Unit 4: Ethereum and Smart Contracts – Study Guide

Introduction: Ethereum is a decentralized blockchain platform that extends Bitcoin's basic design to support *programmable transactions*. It was proposed by Vitalik Buterin in 2013 and launched in 2015 as a "world computer". Ethereum allows developers to write *smart contracts* – self-executing code stored on the blockchain – enabling complex applications without central servers or censorship. Its native cryptocurrency is **ether (ETH)**, which "fuels" the network by paying transaction fees. Unlike Bitcoin, Ethereum's consensus rules do **not** pay a fixed block reward to miners (now validators); instead, only transaction fees are paid, which helps decouple economics from fixed supply [1]. Today Ethereum has the largest developer community and supports thousands of decentralized applications (Dapps), token standards (e.g. ERC-20, ERC-721), and innovations like ICOs and DeFi. In short, Ethereum is a programmable blockchain platform whose design goals are generality and flexibility [1].

- **Blockchain 2.0:** Ethereum's design ("blockchain 2.0") uses a Turing-complete language so that *any* rules or programs (smart contracts) can run on-chain [2]. By contrast, Bitcoin's scripting is very limited.
- **Ether vs Ethereum:** The term *Ethereum* refers to the blockchain platform; *ether* (ETH) is its cryptocurrency. Ether is "fuel" for the network: users pay gas (fees) in ether to execute contracts.
- **Decentralized Apps:** Dapps on Ethereum run exactly as coded without downtime or censorship. Developers deploy smart contracts in languages like Solidity or Vyper, which the Ethereum Virtual Machine (EVM) compiles to bytecode for execution.
- **Permissionless:** Anyone can run an Ethereum node, deploy contracts, or use Dapps without approval. Transactions are public, transparent, and immutable once confirmed.

Summary: Ethereum is a programmable blockchain platform powered by the cryptocurrency ether. It enables smart contracts and decentralized applications through its global EVM, offering much more flexibility than Bitcoin. Users pay gas fees in ETH to execute transactions and contract code, and all operations are validated by a decentralized network of nodes.

2. Ether, Gas, and Transactions

Ethereum's economy relies on *Ether (ETH)* as the base token and *gas* as the computation fee. Each Ethereum **transaction** is a signed message from one account to another, potentially including an ether payment and/or smart contract invocation. Transactions include: sender address, recipient address, optional ether amount, optional contract bytecode/data, and the gas price/limit the sender is willing to pay. Every transaction must be signed with the sender's private key to prove its authenticity.

- **Ether (ETH):** Ether is the cryptocurrency used on Ethereum. It functions like "bitcoin on Ethereum" – users pay ETH to run code. One ether = 10^9 *gwei* (gigawei), and one gwei = 10^9 wei (smallest unit). Ether rewards validators for processing transactions and also deters spam (each operation has a cost in gas).

- **Gas:** *Gas* is the unit of compute on Ethereum. Each operation in a contract or transaction costs a fixed gas amount based on complexity. Users set a **gas limit** (max gas units willing to spend) and a **gas price** (ETH paid per gas unit). If execution uses less gas than the limit, the remainder is refunded; if it runs out of gas, execution halts (the state changes revert) but the spent gas is still consumed. Thus, gas *decouples* Ethereum's fees from the ETH price and prevents infinite loops (since running out of gas stops execution [3]).
- **Transaction Fees:** Gas fees are paid **in ETH** to validators. Common practice is to quote gas prices in gwei. For example, a simple ETH transfer may cost ~21,000 gas (≈0.000021 ETH at 1000 gwei). Complex smart contract calls cost more gas. The gas spent is burned or given to validators as a reward (after Ethereum's upgrade).
- **State Changes:** Every transaction that modifies the blockchain produces a new *state*. Ethereum is often described as a **state machine**: starting from an initial world state σ, each block's transactions execute sequentially to produce the next state σ' [4]. In other words, "the Ethereum computer lives and breathes transactions" – only transactions can change the state[4].

Figure: Every Ethereum transaction transforms the global state from σ to σ' [4]. The network of nodes applies each signed transaction to update account balances and contract state.

