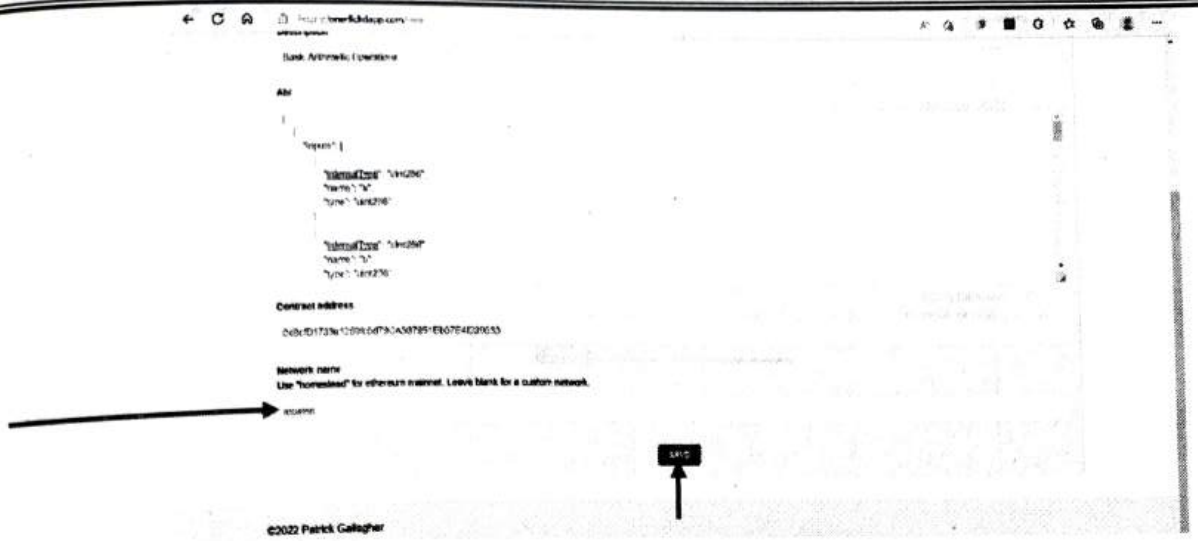


Step 7: Next, we click on the One Click Dapp and create a decentralized application wherein we need to specify the name, description, application binary interface (ABI) code, contract address, and test network name.



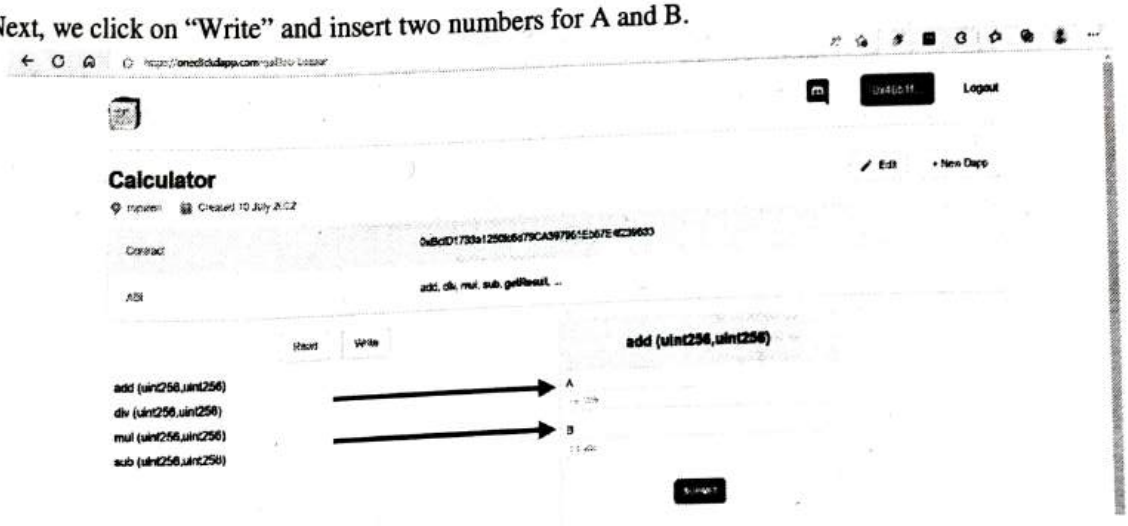
- **Step 8 :** ABI code and contact address are obtained from Remix IDE where the solidity code is written for Calculator program. Also, we need to specify the test network name, which is Ropsten in our case.



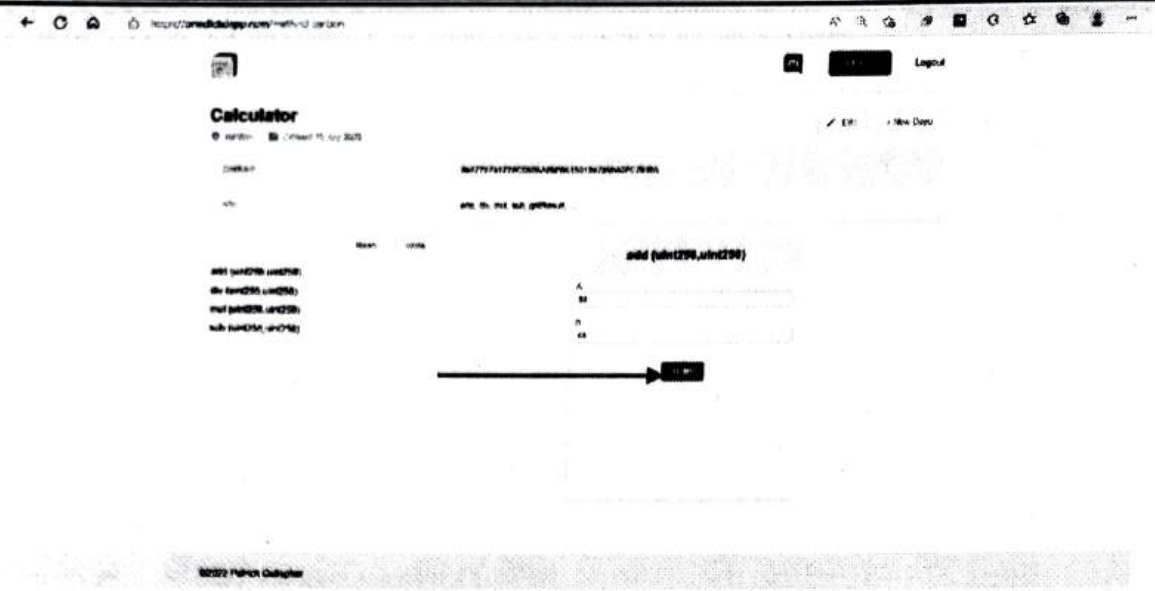


Step 9 : Once done, click on SAVE button. Successful execution indicates that a Dapp has been created, as shown below.

Step 10 : Next, we click on "Write" and insert two numbers for A and B.



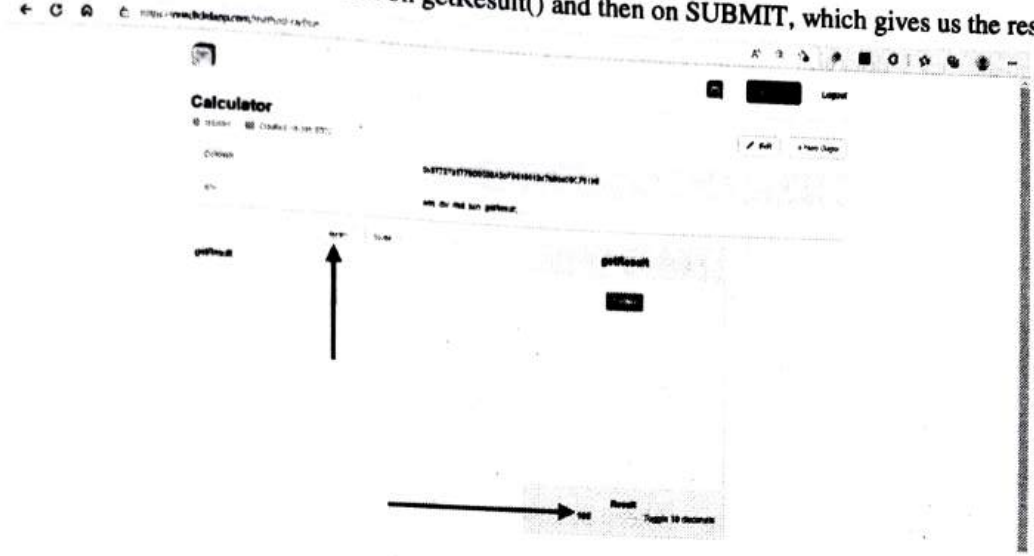
Step 11 : We assign A = 50 and B = 50 and then click on SUBMIT.



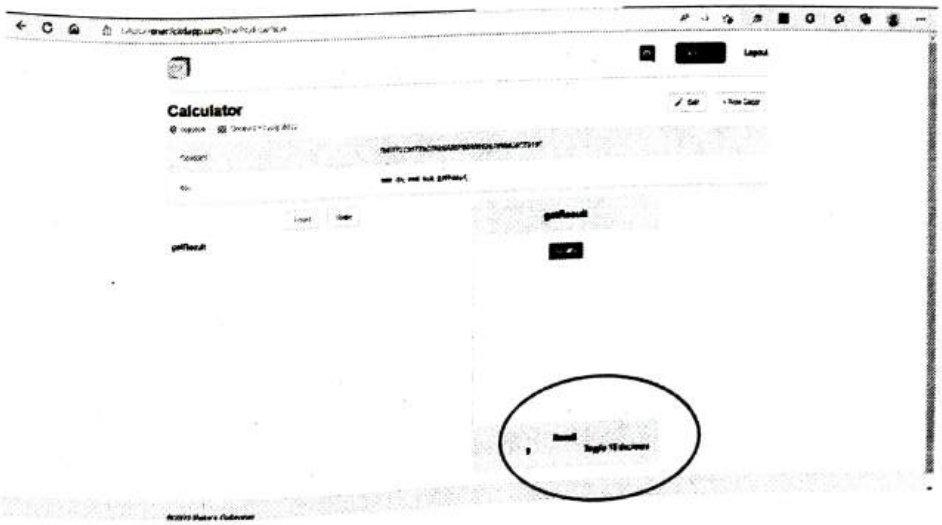
- **Step 12 :** By clicking on SUBMIT, we get connected to MetaMask wallet wherein certain gas fee is estimated.

- **Step 13 :** By clicking on Confirm, we obtain a Success! Message.

Step 14 : Next, we click on Read and click on getResult() and then on SUBMIT, which gives us the result (i.e., 100)



Step 15 : In a similar manner, we can do it for other arithmetic operators.

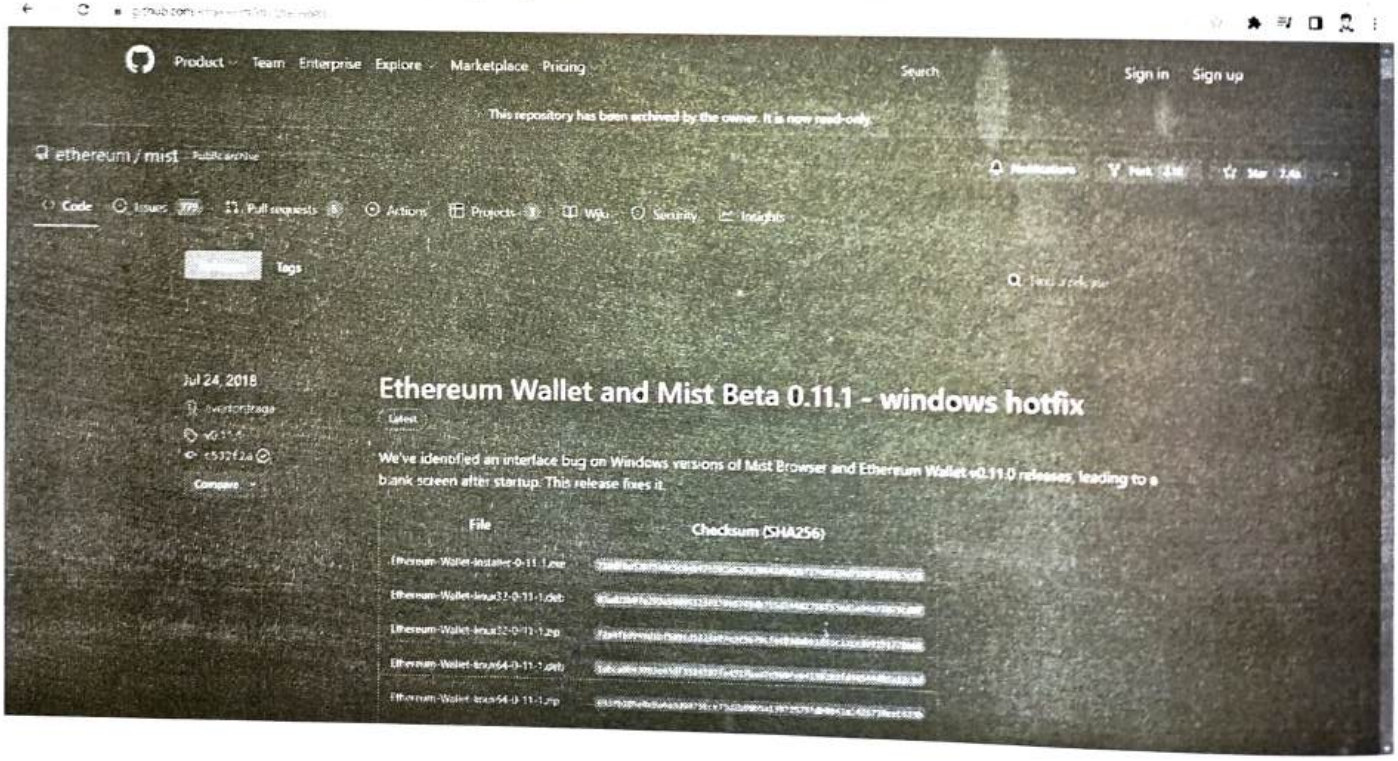


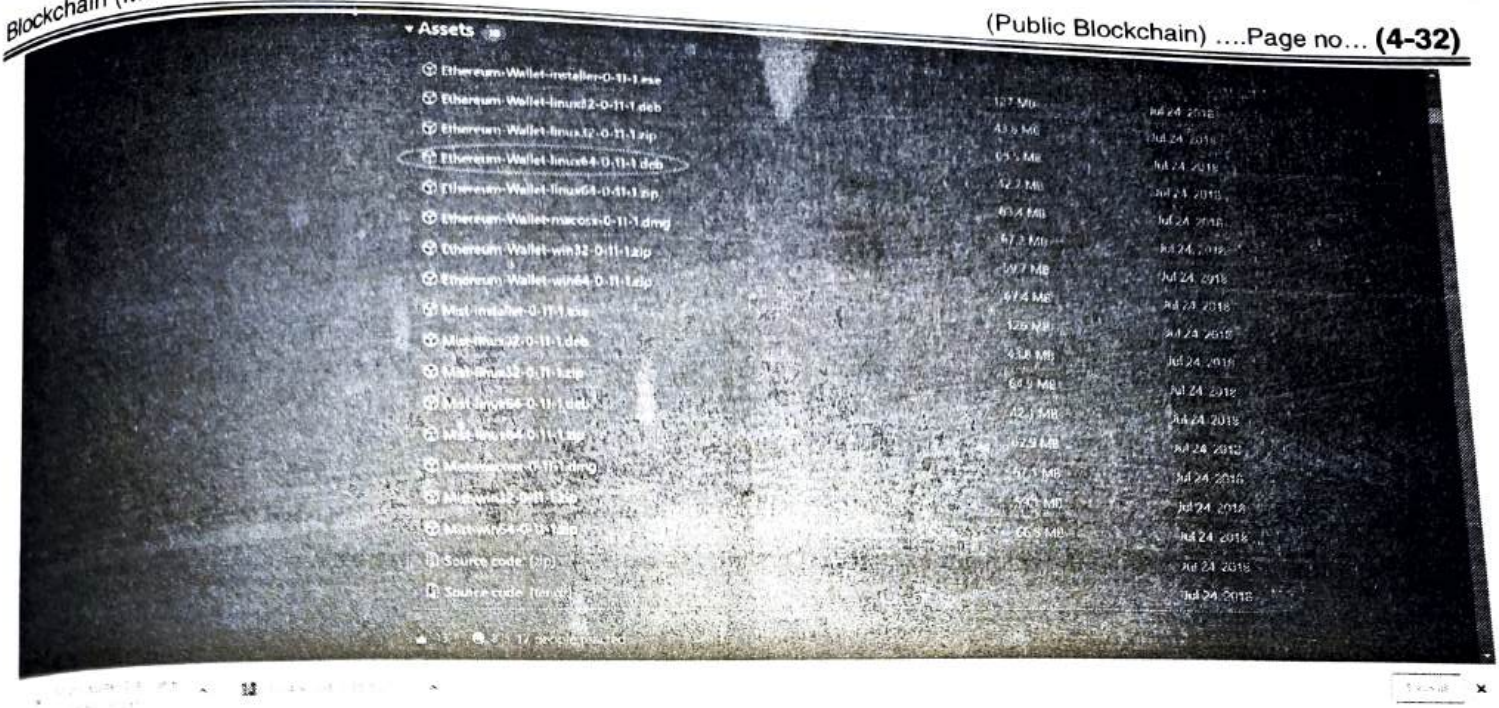
Ethereum Mist Wallet

- It is Ethereum's official blockchain wallet where an individual can hold his/her Ethereum assets.
- This wallet was created by a team of Ethereum Foundation so that decentralized applications and blockchain applications can be executed.
- Ethereum is the second-largest crypto by market capitalization, indicating that it is a developing ecosystem.
- Cybercriminals are often on a lurk looking to target this ecosystem.
- The Ethereum Mist Wallet was launched in 2017. It is a blockchain-based browser that serves as a platform for operating blockchain networks and decentralized applications.
- Thus, Mist is a search engine with several Dapps connected to it, and Ethereum is one of those.
- Note that Ethereum Mist wallet is compatible with Linux, Windows, and macOS.
- The purpose of the wallet is to hold assets and keep them safe until the owner feels the need for it. Ethereum Mist wallet is one of the safest Ethereum wallets that one can adapt to store his/her assets.
- Ethereum Mist wallet saves the private key on an individual's computer and not on third-party servers.
- The wallet also has multiple signature wallet options, thereby permitting multiple accounts. Thus, it is possible to have one mist wallet with multiple signatures.
- With adequate security measures such as recovery services, it is easier to monitor the signatory sign-ins and authorization of transactions.

Steps to setup an Ethereum Wallet Mist

- **Step 1:** Download Mist by visiting https://github.com/ethereum/mist/releases



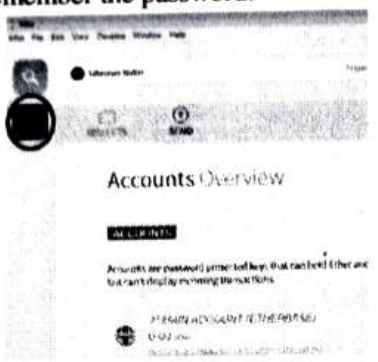


- **Step 2 :** Follow the standard normal procedure for completion of the setup of Ethereum Mist Wallet.

- **Step 3 :** The Mist icon appears. On clicking, the following window opens.

- **Step 4:** In case there is a need to install Geth for accessing Mist, then Geth can be downloaded and installed from https://geth.ethereum.org/downloads/

- **Step 5 :** Next, the Mist Wallet opens only after a password has been set for the account, as shown below. Note that the password cannot be reset, so remember the password.

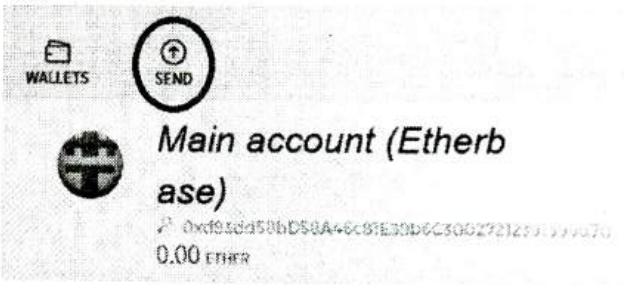


- **Step 6:** Click "MAIN ACCOUNT" and below it you can view and copy your Ethereum Mist wallet address.
- **Step 7:** Paste the address in the relevant exchange or wallet to send funds to your wallet.

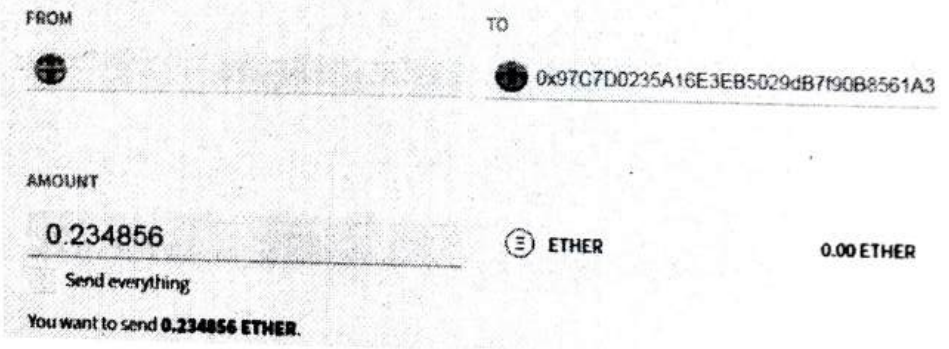
How to send payments on Ethereum Mist wallet ?

- **Step 1 :** Launch your wallet and select the account from which you want to send ETH.

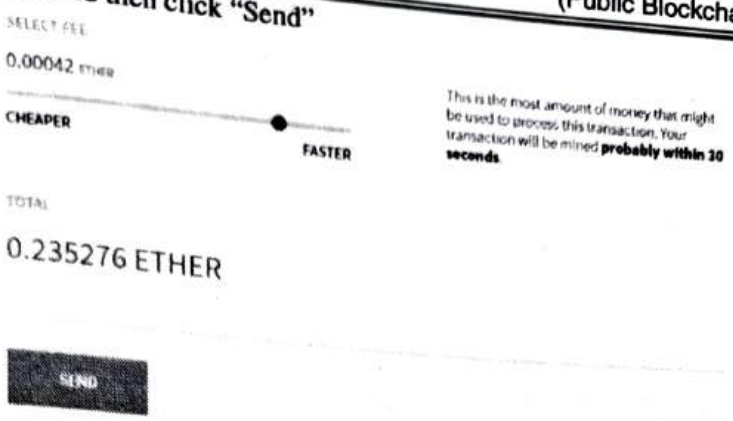
- **Step 2 :** Click on "Send"



- **Step 3 :** In the "To" box, enter your recipient's wallet address and then enter the amount



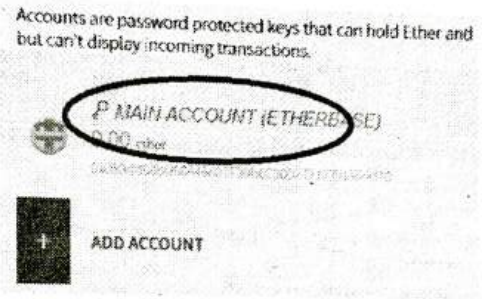
Step 4 : Select gas fee or leave it as it is and then click "Send"



- **Step 5 :** Enter your password to confirm transaction and then hit "Send"

How to receive payments on Ethereum Mist wallet?

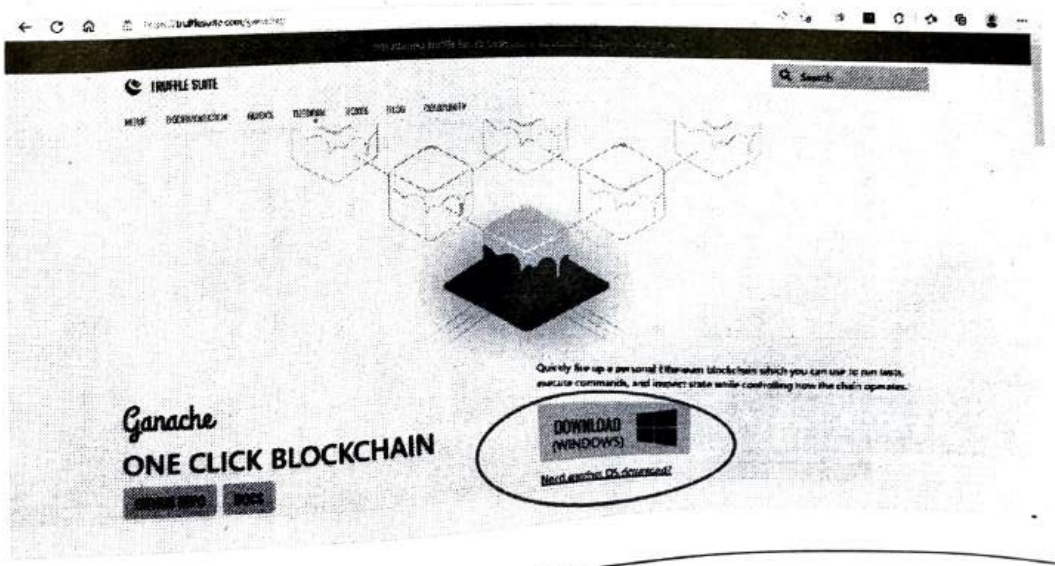
- **Step 1 :** Click on the Accounts Overview button
- **Step 2 :** Click "Main Account" and then copy your wallet address and send it to your counterparty.



Ganache for Ethereum Blockchain

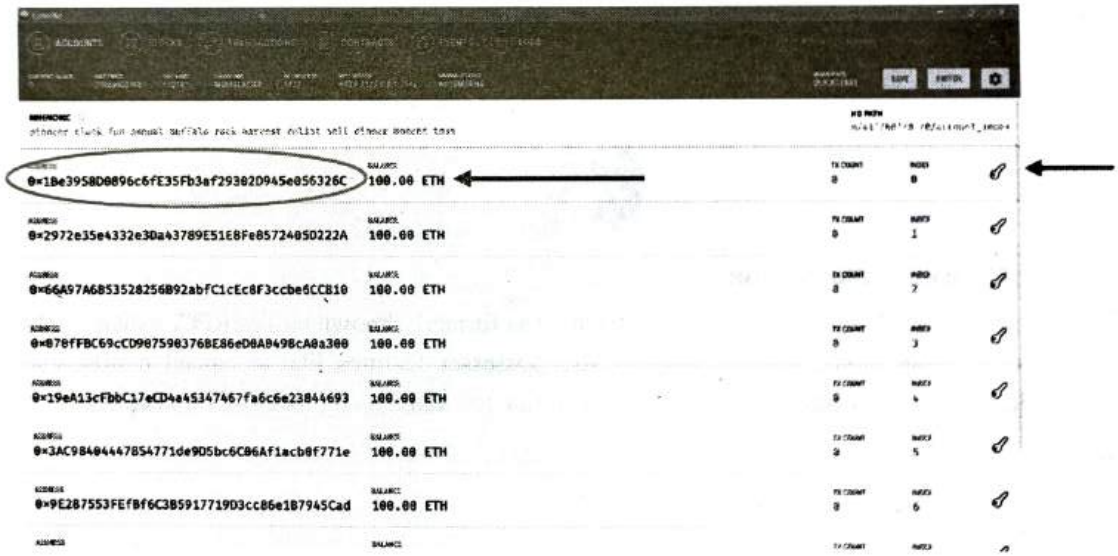
It is a personal blockchain for Ethereum. Ganache was formerly known as TestRPC, which can be used to deploy smart contracts, develop applications, and run tests. It also possesses features like advanced mining controls and configuring advanced mining options in a single check. It provides a link to examine all blocks and transactions.

- **Step 1 :** From Truffle Suite (i.e., https://trufflesuite.com/), Ganache can be downloaded (i.e. https://trufflesuite.com/ganache/).

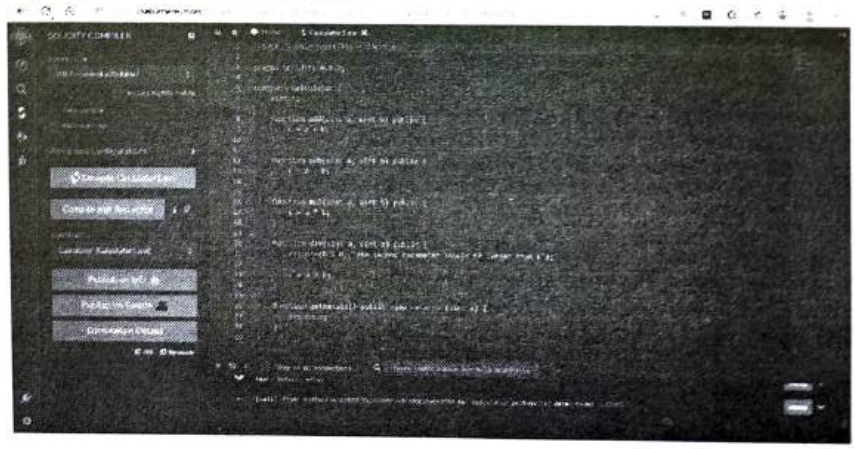


- **Step 2 :** On successfully downloading and installing Ethereum, the following GUI of Ganache is obtained. We select QUICKSTART ETHEREUM.

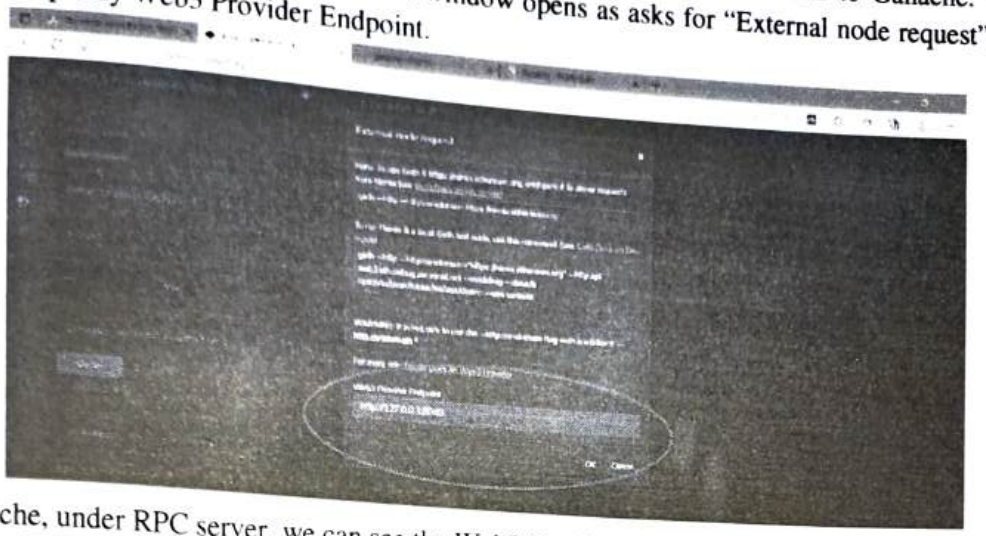
- **Step 3 :** Ganache has pre-determined set of accounts (i.e., 10 accounts) that can be used when we write codes in solidity wherein test ethers (100 ETH) are made available for each of these accounts. Each account possesses its own ADDRESS and Private key to facilitate transactions.



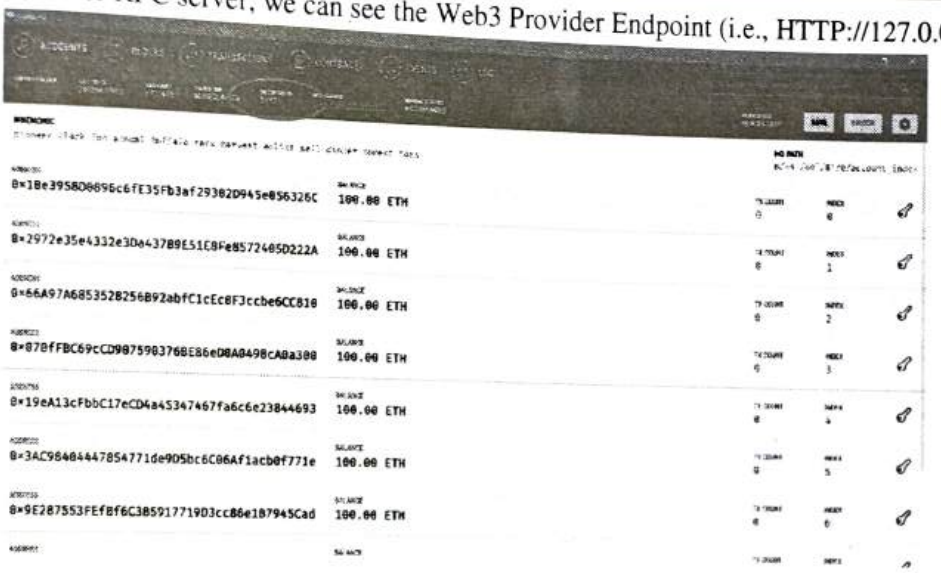
- **Step 4 :** Simple calculator program written in Remix IDE



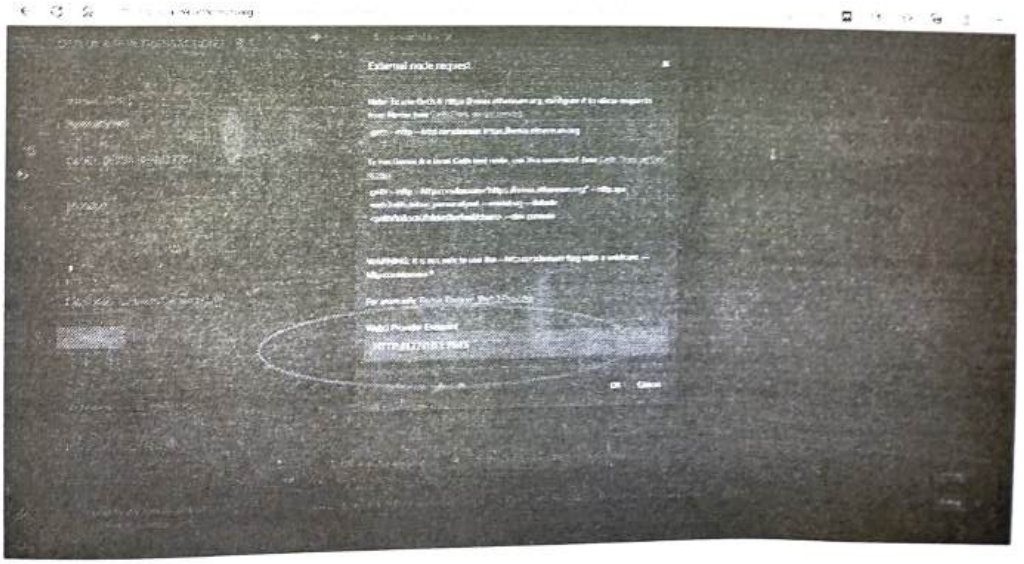
Step 5 : Connecting Remix IDE with Ganache. Once the simple calculator code is written, then under ENVIRONMENT, we need to select **Web3 Provider** to connect Remix IDE to Ganache. On selecting Web3 Provider, a pop-up occurs wherein a window opens as asks for "External node request". Under this, we need to specify Web3 Provider Endpoint.



