Times asked: 6 times
5 times
4 times
3 times
2 times
1 time

# indicates 5-mark question

Blockchain Question bank

1. Introduction to Blockchain

1. Differentiate between public, private and consortium blockchain.
2. Explain Merkle tree with an example? Explain the structure of a Merkle tree.
3. State and explain various challenges that occurs while implementing blockchain.

2. Cryptocurrency

4. Write a short note on UTXO model of Bitcoin.
5. Differentiate between hot and cold wallets. #
6. Explain the concept of double spending with a suitable example. #
7. Explain mining pool and its difficulty. How is it calculated in a proof of work? Explain with an example.
8. What is cryptocurrency? Explain different types of cryptocurrencies.
9. Differentiate between PoW, PoS, PoB, PoET.
10. How does PoW solve the problem of double spending? #
11. Explain the concept of an orphaned block. #

3. Programming for Blockchain

12. Write a program in solidity to implement multi-level inheritance.
13. Explain fallback function in solidity with an example.
14. Write and elaborate a code in solidity to explain visibility and activity qualifiers.
15. Explain view function and pure function in solidity with suitable examples.
16. What is a smart contract? What are the different types of smart contracts? #
17. Explain fixed and dynamic arrays in solidity with suitable examples.
18. Write a program in solidity to check whether anumber is prime or not. #

4. Public Blockchain

19. Describe the architecture of Ethereum.
20. Compare Bitcoin and Ethereum. #
21. Explain the following terms with respect to Ethereum: Miner and Mining Node, Gas, Accounts, Ether, Transactions.
22. List and explain various types of test networks used in Ethereum.
23. Write a short note on Ethereum Virtual Machine.

5. Private Blockchain

24. Explain Hyperledger Fabric v1 architecture.
25. What is RAFT consensus algorithm? Explain in detail.
26. Explain state machine replication with suitable example.
27. Write a short note on PAXOS consensus algorithm.

6. Tools and Applications of Blockchain

28. Write a short note on Ripple.
29. Write a short note on Corda.
30. Write a short note on DeFi (Role of smart contracts, architecture, use in blockchain)
31. Write a short note on Quorum.

	1	2	3	4	5	6
2025 May	10	50	25	30	20	10
2024 Dec	20	25	25	15	40	20
2024 May	20	40	25	20	20	20
2023 Dec	15	35	20	25	40	10
2023 May	20	30	25	10	30	30
2022 Dec	20	25	40	15	25	20
Estimate	**20**	**35-40**	**25**	**15-25**	**30-40**	**20**
Total	105	205	160	115	175	110

Asked once:

1. Introduction to Blockchain

1. Explain the components of Blockchain.
2. Write a short note on Cryptography in Blockchain.
3. With a suitable diagram, explain the structure of a block header with a list of transactions.

2. Cryptocurrency

4. Write a short note on Mining pool and its methods.
5. What is transaction structure? Explain transaction life cycle in detail.
6. Write a short note on Consensus in Bitcoin.

3. Programming for Blockchain

7. Explain the role of address and address payable in solidity with example.
8. Describe how solidity supports multiple inheritance with an example. #
9. Write a program in solidity to find the second largest element in an array. #
10. Write a program in solidity to implement single inheritance.

4. Public Blockchain

11. List and explain various types of nodes used in Ethereum. #

5. Private Blockchain

12. What is a backup in Practical Byzantine Fault Tolerance (PBFT) algorithm?
13. Compare the role of MSP and Fabric CA. Explain their role in Hyperledger blockchain.
14. Compare BFT and PBFT consensus in detail.
15. How is a smart contract represented as a state machine. #

6. Tools and Applications of Blockchain

-

Blockchain Answer bank

# indicates 5-mark question

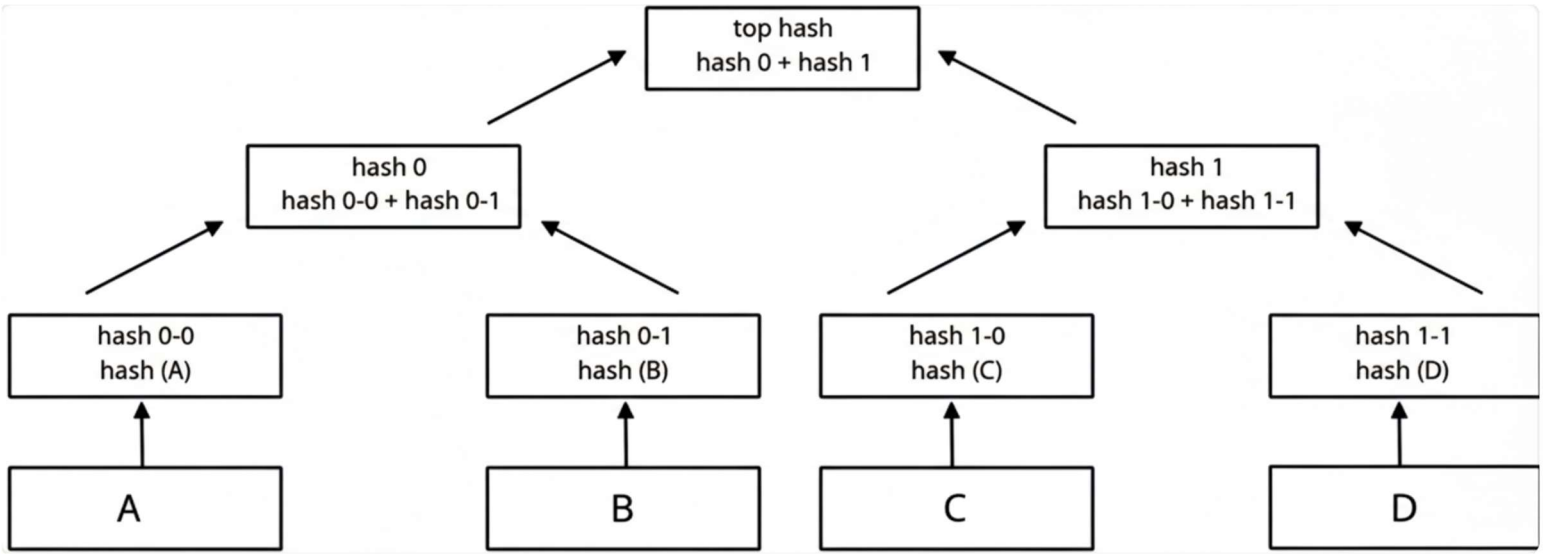
1. Introduction to Blockchain

1. Differentiate between public, private and consortium blockchain.

Aspect	Public Blockchain	Private Blockchain	Consortium Blockchain
Organization Type	Publicly accessible, no central authority	Single entity or organization	Multiple organizations
Users	Anyone can join and participate	Known, trusted members	Known, trusted members
Access	Open and transparent to all	Fully restricted	Partially open, shared among members
Network Type	Decentralized; zero points of failure	Centralized; single point of failure	Partially decentralized; multiple points of failure
Operation	Anyone can read, write, and validate	Only authorized users can transact	Only selected organizations can validate and transact
Verification	Anyone can join consensus and mine	Controlled by one validator or admin	Only selected consortium members validate
Immutability	Secured by global consensus	Secured by central authority	Secured by consortium consensus
Consensus Mechanism	PoW, PoS, etc.	PBFT, PoA, or custom methods	PBFT, RAFT, or voting-based methods
Security	High — based on cryptography and global consensus	Moderate — relies on internal control	High — depends on consortium agreement
Trust	Trustless — verified by code	Trusted central authority	Partial trust among participants
Examples	Bitcoin, Ethereum	Hyperledger Fabric	R3 Corda, Quorum

2. Explain Merkle tree with an example? Explain the structure of a Merkle tree.

A Merkle Tree (also called a Hash Tree) is a mathematical data structure used in blockchain to efficiently and securely verify the integrity of large sets of data. It is composed of hashes of data blocks and is named after Ralph Merkle, who patented the idea in 1979. It is stored in the block header.



1. Leaf Nodes (Bottom Level):

- The lowest level consists of the original data blocks: A, B, C, and D.
- Each data block is hashed individually:
  - hash 0-0 = hash(A)
  - hash 0-1 = hash(B)
  - hash 1-0 = hash(C)
  - hash 1-1 = hash(D)

2. Intermediate Nodes:

- These nodes are computed by concatenating the hashes of their child nodes:
  - hash 0 = hash(0-0 + 0-1) → combines hash(A) and hash(B)
  - hash 1 = hash(1-0 + 1-1) → combines hash(C) and hash(D)

3. Root Node (Top Hash):

- The top of the tree, also called the Merkle Root, is computed by hashing the concatenation of the two intermediate hashes:
  - top hash = hash(hash 0 + hash 1)

Any change in any data block, for example A, will change its hash, affecting all parent hashes up to the root.

3. State and explain various challenges that occurs while implementing blockchain.

1. **Scalability** – Current blockchains can process only limited transactions per second.
   This limits their use in large-scale, high-speed applications.
2. **Lack of Awareness** – Many people don't fully understand the value and uses of blockchain, making adoption slow.
   Without proper knowledge, organisations may hesitate to invest in it.
3. **Limited Technical Talent** – Skilled blockchain developers are few, making implementation difficult. This shortage increases development time and costs.
4. **High Energy Use** – Proof-of-Work blockchains require large amounts of computing power, resulting in high electricity costs and environmental impact.
5. **Immutability Issues** – Once data is recorded, it cannot be changed, which can cause problems if corrections are needed.
   This is helpful for security but inconvenient for fixing errors.
6. **Security Risks** – While the blockchain network is secure, smart contracts or applications built on it can have bugs or be hacked.
7. **Consensus Delays** – The process of all nodes agreeing on a block can be slow and resource-intensive.
   More participants mean longer confirmation times for transactions.
8. **Data Privacy Concerns** – Blockchain's transparency means recorded data is visible to all participants, which can expose sensitive information.

2. Cryptocurrency

4. Write a short note on UTXO model of Bitcoin.

UTXO (Unspent Transaction Output)

- In Bitcoin, a UTXO is the amount of digital currency remaining after a transaction, which can be used as input for future transactions.
- A transaction's output remains "unspent" until it is used as an input in another transaction.
- When a transaction is completed, any unspent amount is stored back into the blockchain as a new UTXO.

UTXO Example:

- You have three UTXOs:
  - John → 0.1 BTC
  - Sarah → 0.7 BTC
  - Sam → 0.4 BTC
- You want to buy a car costing 0.5 BTC. In Bitcoin, you must use whole UTXOs as inputs.
- Use Sarah's 0.7 BTC UTXO:

Input: 0.7 BTC from Sarah

Output: 0.5 BTC to seller

0.2 BTC back to me

- The remaining 0.2 BTC can be returned as change, used as a transaction fee, or sent to someone else. If no recipient is specified, it becomes a miner fee:

Input: 0.7 BTC from Sarah

Output: 0.5 BTC to seller

0.2 BTC assumed as fee

- After this transaction, Sarah's 0.7 BTC UTXO is spent, and new UTXOs are created for the seller and miner.
- UTXOs exist only until spent in another transaction.