- **Transaction Flow:** When a user (an Externally Owned Account) submits a transaction, it is broadcast to the network and placed in the mempool. Miners/validators pick up transactions, execute them on the EVM, and if valid, include them in a block. The resulting state changes (transfers, contract updates, event logs) become part of the immutable blockchain.
- **Limitations:** A sender must always include enough ether to cover the gas limit; otherwise the transaction is invalid. Additionally, the **nonce** of the transaction (the sender's transaction count) ensures correct ordering.

Summary: A transaction on Ethereum is a signed instruction (possibly including ETH transfer or contract code) that, once mined, updates the blockchain state. Users pay gas fees (in ETH) for the computational resources consumed. The gas mechanism prevents abuse by requiring fees proportional to work. Each block of transactions deterministically transforms the state via the **state transition function**[4].

3. Ethereum Virtual Machine (EVM)

The **Ethereum Virtual Machine (EVM)** is the runtime engine that executes smart contract bytecode on every node. Conceptually, it is a global, decentralized computer: each full node runs an instance of the EVM that independently computes the results of transactions. All EVM instances must agree on the output for a given input, which enforces consensus and determinism (if given the same starting state and transactions, every node arrives at the same final state).

Figure: The Ethereum Virtual Machine (EVM) architecture. The EVM is a stack-based, isolated runtime that processes smart contract code in bytecode form [3] [5]. The virtual stack, memory, and storage manage data during execution, and a program counter (PC) steps through instructions. The world state (persistent storage, e.g. account balances) is updated upon successful execution.

- **Stack-Based and Turing-Complete:** The EVM is a **simple stack machine** that processes 256-bit words (32 bytes) in a last-in-first-out stack [3]. It is Turing-complete (can run arbitrary loops/logic), but every operation costs gas, so loops terminate if gas runs out [3]. The stack is limited to 1024

elements, and the machine is **big-endian**. Each opcode pushes or pops values on the stack, performs arithmetic or logic, and may modify memory or storage [3] [6].

- **Isolated and Deterministic:** The EVM runs in a sandbox: contract code cannot directly access external systems (no network or file access)[3]. This isolation ensures that contract execution is fully deterministic and verifiable by all nodes. If any node executes the same transaction and gets the same gas and state, it will reach the same result, ensuring consensus.
- **Memory vs Storage:** During execution, contracts have access to two data areas: **Memory** and **Storage** [5]. Memory is a byte-array that is **cleared** after the transaction ends (volatile, like RAM). Storage is a *key-value store* that persists on-chain (non-volatile, part of the blockchain state). Reading or writing to storage costs much more gas than using memory. For example, contract variables are stored in storage (and survive across transactions), whereas temporary arrays or function-local data are kept in memory.
- **EVM Execution:** When a transaction invokes a contract, its bytecode (from the contract's address) is fed into the EVM. The EVM sets up an *execution environment* (including caller, value, input data, and block context) and then steps through the bytecode instructions [7] [8]. Each instruction uses gas; when the program counter reaches the end or a **STOP** / **RETURN** opcode, execution completes, returning any output and the new state. If an exception occurs (e.g. out-of-gas, invalid opcode), execution halts and the transaction reverts, but spent gas is still consumed[4].
- **Deterministic State Transition:** The EVM's operation embodies Ethereum's state transition function: given an initial state σ and transaction T, the EVM yields a new state σ′ and remaining gas [4]. This property ensures that the network agrees on the ledger's updates.

Summary: The EVM is the backbone of Ethereum's execution model. It is a stack-based, 256-bit virtual machine that each node runs identically, turning smart contract bytecode and transactions into state changes [3]. Its gas mechanism ensures resource limits and security. In essence, Ethereum is a globally synchronized state machine powered by the EVM.

4. Accounts and Transactions

Ethereum maintains a global state consisting of accounts and their balances/storage. There are **two types of accounts**[10]:

- **Externally Owned Accounts (EOAs):** These are user accounts controlled by private keys (like Bitcoin wallets). An EOA has a balance (in wei), a nonce (transaction counter), and **no associated code** [10]. Only an EOA can **initiate** a transaction by signing it with its private key [10].
- **Contract Accounts:** These accounts have an address and a balance, *and* they have contract bytecode (and associated persistent storage) stored on-chain [10]. Contract accounts are *passive*: they cannot initiate transactions on their own. Instead, they are triggered when an EOA sends a transaction to their address [11]. When triggered, the EVM executes the contract's code. Contracts can in turn send messages (internal transactions) to other contracts, but ultimately all transactions originate from an EOA.