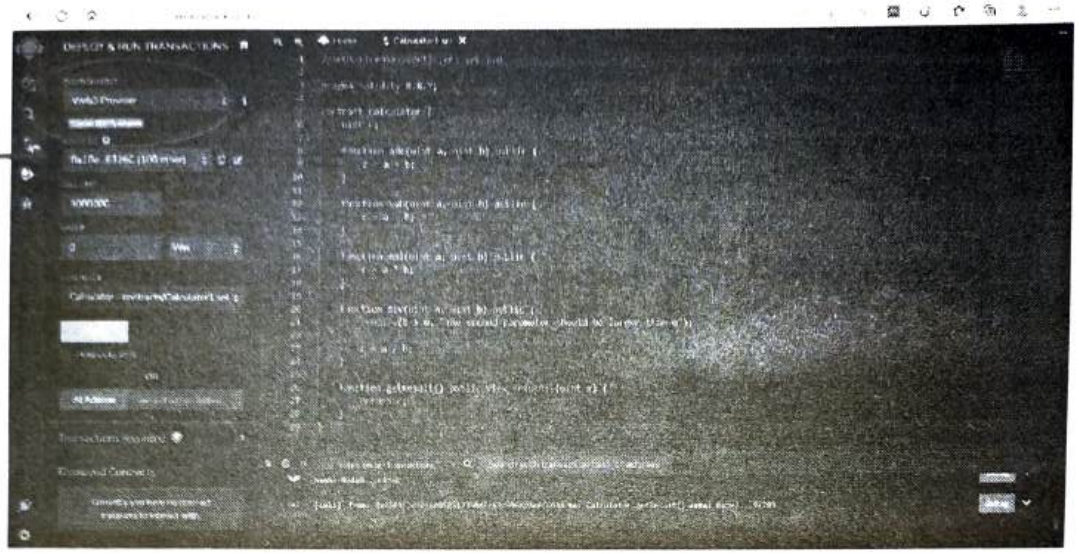
Step 6 : In Ganache, under RPC server, we can see the Web3 Provider Endpoint (i.e., HTTP://127.0.0.1:7545).



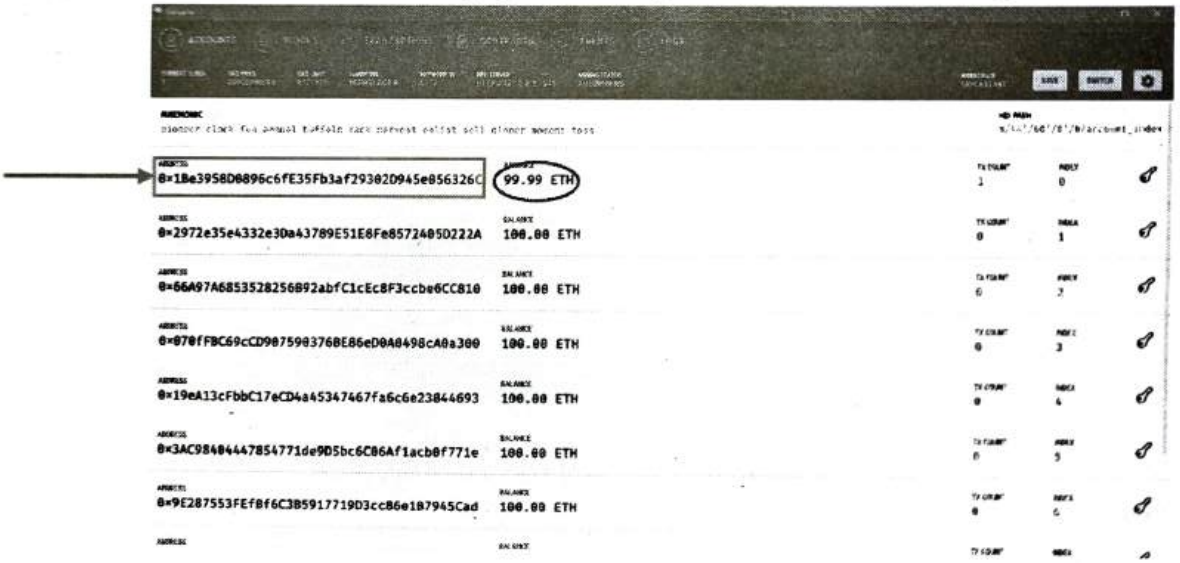
Step 7 : The RPC server from Ganache is specified in the Web3 Provider Endpoint, as shown below. Next, we click on OK.



- **Step 8 :** Now, Remix IDE is successfully connected with Ganache. Next, we deploy the contract.



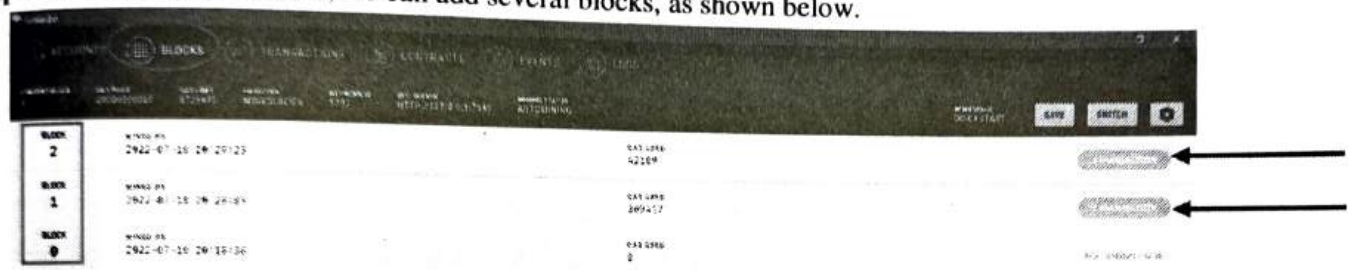
- **Step 9 :** As it can be seen that after deployment, for a specific address, certain ethers get deducted, as shown below.



- **Step 10 :** Furthermore, on clicking on the address, we can see information such as block no., date and time of mining, and gas used.

Step 11 : Moreover, on clicking on BLOCK 1, additional information can be viewed such as gas used, gas limit, mined on, block hash, and transaction (Tx) hash.

Step 12 : In a similar manner, we can add several blocks, as shown below.



Exploring etherscan.io

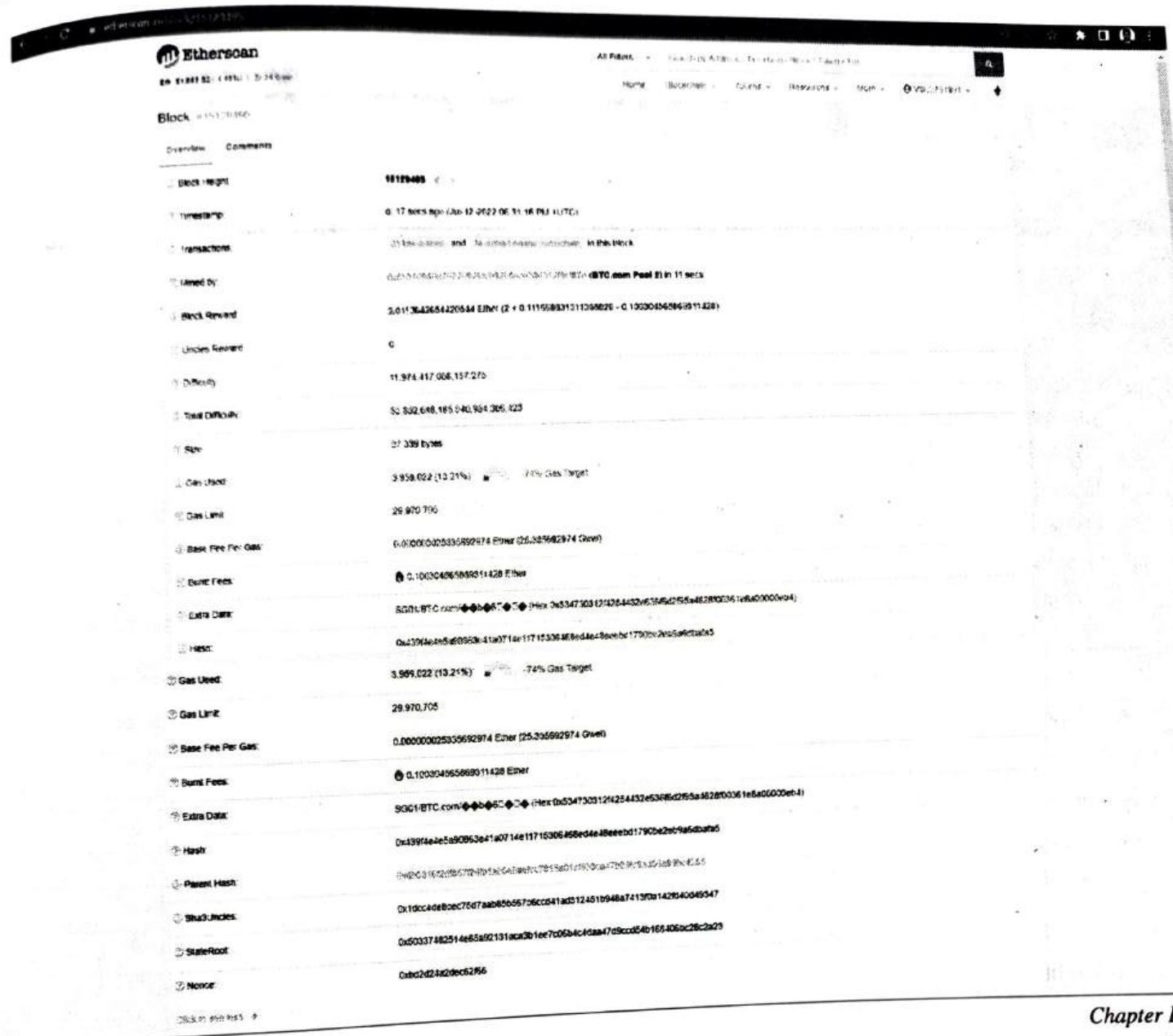
- **Etherscan**
- Etherscan permitsan individual to view the assets held on any public Ethereum wallet address.
- On Etherscan by entering any Ethereum address into the search box,one can see the current balance and transaction history of the wallet under consideration.
- Etherscan displays any gas fees and smart contracts involving that address.
- Etherscan can be used to :
  1. Calculate Ethereum gas fees with the Etherscan gas tracker.
  2. Lookup and verify smart contracts.
  3. View crypto assets held in or associated with a public wallet address.
  4. Observe live transactions taking place on the Ethereum blockchain.
  5. Lookup a single transaction made from any Ethereum wallet.
  6. Discover which smart contracts have a verified source code and security audit.
  7. Keep track of how many smart contracts a user has authorized with their wallet.
  8. Review and revoke access to a wallet for any decentralized applications (DApps).

Etherscan : Ethereum Blockchain Explorer(https://etherscan.io/)

- Contracts, transactions, and accounts

Ethereum daily transactions chart

Example of an Ether block Structure



Chapter Ends...

Module 5

Private Blockchain

Syllabus

5.1 Introduction, Key characteristics, Need of Private Blockchain, Smart Contract in a Private Environment, State Machine Replication, Consensus Algorithms for Private Blockchain - PAXOS and RAFT, Byzantine Faults: Byzantine Fault Tolerant (BFT) and Practical BFT

5.2 Introduction to Hyperledger, Tools and Frameworks, Hyperledger Fabric, Comparison between Hyperledger Fabric & Other Technologies

5.3 Hyperledger Fabric Architecture, Components of Hyperledger Fabric: MSP, Chain Codes, Transaction Flow, Working of Hyperledger Fabric, Creating Hyperledger Network, Case Study of Supply Chain Management using Hyperledger

5.1 PRIVATE BLOCKCHAIN SYSTEM

GQ. What is a private blockchain ? What are its key characteristics.

Introduction

- A private blockchain system is private as all permissions for the blockchain are kept centralized. They are also called as permissioned or centralized blockchains. They offer a closed ecosystem where all participants are well-defined, and only pre-approved entities can run nodes. Private blockchains are usually used in a business-to-business model.
- The degree of decentralization and transparency are entirely in the hands of anorganizations that runs and configures the blockchain. There is no secrecy. Private blockchains do not require mining to validate transactions or execute smart contracts. It doesn't require a crypto-economic incentive or tokens for those who run the nodes, i.e., miners. In a private blockchain, there exists a set of nodes that know each other, but they might not trust each other.
- Before a node can join a blockchain network, they have to go through some authentication or a pre-authorization mechanism through which they can validate themselves.
- In a private blockchain, users are trusted participants and their details are known. The network is not open and is not transparent to all. The network type is centralized and is a single point of failure.

5.2 KEY CHARACTERISTICS OF PRIVATE BLOCKCHAIN

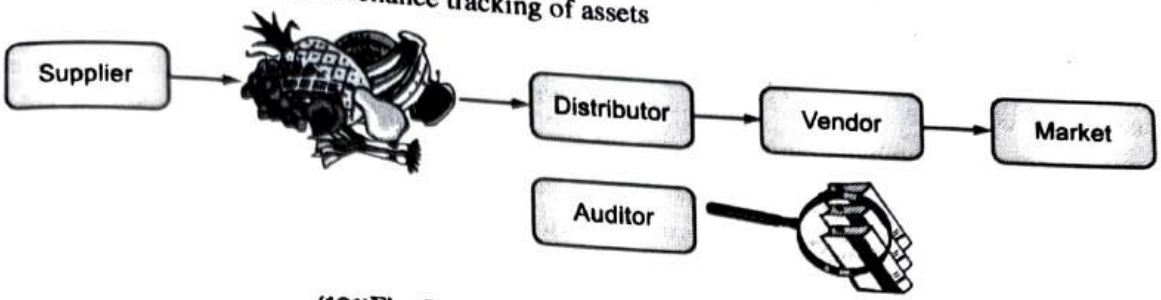
1. **Type of organization :** Single entity or organization
2. **Participants :** Known and trusted
3. **Access :** Fully restricted
4. **Type of network :** Centralized and single point of failure
5. **Operation :** Pre-approved participants can read/initiate transactions
6. **Verification :** A single validator node/central authority to create a block
7. **Immutability :** Secured by distributed consensus
8. **Consensus mechanism :** PoW or PoS consensus mechanism
9. **Security :** Depends on the blockchain architecture that is adopted
10. **Speed of transaction :** High. It takes a few seconds to create a block
11. **Energy consumption :** Low
12. **Scalability :** Better as high storage and computational power are not required.

Only pre-approved participants can read/initialize transactions in a private blockchain. There is no miner, private blockchains do not include the concept of incentivization. Distributed consensus mechanism is used to keep a private blockchain secure, such as PAXOS, RAFT, BFT, and PBFT.

5.3 PRIVATE BLOCKCHAIN EXAMPLES

- A use case or an example of a private blockchain is for a business application, such as to execute specific contracts among a closed set of participants. Private blockchains are used for provenance checking of assets, i.e., a specific asset is moved from a supplier to the distributors, vendors, or market.

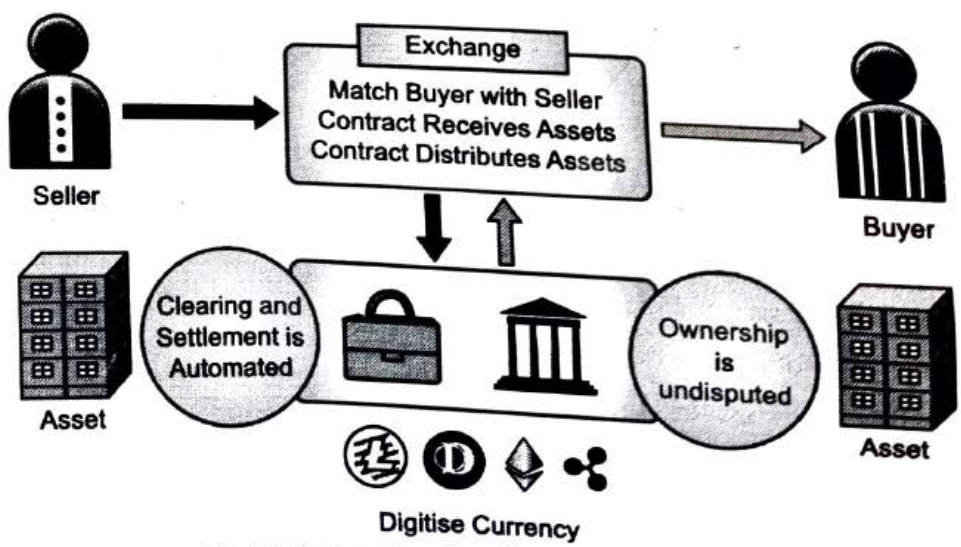
- At each and every stage, a log is maintained where people make an entry, which indicates that an asset has moved from one location to another or from one person to another person.
- Example of a private blockchain: Provenance tracking of assets



(1C1)**Fig. 5.3.1 : Example of a private blockchain**

5.4 SMART CONTRACT IN A PRIVATE ENVIRONMENT

- As discussed earlier, a smart contract is a self-executing contract in which the terms of agreement between a buyer and seller is directly written into the lines of code. So, a question arises that how such types of a contract may get executed or how blockchain can be helpful in executing such type of contracts.
- Remembering bitcoin scripts, as mentioned earlier. It is a small code through which an individual can change the script to control how the money that is being transferred to someone can be spend further/later on. The script can be written in such a way that either one can use the money immediately or one can use the money after three months.
- Now, in case of a smart contract, this idea can be expanded further by using some general purpose programming language so that complex types of contracts can be executed if certain conditions are met/satisfied.
- In the Fig. 5.4.1 below, we can see that there exists a seller and buyer, and there exists certain assets where a contract makes a match with the buyer and the seller. When a buyer buys something, there would be certain scripts that would be executed at the seller's side and there would be certain scripts that would be executed at the buyer's side.
- When the buyer transfers the money to the seller, the ownership of the asset goes from the seller to the buyer, and the code that is written actually validates whether the buyer has transferred certain amount of money.This is how a smart contract can be executed in a closed / permissioned /private environment with the help of a blockchain.
- In a typical business platform, there exists fixed number of buyer and sellers in the market, and the buyers and sellers can register to a central portal where everyone can know each other; however, there may not be any trust relationship between them. There could be a possibility that some sellers could be fraud who is simply taking the money and providing the asset to the buyer. Such type of fraud or malicious activities can be prevented with the help of a permissioned/private blockchain.



(1C2)**Fig. 5.4.1 : Smart contract in a private environment**

☞ Design limitations in a permissioned environment

GQ. Elaborate on the **design limitations in a** permissioned environment.

- **Sequential execution :** The concept of blockchain originated from a permissionless environment such as bitcoin. In that, the issue that was observed is that transactions are executed sequentially based on the consensus. If a certain transaction gets verified or committed in one transaction, then that transaction will get executed first and the transaction that is committed later on, it will be executed next.
- So, the request to the smart contracts/applications are ordered by consensus and they executed in the same manner. Such type of a sequential order gives bound on the effective throughput (i.e., the throughput is inversely proportional to the commitment latency). This could be a possible attack on a smart contract platform (i.e., an attacker can introduce a contract which will take a lot of time to execute).
- Therefore, if certain contracts take a lot of time to execute, then other smart contracts will never get a chance to execute because once consensus for the previous contract has been reached then only it is possible to execute contracts that have been submitted later on.
- **Non-deterministic execution :** The implementation of a smart contract needs some programming language that will offer more power compared to bitcoin script. As bitcoin script is not a Turing complete language, it does not support loops and other constructs. Golang is one other the languages preferred by developers to execute smart contracts.
  - In Golang, there exists a construct called as map. Map is a data structure that consists of keys and values where every value is associated with one key.

```
m : = map [string] string {"key1" : "val1", "key2" : "val2" };
for k, v : = range m {
    fmt.Printf("key [%s] value [%s]\n", k, v)
}
```

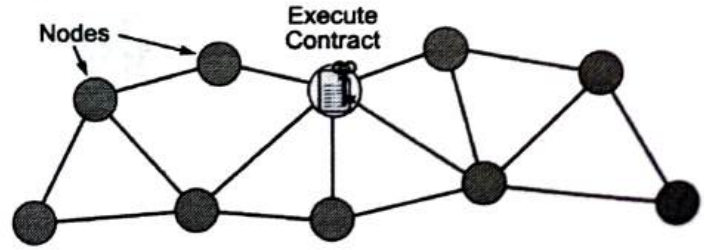
  - From the above code snippet, it can be concluded that iteration over a map may produce a different order in two executions.
- On the other hand, smart contract execution should always be deterministic; otherwise, it may happen that the system might lead to an inconsistent state (i.e., many fork in the blockchain). The solution for this is to have a domain-specific language (DSL) for smart contract.
- **Execution of the smart contract on all nodes :** In general, smart contracts are executed on all nodes and the state is propagated to others, and this is how consensus is reached. Here, consensus mean propagate the same state to all nodes and verify whether the states match. However, the problem here is are there sufficient number of trusted nodes that validate the execution of smart contracts.

Is there a need to execute smart contracts at each node in a permissioned model?

It is not necessary to execute smart contracts at each node in a permissioned model. Only state synchronization is needed across all nodes.

The process followed is as follows :

1. Execute the contract in one node.
2. After executing the contract in one node, propagate the state of the contract to neighboring nodes, which in turn get propagated further.



(1C3)**Fig. 5.4.2**

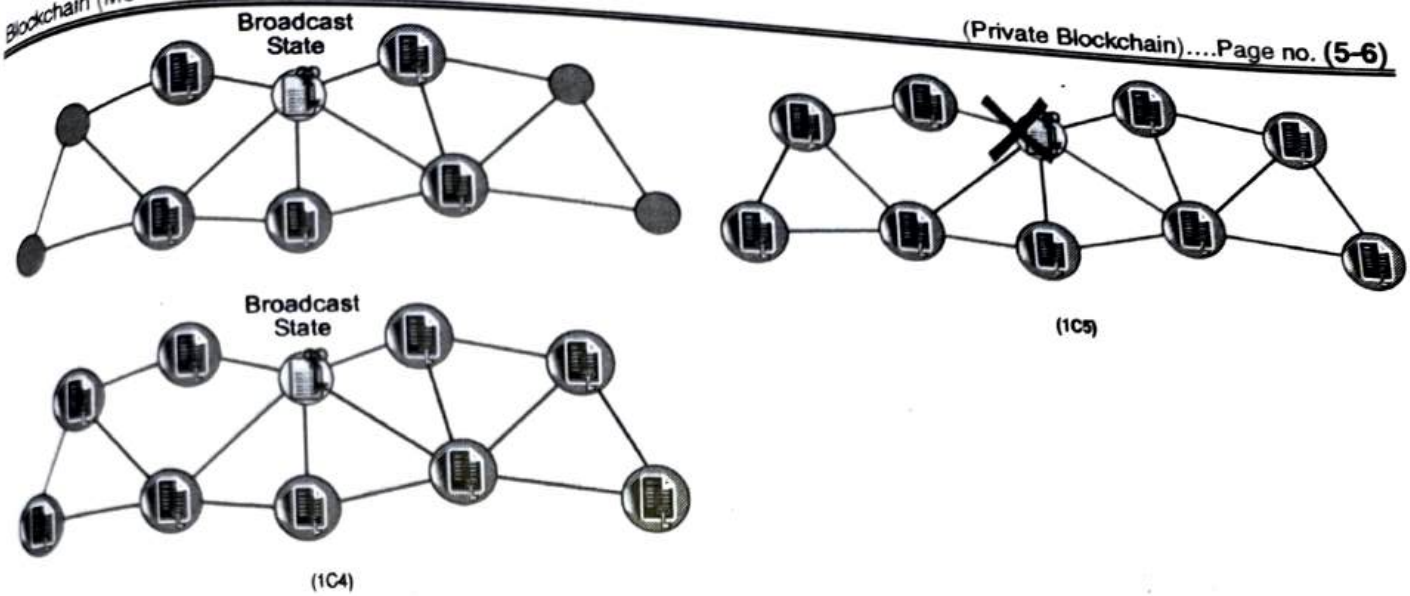
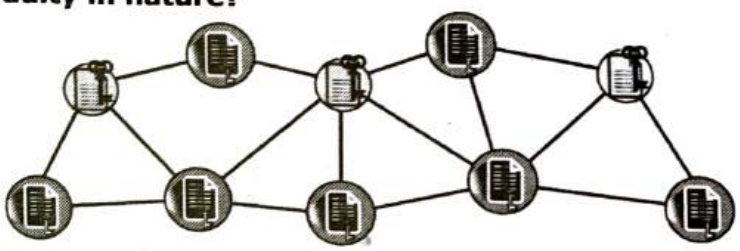


Fig. 5.4.3

So, this is how every node in the system will receive the same state of the contract.

What if a node that executes the contract is faulty in nature?

- If a node that executes the smart contract is faulty, then the entire system will be down, and it will not be able to make any progress further.



(1C6)**Fig. 5.4.4**

- The above problem can be resolved using state machine replication. In other words, it means that one should execute a contract at a subset of nodes rather than a single node and ensure that the same state of the contract is propagated to all the nodes.

(1C7)**Fig. 5.4.5**

5.5 STATE MACHINE REPLICATION

GQ. Describe the concept of state machine replication.

- A state machine can be characterized by a set of parameters that are a set of states (S) based on system design. In the Fig. 5.5.1, there are three states S1, S2, and S3. There exists a set of inputs (I) that specify how a system will behave. There are two inputs 0 and 1, as shown in the Fig. 5.5.1. There exists a set of outputs (O). As shown in the Fig. 5.5.1, S3 is the final output of the system, which is represented by the state machine.

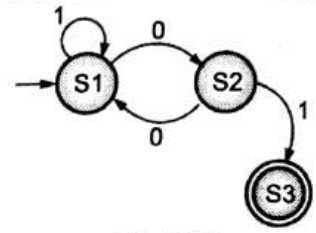
- Next, there is a transition function (S × I → S). Here, from state S1, if we take 0 as an input, then S2 will be the output that will be produced (i.e., (S1, 0) → 0).
- There is also an output function (S × I → O). In addition, there would also be a designated start state (i.e., in the Fig. 5.5.1, S1 would be the designated start state).

☞ Example of a smart contract represented as a state machine

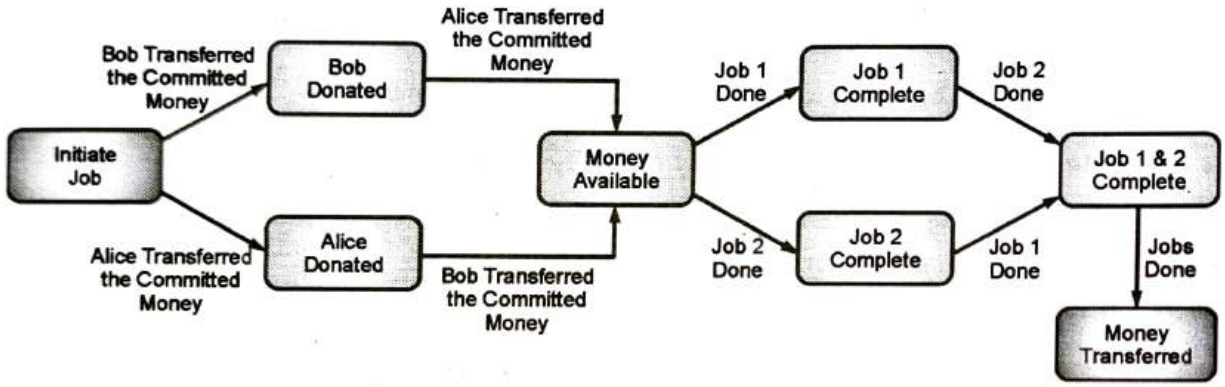
GQ. How is a smart contract represented as a state machine.

Smart contract state machine: Crowd funding platform

- As mentioned earlier about crowd funding, there exists a set of people who have funds available with them and are ready to donate the fund if certain jobs are being done.
- So, project proposals are coming from project proposers where a proposer proposes certain jobs/tasks/projects, and if certain funding agencies are funded, and if they find the proposal interesting, then they would fund for the proposal in a collective manner.

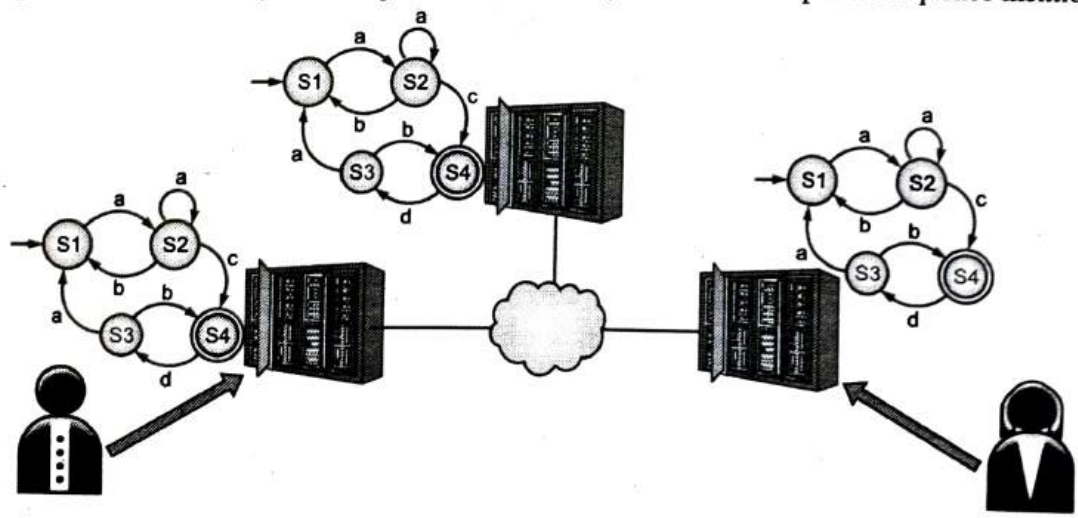


(1C8)**Fig. 5.5.1**



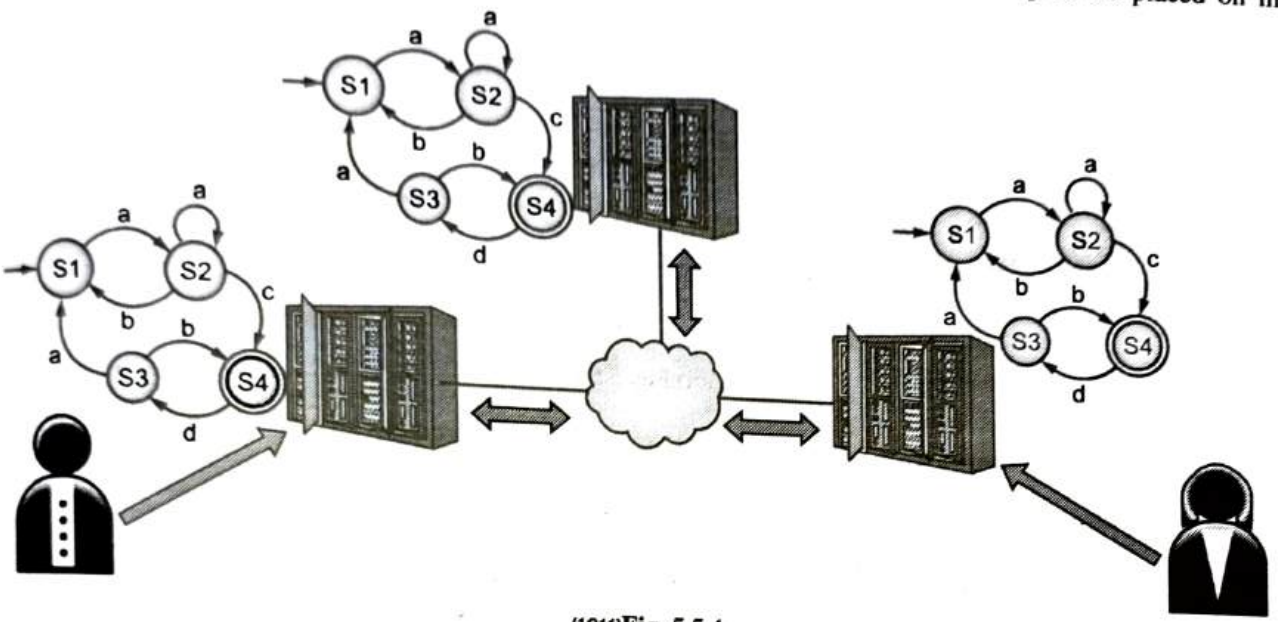
(1C9)**Fig. 5.5.2 : Smart contract state machine : Crowd funding platform**

- There exists an initiation state where some users have submitted certain proposals. Now, when someone likes this proposal, then the interested person would transfer the money. For instance, Bob has committed the money to the system; so, a state is reached which shows that Bob has donated the money. In a similar manner, if Alice has committed the money, then a state is reach which indicates that Alice has donated the money.
- There could be a possibility that either Bob has donated the money first or Alice might have donated the money first. Irrespective of the order (*i.e., Bob first, Alice second OR Alice first, Bob second*), the money is available in the system. Once money is available in the system, the job can be executed, but there is no specific sequence mentioned.



