VIT - BHOPAL UNIVERSITY

TASK ROUND

Google Developer Groups
On Campus
VIT Bhopal University

Submitted By :-
Name:- Anay Verma
Reg. No :- 22BCE10120

Task 1.Blockchain (Immutable,Dis. P2P , Decentralized , CryptoCurrency)

Picture this : a shared digital notebook that everyone in a neighborhood can see and use, but no single person owns or controls it. That's basically what a blockchain is.

How Blockchain Works:

- Think of it like a chain of blocks, where each block contains information (like transactions)
- Every time someone makes a transaction, it gets added as a new block
- Everyone in the network has a copy of this entire chain
- No one can cheat because everyone else would see the changes

Security: Decentralized (Blockchain):

- Harder to hack because information is stored on many computers
- No single point of failure
- Might be slower to fix security issues since changes need group agreement

Centralized (Traditional Systems):

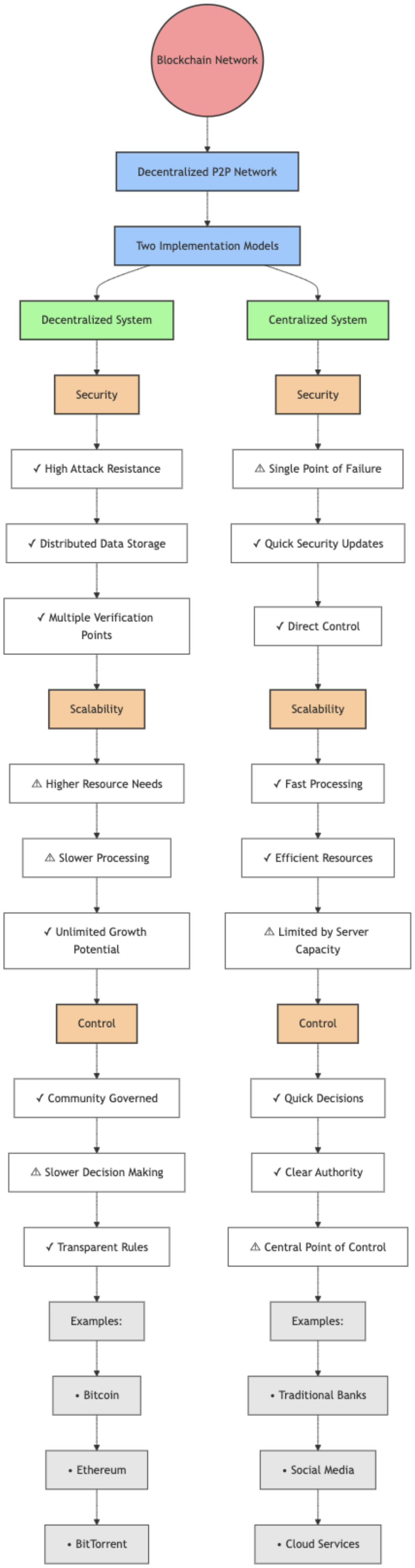
- Easier to implement security updates quickly
- Clear responsibility for security
- Single point of failure (like a bank getting hacked)

Scalability: Decentralized:

- Can handle many users joining the network
- Often slower because all participants need to verify transactions
- Uses more energy and resources

Centralized:

- Faster transaction processing
- More efficient resource use



- Limited by central server capacity

Control: Decentralized:

- No single authority can make changes alone
- More democratic decision-making
- Harder to make quick changes or fixes

Centralized:

- Quick decision-making
- Clear rules and governance
- Power concentrated in few hands

Real-world Example: Think of the difference between:

- WhatsApp (centralized): One company controls everything
- BitTorrent (decentralized): Files shared directly between users

Key Challenges:

- Energy consumption: Decentralized systems often need more power
- Speed vs. Security: More security checks mean slower transactions

---

Distributed Peer to Peer

Transaction Initiation:

- Any peer can initiate a transaction
- The initiating peer validates it first
- No central authority needed

Network Broadcasting:

- Transaction is broadcast to connected peers
- Each peer acts as both client and server
- Information spreads across the network

Verification Process:

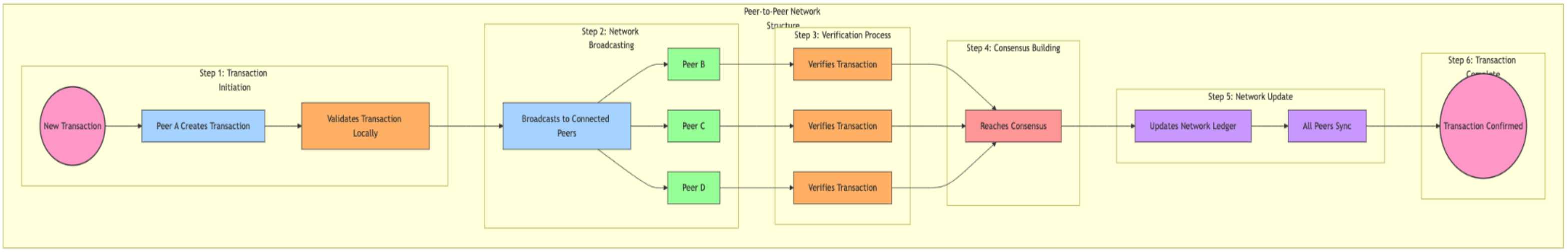
- Each peer independently verifies the transaction
- Uses same validation rules
- No single point of failure

Consensus Building:

- Peers must agree on transaction validity
- Uses predetermined consensus rules
- Prevents double-spending/conflicts

Network Update:

- All peers update their copies of the ledger
- Maintains network consistency
- Creates permanent record


Peer-to-Peer Network Structure
Step 1: Transaction Initiation
New Transaction
Peer A Creates Transaction
Validates Transaction Locally
Step 2: Network Broadcasting
Broadcasts to Connected Peers
Peer B
Peer C
Peer D
Step 3: Verification Process
Verifies Transaction
Verifies Transaction
Verifies Transaction
Step 4: Consensus Building
Reaches Consensus
Step 5: Network Update
Updates Network Ledger
All Peers Sync
Step 6: Transaction Complete
Transaction Confirmed

Task 2 . “ERC20 Token”

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

contract MyERC20Token {
   string public name;
   string public symbol;
   uint8 public decimals;
   uint256 private _totalSupply;

   // ii) balanceOf: Indicates the number of tokens an address holds.
   mapping(address => uint256) private _balances;

   // v) approve: Allows a contract to spend a specified number of tokens from an
account.
   mapping(address => mapping(address => uint256)) private _allowances;

   // Event declarations for Transfer and Approval
   event Transfer(address indexed from, address indexed to, uint256 value);
   event Approval(address indexed owner, address indexed spender, uint256 value);

   // Constructor to initialize the token
   constructor(string memory tokenName, string memory tokenSymbol, uint256
initialSupply) {
       name = tokenName;
       symbol = tokenSymbol;
       decimals = 18;
       _totalSupply = initialSupply * (10 ** uint256(decimals)); // Convert to the
correct decimals
       _balances[msg.sender] = _totalSupply; // Assign the total supply to the creator
       emit Transfer(address(0), msg.sender, _totalSupply);
   }

   // i) totalSupply
   function totalSupply() public view returns (uint256) {
       return _totalSupply;
   }

   // ii) balanceOf
```

```
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    // iii) transfer: Requires that the sender has a sufficient balance to send.
    function transfer(address to, uint256 amount) public returns (bool) {
        require(to != address(0), "ERC20: transfer to the zero address");
        require(_balances[msg.sender] >= amount, "ERC20: transfer amount exceeds balance");

        _balances[msg.sender] -= amount;
        _balances[to] += amount;

        emit Transfer(msg.sender, to, amount);
        return true;
    }

    // v) approve
    function approve(address spender, uint256 amount) public returns (bool) {
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }

    // iv) transferFrom: Allows the transfer from an account that is not making the transaction.
    function transferFrom(address from, address to, uint256 amount) public returns (bool) {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
        require(_balances[from] >= amount, "ERC20: transfer amount exceeds balance");
        require(_allowances[from][msg.sender] >= amount, "ERC20: transfer amount exceeds allowance");

        _balances[from] -= amount;
        _balances[to] += amount;

        _allowances[from][msg.sender] -= amount;

        emit Transfer(from, to, amount);
        return true;
    }

    // vi) allowance: Returns the amount an approved contract is still allowed to spend
or withdraw.
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    // vii) burn: Allows a user to burn tokens, permanently reducing the totalSupply.
    function burn(uint256 amount) public returns (bool) {
        require(_balances[msg.sender] >= amount, "ERC20: burn amount exceeds balance");

        _balances[msg.sender] -= amount;
        _totalSupply -= amount; // Reduce total supply

        emit Transfer(msg.sender, address(0), amount); // Emit transfer to zero address
        return true;
    }
}
```

Contract Address (Deployed) :-
0x1227115292e9b6180e080efa4a3556ca6639b194

Check here (Deployed) :-
https://sepolia.etherscan.io/tx/0x312ade9e7ec83b2d1741f75524d442f7d95e32dc615b3f4b39cb6e7c33bc976d

---

Video Explanations

- https://www.loom.com/share/ab691f1660e74030bf1d50d47097e5ff