Key Features of UTXO Model:

1. **Immutable & Traceable:** Every UTXO is permanently on the blockchain and traceable.
2. **Atomic Transactions:** UTXOs are fully consumed in a transaction; partial spending isn't allowed.
3. **Stateless Balance:** Wallet balance = sum of all unspent UTXOs; no stored account balances.
4. **No double spending:** Spent UTXOs cannot be reused.
5. **Parallel Verification:** Independent UTXOs enable simultaneous transaction processing.

5. Differentiate between hot and cold wallets. #

Basis	Hot Wallet	Cold Wallet
Connectivity	Always connected to the internet. Transfers occur online in real time.	Operates offline; coins are transferred with a switch between online/offline modes.
Security	Less secure; vulnerable to hacking since private keys are stored online.	More secure; private keys are stored offline, away from internet threats.
Price	Cheaper, as they don't require additional hardware.	Costlier due to need for hardware like USB drives or hardware wallets.
Acceptance	Accepted by all cryptocurrencies for storing and transactions.	Only reputed cryptocurrencies may be compatible with cold wallets.
Asset Loss Risk	If the exchange or device is compromised, access to coins can be lost.	Even if exchange closes, coins remain safe as they are not stored online.
Access/Speed	Suitable for fast, daily trading and instant exchange.	Slower and less efficient for trading due to manual transfers.
Backup and Recovery	Usually offer easy recovery with password.	If hardware is lost or damaged without backup, recovery can be difficult.
Best Use Case	Ideal for active traders needing frequent access to funds.	Ideal for long-term holders (HODLers) seeking maximum security.

6. Explain the concept of double spending with a suitable example. #

Double spending problem:

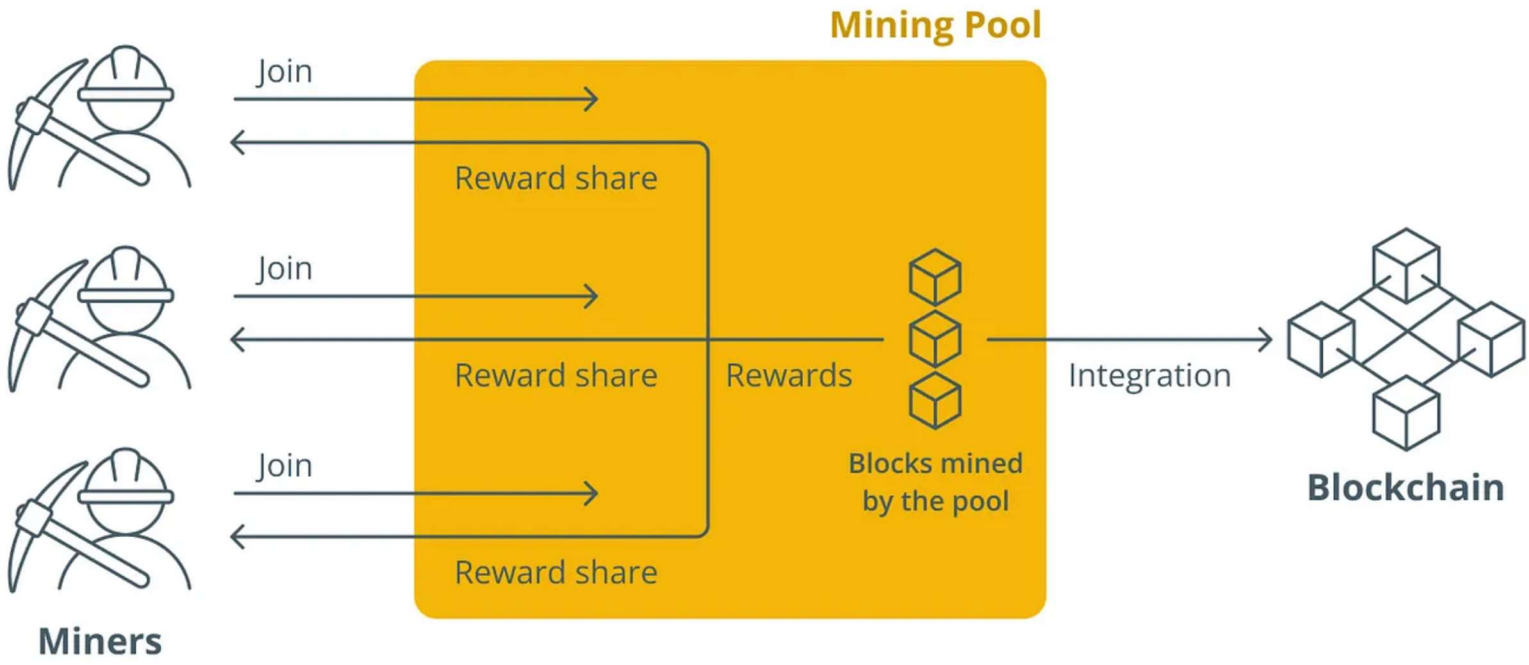
- Double spending is a possible problem in digital currencies, where the same funds or token are spent more than once.
- It happens when someone tries to use the same digital coin in two or more different transactions to cheat the system.
- This is possible because digital files can be easily copied or duplicated.
- Unlike physical cash, which cannot be copied, digital currency exists as data and if it is not properly protected, it can be used again.
- Blockchain solves this issue by verifying every transaction through consensus and recording it permanently on a distributed ledger.

Example: If Alice sends 1 BTC to Bob, she cannot send the same 1 BTC to Carol again since the first transaction is already recorded on the blockchain.

7. Explain mining pool and its difficulty. How is it calculated in a proof of work? Explain with an example.

Mining Pool

- A cryptocurrency mining pool is a group of miners who share their computational resources.
- By combining their resources, miners increase the probability of finding a block or successfully mining cryptocurrency.
- If the pool is successful and earns a reward, it is distributed among members according to their contribution.



Mining Difficulty

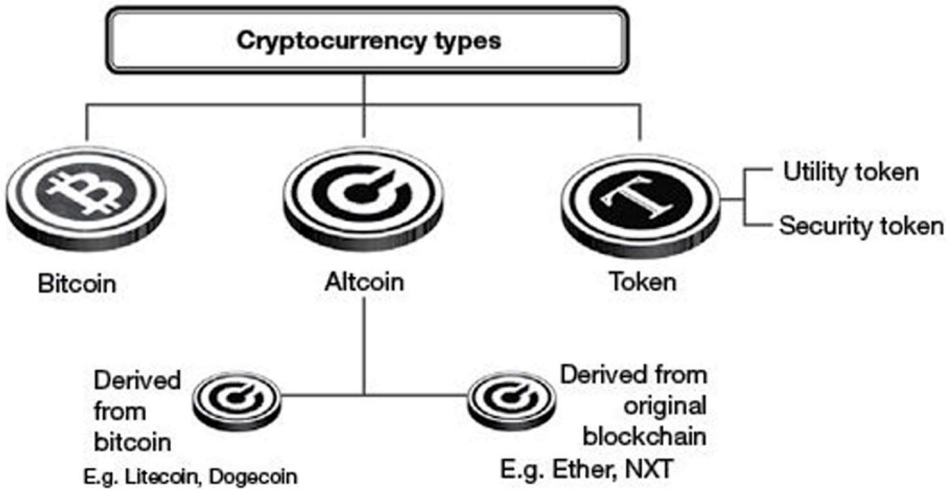
- It is defined as the measure of how difficult it is to find a hash value below a specified target.
- Bitcoin has a global difficulty that changes every 2016 blocks (approximately every two weeks). The aim is to generate one block every 10 minutes, which results in 2016 blocks in two weeks.
- **Adjustment process:**
  - If the last 2016 blocks took less than two weeks to mine then difficulty increases.
  - If it took more than two weeks then difficulty decreases.
- **Formula:**

$$\text{Current Difficulty} = \text{Previous Difficulty} \times \frac{\text{Two weeks in milliseconds}}{\text{Milliseconds to mine last 2016 blocks}}$$

- This ensures the network dynamically adjusts to maintain stable block generation times.

8. What is cryptocurrency? Explain different types of cryptocurrencies.

Cryptocurrency is a digital or virtual currency that uses cryptography for security and operates on a decentralized blockchain network.
It enables peer-to-peer transactions without intermediaries like banks and ensures transparency and immutability through blockchain.



Types of Cryptocurrency

1. Bitcoin

- Bitcoin (BTC) is the original and most well-known cryptocurrency, introduced in 2009 by an anonymous person or group known as Satoshi Nakamoto.
- It runs on a decentralized blockchain and enables peer-to-peer digital transactions without banks or intermediaries.
- Bitcoin is seen as a store of value and a potential alternative to traditional fiat currencies.

2. Altcoin

- Altcoin is a general term that refers to any cryptocurrency other than Bitcoin, with the term being a combination of "alternative" and "coin."
- Altcoins can have various features, purposes, and underlying technologies that differentiate them from Bitcoin.
- Examples of altcoins include Ethereum, Ripple, Litecoin, Cardano, and many others.

3. Token

- A token is a digital unit of value created on an existing blockchain (like Ethereum) for use in projects or decentralized applications (DApps)
- Used for purposes such as accessing services, voting, or representing assets, and can be traded on exchanges.

Types of Tokens

1. **Utility Tokens** – Give access to platform services or products, like BNB or ETH, and act as digital coupons (do not represent ownership).
2. **Security Tokens** – Represent ownership or investment in assets or companies and are regulated like securities.

9. Differentiate between PoW, PoS, PoB, PoET.

Parameter	PoW (Proof of Work)	PoS (Proof of Stake)	PoB (Proof of Burn)	PoET (Proof of Elapsed Time)
Consensus Basis	Solving complex mathematical puzzles	Validators chosen based on stake	Validators chosen based on coins burned	Random wait time verified by trusted hardware
Energy Consumption	High	Low	Low to moderate	Very low
Security	High	High (if properly implemented)	Depends on coins burned	High (depends on TEE trust)
Speed	Slow	Fast	Moderate	Very fast
Scalability	Poor	High	High	Very high
Capital Requirement	High (hardware, energy)	Medium to high (staked coins)	High (coins to burn)	Low (TEE access only)
Environmental Impact	High	Low	Low	Very low
Block Time	Slow (e.g., ~10 minutes for Bitcoin)	Fast (e.g., Ethereum PoS)	Variable, generally faster than PoW	Very fast
Reward Distribution	Block reward + transaction fees	Staking rewards + fees	Rewards proportional to coins burned	Rewards for valid wait completion
Attack Resistance	Strong (requires 51% hash rate)	Risk of 51% staking attack	Depends on burn cost	Relies on trusted hardware security
Example	Bitcoin, Litecoin	Ethereum (Post-Merge), Cardano	Slimcoin	Hyperledger Sawtooth

10. How does PoW solve the problem of double spending? #

Proof of Work stops double spending by ensuring that adding transactions to the blockchain requires a huge amount of computing effort.