(1C10)**Fig. 5.5.3**

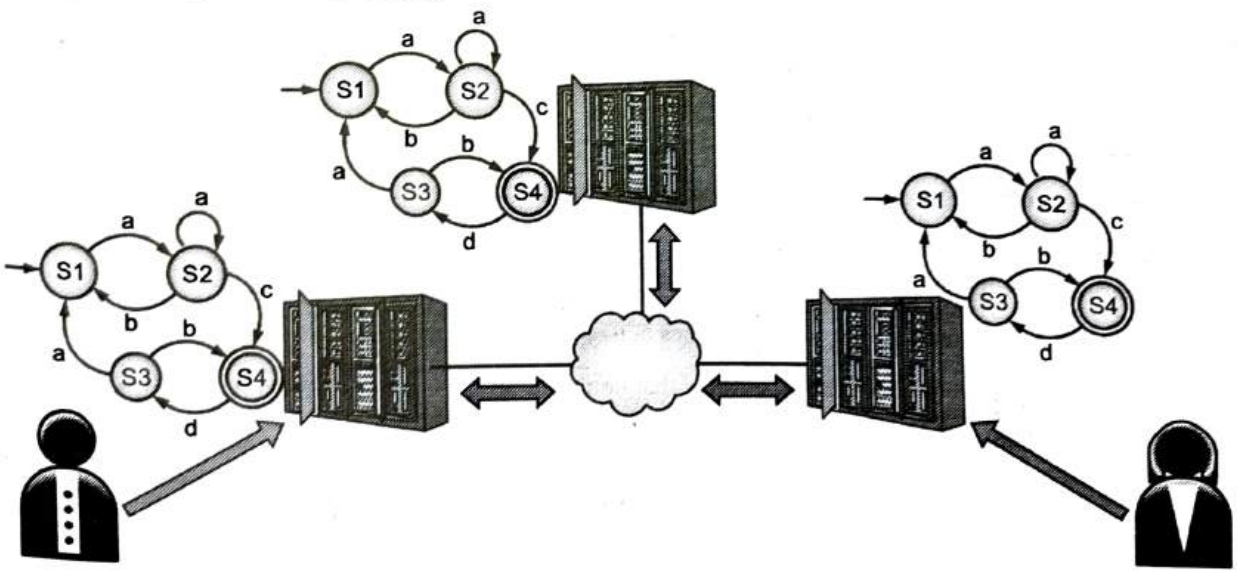
- It could be possible that either Job 1 was done first and then Job 2 OR Job 2 was done first and then Job 1. Once both the jobs are done, then the smart contract will transfer the money from the funding agencies (i.e., Bob and Alice) to the project proposers. This how a smart contract can be represented in the form of a state machine. Indeed any algorithm can be represented in the form of a finite state machine. In a typical distributed architecture, distributed state machine replication works in the following manner.
- There are multiple servers that work in a distributed manner. Firstly, state machine copies are placed on multiple independent servers.



(1C11)**Fig. 5.5.4**

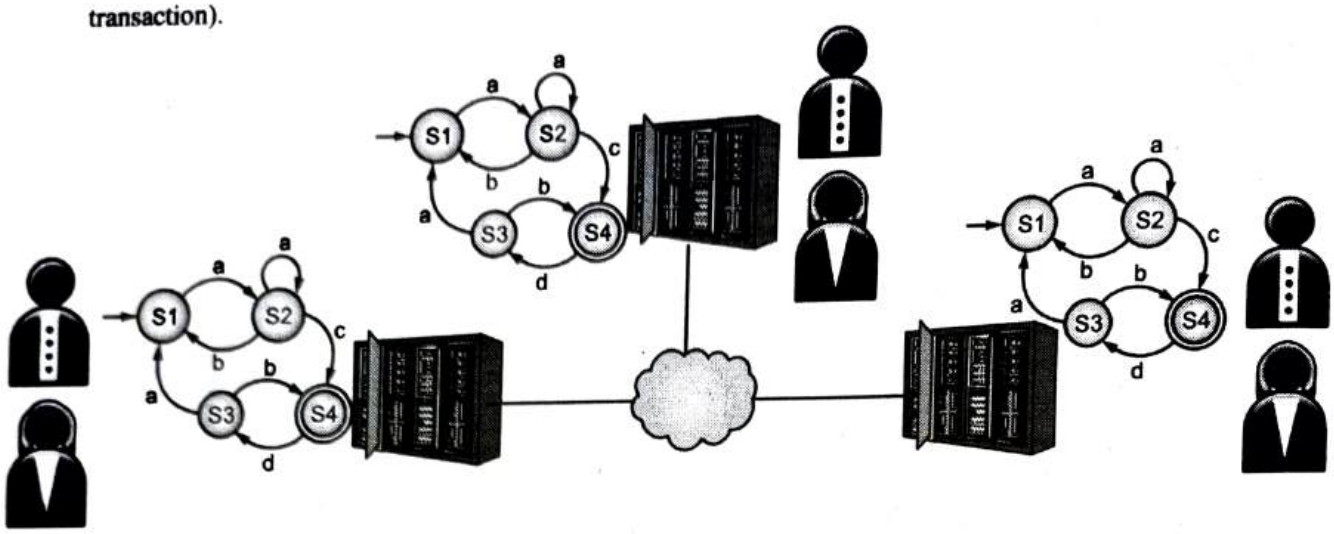
- Next, the servers receive client requests as an input to the state machine. For instance, Alice and Bob are submitting their requests to different servers. Now, if the request is executed independently at each server, then the state that would be there in each of the servers would be different. Collectively, there is a need to ensure that all the three servers reach to a state after a certain time that both Alice and Bob have transferred their share of money.
- In order to achieve this, the state machine replication mechanism uses the following principle:

1. Propagate the inputs to all the servers



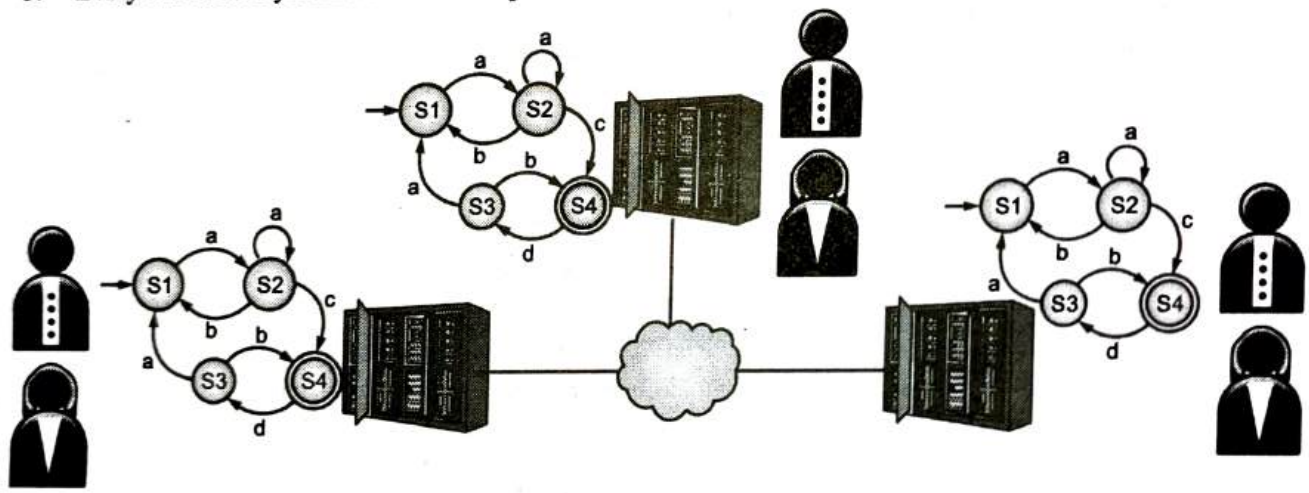
(1C12)**Fig. 5.5.5**

2. Order the inputs based on some ordering algorithms (i.e., have an associated time stamp with every individual transaction).



(1C13)Fig. 5.5.6

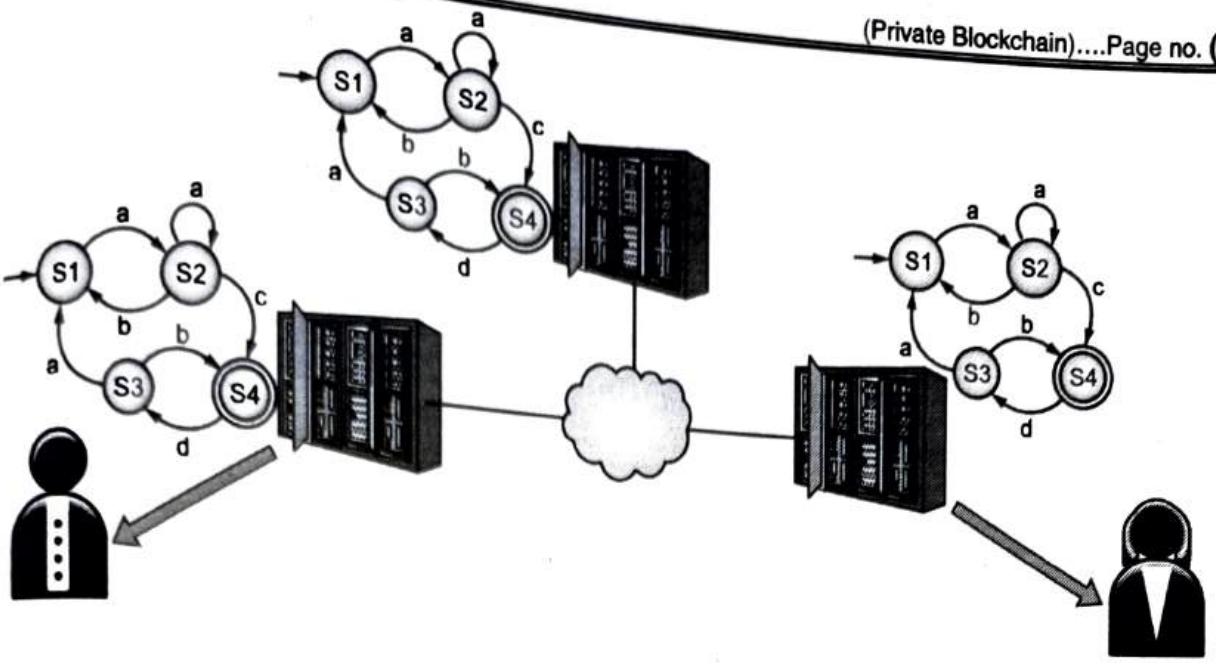
3. Every individual system executes the inputs based on the order decided individually at each server.



(1C14)Fig. 5.5.7

Once the information is executed in each server with the help of a particular ordering algorithm, then it is evident that first the transaction corresponds to Bob, which will get executed, and then the next transaction corresponds to Alice, which will get executed. Ultimately, all the servers will reach the same state implying that both Bob and Alice have transferred their share of money.

4. The next step is to synchronize the state machines across the servers to avoid any failure.
5. In this algorithm, there could be certain outputs that could be produced. In other words, it output state is produced, then inform the clients about the output.



(1C15)Fig. 5.5.8

There are two disadvantages of using this system:

1. Maintenance of an ordering service is needed.
2. In the presence of a failure, there is a need to ensure that all individual servers are in the same page.

☞ Reasons for using state machine replication-based consensus over a permissioned/private blockchain

GQ. Why state machine replication-based consensus is preferred over a permissioned/private blockchain ?

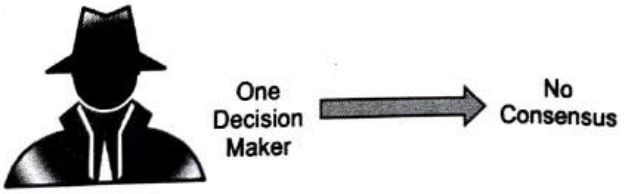
- The network is closed, i.e., the nodes know each other, and so state replication is possible among the known nodes.
- It avoids the overhead of mining (i.e., there is no need to spend anything, such as power, time, and bitcoin, other than message passing.
- Nonetheless, consensus is needed, which means that machines can be faulty or they could behave maliciously.

5.6 DIFFERENT ALGORITHMS OF PERMISSIONED BLOCKCHAIN

☞ Why distributed consensus ?

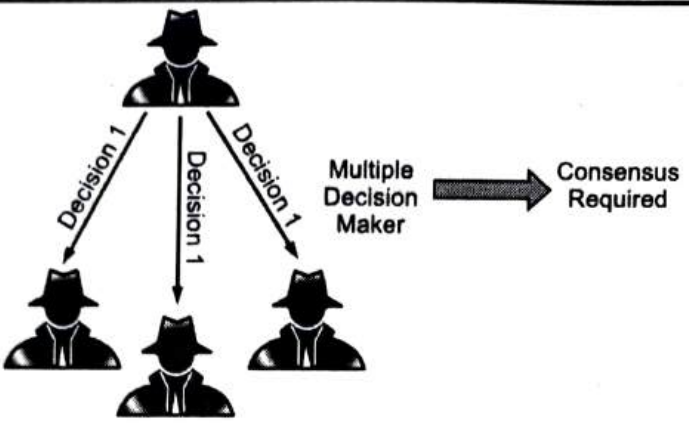
GQ. What is distributed consensus? What are the applications of state machine replication in a distributed environment?

In case of a single decision maker, we do not require any consensus.



(1C16)Fig. 5.6.1

- Whenever there are multiple decision makers and in a collective way the decision makers wants to come to a certain decision, then consensus is required.



(1C17)**Fig. 5.6.2**

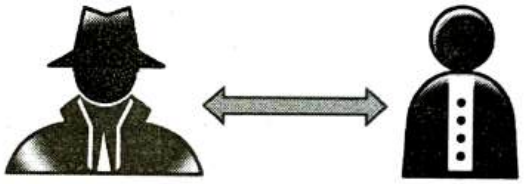
- Distributed consensus helps in reaching a certain agreement in case of a distributed computing. So, in case of state machine replication concept, we replicate a common state so that all the processes have the same view of the state.

☞ Applications of state machine replication in a distributed environment

- Flight control system: When there are multiple flights (like Boeing 777 and 787) and they want to coordinate their positions amongst themselves, then state machine replication technique can be applied in a distributed system to achieve consensus. Fund transferring system: In a closed environment like bitcoin and cryptocurrencies, state machine replication technique can be applied to achieve consensus.
- Leader election/mutual exclusion: When all the nodes need to collectively elect one leader in a system, achieving consensus is necessary so that all nodes are able to select one leader or the same leader is selected at the end of an election round. So, there is no need of consensus in a single node process.

☞ Is distributed consensus needed when there are two nodes?

- In case of a crash fault or network/partitioned fault (i.e., if a node behaves maliciously), achieving consensus is impossible.



(1C18)**Fig. 5.6.3**

- Therefore, in order to reach consensus, more than two nodes are needed.

☞ Consensus for three processes

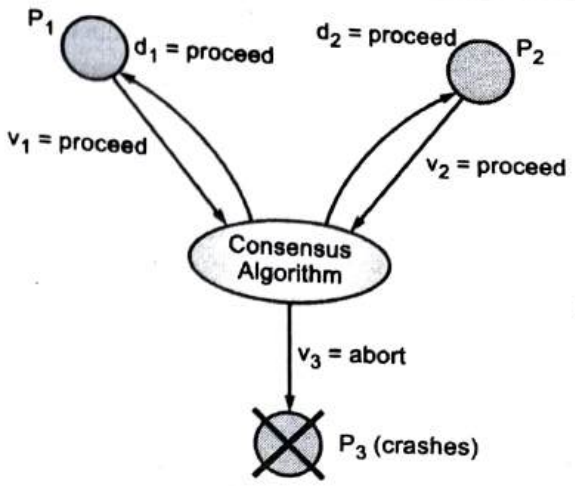
In order to reach consensus for three processes, every process can have one of the three states.

- **Undecided state :** In this state, the process or the node has certain value v_i from a set of feasible values D. So, every node has proposed some value and they are in the undecided state.
- **Communication state :** Here, they are exchanging the value amongst themselves. So, by exchanging the value and applying the consensus algorithm, a decided state can be reached.
- **Decided state :** Here, the decision is set that everyone in the network agrees on the variable d_i.

Requirements of a consensus algorithm

GQ. State the requirements of a consensus algorithm.

- **Termination :** Eventually each correct process sets its decision variable.
- **Agreement :** The decision value of all correct processes is the same.
- **Integrity :** If the correct processes proposed the same value, then any correct process in the decided state has chosen that value.
- Different algorithms for ensuring consensus in a typical distributed system that are applied to a permissioned blockchain and these algorithms are based on the principle of state machine replication.



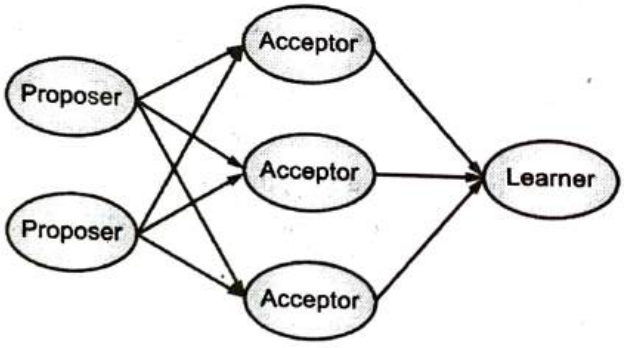
(1C19)Fig. 5.6.4

Crash or network/partitioned faults

GQ. What do you mean by crash or network/partitioned faults ?

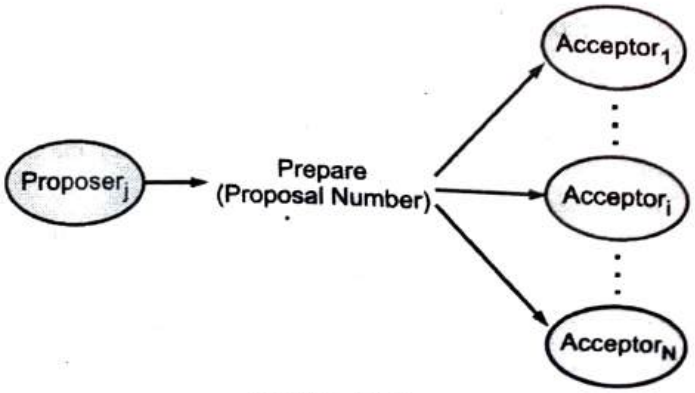
GQ. Explain PAXOS consensus algorithm.

- **PAXOS**
  - It was the first consensus algorithm that was proposed by Leslie Lamport in 1989 and the objective of PAXOS was to choose a single value under a crash or network/partitioned fault.
- The system process consists of the following parameters :
  - Making the proposal
  - Accepting a value
  - Handing failures



(1C20)Fig. 5.6.5

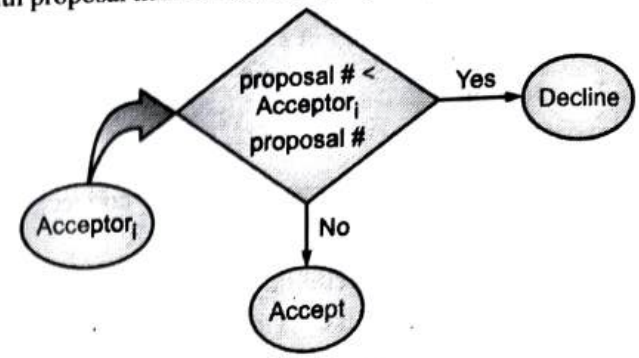
- **Types of nodes in PAXOS**
  - **Proposer :** It proposes values that should be chosen by the consensus.
  - **Acceptor :** It forms the consensus and accept values. The acceptors when they are hearing certain proposals from the proposer, they either accept or reject it.
  - **Learner :** It will learn which value was chosen by each acceptor, and then they will collectively accept a specific value. Everyone is the learner in the network, which learns that what is the majority decision in the network.
- **Making a proposal : Proposer process**
  - A proposal is initially prepared with a proposal number by a proposer *j*. The proposal number needs to be good enough so that it gets accepted by several acceptors.



(1C21)Fig. 5.6.6

- The proposal number forms a timeline, and the biggest number is considered up-to-date. For instance, we have proposal p_1 with proposal number 30 and proposal p_5 with proposal number 35, then proposal p_5 gets accepted.

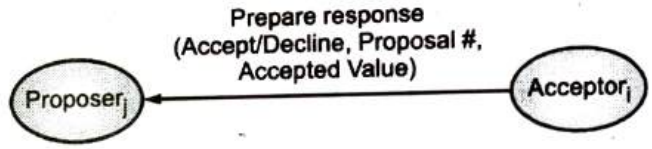
- **Making a proposal :** Acceptor's decision making
  - Each acceptor compares the received proposal number with the current known values for all proposer's prepare message.
  - From the above Fig. 5.6.7, it is evident that if the proposal number is less than the acceptor's current proposal number, then it is declined else it is accepted.



(1C22)Fig. 5.6.7

- **Making a proposal :** Acceptor's Message/Response
  - Based on the proposal number, the acceptor prepares the response. In the response, the acceptor can either accept or decline a message based on the proposed method.
  - The biggest number that the acceptor has seen till now that number the acceptor will add in the response message and it will also add the accepted values that has been accepted from the proposal. So, the accepted values are informed to the proposer.

- **Accepting a value :** Proposer's Decision Making
  - A vote is taken based on a majority decision. The proposer looks whether majority of the acceptors have rejected the proposal. If yes, then the proposer updates with the latest proposal number.

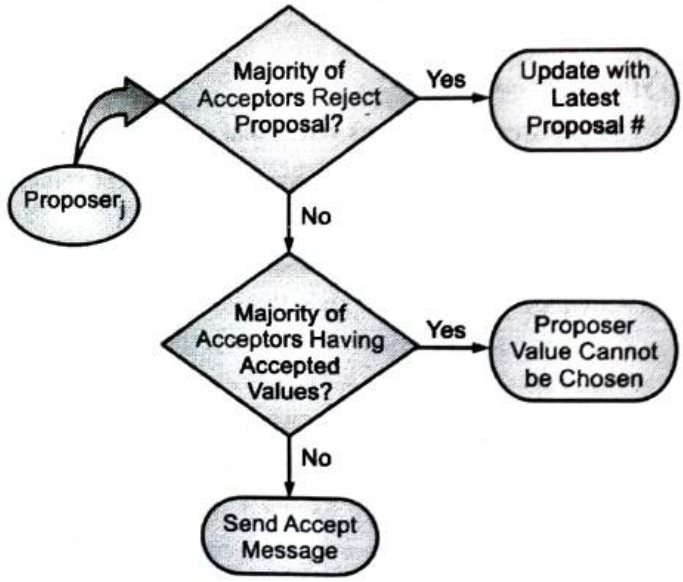


- **Accept/Decline:** Whether prepare accepted or not.
- **Proposal Number:** Biggest number the acceptor has seen.
- **Accept Values:** Already accepted values from other proposer.

(1C23)Fig. 5.6.8

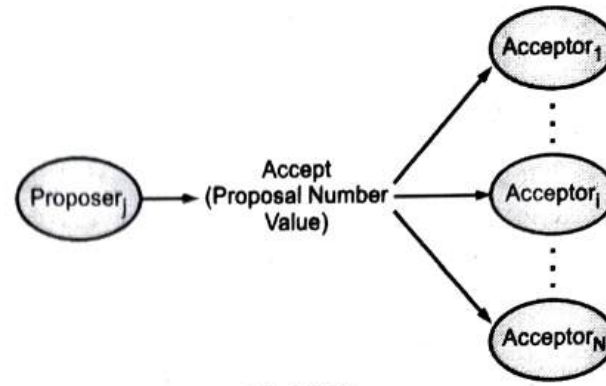
  - If the acceptors have not rejected it, then the proposer sees that majority of the acceptors have accepted the values.
  - If majority of the acceptors have accepted the value, then the proposer's value cannot be chosen.
  - If majority of the acceptors have not accepted the values, then an accept message is sent.

So, the idea is that whenever the majority of the acceptors are sending some accepted values. If they have accepted a proper's value, then the value that has been shared comes to a consensus.



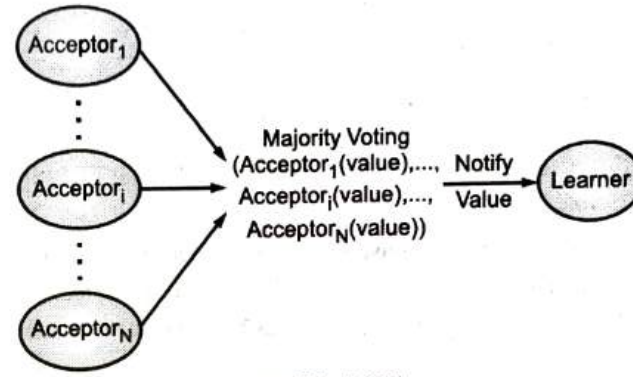
(1C23)Fig. 5.6.9

- Accepting a value : *Accept Message*
  - The final stage is accept message. The proposer sends the accept message to all the acceptors. The accept message consists of the proposal number.
  - Note that the proposer knows that his/her proposal has been accepted. The accept message also consists of a single value that is proposed by the proposer.



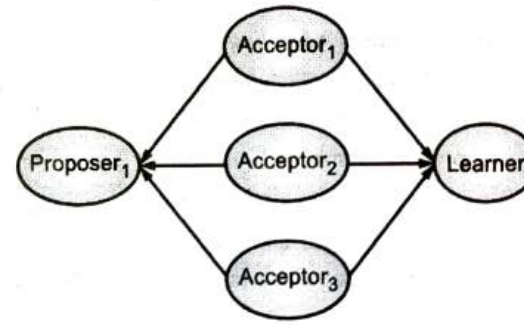
(1C25)**Fig. 5.6.10**

- **Accepting a value :** *Notifying Learner*
  - Whenever the acceptor accepts the value from the proposer, it informs the learner about the majority voted values; so, everyone learns that what is the majority voting in the environment.



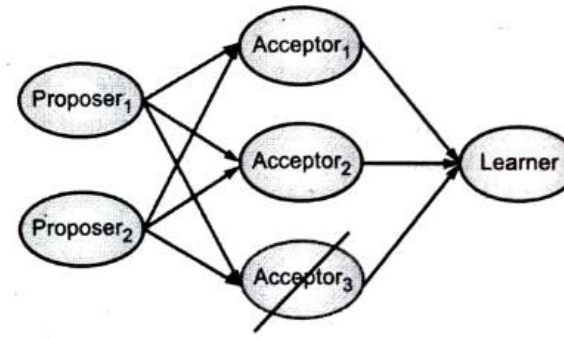
(1C27)**Fig. 5.6.11**

- **Single Proposer :** *No Rejection*
  - If there is a single proposer in the system, then every acceptor will accept that proposal because that will be the biggest.



(1C28)**Fig. 5.6.12**

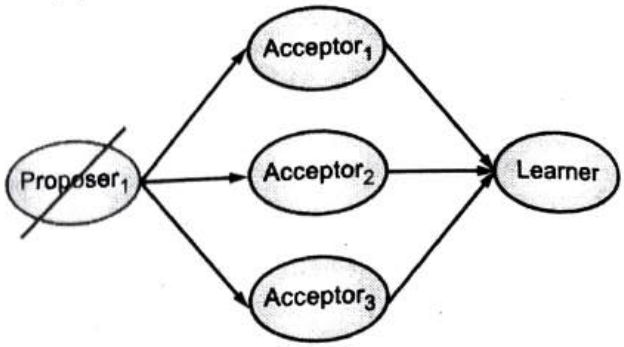
- **Handling Failure :** ***Acceptor Failure***
  - Certain acceptors might fail during the *prepare* phase. However, there could be no issues because there are other acceptors who can hear the proposal and vote either for the proposal or against the proposal.
  - There is a possibility that an acceptor may fail during the *accept* phase. However, there could be no issues because there are other acceptors who can vote for the proposal.



(1C29)**Fig. 5.6.13**

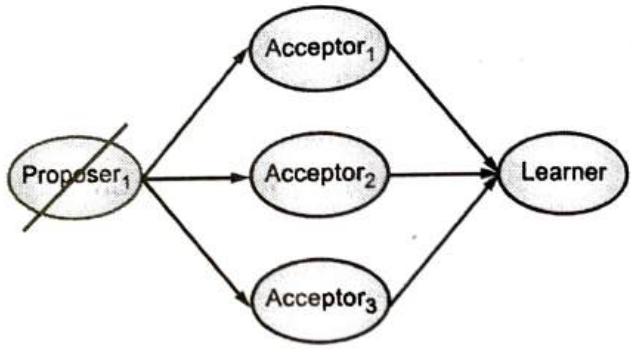
- If more than $\frac{N}{2} - 1$ acceptors fail, then no proposer gets a reply and no values can be accepted, and it is not possible to achieve consensus.

- **Handling Failure :** *Proposer Failure*
  - If the proposer fails during the prepare phase, then the acceptors wait for a certain time and then someone else (i.e., one of the acceptors) becomes the proposer.



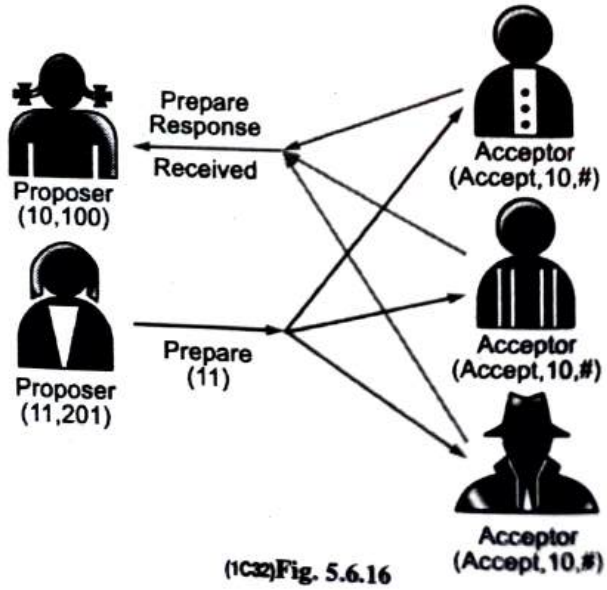
(1C30)**Fig. 5.6.14**

- **Handling Failure :** *Proposer Failure*
- The proposer fails during the *accept* phase. However, the acceptors have already agreed upon whether to choose or not to choose the proposal based on the majority of the vote.
- So, the acceptors have shared the majority of the votes amongst themselves and from there the acceptors can find out whether the proposal has been accepted or not.



(1C31)**Fig. 5.6.15**

- **Handling Failure :** ***Dueling Proposers***
  - There can be an interesting attack here. Assume that there are two proposers. Proposer 1 has sent a proposal (10, **100**) to all the acceptors. Also, Proposer 2 has sent a proposal (11, **201**) to all the acceptors with a higher proposal number.
  - Now, assume that Proposer 1 who is an attacker. When Proposer 1 observes that there is another proposer with a higher proposal number, Proposer 1 sends a proposal with a higher proposal number. For instance, she sends (10, **301**). So, in this manner, there could be dueling between proposers and reaching consensus could be difficult.
  - To break such dueling proposals, some identities/numbers are used. In other words, when dueling proposals are obtained, the proposers with lower IDs are blocked, and this is how the problem can be solved.
  - In order to do this, certain algorithms need to be executed. The most prominent one is the leader election algorithm. In this algorithm, one of the proposers is selected as a leader.



(1C32)**Fig. 5.6.16**

- As it is quite evident that PAXOS is a consensus algorithm, it can be used as a leader election algorithm.

The overall idea of PAXOS is that the proposer is proposing certain views to all the acceptors, and the acceptors collectively look into the proposal and if they agree with the proposal they send the accept message and by receiving that particular accept message if the proposer finds out that his/her message has been accepted then he/she sends back the accept to all other acceptors and then through the majority voting, the acceptors come to a consensus protocol.

This simple view of the PAXOS is easier to understand. It is just like making one selection. But, in reality, there exists a sequence of selection or a sequence of choices rather than having a single choice. So, such kind of a system is called as a multi PAXOS system.

Multi PAXOS system means a sequence of choices are made by applying repeated PAXOS protocol. Nonetheless, applying repeated PAXOS protocol is complicated because all the messages need to be exchanged amongst each other again and again, thereby increasing the complexity of PAXOS protocol.

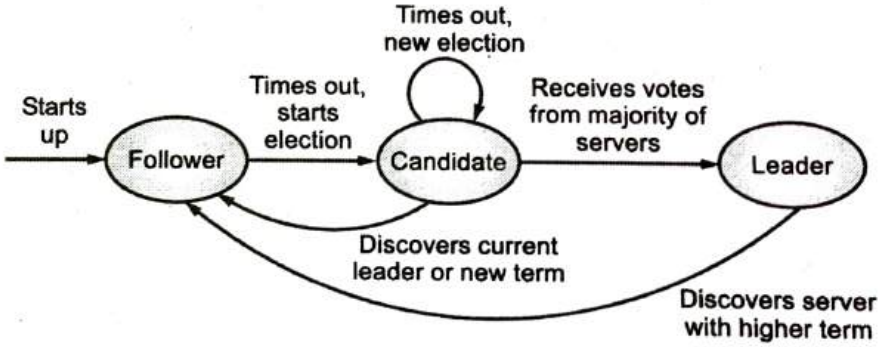
- **RAFT**

GQ. Elaborate on RAFT consensus mechanism.

- It is designed as an alternative to PAXOS.
- A generic way to distribute a state machine amongst a set of servers and ensure that every server agrees upon same series of state transitions.

- **Basic Idea**
  - Nodes collectively select a leader. Once the leader is present in the system,the others become followers.
  - The leader is responsible for state transition log replication across the followers.

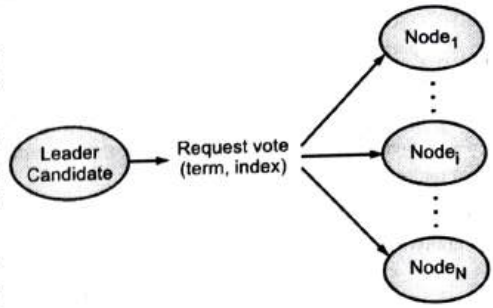
- Whenever a system starts up, there exists a set of follower nodes. Some of the follower nodes simply look whether there is a leader or not. If time out happens, it means that there is no leader in the system and the election process begins.



(1C33)**Fig. 5.6.17**

- In the election process, some of the candidates are chosen and we receive votes from majority of the servers indicating that who is going to be the leader. Now, amongst the candidates who wins for the majority of the votes, that individual becomes the leader, and this is how a leader is chosen.
- The chosen leader finds out what should be the proposed value and then the followers can either vote for the proposal which is coming from the leader or they can go against the leader. The RAFT consensus algorithm is explained with the help of a transaction log, which is the concept from a replicated database.
- For instance, there are multiple replicated servers and a consensus needs to be built amongst multiple replicated servers. So, whenever certain transactions are coming up from the clients, then the multiple replicated servers should take a collective decision about consensus and based on that it is decided whether to commit a transaction or not.

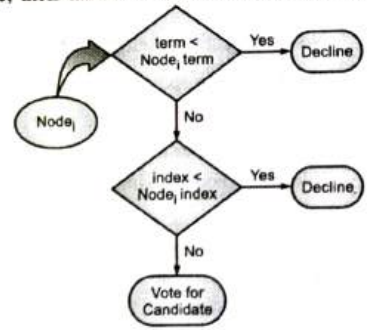
- **Electing the Leader : *Voting Request***
  - The first part of RAFT is to elect a leader. The question that arises is how will a leader be elected. In order to elect a leader, an individual needs to be a leader candidate.
  - Once a leader candidate is identified, then the leader candidate requests for a vote. The vote consists of two parameters (i.e., **term** and **index**).
  - Just like PAXOS, RAFT too runs in rounds. In every round, a decision needs to be taken. Term denotes in which particular round a leader is. *Term is computed as the last calculated number known to the candidate + 1.*
  - For instance, the earlier round was 20 when the term was 20. Now a new vote is needed to elect a new leader, and so, the new term is made as 21. This is now the terms are decided.