Thus, **transactions** on Ethereum always start at an EOA. A transaction has this structure:

- **Sender (from):** EOA address that signs the transaction.
- **Recipient (to):** either another account or a contract address.
- **Value:** amount of ETH to transfer (optional; can be zero).

- **Data/Bytecode:** payload or code to execute (if creating or calling a contract).
- **Gas Price & Gas Limit:** the fee parameters.
- **Nonce:** the sender's transaction count (prevents replay).

When an EOA signs and broadcasts a transaction, the network enforces that the sender has enough ETH to cover both the value + (gas price × gas limit). The **gas fee** (gas used × gas price) is paid out of the sender's balance and credited to the block proposer (validator). If a transaction includes contract bytecode with a function call, the EVM will execute that code. Otherwise, if `data` is empty, it is simply an ETH transfer. The transaction's execution can change balances, update contract storage, or generate events/logs.

Summary: Ethereum's world state is a map of accounts. Only EOAs can send signed transactions; contract accounts run code when called [10]. A transaction includes sender, receiver, optional ETH amount, optional contract code, and gas parameters. The network verifies each transaction, deducts fees, and executes any contract code via the EVM, leading to deterministic state updates.

5. Smart Contracts

A **smart contract** on Ethereum is simply code (a program) stored at a blockchain address, with defined functions and persistent storage. Once deployed, smart contracts run exactly as written: when they receive transactions or messages, the Ethereum nodes automatically execute their logic. Smart contracts enable trustless automation – for example, escrow, token issuance, voting, and many other applications without intermediaries.

- **Definition:** A smart contract is application code living on Ethereum that can be invoked by transactions. It is typically written in a high-level language (Solidity or Vyper), compiled to EVM bytecode, and deployed to an address. Anyone can call its public functions by sending transactions to its address.
- **Lifecycle:** The basic lifecycle of a smart contract is: write the contract code and compile it; **deploy** it in a transaction (paying gas and a one-time deployment cost); then **execute** it whenever it is triggered. Specifically: (1) Encode the business logic in a contract; (2) Broadcast and mine the contract creation transaction; (3) The contract lives at a new address on-chain; (4) When certain conditions or calls occur, its code executes automatically; (5) The resulting state changes (e.g. token balances, stored variables) are recorded in the blockchain. Once deployed, a contract cannot be changed – its code is immutable (unless you include upgrade mechanisms).
- **Attributes:** Smart contracts have several key properties:
- **Security & Trustlessness:** The contract's code runs exactly as written and is secured by cryptography. Since many nodes validate execution, malicious changes are prevented.
- **Autonomy:** Contracts execute automatically without human intervention once triggered; no need for middlemen or central authorities.
- **Efficiency:** They can eliminate costly intermediaries. For example, a lending contract can automatically enforce collateral rules without banks. This tends to reduce fees and speed up processes.
- **Transparency:** All contract transactions and state are on the public blockchain, so anyone can audit the code and the execution history. This transparency builds trust, but also means vulnerabilities are visible.
- **Standard Tokens:** One of the most common uses of Ethereum contracts is tokens. For example, **ERC-20** is a standard interface for fungible tokens (each token is identical). An ERC-20 contract

implements specific functions (like `transfer`, `balanceOf`, etc.) as defined in the ERC-20 specification. By 2018 there were over 100,000 ERC-20 tokens on Ethereum. (There are also ERC-721 and ERC-1155 standards for **non-fungible tokens (NFTs)** – unique digital collectibles or assets [12] [13].)

- **Example – Token Contract:** For instance, a contract could represent a simple coin: the contract's storage holds a mapping from addresses to balances. When someone calls `transfer(to, amount)`, the contract code checks sender's balance, subtracts the amount, and adds it to the recipient. Each call is an on-chain transaction, so the ledger of balances is fully transparent.

Case Study: In 2016 the **DAO** (Decentralized Autonomous Organization) was launched as an on-chain investment fund governed by smart contracts [14]. It raised $150M in ETH, showing Ethereum's flexibility. However, a vulnerability was exploited and ~$50M was drained. The community responded by hard-forking Ethereum to reverse the hack, leading to two chains (ETH and ETC). This episode highlighted smart contracts' power and the importance of secure code [14].

