**Smart Contracts**

**History**

1. Smart contracts were first theorized by Nick Szabo in the late 1990s, but it was almost 20 years before the true potential and benefits of them were truly appreciated.
2. Smart contracts are described by Szabo as follows:
   * “A smart contract is a computerized transaction protocol that executes the terms of a contract.
   * The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries.
   * Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.”

**Smart Contracts**

1. This idea of smart contracts was implemented in a limited fashion in bitcoin in 2009, where bitcoin transactions can be used to transfer the value between users, over a peer-to-peer network where users do not necessarily trust each other and there is no need for a trusted intermediary.

**Definition**

1. There is no consensus on a standard definition of smart contracts.
2. It is essential to define what a smart contract is, and the following is the author's attempt to provide a generalized definition of a smart contract:
   * A smart contract is a secure and unstoppable computer program representing an agreement that is automatically executable and enforceable.

**Dissecting the Definition**

1. A smart contract is a computer program written in a language that a computer or target machine can understand.
2. It encompasses agreements between parties in the form of business logic.
3. Smart contracts are automatically executed when certain conditions are met.
4. They are enforceable, meaning all contractual terms are executed as defined and expected, even in the presence of adversaries.
5. True smart contracts should not rely on traditional methods of enforcement.
6. They work on the principle that "code is law," meaning no need for an arbitrator or third party to control or influence execution.
7. Smart contracts are self-enforcing as opposed to legally enforceable.

**Key Features**

1. Secure and unstoppable: Smart contracts must be fault tolerant and executable in a reasonable amount of time.
2. They should maintain a healthy internal state, even if external factors are unfavorable.
3. Unlike normal computer programs, smart contracts must be immune to issues caused by deviations in the external environment.

**Additional Notes**

1. Secure and unstoppable may well be considered requirements or desirable features.
2. Including security and unstoppable properties in the definition from the beginning helps build strong foundations for further research.
3. Some researchers suggest smart contracts need not be automatically executable but can be automatable with manual human input.
4. For a contract to be truly smart, it must be fully automated, and inputs from people can be automated using oracles.

**Operational Model**

1. Smart contracts usually operate by managing their internal state using a state machine model.
2. The state of a contract is advanced further based on predefined criteria and conditions.

**Legal and Compliance Issues**

1. There is ongoing debate about whether code can be accepted as a contract in a court of law.
2. Legal questions:
   * How can a smart contract be legally binding?
   * How can dispute resolution be implemented in code?
   * Can regulatory and compliance requirements be addressed effectively?
3. Making smart contract code readable by both machines and humans could make it more acceptable in legal situations.
4. Research is ongoing to address semantics, meaning, and interpretation of contracts.

**Deterministic Nature**

1. Smart contracts are inherently required to be deterministic in nature.
2. Determinism ensures that a smart contract run by any node on a network achieves the same result.
3. Non-deterministic functions can lead to inconsistent results, causing consensus failure.
4. Deterministic programming ensures smart contracts produce the same output for specific inputs.

**Summary of Properties**

1. A smart contract has the following four properties:
   * Automatically executable.
   * Enforceable.
   * Semantically sound.
   * Secure and unstoppable.
2. The first two properties are required as a minimum, whereas the latter two may not be required or implementable in certain scenarios and can be relaxed.

**Ricardian Contracts**

1. **Origin**:
   * Proposed in the *Financial Cryptography in 7 Layers* paper by Ian Grigg in the late 1990s.
   * Initially used in a bond trading and payment system called Ricardo.
2. **Key Idea**:
   * Create a document understandable and acceptable by both a court of law and computer software.
   * Address the challenge of issuing value over the Internet.
   * Identify the issuer and capture all terms and clauses to make the contract legally binding.

**Properties of Ricardian Contracts**

1. A contract offered by an issuer to holders.
2. A valuable right held by holders and managed by the issuer.
3. Easily readable by people (like a paper contract).
4. Readable by programs (parseable, like a database).
5. Digitally signed.
6. Carries the keys and server information.
7. Allied with a unique and secure identifier.

**Implementation Process**

1. A single document contains legal prose and machine-readable tags.
2. The issuer digitally signs the document using their private key.
3. The document is hashed using a message digest function, creating a unique identifier hash.
4. This hash is used in every transaction, linking the contract and the identifier as evidence of intent.
5. The **bowtie model**:
   * The document originates from the World of Law (legal side).
   * The hash is used as an identifier in the World of Accountancy (operational side).
   * A secure link is created between the original written contract and its transactions.