(1C34)**Fig. 5.6.18**

  - The index parameter specifies about the committed transaction which is available to the candidate. In other words, the index parameter says up to a certain number of transactions have been committed till now.The *Request Vote* message is passed to all the nodes who are there in the network.

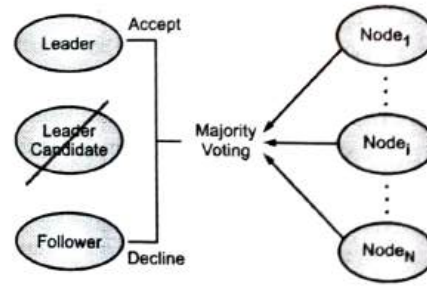
- **Electing the Leader : Follower Node's Decision Making**
  - The node receives a message. Once the node receives a message, their task is to elect a leader. So, this is the mechanism to elect a leader in RAFT consensus.
  - $Node_i$ receives a term value from one of the candidates and it looks into its own term. It first finds out whether its proposed term is less than its own term. If the proposed term is less than its own term, then the node declines the message to be a leader.
  - If the term is either equal to the current term (this could be possible because a leader might fail in the current term and it wants to choose another leader in the current term/round and so a new round would be initiated) or greater than the current term, then the index parameter is considered.
  - If the proposed index is less than the current $Node_i$'s index, then again decline the nodes message to be a leader else make a vote for the candidate. This is how a leader is selected in the RAFT consensus algorithm.



- Each node compares received term and index with corresponding current known values.

(1C35)**Fig. 5.6.19**

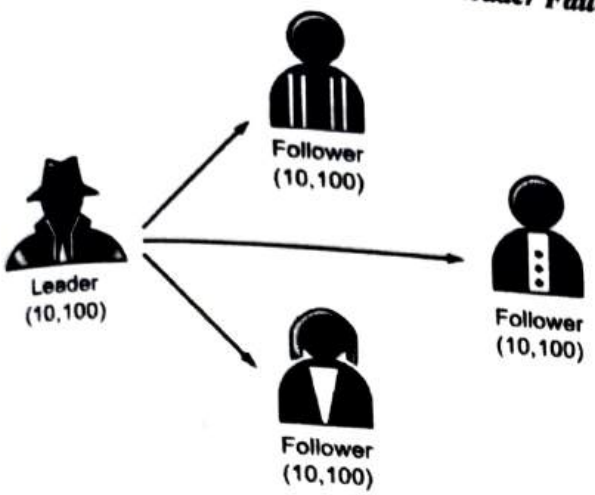
- **Electing the Leader : *Majority Voting***
  - The leader is getting a vote. So, every node sends the vote, and the concept of majority voting is used for leader election and then the log entry is committed.
  - If a certain leader candidate receives majority of the vote from the nodes, then that particular candidate becomes the leader and others become the follower of the node.



- Use of **majority voting**
  - **Leader selection**
  - **Commit the log entry**

(1C36)**Fig. 5.6.20**

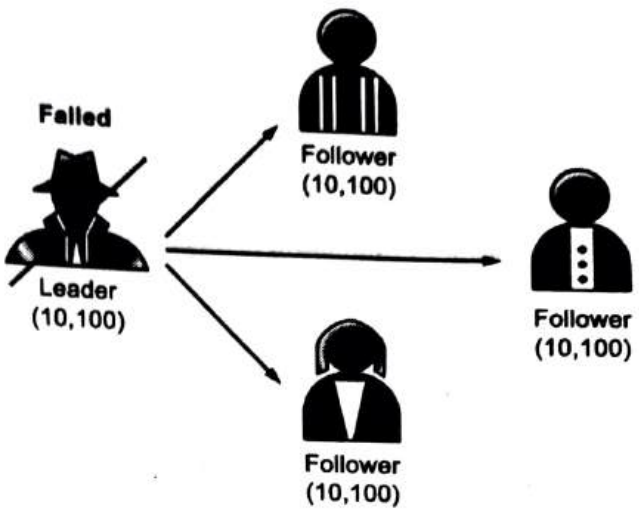
- **Multiple Leader Candidates : *Current Leader Failure***



- A leader with three followers.
- term: 10
- commit index: 100

(1C37)**Fig. 5.6.21**

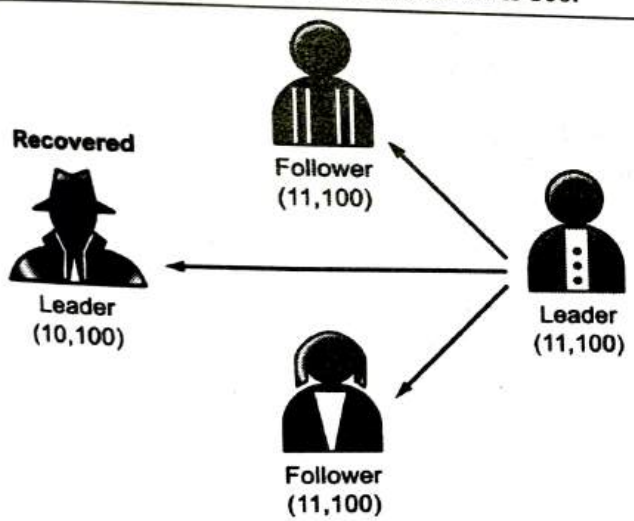
- When there are multiple leader candidates at the failure of the current leader. From the Fig. 5.6.21 it can be seen that a leader has three followers. The current term is 10 and the commit index is 100.



- The leader node failed.

(1C38)**Fig. 5.6.22**

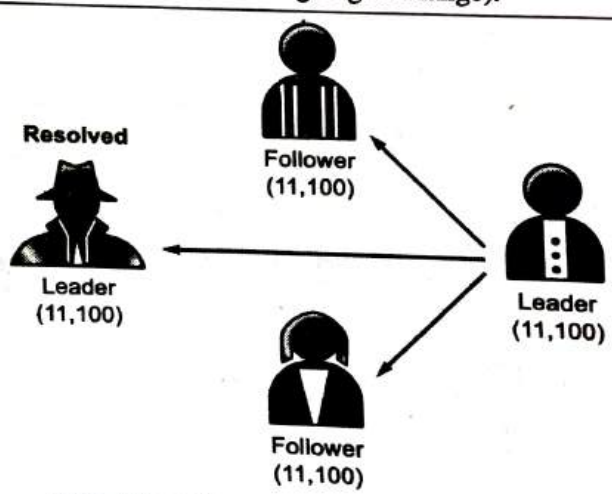
- Now, the leader node has failed. Once the leader node has failed, a new leader needs to be elected with term 11 (i.e., we need to move from round 10 to round 11 because the leader is going to change).



- New leader elected with term 11.
- Old leader recovered.

(1C39)**Fig. 5.6.23**

- At this particular time, there can be an old leader that gets recovered.



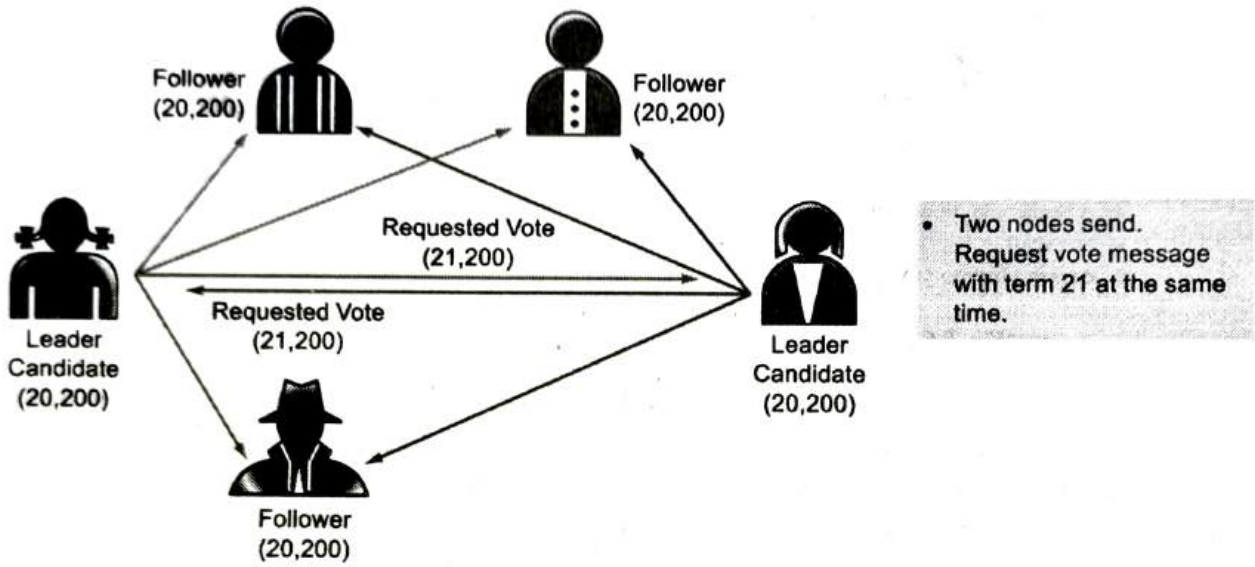
- Old leader received heartbeat message from new leader with greater term.
- Old leader drops to follower state.

(1C40)**Fig. 5.6.24**

- So, once the old leader receives this message. The old leader finds out that he was a leader at round 10 and now there is a new leader in the system, which is at round 11.

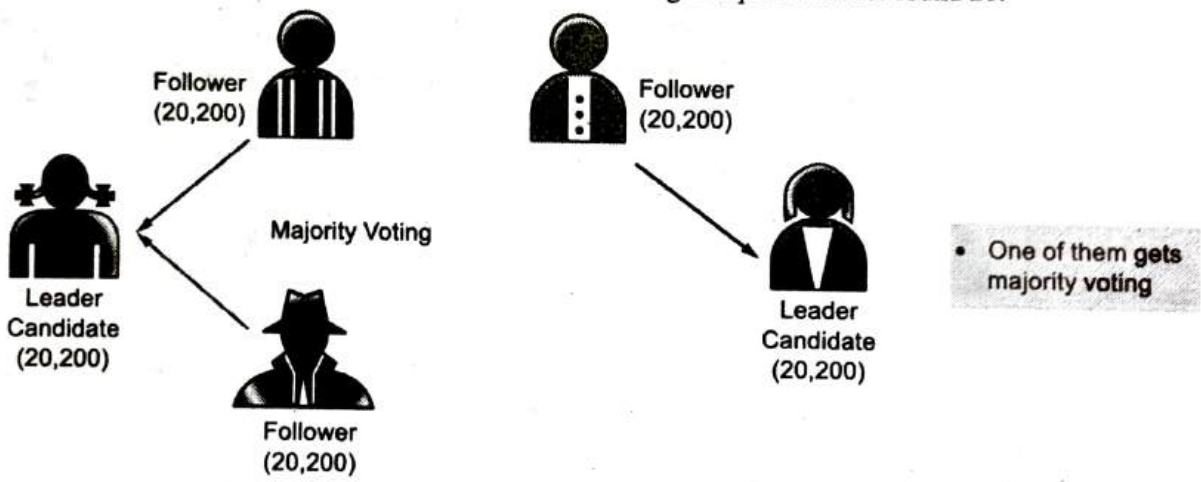
- From the Fig. 5.6.23, we can see that a new leader with (11, 100) is selected among the three followers and the new leader sends the message that he is the leader in round 11 and all of us are in commit index 100. Now, this message reaches the old leader (i.e., (10, 100)).
- So, the old leader falls down from a leader to a follower. This is how we can have a new leader whenever an old leader fails by utilizing the term parameter.

- **Multiple Leader Candidates: *Simultaneous Request Vote***



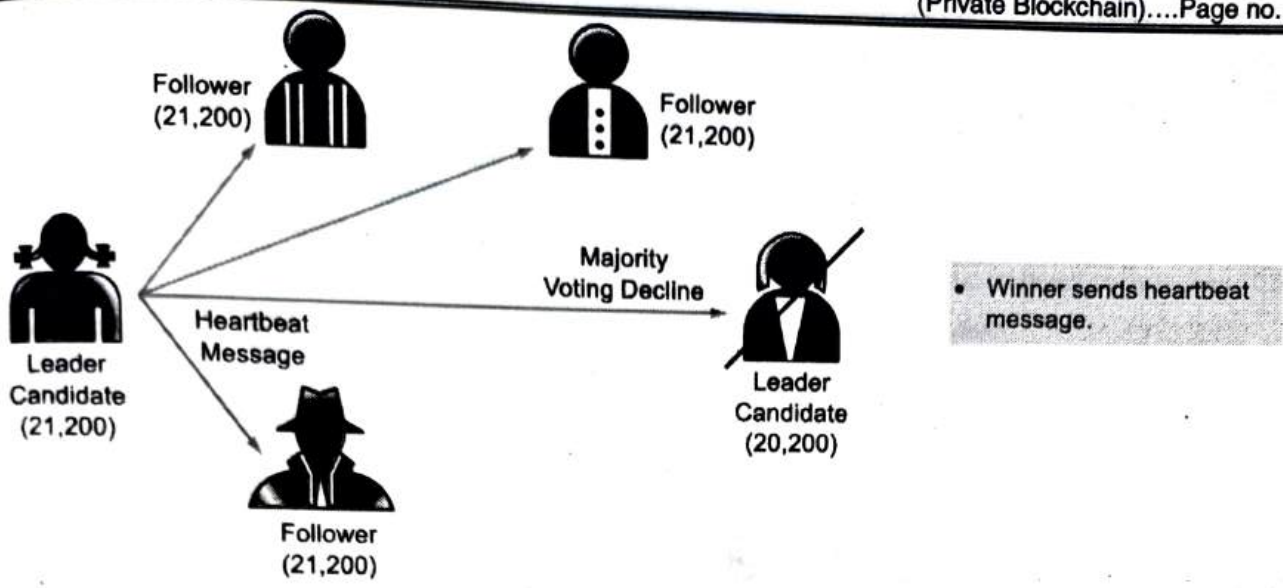
(1C41)**Fig. 5.6.25**

When two nodes send the request vote message with round 21 at the same time. In other words, there are two leader candidates at round 20 and both of them are sending a request vote for round 21.



(1C42)**Fig. 5.6.26**

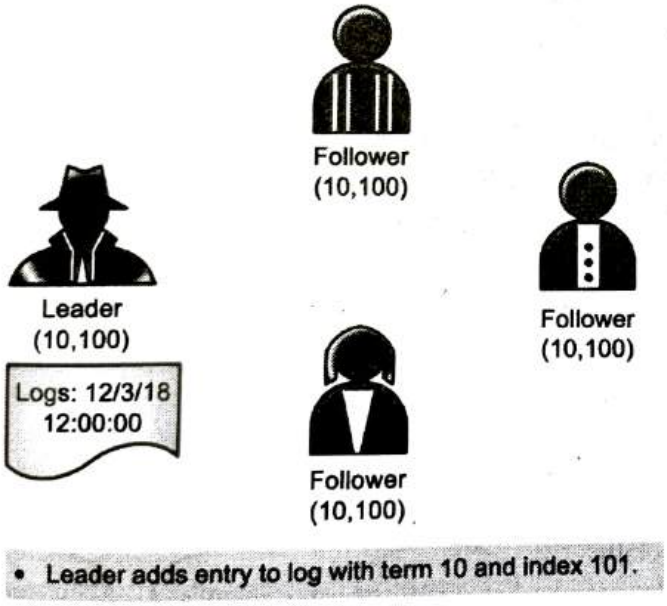
- In that case, they look for the majority of the voting.



(1C43)Fig. 5.6.27

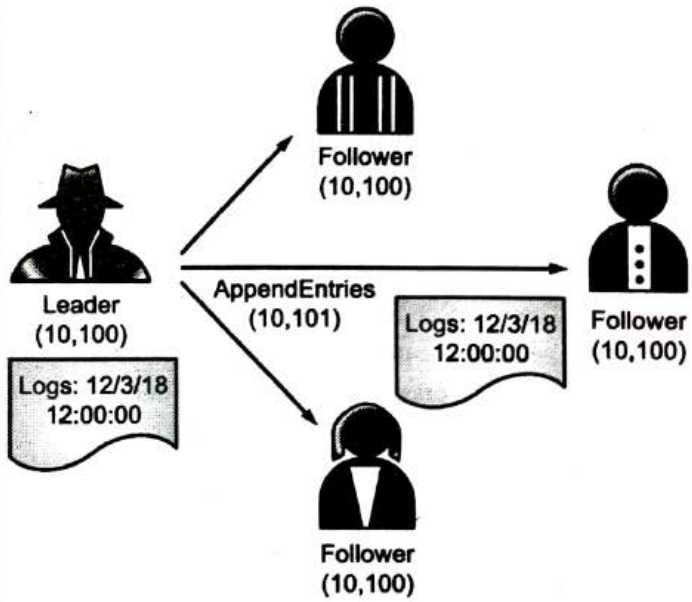
- The leader candidate that receives majority of the votes sends a special message (i.e., the heartbeat message) to others. So, the other leader candidate looks at the heartbeat message from the winner and that leader candidate fall back to a follower from the leader and round 21 begins.

- **Committing Entry Log**
  - Once the leader election is done, committing an index log or a transaction with a certain index in an index log is necessary because it ensure that a particular transaction has been committed as agreed upon by all the followers.
  - So, here once the leader election is done, the leader has the task to propose for a new transaction.



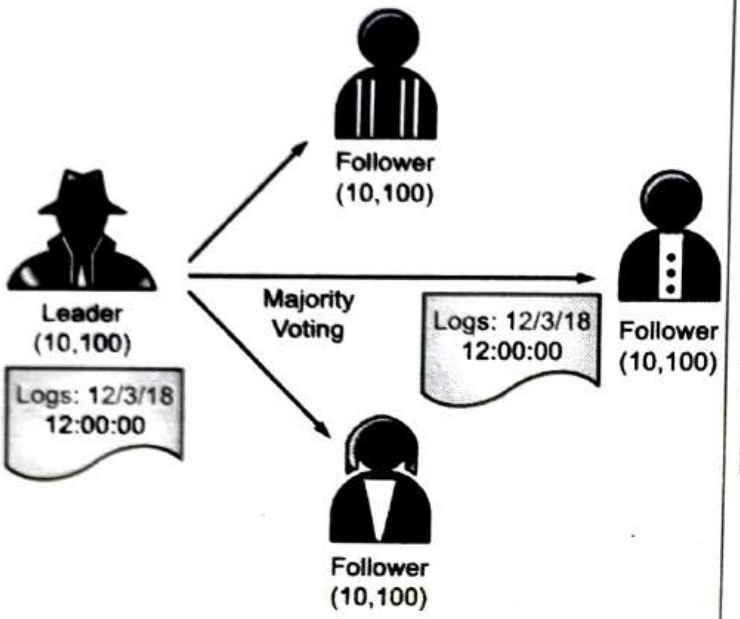
(1C44)Fig. 5.6.28

  - From the Fig. 5.6.29, we can see that everyone is at round 10 and the last committed transaction index was index 100.
  - Now the leader wants to propose a new transaction and so he/she adds an entry log with term 10 (i.e., the current term where everyone is), and the new transaction index as 101.



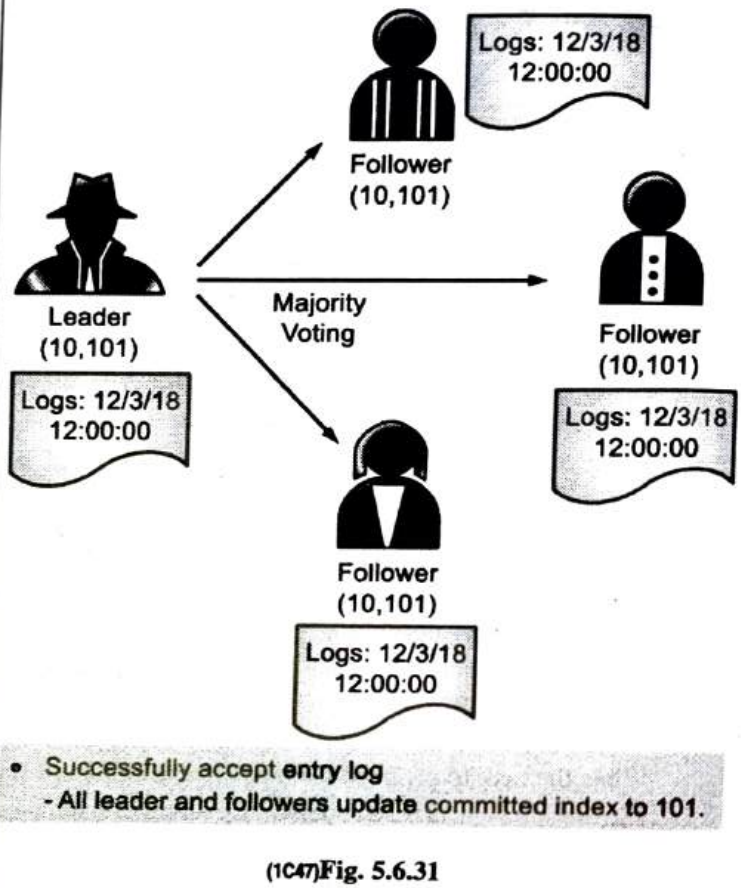
(1C45)Fig. 5.6.29

- Next, the leader sends a message called *AppendEntries* to all the followers. The *AppendEntries* contains the proposal, which is coming from the leader.
- In that proposal, the leader mentions that we are at round 10 and the leader is proposing for transaction 101 and whether it can be committed or not.



(1C46)**Fig. 5.6.30**

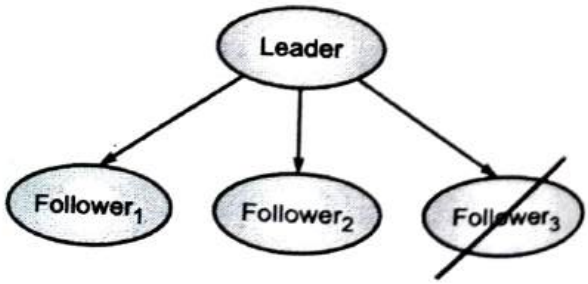
- Once the transaction log is broadcasted to all the followers, the followers collectively vote for the transaction or against the transaction.
- If the leader receives the votes for the transaction 101 and majority of them are fine with committing this log then the leader decides whether to accept or reject that proposed log 101.



(1C47)**Fig. 5.6.31**

- Once this is done, then log 101 gets accepted by everyone. It is placed into the entry log of all the individual followers and then in the current round log 101 gets entered and the leader sends an accept message based on the majority voting to all the individual followers. So the followers update the committed index to 101.

- **Handling Failure**
  - There can be a failure in case of RAFT consensus, and as mentioned earlier that PAXOS and RAFT are good for handling crash and network/partition fault, so it may happen that a follower has crashed.
  - A system can tolerate up to $\frac{N}{2} - 1$ nodes. It does not affect the system as the system is reliant on majority of the voting. Till the time there are majority of the followers and are non-faulty and they can send a vote and the leader can take the decision whether to accept or reject a transaction.



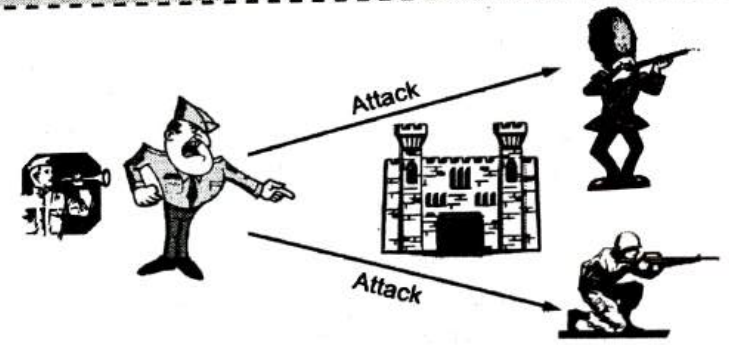
(1C48)**Fig. 5.6.32**

- *If the entire RAFT consensus protocol is viewed from the perspective of PAXOS and if a comparative analysis is made between them, it is observed that the idea of PAXOS was difficult to prove because individual nodes are proposing for certain values and the acceptors need to accept those values and because there are no leaders in the system, we need to wait for certain amount of time to see whether someone is proposing a value and if none of them are proposing a value, then someone needs to propose a value.*
- *So, when you are proposing a value, some of the nodes might accept a proposal while others would reject the proposal. If majority of the voting is received then it is evident that your proposal has been accepted and then the accept message is sent to all the nodes.*
- *Because there can be multiple proposers from the system, it becomes difficult to theoretically prove that how repeated execution of PAXOS (i.e., multi PAXOS) can be done. As PAXOS is defined as a set of protocols rather than a single protocol. So. That set of protocols collectively decide whether a system can go for consensus.*
- *The concept of leader was missing in PAXOS. In order to elect a leader, something similar to a consensus algorithm is needed where everyone should agree on the same leader. However, the entire system becomes streamlined once a leader has been elected.*
- *So, this is how RAFT has improved the concept from PAXOS. Rather than going for multi-PAXOS, the mechanism of leader election is used. So, in such a mechanism, we will apply certain consensus algorithm based on the majority voting. There can also be multiple leader candidates similar to PAXOS proposers.*
- *In RAFT, PAXOS can be applied to elect a leader, but once a leader has been elected based on the majority voting, then it can be ensured that the next series of transactions can be committed just by the leader.*
- *In comparison to multi-PAXOS and RAFT, for every individual transaction in multi-PAXOS, PAXOS algorithm needs to be executed repeatedly to come to a consensus, but in case of RAFT, we need to execute the complicated PAXOS algorithm just once to elect a leader. This is how RAFT consensus algorithm has improved the concept of leader election and has made consensus easy to understand and prove theoretically.*

5.7 BYZANTINE FAULTS (INCLUDING CRASH OR NETWORK FAILURES)

GQ. Describe Byzantine fault tolerant algorithm.

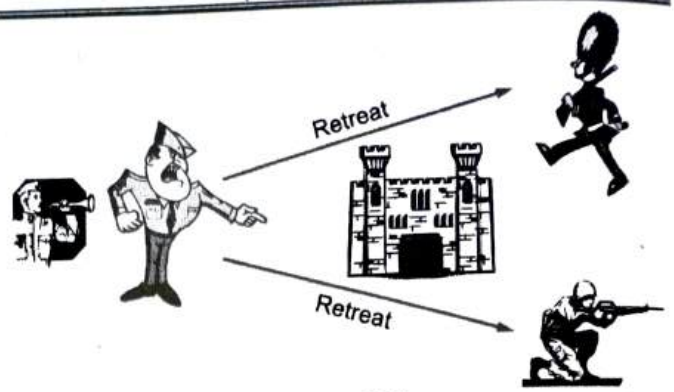
- **Byzantine fault tolerant (BFT)**
  - Both PAXOS and RAFT can tolerate up to $\frac{N}{2} - 1$ number of crash faults.
  - But, what if the nodes behave maliciously. The concept of byzantine faults came from an interesting problem known as the byzantine generals problem.



(1C49)**Fig. 5.7.1**

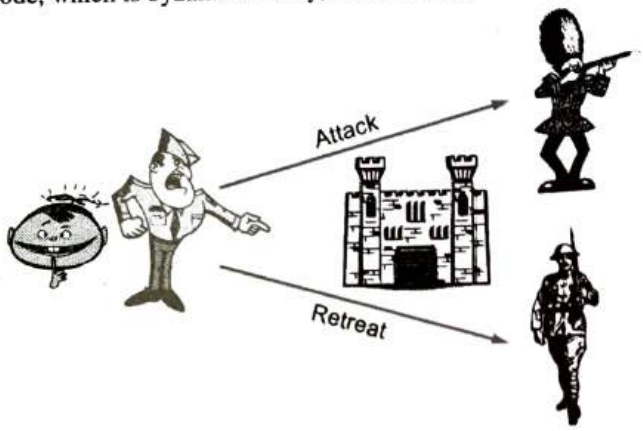
- In this problem, there exists certain forts that a group of army wants to attack. Now, based on the scenario, the general can either make an order to all the armies to perform an attack from different sides of the fort or the general can ask the army to retreat.

- If the general is good, then he will always say the same thing to both the lieutenants/soldiers to either go for an attack or retreat.
- But, unfortunately, the general is not so good (i.e., the general can behave maliciously). The general can say attack to one soldier and say retreat to another soldier. There could be a possibility that the general has taken some bribe from the general of the fort and started behaving maliciously.



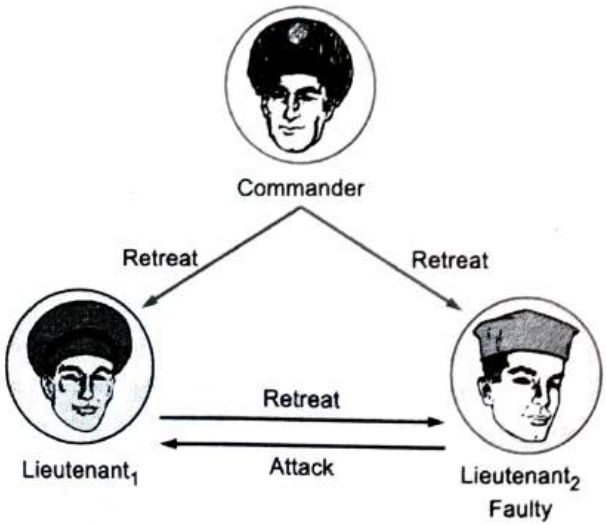
(1C50)**Fig. 5.7.2**

- Such type of a problem is called the byzantine generals problem where a particular node can behave maliciously.Faulty nodes will not send any vote, and from non-faulty nodes, a correct vote will always be received. Such assumption doesn't hold true all the time. It may happen that a faulty node, which is byzantine faulty, is selectively sending votes to some of the nodes.
- Some of the followers receive information that they are in favor of a particular transaction, whereas for other followers, they receive information that they are against the transaction. So, this might happen in the RAFT algorithm *(try to map it with the byzantine generals problem)*, and assume that instead of having a crash fault by the leader, the leader behaves in a byzantine way.



(1C51)**Fig. 5.7.3**

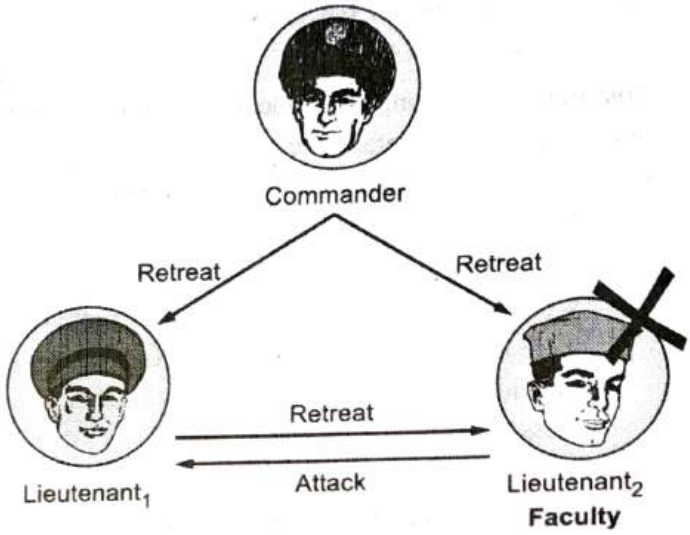
- If the leader behaves in the byzantine way, then he/she proposes transactions to a selective number of nodes/followers amongst all the followers. If the leader behaves in a byzantine way, then the RAFT consensus will not be able to recover for the system. Therefore, there is a need to have a strict class of a consensus algorithm that can tolerate the byzantine fault and such type of a consensus algorithm is known as the byzantine fault tolerant consensus algorithm.
- **Byzantine generals problem under multiple nodes**
  - We consider three byzantine generals problem where there are three generals (i.e., one commander and two lieutenants).
  - So, the commander sends messages to the lieutenants, and the lieutenants can share the messages amongst themselves and try to identify whether the commander is faulty or the lieutenant is faulty.
  - Now, let us consider three generals where we will try to design a solution for the byzantine fault tolerant system.



(1C52)**Fig. 5.7.4**

Case 1

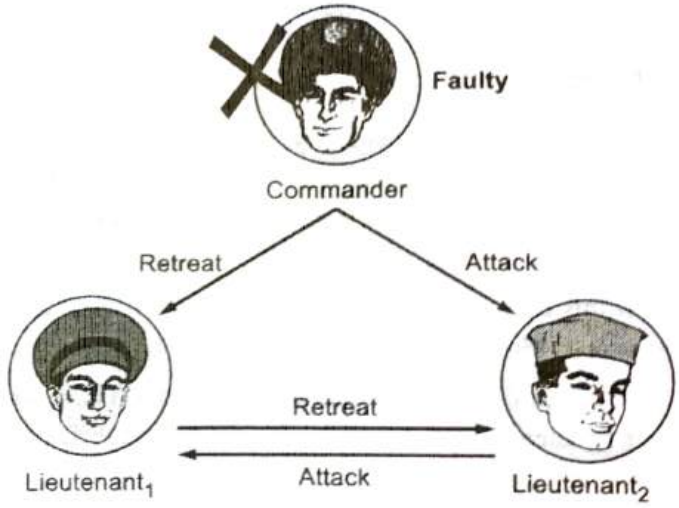
- Herein, we assume that the lieutenant is faulty. If the lieutenant is faulty, then he may send different messages from what he hears. The commander sends the correct message (i.e., retreat) to both the lieutenants.
- Note that lieutenant 2 is a faulty lieutenant who doesn't obey the message sent by the commander. In fact, lieutenant 2 sends an attack message to the other lieutenant.
- Lieutenant 1 sends the correct message (i.e., retreat) that he receives from the commander and send the same message to lieutenant 2. Under such a scenario where the commander is correct and one of the lieutenants is faulty, let us check whether consensus can be achieved.
- Lieutenant 1 receives different messages (i.e., he receives retreat message from the commander and attack message from lieutenant 2). In the general principle of a normal army scenario, lieutenants always obey the commander. With this condition, the lieutenant may obey the commander, and if the commander is non-faculty, then the entire system works correctly even if lieutenant 2 is faulty and sends the wrong message.