Summary: Smart contracts are autonomous programs on Ethereum that manage state and enforce rules without intermediaries. They must be carefully coded (bugs cannot be patched after deployment). Widely-used contract standards (ERC-20 tokens, ERC-721 NFTs) have enabled new applications in finance, art, gaming, etc. Overall, smart contracts turn Ethereum into a programmable ledger.

6. Ethereum Ecosystem and Use Cases

Ethereum's flexibility has led to many real-world use cases:

- **Decentralized Finance (DeFi):** Ethereum is the prime platform for DeFi, a suite of blockchain-based financial applications. DeFi apps let users borrow, lend, trade, and earn interest without banks. For example, **Uniswap** is a popular on-chain automated market maker (decentralized exchange) for swapping tokens, and **Compound** allows token holders to earn interest or borrow assets by locking collateral. DeFi is open to anyone with an internet connection – there are no central authorities to block access [15] [16]. Tens of billions of dollars in crypto have flowed through Ethereum DeFi protocols, enabling services like lending, synthetic assets, and real-time payment streaming [15]. Key innovations include automatic liquidation mechanisms, stablecoins (pegged tokens like DAI or USDC) for reducing volatility, and on-chain insurance.
- **Non-Fungible Tokens (NFTs):** NFTs are unique digital items encoded on Ethereum. Popular uses include digital art, collectibles, and gaming items. Examples: *CryptoKitties* (2017) was one of the first famous NFT projects on Ethereum [13], where each CryptoKitty (a virtual cat) is distinct and owned via an ERC-721 token. *Decentraland* uses NFTs to represent ownership of virtual land in a metaverse. NFTs make it easy to prove ownership and authenticity of digital items, enabling artists and creators to sell digital art on platforms like OpenSea. As Investopedia notes, NFTs "are unique cryptographic tokens that exist on a blockchain and cannot be replicated," representing either digital collectibles or even real-world assets like property or identity [12].
- **Decentralized Autonomous Organizations (DAOs):** DAOs are organizations governed by code and token-holder votes, without centralized leadership. Members use tokens to vote on proposals, and smart contracts automatically execute decisions (e.g. releasing funds from the DAO treasury when a motion passes) [17] [14]. Famous examples include **The DAO** (2016) and more recent ones like **MakerDAO** (governs the DAI stablecoin) or **Lido DAO** (staking services). DAOs enable collective

ownership: for instance, ConstitutionDAO (2021) raised funds on Ethereum to bid on a rare book, showing how communities can pool crypto funds for a goal [14].

- **Enterprise and Government:** Ethereum's technology is also used in private/permissioned contexts for supply chain tracking, identity management, and record-keeping. For example, governments experiment with blockchain voting or land registries. Corporates use private Ethereum-based chains to track assets or automate contracts. (See Unit 5 for these case studies.)
- **Notable Examples:** Ethereum has powered many high-profile projects: e.g. ENS (Ethereum Name Service) for human-readable addresses, NFTs like Beeple's art (sold for millions), and the architecture for hundreds of blockchain games and collectibles. The platform's open-source nature means anyone can build new innovations – for instance, the **layer-2** ecosystem (Polygon, Arbitrum) is expanding scalability and DeFi reach.

Interesting Fact: NASA used Ethereum smart contracts to incentivize asteroid-mining proposals in a NASA challenge, showing the platform's versatility. Also, Ethereum's developer conference (Devcon) often pioneers new ideas for Web3.

Summary: Ethereum is the foundation for a broad ecosystem of decentralized applications. Major use cases include DeFi (blockchain lending, exchanges), NFTs (unique digital assets), DAOs (code-governed organizations), and many more. These applications leverage smart contracts on Ethereum to remove intermediaries and increase transparency [15] [12].

7. Consensus, Upgrades, and Scaling

Ethereum originally used **Proof of Work (PoW)** like Bitcoin: miners solved puzzles to propose blocks and were rewarded in ETH. Key differences from Bitcoin included faster block times (~12–14 seconds) and an uncapped supply rate (up to ~18M ETH per year). However, to address energy use and scale, Ethereum has been transitioning to **Proof of Stake (PoS)** in a series of upgrades.

