L13: Ethereum Architecture

Module B: Ethereum & Smart Contracts

Blockchain & Cryptocurrency Course

December 2025

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the Ethereum Virtual Machine (EVM) architecture
- Distinguish between Externally Owned Accounts (EOA) and Contract Accounts
- Describe Ethereum as a state machine and understand the world state concept
- Explain the Merkle Patricia Trie data structure
- Compare Ethereum's account model to Bitcoin's UTXO model

What is Ethereum?

Key Characteristics:

- Decentralized computation platform
- Turing-complete blockchain
- Smart contract support
- Native cryptocurrency: Ether (ETH)
- Launched July 30, 2015

Beyond Bitcoin:

- Bitcoin: Digital gold, value transfer
- Ethereum: World computer, programmable logic
- Enables decentralized applications (dApps)
- Supports complex financial instruments
- Foundation for DeFi, NFTs, DAOs

Ethereum Virtual Machine (EVM)

The EVM is a quasi-Turing complete state machine:

- Stack-based virtual machine (256-bit word size)
- Executes bytecode compiled from high-level languages (Solidity, Vyper)
- Deterministic execution: same input always produces same output
- Gas metering prevents infinite loops (making it "quasi"-Turing complete)
- Isolated environment: no access to network, filesystem, or other processes

EVM Operations:

- Arithmetic: ADD, MUL, SUB, DIV, MOD, EXP
- Comparison: LT, GT, EQ
- Bitwise: AND, OR, XOR, NOT
- Storage: SLOAD, SSTORE
- Context: CALLER, CALLVALUE, ADDRESS

Account Types: EOA vs Contract

Externally Owned Account (EOA):

- Controlled by private key
- Has an address (derived from public key)
- Can hold Ether balance
- Can send transactions
- No code storage
- Creation is free

EOA Address:

- Last 20 bytes of Keccak-256 hash of public key
- Example: 0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed

Contract Account:

- Controlled by smart contract code
- Has an address (deterministically derived)
- Can hold Ether balance
- Cannot initiate transactions
- Stores code and state
- Creation costs gas

Contract Address:

- Derived from creator address + nonce
- Example: 0xd4e56740f876aef8c010b86a40d5f5675c82f28

Account State Components

Every account (EOA or Contract) has four fields:

1. **Nonce:**
   - EOA: Counter of transactions sent from this address
   - Contract: Counter of contracts created by this contract
   - Prevents replay attacks
2. **Balance:**
   - Amount of Wei (10^{-18} ETH) owned by this address
   - 1 ETH = 1,000,000,000,000,000,000 Wei
3. **StorageRoot:**
   - Hash of the root node of the account's storage trie
   - Empty for EOAs (0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421)
4. **CodeHash:**
   - Hash of the EVM bytecode for this account
   - Empty for EOAs (hash of empty string)
   - Immutable once set for contracts

Ethereum as a State Machine

State Transition Function:

- Ethereum transitions from state S_t to state S_{t+1} via valid transactions
- $S_{t+1} = Y(S_t, T)$ where T is a transaction
- State includes all account balances, storage, and code
- Deterministic: Given state S_t and transaction T, S_{t+1} is uniquely determined

Transaction Execution:

1. Validate transaction (signature, nonce, gas limit, balance)
2. Deduct gas cost from sender
3. Execute transaction (transfer ETH, execute code)
4. Update state (balances, storage, nonces)
5. Refund unused gas
6. Distribute transaction fees to miner/validator

World State

The World State is a mapping between addresses and account states:

- Maps 160-bit addresses to account states
- Stored as a Merkle Patricia Trie
- Root hash included in every block header
- Allows efficient verification of account state
- Enables light clients to verify data without full state

State Root:

- 256-bit hash representing entire world state
- Changes with every block
- Uniquely identifies state at a specific block height
- Example: 0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544

Merkle Patricia Trie (MPT)

Combines three data structures:

1. **Merkle Tree:** Cryptographic verification via hashes
2. **Patricia Trie:** Efficient key-value storage with shared prefixes
3. **Radix Trie:** Optimized path compression

MPT Properties:

- Deterministic: Same key-value pairs always produce same root hash
- Efficient verification: Prove inclusion with $O(\log n)$ proof size
- Efficient updates: Only modified paths need rehashing
- Path compression: Reduces storage for sparse trees

Node Types:

- **Branch Node:** 16 children (hex digits 0-F) + optional value
- **Extension Node:** Shared path prefix + pointer to next node
- **Leaf Node:** Remaining key path + value

MPT Example: World State

Simplified example with 3 accounts:

- Address A: 0xa7f9... Balance: 100 ETH, Nonce: 5
- Address B: 0xa8e3... Balance: 50 ETH, Nonce: 2
- Address C: 0xb2c1... Balance: 200 ETH, Nonce: 10

Trie Structure:

1. Root is a branch node with children at positions 0xa and 0xb
2. Path 0xa has extension node with shared prefix "a"
3. Extension branches to two leaves: 0xa7f9... and 0xa8e3...
4. Path 0xb has leaf node at 0xb2c1...
5. Each leaf stores RLP-encoded account state (balance, nonce, storage root, code hash)

Root Hash: Keccak-256 of root node uniquely identifies this state

MPT in Ethereum: Four Tries

Every block references four MPT roots:

1. **State Trie:**
   - Maps address → account state
   - Global, updated with every block
2. **Storage Trie:**
   - Maps storage key → value for each contract
   - One per contract account
3. **Transaction Trie:**
   - Maps transaction index → transaction in current block
   - One per block
4. **Receipt Trie:**
   - Maps transaction index → receipt (logs, gas used, status)
   - One per block

Ethereum vs Bitcoin: Account Model vs UTXO

Bitcoin UTXO Model:

- Unspent Transaction Outputs
- No accounts, only UTXOs
- Transaction consumes inputs, creates outputs
- Stateless verification
- Parallel transaction validation
- Simpler, more secure
- Limited programmability

Example:

- Alice has UTXO of 5 BTC
- Sends 2 BTC to Bob
- Creates: 2 BTC to Bob, 3 BTC back to Alice
- Original 5 BTC UTXO destroyed

Ethereum Account Model:

- Persistent account balances
- State stored globally
- Transaction modifies balances
- Stateful execution
- Sequential nonce ordering
- More complex
- Full programmability

Example:

- Alice has account with 5 ETH
- Sends 2 ETH to Bob
- Alice balance: 5 - 2 = 3 ETH
- Bob balance: +2 ETH
- Accounts persist

Account Model Advantages

Why Ethereum chose accounts over UTXO:

1. **Simplicity:**
   - Intuitive balance model
   - Easier to reason about state
   - Simpler wallet implementation
2. **Space Efficiency:**
   - No need to track unspent outputs
   - Smaller state size for complex contracts
3. **Fungibility:**
   - All Ether is equivalent
   - No dust accumulation
4. **Smart Contract Design:**
   - Natural fit for persistent contract storage
   - Easier to implement tokens and complex logic
   - Direct interaction with contract state

Account Model Challenges

Tradeoffs of the account model:

1. **Replay Attack Prevention:**
   - Requires nonce tracking
   - Sequential transaction ordering per account
   - Cannot parallelize transactions from same account
2. **State Growth:**
   - All accounts stored indefinitely
   - State size grows continuously
   - Higher storage requirements for full nodes
3. **Privacy:**
   - All account activity linked to single address
   - Harder to implement UTXO-style mixing
   - Requires explicit privacy techniques (e.g., Tornado Cash)

Block Structure

Ethereum block header contains:

- **parentHash:** Hash of parent block
- **ommersHash:** Hash of uncle blocks (pre-merge)
- **beneficiary:** Address receiving block reward
- **stateRoot:** Root hash of state trie
- **transactionsRoot:** Root hash of transaction trie
- **receiptsRoot:** Root hash of receipt trie
- **logsBloom:** Bloom filter for logs (efficient event lookup)
- **difficulty:** Proof-of-work difficulty (pre-merge, now 0)
- **number:** Block height
- **gasLimit:** Maximum gas allowed in this block
- **gasUsed:** Total gas used by all transactions
- **timestamp:** Unix timestamp of block creation
- **extraData:** Arbitrary data (max 32 bytes)
- **mixHash, nonce:** PoW validation (pre-merge)

2024 Milestone: Dencun Upgrade (March 13, 2024)

Major Network Upgrade (Proto-Danksharding):

- **EIP-4844**: Introduces "blob" transactions for Layer 2 data
- **Problem Solved**: L2 rollups pay expensive calldata for data availability
- **Solution**: New data type (blobs) with separate, cheaper gas market

How Blobs Work:

- Blobs: 128 KB data chunks, stored for 18 days (not permanent)
- Separate "blob gas" market (independent of execution gas)
- L2s post transaction batches as blobs instead of calldata
- Data availability guaranteed, but not permanently stored

Impact:

- L2 transaction fees reduced by 90%+ (from \$0.50 to \$0.01)
- Arbitrum, Optimism, Base, zkSync all adopted immediately
- Paves way for full Danksharding (future upgrade)
- Ethereum becomes true settlement/DA layer

Key Takeaways

1. **EVM:** Deterministic, stack-based virtual machine enabling Turing-complete smart contracts
2. **Two Account Types:** EOAs (user-controlled) and Contract Accounts (code-controlled)
3. **State Machine:** Ethereum transitions between states via transactions, with deterministic outcomes
4. **World State:** Mapping of all addresses to account states, stored as Merkle Patricia Trie
5. **MPT:** Efficient cryptographic data structure enabling state verification and light clients
6. **Account Model:** Simpler and more suitable for smart contracts than Bitcoin's UTXO model, but with tradeoffs in privacy and parallelization

Discussion Questions

1. Why is the EVM only "quasi"-Turing complete rather than fully Turing complete?
2. What security benefits does the nonce provide for both EOAs and contracts?
3. How does the Merkle Patricia Trie enable light clients to verify account balances without downloading the full state?
4. In what scenarios might Bitcoin's UTXO model be preferable to Ethereum's account model?
5. How does Ethereum's state growth challenge affect node operators, and what solutions are being explored?

Next Lesson Preview: L14 - Gas Mechanics

Coming up next:

- Understanding gas as a computational unit
- Gas price, gas limit, and transaction cost calculation
- EIP-1559: Base fee and priority fee mechanism
- Gas costs for different EVM operations
- Optimization techniques for reducing gas consumption
- Real-world examples of gas-inefficient code

Preparation:

- Review basic Ethereum transaction structure
- Familiarize yourself with Wei/Gwei/ETH units
- Optional: Browse Etherscan to see gas usage in real transactions