**Comparison with Smart Contracts**

1. **Ricardian Contracts**:
   * Focus on semantic richness and legal prose.
   * Produce a document understandable by humans and partly by programs.
   * Concerned with denotational semantics (real-world meaning of the contract).
2. **Smart Contracts**:
   * Focus on the execution of the contract.
   * Do not include contractual documents.
   * Concerned with operational semantics (execution, correctness, and safety).

**Categorization**

1. Some researchers differentiate between:
   * **Smart Contract Code**: Execution-oriented contracts.
   * **Smart Legal Contracts**: Include both denotational and operational semantics.
2. Ideal smart contracts should encode legal prose and business logic.

**Tuple Representation of Ricardian Contracts**

1. Prose: The legal contract written in regular language.
2. Parameters: Connect parts of the legal contract to equivalent code.
3. Code: Computer-understandable representation of legal prose.

**Systems Using Ricardian Contracts**

1. CommonAccord
2. OpenBazaar
3. OpenAssets
4. Askemos

**Government**

* There are various applications of blockchain being researched currently that can support government functions and take the current model of e-government to the next level.
* First, some background for e-government will be provided, and then a few use cases such as e-voting, homeland security (border control), and electronic IDs (citizen ID cards) will be discussed.
* E-government or electronic government is a paradigm where information and communication technology is used to deliver public services to citizens.
* The concept is not new and has been implemented in various countries around the world.
* With blockchain, a new avenue of exploration has opened up.
* Many governments are researching the possibility of using blockchain technology for managing and delivering public services.
* Transparency, auditability, and integrity are attributes of blockchain that can go a long way in effectively managing various government functions.

**Border Control**

* Automated border control systems help prevent illegal entry, terrorism, and human trafficking.
* Machine-readable travel documents (MRTDs) and biometric passports have enabled automated border control.
* MRTD standards (ICAO 9303) include security features like:
  + Biometric scans (retina, fingerprints, facial recognition).
  + Machine Readable Zone (MRZ) and visible text attributes.
* **Issues with current systems**:
  + Centralized control limits data sharing between law enforcement agencies.
  + No immediate mechanism for blacklisting/revoking passports.
* **Blockchain solutions**:
  + Smart contracts maintain and update blacklists instantly, visible to all agencies.
  + Simplifies systems, reduces costs, and ensures immutability.
  + Backend distributed databases (e.g., BigChainDB, IPFS) store large datasets.
  + Hashes of travel documents and biometric IDs are stored on the blockchain.
  + Enables anti-terrorism efforts through secure, auditable data sharing.
* A high-level system uses RFID scanners and biometric verification to check legitimacy, integrating blockchain for verification and sharing results instantly.

**Voting**

* Voting systems face issues like fraud, process weaknesses, and lack of transparency.
* Blockchain-based voting offers:
  + End-to-end security through public key cryptography.
  + Immutability to prevent vote tampering or double voting.
  + Privacy via zero-knowledge proofs.
* Smart contracts can manage biometric IDs to ensure only valid votes are cast.
* Transparent systems rebuild public trust in elections.

**Citizen Identification (ID Cards)**

* Current national ID cards are secure but can be improved with blockchain:
  + Digital identity on blockchain provides control over data sharing.
  + Citizens can see and control how their information is used.
* A single ID can be used for multiple government services (e.g., pensions, taxation).
* Blockchain ensures immutable, transparent records tied to digital IDs.
* Applications include notarizing documents like birth certificates, marriages, and deeds.
* Challenges:
  + Immutability conflicts with laws like the "right to be forgotten."
  + Current blockchain technology is still immature for full implementation.

**Miscellaneous Government Applications**

* **Areas of potential improvement**:
  + Tax collection, benefits disbursement, land ownership records, and motor vehicle registration.
* Blockchain benefits include immutability, transparency, and decentralization.

**Health**

* Blockchain addresses major healthcare challenges like:
  + Privacy breaches, data security, and counterfeit medicines.
* Benefits include cost savings, faster claims processing, and improved trust.
* Blockchain systems enable high availability and prevent operational errors.
* Examples:
  + FoldingCoin and CureCoin incentivize miners to solve scientific problems like disease research.