(1C53)Fig. 5.7.5

Case 2

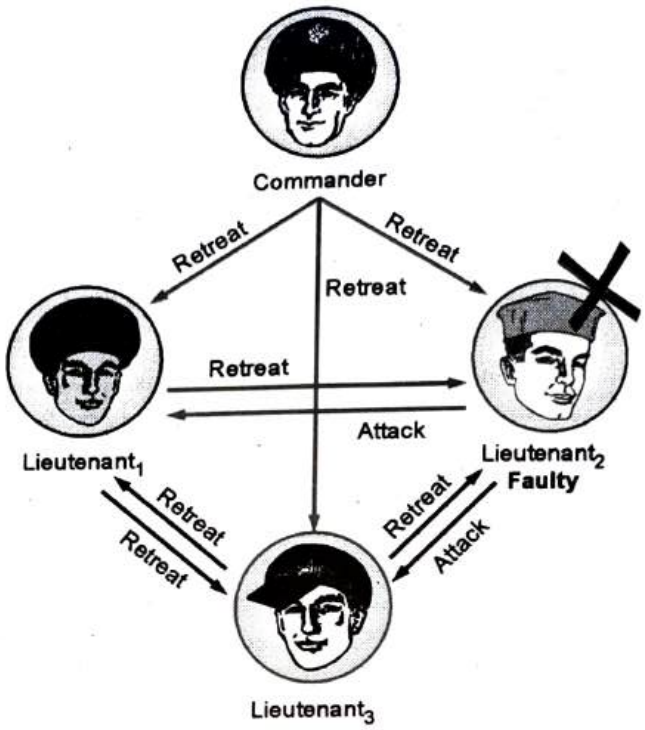
- Now, we shall consider that the commander is faulty and both the lieutenants are correct, which is a non-trivial case. Here, the commander sends the retreat message to one of the lieutenants (i.e., Lieutenant 1) and it sends an attack message to the other lieutenant (i.e., Lieutenant 2).
- As per the general principle of a normal army scenario, Lieutenant 2 will send an attack message to Lieutenant 1 since he received an attack message from the attacker and Lieutenant 1 will send a retreat message to Lieutenant 2 because he received a retreat message from the commander.



(1C54)Fig. 5.7.6

- Thus, by the message passing principle, the entire system will not be able to work because Lieutenant 1 will have different messages from different generals (i.e., he will receive a retreat message from the commander and attack message from Lieutenant 2).
- Because Lieutenant 1 has different messages, with the integrity condition, if he performs a retreat, then he will follow the normal army scenario. On the other hand, for Lieutenant 2, he received an attack message from the commander, and if he performs an attack, then the entire army will get defeated.

- As a result, it is observed that if the instruction of the commander is obeyed and there is no way to identify whether to go with the majority voting or the instruction that has been sent by the commander, then the entire system is in a dilemmaas to which instruction needs to be followed.
- Even under the byzantine generals problem, the problem of byzantine failure can be solved with the help of principle of majority voting, which has been mentioned earlier in the case of PAXOS or RAFT.
- Note that the by using the principle of majority voting for three byzantine generals (i.e., one commander and two lieutenants), the problem still remains unsolvable as equal voting is obtained for an attack and a retreat by a lieutenant and the lieutenant cannot decide what to do.
    - We consider four byzantine generals problem where there are three lieutenants and one commander. Every individual lieutenant talk with each other using the message passing principle.
    - The commander sends a retreat message to all the lieutenants and the lieutenants share the message to each other that has been sent by the commander. Here, we assume that Lieutenant 2 is faulty. Amongst the three lieutenants, if one lieutenant is faulty, then let us see the different cases.

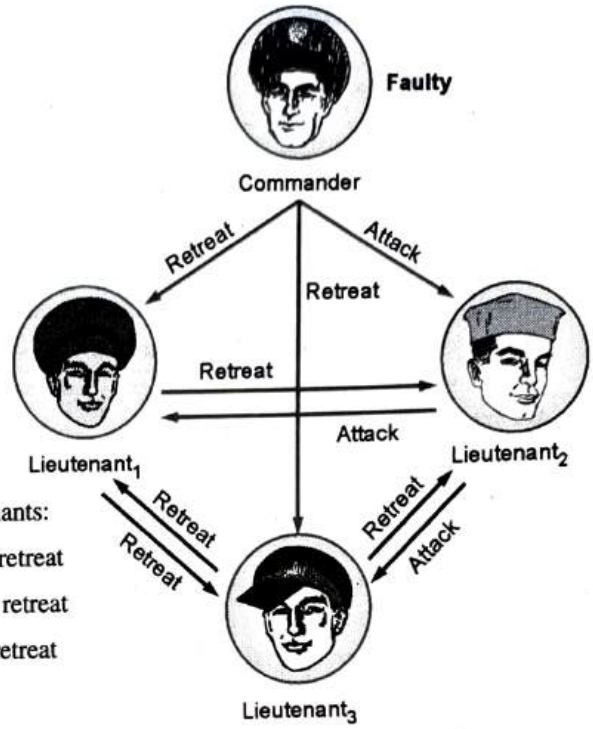


(1C55)**Fig. 5.7.7**

☞ Case 1

- When the lieutenant is faulty, he sends messages that differ from the ones sent by the commander (i.e., the commander sends a retreat message to all the lieutenants). Lieutenants 1 and 3 correctly echo the message to other lieutenants. However, Lieutenant 2 incorrectly echoes the message to other lieutenants (i.e., Lieutenant 2 sends an attack message to Lieutenants 1 and 3).
- In this case, if majority voting principle is considered, then Lieutenant 1 decides on majority(retreat,attack, retreat) = retreat. Moreover, Lieutenant 3 decides on majority(retreat,retreat,attack) = retreat
- Note that the behavior of faulty lieutenant (i.e., Lieutenant 2) is not considered. So, here it has been observed that even if Lieutenant 2 is a byzantine node, then too Lieutenants 1 and 3 are able to correctly decode the message from the majority of the voting.
- However, if two lieutenants are faulty (i.e., Lieutenant 1 and 2), then Lieutenant 1 also behaves in the same malicious manner like Lieutenant 2 (i.e., he sends an attack message instead of retreat). Because of this, Lieutenant 3 receives one retreat message from the commander and two attack messages from faulty Lieutenants 1 and 2, and so, Lieutenant 3 will not be able to take a decision based on the majority voting.
- So, here it is observed that out of the three lieutenants, if one lieutenant is faulty, then the correct message can be decoded. Let us consider another case where the commander is faulty (i.e., when the commander is behaving maliciously),

- The commander sends a retreat message to Lieutenants 1 and 2, but an attack message to Lieutenant 3.
- In such a scenario, all the lieutenants are correct, and since Lieutenant 1 has received retreat message from the commander, he correctly echoes the message to other lieutenants.
- On the other hand, Lieutenant 2 is a correct/genuine lieutenant, but he has received an attack message from the commander, and so, he sends an attack message to Lieutenants 1 and 3.Lieutenant 3 receives a retreat message from the commander and so he echoes the same message to the other lieutenants.
- Considering the principle of majority voting for all three lieutenants:
  - Lieutenant 1 decides on majority(retreat,attack, retreat) = retreat
  - Lieutenant 2 decides on majority(retreat,retreat, retreat) = retreat
  - Lieutenant 3 decides on majority(retreat,retreat,attack) = retreat



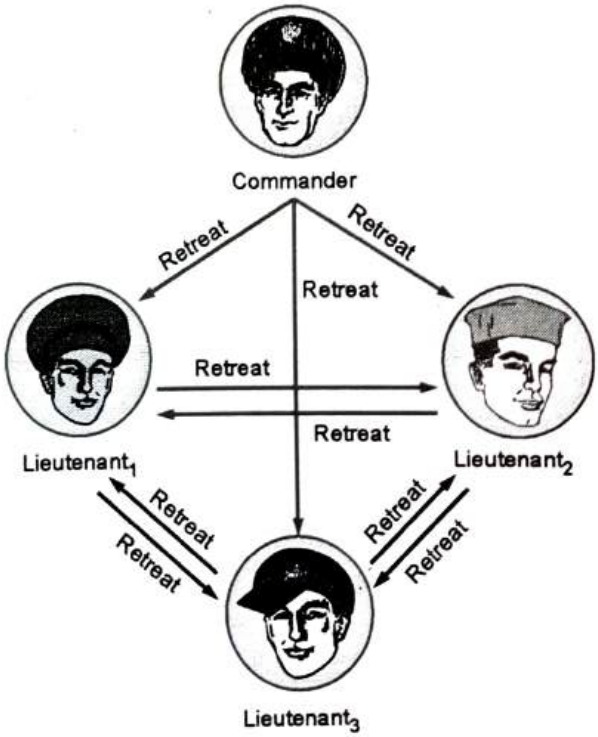
(1C56)**Fig. 5.7.8**

- So, here it has been observed that if there are three lieutenants who are correct/genuine with one commander who is behaving maliciously, then too correct results are obtained by the principle of majority voting. If there exits f faulty nodes and if a lieutenant is faulty, then we can say that there are $2f + 1$ lieutenants in the system, which means that if there are $2f + 1$ lieutenants and 1 commander, then the principle of majority voting can be applied correctly and the byzantine nodes in the system can be found out easily.
- Also, if the commander sends the attack message to more than one lieutenants, then in that case the majority decision taken by the commander with get reflected in the entire system (i.e., attack instead of retreat).

☞ Byzantine Generals Model

GQ. Explain Byzantine Generals Model.

- In this model, we assume that there are N number of processes out of which at most f number of processes can be faulty. This means that there should be $2f + 1$ lieutenants in the system.
- The receiver always knows the identity of the sender, which is a closed model. In the context of blockchain, this model helps us to design an algorithm for a permissioned blockchain environment.
- The system is fully connected. The medium of communication is reliable.
- The system is synchronous, which means that every node will be able to receive all messages within a pre-defined timeout duration/interval.

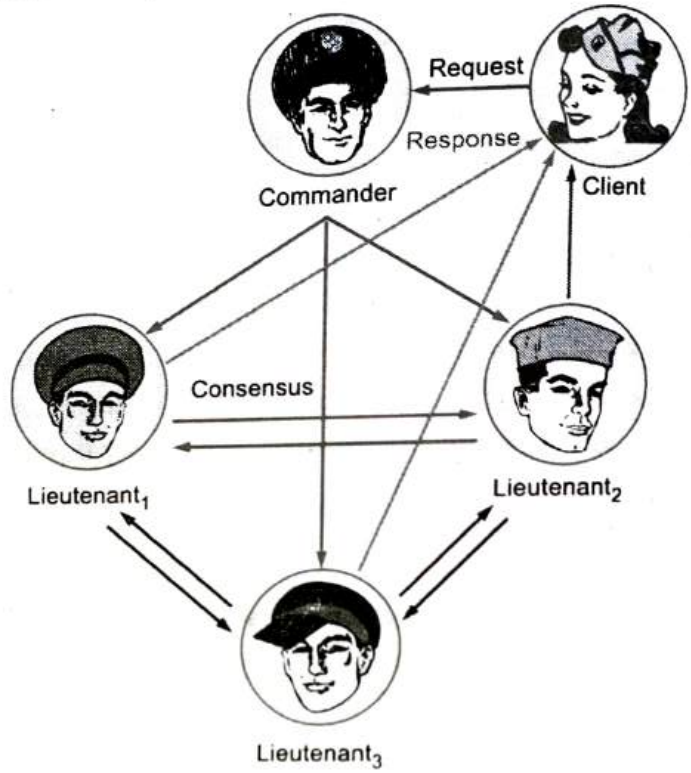


(1C57) **Fig. 5.7.9**

☞ Practical byzantine fault tolerant (PBFT)

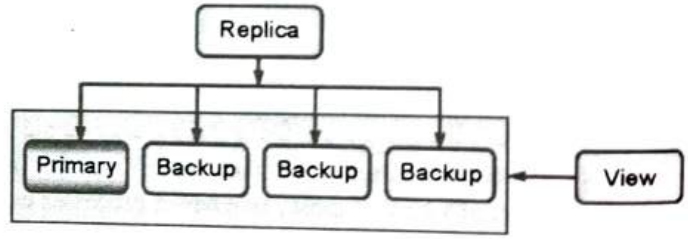
GQ. What is a backup in PFBT ?

- This algorithm is termed as practical because it ensures safety over an asynchronous network. It can handle byzantine failure and has low overhead. PBFT is widely used in permissioned platforms like Tendermint, IBM's Openchain, ErisDB, and Hyperledger.
- In PBFT, there is a client who submits the request to the commander *(clients are those who are submitting the transaction)*. The commanders are certain nodes in the system who are responsible for ensuring consensus in the system.
- The commander in cooperation with multiple nodes (i.e., lieutenants) come to a consensus (i.e., blue lines). With the help of multiple lieutenants in the system, the commander comes to a consensus, and once a system comes to a consensus, it sends the response back to the client (*i.e., whether the system has committed based on the consensus algorithm or not*).
- Since PBFT is an asynchronous distributed system, there can be a delay in message transmission, and there is a possibility that an out-of-order message can be received.
- The system can have a byzantine failure like an arbitrary node behavior. So, some nodes can vote for a transaction and others could vote against a transaction.



(1C58)**Fig. 5.7.10**

- PBFT also supports privacy over the system. It ensures that the messages are tamperproof by applying hashing techniques and applies authentication techniques via digital signature mechanism so that none of the message is transferred from individual nodes in the system.
- In PBFT, a state machine is replicated across different nodes. There are $3f + 1$ replicas where f is the number of faulty replicas. The replicas move through a succession of configurations, which are called*views*.
- One replica in a view is *primary* and others are *backups*.



(1C59)**Fig. 5.7.11**

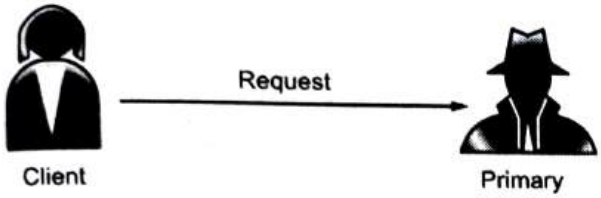
- Note that views are changed when a primary is detected as faulty. Every view is identified by a unique integer number v.
- Only messages from the current views are accepted(i.e., if there is a view change, then the messages that have been broadcasted in the previous view and because of the asynchronous nature of the system there is a possibility that a delayed or an out-of-order message will be received or a previous view message could be received; thus, the messages from the previous view are discarded and only the messages from the current views are accepted).

A simplified view of PBFT algorithm

GQ. Explain PBFT algorithm.

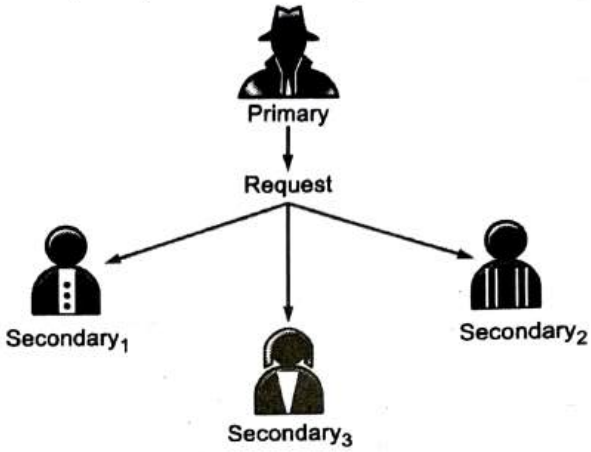
The algorithm was proposed by Castro, Miguel, and Barbara Liskov in 1999. The idea of the algorithm is as follows.

- A client sends a request to invoke a service operation to the primary.



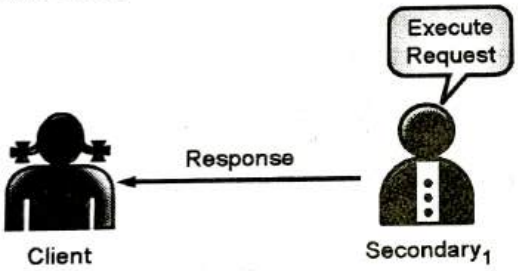
(1C60)Fig. 5.7.12

- The primary then sends the request to all the backups.



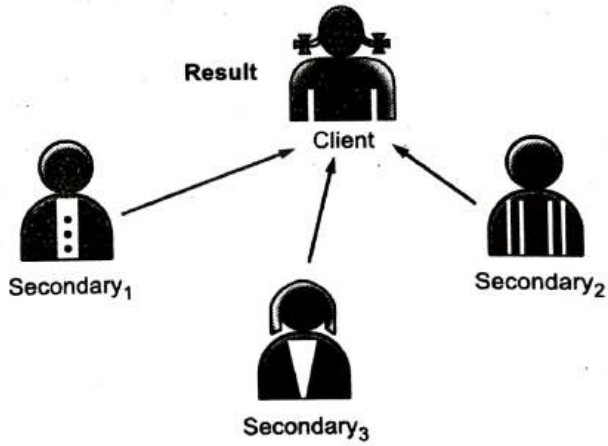
(1C61)Fig. 5.7.13

- Next, the backups execute the request and send a reply to the client.



(1C62)Fig. 5.7.14

- After executing the request, either the client request is successful or the client request is failure. From the perspective of blockchain, a transaction is a valid transaction if all the replicas decide that the current transaction is a correct transaction, then a success/commit message is sent the client else a failure message is sent.



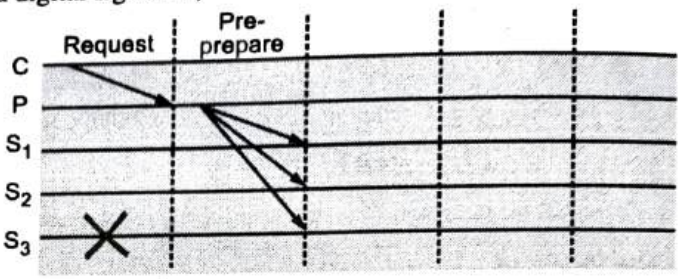
(1C63)Fig. 5.7.15

- Next, the client waits for $f + 1$ replies from different backups with the same result where f represents the maximum number of faulty replicas that can be tolerated.
- Once the client receives $f + 1$ replies from different backups with the same result, this means that the client has received majority of the correct voting. The entire consensus algorithm in PBFT works in three different phases:

GQ. What is the difference between pre-prepare phase, prepare phase, and commit phase in PBFT ?

1. **Pre-prepare phase :** Initially, the client sends a request to the primary. Next, the primary assigns a sequence number *n* to the request and multicast a message <<pre-prepare, v,n,d > σ_p,m> to all the backups.
   - *v* is the current view number
   - *n* is the sequence number that the primary has included in the message
   - *d* is the message digest
   - σ_ p is the private key of the primary (i.e., it works as a digital signature)
   - *m* is the message to transmit.

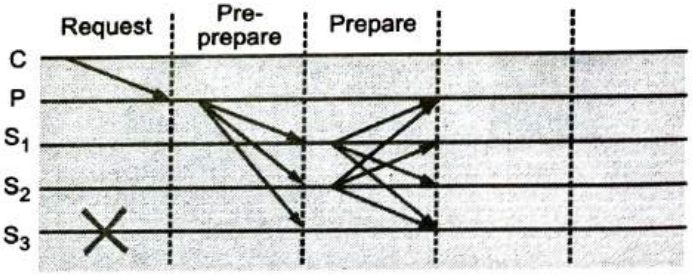
   So, when the client sends the request, the pre-prepare message is broadcasted to all the secondary (i.e., S_1, S_2, and S_3). We assume the S_3 is faulty. Thus, there exits $3f + 1$ replicas, and in this case $f = 1$, which is S_3. Therefore, we have a total of 4 replicas (i.e., 1 primary (P) and three backups (S_1, S_2, and S_3)).



(1C64)**Fig. 5.7.16**

   - The pre-prepare messages are used as a proof that the request that was assigned a sequence number *n* is the view *v*.
   - Note that a backup accepts a pre-prepare message if
     - The signature is correct and d is the digest for m.
     - The backup is in view v.
     - It has not received a different pre-prepare message with a sequence n and view v with a different digest.
     - The sequence number is within a threshold.

2. **Prepare phase :** If the backup accepts the pre-prepare message, then it enters the prepare phase by multicasting a message <prepare, v, n,d i > σ_i to all other replicas.-
   - *i* is the identity of the backup, which is broadcasting the prepare message.
   - σ_i is the private key of the backup.
   - A replica (i.e., both primary and backups) accepts a prepare message if
     - Signatures are correct
     - View number equals to the current view
     - Sequence number is within a threshold
   - Note that the pre-prepare and the prepare phases ensure that non-faulty replicas guarantee on a total order for the requests within a view.

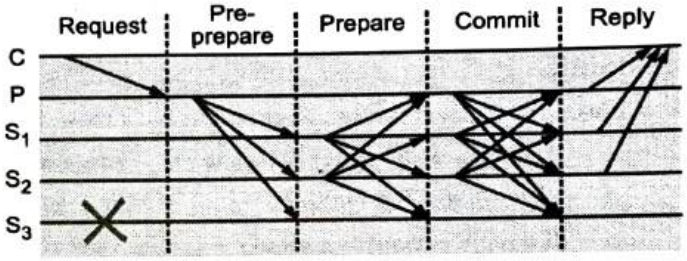


(1C65)**Fig. 5.7.17**

3. **Commit phase :** After the pre-prepare and the prepare phases, once every replica receives a prepare message, then it commits a message if
   - $2f$ prepare messages from different backup matches with the corresponding pre-prepare.
   - There are a total of $2f + 1$ votes (one from primary that already exists) from non-faulty replicas.

- Why is there a need to have $3f + 1$ replicas to ensure safety in an asynchronous system when there are f faulty nodes?
- If there are $2f + 1$ replicas, then all votes are needed to decide the majority, which reduces to a synchronous system/environment. In case of an asynchronous system, votes from certain replicas may not be received due to delay or out-of-order messages.
- $f + 1$ votes do not ensure majority. There is a possibility that f votes might have been received from byzantine nodes and just one vote from a non-faulty node. If a vote is not received then, it means that

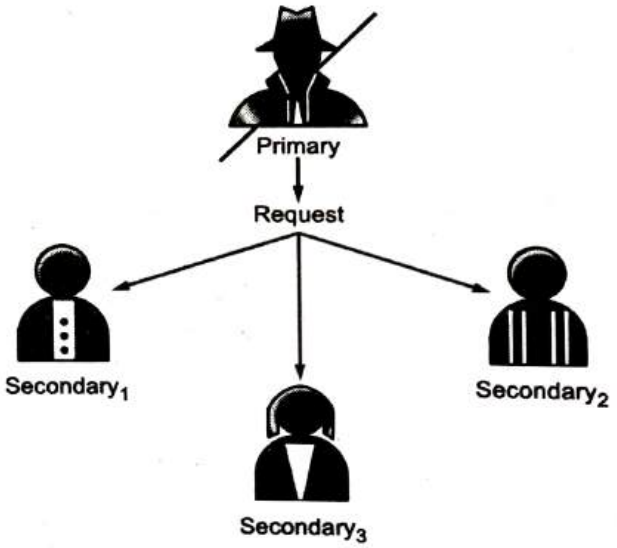
- The node is faulty and not forwarded a vote at all. The node is non-faulty, and it has forwarded a vote, but the vote got delayed.
- Majority can be decided once 2f + 1 votes have arrived, and even if f nodes are faulty, we know that f + 1 nodes are from correct nodes, and there is no need to care about the remaining f votes.
  - Finally, every node multicast a commit message <commit,v,n,d,i> σ_i to all the replicas including the primary.
  - A message is committed when a replica has sent a commit message itself and when it has received $2f + 1$ commits including its own.
  - After getting the final commit message, every individual node takes a decision and it sends the reply back to the client.



(1C66)**Fig. 5.7.18**

What if the primary is faulty ?

- Non-faulty replicas detect the fault, and they together start a view change operation.
- So, the replicas remove the primary from the system or they consider the replica that was designated as primary in view v will get changed and the other replica from the backup will be designated as primary.
- View change protocol ensures liveliness, but to ensure view change, the message should be received from all the nodes within a timeout duration (i.e., allow the system to make progress when primary fails).
- If the primary fails, then backups will not receive any message like *pre-prepare* and *commit* from the primary.



(1C67)**Fig. 5.7.19**

- Note that view changes are triggered by timeouts *(i.e., prevent backups from waiting indefinitely for requests to execute)*. Backup starts a timer when it receives a request, and the timer is not already running.
- The timer is stopped when the request is executed. It restarts when some new request comes/arrives. If the timer expires at view v, then backup starts a view change to move the system to view $v + 1$.
- On timer expiry, a backup stops accepting messages except
  - Checkpoint message
  - View change message
  - New view message

- When a primary has a timeout, the view change multicasts <view_change, v+ 1,n,C,P,i> σ_i to all the replicas.
  - n is the sequence number of the last stable checkpoint *s* known to *i*.
  - C is a set of $2f + 1$ valid checkpoint messages proving the correctness of s.
  - P is a set containing a set P_m for each request *m* that is prepared at i with a sequence number higher than n.
  - Note that each set of P_m contains a valid pre-prepare message and $2f$ matching.
  - The new view is initiated after receiving $2f$ view change messages.
  - The view change operation handles synchronization of checkpoints across replicas and all replicas are ready to start at the new view $v + 1$.

☞ Correctness

GQ. Why is there a need to have $3f + 1$ replicas to ensure safety in an asynchronous system when there are *f* faulty nodes?

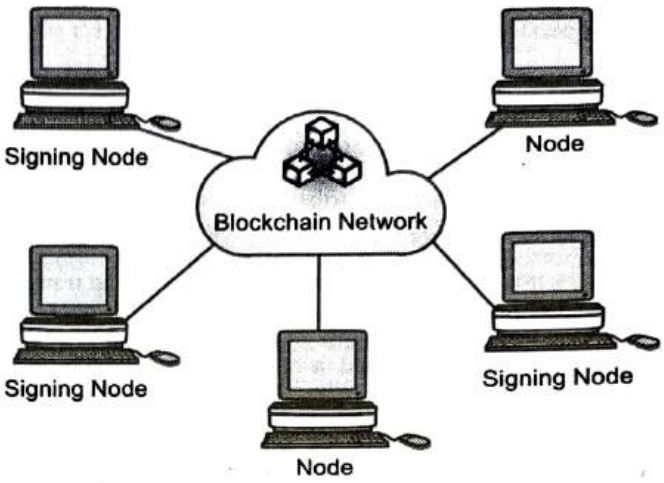
- ***Safety*** : The algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that commit locally.
- ***Liveness*** : To provide liveness, replicas must move to a new view if they are unable to execute a request.
  - A replica waits for $2f + 1$ view change messages and then starts a timer to initiate a new view *(avoid starting a view change too soon)*.
  - If a replica receives a set of $f + 1$ valid view change messages for views greater than its current view, then it sends view change message *(prevents starting the next view change too late)*. Faulty replicas are unable to impede progress by forcing frequent view change.

⏭ 5.8 CONSORTIUM BLOCKCHAIN

GQ. What is a consortium blockchain ?

- Private blockchains are permissioned and have a single authority, which control and manage activities over a network. Control of a single entity makes private blockchain more centralized and it provides a partial decentralized network with a set of rules and regulations.
- Consortium blockchain provides a partially decentralized platform, which attempts to solve the problem of autonomy in a private blockchain network by distributing power amongst a group instead of an individual.
- In consortium blockchain, multiple organizations or group of representatives together govern and make decisions for the best use of an entire network. With a consortium, without building any structures from scratch, new members can join an already established network and access shared data.
- At the same time, by effective collaboration, institutions can avoid redundant work, and by sharing responsibilities according to competencies and expertise, they can solve common problems together at reduced costs and time expenses.
- Consortium blockchain is managed and regulated by a group rather than an individual, where instead of consenting any random node connected to the Internet to indulge in the process of authentication of transactions (as in the case of public blockchain) or to providing only an individual to have full control (as in case of private blockchain), the predetermined selected group of nodes within a network are authorized to manages activities.
- These nodes will be involved in the consensus process, read and write data, and agree on who can log on the ledger. Let's understand the functionality of a consortium blockchain with an example. Consider there is a consortium blockchain with 50 institutions who have defined a rule that a transaction block or any decision within a network is considered to be true after confirmation of more than 40 institutions.

- In this case, if more than 40 institutions approve a transaction then consensusis reached within the consortium and only after that the transaction is considered as a valid transaction and vice versa.
- This way consortium blockchain permits faster transaction processing in a network without relying on the decision of individuals and also resolves the centralization issue of using a permissioned blockchain.
- Consortium blockchain also prevents wrong decisions of fraudulent activities within a network by an individual participant as modifications depend on the majority of participating institutions, which make the network more secure from being misused under the rule of a single entity.



(101)**Fig. 5.8.1 : Consortium blockchain network**

- Consortium blockchain is a permissioned type of blockchain where multiple organizations participate together to maintain a network. It operates in a protected environment which cannot be seen by outsiders.
- The closed group will offer a limited activity in a network with a certain group of people, and access to read the blockchain might be kept public or restricted to participants.
- Fig. 5.8.1 represents a sample consortium blockchain network with signing nodes that participate in the consensus process to add a block in a network and grant permission to other nodes to access the network. Under consortium blockchain, only trusted members are acquainted with the responsibility of executing a consensus protocol.
- Most widely used consortium blockchains are mainly permissioned because organizations avoid keeping confidential business information on a permissionless blockchain open to everyone; however, a few consortium blockchains are publicly accessible to interested individuals, and consortium blockchains can have a public interface where everyone can read data.
- Hyperledger, Ripple, Corda, Quorum, Libra, etc. are examples of a consortium blockchain. Enterprises and organizations can effectively use consortium blockchains by focusing on users who can read from a system, who can write into a system, and who can use a system.

5.9 KEY CHARACTERISTICS OF A CONSORTIUM BLOCKCHAIN

GQ. What is a consortium blockchain ?

- Unlike public networks, consortium blockchain is permissioned and governed by a group of institutions, i.e., the network is neither governed by an individual nor open to everyone.
- The activities over a network are supervised by a group, which makes it semi decentralized. Consortium blockchain or federation blockchain is a hybrid of private and public blockchain platforms where permission to read a block may be restricted based on the need of anorganization that is involved in consortium, and consensus in a network is achieved and controlled with the help of a pre-selected group of members.
- These groups of nodes managing the network can allow or restrict the use of the network; so, only nodes with a permission can use the blockchain. Consortium network doesn't allow anybody with the Internet connection to participate in the network; instead, it demands permission from network administrators.

- Each enterprise node in a network governing the blockchain (holds a copy of the transaction history) is accountable for its actions. For example, often consortium blockchains are shared amongst companies or financial entities in a cooperative format, and these are responsible for placing restrictions on a users' reading, writing, and auditing rights.
- All operations are verified by a set of special pre-approved nodes from the consortium and not by anyone with anInternet connection, like in bitcoin blockchain.
- If all entities managing the network agree, data and rules can be changed. Consortium blockchains are semi-decentralized, and only trusted nodes are acquainted with the responsibility of executing a consensus protocol; so, it can easily support more transactions, which makes consortium blockchain more scalable.
- Every node in the consortium can write or read transactions, but no one can add a block on their own without multiparty consensus, i.e., every node must confirm that block.
- If this rule is unsatisfied, a block cannot be added. Consortium consists of known members that are limited in number, and consensus is like a voting-based system that ensures completion of transactions with low latency and superb speed compared to a public blockchain and makes the network more secure.

5.10 HYPERLEDGER PLATFORM

GQ. What is a hyperledger ? What are its properties?

GQ. Explain Greenhouse Structure of Hyperledger.

- Many different companies interested in blockchain technology realized that with the help of blockchain technology they could work together rather than working separately and achieve their goals quickly. With this aim,such companies decide to pool their resources to develop industry standard open-source blockchain technology which anyone can use, and in this way, hyperledger was created keeping the requirements of an enterprise in mind.
- Hyperledger is an open-source enterprise standard distributed ledger technology (DLT) developed under the flagship of Linux Foundation in 2015. In 2016, Hyperledger accepted code from digital assets, code stream, and IBM, which evolved into an enterprise grade solution Hyperledger Fabric.
- Also, Intel incubated its project called Hyperledger Sawtooth. Different usecases of distributed ledgers can have different requirements, the approach of Hyperledger is to cover an entire spectrum of use cases and apply best practices of distributed systems in blockchain for enterprise solutions. To address this diversity, all Hyperledger projects ensures that the projects must include the following properties:

(1) Modular

- Hyperledger is developing modular, extensible frameworks with common building blocks that can be reused. This approach enables developers to experiment with a variety of components as they evolve and to change individual components without affecting the rest of the system.
- This helps developers to create components that can be combined to build distributed ledger solutions well-suited to different requirements.

(2) Secure

- Security is a key consideration for distributed ledgers due to the involvement of high-value transactions or sensitive data in various use cases.
- Security and robustness are the primary concerns in an enterpriseclass blockchain, and they provide critical infrastructure for next-generation business networks.

(3) Interoperable

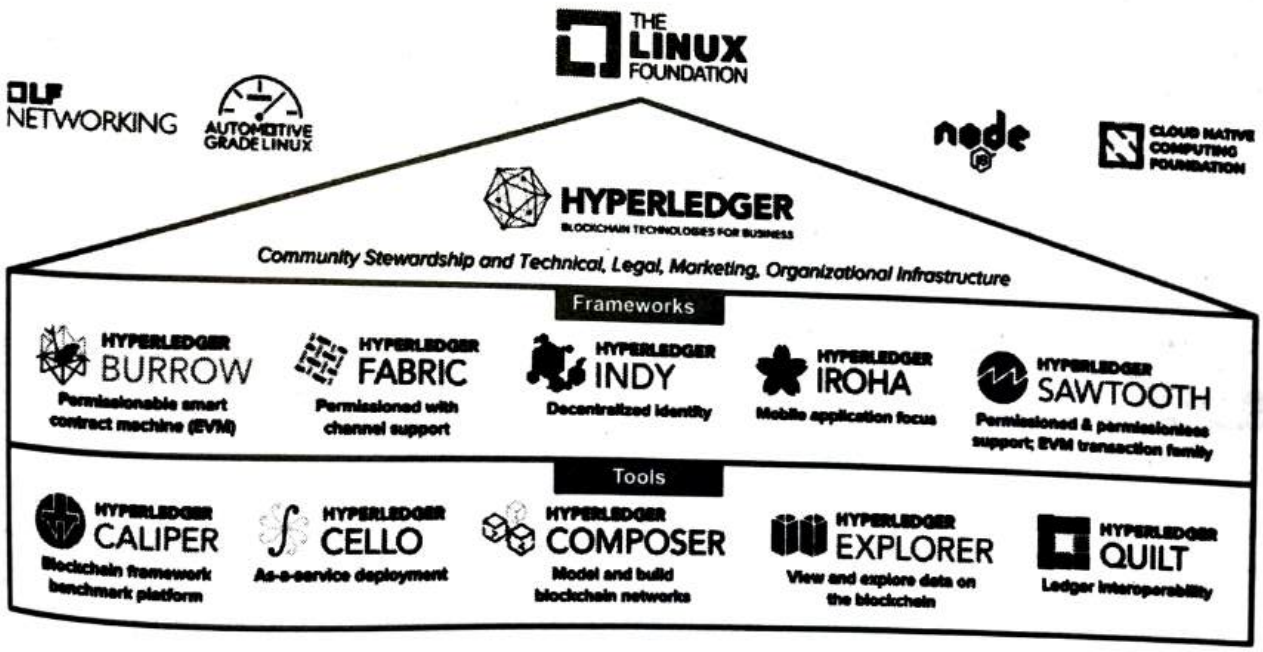
- In future, different blockchain networks will communicate and exchange data to form more complex and powerful networks.
- Hyperledger projects ensure that most smart contracts and applications should be portable across many different blockchain networks, which increases the chances of adoption of hyperledger technology.