- In PoW, miners compete to solve a difficult mathematical puzzle.
- The first miner to solve it gets to add a new block of transactions to the blockchain.
- Once added, the block is linked to all previous blocks, making it very hard to change without redoing the work for all following blocks.
- If someone tries to spend the same coins twice, they will have to outpace the entire network's mining power to rewrite history, which is practically impossible.

11. Explain the concept of an orphaned block. #

An orphaned block is a valid block that was successfully mined but not included in the main blockchain because another block at the same height was accepted first. The miner does not receive a reward, but orphaned blocks help keep the blockchain consistent and secure.

How It Happens:

- Two miners find a valid block almost simultaneously.
- Both blocks are broadcast to the network.
- Some nodes accept one block, while others accept the other.
- Eventually, one branch of the chain becomes longer (more proof of work), and the network adopts it as the main chain.
- The other block (on the shorter branch) gets discarded, this is the orphaned block.

Example:

If Miner A and Miner B both mine Block #101 at the same time, and Miner A's block gets extended first, then Miner B's block becomes orphaned.

3. Programming for Blockchain

12. Write a program in solidity to implement multi-level inheritance.

```
// Base contract
contract RectangleDimensions {
   uint internal a = 10;
   uint internal b = 20;
   uint internal c = 30;

   function settingDimensions() external view returns (uint, uint) {
      return (a, b);
   }
}

// Derived contract 1
contract Area is RectangleDimensions {
   function calculateArea() internal view returns (uint) {
      return a * b;
   }
}

// Derived contract 2
contract Volume is Area {
   function calculateVolume() external view returns (uint) {
      return calculateArea() * c;
   }
}
```

Volume inherits from Area, and Area inherits from RectangleDimensions, making it a multi-level hierarchy.

13. Explain fallback function in solidity with an example.

- A fallback function in Solidity is a special function that is executed when a contract receives Ether or when a function call does not match any existing function in the contract.
- It is unnamed, cannot take arguments, and does not return anything.
- Fallback functions are useful for handling plain Ether transfers or catching invalid function calls.
- In Solidity ≥0.6.0, there are two special functions:
  1. receive() → called when Ether is sent with empty calldata.
  2. fallback() → called when calldata does not match any function or when receive() is absent.

Example:

```
contract FallbackExample {
    uint public lastReceived;

    // Fallback function
    fallback() external payable {
        lastReceived = msg.value;
    }

    // Function to check contract balance
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

Explanation of example:

- If you send Ether to this contract without calling any function, the fallback() function is triggered.
- getBalance() lets you check how much Ether the contract has received.

14. Write and elaborate a code in solidity to explain visibility and activity qualifiers.

1. Visibility Qualifiers – Defines who can access a function or variable:

- **public** → accessible inside and outside the contract.
- **private** → accessible only inside the contract.
- **internal** → accessible inside the contract and derived contracts.
- **external** → accessible only from outside the contract (not internally).

2. Activity Qualifiers – Defines how a function interacts with the blockchain state:

- **pure** → does not read or modify state variables.
- **view** → reads but does not modify state variables.
- **payable** → can receive Ether.
- **Default (no qualifier)** → can read/write state variables.

Examples:

```
contract QualifiersExample {
   uint private x = 10;        // private variable
   uint public y = 20;         // public variable

   // Public function – can be called inside/outside contract
   function getY() public view returns (uint) {
      return y;
   }

   // Private function – accessible only inside this contract
   function getX() private view returns (uint) {
      return x;
   }

   // Internal function – accessible inside this contract and derived contracts
   function doubleX() internal view returns (uint) {
      return x * 2;
   }
```

```
// External function – accessible only externally
function add(uint a, uint b) external pure returns (uint) {
    return a + b;
}

// Payable function – can receive Ether
function receiveEther() external payable {
    // Accepts Ether sent to contract
}
```

15. Explain view function and pure function in solidity with suitable examples.

1. View Function:

- A view function reads state variables but does not modify them.
- It allows you to access contract data without creating a transaction.
- Calling a view function does not consume gas if called externally (no state change).

2. Pure Function:

- A pure function does not read or modify state variables.
- It works only with its inputs and returns a result.
- Pure functions are deterministic and always produce the same output for the same input.

Example:

```
contract FunctionExample {
   uint public x = 10;
   uint public y = 20;

   // View function – reads state but does not modify it
   function getX() public view returns (uint) {
      return x;
   }

   // Pure function – does not read or modify state, works only with inputs
   function add(uint a, uint b) public pure returns (uint) {
      return a + b;
   }
}
```

Explanation of Example:

- getX() → view function that reads state variable x.
- add() → pure function that only uses input parameters a and b.

16. What is a smart contract? What are the different types of smart contracts? #

A smart contract is a self-executing program stored on a blockchain, where the terms of an agreement are directly written into code. When the specified conditions are met, it automatically performs the agreed actions without the need for intermediaries. Smart contracts are secure, transparent, and immutable, making them suitable for reliable automation of transactions and processes.

Types of Smart Contracts

1. **Smart Legal Contracts:**
   Digital versions of legal agreements that enforce terms automatically but remain legally binding.
   Example: Rental contract releasing access after payment.
2. **Decentralized Applications (DApps):**
   Applications that run on blockchain using smart contracts for logic.
   Example: Decentralized exchanges.
3. **Decentralized Autonomous Organizations (DAOs):**
   Organizations governed entirely by smart contracts with rules and voting on-chain.
4. **Smart Contracting Devices:**
   Combine smart contracts with IoT devices for automated physical actions.
   Example: Smart lock opens when payment is confirmed.

17. Explain fixed and dynamic arrays in solidity with suitable examples.

1. Fixed-size Array:

- Declared with a specific size, e.g., uint[5] numbers;.
- Size cannot change after declaration.
- Useful when the number of elements is known in advance.
- Elements are stored in contiguous memory slots, so access is gas-efficient.

2. Dynamic Array:

- Declared without specifying size, e.g., uint[] numbers;.
- Size can grow or shrink at runtime using push() or pop().
- Useful when the number of elements is unknown or changes frequently.
- Offers flexibility but slightly more gas cost due to dynamic storage.

Example:

```
contract ArrayExample {
  uint[3] fixedArray = [1,2,3];   // Fixed-size
  uint[] dynamicArray;          // Dynamic

  function add(uint val) public {
    dynamicArray.push(val);     // Add to dynamic array
  }
}
```

Explanation:

- fixedArray → size = 3, cannot grow.
- dynamicArray → size can increase at runtime with push().

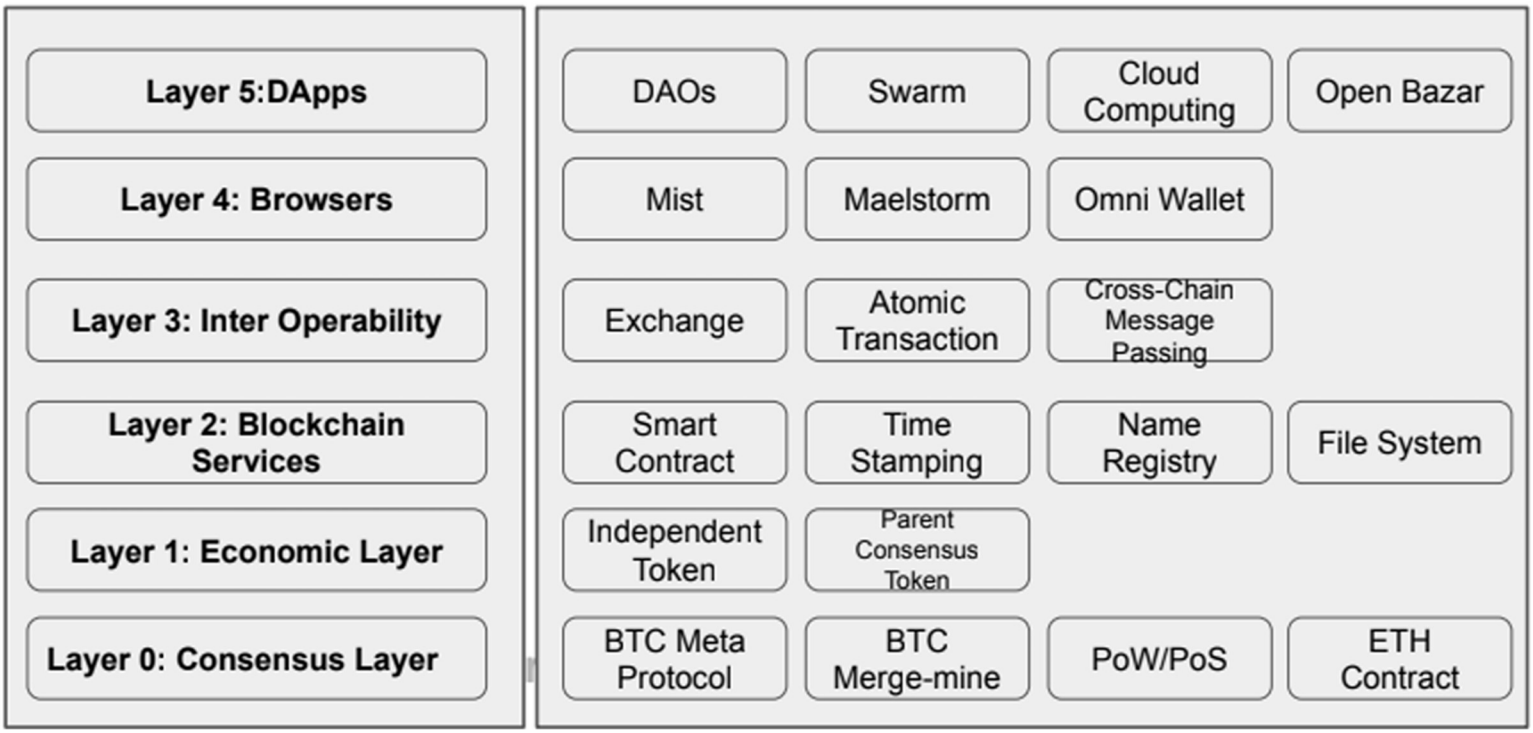
18. Write a program in solidity to check whether a number is prime or not. #