**Finance**

* Widely researched in the finance sector due to cost-saving potential.
* **Applications**:
  + **Insurance**:
    - Stops fraudulent claims and speeds up processing.
    - IoT devices and smart contracts automate claims based on real-time data.
    - Example: Dynamis offers Ethereum-based peer-to-peer insurance.
  + **Post-Trade Settlement**:
    - Blockchain simplifies trade lifecycle steps (execution, clearing, and settlement).
    - Reduces dependency on central clearing houses and intermediaries.
    - Enables immediate peer-to-peer settlement, cutting costs and risks.
  + **Financial Crime Prevention**:
    - Distributed ledgers ensure accurate, shared KYC/AML data across institutions.
    - Regulatory reporting becomes more transparent and cost-efficient.

**Media**

* **Key challenges**:
  + Content distribution, rights management, and royalty payments.
  + Illegal copying and lack of payment transparency.
* Blockchain ensures:
  + Ownership of digital content via cryptographic guarantees.
  + Automated payments through smart contracts.
  + Transparent tracking of ownership and royalties.
* Enables secure transfer and management of digital copyrights.

**Introduction to Blockchain Beyond Currencies**

* Blockchain originated with Bitcoin as a digital currency but gained broader recognition with Blockchain 2.0 in 2013, introducing its use across multiple industries.
* Applications extend to **IoT, government, healthcare, finance, digital rights management, and law**.
* This chapter focuses on **IoT, government, health, and finance**, exploring their potential and associated use cases.

**Internet of Things (IoT):**

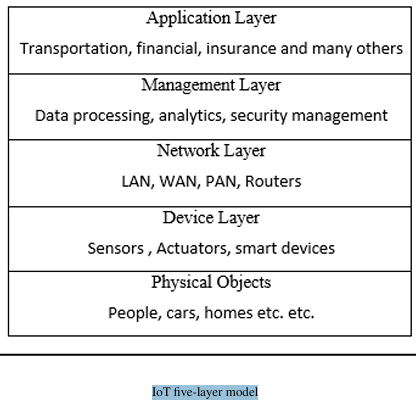
IoT refers to interconnected physical objects that sense, react, collect, and communicate data via the internet.

**Core Functions of IoT:**

1. **Sensing:** Performed by sensors.
2. **Reacting/Controlling:** Managed by actuators.
3. **Collecting Data:** Done through various sensors.
4. **Communicating:** Enabled by network components (e.g., chips).

**IoT Ecosystem Layers:**

1. **Physical Object Layer:**
   * Includes tangible items like vehicles, homes, and factories.
2. **Device Layer:**
   * Contains sensors (temperature, humidity, etc.) and actuators for environmental control.
3. **Network Layer:**
   * Ensures connectivity using technologies like Bluetooth, WiFi, and ZigBee.
4. **Management Layer:**
   * Handles data flow, security, and analytics.
5. **Application Layer:**
   * Features use cases like healthcare, supply chain management, and smart homes.



**Challenges with Centralized IoT Models:**

* Centralized cloud systems face issues like:
  + **Scalability limits** with billions of devices.
  + **Security risks**, including hacking and data theft.
  + **High costs** for infrastructure and maintenance.

**Blockchain Integration in IoT:**

A blockchain-based IoT model decentralizes control, enhancing security, scalability, and autonomy.