(4) Cryptocurrency agnostic

- Hyperledger exists to create blockchain software for enterprises, not to administer any cryptocurrency. The projects are independent and agnostic of all altcoins, cryptocurrencies, and tokens.
- Hyperledger will not issue its own cryptocurrency, and it may create a token used to manage digital objects.

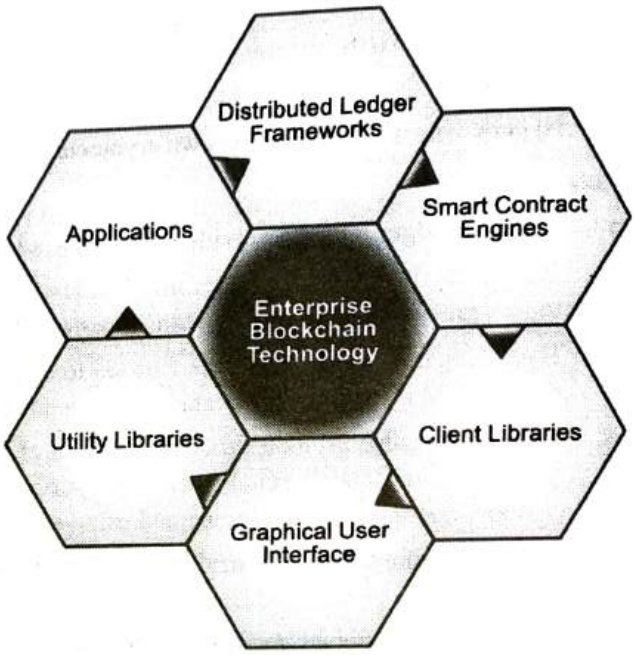
(5) Easy to use

- All hyperledger projects provide rich and easy-to-use APIs that support interoperability with other systems.
- With these designing principles in mind, Hyperledger provides multiple alternative solutions as reusable modules to interface quickly and easily with Hyperledger's core distributed ledger infrastructure.
- Hyperledger Projects (HLP) are focused to create an enterprise grade, open source distributed ledger framework, and to build an open source technical community to support the ecosystem of HLP solutions.
- Hyperledger also promotes the participation of leading members of the ecosystem including developers, solution providers, and end users. Hyperledger serves as a "greenhouse" that brings together users, developers, and vendors interested in developing and using enterprise blockchains from many different sectors and market spaces.
- This greenhouse structure incubates new ideas, supports each other with essential resources, and distributes results widely.
- Hyperledger supports many different varieties with optimum consumption of resources. As the greenhouse organization for open-source blockchain development, Hyperledger improves collaboration by creating an environment that streamlines communication, and with better communication, new participants get faster access to necessary information.



(1D2)**Fig. 5.10.1 : Greenhouse Structure of Hyperledger**

- As newer participants quickly join the collaborative effort, this speeds up development for the benefit of the entire community.
- To improve productivity, Hyperledger's greenhouse structure encourages specialization where instead of competing with each other and duplicating each other's efforts, specialists are encouraged to work together to accelerate their research and development.
- These collaborative efforts streamline the development of new projects, and encourage the creation of common components, and it also enhances interoperability and resolves issues between various distributed ledgers due to the better understanding of other projects.
- The greenhouse structure also helps in the governance of projects and resolutions of dispute and handling of intellectual property.
- Hyperledger includes a wide range of business blockchain technologies, as shown in Fig. 5.10.2.
- In hyperledger,the development of projects encourages the reuse of common modules, enables rapid innovation of components, and promotes interoperability between projects.



(1D3)**Fig. 5.10.2 : Types of Hyperledger enterprise blockchain technologies**

5.10.1 Hyperledger Project – Framework

GQ. Describe the structure of a hyperledger.

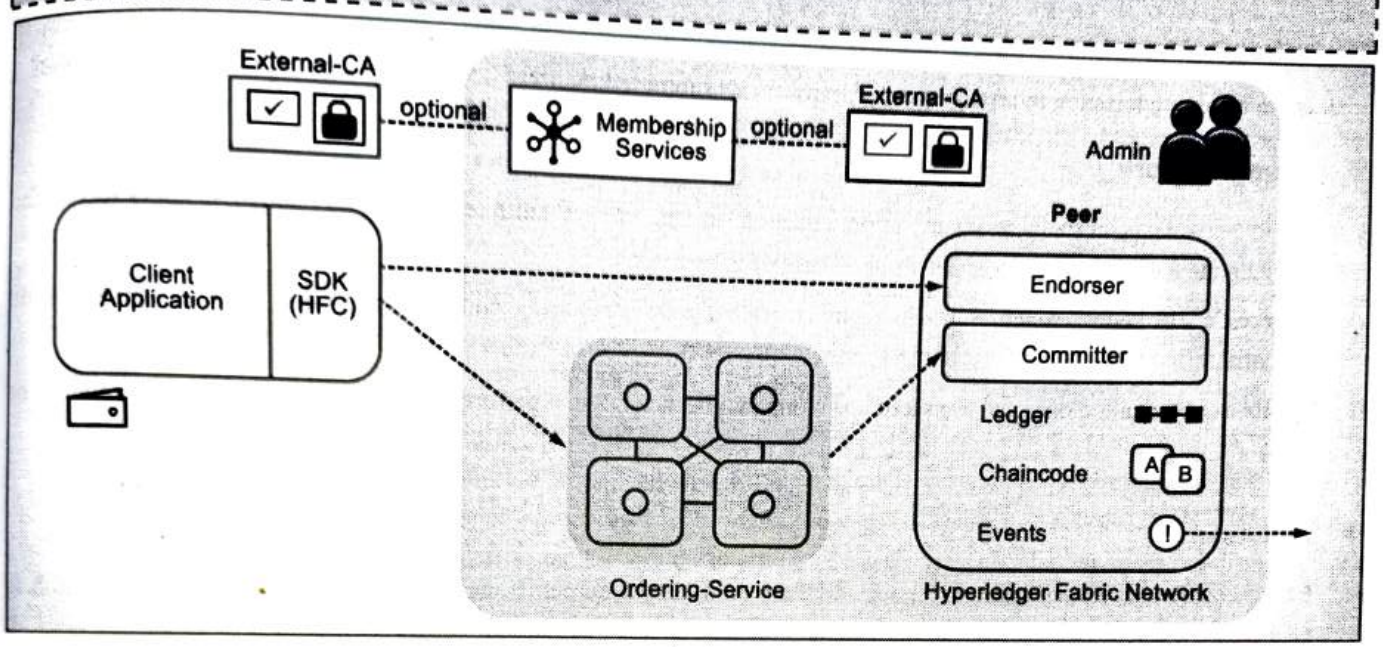
Currently, there are five different types of distributed ledger frameworks included in the Hyperledger ecosystem that are briefly described below :

(1) Hyperledger Fabric	(2) Hyperledger Sawtooth	(3) Hyperledger Iroha
(4) Hyperledger Burrow	(5) Hyperledger Indy	

NOTES

(1) Hyperledger Fabric

GQ. Explain Hyperledger Fabric V1 architecture.



(1D4)**Fig. 5.10.3 : Hyperledger Fabric V1 Architecture**

- Hyperledger Fabric is a platform for developing distributed ledger solutions, including a modular architecture that delivers high degrees of confidentiality, flexibility, resiliency, and scalability.
- The fabric provides components, such as consensus and membership services, to be plug-and-play. It has container technology to host smart contracts called "chain code" that hold the business rules of the system. Fabric is designed to support pluggable components.
- Architecture of Hyperledger includes various components, shown in Fig. 5.10.3, that work together to execute multiple transactions of users in a peer-to-peer network in a specific order.
- Working of components are briefly explained below.

☞ Membership Services

- They provide identity to users using and performing transactions on ablockchain network.
- This identity is recognized in the form of digital certificates. Users will use digital certificates to authorizea user and authenticate transactions through digital signatures.

☞ Certification Authority (CA)

- It issues digital certificates to a user, which includes the attributes of standard digital certificates. The certificate can be issued by a traditional external CA or fabric CA.
- Fabric CA is an optional and pluggable module, which may be customized as per the need of an enterprise. These certificates will contain a public key. This key is required for secure communication over the Internet.
- Client application is used to interact with blockchain using Hyperledger Fabric Client SDK. It is used by clients to perform translation on blockchain.

☞ Admin

It configures the network, defines the endorsement policy, and manages the blockchain network.

☞ Ordering Service

It is run by an organization to arrange a set of transactions submitted by users in order and returns a transaction in order.

☞ Fabric Network

- Fabric Network is on the blockchain, an organization running one or multiple peers on the network. Peers record and maintain the ledger, which stores details and states of transections.
- It stores chain codes, which is nothing but smart contracts containing code running on a hyperledgerblockchain network.
- Events can be emitted once the transaction is complete. Peer nodes can perform functions of anendorser, committer, or both.

☞ Endorser

- Endorser executes achain code on the network and validates the output. It maintains a copy of thechain code which it executes and verifies the correctness of the output.
- Legitimate transactions are submitted to the ordering service, which returns the order of transactions to be committed, and finally these order transactions are committed by the committer node and recorded on the block. Peers can be either an endorser or a committer or can perform the role of both.
- In fabric, transaction flows from endorsement to ordering and then commit. Client proposes transaction for running chain code and provides input.
- This transaction is submitted to multiple endorsers, which execute the transactions. Endorser executes and validates the received transaction.
- If endorsement policy (e.g., all or more than half or specific endorsers sign and validate the transections) is satisfied, then it is submitted to the ordering services. Ordering service will order the transections and create the blocks.
- Fabric is an extensible blockchain platform for running distributed applications. It supports various consensus protocols; so, it can be utilized in various use cases and trust models.
- In contrast to other blockchain platforms which either require code to be written in a domain-specific language or else rely on a cryptocurrency for running smart contracts, Fabric runs distributed applications written in general-purpose programming languages without any native cryptocurrency.
- Furthermore, Fabric uses a portable notion of membership for the permissioned model, which can be integrated with industry-standard identity management. Fabric maintains such flexibility by considering a novel architectural approach and revamps the way blockchains cope with non-determinism, resource exhaustion, and performance attacks.
- Fabric also includes the notion of channels, which is created to enable a group of participants to form a separate ledger of transactions.
- This is helpful for networks where some participants don't want every transaction, such as in the case of competitors a special price offered to some without informing other participants in the network. If a group of participants forms a channel, only those participants and no others have copies of the ledger for that channel.

(2) Hyperledger Sawtooth

GQ. Explain Hyperledger Sawtooth.

- Hyperledger Sawtooth provides a platform for building, deploying, and running distributed ledgers. Distributed ledgers provide a digital record (e.g., asset ownership).
- The records are maintained without any central authority or implementation. Sawtooth aims to facilitate safe and enterprise use of smart contracts keeping distributed ledgers distributed instead of storing on the central server. Sawtooth is highly modular, which enables enterprises and consortiums to select their blockchain applications by themselves.
- Sawtooth contains several technical innovations including :
  - (i) **Dynamic consensus :** Consortiums can change consensus algorithms on a running blockchain simply by issuing a transaction.
  - (ii) **Proof of elapsed time (PoET) :** A consensus algorithm with the scalability of PoW with low power consumption.
  - (iii) **Compatibility :** Transaction families enable users to write smart contract logic in the language of their choice. It can also integrate other smart contract interpreters including Hyperledger Burrow's Ethereum Virtual Machine.
  - (iv) **Parallelism :** Advanced parallel scheduler of Sawtooth splits blocks into parallel flows. Parallelism produces faster block processing to improve the performance of blockchains compared to traditional databases.
  - (v) **Privacy :** Sawtooth nodes can be deployed into clusters with separate permissions. It provides privacy and confidentiality among participants of that distinct chain. There is no exposure or leak of transaction patterns or other confidential information from central services. However, an intermediary Hyperledger Quilt is required to connect separate chains.
- Sawtooth provides scalability, security, privacy, and modular design. Its consensus model PoET boosts scalability, and the transaction families increase the scope of smart contracts and reduce the potential attack surface. Sawtooth also allows trusted execution environments and the rolesthey play in private transactions.

(3) Hyperledger Iroha

GQ. Describe Hyperledger Iroha.

- Hyperledger Iroha is a blockchain framework designed to be simple and easy to incorporate into projects that require DLT. Iroha is the third distributed ledger platform under Hyperledger and was included in October 2016.
- It was developed by Soramitsu in Japan and was proposed to Hyperledger by Soramitsu, Hitachi, NTT Data, and Colu.
- Features of Iroha are :
  - (i) Simple structure
  - (ii) Domain-driven C++ design
  - (iii) Emphasis on mobile application development
  - (iv) Chain-based BFT consensus algorithm (Sumeragi)
- Iroha follows a different approach from Fabric and Sawtooth. It provides features that are helpful for creating applications for endusers.

(4) Hyperledger Burrow

GQ. What is Hyperledger Burrow ?

- Burrow is the fourth distributed ledger platform included within Hyperledger in April 2017. It was developed and proposed to Hyperledger by Monax. Hyperledger Burrow is a permissioned smart contract machine.

- It provides a modular blockchain client with a smart contract interpreter developed partly to the specifications of the EVM. Burrow provides a strongly deterministic, smart contract-focused blockchain design.
- Burrow includes the following components :
  - (i) **Consensus engine :** It maintains the networking stack between nodes, and orders transactions for use by the application engine.
  - (ii) **Application blockchain interface (ABCI) :** It provides the interface specification to connect consensus engine with application engine.
  - (iii) **Smart contract application engine :** It provides a strongly deterministic smart contract engine to developers for operating complex industrial processes.
  - (iv) **Gateway :** It provides programmatic interfaces for system integrations and user interfaces.

(5) Hyperledger Indy

GQ. Explain in brief Hyperledger Indy.

- Hyperledger Indy is a distributed ledger used for decentralized identities. Indy provides tools, libraries, and components for creating and using independent digital identities on blockchains or distributed ledgers. Indy includes reusable components and provides interoperable services across applications, administrative domains, and organizations operating independently and avoids sharing information.
- Indy provides information and verification of data about the other interacting party that enables trusted interactions between enterprises. Using Indy, friends, competitors, and even antagonists can rely upon the shared source of truth and work together.
- Key features of Hyperledger Indy are as follows :
  - (i) **Self-sovereignty :** Indy stores identity artifacts on a ledger with distributed ownership such that only authorized users can change or remove identity. These artifacts include public keys, proofs of existence, cryptographic accumulators that enable revocation, etc.
  - (ii) **Privacy :** Indy maintains privacy so that every identity owner can operate without creating any correlation risk.
  - (iii) **Claim verification :** Identity claims are verifiable from credentials such as birth certificates, driver's licenses, passports, etc. However, these are combined and transformed in powerful ways using zero-knowledge proofs to avail the disclosure of selective data based on the requirement of a specific context.
- This blend of self-sovereignty, privacy, and claim verification is effective and brings lots of potential benefits. Individuals and organizations can benefit from richer and more secure interactions.
- Despite the advanced cryptography under the hood, Indy's API is simple and straightforward. This API includes about 50 C callable functions with idiomatic wrappers for many mainstream programming languages.

5.10.2 Hyperledger Project – Tools

GQ. What are the different types of tools and utility libraries used by hyperledger ?

Hyperledger projects also include tools and utility libraries. Brief description of various Hyperledger block chain tools are given below :

(1) Hyperledger Caliper

- Hyperledger Caliper is a blockchain benchmark tool used to measure the performance of any blockchain implementation.

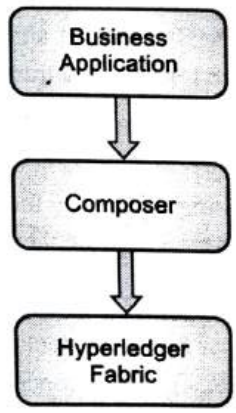
- It is a general tool that provides performance evaluations for different blockchain solutions based on a set of neutral and commonly accepted rules.
- Caliper generates reports based on various performance indicators, includes resource utilization, transaction latency, and transactions per second (TPS).
- Caliper community is continuously setting new performance indicators and benchmark use cases.
- Caliper is used as a reference to help choose the blockchain implementation suitable for a company's specific needs. It is an in-house tool, and its result is not published.
- Hyperledger Caliper gives a functioning benchmark tool that can run on multiple Hyperledger frameworks.

(2) Hyperledger Cello

- Hyperledger Cello facilitates an on-demand deployment model to the blockchain ecosystem for quick and easy adoption of blockchain technologies by enterprises.
- It provides automatic creation, termination, and management of blockchains. The multitenant chain service of Cello is efficient and can run on top of various infrastructures, including bare metal, virtual machines, cloud platforms like Amazon Web Services (AWS), and container platforms like Docker Swarm and Kubernetes, that boost the efficiency of "Blockchain-as-a-Service" (BaaS).
- Cello provides a real-time dashboard for users to :
  (i) View the status of the blockchain system.
  (ii) See statistics of blockchain events, chain code performance, and system utilization.
  (iii) Manage blockchains by creating, configuring, and deleting.
  (iv) Manage chain code by deploying and uploading a private chain code.
- Hyperledger Cello supports Hyperledger Fabric as the main blockchain implementation. The architecture of most of the components follows the microservice style with pluggable implementations.
- The main programming languages used are Python and JavaScript.

(3) Hyperledger Composer

- Hyperledger Composer provides a toolset for easy and fast creation of smart contracts and blockchain applications to solve business problems.
- The main goal is to facilitate easy integration of blockchain applications with present systems and thus accelerate time-to-value.
- Hyperledger Composer can speed up the development of use cases and deploy a blockchain solution. Composer also enables users to quickly model an existing business network and integrate existing systems and data with blockchain applications.
- A network can contain assets such as tangible or intangible goods, services, or property and transactions related to them.
- As part of the model, users can define the interaction of transactions with assets. Business networks include involved participants with a unique identity across several different business networks.
- Hyperledger Composer supports the existing Hyperledger Fabric blockchain infrastructure and runtime.



(105)**Fig. 5.10.4 : Application development with Hyperledger Composer**

(4) Hyperledger Explorer

- Hyperledger Explorer provides a dashboard for viewing blocks, node logs, statistics, smart contracts, transactions, and other information stored in the blockchain.
- Users can query and view complete details of specific blocks or transactions. Explorer can be integrated with any authentication or authorization platforms, either commercial or opensource, to provide the functions appropriate to a user's privileges.
- The characteristics of the explorer are as follows :

  (i) Generic and cross-platform (ii) Easy to install, implement, maintain, and extend

  (iii) Supports standard package manager and use of latest technologies
- Explorer currently supports the Hyperledger Fabric framework.

(5) Hyperledger Quilt

- Hyperledger Quilt is an interoperability solution for Hyperledger projects. It provides interoperability between ledger systems with the help of Interledger Protocol (ILP).
- ILP is a simple, open-source enterprise-grade protocol that maintains a global namespace for accounts to make transactions across ledgers. By implementing ILP, Quilt provides :

  (i) A set of rules for enabling ledger interoperability with basic escrow semantics.

  (ii) A standard for a ledger-independent address format and data packet.

  (iii) A framework for designing high-level protocols for specific use cases.
- Quilt provides libraries and reference implementations of the core interledger components. This will enable distributed ledger solutions from Hyperledger members, the private ledgers from financial institutions, the wallets from IoT companies, and supply chain systems to connect with one another to perform distributed atomic transactions.

☞ Different use cases

- We are living in the information era where the world is highly interconnected. This connection will be stronger, and we will be closer in the future.
- The data is produced and shared on a high scale, and more collaboration will be needed to solve the common challenges. Hyperledger aims to work on resolving the issues by developing an ecosystem where the organization will collaborate together in a trusted environment, which will ensure security and privacy.
- Hyperledger is exploring blockchain technologies to build innovative products which can provide solutions to different domains. The technologies can be applied to a wide range of domains, and a few use cases are listed below :

  (i) Banking : Applying for a loan **(ii) Financial services :** Post-trade processing

  (iii) Healthcare : Credentialing physicians **(iv) IT :** Managing portable identities

  (v) Supply chain management : Product traceability
- Hyperledger has useful tools available which can be applied to the use cases listed above and resolve current issues; in some cases, a proof-of-concept has already been developed.

5.11 HYPERLEDGER FABRIC ARCHITECTURE

- Hyperledger fabric is an opensource project managed by the Linux Foundation and contributed by IBM. It is intended as a foundation for developing enterprise-grade applications and industry solutions.
- The application developed with hyperledger does not require cryptocurrency and allows programmable smart contracts, which include the application logic of the system, hyperledger fabric leverages containers to host smart contracts also called chaincodes. These enterprise applications and solutions are designed with the following features:
  - Modular and Extensible Architecture
  - Pluggable Component
  - Permissioned Network
  - Scalability
  - Security
  - Speed
  - Interoperability
- The modular architecture uses pluggable components (consensus and membership services, etc.).This flexibility opens the possibility of a wide range of customization of applications to support various enterprise usecases. These applications can be scaled to meet a large number of users in a secure way.
- Confidentiality is achieved by creating networks of networks for sharing information only to relevant organizations using channels. Completely isolated transactions can be performedto make data private. So, unlike open and permissionless systems, hyperledger fabric provides a permissioned, scalable, and secure platform to build solutions that support private transactions and confidential contracts.
- Hyperledger fabric architecture of *N* participating organization is described in the figure. An organization is a business entity participating in a hyperledger fabric-based permissioned blockchain network and deployed in the fabric infrastructure. Components of the fabric infrastructure are explained below.

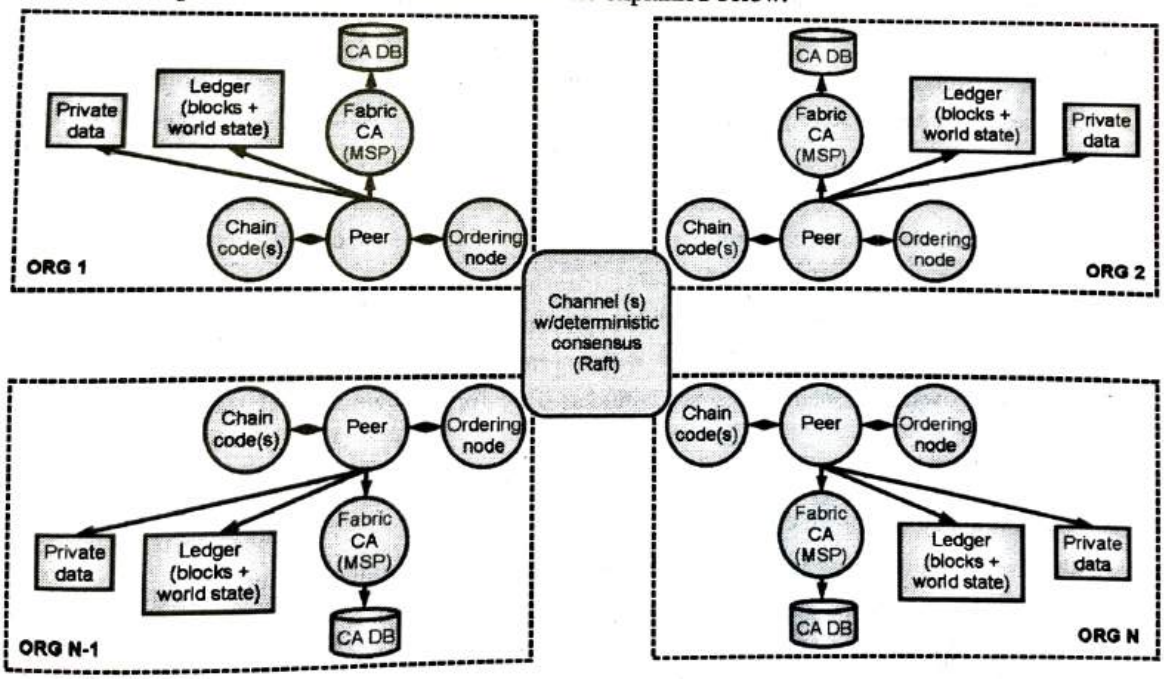


Fig. 5.11.1 : Hyperledger Fabric Architecture

System Architecture

- The distributed network of hyperledger fabric includes many peers interacting with each other. Peers hold state and ledger and are also capable of executing transactions using chaincodes. Transactions are then "endorsed" and only endorsed transactions may be committed and have an effect on the state.
- The validating peers run a Byzantine fault tolerance consensus protocol for executing a replicated state machine that accepts three types of transactions as operations:
  - **Deploy transaction :** Takes a chaincode (representing a smart contract) written in Go as a parameter; the chaincode is installed on the peers and ready to be invoked.
  - **Invoke transaction:** Invokes a transaction of a particular chaincode that has been installed earlier through a deploy transaction. Note that the arguments are specific to the type of transaction. The chaincode executes the transaction and indicates whether it succeeded or failed.
  - **Query transaction:** Returns an entry of the state directly from reading the peer's persistent state, this may not ensure linearizability.
- Each chaincode may define its own persistent entries in the state. The blockchain's hash chain is computed over the executed transactions and the resulting persistent state.
- Validation of transactions occurs through the replicated execution of the chaincode and given the faulty assumption underlying BFT consensus, i.e., that among the n validating peers at most $f<n/3$ may "lie" and behave arbitrarily, but all others execute the chaincode correctly.
- When executed on top of PBFT consensus, it is important that chaincode transactions are deterministic; otherwise, the state of the peers might diverge. A modular solution to filter out non-deterministic transactions is implemented in the SIEVE protocol.
- Membership among the validating nodes running BFT consensus is currently static and the setup requires manual intervention.
- The fabric implements a permissioned ledger.It contains a security infrastructure for authentication and authorization. It supports enrollment and transaction authorization through public-key certificates and confidentiality for chaincode realized through in-band encryption.
- To connect with the network, every peer needs to obtain an enrollment certificate from an enrollment CA that is part of the membership services. It authorizes a peer to connect to the network and to acquire transaction certificates, which are needed to submit transactions.
- Transaction certificates are issued by a transaction CA and support pseudonymous authorization for the peers submitting transactions, in the sense that multiple transaction certificates issued to the same peer (i.e., to the same enrollment certificate) cannot be linked with each other.
- Confidentiality for chain codes and states is provided through symmetrickey encryption of transactions and states with a blockchain-specific key that is available to all peers with an enrollment certificate for the blockchain. Extending the encryption mechanisms towards more fine-grained confidentiality for transactions and state entries is planned for a future version.
- The hyperledger fabric architecture includes various components to facilitate the key blockchain features and provide isolation from the contract development construct.
- Hyperledger fabric architecture is used by a blockchain developer in the following manner:
  1. A developer creates the application and smart contract on the network.
  2. The developer uses DEPLOY to
     (a) Deploy an app on a server (b) Deploy a smart contract on a peer

3. Smart contracts enable every registered user to order
   (a) INVOKE to interact with the app
   (b) QUERY to retrieve information
4. A smart contract can emit an event that the app subscribes.

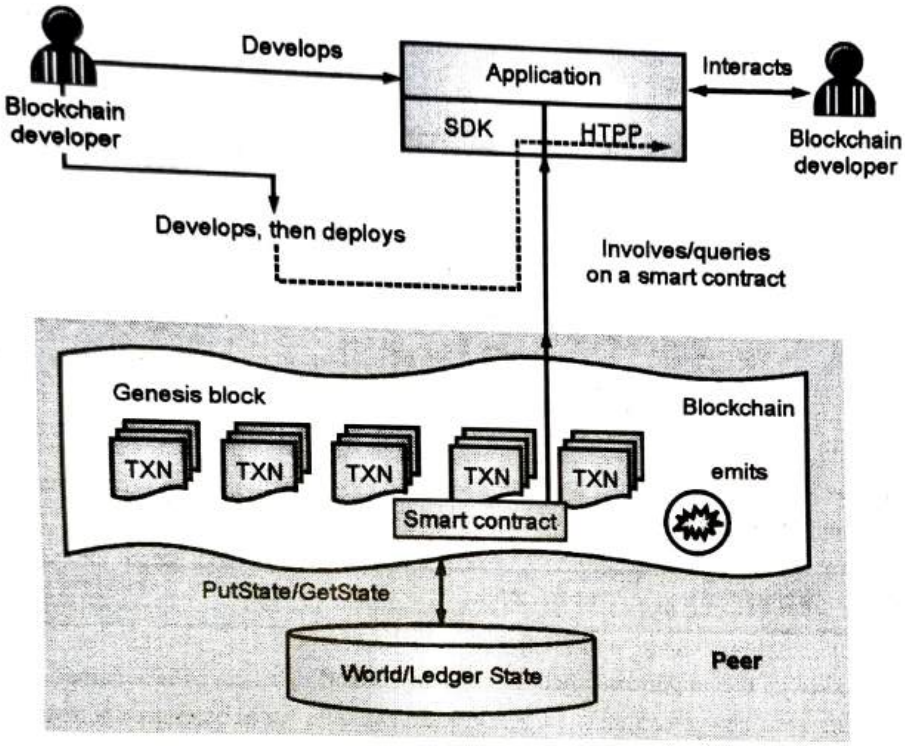


Fig. 5.11.2 : Use of Hyperledger Fabric Architecture

- In the above process, the participating entities or organizations can secretely communicate with each other using channels. Channels create a tunnel between parties engaged in the activities and keep communication isolated from other organizations; in this way, the organization can create and join channels for secretly exchanging information. An organization can be a part of multiple channels and access information or transactions associated with only those channels.

5.12 COMPONENTS OF HYPERLEDGER FABRIC

- Hyperledger fabric components facilitate blockchain features and provide ease of development of the application. These components also represent the hyperledger fabric infrastructure components. In this section, the components of hyperledger fabric and their role and services are explained.

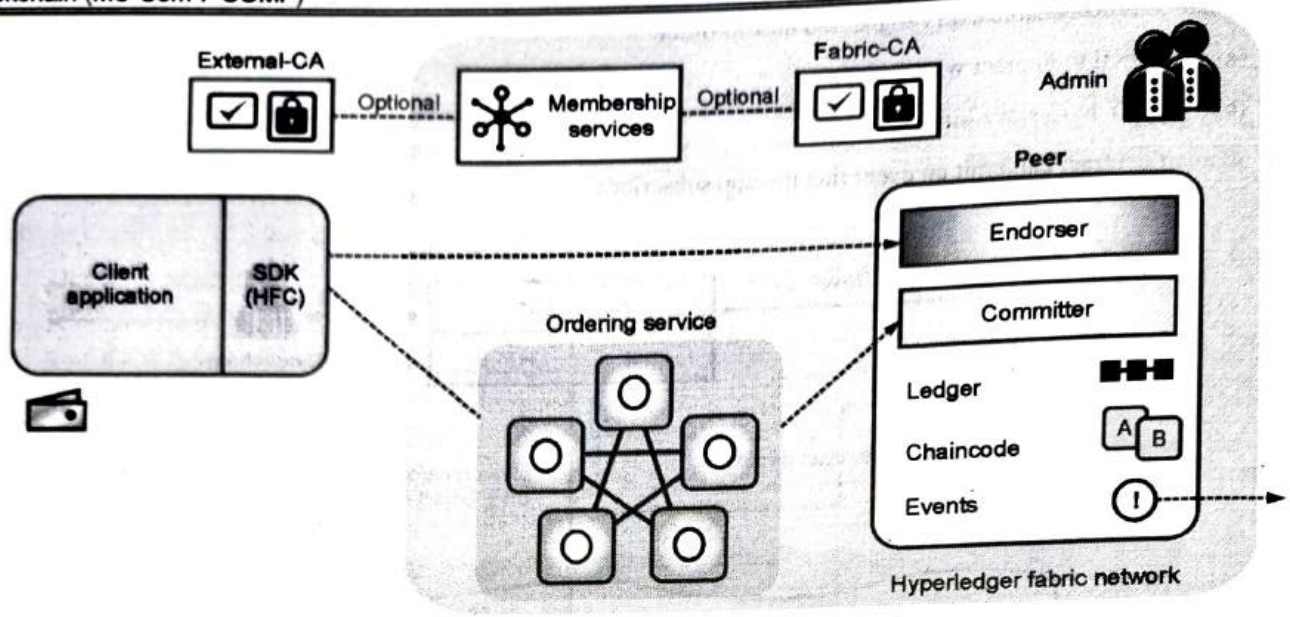


Fig. 5.12.1 : Hyperledger Fabric infrastructure components

- The above figure represents the components of hyperledger fabric infrastructure. It includes three main key components MSP, ordering service, and peers.

5.13 MEMBERSHIP SERVICE PROVIDER (MSP)

- Hyperledger fabric applications run in permissioned blockchain networks; hence, MSPbecomes an essential component of hyperledger fabric.
- It provides identity management services to the network participants and allows the registration of unknown entities to join and participate in the network.
- Certification authority-based membership service can be implemented using the Fabric CA module. Fabric CA can use any X.509-based PKI infrastructure to issue and manage digital certificates.
- The certificate issued by CA provides information about permissions, roles, and attributes for the use of a channel.
- The ability of a user to query or invoke a transaction on any channel depends on the permissions, roles, or attributes maintained in the issued certificate.
- To maintain a high degree of anonymity and unlinkability of transaction,identity mixer-based implementation is used. Identity mixed allows users to transact without revealing their identity and also send multiple transactions without revealing that the transactions are coming from the same user.

Ordering Service

- Ordering services create a new block of ordered transactions and then distribute it to peers belonging to a specific channel.
- The ordering service includes a dedicated order node that picks, order, and create a batch of ordered transactions (blocks) and signs to create a hash chain.
- The implementer of the order node provides Atomic Broadcast API and distributes the newly created block to the peers of a specific channel in the network.

- This service is pluggable and has various implementations for different needs :
  - Solo is used for development testing.
  - RAFT is used for production.
  - Kafka-based implementation is used for production with high fault tolerance.

Peers

- Peers are responsible for executing the smart contract and also stores the ledger.
- A peer who joined the channel can access all the transactions of that channel. Peers can join multiple channels to maintain confidentiality among different organizations.
- To eliminate non-determinism, two types of peers are included in the network:
  - **Endorser peers** only execute a transaction but does not commit it. These uncommitted transactions are picked by an orderer who batches them and forwards them to committer nodes.
  - **Committer peers** get batches of endorsed transactions from an orderer node and performs validation and then commits transactions.