- **The Merge (2022):** On Sept 15, 2022 (block 15,537,393), Ethereum completed *"The Merge"* [13]. This combined the original Ethereum execution layer with the PoS Beacon Chain. After the Merge, **mining** was retired and network security comes from validators who stake ETH. This upgrade cut Ethereum's energy consumption by ~99.95% [18]. As the official Ethereum site notes: it "improved sustainability" and set the stage for future scalability upgrades [18].
- **Proof of Stake (PoS):** In PoS, validators must put up ETH collateral to earn the right to propose/attest new blocks. If they act maliciously, they can lose part of their stake. Blocks are finalized faster and without massive computation. The Merge itself did not introduce sharding or major throughput increases, but it was a prerequisite for those future steps.
- **Future Upgrades:** Post-Merge, Ethereum plans to implement **sharding** and other layer-2 solutions to improve scalability. Sharding will break the blockchain into multiple sub-chains (shards), allowing parallel processing of transactions. This, combined with rollups and off-chain techniques, aims to enable thousands of transactions per second.
- **On-Chain vs Off-Chain vs Sidechains:** To scale and improve privacy, some activity can occur outside the main chain:
- **On-Chain:** All transactions executed directly on Ethereum's blockchain. This ensures full decentralization and immutability. Examples: sending ETH, interacting with smart contracts, recording data on-chain.

- **Off-Chain:** Transactions or state channels conducted privately between parties, settled later on-chain. Off-chain transfers are fast and cheap (because they avoid on-chain consensus) and can be kept confidential. They boost throughput (fewer on-chain writes) but may trade some decentralization. Use cases include payment channels (like Lightning), off-chain swaps, or batch trades.
- **Sidechains:** Independent blockchains that run alongside Ethereum and are connected via a two-way peg. Assets can be moved between Ethereum and a sidechain through locking/unlocking. Sidechains can have their own consensus rules and faster block times for high-throughput or specialized applications. For example, a gaming company might use a sidechain tailored for microtransactions, then bridge results to the main Ethereum chain periodically.
- **Summary of Scaling:** These approaches aim to keep Ethereum's security while increasing capacity. On-chain operations are most secure but slower and costlier; off-chain/sidechain methods improve speed and flexibility at the expense of some decentralization or requiring trust bridges.

Summary: Ethereum's consensus mechanism transitioned from PoW to PoS with *The Merge* in 2022[18], dramatically cutting energy use. This enables future scaling efforts like sharding. Meanwhile, developers use off-chain and sidechain solutions to handle more transactions and specialized use cases, balancing throughput vs decentralization.

8. On-Chain vs. Off-Chain vs. Sidechains

Understanding these modes of operation is key for grasping Ethereum's design tradeoffs:

- **On-Chain:** All data and computation occur on Ethereum's public ledger. Transactions and smart contract executions are validated by the network's consensus (now PoS). On-chain operations benefit from maximum decentralization and security – no single party can censor or alter results. Examples include sending ETH, deploying contracts, or recording NFT ownership directly on Ethereum. The downside is limited throughput and higher fees under heavy load.
- **Off-Chain:** Activities occur outside the main chain, with only summary or settlement posted on-chain. For instance, users might exchange IOUs or batch trades off-chain, then a single on-chain transaction finalizes the net result. Off-chain transactions can be processed quickly and privately (ideal for micropayments, private agreements). They reduce on-chain congestion and fees, but require trust in the off-chain protocol or mechanisms (e.g. payment channels rely on crypto-locked contracts to ensure eventual settlement).
- **Sidechains:** These are separate blockchains pegged to Ethereum. Assets move between mainchain and sidechain via locking mechanisms (two-way pegs). Sidechains can have different rules (faster blocks, different gas models) tailored for specific use cases. They provide scalability (since many transactions can run on the sidechain without affecting mainnet) and a sandbox for innovation. However, they rely on their own security model; moving assets back to Ethereum requires trust in the peg's integrity. Example use cases: gaming networks with thousands of rapid transactions, or enterprise blockchains interoperating with Ethereum.

Summary:
- *On-chain* means everything is on Ethereum itself - fully trustless but relatively slow and expensive.
- *Off-chain* means doing work outside Ethereum to save cost/speed, then anchoring results on-chain.
- *Sidechains* are alternative blockchains linked to Ethereum, boosting capacity and flexibility at some security trade-off.