**Benefits of Blockchain in IoT:**

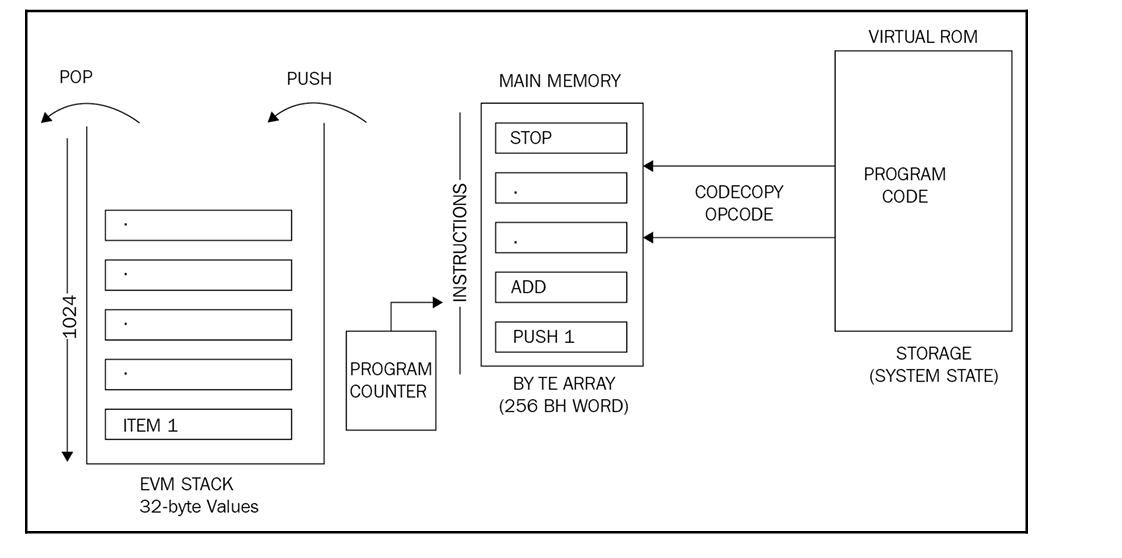
1. **Decentralization:**
   * Removes single points of failure and enables peer-to-peer communication.
2. **Cost Efficiency:**
   * Eliminates the need for centralized data centers.
3. **Security:**
   * Prevents denial-of-service attacks through a distributed structure.
4. **Autonomy:**
   * Smart contracts allow devices to transact and communicate directly (e.g., self-renting hotel rooms, automated online purchases).

**Blockchain-Based IoT Model:**

* Introduces a **blockchain layer** above the network layer.
* Handles smart contracts, privacy, and secure communication.
* Reduces reliance on centralized management for data processing and storage.

**Future Implications and Examples:**

* By 2020, it was estimated that 22 billion devices would be connected to the internet, necessitating scalable and decentralized solutions.
* Projects like **IBM Bluemix** and startups like **Filament** are exploring blockchain-enabled IoT ecosystems.
* **Case Example:** Connecting an IoT device to the Ethereum blockchain to demonstrate smart contract-enabled functionality (e.g., unlocking a door via blockchain transactions).



**Ethereum virtual machine (EVM)**  
EVM is a simple stack-based execution machine that runs bytecode instructions in order to transform the system state from one state to another. The word size of the virtual machine is set to 256-bit. The stack size is limited to 1024 elements and is based on the LIFO (Last in First Out) queue. EVM is a Turing-complete machine but is limited by the amount of gas that is required to run any instruction. This means that infinite loops that can result in denial of service attacks are not possible due to gas requirements. EVM also supports exception handling in case exceptions occur, such as not having enough gas or invalid instructions, in which case the machine would immediately halt and return the error to the executing agent.

EVM is a fully isolated and sandboxed runtime environment. The code that runs on the EVM does not have access to any external resources, such as a network or filesystem.

[ 222 ]  
*Ethereum 101*

As discussed earlier, EVM is a stack-based architecture. EVM is big-endian by design and it uses 256-bit wide words. This word size allows for Keccak 256-bit hash and elliptic curve cryptography computations.

There are two types of storage available to contracts and EVM. The first one is called memory, which is a byte array. When a contract finishes the code execution, the memory is cleared. It is akin to the concept of RAM. The other type, called storage, is permanently stored on the blockchain. It is a key value store.

Memory is unlimited but constrained by gas fee requirements. The storage associated with the virtual machine is a word addressable word array that is non-volatile and is maintained as part of the system state. Keys and value are 32 bytes in size and storage. The program code is stored in a virtual read-only memory (virtual ROM) that is accessible using the CODECOPY instruction. The CODECOPY instruction is used to copy the program code into the main memory. Initially, all storage and memory is set to zero in the EVM.

The following diagram shows the design of the EVM where the virtual ROM stores the program code that is copied into main memory using CODECOPY. The main memory is then read by the EVM by referring to the program counter and executes instructions step by step. The program counter and EVM stack are updated accordingly with each instruction execution.