The remaining components help to manage the node and perform network activities such as

- The **client** interacts with hyperledger fabric blockchain network as per the role, permission, and attributes. These attributes are obtained from the certificate issued and maintained by the specific implementation of CA.
- **Admin** installs the chaincode on the target peers of the network.With the help of the orderer, the network admin instantiates the chaincode on a specific channel. Admin also creates an endorsement policy, which defines which peer can authenticate transaction result before adding it onto the ledger of all the peers on the channel.
- **Ledger** is channels chain and current state data, which is maintained by each peer on the channel. State data is a database that stores the current state of the ledger state, which can be created, updated, and deleted. In the beginning, when a ledger is created, the world state is empty, and as any valid transaction is committed first, the state data is updated first, and then it is updated in the ledger.
- **Chaincode** is installed by the network admin on the peers and channels. It interacts with the ledger shared on the blockchain network.

There are two different types of chaincodes:

- **System chaincode:**It typically handles system-related transactions such as lifecycle management and policy configuration.
- **Application chaincode:**It manages application states on the ledger, including digital assets or arbitrary data records.
- Smart contracts are defined within a chaincode and a chaincode can include multiple smart contracts. Chaincode acts as a container for a group of smart contracts, and these smart contracts include application domain-specific logic along with endorsement policy provided via chaincode.If the transaction executes and is validated based on the endorsement policy, then the world state changes and results are appended into the ledger.
- **Events** create notifications for important blockchain operations, e.g., addition of a new block and notification related to a smart contract.

5.14 WORKING OF HYPERLEDGER FABRIC

- Every user or participant on the fabric's network must register proof of identity to membership services.
- The registration gives the participant access to the system.
- The fabric issues a transaction with derived certificates from the implemented certification authorities server.
- The content of each transaction is encrypted. It ensures that only the intended participants can view the content.

- Every transaction is secure, private, and confidential on the fabric network.
- Only the consensus of the peers on the network can update hyperledger fabric.
- Each transaction executes without a cryptocurrency, and the events are structured as transactions and distributed among different participants.
- Every transaction relies on a smart contract system (chaincode), which every participant on the network runs in docker containers.

5.15 TRANSACTION FLOW

Consider a standard asset exchange wherein there are two clients, i.e., A and B, who are buying and selling radishes. They each have a peer on the network through which they send their transactions and interact with the ledger.

Fig. 5.15.1

Setup and Installation

1. The channel is set up and running. The writing policy is defined at channel creation time and determines which users are entitled to submit a transaction to that channel.
2. The application user has registered and enrolled with the organization's certificate authority (CA) and received back the necessary credentials to authenticate themselves on the network.
3. The chaincode (containing a set of key–value pair representing the initial state of the radish market) is installed on the peers and instantiated on the channel.
4. The chaincode contains logic defining a set of transaction instructions and the agreed-upon price for a radish.
5. An endorsement policy has also been set for this chaincode, stating that both peerA and peerB must endorse any transaction.

Step 1 : Transaction is initiated by PeerA

- Client A wants to send Client B a request to purchase radishes.
- The request targets peerA and peerB, who are respectively representative of Client A and Client B. The endorsement policy states that both peers must endorse any transaction; therefore, the request goes to both peerA and peerB.

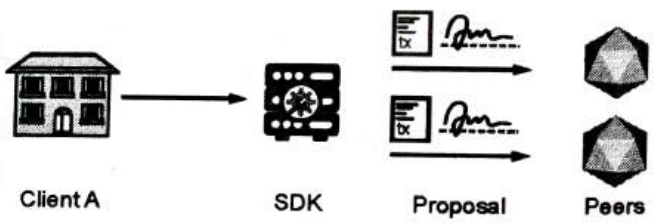


Fig. 5.15.2

- First, the transaction proposal is constructed using SDK.
- The proposal is a request to invoke a chaincode function so that data can be read and/or written to the ledger (i.e., write new key-value pairs for the assets).

Step 2 : Verification of signature & execution of the transaction by endorsing peers

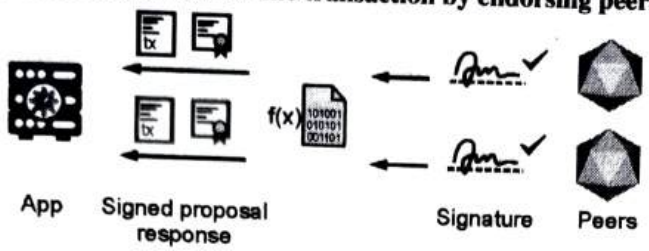


Fig. 5.15.3

- The endorsing peers verify :
  1. Formation of transaction proposal
  2. It hasn't previously been submitted in the past for replay attack protection
  3. Validity of signature using MSP
  4. Authorization of submitter, i.e., client A is properly authorized to perform the proposed operation on that channel or not, each endorsing peer ensures that the submitter satisfies the channel's *Writers* policy.
- After verification, the endorsing peers invokes the chaincode's function, and the proposal input is considered as arguments.
- The chaincode is then executed against the current state database to produce transaction results including a response value, read set, and write set. No updates are made to the ledger.
- The set of these values along with the endorsing peer's signature is passed back as a "proposal response" to the SDK, which parses the payload for the application to consume.

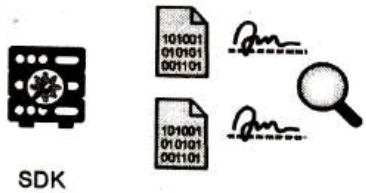


Fig. 5.15.4

Step 3 : Proposal response Inspection

- The application verifies the endorsing peer signatures and compares the proposal responses to determine if the proposal responses are the same.

Note : If the chaincode only queried the ledger, then the application would inspect the query response and won't be submitting the transaction to ordering service.

- The application determines if the specified endorsement policy has been fulfilled, i.e., peerA and peerB both endorse.
- Client A (proposed a new value in the transaction) will submit the transaction to Ordering Service for updating the ledger.
- Before submitting, even if an application chooses not to inspect responses and just forwards an unendorsed transaction to the ordering service, the endorsement policy will still be enforced by peers and upheld at the commit validation phase.

Step 4 : Ordering and Assembly of Transactions

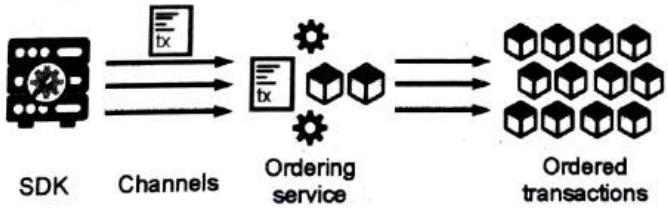


Fig. 5.15.5

- The application "broadcasts" the transaction proposal and response within a "transaction message" to the ordering service.
- The transaction will contain the read/write sets, the endorsing peer's signatures, and the channel ID.
- The ordering service receives transactions from all channels in the network, orders them chronologically by channel, and creates blocks of transactions per channel.

▶ **Step 5 : Transaction validation and Transaction commit**

- The ordering service broadcasts the blocks of transactions to all peers on the channel.
- The transactions within the block are validated by the peers to ensure the endorsement policy is fulfilled and that there have been no changes to the ledger state for reading set variables since the read set was generated by the transaction execution.

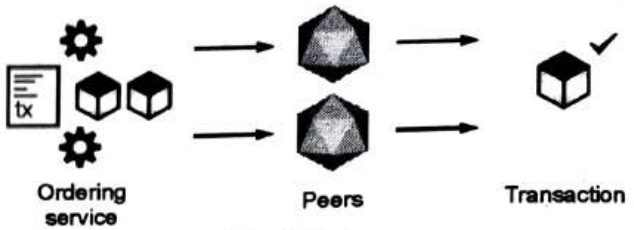


Fig. 5.15.6

- Transactions in the block are tagged as being valid or invalid.

▶ **Step 6 : Ledger Updated**

- Each peer appends the block to the channel's chain, and for each valid transaction, the write sets are committed to a current state database.
- An event is emitted to notify the client application that the transaction (invocation) has been immutably appended to the chain as well as notification of whether the transaction was validated or invalidated.

5.16 CREATING HYPERLEDGER NETWORK

- In this section, we will understand the process for setting the hyperledger network, The build your first network (BYFN) scenario provisions a sample Hyperledger Fabric network consisting of two organizations, each maintaining two peer nodes and a "solo" ordering service.
- Prerequisite to start with the network setup process (use *curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh*, then unzip using sudo for installation)
  - Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS
  - Docker Engine: Version 17.03 or higher
  - Docker-Compose: Version 1.8 or higher
  - Go programming language: Go version 1.12.x
  - Node: 8.9 or higher
  - npm: v5.x
  - git: 2.9.x or higher
  - Python: 2.7.x

5.16.1 Steps to create a Network

1. Setting Ordering Service: Ordering service determines how transactions are ordered in the network. It could be solo or distributed ordering node based on the requirements.
2. Setting Peers: There are multiple organizations participating in the network can run one or more peers. These peers are configured and started for each enforcer and committer.

3. *Note: Identities of ordering services and each peers are maintained in the network with MSP and provided by the certification authority.*
4. Chaincode Installation: Install one or more chaincode on the endorsing peers (subset of peers) that can execute it. Installation of chaincode on the subset of peers will provide privacy and only the peers with installed chaincode can execute it instead of all the peers available on the distributed network. These chaincodes are implemented using containers and includes the business logic.
5. **Channel Configuration:** The next stage is to begin with connection setup with the channel configuration on ordering services with multiple nodes, one or more ordering services can be configured with multiple channel configuration. After the channel configuration to complete the connection with the ordering service using the client application organizations peers with appropriate permission joins the channel. Different subset of node can join different channel and also one peer can be part of multiple channel.
6. **Instantiate chaincode on channel:** In the step the peers can instantiate the chaincode on the channel with connection. Channels are also mapped with the chaincode and based on the endorsement policy the peers executes the chaincode on appropriate channel. Each chaincode will have different state and ledger in each channel and all the chaincode across the peers in a same channel are same this consistency is achieved through consensus

The above setup can be performed and executed using simple CLI command or SDK

5.17 CASE STUDY OF SUPPLY CHAIN MANAGEMENT USING HYPERLEDGER

GQ. State various use cases where hyperledger is used.

- Hyperledger fabric is the foundation for developing applications for different types of industry use cases, in this section, we will understand supply chain solutions using Hyperledger Fabric.
- A supply chain is a network of suppliers to manufacturers and distributors of a certain product to end-users. This entire network chain contains a variety of activities, information, people, and resources. It's the set of steps and actions the industry takes to deliver its product to the end-user. There are five actors in the supply chain namely producer, supplier, distributor, retailers, and customers. The components of the supply chain will also perform a variety of tasks, such as receiving the customer's order. Thus, a wider range of functions, including marketing, development, operations, customer services, and many more, are included. In supply chain management a firm will handle every aspect, from delivering the final product to managing the raw materials.

Phases of a Traditional Supply Chain Management

There are mainly six phases in supply chain management briefly explained bellow

- **Planning :** The manager of the firm will design the manufacturing line at this phase to meet client requests and provide the highest output with the fewest faults. Furthermore, the manager will choose the measurements necessary for a flawless chain after the supply chain is established. It would often rely on the company's objective.
- **Manufacturing :** This phase includes product manufacturing, quality checks, and packing. As a result, the manager will plan each operation for creating the product before scheduling it for delivery.
- **Delivering :** This step includes taking client orders, arranging the delivery, invoicing, and receiving payments. Furthermore, everything must be synchronized to reduce human mistakes.
- **Returning :** Managing any faults and returned items and guaranteeing their return to the factory.
- **Enabling :** The manager will guarantee that everything complies with the rules. These primarily consist of sales, quality control, finance, human resources, and product design.

☞ Issues in Supply Chain

- There are many major issues with the supply chain. Indeed, things look to be simple and easy to manage. In reality, the supply chain is challenged by several problems that are costing businesses their due share of the revenue. A few of these challenges are listed bellow
  - Globalization:
  - Unpredictable Market
  - Transparency
  - Compliance and Quality
  - Cost and Corruption
  - Speed
  - Customer services

☞ Blockchain-based Supply Chain

- Blockchain platform like Hyperledger Fabric provides permissioned blockchain network with scalability, speed, and privacy of data.
- Using Hyperledger querying of data is much easier and all the applications are developed with a pluggable modular design. In this section with the help of a sample supply chain and blockchain platform, we will understand how a supply chain can actually work with a blockchain platform.

Step 1

- In this step, we will first start with meeting the perfect business partners on the blockchain for the supply chain platform. So, using the blockchain in the supply chain platform, both parties can look for their authenticated suppliers or manufacturers.
- As every person on the marketplace will have to show proper identification before getting access to the network, there would be no fake company issues after two parties find each other on the platform they can reach an agreement.
- Furthermore, using smart contracts, the manufacturer can place the order in the blockchain for the supply chain platform.

Step 2

- The supplier then collects all the raw materials of the order and uploads their quality test documents to the blockchain in the supply chain platform. Furthermore, the manufacturer checks the authenticity of the documents and clears the order for transportation.
- Once the supplier gets the confirmation, they tag all the material with tracking chip for proper tracking and inform the producers. After that, they will ship the materials and notify the producers.
- Therefore, both parties will track the shipment in real-time and make sure that the product reaches its destination in a good environment.

Step 3

- Once the manufacturer gets the product, the smart contract will automatically initiate and release the payment to the supplier. Now it's time to track the production line in every stage with the help from the blockchain for the supply chain scheme.
- Furthermore, using the supply chain and blockchain platform, they can also see the condition of the products at every stage. This will allow them to anticipate how long it will take a product to finalize.

- After the finished good is ready, the quality control manager will test out the products and upload the test documents for proper validation. Furthermore, every product will get its relevant QR code and RFID chip added before moving to the warehouse.
- Next, the authority will monitor the warehouse storing process using the blockchain for the supply chain platform. And once it's all done they will initiate another smart contract with the distributor and ship the products. Therefore, everything will go on in the blockchain supply chain examples.

▶ **Step 4**

- In this stage, the supply chains shift to the distributors. After tracking the shipment from the producers upon receipt, the smart contract will automatically release the payment. However, they might choose to store it in their warehouse or ship it directly to the retailers.
- In any case, both scenarios would need blockchain for the supply chain platform for monitoring.

▶ **Step 5**

- Retailers come into play in this one. The same tracking process goes here, as well. Therefore, when the products reach certain retail points, the payment process will initiate automatically. However, blockchain in supply chain platforms can help out in other ways as well.
- For example, it can help forecast consumer behavior based on market analysis. So, using the supply chain and blockchain platform retailers can know exactly what the consumer wants.
- Retailers can also offer blockchain-powered applications for the end-user. It would help the end-user place the order smoothly without any hassles.

▶ **Step 6**

- It's the last step of the blockchain for the supply chain, and the end-user will be in the sole power in this one. Firstly using the app, they can track their orders from the retailer shops. Secondly, they can scan the QR code to know whether it's authentic or not.
- Lastly, if he/she faces any issues, they can directly complain to the customer service using the app and get a refund or exchange the product. So, you see blockchain supply chain examples can really change the ways how the typical supply chain works.
- The suitable platforms for developing blockchain-based applications for Supply chain management are Hyperledger, Enterprise Ethereum, Corda, Quorum, etc. There are various organizations namely Unilever, Walmart, SAP, Ford, Rolls Royce, and Airbus, etc. using blockchain for the supply chain management.
- Supply chain is a very important process for any organization delivering their product or services to the end-user. The blockchain technology used in the Supply chain provides a transparent ledger, real-time tracking, and faster transection with high security and low cost. The process of supply chain looks very simple and straight forward but involves various issues, and many of them can be eliminated by applying blockchain technology.

Chapter Ends...

❑❑❑

Module 6

Tools and Applications of Blockchain

Syllabus

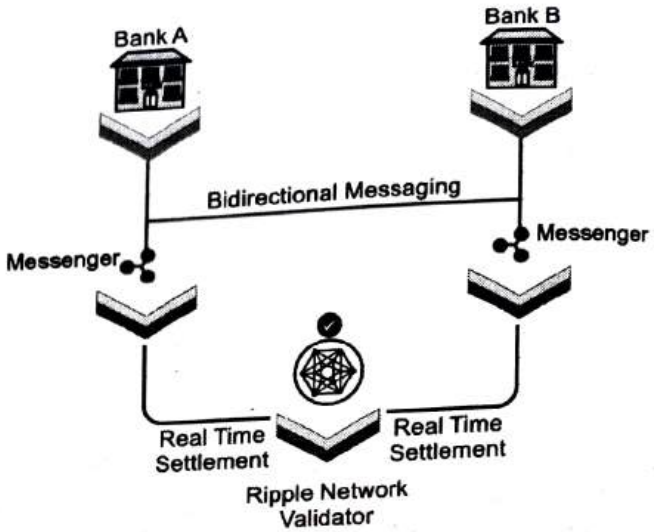
Corda, Ripple, Quorum and other Emerging Blockchain Platforms, Blockchain in DeFi: Case Study on any of the Blockchain Platforms.

6.1 OVERVIEW OF RIPPLE

GQ. What is ripple? Explain its payment system.

- Ripple is a real-time gross settlement system, currency exchange, and remittance network created by Ripple Labs and released in 2012.
- It is a blockchain-based digital payment network for financial transactions and protocol with its own cryptocurrency, i.e., XRP. Ripple is the name of the company and the network, and XRP is the cryptocurrency token.
- Ripple solves the issue of high cost and longer time to settle international transactions of the existing system used for money transfers and used by banks and financial institutions dealing with currencies across the world.
- With the features of low power and low cost compared to bitcoin and quick completion of transactions in seconds, it provides an easy, efficient, and instant transfer of money at low cost across the world compared to the existing system. The technology is adopted by leading banks and financial institutions including JP Morgan and Bank of America.
- Ripple relies on a common shared ledger storing information about all Ripple accounts and provides a digital portal, which is used by financial institutions and banks to join the ripple network which is called as a ripple-net.
- A Ripple network consists of two groups of users, namely ripple network users and network members. The primary users of ripple networks are corporate, banks, and institutions that only make payments.
- Another group of users called network members include bank and payment providers that are the foundation of the ripple network as they process payments and liquidity.
- Anyone or any business can use the portal to register and open a gateway, which authorizes the registrant to act as the middleman for exchanging currencies, maintaining liquidity, and transferring payments on the network.
- Ripple uses these gateways as the link in the trust chain between two parties wanting to make a transaction. Gateway acts as the credit intermediary that receives and sends currencies to public addresses over the Ripple network.
- Ripple network also works as a money exchange for all types of fiat currency with the help of XRP. XRP is the cryptocurrency that is built for providing sources of liquidity to payment providers and banks, which makes it easy for any currency to be exchanged for another. Each currency on the network has its own gateway, e.g., CAD Bluzelle, BTCbitstamp, and USDsnapswap.
- If anyone wants bitcoins as payment for the services and payer can make payment in Canadian dollars (CAD), then it is not mandatory for the payer to have possession of bitcoins. Instead, the payer can send the payment to the gateway in CAD, and the service provider can receive bitcoins from his/her gateway.
- Instead of a single gateway to initiate a complete transaction, multiple gateways are used, forming a chain of trust rippling across the users.
- Anyone including universities, exchanges, and even financial institutions can become a validator that cryptographically confirms success or failure of each payment and coordinates movement of funds across ledgers of transacting parties. Ripple uses a consensus mechanism to confirm transactions and also prevents double-spending and improves the integrity of the system.
- Transferring money with a gateway exposes the user to counterparty risk, which is also present in the traditional banking system. If the gateway is not trustworthy, then the user could lose the value of their money held at that gateway. In such situations, users must transact with a trustworthy gateway to deal with the counterparty risk.
- This way the "I owe you" IOU will be safely transacted through the trusted gateway. Ripple also records all IOUs in a given currency for users or gateway. IOU credits and transactions occurring between Ripple wallets are publicly available on the Ripple consensus ledger.

- It records financial transaction history publicly, and it is made available on a blockchain, but these details are not made available to any individual or business. However, the record of all dealings on the blockchain makes the information susceptible to de-anonymization measures.
- The Ripple system is revolutionary technology for transferring large amounts of money across the world and can handle large amounts of transactions (~1500/second). It is useful to large corporations like Amazon or Apple who spend millions around the world. XRP is created with a total supply of 100 billion, and ~45 billion are in circulation and remaining are held by the Ripple labs.
- Most of the banks signed up to RippleNet were using Ripple's XCurrent to process the transaction and avoid the XRP cryptocurrency due to its volatility problems.
- xCurrent is a messaging technology that enables banks to message and settle their transactions with increased speed, transparency, and efficiency with RippleNet members. xCurrent is a messaging module that facilitates bidirectional communication between banks connected on RippleNet.



(1D6)**Fig. 6.1.1 : Ripple payment system**

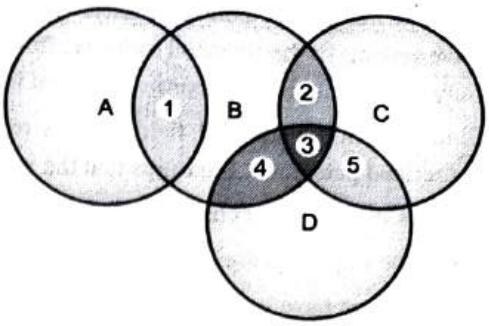
- Messenger is used to exchange information regarding risk and compliance, fees, FX rates, payment details, and expected time of funds delivery during the transaction processing. It is built around ILP, which is an open, neutral protocol that allows interoperation between different ledgers and payment networks. The solution provides a secure, end-to-end payment flow with transaction immutability and information redundancy.

6.2 OVERVIEW OF CORDA

GQ. How contract sharing takes place in Corda?

- Corda is an open source DLT developed by R3.It is a permissioned-type blockchain platform that facilitates businesses to transact value directly under strict privacy with one another. Corda is supported by world's largest technology and organizations like Amazon web services (AWS), Intel, Microsoft, and world's largest banks.
- Corda was initially designed for financial institutions only, but later it opened for others business with an aim of bringing all types of businesses on a single global distributed ledger. Corda is designed considering privacy as its prime requirement.
- In Corda, privacy is implemented in such a way that only parties involved in the transaction have access to the details of transactions. In other words, with Corda, two or more parties can perform transactions amongst themselves and share necessary details only to the parties involved in the transaction instead of broadcasting to other members in the network.
- These transactions are recorded on a ledger legally binding all the parties involved in the contract, which can be produced as an evidence in case of a dispute. Corda provides assured identity of participants on the network with the help of its KYC requirement and identity management features.
- Corda partnered with Microsoft to provide its services with Azure, and it is also accessible from AWS to use. Messaging in Corda is similar to the email system, and its privacy can be understood with the example given in Fig. 6.2.1, which is representing different users and the shared contract. There is no single central storage, and contracts are not shared with all the nodes in the system.

- Let us consider the example of users B, C, and D who are sharing a common contract 3, which is unavailable to user A, and contract 5 is shared between users C and D, and communication related with contract 5 is not shared with remaining users A and B.
- Consider the application where the seller of a product does not want to reveal the identity of its consumers to its competitor. In such scenarios, sellers can perform transactions without sharing details to other parties ,which are not involved in the transaction.
- Corda is scalable and supports billions of transactions daily across the industry and allows interoperability amongst multiple applications coexisting together on the network.



(107)**Fig. 6.2.1 : Cordacontract sharing**

- Corda is architected in such a way that different versions of corda can coexist on the same network, and applications will continue running on the later versions. Corda contracts involve external facts and conditions governing contracts like share price, interest rate, FX conversion, etc.
- These details are kept off-chain and provided by trusted oracles. Consensus pool (notary node) in the Corda architecture provides consistent, transparent, and resilient unique consensus services, and Corda's UTXO state machine model prevents the double-spending problem.
- Corda allows businesses and individuals to perform transactions privately concerning only members involved without broadcasting over the network. The parties involved in the transactions are legally identifiable on the network, which is highly scalable. Developers on Corda use network wide standards to develop interoperable applications.
- Similar to other blockchain networks, in Corda, transactions are cryptographically chained to the transactions it depends upon, but it does not permit global broadcast instead it prefers peer-to-peer message sending and avail information of transaction to authorized parties whose assured identity is maintained. All messages on Corda are encrypted, and it maintains a global ledger including all transactions.

- Quorum is an Ethereum-based distributed ledger protocol that allows for the creation of a permissioned blockchain network with transaction privacy support. J.P. Morgan Chase created the open-source project with a focus on financial use cases; it was recently acquired by ConsenSys.
- Quorum was introduced into the larger ecosystem of permissioned blockchains to address the following requirements:
  - Carry out private transactions as well as smart contract operations.
  - Implement multiple consensus mechanisms in a plug-and-play manner.
  - Manage network permissions in a flexible and expressive manner.
- Quorum is modelled after the official Go implementation of the Ethereum protocol (geth).The reasoning behind this choice is that by minimizing changes to Go-Ethereum, future versions of the public Ethereum code base can be easily maintained. Each Quorum node provides two primary services.

1. **Quorum Client :** This is the extended geth client that is in-charge of executing the Ethereum peer-to-peer (p2p) protocol (by accepting connections only from permissioned network participants) and the consensus algorithm.
2. **Privacy Manager :** It is a software module that allows for private transactions and private smart contract operations. It is made up of two parts, i.e., the transaction manager and the crypto enclave. Because of its cryptographically secure, auditable, and immutable characteristics, blockchain or distributed ledger provides a secure, shared platform for decentralized applications (Dapps) and data. However, for a blockchain to be suitable for enterprises, several enterprise-driven requirements must be met. Privacy, performance, and permissioning are the most important of these requirements.

- Ensures transaction confidentiality: It is a requirement in many industries, including finance, health care, law, and government. In the financial industry, for example, it is required that transaction details be kept confidential and shared only with the authorized parties involved in the transaction.
- Similarly, in the medical field, patient records are extremely sensitive information that should only be accessed by authorized personnel. It ensures that the network's speed and scalability are adequate for handling enterprise use cases.
- Ensures that the blockchain network is only accessible to authorized entities.
- All of the requirements listed above are critical in any enterprise use case.
- Now that we have covered the basics of enterprise features, let us take a look at how Quorum achieves them. First, we will go over the Quorum architecture.

6.3 QUORUM ARCHITECTURE

GQ. Explain Quorum Architecture in detail.

Quorum, in comparison to public Ethereum, offers several enterprise features, which are listed below.

- Transaction privacy
- A variety of pluggable consensus mechanisms appropriate for enterprise use cases
- Enterprise-grade permissions management (access control) for network nodes and participants
- Quorum is essentially a public Ethereum client that has been enhanced with enterprise features. In a permissioned network, it provides privacy features, enterprise permissioning, and improved performance. Off-chain privacy is provided by a component called private transactions manager. Quorum communicates with the private transaction manager over HTTPS and maintains a reference to private transactions on the blockchain with relevant state trees.
- The figure below depicts the high-level architecture.

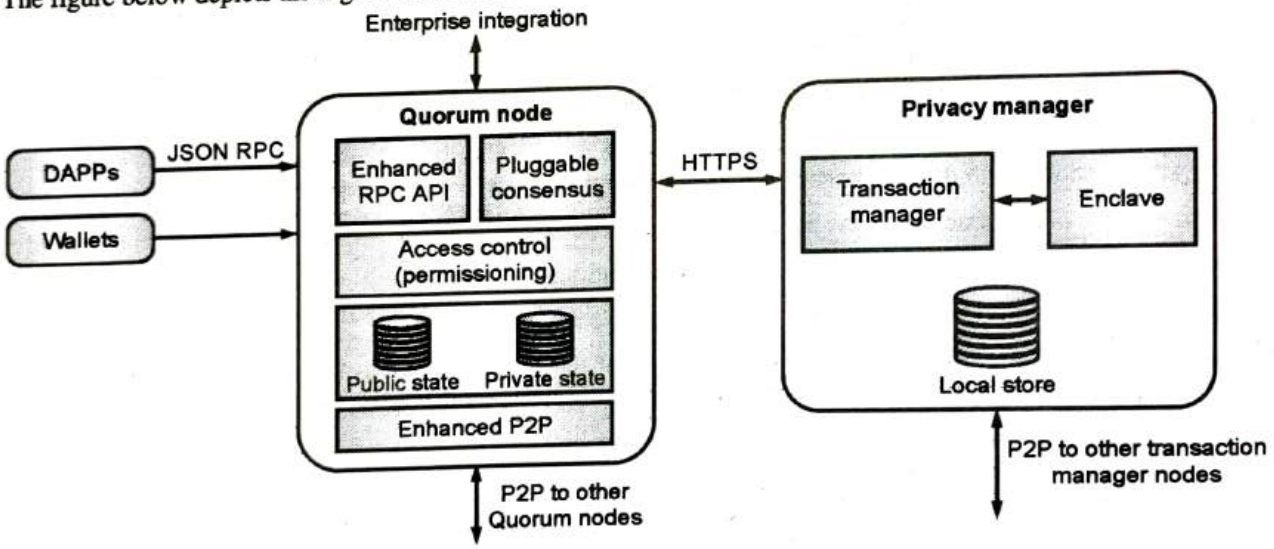


Fig. 6.3.1 : Quorum high-level architecture

Quorum node

The Quorum node is a small fork of geth. Because it is a fork of geth, it continues to benefit from the ongoing research and development within the ever-expanding Ethereum community as well as the incredible work of the geth development team. As a result, Quorum is regularly updated in tandem with Geth releases to reflect the most recent enhancements.

The Quorum node differs from the public geth client in the following ways:

- Instead of using PoW, the consensus is achieved using RAFT, proof-of-authority (PoA), or Istanbul BFT consensus algorithms. Because all of these different protocols are available, it is possible to select any of these algorithms based on business requirements.
- The p2p layer has been modified to allow connections only to and from permissioned nodes.
- The block generation logic has been changed to replace the'global state root' check with a new'global public state root' check.
- The State Patricia trie has been divided into two parts, i.e., public and private.
- The block validation logic has been changed to replace the phrase "global state root" in the block header with "global public state root."
- The block validation logic has been changed to accommodate 'Private Transactions.'
- Transaction creation has been modified to allow transaction data to be replaced by hashes of encrypted payloads where necessary to preserve private data.
- The Gas pricing has been removed, but the Gas itself remains.

Quorum is compatible with both public and private transactions. Public transactions operate normally, as they do in public Ethereum, whereas private transactions are enabled by a separate component known as the private transaction manager (privacy manager).Now that we have introduced the Quorum node, let's look at the privacy manager.

Privacy manager

On the Quorum network, the privacy manager component (private transaction manager) is in-charge of ensuring transaction privacy. In other words, this component enables Quorum nodes to securely share transaction payload between authorized transaction parties. It is made up of two parts, i.e., the transaction manager and the enclave.

Transaction manager

It is a restful and stateless service that is primarily in-charge of the following operations:

- Automatically discovers other transaction manager nodes on the network
- Exchanges encrypted payloads with transaction managers on other nodes
- Stores and allows access to encrypted transaction data

There are currently two types of transaction managers available. i.e., ConstellationSM and TesseraSM. Constellation is the original Haskell privacy manager. It is no longer being developed in favor of Tessera, a more feature-rich and actively developed project. As a result, this report focuses solely on Tessera. The transaction manager is a general-purpose mechanism for securely exchanging data. It is analogous to a network of message transfer agents (MTAs) in which pretty good privacy (PGP) provides message encryption. The private transaction manager is a non-blockchain technology. It can be used in any application that requires individually sealed and secure message exchange across a network of participants.

Tessera

- Tessera is a transaction manager for enterprise. It is a Java-based stateless software that allows Quorum to encrypt, decrypt, and distribute private transactions.
- A Tessera node is responsible for the following tasks:
  - Generates and hosts a large number of public/private key pairs.
  - Discovers all nodes on the network (i.e., their public keys) automatically by connecting to as few as one other node.
  - Supports various trust models such as trust on first use (TOFU), IP whitelists, and certificate authorities.
  - It can connect to any SQL database that supports the JDBC client.

- Synchronizes a directory of public keys associated with recipient hosts with other network nodes.
- Provides a public API for communication between Tessera peer nodes.
- Provides a private API for communicating with Quorum nodes
- Allows you to send a byte string to one or more public keys and receive a content-addressable identifier in return. Before being transmitted over the wire to the correct recipient nodes, this byte string is encrypted transparently and efficiently (at symmetric encryption speeds) (and only those nodes). The identifier is a hash digest of the encrypted payload received by each recipient node. In addition, each recipient node receives a small blob encrypted for their public key, which contains the master key for the encrypted payload.
- Enables the reception of a decrypted payload based on an identifier. Payloads sent or received by a node can be decrypted and retrieved in this manner.
- Supports a variety of storage backends, including Level DB, Berkeley DB, SQLite, and Directory/Maildir style file storage that can be used with any Filesystem in Userspace - FUSE adapter, for example, for AWS S3.
- Tessera can be thought of as a hybrid of a distributed key server, PGP encryption (using modern cryptography), and mail transfer agents (MTAs).

Enclave

- Cryptographic techniques are commonly used in distributed ledger protocols for transaction authentication, participant authentication, and historical data preservation (i.e., through a chain of cryptographically linked data). To achieve "separation of concerns," the Enclave handles the majority of cryptographic operations, such as symmetric key generation and data encryption/decryption. As a result of this separation, security is improved through modularization, and performance is improved through parallelization of certain cryptographic operations.
- The Enclave collaborates with the transaction manager to improve privacy by independently managing cryptography operations. It stores private keys and functions as a "Virtual HSM" separate from the rest of the system. An enclave only communicates with its own transaction manager. The following information is handled by the enclave:
  - Public/Private key access
  - Extra recipients' public keys
  - Public/Private key access
- The following are the specific operations that an enclave performs:
  - Obtaining the attached node's default identity (default public key)
  - Providing forwarding keys for all transactions
  - Returning all public keys managed by the enclave
  - Obtaining the attached node's default identity (default public key)
  - Providing forwarding keys for all transactions
  - Returning all public keys managed by the enclave
  - Encrypting a payload for a given sender and recipient(s)
  - Encrypting raw payloads for a given sender
  - Decrypting transactions for a given recipient (or sender)
- Now that we've covered the high level architecture of Quorum, let's look at how it achieves all of the enterprise features we discussed earlier.

Private transactions

- We'll look at how Quorum supports key enterprise features like privacy, performance, and permissioning.