9. Summary of Unit 4

- **Ethereum Basics:** A platform for decentralized computing and smart contracts; native token = ether (ETH). Introduced by Vitalik Buterin, Ethereum extends Bitcoin's ledger into a **global state machine**.
- **Ether vs Gas:** Ether is the cryptocurrency (also an incentive), while *gas* measures computation cost. Users pay gas (in ETH) to execute transactions, which prevents abuse. Gas price is quoted in gwei (10^9 wei).
- **Accounts & Transactions:** Two account types – EOAs (controlled by users) and contract accounts (with code) [10]. Only EOAs can originate transactions; each transaction is signed and can transfer ETH or call a contract. All state changes happen through transactions, making Ethereum a state-transition system [4].
- **Ethereum Virtual Machine (EVM):** The EVM is the runtime environment that executes smart contract code on every node [3] [5]. It is a stack-based, deterministic machine (256-bit word size) governed by gas limits [3]. Contracts run in the EVM sandbox, which ensures consistent execution across nodes.
- **Smart Contracts:** Programs on Ethereum that automatically execute when called. They hold state and define rules. Smart contracts are immutable once deployed, so code must be correct. They enable Dapps like ERC-20 tokens, NFTs (ERC-721), DeFi protocols, etc. Key advantages include transparency, security by code, and removal of intermediaries.
- **Use Cases:** Ethereum powers DeFi (lending, DEXs), NFTs (digital collectibles, gaming assets), DAOs (on-chain governance), and many enterprise solutions. Examples: Uniswap for token swaps, CryptoKitties as collectible NFTs [13], MakerDAO for a stablecoin system, and numerous real-world pilots (supply chain, identity).
- **Consensus & Upgrades:** Ethereum moved from PoW to PoS with *The Merge* (Sep 2022) [18], drastically reducing energy use. Validators now secure the network by staking ETH. Future scaling upgrades (sharding, rollups) will greatly increase throughput. Off-chain and sidechain techniques are also used to scale and add privacy.

Each major point above is grounded in current literature: *Basics of Blockchain*, *Mastering Blockchain* [3] [10], and respected sources like Ethereum.org [18] [15] and Investopedia [12] [14]. These ensure accuracy and depth.

10. Practice Exam Questions

Question 1 (8 marks): *Describe the Ethereum Virtual Machine (EVM) architecture and explain how it processes transactions. Why is gas important in the EVM?*

Answer: The EVM is a virtual execution environment that runs on every Ethereum full node. It is a *stack-based, 256-bit* virtual machine [3]. Each smart contract is compiled to EVM bytecode. When a transaction invokes a contract, that bytecode is loaded into the EVM. The EVM maintains several data structures: a **stack** (last-in, first-out, 1024 elements max), **memory** (byte-array cleared after execution), and **storage** (persistent key-value store) [3] [5]. Execution follows the program counter (PC), reading opcodes one by one. For each opcode (like ADD, MUL, CALL), the EVM pops operands from the stack, performs the operation, pushes results back, and updates PC. If an opcode accesses memory or storage, it reads/writes there. This continues until a STOP or RETURN, or until gas runs out (in which case execution aborts).

Gas is crucial because it *meter's the EVM's computation*. Every opcode has a fixed gas cost. Before execution, the transaction sender specifies a **gas limit** and **gas price**. The EVM decrements the gas counter as it executes each step. If the gas counter hits zero, execution halts (with an "out of gas" error) and state changes revert. This mechanism prevents infinite loops and abuse. It also decouples transaction fees from ETH's price, since the user buys gas in ETH. The spent gas (gas used × gas price) is paid to validators as a fee. Thus, gas ensures that running code on the EVM has a cost proportional to resource usage [3].

Question 2 (9 marks): *Compare Ether and Gas in Ethereum. Include definitions and their roles. How is gas used to deter malicious behavior?*

Answer: Ether (ETH) is the native cryptocurrency of Ethereum. It exists as balances in accounts on the ledger. Ether can be transferred between accounts and is used to pay for services. **Gas** is a separate unit that measures the computational effort of running transactions and smart contract code. Every operation has a gas cost (e.g. addition might cost 3 gas, a storage write 20,000 gas, etc.). The sender of a transaction specifies a *gas limit* (max gas to use) and *gas price* (ETH per gas). When the transaction is executed on the EVM, the actual gas used is multiplied by the gas price, and that amount of ETH is paid as a fee.