```
contract PrimeCheck {
  function isPrime(uint n) public pure returns (bool) {
    if (n <= 1) {
      return false;
    }
    for (uint i = 2; i * i <= n; i++) {
      if (n % i == 0) {
        return false;
      }
    }
    return true;
  }
}
```

4. Public Blockchain

19. Describe the architecture of Ethereum.

Ethereum's architecture consists of six functional layers, from the core consensus at the base to user-facing DApps at the top. Each layer builds on the one below to enable secure, interoperable, and decentralized applications.



Layer 0: Consensus Layer

- Responsible for agreeing on the state of the blockchain and validating new blocks.
- Ethereum uses the Proof of Stake (PoS) consensus mechanism after The Merge.

Layer 1: Economic Layer

- Deals with native tokens (e.g., Ether) that provide incentives for nodes to act honestly.
- Supports payment and rewards tied to computational or storage work.

Layer 2: Blockchain Services Layer

- Provides essential features like smart contracts, file storage, time-stamping, and registries.
- Enables core programmable and service capabilities used by DApps and upper layers.

Layer 3: Interoperability Layer

- Handles exchange and value transfer between different tokens or blockchains.
- Supports atomic transactions, cross-chain messaging, and wider ecosystem integration.

Layer 4: Browser Layer

- User-facing software (like Mist, Omni Wallet) for accessing and interacting with DApps.
- Bridges users to the decentralized world by handling wallet, identity, and UI/UX.

Layer 5: DApps Layer

- Contains decentralized applications and autonomous organizations (DAOs) running on the network.
- Delivers end-user services (from finance to cloud storage) that leverage the full blockchain stack.

Ethereum Workflow: (Only asked once, along with architecture)

1. **User Interaction:** The user accesses a DApp through a web interface using wallets like MetaMask.
2. **Smart Contract Execution:** The DApp sends transactions to smart contracts deployed on the Ethereum network.
3. **Transaction Validation:** The transaction is broadcast to the Ethereum nodes in the P2P network.
4. **Consensus:** Nodes use Proof of Stake (PoS) to validate and confirm the transaction.
5. **Block Addition:** The validated transaction is added to the Ethereum blockchain.
6. **State Update:** The blockchain state updates, and the result is reflected back to the DApp for the user.

20. Compare Bitcoin and Ethereum. #

Basis	Bitcoin	Ethereum
Definition	A decentralized digital currency that can be transferred on the peer-to-peer Bitcoin network.	A decentralized global software platform powered by blockchain technology. Known for its native cryptocurrency, Ether (ETH).
Purpose	To replace national currencies during the financial crisis of 2008.	To utilize blockchain technology for a decentralized payment network and to store computer code.
Smart Contracts	Has smart contracts, but they are not as flexible or complete as Ethereum's.	Allows for the creation of smart contracts that are computer codes stored on a blockchain and executed when predetermined conditions are met.
Smart Contract Language	Written in programming languages like Script and Clarity.	Written in programming languages like Solidity, Vyper, etc.
Transactions	Generally, Bitcoin transactions are only for keeping notes.	Ethereum transactions may contain executable code.
Consensus Mechanism	Uses Proof-of-Work (PoW).	Uses Proof-of-Stake (PoS).
Block Time	Approximately 10 minutes.	Approximately 14 to 15 seconds.
Block Limit	Has a block limit of 1 MB.	Does not have a block limit.
Structure	The structure is simple and robust.	The structure is complex and feature-rich.
Rewards	A miner gets nearly 6.25 BTC along with some additional rewards for successfully adding a new block.	A miner gets nearly 5 ETH along with some additional rewards for successfully adding a new block.
Hash Algorithm	Runs on the SHA-256 algorithm.	Runs on the Keccak-256 algorithm.
Assets	The asset of Bitcoin is BTC.	The asset of Ethereum is Ether.

21. Explain the following terms with respect to Ethereum: Miner and Mining Node, Gas, Accounts, Ether, Transactions.

1. Miner and Mining Node:

- **Miner:** A person or system that verifies transactions, creates new blocks, and earns Ether as a reward.
- **Mining Node**: A computer in the network that performs mining tasks, helping validate transactions and update the blockchain.

2. Gas:

- A measure of the computational effort needed to run transactions or smart contracts.
- Users pay Gas fees in Ether to reward miners and avoid unnecessary network load.

3. Accounts:

- Ethereum has two types of accounts:
    1. **Externally Owned Account (EOA):** Controlled by private keys, can send/receive Ether and initiate transactions.
    2. **Contract Account:** Controlled by smart contract code, can execute code and interact with other contracts.

4. Ether (ETH):

- The native cryptocurrency of Ethereum.
- Used to pay for transactions, gas fees, and act as a medium of value within the network.

5. Transactions:

- A signed message sent from one account to another.
- Can transfer Ether, interact with smart contracts, or deploy new contracts.
- Each transaction consumes Gas, and miners validate them before adding to the blockchain.

22. List and explain various types of test networks used in Ethereum.

Test networks (testnets) are separate blockchains used for testing smart contracts and dApps without risking real Ether (ETH). They mimic Ethereum's mainnet environment but use tokens of no monetary value.

1. Ropsten

- A Proof-of-Work (PoW) based testnet that behaves most like the Ethereum mainnet, making it useful for realistic testing of contracts and dApps.
- Developers use test Ether here to simulate transactions before deployment.

2. Rinkeby

- Based on Proof-of-Authority (PoA), with validators chosen and operated by trusted entities (like the Ethereum Foundation).
- Considered more stable than PoW testnets; test Ether here is requested through faucets, often linked to social media verification.

3. Kovan

- Uses the Clique PoA consensus (different from Ropsten's PoW).
- It provides a fast block time and supports Parity's Aura consensus engine.

4. Goerli

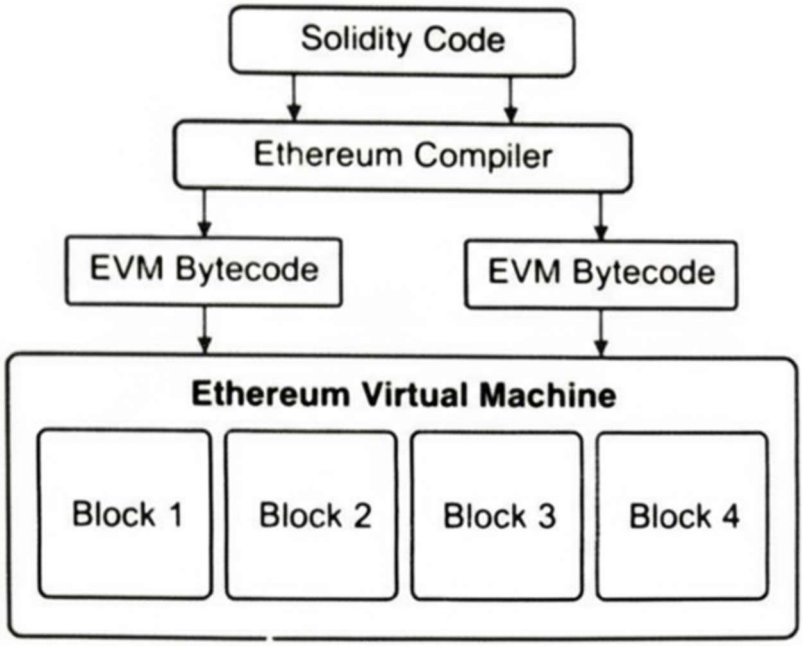
- A cross-client testnet, designed to work with multiple Ethereum clients like Geth, Besu, and Nethermind.
- Combines PoW and PoA features, aiming to provide a stable testbed for both Ethereum 1.0 and Ethereum 2.0 projects.

23. Write a short note on Ethereum Virtual Machine.

The Ethereum Virtual Machine (EVM) is the runtime environment that executes Ethereum smart contracts. Ethereum has its own Turing-complete language called Solidity, and the EVM executes the compiled code.

It runs on top of the Ethereum network so that all nodes reach consensus on what code should run at any time. Unlike Bitcoin, which only stores data, Ethereum also executes contract logic on the blockchain. The EVM can run any kind of crypto-contract written in Solidity, enabling smart contracts that automatically execute when conditions are met (or exit if conditions fail).

EVM Workflow:



- Developers write smart contracts using the Solidity programming language.
- The written code is compiled by the Ethereum compiler to produce bytecode.
- Compiled bytecode is a format the Ethereum Virtual Machine (EVM) can process.