- As previously stated, private transactions in Quorum are supported by an off-chain mechanism known as the Privacy transaction manager. We will now describe how the privacy transaction manager works to enable private transactions with an end-to-end example. This example demonstrates how all components of Quorum collaborate to provide privacy features.
- Before we get into the example, keep in mind that Quorum not only supports private transactions but also standard public transactions. All transactions, as usual, must be signed by the sender. Quorum has two transaction signing mechanisms. The Ethereum EIP-155 based transaction signing mechanism is used for public transactions, and the Ethereum Homestead based transaction signing mechanism is used for private transactions. Furthermore, Quorum supports raw private transactions, which means that transactions can be signed externally without the use of Quorum's signing mechanism. This feature provides more flexibility and security.
- Consider three parties, A, B, and C. A and B are aware of a transaction known as 'AB,' but C is not. We now look at the transaction flow from the view of each of these parties.

View of Parties A & B

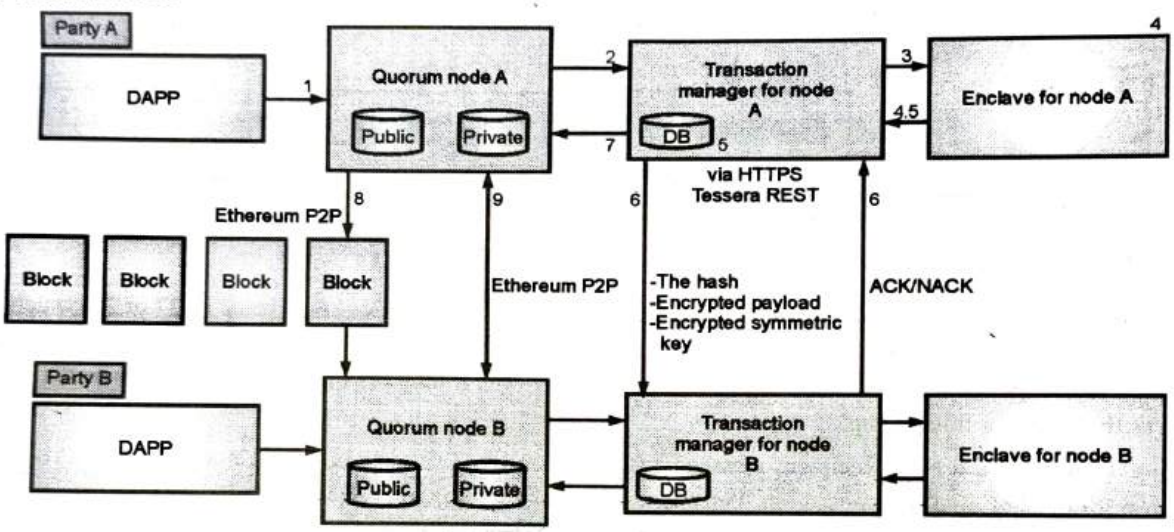


Fig. 6.3.2 : Parties A & B

Looking at the above figure, we can describe the process step by step as presented below:

1. Party A sends a transaction to their Quorum node, indicating the transaction payload and specifying private for to be Party B's public keys. It is optional for party A as well.
2. Party A's Quorum node sends the transaction to its paired transaction manager, requesting that the transaction payload be stored.
3. The transaction manager of Party A contacts the associated enclave to validate the sender and encrypt the payload.
4. Party A's enclave validates Party A's private key and, if valid, processes the transaction.
5. Party A's transaction manager computes the encrypted payload's SHA3-512 hash and stores the encrypted payload and encrypted random master keys (RMKs) against the hash in the database.
6. The transaction manager of Party A then securely transfers (via HTTPS):
   - The encrypted payload
   - RMK encrypted with the shared key generated by the enclave processing in Step 4.
   - The nonces to Party B's transaction manager.

- It should be noted that if Party A does not receive a response/a negative acknowledgement (NACK) from Party B, then the transaction will not be propagated to the network. It is required that the communicated payload be stored by the recipients.

1. After successfully transmitting data to Party B's transaction manager, Party A's transaction manager returns the hash to the Quorum node, which replaces the transaction's original payload with that hash. It also changes the V value of the transaction to 37 or 38. This value indicates to other nodes that this hash represents a private transaction with an encrypted payload rather than a public transaction with nonsensical bytecode.
2. The transaction is then broadcast to the rest of the network via the Ethereum P2P protocol.
3. A block containing transaction 'AB' is created and distributed to all network parties.

View of party C

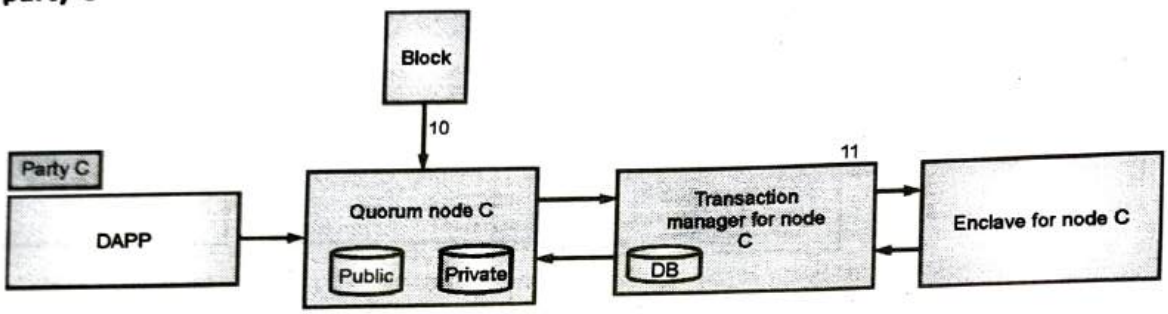


Fig. 6.3.3 : Party C

- When the block is processed, all parties will attempt to process the transaction. A V value of 37 or 38, which identifies the transaction as a private transaction whose payload requires decryption, will be recognised by each Quorum node. The node then queries their associated transaction manager to see if they are the owner of the transaction. The hash is used as the index for this lookup.
- Party C will receive a NotARecipient message and will skip the Transaction. It will not update its Private StateDB because it does not own the transaction. Party A and B will look up the hash in their respective transaction managers and discover that they do indeed own the transaction. After that, each transaction manager will make a call to its paired enclave, passing in the encrypted payload, encrypted symmetric key (RMK), and signature.

View of Party B

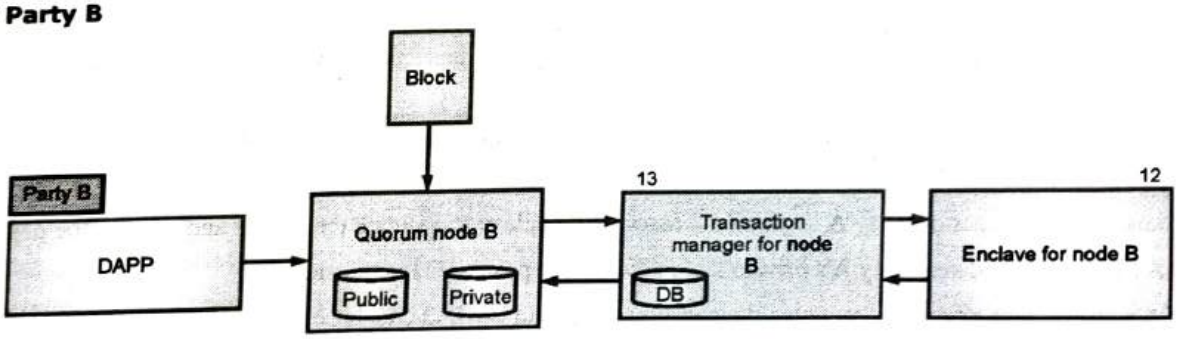


Fig. 6.3.4 : Party B

1. The enclave validates the signature, then decrypts the symmetric key with the party's private key stored in the enclave, decrypts the transaction payload with the now-revealed symmetric key, and returns the decrypted payload to the transaction manager.
2. The decrypted payload is then sent to the EVM by the transaction managers for parties A and B for contract code execution. This execution will only update the Quorum node's Private StateDB.

What happens inside the enclave?

- Now we expand on Step 4 from above, which involves enclave processing.

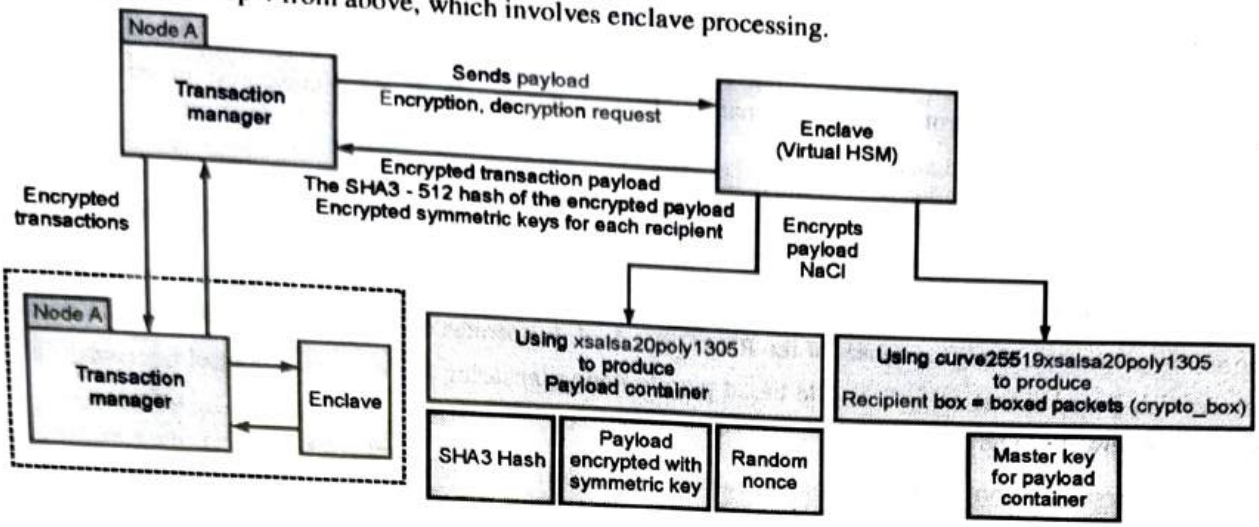


Fig. 6.3.5 : Enclave processing

- "Party A's Enclave validates Party A's private key and, if validated, processes the transaction."
- This procedure consists of several steps:
  1. Create a random master key (symmetric key) and nonce.
  2. Use the symmetric key generated in step 1 to encrypt the transaction payload. The payload container is created using the authenticated encryption algorithm xsalsa20poly1305. It is based on the Salsa20 stream cypher and the poly1305 universal hash function. The 'crypto box' is created with a public-key authenticated-encryption scheme that combines three constructs: Curve25519, XSalsa20, and Poly1305.3.1.
  3. Compute the hash (SHA3 - 512 bit) of the previously encrypted payload.
  4. Using the recipient's public key to encrypt the symmetric key from Step 1. This procedure is repeated for each recipient one at a time. In our example, it only applies to Parties A and B.
  5. Enclave returns three objects to the transaction manager.
     - Encrypted transaction payload from Step 2
     - Hash from Step 3
     - Encrypted symmetric keys for each recipients from Step 4
- Other Elliptic curves are also supported by Tessera for the creation of public/private key pairs as well as data encryption and decryption. Tessera also works with external hardware security modules (HSMs) and cloud-hosted key management. Tessera's support for external key vault integration with third-party key vaults such as Azure, Hashicorp, and AWS is a notable feature. This feature enables fully decoupled and dependable key management.

Enterprise-grade performance

- Quorum includes several appropriate consensus mechanisms for enterprise networks. When compared to a typical PoW mechanism on public blockchains like Bitcoin and Ethereum, these consensus algorithms provide immediate finality and higher transaction throughput.
- An independent performance evaluation study reported a transaction per second (TPS) speed of approximately 2500 TPS.
- In another study, the transaction throughput of private contract deployments was measured to be approximately 700 TPS, and the performance of normal transactions was measured to be approximately 2000 TPS.

- Quorum is a good choice for enterprise use cases because of its improved performance.

Enterprise permissioning mechanism

- The role based access control (RBAC) mechanism is a common and standard enterprise grade scheme used for providing organisational level access control.
- It is a de facto mechanism for providing a common enterprise grade access control mechanism to enterprise systems. It is used in a lot of business systems. RBAC is also implemented in operating systems such as Windows and RedHat, demonstrating its industry-wide acceptance and usability.
- Quorum implements a modified subset of the RBAC standard. It operates on the same principles as standard RBAC. It provides role-based access as well as rule-based permissioning, ensuring the necessary control over who can join the network and how it can be operated.
- To better understand Quorum's permissioning features, we must first define some terminology.
  - **Network :** A collection of interconnected nodes that represent an enterprise blockchain.
  - **Organization :** A collection of roles, Ethereum accounts, and nodes with network access control permissions.
  - A sub-organization is a group within a larger organisation.
  - **Account :** An Ethereumexternally owned account (EOA)
  - **Voter :** An account with voting privileges.
  - **Role :** A specific job function within an organisation.
  - **Node :** A geth node that is a member of the network and belongs to an organisation or a sub-organization.
  - **Permission :** A description of the types of actions that an account is permitted to carry out. (*for example, value transfer, smart contract deployment, or smart contract execution*)

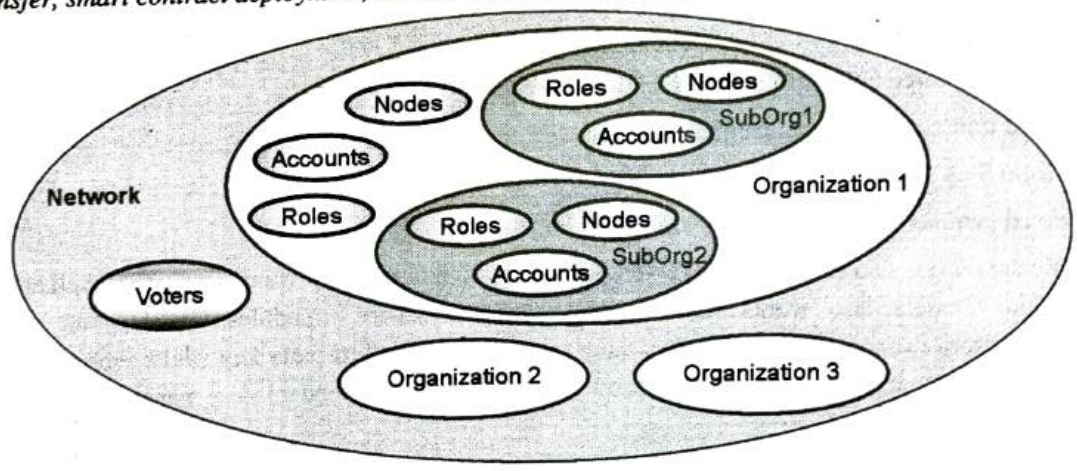


Fig. 6.3.6 : Quorum permissioning mechanism

- The quorum permissioning mechanism is implemented using smart contracts and some client software changes. As a result, this model is divided into two parts, the first of which deals with the access control decision output, which represents a decision about whether or not an account is permitted to perform a function.
- This is the 'enforcement logic,' and it is implemented in the Quorum client software. The other component is in charge of managing the underlying rules associated with the permissioning logic. Based on the roles assigned to an entity, this "rule engine" generates an access control decision.

- It defines what an object can and cannot do on a blockchain network. This is referred to as 'policy management.' This component is fully implemented using solidity language smart contracts. Both of these components work together to form the Quorum permissioning mechanism.
- The quorum permissioning mechanism is currently used in conjunction with the RAFT, IstanbulBFT, and PoA consensus mechanisms. The network in the Quorum permissions model is made up of various organisations, as illustrated in Fig. 6.3.7. The network administrator account(s) defined at the network level have the authority to propose and approve new organisations requesting to join the network. They can also grant administrative privileges to an account to serve as the administration account for an organisation.
- The organisation administrator account can perform the following functions.
  - Create new roles
  - Create sub organisations
  - Assign roles to its domain accounts
  - Add new nodes to the organisation.
- A sub-organization can also have its own set of roles, accounts, and sub-organizations. The organisation administration account manages and controls all organisational activities.
- To allow a different account to administer the sub organisation, the organisation administrator can create an administrator role and assign it to that account.
- The role assigned to an account determines its access rights. An administrator role, for example, can execute a smart contract, whereas a trainee role can only read. An account that exists at the organisational level can transact through any node that exists in the sub-organizations beneath or at the top-level organisations.

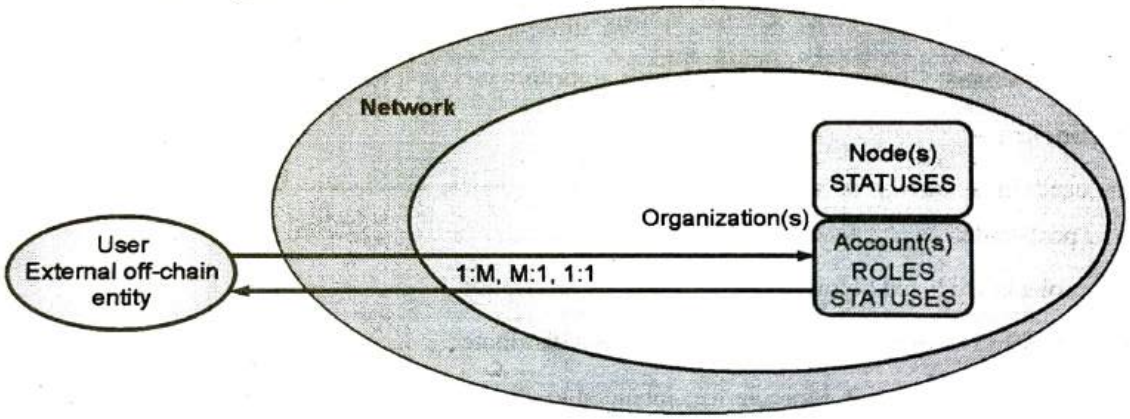


Fig. 6.3.7 : Relationship between different entities in Quorum permissioning model

- This architecture is depicted in Figure 7, where the network is the top level entity, containing an organisation or several organisations, each of which contains accounts and nodes as well as relevant access and status types.
- Also, keep in mind that the user is an external entity that exists outside of the network. It could be a company or a single user. The key concept here is that in the permissioning mechanism, a user is mapped to Ethereum accounts. It can be a one-to-many, many-to-one, or one-to-one relationship. A single organisation, for example, represented by a username, can be linked to multiple accounts on the blockchain.
- Similarly, multiple external entities can be represented on the chain by the same account. A single user can be assigned to a single account. The benefit of this approach is that there is no need to keep on-chain records of users, which not only incurs high storage costs but is also inappropriate for privacy reasons.

- Accounts are assigned roles and statuses based on their business function and access level, whereas nodes are assigned statuses that represent their network access level.
- Statuses can also be assigned to organisations. This feature is especially useful when an entire organisation needs to be assigned a network status, such as when an organisation leaves the network.
- The administrator can simply assign suspended status to the organisation, which will apply to all entities within that organisation, including accounts and sub organisations.

Consensus

- There is no need for an expensive PoW consensus mechanism because consortium chains are permissioned.
- Furthermore, due to performance constraints, slower public chain consensus mechanisms are unsuitable for consortium chains. As a result, Quorum provides various consensus mechanisms that are better suited for private blockchains.
- These mechanisms are detailed further below.
  - **RAFT-based Consensus:** A crash fault tolerant (CFT) consensus model for faster block generation, transaction finality, and on-demand block generation.
  - **Istanbul BFT Convergence:** It is a BFT algorithm that is based on the Practical Byzantine Fault Tolerant (PBFT) consensus algorithm. It allows for immediate transaction completion. It guarantees liveness and safety under standard Byzantine fault threshold assumptions of $\frac{n-1}{3}$ in a partially synchronous network and 3f + 1 in a 3f + 1 network configuration.
  - **Clique Consensus:** Clique is a POA consensus algorithm included with the public Go Ethereum client (geth).

Use cases of Quorum

- Quorum is used in a variety of applications, including logistics, healthcare, identity, property, payments, capital markets, and post-trade.
- Some of the projects are listed below, along with a brief description:
  1. Tokenised cash: Developed by IHS Mark it, this is a distributed ledger that records all cash transactions.
  2. JPM Coin®: Created by J. P. Morgan, it is intended to allow for the immediate settlement of transactions between clients of the bank's wholesale payments business.
  3. Loan Marketplace: It is a decentralised loan marketplace. Stream Source technologies created it.
  4. Proxy voting increases transparency in AGM voting between issues and investors.
  5. Post-trade processing platform: An oil trading post-trade processing platform.
  6. Interbank Information Network® (IIN): IIN enables network members to exchange information in real time to verify payments.
  7. Supply chain tracking: A cryptographic provenance platform that allows the authenticity of high-priced goods to be proven.
  8. Physical tracking of gold bars and title deed registration Med ledger : Counterfeit medicine detection.

Comparison of best blockchain platforms used by companies for building blockchain-based applications

	XDC Network	Ethereum	Hyperledger Fabric	R3 Corda	Ripple	Quorum	Hyperledger Sawtooth	EOS	Hyperledger Iroha	Openchain	Stellar
Industry focus	Cross-Industry	Cross-Industry	Cross-Industry	Financial Services	Financial Services	Cross-Industry	Cross-Industry	Cross-Industry	Cross-Industry	Digital Asset Management	Financial Services
Ledger Type	Permissionless	Permissionless	Permissioned	Permissioned	Permissioned	Permissioned	Permissioned	Permissioned	Permissioned	Permissioned	Both Public & Private
Consensus Algorithm	XDC Delegated Proof-of-Stake	Proof of Work	Pluggable Framework	Pluggable Framework	Probabilistic Voting	Majority Voting	Pluggable Framework	Delegated Proof-of-Stake	Chain-based Byzantine Fault Tolerant	Partionned Consensus	Stellar Consensus Protocol
Smart Contract	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Governance	XDC Network	Ethereum Developers	Linux Foundation	R3 Consortium	Ripple Labs	Ethereum Developers and JP Morgan Chase	Linux Foundation	EOSIO Core Arbitration Forum(ECAF)	Linux Foundation	CoinPrism	Stellar Development Foundation

6.4 BLOCKCHAIN FOR DECENTRALIZED FINANCE (DEFI)

GQ. Write a short note on DeFi.

What is Decentralized Finance (DeFi)?

- Decentralized finance, also known as DeFi, refers to the transition away from traditional, centralized financial systems and toward peer-to-peer finance enabled by decentralized technologies built on the Ethereum blockchain. The DeFi ecosystem has launched a vast network of integrated protocols and financial instruments, ranging from lending and borrowing platforms to stablecoins and tokenized BTC.
- DeFi has emerged as the most active sector in the blockchain space with over $13 billion in value locked in Ethereum smart contracts and a wide range of use cases for individuals, developers, and institutions.
- Whereas our traditional financial system is based on centralized infrastructure that is managed by central authorities, institutions, and intermediaries, decentralized finance is powered by code that runs on the Ethereum blockchain's decentralized infrastructure. DeFi developers can launch financial protocols and platforms that run exactly as programmed and are accessible to anyone with an Internet connection by deploying immutable smart contracts on Ethereum.
- DeFi's breakthrough is that crypto assets can now be used in ways that fiat or "real world" assets cannot. Decentralized exchanges, synthetic assets, and flash loans are entirely new applications that can exist only on blockchains. This paradigm shift in financial infrastructure offers several advantages in terms of risk, trust, and opportunity.

6.4.1 Features of DeFi

DeFi can be viewed as the most realistic use of blockchain because it incorporates its features, such as:

1. **Eliminate third parties :** DeFi entirely eliminates third parties, eliminating the need for a bank, financial institution, or other middleman like traditional finance, allowing customers to keep ownership of their money. Any disagreements will be settled in a predetermined way.
2. **Smart contract :** Indisputable logic code on the blockchain network, as opposed to human intervention, is used in a general way of eliminating third parties. Smart contracts frequently help DeFi.
3. **Automation:** Utilize smart contracts to streamline the procedure while maintaining each contract's responsiveness and accuracy.
4. **Cost savings :** Cutting costs in the role of lawyers, personnel to perform contracts.
5. **Autonomy :** Once involved, stakeholders are only bound by the terms they have accepted in the smart contract and not to any other external authority. Additionally, because of this feature, neither side may modify them, guaranteeing the safety and security of contractual agreements.

What are the benefits of Decentralized Finance?

DeFi uses key Ethereum blockchain principles to improve financial security and transparency, open up liquidity and growth opportunities, and support an integrated and standardised economic system.

- **Programmability :** Smart contracts with high programmability automate execution and enable the creation of new financial instruments and digital assets.
- **Immutability:** Tamper-proof data coordination across the decentralized architecture of a blockchain increases security and auditability.

- **Interoperability:** The modular software stack of Ethereum ensures that DeFi protocols and applications are designed to integrate and complement one another. Developers and product teams can build on top of existing protocols, customise interfaces, and integrate third-party applications with DeFi. As a result, DeFi protocols are frequently referred to as "money legos."
- **Transparency :** Every transaction on the public Ethereum blockchain is broadcast to and verified by other network users (note: Ethereum addresses are encrypted keys that are pseudo-anonymous). This level of transparency in transaction data not only enables rich data analysis, but also ensures that network activity is accessible to all users. Ethereum and the DeFi protocols that run on it are also built with open source code that anyone can view, audit, and extend.
- **Permissionless :** Unlike traditional finance, DeFi is distinguished by its open, permissionless access: anyone with a crypto wallet and an Internet connection, regardless of location and frequently without a minimum amount of funds required, can access DeFi applications built on Ethereum.
- **Self-Custody :** DeFi market participants maintain custody of their assets and control of their personal data by interacting with permissionless financial applications and protocols via Web3 wallets such as MetaMask.

☞ What are the use cases for DeFi?

- Decentralized finance protocols, ranging from DAOs to synthetic assets, have opened up a world of new economic activity and opportunity for users all over the world. The extensive list of use cases provided below demonstrates that DeFi is much more than an emerging ecosystem of projects. Rather, it is a comprehensive and integrated effort to build a parallel financial system on Ethereum that rivals centralized services in terms of accessibility, resilience, and transparency.

☞ Asset management

- You are the custodian of your own crypto funds with DeFi protocols. Crypto wallets such as MetaMask, Gnosis Safe, and Argent enable you to interact with decentralized applications to do everything from buying, selling, and transferring crypto to earning interest on your digital assets in a simple and secure manner. You own your data in the DeFi space: MetaMask, for example, stores your seed phrase, passwords, and private keys locally on your device in an encrypted format so that only you have access to your accounts and data.
- The rules of the game have changed for organisations that have increased institutional-grade requirements for allocating capital to DeFi. Wallets like MetaMask Institutional make cryptoeconomic research, pre- and post-trade compliance, best trade execution, reporting, and, of course, crypto custody easier for these organisations.

☞ Compliance and Know-Your-Transaction (KYT)

- In traditional finance, anti-money laundering (AML) and counter-financing-of-terrorism (CFT) compliance are based on know-your-customer (KYC) guidelines.
- In the DeFi space, Ethereum's decentralized infrastructure allows for next-generation compliance analysis based on participant behaviour rather than participant identity.
- These KYT services, such as those provided by MetaMask Institutional, assist in real-time risk assessment and protect against fraud and financial crimes.

☞ DAOs

- A DAO is a decentralized autonomous organisation that cooperates in accordance with transparent rules encoded on the Ethereum blockchain, obviating the need for a centralized, administrative entity.

- Several popular DeFi protocols, including Maker and Compound, have established DAOs to raise funds, manage financial operations, and decentralise governance to the community.

☞ Data and analytics

- DeFi protocols provide unique advantages for data discovery, analysis, and decision-making around financial opportunities and risk management due to their unprecedented transparency around transaction data and network activity.
- The rapid development of new DeFi applications has resulted in the creation of a plethora of tools and dashboards, such as DeFi Pulse, that assist users in tracking the value locked in DeFi protocols, assessing platform risk, and comparing yield and liquidity.

☞ Derivatives

- Ethereum-based smart contracts allow for the creation of tokenized derivatives, the value of which is determined by the performance of an underlying asset and in which counterparty agreements are hardcoded in code.
- DeFi derivatives can represent both real-world assets and cryptocurrencies, such as fiat currencies, bonds, and commodities.

☞ Developer and Infrastructure Tooling

- Composability is one of the core design principles of DeFi protocols, which means that different components of a system can easily connect and interoperate. As evidenced by the wide range of integrated DeFi applications, composable code has created a powerful network effect in which the community builds on the work of others.
- Many people compare DeFi development to building with legos, hence the increasingly popular moniker "money legos." Ethereum developers and product teams can now build and launch DeFi protocols with the full-stack tooling and security integrations they require, thanks to Truffle's smart contract libraries, Infura's API suite, and Diligence's security tools.

☞ DEXs

- DEXs are cryptocurrency exchanges that operate without a central authority, allowing users to transact peer-to-peer while maintaining control over their funds. Because crypto assets are never in the custody of the exchange, DEXs reduce the risk of price manipulation, as well as hacking and theft.
- DEXs also provide token projects with liquidity that often rivals that of centralized exchanges, with no listing fees. Until recently, projects would pay millions of dollars to have their token listed on a centralized exchange.
- Some exchanges use varying degrees of decentralization, with centralized servers hosting order books and other features but not storing users' private keys. AirSwap, Liquality, Mesa, Oasis, and Uniswap are some of the most popular DeFi DEXs right now. MetaMask Swaps, a DeFi liquidity data aggregator, optimises trading experiences by providing DeFi users with unparalleled insight, allowing them to identify the best price quote, optimal gas prices for the given quote, and the lowest failure rates.

☞ Gaming

- DeFi's modularity has opened up opportunities for product developers to integrate DeFi protocols directly into platforms across a wide range of industries. Because of their built-in economies and innovative incentive models, Ethereum-based games have become a popular use case for decentralized finance.
- PoolTogether, for example, is a no-loss audited savings lottery that allows users to buy digital tickets by depositing DAI stablecoins, which are then pooled and lent to the Compound money market protocol to earn interest.

☞ Identity

- Decentralized finance protocols combined with blockchain-based identity systems have the potential to enable previously excluded users to gain access to a truly global economic system. DeFi solutions, rather than traditional data points such as home ownership and income, can reduce collateralization requirements for people who do not have extra funds and help assess users' creditworthiness through attributes such as reputation and financial activity.
- The DeFi space values data privacy and open access to personally identifiable information. Anyone with an Internet connection can use DeFi applications while keeping control over their data and assets.

☞ Insurance

- DeFi is still a developing space, with risks associated with smart contract bugs and breaches. A number of innovative insurance alternatives have entered the market to assist users in purchasing coverage and protecting their holdings.
- Nexus Mutual, for example, offers Smart Contract Cover, which protects against unintended uses of smart contract code.

☞ Lending and Borrowing

- Some of the most popular applications in the DeFi ecosystem are peer-to-peer lending and borrowing protocols. Compound, for example, is an algorithmic, self-governing interest rate protocol that integrates with and underpins a slew of DeFi platforms, including PoolTogether,
- Argent, and Dharma. Compound allows users to earn interest on crypto that they have supplied to the lending pool by providing interest rate markets on Ethereum.
- The Compound smart contract matches borrowers and lenders automatically and calculates interest rates based on the borrowed-to-supplied asset ratio. Compound is a compelling example of the DeFi space's exponential opportunity: as more products integrate the Compound protocol, an increasing number of crypto assets will be able to earn interest even when idle.

☞ Margin Trading

- Unlike traditional finance, where margin traders can leverage their trades by borrowing funds from a broker (which then serves as collateral for a loan), DeFi margin trading is powered by decentralized, non-custodial lending protocols like Compound and dYdX.
- Because smart contracts automate traditional brokerage activity, some in the DeFi ecosystem have begun to refer to the rise of "autonomous money markets."

☞ Payments

- Peer-to-peer payment is arguably the cornerstone of the DeFi space and the blockchain ecosystem as a whole. Blockchain technology is designed to allow users to exchange cryptocurrency securely and directly with one another, eliminating the need for middlemen.
- DeFi payment solutions are helping large financial institutions streamline market infrastructure and better serve wholesale and retail customers while also creating a more open economic system for underbanked and unbanked populations.

☞ Prediction markets

- Blockchain-based prediction markets capitalise on the collective wisdom of the crowd by allowing users to vote and trade value on the outcome of events.
- Market prices then become crowdsourced indicators of an event's likelihood. Augur, a popular DeFi betting platform, includes prediction markets for election results, sports games, economic events, and other topics.

☞ **Savings**

- Many DeFi apps offer interest-bearing accounts that can earn exponentially more than traditional savings accounts by plugging into lending pool protocols like Compound, based on a dynamic interest rate tied to supply and demand.
- Popular savings apps include Argent, Dharma, and PoolTogether, a no-loss savings game in which participants win or lose all of their money.
- "Yield farming" is one DeFi activity that has exploded in response to these innovative savings mechanisms. Yield farming is the practise of moving idle crypto assets around in different liquidity protocols in order to maximise returns. The frenzy of interest in DeFi yield farming has resulted in an abundance of memes.

☞ **Stable coins**

- Any cryptocurrency that is pegged to a stable asset or basket of assets, such as fiat, gold, or other cryptocurrencies, is referred to as a stablecoin.
- Stablecoins were created in order to reduce the volatility of cryptocurrency prices and make blockchains a viable payment solution. They are now used for remittance payments, lending and borrowing platforms, and even institutional use cases such as central bank digital currency (CBDC).

☞ **Staking**

- Users will be able to stake their ETH and earn rewards as validators or through staking providers as the Ethereum network transitions to a Proof of Stake consensus algorithm with Ethereum 2.0.
- Staking on Eth2 is similar to investing in an interest-bearing savings account: stakers earn interest (rewards) for validating blocks on the Ethereum protocol.

☞ **Synthetic Assets**

- Synthetic assets, like stablecoins, are crypto assets that provide exposure to other assets such as gold, fiat currencies, and cryptocurrencies.
- Tokens locked into Ethereum-based smart contracts with built-in agreements and incentive mechanisms serve as collateral. The Synthetix protocol, for example, uses a collateralization ratio of 750 percent to help the network absorb price shocks.

☞ **Tokenization**

- Tokenization is a pillar of decentralized finance and a built-in feature of the Ethereum blockchain. Tokens not only power the network but also open up new economic opportunities.
- To put it simply, a token is a digital asset created, issued, and managed on a blockchain. Tokens are secure and instantly transferable, and they can be programmed with a variety of built-in functions.
- From fractionalized real estate security tokens to platform-specific tokens that incentivize the use of a specific application, Ethereum-based tokens have emerged as a secure and digital alternative for users all over the world to access, trade, and store value.

☞ **Trading**

- Trading in the DeFi space includes a variety of activities, such as derivatives trading, margin trading, and token swaps, and takes place across an ever-growing and integrated network of exchanges, liquidity pools, and marketplaces.
- Decentralized exchanges provide cryptocurrency traders with lower exchange fees, faster transaction settlement, and full custody of their assets.

Chapter Ends...

❑❑❑