**EVM operation**  
[ 223 ]  
*Ethereum 101*

EVM optimization is an active area of research and recent research has suggested that EVM can be optimized and tuned to a very fine degree in order to achieve high performance. Research into the possibility of using Web assembly (WASM) is underway already. WASM is developed by Google, Mozilla, and Microsoft and is now being designed as an open standard by the W3C community group. The aim of WASM is to be able to run machine code in the browser that will result in execution at native speed. Similarly, the aim of EVM 2.0 is to be able to run the EVM instruction set (Opcodes) natively in CPUs, thus making it faster and efficient.

**Elements of the Ethereum Blockchain**

1. **Ethereum Blockchain**:
   * A decentralized ledger that records transactions and state changes across the network.
2. **Accounts**:
   * Two types:
     + **Externally Owned Accounts (EOAs)**: Controlled by private keys, used for sending transactions.
     + **Contract Accounts (CAs)**: Associated with smart contracts, triggered by EOAs or other contracts.
3. **Blocks**:
   * Contain:
     + A block header with metadata (e.g., parent hash, state root).
     + A list of transactions.
     + A list of uncle blocks (stale blocks).
4. **State**:
   * Represents the current state of all accounts and balances.
   * Stored as a **Merkle Patricia Trie**.
5. **Transactions**:
   * Instructions to change the state.
   * Two types:
     + **Message Calls**: Transfers ETH or data.
     + **Contract Creations**: Deploy new smart contracts.
6. **Ethereum Virtual Machine (EVM)**:
   * Executes smart contracts and ensures state transitions.
7. **Gas**:
   * A unit to measure and pay for computational resources during transactions.
8. **Consensus Mechanism**:
   * **Proof of Work (PoW)** (currently being replaced by **Proof of Stake**).
   * Ensures blocks are added securely to the blockchain.

**EVM: Stack-Based Architecture**

The Ethereum Virtual Machine (EVM) operates on a stack-based architecture, designed to execute smart contracts efficiently and securely. Below are the key aspects of its stack-based design:

**1. Core Components**

* **Stack**:
  + EVM uses a **Last-In-First-Out (LIFO)** stack to store intermediate results during computation.
  + **Maximum size**: 1024 elements.
  + Each element: **256 bits** (32 bytes), optimized for cryptographic operations like **Keccak-256 hashing**.
* **Memory**:
  + Temporary, byte-addressable storage cleared after execution.
  + Used for dynamic data handling during contract execution.
  + Gas cost increases with memory usage.
* **Storage**:
  + Persistent, key-value storage associated with smart contracts.
  + Stored on the blockchain; changes incur higher gas costs compared to memory.
* **Program Counter (PC)**:
  + Tracks the current position in the execution of bytecode instructions.

**2. Execution Flow**

* EVM executes **bytecode instructions sequentially**, manipulating the stack at each step.
* Operations are performed using opcodes like **PUSH, POP, ADD, MUL**, etc.
* The stack handles intermediate values, while memory and storage manage more extensive or persistent data.

**3. Gas Management**

* **Gas** limits execution and prevents infinite loops.
* Each opcode consumes a **predefined amount of gas** based on its complexity (e.g., ADD is cheaper than SHA3).
* **Unused gas** is refunded to the sender, but exceeding the gas limit causes the transaction to revert.

**4. Instruction Set**

* **Arithmetic Operations**: **ADD, MUL, SUB**, etc., work directly on stack elements.
* **Data Handling**:
  + **PUSH**: Adds data to the stack.
  + **POP**: Removes the top element.
  + **DUP**: Duplicates a stack item.
  + **SWAP**: Swaps the position of two stack elements.
* **Flow Control**: **JUMP** and **JUMPI** modify the program counter based on conditions.