- The EVM executes the bytecode on all Ethereum nodes, running the smart contract.
- The contract and all transactions are stored and managed in blocks on the Ethereum blockchain.

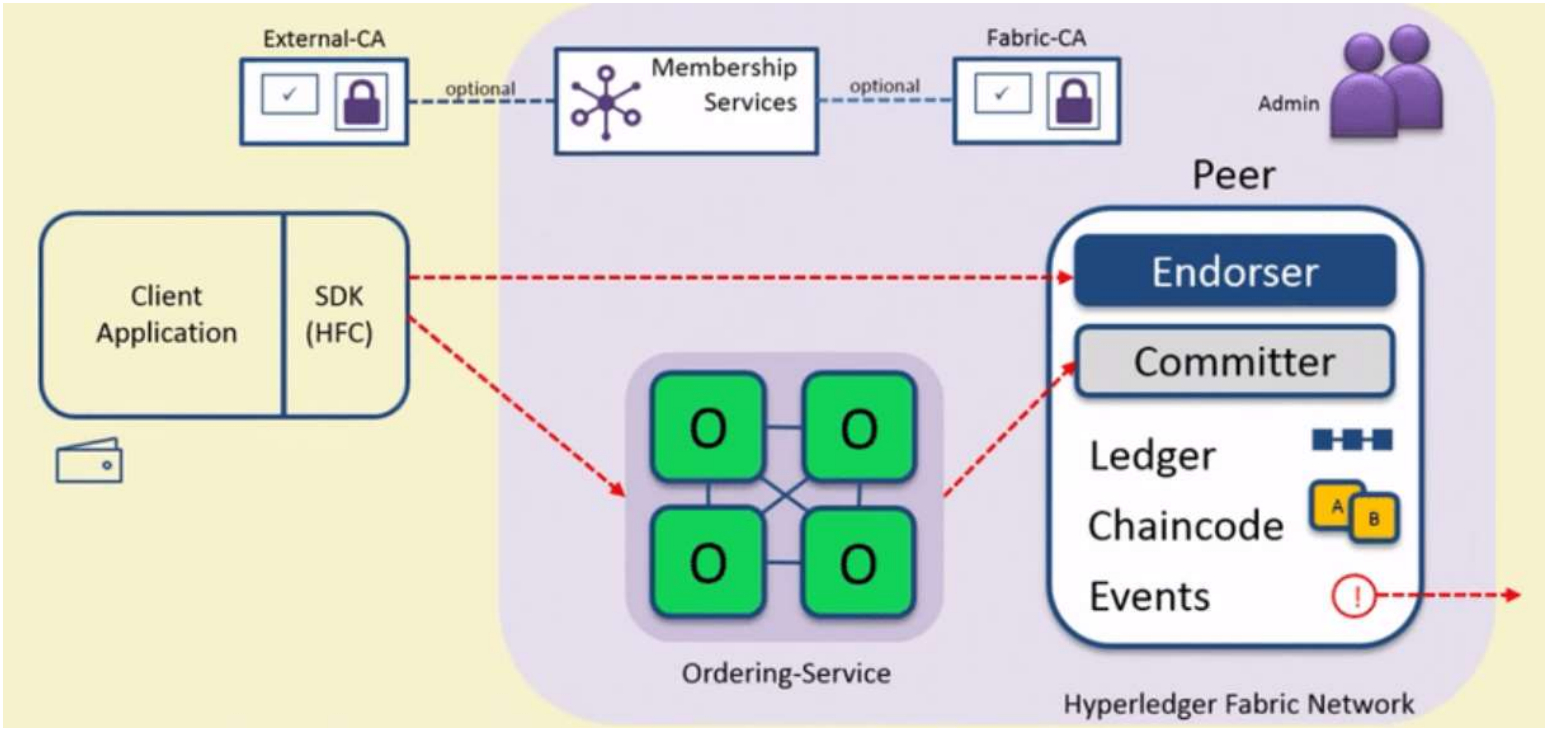
Benefits:

- Executes untrusted code without risking data.
- Can run complex smart contracts.
- Distributed consensus and robustness against failure.

5. Private Blockchain

24. Explain Hyperledger Fabric v1 architecture.

Hyperledger Fabric v1 Architecture:

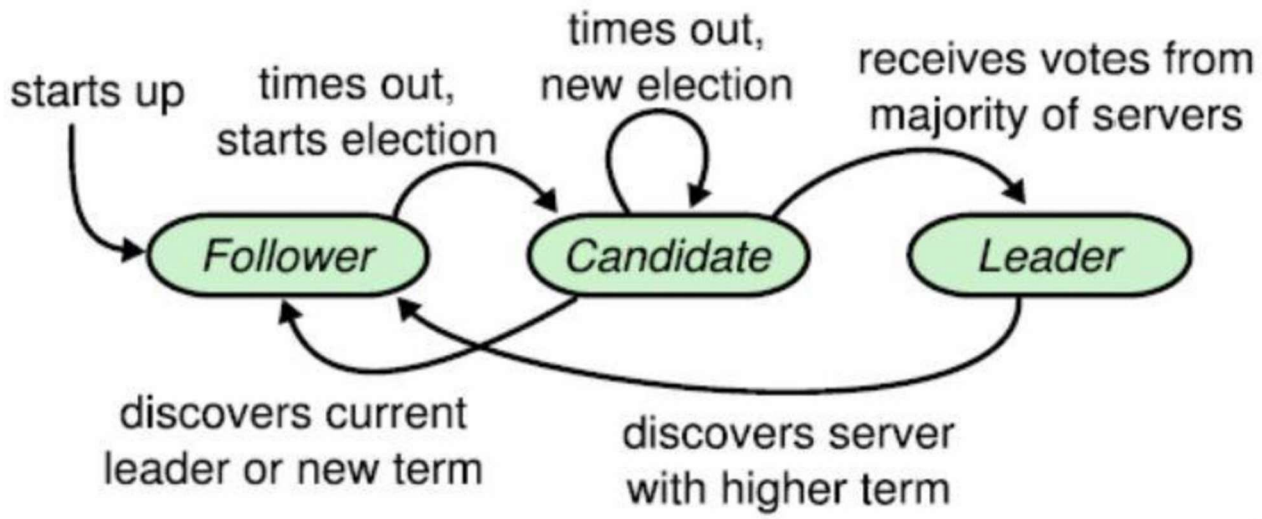


1. **Client Application & SDK (HFC):**
   - The client is used by end-users to send transaction proposals to the network.
   - The SDK (Software Development Kit) helps connect to peers and the ordering service.
2. **Membership Services (MSP / Fabric-CA / External-CA):**
   - Handles identity management and authentication of all participants.
   - Uses Certificate Authorities (Fabric-CA or External-CA) to issue digital certificates.
   - Ensures only authorized users can access the blockchain.
3. **Peers:**
   - Core components that maintain the ledger and execute chaincode (smart contracts).
   - Two main roles:
     - **Endorser:** Simulates and endorses (signs) transaction proposals.
     - **Committer:** Validates and commits transactions to the ledger.
   - **Ledger:** Stores all transactions and current world state.
   - **Chaincode:** The smart contract defining business logic.
   - **Events:** Notify clients when transactions are completed.

4. **Ordering Service:**
    - Collects endorsed transactions from clients.
    - Orders them chronologically and creates blocks.
    - Delivers these blocks to all peers for validation and commitment.
5. **Admin:**
    - Manages the network configuration, permissions, and setup of peers and orderers.
    - Typically has control over membership and policies.

25. What is RAFT consensus algorithm? Explain in detail.

RAFT is a leader-based distributed consensus algorithm designed to be simpler and more understandable than PAXOS while achieving the same goal: maintaining consistency across multiple nodes in a distributed system.



Follower:
Every server starts as a follower. It waits for messages from a leader.
If it doesn't hear from a leader for a while (times out), it becomes a candidate.

Candidate:
The candidate starts an election and asks other servers for votes.

- If it gets votes from the majority, it becomes the leader.
- If it doesn't win, it may start a new election after timing out.
- If it discovers a server with a higher term (newer leader), it goes back to being a follower.

Leader:
The leader manages and coordinates the system.
If it finds out another server has a higher term; it steps down and becomes a follower again.

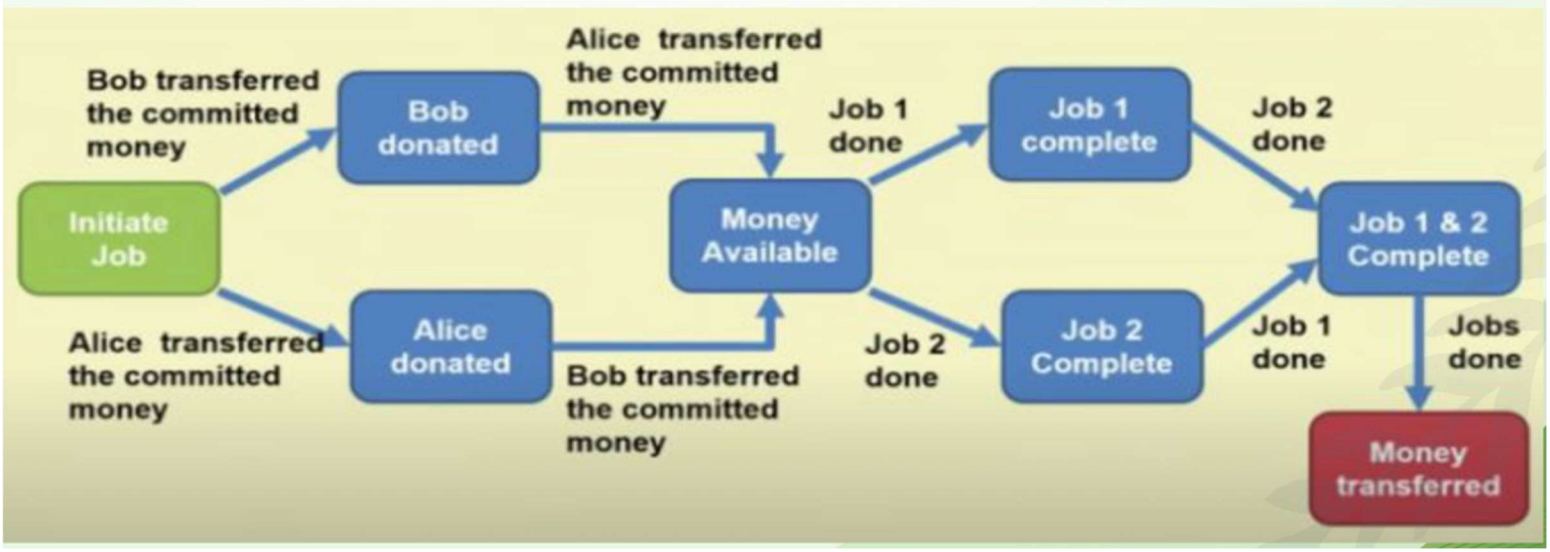
Example: RAFT in Supply Chain Blockchain

- Suppose a blockchain network has nodes: Manufacturer (Leader), Supplier, Distributor, Retailer, and Auditor.
- The Supplier initiates a transaction (e.g., record delivery).
- The Leader (Manufacturer) validates and logs the transaction, then notifies followers.
- Majority of followers (Supplier, Distributor, Retailer) confirm, but Auditor node fails to respond (offline or faulty).
- Since 4 out of 5 nodes acknowledge, consensus is reached and transaction is committed.

26. Explain state machine replication with suitable example.

- State Machine Replication is a technique used to ensure consistency in distributed systems.
- Multiple replicas (nodes) maintain the same state and execute the same operations in the same order.
- Operations are deterministic — given the same input and initial state, all replicas produce the same output and next state.
- It provides fault tolerance, meaning the system continues to function correctly even if some nodes fail.
- A consensus mechanism is used to agree on the order of operations among all replicas.
- It ensures all nodes behave identically, maintaining reliability and synchronization across the network.

Example of State Machine Replication: Crowdfunding



1) Multiple servers maintain copies of the state machine, each tracking donations and job completion.
2) **Initial state:** The process begins with the "Initiate Job" state.
3) **Inputs/transactions:** Donations from users like Bob and Alice are transactions (inputs) that all servers must process. The consensus protocol ensures these donations are processed in the same order on every server.
4) **State transitions:** After donations, the state machine transitions to "Money Available." As tasks are completed, the state changes to "Job 1 Complete" or "Job 2 Complete" and eventually to "Job 1 & 2 Complete".
5) **Final state:** When all conditions are met, the state machine reaches the final state of "Money Transferred," and the process is complete.

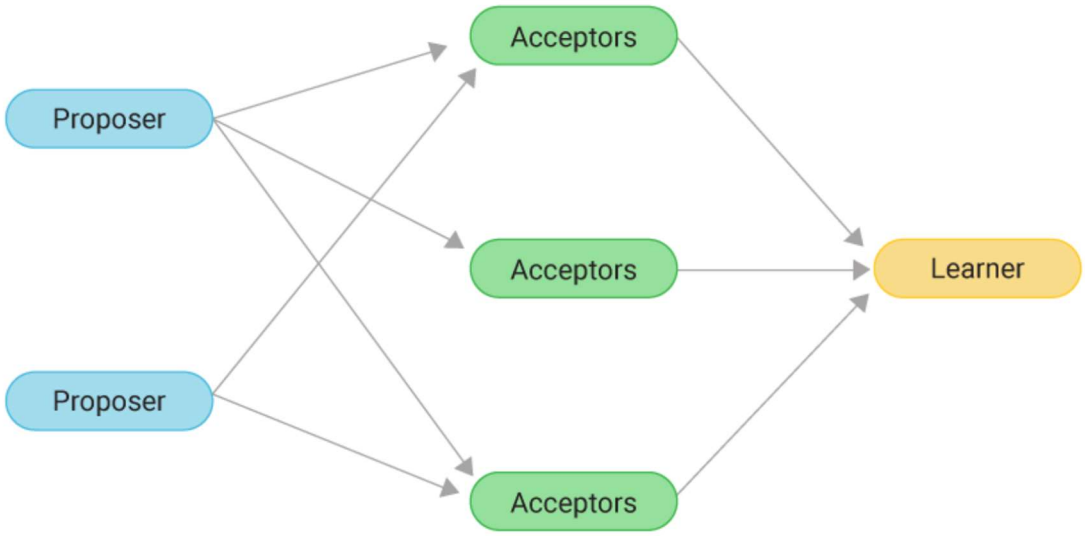
This example shows how all servers independently track the same sequence of events, ensuring that even if some servers fail, the crowdfunding process remains consistent and correct across the network.

27. Write a short note on PAXOS consensus algorithm.

Paxos is a consensus algorithm that ensures a group of distributed nodes agree on a single value, even if some nodes fail or the network partitions. It is widely used in databases and distributed systems where durability and consistency are required.

Roles in Paxos:

- **Proposers** – Propose values to be agreed upon.
- **Acceptors** – Vote on proposals; a majority vote ensures consensus.
- **Learners** – Learn and announce the final agreed value.



Working Principle:

- Proposers generate and send proposals (containing candidate values, like "who should be leader?") to all Acceptors.
- Acceptors receive proposals and vote. Each Acceptor can promise to only accept newer proposals with higher ballot numbers, ensuring safety (no conflicting decisions).
- Consensus is achieved when a majority of Acceptors vote to accept the same proposal, forming an agreement.
- Learners collect the results from Acceptors. Once consensus is reached, the agreed value is learned and announced to all nodes in the system.

Example

Imagine four servers (A, B, C, D) need to agree on a leader.

- A proposes “Leader = X” with proposal #1.
- B and C accept while D rejects.
- Majority (3 out of 4) agrees → consensus achieved → Leader = X.

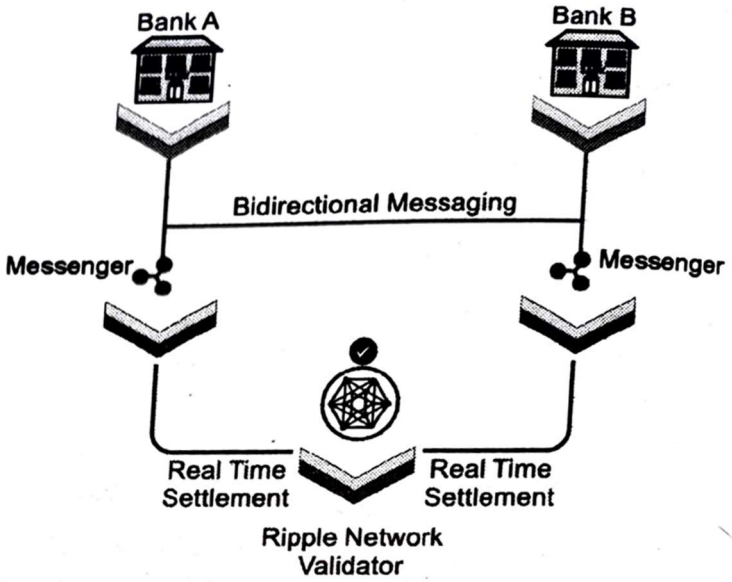
Even if one node fails, the system continues to operate correctly.

6. Tools and Applications of Blockchain

28. Write a short note on Ripple.

- Ripple is a real-time gross settlement system (RTGS), currency exchange, and remittance network built on a decentralized blockchain-like ledger.
- Its native cryptocurrency is XRP, which acts as a bridge currency to facilitate fast cross-border transactions.
- Ripple enables instant, low-cost international payments, targeting banks and financial institutions rather than individual users.
- Unlike Bitcoin or Ethereum, Ripple uses the Ripple Protocol Consensus Algorithm (RPCA) instead of Proof of Work, making it faster and energy-efficient.
- Provides a secure and flexible payment infrastructure, including features like multi-currency transfers and trust-based accounts.

Example:



(1D6)**Fig. 6.1.1 : Ripple payment system**

Ripple Payment System

Ripple enables real-time cross-border payments between banks using distributed ledger technology.

- Banks exchange payment details via Ripple's secure messaging layer.