Roles: Ether is the **currency**; gas is the **fee mechanism**. You pay gas **in ETH**. One ethersworth of gas is called 1 gwei (10^9 wei), and 1 ETH = 1,000,000,000 gwei. By paying gas, users compensate validators for their work. So, ETH incentivizes network participation, while gas translates computational work into ETH terms.

Deterring Abuse: Gas serves as a penalty for running code. Since every instruction requires gas, malicious or infinite loops would consume gas rapidly and eventually halt when gas runs out [3]. This means attackers must pay for any computation they force. It prevents spam (flooding network with transactions) and denial-of-service attacks, because attackers would have to spend exorbitant gas fees to slow the network. In summary, gas ensures economic cost is tied to computation, deterring wasteful use of resources [3].

Question 3 (9 marks): *Explain the concept of a smart contract on Ethereum, and outline its lifecycle. Give an example of a simple smart contract (e.g. a token) and its key features.*

Answer: A *smart contract* is a program stored on Ethereum that automatically executes when triggered by transactions. Its lifecycle is: (1) **Development:** Write the contract logic in Solidity or another language. (2) **Deployment:** Compile to EVM bytecode and send a *contract creation* transaction. When mined, the code lives at a new address on the blockchain. (3) **Execution:** Anytime a user sends a transaction to that contract's address (or another contract calls it), the EVM runs the contract's code. (4) **Completion:** Execution may update the contract's own storage, send ETH or tokens, or call other contracts. Those changes become part of the new global state. Contracts can be invoked repeatedly and exist indefinitely. Once deployed, code is immutable, so careful testing is crucial.

Example – ERC-20 Token: Consider a simple cryptocurrency token contract following ERC-20. Its state includes a **mapping(address => uint256) balances** and a **uint256 totalSupply** . Key functions: **transfer(to, value)** subtracts **value** from the sender's **balances** and adds it to **to** 's balance; **balanceOf(addr)** returns balances[addr]; **approve** / **transferFrom** handle allowances. Each of these operations is a function call transaction. Features include events (for Transfer logs) and ownership rules.

Since it's ERC-20, any wallet or exchange knows how to interact with it. The contract must check for underflows (cannot send more tokens than balance) or throw (revert) otherwise.

Key features of this smart contract example:
- **Autonomy:** It enforces token transfers without any central custodian.
- **Transparency:** All balances and transfers are public on-chain.
- **Security:** The code is enforced by the EVM; for instance, if a user tries to transfer 100 tokens but only has 50, the contract will revert (reject) the transaction.
- **Standardization:** By following ERC-20, the token is automatically compatible with wallets and exchanges.

Thus, a smart contract like a token is essentially an account on Ethereum with code defining its behavior. Users interact with it by sending transactions that invoke its functions. This example illustrates the general concept: smart contracts model agreements and logic in code, executed on the blockchain.

Question 4 (8 marks): *Discuss Ethereum's transition from Proof of Work to Proof of Stake (The Merge). What were the motivations and outcomes of this change?*

Answer: Ethereum originally used Proof of Work (PoW) consensus, where miners expended energy to solve cryptographic puzzles, securing the network. However, PoW is energy-intensive. To address this and improve scalability, Ethereum planned to shift to Proof of Stake (PoS) as part of "Ethereum 2.0".

The **Merge** (September 15, 2022) was the key upgrade that merged the Ethereum mainnet with the PoS Beacon Chain [18]. From that point on, PoW mining was disabled. Validators now secure Ethereum by *staking* (locking up) ETH, and in return they get the right to propose/attest new blocks. The motivations were: **sustainability** and **scalability**.

- **Energy Savings:** The Merge reduced Ethereum's energy consumption by ~99.95% [18] because PoS doesn't require huge hashpower. This made Ethereum much greener.
- **Security and Economics:** PoS changes the attack model – an attacker must own a large fraction of ETH (which would be at stake). It also opens up new economic models (earnings from staking yields instead of mining rewards).
- **Scalability Roadmap:** PoS (the Merge) was a prerequisite for future upgrades like sharding. It paves the way for dramatically increased throughput in later phases.
- **Network Effects:** After the Merge, the network's block time remained ~12 seconds but without constant hash rate inflation. The inflation rate of ETH supply also changed (lower, since miner block rewards ended).