**5. Benefits of Stack-Based Design**

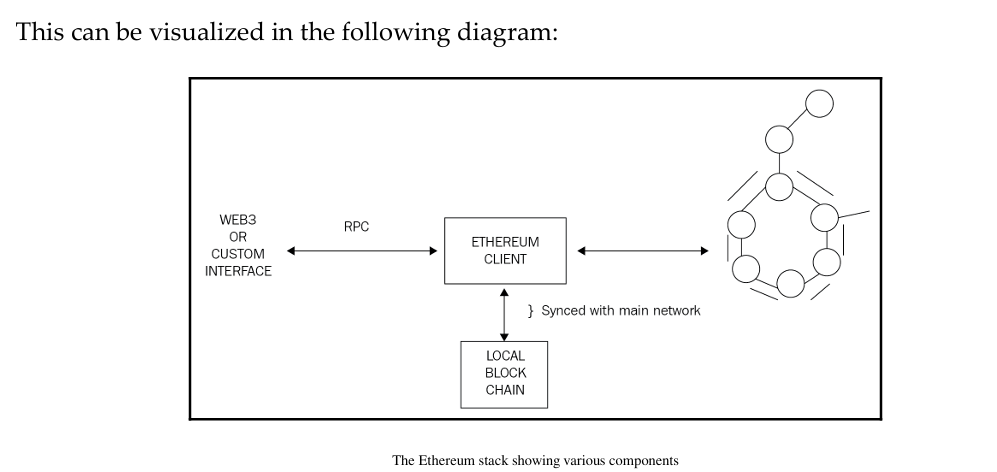
* **Simplicity**: Minimalist design makes the EVM lightweight and deterministic.
* **Efficiency**: Optimized for cryptographic operations and smart contract execution.
* **Isolation**: Ensures the sandboxed execution of contracts without affecting external systems.

**The Ethereum Stack**

The Ethereum stack consists of various components. At the core, there is the **Ethereum blockchain** running on the **P2P Ethereum network**.

1. **Ethereum Client**:
   * Usually **Geth** (Go Ethereum).
   * Runs on the nodes and connects to the peer-to-peer Ethereum network, from where the blockchain is **downloaded and stored locally**.
   * Provides various functions, such as **mining** and **account management**.
   * The local copy of the blockchain is **synchronized regularly** with the network.
2. **Web3.js Library**:
   * Allows interaction with **geth** via the **Remote Procedure Call (RPC)** interface.

The Ethereum stack consists of various components. At the core, there is the Ethereum blockchain running on the P2P Ethereum network. Secondly, there's an Ethereum client (usually geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally. It provides various functions, such as mining and account management. The local copy of the blockchain is synchronized regularly with the network. Another component is the web3.js library that allows interaction with geth via the Remote Procedure Call (RPC) interface.



**EVM: Stack-Based Architecture**

The Ethereum Virtual Machine (EVM) operates on a stack-based architecture, designed to execute smart contracts efficiently and securely. Below are the key aspects of its stack-based design:

**1. Core Components**

* **Stack**:
  + EVM uses a **Last-In-First-Out (LIFO)** stack to store intermediate results during computation.
  + **Maximum size**: 1024 elements.
  + Each element: **256 bits** (32 bytes), optimized for cryptographic operations like **Keccak-256 hashing**.
* **Memory**:
  + Temporary, byte-addressable storage cleared after execution.
  + Used for dynamic data handling during contract execution.
  + Gas cost increases with memory usage.
* **Storage**:
  + Persistent, key-value storage associated with smart contracts.
  + Stored on the blockchain; changes incur higher gas costs compared to memory.
* **Program Counter (PC)**:
  + Tracks the current position in the execution of bytecode instructions.

**2. Execution Flow**

* EVM executes **bytecode instructions sequentially**, manipulating the stack at each step.
* Operations are performed using opcodes like **PUSH, POP, ADD, MUL**, etc.
* The stack handles intermediate values, while memory and storage manage more extensive or persistent data.

**3. Gas Management**

* **Gas** limits execution and prevents infinite loops.
* Each opcode consumes a **predefined amount of gas** based on its complexity (e.g., ADD is cheaper than SHA3).
* **Unused gas** is refunded to the sender, but exceeding the gas limit causes the transaction to revert.

**4. Instruction Set**

* **Arithmetic Operations**: **ADD, MUL, SUB**, etc., work directly on stack elements.
* **Data Handling**:
  + **PUSH**: Adds data to the stack.
  + **POP**: Removes the top element.
  + **DUP**: Duplicates a stack item.
  + **SWAP**: Swaps the position of two stack elements.
* **Flow Control**: **JUMP** and **JUMPI** modify the program counter based on conditions.

**5. Benefits of Stack-Based Design**

* **Simplicity**: Minimalist design makes the EVM lightweight and deterministic.
* **Efficiency**: Optimized for cryptographic operations and smart contract execution.
* **Isolation**: Ensures the sandboxed execution of contracts without affecting external systems.