- Validator nodes confirm authenticity and prevent double spending.
- Settlement occurs instantly: funds are debited from Bank A and credited to Bank B.

Advantages:

- Very fast (3–5 seconds) settlement.
- Low transaction cost.
- Energy-efficient and highly scalable.

29. Write a short note on Corda.

Corda is an open-source, permissioned blockchain platform developed by R3 for businesses. It focuses on secure, private transactions for industries like finance, supply chain, and healthcare.

Corda Architecture & Components:

- **Nodes & Vaults –** Each participant runs a node with a vault storing ledger states.
- **States –** Immutable facts on the ledger that evolve by consuming old and creating new ones.
- **Contracts –** Define rules for updating states, written in JVM languages.
- **Flows –** Govern communication and transaction coordination between nodes.
- **Consensus –** Ensures validity (rules check) and uniqueness (no double-spending).
- **Notary Service –** Trusted nodes ensuring uniqueness while maintaining privacy.

Working of Corda (Transaction Lifecycle)

1. Transaction Proposal

- Party A wants to transfer an asset (e.g., a bond) to Party B.
- A new transaction object is created with input state (old ownership) and output state (new ownership).

2. Verification

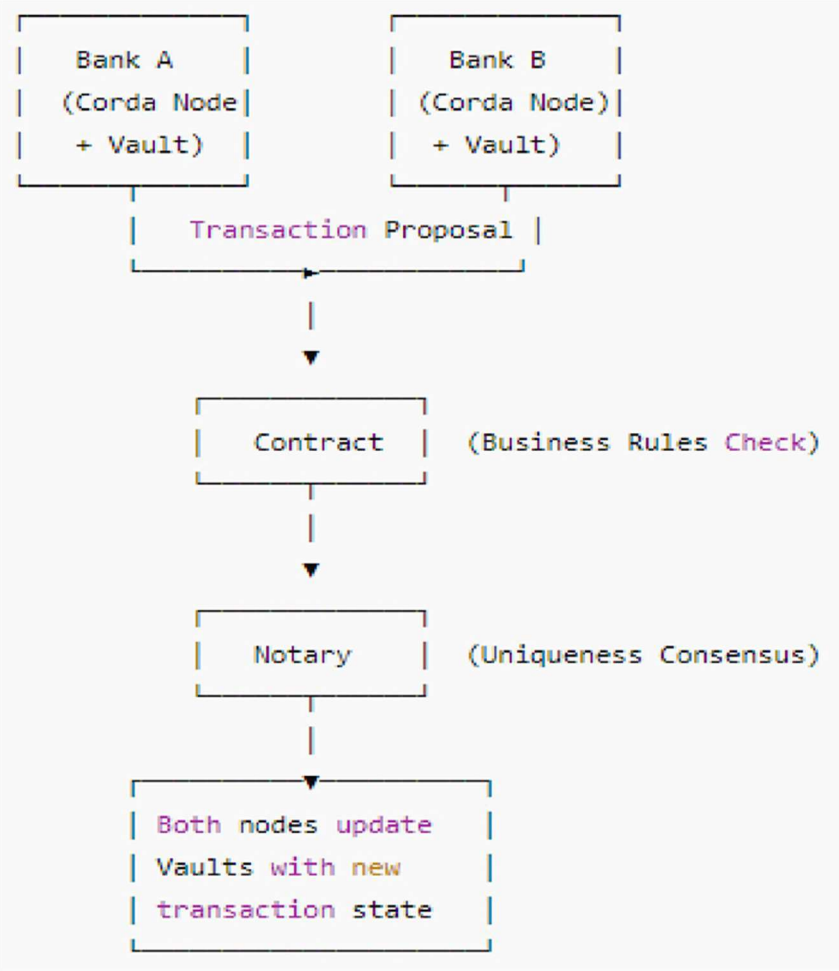
- The contract code verifies rules (e.g., "Only current owner can transfer asset").

3. Signing

- All required parties digitally sign the transaction.

4. Notarization

- Transaction sent to Notary node → checks uniqueness (ensures no double-spend).



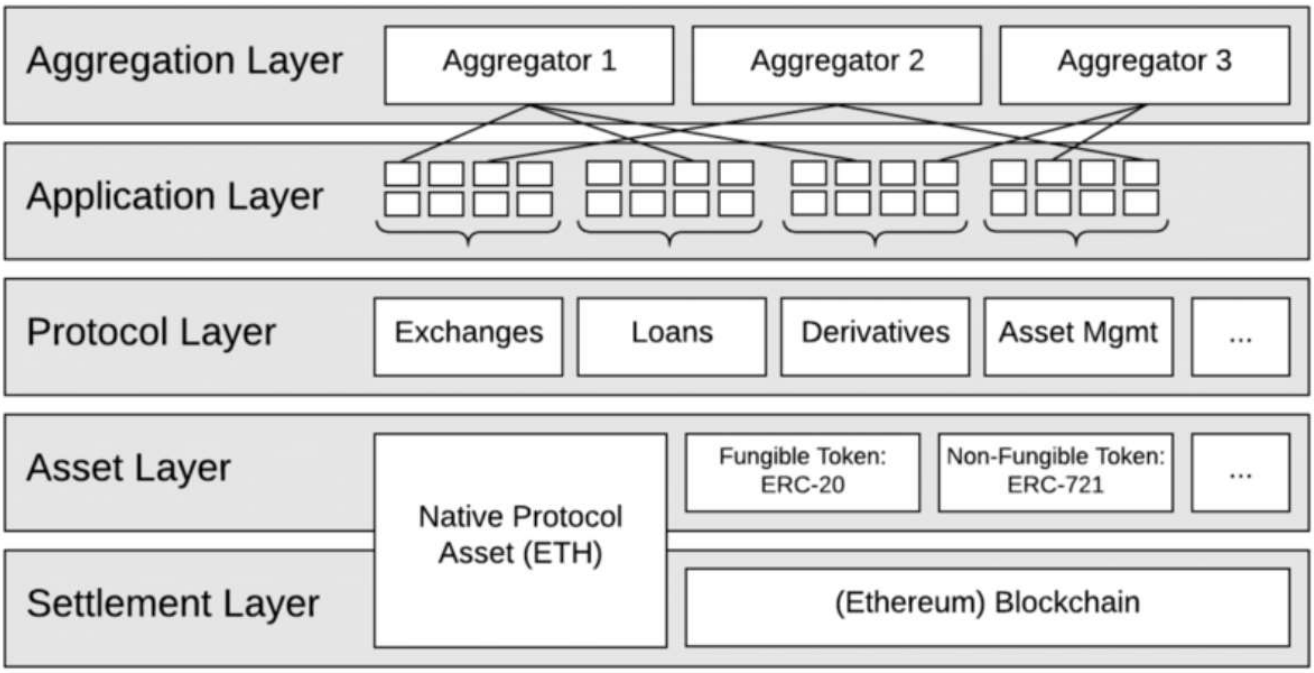
5. Recording

- Once notarized, both Party A and Party B record the transaction in their Vaults.
- Only involved parties see it, unlike public blockchains.

30. Write a short note on DeFi (Previously asked as three separate questions: Role of Smart Contracts in DeFi, DeFi Architecture, and Blockchain for DeFi in different papers)

DeFi (Decentralized Finance) is a financial system built on blockchain networks that allows users to access financial services like lending, borrowing, trading, and payments without traditional intermediaries such as banks.

DeFi Architecture:



1. **Settlement Layer:**
   - The base blockchain (Ethereum) that records all transactions.
   - Provides the secure and immutable ledger for all assets and operations.
2. **Asset Layer:**
   - Represents the digital assets used in DeFi, such as:
     - **Native assets:** ETH (Ethereum's native cryptocurrency).
     - **Fungible tokens** such as ERC-20 tokens.
     - **Non-fungible tokens:** ERC-721 tokens (unique digital assets like NFTs).
3. **Protocol Layer:**
   - Smart contract protocols that define financial services such as Exchanges, Loans, Derivatives, Asset Management, etc.
4. **Application Layer:**
   - User-facing DApps (Decentralized Applications) that interact with the protocols.
   - Provides interfaces for trading, lending, borrowing, etc.
5. **Aggregation Layer:**
   - Combines multiple DeFi services to offer better rates and convenience to users.
   - Acts as a bridge connecting different applications and protocols.

Role of Smart Contracts in DeFi:

- Smart contracts are self-executing programs on the blockchain that automatically perform financial actions when predefined conditions are met.
- They remove the need for intermediaries like banks or brokers.
- Enable trustless, transparent, and secure transactions among users.
- Used in DeFi for lending, borrowing, trading, staking, and insurance.

Example: In a lending protocol like Aave, a smart contract automatically transfers interest to the lender once repayment occurs.

Use of DeFi in Blockchain:

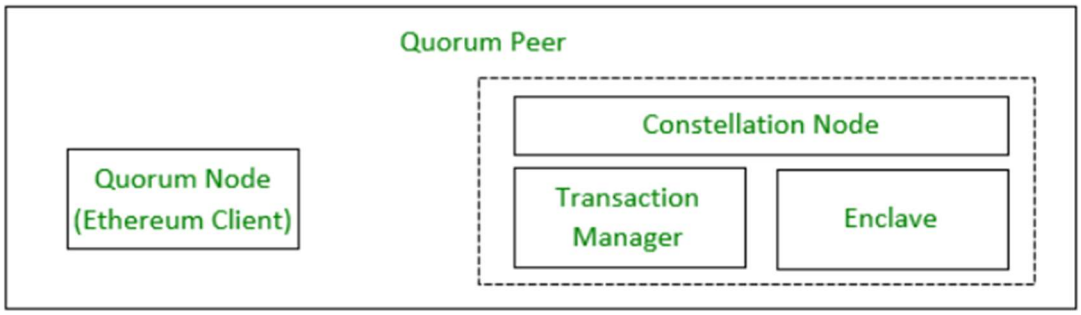
- Enables peer-to-peer (P2P) financial services such as loans, exchanges, and savings.
- Supports decentralized exchanges (DEXs), stablecoins, and yield farming.
- Provides open, global access to financial systems without centralized control.

Example: Users can trade tokens directly on Ethereum through DeFi apps like Uniswap.

31. Write a short note on Quorum.

Quorum is an enterprise-focused version of Ethereum developed by J.P. Morgan, designed for businesses that require permissioned blockchain networks with enhanced privacy and performance.

Quorum Architecture:



Quorum Architecture

- **Quorum Node (Ethereum Client):**
  Modified Geth client that manages blockchain operations like block validation and consensus.
- **Constellation Node:**
  The privacy component of Quorum.
  - **Transaction Manager:** Handles encrypted private transaction payloads.
  - **Enclave:** Performs encryption, decryption, and signing operations to secure private data.
- **Quorum Peer:**
  Combination of the Quorum Node and Constellation Node; enables both public and private transaction handling securely.

Key Features of Quorum

1. **Privacy and Confidentiality**
   Quorum supports both public and private transactions using smart contracts, ensuring confidentiality through the Private Transaction Manager (Tessera/Constellation).
2. **Enhanced Performance**
   It replaces Ethereum's Proof-of-Work with faster consensus algorithms like RAFT and IBFT, improving speed and reliability of transactions.
3. **Permissioned Network**
   Quorum is a permissioned blockchain, meaning only authorized participants can join and validate transactions, ensuring better control and security.
4. **Enterprise Integration**
   It is designed for business use, making it easy to integrate with enterprise systems and financial applications that require transparency and privacy.
5. **Smart Contract Support**
   Quorum supports Ethereum-compatible smart contracts, allowing businesses to automate workflows and maintain interoperability with Ethereum tools.

Asked once:

# indicates 5-mark question

1. Introduction to Blockchain

1. Explain the components of Blockchain.

A blockchain is a distributed ledger consisting of a chain of blocks, each containing data. Its main components are:

1. **Block:**
   - The basic unit of blockchain containing data, hash, and previous block hash.
   - Ensures integrity as each block links to the previous one.
2. **Hash:**
   - A unique identifier for a block, generated using a cryptographic hash function.
   - Any change in block data changes its hash, ensuring tamper-resistance.
3. **Mining:**
   - The process of validating and adding a new block to the blockchain.
   - Miners solve complex cryptographic puzzles to ensure security and consensus.
4. **Data:**
   - Can be transactions, contracts, or other records depending on the blockchain type.
   - Stored securely and permanently in each block.
5. **Nodes:**
   - Computers participating in the blockchain network.
   - Each node maintains a copy of the blockchain.
6. **Consensus Mechanism:**
   - Protocol to validate and agree on new blocks (e.g., Proof of Work, Proof of Stake).
   - Ensures all nodes agree on the blockchain state.
7. **Network:**
   - Peer-to-peer (P2P) network connecting nodes.
   - Allows decentralized data sharing without a central authority.
8. **Smart Contracts:**
   - Programs stored on blockchain that execute automatically when conditions are met.

2. Write a short note on Cryptography in Blockchain.

Cryptography is essential for securing blockchain data and maintaining trust in a decentralized system. It ensures that transactions are safe, verifiable, and tamper-proof.