In summary, **The Merge** completed Ethereum's shift to Proof of Stake[18], achieving a sustainable consensus model. It maintained continuity of state (no user funds or contracts lost) while slashing energy usage. Going forward, PoS enables faster innovation (e.g. sharding) without needing to change Ethereum's core execution layer again.

Question 5 (10 marks): *Outline the key differences between on-chain, off-chain, and sidechain transactions in Ethereum. Provide use-case examples of each.*

Answer:
- **On-Chain:** These transactions occur **directly on Ethereum** and are recorded in the main blockchain. They

inherit Ethereum's full security and decentralization. Examples include sending ETH from one wallet to another, or calling a smart contract function (like executing a Uniswap trade) that must be confirmed on-chain. On-chain operations are validated by consensus (now PoS). They can include smart contract execution and writing data to the ledger. The main drawback is lower throughput (e.g. 15 tx/sec) and higher gas costs during congestion.

- **Off-Chain:** Off-chain transactions happen **outside of Ethereum's main chain**, with only the final result settled on-chain. For example, two users might open a payment channel: they perform 100 token transfers off-chain, keeping track privately, and only at the end they close the channel by submitting a single on-chain transaction that finalizes the net balances. Another example is a centralized exchange: a user moves tokens off-chain (within the exchange's database) instead of on Ethereum. Off-chain methods are faster and cheaper per transaction, and can be confidential. They **reduce load on Ethereum**, but they introduce trust assumptions (you trust the off-chain mechanism or the entity managing it). Use cases: micropayments (so users don't pay high fees for tiny amounts) and high-frequency trading between known parties.

- **Sidechain:** A sidechain is an **entirely separate blockchain** that runs in parallel to Ethereum but can interoperate via a two-way peg. Tokens or assets are locked on Ethereum and an equivalent amount is issued on the sidechain, and vice versa. Sidechains can have different parameters (e.g. faster blocks, different consensus). They scale by offloading work: for instance, a gaming platform might deploy a sidechain optimized for rapid NFT minting. Transactions on a sidechain (like transferring game items or conducting in-game trades) happen on the sidechain and only the net result or checkpoint may be anchored to Ethereum periodically. This allows much higher throughput. The trade-off is that sidechains have their own security (often fewer validators) and rely on the peg mechanism to enforce asset consistency. Examples: Polygon (an Ethereum sidechain) or xDai, where users can bridge ETH/tokens to the sidechain to use cheaper fees.

Each mode has its trade-offs. On-chain is the most trustless but slow/expensive. Off-chain is very fast/cheap but requires trust in off-chain protocols or intermediaries. Sidechains strike a balance by maintaining a separate blockchain with less congestion but still pegged to Ethereum's value. In practice, Ethereum's ecosystem uses all three: pure on-chain for critical transactions (e.g. large DeFi swaps), off-chain for scaling (like channels), and sidechains for specialized apps (like gaming or private enterprise chains).

Answer Key Citations: All answers above draw on authoritative sources: Ethereum documentation[18], blockchain textbooks [3], and industry articles [4] to ensure accuracy and depth.

---

[1] unit 4.pdf
file://file-GvBMhcaKAFNap86p5xqzX6

[2] [3] [5] [6] [7] [8] [10] [11] Mastering Blockchain_full book.pdf
file://file-ECpnhPTvEeamyqEbvtL7f1

[4] How do Ethereum transactions work?
https://www.alchemy.com/docs/how-ethereum-transactions-work

9 How does Ethereum Virtual Machine (EVM) work? A deep dive into EVM Architecture and Opcodes | QuickNode Guides
https://www.quicknode.com/guides/ethereum-development/smart-contracts/a-dive-into-evm-architecture-and-opcodes

12 13 Non-Fungible Token (NFT): What It Means and How It Works
https://www.investopedia.com/non-fungible-tokens-nft-5115211

14 17 Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example
https://www.investopedia.com/tech/what-dao/

15 16 What is DeFi? | Benefits and Use of Decentralised Finance
https://ethereum.org/en/defi/

18 The Merge
https://ethereum.org/en/roadmap/merge/