1. **Hashing:**
    - Every block contains a cryptographic hash of its data and the previous block's hash.
    - Hash functions like SHA-256 create a unique fingerprint for each block.
    - Any change in data alters the hash, preventing fraud or tampering.
2. **Public-Key Cryptography:**
    - Users have a private key to sign transactions and a public key to verify them.
    - Ensures that only the owner of funds can authorize a transaction.
3. **Digital Signatures:**
    - Provide authentication and non-repudiation.
    - Everyone can verify the transaction came from the claimed sender.

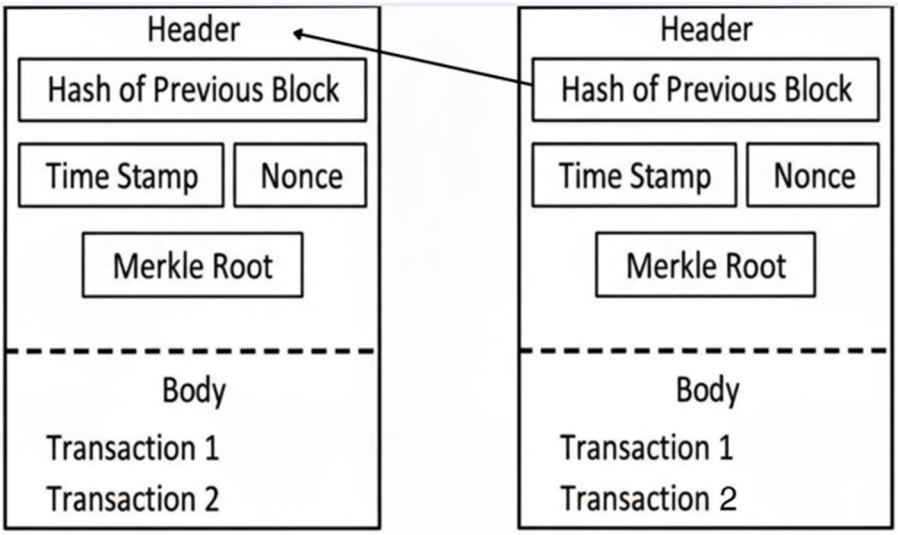
Role of Cryptography in Blockchain

Blockchain relies on cryptography to:

- Protect user identities and transaction details from unauthorized access.
- Ensure data integrity, meaning once recorded, information cannot be changed.
- Support authentication and verification of digital transactions using digital signatures.

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

Block Header



Block 0 Block 1

Split up details	Block Header Contents (80 Byte)
4 Byte	Bitcoin Version number
32 Byte	Previous block hash
32 Byte	Merkle Root
4 Byte	Time Stamp
4 Byte	Difficulty Target
4 Byte	Nonce used by miners.
80 Byte	**Total**

- **Version Number (4 bytes)** – Indicates the Bitcoin protocol rules under which the block was created. This helps the network understand how to interpret the block's data structure.
- **Previous Block Hash (32 bytes)** – A cryptographic hash of the block before the current one. This creates the chain structure by linking each block to its predecessor, ensuring immutability.
- **Merkle Root (32 bytes)** – A single hash value representing all transactions in the block, created using the Merkle tree structure. It allows the verification of transactions without checking each one individually.
- **Timestamp (4 bytes)** – The approximate time the block was mined, stored in UNIX time format. It helps in arranging blocks chronologically.
- **Difficulty Target (4 bytes)** – Represents the target threshold that the block's hash must meet for it to be accepted. It ensures mining difficulty adjusts according to network conditions.
- **Nonce (4 bytes)** – A number miners change repeatedly during Proof of Work to find a valid block hash that satisfies the difficulty target.
- **Transactions** - The body of the block contains a list of all transactions included in that block. Each transaction records the transfer of value or data on the blockchain and is verified by network nodes before being added.

2. Cryptocurrency

4. Write a short note on Mining pool and its methods.

Mining Pool

- A cryptocurrency mining pool is a group of miners who share their computational resources.
- By combining their resources, miners increase the probability of finding a block or successfully mining cryptocurrency.
- If the pool is successful and earns a reward, it is distributed among members according to their contribution.

Methods of Mining Pools:

1. **Pay-Per-Share (PPS):**
    - Each miner is paid a fixed amount for every share they contribute to the pool.
    - Payment is guaranteed, even if the pool does not find a block.
    - Reduces reward variability for miners, making income more predictable.
    - Pool operator takes a fee for managing the pool.
2. **Proportional Method:**
    - Rewards are distributed according to the number of shares contributed in the round when a block is found.
    - Payment is only made if the pool finds a block.
    - Encourages miners to contribute more shares in each round to increase potential rewards.
    - High variance in income compared to PPS.
3. **Pay-Per-Last-N-Shares (PPLNS):**
    - Rewards are calculated based on shares submitted in the last N rounds.
    - Encourages miners to stay longer in the pool, reducing pool hopping.
    - Helps evenly distribute rewards over time among active miners.
    - Reduces the risk for the pool operator compared to PPS.

5. What is transaction structure? Explain transaction life cycle in detail.

Transaction Structure

A transaction is a data structure that represents the transfer of value between participants in a blockchain network.
Each transaction typically contains the following fields:

1. **Transaction ID:** A unique hash generated for each transaction.
2. **Input:** Refers to the source of the funds being spent (previous transaction outputs).
3. **Output:** Specifies the destination address and amount to be sent.
4. **Amount:** Value of cryptocurrency being transferred.
5. **Digital Signature:** Verifies the authenticity of the sender.
6. **Timestamp:** Records the time when the transaction was created.

Transaction Life Cycle

1. **Creation:**
   A user creates a transaction specifying sender, receiver, and amount.
2. **Signing:**
   The sender digitally signs the transaction using their private key to ensure authenticity.
3. **Broadcasting:**
   The signed transaction is sent to nearby network nodes through the peer-to-peer network.
4. **Validation:**
   Nodes verify that the sender has sufficient balance and the digital signature is valid.
5. **Propagation:**
   The validated transaction is shared across the network and stored temporarily in the mempool.
6. **Mining (Inclusion in Block):**
   Miners pick valid transactions from the mempool and include them in a new block.
7. **Confirmation:**
   Once the block is added to the blockchain, the transaction is considered confirmed and irreversible.

6. Write a short note on Consensus in Bitcoin.

Consensus in Bitcoin

- Consensus in Bitcoin ensures that all nodes in the decentralized network agree on a single, consistent version of the blockchain.
- It prevents double spending, fraud, and conflicting transactions, maintaining trust without a central authority.

How Bitcoin Achieves Consensus:

1. **Proof of Work (PoW):**
   - Miners compete to solve a cryptographic puzzle to add a new block.
   - The first miner to solve it adds the block and receives a block reward.
   - PoW requires significant computational effort, which secures the network against attacks.
2. **Longest Chain Rule:**
   - Nodes consider the longest valid chain as the correct blockchain.
   - If multiple blocks are mined simultaneously, eventually the chain with the most cumulative work becomes accepted.
3. **Decentralized Verification:**
   - Each node independently verifies transactions in a block before accepting it.
   - Ensures that all participants agree on transaction validity and maintain the blockchain's integrity.

Importance:

- Maintains a trustless and decentralized network.
- Ensures blockchain integrity and prevents tampering.
- Enables secure, peer-to-peer transactions without a central authority.

3. Programming for Blockchain

7. Explain the role of address and address payable in solidity with example.

1. address

- Represents a 20-byte Ethereum account (externally owned account or contract).
- Can store an account address and check balances using balance.
- Cannot receive Ether directly using transfer() or send().
- Useful for reading account information but not sending Ether.

2. address payable

- A special type of address that can receive Ether.
- Has functions like transfer(), send(), and call{value: ...}() to send Ether.
- Use when you want a contract or account to accept payments.

Example:

```
contract AddressExample {
  address public normalAddress;
  address payable public payableAddress;

  // Send Ether to payable address
  function sendEther() public payable {
    payableAddress.transfer(msg.value);
  }

  // Get balance of normal address
  function getBalance() public view returns (uint) {
    return normalAddress.balance;
  }
}
```

Explanation:

- normalAddress → just stores an address; cannot receive Ether.
- payableAddress → can receive Ether using transfer().
- getBalance() shows how you can read balances of any address.

8. Write a program in solidity to implement single inheritance.

```
// Base contract
contract RectangleDimensions {
   uint internal a = 10;
   uint internal b = 20;

   function settingDimensions() external view returns (uint, uint) {
      return (a, b);
   }
}

// Derived contract
contract Area is RectangleDimensions {
   function calculateArea() public view returns (uint) {
      return a * b;
   }
}
```

This code shows single inheritance in Solidity. The Area contract inherits variables and the function from RectangleDimensions, allowing it to directly access a and b to calculate area.

9. Describe how solidity supports multiple inheritance with an example.

Multiple inheritance:

- Solidity allows a contract to inherit from multiple parent contracts.
- Solidity resolves the inheritance order automatically using C3 linearization.
- Variables and functions from all parent contracts are accessible in the derived contract.

Example:

```
contract A { uint x = 10; }

contract B { uint y = 20; }


contract C is A, B {
   function sum() public view returns (uint) {
      return x + y;
   }
}
```

Explanation of example:

C inherits from both A and B.

sum() can access variables x and y from both parents.

10. Write a program in solidity to find the second largest element in an array. #

```
contract SecondLargest {
    function findSecondLargest(uint[] memory arr) public pure returns (uint) {
        require(arr.length >= 2, "Array must have at least 2 elements");
        uint largest = arr[0];
        uint second = arr[0];

        for (uint i = 1; i < arr.length; i++) {
            if (arr[i] > largest) {
                second = largest;
                largest = arr[i];
            } else if (arr[i] > second && arr[i] != largest) {
                second = arr[i];
            }
        }
        return second;
    }
}
```

4. Public Blockchain

11. List and explain various types of nodes used in Ethereum.

Types of Ethereum Nodes:

1. **Full Node:**
    - Stores the entire blockchain and validates all transactions and blocks.
    - Ensures the node follows consensus rules and can serve other nodes.
2. **Light Node (Light Client):**
    - Stores only block headers instead of the full blockchain.
    - Relies on full nodes for transaction and block verification.
    - Saves storage and bandwidth, suitable for mobile or low-resource devices.
3. **Archive Node:**
    - Stores everything a full node does plus all historical states of the blockchain.
    - Useful for data analysis or querying past states, but requires large storage.

5. Private Blockchain

12. What is a backup in Practical Byzantine Fault Tolerance (PBFT) algorithm?

The Practical Byzantine Fault Tolerance (PBFT) algorithm is a consensus mechanism used in distributed systems and blockchains to achieve agreement even when some nodes are malicious or faulty.

PBFT uses a replica-based architecture with two main roles:

1. **Primary (Leader)** – Coordinates the ordering of requests.
2. **Backups (Replicas)** – Verify and agree on the leader's proposal.

In the Practical Byzantine Fault Tolerance (PBFT) algorithm, a backup is any replica (node) in the network that is not the primary (leader) node.

Role of Backups in PBFT:

Backups are responsible for:

1. **Receiving requests** broadcast by the primary node.
2. **Verifying correctness** of the request/order proposed.
3. **Participating in the three consensus phases**-
    - **Pre-prepare**: Receive message from primary.
    - **Prepare**: Broadcast confirmation to other replicas.
    - **Commit**: Final confirmation that all replicas agree.
4. **Executing the client request** after consensus.

Example

If there are 4 replicas (R0, R1, R2, R3):

- **R0** = Primary
- **R1, R2, R3** = Backups

If the primary proposes an incorrect transaction, backups detect the inconsistency and trigger a view change to elect a new primary.

Thus, system integrity is maintained even if one node acts maliciously.

13. Compare the role of MSP and Fabric CA. Explain their role in Hyperledger blockchain.

Parameter	MSP (Membership Service Provider)	Fabric CA (Certificate Authority)
Purpose	Defines and manages identities and roles of network participants	Issues digital certificates to authenticate identities
Function	Manages user identities and verifies signatures	Creates and provides certificates for users and nodes
Scope	Works within the network to control access	Manages certificates for each organization
Integration	Used by the network to validate transactions	Provides certificates that MSP uses
Centralization	Distributed across organizations	Central service per organization
Updates	Can add/remove members dynamically	Can issue, renew, or revoke certificates

Role in Hyperledger Blockchain

1. MSP (Membership Service Provider)

- Ensures only authorized participants can join the network.
- Defines roles (peer, orderer, client) and access rights.
- Provides a decentralized trust mechanism without relying on a single authority.
- Helps in managing identity across multiple organizations in a blockchain consortium.

2. Fabric CA

- Acts as the certificate authority for the network.
- Issues and manages digital certificates (X.509) for users, peers, and orderers.
- Handles enrolment, renewal, and revocation of certificates.
- Works with MSP to provide cryptographic credentials needed for authentication.

14. Compare BFT and PBFT consensus in detail.

Parameter	BFT (Byzantine Fault Tolerance)	PBFT (Practical Byzantine Fault Tolerance)
Definition	Theoretical model to tolerate Byzantine faults in distributed systems	Concrete algorithm implementing BFT in real-world systems
Purpose	General concept to handle faulty/malicious nodes	Makes BFT practical and efficient in asynchronous systems
Implementation	No specific implementation – serves as a foundation	Specific algorithm designed by Castro and Liskov (1999)
Consensus Process	Varies depending on the system	3-phase process: Pre-Prepare → Prepare → Commit
Communication Complexity	Depends on protocol; may be optimized	High – $O(n^2)$ messages per consensus round
Performance	Can be inefficient in practice	More efficient, especially in small networks
Leader Election	May or may not involve a leader	Uses a designated primary (leader) node
Scalability	Poor without optimization	Limited due to high message overhead
Examples	Tendermint	Hyperledger Fabric v0.6

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

A smart contract on a blockchain can be represented as a state machine because it maintains a current state (e.g., token balances or ownership) and changes it when a transaction occurs.

- Each transaction acts as an input that triggers a state transition.
- The contract then updates to a new state in a fixed, predictable way.
- Since all nodes run the same code on the same input, they reach the same result, keeping the blockchain consistent.
- Thus, a smart contract behaves like a finite state machine, where transitions are controlled by transaction logic and consensus ensures identical updates on all nodes.